Максим Милютин
m.milyutin@gmail.com

мастер-класс

Patroni и Stolon:

Инсталляция, настройка и отработка падений

План мастер-класса

- Поэтапная инсталляция кластеров на 3-х виртуальных узлах:
  - Etcd
  - Stolon
  - Patroni
  - HAProxy + Confd
- Разбор основных падений
- Инсталляция сделана по экспериментальным ansible ролям и плейбукам, подготовленным внутри PostgresPro
  - существуют готовые ansible плейбуки кластера Patroni “под ключ” [1]

1. https://github.com/vitabaks/postgresql_cluster

Введение

Кластера на базе физической репликации

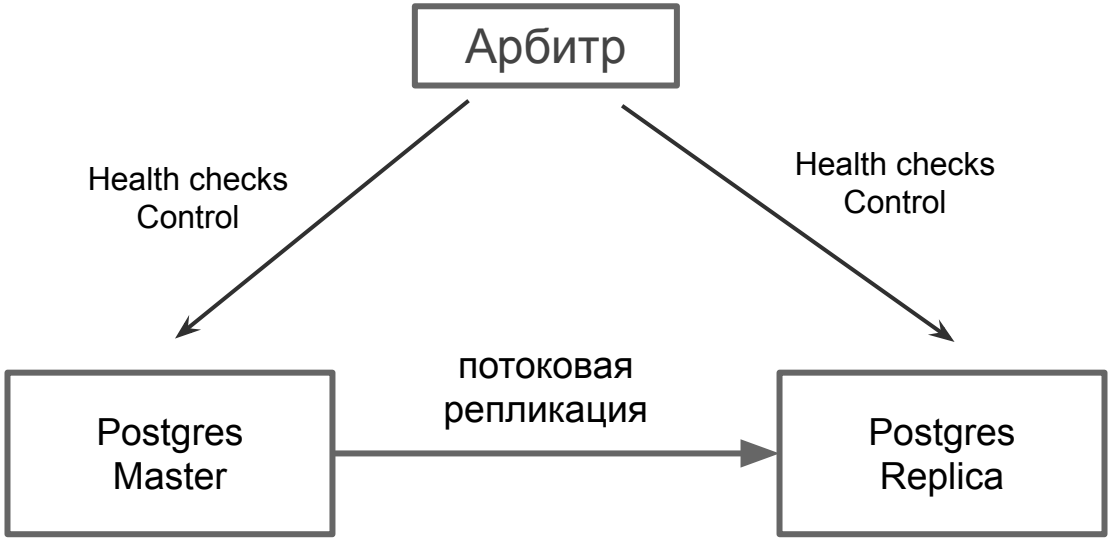
- Архитектура shared-nothing (без общего диска)
- Реплицируются физические изменения, описанные WAL
- Поддерживается режим чтения из реплики **hot standby**
- Готовые инструменты управления:
  - создание реплики - **pg_basebackup** от мастера, восстановление из стороннего бэкапа
  - **promote** нового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

Сложности построения кластера на логической репликации

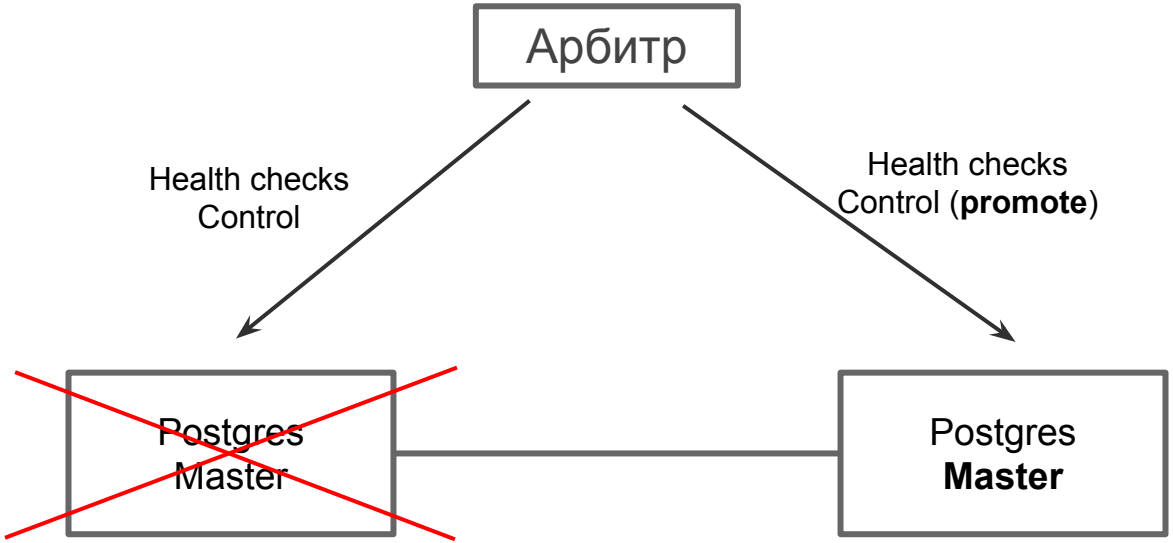
- Мотивация: снижение трафика репликации на высоконагруженной базе в распределенной конфигурации
  - кейс Убера[1]
- Ограниченные возможности
  - не реплицируются DDL, последовательности (sequence)
- Нет возможности перепозиционирования реплик на новый мастер
  - прогресс репликации завязан на позицию в WAL мастера - LSN (log sequence number)
  - нет аналога GTID как в MySQL, CSN как в MS SQL Server
- Нет аналога **pg_rewind** (легковесная ресинхронизация бывшего мастера)
- TODO в кластерных решениях Stolon/Patroni [2,3]
  - для обновления мажорной версии PostgreSQL без простоя кластера (rolling upgrade)

1. https://eng.uber.com/mysql-migration/
2. https://github.com/sorintlab/stolon/issues/519
3. https://github.com/zalando/patroni/issues/538

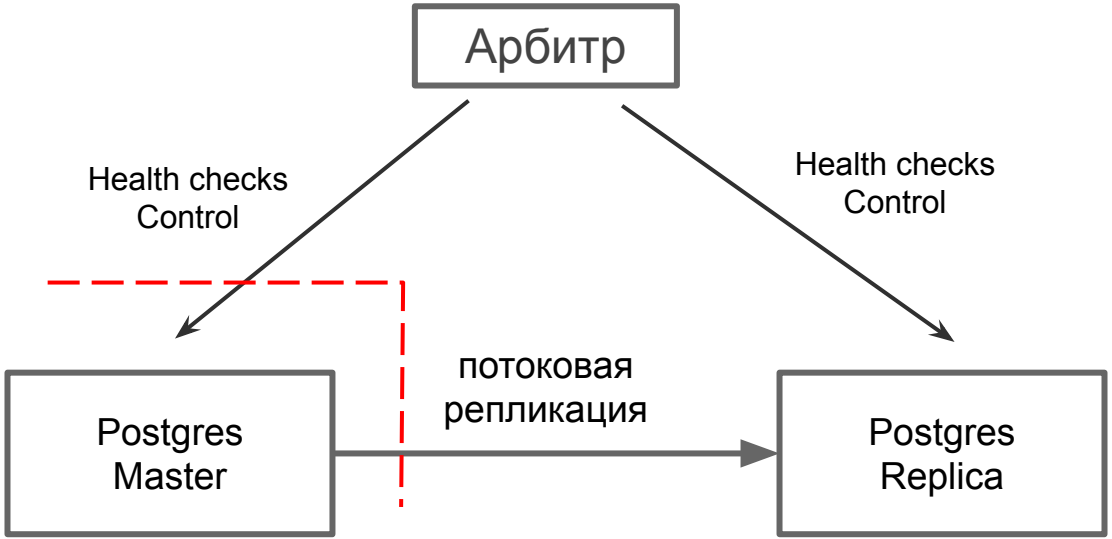
Примитивный автофейловер



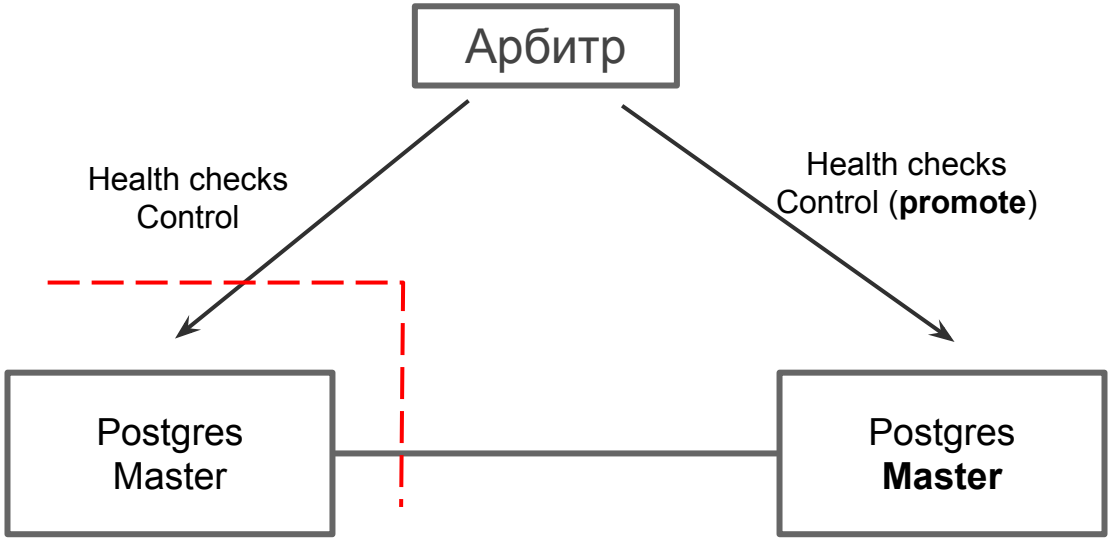
Примитивный автофейловер



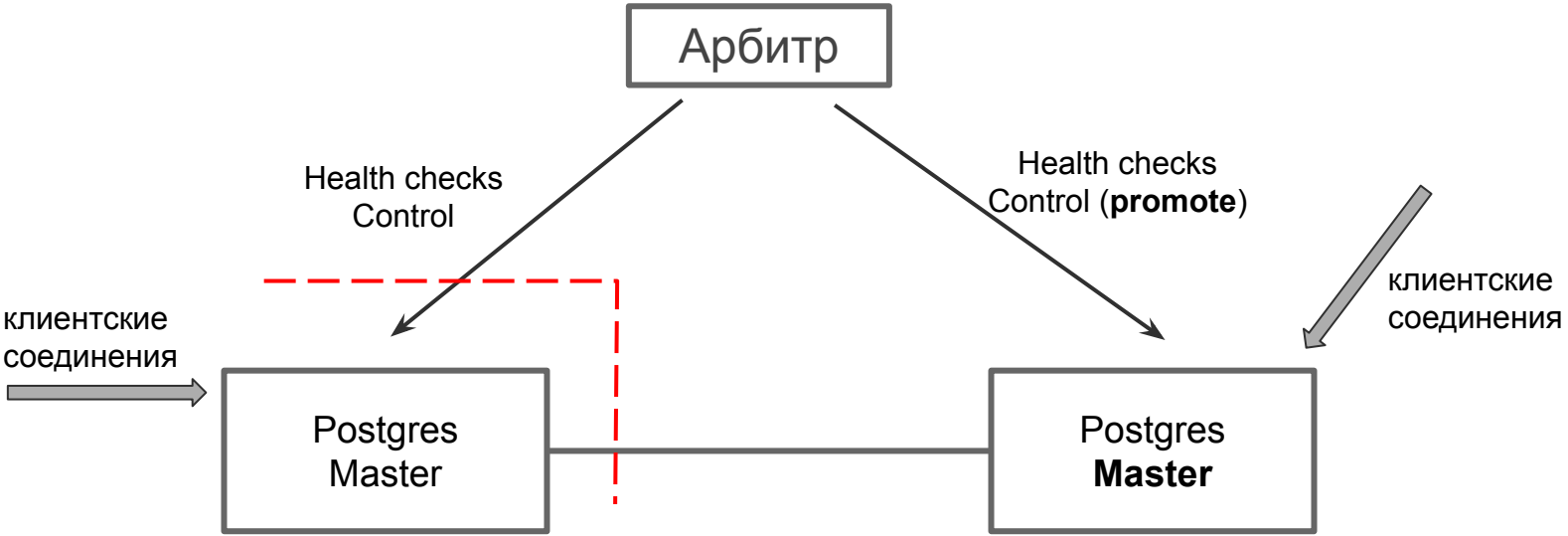
Примитивный автофейловер. Изоляция мастера



