Севастопольский государственный университет
Кафедра «Информационные системы»

Управление данными

курс лекций

лектор:
ст. преподаватель кафедры ИС Абрамович А.Ю.

Лекция 8

Распределенные данные. Репликация. Шардинг

ПРИЧИНЫ РАСПРЕДЕЛЕНИЯ БАЗЫ ДАННЫХ ПО НЕСКОЛЬКИМ МАШИНАМ

МАСШТАБИРУЕМОСТЬ

Если объем данных, нагрузка по чтению или записи перерастают возможности одной машины, то можно распределить эту нагрузку на несколько компьютеров

ОТКАЗОУСТОЙЧИВОСТЬ/ ВЫСОКАЯ ДОСТУПНОСТЬ

Если приложение должно продолжать работать даже в случае сбоя одной из машин, то можно использовать избыточные компьютеры. При отказе одного из них выполнение задач делегируется другому

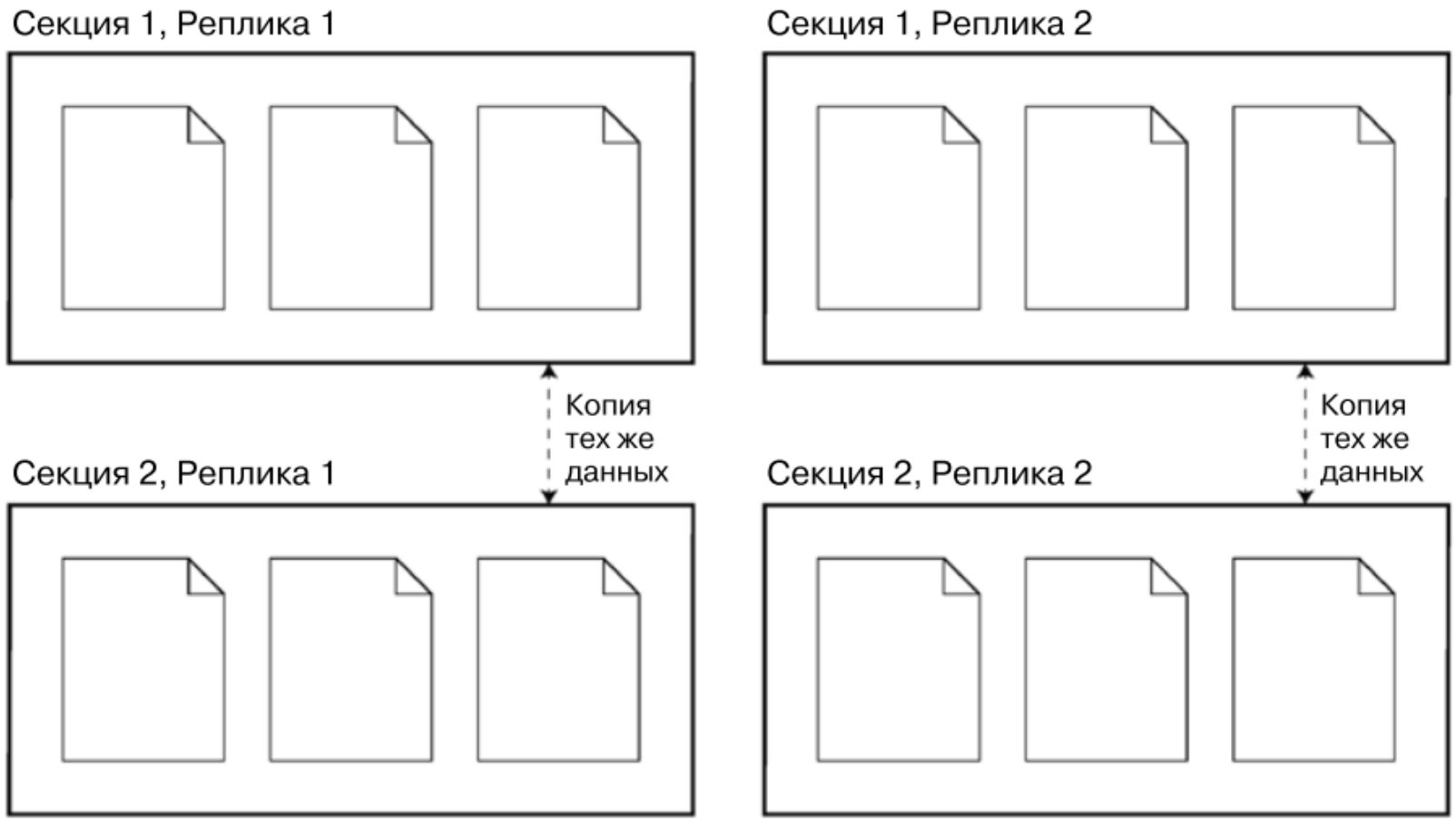
ЗАДЕРЖКА

При наличии пользователей по всему миру необходимы серверы в разных точках земного шара. При этом пользователям не нужно будет ждать, пока сетевые пакеты обойдут половину земного шара

Вертикальное масштабирование (vertical scaling, scaling up): объединяет много процессоров, чипов памяти и жестких дисков **под управлением одной операционной системы**, а быстрые соединения между ними позволят любому из процессоров обращаться к любой части памяти или диска. В подобной **архитектуре с разделяемой памятью** (shared-memory architecture) можно рассматривать все компоненты как единую машину.

Архитектуры без разделения ресурсов (shared-nothing architectures), известные под названием **горизонтального масштабирования** (horizontal scaling, scaling out) подразумевают, что каждый компьютер или виртуальная машина, на которой работает база данных, называется узлом (node). Все узлы используют ресурсы независимо друг от друга. Согласование узлов выполняется на уровне программного обеспечения с помощью обычной сети.

БАЗА ДАННЫХ, РАЗБИТАЯ НА ДВЕ СЕКЦИИ, ПО ДВЕ РЕПЛИКИ НА СЕКЦИЮ



Репликация — это синхронное или асинхронное копирование данных между несколькими серверами.

Шардинг — прием, который позволяет распределять данные между разными физическими серверами.

РЕПЛИКАЦИЯ

Возможные причины репликации данных:

- ради хранения данных географически близко к пользователям (сокращения, таким образом, задержек);
- чтобы система могла продолжать работать при отказе некоторых ее частей (повышения, таким образом, доступности);
- для горизонтального масштабирования количества машин, обслуживающих запросы на чтение (и повышения, таким образом, пропускной способности по чтению).

