Spark Core: Transformations

transformations defined + perform action → spark finds the best way to execute this computation and then start all the necessary tasks in our worker nodes

when we apply a transformation, nothing happens right away. we are basically preparing our big data pipeline to be executed later

programming model:
- RDDs get generated from external datasets + gets partitioned

- RDDs are inmutable: they can't be changed in place even partially
- they need transformation operation applied and get converted into new RDD

* essential for keeping track of all the processing that has been applied to dataset
↳ providing the ability to keep a linear chain of RDDs

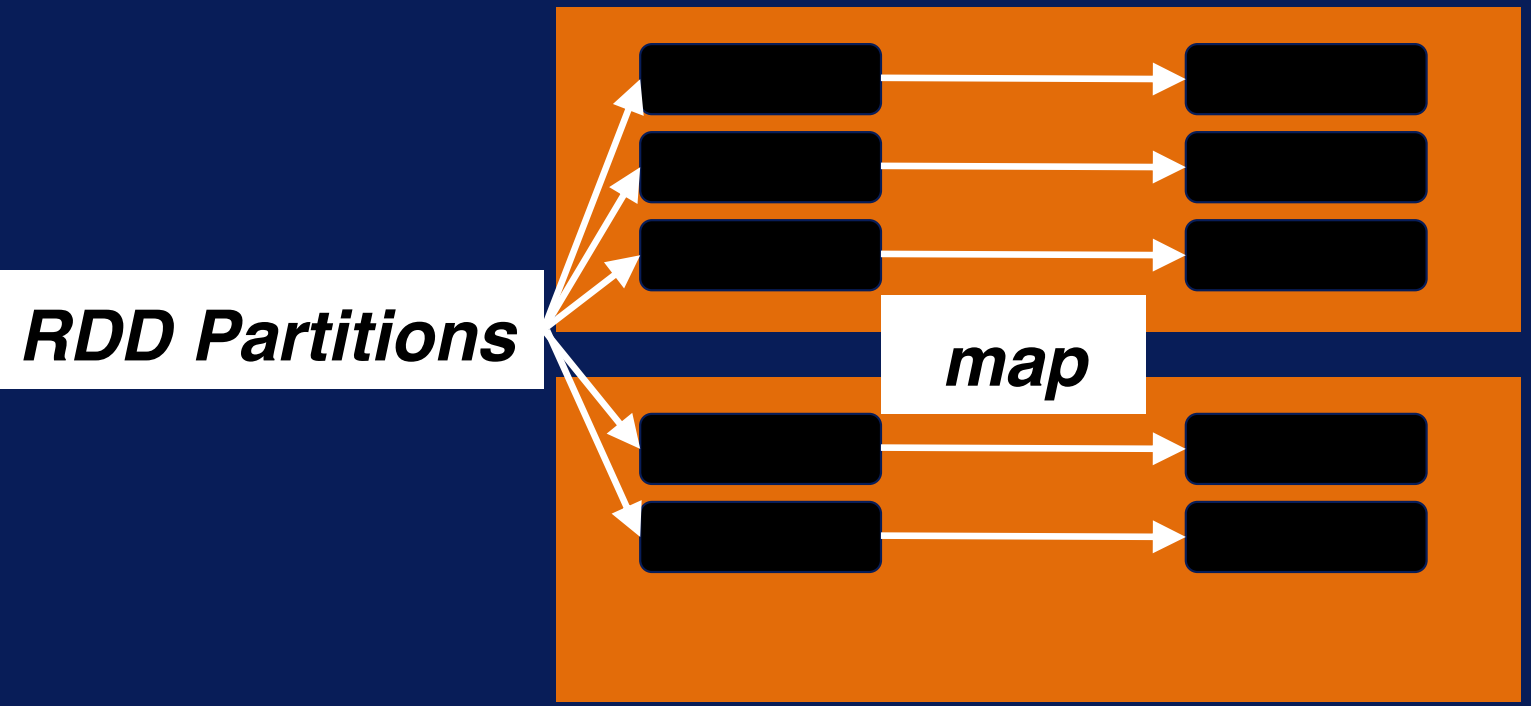
** all these transformations are lazy:
they don't execute immediately when applied

After this video you will be able to..

- Explain the difference between a narrow transformation and wide transformation
- Describe map, flatmap, filter and coalesce as narrow transformations
- List two wide transformations

