Максим Милютин
m.milyutin@postgrespro.ru

Stolon vs. Patroni

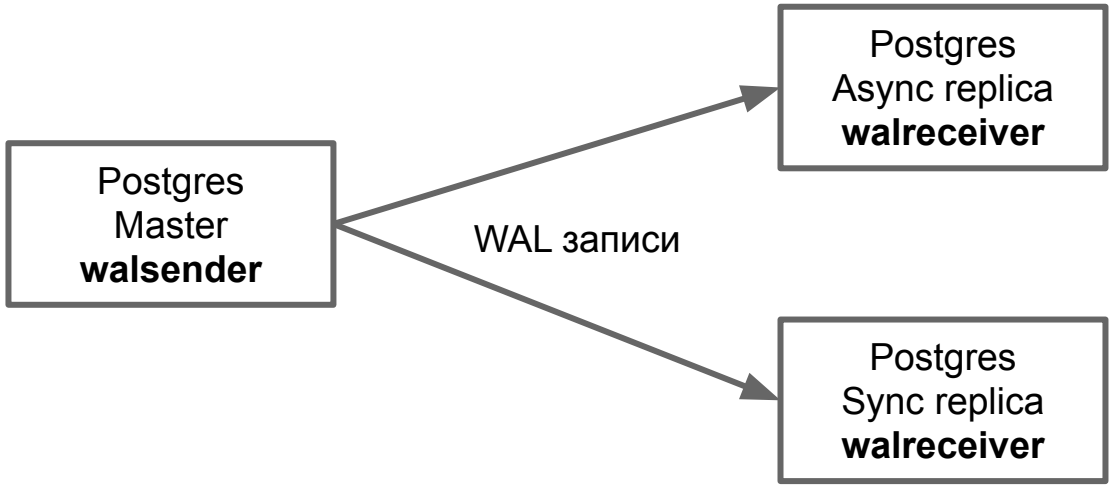
HA решения для PostgreSQL

Для успеха технологии реальные факты должны быть важнее пиара, ибо природу обмануть невозможно

Ричард Фейнман
(из доклада в комиссии Роджерса)

Матчасть

Физическая репликация



Кластера на базе физической репликации

- Архитектура shared-nothing (без общего диска)
- Реплицируются физические изменения, описанные WAL
- Поддерживается режим чтения из реплики **hot standby**
- Готовые инструменты управления:
  - создание реплики - **pg_basebackup** от мастера, восстановление из стороннего бэкапа
  - **promote** нового мастера
  - откат не реплицированных изменений на бывшем мастере - **pg_rewind**
  - мониторинг реплик - **pg_stat_replication**

Сложности кластера на базе встроенной логической репликации

- Нет перепозиционирования реплик на новый мастер
  - прогресс репликации завязан на позицию в WAL мастера - LSN (log sequence number)
  - нет аналога GTID как в MySQL
- Нет аналога **pg_rewind** (легковесная ресинхронизация бывшего мастера)
- Ограниченные возможности
  - не реплицируются DDL, последовательности (sequence)
- TODO в кластерных решениях Stolon/Patroni
  - для обновления мажорной версии PostgreSQL без простоя кластера (rolling upgrade)
  - https://github.com/sorintlab/stolon/issues/519
  - https://github.com/zalando/patroni/issues/538

Postgres PROFESSIONAL

Типы репликации

- Асинхронная
- Синхронная (мастер ожидает подтверждения коммита на репликах)
  - **remote_write** - ожидание записи коммита в кэш ОС
  - **remote_flush** (synchronous_commit = on) - ожидание сброса коммита на диск (режим по умолчанию)
  - **remote_apply** - ожидание “проигрывания” коммита на реплике (консистентное чтение)
- “Степень синхронности” регулируется **на уровне транзакций**

https://www.postgresql.org/docs/current/warm-standby.html#SYNCHRONOUS-REPLICATION-PERFORMANCE

Полусинхронная репликация

- При синхронной репликации потеря реплики **блокирует** кластер **на запись**
- Подмножество (вариативное со временем) реплик являются синхронными
- Синхронные реплики задаются:
  - приоритетами узлов (priority based synchronous replication)

    `synchronous_standby_names = 'FIRST k (s1 s2 … sN)'`
  - кворумом на коммит (quorum based synchronous replication)

    `synchronous_standby_names = 'ANY k (s1 s2 … sN)'`

https://www.postgresql.org/docs/current/warm-standby.html#SYNCHRONOUS-REPLICATION-HA

Postgres PROFESSIONAL

Потеря коммитов при синхронной репликации

- Сброс на диск коммита на мастере происходит до подтверждения на синхронных репликах
- Рестарт мастера “восстанавливает” изменения транзакций, ожидавших подтверждения коммита на репликах
- Failover на синхронную реплику теряет не реплицированные зависшие транзакции

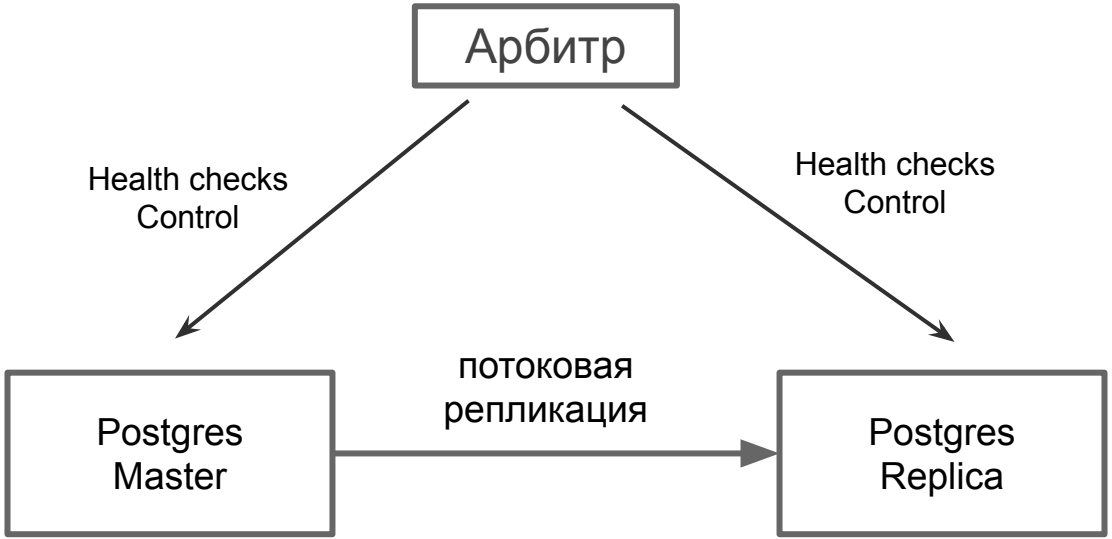
https://github.com/sorintlab/stolon/blob/master/doc/syncrepl.md#handling-postgresql-sync-repl-limits-under-such-circumstances

Итого

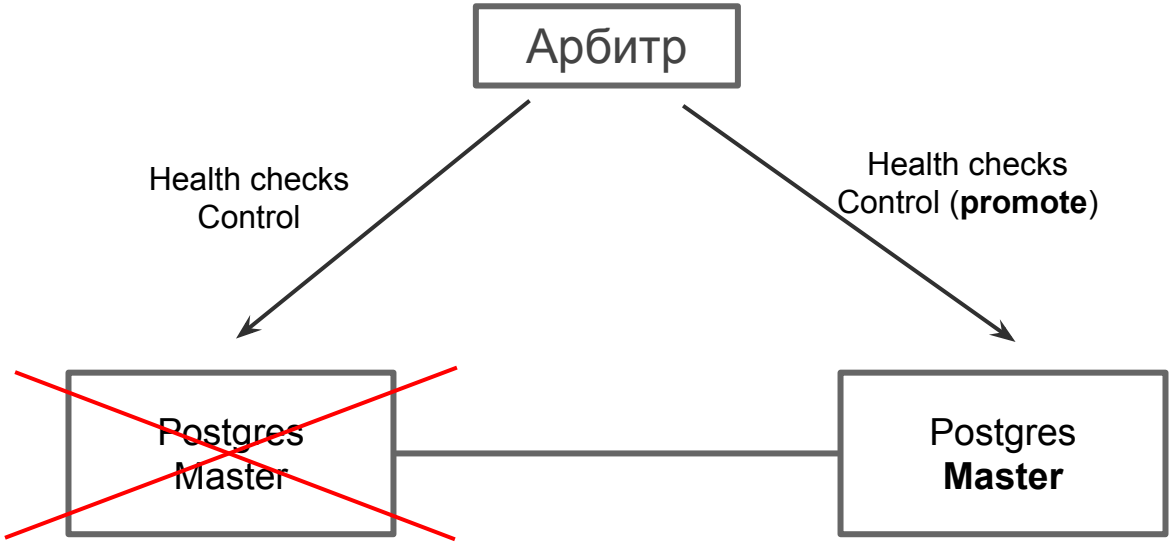
- Есть инструменты управления PostgreSQL узлами:
  - создание и инициализация узлов БД (**initdb**, **pg_basebackup**, восстановление из стороннего бэкапа)
  - **promote** нового мастера, **pg_rewind** (demote) старого мастера
- Нет “третьего” арбитра, который:
  - определял падение действующего мастера
  - выдерживал желаемую конфигурацию кластера:
    - promote/demote узлов БД
    - перевод реплики в синхронный/асинхронный режим

Архитектура кластеров класса Stolon/Patroni

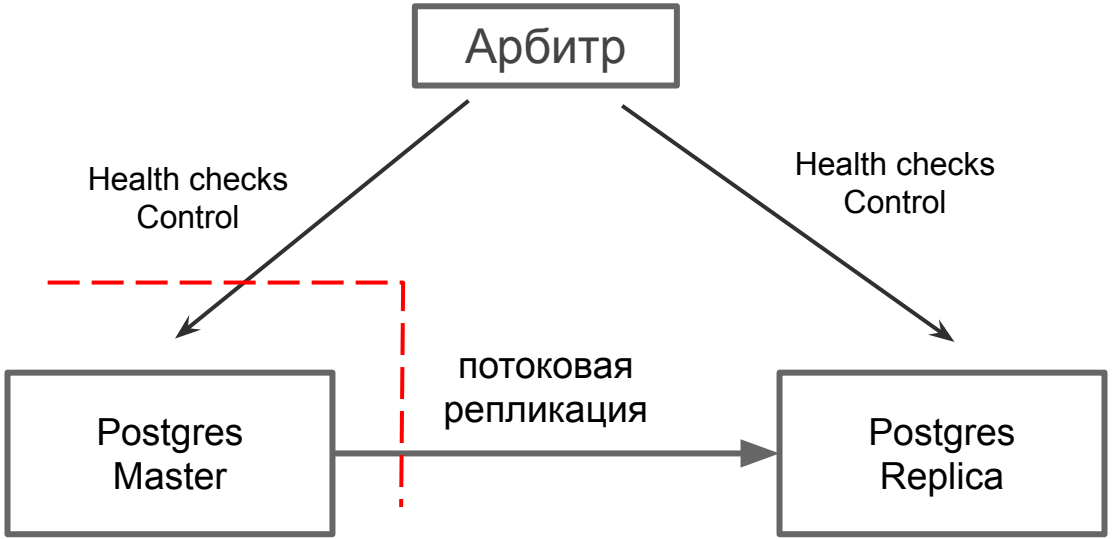
Примитивный автофейловер



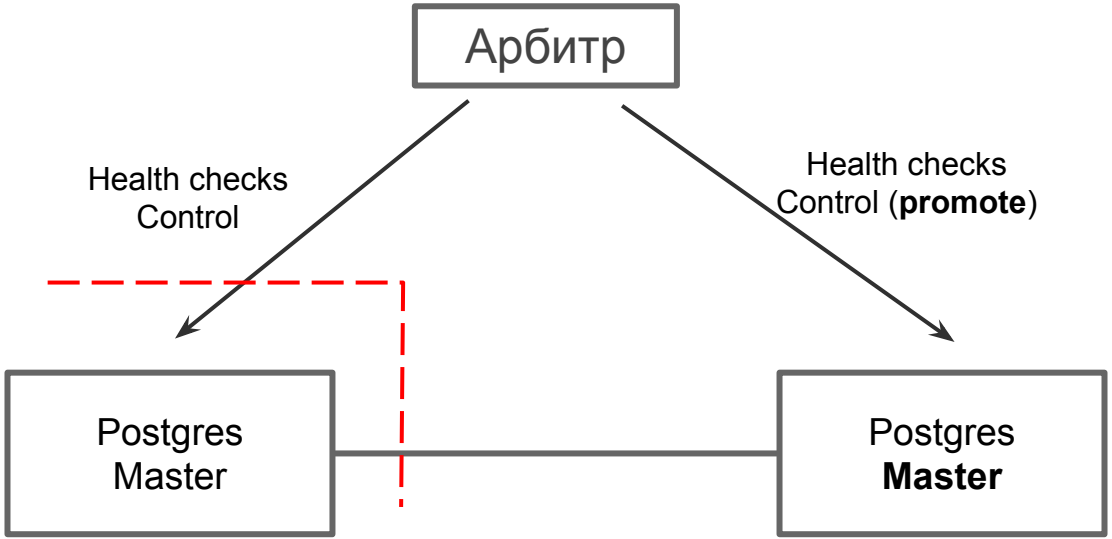
Примитивный автофейловер



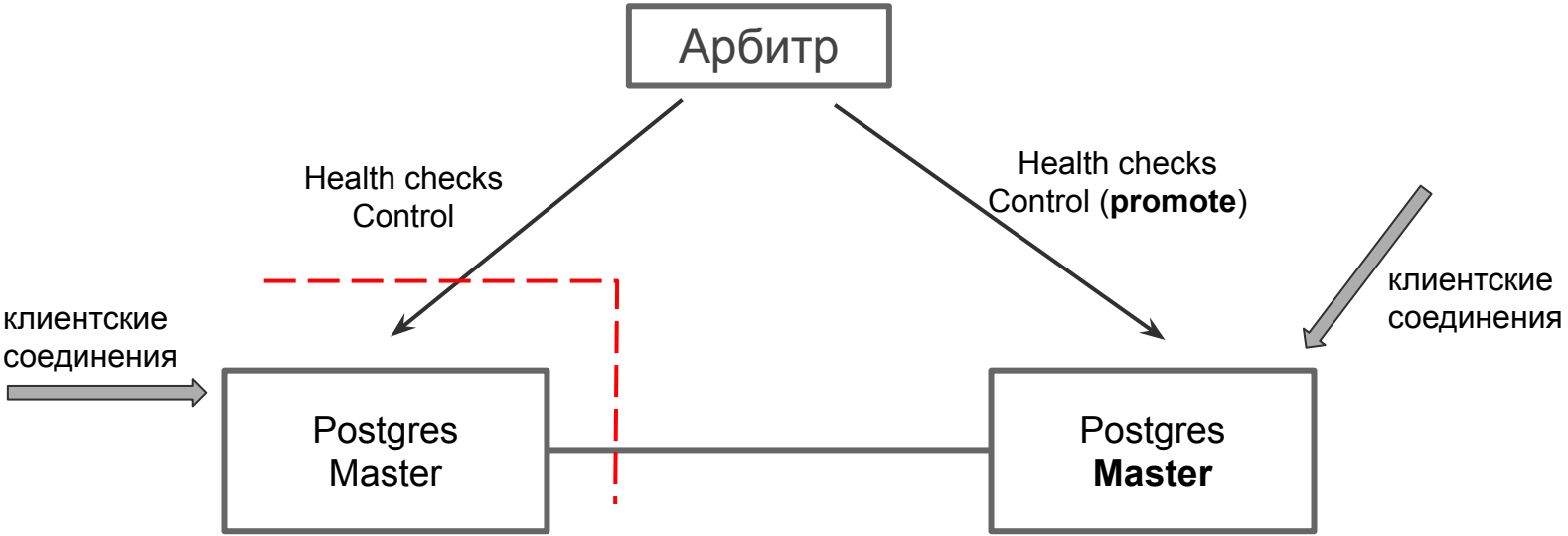
Примитивный автофейловер. Изоляция мастера



