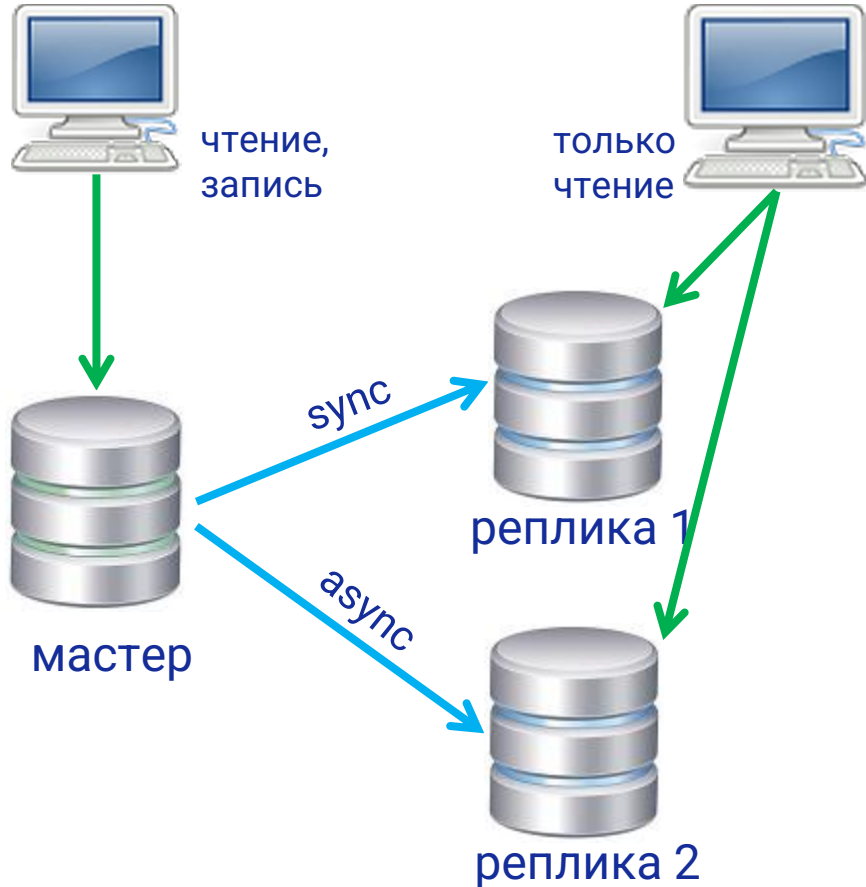


Postgres Pro BiHA & Proxima



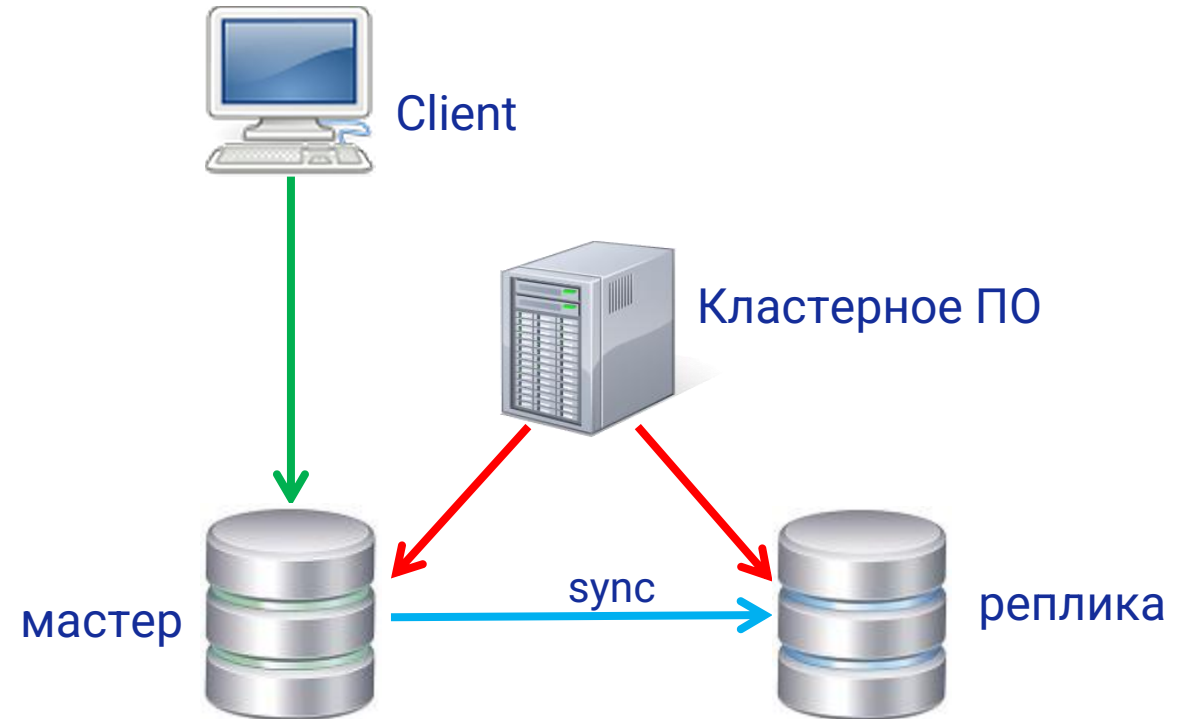
Postgres Pro : Физическая репликация



- 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

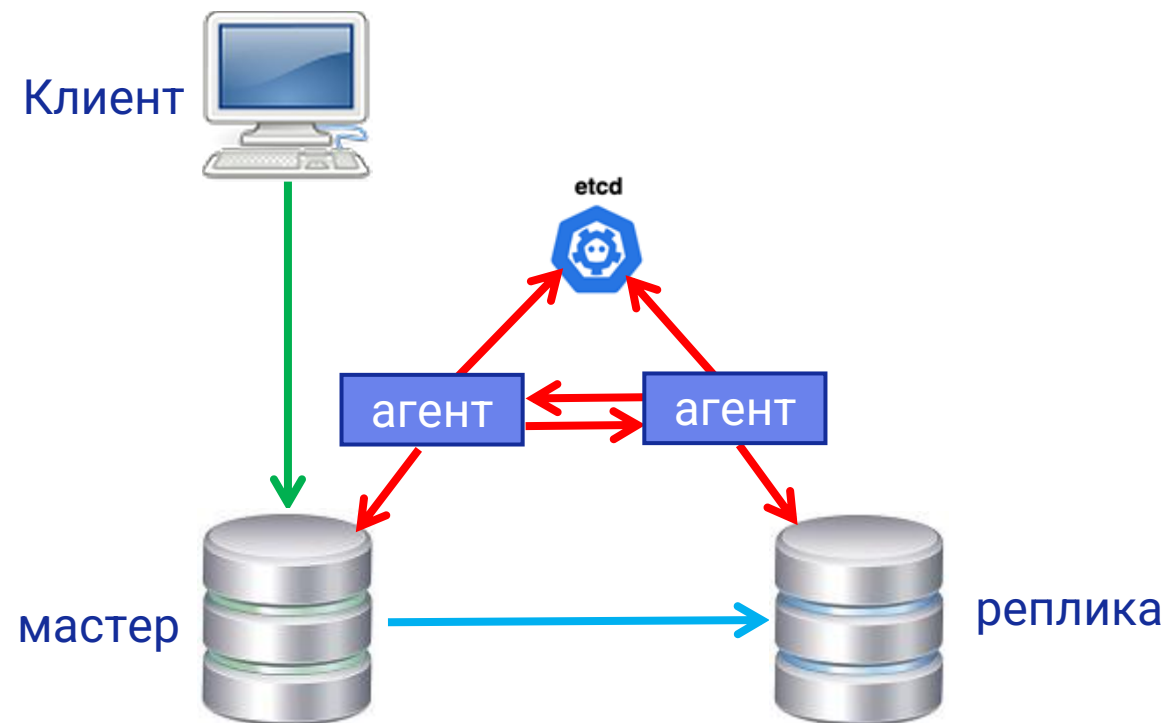
- Решение о смене ролей в отказоустойчивом кластере при сбое мастера может приниматься автоматически
- Необходимо также автоматически переключить на новый мастер и клиентов
- Основная задача кластерного ПО обнаружить сбой, сменить роль реплики на новый мастер, но при этом не допустить работу двух узлов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