map

map : apply function to each element of RDD

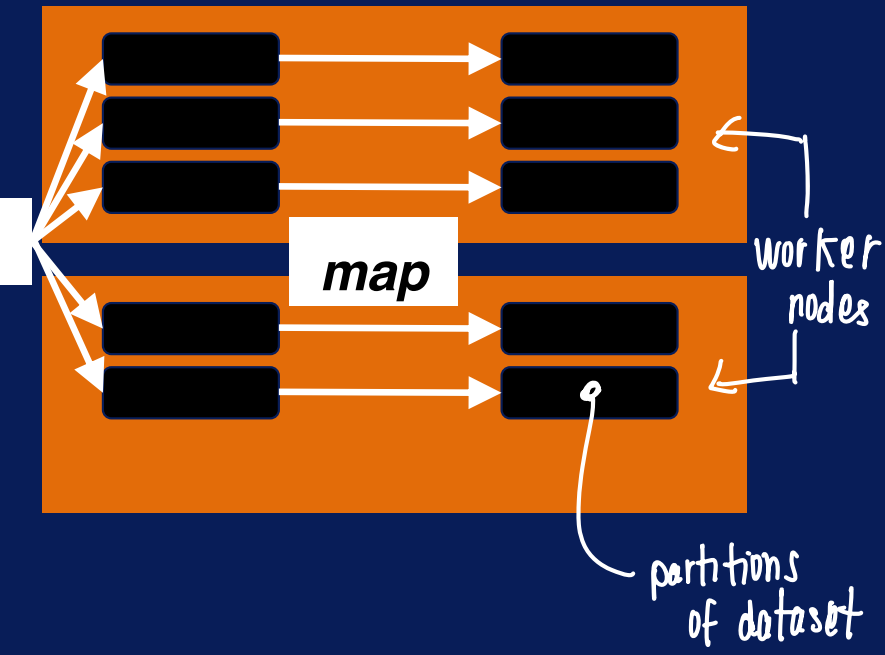


- one to one
- element wise transformations

map

map : apply function to each element of RDD

in partition in each worker node locally

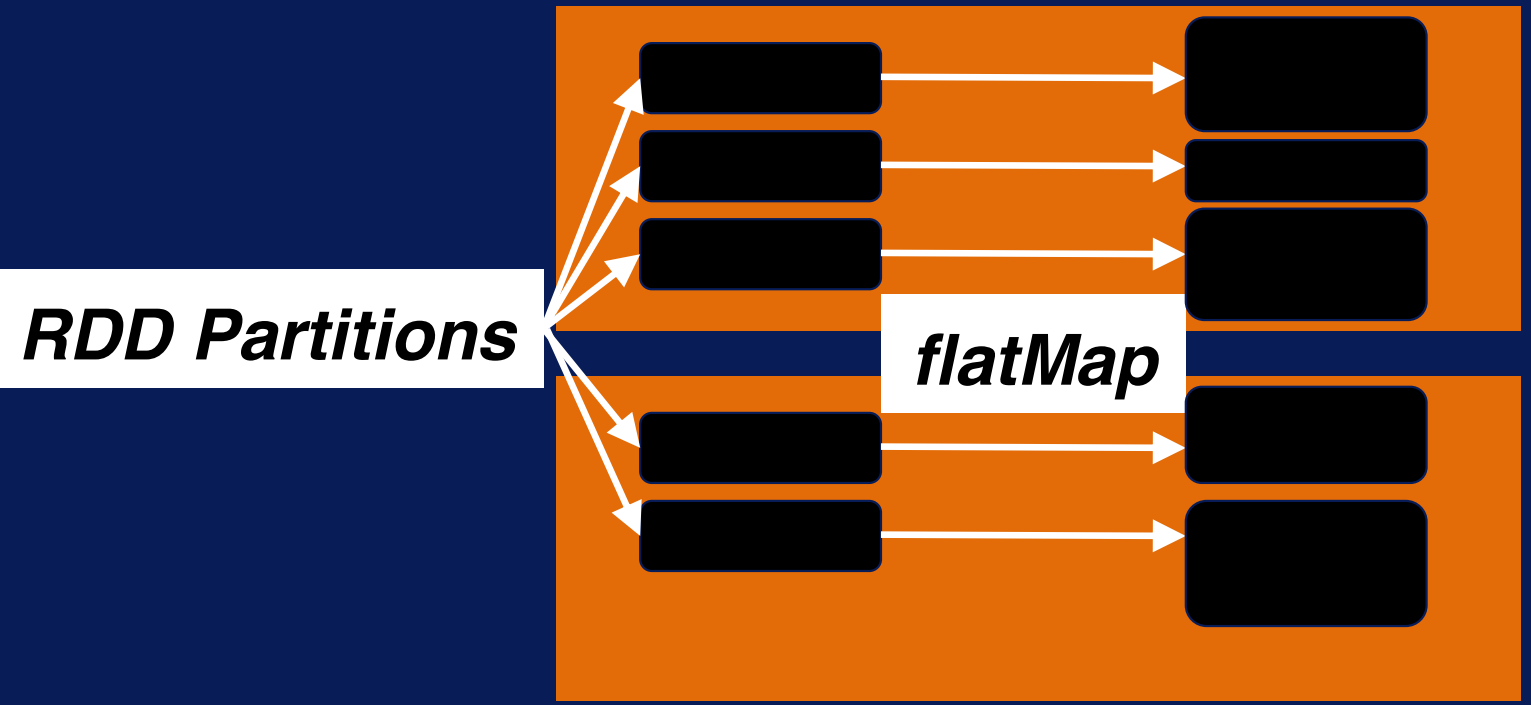



```
def lower(line):
    return line.lower()
lower_text_RDD = text_RDD.map(lower)
```