Примитивный автофейловер. Изоляция мастера

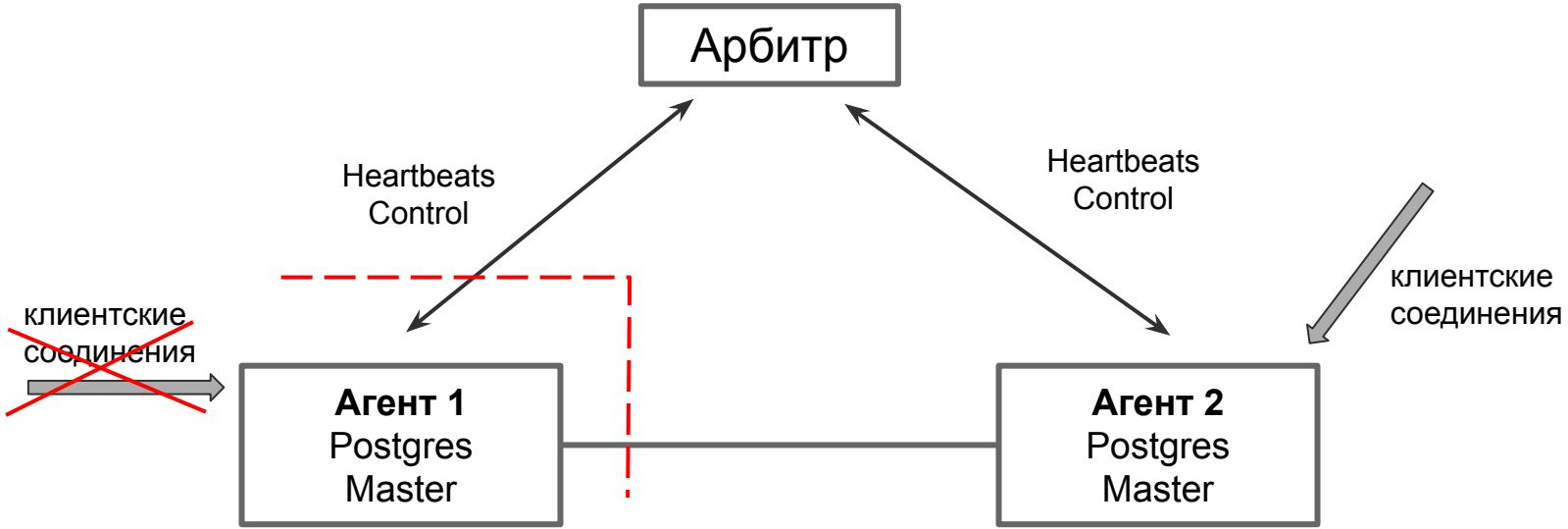


Примитивный автофейловер. Split brain



Клиенты на старом мастере && **асинхронная репликация**

Примитивный автофейловер. Фенсинг изолированного мастера



Агенты:

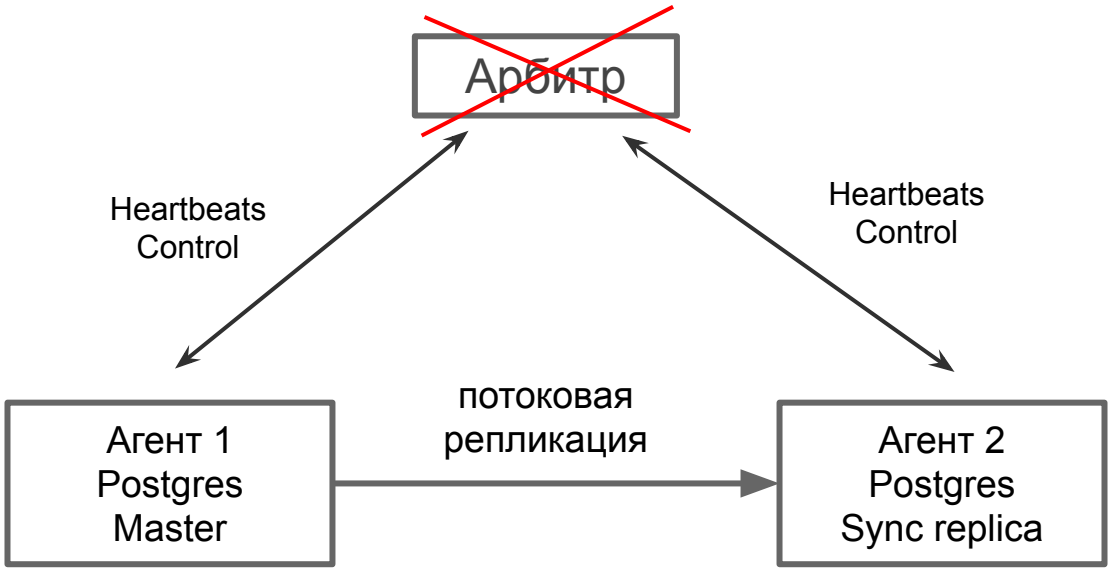
- управляют узлами PostgreSQL
- отправляют статус арбитру
- “**фенсят**” изолированный узел

Изоляция неактуального мастера (fencing)

“Ограждение” от изменений мастера, изолированного от основного кластера (избегание **split brain**):

- выключение узла (экземпляра PostgreSQL) - **STONITH**
- разрыв всех текущих соединений и отказ от приёма новых
- перевод БД в режим read-only

Примитивный автофейловер. Доступность арбитра



Арбитр - единая точка отказа

Схема HA кластера, осуществляющего автофейловер

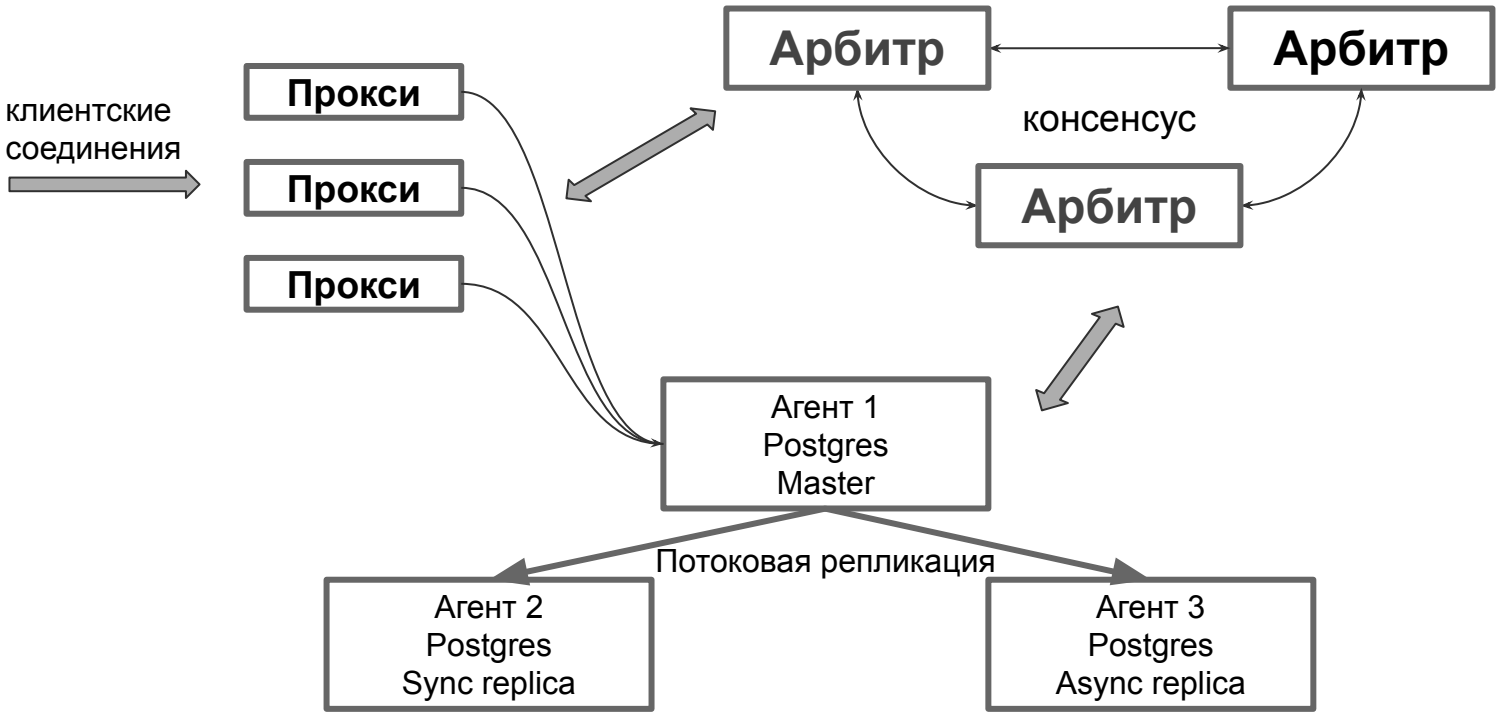
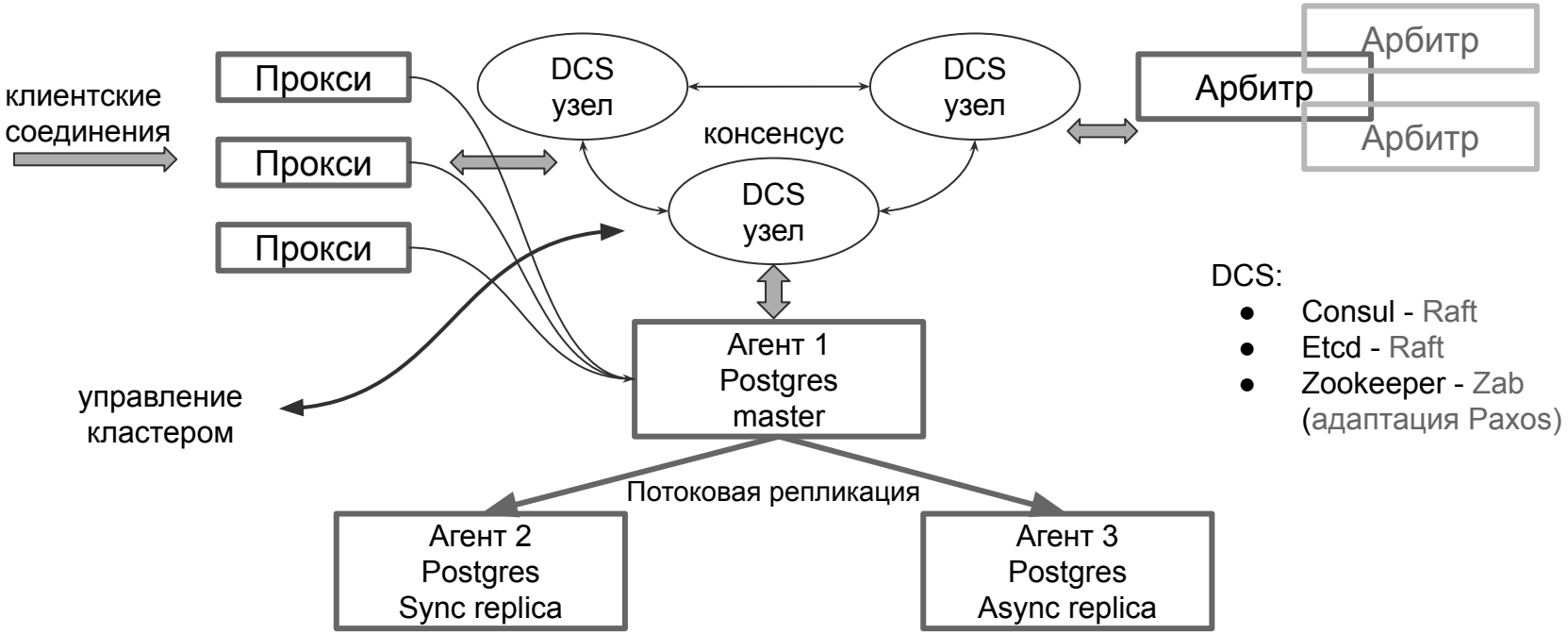


Схема HA кластера класса Patroni/Stolon



DCS - Distributed Configuration System (Распределённая система конфигурации)

Архитектура Stolon

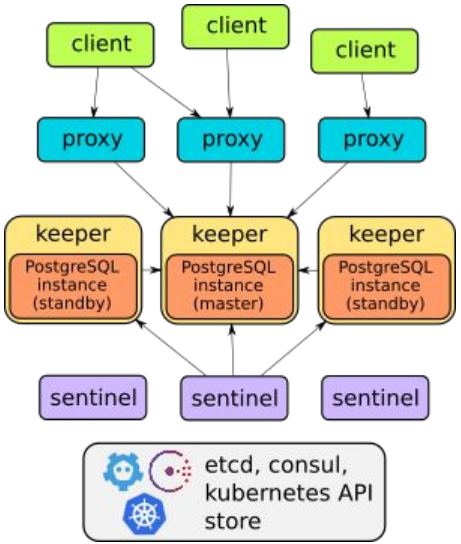
- **keeper (postgres-агент)**

  управляет экземпляром PostgreSQL по спецификации, “спущенной” от sentinel
- **sentinel (арбитр кластера)**

  обнаруживает и мониторит все остальные компоненты, вычисляет оптимальную конфигурацию “мастер-реплики”
- **proxy**

  точка доступа к мастер узлу от клиентов
- **KV-store (consul, etcd, etc)**

  шина данных/управления/событий



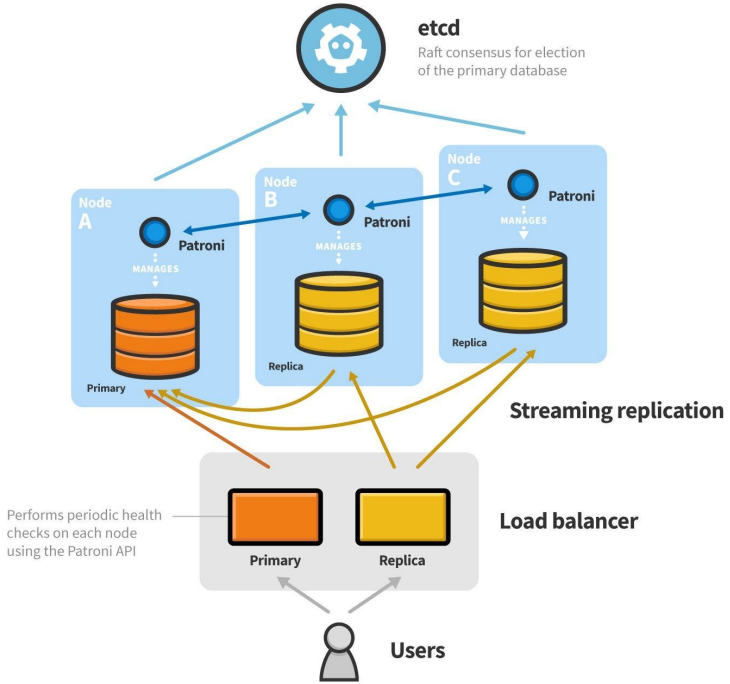
https://github.com/sorintlab/stolon#architecture

Postgres PROFESSIONAL

Архитектура Patroni

- **Patroni bot (postgres-агент)**
  - управляет экземпляром PostgreSQL
  - связаны друг с другом по REST API
- **Арбитр кластера**
  - DCS
    - имплементация распределённого CAS
    - поддержка TTL для ключа мастер узла
  - Patroni агенты. По REST API:
    - участвуют в выборе нового мастера
    - подтверждают падение мастер узла
- **Прокси (балансировщик)**