Примеры кластерного ПО : Patroni, Stolon, Corosync + Pacemaker

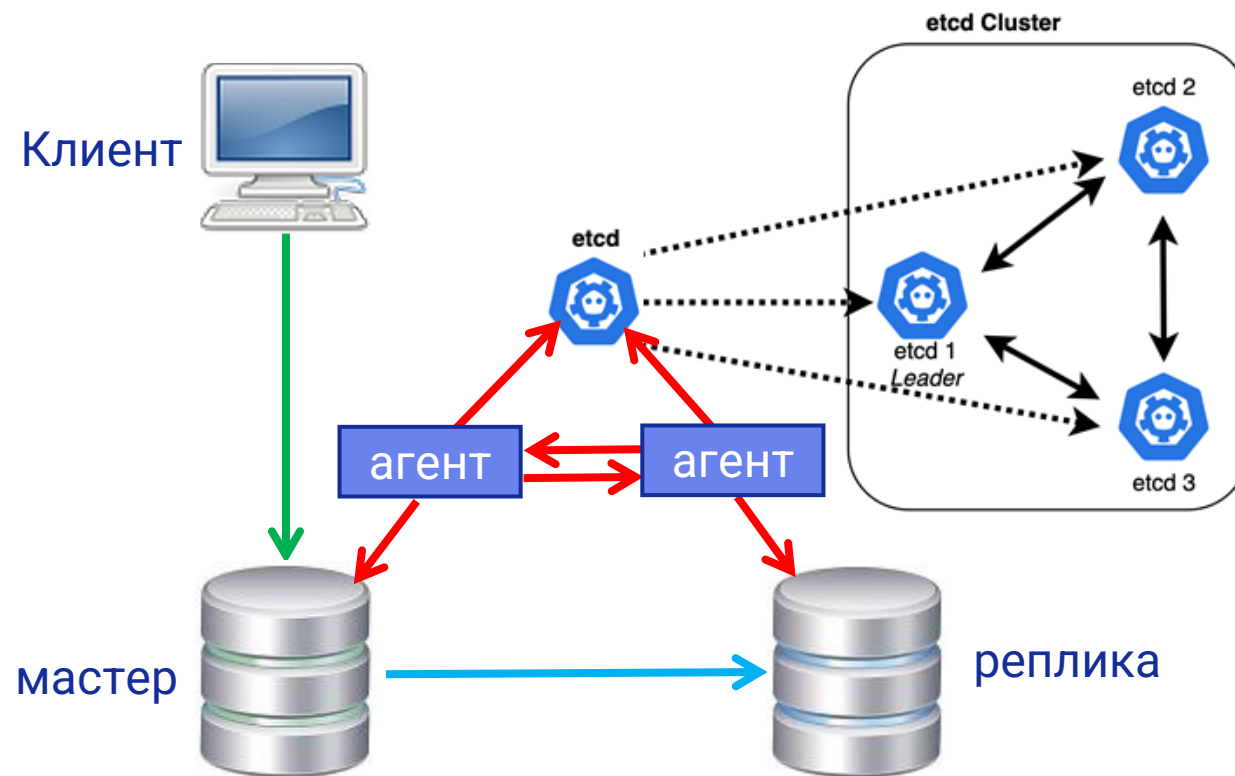
Недостатки внешнего кластерного ПО

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



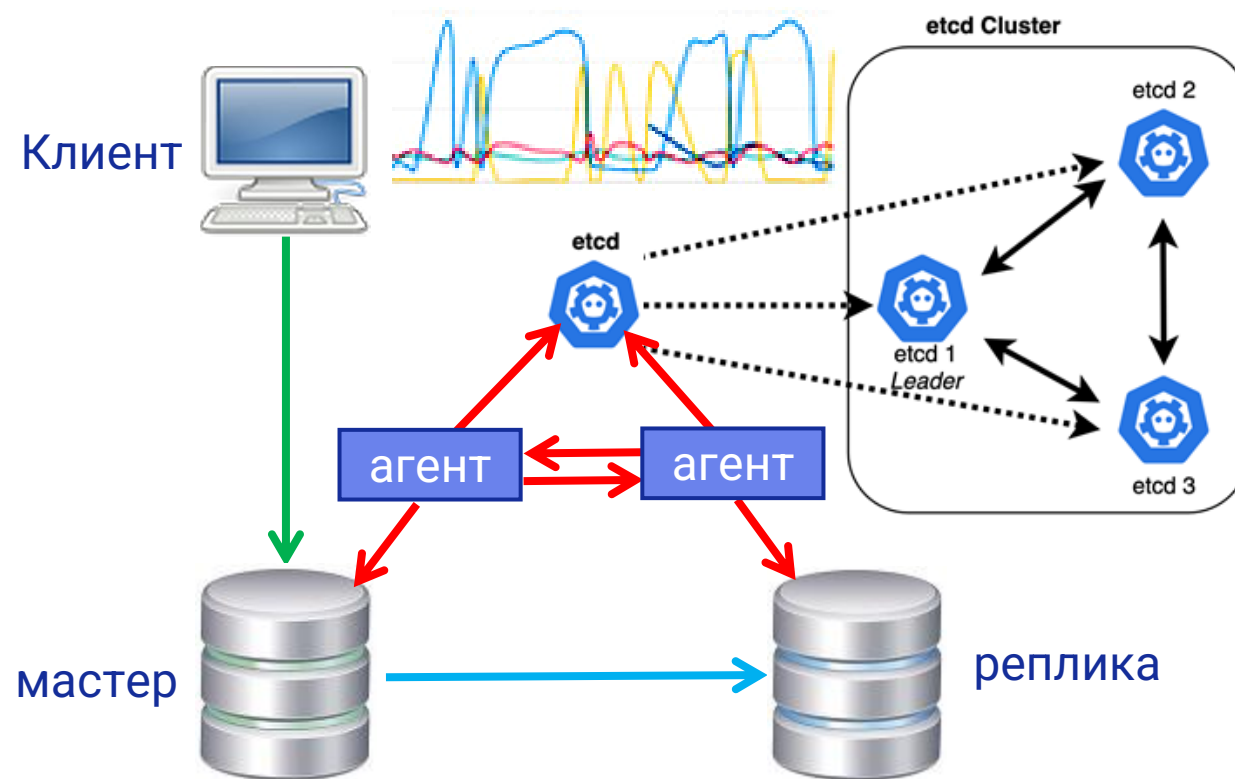
Недостатки внешнего кластерного ПО

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



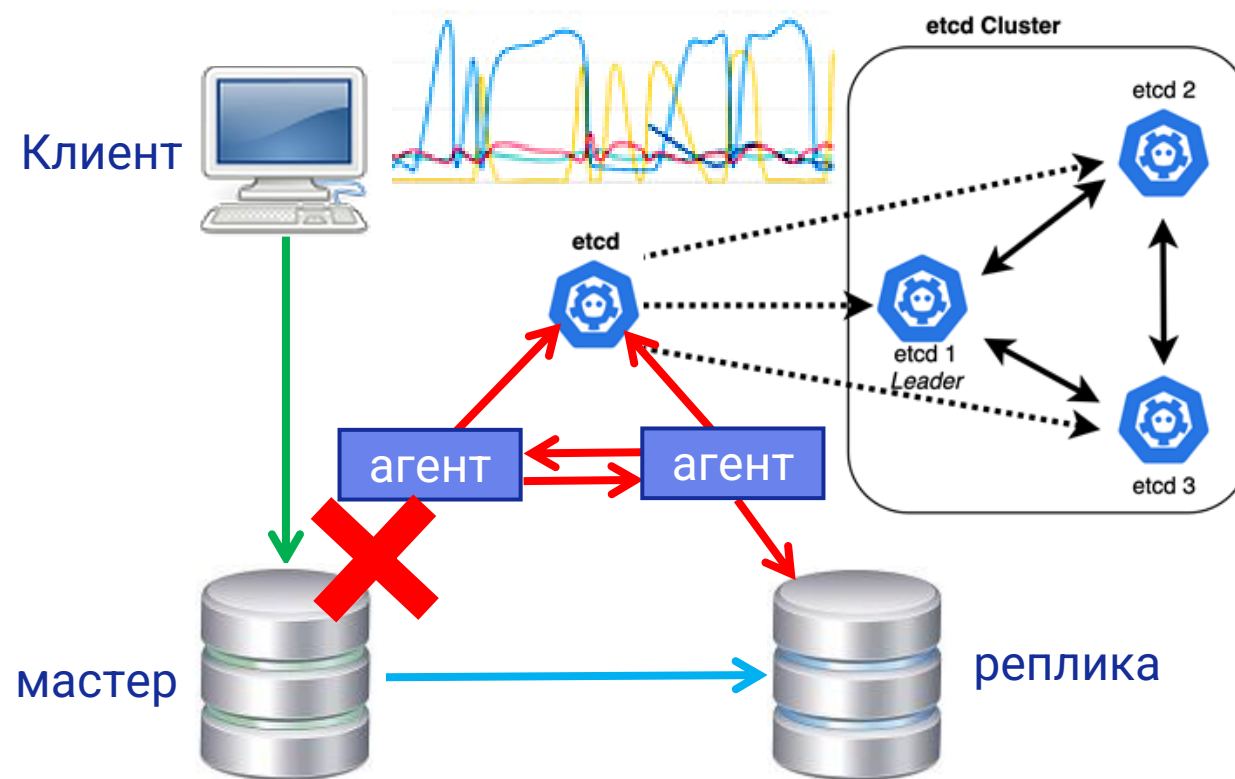
Недостатки внешнего кластерного ПО

- 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

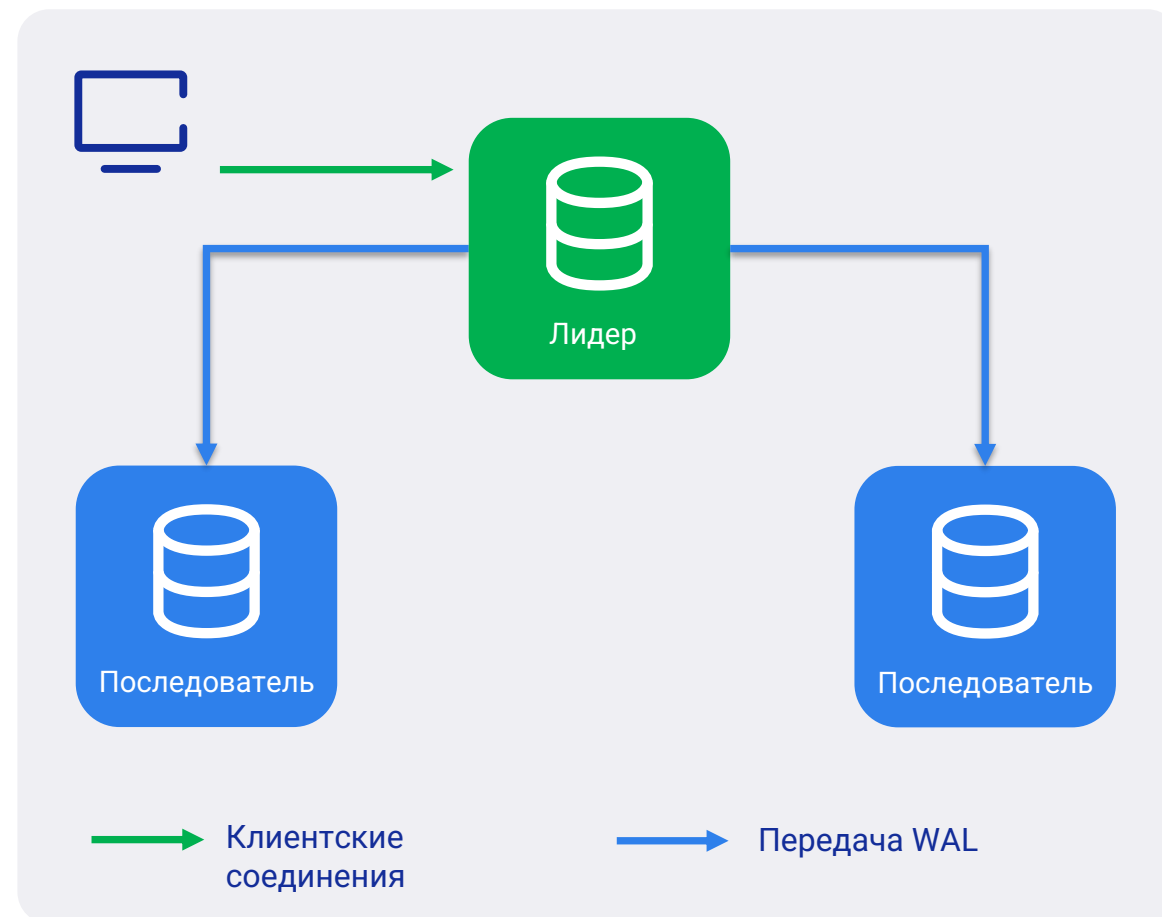


BiHA — Built-in High Availability

Кластер с физической репликацией и встроенным аварийным переключением узлов, отказоустойчивостью и автоматическим восстановлением после отказа узлов.

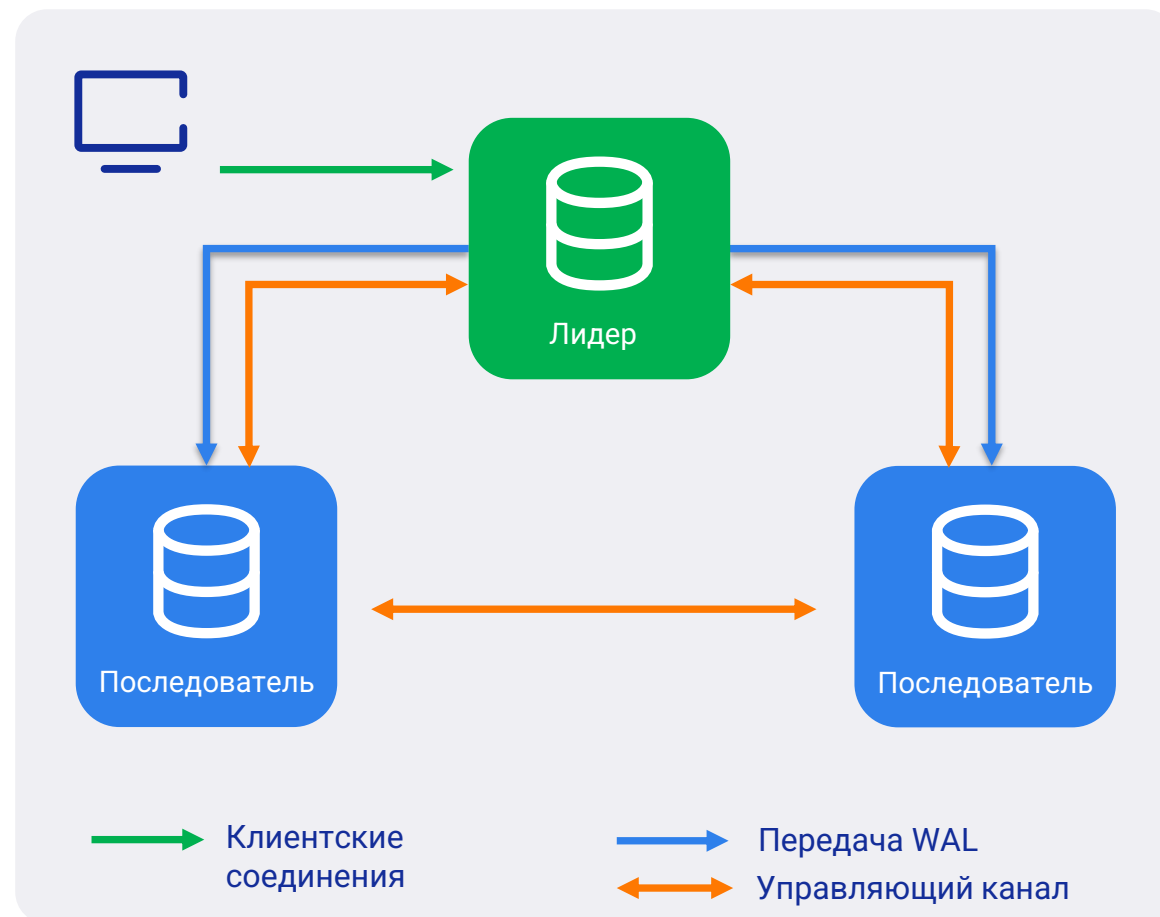
Встроен в Postgres Pro:

- в Enterprise, начиная с версии 16.
- в Standard в 17 и 18.



BiHA – Built-in High Availability

- 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.

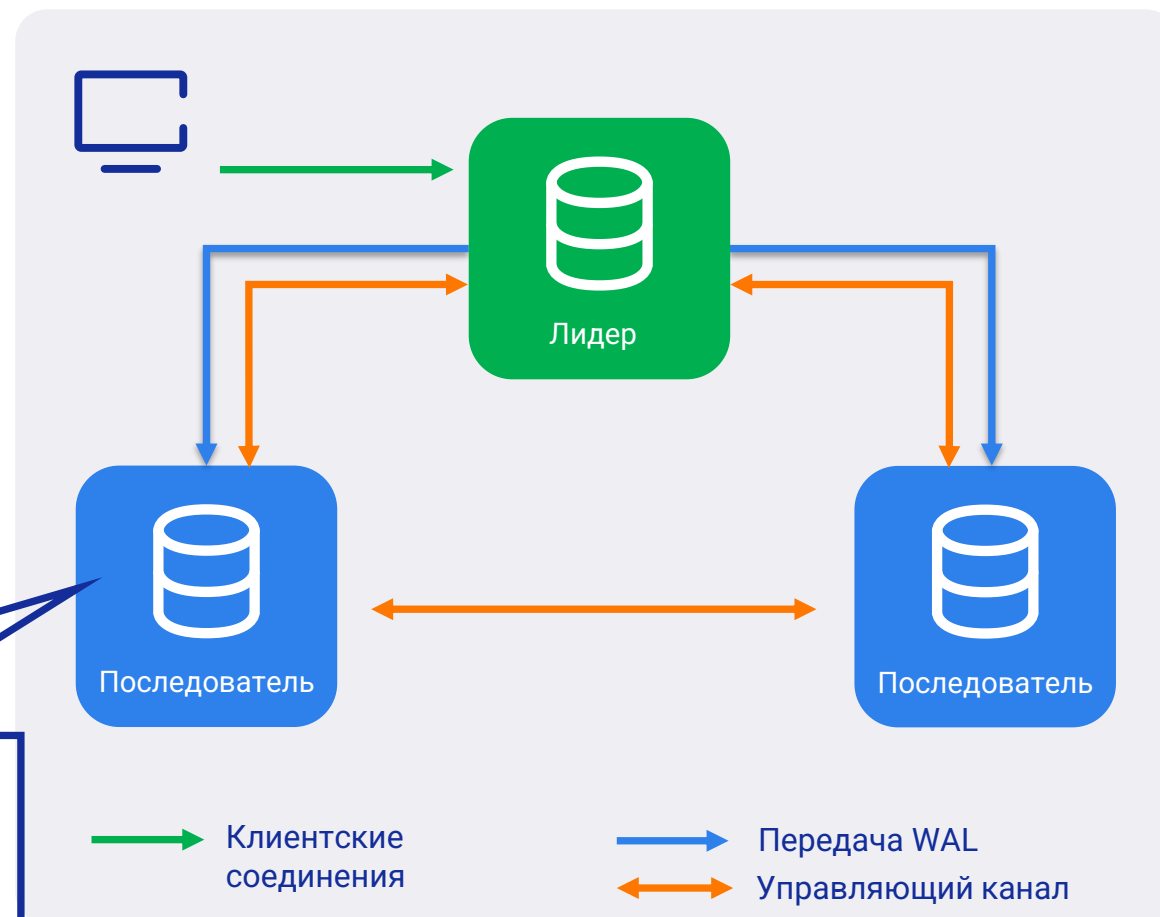


BiHA – Built-in High Availability

Встроен в Postgres Pro:

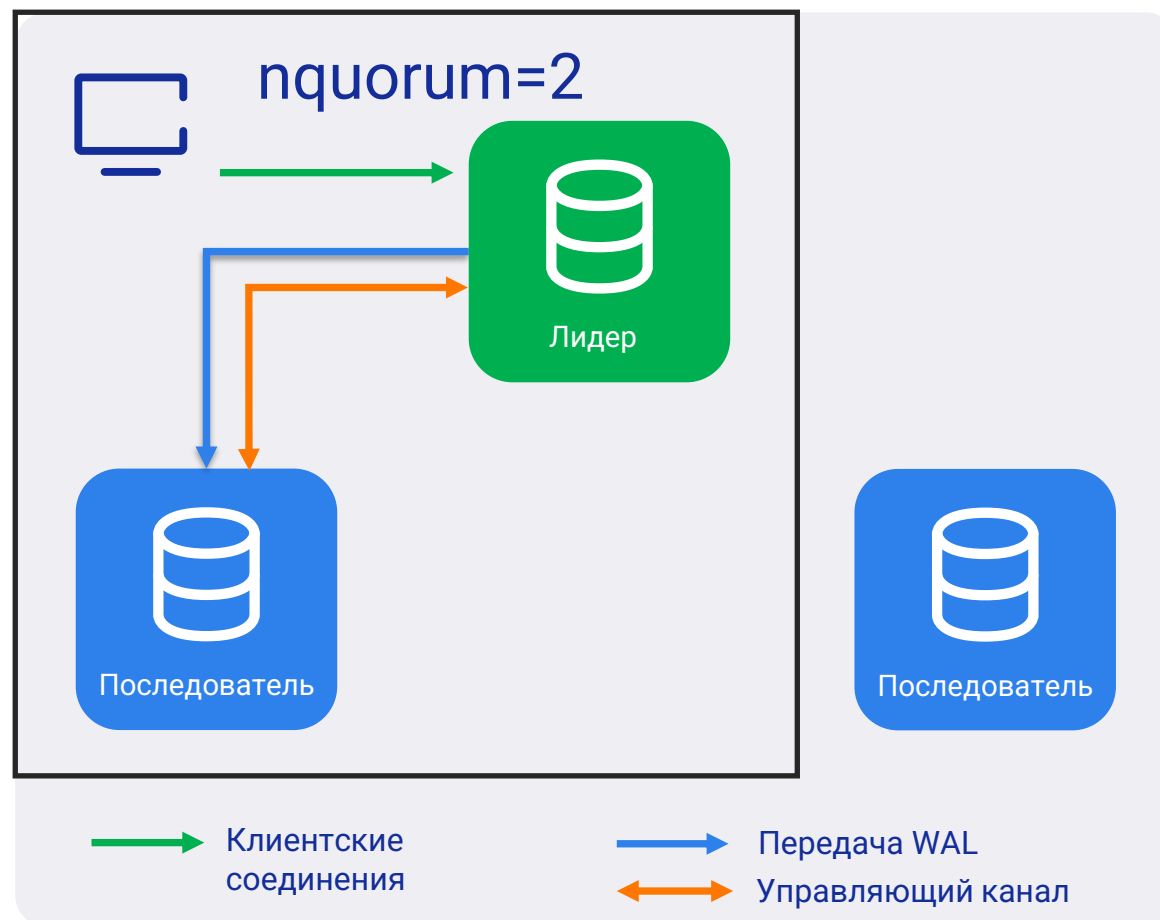
- Простая установка и конфигурирование
- Не требуется установка дополнительного ПО
- Оперативные обновления версий

```
# apt-get install postgrespro-ent-18  
$ bihactl init | add | --convert-standby ...
```



