PLATFORM V

Platform V User Sessions распределённое хранилище сессионных данных

Андрей Чернов
СберТех

О себе

Андрей Чернов


15 лет опыта разработки


Кандидатская диссертация
по индексам в БД


Java архитектор в СберТех


chernovaf@mail.ru


chernovaf

План

План

План

План

Решаемая задача

Хранение данных всех активных сессий СберБанк Онлайн в RAM.

Любой микросервис

может обращаться к данным сессии

Хранилище в backend

Данные могут быть персональными и конфиденциальными:

информация о клиенте, его счетах, картах и пр.

Решаемая задача

Хранение данных всех активных сессий СберБанк Онлайн в RAM.

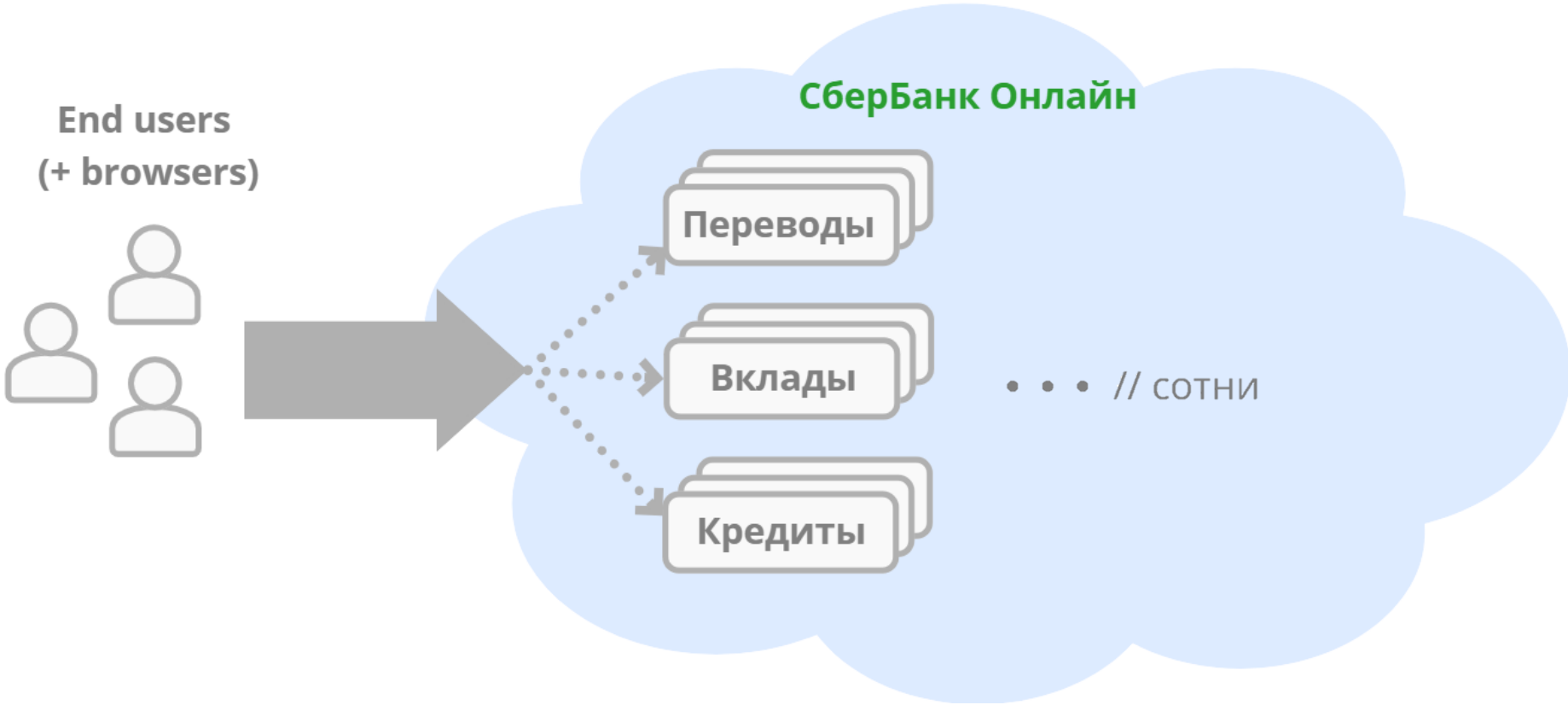
Любой микросервис

может обращаться к данным сессии

Хранилище в backend

Данные могут быть персональными и конфиденциальными:

информация о клиенте, его счетах, картах и пр.



Решаемая задача

Хранение данных всех активных сессий СберБанк Онлайн в RAM.

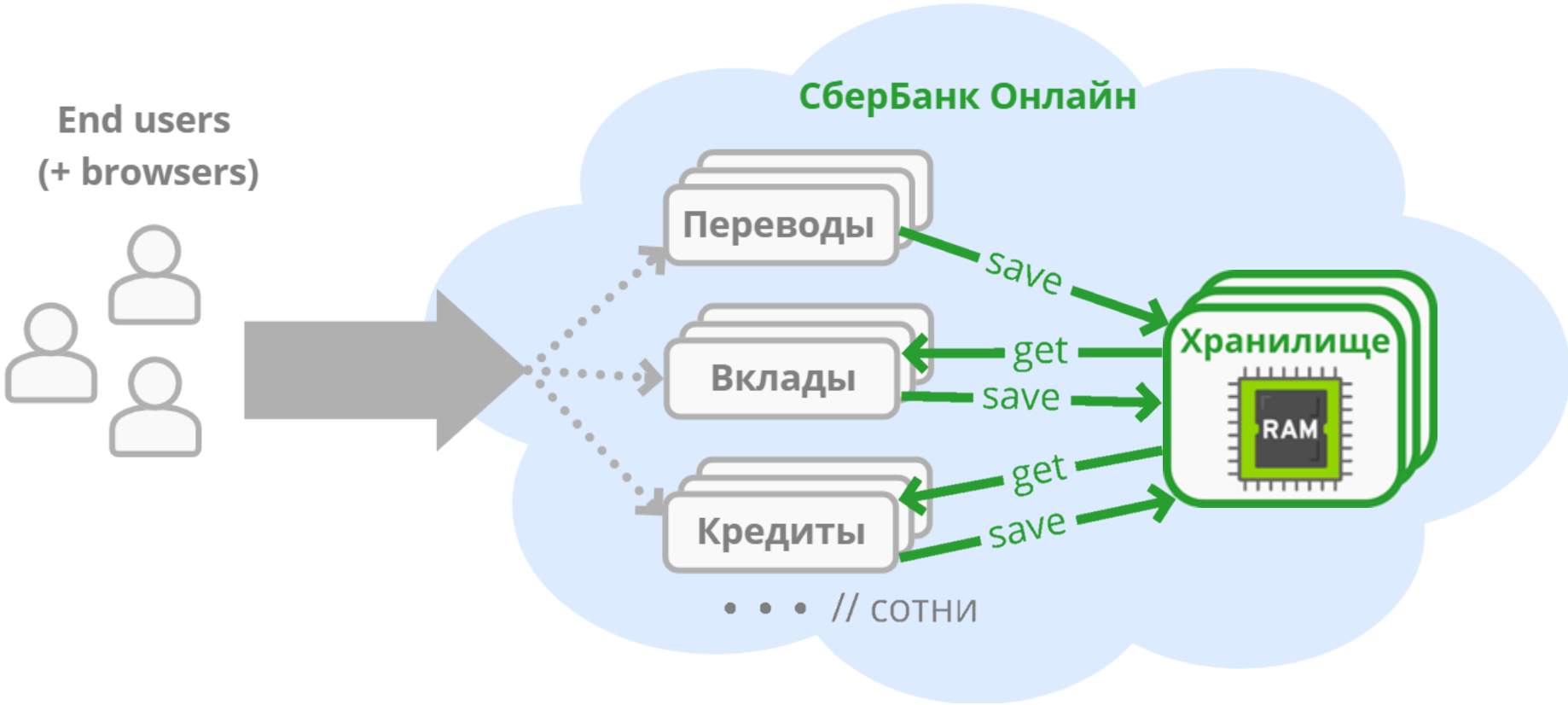
Любой микросервис

может обращаться к данным сессии

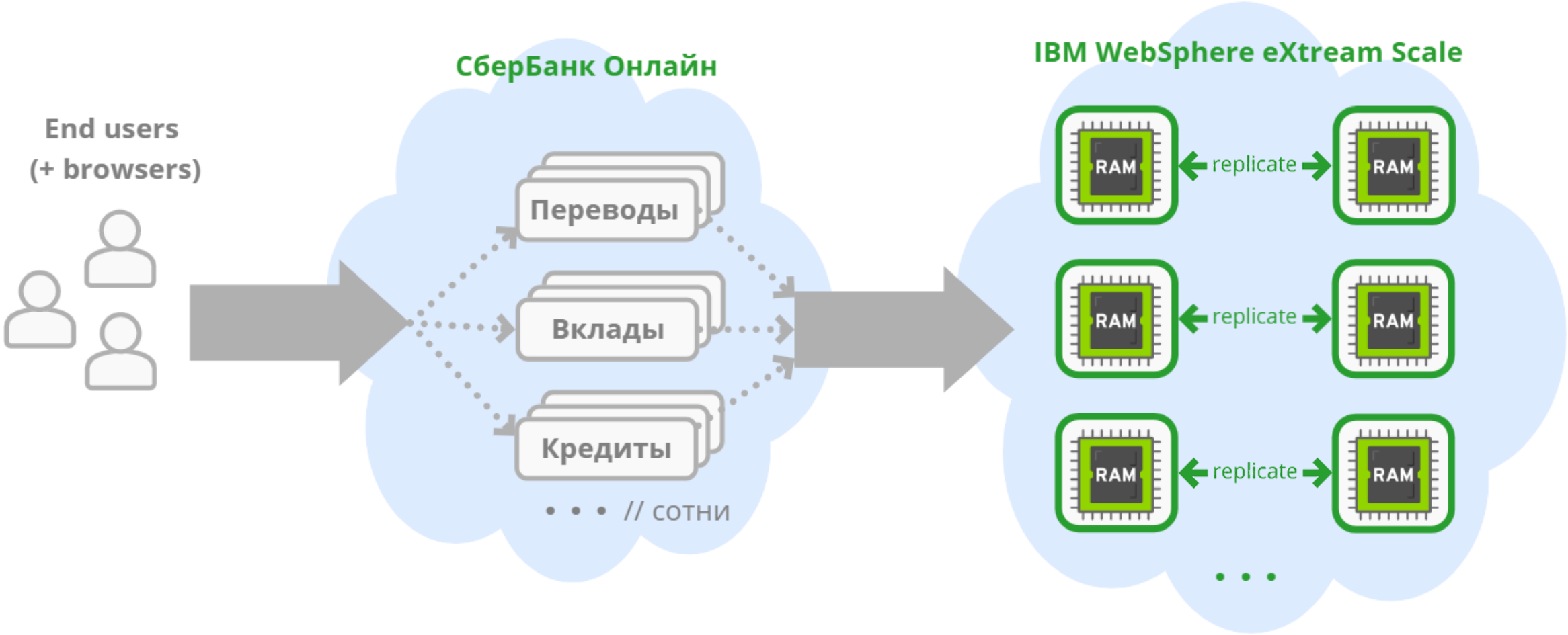
Хранилище в backend

Данные могут быть персональными и конфиденциальными:

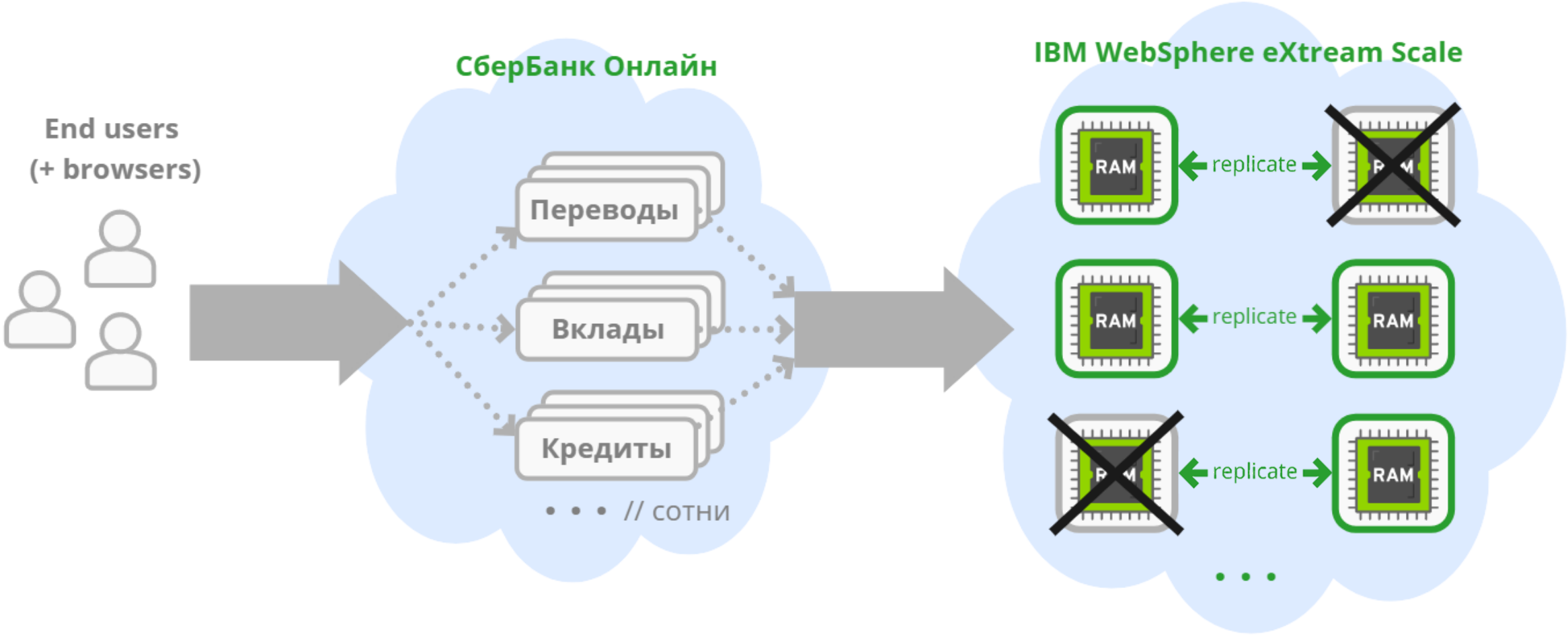
информация о клиенте, его счетах, картах и пр.



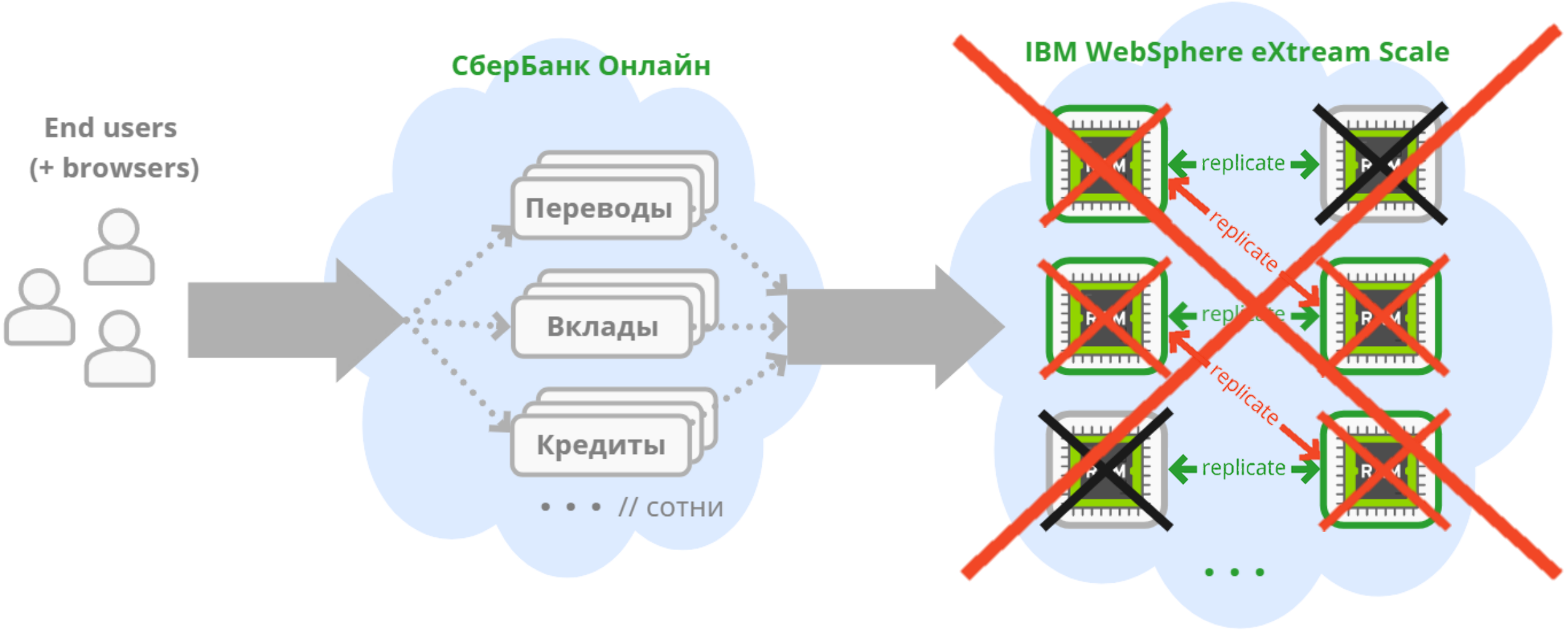
Предыдущее неудачное решение



Предыдущее неудачное решение



Предыдущее неудачное решение



Создание Platform V User Sessions

Продукт на замену IBM WebSphere eXtream Scale.

User Sessions – распределённое in-memory хранилище для сессий online-клиентов:

https://platformv.sber.ru/products/platform-v-user-sessions

Создание Platform V User Sessions

Продукт на замену IBM WebSphere eXtream Scale.

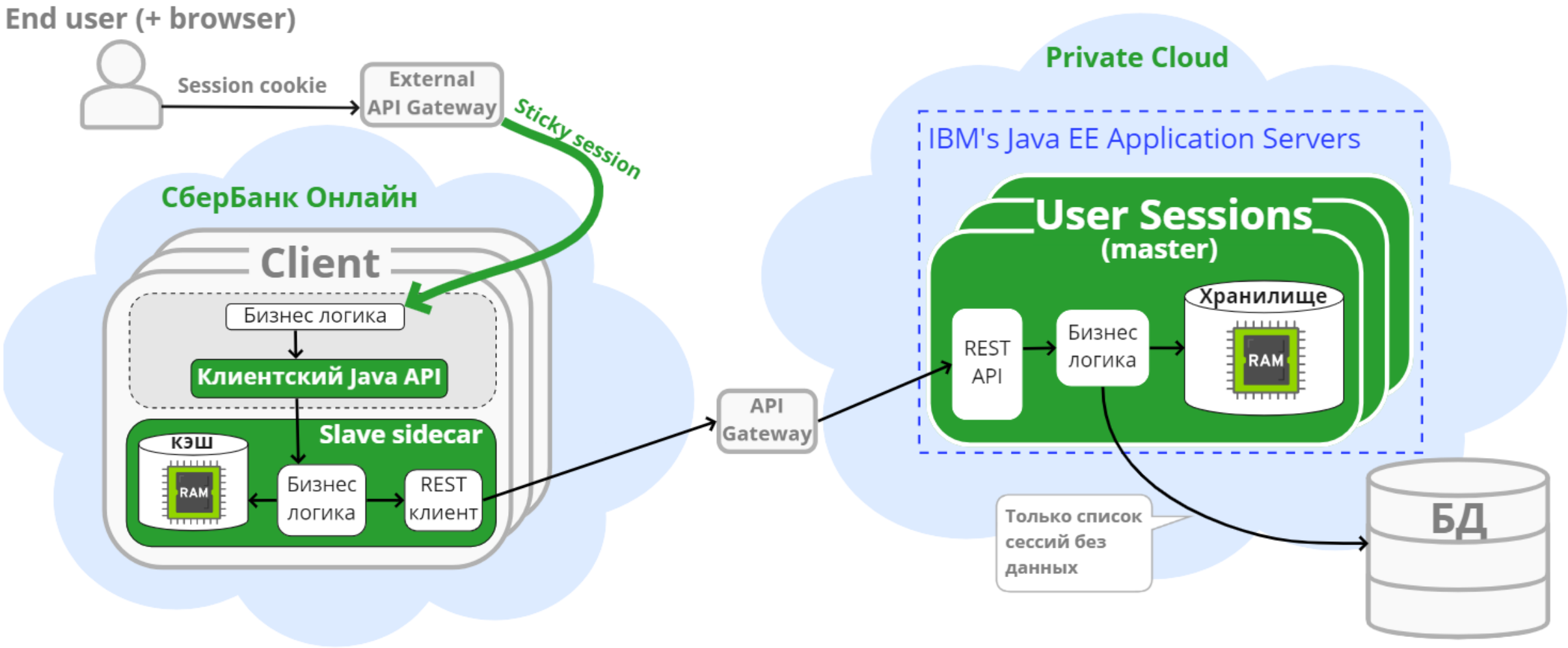
User Sessions – распределённое in-memory хранилище для сессий online-клиентов:

https://platformv.sber.ru/products/platform-v-user-sessions

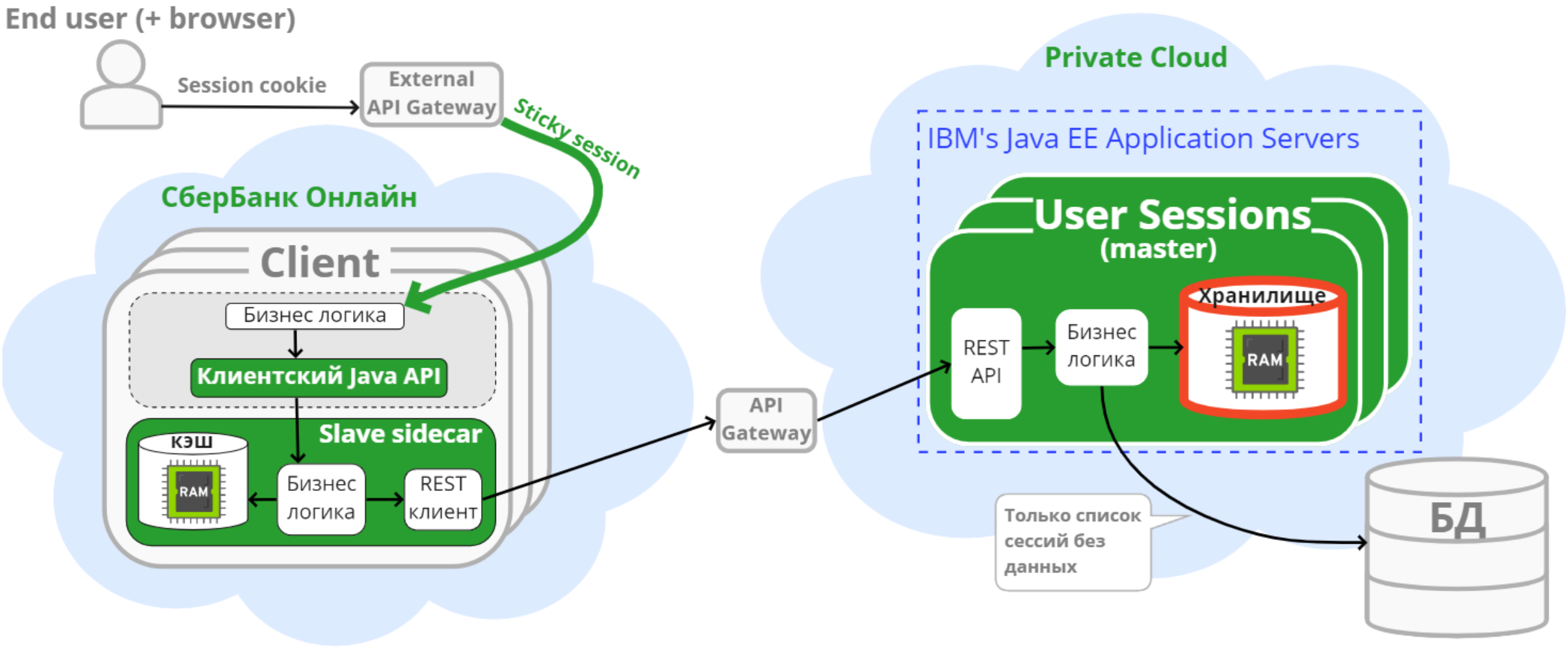
Platform V – цифровая облачная платформа Сбера для разработки бизнес-решений.

https://platformv.sber.ru/

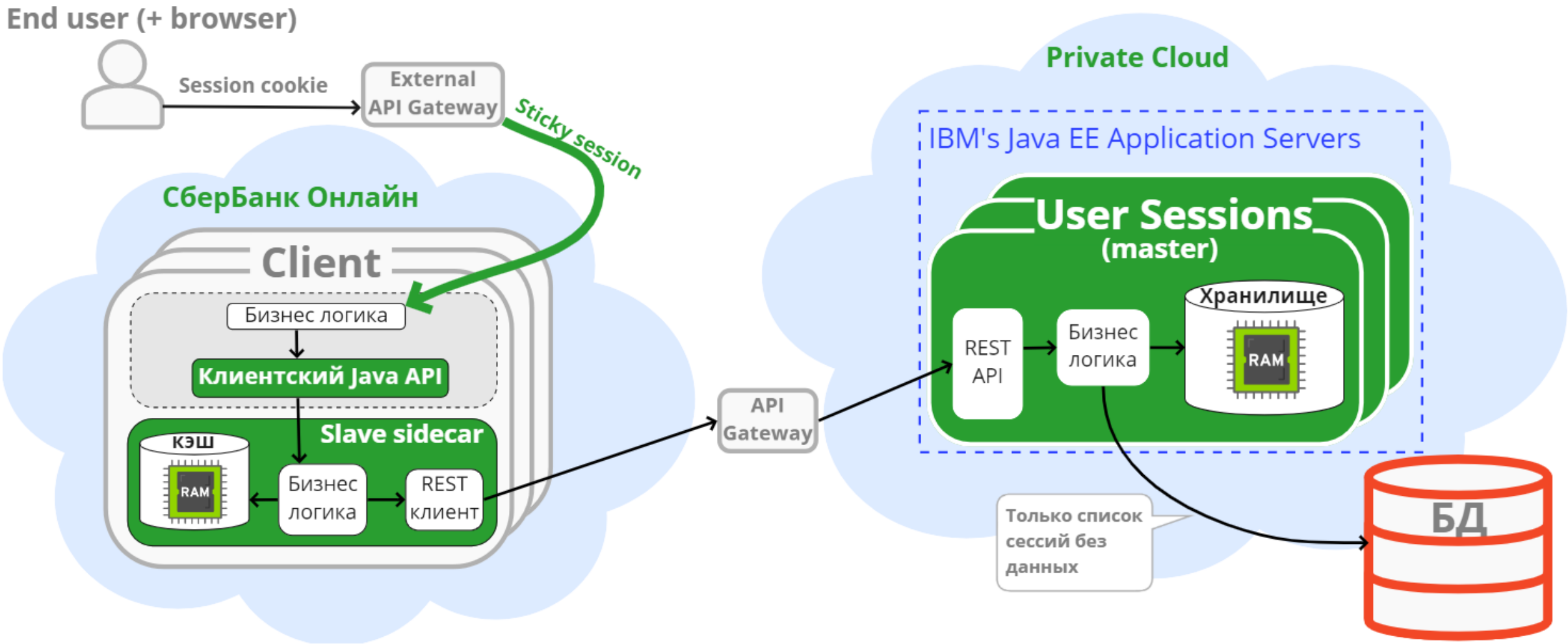
Архитектура Platform V User Sessions



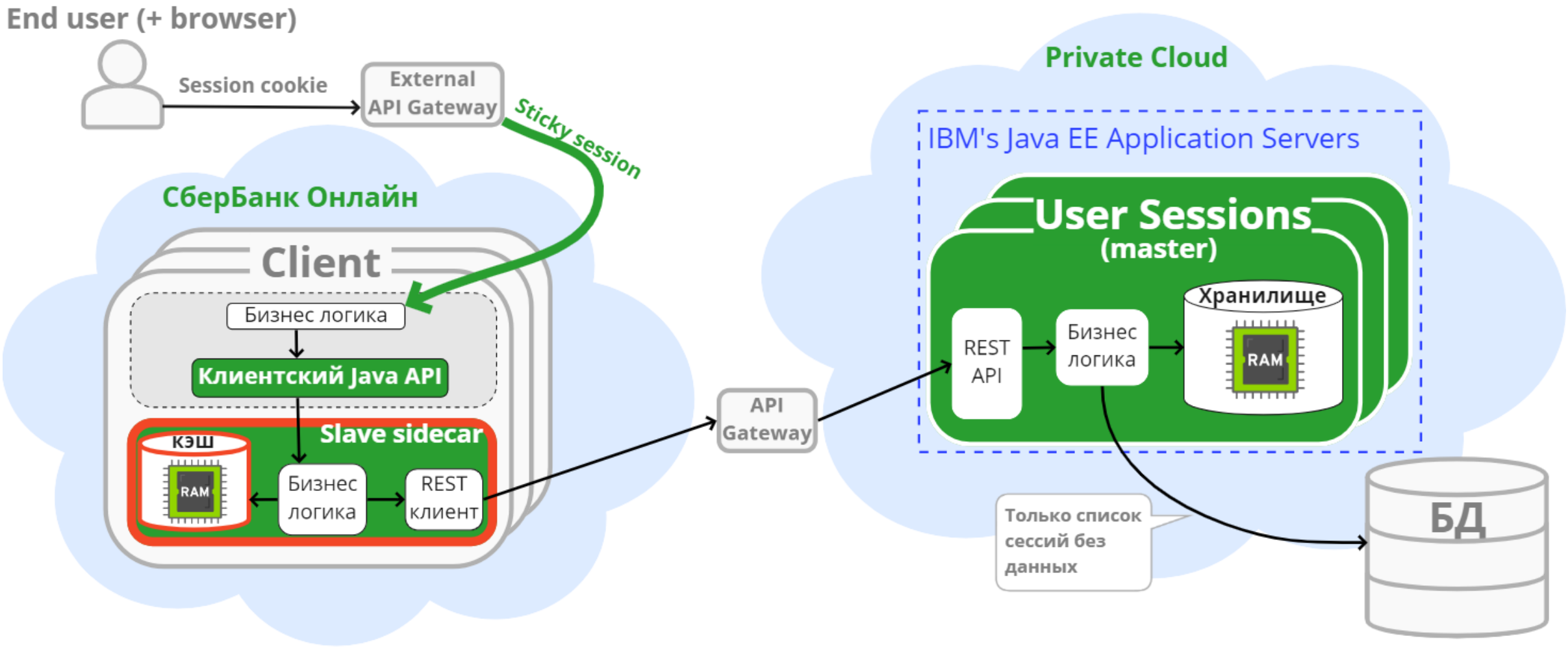
Архитектура Platform V User Sessions



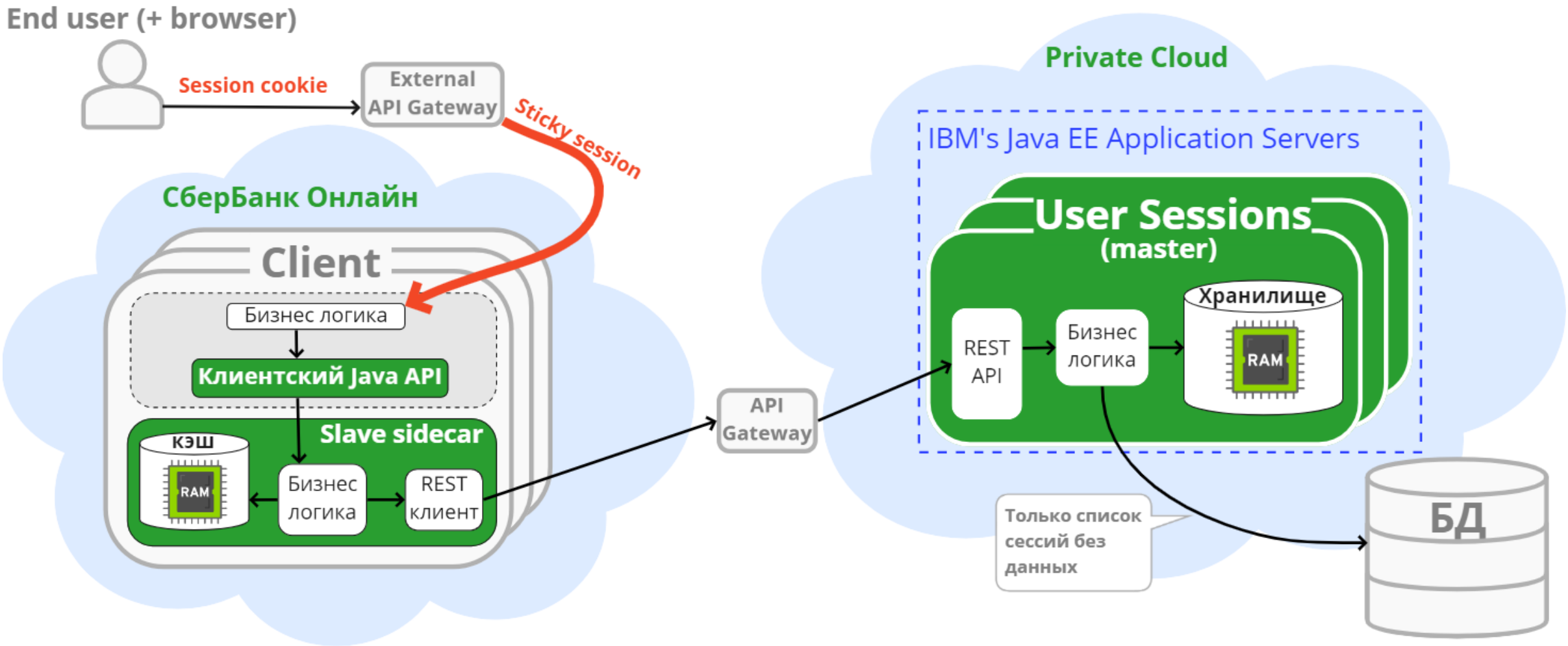
Архитектура Platform V User Sessions



Архитектура Platform V User Sessions



Архитектура Platform V User Sessions



Архитектура Platform V User Sessions

Первый успех

1

Агрессивное кэширование данных на стороне клиента.

2

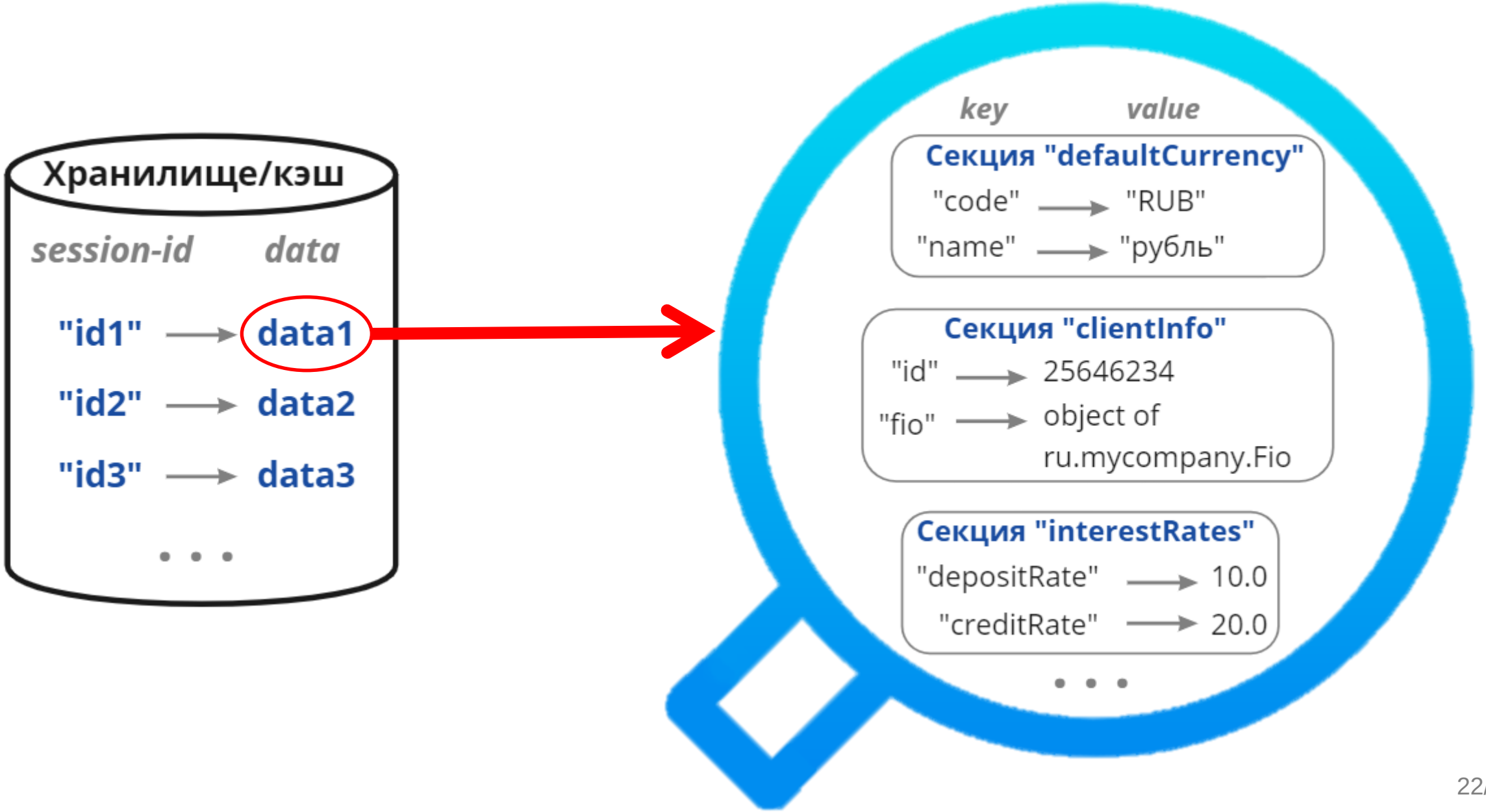
Липкая сессия (sticky session) к узлам клиента.

Результат

Более 30% запросов не обращаются к сети.

План

Сессии и секции данных



Сохранение любых объектов потребителя

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);


SdsSessionSection mySection =
    session.getSection("mySection");
mySection.setAttribute("int", 123);
mySection.setAttribute("double", 456D);
mySection.setAttribute("String", "String value");
mySection.setAttribute("Object",
    new MyClass(Map.of(123,
                       List.of("one", "two"))));


sds.saveSessionSections(session, ...);
```

```
private static class MyClass {

  public final Map<Integer, List<String>> field;

  public MyClass(Map<Integer, List<String>> field){
    this.field = field;
  }
}
```

Сериализация в JSON объектов потребителя

Тюнинг Jackson-a

```
ObjectMapper mapper = new ObjectMapper()
      .enable(ACCEPT_SINGLE_VALUE_AS_ARRAY, ACCEPT_EMPTY_STRING_AS_NULL_OBJECT,
              ACCEPT_EMPTY_ARRAY_AS_NULL_OBJECT, READ_UNKNOWN_ENUM_VALUES_AS_NULL,
              UNWRAP_SINGLE_VALUE_ARRAYS)
      .disable(FAIL_ON_INVALID_SUBTYPE, FAIL_ON_NULL_FOR_PRIMITIVES,
               FAIL_ON_IGNORED_PROPERTIES, FAIL_ON_UNKNOWN_PROPERTIES,
               FAIL_ON_NUMBERS_FOR_ENUMS, FAIL_ON_UNRESOLVED_OBJECT_IDS,
               WRAP_EXCEPTIONS)
      .enable(MapperFeature.PROPAGATE_TRANSIENT_MARKER)
      .enable(ALLOW_SINGLE_QUOTES)
      .disable(FAIL_ON_EMPTY_BEANS)
      .setDateFormat(new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSZZ"))
      .configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS, false)
      .setVisibility(ALL, NONE)
      .setVisibility(FIELD, ANY)
      .enableDefaultTyping(NON_FINAL);
```

Сериализованные в JSON объекты потребителя

Тюнинг Jackson-a

```
var object =
    MyClass(Map.of(123,
                   List.of("one", "two")));

String json =
    objectMapper.writeValueAsString(object);
```

```
[
  "sber.platform.joker2022.MyClass",
  {
    "field":
    [
      "java.util.ImmutableCollections$MapN",
      {
        "123":
        [
          "java.util.ImmutableCollections$List12",
          ["one", "two"]
        ]
      }
    ]
  }
]
```

Передача секций данных по сети

Обработка данных

- Секция – набор key-value пар с объектами потребителя

Передача секций данных по сети

Обработка данных

- Секция – набор key-value пар с объектами потребителя
- Сериализуется в тюненный JSON
- Сжимается Zip-ом

Передача секций данных по сети

Обработка данных

- Секция – набор key-value пар с объектами потребителя
- Сериализуется в тюненный JSON
- Сжимается Zip-ом
- Несколько секций в DTO

DTO для передачи

```
public class <DTO> {
    public final String sourceVersion;
    public final String id;
    public final Map<String, Serializable> attributes;
    public final List<SectionDto> sectionList;
    ...
}
public class SectionDto {
    public final String sectionName;
    public final Integer characteristics;
    public final Long version;
    public final byte[] serializedBinData;
    ...
}
```

Передача секций данных по сети

Обработка данных

- Секция – набор key-value пар с объектами потребителя
- Сериализуется в тюненный JSON
- Сжимается Zip-ом
- Несколько секций в DTO
- Сериализуется в обычный JSON
- Передаётся по сети

DTO для передачи

```
public class <DTO> {
    public final String sourceVersion;
    public final String id;
    public final Map<String, Serializable> attributes;
    public final List<SectionDto> sectionList;
    ...
}
public class SectionDto {
    public final String sectionName;
    public final Integer characteristics;
    public final Long version;
    public final byte[] serializedBinData;
    ...
}
```

Два уровня сериализации данных в User Sessions

1 Бизнесовый

Секции с объектами потребителя.
Тюненый JSON.

2 Транспортный

DTO с несколькими секциями.
Обычный JSON.

План

~~1. Platform V User Sessions. Начало пути~~

~~1.1. Работа с данными и их передача по сети~~

2. Развитие User Sessions для выдерживания highload

2.1. Сериализация данных в бинарный формат One Nio

2.2. Сохранение данных «дельтами»

2.3. Чтение данных пачками

3. Как User Sessions достигает high availability

3.1. Load balancing & true sticky

3.2. Асинхронная работа с базой данных

3.3. Хранилище не в контейнере

3.4. No vendor lock

3.5. Репликация данных (в процессе реализации)

4. Дальнейшее развитие User Sessions

4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД

4.2. Авторизация доступа к данным через Open Policy Agent

4.3. «Не клиентские» сессии

Нагрузка на User Sessions в СберБанк Онлайн

3 года в production

> 110 000

Входов пользователей в минуту

~ 5 минут

Период активности каждой сессии

> 550 000

Сессий в моменте являются активными

> 120 000

Запросов в секунду.

В среднем 67 запросов за сессию.

70-80 КБ

Средний размер сессии (все секции)

3 МБ

Максимальный размер сессии (все секции)

Начало противостояния с highload

> 100

микросервисов-потребителей у User Sessions в СберБанк Онлайн

До 10 мс

доходило время отклика User Sessions

Тормоза при сериализации в JSON

Что выяснили?

- Тормозила сериализация в JSON Jackson-ом
- Суммарно **> 50%** времени работы сервиса
- Основные тормоза на транспортном уровне

Тормоза при сериализации в JSON

Что выяснили?

- Тормозила сериализация в JSON Jackson-ом
- Суммарно **> 50%** времени работы сервиса
- Основные тормоза на транспортном уровне

Сериализуемая DTO

```
public class <DTO> {
    ...
    public final List<SectionDto> sectionList;
    ...
}
public class SectionDto {
    public final String sectionName;
    public final Integer characteristics;
    public final Long version;
    public final byte[] serializedBinData;
    ...
}
```

Тормоза при сериализации в JSON

Что выяснили?

- Тормозила сериализация в JSON Jackson-ом
- Суммарно **> 50%** времени работы сервиса
- Основные тормоза на транспортном уровне
- Сериализовывать byte[] в JSON – **нетиповой case!**

  Требуется преобразование.

Сериализуемая DTO

```
public class <DTO> {
    ...
    public final List<SectionDto> sectionList;
    ...
}
public class SectionDto {
    public final String sectionName;
    public final Integer characteristics;
    public final Long version;
    public final byte[] serializedBinData;
    ...
}
```

Тормоза при сериализации в JSON

Base64 кодирование byte[]

Сериализованная DTO

```
{
  ...
  "sectionList": [
    {
      "sectionName": "COMMON",
      "characteristics": 0,
      "version": 0,
      "serializedBinData": "eJyLVspKLEvUKy3JzNHzSCzO8E0sUNKpVnIMCQ..."
      ...
    }
  ]
  ...
}
```

Проблемы

- Медленная сериализация
  1 КБ в 2,5 раза медленнее,
  чем строка 1024 символа

Тормоза при сериализации в JSON

Base64 кодирование byte[]

Сериализованная DTO

```
{
  ...
  "sectionList": [
    {
      "sectionName": "COMMON",
      "characteristics": 0,
      "version": 0,
      "serializedBinData": "eJyLVspKLEvUKy3JzNHzSCzO8E0sUNKpVnIMCQ..."
      ...
    }
  ]
  ...
}
```

Проблемы

- Медленная сериализация
  1 КБ в 2,5 раза медленнее, чем строка 1024 символа
- Лишний трафик
  На 33.3% больше, чем размер самого byte[]

Тормоза при сериализации в JSON

String вместо byte[] – немного быстрее, чем base64

Сериализуемая DTO

```
// serializedData = new String(serializedBinData,
//                             StandardCharsets.ISO_8859_1)

public class <DTO> {
    ...
    public final List<SectionDto> sectionList;
    ...
}
public class SectionDto {
    public final String sectionName;
    public final Integer characteristics;
    public final Long version;
    public final String serializedData;
    ...
}
```

Проблемы

- Медленная сериализация
  1 КБ в 2,3 раза медленнее,
  чем строка 1024 символа

Тормоза при сериализации в JSON

String вместо byte[] – немного быстрее, чем base64

Сериализуемая DTO