* we work by partition, not by element

flatMap

flatMap : map then flatten output

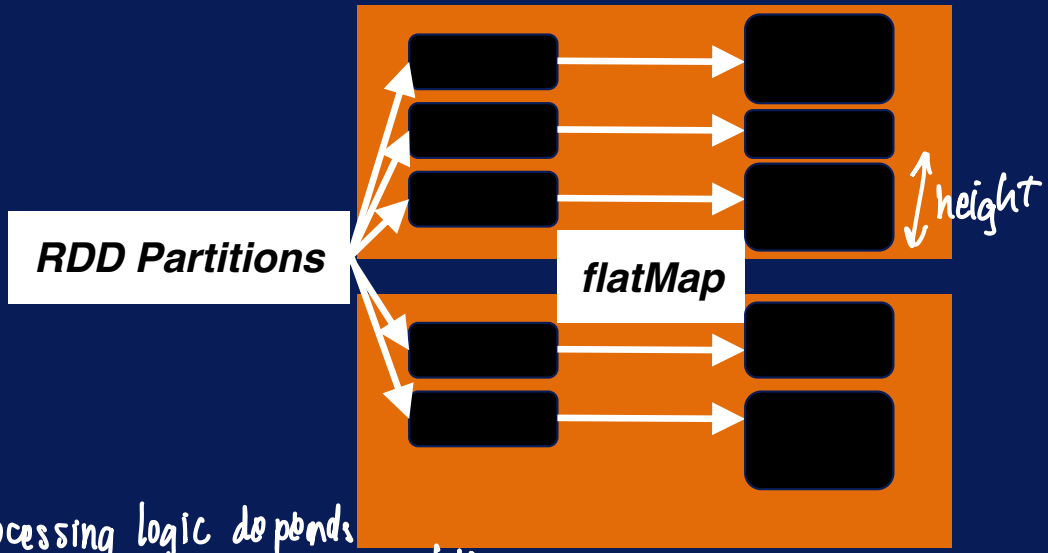


instead of returning an individual element for each map,

it returns an RDD with an aggregate of all the results for all the elements

flatMap

flatMap : map then flatten output



map + flatmap are narrow transformations : processing logic depends only on data that is already residing in partition
* data shuffling is not necessary

```
def split_words(line):
    return line.split()

words_RDD = text_RDD.flatMap(split_words)
words_RDD.collect()
```

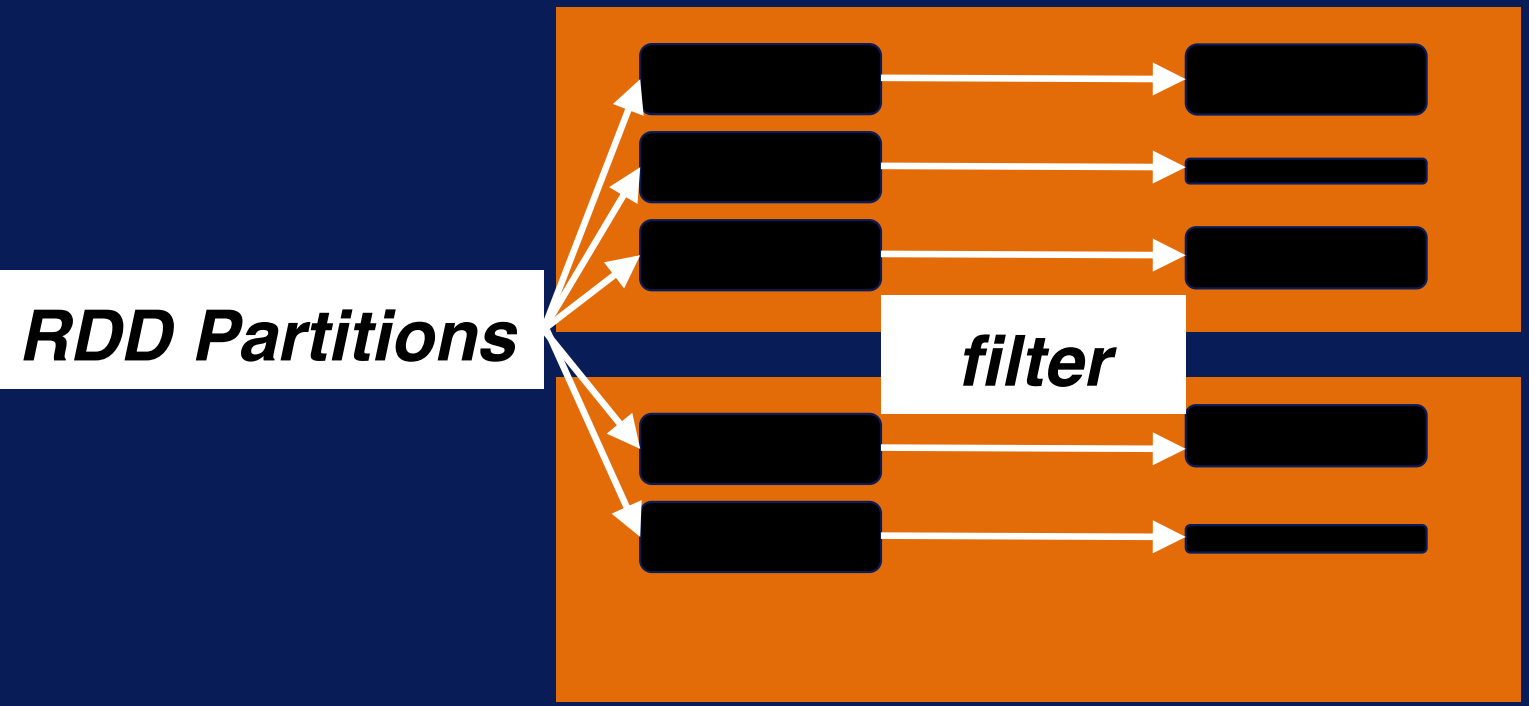
input: line (one element)
↓
split
↓
output : each word as single elem

