Лекция 8. Spark Streaming

Основные темы

- Особенности Spark Streaming
- Операции
- Архитектура

Существующие системы

TRIDENT

Spark Streaming

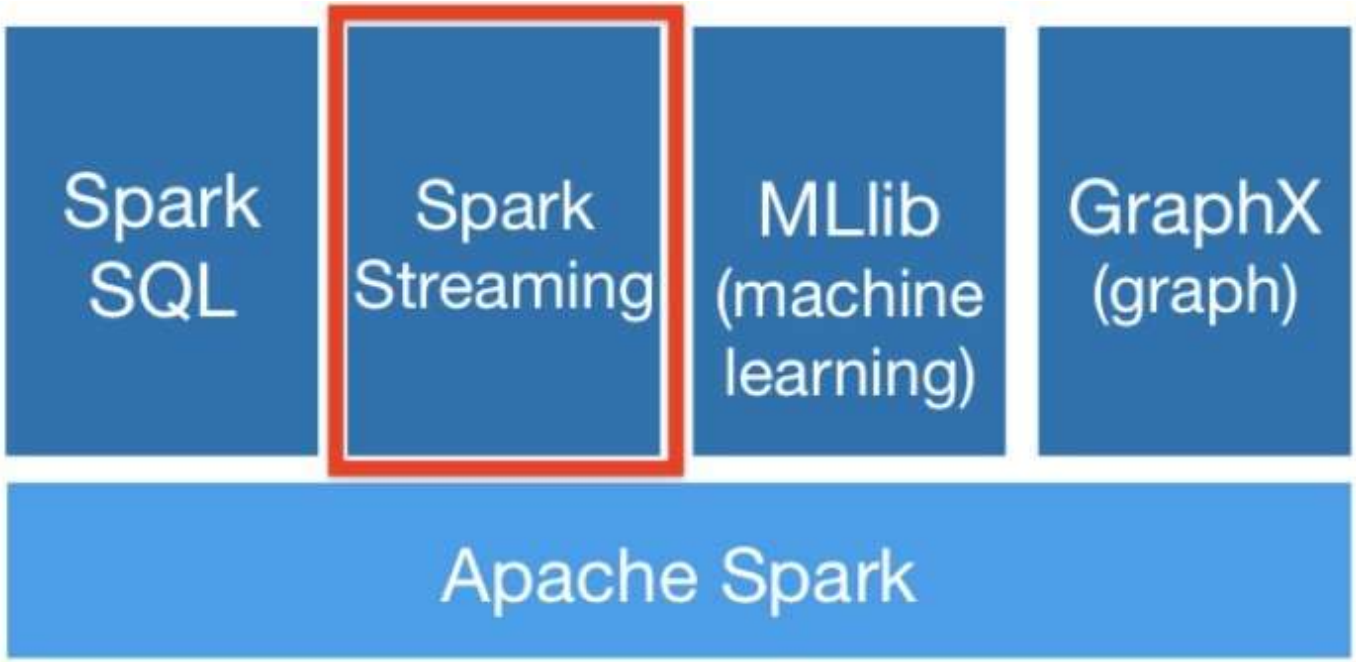
samza

Streaming Model	Native	Micro-batching	Micro-batching	Native	Native
API	Compositional		Declarative	Compositional	Declarative
Guarantees	At-least-once	Exactly-once	Exactly-once	At-least-once	Exactly-once
Fault Tolerance	Record ACKs		RDD based Checkpointing	Log-based	Checkpointing
State Management	Not build-in	Dedicated Operators	Dedicated DStream	Stateful Operators	Stateful Operators
Latency	Very Low	Medium	Medium	Low	Low
Throughput	Low	Medium	High	High	High
Maturity	High		High	Medium	Low

