Akka Streams в реальных задачах

Евгений Ненахов

MTC Digital

@neltari

newnew94@mail.ru

JPoint

MTC DIGITAL

О чём пойдёт речь…

- Постановка задачи

О чём пойдёт речь…

- Постановка задачи
- Akka – в качестве инструмента

О чём пойдёт речь…

- Постановка задачи
- Akka – в качестве инструмента
- Как решали задачу с помощью Akka Stream

О чём пойдёт речь…

- Постановка задачи
- Akka – в качестве инструмента
- Как решали задачу с помощью Akka Stream
- Проблемы, неприятности и их решения

О чём пойдёт речь…

- Постановка задачи
- Akka – в качестве инструмента
- Как решали задачу с помощью Akka Stream
- Проблемы, неприятности и их решения
- Инфраструктура

О чём пойдёт речь…

- Постановка задачи
- Akka – в качестве инструмента
- Как решали задачу с помощью Akka Stream
- Проблемы, неприятности и их решения
- Инфраструктура
- Подведение итогов

Постановка задачи – что имеем?

Данные от абонентов и коммутационных устройств

Постановка задачи – что имеем?

Данные от абонентов и коммутационных устройств

- Геолокация

Постановка задачи – что имеем?

Данные от абонентов и коммутационных устройств

- Геолокация
- Звонки

Постановка задачи – что имеем?

Данные от абонентов и коммутационных устройств

- Геолокация
- Звонки
- Роуминг

Постановка задачи – что имеем?

Данные от абонентов и коммутационных устройств

- Геолокация
- Звонки
- Роуминг
- Кликстрим

Постановка задачи – что имеем?

Данные от абонентов и коммутационных устройств

- Геолокация
- Звонки
- Роуминг
- Кликстрим
- Прочее

Постановка задачи – что хотим?

Постановка задачи – что хотим?

- Точечная реклама

Постановка задачи – что хотим?

- Точечная реклама
- Информация о скидках в retail рядом

Постановка задачи – что хотим?

- Точечная реклама
- Информация о скидках в retail рядом
- Предложение по кредиту

Постановка задачи – что хотим?

- Точечная реклама
- Информация о скидках в retail рядом
- Предложение по кредиту
- Тарифный план в роуминге

Постановка задачи – что хотим?

- Точечная реклама
- Информация о скидках в retail рядом
- Предложение по кредиту
- Тарифный план в роуминге
- Что-то другое не менее прекрасное…

Постановка задачи – детали

Постановка задачи – детали

- ~ 5 – 7 млн. событий в секунду

Постановка задачи – детали

- ~ 5 – 7 млн. событий в секунду
- Отказоустойчивость

Постановка задачи – детали

- ~ 5 – 7 млн. событий в секунду
- Отказоустойчивость
- Масштабируемость

Постановка задачи – детали

- ~ 5 – 7 млн. событий в секунду
- Отказоустойчивость
- Масштабируемость
- Фильтрация, трансформация, дедупликация, обогащение

Постановка задачи – детали

- ~ 5 – 7 млн. событий в секунду
- Отказоустойчивость
- Масштабируемость
- Фильтрация, трансформация, дедупликация, обогащение
- Минимальный порог входа для потребителей и… разработчиков

Постановка задачи – детали

- ~ 5 – 7 млн. событий в секунду
- Отказоустойчивость
- Масштабируемость
- Фильтрация, трансформация, дедупликация, обогащение
- Минимальный порог входа для потребителей и… разработчиков
- Создание новых потоков данных в режиме реального времени

Решение задачи - организация потоковой обработки данных

- Видео доклада:

- Статьи на Хабр:

Инструментарий

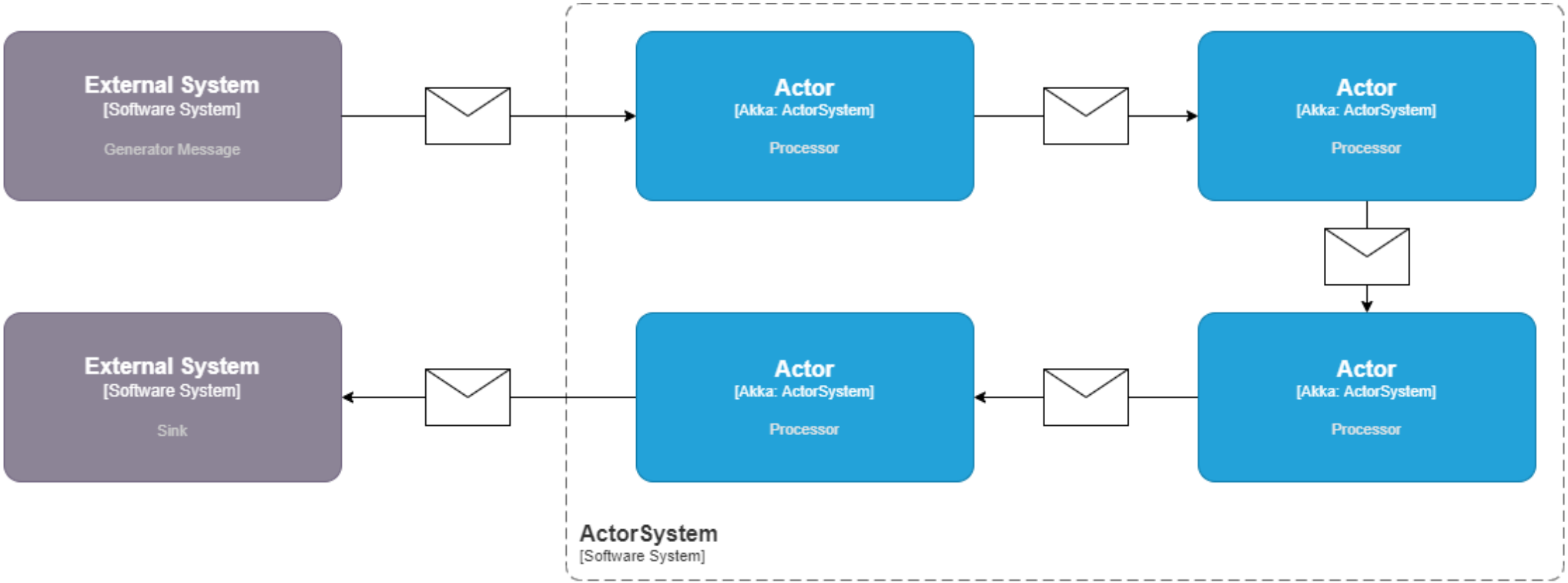
Инструментарий

- Готовые фреймворки:
  - Apache Spark Structured Streaming
  - Apache Flink
  - Apache NiFi
  - Apache Storm
  - KSQL и Kafka Streams
  - …

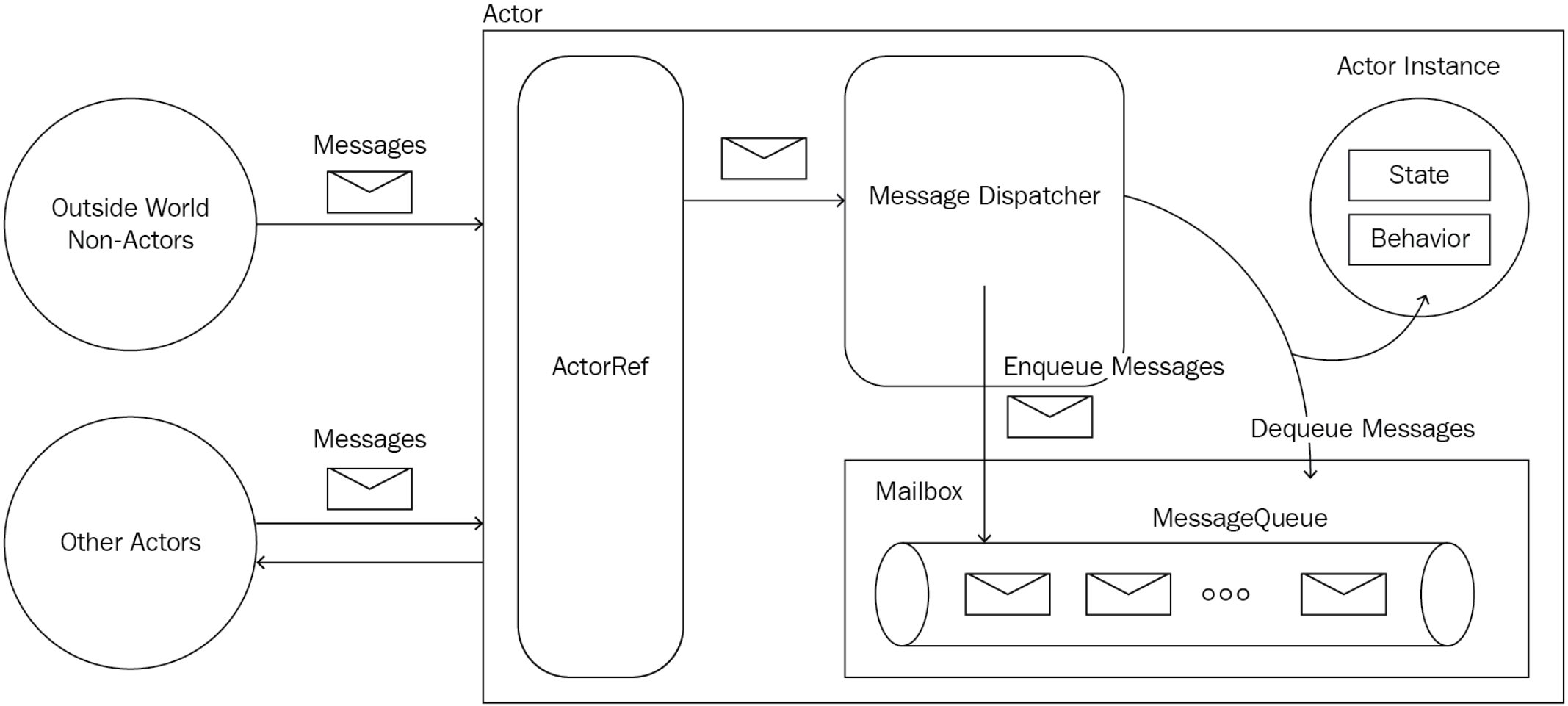
Инструментарий

- Готовые фреймворки:
  - Apache Spark Structured Streaming
  - Apache Flink
  - Apache NiFi
  - Apache Storm
  - KSQL и Kafka Streams
  - …
  - Akka и Akka Streams

Akka – акторная модель



Akka – структура Actor



Akka – ключевые преимущества

Akka – ключевые преимущества

- Динамическое создание акторов

Akka – ключевые преимущества

- Динамическое создание акторов
- Эффективная утилизация ресурсов

Akka – ключевые преимущества

- Динамическое создание акторов
- Эффективная утилизация ресурсов
- Легко масштабируется