flat : get simple one-dimensional list of words

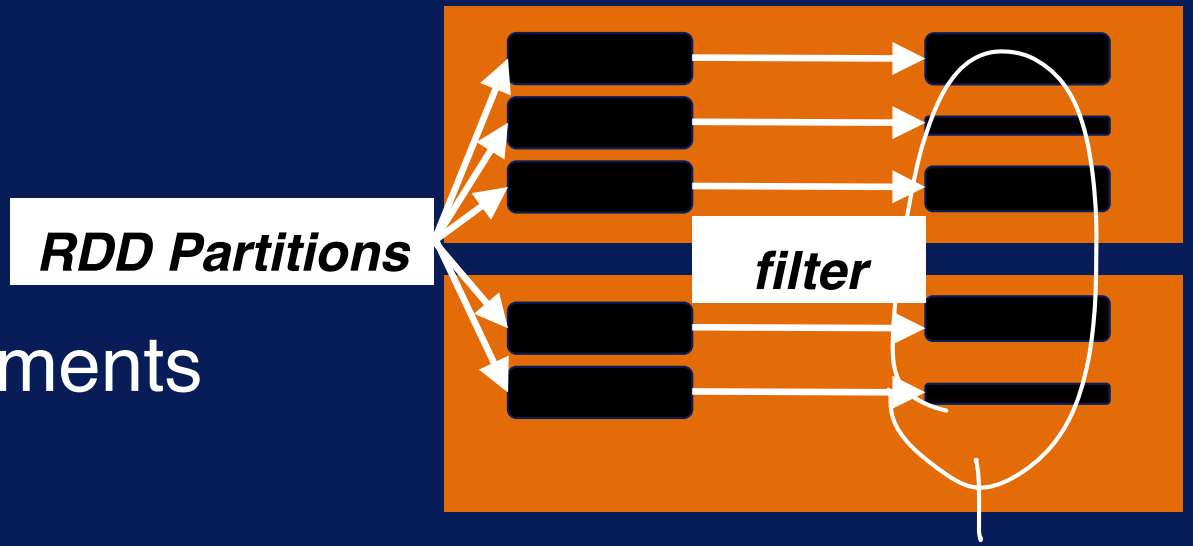
filter

subset of data
get rid of bad data

filter : keep only elements where function is true



filter



filter : keep only elements

where function is true

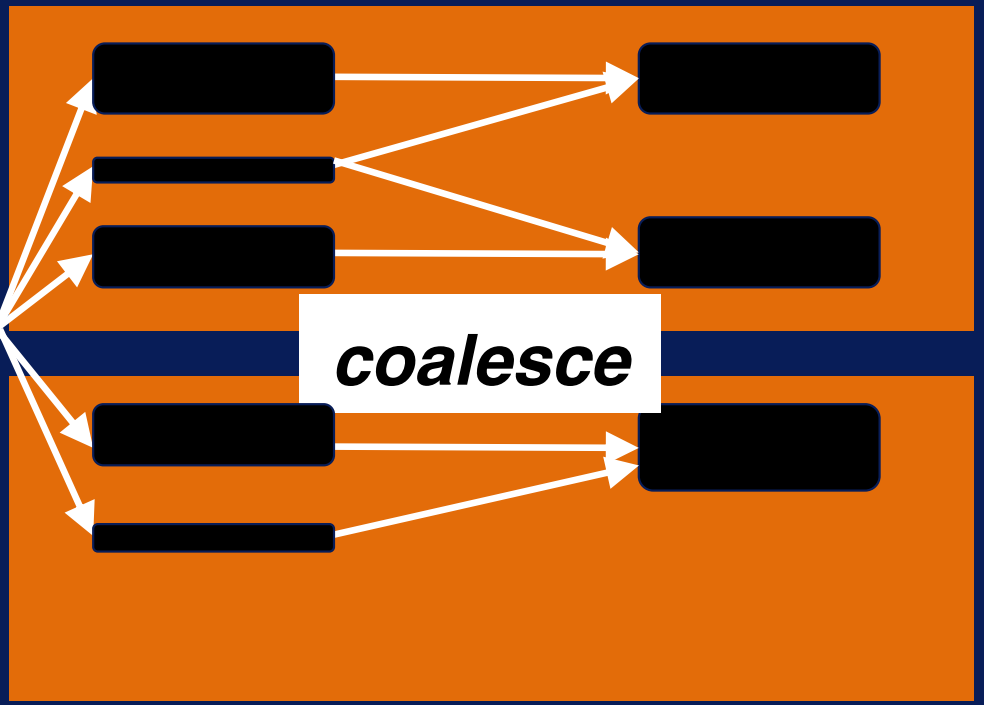
```
def starts_with_a(word):
    return word.lower().startswith("a")
words_RDD.filter(starts_with_a).collect()
```

join partitions to increase performance and even out processing across clusters

* also narrow : only gets executed locally without the need to shuffle RDD partitions across worker kernels

