FIREBIRD CONF'26

Организатор конференции РЕДСОФТ

Артём Абакумов

Как мониторить СУБД не привлекая внимание сисадминов

В начале была мечта

Узнавать о проблемах сразу и без жонглирования утилитами

Надо получать

- ОЗУ
- Диски/рейды
- Запросы
- Логи
- Подключения
- Нагрузку на процессор

Требования

- Не объесться ОЗУ
- Не нагружать процессор
- Не влиять на СУБД
- Не нагружать диск

РБД МОНИТОР

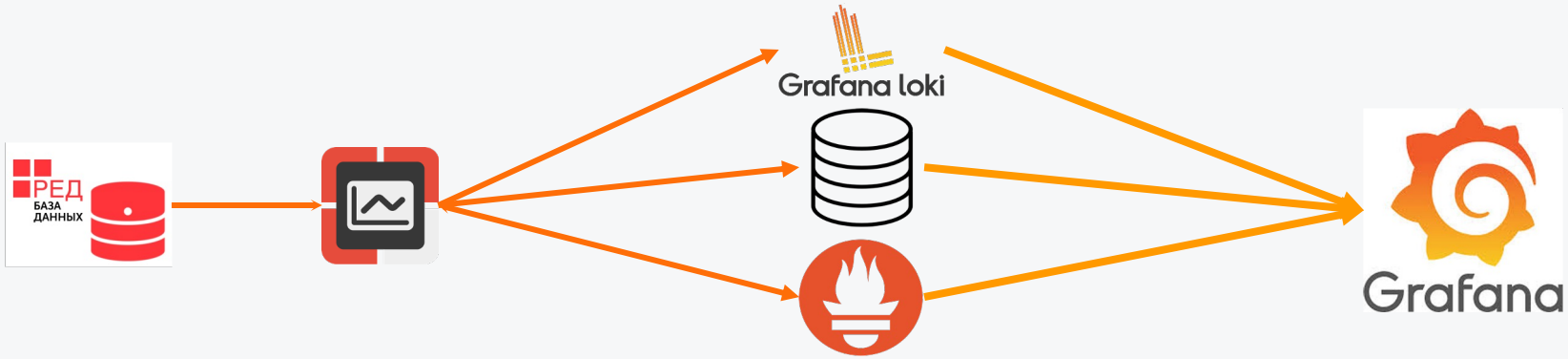
Мониторим БД под высокой нагрузкой

Бэкэнд

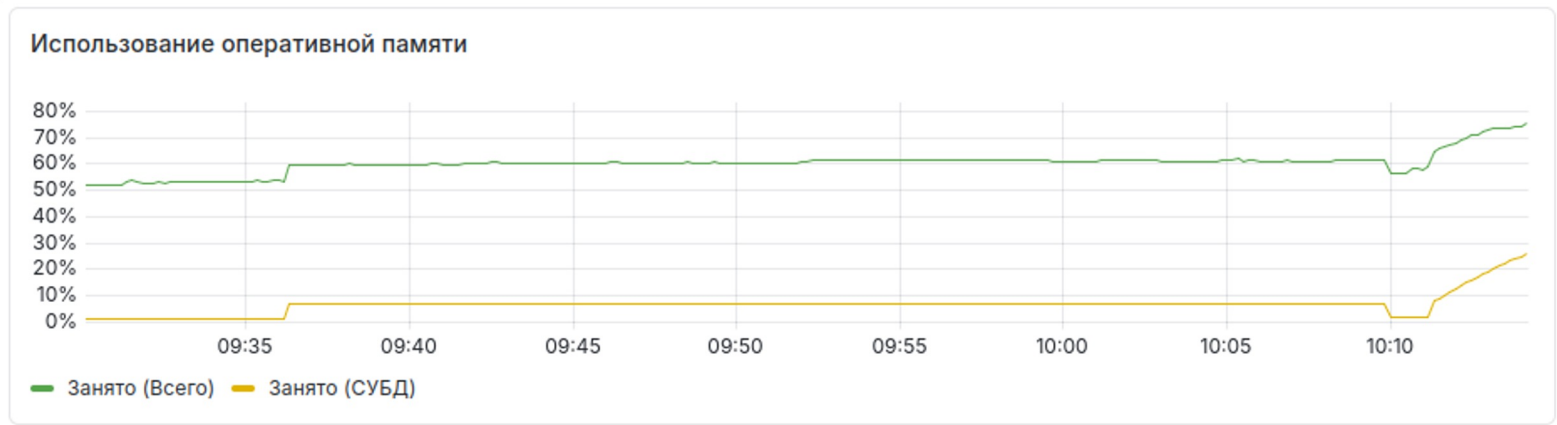
СУБД - Firebird/РЕД База Данных
БД – то, что мониторим
Агент - собирает и отправляет метрики

