Внутри Метастора S3

Эволюция современного хранилища метаданных

Данил Кислов

Разработчик хранилищ
для внутреннего облака One-cloud

S3 в One-Cloud

50
кластеров

~43 млрд
объектов

~960
хостов метастора (12 000 ядер)

~56 PB
данных

~3,61 PB
читается каждый день

Используется
множеством продуктов
и внутренних сервисов

~544 TB
пишется каждый день

~496 TB
удаляется каждый день

Что такое S3?

У S3 нет спецификации,
но есть референс

S3 – объектное
хранилище данных от
компании Amazon

S3

Объектные хранилища,
подстраивающиеся
под референс S3, называют
S3-совместимыми

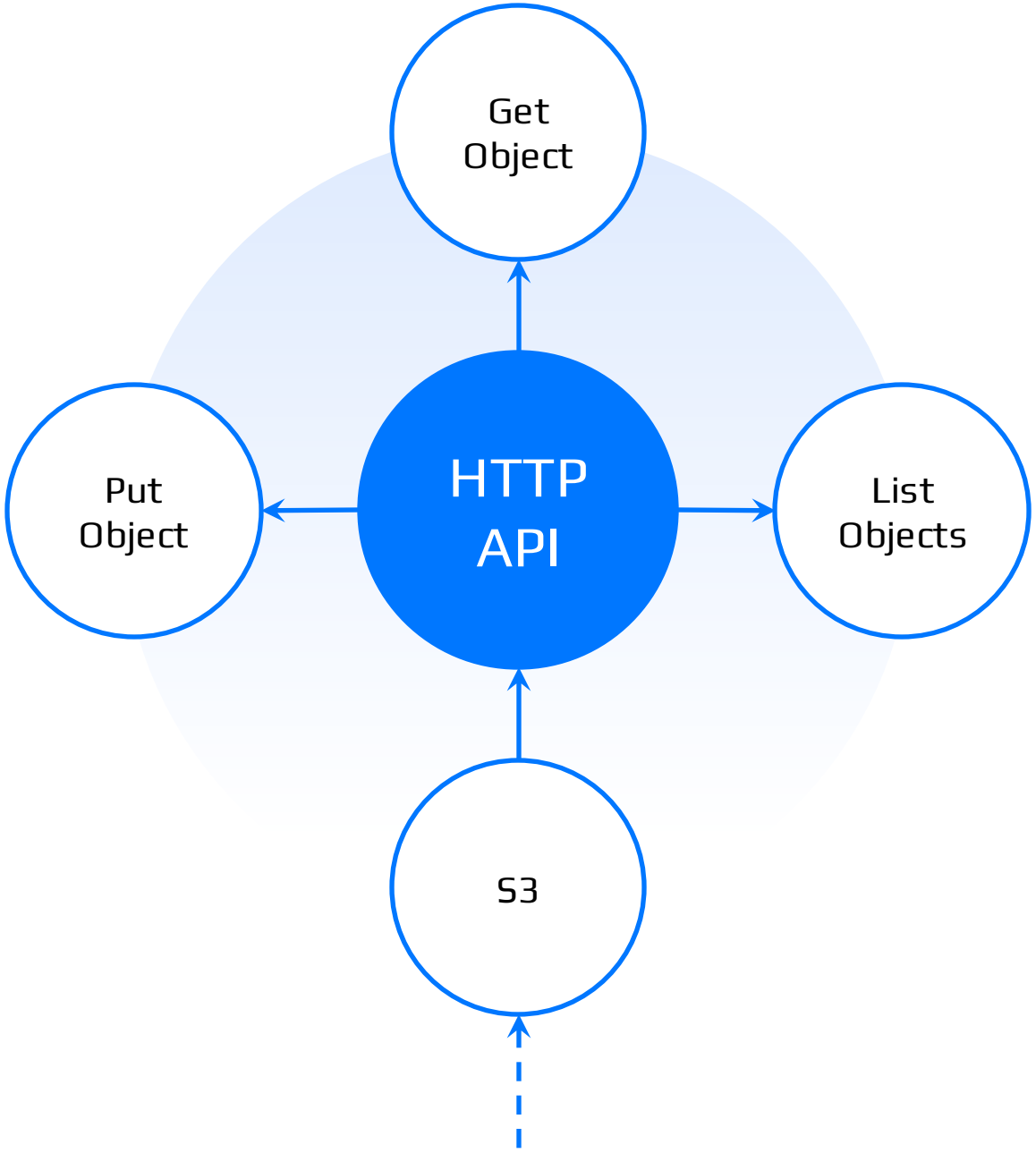
MINIO

ceph

Базовые возможности

HTTP API

- GetObject
- PutObject
- DeleteObject(s)
- CopyObject
- ListObjects



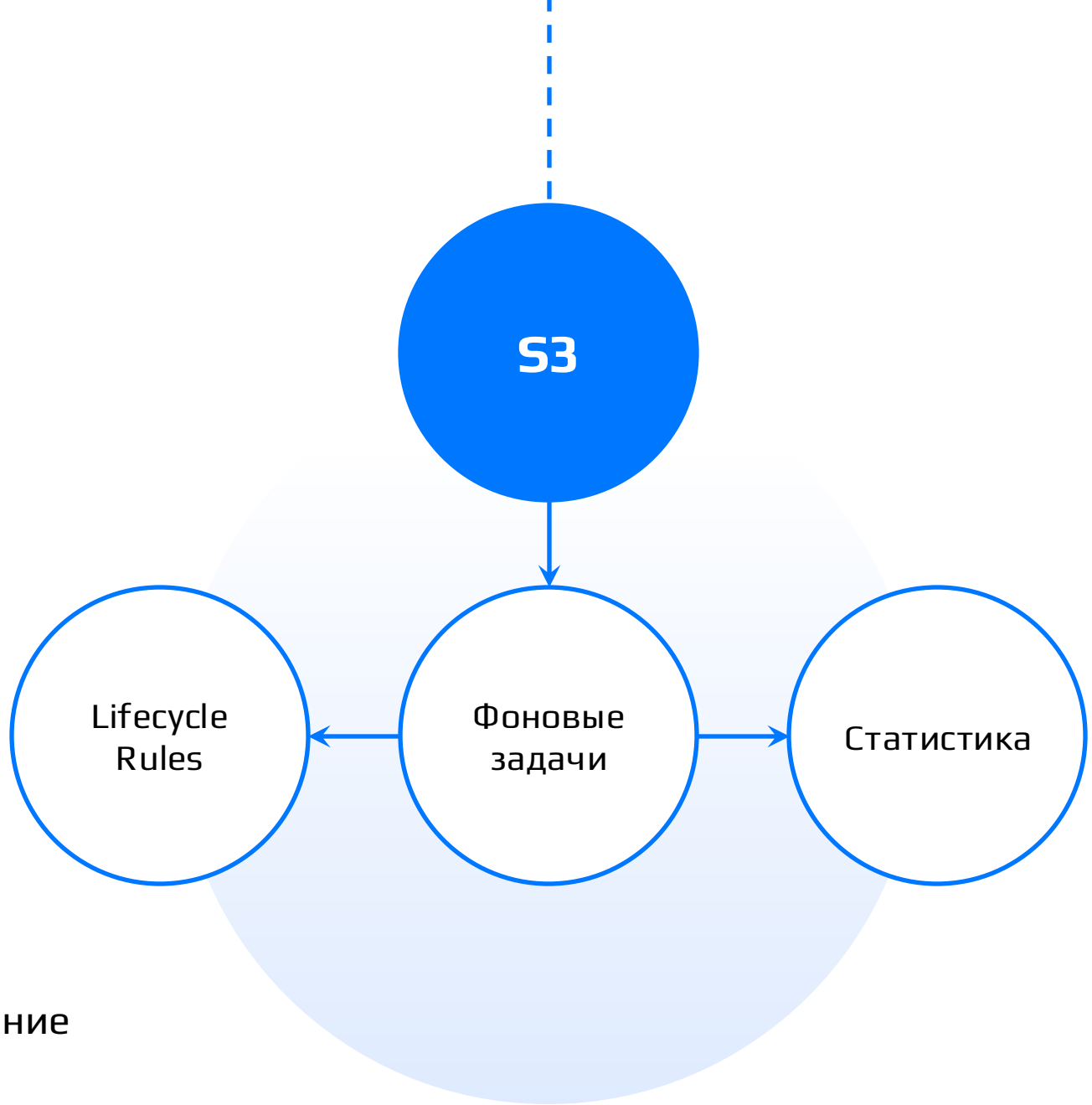
Базовые возможности

Lifecycle rules

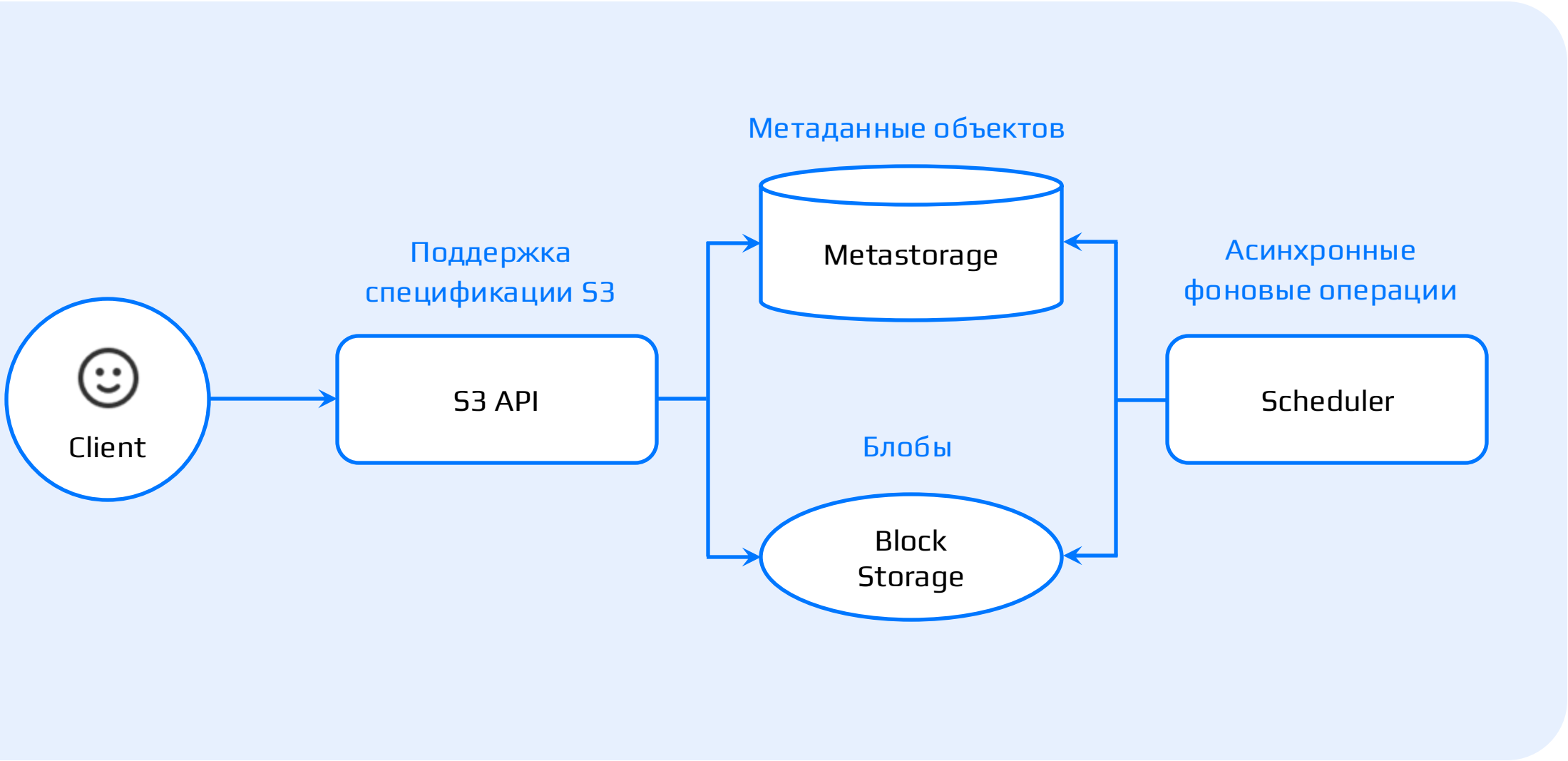
- Transition action
- Expiration action
- Expire noncurrent
- Expire after days / after date
- AbortIncompleteMultipartUpload
- ExpiredObjectDeleteMarker

Статистика

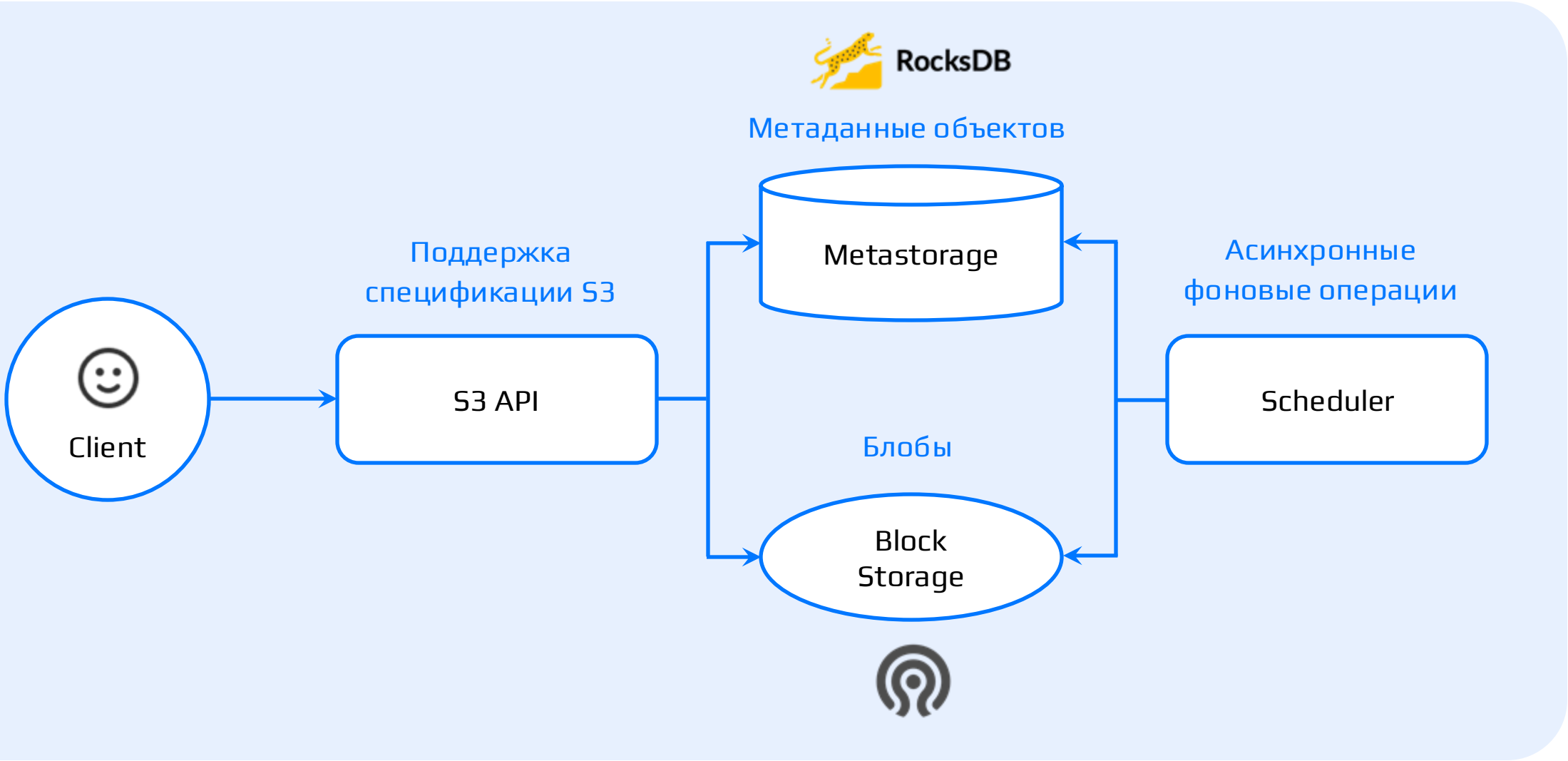
Мониторинг | Биллинг | Квотирование



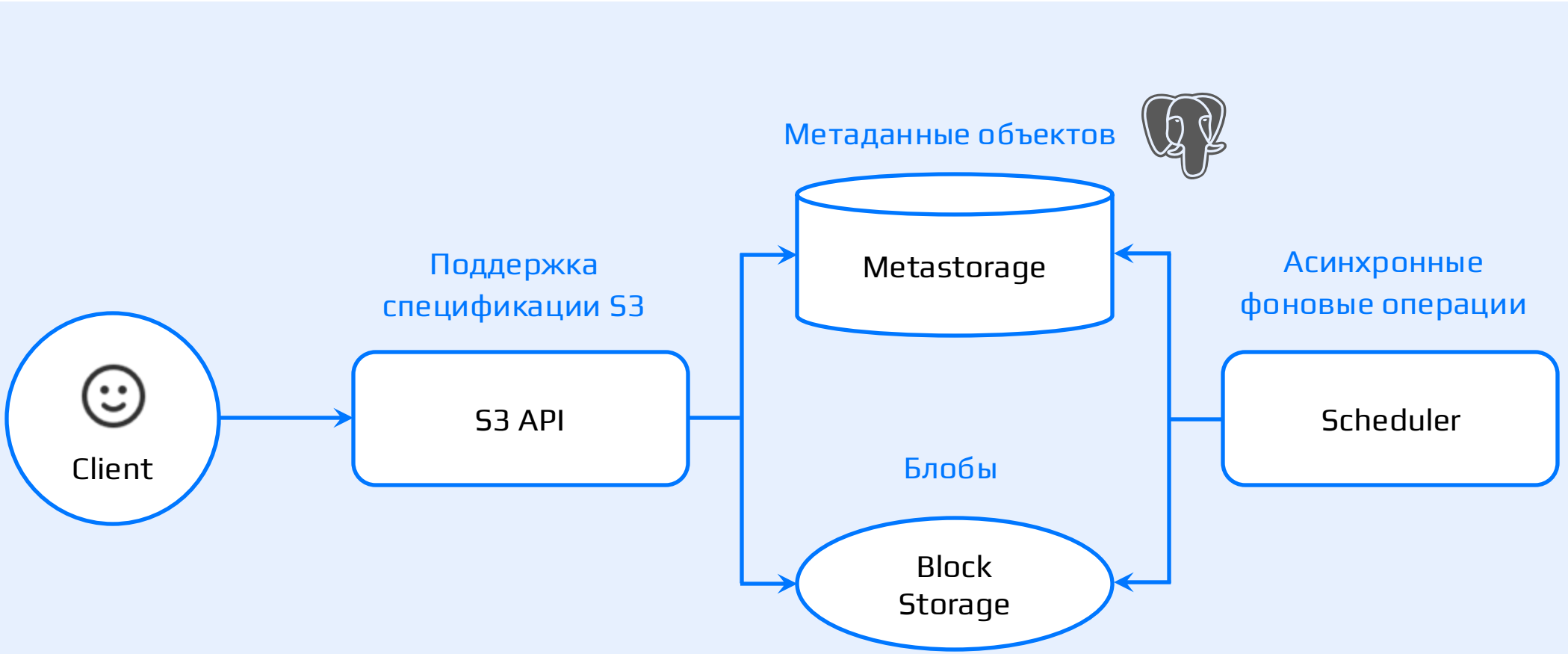
Архитектура: шаблон



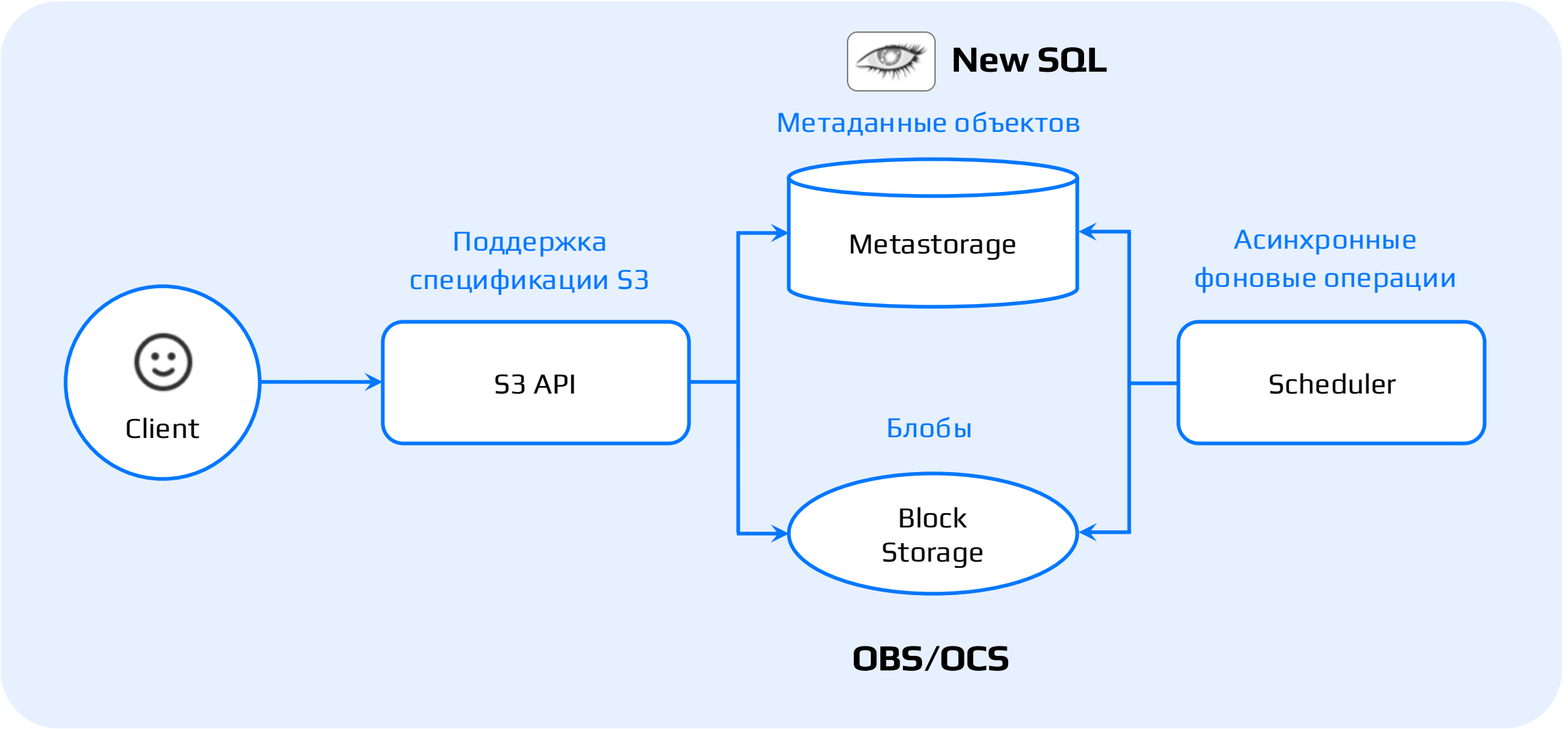
Архитектура: Ceph



Архитектура: метастор на Postgres



Архитектура: one-object-storage



*Александр Христофоров. Дешевле, надёжнее, проще: хранение петабайтов видео и фото в Одноклассниках

Зачем своё решение?

Одна или несколько из этих проблем:

Сложная поддержка
и масштабирование

*Вадим Цесько. OK S3: Строим Систему Сами (RU)

Зачем своё решение?

Одна или несколько из этих проблем:

Сложная поддержка
и масштабирование

Отсутствие
достоверных данных
о крупных кластерах

*Вадим Цесько. OK S3: Строим Систему Сами (RU)

Зачем своё решение?

Одна или несколько из этих проблем:

Сложная поддержка
и масштабирование

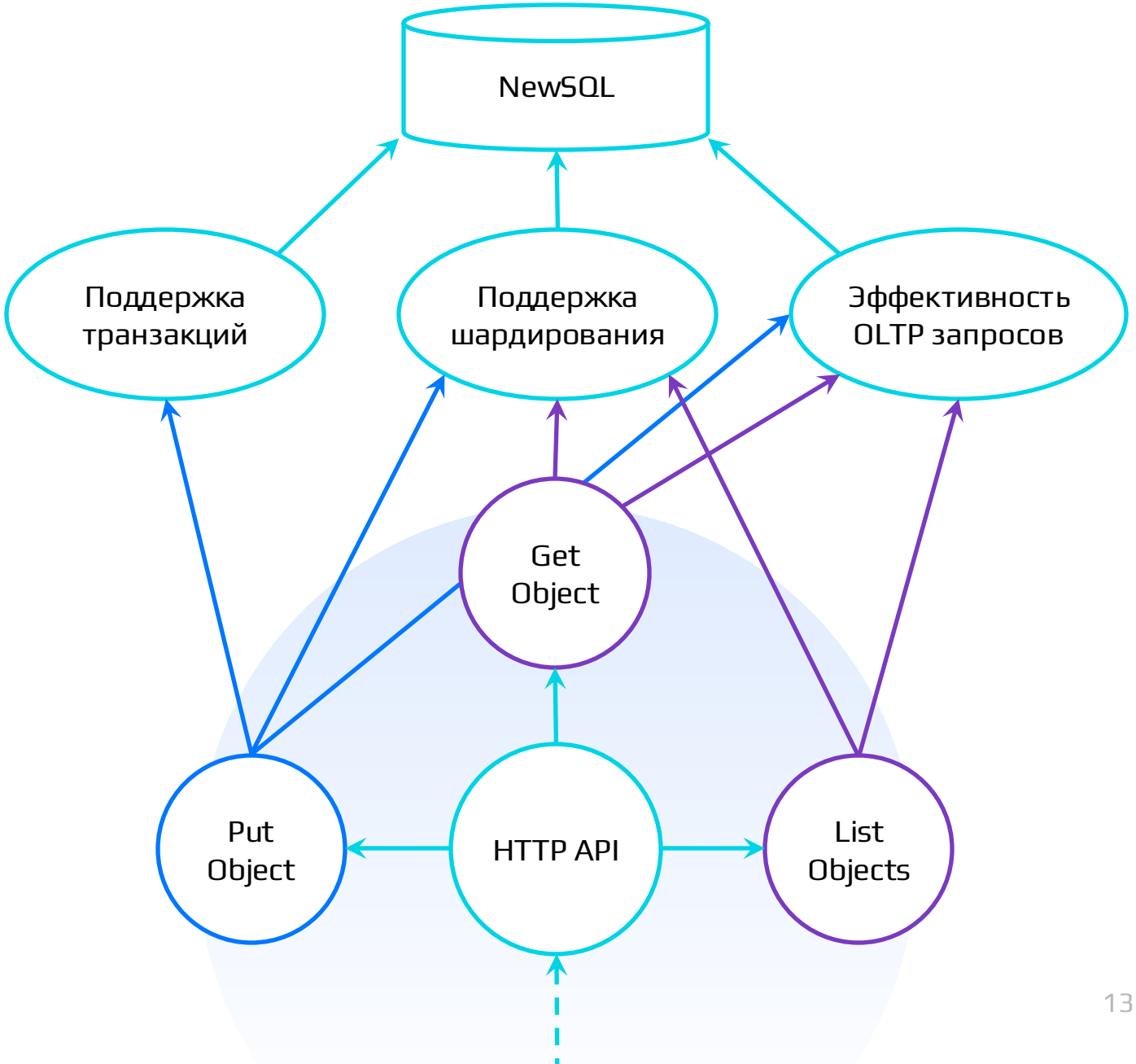
Отсутствие
достоверных данных
о крупных кластерах

Незнакомый
стек технологий

*Вадим Цесько. OK S3: Строим Систему Сами (RU)

NewSQL*

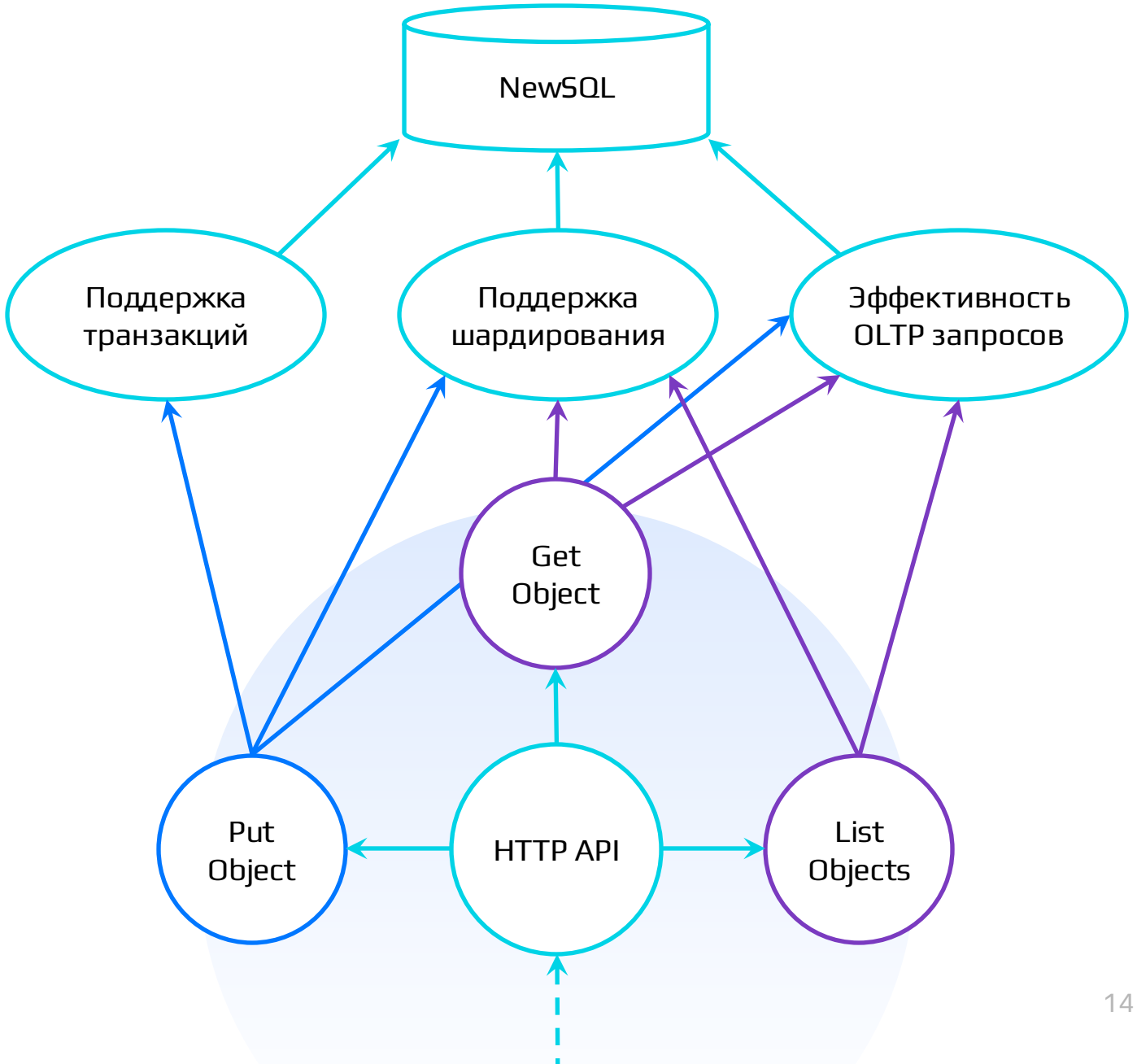
- **На базе Apache Cassandra**
- **+ координатор транзакций**



*Oleg Anastasev. NewSQL = NoSQL + ACID (RU)

NewSQL*

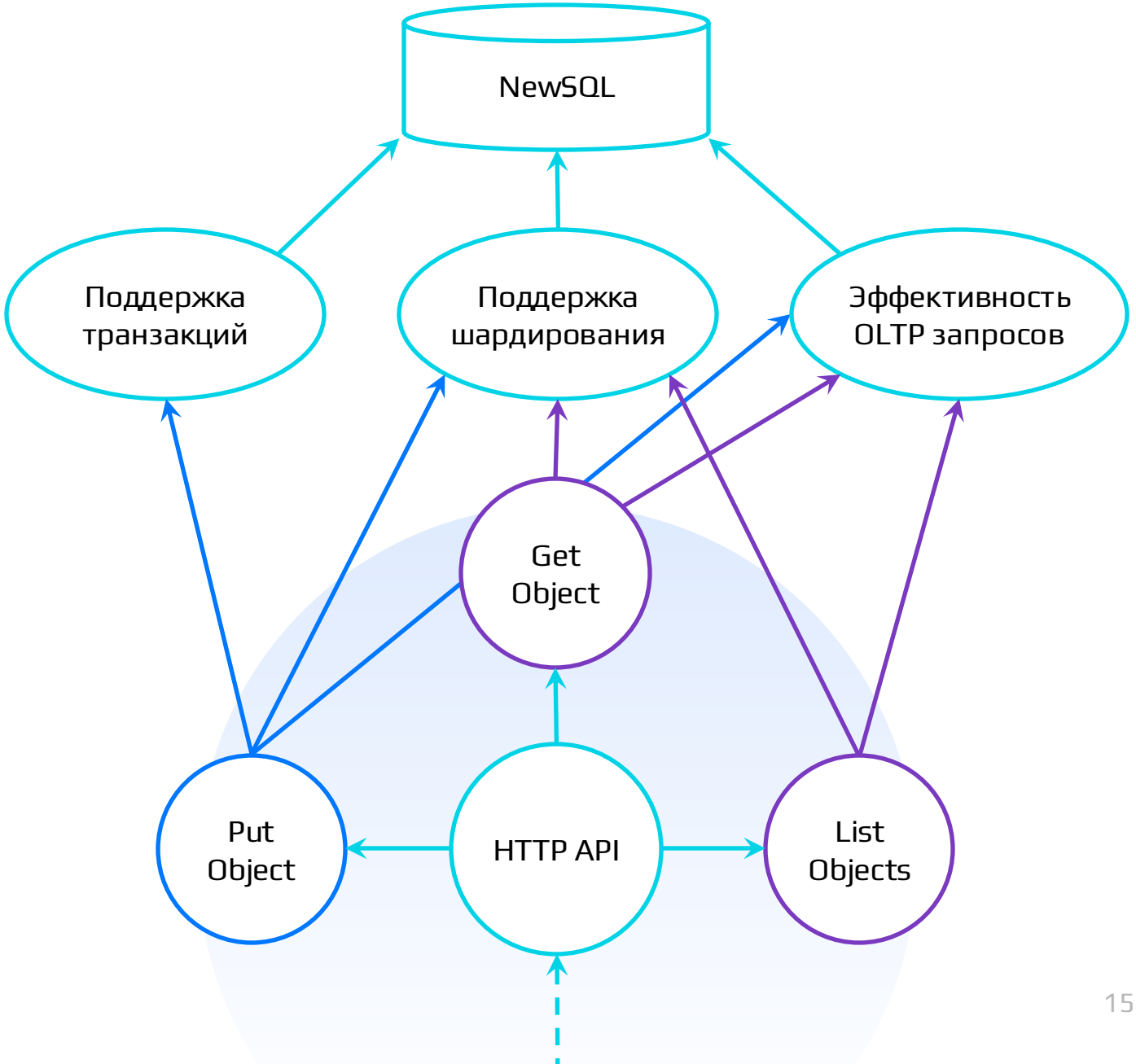
- На базе Apache Cassandra
- + координатор транзакций
- **Поддерживает ACID транзакции в рамках одной партиции**



*Oleg Anastasev. NewSQL = NoSQL + ACID (RU)

NewSQL*

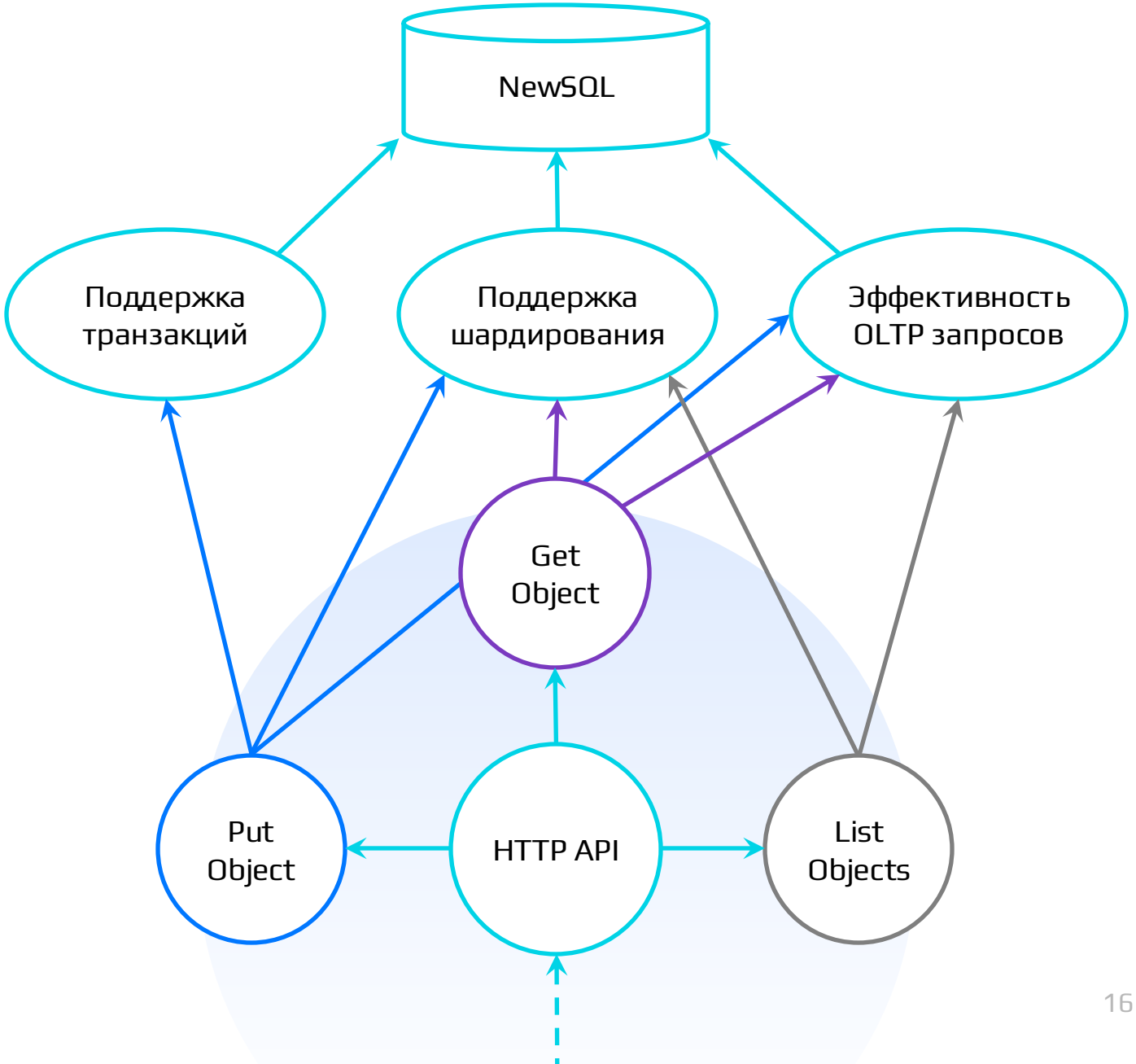
- На базе Apache Cassandra
- + координатор транзакций
- Поддерживает ACID транзакции в рамках одной партиции
- **Шардированная**



