Spark streaming and Spark R

TANG Gen

Outline

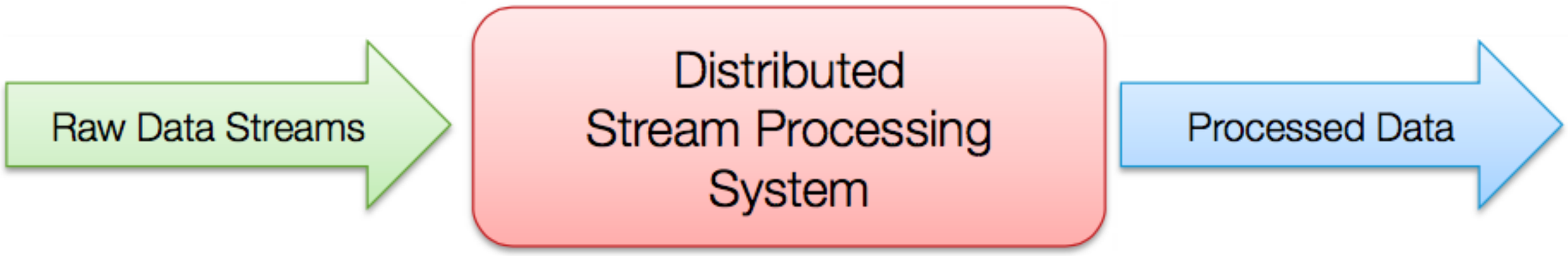
- Why Spark streaming?
- A simple example
- Architecture and abstraction
- Transformations and output operations
- Spark R

Why Spark streaming

- Many Big Data applications need to process large data steams in realtime

From spark summit 2014

Why Spark streaming



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

From spark summit 2014

A simple example

/Localhost stream

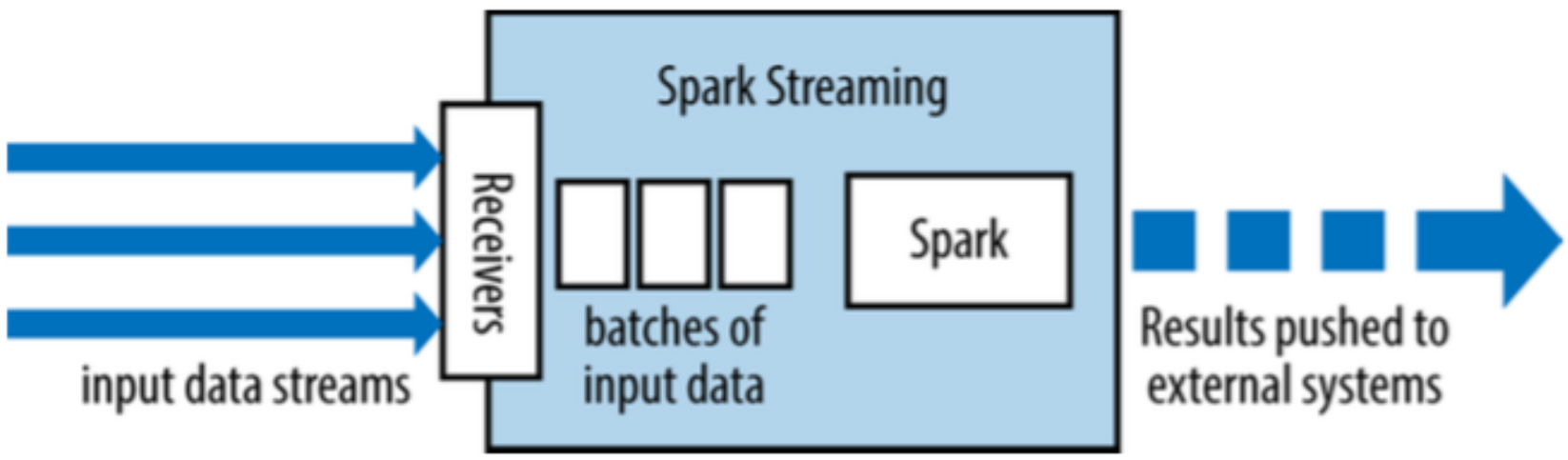
```