Репликация — одна из техник масштабирования баз данных. Состоит эта техника в том, что **данные с одного сервера базы данных постоянно копируются (реплицируются) на один или несколько других (называемые репликами).** Для приложения появляется возможность **использовать не один сервер** для обработки всех запросов, а **несколько**. Таким образом появляется возможность распределить нагрузку с одного сервера на несколько.

Узлы, в которых хранятся копии БД, называются **репликами.**

ВЕДУЩИЙ (MASTER, PRIMARY) УЗЕЛ
используется для изменения данных

ВЕДОМЫЙ (FOLLOWERS, READ REPLICAS, SLAVES, SECONDARIES, HOT STANDBYS) УЗЕЛ
используется для чтения данных

Проблемы, решаемые репликацией

Распространение данных. Обычно репликация потребляет не очень большую часть пропускной способности сети, к тому же ее можно в любой момент остановить и затем возобновить.

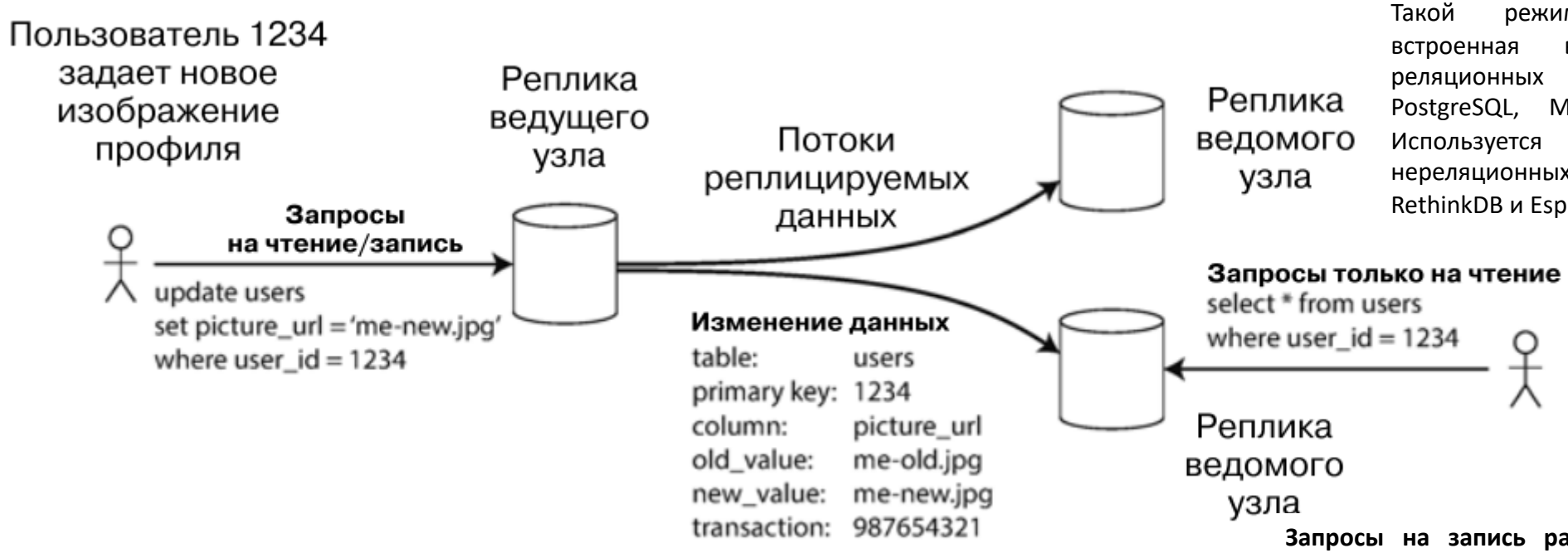
Балансировка нагрузки. С помощью репликации можно распределить запросы на чтение между несколькими серверами; в приложениях с интенсивным чтением эта тактика работает очень хорошо.

Резервное копирование. Подчиненный сервер не может использоваться в качестве резервной копии и не является заменой настоящему резервному копированию.

Высокая доступность и аварийное переключение на резервный сервер(failover). Хорошая сис тема аварийного переключения при отказе, имеющая в составе реплицированные подчиненные серверы, способна существенно сократить время простоя.

Каждая операция записи в базу должна учитываться каждой репликой, иначе нельзя гарантировать, что реплики содержат одни и те же данные. Наиболее распространенное решение этой проблемы называется **репликацией с ведущим узлом (leader-based replication)**.

«репликации типа “активный/пассивный”» (active/passive replication) или «репликации типа “главный-подчиненный”» (master-slave replication)



Такой режим репликации — встроенная возможность многих реляционных баз данных, например PostgreSQL, MySQL, Oracle и т.д. Используется и в некоторых нереляционных БД: MongoDB, RethinkDB и Espresso

Запросы на запись разрешено отправлять только ведущему (ведомые с точки зрения клиента предназначены только для чтения).

Клиенты, желающие данные в базу, должнь свои запросы ведуш который сначала записы данные в свое хранилище.

Когда ведущий узел записывает в свое хранилище новые данные, он также отправляет информацию об изменениях данных всем ведомым узлам в качестве части журнала репликации (replication log) или потока изменений (change stream).

узлы получают журнал его и обновляют щим образом свою опию БД, применяя все аписи в порядке их едущим узлом

Когда клиенту требуется прочитать данные из базы, он может выполнить запрос или к ведущему узлу, или к любому из ведомых.

СИНХРОННАЯ И АСИНХРОННАЯ РЕПЛИКАЦИЯ

В случае **синхронной репликации**, если данная реплика обновляется, все другие реплики того же фрагмента данных также должны быть обновлены в одной и той же транзакции. Логически это означает, что **существует лишь одна версия данных**.

Преимущество синхронной репликации: **копия данных на ведомом узле гарантированно актуальна и согласуется с ведущим узлом.** В случае внезапного сбоя последнего можно быть уверенным, что данные по-прежнему доступны на ведомом узле. **Недостаток** состоит в следующем: **если синхронный ведомый узел не отвечает** (из-за его сбоя, или сбоя сети, или по любой другой причине), то **операцию записи завершить не удастся**. Ведущему узлу придется блокировать все операции записи и ждать до тех пор, пока синхронная реплика не станет снова доступна.