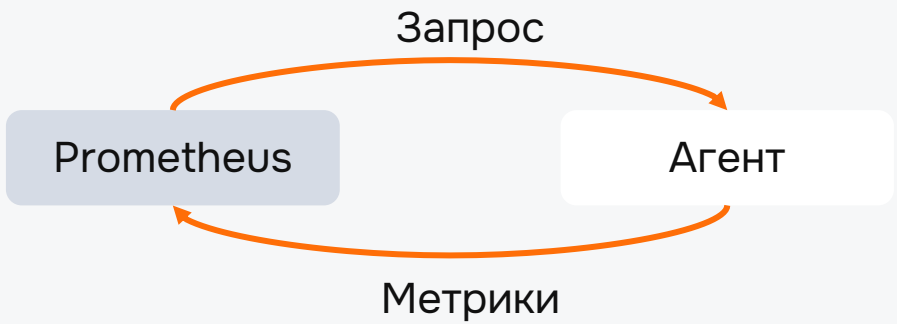
Фронтэнд

Grafana - рисуем красивые графики
Prometheus - запрашивает и хранит метрики
loki - получаем логи
БД справочник - храним запросы и их планы



Временные ряды





Метрика	Значение	Время
rdb_memory_usage{"type"="total"}	60%	10:00
rdb_memory_usage{"type"="rdb"}	10%	10:00
rdb_memory_usage{"type"="total"}	18%	10:05
rdb_memory_usage{"type"="rdb"}	10%	10:05
rdb_memory_usage{"type"="total"}	50%	10:10
rdb_memory_usage{"type"="rdb"}	5%	10:10

А как ловить SQL запросы?

Таблицы мониторинга	Аудит (трейс)	Свой механизм
Плюсы • Готовый механизм • Легко получить данные	**Плюсы** • Готовый механизм • Можно запускать несколько сессий, в том числе удалённо • Есть текстовый и бинарный форматы	**Плюсы** • Ничего лишнего, полный контроль
Минусы • Тяжело с точки зрения производительности • Можно упустить запросы	**Минусы** • Парсить текстовый формат - медленно • Бинарный формат есть только в РЕД Базе Данных	**Минусы** • Непонятно, как сделать правильно

Что нужно знать о запросах?

Нужно знать:

- Текст запроса
- План запроса (если есть)
- Время
- Счётчики страниц
- Доп. счётчики (память, выделенная на сортировку)
- Изменение плана

Изменение всего этого во времени

Начало разработки

2023

РБД Монитор 0.1	РБД Монитор 0.2	РБД Монитор 0.3	РБД Монитор 0.4
Запросы через таблицы мониторинга	Запросы через парсинг текстового трейса		Запросы через свой, самописный плагин трейса
! Получилось медленно и неэффективно! Требуются постоянные повторяющиеся запросы к таблицам	! Получилось медленно и неэффективно! Накладные расходы на формирование текста, отправку и его парсинг		Ничего лишнего

