大数据Hadoop高薪直通车课程

Spark 高阶应用

讲师：轩宇（北风网版权所有)

课程大纲

1. Spark 应用开发
2. Spark HistoryServer
3. Spark on YARN
4. Spark Streaming
5. Spark Streaming 案例

课程大纲

1. Spark 应用开发
2. Spark HistoryServer
3. Spark on YARN
4. Spark Streaming
5. Spark Streaming 案例

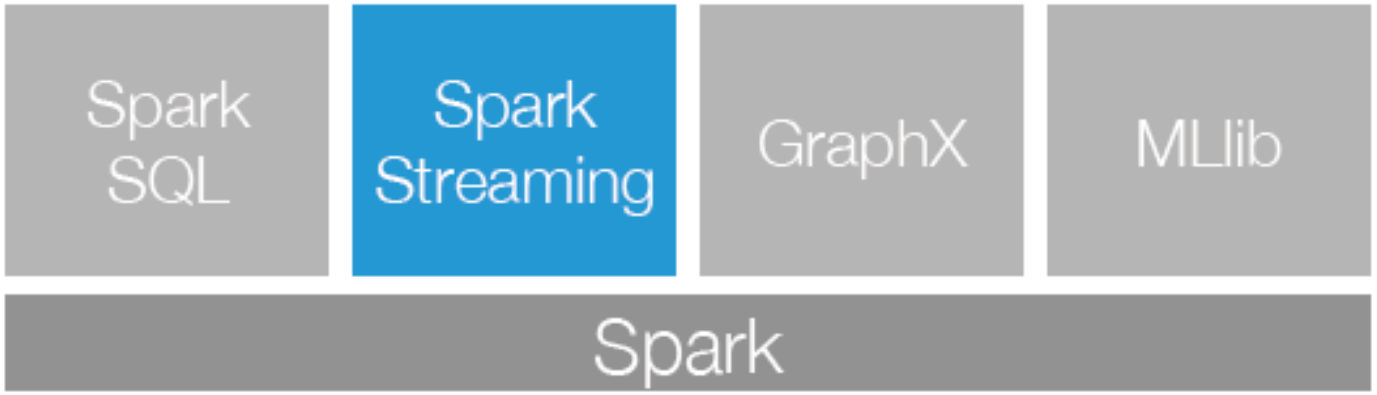
Streaming

- 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。

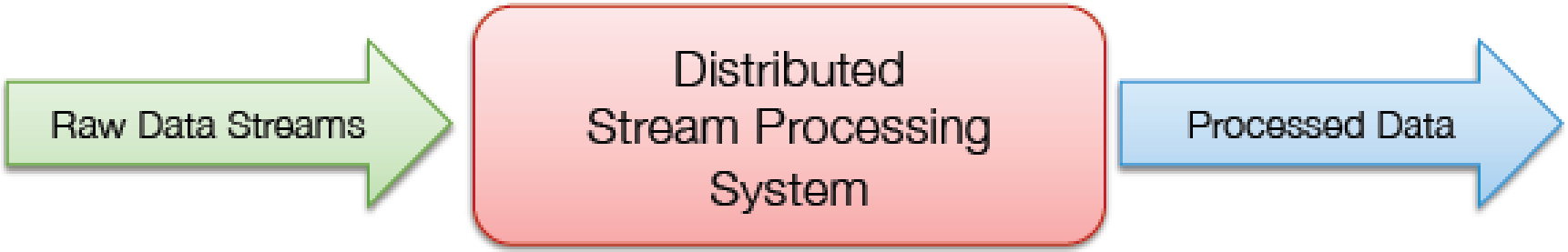
http://www.csdn.net/article/2015-03-09/2824135

What is Spark Streaming?

