ТИНЬКОФФ

Оптимизация join’а в Kafka Streams

Или как мы построили самобалансирующийся мониторинг обработки топиков Kafka

с помощью самой Kafka

Никита Рьянов, TeamLead
Тинькофф «АО»
github.com/nryanov

24.04.2024

О чем пойдет речь

- Постановка задачи
- Актуальность: что такое и как считать?
- Исходное решение
- С какими сложностями столкнулись
- Устраняем часть проблем
- Подключаем Kafka Streams
- Оптимизируем join
- Выводы

Постановка задачи

Хотим мониторить состояние обработки топиков

ФТ

- Показываем метрику лага
- Можем выбрать SLA
- Метрику сохраняем в Prometheus

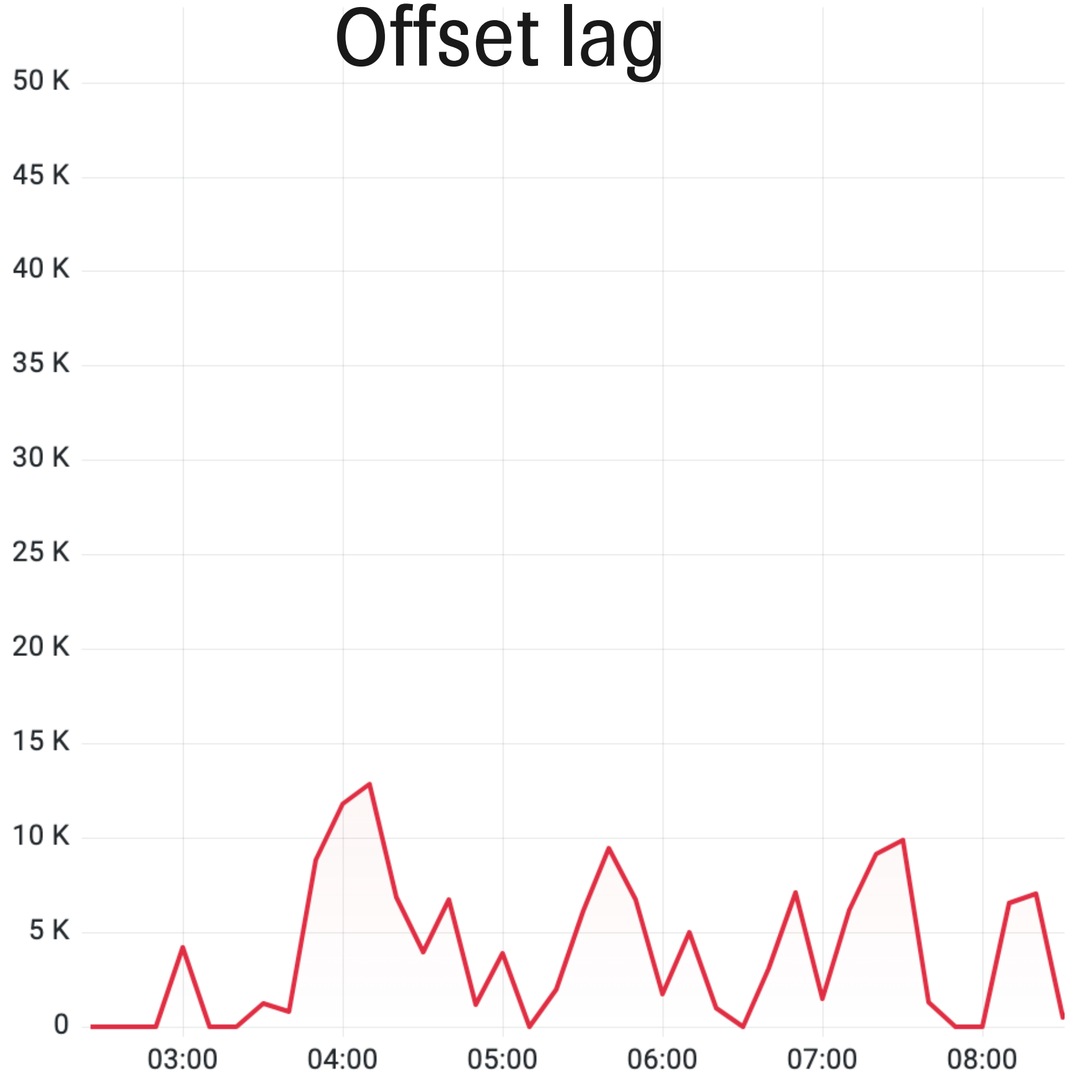
НФТ

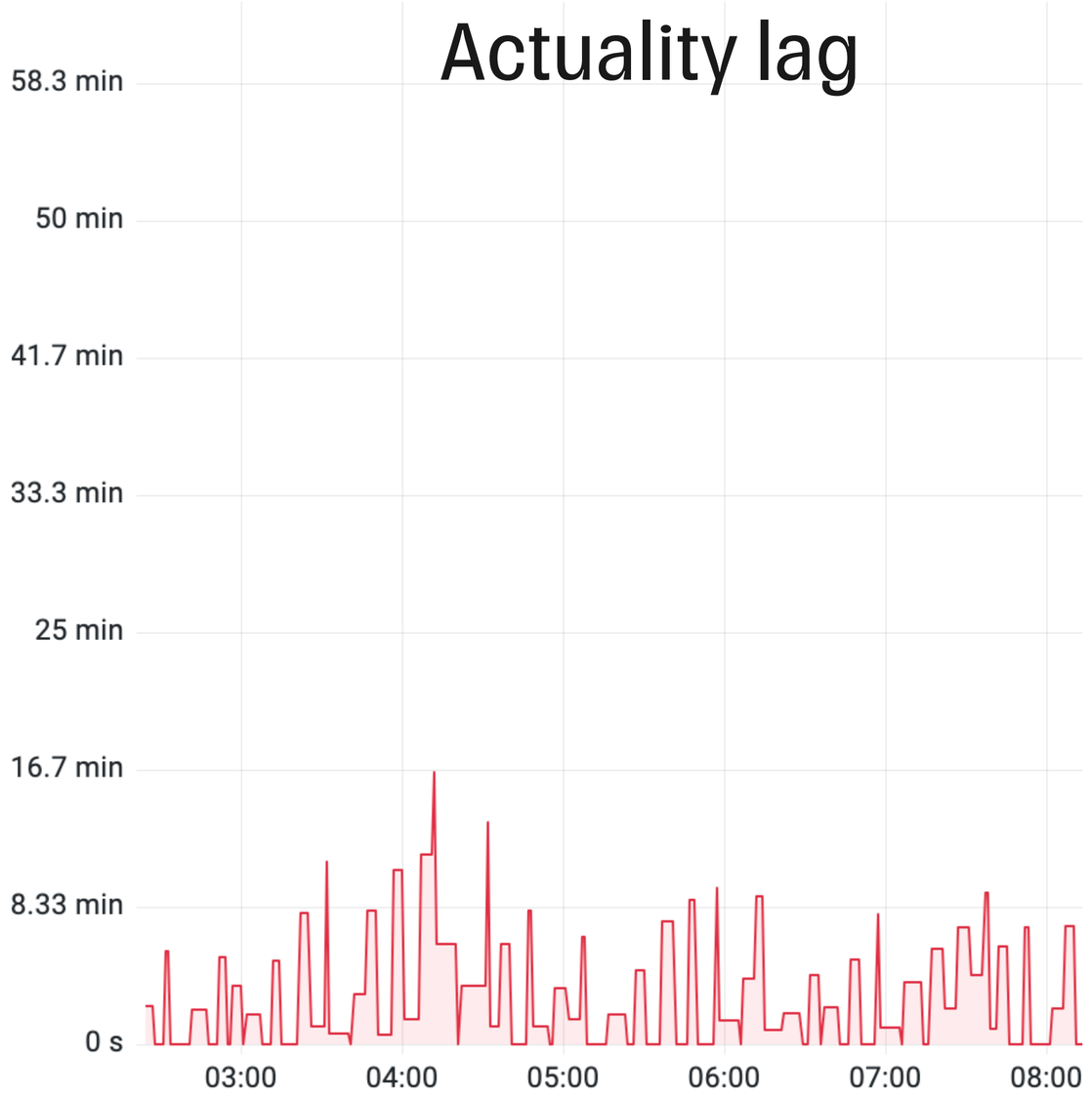
- Масштабируемость
- Устойчивость к падениям
- Метрика понятна бизнесу

Актуальность: что такое и как считать?

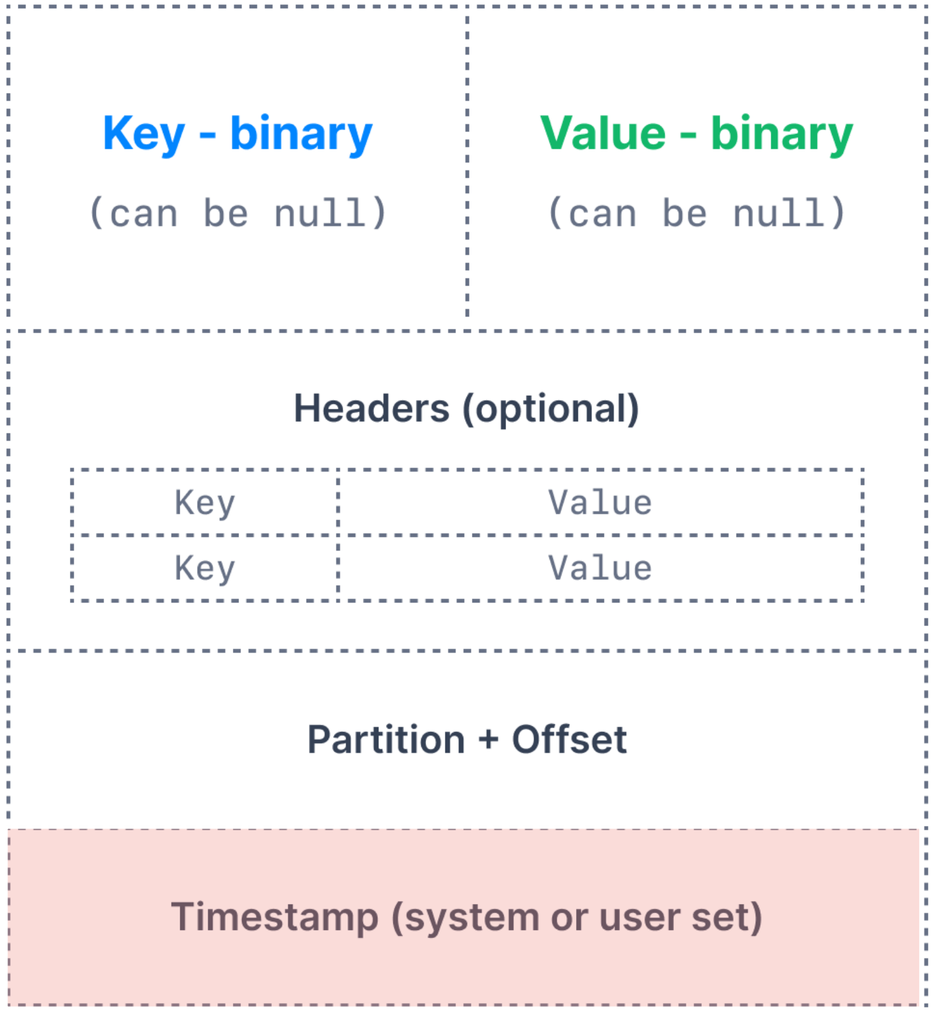
Как ответить на вопрос: «на сколько мы отстаем?»

Актуальность: что такое и как считать?



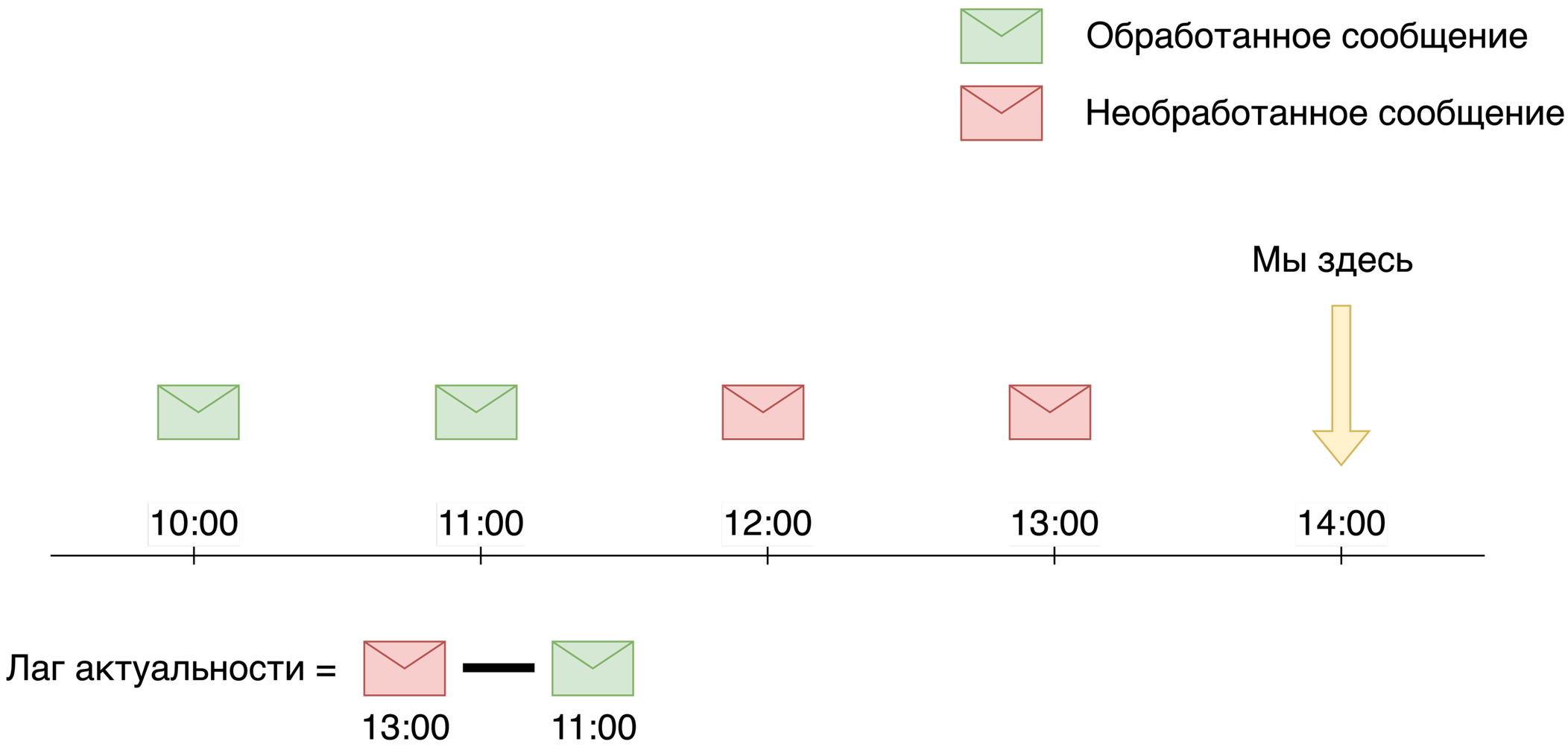


Актуальность: что такое и как считать?



Важно: message.timestamp.type=LogAppend

Актуальность: что такое и как считать?



