Работа с распределенными данными

ИТМО ВТ

Что мы хотим?

Высокий уровень доступности — (high availability) способность системы работать без сбоев (непрерывно) в течение определенного периода времени.

Система должна работать даже в случае **выхода из строя** части системы

Как обеспечить высокий уровень доступности:

- обеспечить избыточность важных частей системы;

ИТМО ВТ

Что мы хотим?

Масштабируемость — может не хватать ресурсов одного сервера.

Решение 1.

Вертикальное масштабирование — использовать более мощный сервер.

Недостатки:

- ограничение по ресурсам
- соотношение цена системы/производительность.

ИТМО ВТ

Масштабируемость

Решение 2

Горизонтальное масштабирование — распределить нагрузку на несколько машин.

- **Узлы** — отдельные серверы БД, обеспечивающие работу.
- Узлы физически независимы друг от друга
- Данные могут быть расположены в разных географических локациях.

Недостатки:

- Более сложная архитектура

ИТМО ВТ

Репликация

- **Репликация данных** — способ организации распределенных данных.
- Репликация заключается в сохранении и поддержании нескольких копий данных на разных устройствах.

- Базовые понятия:
  - **Узлы** — отдельные серверы БД, обеспечивающие работу. Совокупность узлов — **кластер**.
  - У каждого узла — своя БД (оригинал или копия, реплика). Копии данных синхронизируются в реальном времени.

ИТМО ВТ

Репликация

- Репликация в СУБД:
  - Как **осуществить**?

    Сделать **копии** данных на всех серверах.

  - В чем **сложность**?

    Данные **изменяются** — как поддерживать актуальные данные на всех копиях?

ИТМО ВТ

Репликация

Варианты реализации репликации:

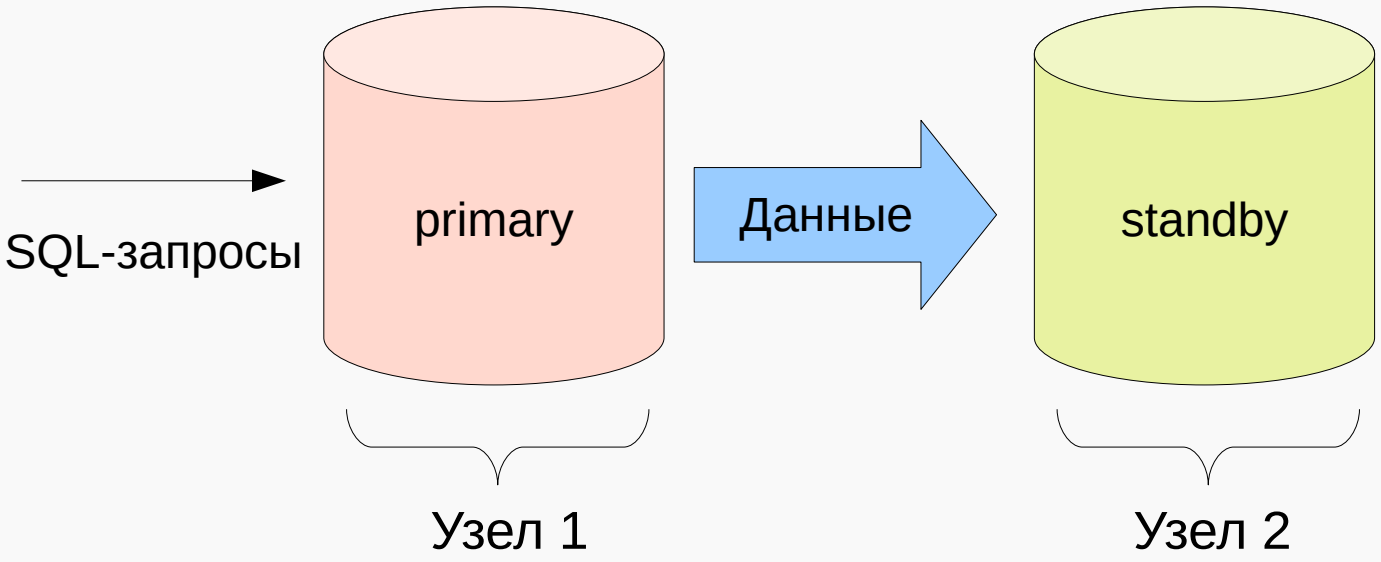
- Master/Slave репликация
- Master/Master репликация
- Репликация без мастера

Терминология:

- Один из серверов БД с репликой — главный (**master**). Режим — чтение/запись. Если несколько — multimaster-репликация.
- Остальные узлы — зависимые (**slave**, **standby**). Если режим — только чтение — **hot standby**.

ИТМО ВТ

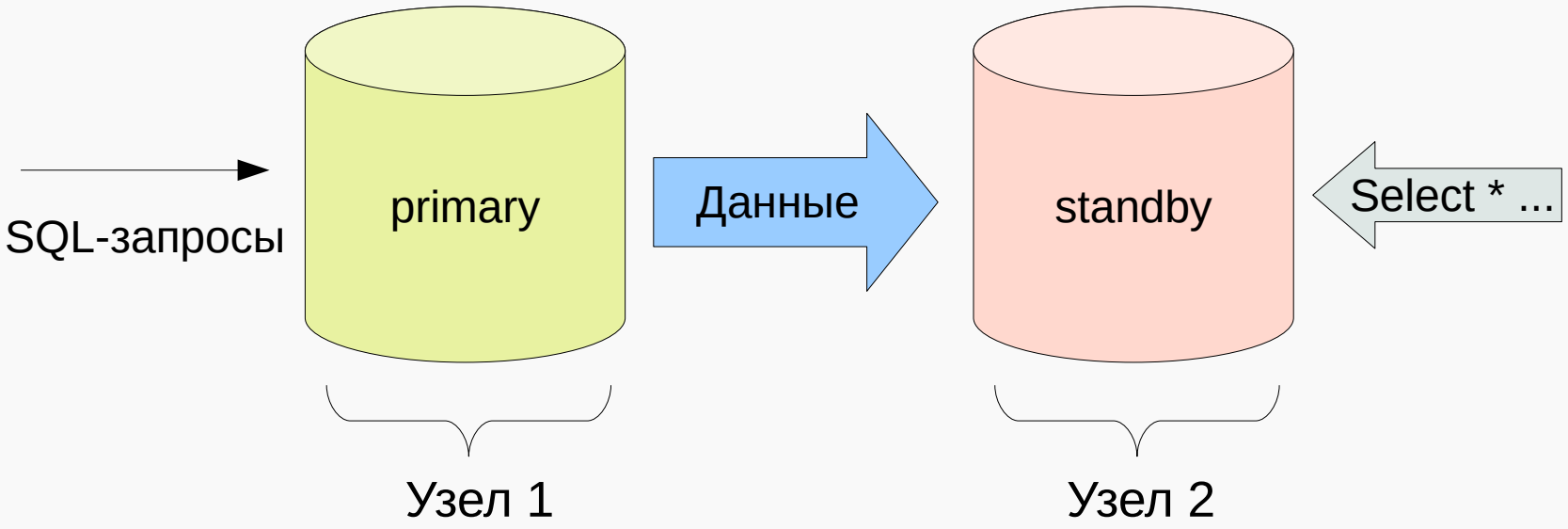
Master/Slave репликация



- Мастер принимает все запросы на изменение данных от клиентов.
- Данные об изменениях отправляется зависимым узлам.

ИТМО ВТ

Master/Slave репликация



- Зависимый узел применяет полученные изменения.
- Зависимый узел может принимать запросы на чтение — **hot standby**.
- Реализовано: в PostgreSQL, MongoDB, MySQL, ...

ИТМО ВТ

Виды репликации

- Репликация может быть:
  - **асинхронная** — данные для синхронизации состояний серверов БД пересылаются без подтверждения получения.
    - Мастер отправляет изменения зависимым узлам, но не ждет от них подтверждения.
    - Изменения становятся видны, когда их локально выполнил мастер.
  - **Преимущество**: зависимые узлы не оказывают серьезного влияния на время выполнения операций — мастер их не ждет.
  - **Недостаток**: возможна потеря данных — проблема с мастером, последние записи не попали на зависимый узел.

ИТМО ВТ

Виды репликации

- Репликация может быть:
  - **синхронная** — пересылки данных идут с подтверждением.
    - Мастер ждет, когда зависимые узлы подтвердят получение данных об изменениях мастера.
    - Изменения становятся доступны, когда зависимые узлы подтвердят их получение
  - **Преимущество**: данные на зависимом узле точно актуальны и синхронизированы с мастером.
  - **Недостаток**: мастеру нужно будет блокировать операцию, если зависимый узел не доступен/не отвечает.

ИТМО ВТ

Синхронная репликация в PostgreSQL

- Определяет, нужно ли ждать обработки WAL на репликах, когда происходит COMMIT транзакции.
- параметр synchronous_commit (default = on) + synchronous_standby_names:
  - по умолчанию включена асинхронная репликация.
- Дает возможность подтвердить, что транзакция реплицирована хотя бы на одном standby.
- Указать список standby серверов с синхронной репликацией:

  synchronous_standby_names = 'node2, node3, node4'

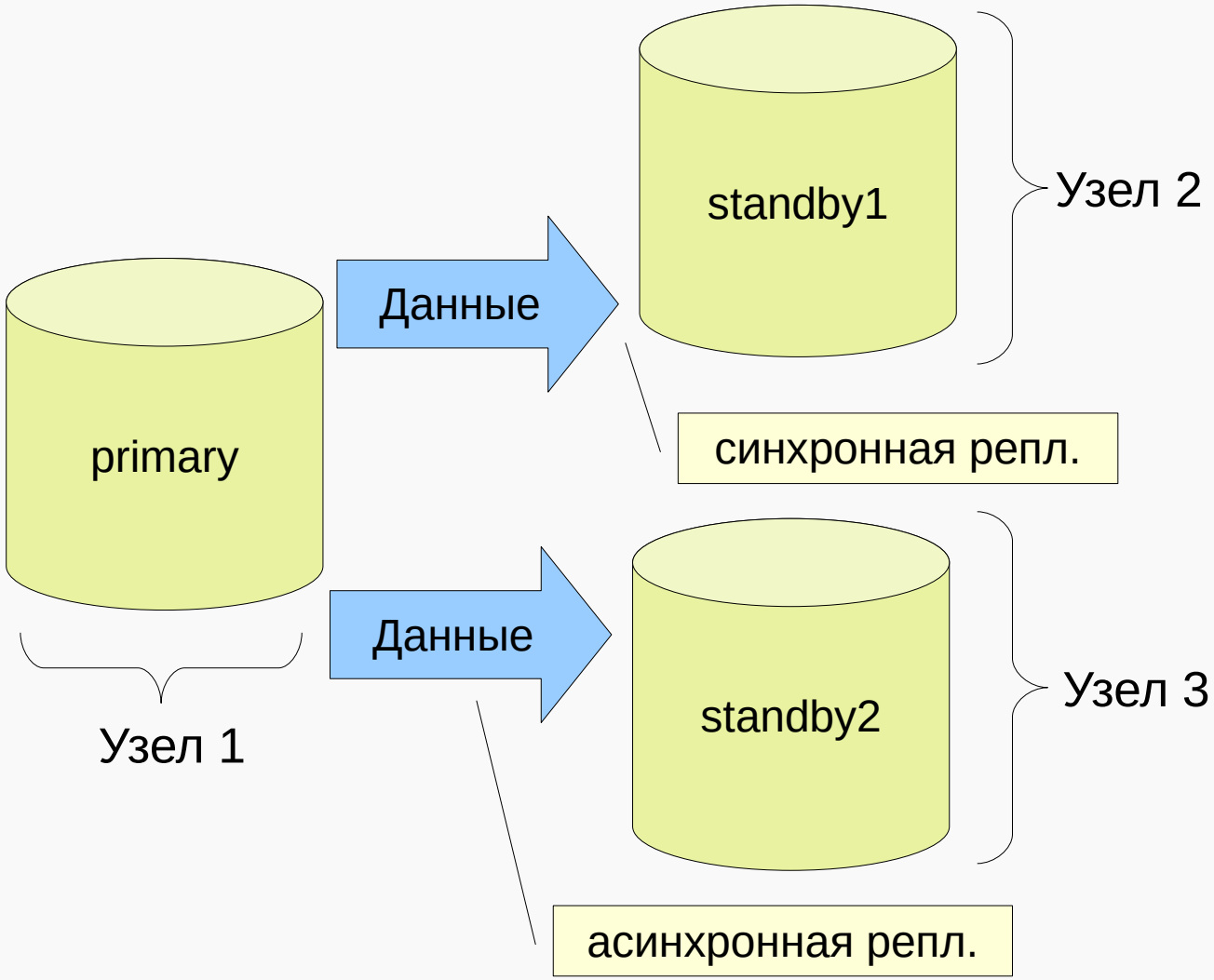
ИТМО ВТ

Synchronous commit

	Долгов-ть локального коммита	Долгов-ть сбой БД	Долгов-ть сбой ОС	standby - согл-ть запросов
remote_apply	+	+	+	+
on	+	+	+	
remote_write	+	+		
local	+			
off				

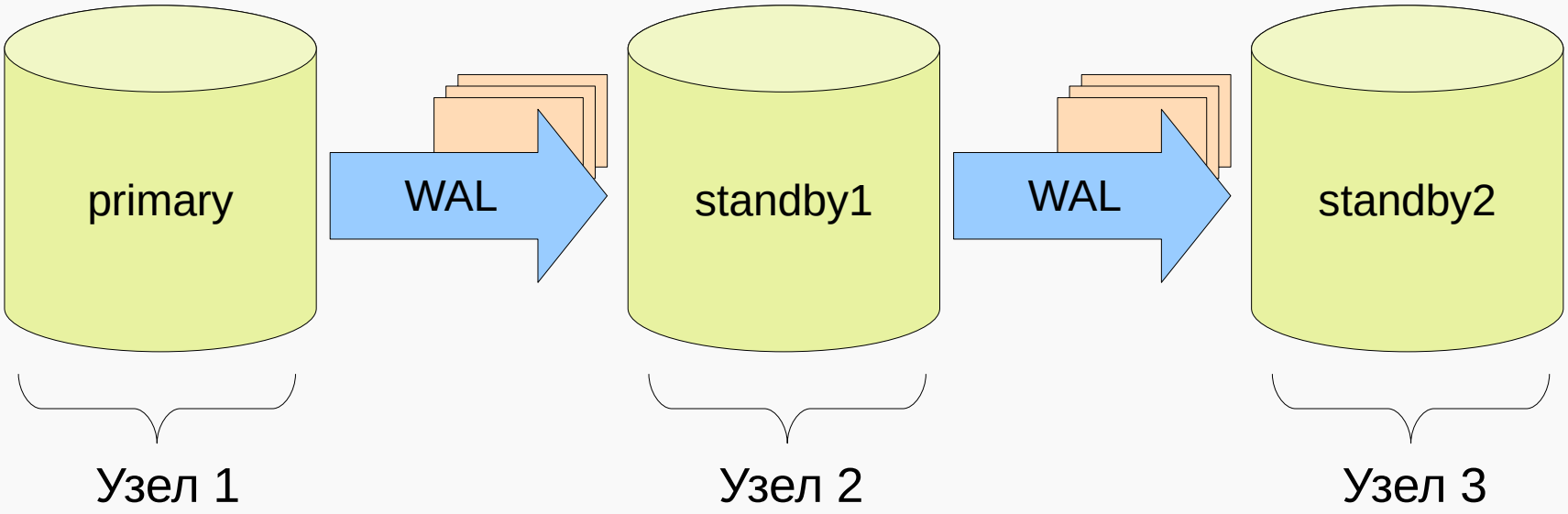
ИТМО ВТ

Полусинхронная репликация



ИТМО ВТ

Ступенчатая репликация: cascading replication



ИТМО ВТ

Реализация репликации (M/S)

- Реализация репликации может быть:
  - **Физическая** — пересылаются файлы (WAL). standby — содержит копию primary.
  - **Логическая** — пересылаются декодированные из WAL операции.
  - **Дублирование команд** — на зависимые узлы пересылаются все команды с мастера.
  - **Триггерная** — используются средства БД для фиксации изменений, дальше изменения пересылаются в другую БД.

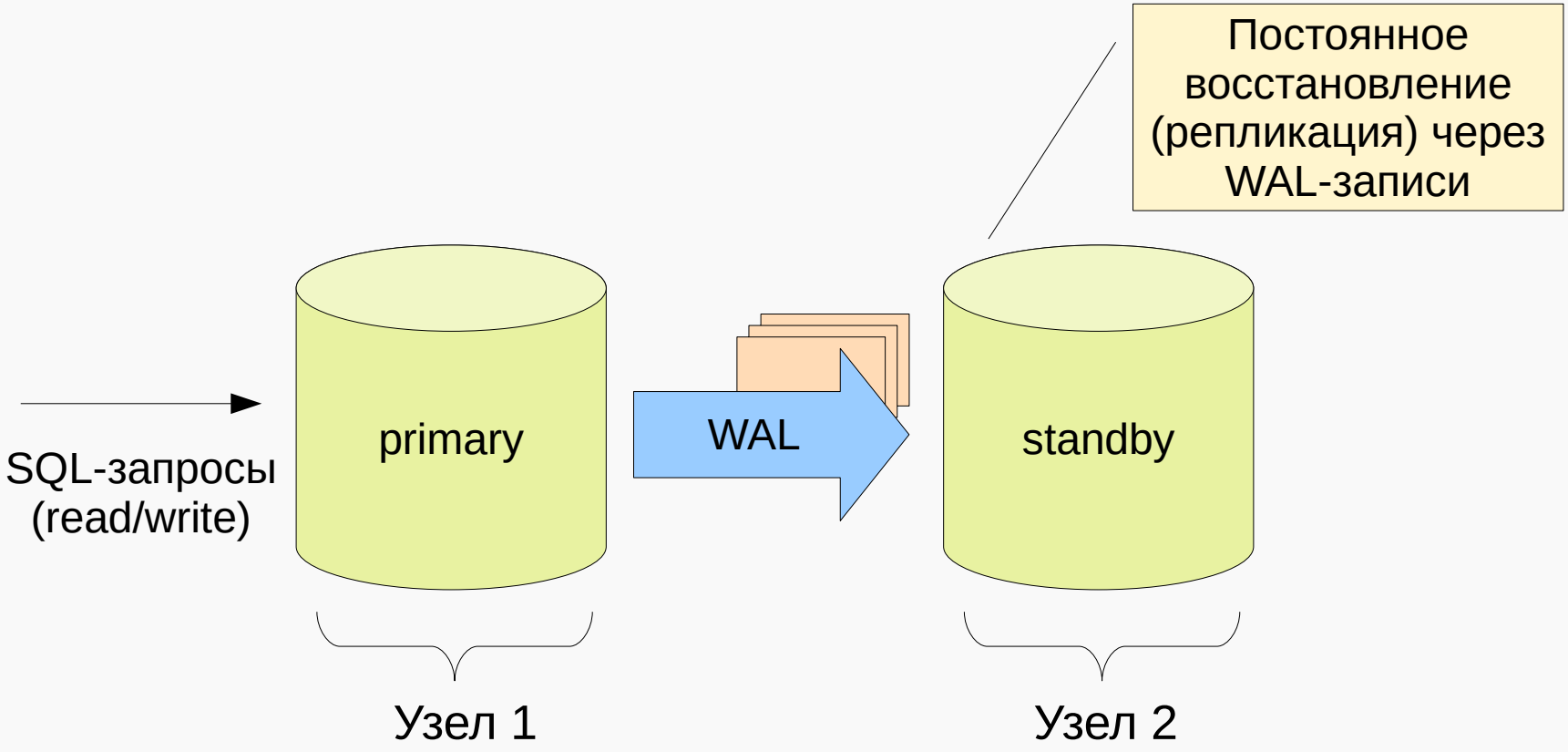
ИТМО ВТ

Физическая репликация

- **Физическая репликация** — пересылаются файлы (WAL). Standby — содержит копию primary.
- Требуется — wal_level = replica (по умолчанию).
- Потоковая репликация (**streaming replication**) — WAL-записи передаются по соединению между primary и standby:
  - Использует свой протокол для отправки WAL-файлов — WAL-sender, WAL-receiver.
- standby — осуществляет восстановление на основе полученных WAL для поддержки актуального состояния.