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

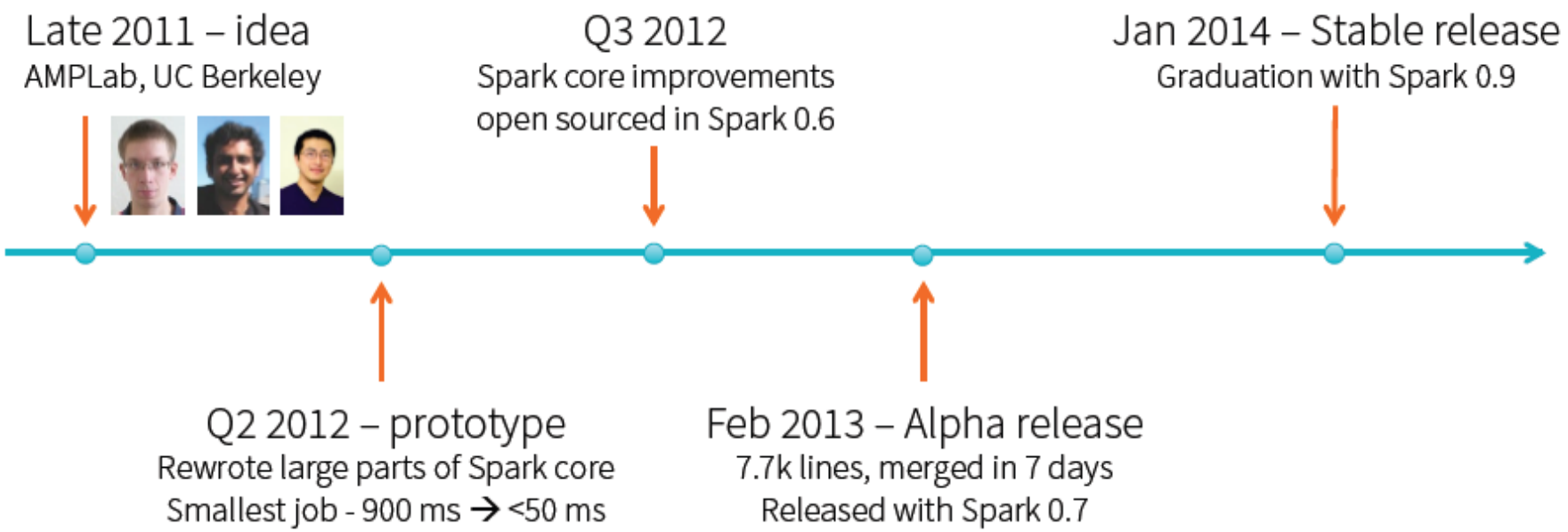


What is Spark Streaming?



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

Streaming History



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.

```
# TERMINAL 1:
# Running Netcat

$ nc -lk 9999

hello world



...
```

