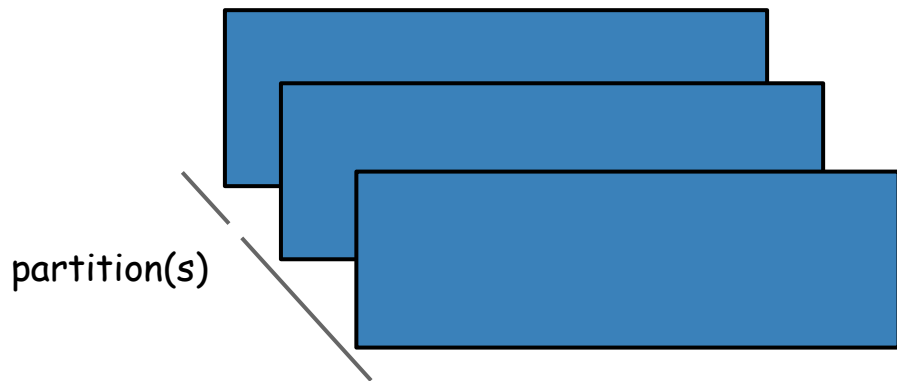


pyspark-pictures

Learn the pyspark API through pictures and simple examples

<https://github.com/jkthompson/pyspark-pictures>

RDD

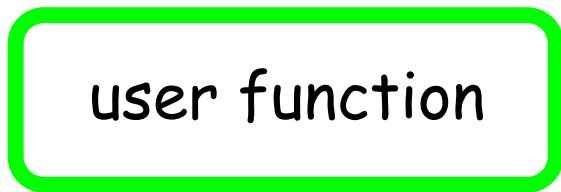


user input



user function

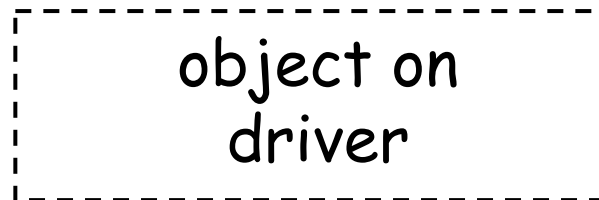
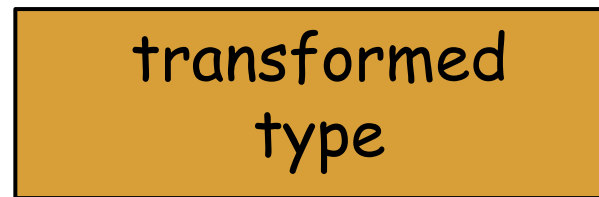
output ←



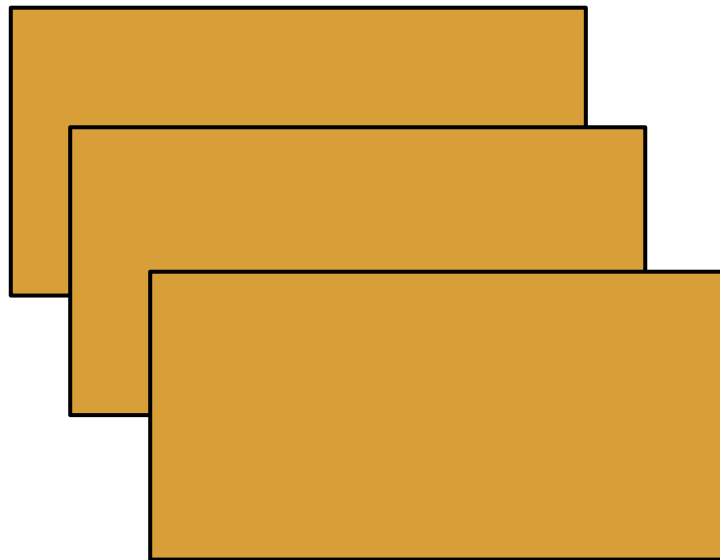
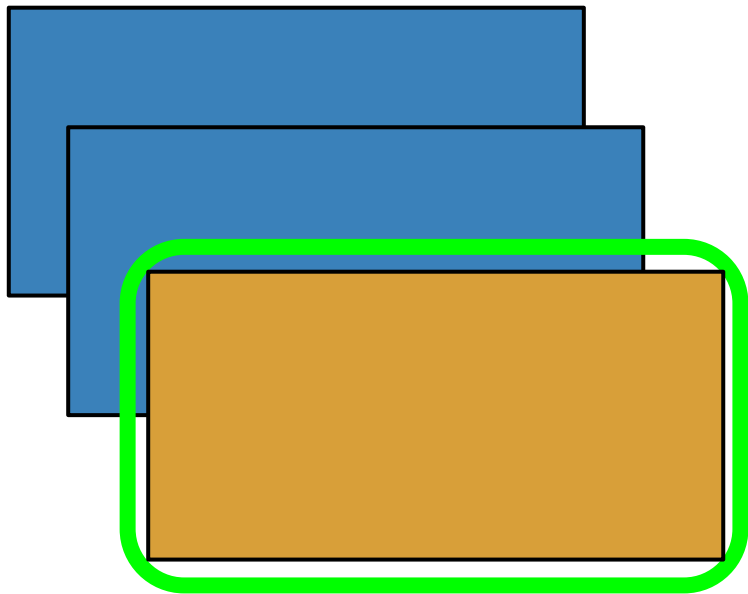
spark input



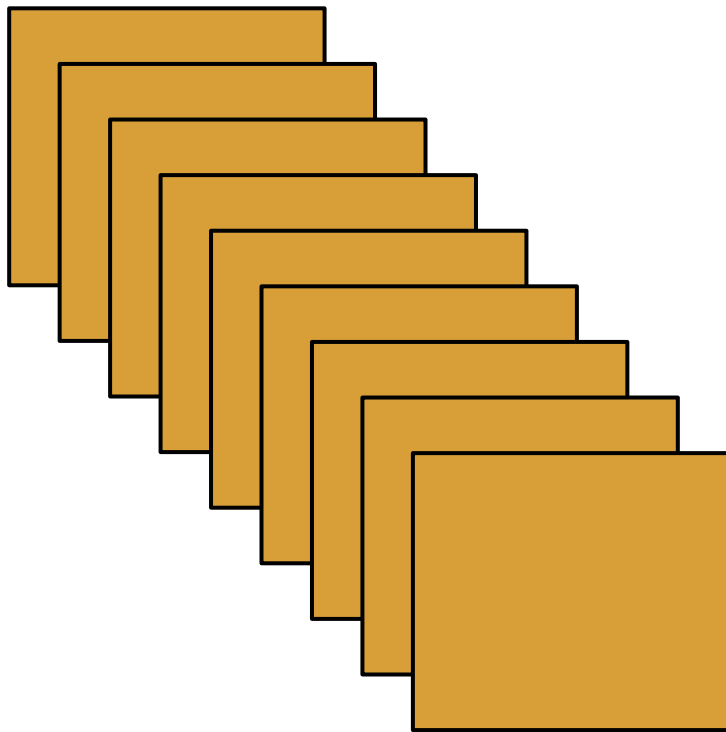
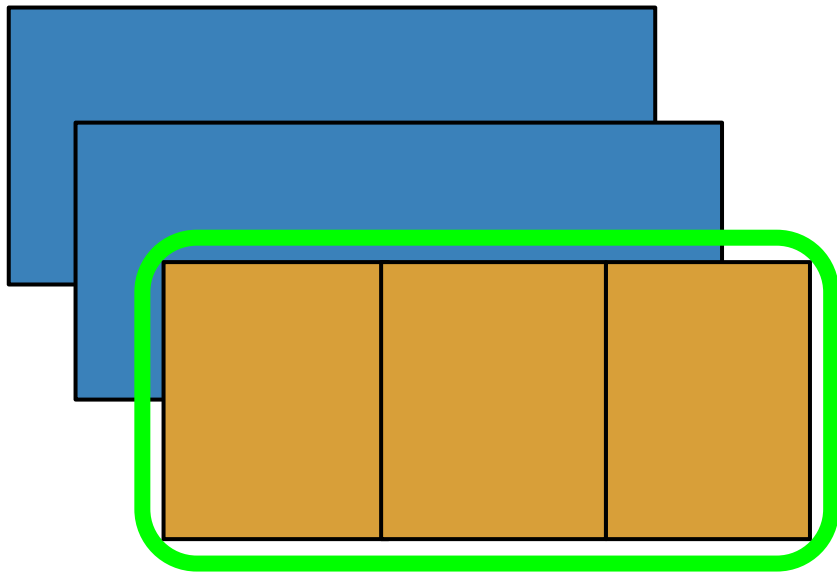
RDD Elements



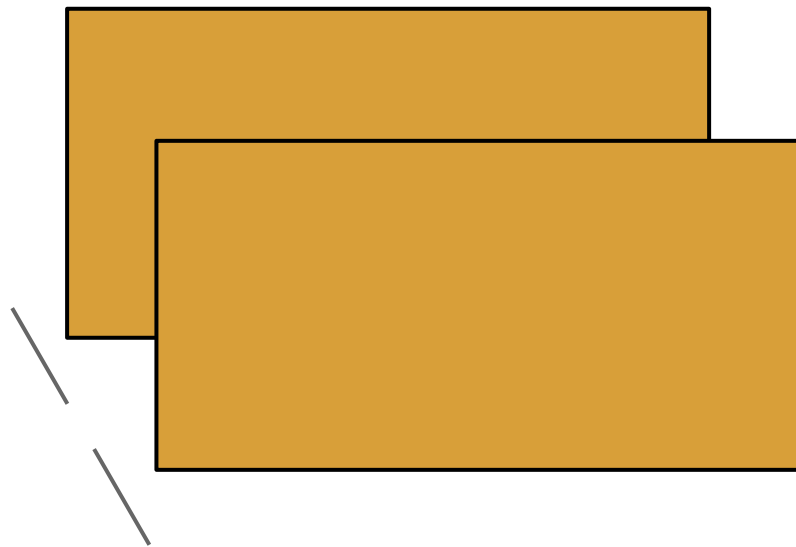
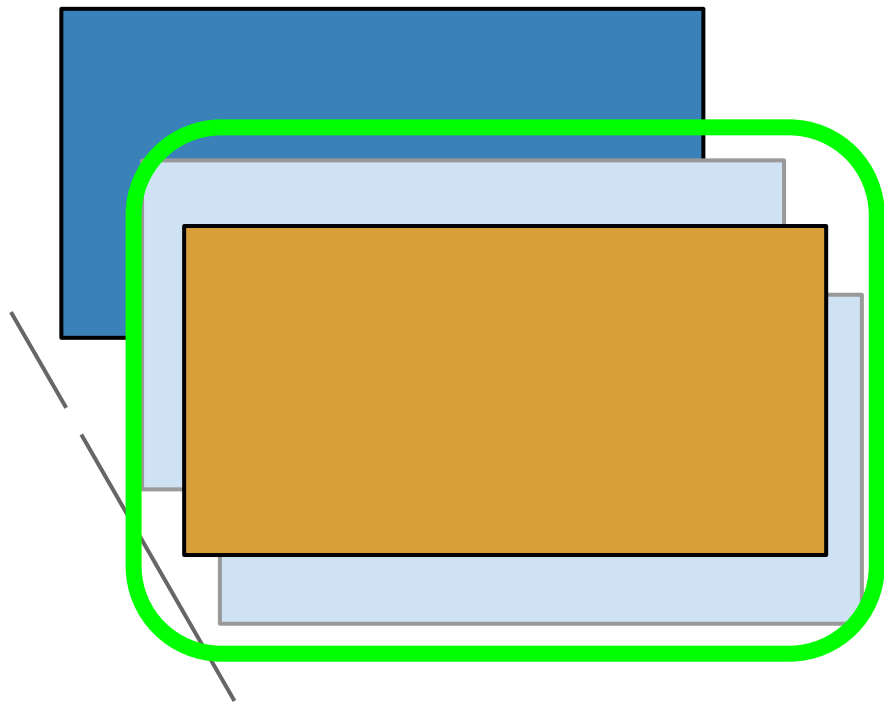
map