coalesce

coalesce : reduce the number of partitions

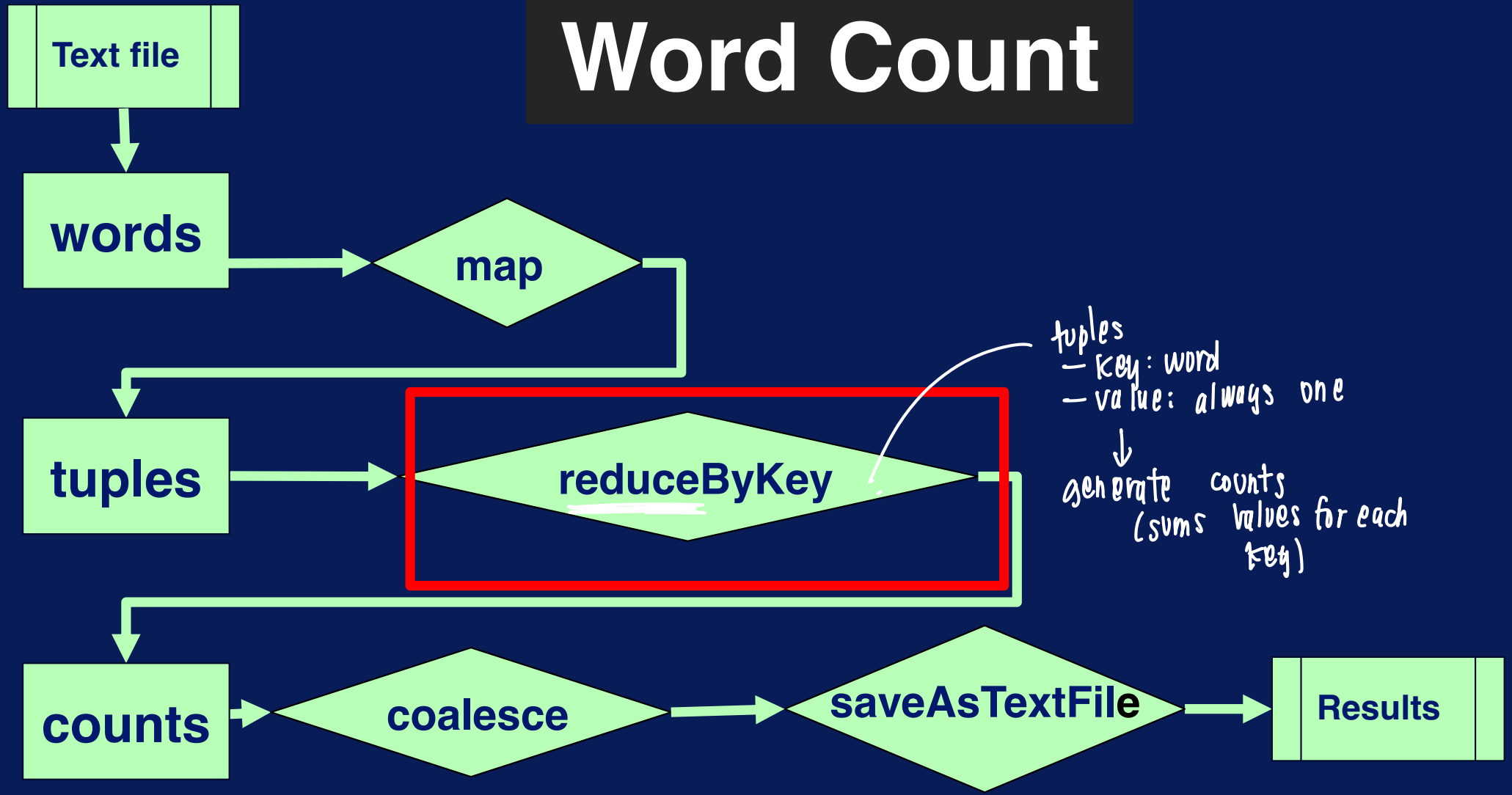


also narrow :
happen in a worker node locally without having to transfer data through the network