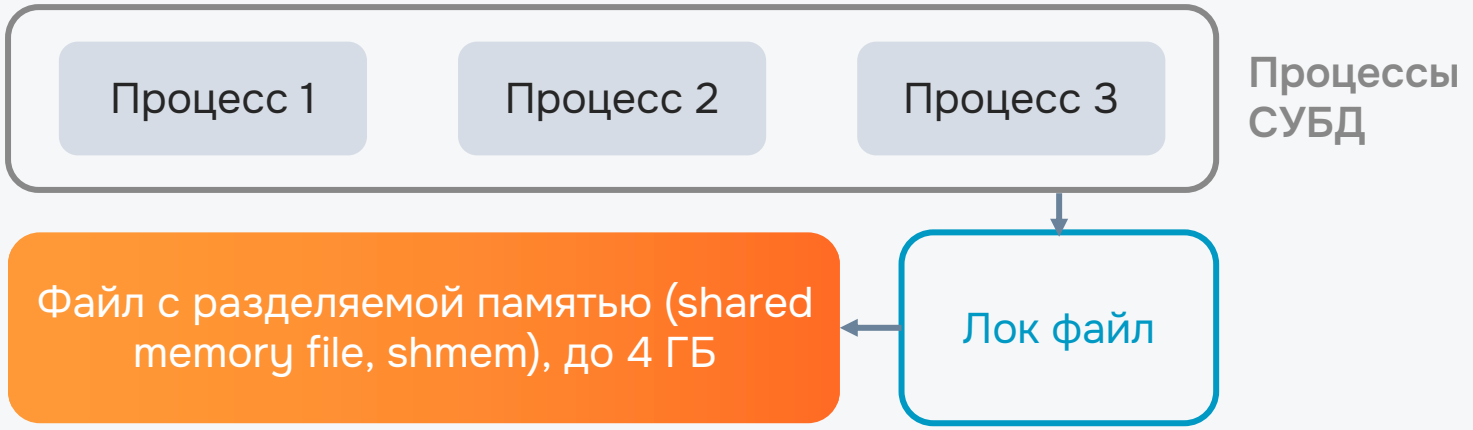
Как работает аудит в Firebird/РЕД База Данных