*Oleg Anastasev. NewSQL = NoSQL + ACID (RU)

NewSQL*

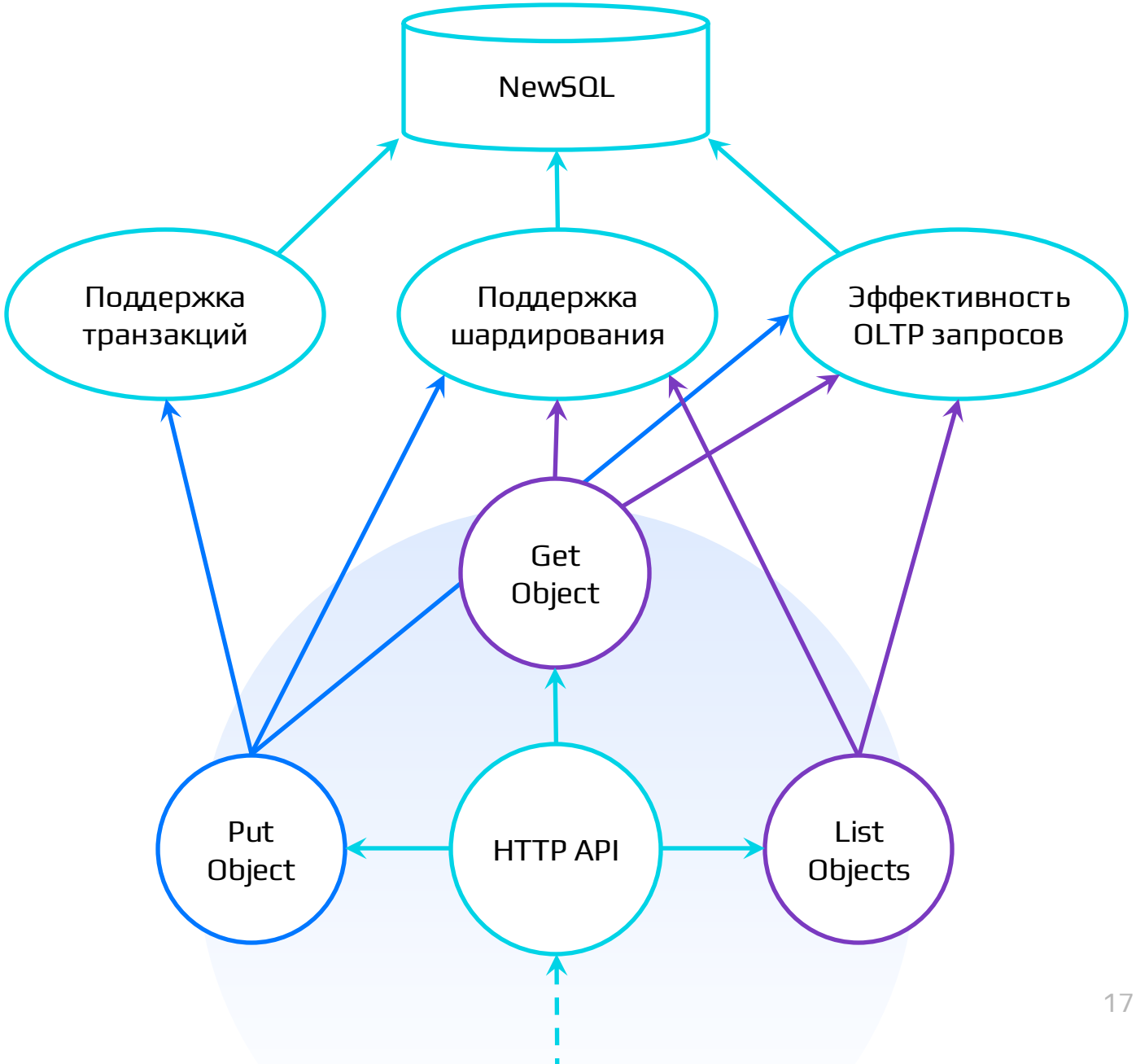
- На базе Apache Cassandra
- + координатор транзакций
- Поддерживает ACID транзакции в рамках одной партиции
- Шардированная
- **Легко масштабируется**



*Oleg Anastasev. NewSQL = NoSQL + ACID (RU)

NewSQL*

- На базе Apache Cassandra
- + координатор транзакций
- Поддерживает ACID транзакции в рамках одной партиции
- Шардированная
- Легко масштабируется
- **Высокая скорость запросов и доступность**



*Oleg Anastasev. NewSQL = NoSQL + ACID (RU)

Что хранится в метасторе?

- **Список версий и маппинг версий на блоки**

Parent (text)	Name (text)	Version (timeuuid)	Blocks (blob)
a/	obj1.txt	a1423c3e-...-47f7bc029f10	blockId1, blockId2, blockId3
a/b/	obj2	a1423c4e-...-e41a4343efd7	blockId

Что хранится в метасторе?

- Список версий и маппинг версий на блоки
- **Метаданные версии (Content-Type, Content-Encoding, Cache-Control…)**

cache_control (text)	encoding (text)	type (text)	Version (timeuuid)
null	null	image/jpeg	a1423c3e-…-47f7bc029f10

Что хранится в метасторе?

- Список версий и маппинг версий на блоки
- Метаданные версии (Content-Type, Content-Encoding, Cache-Control...)
- **Uploads и upload parts**

Parent (text)	Name (text)	Upload (timeuuid)	meta (text)
a/	obj1.txt	a1423c3e-...-47f7bc029f10	null
a/b/	obj2	a1423c4e-...-e41a4343efd7	null

Parent (text)	Name (text)	Upload (timeuuid)	Part (int)	Blocks (blob)
a/	obj1.txt	a1423c3e-...-47f7bc029f10	1	blockId1, blockId2, blockId3
a/b/	obj2	a1423c4e-...-e41a4343efd7	2	blockId

Что хранится в метасторе?

- Список версий и маппинг версий на блоки
- Метаданные версии (Content-Type, Content-Encoding, Cache-Control…)
- Uploads и upload parts
- **Индексные таблицы (objects, folder…)**

Parent	Name
images/	pets
images pets/	cats
images/ pets/cats	black

Что хранится в метасторе?

- Список версий и маппинг версий на блоки
- Метаданные версии (Content-Type, Content-Encoding, Cache-Control...)
- Uploads и upload parts
- Индексные таблицы (objects, folder...)
- **Настройки бакетов**

name (text)	value (text)
LIFECYCLE	<?xml version="1.0" encoding="UTF-8"?> <LifecycleConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/"> <Rule> <ID>Cleanup old versions after 1y and uploads after 3d</ID> <Status>Enabled</Status> <NoncurrentVersionExpiration> <NoncurrentDays>365</NoncurrentDays> </NoncurrentVersionExpiration> <AbortIncompleteMultipartUpload> <DaysAfterInitiation>3</DaysAfterInitiation> </AbortIncompleteMultipartUpload> </Rule> </LifecycleConfiguration>
VERSIONING	<?xml version="1.0" encoding="UTF-8"?> <VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/"> <Status>Suspended</Status> </VersioningConfiguration>

Что хранится в метасторе?

- Список версий и маппинг версий на блоки
- Метаданные версии (Content-Type, Content-Encoding, Cache-Control...)
- Uploads и upload parts
- Индексные таблицы (objects, folder...)
- Настройки бакетов
- **Внутренние служебные таблицы (locks, block status, block reference...)**

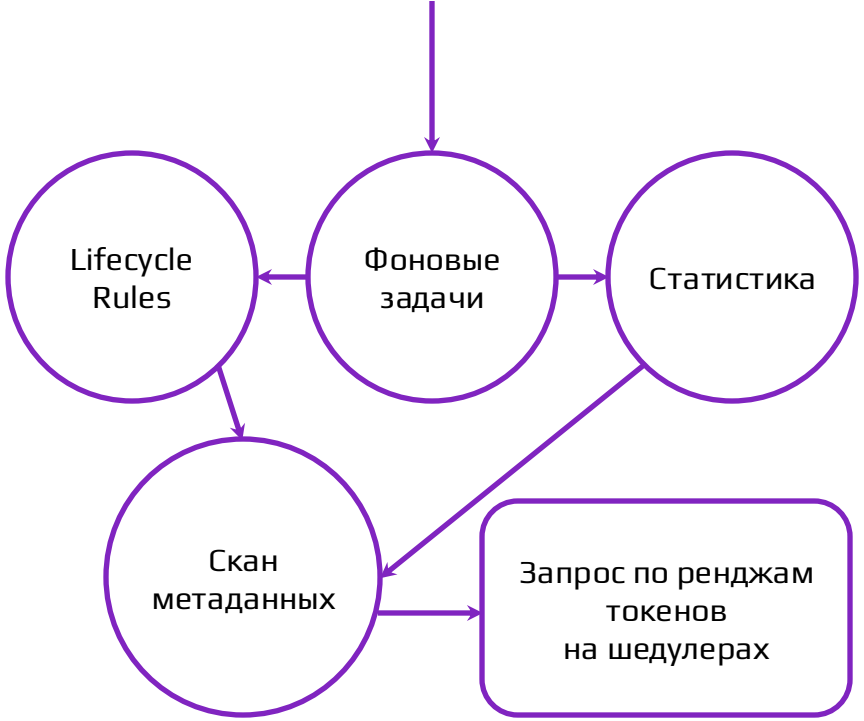
block (blob)	md5 (blob)	status (int)
0x0000000000500000e1eebbcbfbccae6b3c78fb94733266811 0a69565ee544e11143641ec88632978	0xe9d79a274c80c1f8199817f5db425165	0
0x0000000005000005a05eae7b5c534524100b9948f3d0ff696b12a726ad4bf0d21ab2b1c82ad2c0c	0x3f9cbf5f14264a250b55ccda7dca1610	1

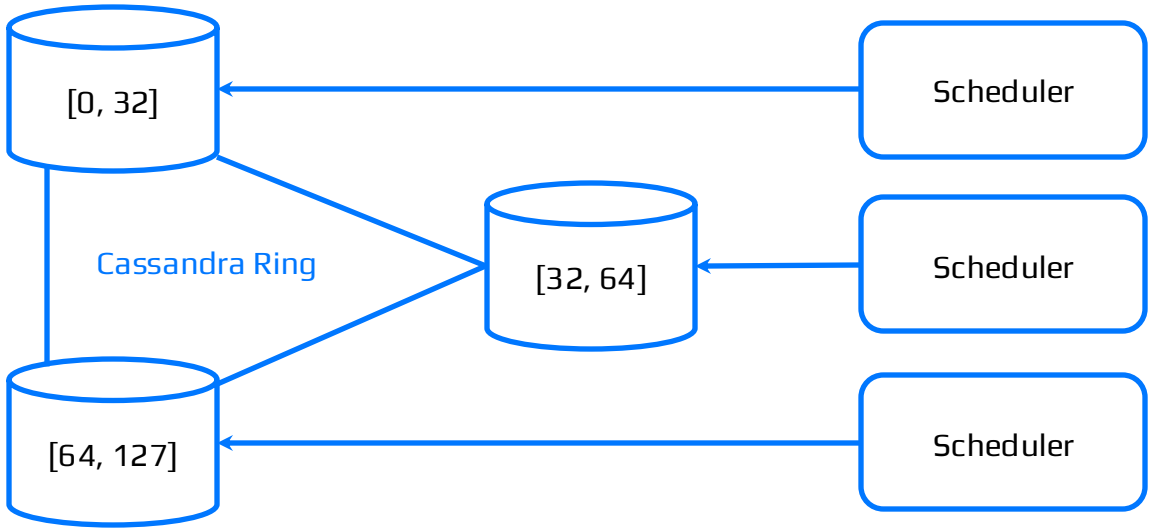
Фоновые задачи: скан

Шаг 1

Запрос ренжа версий → CQL-запрос

- Данные в Кассандре шардированы на основании токена
- Токен вычисляется как хэш от Partition Key
- Ренжи токенов равномерно разделены между шедулерами — шедулеры не мешают друг другу





Фоновые задачи: фильтр

Шаг 2

Фильтр на соответствие правилу

После запроса на шедулере
уже есть список версий в памяти

** Условия могут быть и более хитрые (NewerNoncurrentVersions)*

```
"NoncurrentVersionExpiration": {
    "NewerNoncurrentVersions": 2
}
```

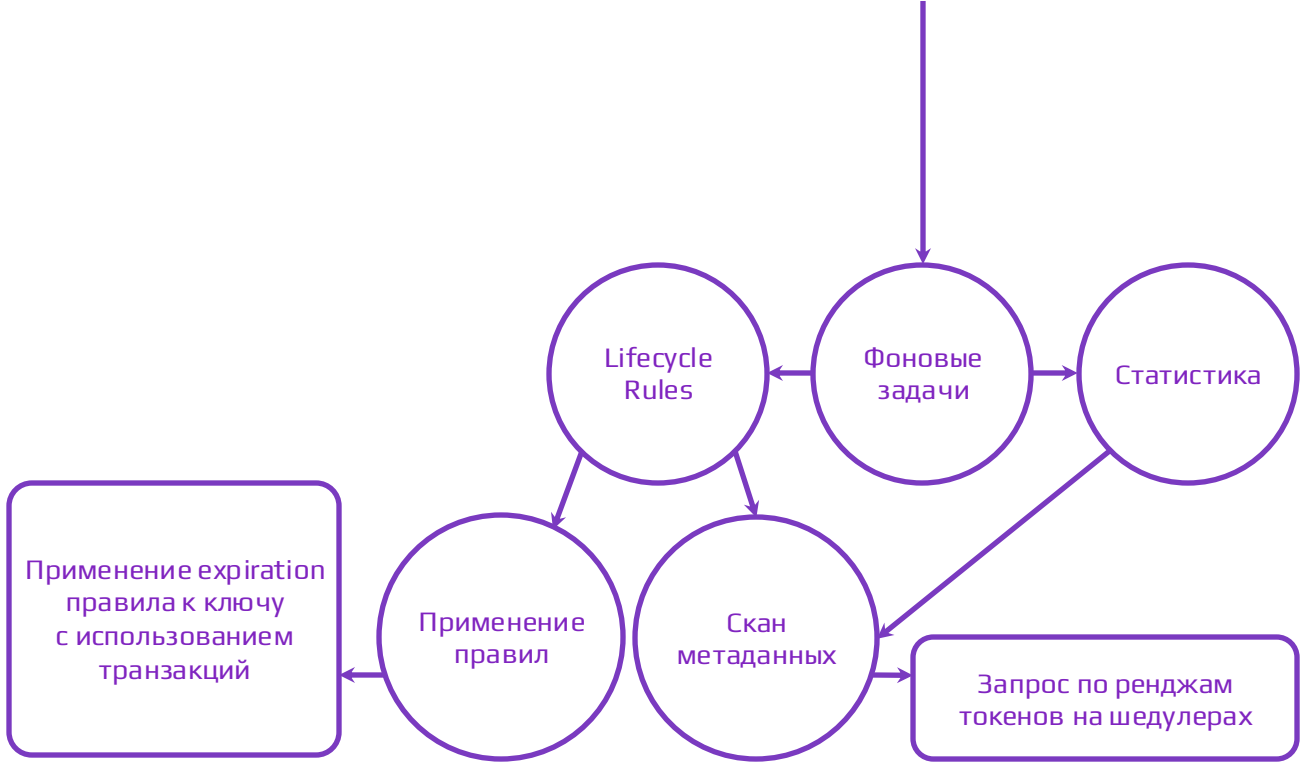
Version	created_at
8bc88735-ae75-4011-af23-4dc7f518b7ba	2026-02-02
15f44bca-272d-4b33-9bd3-5f79f2630bc4	2026-02-10
d7de823a-e668-4a75-913b-57ceb415922d	2026-01-01

Фоновые задачи:
применение правил

Шаг 3

Удаление версии

Правила применяются
на шедулере с использованием
транзакций

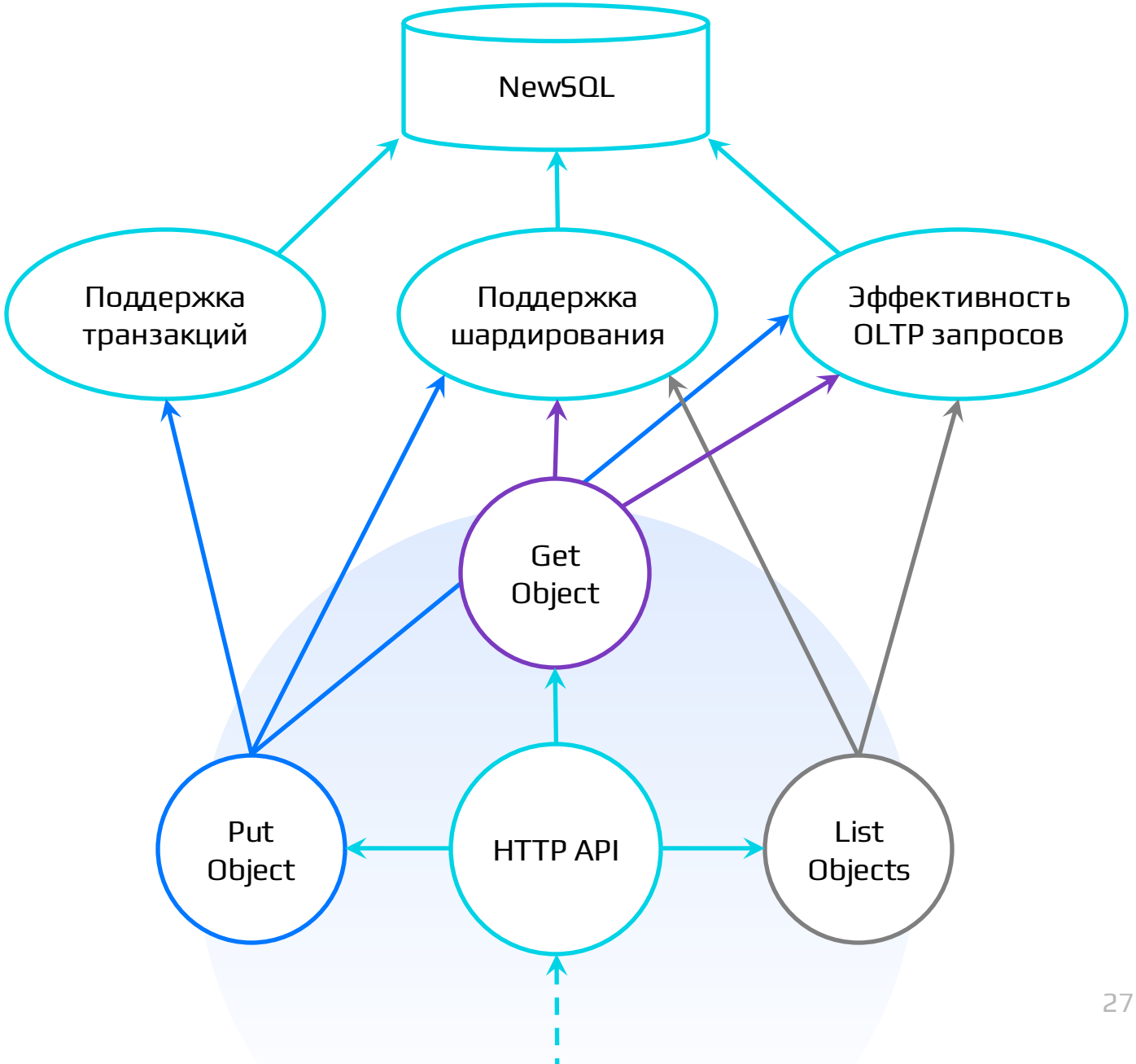


Version	created_at
8bc88735-ae75-4011-af23-4dc7f518b7ba	2026-02-02T00:00:00.000Z
15f44bca-272d-4b33-9bd3-5f79f2630bc4	2026-02-10T00:00:00.000Z

Резюме: метастор для HTTP API

Метастор S3 должен уметь:

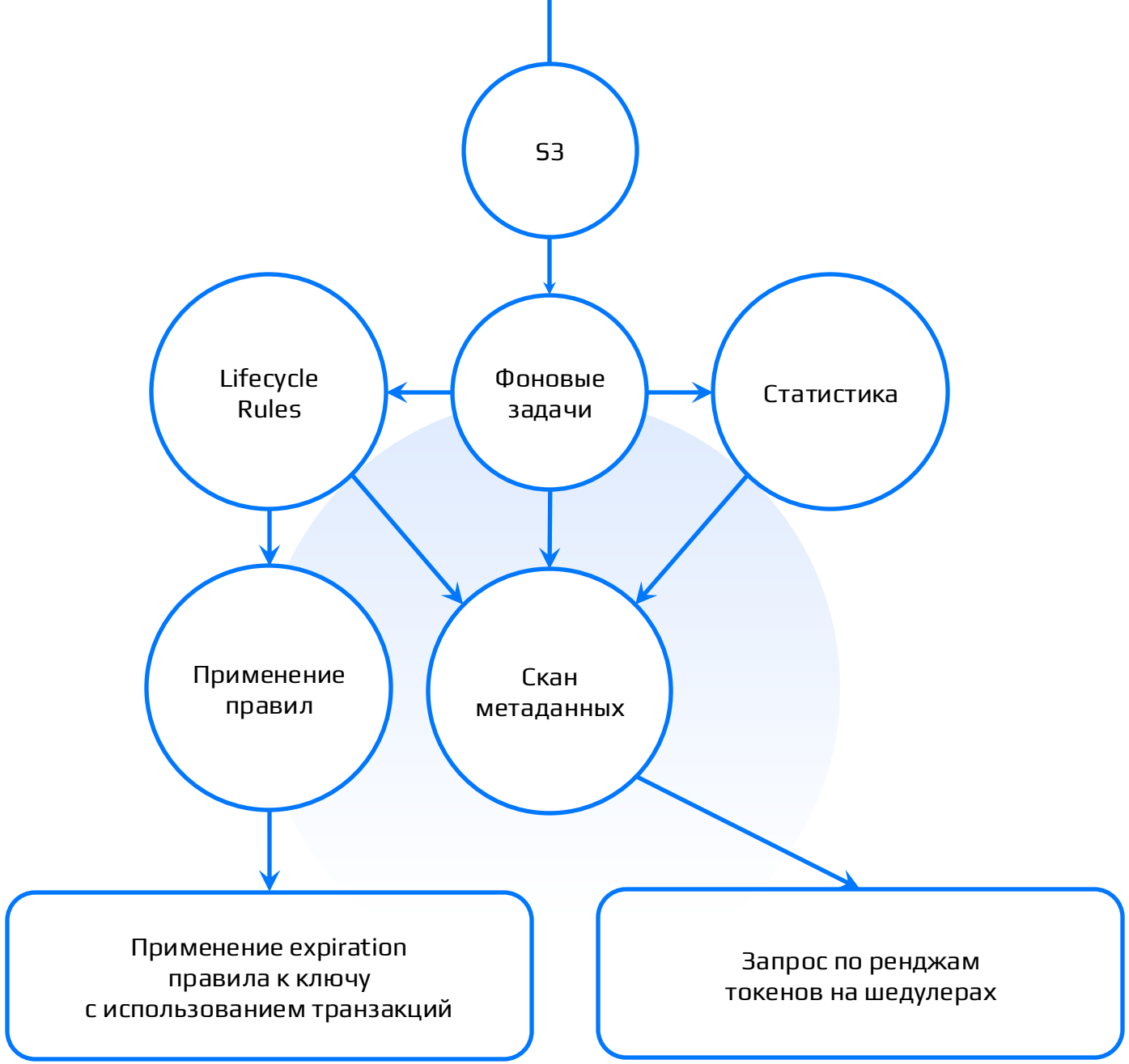
- Эффективно отдавать данные для клиентских HTTP-запросов



Резюме: метастор для фоновых операций

Метастор S3 должен уметь:

- Выполнять lifecycle правила
- Собирать статистику (для биллинга и мониторинга)



Почему Lifecycle Rules — это важно?

В S3 хранится огромное количество логов, дампов, бэкапов, кэшей и.т.д.

Почему Lifecycle Rules — это важно?

В S3 хранится огромное количество логов, дампов, бэкапов, кэшей и.т.д.

Клиенты редко удаляют по номеру версии

Почему Lifecycle Rules — это важно?

В S3 хранится огромное количество логов, дампов, бэкапов, кэшей и.т.д.

Клиенты редко удаляют по номеру версии

Массовое удаление через S3 API менее оптимизировано

Почему Lifecycle Rules — это важно?

На практике, Lifecycle Rules — основной способ освободить место в бакете

В S3 хранится огромное количество логов, дампов, бэкапов, кэшей и.т.д.

Клиенты редко удаляют по номеру версии

Массовое удаление через S3 API менее оптимизировано

Что такое tombstone в Cassandra

Данные в Cassandra хранятся в отсортированных неизменяемых таблицах

Что такое tombstone в Cassandra

- Данные в Cassandra хранятся в отсортированных неизменяемых таблицах

- **Удаление — это запись с особой пометкой**

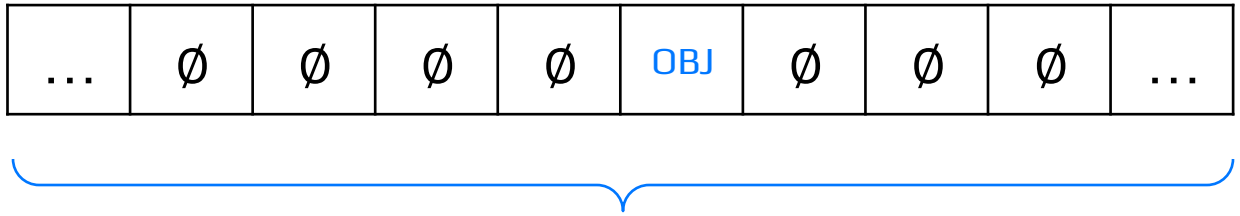
Что такое tombstone в Cassandra

- Данные в Cassandra хранятся в отсортированных неизменяемых таблицах
- Удаление — это запись с особой пометкой
- **Такая запись называется tombstone, или могила**

Что такое tombstone в Cassandra

- Данные в Cassandra хранятся в отсортированных неизменяемых таблицах
- Удаление — это запись с особой пометкой
- Такая запись называется tombstone, или могила
- Range запросы вычитывают могилы наравне с другими данными и фильтрует

SELECT * FROM cf WHERE k >= 0 and k < 1000



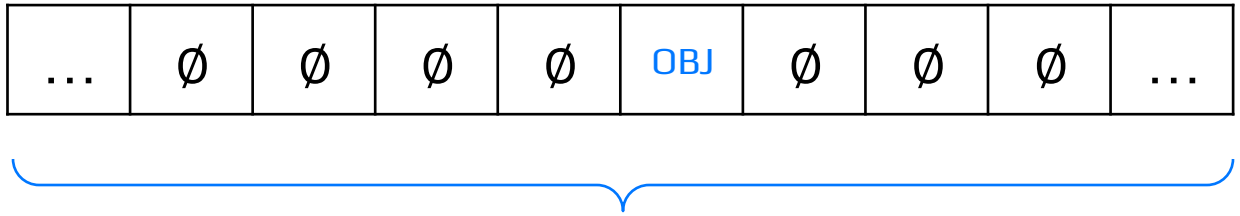
10 живых записей в партиции

100000 могил

Что такое tombstone в Cassandra

- Данные в Cassandra хранятся в отсортированных неизменяемых таблицах
- Удаление — это запись с особой пометкой
- Такая запись называется tombstone, или могила
- Range запросы вычитывают могилы наравне с другими данными и фильтрует
- **Большое количество могил в одной партиции приводит к зависанию range запросов**

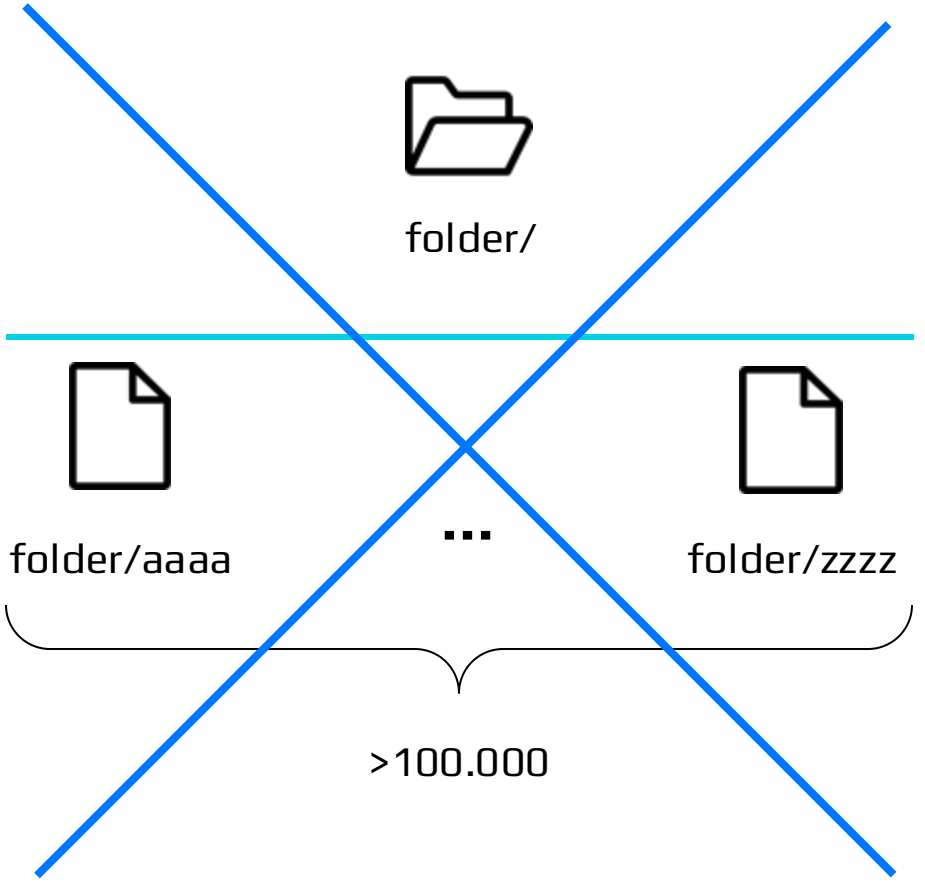
SELECT * FROM cf WHERE k >= 0 and k < 1000



10 живых записей в партиции
100000 могил

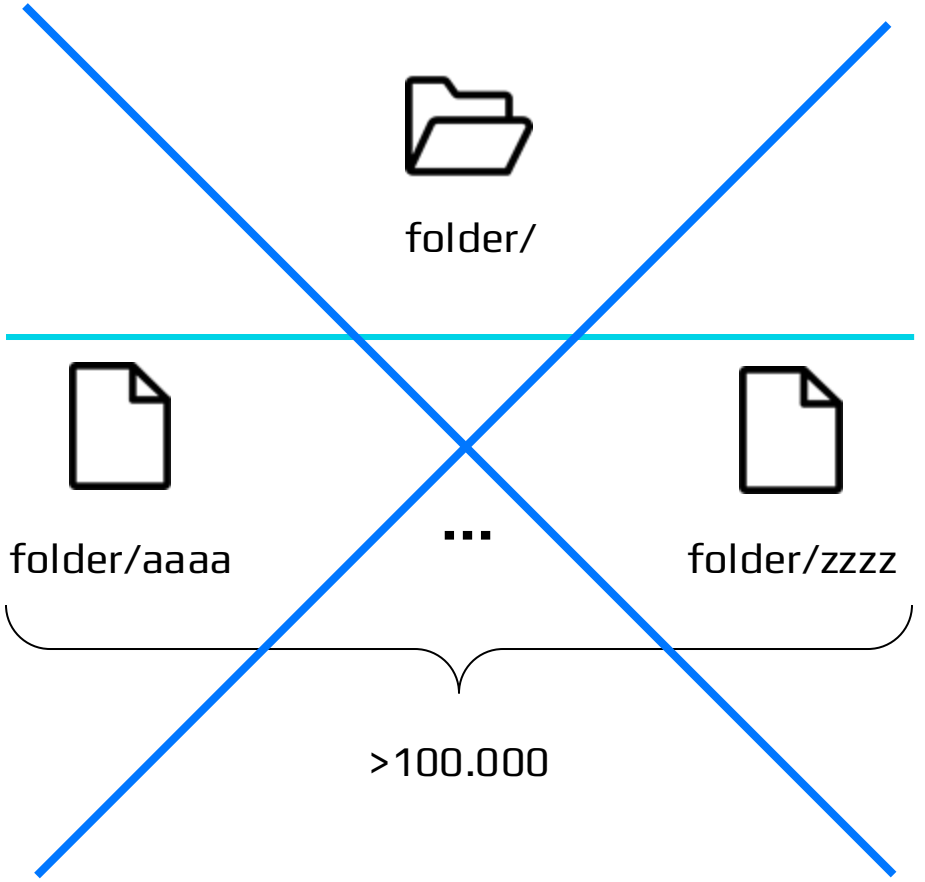
Ограничения one-object-storage

- **Для быстрой работы листинга, объекты партиционированы по родительской директории**
- **Одна директория = одна партиция**

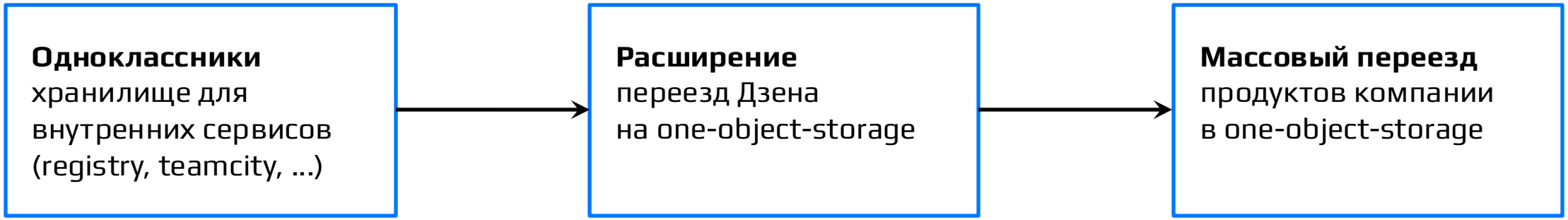


Ограничения one-object-storage

- Для быстрой работы листинга, объекты партиционированы по родительской директории
- Одна директория = одна партиция
- **Оптимально 1К объектов/версий/подкаталого в на каталог**



История использования one-object-storage



- Увеличение количества сценариев использования S3
- Невозможно требовать от клиентов соблюдения ограничения на количество объектов в директории
- Невозможно договариваться со всеми клиентами о «правильном» использовании

История использования one-object-storage

Одноклассники
хранилище для внутренних сервисов (registry, teamcity, ...)