Примитивный автофейловер. Изоляция мастера

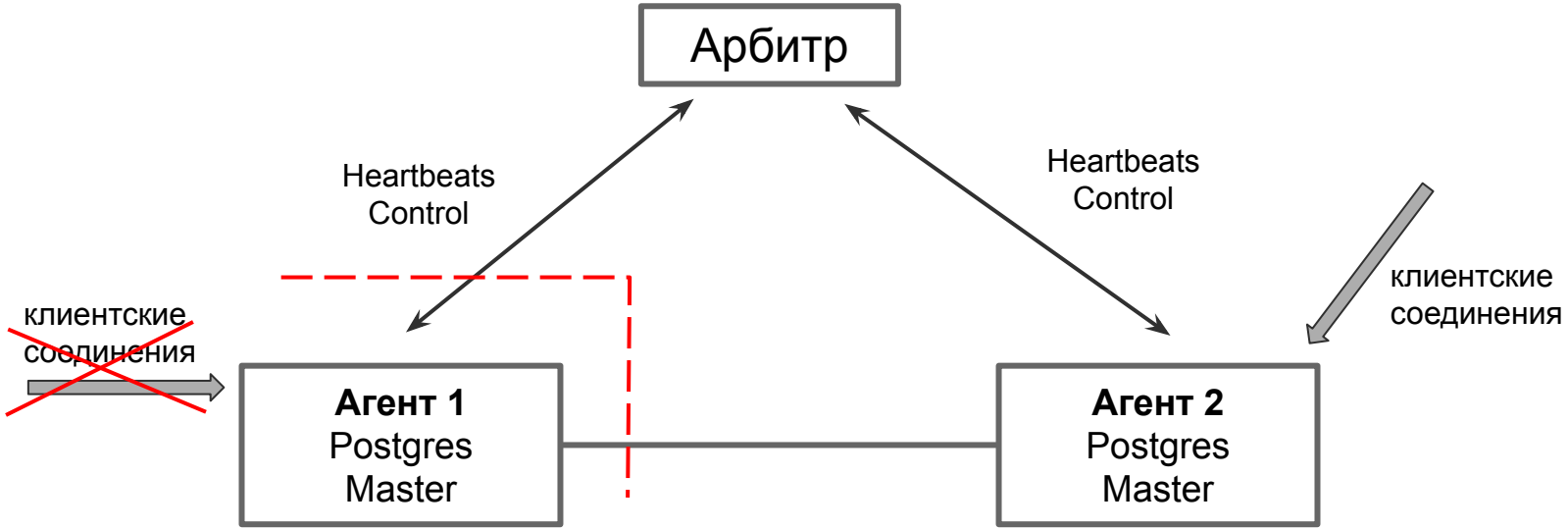


Примитивный автофейловер. Split brain



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

Примитивный автофейловер. Фенсинг изолированного мастера



Агенты:

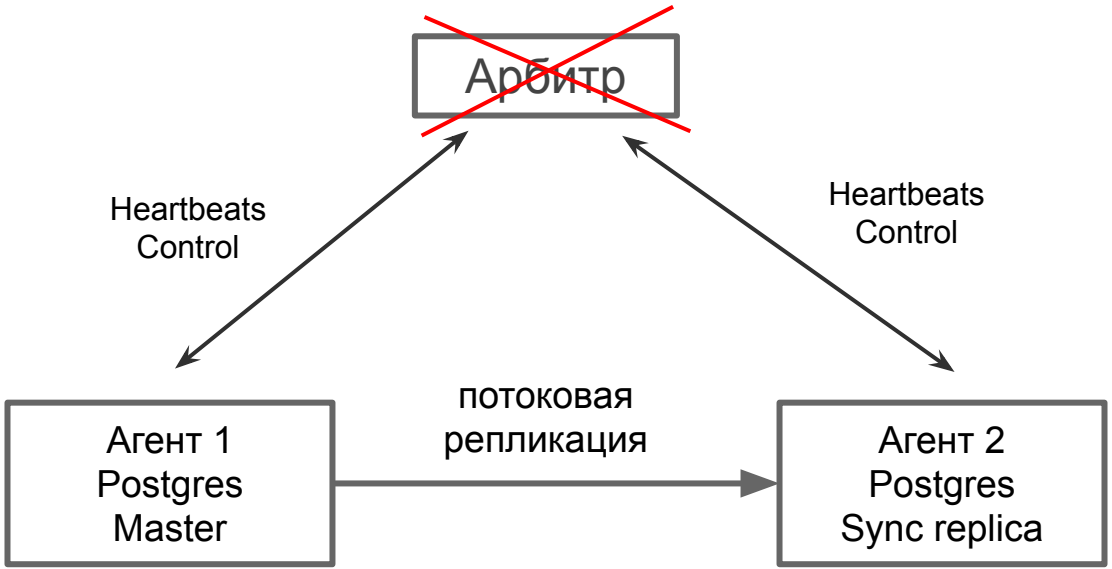
- управляют узлами PostgreSQL
- отправляют статус арбитру
- “**фенсят**” изолированный узел по таймауту

Изоляция неактуального мастера (fencing)

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

- выключение всего 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

Примитивный автофейловер. Доступность арбитра

- Рабочая схема в 3-х узловом кластере - два узла БД + арбитр
  - выдерживает падение одного узла
- Пример pg_auto_failover [1]
- Фенсинг мастера только если **не доступены арбитр && реплика** [2]
- Падение арбитра замораживает конфигурацию кластера

1. https://github.com/citusdata/pg_auto_failover
2. https://github.com/citusdata/pg_auto_failover/issues/12#issuecomment-490551255