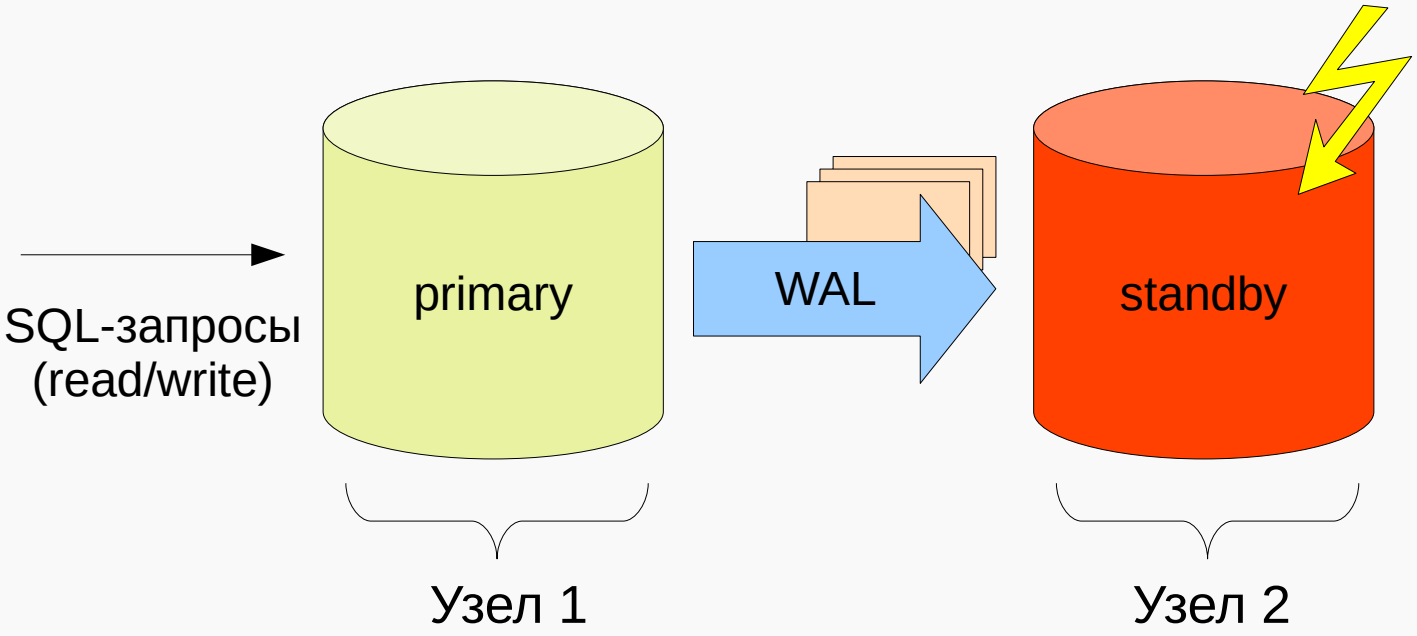
ИТМО ВТ

Потоковая репликация

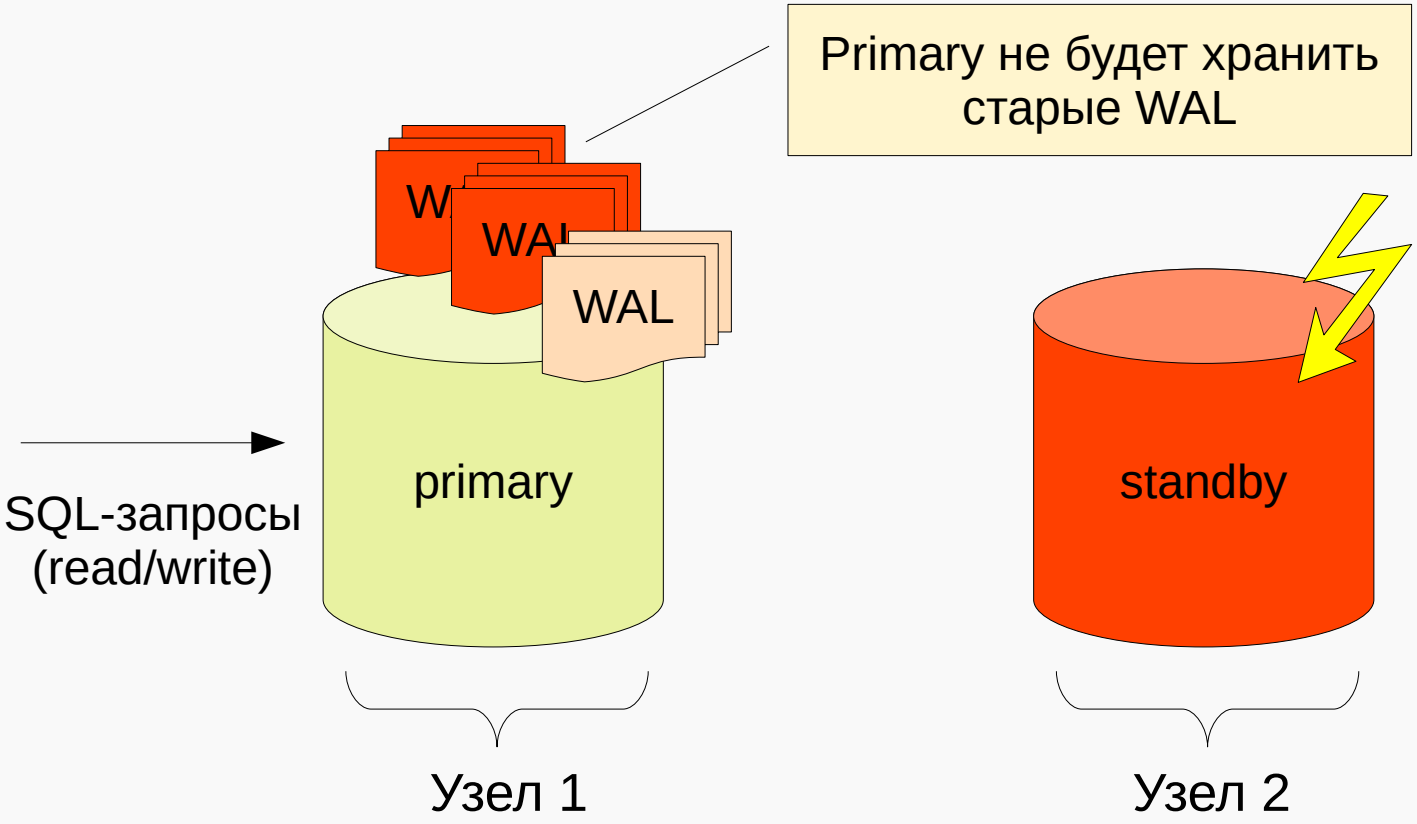


ИТМО ВТ

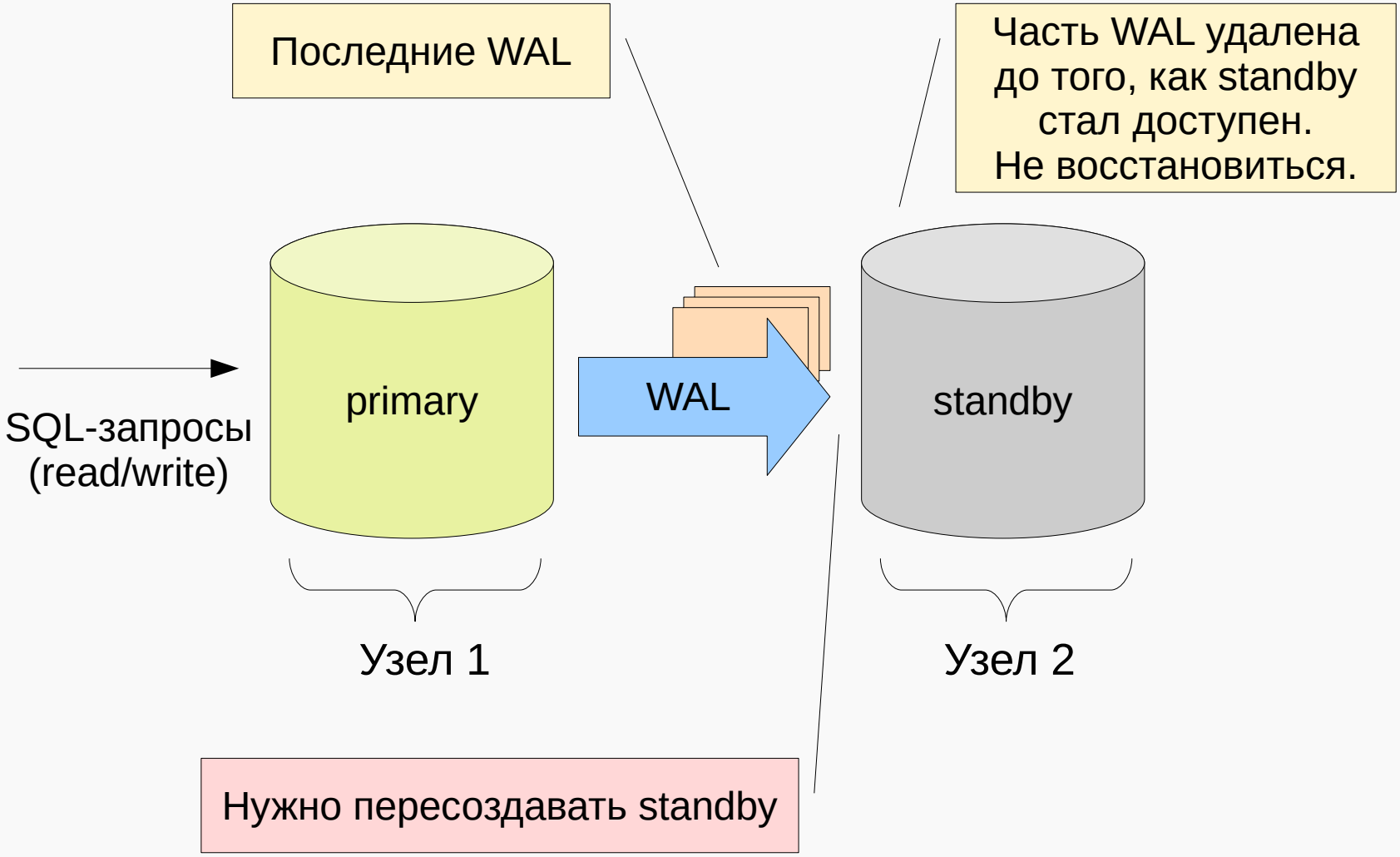
Что если произошла временная проблема со standby?



Что если произошла временная проблема со standby?



Что если произошла временная проблема со standby?



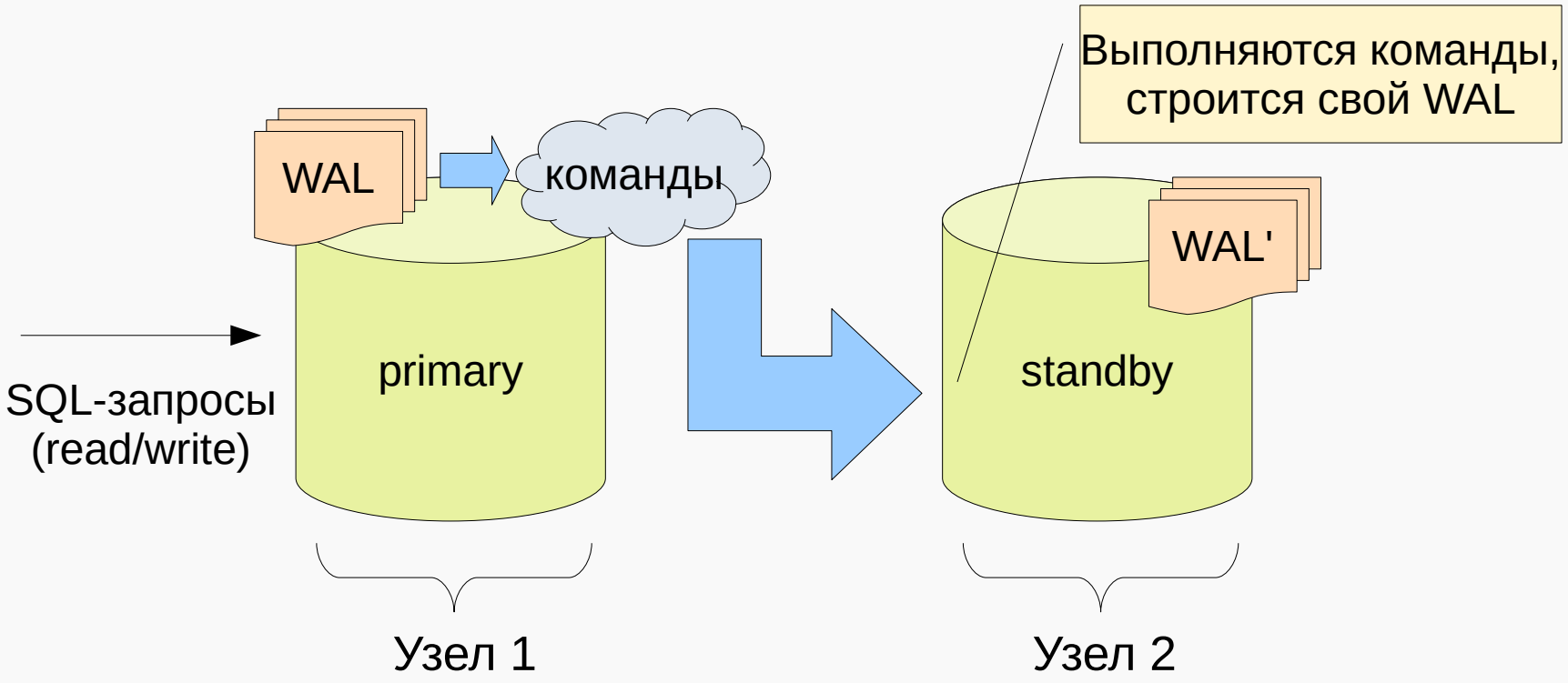
ИТМО ВТ

Логическая репликация

- **Логическая репликация** — пересылаются команды, реконструированные из WAL.
- Требуется — wal_level = logical.
- standby — осуществляет восстановление на основе полученных команд для поддержки актуального состояния.
- Используется модель «публикаций» «подписок»:
  - после логического декодирования команды публикуются на мастере;
  - опубликованные команды могут получить подписанные standby.

ИТМО ВТ

Логическая репликация



ИТМО ВТ

Особенности

- Можно реплицировать часть БД (отдельный набор таблиц).
- Не поддерживается репликация DDL.
- Возможна репликация даже для узлов с разными версиями PostgreSQL.
- Standby может работать не в read-only режиме.

ИТМО ВТ

Репликация через дублирование команд

- На зависимые узлы пересылаются все команды с мастера (SQL).
- Каждый узел воспроизводит эти команды самостоятельно.
- Недостатки:
  - при выполнении команд на узлах возможно получение различных результатов — now(), побочные эффекты.
  - нужен контроль одинакового порядка выполнения команд на разных узлах.

ИТМО ВТ

Триггерная репликация

- Через триггеры (или аналогичные средства СУБД) можно отлавливать события, связанные с изменением данных.
- Изменения фиксируются в вспомогательные таблицы/объекты.
- Таблицы/объекты с изменениями используются для переноса изменений на другие узлы внешними процессами.

ИТМО ВТ

hot standby

- **warm standby** — режим standby, при котором логи (WAL), архивированные главным сервером, собираются standby и происходит непрерывное восстанавление состояния БД по полученным WAL. Warm standby не позволяет осуществлять запросы.
- **hot standby** — сервер работает в режиме чтения — кроме восстановления по WAL, позволяет осуществлять запросы к данным (read-only).

- Благодаря hot standby — мы можем **снизить нагрузку** на мастер узел.

ИТМО ВТ

Проблемы при организации hot standby (1)

- Если все зависимые узлы — **синхронные**, то система фактически будет **недоступной:**
  - при отказе одного из узлов во время операции изменения данных.
- если зависимые узлы работают, изменение данных будет применено только при получении ответа мастером от всех его синхронных узлов.

Проблемы при организации hot standby (2)

- Если зависимые узлы — **асинхронные**, то клиент при чтении данных с такого узла может получать устаревшие данные.
- До момента применения изменений на зависимом узле:
  - для одного и того же запроса может быть возвращен разный результат при его выполнении на мастере и асинхронном зависимом узле.
  - После применения изменений на зависимом узле — данные снова согласованы.

ИТМО ВТ

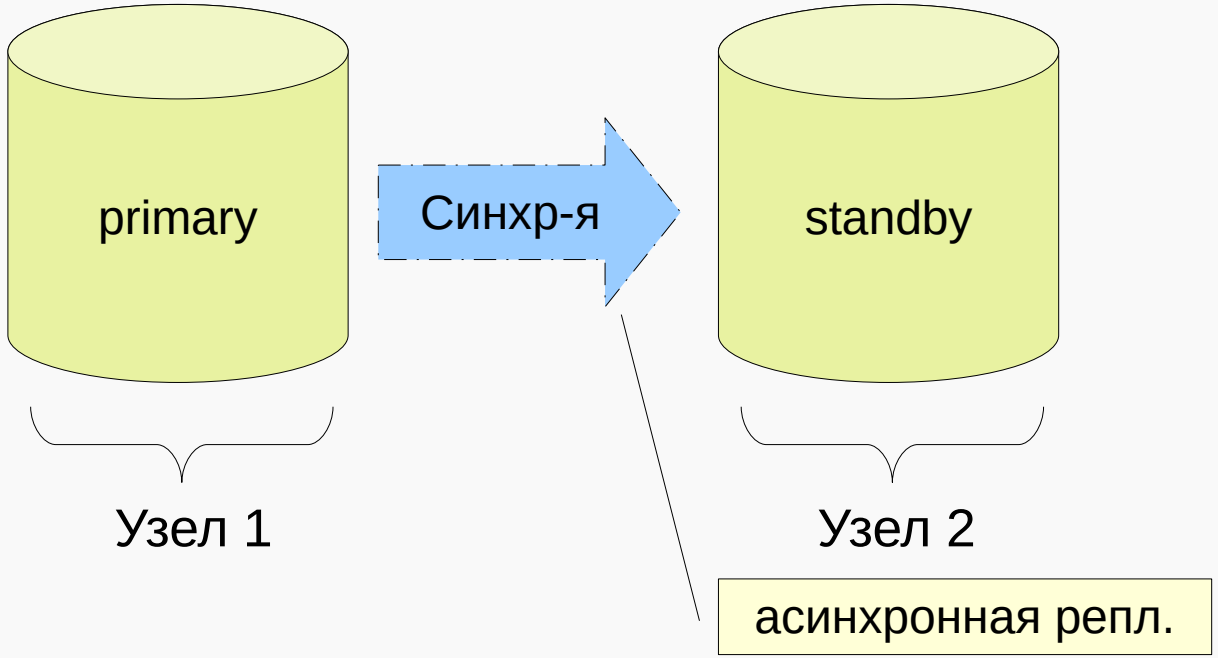
Конечная согласованность

- **Конечная согласованность** (eventual consistency) — эффект, при котором актуальные изменения распределяются по узлам и применяются постепенно.
- Часть кластера в некоторый момент времени может содержать **устаревшие** данные:
  - в случае M/S зависимые узлы не успели получить/применить актуальные изменения мастера.
- **Задержка репликации** (replication lag) — задержка во времени между завершением изменения на мастере и ее применении на зависимом узле.

Проблема чтения своих записей

Есть M/S кластер, ведомый узел — в режиме **асинхронной репликации**.

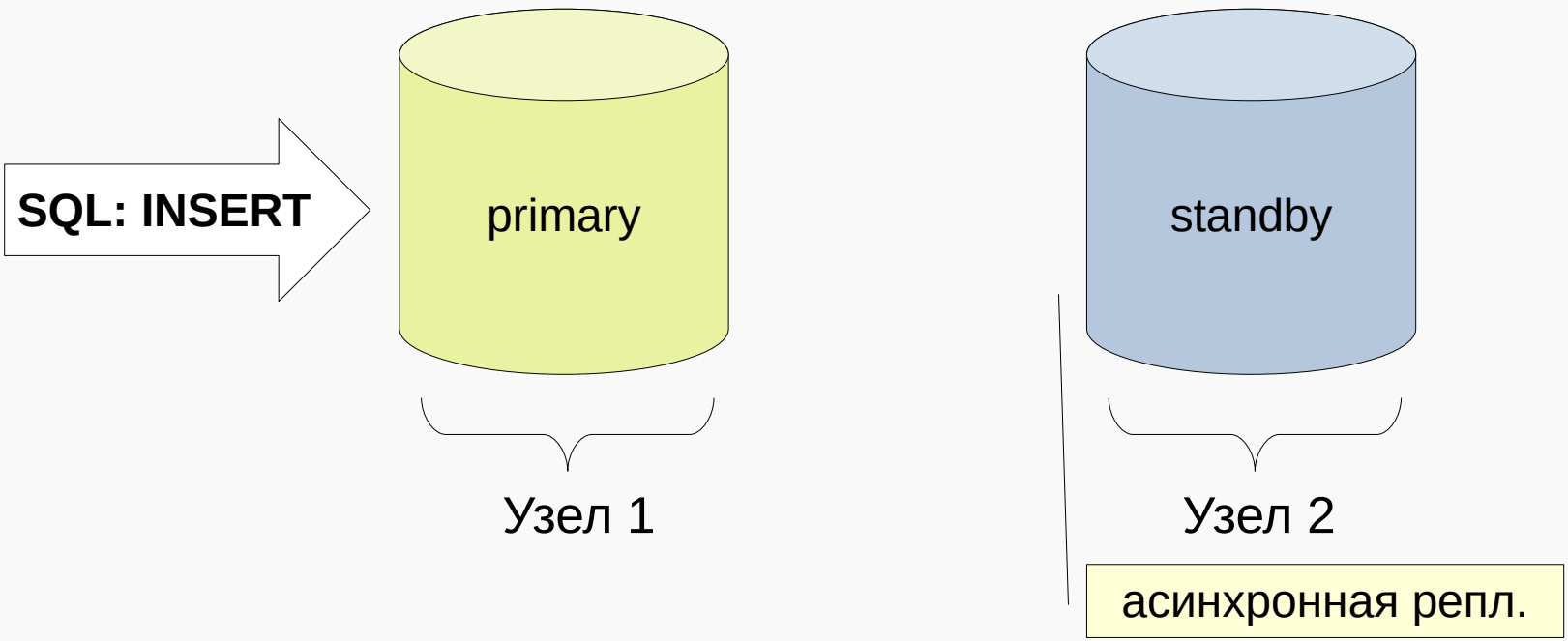
Ведомый узел — **hot standby**, balancer отдает ему некоторые запросы на выполнение



Проблема чтения своих записей

1. Если пользователь добавил/изменил свои данные:

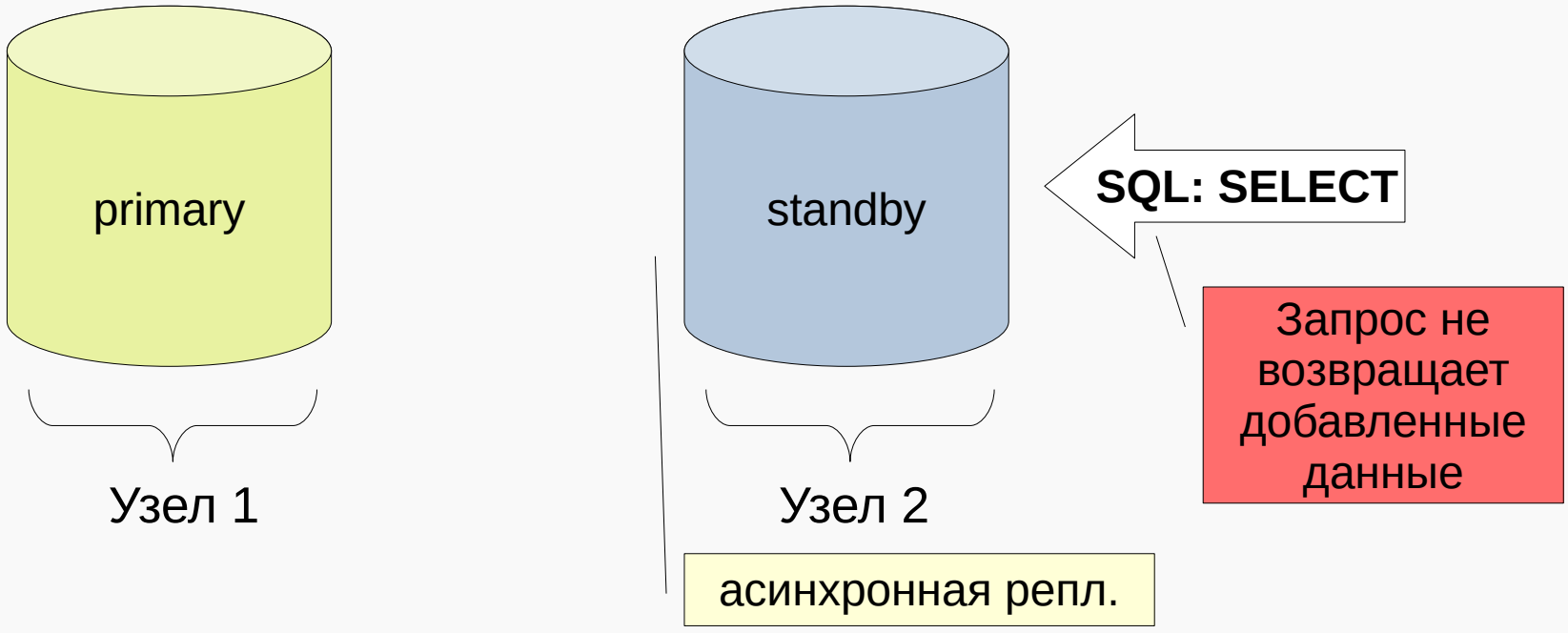
INSERT STUDENT (id) VALUES (3);



Проблема чтения своих записей

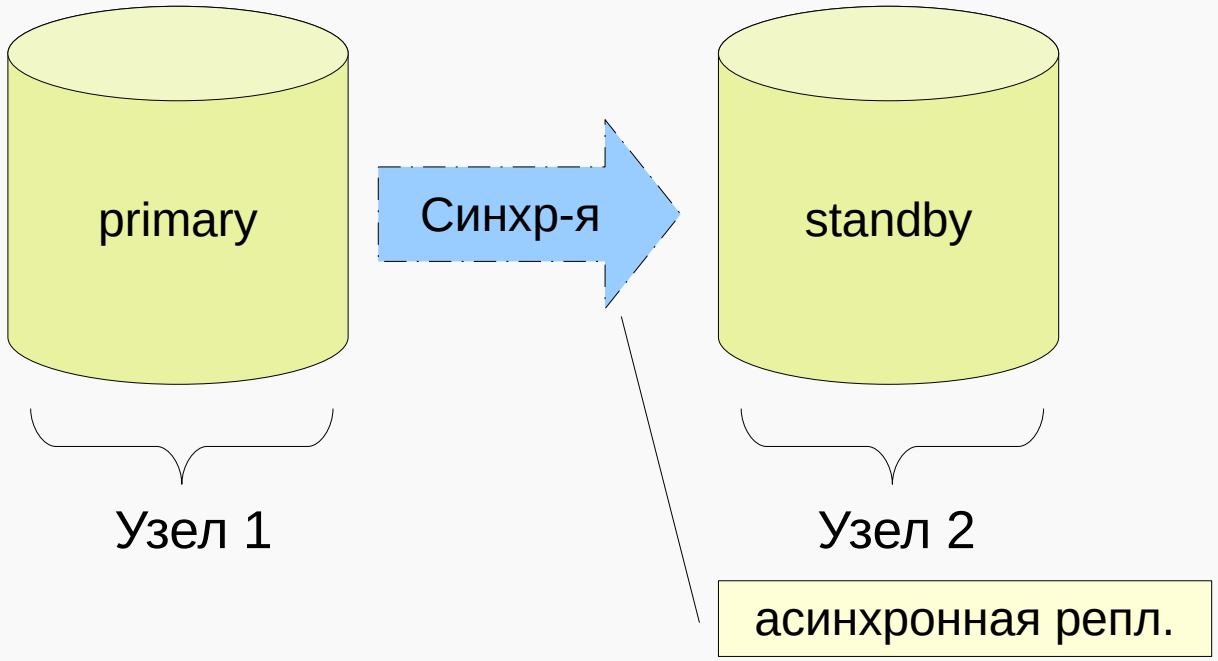
2. Пользователь запрашивает только что внесенные данные:

SELECT * FROM STUDENT WHERE id = 3;



Проблема чтения своих записей

3. Синхронизация узлов происходит после обработки этих запросов.