→

Расширение
переезд Дзена на one-object-storage

→

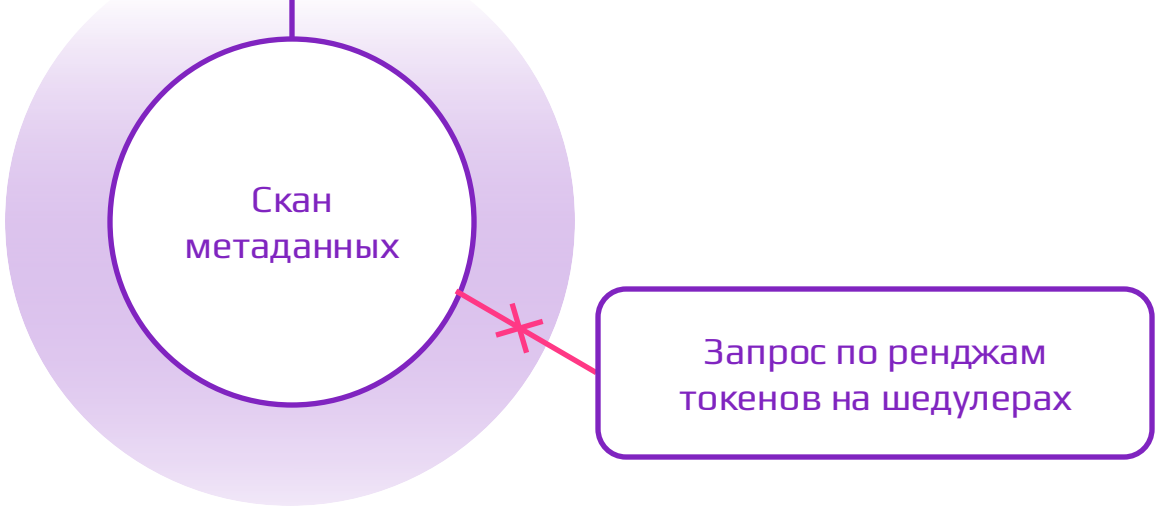
Массовый переезд
продуктов компании в one-object-storage

- Увеличение количества сценариев использования S3
- Невозможно требовать от клиентов соблюдения ограничения на количество объектов в директории
- Невозможно договариваться со всеми клиентами о «правильном» использовании

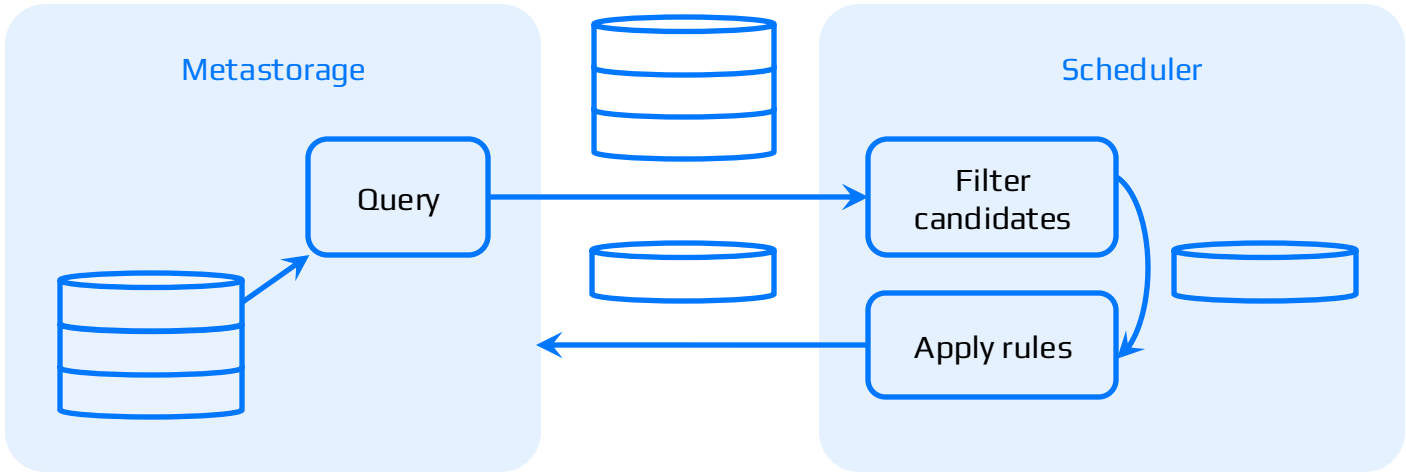
Результат: **неизбежное образование широких партиций**

43 бакета, где есть директории на **более чем 100к объектов**

Проблемы скана метаданных



- Скан использует range-запросы
- Из-за «неправильного» использования появляются широкие партиции
- В партициях копятся могилы —чистка встаёт
- **Необходимо быстро решить проблему**



Отказаться от Кассандры?

- Не будет проблем с могилами

Отказаться от Кассандры?

- Не будет проблем с могилами

- Нужно позаботиться о шардировании
- Риск потерять эффективность на HTTP запросах
- Быстро не получится

Отказаться от Кассандры?

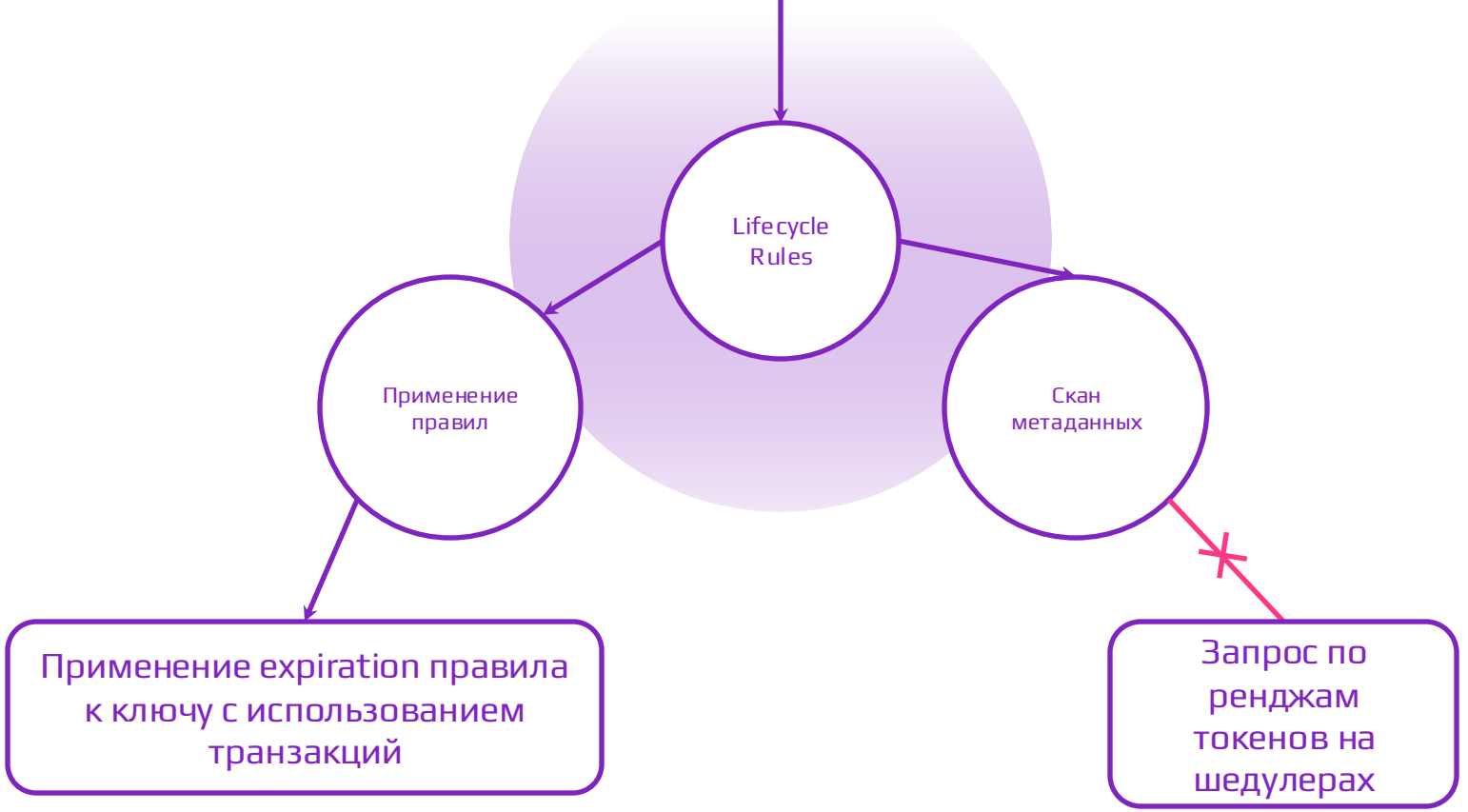
- Не будет проблем с могилами

- Нужно позаботиться о шардировании
- Риск потерять эффективность на HTTP запросах
- Быстро не получится

Вывод: пока не вариант

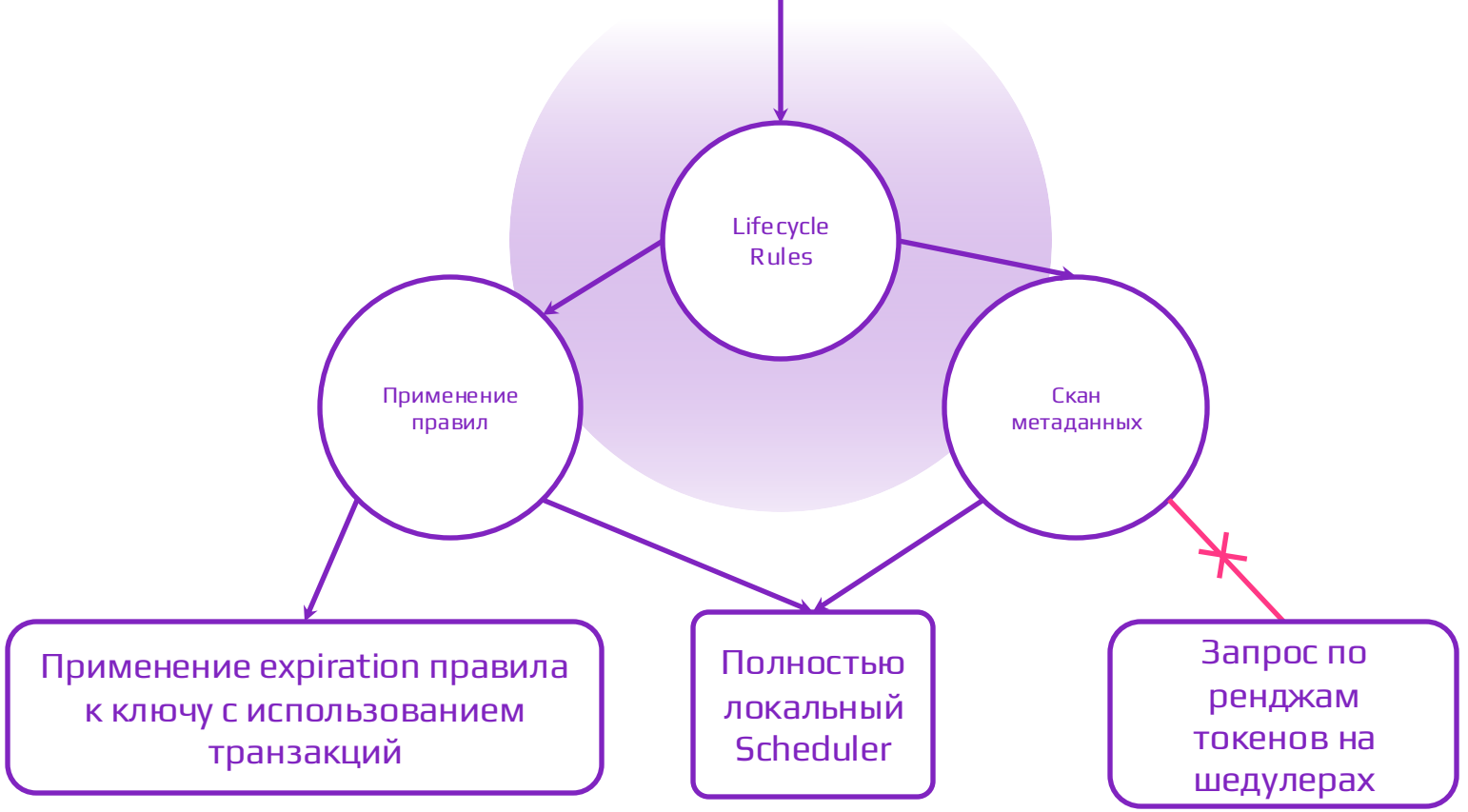
Идея

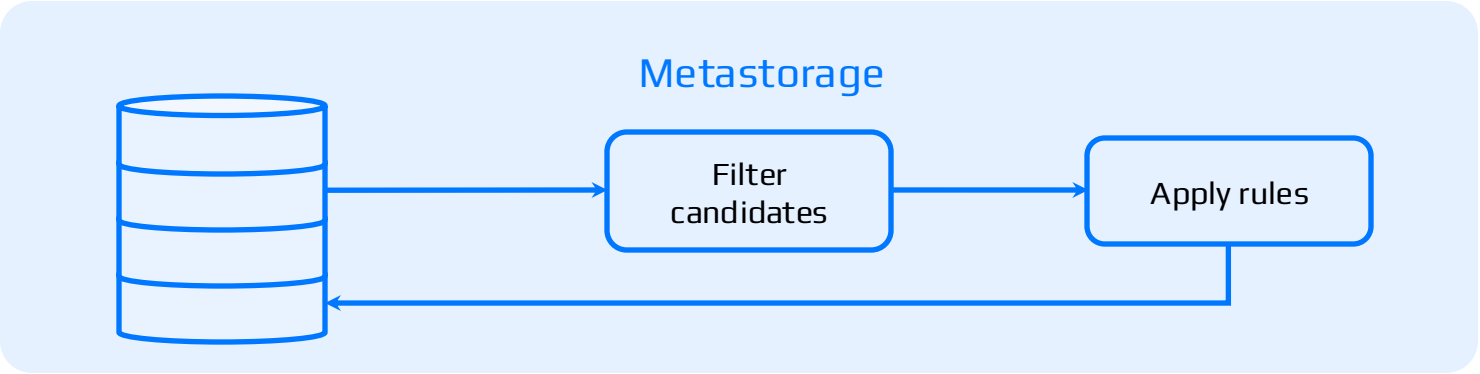
- Мы не можем запрашивать данные для скана извне из-за могил



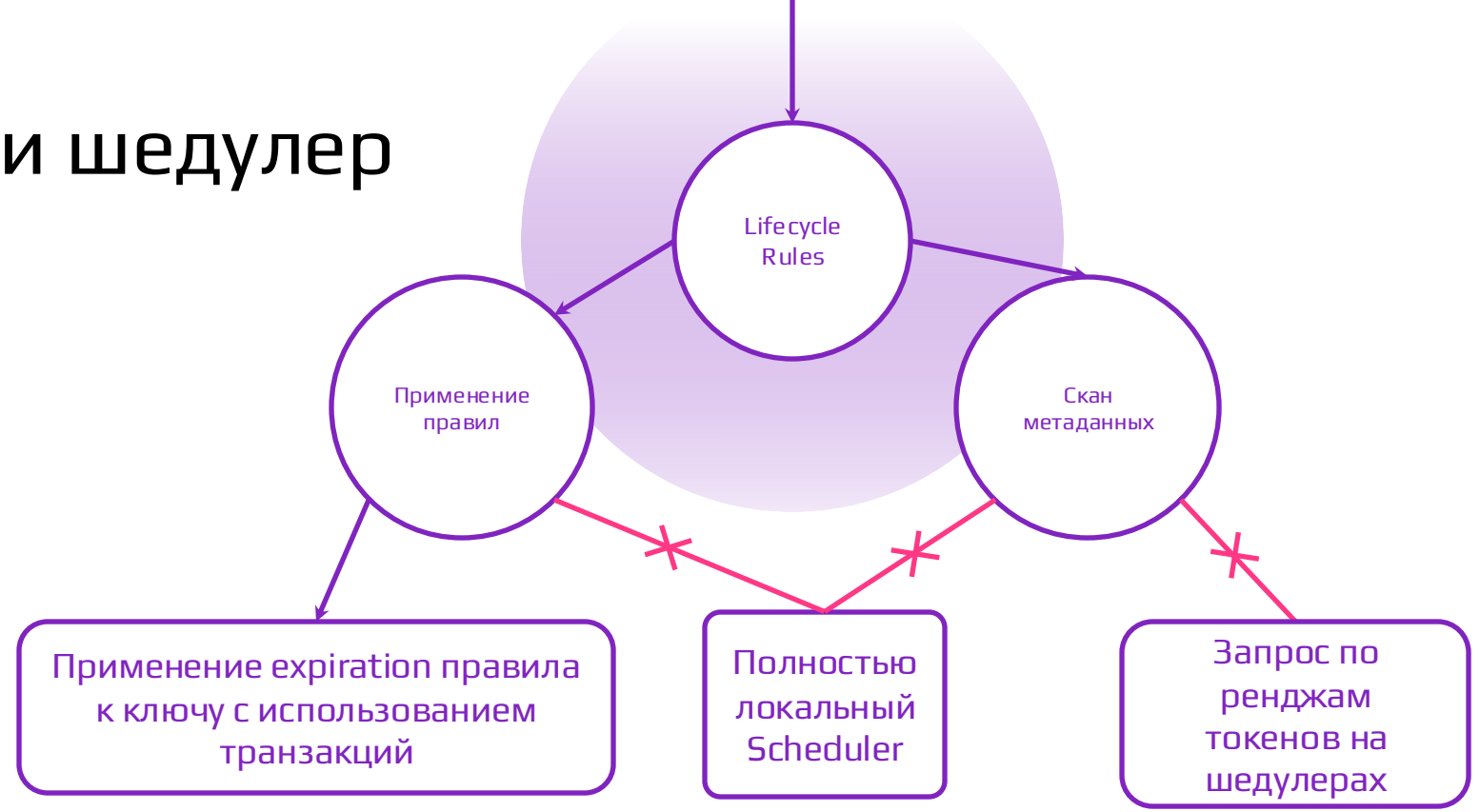
Идея

- Мы не можем запрашивать данные для скана извне из-за могил
- **Значит будем сканировать изнутри!**



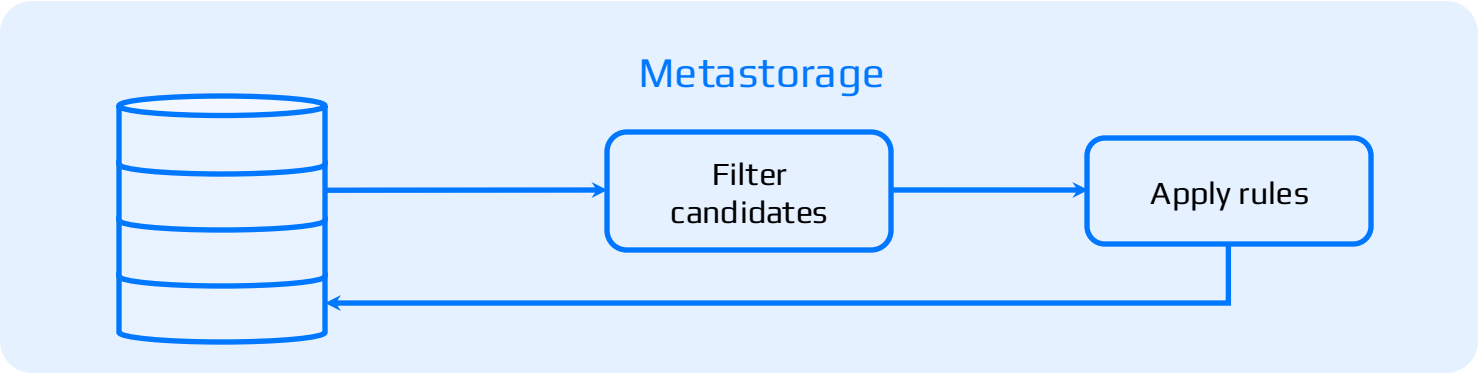


Полностью перенести шедулер в метастор?



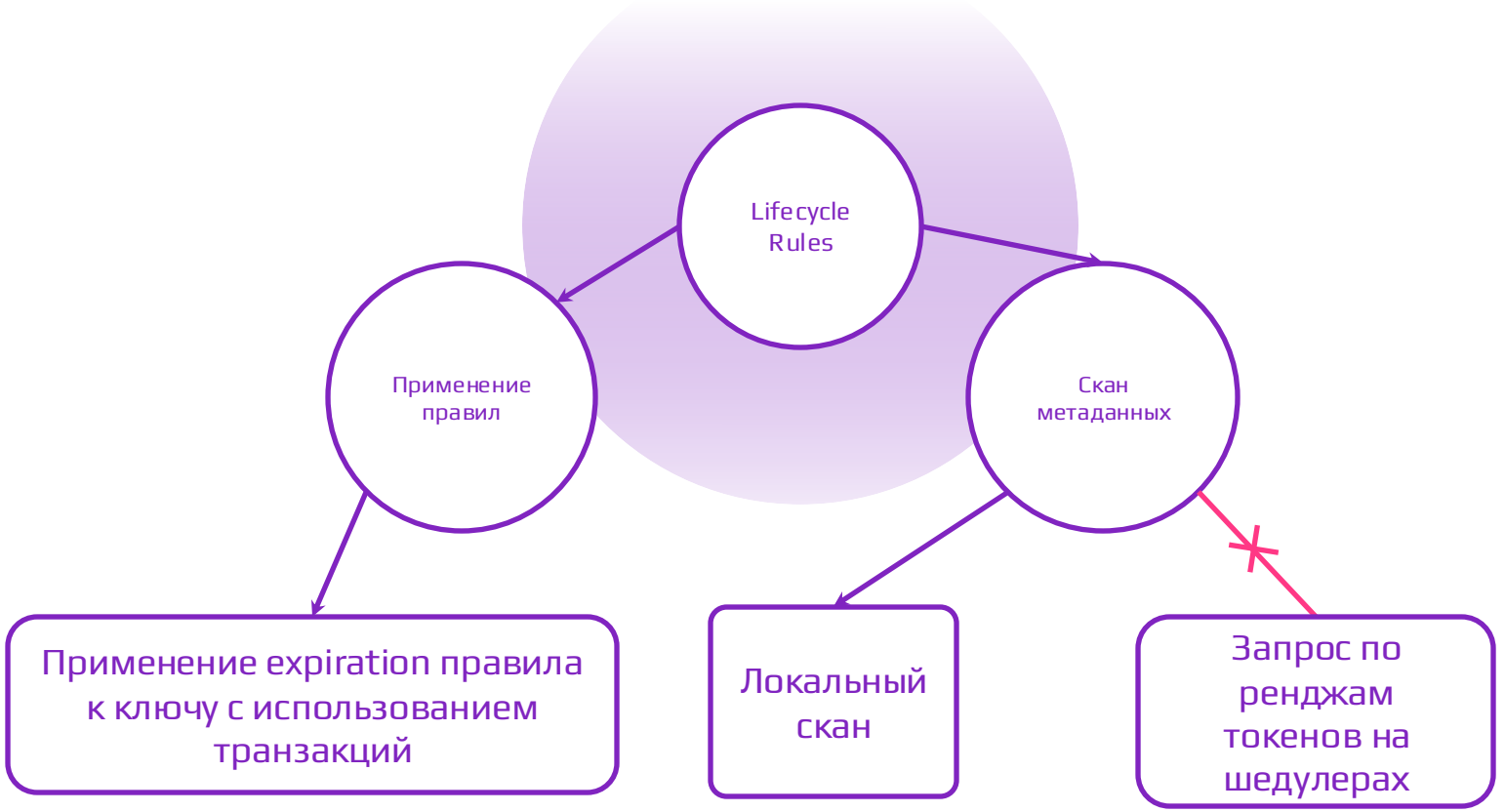
- Решается проблема с могилами

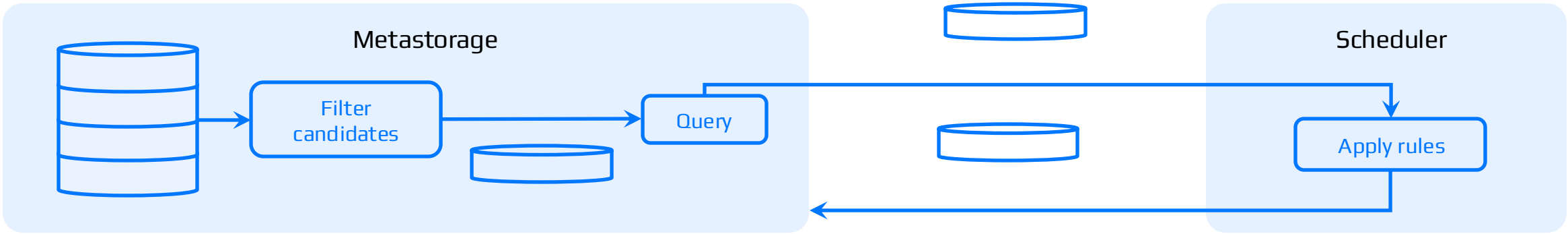
- Неконтролируемое потребление ресурсов
- Усложняет деплой
- Замедляет миграцию с NewSQL
- Непрозрачно



Компромисс

Фильтруем локально — применяем правила на шедулере





Проблема локального скана

Наша реализация S3 не ограничивает количество версий одного объекта

Вычитывание ~100k версий на метасторе стоит слишком много памяти

Проблема локального скана

Наша реализация S3 не ограничивает количество версий одного объекта

Вычитывание ~100k версий на метасторе стоит слишком много памяти

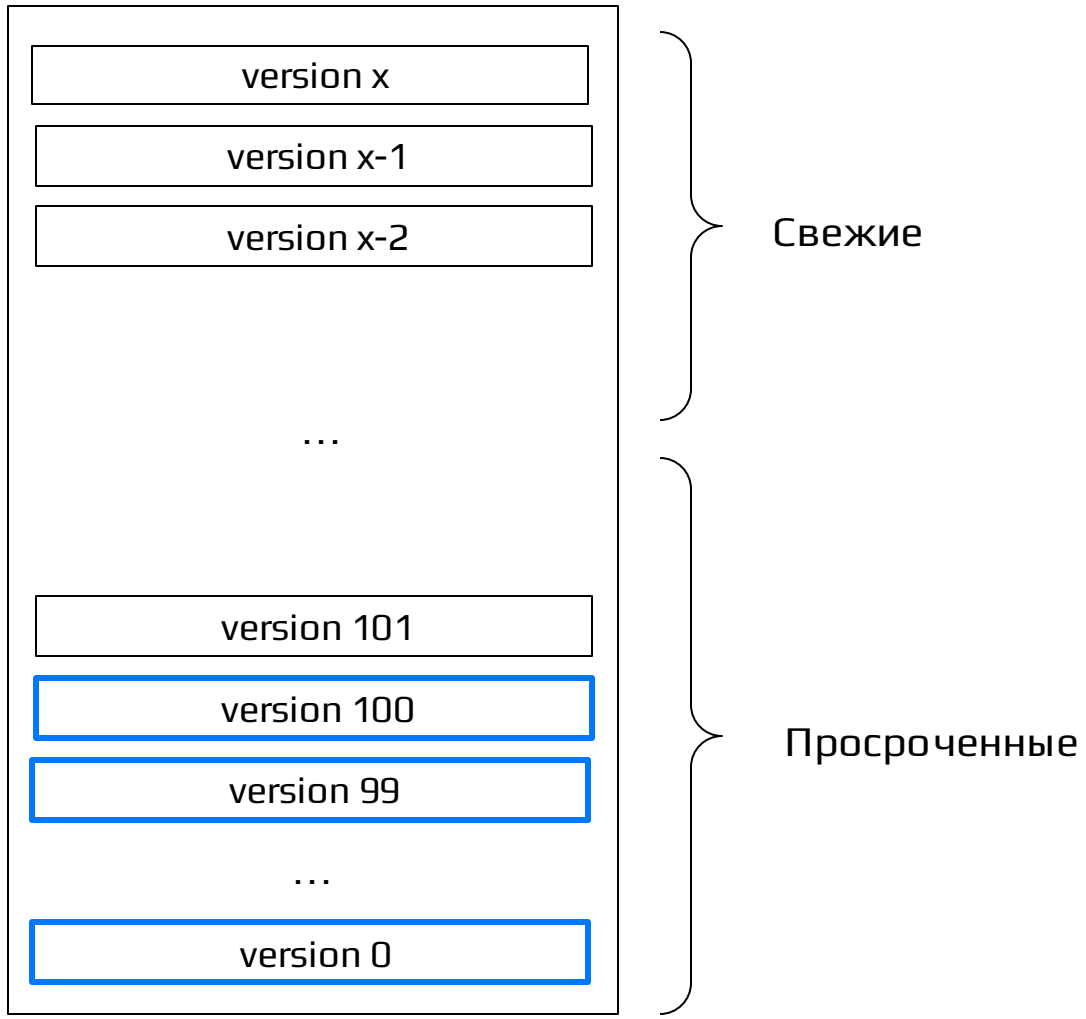
Нужна оптимизация

Оптимизированный скан

- **Для проверки при скане нам не нужны все версии объекта**
- **Достаточно хотя бы одной подходящей под правило**

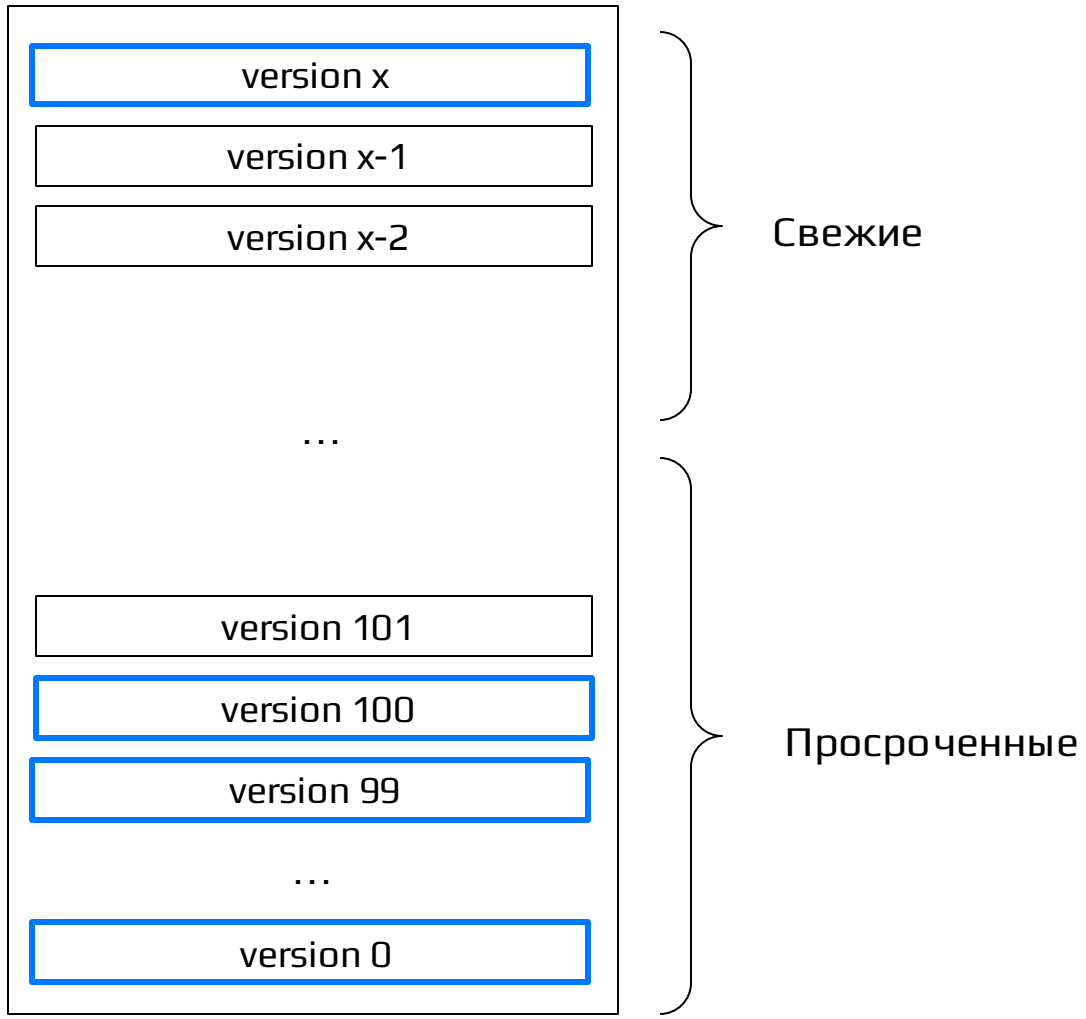
Оптимизированный скан

- Для проверки при скане нам не нужны все версии объекта
- Достаточно хотя бы одной подходящей под правило
- **Будем хранить 100 последних версий, чтобы определить, есть ли среди них просроченные**
- **Правило NewerNoncurrentVersions ограничено 100 версиями**



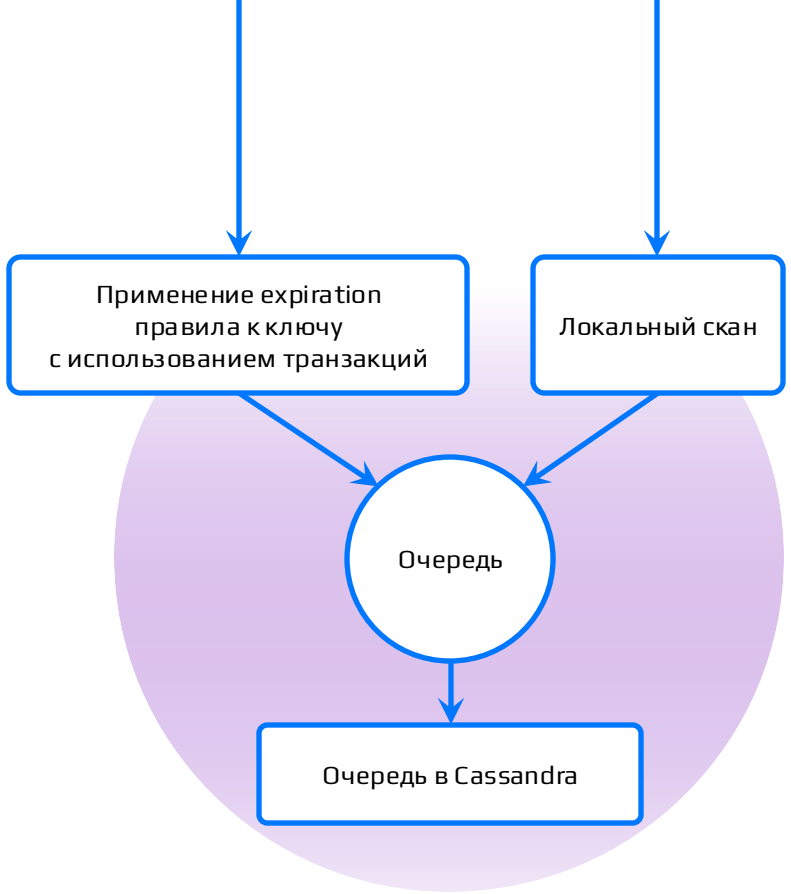
Оптимизированный скан

- Для проверки при скане нам не нужны все версии объекта
- Достаточно хотя бы одной подходящей под правило
- Будем хранить 100 последних версий, чтобы определить, есть ли среди них просроченные
- Правило NewerNoncurrentVersions ограничено 100 версиями
- **Будем хранить актуальную версию чтобы избежать ложно-позитивных экспираций**



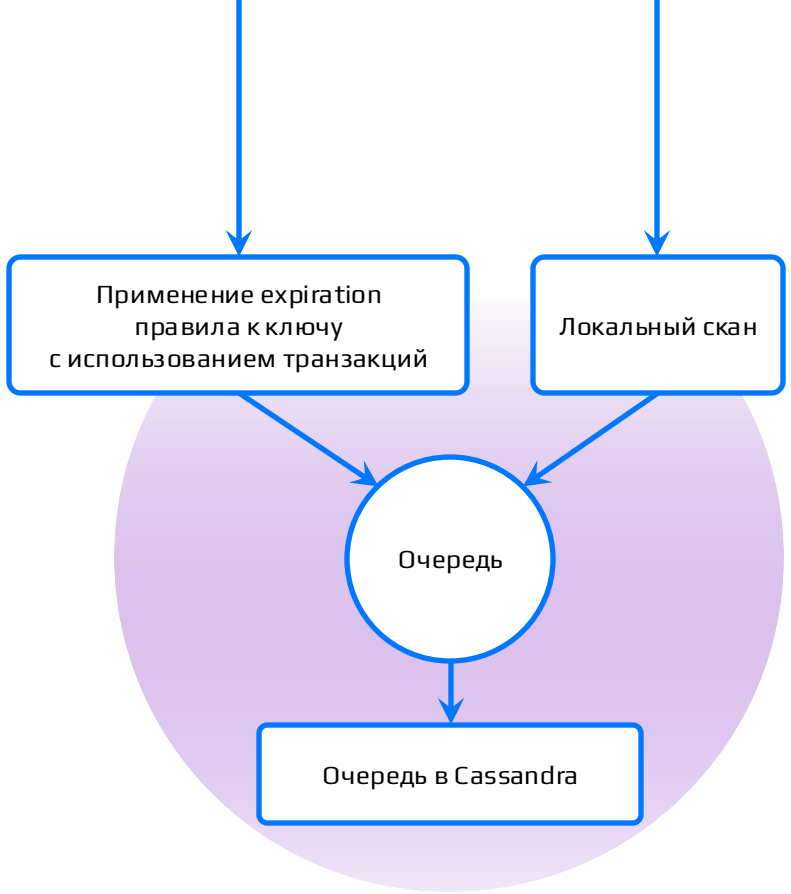
Общение с шедулером: очередь

- **Локальный скан подразумевает место, где будут храниться кандидаты**
- **Нам нужна очередь**