Плагин aggtrace - не просто собираем, но и агрегируем

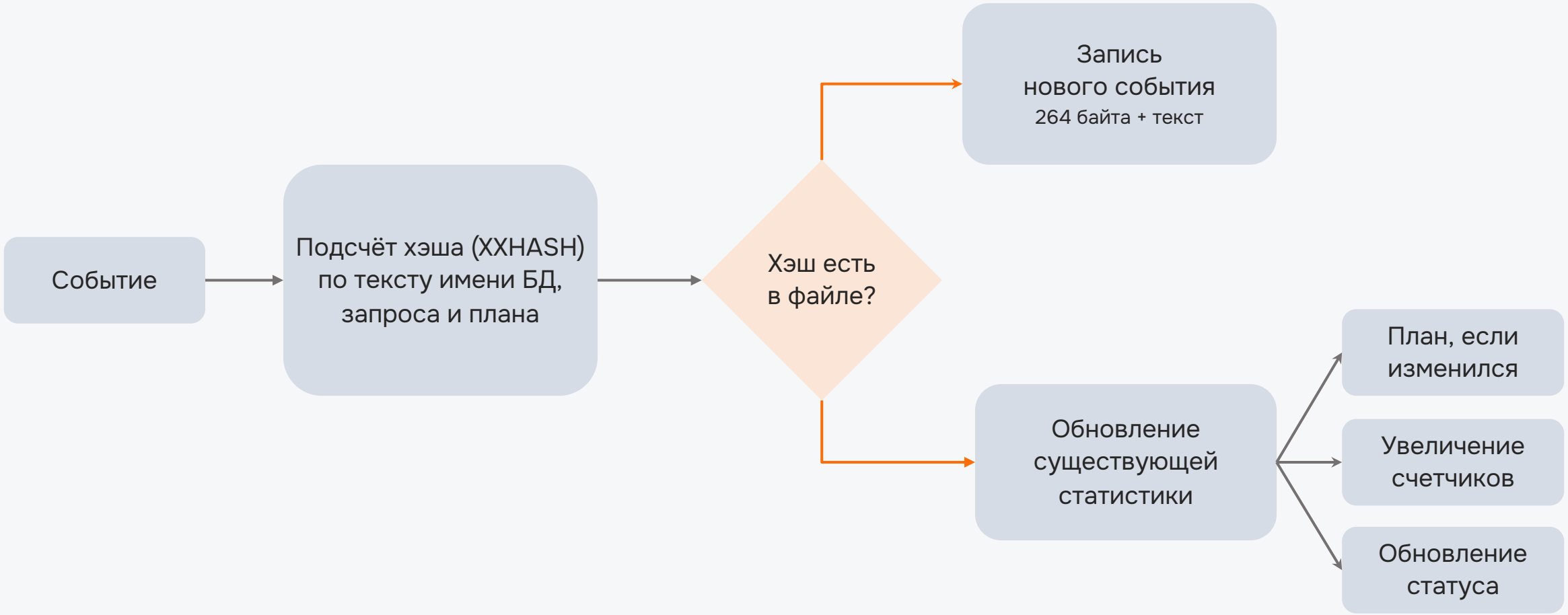
Новый плагин аудита

1. Ядро отправляет запрос в плагин
2. Плагин агрегирует статистику запроса по его по тексту, имени БД и плану.
3. Запрос и статистика хранятся в файле в бинарном формате (avg, min, max, count)
4. Выводим статистику по требованию

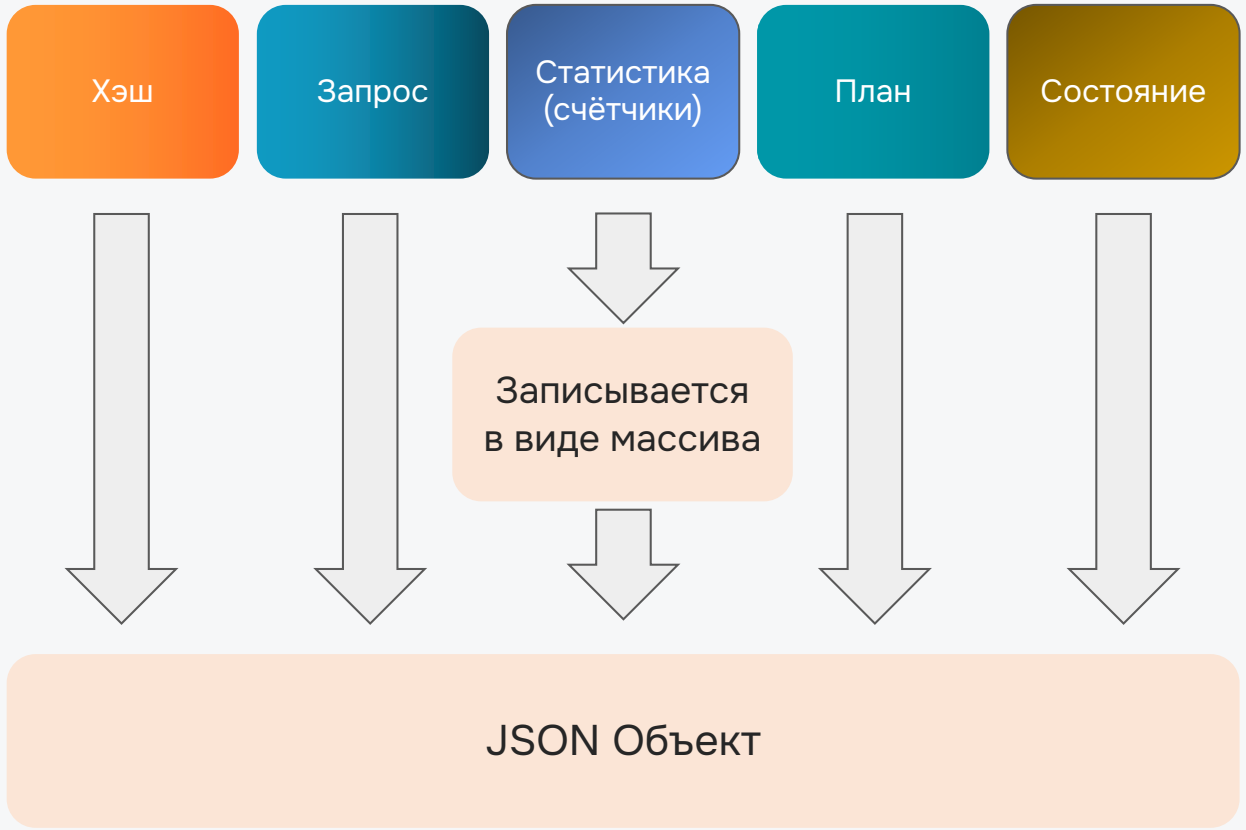
Всё просто... если забыть про архитектуру Классик. Нужна синхронизация между процессами