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.dstream.DStream
import org.apache.spark.streaming.Duration
import org.apache.spark.streaming.Seconds
import org.apache.spark._

val ssc = new StreamingContext(sc, Seconds(10))
val lines = ssc.socketTextStream("localhost", 9999)
val errorLines = lines.filter(_.contains("error"))
errorLines.print()

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

- Like Spark, StreamingContext is the entry point of Spark streaming

- .print() is an output operation of DStream. Without an action, we can not start a stream

Architecture and Abstraction

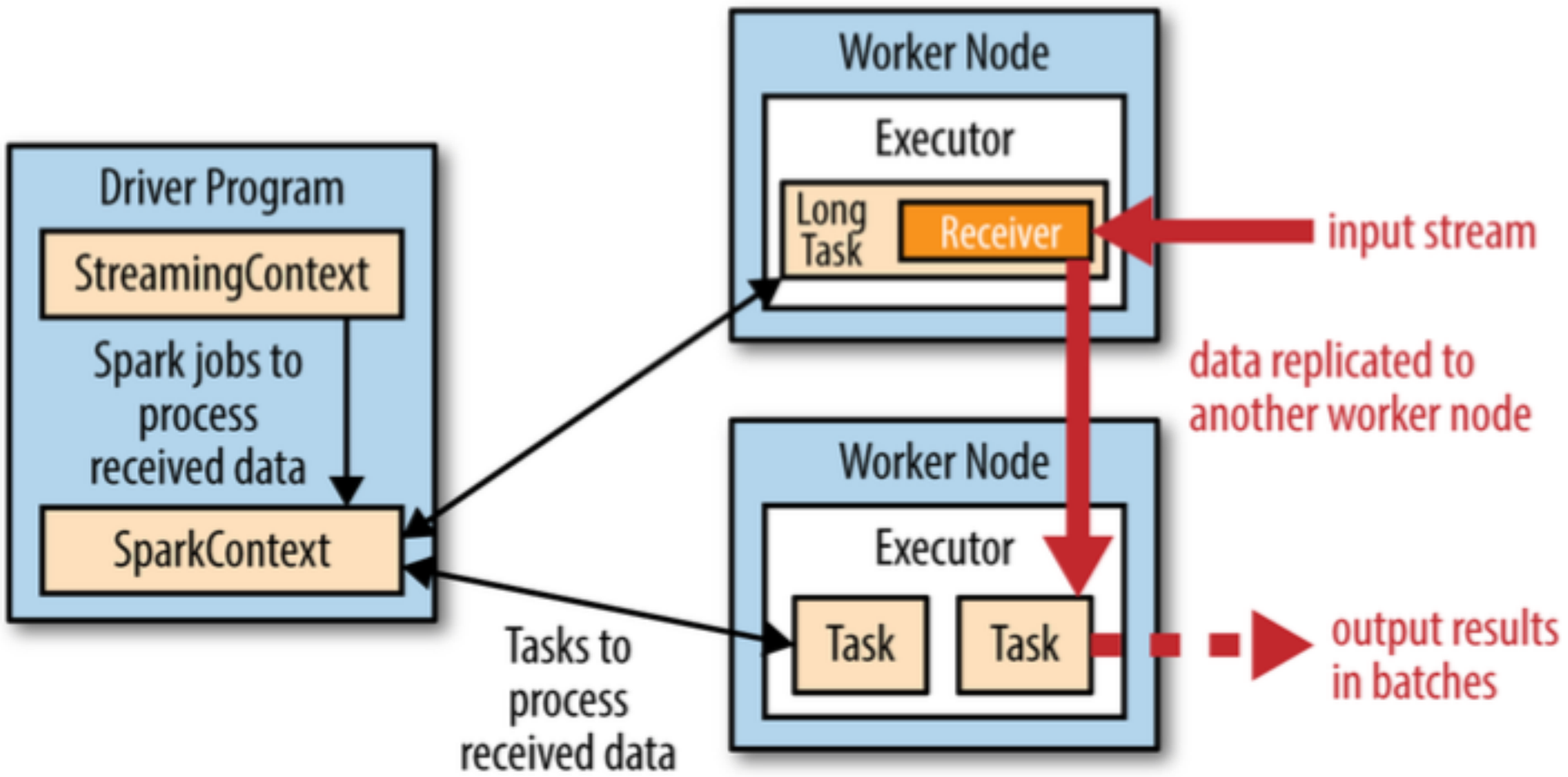
/Architecture



High-level architecture

Architecture and Abstraction

/Architecture

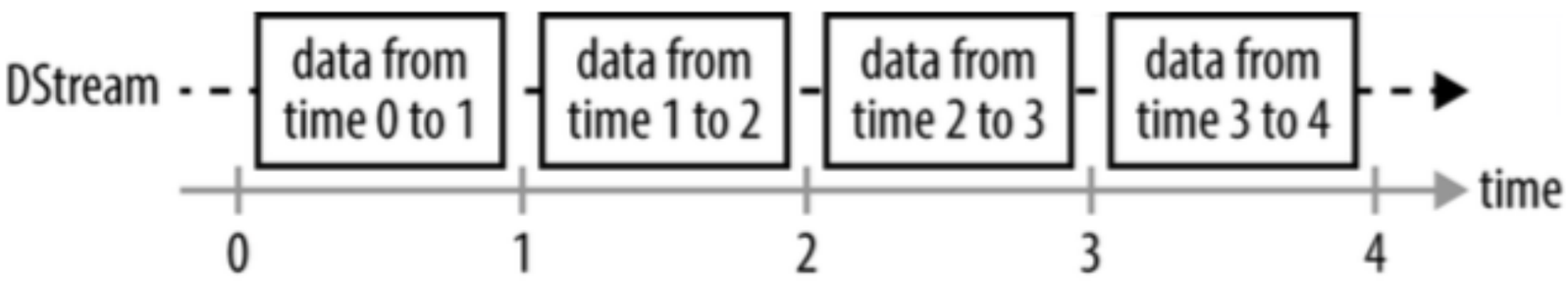


Architecture and Abstraction

/Abstraction

- Spark Streaming runs a streaming computation as a **series of ver small, deterministic batch jobs**

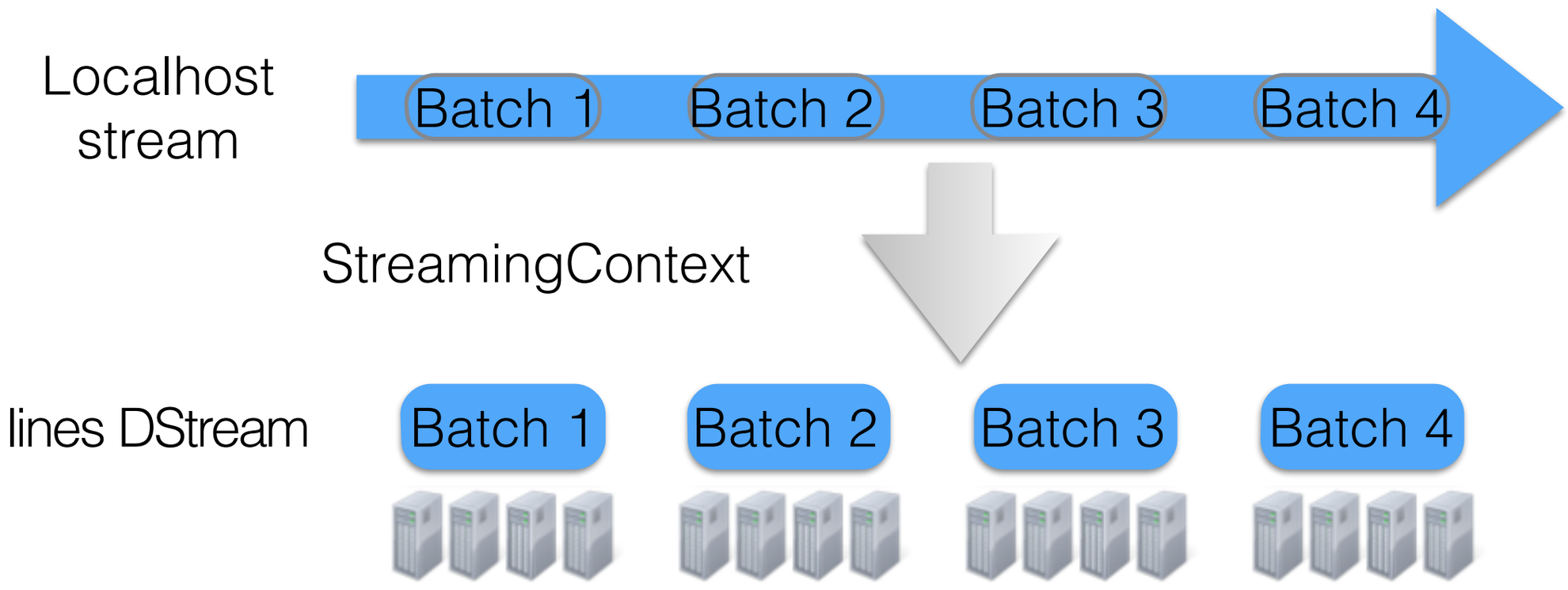
- The programming abstraction in Spark Streaming is a discretized stream or a **DStream**, which is a sequence of RDDs, where each RDD has one time slice of the data in the stream