Общение с шедулером: очередь

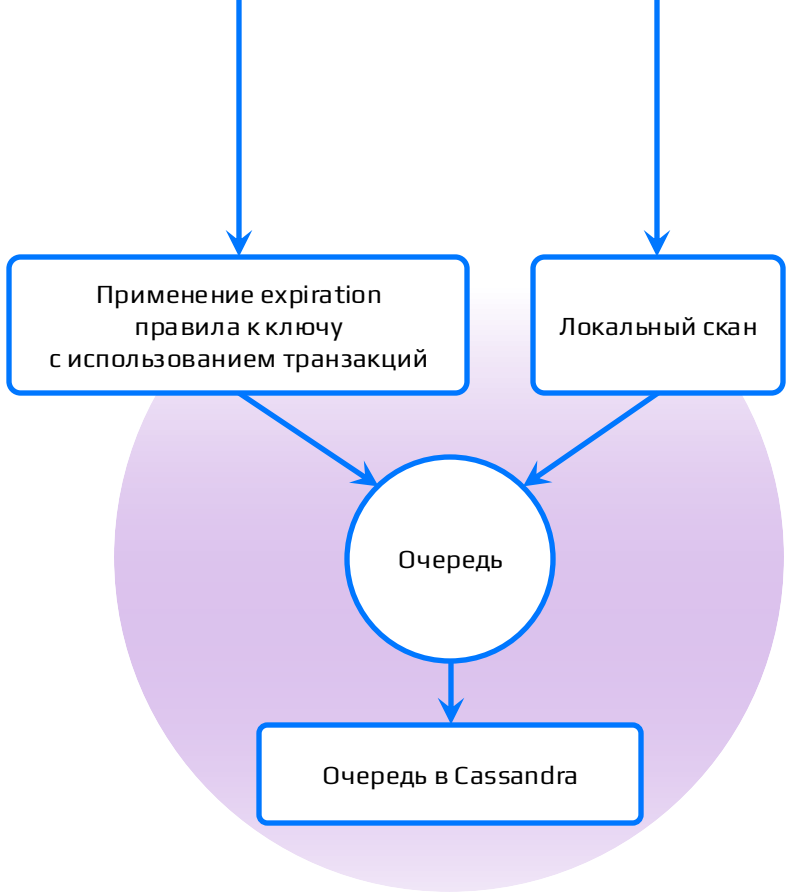
- Локальный скан подразумевает место, где будут храниться кандидаты
- Нам нужна очередь
- **Сделаем её с помощью таблицы в Кассандре**
- **Для масштабирования поддержим партиции**



part	num	value
0	100	files/temp1
1	120	files/temp2
2	90	files/temp3

Общение с шедулером: очередь

- Локальный скан подразумевает место, где будут храниться кандидаты
- Нам нужна очередь
- Сделаем её с помощью таблицы в Кассандре
- Для масштабирования поддержим партиции
- **Будем хранить offset чтобы можно было возобновлять работу**

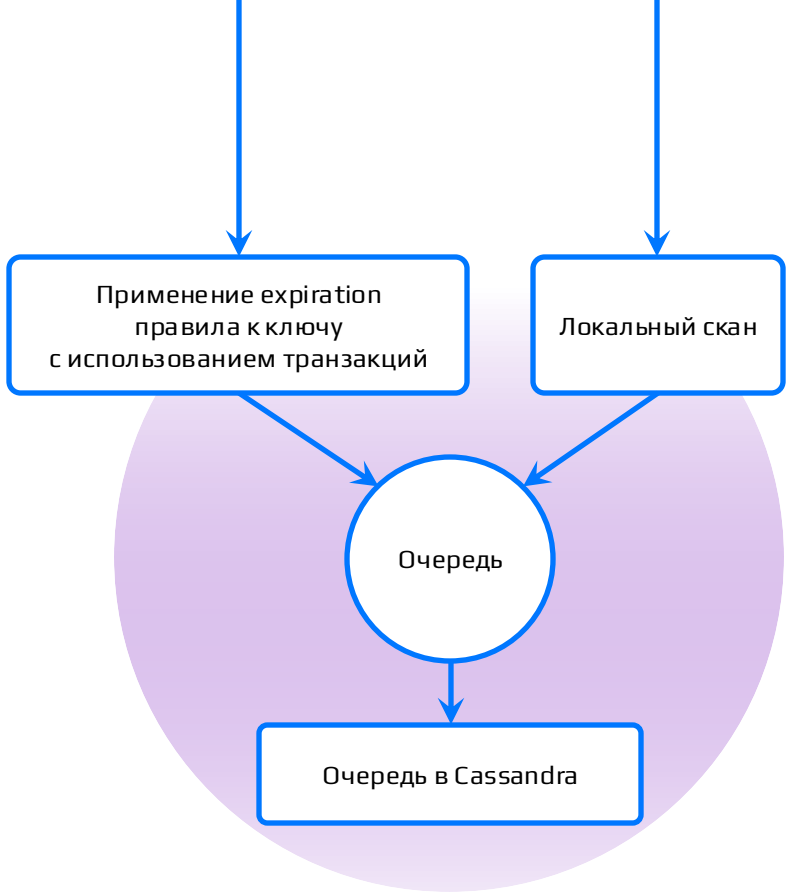


part	num	value
0	100	files/temp1
1	120	files/temp2
2	90	files/temp3

part	reader	offset
0	1.scheduler.s3-infra-prod.pc	96
1	1.scheduler.s3-infra-prod.hc	120
2	1.scheduler.s3-infra-prod.sc	80

Общение с шедулером: очередь

- Локальный скан подразумевает место, где будут храниться кандидаты
- Нам нужна очередь
- Сделаем её с помощью таблицы в Кассандре
- Для масштабирования поддержим партиции
- Будем хранить offset чтобы можно было возобновлять работу
- **Удаление только по ttl**
- **gc_grace_seconds=0**

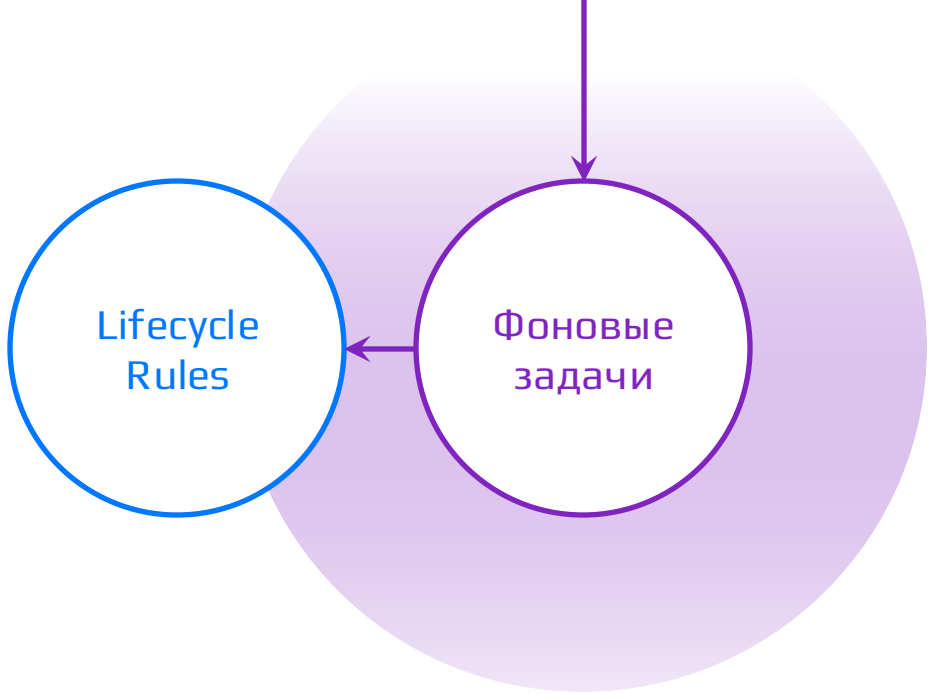


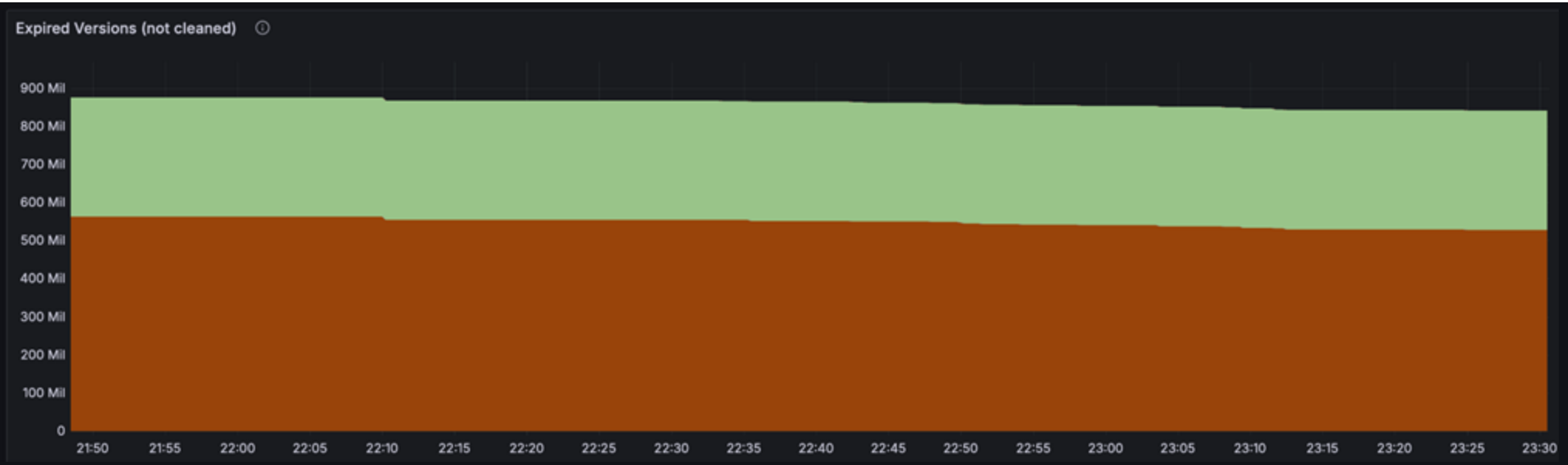
part	num	value
0	100	files/temp1
1	120	files/temp2
2	90	files/temp3

part	reader	offset
0	1.scheduler.s3-infra-prod.pc	96
1	1.scheduler.s3-infra-prod.hc	120
2	1.scheduler.s3-infra-prod.sc	80

Метрика на количество просроченных версий

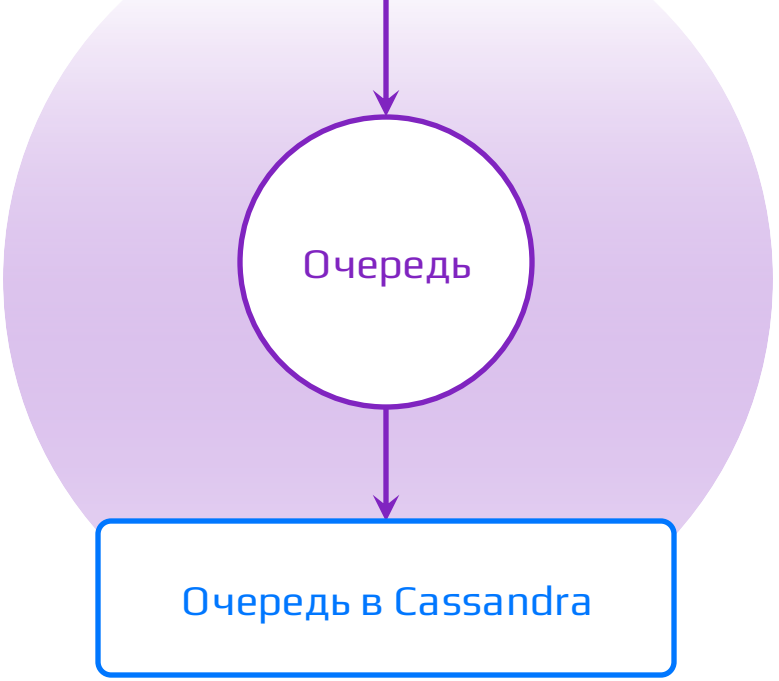
- Сделали метрику с количеством просроченных версий
- Собирается при локальном скане
- Что-то идёт не так





Таймауты на poll очереди

Массовые таймауты
на запрос ключей из очереди



Логи шедулера:

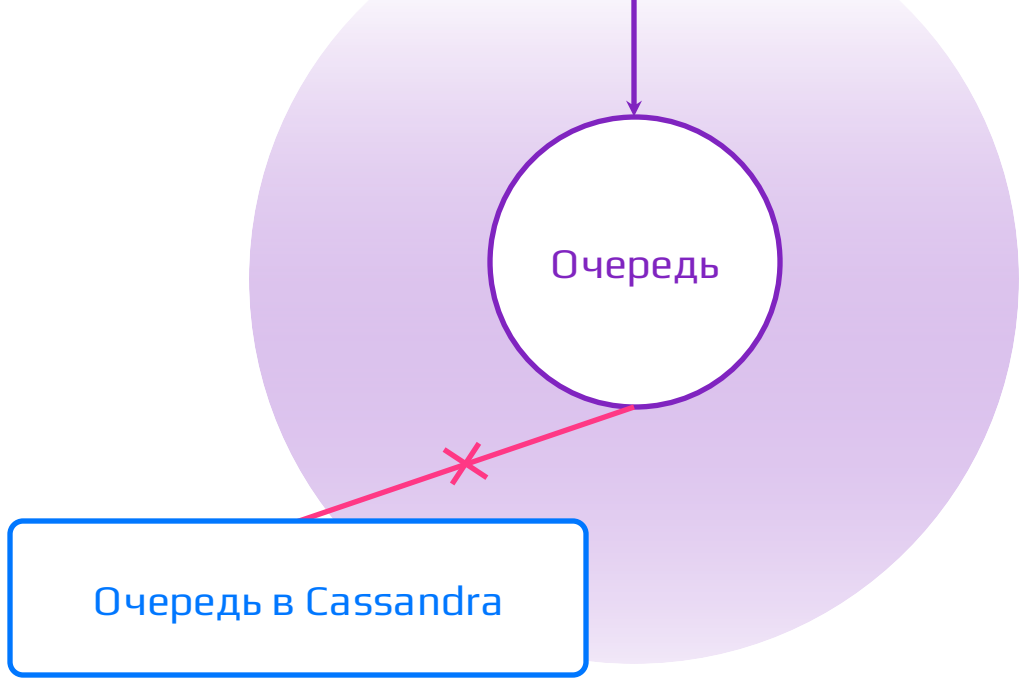
```
one.ejb.dao.DAOSysException: Timeout executing [SELECT n FROM version_meta_queue WHERE p=? ORDER BY n DESC LIMIT 1], bound: [SELECT n FROM version_meta_queue WHERE
db.newsql-s3-mail-misc.kc.one-infra.ru-1 ORDER BY n DESC LIMIT 1], TX ID= null, Request ID= e8123141-09ab-11f1-8c69-1dc4f0d198c9 : org.apache.cassandra.exceptions.
eoutException: Operation timed out - received only 1 responses of required 2.
```

Логи Cassandra:

```
16:49:51,840 ERROR org.apache.cassandra.db.filter.SliceQueryFilter Scanned over 250000 tombstones in cloud_av_cache.version_meta_queue; query aborted (see tombstone_fail
_threshold)
```

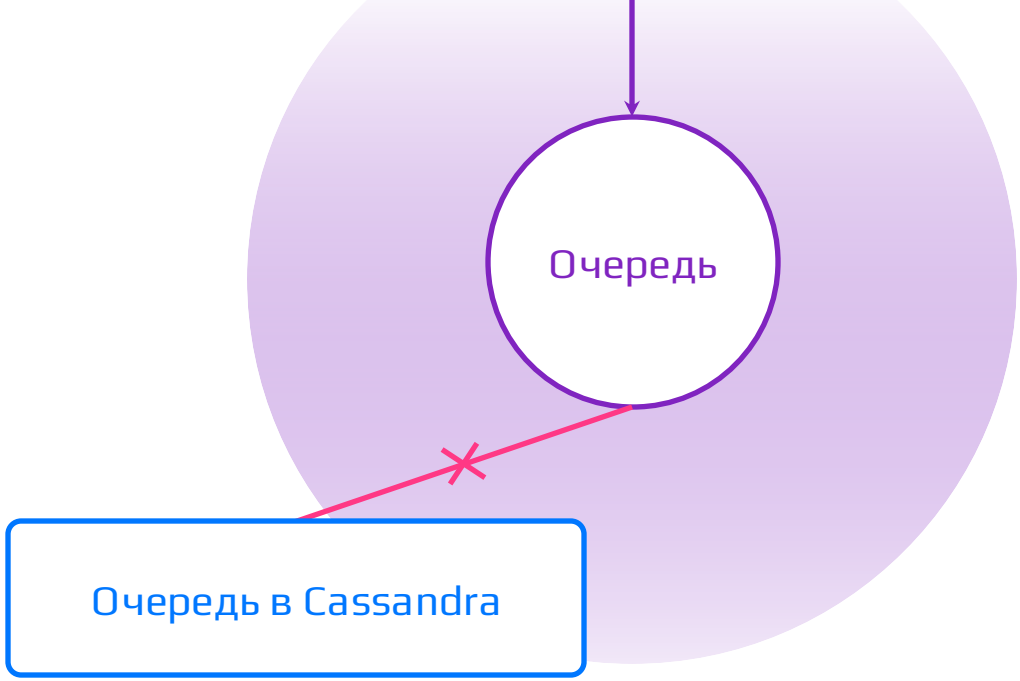
Почему очередь на Cassandra — плохая идея

- **Очередь подразумевает большое количество удалений и range запросов**



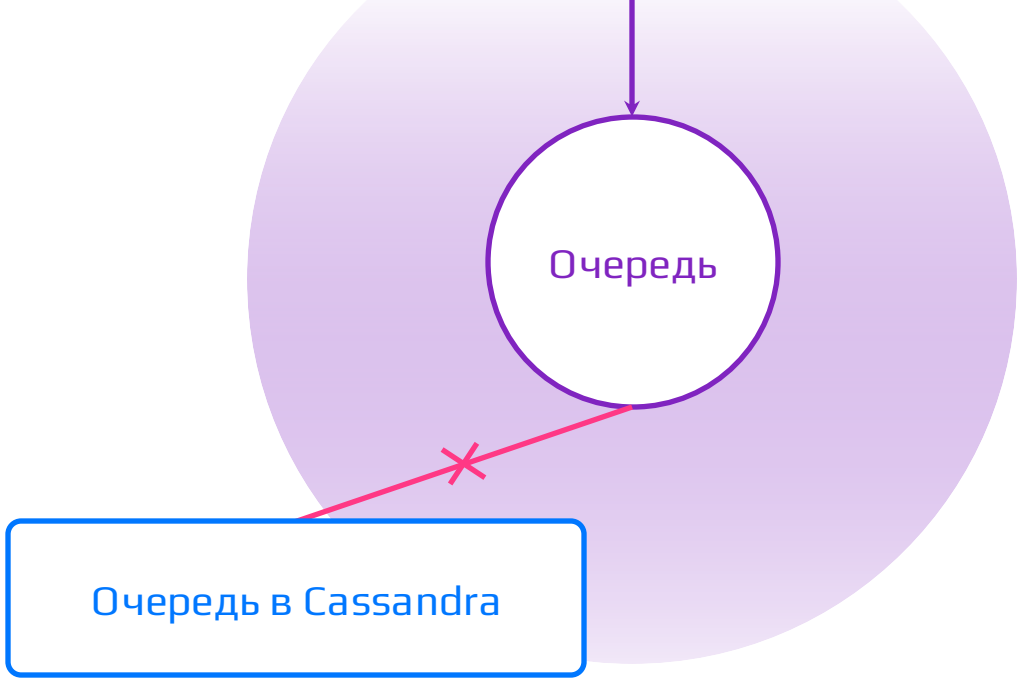
Почему очередь на Cassandra — плохая идея

- Очередь подразумевает большое количество удалений и range запросов
- **Из-за удалений создается большое количество могил**



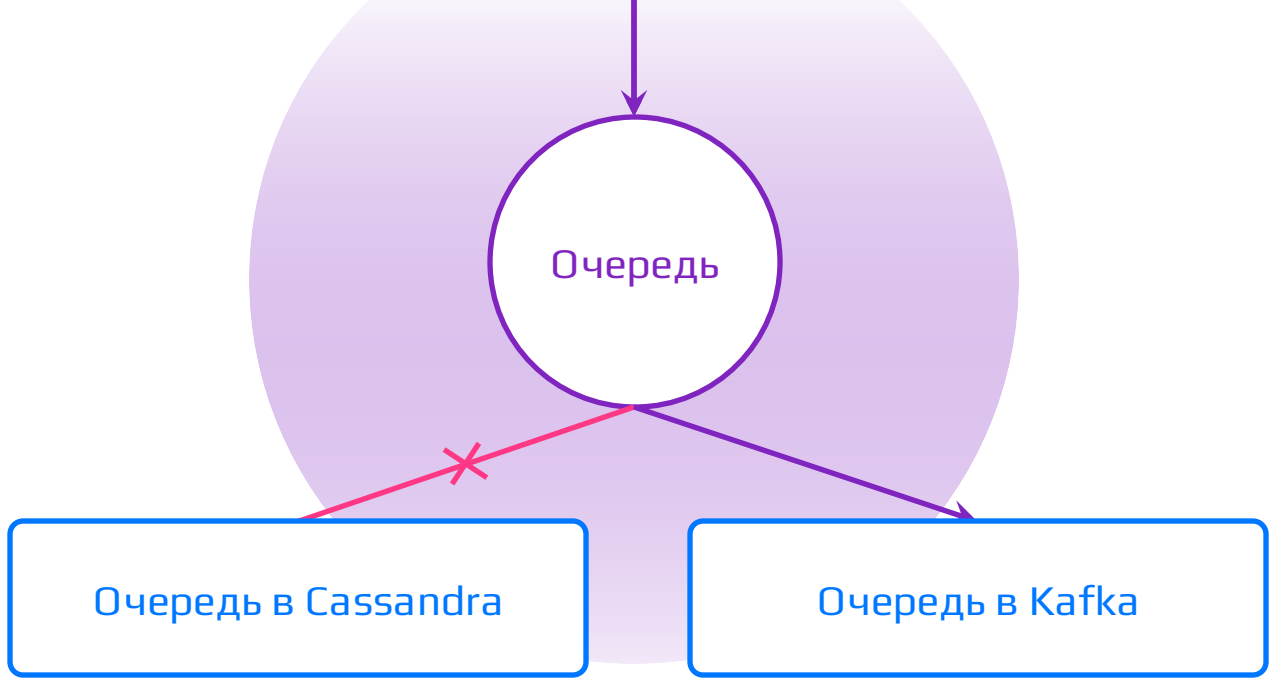
Почему очередь на Cassandra — плохая идея

- Очередь подразумевает большое количество удалений и range запросов
- Из-за удалений создается большое количество могил
- **Range-запросы падают из-за лимита на прочитанные могилы**



Почему очередь на Cassandra — плохая идея

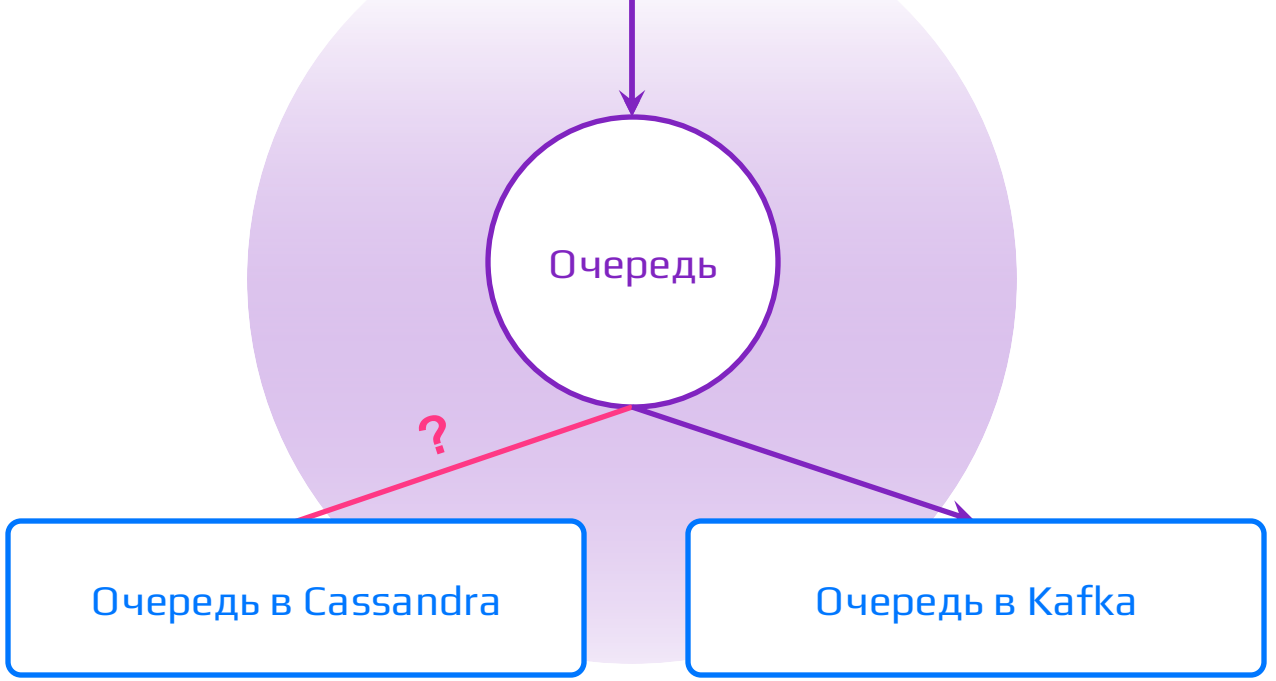
- Очередь подразумевает большое количество удалений и range запросов
- Из-за удалений создается большое количество могил
- Range-запросы падают из-за лимита на прочитанные могилы



Простое решение — **перенести очередь на Kafka**

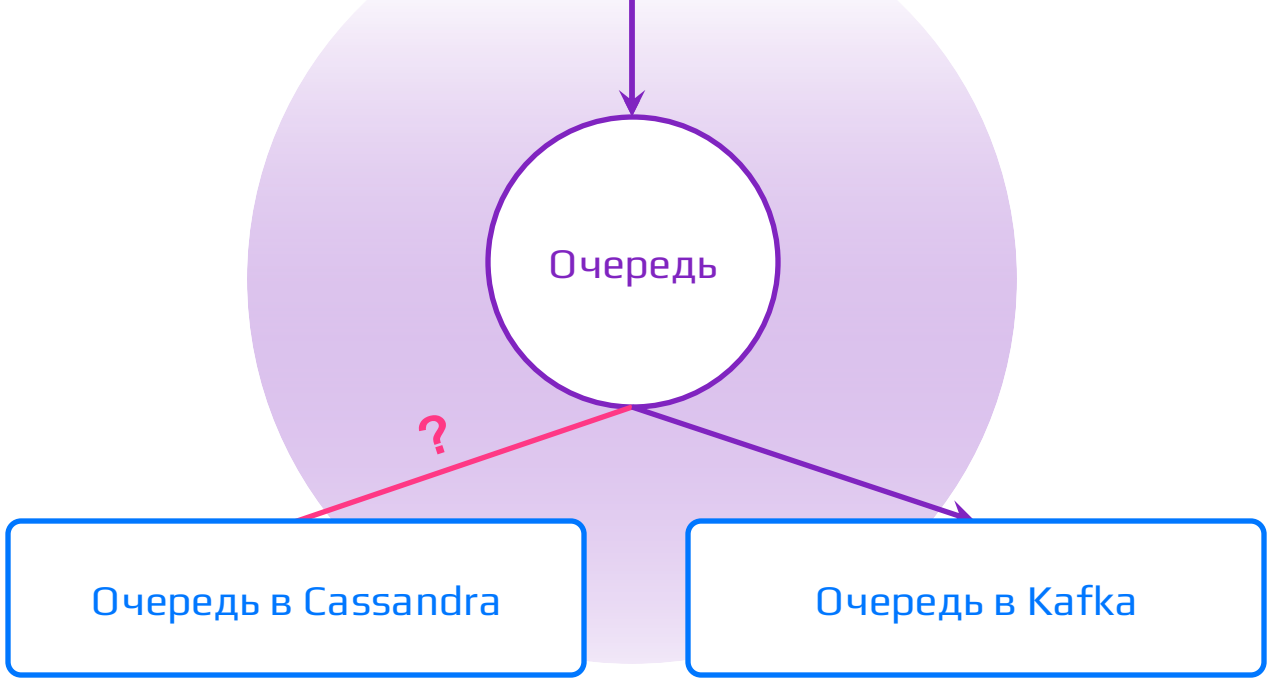
Почему очередь на Cassandra — хорошая идея (иногда)

- **Очередь на Cassandra позволила быстро мигрировать на метастор (1 кластер = 1 день)**



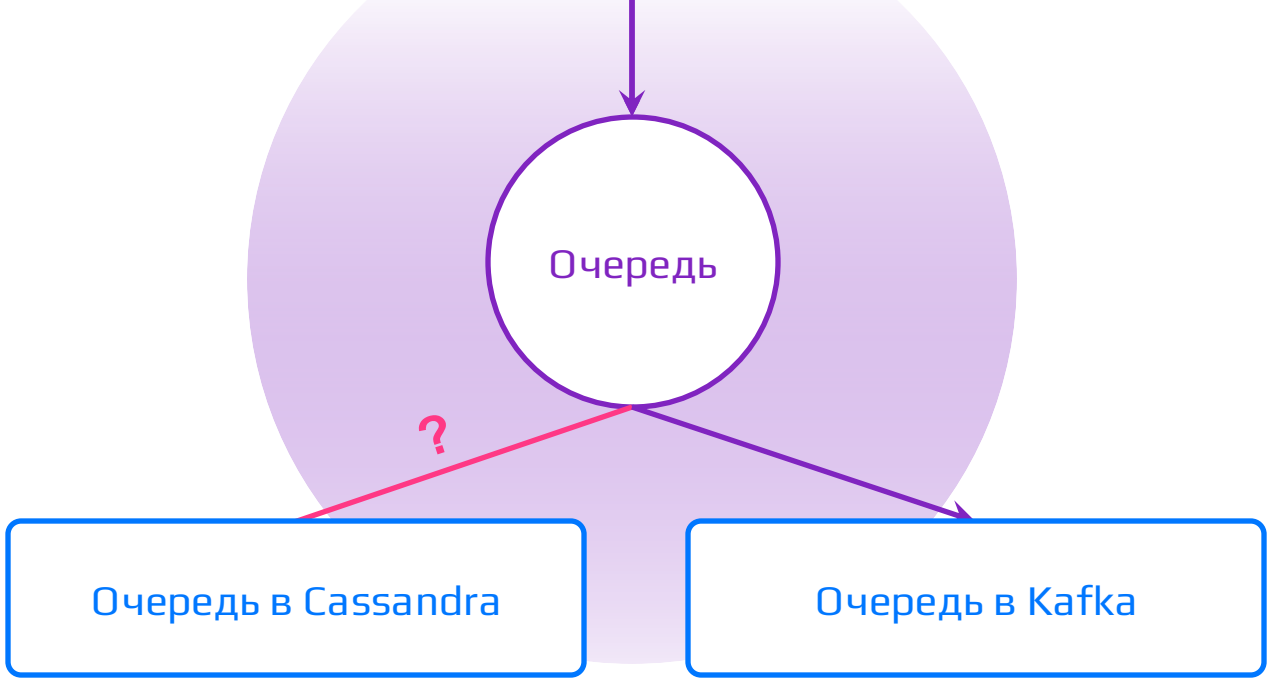
Почему очередь на Cassandra — хорошая идея (иногда)

- Очередь на Cassandra позволила быстро мигрировать на метастор (1 кластер = 1 день)
- **С небольшими объёмами она справлялась**



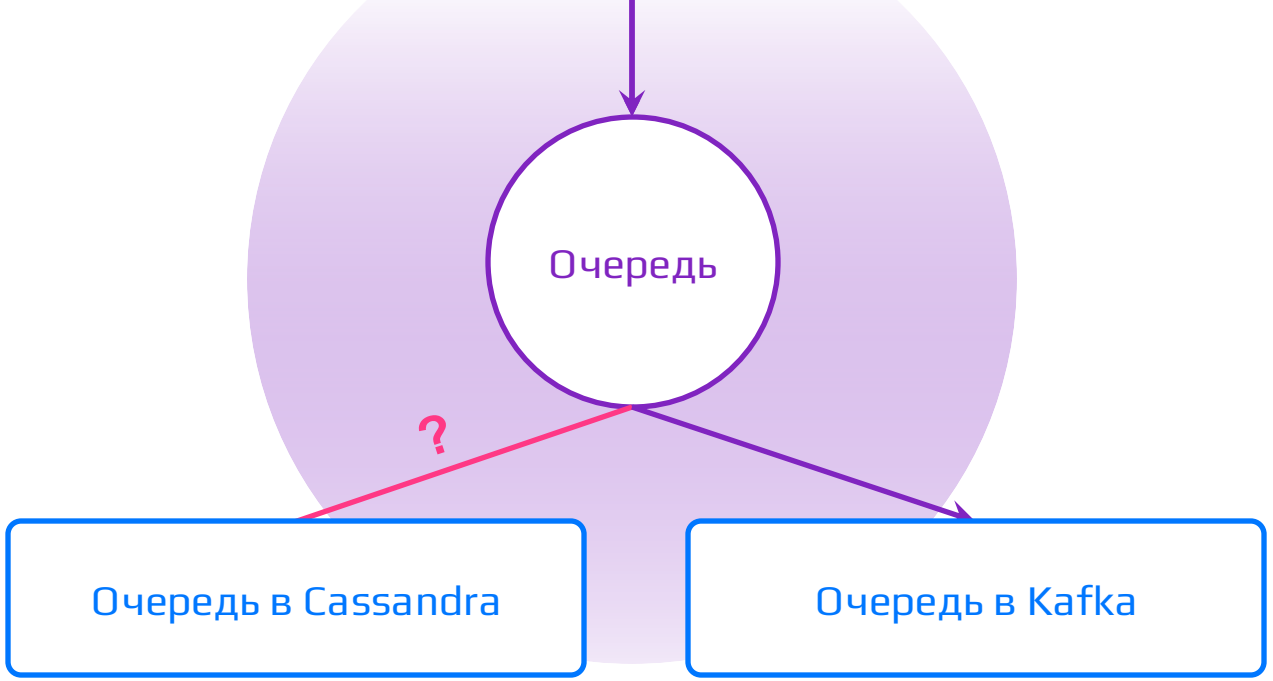
Почему очередь на Cassandra — хорошая идея (иногда)

- Очередь на Cassandra позволила быстро мигрировать на метастор (1 кластер = 1 день)
- С небольшими объёмами она справлялась
- **С этим можно жить (truncate таблиц с очередью, троттлинг чистки...)**



Почему очередь на Cassandra — хорошая идея (иногда)

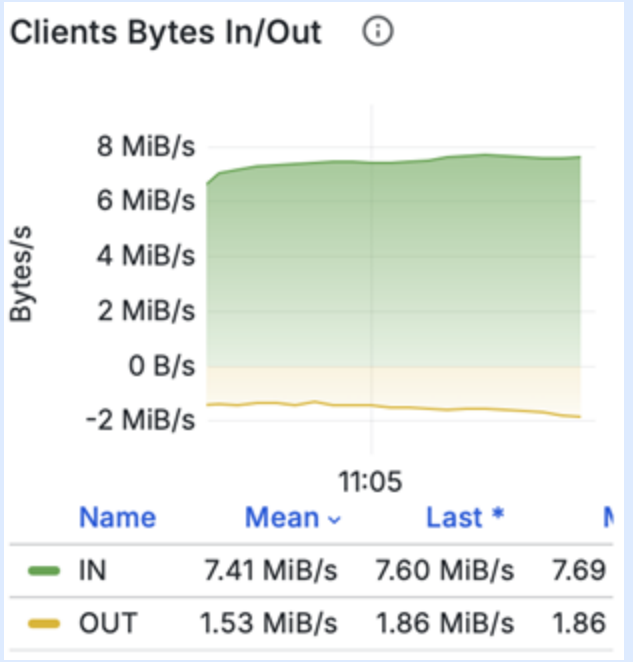
- Очередь на Cassandra позволила быстро мигрировать на метастор (1 кластер = 1 день)
- С небольшими объёмами она справлялась
- **С этим можно жить (truncate таблиц с очередью, троттлинг чистки...)**