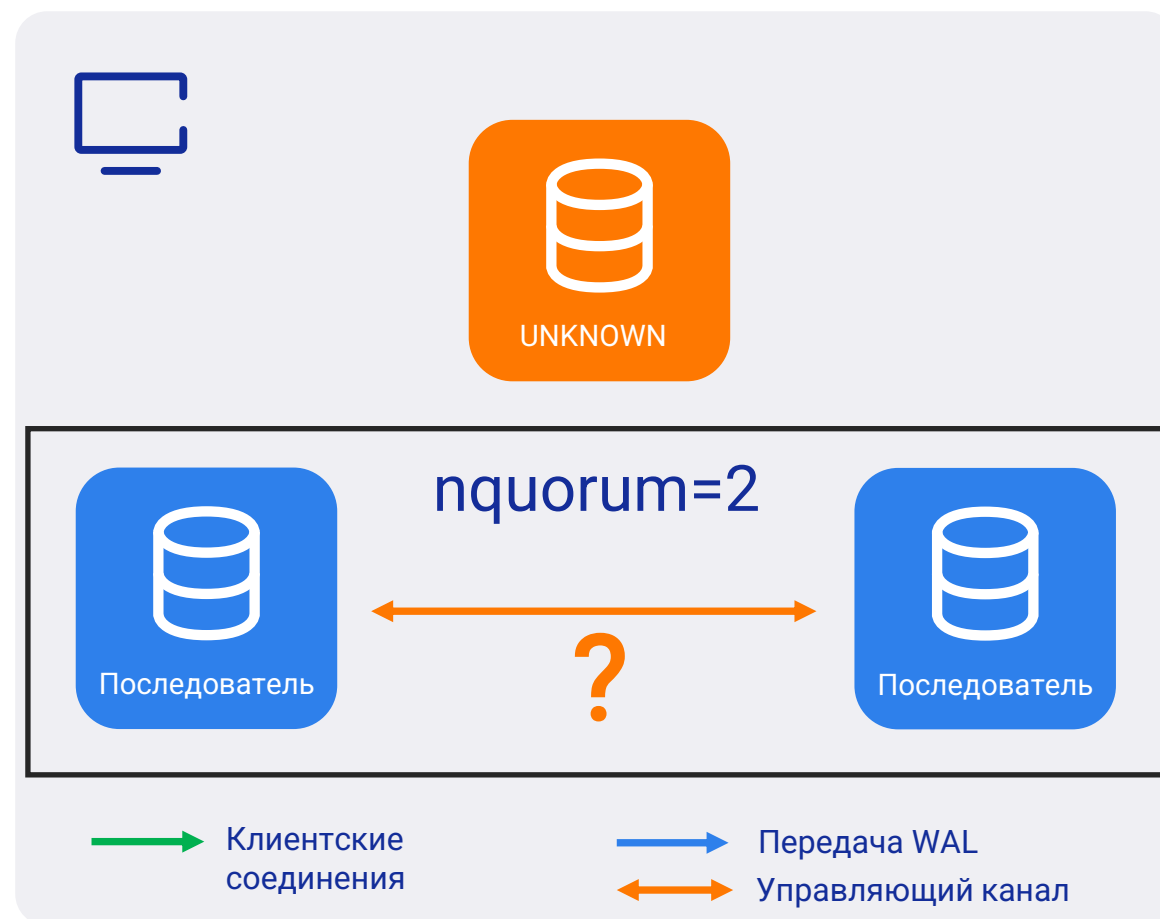
ViHA : Кластерный кворум

- Кворум определяет минимальное количество узлов кластера
- Лидер продолжает работать, если соблюдается кворум



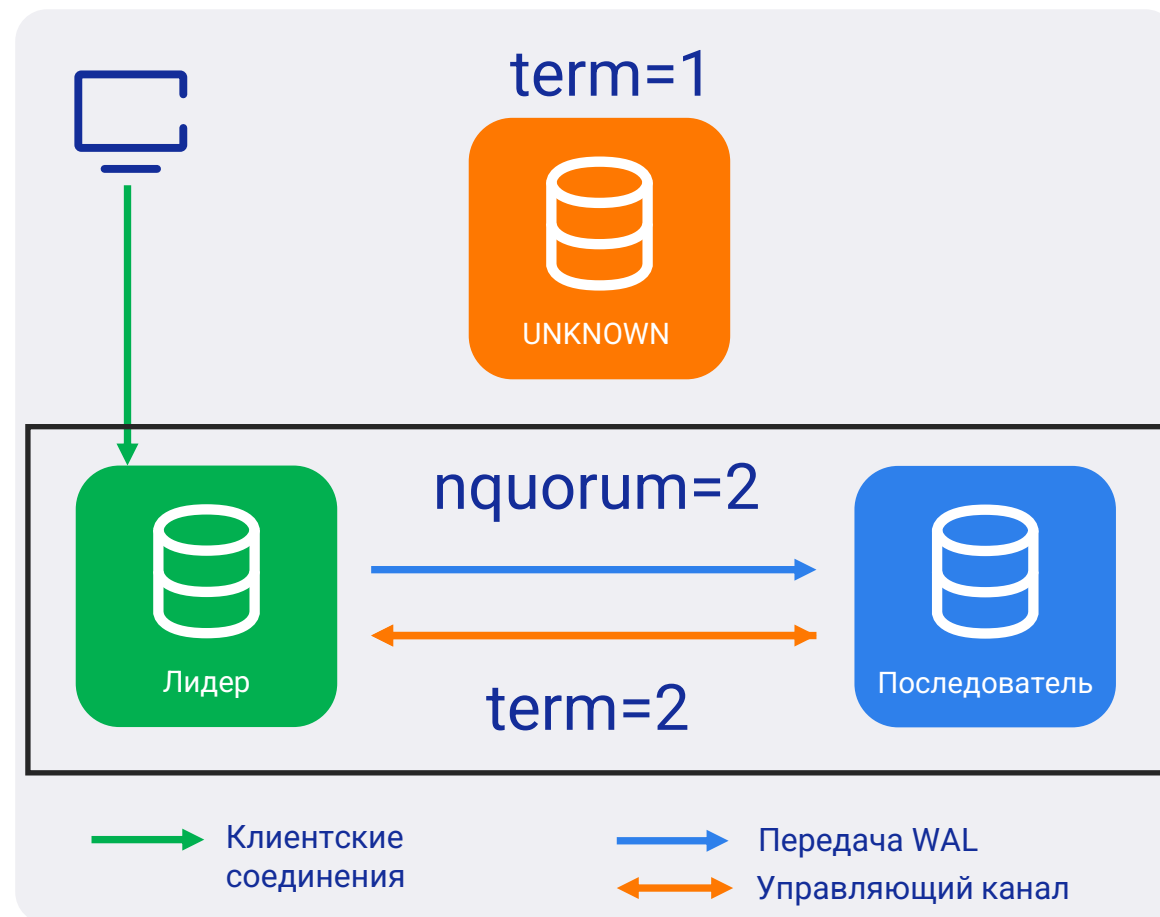
ВiHA : Кластерный кворум

- Лидер не может продолжать работу, если не соблюдается кворум
- Ведомые организуют выборы нового лидера, если кластер содержит достаточное количество узлов



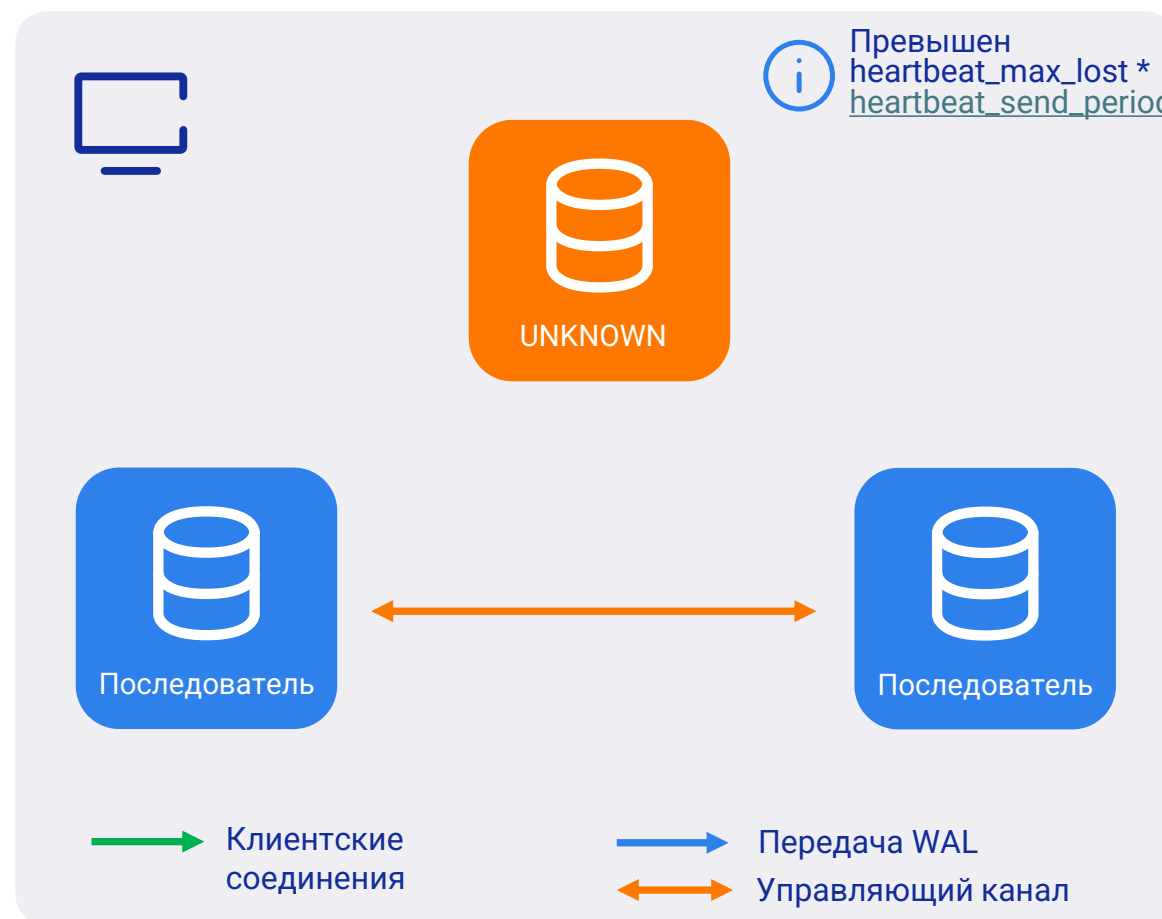
ВiHA : поколение кластера

- После выбора нового лидера в кластере меняется поколение
- Старый лидер остаётся в старом поколении
- При возвращении старого лидера в кластер он не может быть уже лидером и переходит в режим ведомого



ViHA : время обнаружения сбоя

- `biha.heartbeat_max_lost` - максимальное число сообщений о контроле состояния, которые можно не получить до того, как узел будет считаться недоступным (10 по умолчанию)
- `biha.heartbeat_send_period` - частота отправки сообщений о контроле состояния в миллисекундах (1000 по умолчанию)



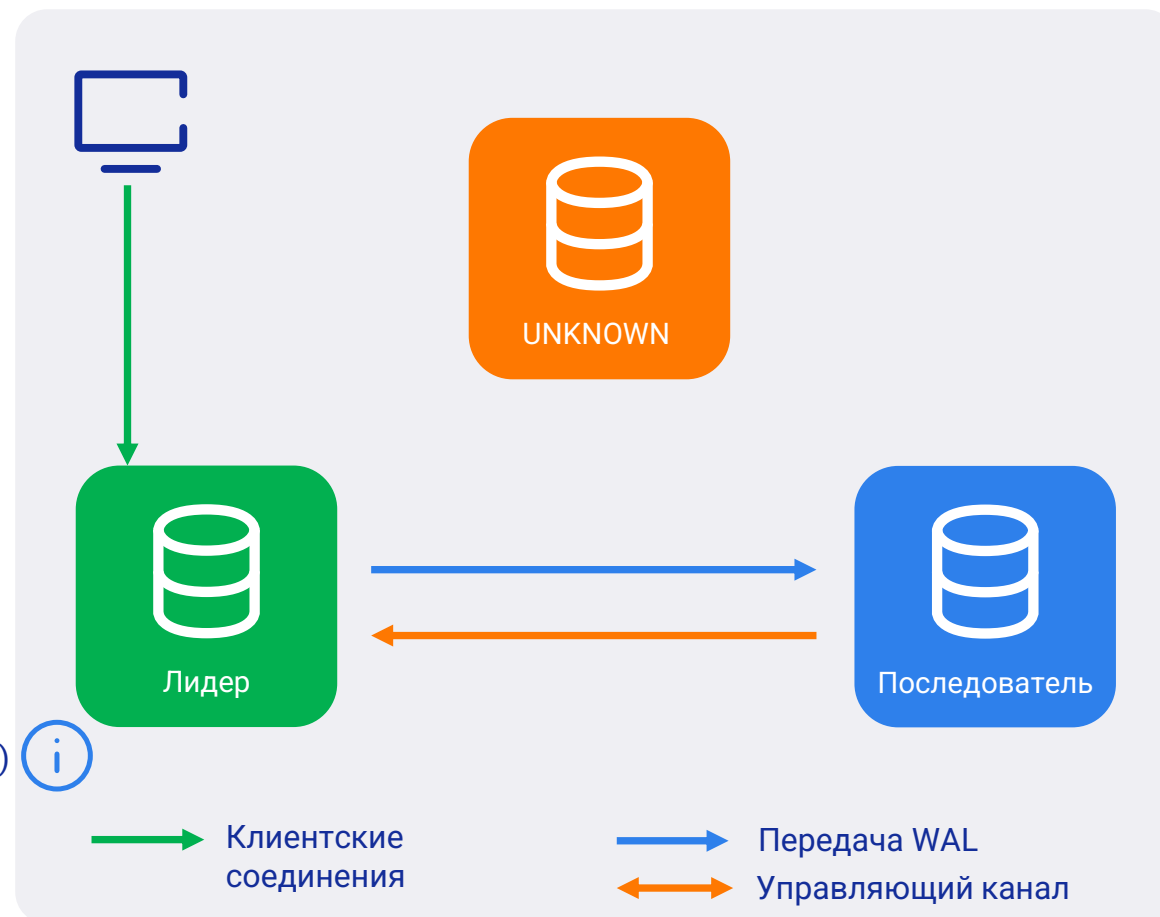
ВiНА : выборы нового лидера

В случае сбоя ведомые предлагают себя кандидатами и организуются выборы нового лидера.