Scala | Java | Python

```
# 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

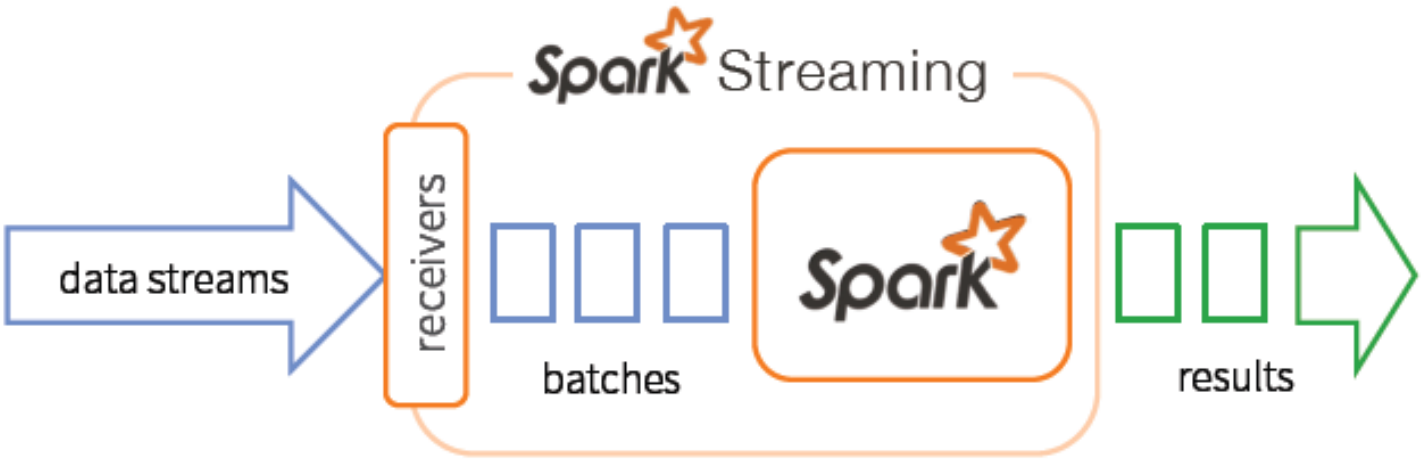
split lines into words

count the words

nc -lk 9999

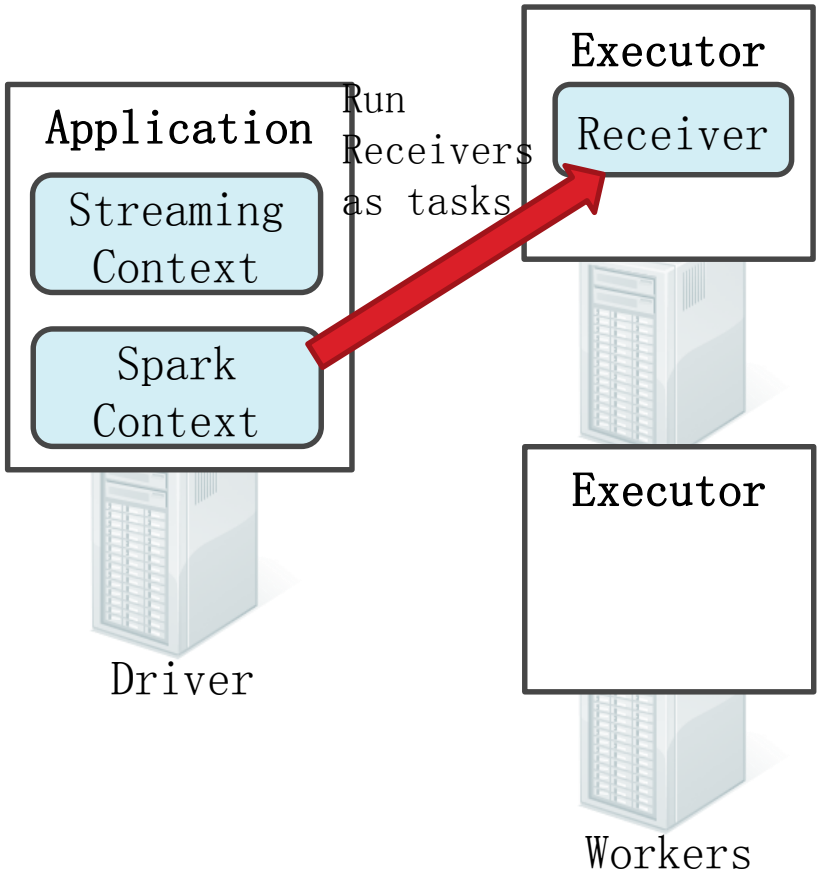
How does it work?

Data streams are chopped up into batches;
Each batch is processed in Spark;
Results pushed out in batches;



