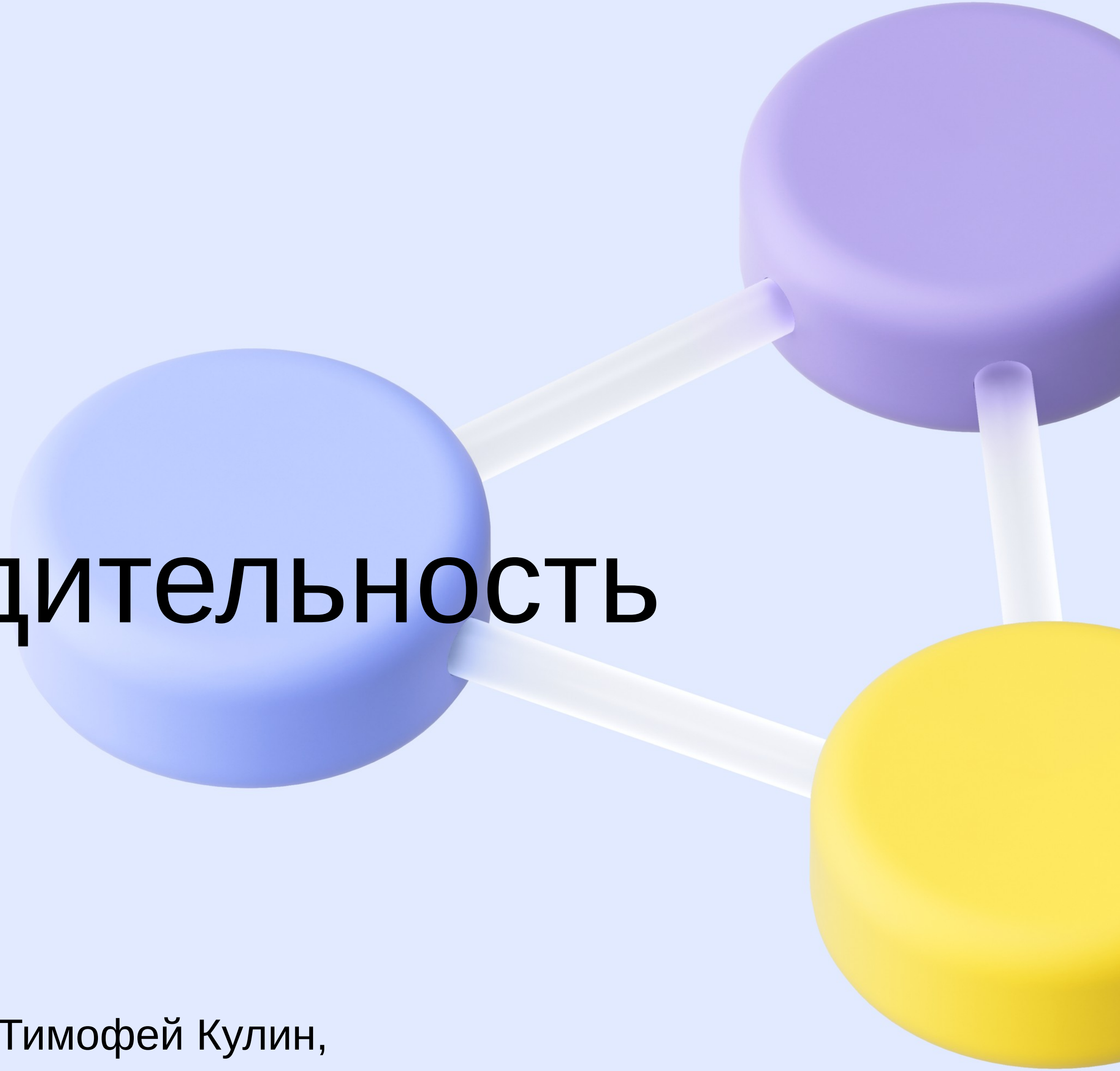


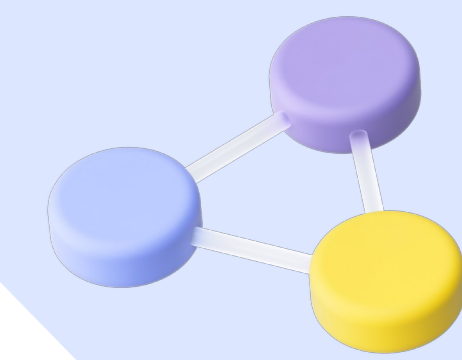
YDB Java SDK: борьба за производительность



Горшенин Александр,
YDB
Старший разработчик

Тимофей Кулин,
YDB
Старший разработчик

YDB — Open-Source Distributed SQL Database



- Распределенная
- Запускается в нескольких дата центрах (AZ)
- Переживает выключение одного дата центра + одной стойки без вмешательства человека
- Работа 24x7
- Обновление без простоев
- Строгая консистентность данных

YDB Java SDK — клиентская библиотека для YDB

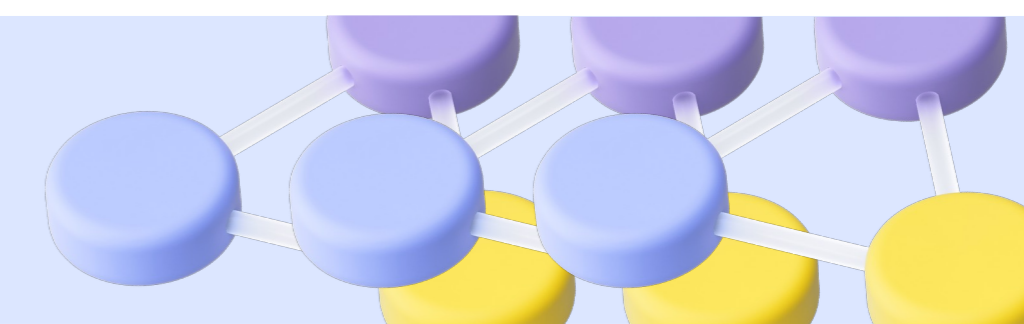
Одна из наиболее широко используемых SDK

- Спроектирована для создания высокопроизводительных приложений
- Построена поверх netty и grpc-java
- Реализует в себе всю сложную логику клиента распределенной БД, что упрощает код клиентского приложения

Сервисы уже использующие Java SDK

- Яндекс.Метрика
- Яндекс.Музыка
- Яндекс.Маркет
- Кинопоиск
- Алиса

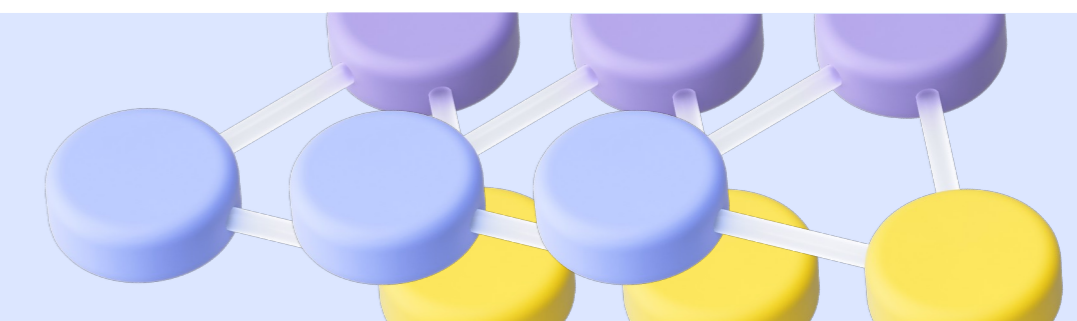
<https://github.com/ydb-platform/ydb-java-sdk>



YDB Java SDK — о чем мы расскажем в данном докладе

- Как проводились измерения
- Клиентская балансировка
- Сессии YDB и зачем они нужны клиенту
- Оптимизация пути выполнения запроса

<https://github.com/ydb-platform/ydb-java-sdk>



Тестовое приложение



Тестовое приложение для измерения скорости работы SDK

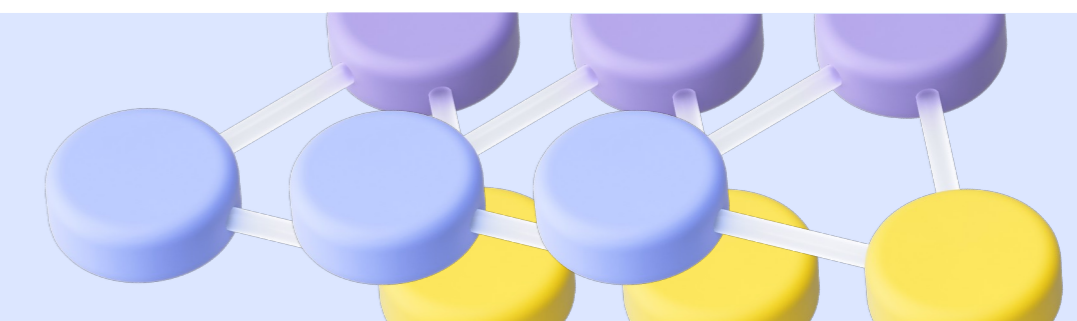
Особенности

- Настраиваемое число одновременно работающих потоков и размер записи в таблице
- Несколько возможных сценариев нагрузки
- Возможность использования разных версий Java SDK
- Предварительный разогрев пула сессий для получения несмещенных измерений

Основные сценарии нагрузки

- READ — синхронное чтение данных в параллельных потоках. Задается число одновременно запущенных потоков
- REACTIVE — асинхронное чтение данных. Используется пул потоков в размере количества процессорных ядер в системе. Задается число одновременно обрабатываемых задач

<https://github.com/alex268/ydb-performance-app>



Тестовые окружения в которых проводились измерения

Место запуска

- Рабочий ноутбук
- Небольшая виртуальная машина в Яндекс.Облаке (2 виртуальных ядра)
- Виртуальная машина в Яндекс.Облаке с большим количеством доступных процессоров (до 32 ядер)

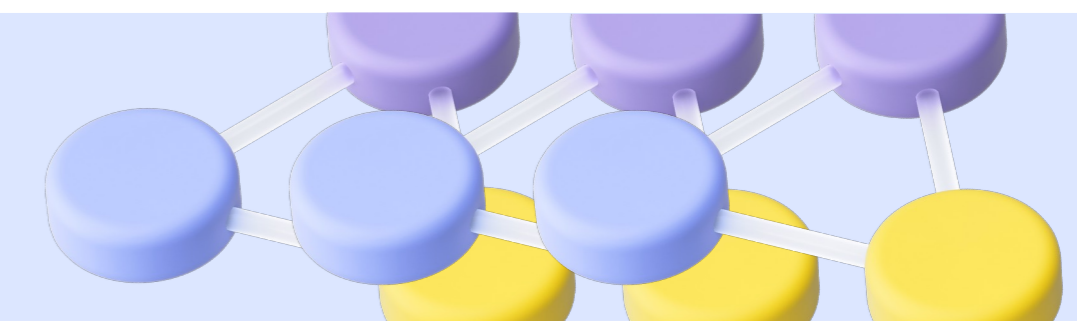
Размещение YDB

- Используется база данных в Яндекс.Облаке, распределенная по трем разным ДЦ

Тестовые данные

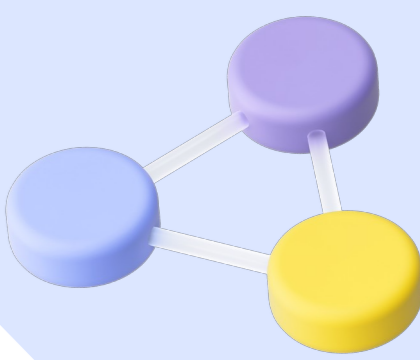
- Использовались три различные таблицы с длинами записей в 100, 500 и 2000 байт

