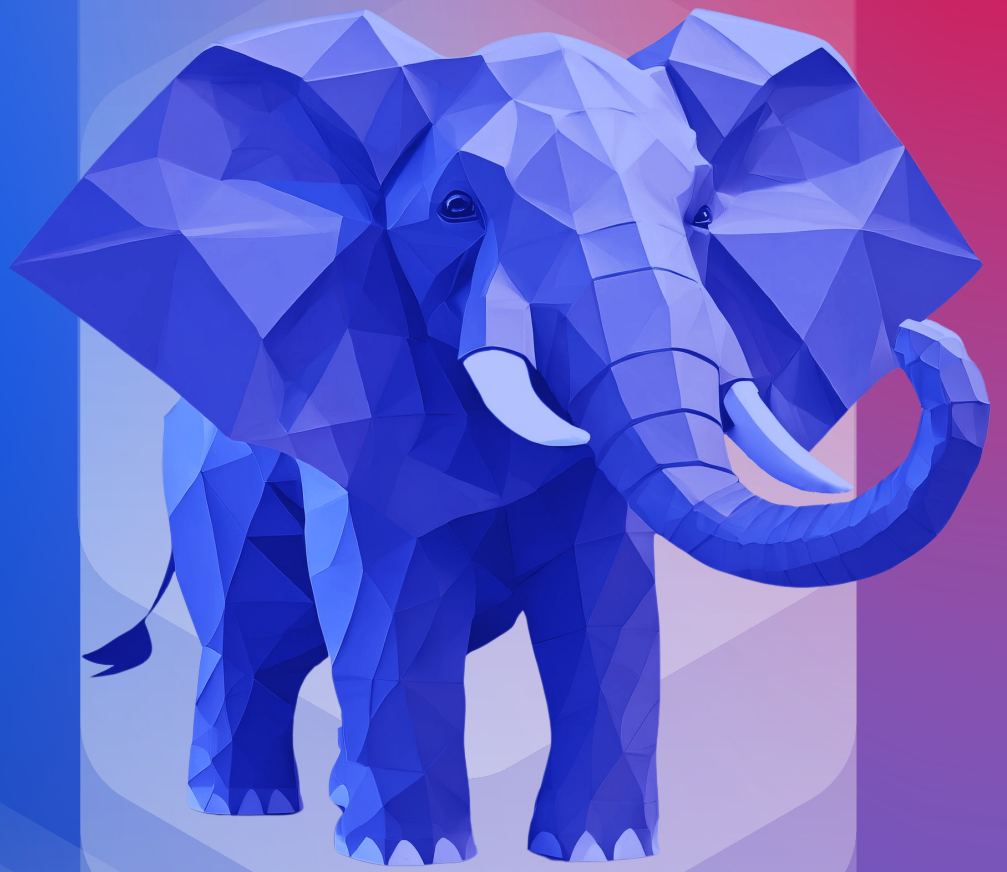
PostgresPro

Реализация архитектуры максимальной доступности в СУБД Postgres Pro Enterpise

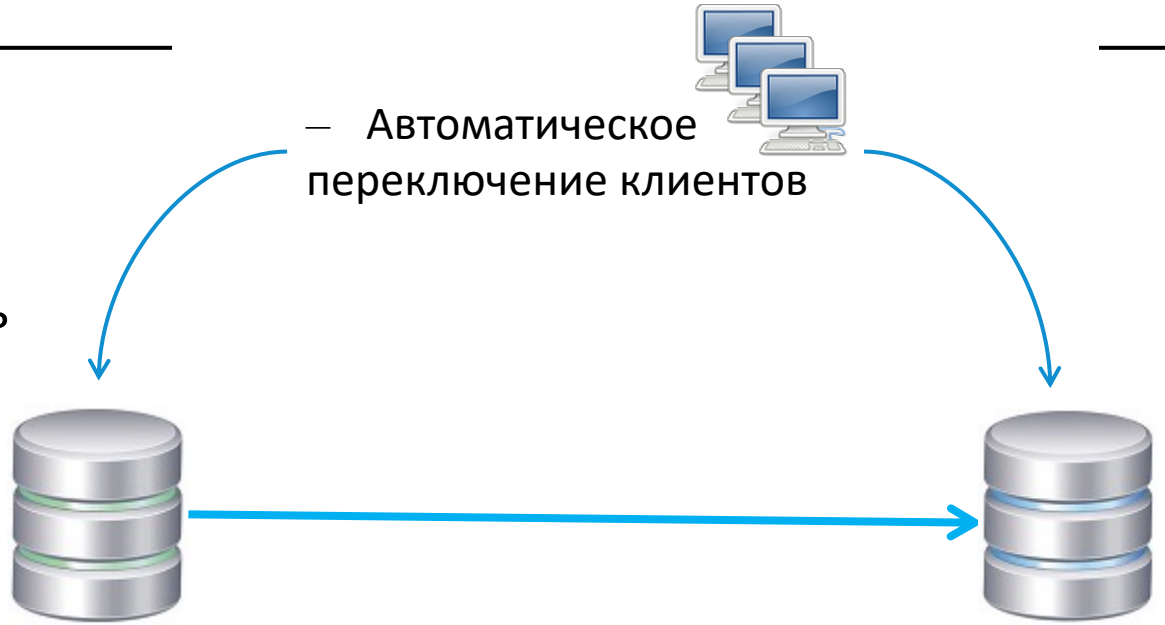


Забелин Андрей

a.zabelin@postgrespro.ru

Postgres Pro : Технологии высокой доступности

– Автоматическое переключение клиентов



Мультимастер
- Масштабируемость
- Защита от сбоя сервера

pg_probackup
- Бэкап на диск или в облако

pg_repack
- Снижение плановых простоев

shardman
- Горизонтальная масштабируемость

Физическая репликация
- Защита от сбоя СХД,
- Защита от катастрофы
- Вынос read-only нагрузки

BiHA
- Автоматическое переключение мастера на реплику

Логическая репликация
в режиме Active-active

pg probackup Основные преимущества

PostgresPro

- Инкрементальное копирование:
  - экономия места на диске
  - копии создаются быстрее
  - восстановление быстрее, чем воспроизведение файлов WAL.
- Инкрементальное восстановление:
  - ускорение восстановления благодаря повторному использованию неизменённых страниц, имеющихся в PGDATA.
- Прямое чтение и запись в S3
- Поддержка СРК-систем

pg probackup Инкрементальное копирование

Инкрементальные копии создаются на уровне страниц и включают только те данные, которые изменились со времени последнего копирования.

pg_probackup поддерживает следующие режимы :

- В режиме DELTA pg_probackup считывает все файлы баз данных и копирует только те страницы, которые изменились со времени предыдущего копирования.

  объём ввода/вывода может равняться объёму при полном резервном копировании.

- В режиме PAGE pg_probackup сканирует все файлы WAL в архиве с момента создания предыдущей полной или инкрементальной копии и копирует страницы, фигурирующие в записях WAL.

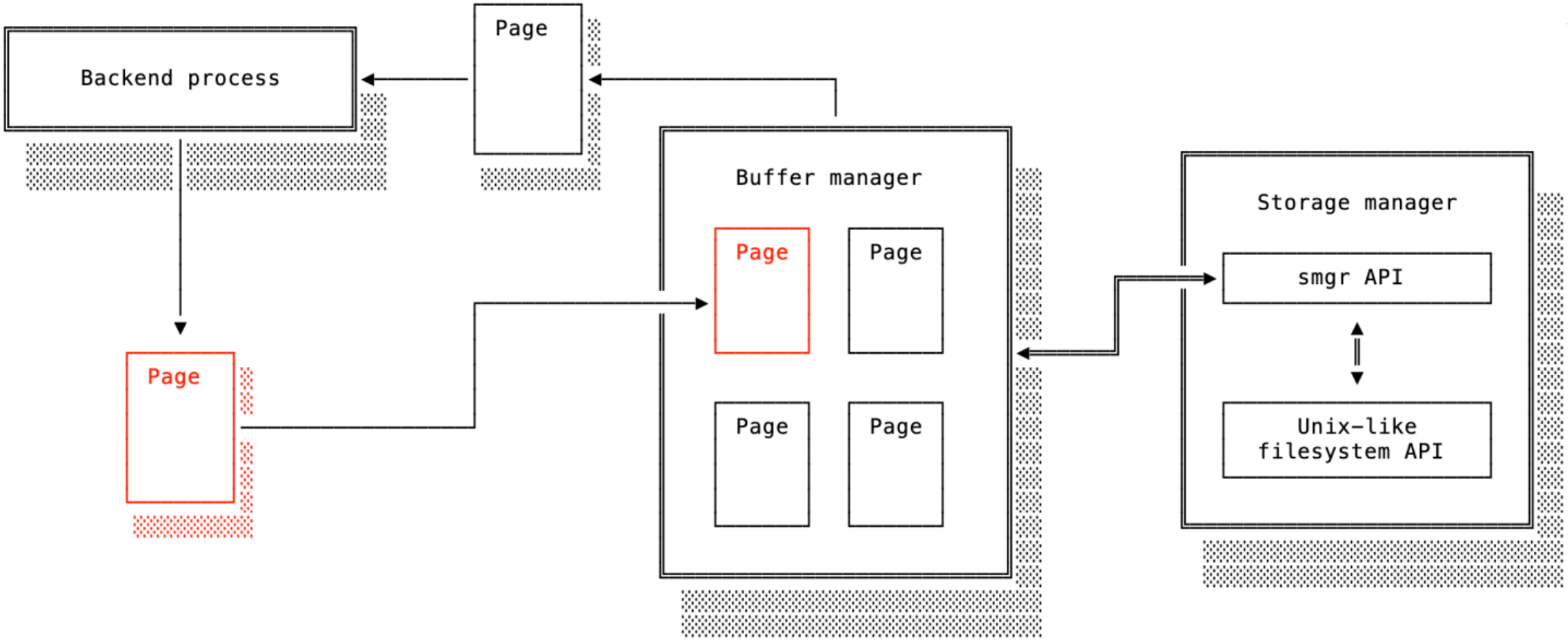
  если размер WAL файлов сравним с размером базы данных, ускорение будет менее значительным, но размер копии будет меньше.

- В режиме PTRACK Postgres Pro отслеживает изменения страниц на лету. При каждом изменении страницы она помечается в специальной карте PTRACK.

  отслеживание привносит небольшие издержки в работу сервера, но значительно ускоряет инкрементальное копирование.

PTRACK

PTRACK — это механизм, предназначенный для инкрементального резервного копирования базы данных Postgres Pro на уровне блоков.



PTRACK

Измененые блоки фиксируются при их записи на диск (используется storage manager API).
Карты блоков PTRACK из оперативной памяти сбрасываются на диск во время контрольных точек.

pg probackup Объединение резервных копий

PostgresPro

pg_probackup merge объединяет копии, относящиеся к одной цепочке инкрементальных копий.

pg_probackup merge -B ***каталог_копий*** *--instance* ***имя_экземпляра*** \

-i ***идентификатор_резервной_копии***

Если выбрана полная копия, она будет объединена с первой инкрементальной копией после неё.

Если выбрана инкрементальная копия, она будет объединена с родительской полной копией, включая все инкрементальные копии между ними.