```
// serializedData = new String(serializedBinData,
//                             StandardCharsets.ISO_8859_1)

public class <DTO> {
    ...
    public final List<SectionDto> sectionList;
    ...
}
public class SectionDto {
    public final String sectionName;
    public final Integer characteristics;
    public final Long version;
    public final String serializedData;
    ...
}
```

Проблемы

- Медленная сериализация
  1 КБ в 2,3 раза медленнее,
  чем строка 1024 символа
- Лишний трафик
  Не менее, чем при
  кодировании byte[] в base64

Тормоза при сериализации в JSON

String вместо byte[] – экранирование всё испортило

Экранированию подлежат

- 8 специальных символов

```
"    quotation mark  U+0022
\    reverse solidus U+005C
/    solidus         U+002F
b    backspace       U+0008
f    form feed       U+000C
n    line feed       U+000A
r    carriage return U+000D
t    tab             U+0009
```

- 32 служебных символа от U+0000 до U+001F

RFC 4627 on Json: https://www.ietf.org/rfc/rfc4627.txt

Тормоза при сериализации в JSON

String вместо byte[] – экранирование всё испортило

Сериализованная DTO

```
{
  ...
  "sectionList": [
    {
      "sectionName": "COMMON",
      "characteristics": 0,
      "version": 0,
      "serializedData": "Bae n\fQa\f?rA®AA??~n?.np\t ??\u0000§o y¬iA\u0002?aE??,#E?..."
      ...
    }
  ]
  ...
}
```

План

~~1. Platform V User Sessions. Начало пути~~

~~1.1. Работа с данными и их передача по сети~~

2. Развитие User Sessions для выдерживания highload

2.1. Сериализация данных в бинарный формат One Nio

2.2. Сохранение данных «дельтами»

2.3. Чтение данных пачками

3. Как User Sessions достигает high availability

3.1. Load balancing & true sticky

3.2. Асинхронная работа с базой данных

3.3. Хранилище не в контейнере

3.4. No vendor lock

3.5. Репликация данных (в процессе реализации)

4. Дальнейшее развитие User Sessions

4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД

4.2. Авторизация доступа к данным через Open Policy Agent

4.3. «Не клиентские» сессии

Требования к новому сериализатору

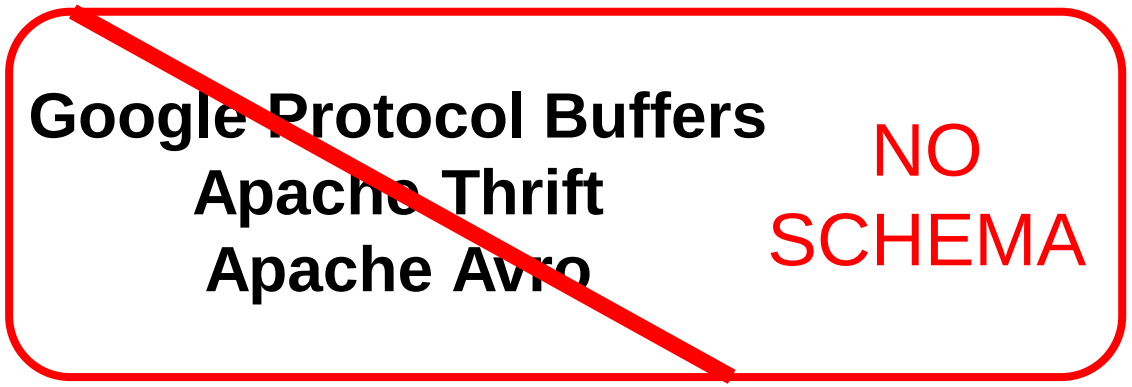
Бизнесовый уровень

- Не должно быть требований к структуре сериализуемых объектов

Требования к новому сериализатору

Бизнесовый уровень

- Не должно быть требований к структуре сериализуемых объектов



Требования к новому сериализатору

Бизнесовый уровень

- Не должно быть требований к структуре сериализуемых объектов

~~**Google Protocol Buffers**~~
~~**Apache Thrift**~~
~~**Apache Avro**~~

NO SCHEMA

Транспортный уровень

- Эффективная сериализация byte[]
- Минимальный размер результата

Требования к новому сериализатору

Бизнесовый уровень

- Не должно быть требований к структуре сериализуемых объектов

~~**Google Protocol Buffers**~~
~~**Apache Thrift**~~
~~**Apache Avro**~~

NO SCHEMA

Транспортный уровень

- Эффективная сериализация byte[]
- Минимальный размер результата

Только сериализаторы в бинарный формат

Сериализаторы-участники соревнования

1. Java standard
2. Jackson Smile
3. Bson4Jackson
4. BSON MongoDb
5. Kryo
6. FST
7. One Nio

Сериализаторы-участники соревнования

0.Jackson JSON
1.Java standard
2.Jackson Smile
3.Bson4Jackson
4.BSON MongoDb
5.Kryo
6.FST
7.One Nio

Гонки между сериализаторами-участниками

Сериализация и десериализация на скорость

Собственный benchmark

- DTO специфичные – не стали доверять другим
- На Java Microbenchmark Harness (JMH)
- Выложен на GitHub: https://github.com/chernov-af/serializers-benchmark

Гонки между сериализаторами-участниками

Сериализация и десериализация на скорость

Собственный benchmark

- DTO специфичные – не стали доверять другим
- На Java Microbenchmark Harness (JMH)
- Выложен на GitHub: https://github.com/chernov-af/serializers-benchmark

Runtime конфигурация

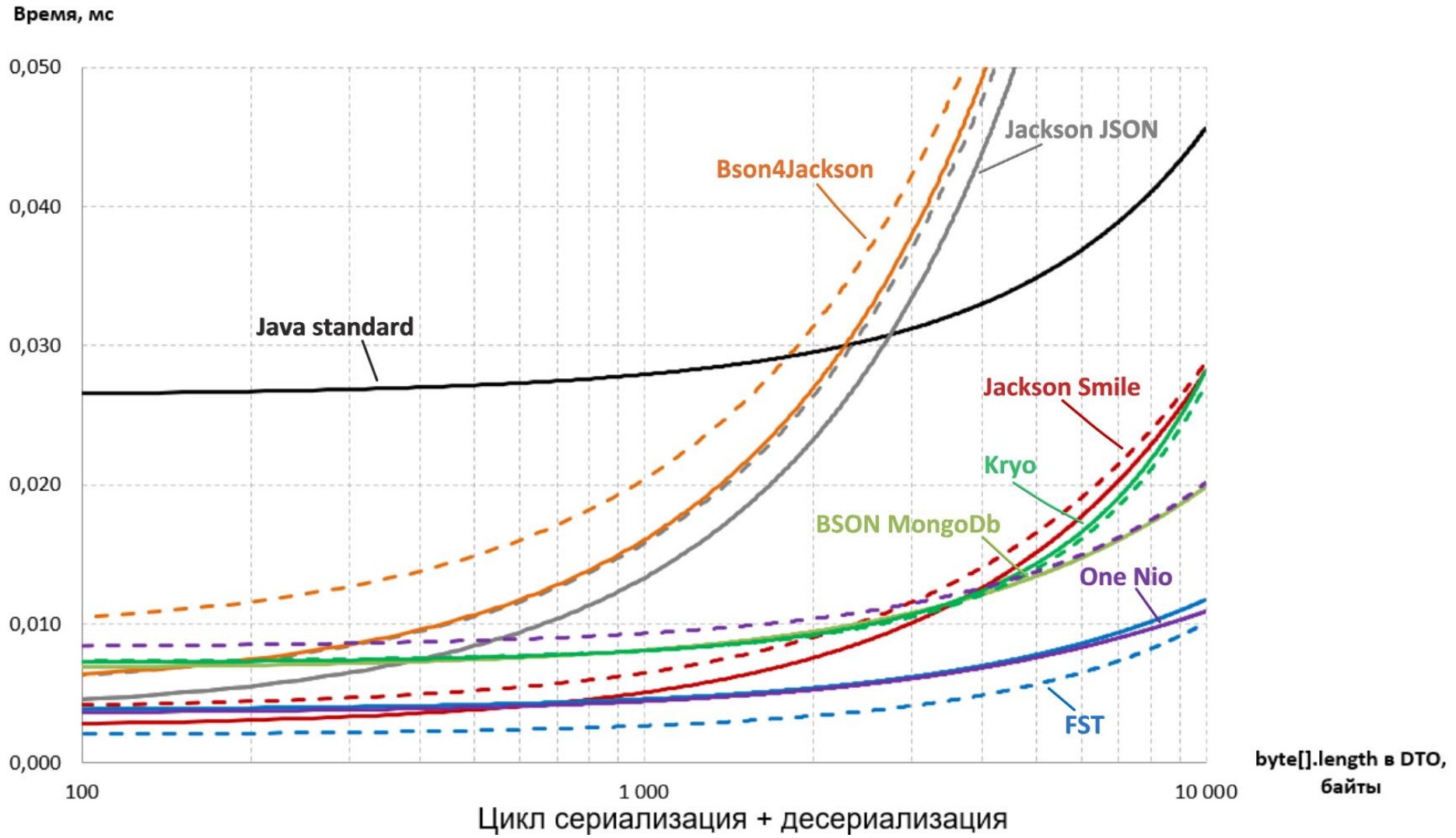
- Процессор:
  Intel Core i7-6700 CPU, 3.4GHz,
  8 cores
- Память: 16 GB
- ОС: Microsoft Windows 10 (64-bit)
- **JRE: IBM J9 VM 1.7.0**

Гонки между сериализаторами-участниками

Все 8 участников

Детали здесь:

https://habr.com/ru/company/sberbank/blog/488612/

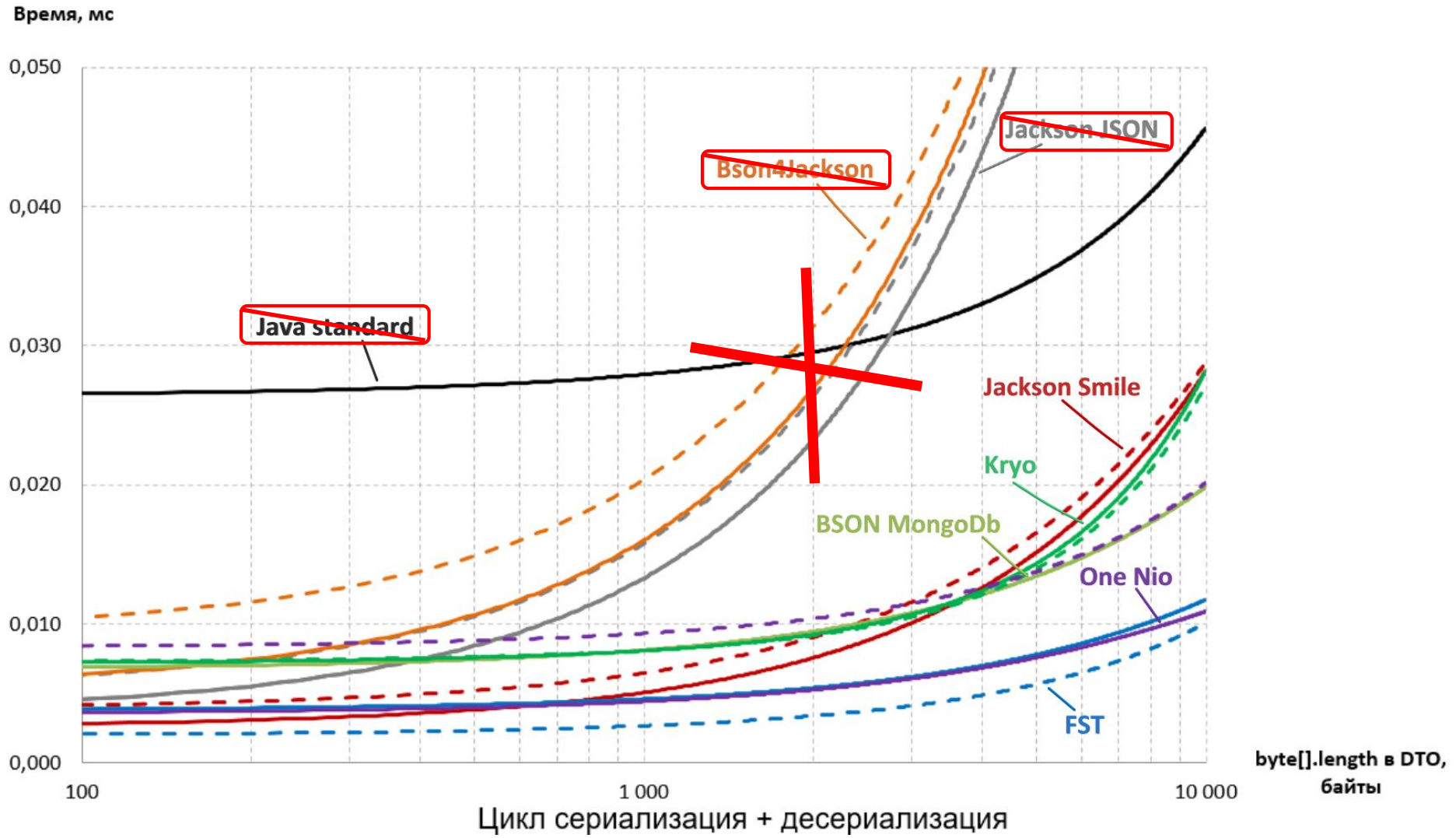


Гонки между сериализаторами-участниками

3 аутсайдера

Детали здесь:

https://habr.com/ru/company/sberbank/blog/488612/

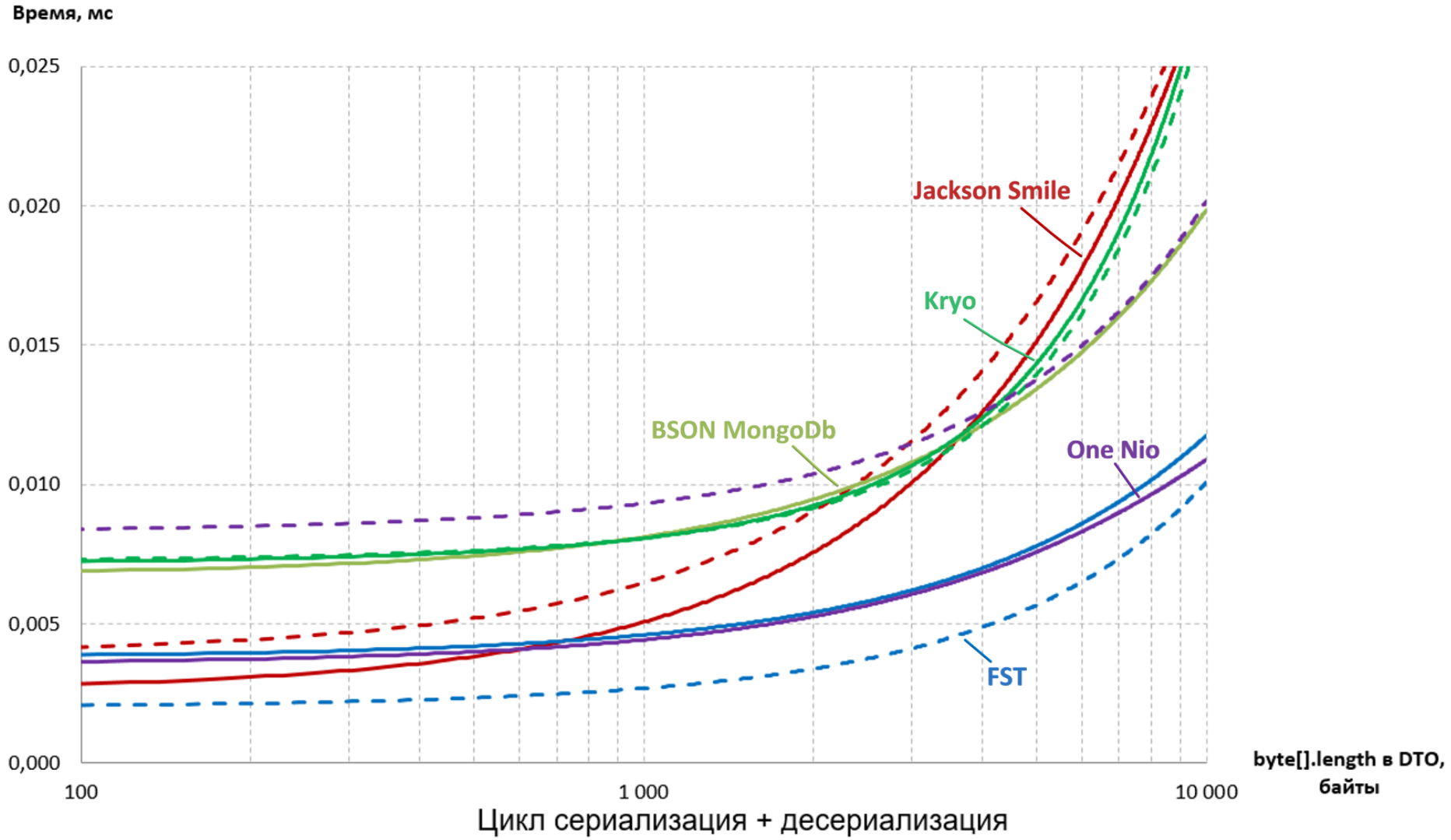


Гонки между сериализаторами-участниками

5 оставшихся участников

Детали здесь:

https://habr.com/ru/company/sberbank/blog/488612/

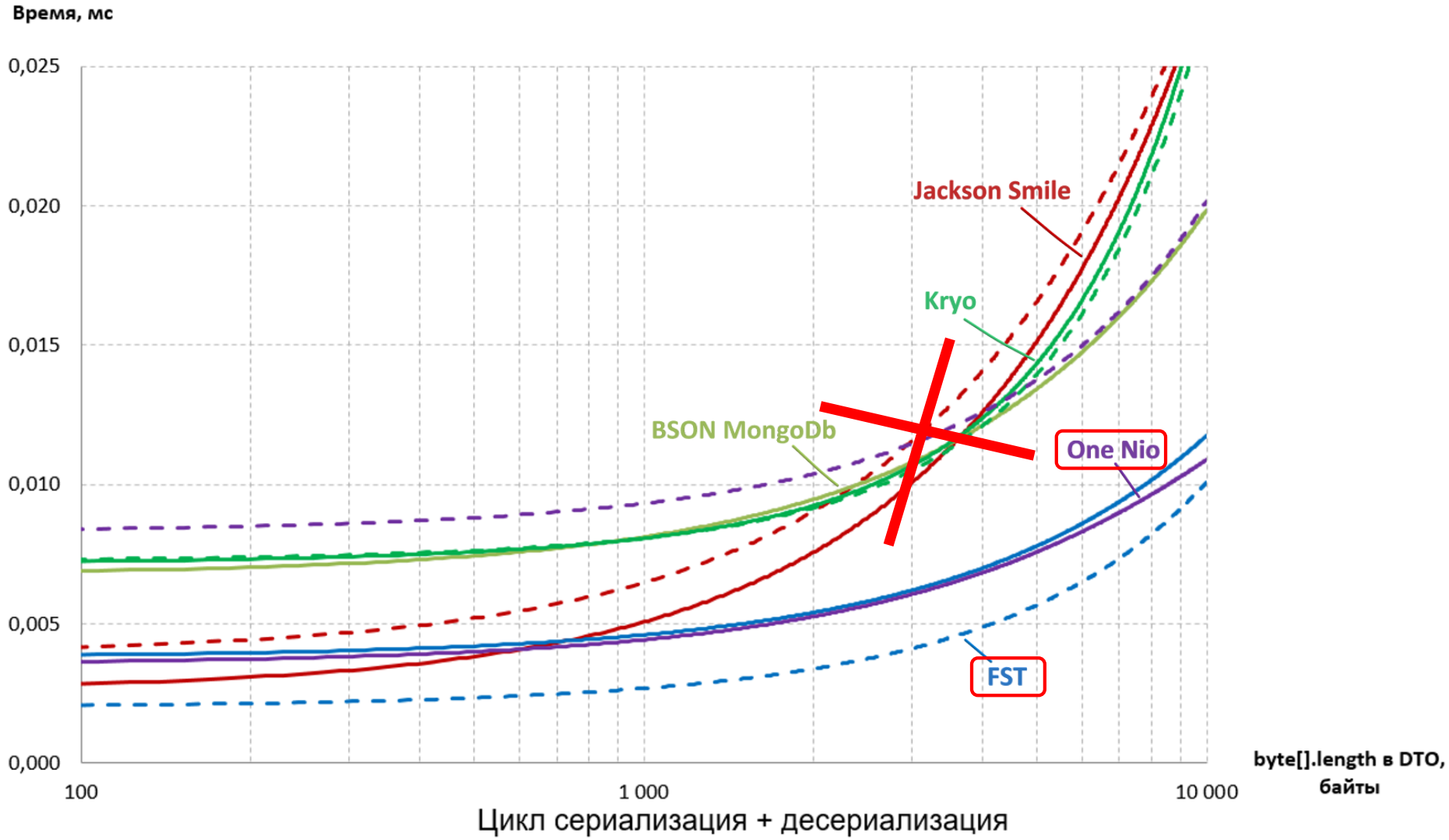


Гонки между сериализаторами-участниками

2 лидера

Детали здесь:

https://habr.com/ru/company/sberbank/blog/488612/

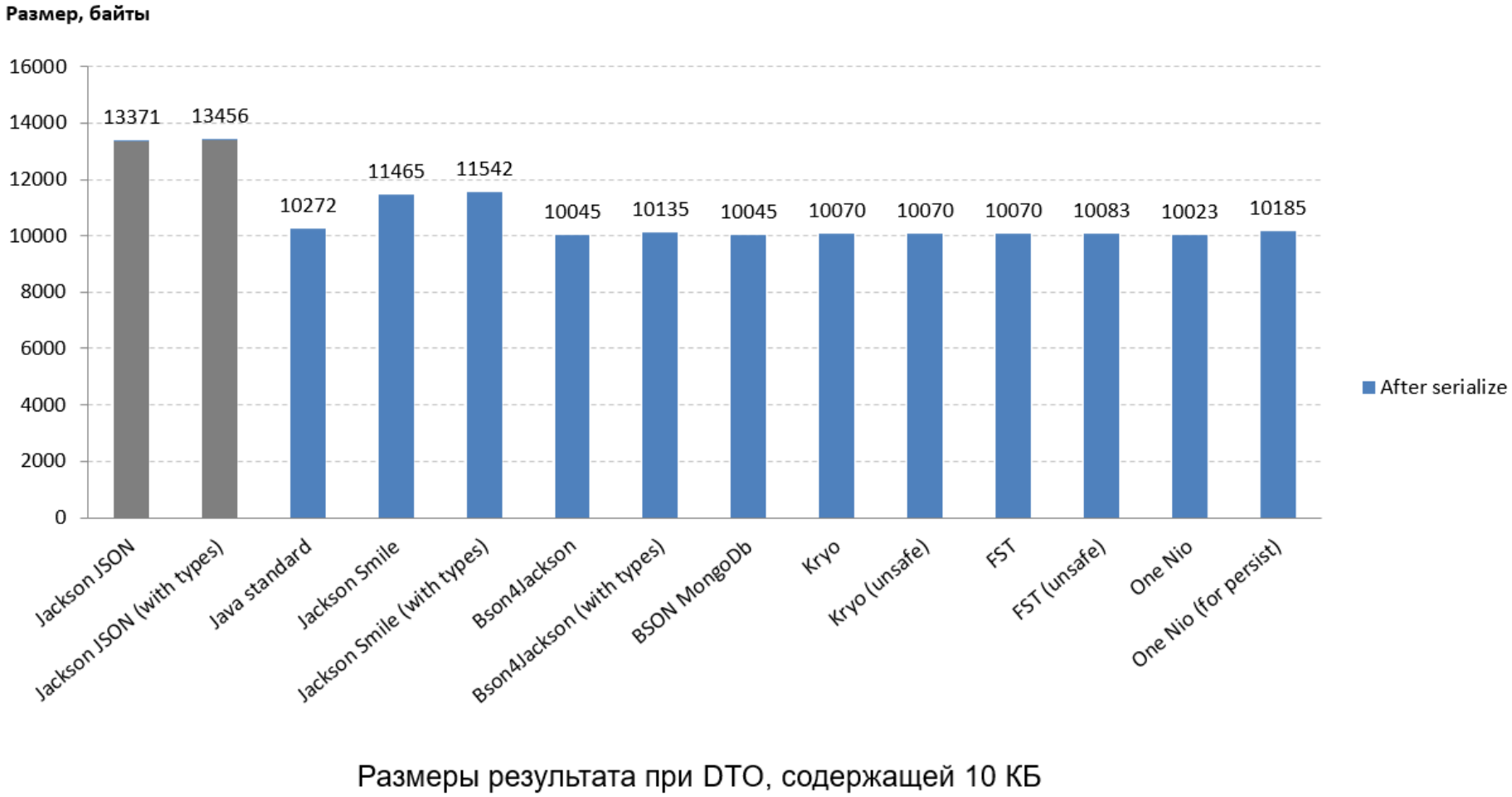


Взвешивание результатов сериализаторов-участников

Все 8 участников (некоторые в 2-х конфигурациях)

Детали здесь:

https://habr.com/ru/company/sberbank/blog/488612/



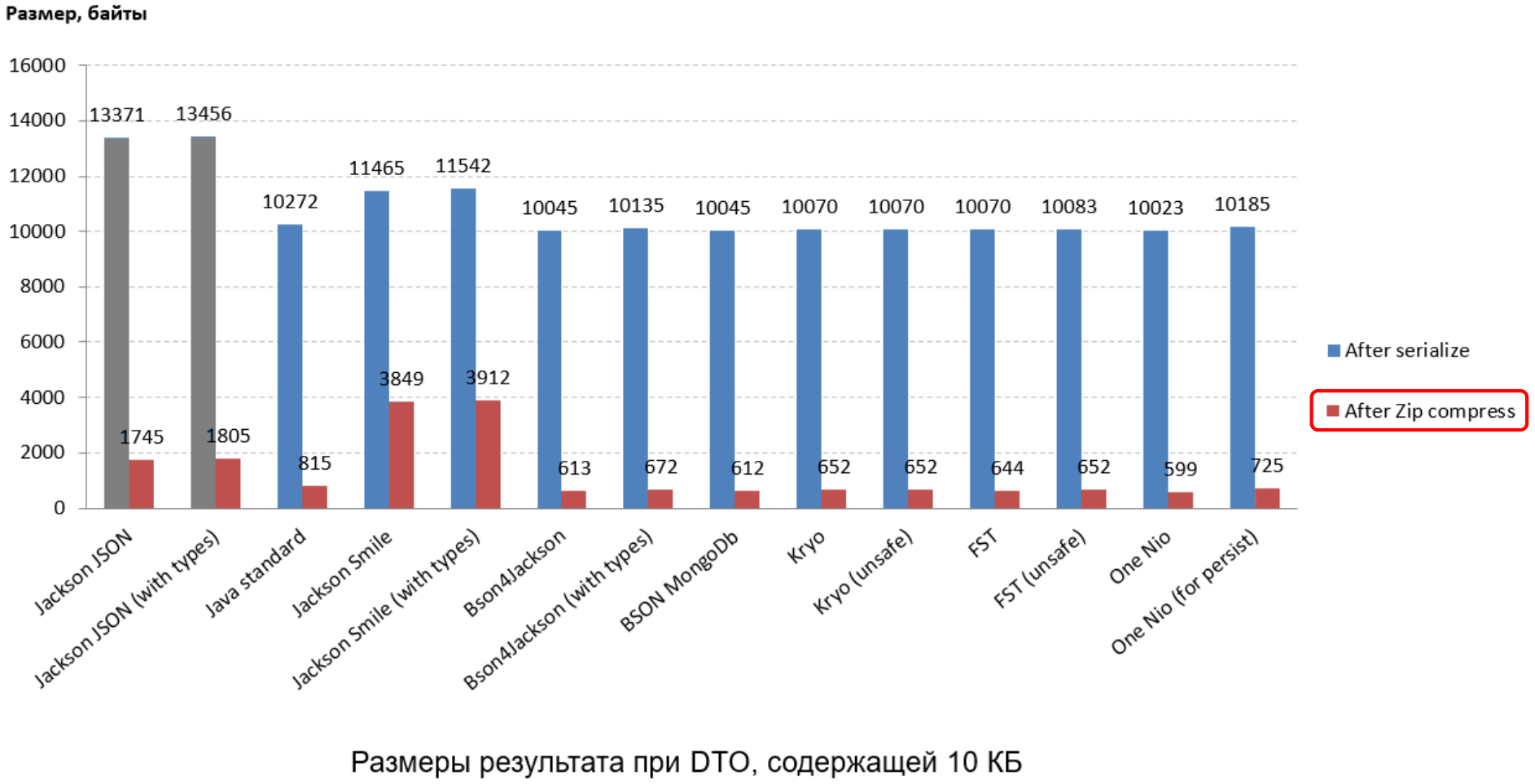
Размеры результата при DTO, содержащей 10 КБ

Взвешивание результатов сериализаторов-участников

Все 8 участников (некоторые в 2-х конфигурациях)

Детали здесь:

https://habr.com/ru/company/sberbank/blog/488612/



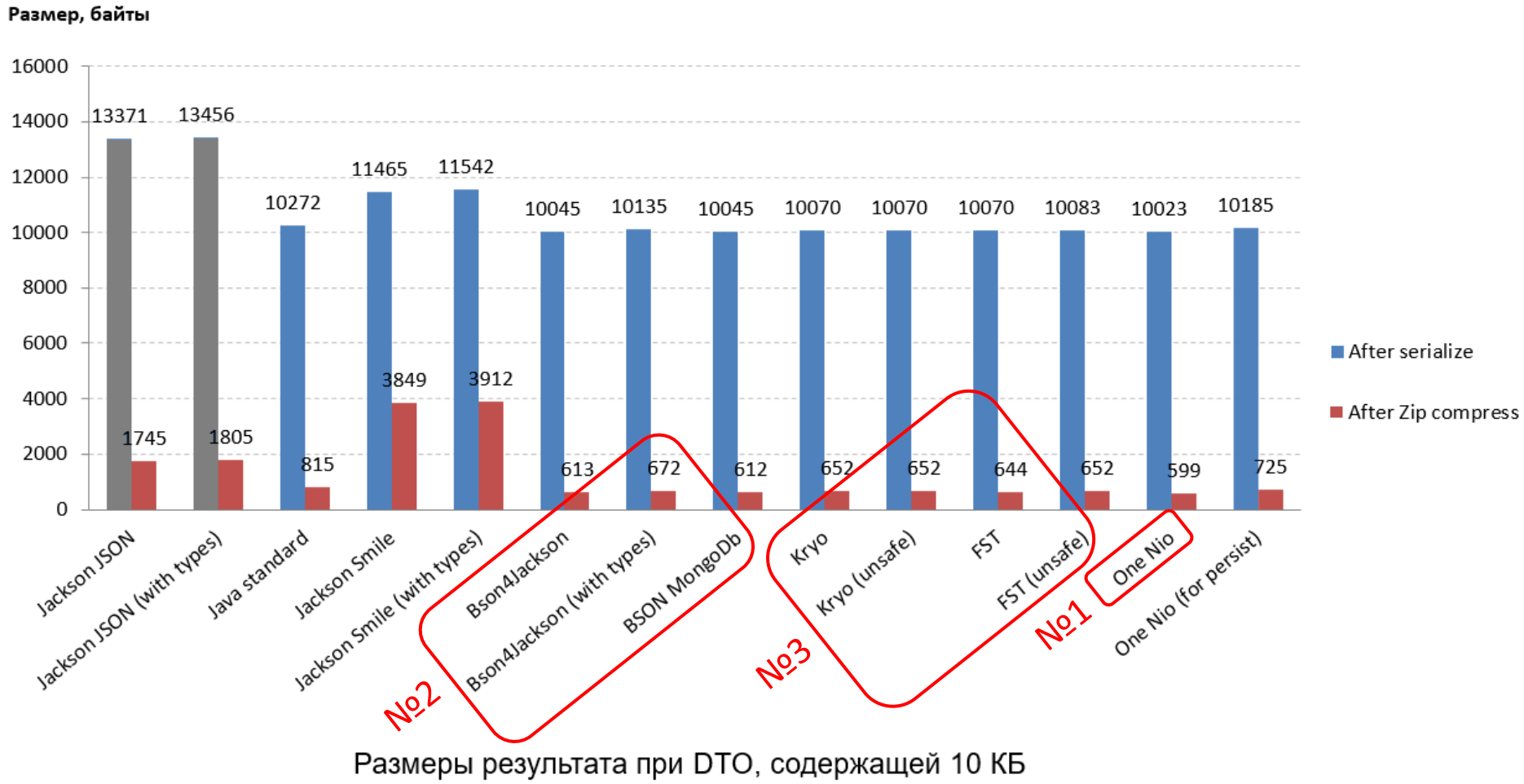
Размеры результата при DTO, содержащей 10 КБ

Взвешивание результатов сериализаторов-участников

Лидеры

Детали здесь:

https://habr.com/ru/company/sberbank/blog/488612/



Размеры результата при DTO, содержащей 10 КБ

Гибкость сериализаторов-участников

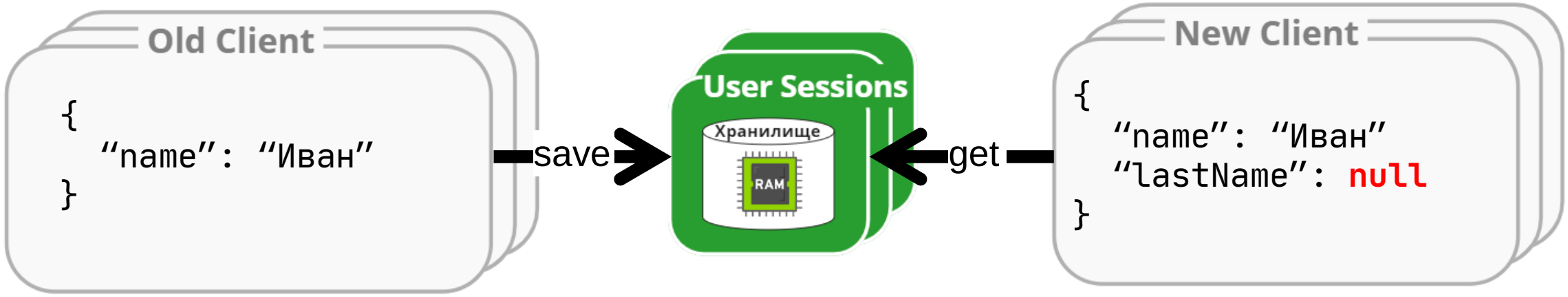
Потребители разных версий должны уметь работать с одними объектами

Критерий	Jackson JSON	Jackson JSON (with types)	Java standard	Jackson Smile	Jackson Smile (with types)	Bson4Jackson	Bson4Jackson (with types)	BSON MongoDb	Kryo	Kryo (unsafe)	FST	FST (unsafe)	One Nio	One Nio (for persist)

Все 20 критериев здесь: https://habr.com/ru/company/sberbank/blog/488612/

Гибкость сериализаторов-участников

Критерий	Jackson JSON	Jackson JSON (with types)	Java standard	Jackson Smile	Jackson Smile (with types)	Bson4Jackson	Bson4Jackson (with types)	BSON MongoDb	Kryo	Kryo (unsafe)	FST	FST (unsafe)	One Nio	One Nio (for persist)
Обратная совместимость	+	+	+	+	+	+	+	+	+	+	–	–	–	+



Гибкость сериализаторов-участников

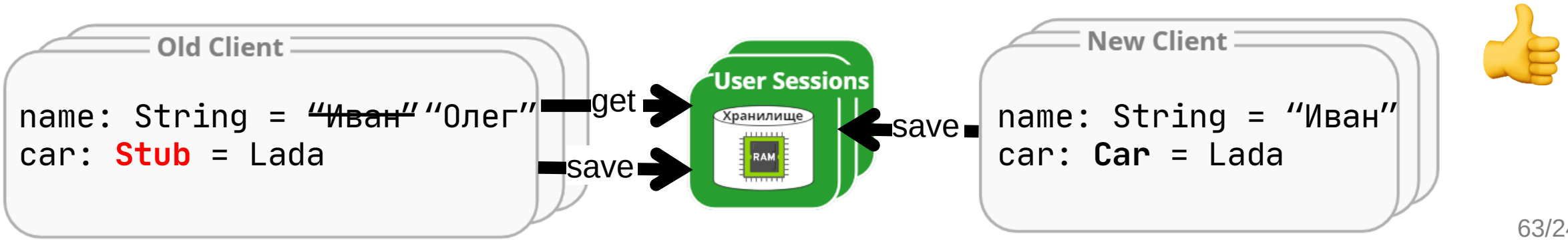
Критерий	Jackson JSON	Jackson JSON (with types)	Java standard	Jackson Smile	Jackson Smile (with types)	Bson4Jackson	Bson4Jackson (with types)	BSON MongoDb	Kryo	Kryo (unsafe)	FST	FST (unsafe)	One Nio	One Nio (for persist)
Обратная совместимость	+	+	+	+	+	+	+	+	+	+	–	–	–	+
Прямая совместимость	+	+	+	+	+	+	+	+	+	+	–	–	–	+

Гибкость сериализаторов-участников

Критерий	Jackson JSON	Jackson JSON (with types)	Java standard	Jackson Smile	Jackson Smile (with types)	Bson4Jackson	Bson4Jackson (with types)	BSON MongoDb	Kryo	Kryo (unsafe)	FST	FST (unsafe)	One Nio	One Nio (for persist)
Обратная совместимость	+	+	+	+	+	+	+	+	+	+	–	–	–	+
Прямая совместимость	+	+	+	+	+	+	+	+	+	+	–	–	–	+
Десериализация объекта, класса которого нет в classpath	+/-	–	–	+/-	–	+/-	–	+/-	–	–	–	–	–	+

Гибкость сериализаторов-участников

Критерий	Jackson JSON	Jackson JSON (with types)	Java standard	Jackson Smile	Jackson Smile (with types)	Bson4Jackson	Bson4Jackson (with types)	BSON MongoDb	Kryo	Kryo (unsafe)	FST	FST (unsafe)	One Nio	One Nio (for persist)
Обратная совместимость	+	+	+	+	+	+	+	+	+	+	–	–	–	+
Прямая совместимость	+	+	+	+	+	+	+	+	+	+	–	–	–	+
Десериализация объекта, класса которого нет в classpath	+/-	–	–	+/-	–	+/-	–	+/-	–	–	–	–	–	+



Гибкость сериализаторов-участников

Лидеры

Критерий	Jackson JSON	Jackson JSON (with types)	Java standard	Jackson Smile	Jackson Smile (with types)	Bson4Jackson	Bson4Jackson (with types)	BSON MongoDb	Kryo	Kryo (unsafe)	FST	FST (unsafe)	One Nio	One Nio (for persist)
Обратная совместимость	+	+	+	+	+	+	+	+	+	+	–	–	–	+
Прямая совместимость	+	+	+	+	+	+	+	+	+	+	–	–	–	+
Десериализация объекта, класса которого нет в classpath	+/-	–	–	+/-	–	+/-	–	+/-	–	–	–	–	–	+
Все 20 критериев здесь: https://habr.com/ru/company/sberbank/blog/488612/														
Всего баллов:	16	11	13	16	11	16	11	14,5	13	13	11	11	10,5	19

Гибкость сериализаторов-участников

Лидеры

Критерий	Jackson JSON	Jackson JSON (with types)	Java standard	Jackson Smile	Jackson Smile (with types)	Bson4Jackson	Bson4Jackson (with types)	BSON MongoDb	Kryo	Kryo (unsafe)	FST	FST (unsafe)	One Nio	One Nio (for persist)
Обратная совместимость	+	+	+	+	+	+	+	+	+	+	–	–	–	+
Прямая совместимость	+	+	+	+	+	+	+	+	+	+	–	–	–	+
Десериализация объекта, класса которого нет в classpath	+/-	–	–	+/-	–	+/-	–	+/-	–	–	–	–	–	+

Все 20 критериев здесь: https://habr.com/ru/company/sberbank/blog/488612/

Всего баллов:	16	11	13	16	11	16	11	14,5	13	13	11	11	10,5	19

После обмен схемами, быстрая One Nio стала гибкой, как One Nio (for persist)!

Результат соревнования сериализаторов

Победитель

One Nio

Что он получил

- Использование на транспортном уровне сериализации
- Использование на бизнесовом уровне сериализации

Fork One Nio

Основные доработки

1.sun.misc.Unsafe вместо sun.reflect.MagicAccessorImpl
2.Оптимизация сериализации строк
3.Возможность переименовывать классы несколько раз
4.Возможность регистрировать custom-ные сериализаторы
5.Десериализация Collection/Map с приватным конструктором

Fork One Nio

Основные доработки

1.sun.misc.Unsafe вместо sun.reflect.MagicAccessorImpl

Доработан код генерации байткода сериализаторов.
MagicAccessorImpl не работает на IBM JDK, а также на любом JDK при Java >= 9.
Автоматическое детектирование, что можно использовать MagicAccessorImpl.

2.Оптимизация сериализации строк

3.Возможность переименовывать классы несколько раз

4.Возможность регистрировать custom-ные сериализаторы

5.Десериализация Collection/Map с приватным конструктором

Fork One Nio

Основные доработки

1.sun.misc.Unsafe вместо sun.reflect.MagicAccessorImpl

2.Оптимизация сериализации строк

String.getBytes(UTF8) и Unsafe.copyMemory() вместо метода Uft8.write()
Скорость сериализации строк увеличилась на 30-40%.

3.Возможность переименовывать классы несколько раз

4.Возможность регистрировать custom-ные сериализаторы

5.Десериализация Collection/Map с приватным конструктором

Fork One Nio

Основные доработки

1. sun.misc.Unsafe вместо sun.reflect.MagicAccessorImpl
2. Оптимизация сериализации строк
3. **Возможность переименовывать классы несколько раз**
   @Renamed позволяет указать только 1 предыдущее имя класса/поля.
   Новая @Renames позволяет указать массив хронологических переименований.
4. Возможность регистрировать custom-ные сериализаторы
5. Десериализация Collection/Map с приватным конструктором

Fork One Nio

Основные доработки

1. sun.misc.Unsafe вместо sun.reflect.MagicAccessorImpl
2. Оптимизация сериализации строк
3. Возможность переименовывать классы несколько раз

4. Возможность регистрировать custom-ные сериализаторы

Теперь можно так:

