

YDB Topics: история взаимоотношений с Kafka

Зевайкин Александр

Руководитель группы разработки

infra.conf

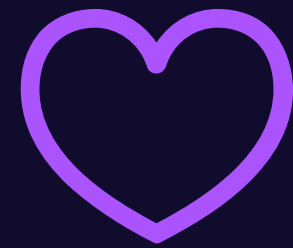
Содержание

- 1** Введение
- 2** Архитектура очередей сообщений
- 3** Почему не Kafka
- 4** Первые результаты нагрузочного тестирования
- 5** Применённые оптимизации
- 6** Обновлённое нагрузочное тестирование
- 7** Kafka API
- 8** Доступны в облаке

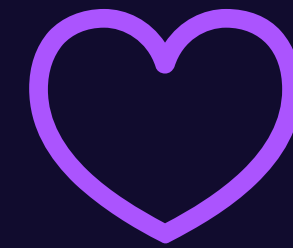


Введение

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

1. Распределённый консенсус
2. Распределённый слой хранения данных
3. Горизонтальное масштабирование, вплоть до тысяч узлов
4. Катастрофоустойчивость и автоматическое восстановление после сбоев
5. Работа в одnodатацентровом и геораспределённом режимах, в том числе в Yandex Cloud в режиме Serverless

Компоненты платформы YDB

1. Распределённый слой хранения
2. Движок транзакций
3. Единый язык запросов YQL
4. OLAP-таблицы
5. OLTP-таблицы
6. Key-Value-хранилище
7. Федеративные запросы
8. 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

