Яндекс

Яндекс Облако

Базы данных в облаках

Владимир Бородин, руководитель службы
сервисов хранения данных Яндекс.Облака

Немного истории

2012

У каждого сервиса:

- Изолированные железные сервера
- Своя эксплуатация
- Свои велосипеды
  - Для деплоя и оркестрации сервисов
  - Для хранения данных

2016

- Общий object storage
- Общий map-reduce

- Три облака для запуска stateless-приложений
- Всё ещё ничего для хранения метаданных

2016

- habr.com/post/321756/
- К нам стали приходить разработчики и просить поднять базу
- Решили делать PostgreSQL as a Service

Только PostgreSQL

- Много экспертизы
- Унификация технологий

Задачи сервиса

- Масштабирование
- Обновления
- Резервные копии
- Отказоустойчивость
- Мониторинг
- Возможность управления и разумной настройки

Первая попытка

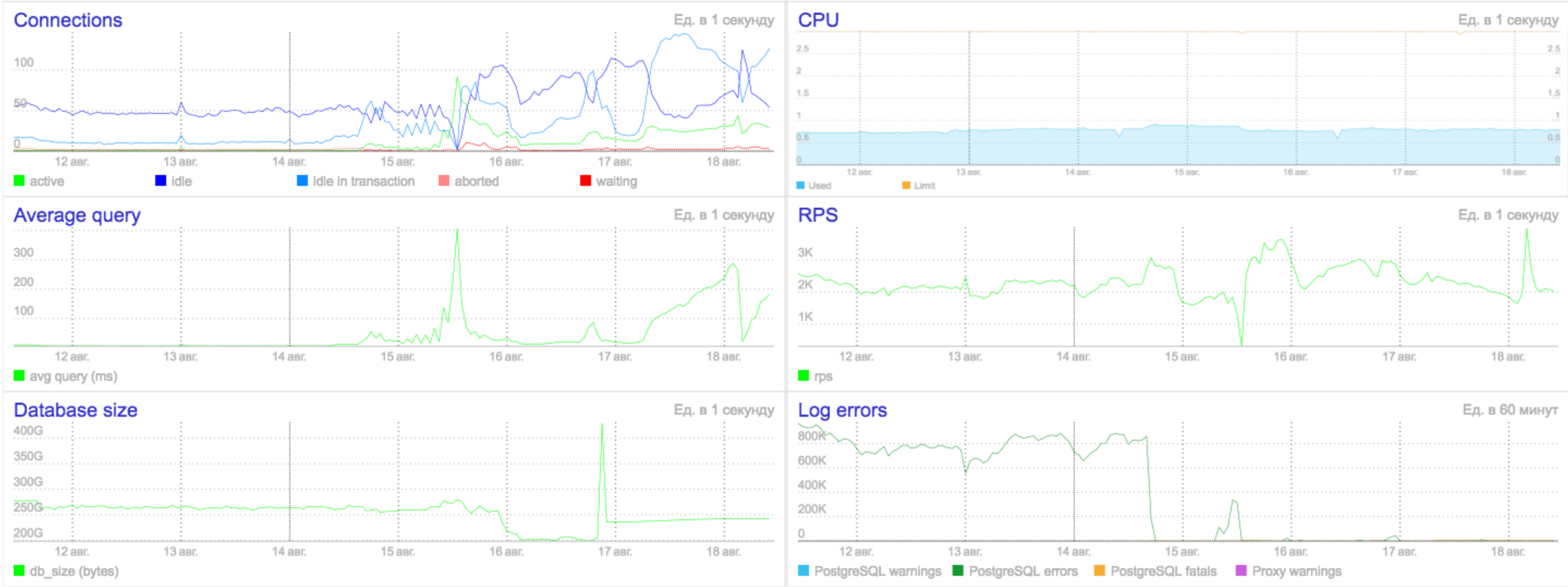
Пока PGaaS ещё не готов, поднимете мне мааленькую базку?

Разработчик

Коммуналки

- Несколько баз в одном кластере PostgreSQL
- Hard limit на количество соединений
- Soft limit по занимаемому месту
- Процессы одной БД в отдельной cgroup’е

Коммуналки


Connections
Ед. в 1 секунду
100
50
0
12 авг.
13 авг.
14 авг.
15 авг.
16 авг.
17 авг.
18 авг.
active
idle
idle in transaction
aborted
waiting
CPU
Ед. в 1 секунду
2.5
2
1.5
1
0.5
0
Used
Limit
Average query
Ед. в 1 секунду
300
200
100
avg query (ms)
RPS
Ед. в 1 секунду
3K
2K
1K
rps
Database size
Ед. в 1 секунду
400G
350G
300G
250G
200G
db_size (bytes)
Log errors
Ед. в 60 минут
800K
600K
400K
200K
0
PostgreSQL warnings
PostgreSQL errors
PostgreSQL fatals
Proxy warnings

Нет ничего более постоянного чем временное

Альберт Джей Нок

Коммуналки

Отсутствие нормальной изоляции:

- Восстановление из бэкапа только для всех БД кластера
- Нет способа запереть по месту одну базу, не создав проблем остальным
- Некоторые подсистемы PostgreSQL общие на весь кластер
  - Запись в WAL
  - Subtransactions
- Cgroup может прижимать процессы в горячих кусках кода
- Один потребитель может съесть все TCP-сессии