Исходное решение: последнее обработанное сообщение

1. Получили пачку сообщений

Исходное решение: последнее обработанное сообщение

1. Получили пачку сообщений
2. Обработали

Исходное решение: последнее обработанное сообщение

1. Получили пачку сообщений
2. Обработали
3. Сохранили последнее сообщение пачки в compact-топик

Исходное решение: последнее необработанное сообщение

1. KafkaAdminClient: получили последние оффсеты

Исходное решение: последнее необработанное сообщение

1. KafkaAdminClient: получили последние оффсеты
2. KafkaConsumer: прочитали последние сообщения

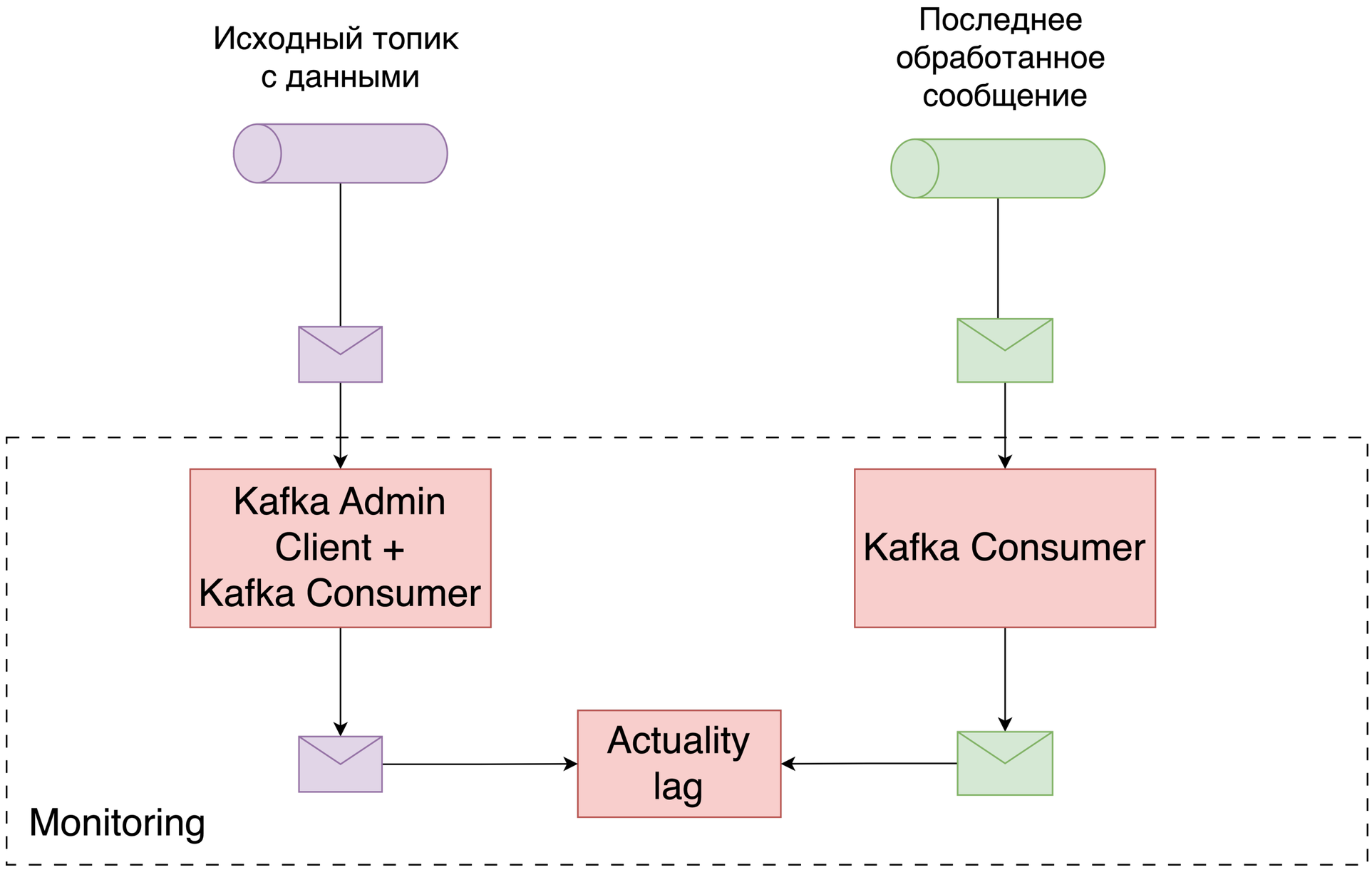
Актуально до версии 3.0.0

KIP-734

Исходное решение: последнее необработанное сообщение

1. KafkaAdminClient: получили последние оффсеты
2. KafkaConsumer: прочитали последние сообщения
3. Сохранили результат в in-memory state

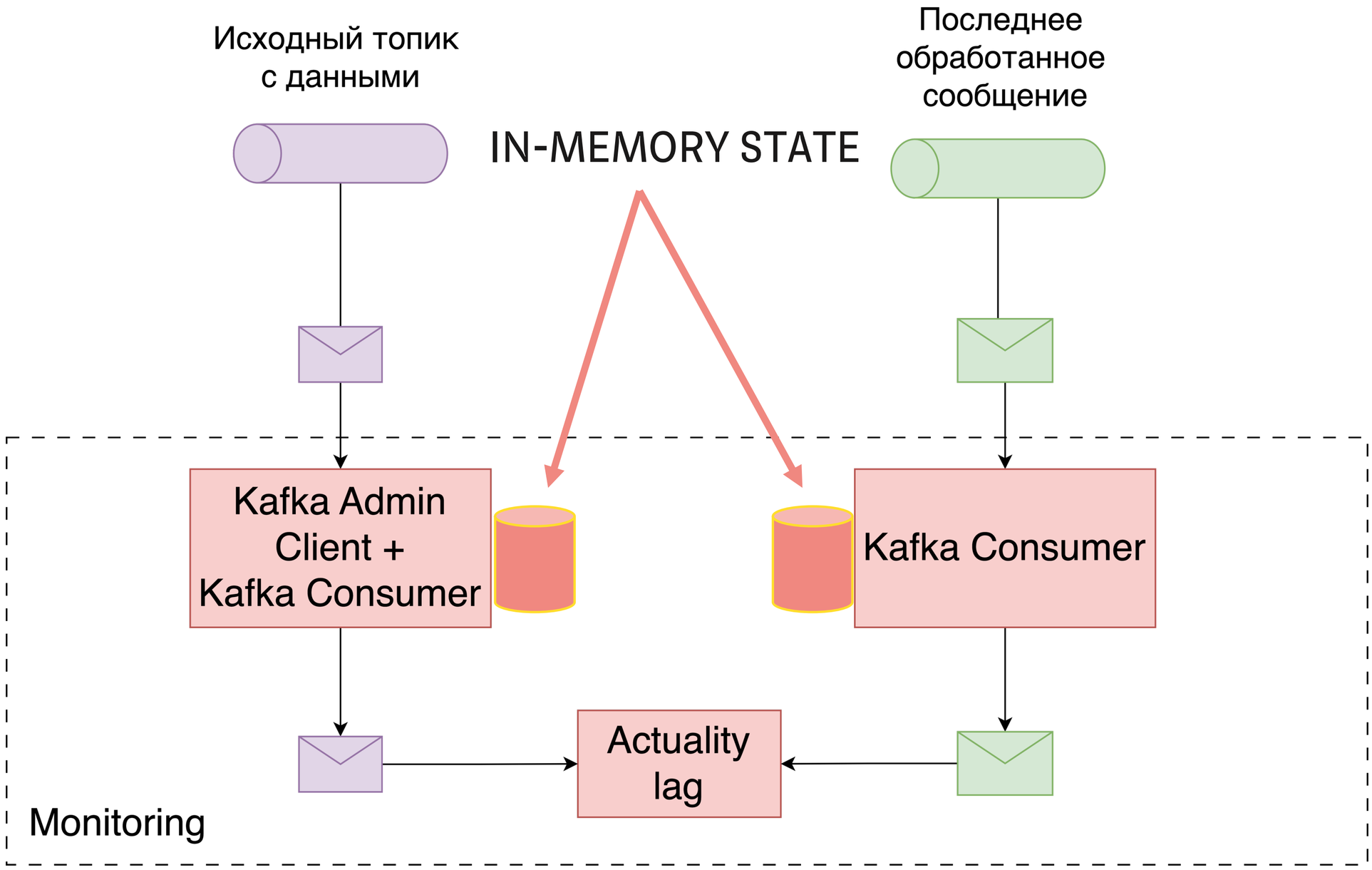
Исходное решение: вид сверху



С какими сложностями столкнулись

- Любое падение -> долгий рестарт

С какими сложностями столкнулись: долгий рестарт



Устраняем часть проблем: долгий рестарт

Проблема: долго получаем необработанные сообщения

Устраняем часть проблем: долгий рестарт

Проблема: долго получаем необработанные сообщения

Решение #1: сохраняем результат в compact-topic

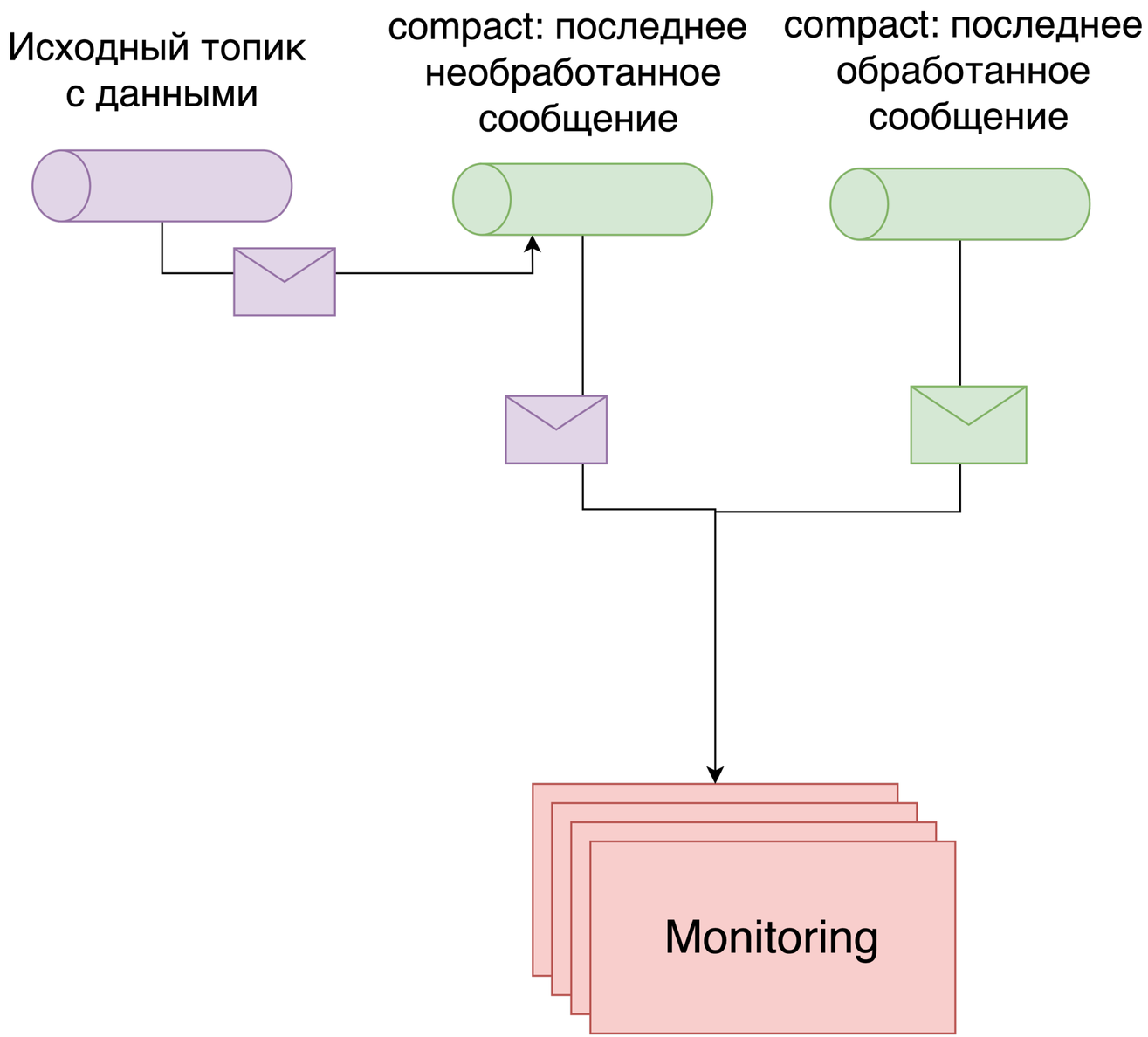
Устраняем часть проблем: долгий рестарт

Проблема: долго получаем необработанные сообщения

Решение #1: сохраняем результат в compact-topic

Решение #2: добавим реплик

Устраняем часть проблем: долгий рестарт



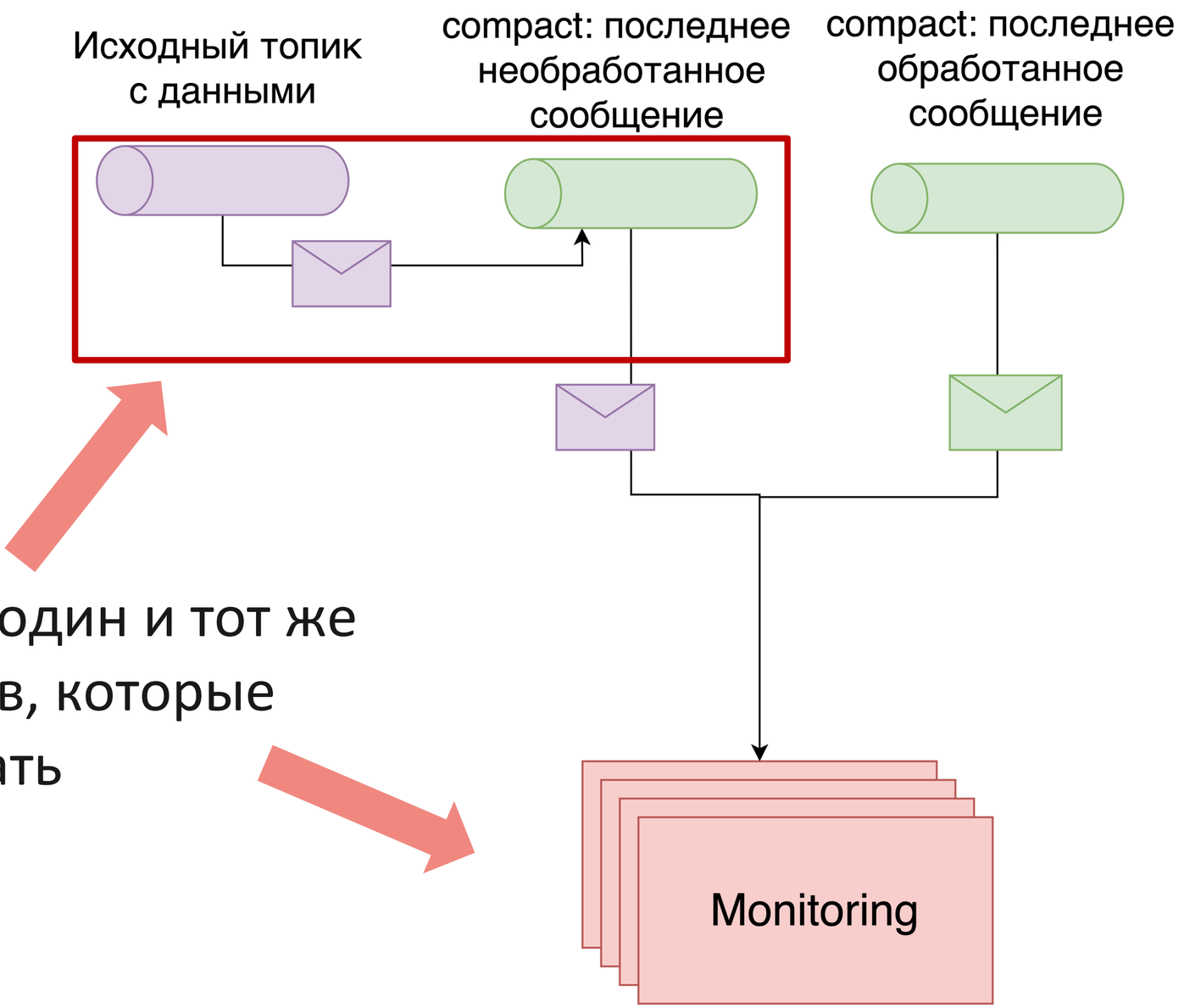
С какими сложностями столкнулись

- Любое падение -> долгий рестарт
- Дублирование метрик

Устраняем часть проблем: дублирование метрик

Проблема: обрабатываем весь список исходных топиков

Устраняем часть проблем: дублирование метрик



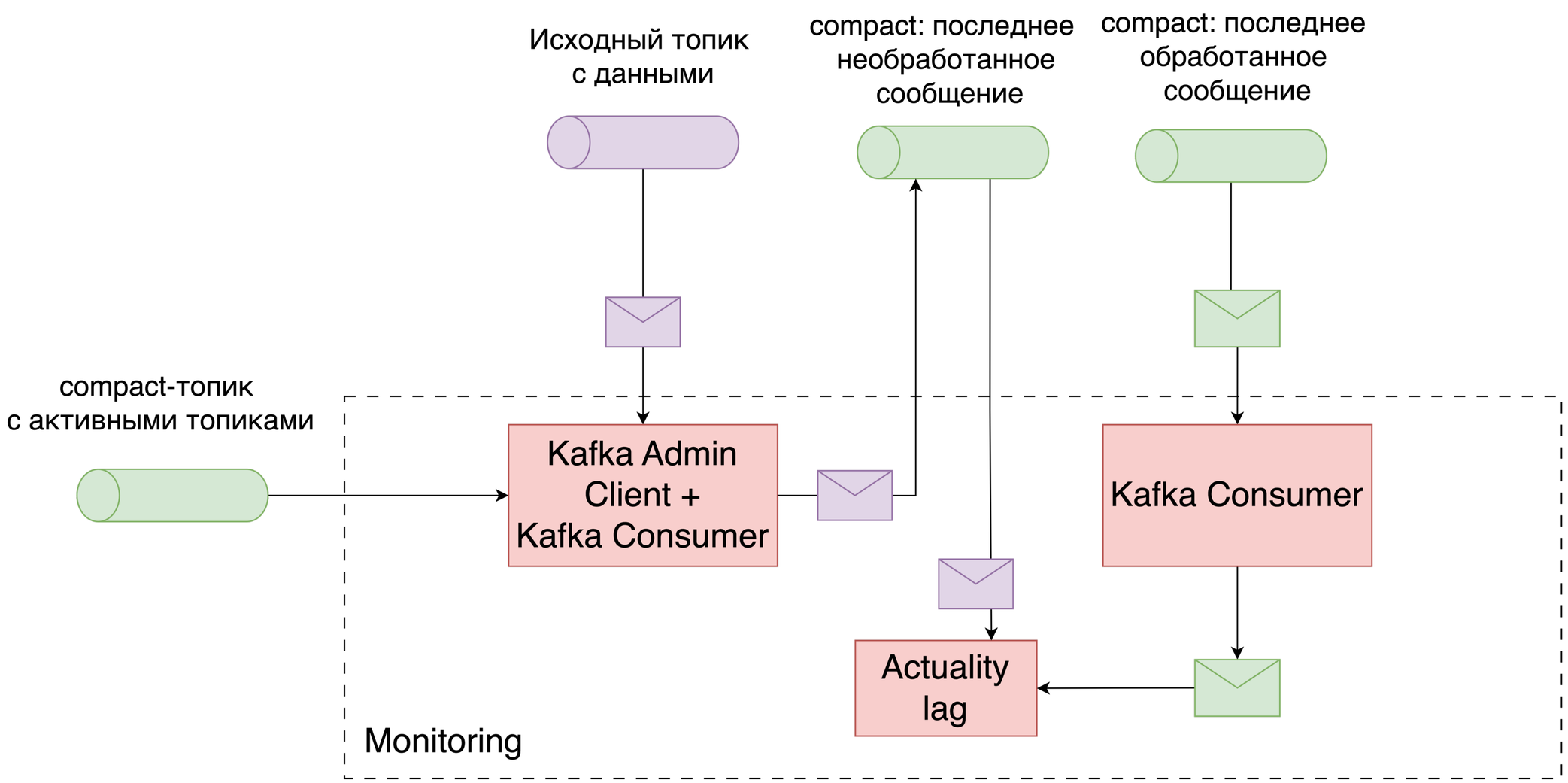
У всех реплик один и тот же список топиков, которые надо обработать