Защита от split-brain:

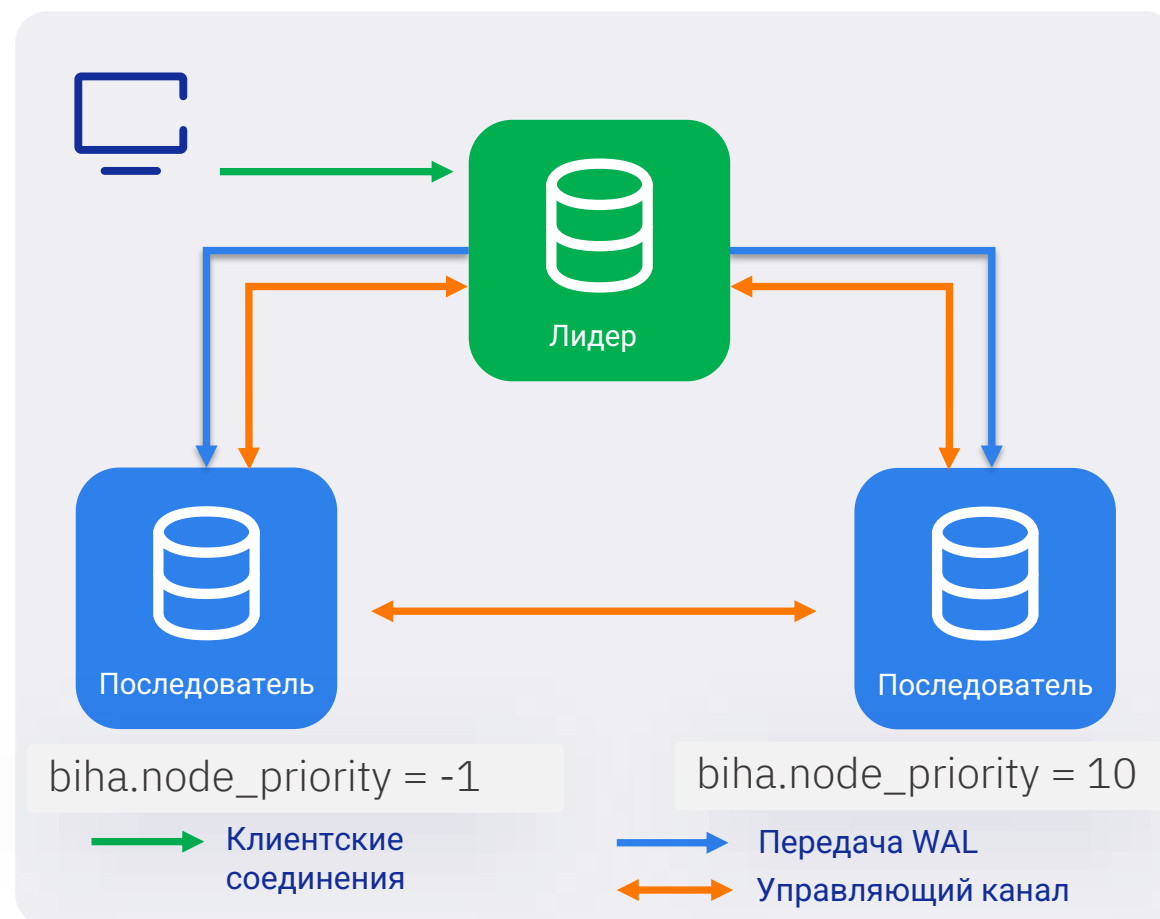
- Лидер продолжает операции RW только при наличии Последователей.
- Голосование проходит при наличии кворума ($nquorum$).

Соблюдается кворум ($nquorum = 2$) и последователь с наибольшим WAL выбран Лидером



ВiНА : приоритеты выбора нового лидера

- Можно установить приоритеты для выдвижения узла в качестве кандидата :
параметр `biha.node_priority`
- Приоритет узла в секундах определяет тайм-аут, по достижении которого узел предложит себя в качестве кандидата на выборах.
- Только в синхронном кластере



ViHA : функции-обработчики смены состояний

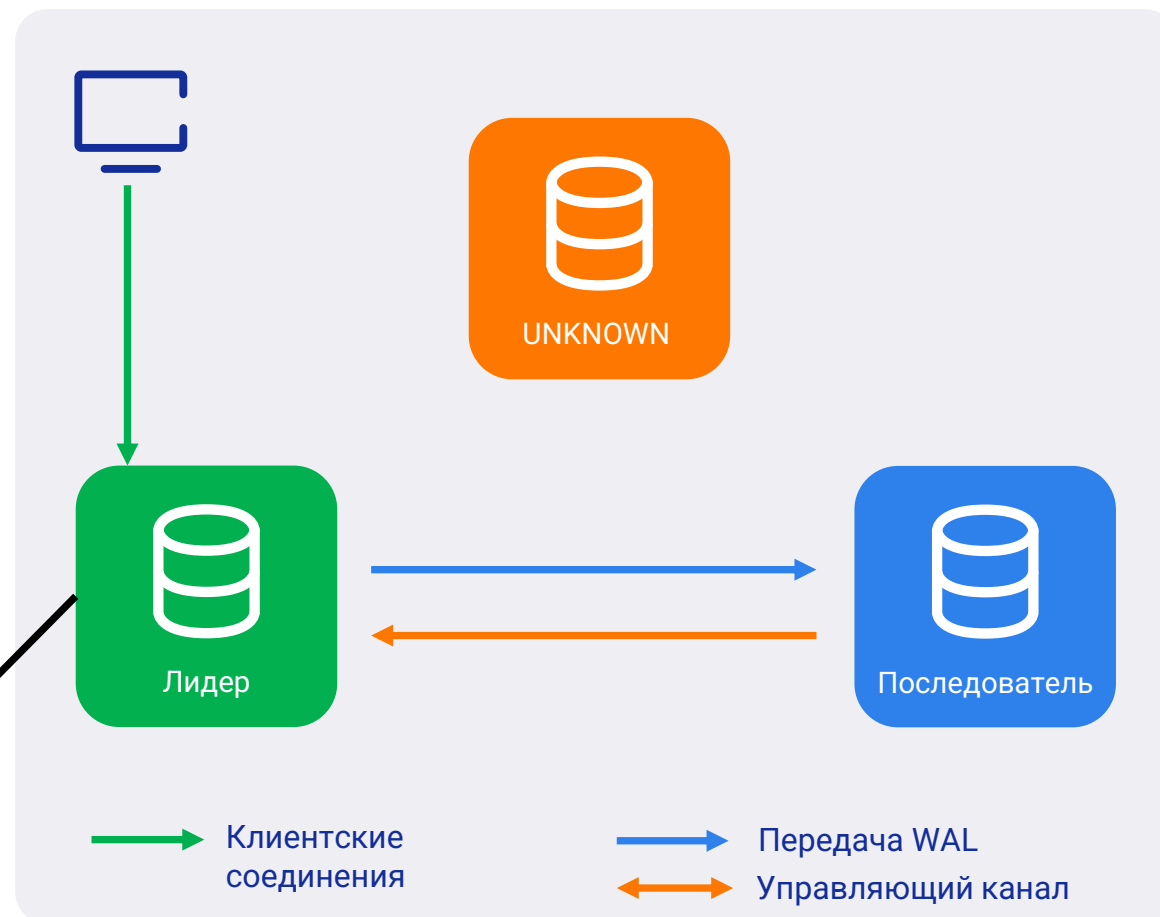
Вы создаёте SQL-функцию и регистрируете её в качестве обработчика. SQL-функция может уведомлять внешние сервисы о событиях в ViHA-кластере.

Например:

- `CANDIDATE_TO_LEADER` вызывается на узле, выбранном в качестве нового лидера.
- `LEADER_CHANGED` вызывается на каждом узле ViHA-кластера при смене лидера.

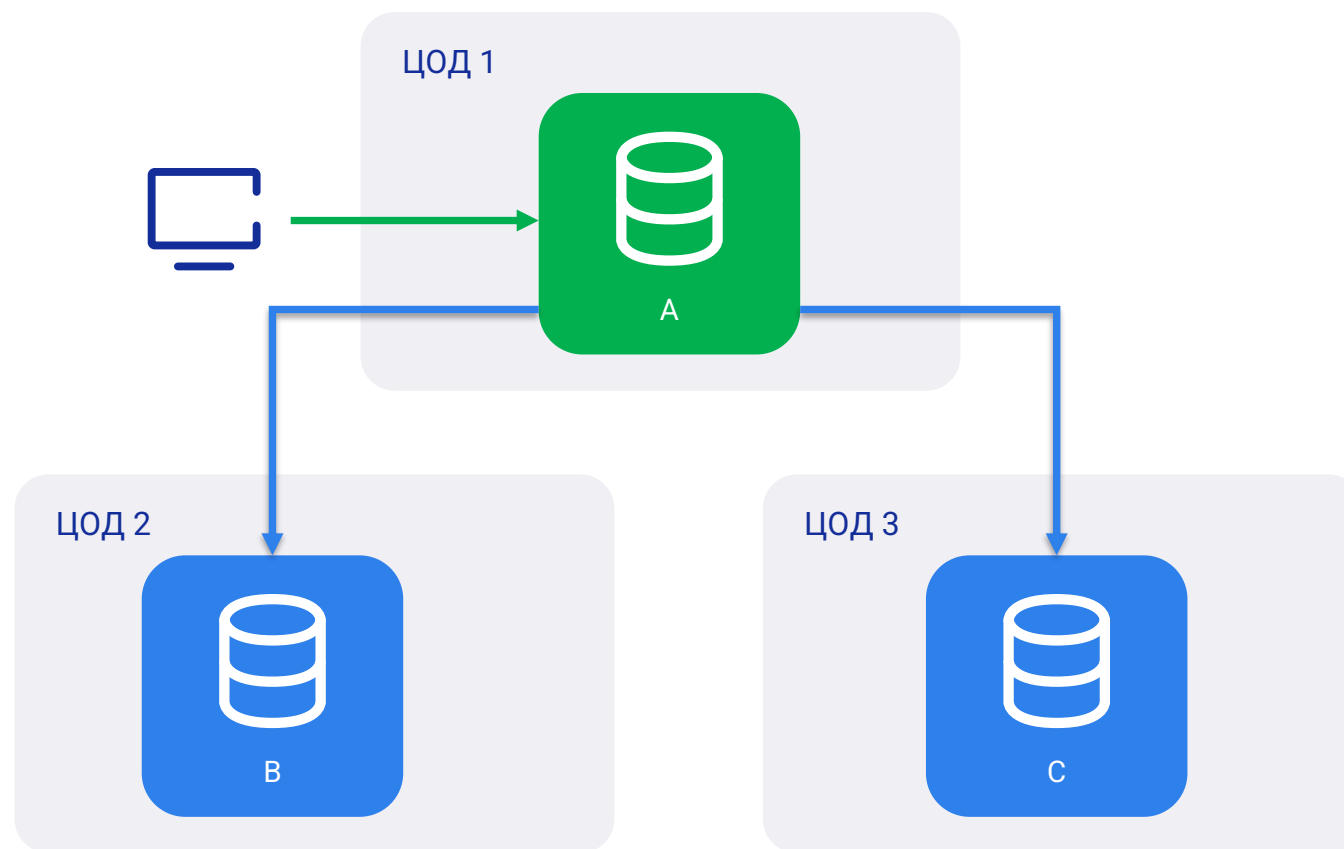
Если исполнение функции-обработчика длится дольше, чем `biha.callbacks_timeout`, ViHA останавливает исполнение и продолжает работать в обычном режиме.

ALERT 



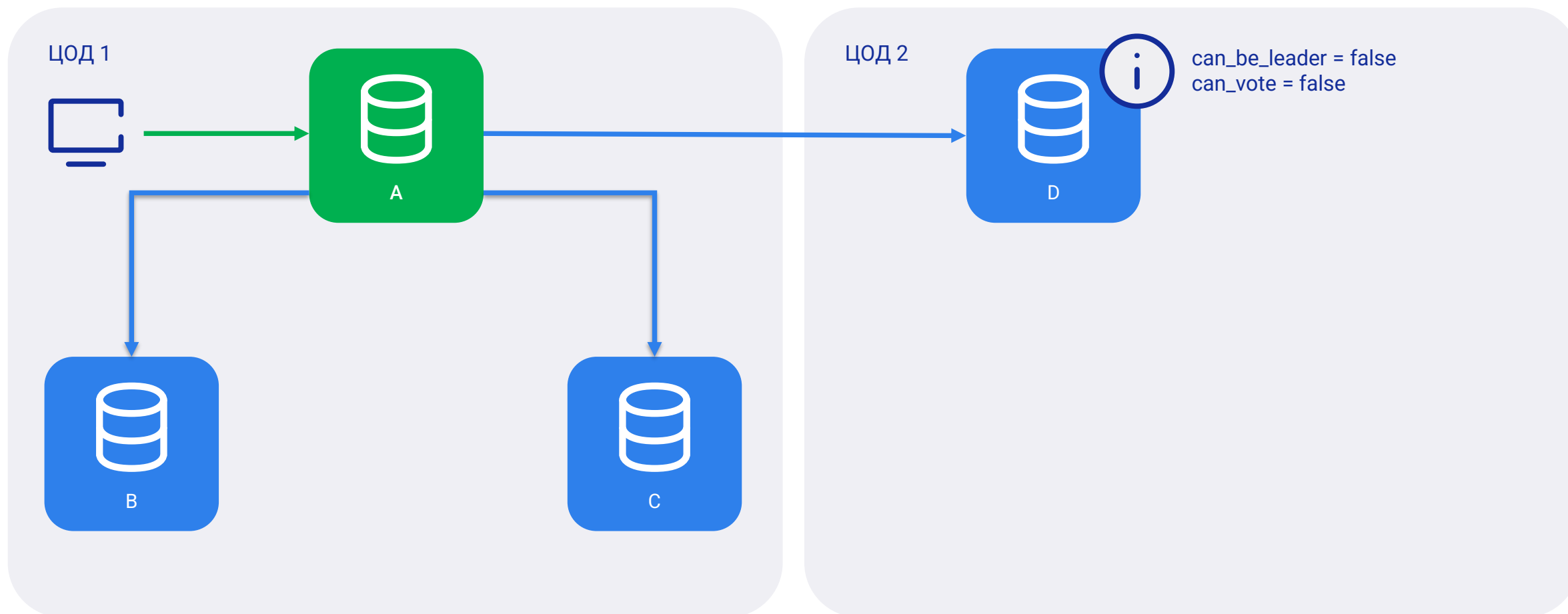
Георезерв

- ✓ Узлы кластера ВiНА могут располагаться в удаленных ЦОД.