ИТМО ВТ

Чтение своих записей

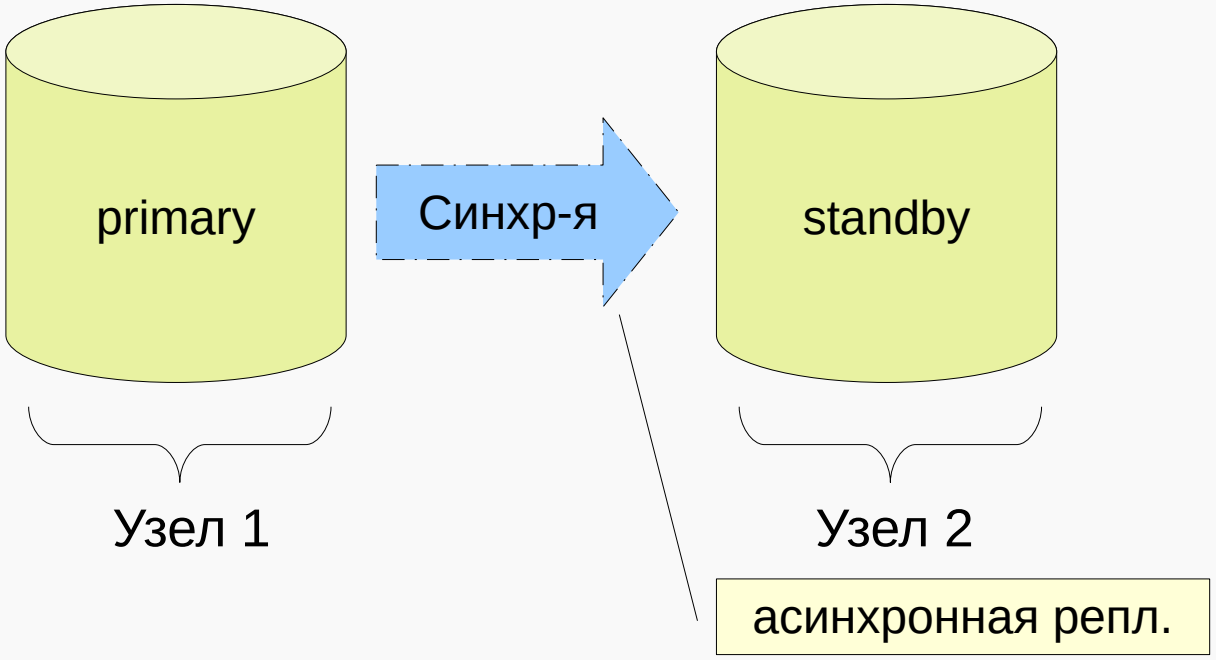
Возможные варианты решения проблемы:

- Читать данные, которые пользователь мог изменить/добавить с мастера, остальные — с ведомых узлов.
- Следить за временем обновления данных в разных объектах/таблицах:
  - если **не прошел** установленный интервал времени с последнего обговления — читать данные с **мастера**;
  - если время **прошло** — с **зависимых** узлов.

Проблема монотонных чтений

Есть M/S кластер, ведомый узел — в режиме **асинхронной репликации**.

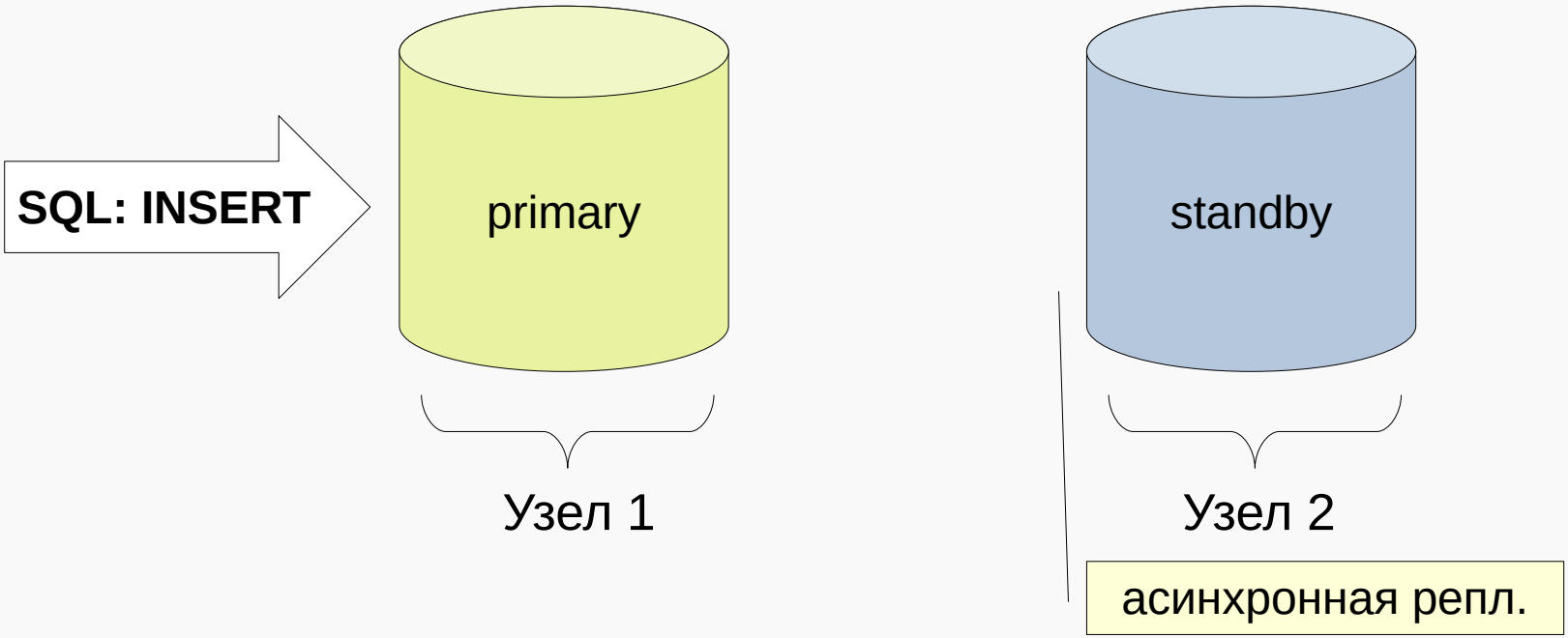
Ведомый узел — **hot standby**, balancer отдает ему **некоторые** запросы на выполнение



Проблема монотонных чтений

1. Кто-то добавил/изменил данные:

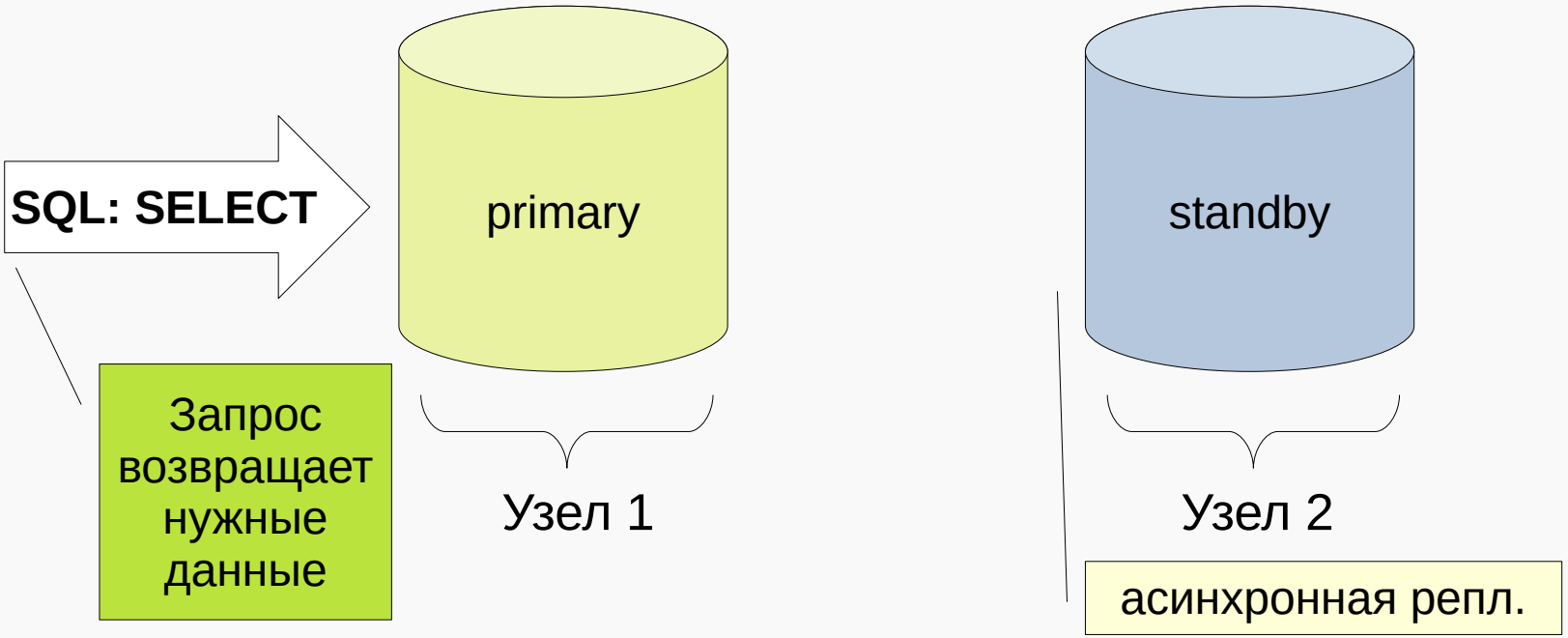
INSERT STUDENT (id) VALUES (4);



Проблема монотонных чтений

2. Пользователь читает данные:

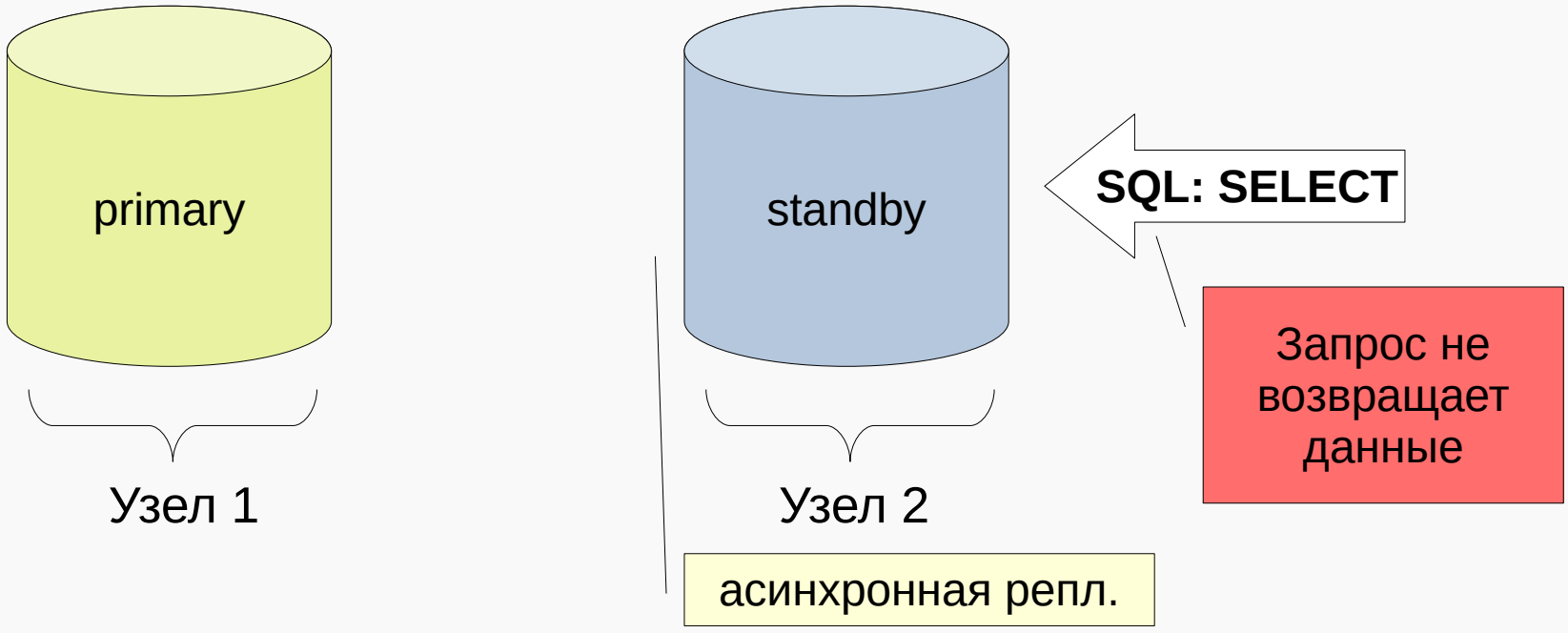
SELECT * FROM STUDENT WHERE id = 4;



Проблема монотонных чтений

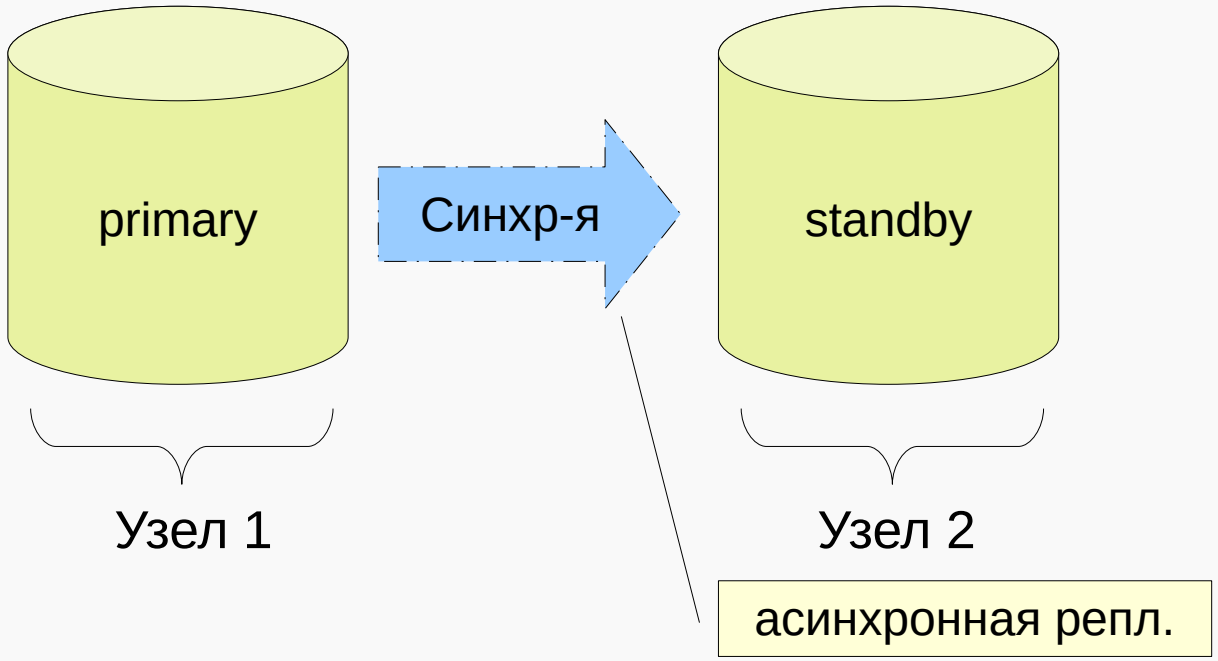
2. Пользователь запрашивает эти же данные еще раз:

SELECT * FROM STUDENT WHERE id = 4;



Проблема монотонных чтений

3. Синхронизация узлов происходит после обработки этих запросов.



ИТМО ВТ

Проблема монотонных чтений

При первом запросе пользователь видит более позднее по времени состояние системы — такого быть не должно:

- лучше, если первый запрос тоже ничего не вернет.

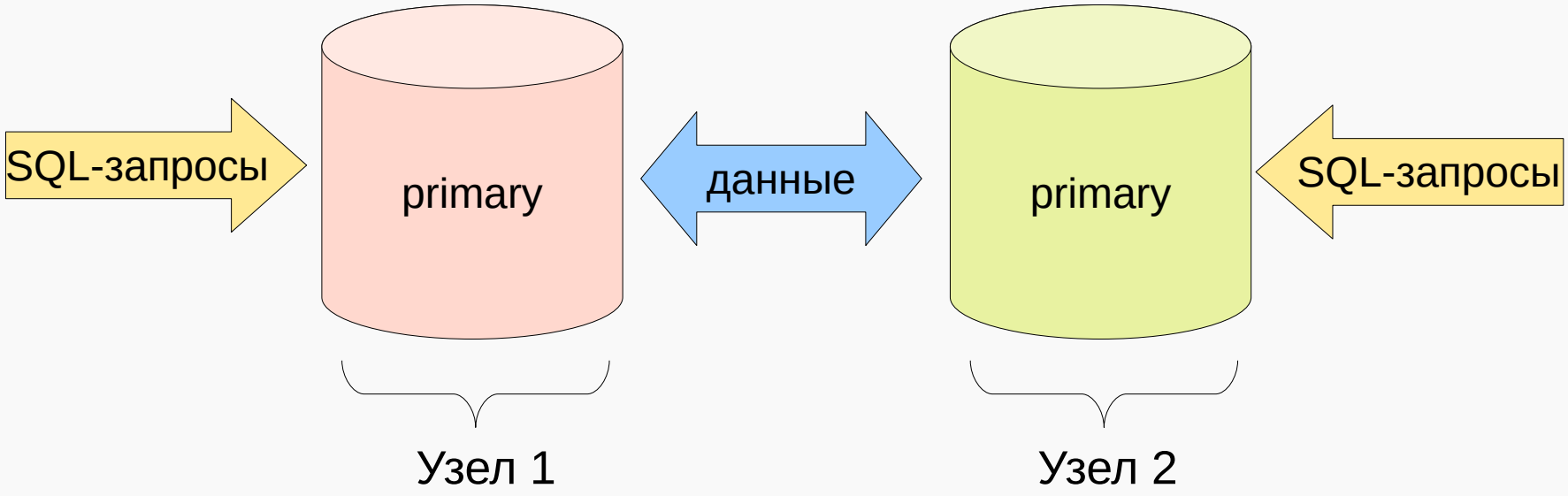
Возможные варианты решения проблемы:

- перенаправлять запросы пользователя в **одну и ту же** реплику.

Master/Master репликация

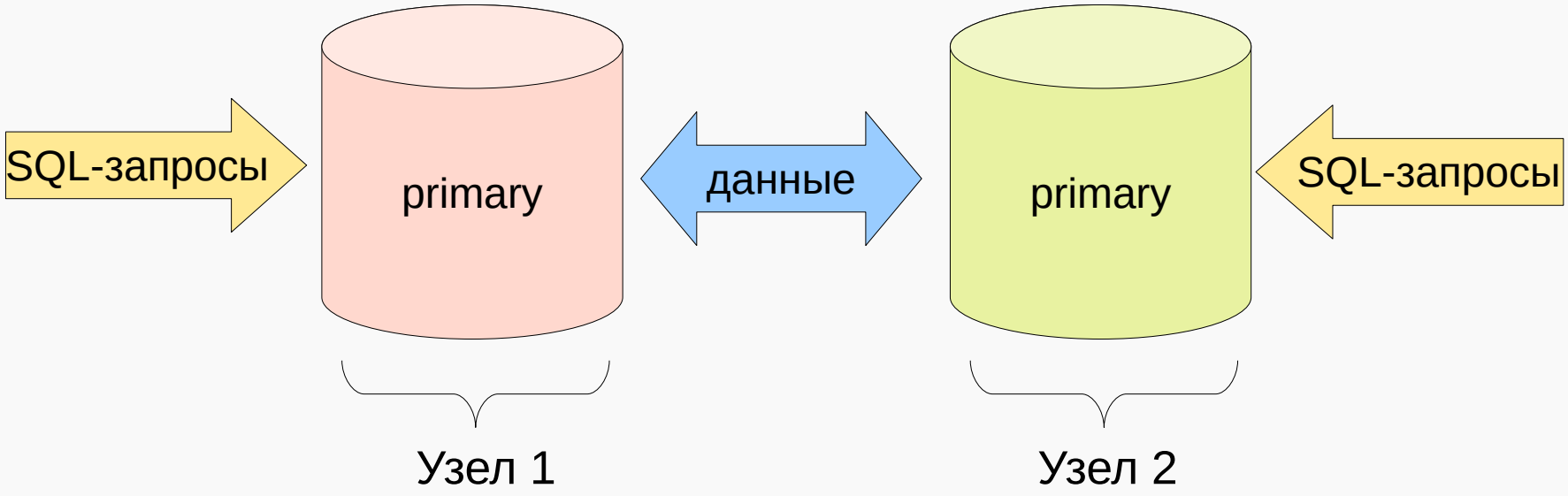
ИТМО ВТ

Master/Master репликация (1)



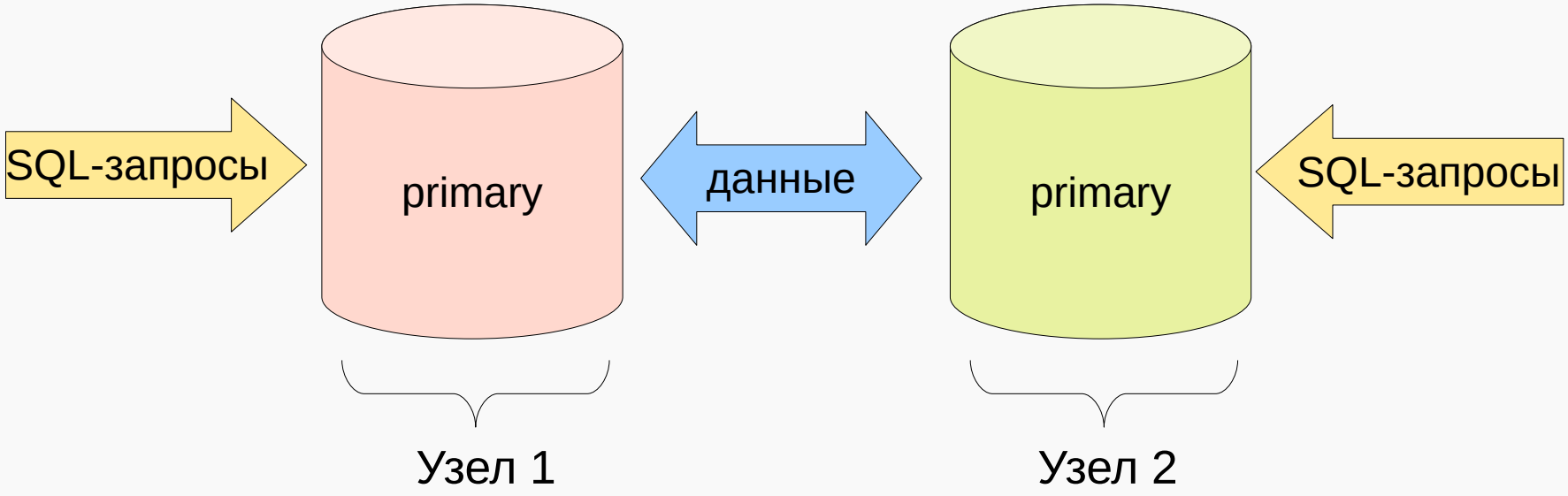
- Мастер принимает все запросы на изменение данных от клиентов.
- Запросы обрабатываются мастером, результаты — записываются локально.
- Данные об изменениях отправляется другим узлам.

Master/Master репликация (2)



- Мастеров может быть много.
- Каждый мастер — зависимый для других мастеров
- Часто M/M репликация реализуется с помощью сторонних инструментов (в MySQL, PostgreSQL).

Проблема Master/Master репликации



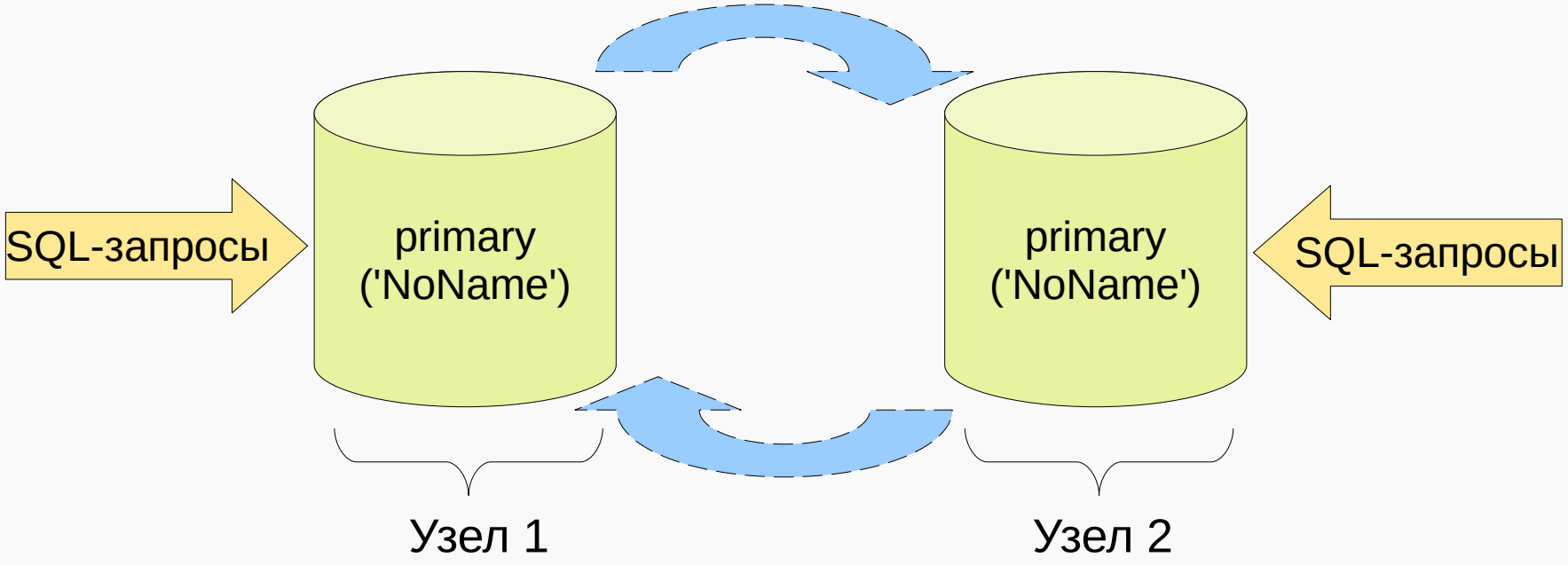
- Так как несколько узлов могут изменять данные — возможно возникновение **конфликтов записи**.

