

СИСТЕМЫ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ

Разработка приложений



К.Т.Н.
Папулин Сергей Юрьевич
papulin_bmstu@mail.ru

Лекция 8. Поточковая обработка и обработка графов



- Особенности Spark Streaming
- Операции
- Архитектура

Существующие системы



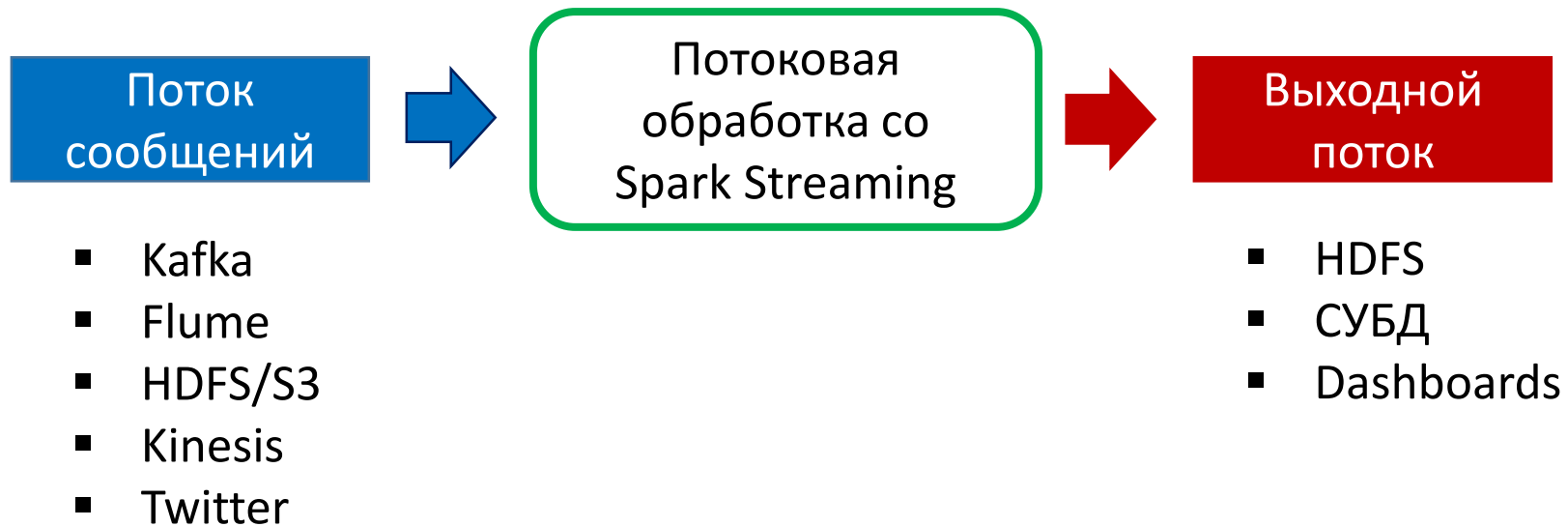
Samza



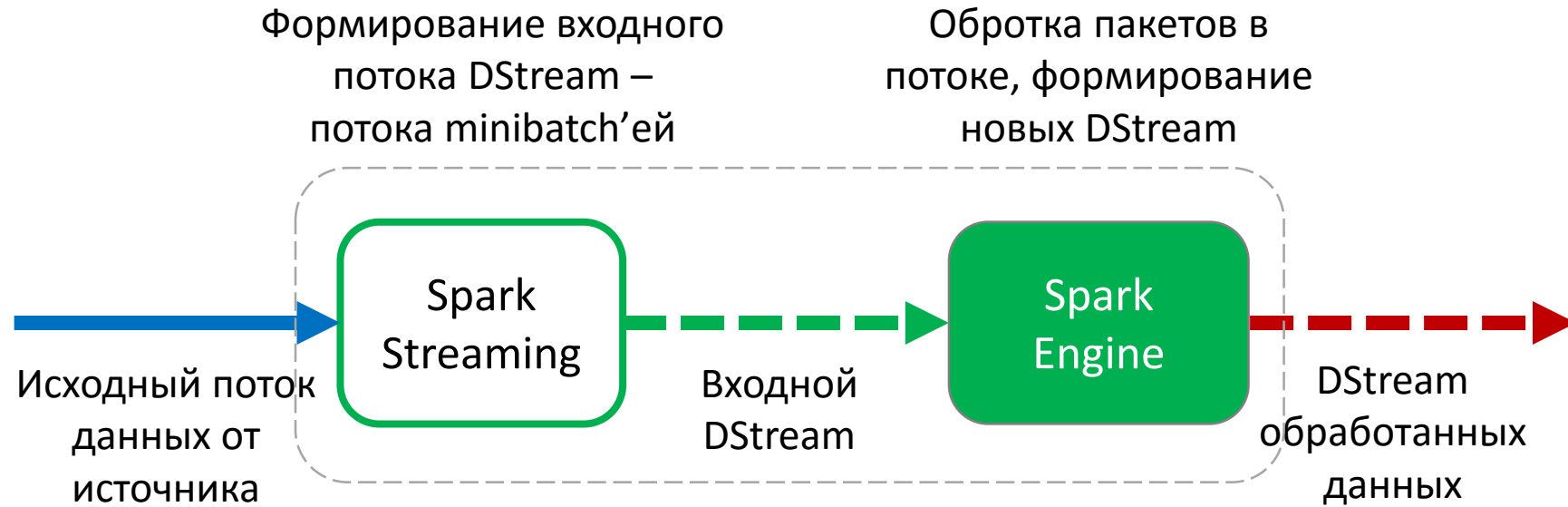
Streaming Model	Native		Micro-batching		Native	Native
API	Compositional		Declarative		Compositional	Declarative
