Apache Kafka

и как это связано с DKB

Голосова Марина

НИЦ «Курчатовский Институт»

Лаборатория Больших Данных

Содержание

- …Streaming Platform/Message Broker
  - Что это за системы и для чего они нужны
- Почему Kafka?
  - Технические детали
- Kafka в DKB
  - Текущее состояние разработки
- С чего начать?
  - Краткий инструктаж

Streaming Platform/Message Broker

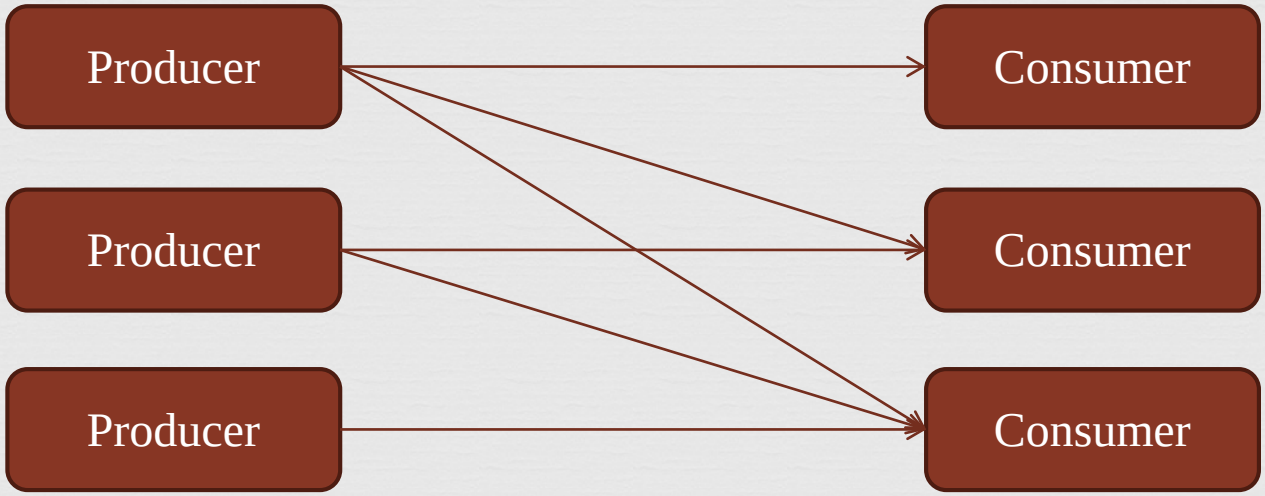
- Для чего нужны такие системы?

Streaming Platform/Message Broker

- **Основная задача**
  - Передача данных между системами/программными модулями/…
- **Проблемы**
  - Надёжность (гарантированность доставки данных)
  - Неизбыточность (доставка данных не более 1 раза)
  - Минимизация задержки (как только данные сформированы, они могут быть переданы дальше)
  - Автоматизация

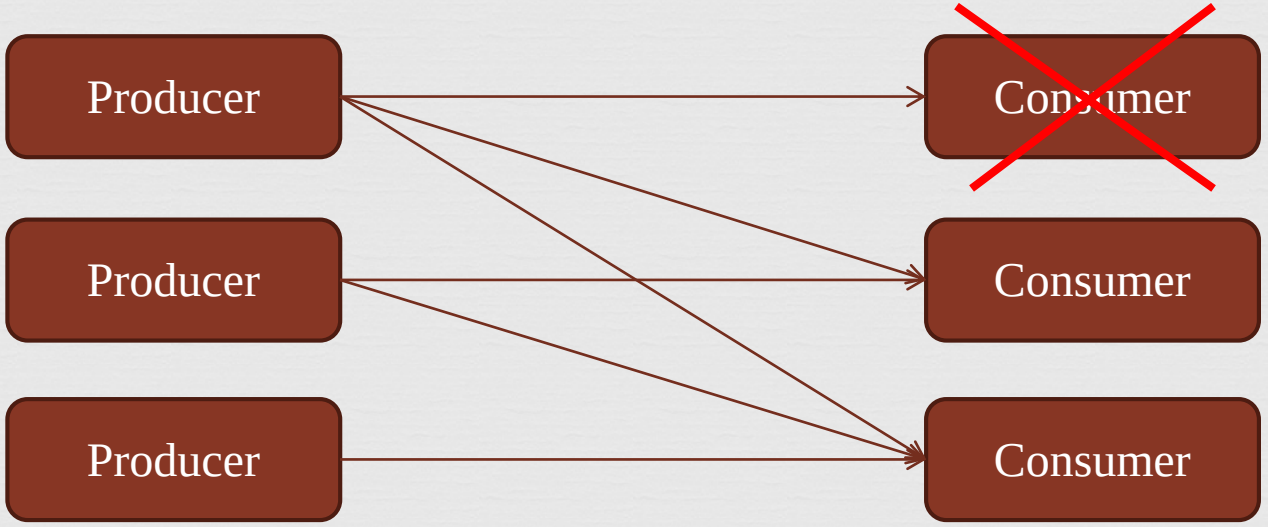
Надёжность

- Чтобы данные были гарантированно доставлены получателю, они должны быть где-то сохранены сразу после того, как они сгенерированы источником данных



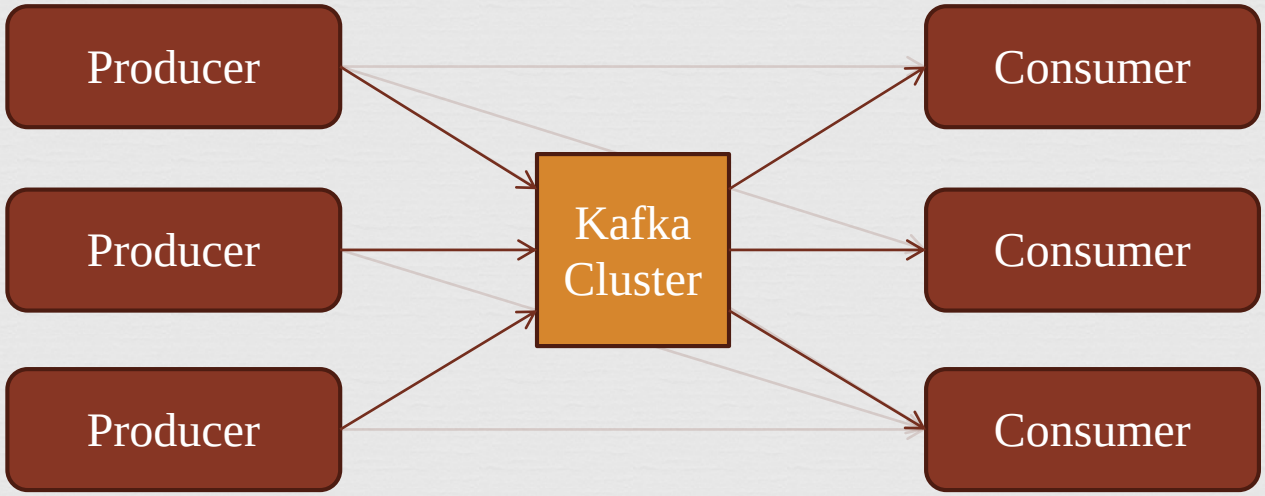
Надёжность

- Чтобы данные были гарантированно доставлены получателю, они должны быть где-то сохранены сразу после того, как они сгенерированы источником данных