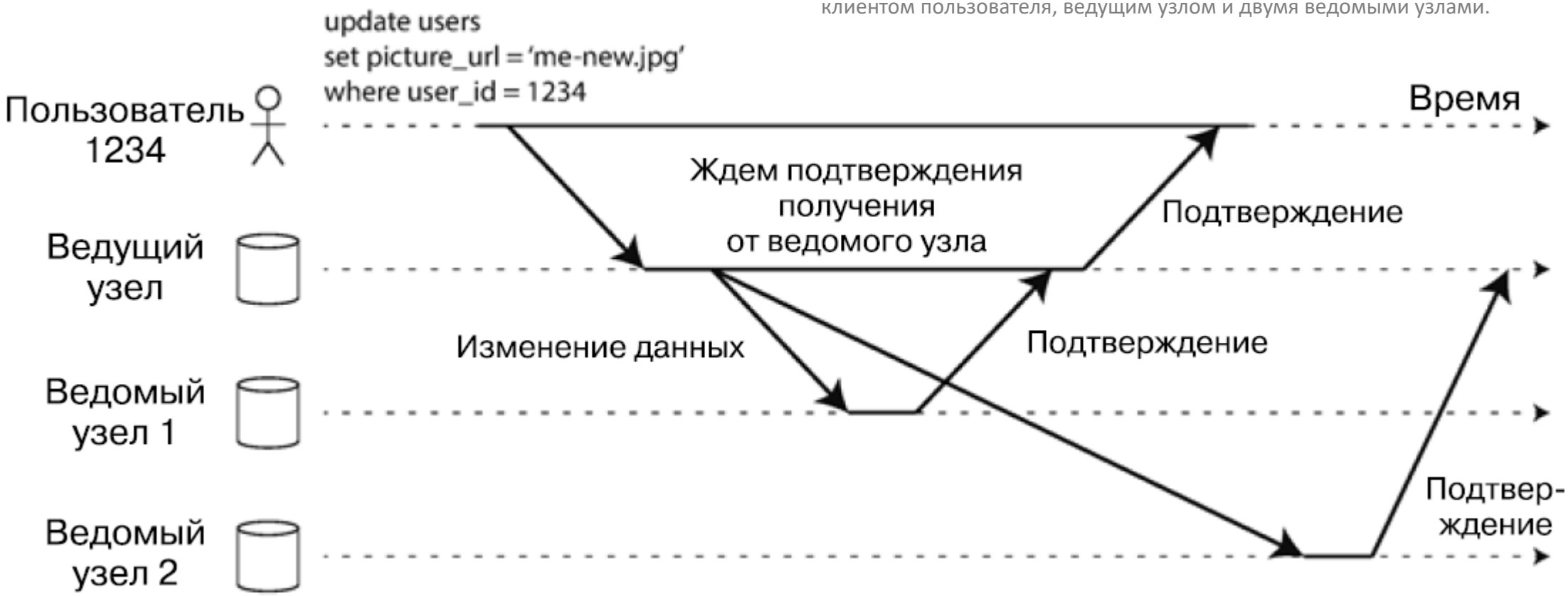
На практике активизация в СУБД синхронной репликации означает, что один из ведомых узлов — синхронный, а остальные — асинхронны. **В случае замедления или недоступности синхронного ведомого узла в него превращается один из асинхронных ведомых узлов.** Такая конфигурация иногда называется ***полусинхронной.***

В случае **асинхронной репликации** обновление одной реплики распространяется на другие **спустя некоторое время**, а не в той же транзакции. Таким образом, при **асинхронной репликации вводится задержка, или время ожидания**, в течение которого отдельные реплики могут быть фактически неидентичными.

Преимущество асинхронной репликации состоит в том, что дополнительные издержки репликации не связаны с транзакциями обновлений, которые могут иметь важное значение для функционирования всего предприятия и предъявлять высокие требования к производительности. **К недостаткам** относится то, что данные могут оказаться несовместимыми (с точки зрения пользователя). **Избыточность может проявляться на логическом уровне.**

Репликация с ведущим узлом делается полностью асинхронной. В этом случае **при фатальном сбое ведущего узла** все еще **не реплицированные** на ведомые узлы **операции записи теряются.** Сохраняемость записи не гарантируется, даже если клиент получил ее подтверждение. Однако **преимуществом полностью асинхронной конфигурации** является возможность ведущего узла выполнять операции записи даже в случае запаздывания всех ведомых.

На рисунке показаны взаимодействия между разными компонентами системы: клиентом пользователя, ведущим узлом и двумя ведомыми узлами.



Репликация на ведомый узел 1 синхронна: ведущий узел ждет до тех пор, пока ведомый узел 1 не подтвердит получение операции записи, прежде чем сообщить пользователю об успехе и сделать результаты записи видимыми другим клиентам. **Репликация на ведомый узел 2 асинхронна**: ведущий узел отправляет сообщение, но не ждет ответа от ведомого.

