```
val rdd = sc.textFile("/word.txt")
rdd.flatMap(_.split(" ")).map((_,1)).groupByKey().map(t => (t._1, t._2.sum)).saveAsTextFile("/r1")
rdd.flatMap(_.split(" ")).map((_,1)).reduceByKey(_ + _).saveAsTextFile("/r2")
```

groupByKey和reduceByKey API的区别和相同点

1. reduceByKey API在shuffle write过程中，会对当前分区的数据进行combiner合并操作，最终形成的shuffle文件中，key对应的value是聚合之后的数据值；而groupByKey API不会进行combiner数据合并操作

2. 在shuffle fetch过程中，groupByKey API会将一个key对应的所有value数据全部加载到内存中，当一个key对应的value数据特别多的时候，在后续的处理过程中就有可能出现内存溢出的问题；但是reduceByKey不会将一个key的所有数据全部加载到内存中，而是使用reduceByKey API调用时候给定的聚合函数，对一个key的所有value数据进行一个聚合操作

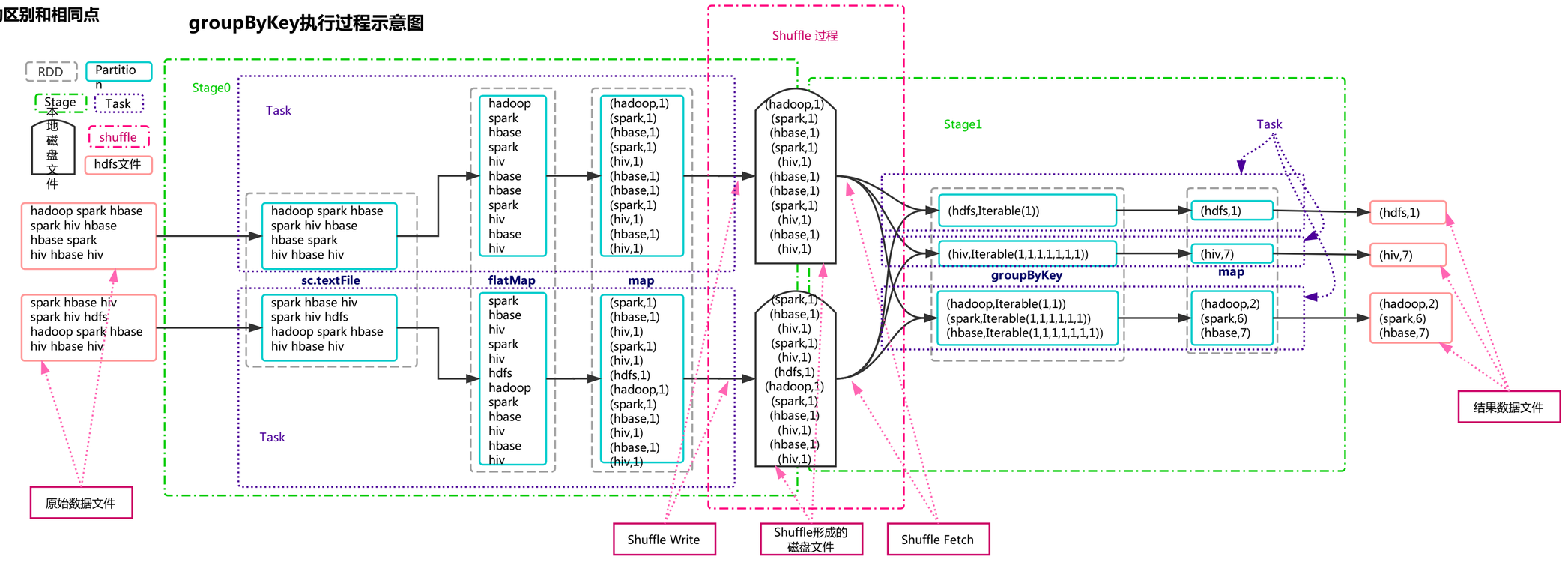
3. 两个API的底层都是调用RDD的combineByKeyWithClassTag方法