ИТМО ВТ

Конфликты при M/M

Есть M/M кластер, асинхронная репликация.

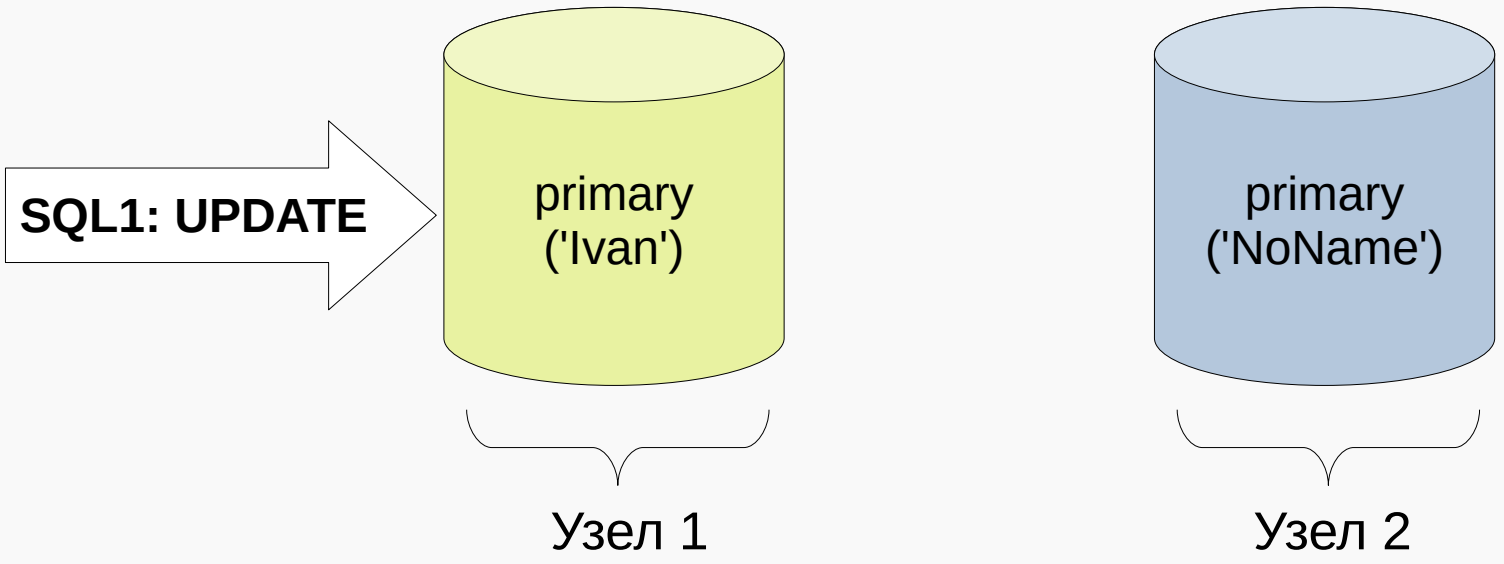
В БД есть студент, у которого имя - 'NoName'



Конфликты при M/M

1. Пользователь1 изменил имя студента:

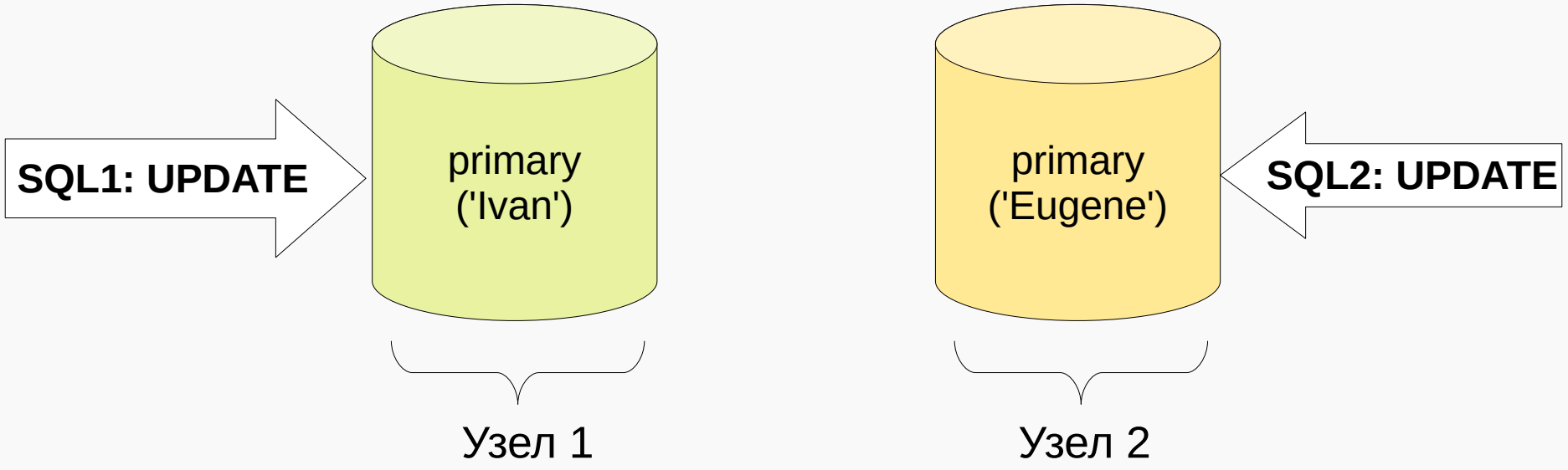
UPDATE STUDENT SET Name = 'Ivan' WHERE id = 4;



Конфликты при M/M

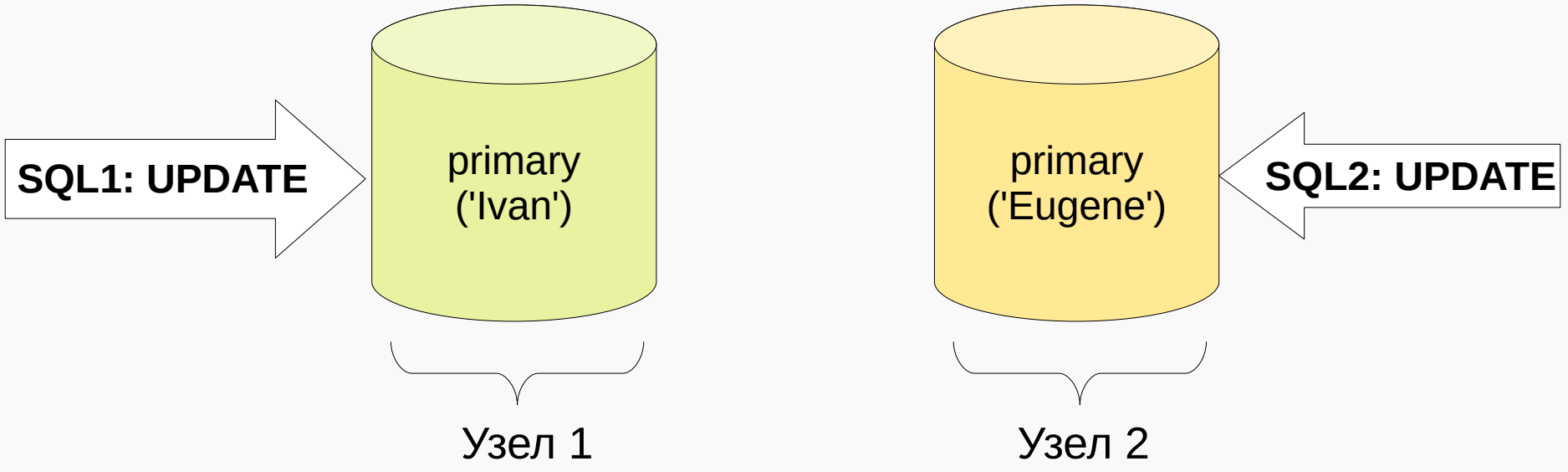
2. В это же время Пользователь2 изменил имя студента:

UPDATE STUDENT SET Name = 'Eugene' WHERE id = 4;



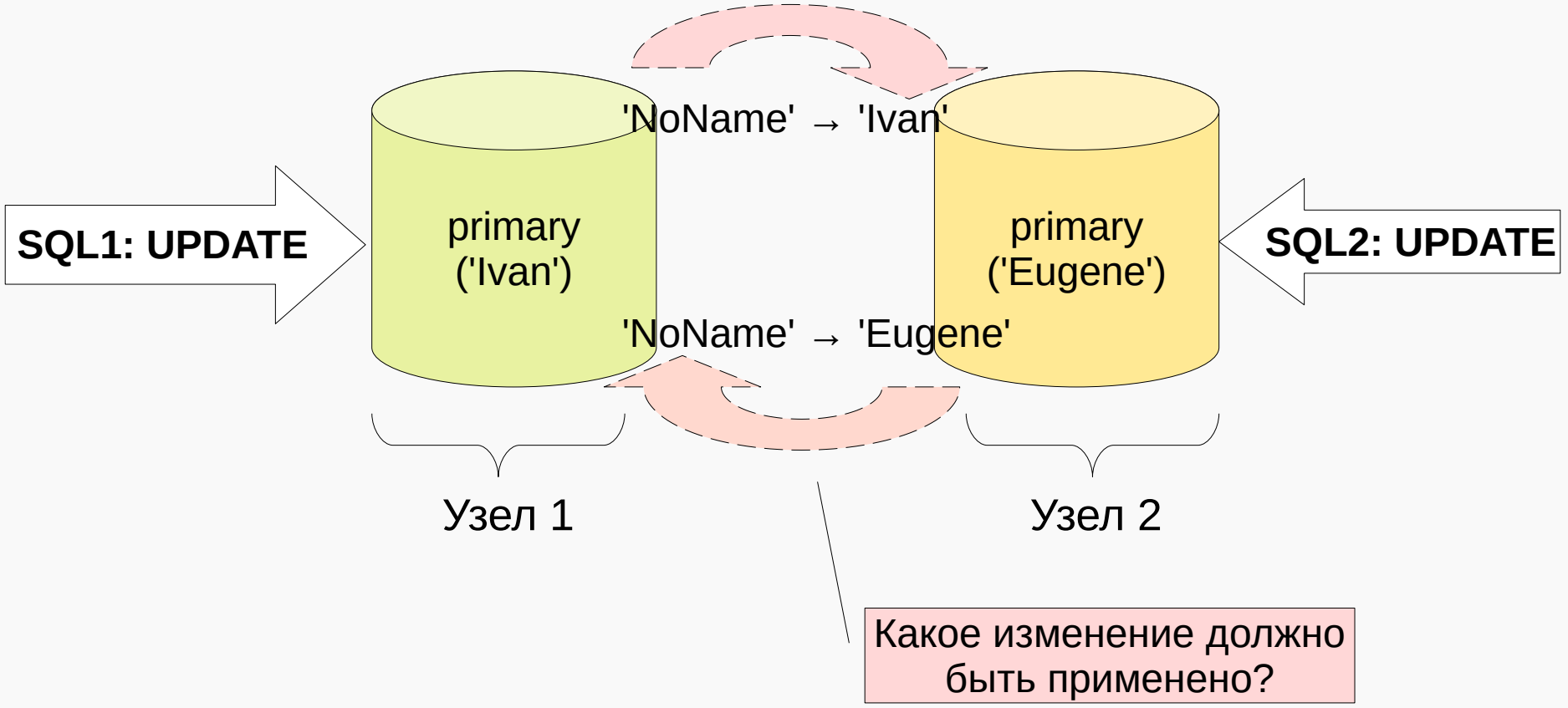
Конфликты при M/M

3. Локально изменения применились успешно на обоих узлах:



Конфликты при M/M

4. Передаваемые изменения не применимы на других узлах.



ИТМО ВТ

M/M: предотвращение конфликтов

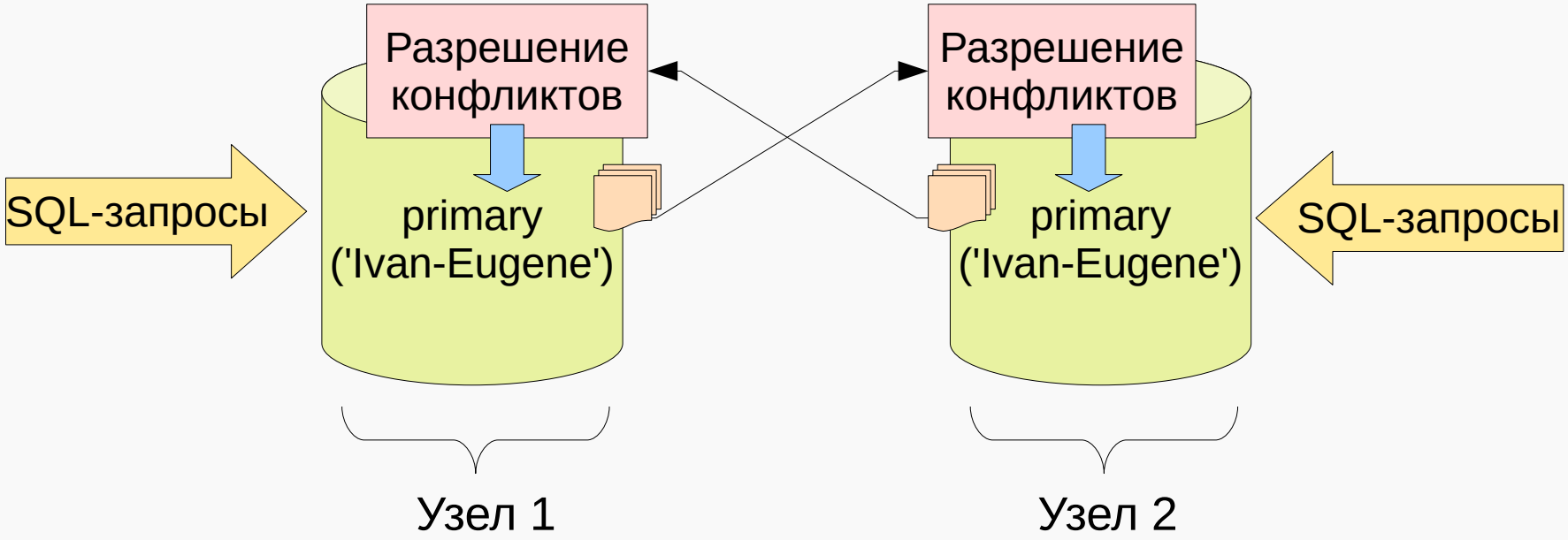
Возможное решение: предотвращать и избегать конфликты:

- однотипные операции проводить через один и тот же узел.

Проблемы:

- сложно разделить операции;
- перераспределение обязанностей при ребалансировке.

Разрешение конфликтов



- Конфликты требуют разрешения.
- Цель: конфликтные данные во всех узлах должны быть сведены к единому значению после выполнения процесса репликации.

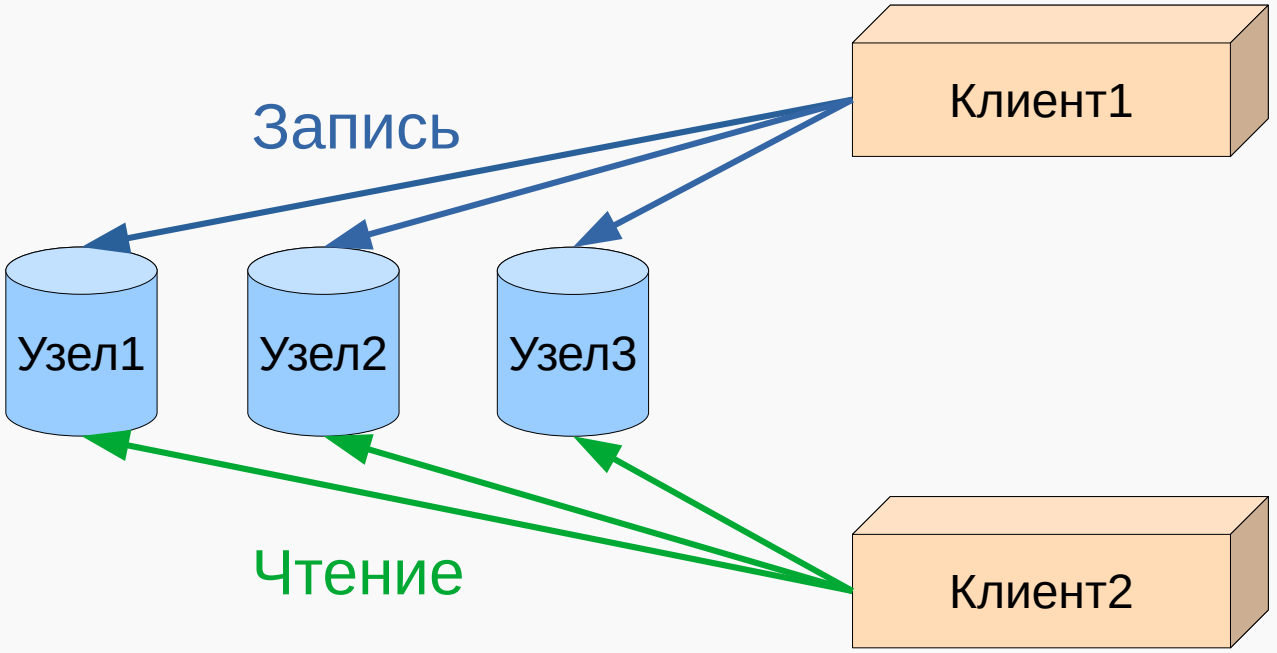
ИТМО ВТ

Способы разрешения конфликтов

- LWW (last write wins) — каждой операции присвоить уникальный идентификатор:
  - при возникновении конфликта использовать его для определения итогового значения;
  - возможна потеря данных.
- Присвоить каждому узлу идентификатор, использовать его при разрешении конфликта:
  - при возникновении конфликта использовать его для определения итогового значения;
  - возможна потеря данных.
- Конкатенация значений.
- Использовать вспомогательную структуру данных для хранения конфликтующих значений, связанных с некоторым объктом.

Репликация без ведущего узла

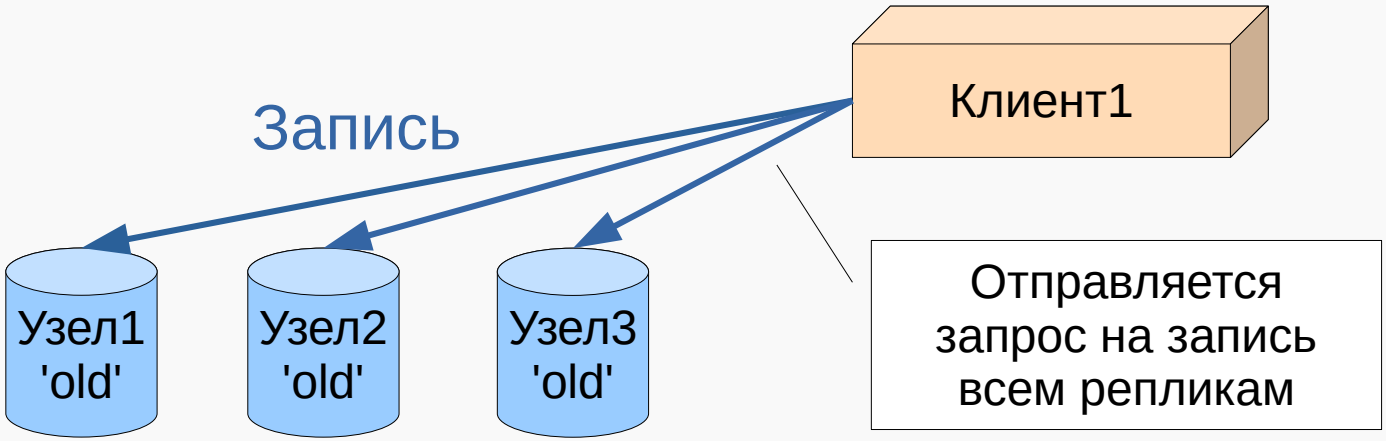
Репликация без ведущего узла



- Операции чтения и записи поступают на все реплики, **нет ведущего узла**.
- Dynamo — style database: Cassandra, Riak

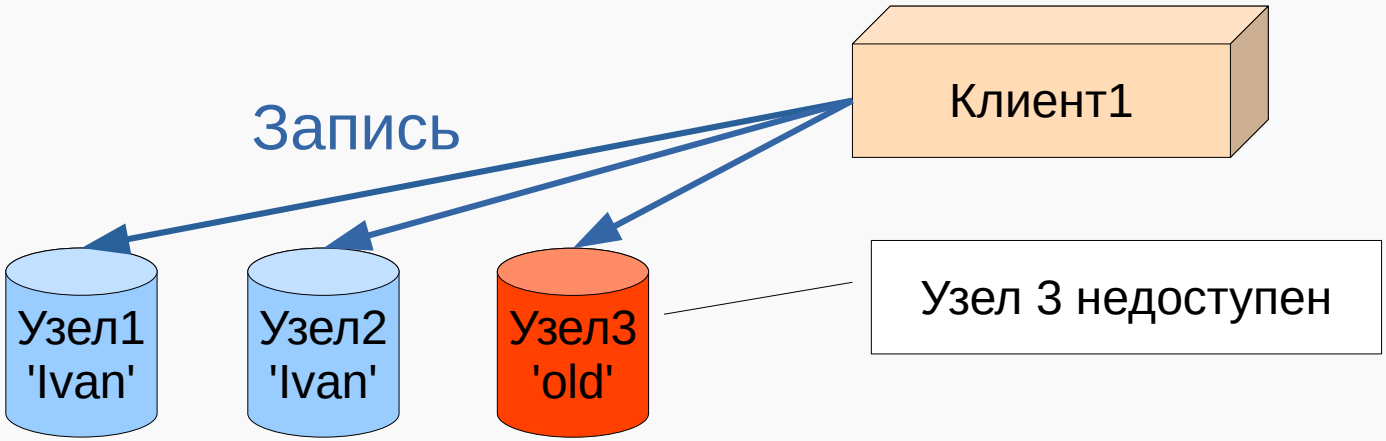
Репликация без ведущего узла: запись

- Клиент1 хочет установить имя в БД:
  stud1.stud_name = 'Ivan'



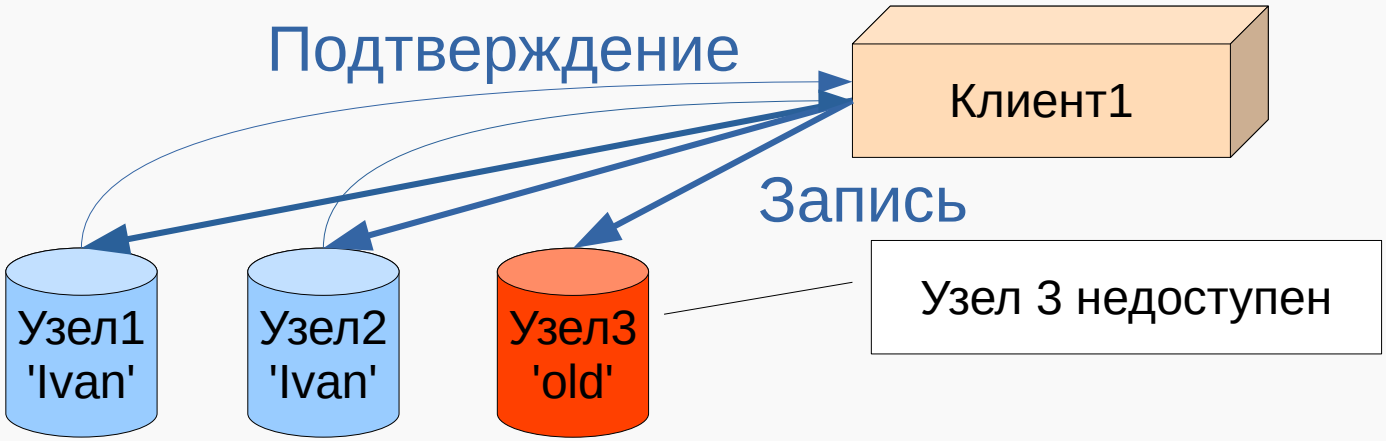
Репликация без ведущего узла: запись

- Клиент1 хочет записать имя в БД:
  stud1.stud_name = 'Ivan'



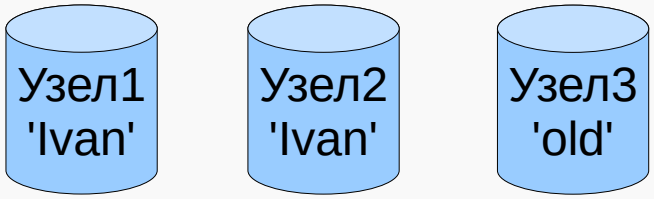
Репликация без ведущего узла: запись

- Узел1 и Узел2 подтвердили запись — операция успешна.