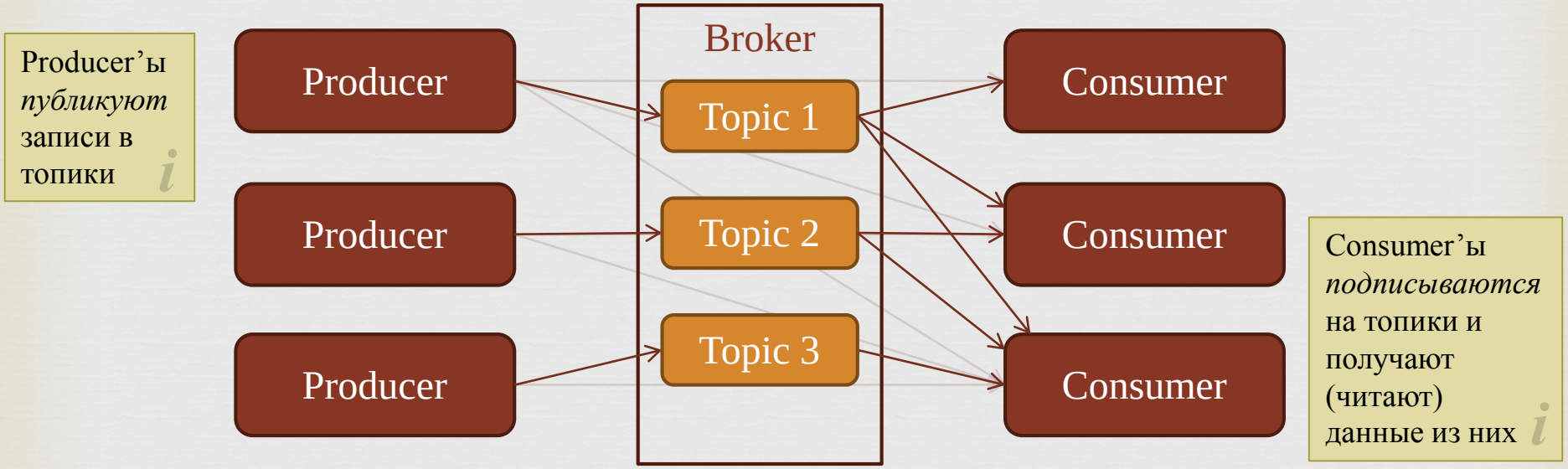
Надёжность

- Чтобы данные были гарантированно доставлены получателю, они должны быть где-то сохранены сразу после того, как они сгенерированы источником данных



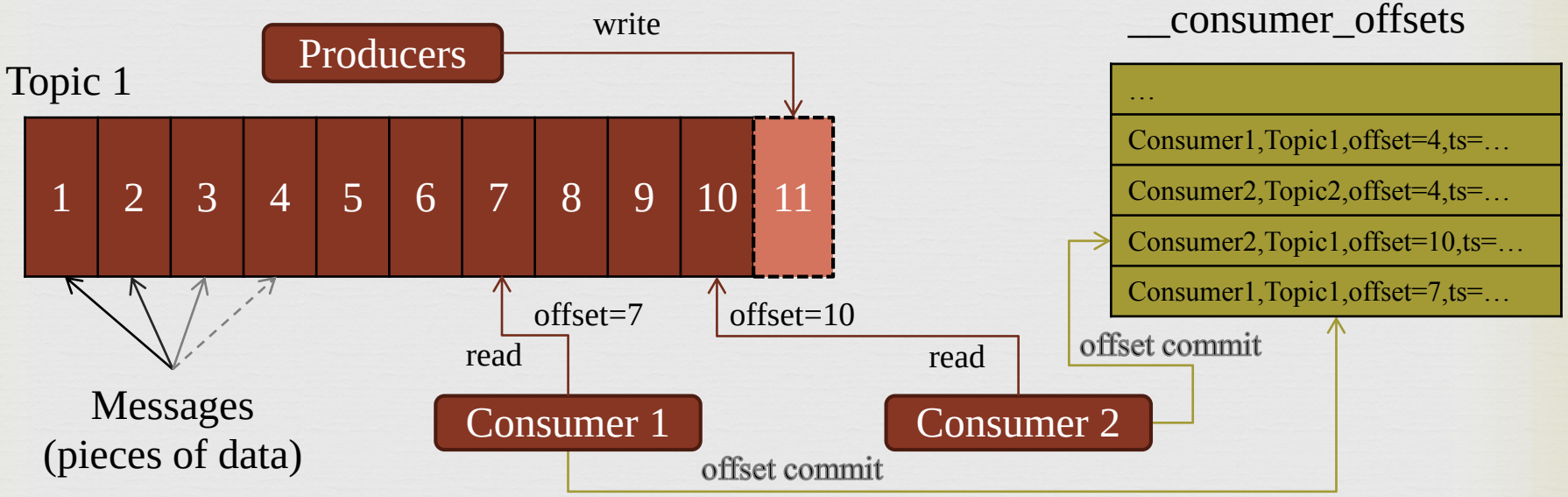
Надёжность

- Чтобы данные были гарантированно доставлены получателю, они должны быть где-то сохранены сразу после того, как они сгенерированы источником данных