<https://github.com/alex268/ydb-performance-app>

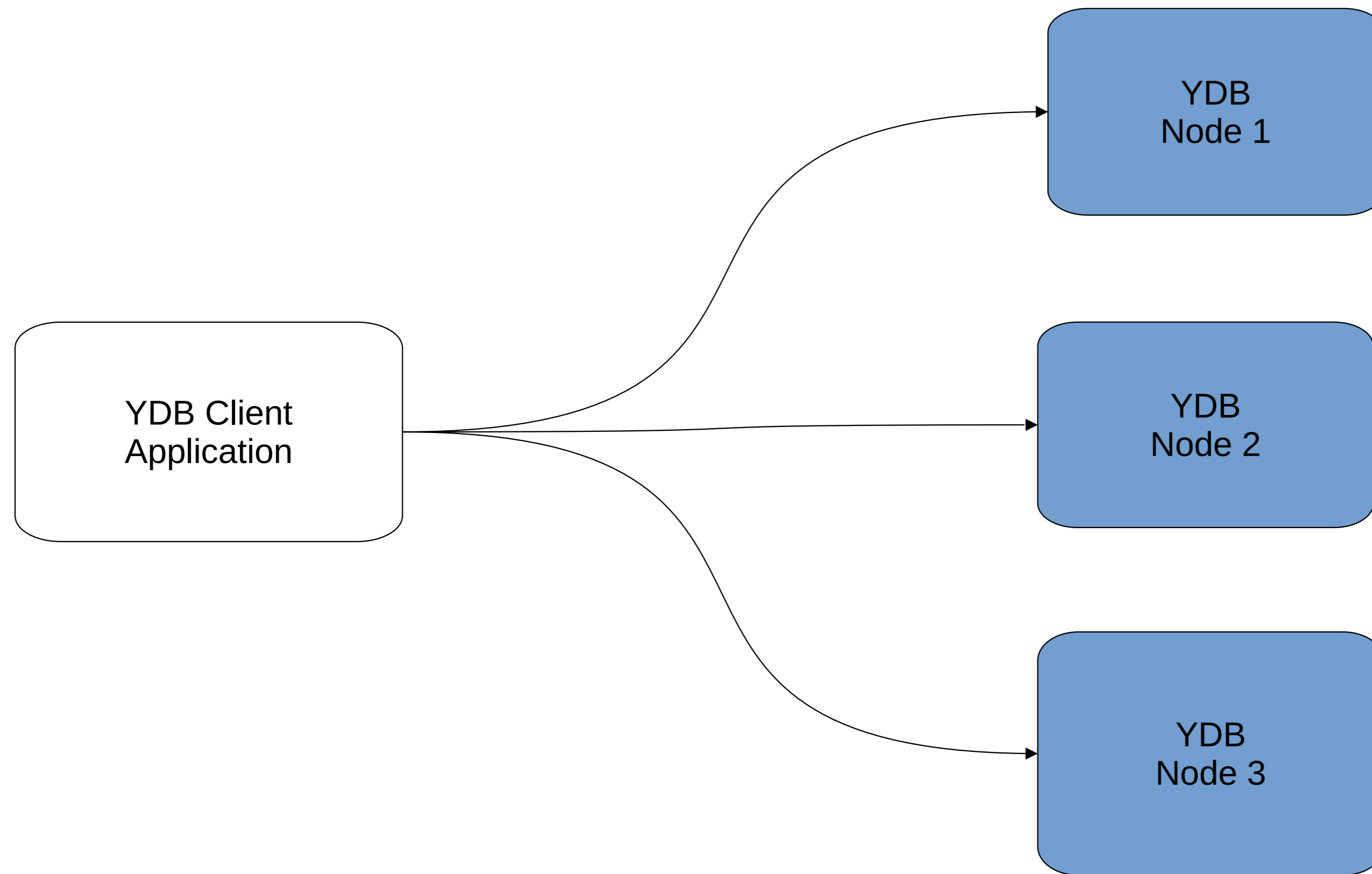


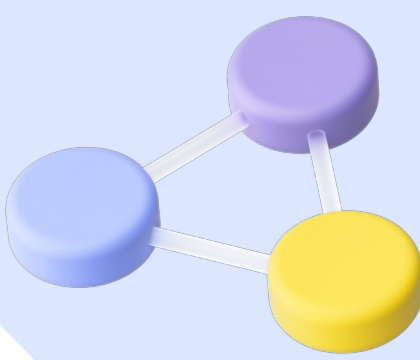
Балансировка нод



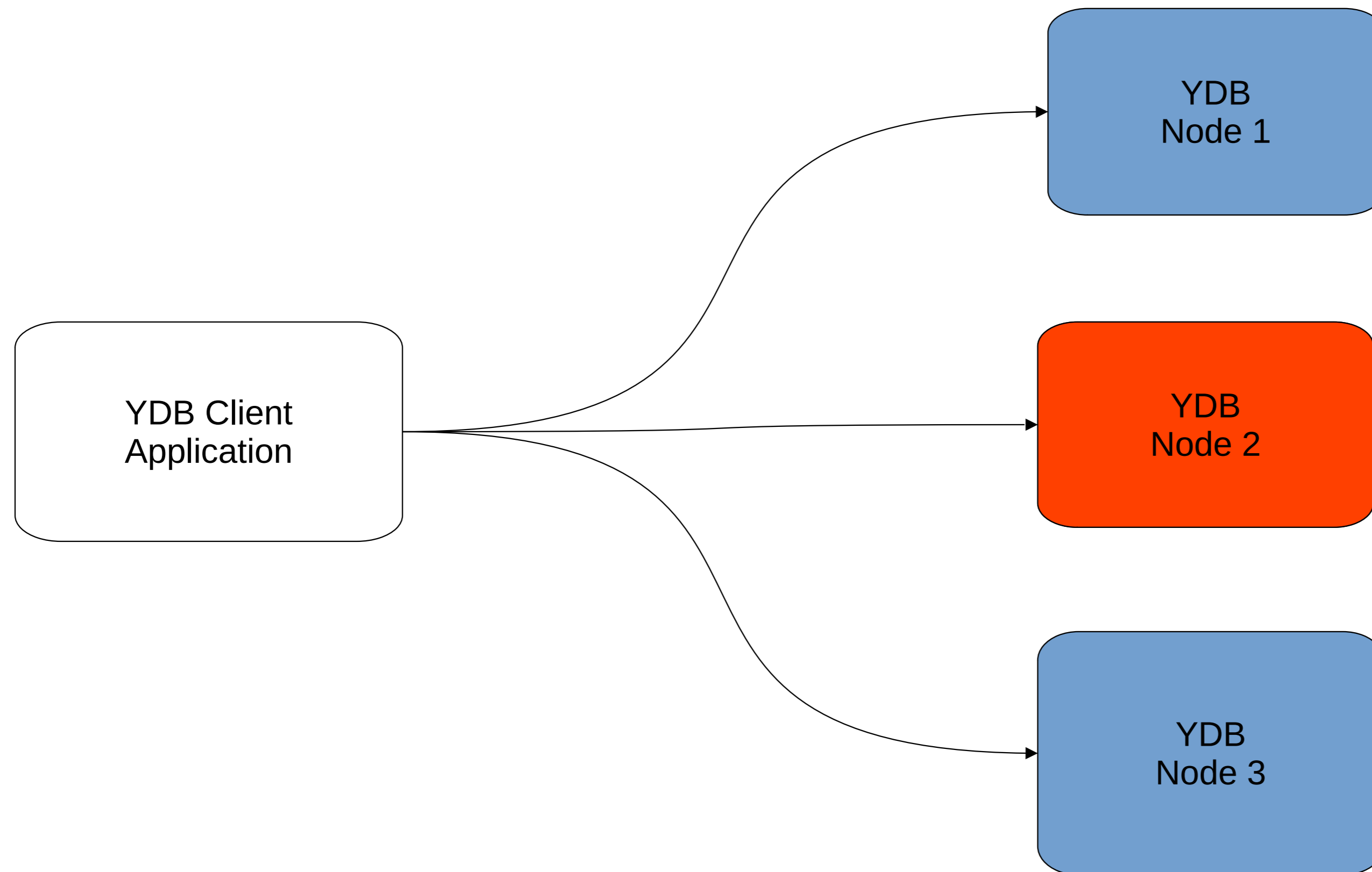


Балансировка запросов



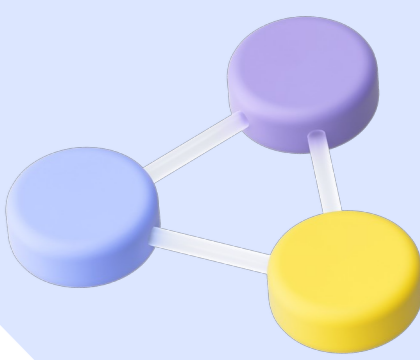


Отключение ноды

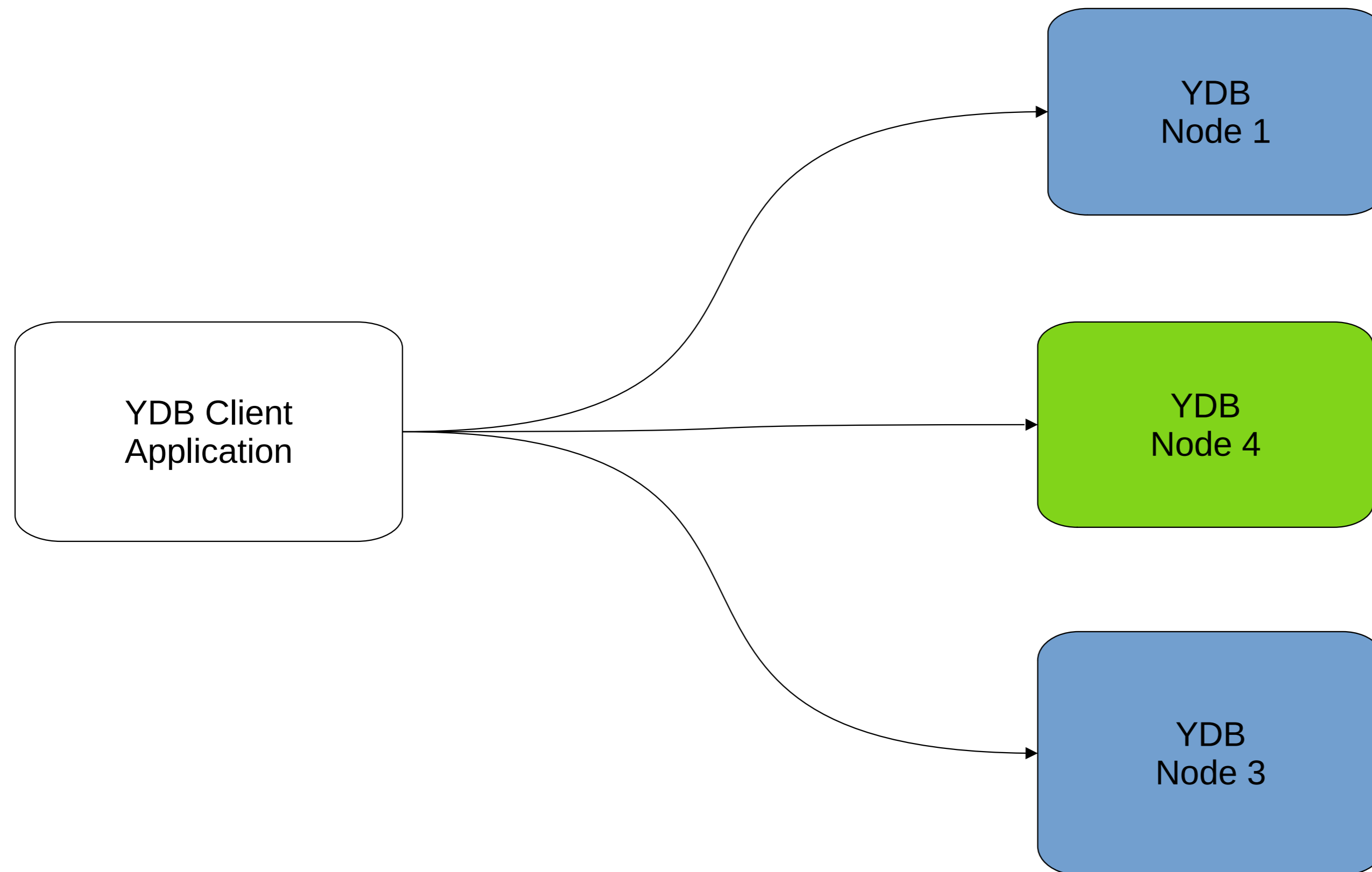


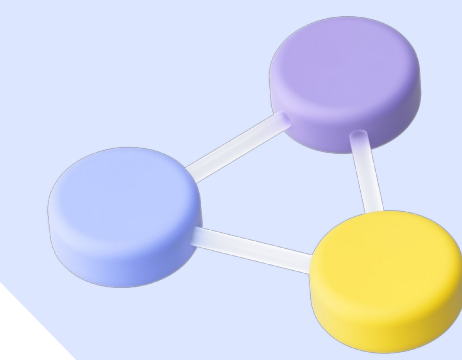
Причины

- Потеря связи
- Технические проблемы
- Обновление с рестартом
- Сервисное обслуживание