Примитивный автофейловер. pg_auto_failover

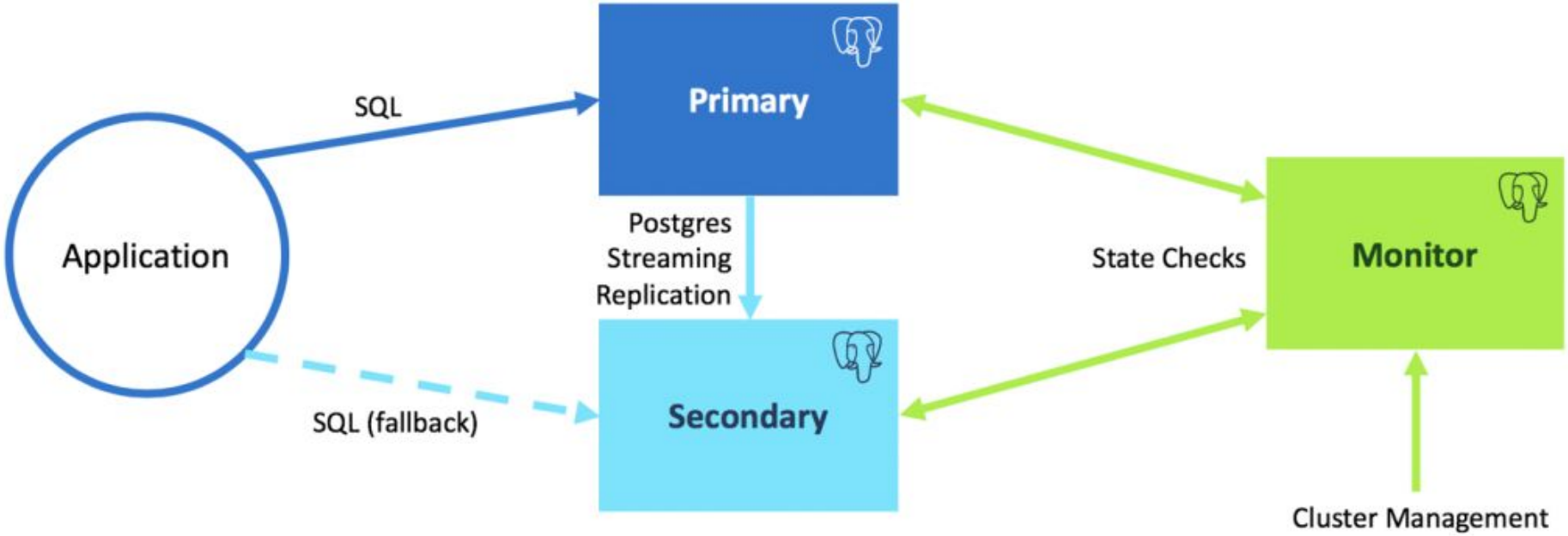


Схема HA кластера, осуществляющего автофейловер

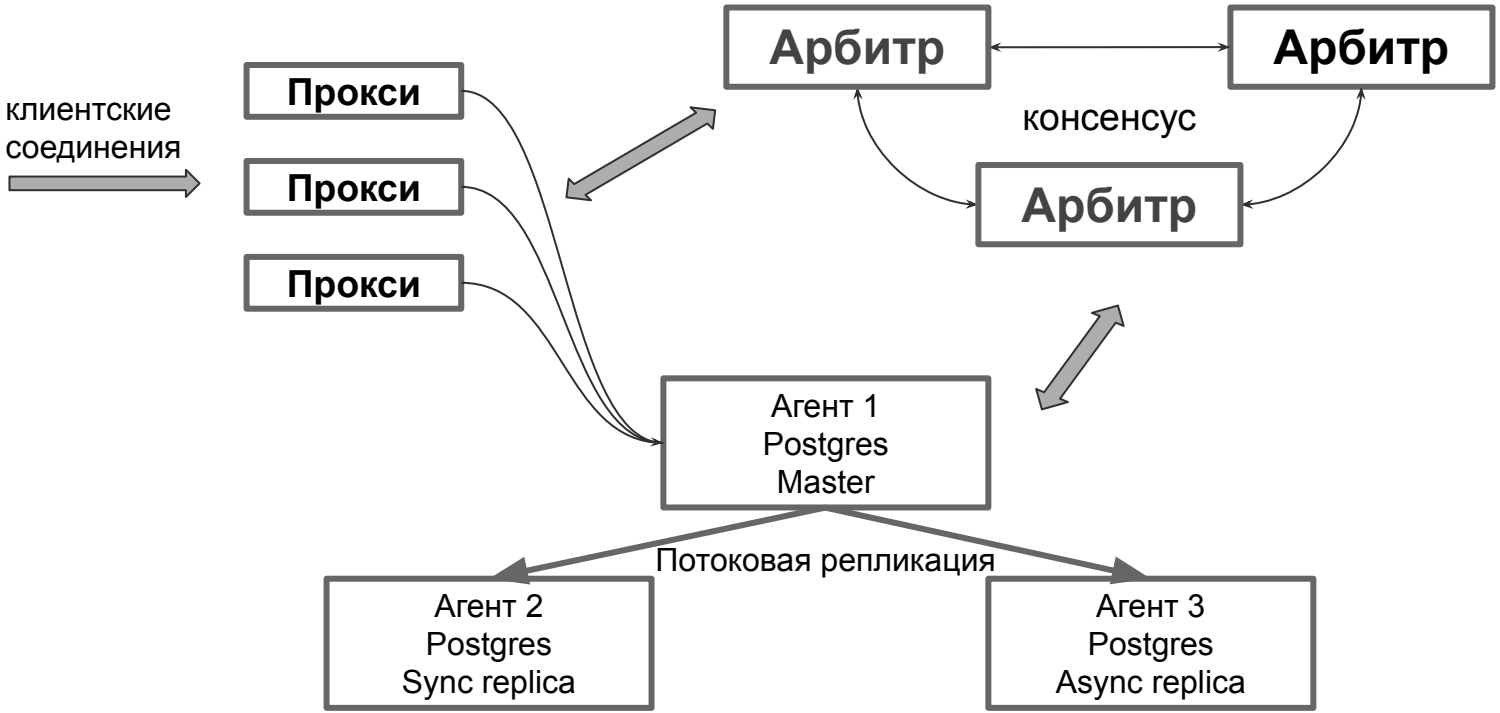
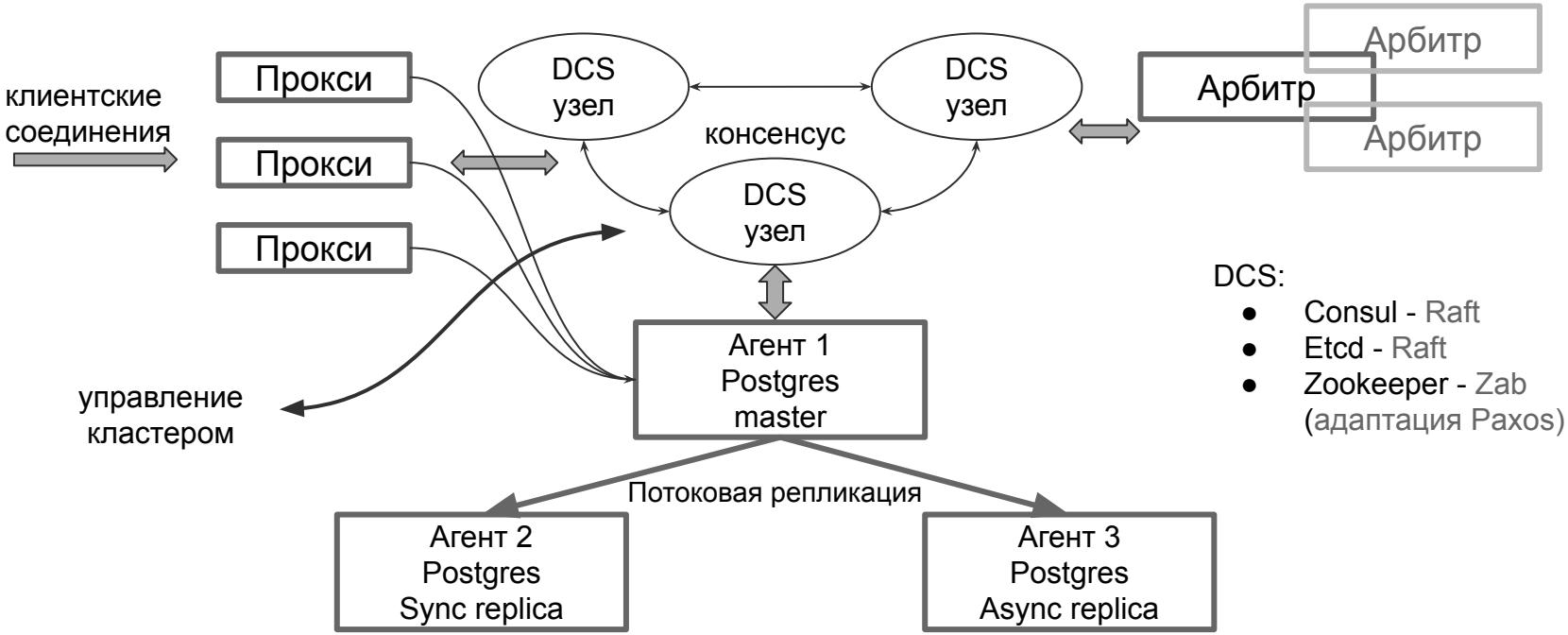


Схема HA кластера класса Patroni/Stolon



DCS - Distributed Configuration System (Распределённая система конфигурации)

HA кластер Patroni/Stolon

- Автоматическая реакция на инциденты в кластере (переключение роли мастера при недоступности текущего) - **autofailover**
- Автоматически настраиваемые узлы баз данных (агенты берут на себя всю рутину по настройке и управлению PostgreSQL) - **оркестратор** БД узлами
- Тесная интеграция с облачными решениями (kubernetes)
- Продукт класса DBaaS (database as a service)
- Мотивация:
  - узлов БД столько (100+), что инциденты - это больше правило, нежели исключение
  - жёсткие ограничения на RTO + желание спать по ночам :)

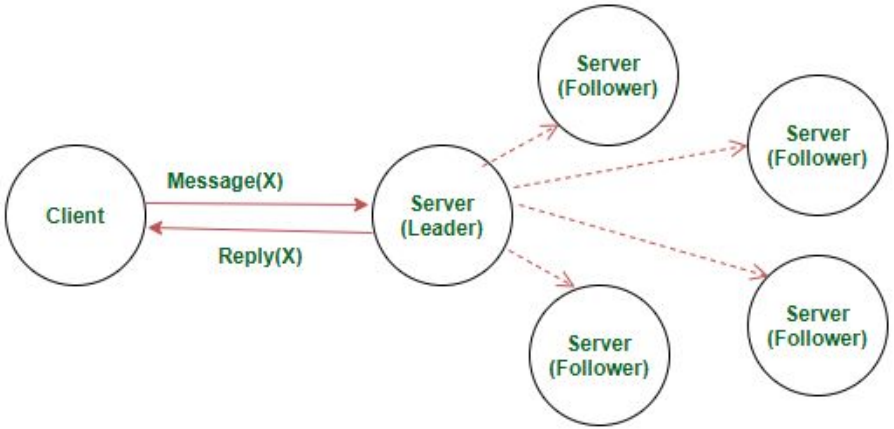
Распределенная система конфигурации (DCS) на примере Etcd

Распределённая система конфигурации (DCS) на примере etcd

- DCS - это **“единая” распределенная** точка отказа кластеров вокруг него
- Устойчивость кластера гарантируется работоспособностью большинством узлов кластера
- **минимум 3 серверных узла**, рекомендуется 5 узлов в 3-х зонах доступности
  - выдерживать одно падение при выводе одного узла на поддержку
- **RAFT** [1] как консенсус-протокол между узлами

1. https://raft.github.io/

Кластер Etcd



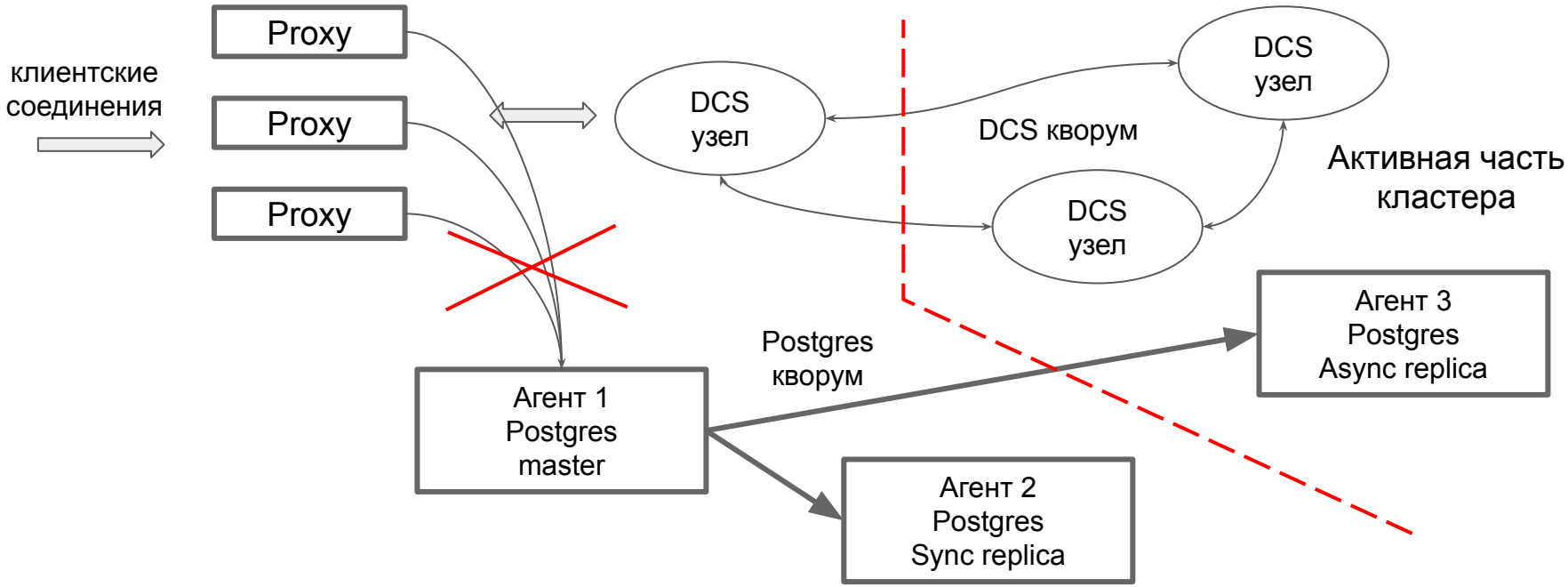
- Любая запись проходит через кворум узлов
  - как минимум, + (1-2 RTT + fsync) к обычному коммиту на одном узле
- Чтение либо из локального узла (stale read), либо с лидера, либо кворумное
- Временные характеристики:
  - ETCD_HEARTBEAT_INTERVAL - частота нотификаций от лидера к ведомым узлам (~ RTT между узлами)
  - ETCD_ELECTION_TIMEOUT - таймаут без нотификаций от лидера для ведомого узла, чтобы переключиться на выборы нового лидера (~ 10x RTT между узлами)

Распределённая система конфигурации (DCS) на примере etcd

- Нестабильность одного узла может деградировать весь кластер
  - каждый узел хранит состояние, является мини БД и требует стабильное (по времени) по записи/чтению хранилище
- Настройка ETCD_HEARTBEAT_INTERVAL / ETCD_ELECTION_TIMEOUT для геораспределенной (неустойчивой) конфигурации [1]
- Один кластер etcd способен поддерживать множество кластеров patroni/stolon [2]
- Необходимо отдельно поддерживать (бэкапы) / мониторить / настраивать

1. https://github.com/etcd-io/etcd/blob/master/Documentation/tuning.md#time-parameters
2. https://github.com/etcd-io/etcd/blob/master/Documentation/op-guide/hardware.md#example-hardware-configurations