Неизбыточность

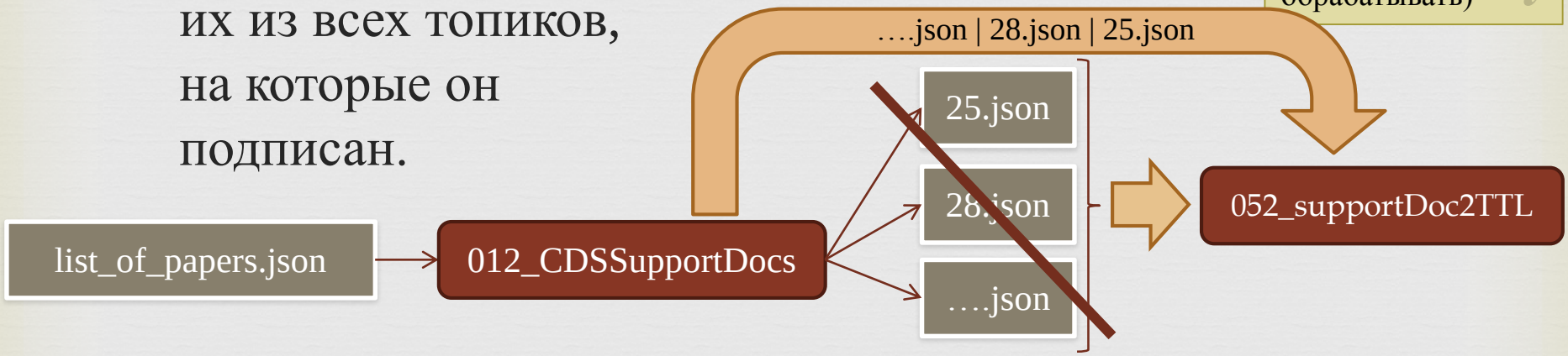
- Чтобы данные были доставлены получателю не более одного раза, необходимо хранить информацию о том, какие данные уже были доставлены



Минимизация задержки & Автоматизация

- Как только сообщение опубликовано в топик, оно доступно для чтения всем получателям, подписанным на этот топик.
- Как только получатель готов обрабатывать новые данные, он запрашивает их из всех топиков, на которые он подписан.

Данные из 25.json попадают в 052 сразу, как они опубликованы (если 052 больше нечего обрабатывать)

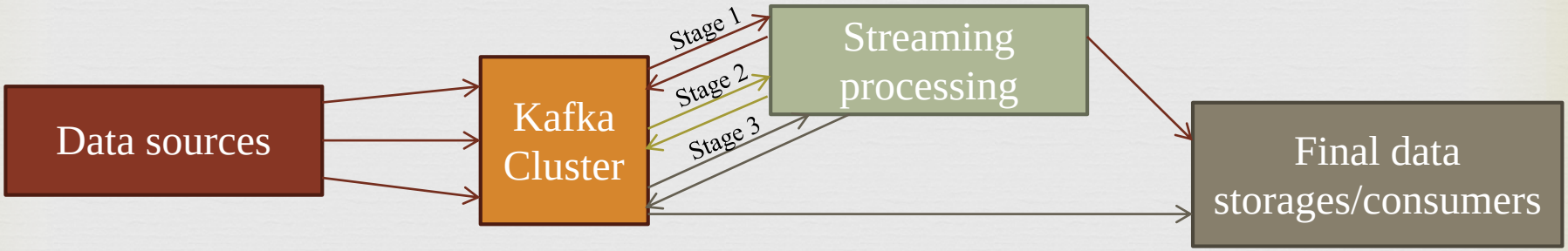


Почему Apache Kafka?

- Почему?
- Архитектура: масштабируемость
- Kafka Connect
- Потоковая обработка данных

Потоковая обработка данных

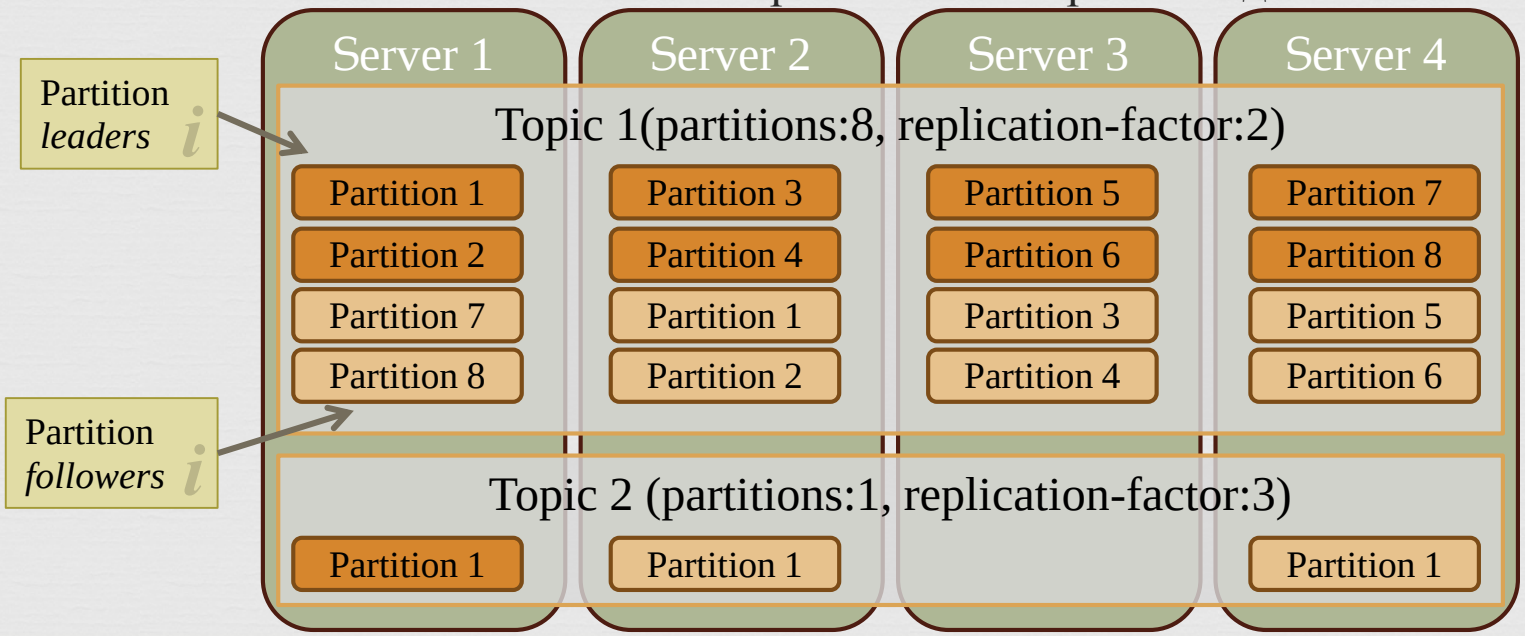
- Помимо передачи от источника получателю и тривиального преобразования данных, их в процессе нужно ещё обрабатывать и дополнять.
- Большинство систем доставки сообщений предполагают использование для этой цели внешних систем (Apache Spark, Apache Storm, etc)