BigData
Processing Systems

Spark Streaming

Spark Streaming



Spark Streaming (DStreams) - RDD-based API

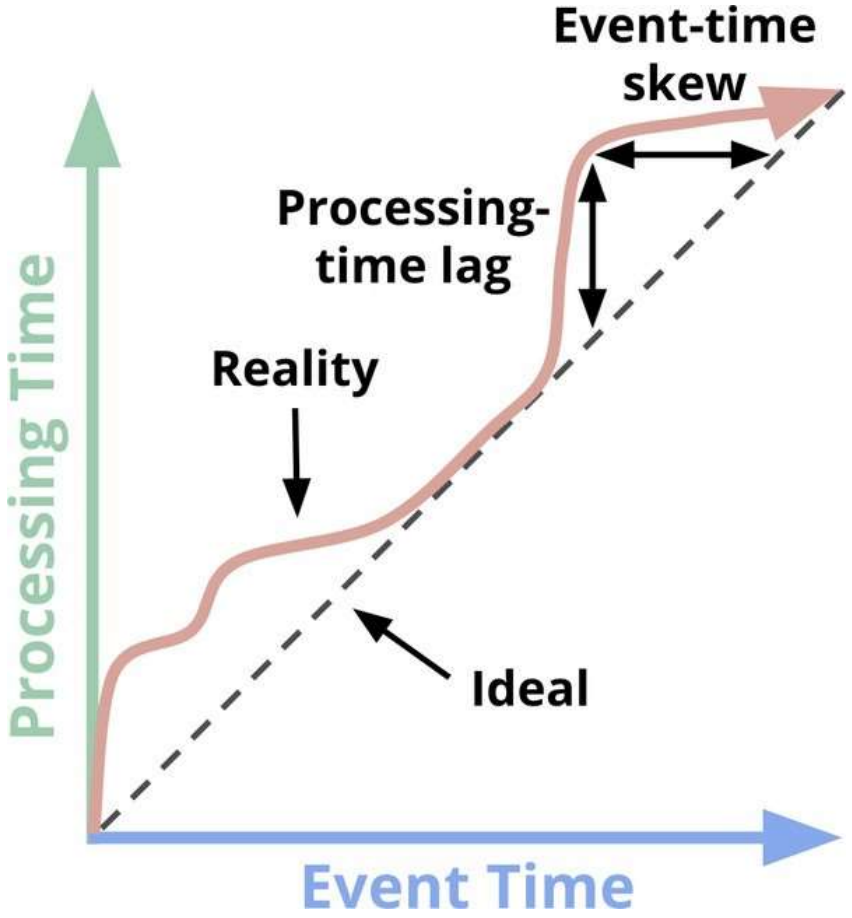
Dataflow Model

The Dataflow Model, 2015, https://research.google/pubs/pub43864

- Попытка построить логическую модель для различных ситуаций потоковой обработки без привязки к физической реализации
- Последующие реализации:
  - Google Cloud Dataflow https://cloud.google.com/dataflow
  - Apache Beam https://beam.apache.org, https://spotify.github.io/scio/

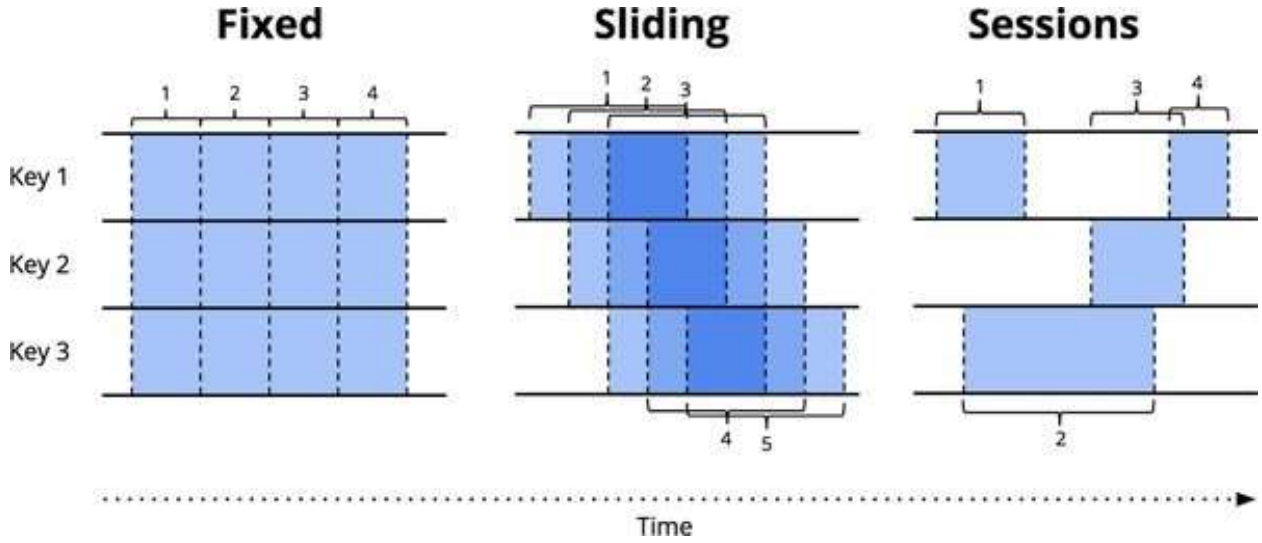
Основные понятия

- **Время события (event time)** – время, когда событие на самом деле произошло
- **Время обработки (processing time)** – время, когда событие обработано
- **Задержка обработки (processing-time lag / event-time skew)** – задержка между временем события и временем обработки
- **Ограниченные данные (bounded data)** – когда известно, что в какой-то момент данные закончатся (пакет)
- **Неограниченные данные (unbounded data)** – поток



Основные понятия

- **Фиксированное (fixed) окно** - зафиксированы частота и длина окна, и они равны
- **Скользящее (sliding) окно** - зафиксированы частота и длина окна, но частота меньше, чем длина
- **Сессии (sessions)** - динамическое окно, где окно определяется для отдельного значения ключа, а длина зависит от бизнес-логики (обычно определяется таймаутом / отсутствием активности за определённое время)



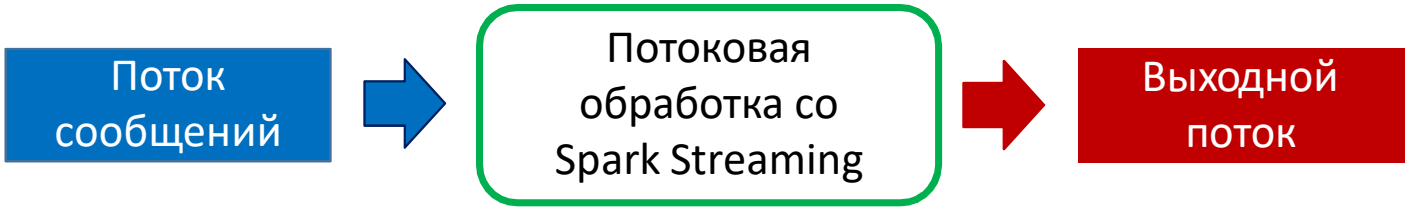
Четыре вопроса к приложению обработки потоковых данных

- **Что считаем? (What?)** - Что считает приложение? Описание трансформации над событием.
- **Где по времени события? (Where?)** - Над какими событиями на линии времени происходят трансформации? Описание используемых окон.
- **Когда по времени обработки? (When?)** - Когда по времени обработки материализовать результат трансформации? Описание водяных знаков и триггеров.
- **Как связаны? (How?)** - Как связаны последующие результаты трансформации с предыдущими? Описание агрегации.