  сторонний настраиваемый компонент



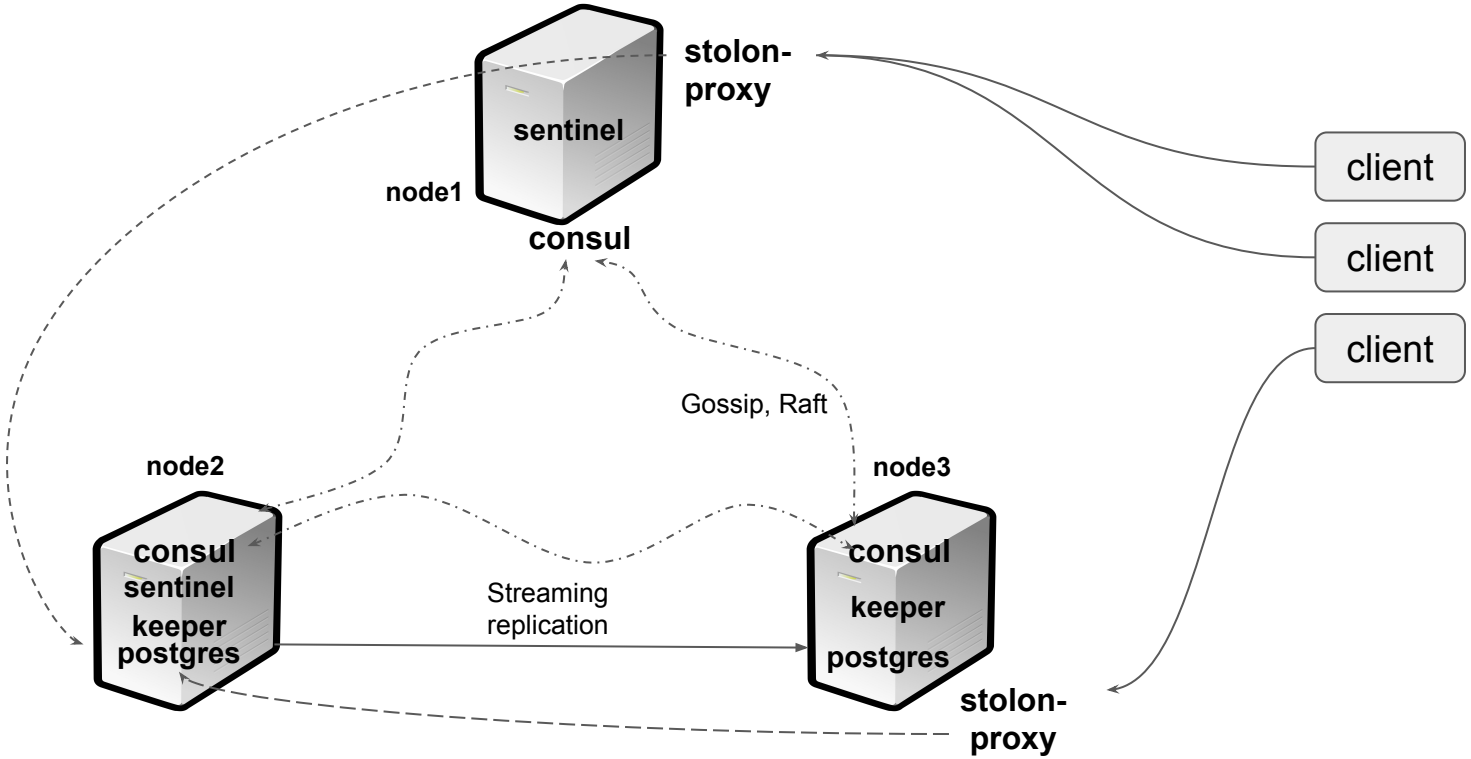
https://blog.timescale.com/high-availability-timescaledb-postgresql-patroni-a4572264a831/#getting-started-with-patroni

HA кластер на базе PostgreSQL класса Patroni/Stolon

- Автоматическая реакция на инциденты в кластере (переключение роли мастера при недоступности текущего)
- Автоматически настраиваемые узлы баз данных (агенты берут на себя всю рутину по настройке и управлению PostgreSQL)
- Тесная интеграция с облачными решениями (kubernetes)
- Продукт класса DBaaS (database as a service)

Инсталляция кластера под управлением Stolon/Patroni

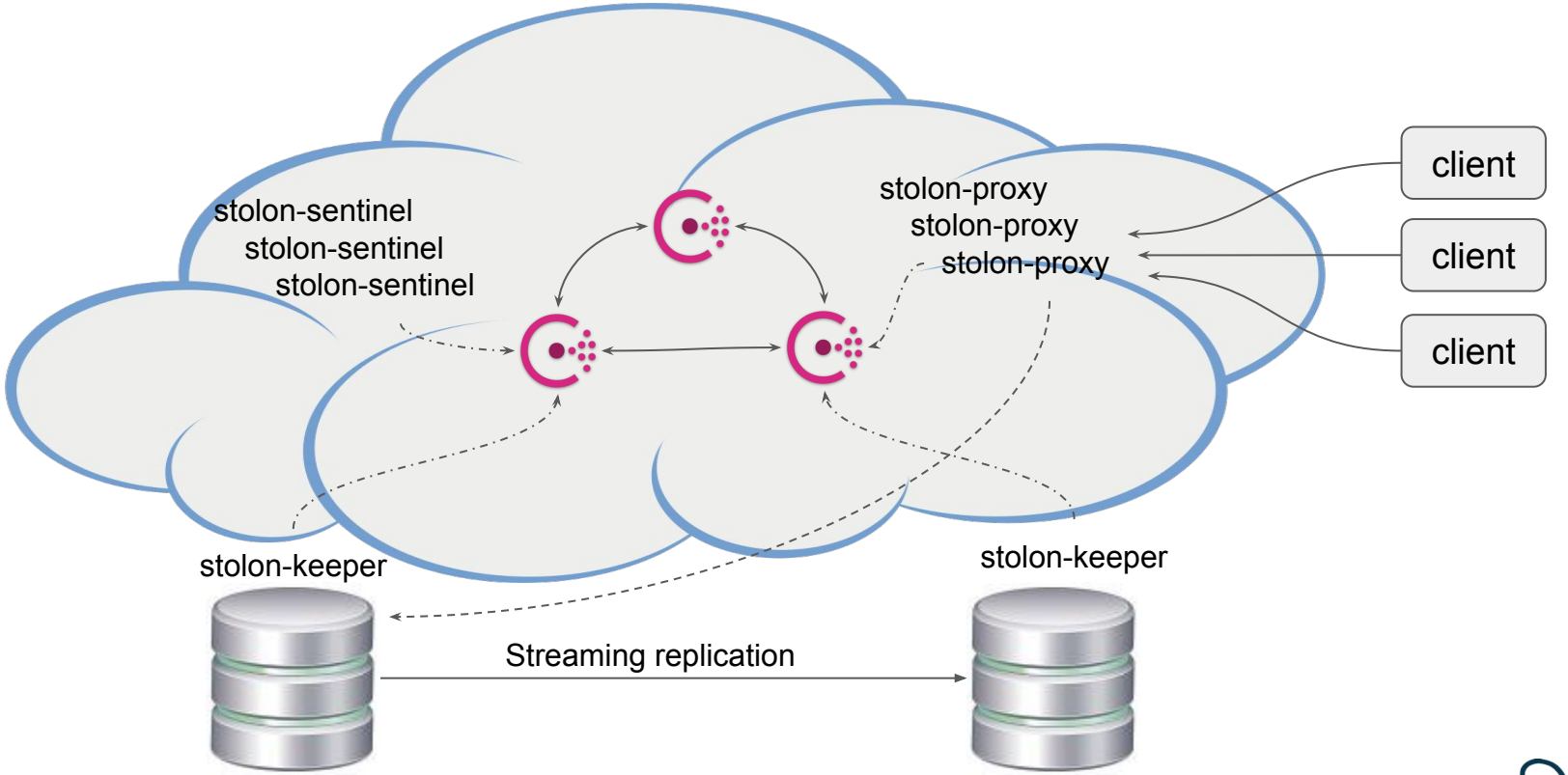
Классическая инсталляция Stolon на 3-х узлах



https://github.com/sorintlab/stolon/issues/313
https://github.com/maksm90/stolon_vmcluster

Postgres PROFESSIONAL

Облачная инсталляция Stolon



- consul (DCS) узел

Распределённая система конфигурации (DCS) на примере consul

- **минимум 3 серверных узла**, остальные клиентские
- серверный узел хранит состояние, является мини БД и **требует стабильное (по времени) по записи/чтению хранилище**
- **RAFT** как консенсус-протокол между сервер-узлами
- протокол **Gossip** для membership list / failure detection / leader election
- внешние запросы принимаются любым узлом по **HTTP**/DNS.
- уровни консистентности чтения:
  - **stale** - перенаправление на сервер-узел
  - **default** - перенаправление на лидер (+1 round-trip)
  - ***consistent*** - лидер проверяет удержание кворума (+2 round-trip) ✓

https://www.consul.io/docs/internals/architecture.html

Прокси в Patroni (master discovery)

- С использованием REST API патрони агентов
  - /master - { 200: master, 503: replica }
  - /replica - { 503: master, 200: replica }
  - пример с HAProxy [1]
- Через callback вызовы
  - on_start, on_stop, on_reload, on_restart, on_role_change
- Подписка к изменениям (периодическое обновление) DCS
  - confd, consul-template, custom scripts
  - пример с confd/pgbouncer [2]
- Множество хостов в строке подключения
  - `jdbc:postgresql://node1,node2,node3/postgres?targetServerType=master`
  - `postgresql://host1:port2,host2:port2/?target_session_attrs=read-write`

1. https://www.consul.io/docs/internals/architecture.html
2. https://github.com/zalando/patroni/tree/master/extras/confd

Postgres PROFESSIONAL

Прокси в Patroni (master discovery)

- Через DNS сервер consul (service discovery)
  - требует проброс DNS в локальной окружение клиента [1] и настройки кэширования [2]
  - критика Github в кластере под управлением orchestrator [3]
  - пример использования в проде [4]
- Единый IP-адрес для мастер узла
  - HAProxy + keepalived
  - vip-manager [5]

1. https://www.consul.io/docs/guides/forwarding.html
2. https://learn.hashicorp.com/consul/day-2-operations/advanced-operations/dns-caching
3. https://github.blog/2018-06-20-mysql-high-availability-at-github/#moving-away-from-vip-and-dns-based-discovery
4. https://pgconf.ru/2019/242817 https://pgconf.ru/2019/242821
5. https://github.com/cybertec-postgresql/vip-manager

Workflow работы с кластером Stolon

- Все компоненты (keeper, sentinel, proxy) могут запускаться независимо друг от друга
- Инициализация кластера и баз данных:

  stolonctl init …

- Изменение параметров кластера (в т.ч. **postgresql.conf / pg_hba.conf**):

  stolonctl update …

- Перезагрузка PostgreSQL:
  - вручную (рестарт агентов stolon-keeper)
  - автоматически (параметр **automaticPgRestart**)
- Добавление/удаление узлов кластера происходит автоматически при запуске/останове **stolon-keeper**

Workflow работы с кластером Patroni

- Инициализация кластера и баз данных при запуске patroni агентов
  - конфигурация кластера (секция *bootstrap*) передаётся в конфиге агента (можно в единственном экземпляре)
- Изменение параметров кластера (в т.ч. **postgresql.conf / pg_hba.conf**):

  patronictl edit-config …
- Перезагрузка PostgreSQL:
  - **patronictl restart ...**
  - **patronictl restart --role …** соблюдая очерёдность мастер/реплики [1]
  - **patronictl restart --scheduled …** по расписанию
- Добавление/удаление узлов кластера происходит автоматически при запуске/останове **patroni**

1. https://www.postgresql.org/docs/current/hot-standby.html#HOT-STANDBY-ADMIN

Возможности Stolon/Patroni

Инициализация кластера. Patroni

- **initdb** - “с нуля”
- **PITR** - из бэкапа
  - сторонняя бэкап-система кастомизируется
- **Standby cluster**
  - “бесшовная” миграция с существующей инсталляции
- На базе существующей инсталляции [1]
  - не поддерживается stolon (свой формат PGDATA)
- Восстановление реплик **кастомизируется**

1. https://patroni.readthedocs.io/en/latest/existing_data.html

Восстановление кластера из бэкапа

Возможность восстановиться из бэкапа на любую точку по:

- времени
- id транзакции (xid)
- lsn транзакционной записи в журнале
- именной записи в журнале

```
bootstrap:
    method: probackup
    probackup:
        command: “pg_probackup restore -B /path/to/backup --instance <scope> -D <datadir> --time=’2019-04-27 00:00:00 MSK’ \
                                        --recovery-target-action=promote --timeline=latest”
        keep_existing_recovery_conf: true
        # recovery_conf:
            # recovery_target_action:
            # recovery_target_timeline:
            # restore_command:
            # recovery_target_time:
```

Postgres PROFESSIONAL

Восстановление реплик. Patroni

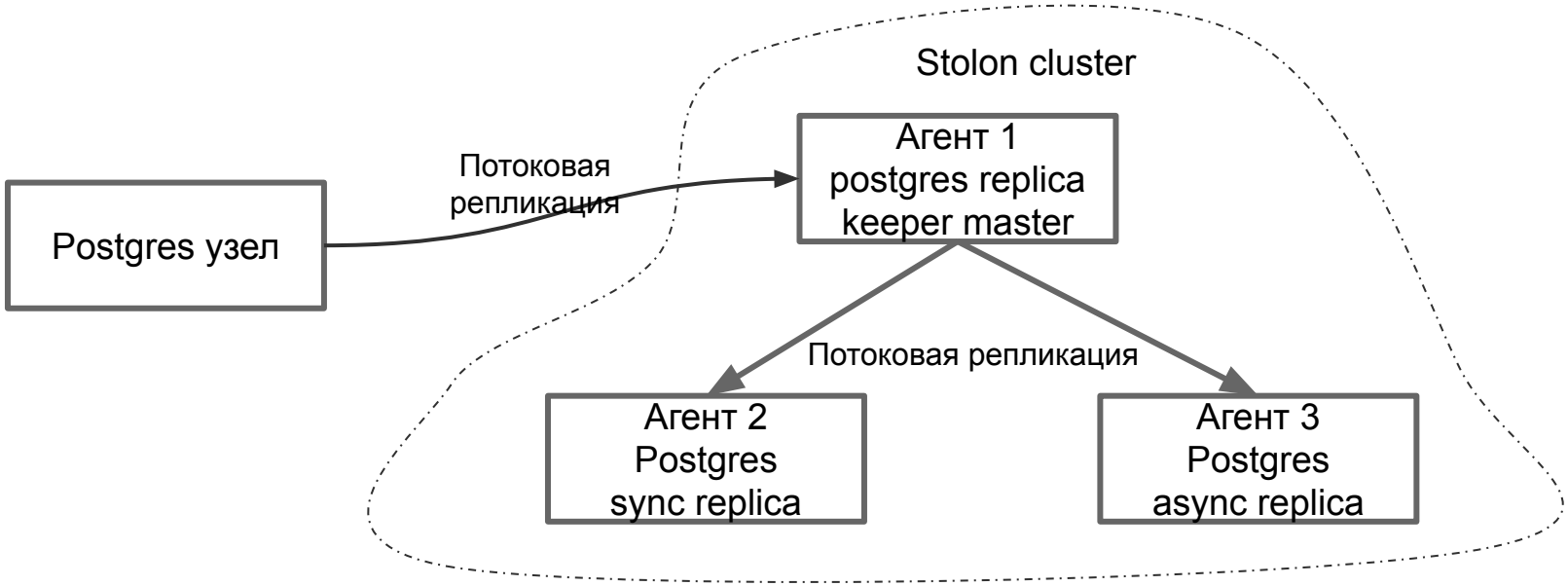
- Снятие “снимка” с мастера (**pg_basebackup**) по умолчанию
  - **единственная** стратегия в Stolon
- “Заливка” из бэкапа
  - актуально для больших баз (Тб+)

- Кастомизируемая последовательность стратегий восстановления

```
postgresql:
  create_replica_methods:
    - probackup
    - basebackup
  probackup:
    command: pg_probackup restore -B /path/to/backup -D <datadir> --instance <scope>
    keep_data: True
    no_params: True
  basebackup:
    max-rate: '100M'
    checkpoint: 'fast'
```

PROFESSIONAL Postgres

Инициализация кластера. Standby cluster

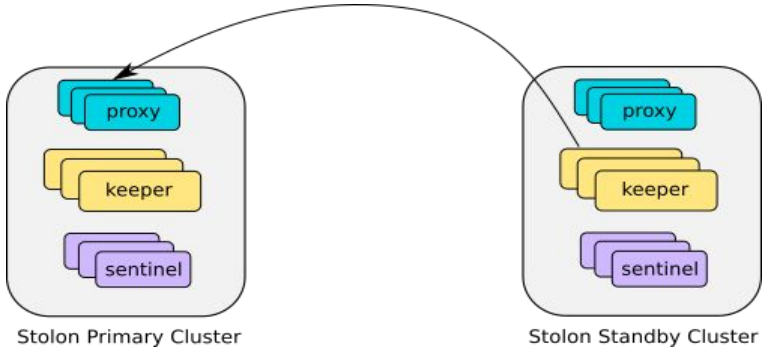


https://github.com/sorintlab/stolon/blob/master/doc/standbycluster.md

Инициализация кластера. Standby cluster

Применение:

- прозрачная миграция с существующей инсталляции (требуется всего лишь перезаливка данных)
- мульти ДЦ кластер (теоретически возможен)