Добавление новой ноды





Больше информации об архитектуре YDB



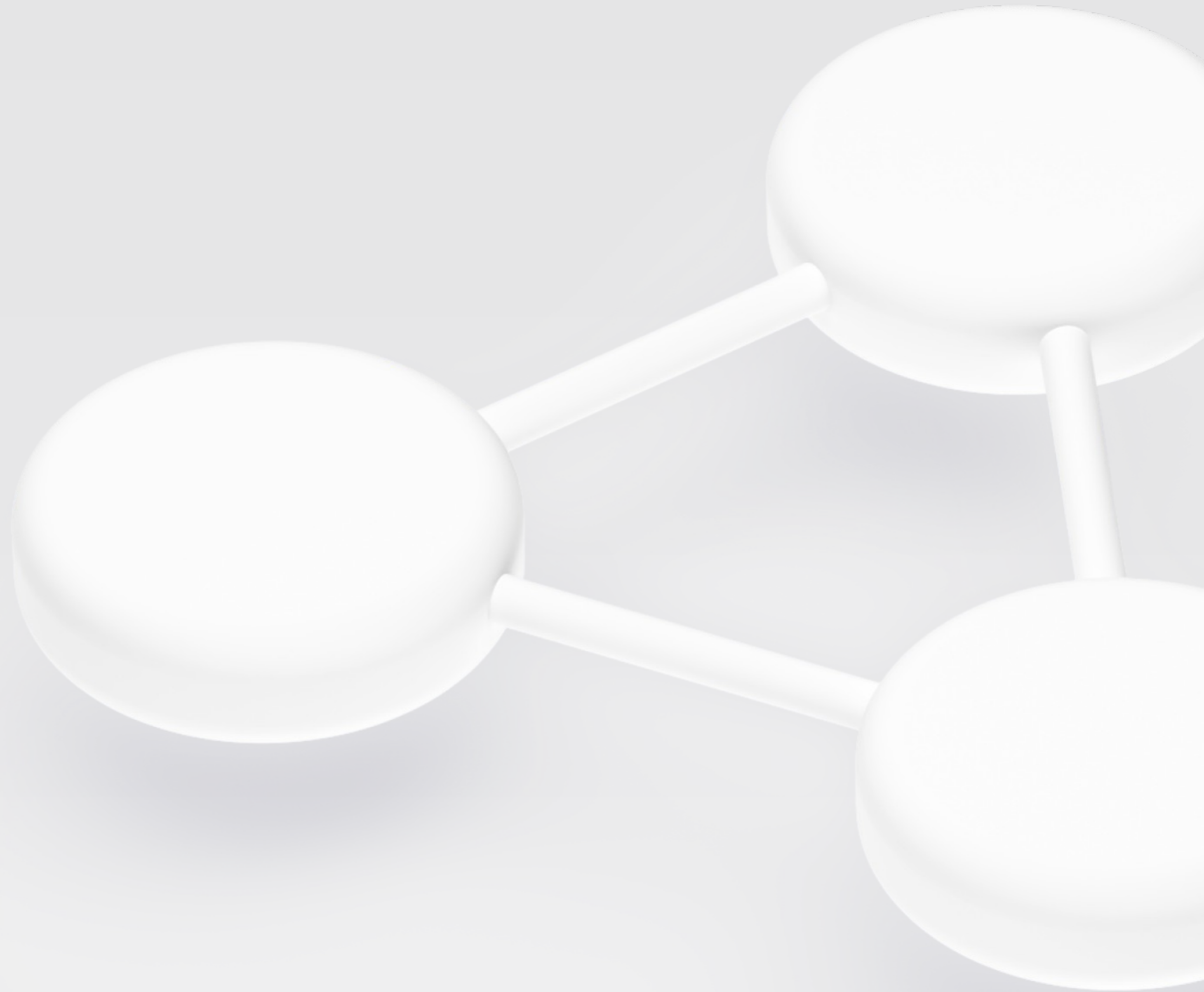
Доклад Андрея Фомичева на конференции Hydra 2021

Serverless nature of Yandex Database

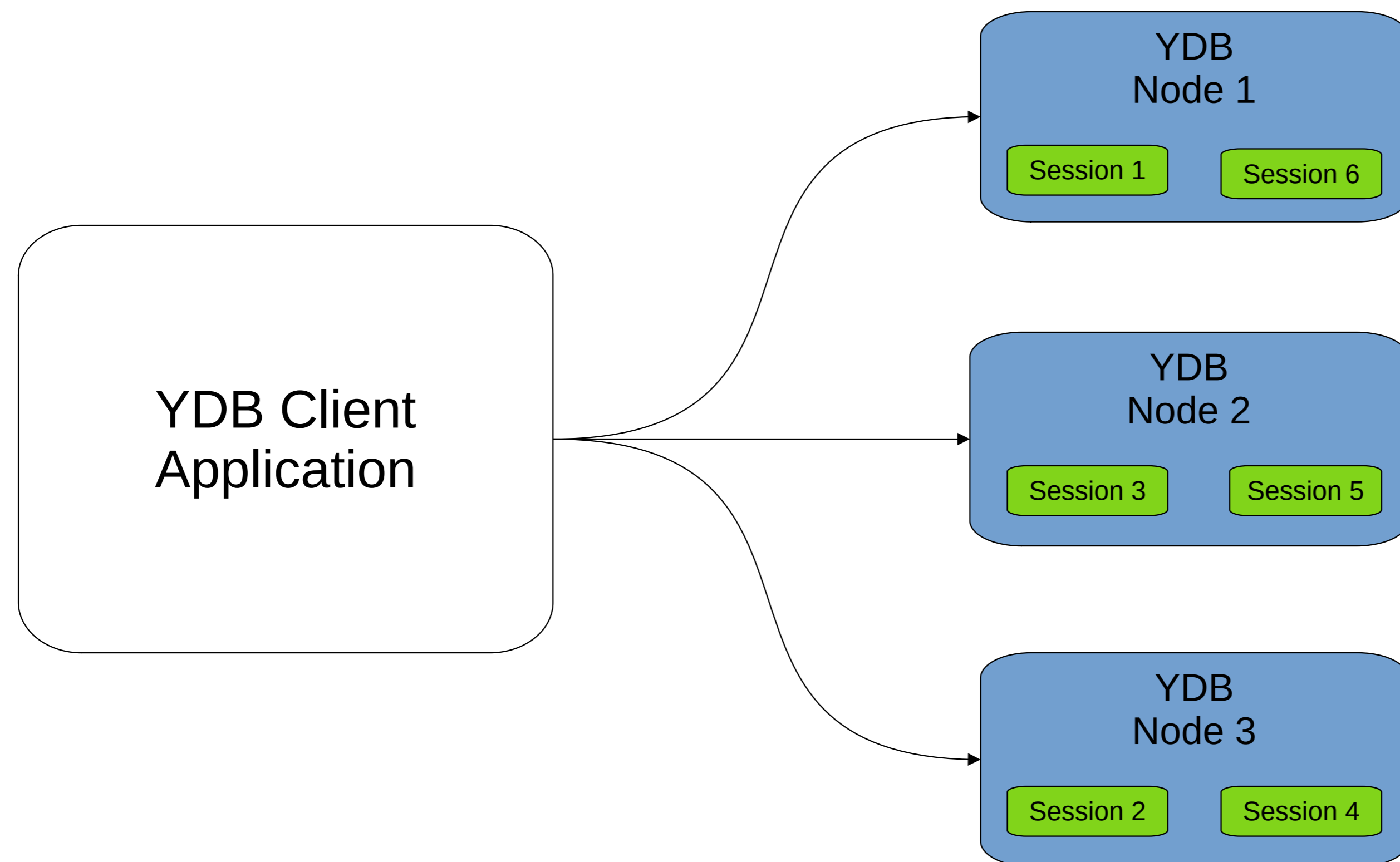
<https://hydraconf.com/talks/6NHcltWgIpZCYzUBzJjNG/>



Сессии YDB

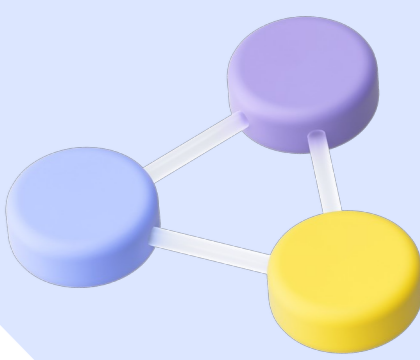


Сессии YDB

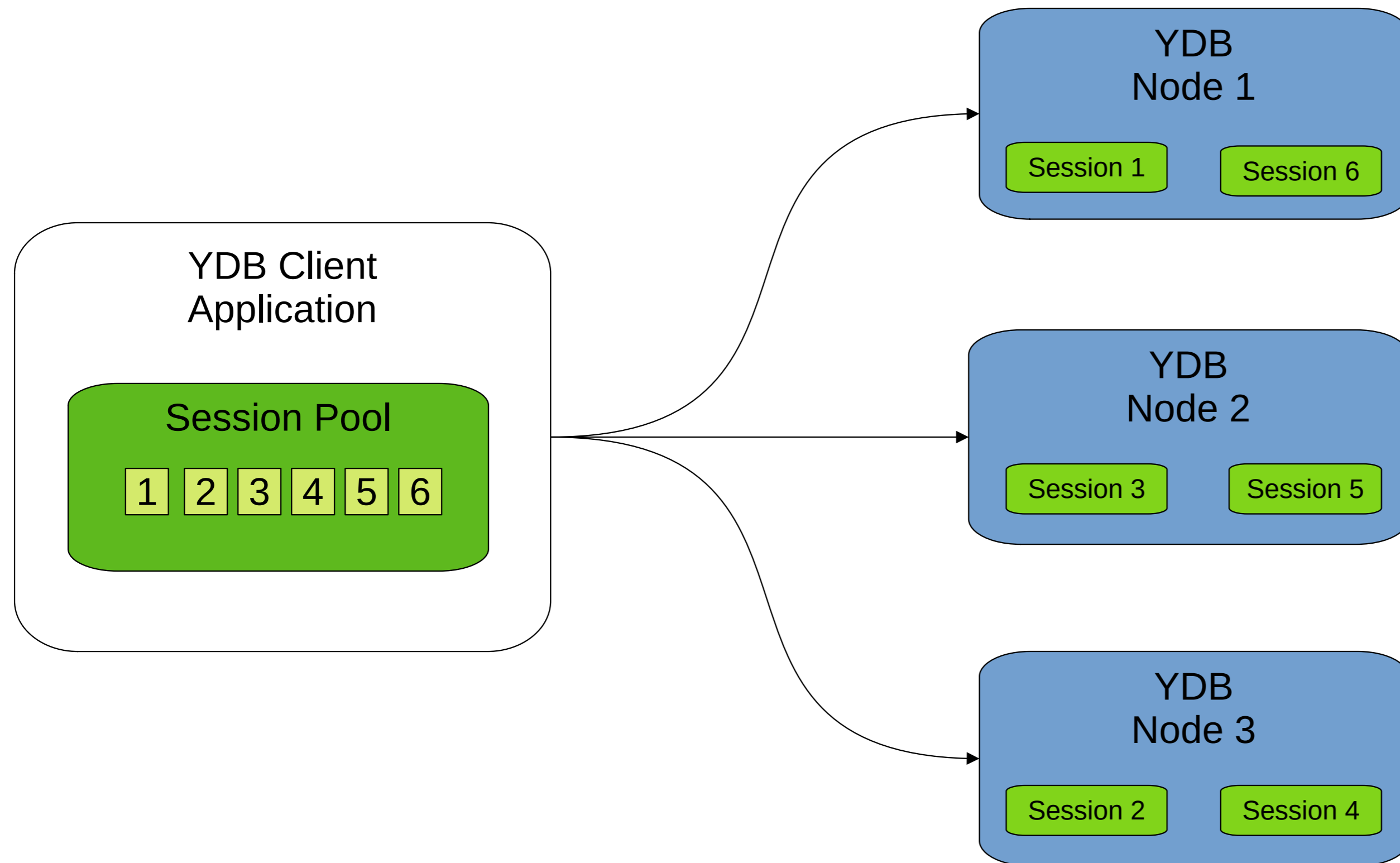


Каждая сессия

- Представляет из себя квант вычислительных ресурсов сервера
- Работает на одной конкретной ноде
- Создается и удаляется по требованию
- Может выполнять только один запрос в один момент времени