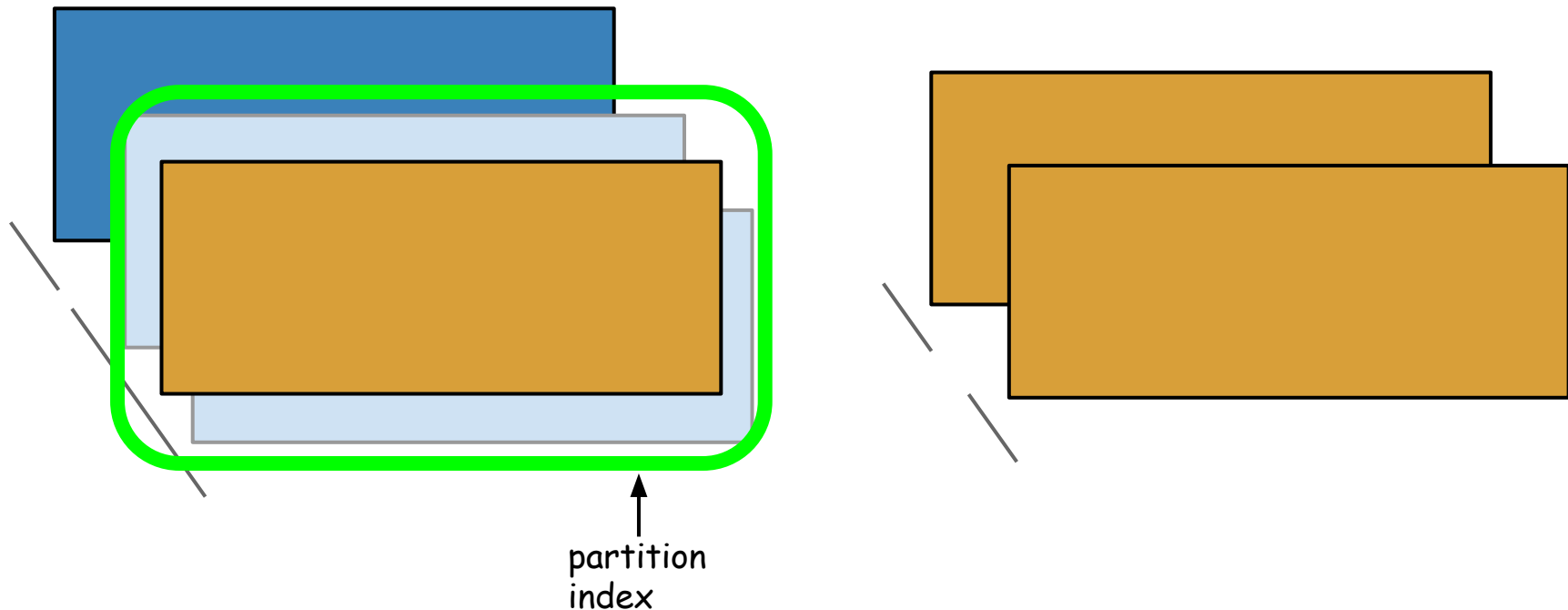
flatMap



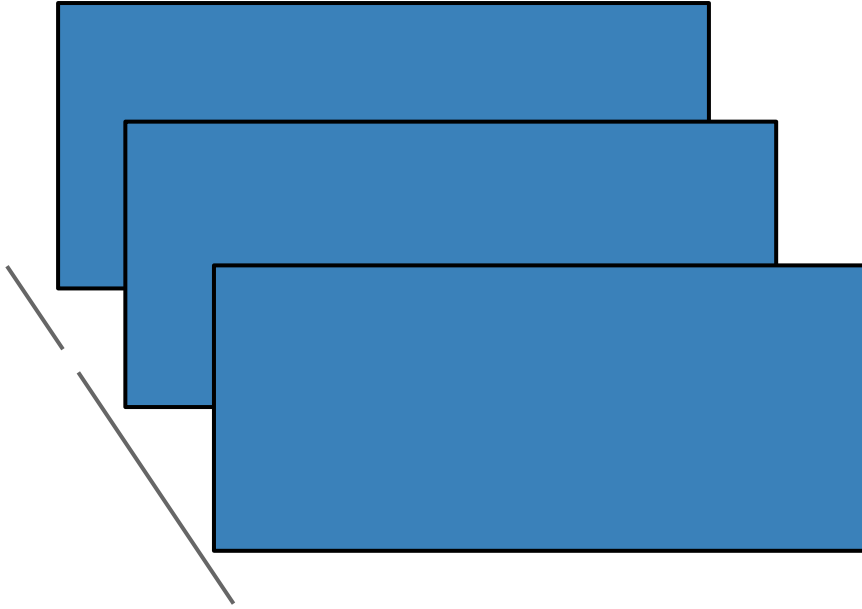
mapPartitions



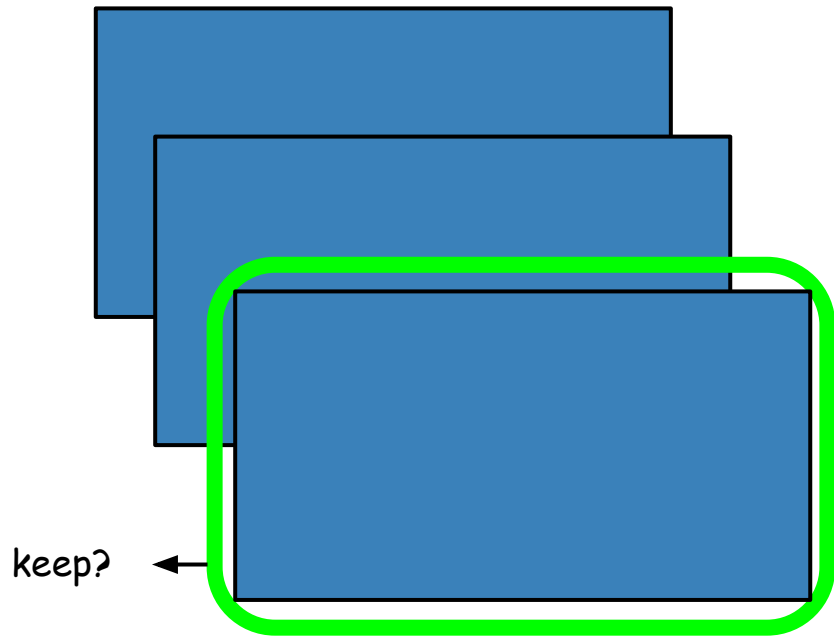
mapPartitionsWithIndex



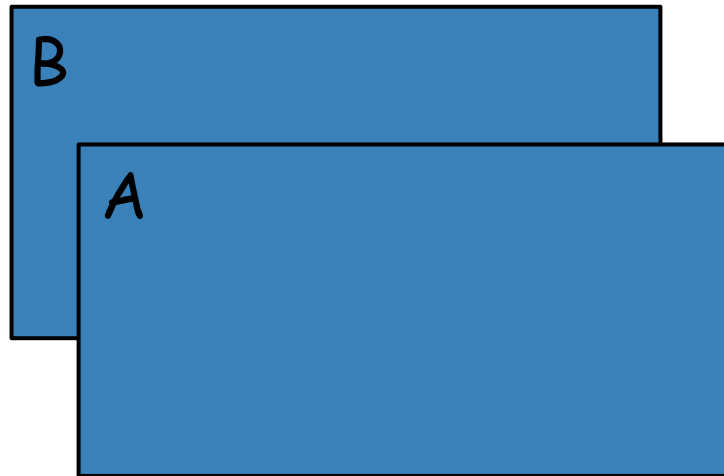
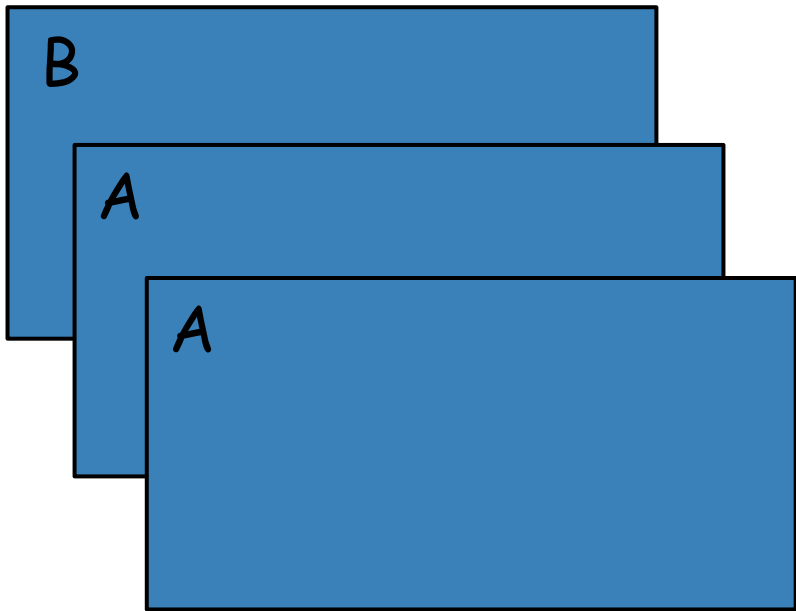
getNumPartitions



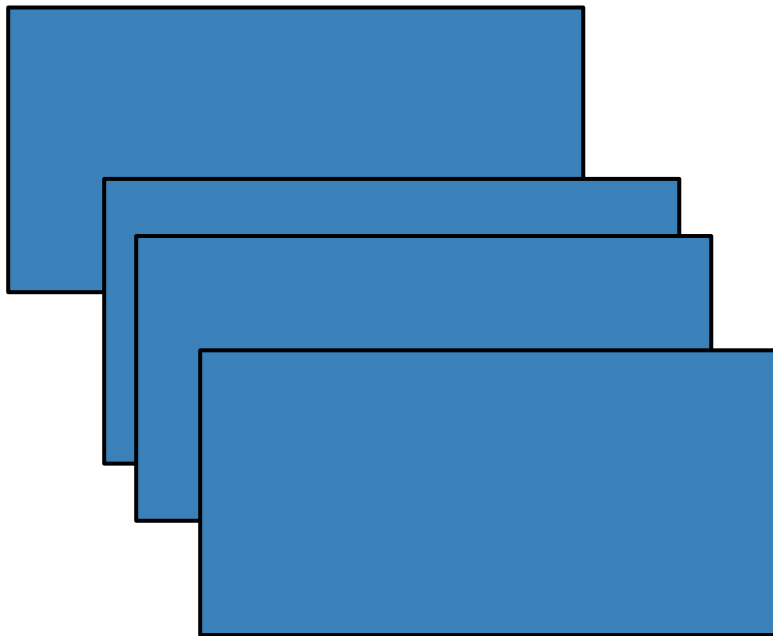
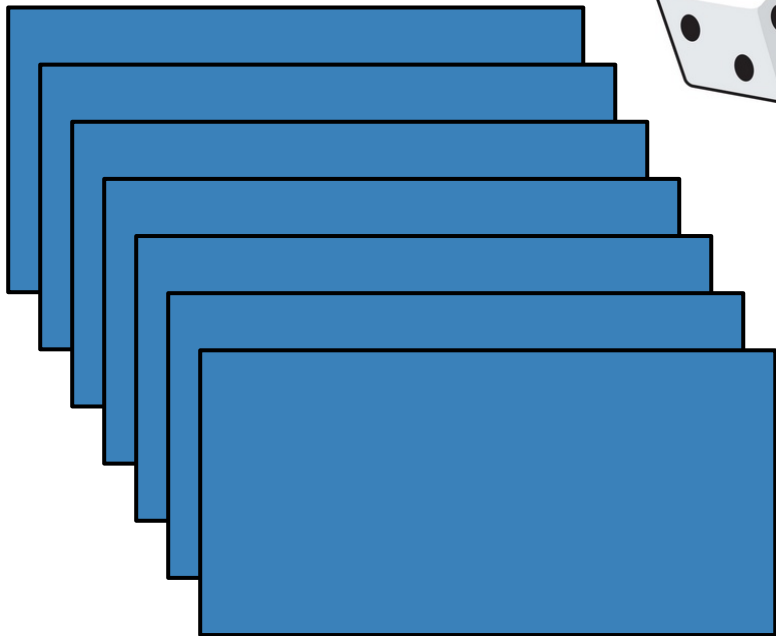
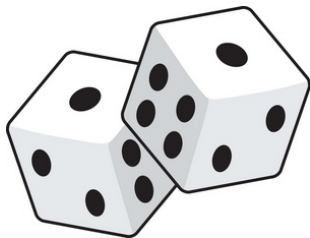
filter



distinct

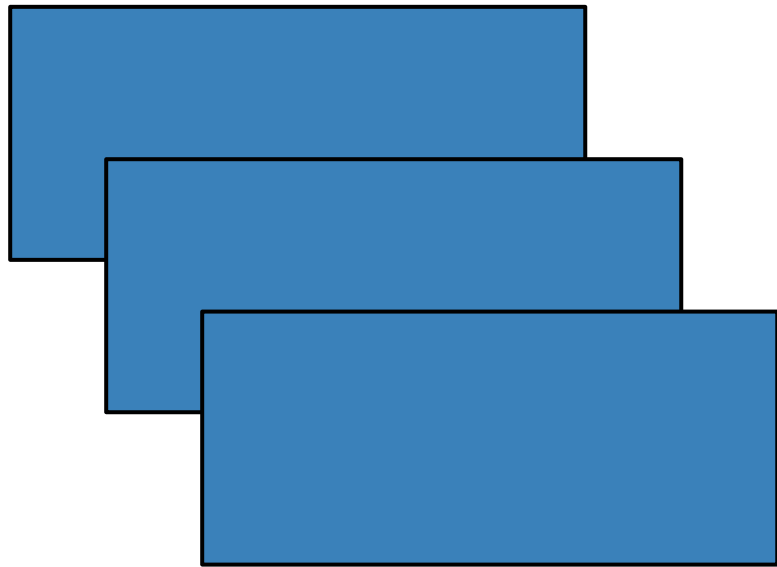
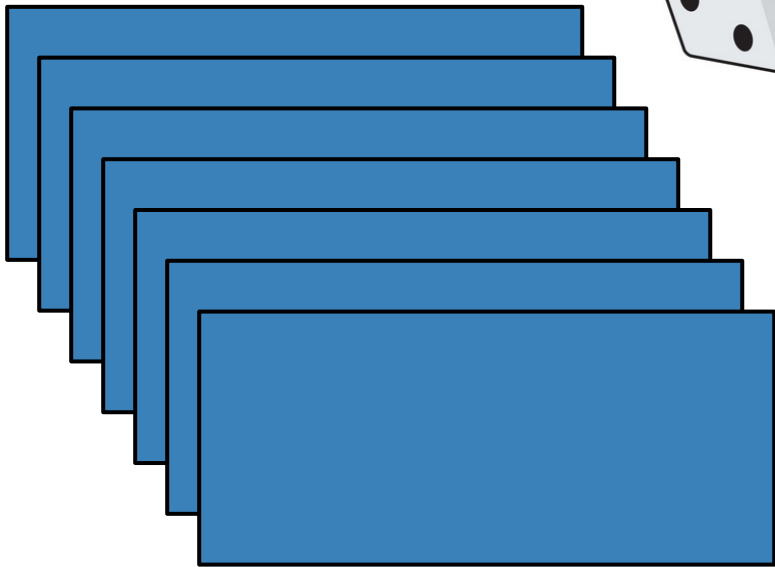
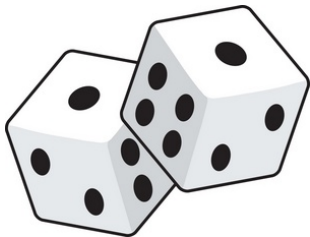