Словарь Spark Streaming

- Driver
- Executor
- Task
- Receiver
- RDD
- DStream
- Block
- Partition
- Transformation
- Window
- Output operation

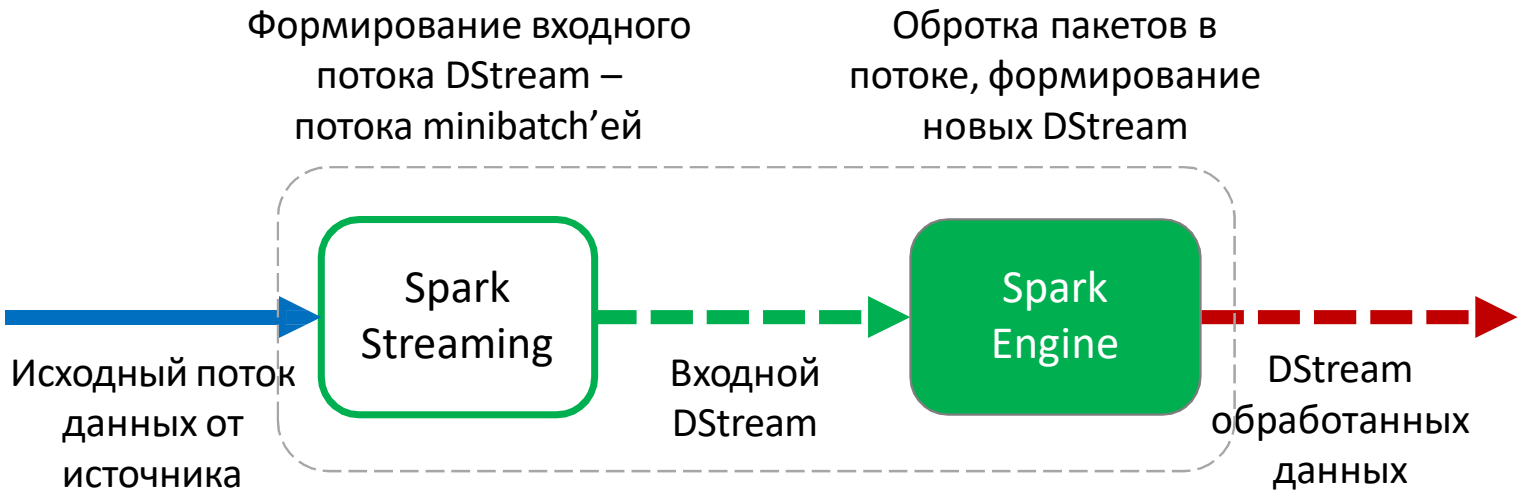
Spark Streaming



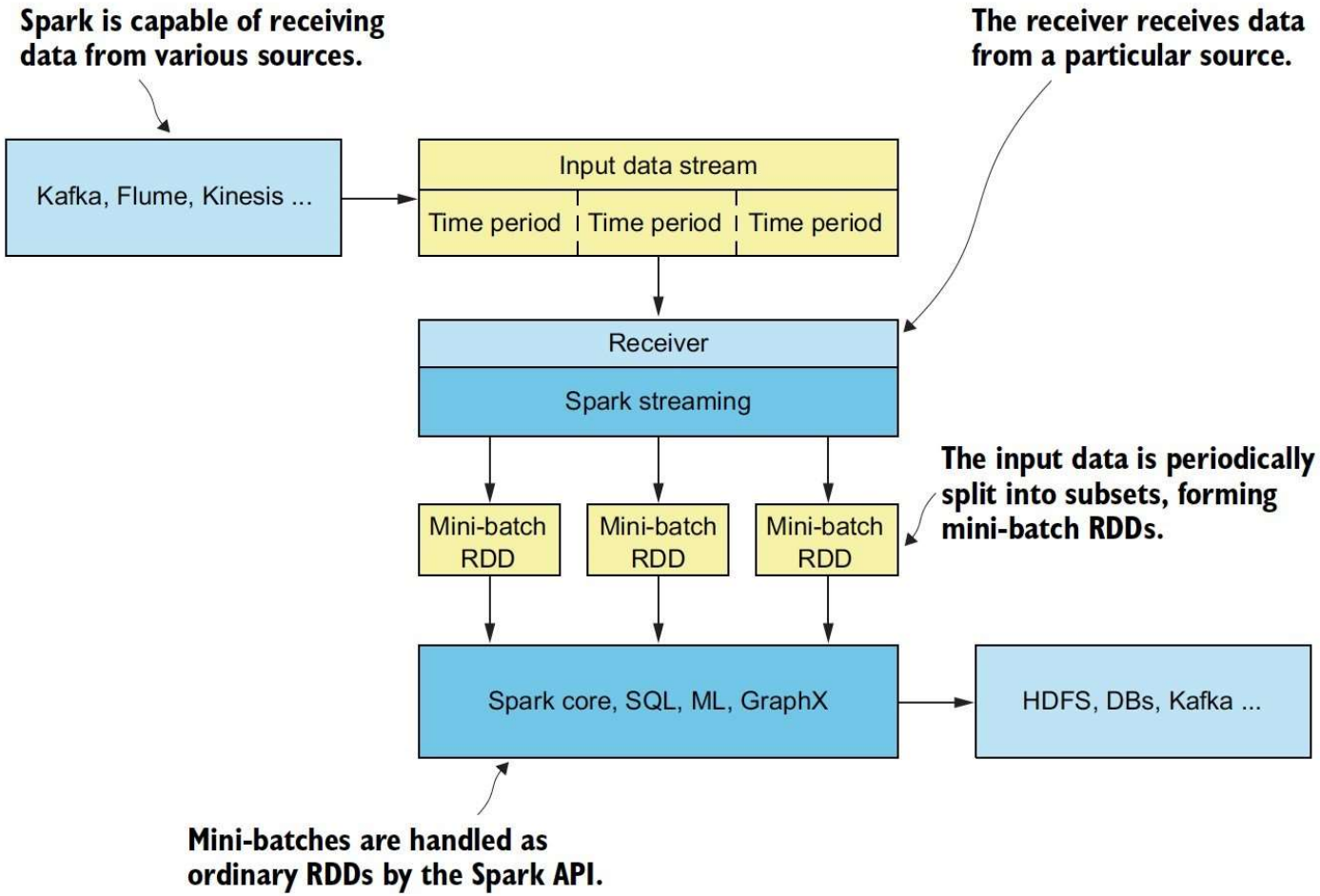
- Kafka
- Flume
- HDFS/S3
- Kinesis
- Twitter