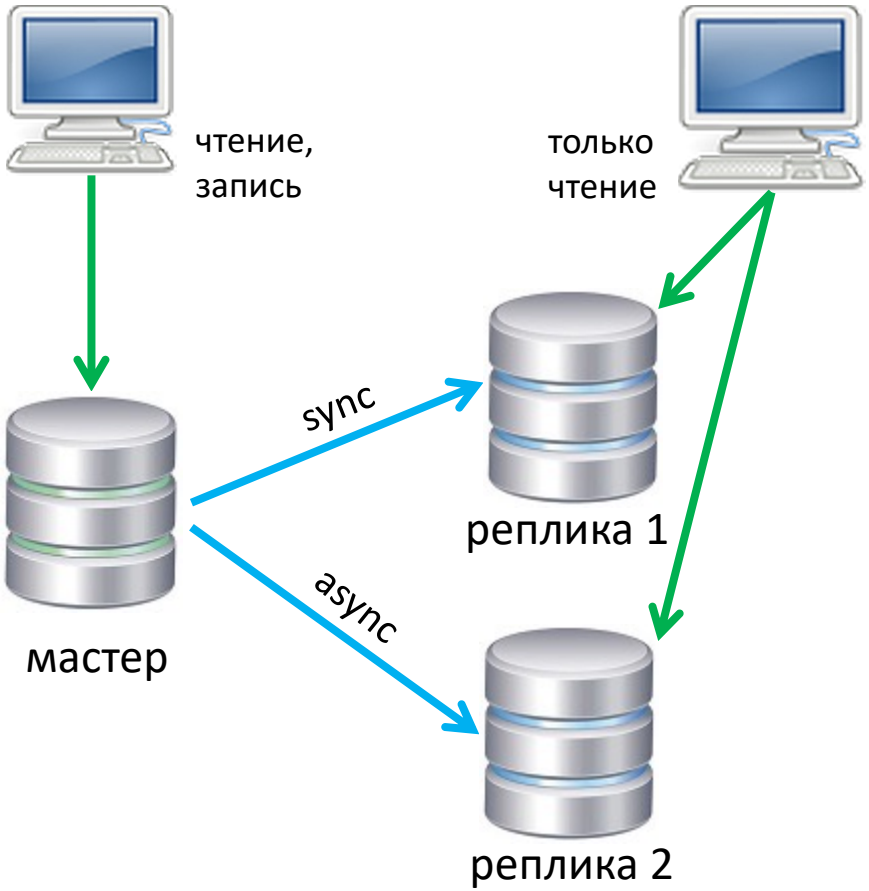
После завершения объединения результирующая полная копия будет вмещать в себя все данные, а инкрементальные копии будут удалены как избыточные.

Поддерживается сжатие, если объединяемые копии выполнялись с одинаковой степенью сжатия

pg probackup дополнительные преимущества

- Параллельное выполнение: выполнение внутренних процессов команд backup, restore, merge, delete, validate и checkdb в несколько параллельных потоков.
- Сжатие: хранение копируемых данных в сжатом состоянии для экономии дискового пространства.
- Исключение дублирования: экономия дискового пространства за счёт фильтрации при инкрементальном копировании файлов, не содержащих непосредственно данные (например, файлов _vm или _fsm), если эти файлы не изменялись с момента создания предыдущей копии в цепочке инкрементальных копий.
- Политика хранения: управление архивами WAL и резервными копиями в соответствии с установленными правилами их хранения. Вы можете ограничить хранение резервных копий по времени или их количеству, а также переопределить время жизни (TTL) для избранных копий. Потерявшие актуальность резервные копии могут объединяться или удаляться.
- Каталогизация резервных копий: получение списка резервных копий, архивов WAL и соответствующей метаинформации в виде простого текста или JSON.
- Удалённый режим работы: выполнение резервного копирования экземпляра Postgres Pro, находящегося в удалённой системе, и удалённое восстановление.
- Получение резервной копии с ведомого: исключение дополнительной нагрузки на ведущий сервер.
- Архивирование внешних каталогов: резервное копирование файлов и каталогов, расположенных вне каталога данных Postgres Pro (PGDATA), например скриптов, файлов конфигурации, журналов или SQL-дампов.

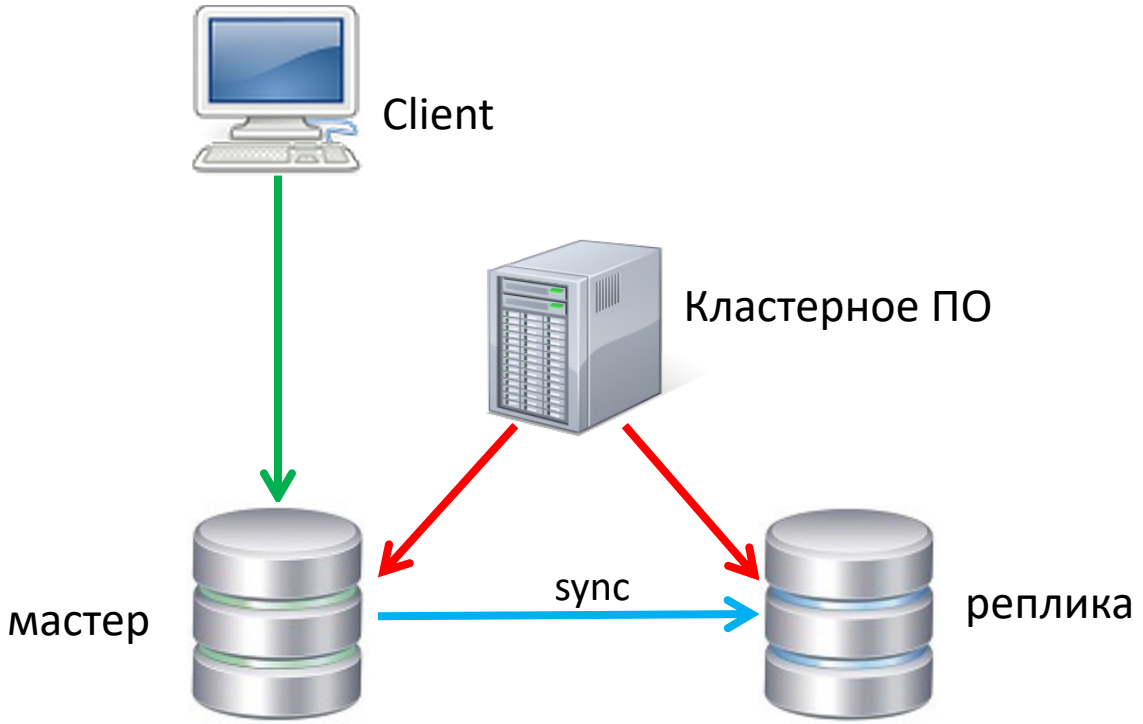
Postgres Pro : Физическая репликация



- Репликация :
  - синхронная/асинхронная,
- Реплика может быть открыта на чтение
  - часть нагрузки переносится с мастера
  - небольшие оперативные in-memory таблицы открыты на запись
  - резервная копия может выполняться на реплике
  - восстановление битых блоков БД из реплики
  - проверка битых записей журналов WAL
- Реплика может быть географически удалена

Автоматическое переключение с мастера на реплику

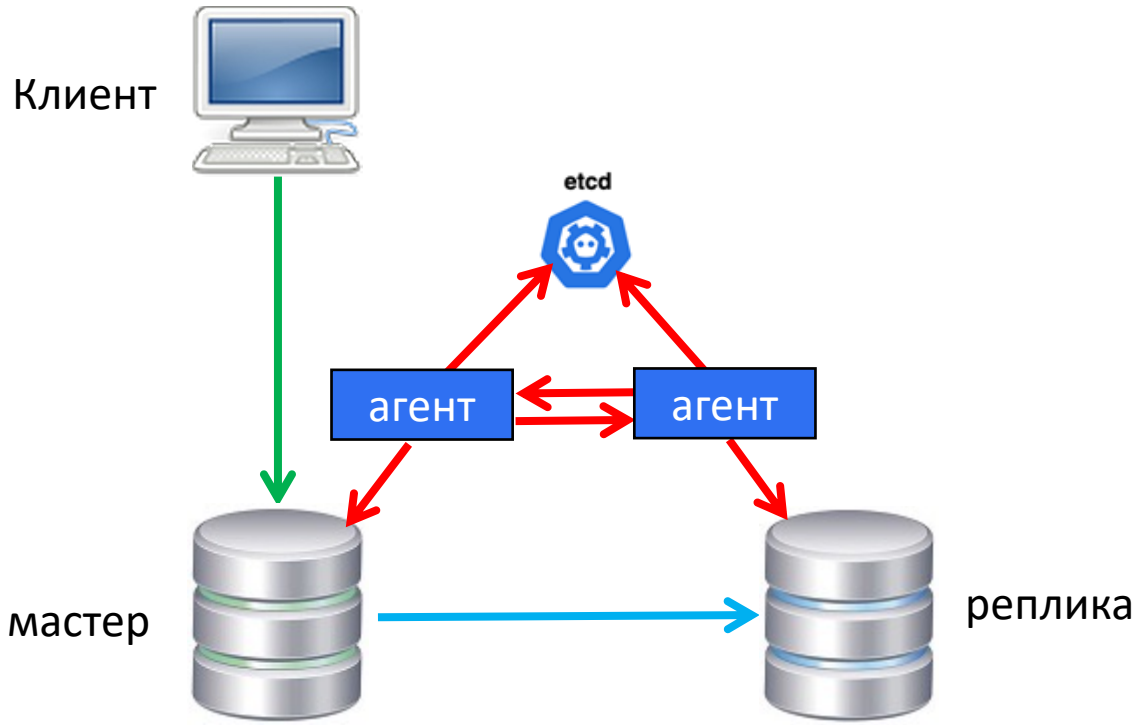
- Решение о смене ролей в отказоустойчивом кластере при сбое мастера может приниматься автоматически
- Необходимо также автоматически переключить на новый мастер и клиентов
- Основная задача кластерного ПО обнаружить сбой, сменить роль реплики на новый мастер, но при этом не допустить работу двух узлов в режиме записи



Примеры кластерного ПО : Patroni, Stolon, Corosync ….

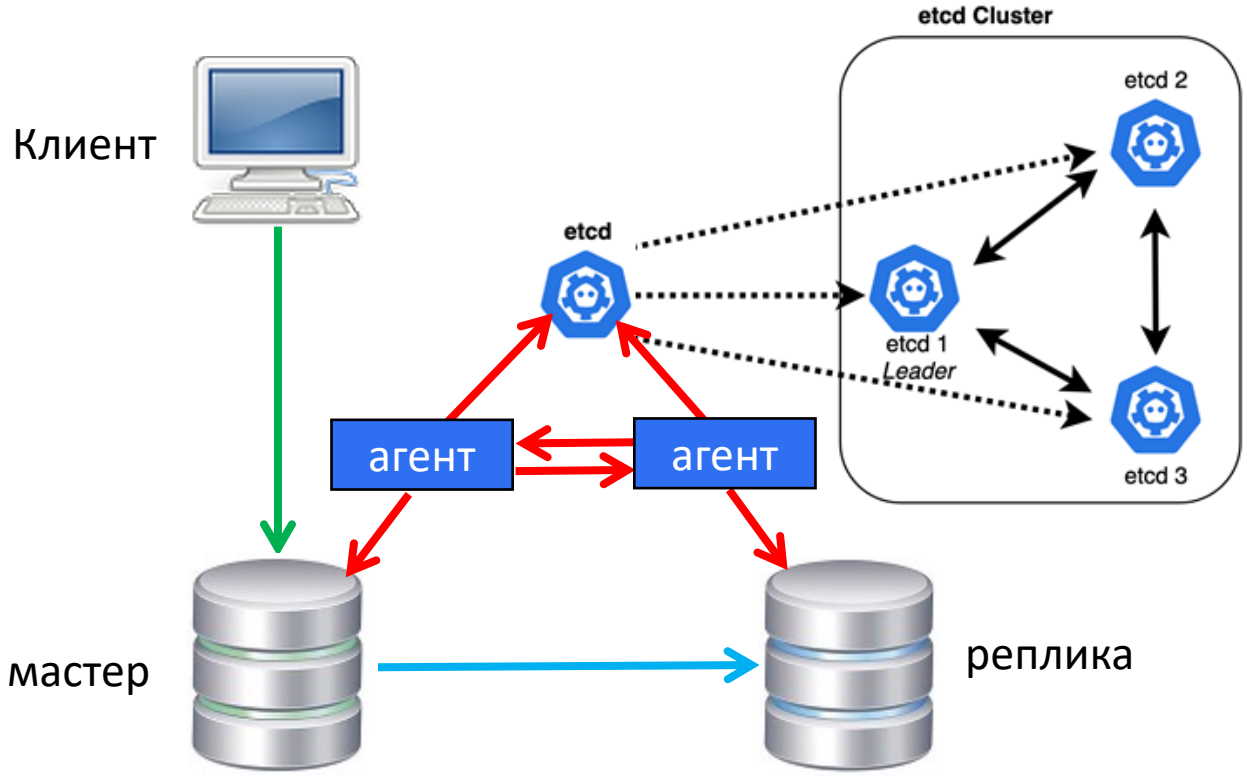
Недостатки внешнего кластерного ПО

- Внешний кластер имеет сложную архитектуру (дополнительные узлы, сетевые каналы и т.п.)



