



# 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 streams in realtime

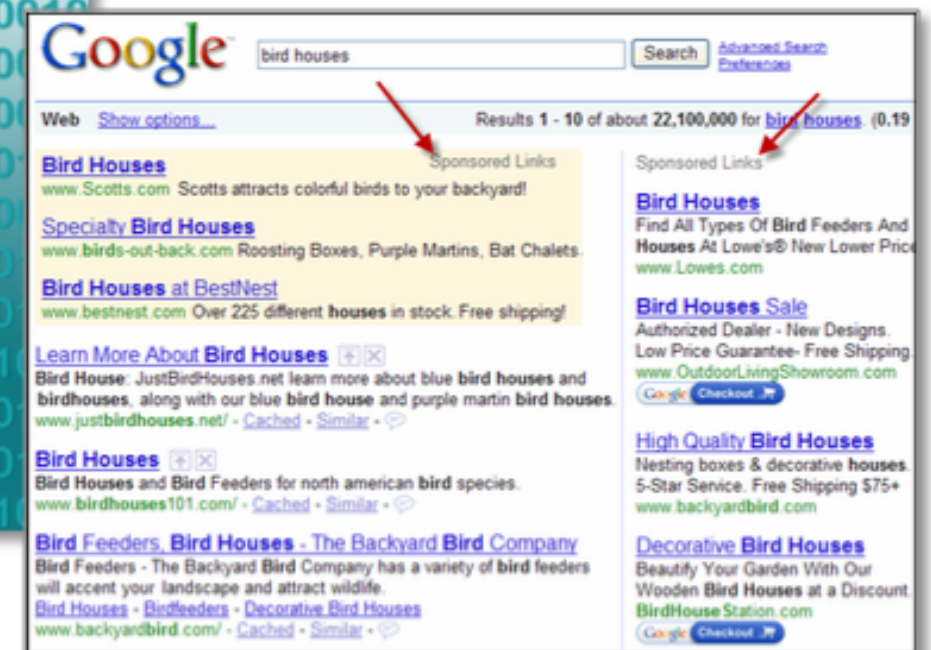
## Website monitoring



## Fraud detection

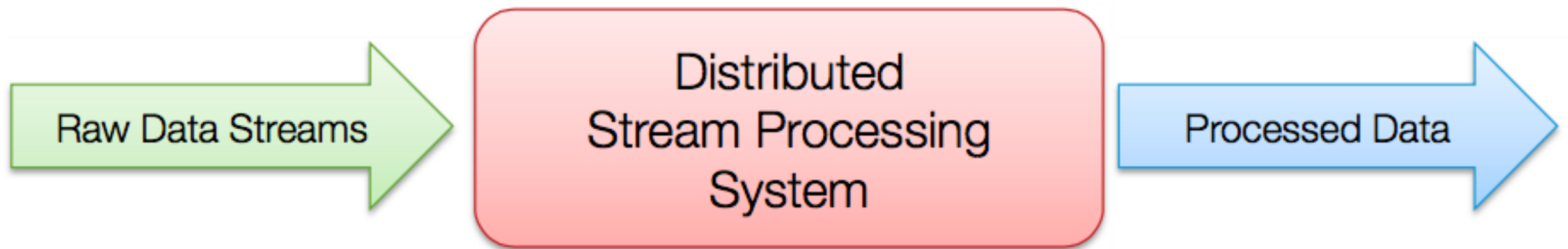


## Ad monetization