- HDFS
- СУБД
- Dashboards

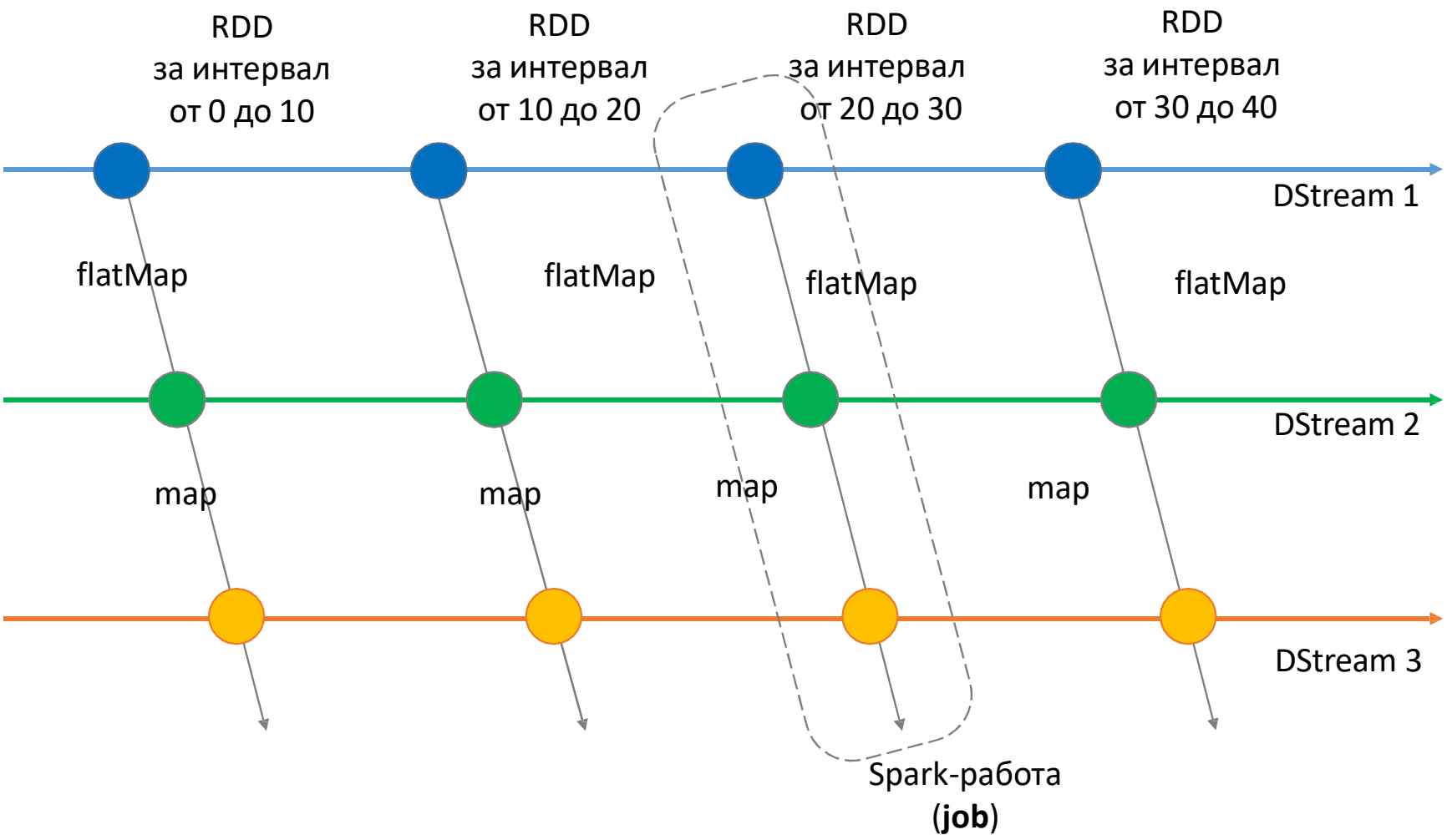
Spark Streaming. DStream



Spark Streaming. DStream



Spark Streaming. Dstream, RDD, операции



Input DStreams

- **Input DStream** – поток данных от источника
- Встроенные источники:
  - **Basic** – доступны непосредственно:
    - **File** – любая файловая система, совместимая с HDFS API (HDFS, S3, NFS, ...)
    - **Socket**
  - **Advanced** – доступны через дополнительные классы:
    - **Kafka** – http://spark.apache.org/docs/latest/streaming-kafka-0-10-integration.html
    - **Kinesis** – http://spark.apache.org/docs/latest/streaming-kinesis-integration.htm
- Пользовательские источники (недоступны в Python) – http://spark.apache.org/docs/latest/streaming-custom-receivers.html

Custom Receivers

Receiver – абстрактный класс:

- *onStart()* – запустить поток потребления данных
- *onStop()* – остановить поток
- для сохранения данных использовать метод *store()*
- для обработки исключений использовать метод *restart(exception)*

```
class CustomReceiver(host: String, port: Int)
  extends Receiver[String](StorageLevel.MEMORY_AND_DISK_2) with Logging {

  def onStart() {
    // Start the thread that receives data over a connection
    new Thread("Socket Receiver") {
      override def run() { receive() }
    }.start()
  }

  def onStop() {
    // There is nothing much to do as the thread calling receive()
    // is designed to stop by itself if isStopped() returns false
  }
```

Операции

- Трансформации

  map, *filter* и др.