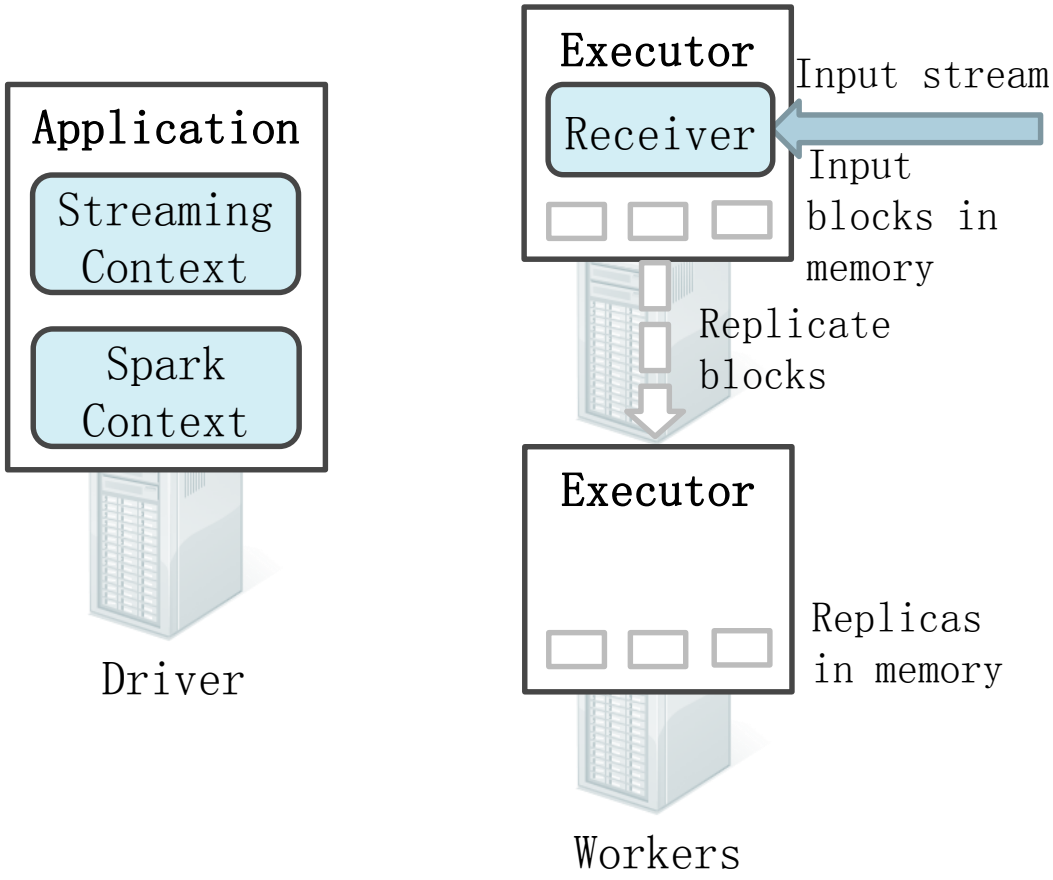
How does it work?

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



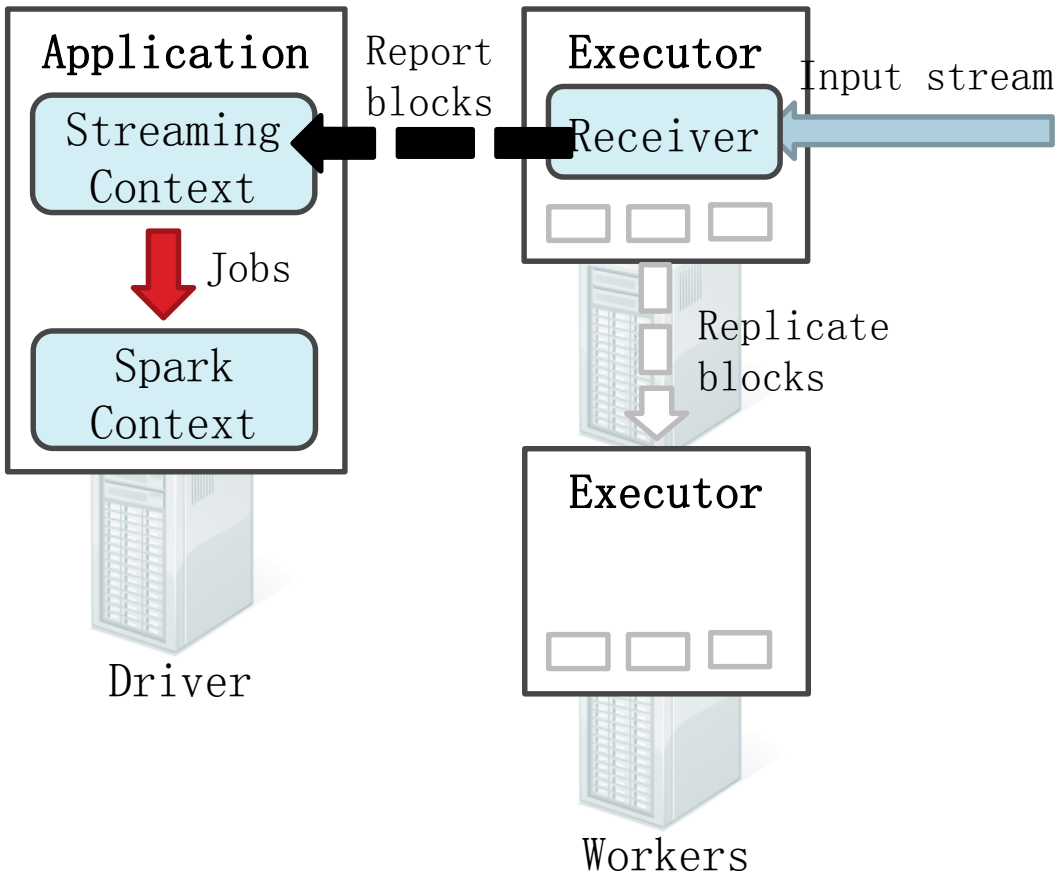
How does it work?

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



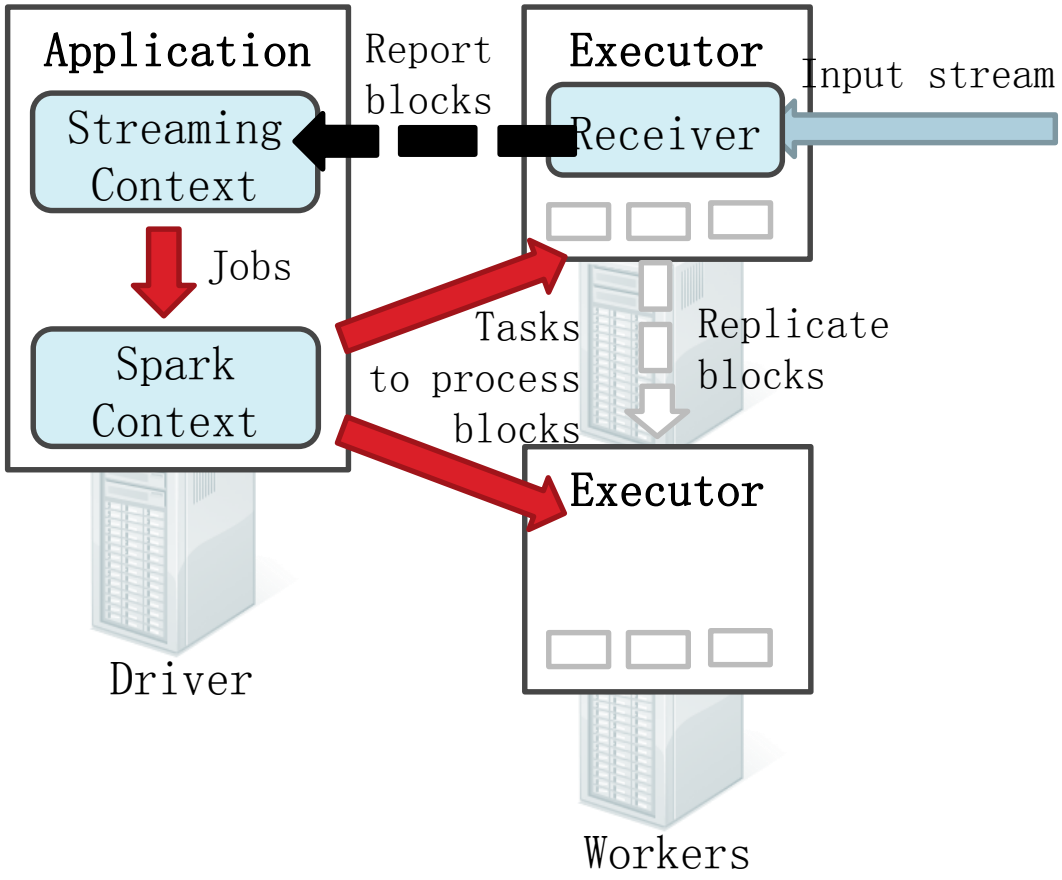
How does it work?

- 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.