*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._

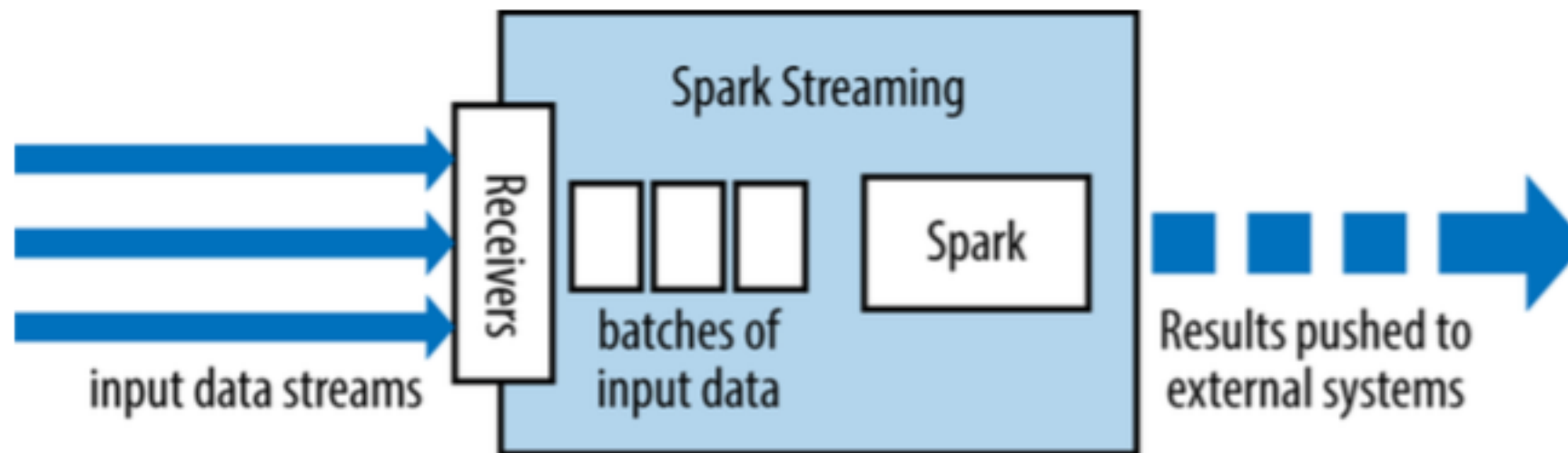
val ssc = new StreamingContext(sc, Seconds(10))
val lines = ssc.socketTextStream("localhost", 9999)
val errorLines = lines.filter(_.contains("error"))
errorLines.print()