- Для Apache Kafka разработана библиотека Kafka Streams, позволяющая производить потоковую обработку данных, не «выводя» их из системы (март 2016).
- **Чем меньше разных технологий – тем проще управлять зоопарком.**

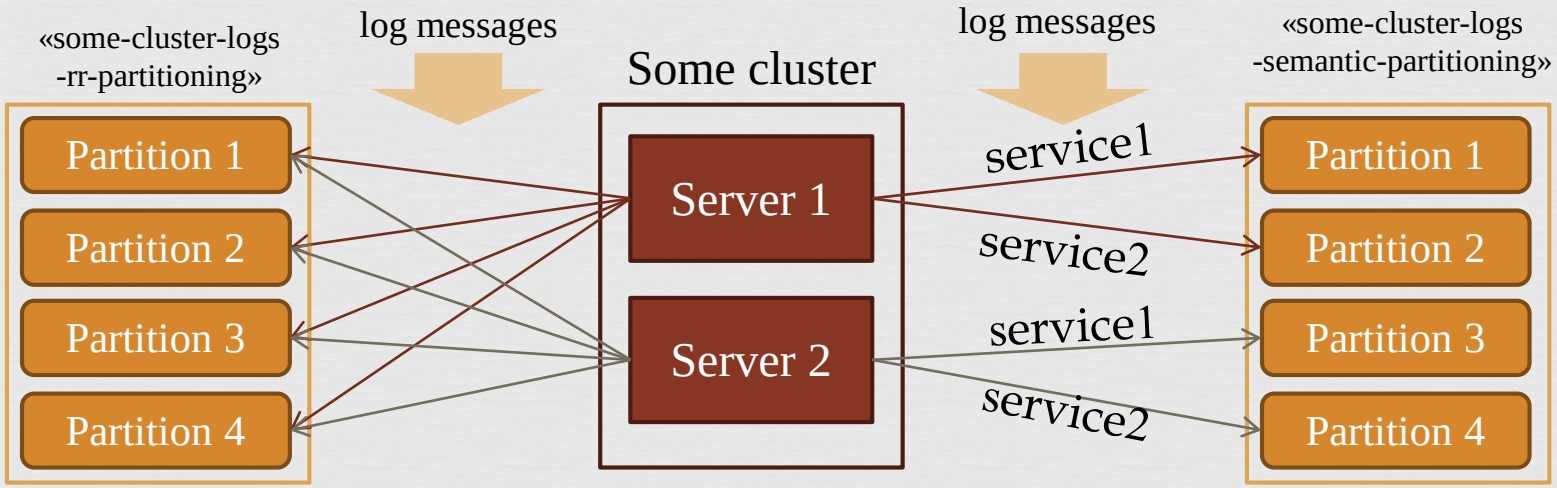
Масштабируемость: партиции

- Данные в каждом топике разделены на партиции, что
  - позволяет разделить топик между несколькими узлами кластера (горизонтальное масштабирование)
  - обеспечивает возможность параллельной обработки данных



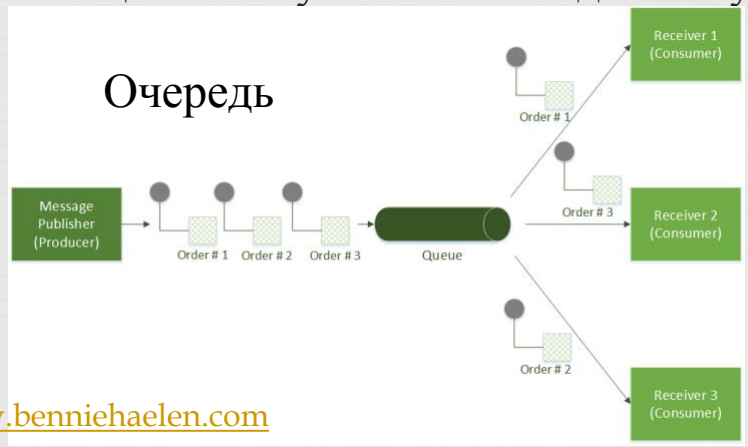
Масштабируемость: партиции

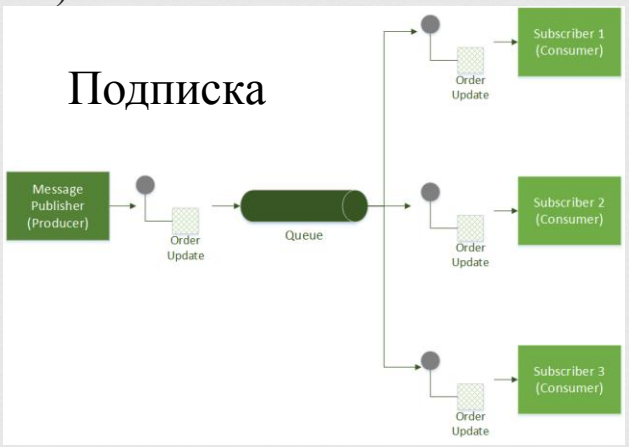
- Producer`ы публикуют данные в партиции одним из двух способов:
  - равномерно распределяя записи по партициям
  - определяя партицию семантически