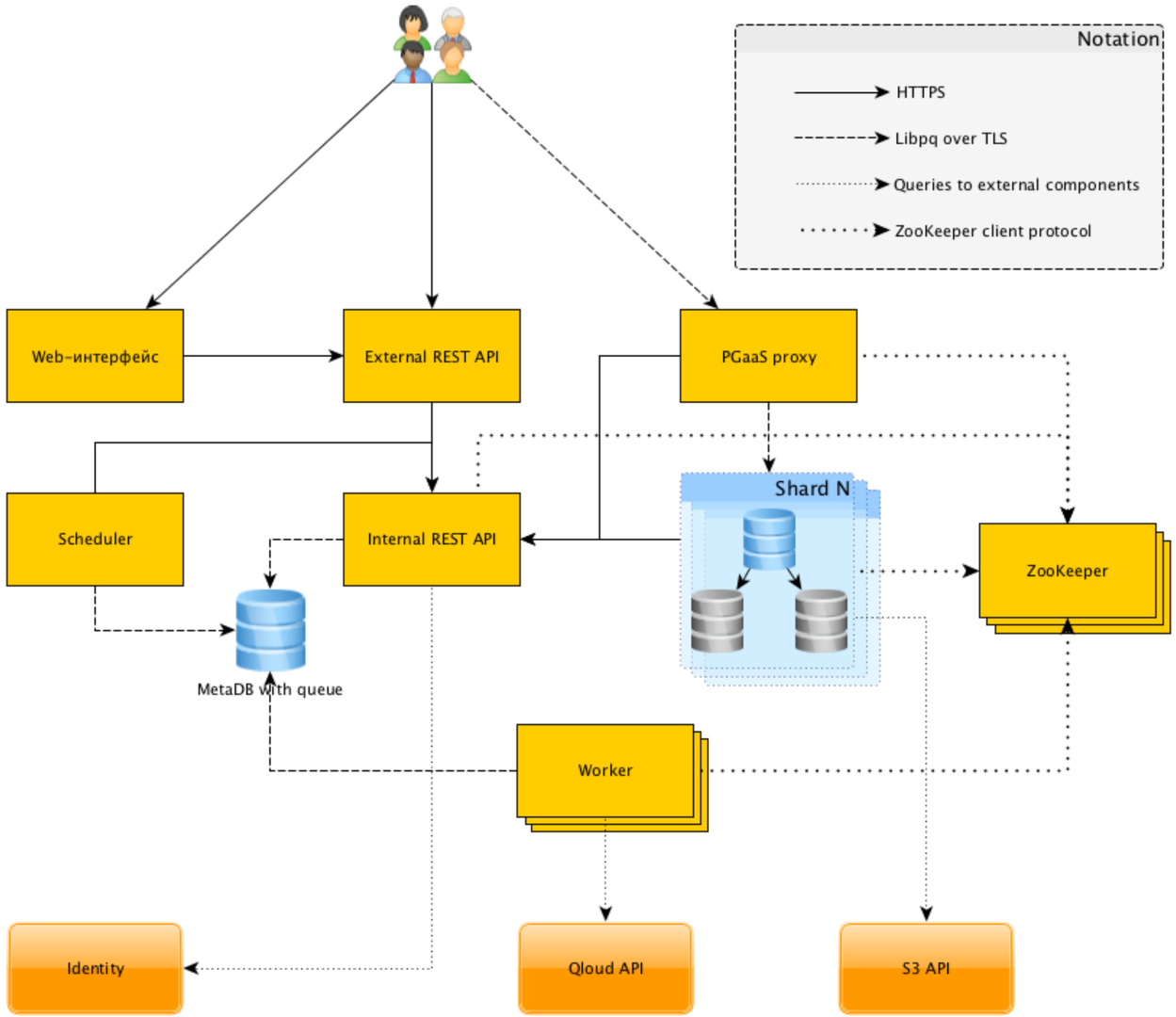
Коммуналки

- Очень дёшево
- Упростило жизнь другим командам
- Лучше осознали потребности реальных пользователей

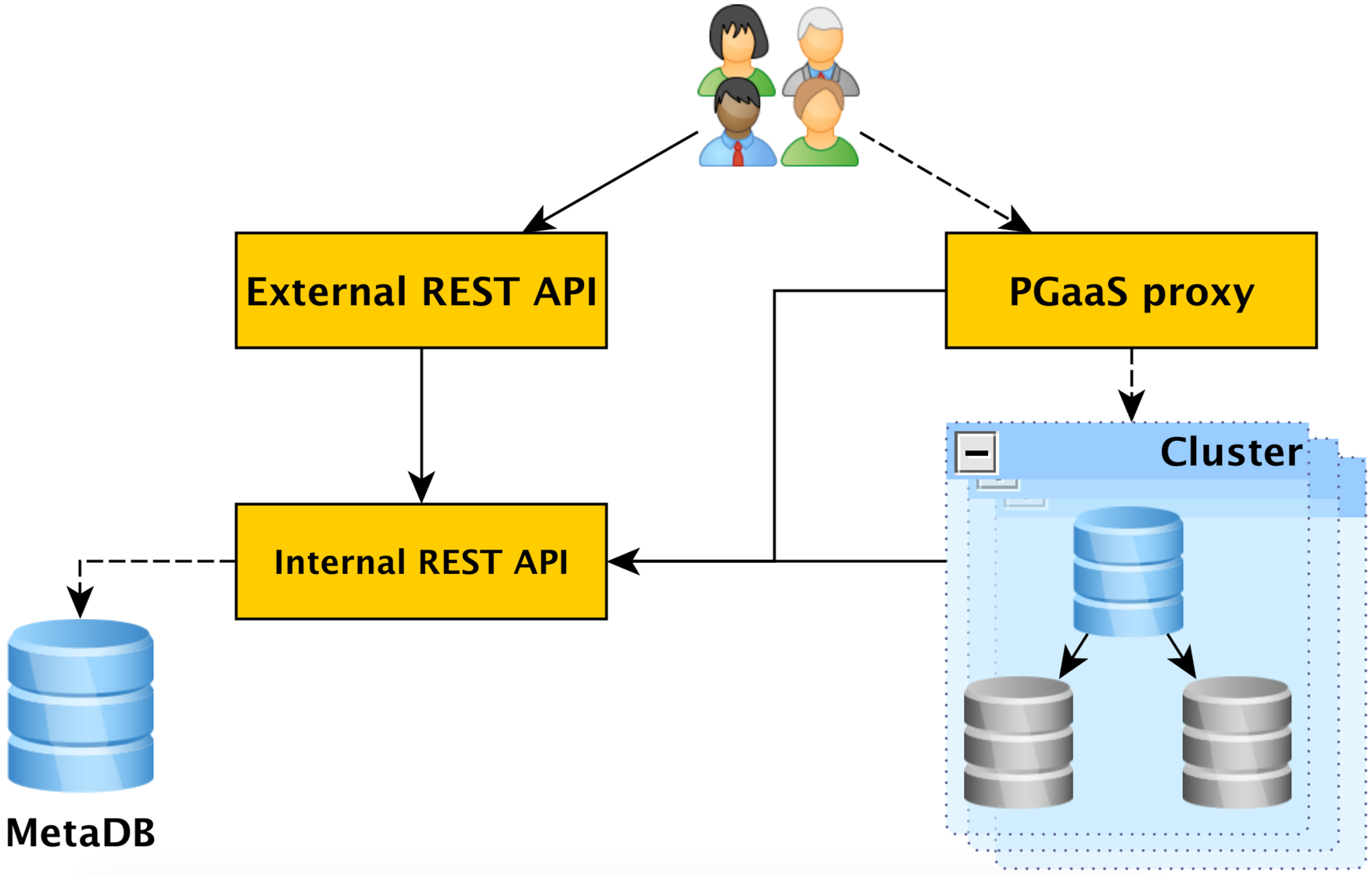
- Maintenance hell
- Не масштабируется

Вторая попытка

PostgreSQL as a service



PostgreSQL as a service



PostgreSQL as a service

- Внутренние облако и S3
- github.com/yandex/porto для изоляции
- github.com/wal-g/wal-g для резервных копий
- Самописное решение для отказоустойчивости с использованием ZK
- PGaaS-agent на хостах для управления
- PGaaS proxy

PGaaS-агент

- Постоянно запущен в контейнере
- Забирает из API желаемую ревизию и сравнивает с применённой
- Если отличается, получает diff и применяет его наименее инвазивным образом

PGaaS proxy

Нужны, чтобы скрыть от пользователя детали реализации (например, кто является мастером)

- Набор stateless хостов за SLB
- Каждый poll’ит все хосты всех баз
- Меняет конфигурацию pgbouncer’ов

PGaaS proxy

Предоставляет пользователям три строки подключения для БД:

- testdb — мастер
- testdb_ro_sync — синхронная реплика
- testdb_ro_local — ближайший хост

PGaaS proxy

Не больше ~ 10.000 кластеров на один кластер pgaas-proxy
Единая точка входа для всех пользователей

- Есть вопросы с точки зрения безопасности
- Много всего сломается при проблемах с ними
- Имя базы должно быть глобально уникальным

Проксирование длинных запросов (например, pg_dump) – боль

Особенности

Только PostgreSQL
Из коробки настроено для высоких нагрузок
Многие вещи средствами СУБД из-за ограничений платформы:

- Нет floating IP
- Нет сетевых дисков

Нужно не в porto и не только postgres, а ещё в отдельных сетях и ...