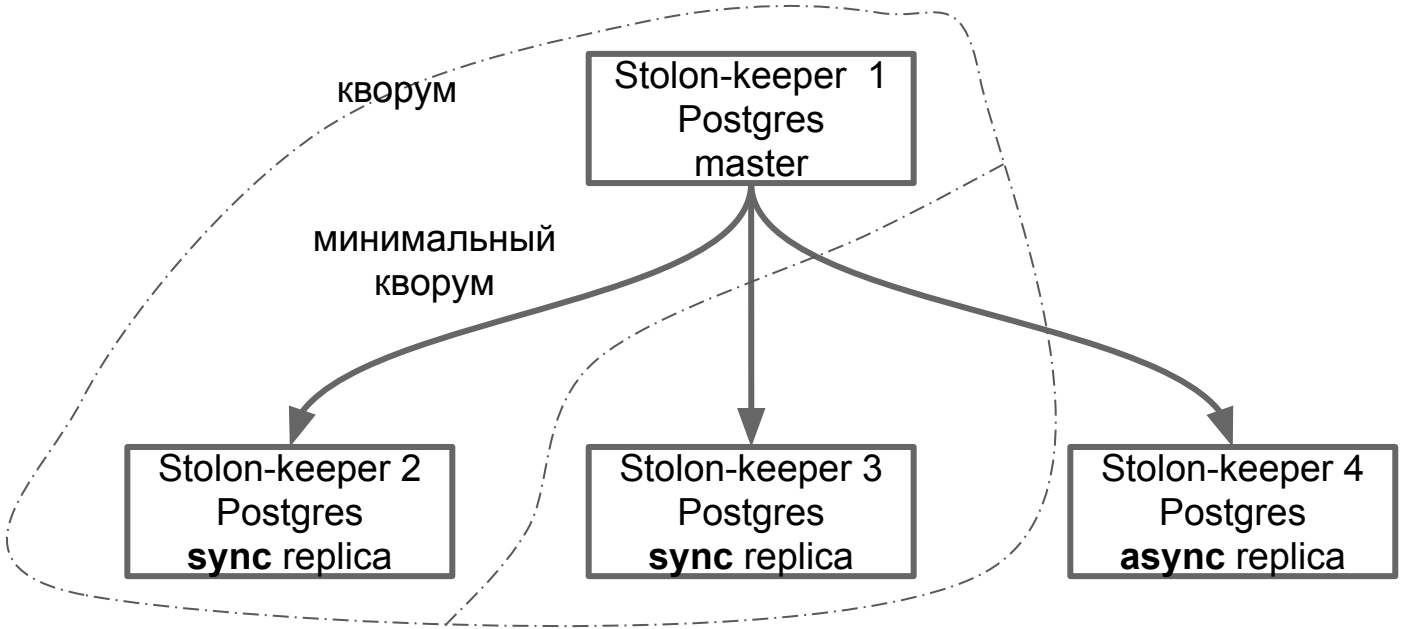
- *stolonctl* ***promote*** *…* для приведения кластера из standby режима в нормальный

Варианты репликации. Stolon

- по умолчанию асинхронная
- при синхронной можно формировать кворум на запись (полусинхронная репликация) параметрами [1]:
  - **minSynchronousStandbys** - кол-во синхронных реплик в кластере, ниже которого запись в БД блокируется
  - **maxSynchronousStandbys** - желаемое кол-во синхронных реплик
  - Patroni: кворум из 2-ух узлов - одна синхронная реплика в `synchronousStandbyNames` [2]
- асинхронная реплика не становится мастером:
  - при синхронной репликации
  - при асинхронной репликации, если отставание по транзакционным логам (WAL) от мастера превышает некоторый порог (**maxStandbyLag**)

1. https://github.com/sorintlab/stolon/blob/master/doc/syncrepl.md
2. https://patroni.readthedocs.io/en/latest/replication_modes.html#synchronous-mode-implementation

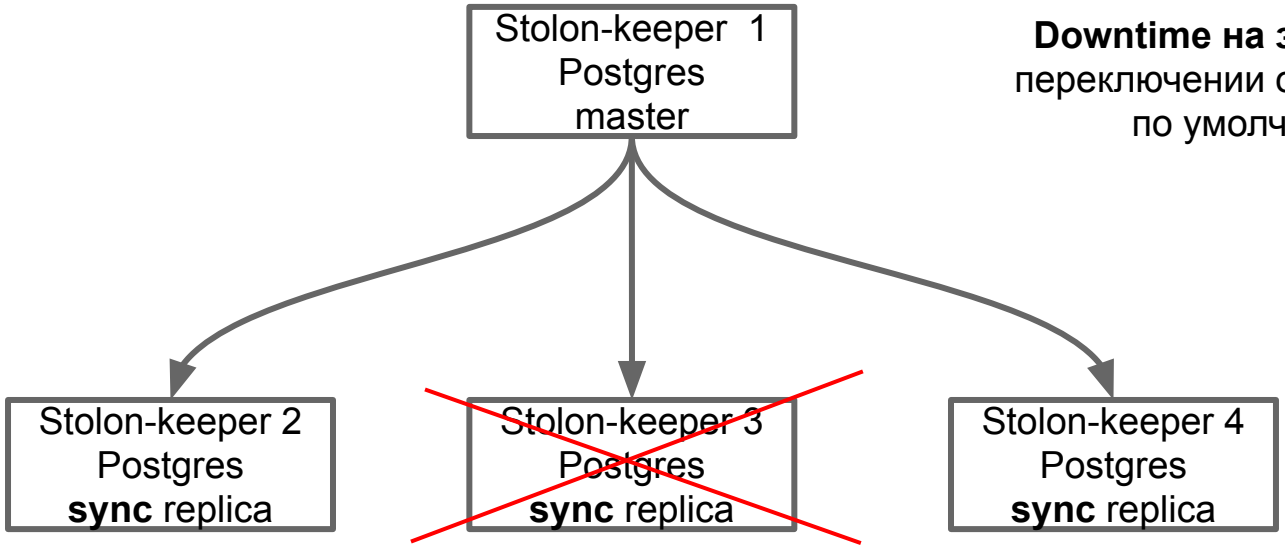
Полусинхронная репликация. Stolon



minSynchronousStandbys = 1

maxSynchronousStandbys = 2

Полусинхронная репликация. Stolon

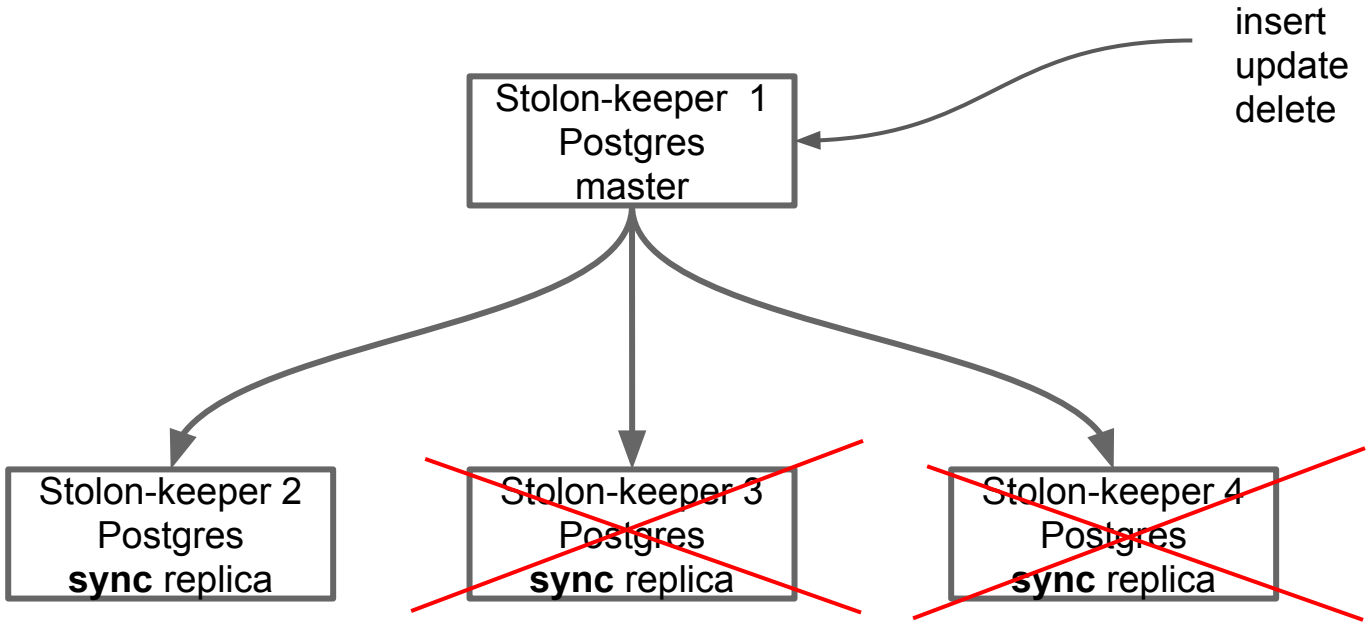


Downtime на запись при переключении от 20-40 сек. по умолчанию

minSynchronousStandbys = 1

maxSynchronousStandbys = 2

Полусинхронная репликация. Stolon

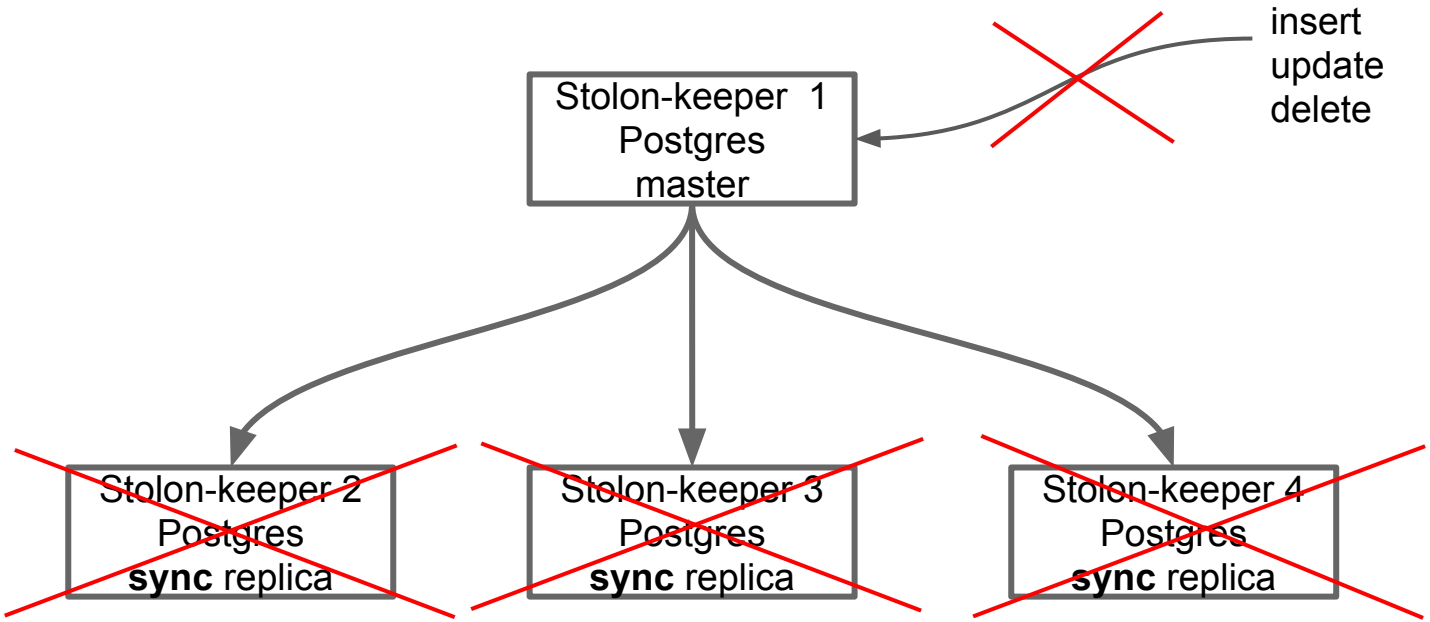


minSynchronousStandbys = 1

maxSynchronousStandbys = 2

Postgres PROFESSIONAL

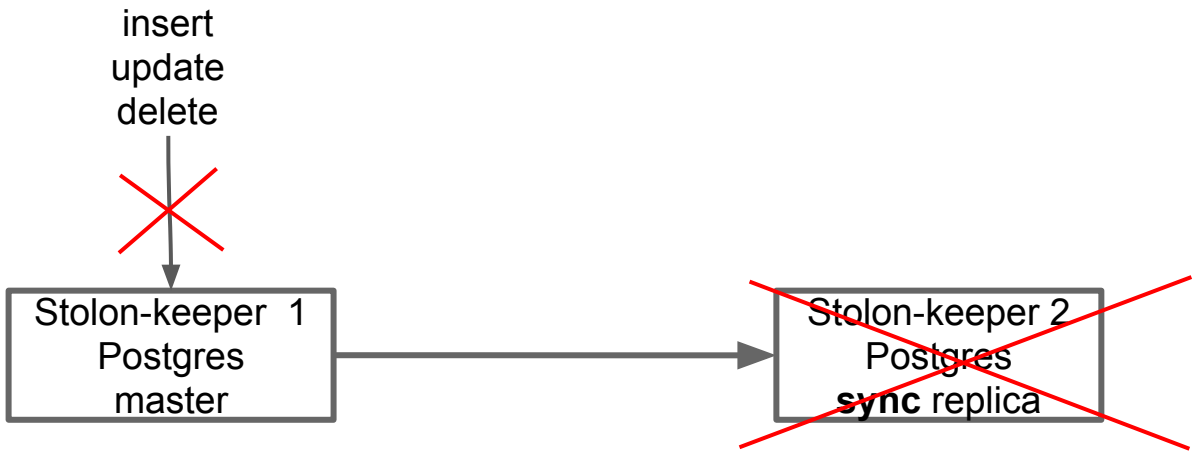
Полусинхронная репликация. Stolon



minSynchronousStandbys = 1

maxSynchronousStandbys = 2

Двухузловой кластер. Синхронная репликация. Stolon



minSynchronousStandbys >= 1

“Нестрогий” синхронный режим. Patroni

- `synchronous_mode_strict: off` (по умолчанию)
- Failover на не синхронизированную реплику не происходит
- Недоступность на запись в кластере при потере синхронной реплики по умолчанию, 20-30 сек.
  - При штатном выводе реплики из строя переключение в асинхронный режим практически мгновенный
- Переключение “догоняющей” реплики в синхронный режим происходит при полной синхронизации

Репликация. Stolon/Patroni

- Не используются встроенные кворумные методы (`FIRST` или `ANY`) репликации
  - список синхронных реплик не определён при падении мастера [1]
  - TODO: quorum commit (`ANY`) в Patroni [2]
- На каждый реплицирующий узел автоматически создаётся репликационный слот
  - в patroni определяется флагом **use_slots**
- При смене мастер роли возможна потеря реплики (позиции слотов не реплицируются)
  - подбор **wal_keep_segment** (**stolon**)
  - восстановление wal из архива (**patroni**)

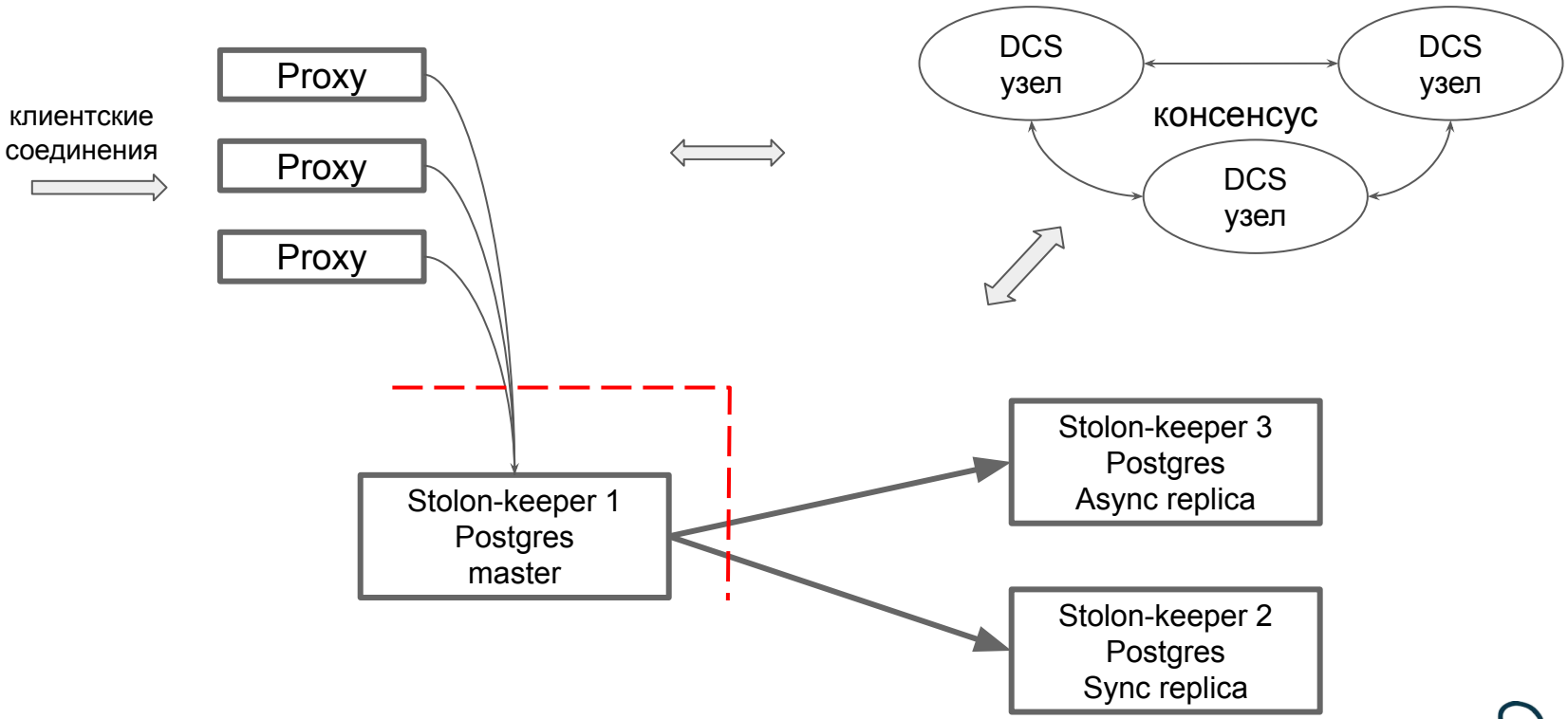
1. https://github.com/sorintlab/stolon/blob/master/doc/faq.md#does-stolon-uses-postgres-sync-replication-quorum-methods-first-or-any
2. https://github.com/zalando/patroni/pull/672