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

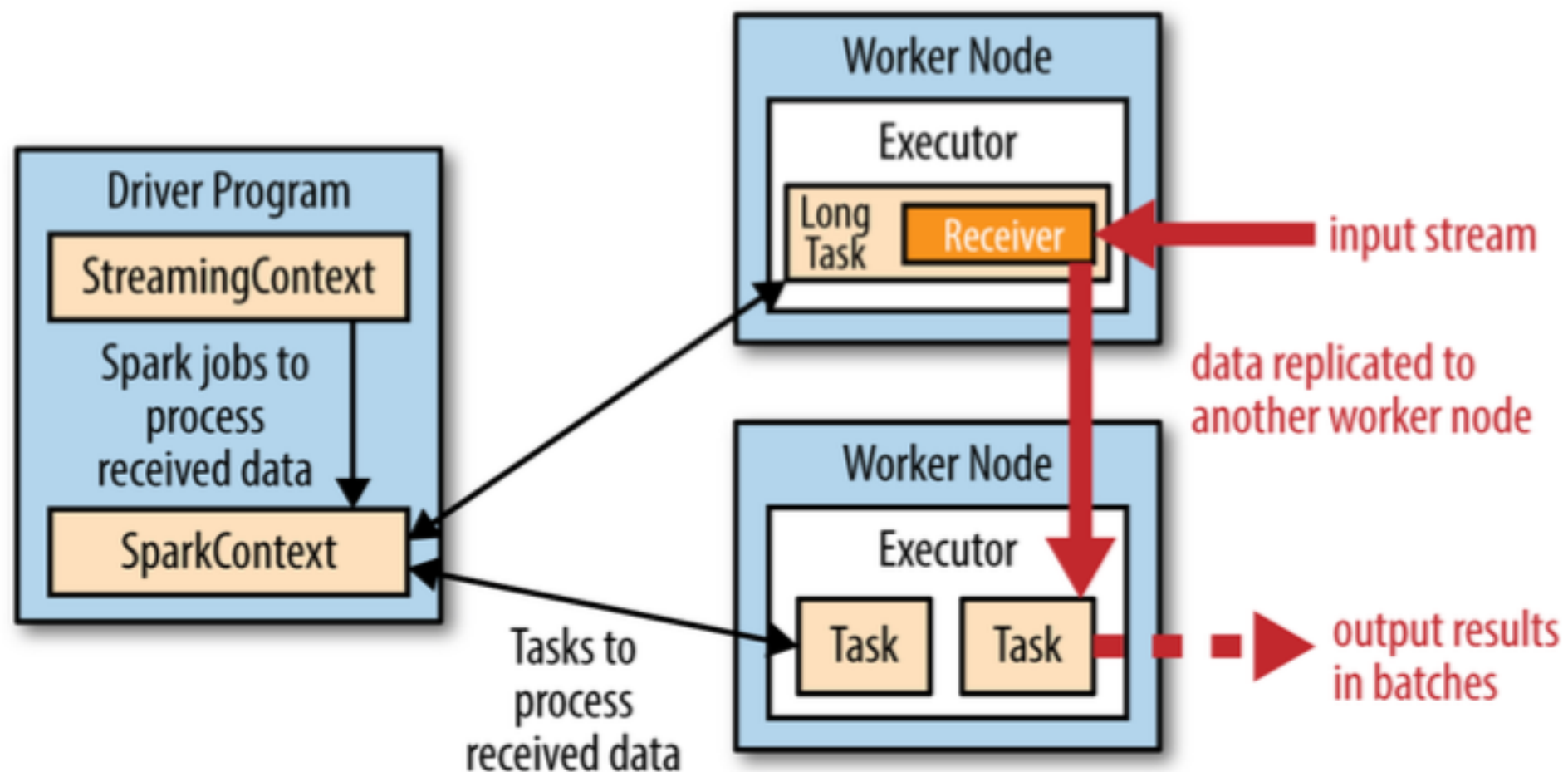
/Architecture



High-level architecture

# Architecture and Abstraction

