# YDB Topics: как мы повышали производительность очереди сообщений

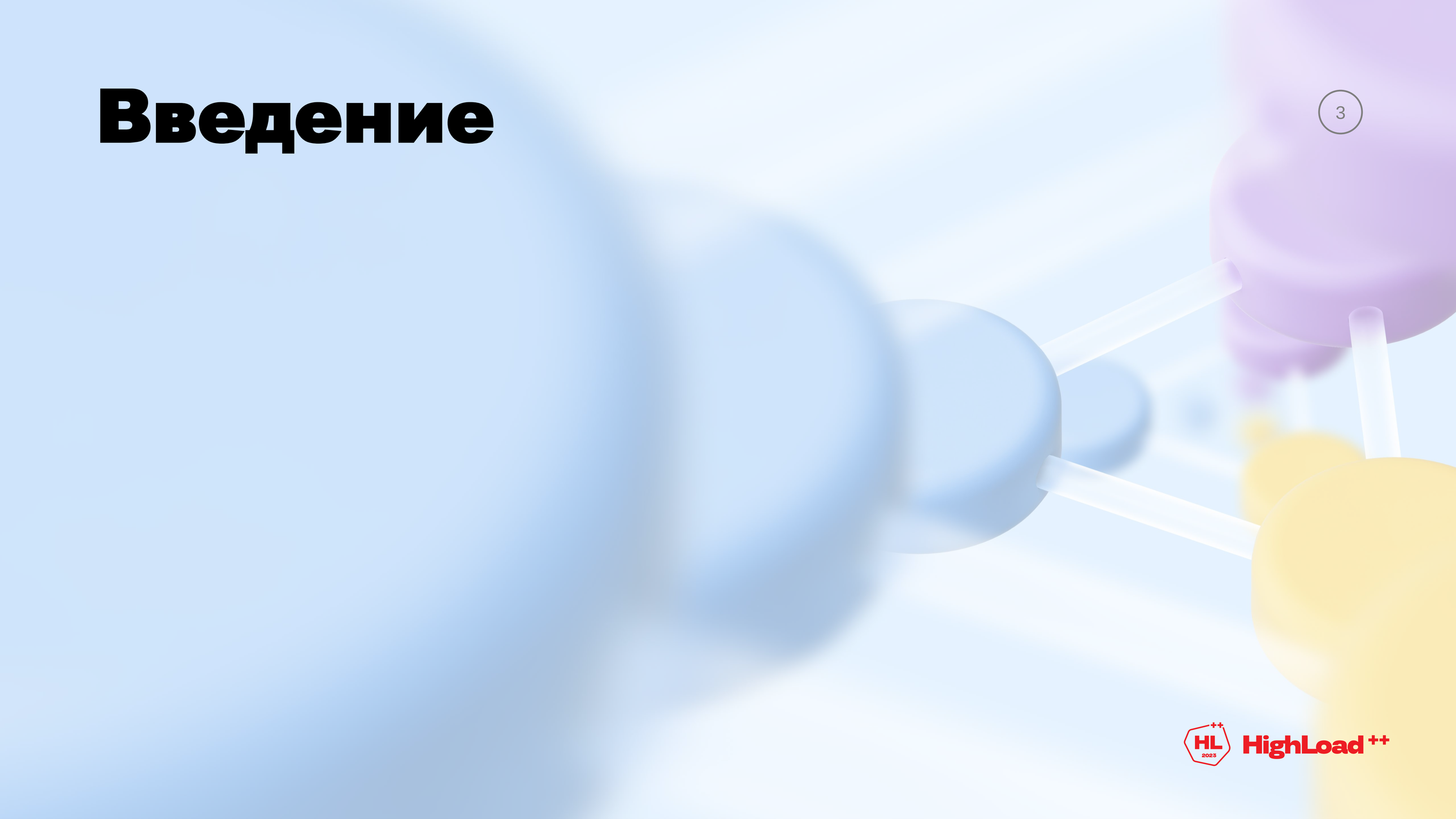
Зевайкин Александр,
технический лидер,
кандидат технических наук, доцент

HL 2023 HighLoad++

# Содержание

# Введение





# Что такое YDB Topics



- Масштабируемый сервис для надёжной передачи упорядоченных потоков сообщений
- Позволяет приложениям обмениваться сообщениями через очереди по модели pub/sub

**2013** — Production Яндекса поверх Kafka

**2017** — Production Яндекса поверх YDB

**2022** — Выведен в опенсорс

# Платформа YDB

- Распределённый консенсус
- Распределённый слой хранения данных
- Горизонтальное масштабирование, вплоть до тысяч узлов
- Катастрофоустойчивость и автоматическое восстановление после сбоев
- Работа в однодатацентровом и георасределённом режимах, в том числе в Yandex Cloud в режиме Serverless

# Компоненты платформы YDB



- Распределённый слой хранения
- Движок транзакций
- Единый язык запросов YQL
- OLAP-таблицы
- OLTP-таблицы
- Key-Value-хранилище
- Федеративные запросы
- Очереди сообщений

# Немного цифр о YDB Topics в Яндексе



Скорость передачи: запись + чтение

×200 Гбайт/с
80+120
>300 на пике

×21 млн EPS
6+15

5 ДЦ

1
дежурный SRE

1000
команд

30k
топиков

1500
серверов

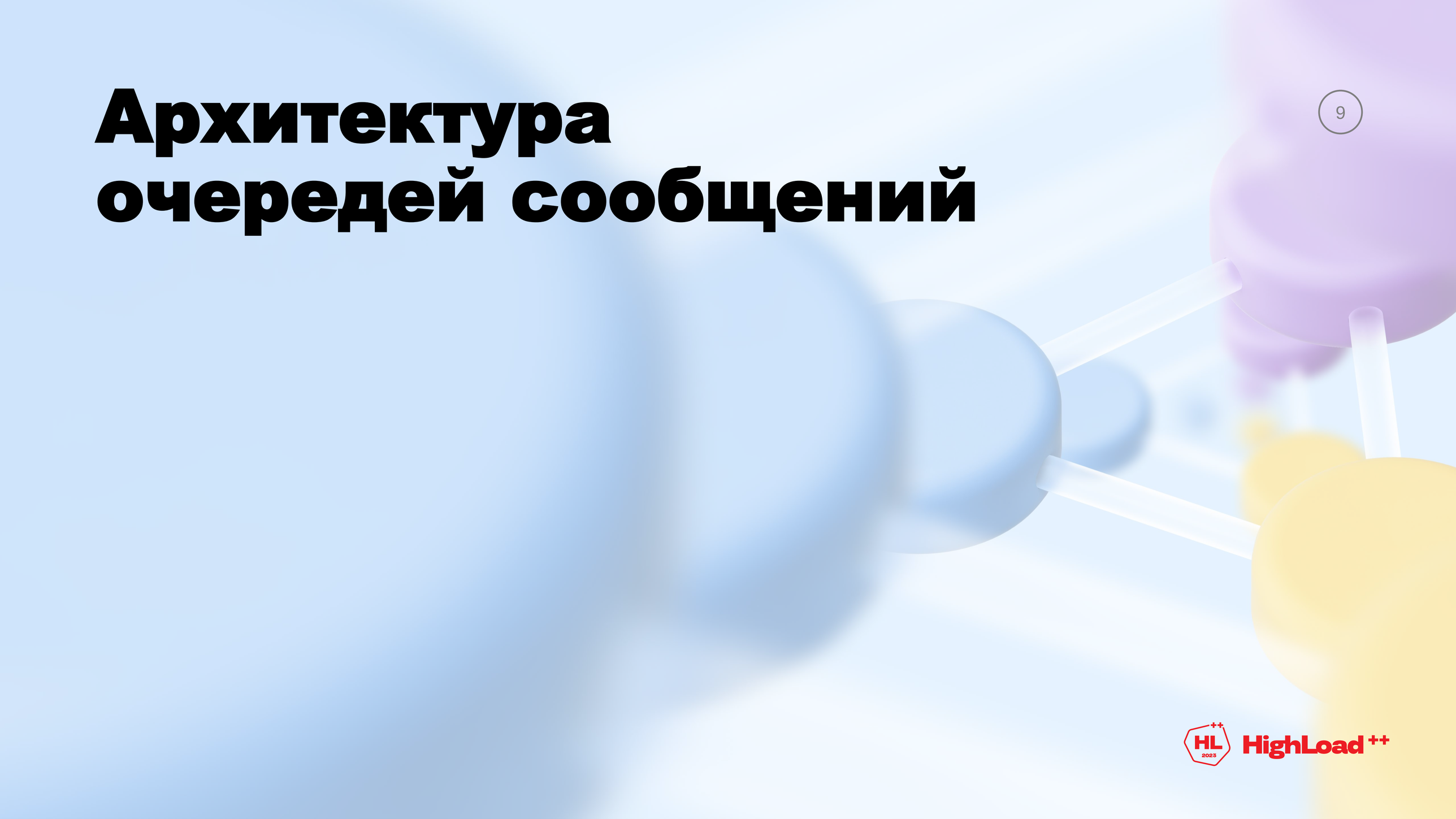
# Пример реальных данных



| Name | All topics | Active topics | Write speed, MB/s | Topics read speed, MB/s | Consumers read speed, MB/s |
|---|---|---|---|---|---|
| [redacted] | 137 | 89 | 8 973,15 | 35 124,49 | 0,00 |
| [redacted] | 29 | 23 | 9 549,97 | 9 777,50 | 2 298,22 |
| [redacted] | 81 | 53 | 2 927,82 | 9 615,71 | 1 160,34 |
| [redacted] | 3059 | 2681 | 2 467,30 | 7 847,45 | 3 264,39 |
| [redacted] | 56 | 33 | 4 037,52 | 7 554,01 | 52,22 |
| [redacted] | 447 | 297 | 3 131,24 | 7 280,30 | 5 521,75 |
| [redacted] | 139 | 90 | 4 633,40 | 5 417,97 | 369,11 |
| [redacted] | 25 | 23 | 1 203,84 | 5 012,26 | 2 397,34 |
| [redacted] | 4 | 4 | 4 511,24 | 4 474,83 | 100,30 |
| [redacted] | [redacted] | [redacted] | 1 093,41 | 4 006,89 | 1 116,68 |