Руководитель Яндекс.Облака

Третья попытка

Два стула

Виртуализация

- Porto-контейнеры
- Qemu-виртуалки

Сетевая конфигурация
Auth-провайдеры
Требования безопасности
DNS
Рисовалки графиков

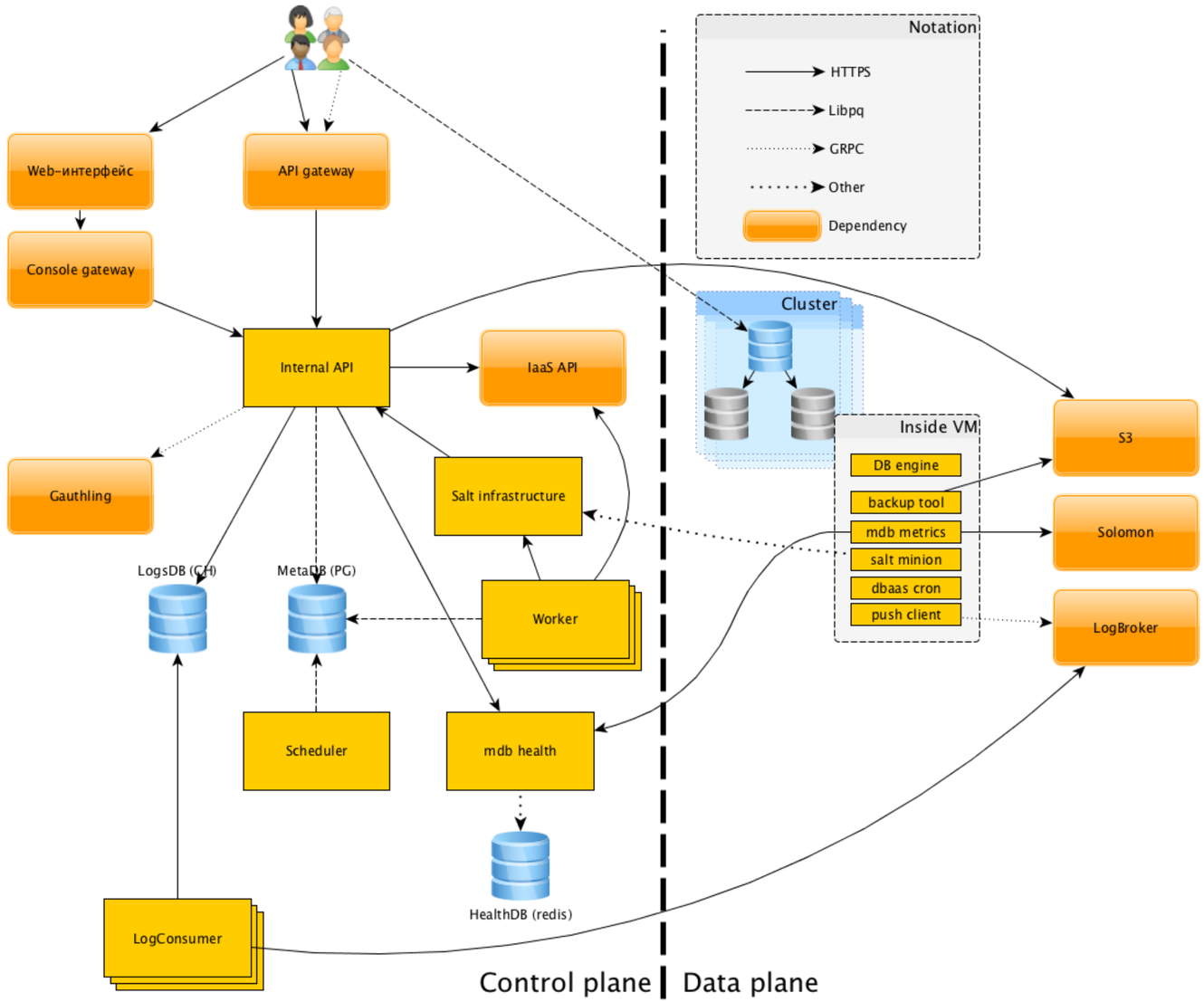
Разные платформы

- Вернулись к использованию SaltStack
- Стали запускать в контейнерах полноценную ОС, чтобы было как в VM
- Много мелких различий спрятали за if’ы

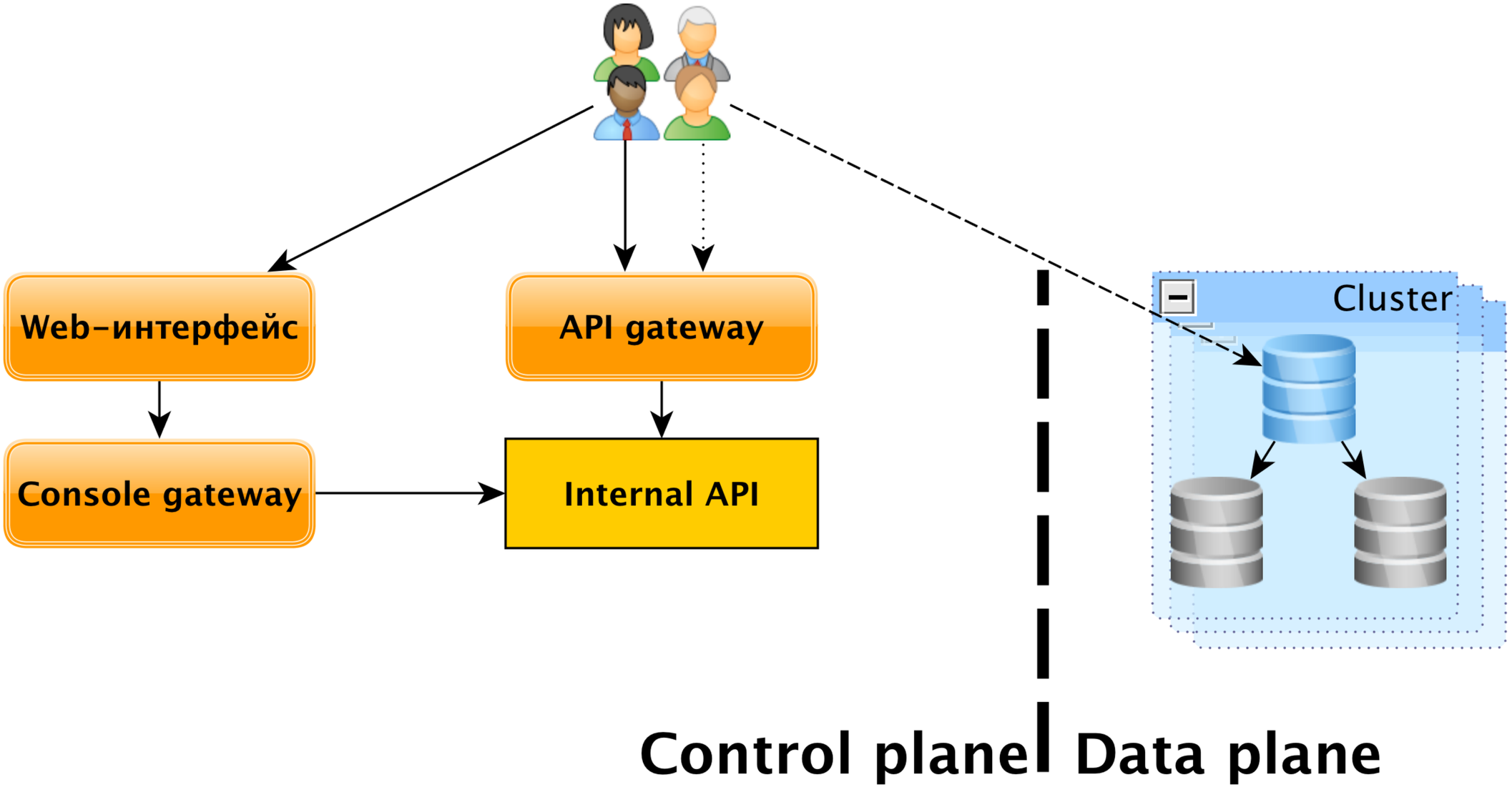
Разные типы СУБД

- Свой control plane для каждого типа
- Универсальный control plane

MDB


Notation
HTTPS
Libpq
GRPC
Other
Dependency
Web-интерфейс
API gateway
Console gateway
Internal API
IaaS API
Gauthling
Salt infrastructure
Cluster
Inside VM
DB engine
backup tool
mdb metrics
salt minion
dbaas cron
push client
S3
Solomon
LogBroker
LogsDB (CH)
MetaDB (PG)
Worker
Scheduler
mdb health
HealthDB (redis)
LogConsumer
Control plane
Data plane