1000

команд

30к

топиков

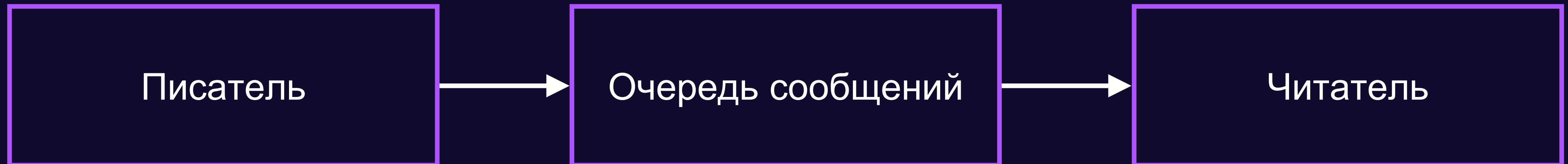
1500

серверов



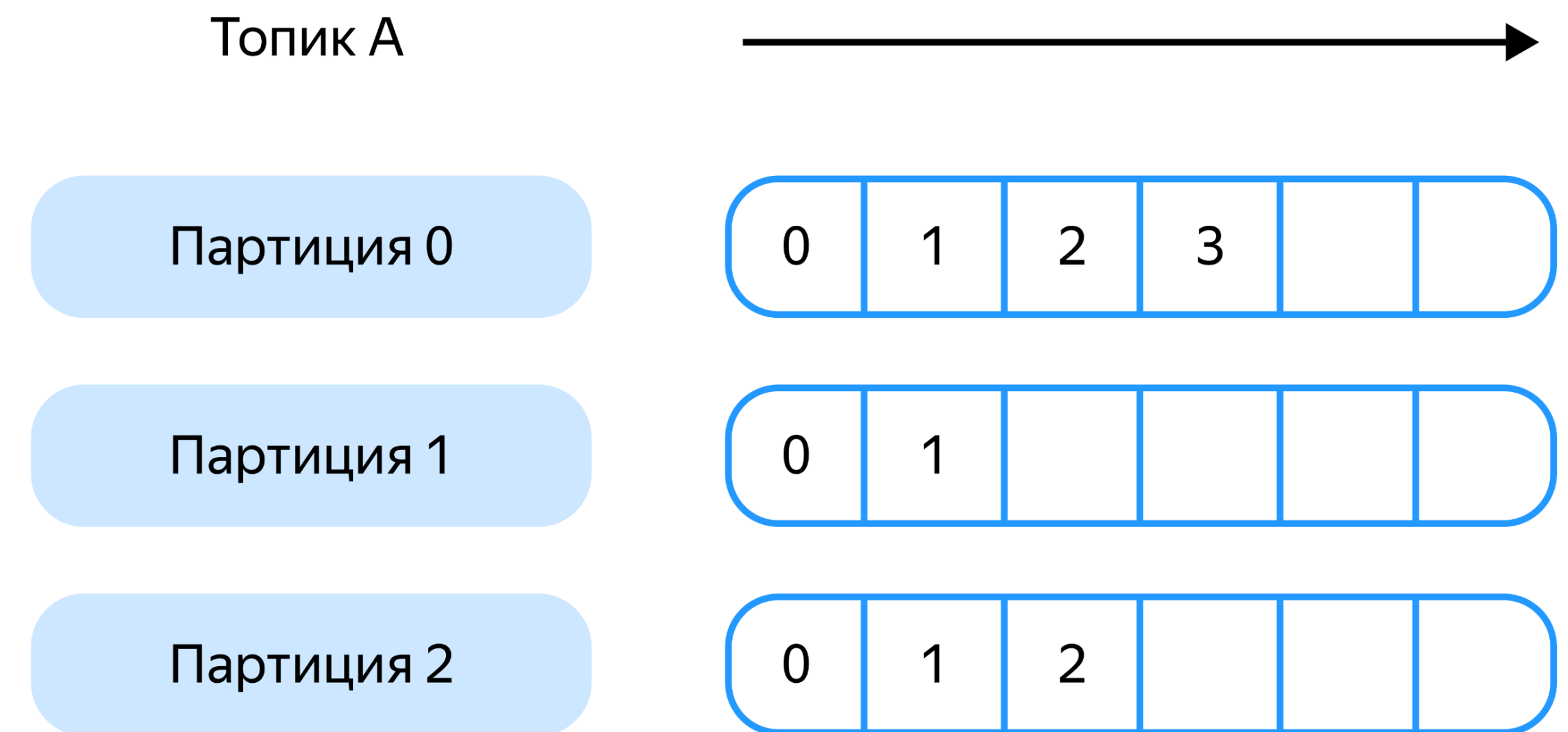
Архитектура очередей сообщений

Что такое потоковая очередь сообщений

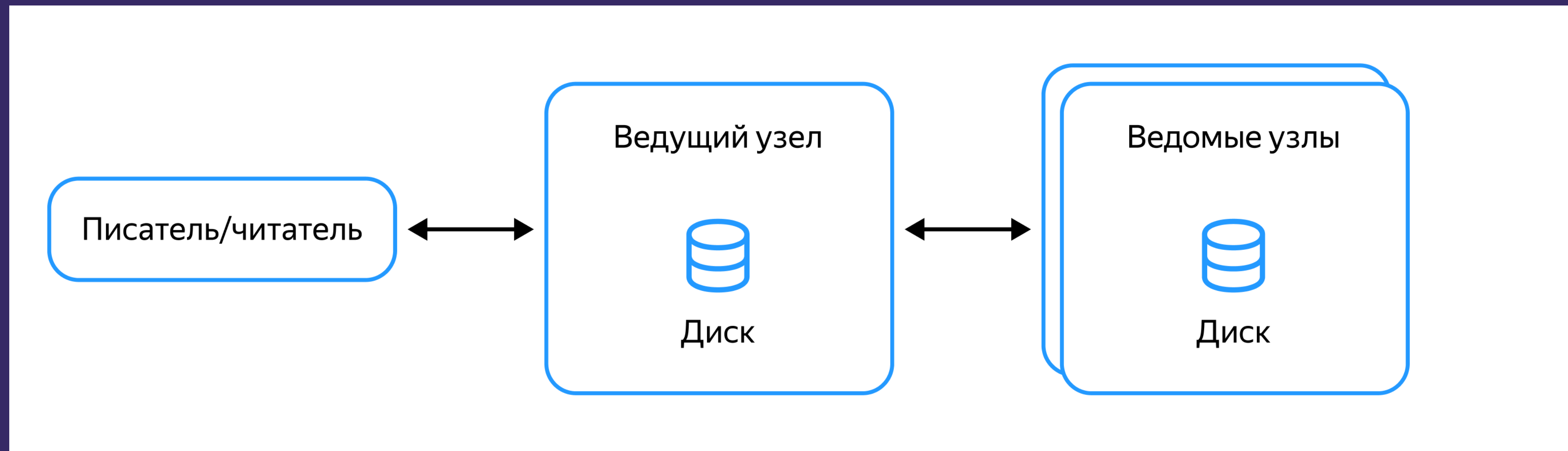


Сущности потоковой очереди сообщений

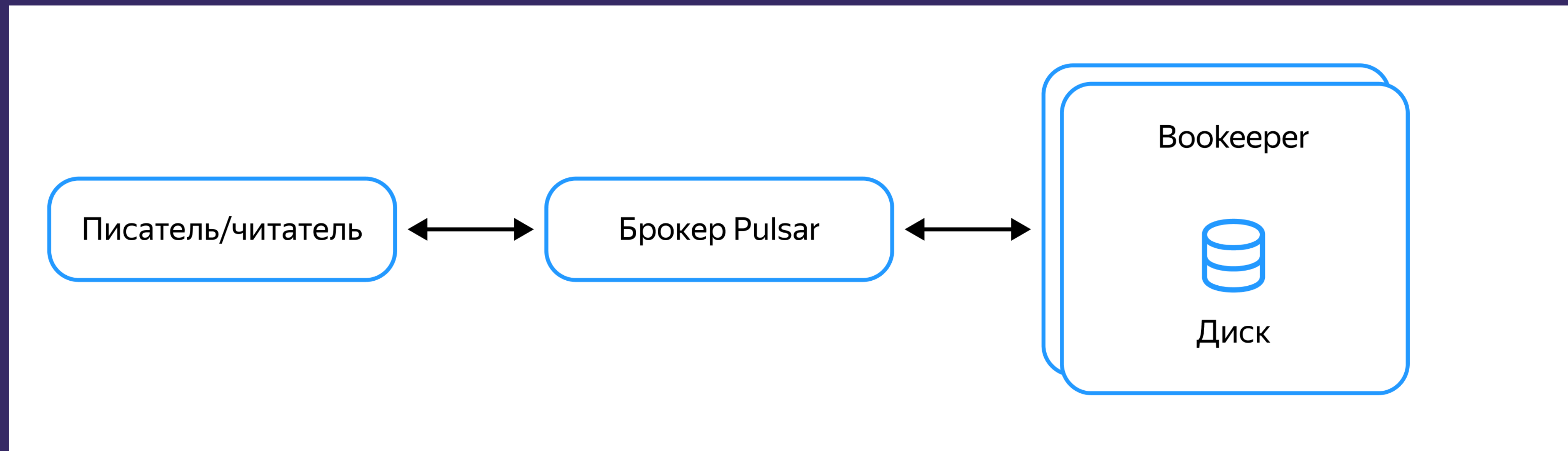
- Пользовательские данные сгруппированы по **топикам** (topics)
- Топик разделён на **партиции** (partitions)
- Одна партиция — это распределённый лог **сообщений**
- Номер сообщения в партиции — **смещение** (offset)