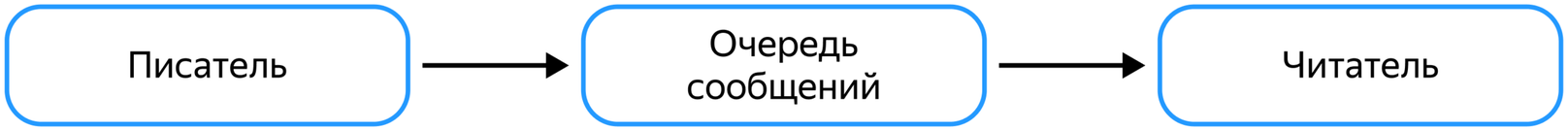
# Архитектура очередей сообщений





# Что такое потоковая очередь сообщений

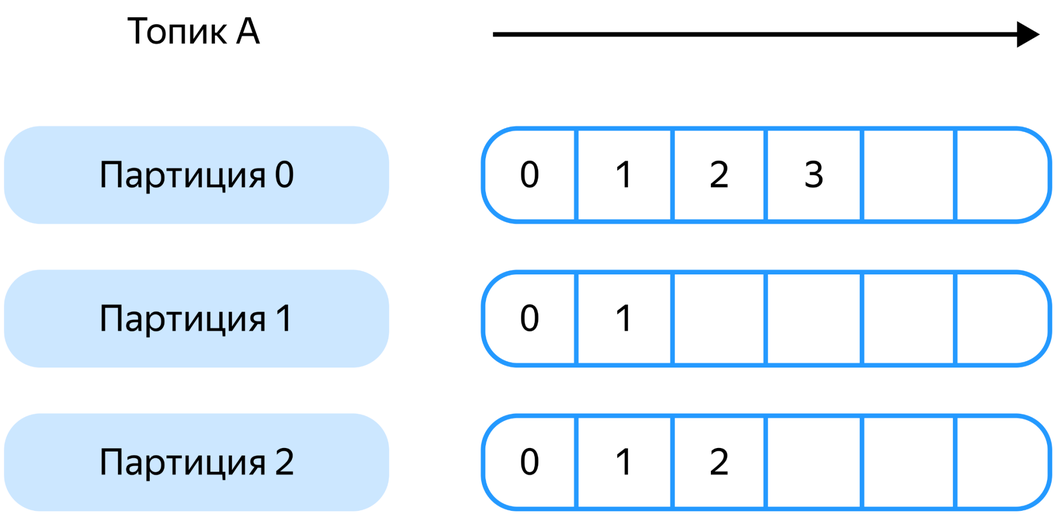




# Сущности потоковой очереди сообщений

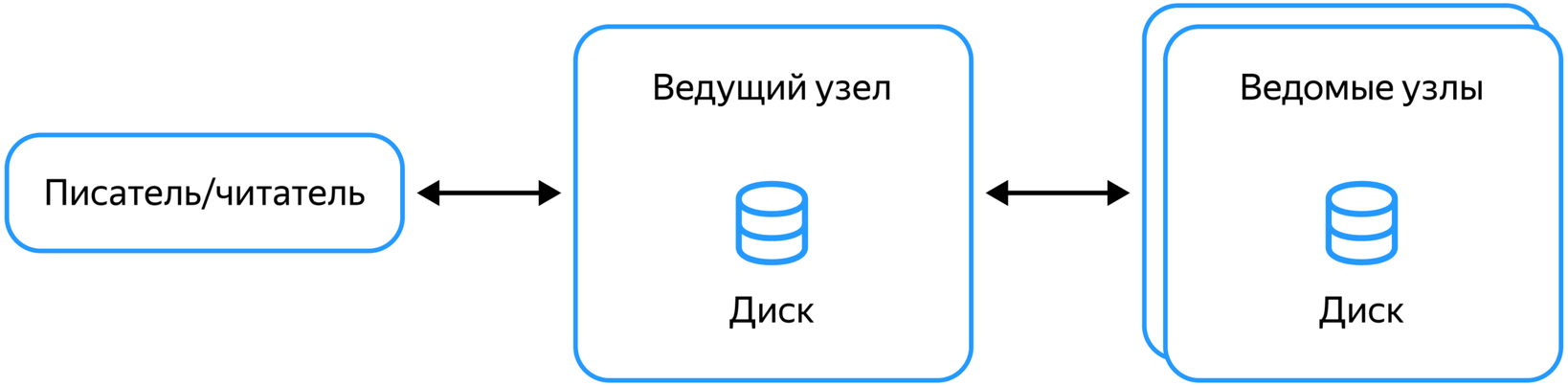


- Пользовательские данные сгруппированы по топикам (topics)
- Топик разделен на партиции (partitions)
- Одна партиция — это распределённый лог сообщений
- Номер сообщения в партиции — смещение (offset)