Guarantees	At-least-once	Exactly-once	Exactly-once		At-least-once	Exactly-once
Fault Tolerance	Record ACKs		RDD based Checkpointing		Log-based	Checkpointing
State Management	Not build-in	Dedicated 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

Spark Streaming

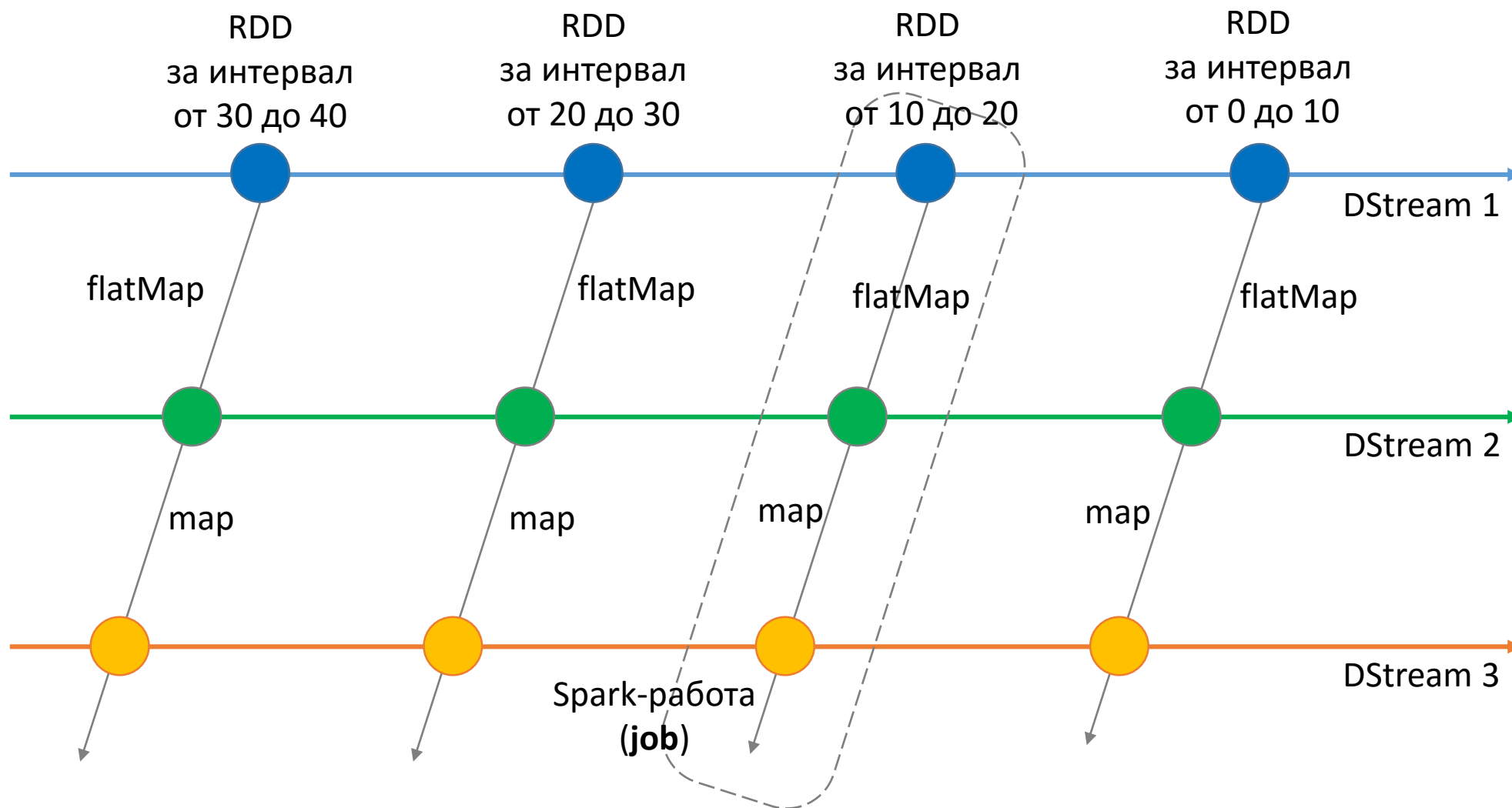
- | | | |
|------------|-----------|--------------------|
| ➤ Driver | ➤ RDD | ➤ Partition |
| ➤ Executor | ➤ DStream | ➤ Transformation |
| ➤ Task | ➤ Block | ➤ Window |
| ➤ Receiver | | ➤ Output operation |