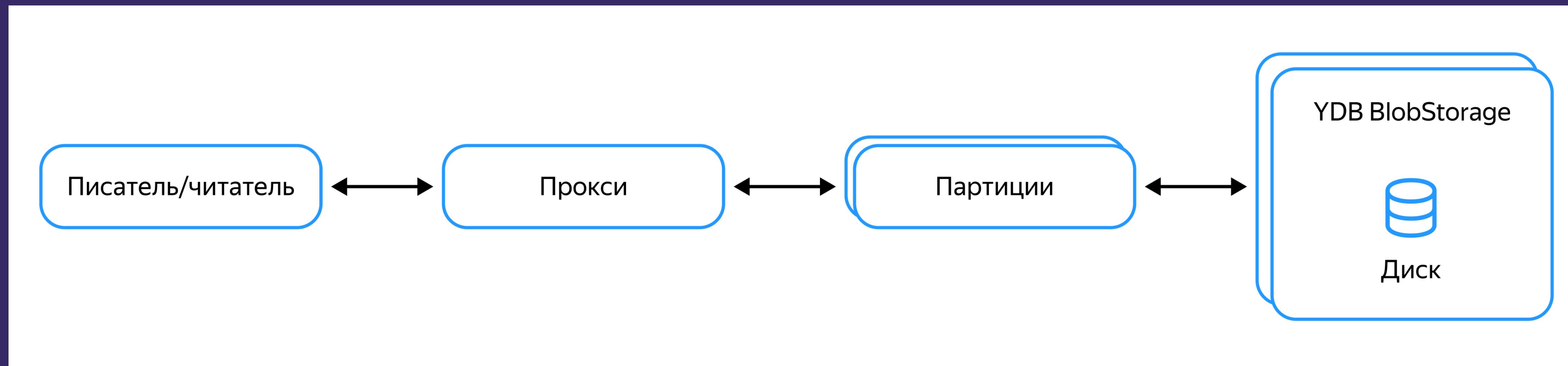
Kafka



Pulsar



YDB Topics



Mirror-3-dc

3

зоны доступности

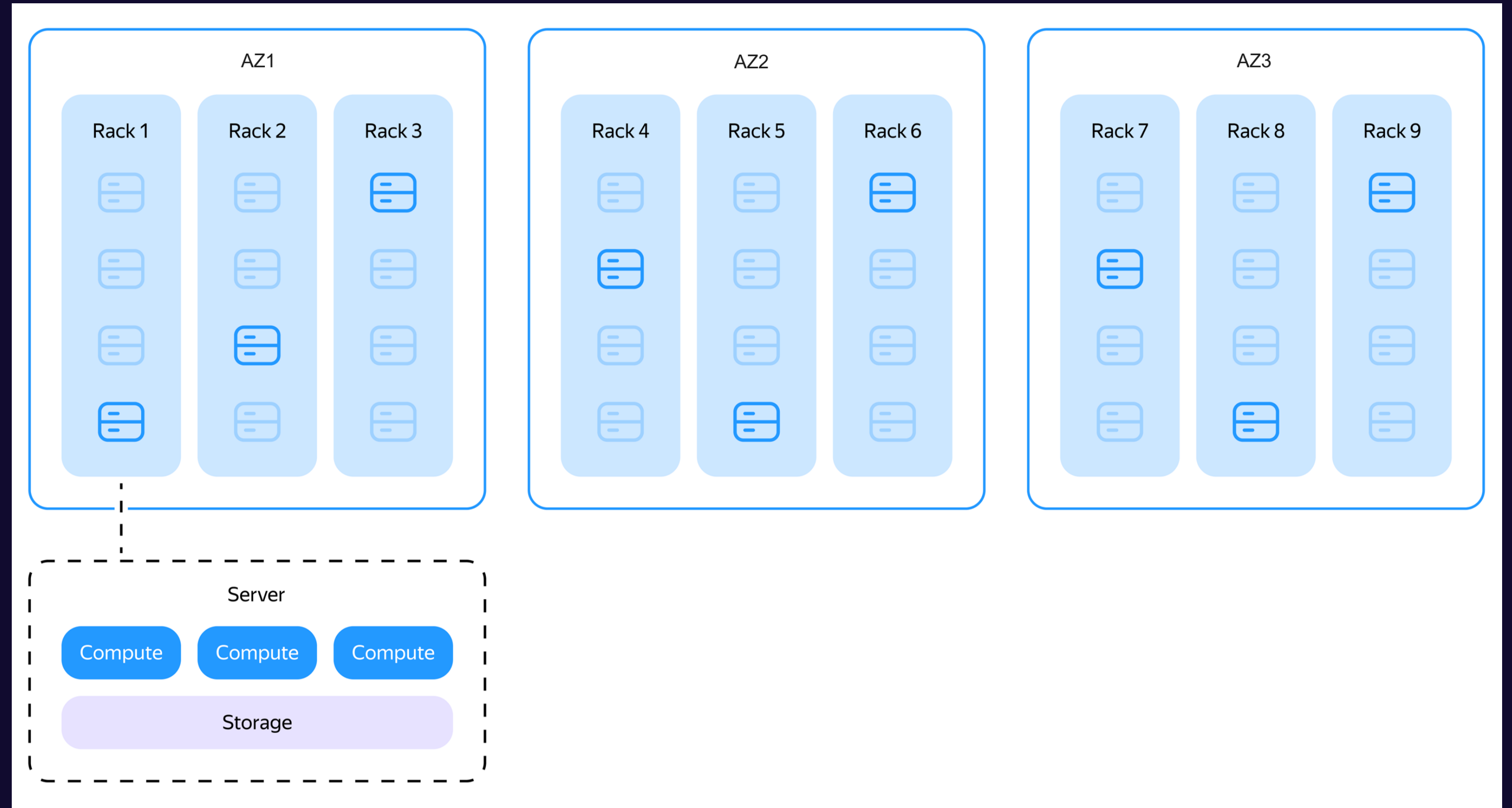
9+

узлов хранения

×3

множитель объёма хранения

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



Block-4-2

Erasure-кодирование, коды Reed-Solomon

1

зоны доступности

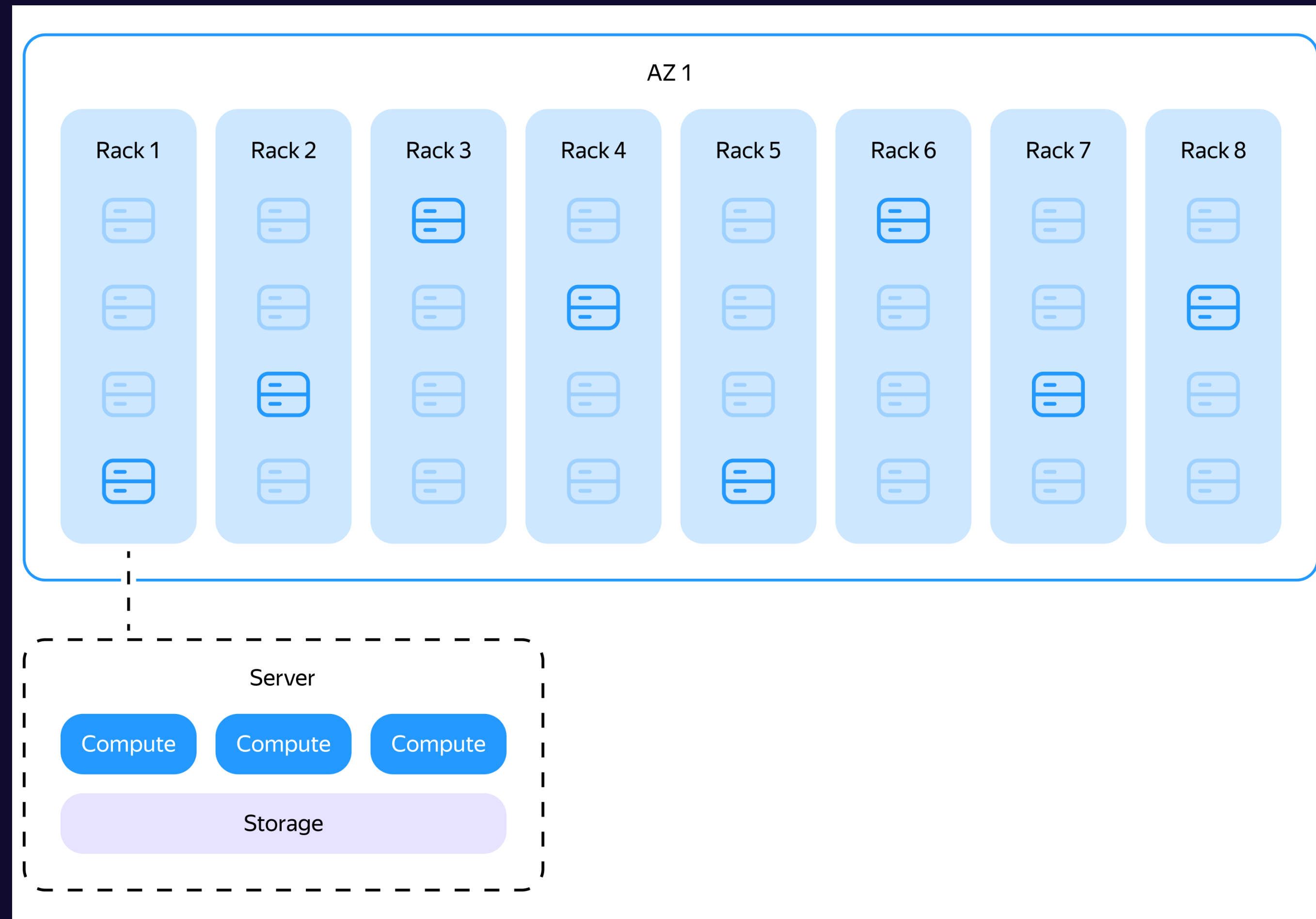
8+

узлов хранения

×1,5

множитель объёма 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
- Геораспределённый 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
- Геораспределённый 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
- Геораспределённый 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 2024 году

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

Почему не Kafka в 2024 году

- Отказ от Zookeeper,
но Kraft ещё 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 2024 году

- Отказ от Zookeeper, но Kraft ещё не доработан
- Геораспределённый 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 2024 году