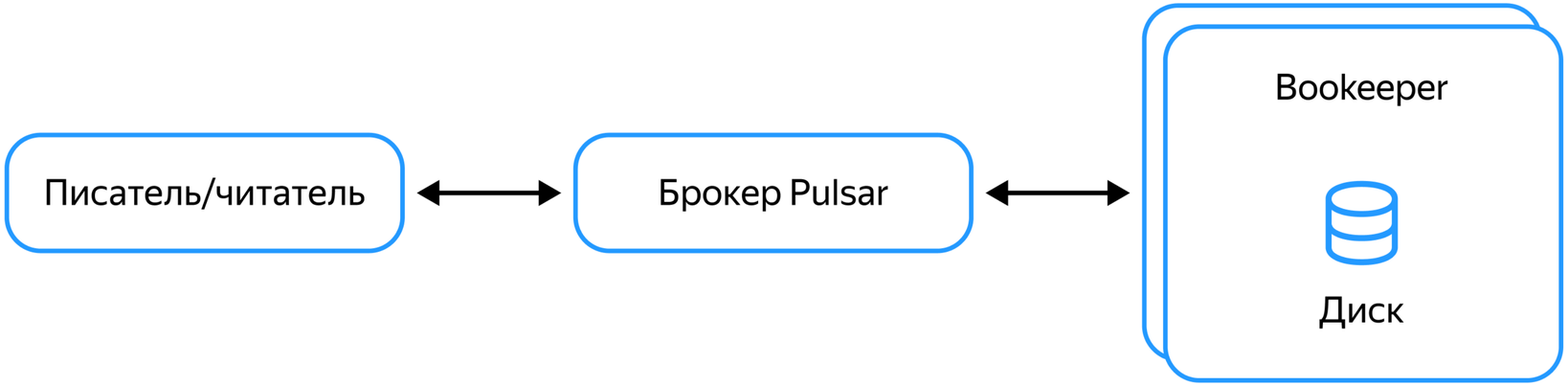
# Kafka





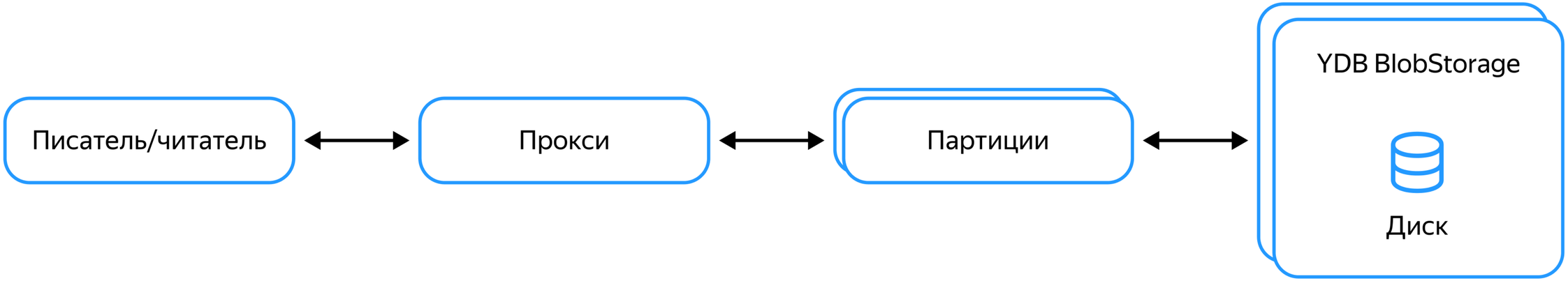
# Pulsar





# YDB Topics





# Mirror-3-dc



**3**

зоны доступности

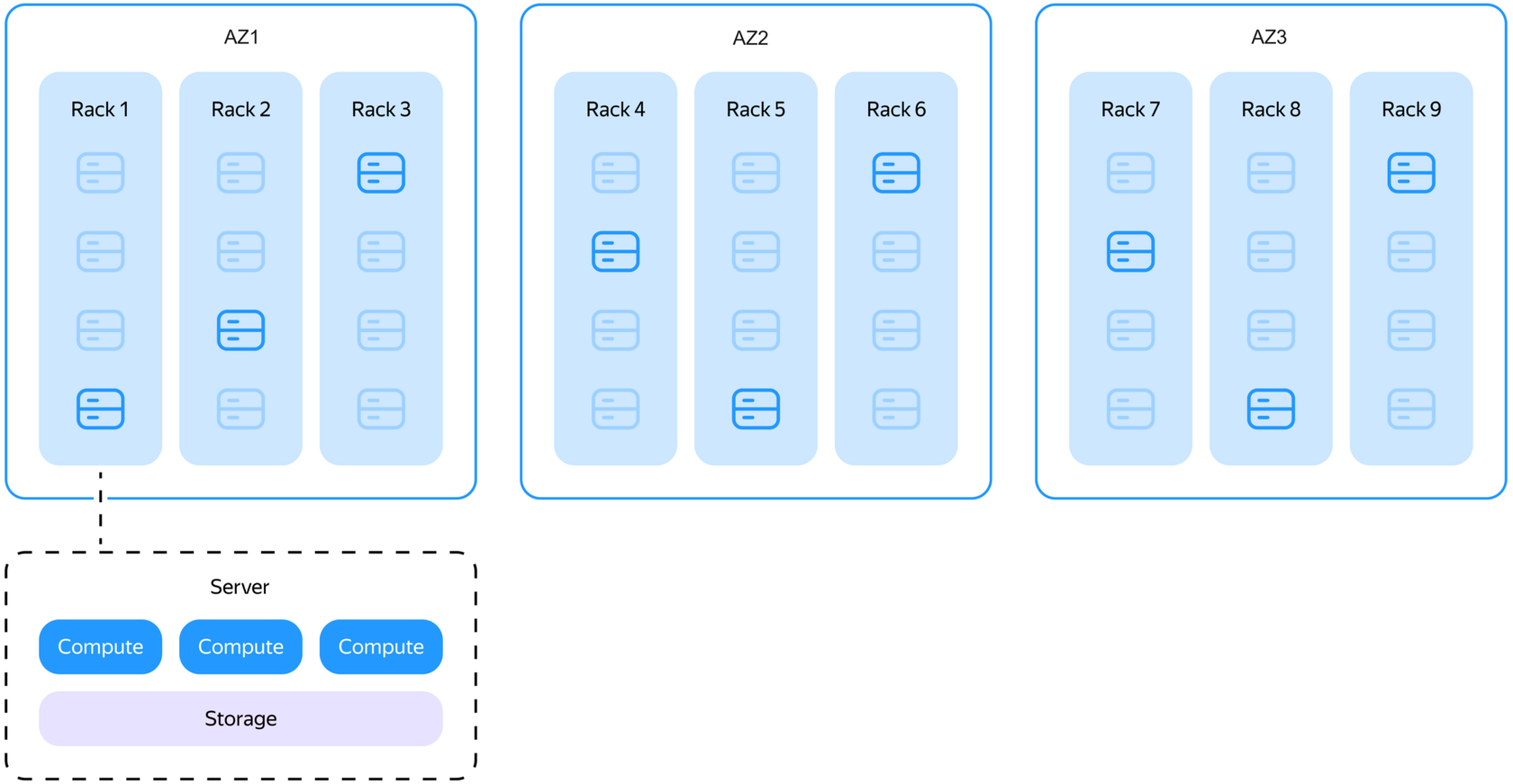
**9+**

узлов хранения

**×3**

множитель объема хранения

Переживает потерю одной зоны доступности + одной серверной стойки в любой другой зоне



# Block-4-2



Erasure-кодирование,
коды Reed-Solomon

1

зона доступности

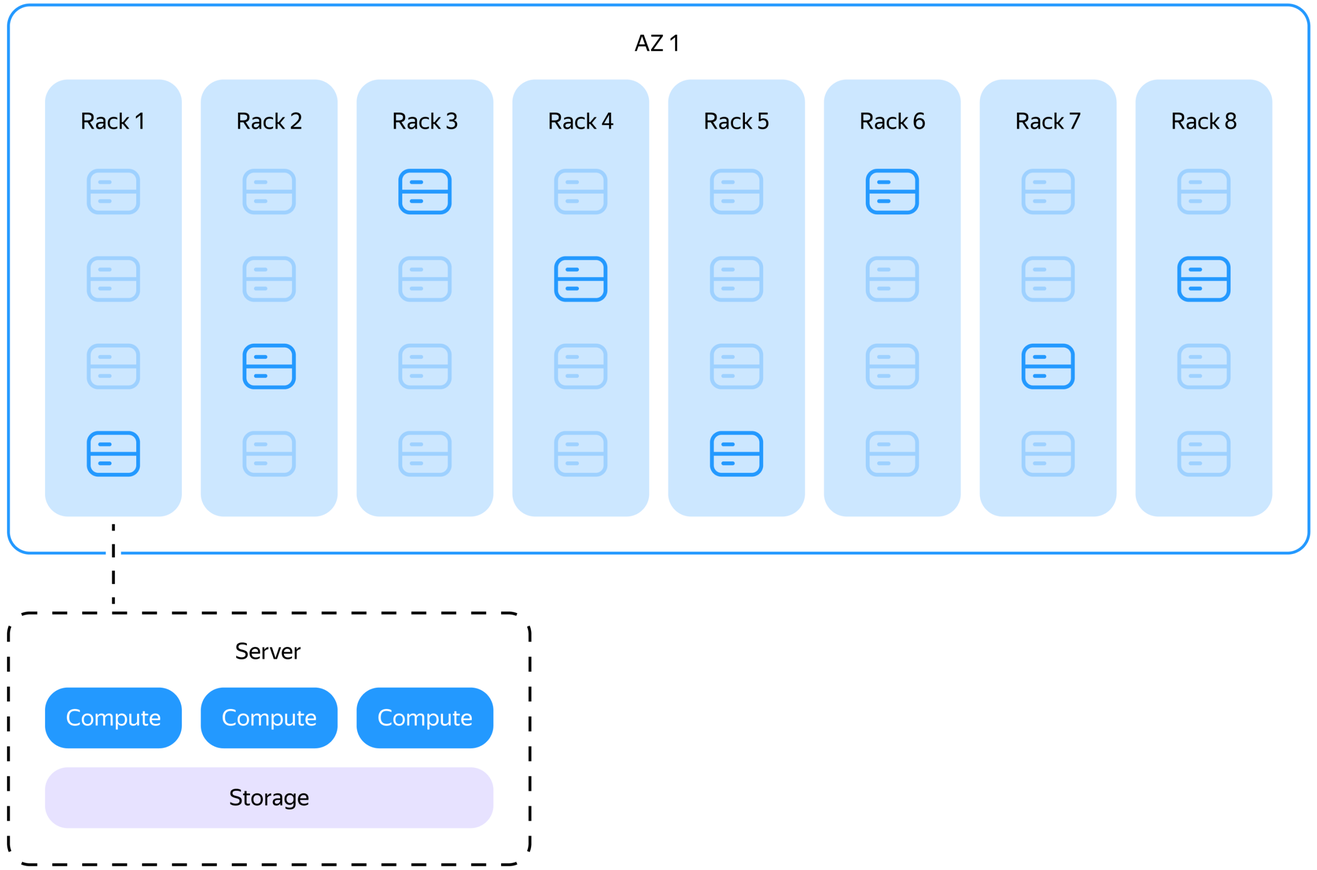
8+

узлов хранения

×1,5

множитель объема хранения

Переживает потерю
2 серверных стоек из 8



# Почему не Kafka



# Почему не Kafka в 2017 году

- Ограничения ZooKeeper
  - Потолок 10 000+ партиций

# Почему не Kafka в 2017 году



- Ограничения ZooKeeper
- Георазпределённый кластер
  - В принципе не было

# Почему не Kafka в 2017 году



- Ограничения ZooKeeper
- Геораспределённый кластер
- Отсутствие exactly-once и транзакций
  - Только at-least-once
  - Нет at-most-once

# Почему не Kafka в 2017 году



- Ограничения ZooKeeper
- Георaспределённый кластер
- Отсутствие exactly-once и транзакций
- Отсутствие квот и гибкого разграничения доступа
  - Один «шумный сосед» может потребить всю пропускную способность или процессорное время

# Почему не Kafka в 2017 году



- Ограничения ZooKeeper
- Георaспределённый кластер
- Отсутствие exactly-once и транзакций
- Отсутствие квот и гибкого разграничения доступа

# Мы сильно выросли



Скорость записи:

2017 — Сейчас

**До 1 ГБайт/с** — **До 80 ГБайт/с**

Но и Kafka тоже развивается

# Почему не Kafka в 2023 году



- Отказ от Zookeeper, но Kraft еще не доработан
  - По-прежнему «Limitations and known issues»
  - Окончательный переход – в 2024

# Почему не Kafka в 2023 году



- Отказ от Zookeeper, но Kraft еще не доработан
- Геораспределённый кластер
  - Упрощенный режим — федерация
  - Запись только в локальный дата-центр
  - Асинхронное зеркалирование
  - Нет подтверждения о записи в другие дата-центры
  - Нет exactly-once

# Почему не Kafka в 2023 году



- Отказ от Zookeeper, но Kraft еще не доработан
- Георасределённый кластер
- Ограниченный exactly-once
  - По умолчанию гарантия at-least-once
  - Пользователи могут реализовать at-most-once
  - Есть enable.idempotence, но после перезапуска писателя счетчики сбрасываются
  - Типичный пример правильной реализации: Kafka Streams

# Почему не Kafka в 2023 году



- Отказ от Zookeper, но Kraft еще не доработан
- Георасределённый кластер
- Ограниченный exactly once
- Ограниченный мониторинг
  - Нет метрик по клиентам, только по брокерам и топикам
  - Не годится для serverless