```
Repository.provideSerializer(new CustomSerializer(CustomClass.class))
```

Раньше это приводило к StackOverflowError.

5. Десериализация Collection/Map с приватным конструктором

Fork One Nio

Основные доработки

1. sun.misc.Unsafe вместо sun.reflect.MagicAccessorImpl
2. Оптимизация сериализации строк
3. Возможность переименовывать классы несколько раз
4. Возможность регистрировать custom-ные сериализаторы

5. Десериализация Collection/Map с приватным конструктором

Результат – исходный класс, а не ArrayList / HashSet / HashMap.
Просто использовали constructor.setAccessible(true)

Код сериализации с использованием One Nio

См. benchmark:
https://github.com/chernov-af/serializers-benchmark

Оптимизационный трюк для сериализации PersistStream-ом (спойлер – пытаемся угадать размер).

Ускорение от использования One Nio

После замены JSON-а на One Nio время отклика микросервиса User Sessions уменьшилось **~ в 2 раза**!

План

~~1. Platform V User Sessions. Начало пути~~

~~1.1. Работа с данными и их передача по сети~~

2. Развитие User Sessions для выдерживания highload

~~2.1. Сериализация данных в бинарный формат One Nio~~

2.2. Сохранение данных «дельтами»

2.3. Чтение данных пачками

3. Как User Sessions достигает high availability

3.1. Load balancing & true sticky

3.2. Асинхронная работа с базой данных

3.3. Хранилище не в контейнере

3.4. No vendor lock

3.5. Репликация данных (в процессе реализации)

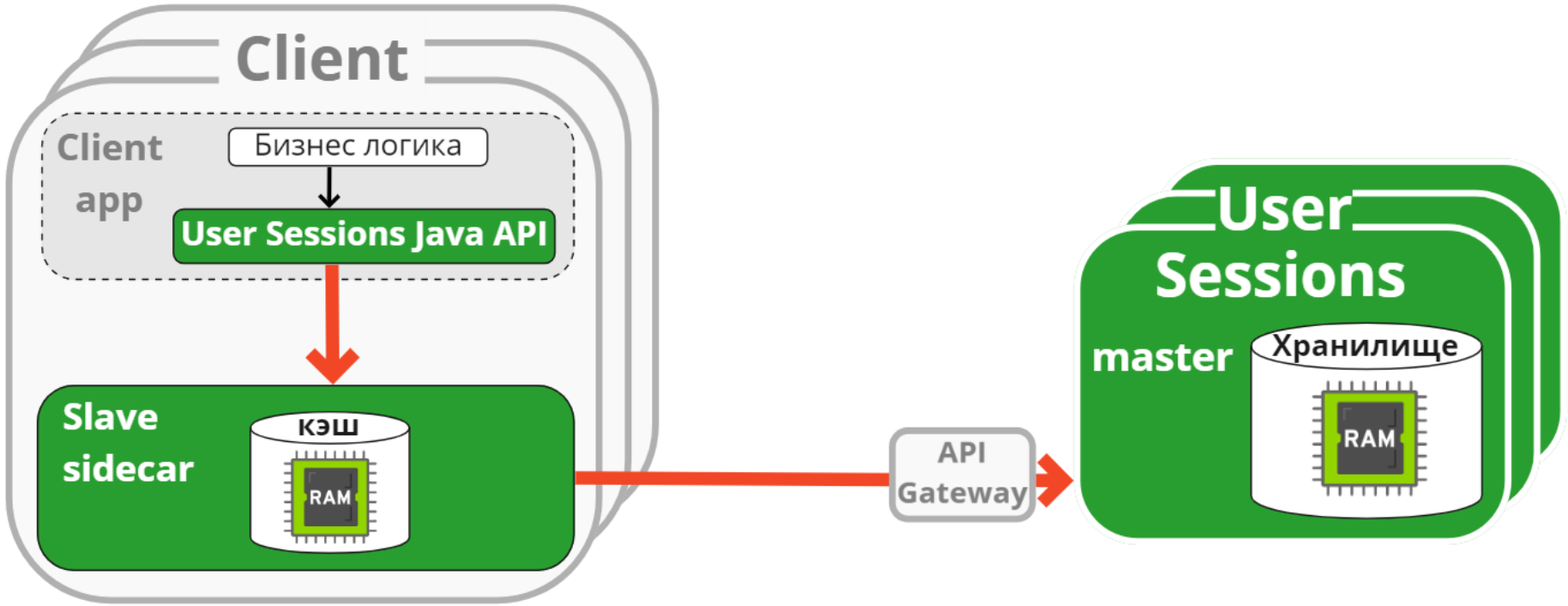
4. Дальнейшее развитие User Sessions

4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД

4.2. Авторизация доступа к данным через Open Policy Agent

4.3. «Не клиентские» сессии

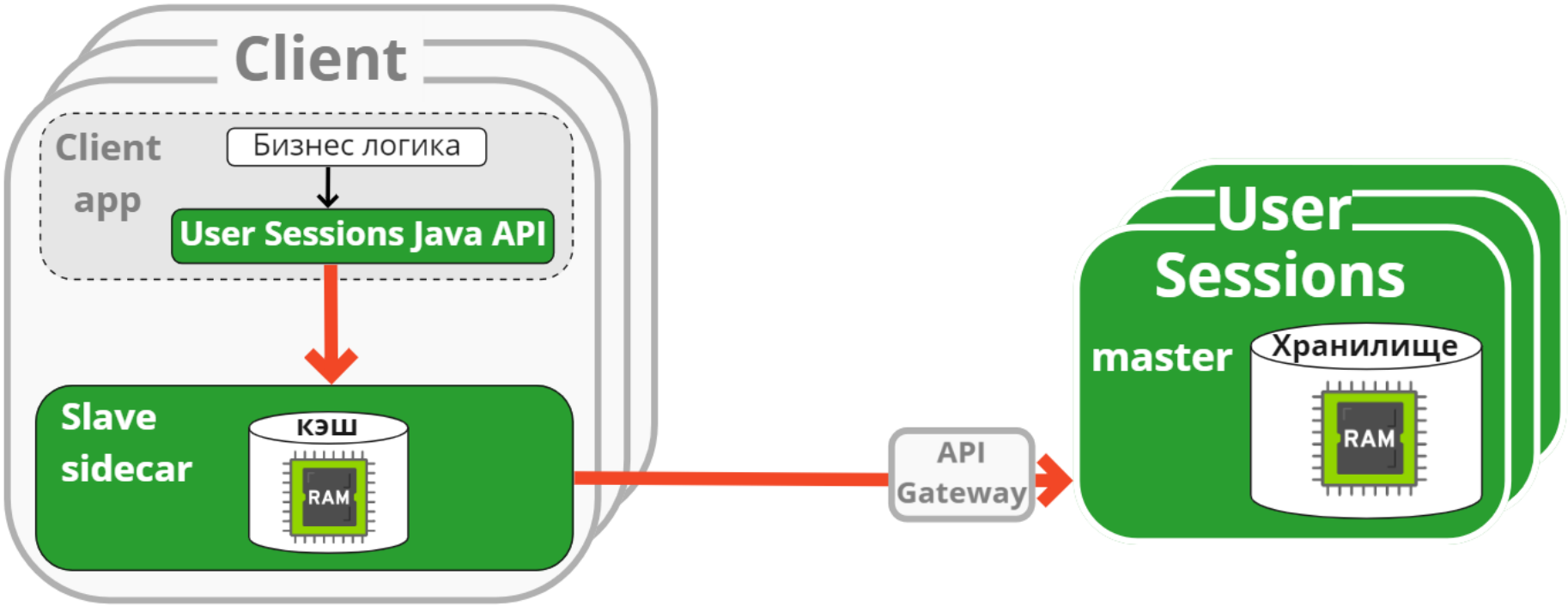
Сетевой трафик User Sessions



Сетевой трафик User Sessions

Секция

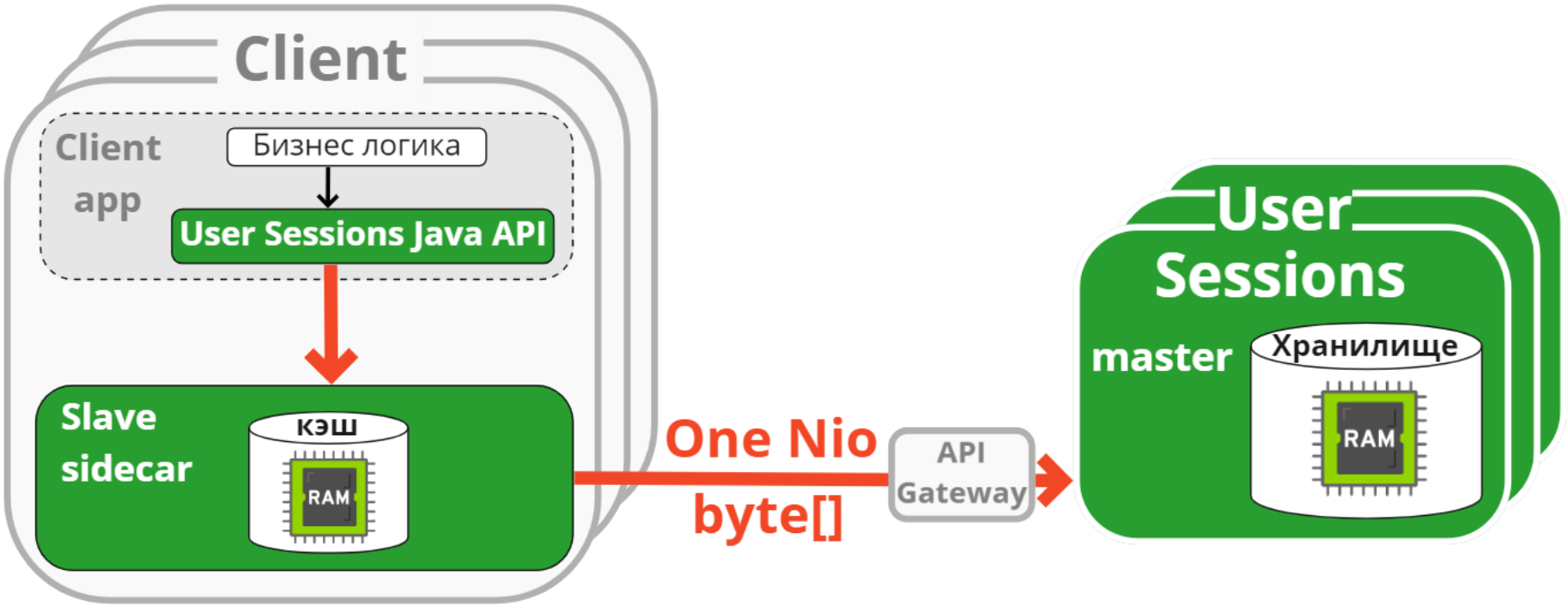
Минимальная единица передачи данных по сети



Сетевой трафик User Sessions

Секция

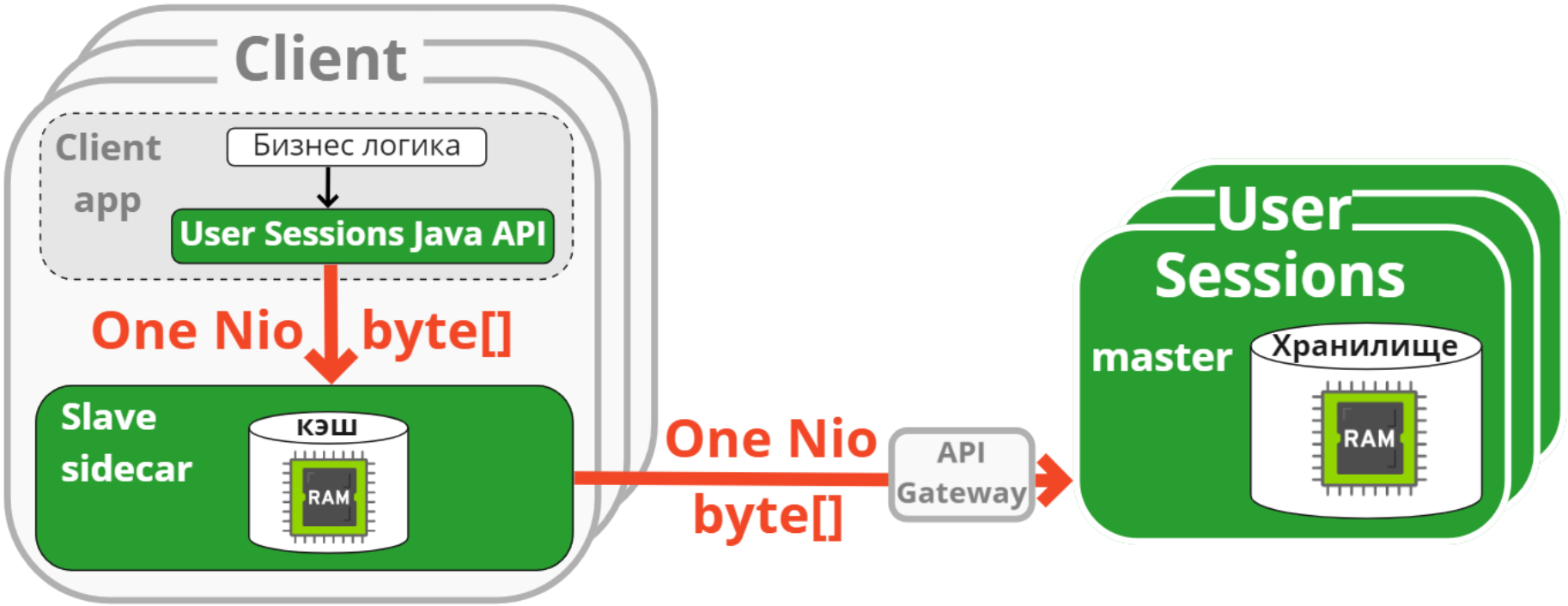
Минимальная единица передачи данных по сети



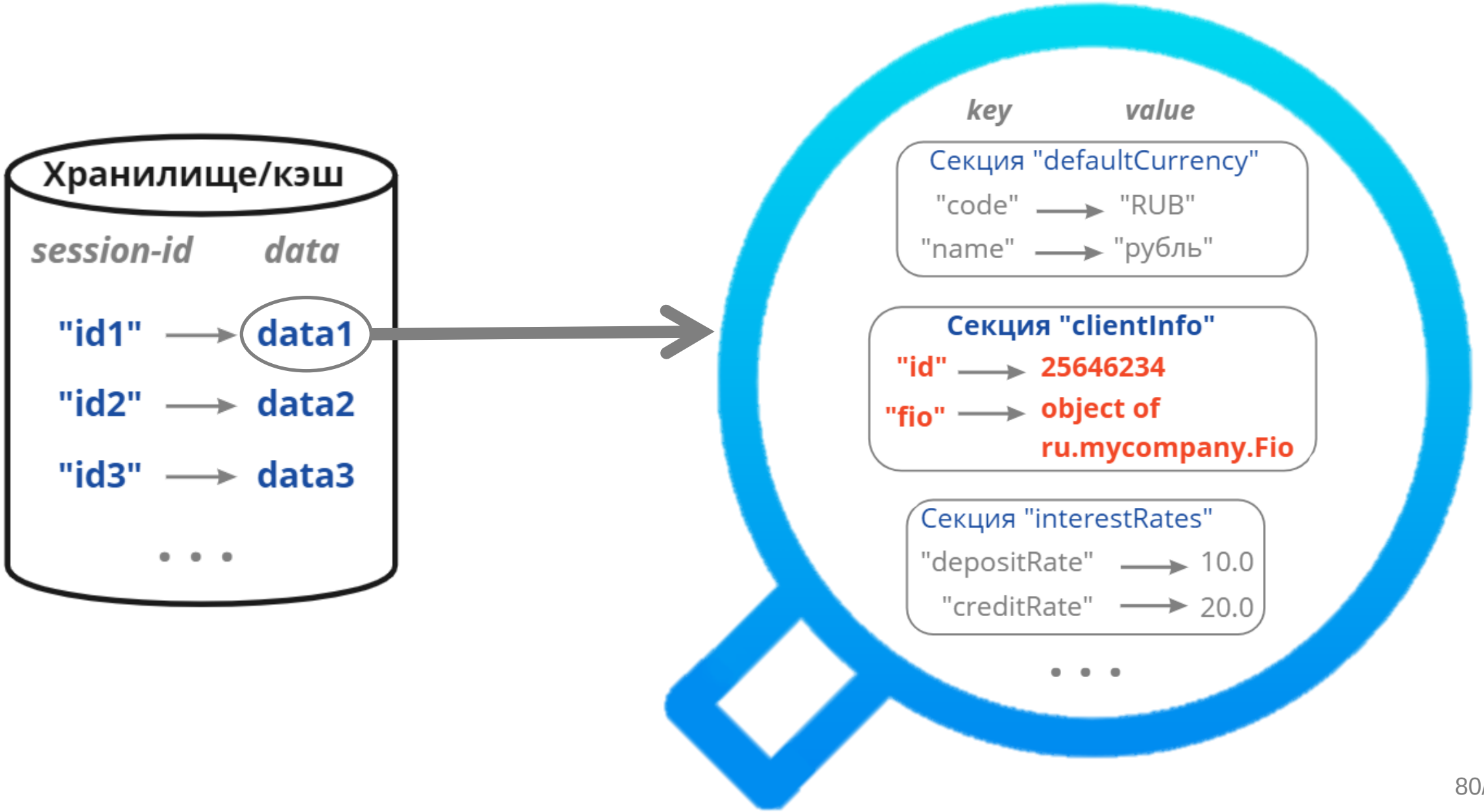
Сетевой трафик User Sessions

Секция

Минимальная единица передачи данных по сети



Передаваемая по сети секция состоит из атрибутов



Работа с User Sessions и передача данных

Микросервис 1

```
SdsSession session = sds.newSession(...);
```

Работа с User Sessions и передача данных

Микросервис 1

```
SdsSession session = sds.newSession(...);
```

Микросервис 2

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);


SdsSessionSection section =
    session.getSection("clientInfo");
section.setAttribute("name", "Иван");
section.setAttribute("lastName", "Иванов");
section.setAttribute("patronymic", "Иванович");
section.setAttribute("cards",
                     List.of("МИР Сберката", ...));
section.setAttribute("deposits",
                     List.of("СберВклад Прайм", ...));


sds.saveSessionSections(session, ...);
```

Работа с User Sessions и передача данных

Микросервис 1

```
SdsSession session = sds.newSession(...);
```

section целиком сериализуется, сжимается и передаётся по сети →

Микросервис 2

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);

SdsSessionSection section =
    session.getSection("clientInfo");
section.setAttribute("name", "Иван");
section.setAttribute("lastName", "Иванов");
section.setAttribute("patronymic", "Иванович");
section.setAttribute("cards",
                     List.of("МИР Сберката", ...));
section.setAttribute("deposits",
                     List.of("СберВклад Прайм", ...));

sds.saveSessionSections(session, ...);
```

Работа с User Sessions и передача данных

Микросервис 1

```
SdsSession session = sds.newSession(...);
```

section целиком сериализуется,
сжимается и передаётся по сети

нет лишнего трафика

Микросервис 2

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);

SdsSessionSection section =
    session.getSection("clientInfo");
section.setAttribute("name", "Иван");
section.setAttribute("lastName", "Иванов");
section.setAttribute("patronymic", "Иванович");
section.setAttribute("cards",
                     List.of("МИР Сберката", ...));
section.setAttribute("deposits",
                     List.of("СберВклад Прайм", ...));

sds.saveSessionSections(session, ...);
```

Работа с User Sessions и передача данных

Микросервис 3

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);


SdsSessionSection section =
    session.getSection("clientInfo");
section.setAttribute("deposits",
                     List.of("СберВклад Прайм",
                             "Лучший %"));


sds.saveSessionSections(session, ...);
```

Работа с User Sessions и передача данных

Микросервис 3

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);

SdsSessionSection section =
    session.getSection("clientInfo");
section.setAttribute("deposits",
                     List.of("СберВклад Прайм",
                             "Лучший %"));

sds.saveSessionSections(session, ...);
```

загружаются все 5 атрибутов:
name, lastName, patronymic, cards, deposits

Работа с User Sessions и передача данных

Микросервис 3

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);

SdsSessionSection section =
    session.getSection("clientInfo");
section.setAttribute("deposits",
                     List.of("СберВклад Прайм",
                             "Лучший %"));

sds.saveSessionSections(session, ...);
```

загружаются все 5 атрибутов:
name, lastName, patronymic, cards, deposits

нет лишнего трафика

Работа с User Sessions и передача данных

Микросервис 3

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);

SdsSessionSection section =
    session.getSection("clientInfo");
section.setAttribute("deposits",
                     List.of("СберВклад Прайм",
                             "Лучший %"));

sds.saveSessionSections(session, ...);
```

загружаются все 5 атрибутов:
name, lastName, patronymic, cards, deposits

section целиком сериализуется,
сжимается и передаётся по сети

Работа с User Sessions и передача данных

Микросервис 3

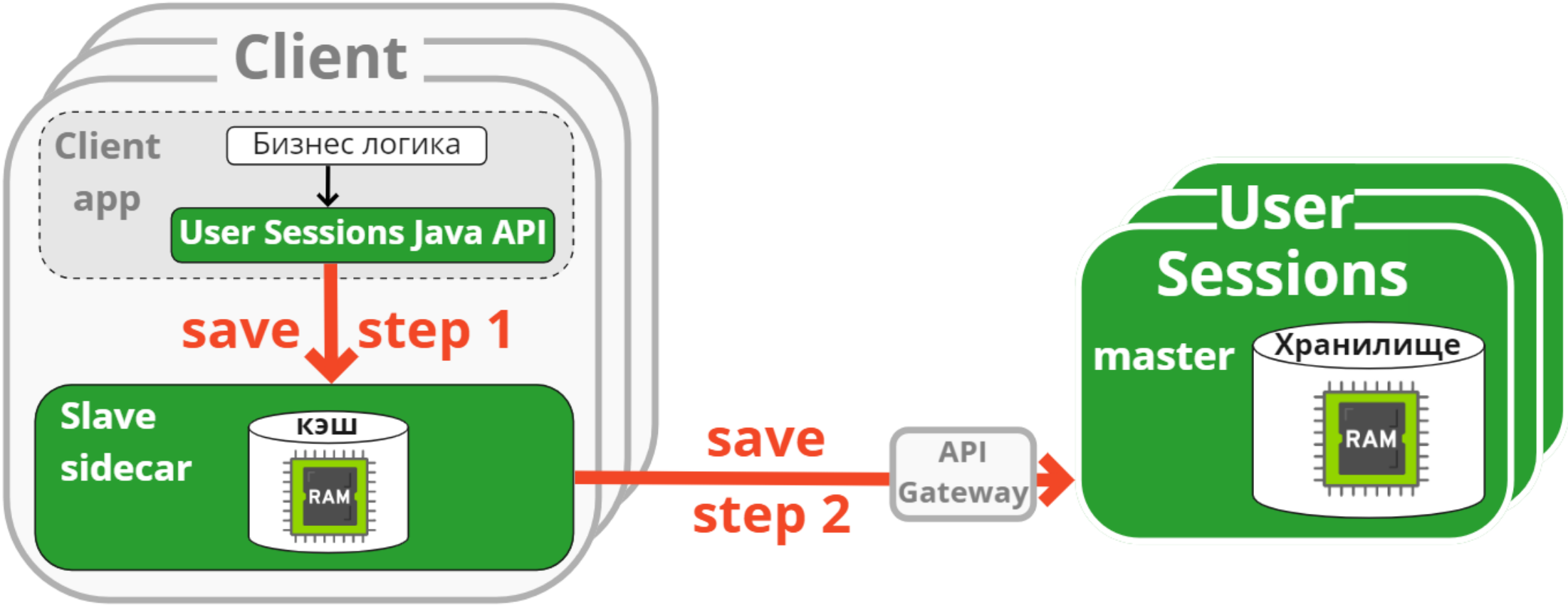
```
SdsSession session = sds.getSession(
    <session cookie value>, ...);

SdsSessionSection section =
    session.getSection("clientInfo");
section.setAttribute("deposits",
                     List.of("СберВклад Прайм",
                             "Лучший %"));

sds.saveSessionSections(session, ...);
```

загружаются все 5 атрибутов:
name, lastName, patronymic, cards, deposits

section целиком сериализуется,
сжимается и передаётся по сети

👎 **есть лишний трафик!**

Save всегда идёт до master-а



«Грязные» изменения атрибутов не сохраняются

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);

SdsSessionSection section =
    session.getSection("clientInfo");

List<String> deposits = section.getAttribute("deposits");
deposits.add("Лучший %");
section.setAttribute("deposits", deposits);

sds.saveSessionSections(session, ...);
```

«Грязные» изменения атрибутов не сохраняются

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);

SdsSessionSection section =
    session.getSection("clientInfo");

List<String> deposits = section.getAttribute("deposits");
deposits.add("Лучший %");
section.setAttribute("deposits", deposits);

sds.saveSessionSections(session, ...);
```

«Грязные» изменения атрибутов не сохраняются

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);

SdsSessionSection section =
    session.getSection("clientInfo");

List<String> deposits = section.getAttribute("deposits");
deposits.add("Лучший %");
section.setAttribute("deposits", deposits);

sds.saveSessionSections(session, ...);
```

недостаточно!

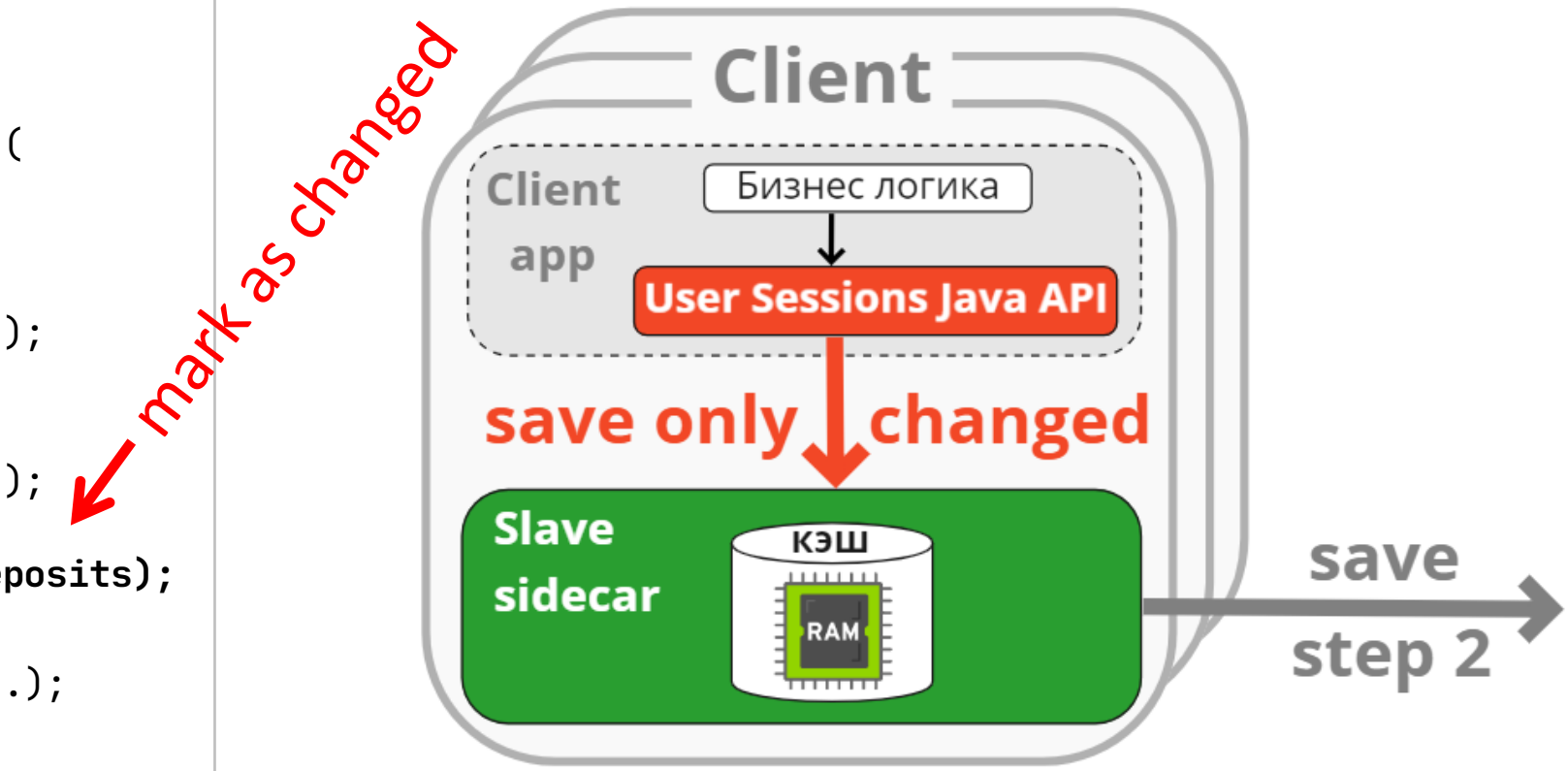
нужно явно сохранить

Только изменённые атрибуты передаются по сети

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);

SdsSessionSection section =
    session.getSection("clientInfo");

List<String> deposits =
    section.getAttribute("deposits");
deposits.add("Лучший %");
section.setAttribute("deposits", deposits);

sds.saveSessionSections(session, ...);
```

Только изменённые атрибуты передаются по сети

```
SdsSession session = sds.getSession(
    <session cookie value>, ...);

SdsSessionSection section =
    session.getSection("clientInfo");

List<String> deposits =
    section.getAttribute("deposits");
deposits.add("Лучший %");
section.setAttribute("deposits", deposits);

sds.saveSessionSections(session, ...);
```



Доработка DTO для передачи данных по сети

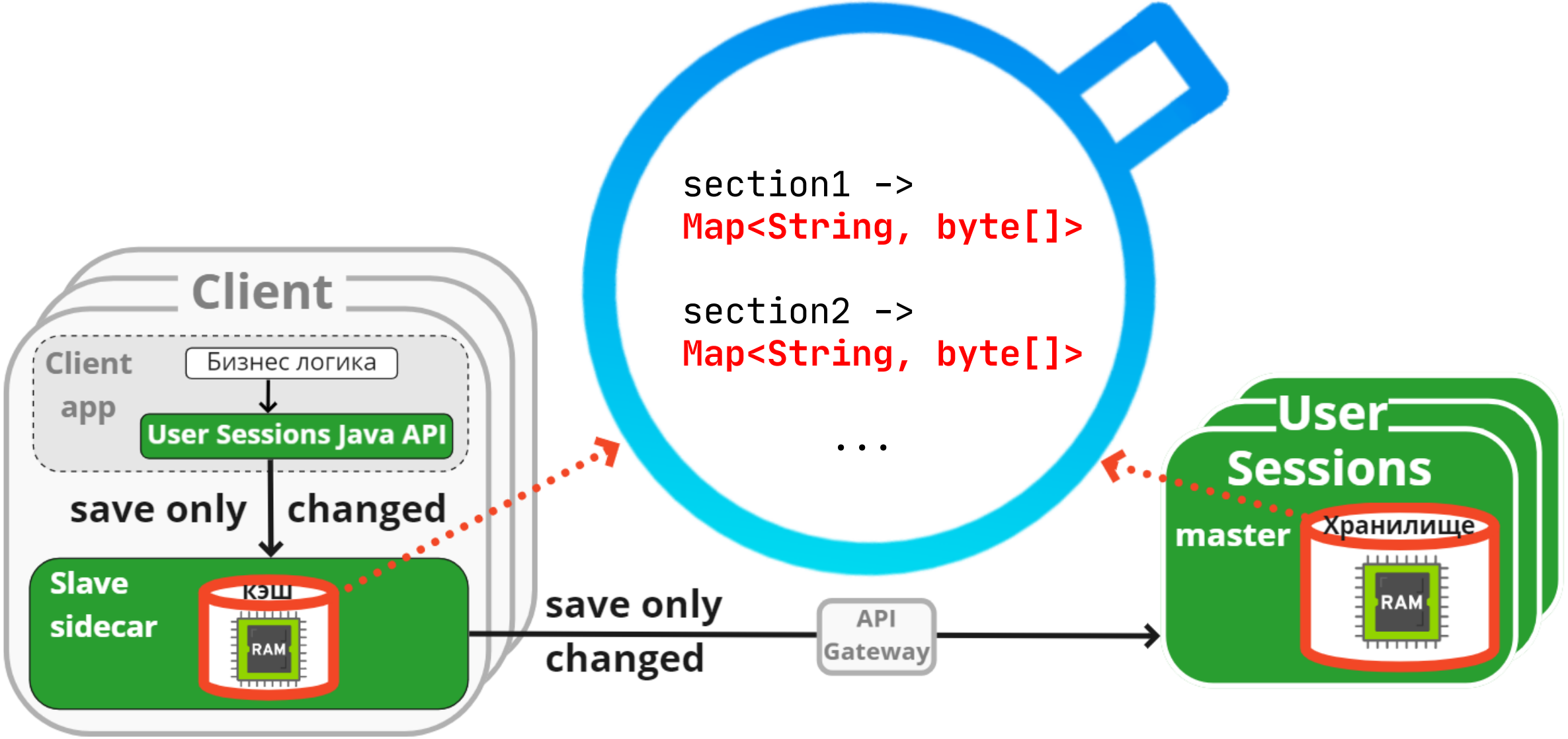
Было

```
public class <DTO> {
    ...
    public final List<SectionDto> sectionList;
    ...
}
public class SectionDto {
    public final String sectionName;
    public final Integer characteristics;
    public final Long version;
    public final byte[] serializedBinData;
    ...
}
```

Стало

```
public class <DTO> {
    ...
    public final List<SectionDto> sectionList;
    ...
}
public class SectionDto {
    public final String sectionName;
    public final Integer characteristics;
    public final Long version;
    public final Map<String, byte[]> serializedDataMap;
    ...
}
```

Доработка хранилища данных



Ускорение от сохранения данных «дельтами»

Среднее время отклика микросервиса User Sessions уменьшилось **~ на 33%**!

Ускорение от сохранения данных «дельтами»

Среднее время отклика микросервиса User Sessions уменьшилось **~ на 33%**!

Суммарно с One Nio общее ускорение **~ в 3 раза**!

План

~~1. Platform V User Sessions. Начало пути~~

~~1.1. Работа с данными и их передача по сети~~

2. Развитие User Sessions для выдерживания highload

~~2.1. Сериализация данных в бинарный формат One Nio~~

~~2.2. Сохранение данных «дельтами»~~

2.3. Чтение данных пачками

3. Как User Sessions достигает high availability

3.1. Load balancing & true sticky

3.2. Асинхронная работа с базой данных

3.3. Хранилище не в контейнере

3.4. No vendor lock

3.5. Репликация данных (в процессе реализации)

4. Дальнейшее развитие User Sessions

4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД

4.2. Авторизация доступа к данным через Open Policy Agent

4.3. «Не клиентские» сессии

Чтение данных «дельтами» невозможно

Факт

Неизвестно, какими атрибутами прочитанной секции воспользуются.

Чтение данных «дельтами» невозможно

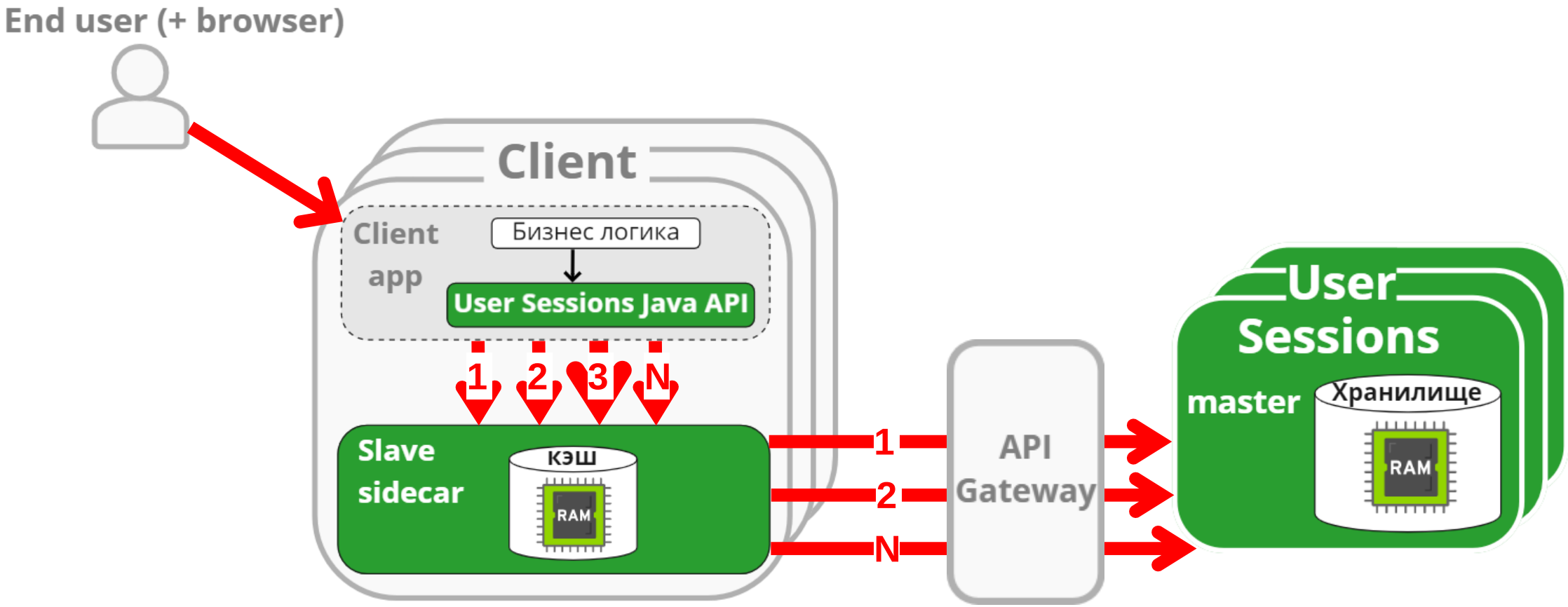
Факт

Неизвестно, какими атрибутами прочитанной секции воспользуются.

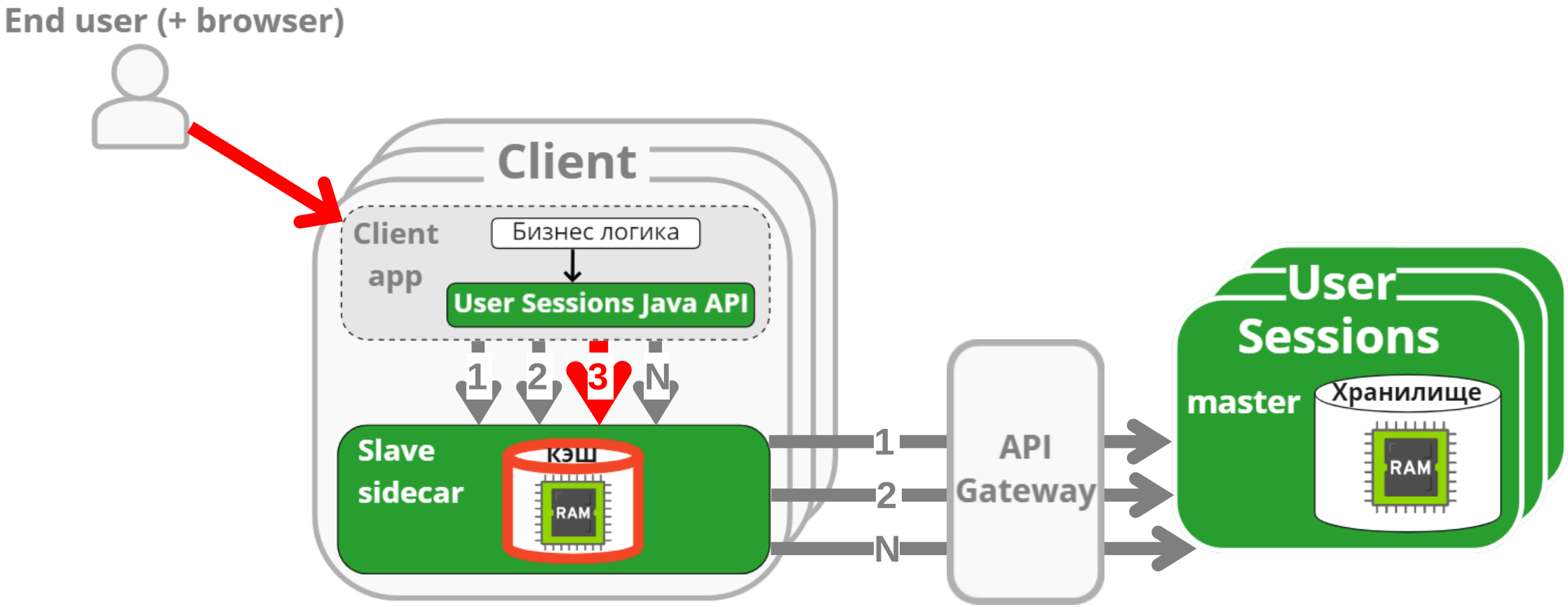
Следствие

Секция данных читается целиком, а не «дельтами».

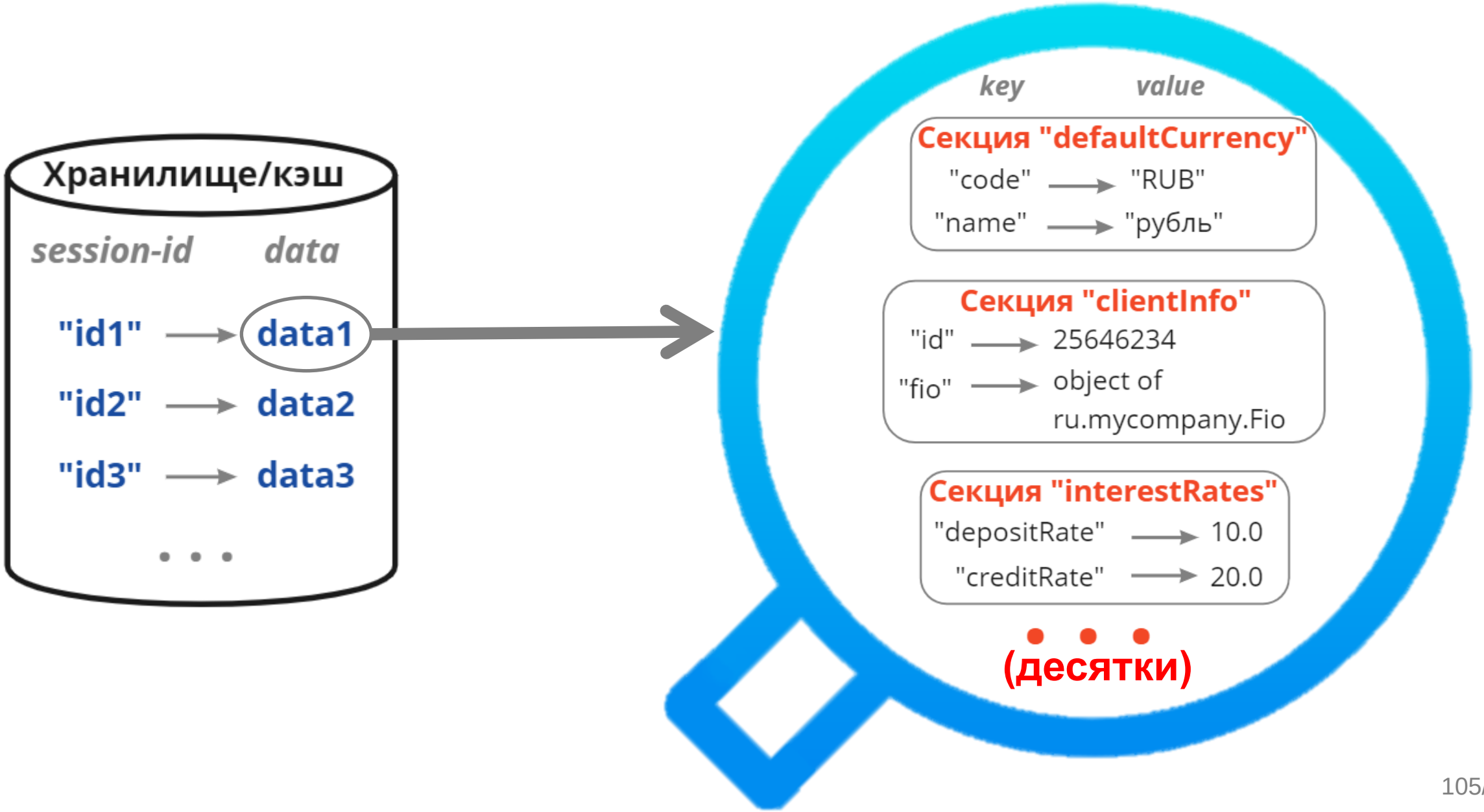
Несколько чтений за 1 запрос end user-а



Несколько чтений за 1 запрос end user-а



В сессии много секций данных



Для 1 бизнес-операции нужно несколько секций

Пример микросервиса

Контроллер

```
@RestController
@RequestMapping(path = "/v1", ...)
public class DepositController {

    @Autowired
    private DepositService deposits;

    @PostMapping("/deposit")
    public Deposit createDeposit(@CookieValue("SESSION-ID") String sessionId) {
        return deposits.createDeposit(sessionId);
    }
}
```

Для 1 бизнес-операции нужно несколько секций

Пример микросервиса

Контроллер

```
@RestController
@RequestMapping(path = "/v1", ...)
public class DepositController {

    @Autowired
    private DepositService deposits;

    @PostMapping("/deposit")
    public Deposit createDeposit(@CookieValue("SESSION-ID") String sessionId) {
        return deposits.createDeposit(sessionId);  // создание вклада
    }
}
```

Для 1 бизнес-операции нужно несколько секций

Пример микросервиса

Spring бин

```
public Deposit createDeposit(String sessionId) {
    SdsSession session = sds.getSession(sessionId, ...);
    var deposit = new Deposit();

    SdsSessionSection clientInfo = session.getSection("clientInfo");
    deposit.setClientId(clientInfo.getAttribute("id"));

    SdsSessionSection defaultCurrency = session.getSection("defaultCurrency");
    deposit.setCurrency(defaultCurrency.getAttribute("code"));

    SdsSessionSection rates = session.getSection("interestRates");
    deposit.setRate(rates.getAttribute("depositRate"));

    // ... сохранение deposit в БД
    return deposit;
}
```

получение сессии – запрос к slave

Для 1 бизнес-операции нужно несколько секций

Пример микросервиса

Spring бин

```
public Deposit createDeposit(String sessionId) {
    SdsSession session = sds.getSession(sessionId, ...);
    var deposit = new Deposit();

    SdsSessionSection clientInfo = session.getSection("clientInfo");
    deposit.setClientId(clientInfo.getAttribute("id"));

    SdsSessionSection defaultCurrency = session.getSection("defaultCurrency");
    deposit.setCurrency(defaultCurrency.getAttribute("code"));

    SdsSessionSection rates = session.getSection("interestRates");
    deposit.setRate(rates.getAttribute("depositRate"));

    // ... сохранение deposit в БД
    return deposit;
}
```

получение сессии – 1 запрос к slave

получение сессии – 2 запрос к slave

Для 1 бизнес-операции нужно несколько секций

Пример микросервиса

Spring бин

```
public Deposit createDeposit(String sessionId) {
    SdsSession session = sds.getSession(sessionId, ...);
    var deposit = new Deposit();

    SdsSessionSection clientInfo = session.getSection("clientInfo");
    deposit.setClientId(clientInfo.getAttribute("id"));

    SdsSessionSection defaultCurrency = session.getSection("defaultCurrency");
    deposit.setCurrency(defaultCurrency.getAttribute("code"));

    SdsSessionSection rates = session.getSection("interestRates");
    deposit.setRate(rates.getAttribute("depositRate"));

    // ... сохранение deposit в БД
    return deposit;
}
```

получение сессии – 1 запрос к slave

получение секции – 2 запрос к slave

получение секции – 3 запрос к slave

Для 1 бизнес-операции нужно несколько секций

Пример микросервиса

Spring бин

```
public Deposit createDeposit(String sessionId) {
    SdsSession session = sds.getSession(sessionId, ...);
    var deposit = new Deposit();

    SdsSessionSection clientInfo = session.getSection("clientInfo");
    deposit.setClientId(clientInfo.getAttribute("id"));

    SdsSessionSection defaultCurrency = session.getSection("defaultCurrency");
    deposit.setCurrency(defaultCurrency.getAttribute("code"));

    SdsSessionSection rates = session.getSection("interestRates");
    deposit.setRate(rates.getAttribute("depositRate"));

    // ... сохранение deposit в БД
    return deposit;
}
```

получение сессии – 1 запрос к slave

получение секции – 2 запрос к slave

получение секции – 3 запрос к slave

получение секции – 4 запрос к slave

Для 1 бизнес-операции нужно несколько секций

Пример микросервиса

Spring бин

```
public Deposit createDeposit(String sessionId) {
    SdsSession session = sds.getSession(sessionId, ...);
    var deposit = new Deposit();

    SdsSessionSection clientInfo = session.getSection("clientInfo");
    deposit.setClientId(clientInfo.getAttribute("id"));

    SdsSessionSection defaultCurrency = session.getSection("defaultCurrency");
    deposit.setCurrency(defaultCurrency.getAttribute("code"));

    SdsSessionSection rates = session.getSection("interestRates");
    deposit.setRate(rates.getAttribute("depositRate"));

    // ... сохранение deposit в БД
    return deposit;
}
```

получение сессии – 1 запрос к slave

получение секции – 2 запрос к slave

получение секции – 3 запрос к slave

получение секции – 4 запрос к slave

Для 1 бизнес-операции нужно несколько секций

Пример микросервиса

Spring бин

```
public Deposit createDeposit(String sessionId) {
    SdsSession session = sds.getSession(sessionId, ...);
    var deposit = new Deposit();

    SdsSessionSection clientInfo = session.getSection("clientInfo");
    deposit.setClientId(clientInfo.getAttribute("id"));

    SdsSessionSection defaultCurrency = session.getSection("defaultCurrency");
    deposit.setCurrency(defaultCurrency.getAttribute("code"));

    SdsSessionSection rates = session.getSection("interestRates");
    deposit.setRate(rates.getAttribute("depositRate"));

    // ... сохранение deposit в БД
    return deposit;
}
```

Потребитель указывает нужные ему секции

Пример микросервиса

Spring бин

```
public Deposit createDeposit(String sessionId) {
    SdsSession session = sds.getSession(sessionId, ...,
        Map.of("sections.intended.to.use",
               List.of("clientInfo", "defaultCurrency", "interestRates")));
    var deposit = new Deposit();

    SdsSessionSection clientInfo = session.getSection("clientInfo");
    //...
    SdsSessionSection defaultCurrency = session.getSection("defaultCurrency");
    //...
    SdsSessionSection rates = session.getSection("interestRates");
    // ...

    // ... сохранение deposit в БД
    return deposit;
}
```

Потребитель указывает нужные ему секции

Пример микросервиса

Spring бин

```
public Deposit createDeposit(String sessionId) {
    SdsSession session = sds.getSession(sessionId, ...,
        Map.of("sections.intended.to.use",
               List.of("clientInfo", "defaultCurrency", "interestRates")));
    var deposit = new Deposit();

    SdsSessionSection clientInfo = session.getSection("clientInfo");
    //...
    SdsSessionSection defaultCurrency = session.getSection("defaultCurrency");
    //...
    SdsSessionSection rates = session.getSection("interestRates");
    // ...

    // ... сохранение deposit в БД
    return deposit;
}
```

получение сессии и 3-х секций за 1 запрос к slave

нет запроса

нет запроса

нет запроса

Аннотация для указания нужных секций

Пример микросервиса

Контроллер

```
@RestController
@RequestMapping(path = "/v1", ...)
public class DepositController {

    @Autowired
    private DepositService deposits;

    @SdsSectionsIntendedToUse(
        url = "/v1/deposit",
        sectionNames = { "clientInfo", "defaultCurrency", "interestRates" }
    )
    @PostMapping("/deposit")
    public Deposit createDeposit(@CookieValue("SESSION-ID") String sessionId) {
        return deposits.createDeposit(sessionId);
    }
}
```

Аннотация для указания нужных секций

Пример микросервиса

Контроллер

```
@RestController
@RequestMapping(path = "/v1", ...)
public class DepositController {

    @Autowired
    private DepositService deposits;

    @SdsSectionsIntendedToUse(
        url = "/v1/deposit",
        sectionNames = { "clientInfo", "defaultCurrency", "interestRates" }
    )
    @PostMapping("/deposit")
    public Deposit createDeposit(@CookieValue("SESSION-ID") String sessionId) {
        return deposits.createDeposit(sessionId);
    }
}
```

внутри получение сессии и 3-х секций за 1 запрос к slave

Ускорение от чтения данных пачками

В несколько раз

Уменьшилось суммарное время обращений к User Sessions

Ускорение от чтения данных пачками

В несколько раз

Уменьшилось суммарное время обращений к User Sessions

Было 3,3 мс

Latency после One Nio и сохранений «дельтами»

Немного ↑

Чтение нескольких секций незначительно «тяжелее» чтения одной

Ускорение от чтения данных пачками

В несколько раз

Уменьшилось суммарное время обращений к User Sessions

Было 3,3 мс

Latency после One Nio и сохранений «дельтами»

Немного ↑

Чтение нескольких секций незначительно «тяжелее» чтения одной

В среднем стало 1 мс

На обращение к 1 секции

Это уровень Apache Ignite, Redis, Hazelcast и др.

