法律声明

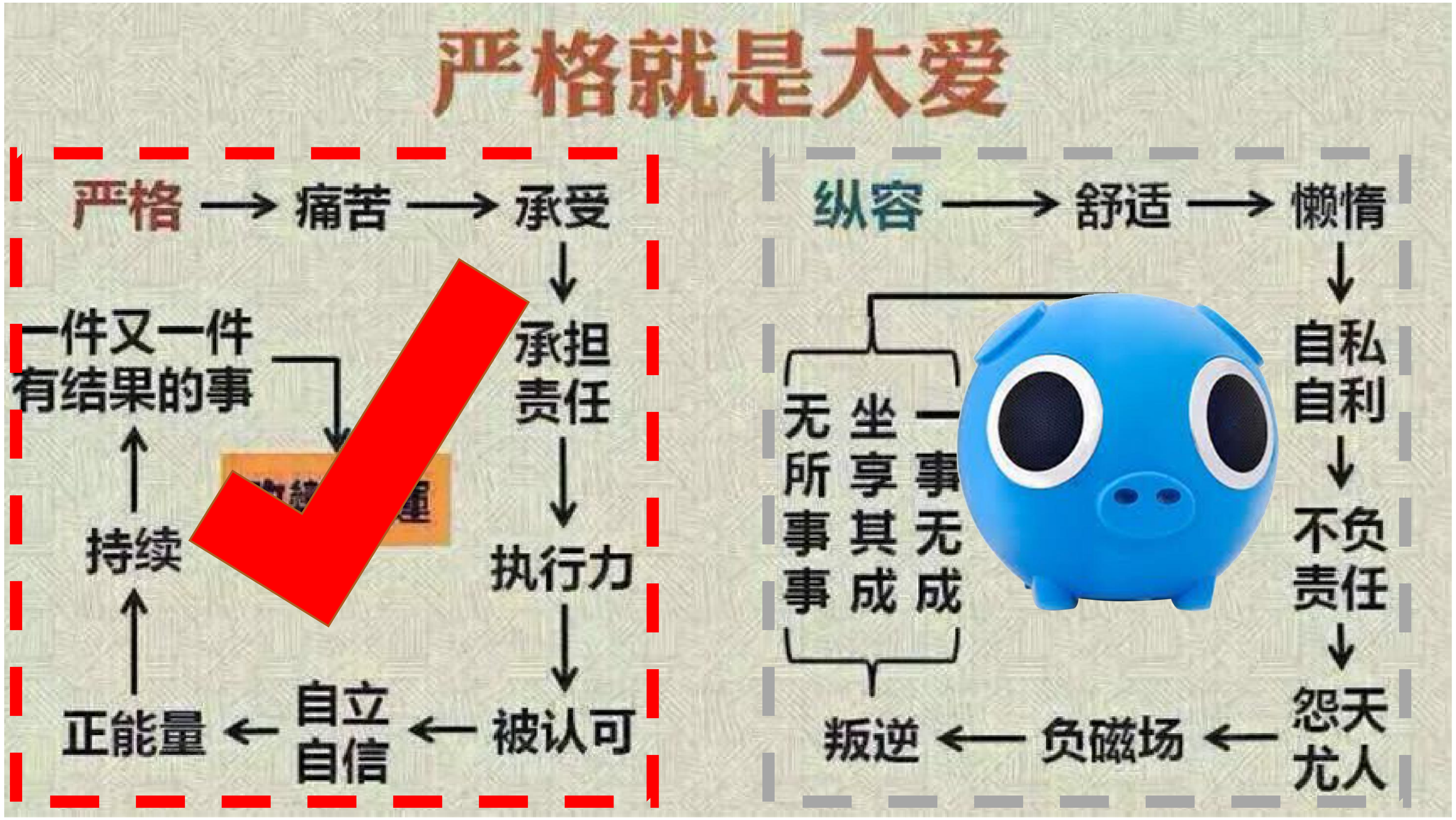
大数据内存计算框架Spark

Spark Streaming

主讲人：Gerry

上海育创网络科技有限公司

严格是大爱


严格就是大爱
严格 → 痛苦 → 承受
承担责任
执行力
被认可
自立自信
正能量
持续
一件又一件有结果的事
纵容 → 舒适 → 懒惰
自私自利
不负责任
怨天尤人
负磁场
叛逆
无所事事
坐享其成
一事无成

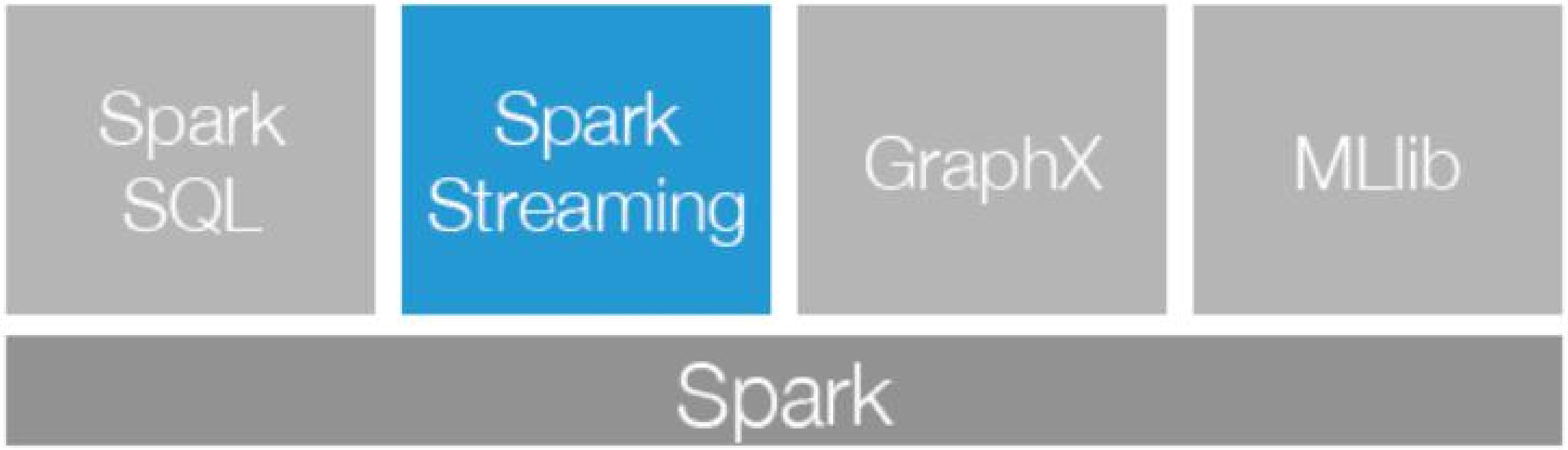
Streaming

- Streaming：是一种**数据传送技术**，它**把客户机收到的数据变成一个稳定连续的流，源源不断地送出**，使用户听到的声音或看到的图象十分平稳，而且用户在整个文件送完之前就可以开始在屏幕上浏览文件。
- Streaming Compute
  - **Apache Storm**
  - **Spark Streaming**
  - **Apache Samza**
- 上述三种实时计算系统都是**开源的分布式系统，具有低延迟、可扩展和容错性诸多优点**，它们的共同特色在于：**允许你在运行数据流代码时，将任务分配到一系列具有容错能力的计算机上并行运行**。此外，它们都提供了简单的API来简化底层实现的复杂程度。