Architecture and Abstraction

/Example

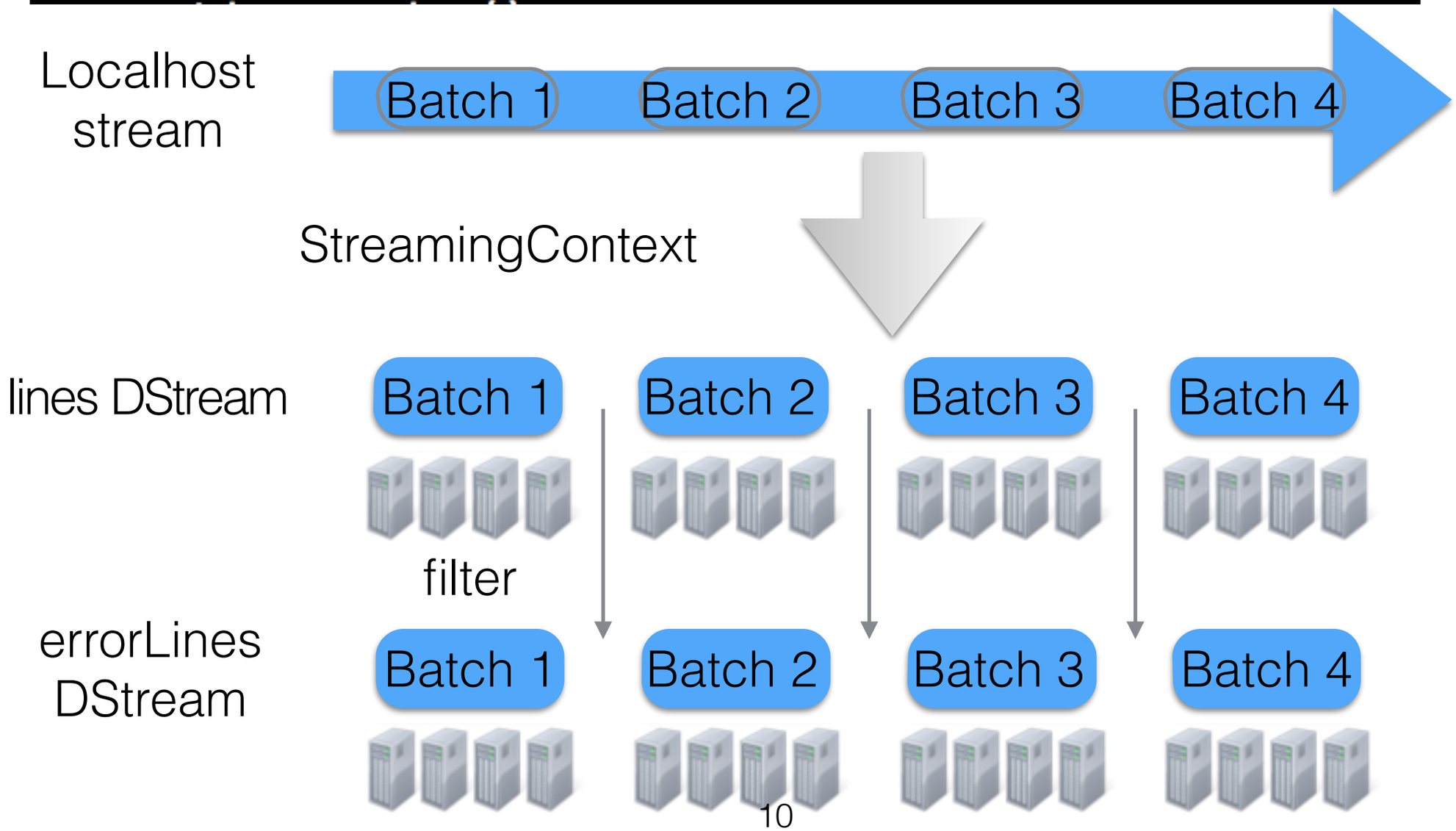
```
val ssc = new StreamingContext(sc, Seconds(10))
val lines = ssc.socketTextStream("localhost", 9999)
```



Architecture and Abstraction

/Example

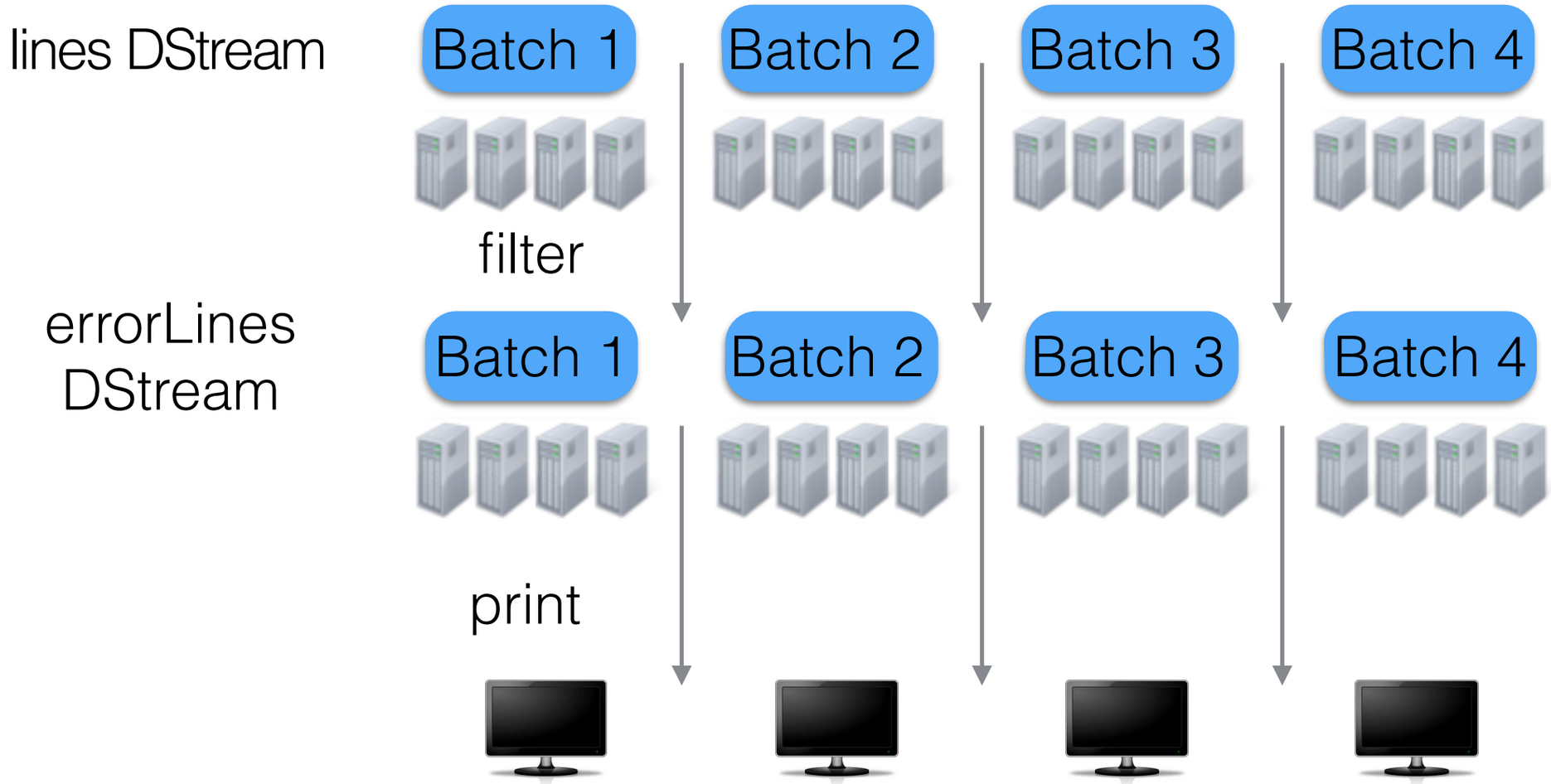
```
val ssc = new StreamingContext(sc, Seconds(10))
val lines = ssc.socketTextStream("localhost", 9999)
val errorLines = lines.filter(_.contains("error"))
```



Architecture and Abstraction

/Example

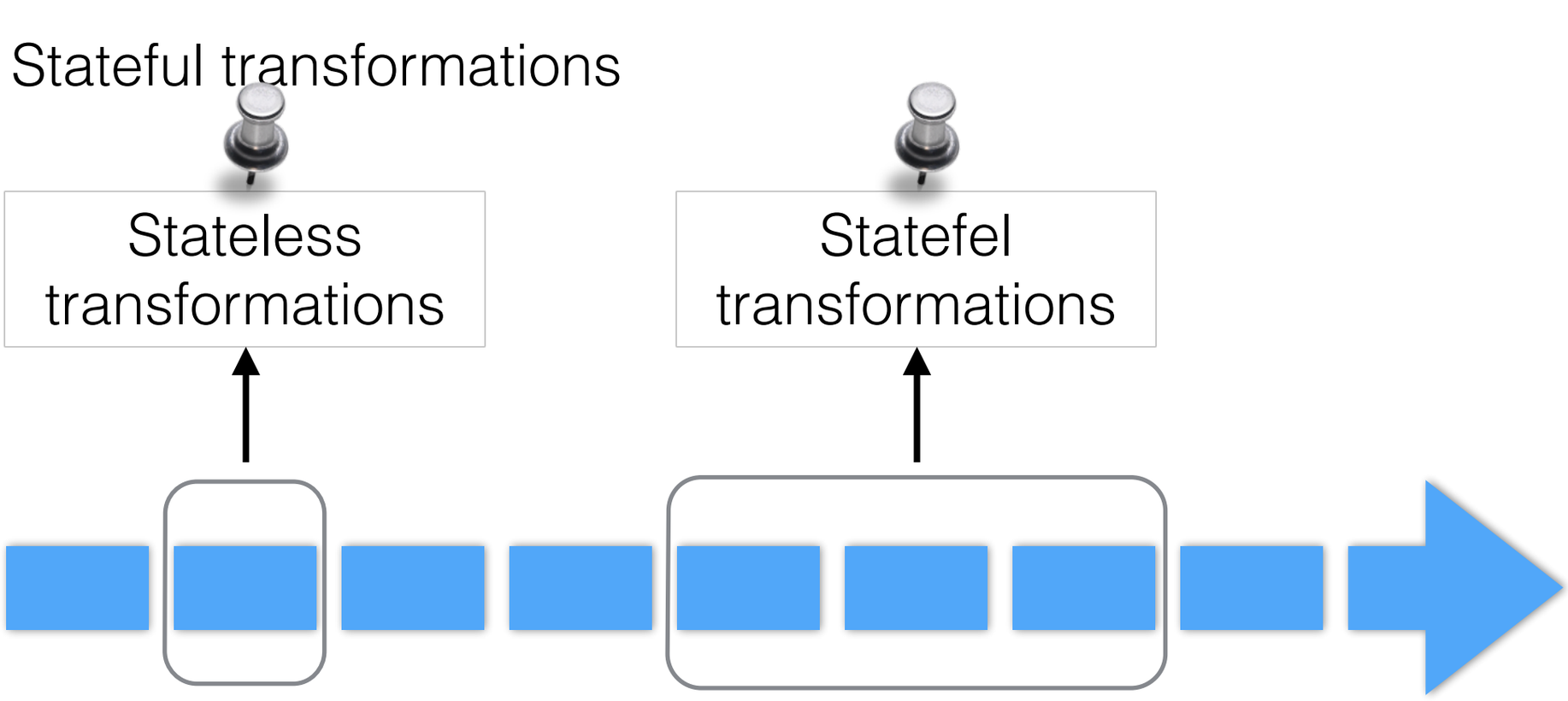
```