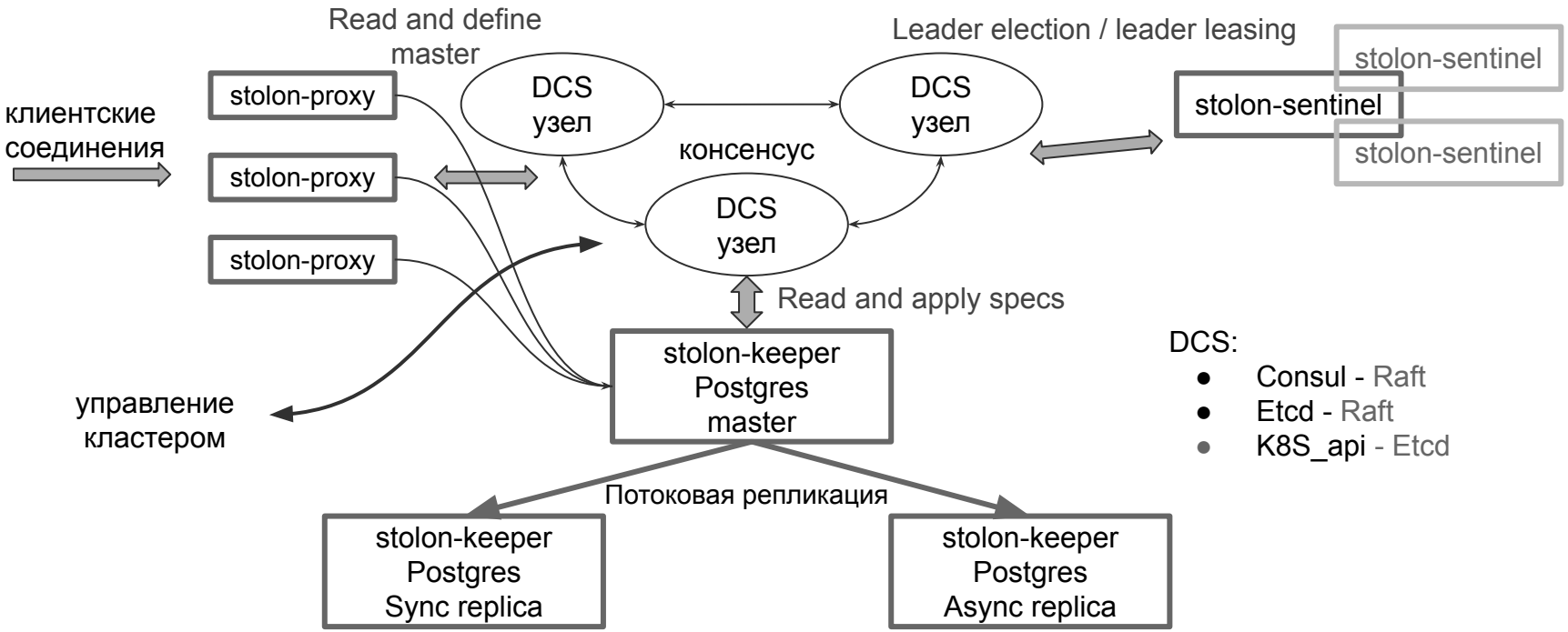
Привязка к кворуму DCS при фейловере



Stolon

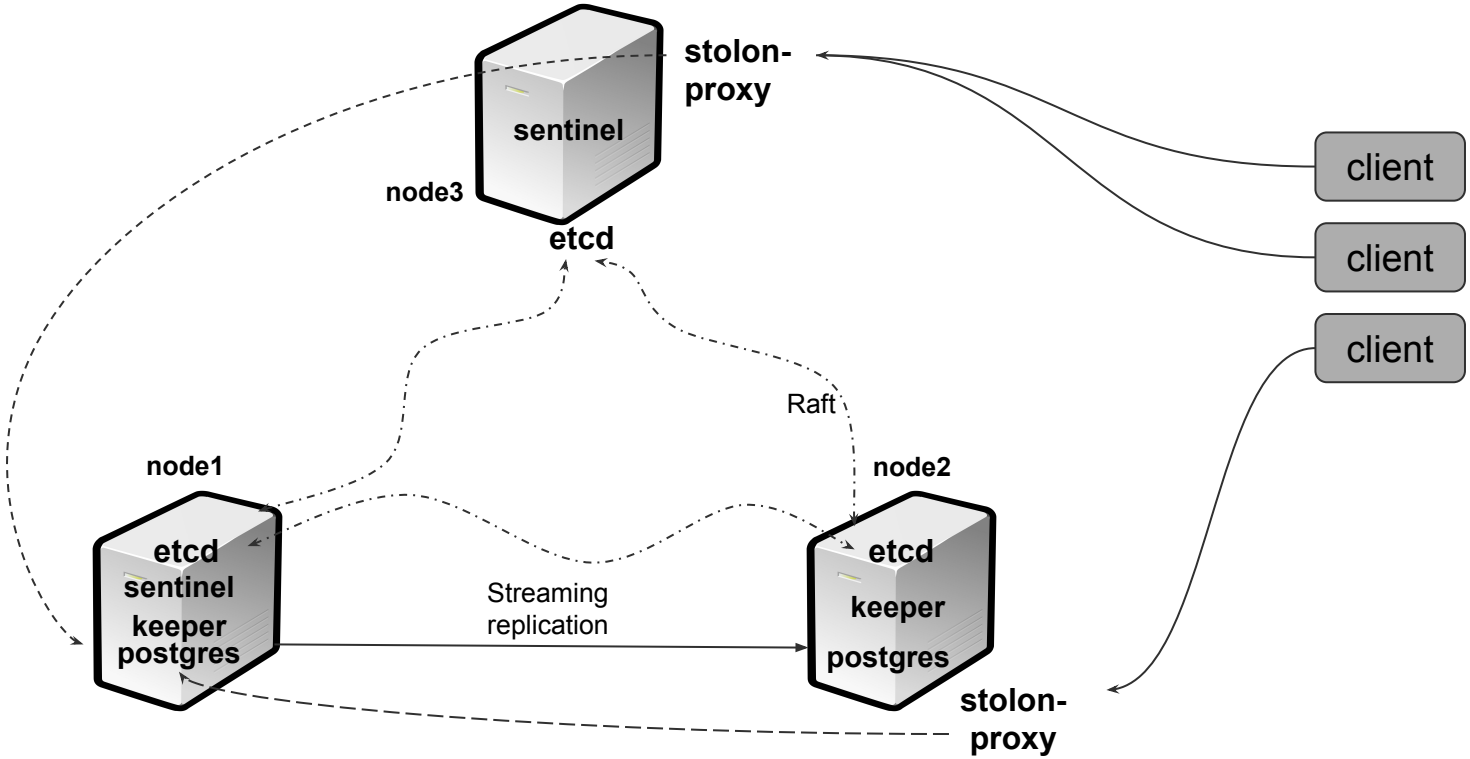
оркестратор PostgreSQL и автофейловер

Схема HA кластера Stolon



DCS - Distributed Configuration System (Распределённая система конфигурации)

Инсталляция Stolon на 3-х узлах



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

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: *kill -TERM/INT/QUIT/KILL!!! postmaster* [1] или рестарт агентов stolon-keeper
  - автоматически: параметр **automaticPgRestart** (не отрабатывает необходимый порядок рестарта узлов [2]!!!)
- Добавление/удаление узлов кластера происходит автоматически при запуске/останове **stolon-keeper**

1. https://postgrespro.ru/docs/postgrespro/12/server-shutdown
2. https://github.com/sorintlab/stolon/issues/707

Инициализация кластера Stolon

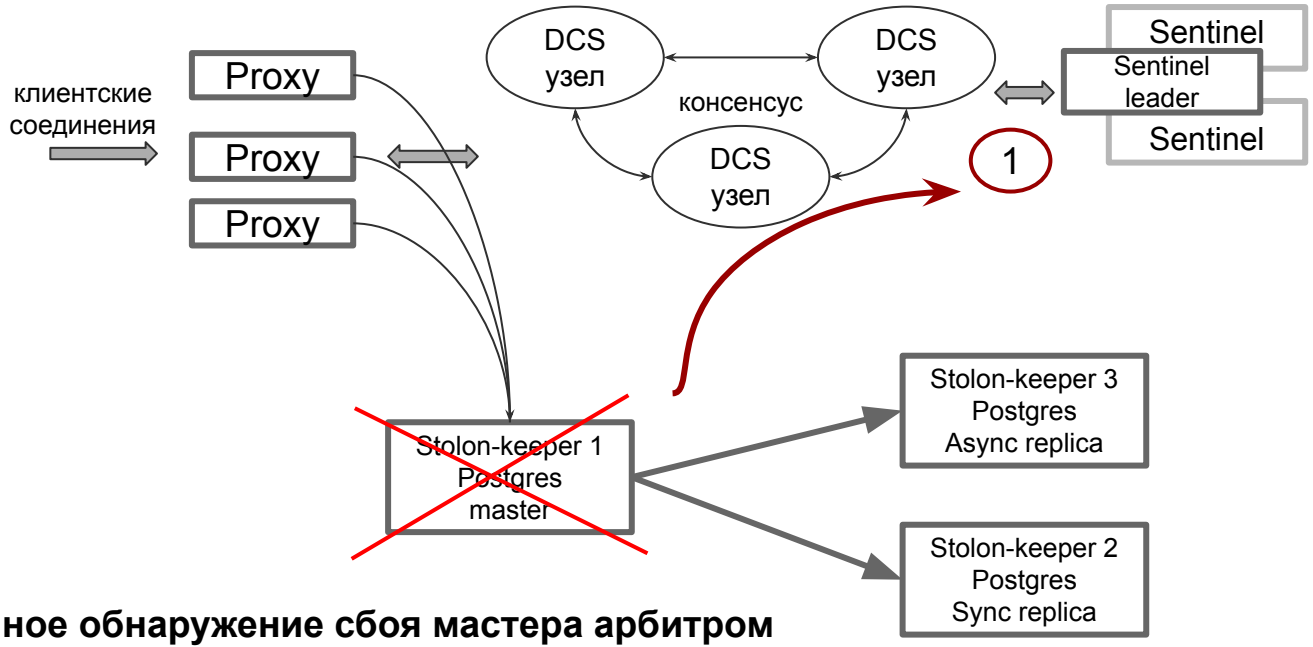
- **new** - “с нуля”
- **PITR** - из бэкапа
  - Standby Cluster [2]
- **existing**
  - восстановление кластера при потере данных в DCS
  - навязать мастер роль выбранному узлу

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

Временные характеристики при фейловере

- **sleepInterval** - период срабатывания для всех компонентов (по умолчанию, 5сек)
- **requestTimeout** - deadline операций к PostgreSQL (по умолчанию, 10сек)
  - Deadline операций к DCS - 5сек
- **failInterval** - таймаут подтверждения падения узла stolon-keeper для stolon-sentinel (по умолчанию, 20сек)