Spark Streaming. DStream



Spark Streaming. Dstream, RDD, операции





Трансформации

map, filter и др.



Операции вывода (output operations) – действия

print, saveAsTextFiles и др.

➤ **Stateless**

Обработка пакета (batch) не зависит от предыдущего пакета

➤ **Stateful**

Для получения результата обработки текущего пакета используются результаты обработки предыдущих пакетов

map(func)

flatMap(func)

filter(func)

repartition(numPartitions)

union(otherStream)

count()

reduce(func)

countByValue()

reduceByKey(func, [numTasks])

join(otherStream, [numTasks])

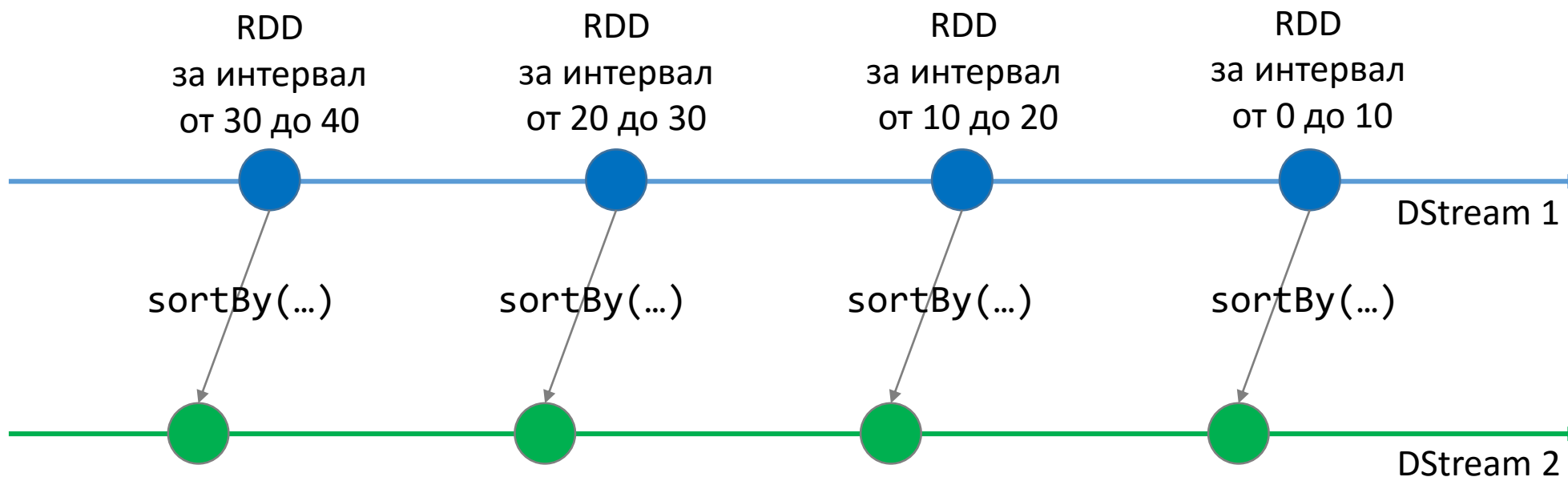
cogroup(otherStream, [numTasks])

transform(func)