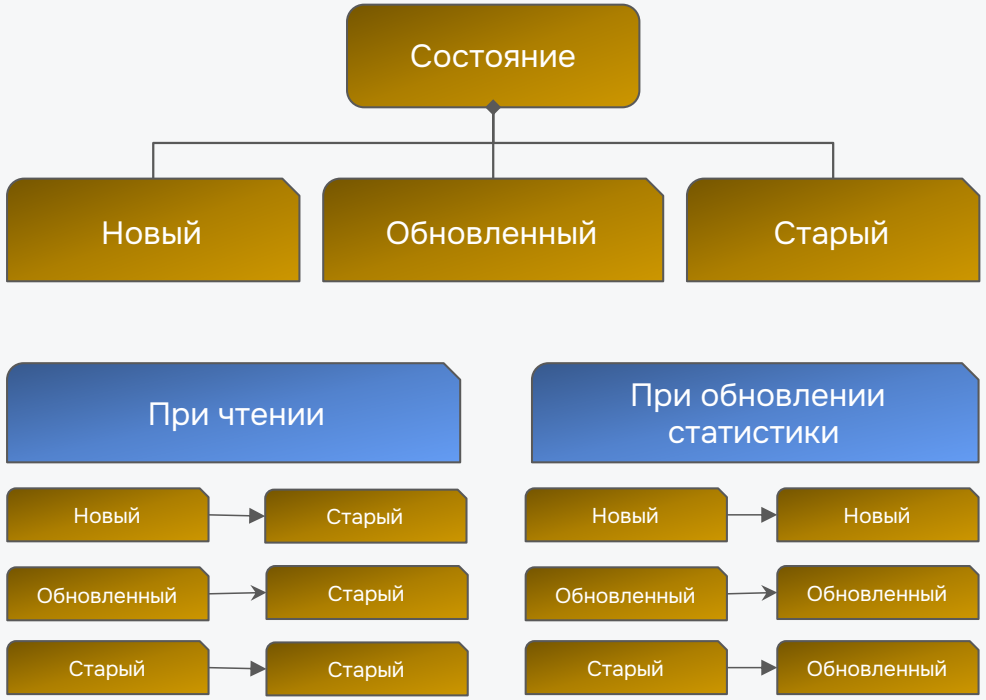


Архитектура плагина aggtrace 1.0



Формат выходных данных о запросе





High Load ошибок не прощает, или что пошло не так

1) Универсальность - парсить JSON долго
2) Синхронизация на Классике - лок-файлы стали узким местом
3) Плагин в составе СУБД – для обновления плагина приходилось обновлять СУБД
4) Место быстро заканчивалось из-за больших текстов. Хранить обрезанный вариант - бесполезно
5) Оверхэд из-за синхронизации кэша
6) Мьютексы (на Супере)
7) На Классике приходится полностью перечитывать файл для каждого соединения
8) Сложно синхронизировать расширение разделяемого файла

Что получилось

1) Разделяемая память

эффективно и быстро
для Interprocess Communication

2) Агрегация

метрики сразу готовы
для экспорта

3) Хэширование через XXHASH

быстро и эффективно

4) Свой плагин

всё ещё быстрее
остальных подходов

aggtrace 2.0 или как всё исправить

2025

РБД Монитор 0.10

Плагин

- Вынести из состава СУБД
- Для каждой БД - создавать свой разделяемый файл
- Кэшировать конфиги и перечитывать их при изменении

Разделяемая память

- Убрать ненужные метрики (min, max, avg) , чтобы сократить использование памяти
- Разделить разделяемый файл на две секции - с текстами и со статистикой

Агент

- Хранить кэш на стороне агента
- Вычитывать тексты, чтобы переиспользовать секцию
- Читать разделяемый файл напрямую из агента

aggtrace 2.0


Разделяемая память
Плагин aggtrace
Буфер текстов
Статическая секция
Если места для записи текста не хватило - ставим флаг и пробуем записать текст в следующий раз
Агрегатируемая статистика
Расширяемая секция
104 байта на запрос
Читаем и очищаем
Читаем и кэшируем
Агент