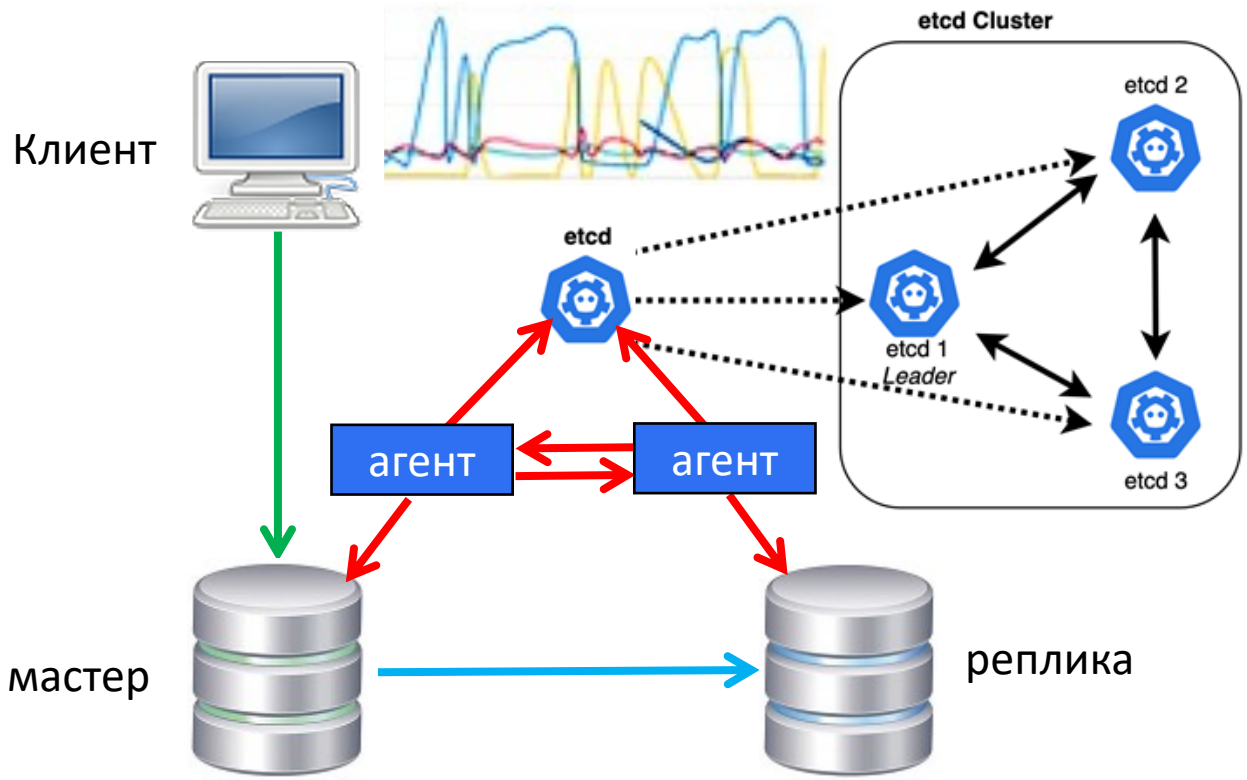
Недостатки внешнего кластерного ПО

- Внешний кластер имеет сложную архитектуру (дополнительные узлы, сетевые каналы и т.п.)
- Для элементов кластерного ПО тоже требуется отказоустойчивость



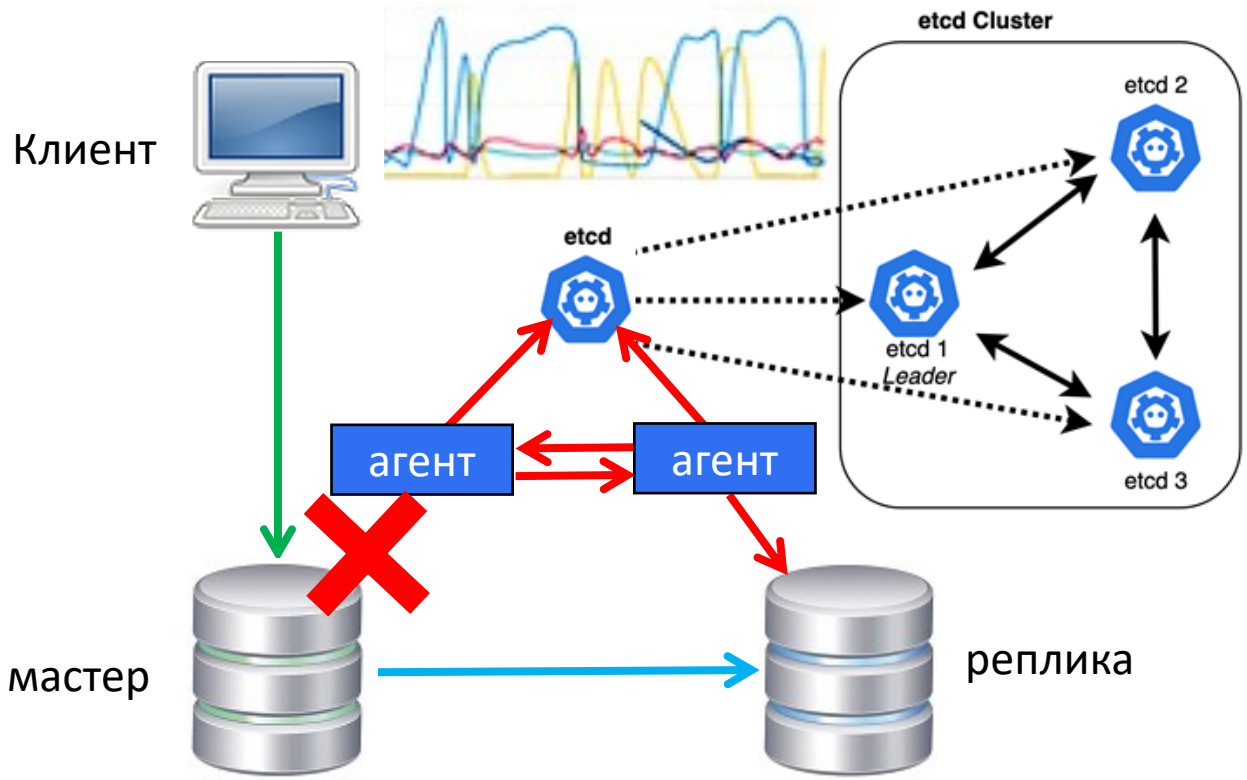
Недостатки внешнего кластерного ПО

- Внешний кластер имеет сложную архитектуру (дополнительные узлы, сетевые каналы и т.п.)
- Для элементов кластерного ПО тоже требуется отказоустойчивость
- Сложность мониторинга



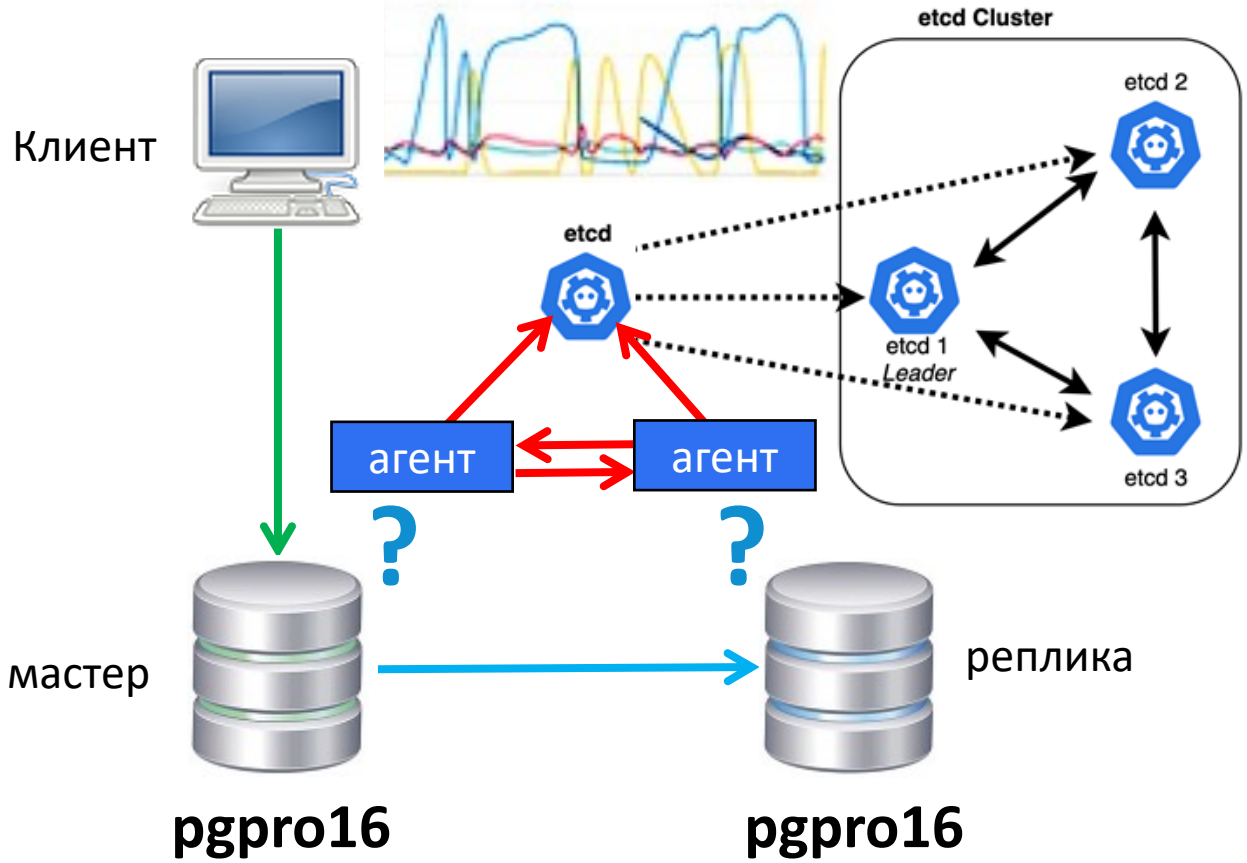
Недостатки внешнего кластерного ПО

- Внешний кластер имеет сложную архитектуру (дополнительные узлы, сетевые каналы и т.п.)
- Для элементов кластерного ПО тоже требуется отказоустойчивость
- Сложность мониторинга
- Большая нагрузка на БД может расцениваться как отказ узла



Недостатки внешнего кластерного ПО

- Внешний кластер имеет сложную архитектуру (дополнительные узлы, сетевые каналы и т.п.)
- Для элементов кластерного ПО тоже требуется отказоустойчивость
- Сложность мониторинга
- Большая нагрузка на БД может расцениваться как отказ узла
- Задержка с обновлениями версий