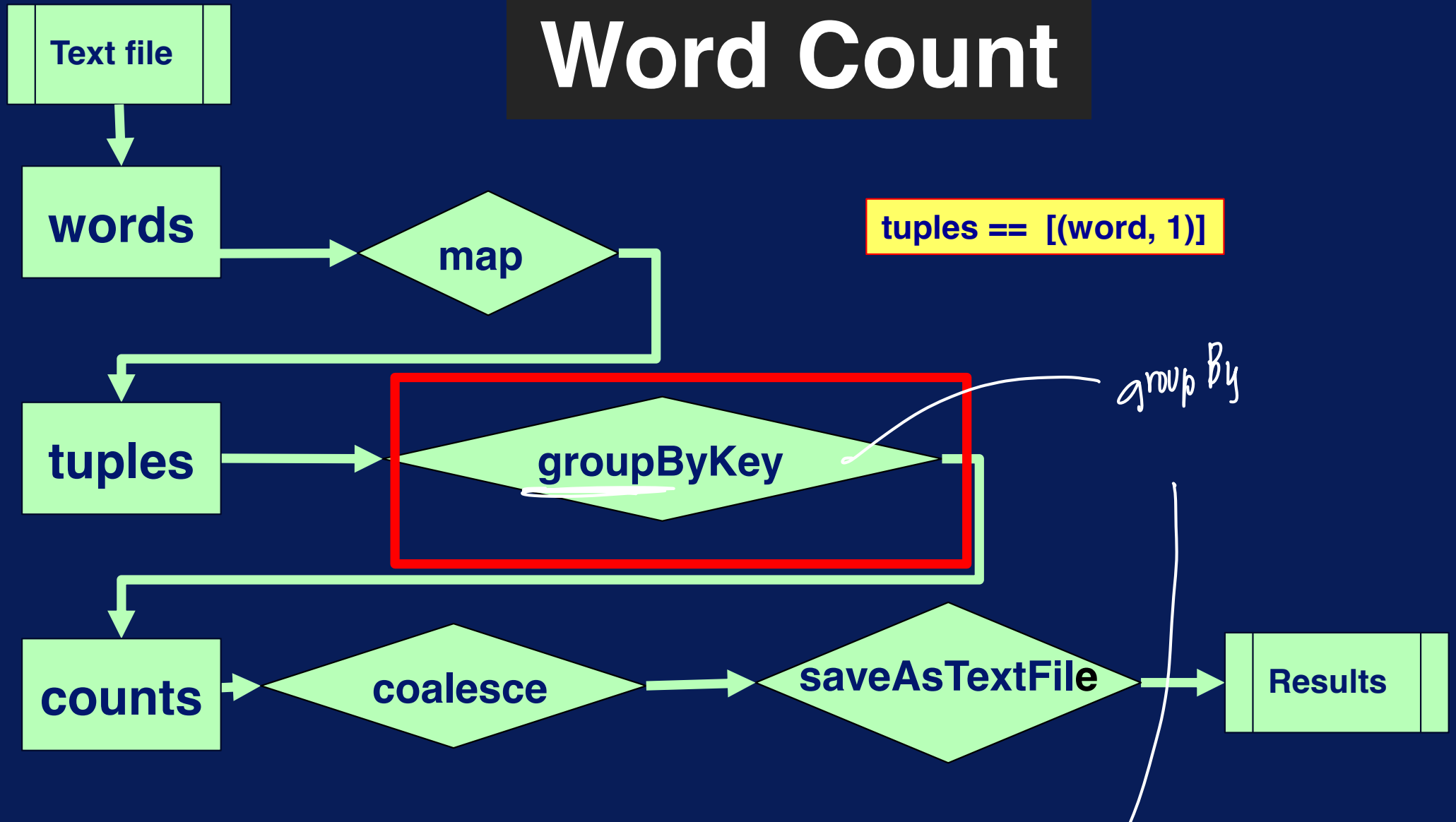
- balances data partition numbers and sizes
- when you reduced your initial data after filters and transformations, having a large number of partitions might not be useful anymore ↓ coalesce (to) reduce number of partitions to manageable number

Wide Transformations

Word Count

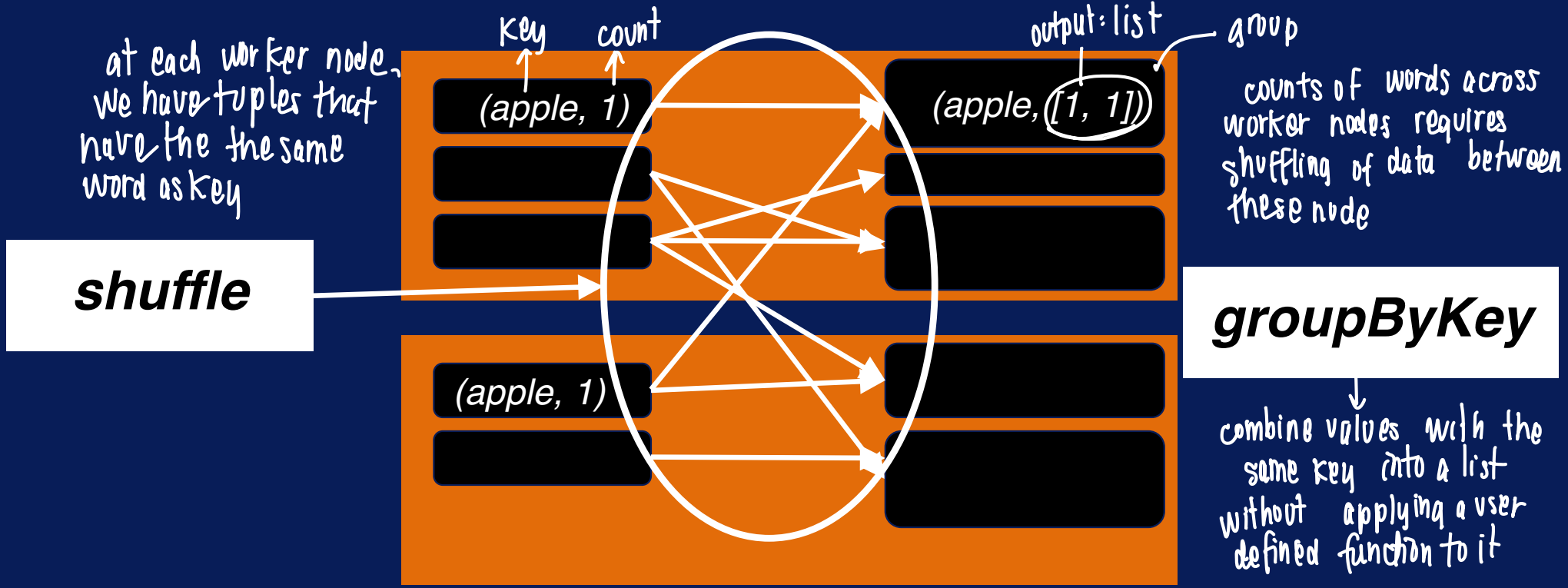

Text file
words
map
tuples
reduceByKey
tuples
– key: word
– value: always one
generate counts (sums values for each key)
counts
coalesce
saveAsTextFile
Results