# Почему не Kafka в 2023 году

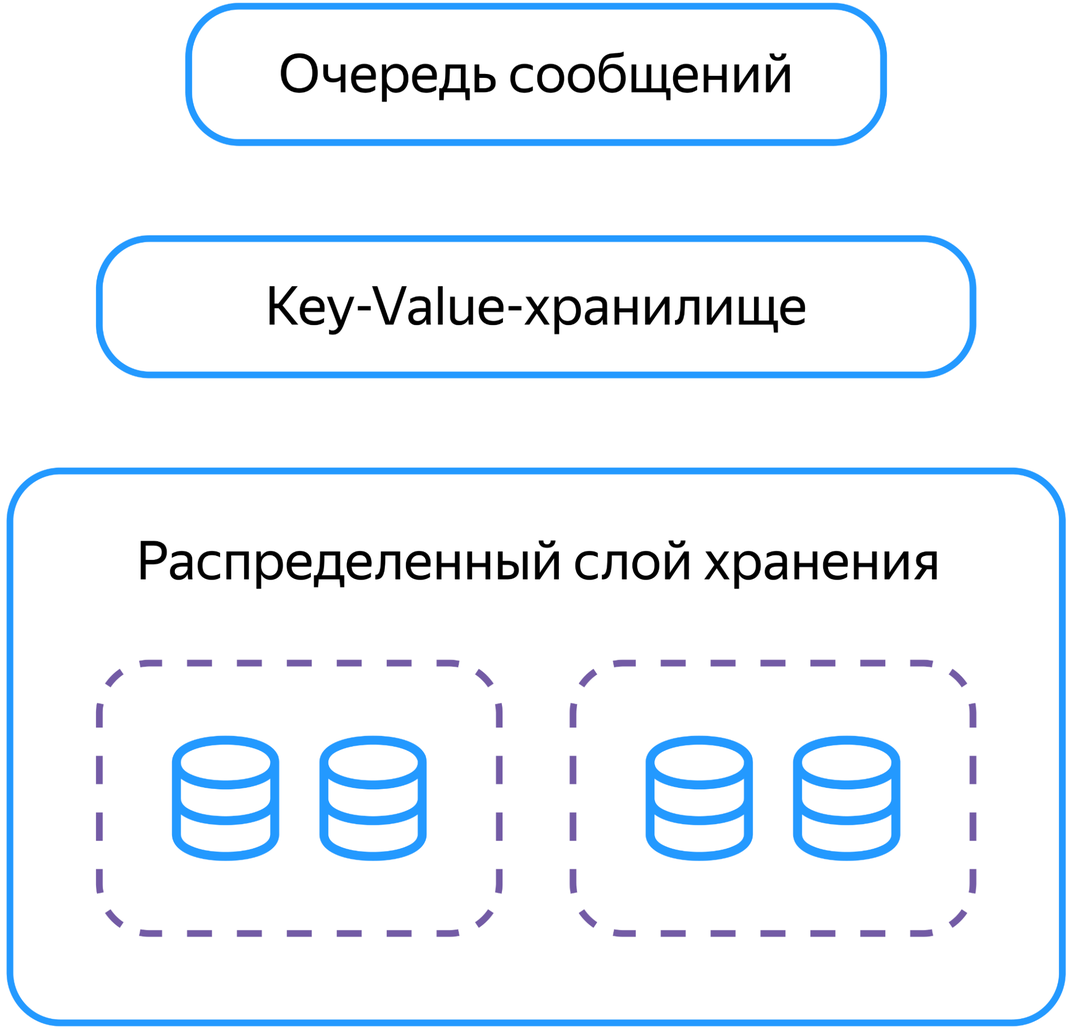


- Отказ от Zookeper, но Kraft еще не доработан
- Георасределённый кластер
- Ограниченный exactly once
- Ограниченный мониторинг

# Почему именно YDB Platform?

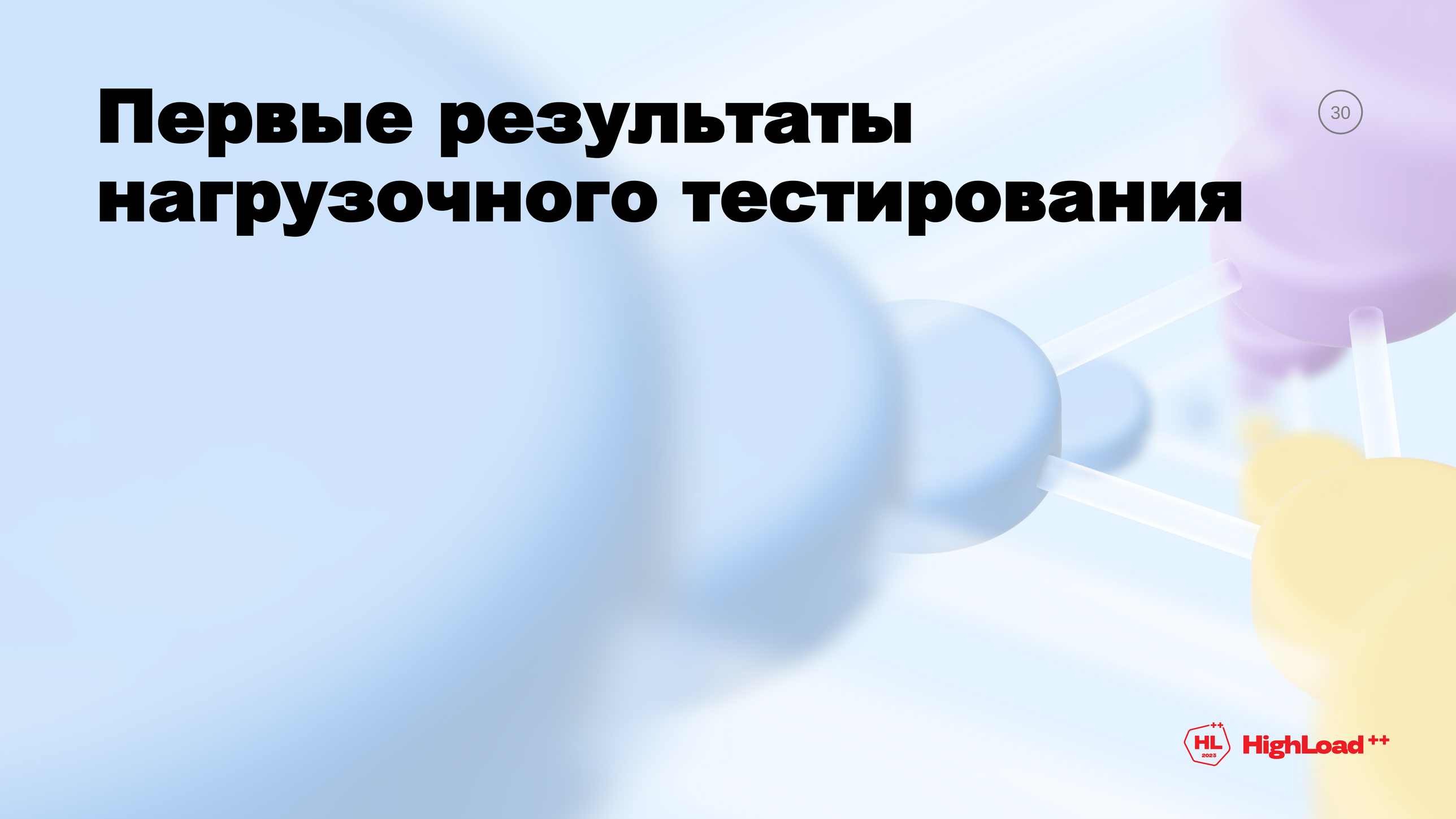
Персистентная очередь поверх Key-Value-хранилища





# Первые результаты нагрузочного тестирования





# Цели тестирования



**Конкуренты**

Понять место среди аналогичных решений

**Лимиты**

Выявление граничных значений производитель-ности на разных тестах

**Стабильность**

Гарантия неизменности производитель-ности между разными версиями (во времени)

**Бенчмарк оборудования**

Сравнение производитель-ности на разном аппаратном обеспечении

# Характеристики стенда



8
физических серверов в одном дата-центре

2
процессора CPU Xeon E5-2660V4

56
логических ядер

256
ГБ ОЗУ

2
NMVE-диска 3,2 ТБ

48
Гбит/с диски

10
Гбит/с сеть

# Замеряемые показатели



## Системные показатели

- Загрузка CPU
- Загрузка памяти
- Скорость дискового ввода/вывода
- Скорость сетевого ввода/вывода

## Прикладные показатели

- Скорость чтения/записи
- Полное время между записью и чтением

# Штатные утилиты имитации



1

kafka-producer-perf-test, kafka-consumer-perf-test

2

pulsar-perf

3

ydb workload topic

# Базовые требования ко всем сценариям



- Писатели и читатели запускаются на 8 серверах одновременно
- Не должен расти лаг чтения
- Фактор репликации равен 3 (в YDB сравнимый режим работы — block-4-2)
- Подтверждение записи ожидается от всех брокеров
- Сжатие сообщений отключено

# Основные сценарии



**Максимальная скорость**

Максимизируется скорость, невзирая на латентность

**Минимальное время (end-to-end, от генерации до вычитывания)**

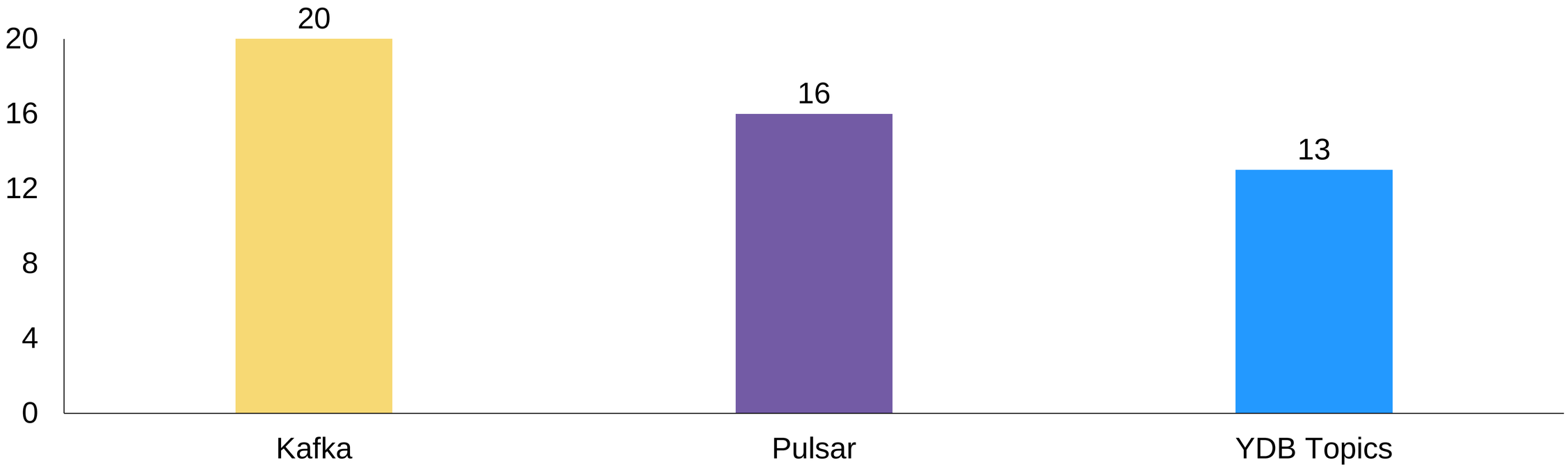
Минимизируется 50 процентиль времени

- Без нагрузки: 100 Кбит/с
- Под нагрузкой: 6,4 Гбит/с (порядка 1/3 от максимальной)