sample

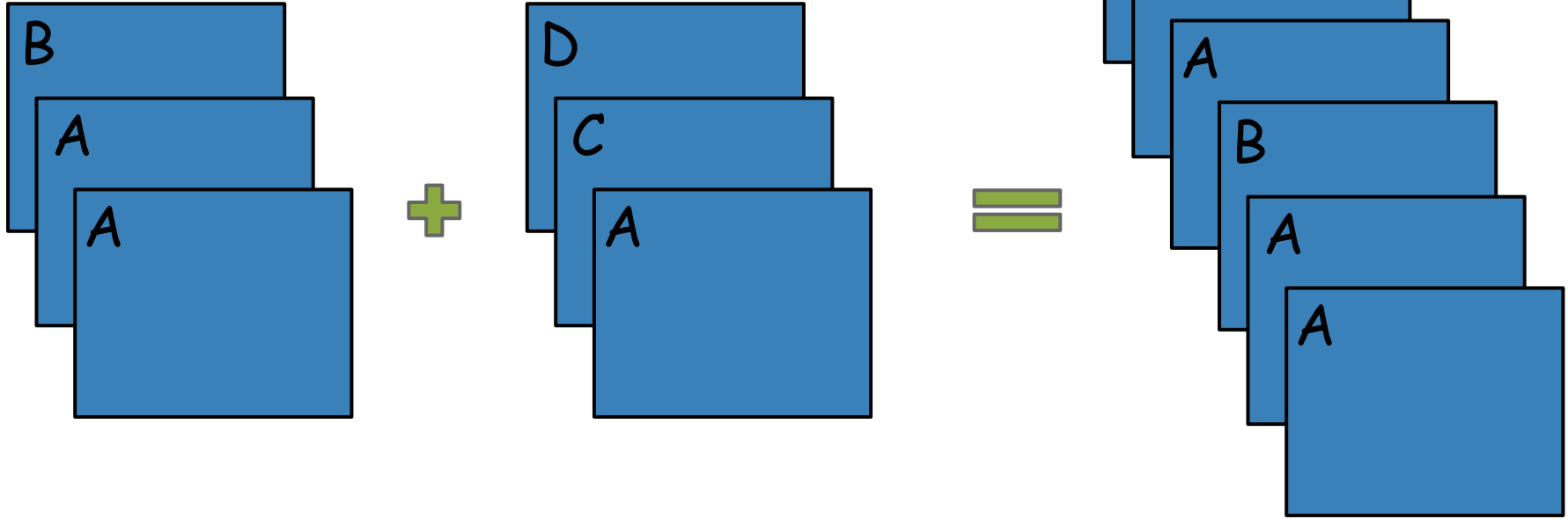


takeSample

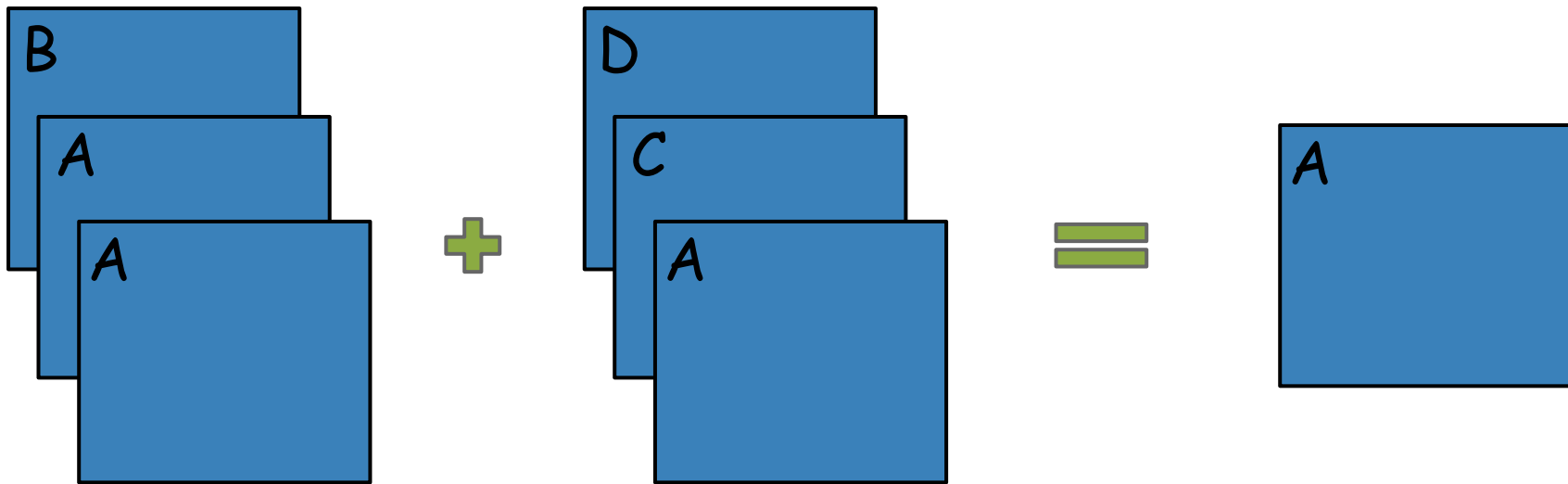
num = 3



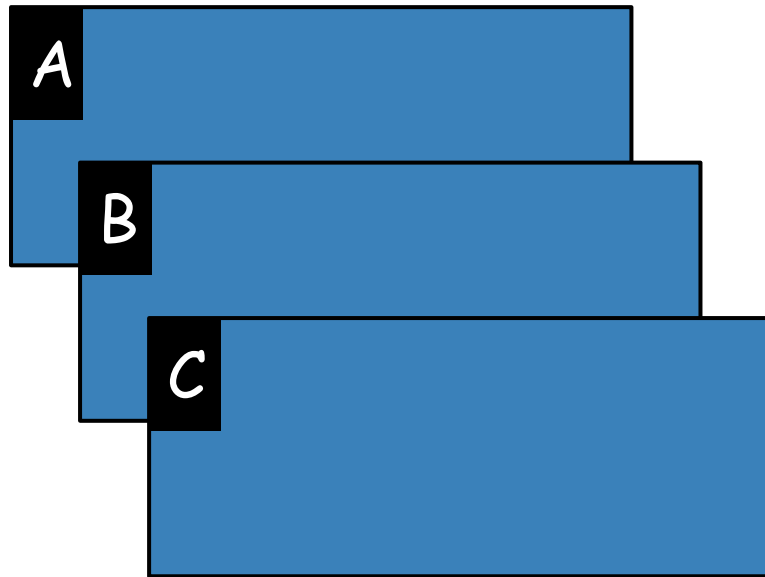
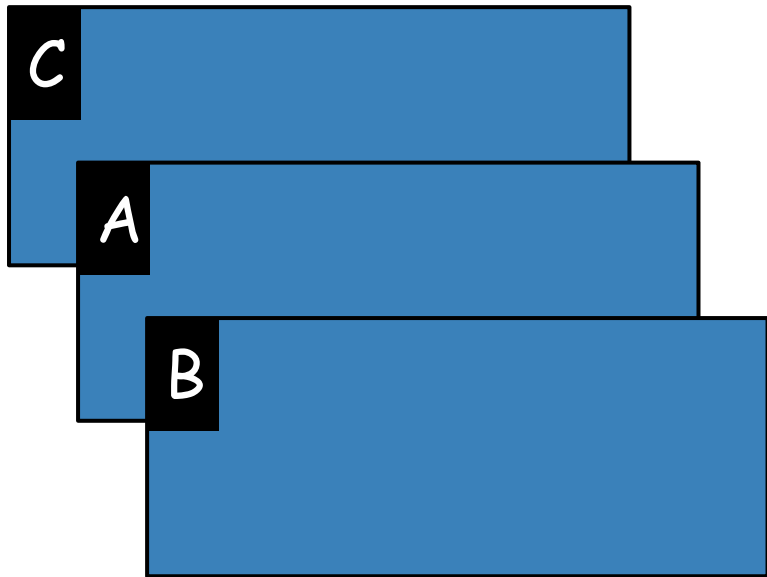
union (+)



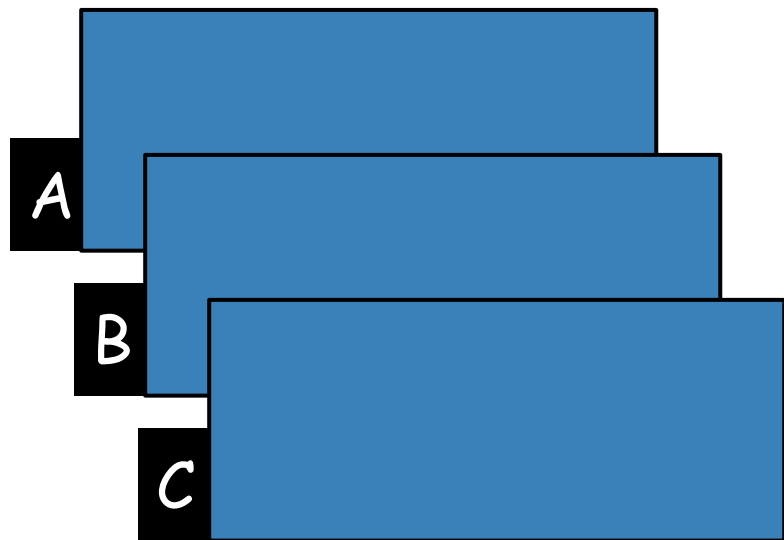
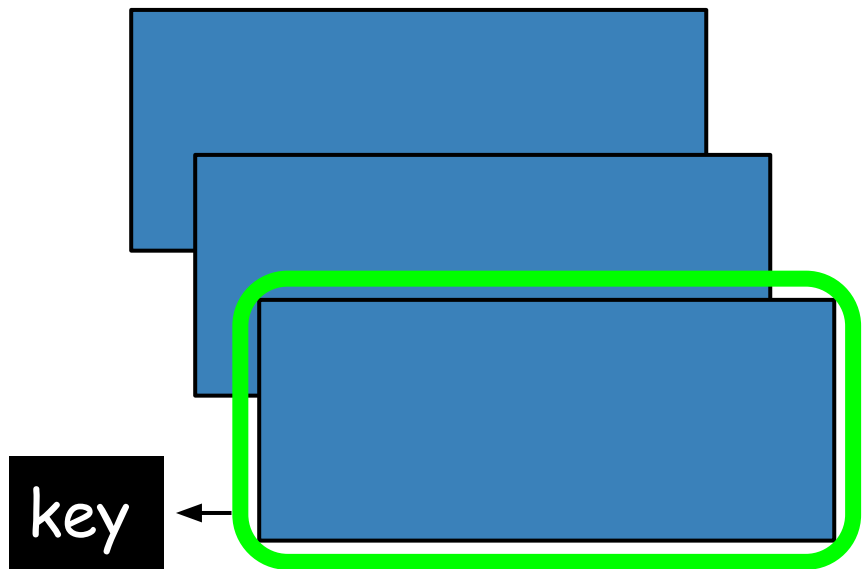
intersection



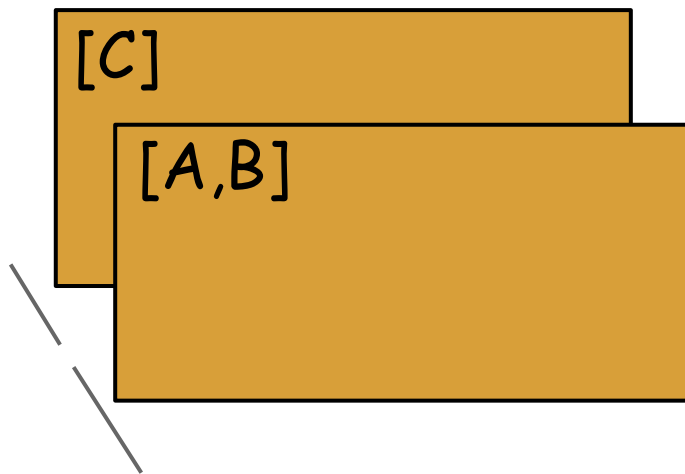
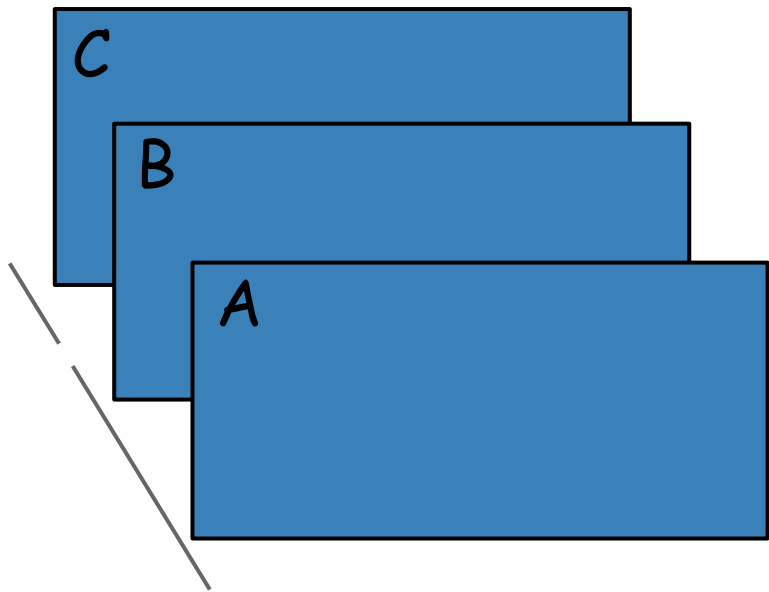
sortByKey