Изоляция неактуального мастера (fencing)

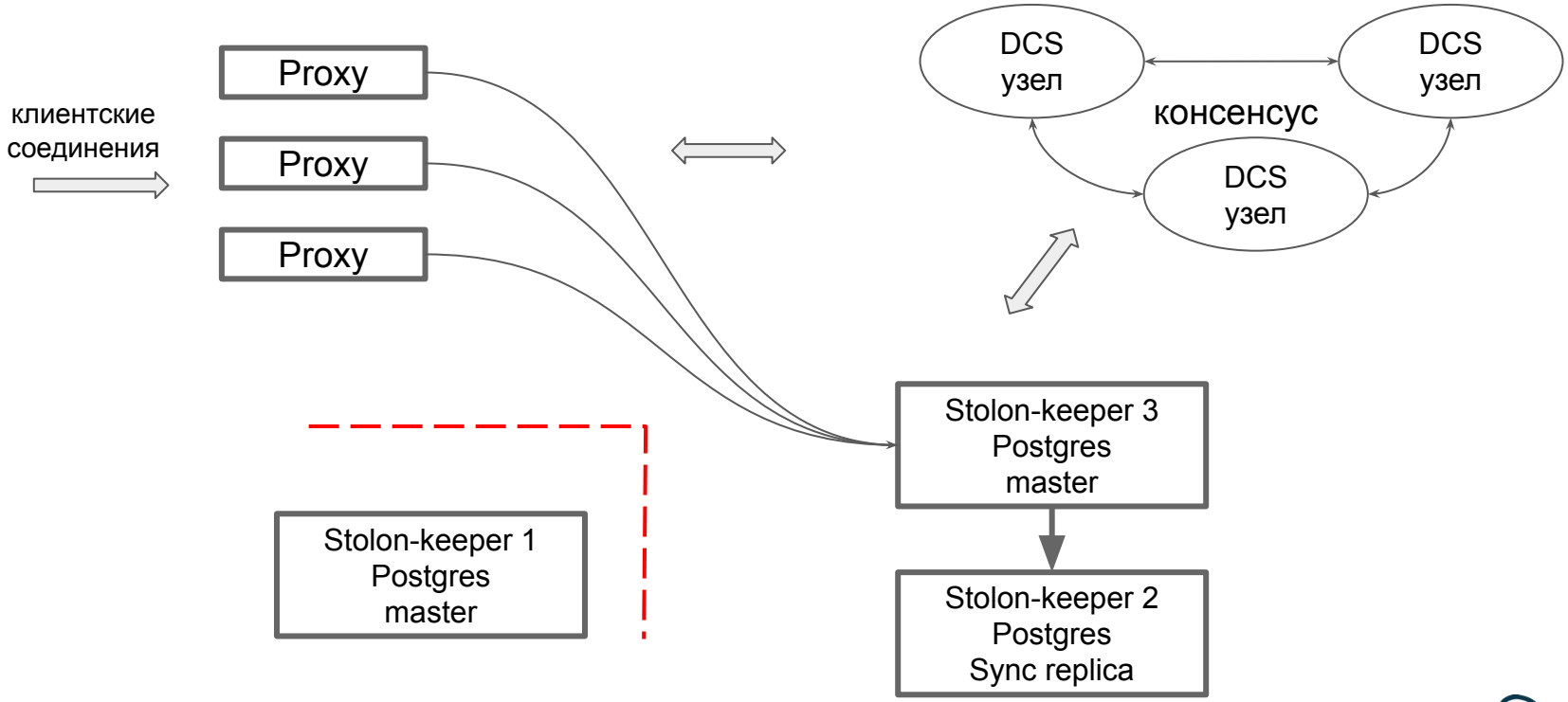
- “Ограждение” от изменений мастера, изолированного от основного кластера (избегание **split brain**):
  - выключение узла (экземпляра PostgreSQL) - STONITH (**patroni + watchdog** [1])
  - разрыв всех текущих соединений и отказ от приёма новых (**stolon**)
  - перевод БД в режим read-only (**patroni**)
- **Привязка к кворуму DCS**
- **Stolon-proxy**, будучи TCP-прокси на мастер узел, является “fencer” узлов БД
  - связка **cond + HAProxy** эмулирует stolon-proxy в **Patroni**
- Фенсинг изолированных stolon-proxy происходит по таймауту (15 сек.)

1. https://github.com/zalando/patroni/blob/master/docs/watchdog.rst

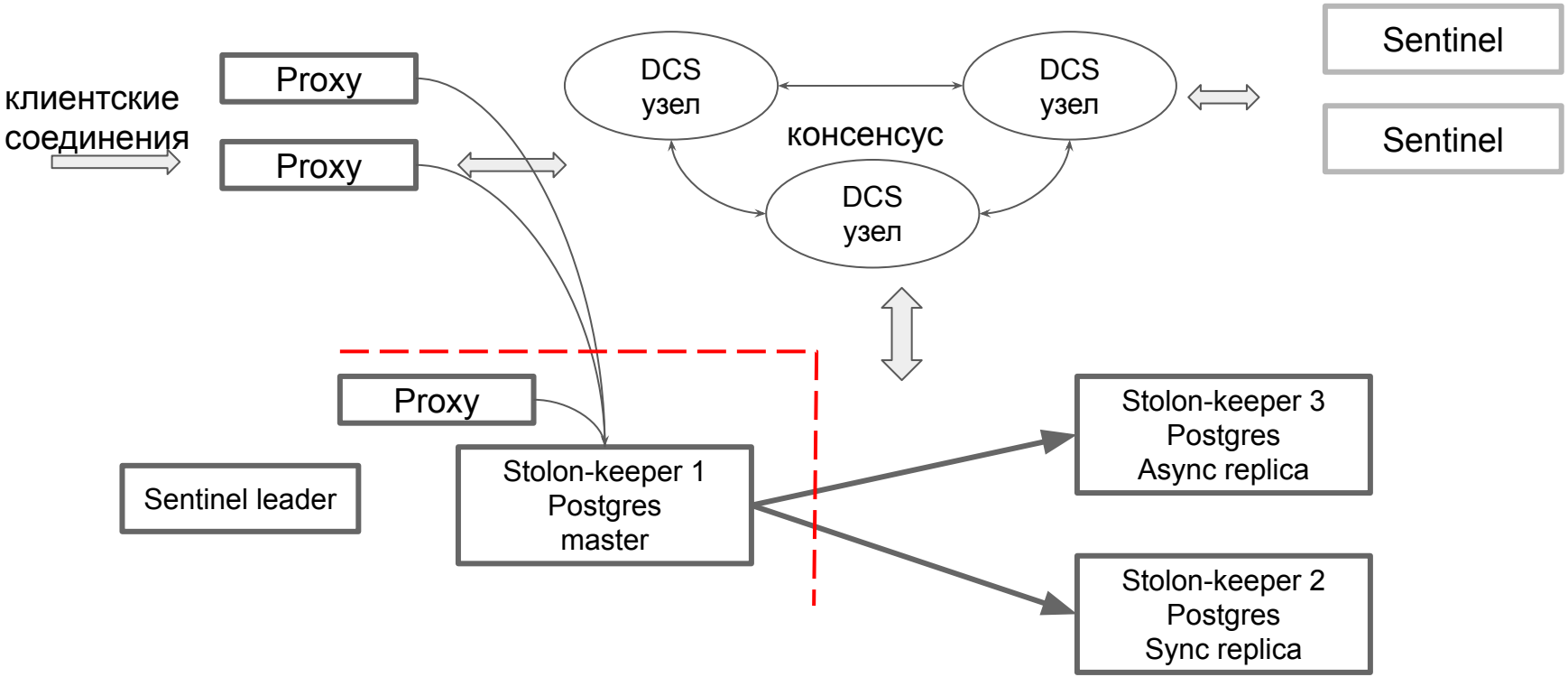
Изоляция неактуального мастера (fencing)



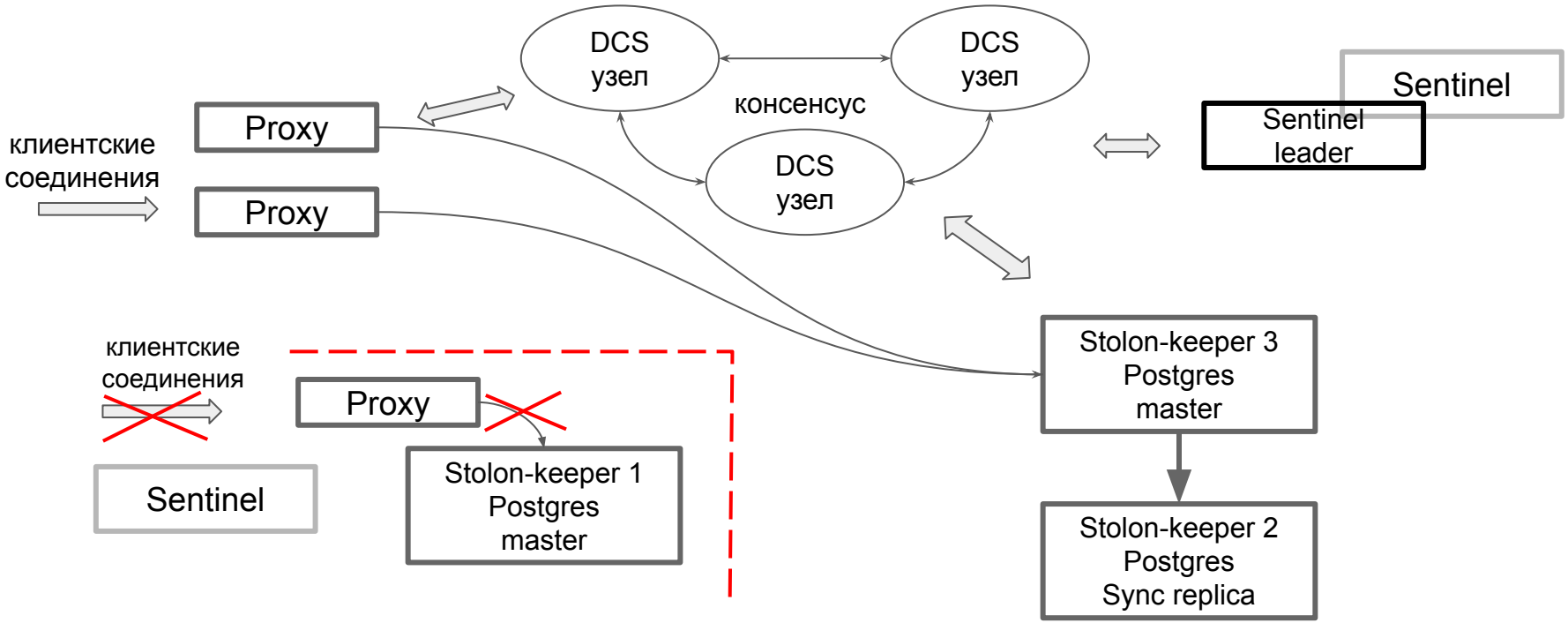
Изоляция неактуального мастера (fencing)



Изоляция неактуального мастера (fencing)

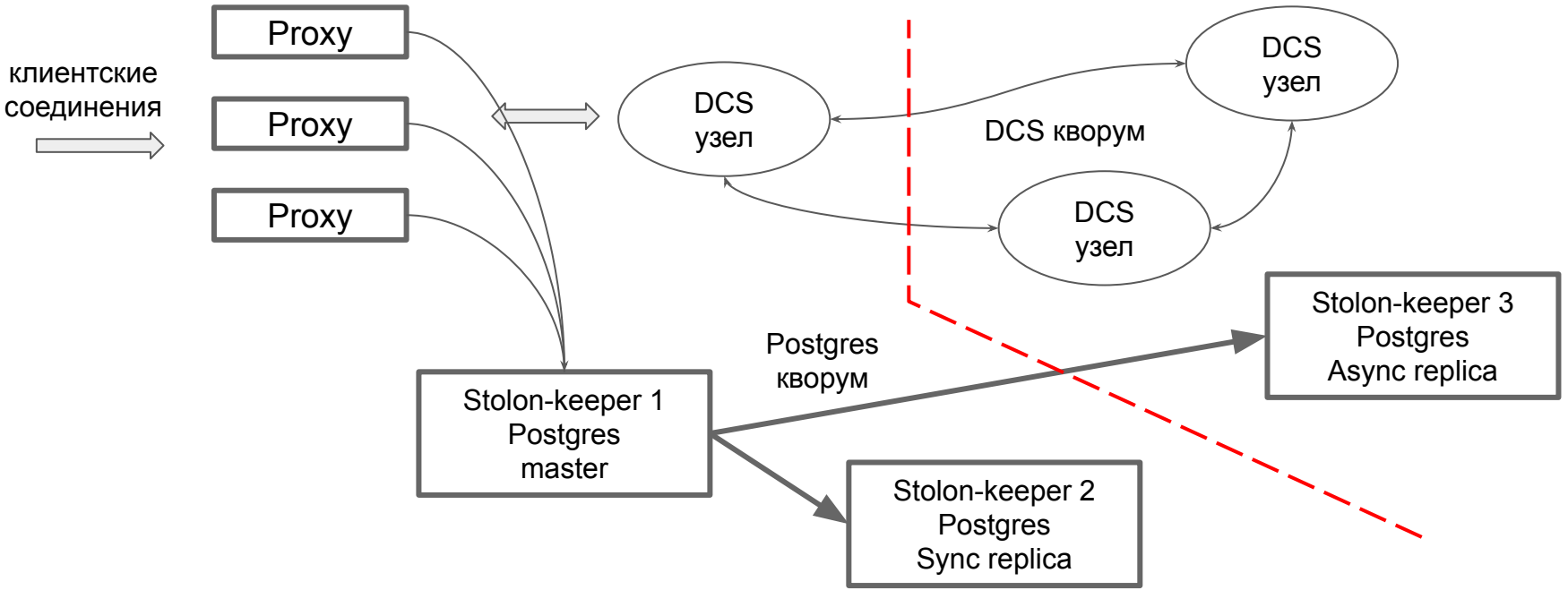


Изоляция неактуального мастера (fencing)