Что получилось

1. Отдельная секция для текстов
2. Вычитывание текстов
3. Отдельный файл на каждую БД

Что пошло не так

1. Тяжёлая синхронизация между агентом (Python) и плагином СУБД
2. Блокировки из-за локов и мьютексов
3. Без кода СУБД приходится реализовывать парсеры и прочие вещи вручную
4. Сложная настройка (много конфигов)
5. Python медленно читает и формирует метрики
6. Python медленно читает и отправляет тексты в БД
7. Файл рано или поздно переполнялся

А что если нам агрегировать не в плагине - messagetrace (aggtrace 3.0)

2025 | 2026

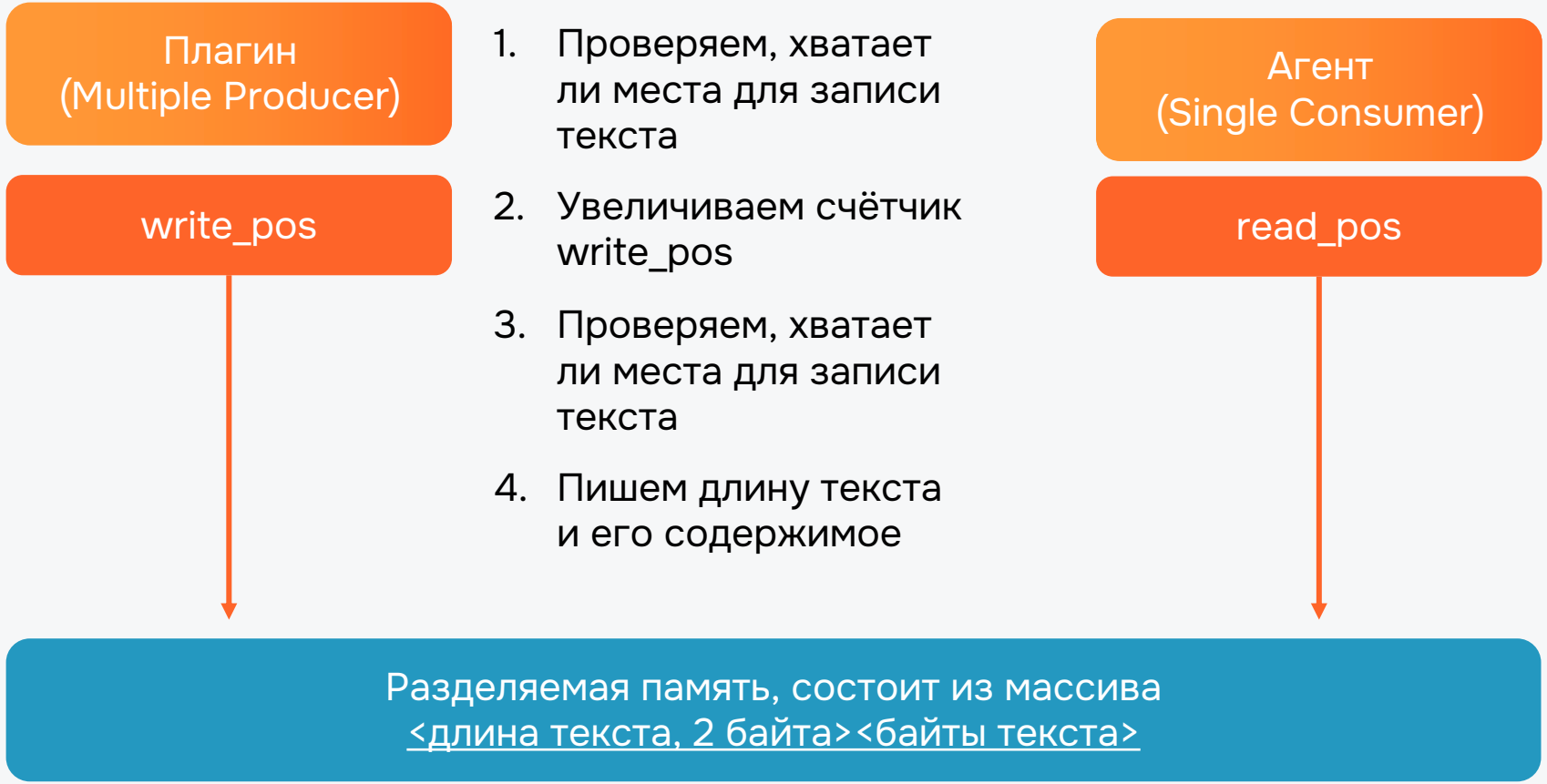
РБД Монитор 0.10 | **РБД Монитор 0.11**

Плагин	Разделяемая память	Агент
- Передавать статистику через сокет (ZeroMQ) - надёжно, без блокировок и без переполнения файла - Передавать тексты по прежнему через разделяемую память - Настройка плагина через отдельный конфиг	- Использовать tempfs, а не просто разделяемый файл - lock free для синхронизации чтения/записи текстов	- Агрегировать на стороне агента - Написать библиотеку на C++ для агента, чтобы использовать одинаковый с плагином код - Вставлять тексты в справочник сразу после чтения - Сразу формировать метрики в формате OpenMetrics (Prometheus) - Двойной буфер при получении/отправке статистики

Запись статистики и текстов

Проблема	Решение
Разделяемая память переполняется	Передаём статистику запросов через сокет и агрегируем на стороне агента
Без статистики на стороне плагина непонятно: отправляли уже текст или нет	Используем фильтр Блума - вероятностную структуру данных типа Множество (set)
