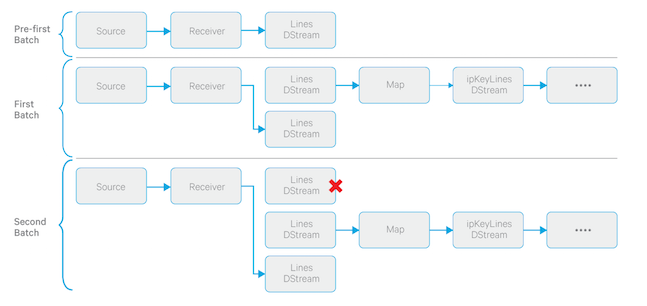
**Transformations on DStreams**

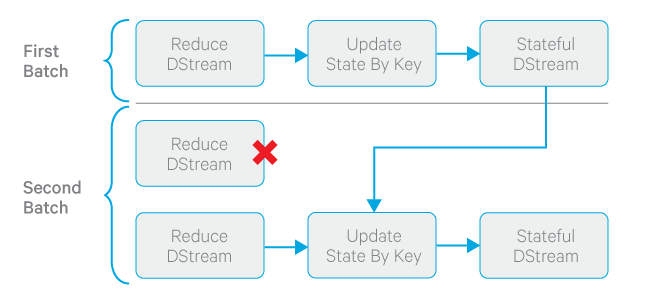
* All the transformations which we have for RDD exists and work on same way to Dstreams.
* Apart from that we have a new Transformations for Dstreams.

**UpdateStateByKey**

Upon receiving new data in Dstream the information about the previous data is lost.



But for statefullness, Spark Streaming have a trasformation UpdateStateByKey through which the previous state of the RDD is perserved.



**Usage:**

* To use this, we will have to fallow two steps
* Define the state - The state can be of 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 input stream.
* Suppose to say in NetworkWordCount example.

val lines = ssc.socketTextStream(args(0), args(1).toInt)

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

val wordDstream = words.map(x => (x, 1))

* The wordDstream will have the (key,value)🡺 (word,1) value

val stateDstream = wordDstream.updateStateByKey[Int](updateFunc)

* Now wordDstream is the state
* And updateFunc is the update function.

val updateFunc = (values: Seq[Int], state: Option[Int]) => {

val currentCount = values.foldLeft(0)(\_ + \_)

val previousCount = state.getOrElse(0)

Some(currentCount + previousCount) }

* In the updateFunc the state and the value for the key is passed.
* The command state.getOrElse(0)
* Will get the previous count for the key if present or else will get 0
* The previous count is added to present count and sent back.

References:

[1]: <http://blog.cloudera.com/blog/2014/11/how-to-do-near-real-time-sessionization-with-spark-streaming-and-apache-hadoop/>

[2]: <https://spark.apache.org/docs/1.1.0/streaming-programming-guide.html>