- Операции вывода (output operations) – действия

  print, *saveAsTextFiles* и др.

Типы трансформаций

- **Stateless**

  Обработка пакета (batch) не зависит от предыдущего пакета

- **Stateful**

  Для получения результата обработки текущего пакета используются результаты обработки предыдущих пакетов

Трансформации

map(func)

flatMap(func)

filter(func)

repartition(numPartitions)

union(otherStream)

count()

reduce(func)

countByValue()

reduceByKey(func, [numTasks])

join(otherStream, [numTasks])

cogroup(otherStream, [numTasks])

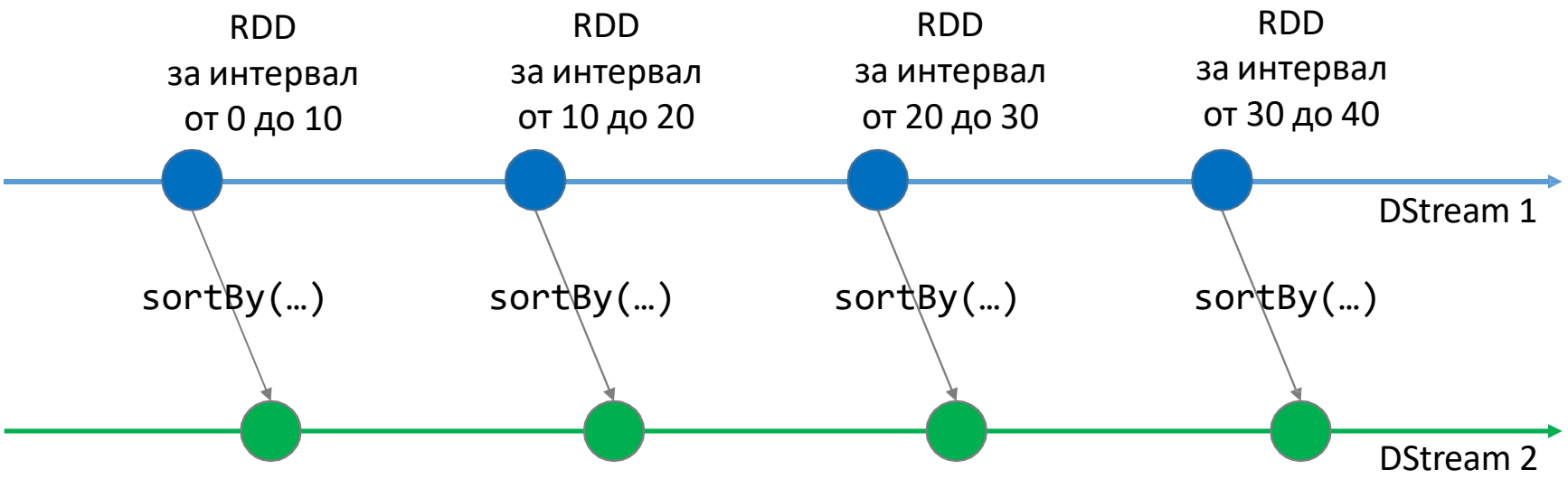
transform(func)

updateStateByKey(func)

Трансформации. Transform

- Преобразование RDD в RDD для каждого RDD потока DStream

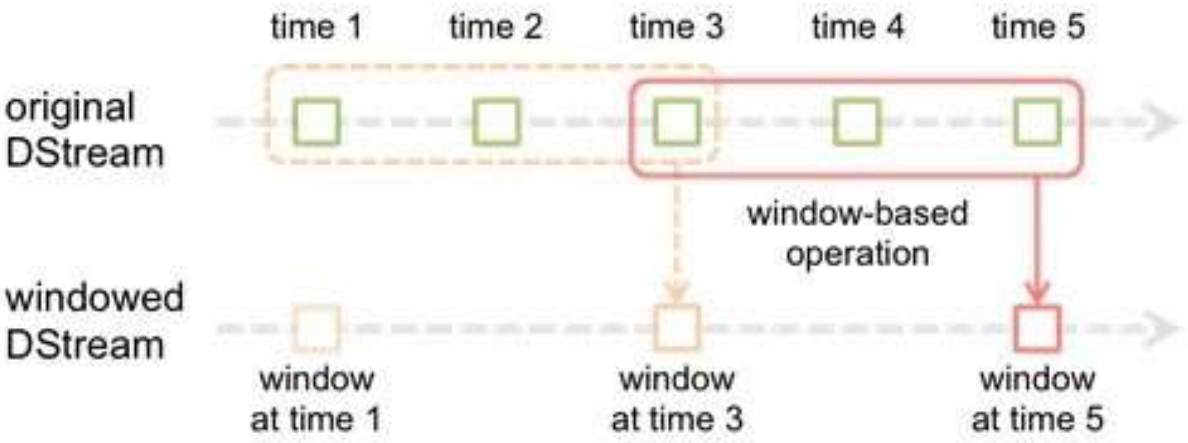
```
dstream2 = dstream1.transform(
    lambda rdd: rdd.sortBy(lambda x: x[1], ascending=False))
```