MDB

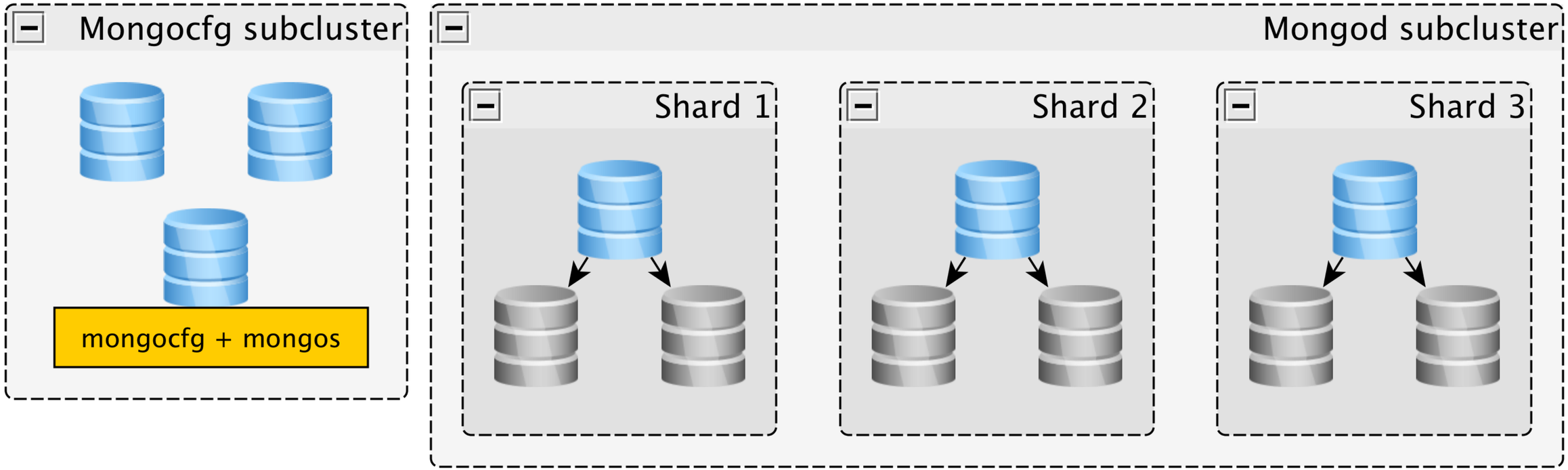

Web-интерфейс
API gateway
Console gateway
Internal API
Cluster
Control plane
Data plane

Универсальный control plane

Новая модель данных:

- Cluster
- Subcluster
- Shard
- Host
- Service

Универсальный control plane



Универсальный control plane

Денормализация хранения многих вещей в базе:

- Пользователи
- Базы
- Специфичные для СУБД вещи

Быстрый старт

Нетипизированное API

```
POST /mdb/v1/clusters/{clusterId}/updateOptions

{
    "infrastructureOptions": {
        "instanceType": "string",
        "volumeSize": 0
    },
    "databaseOptions": {}
}
```

Нетипизированное API

```
POST /mdb/v1/clusters/e4ul8kt76j2p0tp4mup3/updateOptions

{
    "databaseOptions": {
        "postgres": {
          "wal_level":"logical"
        }
    }
}
```

Нетипизированное API

```
POST /mdb/v1/clusters/e4uov44f3qoo7sro6gqb/updateOptions

{
    "databaseOptions": {
        "mongodb": {
            "mongod": {
                "storage": {
                    "wiredTiger": {
                        "engineConfig": {
                            "cacheSizeGB": 16
}}}}}}}
```

Нетипизированное API

- Удобно с точки зрения кода (сначала)
- Требует очень подробной документации
- Неочевидно для пользователей

Типизированное API

```
syntax = "proto3";

// Updates the specified PostgreSQL cluster.

  rpc Update (UpdateClusterRequest) returns (operation.Operation) {

    option (google.api.http) = {

       patch: "/managed-postgresql/v1/clusters/{cluster_id}" body: "*" };

    option (yandex.api.operation) = {

      metadata: "UpdateClusterMetadata”

      response: "Cluster” };

  }
```

Типизированное API

```
message UpdateClusterRequest {
  string cluster_id = 1;
  string description = 2;
  map<string, string> labels = 3;
  // New configuration and resources for hosts in the cluster.
  ConfigSpec config_spec = 4;
}
```

Типизированное API

```
message ConfigSpec {
  oneof postgresql_config {
    config.PostgresqlConfig9_6 postgresql_config_9_6 = 1;
    config.PostgresqlConfig10 postgresql_config_10 = 2;
  }
  <…>
}
```

Типизированное API

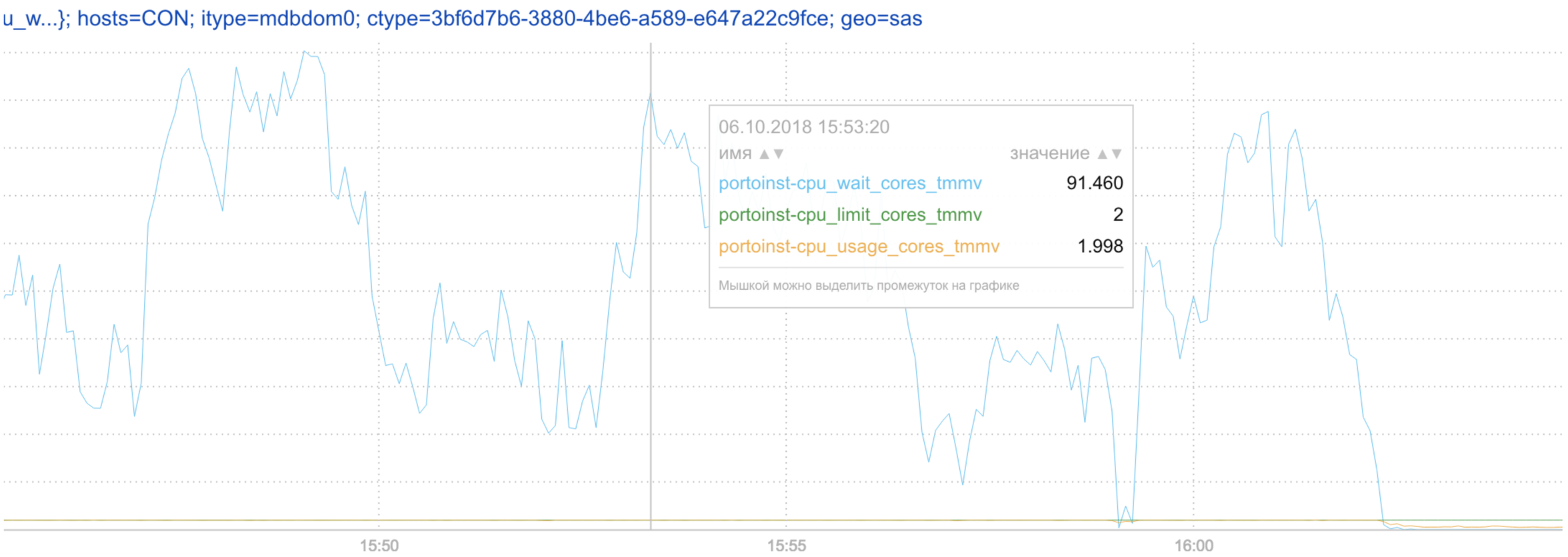
```
message PostgresqlConfig10 {
  enum WalLevel {
    WAL_LEVEL_UNSPECIFIED = 0;
    WAL_LEVEL_REPLICA = 1;
    WAL_LEVEL_LOGICAL = 2;
  }
  google.protobuf.Int64Value max_connections = 1;
  WalLevel wal_level = 2;
}
```