У фильтр Блума фиксированный размер. Чем больше элементов, чем чаще ошибается	Используем два фильтра и меняем их раз в день

lock free запись и чтение текстов



1. Проверяем, хватает ли места для записи текста
2. Увеличиваем счётчик write_pos
3. Проверяем, хватает ли места для записи текста
4. Пишем длину текста и его содержимое

1. Читаем длину текста
2. Если длина больше 0 и не выходит за границы файла и write_pos, то читаем текст
3. Увеличиваем read_pos
4. Если read_pos больше половины файла – очищаем первую половину
5. Сдвигаем указатель write_pos на начало
6. Читаем остаток со второй половины

Настройка

firebird.conf

```
TracePlugin = fbtrace, messagetrace
```

plugins.conf

```
Plugin = messagetrace
{
  Module = /opt/rdbmonitor/messagetrace
  Config = monitor_config
}
Config = monitor_config {
  ConfigFile = /opt/exporter_conf.json
}
```

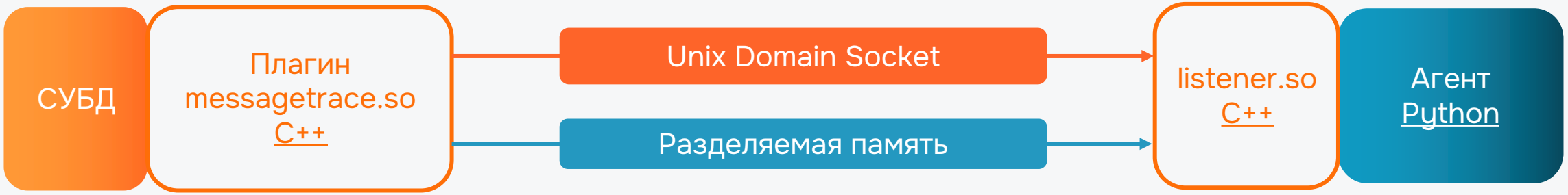
Конфиг трейса
(через пользовательскую сессию)

```
#MESSAGETRACE - ставим специальный тэг
# - это знак комментария для обычного трейса
database
{
 # обычный трейс не запустится
  enabled = false
}
```

Что получилось

1. Успеваем парсить файл с разделяемой памятью без переполнений
2. lock free при чтении/записи текстов
3. Минимум кода в плагине - минимум шанса уронить СУБД
4. Единый конфиг для агента и плагина
5. Вся тяжёлая работа написана на C++
6. Python только отсылает метрики
7. Агент и СУБД работают независимо
8. Фильтр Блума

Текущая архитектура



Последняя проблема - объём метрик

2026

РБД Монитор 1.0

Prometheus

Склеенные запросы (без параметризации):