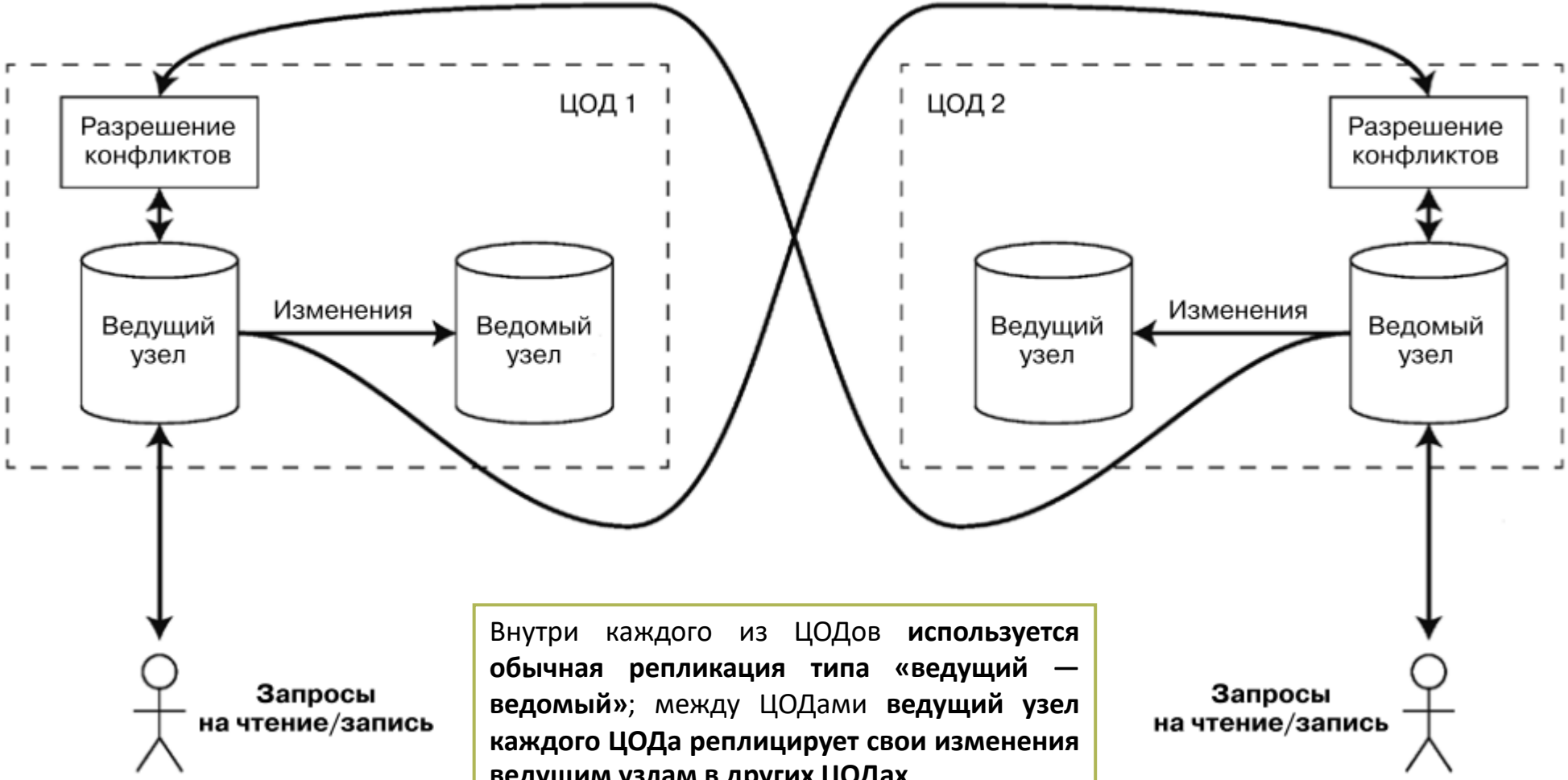
РЕПЛИКАЦИЯ С НЕСКОЛЬКИМИ ВЕДУЩИМИ УЗЛАМИ

Репликация будет выполняться так же, как и ранее: **каждый узел, обрабатывающий операцию записи, должен перенаправлять информацию о данных изменениях всем остальным узлам.** Это называется схемой **репликации с несколькими ведущими узлами** (multi-leader replication), или **репликацией типа «главный — главный»** (master — master), или **репликацией типа «активный/активный»** (active/active replication). При такой схеме **каждый из ведущих узлов одновременно является ведомым для других ведущих**.

Выход из строя одного из серверов практически всегда приводит к потере каких-то данных. Последующее восстановление также сильно затрудняется необходимостью ручного анализа данных, которые успели либо не успели скопироваться. **Использовать репликацию с несколькими ведущими узлами в пределах одного центра обработки данных (ЦОД) редко имеет смысл**, поскольку выгоды от такого решения едва ли перевесят привнесенное усложнение.

ЭКСПЛУАТАЦИЯ С НЕСКОЛЬКИМИ ЦОДами

В схеме с несколькими ведущими узлами **можно установить ведущий узел в каждом из ЦОДов.**



Внутри каждого из ЦОДов **используется обычная репликация типа «ведущий — ведомый»**; между ЦОДами **ведущий узел каждого ЦОДа реплицирует свои изменения ведущим узлам в других ЦОДах**.

Сравнение схемы с одним и несколькими ведущими узлами

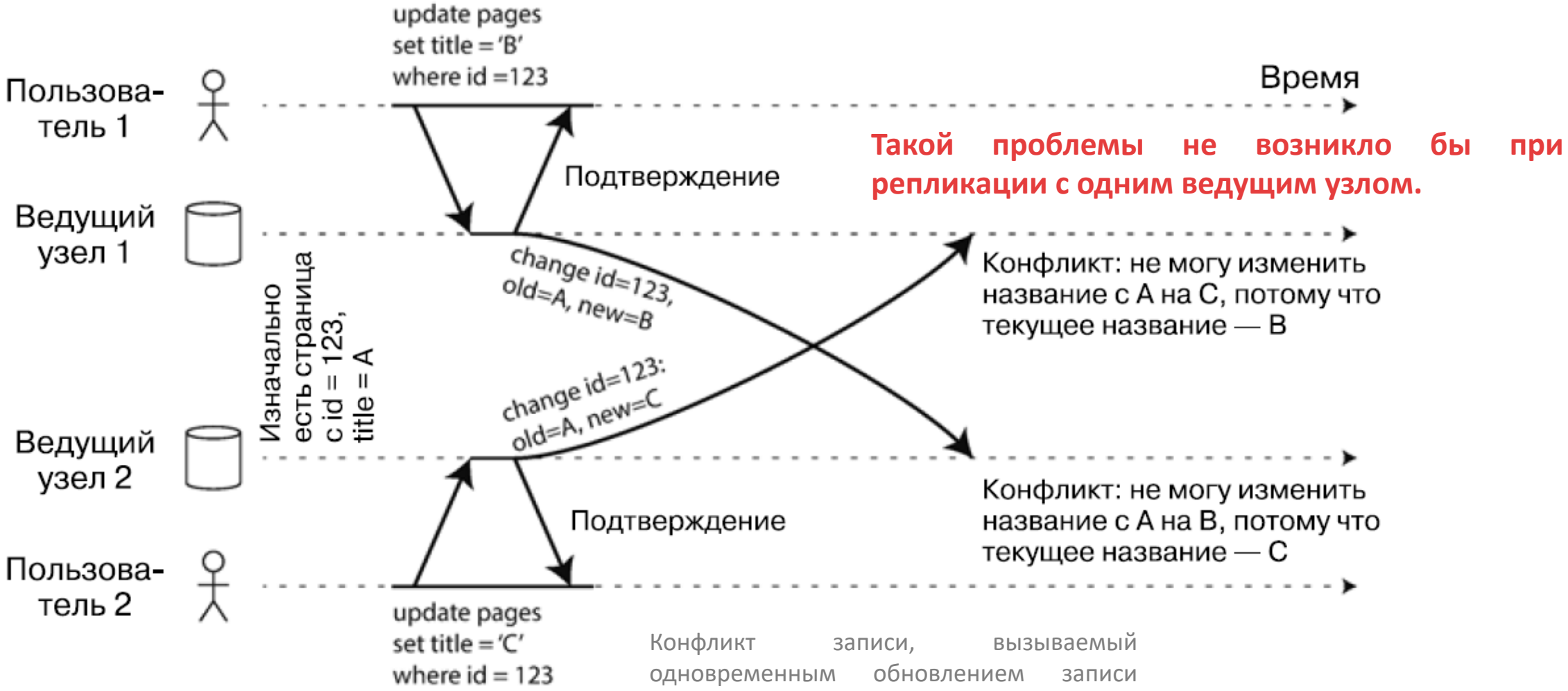
Производительность. При схеме **с одним ведущим узлом** каждая операция записи должна проходить через Интернет в ЦОД, содержащий ведущий узел. При схеме **с несколькими ведущими узлами** все операции записи будут обрабатываться в локальных ЦОДах и реплицироваться асинхронно в остальные ЦОДы. Таким образом, **сетевые задержки между ЦОДами становятся незаметными для пользователей**, а значит, **субъективная производительность возрастет.**