Akka – ключевые преимущества

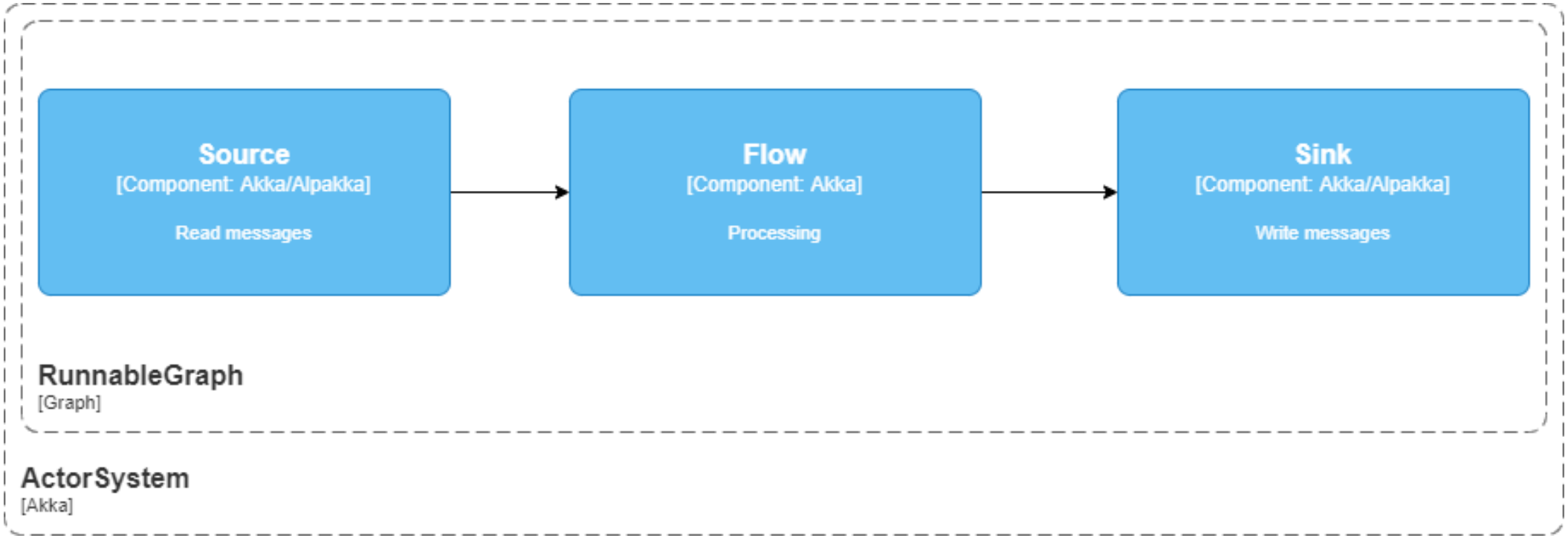
- Динамическое создание акторов
- Эффективная утилизация ресурсов
- Легко масштабируется
- Подходит для потоковой обработки данных

Akka Stream

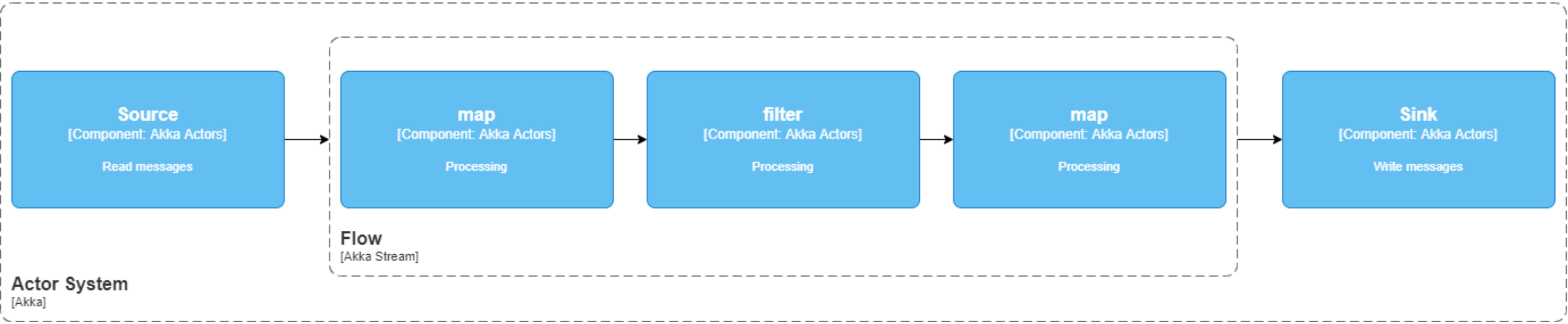
- Java Reactive Stream (JDK 9) + Akka Actors = Akka Stream