Репликация без ведущего узла

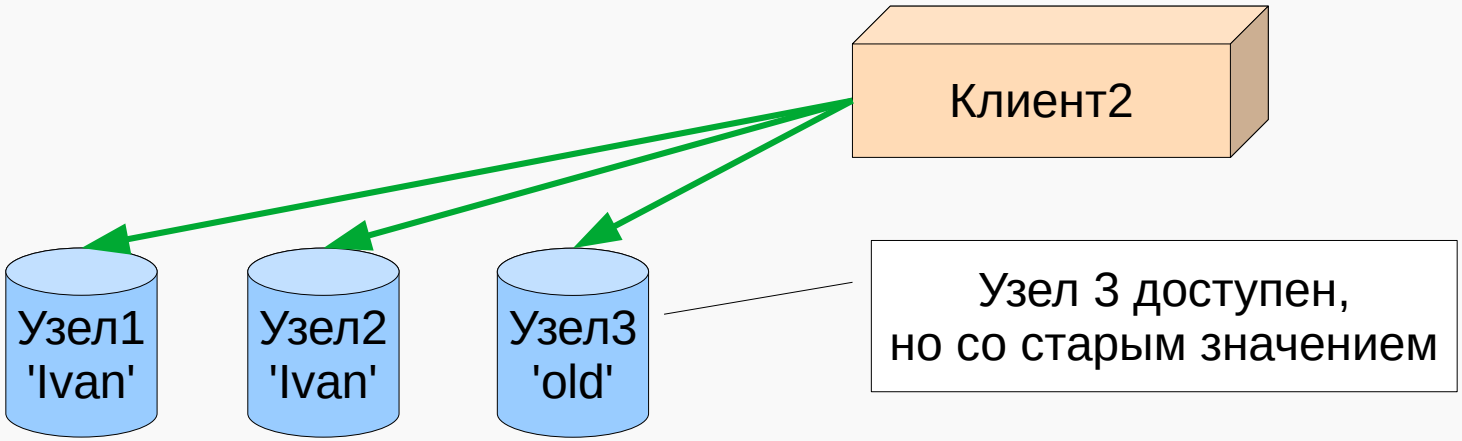
- Узел3 возвращаем в работоспособное состояние



Репликация без ведущего узла: чтение

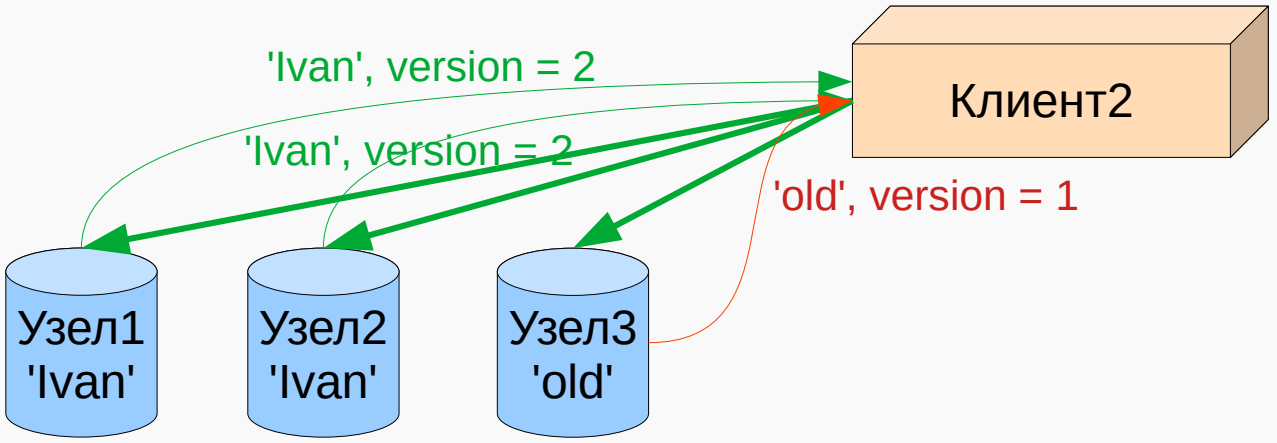
- Клиент2 хочет прочитать данные:

  get stud1.stud_name
- Запрос на чтение отправляется всем* узлам параллельно.



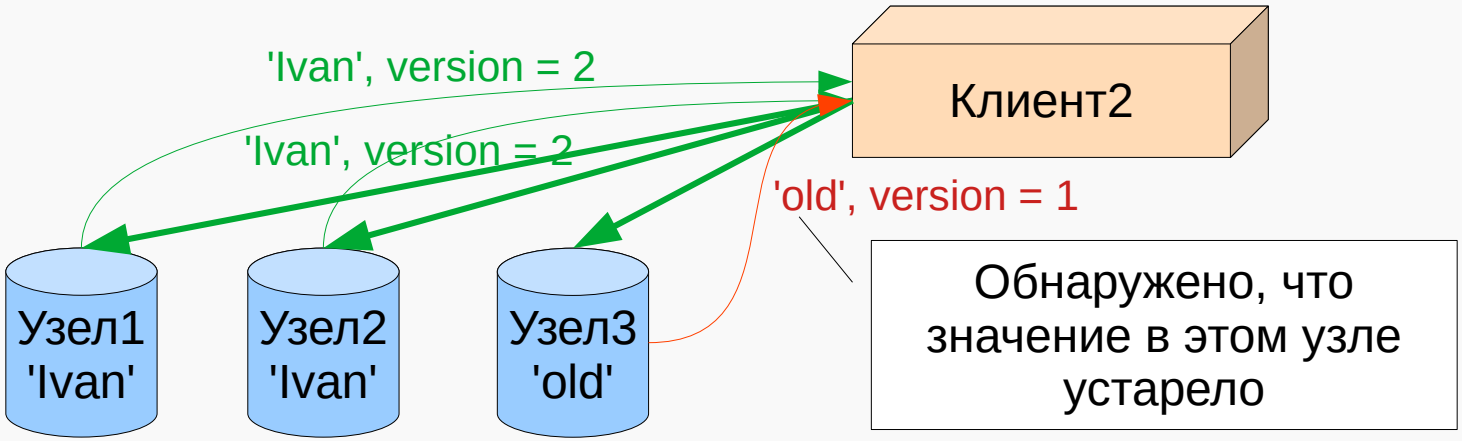
Репликация без ведущего узла: чтение

- Ответ приходит от всех узлов.
- Для определения корректного значения используется версия.



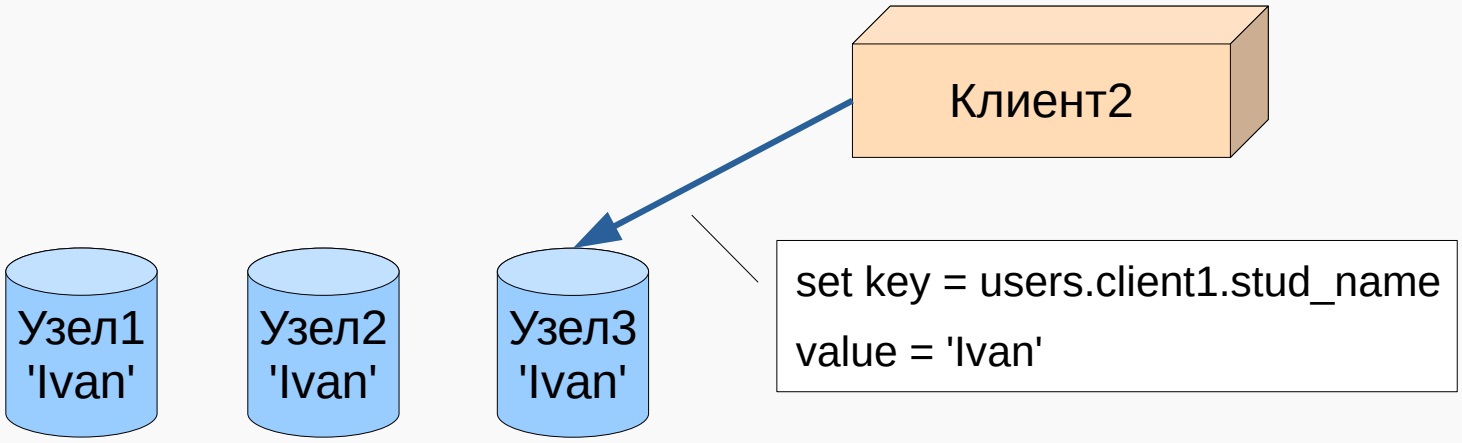
Репликация без ведущего узла: чтение

- При чтении клиент обнаруживает устаревшие ответы, значения в этих узлах должны быть переписаны.



Репликация без ведущего узла: чтение

- Клиент обновляет данные в узле3, таким образом распространяя актуальные данные по кластеру:



ИТМО ВТ

Разрешение конфликтов

- **При чтении**:

  клиент обнаруживает устаревшие ответы при чтении и отправляет запросы на их обновление.

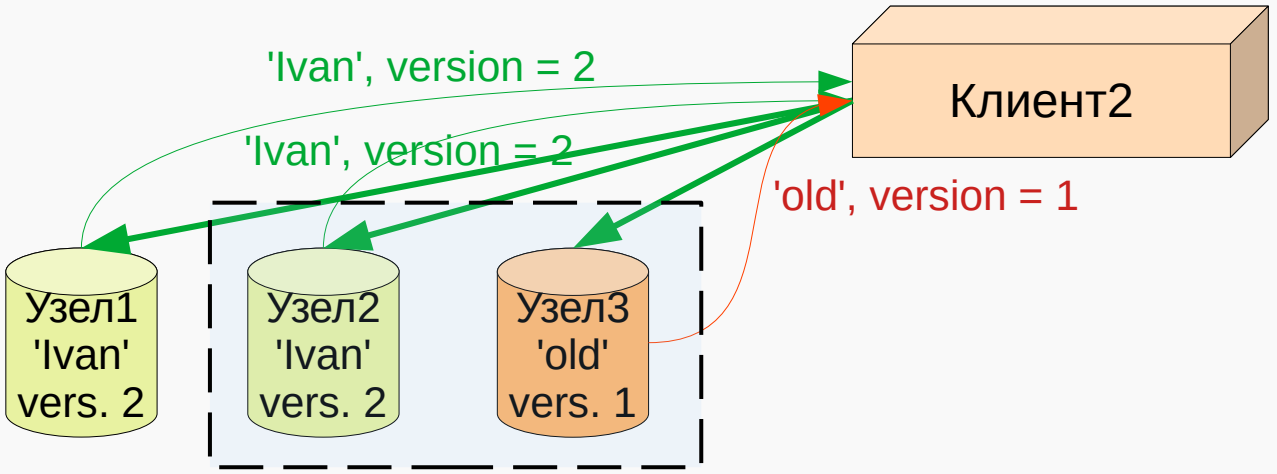
- **Процесс противодействие энтропии**:

  фоновый процесс, который ищет устаревшие/отсутствующие данные в узлах и распространяет изменения.

ИТМО ВТ

Есть ли гарантия?

- Если известно, что запись произведена на 2 узла из 3х → максимум одна устаревшая.
- В таком случае: при чтении данных (даже из 2х узлов) — один узел содержит актуальные данные:



ИТМО ВТ

Условие кворума

Чтобы результат операции чтения был актуальным должно выполняться соотношение:

$$W + R > N$$

- N — число узлов
- операция записи должна подтверждаться W узлами
- операция чтения читать R узлов.

ИТМО ВТ

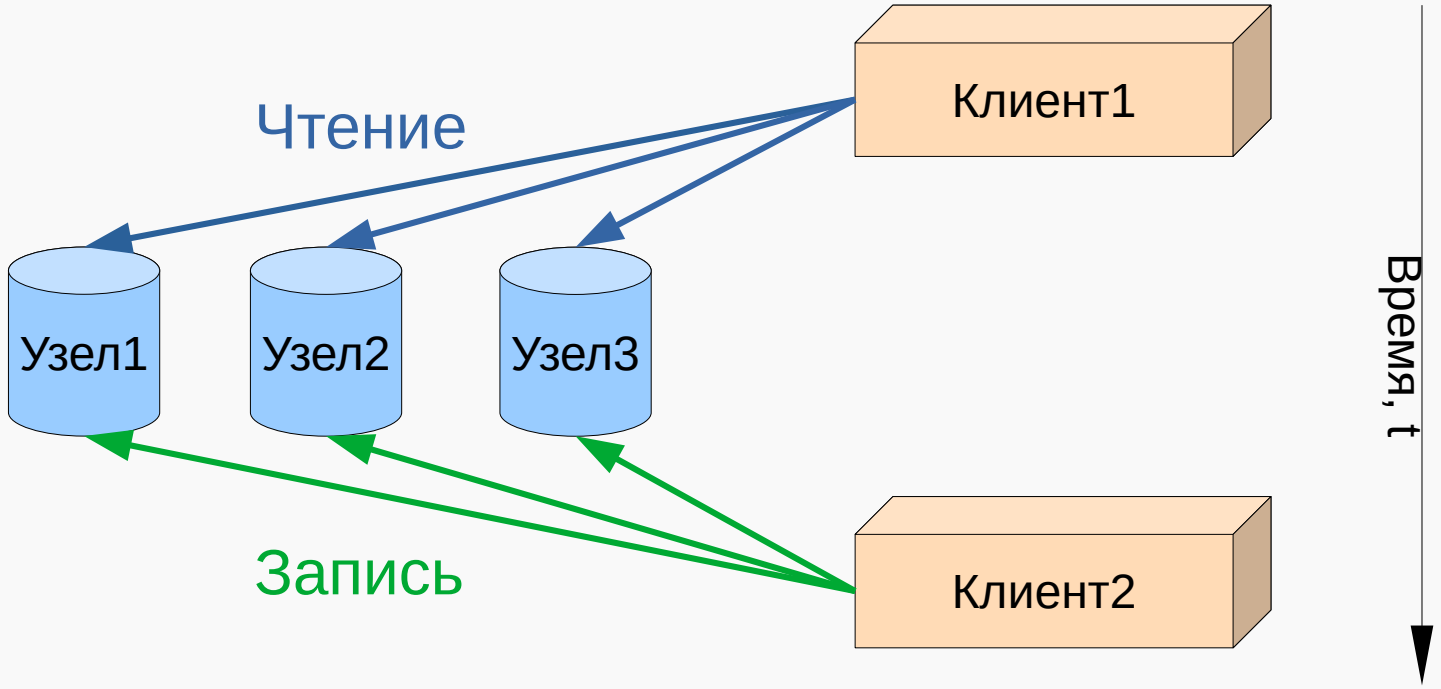
Условие кворума (1)

Условие кворума дает системе **устойчивость к недоступности узлов**

Если выполняется соотношение: $W + R > N$

- При $W < N$ можно продолжать запись, если не все узлы доступны.
- При $R < N$ можно продолжать чтение, если не все узлы доступны.
- W, R — определяют число узлов, от которых мы должны получить ответ, чтобы можно было считать операцию успешной.

Условие кворума (2)



$W + R > N$

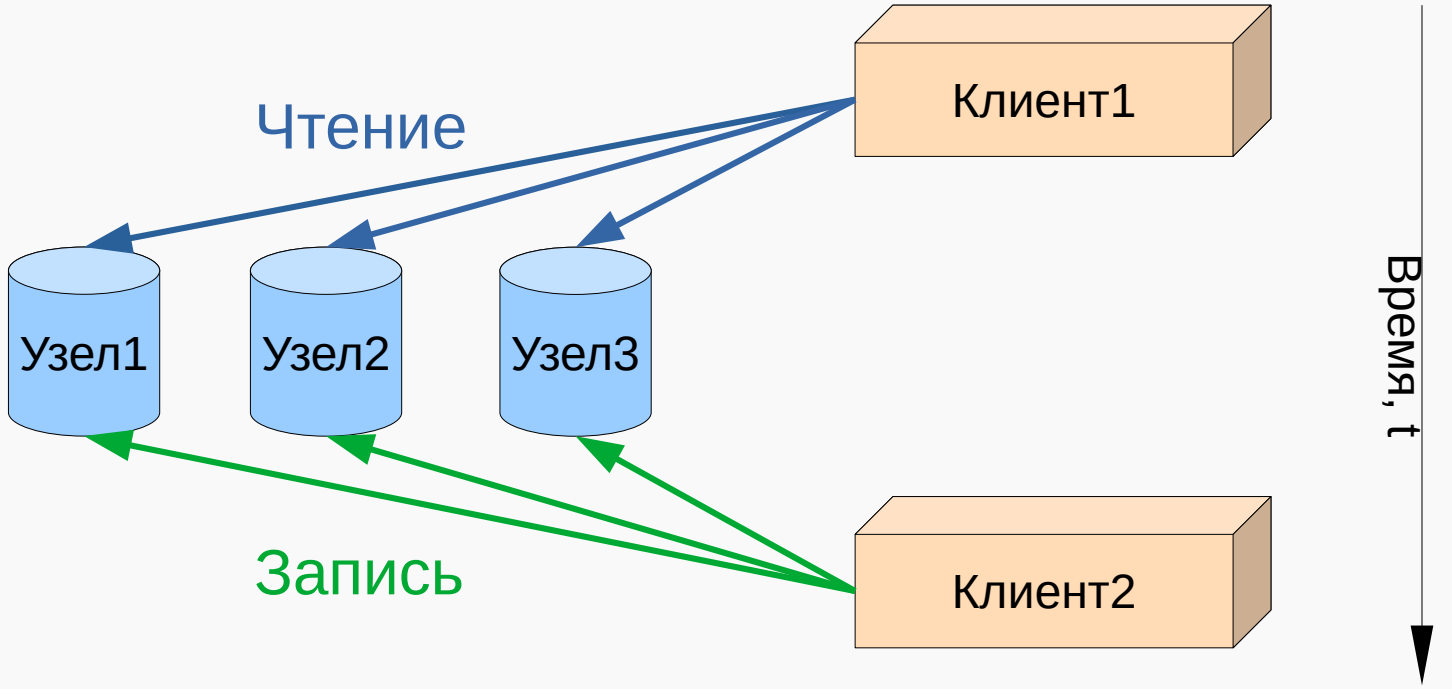
W, R — задаются при настройке хранилища,

W — узлов должны подтвердить успешность записи

R — узлов должны подтвердить успешность чтения

ИТМО ВТ

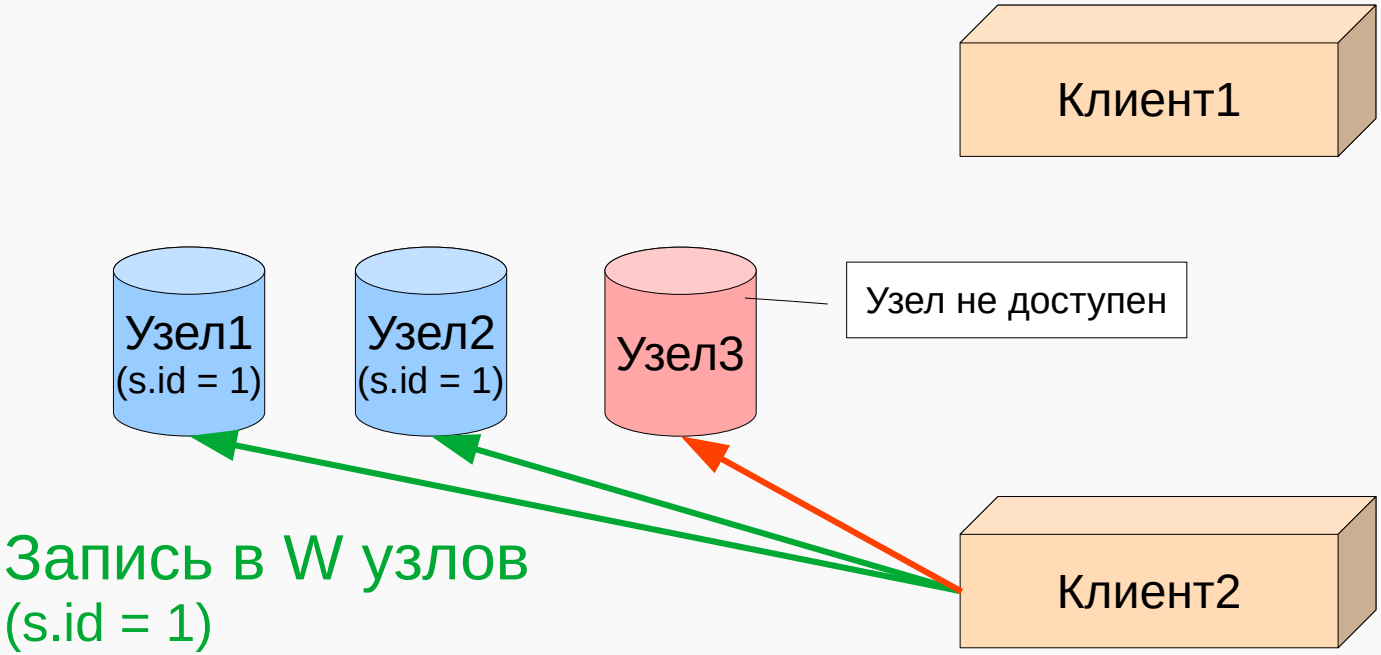
Условие кворума (3)



$W + R > N$

Если пришло меньше подтверждений, чем W — для записи, R — для чтения, операция не успешна.