Устойчивость к перебоям в обслуживании ЦОДов. При схеме **с одним ведущим узлом** в случае отказа ЦОДа, в котором находится ведущий узел, восстановление после сбоя сделает ведомый узел в другом ЦОДе ведущим. При схеме **с несколькими ведущими узлами** каждый ЦОД может работать независимо от остальных и репликация наверстывает упущенное после возобновления работы отказавшего ЦОДа.

Устойчивость к проблемам с сетью. Схема **с одним ведущим узлом** очень чувствительна к сбоям работы этого соединения между ЦОДами, поскольку операции записи выполняются через него синхронно. Схема асинхронной репликации **с несколькими ведущими узлами** обычно более устойчива к проблемам с сетью: временные сбои работы сети не мешают обработке операций записи

Обработка конфликтов записи

Основная **проблема репликации с несколькими ведущими узлами** — возможность **возникновения конфликтов записи**, которые требуют разрешения.



Конфликт записи, вызываемый одновременным обновлением записи двумя ведущими узлами

Предотвращение конфликтов

Простейшая стратегия для конфликтов — просто избегать их: если приложение способно гарантировать, что все операции изменения конкретной записи проходят через один и тот же ведущий узел, то конфликт просто невозможен. Поскольку многие реализации репликации с несколькими ведущими узлами не очень хорошо разрешают конфликты, **зачастую предотвращение конфликтов — рекомендуемый подход.**

Например, если в приложении пользователь может редактировать собственные данные, то ***необходимо обеспечить, чтобы запросы от конкретного человека всегда маршрутизировались на один и тот же ЦОД и применяли ведущий узел последнего для чтения и записи***. У разных людей могут оказаться разные «родные» ЦОДы (например, подобранные исходя из географической близости к пользователю), но с точки зрения любого отдельного пользователя это фактически конфигурация с одним ведущим узлом.

Сходимость к согласованному состоянию

В схеме с несколькими ведущими узлами нельзя задать порядок операций записи, так что непонятно, **каким должно быть итоговое значение**. Если все реплики просто будут применять операции записи в том порядке, в котором они эти операции записи получают, то **база данных в конце концов окажется в несогласованном состоянии.** Это недопустимо: каждая схема репликации обязана обеспечить наличие одинаковых данных во всех репликах. Следовательно, **база должна разрешать конфликт конвергентным способом**, то есть **все реплики должны сойтись к одному значению после репликации всех изменений.**

Присвоить каждой операции записи уникальный идентификатор (метку даты/времени, случайное длинное число, UUID или хеш ключа и значения), после чего просто выбрать операцию («победителя») с максимальным значением этого идентификатора, а остальные отбросить.

Присвоить уникальный идентификатор каждой реплике и считать, что у исходящих от реплик с большим номером операций записи есть приоритет перед теми, которые исходят от реплик с меньшим. Этот подход также приводит к потерям данных.

Каким-либо образом слить значения воедино, например, выстроить их в алфавитном порядке, после чего выполнить их конкатенацию.

Заносить конфликты в заданную в явном виде структуру данных для хранения и написать код приложения, который бы разрешал конфликты позднее (возможно, спрашивая для этого пользователя).

ТОПОЛОГИИ РЕПЛИКАЦИИ С НЕСКОЛЬКИМИ ВЕДУЩИМИ УЗЛАМИ

Топология репликации (replication topology) описывает **пути, по которым операции записи распространяются с одного узла на другой.** В случае двух ведущих узлов, существует только одна разумная топология: ведущий узел 1 должен отправлять информацию обо всех своих операциях записи ведущему узлу 2 и наоборот. В случае более чем двух ведущих узлов возможны различные варианты топологии.

Топология типа «кольцо», при которой каждый узел получает информацию об операциях записи от ровно одного узла и передает ее (плюс информацию о своих собственных операциях записи) ровно одному из других узлов.

Топология «звезда»: один узел, назначенный корневым, пересылает информацию об операциях записи всем остальным узлам. Топологию

Наиболее общий вариант топологии — **каждый с каждым**, при которой каждый ведущий узел отправляет информацию об операциях записи всем остальным таким узлам.