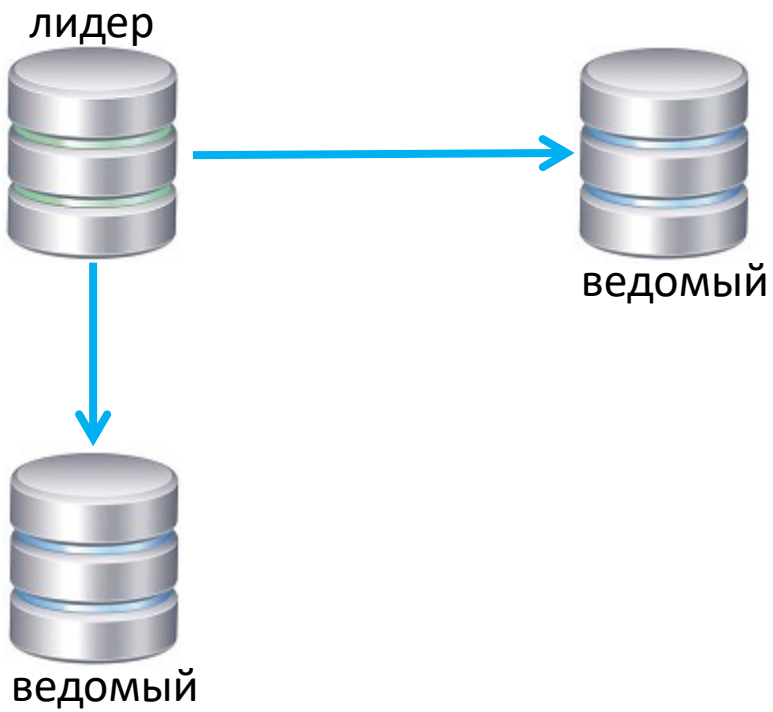


Встроенный отказоустойчивый кластер BiHA

Архитектура

Кластер состоит из нескольких узлов

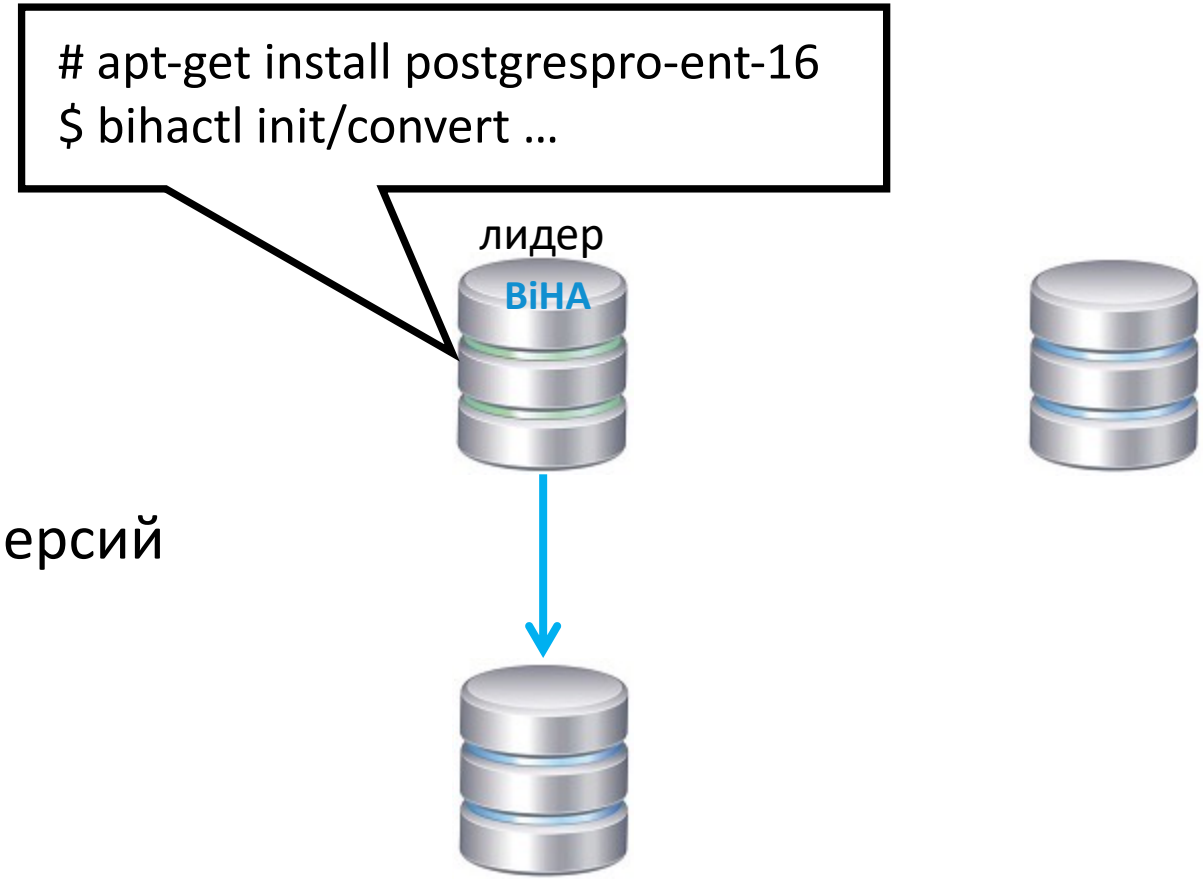
- один является лидером (leader),
- другие являются ведомыми (follower).



Встроенный отказоустойчивый кластер BiHA

Простая установка

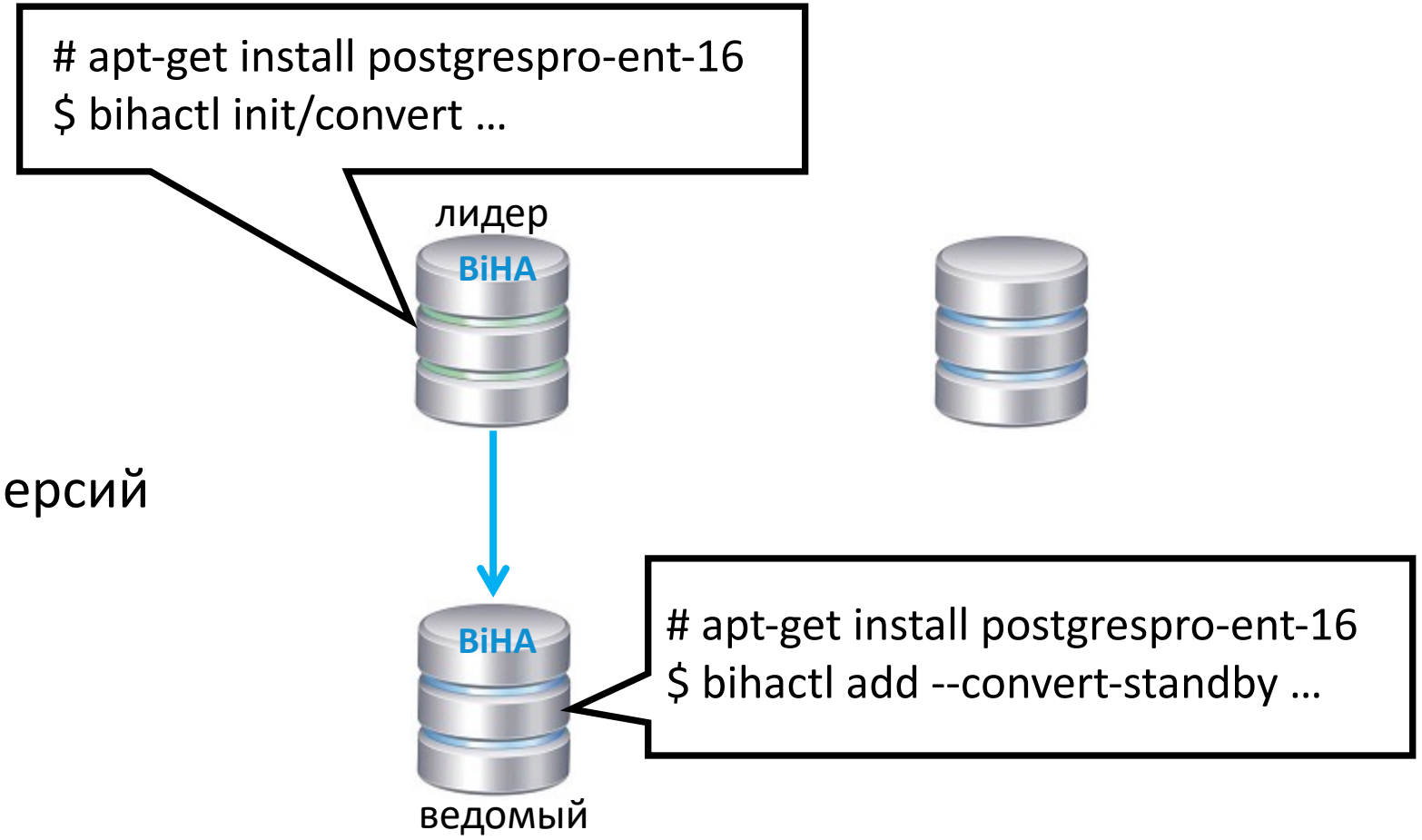
- BiHA кластер встроен в Postgres Pro.
- Простая установка и конфигурирование
- Не требуется установка дополнительного ПО
- Оперативные обновления версий



Встроенный отказоустойчивый кластер BiHA

Простая установка

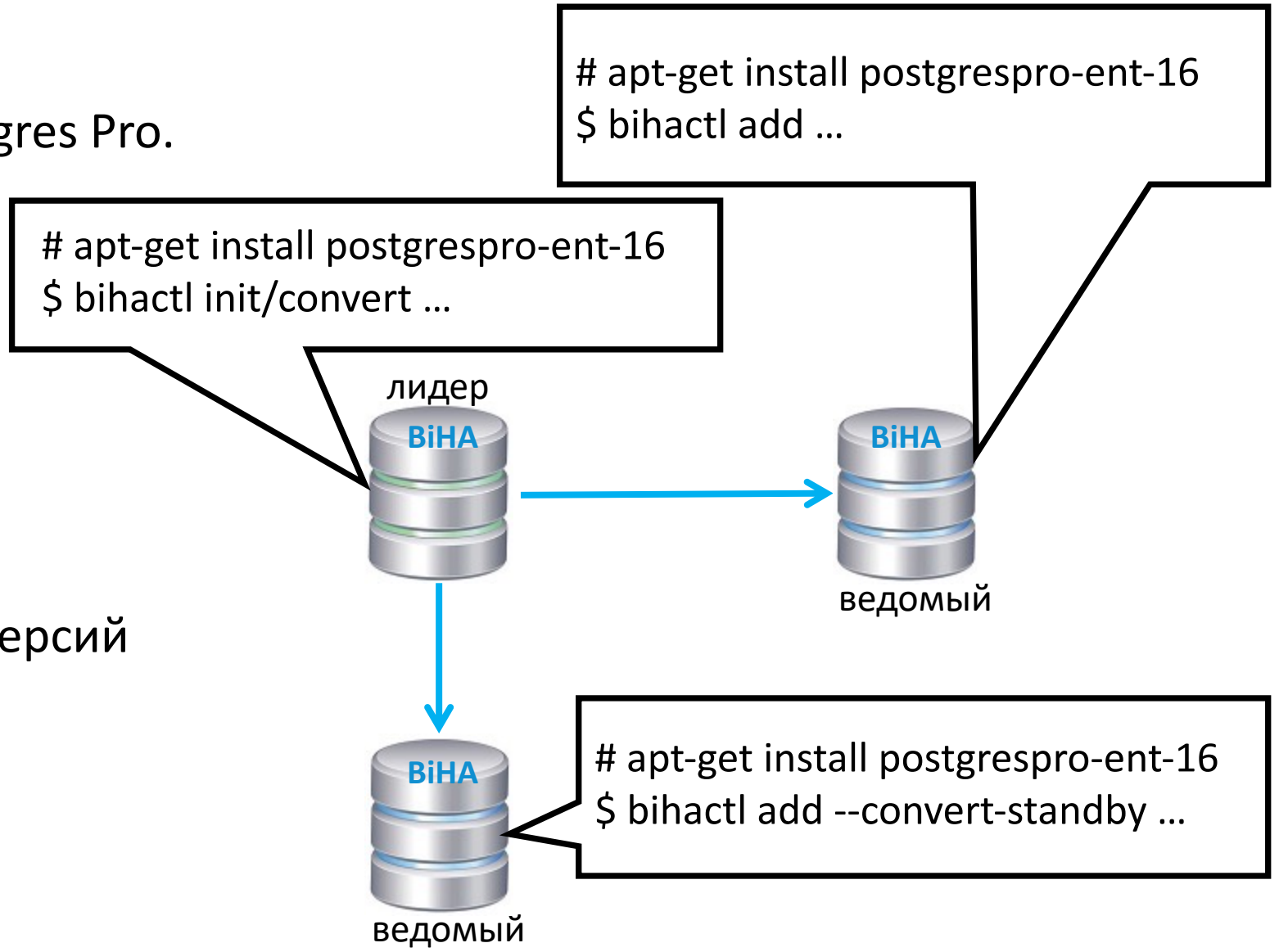
- BiHA кластер встроен в Postgres Pro.
- Простая установка и конфигурирование
- Не требуется установка дополнительного ПО
- Оперативные обновления версий