Окна в Spark Streaming

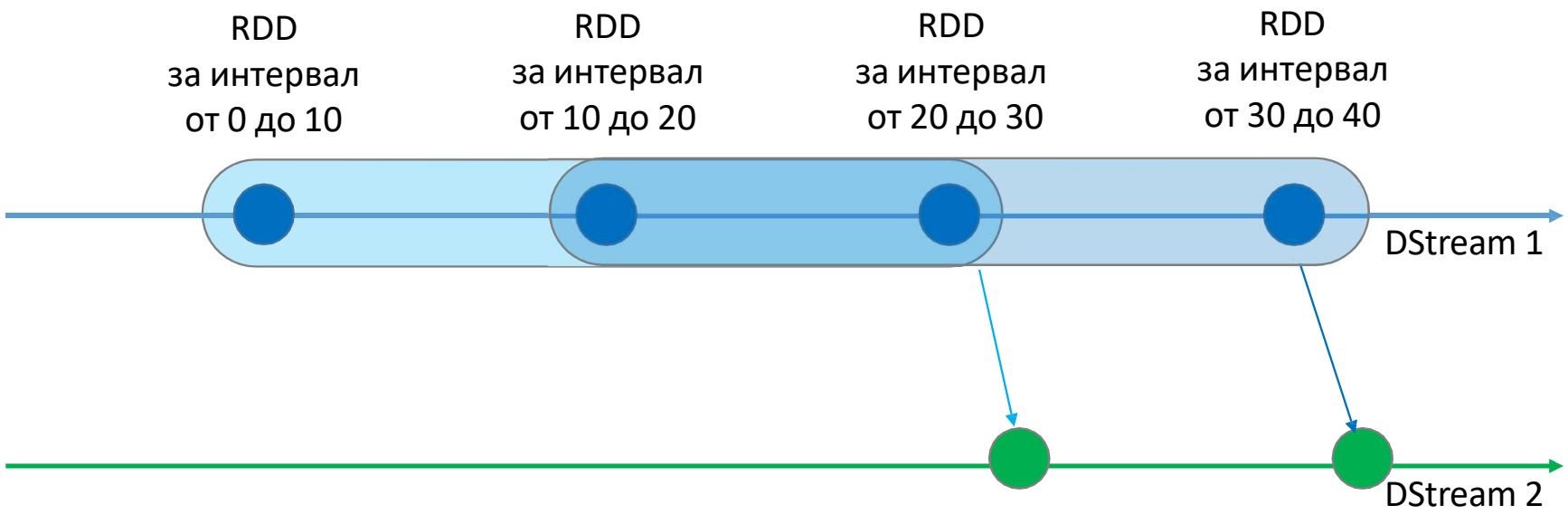
Трансформации над скользящим окном

- window length – продолжительность окна
- sliding interval – интервал, с которым применяются операции



Окна в Spark Streaming

- batchInterval=10
- windowLength=30
- slidingInterval=10



Трансформации с окном

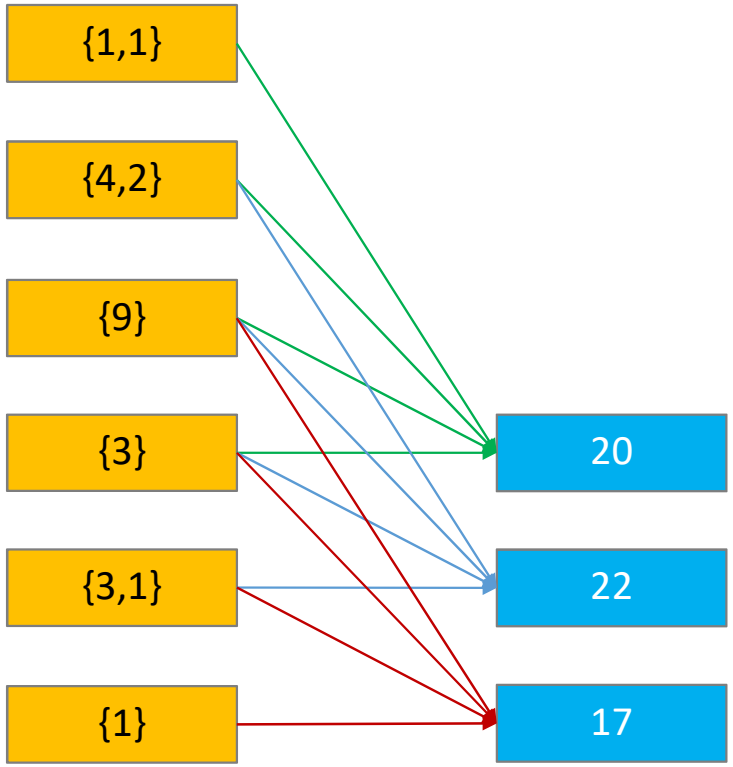
- **window**(*windowLength, slideInterval*) – поток с применением окон
- **countByWindow**(*windowLength, slideInterval*) – подсчёт по скользящему окну
- **reduceByWindow**(*func, windowLength, slideInterval*) - поток с одним элементом
  - агрегация элементов используя *func* по окну
- **reduceByKeyAndWindow**(*func, windowLength, slideInterval, [numTasks]*) - агрегация по K используя *func* по окну
- **reduceByKeyAndWindow**(*func, invFunc, windowLength, slideInterval, [numTasks]*) - значения reduce каждого окна вычисляется с использованием значений reduce предыдущего окна. *invFunc* – “обратная редукционная функция” к func
- **countByValueAndWindow**(*windowLength, slideInterval, [numTasks]*) - поток пар (K, Long) - кол-во K в RDD по окну