Akka Stream

- Java Reactive Stream (JDK 9) + Akka Actors = Akka Stream

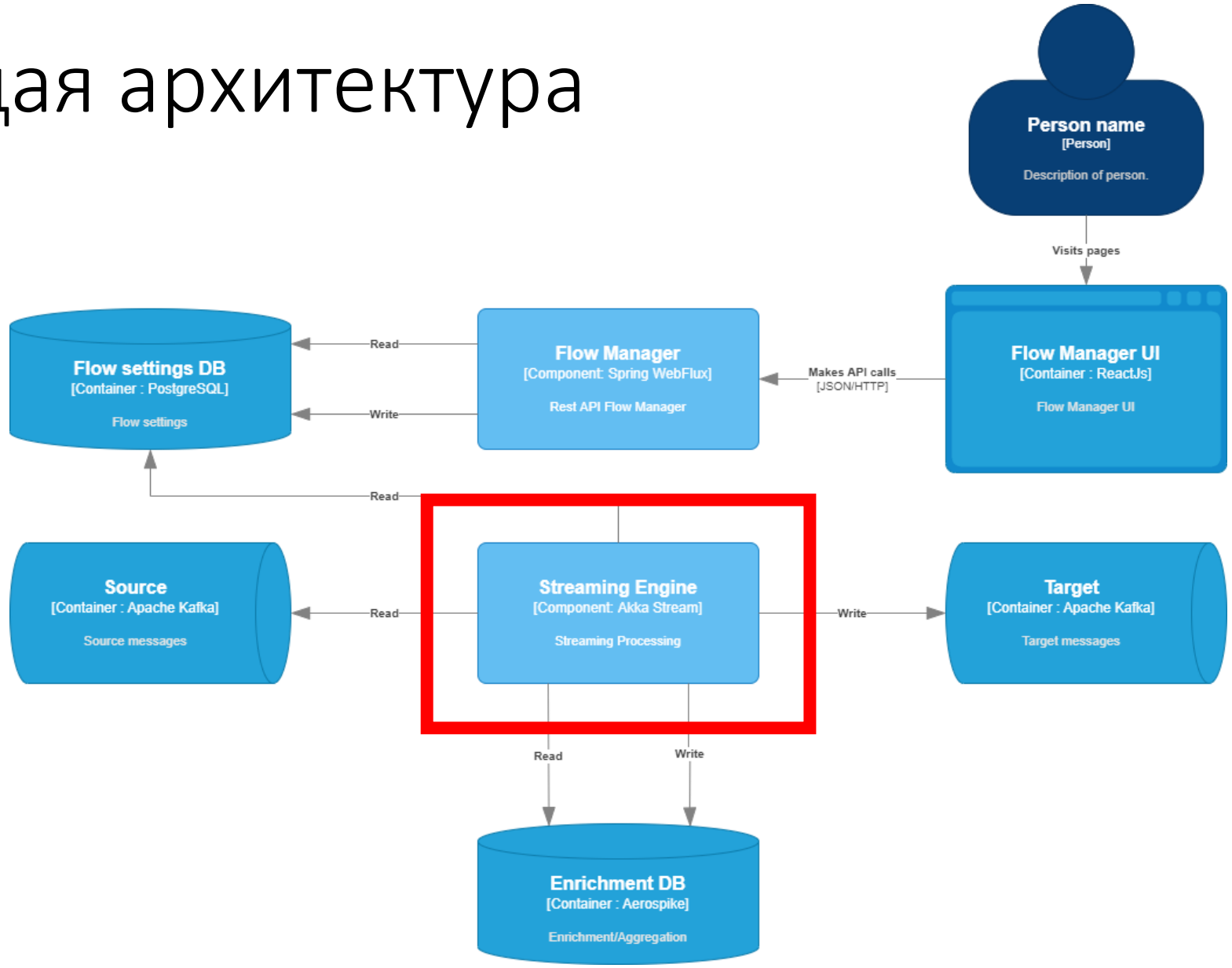


Akka Stream - Flow



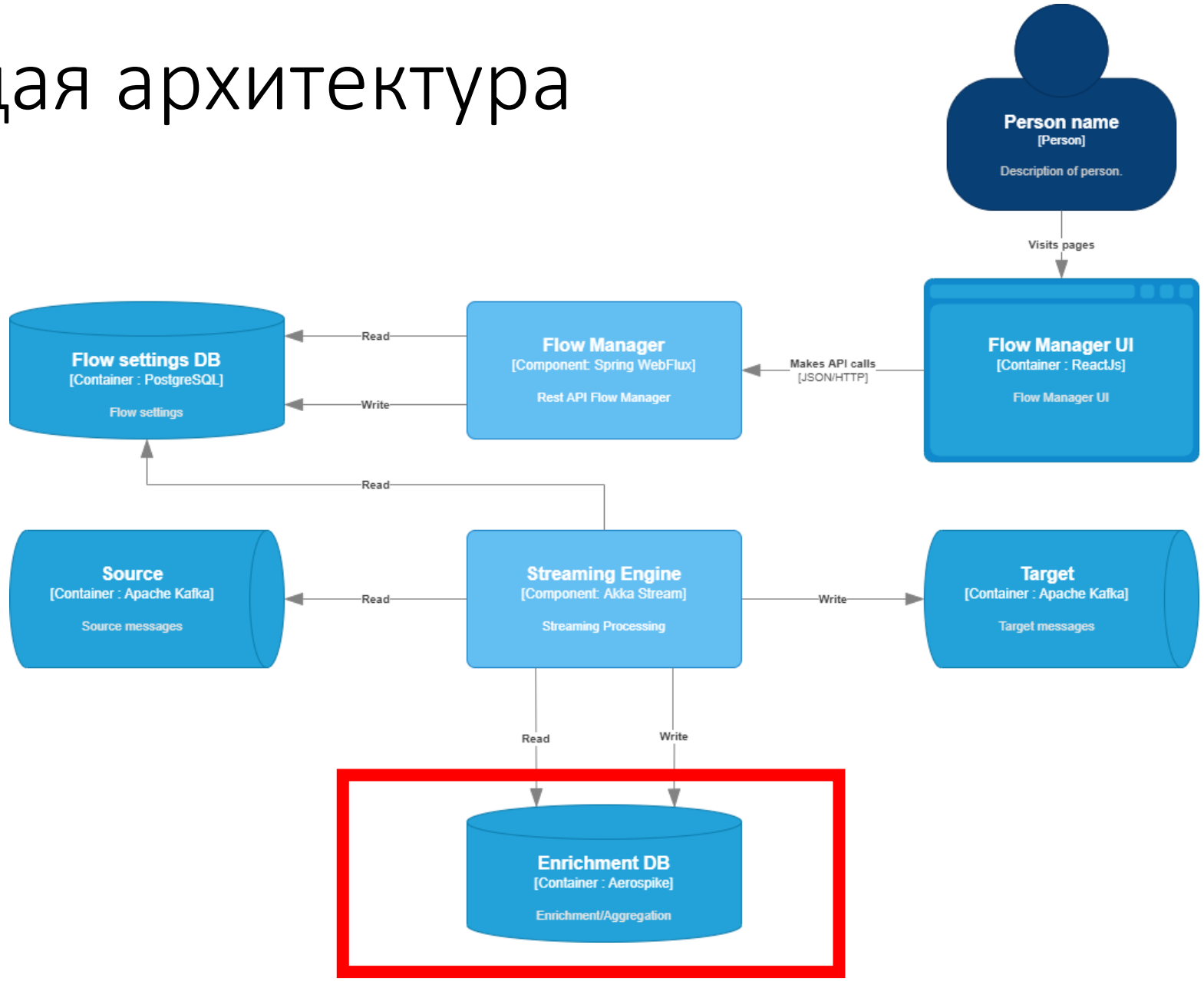
Общая архитектура

Общая архитектура



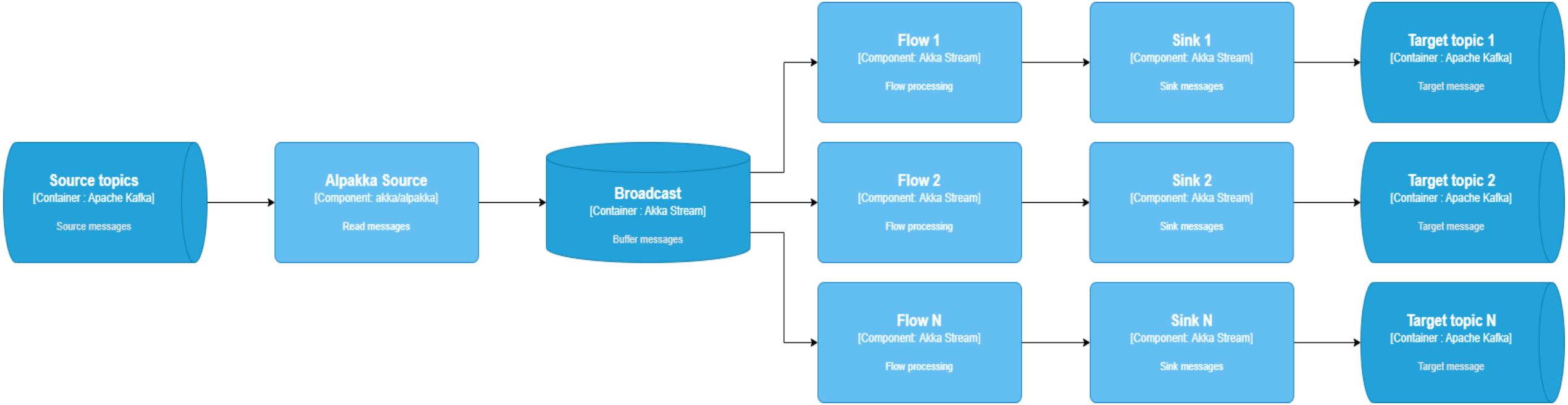
Общая архитектура

Общая архитектура



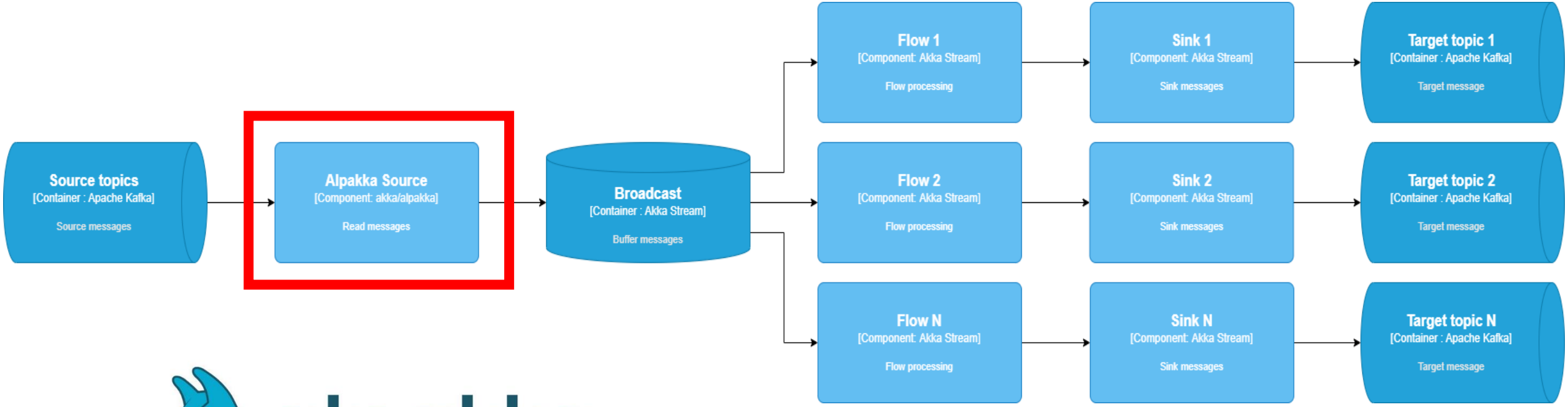
Общая архитектура

Решение 0 – первый концепт



akka streams

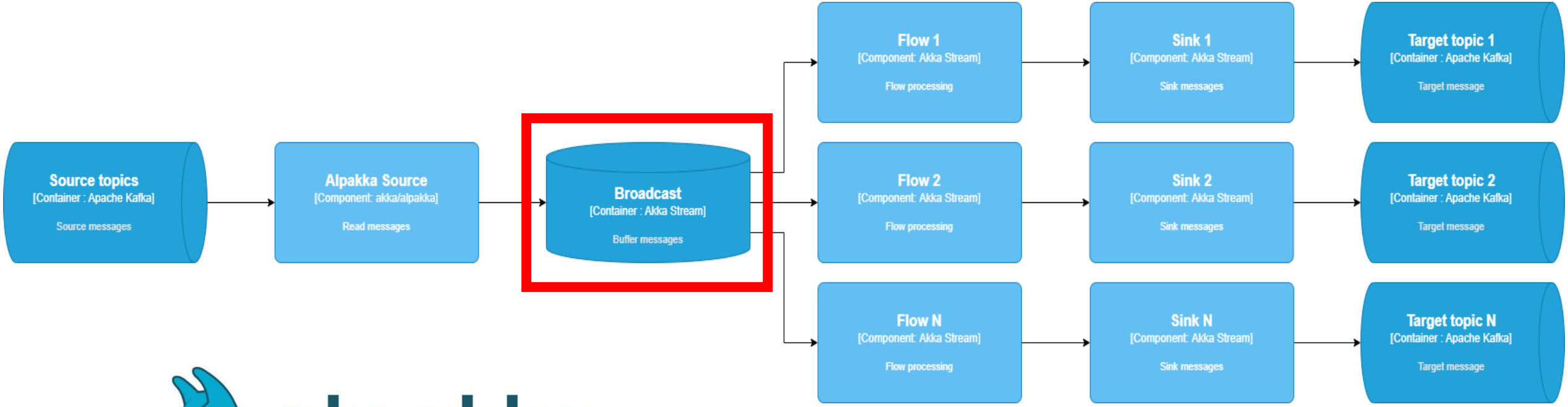
Решение 0 – первый концепт



alpakka

akka streams

Решение 0 – первый концепт

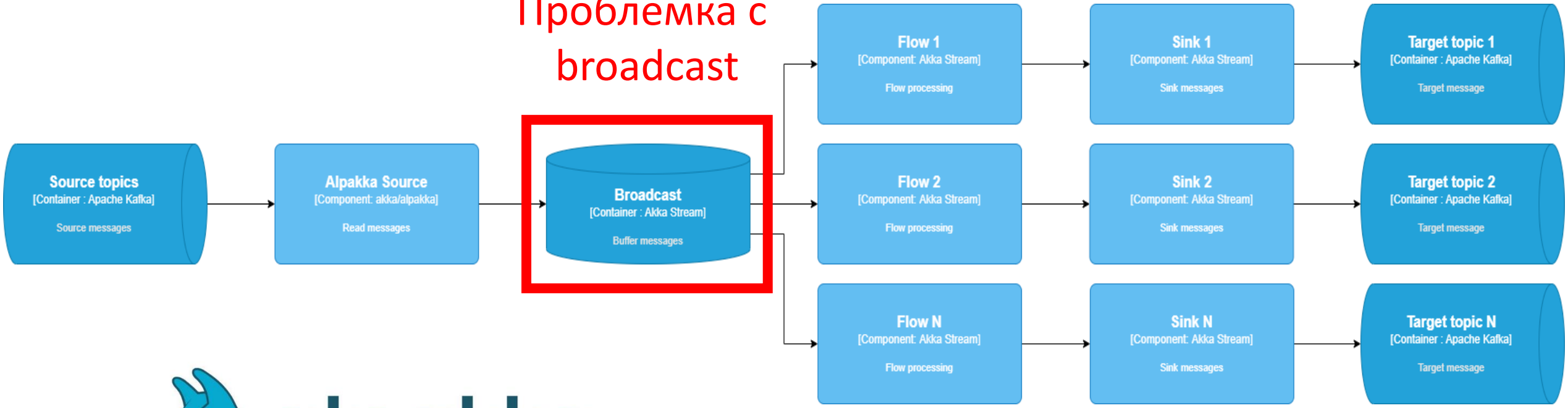


alpakka

akka streams

Решение 0 – первый концепт

Проблемка с broadcast



alpakka

akka streams

Решение 0 – проблемы

Решение 0 – проблемы

- Переполнение Broadcast
  - OverflowStrategy.backpressure()

Решение 0 – проблемы

- Переполнение Broadcast
  - OverflowStrategy.backpressure()
- Плохая утилизация RAM и CPU

Решение 0 – проблемы

- Переполнение Broadcast
  - OverflowStrategy.backpressure()
- Плохая утилизация RAM и CPU
- Производительность

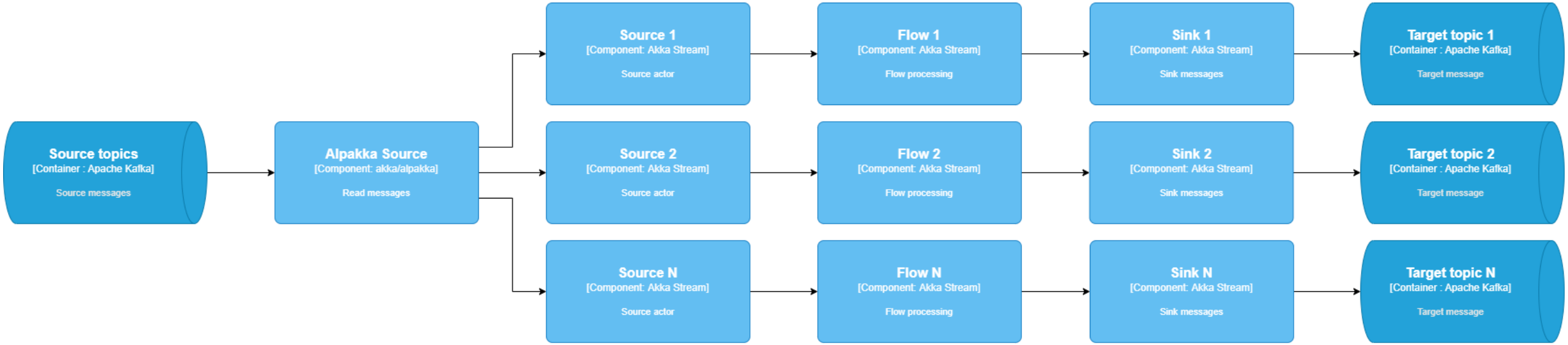
Решение 0 – проблемы

- Переполнение Broadcast
  - OverflowStrategy.backpressure()
- Плохая утилизация RAM и CPU
- Производительность
- Сложная конфигурация