Георезерв

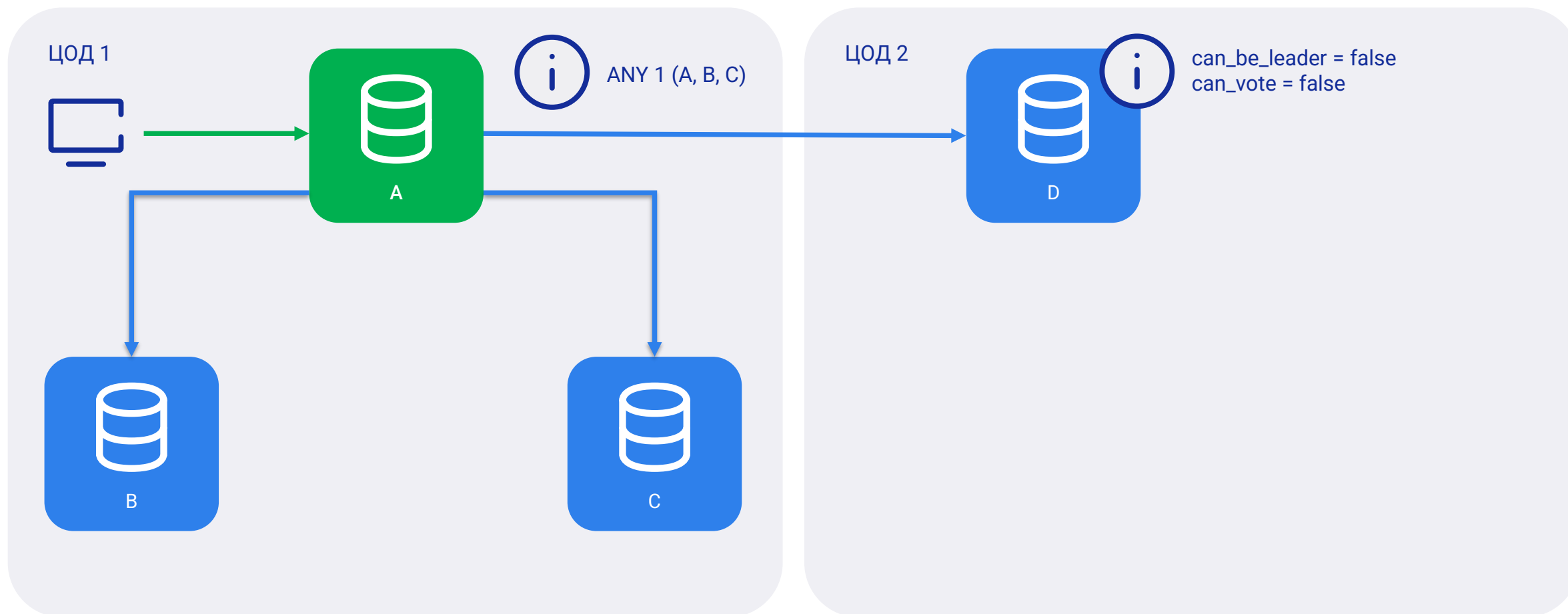
- ✓ Узлы в удаленном ЦОД не голосуют и не выдвигают себя в Лидеры.



Георезерв : список синхронных реплик

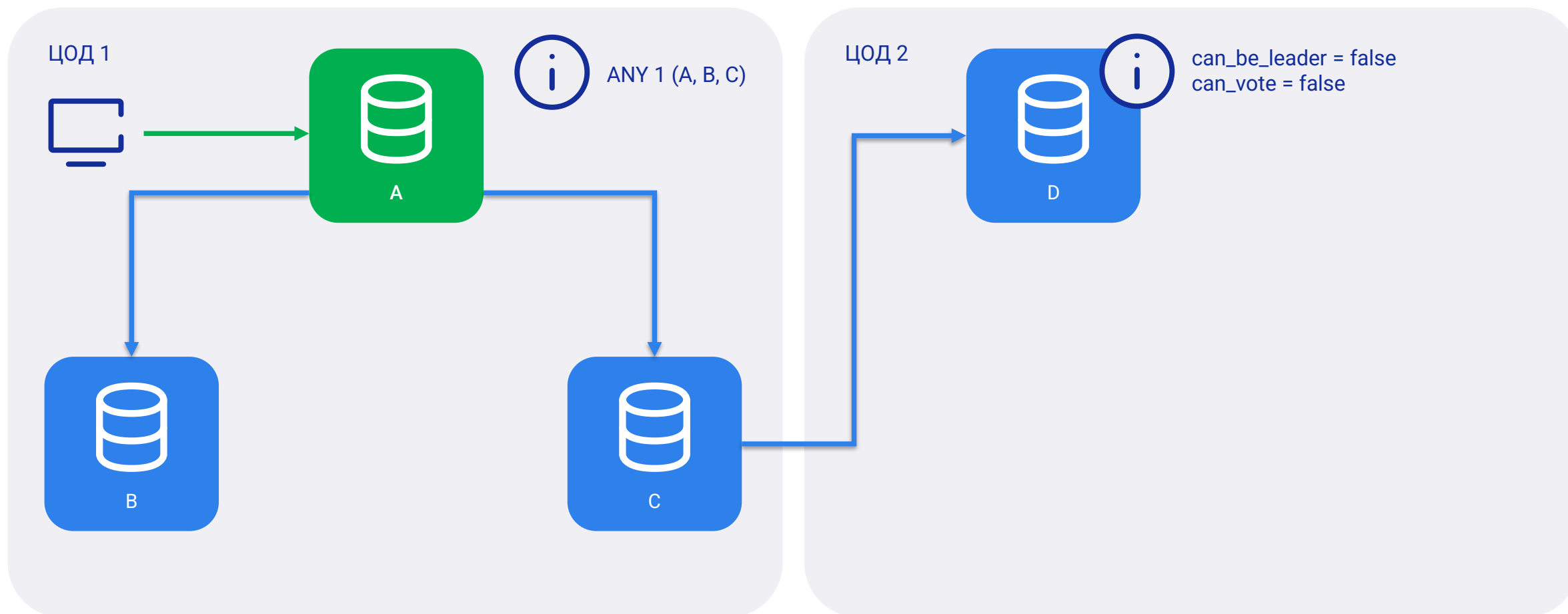
✓ Управление параметром `synchronous_standby_names`:

- `biha.add_to_ssn (id integer)` — добавляет узел.
- `biha.remove_from_ssn(id integer) returns boolean` — удаляет узел.
- `biha.get_ssn() returns varchar` — возвращает значение параметра `synchronous_standby_names`.



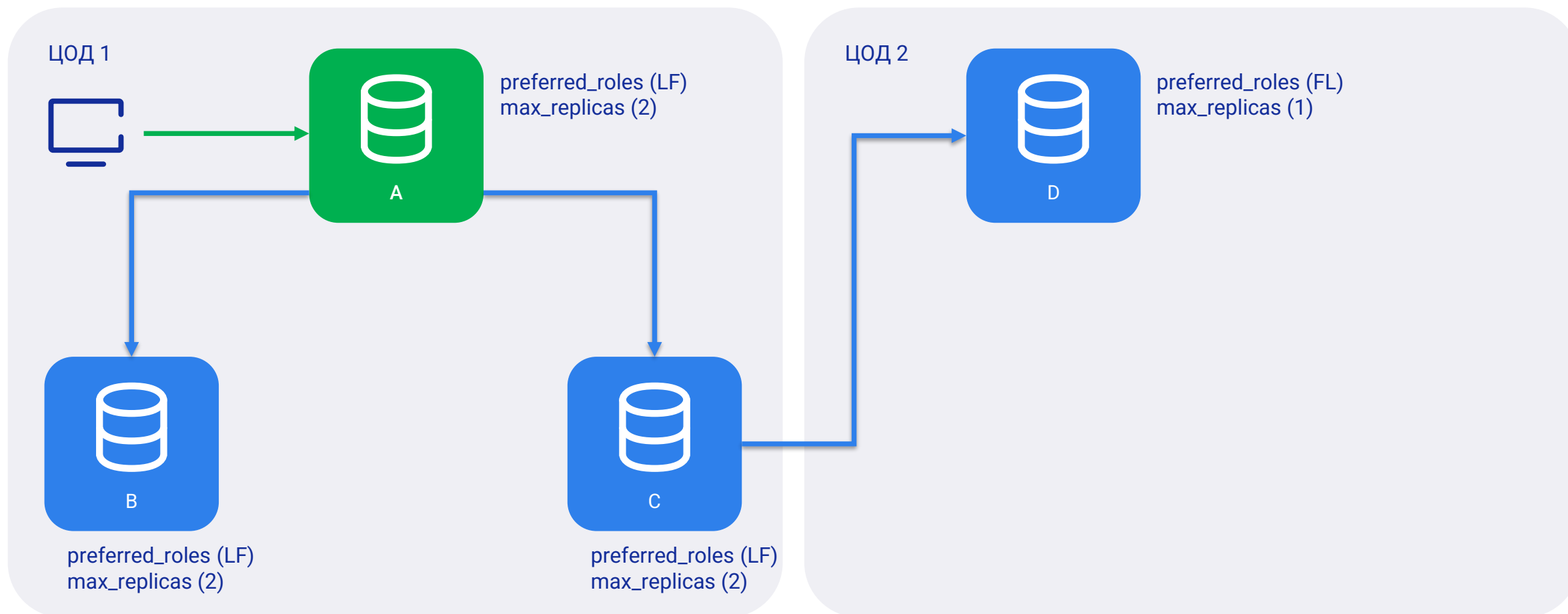
Георезерв : Каскадная репликация

- ✓ Узлы в удаленном ЦОД не голосуют и не выдвигают себя в Лидеры.
- ✓ Снижение нагрузки на Лидера.



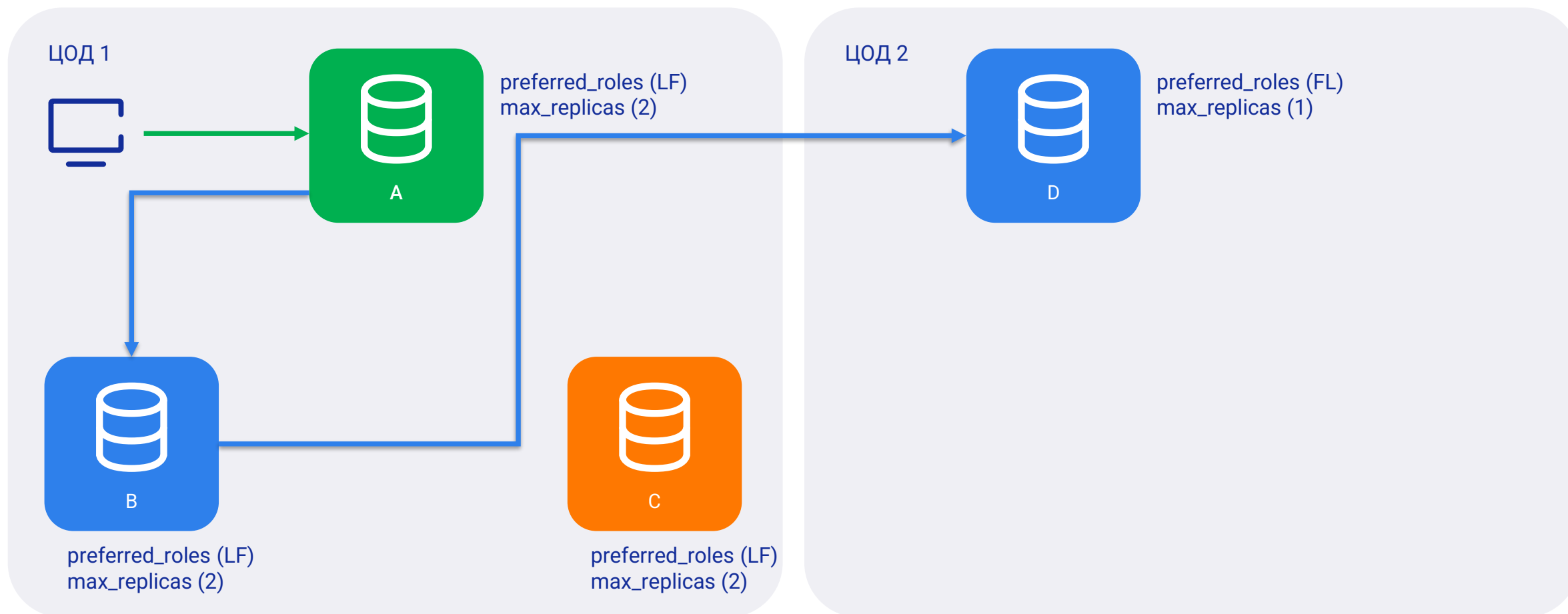
Георезерв: Каскадная репликация

- ✓ Каскадная репликация перестраивается автоматически в зависимости от параметров узла:
 - `preferred_roles`: приоритет роли источника (L, F, R). Устанавливается функцией `biha.set_pref_roles`.
 - `max_replicas`: количество репликаций для ВiНА-узлов. Устанавливается функцией `biha.set_max_replicas`.



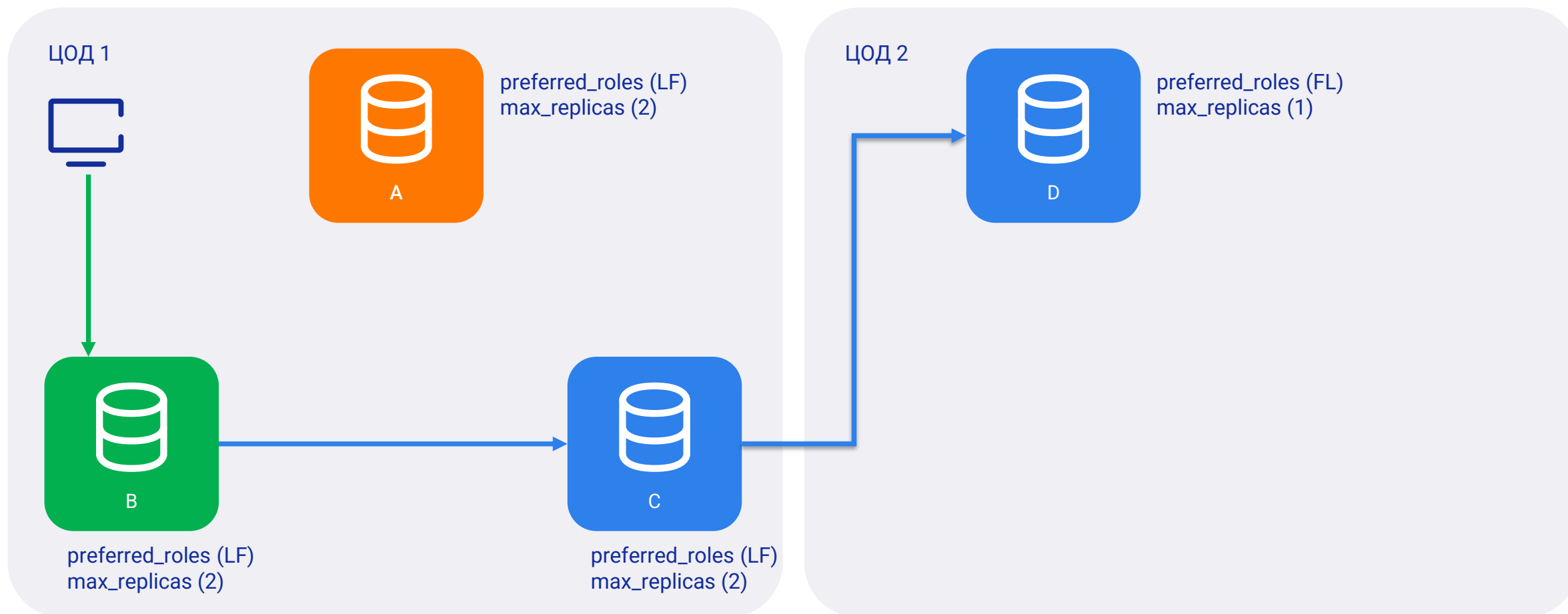
Георезерв : Каскадная репликация

- ✓ Каскадная репликация перестраивается автоматически в зависимости от параметров узла:
 - `preferred_roles`: приоритет роли источника (L, F, R). Устанавливается функцией [biha.set_pref_roles](#).
 - `max_replicas`: количество репликаций для BiHA-узлов. Устанавливается функцией [biha.set_max_replicas](#).