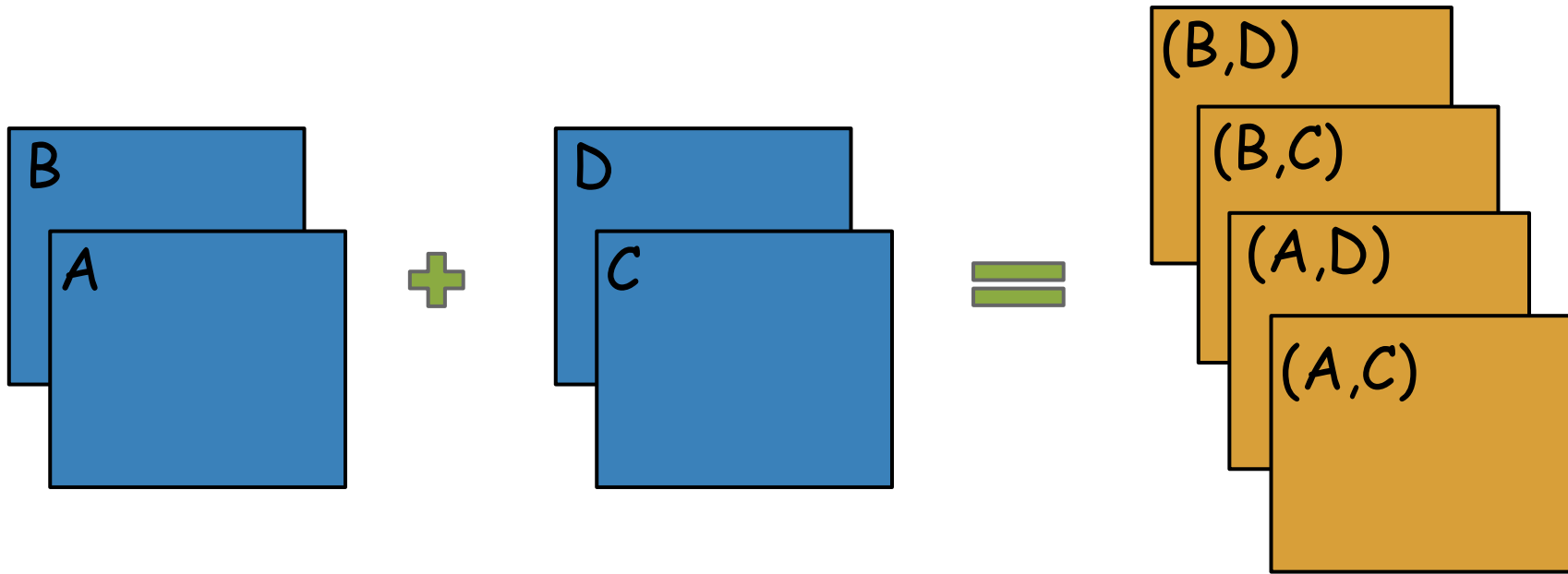
sortBy



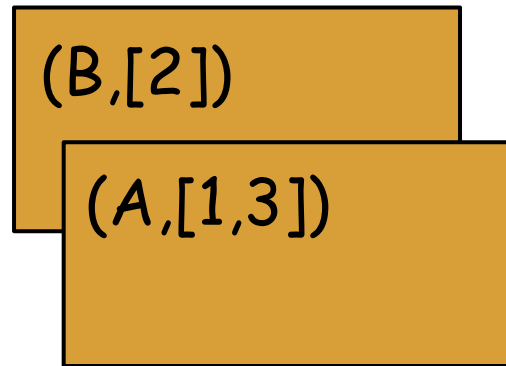
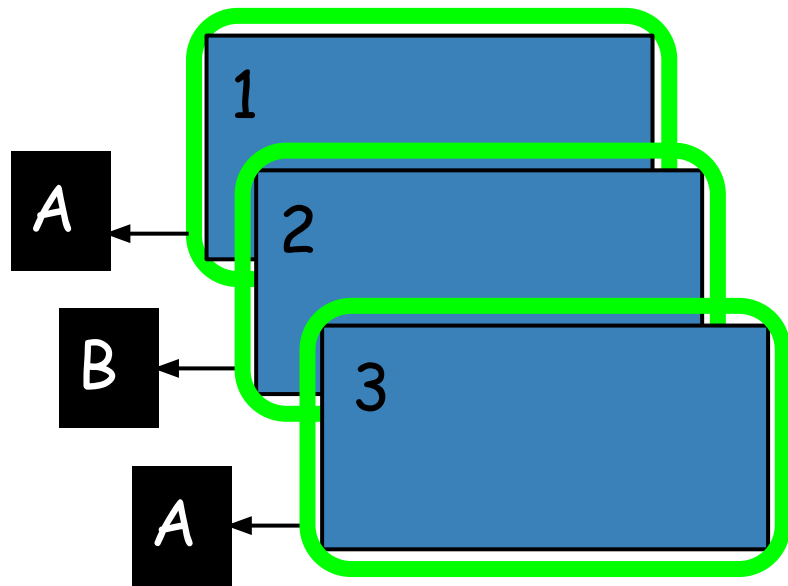
glom



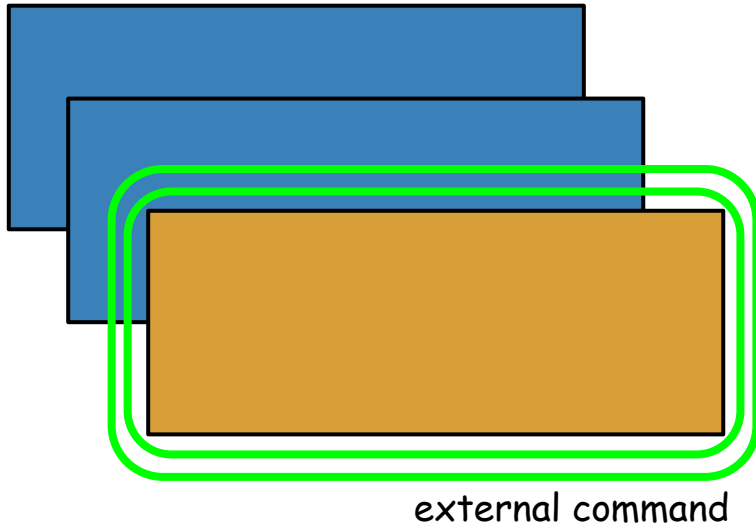
cartesian



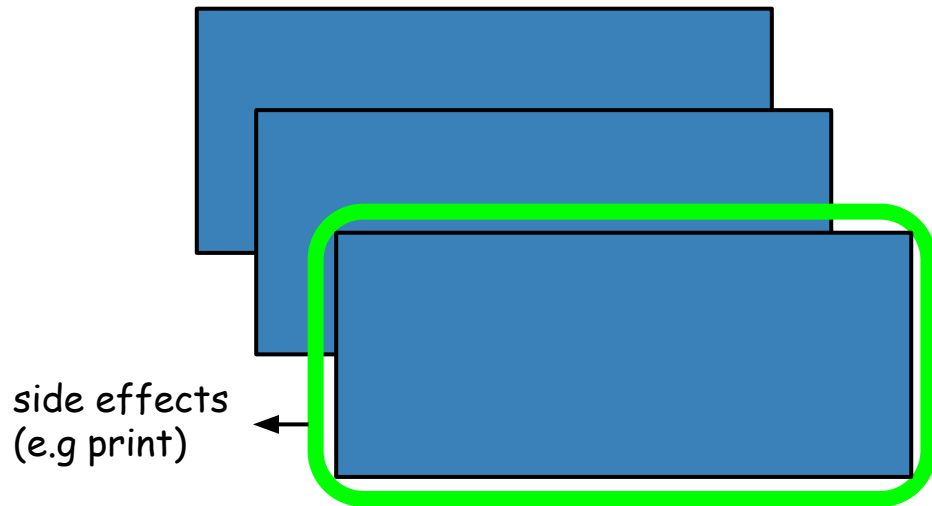
groupBy



pipe

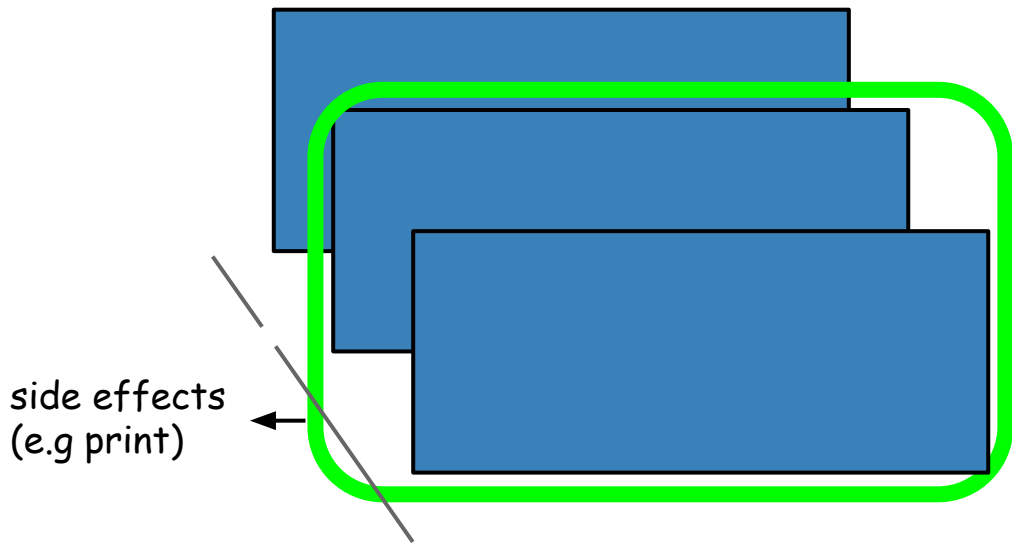


foreach



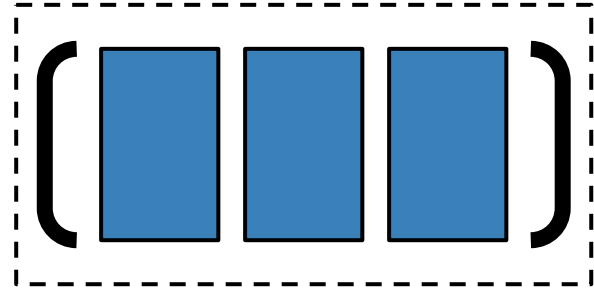
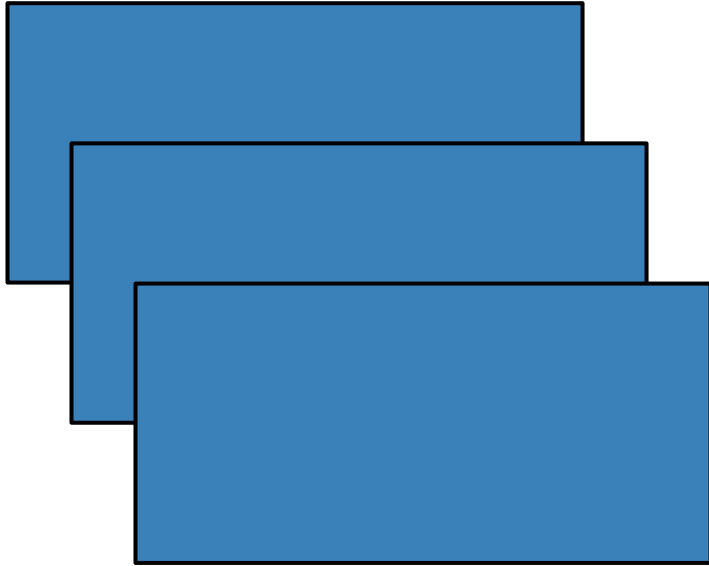
*no return value,
original RDD unchanged

foreachPartition

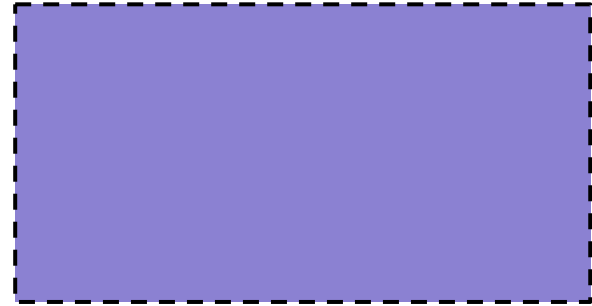
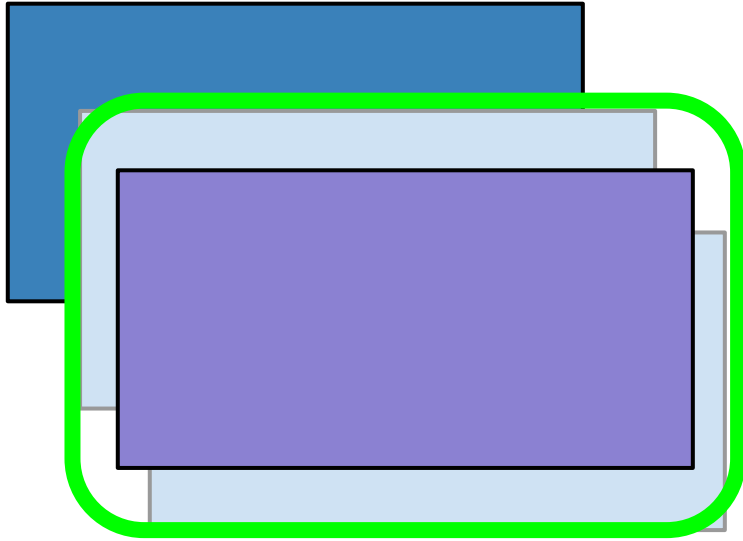


