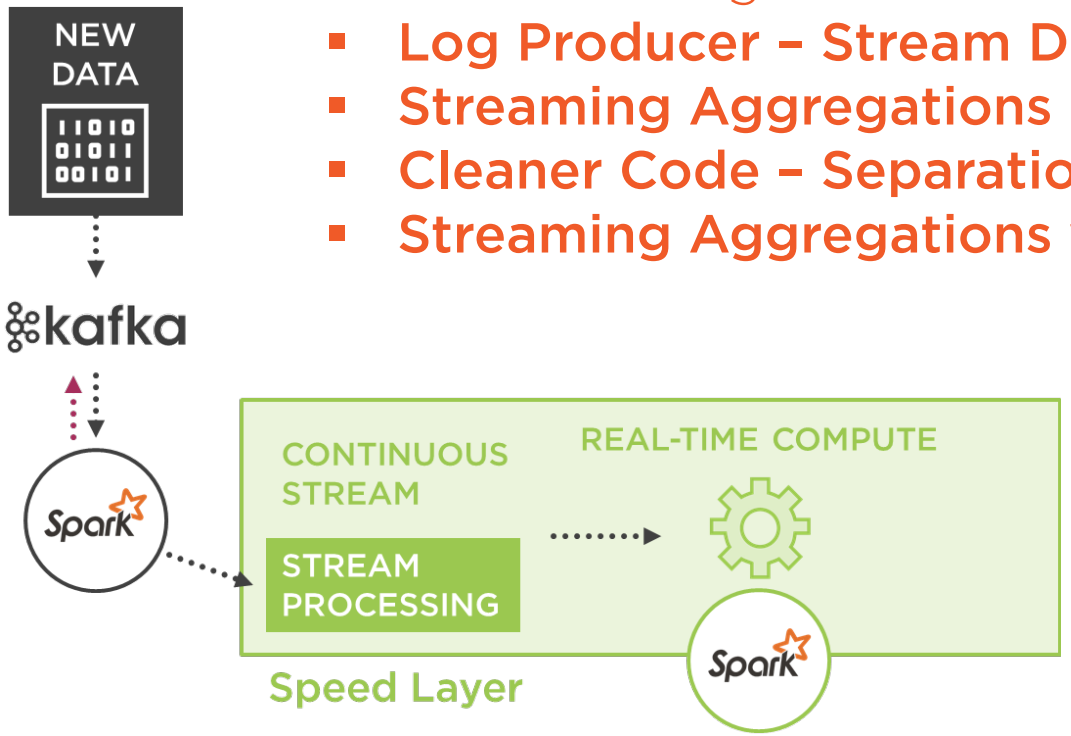
# Speed Layer with Spark Streaming

**Ahmad Alkilani**
DATA ARCHITECT

@akizl

# Speed Layer with Spark Streaming

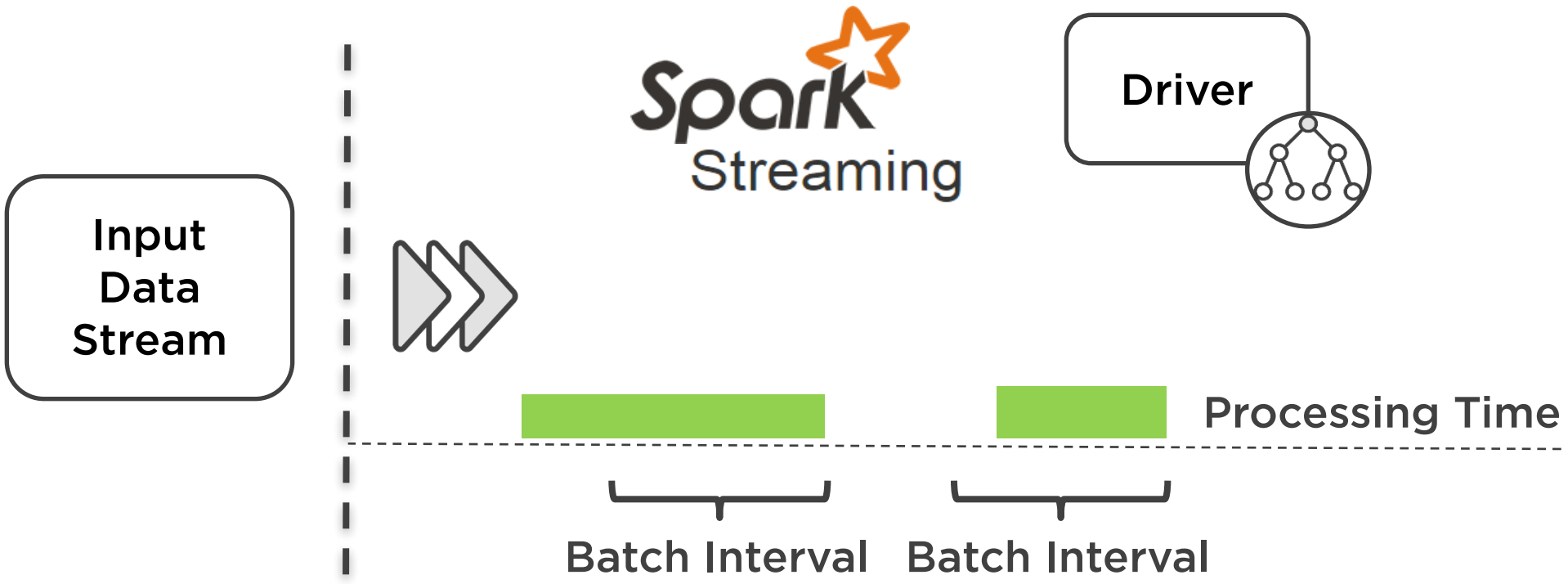
- **Spark Streaming Fundamentals**
  - SparkSQL and DataFrames with Spark Streaming
  - Streaming Receiver Model
- **Log Producer – Stream Data to Files**
- **Streaming Aggregations**
- **Cleaner Code – Separation of Concerns**
- **Streaming Aggregations with Apache Zeppelin**