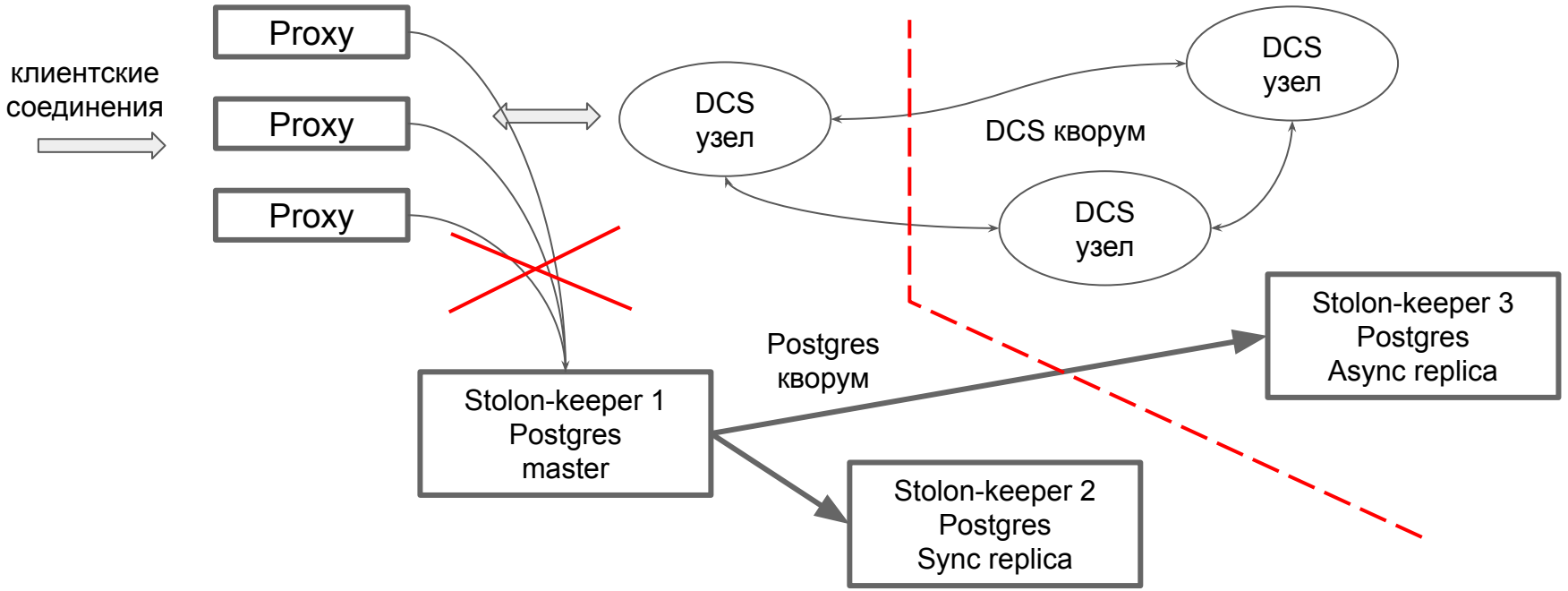


Фейловер **stolon-sentinel** лидера за **20 сек**
Stolon-proxy фенсится за **15 сек**
Stolon-sentinel leader - по умолчанию, за **15 сек**

Изоляция неактуального мастера (fencing)

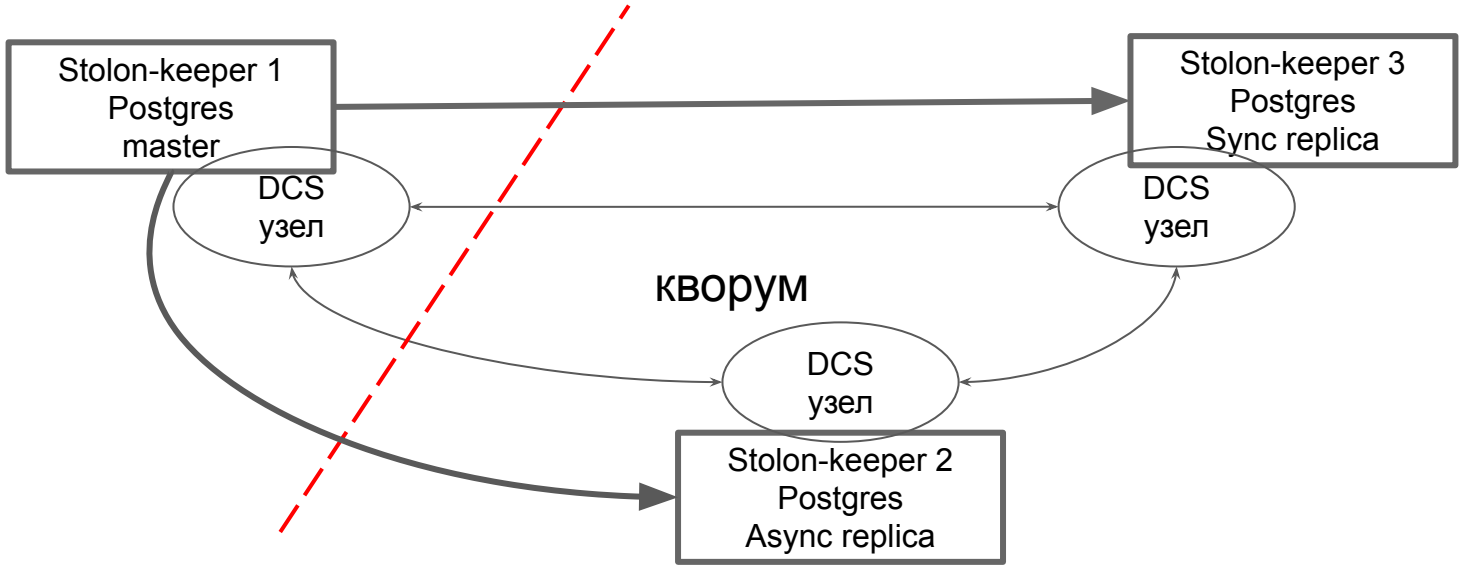


Изоляция неактуального мастера (fencing)

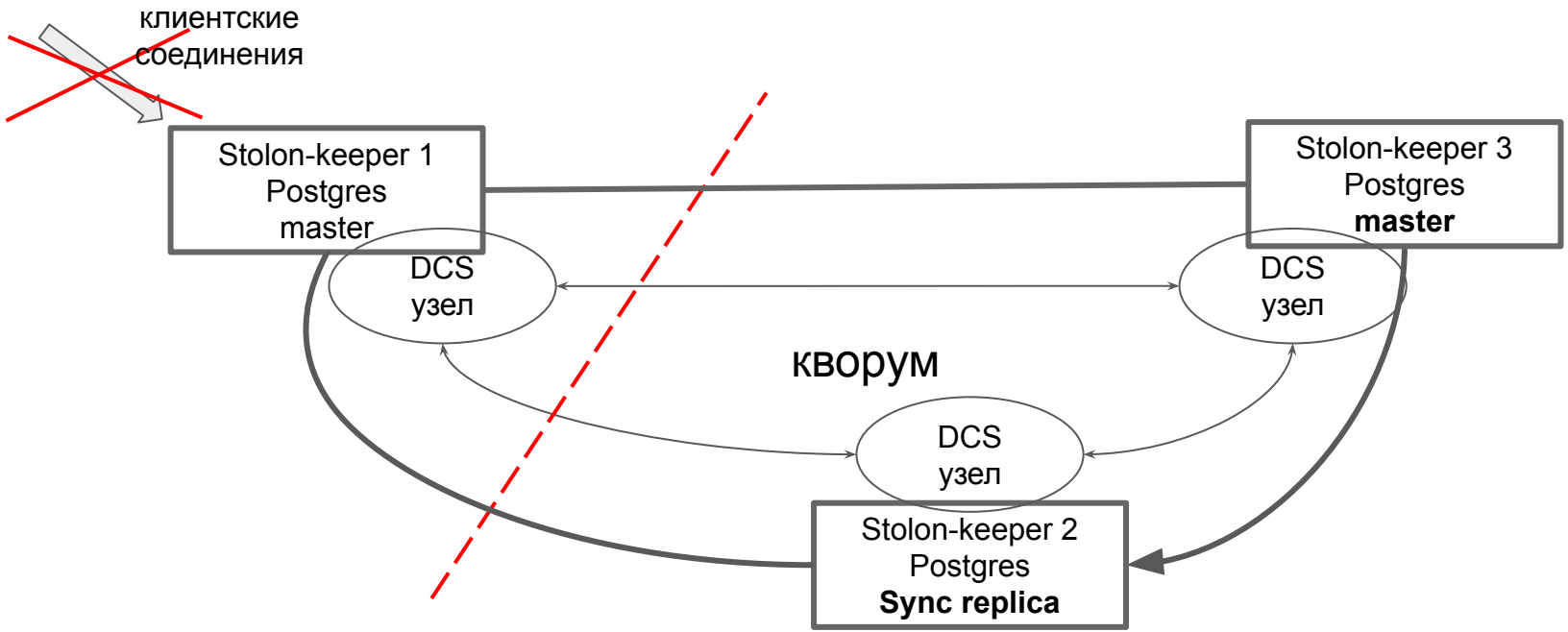


Привязка к кворуму DCS
synchronousReplication: true

Изоляция неактуального мастера (fencing)



Изоляция неактуального мастера (fencing)



Аварийное переключение master-роли (failover)

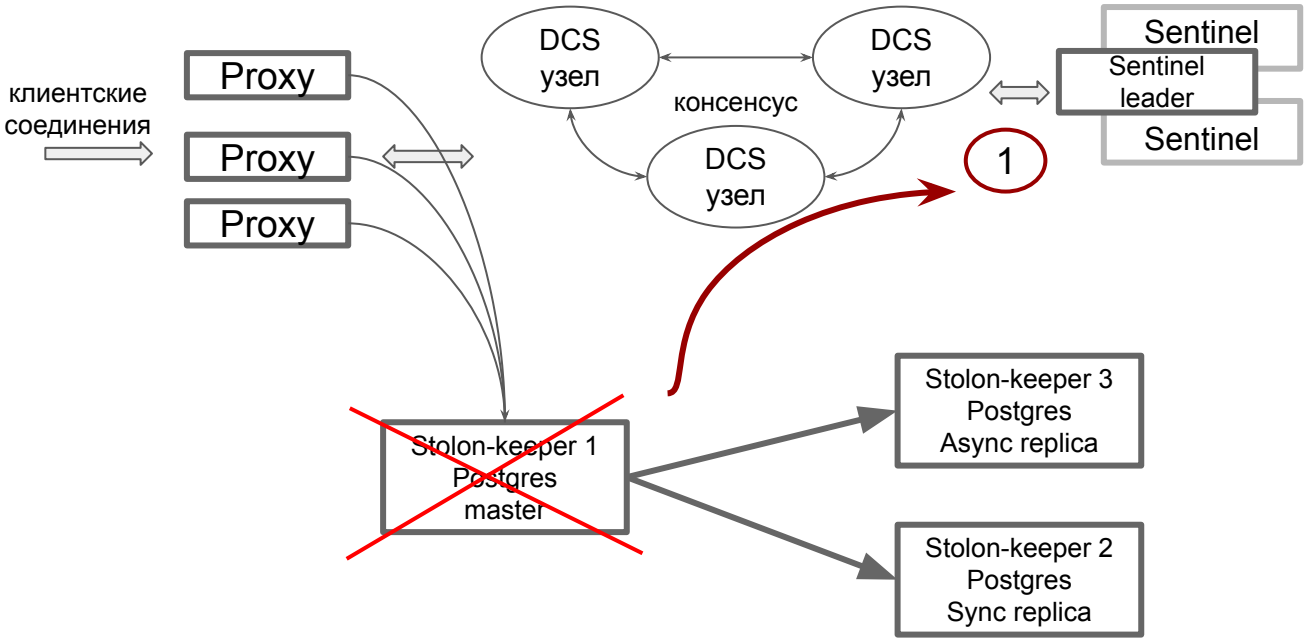
- Автоматическое восстановление в кластер бывшего мастера
  - “бесшовная” синхронизация (**recovery_target_timeline=latest**) (patroni)
  - откат не реплицированных изменений (**pg_rewind)**
  - перезаливка узла от нового мастера или из бэкапа (patroni)
- *Время недоступности кластера =*

  длительность выявления сбоя мастера +

  длительность переключения мастер роли +

  длительность обнаружения нового мастера клиентами

Failover. Stolon

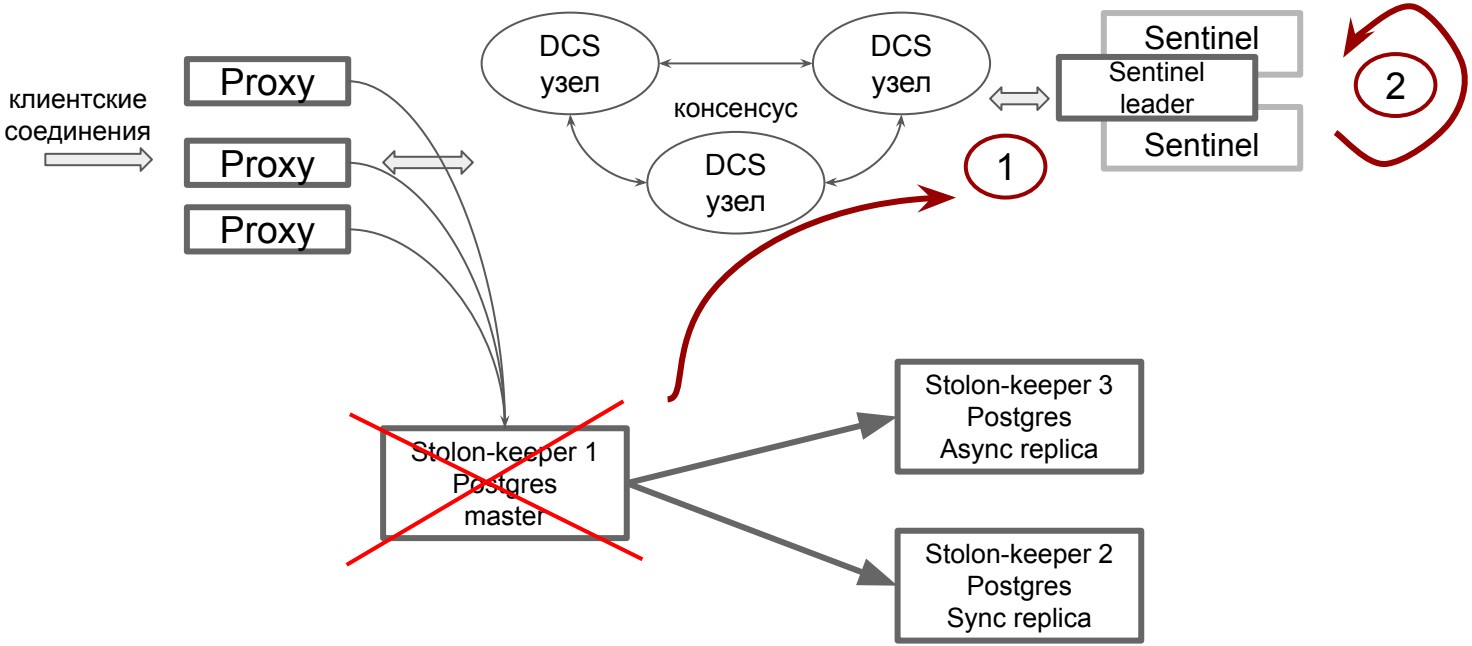


1. **Первичное обнаружение сбоя мастера арбитром**

   $(\lambda_1 + \lambda_2)$ * *sleepInterval*, где: **(до 10 сек по умолчанию)**

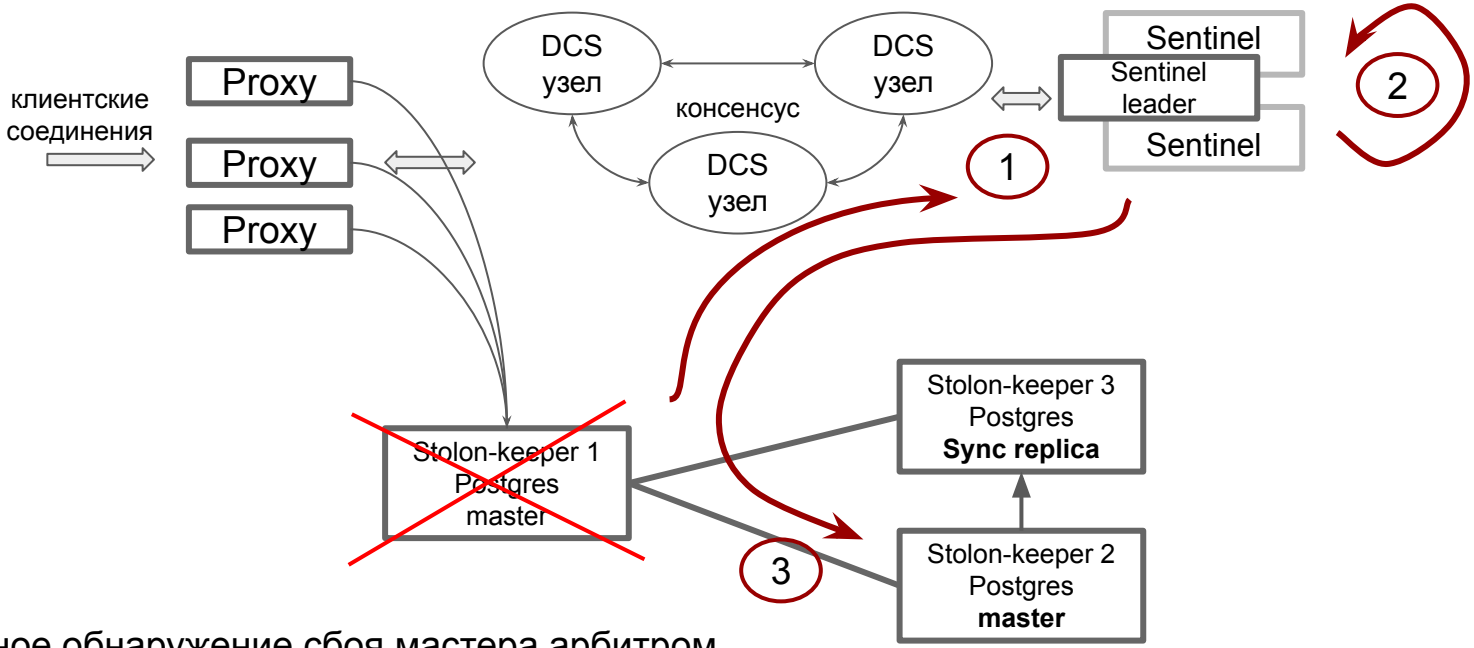
   - λ_i - случайная величина от 0 до 1
   - *sleepInterval* - период срабатывания каждого компонента (keeper, sentinel, proxy) в цикле (по умолчанию, 5 сек)