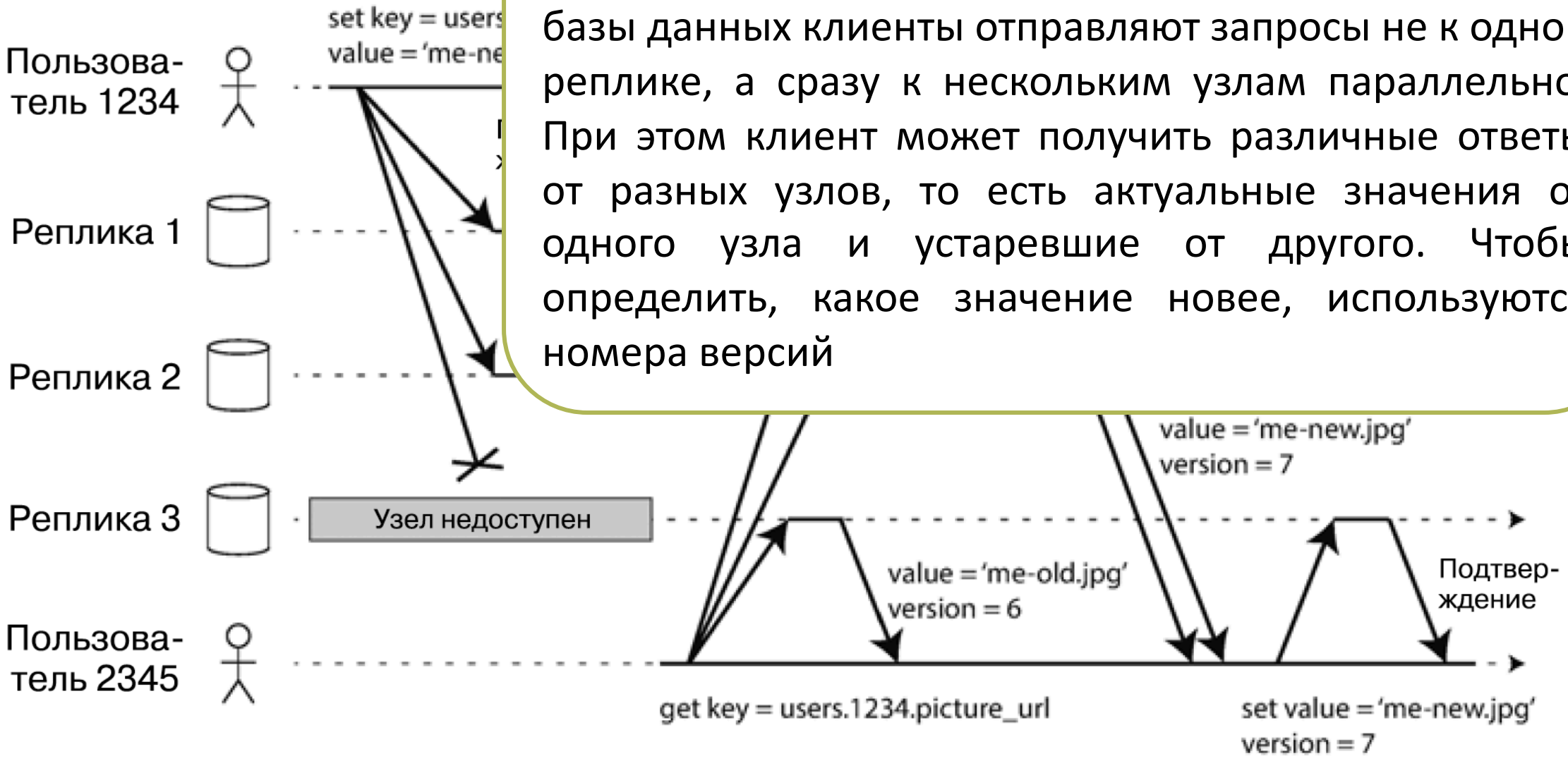
РЕПЛИКАЦИЯ БЕЗ ВЕДУЩЕГО УЗЛА

Некоторые системы хранения данных используют другой подход, отказываясь от концепции ведущего узла и позволяя **непосредственное поступление информации об операциях записи на все реплики.**

В некоторых реализациях **репликации без ведущего узла** клиент непосредственно отправляет информацию о своих операциях записи **на несколько реплик, в то время как для остальных данную манипуляцию от имени клиента совершает узел-координатор.** В отличии от БД с ведущим узлом, он **не навязывает определенный порядок операций записи.**

Такая архитектура снова вошла в моду после того, как Amazon задействовал ее для своей предназначенной для внутреннего использования системы Dynamo. Riak, Cassandra и Voldemort представляют собой вдохновленные Dynamo склады данных с открытым исходным кодом. Поэтому подобный тип БД называют Dynamo-подобной базой данных (Dynamo-style database).

Для решения указанной проблемы при чтении из базы данных клиенты отправляют запросы не к одной реплике, а сразу к нескольким узлам параллельно. При этом клиент может получить различные ответы от разных узлов, то есть актуальные значения от одного узла и устаревшие от другого. Чтобы определить, какое значение новее, используются номера версий

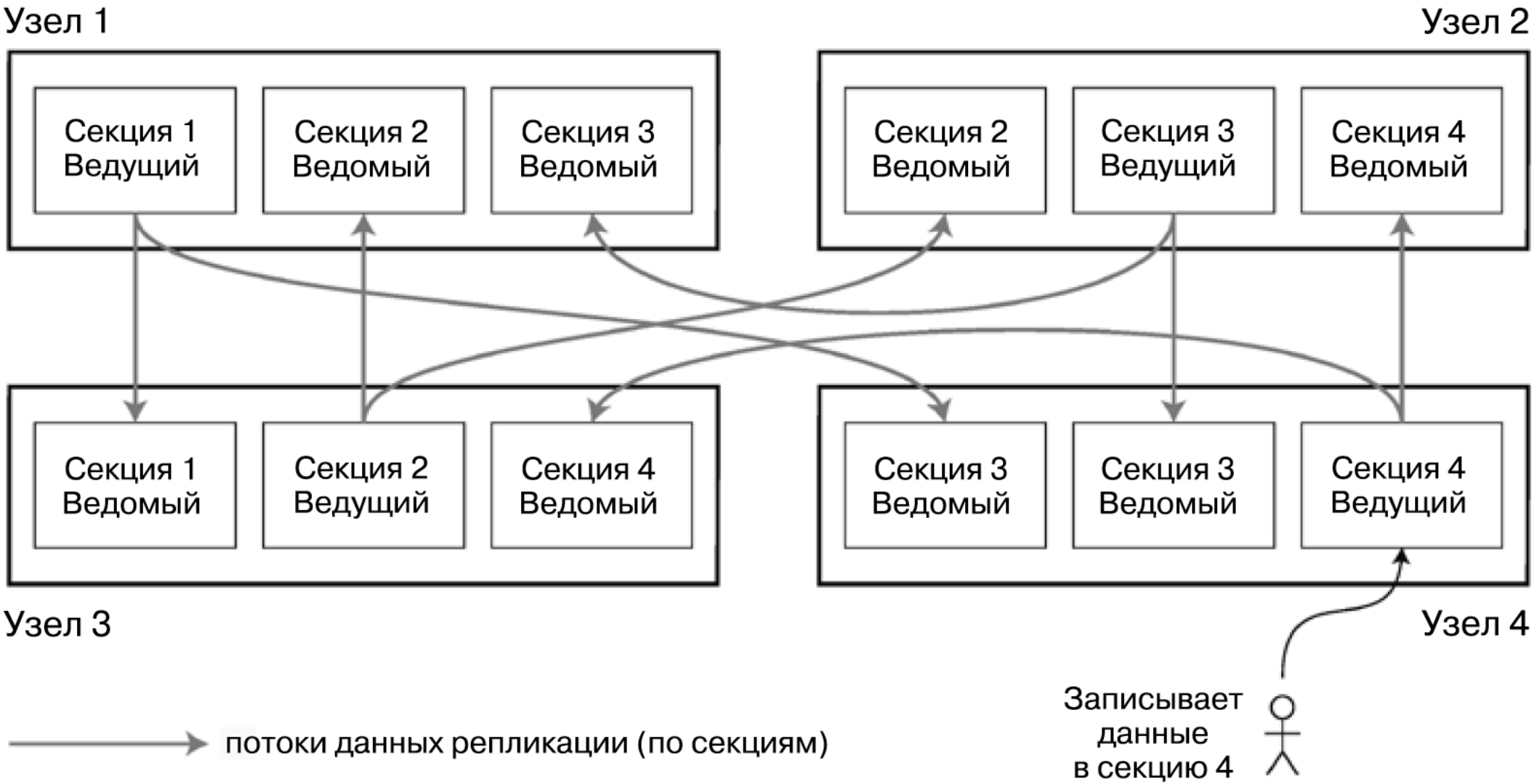


СЕКЦИОНИРОВАНИЕ

Чтобы разбить данные по машинам, используют **шардинг (секционирование).** То есть разбиение данных на секции и блоки по ключу (или другим признакам). **Смысл именно в том, чтобы по определенному признаку разделить данные и отправить их на разные машины.**