```
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.

```
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()`.

textFileStream

```scala
/**
 * Create a input stream that monitors a Hadoop-compatible filesystem
 * for new files and reads them as text files (using key as LongWritable, value
 * as Text and input format as TextInputFormat). Files must be written to the
 * monitored directory by "moving" them from another location within the same
 * file system. File names starting with . are ignored.
 * @param directory HDFS directory to monitor for new file
 */
def textFileStream(directory: String): DStream[String] = {
  fileStream[LongWritable, Text, TextInputFormat](directory).map(_._2.toString)
}
```

HDFS 数据源

```
>>>>wc.txt
hadoop  spark   streaming
spark   hdfs    streaming
spark

bin/hdfs dfs -mkdir -p streaming/input/hdfs
bin/hdfs dfs -put /opt/datas/wc.txt streaming/input/hdfs/
```

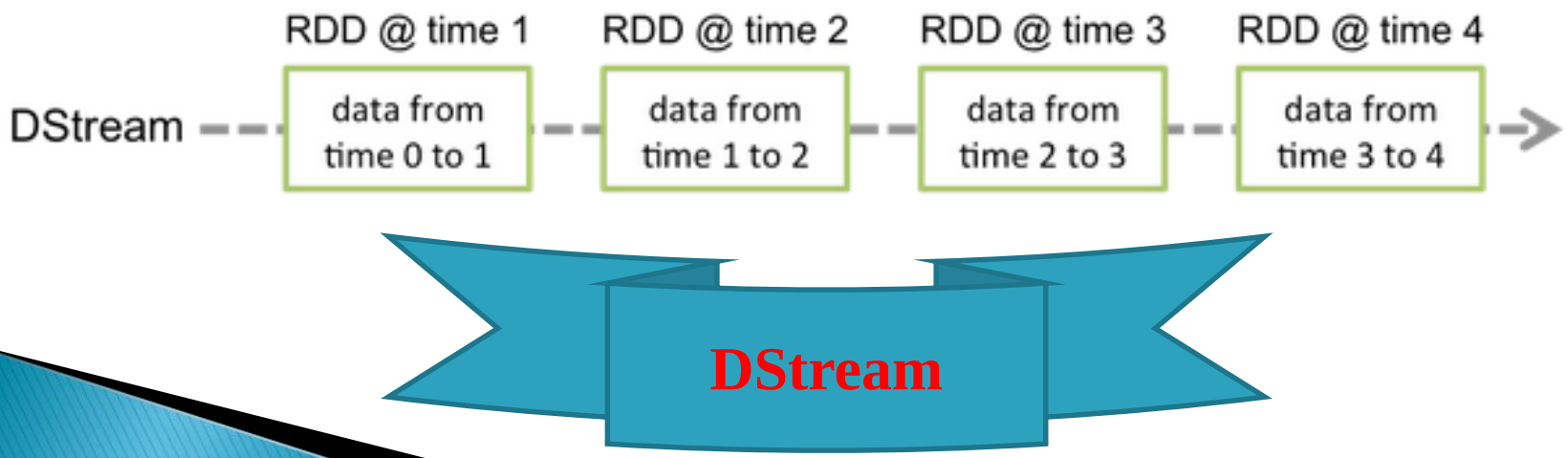
```
>>>>HdfsWordCount

import org.apache.spark._
import org.apache.spark.streaming._

val ssc = new StreamingContext(sc, Seconds(5))

val lines = ssc.textFileStream("/user/beifeng/streaming/input/hdfs/")
val words = lines.flatMap(_.split("\t"))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts.print()