*no return value,
original RDD unchanged

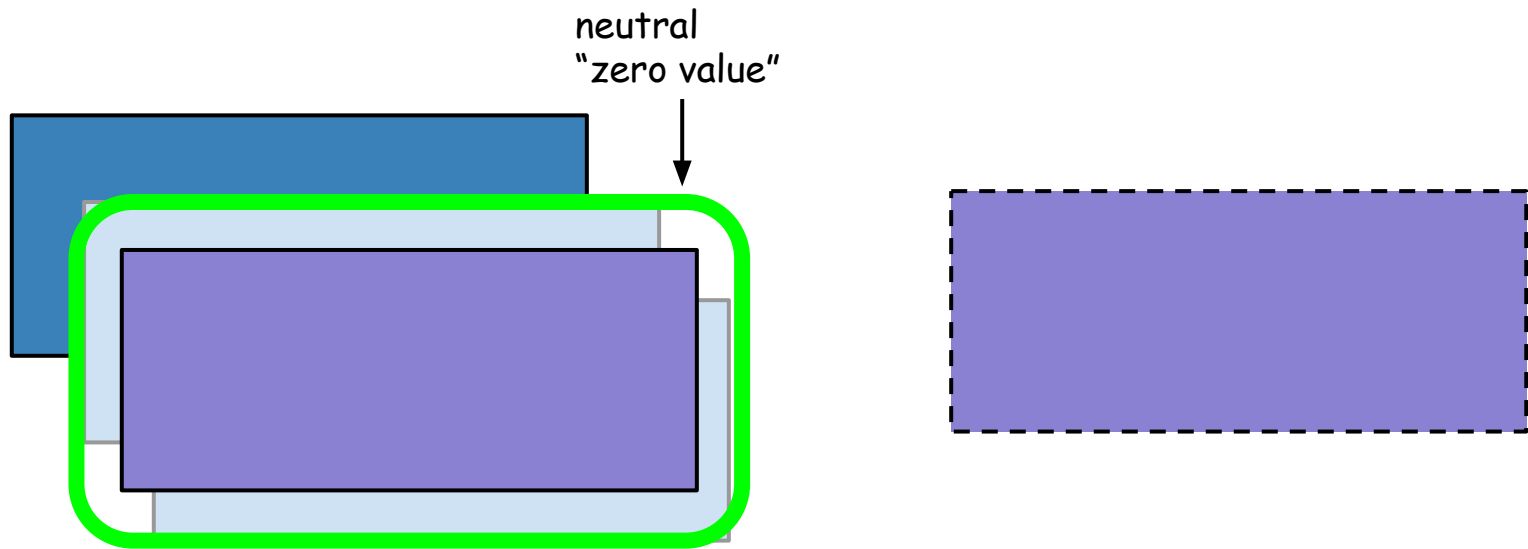
collect



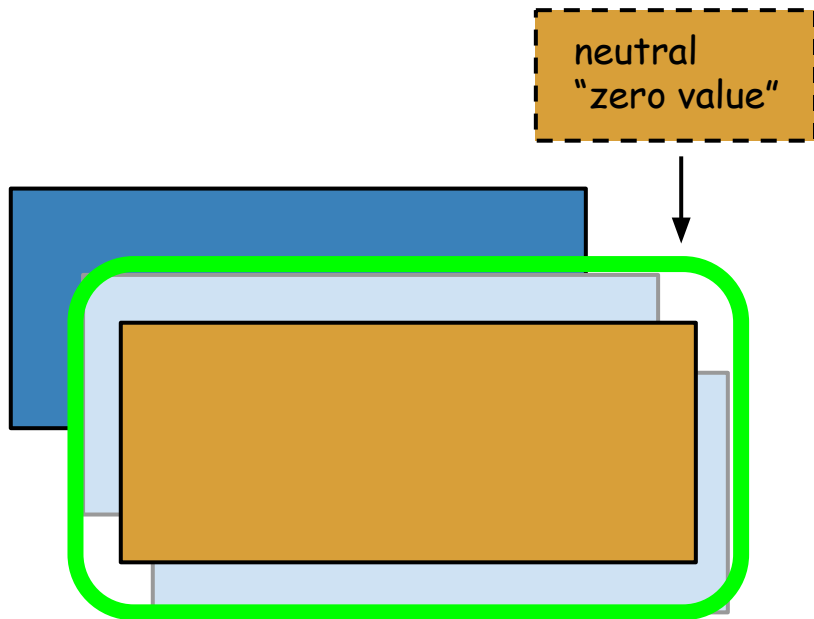
reduce



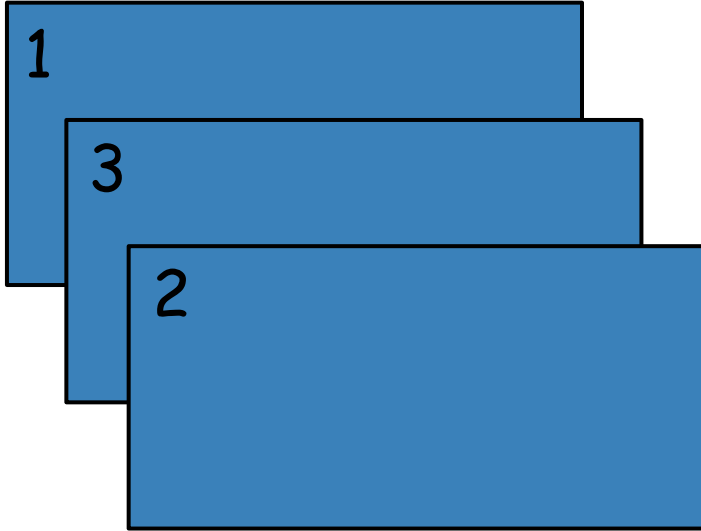
fold



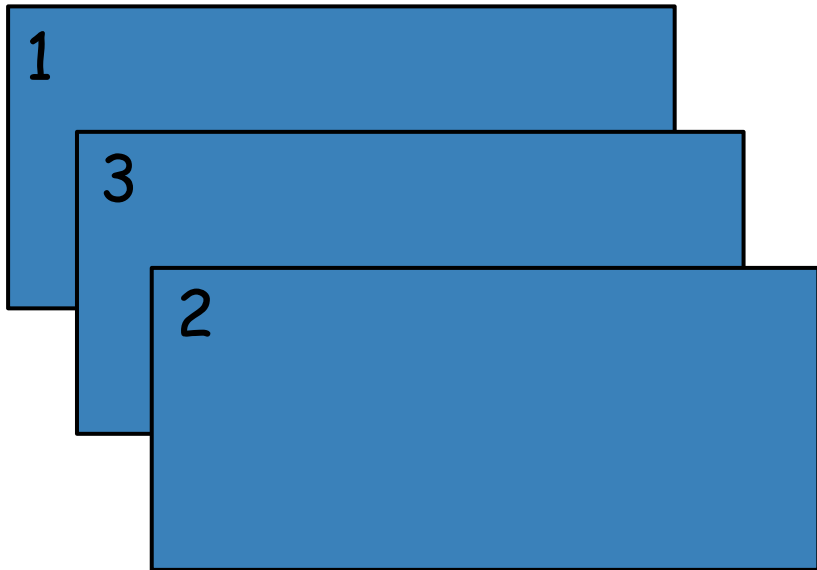
aggregate



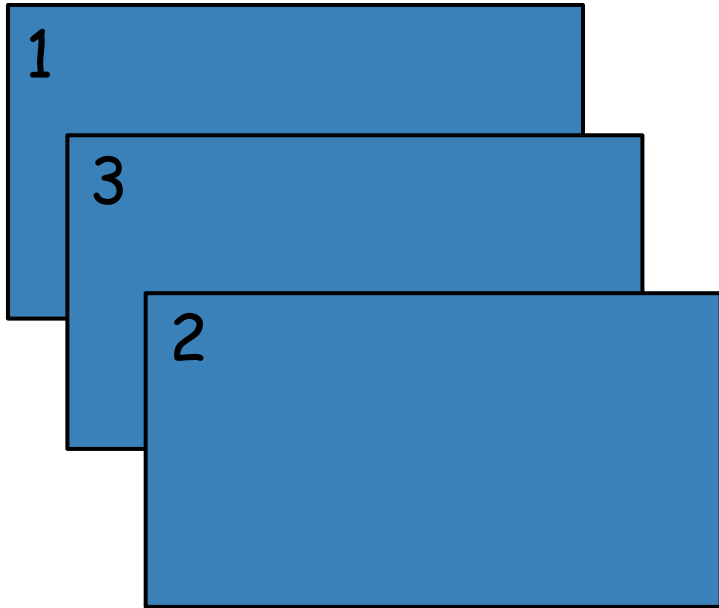
max



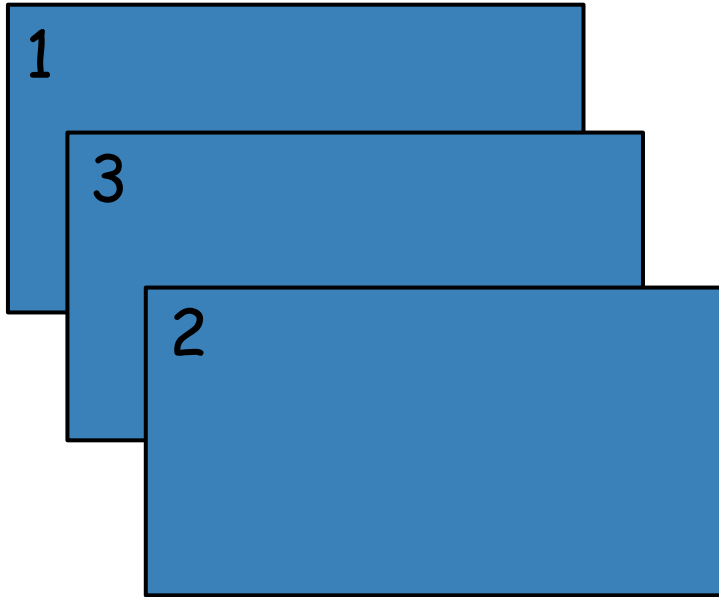
min



sum

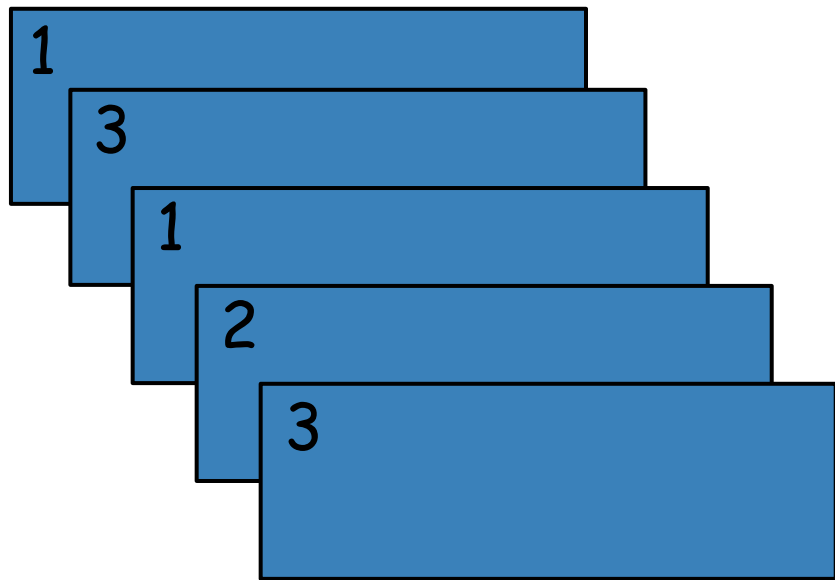


count



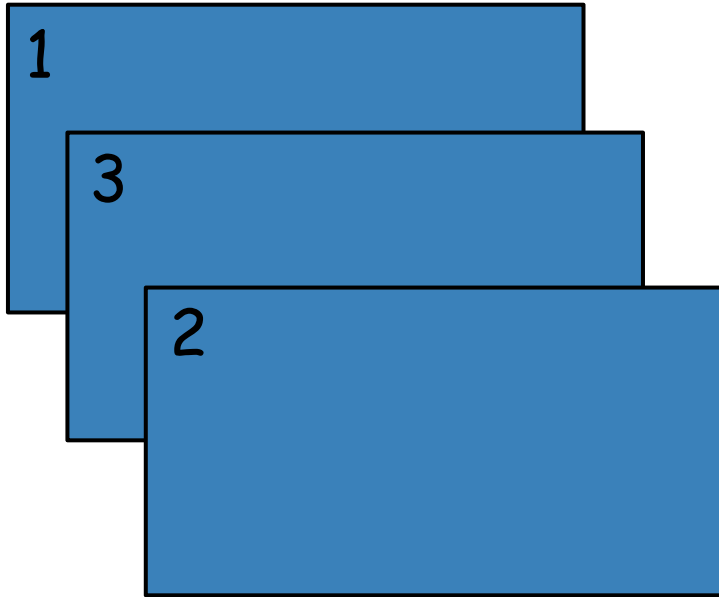
histogram

buckets = 2

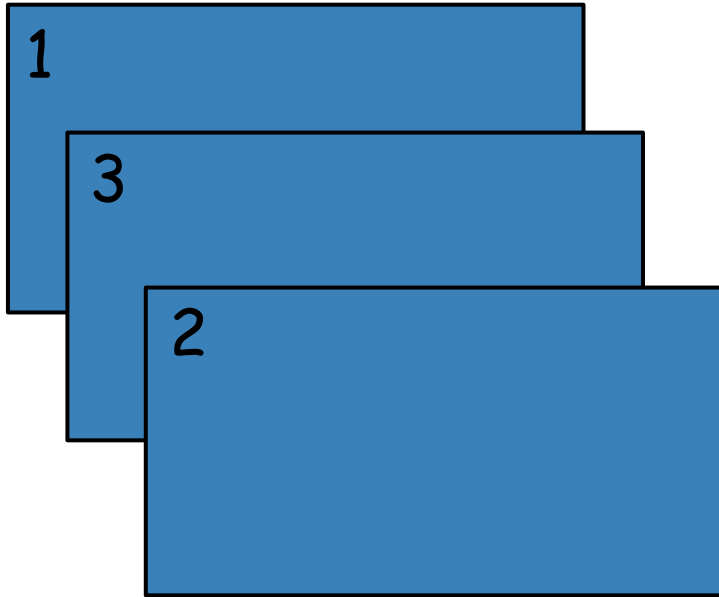


$((1,2,3),[2,3])$

mean

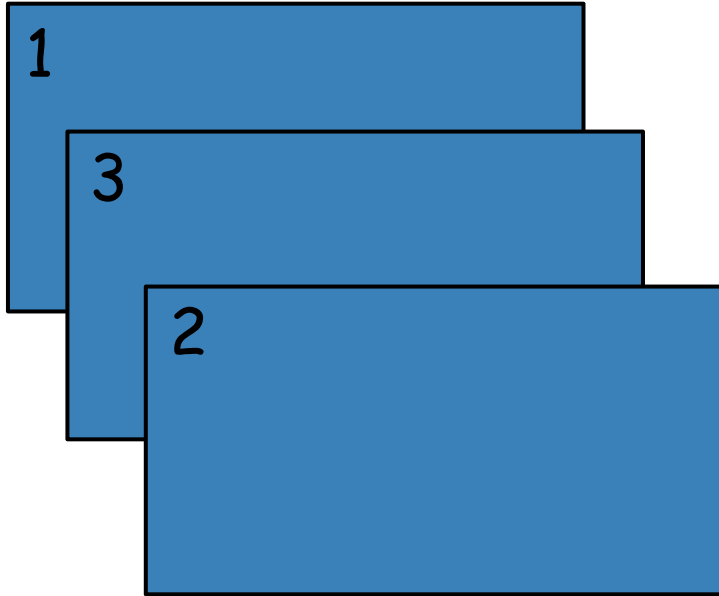


variance



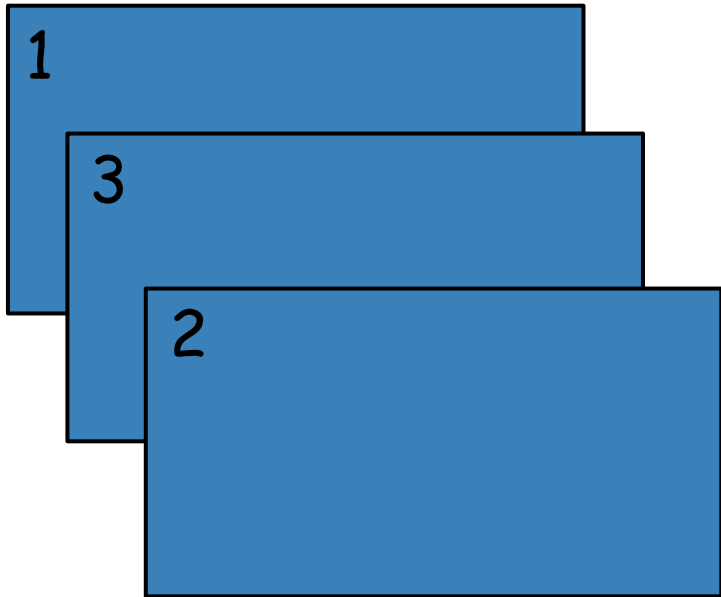
0.666...