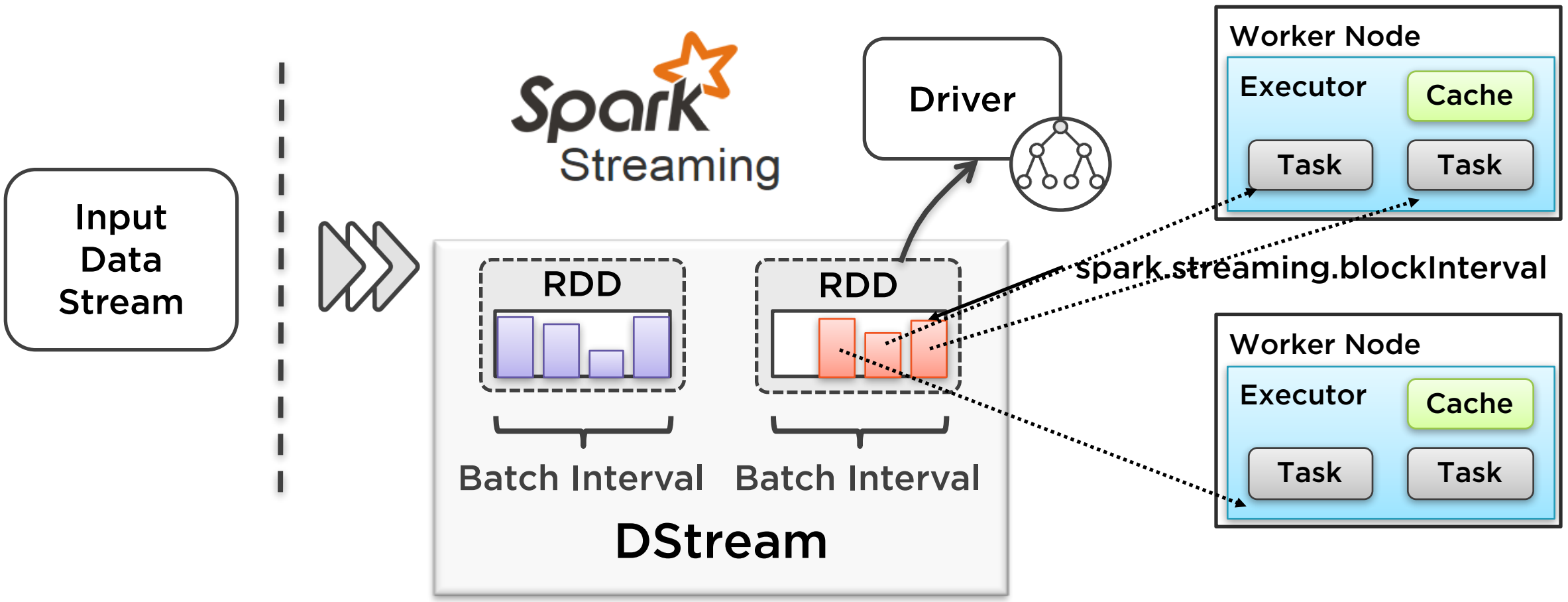

Extension of the core Spark API that enables building scalable, high-throughput and fault-tolerant streaming applications