Ускорение от чтения данных пачками

В несколько раз

Уменьшилось суммарное время обращений к User Sessions

Было 3,3 мс

Latency после One Nio и сохранений «дельтами»

Немного ↑

Чтение нескольких секций незначительно «тяжелее» чтения одной

В среднем стало 1 мс

На обращение к 1 секции

Это уровень Apache Ignite, Redis, Hazelcast и др.

Будет ещё быстрее

Предстоят другие оптимизации (например, zip -> lz4)

План

Требования к доступности в СберБанк Онлайн

99,99%

Времени микросервис должен быть доступен

52 минуты

В год – допустимый downtime

Mission critical

Сервисом является User Sessions

План

~~1. Platform V User Sessions. Начало пути~~

~~1.1. Работа с данными и их передача по сети~~

~~2. Развитие User Sessions для выдерживания highload~~

~~2.1. Сериализация данных в бинарный формат One Nio~~

~~2.2. Сохранение данных «дельтами»~~

~~2.3. Чтение данных пачками~~

3. Как User Sessions достигает high availability

3.1. Load balancing & true sticky

3.2. Асинхронная работа с базой данных

3.3. Хранилище не в контейнере

3.4. No vendor lock

3.5. Репликация данных (в процессе реализации)

4. Дальнейшее развитие User Sessions

4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД

4.2. Авторизация доступа к данным через Open Policy Agent

4.3. «Не клиентские» сессии

От чего отталкиваемся

6,5 лет

Наработка на отказ у современных серверов

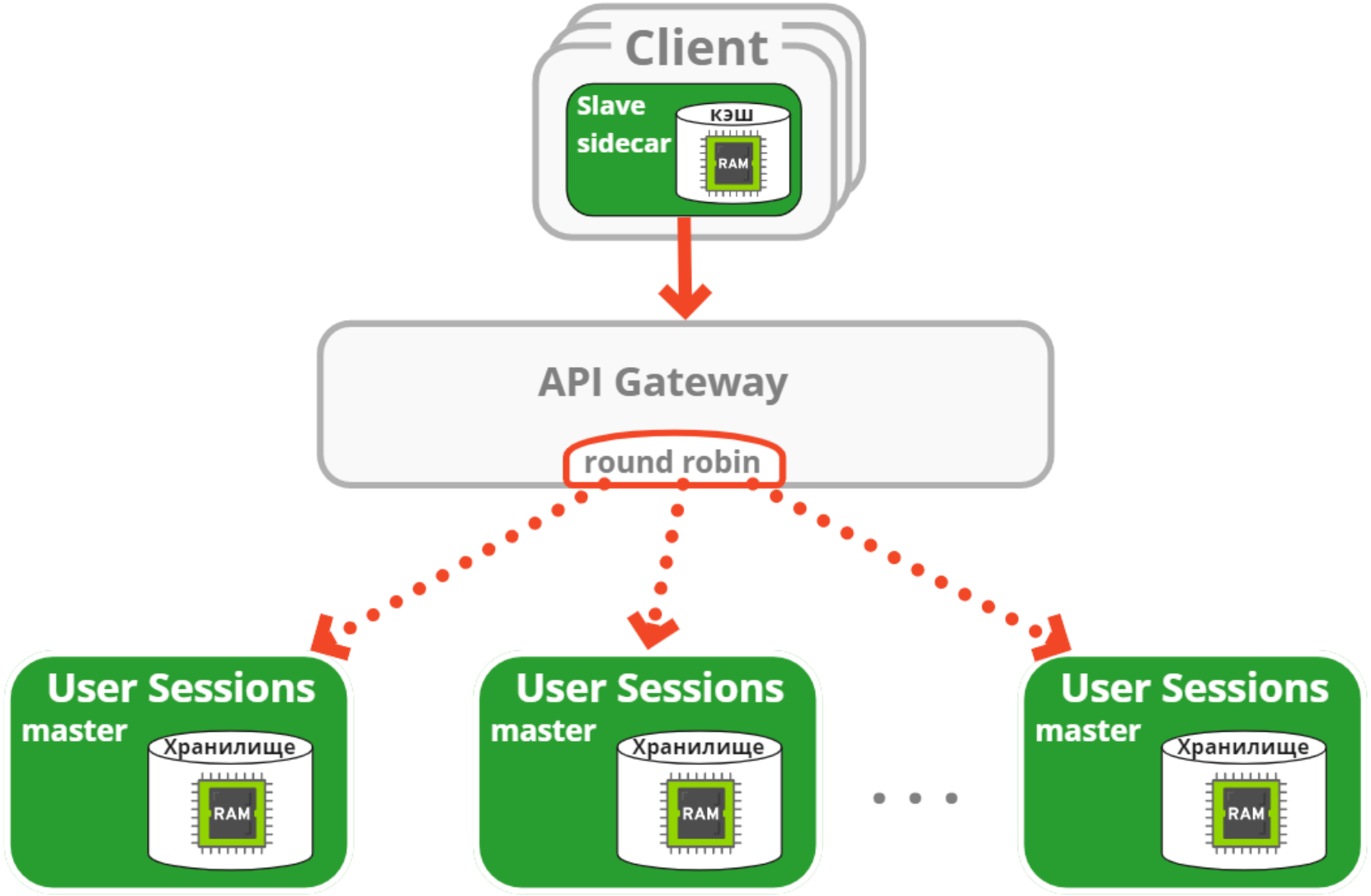
1 сутки

Оперативная замена сервера

99,96%

Получаемый уровень доступности

Много серверов и load balancing

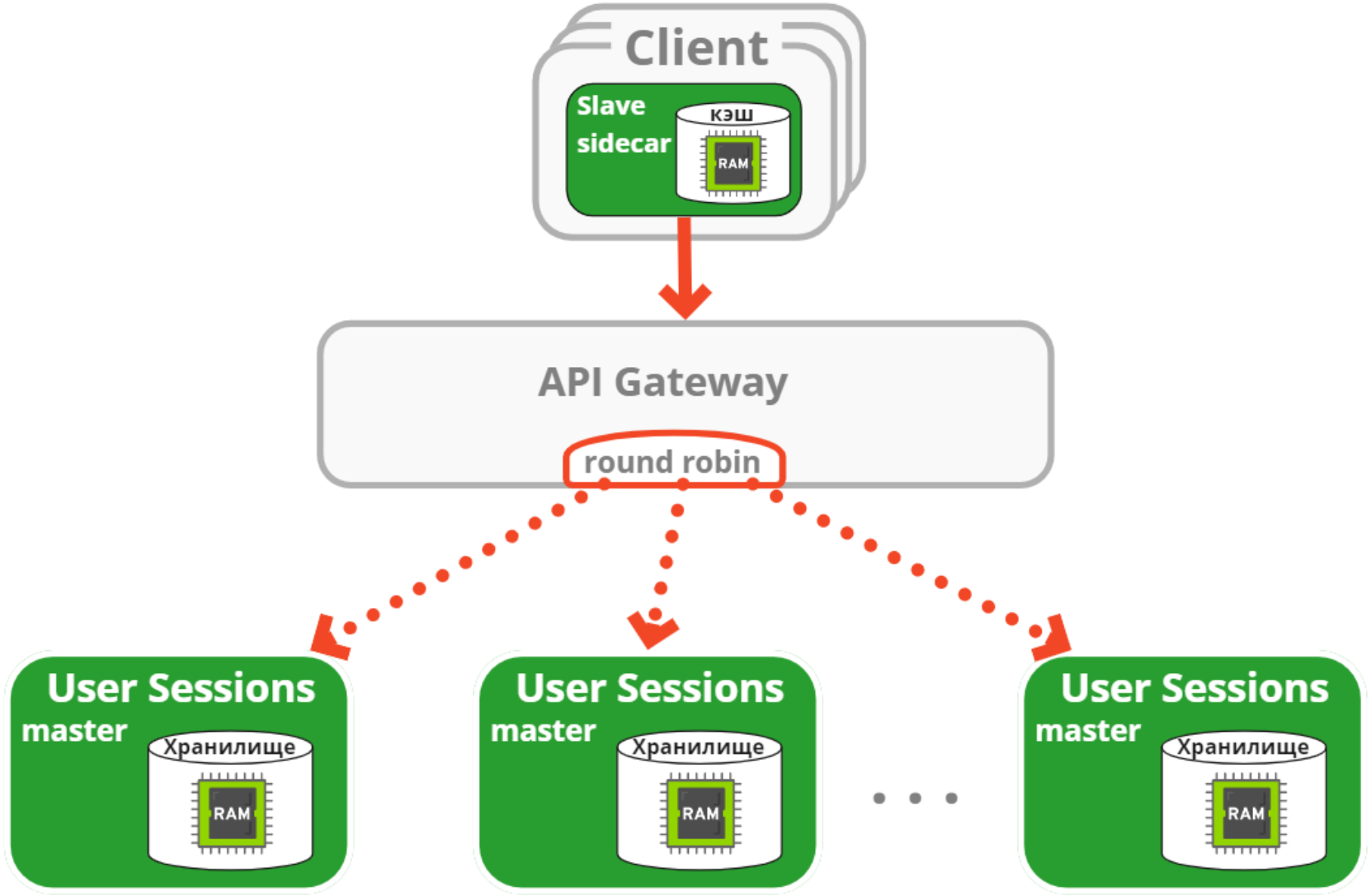


Много серверов и load balancing

Шлюз на базе Nginx

из Platform V Synapse API Management

https://platformv.sber.ru/products/synapse-api-management

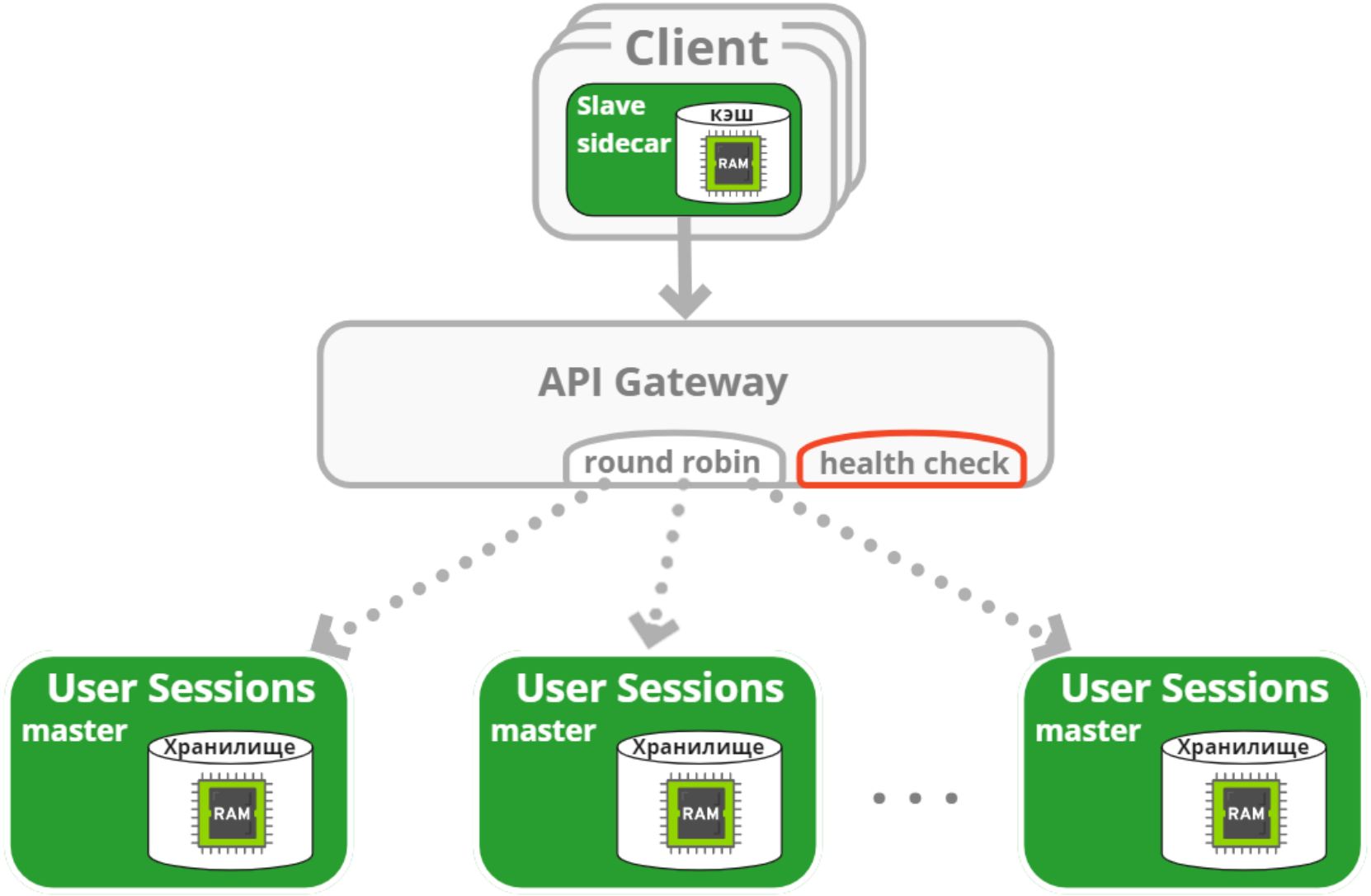


Active health check узлов master-а

Active health check

из Nginx доработан в Platform V:

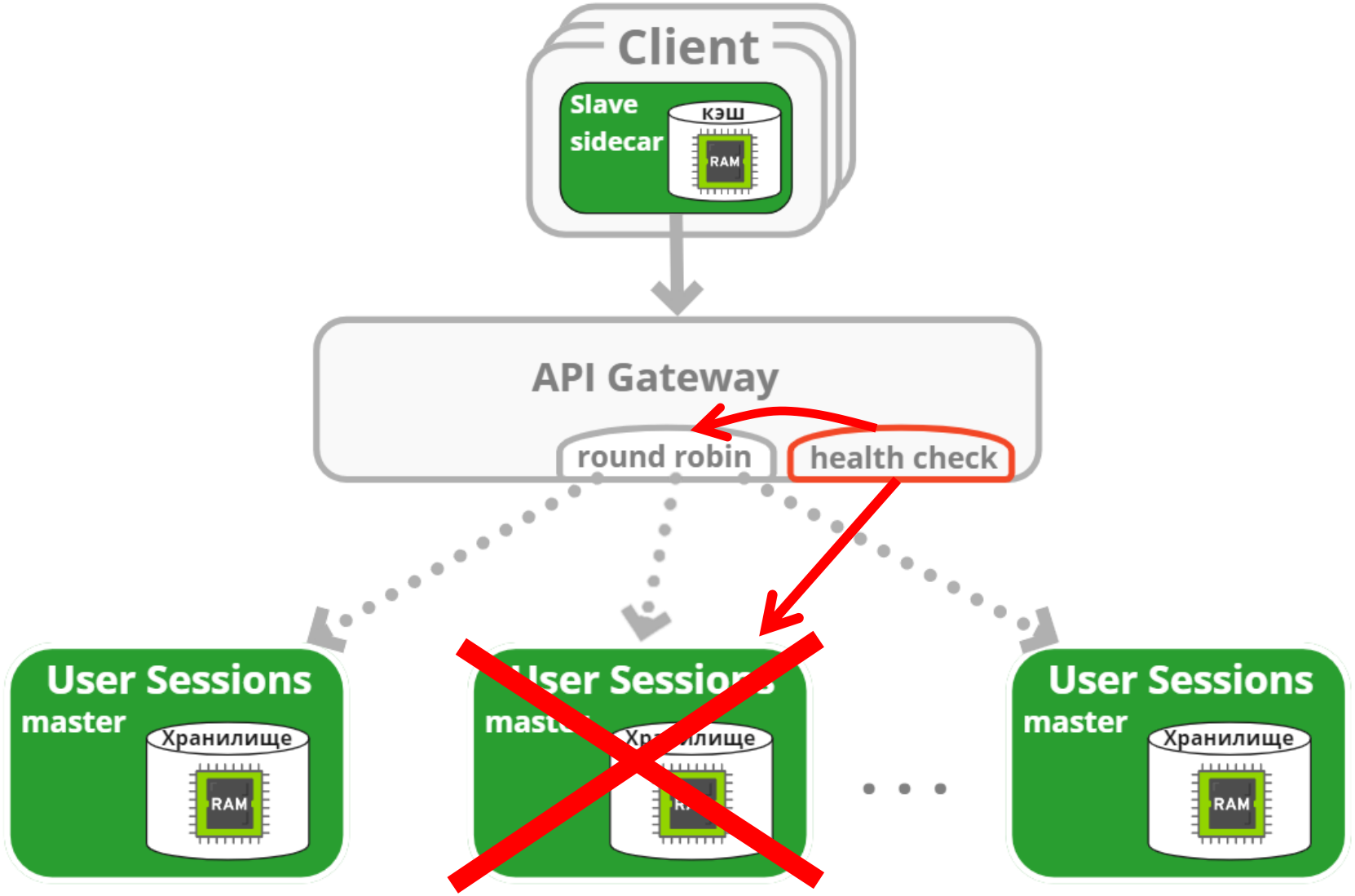
- Используется mTLS
- Добавлены метрики
- и др.



Active health check узлов master-а

Active health check

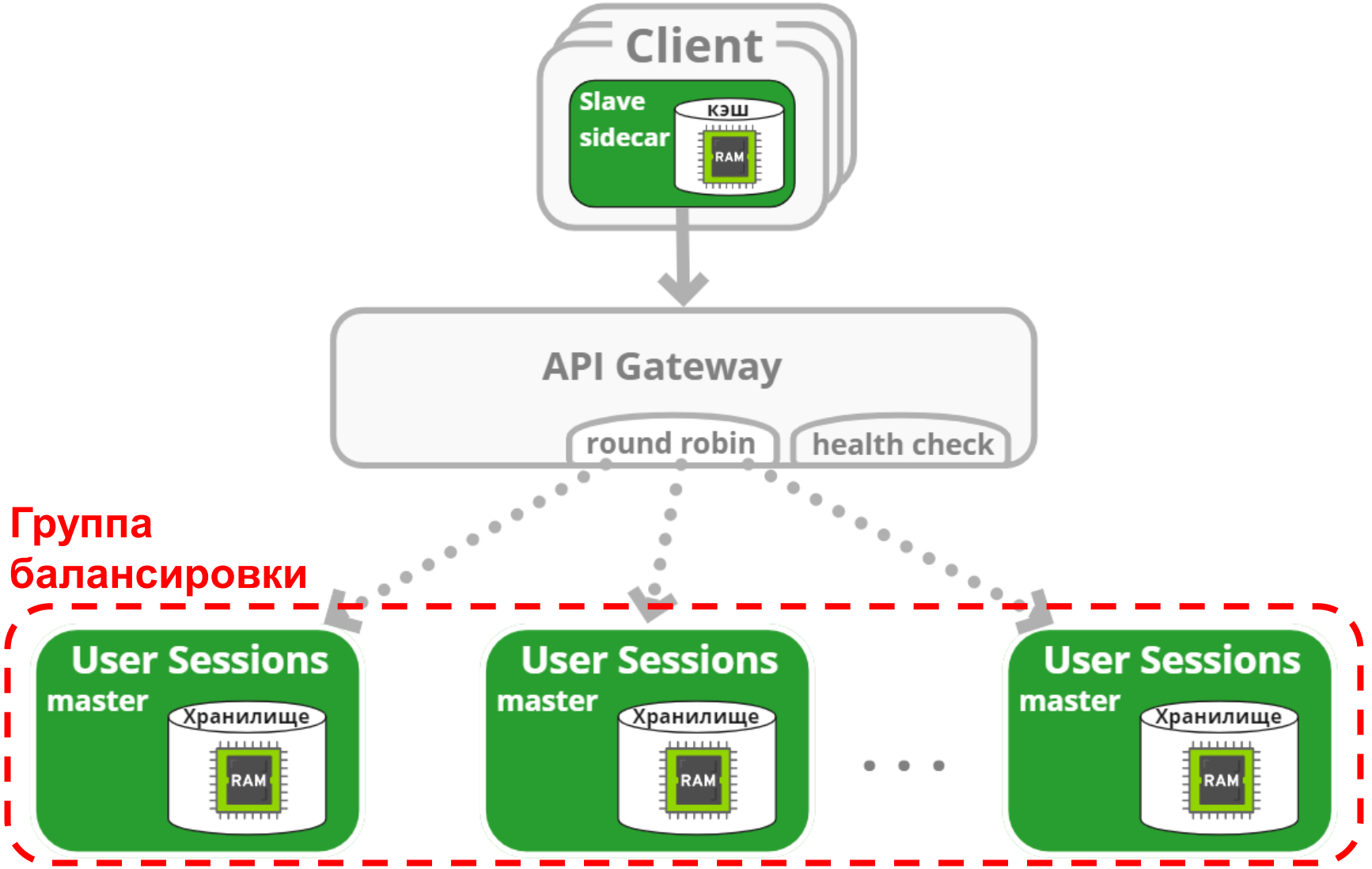
работает, как readiness probe в Kubernetes



Размер группы балансировки

24 узла master-а

в одной группе балансировки в СберБанк Онлайн



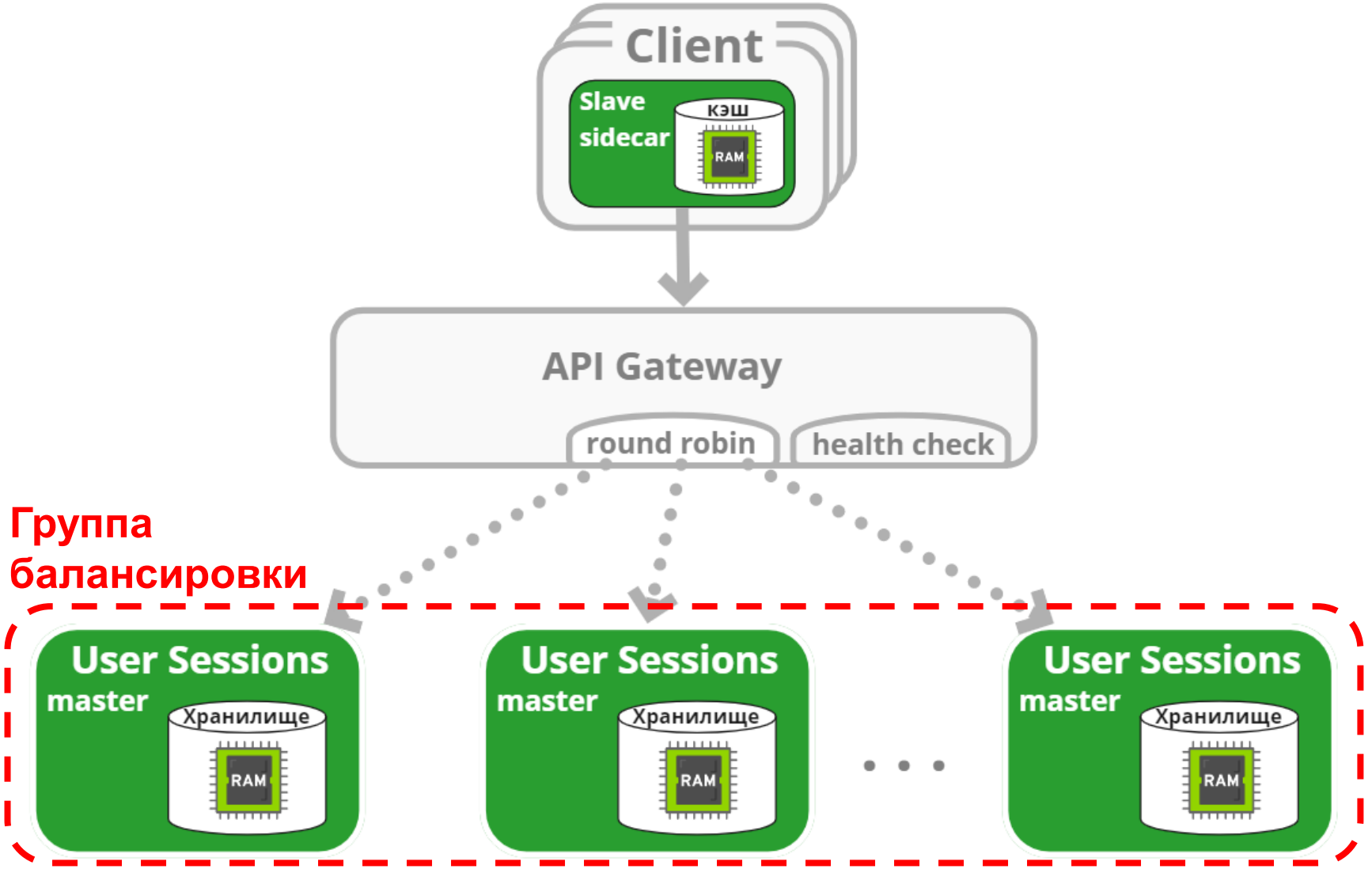
Размер группы балансировки

24 узла master-а

в одной группе балансировки в СберБанк Онлайн

Соблюдаются 9999

ничтожна вероятность краха 24-х серверов одновременно



Не всё так просто

Stateful by design

Данные хранятся в RAM

Всё равно

На какой master придёт запрос **только при создании сессии**

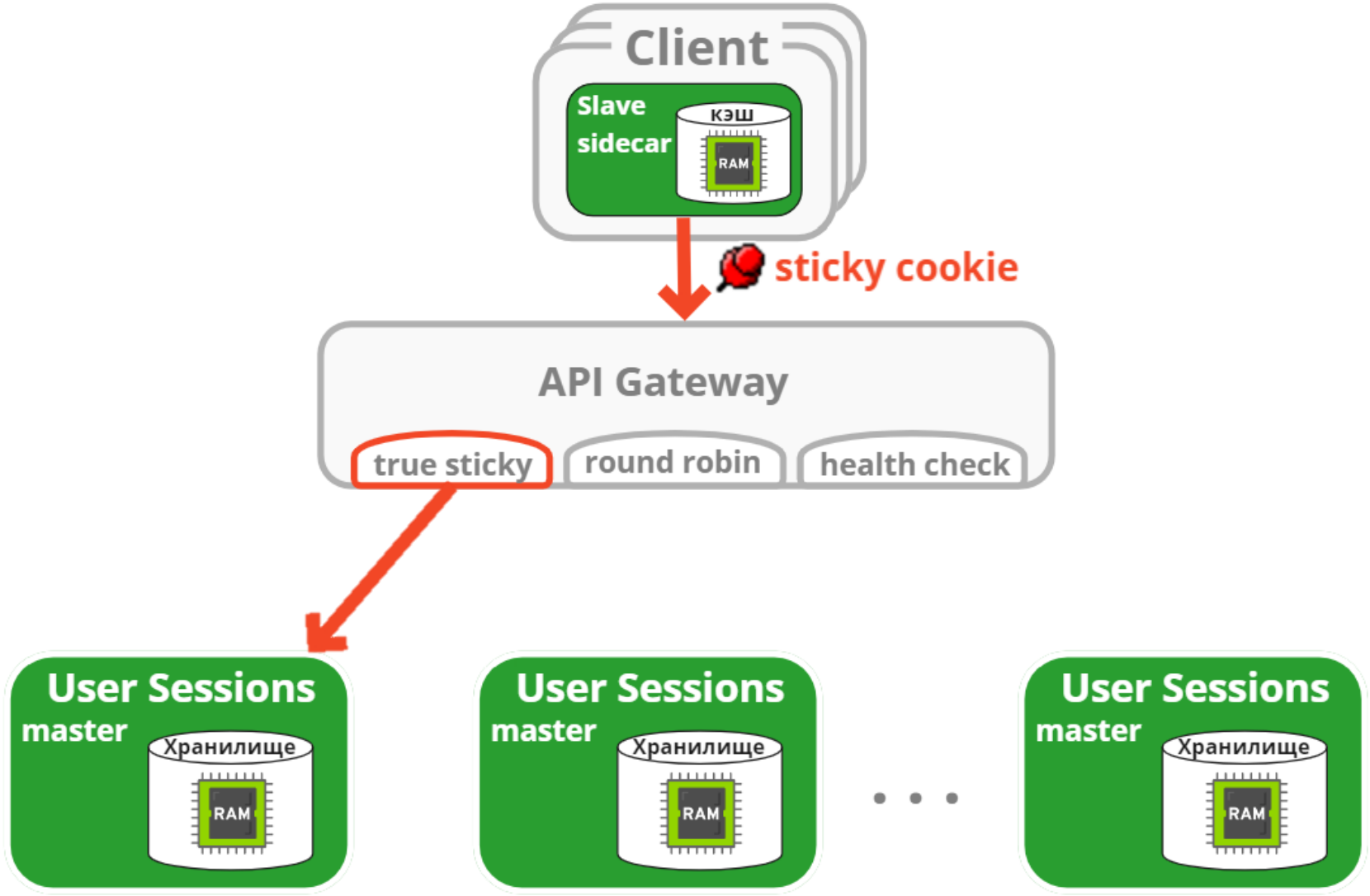
Load balancing 👍

Не всё равно

Куда придёт запрос по уже созданной сессии

Load balancing 👎

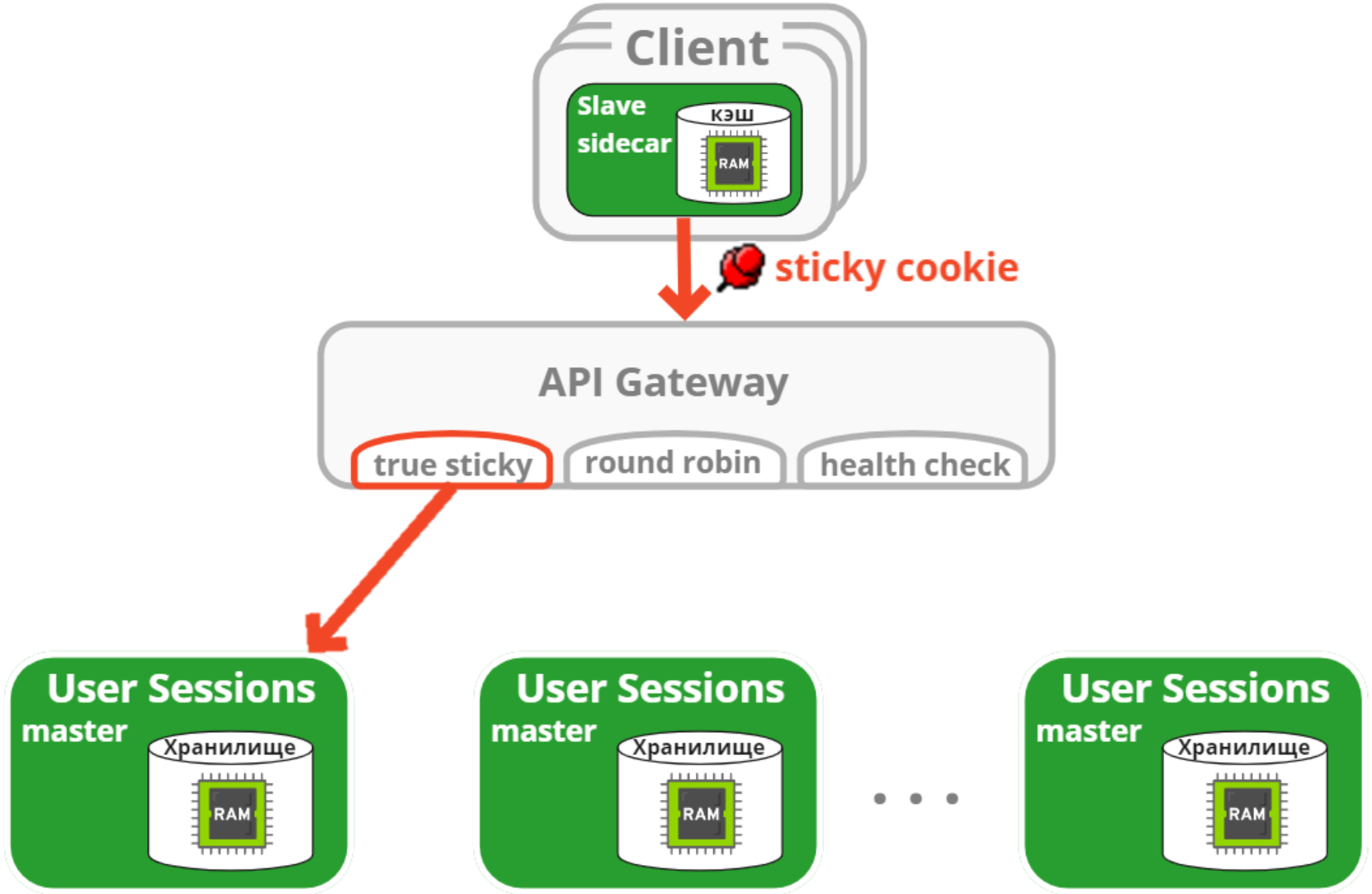
True sticky к узлам master-а



True sticky к узлам master-а

True sticky

не ломает прилипания при изменении группы балансировки



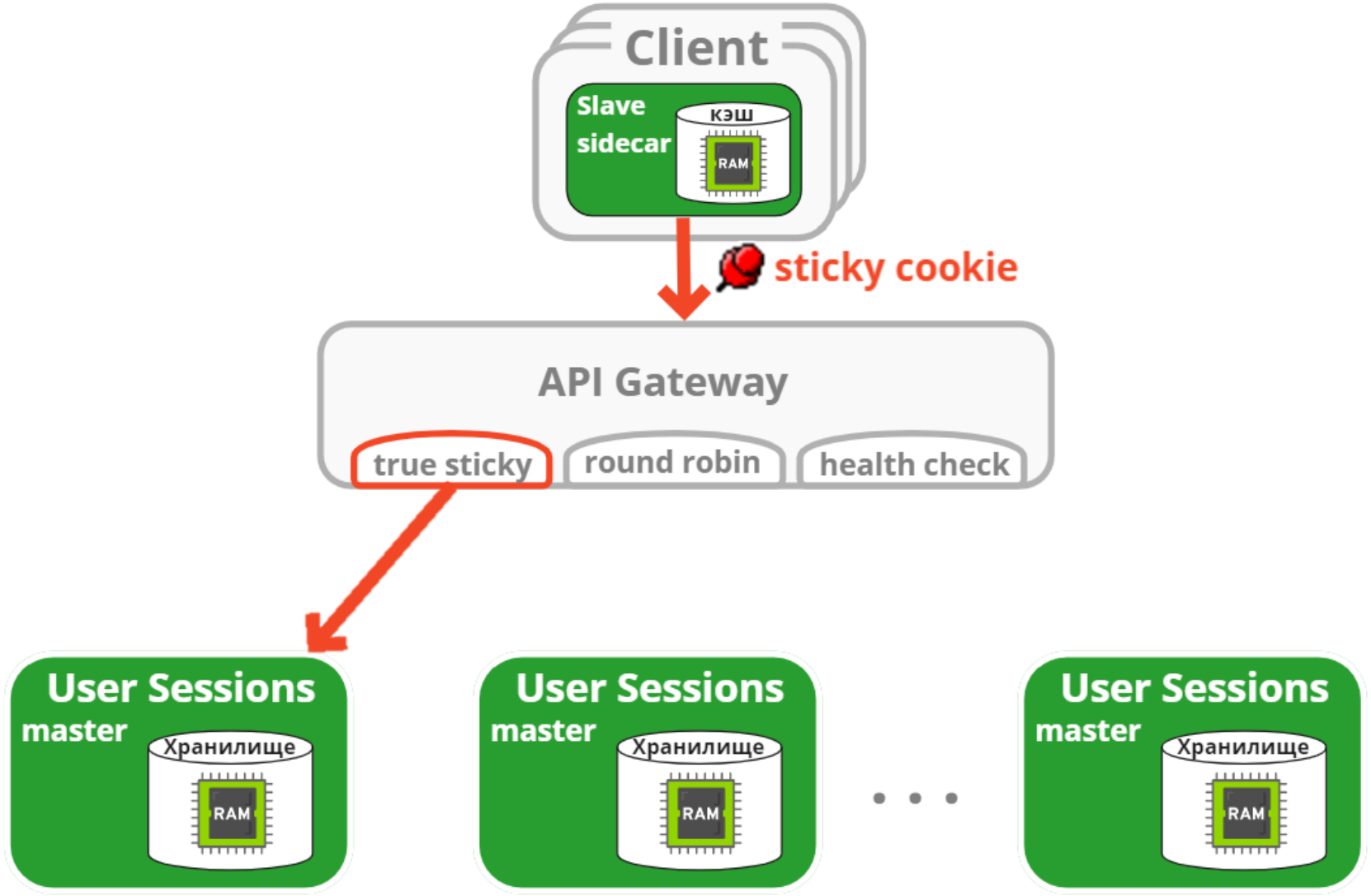
True sticky к узлам master-а

True sticky

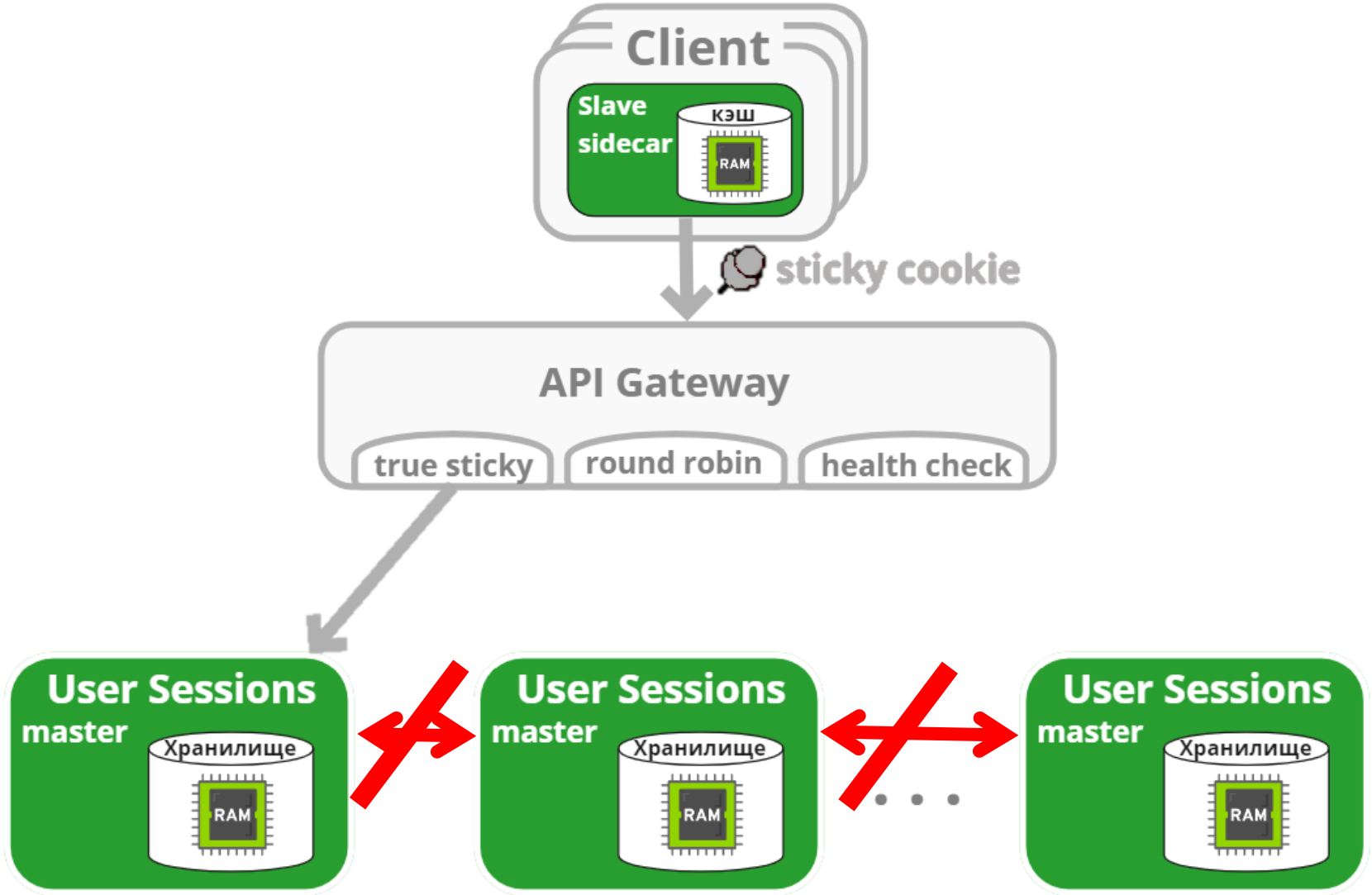
не ломает прилипания при изменении группы балансировки

Позволяет масштабировать

master-а «на горячую»



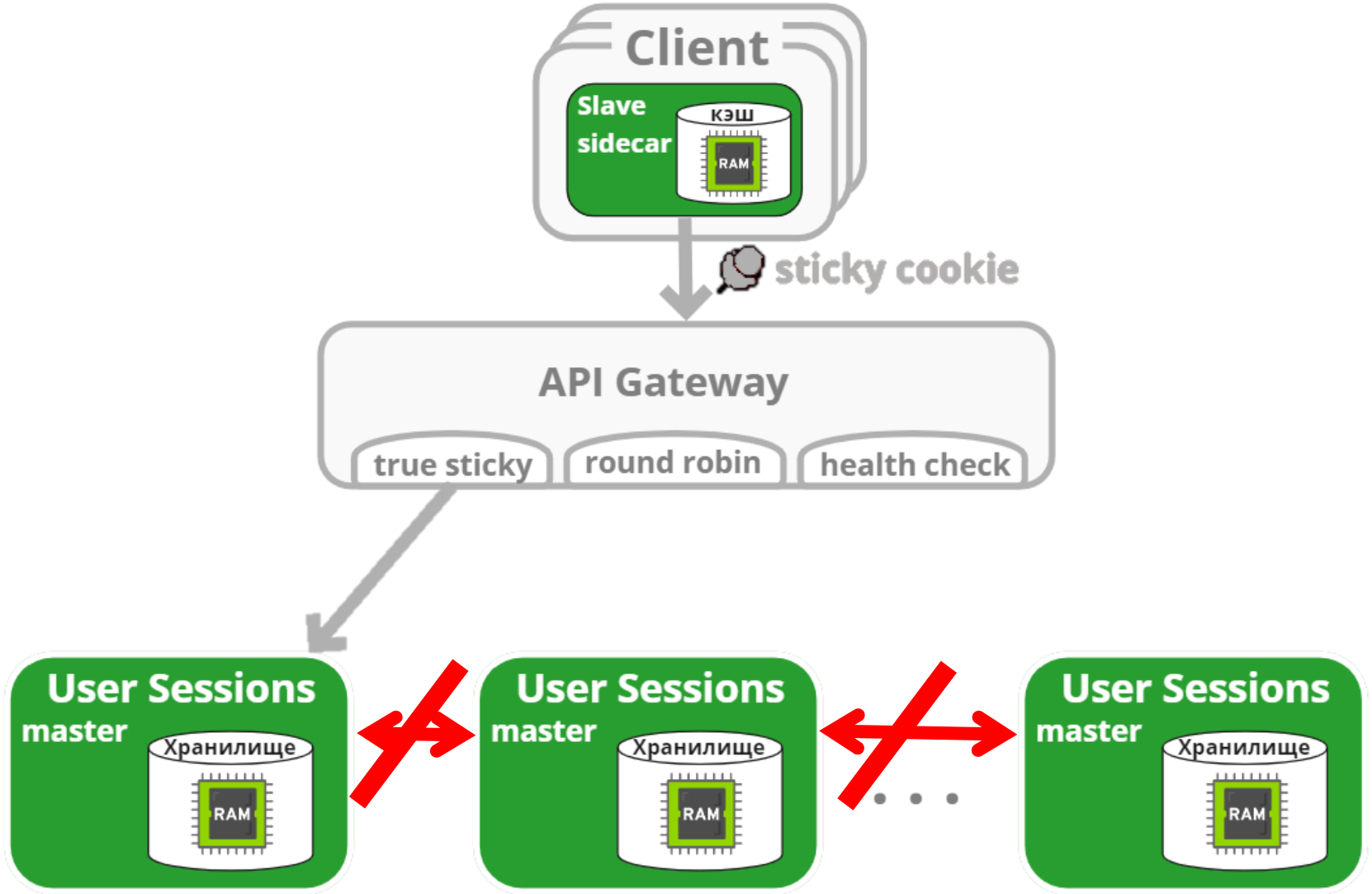
Нет горизонтальных связей



Нет горизонтальных связей

Предсказуемость

latency – важное достоинство



План

~~1. Platform V User Sessions. Начало пути~~

- ~~1.1. Работа с данными и их передача по сети~~

~~2. Развитие User Sessions для выдерживания highload~~

- ~~2.1. Сериализация данных в бинарный формат One Nio~~
- ~~2.2. Сохранение данных «дельтами»~~
- ~~2.3. Чтение данных пачками~~