Устраняем часть проблем: дублирование метрик

Проблема: обрабатываем весь список исходных топиков

Решение: партиционируем список (compact topic)

Устраняем часть проблем: вид сверху



Подключаем Kafka Streams

Основная задача:
выполнить join
нескольких топиков

Подключаем Kafka Streams

compact-топик
с активными топиками

compact: последнее
необработанное
сообщение

compact: последнее
обработанное
сообщение

```
SELECT *
FROM активные_топики t1
         LEFT JOIN необработанные_сообщения t2 ON t1.topic_name = t2.topic_name
         LEFT JOIN последние_обработанные_сообщения t3
                   ON t2.topic_name = t3.topic_name AND t2.partition_id = t3.partition_id;
```

Подключаем Kafka Streams

compact-топик
с активными топиками

compact: последнее
необработанное
сообщение

compact: последнее
обработанное
сообщение

```
SELECT *
FROM активные_топики t1
         LEFT JOIN необработанные_сообщения t2 ON t1.topic_name = t2.topic_name
         LEFT JOIN последние_обработанные_сообщения t3
                   ON t2.topic_name = t3.topic_name AND t2.partition_id = t3.partition_id;
```

Подключаем Kafka Streams

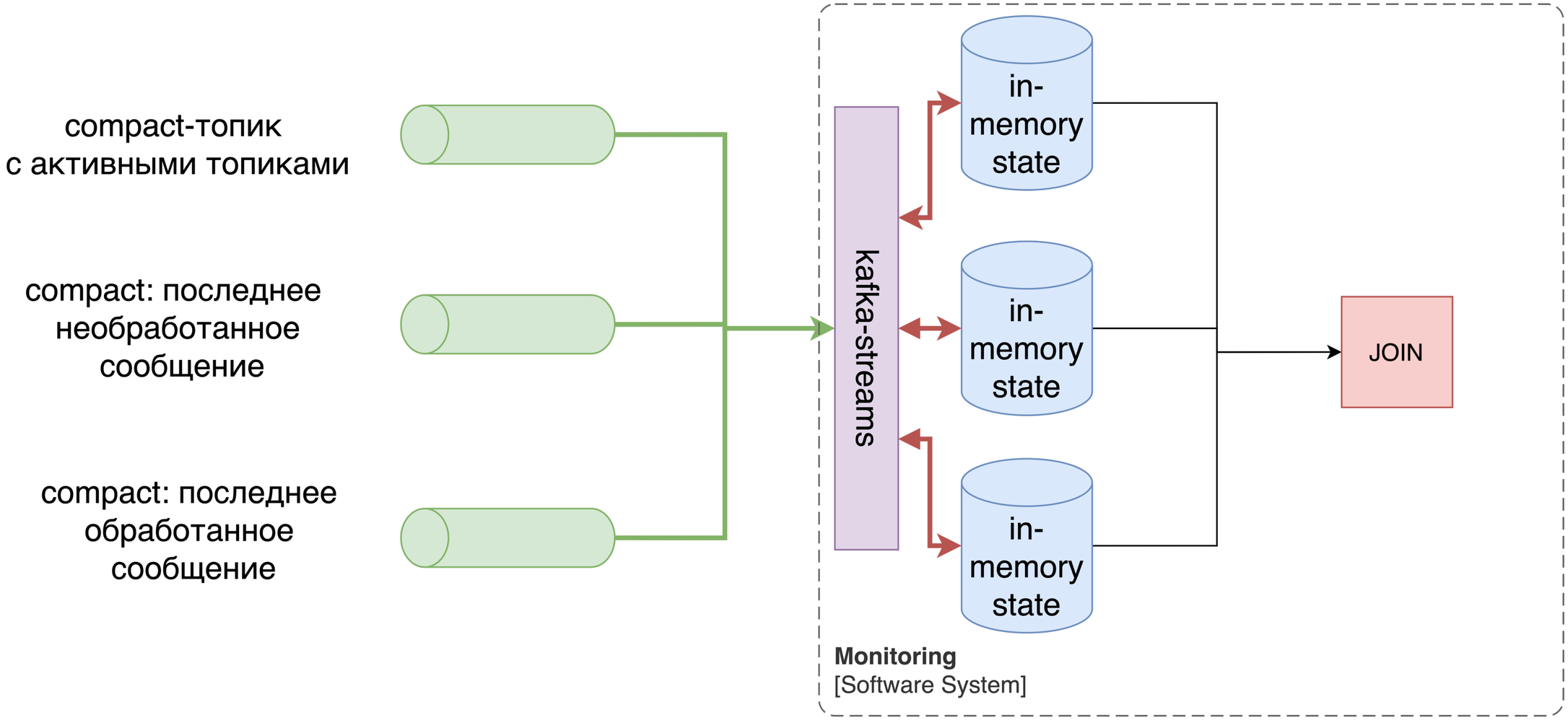
compact-топик
с активными топиками

compact: последнее
необработанное
сообщение

compact: последнее
обработанное
сообщение

```
SELECT *
FROM активные_топики t1
         LEFT JOIN необработанные_сообщения t2 ON t1.topic_name = t2.topic_name
         LEFT JOIN последние_обработанные_сообщения t3
                   ON t2.topic_name = t3.topic_name AND t2.partition_id = t3.partition_id;
```

Подключаем Kafka Streams: ожидаемая схема

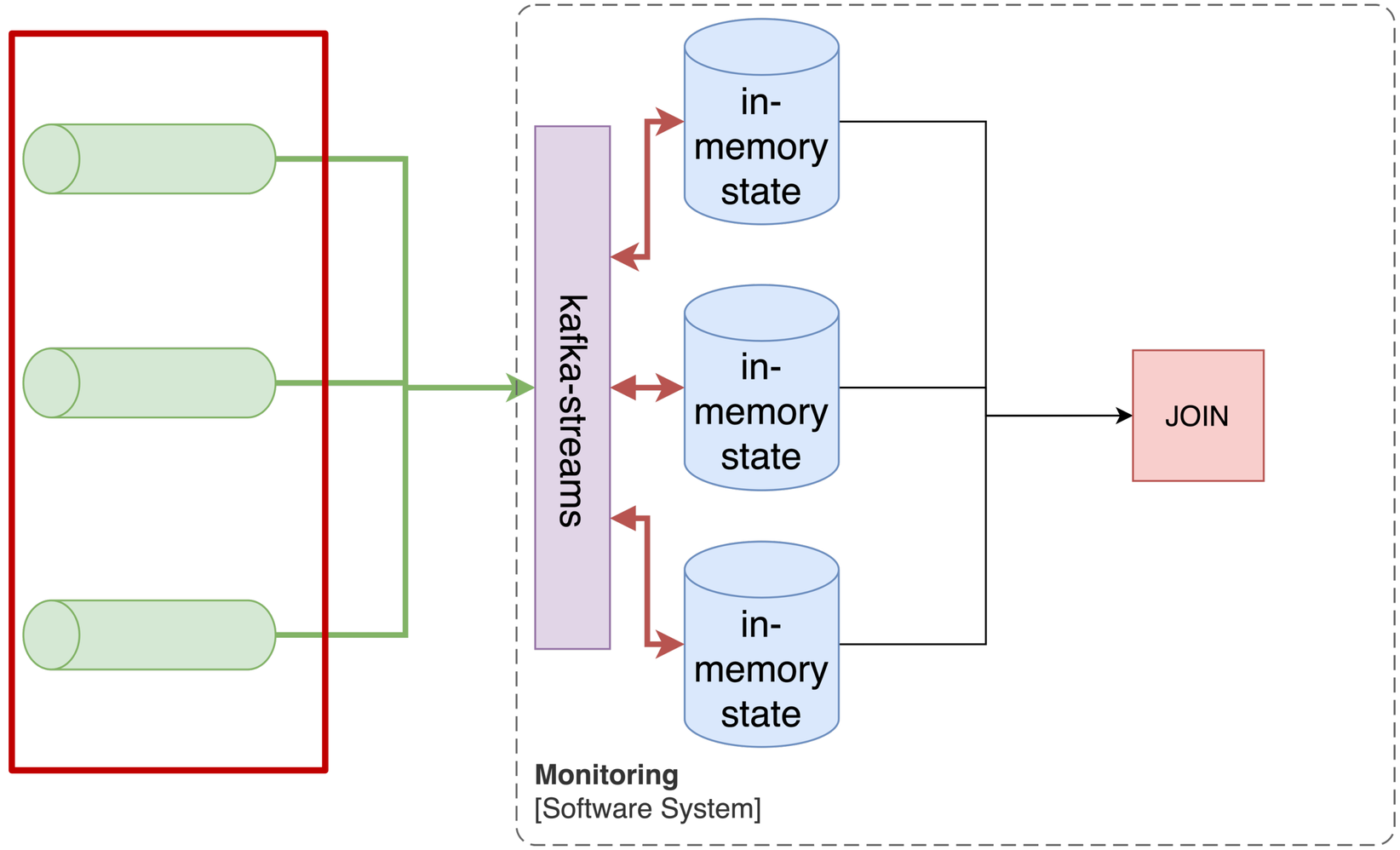


Подключаем Kafka Streams: ожидаемая схема

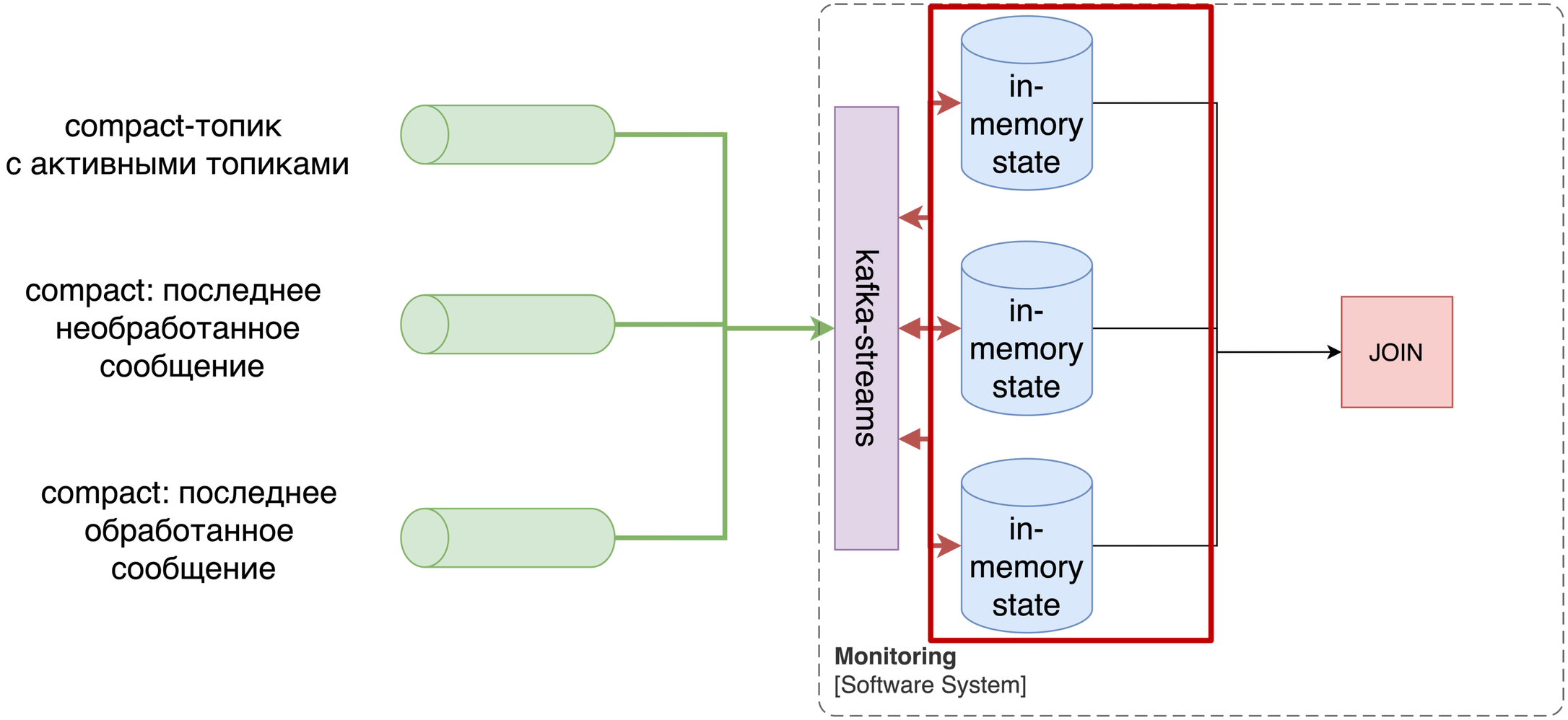
compact-топик
с активными топиками

compact: последнее
необработанное
сообщение

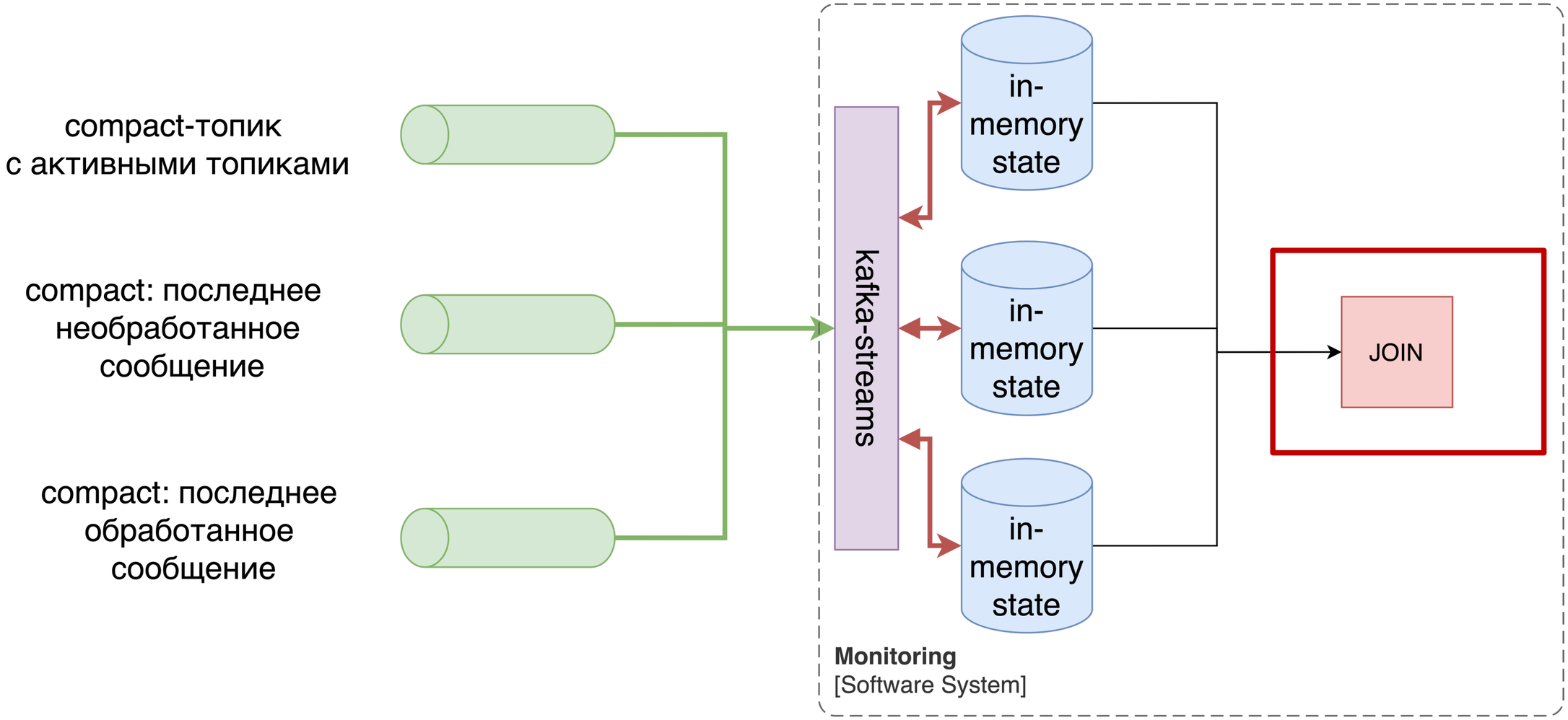
compact: последнее
обработанное
сообщение