# Spark Streaming

# Spark Streaming



# Spark Streaming



# Spark Streaming

| Spark Streaming | Others (Apache Storm etc.) |
| --- | --- |
| Moderate Latency | Single Record at a Time; Very Low Latency |
| Relies on RDDs (Delivery Guarantees) | Different Systems |
| Higher Throughput | Continuous Operator Model |
| Same Core as Batch | Different Systems for Batch and Streaming |
| Excellent for Lambda Architectures | Higher Total Cost of Ownership |

# DStream vs. RDD

DStream[T]

generatedRDDs : HashMap[Time, RDD[T]]

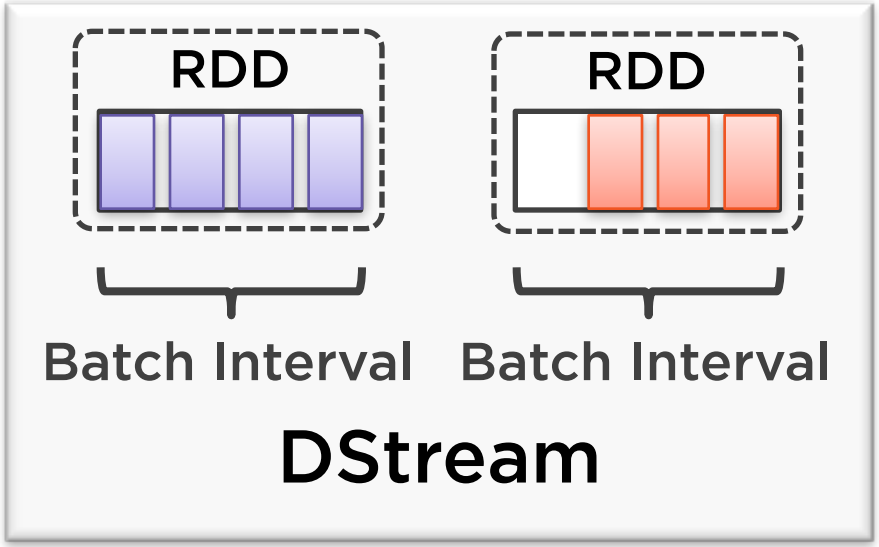
rememberDuration : Duration

Transformations:

- map, filter, reduce, union ..
- window, reduceByKeyAndWindow
- updateStateByKey, mapWithState
- transform

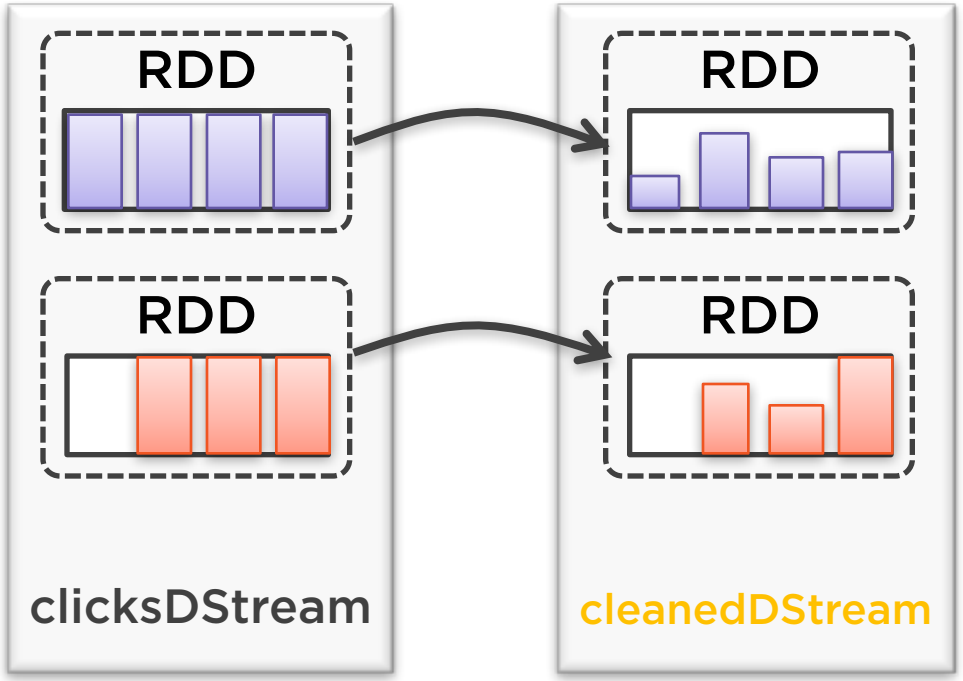
Output Operations/Actions:

- print, saveAsTextFiles, saveAsHadoopFiles
- foreachRDD



# DStream transform

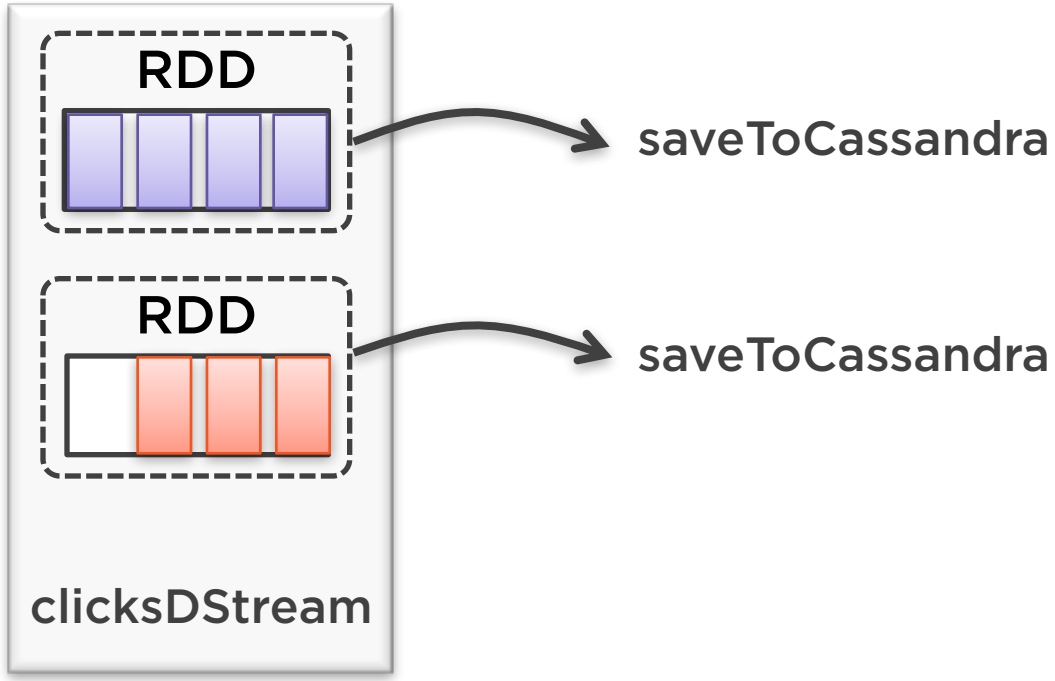
```
def transform[U](transformFunc: RDD[T] => RDD[U]): DStream[U]
```




```
val textStream : DStream[String] = ssc.textFileStream(..)
val personStream : DStream[Person] = textStream
          .transform( rdd => {
             rdd.map ( s => Person( .... ) )
          })

 filterDomainRDD = ...
 val cleanedDStream = clicksDStream
          .transform( rdd => {
             rdd.join(filterDomainRDD).filter(..)
          })
```

# DStream foreachRDD

```
def foreachRDD(foreachFunc: RDD[T] => Unit): Unit
```




```
clicksDStream.foreachRDD( rdd => {
   rdd.saveToCassandra(... , ...)
})
```