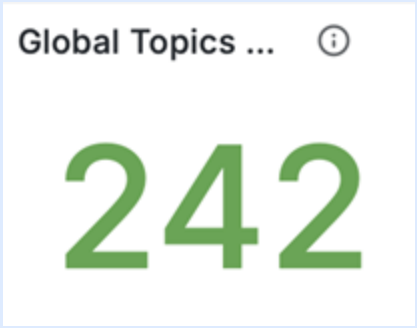


Гамбит Кассандры

Простой и быстрый переезд

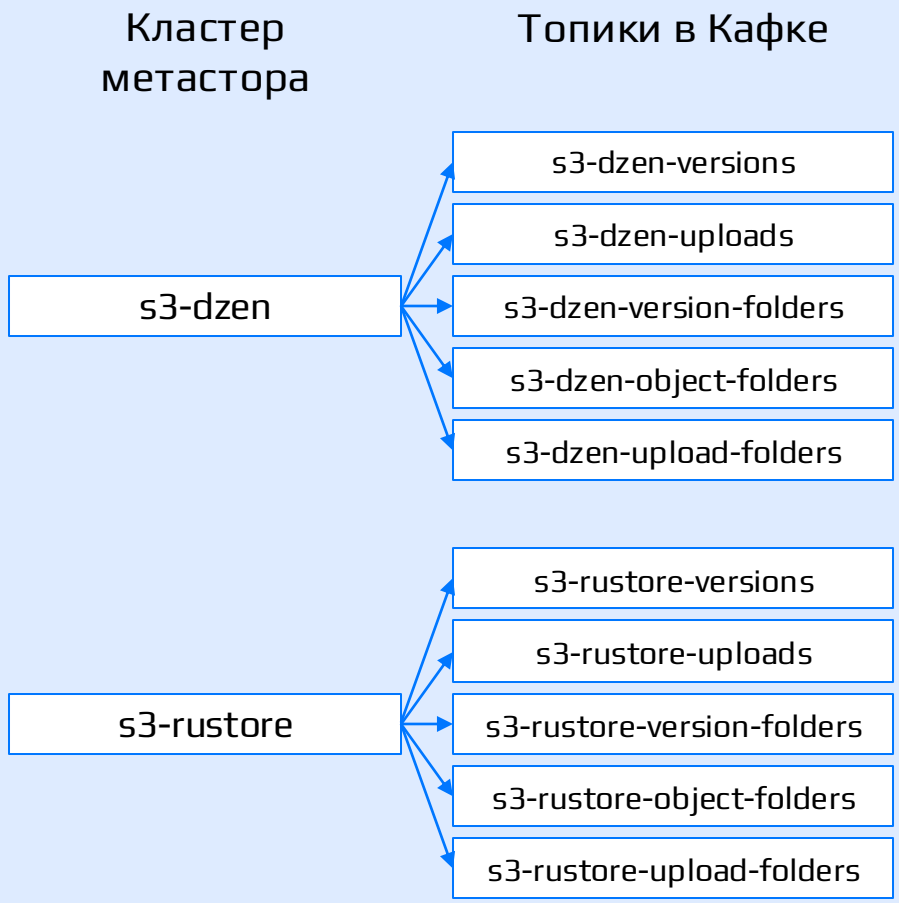
- **Один «большой» кластер Kafka**





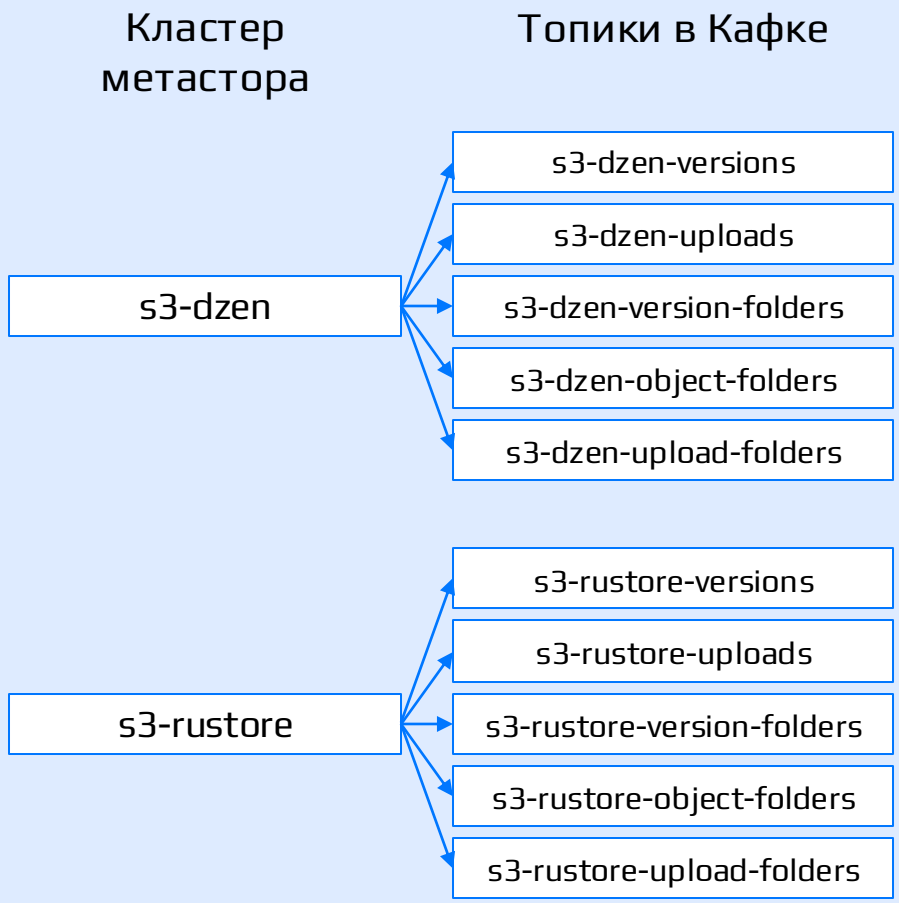
Простой и быстрый переезд

- Один «большой» кластер Kafka
- **1 топик на каждую очередь каждого кластера метастора**



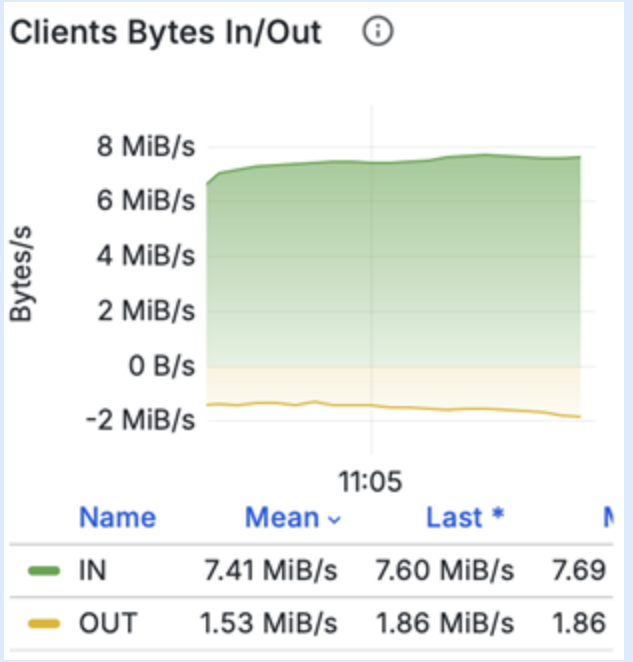
Простой и быстрый переезд

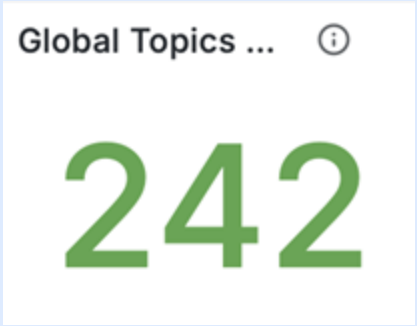
- Один «большой» кластер Kafka
- 1 топик на каждую очередь каждого кластера метастора
- **Включено автоматическое создание новых топиков**



Простой и быстрый переезд

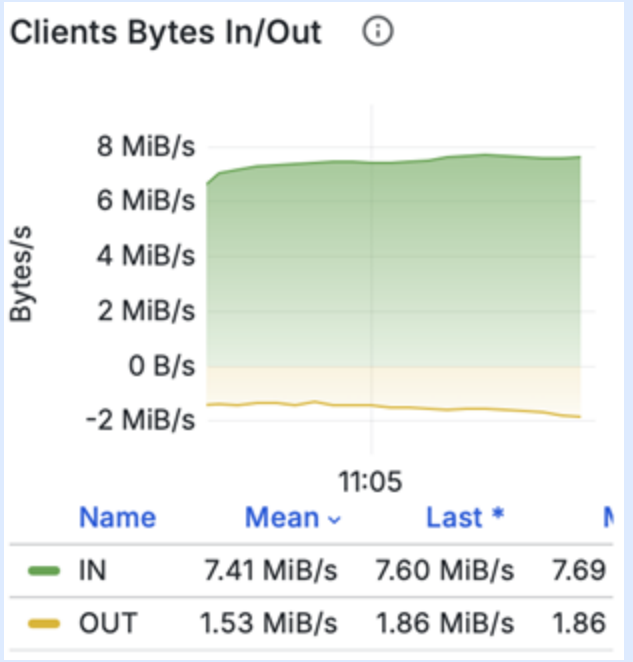
- Один «большой» кластер Kafka
- 1 топик на каждую очередь каждого кластера метастора
- Включено автоматическое создание новых топиков
- **3 патриции на топик по умолчанию и до 24 на крупных кластерах**

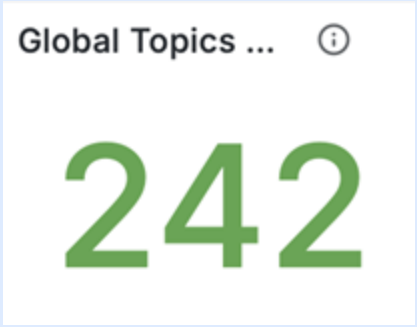




Простой и быстрый переезд

- Один «большой» кластер Kafka
- 1 топик на каждую очередь каждого кластера метастора
- Включено автоматическое создание новых топиков
- 3 патриции на топик по умолчанию и до 24 на крупных кластерах
- **Создан с помощью Managed Kafka (появился после новой версии метсатор)**

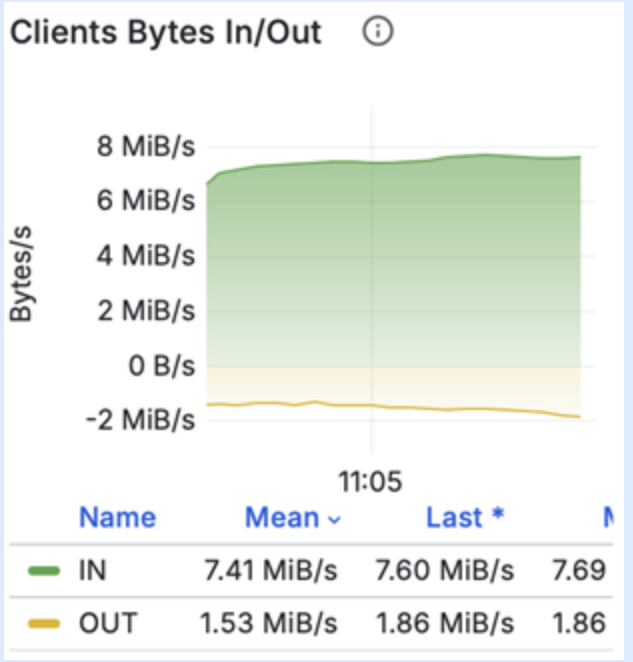


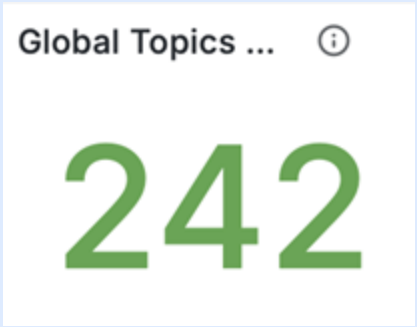


Простой и быстрый переезд

- Один «большой» кластер Kafka
- 1 топик на каждую очередь каждого кластера метастора
- Включено автоматическое создание новых топиков
- 3 патриции на топик по умолчанию и до 24 на крупных кластерах
- Создан с помощью Managed Kafka (появился после новой версии метсатор)
- **Feature toggle с имплементациями очереди**

Для смены очереди нужно только переключить настройку

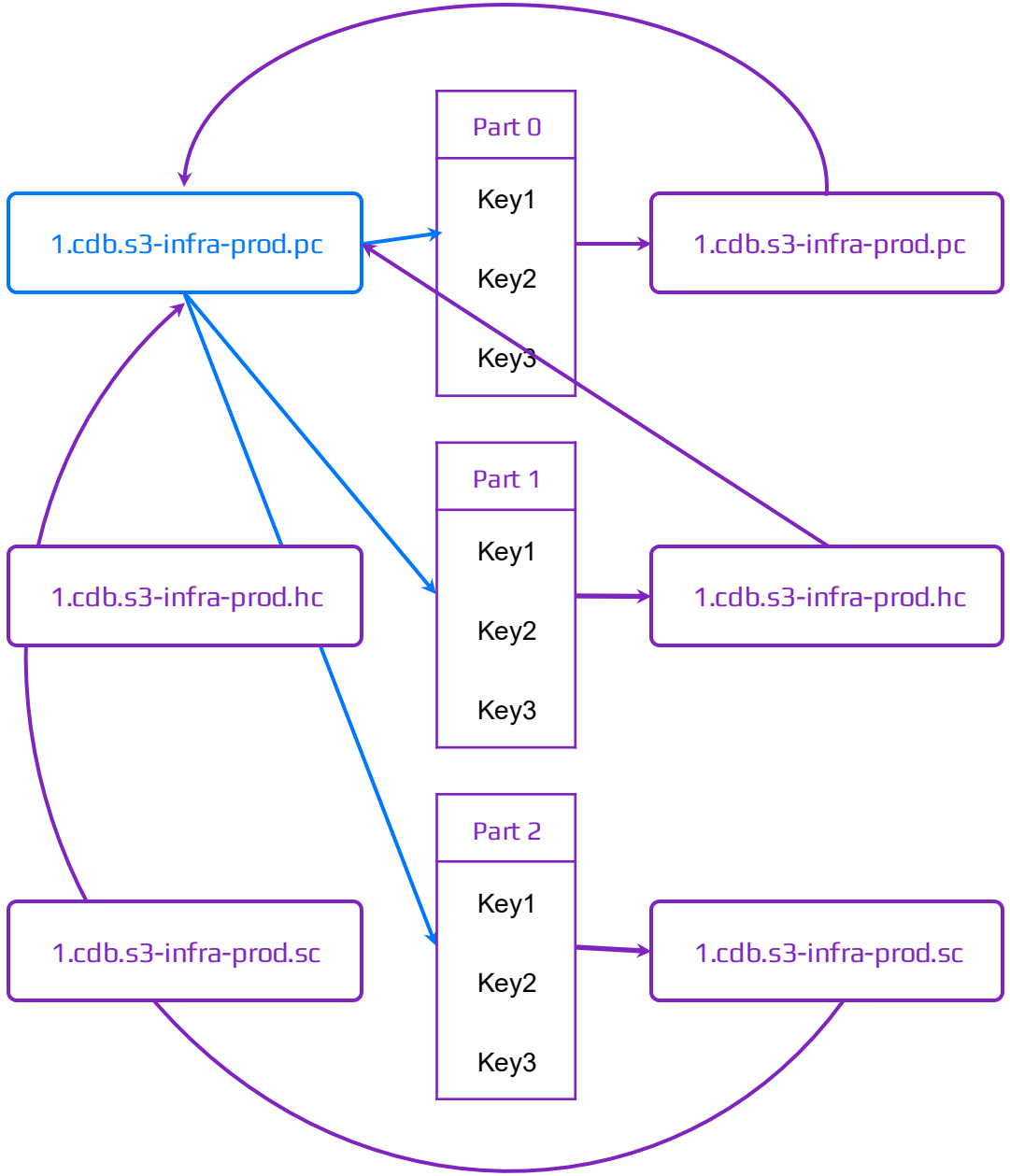




Равномерное партиционирование

Равномерное партиционирование ключей:

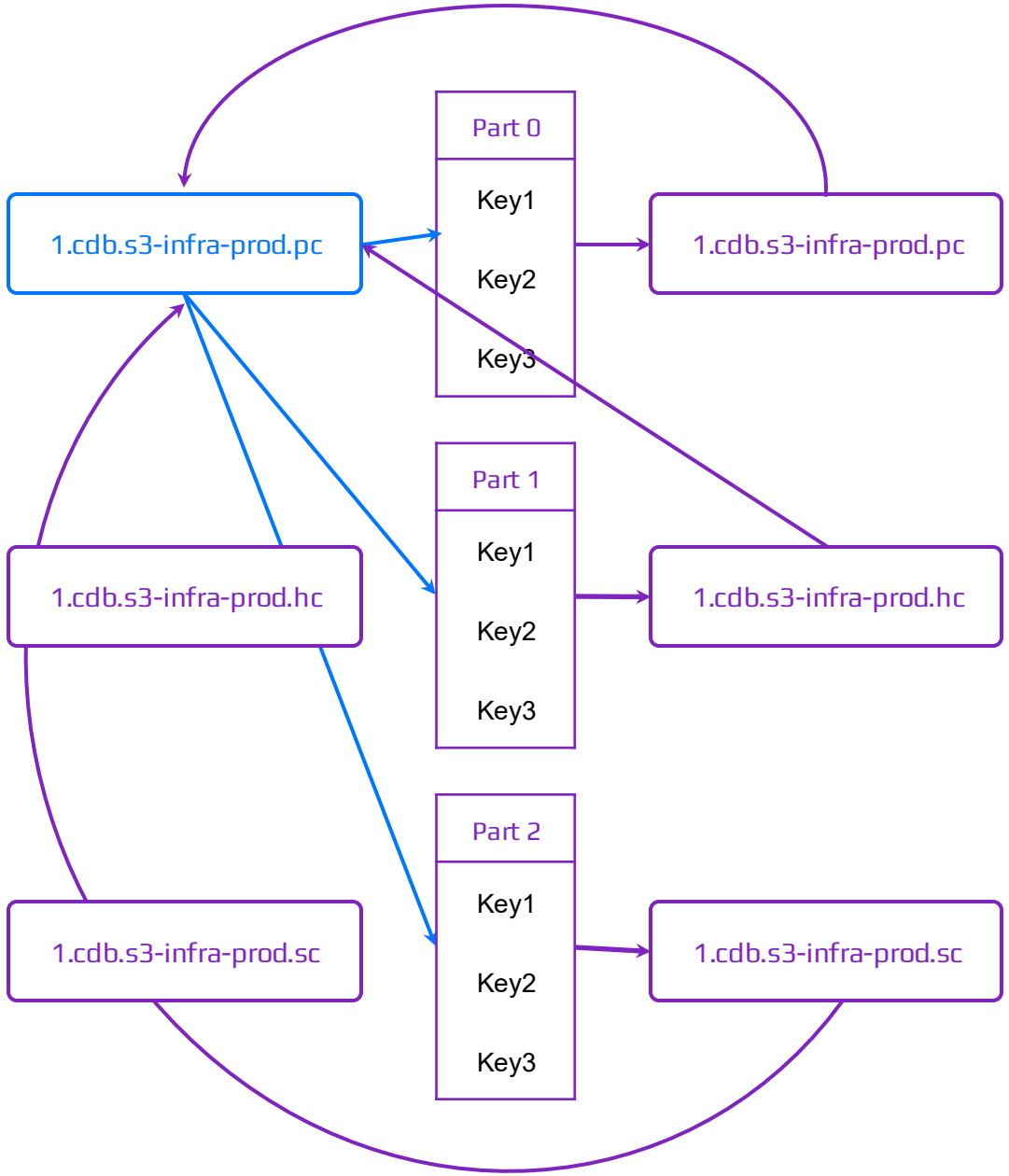
- **Хост начинает сканирование и запись кандидатов в очередь**



Равномерное партиционирование

Равномерное партиционирование ключей:

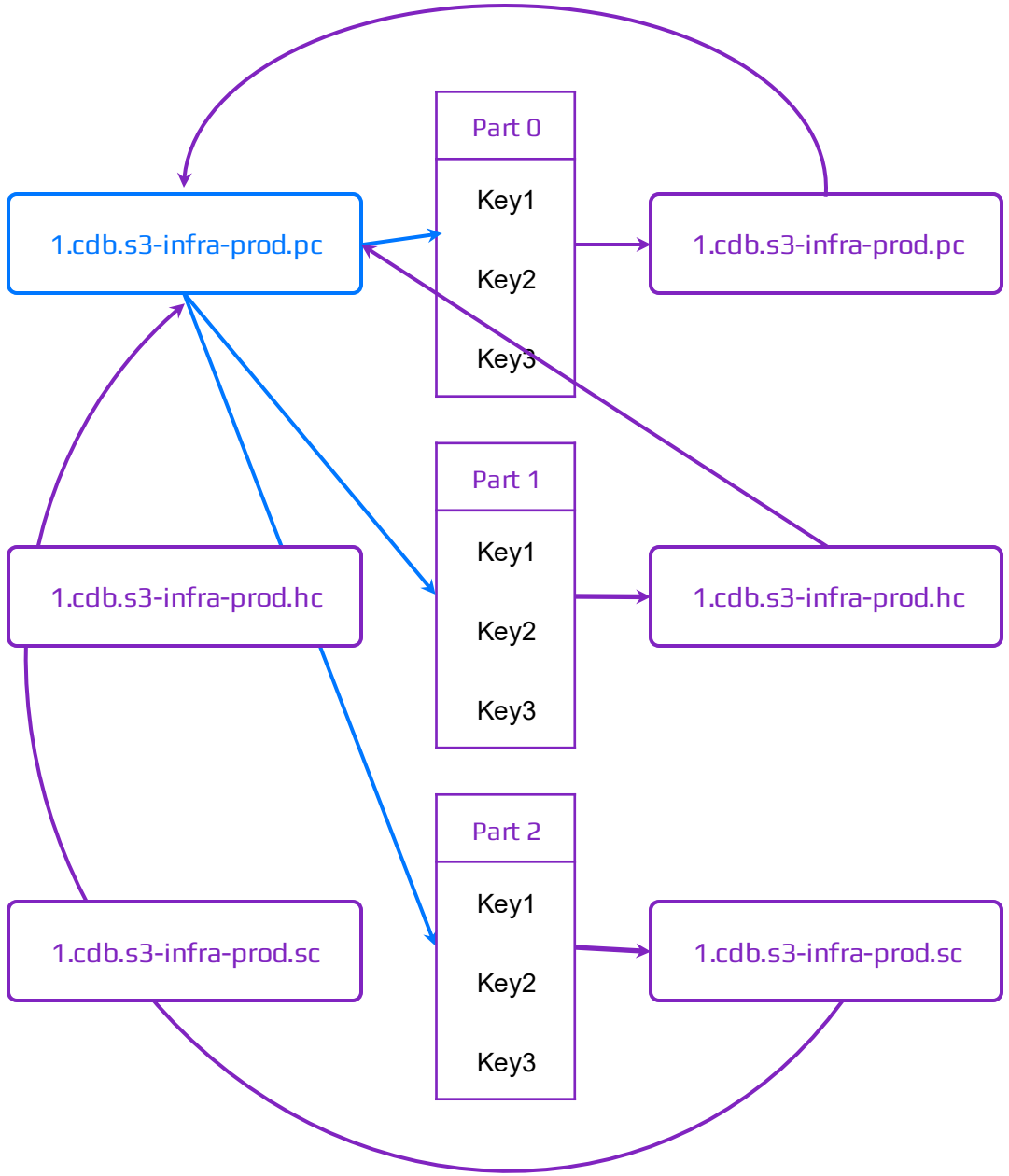
- Хост начинает сканирование и запись кандидатов в очередь
- **Кандидаты равномерно распределяются по всем партициям**



Равномерное партиционирование

Равномерное партиционирование ключей:

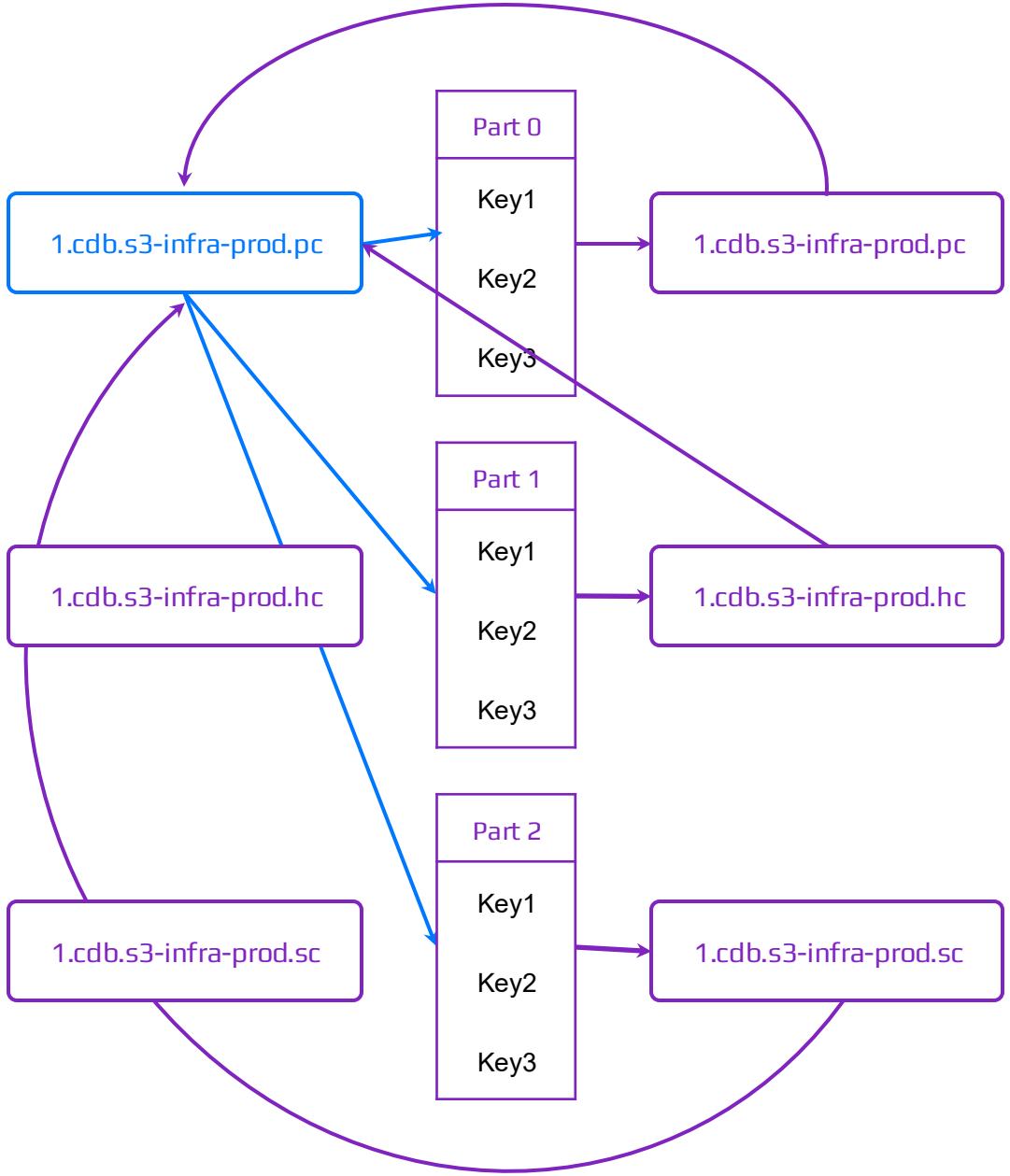
- Хост начинает сканирование и запись кандидатов в очередь
- Кандидаты равномерно распределяются по всем партициям
- **Партиции равномерно распределены по шедулерам**



Равномерное партиционирование

Равномерное партиционирование ключей:

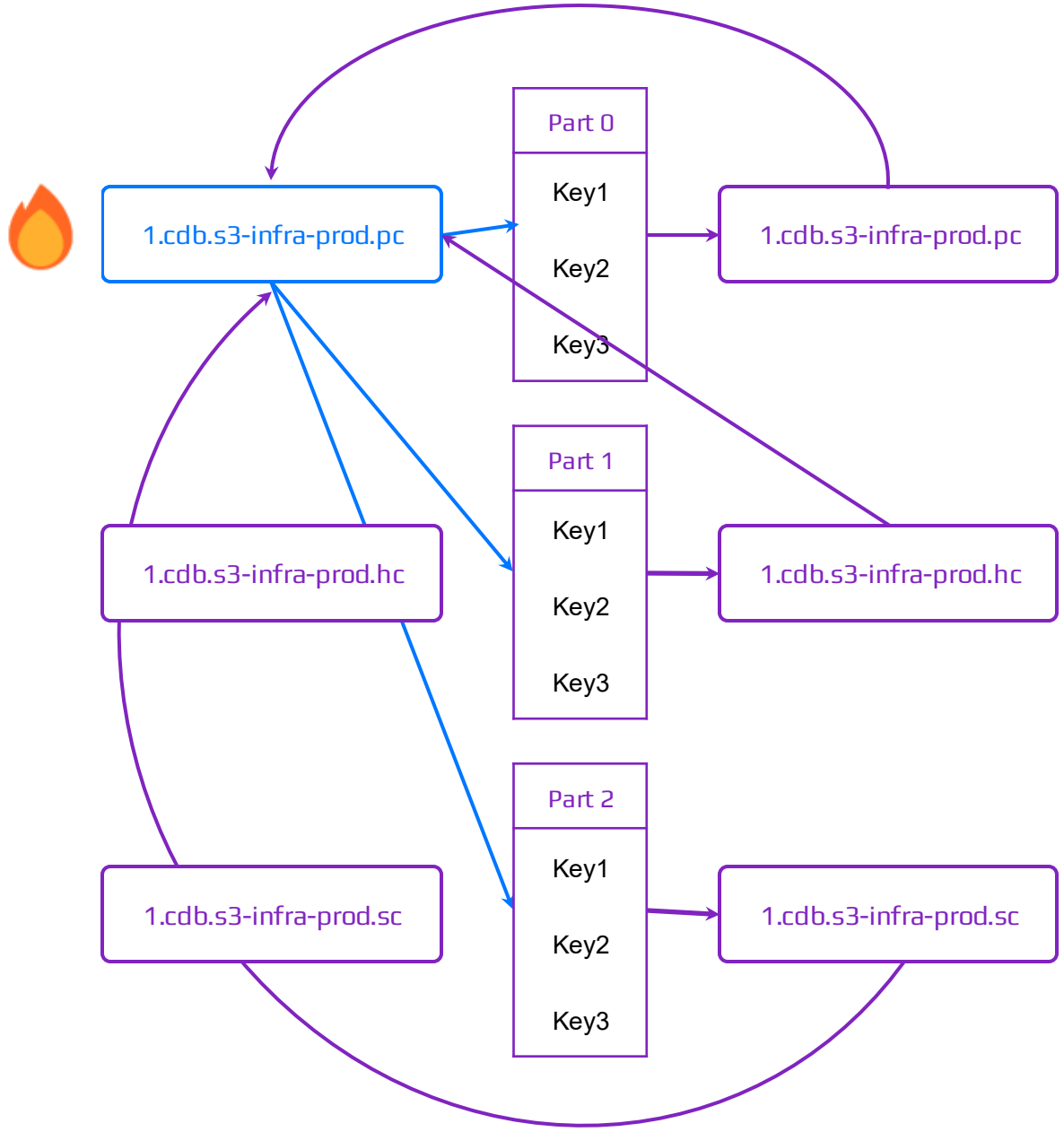
- Хост начинает сканирование и запись кандидатов в очередь
- Кандидаты равномерно распределяются по всем партициям
- Партиции равномерно распределены по шедулерам
- **Все шедулеры одновременно начинают зачистку кандидатов с одного хоста**



Равномерное партиционирование

Равномерное партиционирование ключей:

- Хост начинает сканирование и запись кандидатов в очередь
- Кандидаты равномерно распределяются по всем партициям
- Партиции равномерно распределены по шедулерам
- Все шедулеры одновременно начинают зачистку кандидатов с одного хоста

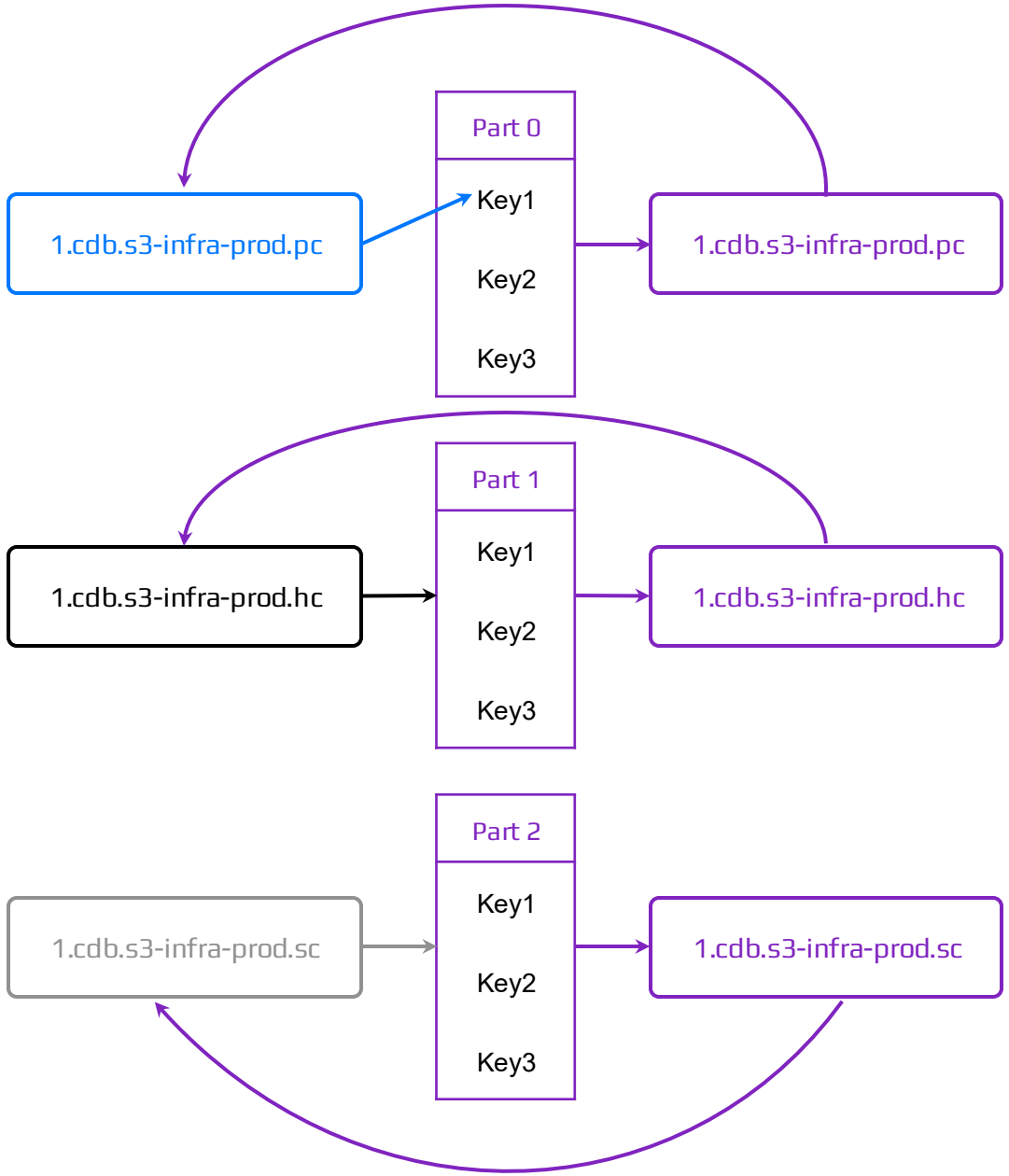


Результат: избыточная нагрузка на 1 хост Кассандры.
Чистка упирается в возможности хоста.
Тормозят запросы с API

Эффективное партиционирование

Партиционирование по хосту:

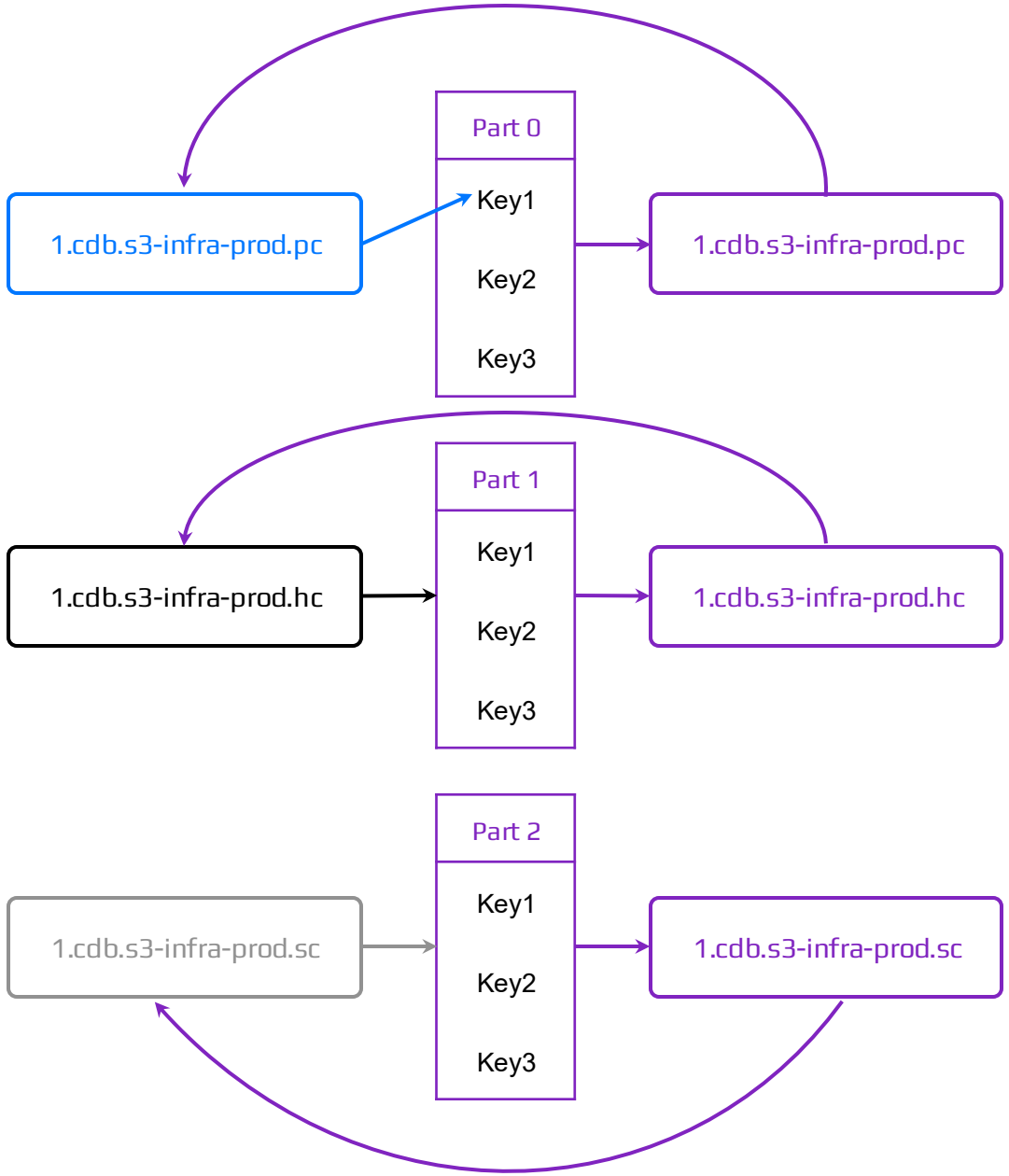
- **Каждый хост всегда пишет в одну и ту же партицию**