Типизированное API

- Удобно для пользователя
- API spec зачастую достаточно и без документации
- Единообразно со всем API Облака
- Сложнее работа с общим кодом

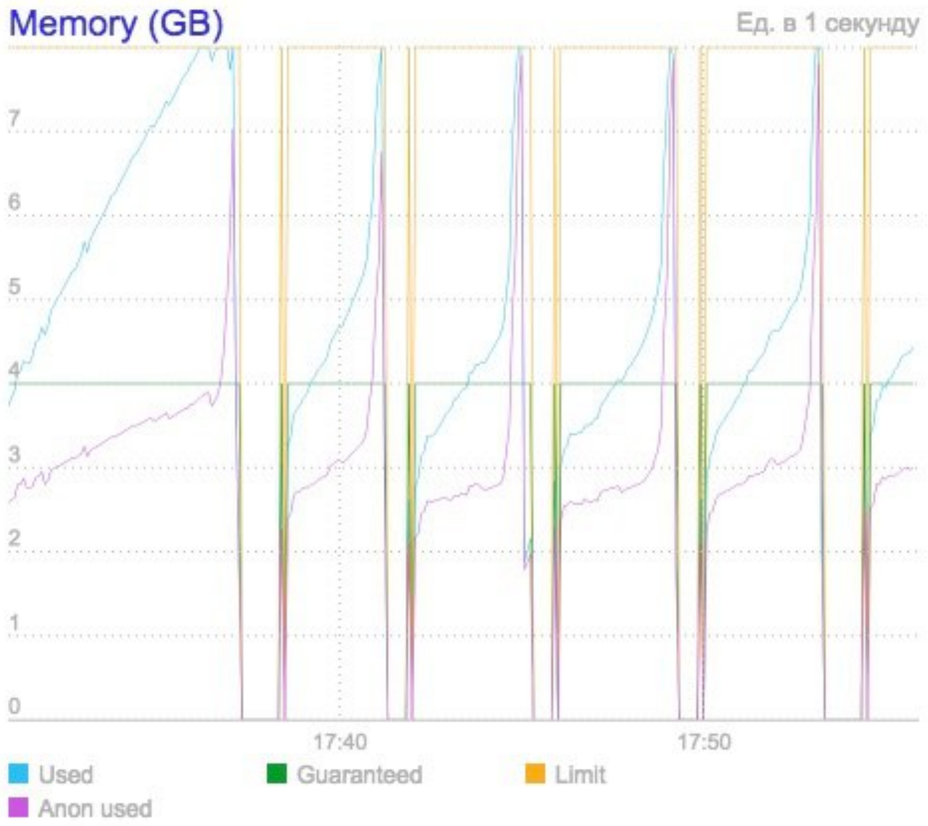
Проблемы

Нехватка ресурсов



Autofailover, бэкапы и мониторинг в таких ситуациях работают плохо

Нехватка ресурсов



Autofailover веерно переключает мастера из-за OOM

Нехватка ресурсов

Porto умеет:

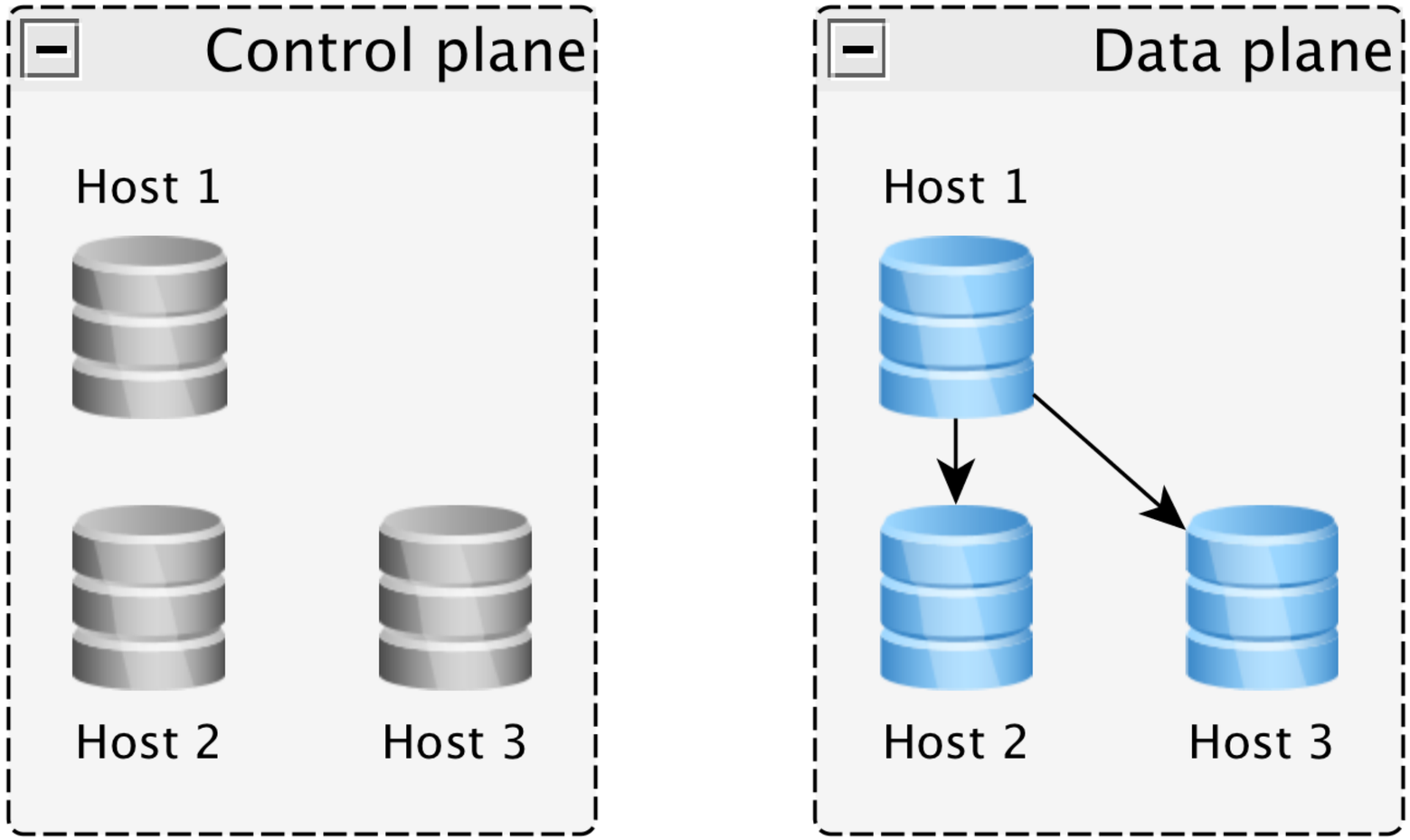
- Подконтейнеры
- Гарантии некоторых ресурсов