Масштабируемость: партиции

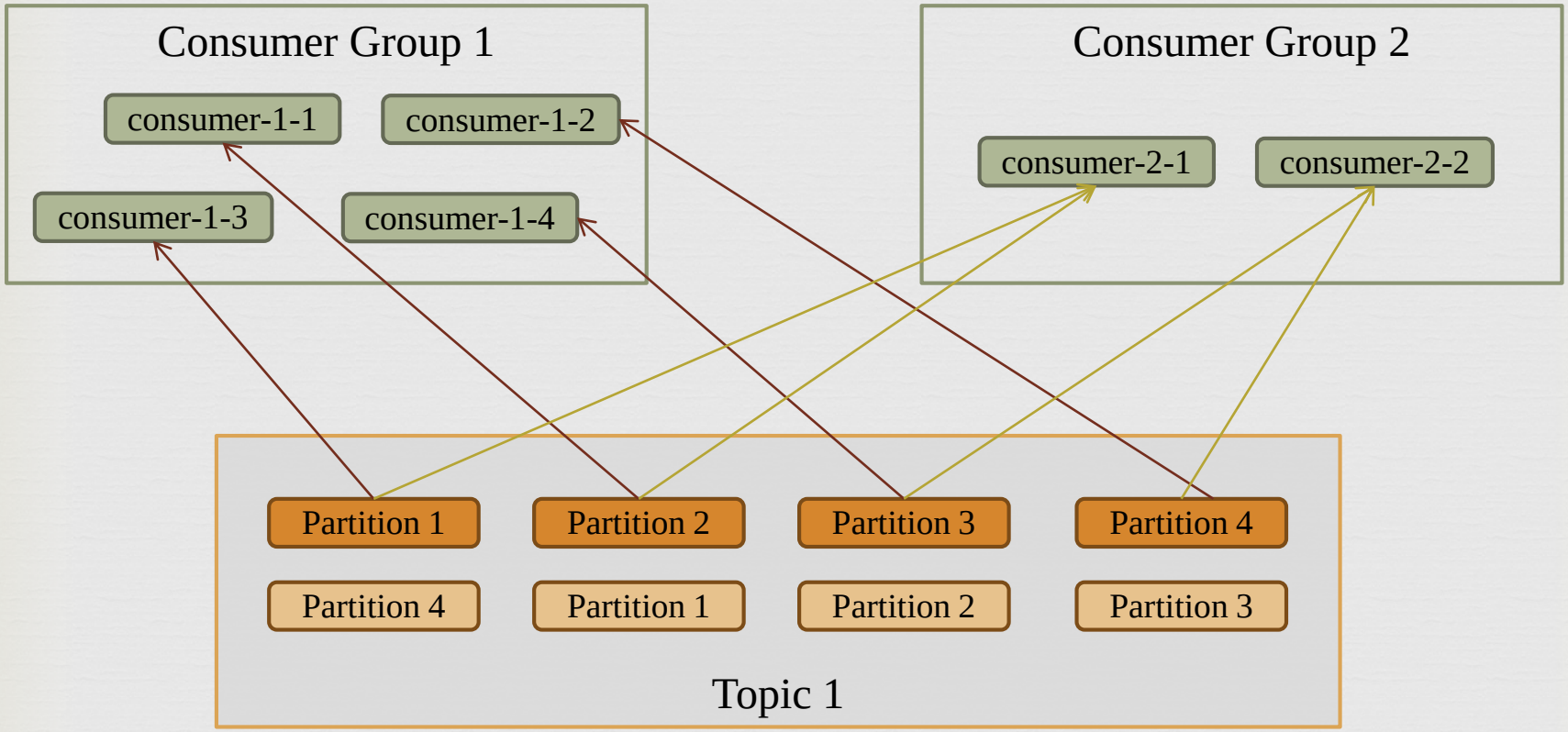
- Consumer Group – группа Consumer`ов с общим *consumer-group-id*.
- Consumer Group выступает как «логический подписчик» топика(-ов)
- Каждый Consumer получает данные из определённого набора партиций топика(-ов), на который подписана группа
- Это работает как гибрид «очереди сообщений» (гарантирующей однократность доставки сообщения) и «подписки» (гарантирующей, что сообщение получит хотя бы один получатель)