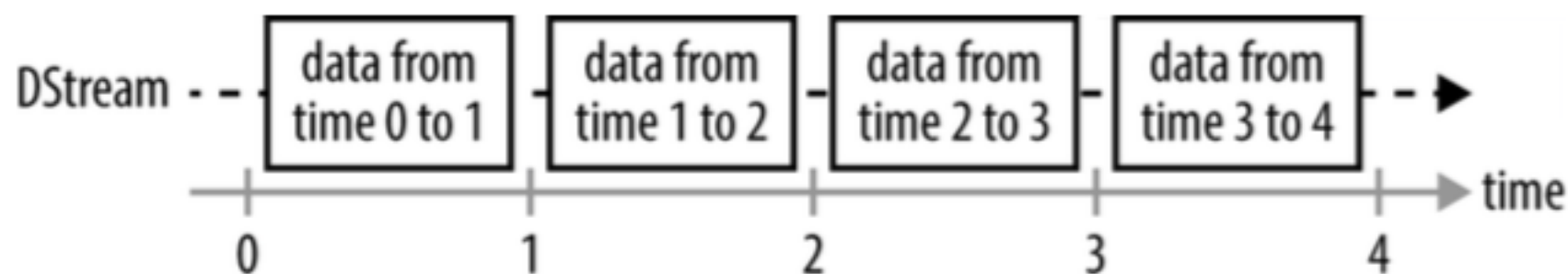
/Architecture



# 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

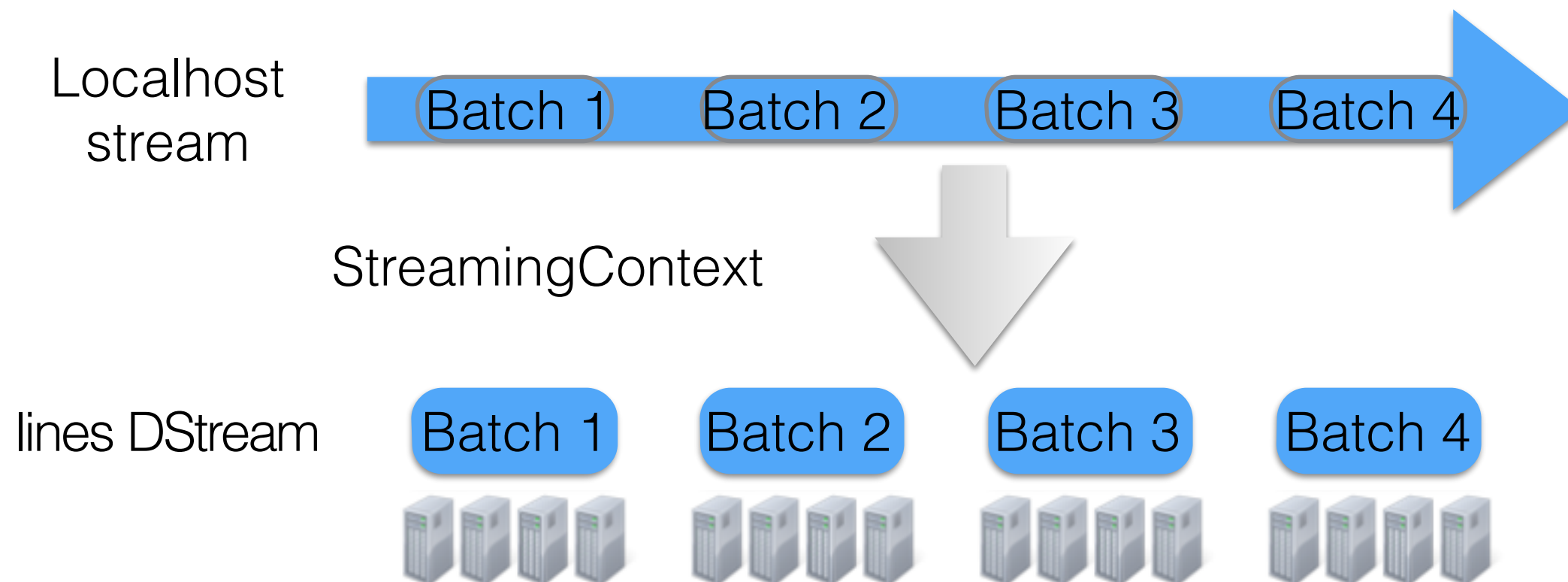