- Prometheus потребляем много ресурсов при агрегации больших интервалов
- Видим некорректную статистику
- Много временных рядов (52 КБ памяти на 1 запрос)

Плагин

- Добавить нормализацию запросов:

select * from mytable where id = **1**

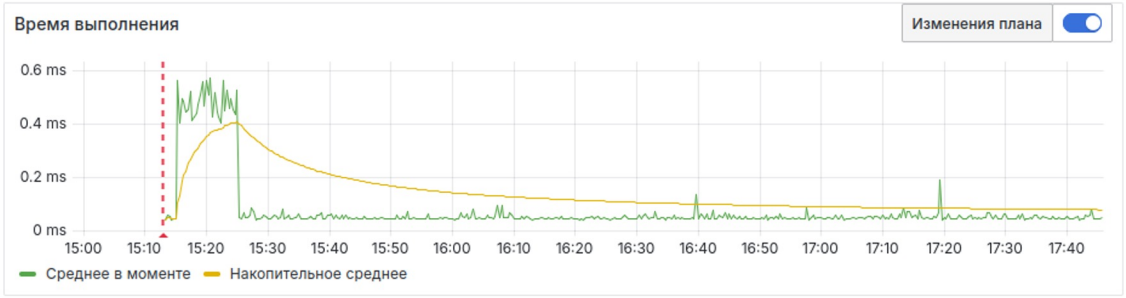
select * from mytable where id = ?

- Считает потоковый хэш только по частям без констант
- Нет аллокаций памяти
- Менее 1 микросекунды для парсинга и подсчёта хэша

Итог разработки
Версия 0.4 -> 1.0

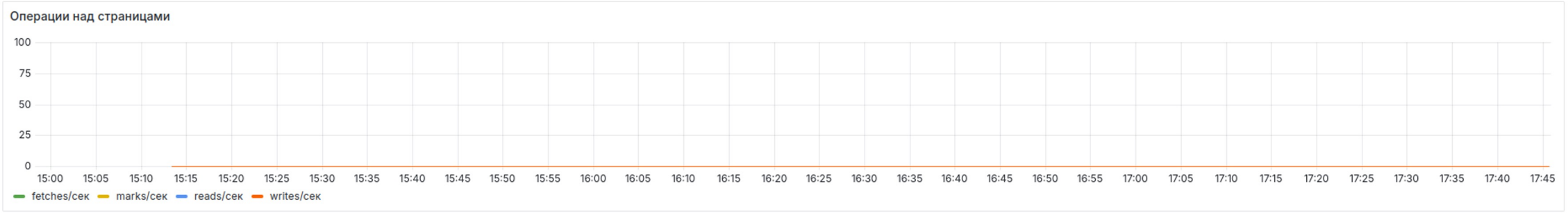
1. Мониторим все запросы круглосуточно
2. Работает с РБД и ФБ
3. Кроссплатформенно
4. Размер разделяемой памяти 4 гб -> 128 МиБ
5. Память Prometheus: 64 ГиБ -> 9 ГиБ (данные хранятся за 2 недели)
6. Кол-во запросов: 13921133 -> 9471 (на БД размером 3.6 ТБ, ~100 подключений)
7. Оверхэд плагина СУБД - меньше 0.8 мс для prepare и ~0.04 мс для execute (для запроса длиной 34 КиБ и его плана)
8. Отслеживаем изменение плана

Фронтенд



Выбранное изменение

```
Select Expression
    -> Aggregate
        -> Sort (record length: 172, key length: 8)
            -> Filter
                -> Hash Join (inner)
                    -> Table "MON$RECORD_STATS" as "RS" Full Scan
                    -> Record Buffer (record length: 33)
                        -> Table "MON$STATEMENTS" as "ST" Full Scan
                    -> Record Buffer (record length: 41)
                        -> Table "MON$MEMORY_USAGE" as "MU" Full Scan
                    -> Record ...
```




FIREBIRD CONF'26
Организатор конференции
РЕДСОФТ
Спасибо за внимание!

Участвуйте в развитии РБД Монитора

Бета-тестирование

Регистрация на **вебинар**