Эффективное партиционирование

Партиционирование по хосту:

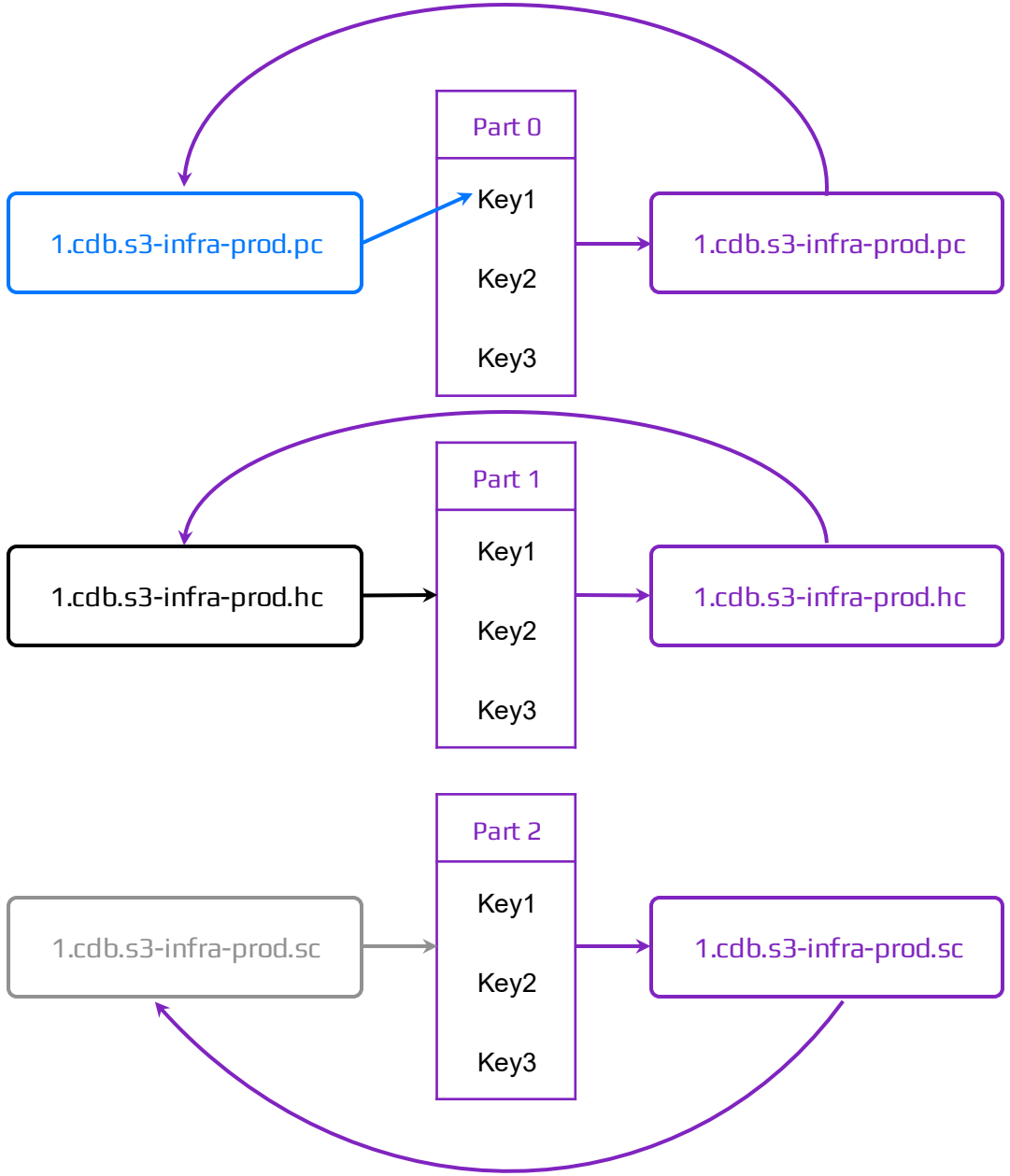
- Каждый хост всегда пишет в одну и ту же партицию
- **Каждую партицию читает не более чем 1 шедулер**



Эффективное партиционирование

Партиционирование по хосту:

- Каждый хост всегда пишет в одну и ту же партицию
- Каждую партицию читает не более чем 1 шедулер
- **Равномерная нагрузка на хосты**

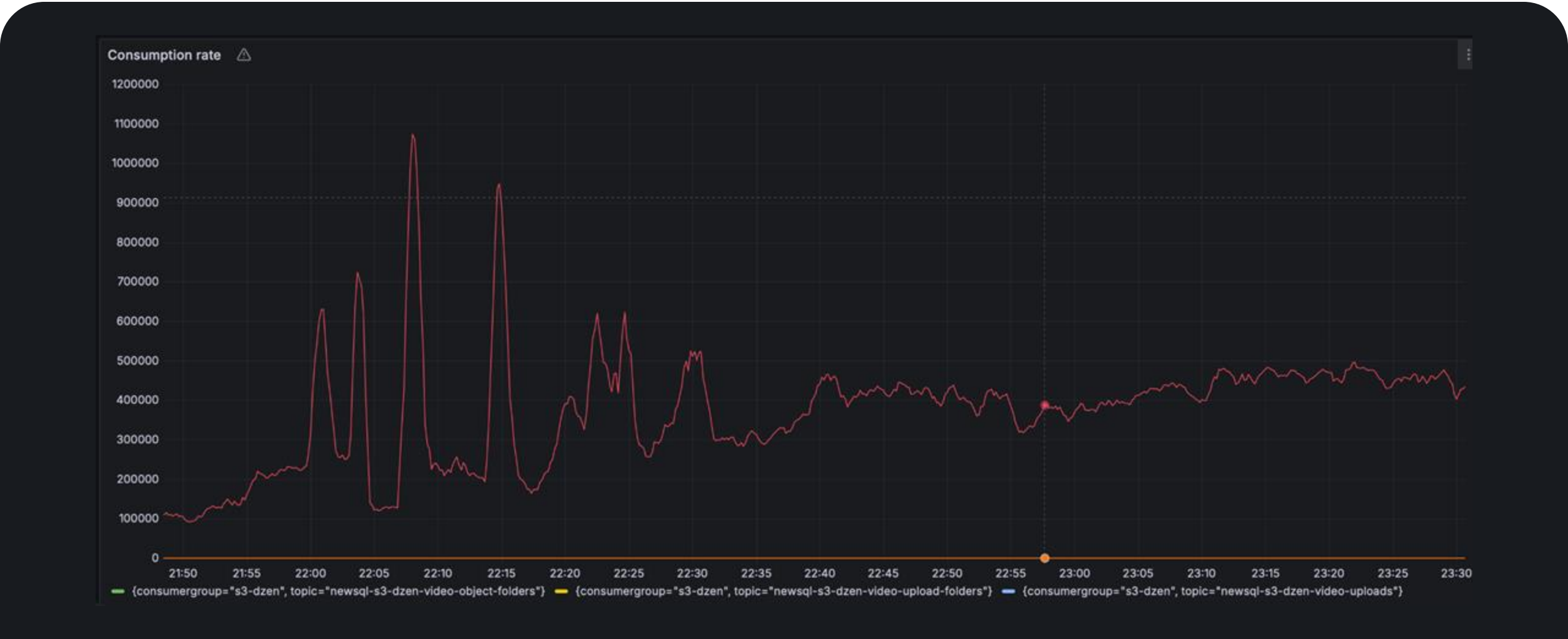


Скорость скана

18 хостов метастора — **14m rpm**

Скорость чистки

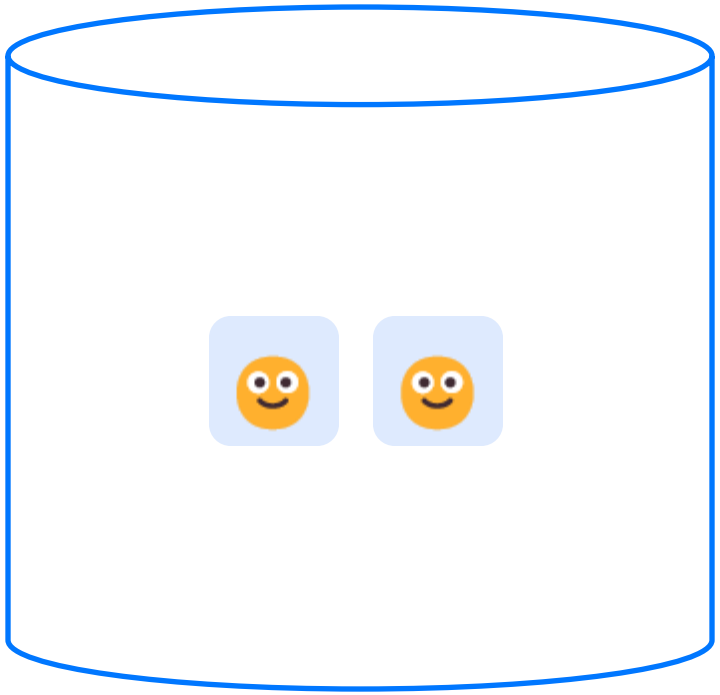
24 хоста шедулера — **500k rpm (1.1m peak)**



Статистика

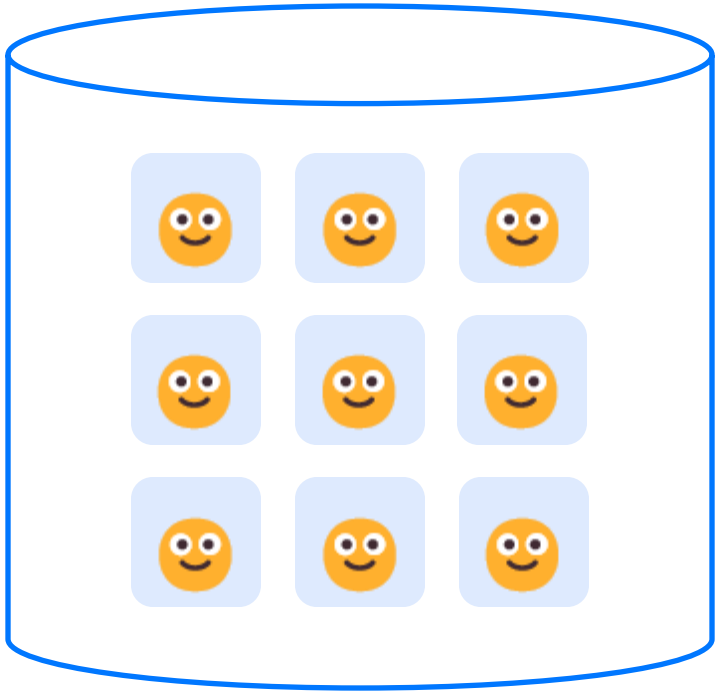
Задача: ограничение размера бакета

- **Изначально у S3 было немного клиентов**
- **Один бакет = одно хранилище**



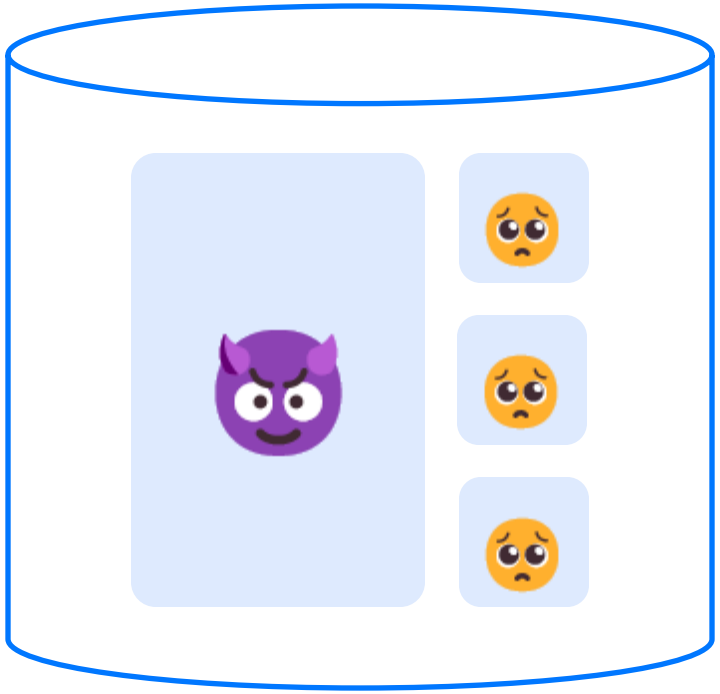
Задача: ограничение размера бакета

- Изначально у S3 было немного клиентов
- Один бакет = одно хранилище
- **Количество клиентов выросло**
- **Стали селить разные команды в общие хранилища**



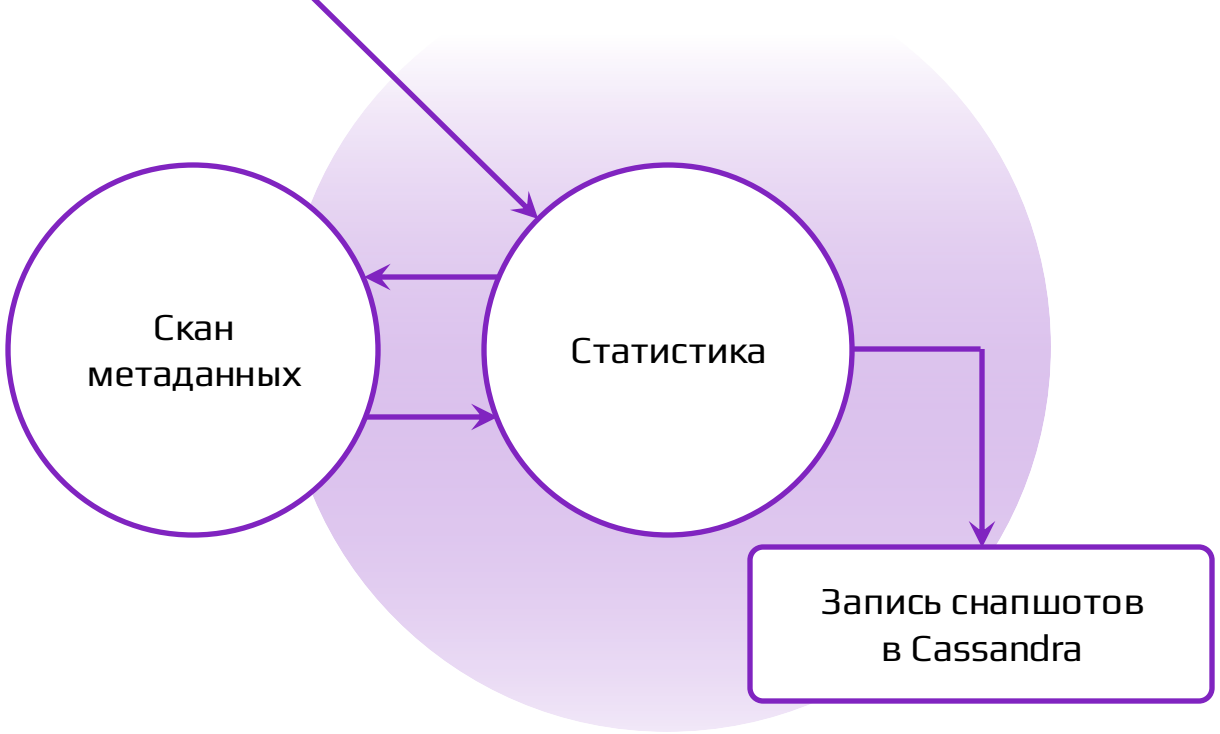
Задача: ограничение размера бакета

- Изначально у S3 было немного клиентов
- Один бакет = одно хранилище
- Количество клиентов выросло
- Стали селить разные команды в общие хранилища
- **Кто-то потребляет слишком много места – страдают все**



Сбор статистики

В результате скана запоминаем состояние кластера — **снапшот**



source	ks	ts	handler	stat	status
d3a93007-...-066477d79853	sandbox	1771271254	versions	<BLOB>	completed
09496823-...-7f183c454e11	sandbox	1771271496	uploads	<BLOB>	resumeKey:1045

Сбор статистики

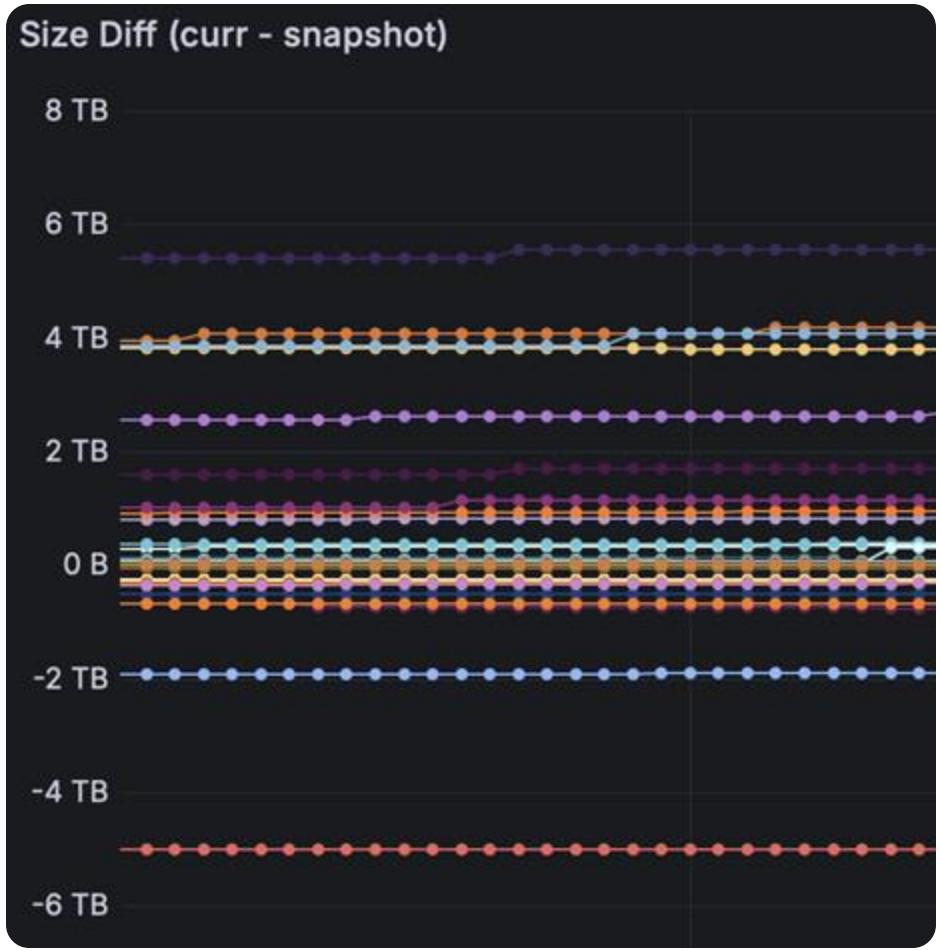
source	ks	ts	handler	stat	status
d3a93007-...-066477d79853	sandbox	1771271254	versions	<BLOB>	completed
09496823-...-7f183c454e11	sandbox	1771271496	uploads	<BLOB>	resumeKey:1045

- Source — ID хоста
- Status — либо ключ, на котором закончился скан, либо completed
- Handler — сканируемая таблица
- Stat содержит всю статистик в сериализованном виде

- **Размер всех версий бакета**
- Топ объектов по размеру, количеству версий и т. д.
- Топ директорий по количеству объектов
- Блоки с наибольшим числом ссылок
- ...

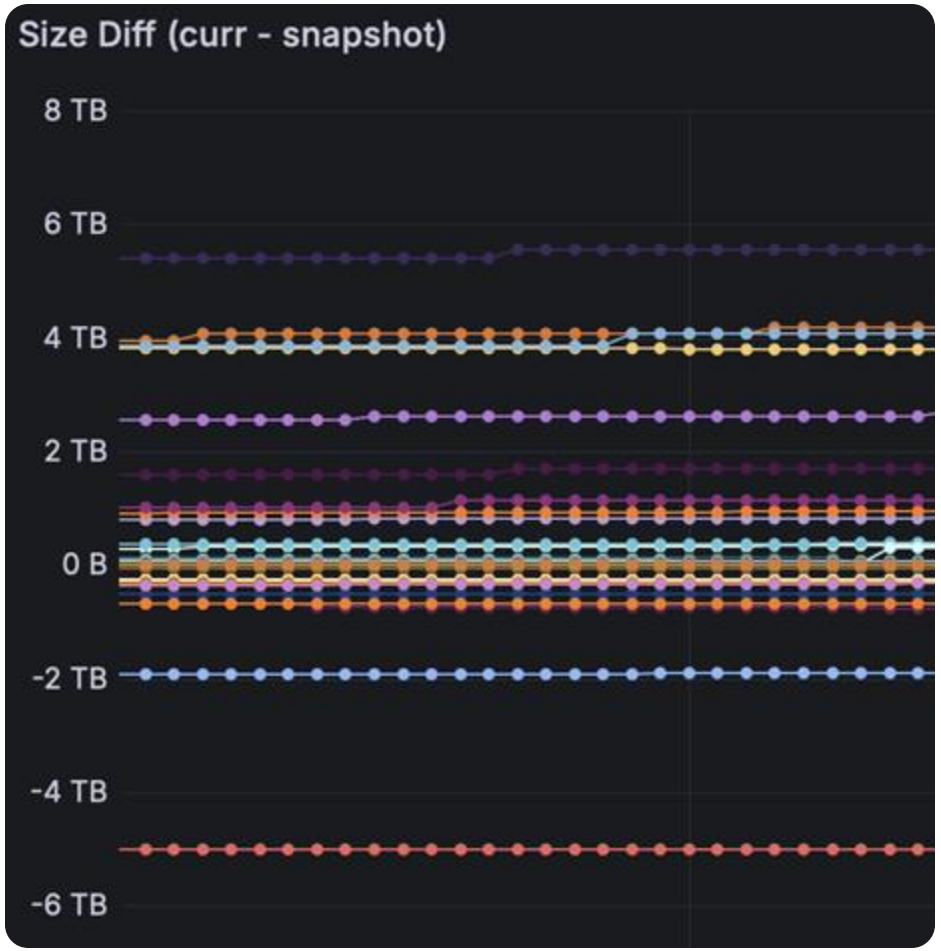
Проблемы статистики из снапшотов

- **Статистика собирается долго (до 1 часа) и быстро теряет актуальность**



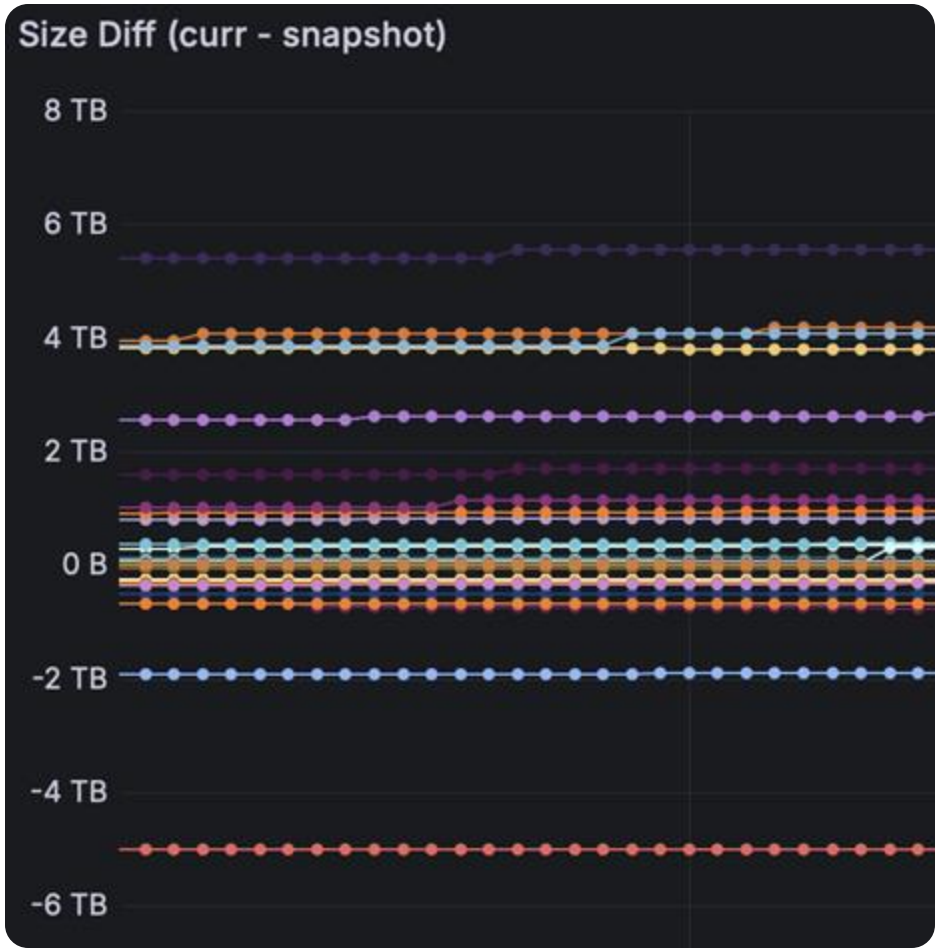
Проблемы статистики из снапшотов

- Статистика собирается долго (до 1 часа) и быстро теряет актуальность
- **Невозможно вовремя ограничить запросы на бакет при превышении квоты**



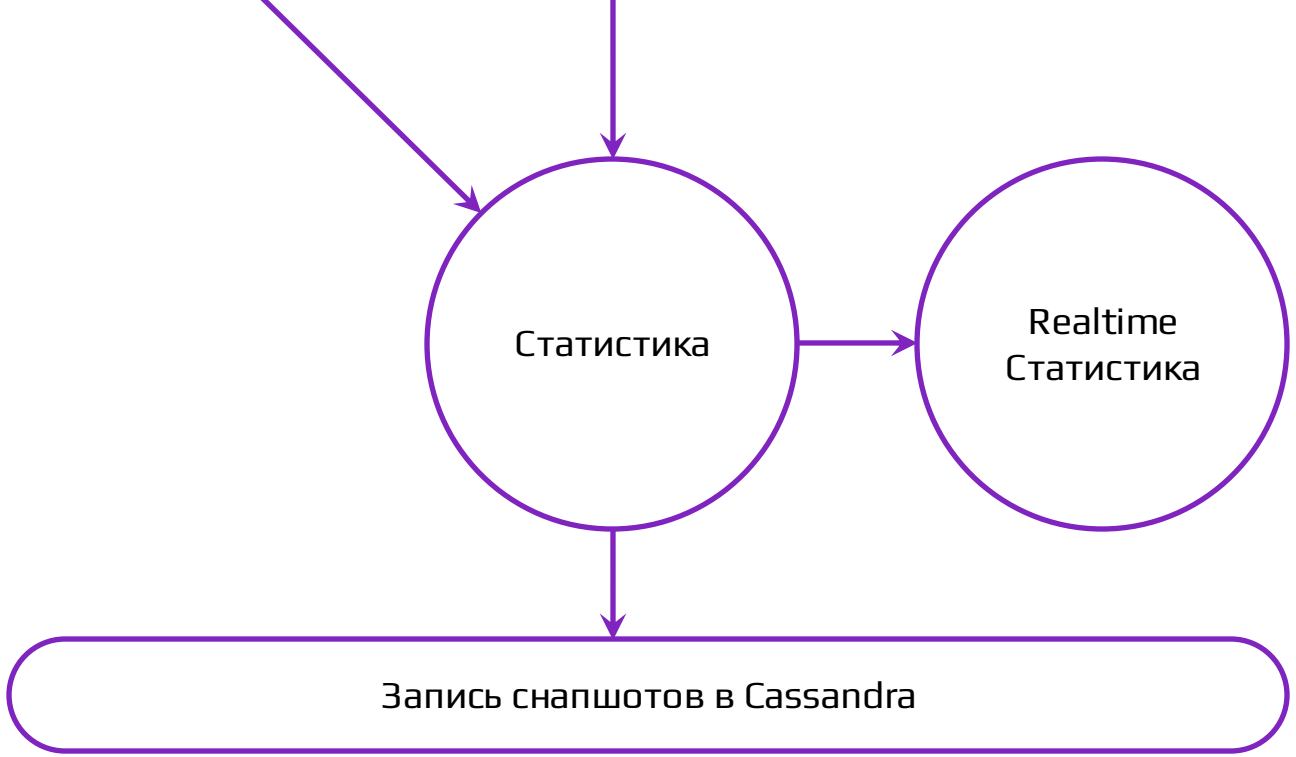
Проблемы статистики из снапшотов

- Статистика собирается долго (до 1 часа) и быстро теряет актуальность
- Невозможно вовремя ограничить запросы на бакет при превышении квоты
- **Самому пользователю сложно принимать решения**



Проблемы статистики из снапшотов

- Статистика собирается долго (до 1 часа) и быстро теряет актуальность
- Невозможно вовремя ограничить запросы на бакет при превышении квоты
- Самому пользователю сложно принимать решения



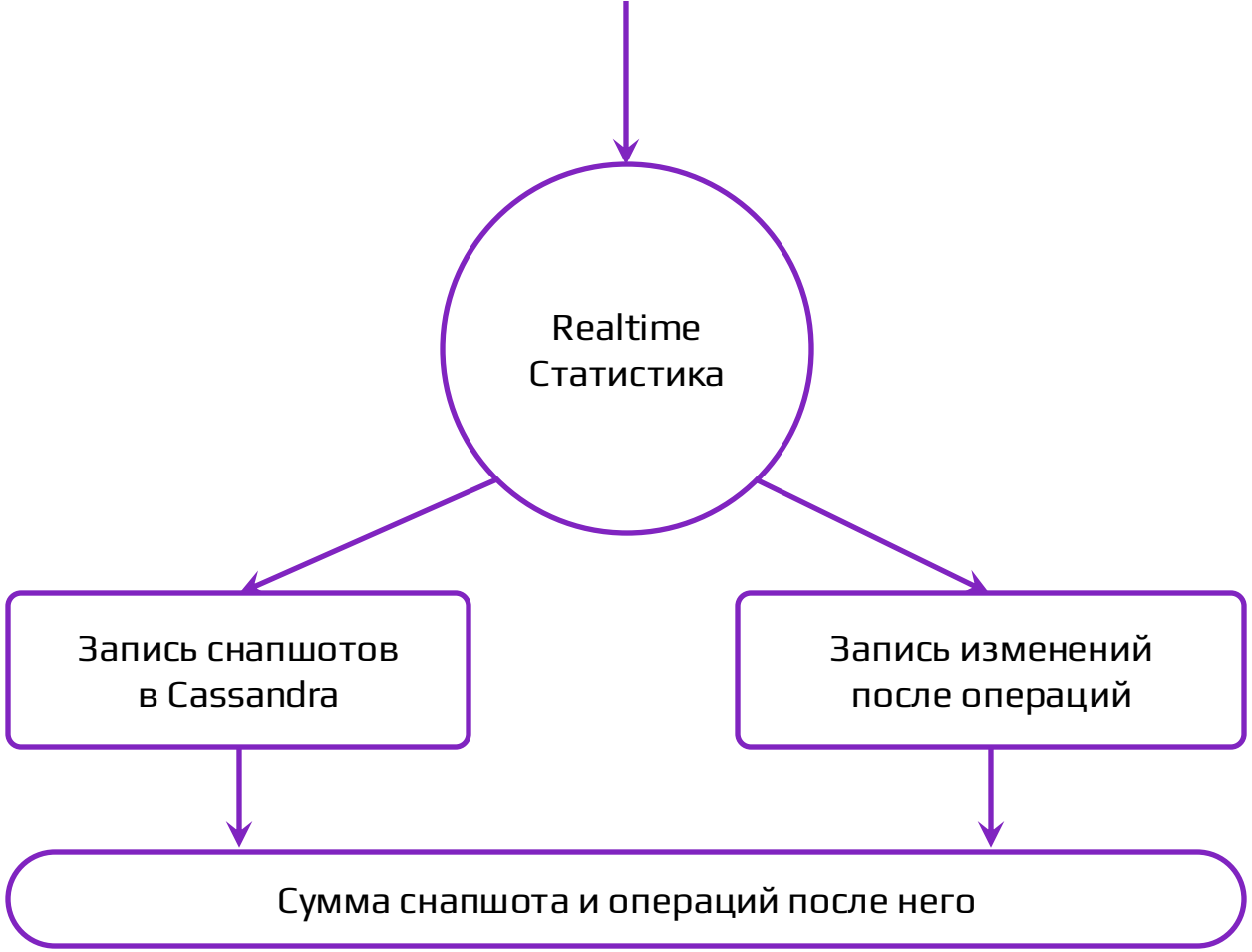
Решение: нужна статистика в реальном времени

Статистика
в реальном времени

Идея — считать размер бакета
как результат из снапшота + сумма diff
для всех операций, прошедших
с последнего снапшота

Операции, меняющие размер бакета:

- Запись нового блока
- Удаление существующего блока
- Копирование блока



Завершилась задача сбора статистики, получен снапшот

+blkStore

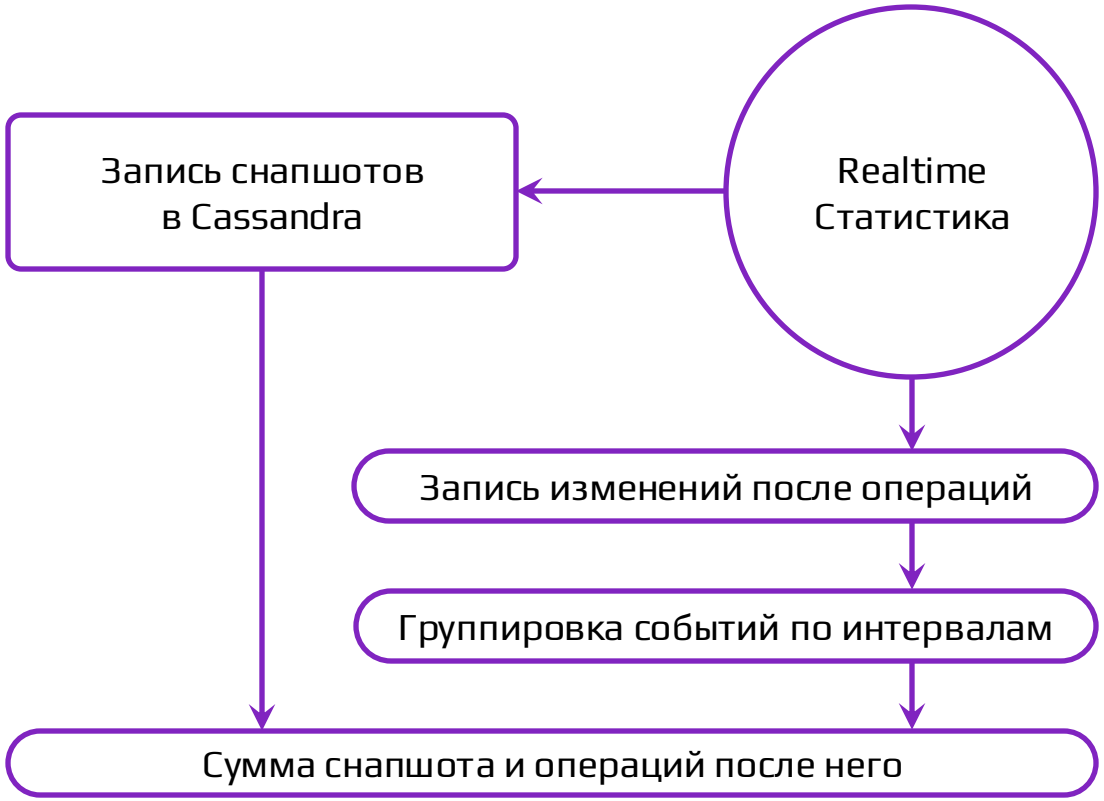
+blkCopy

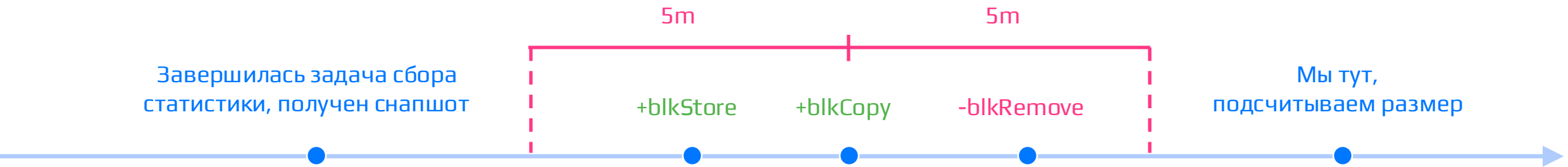
-blkRemove

Мы тут,
подсчитываем размер

Статистика
в реальном времени

С целью оптимизации работает
отдельная фоновая задача,
которая группирует операции
по пятиминутным интервалам