ssc.start()
ssc.awaitTermination()
```

DStream

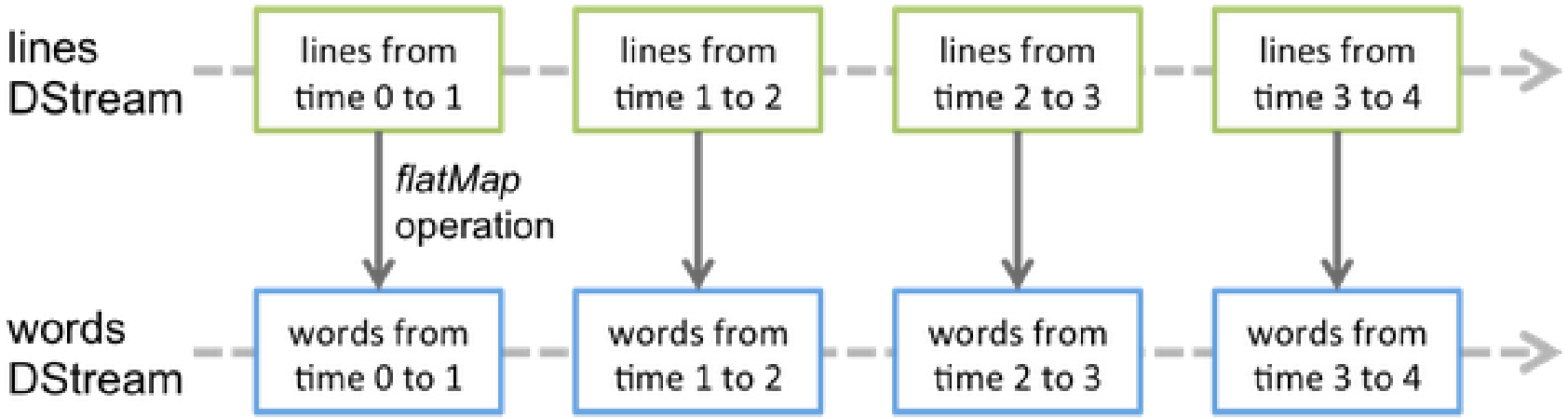
✓**Discretized Stream** or **DStream** is the basic abstraction provided by Spark Streaming;

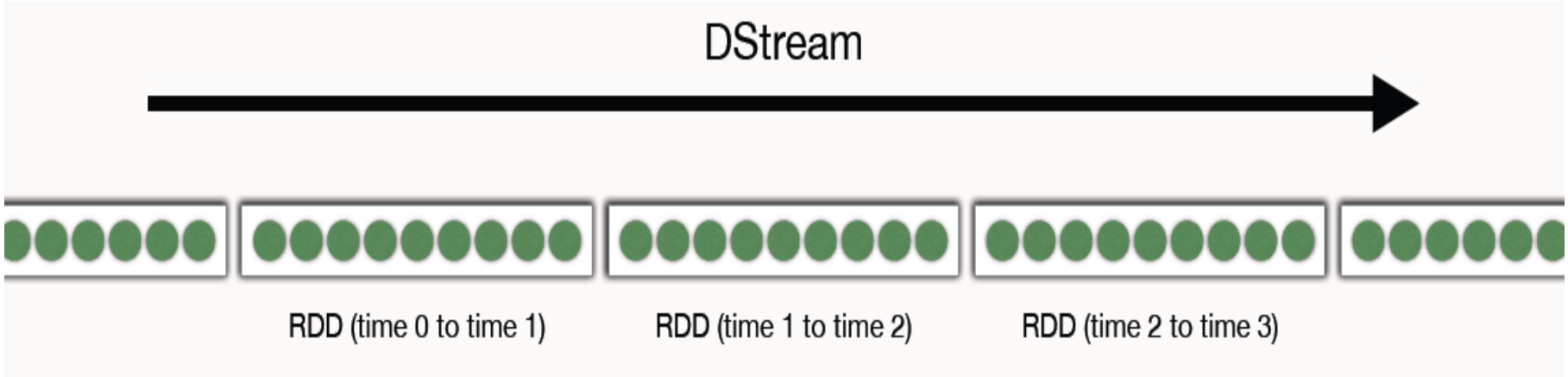
✓It represents **a continuous stream of data**, either the input data stream **received from source**, or the processed data stream generated by **transforming the input stream**.

✓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

Spark Streaming Application Develop Ways

- Spark Shell Code：开发、测试
- Spark Shell Load Scripts：开发、测试
- IDE Develop App：开发、测试、打包JAR（生产环境），spark-submit提交应用程序

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.

课程大纲

1. Spark 应用开发
2. Spark HistoryServer
3. Spark on YARN