Георезерв : Каскадная репликация

- ✓ Каскадная репликация перестраивается автоматически в зависимости от параметров узла:
 - `preferred_roles`: приоритет роли источника (L, F, R). Устанавливается функцией `biha.set_pref_roles`.
 - `max_replicas`: количество репликаций для ВiНА-узлов. Устанавливается функцией `biha.set_max_replicas`.



Многоуровневый катастрофоустойчивый кластер (GDBiHA)

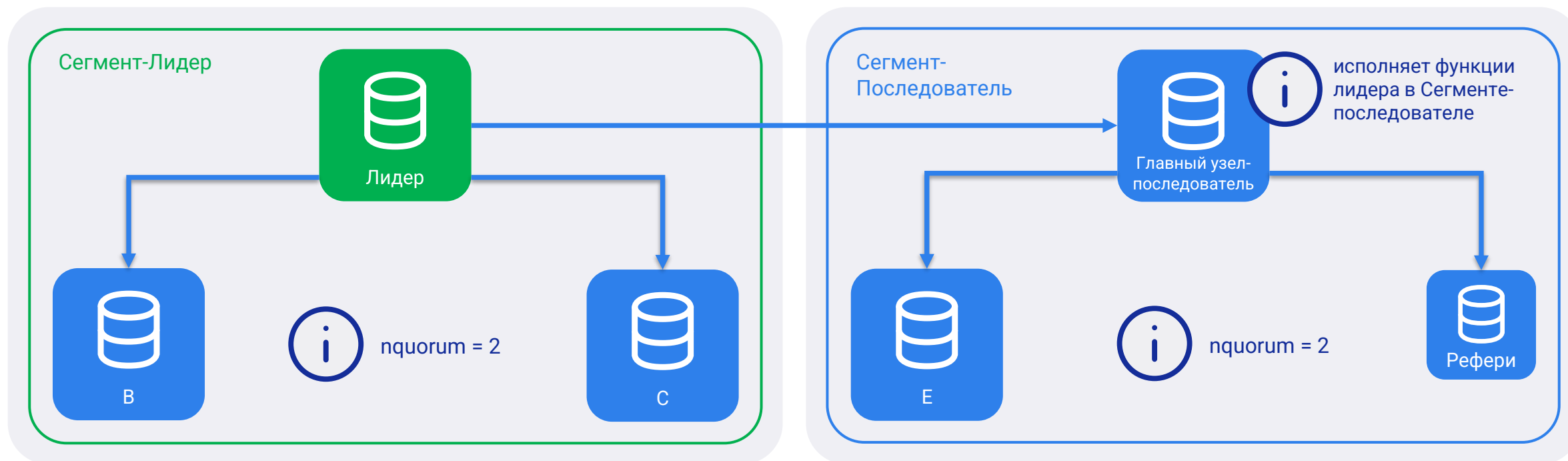
Узлы сгруппированы по сегментам – логическим узлам.

Новые термины: Сегмент-лидер, Сегмент-последователь, Главный узел последователь.

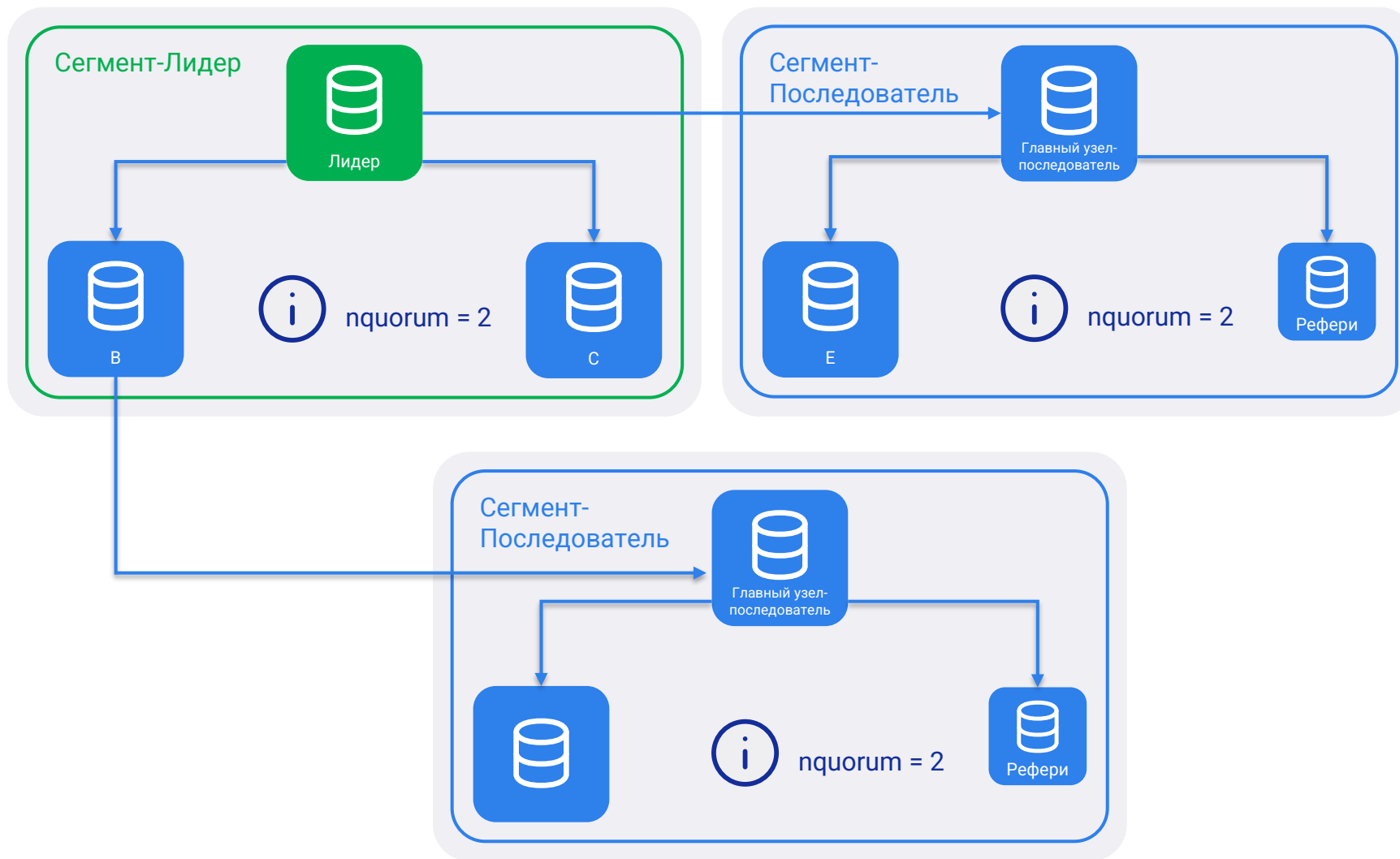
Каждый сегмент имеет собственную систему выборов.

Сегменты имеют свои параметры кворума и минимально количества нод.

Сегмент-последователь может стать автоматически Сегмент-лидером, только если сегментов более чем 2 (будет реализовано в будущей версии).



GDBiHA



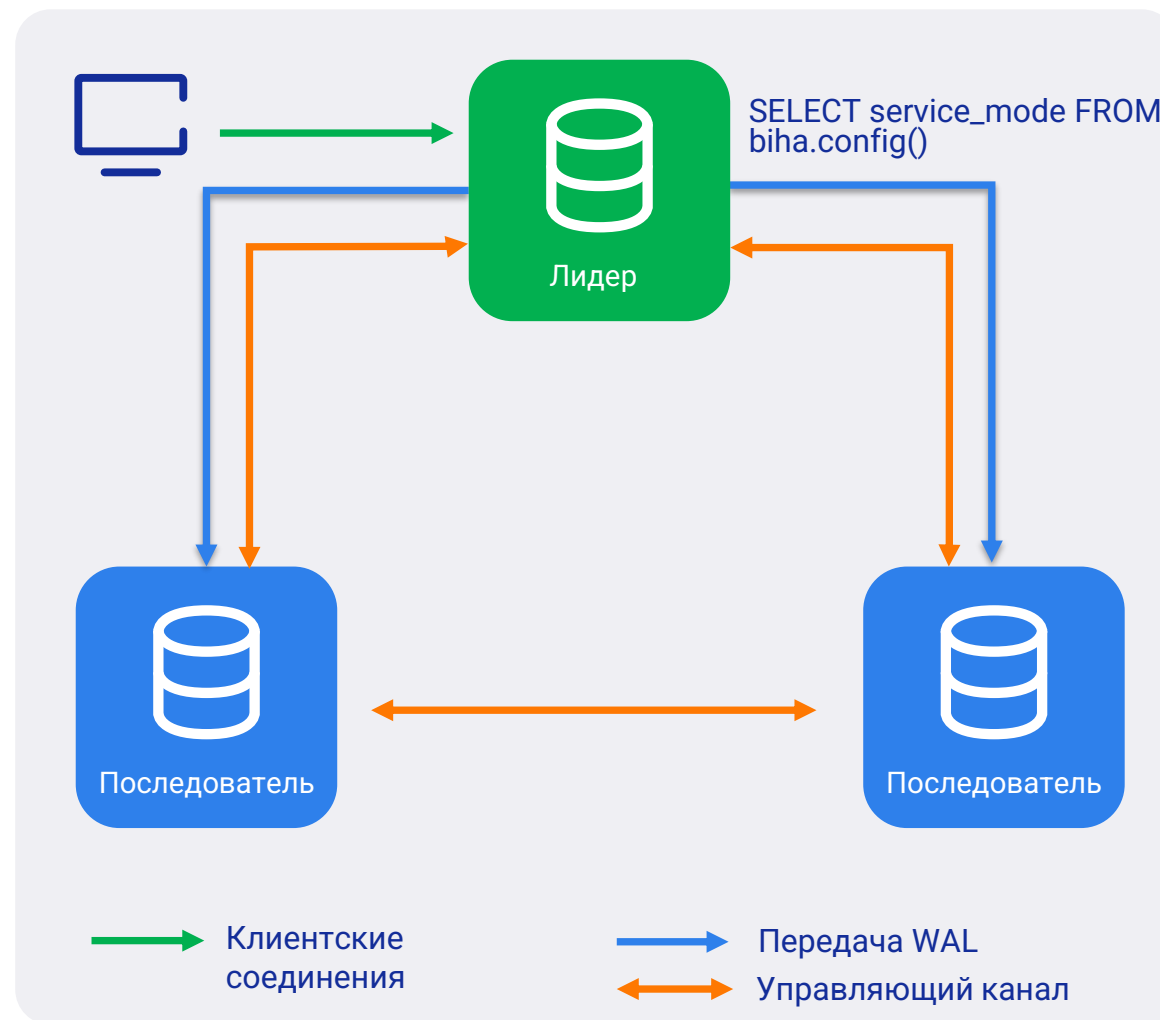
Postgres Pro BiHA: Сервисный режим

Отключает автоматическое переключение
Лидера.

Смена Лидера возможна в ручном режиме
(set_leader).

Применяется для:

- Обновления системного ПО.
- Обновления сервера.
- Гарантированного применения изменений параметров Postgres на всех узлах кластера.



Postgres Pro BiHA: Сервисный режим

Команда выполняется только на Лидере:
`biha.service_mode (enable boolean, force boolean)`
 returns boolean

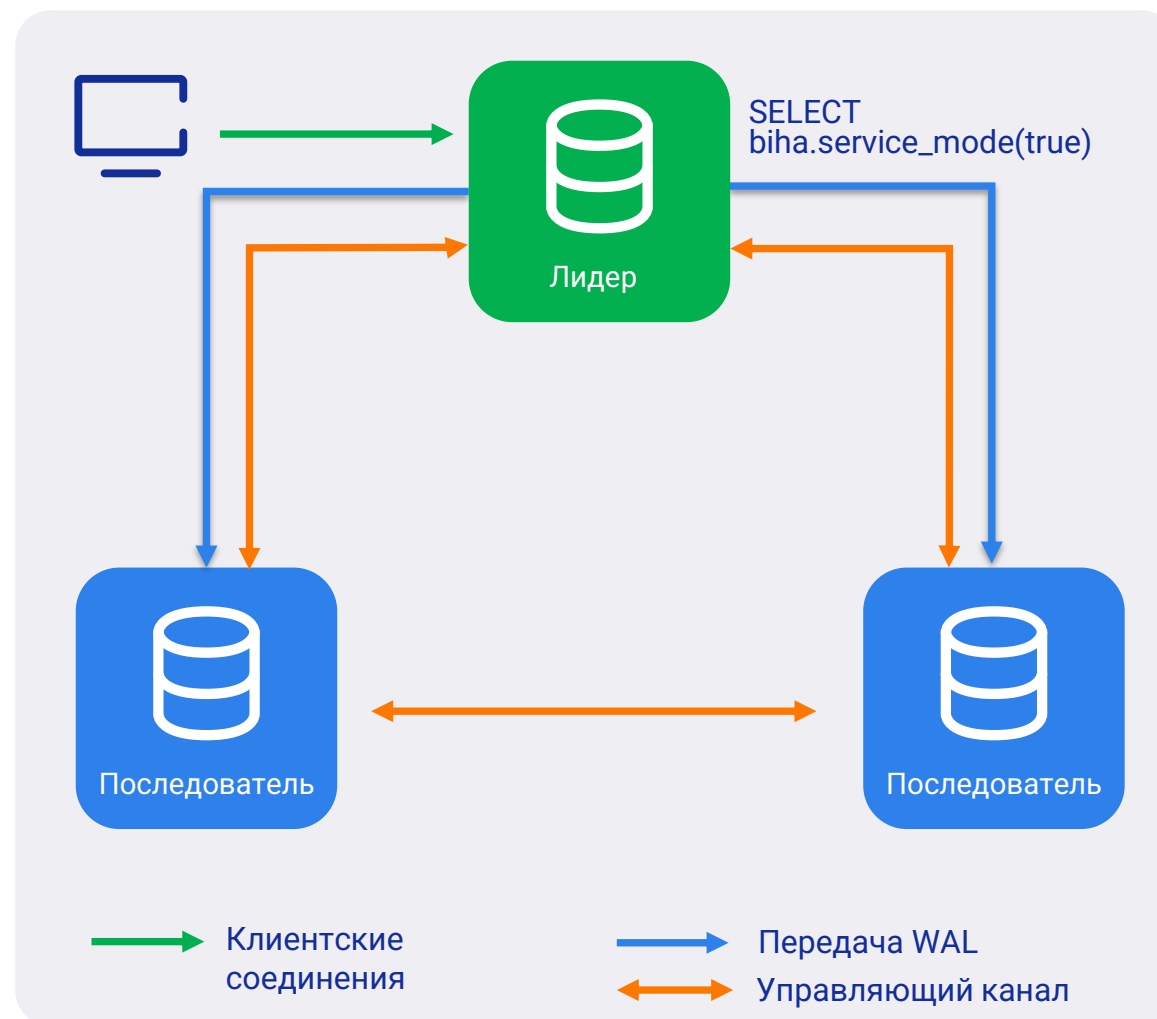
Включение сервисного режима:
`SELECT biha.service_mode(true);`

Выход из сервисного режима:
`SELECT biha.service_mode(false);`

Выполняется проверка измененных параметров, если не все параметры применены, команда не будет выполнена:

```
biha_db=# SELECT biha.service_mode(false);
ERROR:  [BiHA] Config changes
HINT:  Some parameters have been changed but not applied:
Node 2: max_connections,biha.autorewind
To force exit service mode, use: "select * from biha.service_mode(false, true)"
```

принудительное игнорирование ошибки:
`SELECT biha.service_mode(false, true)`



ВiНА: Мониторинг

В управляющий канал ВСР добавлена информация о статусе узла.