OOM будем обрабатывать явно

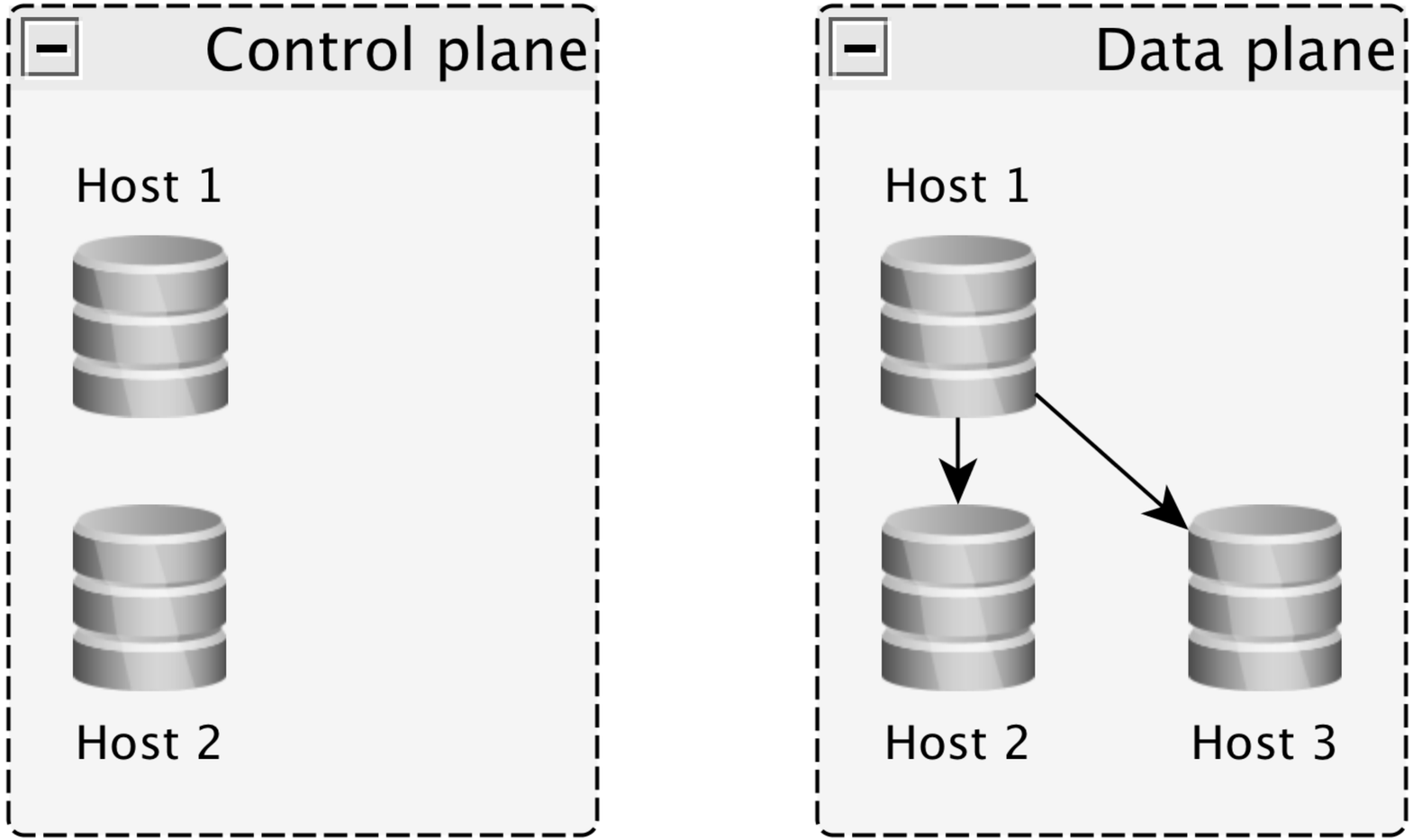
Конкурентные изменения состава кластера

- Все операции над одним кластером выполняются строго последовательно
- Но целевое состояние в MetaDB обновляется сразу
- В итоге более ранняя операция может применить изменения от более поздней

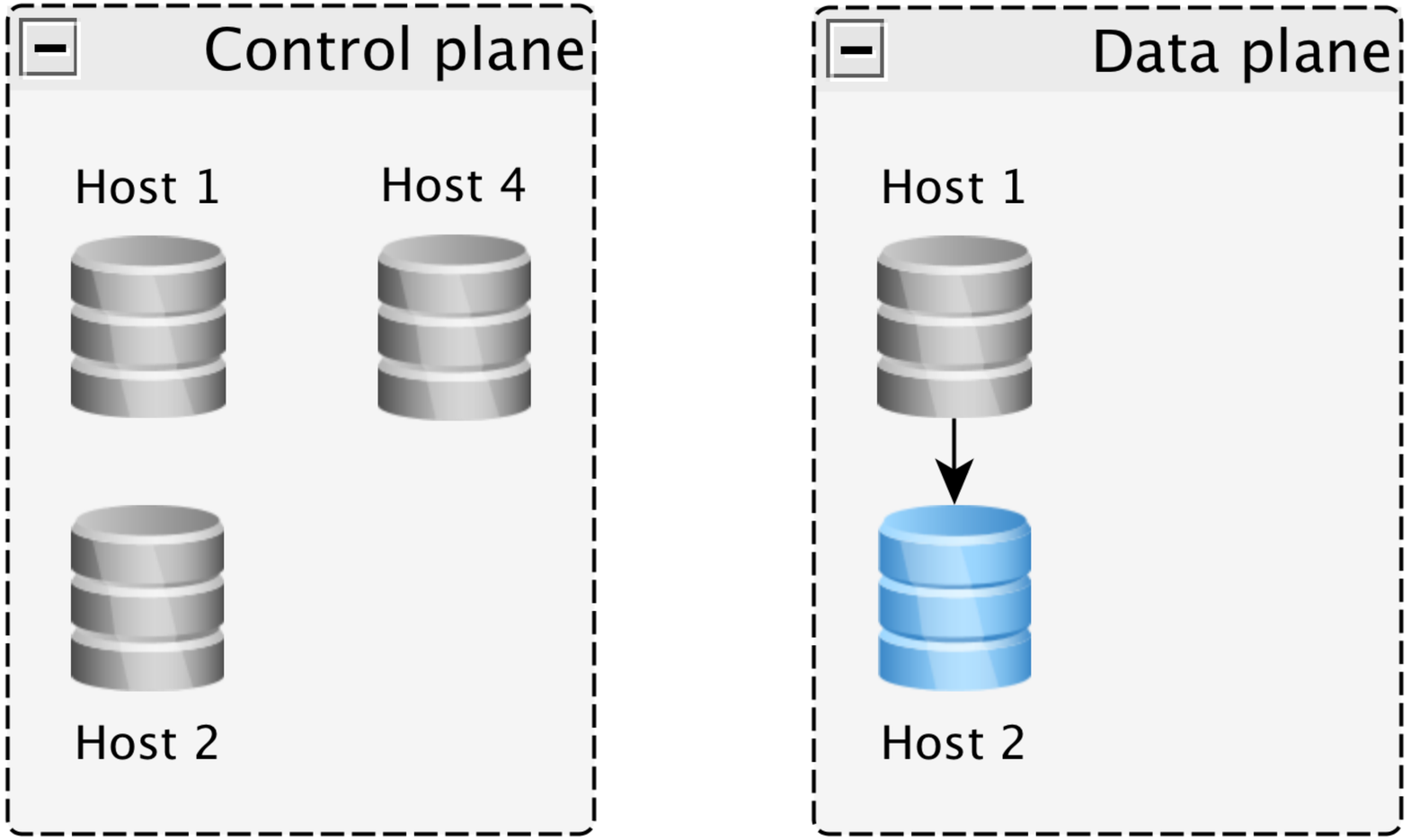
Конкурентные изменения состава кластера



Конкурентные изменения состава кластера



Конкурентные изменения состава кластера



Конкурентные изменения состава кластера

- Host 1 не сможет добавить правила для соединений с host 4
- Deploy на host 4 не удастся