4. Spark Streaming
5. Spark Streaming 案例

Spark Streaming Integration

http://spark.apache.org/docs/1.3.0/streaming-flume-integration.html

http://spark.apache.org/docs/1.3.0/streaming-kafka-integration.html

http://spark.apache.org/docs/1.3.0/streaming-kinesis-integration.html

http://spark.apache.org/docs/1.3.0/streaming-custom-receivers.html

Apache Kafka

Apache Kafka is publish-subscribe messaging rethought as a distributed commit log.

Fast Scalable Durable Distributed by Design

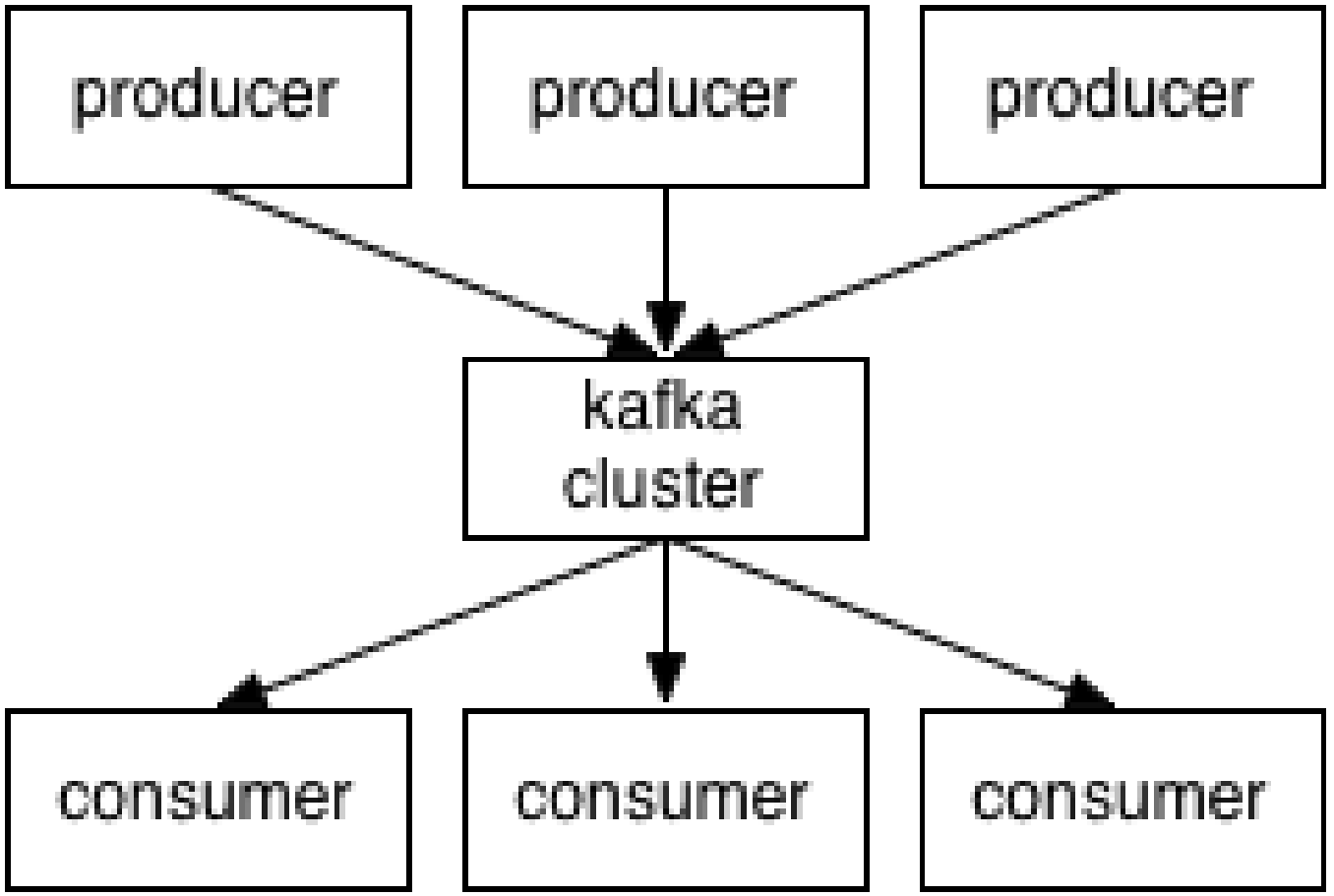
Apache Kafka

Apache Kafka是分布式发布-订阅消息系统。它最初由LinkedIn公司开发，之后成为Apache项目的一部分。Kafka是一种快速、可扩展的、设计内在就是分布式的，分区的和可复制的提交日志服务。

Apache Kafka与传统消息系统相比，有以下不同：

- 它被设计为一个分布式系统，易于向外扩展；
- 它同时为发布和订阅提供高吞吐量；