Статус узла известен даже если Postgres еще не доступен для подключений (не достиг точки консистентности).

Процент прогресса Startup вычисляется на основании конечного и начального lsn.

Описание полей функции `biha.monitoring()`

Поле	Описание
<code>id</code>	уникальный идентификатор узла
<code>time</code>	время получения данных мониторинга
<code>biha_state</code>	текущее состояние узла (см. <code>biha.status_v</code>)
<code>pg_state</code>	состояние Postgres Pro: Prestartup, Startup, Recovery, Recovery_Pause, Promoting, Promoted
<code>online</code>	показывает, в сети ли узел
<code>startup_progress</code>	показывает прогресс запуска в процентах, если значение <code>pg_state</code> – Startup

ViHA: мажорный апгрейд с минимальным простоем

Выполнение требований:

- Гарантия отказоустойчивость при единичном сбое
 - RTO -> 0
 - Возможность возврата (downgrade)
 - RPO = 0
-

Реализовано:

- Доработана утилита pg_createsubscriber
 - Реализована совместимость ViHA с утилитой pg_createsubscriber
 - Проработано 2 варианта процедуры мажорного апгрейда с минимальным простоем
-

Особенности выполнения обновления:

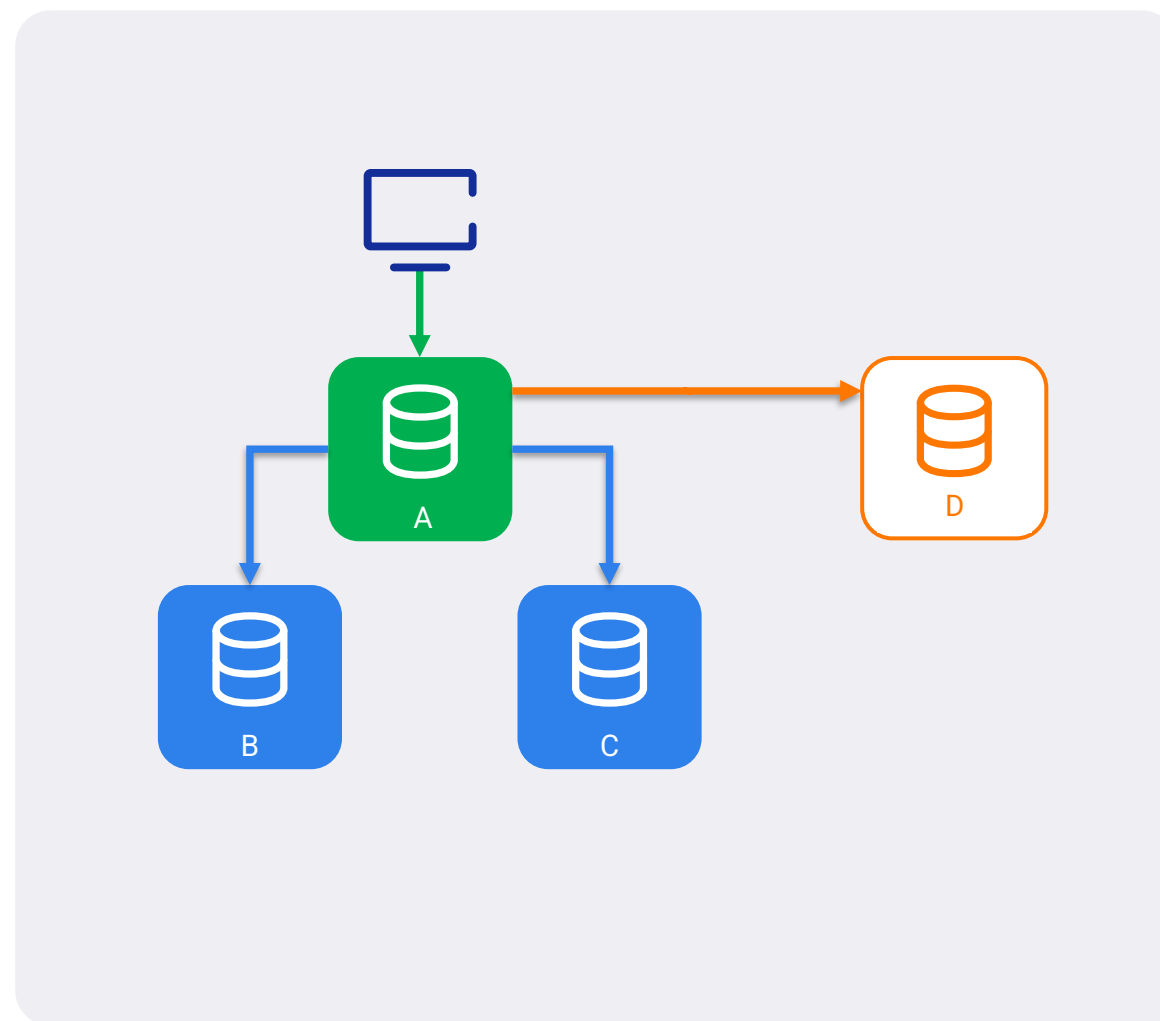
- Во время обновления не поддерживаются операции DDL, рекомендуется отказаться от автоматического срабатывания кластера ViHA, выбирать время с минимальной нагрузкой

ViHA: апгрейд с минимальным простоем

Этап 1: `bihactl upgrade start`

На дополнительном сервере:

- Создается реплика лидера старого кластера ViHA;
- С помощью `pg_createsubscriber` эта физическая реплика конвертируется в логическую реплику для всех баз данных экземпляра;
- С помощью `pg_upgrade` выполняется апгрейд этого экземпляра на новую версию;
- Этот экземпляр конвертируется в лидера нового кластера ViHA, который работает в сервисном режиме и не доступен для пользователей;
- Все пользовательские DML в старом кластере поступают на лидера нового кластера через логическую репликацию.

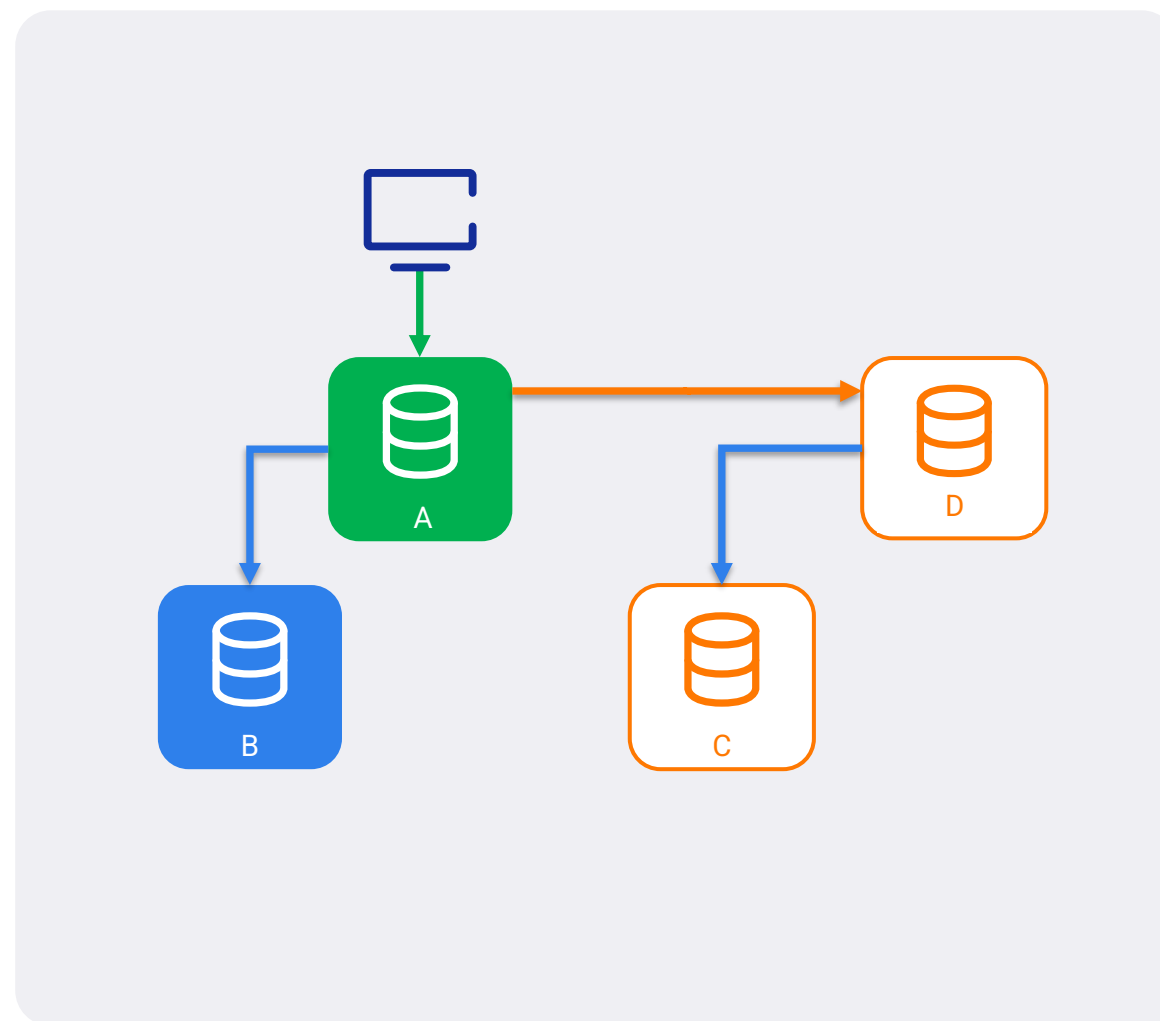


ViHA: апгрейд с минимальным простоем

Этап 2: `bihactl upgrade move`

На каждой реплике старого кластера:

- Удаляется этот узел из старого кластера ViHA;
- Создается реплика лидера нового кластера ViHA;
- Параметры этого узла ViHA устанавливаются в значения узла из старого кластера ViHA;
- Все пользовательские DML в старом кластере поступают на этот узел с лидера нового кластера через физическую репликацию.



ViHA: апгрейд с минимальным простоем

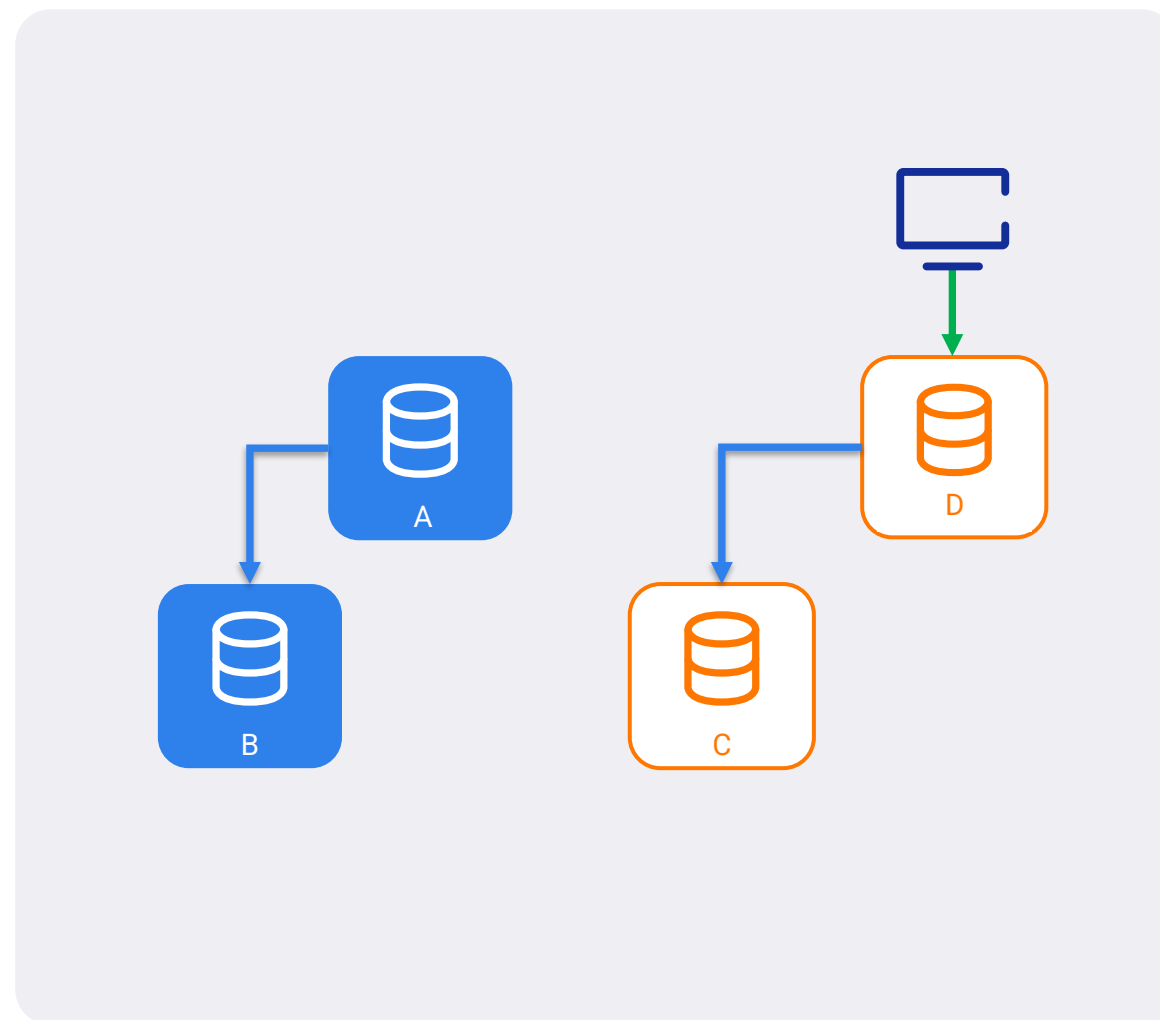
Этап 3: `bihactl upgrade finish` (DOWNTIME)

Перед выполнением команды `finish` отключите пользователей от старого кластера ViHA

В процессе выполнения команды `finish`:

- Старый лидер переводится в состояние `LEADER_RO`;
- На лидере нового кластера ViHA подписки на логическую репликацию удаляются;
- Параметры конфигурации нового кластера ViHA устанавливаются в значения старого кластера ViHA;
- Новый лидер выходит из сервисного режима и становится доступным для операций записи.