Подключаем Kafka Streams: ожидаемая схема



Подключаем Kafka Streams: ожидаемая схема



Подключаем Kafka Streams: создаем localState

```
var storeSupplier = Stores.inMemoryKeyValueStore(storeName);

var storeBuilder = Stores
    .keyValueStoreBuilder(storeSupplier, keySerde, valueSerde)
    .withLoggingEnabled(Map.of())
    .withCachingDisabled();

builder.addStateStore(storeBuilder);
```

Подключаем Kafka Streams: создаем localState

```
var storeSupplier = Stores.inMemoryKeyValueStore(storeName);

var storeBuilder = Stores
    .keyValueStoreBuilder(storeSupplier, keySerde, valueSerde)
    .withLoggingEnabled(Map.of())
    .withCachingDisabled();

builder.addStateStore(storeBuilder);
```

Подключаем Kafka Streams: создаем localState

```
var storeSupplier = Stores.inMemoryKeyValueStore(storeName);

var storeBuilder = Stores
    .keyValueStoreBuilder(storeSupplier, keySerde, valueSerde)
    .withLoggingEnabled(Map.of())
    .withCachingDisabled();

builder.addStateStore(storeBuilder);
```

Подключаем Kafka Streams: наполняем localState

```
topicKTable
    .transformValues(() -> createStoreProducer(storeName), storeName)
    .toStream()
    .foreach((key, value) -> {});

private static <K, V> ValueTransformerWithKey<K, V, V> createStoreProducer(String storageName) {
    return new ValueTransformerWithKey<>() {
        private KeyValueStore<K, V> storage;
        private ProcessorContext context;

        @Override
        public void init(ProcessorContext context) {
            this.storage = (KeyValueStore<K, V>) context.getStateStore(storageName);
            this.context = context;
        }

        @Override
        public V transform(K readOnlyKey, V value) {
            storage.put(readOnlyKey, value);
            return value;
        }
}
```

Подключаем Kafka Streams: наполняем localState

```
topicKTable
    .transformValues(() -> createStoreProducer(storeName), storeName)
    .toStream()
    .foreach((key, value) -> {});

private static <K, V> ValueTransformerWithKey<K, V, V> createStoreProducer(String storageName) {
    return new ValueTransformerWithKey<>() {
        private KeyValueStore<K, V> storage;
        private ProcessorContext context;

        @Override
        public void init(ProcessorContext context) {
            this.storage = (KeyValueStore<K, V>) context.getStateStore(storageName);
            this.context = context;
        }

        @Override
        public V transform(K readOnlyKey, V value) {
            storage.put(readOnlyKey, value);
            return value;
        }
}
```

Подключаем Kafka Streams: наполняем localState

```
topicKTable
    .transformValues(() -> createStoreProducer(storeName), storeName)
    .toStream()
    .foreach((key, value) -> {});

private static <K, V> ValueTransformerWithKey<K, V, V> createStoreProducer(String storageName) {
    return new ValueTransformerWithKey<>() {
        private KeyValueStore<K, V> storage;
        private ProcessorContext context;

        @Override
        public void init(ProcessorContext context) {
            this.storage = (KeyValueStore<K, V>) context.getStateStore(storageName);
            this.context = context;
        }

        @Override
        public V transform(K readOnlyKey, V value) {
            storage.put(readOnlyKey, value);
            return value;
        }
}
```

Подключаем Kafka Streams: interactive queries

```
var storeType = QueryableStoreTypes.keyValueStore();
var storeFilter = StoreQueryParameters.fromNameAndType(storeName, storeType);

ReadOnlyKeyValueStore<Key, Value> storage = kafkaStreams.store(storeFilter);
```

Подключаем Kafka Streams: interactive queries

```
var storeType = QueryableStoreTypes.keyValueStore();
var storeFilter = StoreQueryParameters.fromNameAndType(storeName, storeType);

ReadOnlyKeyValueStore<Key, Value> storage = kafkaStreams.store(storeFilter);
```

Подключаем Kafka Streams: сам join

```
ReadOnlyKeyValueStore<Key1, Value1> state1;
ReadOnlyKeyValueStore<Key2, Value2> state2;
ReadOnlyKeyValueStore<Key3, Value3> state3;

public void join() {
    try (
            var stateIter1 = state1.all();
            var stateIter2 = state2.all();
            var stateIter3 = state3.all();
            ) {
    // process each record as you wish
    } catch (InvalidStateStoreException e) {
        logger.warnf("State is currently migrating: %s", e.getMessage());
    }
}
```

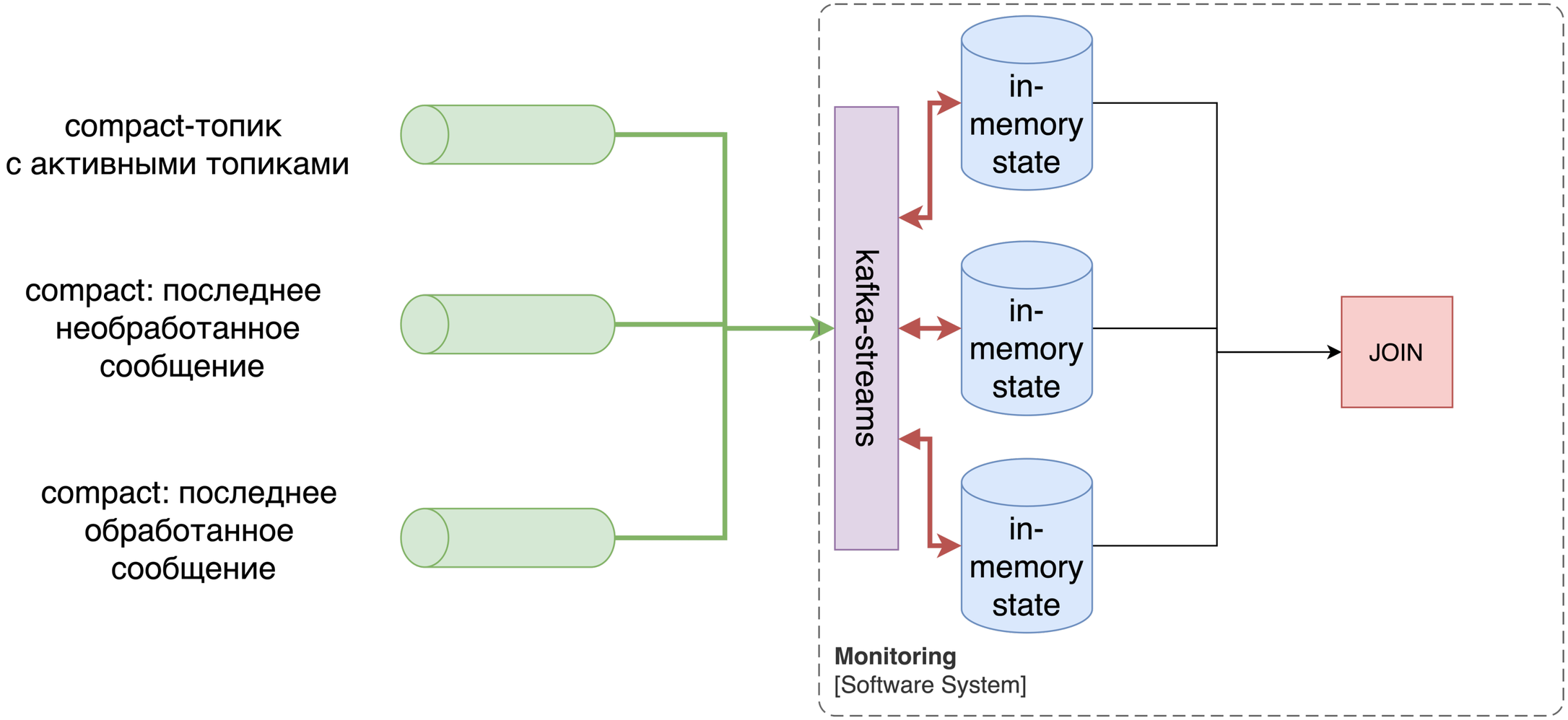
Подключаем Kafka Streams: сам join

```
ReadOnlyKeyValueStore<Key1, Value1> state1;
ReadOnlyKeyValueStore<Key2, Value2> state2;
ReadOnlyKeyValueStore<Key3, Value3> state3;

public void join() {
    try (
            var stateIter1 = state1.all();
            var stateIter2 = state2.all();
            var stateIter3 = state3.all();
            ) {
    // process each record as you wish
    } catch (InvalidStateStoreException e) {
        logger.warnf("State is currently migrating: %s", e.getMessage());
    }
}
```

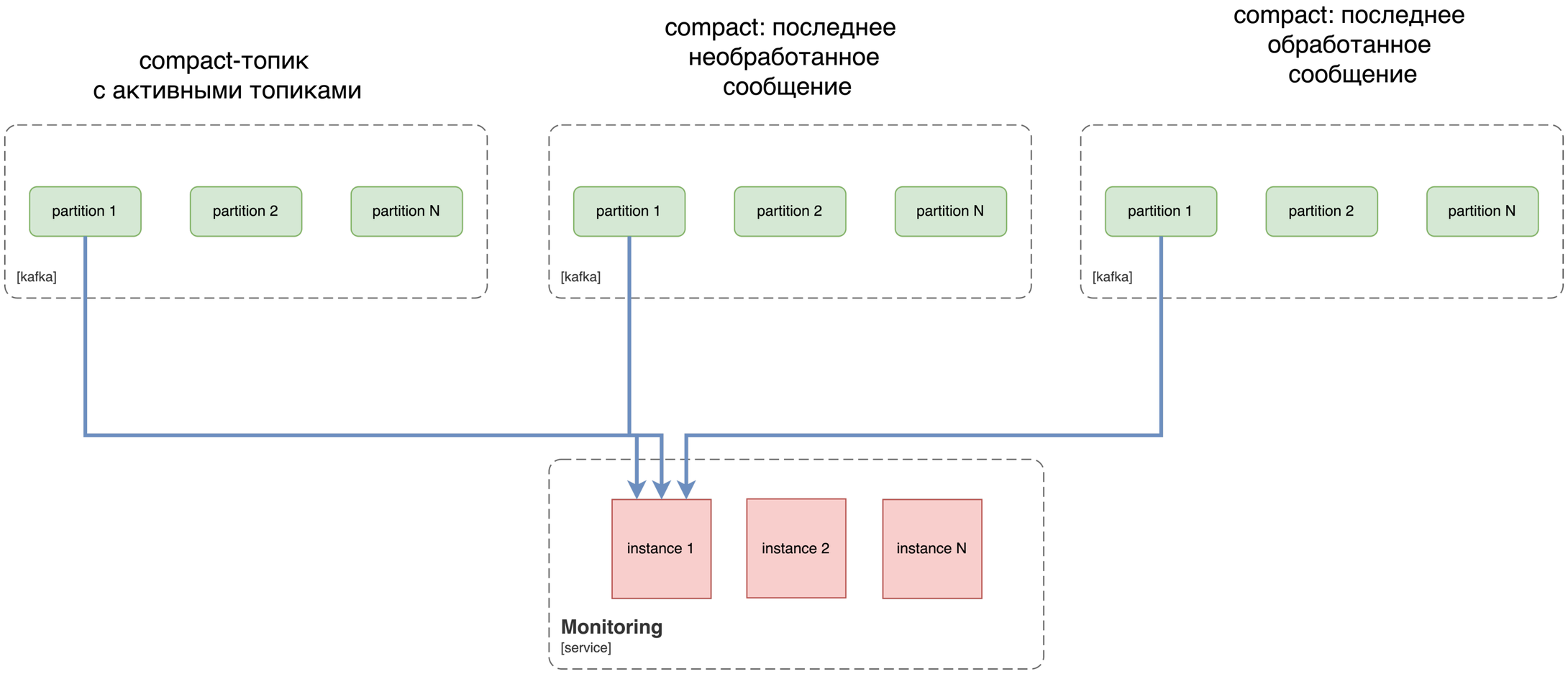
Подключаем Kafka Streams: сам join

```
ReadOnlyKeyValueStore<Key1, Value1> state1;
ReadOnlyKeyValueStore<Key2, Value2> state2;
ReadOnlyKeyValueStore<Key3, Value3> state3;

public void join() {
    try (
            var stateIter1 = state1.all();
            var stateIter2 = state2.all();
            var stateIter3 = state3.all();
            ) {
    // process each record as you wish
    } catch (InvalidStateStoreException e) {
        logger.warnf("State is currently migrating: %s", e.getMessage());
    }
}
```