# Architecture and Abstraction

/Example

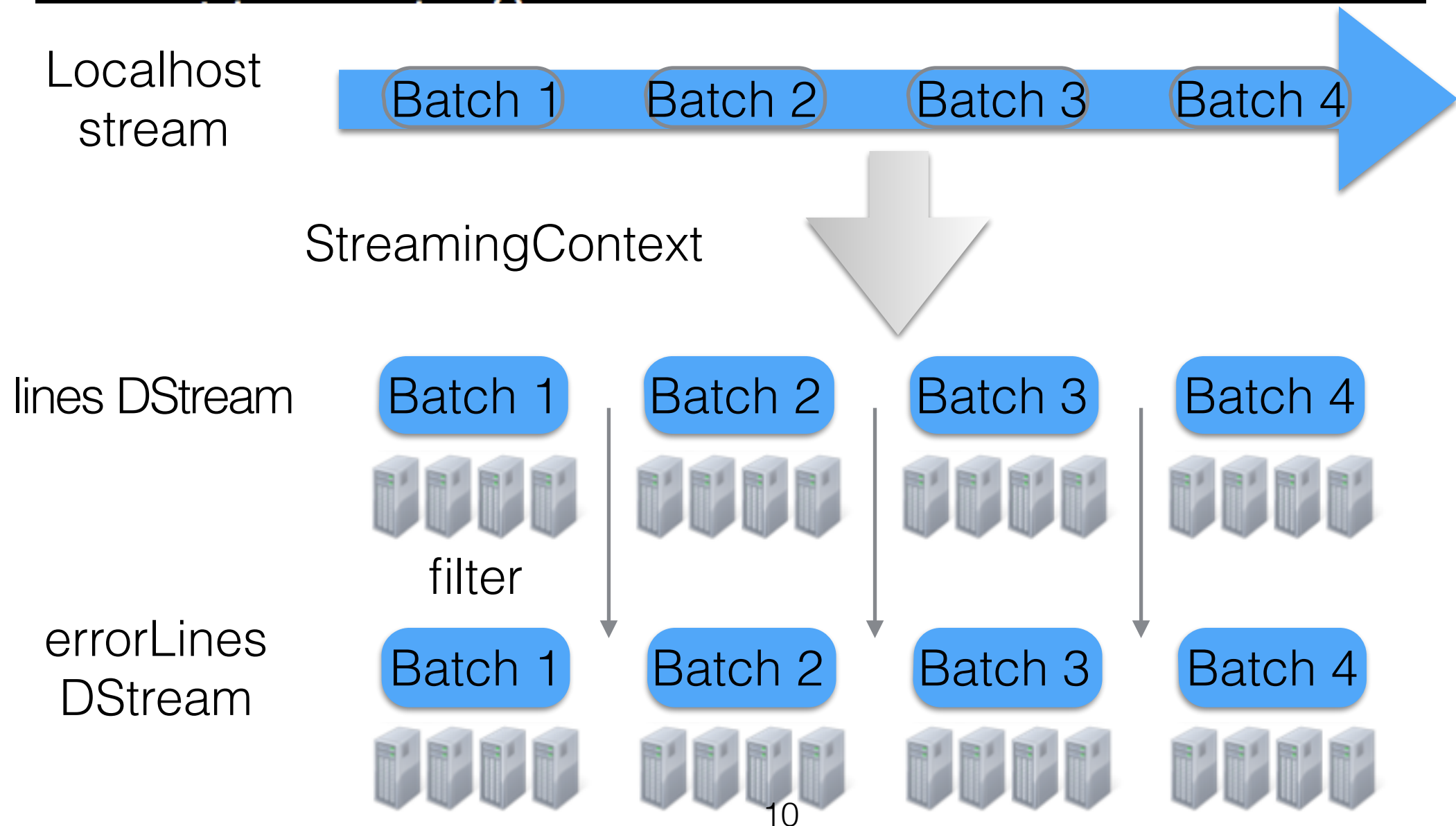
```
val ssc = new StreamingContext(sc, Seconds(10))  
val lines = ssc.socketTextStream("localhost", 9999)
```