Встроенный отказоустойчивый кластер BiHA

Простая установка

- BiHA кластер встроен в Postgres Pro.
- Простая установка и конфигурирование
- Не требуется установка дополнительного ПО
- Оперативные обновления версий

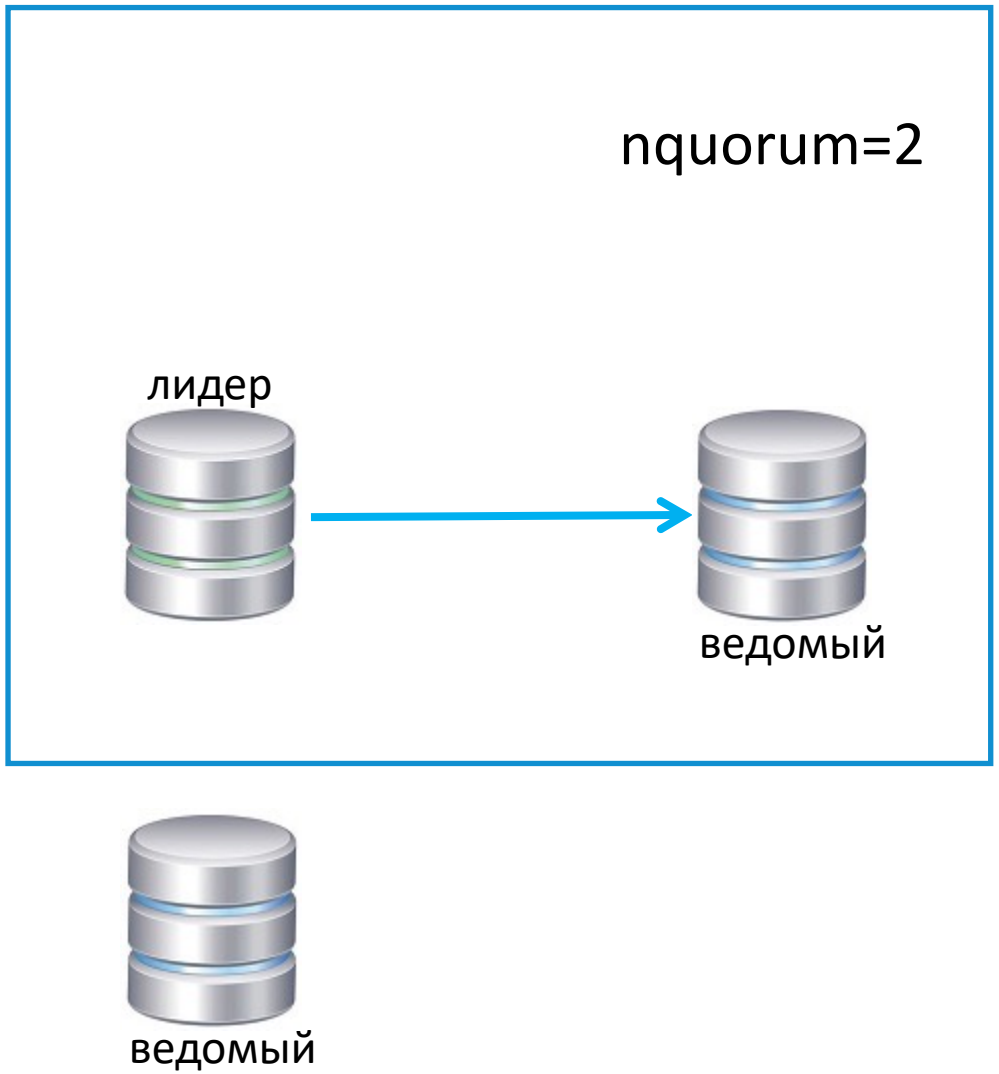


Встроенный отказоустойчивый кластер BiHA

Кластерный кворум

Кворум определяет минимальное количество узлов кластера

Лидер продолжает работать, если соблюдается кворум

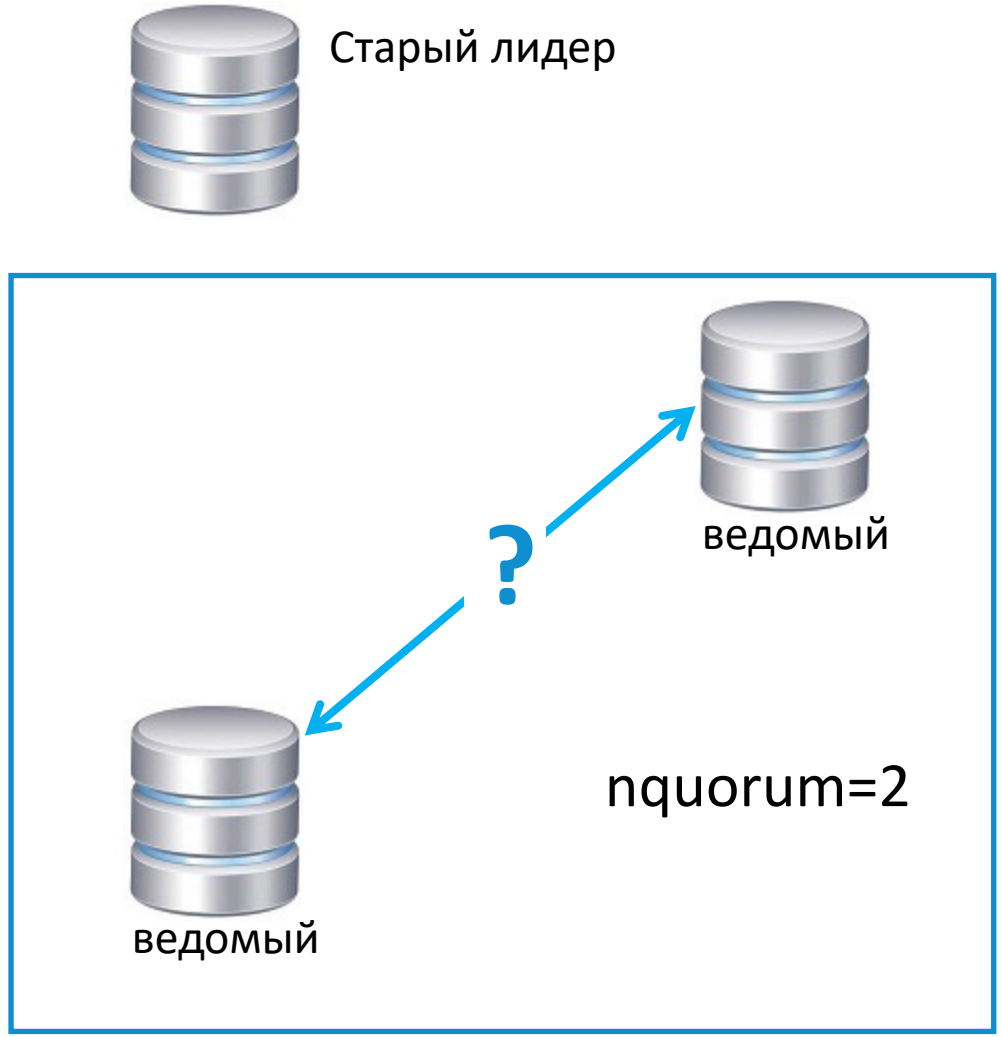


Встроенный отказоустойчивый кластер BiHA

Кластерный кворум

Лидер не может продолжать работу, если не соблюдается кворум

Ведомые организуют выборы нового лидера, если кластер содержит достаточное количество узлов

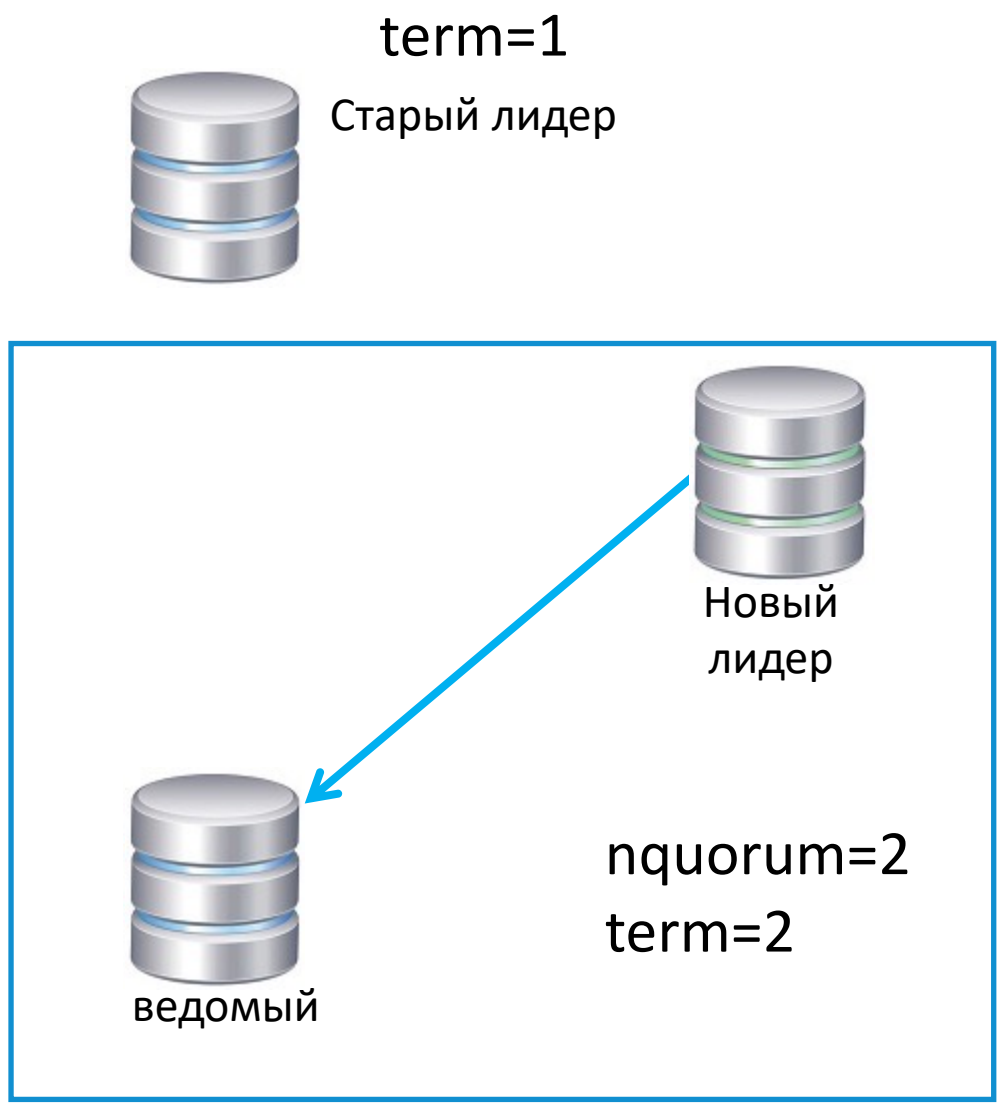


Встроенный отказоустойчивый кластер BiHA

Поколение кластера

После выбора нового лидера
в кластере меняется поколение

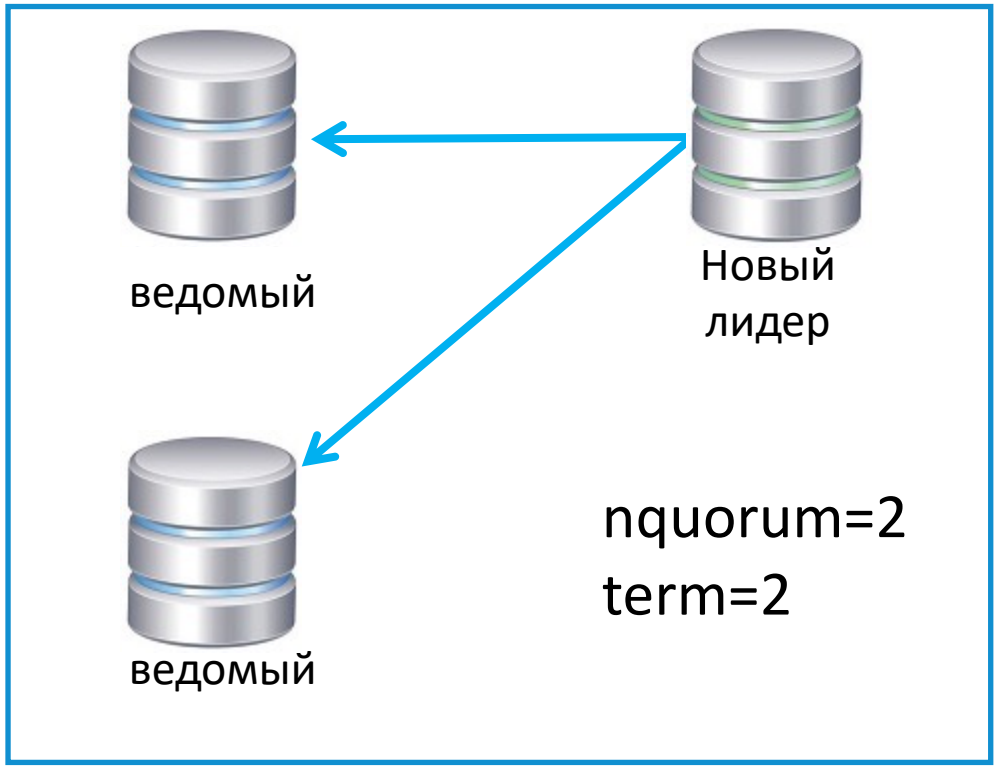
Старый лидер остаётся в старом
поколении