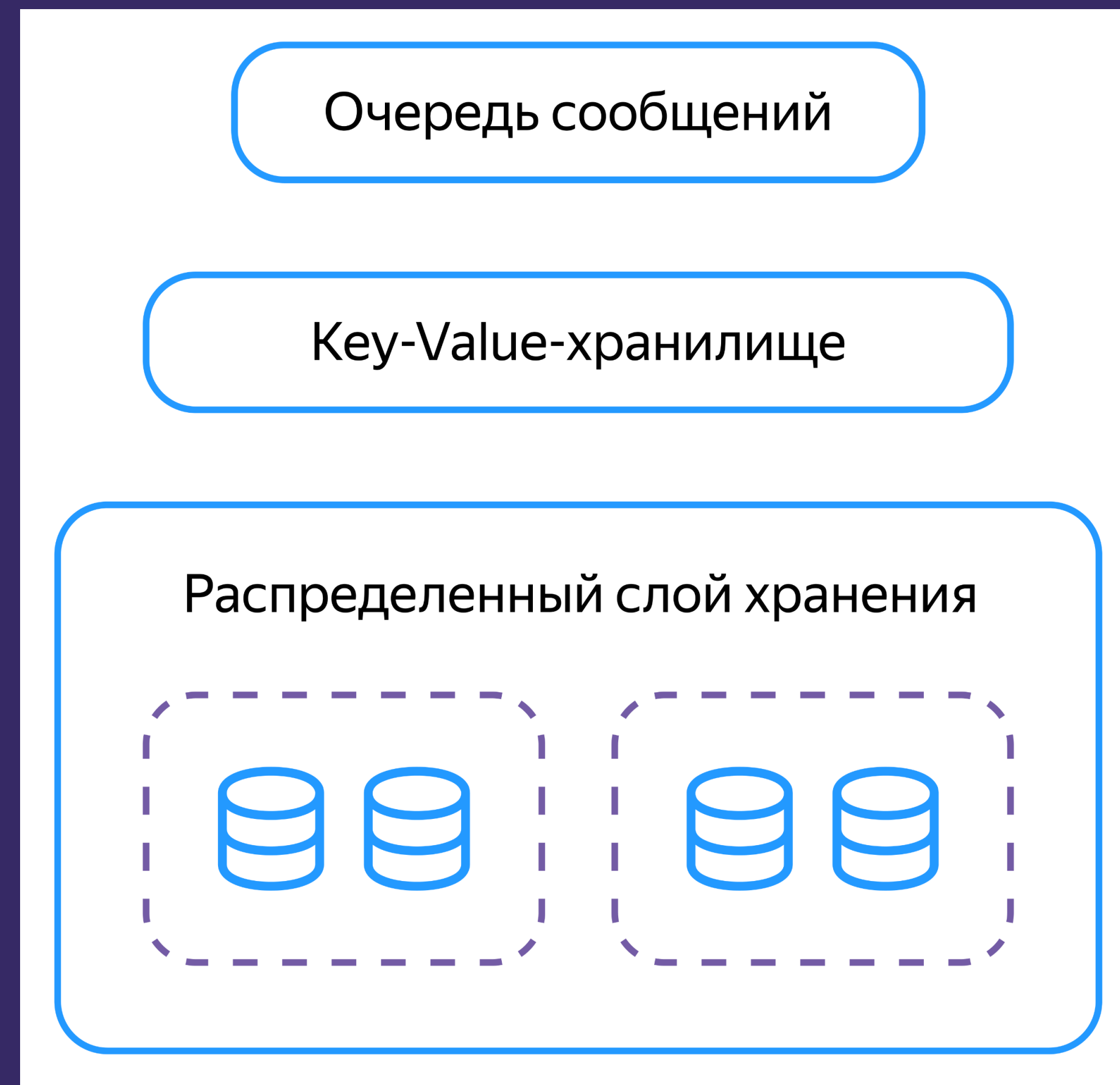
- Отказ от Zookeeper, но Kraft ещё не доработан
- Геораспределённый кластер
- Ограниченный exactly-once
- Ограниченный мониторинг
 - Нет метрик по клиентам, только по брокерам и топикам
 - Не годится для serverless

Почему не Kafka в 2024 году

- Отказ от Zookeeper, но Kraft ещё не доработан
- Геораспределённый 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 производительности
на разных тестах



Стабильность

Гарантия неизменности
производительности между
разными версиями
(во времени)



Бенчмарк оборудования

Сравнение производительности
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)

Максимизируется скорость,
невзирая на латентность

100 Кбит/с

без нагрузки

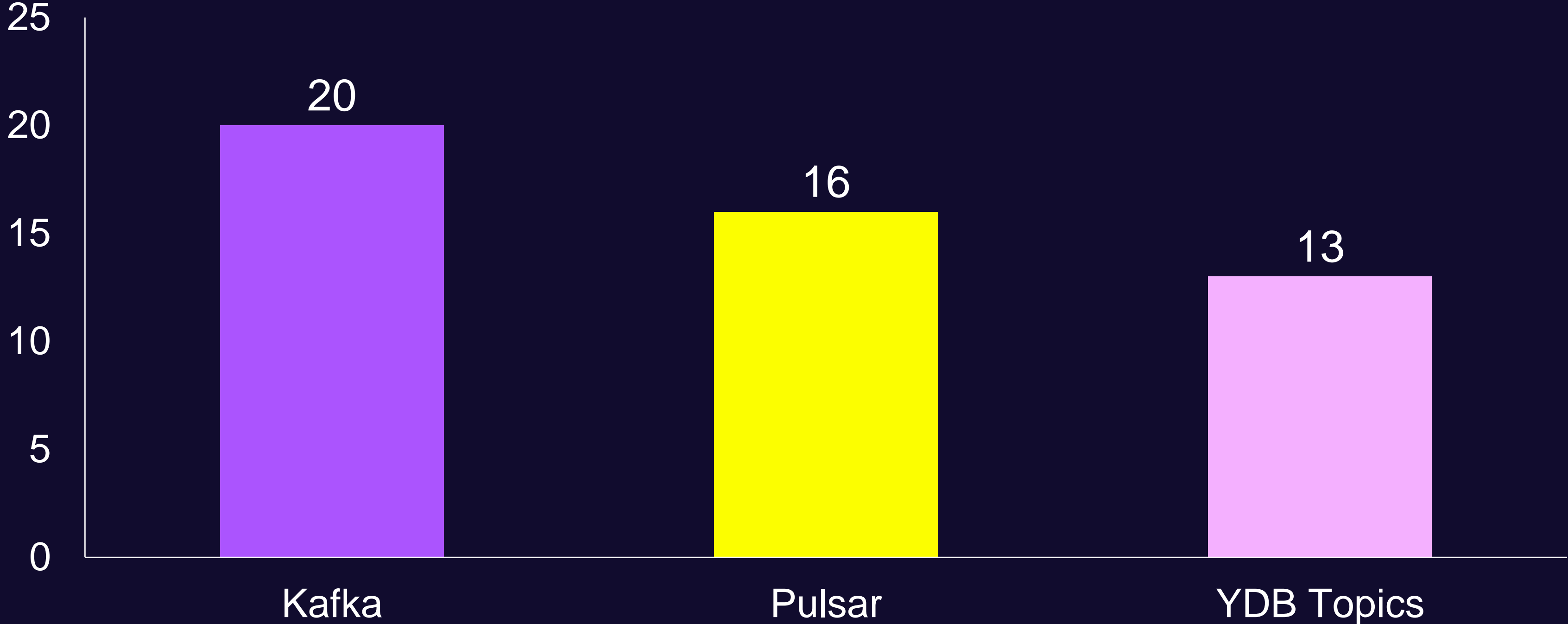
6,4 Гбит/с

под нагрузкой
(порядка 1/3
от максимальной)