# Architecture and Abstraction

/Example

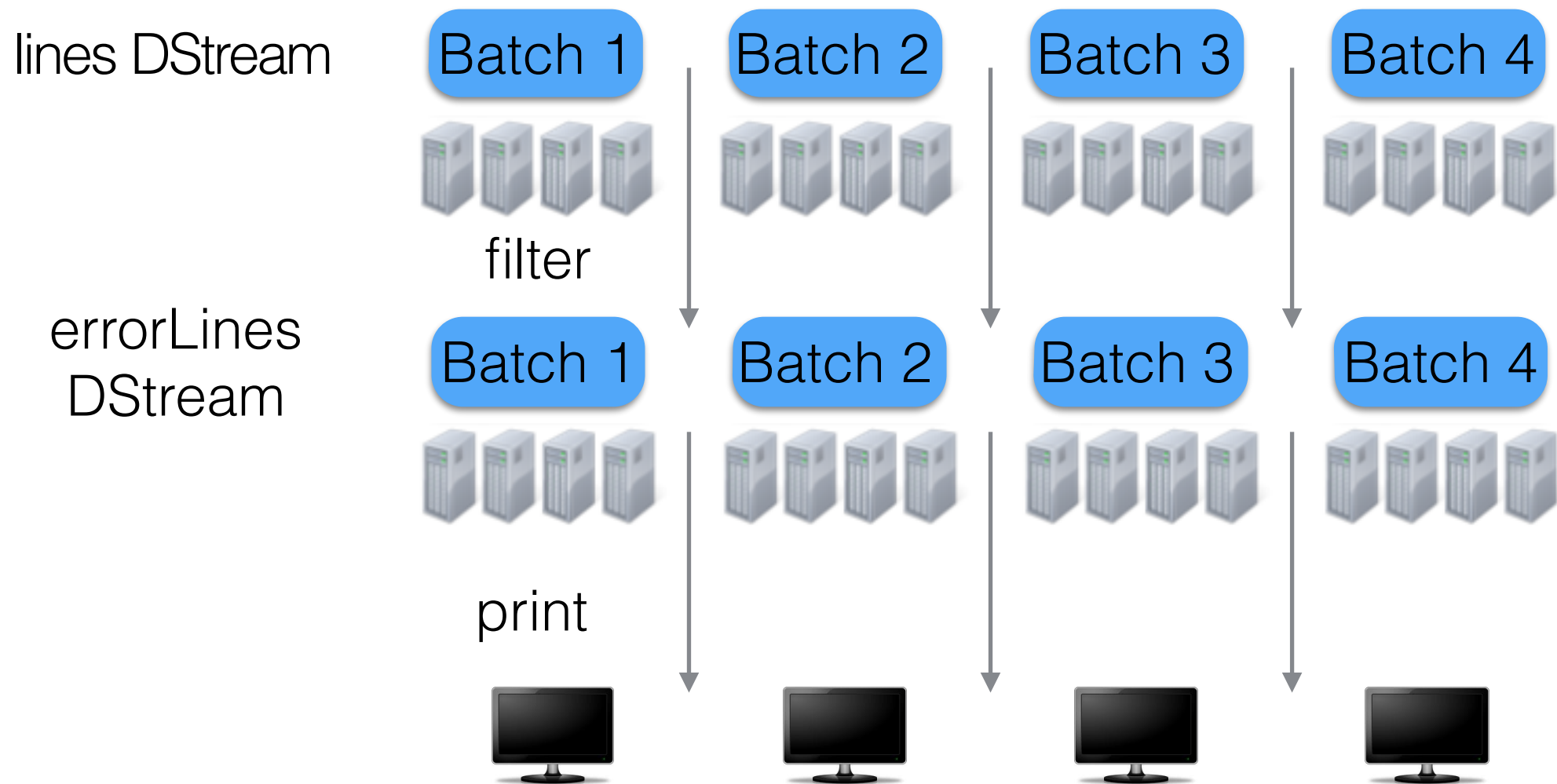
```
val ssc = new StreamingContext(sc, Seconds(10))  
val lines = ssc.socketTextStream("localhost", 9999)  
val errorLines = lines.filter(_.contains("error"))
```