Failover. Stolon

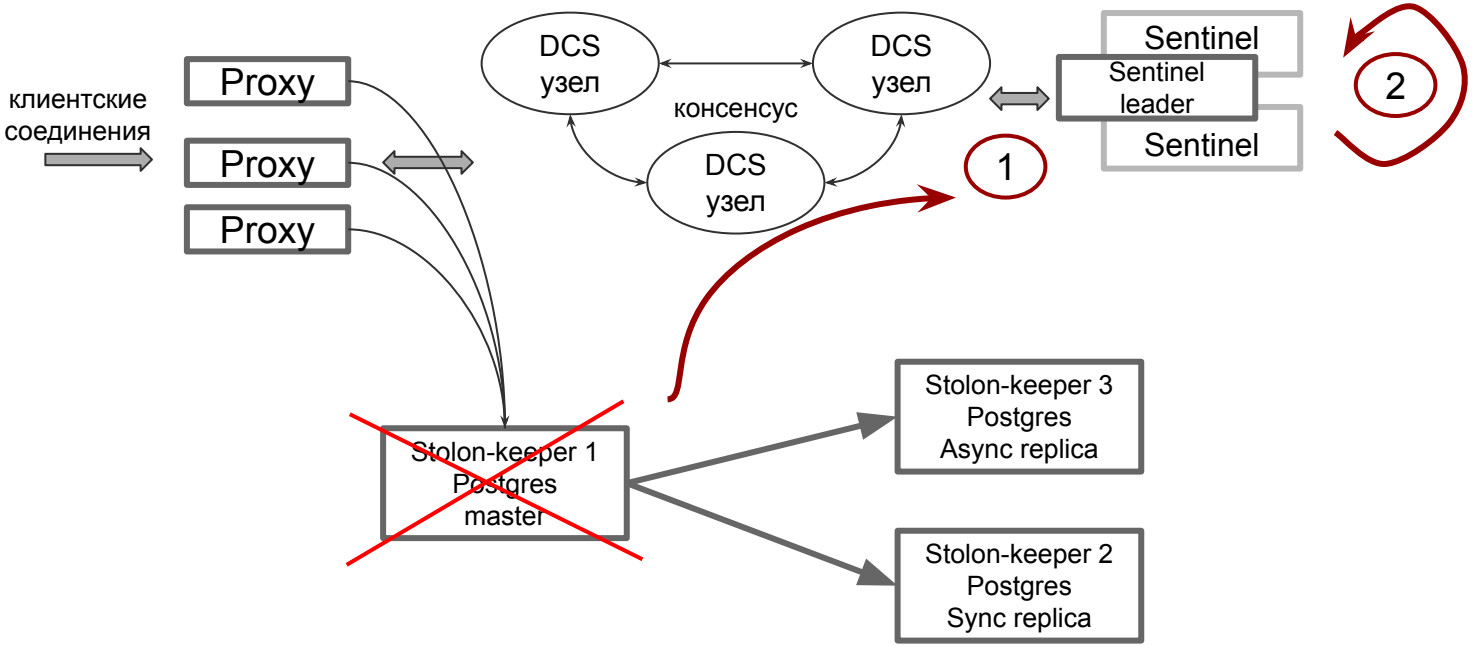


1. **Первичное обнаружение сбоя мастера арбитром**

   $(\lambda_1 + \lambda_2) * sleepInterval$

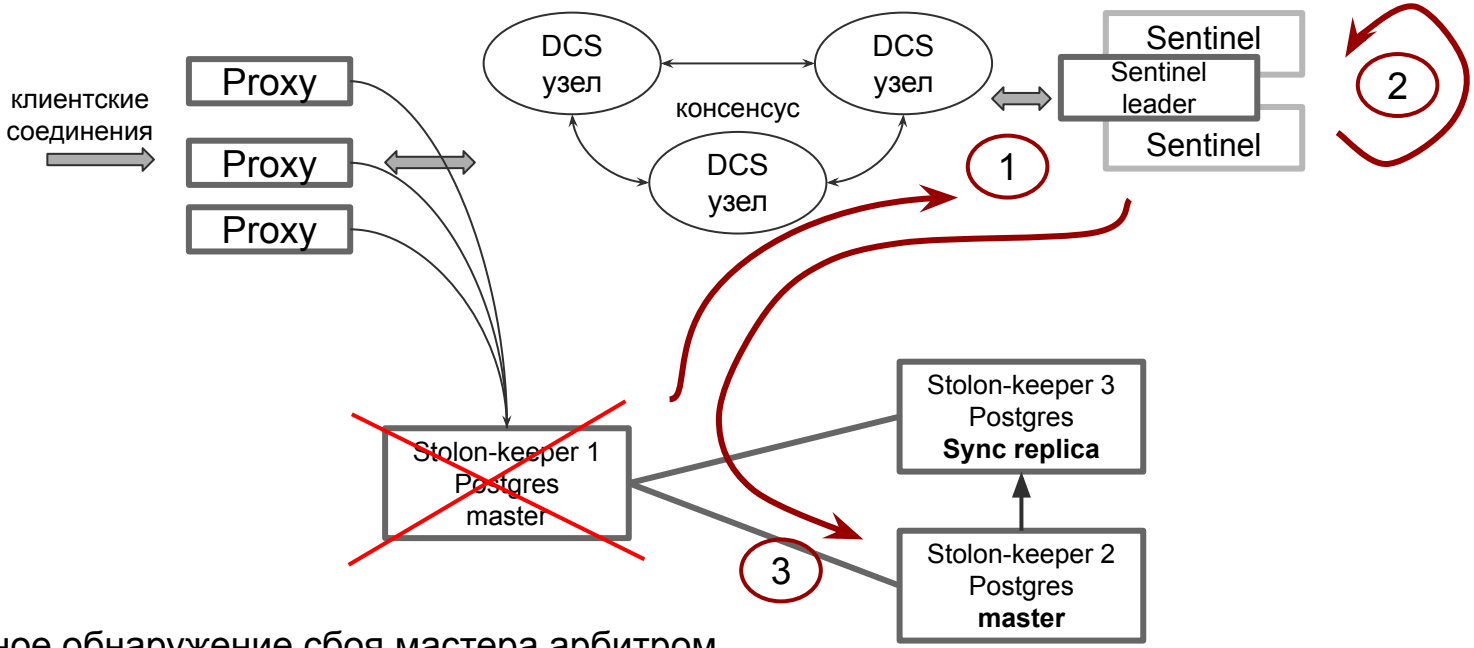
   - λ_i - случайная величина от 0 до 1
   - *sleepInterval* - период срабатывания каждого компонента (keeper, sentinel, proxy) в цикле (по умолчанию, 5 сек)

Failover. Stolon



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

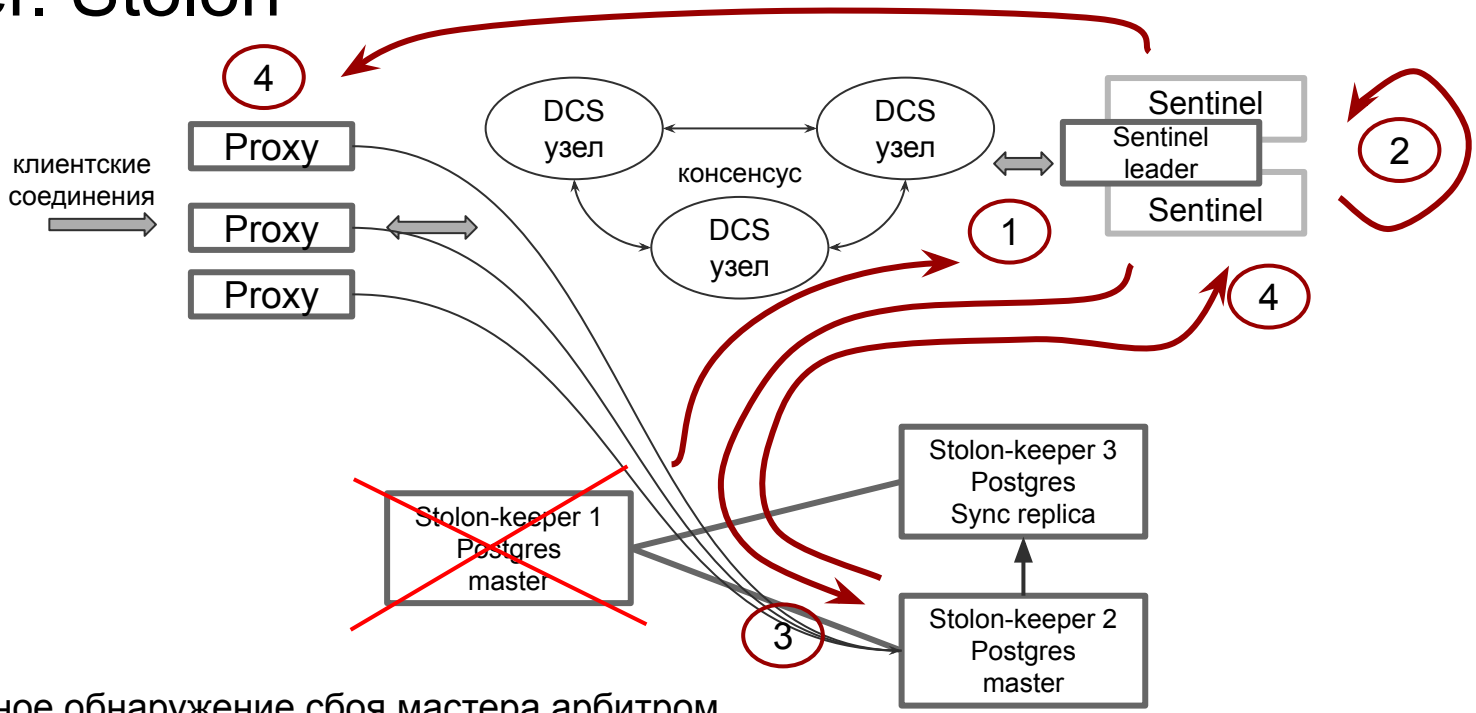
Failover. Stolon



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

   $(\lambda_1 + \lambda_2) * sleepInterval$

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**

   *(1 + λ) * sleepInterval*

Failover. Stolon

- *Время недоступности кластера =*

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

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

  $(1 + \lambda_5)$ * *sleepInterval*

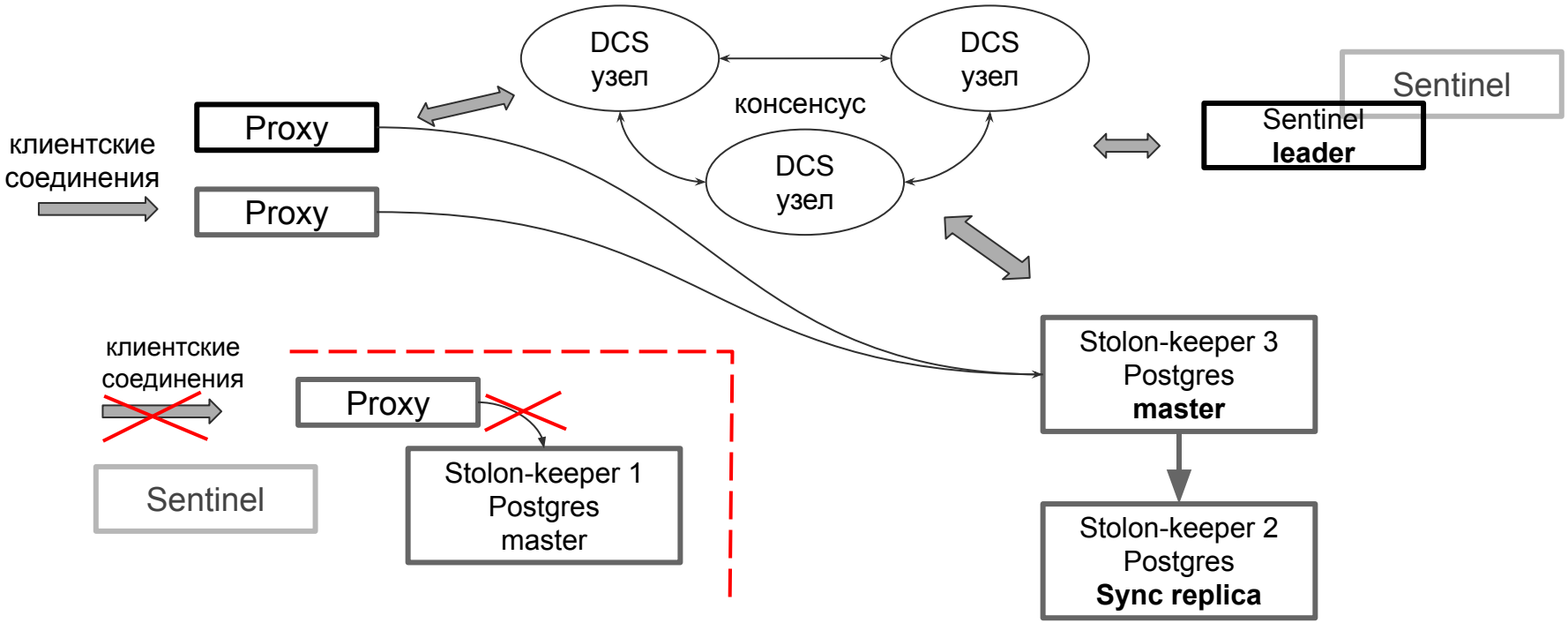
  где:

  - 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 25 до 50 сек.**

Фенсинг изолированного мастера и stolon-proxy



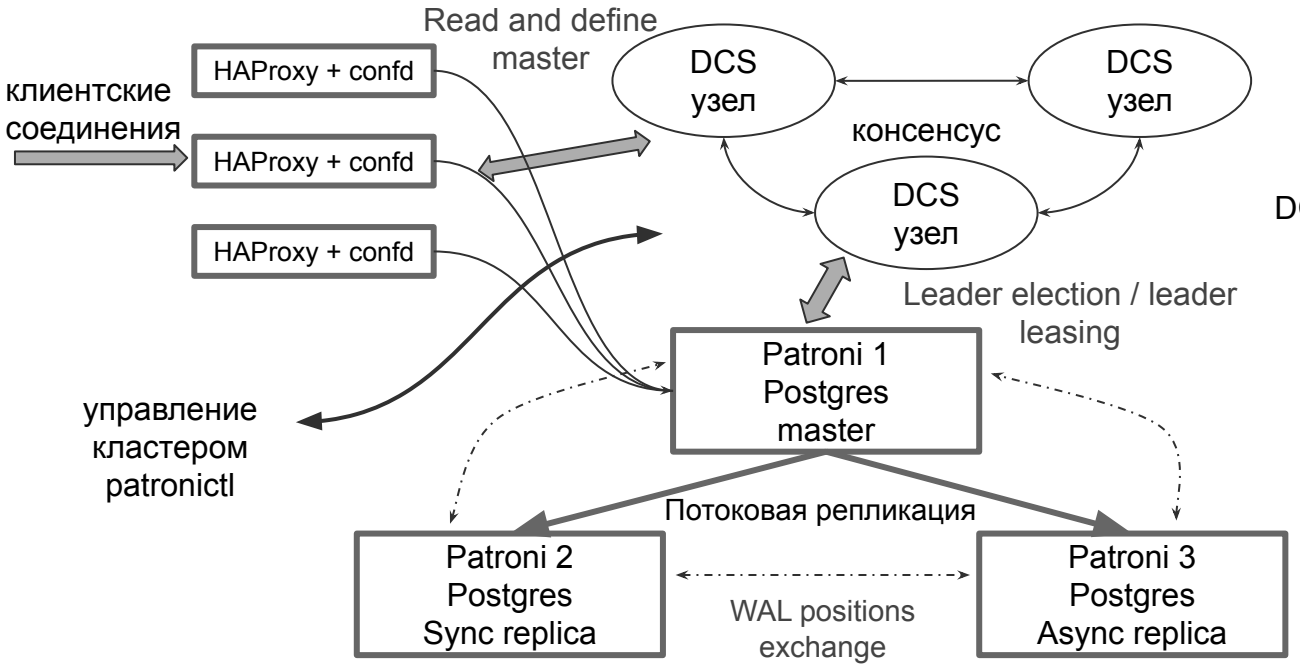
Фейловер **stolon-sentinel** лидера за **20 сек**
Stolon-proxy фенсится за **15 сек**

Завершающие штрихи

- Узел после падения удаляется из кластера (удаляется его слот репликации) по истечении **deadKeeperRemovalInterval** (по умолчанию, 48 часов)
  - удаление вручную через **stolonctl removekeeper …**
  - настройка **wal_keep_segments** всё равно необходима при switchover/failover
  - Stolon не поддерживает восстановление из архива (до PG12+)
- **dbWaitReadyTimeout** - deadline на восстановление при старте базы (по умолчанию, 60 сек.)
- **syncTimeout** (скрытый параметр) - deadline на восстановление базы при PITR (по умолчанию, 30 мин., с новой версии - отключен)
- **initTimeout** (скрытый параметр) - deadline на инициализацию базы (initdb, по умолчанию, 5мин.)

Patroni
оркестратор PostgreSQL и автофейловер

Схема HA кластера Patroni



DCS:

- Consul - Raft
- Etcd - Raft
- Zookeeper - Zab (адаптация Paxos)
- K8S_api - Etcd
- etc.

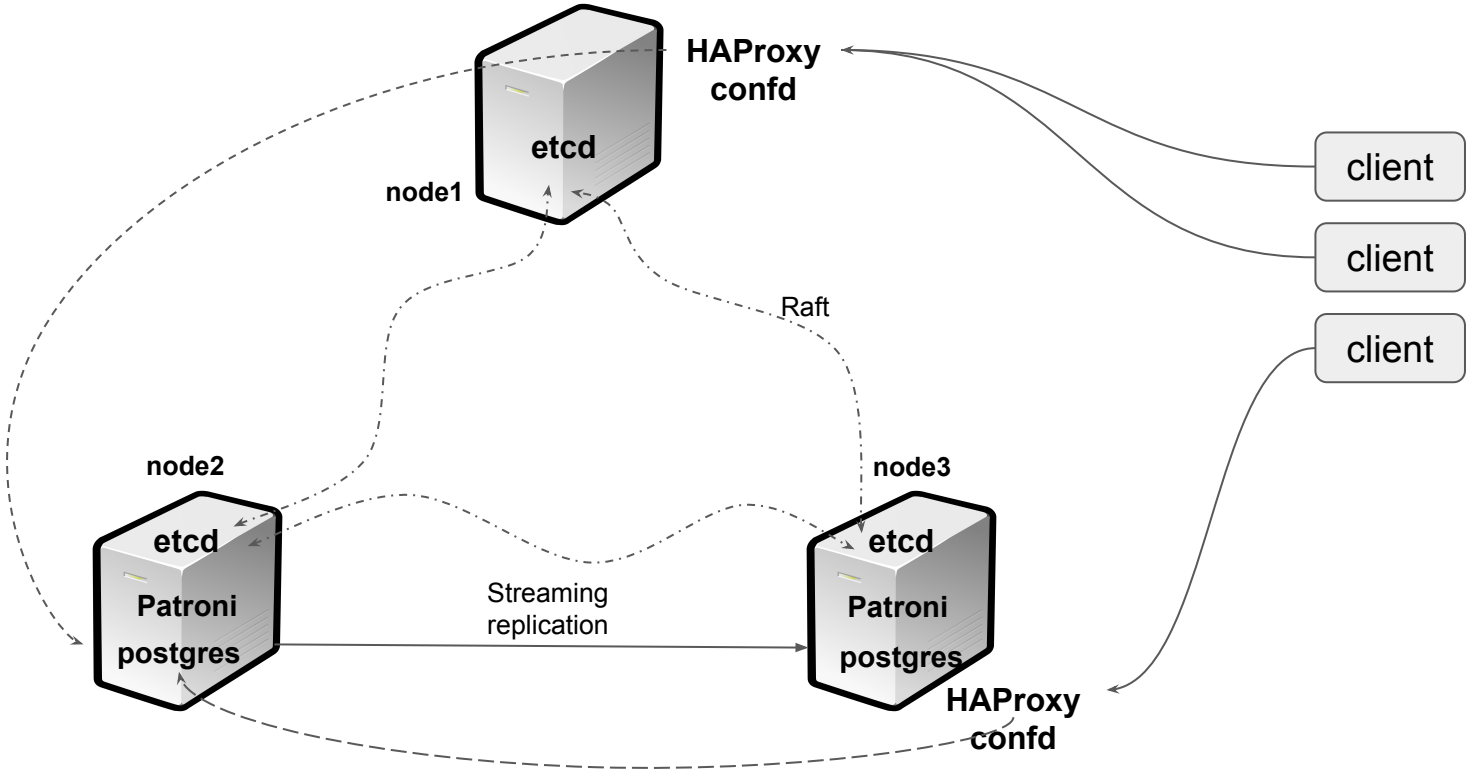
DCS - Distributed Configuration System (Распределённая система конфигурации)

Архитектура Patroni

- **Patroni bot (postgres-агент)**
  - управляет экземпляром PostgreSQL
  - связаны друг с другом по REST API
- **Арбитр кластера**
  - DCS
    - поддержка высоко доступного и **линеаризованного** хранилища конфигурации
    - атомарный выбор лидера и поддержка TTL ключа лидера (leader election and leasing)
  - Patroni агенты. По REST API:
    - участвуют в выборе нового мастера
    - подтверждают падение мастер узла
- **Прокси (балансировщик)**

  сторонний настраиваемый компонент

Инсталляция Patroni на 3-х узлах



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]
- Восстановление реплик **кастомизируется**

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

Workflow работы с кластером Patroni

- Инициализация кластера и баз данных при запуске patroni агентов
  - конфигурация кластера (секция *bootstrap*) в файле конфигурации агента (можно в единственном экземпляре)
- Изменение параметров кластера (в т.ч. **postgresql.conf / pg_hba.conf / recovery.conf**):
  - **patronictl edit-config …** - глобально для кластера
  - секция **postgresql** в файле конфигурации - локально для узла
- Перезагрузка PostgreSQL:
  - **patronictl restart ...**
  - **patronictl restart --role …** соблюдая очерёдность мастер/реплики [1]
- Добавление/удаление узлов кластера происходит автоматически при запуске/останове **patroni**

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

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 + haproxy/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://github.com/zalando/patroni/blob/master/haproxy.cfg
2. https://github.com/zalando/patroni/tree/master/extras/confd

Прокси в Patroni (master discovery)

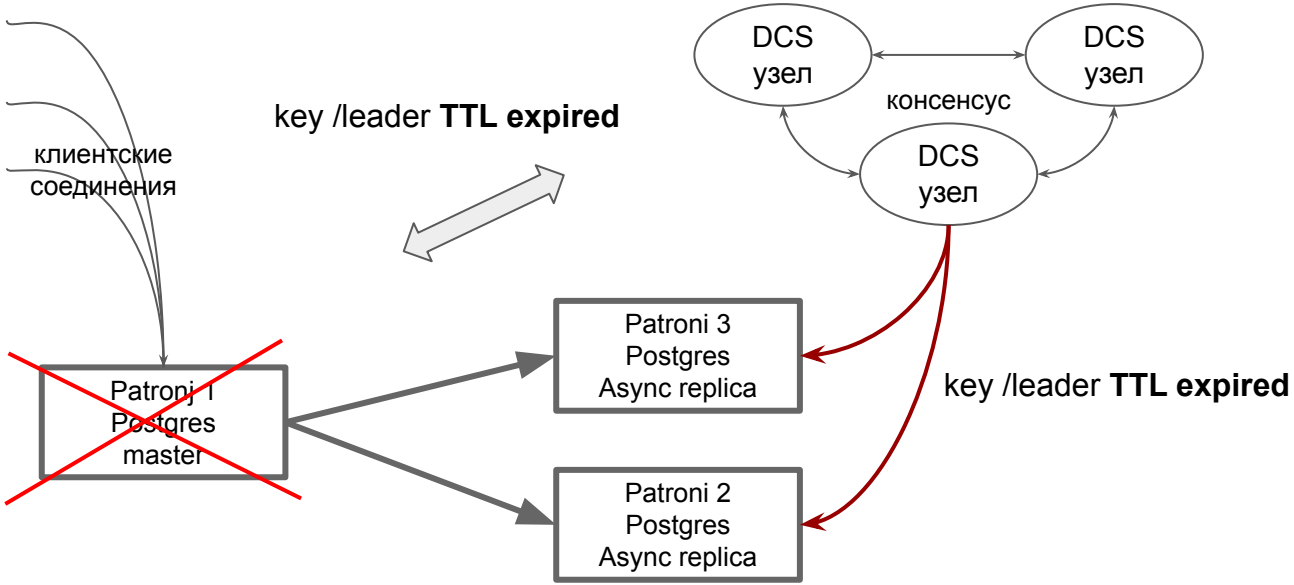
- Через DNS сервер consul (service discovery)
  - требует проброс DNS в локальное окружение клиента [1] и настройки кэширования [2]
  - пример использования в проде [3]
- Единый IP-адрес для мастер узла
  - HAProxy + keepalived
  - vip-manager [4]

1. https://www.consul.io/docs/guides/forwarding.html
2. https://learn.hashicorp.com/consul/day-2-operations/advanced-operations/dns-caching
3. https://pgconf.ru/2019/242817 https://pgconf.ru/2019/242821
4. https://github.com/cybertec-postgresql/vip-manager