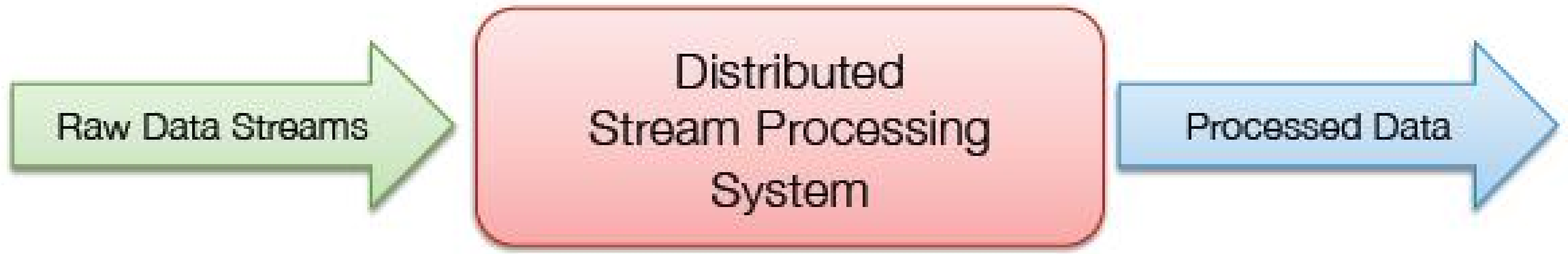
http://www.csdn.net/article/2015-03-09/2824135

What is Spark Streaming?

Spark Streaming is an extension of the Spark core API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

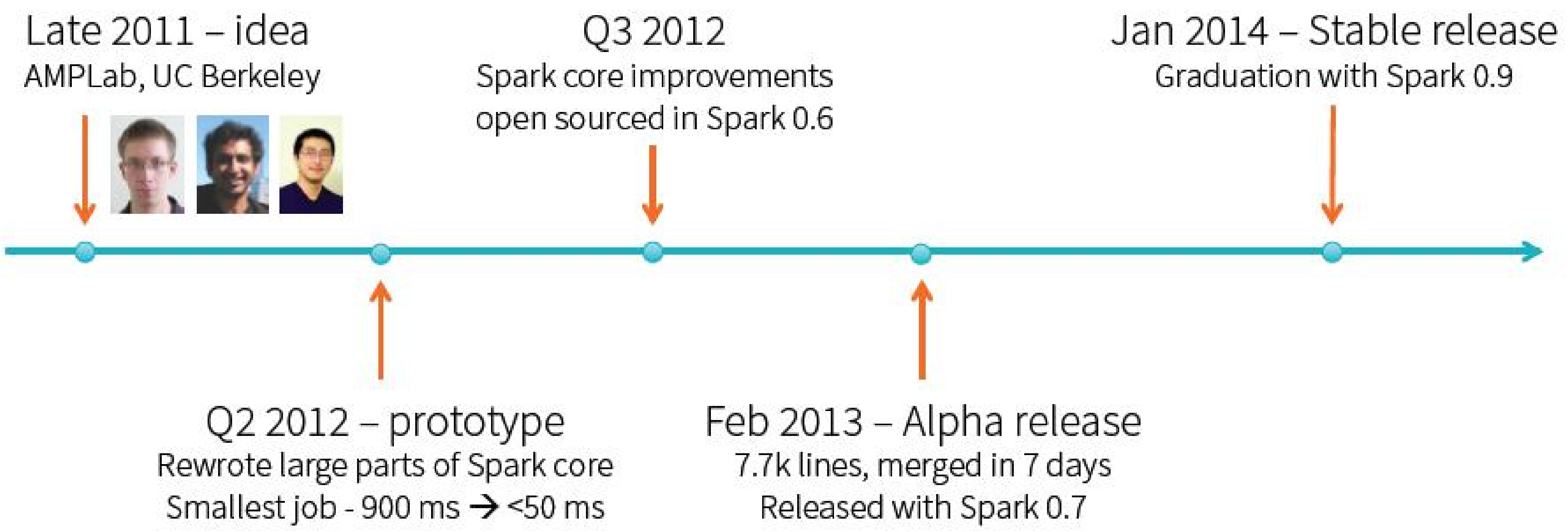


What is Spark Streaming?



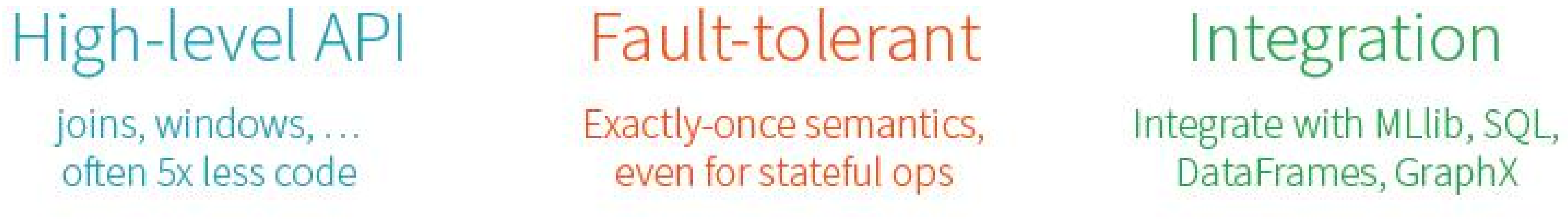
- Scales to hundreds of nodes;
- Achieves low latency;
- Efficiently recover from failures;
- Integrates with batch and interactive processing;

Streaming History


Late 2011 – idea
AMPLab, UC Berkeley
Q3 2012
Spark core improvements open sourced in Spark 0.6
Jan 2014 – Stable release
Graduation with Spark 0.9
Q2 2012 – prototype
Rewrote large parts of Spark core
Smallest job - 900 ms → <50 ms
Feb 2013 – Alpha release
7.7k lines, merged in 7 days
Released with Spark 0.7

What is Spark Streaming?

Scalable, fault-tolerant stream processing system.