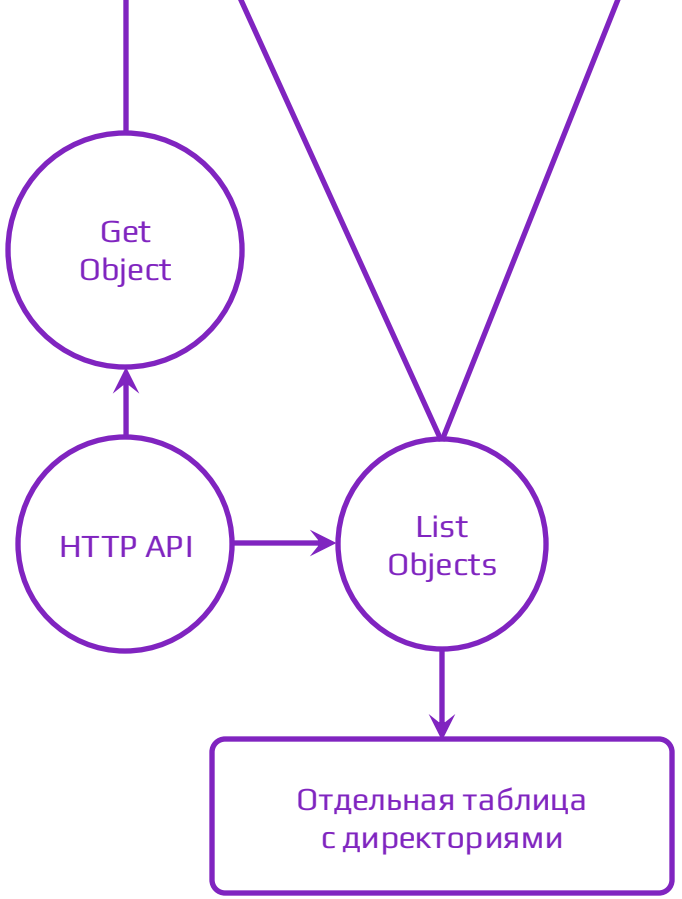


Прочие фоновые задачи

ListObjects

images/pets/cats/black/blackCat.png

Parent	Name
images/	pets
images pets/	cats
images/ pets/cats	black



- Для ускорения листинга существует таблица-индекс, отражающая структуру «Директорий»
- Таблицы и версии партиционированы по директориям

Обновление индекса при удалении объекта

1. Удаляем объект

Objects

Parent	Name
images/pets	dog.png

2. Проверяем, стала ли parent-директория пустой

Objects

Parent	Name

```
SELECT * FROM objects WHERE
parent='images/pets'
limit 1
```

Folders

Parent	Name
images/	pets

3. Удаляем пустую директорию

Folders

Parent	Name
~~images/~~	~~pets~~

Обновление индекса при удалении объекта

Проблема: большинство директорий содержат **больше одного объекта!**

1

Objects

Parent	Name
content/tmp	aaa
content/tmp	aab
content/tmp	aac
content/tmp	aad
content/tmp	aba
…	…

2

Objects

Parent	Name
content/tmp	aad
content/tmp	aba
…	…

```
SELECT * FROM objects WHERE
parent='content/tmp'
limit 1

SELECT * FROM objects WHERE
parent='content/tmp'
limit 1

SELECT * FROM objects WHERE
parent='content/tmp'
limit 1

…
```

Folders

Parent	Name
content/	tmp

Большинство проверок оказываются ложными

Обновление индекса при листинге директории

1. Удаляем объект, не обновляем индекс

Objects

Parent	Name
~~images/pets~~	~~dog.png~~

2. Начинается рекурсивный листинг, собираем список директорий

aws s3 ls -r images/

Folders

Parent	Name
images/	pets

3. Рекурсивный листинг автоматически проверяет, является ли директория пустой

Objects

Parent	Name

```
SELECT * FROM objects WHERE
parent='images/pets'
```

Folders

Parent	Name
images/	pets

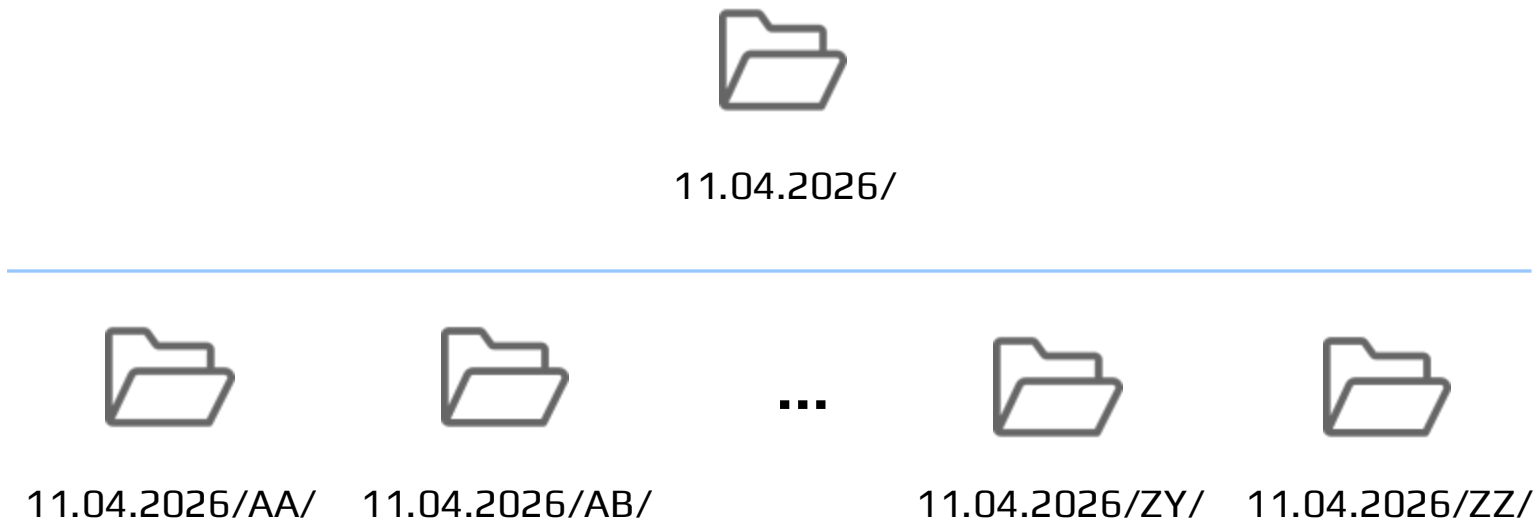
4. Удаляем пустую директорию

Folders

Parent	Name
~~images/~~	~~pets~~

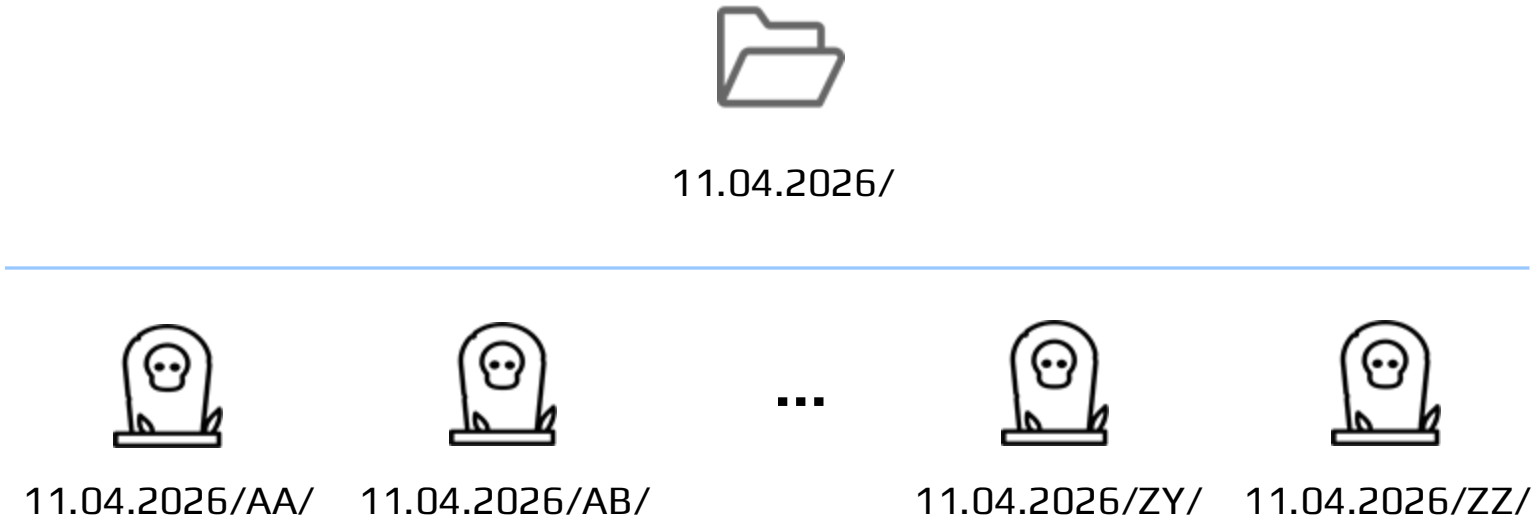
Чистка пустых директорий

- **При ленивом удалении под одним родителем может скопиться большое количество пустых директорий**



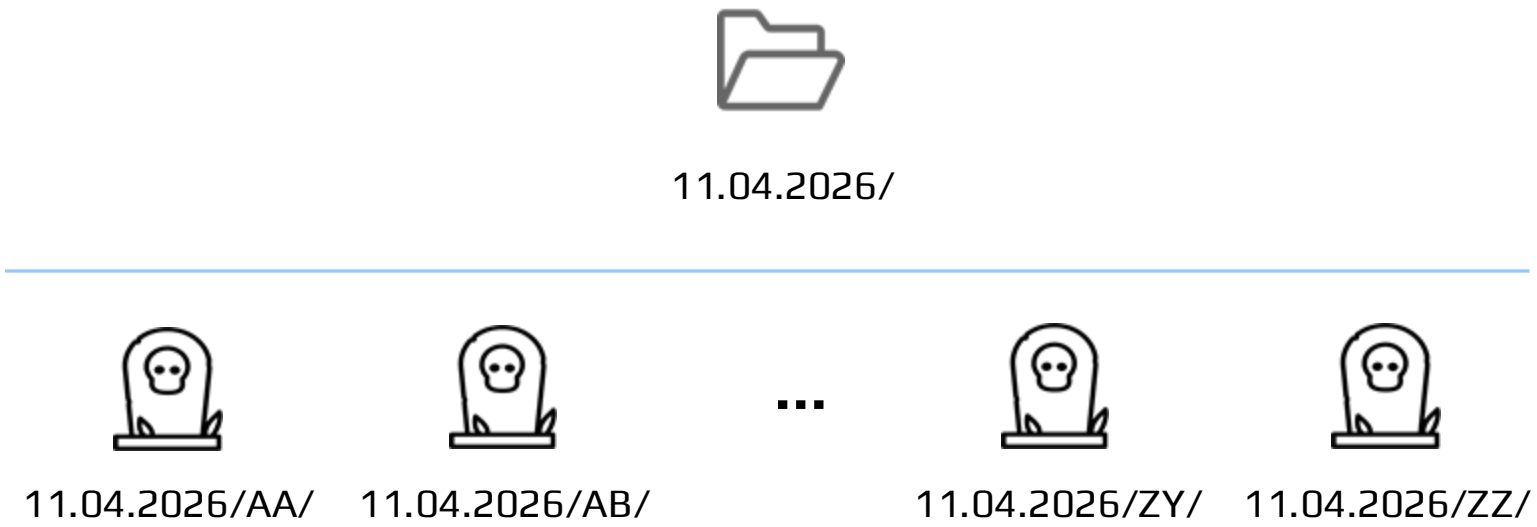
Чистка пустых директорий

- При ленивом удалении под одним
  родителем может скопиться большое
  количество пустых директорий
- **Их одновременное удаление замогилит партицию родительской директории**



Чистка пустых директорий

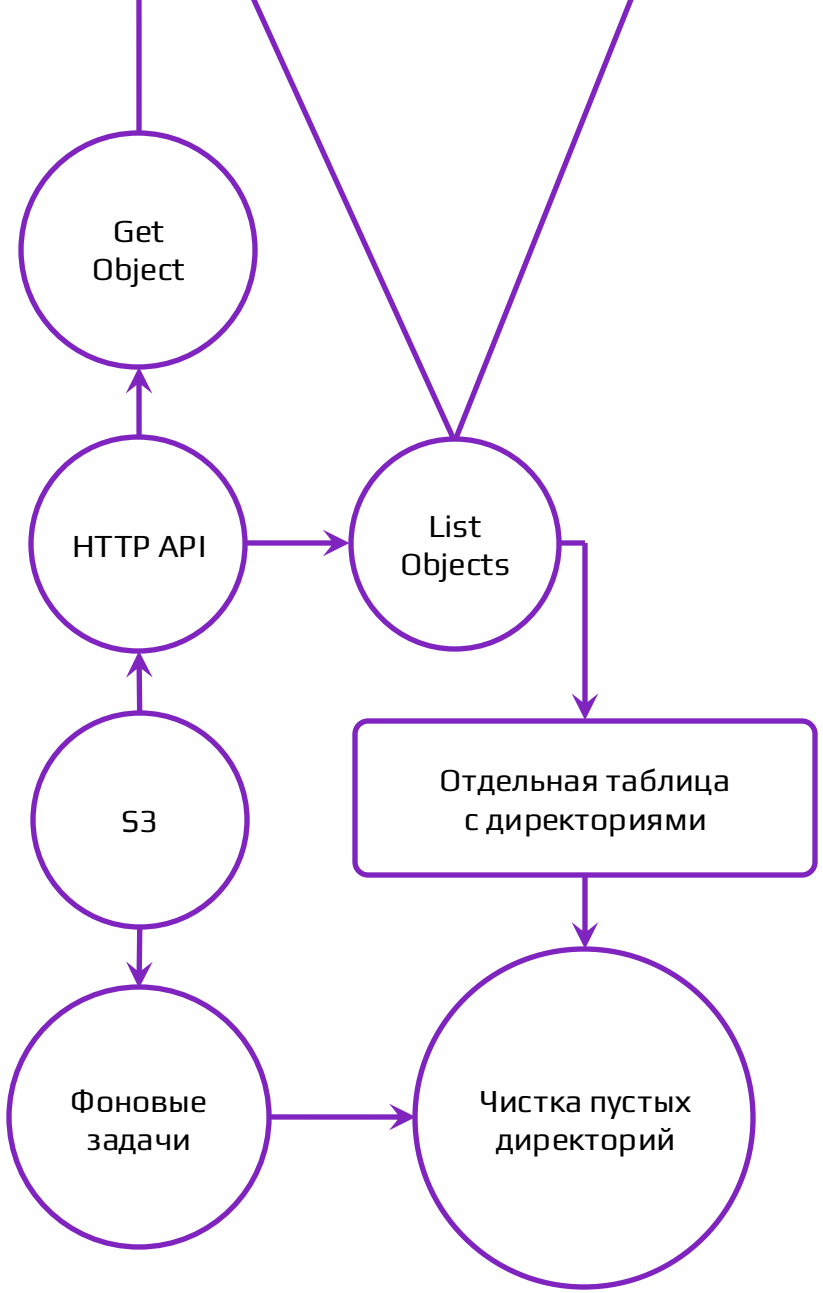
- При ленивом удалении под одним родителем может скопиться большое количество пустых директорий
- Их одновременное удаление замогилит партицию родительской директории
- **Могут сильно замедлить рекурсивный листинг**



Чистка пустых директорий

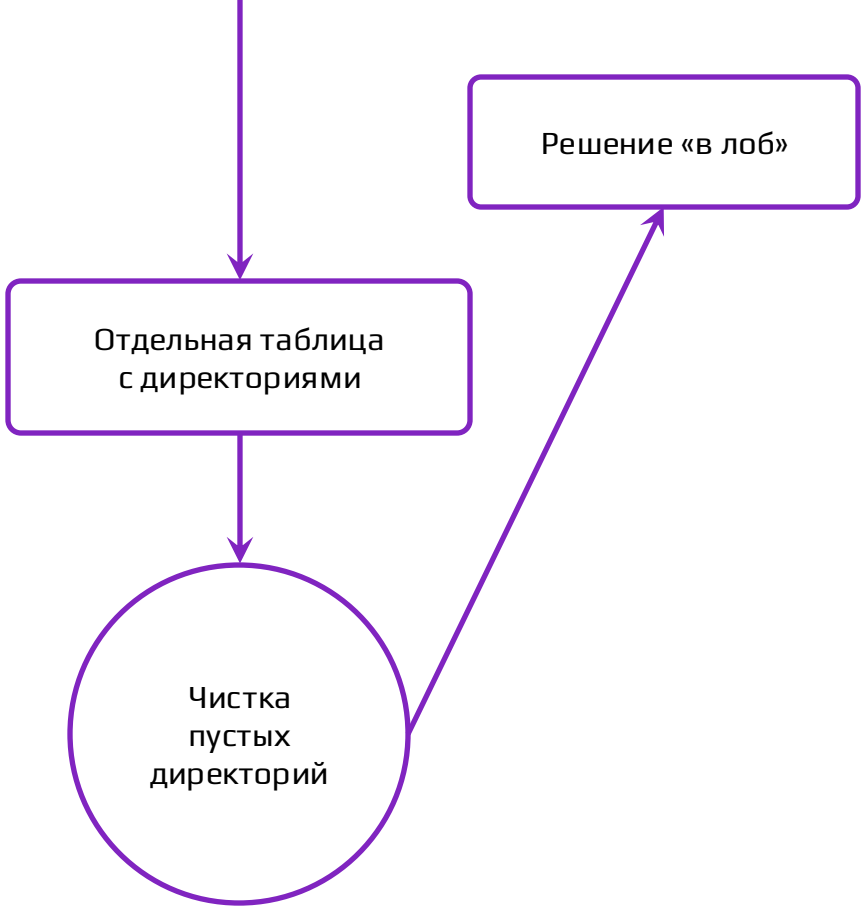
- При ленивом удалении под одним родителем может скопиться большое количество пустых директорий
- Их одновременное удаление замогилит партицию родительской директории
- **Могут сильно замедлить рекурсивный листинг**

Необходима фоновая чистка пустых директорий



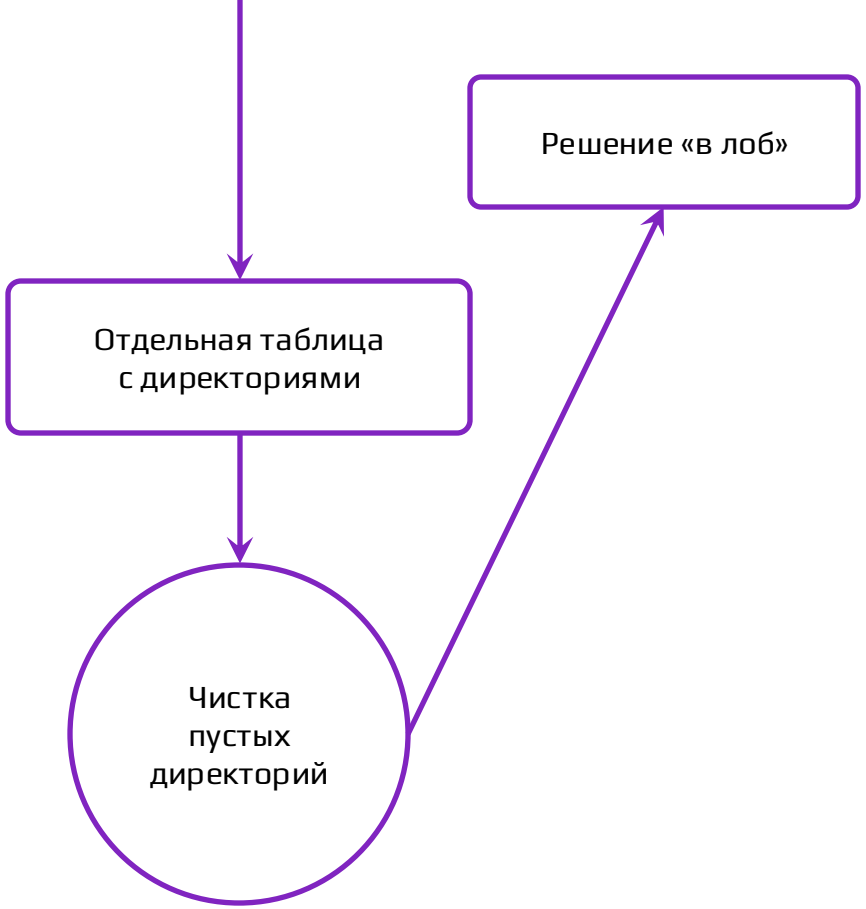
Решение «в лоб»

- **Локально сканируем таблицу с директориями**



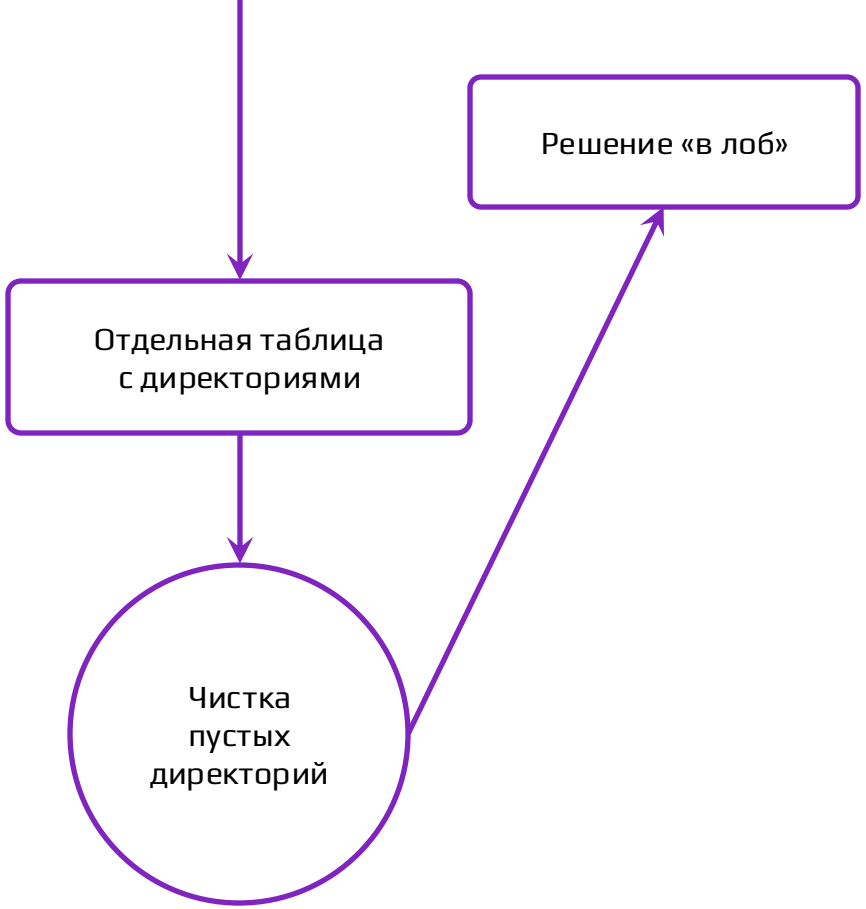
Решение «в лоб»

- Локально сканируем таблицу с директориями
- **Также локально проверяем наличие в ней объектов**



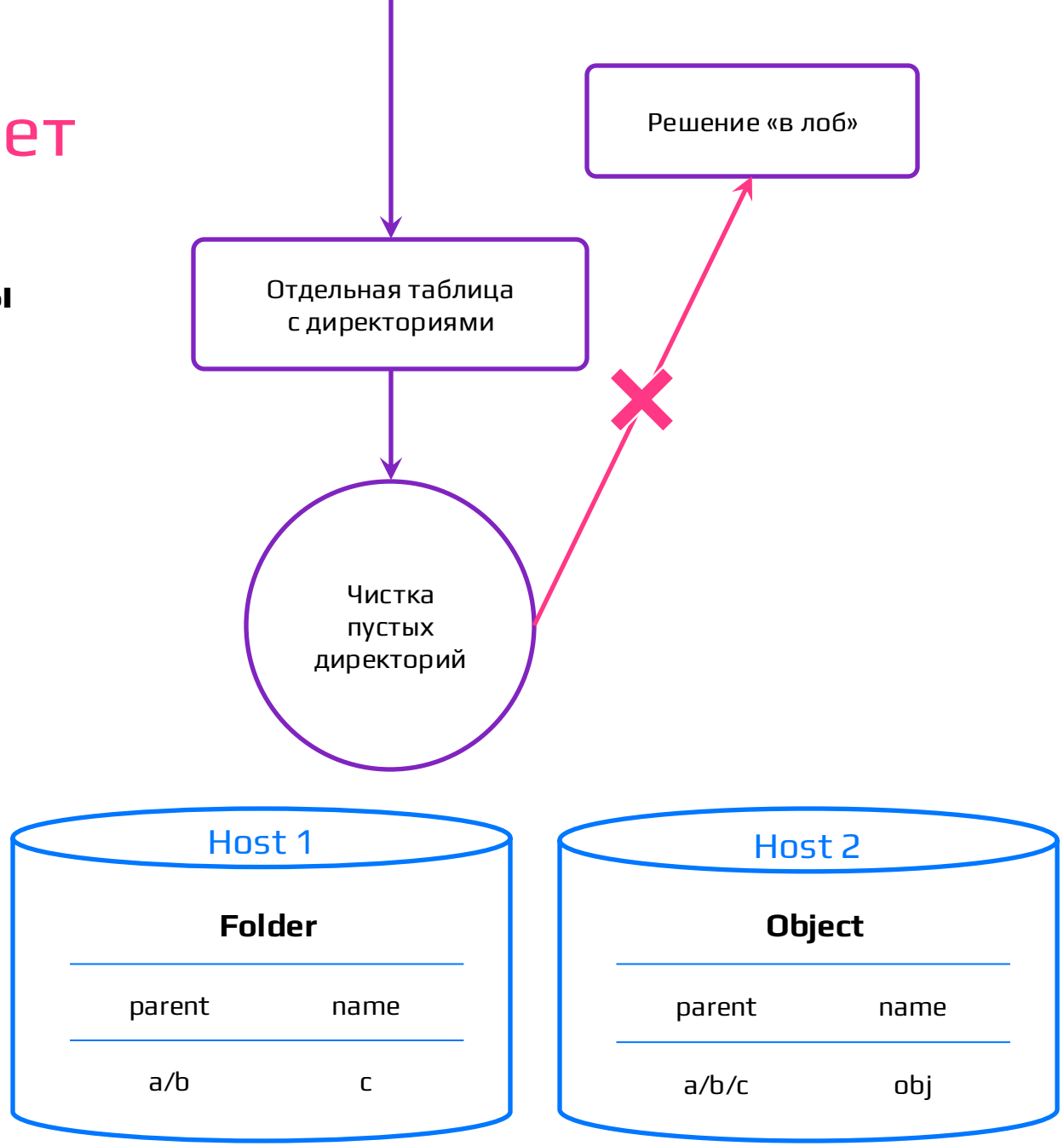
Решение «в лоб»

- Локально сканируем таблицу с директориями
- Также локально проверяем наличие в ней объектов
- **Кладём в очередь директории без объектов**



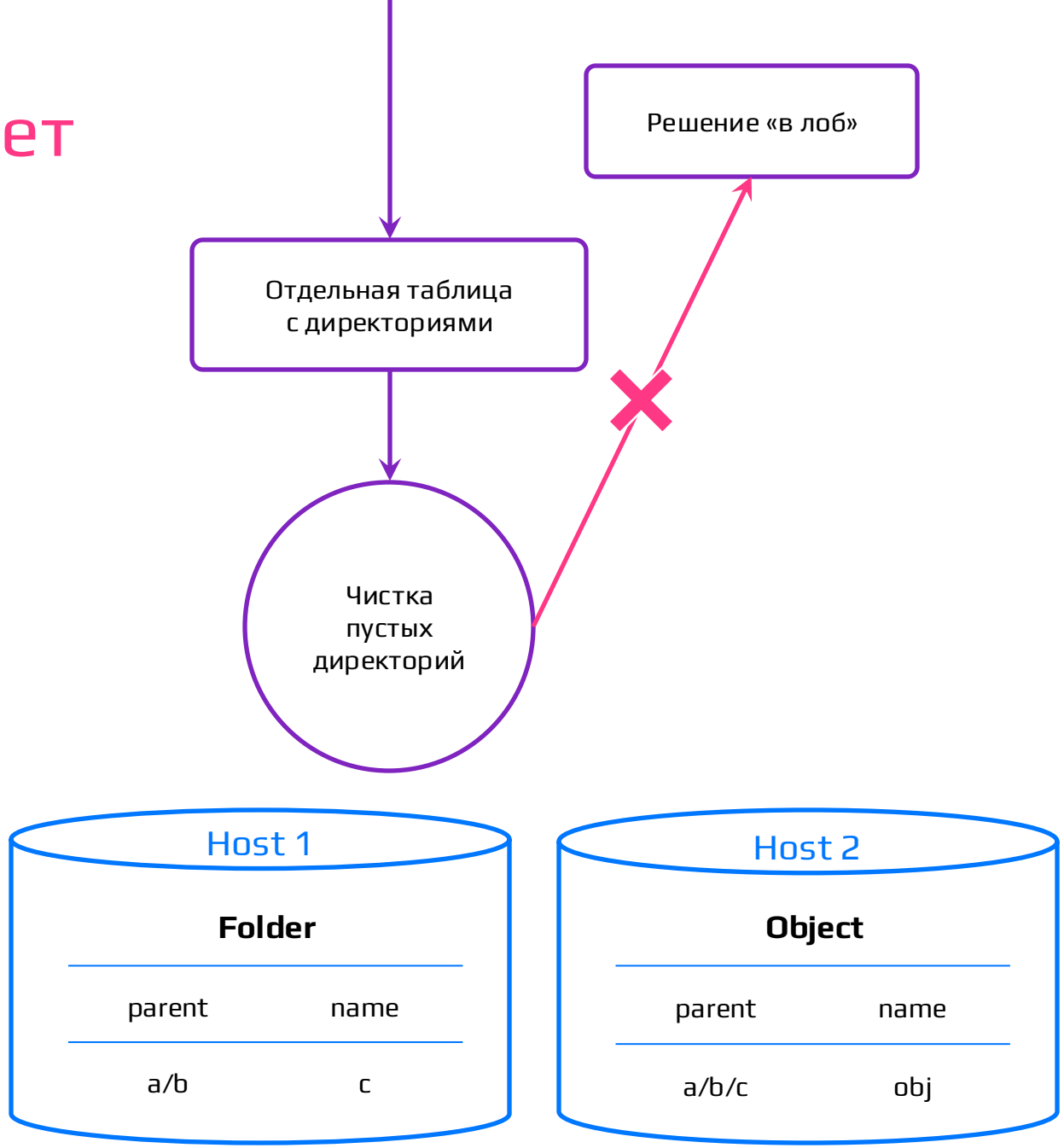
Решение «в лоб» не работает

- **Объекты и директории партиционированы по родительской директории**



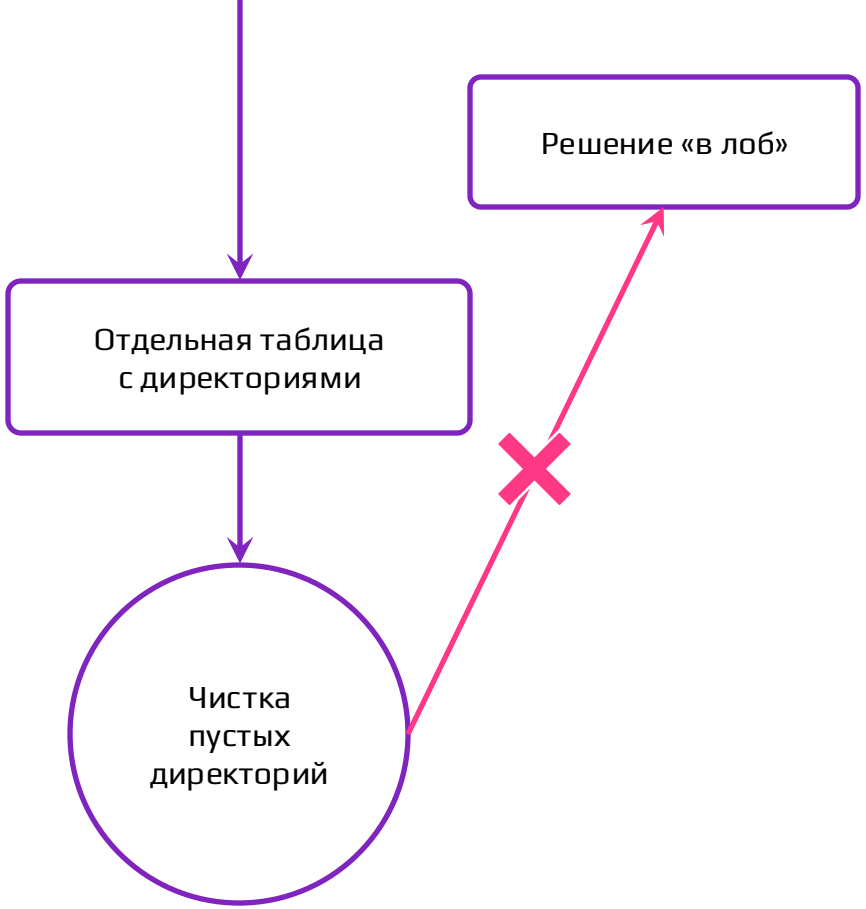
Решение «в лоб» не работает

- Объекты и директории партиционированы по родительской директории
- Пусть существует объект с ключом a/b/c/obj
- Запись с директорией a/b/c и объект a/b/c/obj могут лежать на разных хостах, так как имеют разные родительские директории (a/b и a/b/c соответственно)



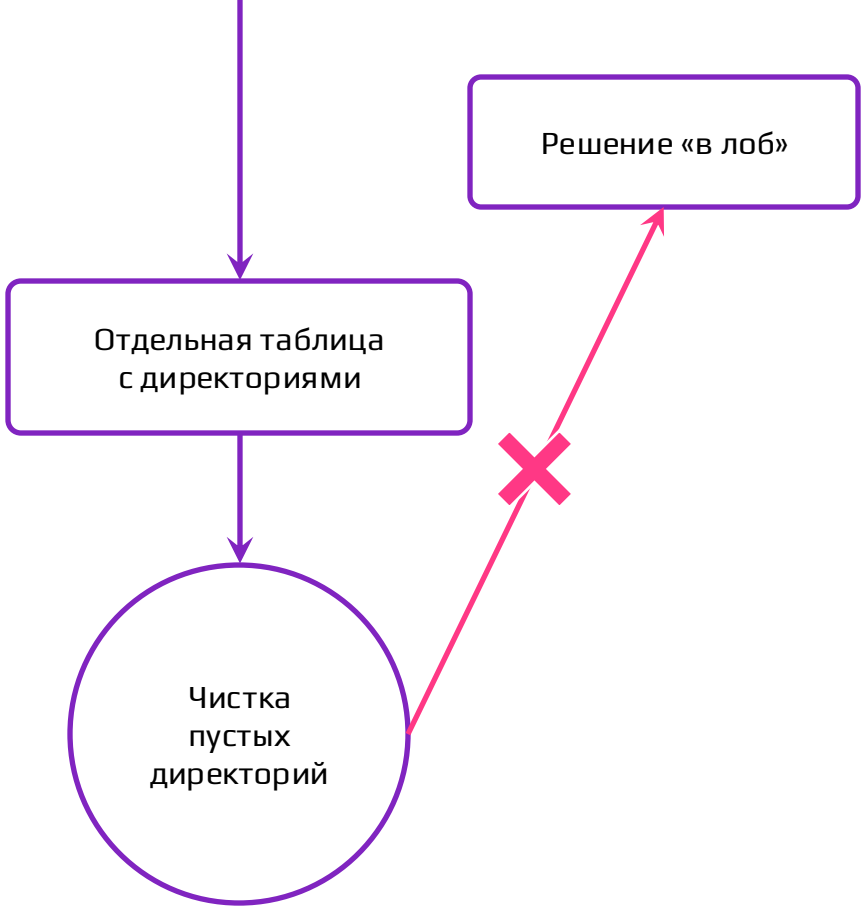
Идея

- **Мы не можем локально определить, пустая директория или нет**
- **Из-за того что объекты директории и она сама лежат на разных хостах**



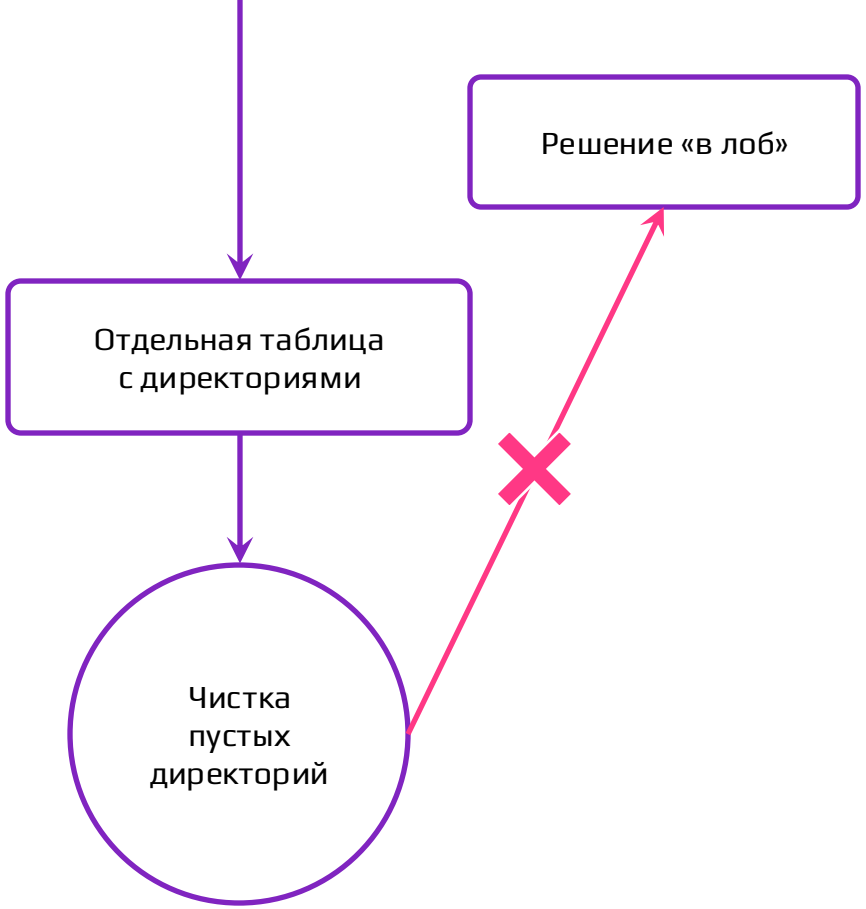
Идея

- Мы не можем локально определить, пустая директория или нет
- Из-за того что объекты директории и она сама лежат на разных хостах
- **Значит, нужно собрать данные об объектах перед сканом директорий**