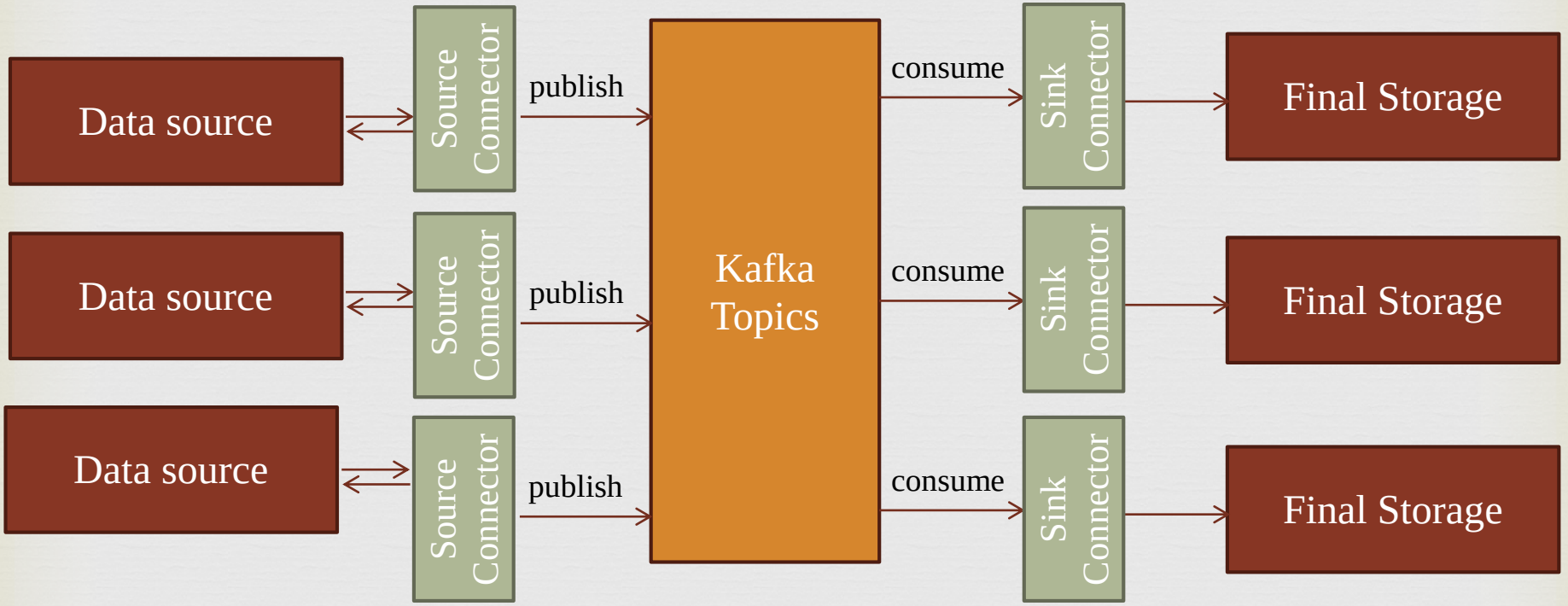


Pictures from:
http://www.benniehaelen.com

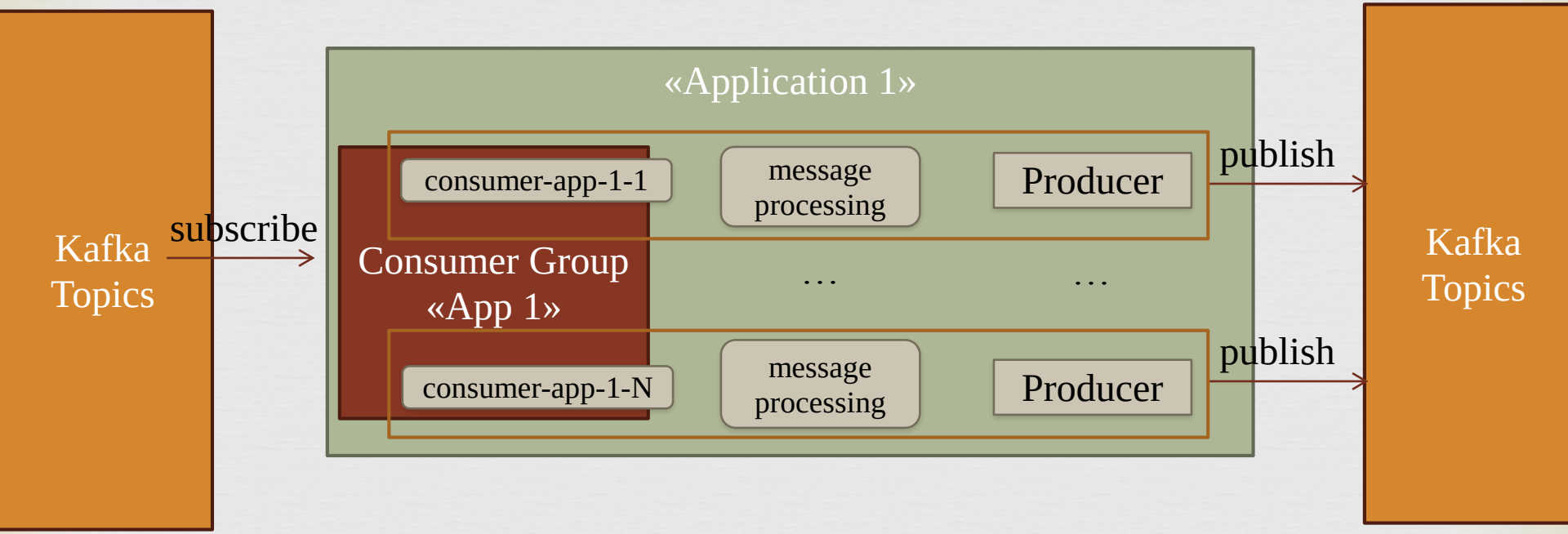
Масштабируемость: партиции



Kafka Connect



Потоковая обработка данных



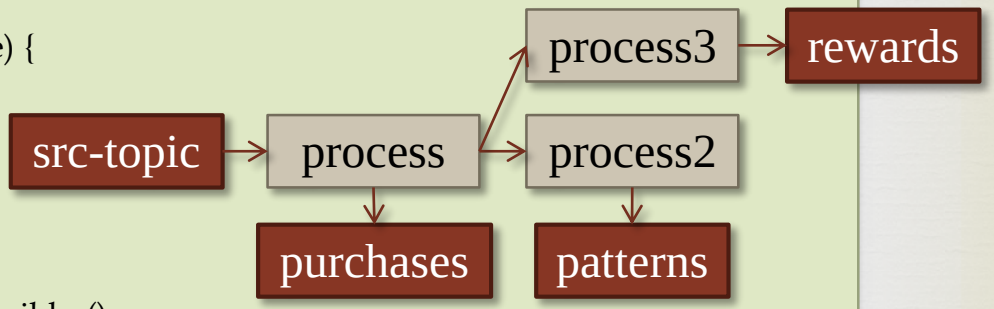
Kafka Streams

Kafka Streams – JAVA-библиотека для создания приложений потоковой обработки данных.

```
Map<String, Object> props = new HashMap<>();
props.put(StreamsConfig.APPLICATION_ID_CONFIG, "my-stream-processing-application");
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
props.put(StreamsConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
props.put(StreamsConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
props.put(StreamsConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
props.put(StreamsConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
StreamsConfig config = new StreamsConfig(props);

KStreamBuilder builder = new KStreamBuilder();
builder.from("my-input-topic").mapValue(value -> value.length().toString()).to("my-output-
topic");

KafkaStreams streams = new KafkaStreams(builder, config);
streams.start();
```

Kafka Streams: Processor API

```
public class CreditCardAnonymizer extends AbstractProcessor<String, Purchase> {
   @Override
   public void process(String key, Purchase purchase) {
   …
   }
}

…

TopologyBuilder topologyBuilder = new TopologyBuilder();
topologyBuilder.addSource("SOURCE", stringDeserializer, purchaseJsonDeserializer, "src-topic")

   .addProcessor("PROCESS", CreditCardAnonymizer::new, "SOURCE")
   .addProcessor("PROCESS2", PurchasePatterns::new, "PROCESS")
   .addProcessor("PROCESS3", CustomerRewards::new, "PROCESS")

   .addSink("SINK", "patterns", stringSerializer, purchasePatternJsonSerializer, "PROCESS2")
   .addSink("SINK2", "rewards",stringSerializer, rewardAccumulatorJsonSerializer, "PROCESS3")
   .addSink("SINK3", "purchases", stringSerializer, purchaseJsonSerializer, "PROCESS");

KafkaStreams streaming = new KafkaStreams(topologyBuilder, streamingConfig);
streaming.start();
```