Конкурентные изменения состава кластера

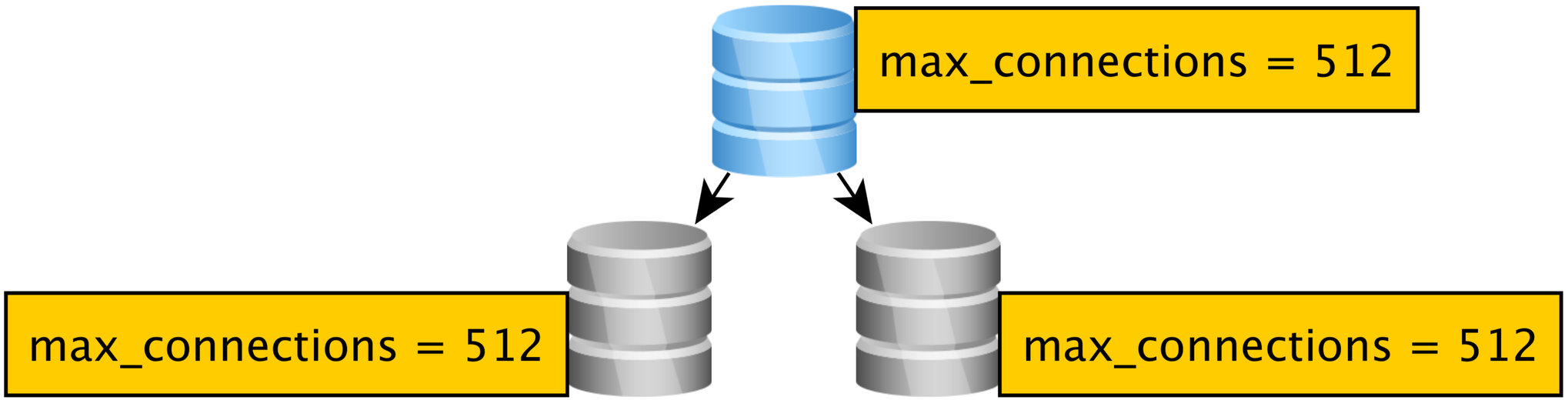
- Пока потенциально деструктивные операции сделали эксклюзивными
- Делаем версионирование в MetaDB

Порядок применения настроек

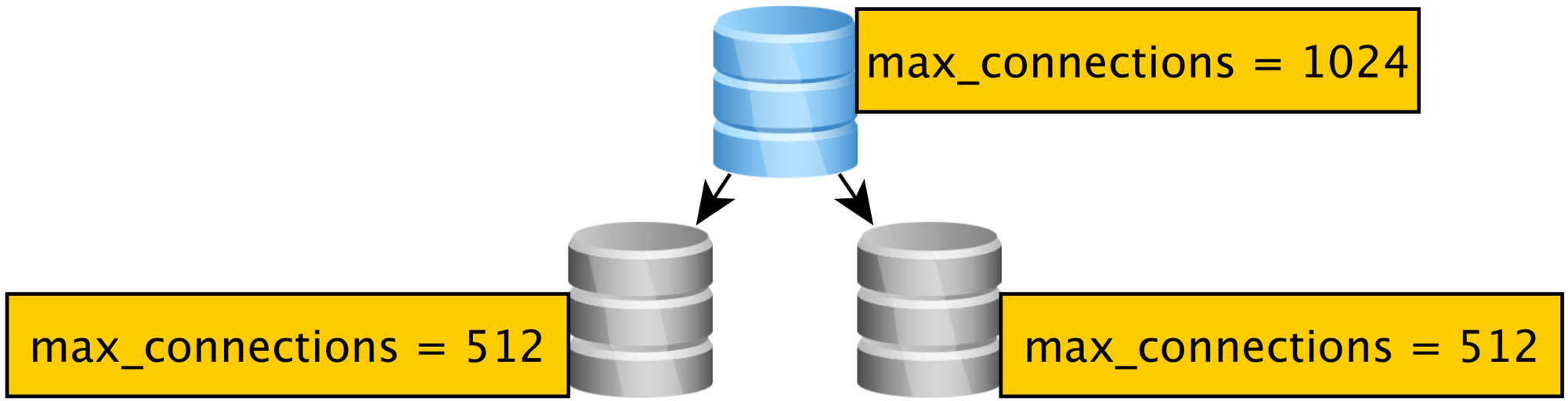
Некоторые настройки:

- (не) требуют перезапуска
- требуют особого порядка применения на хостах кластера
- зависят от доступных ресурсов