Трансформации reduceByWindow. Вариант 1

batchInterval=10

windowLength=40

slidingInterval=10

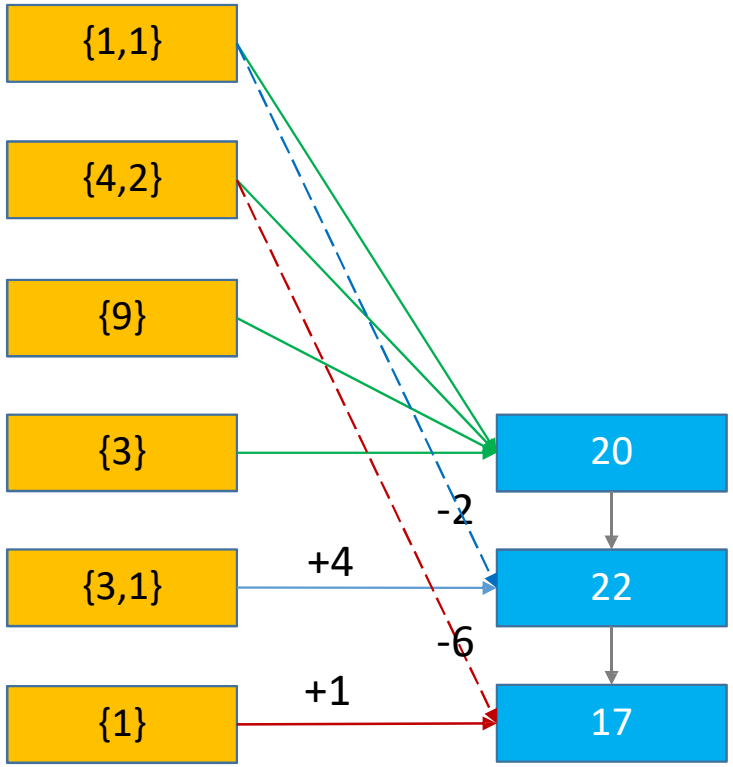


Трансформации reduceByWindow. Вариант 2

batchInterval=10

windowLength=40

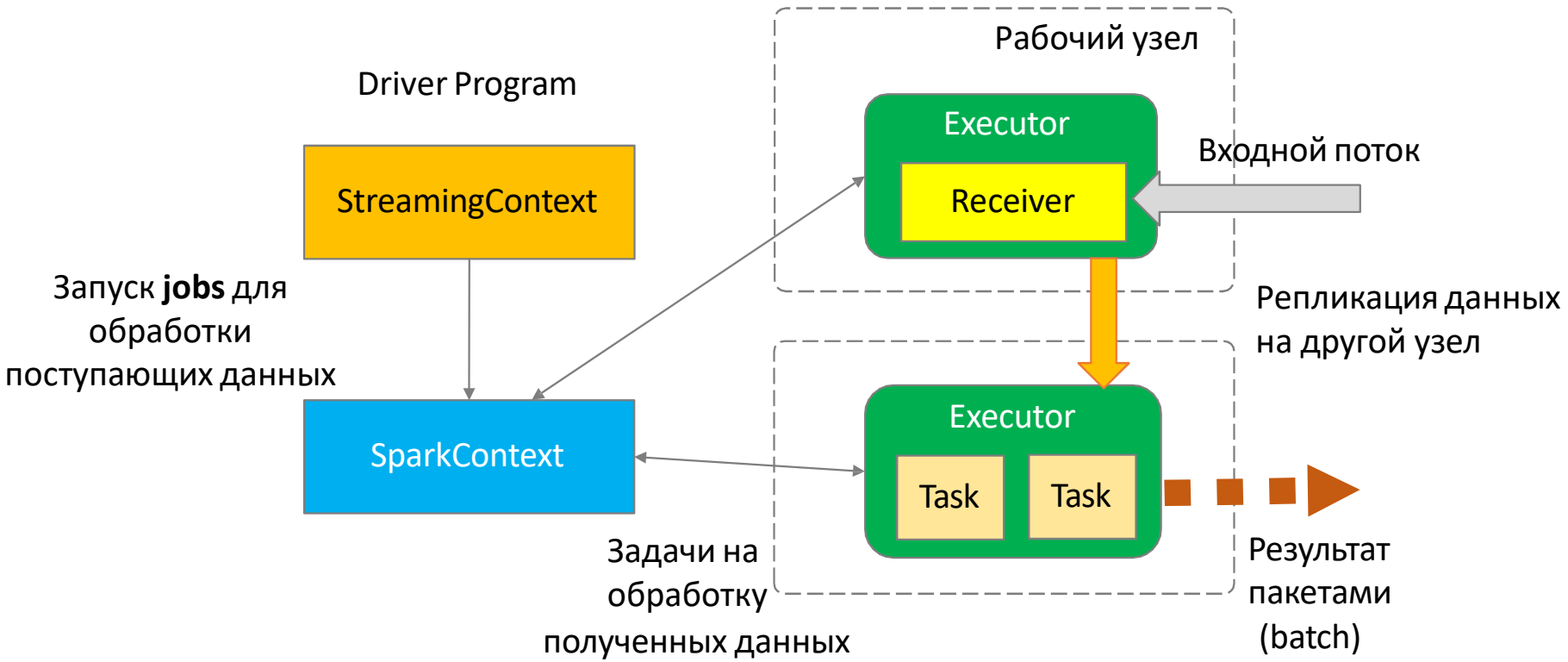
slidingInterval=10



Действия. Output Operation

- **print**() - печатает первые 10 элементов (**pprint**() в Python)
- **saveAsTextFiles**(*prefix*, [*suffix*]) - сохраняет DStream в текстовые файлы "*prefix-TIME_IN_MS[.suffix]*"
- **saveAsObjectFile**(*prefix*, [*suffix*]) - сохраняет DStream в файлы "*prefix- TIME_IN_MS[.suffix]*" как SequenceFiles (Недоступен в Python)
- **saveAsHadoopFiles**(*prefix*, [*suffix*]) - сохраняет DStream в файлы "*prefix-TIME_IN_MS[.suffix]*" как файл Hadoop (Недоступен в Python)
- **foreachRDD**(*func*) - универсальный оператор вывода, *func* применяется к набору данных