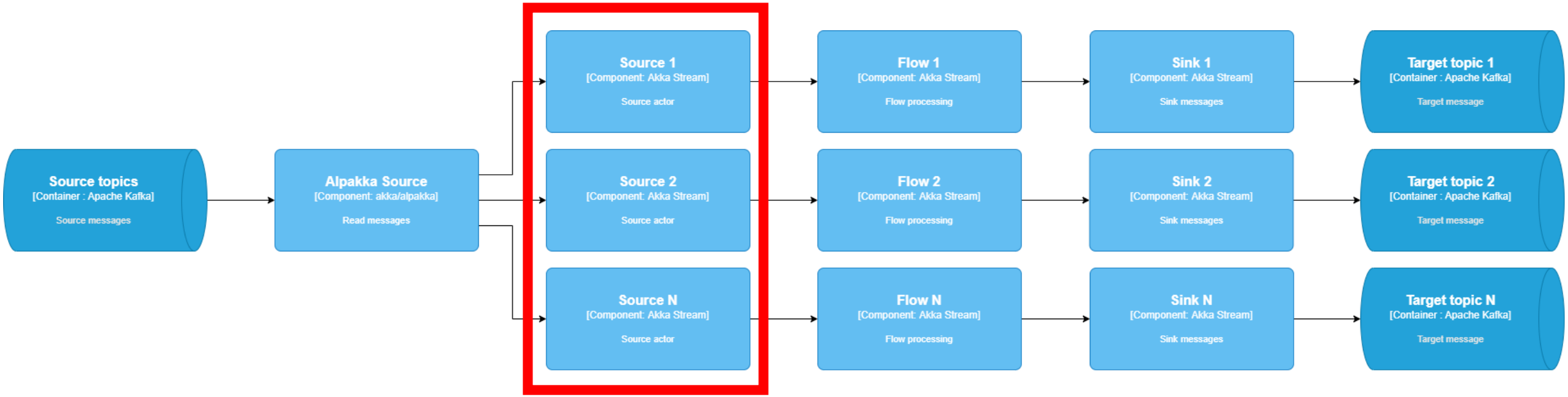
Решение 0 – проблемы

- Переполнение Broadcast
  - OverflowStrategy.backpressure()
- Плохая утилизация RAM и CPU
- Производительность
- Сложная конфигурация

Решение 1 – исправление broadcast

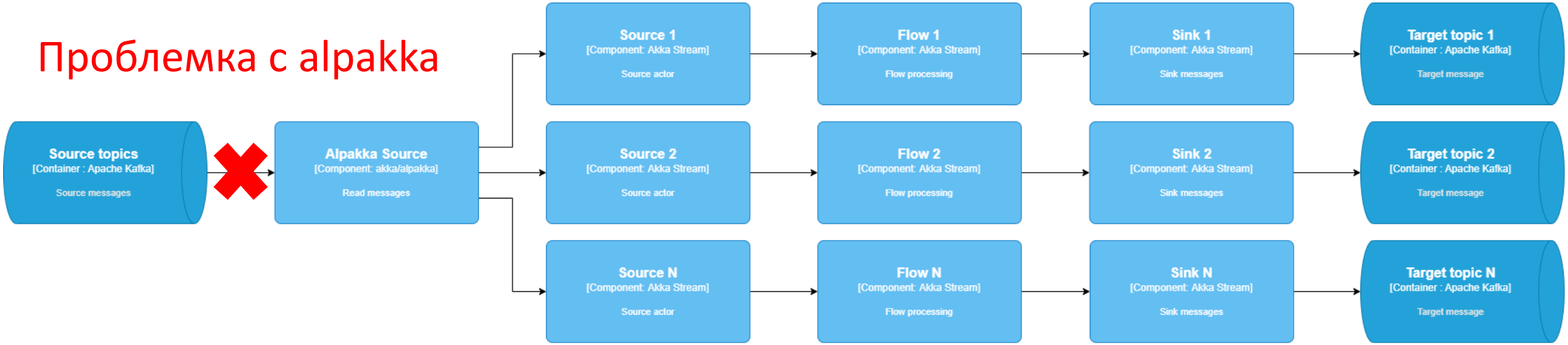


Решение 1 – исправление broadcast



Решение 1 – исправление broadcast

Проблемка с alpakka



Решение 1 – исправление broadcast

Решение 1 – исправление broadcast

- Ненадёжное чтение

Решение 1 – исправление broadcast

- Ненадёжное чтение
- Большое количество акторов

Решение 1 – исправление broadcast

- Ненадёжное чтение
- Большое количество акторов
- Всё ещё сложная конфигурация

Решение 1 – исправление broadcast

- Ненадёжное чтение
- Большое количество акторов
- Всё ещё сложная конфигурация
- Всё ещё производительность

Решение 1 – исправление broadcast

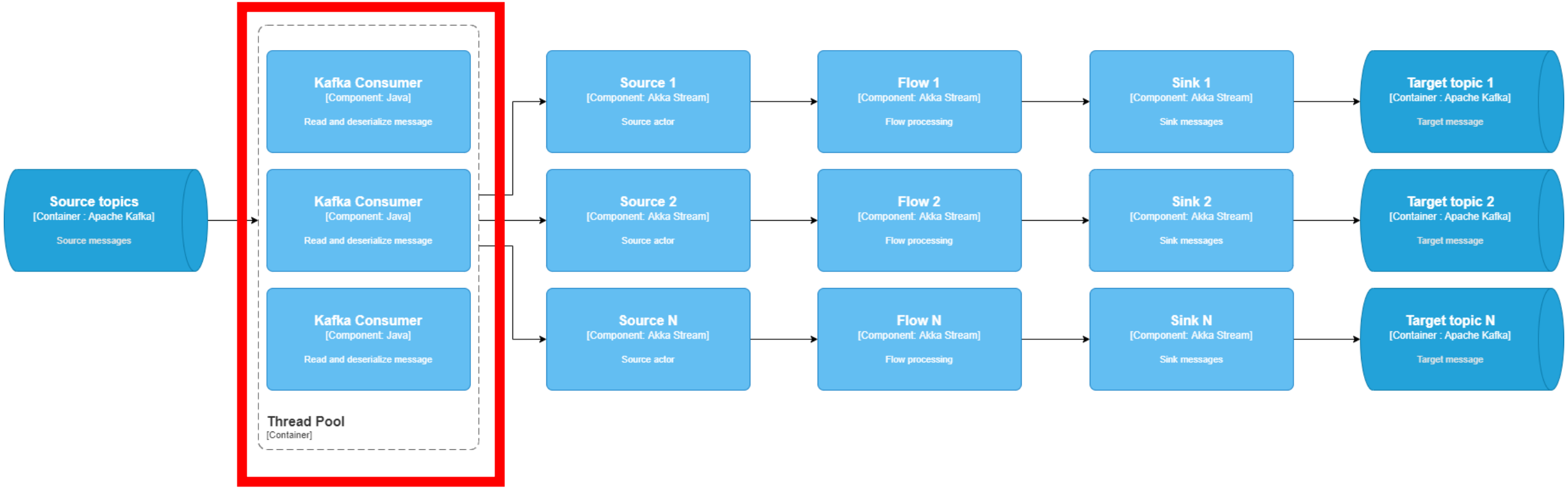
- Ненадёжное чтение
- Большое количество акторов
- Всё ещё сложная конфигурация
- Всё ещё производительность
- Всё ещё плохая утилизация ресурсов

Решение 1 – исправление broadcast

- Ненадёжное чтение
- Большое количество акторов
- Всё ещё сложная конфигурация
- Всё ещё производительность
- Всё ещё плохая утилизация ресурсов

Решение 2 – избавляемся от alpakka

Решение 2 – избавляемся от alpakka



Решение 2 – избавляемся от alpakka

Проблемка с зависимостью потоков

Решение 2 – избавляемся от alpakka

Проблемка с новым форматом данных

Проблемка с зависимостью потоков

Решение 2 – избавляемся от alpakka

Решение 2 – избавляемся от alpakka

- Потоки (Flow) влияют друг на друга

Решение 2 – избавляемся от alpakka

- Потоки (Flow) влияют друг на друга
- Всё ещё большое количество акторов

Решение 2 – избавляемся от alpakka

- Потоки (Flow) влияют друг на друга
- Всё ещё большое количество акторов
- Всё ещё плохая утилизация ресурсов

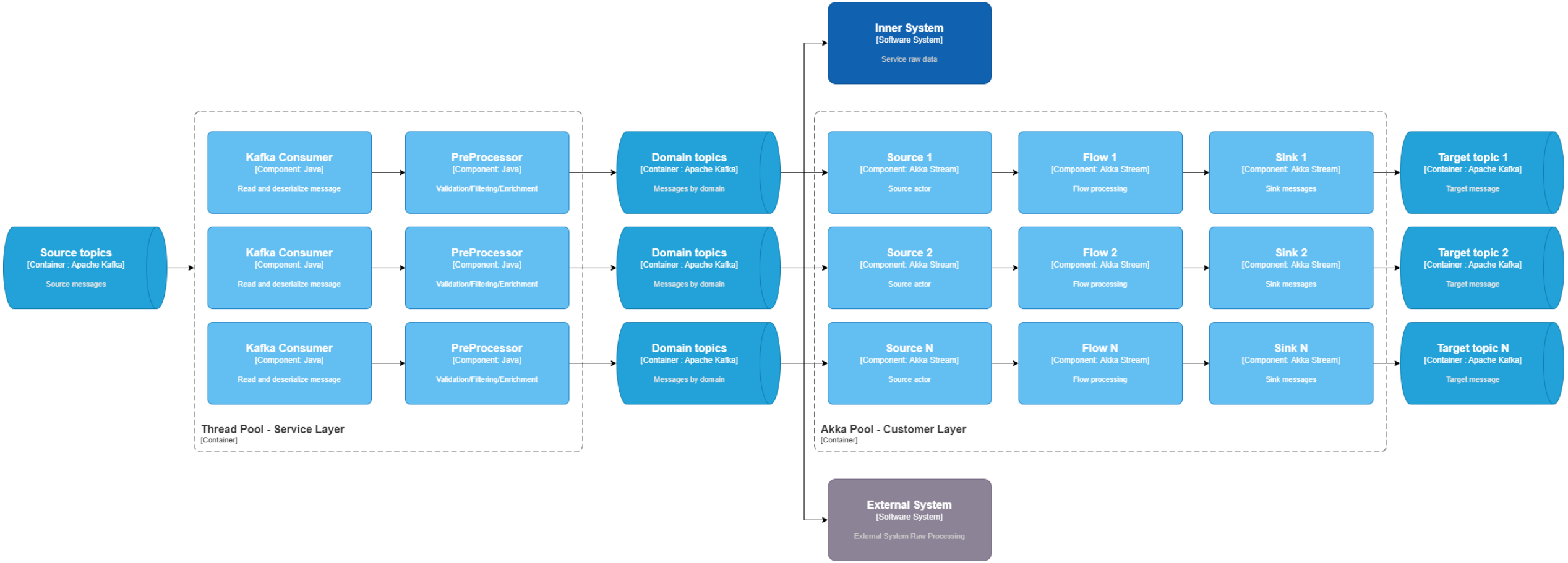
Решение 2 – избавляемся от alpakka

- Потоки (Flow) влияют друг на друга
- Всё ещё большое количество акторов
- Всё ещё плохая утилизация ресурсов
- Новый источник – рестарт приложения