# DStream with DataFrames and SQL

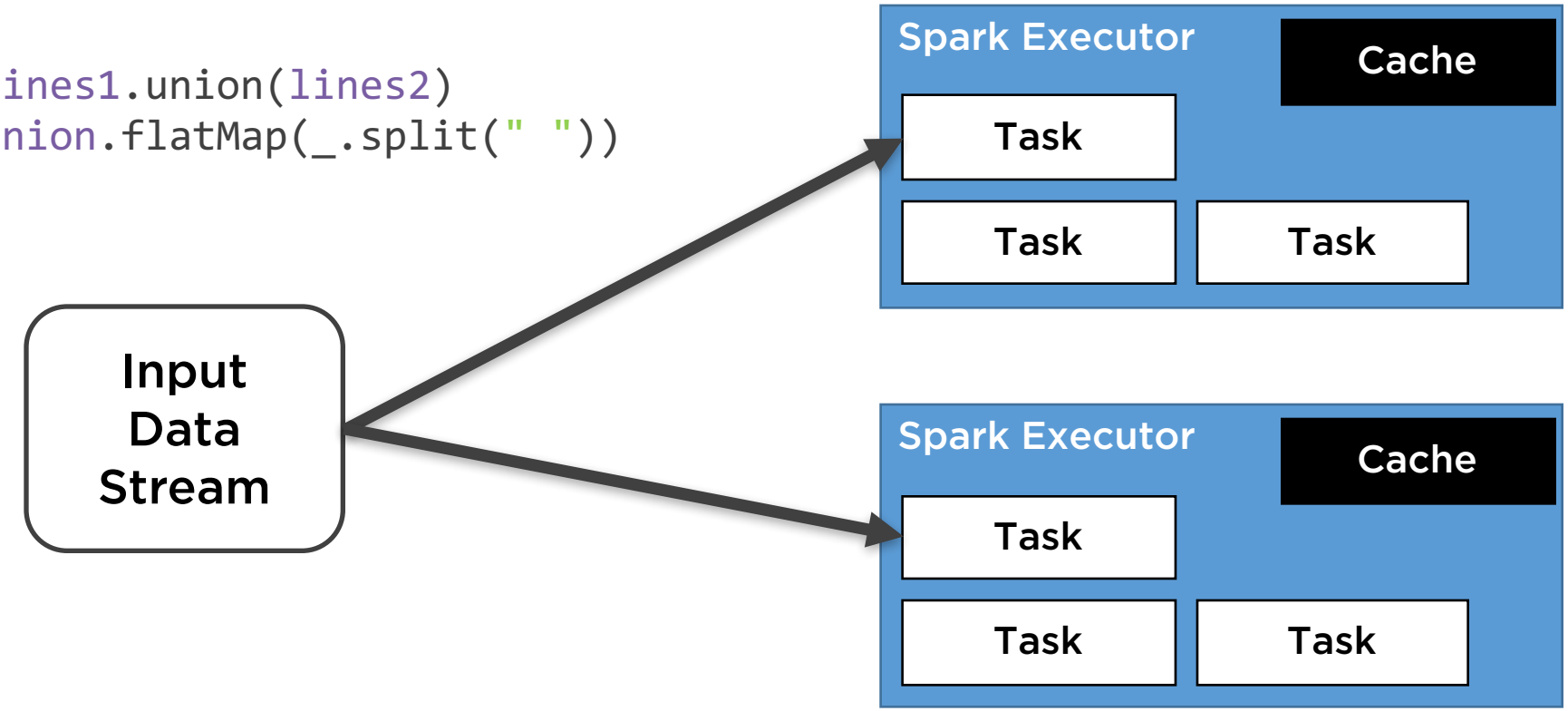
- Drop to RDD level with either .transform or .foreachRDD
- Use RDD API to get DataFrame or use SQLContext

```
val words: DStream[String] = ...

words.foreachRDD { rdd =>

  // Get the singleton instance of SQLContext
  val sqlContext = SQLContext.getOrCreate(rdd.sparkContext)
  import sqlContext.implicits._

  // Convert RDD[String] to DataFrame
  val wordsDataFrame = rdd.toDF("word")

  // Register as table
  wordsDataFrame.registerTempTable("words")

  // Do word count on DataFrame using SQL and print it
  val wordCountsDataFrame =
    sqlContext.sql("select word, count(*) as total from words group by word")
  wordCountsDataFrame.show()
}
```

Example source: http://spark.apache.org/docs/latest/streaming-programming-guide.html#dataframe-and-sql-operations

# Receiver Model

```
val lines1 = ssc.socketTextStream("localhost", 9999)
val lines2 = ssc.socketTextStream("localhost", 9998)

val linesUnion = lines1.union(lines2)
val words = linesUnion.flatMap(_.split(" "))
```




Spark Streaming

# Receiver Reliability

## Reliable Receivers

Use different techniques to achieve reliability including source stream acknowledgments, Spark checkpoints and Spark's write-ahead log

## Unreliable Receivers

Data loss very likely in case of receiver failure or application restarts.

# Summary

- **Spark Streaming Context**
- **DStream is just a bunch of RDDs**
  - transform `RDD[T] => RDD[U]`
  - foreachRDD `RDD[T] => Unit`
  - SparkSQL & DataFrames
- **Receiver Model**
- **Checkpoints**