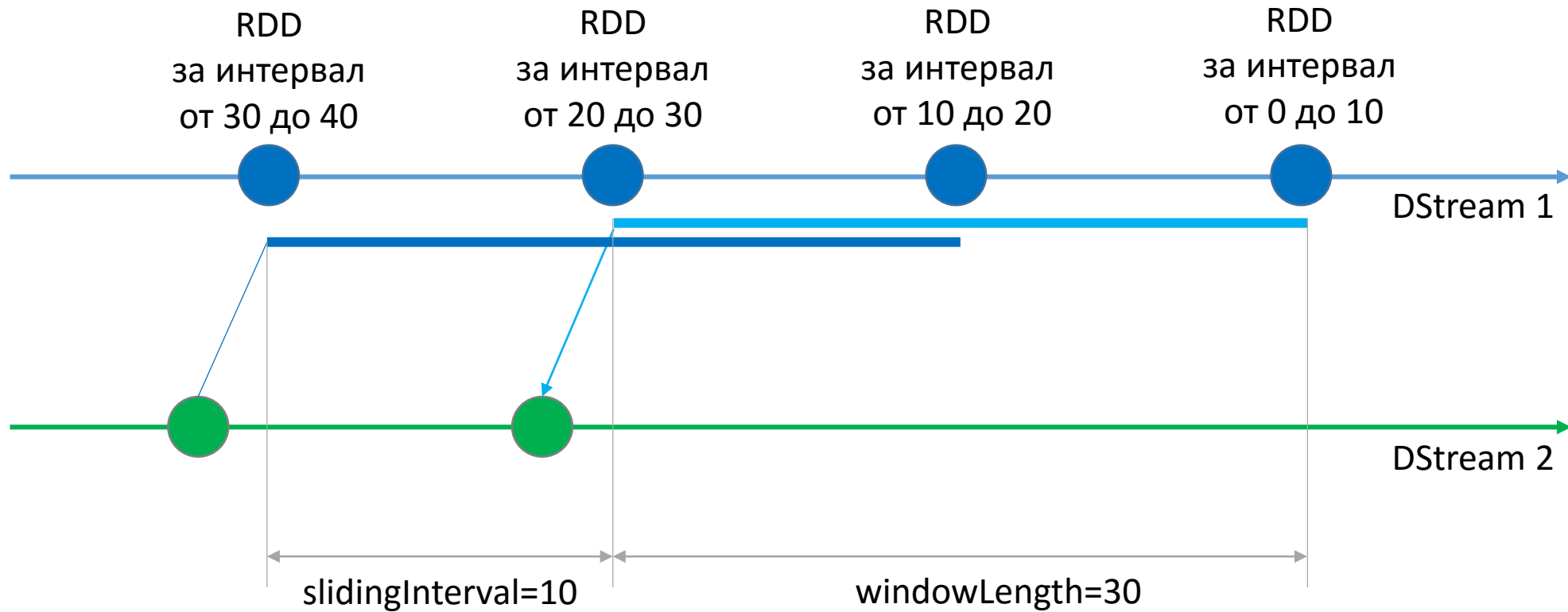
updateStateByKey(func)

➤ 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



window(windowLength, slideInterval)

countByWindow(windowLength, slideInterval)

reduceByWindow(func, windowLength, slideInterval)