High-level API

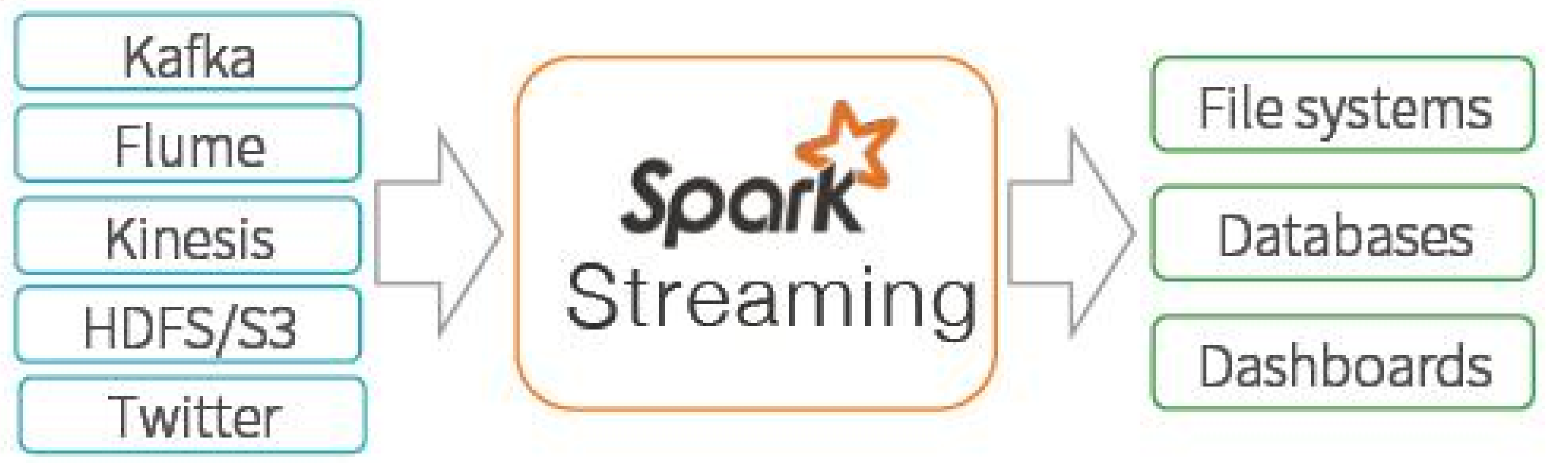
joins, windows, ...
often 5x less code

Fault-tolerant

Exactly-once semantics,
even for stateful ops

Integration

Integrate with MLlib, SQL,
DataFrames, GraphX



A Quick Example

```
$ ./bin/run-example streaming.NetworkWordCount localhost 9999
```

Then, any lines typed in the terminal running the netcat server will be counted and printed on screen every second. It will look something like the following.

Scala Java Python

```
# TERMINAL 1:
# Running Netcat

$ nc -lk 9999

hello world



...
```

```
# TERMINAL 2: RUNNING NetworkWordCount

$ ./bin/run-example streaming.NetworkWordCount localhost 9999
...
-------------------------------------------
Time: 1357008430000 ms
-------------------------------------------
(hello,1)
(world,1)
```

Streaming Word Count

```
val conf = new SparkConf().setAppName("NetworkWordCount")

val ssc = new StreamingContext(conf, Seconds(1))

val lines = ssc.socketTextStream("localhost", 9999)

val words = lines.flatMap(_.split(" "))

val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)

wordCounts.print()

ssc.start()

ssc.awaitTermination()
```

create DStream from data over socket

split lines into words

count the words

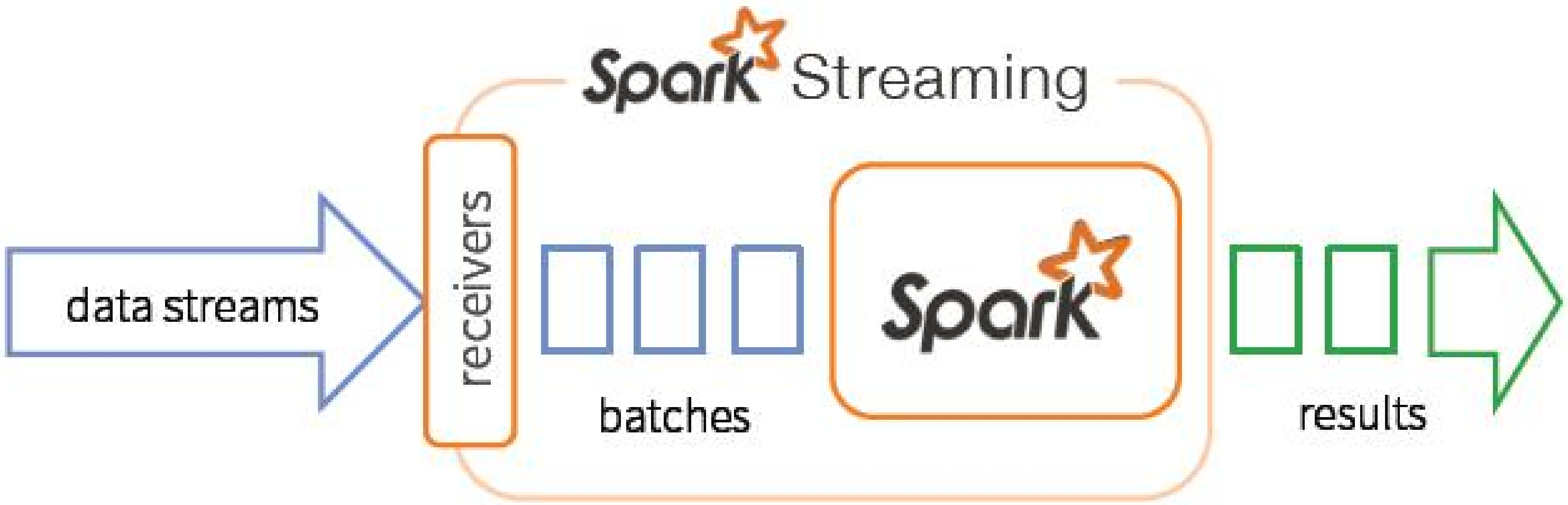
nc -lk 9999

How does it work?