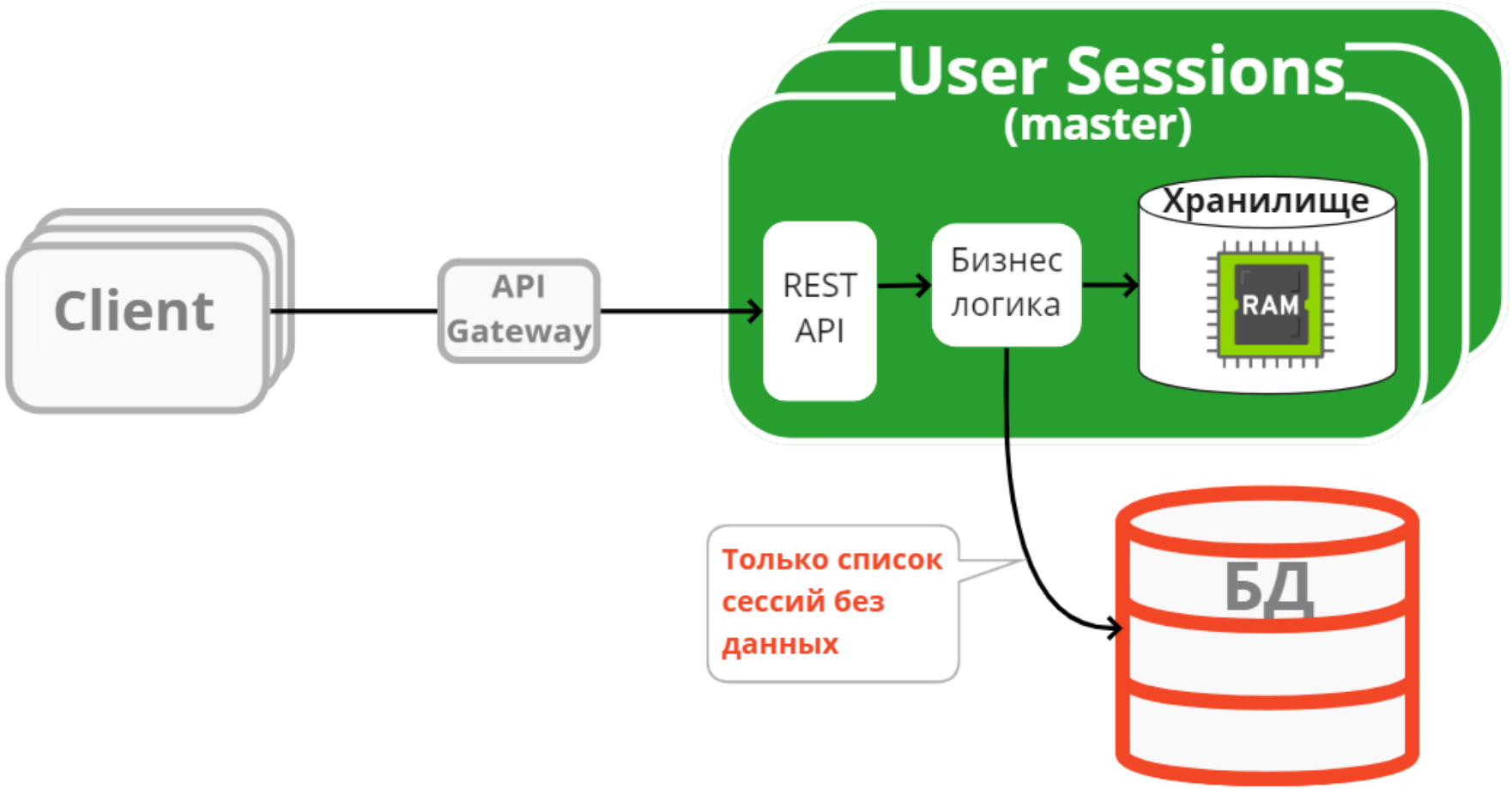
3. Как User Sessions достигает high availability

- ~~3.1. Load balancing & true sticky~~
- 3.2. Асинхронная работа с базой данных
- 3.3. Хранилище не в контейнере
- 3.4. No vendor lock
- 3.5. Репликация данных (в процессе реализации)

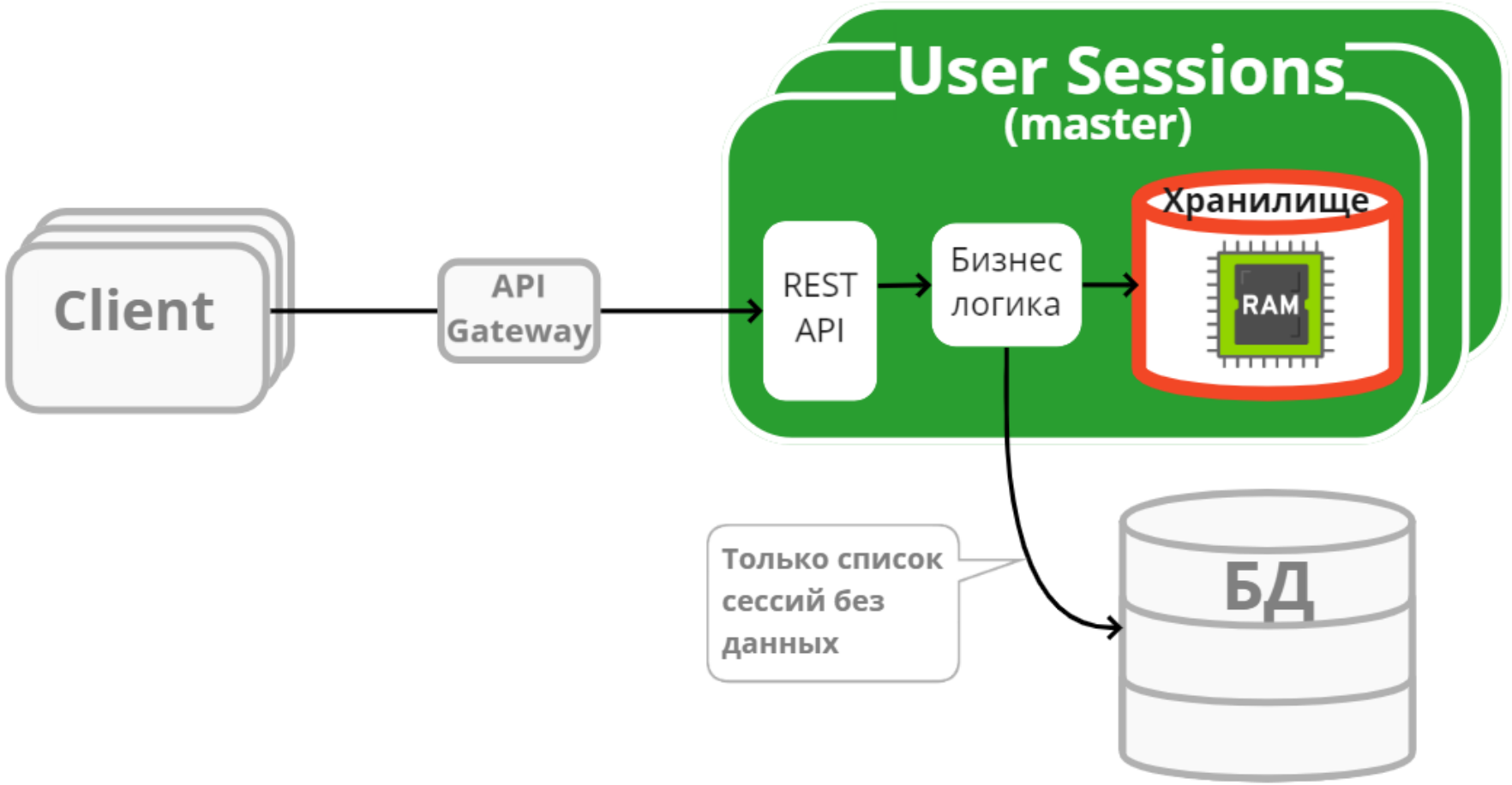
4. Дальнейшее развитие User Sessions

- 4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД
- 4.2. Авторизация доступа к данным через Open Policy Agent
- 4.3. «Не клиентские» сессии

Список сессий в базе данных



Данные сессий в памяти master-ов



База данных для администрирования сервиса

UI консоль

Администратор видит активные сессии

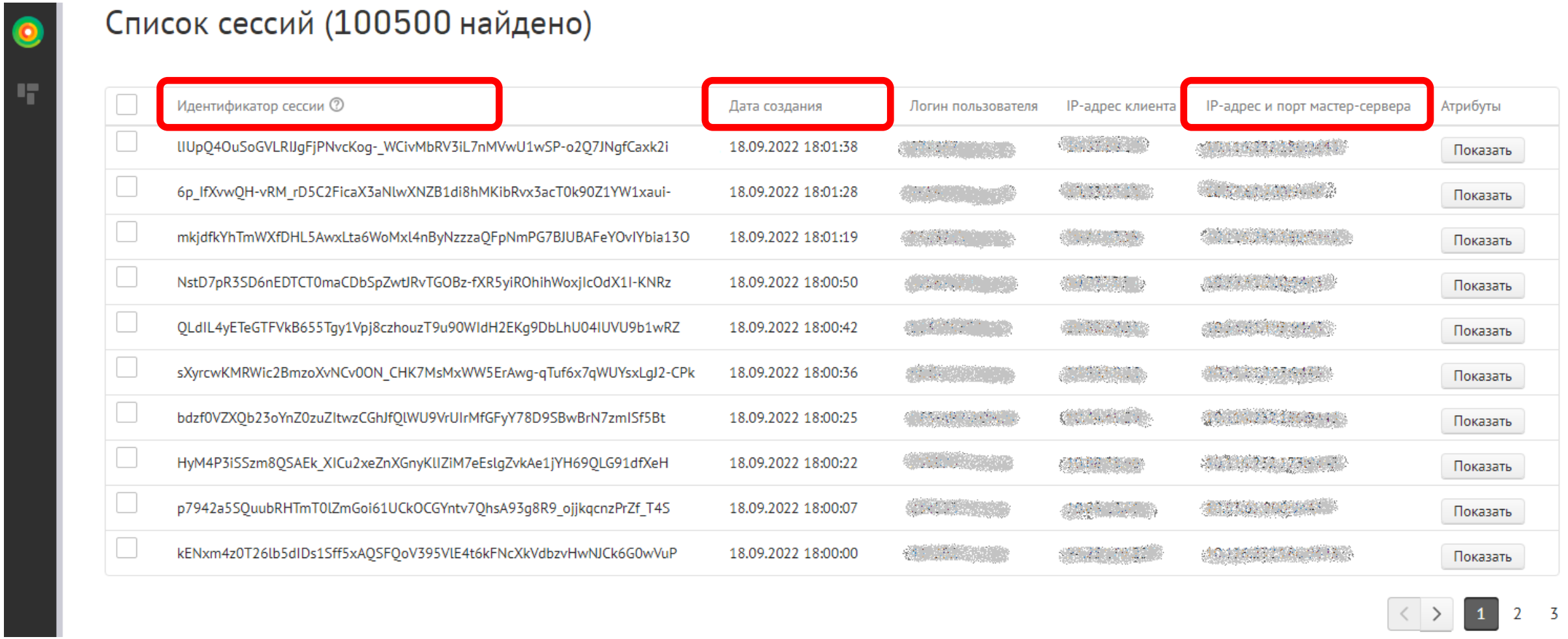
Фильтры и поиск

Гибкие фильтры сессий и поиск по ID

Можно удалять

сессии уполномоченным сотрудникам

Хранимая в базе данных информация

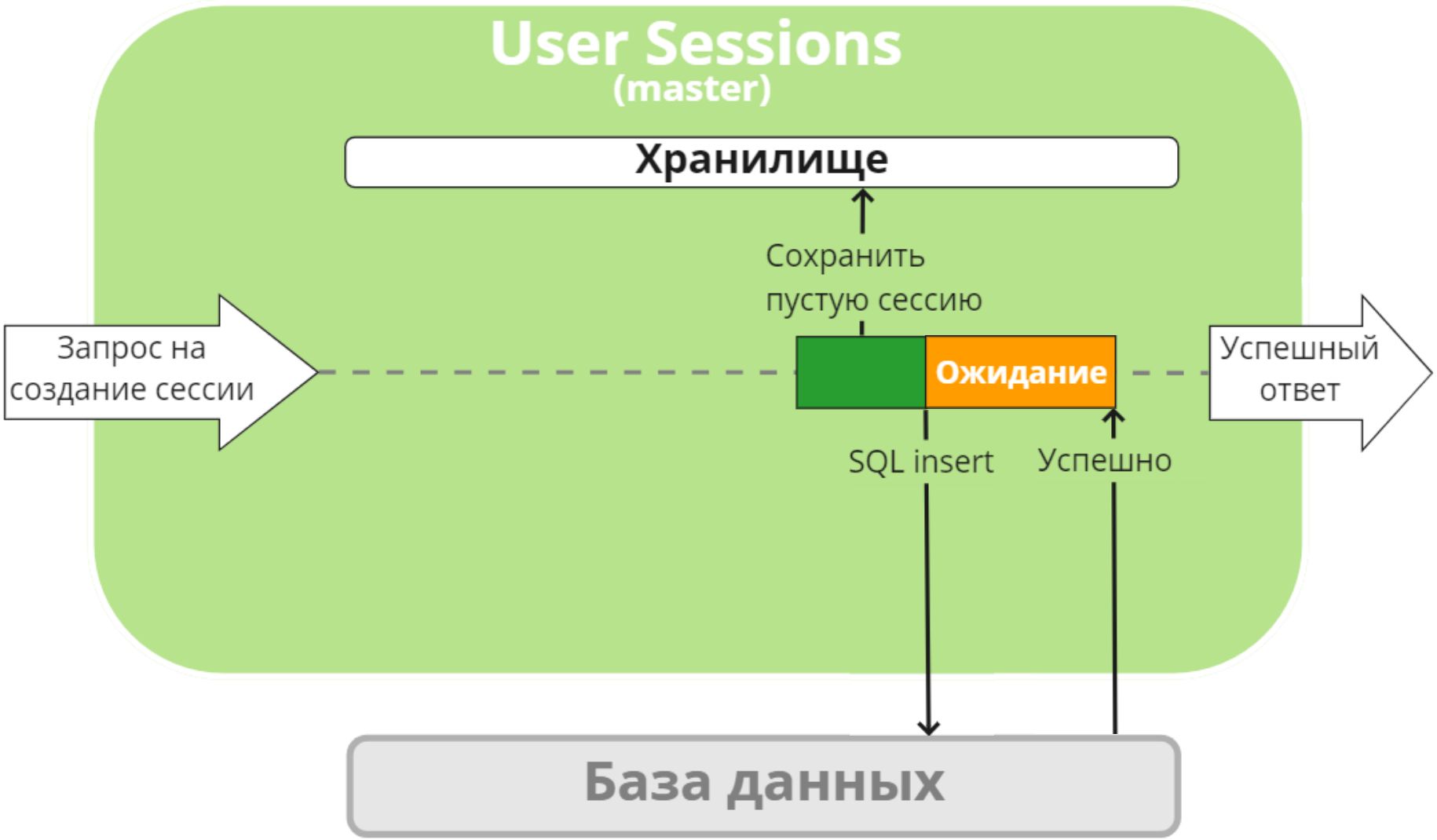


Список сессий (100500 найдено)

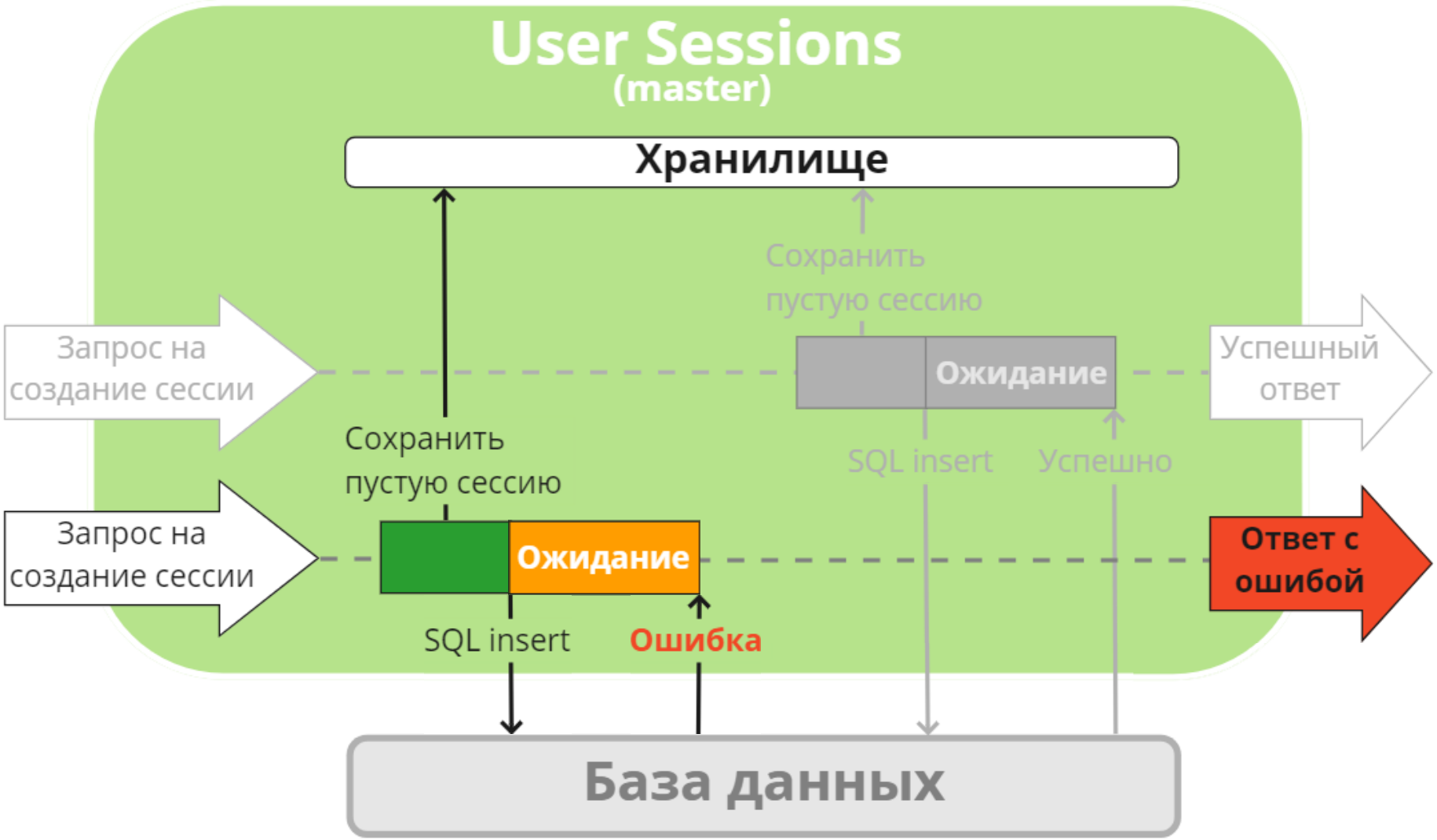
Идентификатор сессии	Дата создания	Логин пользователя	IP-адрес клиента	IP-адрес и порт мастер-сервера	Атрибуты
lIUpQ4OuSoGVLRIJgFjPNvcKog-_WCivMbRV3iL7nMVwU1wSP-o2Q7JNgfCaxk2i	18.09.2022 18:01:38				Показать
6p_IfXvwQH-vRM_rD5C2FicaX3aNlwXNZB1di8hMKibRvx3acT0k90Z1YW1xaui-	18.09.2022 18:01:28				Показать
mkjdfkYhTmWXfDHL5AwxLta6WoMxl4nByNzzzaQFpNmPG7BJUBAFeYOvIYbia13O	18.09.2022 18:01:19				Показать
NstD7pR3SD6nEDTCT0maCDbSpZwtJRvTGOBz-fXR5yiROhihWoxjIcOdX1I-KNRz	18.09.2022 18:00:50				Показать
QLdIL4yETeGTFVkB655Tgy1Vpj8czhouzT9u90WIdH2EKg9DbLhU04IUVU9b1wRZ	18.09.2022 18:00:42				Показать
sXyrcwKMRWic2BmzoXvNCv0ON_CHK7MsMxWW5ErAwg-qTuf6x7qWUYsxLgJ2-CPk	18.09.2022 18:00:36				Показать
bdzf0VZXQb23oYnZ0zuZltwzCGhJfQlWU9VrUlrMfGFyY78D9SBwBrN7zmISf5Bt	18.09.2022 18:00:25				Показать
HyM4P3iSSzm8QSAEk_XlCu2xeZnXGnyKlIZiM7eEslgZvkAe1jYH69QLG91dfXeH	18.09.2022 18:00:22				Показать
p7942a5SQuubRHTmT0lZmGoi61UCkOCGYntv7QhsA93g8R9_ojjkqcnzPrZf_T4S	18.09.2022 18:00:07				Показать
kENxm4z0T26lb5dIDs1Sff5xAQSFQoV395VlE4t6kFNcXkVdbzvHwNJCk6G0wVuP	18.09.2022 18:00:00				Показать

1 2 3

Синхронная вставка в базу данных



Синхронная вставка в базу данных



User Sessions может работать при недоступной БД

Асинхронно

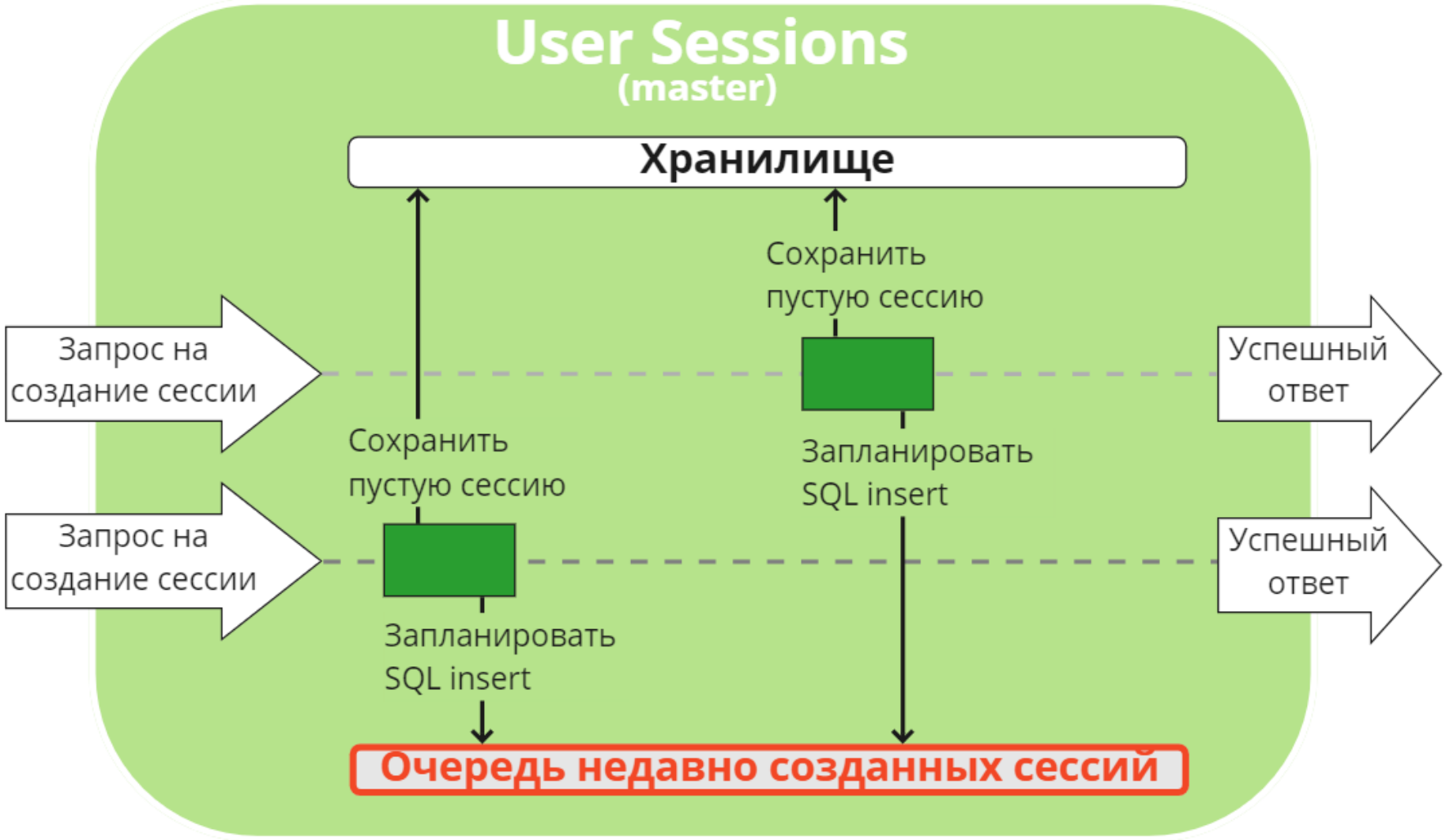
пачками теперь вставляются сессии в БД

Если БД отказала

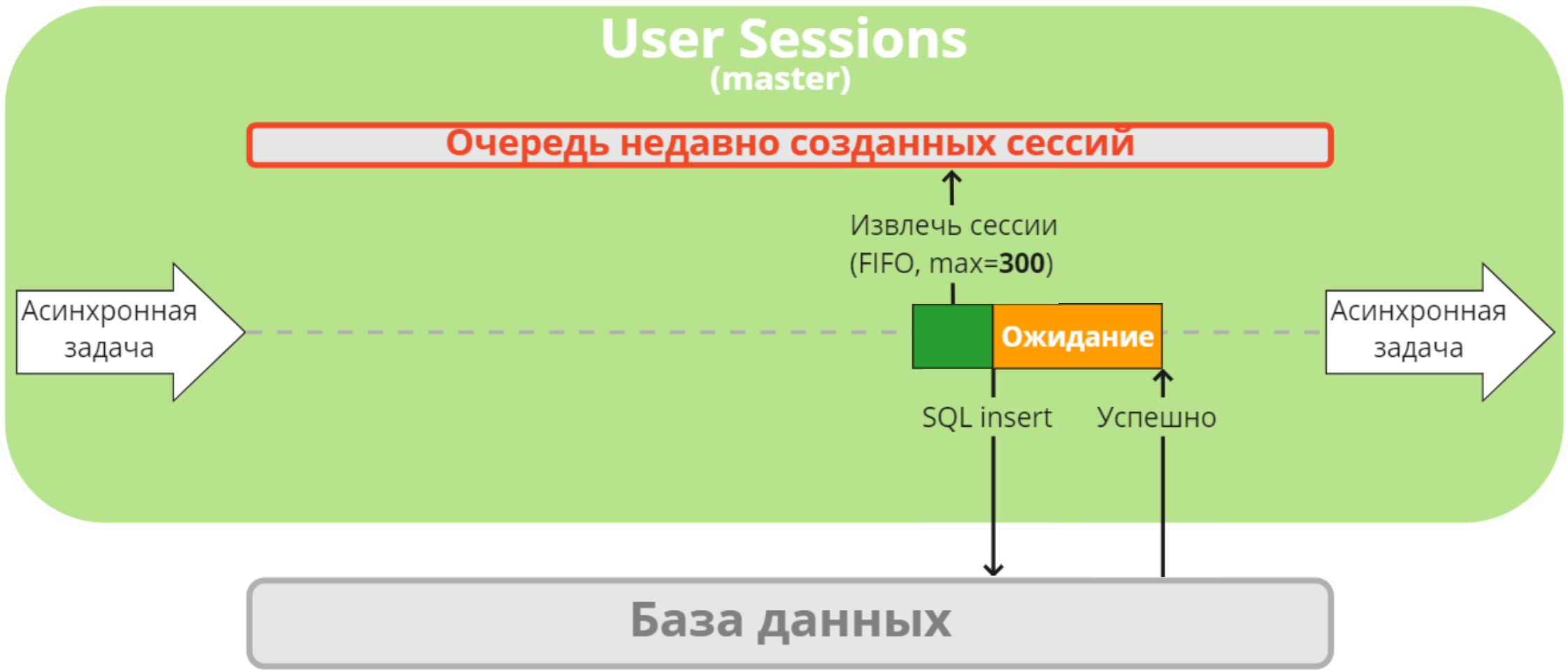
- Обслуживание потребителей не прекращается
- Сессии не видны в админке

Меньшее зло, чем downtime

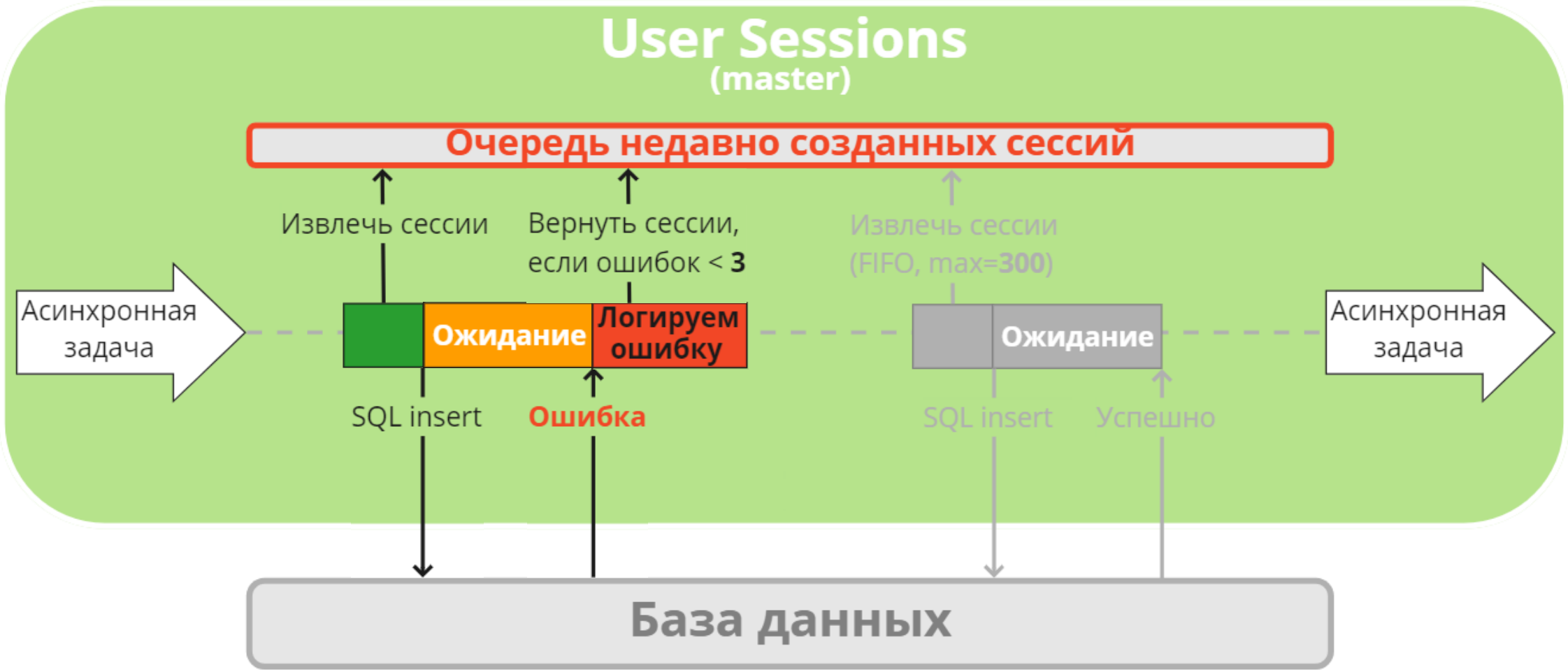
Асинхронная пакетная вставка в базу данных



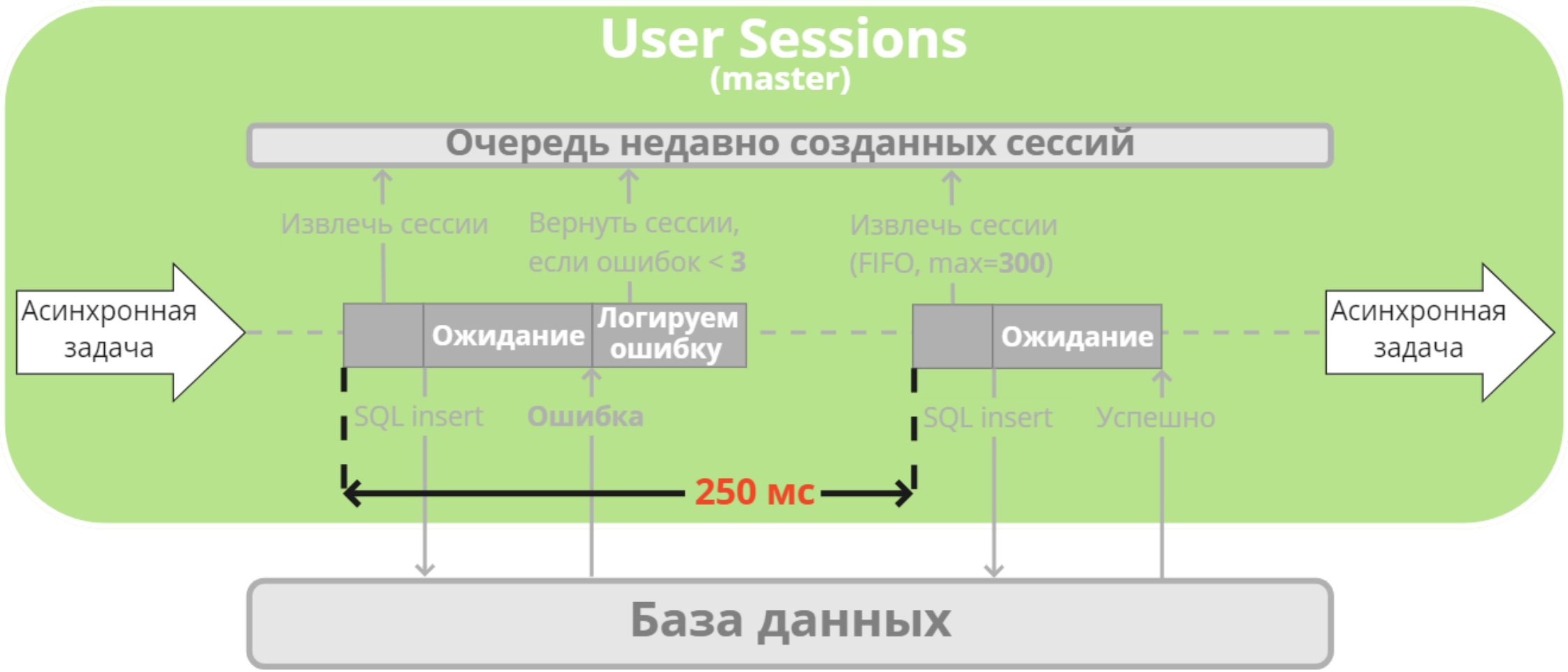
Асинхронная пакетная вставка в базу данных



Асинхронная пакетная вставка в базу данных



Асинхронная пакетная вставка в базу данных



Асинхронное пакетное удаление из базы данных

Аналогично вставке

пачками удаляются сессии из БД

Отдельная очередь

используется для планирования SQL delete

Убили двух зайцев

1

Ускорили создание/удаление сессий

2

Перестали зависеть от отказов БД

План

~~1. Platform V User Sessions. Начало пути~~

- ~~1.1. Работа с данными и их передача по сети~~

~~2. Развитие User Sessions для выдерживания highload~~

- ~~2.1. Сериализация данных в бинарный формат One Nio~~
- ~~2.2. Сохранение данных «дельтами»~~
- ~~2.3. Чтение данных пачками~~

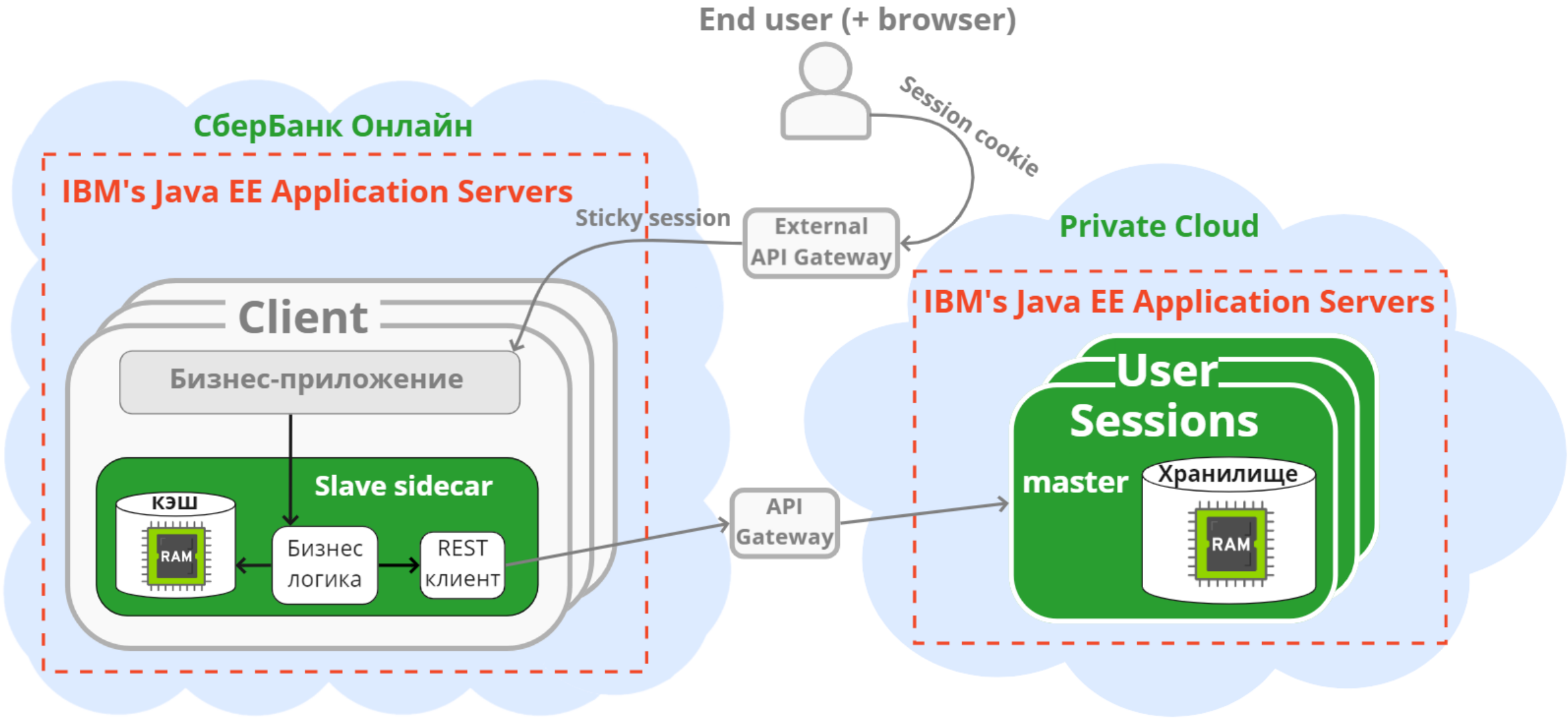
3. Как User Sessions достигает high availability

- ~~3.1. Load balancing & true sticky~~
- ~~3.2. Асинхронная работа с базой данных~~
- 3.3. Хранилище не в контейнере
- 3.4. No vendor lock
- 3.5. Репликация данных (в процессе реализации)

4. Дальнейшее развитие User Sessions

- 4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД
- 4.2. Авторизация доступа к данным через Open Policy Agent
- 4.3. «Не клиентские» сессии

IBM’s Java EE Application Servers



Потребители пошли в облако

Вызов 1

Микросервисы в СберБанк Онлайн стали развёртываться в контейнерах

Потребители пошли в облако

Вызов 1

Микросервисы в СберБанк Онлайн стали развёртываться в контейнерах

Вызов 2

Появился новый крупный потребитель с сервисами в контейнерах

Другой оркестратор

Не всё так просто

Мы stateful

Нам “противопоказан” рестарт узлов с master-хранилищем

Не всё так просто

Мы stateful

Нам “противопоказан” рестарт узлов с master-хранилищем

Не готовы на

- autoscaling
- вытестение с узла из-за развёртывания другого сервиса
- и многое другое

Не всё так просто

Мы stateful

Нам “противопоказан” рестарт узлов с master-хранилищем

Не готовы на

- autoscaling
- вытестение с узла из-за развёртывания другого сервиса
- и многое другое