Решение 2 – избавляемся от alpakka

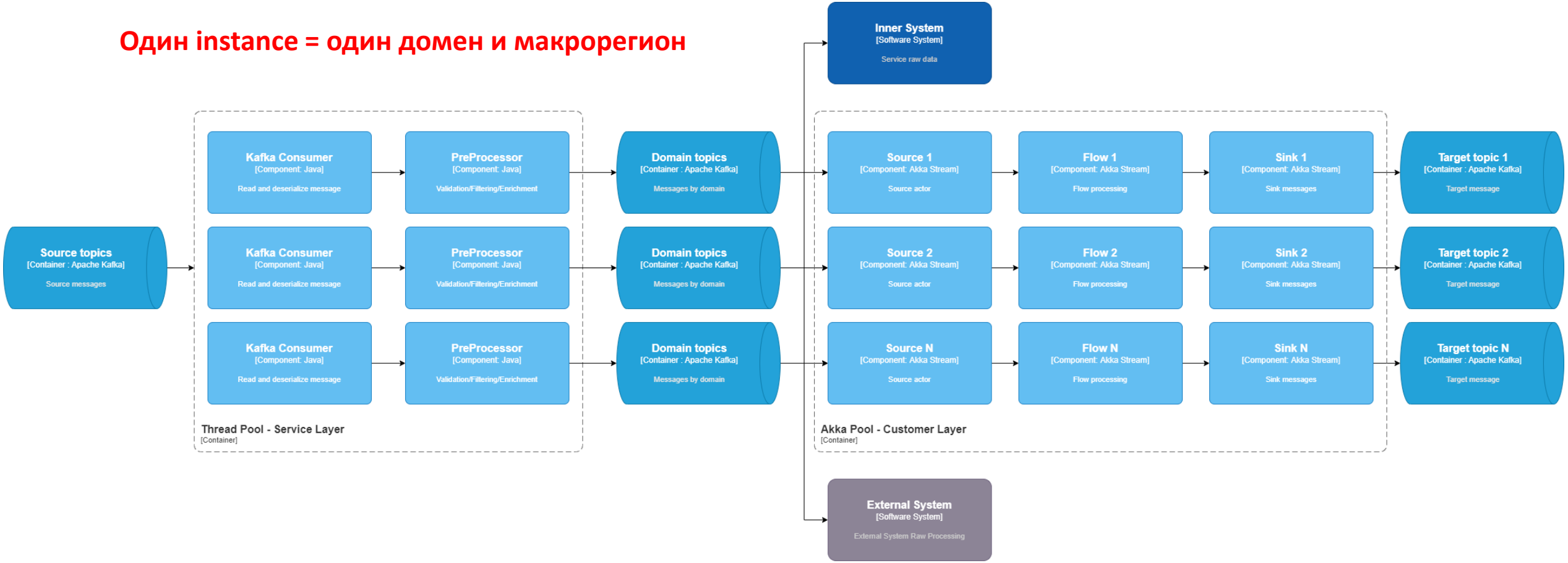
- Потоки (Flow) влияют друг на друга
- Всё ещё большое количество акторов
- Всё ещё плохая утилизация ресурсов
- Новый источник – рестарт приложения