stdev

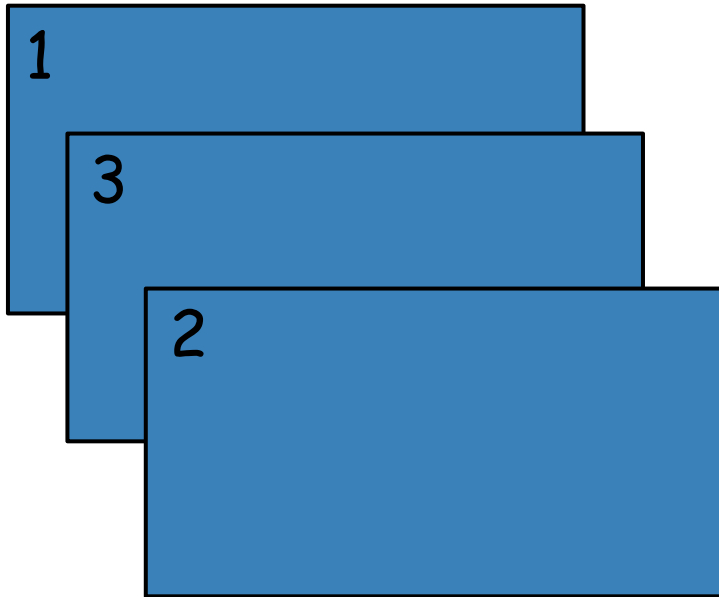


0.816...

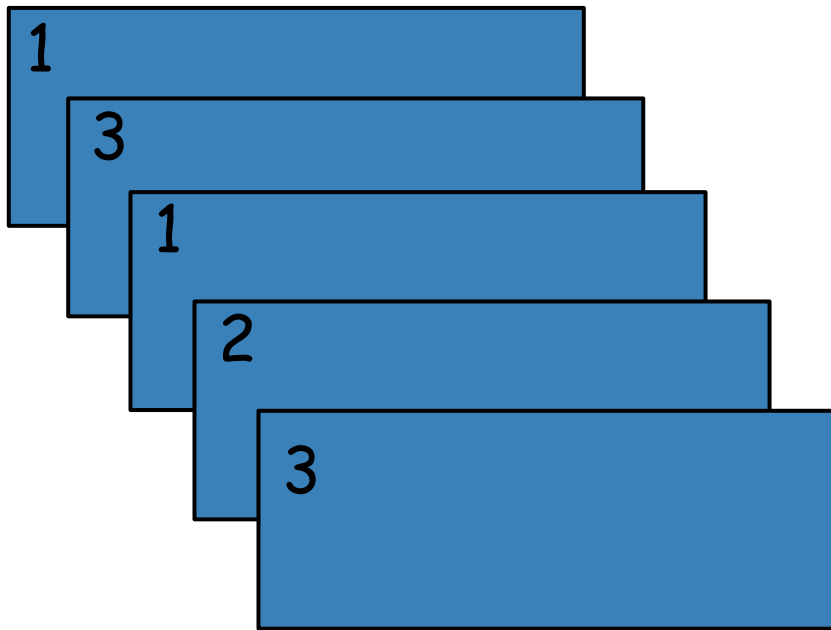
sampleStdev



sampleVariance



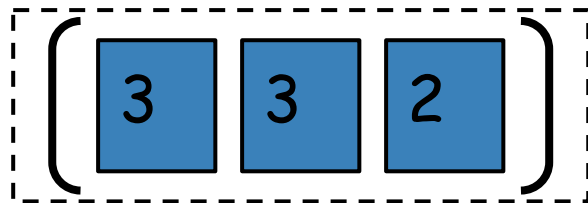
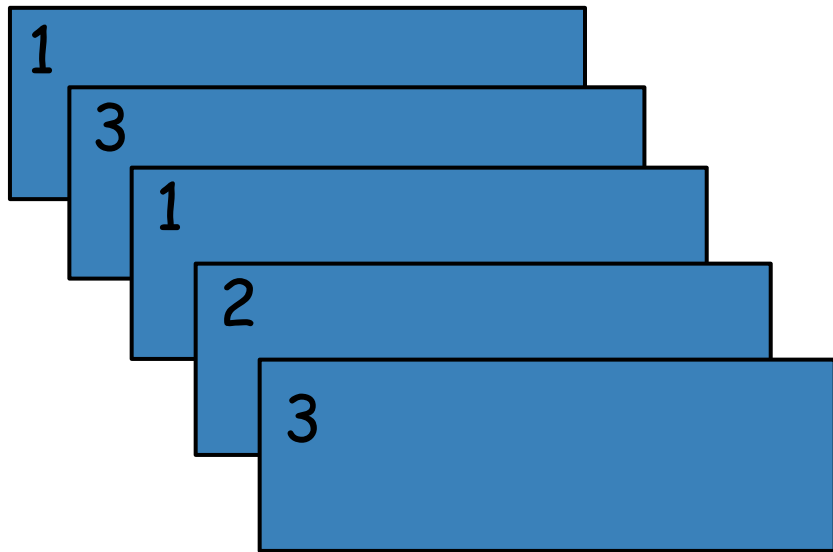
countByValue



$\{ 1 : 2, 2 : 1, 3 : 2 \}$

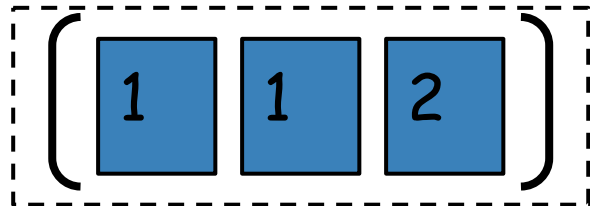
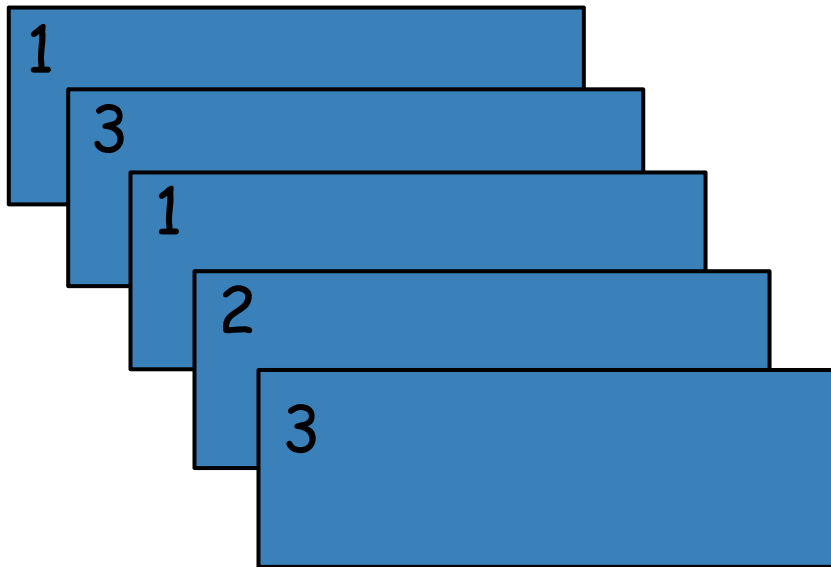
top

num = 3



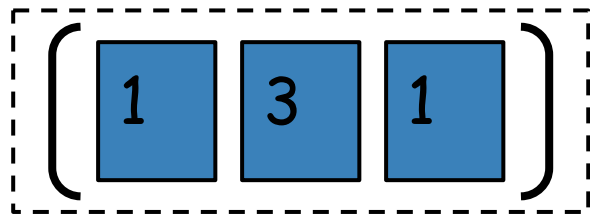
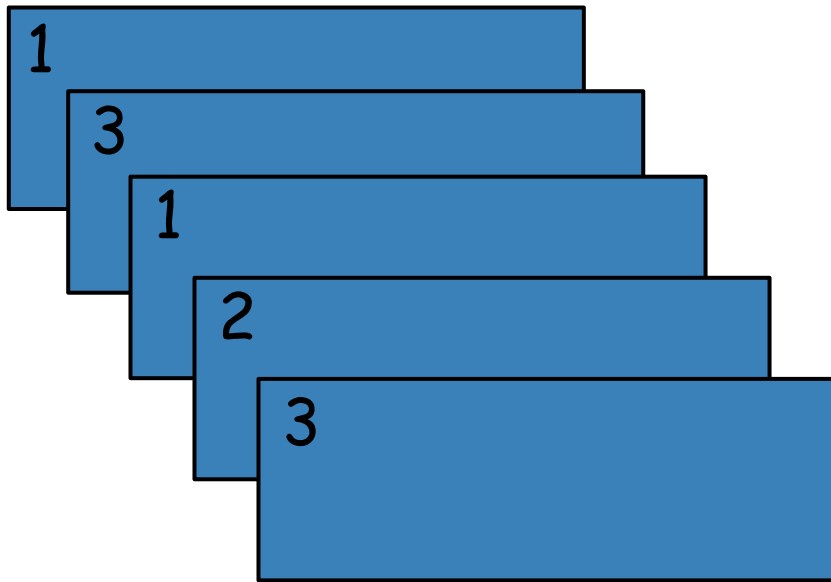
takeOrdered

num = 3

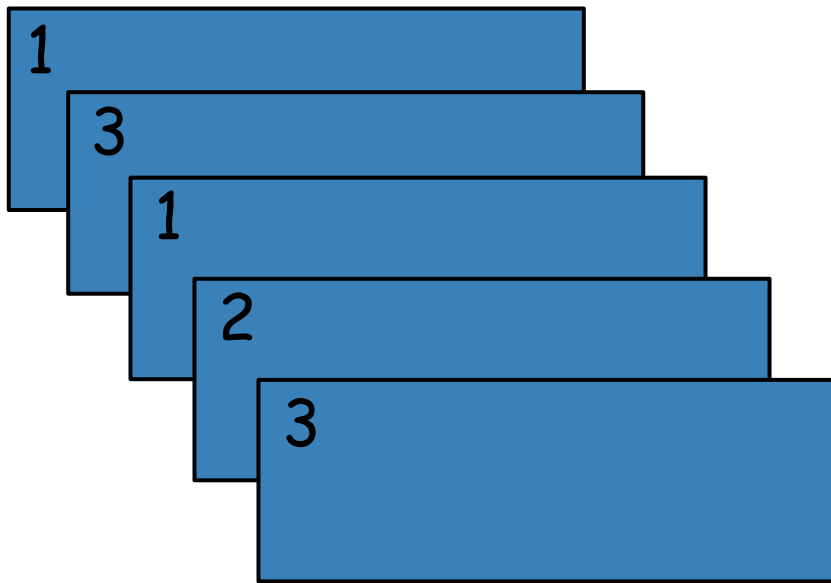


take

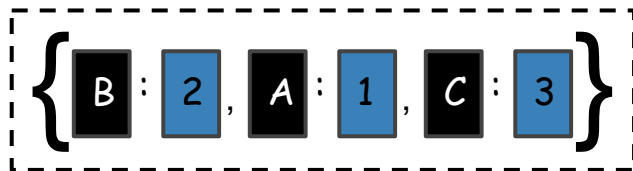
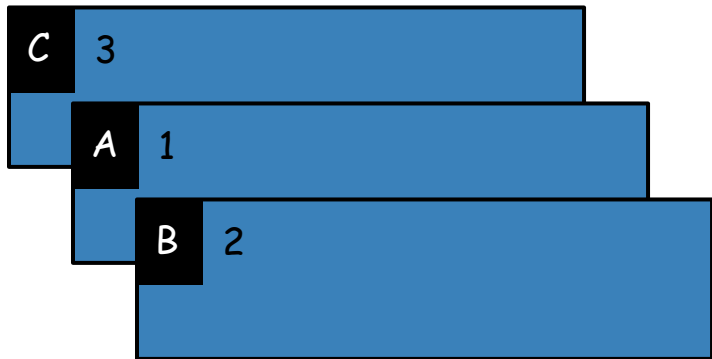
num = 3



first



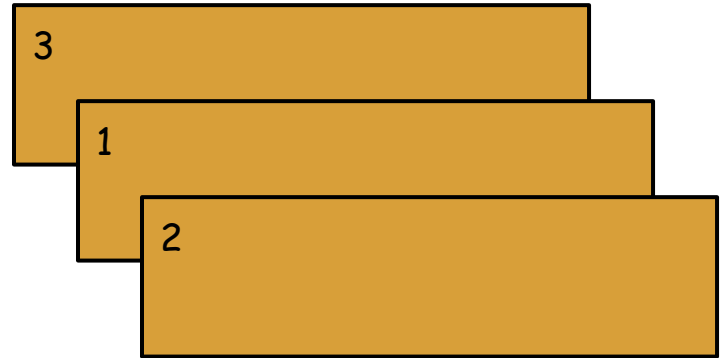
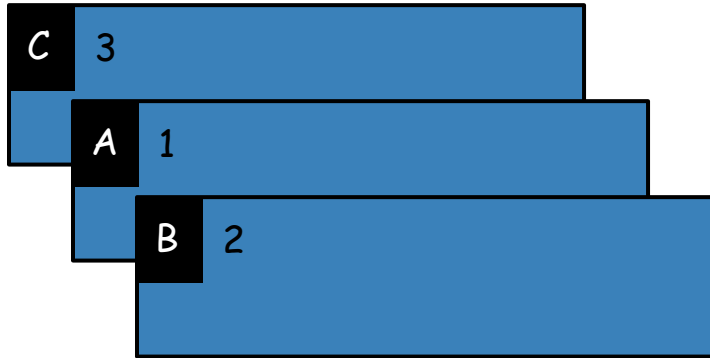
collectAsMap