Порядок применения настроек



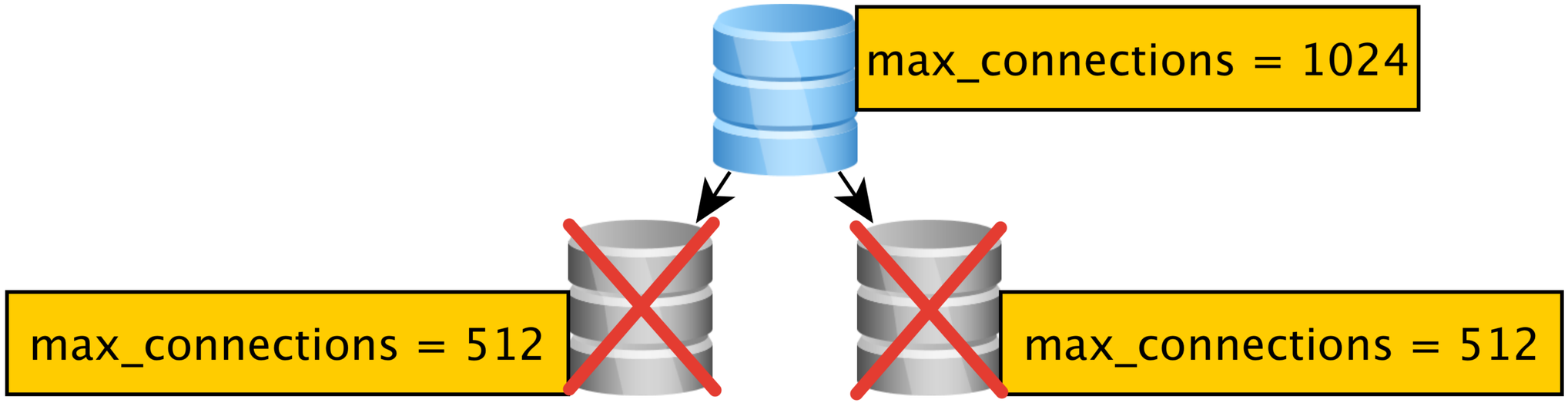
Порядок применения настроек



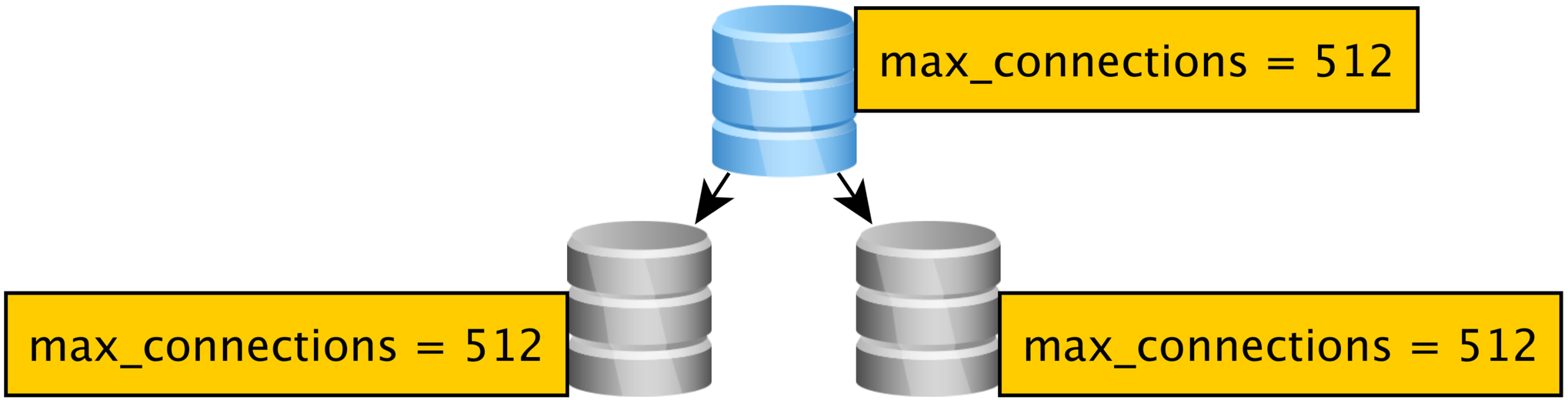
Порядок применения настроек

```
FATAL:  hot standby is not possible because max_connections = 512 is a lower setting than on the master server (its value was 1024)
CONTEXT:  xlog redo at 38/87000098 for XLOG/PARAMETER_CHANGE: max_connections=1024 max_worker_processes=8 max_prepared_xacts=0 max_locks_per_xact=64 wal_level=logical wal_log_hints=off track_commit_timestamp=off
LOG:  startup process (PID 706305) exited with exit code 1
```

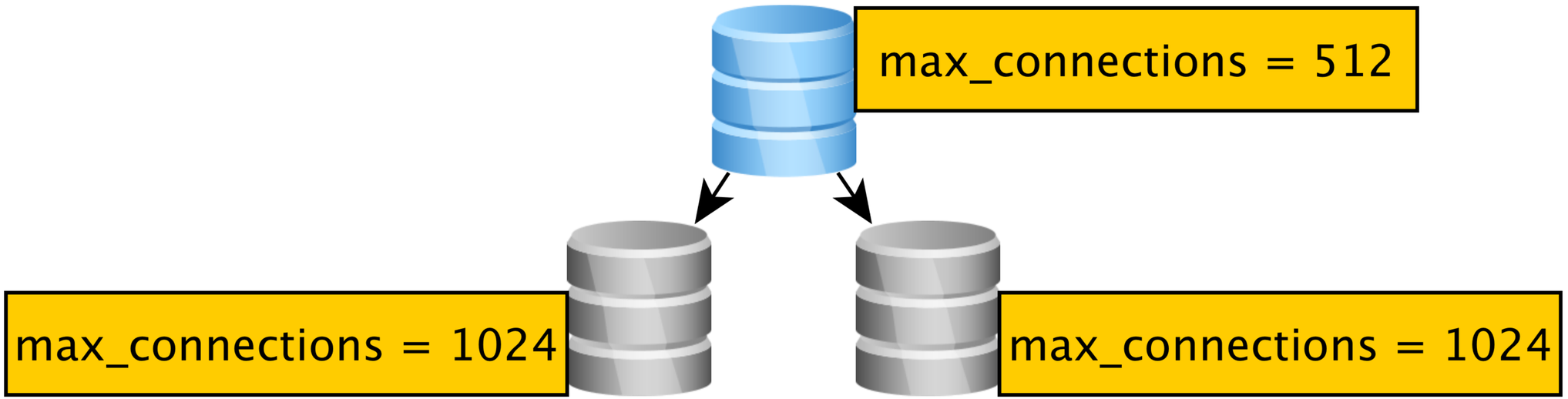
Порядок применения настроек



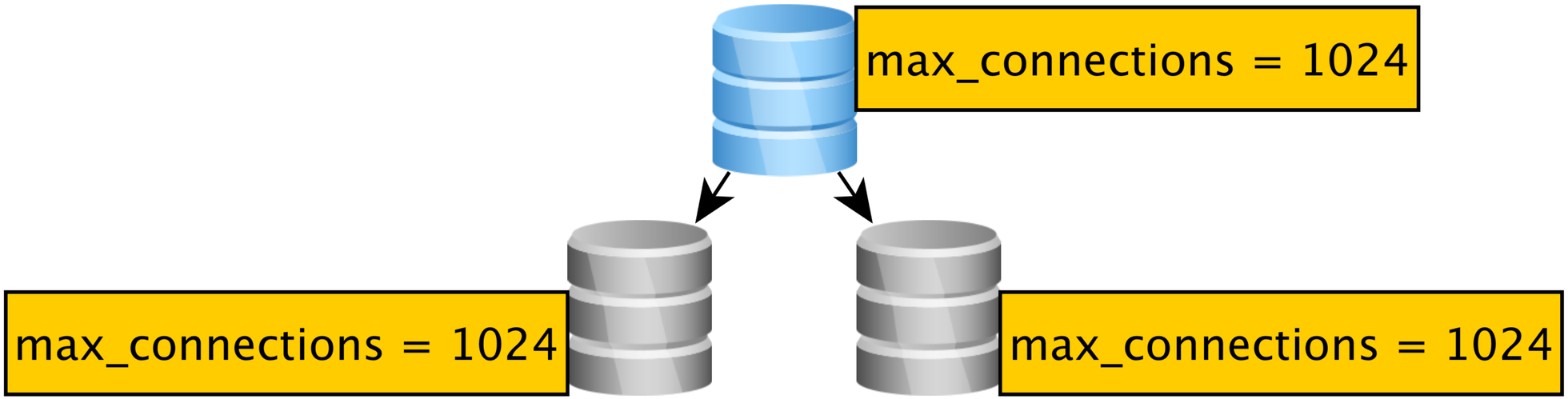
Порядок применения настроек



Порядок применения настроек



Порядок применения настроек



Порядок применения настроек

Если вверх, то сначала реплики, потом мастер

Если вниз, то сначала мастер, потом реплики

API при постановке задачи обновления настроек назначает одну из политик применения

Заключение

Итог

Два года в production
1000+ кластеров
Один control plane для нескольких СУБД:

- PostgreSQL, ClickHouse, MongoDB
- Ещё три в разработке

Одна кодовая база внутри и снаружи

Отличия

- Тюнинг СУБД из коробки
- Autofailover и бэкапы средствами СУБД
- Нормальные графики
- И сетевые, и локальные диски

- Многие вещи только через API
- Уникальная модель данных

cloud.yandex.ru/services/mdb

Вопросы?

Бородин Владимир

Руководитель службы сервисов хранения данных

 d0uble@yandex-team.ru

 +7 495 739-70-00 , доб. 7255