Пул сессий YDB

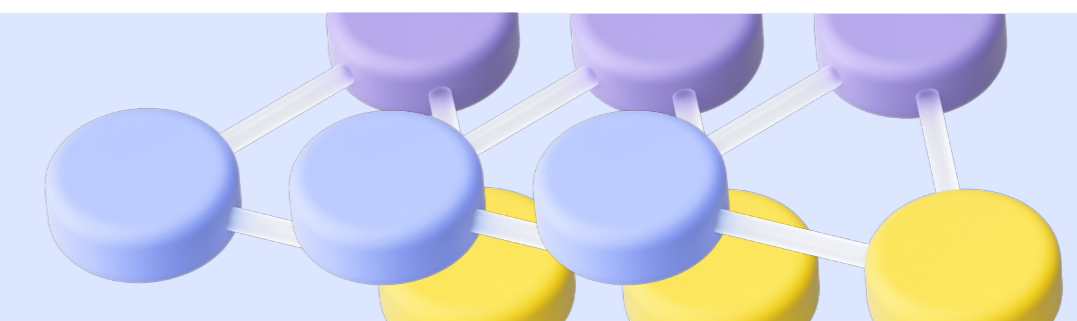


- Позволяет ускорить выполнение запросов за счет переиспользования уже существующих сессий
- Контролирует жизненный цикл сессии - создает новые сессии и удаляет невалидные или недоступные сессии
- Разумно выбранный максимальный размер пула сессий позволяет защитить базу YDB от ddos, вызванных ошибками в клиентских приложениях

Пул сессий в YDB Java SDK

Основные требования к пулу сессий

- Подразумевается что клиент создает его однажды и затем использует в течении всего времени жизни приложения
- Обращение к пулу может производиться из любого потока приложения — он должен быть thread-safe
- Сессии создаются и удаляются динамически — если все сессии заняты, но лимит не достигнут — пул должен делать запрос на создание новой. Если сессия долгое время не используется — пул должен закрывать ее
- Максимальный размер пула лимитирует число одновременно выполняемых запросов на сервере, но это не должно ограничивать число запросов к пулу. Если в конкретный момент времени все сессии заняты, запрос становится в очередь ожидания первой освободившейся сессии

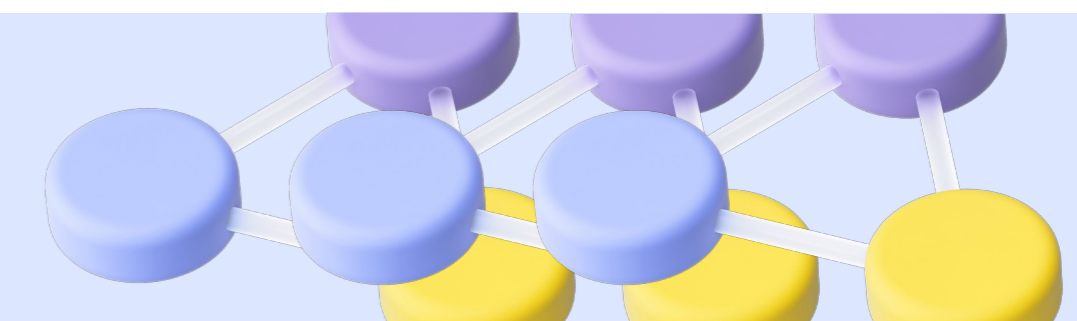


Пул сессий в Java SDK — с чего начали

Структура хранения

- Двухнаправленный список для хранения созданных сессий
- Коллекция для отслеживания текущих сессий в использовании
- Очередь ожидания
- Счетчик общего числа сессий
- Счетчик очереди ожидания

```
public final class FixedAsyncPool<T> {  
    private final Deque<PooledObject<T>> objects =  
        new LinkedList<>();  
    private final ConcurrentHashMap<T, T> acquired =  
        new ConcurrentHashMap<>();  
    private final Queue<PendingTask> pending =  
        new ConcurrentLinkedQueue<>();  
  
    private final AtomicInteger objectsCount =  
        new AtomicInteger(0);  
    private final AtomicInteger pendingCount =  
        new AtomicInteger(0);  
  
    ...  
}
```

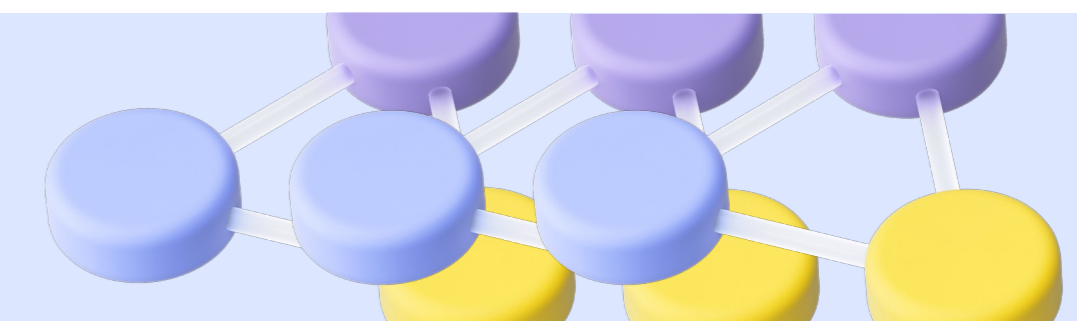


Пул сессий в Java SDK — с чего начали

Получение сессии

- Расчитываем время дедлайна
- Проверяем текущий размер пула
- И дальше пытаемся получить сессию
- Либо добавляем запрос в очередь ожидания

```
public CompletableFuture<T> acquire(Duration timeout) {  
    final CompletableFuture<T> promise =  
        new CompletableFuture<>();  
    final long deadlineAfter =  
        System.nanoTime() + timeout.toNanos();  
  
    int count = objectsCount.get();  
    while (count < maxSize) {  
        if (!objectsCount.compareAndSet(count,  
            count + 1)) {  
            count = objectsCount.get();  
            continue;  
        }  
  
        doAcquireOrCreate(promise, deadlineAfter);  
        return promise;  
    }  
    // try to create pending task  
    ...  
}
```



Пул сессий в Java SDK — с чего начали

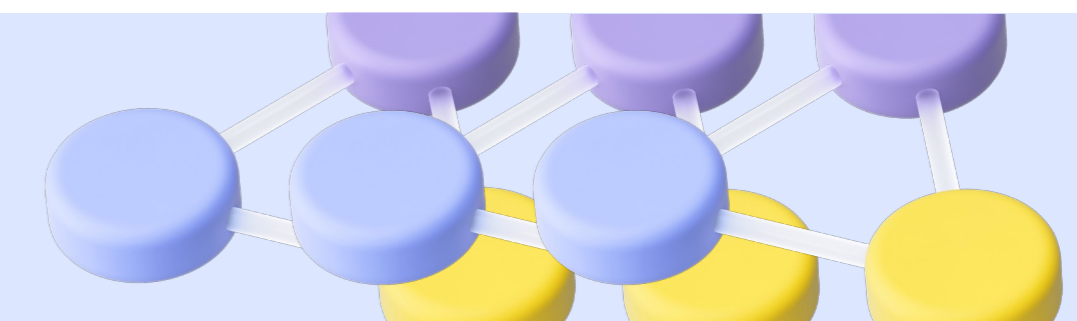
Получение сессии

- Пытаемся получить свободную сессию
- Если получилось — регистрируем ее как активную и возвращаем клиенту
- Если нет — то отправляем запрос на создание новой сессии

```
private void doAcquireOrCreate(
    CompletableFuture<T> promise, long deadline) {

    final PooledObject<T> object = pollObject();
    if (object != null) {
        acquired.put(object, object);
        promise.complete(object.getValue());
        return;
    }
    // no objects left in the pool, so create new one
    ...
}

private PooledObject<T> pollObject() {
    synchronized (objects) {
        return objects.pollLast();
    }
}
```

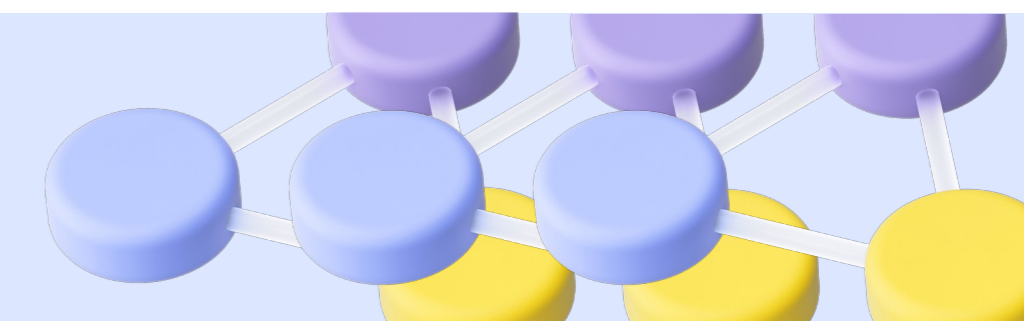


Пул сессий в Java SDK — новый вариант

Структура хранения

- LockFree список для idle сессий
- Множество для отслеживания сессий в использовании
- Очередь ожидания
- Один общий счетчик размера пула

```
public class WaitingQueue<T> {  
    private final ConcurrentLinkedDeque<T> idle =  
        new ConcurrentLinkedDeque<>();  
    private final Map<T, T> used =  
        new ConcurrentHashMap<>();  
    private final Map<CompletableFuture<T>,  
        CompletableFuture<T>> pendingRequests =  
        new ConcurrentHashMap<>();  
  
    private final AtomicInteger queueSize =  
        new AtomicInteger(0);  
    ...  
}
```



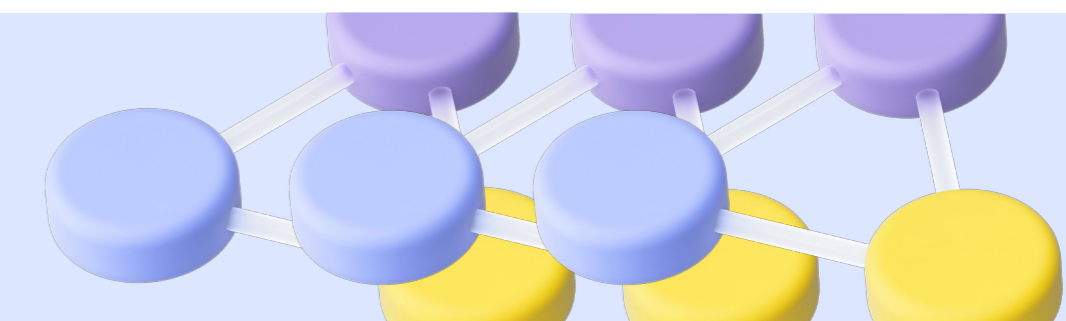
Пул сессий в Java SDK — НОВЫЙ ВАРИАНТ

Получение сессии

- Сначала пытаемся получить сессию из списка idle
- Для этого запрашиваем ее из LockFree коллекции
- И если такая нашлась — помечаем ее как в использовании и отправляем клиенту
- Если не получилось получить idle сессию — последовательно пытаемся отправить запрос на создание сессии или добавить запрос в очередь ожидания. Если ни один из вариантов не сработал — сообщаем об этом клиенту

```
public void acquire(CompletableFuture<T> f) {  
    boolean ok = tryToPollIdle(f);  
  
    if (!ok) {  
        ok = tryToCreateNewPending(f)  
            || tryToCreateNewWaiting(f);  
        ...  
    }  
}
```