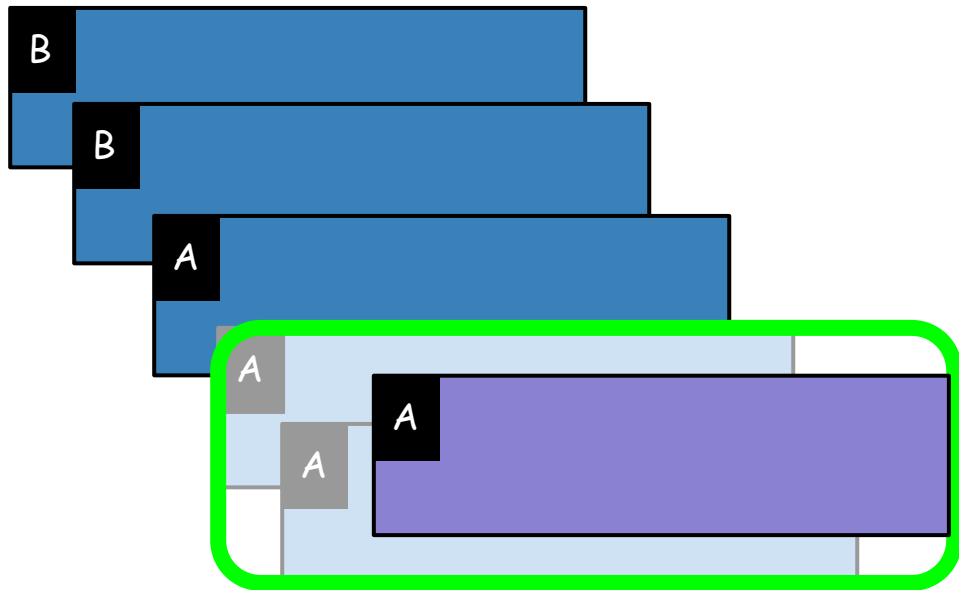
keys



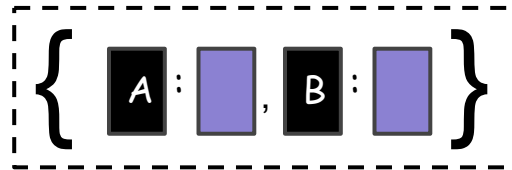
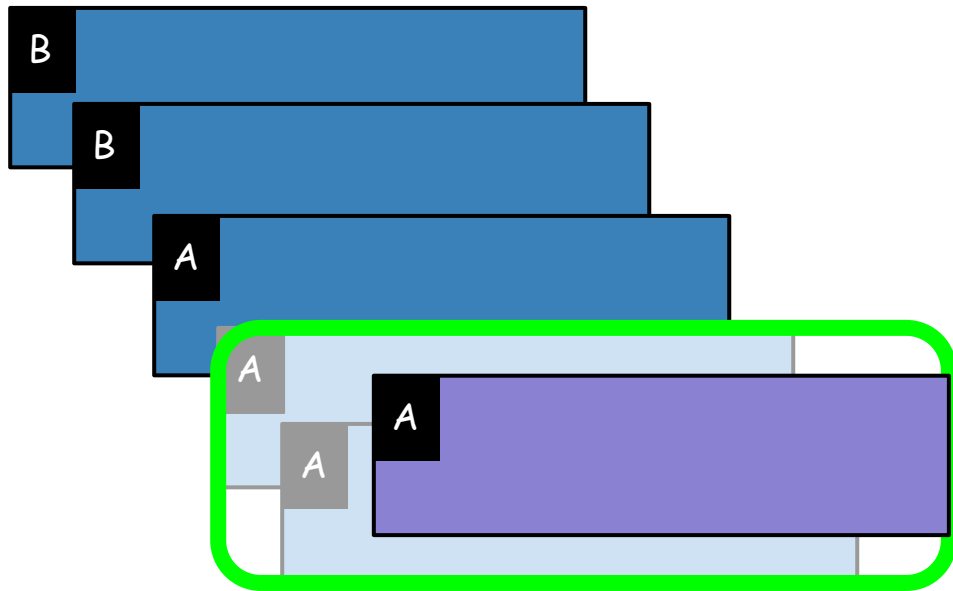
values