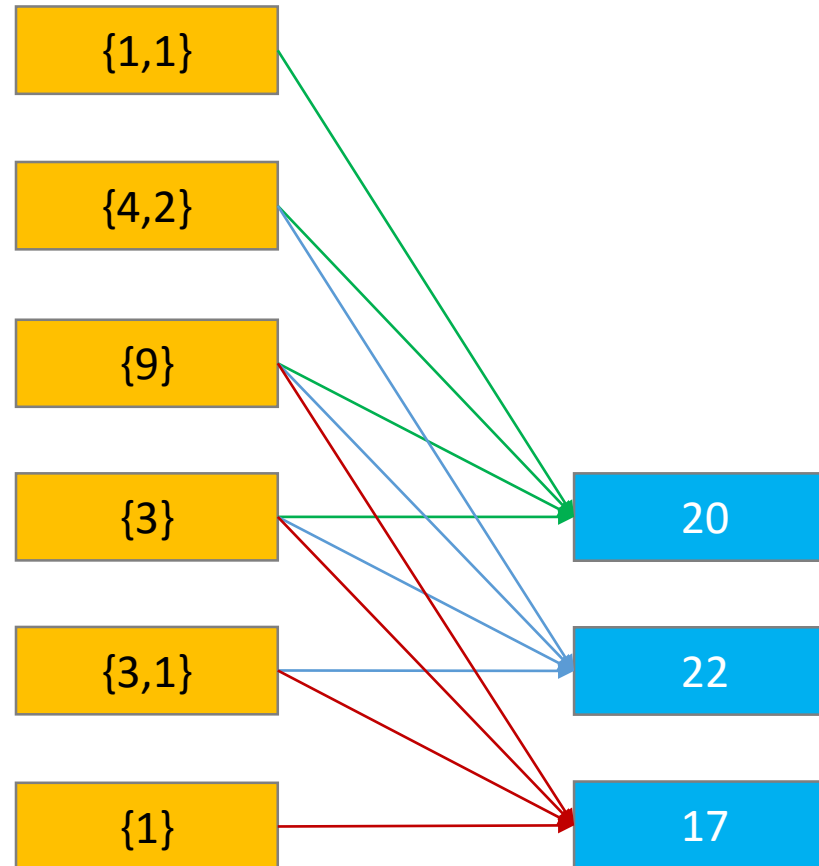
reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])

reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])

countByValueAndWindow(windowLength, slideInterval, [numTasks])

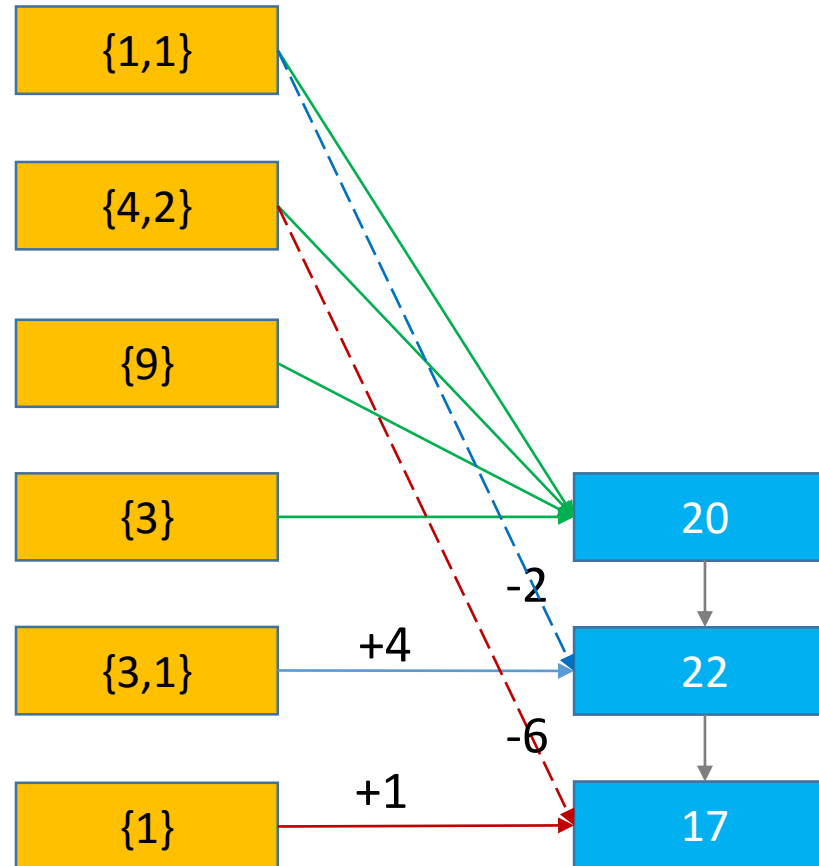
Трансформации reduceByWindow. Вариант 1

batchInterval=10
windowLength=30
slidingInterval=10



Трансформации reduceByWindow. Вариант 2

batchInterval=10
windowLength=30
slidingInterval=10



print()