Failover. Stolon



1. Первичное обнаружение сбоя мастера арбитром
2. **Таймаут подтверждения сбоя (**параметр *failInterval,* **по умолчанию, 20 сек)**

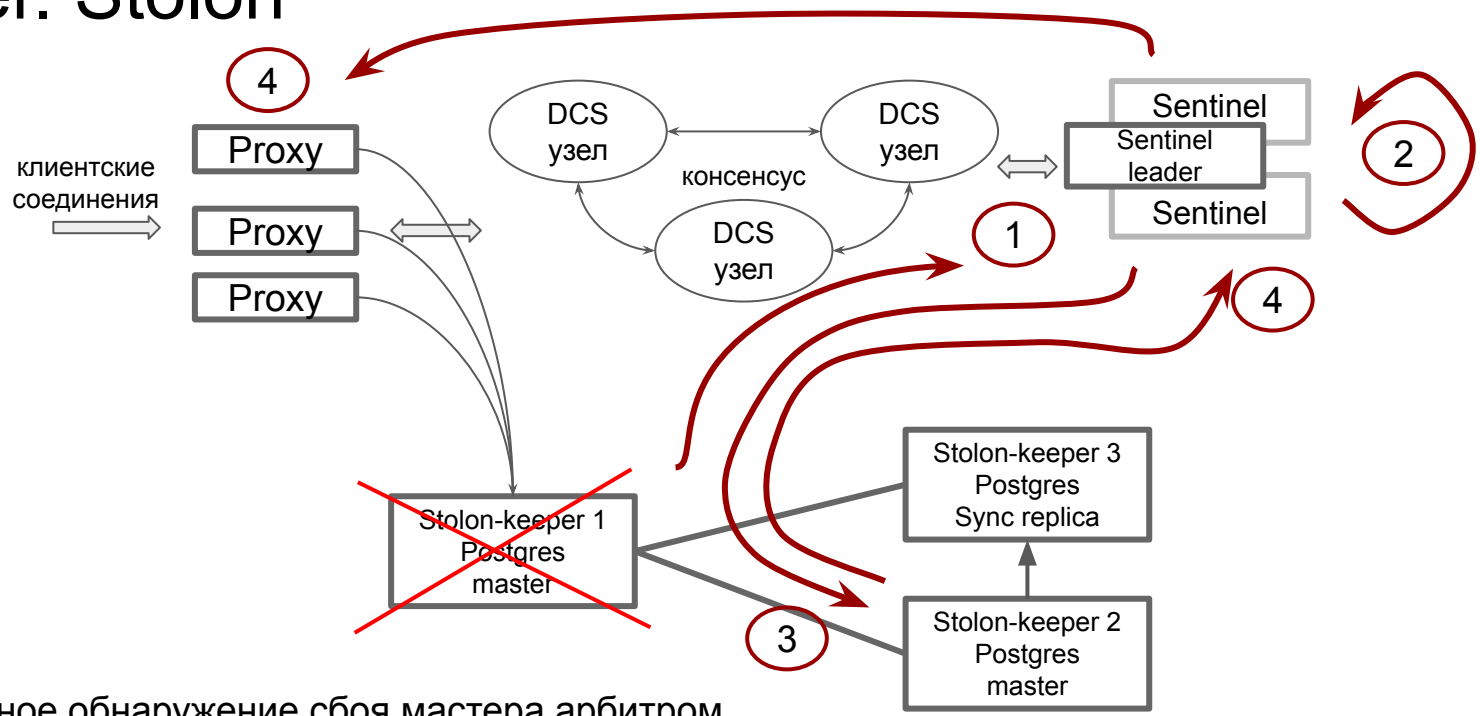
Failover. Stolon



1. Первичное обнаружение сбоя мастера арбитром
2. Таймаут подтверждения сбоя
3. **Вычисление нового мастера и promote выбранной реплики**

   $(\lambda_1 + \lambda_2)$ * *sleepInterval* (**по умолчанию, до 10 сек**)

Failover. Stolon



1. Первичное обнаружение сбоя мастера арбитром
2. Таймаут подтверждения сбоя
3. Время вычисления нового мастера и promote выбранной реплики
4. **Переподключение прокси на новый мастер**

   $(\lambda_1 + \lambda_2)$ * *sleepInterval* (**по умолчанию, до 10 сек**)

Failover. Stolon

- *Время недоступности кластера* =

  $(\lambda_1 + \lambda_2) * sleepInterval + failInterval +$

  $(\lambda_3 + \lambda_4) * sleepInterval +$

  $(\lambda_5 + \lambda_6) * sleepInterval$

  где:

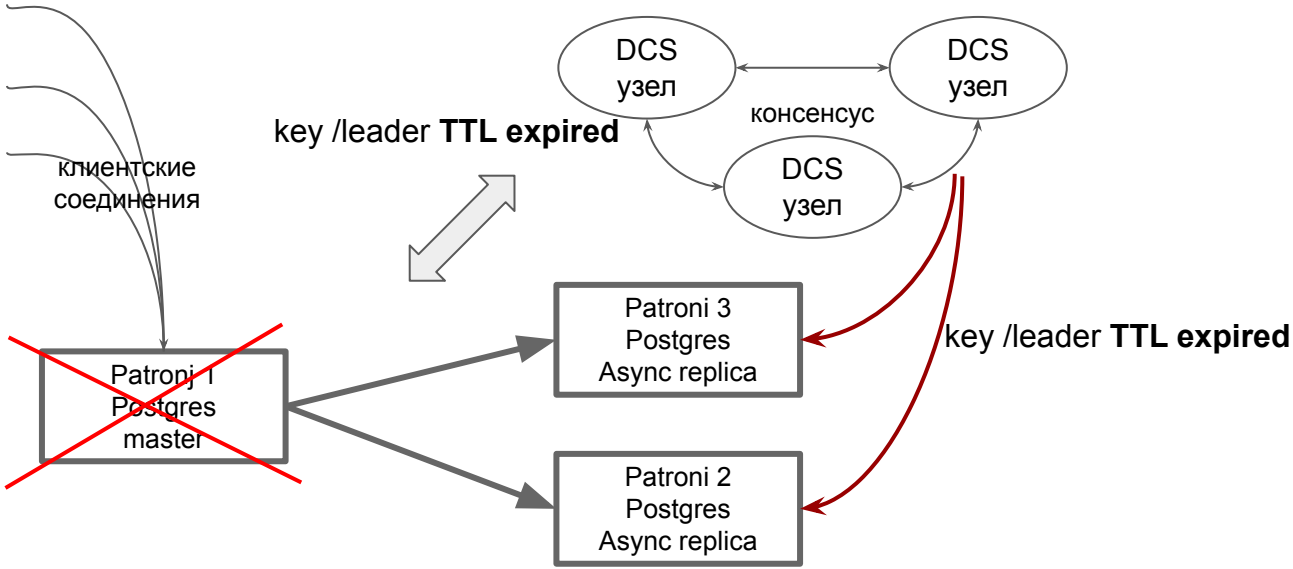
  - λ_i - случайная величина от 0 до 1
  - *sleepInterval* - период срабатывания каждого компонента (keeper, sentinel, proxy) в цикле (по умолчанию, 5 сек)
  - *failInterval* - таймаут подтверждения сбоя (по умолчанию, 20 сек)

  при допушении, что

  - время записи/чтения из DCS пренебрежимо мало
  - Raft лидер / Sentinel лидер работоспособны (**+ 20 сек. при sentinel failover**)
- С настройками по умолчанию, **от 20 до 50 сек.**

PROFESSIONAL Postgres

Failover. Patroni

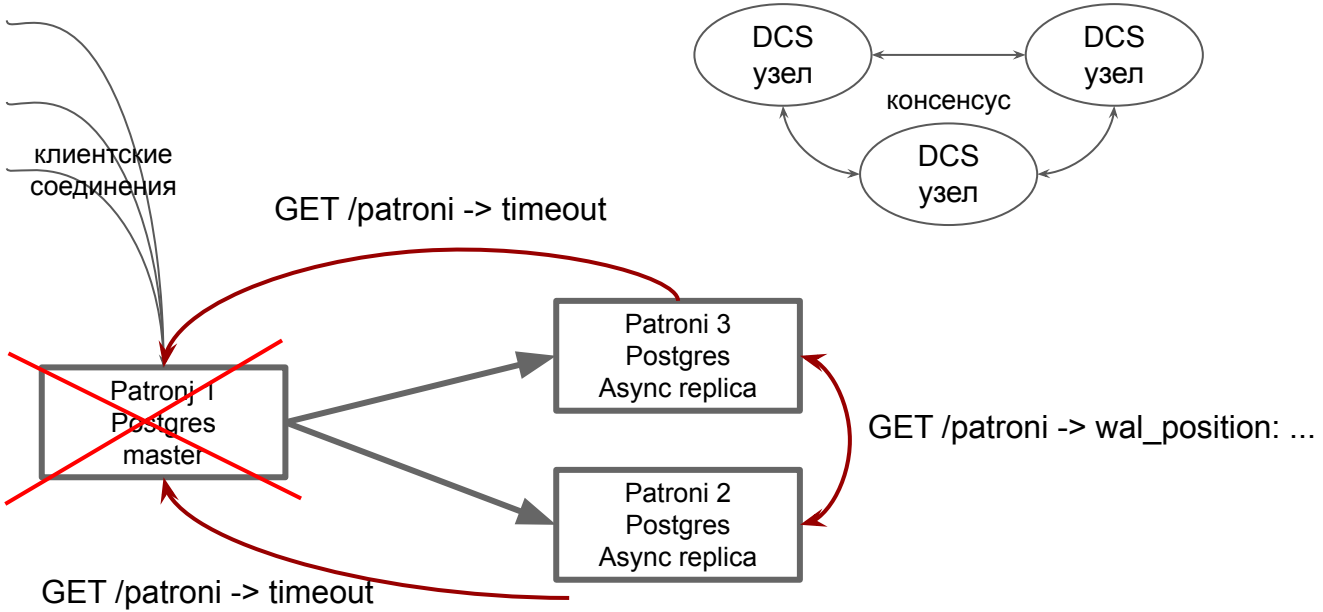


1. **Выявление сбоя мастера**

   *ttl - λ * loop_wait*, где: **(20-30 сек по умолчанию)**

   - *λ* - случайная величина от 0 до 1
   - *ttl* - TTL ключа лидера в DCS
   - *loop_wait* - период срабатывания patroni агентов

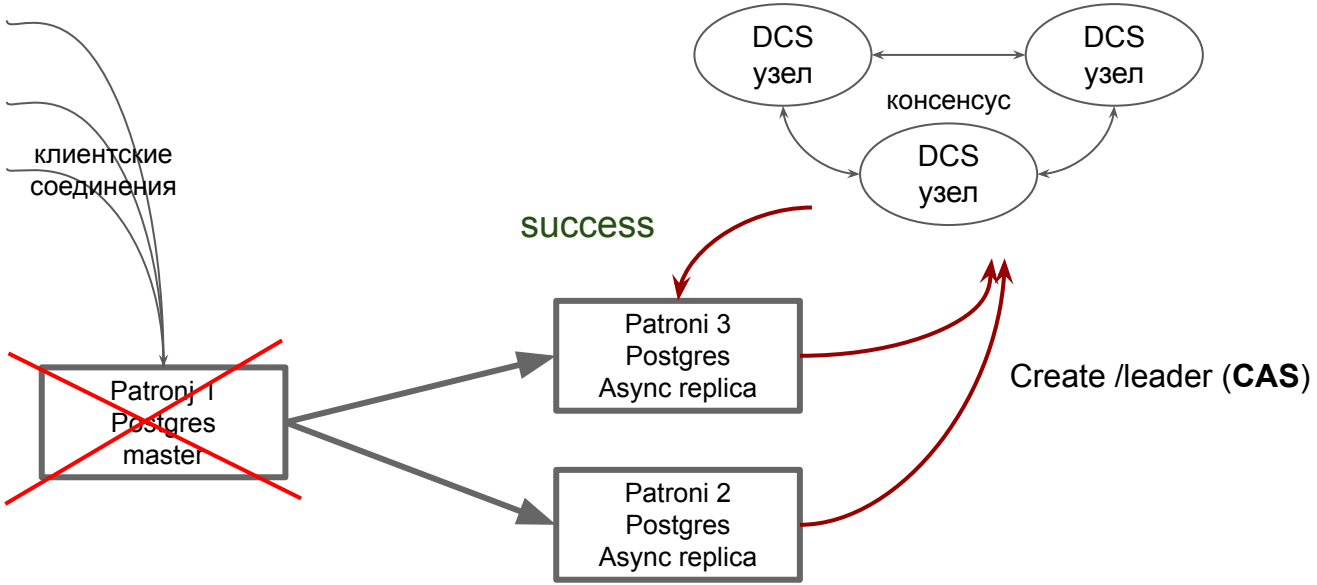
Failover. Patroni



1. Выявление сбоя мастера
2. **Проверка сбоя и определение кандидатов для нового лидера (2 сек.)**

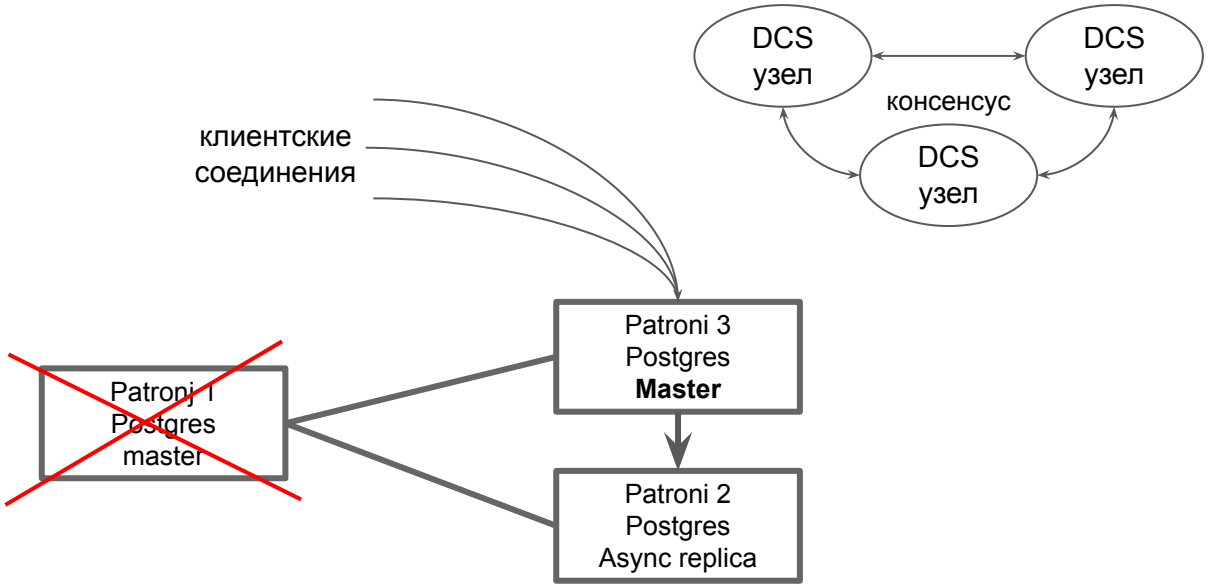
   2 сек. - timeout на GET /patroni

Failover. Patroni



1. Выявление сбоя мастера
2. Проверка сбоя и определение кандидатов для нового лидера
3. **Выборы нового лидера (практически мгновенно)**