Первые результаты по скорости

Узкое место: сеть

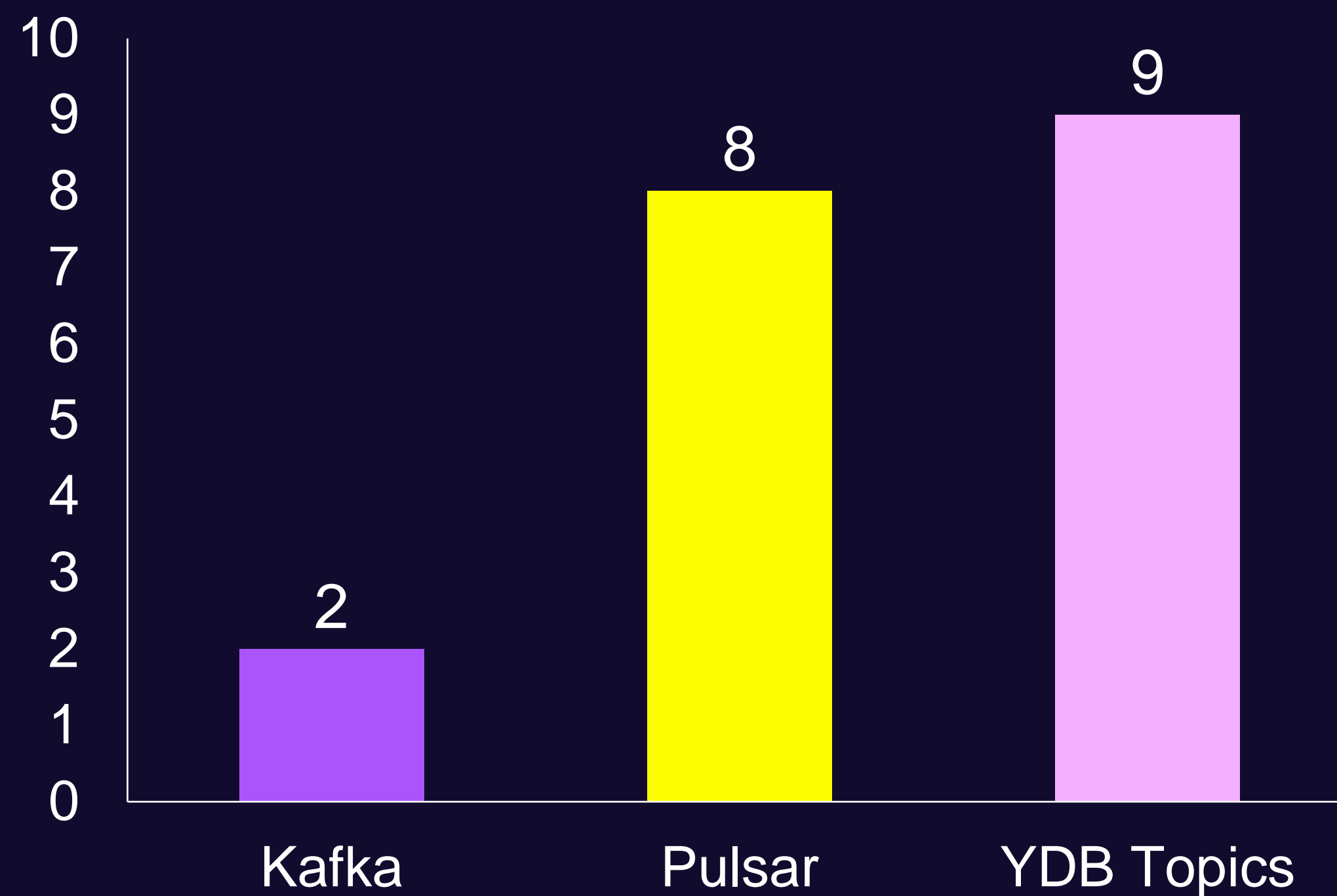
Максимальная скорость, Гбит/с



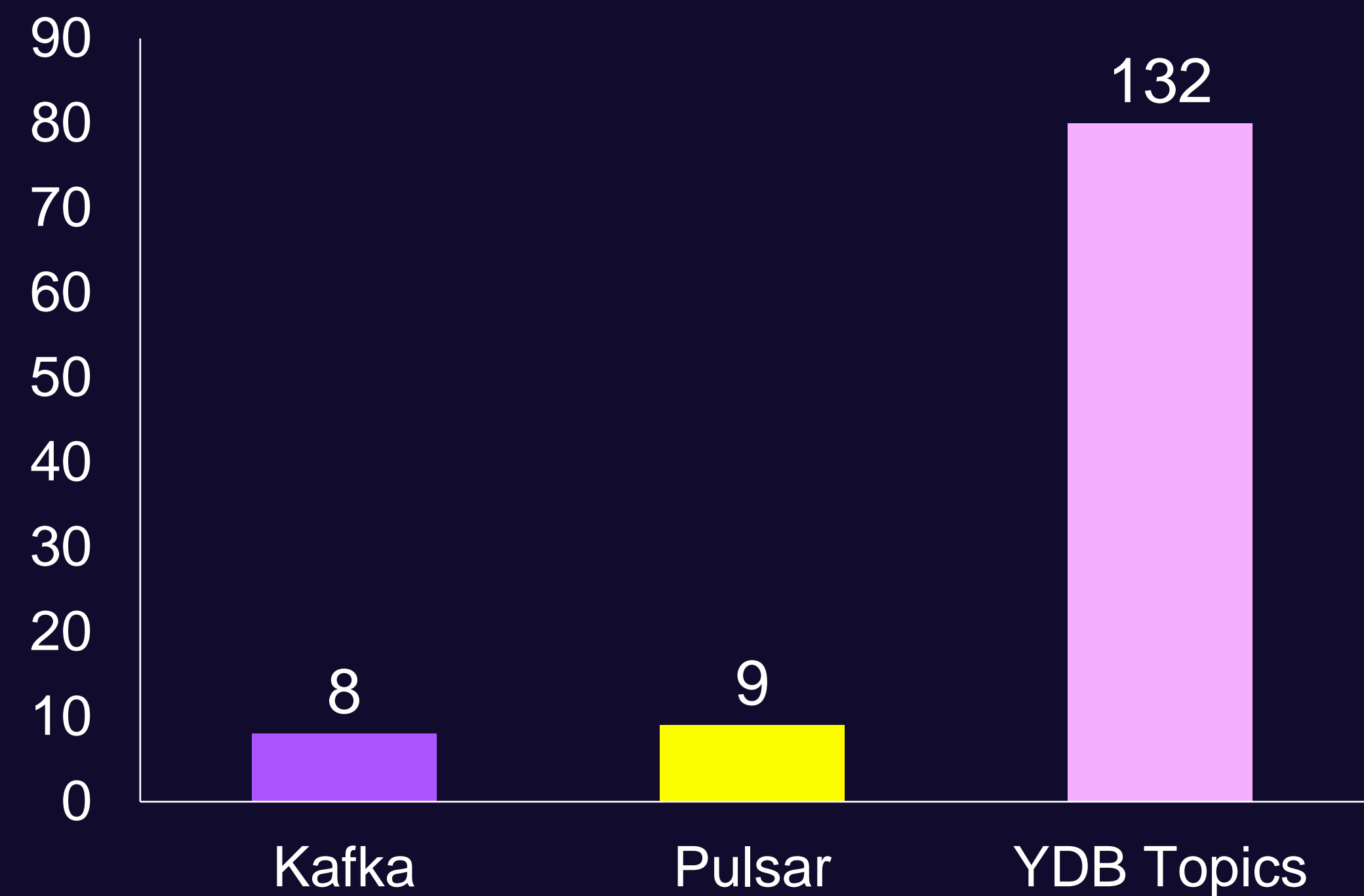
Первые результаты по времени

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

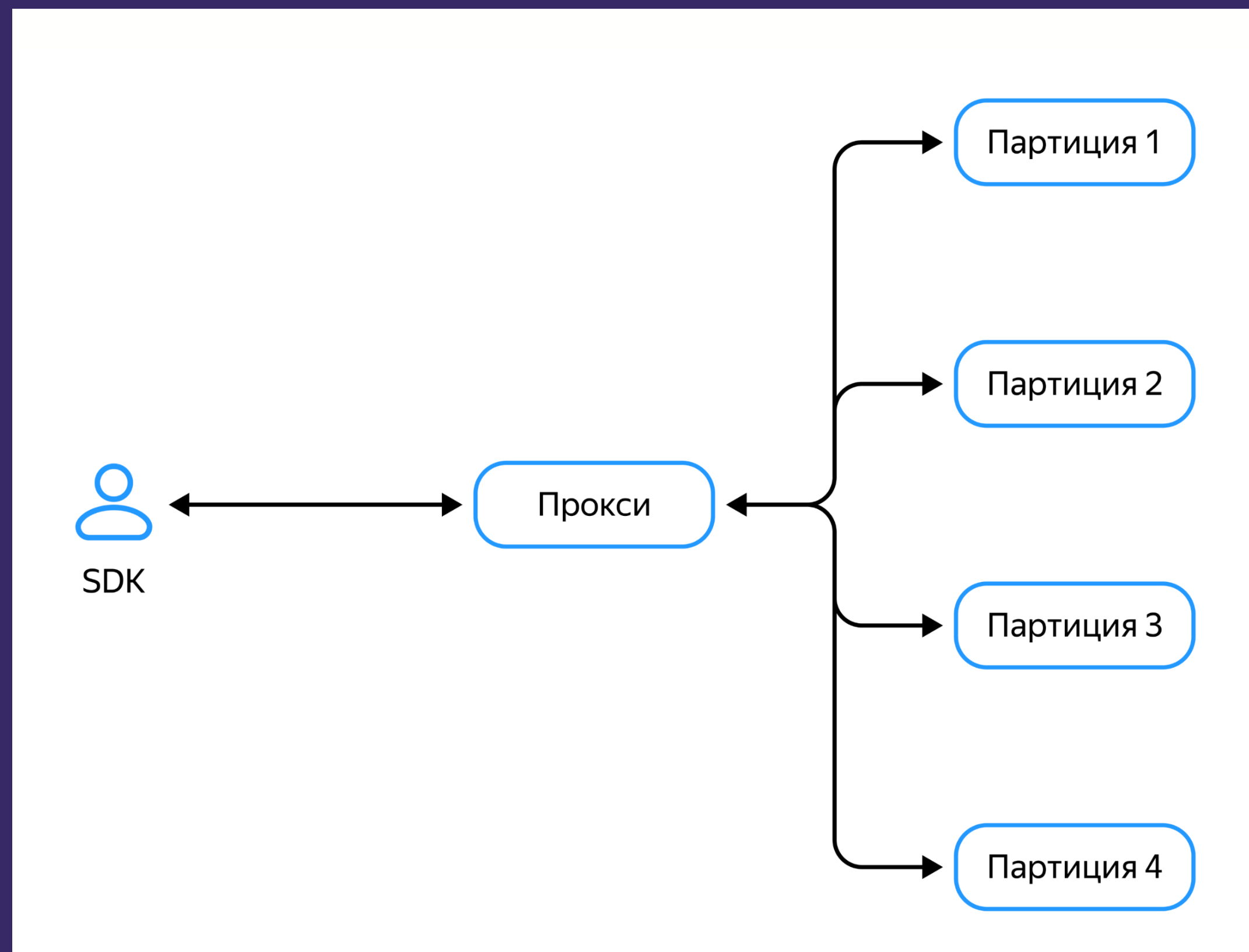


clck.ru/362NSn

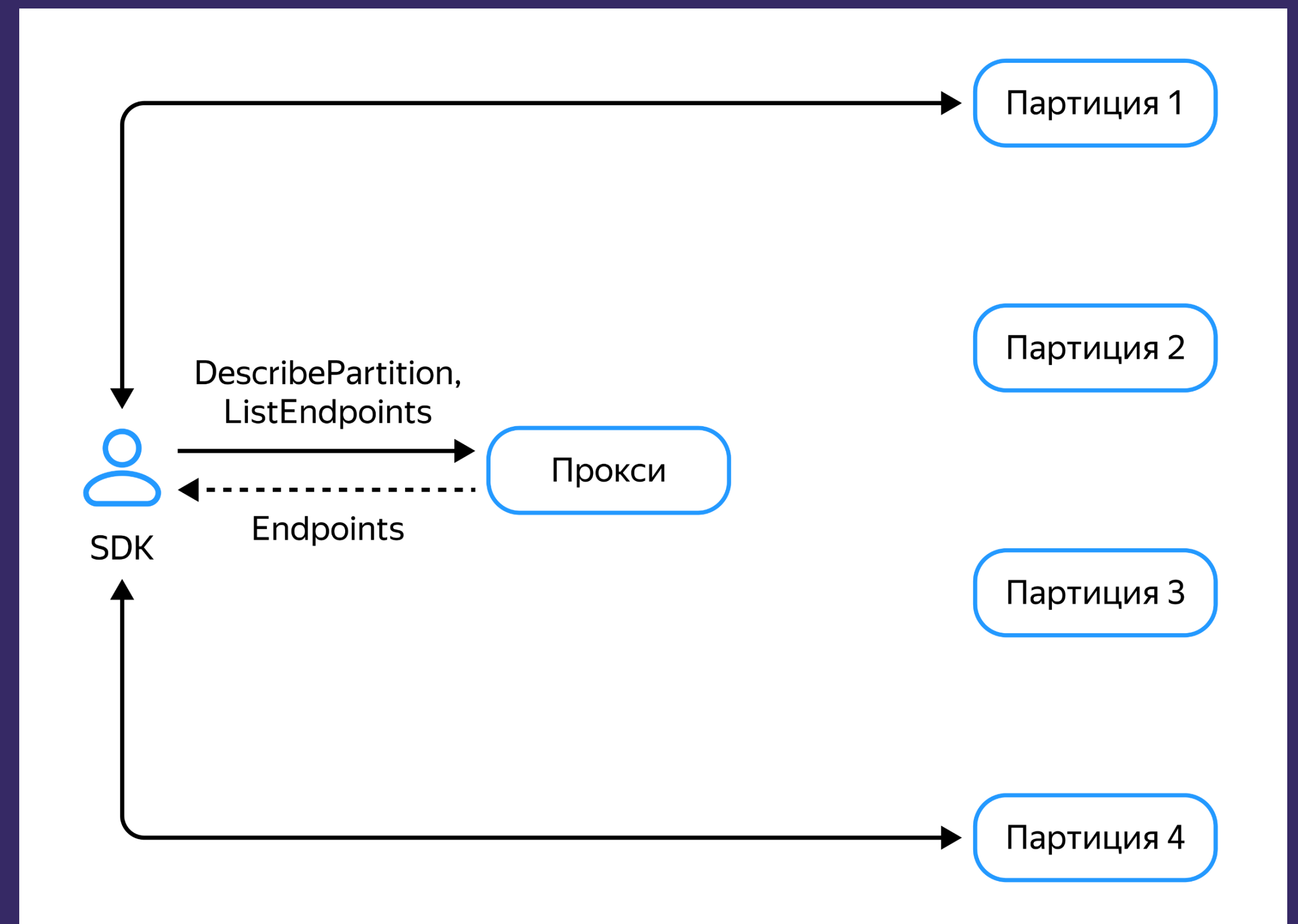
Прямая запись в партицию

Результат: экономия сети и процессора

До



После



Профилирование: флеймграф

```
all
kikimr.User
[unknown]
__clone
start_thread
NActors::TExecutorThread::ThreadProc
operator
bool NActors::TExecutorThread::Execute<NActors::TMailboxTable::THTSwapMailbox, false>