Решение 3 – зоны ответственности



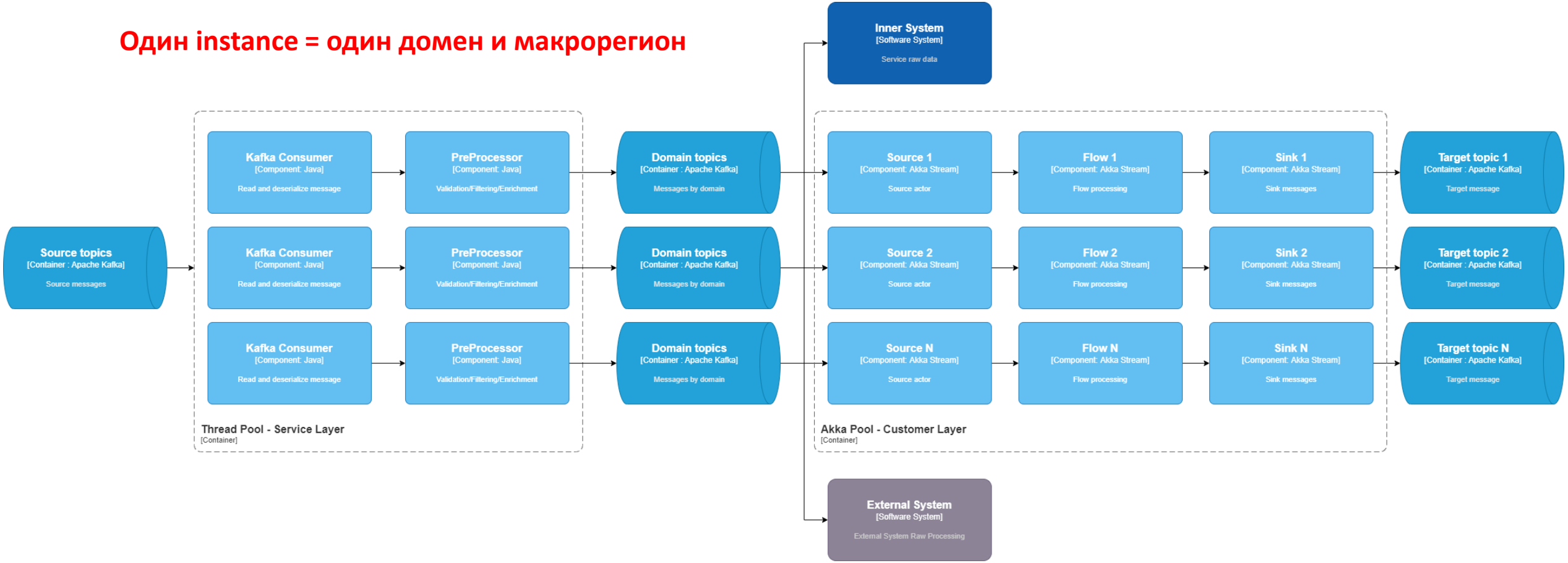
Решение 3 – зоны ответственности

Один instance = один домен и макрорегион



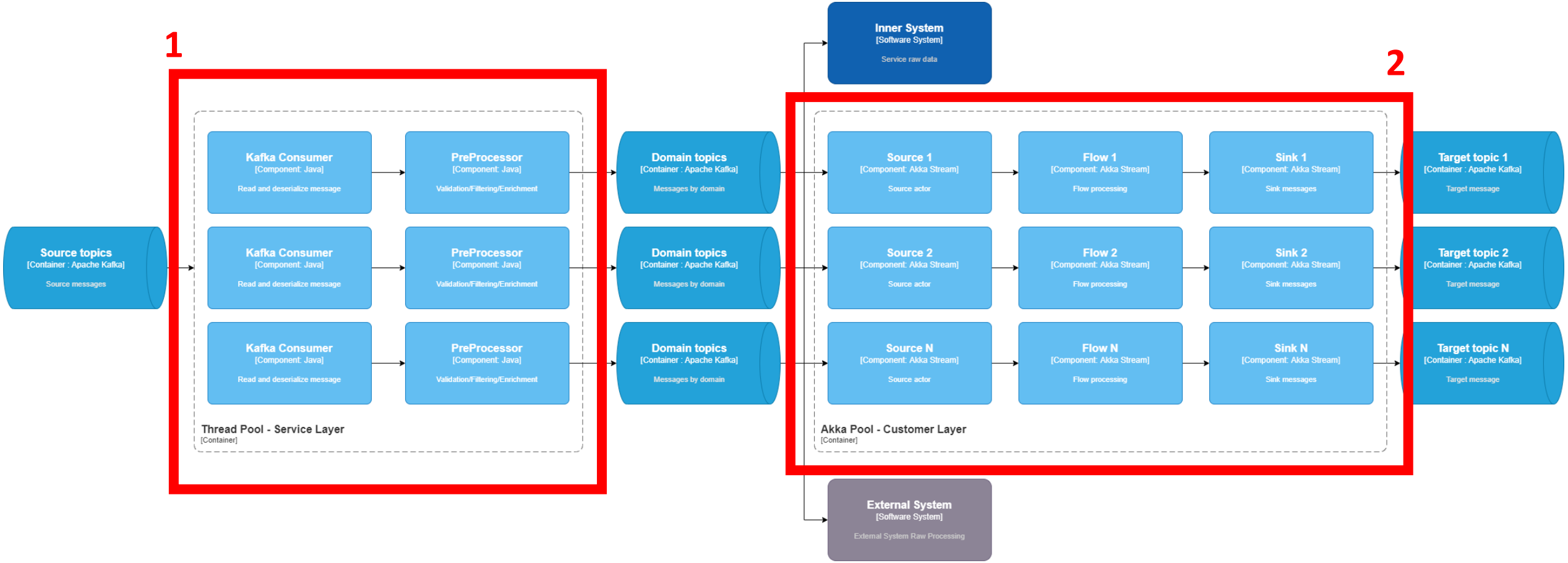
Решение 3 – зоны ответственности

Один instance = один домен и макрорегион

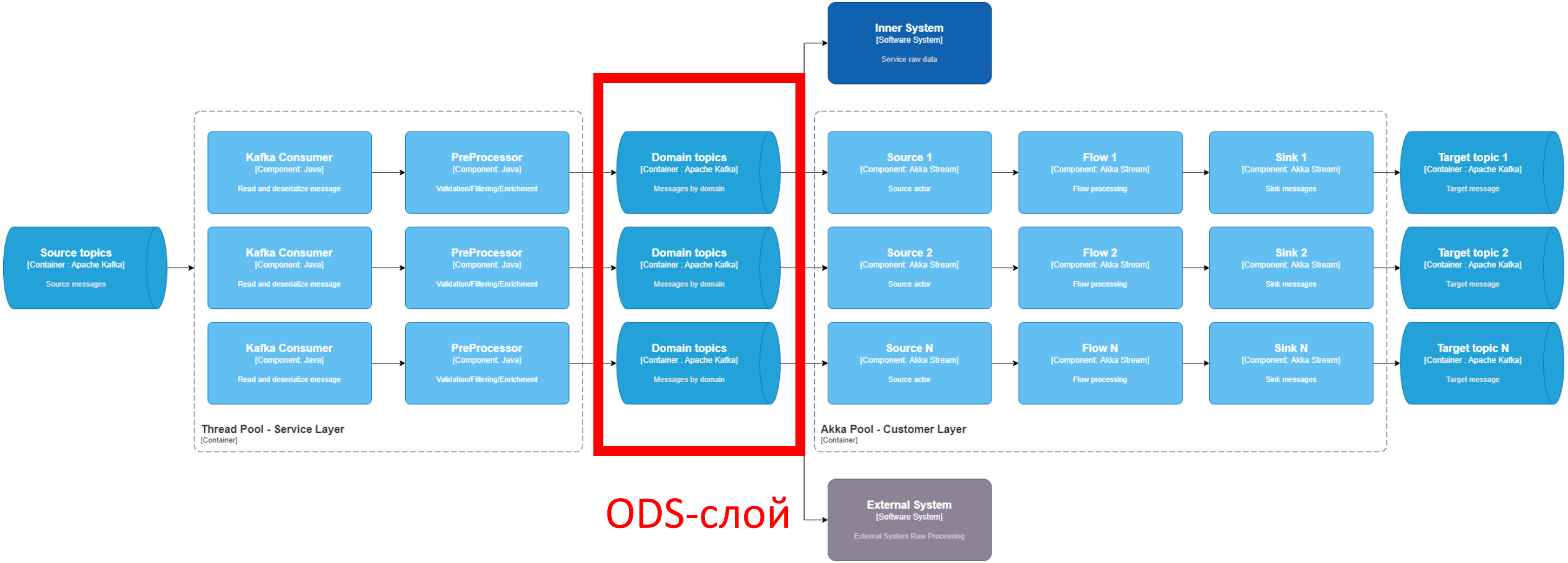


Пример: instance для url по Сибири

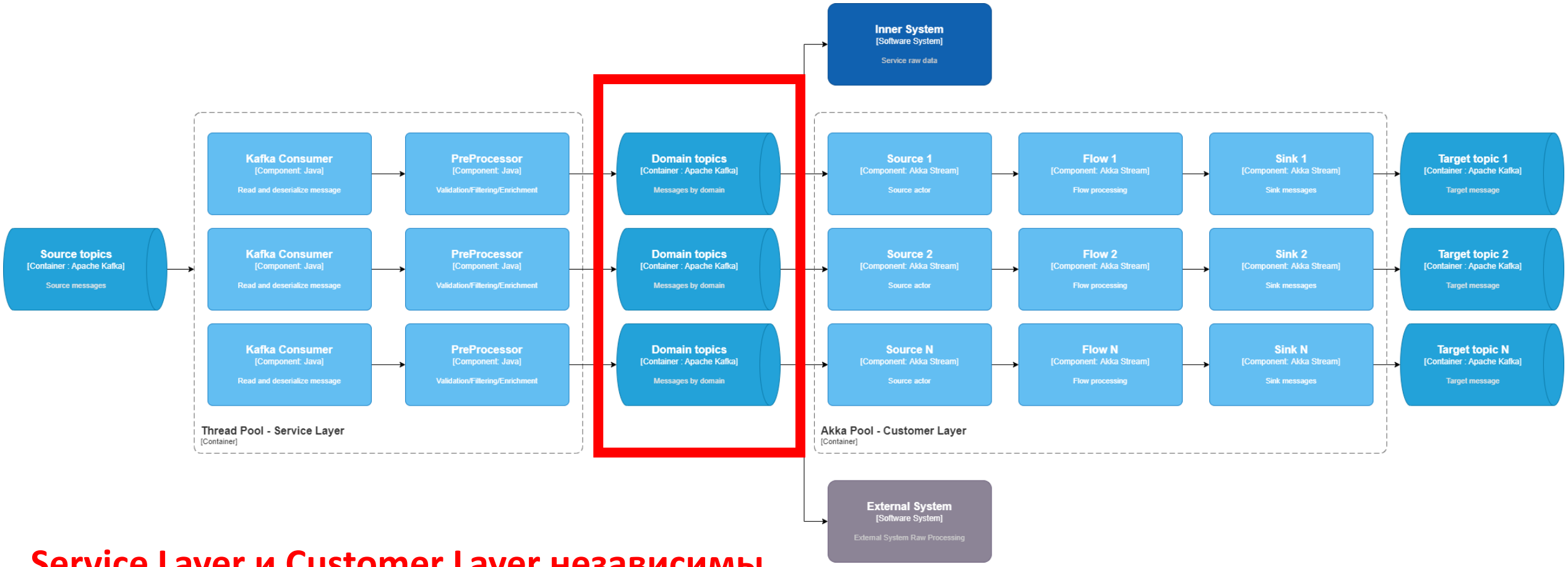
Решение 3 – зоны ответственности



Решение 3 – зоны ответственности

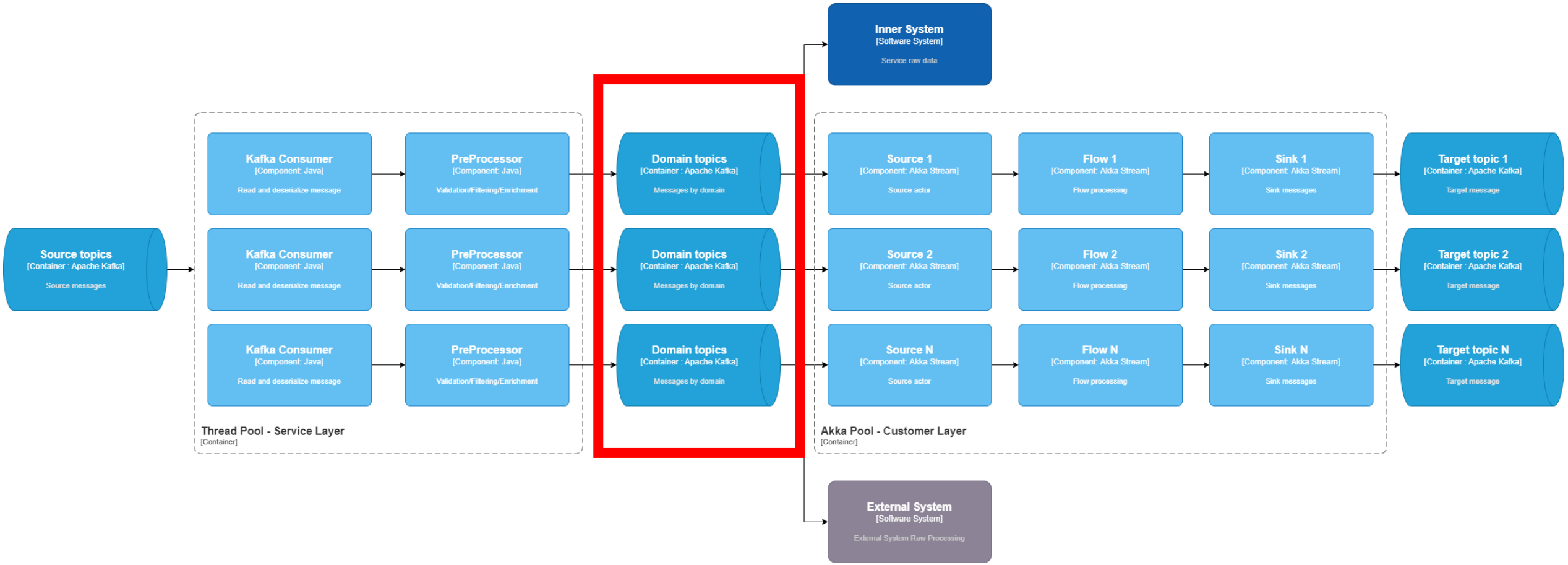


Решение 3 – зоны ответственности



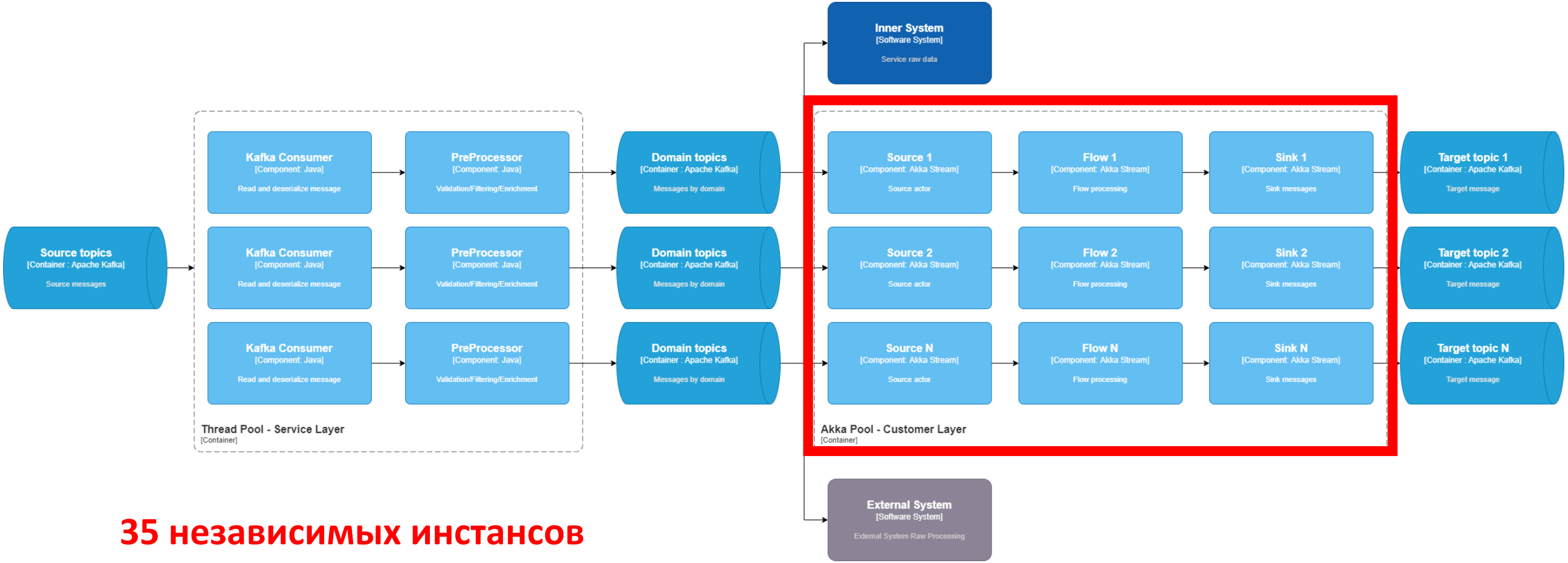
Service Layer и Customer Layer независимы

Решение 3 – зоны ответственности



Добавление нового источника – рестарт только Service Layer

Решение 3 – зоны ответственности



35 независимых инстансов

Решение 3 – зоны ответственности

Решение 3 – зоны ответственности

- Система акторов на 1 макрорегион на 1 домен данных

Решение 3 – зоны ответственности

- Система акторов на 1 макрорегион на 1 домен данных
- Подключение новых источников не влияет на Customer

Решение 3 – зоны ответственности

- Система акторов на 1 макрорегион на 1 домен данных
- Подключение новых источников не влияет на Customer
- Независимые процессы CI/CD и конфигурации

Решение 3 – зоны ответственности

- Система акторов на 1 макрорегион на 1 домен данных
- Подключение новых источников не влияет на Customer
- Независимые процессы CI/CD и конфигурации
- Балансировка нагрузки