# Первые результаты по скорости

Узкое место: сеть



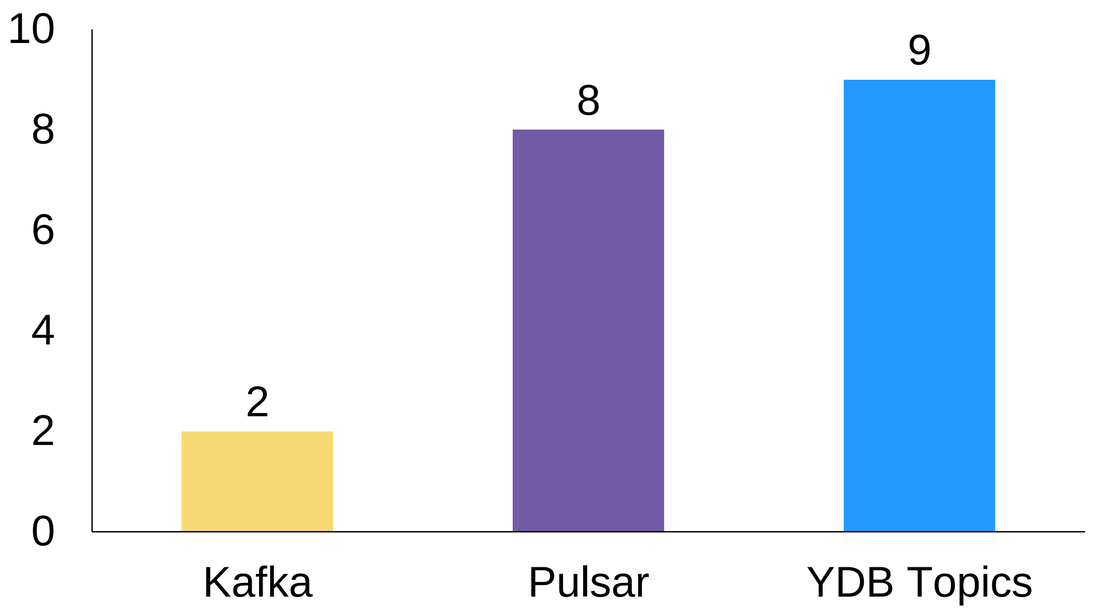


# Первые результаты по времени

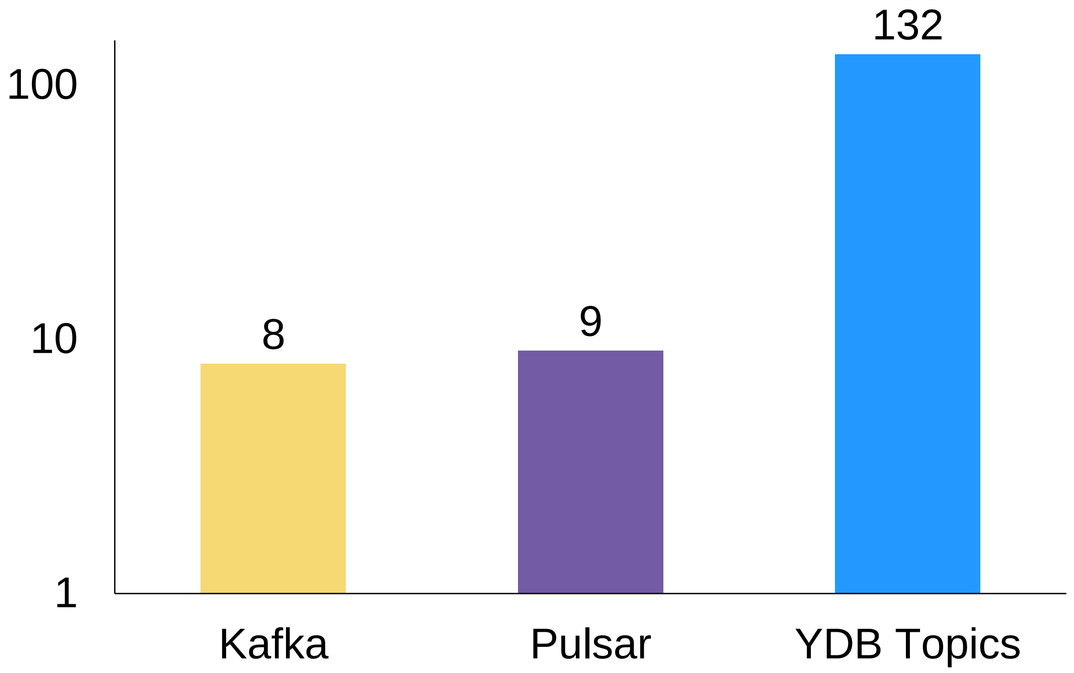
Узкое место: лишние копирования



Время без нагрузки, мс

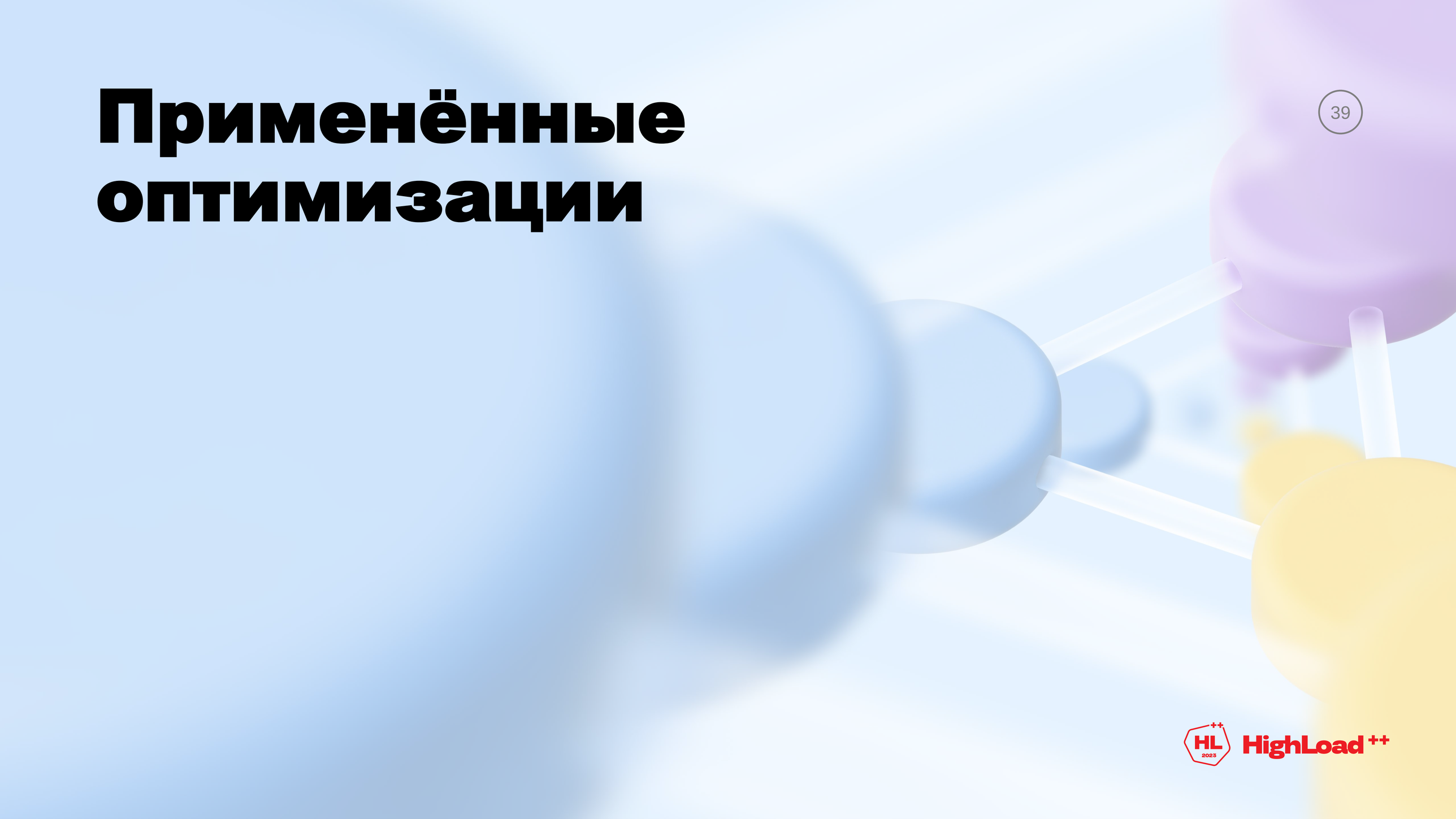



Время под нагрузкой, мс



# Применённые оптимизации





# Правильный генератор нагрузки



- Правильное число разделов
- Не упираться в квоты
- Правильный механизм генерации сообщений без всплесков скорости
- Метрики мониторинга