4. reduceByKey API调用要求key不能是数组类型；groupByKey默认数据分区器的情况下，key也不能使数组类型

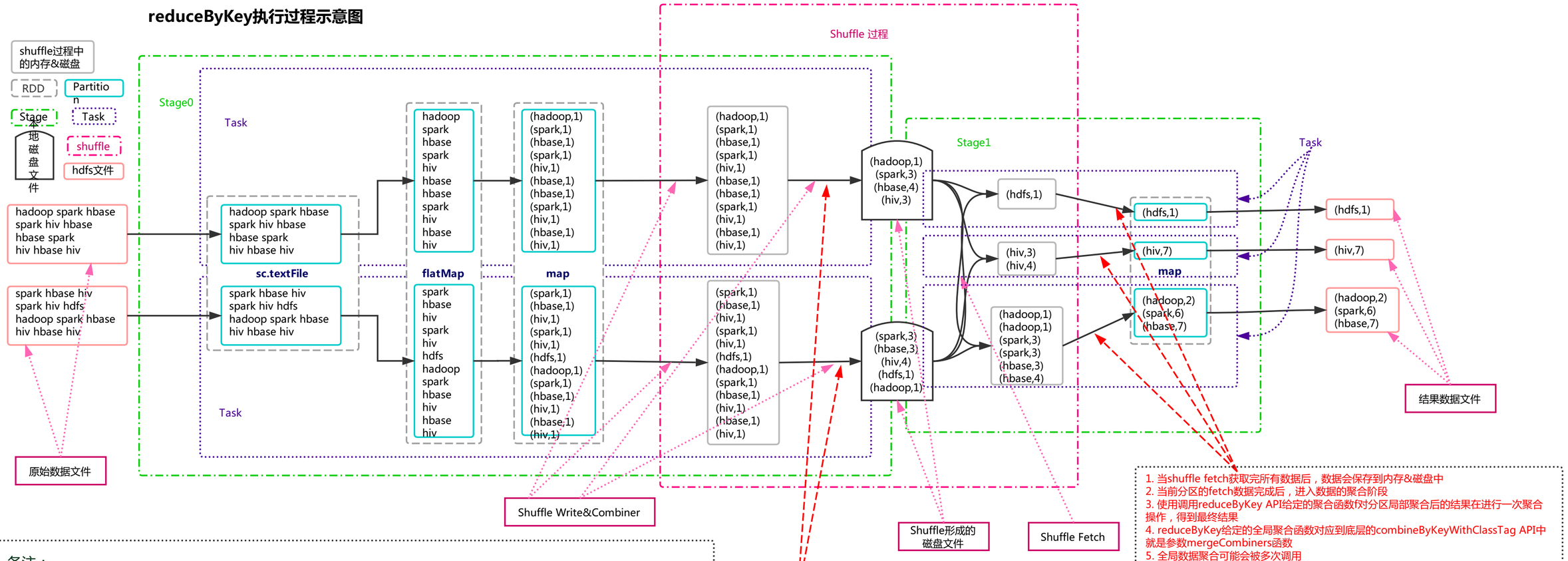
5. 两个API调用过程中均可以给定分区数(构建HashPartitioner数据分区器)或者数据分区器；当调用API的时候没有给定分区数量或者没有给定数据分区器的情况下，采用默认数据分区器，默认分区器构造规则如下：

a. 当父RDD中存在数据分区器的情况下，默认数据分区器其实就是父RDD中分区数量最大的那个RDD对应的数据分区器，子RDD的分区数目就是对应数据分区器中的分区数。
b. 当父RDD中不存在数据分区器的情况下，此时默认数据分区器是一个HashPartitioner；子RDD的分区数目由参数spark.default.parallelism决定，当SparkConf中配置该参数，那么子RDD的分区数目就是该参数对应的配置value值，否则子RDD的分区数目等于最大的父RDD的分区数。

groupByKey执行过程示意图



reduceByKey执行过程示意图



备注：
groupByKey、reduceByKey、aggregateByKey底层调用的API均是combineByKeyWithClassTag API

groupByKey API中不进行map端的combiner合并操作；reduceByKey和aggregateByKey API均进行combiner局部数据聚合操作

reduceByKey API中使用给定的聚合函数进行combiner局部聚合和shuffle之后的全局数据聚合

aggregateByKey API中使用给定的seqOp函数进行combiner局部聚合，使用给定的combOp函数进行shuffle之后的全局数据聚合