NKikimr::NPQ::TPartition::StateWrite
NKikimr::NPQ::TPartition::HandleWriteResponse
NKikimr::NPQ::TPartition::HandleWrites
NKikimr::NPQ::TPartition::ProcessWrites
NKikimr::NPQ::TPartition::AddNewWriteBlob
NKikimr::NPQ::CheckBlob NKikimr::NPQ::TBatch::Serialize
NKikimr::NPQ::T.. TBlobI.. operator+
NKikimr::NPQ::T.. NKik.. TBasicString<char, std::__y1::char_traits<char> >& TBasicSt..
TBatch TBatch TBasicString<char, std::__y1::char_traits<char> >::append
TBasicString TBas.. std::__y1::basic_string<char, std::__y1::char_traits<char>, ..
TIntrusivePtr<.. TIntr.. std::__y1::basic_string<char, std::__y1::char_traits<char>, ..
TStdString<con.. TStdS.. std::__y1::basic_string<char, std::__y1::char_traits<char>, s..
basic_string basic.. std::__y1::char_traits<char>::copy
std::__y1::basic.. std:.. __memmove_avx_unaligned_erms_rtm
std::__y1::cha.. std:..
__memmove_.. __..
```

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

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 $\times 1,5$ хранимого места при гарантиях доступности, сравнимых с $\times 3$ 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

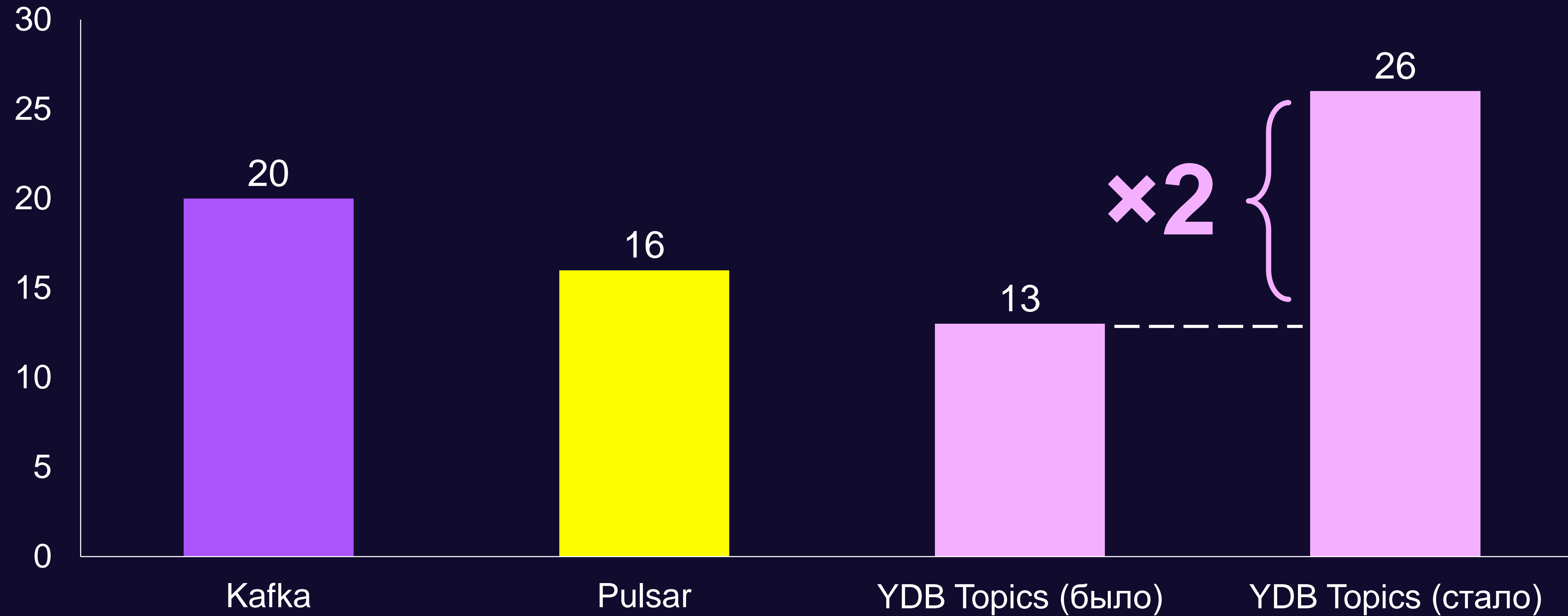
ускорено декодирование
с 1,7 Гбайт/с до 3 Гбайт/с