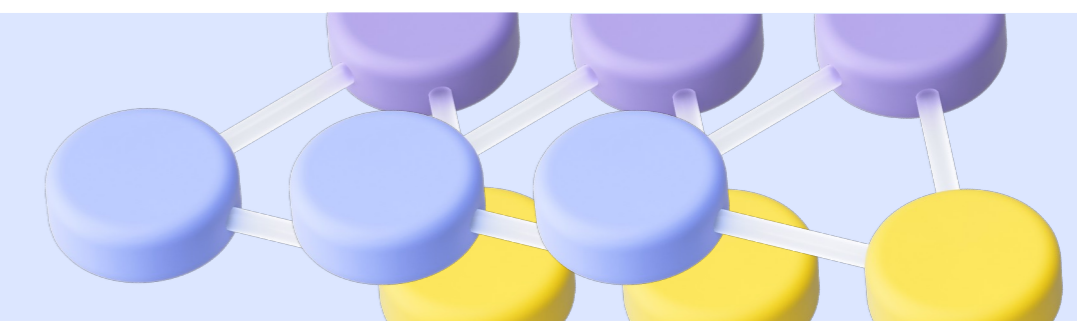
```
private boolean tryToPollIdle(CompletableFuture<T> f) {  
    T next = idle.pollFirst();  
    if (next != null) {  
        used.put(object, object);  
        f.complete(object);  
        return true;  
    }  
    return false;  
}
```



Пул сессий в Java SDK

Сравнение двух реализаций

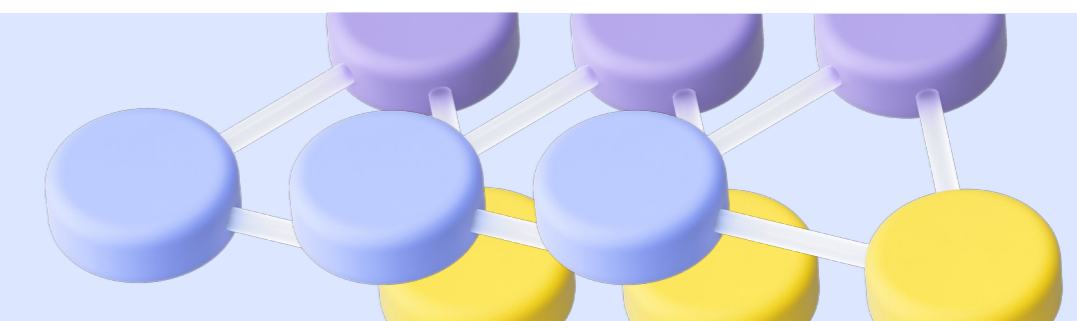
- Большая часть изменения — косметическая. Код немного упрощен, удалены лишние сущности
- Логика работы пула изменилась незначительно
- Самое важное изменения — убрали блокировку при изменении списка объектов в пуле

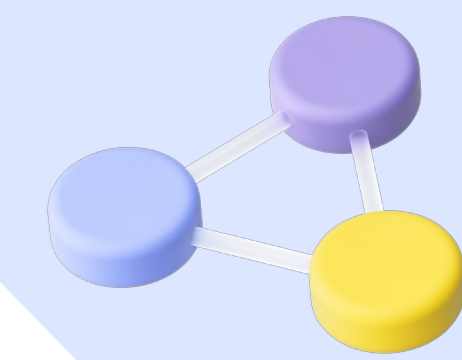


Тестирование пула сессий

Измерение разницы в скорости работы двух реализаций

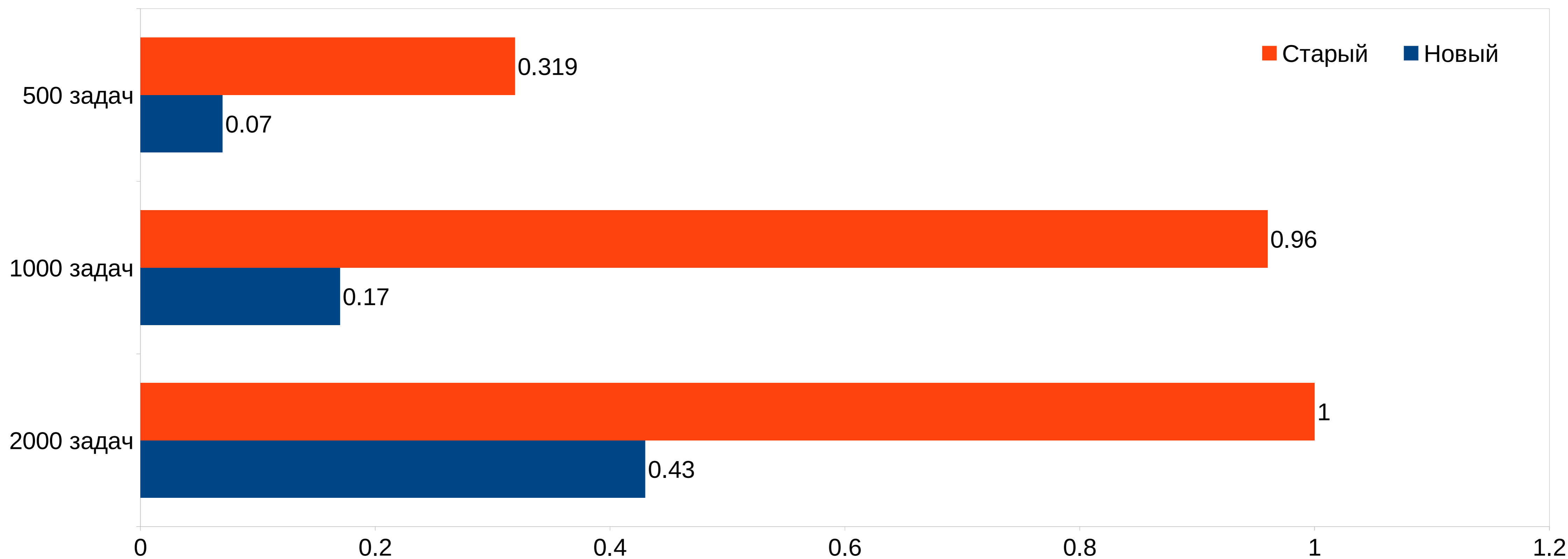
- Тестировать пул будем асинхронным чтением большого количества записей
- Все чтения выполняются в одном фиксированном пуле потоков с размером равным числу ядер процессора
- Измерять будем процессорное время в миллисекундах, прошедшее между запросом сессии и получением объекта сессии. Чем меньше время — тем лучше
- Пул предварительно разогрет — все сессии создаются заранее