Условие кворума (4)



Пример:

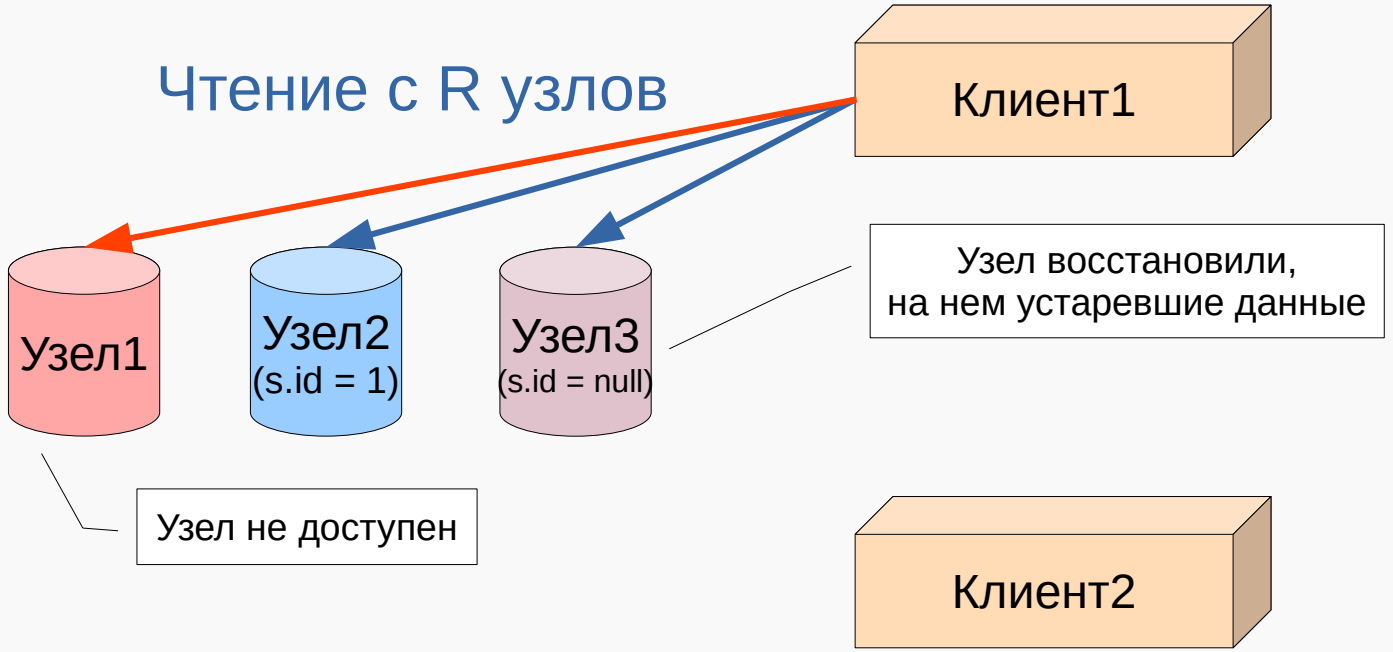
W = 2

R = 2

N = 3

Может быть 1 недоступный узел

Условие кворума (5)



Пример:

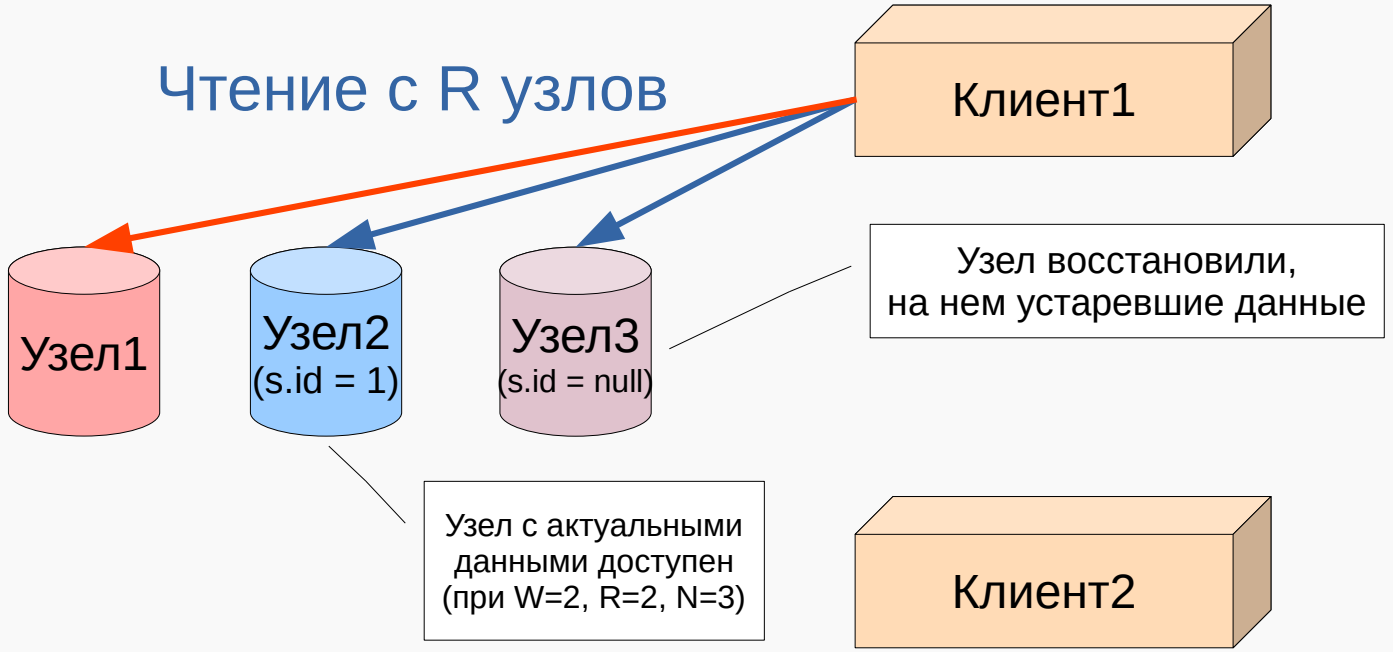
W = 2

R = 2

N = 3

Может быть 1 недоступный узел

Условие кворума (6)



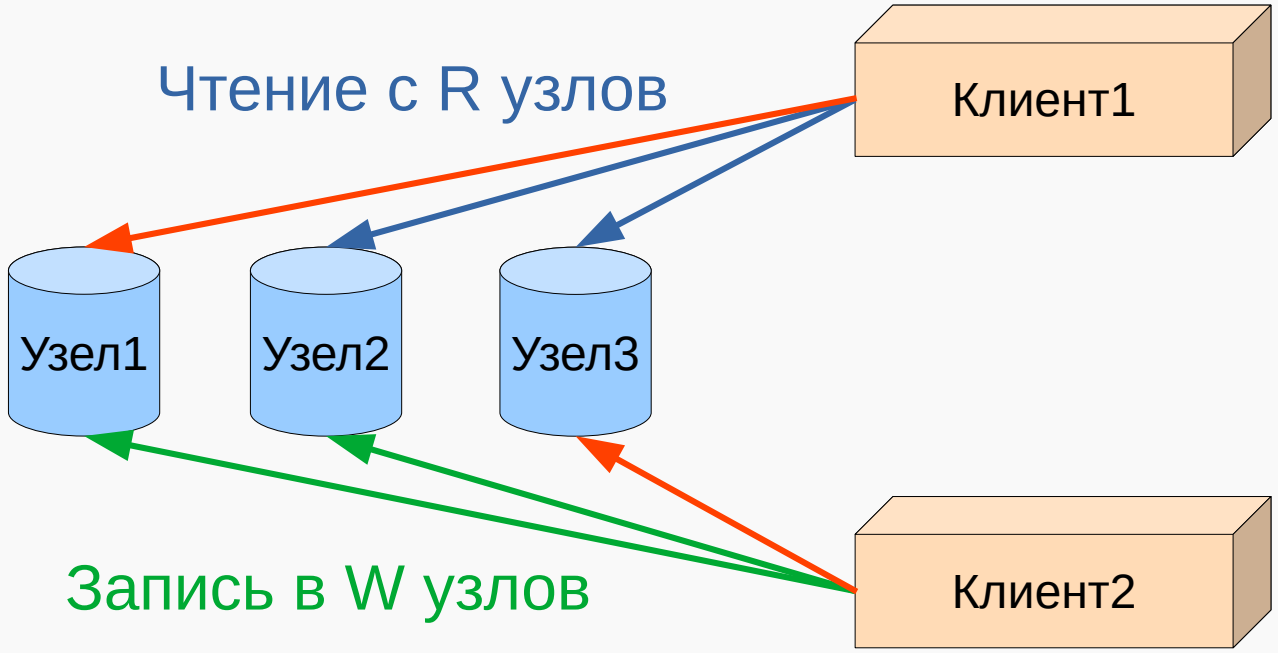
Пример:

W = 2

R = 2

N = 3

Условие кворума (7)



Пример:

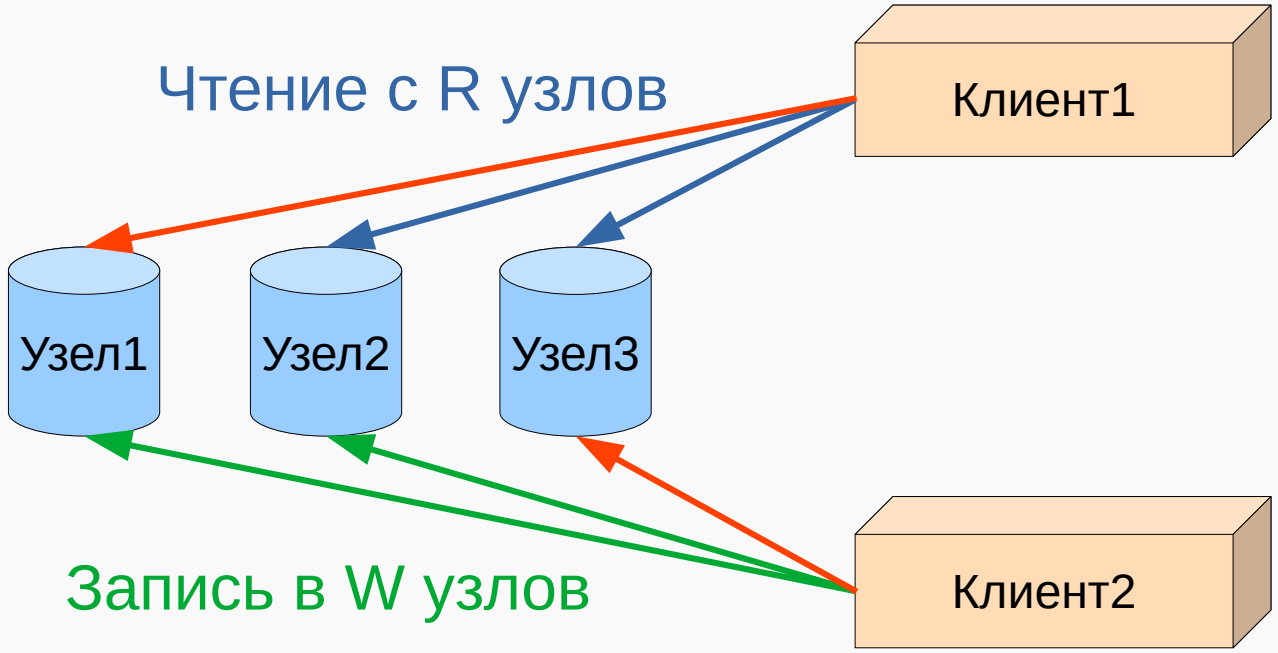
W = 2

R = 2

N = 3

Может быть 1 недоступный узел

Условие кворума (8)

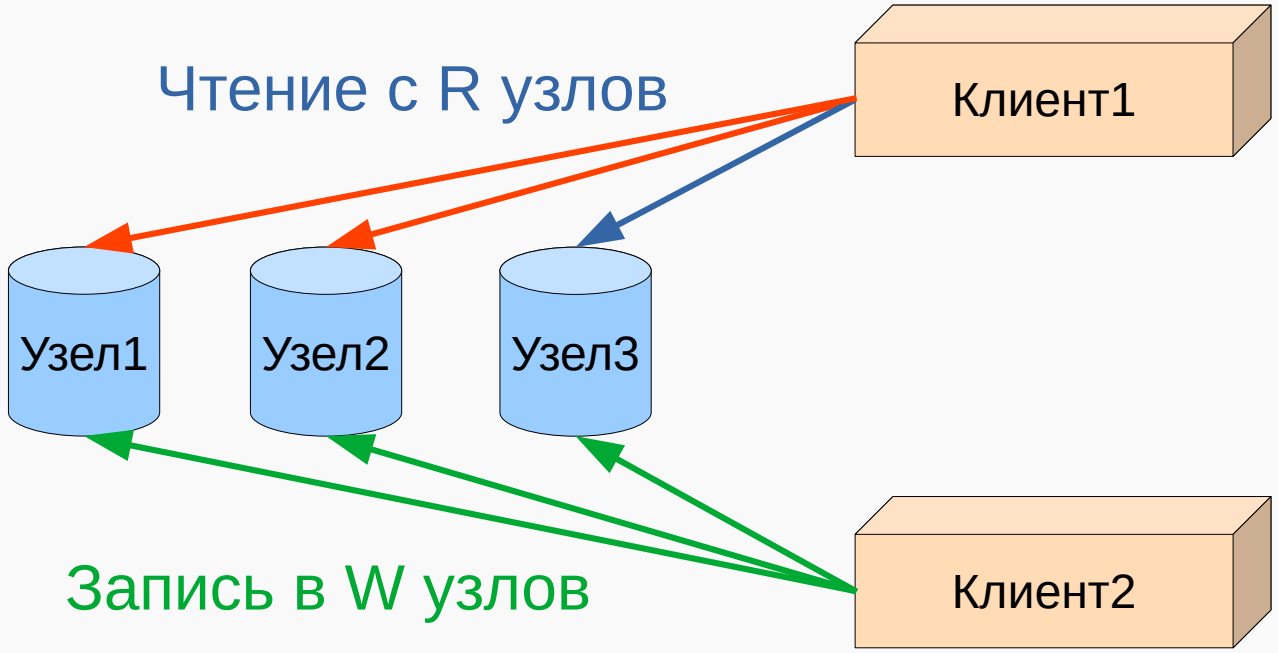


Часто W, R устанавливают следующим образом:

W = CEIL((N + 1)/2)

R = CEIL((N + 1)/2)

Условие кворума (9)



Пример:

W = 3

R = 1

N = 3

При таком раскладе для чтения достаточно одного доступного узла

Несоблюдение кворума. Как действовать?

ИТМО ВТ

W + R > N

Задано: W=2, R=2. Данные **прочитаны с 1 узла**.

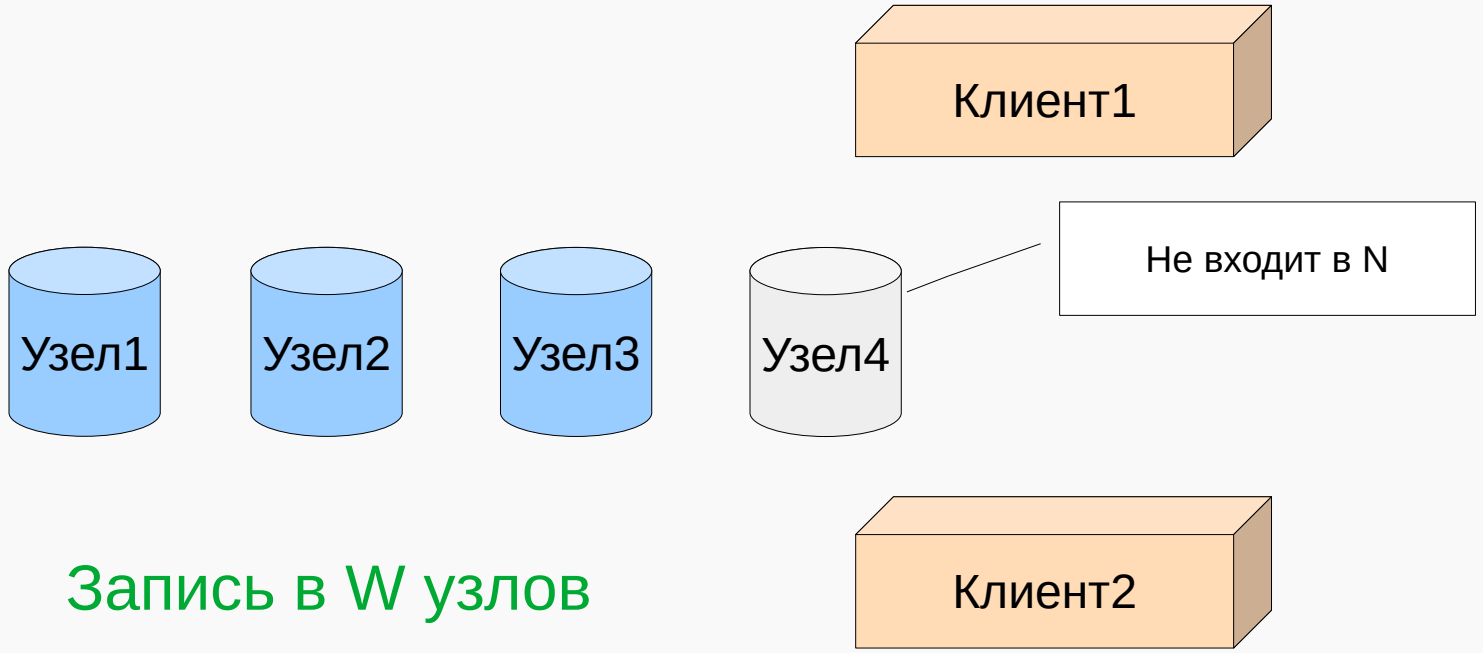
Ситуация: в кластере много узлов не доступно, кворум нельзя обеспечить.

Что делать (стратегии)?

- Вернуть ошибку при попытке совершить операцию
- Как-то продолжить исполнение (нестрогий кворум).

Нестрогий кворум

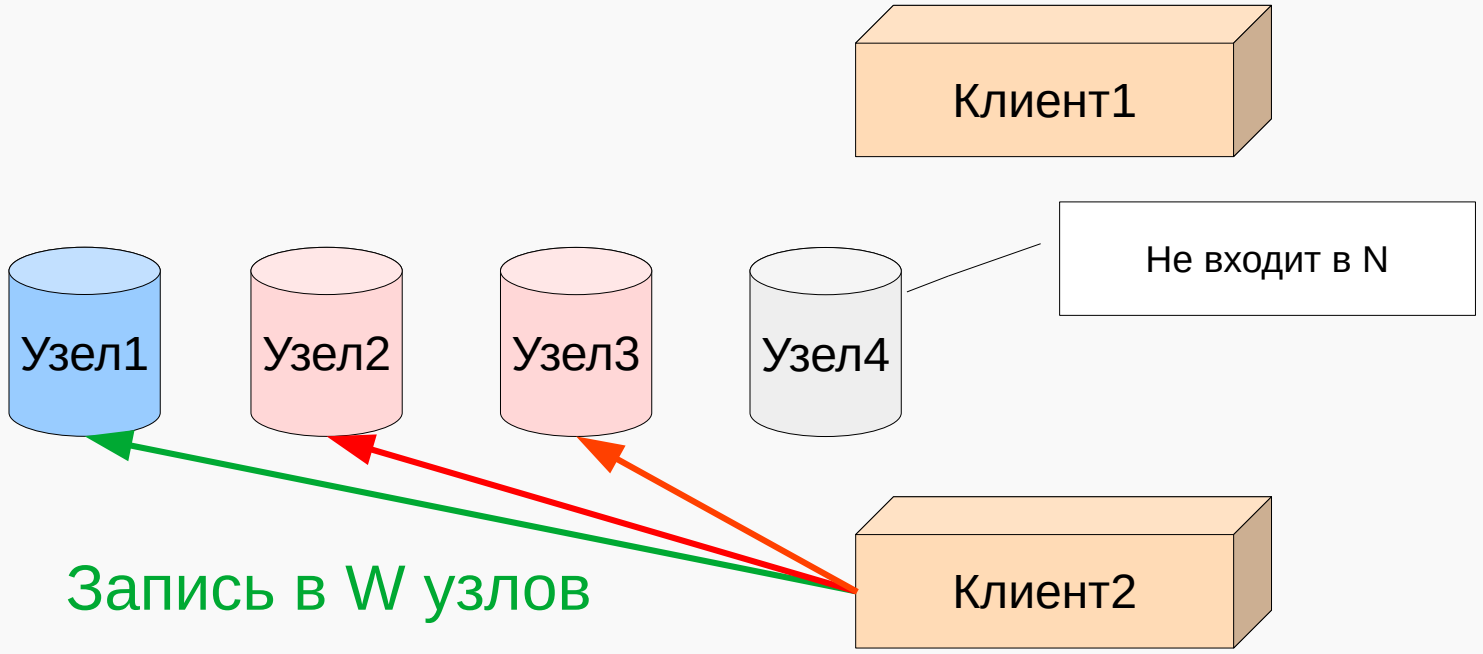
Пример: W=2, R=2, N=3



Запись в W узлов

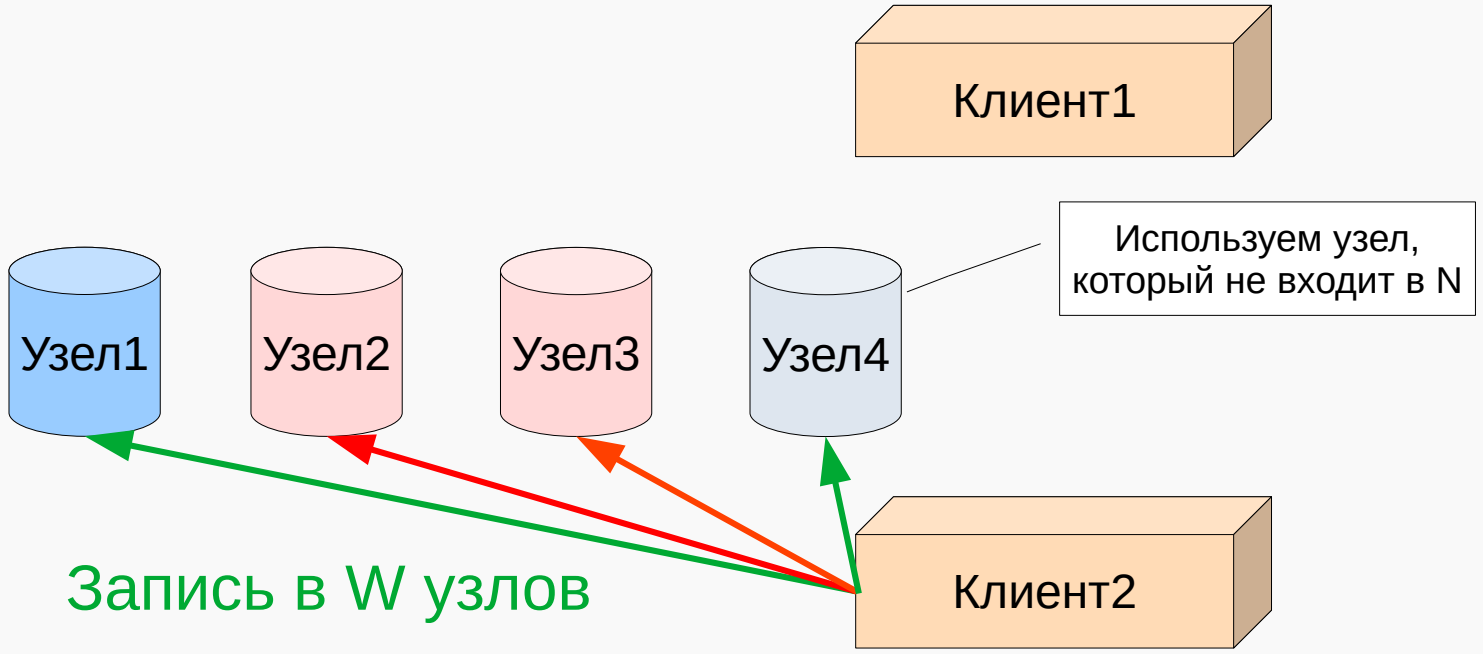
Нестрогий кворум

Пример: W=2, R=2. Данные можно записать в **1 узел**.



Нестрогий кворум

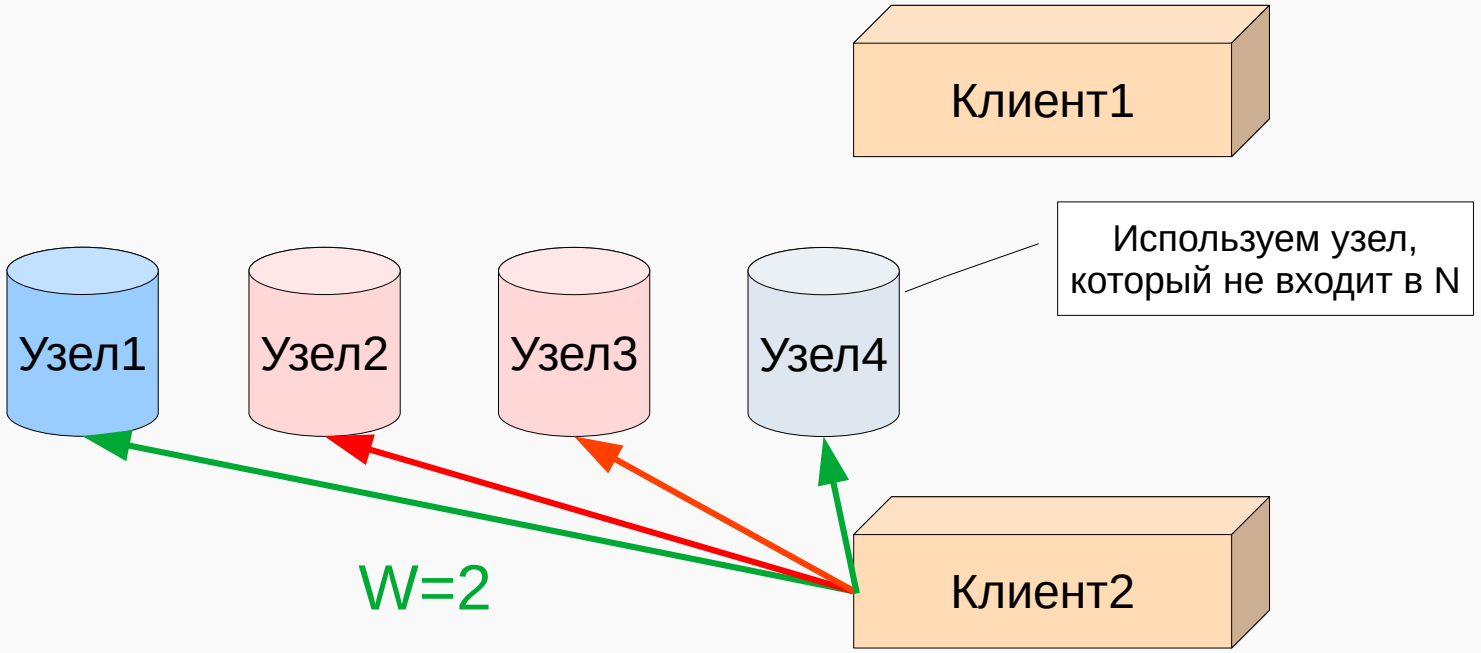
Пример: W=2, R=2. Используем Узел4.