Data streams are chopped up into batches;

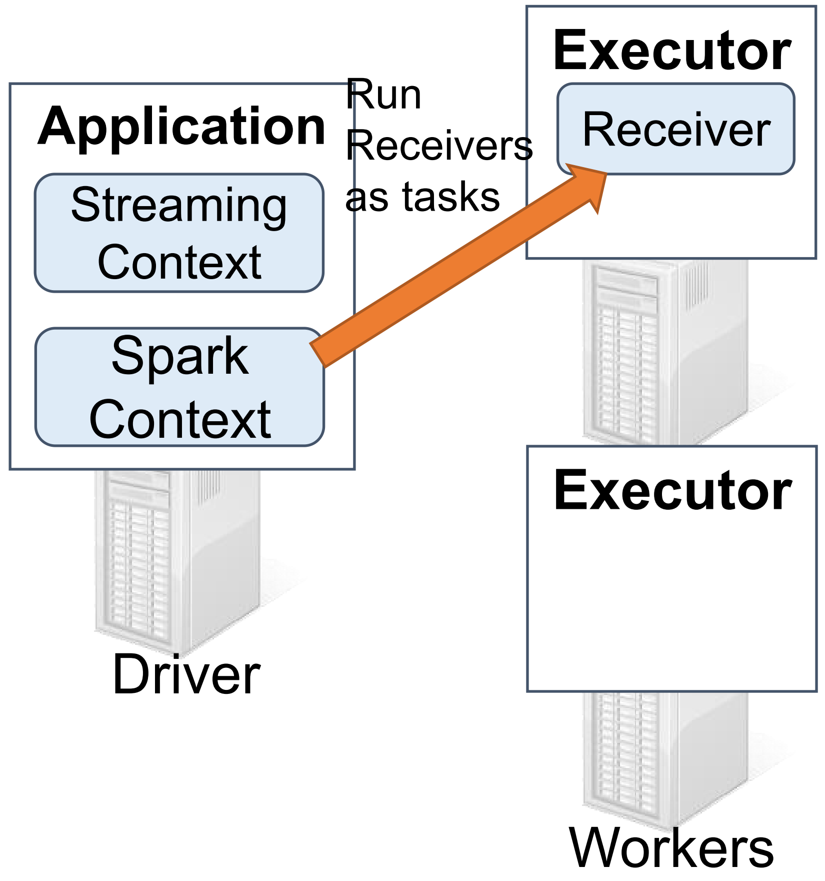
Each batch is processed in Spark;

Results pushed out in batches;



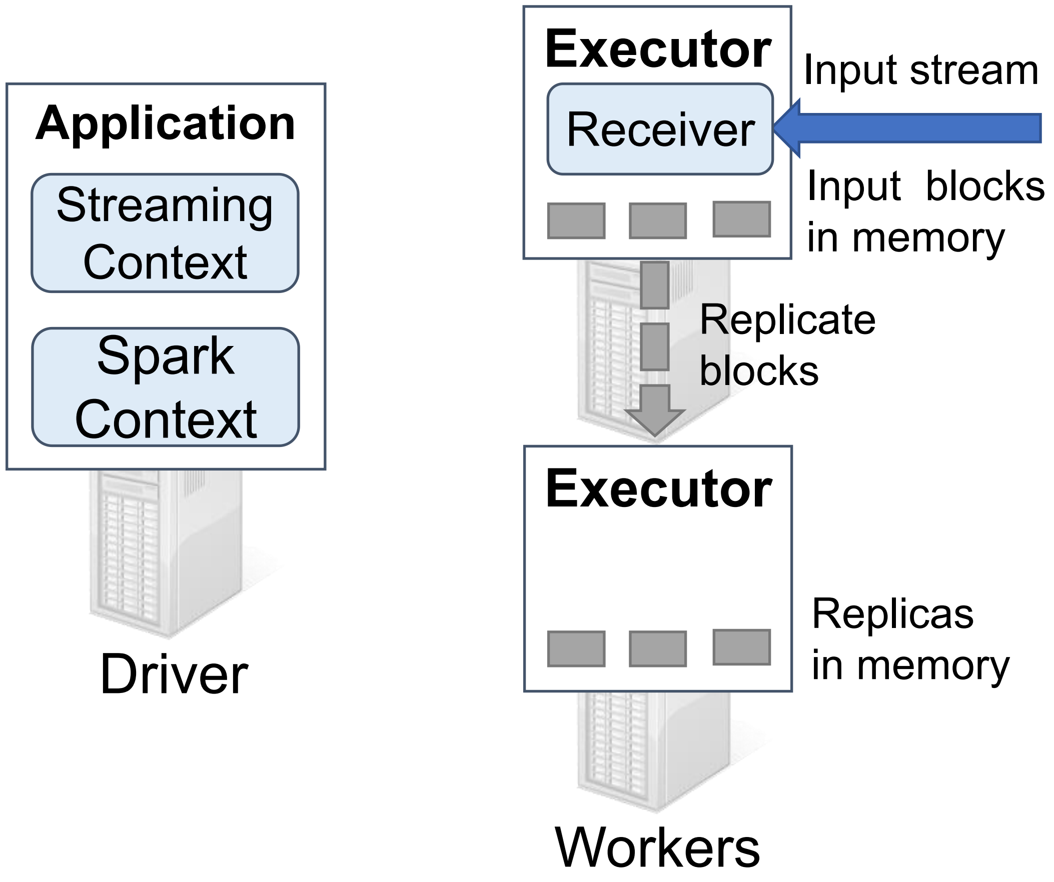
How does it work?

- ◆ Application runs StreamingContext and an underlying SparkContext
- ◆ Driver launches Receivers to run as long running tasks on Executors



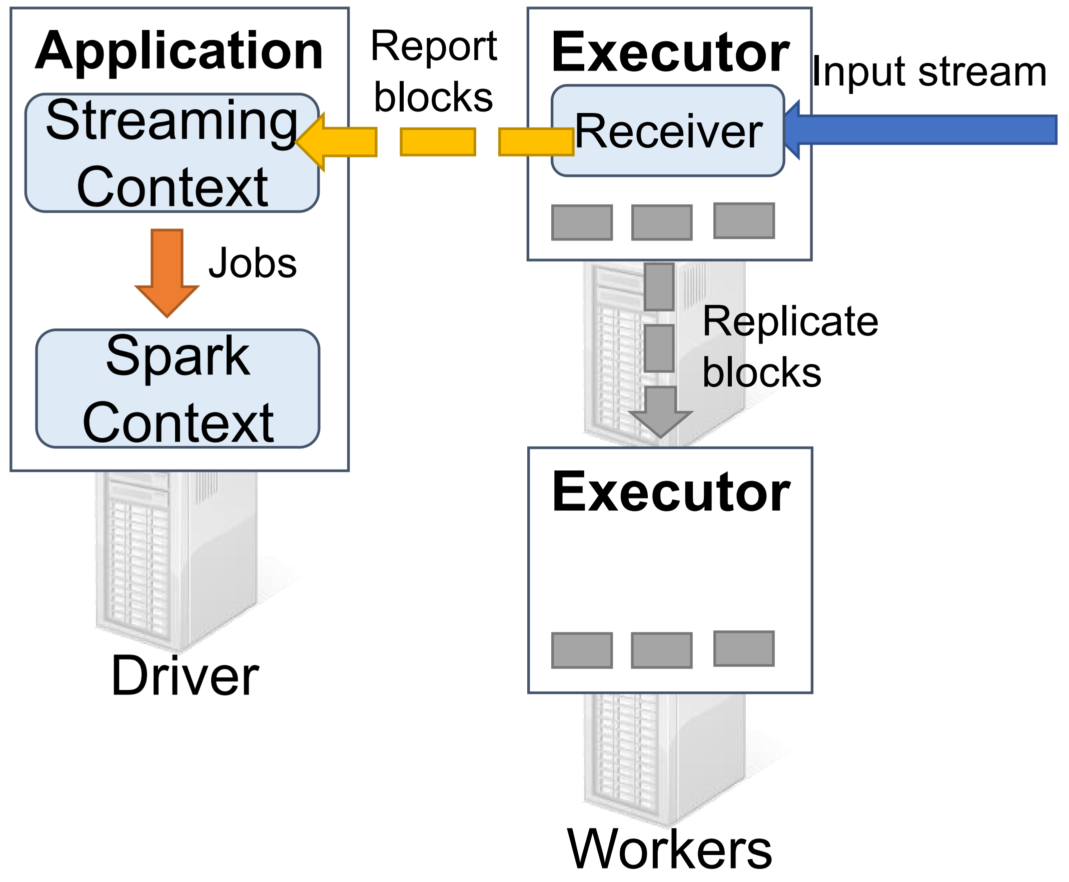
How does it work?