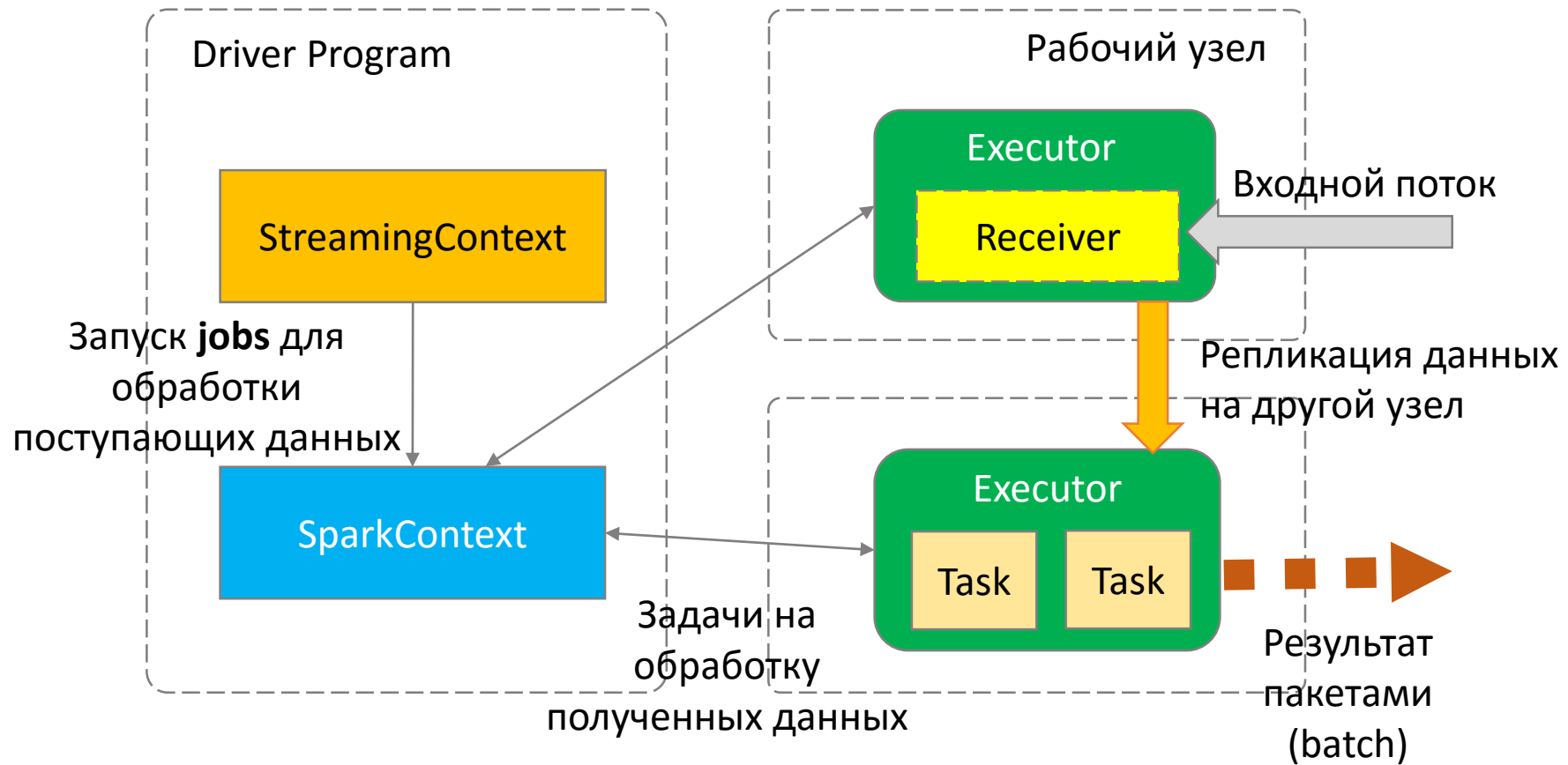
saveAsTextFiles(prefix, [suffix])

saveAsObjectFiles(prefix, [suffix])

saveAsHadoopFiles(prefix, [suffix])

foreachRDD(func)

Архитектура Spark Streaming



- **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

- **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 $N = \text{batchInterval} / \text{blockInterval}$
- **Блоки** сгенерированные в течение **batchInterval** являются **partition**'ами **RDD**
- **batchInterval** обычно от 0.5 до нескольких секунд

- **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**)
- Map задачи (**task**) над **блоками** обрабатываются на **executor**'ах (на том, который получил блок от **receiver**'а, и на том, где реплика блока)

- **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

➤ 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'a

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

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

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

➤ **Метаданные**

для восстановления после отказа driver'а

➤ **Данные обработки (RDD)**

для stateful трансформаций

➤ Конфигурация

Конфигурация, которая использовалась для создания приложения

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

➤ Различные источники входных данных дают разные гарантии

➤ **Надежный receiver:**

подтверждает надёжный источник только после репликации полученных данных. Если отказал receiver, то источник не получит 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

- 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)

- 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)

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

[Spark Streaming Programming Guide](#) (doc)