Результат: полноценное использование ресурсов кластера

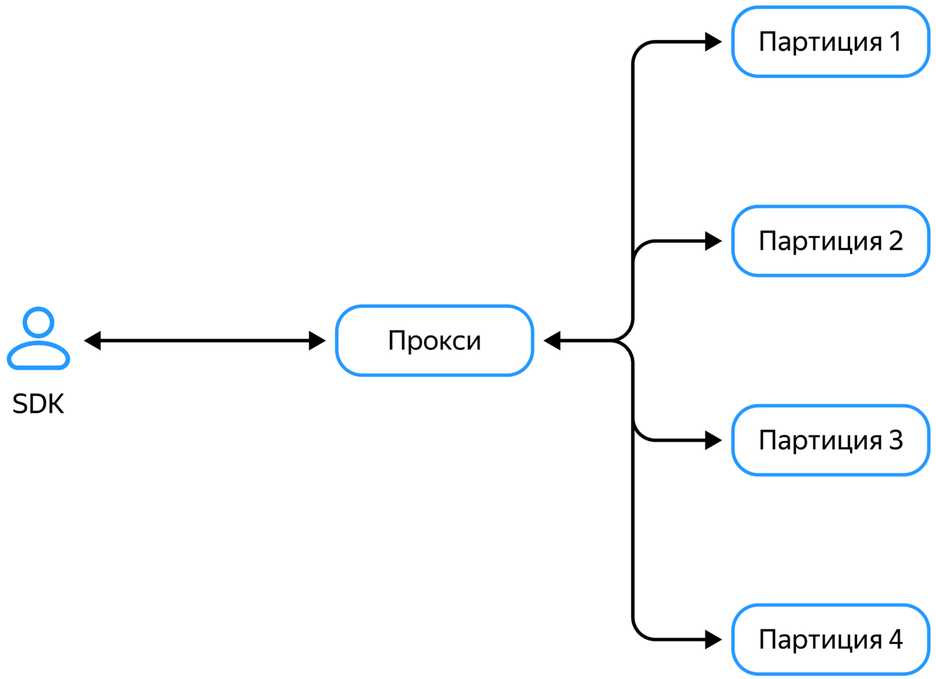
clck.ru/362NSn

# Прямая запись в партицию

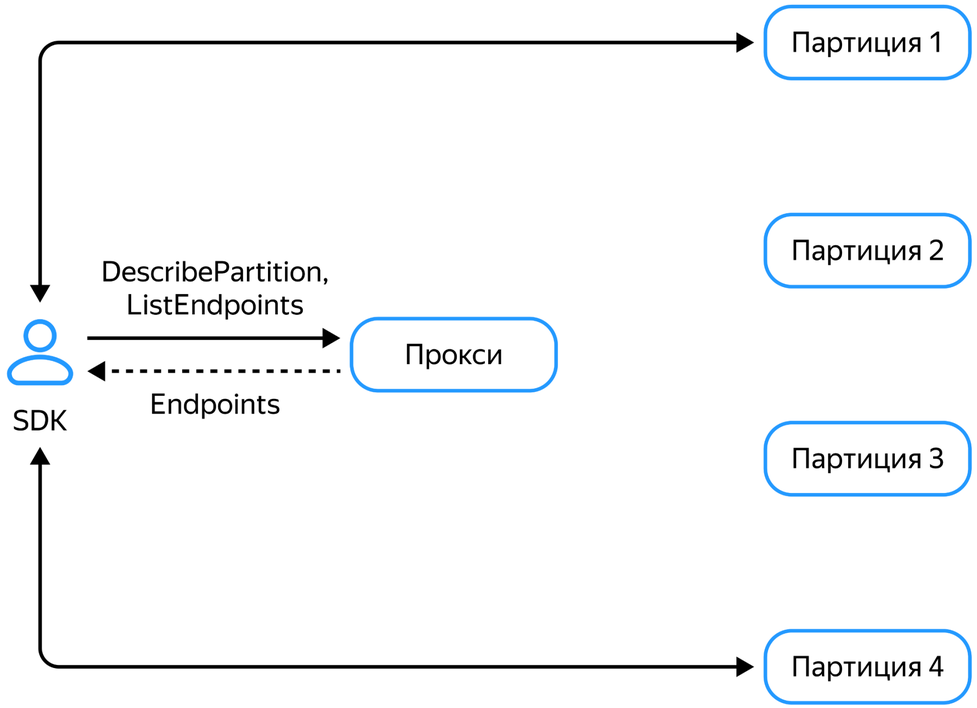
Результат: экономия сети и процессора



## До

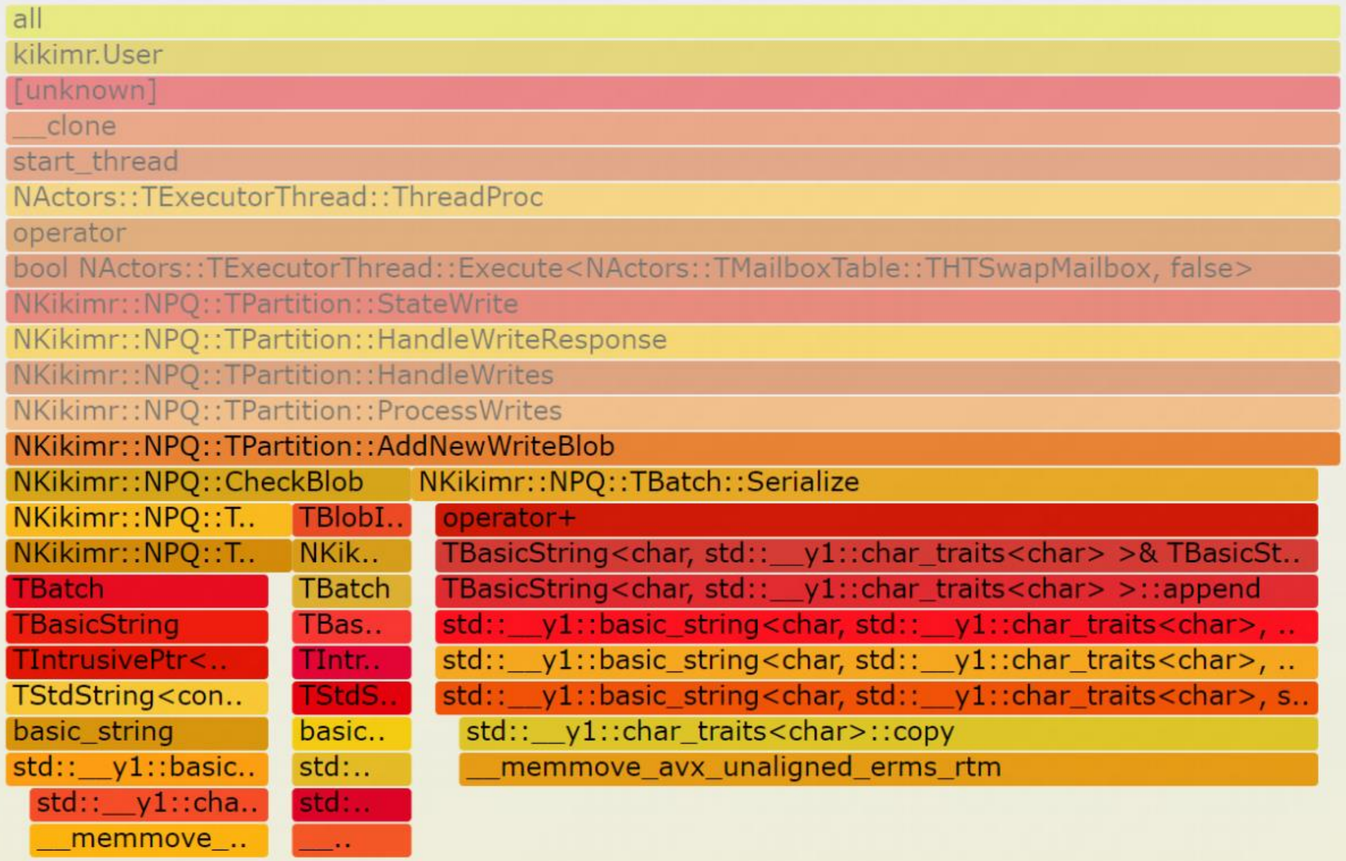



## После



# Профилирование: флеймграф





# Профилирование: узкое место



## До

```
string TBatch::Serialize() {
    uint headerSize = Header.ByteSize();
    string_view headerSizeString(
        (const char*)&headerSize, sizeof(uint));

    string headerString;
    Header.SerializeToString(&headerString);

    return headerSizeString + headerString +
PackedData;
}
```

# Профилирование: узкое место



## До

```
string TBatch::Serialize() {
    uint headerSize = Header.ByteSize();
    string_view headerSizeString(
        (const char*)&headerSize, sizeof(uint));

    string headerString;
    Header.SerializeToString(&headerString);

    return headerSizeString + headerString +
PackedData;
}
```

## После

```
void TBatch::SerializeTo(string& res) {
    uint headerSize = Header.ByteSize();
    res.append(
        (const char*)&headerSize, sizeof(uint));

    Header.AppendToString(&res);

    res += PackedData;
}
```

Результат: столбик на флеймграфе был устранён

# Erasure Encoding (стирающий код)



**Потребляет всего ×1,5 хранимого места**
**при гарантиях доступности, сравнимых с ×3 репликацией**

ydb.tech/ru/docs/cluster/topology

wikipedia.org/wiki/Стирающий_код

# Erasure Encoding: ускорение



## До

```
for (ui64 blockIdx = firstBlock; blockIdx != lastBlock; ++blockIdx) {
    for (ui32 t = mint; t < DataParts; ++t) {
        adj ^= IN_EL(m - 1 - t, t);
    }
    for (ui32 l = 0; l < LineCount; ++l) {
        ui64 sourceData = IN_EL(l, 0);
        OUT_M1(l) = adj ^ sourceData;
        OUT_M(l) = sourceData;
        if (!isFromDataParts)
            OUT_EL(l, 0) = sourceData;
    }
    for (ui32 t = 1; t < DataParts; ++t) {
        for (ui32 l = 0; l < LineCount; ++l) {
            ui64 sourceData = IN_EL(l, t);
            OUT_M(l) ^= sourceData;
            if (!isFromDataParts)
                OUT_EL(l, t) = sourceData;
        }
```

# Erasure Encoding: ускорение



## До

```
for (ui64 blockIdx = firstBlock; blockIdx != lastBlock; ++blockIdx) {
    for (ui32 t = mint; t < DataParts; ++t) {
        adj ^= IN_EL(m - 1 - t, t);
    }
    for (ui32 l = 0; l < LineCount; ++l) {
        ui64 sourceData = IN_EL(l, 0);
        OUT_M1(l) = adj ^ sourceData;
        OUT_M(l) = sourceData;
        if (!isFromDataParts)
            OUT_EL(l, 0) = sourceData;
    }
    for (ui32 t = 1; t < DataParts; ++t) {
        for (ui32 l = 0; l < LineCount; ++l) {
            ui64 sourceData = IN_EL(l, t);
            OUT_M(l) ^= sourceData;
            if (!isFromDataParts)
                OUT_EL(l, t) = sourceData;
        }
```

## После

```
while (iter0.Valid() && size >= blockSize) {
    while (numBlocks--) {
        ui64 d0 = 0;
        ui64 d1 = 0;
        ui64 d2 = 0;
        ui64 d3 = 0;
        ui64 s = 0;

        ptr[0] = a0[0] ^ a1[0] ^ a2[0] ^ a3[0];
        d0 ^= a0[0];
        d1 ^= a1[0];
        d2 ^= a2[0];
        d3 ^= a3[0];
```