- Each Receiver receives input stream and divides it into blocks
- Blocks stored in Executor memory
- Blocks replicated to another executor



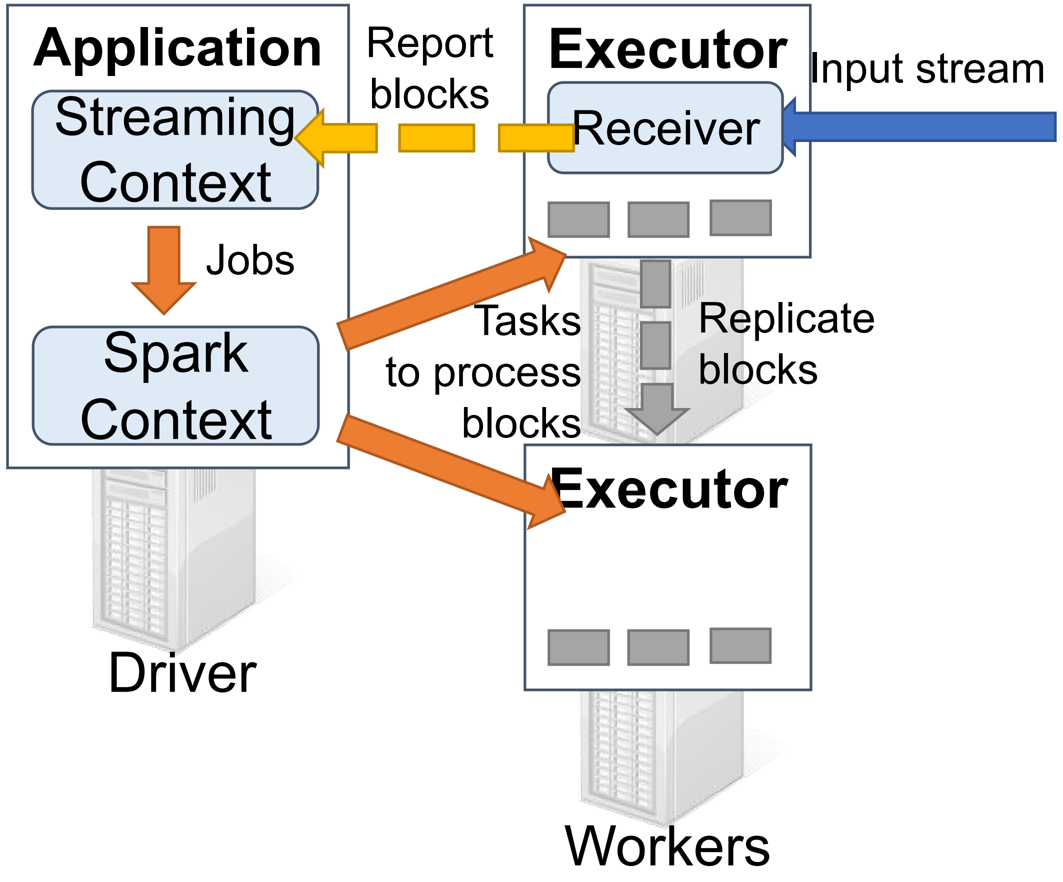
How does it work?

- Saved blocks reported to StreamingContext
- After every batch interval, StreamingContext treats received blocks as RDDs, and launches Spark jobs on SparkContext

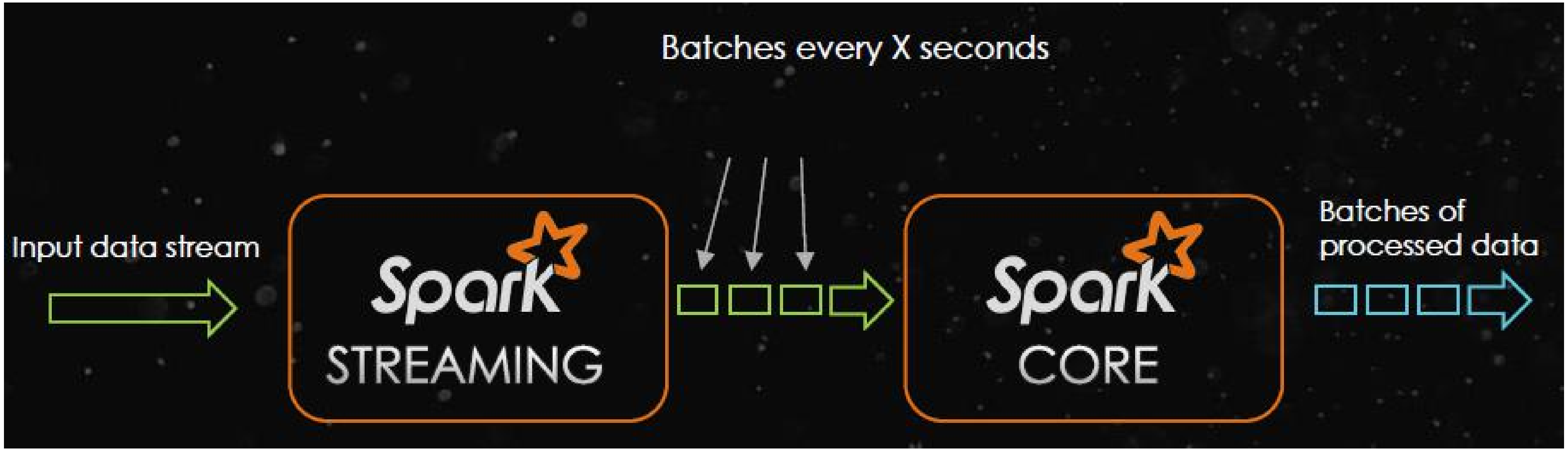


How does it work?