Контейнеры для stateless

Не рекомендуется развёртывать stateful сервисы в production

Решение – хранилище не в контейнере

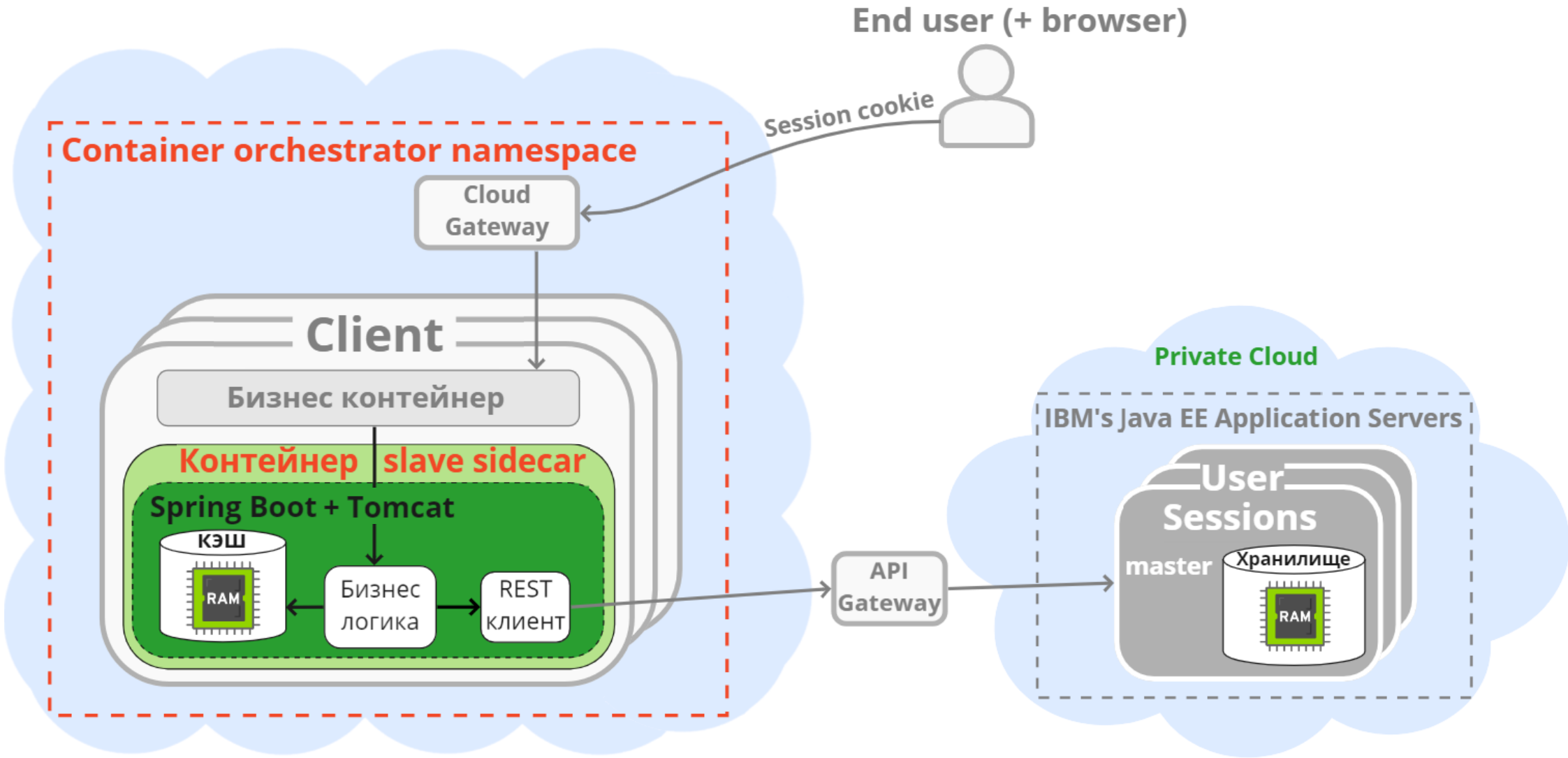
Master не в контейнере

Остаётся на IBM’s Java EE Application Servers

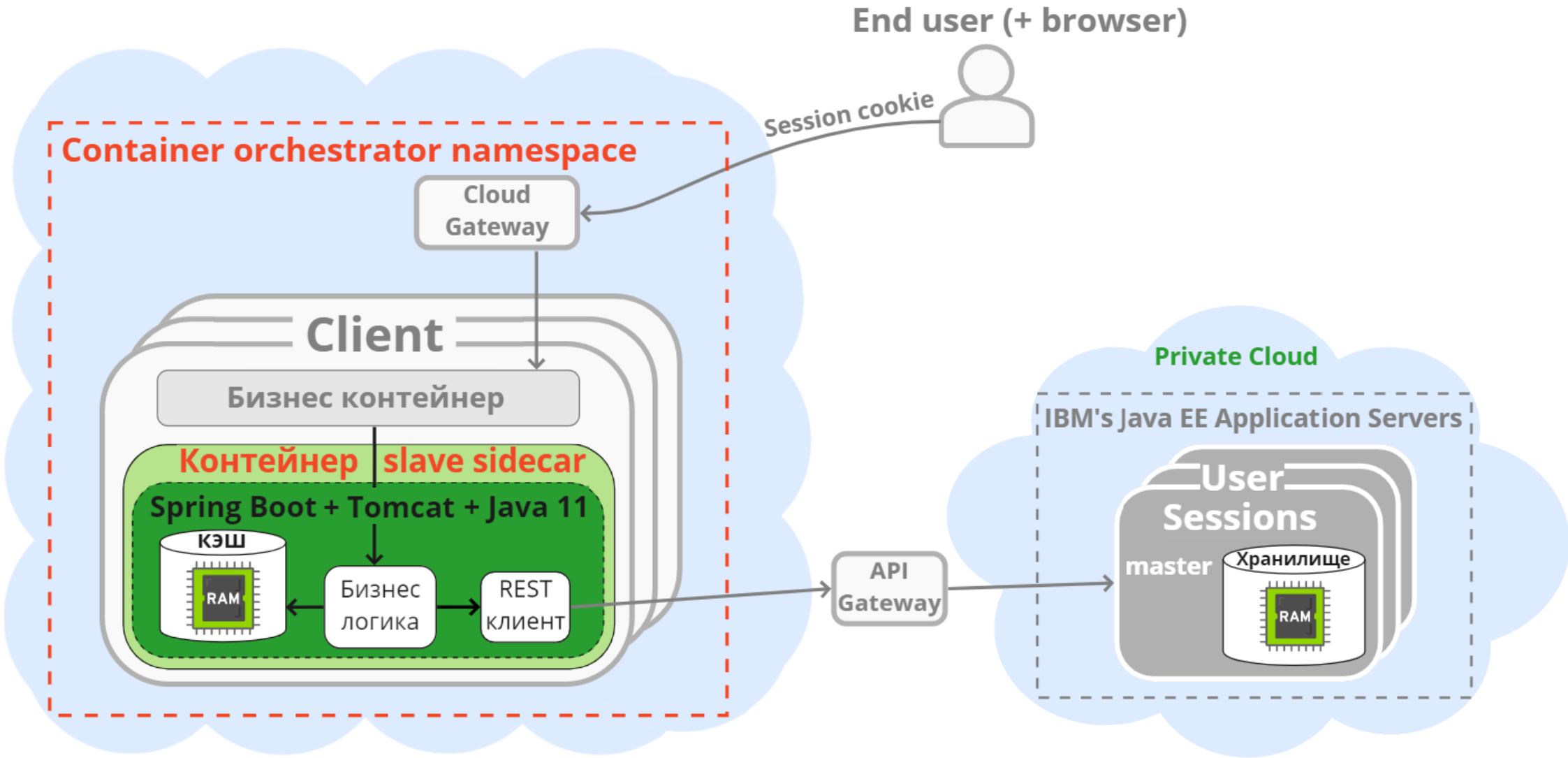
Клиенты в контейнере

Потребовался ряд доработок User Sessions

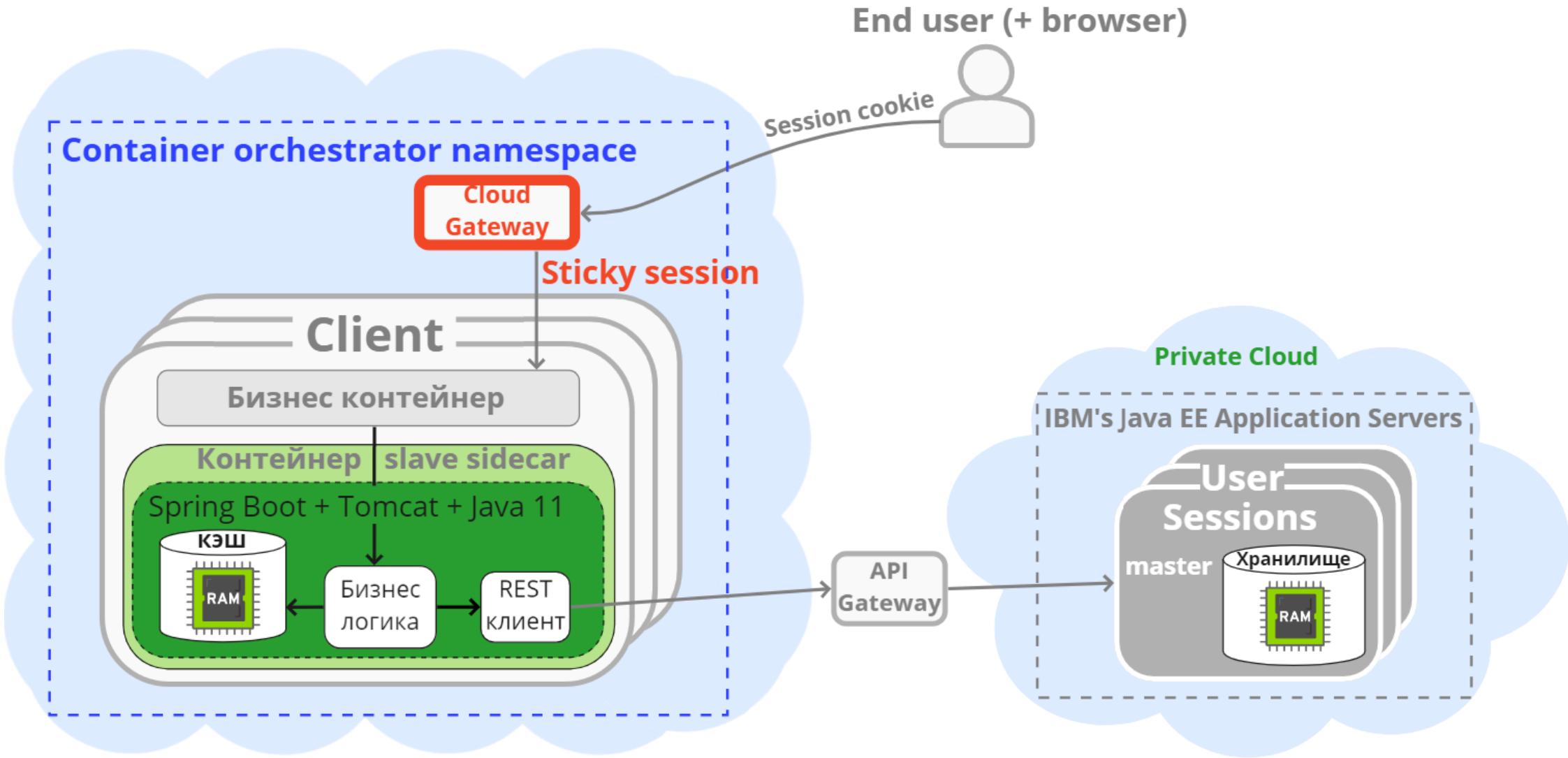
Slave sidecar в контейнере



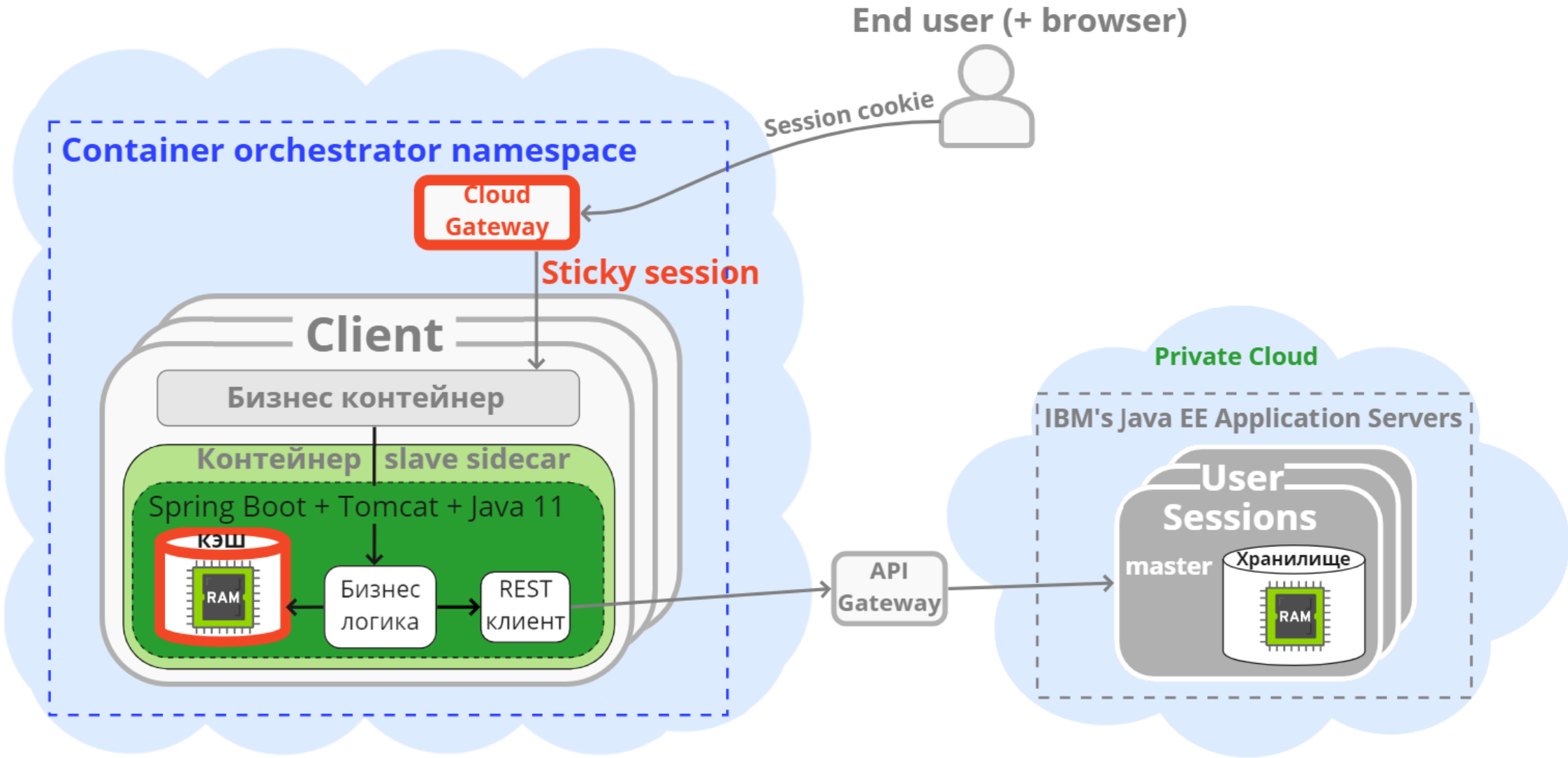
Slave sidecar в контейнере



Sticky session в облаке



Sticky session в облаке



Поддержали облачных потребителей

Клиенты в контейнере

Потребовался ряд доработок:

- slave на Spring Boot
- переход на Java 11
- sticky session на Cloud Gateway

Доступность на высоте

Master-хранилище осталось на IBM’s Java EE Application Servers

План

~~1. Platform V User Sessions. Начало пути~~

- ~~1.1. Работа с данными и их передача по сети~~

~~2. Развитие User Sessions для выдерживания highload~~

- ~~2.1. Сериализация данных в бинарный формат One Nio~~
- ~~2.2. Сохранение данных «дельтами»~~
- ~~2.3. Чтение данных пачками~~

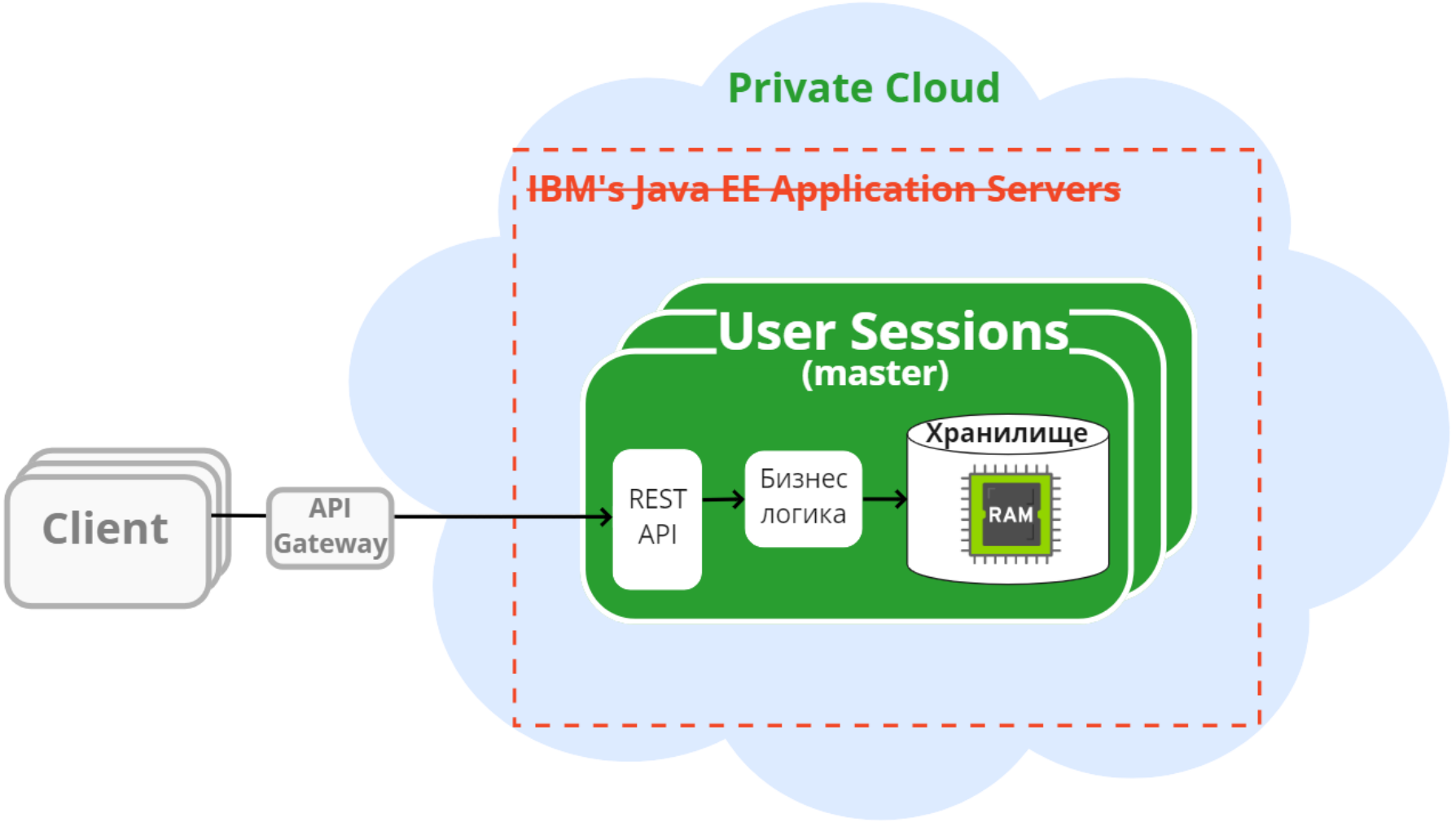
3. Как User Sessions достигает high availability

- ~~3.1. Load balancing & true sticky~~
- ~~3.2. Асинхронная работа с базой данных~~
- ~~3.3. Хранилище не в контейнере~~
- 3.4. No vendor lock
- 3.5. Репликация данных (в процессе реализации)

4. Дальнейшее развитие User Sessions

- 4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД
- 4.2. Авторизация доступа к данным через Open Policy Agent
- 4.3. «Не клиентские» сессии

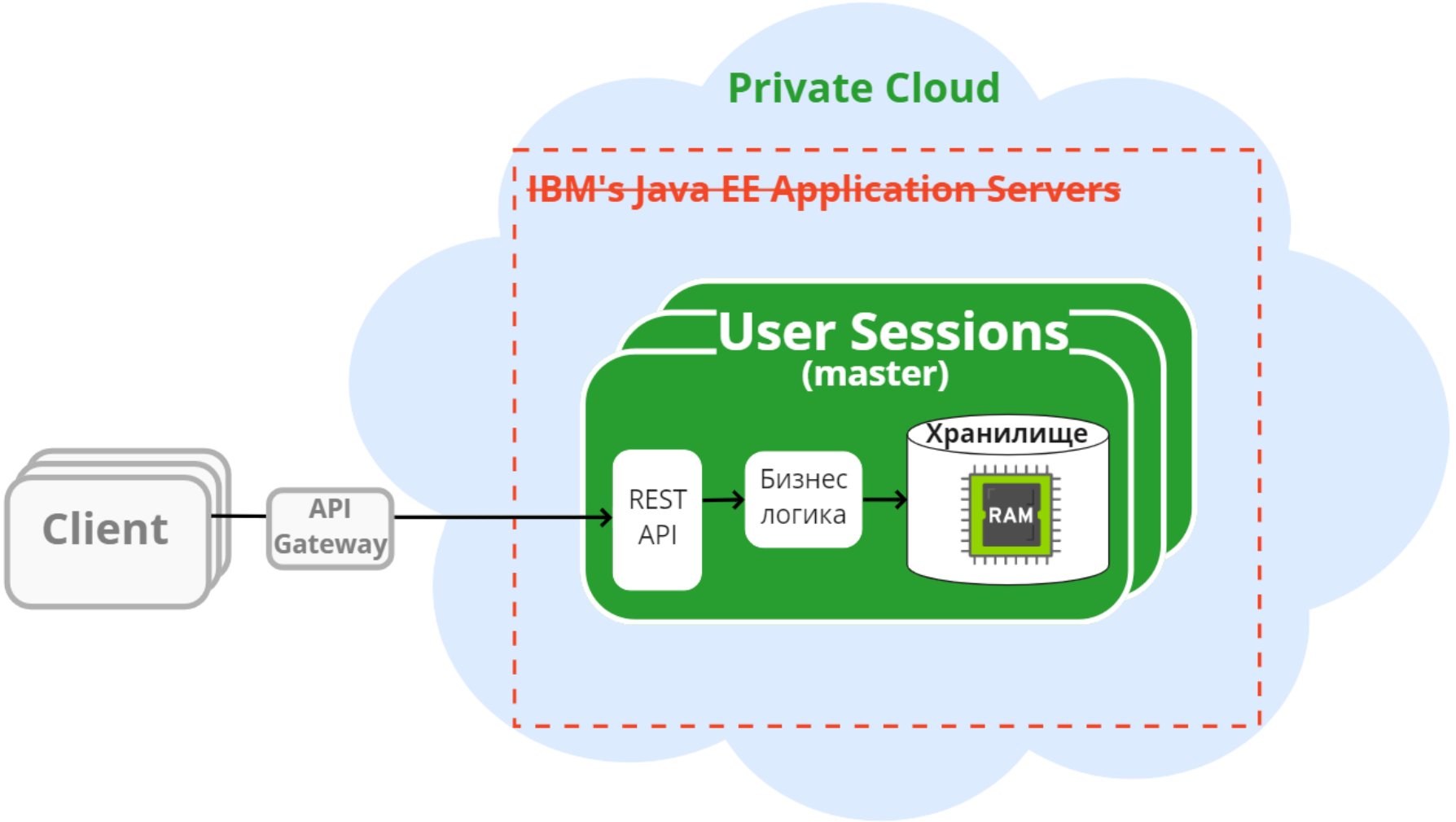
~~IBM’s Java EE Application Servers~~



~~IBM's Java EE Application Servers~~

Уход от западного

Сбер давно начал уходить от западных продуктов



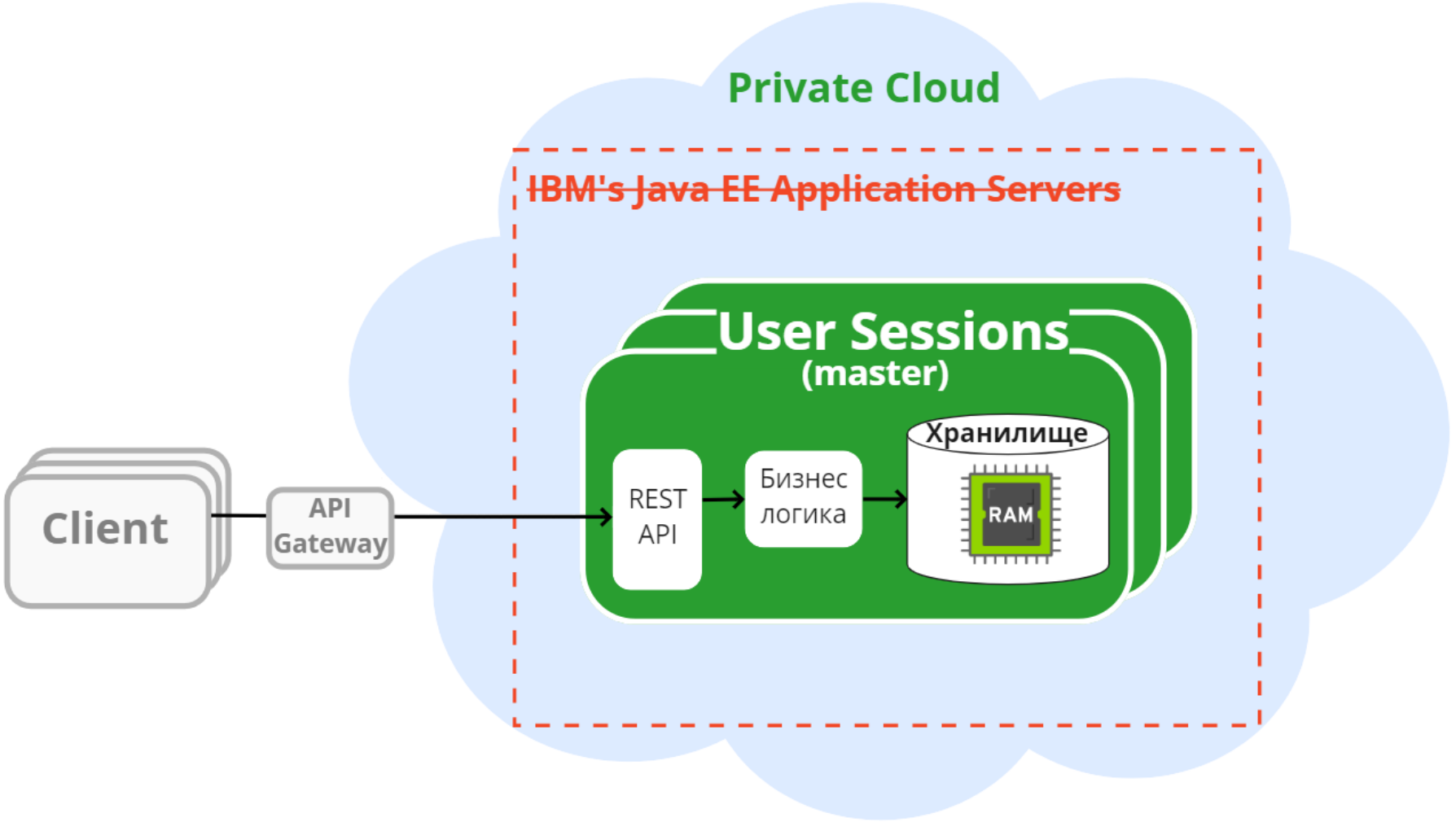
~~IBM's Java EE Application Servers~~

Уход от западного

Сбер давно начал уходить от западных продуктов

Оказались подготовлены

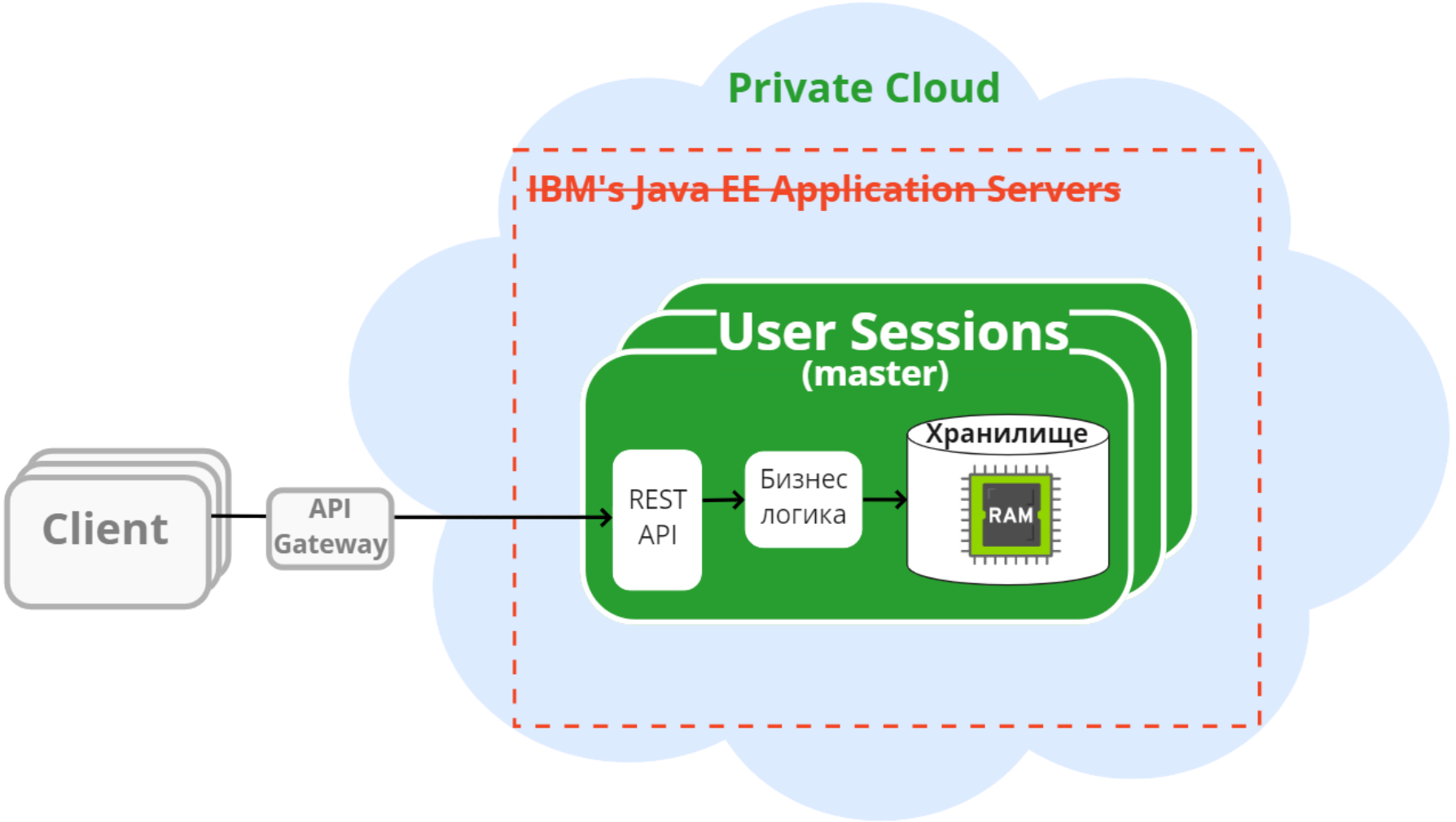
к сегодняшним реалиям



~~IBM’s Java EE Application Servers~~

Помним

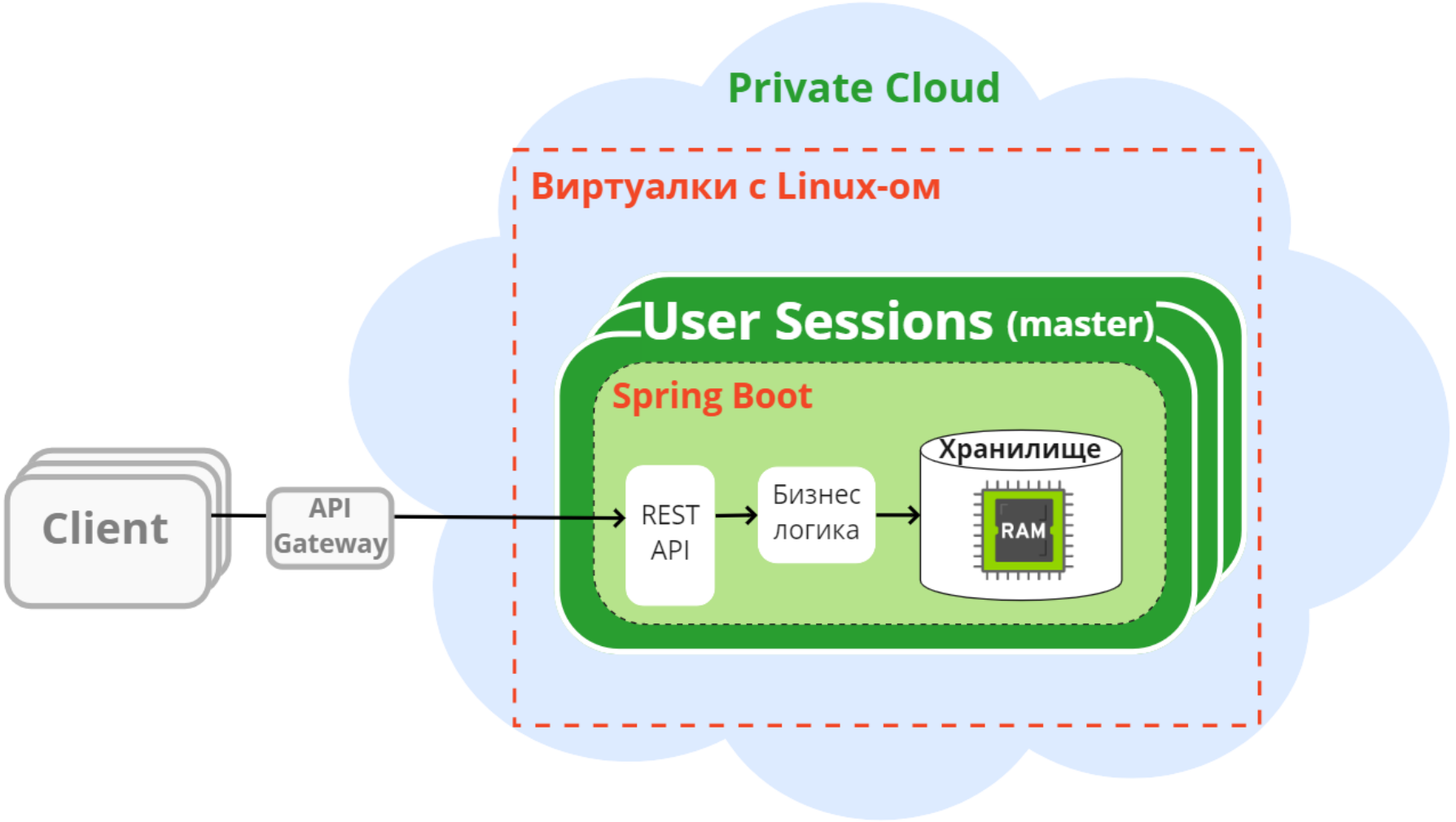
Master-у нельзя в контейнеры (OpenShift / k8s)



Master-хранилище на виртуалках

Spring Boot

Долой Java EE



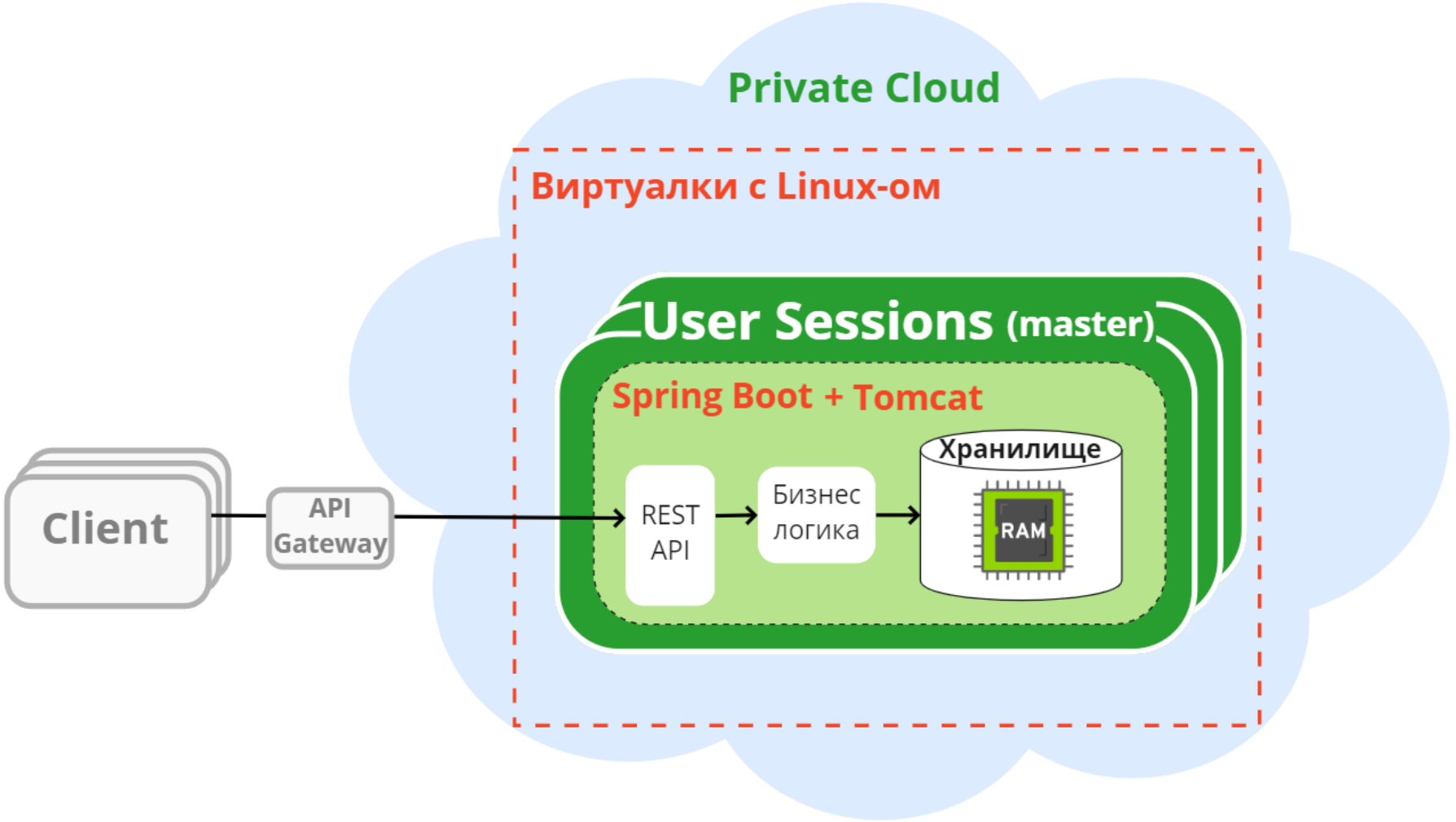
Master-хранилище на виртуалках

Spring Boot

Долой Java EE

Tomcat

Вместо IBM’s application server



Master-хранилище на виртуалках

Spring Boot

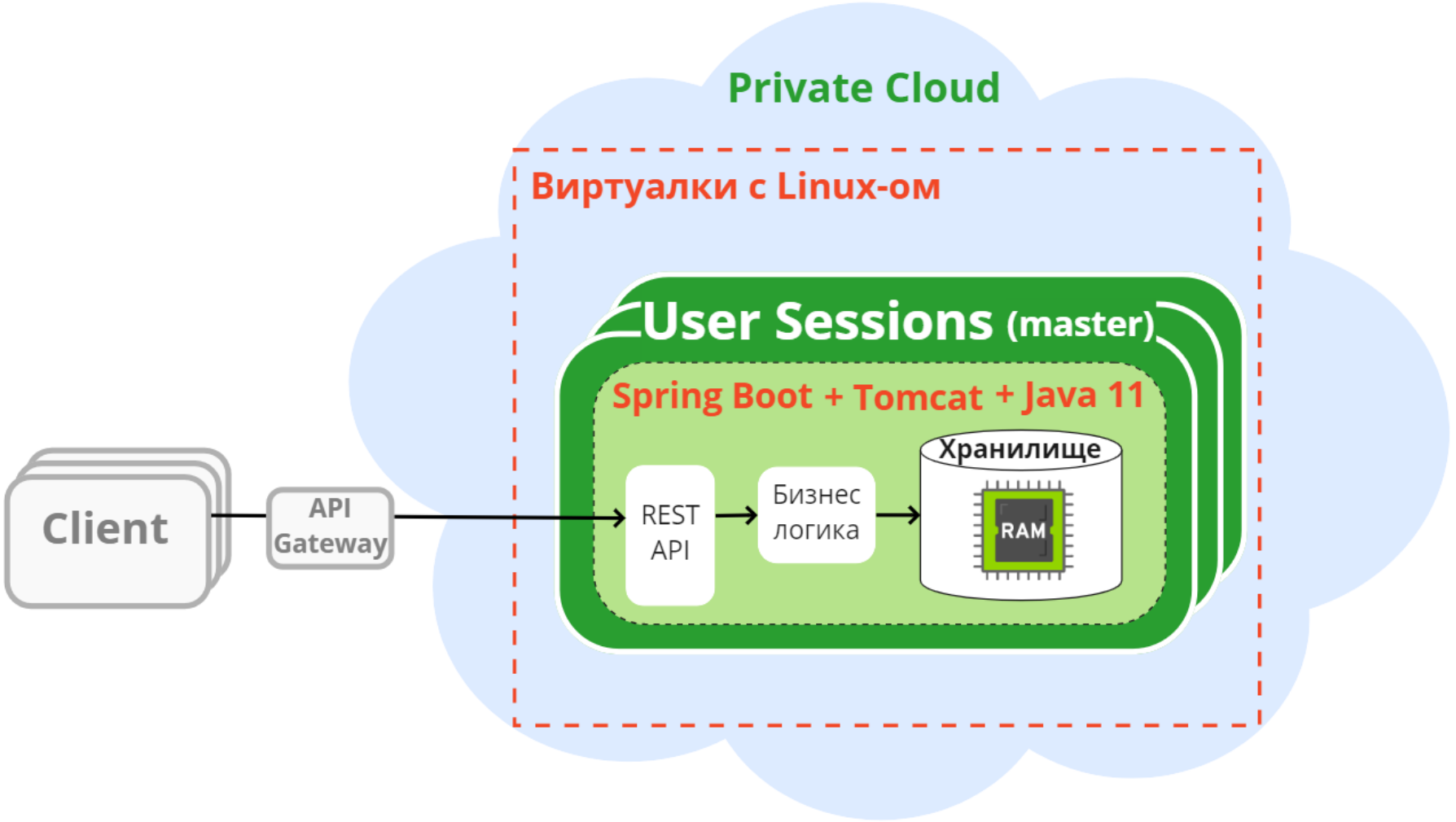
Долой Java EE

Tomcat

Вместо IBM’s application server

Java 11

Вместо Java 8



Старт master-а на виртуалке

Старт fat jar

Запуск на встроенном в Spring Boot Tomcat-е:

```
java -jar master-spring-boot-fat.jar
```

Старт master-а на виртуалке

Старт fat jar

Запуск на встроенном в Spring Boot Tomcat-е:

```
java -jar master-spring-boot-fat.jar
```

Скрипт-стартер

Готовит Spring Boot окружение для старта master-хранилища.

Скрипт-стартер написан на Java

JEP 330

Launch Single-File Source-Code Programs
https://openjdk.org/jeps/330

Старт простой командой:

```
java Run.java
```

Скрипт-стартер написан на Java

JEP 330

Launch Single-File Source-Code Programs
https://openjdk.org/jeps/330

Старт простой командой:
`java Run.java`

Run.java

```
String command =
    “java -jar master-spring-boot-fat.jar”;

ProcessBuilder builder =
    new ProcessBuilder(command.split(“ “);

Process process = builder.start();
process.waitFor(<timeout>);
return process.exitValue();
```

Скрипт-стартер запускается службой Linux

После деплоя

```
systemctl start <service-name>
```

Что даёт

Автоматический старт после:

- форс мажоров
- обновлений
- рестарта Linux-а
- и пр.

Скрипт-стартер запускается службой Linux

Systemd юнит-файл

```
[Unit]
Description=sds master
After=syslog.target
After=network.target
[Service]
User=sds-master
Group=sds-master
WorkingDirectory***************
OOMScoreAdjust=-100
Type=forking
ExecStart=/usr/bin/java ./app/Run.java
ExecStop=/usr/bin/java ./app/Stop.java
Restart=on-failure
RestartSec=60
[Install]
WantedBy=multi-user.target
```

No vendor lock result

Избавились

от потенциального downtime-а из-за блокировки вендорского ПО

Сэкономили

на отказе от
IBM’s Java EE Application Servers

План

~~1. Platform V User Sessions. Начало пути~~

- ~~1.1. Работа с данными и их передача по сети~~

~~2. Развитие User Sessions для выдерживания highload~~

- ~~2.1. Сериализация данных в бинарный формат One Nio~~
- ~~2.2. Сохранение данных «дельтами»~~
- ~~2.3. Чтение данных пачками~~

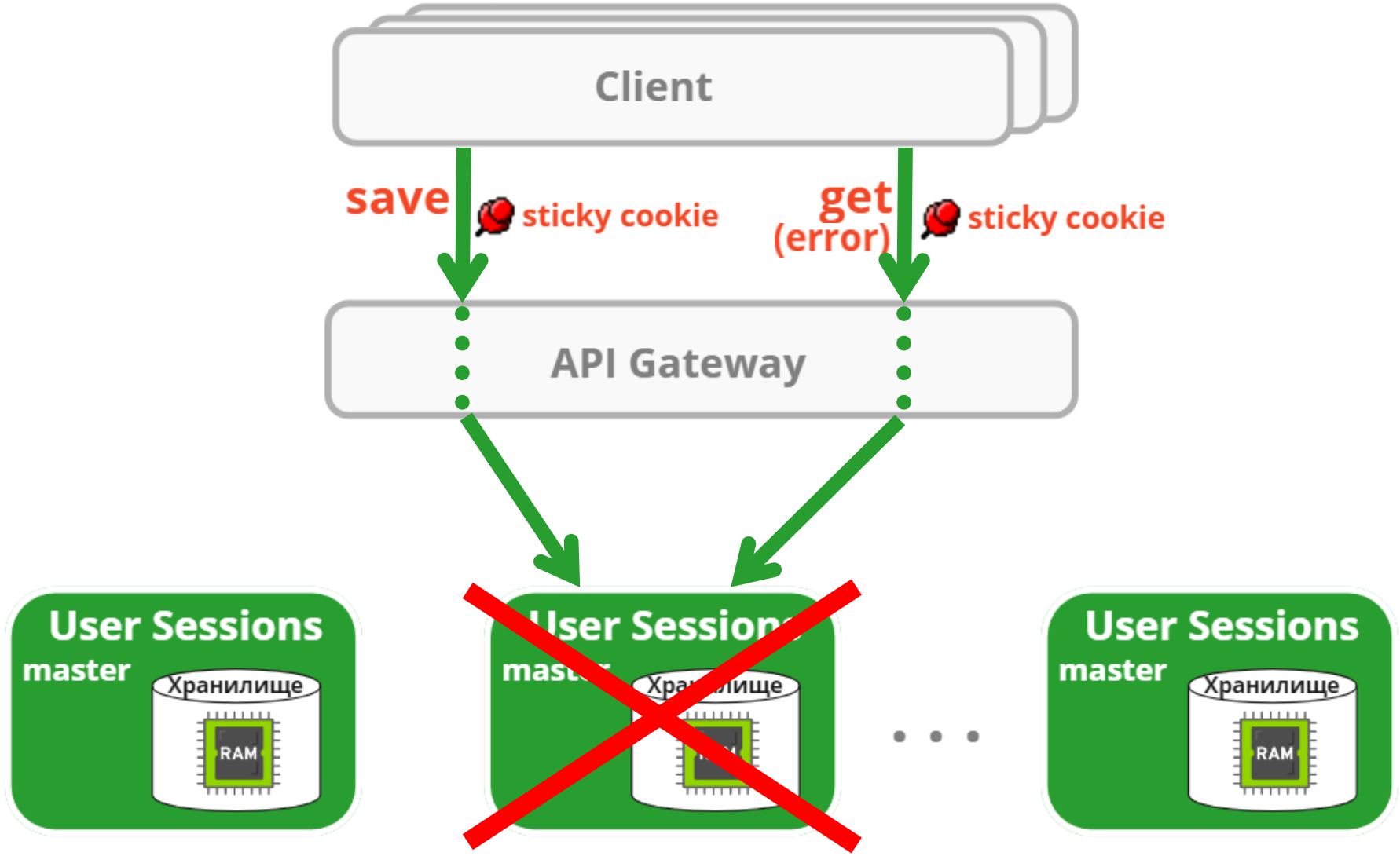
3. Как User Sessions достигает high availability

- ~~3.1. Load balancing & true sticky~~
- ~~3.2. Асинхронная работа с базой данных~~
- ~~3.3. Хранилище не в контейнере~~
- ~~3.4. No vendor lock~~
- 3.5. Репликация данных (в процессе реализации)

4. Дальнейшее развитие User Sessions

- 4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД
- 4.2. Авторизация доступа к данным через Open Policy Agent
- 4.3. «Не клиентские» сессии

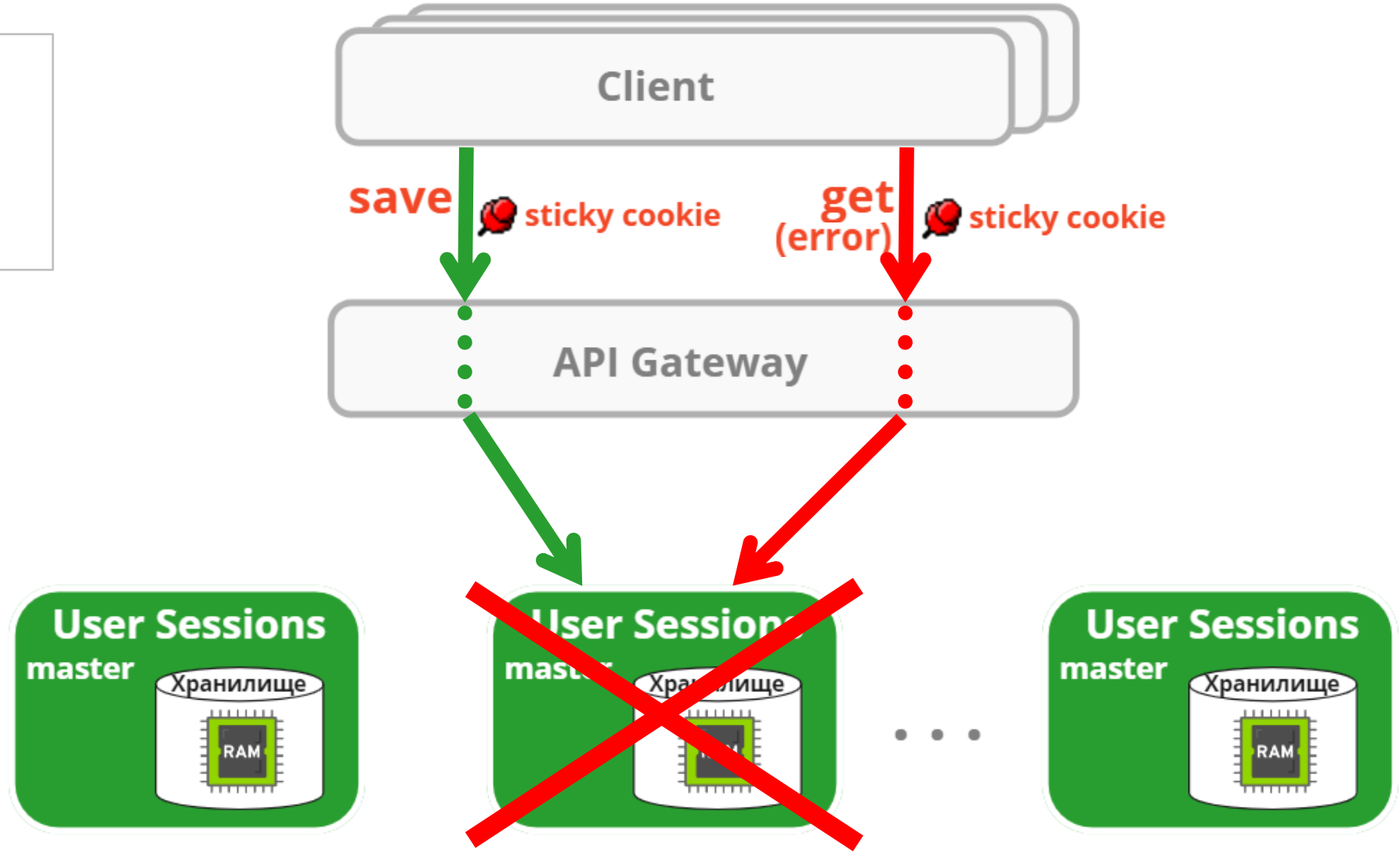
Пока нет репликации



Пока нет репликации

Теряются сессии

Хранимые на потерянном узле



«Купирование» отсутствия репликации

Single Sign On

В СберБанк Онлайн автоматически пересоздаются потерянные сессии

«Купирование» отсутствия репликации

Single Sign On

В СберБанк Онлайн автоматически пересоздаются потерянные сессии

На стартовый экран

выкинет клиента при потере сессии.