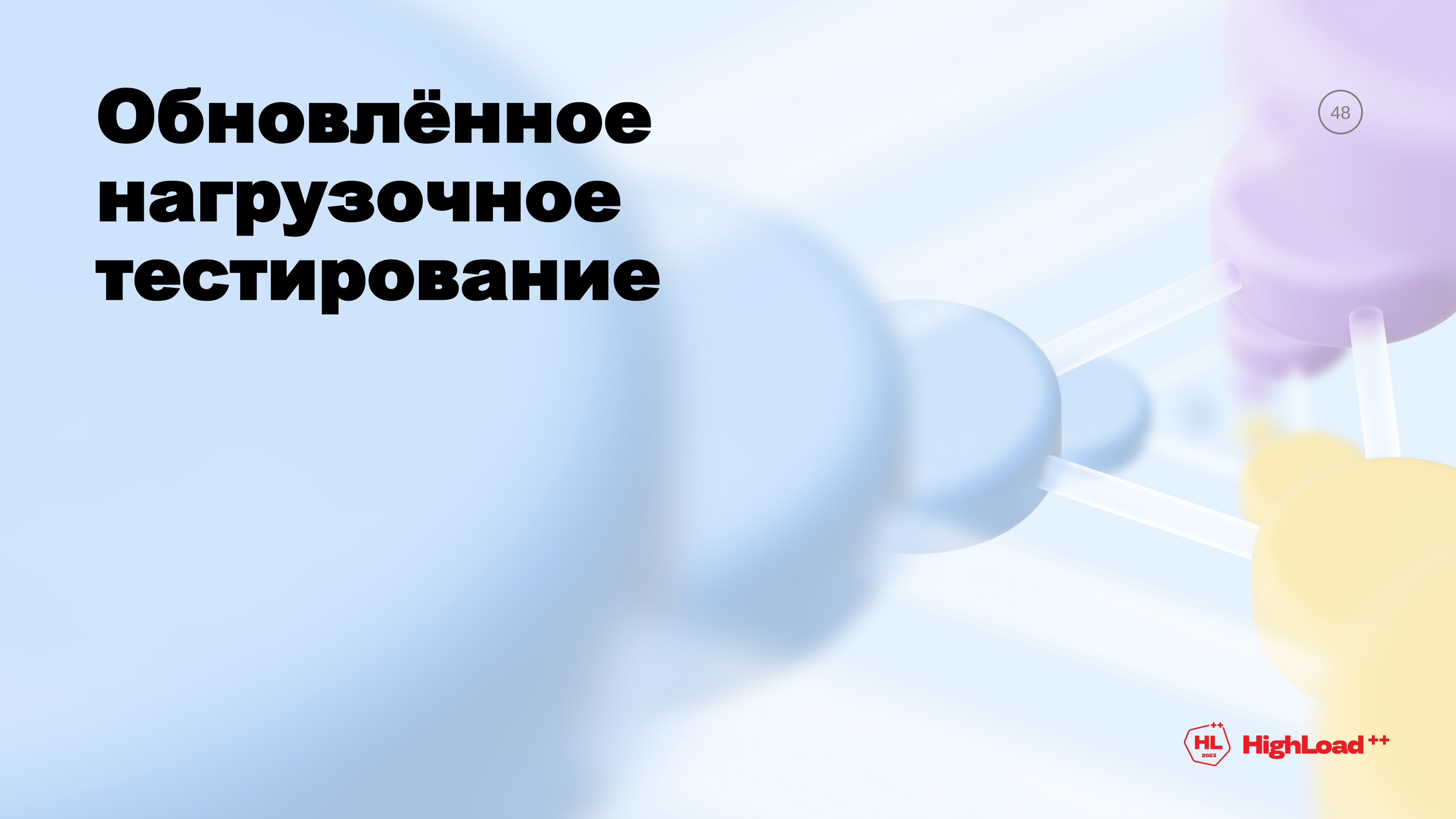
Результаты:

**×3**
ускорено кодирование
с 0,7 Гбайт/с до 2 Гбайт/с

**×1,5**
ускорено декодирование
с 1,7 Гбайт/с до 3 Гбайт/с

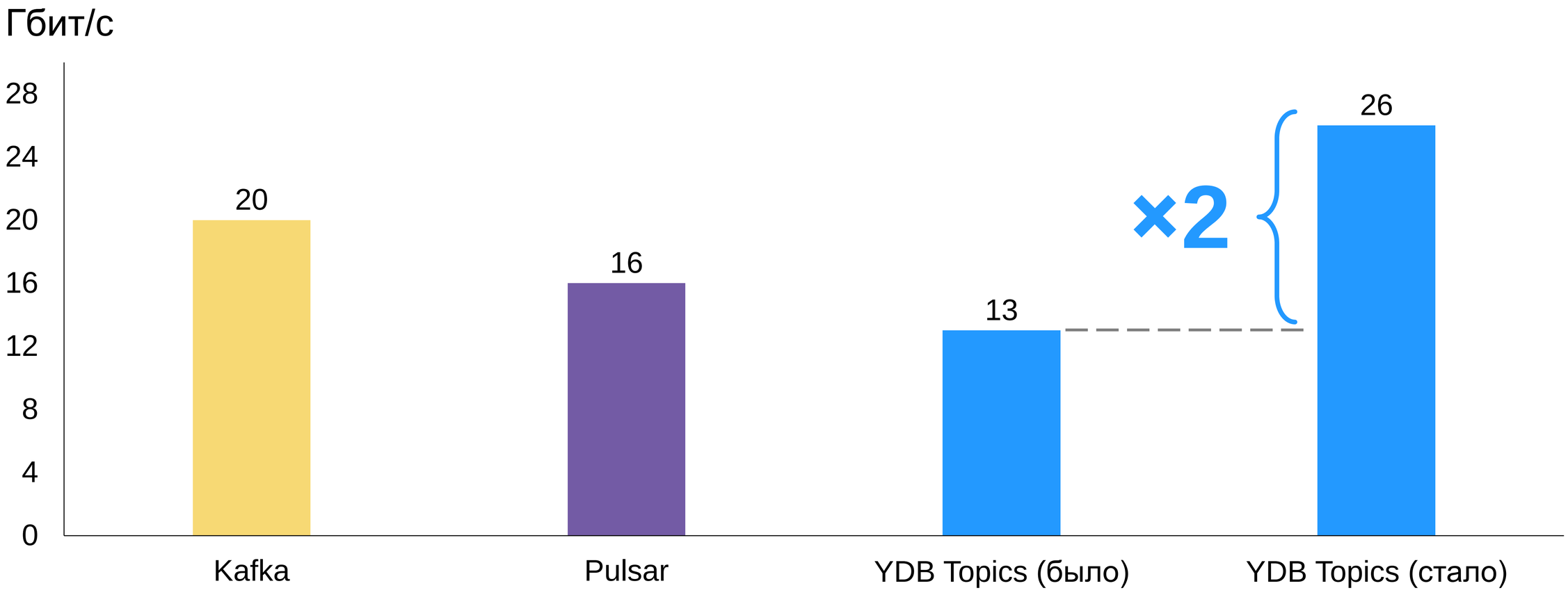
# Обновлённое нагрузочное тестирование





# Максимальная скорость

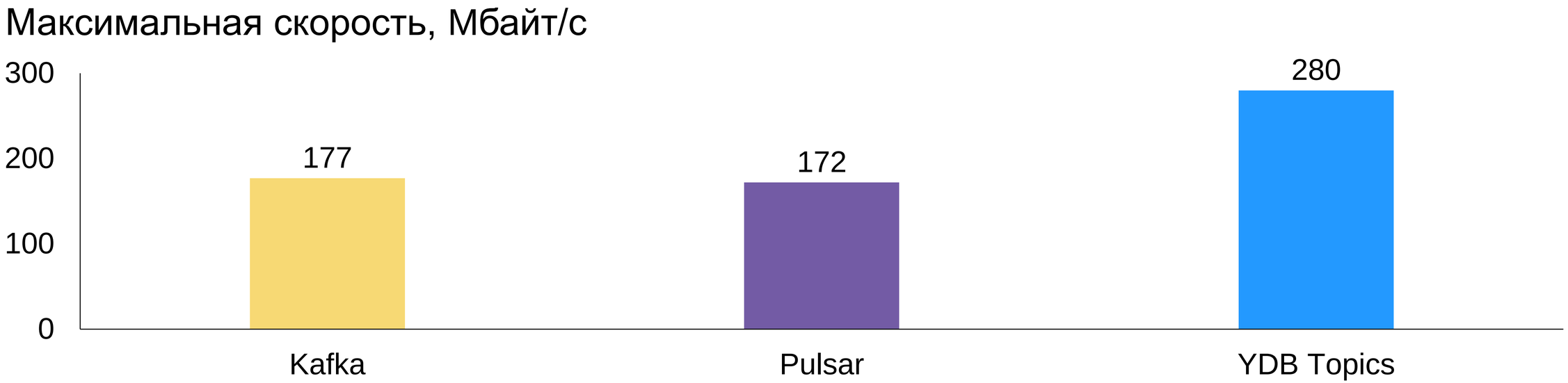




# Эффективность использования диска при длительной эксплуатации



- Устанавливается retention = 1 сутки. Объем хранимых данных ограничен дисками
- Находится максимальная скорость, чтобы диски были заполнены и не было ошибок