Apache Kafka в DKB

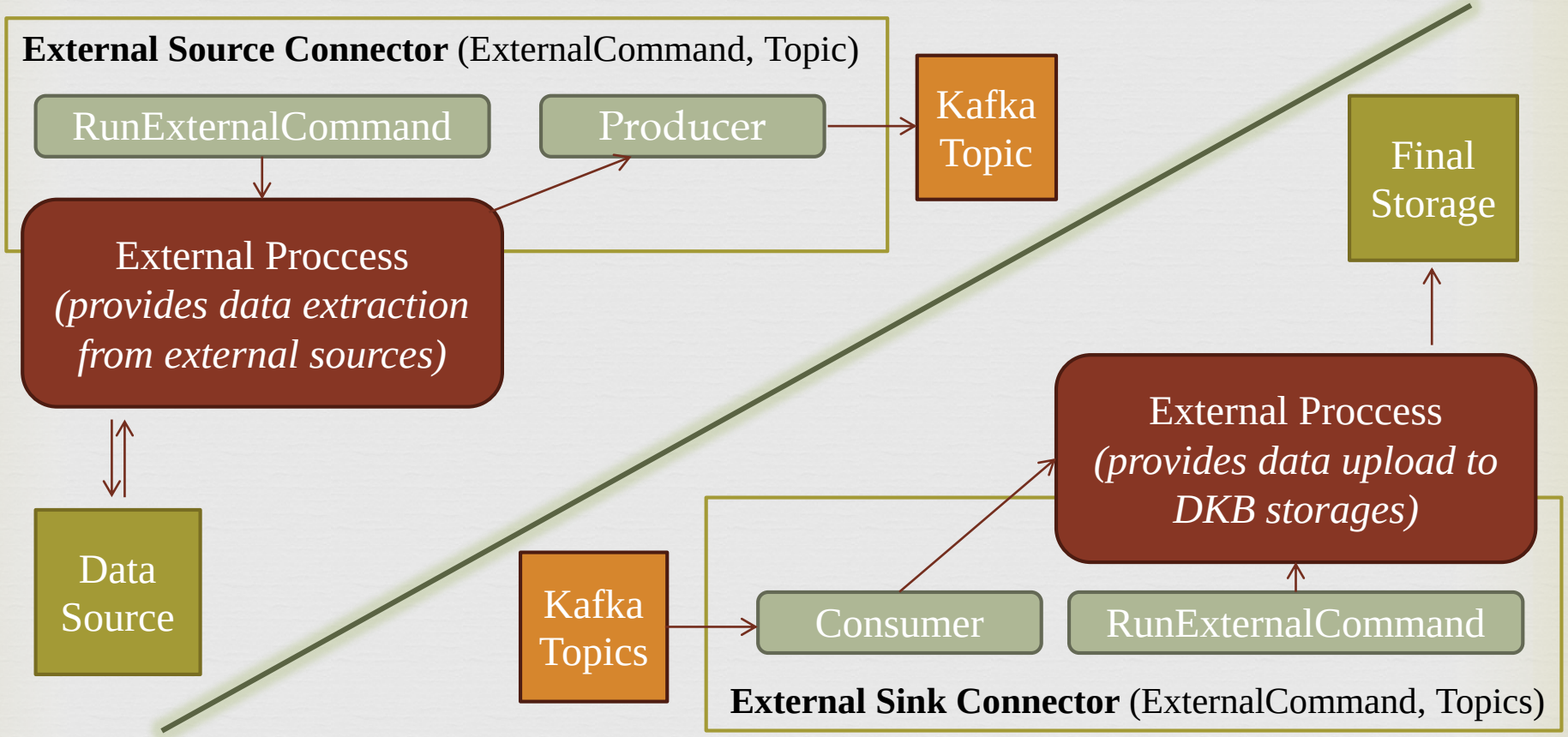
- Идеология
  - Критерии «потоковости» внешней программы
- Текущее состояние разработки
  (на 27.01.2017)

Идеология

- Есть
  - много разных модулей
  - передача данных между модулями:
    - файлами в HDFS
    - файлами по почте
  - запуск модулей в ручном режиме

- Хочется
  - чтобы оно как-то само

Идеология: Connect



Идеология: Connect

Требования к внешним программным модулям

Source

- работа в потоковом режиме [1]
- отдавать данные на STDOUT
- все информационные сообщения и/или сообщения об ошибках направлять на STDERR
- [2] генерировать данные с некоторой периодичностью (задаваемой с помощью вх. параметра)
- разбивать данные на сообщения с помощью \0

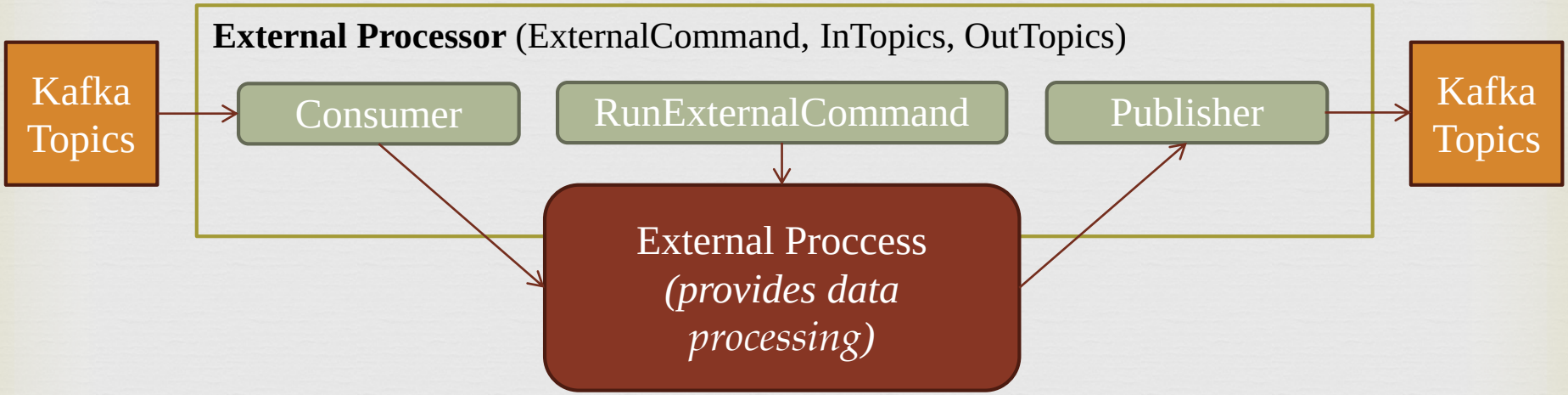
Sink

- работа в потоковом режиме
- получать данные с STDIN
- все информационные сообщения и/или сообщения об ошибках направлять на STDERR
- разбивка вх. данных на сообщения по \n или \0
- разбивка вх. данных на «батчи» по \0
- ??? сигнализировать о неудачной попытке записи

[1] Работа в потоковом режиме: получение входящего сообщения; обработка; ожидание нового входящего сообщения; и т.д.

[2] Символом ○ обозначены пункты, которые нуждаются в доработке.

Идеология: обработка данных



Идеология: обработка данных

Требования к внешним программным модулям

Processing

- работа в потоковом режиме [1]
- получать данные с STDIN
- отдавать данные на STDOUT
- все информационные сообщения и/или сообщения об ошибках направлять на STDERR
- разбивать данные на сообщения с помощью \n
- сигнализировать о конце обработки входного сообщения символом \0 (посылаемым после последнего выходного сообщения)
- [2]???