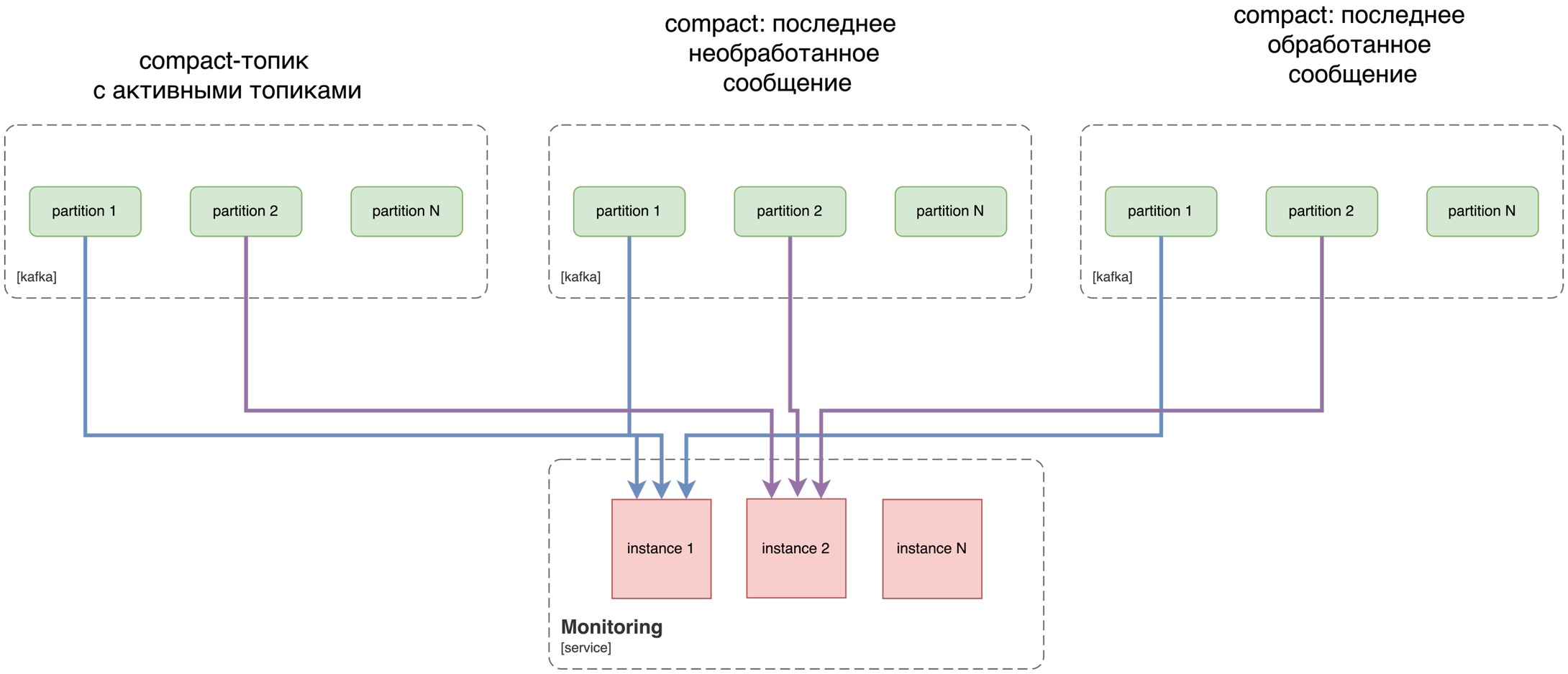
Подключаем Kafka Streams: вид сверху



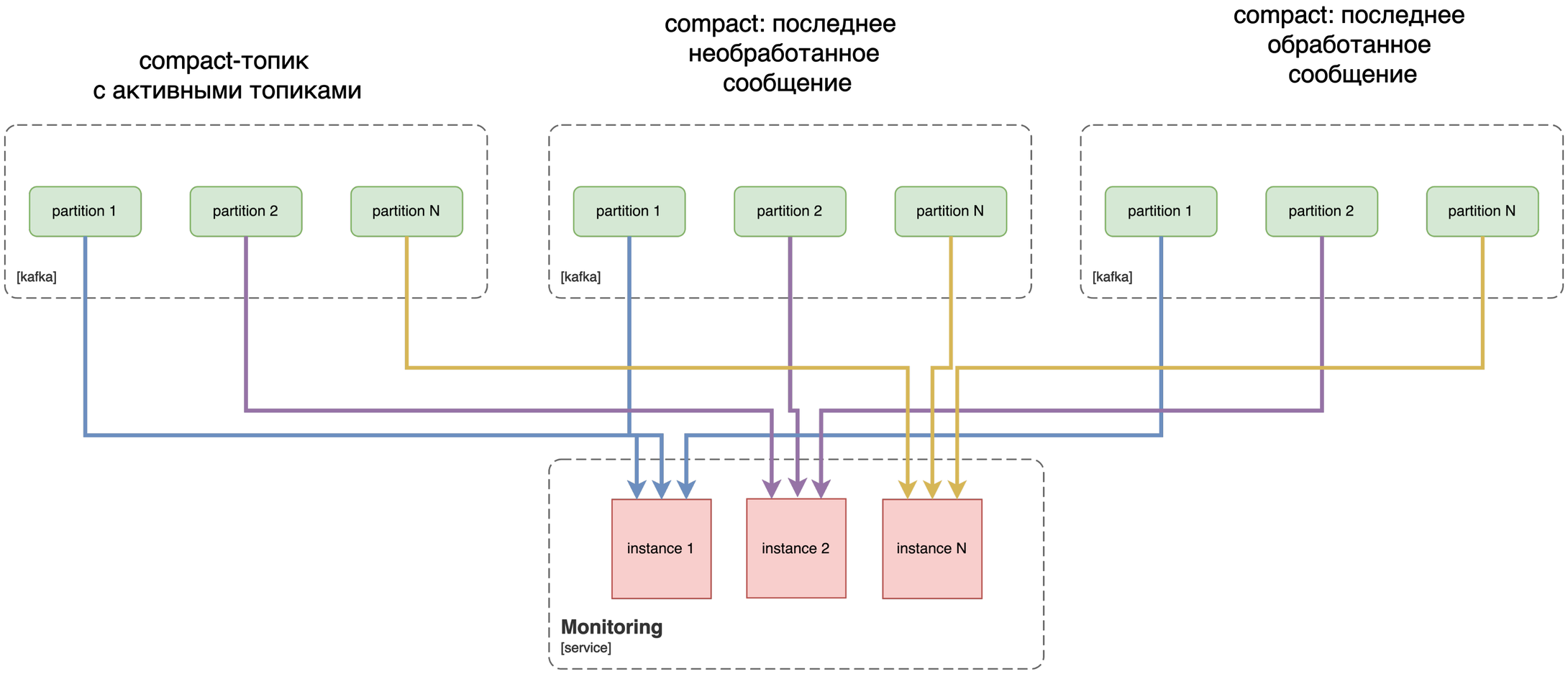
Подключаем Kafka Streams: что ожидали



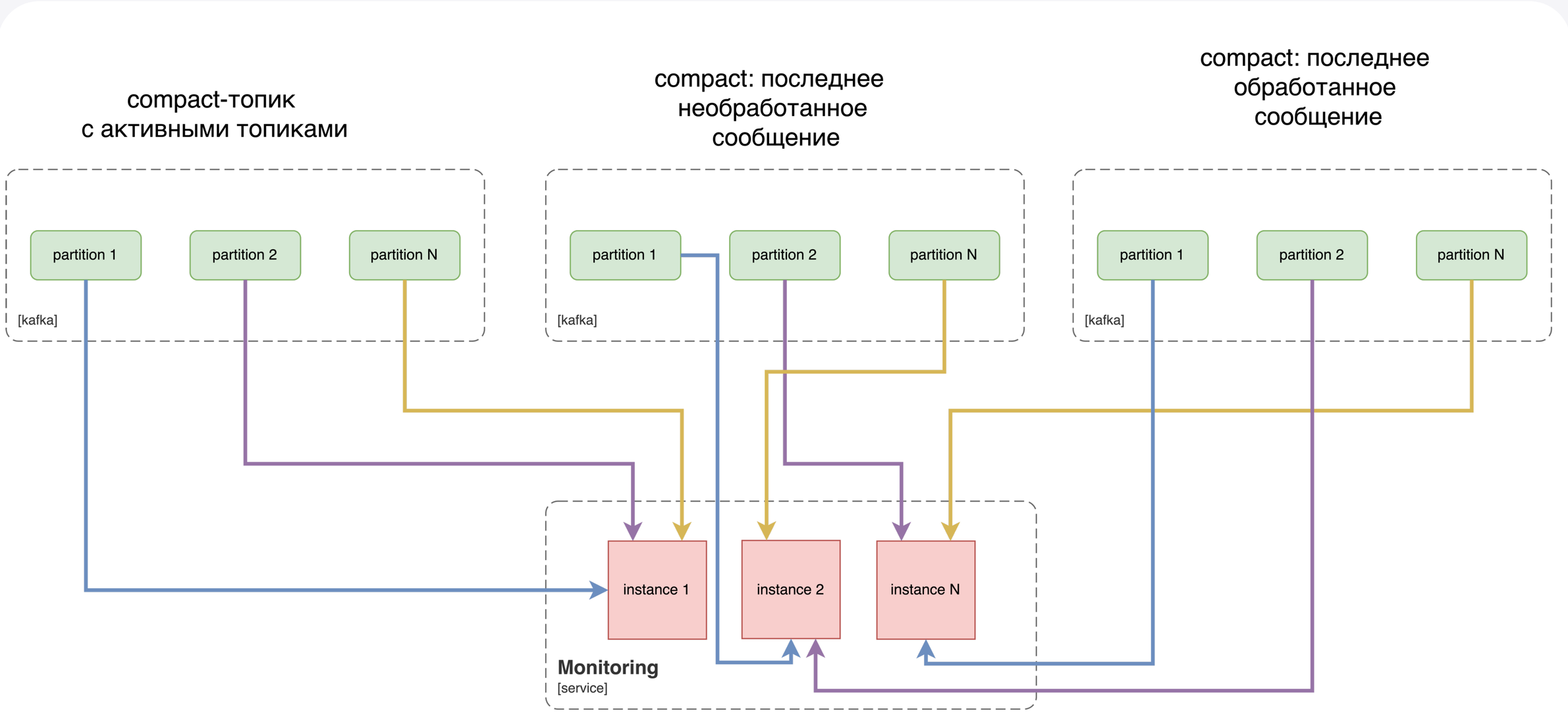
Подключаем Kafka Streams: что ожидали



Подключаем Kafka Streams: что ожидали

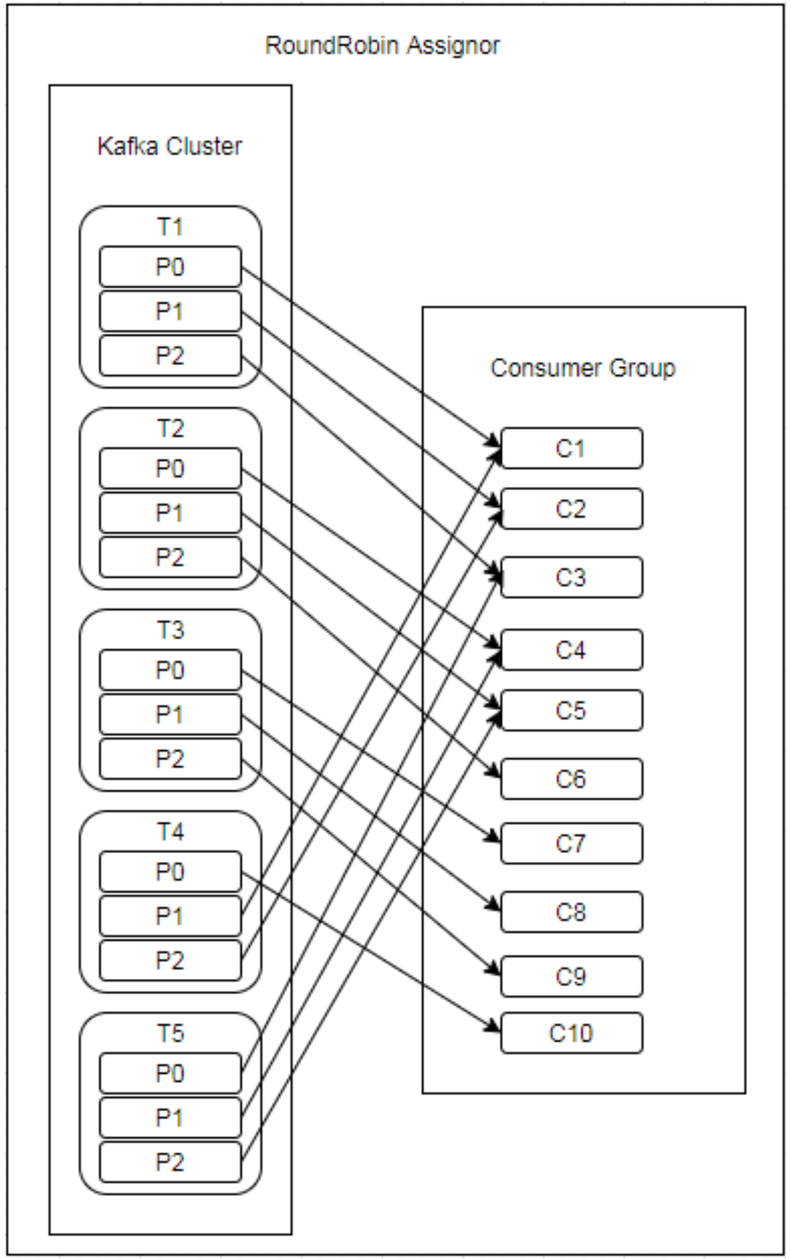


Подключаем Kafka Streams: что получили



Подключаем Kafka Streams: стратегии распределения партиций

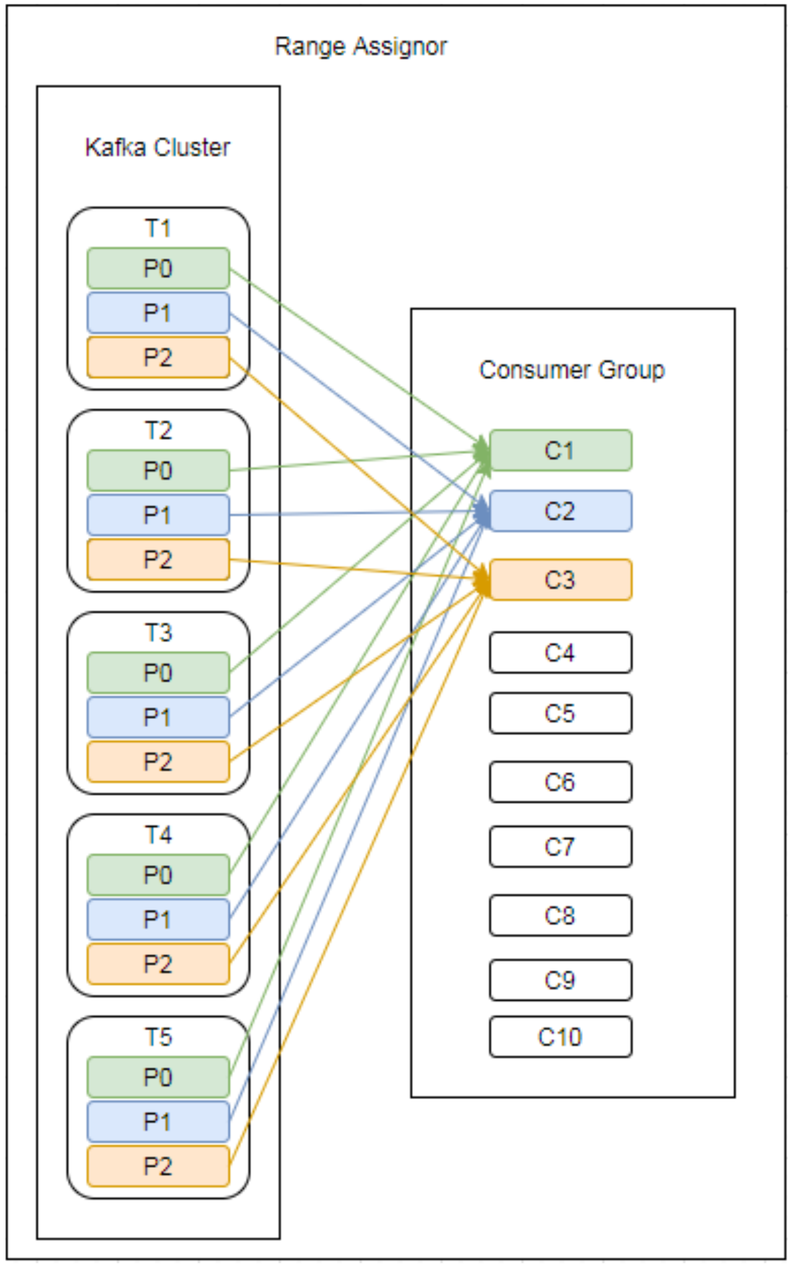
RoundRobinAssignor



https://docs.confluent.io/platform/current/installation/configuration/consumer-configs.html#partition-assignment-strategy

Подключаем Kafka Streams: стратегии распределения партиций

RangeAssignor



https://docs.confluent.io/platform/current/installation/configuration/consumer-configs.html#partition-assignment-strategy

Подключаем Kafka Streams: RangeAssignor – то, что надо!

```
Kafka Streams uses the StreamsPartitionAssignor class and doesn’t let you change to a different assignor. If you try to use a different assignor, Kafka Streams ignores it.
```

Подключаем Kafka Streams: RangeAssignor – то, что надо!

```
Kafka Streams uses the StreamsPartitionAssignor class and doesn’t let you change to a different assignor. If you try to use a different assignor, Kafka Streams ignores it.
```

Подключаем Kafka Streams: RangeAssignor – то, что надо!

```
Kafka Streams uses the StreamsPartitionAssignor class and doesn’t let you change to a different assignor. If you try to use a different assignor, Kafka Streams ignores it.
```

Подключаем Kafka Streams: StreamsPartitionAssignor

```
            }
        }
        return assignment;
    }

    /*
     * This assigns tasks to consumer clients in the following steps.
     *
     * 0. decode the subscriptions to assemble the metadata for each client and check for version probing
     *
     * 1. check all repartition source topics and use internal topic manager to make sure
     *    they have been created with the right number of partitions. Also verify and/or create
     *    any changelog topics with the correct number of partitions.
     *
     * 2. use the partition grouper to generate tasks along with their assigned partitions, then use
     *    the configured TaskAssignor to construct the mapping of tasks to clients.
     *
     * 3. construct the global mapping of host to partitions to enable query routing.
     *
     * 4. within each client, assign tasks to consumer clients.
     */
    @Override
    public GroupAssignment assign(final Cluster metadata, final GroupSubscription groupSubscription) {
        final Map<String, Subscription> subscriptions = groupSubscription.groupSubscription();

        // ---------------- Step Zero ---------------- //

        // construct the client metadata from the decoded subscription info

        final Map<UUID, CustomStreamPartitionAssignor.ClientMetadata> clientMetadataMap = new HashMap<>();
        final Set<TopicPartition> allOwnedPartitions = new HashSet<>();
        final Map<UUID, Map<String, Optional<String>>> racksForProcessConsumer = new HashMap<>();
```

Подключаем Kafka Streams: StreamsPartitionAssignor

```
            }
        }
        return assignment;
    }
```

0. decode the subscriptions to assemble the metadata for each client and check for version probing

1. check all repartition source topics and use internal topic manager to make sure
   they have been created with the right number of partitions. Also verify and/or create
   any changelog topics with the correct number of partitions.