Встроенный отказоустойчивый кластер BiHA

Поколение кластера

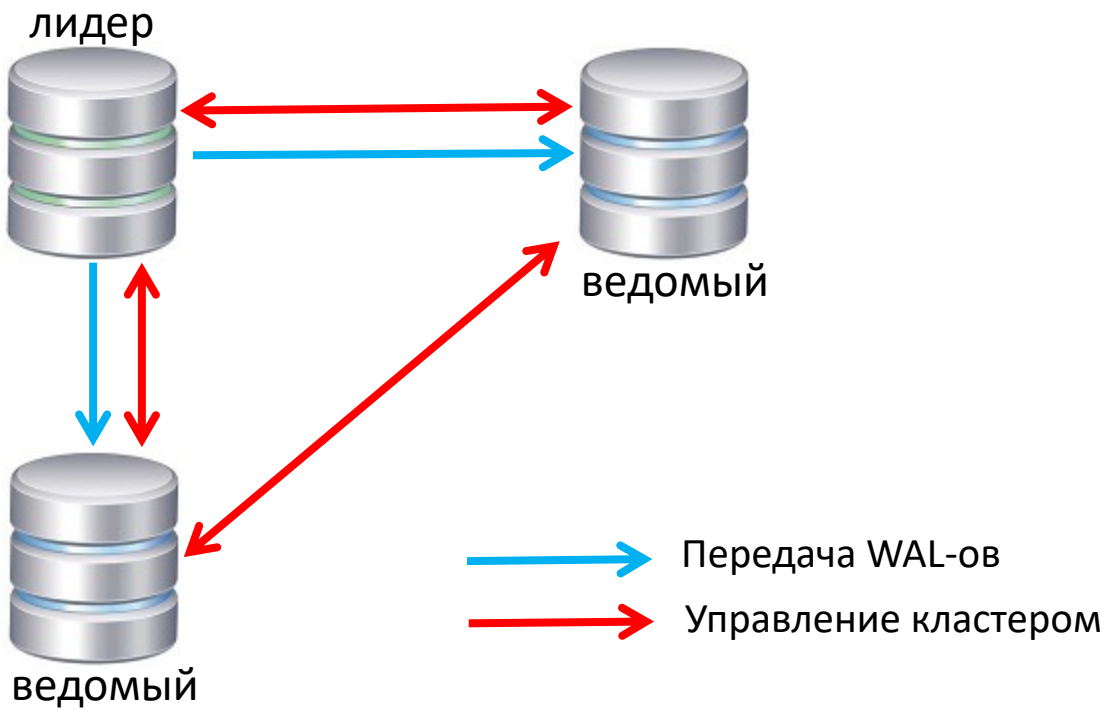
При возвращении старого лидера в кластер он не может быть уже лидером и переходит в режим ведомого



Встроенный отказоустойчивый кластер BiHA

Управляющий канал

- Взаимодействие узлов друг с другом осуществляется с использованием управляющего канала
- между любыми двумя узлами устанавливается сетевое соединение по протоколу TCP.
- Непрерывный мониторинг состояния узлов кластера.



Автоматическое переключение соединения на стороне клиента на новый мастер

На клиенте (libpq, JDBC) можно перечислить все узлы кластера,

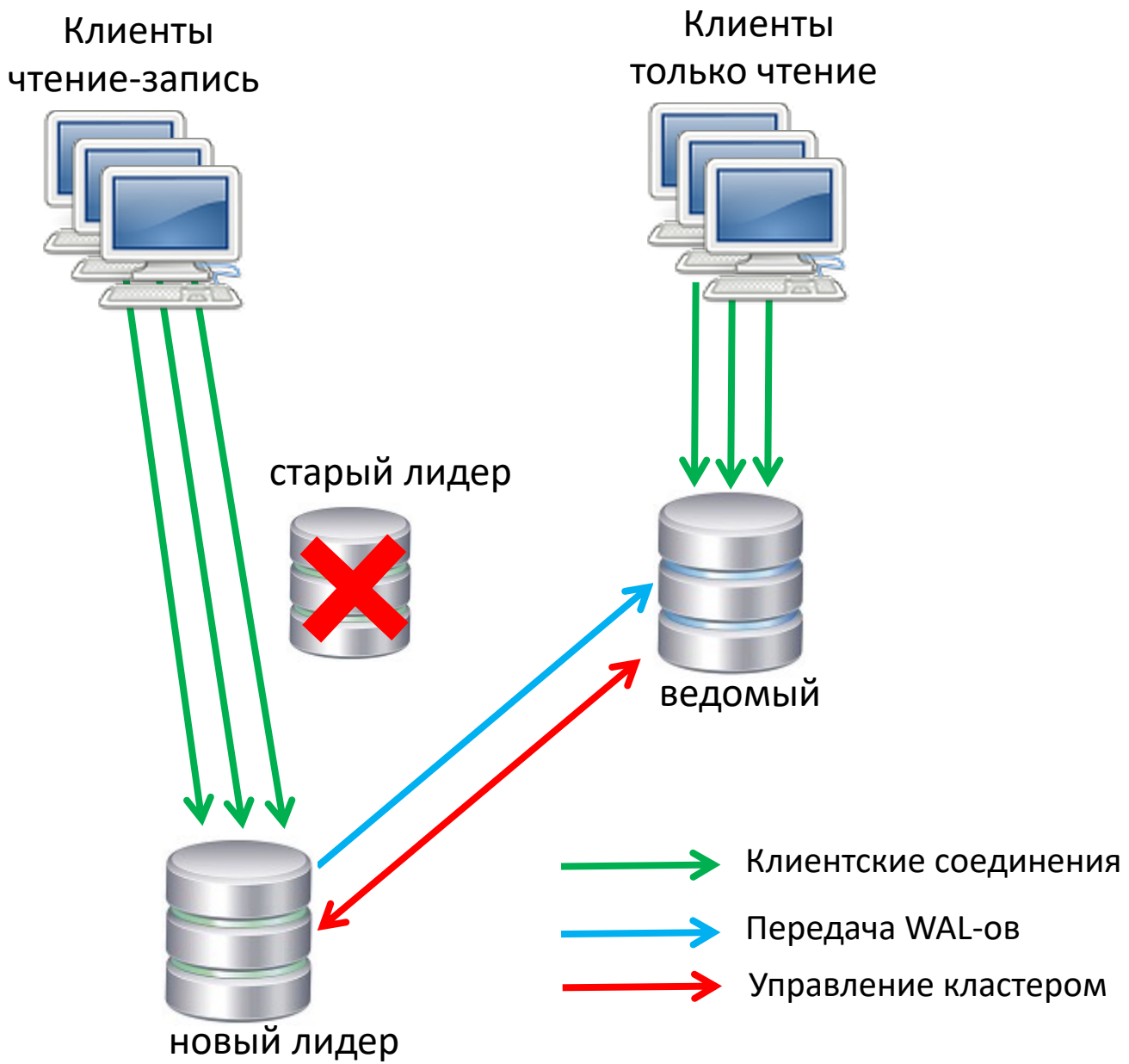
а также указать параметр *target_session_attrs=read-write* .

При сбое узла клиент автоматически подключится к новому лидеру

Встроенный отказоустойчивый кластер BiHA

Отказ лидера

- Автоматическая смена лидера происходит в аварийных ситуациях
- При выходе из строя лидера ведомые организуют процесс голосования для выбора нового лидера.
- Новым лидером становится ведомый узел с максимальным WAL (у него минимум потерь)



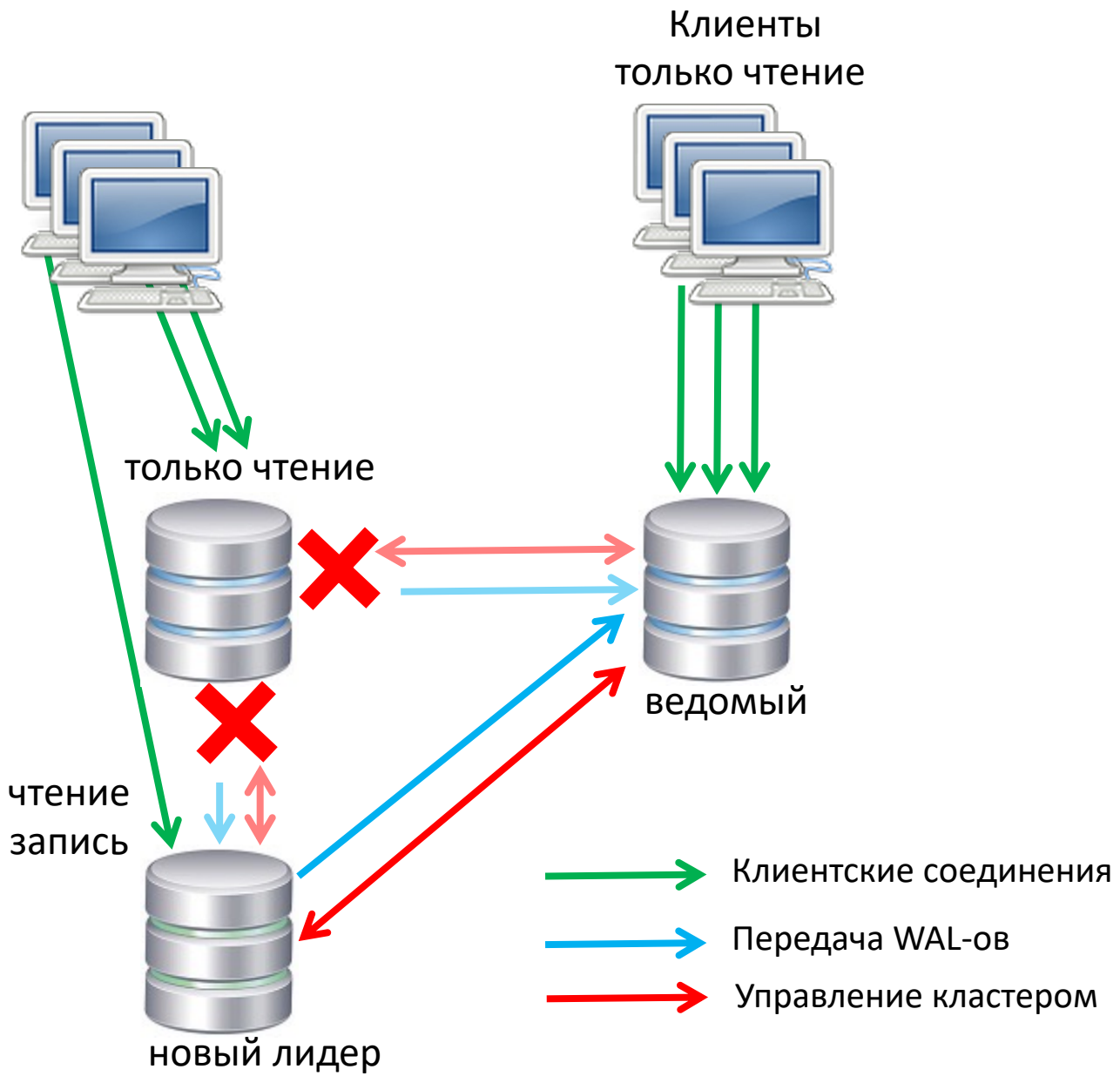
Встроенный отказоустойчивый кластер BiHA

Сетевая изоляция лидера

Когда лидер теряет связь
с необходимым количеством узлов,
лидер переводится в режим только чтение
до разрешения конфликта:

- либо когда восстановится соединение с недостающими узлами,
- либо когда администратор устранит сбой вручную.