После выполнением команды `finish` подключите пользователей к новому кластеру ViHA.



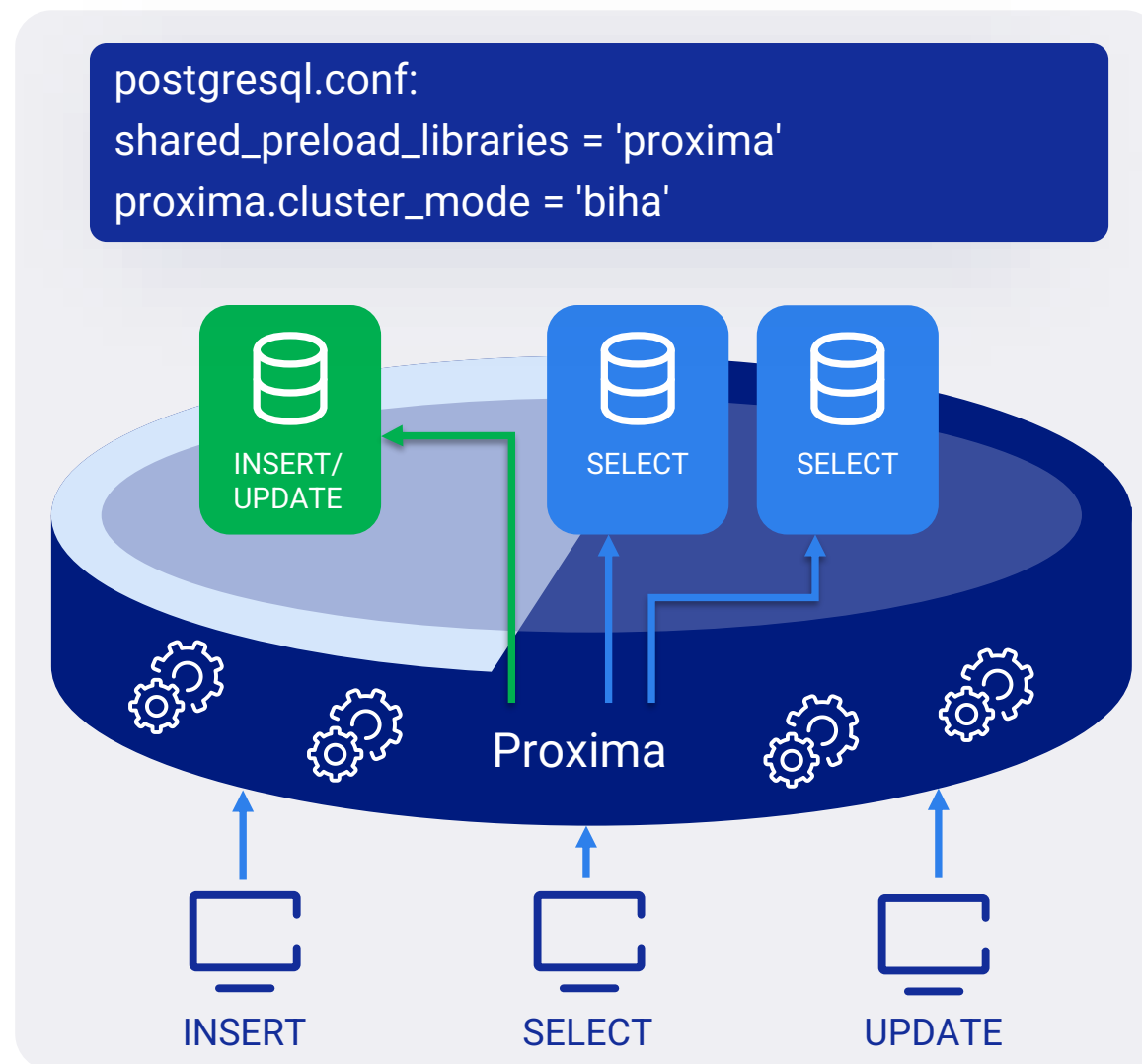
Технологии отказоустойчивости Postgres Pro

- Proxima

Proxima: Pooler, Proxy, Load Balancer

Встроен в Postgres Pro Enterprise с 17 версии.

Любой узел может быть точкой входа в кластер, Proxima всегда знает какой узел кластера является лидером.



Proxima: Pooler

Аутентификация пользователя происходит на Proxima.

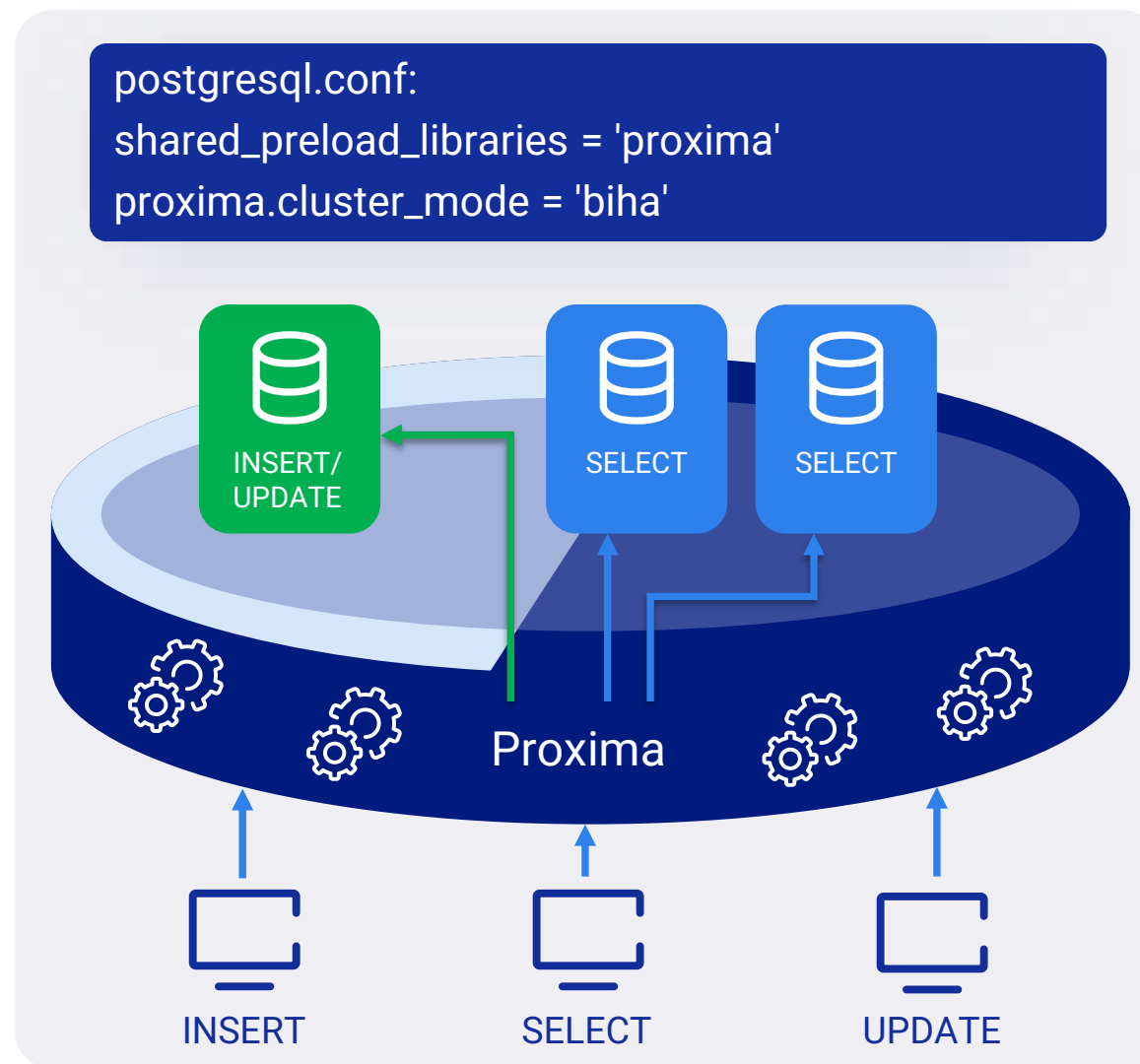
Каждый свободный Backend может быть использован для выполнения транзакции клиента, затем возвращается в pool в качестве свободного.

В рамках одной клиентской сессии может происходить переход в режим Dedicated и выход из него.

Dedicated режим: если клиент создает в процессе своей работы сессионный объект, то дальнейшую работу с ним он проводит через конкретный Backend процесс.

Меняют сессионное состояние:

- Временные таблицы
- SET [SESSION]
- Set role
- Блокировки
- Незавершённые транзакции и т.п.



Proxima: Proxy

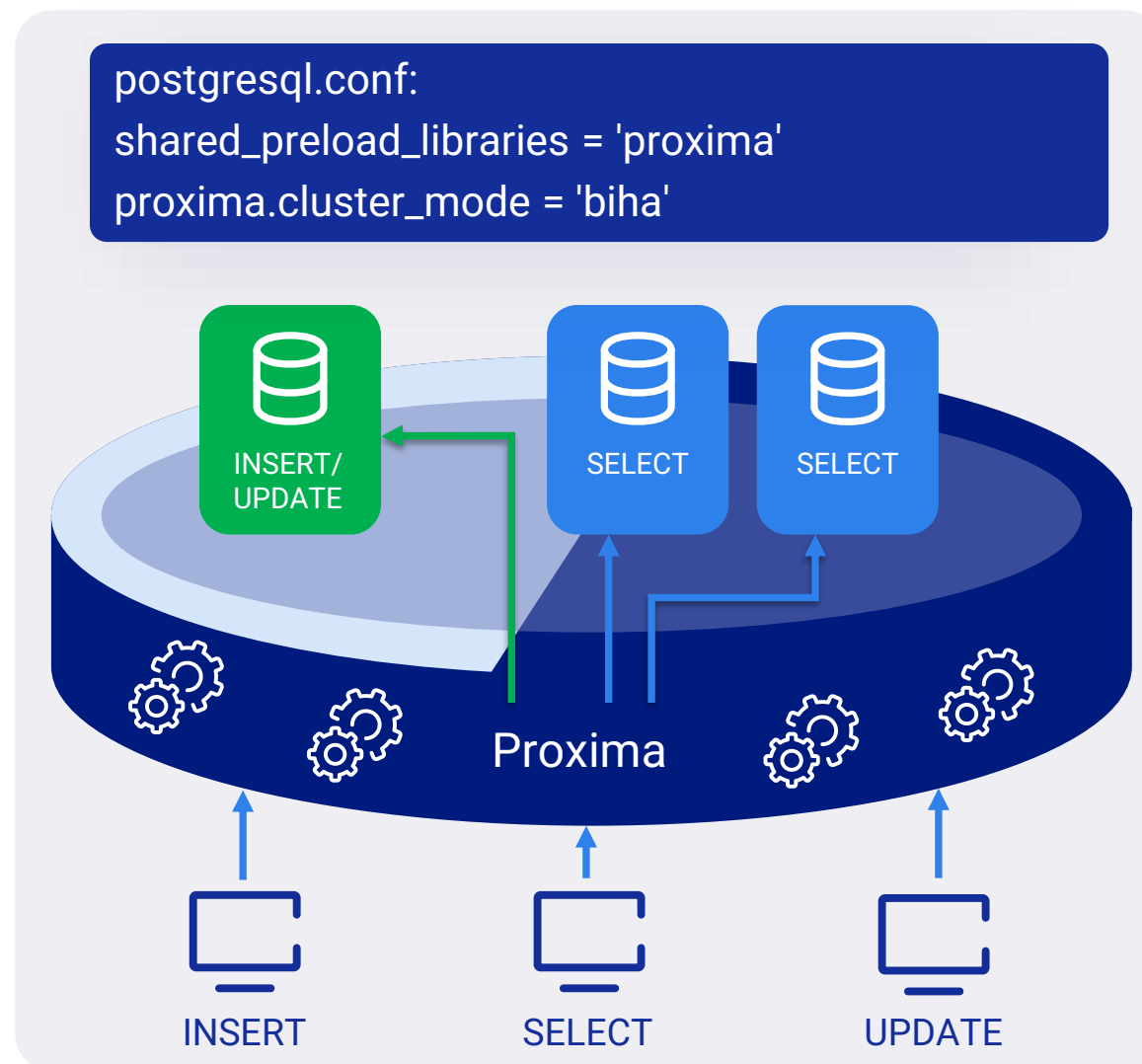
Proxima работает внутри экземпляров Postgres Pro на каждом узле кластера BiHA.

Любой узел может быть точкой входа в кластер.

Определение лидера BiHA происходит на лету.

В случае сбоя одного узла клиент может подключиться к Proxima на любом другом узле.

После обработки сбоя Proxima перенаправляет соединения на новый лидер BiHA.



Proxima: Load Balancer – балансировка читающей нагрузки

Proxima имеет 2 порта:

- proxima.port (integer) – порт для перенаправления на лидера, по умолчанию 4545;
- proxima.p2f_port (integer) – порт читающей нагрузки для перенаправления на реплики, по умолчанию 4547.

Пример строк подключения для libpq:

Для пишущей нагрузки:

```
"host=host1,host2,host3 port=4545 dbname=testdb user=postgres password=secret"
```

Для читающей нагрузки:

```
"host=host1,host2,host3 port=4547 dbname=testdb user=postgres password=secret"
```

- Proxima: Load Balancer — балансировка читающей нагрузки

Алгоритм указывается в параметре: `load_balancer_algorithm` (text).

Возможные значения распределения нагрузки между репликами:

- `round-robin` — по очереди.
- `weighted-round-robin` — по очереди, пропорционально их весам, заданным в `proxima.load_balancer_node_weight`.
- `least-connections` — на реплику с низким числом активных подключений.
- `random` — в случайном порядке.
- `adaptive-cpu` — пропорционально загрузке CPU.

В работе: балансировка по RAM и I/O.

Спасибо за внимание!