# Architecture and Abstraction

/Example

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

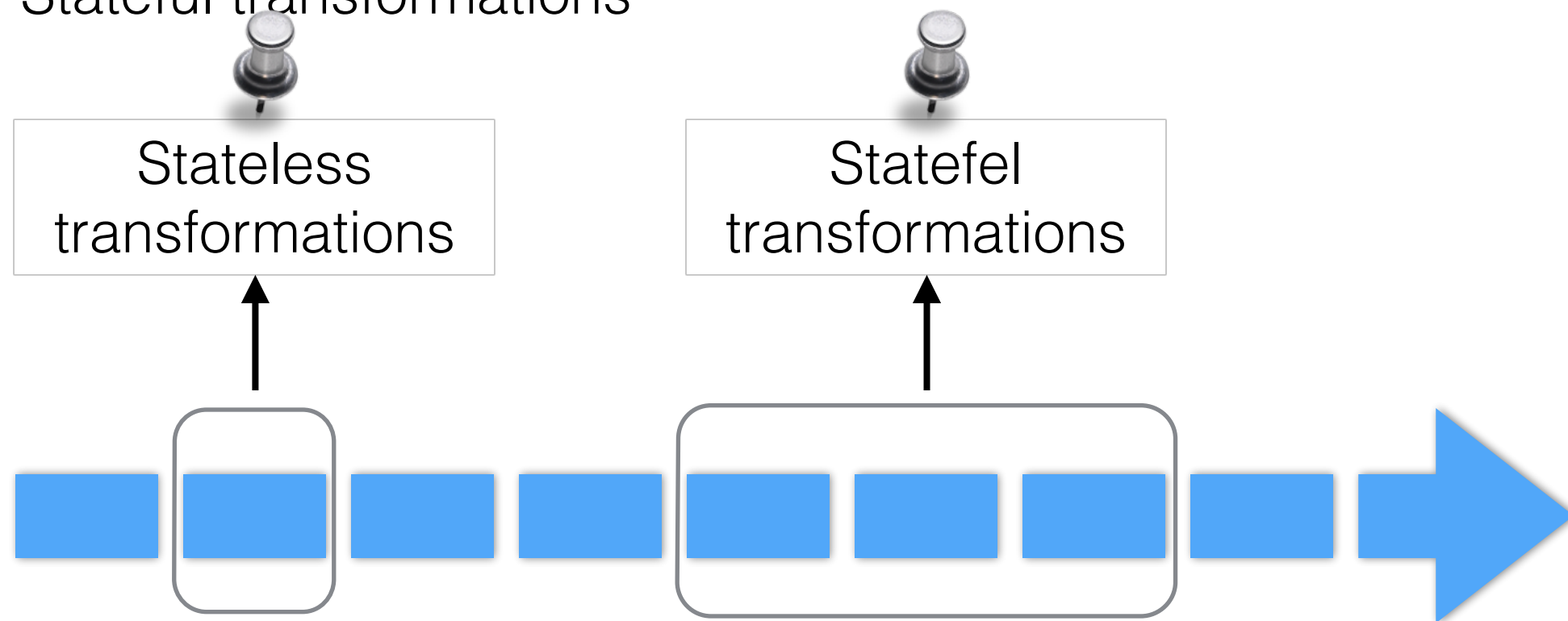




# Transformations and output operations

## /Transformations

- There are two types of transformation
  - Stateless transformations
  - 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

The screenshot shows the Spark Streaming UI for an application named 'IndexTweetsLive'. The browser address bar indicates the URL is 'localhost:4040/streaming/'. The UI has a navigation bar with tabs for 'Stages', 'Storage', 'Environment', 'Executors', and 'Streaming', with 'Streaming' currently selected. The main content area is titled 'Streaming' and displays the following information:

- Started at: Wed Oct 22 06:11:53 PDT 2014
- Time since start: 27 minutes 20 seconds
- Network receivers: 1
- Batch interval: 1 second
- Processed batches: 1641
- Waiting batches: 0

Below this, there is a section titled 'Statistics over last 100 processed batches' which includes a sub-section 'Receiver Statistics'.

Receiver	Status	Location	Records in last batch [2014/10/22 06:39:14]	Minimum rate [records/sec]	Median rate [records/sec]	Maximum rate [records/sec]	Last Error
TwitterReceiver-0	ACTIVE	localhost	39	0	61	151	-

Below the receiver statistics, there is a section titled 'Batch Processing Statistics'.