Word Count


Text file
words
map
tuples == [(word, 1)]
tuples
groupByKey
group By
counts
coalesce
saveAsTextFile
Results

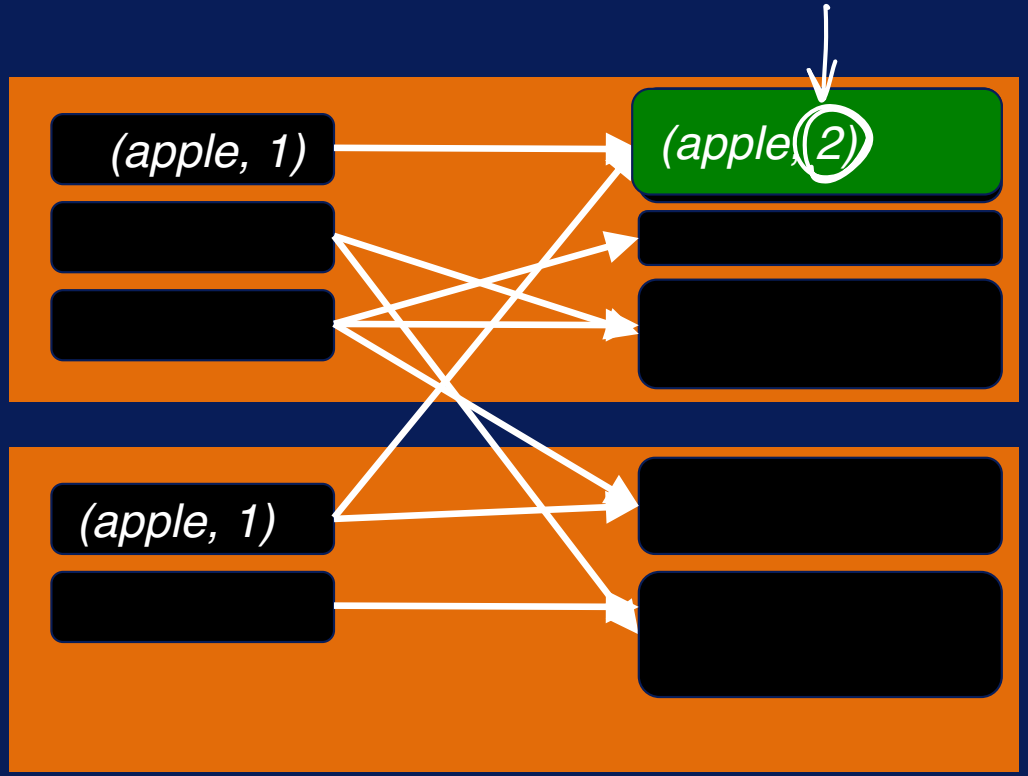
groupByKey

groupByKey : (K, V) pairs => (K, list of all V)



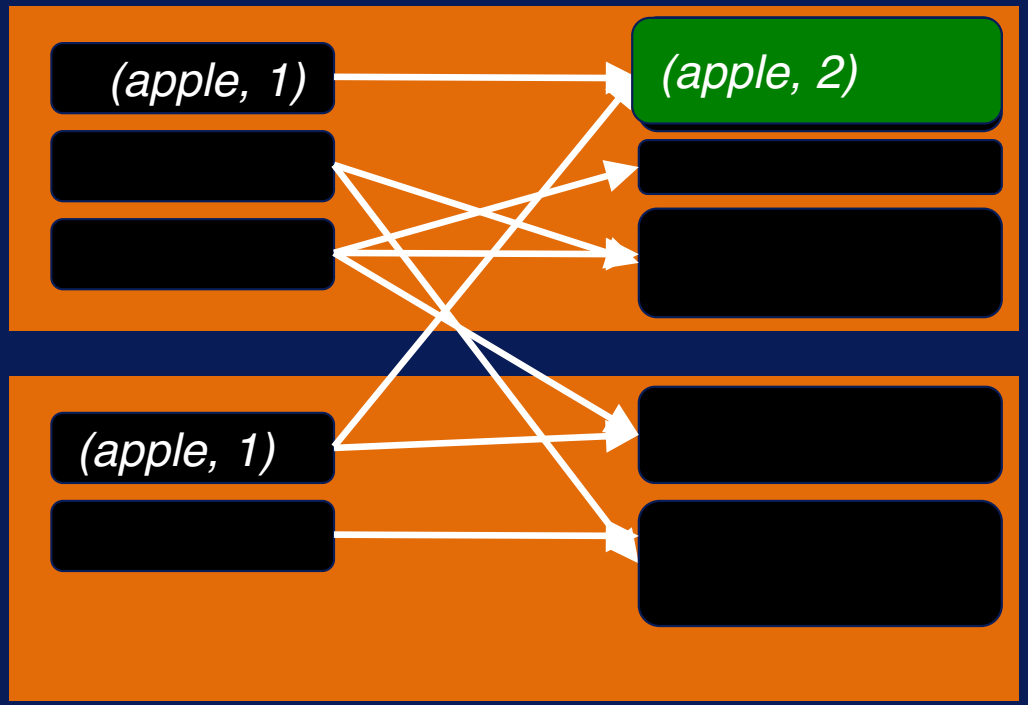
groupByKey + reduce = reduceByKey

*applying function to list (like sum)



reduceByKey

Combine values using reduce function (eg summation)