Решение 3 – зоны ответственности

- Система акторов на 1 макрорегион на 1 домен данных
- Подключение новых источников не влияет на Customer
- Независимые процессы CI/CD и конфигурации
- Балансировка нагрузки
- Чтение по ключам на instance

Решение 3 – зоны ответственности

- Система акторов на 1 макрорегион на 1 домен данных
- Подключение новых источников не влияет на Customer
- Независимые процессы CI/CD и конфигурации
- Балансировка нагрузки
- Чтение по ключам на instance
- Миллионы событий в секунду

Скелет Streaming Flow

```
var actor = createSourceActor()
        .filter(elem -> {…})
        .via(workFlow.processingStream(flowConfiguration))
        .viaMat(KillSwitches.single(), Keep.both())
        .to(Sink.actorRef(getProducer(), COMPLETE_MESSAGE))
        .run(actorSystem);

sourceActors.get(getDomain()).add(actor.first());

createConsumers(getDomain(), sourceActors);
```

Скелет Streaming Flow

```
var actor = createSourceActor()
        .filter(elem -> {…})
        .via(workFlow.processingStream(flowConfiguration))
        .viaMat(KillSwitches.single(), Keep.both())
        .to(Sink.actorRef(getProducer(), COMPLETE_MESSAGE))
        .run(actorSystem);

sourceActors.get(getDomain()).add(actor.first());

createConsumers(getDomain(), sourceActors);
```

Скелет Streaming Flow

```
var actor = createSourceActor()
        .filter(elem -> {…})
        .via(workFlow.processingStream(flowConfiguration))
        .viaMat(KillSwitches.single(), Keep.both())
        .to(Sink.actorRef(getProducer(), COMPLETE_MESSAGE))
        .run(actorSystem);

sourceActors.get(getDomain()).add(actor.first());

createConsumers(getDomain(), sourceActors);
```

Скелет Streaming Flow

```
var actor = createSourceActor()
        .filter(elem -> {…})
        .via(workFlow.processingStream(flowConfiguration))
        .viaMat(KillSwitches.single(), Keep.both())
        .to(Sink.actorRef(getProducer(), COMPLETE_MESSAGE))
        .run(actorSystem);

sourceActors.get(getDomain()).add(actor.first());

createConsumers(getDomain(), sourceActors);
```

Скелет Streaming Flow

```
var actor = createSourceActor()
        .filter(elem -> {…})
        .via(workFlow.processingStream(flowConfiguration))
        .viaMat(KillSwitches.single(), Keep.both())
        .to(Sink.actorRef(getProducer(), COMPLETE_MESSAGE))
        .run(actorSystem);

sourceActors.get(getDomain()).add(actor.first());

createConsumers(getDomain(), sourceActors);
```

Скелет Streaming Flow

```
var actor = createSourceActor()
        .filter(elem -> {…})
        .via(workFlow.processingStream(flowConfiguration))
        .viaMat(KillSwitches.single(), Keep.both())
        .to(Sink.actorRef(getProducer(), COMPLETE_MESSAGE))
        .run(actorSystem);

sourceActors.get(getDomain()).add(actor.first());

createConsumers(getDomain(), sourceActors);
```

Скелет Streaming Flow

```
var actor = createSourceActor()
        .filter(elem -> {…})
        .via(workFlow.processingStream(flowConfiguration))
        .viaMat(KillSwitches.single(), Keep.both())
        .to(Sink.actorRef(getProducer(), COMPLETE_MESSAGE))
        .run(actorSystem);

sourceActors.get(getDomain()).add(actor.first());

createConsumers(getDomain(), sourceActors);
```

Скелет - Source

```
private Source<AbstractEvent, ActorRef> createSourceActor() {
    return Source.actorRef(elem -> {
                if (elem == Done.done()) {
                    return Optional.of(CompletionStrategy.immediately());
                } else {
                    return Optional.empty();
                }
            },
            elem -> Optional.empty(),
            actorBufferSize,
            OverflowStrategy.dropHead()
    );
}
```

Скелет - Source

```
private Source<AbstractEvent, ActorRef> createSourceActor() {
    return Source.actorRef(elem -> {
                if (elem == Done.done()) {
                    return Optional.of(CompletionStrategy.immediately());
                } else {
                    return Optional.empty();
                }
            },
            elem -> Optional.empty(),
            actorBufferSize,
            OverflowStrategy.dropHead()
    );
}
```

Скелет - Source

```
private Source<AbstractEvent, ActorRef> createSourceActor() {
    return Source.actorRef(elem -> {
                if (elem == Done.done()) {
                    return Optional.of(CompletionStrategy.immediately());
                } else {
                    return Optional.empty();
                }
            },
            elem -> Optional.empty(),
            actorBufferSize,
            OverflowStrategy.dropHead()
    );
}
```

Скелет - Source

```
private Source<AbstractEvent, ActorRef> createSourceActor() {
    return Source.actorRef(elem -> {
                if (elem == Done.done()) {
                    return Optional.of(CompletionStrategy.immediately());
                } else {
                    return Optional.empty();
                }
            },
            elem -> Optional.empty(),
            actorBufferSize,
            OverflowStrategy.dropHead()
    );
}
```

.backpresurre()

Скелет - Sink

```
public class KafkaProducerActor extends AbstractActor {

   private final Producer producer;

   …

   @Override
   public Receive createReceive() {
      return receiveBuilder()
            .match(ProducerMessage.class, message -> { producer.send(message); })
            .matchAny(unknown -> log.error("Received unknown message {}", unknown))
            .build();
   }

   …
}
```

Скелет - Sink

```
public class KafkaProducerActor extends AbstractActor {

   private final Producer producer;

   …

   @Override
   public Receive createReceive() {
      return receiveBuilder()
            .match(ProducerMessage.class, message -> { producer.send(message); })
            .matchAny(unknown -> log.error("Received unknown message {}", unknown))
            .build();
   }

   …
}
```

Скелет - Sink

```
public class KafkaProducerActor extends AbstractActor {

   private final Producer producer;

   …

   @Override
   public Receive createReceive() {
      return receiveBuilder()
            .match(ProducerMessage.class, message -> { producer.send(message); })
            .matchAny(unknown -> log.error("Received unknown message {}", unknown))
            .build();
   }

   …
}
```

Скелет – RunnableGraph

```
Graph<FlowShape<AbstractEvent, Message>, NotUsed> processingStream () {
    return GraphDSL.create(
            builder -> {

                …
                FlowShape<AbstractEvent, Message> flow = builder.add(
                      Flow.of(AbstractEvent.class)
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .async()
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .filter(…)
                );
                return FlowShape.of(flow.in(), flow.out());
            });
}
```

Скелет – RunnableGraph

```
Graph<FlowShape<AbstractEvent, Message>, NotUsed> processingStream () {
    return GraphDSL.create(
            builder -> {

                …
                FlowShape<AbstractEvent, Message> flow = builder.add(
                      Flow.of(AbstractEvent.class)
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .async()
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .filter(…)
                );
                return FlowShape.of(flow.in(), flow.out());
            });
}
```

Скелет – RunnableGraph

```
Graph<FlowShape<AbstractEvent, Message>, NotUsed> processingStream () {
    return GraphDSL.create(
            builder -> {

                …
                FlowShape<AbstractEvent, Message> flow = builder.add(
                      Flow.of(AbstractEvent.class)
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .async()
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .filter(…)
                );
                return FlowShape.of(flow.in(), flow.out());
            });
}
```

Скелет – RunnableGraph

```
Graph<FlowShape<AbstractEvent, Message>, NotUsed> processingStream () {
    return GraphDSL.create(
            builder -> {

                …
                FlowShape<AbstractEvent, Message> flow = builder.add(
                      Flow.of(AbstractEvent.class)
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .async()
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .filter(…)
                );
                return FlowShape.of(flow.in(), flow.out());
            });
}
```

Скелет – RunnableGraph

```
Graph<FlowShape<AbstractEvent, Message>, NotUsed> processingStream () {
    return GraphDSL.create(
            builder -> {

                …
                FlowShape<AbstractEvent, Message> flow = builder.add(
                      Flow.of(AbstractEvent.class)
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .async()
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .filter(…)
                );
                return FlowShape.of(flow.in(), flow.out());
            });
}
```

Скелет – RunnableGraph

```
Graph<FlowShape<AbstractEvent, Message>, NotUsed> processingStream () {
    return GraphDSL.create(
            builder -> {

                …
                FlowShape<AbstractEvent, Message> flow = builder.add(
                      Flow.of(AbstractEvent.class)
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .async()
                        .mapAsyncUnordered(parallelismMapAsync, dto -> {…})
                        .filter(…)
                );
                return FlowShape.of(flow.in(), flow.out());
            });
}
```

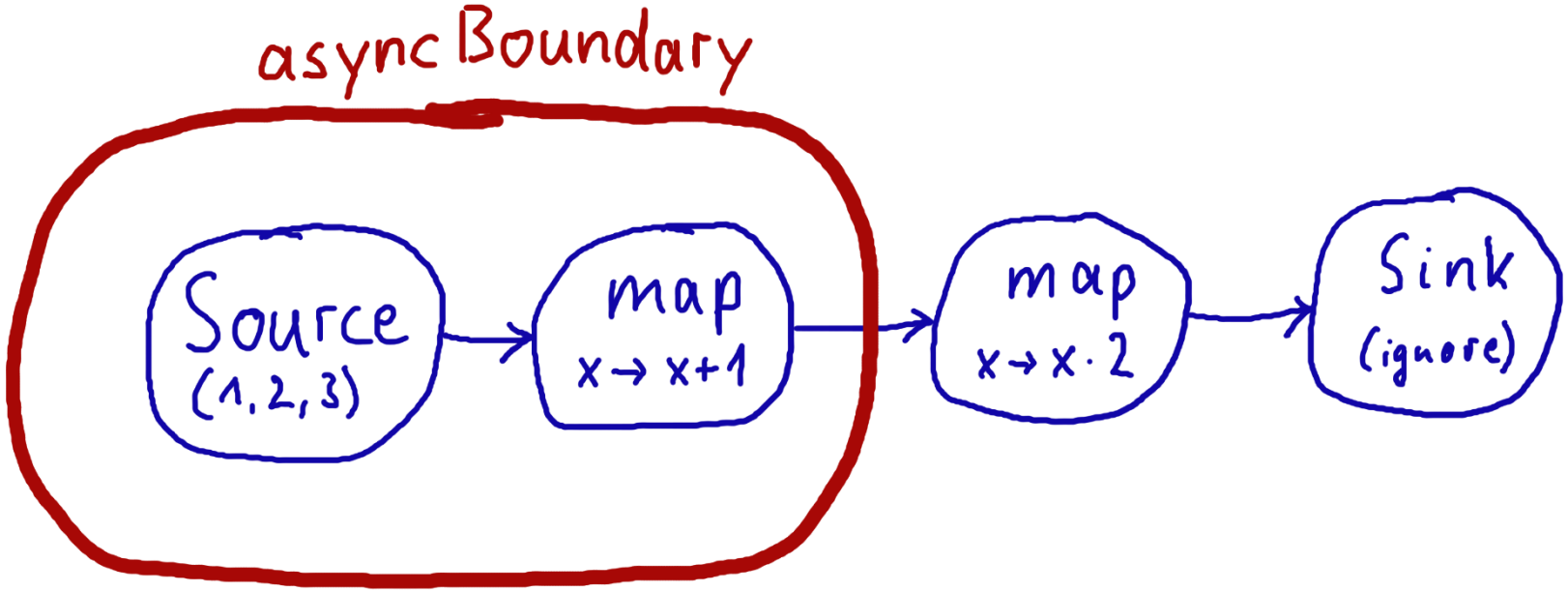
Решение 3.1 – тюниг

Решение 3.1 – тюниг

- Настройка GC – G1 vs ZGC vs CMS

Решение 3.1 – тюниг

- Настройка GC – G1 vs ZGC vs CMS
- Границы асинхронности для Flow



Решение 3.1 – тюниг

- Настройка GC – G1 vs ZGC vs CMS
- Границы асинхронности для Flow
- OverflowStrategy – DROP vs Backpressure