val errorLines = lines.filter(_.contains("error"))
errorLines.print()
```



Transformations and output operations

/Transformations

- There are two types of transformation
  - Stateless tranformations
  - Stateful transformations



Transformations and output operations

/Output operations

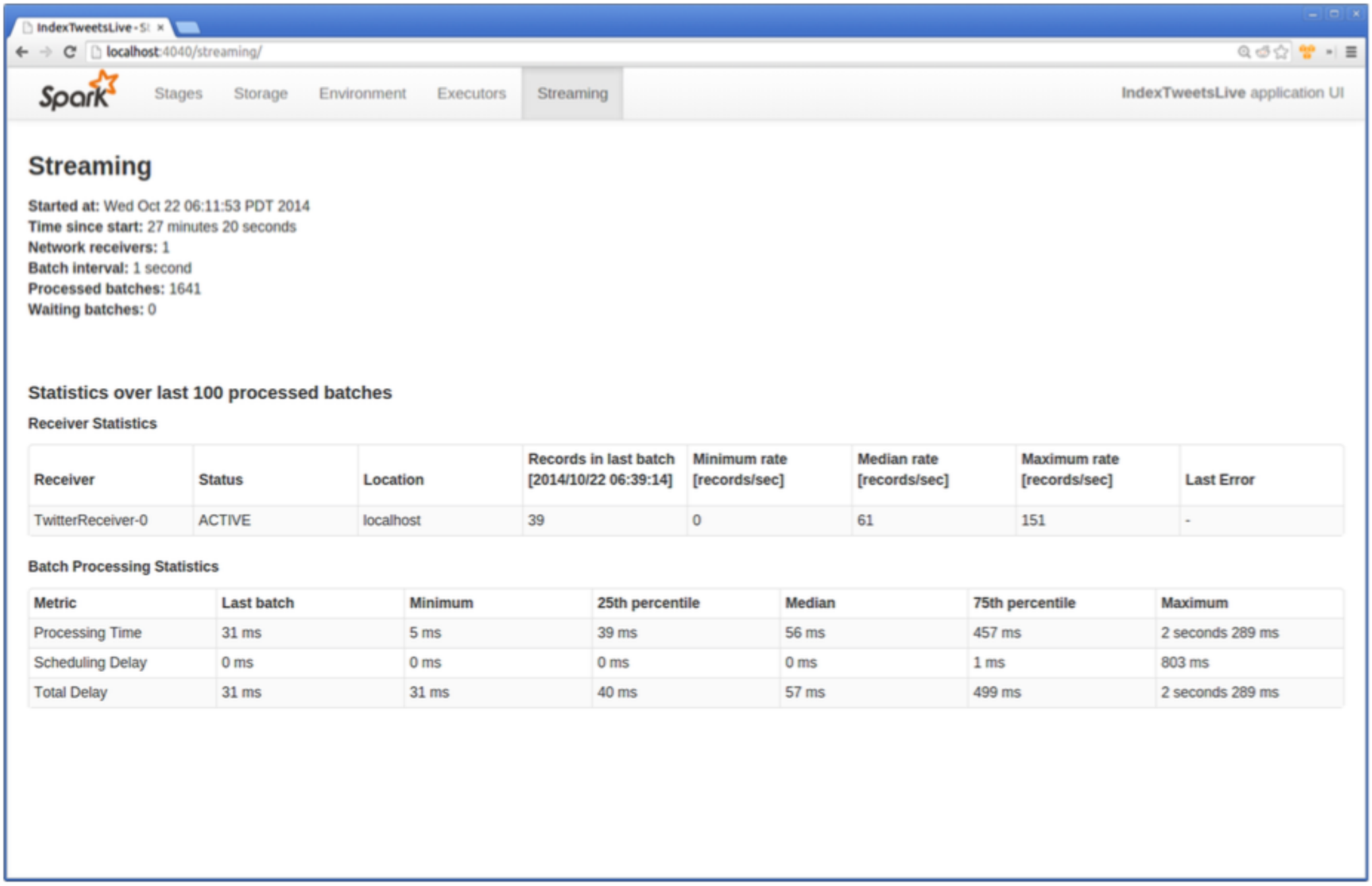
- Output operations allow DStream's data to be pushed out external systems like a database or a file systems

- They trigger the actual execution of all the DStream transformations

- Without an output operation, a Spark streaming can not be started