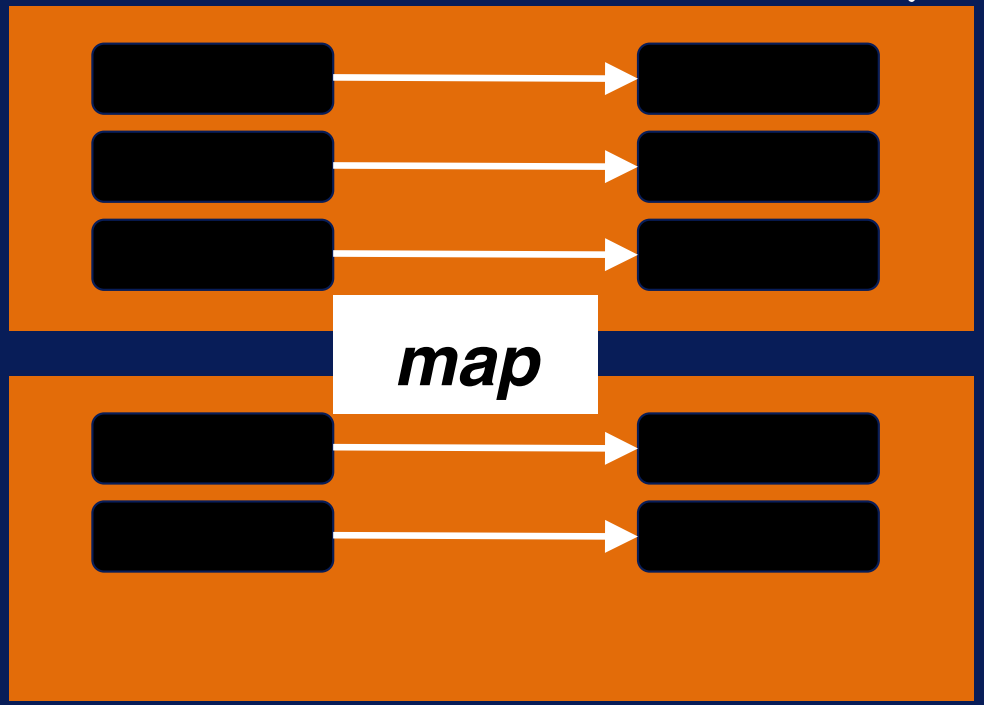


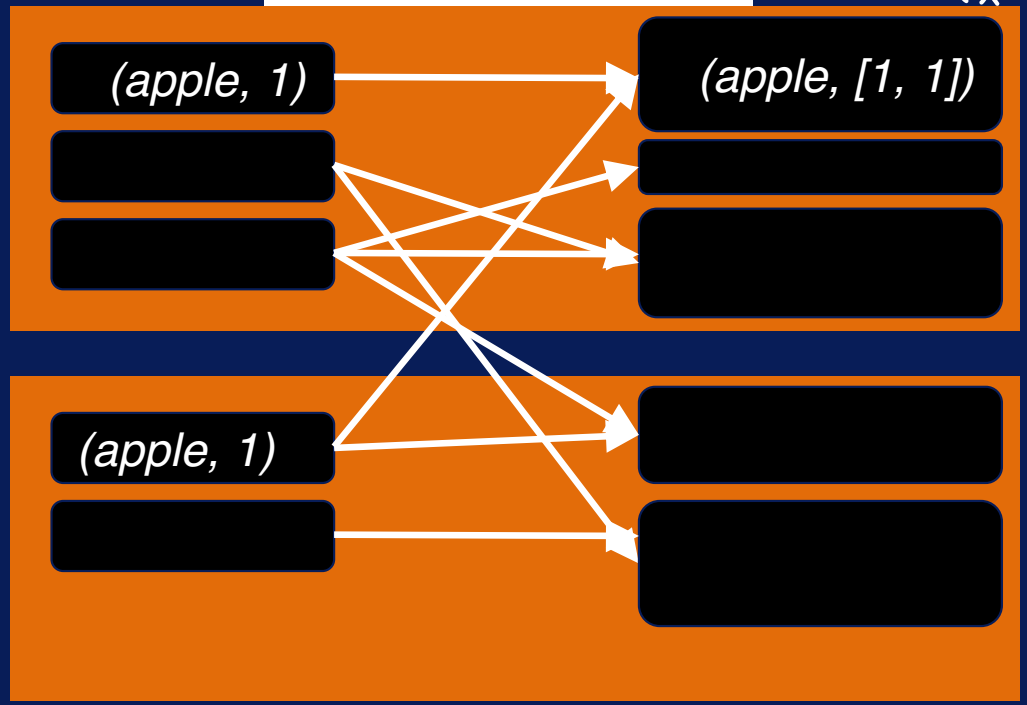
Narrow vs Wide

require shuffling of data across work nodes

* distributed across worker nodes and this requires data shuffling over the network to bring related datasets together

* processing depends on data residing in multiple partitions **





Many more transformations…

Full list of transformations at:

https://spark.apache.org/docs/1.2.0/programming-guide.html#transformations