Основная цель секционирования данных — масштабируемость. Разные секции можно разместить в различных узлах в кластере, не предусматривающем разделения ресурсов. Следовательно, большой набор данных можно распределить по многим жестким дискам, а запросы — по многим процессорам.

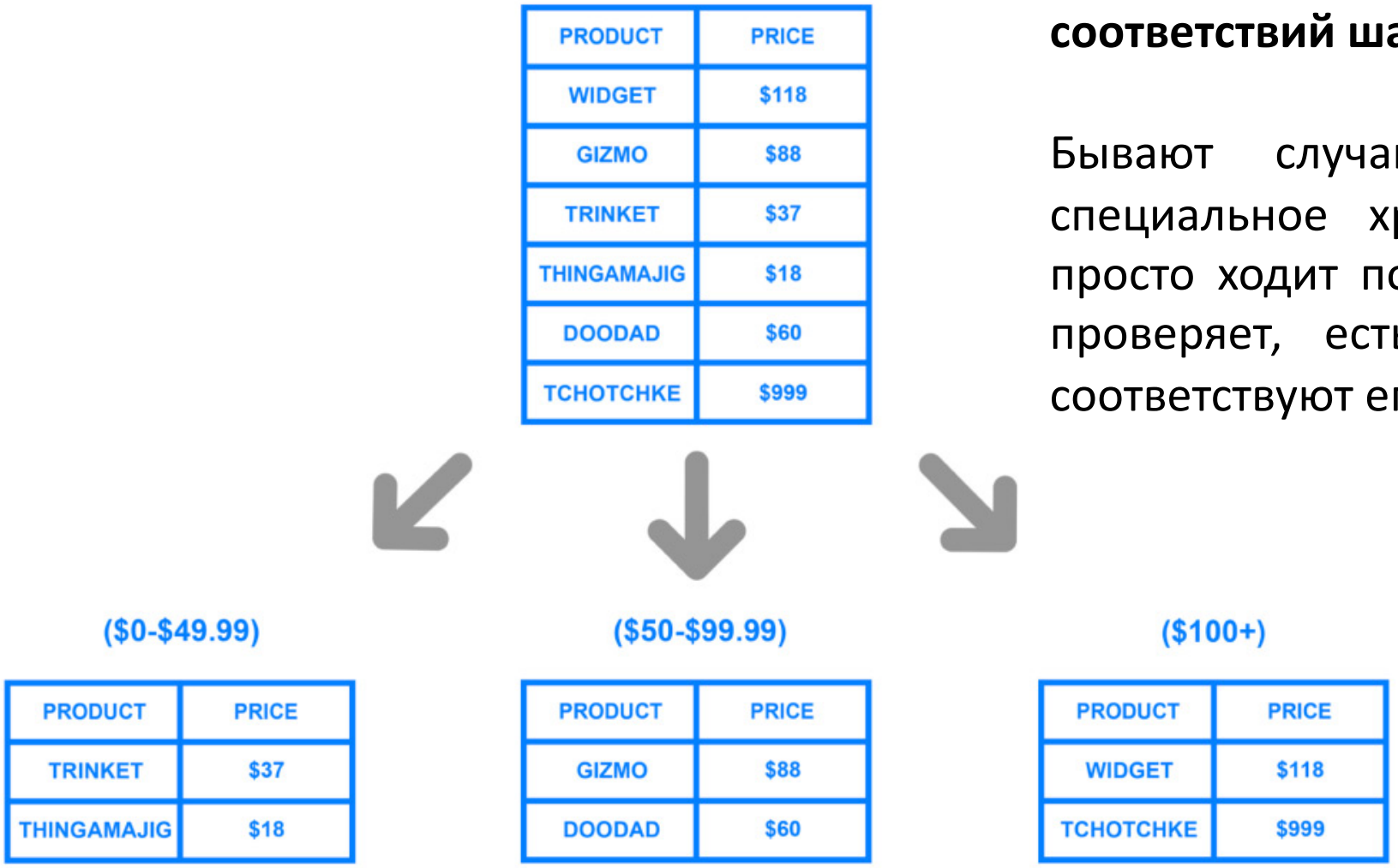
Секционирование обычно «идет бок о бок» с репликацией, вследствие чего **копии каждой из секций хранятся на нескольких узлах.** Это значит, что, хотя каждая конкретная запись относится только к одной секции, храниться она может в нескольких различных узлах в целях отказоустойчивости.



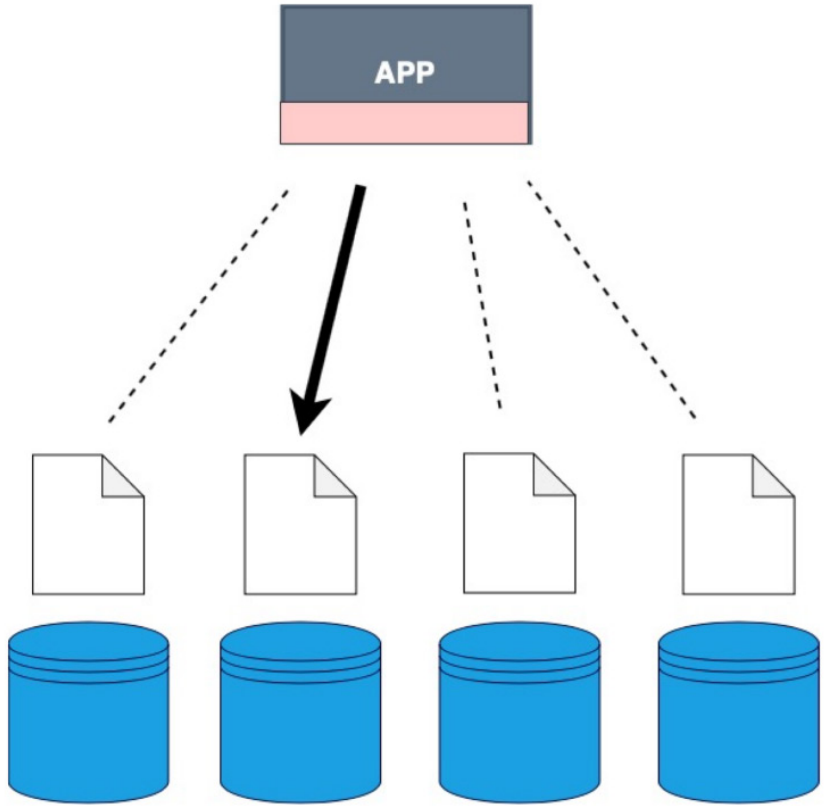
Комбинация секционирования и репликации: каждый узел выступает в качестве ведущего для одних секций и ведомого для других