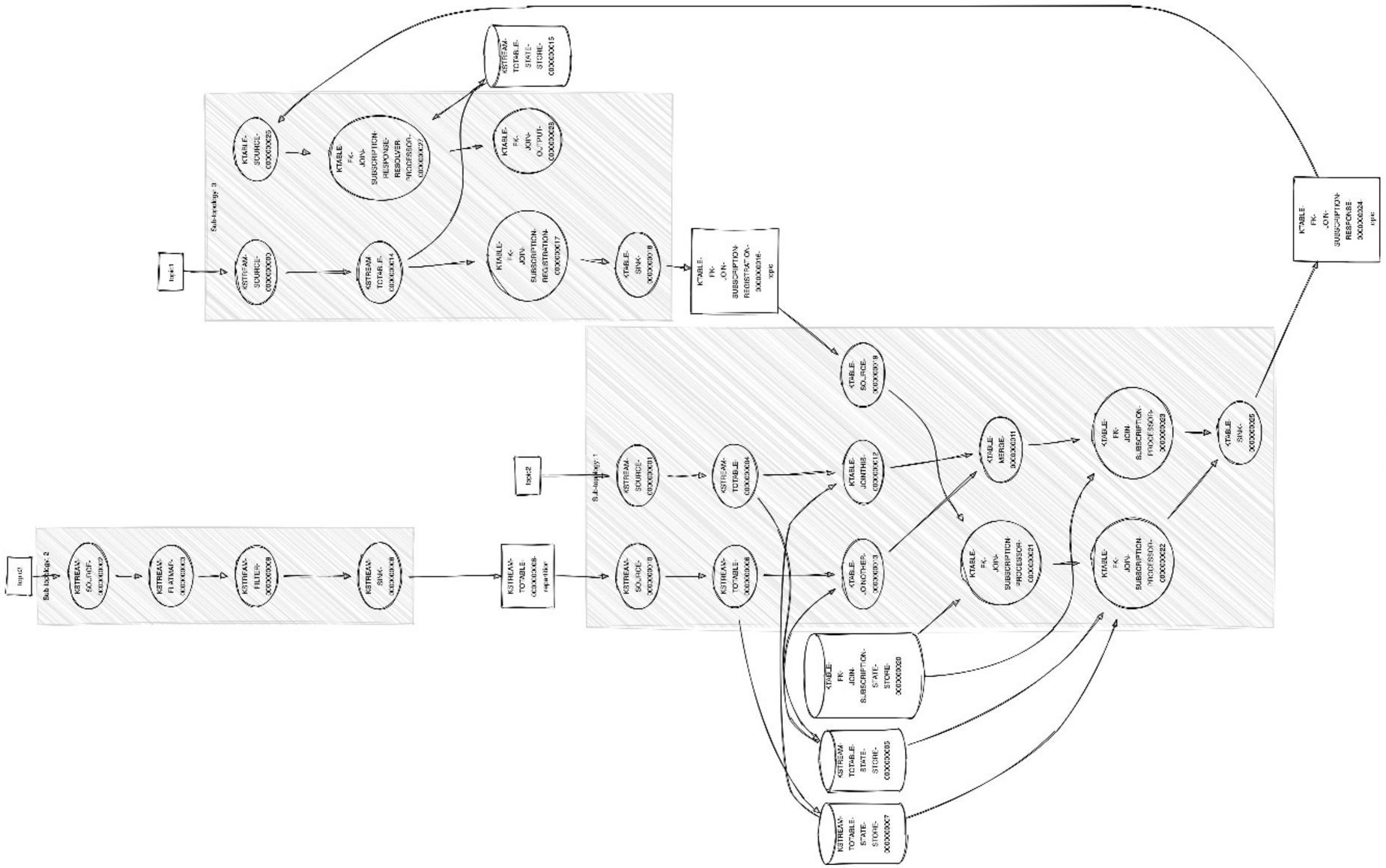
2. use the partition grouper to generate tasks along with their assigned partitions, then use
   the configured TaskAssignor to construct the mapping of tasks to clients.

3. construct the global mapping of host to partitions to enable query routing.

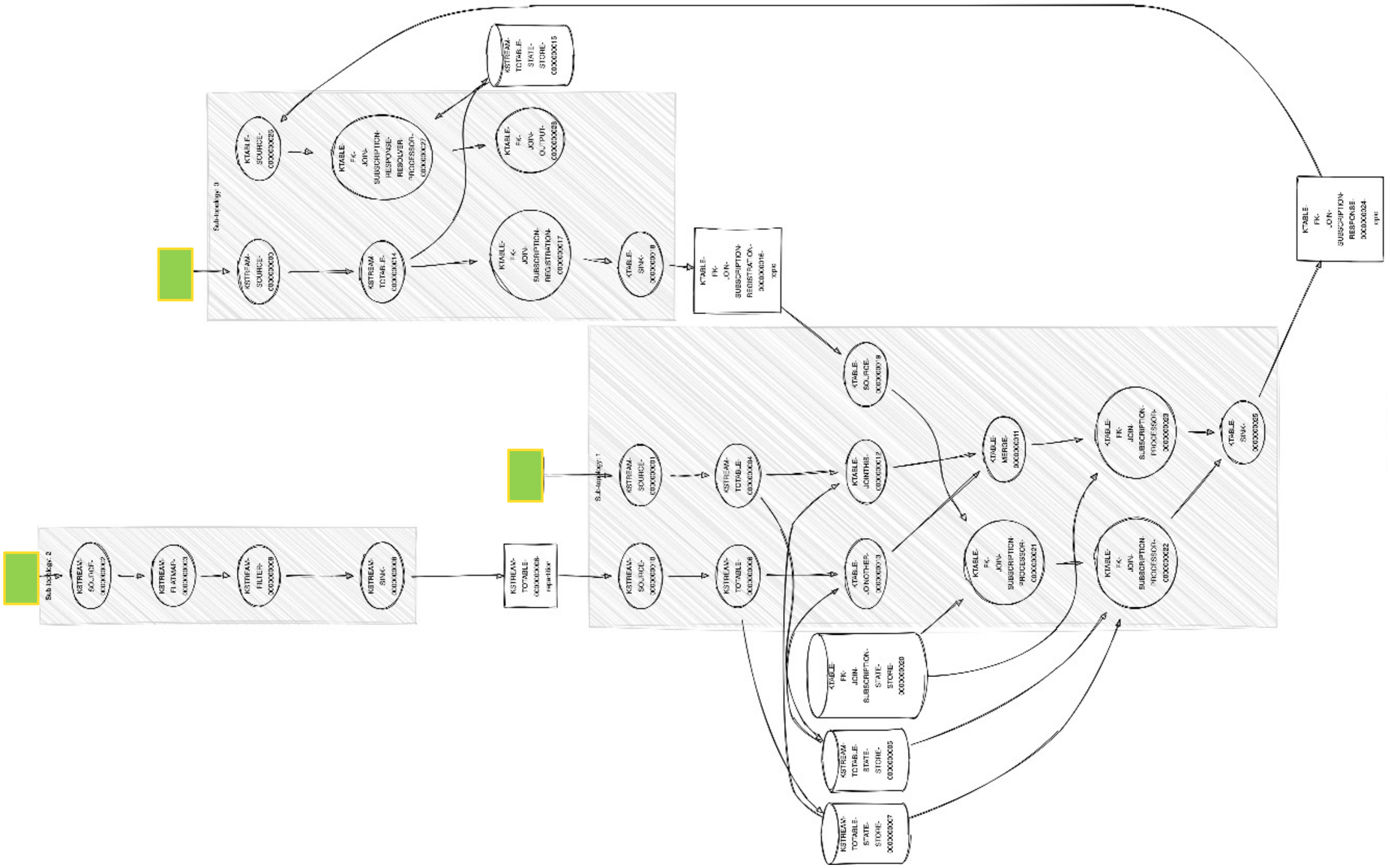
4. within each client, assign tasks to consumer clients.

```
        // construct the client metadata from the decoded subscription info

        final Map<UUID, CustomStreamPartitionAssignor.ClientMetadata> clientMetadataMap = new HashMap<>();
        final Set<TopicPartition> allOwnedPartitions = new HashSet<>();
        final Map<UUID, Map<String, Optional<String>>> racksForProcessConsumer = new HashMap<>();
```

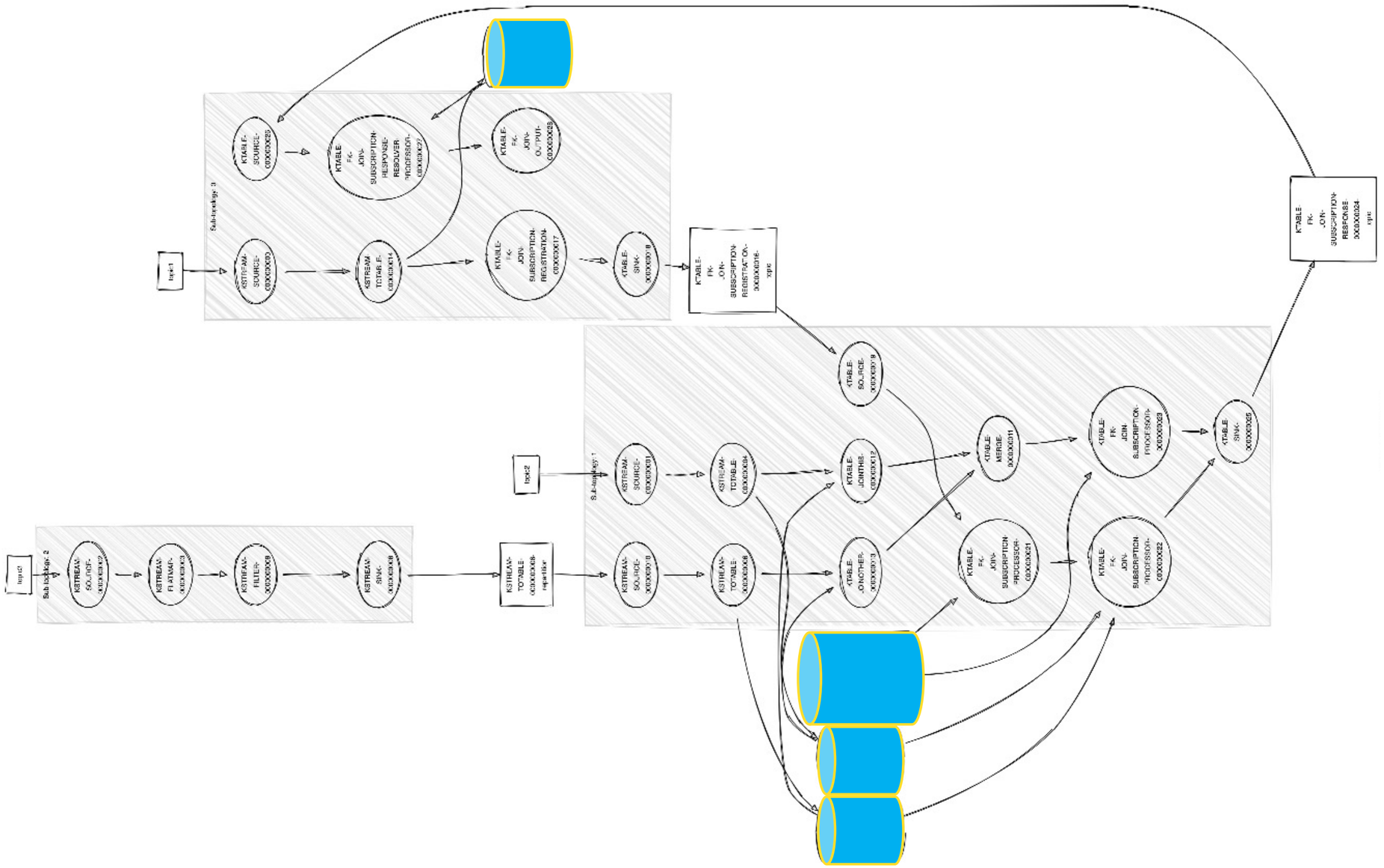
Подключаем Kafka Streams: «явный» джоин



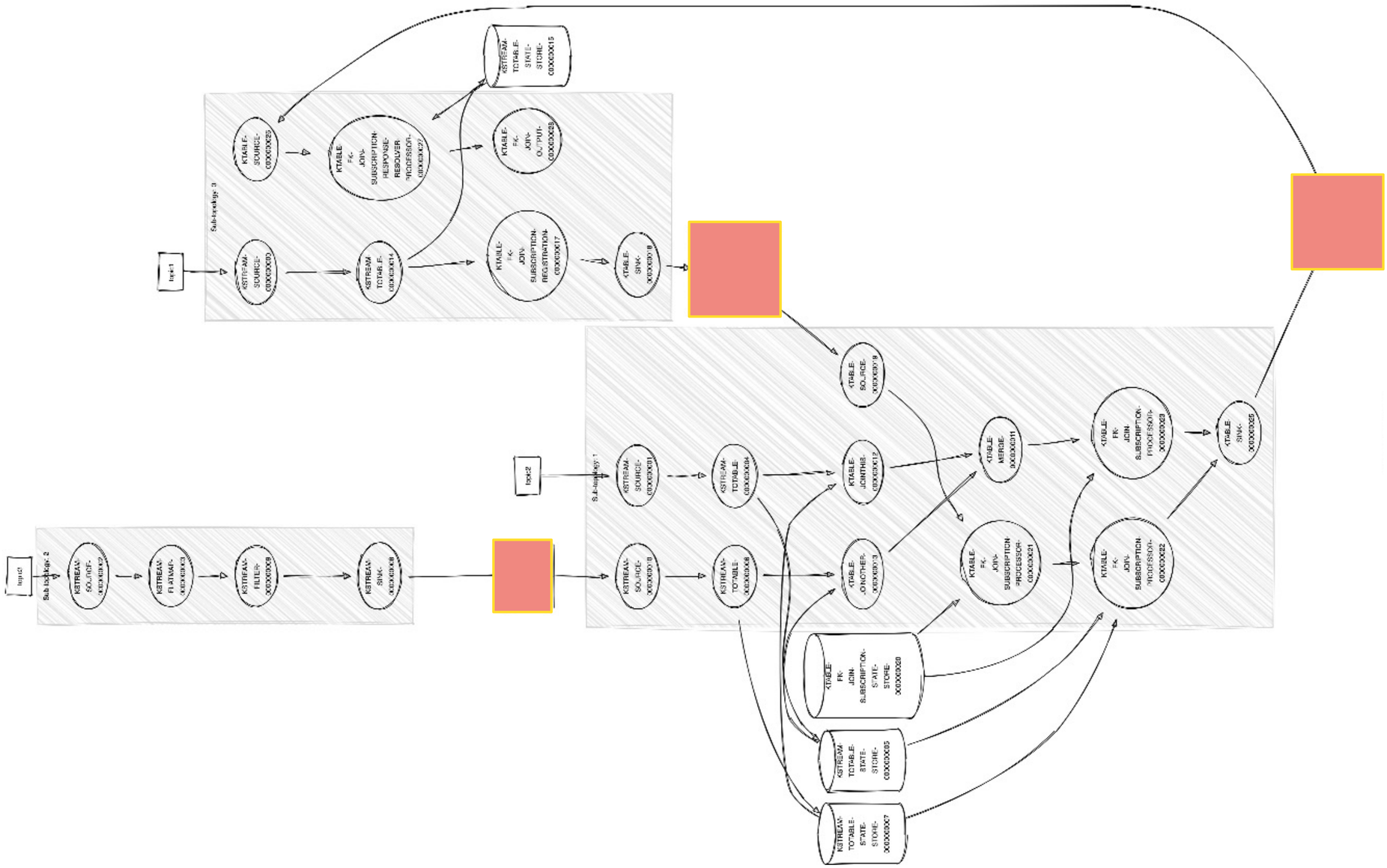
Подключаем Kafka Streams: «явный» джоин



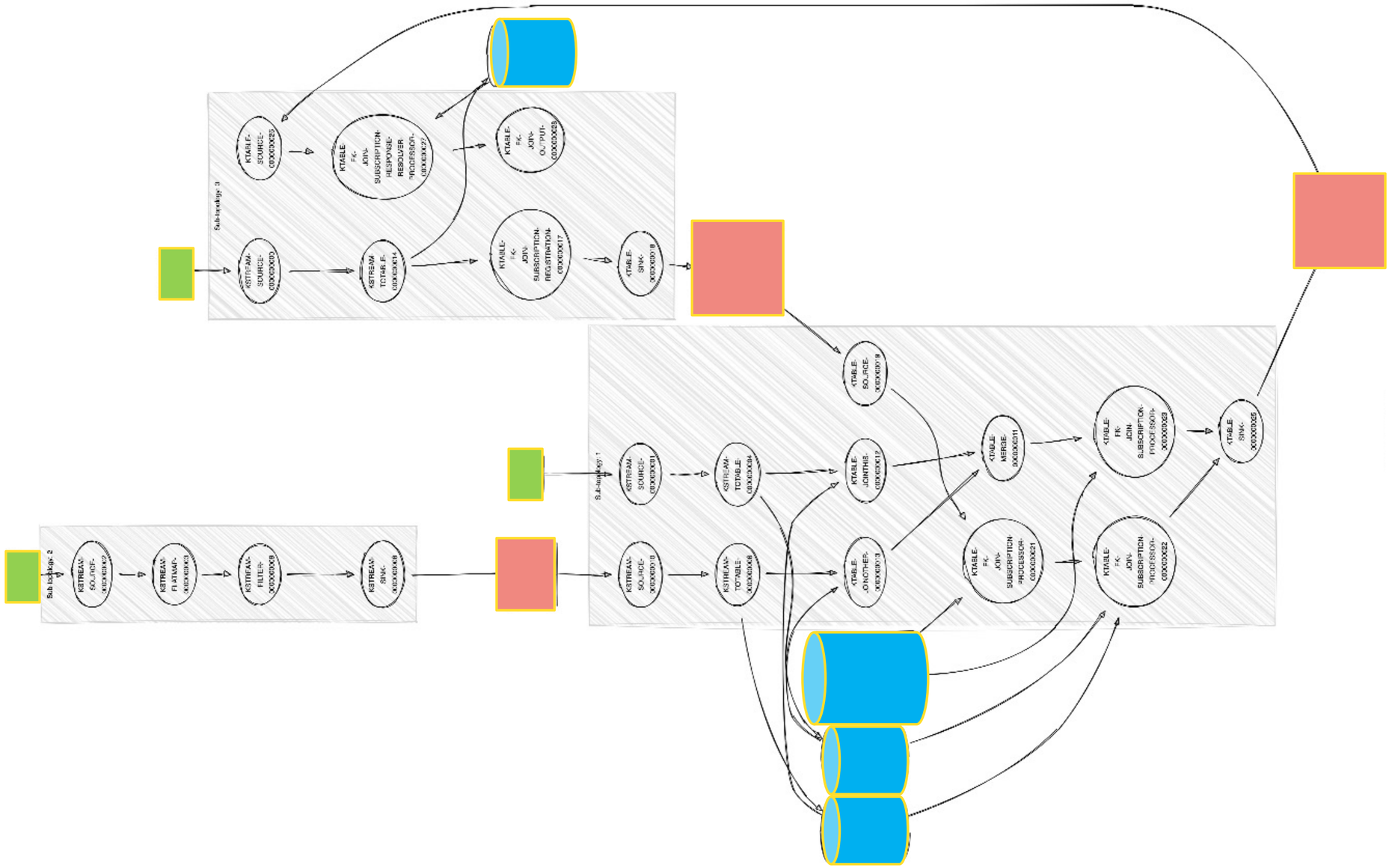
Подключаем Kafka Streams: «явный» джоин



Подключаем Kafka Streams: «явный» джоин



Подключаем Kafka Streams: «явный» джоин



Оптимизируем join: что мы знаем о наших топиках?