- SparkContext runs jobs by running tasks to process the blocks in executors' memory
- This cycle continues every batch interval

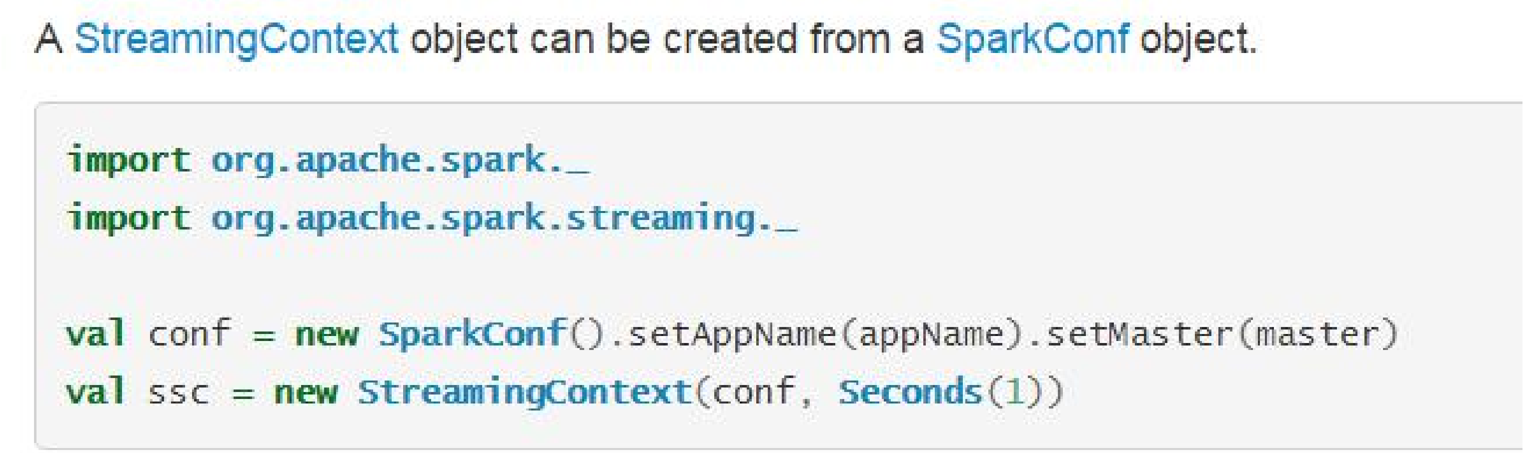


Spark Streaming



Initializing StreamingContext

- 第一种方式

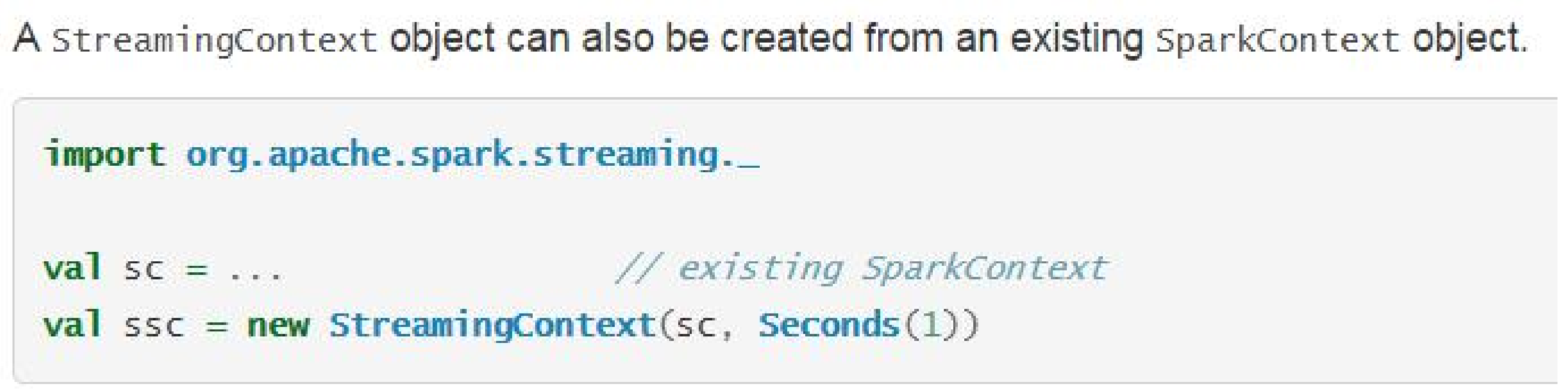


A StreamingContext object can be created from a SparkConf object.

```scala
import org.apache.spark._
import org.apache.spark.streaming._

val conf = new SparkConf().setAppName(appName).setMaster(master)
val ssc = new StreamingContext(conf, Seconds(1))
```

- 第二种方式



A `StreamingContext` object can also be created from an existing `SparkContext` object.

```scala
import org.apache.spark.streaming._

val sc = ...                // existing SparkContext
val ssc = new StreamingContext(sc, Seconds(1))
```

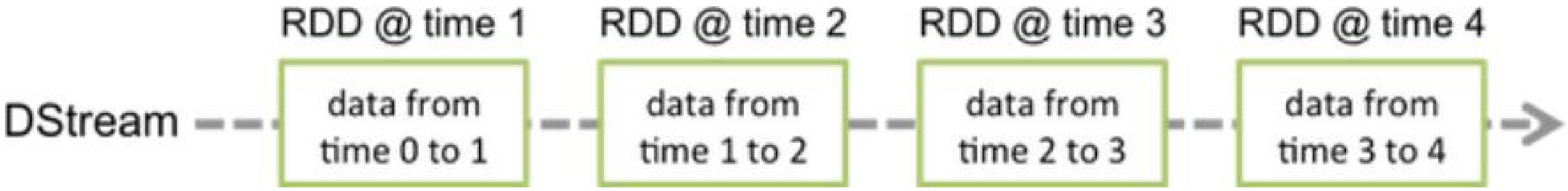
A Spark Streaming program

1. Define the input sources by creating input DStreams.
2. Define the streaming computations by applying transformation and output operations to DStreams.
3. Start receiving data and processing it using `streamingContext.start()`.
4. Wait for the processing to be stopped (manually or due to any error) using `streamingContext.awaitTermination()`.
5. The processing can be manually stopped using `streamingContext.stop()`.