Failover. Patroni



1. Выявление сбоя мастера
2. Проверка сбоя и определение кандидатов для нового лидера
3. Выборы нового лидера (практически мгновенно)
4. **Promote выбранного лидера и переподключение клиентов**

   promote мгновенный, время переподключения зависит от типа прокси

Failover. Patroni

- *Время недоступности кластера* =

  *ttl - ƛ * loop_wait +*

  2 +

  client_reconnection_timeout

  где:

  - *ƛ* - случайная величина от 0 до 1
  - ttl - TTL ключа лидера в DCS
  - *loop_wait* - период срабатывания patroni агента в цикле (по умолчанию, 10 сек)
  - *client_reconnection_timeout* - время переподключения клиентов к мастеру

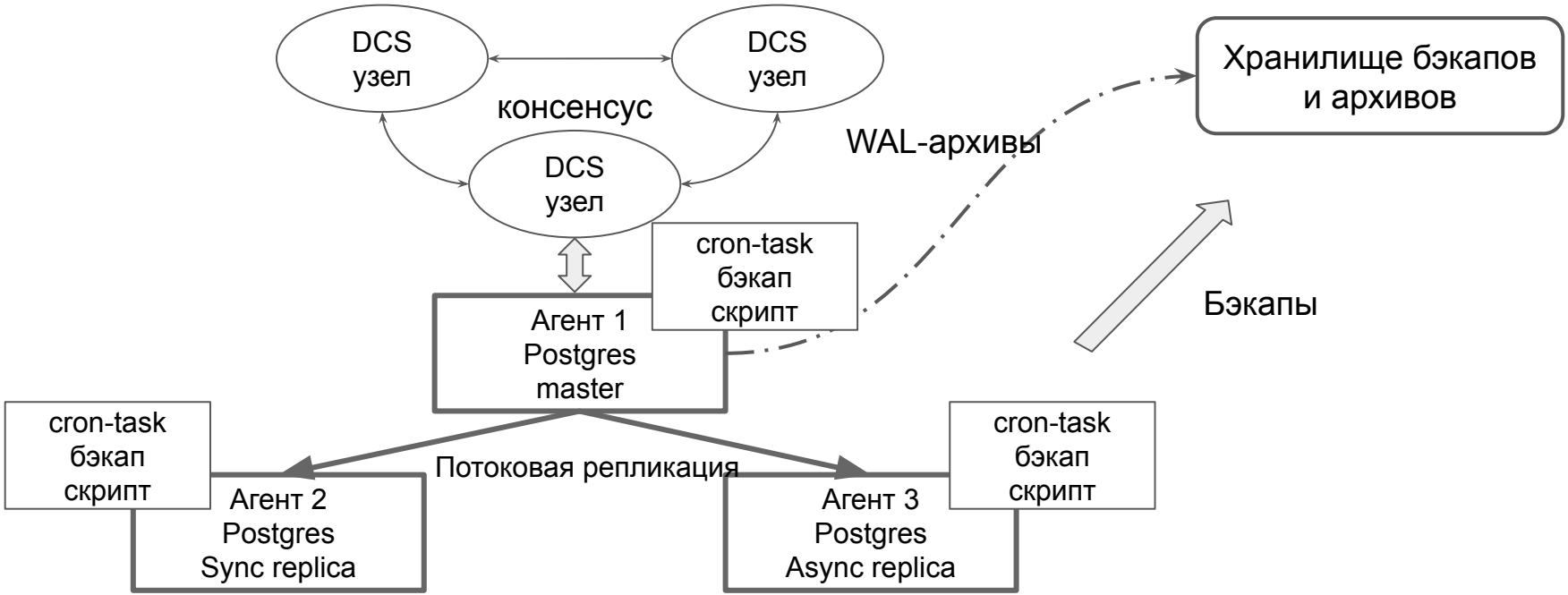
  при допушении, что

  - время записи/чтения из DCS пренебрежимо мало
- С настройками по умолчанию, **от 22 до 37 сек.**
  - время переподключения, положим, до 5 сек.

Плановое переключение мастер роли (switchover)

- **Stolon**
  - **stolonctl failkeeper …** универсальная команда временного “отказа” узла
  - Нет таймаута на подтверждение сбоя
  - Способ планово вывести узел из кластера
  - Master switchover: **stolonctl failkeeper master_node …**
  - Sync standby switchover: **stolonctl failkeeper sync_standby_node …**
  - Узел-кандидат на мастер роль выбирается **автоматически** среди наиболее “продвинутых” по реплицированным WAL
- **Patroni**
  - **patronictl switchover --master=... --candidate=...** направляемый switchover 
  - Switchover по расписанию

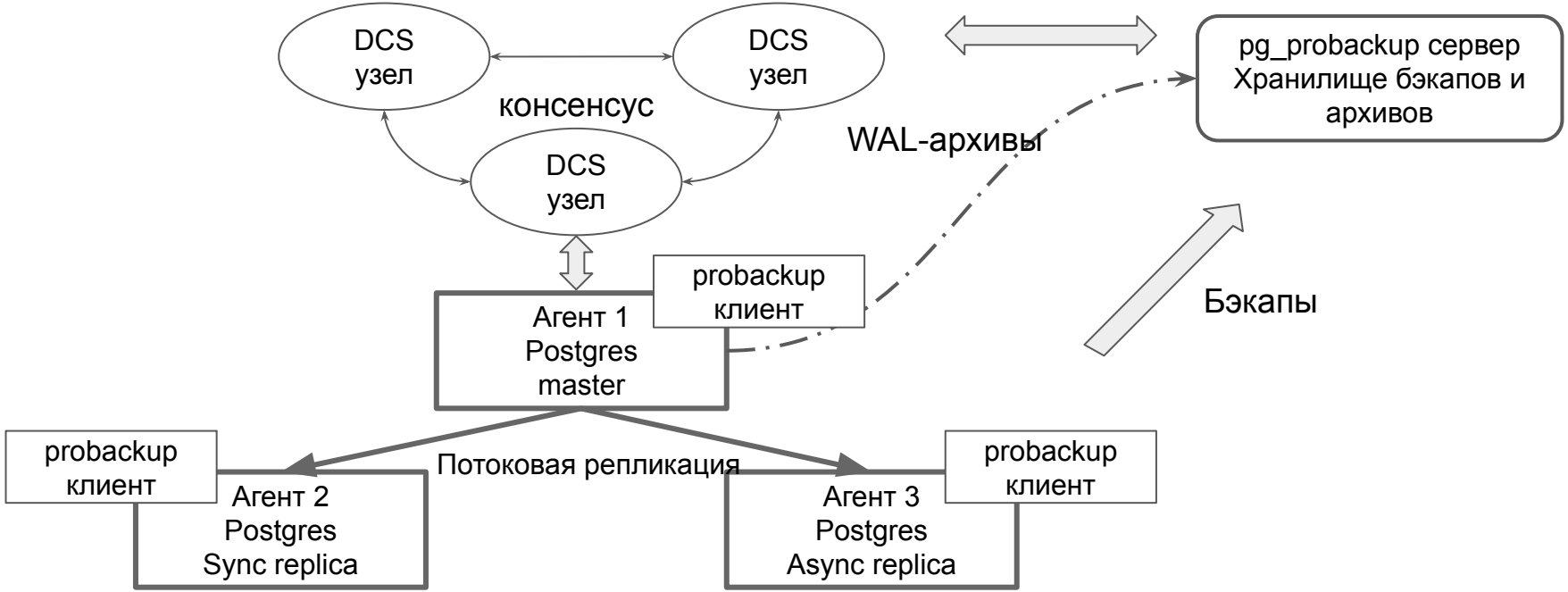
Бэкап кластера



Бэкап кластера

- Полные и инкрементные бэкапы создаются кастомными скриптами по плану (cron)
- Роль узла в кластере можно узнать запросом к DCS
- Архивные транзакционные логи (WAL):
  - сегментами в 16 Мб с мастер узла (**archive_command=on**)
  - потоком по протоколу физической репликации (**pg_receive_wal**)
- Наличие “внешнего” слота репликации для pg_receive_wal (параметр **additionalMasterReplicationSlots** в **stolon**)
  - пересоздание слота автоматическое при смене мастер роли

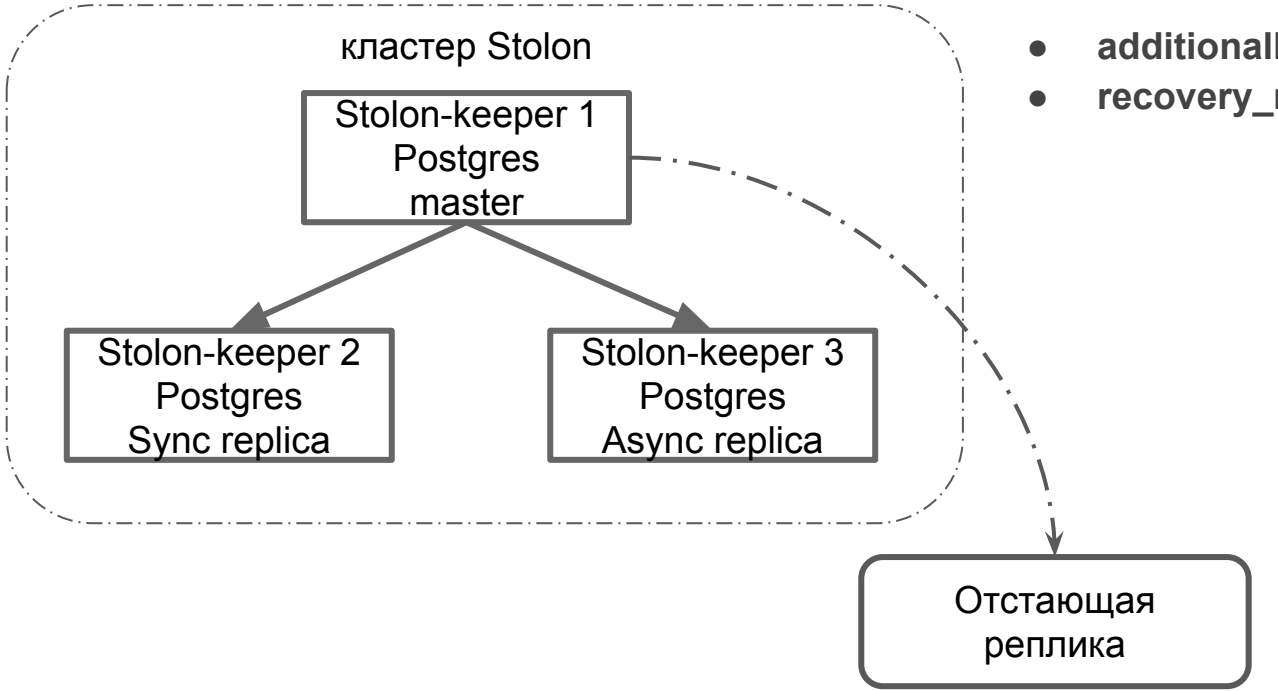
Бэкап кластера с pg_probackup



pg_probackup с поддержкой удалённых бэкапов

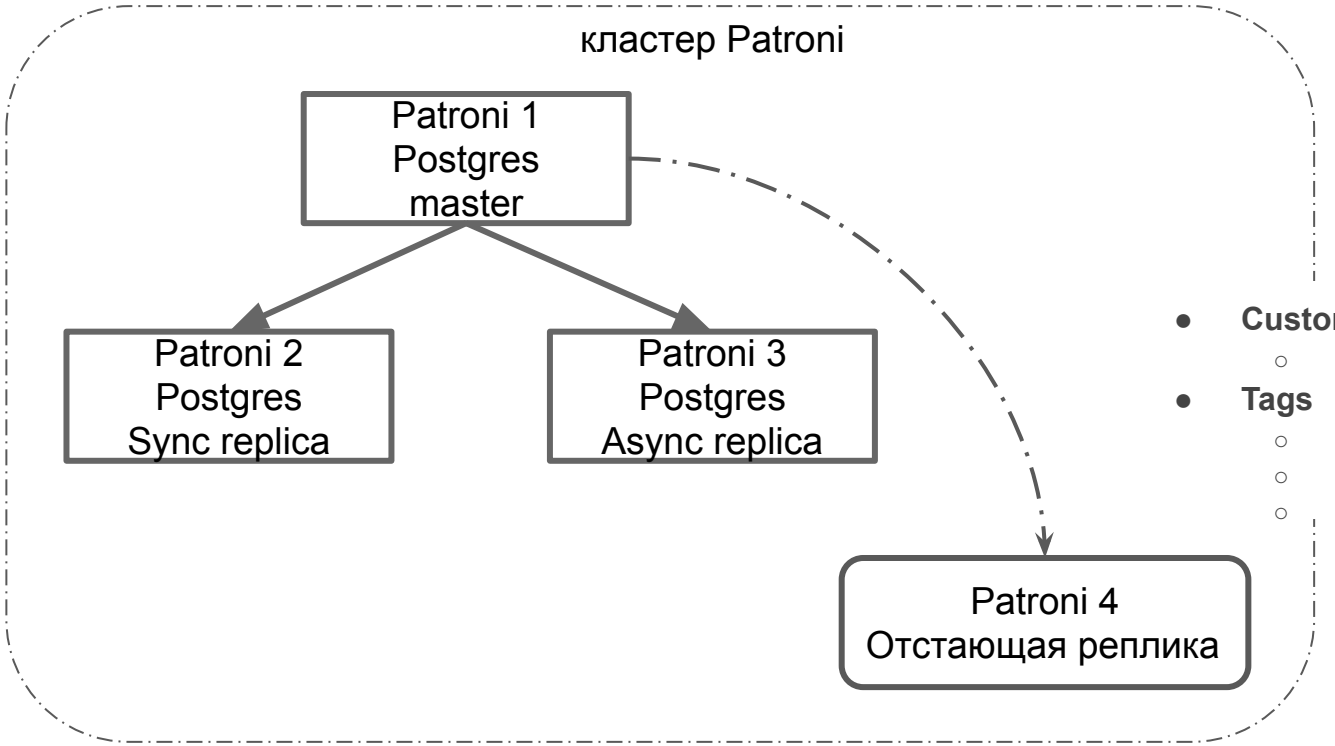
Postgres PROFESSIONAL

Поддержка “отстающих” реплик. Stolon



- **additionalMasterReplicationSlots**
- **recovery_min_apply_delay = '12h'**

Поддержка “отстающих” реплик. Patroni



- **Custom recovery.conf**
  - **recovery_min_apply_delay = '12h'**
- **Tags**
  - **nosync**
  - **nofailover**
  - **nobalance**

Легковесное восстановление из отстающих реплик

- Определить момент повреждения данных (логи, расширение **pgAudit**, кастомные расширения)
- Отсоединить отстающую реплику от мастера и настроить восстановление до момента повреждения
- Получить из восстановленной реплики консистентные данные и “залить” их в мастер
- Заново настроить репликацию с отстающим проигрыванием изменений от мастер узла в кластере

https://www.percona.com/blog/2018/06/28/faster-point-in-time-recovery-pitr-postgresql-using-delayed-standby/

Postgres PROFESSIONAL

Мониторинг кластера. Stolon

- **stolonctl status …** человекочитаемый формат
  - списки всех компонентов (keeper, sentinel, proxy)
  - состояния (health checks) агентов stolon-keeper и экземпляров PostreSQL
  - дерево репликации узлов БД
- **stolonctl clusterdata …** - json-представление состояния кластера в DSC (машиночитаемый формат)
  - позиции проигранных WAL => задержка репликации для всех узлов кластера (аналог **pg_stat_replication** с задержкой)
  - роли (sync/async) для реплик

Мониторинг кластера. Patroni

- **patronictl list …** человекочитаемый формат
  - состояния узлов (действия, выполняемые в данный момент)
  - роли узлов БД (master/sync/async)
  - задержка репликации
- **GET /patroni …** - json-представление состояния узла
  - GET /master GET /replica - роли узлов
- KV хранилище DCS

Резюмируя

Stolon vs. Patroni

- Patroni богаче на фичи (enterprise уровень)
- Stolon проще и удобнее в настройке и в эксплуатации
- SQL интерфейс - единственное средство контроля БД (общий недостаток сторонних решений)
  - Нет возможности:
    - следить за логами
    - видеть внутреннее состояния PostgreSQL
  - **Неявные** таймауты, убивающие слабо контролируемые процессы (recovery)
    - **convergenceTimeout**, **initTimeout**, **syncTimeout** в stolon
- В ожидании встроенного автофейловера
  - MongoDB [1], CocroachDB [2]

1. http://openlife.cc/system/files/4-modifications-for-Raft-consensus.pdf
2. https://www.cockroachlabs.com/docs/stable/architecture/replication-layer.html

Спасибо за внимание!

Максим Милютин m.milyutin@postgrespro.ru