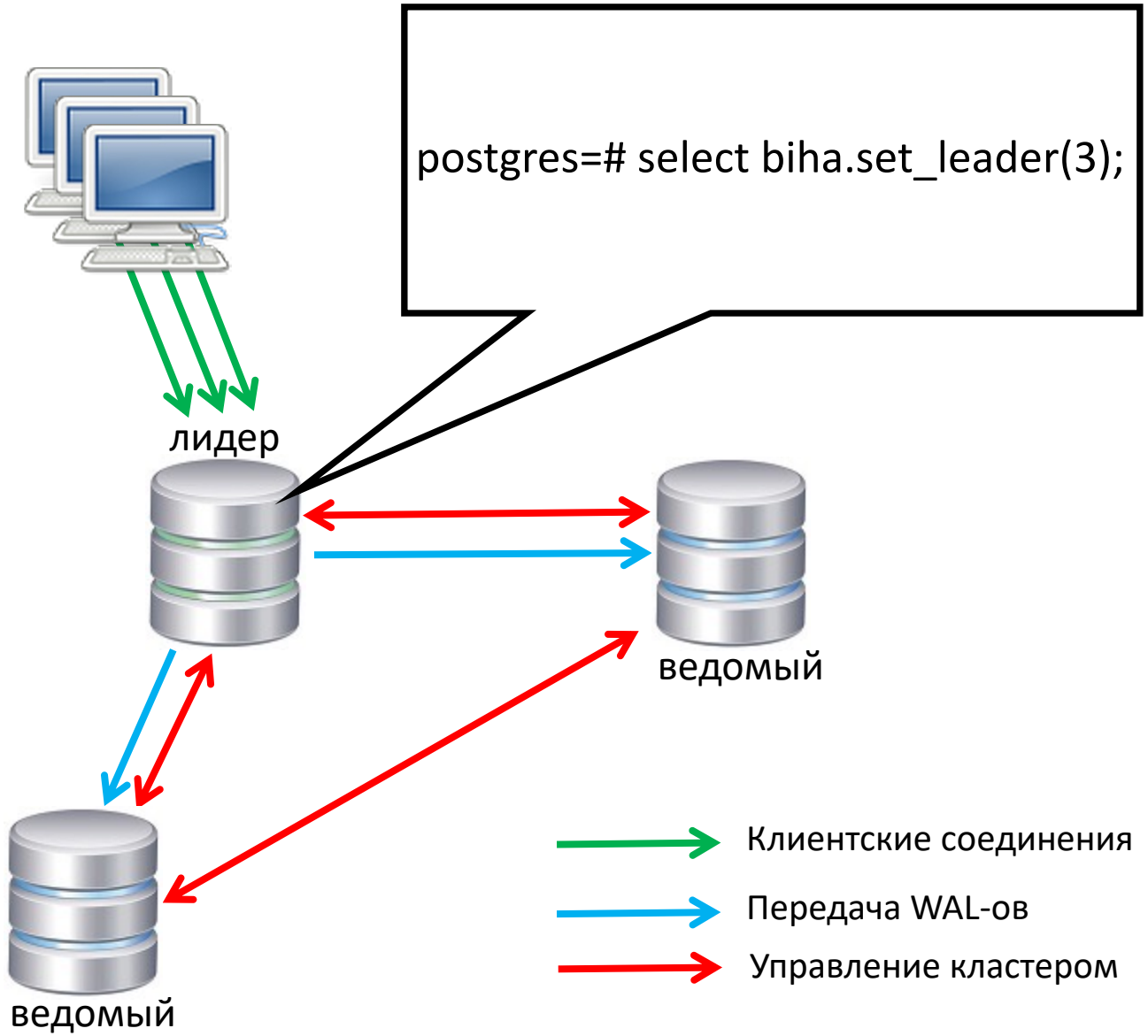
Эта защита обеспечивает запрет на выполнение любых операций , модифицирующих WAL, для предотвращения записи одновременно на несколько лидеров (split-brain).



Встроенный отказоустойчивый кластер BiHA

Назначение лидера вручную

- для перевода лидера в режим обслуживания
- для назначения лидера на предпочтительный хост
- после возврата старого лидера

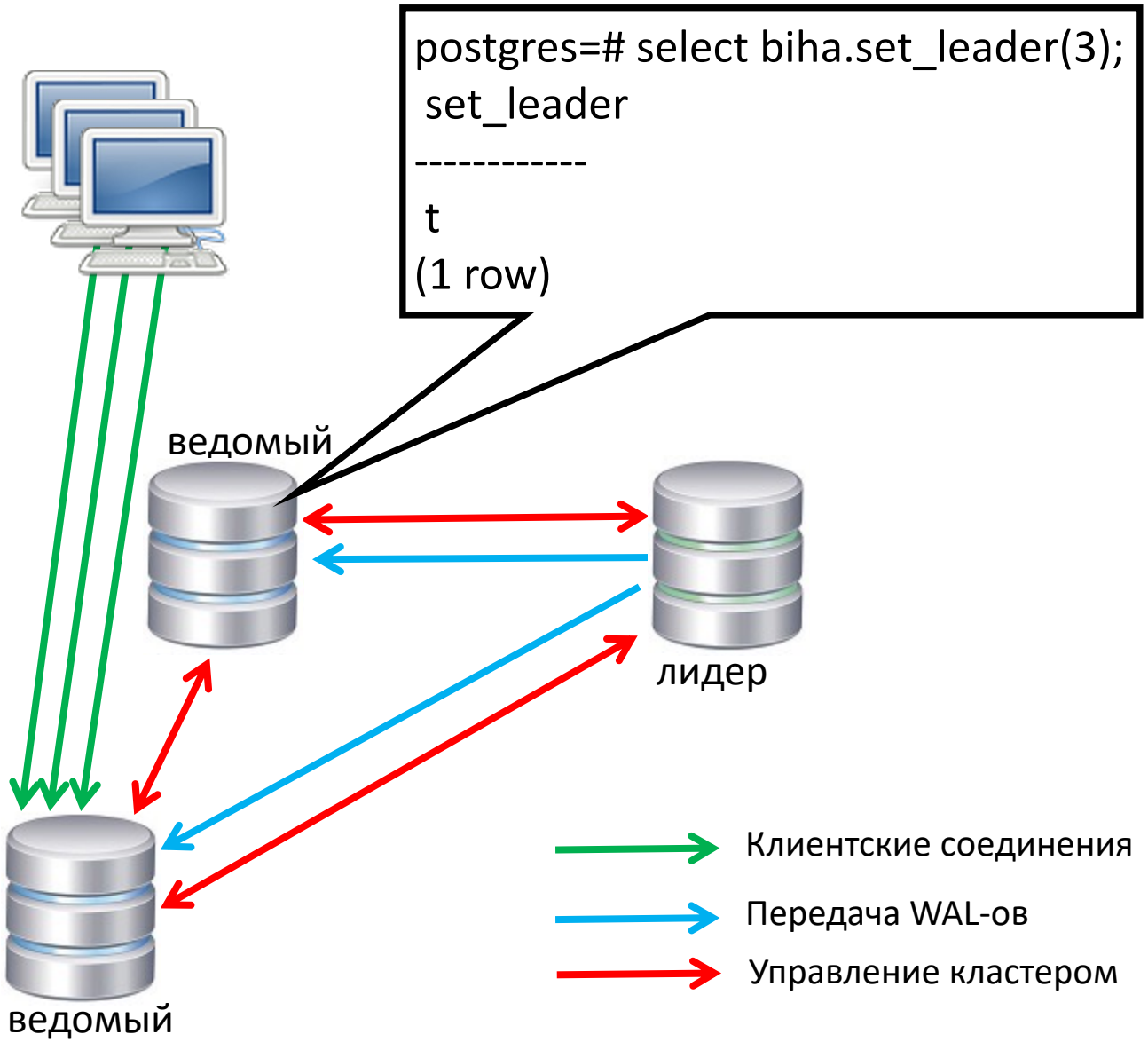


Встроенный отказоустойчивый кластер BiHA

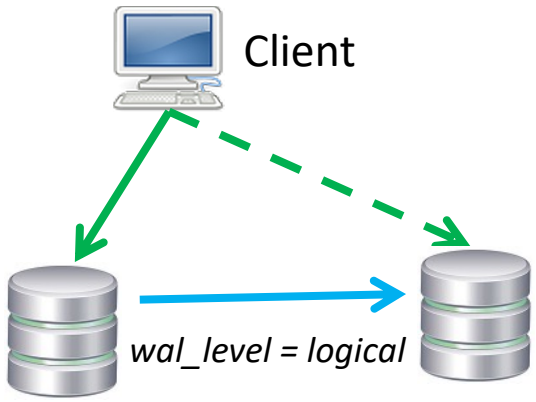
Назначение лидера вручную

Назначение лидера через SQL-интерфейс используя функцию set_leader(id):

- в кластере блокируются все попытки выборов (устанавливается таймаут)
- текущий лидер переключается в режим ведомого
- выбранный узел становится новым лидером
- Если за выделенный таймаут процедура не завершена, выбранный узел становится ведомым, а нового лидера выбирает голосование



Логическая репликация



Логическая репликация — это метод репликации изменений объектов в базе данных, использующий идентификаторы (обычно это первичный ключ), в отличие от физической, которая построена на точных адресах блоков и побайтовом копировании.

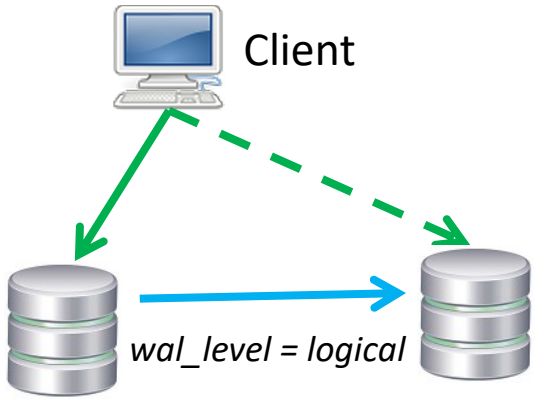
Требуется параметр конфигурации wal_level = logical

Публикация изменения таблиц:
CREATE PUBLICATION FOR TABLE | FOR ALL TABLES | FOR TABLES IN SCHEMA ;

Подписка на изменения таблиц :
CREATE SUBSCRIPTION … CONNECTION 'host=… dbname=…' PUBLICATION … ;

Логическая репликация начинается с создания снимка в публикуемой базе данных и копирования её подписчику. После этого изменения передаются подписчику в реальном времени. Подписчик применяет изменения в том же порядке, что и узел публикации, так что в рамках одной подписки гарантируется транзакционная целостность.