Число партиций одинаковое

Заранее контролируем число партиций, чтобы было равномерное распределение по консюмерам

Разные ключи сообщений

Ключи сообщений имеют разную авро-схему

Одинаковый Partitioner

Все сообщения партиционированны по имени топика

Оптимизируем join: что мы знаем о наших топиках?

Число партиций одинаковое

Заранее контролируем число партиций, чтобы было равномерное распределение по консюмерам

Разные ключи сообщений

Ключи сообщений имеют разную авро-схему

Одинаковый Partitioner

Все сообщения партиционированны по имени топика

Топики ко-партиционированы!

Оптимизируем join: что мы знаем о наших топиках?

Число партиций одинаковое

Заранее контролируем число партиций, чтобы было равномерное распределение по консюмерам

Разные ключи сообщений

Ключи сообщений имеют разную авро-схему

Одинаковый Partitioner

Все сообщения партиционированны по имени топика

Как донести эту информацию до Kafka Streams?

Оптимизируем join: TaskAssignor

```
            }
        }
        return assignment;
    }
```

0. decode the subscriptions to assemble the metadata for each client and check for version probing

1. check all repartition source topics and use internal topic manager to make sure
   they have been created with the right number of partitions. Also verify and/or create
   any changelog topics with the correct number of partitions.

2. use the partition grouper to generate tasks along with their assigned partitions, then use
   the configured TaskAssignor to construct the mapping of tasks to clients.

3. construct the global mapping of host to partitions to enable query routing.

4. within each client, assign tasks to consumer clients.

```
// construct the client metadata from the decoded subscription info

final Map<UUID, CustomStreamPartitionAssignor.ClientMetadata> clientMetadataMap = new HashMap<>();
final Set<TopicPartition> allOwnedPartitions = new HashSet<>();
final Map<UUID, Map<String, Optional<String>>> racksForProcessConsumer = new HashMap<>();
```

Оптимизируем join: напишем свой TaskAssignor

```
public class SpecialTaskAssignor implements TaskAssignor {
    @Override
    public boolean assign(Map<UUID, ClientState> clients, Set<TaskId> allTaskIds,
                          Set<TaskId> statefulTaskIds, RackAwareTaskAssignor rackAwareTaskAssignor,
AssignorConfiguration.AssignmentConfigs assignmentConfigs) {
        // Так как мы точно знаем, что у всех топиков одинаковое число партиций,
        // то мы можем сгруппировать таски по номерам партиций.
        // Каждый таск при этом -- это юнит чтения данных из конкретной партиции топика
        var tasksPerPartition = new HashMap<Integer, Set<TaskId>>();

        allTaskIds.forEach(task -> tasksPerPartition.compute(task.partition(), (ignored, set) -> {
                if (set == null) {
                    set = new HashSet<>();
                }

                set.add(task);
                return set;
            }));

        var clientsSize = clients.size();

        // Собираем список клиентов. Клиент != consumer
        var clientList = new ArrayList<UUID>();
        clients.forEach((uuid, state) -> clientList.add(uuid));

        // Каждому клиенту по принципу RoundRobin назначаем таски, привязанные к конкретной партиции
        var currentClient = 0;
        for (var tasks : tasksPerPartition.entrySet()) {
            var client = clients.get(clientList.get(currentClient % clientsSize));

            tasks.getValue().forEach(client::assignActive);
            currentClient++;
        }

        // Дополнительная ребалансировка не нужна, поэтому всегда возвращаем false
        return false;
    }
}
```

Оптимизируем join:

```java
var tasksPerPartition = new HashMap<Integer, Set<TaskId>>();

allTaskIds.forEach(task -> tasksPerPartition.compute(task.partition(), (ignored, set) -> {
        if (set == null) {
            set = new HashSet<>();
        }

        set.add(task);
        return set;
    }));
```

Группируем партиции по их номерам

Оптимизируем join:

Собираем список клиентов

```
var clientList = new ArrayList<UUID>();
clients.forEach((uuid, state) -> clientList.add(uuid));
```

```
public class SpecialTaskAssignor implements TaskAssignor {
    @Override
    public boolean assign(Map<UUID, ClientState> clients, Set<TaskId> allTaskIds,
                          Set<TaskId> statefulTaskIds, RackAwareTaskAssignor rackAwareTaskAssignor,
AssignorConfiguration.AssignmentConfigs assignmentConfigs) {
        // Так как мы точно знаем, что у всех топиков одинаковое число партиций,
        // то мы можем сгруппировать таски по номерам партиций.
        // Каждый таск при этом -- это юнит чтения данных из конкретной партиции топика
        var tasksPerPartition = new HashMap<Integer, Set<TaskId>>();

        allTaskIds.forEach(task -> tasksPerPartition.compute(task.partition(), (ignored, set) -> {
                if (set == null) {
                    set = new HashSet<>();
                }

                set.add(task);
                return set;

        // Каждому клиенту по принципу RoundRobin назначаем таски, привязанные к конкретной партиции
        var currentClient = 0;
        for (var tasks : tasksPerPartition.entrySet()) {
            var client = clients.get(clientList.get(currentClient % clientsSize));

            tasks.getValue().forEach(client::assignActive);
            currentClient++;
        }

        // Дополнительная ребалансировка не нужна, поэтому всегда возвращаем false
        return false;
    }
}
```

Оптимизируем join:

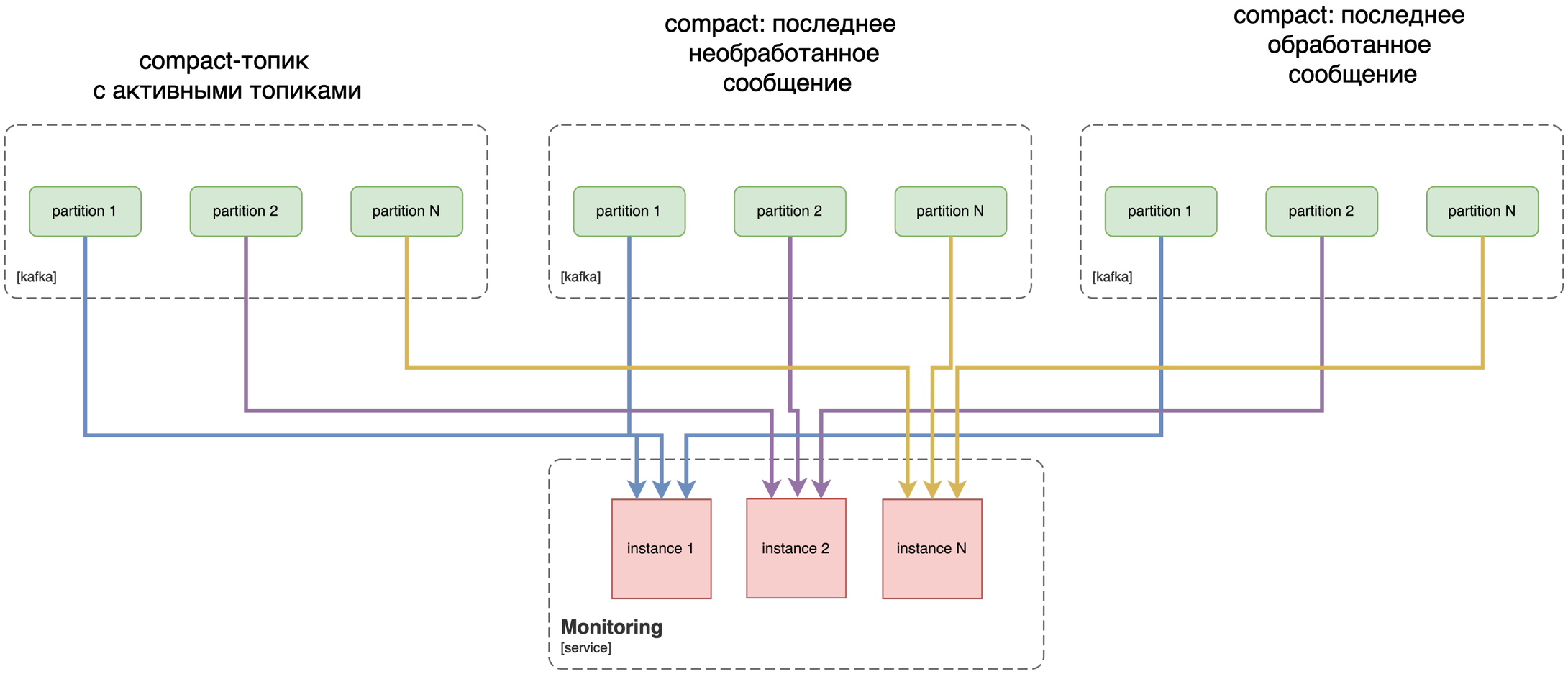
Назначаем партиции клиентам

```
public class SpecialTaskAssignor implements TaskAssignor {
    @Override
    public boolean assign(Map<UUID, ClientState> clients, Set<TaskId> allTaskIds,
                          Set<TaskId> statefulTaskIds, RackAwareTaskAssignor rackAwareTaskAssignor,
AssignorConfiguration.AssignmentConfigs assignmentConfigs) {
        // Так как мы точно знаем, что у всех топиков одинаковое число партиций,
        // то мы можем сгруппировать таски по номерам партиций.
        // Каждый таск при этом -- это юнит чтения данных из конкретной партиции топика
        var tasksPerPartition = new HashMap<Integer, Set<TaskId>>();

        allTaskIds.forEach(task -> tasksPerPartition.compute(task.partition(), (ignored, set) -> {
                if (set == null) {
                    set = new HashSet<>();
                }
```

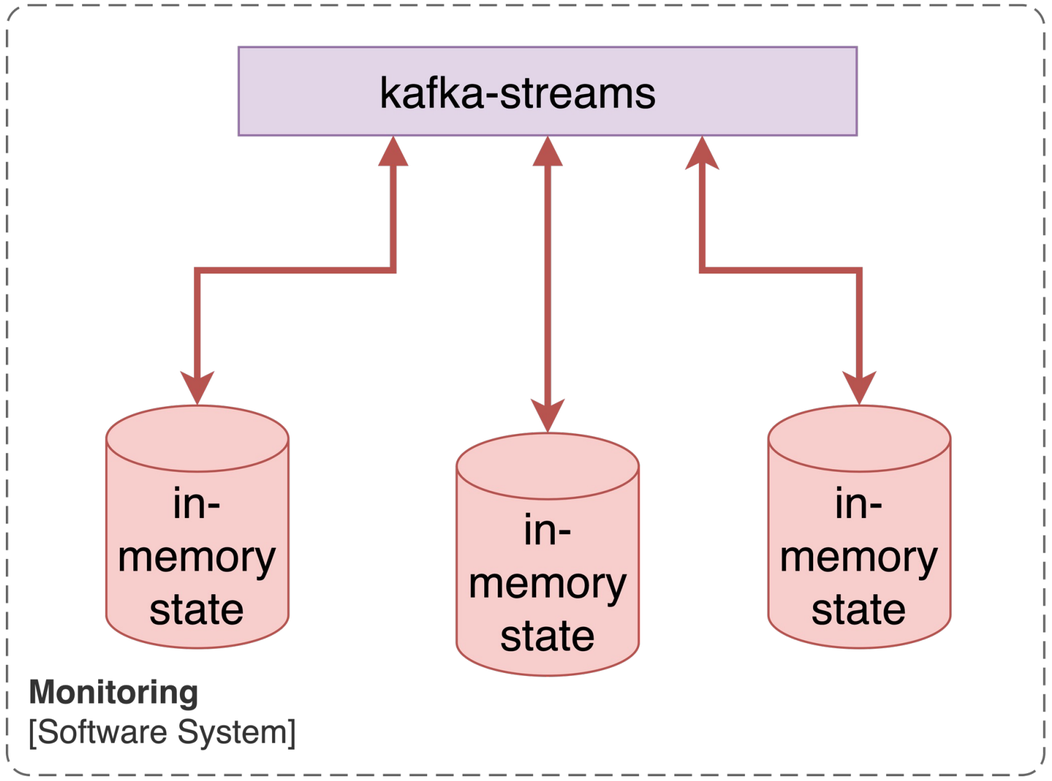
```
var currentClient = 0;
for (var tasks : tasksPerPartition.entrySet()) {
    var client = clients.get(clientList.get(currentClient % clientsSize));

    tasks.getValue().forEach(client::assignActive);
    currentClient++;
}
```

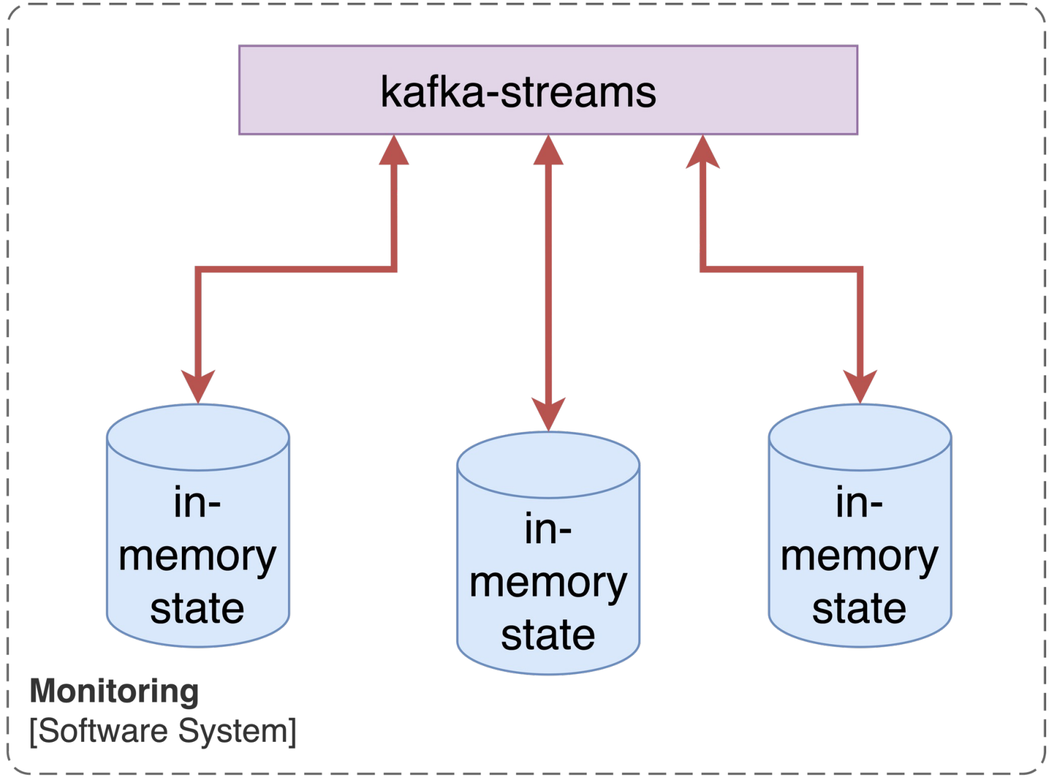
```
        }

        // Дополнительная ребалансировка не нужна, поэтому всегда возвращаем false
        return false;
    }
}
```