**Обновлённое нагрузочное
тестирование**

Максимальная скорость

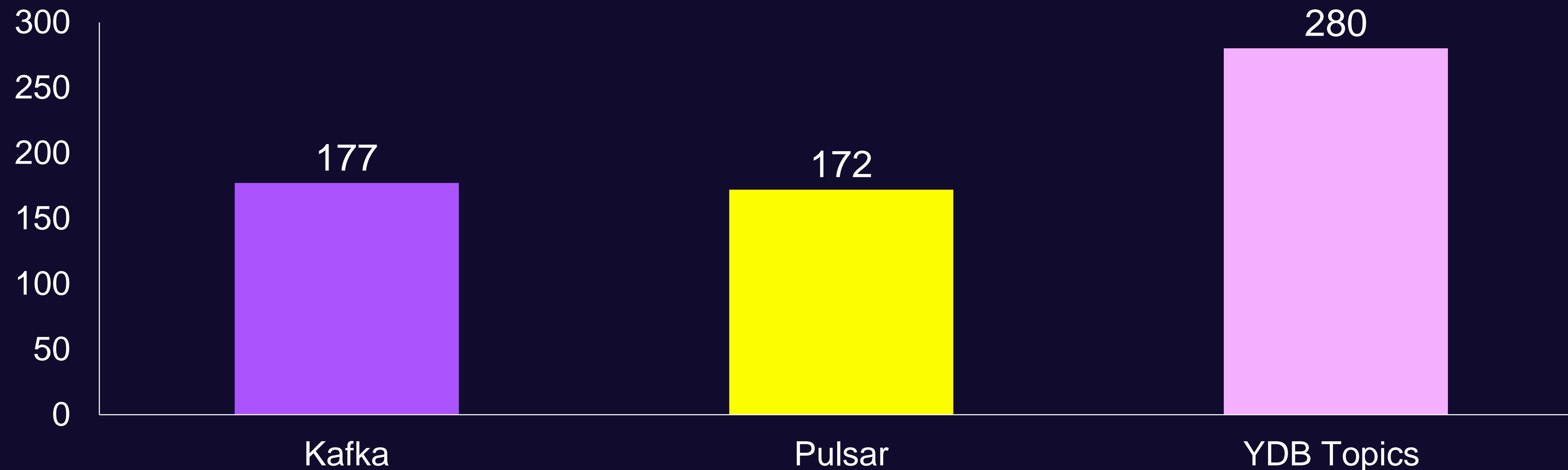
Гбит/с



Эффективность использования диска при длительной эксплуатации

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

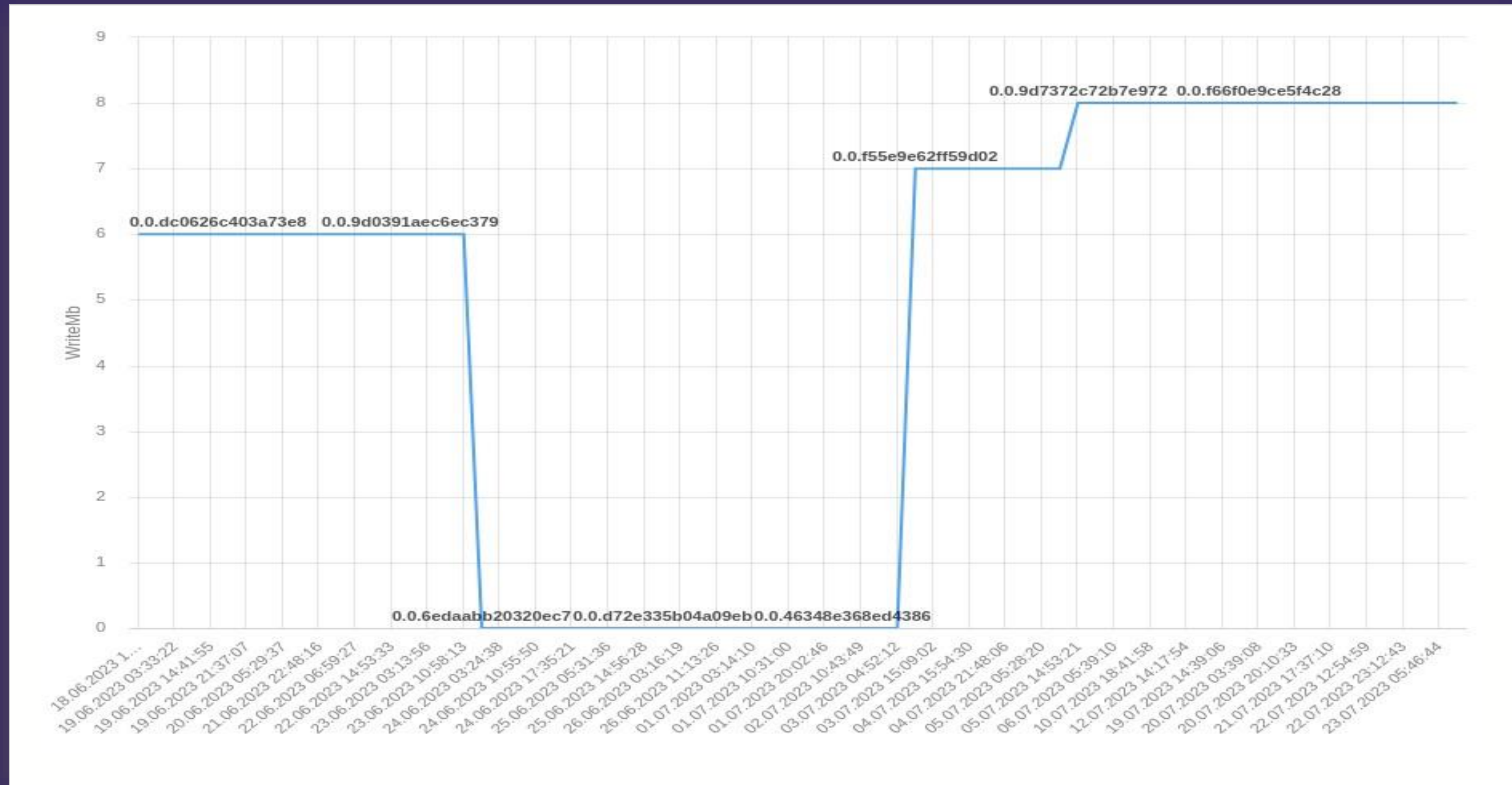
Максимальная скорость, Мбайт/с



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





Kafka API

Kafka API







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

- Мы добавили Kafka API*
- Сохранили преимущества платформы YDB и повышенную производительность
- Можно **бесшовно мигрировать** с Kafka на YDB Topics
- Получили Kafka в облаке в режиме **serverless/dedicated**
- Доступно в openсорс

Работайте, как с Kafka

Работают любые
привычные инструменты:

-  Kafka Cli
-  Kafka Connect
-  Logstash
-  Fluent Bit

...

Можно использовать
Kafka SDK

+ теперь можно
Serverless Kafka

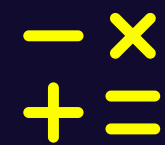


**Доступны в облаке:
Yandex Data Streams**

Что такое Serverless

Serverless — стратегия организации облачных услуг, при котором облако автоматически и динамически управляет выделением ресурсов в зависимости от нагрузки

Не нужно думать
о выделении ресурсов



Платишь за реальное
потребление



Легко масштабировать



Дёшево!



Yandex Data Streams

Serverless Kafka

И не только

- Просто сделать
- Платишь за реальное потребление
- Free tier
 - Скорость записи в сегмент 128 Кб/с, время хранения — час
- Легко масштабировать

Yandex Data Streams — больше, чем Serverless Kafka

Yandex Data Streams — часть платформы YDB:

- Таблицы и потоки данных — в одной БД
– Change Data Capture
- Управление доступом в рамках одной системы
- Интеграции с другими сервисами облака
- Обеспечиваем различные гарантии, в том числе exactly-once
- **Новое:** транзакции между топиками и таблицами
- **Новое:** масштабирование потоков данных с гарантиями порядка

Почему Yandex Data Streams, а не Kafka On-Premise

Не нужно думать о:

1 Kafka (брокеры,
координаторы, реплики)

2 ZooKeeper (KRaft)

3 Выборы лидера

4 Перебалансировка партиций

5 Количество партиций

6 Группы читателей

7 Настройка брокера и клиентов

Почему Yandex Data Streams, а не Managed Kafka

Разрабатываем сами: можем поддержать Enterprise features

Уже есть:

- Exactly-once из коробки
- Нативная интеграция в облако (IAM)
- Метрики чтения (по потоку и по сегменту)
- Просмотр содержимого в веб-интерфейсе

Скоро будет:

- Транзакции потоки данных + таблицы
- Масштабирование потоков данных при нагрузке
- Можем сделать другое по запросу

Итоги

1

YDB Topics —
7 лет в проде
Яндекса как
основная очередь
сообщений

2

Доступны
в Yandex Cloud
в режиме
serverless/dedicated
как «Yandex
Data Streams»

3

Вышли
в опенсорс

4

Подтянули
производительность
до уровня аналогов
и даже местами
обогнали

Спасибо

за внимание!

Зевайкин Александр

Руководитель группы разработки

infra.conf