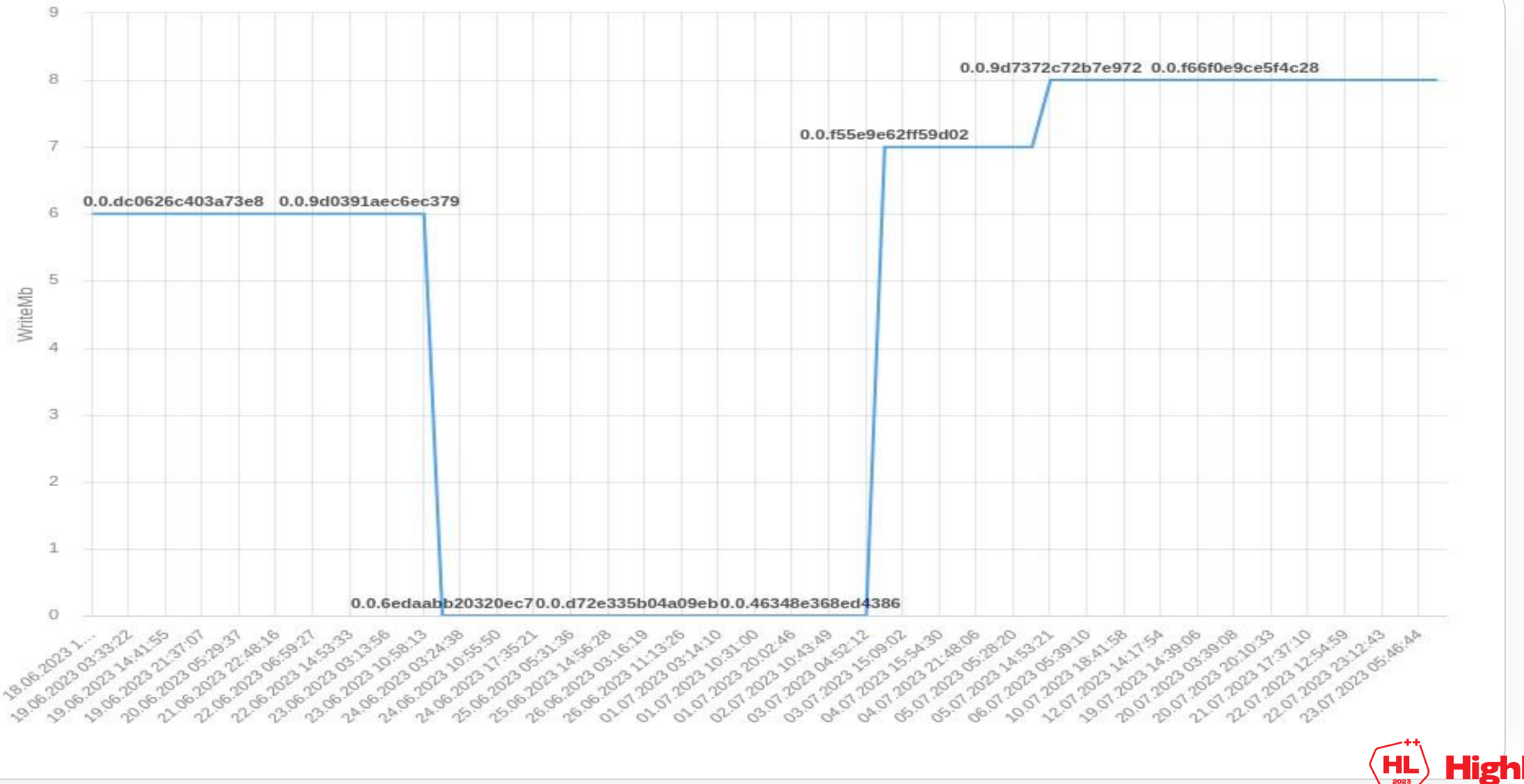
# Регрессионное нагрузочное тестирование



- Гарантии неухудшения производительности во времени между разными версиями
- Регулярно (каждые 4 часа) или по выходу релизного тега
- Запуск в Kubernetes
- Графики в DataLens
- Предупреждения в Telegram

# Пример графика скорости





# Пример сообщения в Telegram



YDBPerformanceReporter
Отчет производительности, за период
С: 2023-06-16
По: 2023-06-23

Тест: topic
Сообщения и потоки: 800k-1
lbkt за 24 прогона(-ов): ✅ 1 ❌ 0
ydb-oss за 14 прогона(-ов): ✅ 1 ❌ 0
Сообщения и потоки: 8M-10
lbkt за 24 прогона(-ов): ✅ 1 ❌ 0
ydb-oss за 14 прогона(-ов): ✅ 1 ❌ 0

# Что еще будем улучшать в производительности



1

Полное время

2

Прямое чтение для повышения скорости

3

Укрупнение передаваемых сообщений в пачки

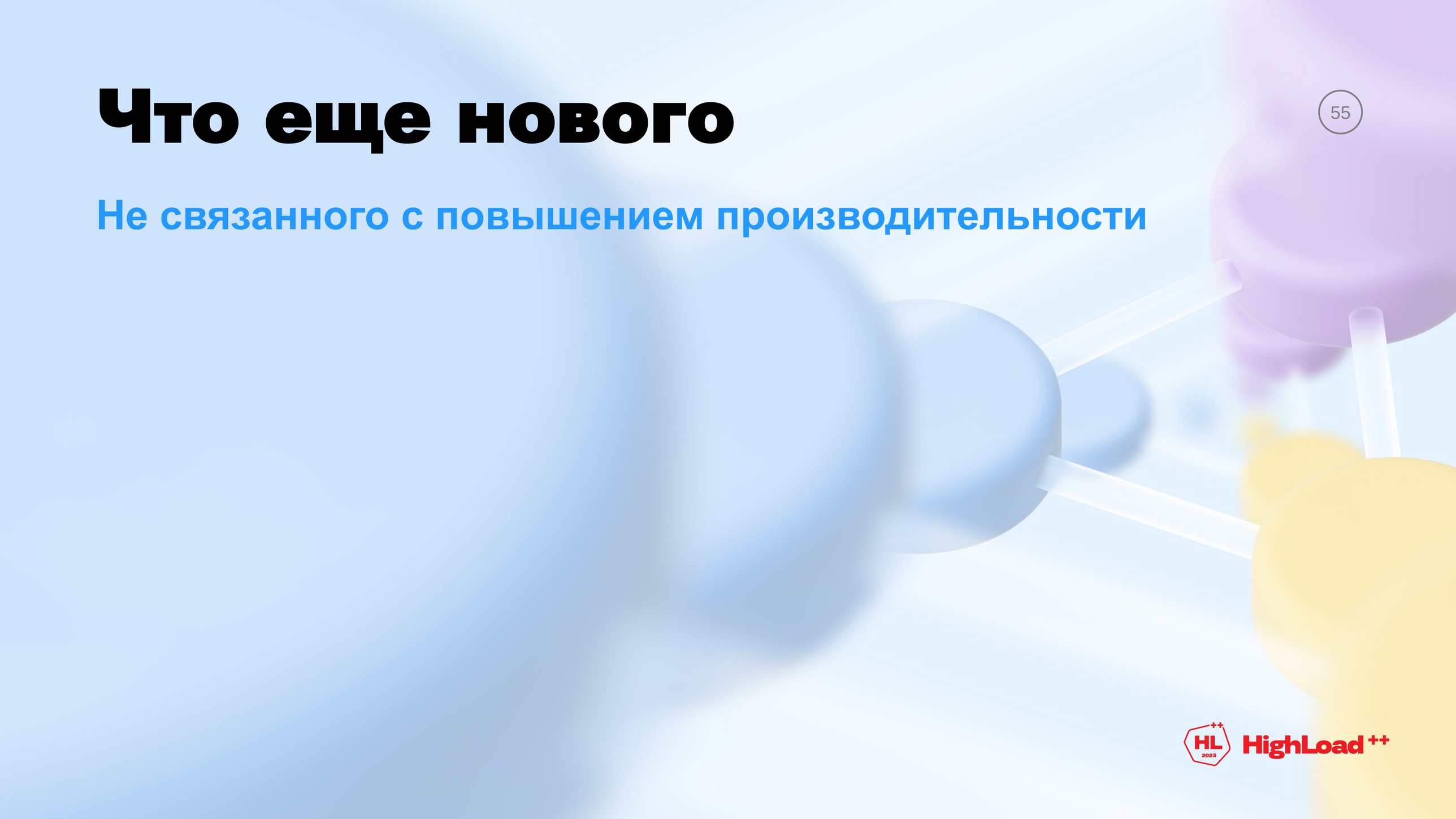
4

Не упаковывать в protobuf данные, а только метаданные

# Что еще нового



## Не связанного с повышением производительности



# Kafka API

https://cloud.yandex.ru/docs/data-streams/kafkaapi/

- Мы добавили Kafka API*
- Сохранили преимущества платформы YDB и повышенную производительность
- Получили Kafka в облаке в режиме serverless/dedicated
- Доступно в опенсорс



* Пока в ограниченном режиме

# Транзакции между топиками и таблицами

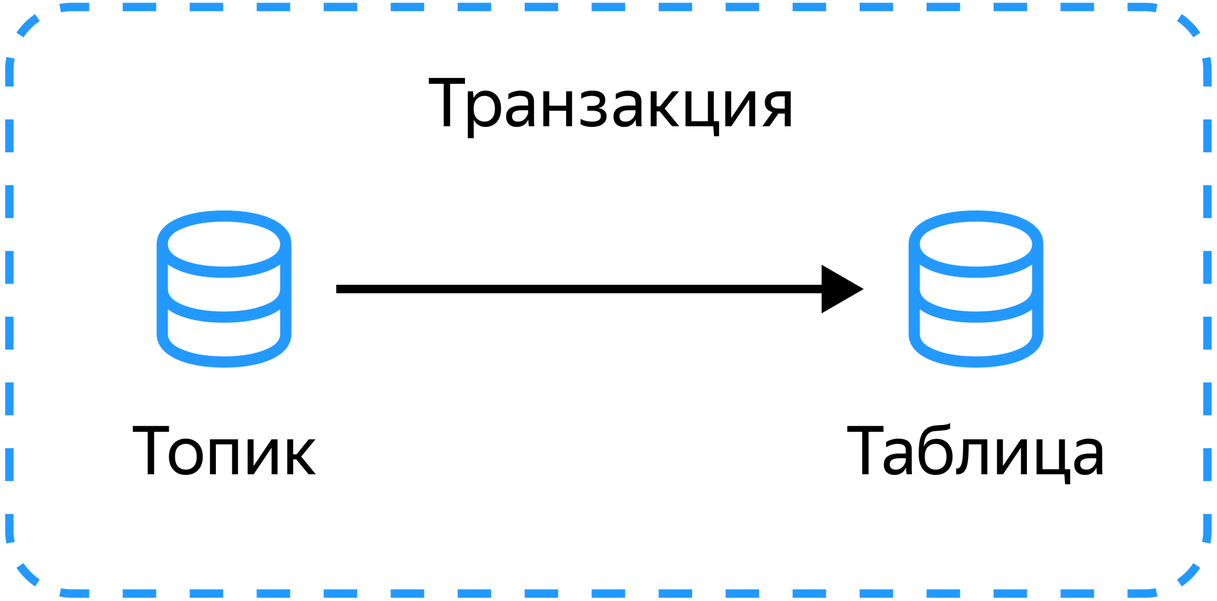


- Популярный сценарий — data ingestion, когда данные из топиков перекладываются в таблицы
- Транзакция

  Прочитать из топика

  Записать в таблицу

  Зафиксировать смещение в топике



# Итоги

- YDB Topics — 6 лет в проде Яндекса
  как основная очередь сообщений
- Доступны в Yandex Cloud
- Вышли в опенсорс
- Подтянули производительность
  до уровня аналогов и даже местами обогнали

**Зевайкин Александр, Яндекс**

HL++ 2023 HighLoad++