Логическая репликация

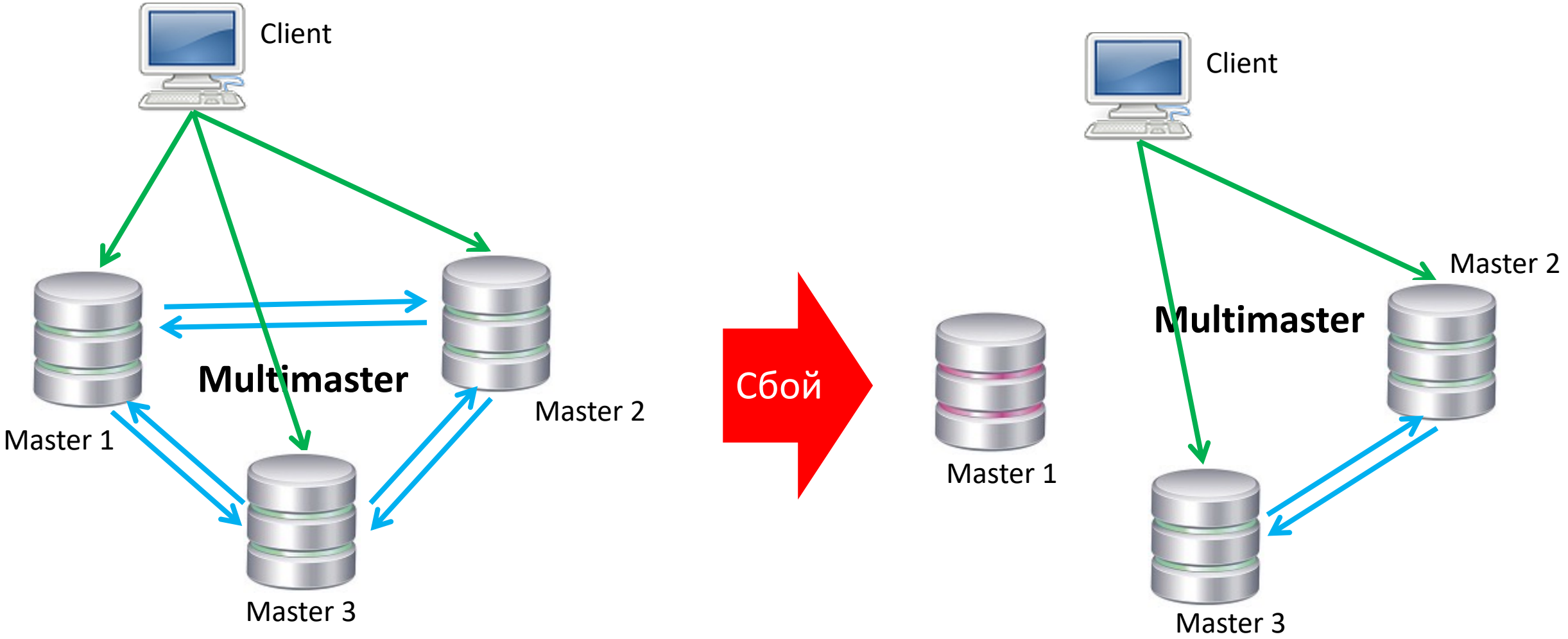


Типичные сценарии использования логической репликации:

- Передача подписчикам инкрементальных изменений в одной базе данных или подмножестве базы данных, когда они происходят.
- Срабатывание триггеров для отдельных изменений, когда их получает подписчик.
- Объединение нескольких баз данных в одну (например, для целей анализа).
- Репликация между разными основными версиями PostgreSQL.
- Репликация между экземплярами PostgreSQL на разных платформах (например, с Linux на Windows)
- Предоставление доступа к реплицированным данным другим группам пользователей.
- Разделение подмножества базы данных между несколькими базами данных.
- Обновление на новую версию

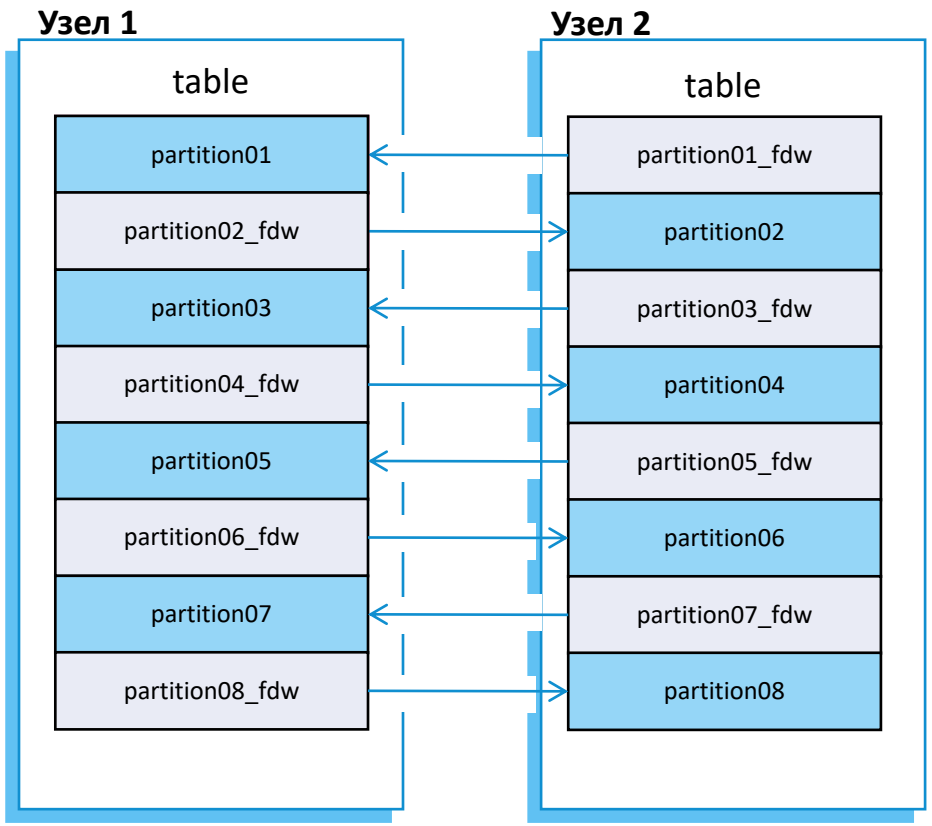
Мультимастер

PostgresPro



- Узлы открыты на чтение/запись
- 3 фазная фиксация изменений – замедляет работу
- Разрешение конфликтов
- Быстрое переключение

Shardman



- Распределенная СУБД PostgreSQL
  - PostgreSQL + CFS, PTRACK
  - Расширение Shardman
  - Расширение postgres_fdw
- Shardmanctl – управление
- Etcd – хранение состояния и конфиурации
- Shardmand – отказоустойчивость

bookings

book_ref	book_date	amount
12345	01.01.2023	300
34567	20.03.2023	1000
56789	13.07.2023	1300

tickets

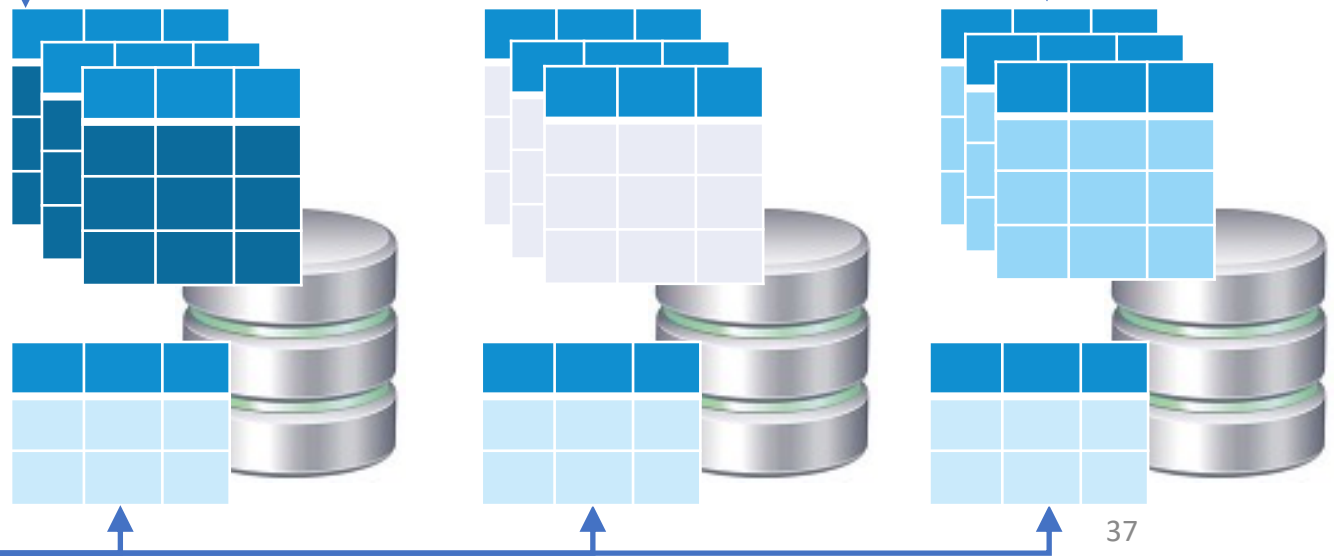
book_ref	ticket_no	passenger_name
12345	12345678	Андрей
34567	34567890	Сергей
56789	56789123	Иван
34567	34567891	Екатерина
56789	56789124	Анна

ticket_flights

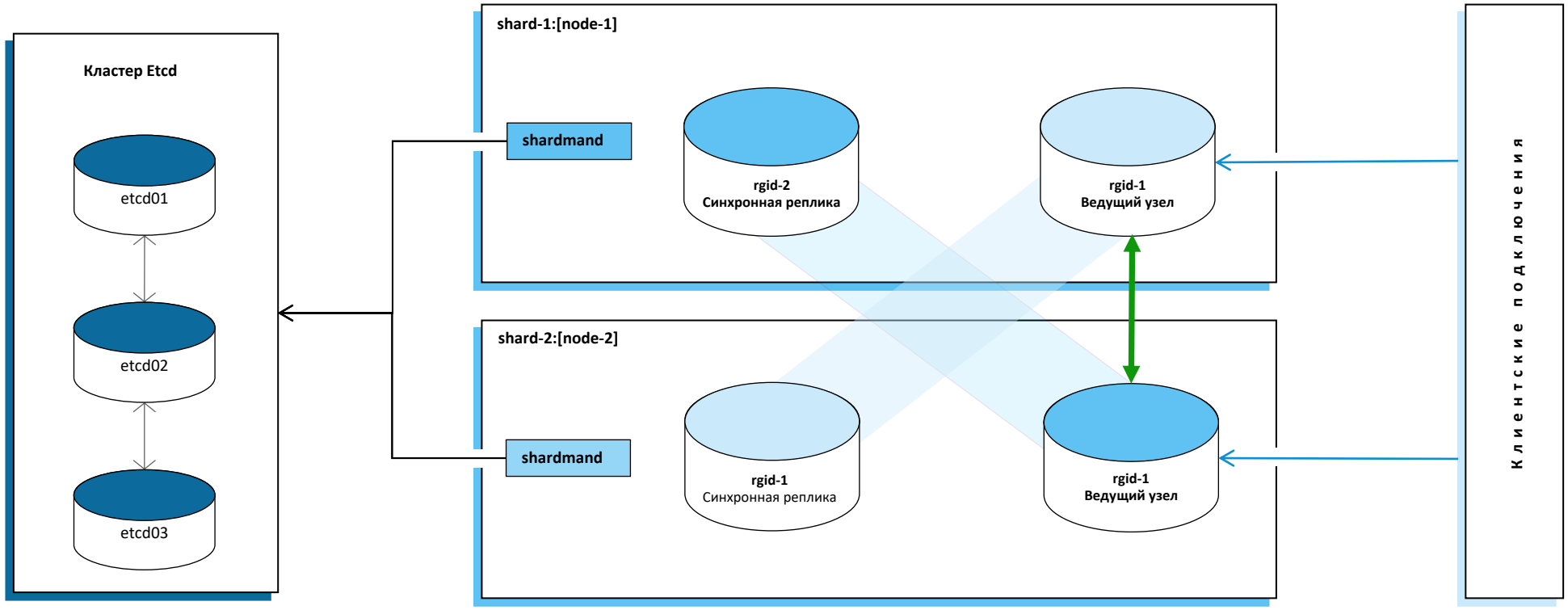
book_ref	ticket_no	flight_id
12345	12345678	U755
34567	34567890	F444
56789	56789123	R234
34567	34567891	F444
56789	56789124	R234

flights

flight_id	departure_airport	arrival_airport
U755	Москва	Самара
F444	Владивосток	Москва
R234	Самара	Сочи



Отказоустойчивость Shardman



PostgresPro

Спасибо за внимание!

PostgresPro

Q&A