Output operation	Description
print	Print the first ten elements
saveAsTextFiles()	Save the data in DStream as text file
saveAsObjectFiles()	Save the data in DStream as object file
saveAsHadoopFiles()	Save the data in DStream as hadoop file
foreachRDD()	Apply the function to every element in DStream

Spark streaming UI



Spark R

/R vs. Python

- They are all developed by C
- They are have packages for almost all the topics in **statistics** and **machine learning**
- However, R is most used by statistician and Python by machine learner
- Python is easier to learn
- R has less code to write
- To my knowledge, R have a lot of package about time series, but python not

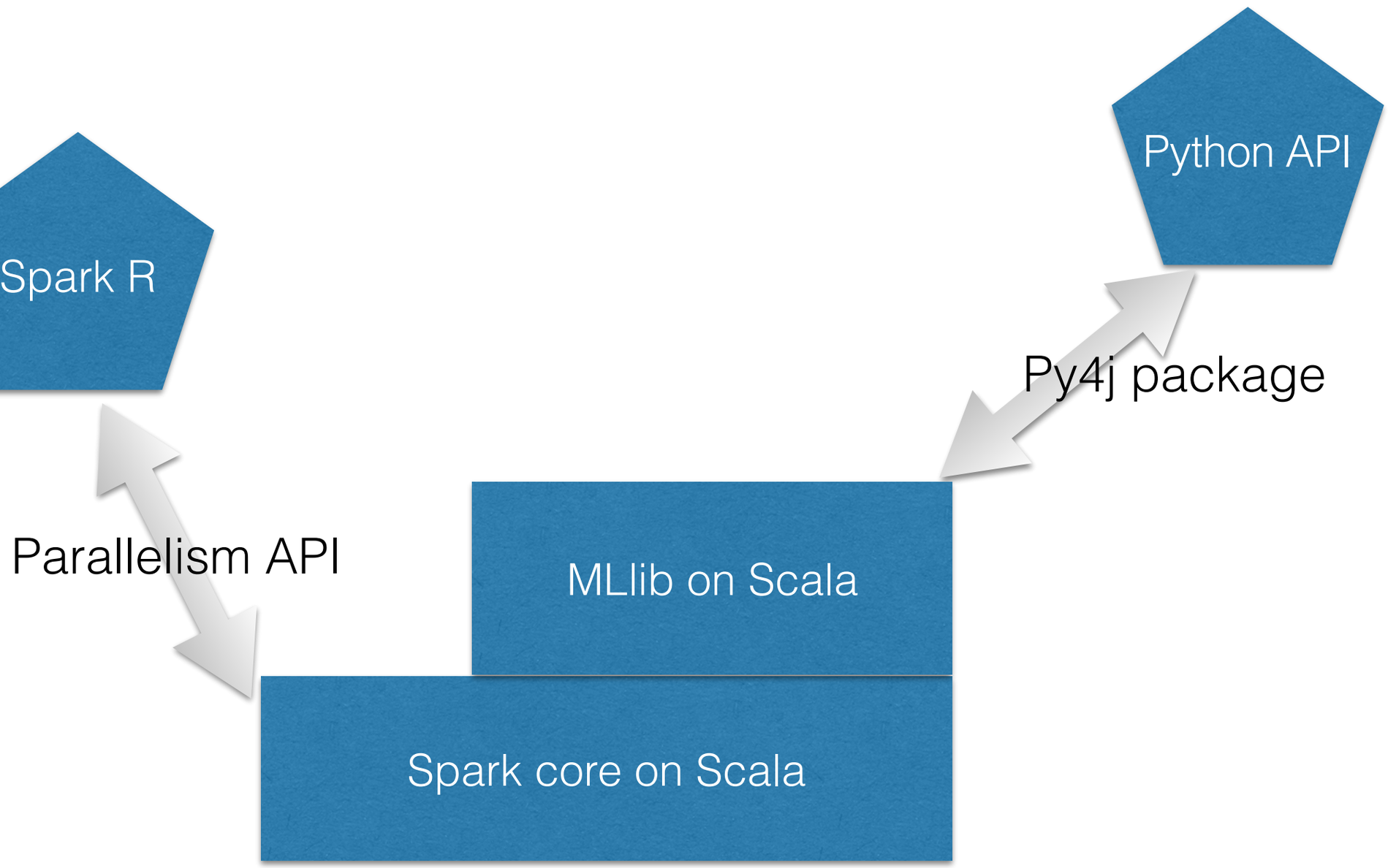
Spark R

/R vs. Python

- R has a project on parallel calculation: RHadoop
  - It is Open-source at the begining
  - It is now owned by RevolutionAnalytics (brought by MS)
    - **Strongly strongly not recommended**
- For the python machine learner community, parallel calculation is not their focus
  - To my knowledge, there is no project as RHadoop

Spark R

/MLlib on python vs. Spark R



Spark R

/MLlib on python vs. Spark R

- There is no “True” development in Python API, all the machine learning algorithm is coded in Scala

- All the algorithm of statistics on Spark R will be developed in R

Case studies

/Spark at sharethrough

http://blog.cloudera.com/blog/2014/03/letting-it-flow-with-spark-streaming/

- Overcoming 3 major challenges encountered while developing production streaming jobs

- Write streaming applications the same way you write batch jobs, reusing code

Case studies

/Spark at sharethrough