- 它支持多订阅者，当失败时能自动平衡消费者；
- 它将消息持久化到磁盘，因此可用于批量消费，例如ETL，以及实时应用程序。

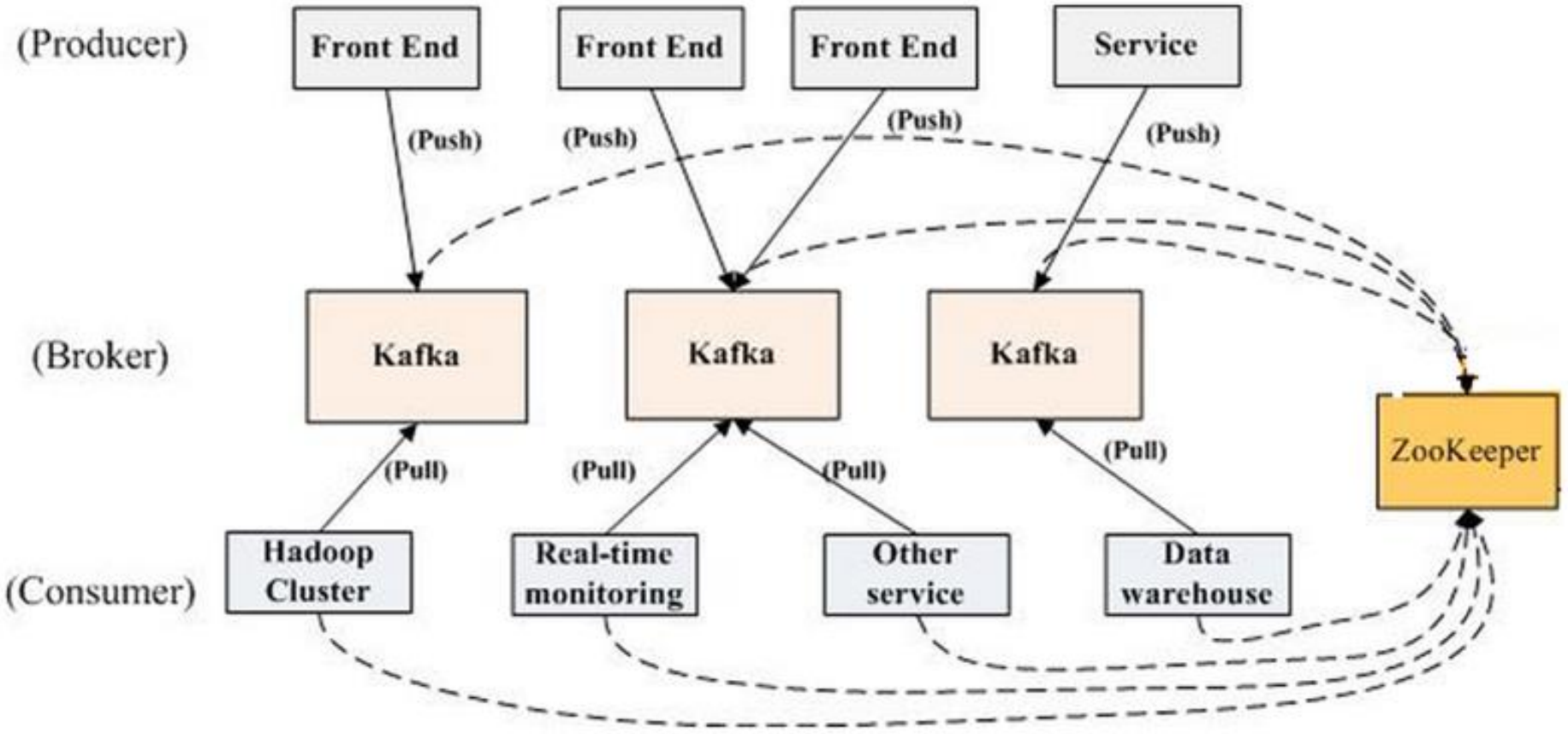
Apache Kafka Architecture



Apache Kafka Architecture

- 生产者（Producer）是能够发布消息到话题的任何对象。
- 已发布的消息保存在一组服务器中，它们被称为代理（Broker）或Kafka集群
- 消费者可以订阅一个或多个话题，并从Broker拉数据，从而消费这些已发布的消息。
- 话题（Topic）是特定类型的消息流。消息是字节的有效负载（Payload），话题是消息的分类名或种子（Feed）名。

Apache Kafka Architecture



Apache Kafka Install

- Step 1：Install Java
- Step 2：Install Zookeeper
- Step 3：Install Scala
- Step 4：Install Kafka

Apache Kafka Install

http://kafka.apache.org/documentation.html#quickstart

- Step 1: Download the code
- Step 2: Start the server
- Step 4: Send some messages
- Step 4：Install Kafka
- Step 5: Start a consumer

本课程版权归北风网所有

欢迎访问我们的官方网站

www.ibeifeng.com