Логин/пароль повторно не вводятся.

«Купирование» отсутствия репликации

Single Sign On

В СберБанк Онлайн автоматически пересоздаются потерянные сессии

На стартовый экран

выкинет клиента при потере сессии.

Логин/пароль повторно не вводятся.

Не downtime

формально...

Но жирный минус

Репликация в процессе реализации

Прямо сейчас

реализуется
репликация данных
в User Sessions

Репликация в процессе реализации

Прямо сейчас

реализуется
репликация данных
в User Sessions

CAP теорема

- ✓ Consistency
- Availability
- ✓ Partition tolerance

Репликация в процессе реализации

Прямо сейчас

реализуется репликация в User Sessions

CAP теорема

✅ Consistency

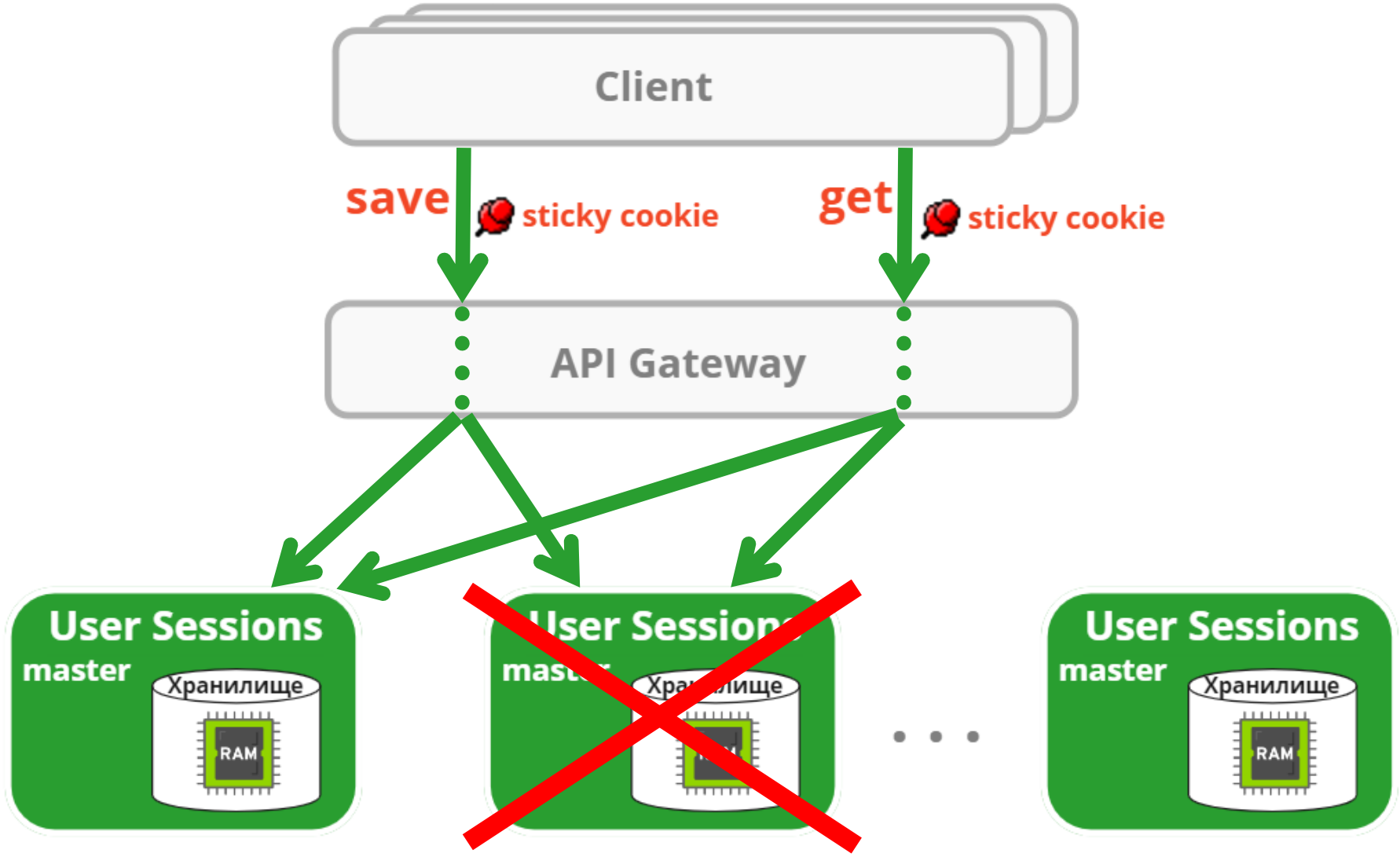
Availability

✅ Partition tolerance

Когда все живы

Latency останется практически неизменной

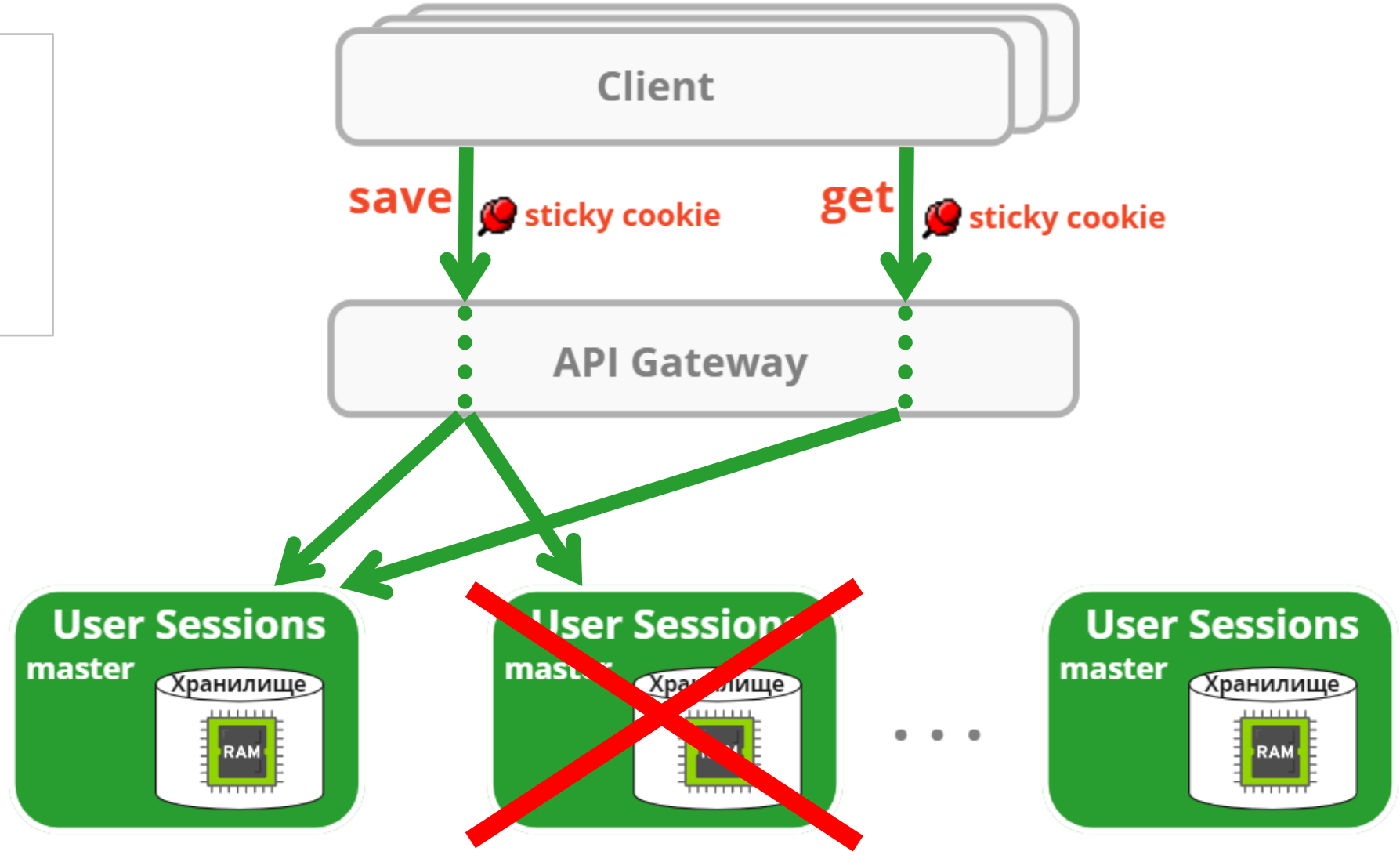
Будет после реализации репликации



Будет после реализации репликации

Не заметно

для потребителя выполняется переключение на реплику



Итог по достижению high availability

Целый ряд решений

уже реализован.

Часто сложно измерить эффект.

Не прекращаем

анализ потенциальных проблем и борьбу с ними

3 года

без отказов в production

План

~~1. Platform V User Sessions. Начало пути~~

~~1.1. Работа с данными и их передача по сети~~

~~2. Развитие User Sessions для выдерживания highload~~

~~2.1. Сериализация данных в бинарный формат One Nio~~

~~2.2. Сохранение данных «дельтами»~~

~~2.3. Чтение данных пачками~~

~~3. Как User Sessions достигает high availability~~

~~3.1. Load balancing & true sticky~~

~~3.2. Асинхронная работа с базой данных~~

~~3.3. Хранилище не в контейнере~~

~~3.4. No vendor lock~~

~~3.5. Репликация данных (в процессе реализации)~~

4. Дальнейшее развитие User Sessions

4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД

4.2. Авторизация доступа к данным через Open Policy Agent

4.3. «Не клиентские» сессии

План

~~1. Platform V User Sessions. Начало пути~~

- ~~1.1. Работа с данными и их передача по сети~~

~~2. Развитие User Sessions для выдерживания highload~~

- ~~2.1. Сериализация данных в бинарный формат One Nio~~
- ~~2.2. Сохранение данных «дельтами»~~
- ~~2.3. Чтение данных пачками~~

~~3. Как User Sessions достигает high availability~~

- ~~3.1. Load balancing & true sticky~~
- ~~3.2. Асинхронная работа с базой данных~~
- ~~3.3. Хранилище не в контейнере~~
- ~~3.4. No vendor lock~~
- ~~3.5. Репликация данных (в процессе реализации)~~

4. Дальнейшее развитие User Sessions

- 4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД
- 4.2. Авторизация доступа к данным через Open Policy Agent
- 4.3. «Не клиентские» сессии

Зачем отказываться от БД?

Особенности хранимых данных

Не mission critical

При отказе БД просто не видно сессий в админке

Зачем отказываться от БД?

Особенности хранимых данных

Не mission critical

При отказе БД просто не видно сессий в админке

Эфемерные

Сессии живут не долго:

- 5 минут активны
- 20-25 живут «брошенные»

Зачем отказываться от БД?

Особенности хранимых данных

Не mission critical

При отказе БД просто не видно сессий в админке

Эфемерные

Сессии живут не долго:

- 5 минут активны
- 20-25 живут «брошенные»

Компактные

Теоретический пик 500 тысяч сессий влезает в RAM

Что вместо БД?

Platform V DataGrid

Распределённая in-memory база данных

https://platformv.sber.ru/products/datagrid

Что вместо БД?

Platform V DataGrid

Распределённая in-memory база данных

https://platformv.sber.ru/products/datagrid

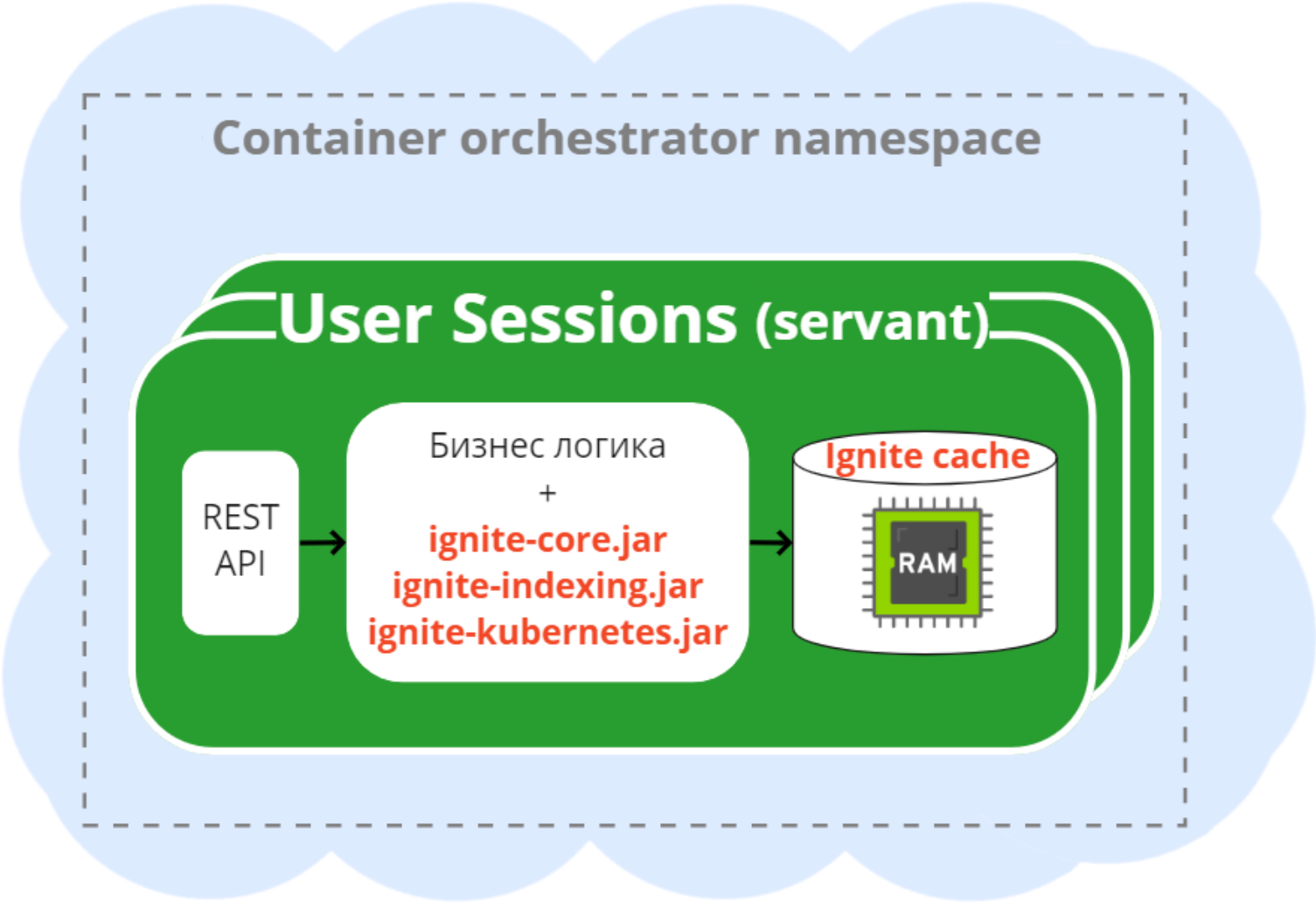
Свой кластер

Вместо выделенного кластера

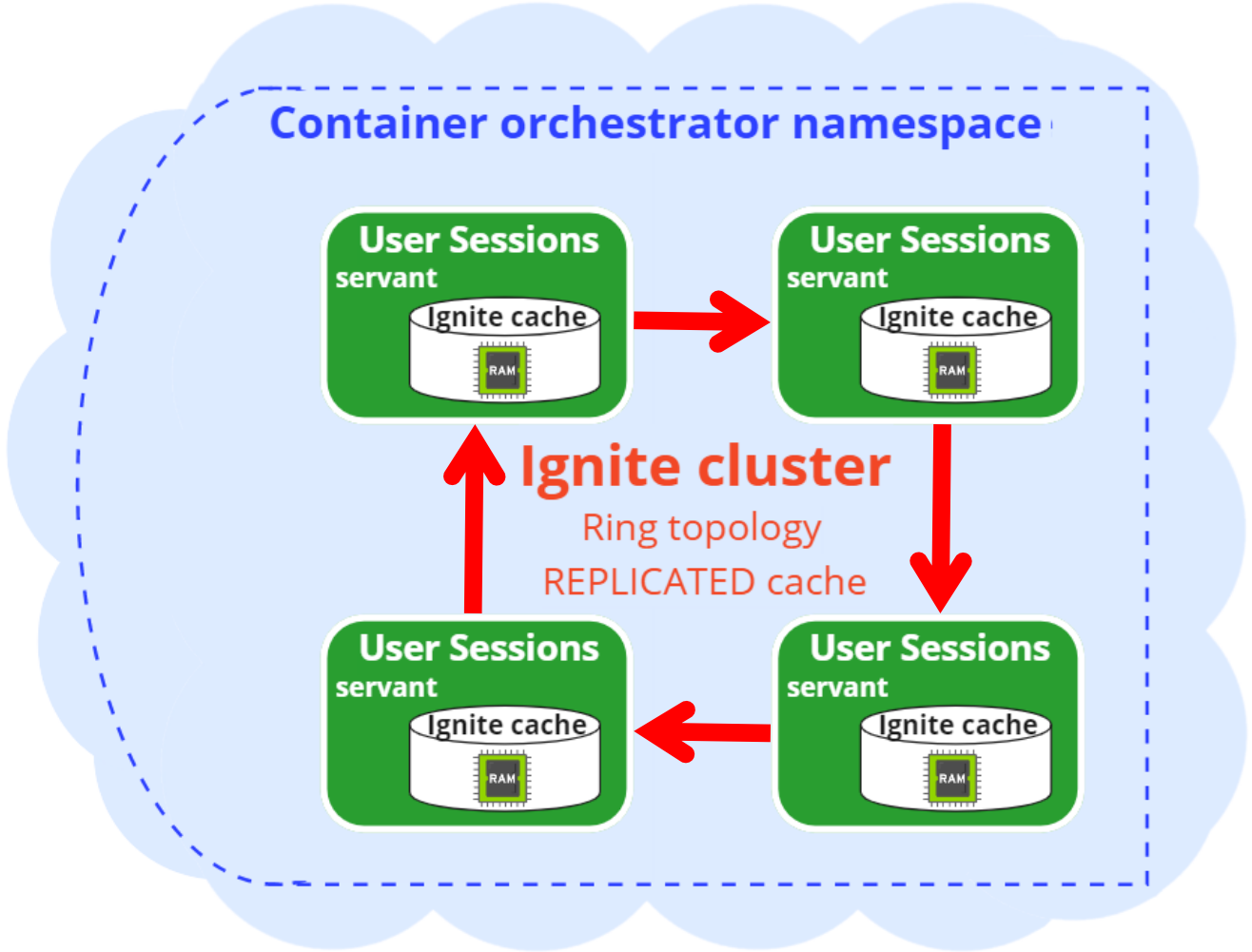
Embedded кластер DataGrid (Ignite)

Maven dependencies

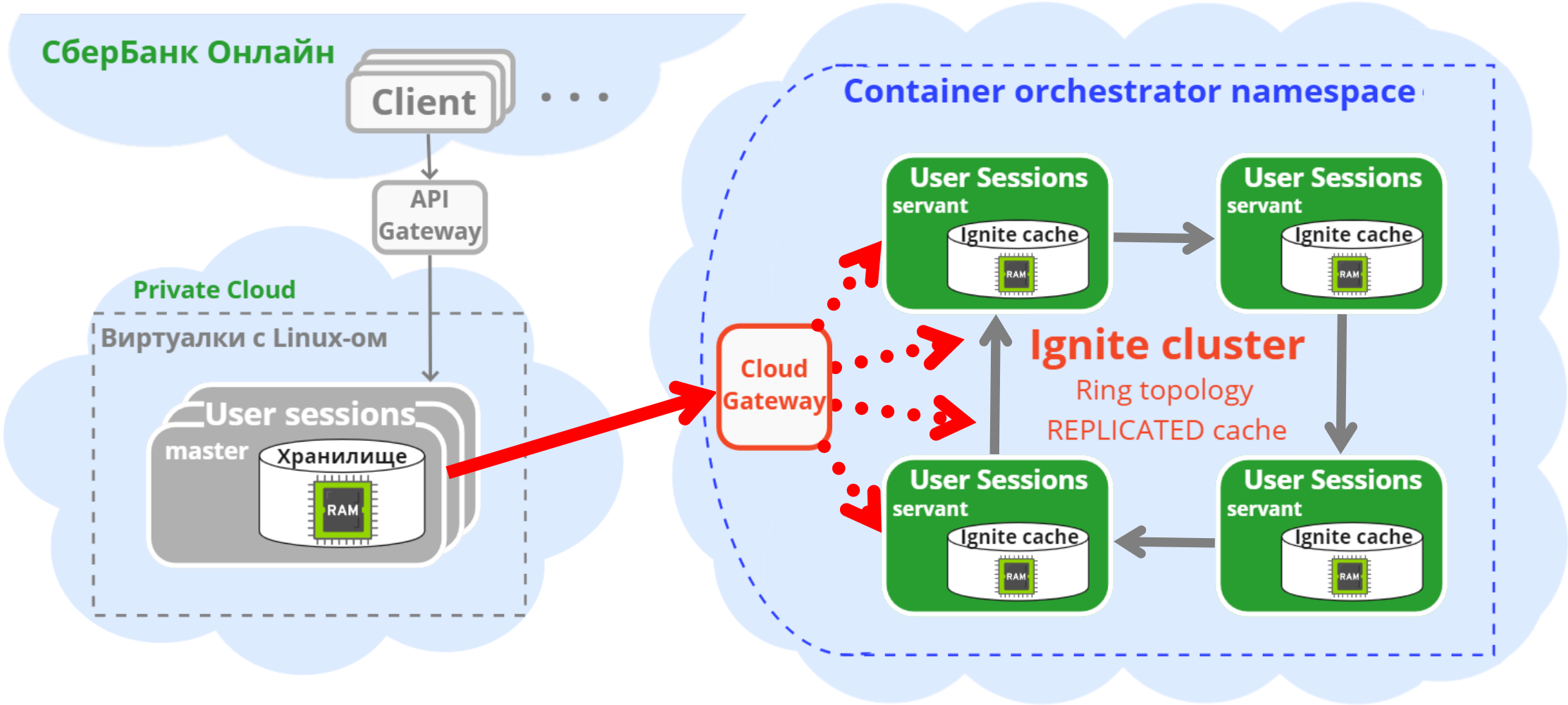
```
<dependency>
 <groupId>com.sbt.ignite</groupId>
 <artifactId>ignite-core</artifactId>
</dependency>
<dependency>
 <groupId>com.sbt.ignite</groupId>
 <artifactId>ignite-indexing</artifactId>
</dependency>
<dependency>
 <groupId>com.sbt.ignite</groupId>
 <artifactId>ignite-kubernetes</artifactId>
</dependency>
```



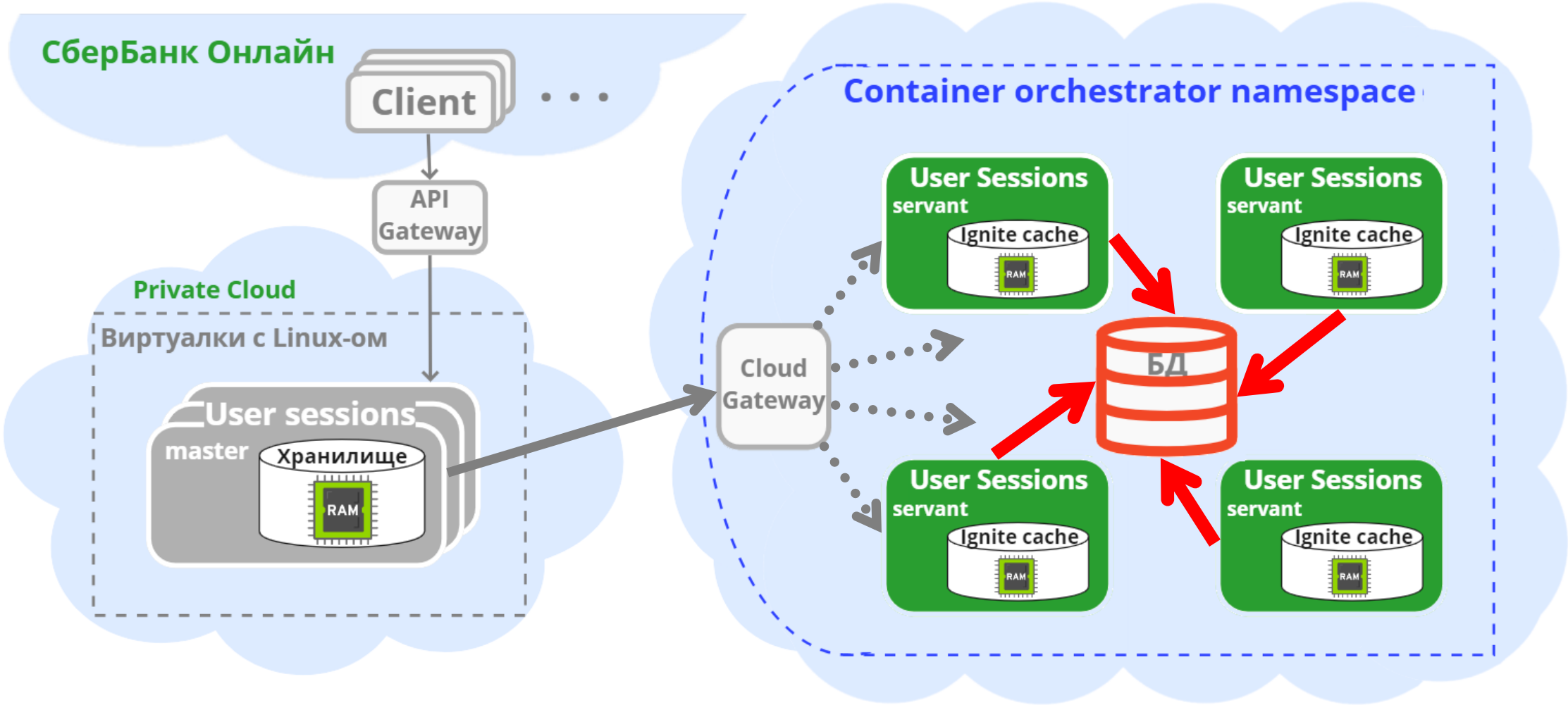
Embedded кластер DataGrid (Ignite)



Embedded кластер DataGrid (Ignite)



Servant может работать с БД



Что получим, избавившись от БД?

+ устойчивость

Независимость от отказов БД

- избыточность

В архитектуре сервиса

+ экономия

Если СУБД была платной

План

~~1. Platform V User Sessions. Начало пути~~

- ~~1.1. Работа с данными и их передача по сети~~

~~2. Развитие User Sessions для выдерживания highload~~

- ~~2.1. Сериализация данных в бинарный формат One Nio~~
- ~~2.2. Сохранение данных «дельтами»~~
- ~~2.3. Чтение данных пачками~~

~~3. Как User Sessions достигает high availability~~

- ~~3.1. Load balancing & true sticky~~
- ~~3.2. Асинхронная работа с базой данных~~
- ~~3.3. Хранилище не в контейнере~~
- ~~3.4. No vendor lock~~
- ~~3.5. Репликация данных (в процессе реализации)~~

4. Дальнейшее развитие User Sessions

- ~~4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД~~
- 4.2. Авторизация доступа к данным через Open Policy Agent
- 4.3. «Не клиентские» сессии