Оптимизируем join: финальный результат

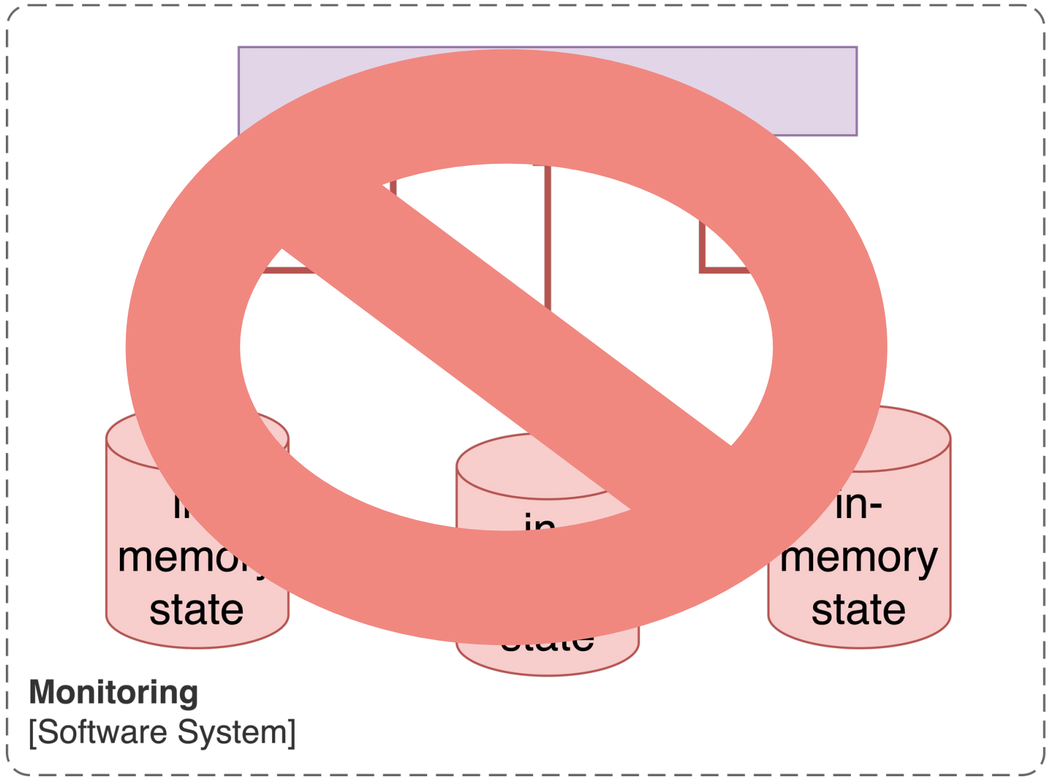


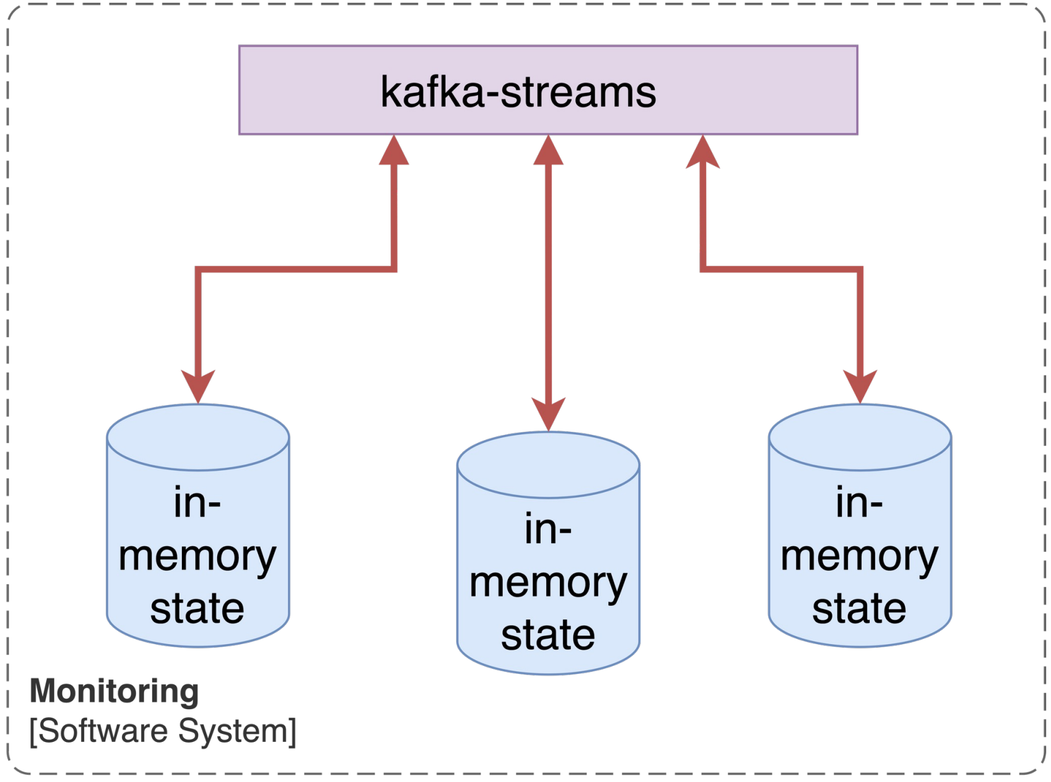
Оптимизируем join: fault tolerance



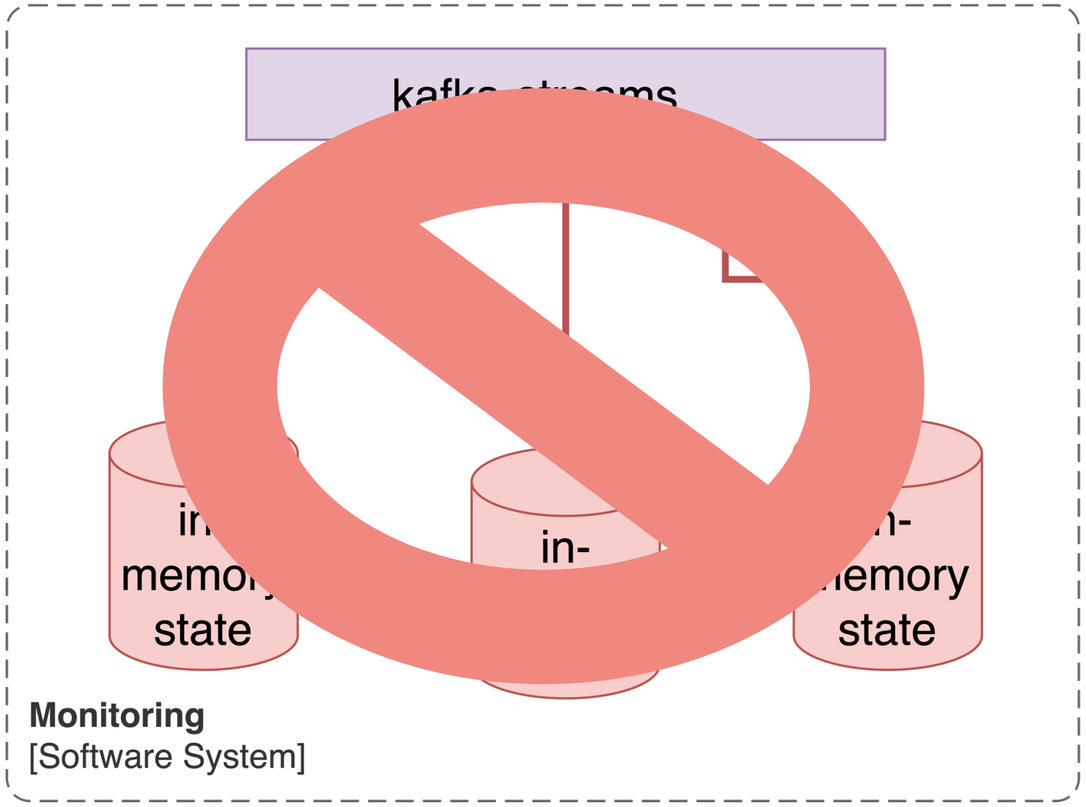


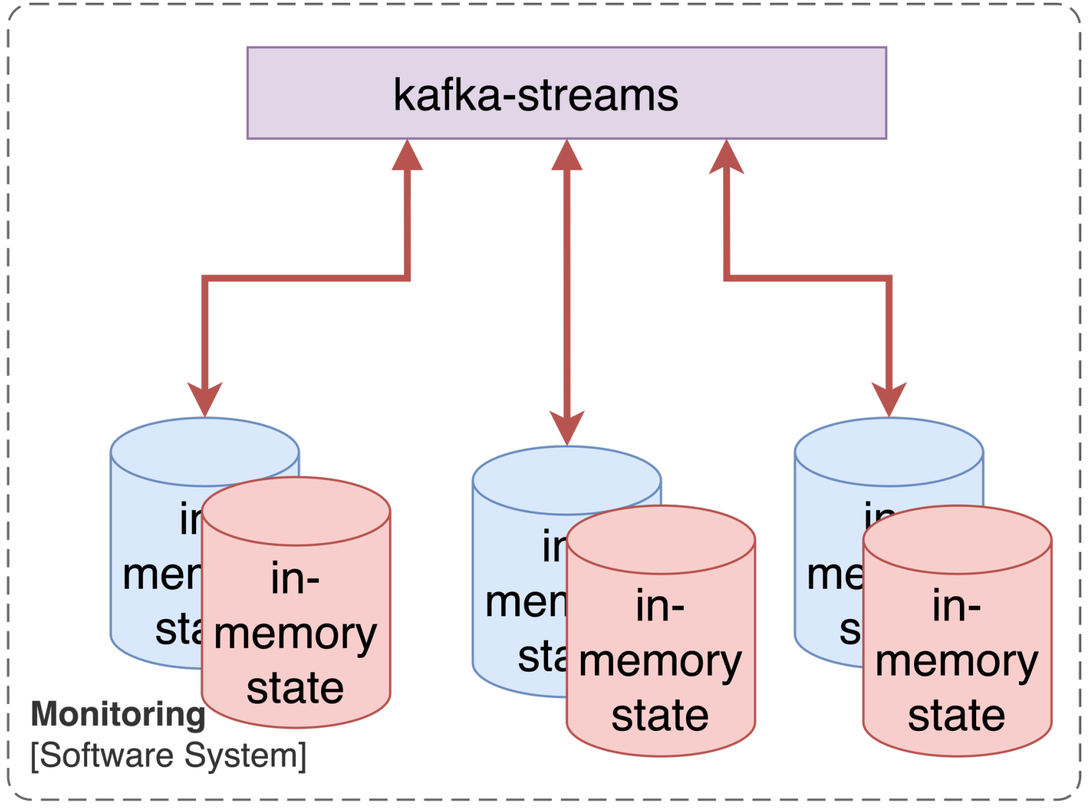
Оптимизируем join: fault tolerance



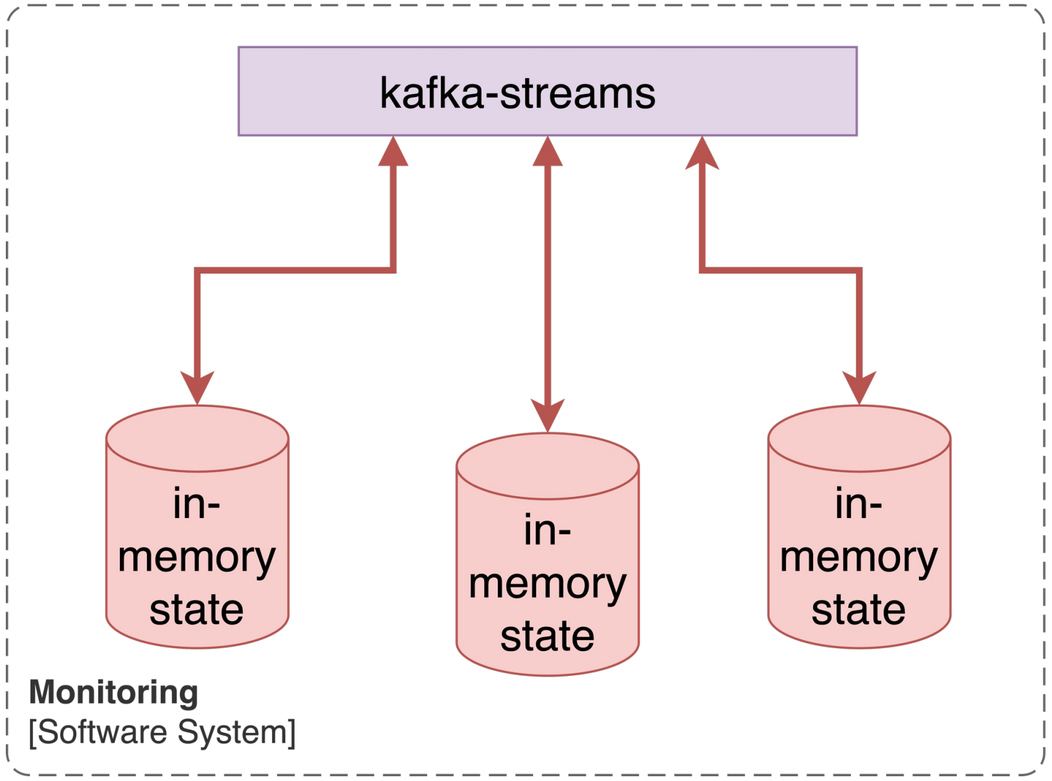


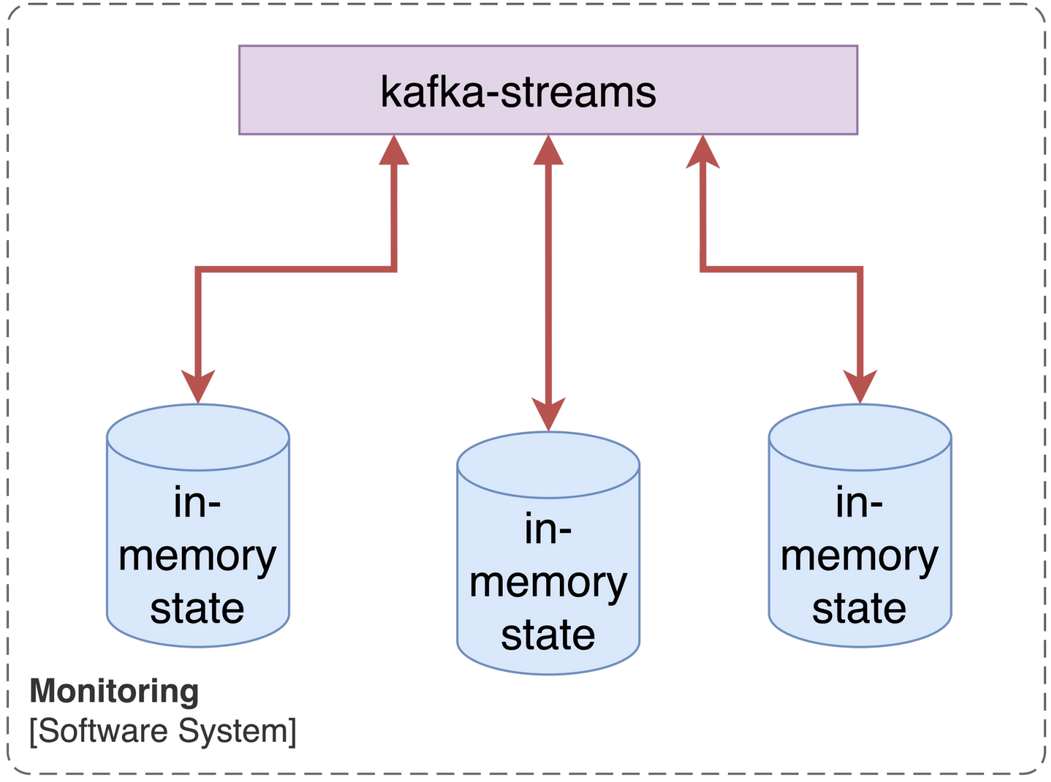
Оптимизируем join: fault tolerance





Оптимизируем join: fault tolerance





Оптимизируем join: Scalability

Как масштабироваться дальше?

Правила и нюансы увеличения параллелизма все те же, что и для обычных консюмеров

Распределение ключей изменится, поэтому есть несколько вариантов workaround’а:

- Остановить временно поставку
- Нужно дочитать все, до увеличения партиций

Оптимизируем join: Scalability

Как масштабироваться дальше?

Правила и нюансы увеличения параллелизма все те же, что и для обычных консюмеров

Распределение ключей изменится, поэтому есть несколько вариантов workaround’а:

- Пересоздать все с 0

Выводы: Monitoring

Бизнесу непонятны оффсеты

На оффсеты сложно повесить SLA

Проще управлять метрикой, основанной на времени

Это понятнее и бизнесу, и разработчикам

Выводы: Kafka Streams

Мониторинг – частный пример задачи обогащения данных

KafkaStreams – библиотека, которую можно встроить практически в любой микросервис

При небольших усилиях можно существенно **упростить топологию**

ТИНЬКОФФ

Спасибо!

ТИНЬКОФФ

QA