ИТМО ВТ

Нестрогий кворум

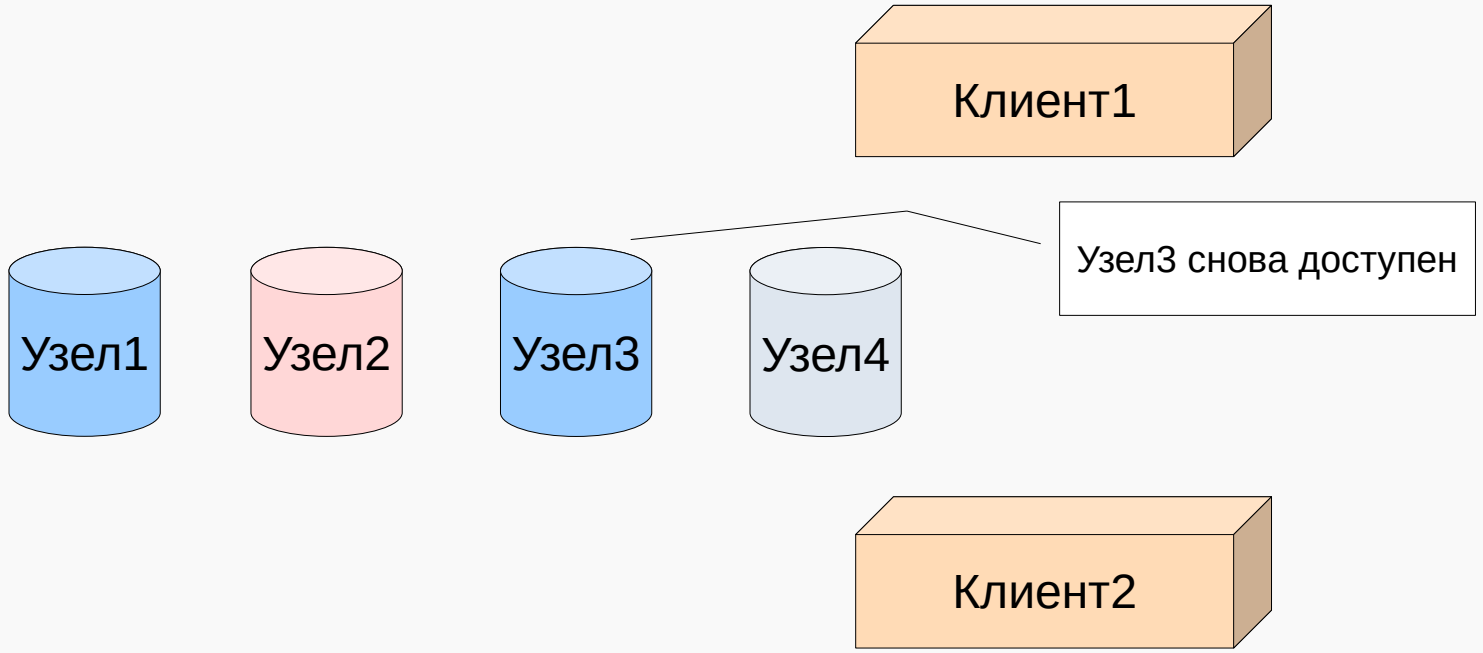
Пример: W=2, R=2. Условия выполнены



ИТМО ВТ

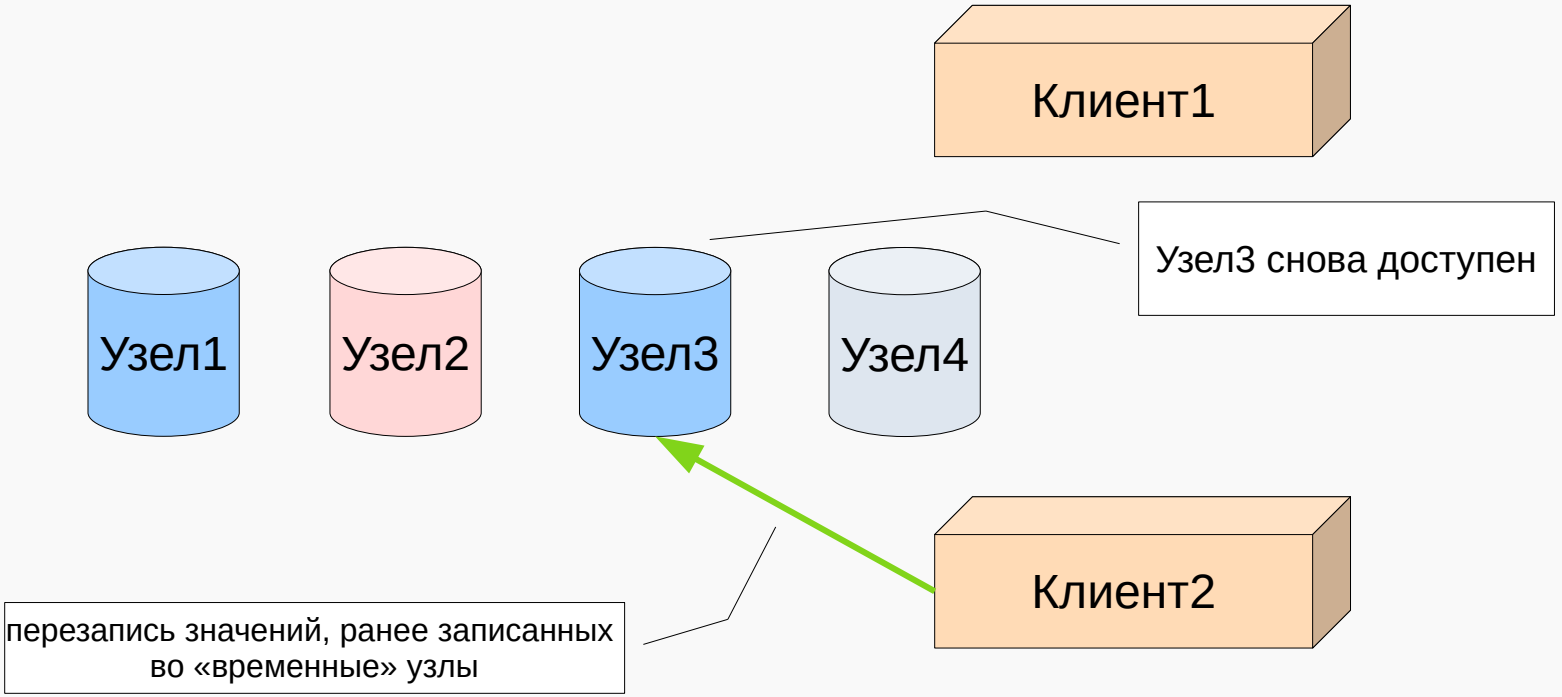
Нестрогий кворум

Пример: W=2, R=2.



Нестрогий кворум

Пример: W=2, R=2. Используем Узел3



Шардирование данных

ИТМО ВТ

Шардирование

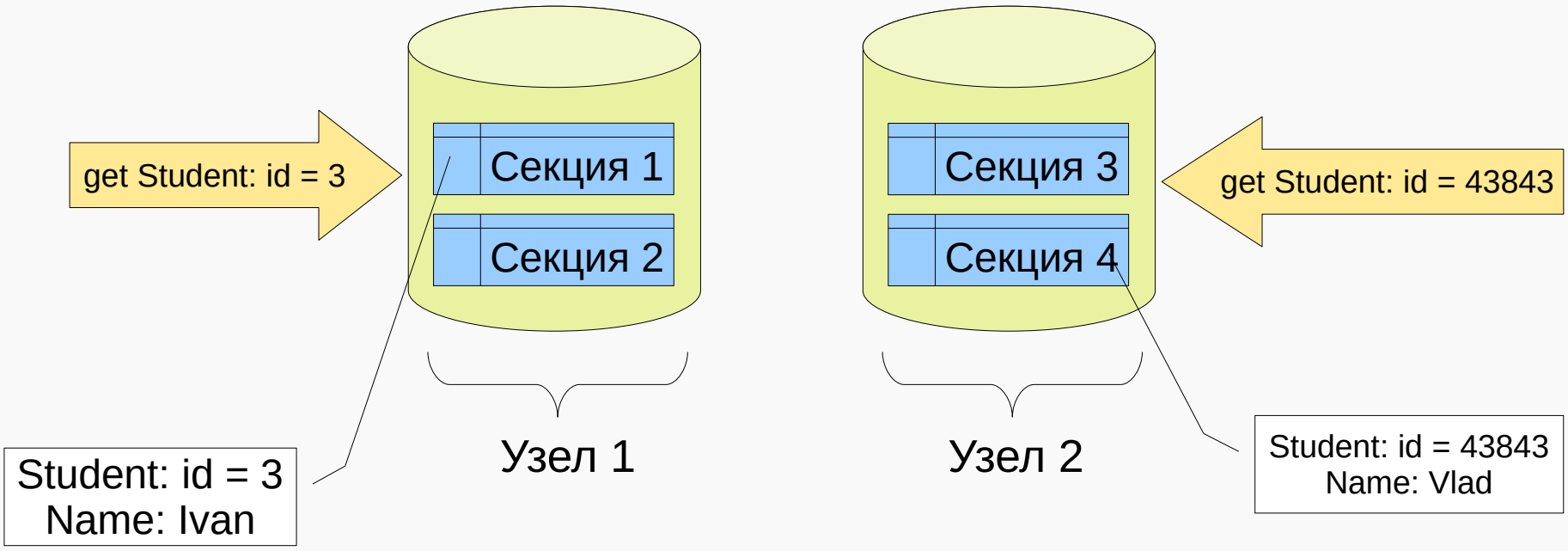
- Шардирование — способ организации распределенных данных.
- При шардировании происходит разбиение данных на части (секции, шарды).
- Секции хранятся на разных узлах.

- Базовые понятия:
  - **Узлы** — отдельные серверы БД
  - У каждого узла — своя БД с секциями.
  - Секция — регион, шард. Шардирование — секционирование.

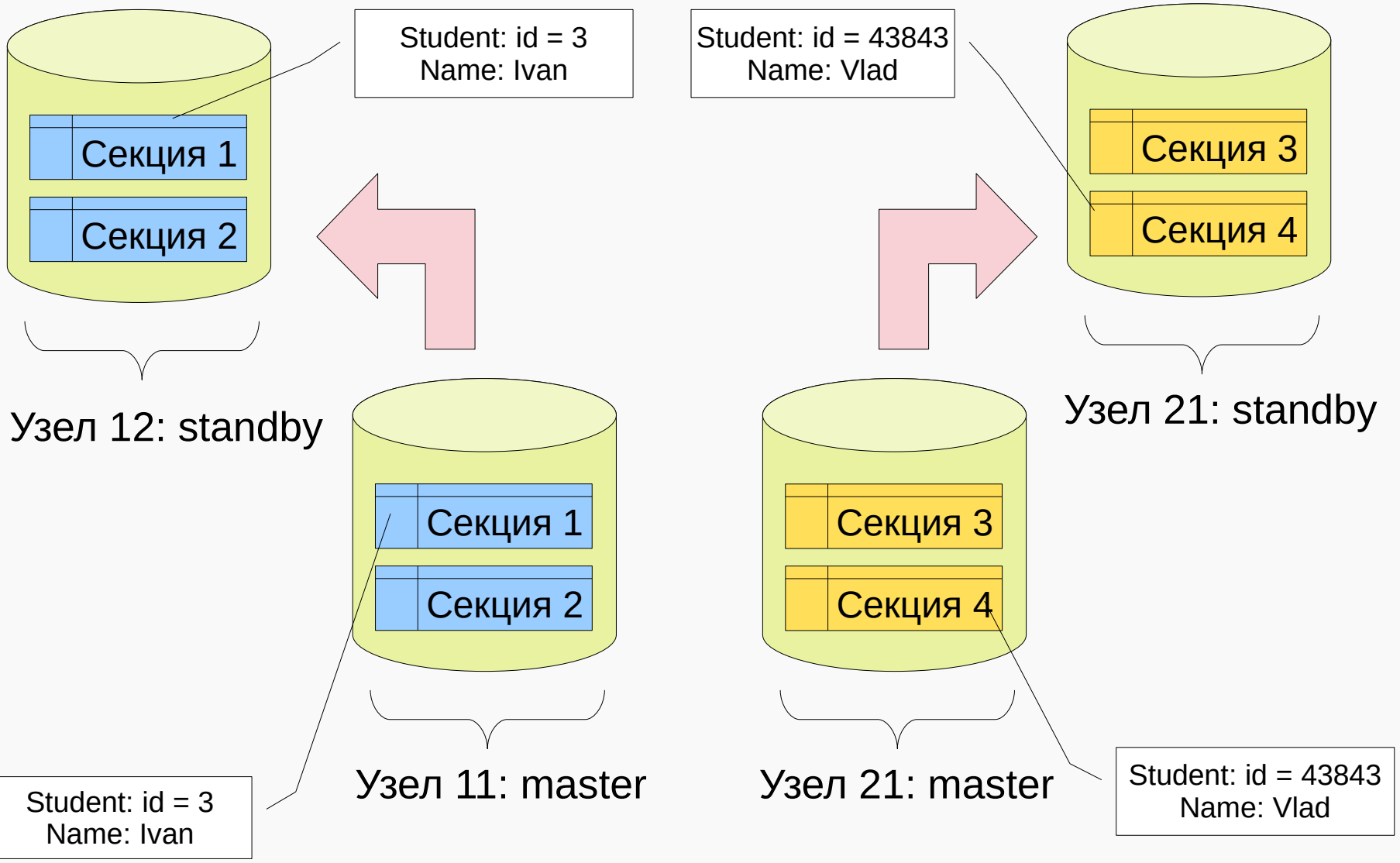
ИТМО ВТ

Шардирование

- При шардировании происходит разбиение данных на части (секции).
- Один элемент данных хранится в одной секции

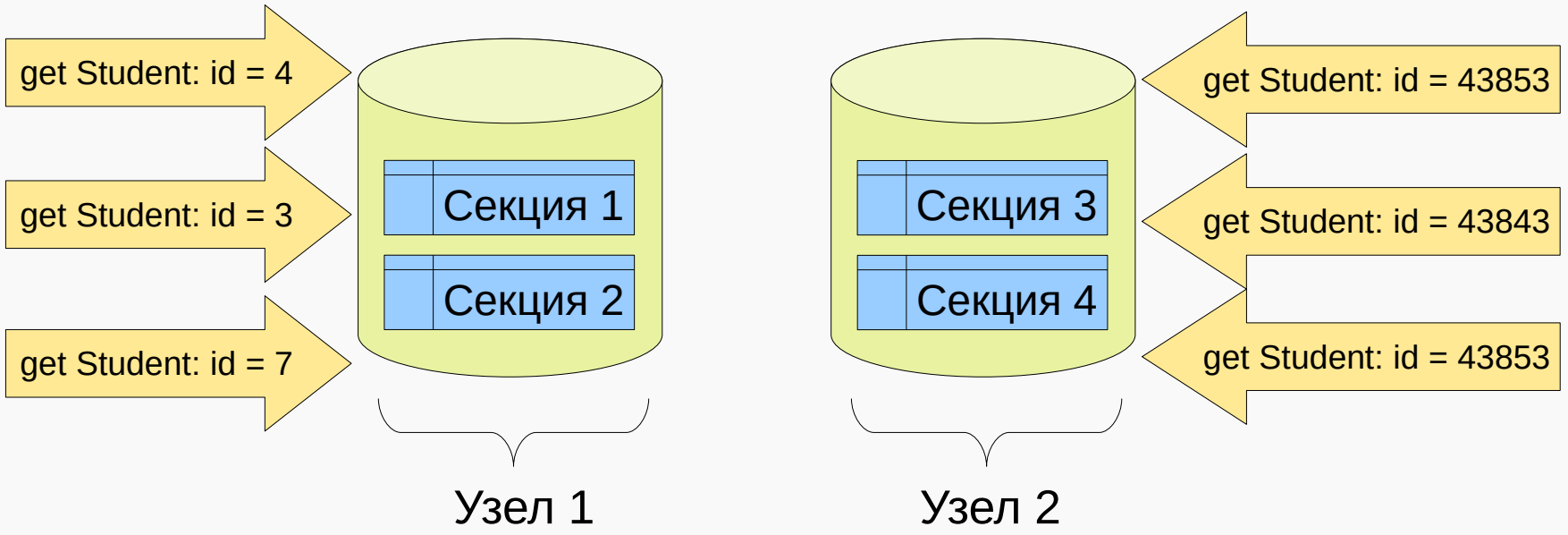


Шардирование + Репликация



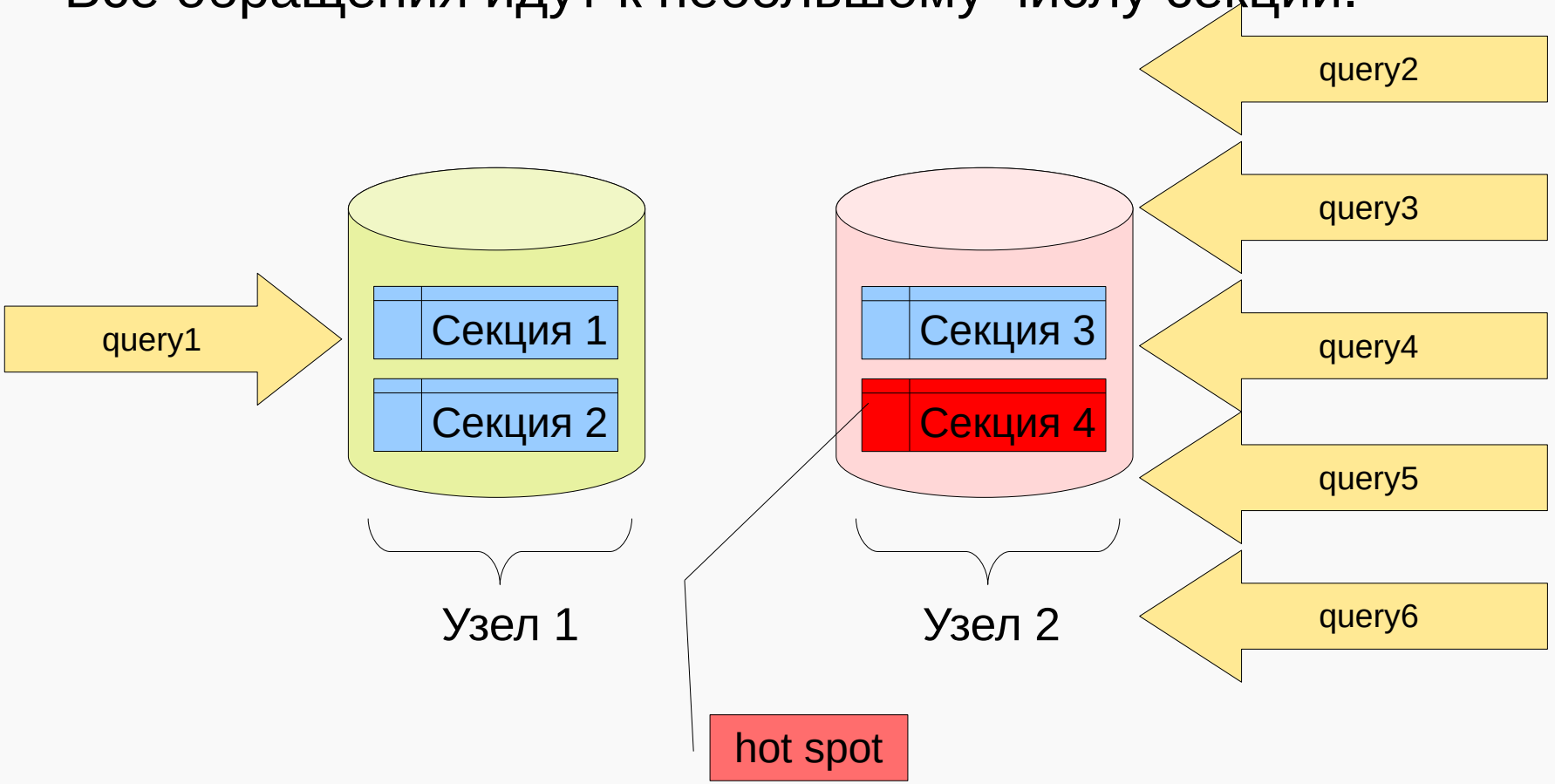
Шардирование

- Нагрузка на узлы должна распределяться равномерно:



Асимметричное шардирование

- Все обращения идут к небольшому числу секций.



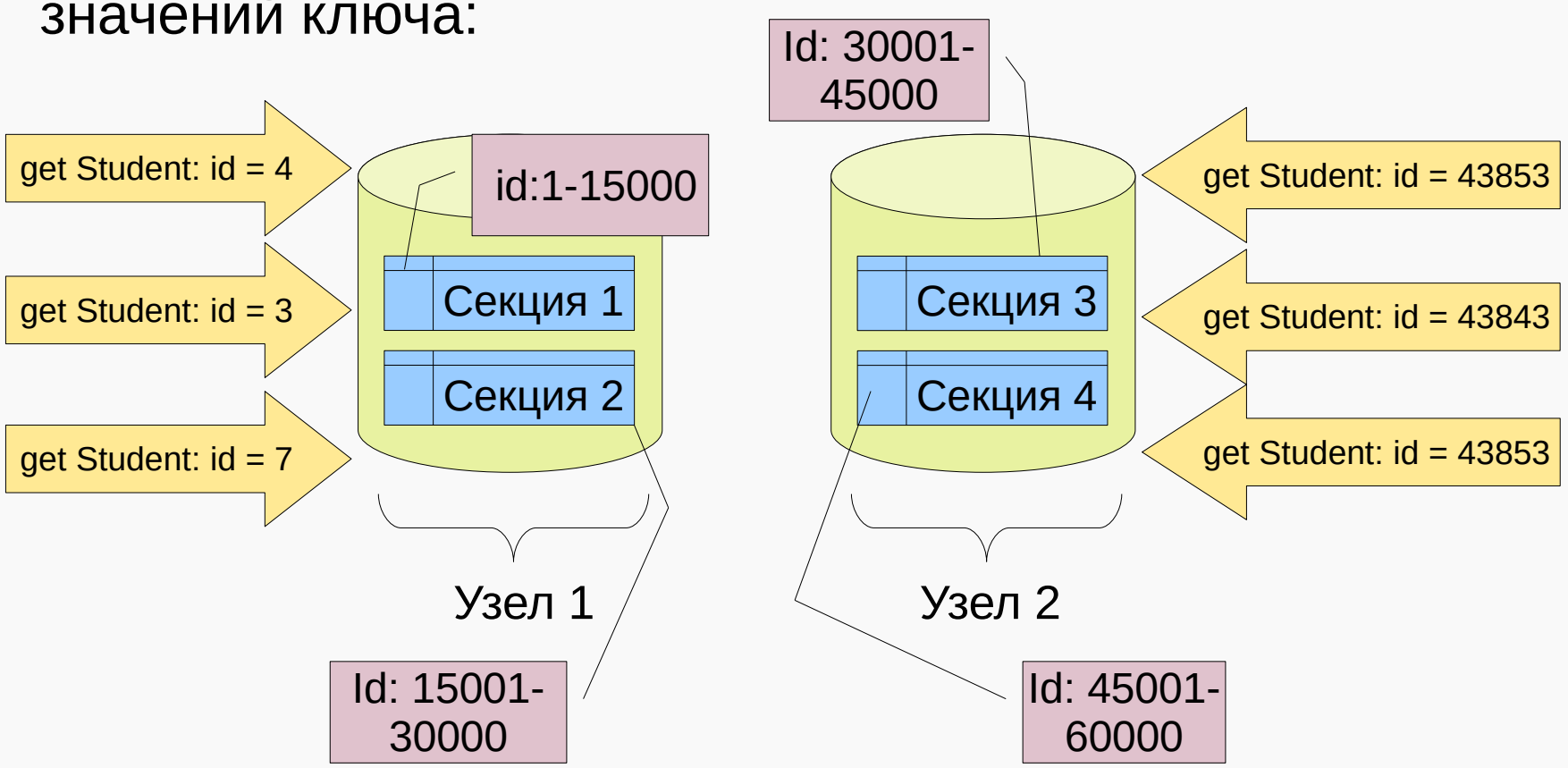
ИТМО ВТ

Шардирование по диапазонам

- **Стратегия**: определять узлы для записей по диапазону значений ключа.
- Система хранения фиксирует диапазоны значений для каждой секции (от минимального до максимального значения).
- Известно, на каком узле, какие секции располагаются => не нужно опрашивать все узлы.