Чтобы в целом понять, **где какие данные лежат, нужна определенная таблица соответствий шарда, копии и данных.**

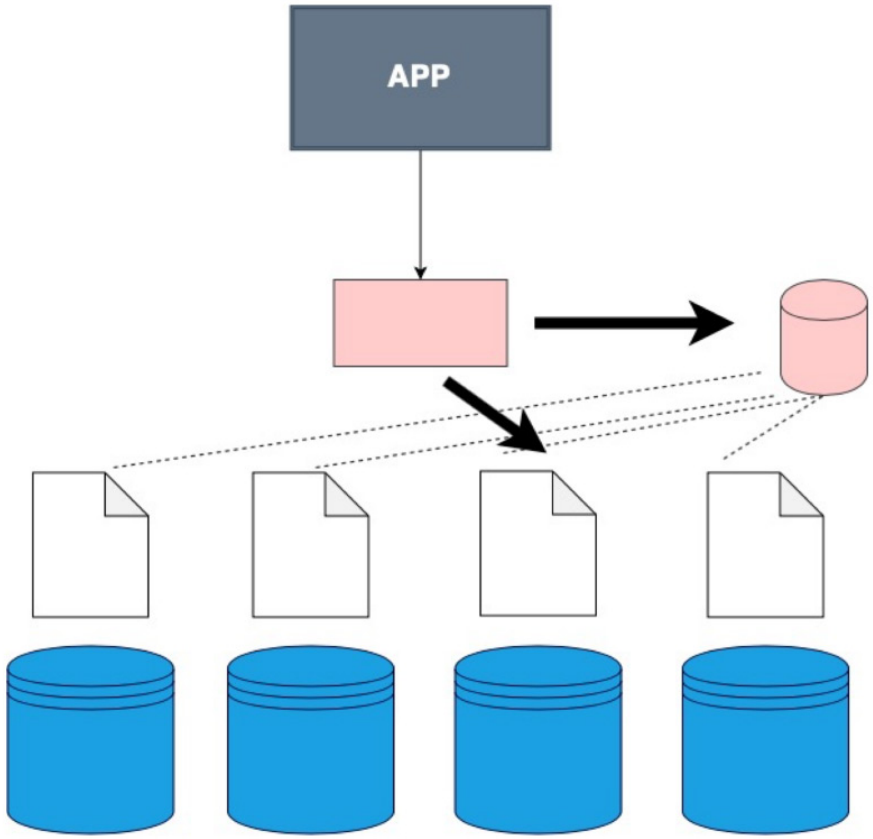
Бывают случаи, когда не используется специальное хранилище данных и клиент просто ходит по очереди на каждый шард и проверяет, есть ли там данные, которые соответствуют его запросу.



Бывает специальная программная прослойка, **которая хранит в себе определенные знания о том, в каком шарде какой диапазон данных лежит.**



Бывает толстый клиент. Это когда не **зашивается сам клиент в отдельную прослойку, а зашиваются в него самого данные о том, как шардированы данные.**



Распределение данных

Существует **два основных способа распределения данных по секциям**: **фиксированное и динамическое**. Для обеих стратегий необходима **функция** секционирования, которая принимает на **входе ключ секционирования строки и возвращает номер секции, в которой эта строка находится**.

Фиксированное распределение

Для фиксированного распределения **применяется функция разбиения**, которая зависит только от **значения ключа секционирования**. В качестве примеров можно привести деление по модулю или хеш-функции. Такие функции отображают значения ключей секционирования на конечное число «ячеек» (buckets), в которых хранятся данные.

Недостатки стратегии фиксированного распределения:

- если секции **велики и их немного**, то **балансировать нагрузку** между ними будет **сложно**;
- при **фиксированном распределении отсутствует возможности решать, куда помещать конкретную запись**;
- **изменить алгоритм** секционирования **сложнее**, потому что требуется перераспределить все существующие данные.

Динамическое распределение

Альтернативой фиксированному распределению является **динамическое распределение, описание которого хранится отдельно в виде отображения единицы секционирования на номер секции**.

```
CREATE TABLE user_to_shard (
  user_id INT NOT NULL,
  shard_id INT NOT NULL,
  PRIMARY KEY (user_id)
);
```

Функцией разбиения служит сама таблица. Зная ключ секционирования (идентификатор пользователя), можно найти соответствующий номер секции. Если подходящей строки не существует, можно выбрать нужную секцию и добавить строку в таблицу. Впоследствии сопоставление можно будет изменить, потому стратегия и называется динамической.

Динамическое распределение дает возможность создавать несбалансированные секции. Это полезно, когда не все сервера одинаково мощные или когда некоторые серверы используются еще и для других целей. Если при этом имеется еще и возможность в любой момент перебалансировать шарды, то можно поддерживать взаимно-однозначное соответствие между секциями и узлами, не растрачивая впустую емкость дисков.

Перераспределине данных (решардинг)

При необходимости можно переместить данные из одной секции в другую, чтобы сбалансировать нагрузку. Но *по возможности необходимо избегать перебалансирования*, так как это может вызвать приостановку обслуживания. Из-за перемещения данных становится сложнее добавлять в приложение новые функции, поскольку их приходится учитывать в сценариях перебалансирования.

Один из подходов – «update is a move». Всегда, при изменении ключа, он неявно двигается.

Второй подход – «data expiration». В нем добавляются несколько серверов, явно не проводится решардинг, – получилась новая схема шардинга. Этот же подход можно использовать где угодно, где есть возможность старые данные просто удалять. Тогда новые данные сохраняются на новые узлы (или закрываются какие-то старые). Данные сами собой просто постепенно ***переезжают.***