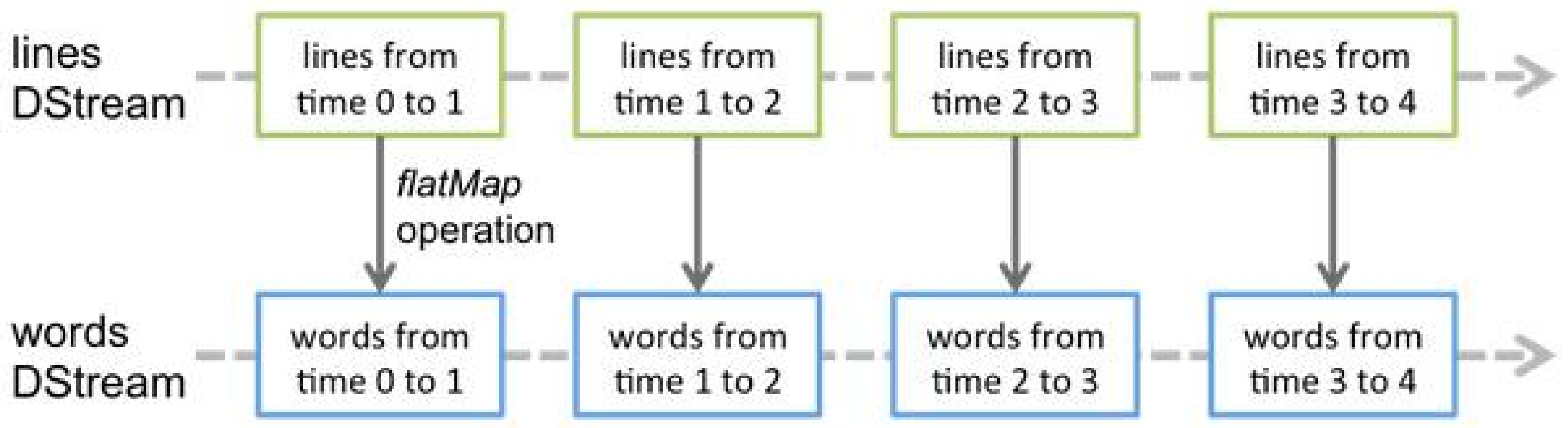
StreamingContext

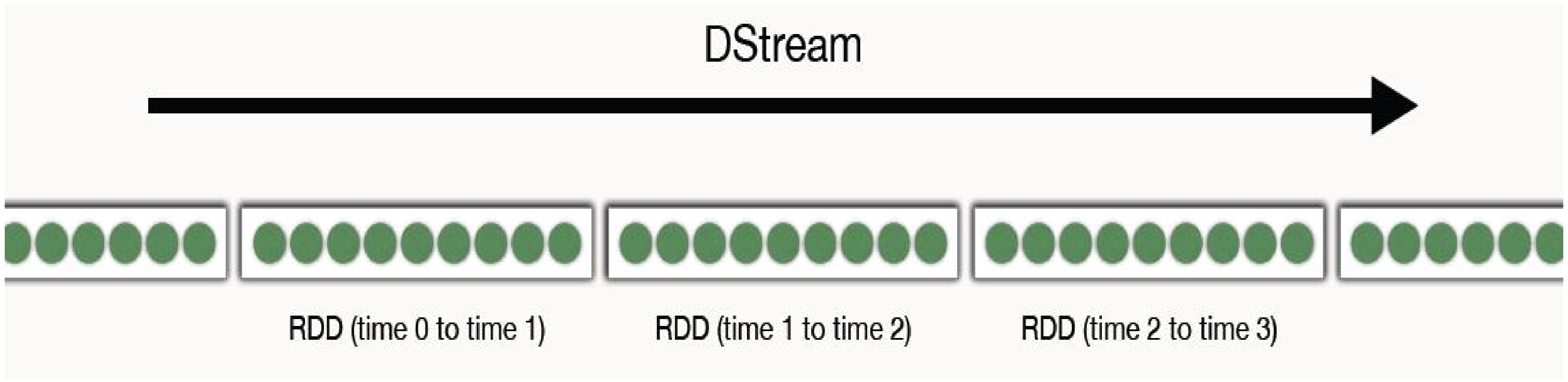
DStream

- Discretized Stream or DStream is the basic abstraction provided by Spark Streaming;
- It represents **a continuous stream of data**, either the input data stream **received from source**, or the processed data stream generated **by transforming the input stream**.
- Internally, a DStream is represented by a continuous series of RDDs, which is Spark’ s abstraction of an immutable, distributed dataset. Each RDD in a DStream contains data from a certain interval, as shown in the following figure.



DStream

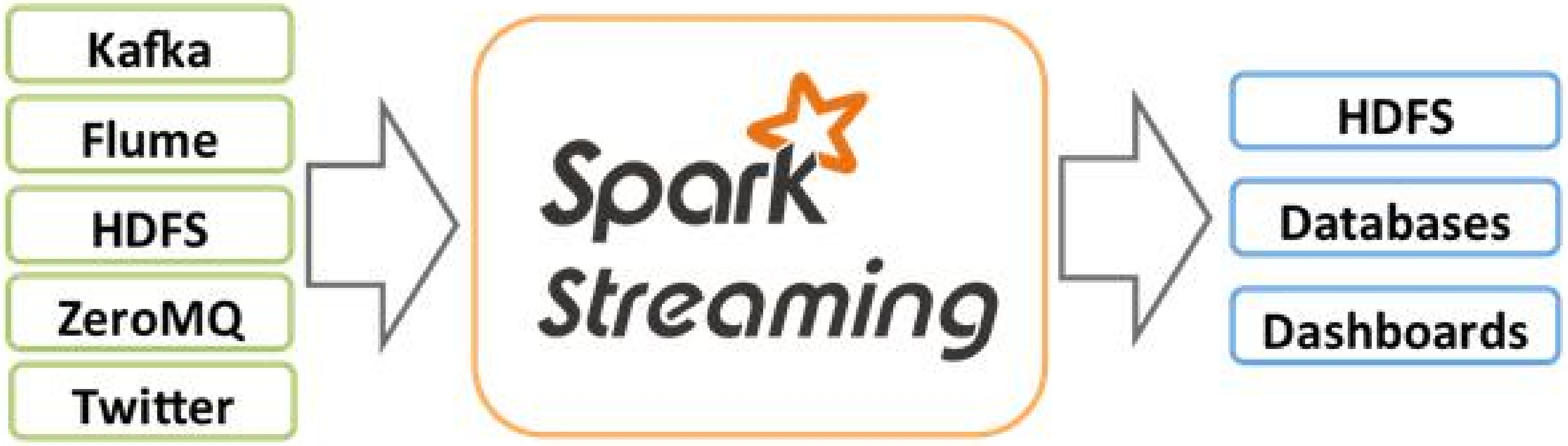




A transformation on a DStream = transformations on its RDDs

Input and Output Sourses

- Spark Streaming provides two categories of built-in streaming sources:
  - Basic sources: Sources directly available in the StreamingContext API. Example: file systems, socket connections, and Akka actors.
  - Advanced sources: Sources like Kafka, Flume, Kinesis, Twitter, etc. are available through extra utility classes.