[1] Работа в потоковом режиме: получение входящего сообщения; обработка; ожидание нового входящего сообщения; и т.д.

[2] Символом ○ обозначены пункты, которые нуждаются в доработке.

Текущее состояние разработки Kafka Connect

- **runPipeConnector**
  - Kafka Connector запускается как обычное JAVA-приложение;
  - внешняя программа запускается отдельно (напр., по cron`у) ИЛИ
  - внешняя программа запускается из runPipeConnector.
    При этом внешняя программа должна:
    - понимать параметр --pipe PIPE;
    - обеспечивать регулярное выполнение обновления данных;
  - выходные данные внешней программы записываются в именованный pipe (указанный параметром --pipe или созданный отдельно и используемый внешней программой как обычный файл);
  - данные забираются из pipe`а и записываются в топики с помощью FileStreamSourceConnector.

```
java -cp $CLASSPATH ru.kiae.dkb.kafka.connect.runPipeConnector \
   -S ../010_glancePapers/list_of_papers -O glance-raw \
   -n glance-connect [-c "myExternalCommand –m s …"]
```

Текущее состояние разработки Kafka Connect

- **ExternalSinkConnector**
  - запускается с помощью утилиты Kafka connect-standalone.sh
  - внешняя программа (строка запуска) указывается параметром в конфигурационном файле:
    external.program=../099_myStage/myStage.sh –m s ...
  - входные данные поступают во внешнюю программу через STDIN
  - 1 сообщение = 1 строка;
    возможна отправка спецсимвола \0 каждые N строк (при N < 0 спецсимвол не отправляется):
    batch.size=N

```
$KAFKA_HOME/bin/connect-standalone.sh \
    $CONFIGS/connect-sinks-standalone.properties \
    $CONFIGS/virtuoso-ttl-sink.properties \
    $CONFIGS/virtuoso-sparql-sink.properties
```

Текущее состояние разработки Kafka Streams

- **runExternalProcessor**
  - запускается как JAVA-приложение;
  - строка запуска внешней программы указывается параметром командной строки;
- **runYourNameProcessor**
  - запускается как JAVA-приложение;
  - запуск внешней команды явно задан в коде.

```
java -cp $CLASSPATH ru.kiae.dkb.kafka.streams.runExternalProcessor \
  -O dataset-metadata-ttl -S dataset-metadata-csv \
  -c "../053_datasets2TTL/csv2sparql.py -m s -T"

java -cp $CLASSPATH ru/kiae/dkb/kafka/streams/csv2ttlProcessor
```

Текущее состояние разработки Управляющая утилита

- **run.sh YourStage {start|stop|restart}**
  - запускает указанный этап потоковой обработки данных
  - на данный момент реализовано только для VirtuosoSink (отправка TTL-данных и SPARQL-запросов в Virtuoso из топиков, перечисленных к конфигурационных файлах: virtuoso-ttl-sink.properties virtuoso-sparql-sink.properties)

```
./run.sh VirtuosoSink start
```

Ближайшие планы

- Доработка README & загрузка последних разработок в Subversion
- Доработка существующих JAVA-приложений:
  - работа с конфигурационными файлами
  - удаление частных реализаций (runYourNameProcessor)
- Доработка управляющей утилиты
  - создание конфигурационных файлов для всех этапов обработки данных
  - реализация запуска всех этапов разработки данных

С чего начать?

версия актуальна на 26.01.2017

- Тип этапа обработки данных
- Запуск этапа как Kafka-приложения

Тип этапа

- Этапы обработки данных бывают трёх типов:
  - Source (получение данных из внешних источников)
  - Processing
    - преобразование данных и/или
    - выполнение действий, регулируемых входными данными (например, загрузка PDF документов из CDS по списку получаемых на предыдущем этапе ссылок)
  - Sink (запись данных в конечные хранилища, такие как HDFS, Impala, Virtuoso)

Source

- Запустить **runPipeConnector**

```
runPipeConnector --out-topic TOPIC --source PIPE \
  [--command "myCommand --opt1 val1 --opt2 val2 ..."] [--name NAME]
```

  - **если** внешняя программа:
    - понимает опцию --pipe PIPE И
    - генерирует данные с некоторой периодичностью

    использовать опцию --command
  - **иначе**
    - использовать опцию --name (NAME – уникальное имя запускаемого Kafka-процесса) И
    - запустить программу, которая будет записывать данные в PIPE

- Прочитать выходные данные в соответствующем топике

```
$KAFKA_HOME/bin/kafka-console-consumer.sh \
  --topic OUT_TOPIC --zookeeper localhost:2181 --from-beginning
```

Processing

- Создать топик для исходных данных (если его ещё нет)

```
$KAFKA_HOME/bin/kafka-topics.sh --create --topic SRC_TOPIC --partitions 1 \
  --replication-factor 1 --zookeeper localhost:2181
```

- Запустить **runExternalProcessor**

```
runExternalProcessor --source-topic SRC_TOPIC --out-topic OUT_TOPIC \
  --command "myCommand --opt1 val1 --opt2 val2 ..." [--name NAME]
```

- Записать данные в исходный топик (1 строка = 1 сообщение)

```
cat mySrcData.txt | $KAFKA_HOME/bin/kafka-console-producer.sh \
  --topic SRC_TOPIC --broker localhost:9092
```

- Записать данные в исходный топик (1 строка = 1 сообщение)

```
$KAFKA_HOME/bin/kafka-console-consumer.sh \
  --topic OUT_TOPIC --zookeeper localhost:2181 --from-beginning
```

Sink

- Создать топик для исходных данных (если его ещё нет)

```
$KAFKA_HOME/bin/kafka-topics.sh --create --topic SRC_TOPIC --partitions 1 \
  --replication-factor 1 --zookeeper localhost:2181
```

- Создать конфигурационные файлы:
  - myConnect.properties
    (см. 000_kafka/config/connect-sinks-standalone.properties)
  - mySink.properties
    (см. 000_kafka/config/virtuoso-ttl-sink.properties)
- Запустить коннектор

```
$KAFKA_HOME/bin/connect-standalone.sh \
    $CONFIGS/myConnect.properties $CONFIGS/mySink.properties
```

- Записать данные в исходный топик (1 строка = 1 сообщение!)

```
cat mySrcData.txt | $KAFKA_HOME/bin/kafka-console-producer.sh \
  --topic SRC_TOPIC --broker localhost:9092
```

- Проверить в хранилище, записались ли данные

Спасибо!