Архитектура Spark Streaming



Особенности

- **DStream** соответствует одному **receiver**
- Для получения нескольких параллельных потоков (**DStream'**ов) необходимо несколько **receiver'**ов
- **Receiver** запускается на **executor'**е как задача
- Для работы **receiver'**а требуется одно ядро
- **Receiver'**ы назначаются **executor'**ам по Round-Robin манере

Особенности

- **Receiver** создает **блоки** данных по приему потока данных от источника
- Новый **блок** формируется в течение периода **blockInterval**
- **RDD** создается на **driver'**е для блоков сформированных в течение периода **batchInterval**
- В течение **batchInterval** создается **N** блоков $N = \text{batchInterval} / \text{TblockInterval}$
- **Блоки** сгенерированные в течение **batchInterval** является **partition'**ами **RDD**
- **batchInterval** обычно от 0.5 до нескольких секунд

Особенности

- **BlockManager executor’**а отвечает за распределение **блоков** на другие **executor’**ы
- По умолчанию полученные данные копируются на два узла (две реплики)
- **Network Input Tracker** на **driver** информируется о расположении **блоков** для дальнейшей обработки
- Каждая **partition** соответствует задаче (**task**)
- Мар задачи (**task**) над **блоками** обрабатываются на **executor’**ах (на том, который получил блок от **receiver’**а, и на том, где реплика блока)

Особенности

- **JobScheduler** на **driver’**е планирует обработку RDD в виде работы (**job**)
- В каждый момент времени только одно **работа** активна
- Если одна **работа** выполняется, то другая – в очереди
- Если два потока **DStream**, то будет два RDD и две работы (**job**), которые будут запланированы последовательно (одна после другой)
- Чтобы избавиться от нескольких последовательных одинаковых **работ** для множества **DStream**, можно объединить потоки при этом это никак не повлияет на **partition’**ы RDD
- Если обработка входных данных (mini-batch) занимает больше времени чем **batchInterval**, то данные будут накапливаться на **executor’**ах, что может привести к переполнению и исключению

Отказоустойчивость

- Отказ **на рабочем узле**

  Все данные в памяти будут потеряны. Если на узле был receiver, то полученные данные в буфере также будут потеряны

- Отказ **узла driver'а**

  Spark Streaming приложение откажет, SparkContext будет утерян, и все executor'ы потеряют свои данные, хранящиеся в памяти (в оперативной – in-memory)

BigData
Processing Systems

Отказоустойчивость. Отказ при получении данных

- Данные получены и **сформированы реплики** (две по умолчанию)

  В случае отказа узла будет копия на другом

- Данные получены, но **без реплик**

  Повторный запрос к источнику

Checkpoint

- **Checkpointing** в Spark Streaming – процесс периодического сохранения текущего состояния в надежном хранилище (например, HDFS)

- **Метаданные.** *Metadata checkpointing* – сохранение информации, определяющей потоковое приложение, в отказоустойчивое хранилище типа HDFS

  Configuration - конфигурация приложения

  DStream operations - набор операций Dstream

  Incomplete batches - пакеты, обработка которых ещё не завершена

- **Данные обработки** (RDD). *Data checkpointing* – сохранение сгенерированных RDD в надёжное хранилище

Checkpoint. Метаданные

- **Конфигурация**

  Конфигурация, которая использовалась для создания приложения

- **DSteam операции**

  Множество **DStream** операций приложения

  Незавершенные пакеты (**batch**)

  Batch’и, которые в очереди работ (**job**)

Checkpoint. Данные

- Сохраняет созданные RDD в надежном хранилище
- Это необходимо для некоторых **stateful** трансформаций, которые комбинируют данные от нескольких **batch'**ей
- В таких трансформациях сгенерированная RDD зависит от RDD предыдущего пакета (**batch**)
- В свою очередь это ведет к увеличению последовательных зависимостей
- Чтобы избежать увеличения времени восстановления, промежуточные RDD периодически записываются в надежное хранилище (например, HDFS)
- Checkpoint каждые 5-10 batch'ей (рекомендация)

Гарантии

- Гарантии **получения** входных данных
- Гарантии в процессе **обработки** данных
- Гарантии **передачи** выходных данных (например, в систему хранения данных - СУБД, HDFS)

Гарантии получения входных данных

- Различные источники входных данных дают разные гарантии
- **Надежный receiver**:

  подтверждает надёжный источник только после репликации полученных данных. Если отказал receiver, то источник не получить подтверждение, и после перезапуска receiver’а, источник повторно отправит данные

- **Ненадежный receiver**:

  receiver не отправляет подтверждение и поэтому может потерять данные, когда выйдет из строя executor или driver

Гарантии в процессе обработки данных

- Все полученные данные обработаются гарантированно один раз (exactly once).
- Если произошел отказ, то до тех пор пока доступны исходные данные, может быть достигнут одинаковый конечный результат обработки (за счет отказоустойчивости при выполнении RDD)

Гарантии в процессе обработки данных

- Выходные операции по умолчанию обеспечивают семантику «at-least once» (по крайней мере один раз, но может и больше).
- Семантика зависит:
  - от типа выходной операции (идемпотентная или нет)
  - от семантики системы хранения (есть механизм транзакций или нет)

Источники

Learning Spark by H. Karau, A. Konwinski, P. Wendell, and M. Zaharia (book)

Spark Streaming Programming Guide (doc)