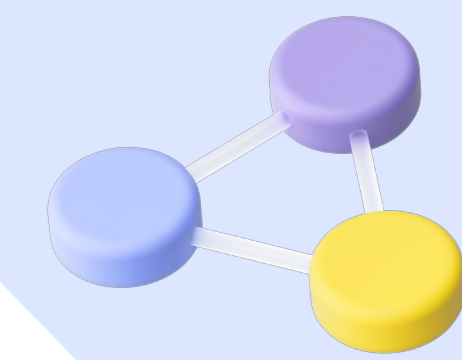




Оценка времени получения сессии из пула

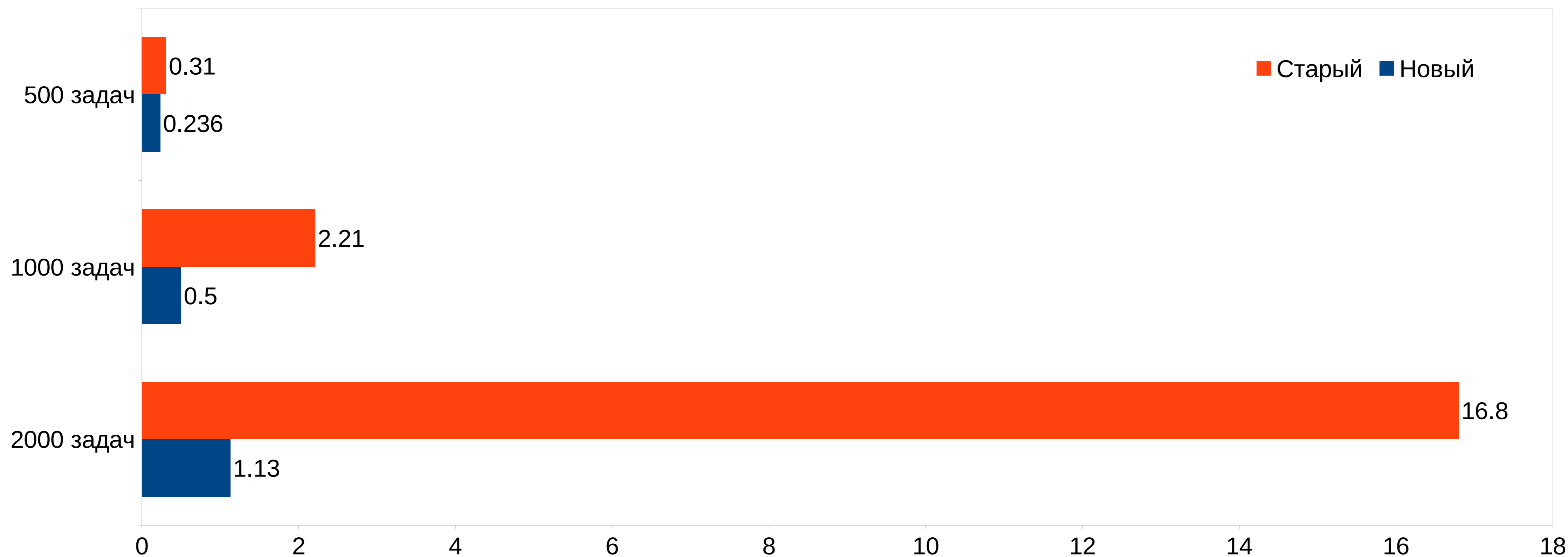
Рабочий ноутбук с 8 физическим ядрами

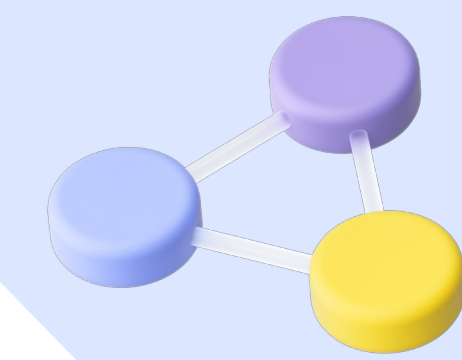




Оценка времени получения сессии из пула

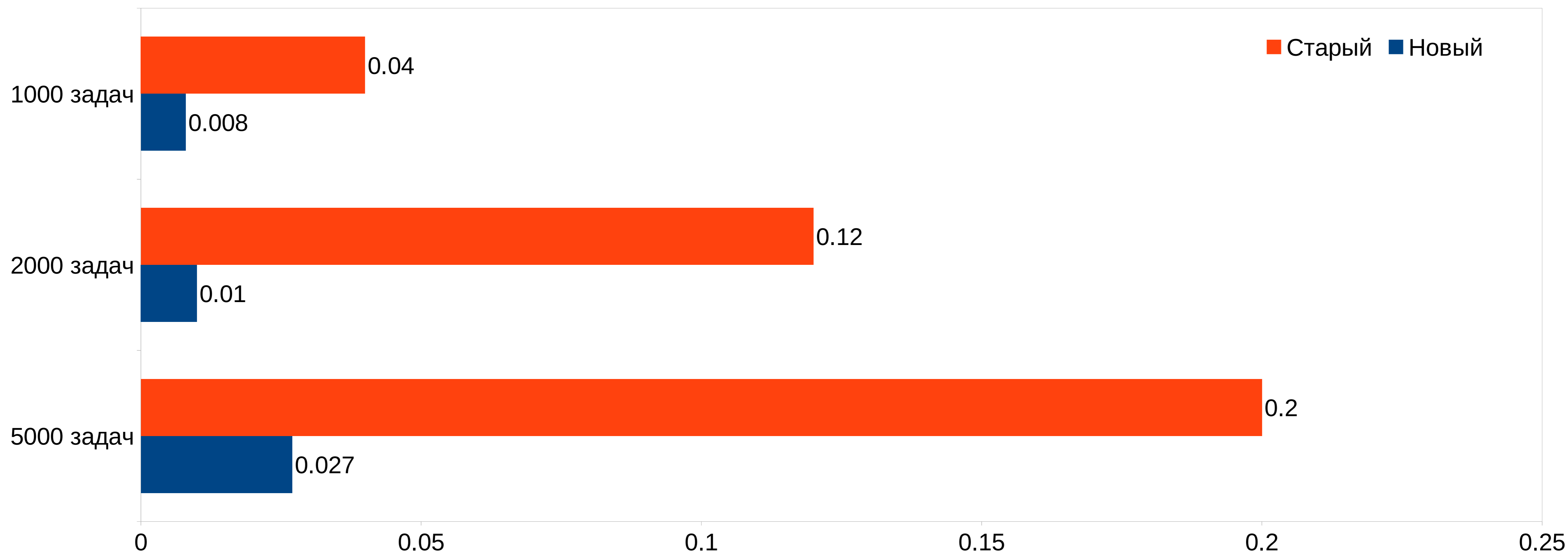
Небольшая виртуальная машина с 2 ядрами





Оценка времени получения сессии из пула

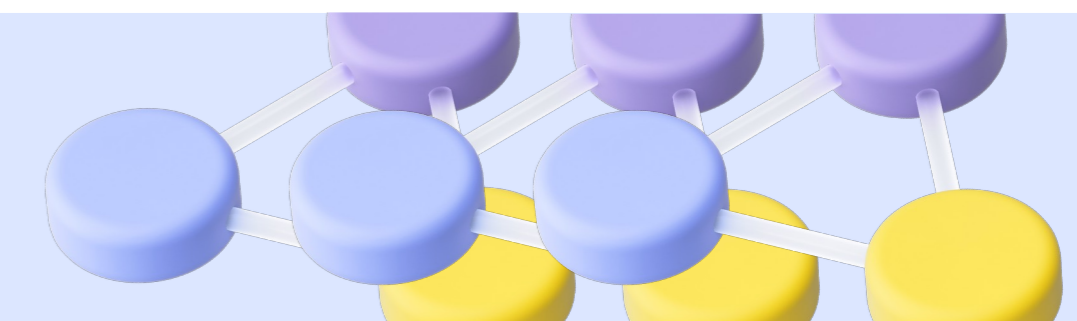
Большая виртуальная машине с 32 ядрами



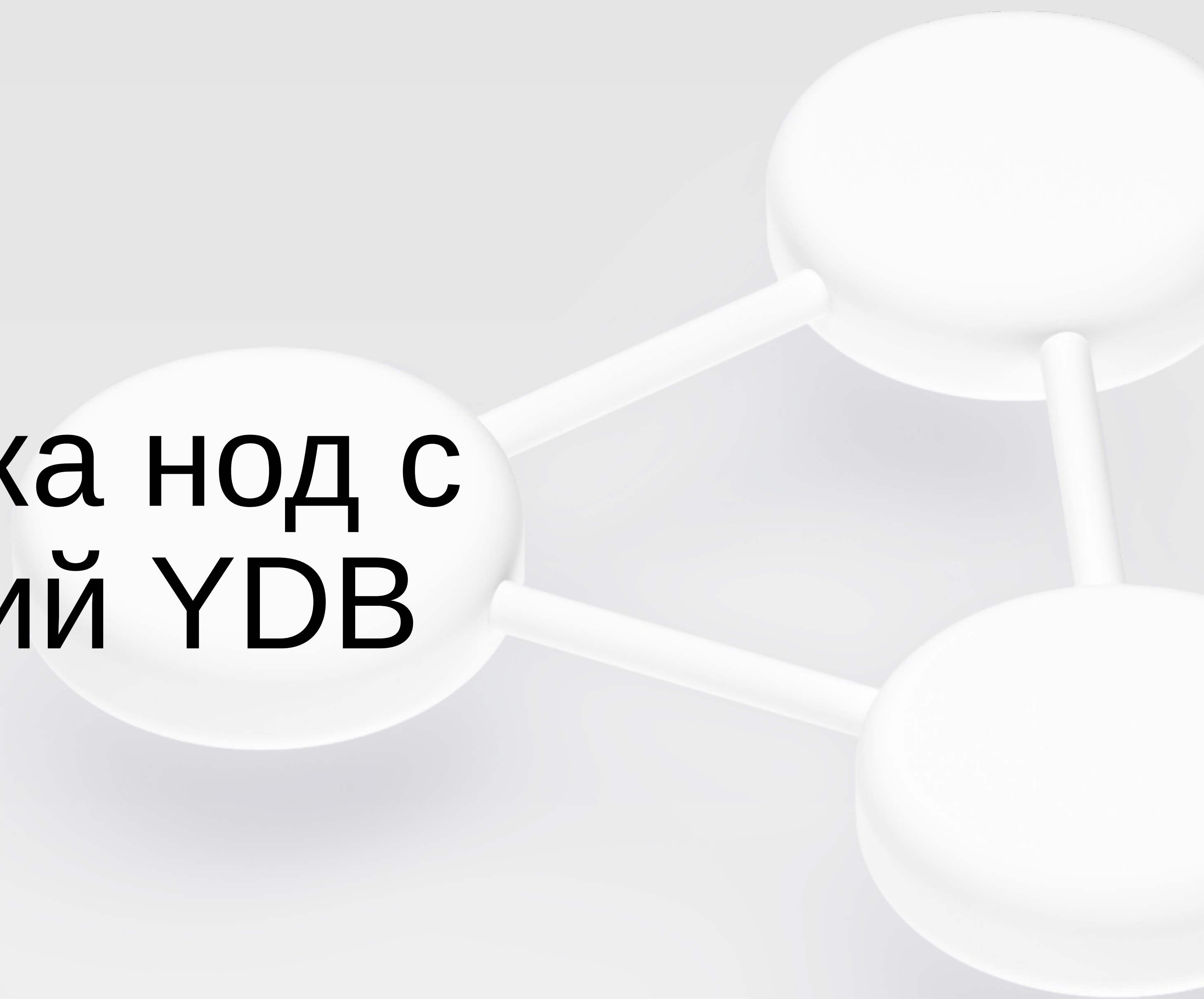
Пул сессий в Java SDK

Итоги

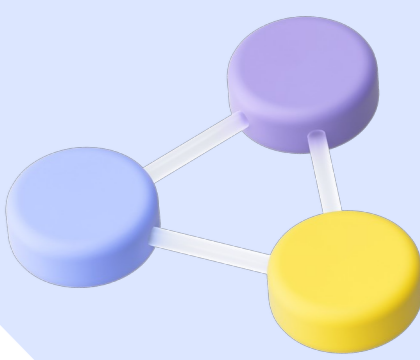
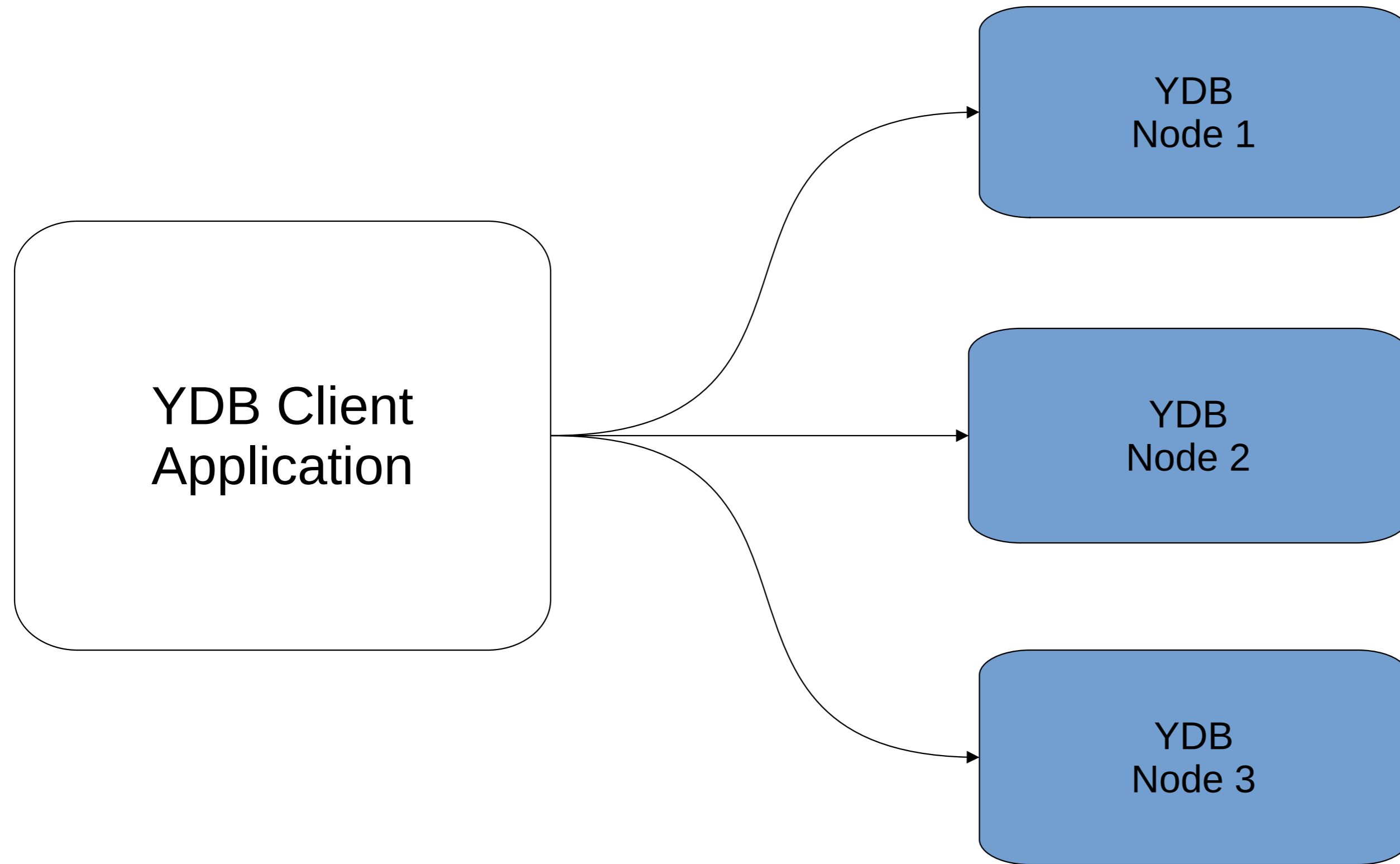
- Даже единственная блокировка в коде может вносить значительные задержки в работу многопоточного кода
- Заметный эффект от этого наступает только при большом количестве обращений к пулу — и это эффект может достигать десятков процентов
- На небольшом количестве задач — эффект не ощутим



Балансировка нод с учетом сессий YDB



Случайный выбор ноды



Балансировка запросов

Наивная реализация

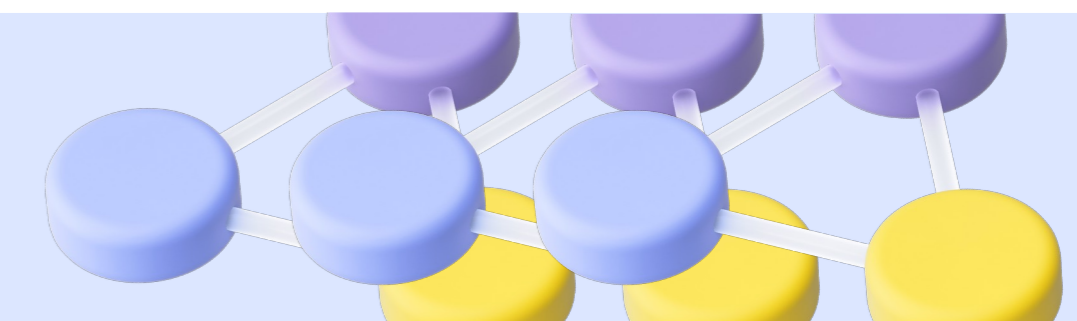
- Основана на RoundRobinLoadBalancer
- Каждый запрос отправляет на случайную ноду