Metric	Last batch	Minimum	25th percentile	Median	75th percentile	Maximum
Processing Time	31 ms	5 ms	39 ms	56 ms	457 ms	2 seconds 289 ms
Scheduling Delay	0 ms	0 ms	0 ms	0 ms	1 ms	803 ms
Total Delay	31 ms	31 ms	40 ms	57 ms	499 ms	2 seconds 289 ms



# 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

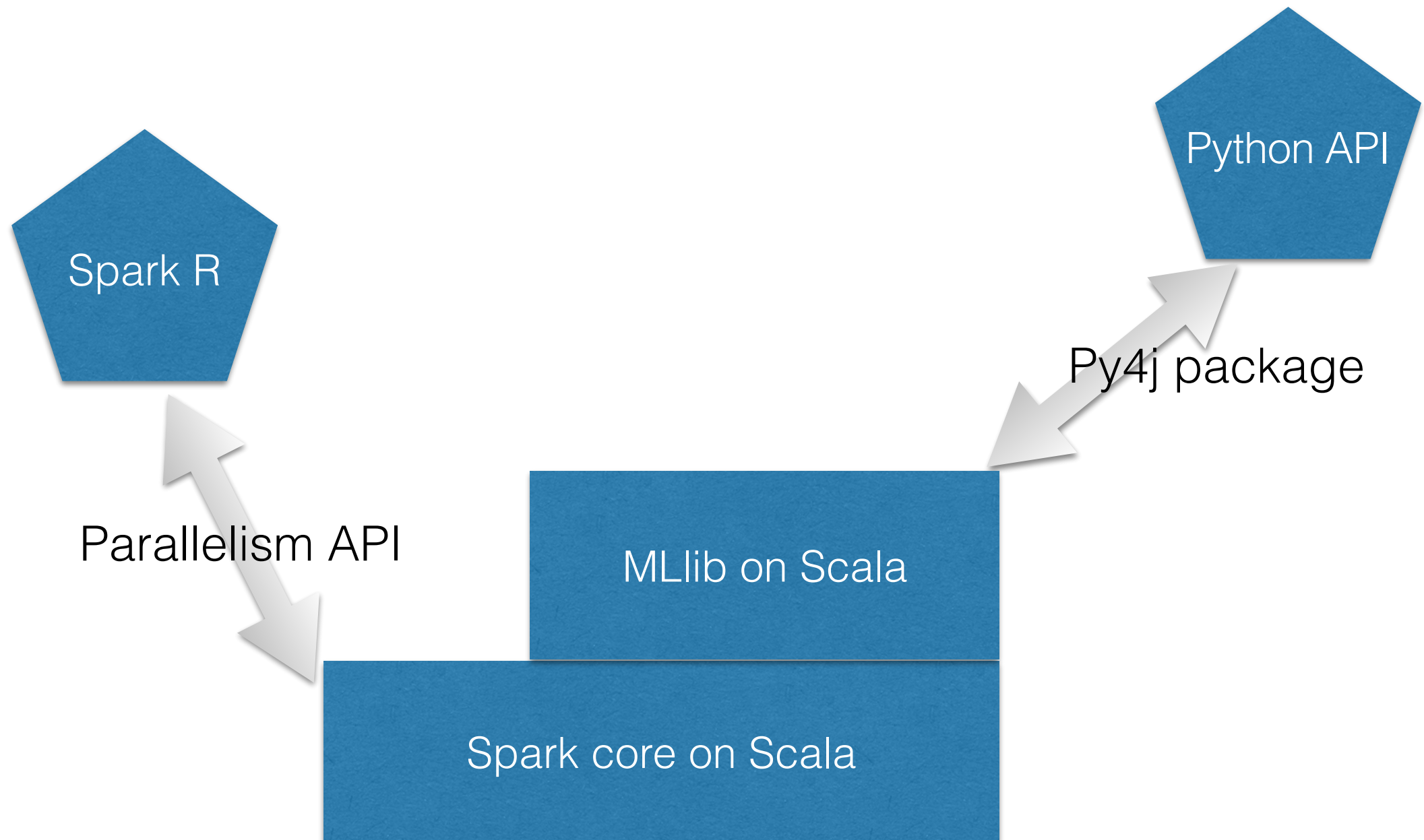
# Spark R

## /R vs. Python

- R has a project on parallel calculation: RHadoop
  - It is Open-source at the beginning
  - 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