Решение 3.1 – тюниг

- Настройка GC – G1 vs ZGC vs CMS
- Границы асинхронности для Flow
- OverflowStrategy – DROP vs Backpressure
- Autoscaling

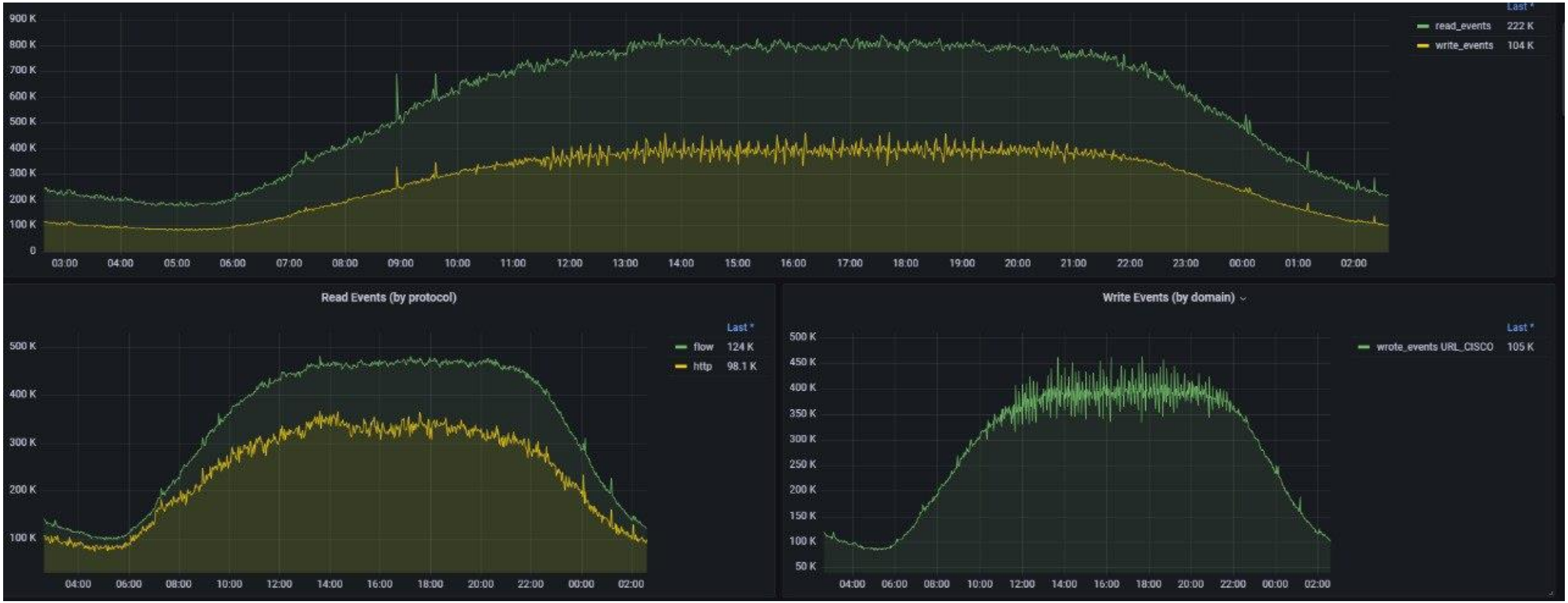
Мониторинг

- Kafka Lags
- Actor Buffers
- Метрики времени обработки (Prometheus - Histogram)

Мониторинг


Kubernetes
CPU utilization
requests 37.1%
limits 18.5%
RAM utilization
requests 96.2%
limits 96.2%
deployment replicas
2
0
pod restarts total
0
tag
1.16.0-release
CPU utilization by pod
pod
requests
limits
shrinker-engine-prod1-...
41.5%
20.7%
shrinker-engine-prod1-...
32.7%
16.4%
RAM utilization by pod
pod
requests
limits
shrinker-engine-prod1-...
96.2%
96.2%
shrinker-engine-prod1-...
96.2%
96.2%
kube_pod_status_phase
Failed 0
Pending 0
Running 2
Succeeded 0
Unknown 0
terminated_reason
No data
kube_pod_status_reason
Evicted 0
NodeLost 0
UnexpectedAdmissionError 0

Мониторинг


Last *
read_events 222 K
write_events 104 K
Read Events (by protocol)
Last *
flow 124 K
http 98.1 K
Write Events (by domain)
Last *
wrote_events URL_CISCO 105 K

Мониторинг

general

Availability	Restarts	CPU Requests	CPU usage (requests	CPU usage (limits)	Heap Size	RAM Requests	RAM usage (requests	RAM usage (limits)	Replicas Available	Replicas Unavailable	Phase Problems	Phase Reason	GC PS MarkSweep	GC PS Scavenge
100%	0	1	15.5%	3.87%	863 MiB	20 GiB	34.8%	34.8%	1	0	0	0	0 s	181 ms
100%	0	1	17.4%	4.36%	333 MiB	16 GiB	39.7%	39.7%	1	0	0	0	0 s	165 ms
100%	0	1	20.2%	5.04%	1.91 GiB	32 GiB	25.5%	25.5%	1	0	0	0	0 s	212 ms
100%	0	1	20.8%	5.19%	793 MiB	20 GiB	34.0%	34.0%	1	0	0	0	0 s	261 ms
100%	0	1	21.9%	5.47%	2.27 GiB	32 GiB	28.5%	28.5%	1	0	0	0	0 s	225 ms
100%	0	1	31.6%	7.90%	1.06 GiB	16 GiB	45.1%	45.1%	1	0	0	0	0 s	214 ms
100%	0	1	37.9%	9.48%	1.67 GiB	20 GiB	38.7%	38.7%	1	0	0	0	0 s	365 ms

Инфраструктура – K8s

5 нод:

Cores	vCores	RAM GB	HDD GB
36	72	1512	462

5 нод:

Cores	vCores	RAM GB	HDD GB
48	96	1512	4692

2 ноды:

Cores	vCores	RAM GB	HDD GB
36	72	384	6793

Инфраструктура - Kafka

7 нод:

Cores	vCores	RAM GB	HDD GB	RAID
36	72	1008	8617	6x1.636GB RAID5

Итоги

Итоги

- Теория в документации не равно реальные задачи

Итоги

- Теория в документации не равно реальные задачи
- Дополнительные слои абстракции могут быть не надёжны

Итоги

- Теория в документации не равно реальные задачи
- Дополнительные слои абстракции могут быть не надёжны
- Меньше акторов в системе акторов – выше производительность

Итоги

- Теория в документации не равно реальные задачи
- Дополнительные слои абстракции могут быть не надёжны
- Меньше акторов в системе акторов – выше производительность
- Статичная обработка в собственный thread pool

Итоги

- Теория в документации не равно реальные задачи
- Дополнительные слои абстракции могут быть не надёжны
- Меньше акторов в системе акторов – выше производительность
- Статичная обработка в собственный thread pool
- Тюнинг Akka, JVM (GC) - обязательно


Akka Stream
Опять ничего не получается
Разработчик
И так тоже не получается
ну и пожалуйста
Разработчик
Akka Stream
ну и не надо
Разработчик
Akka Stream

Чек-лист по использованию Akka Streams

Чек-лист по использованию Akka Streams

- ❑Средние потоки данных (сообщений в секунду)

Чек-лист по использованию Akka Streams

- ❑ Средние потоки данных (сообщений в секунду)
- ❑ Нет требования по exactly once

Чек-лист по использованию Akka Streams

- ❑Средние потоки данных (сообщений в секунду)
- ❑Нет требования по exactly once
- ❑Не страшно что-то потерять (отбросить)

Чек-лист по использованию Akka Streams

- ❑Средние потоки данных (сообщений в секунду)
- ❑Нет требования по exactly once
- ❑Не страшно что-то потерять (отбросить)
- ❑Нужно минимальное latency

Чек-лист по использованию Akka Streams

- ❑Средние потоки данных (сообщений в секунду)
- ❑Нет требования по exactly once
- ❑Не страшно что-то потерять (отбросить)
- ❑Нужно минимальное latency
- ❑Нужно сделать быстро

Чек-лист по использованию Akka Streams

- ❑Средние потоки данных (сообщений в секунду)
- ❑Нет требования по exactly once
- ❑Не страшно что-то потерять (отбросить)
- ❑Нужно минимальное latency
- ❑Нужно сделать быстро
- ❑Нужно сделать очень гибко

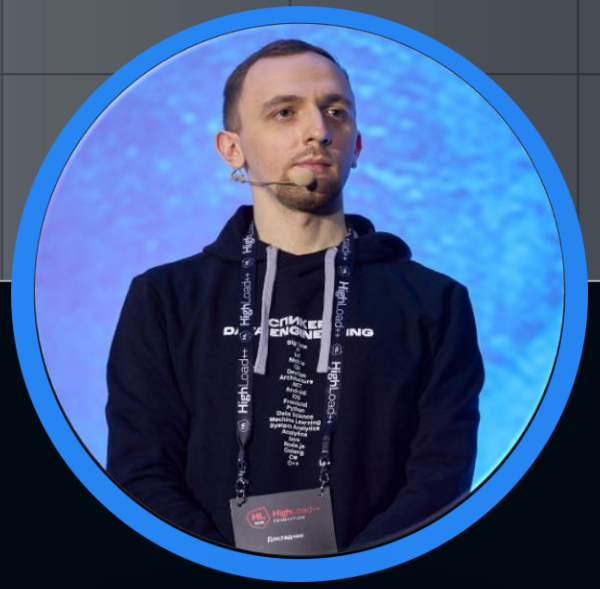
Чек-лист по использованию Akka Streams

- ❑Средние потоки данных (сообщений в секунду)
- ❑Нет требования по exactly once
- ❑Не страшно что-то потерять (отбросить)
- ❑Нужно минимальное latency
- ❑Нужно сделать быстро
- ❑Нужно сделать очень гибко
- ❑Низкий порог входа (Reactive Streams JDK 9)

Чек-лист по использованию Akka Streams

- ❑Средние потоки данных (сообщений в секунду)
- ❑Нет требования по exactly once
- ❑Не страшно что-то потерять (отбросить)
- ❑Нужно минимальное latency
- ❑Нужно сделать быстро
- ❑Нужно сделать очень гибко
- ❑Низкий порог входа (Reactive Streams JDK 9)
- ❑Компетенции в команде Java, Scala

Иви



Евгений Ненахов

к.ф.-м.н., Tech Lead

@neltari

newnew94@mail.ru

QR-code

МТС DIGITAL