Проблема отсутствия авторизации

Любой может

Любой микросервис может прочитать/изменить секцию данных, зная её имя

Проблема отсутствия авторизации

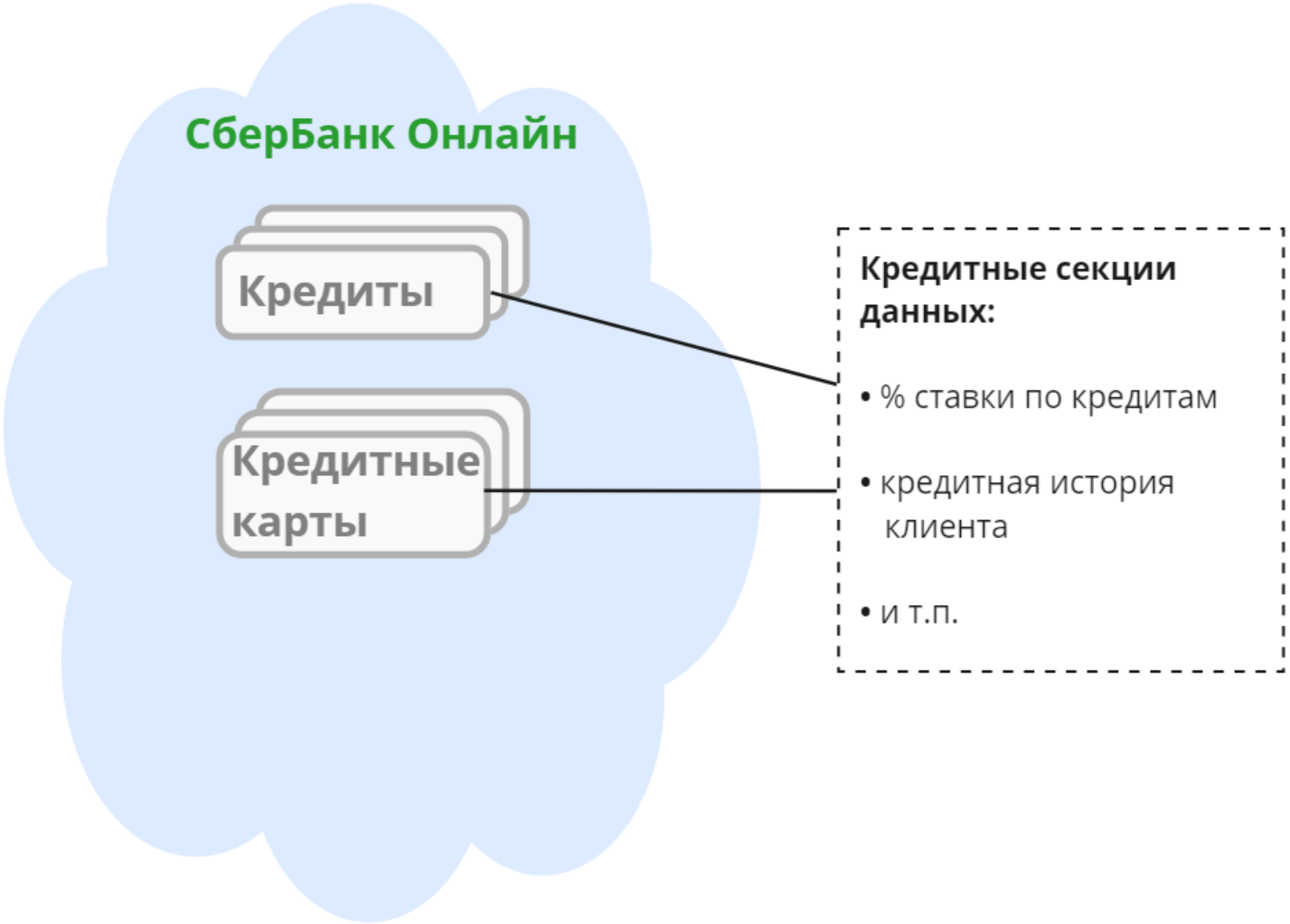
Любой может

Любой микросервис может прочитать/изменить секцию данных, зная её имя

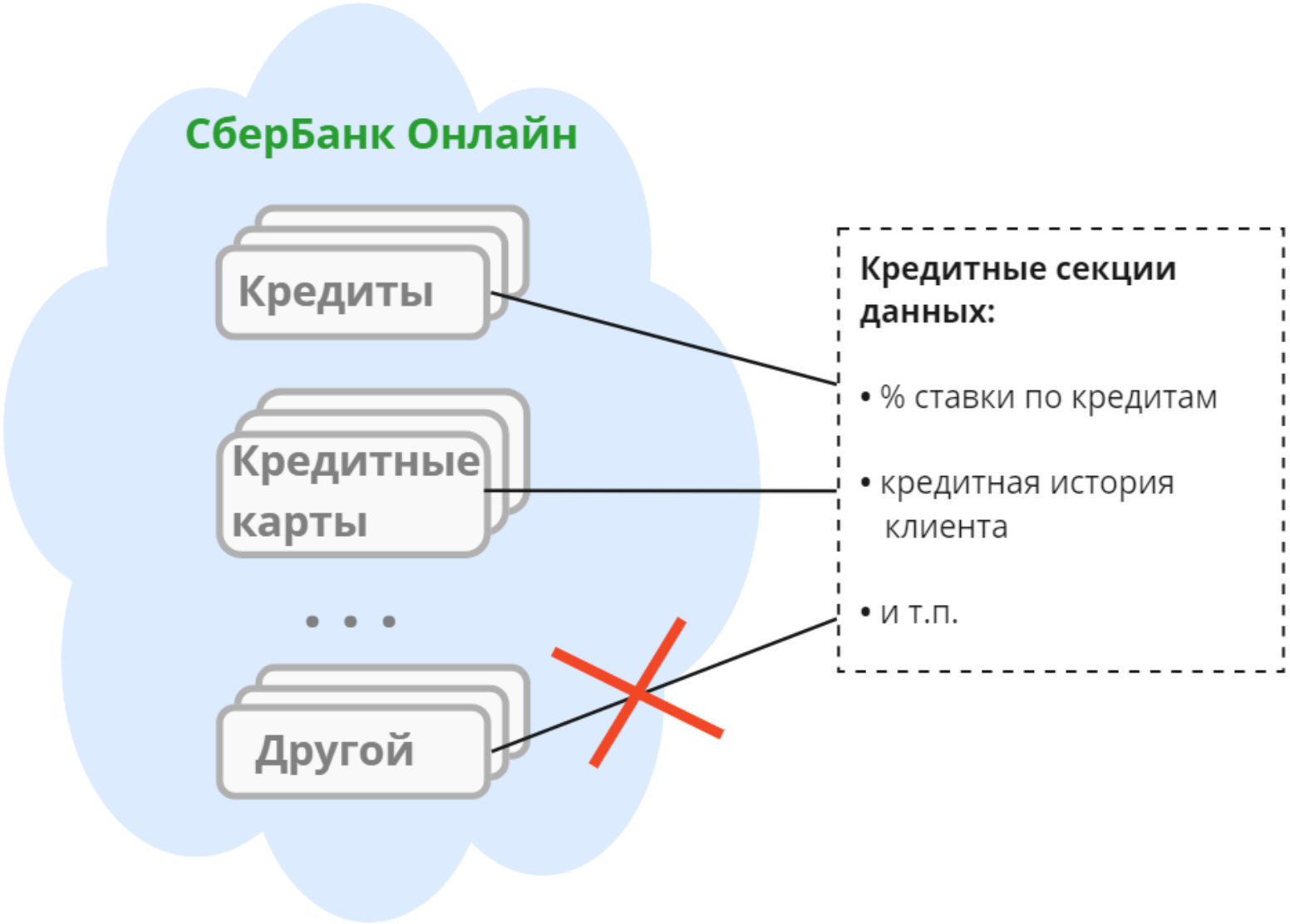
Нужно защищать

Микросервисам нужно защищать свои контракты от других микросервисов

Проблема отсутствия авторизации



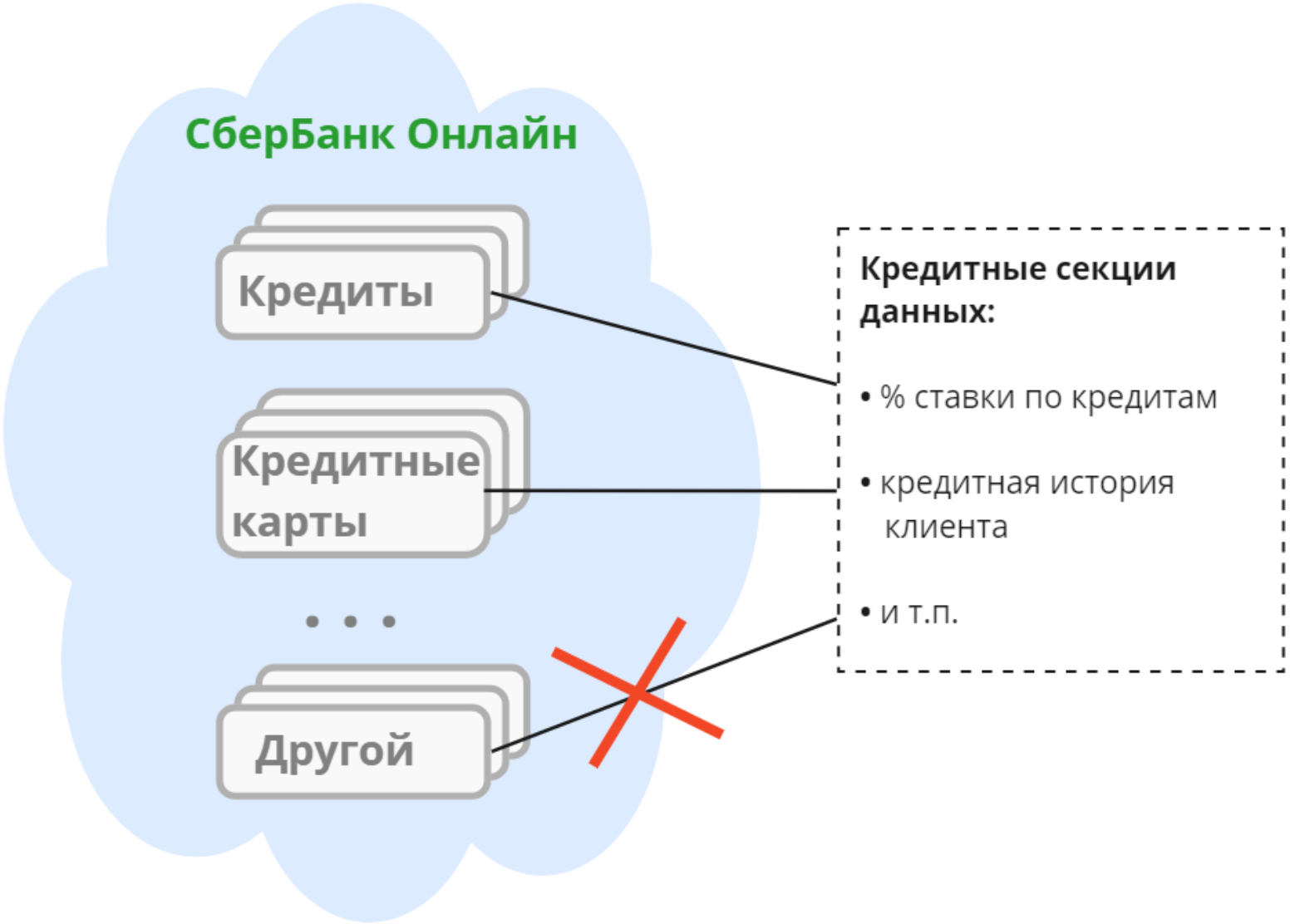
Проблема отсутствия авторизации



Проблема отсутствия авторизации

Микросервис, а не end user

является субъектом прав доступа к секциям

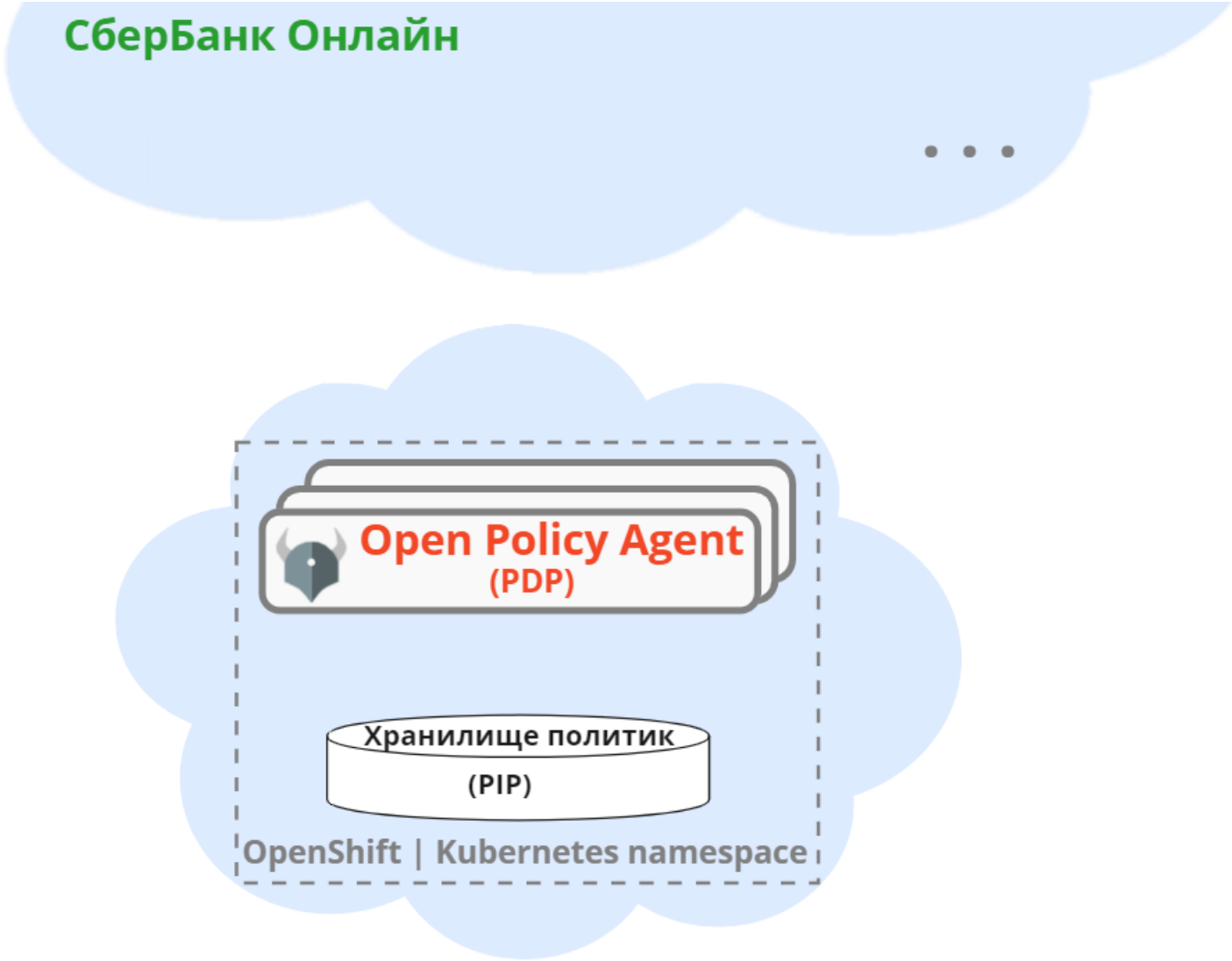


Планируем использовать Open Policy Agent

Platform V IAM SE

имеет в составе Open Policy Agent, доработанный до enterprise-уровня

https://platformv.sber.ru/products/iam

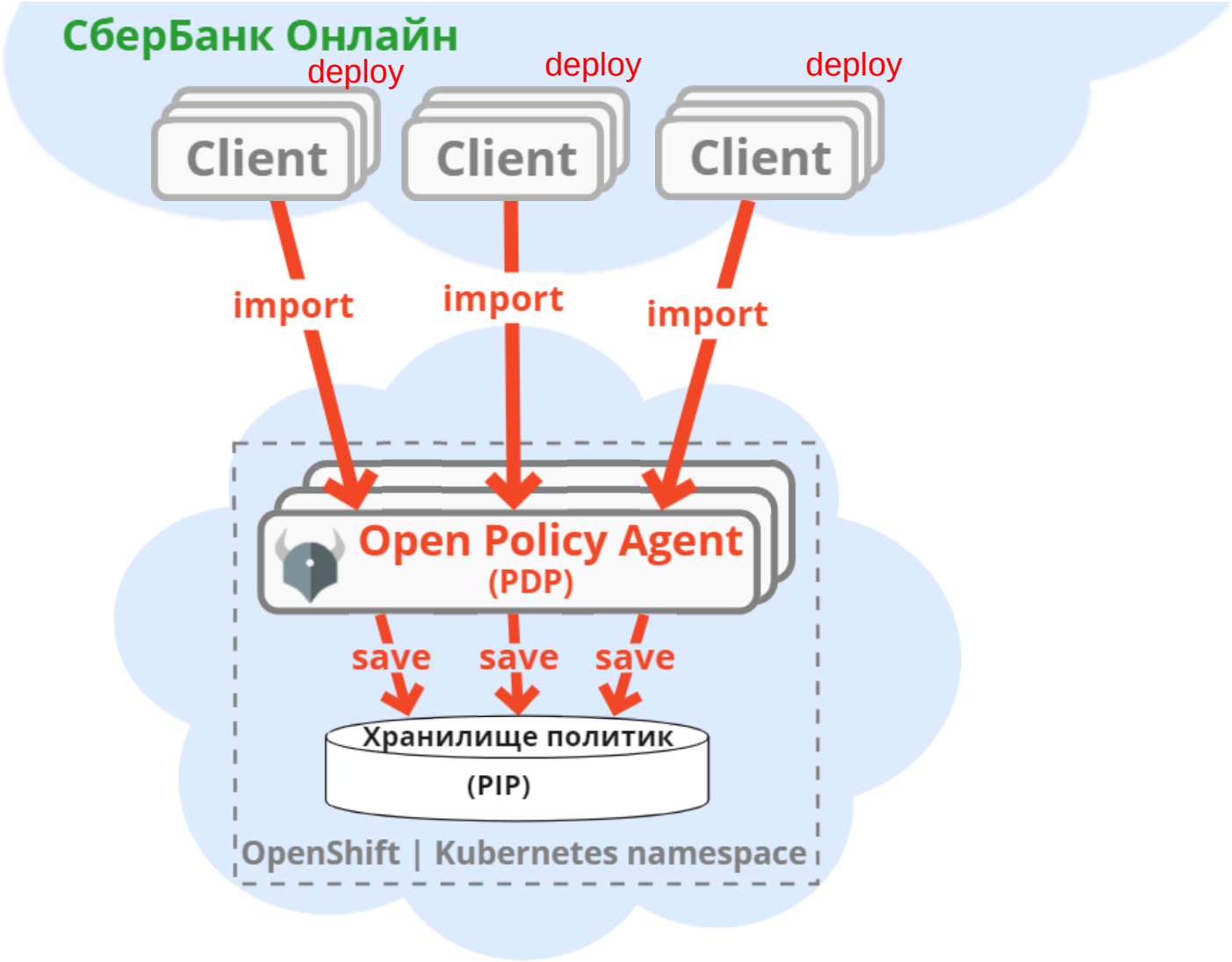


Импорт политик в Open Policy Agent

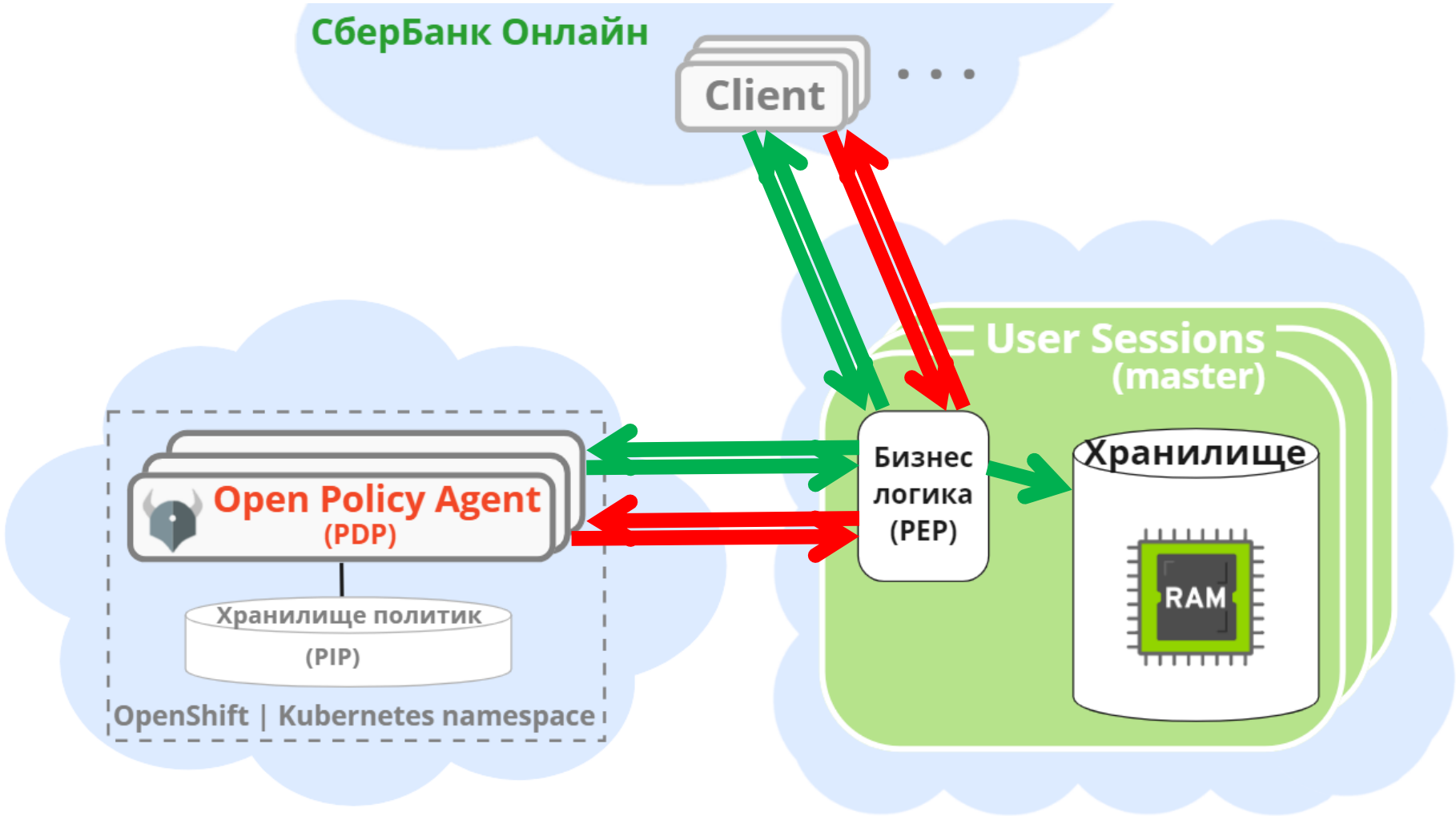
Platform V IAM SE

имеет в составе Open Policy Agent, доработанный до enterprise-уровня

https://platformv.sber.ru/products/iam



Master обращается к Open Policy Agent



JSON политики для Open Policy Agent

Rego политика

Политики для OPA пишутся на языке Rego

JSON политики для Open Policy Agent

Rego политика

Политики для OPA пишутся на языке Rego

Читает JSON

В Platform V Rego политика читает декларации из JSON

JSON политики для Open Policy Agent

Rego политика

Политики для OPA пишутся на языке Rego

Читает JSON

В Platform V Rego политика читает декларации из JSON

JSON ACL Policy

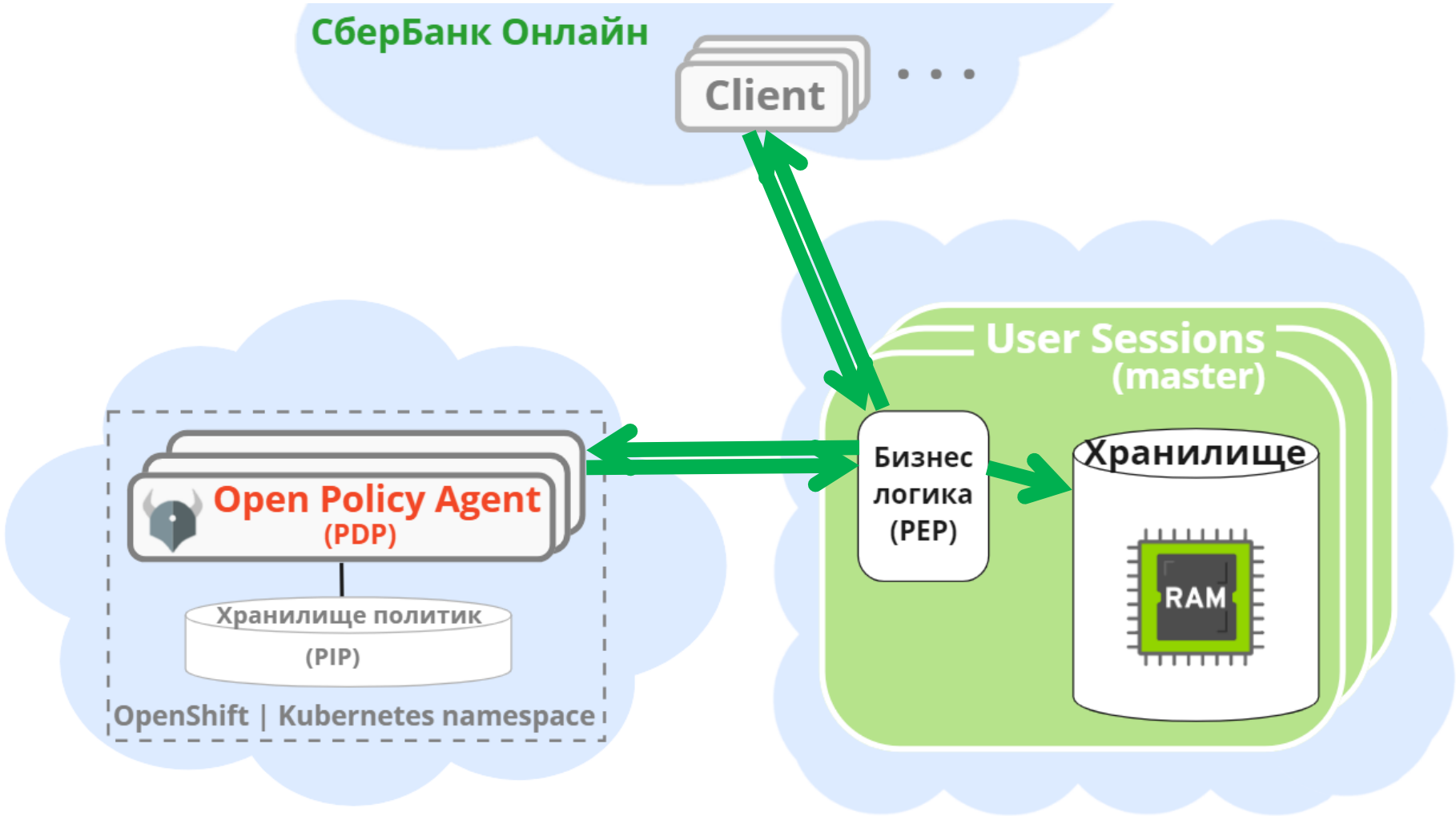
Формат, как в Amazon Web Services

Пример JSON политики для Open Policy Agent

«Кредиты» дали себе права на чтение/запись секции «rates»

```
{
  "Version": ***,
  "Statement": [
    {
      "Sid": "rates_access",
      "principal": {
        "groups": ["rn:sbrf:credits:*:private:sbol:*"]
      },
      "Effect": "Allow",
      "Action": [
        "sessions:get",
        "sessions:save"
      ],
      "Resource": ["rn:sbrf:sessions:*:private:sbol:namespaces/some-ns/sections/rates"]
    }
  ]
}
```

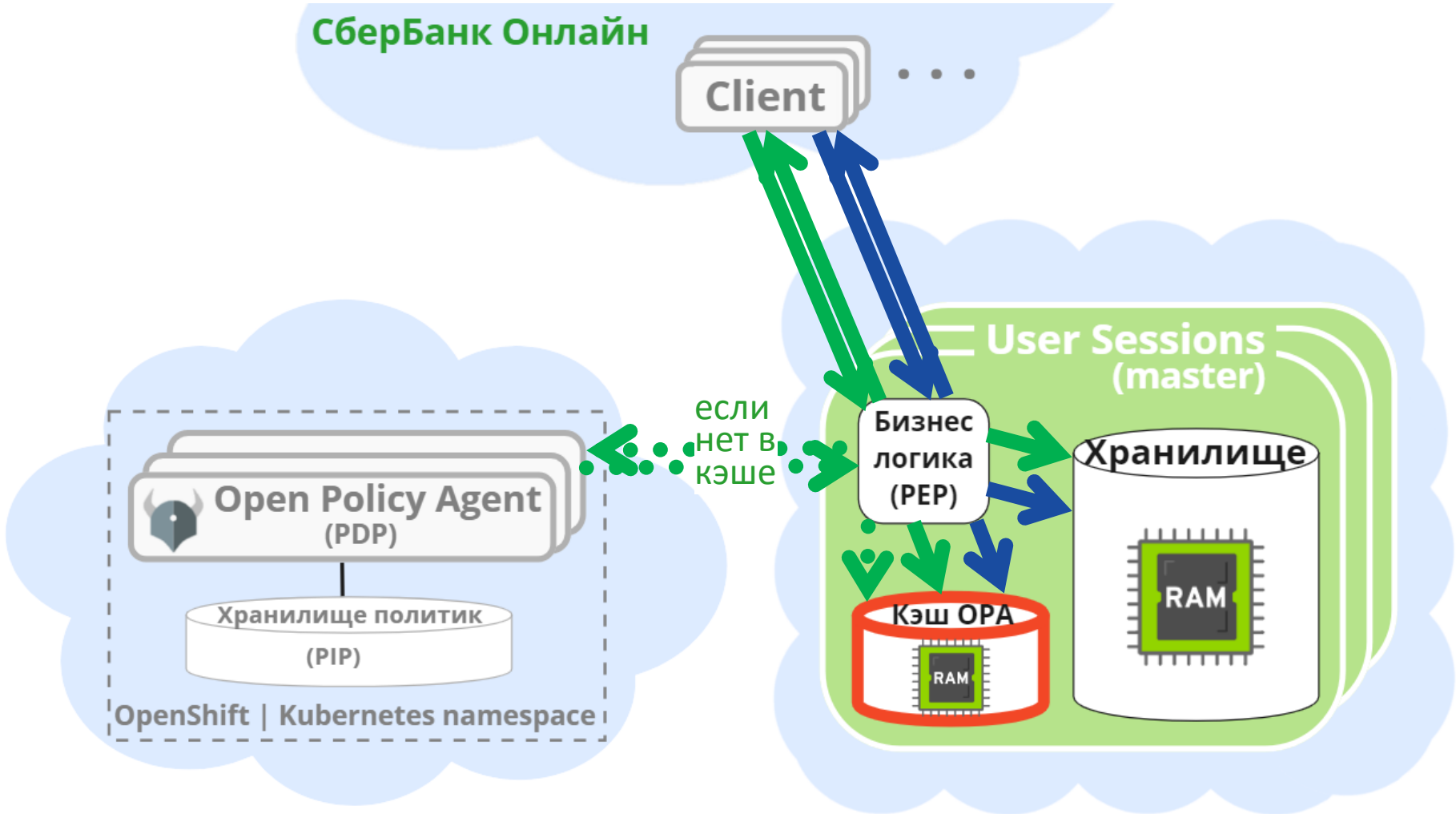
Что же станет с latency master-хранилища?



Кэширование ответов Open Policy Agent

Ответы статичны

Изменения только при редеплое



Кэширование ответов Open Policy Agent

Ответы статичны

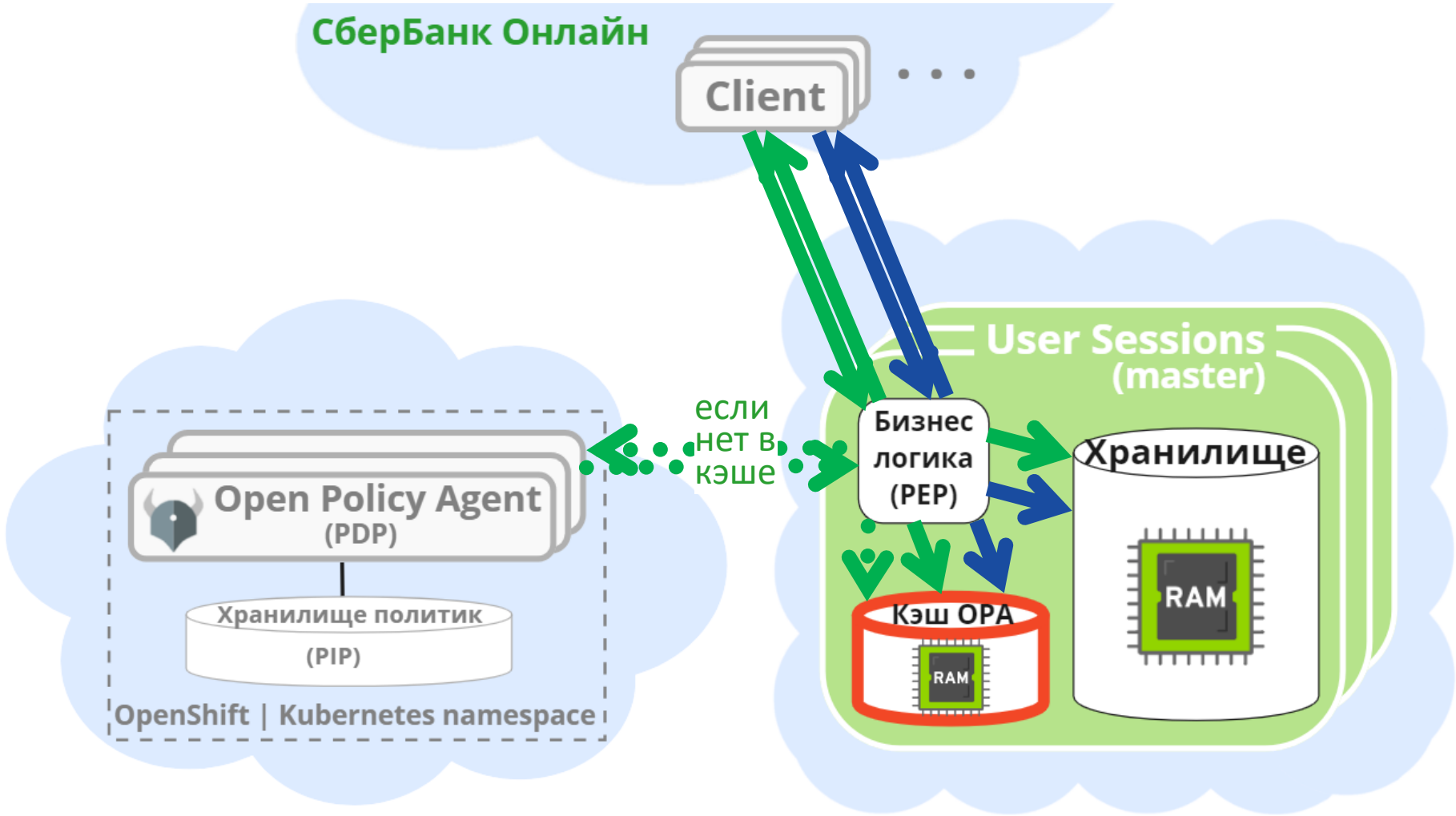
Изменения только при редеплое

~15000 вариантов

ответов OPA

150 сервисов * 100 секций

Влезет в память



Кэширование ответов Open Policy Agent

Ответы статичны

Изменения только при редеплое

~15000 вариантов

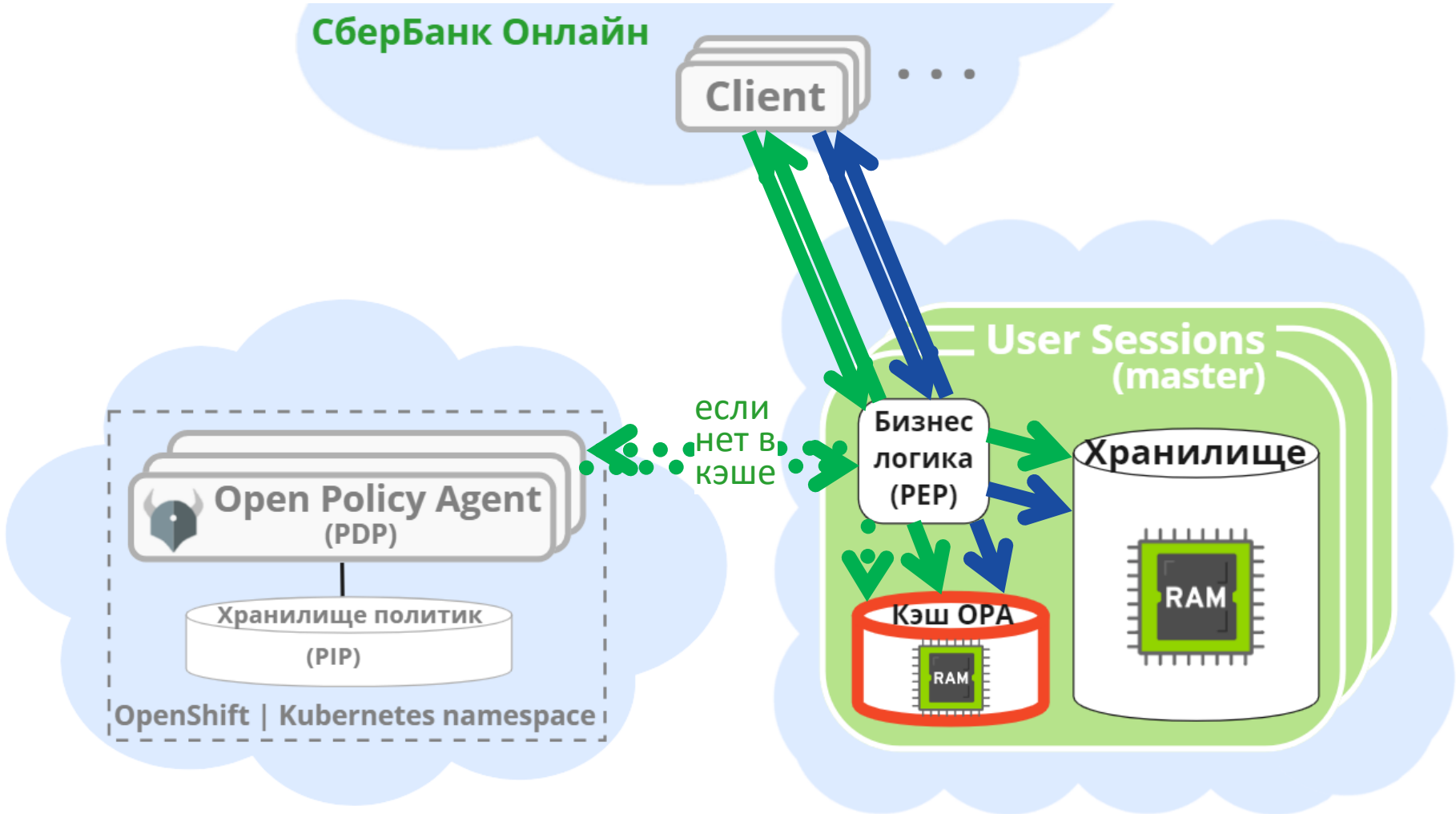
ответов OPA

150 сервисов * 100 секций

Влезет в память

Нет инвалидации

кэша – будет асинхронная актуализация



Что получим от авторизации?

Безопасность

использования данных
в User Sessions

Защищённые контракты

Наших потребителей от других
микросервисов

План

~~1. Platform V User Sessions. Начало пути~~

- ~~1.1. Работа с данными и их передача по сети~~

~~2. Развитие User Sessions для выдерживания highload~~

- ~~2.1. Сериализация данных в бинарный формат One Nio~~
- ~~2.2. Сохранение данных «дельтами»~~
- ~~2.3. Чтение данных пачками~~

~~3. Как User Sessions достигает high availability~~

- ~~3.1. Load balancing & true sticky~~
- ~~3.2. Асинхронная работа с базой данных~~
- ~~3.3. Хранилище не в контейнере~~
- ~~3.4. No vendor lock~~
- ~~3.5. Репликация данных (в процессе реализации)~~

4. Дальнейшее развитие User Sessions

- ~~4.1. Platform V DataGrid (на основе Apache Ignite) вместо СУБД~~
- ~~4.2. Авторизация доступа к данным через Open Policy Agent~~
- 4.3. «Не клиентские» сессии

Что такое «не клиентские» сессии?

Не для end user

Сессии не связаны с конкретным пользователем

Что такое «не клиентские» сессии?

Не для end user

Сессии не связаны с конкретным пользователем

Большой TTL

Данные живут долго, например, месяц

Что такое «не клиентские» сессии?

Не для end user

Сессии не связаны с конкретным пользователем

Большой TTL

Данные живут долго, например, месяц

Термина нет

Варианты:

- «не клиентские»
- служебные
- технические

«Не клиентские» сессии – важная фича

Расширяет

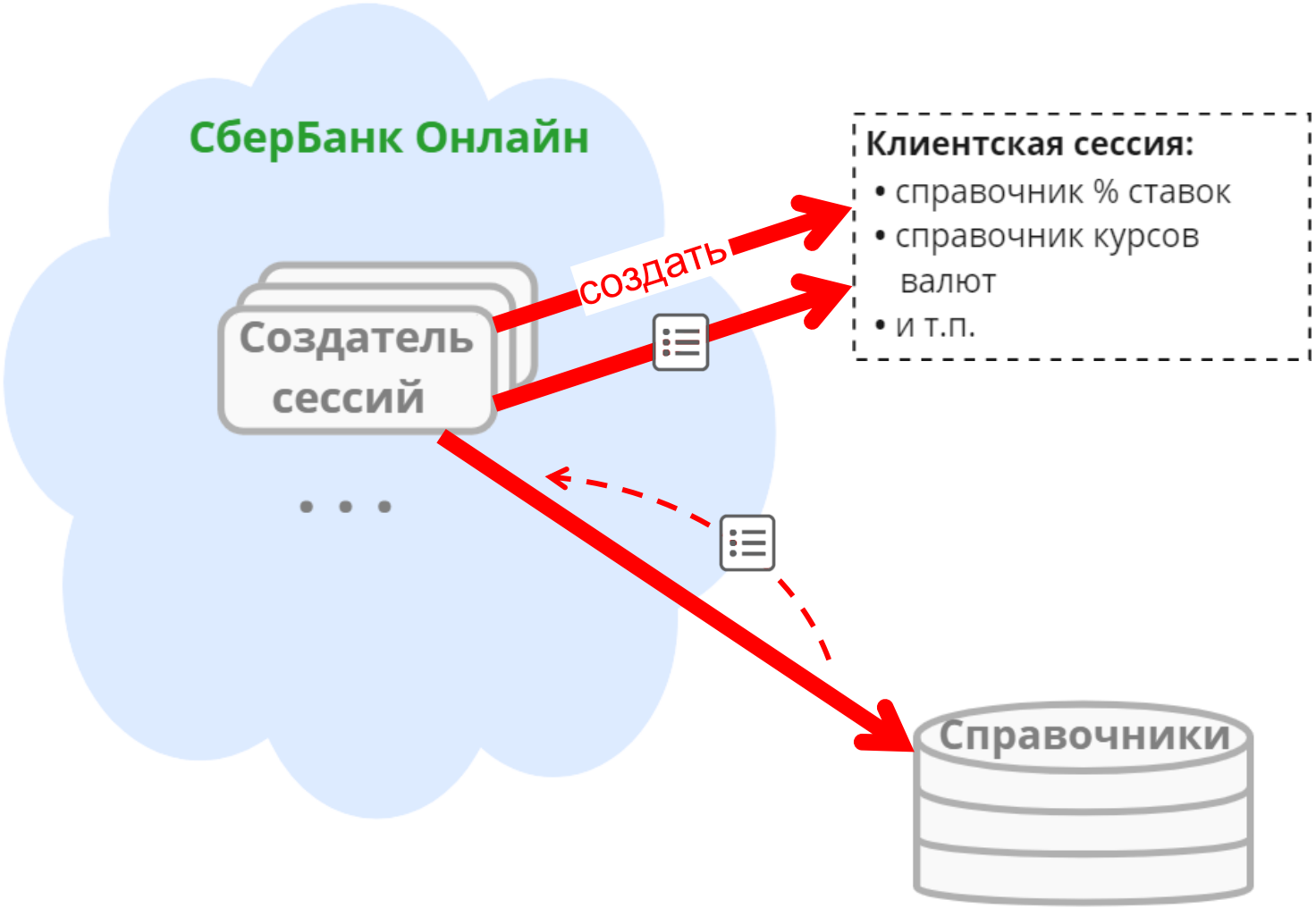
область применения User Sessions

Масса use case-ов

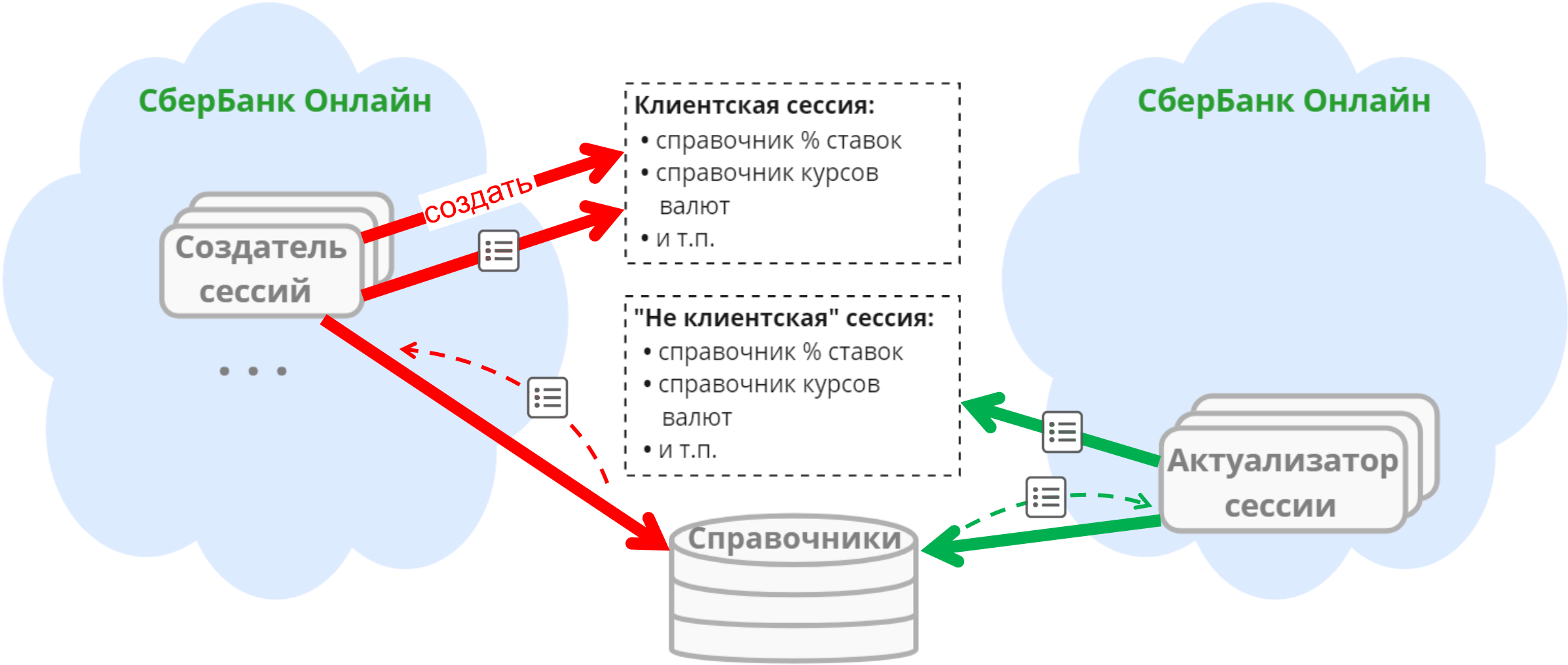
- просто in-memory хранилище
- кэш перед БД
- асинхронная обработка данных
- …

Use case: комбинация клиентской и не клиентской сессий

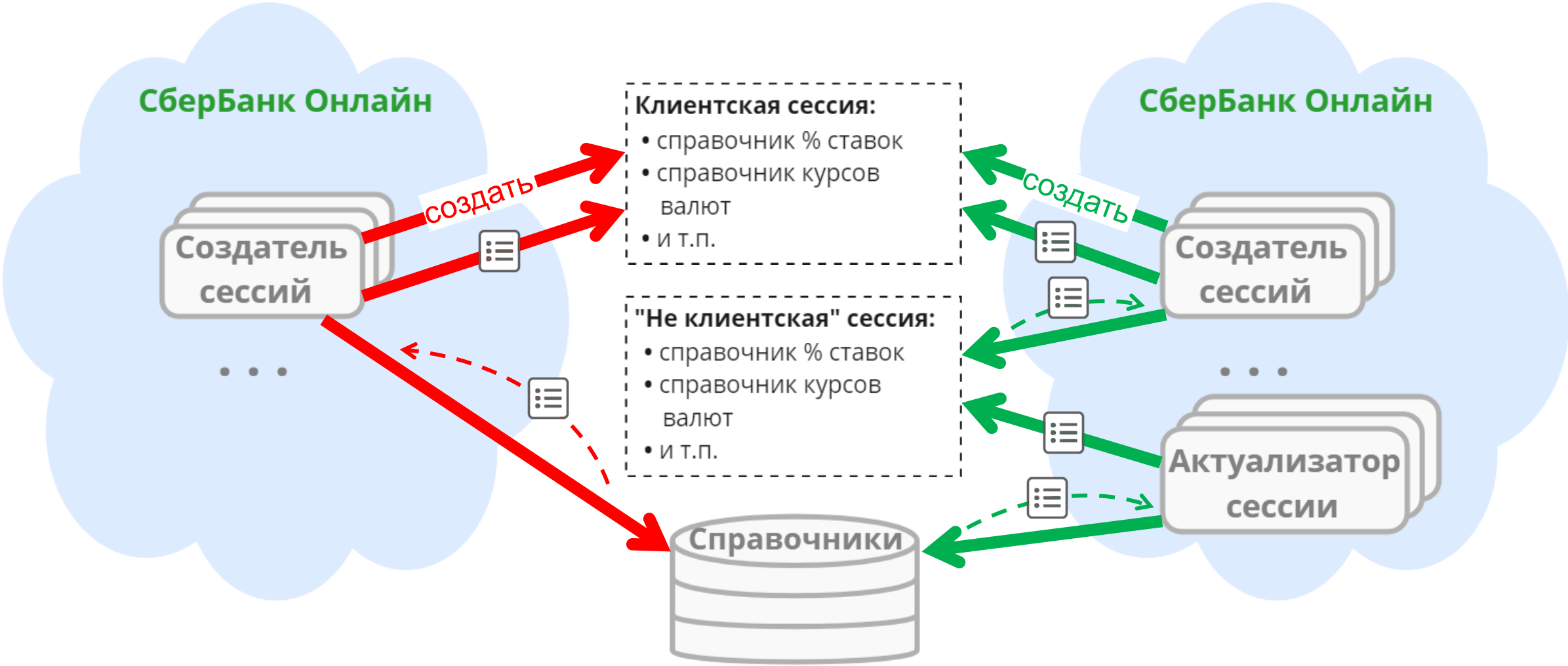
Use case: комбинация клиентской и не клиентской сессий



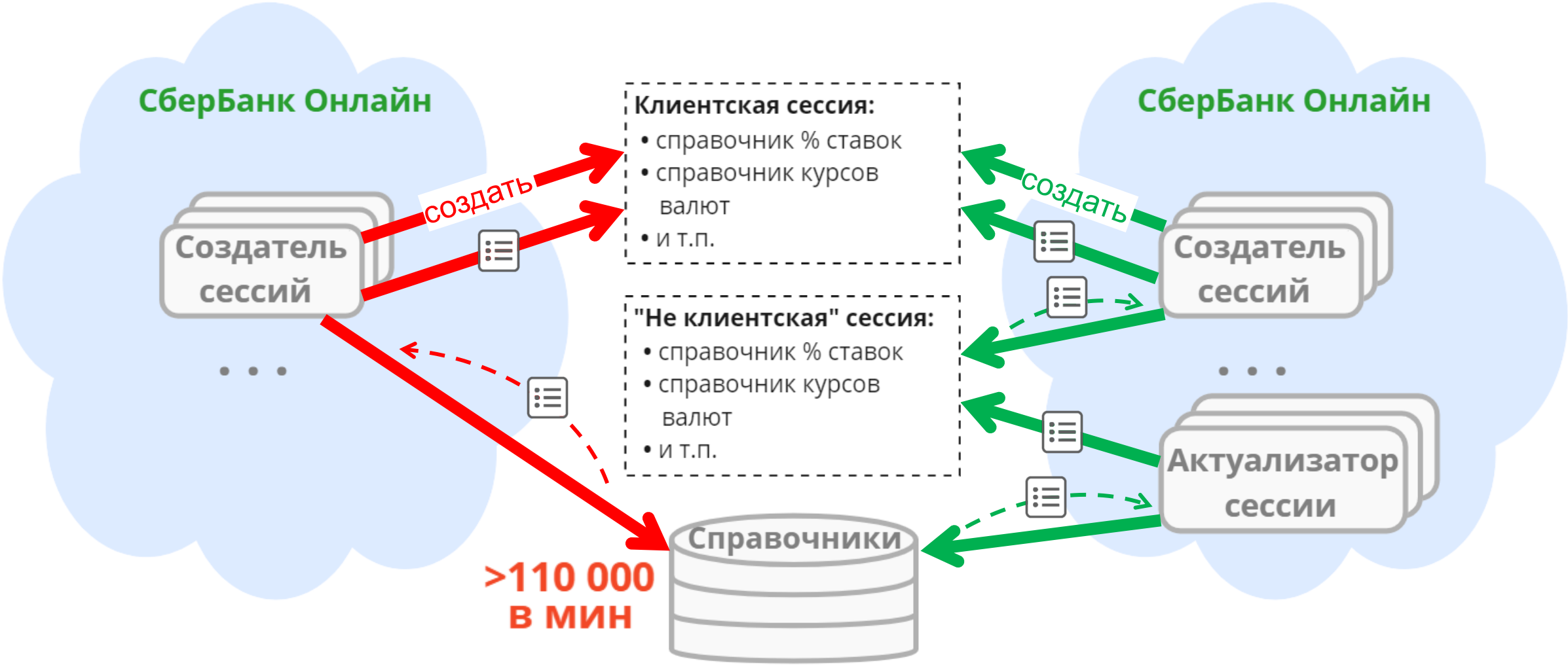
Use case: комбинация клиентской и не клиентской сессий



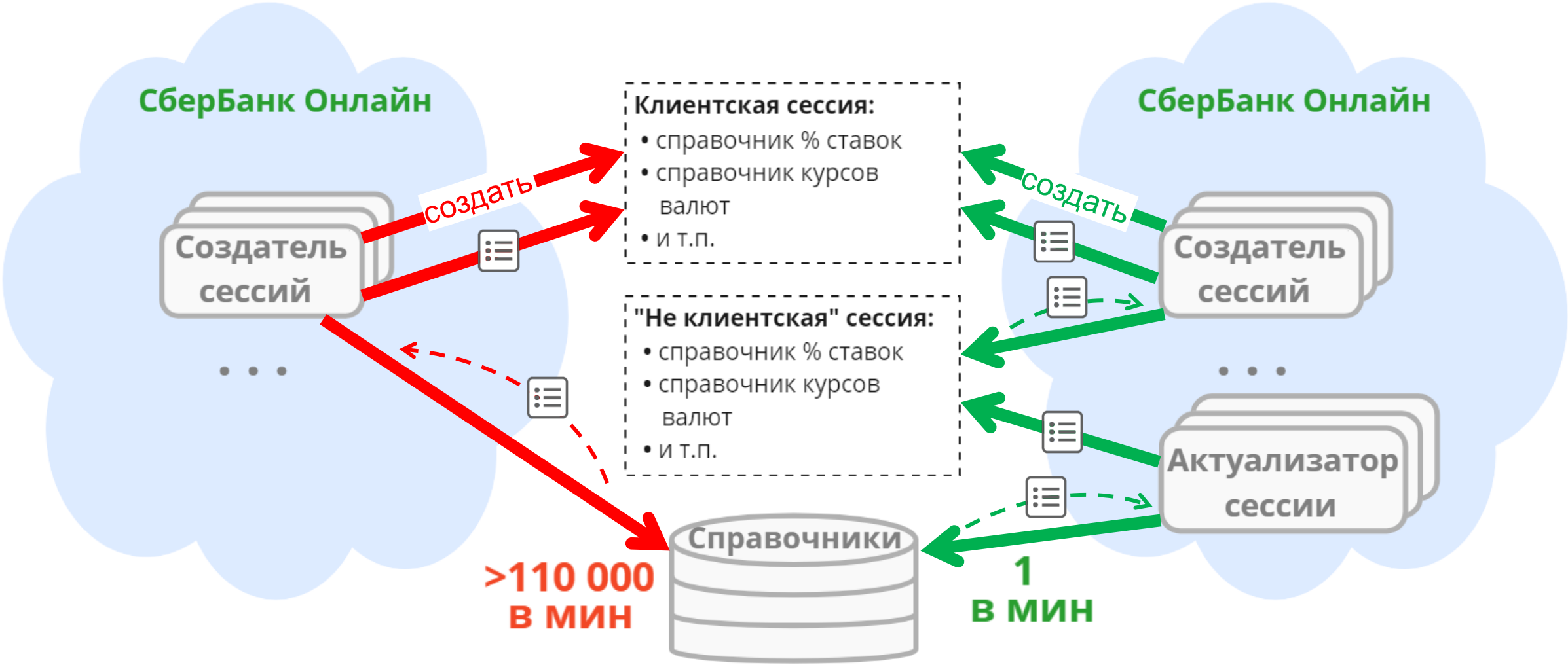
Use case: комбинация клиентской и не клиентской сессий



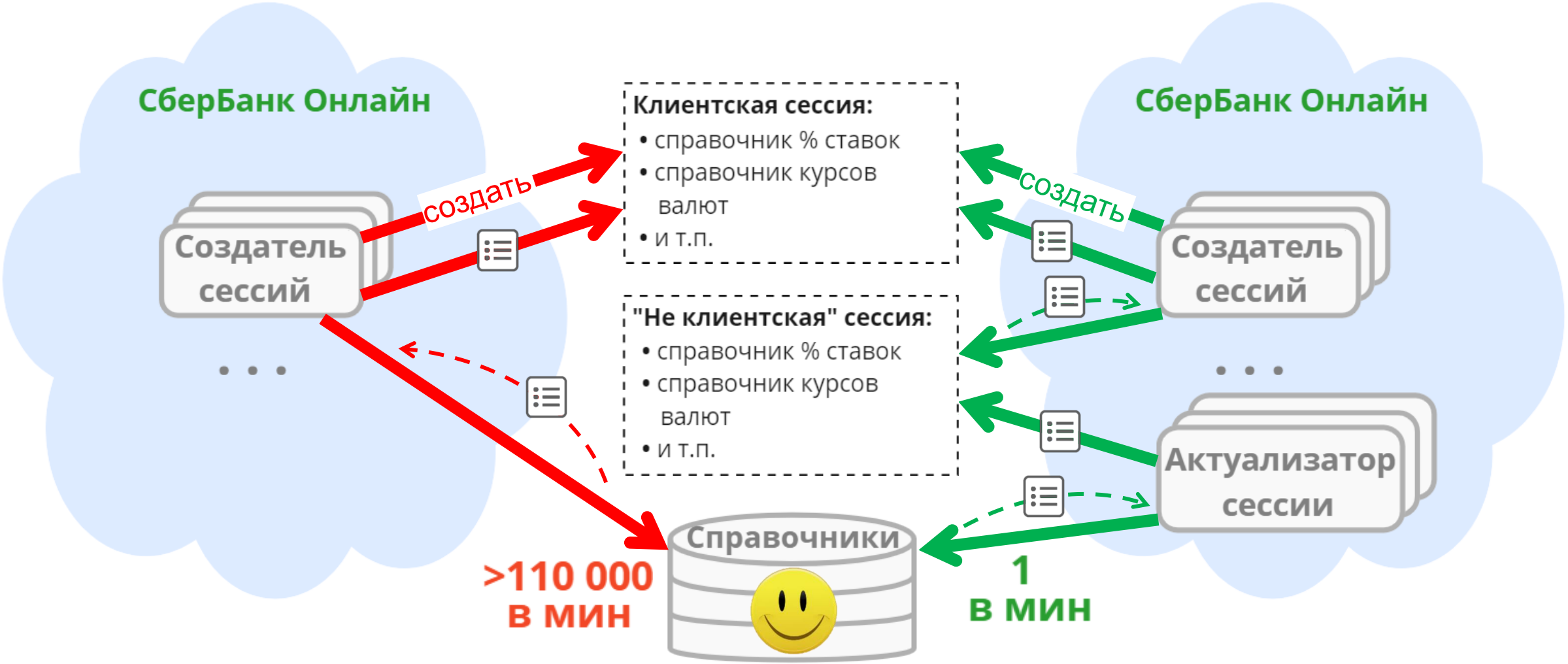
Use case: комбинация клиентской и не клиентской сессий



Use case: комбинация клиентской и не клиентской сессий



Use case: комбинация клиентской и не клиентской сессий



Что получим от «не клиентских» сессий?

Мы надеемся

- фича будет востребована
- потребителей станет больше

Особенно вместе с

- репликацицей данных
- авторизацией доступа к секциям

Выводы

1. Не сериализуйте байтовые массивы в JSON
2. Берите One Nio для Java сериализации - не ошибётесь
3. Посмотрите мой benchmark с примерами Java сериализации
4. Не развёртывайте stateful сервисы в контейнерах
5. Подумайте, стоит ли продолжать использовать западное ПО

Выводы

1. **Не сериализуйте байтовые массивы в JSON**
   JSON – текст, а byte[] в текст приходится преобразовывать.
2. Берите One Nio для Java сериализации - не ошибётесь
3. Посмотрите мой benchmark с примерами Java сериализации
4. Не развёртывайте stateful сервисы в контейнерах
5. Подумайте, стоит ли продолжать использовать западное ПО

Выводы

1. Не сериализуйте байтовые массивы в JSON
2. **Берите One Nio для Java сериализации - не ошибётесь**
   ИМХО, One Nio – лучшее решение для Java сериализации.
3. Посмотрите мой benchmark с примерами Java сериализации
4. Не развёртывайте stateful сервисы в контейнерах
5. Подумайте, стоит ли продолжать использовать западное ПО

Выводы

1. Не сериализуйте байтовые массивы в JSON
2. Берите One Nio для Java сериализации - не ошибётесь

3. **Посмотрите мой benchmark с примерами Java сериализации**
   https://github.com/chernov-af/serializers-benchmark
   Конкретные примеры, как разными библиотеками сериализовывать данные в Java.
   Для One Nio там есть один оптимизирующий трюк.

4. Не развёртывайте stateful сервисы в контейнерах
5. Подумайте, стоит ли продолжать использовать западное ПО

Выводы

1. Не сериализуйте байтовые массивы в JSON
2. Берите One Nio для Java сериализации - не ошибётесь
3. Посмотрите мой benchmark с примерами Java сериализации

4. **Не развёртывайте stateful сервисы в контейнерах**
   Системы управления контейнерами изначально спроектированы для развёртывания stateless сервисов, и это они делают хорошо.

5. Подумайте, стоит ли продолжать использовать западное ПО

Выводы

1. Не сериализуйте байтовые массивы в JSON
2. Берите One Nio для Java сериализации - не ошибётесь
3. Посмотрите мой benchmark с примерами Java сериализации
4. Не развёртывайте stateful сервисы в контейнерах

5. **Подумайте, стоит ли продолжать использовать западное ПО**
   Риски недоступности такого ПО, риски безопасности, да и просто не комфортно.
   Переходите на отечественное ПО, где это возможно.

Спасибо!

Андрей Чернов

Java архитектор в СберТех

@ chernovaf@mail.ru

chernovaf