Временные характеристики при автофейловере

- Параметры, влияющие на срабатывание failover’а:
  - *ttl* - время жизни ключа лидера
  - *loop_wait* - период срабатывания патрони-агента
  - *retry_timeout* - максимальная длительность операций к DCS и PostgreSQL
  - *master_start_timeout* - максимальная длительность восстановления после падения PostgreSQL экземпляра (не патрони агента)
- *ttl* ≥ *loop_wait + 2*retry_timeout*, чтобы исключить фейковые переключения мастер роли

Failover. Patroni

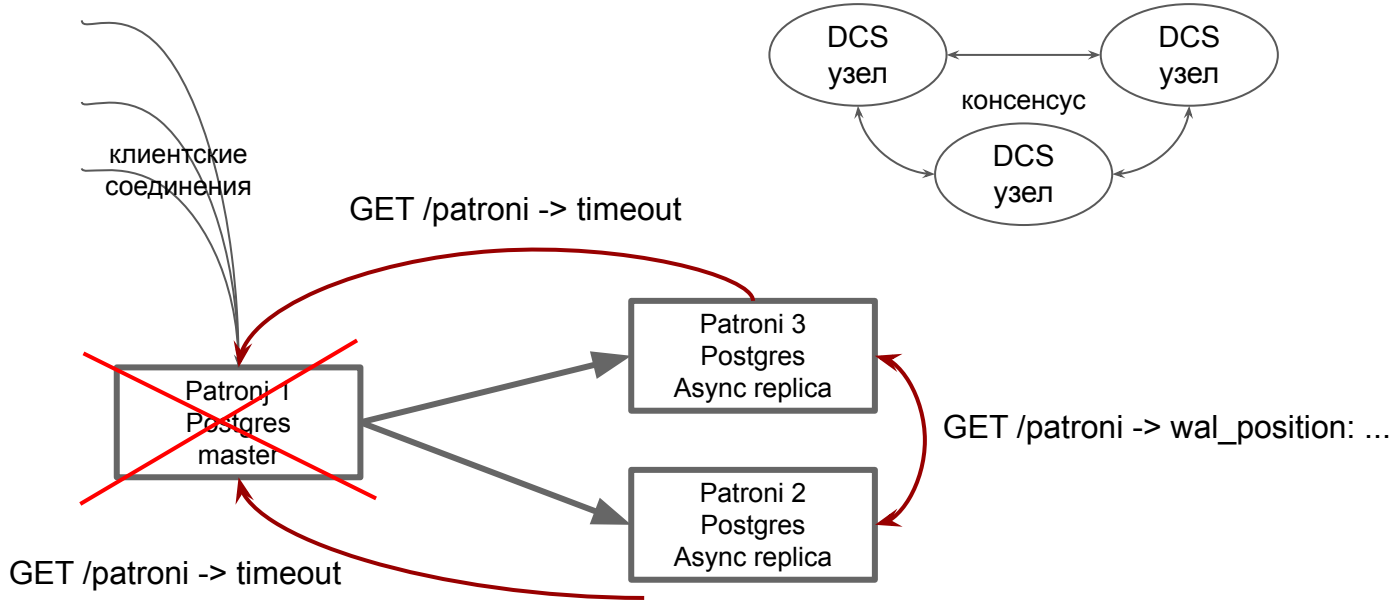


1. **Выявление сбоя мастера**

   *ttl - ƛ * loop_wait*, где
   ƛ - случайная величина от 0 до 1
   ttl - TTL ключа лидер в DCS
   loop_wait - период срабатывания patroni-агентов

20-30 сек. по умолчанию

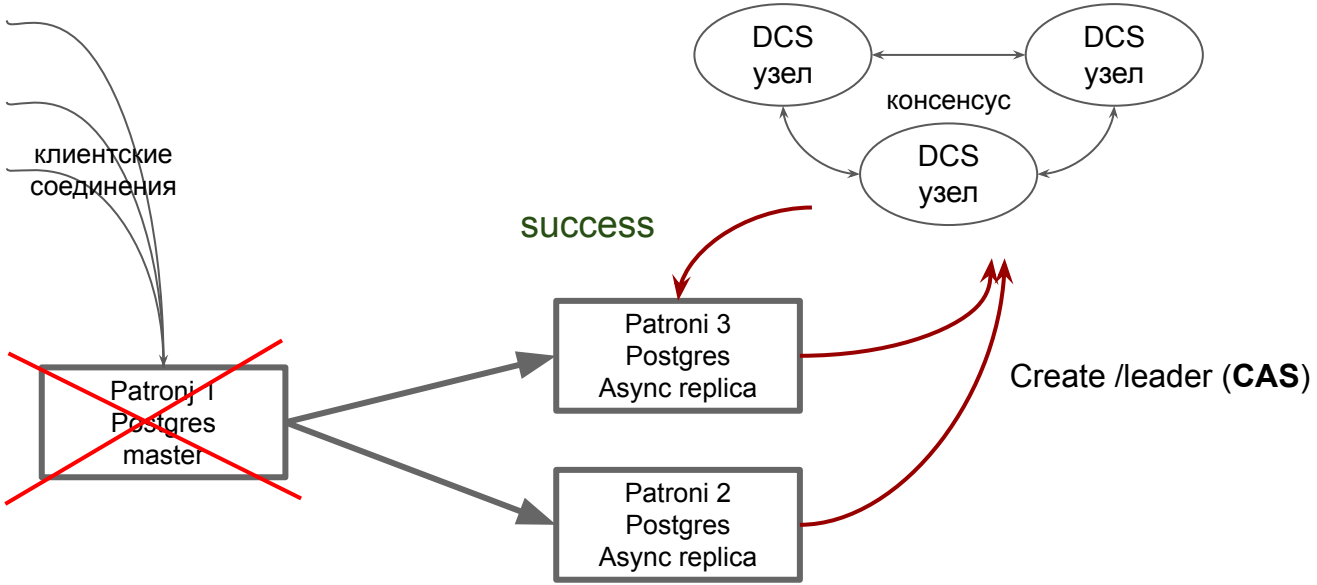
Failover. Patroni



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

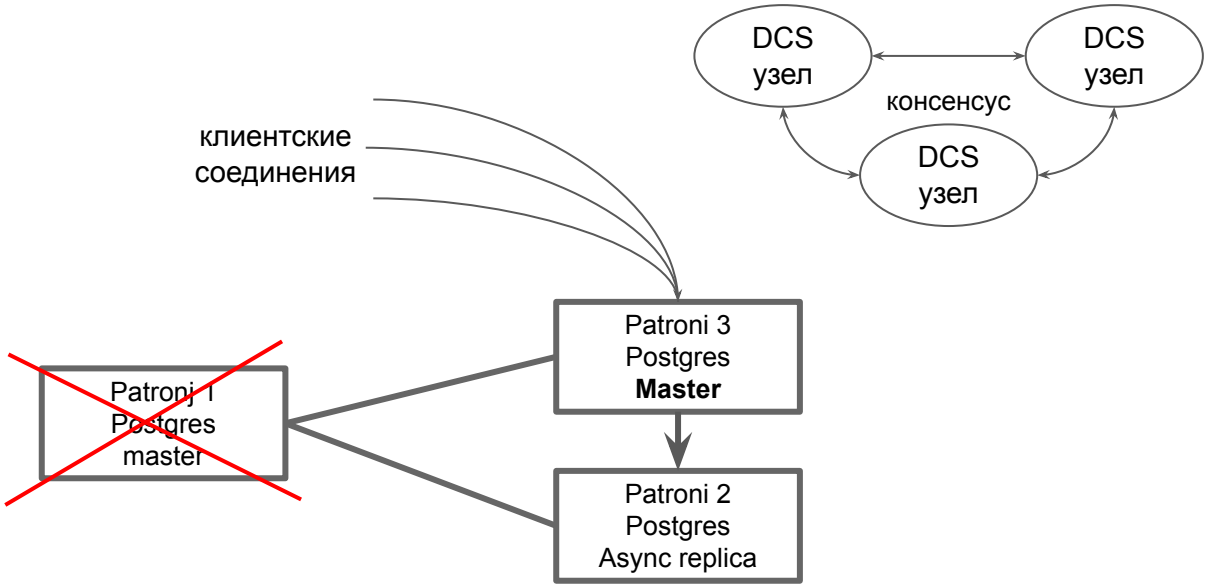
   2 сек. - timeout на GET /patroni

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 +*

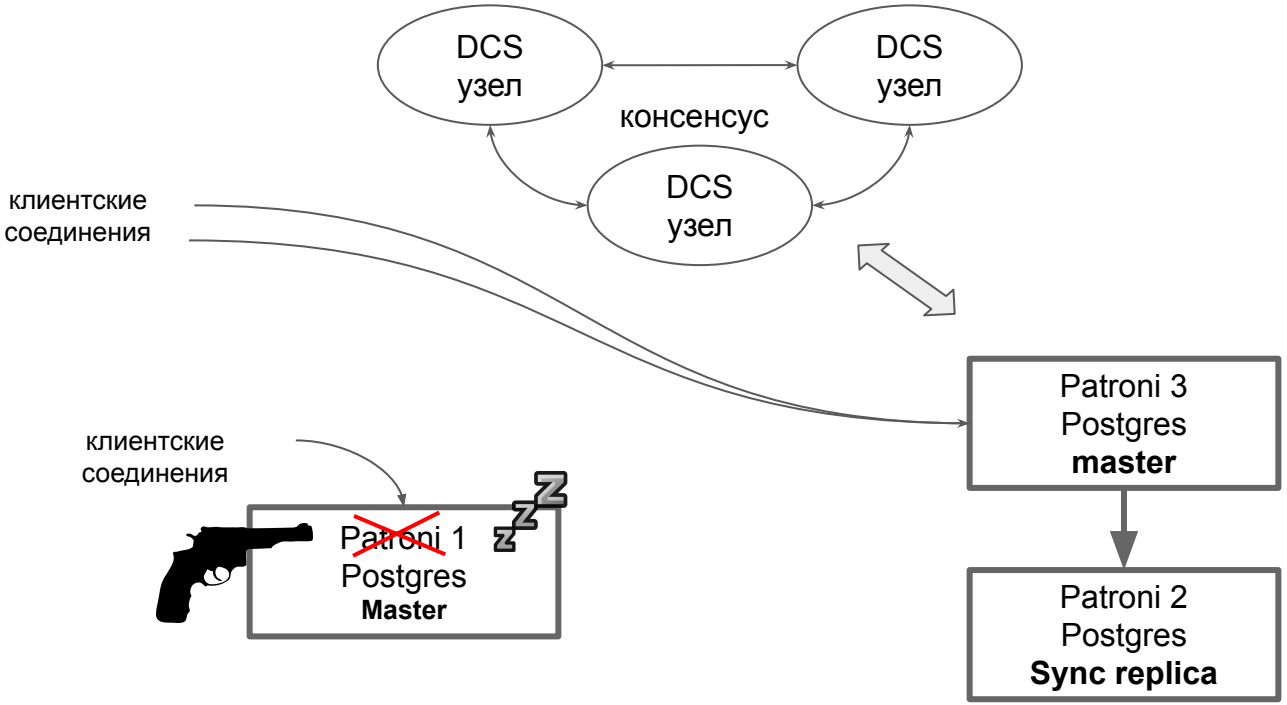
  2 +

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

Изоляция неактуального мастера (fencing)



