**What is reduceByKey?**

reduceByKey groups values by key and aggregates them using a function (like sum, count, max). It does local aggregation (combiner) on each partition before shuffling.

Example:

python

CopyEdit

rdd = sc.parallelize([('A',1), ('A',2), ('B',3)])

rdd.reduceByKey(lambda x,y: x+y).collect()

# Output: [('A',3), ('B',3)]

**What is groupByKey?**

groupByKey groups values by key and creates a collection of all values for each key.

➡ Example:

python

CopyEdit

rdd.groupByKey().collect()

# Output: [('A', [1,2]), ('B', [3])]

You would then need to manually aggregate if required (e.g., sum the list).

**Key difference in how they work**

| **Feature** | **reduceByKey** | **groupByKey** |
| --- | --- | --- |
| What it does | Groups + aggregates (e.g., sums) in one step | Groups values only |
| Local combine | Combines locally before shuffle | No local combine; all values shuffled |
| Shuffle data size | Small — only partial sums are shuffled | ⚠ Large — all individual values are shuffled |
| Performance | Faster + more memory efficient | Slower + high memory usage on reducer |

**How reduceByKey optimizes Spark**

**Reduces shuffle size**: By aggregating locally, much less data crosses the network.  
 **Uses memory better**: Reducer nodes don’t have to hold all raw values.  
 **Faster job completion**: Because of smaller shuffle and lower memory footprint.

Example:  
Suppose we have:

pgsql

CopyEdit

Partition 1: (A,1), (A,2) → combined to (A,3)

Partition 2: (A,4), (A,5) → combined to (A,9)

Only (A,3) and (A,9) are shuffled (not all individual values).

With groupByKey, Spark would shuffle:

css

CopyEdit

(A,1), (A,2), (A,4), (A,5)

Bigger shuffle, slower job.

**Summary**

|  | **reduceByKey** | **groupByKey** |
| --- | --- | --- |
| Combines values? | Yes (with user function) | No |
| Shuffle data size | Small | Large |
| Memory efficient? | Yes | No |
| Use case | When aggregation is needed | When you really want raw groups (rare in practice) |