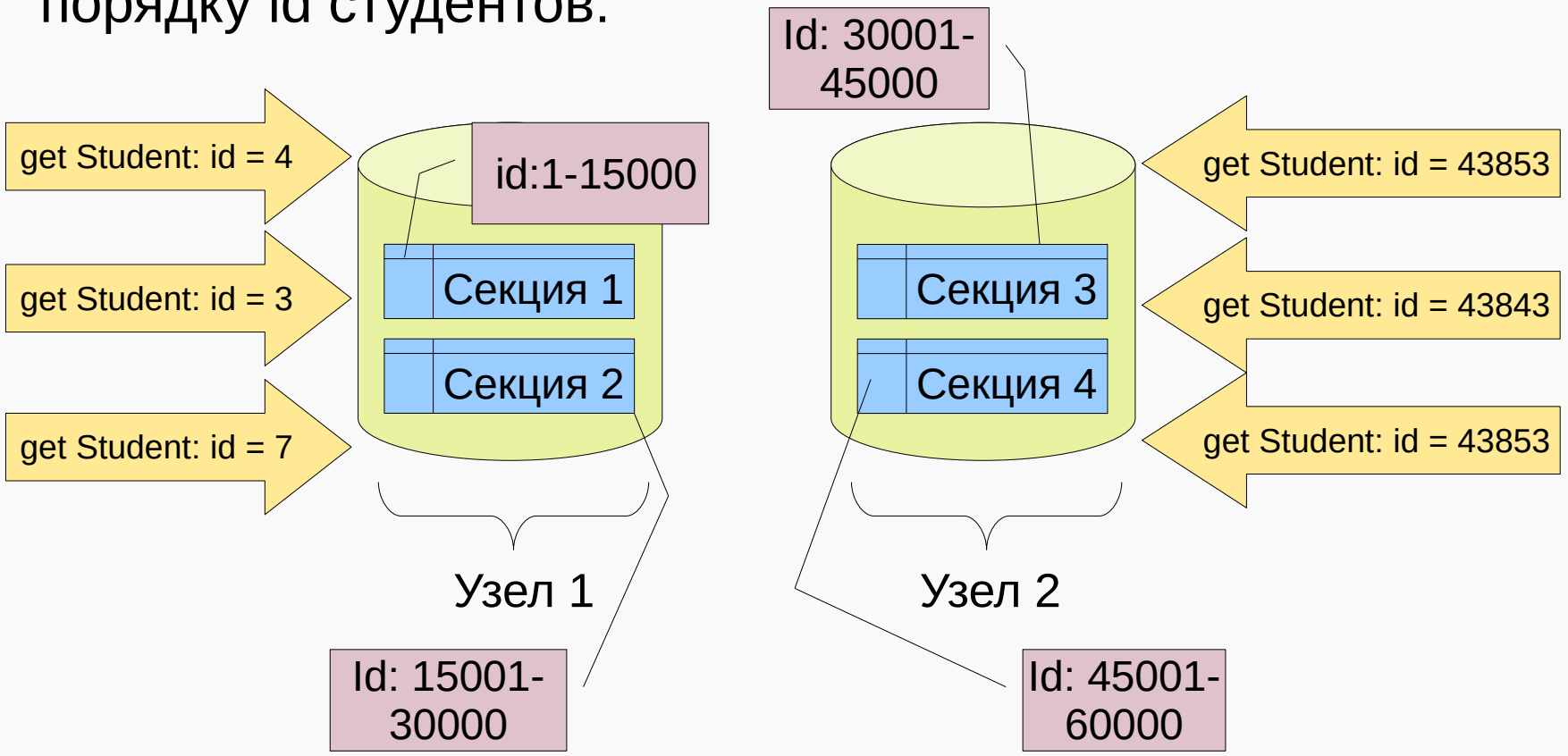
Шардирование по диапазонам

- **Стратегия**: определять узлы для записей по диапазону значений ключа:

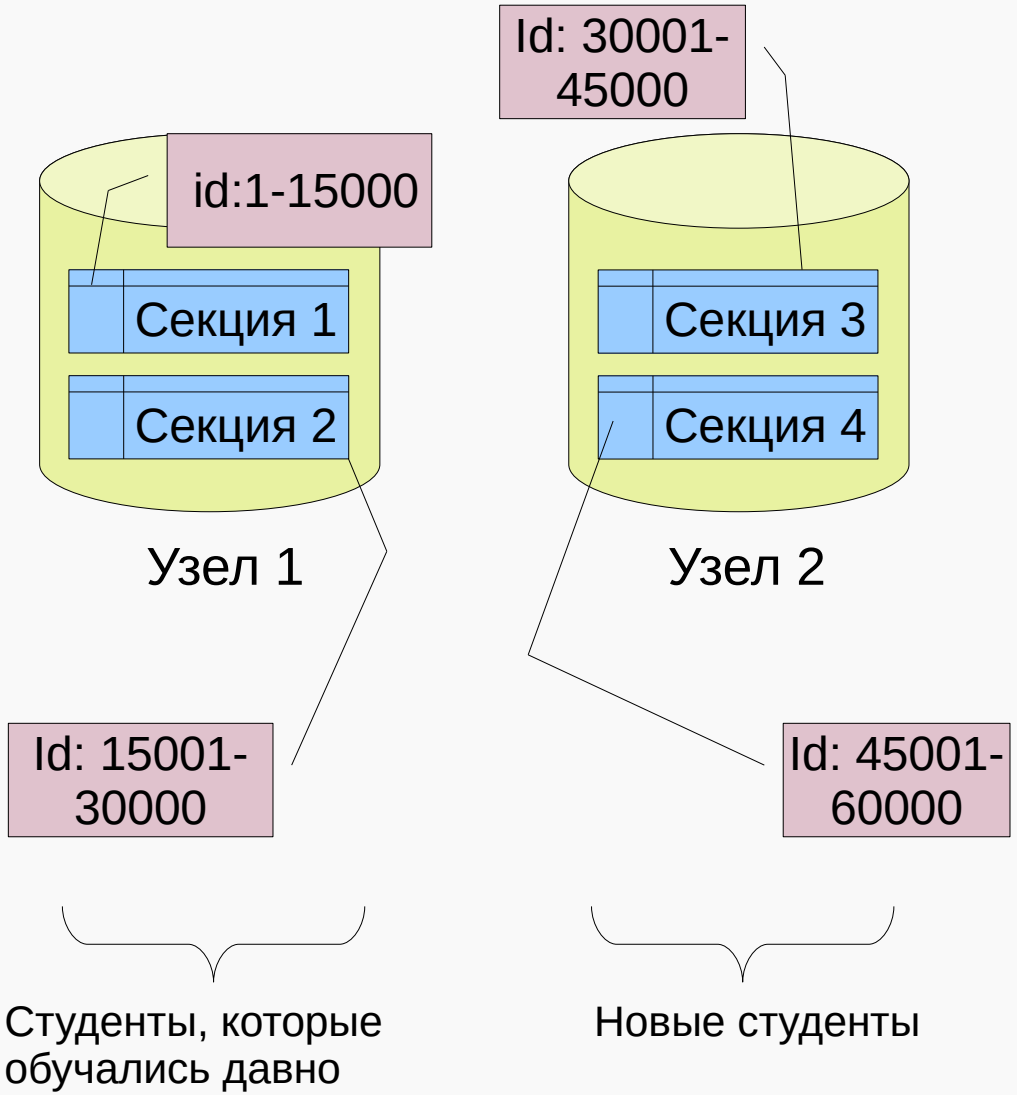


Шардирование по диапазонам

- Предположим, что в секциях содержатся данные по порядку id студентов:

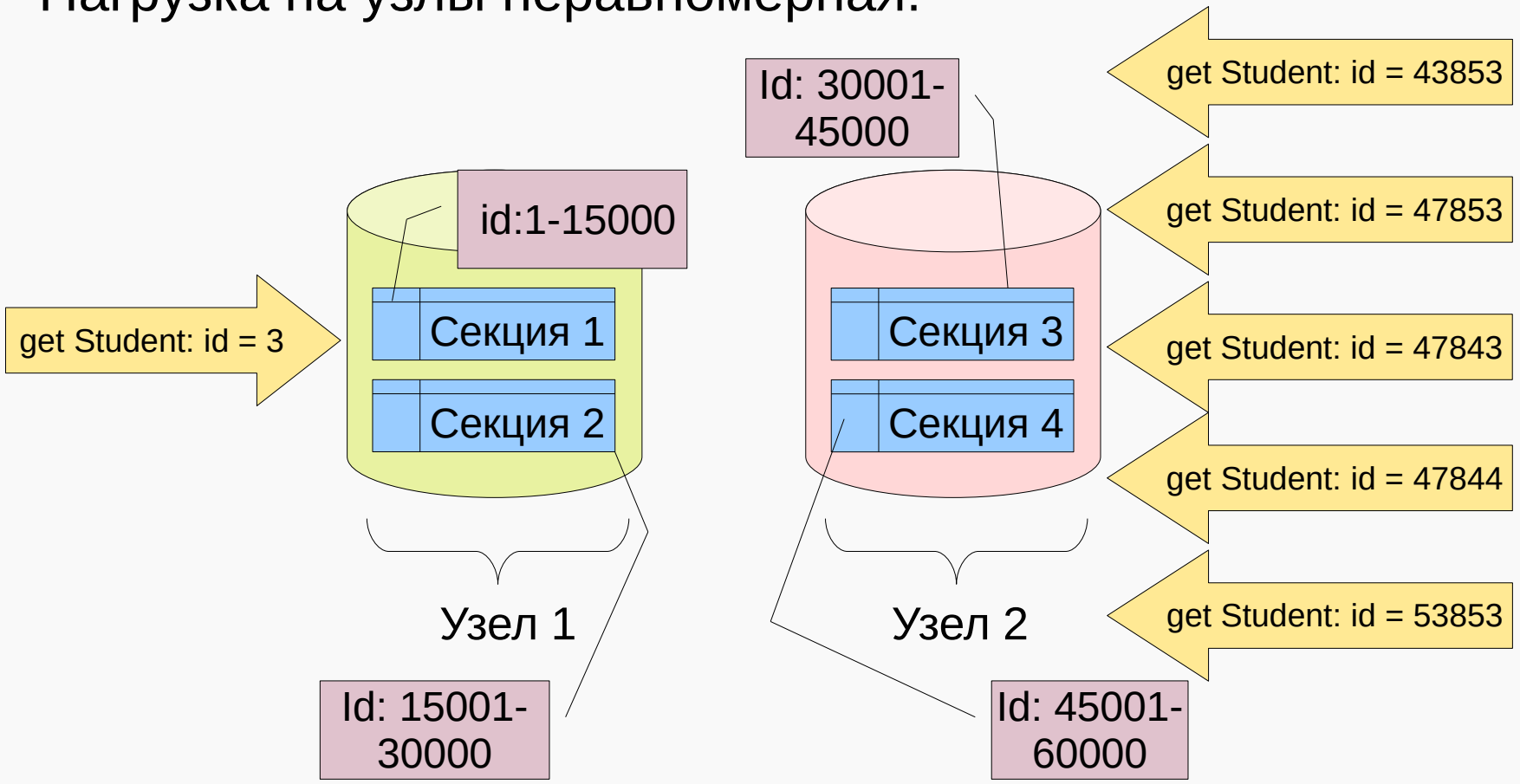


Шардирование



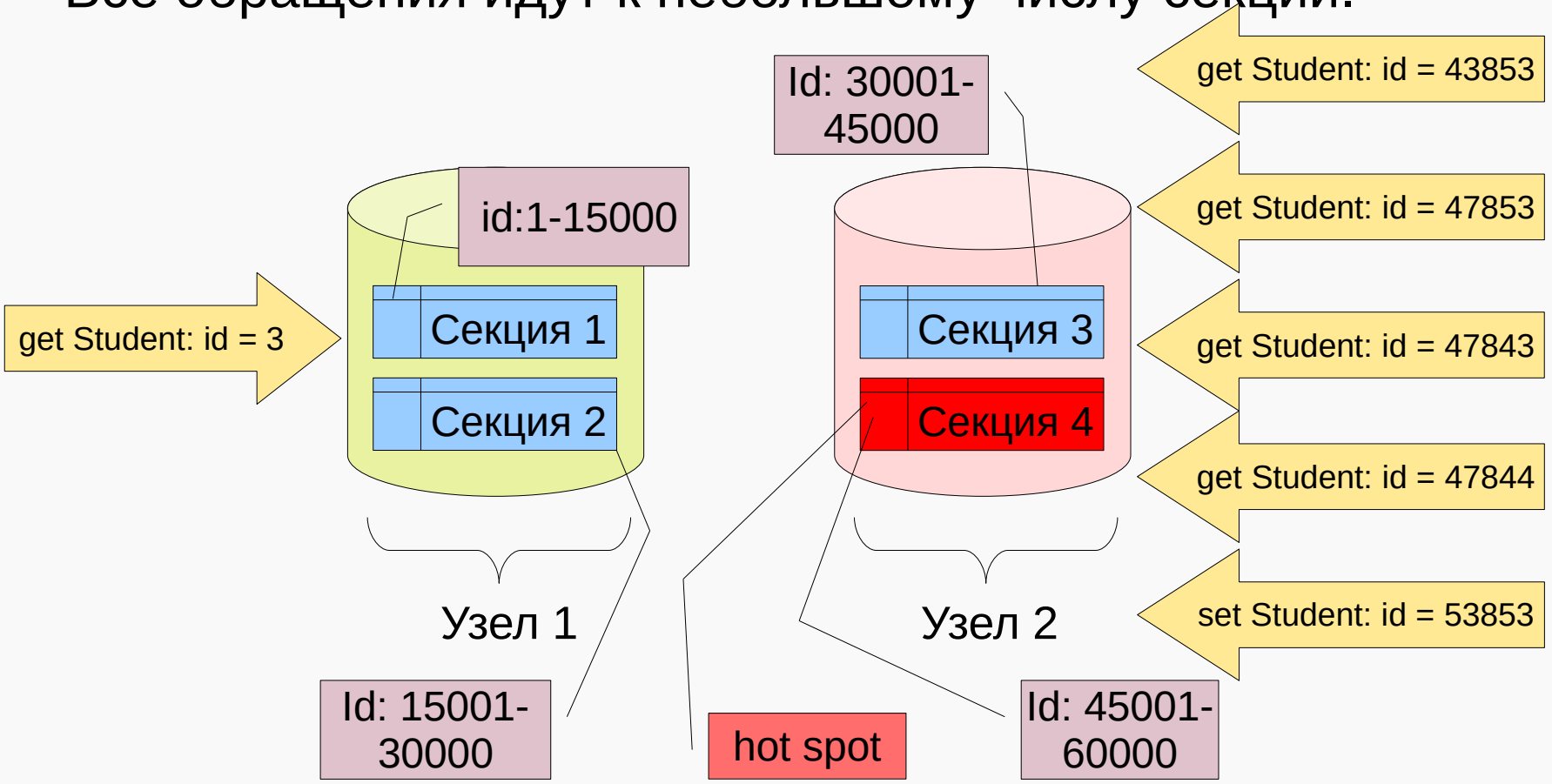
Асимметричное шардирование

- Нагрузка на узлы неравномерная:



Асимметричное шардирование

- Все обращения идут к небольшому числу секций.



ИТМО ВТ

Шардирование по диапазонам

- Эффективно выполнять операции, связанные с использованием диапазонов значений.
- **Недостаток**: при некоторых ситуациях возможно асимметричное шардирование, возникновение hot spot:
  - если некоторые диапазоны «популярнее» других.

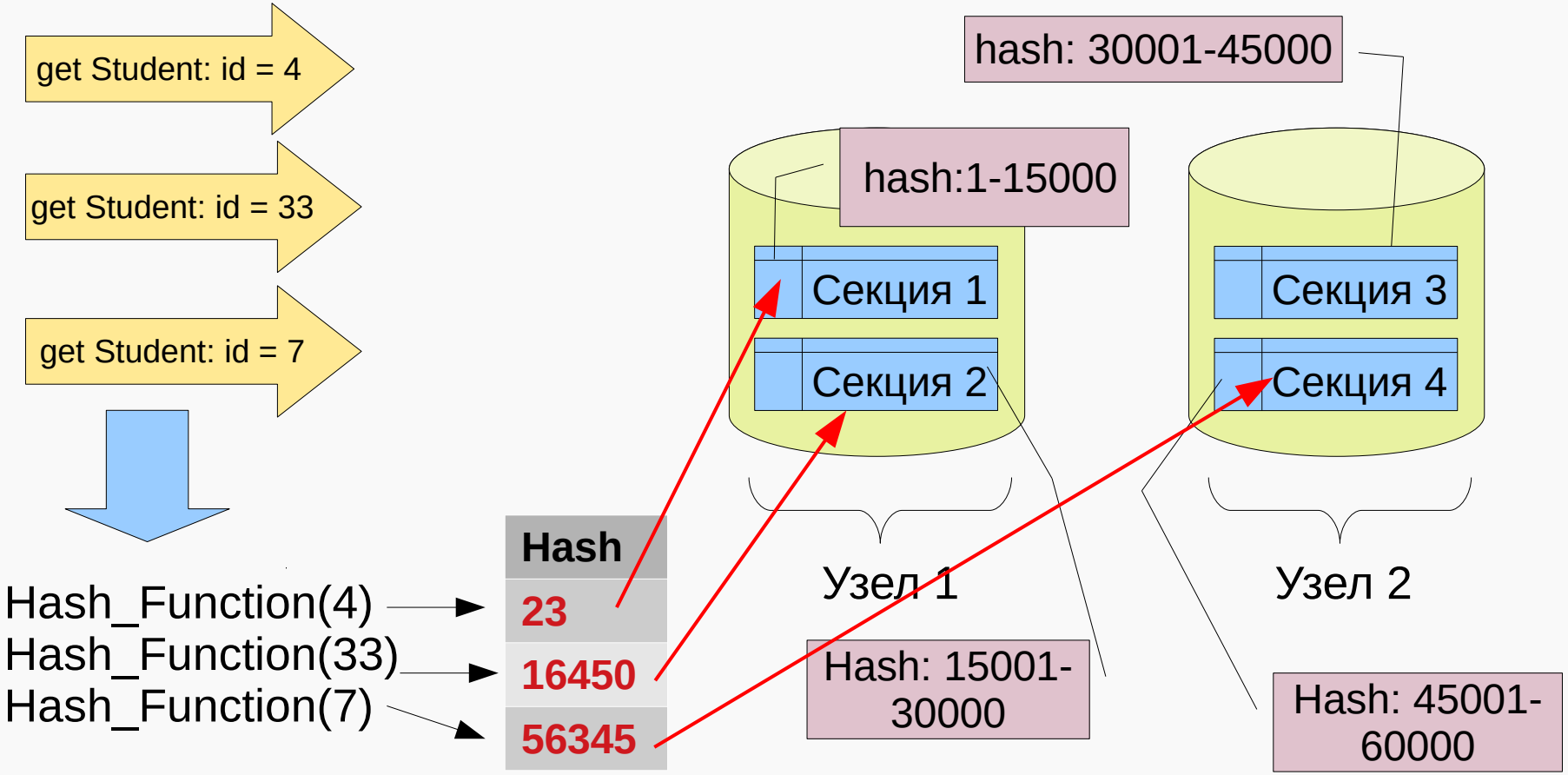
ИТМО ВТ

Шардирование по хешу

- **Стратегия**: определять узлы для записей по хешу ключа
- Система хранения фиксирует диапазоны значений хеша для каждой секции.
- Известно, на каком узле, какие секции располагаются => не нужно опрашивать все узлы.

Шардирование по хешу

- **Стратегия**: определять узлы для записей по хешу ключа:



ИТМО ВТ

Шардирование по хешу

- Решает проблему с разбиением данных, когда диапазоны значений неравномерно используются (если есть более часто используемые диапазоны).
- **Недостаток**: теряем «локальность» данных, связанную с использованием диапазонов значений.
- В некоторых случаях тоже возможно появление асимметрии и горячих точек (например, если одна запись очень популярна).

ИТМО ВТ

Перебалансировка секций

- С течением времени данные добавляются в существующие секции, секции растут.
- Старое разбиение на секции становится менее эффективным, нагрузка на узлы растет.
- Требуется изменение текущей конфигурации секций, перемещение данных между узлами. Такой процесс — **перебалансировка**.
- При перебалансировке хочется уменьшить объем работы по изменению существующих секций.

ИТМО ВТ

Перебалансировка фиксированного числа секций

- Изначально: требуется создать больше секций, чем доступно узлов.
- Количество секций задается при первоначальной настройке хранилища.
- При добавлении нового узла — забрать некоторые секции из существующих узлов.
- Секции перемещаются полностью, отдельные записи не перераспределяются.
- Обычно: число первоначально созданных секций соответствует максимальному числу узлов, которое может быть добавлено.
- Пример: Riak, CouchBase

ИТМО ВТ

Перебалансировка фиксированного числа секций

Преимущества:

- Секции перемещаются полностью, отдельные записи не перераспределяются.

Недостатки:

- Не динамичная система, по сути — при начальной конфигурации задаем число секций.
- Секции могут сильно «разрастаться»
- Перемещение секций между узлами может быть ресурсозатратным.

ИТМО ВТ

Динамическое шардирование

- Шардирование по диапазонам значений (обычное или хеш).
- Определяется граница секции (предельный размер)
- По мере добавления данных, когда секция увеличивается — она разбивается на две секции.
- Секции отображаются на узлы, на одном узле — может быть несколько секций.
- При добавлении нового узла его можно заполнить секциями с других узлов.
- Реализуется в MongoDB, HBase

ИТМО ВТ

Динамическое шардирование

Преимущества:

- Гибкий подход, секции не увеличиваются бесконтрольно.

Недостатки:

- Требуется контроль за первоначальным разбиением данных:
  - Если данных мало они могут оказаться на одном узле.

ИТМО ВТ

Шардирование в соответствии с числом узлов

- Число секций определяется количеством действующих узлов.
- На узле находится определенное число секций.
- Если добавляется новый узел, некоторые секции на других узлах делятся и перераспределяются на новый узел. Число секций на узлах не меняется.
- Возможная проблема: секции могут сильно отличаться по размерам. Зависит от выбора секции при перебалансировке.

ИТМО ВТ

Конечная согласованность

- **Конечная согласованность** (eventual consistency) — эффект, при котором актуальные изменения распределяются по узлам и применяются постепенно.
- Часть кластера в некоторый момент времени может содержать **устаревшие** данные.
- **Задержка репликации** (replication lag) — задержка во времени между завершением изменения на мастере и ее применении на зависимом узле.

ИТМО ВТ

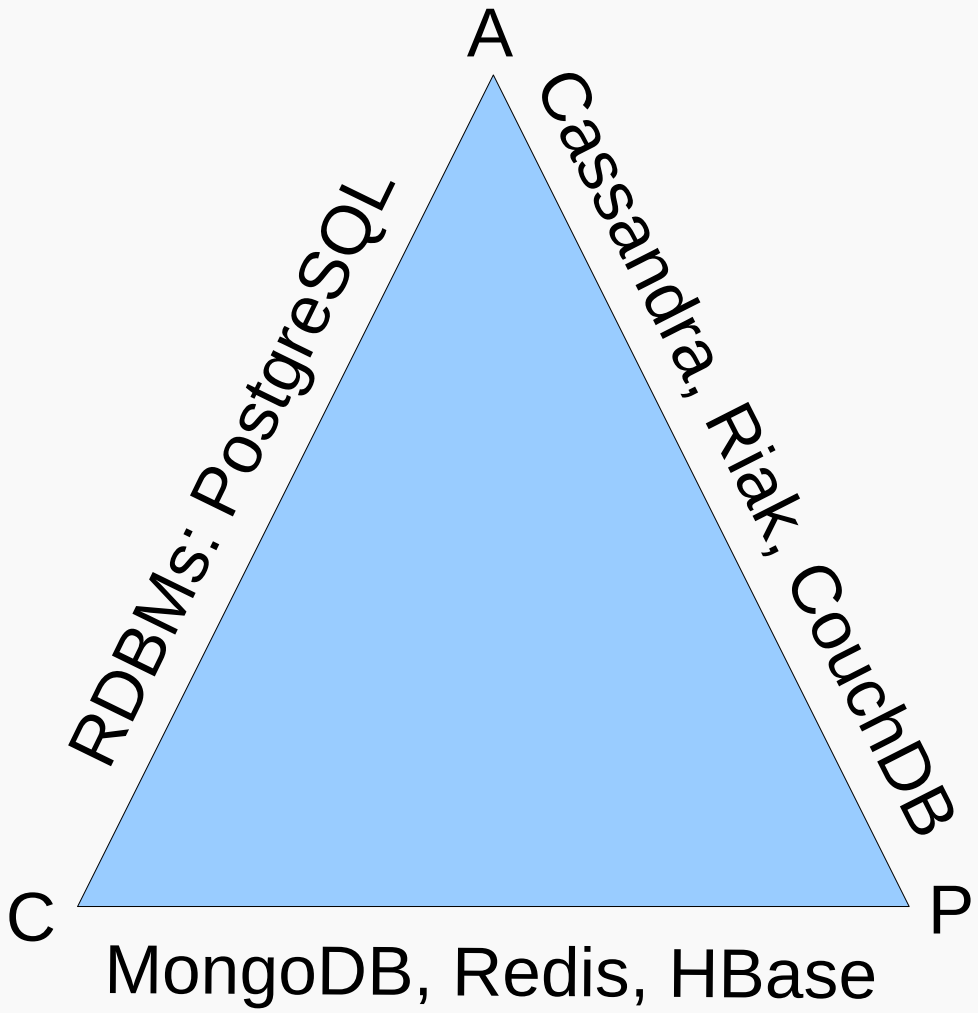
Теорема CAP

В любой реализации распределённых вычислений возможно обеспечить не более двух из следующих свойств:

- **Согласованность** (consistency) — в один момент времени данные не противоречат друг другу вне зависимости от узла, к которому происходит обращение.
- **Доступность** (availability) — на любой запрос к распределённой системе будет получен ответ/ответы с разных узлов могут отличаться.
- **Устойчивость к делению** (partition tolerance) — если теряется связь между узлами, система продолжает работать.

ИТМО ВТ

CAP-системы



ИТМО ВТ

BASE

Для распределенных систем во многих случаях ACID сложно реализуем.

Basically Available, Soft-state, Eventually consistent:

- **Basic Availability** - система отвечает на любой запрос, даже если не все узлы доступны. Ответ может быть содержать несогласованные данные.
- **Soft-state** - состояние системы может изменяться со временем из-за eventual consistency.
- **Eventual consistency** - система в итоге прийдет к согласованному состоянию.