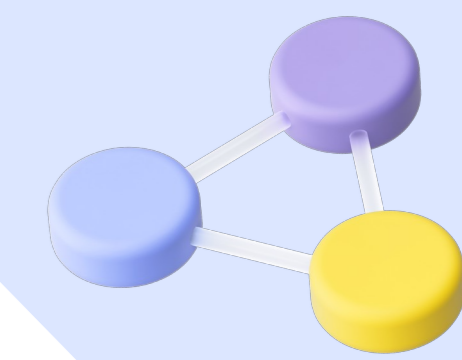
```
static final class RandomReadyPicker extends
    SubchannelPicker {

    private final List<Subchannel> list;
    ...

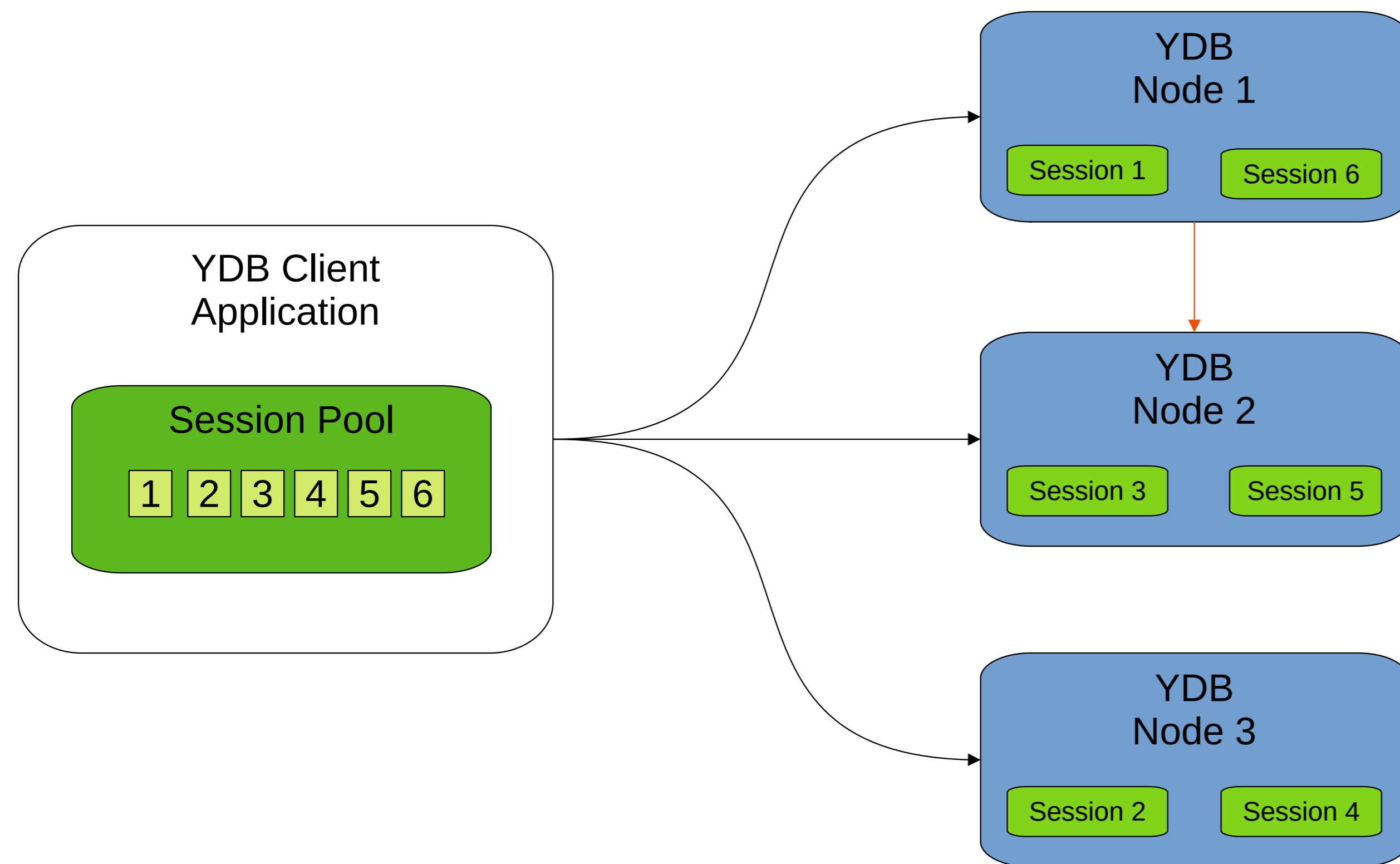
    private Subchannel nextSubchannel() {
        int nextIdx = ThreadLocalRandom.current()
            .nextInt(list.size());
        return list.get(nextIdx);
    }
}
```

<https://github.com/yandex-cloud/ydb-java-sdk/blob/master/core/src/main/java/com/yandex/ydb/core/grpc/impl/grpc/YdbLoadBalancer.java>





Случайный выбор ноды и сессии



- Большая часть запросов направляется на конкретную сессию
- Запрос при этом может быть отправлен на ноду, которая эту сессию не содержит
- В таком случае нода перенаправит запрос на правильную ноду
- Это создает дополнительный проху запрос между нодами

Балансировка запросов — доработанное решение

Возможность указать предпочтительную ноду

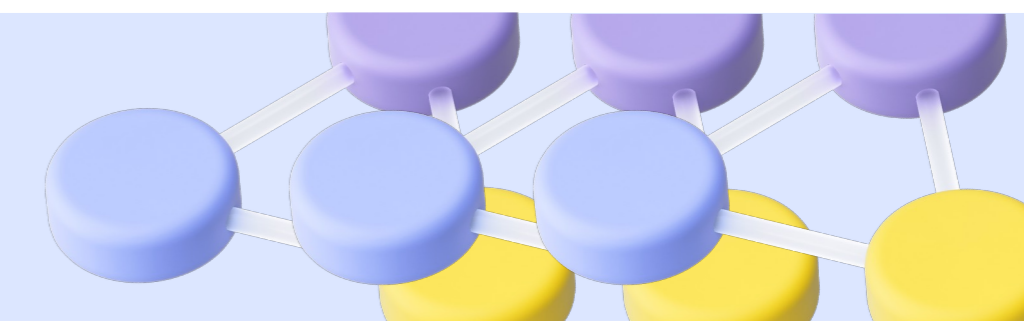
- Сохраняем информацию о ноде в отдельный HashMap
- Каждый запрос может указать предпочтительную ноду — и если она доступна, то он будет направлен на нее
- Иначе используется случайная нода из списка нод с наилучшим приоритетом

```
private List<Endpoint> records = new ArrayList<>();
private Map<Integer, Endpoint> byNodeID = new HashMap<>();

public Endpoint getEndpoint(Integer preferredNodeID) {
    if (preferredNodeID != null) {
        Endpoint known = byNodeID.get(preferredNodeID);
        if (known != null) {
            return known;
        }
    }

    int idx = ThreadLocalRandom.current()
        .nextInt(bestEndpointsCount);
    return records.get(idx);
}
```

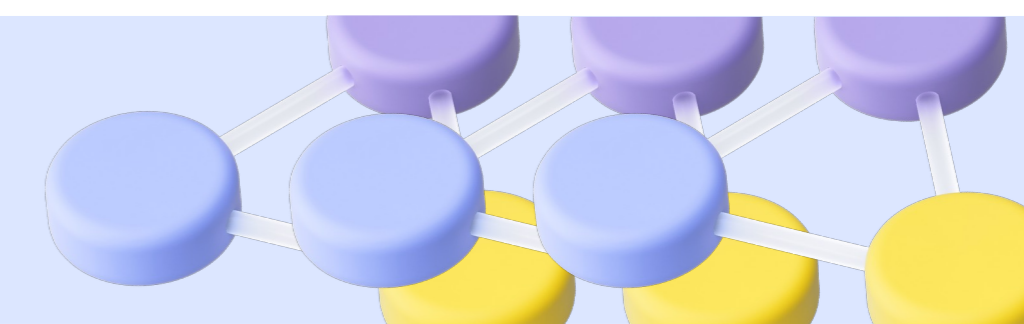
<https://github.com/ydb-platform/ydb-java-sdk/blob/develop/core/src/main/java/tech/ydb/core/impl/pool/EndpointPool.java>

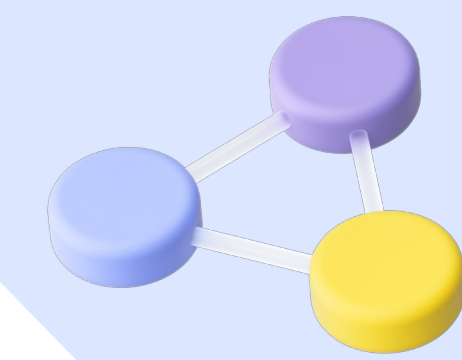


Тестирование привязки сессий к нодам

Измерение разницы в скорости работы двух реализаций

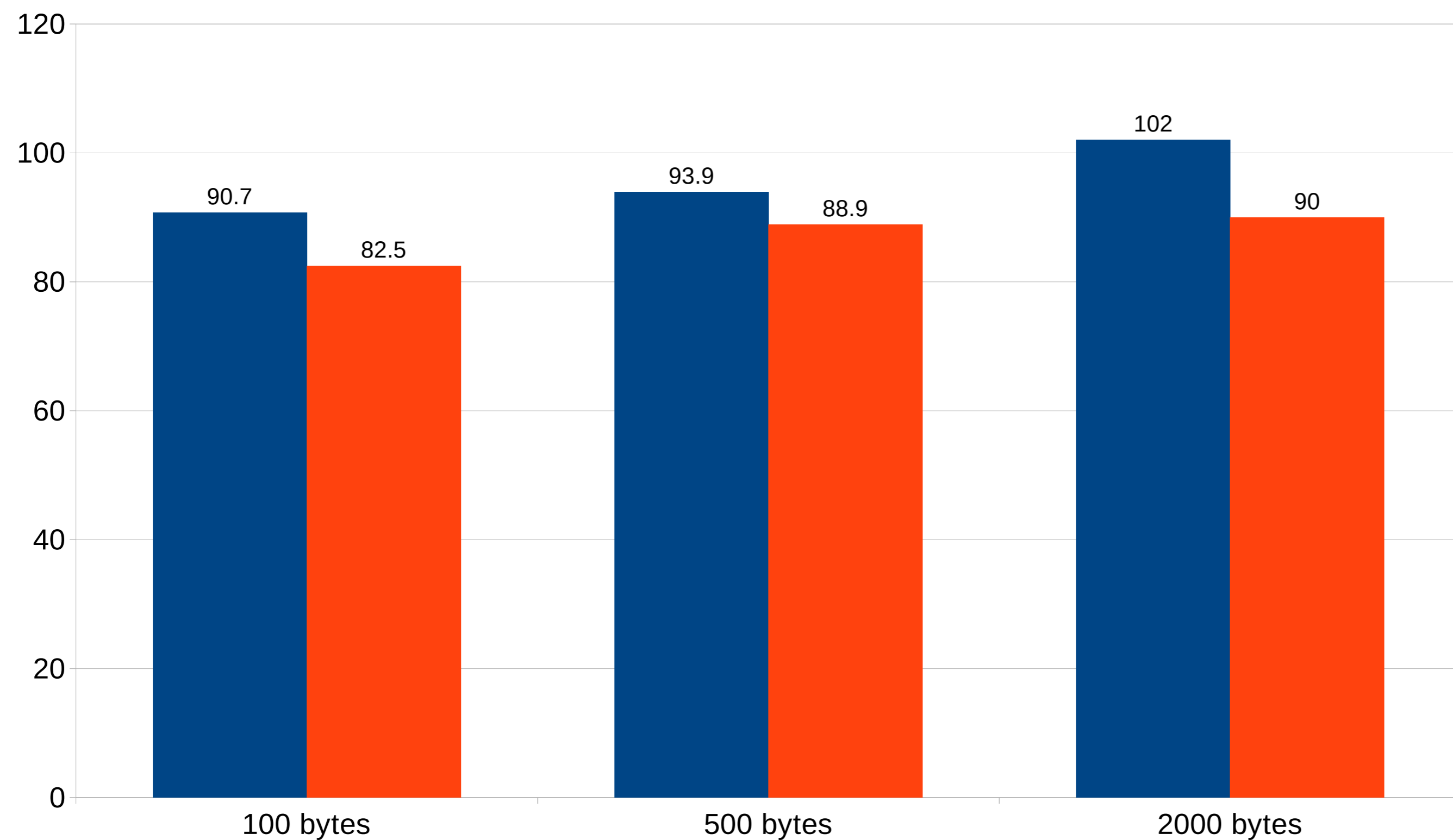
- Тестировать пул будем последовательным синхронным чтением записей различного размера
- Все чтения выполняются в нескольких параллельных потоках
- Измерять будем среднее время в миллисекундах, за которое выполнилось чтение — от отправки запроса до получения всех данных записи. Чем меньше время — тем лучше