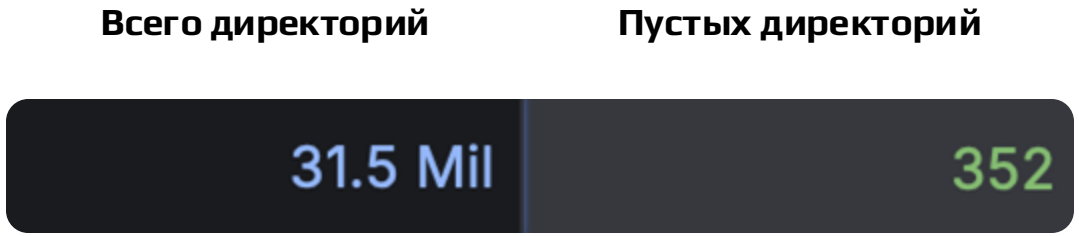
Идея

- Мы не можем локально определить, пустая директория или нет
- Из-за того что объекты директории и она сама лежат на разных хостах
- Значит, нужно собрать данные об объектах перед сканом директорий
- **Чтобы во время скана директорий можно было быстро совершить проверку**



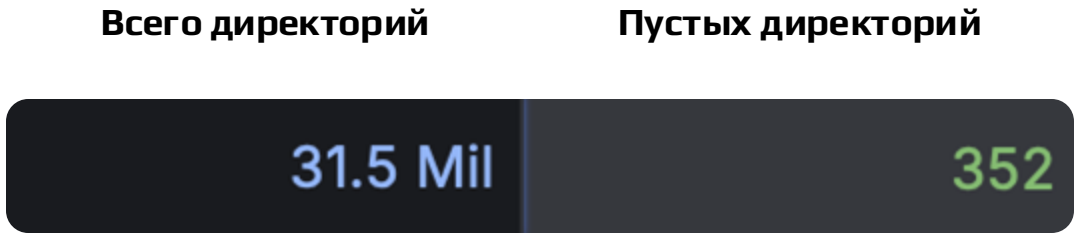
Как хранить информацию

- Нельзя позволить делать запрос для проверки
  (30 миллионов проверок, большинство ложные)
- Все необходимые данные должны помещаться в память

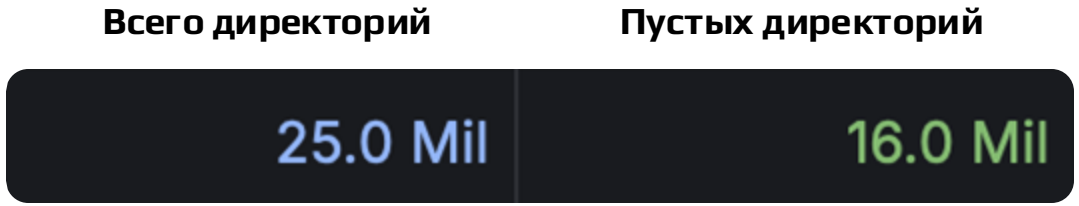


Как хранить информацию

- Нельзя позволить делать запрос для проверки
  (30 миллионов проверок, большинство ложные)
- Все необходимые данные должны помещаться в память



- Нельзя построить хэш-сет (16 миллионов строк)
- Без частичной потери информации не получится



Как хранить информацию

- Нельзя позволить делать запрос для проверки
  (30 миллионов проверок, большинство ложные)
- Все необходимые данные должны помещаться в память

Всего директорий **Пустых директорий**

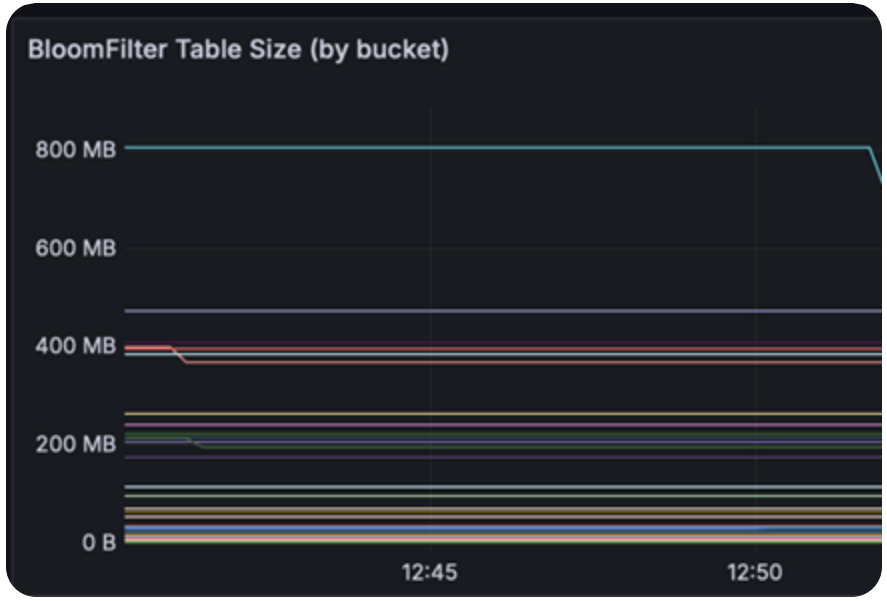
31.5 Mil 352

- Нельзя построить хэш-сет (16 миллионов строк)
- Без частичной потери информации не получится

Всего директорий **Пустых директорий**

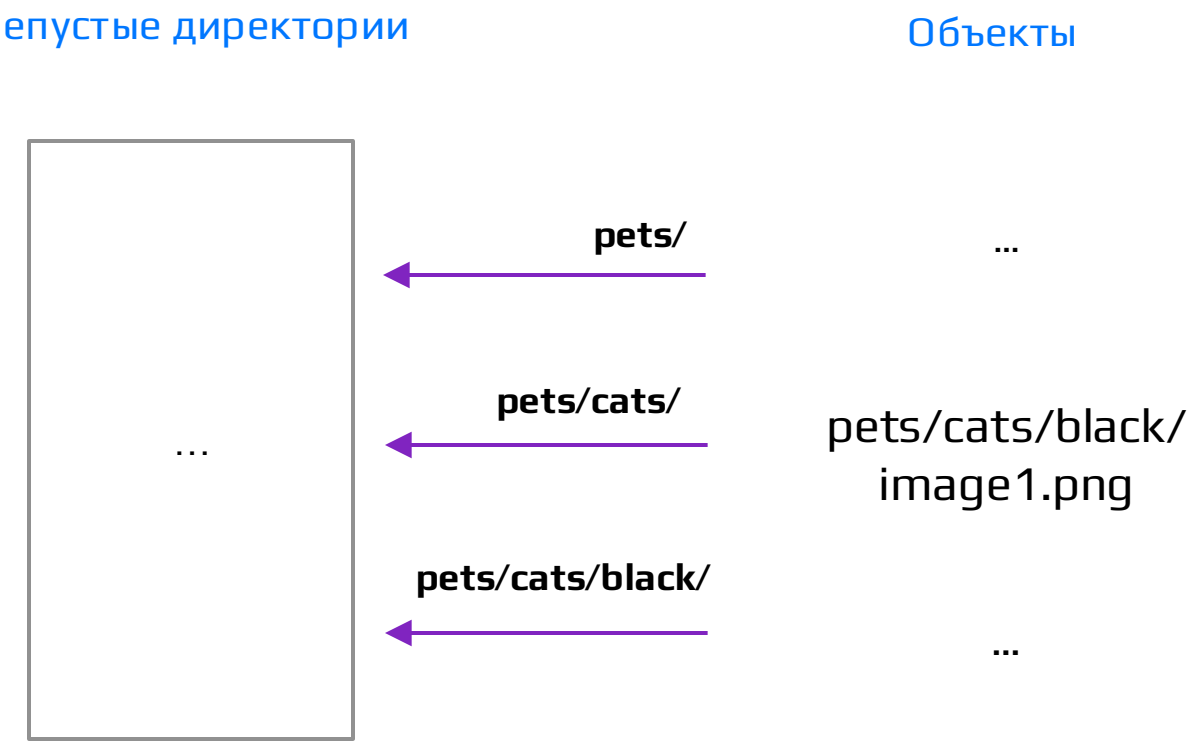
25.0 Mil 16.0 Mil

- Подойдет вероятностное множество
  на основе фильтра Блума



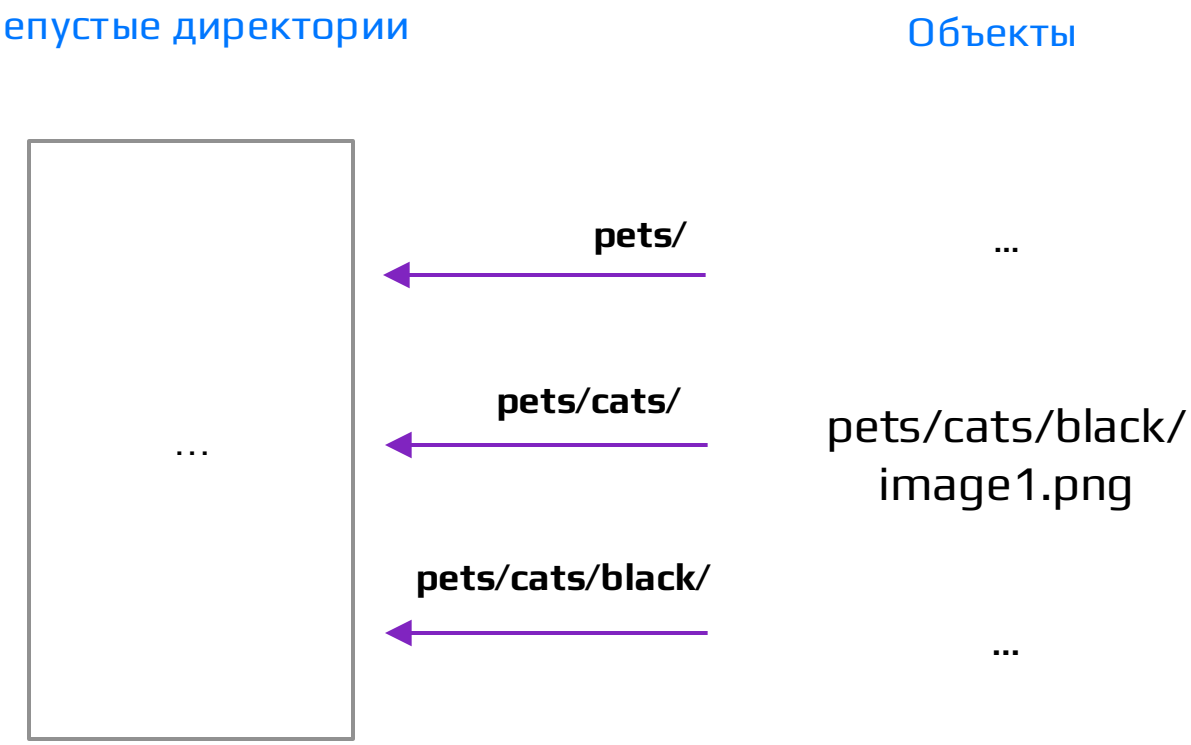
Строим Фильтр Блума

- **Сделаем новую задачу на локальный скан объектов**



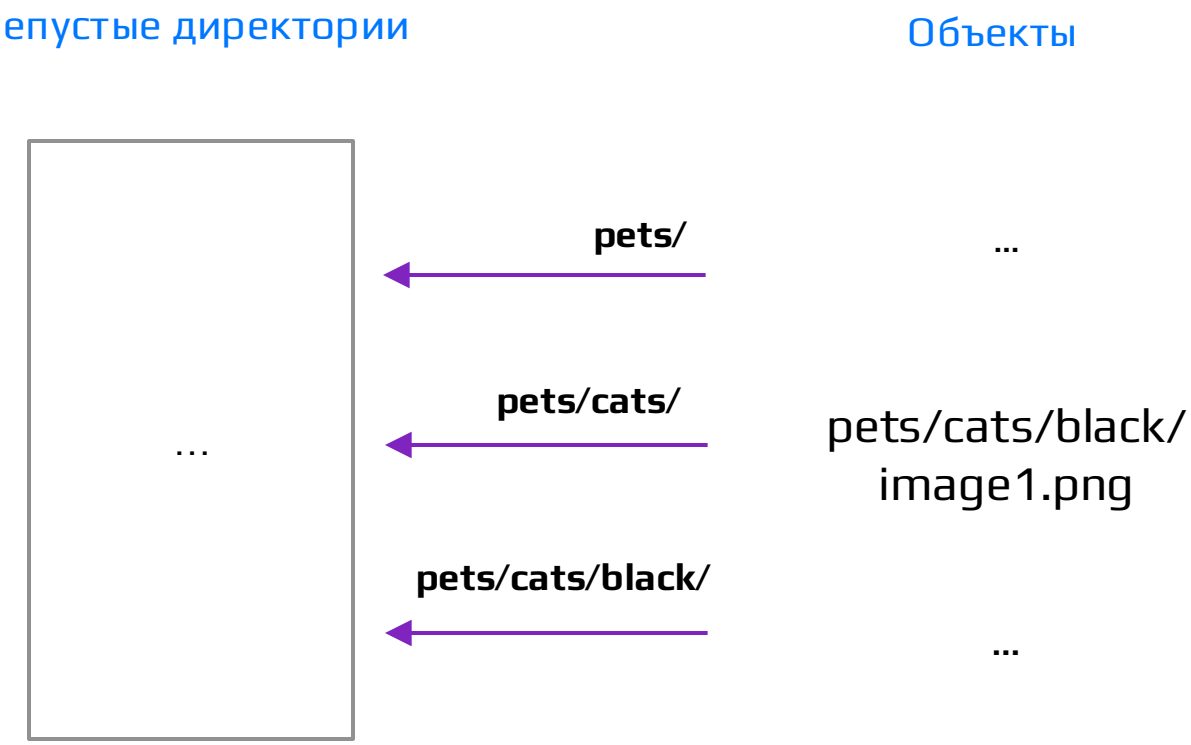
Строим Фильтр Блума

- Сделаем новую задачу на локальный скан объектов
- **Создадим в памяти фильтр Блума**



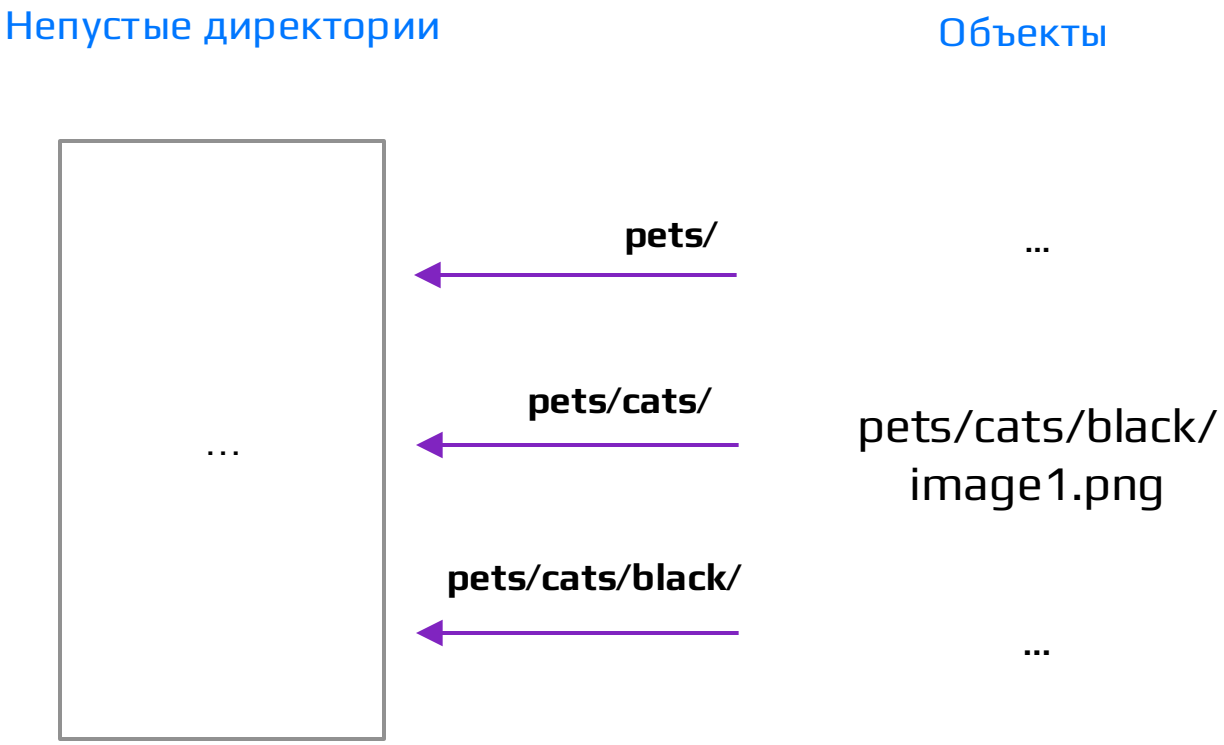
Строим Фильтр Блума

- Сделаем новую задачу на локальный скан объектов
- Создадим в памяти фильтр Блума
- **Размер оценим из общего количество директорий**



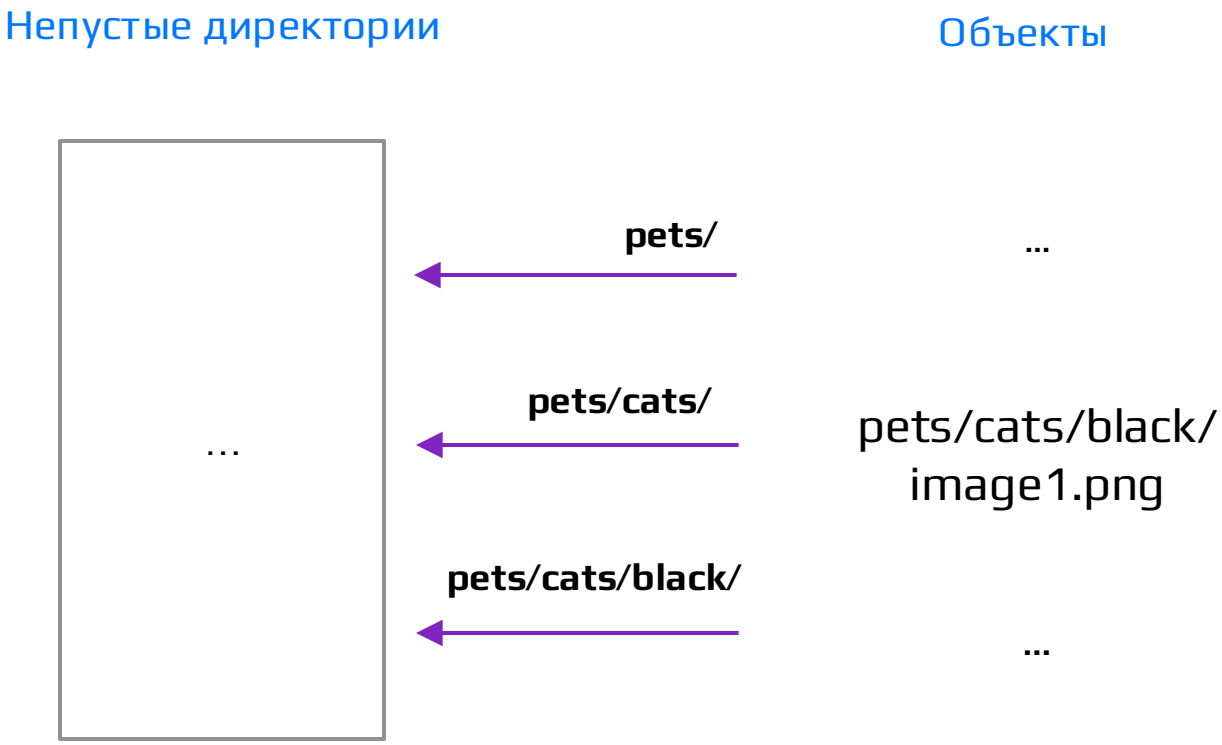
Строим Фильтр Блума

- Сделаем новую задачу на локальный скан объектов
- Создадим в памяти фильтр Блума
- Размер оценим из общего количество директорий
- **Для каждого объекта будем складывать в фильтр все родительские директории**



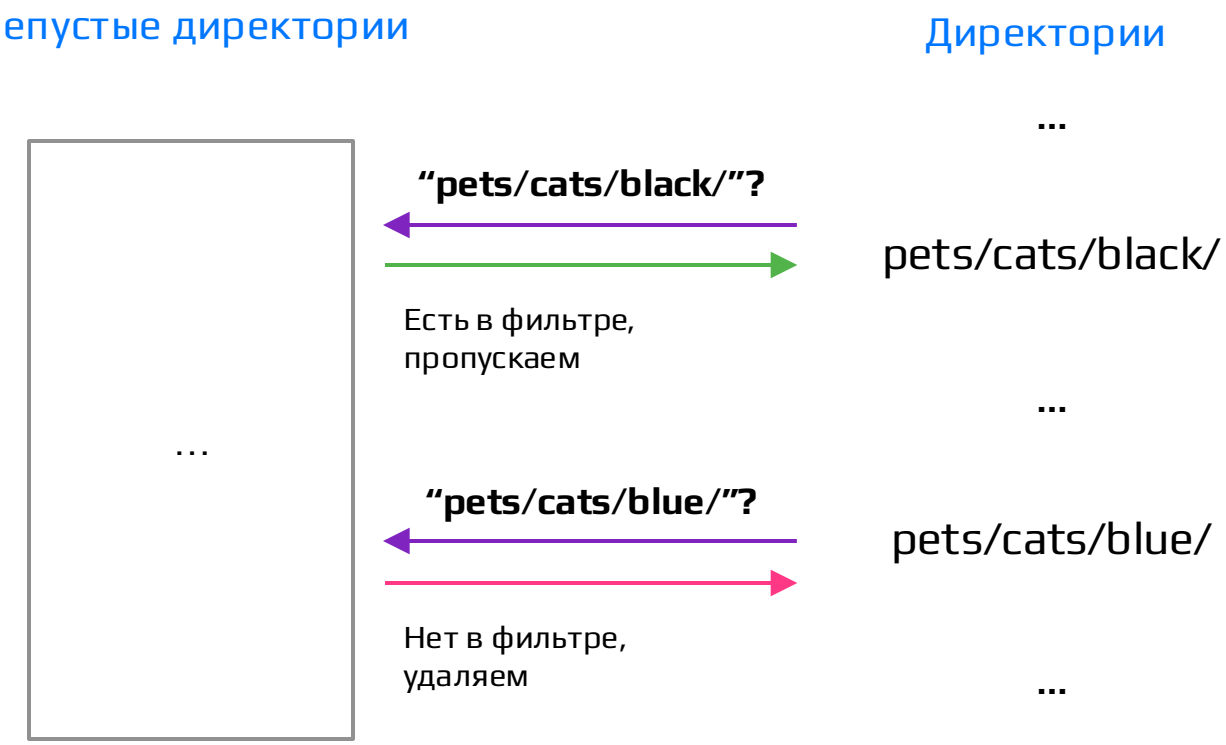
Строим Фильтр Блума

- Сделаем новую задачу на локальный скан объектов
- Создадим в памяти фильтр Блума
- Размер оценим из общего количество директорий
- Для каждого объекта будем складывать в фильтр все родительские директории
- **В конце скана запишем сериализованный фильтр в отдельную таблицу**



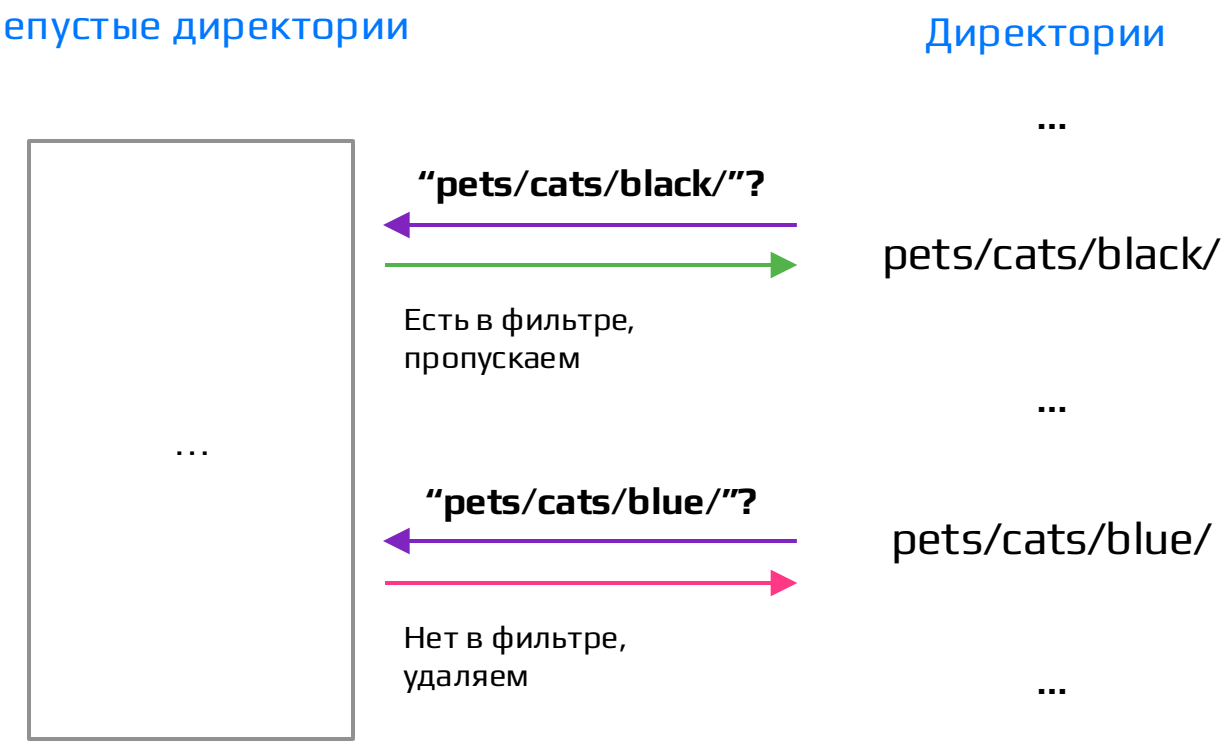
Запускаем чистку

- **Перед сканом директорий прочитаем в память фильтр Блума**



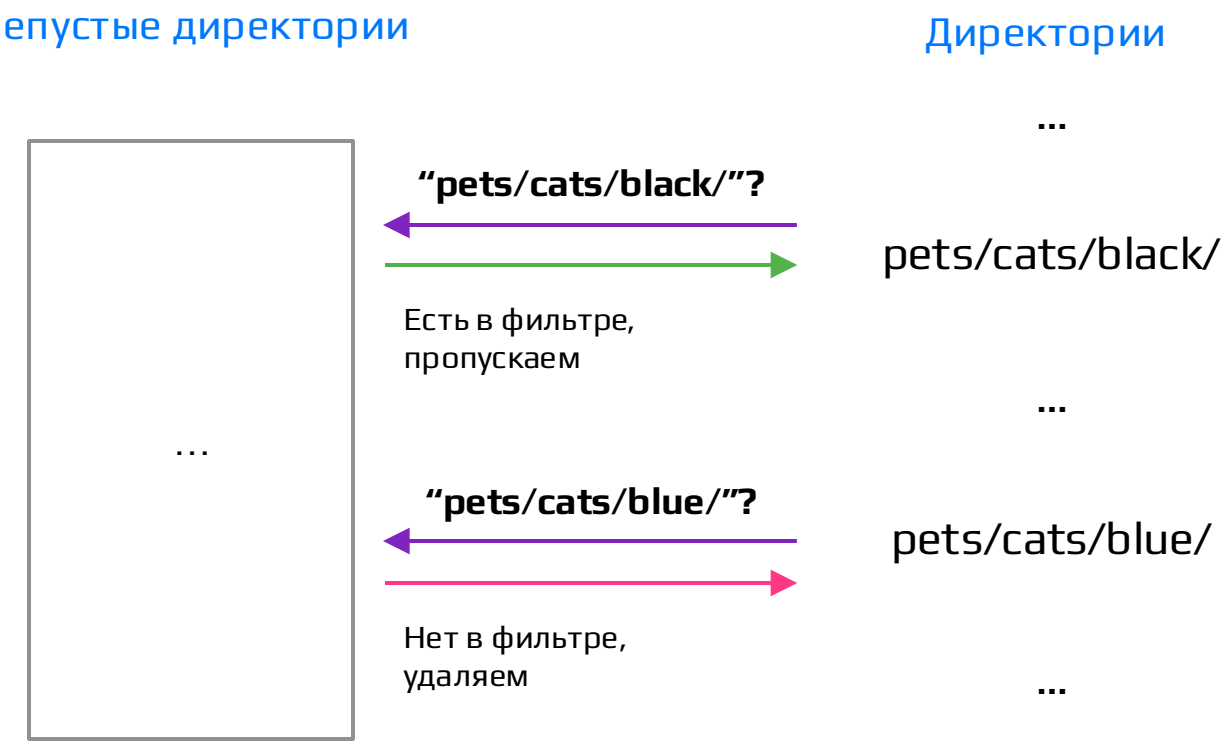
Запускаем чистку

- Перед сканом директорий прочитаем в память фильтр Блума
- **Будем проверять каждую директорию**



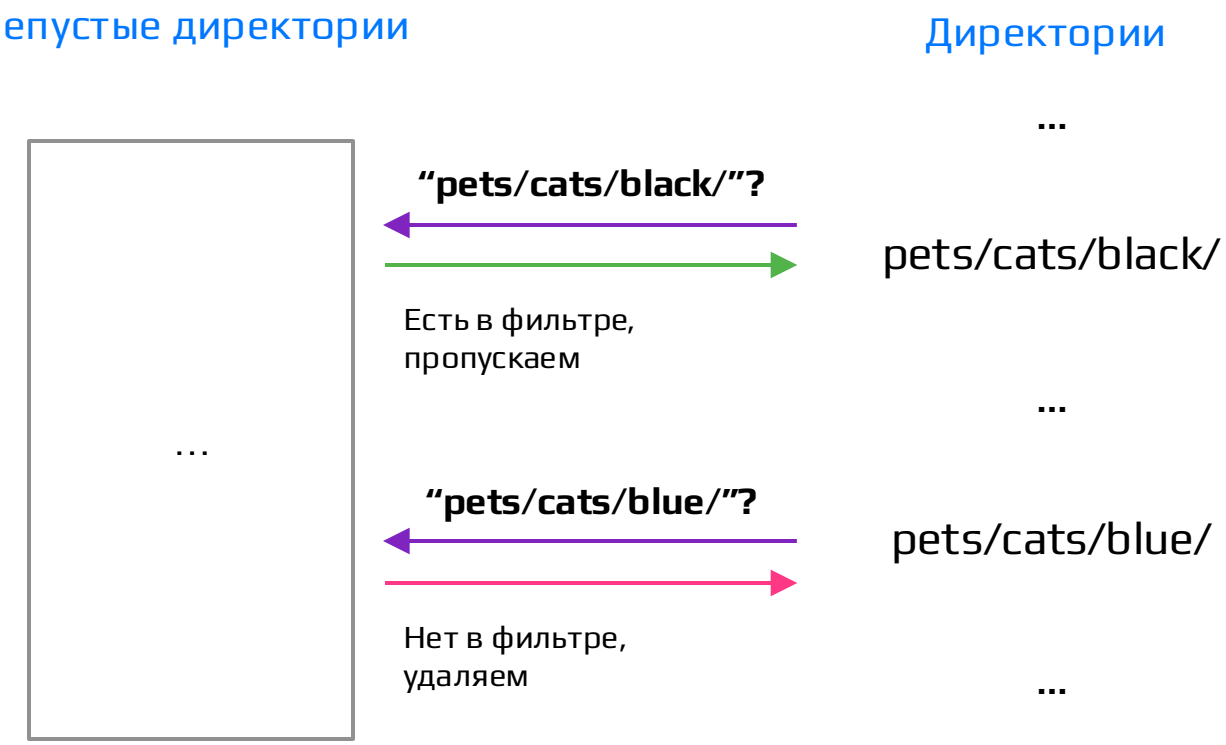
Запускаем чистку

- Перед сканом директорий прочитаем в память фильтр Блума
- Будем проверять каждую директорию
- **Если директории нет в фильтре — кладём в очередь**



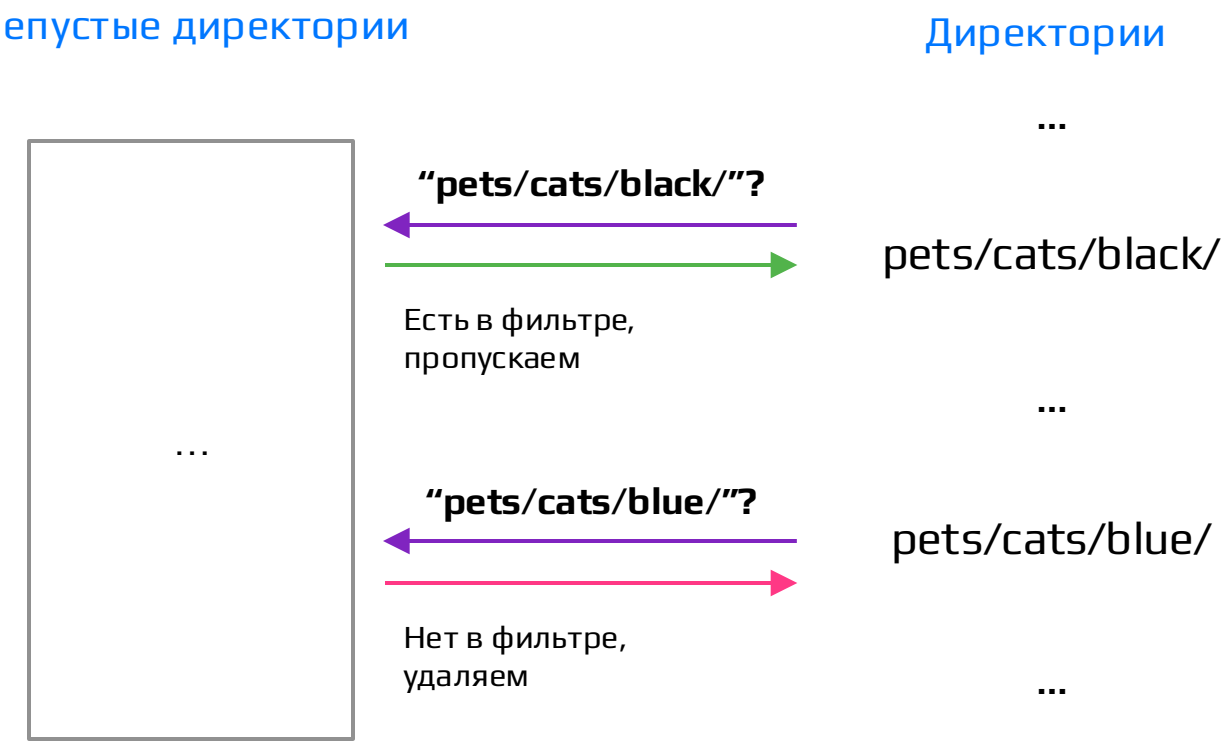
Запускаем чистку

- Перед сканом директорий прочитаем в память фильтр Блума
- Будем проверять каждую директорию
- Если директории нет в фильтре — кладём в очередь
- **В случае false-positive мы пропустим пустую директорию**



Запускаем чистку

- Перед сканом директорий прочитаем в память фильтр Блума
- Будем проверять каждую директорию
- Если директории нет в фильтре — кладём в очередь
- В случае false-positive мы пропустим пустую директорию



Цель — не допустить большого количества пустых директорий. Единичные экземпляры не опасны

Итоги

Резюме

«Обычная» база данных не справлялась с фоновыми задачами

Резюме

- «Обычная» база данных не справлялась с фоновыми задачами
- **Сделали локальный скан**

Резюме

- «Обычная» база данных не справлялась с фоновыми задачами
- Сделали локальный скан
- **Добавили возможность считать статистику в реальном времени**

Резюме

- «Обычная» база данных не справлялась с фоновыми задачами
- Сделали локальный скан
- Добавили возможность считать статистику в реальном времени
- **Научились фоном чистить пустые директории**

Резюме

- «Обычная» база данных не справлялась с фоновыми задачами
- Сделали локальный скан
- Добавили возможность считать стастистику в реальном времени
- Научились фоном чистить пустые директории
- **Уткнулись в ограничения Кассандры в качестве очереди и перешли на Кафку**

Выводы

Остерегайтесь могил
в Кассандре

Выводы

Остерегайтесь могил
в Кассандре

Иногда использование
антипаттернов
может быть оправдано

Выводы

Остерегайтесь могил
в Кассандре

Иногда использование
антипаттернов
может быть оправдано

Вероятностные структуры
данных могут сэкономить очень
много ресурсов,
если точность не так важна

Выводы

Остерегайтесь могил
в Кассандре

Иногда использование
антипаттернов
может быть оправдано

Вероятностные структуры
данных могут сэкономить очень
много ресурсов,
если точность не так важна

Схема партиционирования
задает ограничения всей
системе

Выводы

Остерегайтесь могил
в Кассандре

Иногда использование
антипаттернов
может быть оправдано

Вероятностные структуры
данных могут сэкономить очень
много ресурсов,
если точность не так важна

Схема партиционирования
задает ограничения всей
системе

В распределенных системах
не бывает правильных
решений — только
компромиссы

Спасибо!

Данил Кислов

Разработчик хранилищ
для внутреннего облака One-cloud