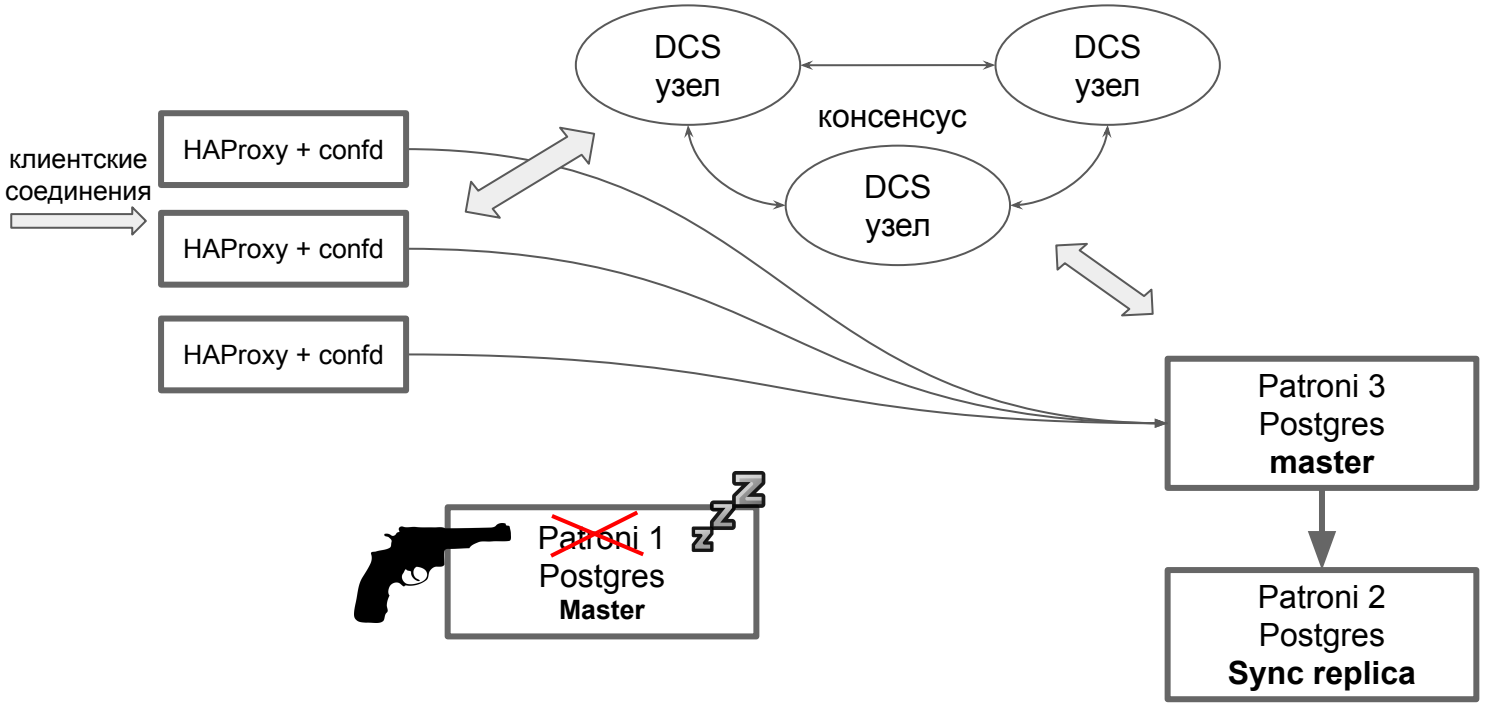
Split brain

Избегание split brain

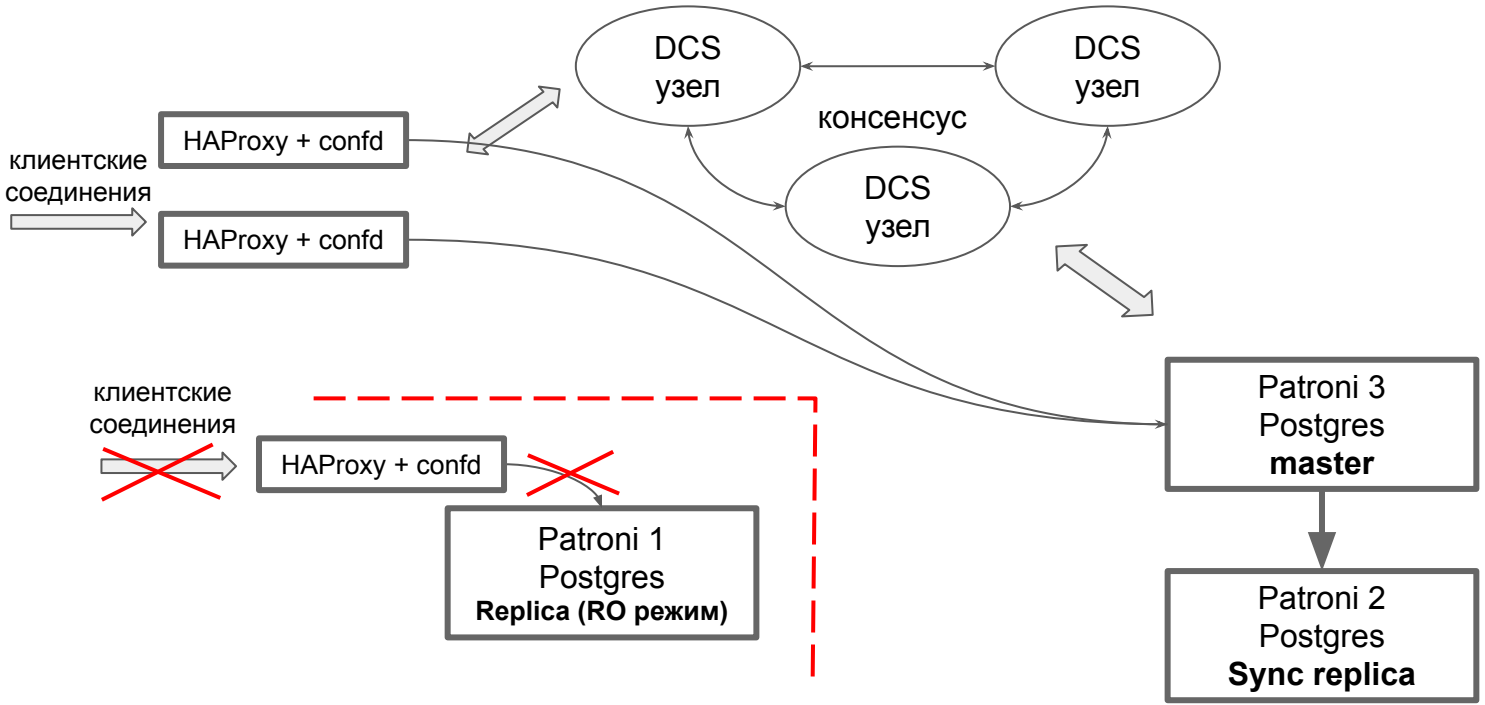
- Эксплуатация кластера в режиме синхронной репликации
  - любой коммит в бывший мастер зависает
  - однако прерывание запроса (Ctrl+C) / завершение бэкенда (Ctrl+D) / рестарт постгреса делает изменения видимыми [1]
- Привязка patroni-агента к watchdog`у (рекомендация от Zalando [2])
  - смущает реализация: `chown postgres /dev/watchdog`
- Правильно настроенный прокси (HAProxy + Confd)
  - перенаправление соединений при смене лидера **в DCS**
  - фенсинг при изоляции прокси от кластера
    - HAProxy http-check эмулирует такое поведение
- Явный фенсинг от нового мастера до promote`а (Corosync & Pacemaker)

1. https://www.postgresql.org/message-id/C1F7905E-5DB2-497D-ABCC-E14D4DEE506C@yandex-team.ru
2. https://github.com/zalando/patroni/blob/master/docs/watchdog.rst

Изоляция неактуального мастера (fencing)

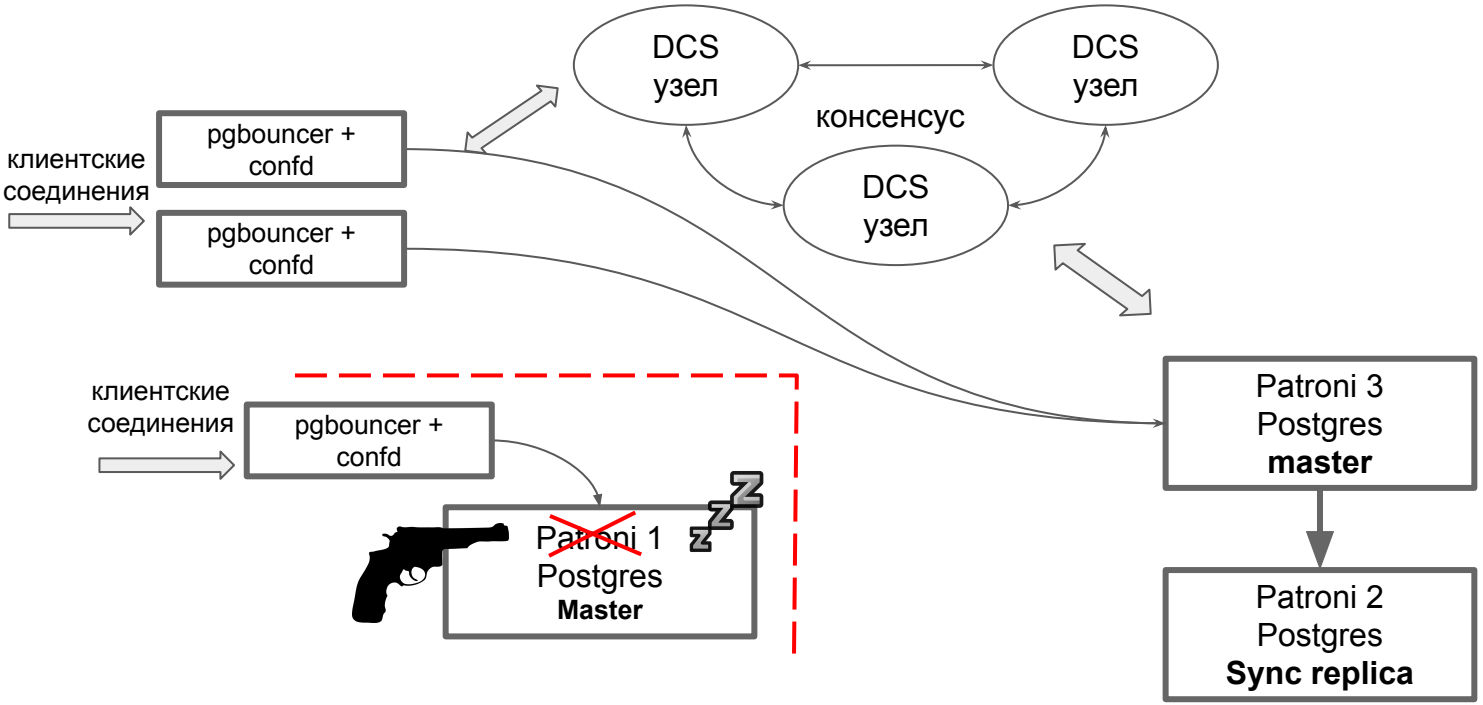


Изоляция неактуального мастера (fencing)



HAProxy фенсится за счёт **http-check** на patroni-агент
Confd замораживает конфигурацию HAProxy

Изоляция неактуального мастера (fencing)



Split brain

Завершающие штрихи

- Слоты репликации удаляются практически сразу после выхода узла из строя
  - реплика после длительного простоя может не встать в кластер
  - необходимо настраивать **wal_keep_segments**, либо восстанавливаться из архива
- При синхронной репликации лишь **одна** реплика по факту синхронная
  - кластер PostgreSQL образует кворум из 2-ух узлов
- Patroni по сравнению со Stolon имеет наиболее продвинутые фичи для enterprise баз данных:
  - кастомизируемые политики восстановления реплик из бэкапа
  - “нестрогий” синхронный режим (аналог Max Availability в Oracle Data Guard)
  - более гибкие настройки узла PostgreSQL

Спасибо за внимание!

Максим Милютин milyutinma@gmail.com