Среднее время выполнения запроса

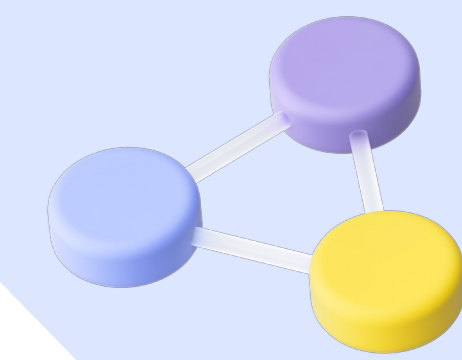
Запуск на личном ноутбуке



■ Старый вариант
■ Новый вариант

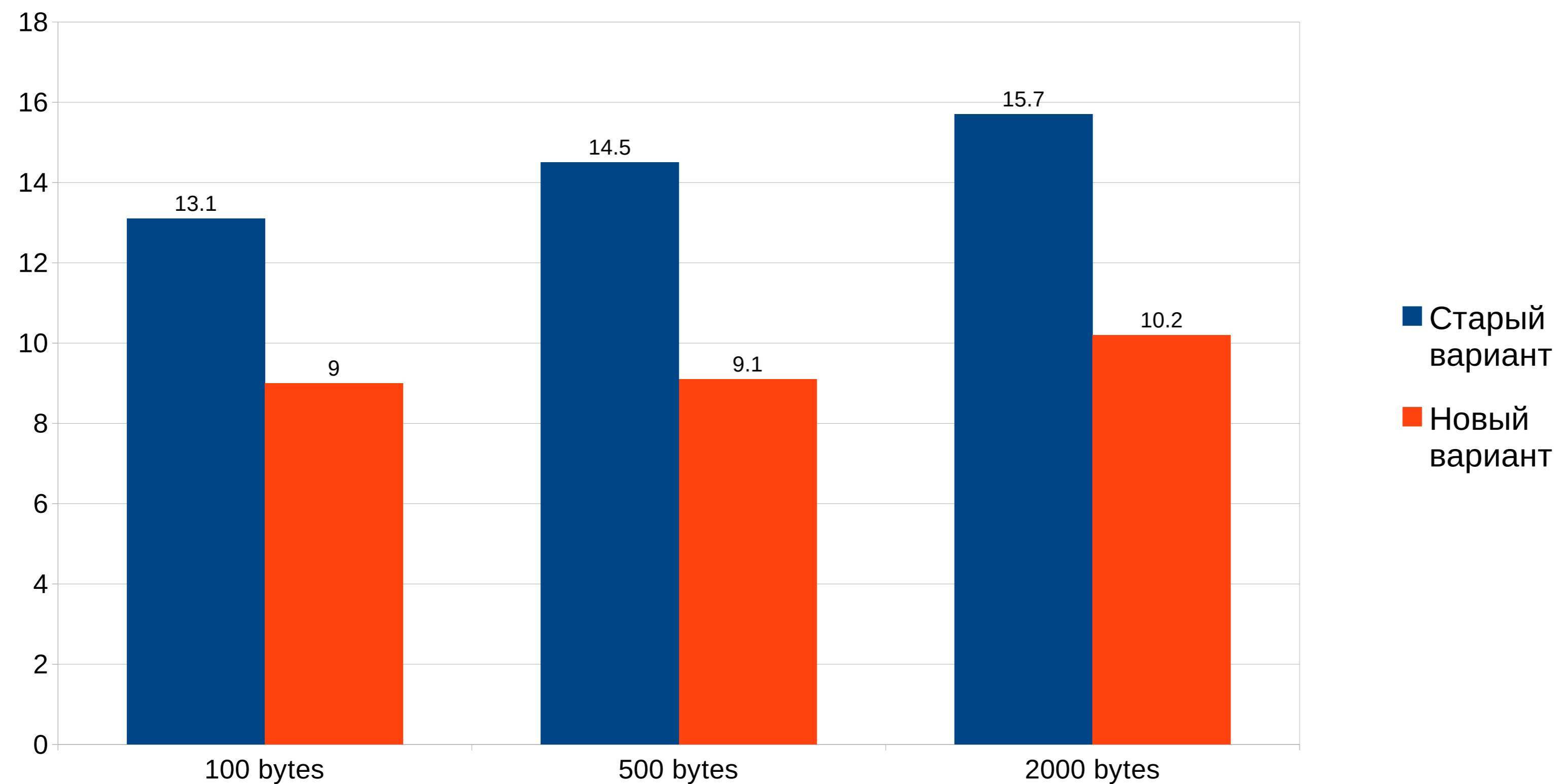
Итого в среднем

11.43%



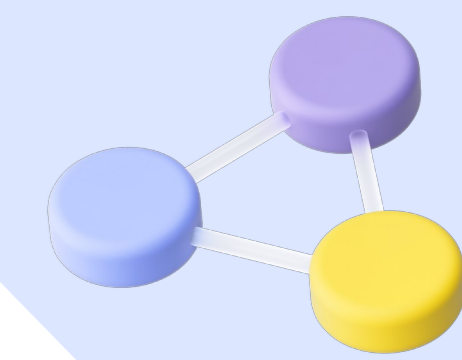
Среднее время выполнения запроса

Запуск на небольшой виртуальной машине



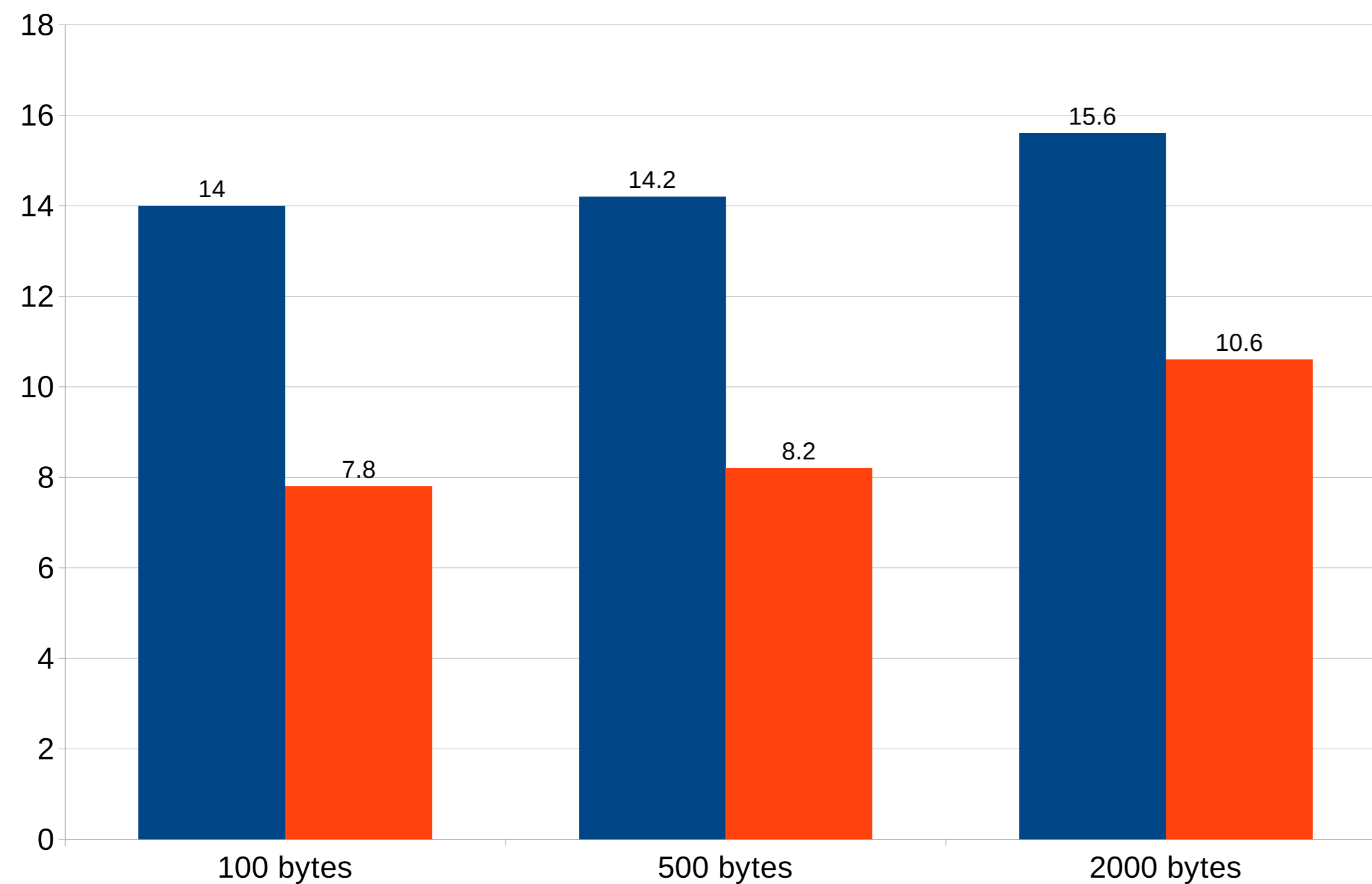
Итого в среднем

34.58%



Среднее время выполнения запроса

Запуск на большой виртуальной машине



■ Старый вариант
■ Новый вариант

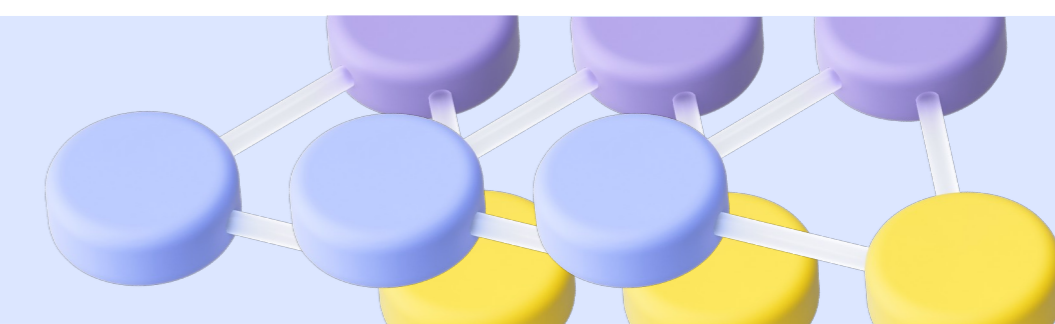
Итого в среднем

39.59%

Привязка сессий к нодам в Java SDK

Итоги

- Добрались уменьшения среднего времени выполнения запроса
- Эффект тем больше, чем больше число нод и чем ближе к серверу расположено клиентское приложение



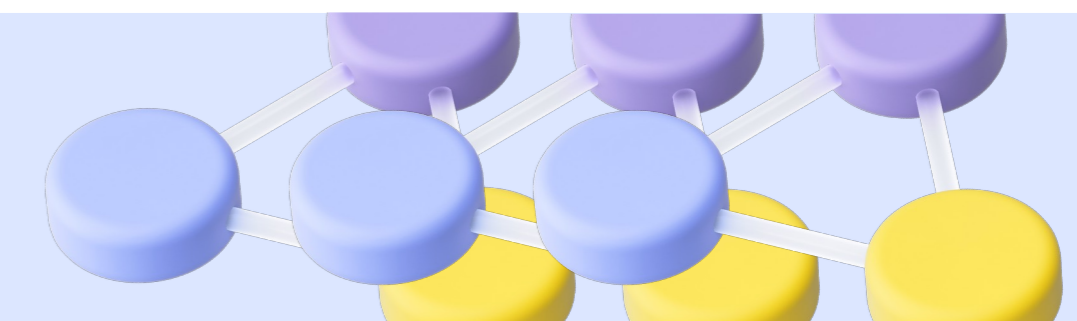
YDB Java SDK — другие изменения

Борьба с ошибкам

- Пессимизация нод
- Переработка различных методов SDK с целью повышения очевидности и простоты в использовании
- Удаление старых оптимизаций, оказавшихся тупиком в развитии
- Реализация graceful shutdown для сессий

Работа над производительностью

- Использование серверного кеша запросов по умолчанию
- Избавление от внутренних блокировок SDK



Спасибо за внимание

Ссылки

- Документация <https://ydb.tech/ru/docs/>
- Чат телеграмма https://t.me/ydb_ru
- github <https://github.com/ydb-platform>



Горшенин Александр,
Yandex, YDB, Старший разработчик
alexandr268@ydb.tech



Кулин Тимофей,
Yandex, YDB, Старший разработчик
rekby@yandex-team.ru