Spark Streaming Integration

Flume Integration

http://spark.apache.org/docs/1.6.1/streaming-flume-integration.html

Kafka Integration

http://spark.apache.org/docs/1.6.1/streaming-kafka-integration.html

Kinesis Integration

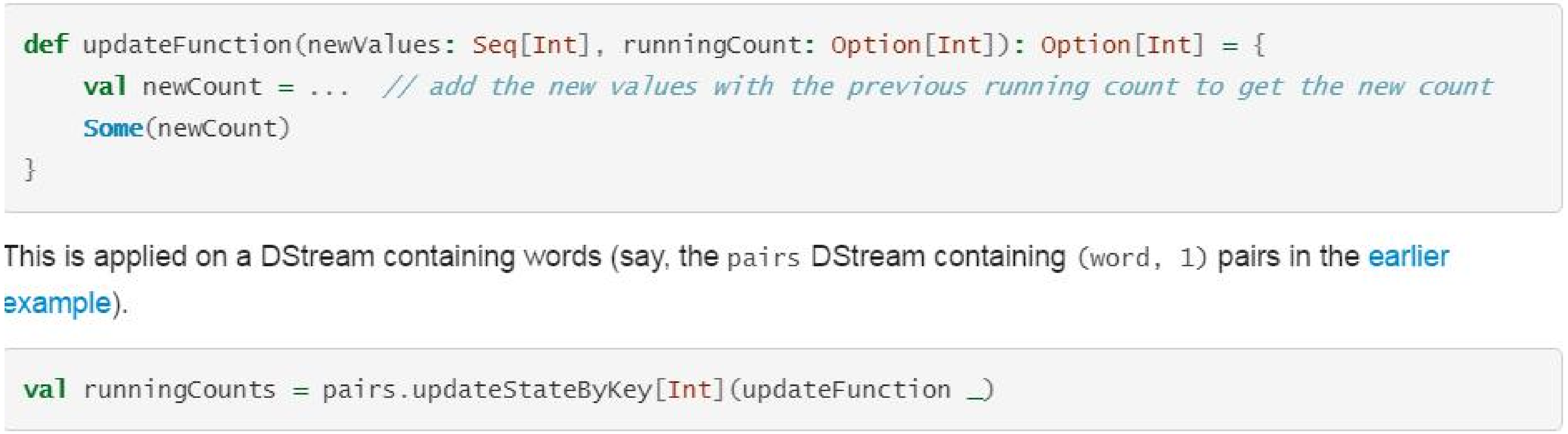
http://spark.apache.org/docs/1.6.1/streaming-kinesis-integration.html

Custom Receiver

http://spark.apache.org/docs/1.6.1/streaming-custom-receivers.html

updateStateByKey

- The updateStateByKey operation allows you to maintain arbitrary state while continuously updating it with new information.
  - **Define the state** - The state can be an arbitrary data type.
  - **Define the state update function** - Specify with a function how to update the state using the previous state and the new values from an input stream.



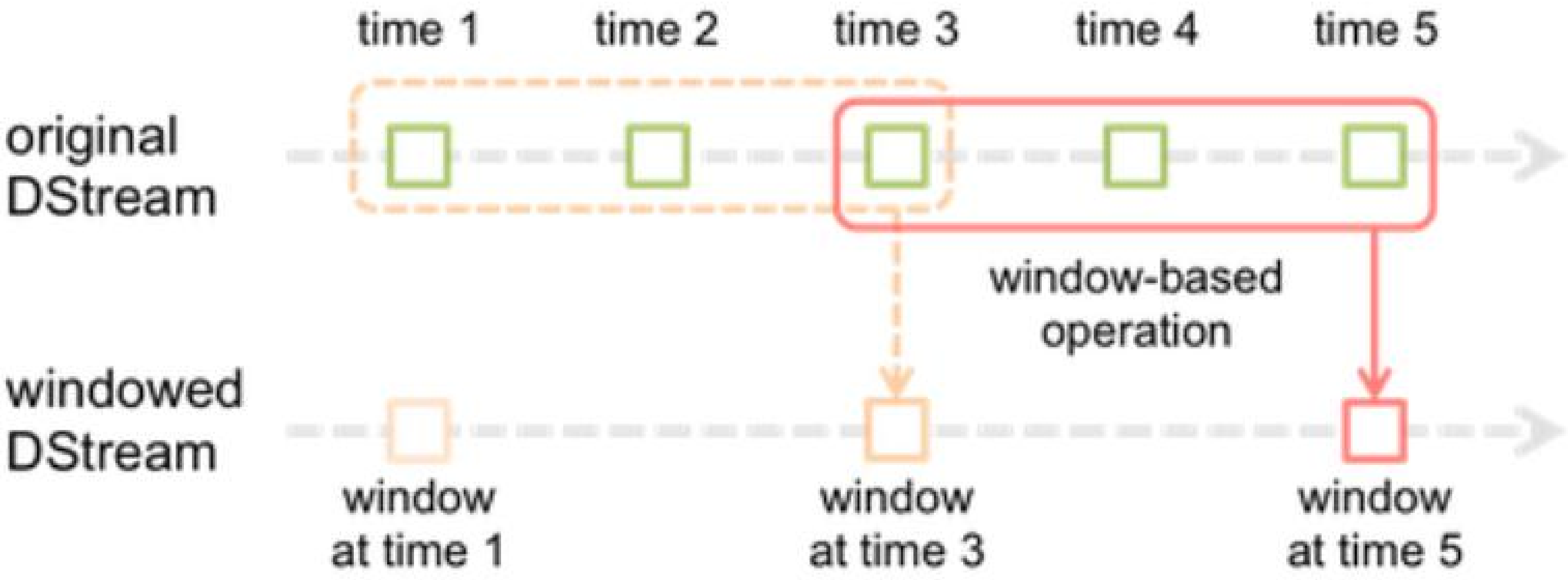
```
def updateFunction(newValues: Seq[Int], runningCount: Option[Int]): Option[Int] = {
    val newCount = ...  // add the new values with the previous running count to get the new count
    Some(newCount)
}
```

This is applied on a DStream containing words (say, the `pairs` DStream containing `(word, 1)` pairs in the earlier example).

```
val runningCounts = pairs.updateStateByKey[Int](updateFunction _)
```

Window Operations

- Spark Streaming also provides windowed computations, which allow you to apply transformations over a sliding window of data.



Window Operations

- 对**每三秒钟的数据**执行一次滑动窗口计算，这3秒内的3个RDD会被聚合起来进行处理，然后**过了两秒钟**，又会对最近三秒内的数据执行滑动窗口计算。所以每个滑动窗口操作，都必须指定两个参数，**窗口长度以及滑动间隔**，而且这两个参数值都必须是batch间隔的整数倍。

- *window length* - The duration of the window (3 in the figure).
- *sliding interval* - The interval at which the window operation is performed (2 in the figure).

```
// Reduce last 30 seconds of data, every 10 seconds
val windowedWordCounts = pairs.reduceByKeyAndWindow((a:Int,b:Int) => (a + b), Seconds(30), Seconds(10))
```

THANK YOU

上海育创网络科技有限公司