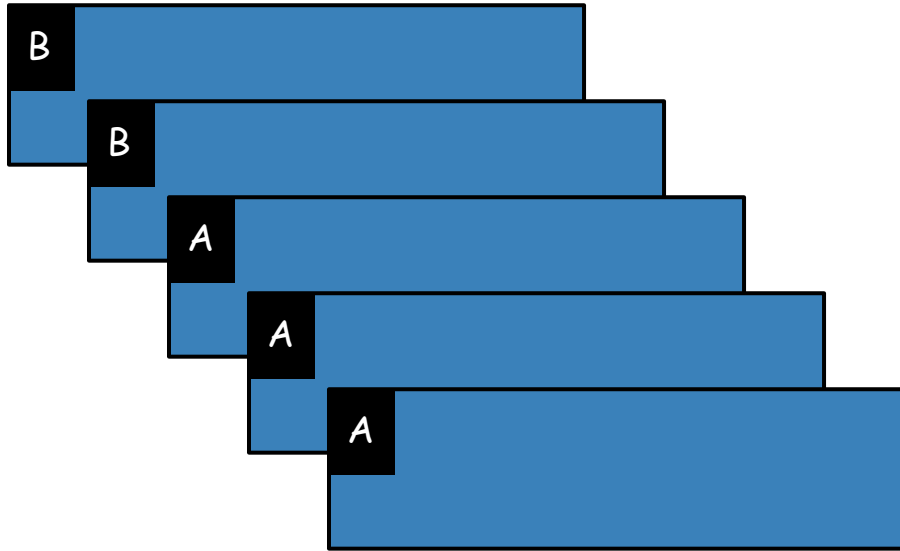
reduceByKey



reduceByKeyLocally

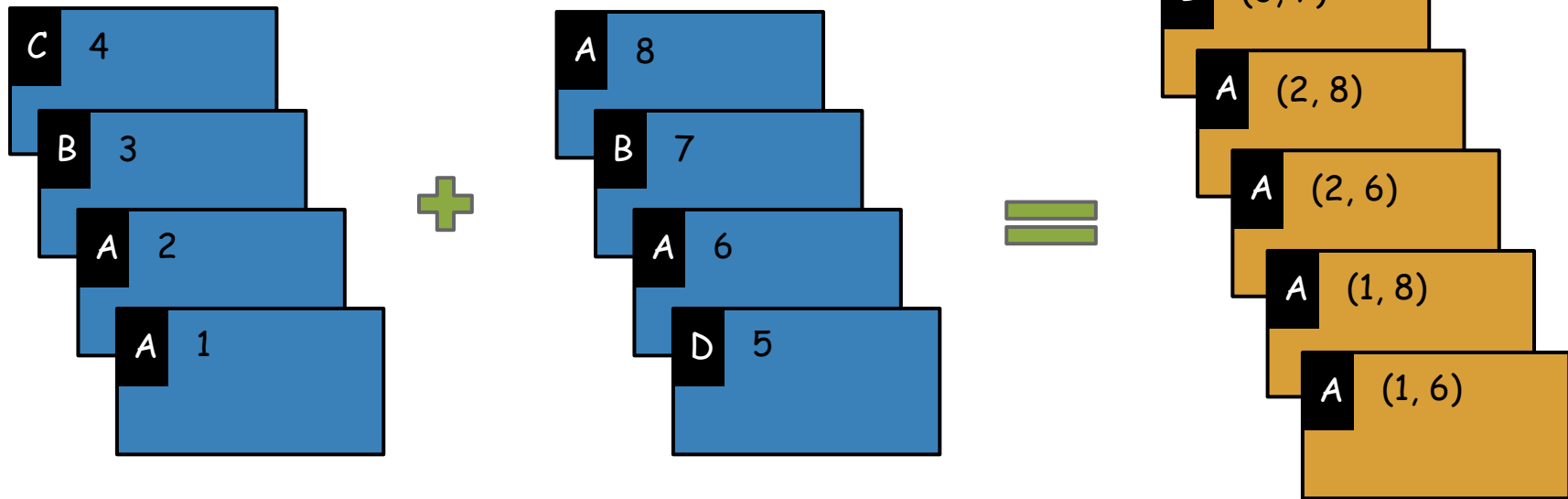


countByKey

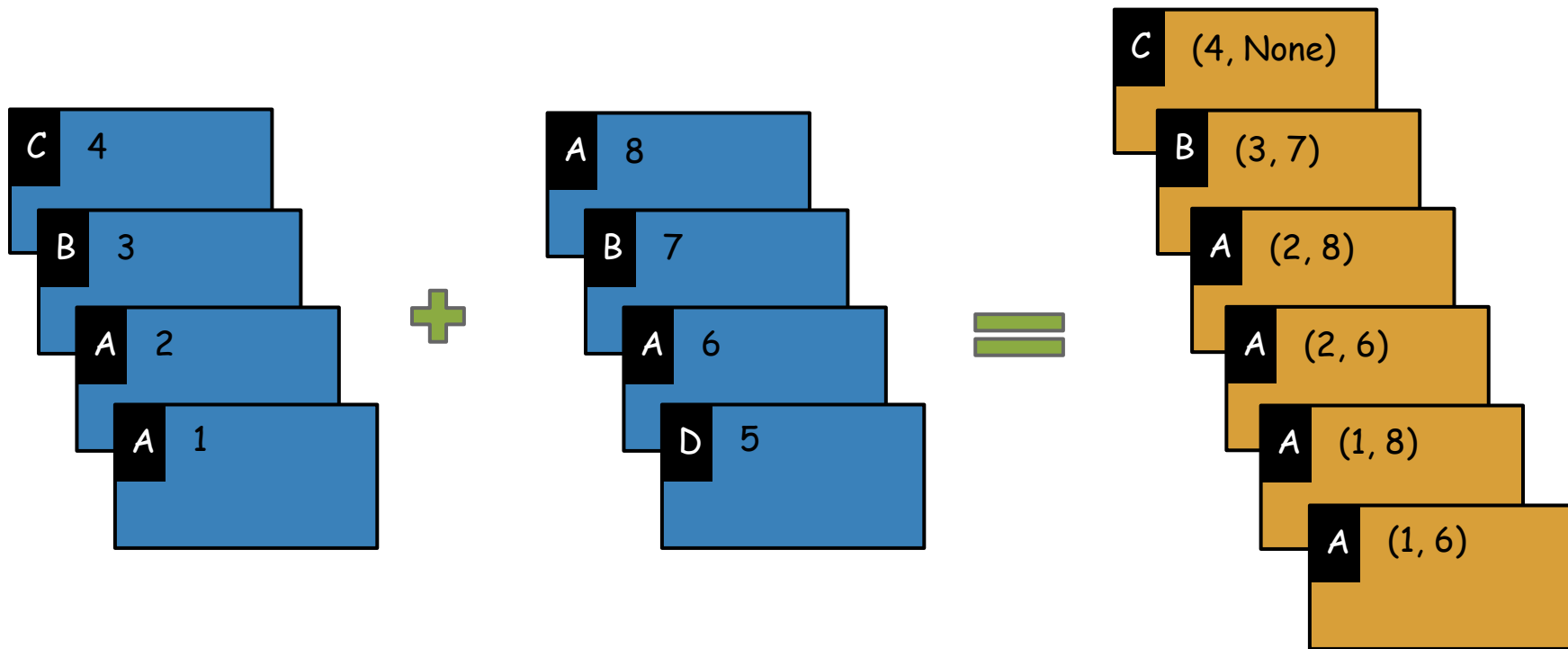


{ A : 3 , B : 2 }

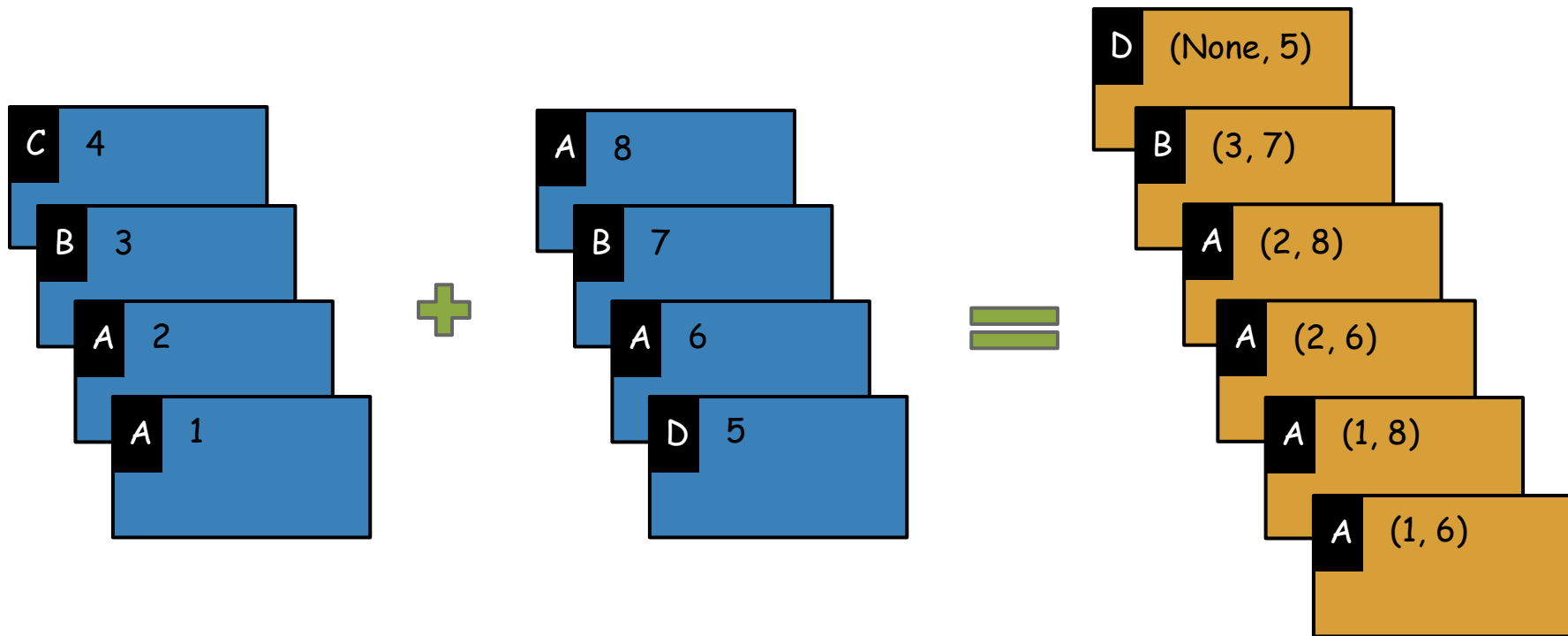
join



leftOuterJoin



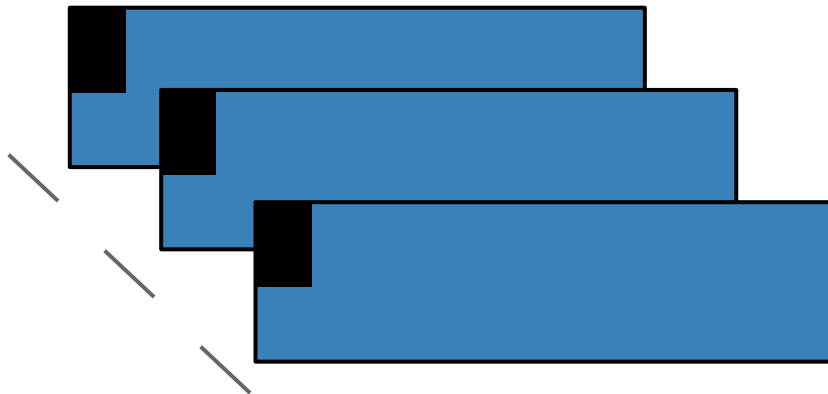
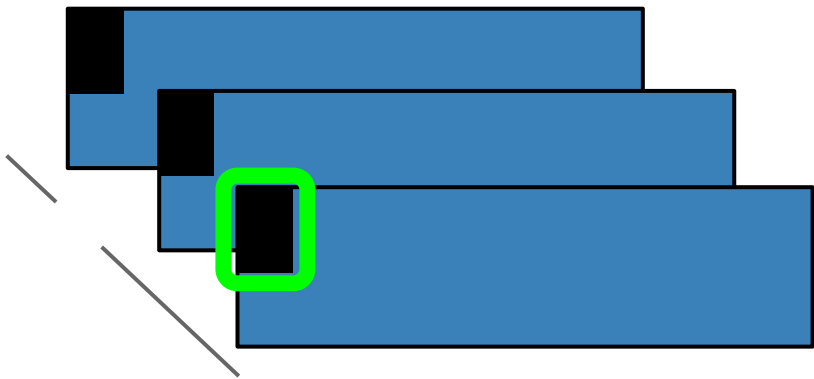
rightOuterJoin



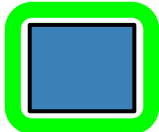

partitionBy

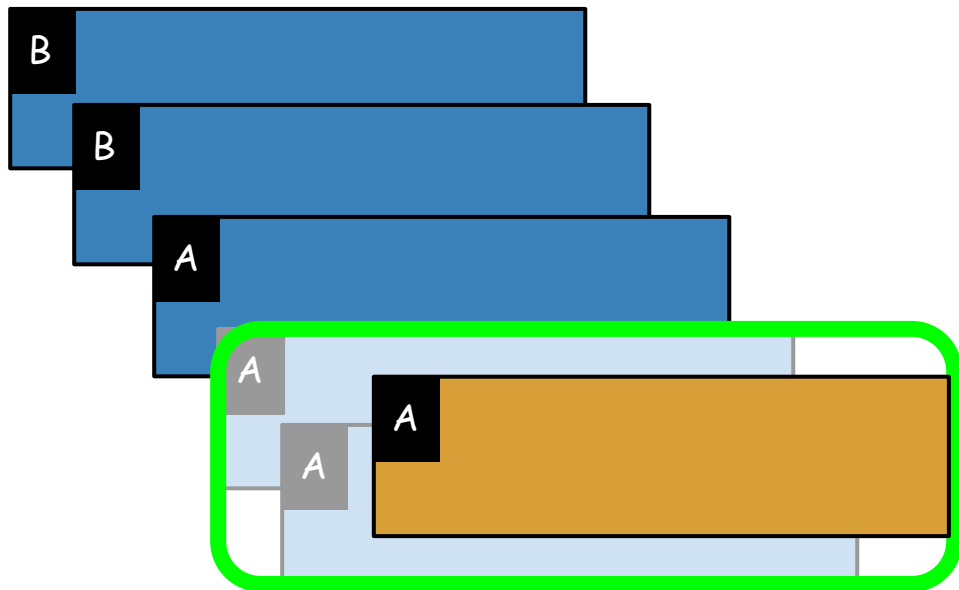
new
partition = key % numPartitions
index

numPartitions = 3



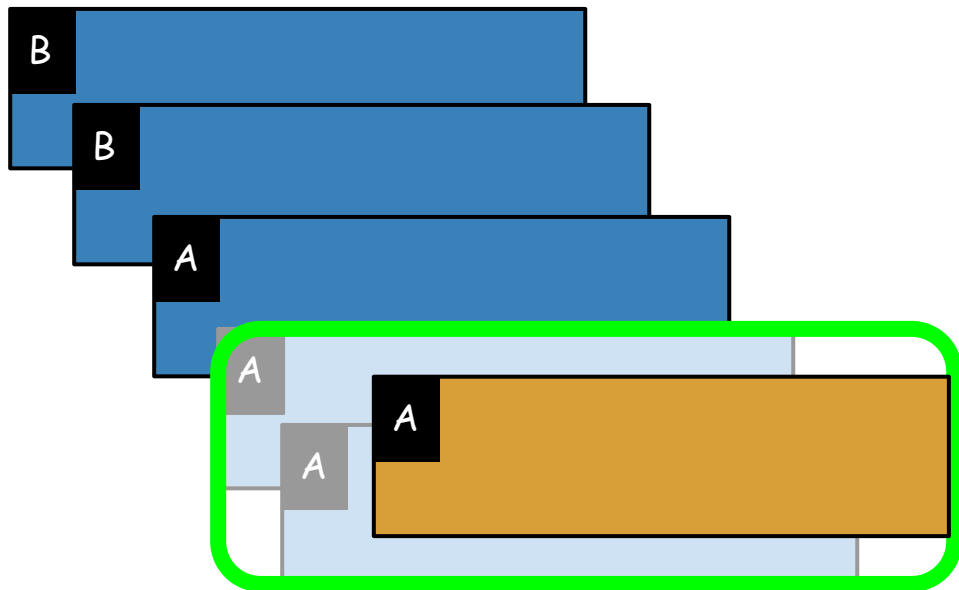
combineByKey

createCombiner =  → 




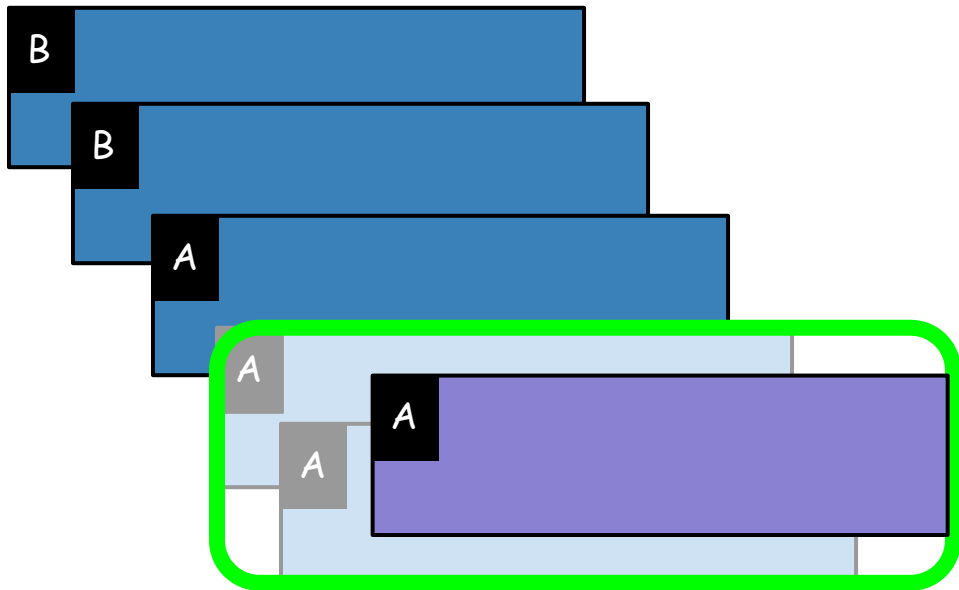
aggregateByKey

zeroValue =

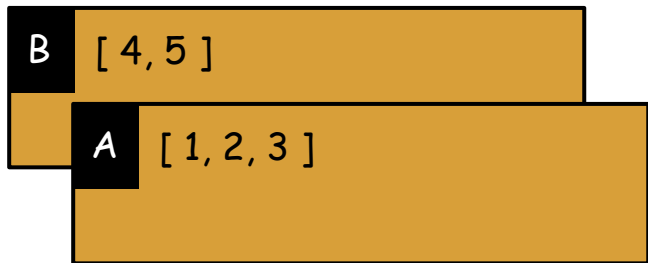
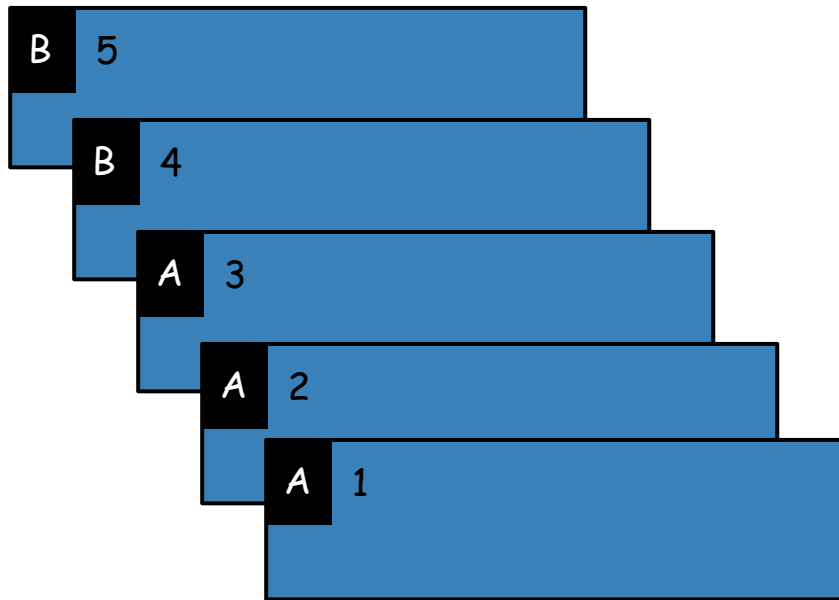


foldByKey

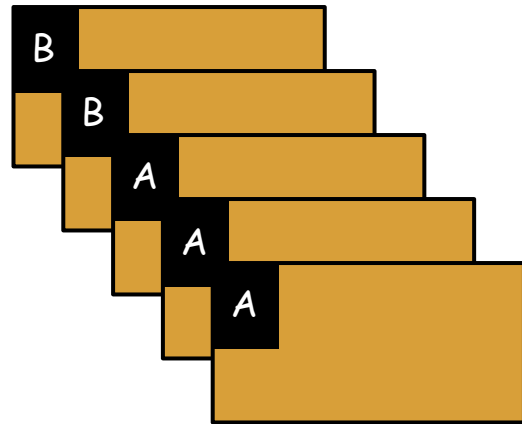
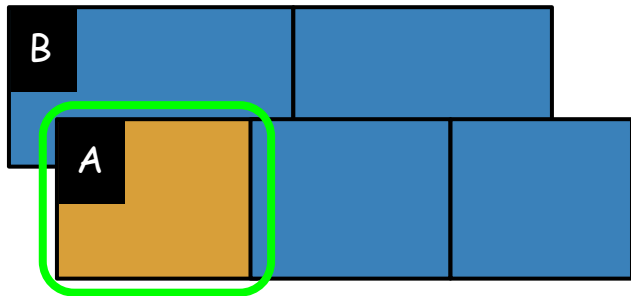
zeroValue = 



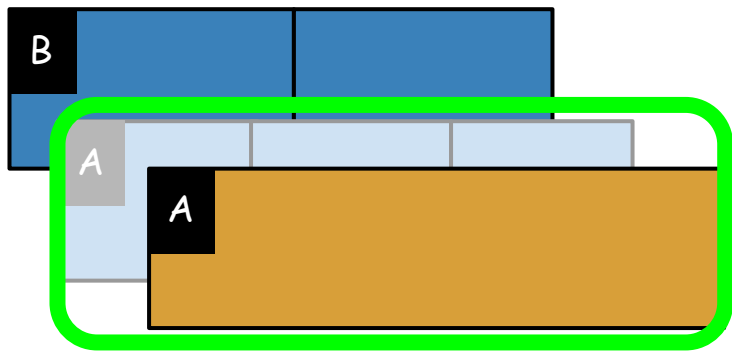
groupByKey



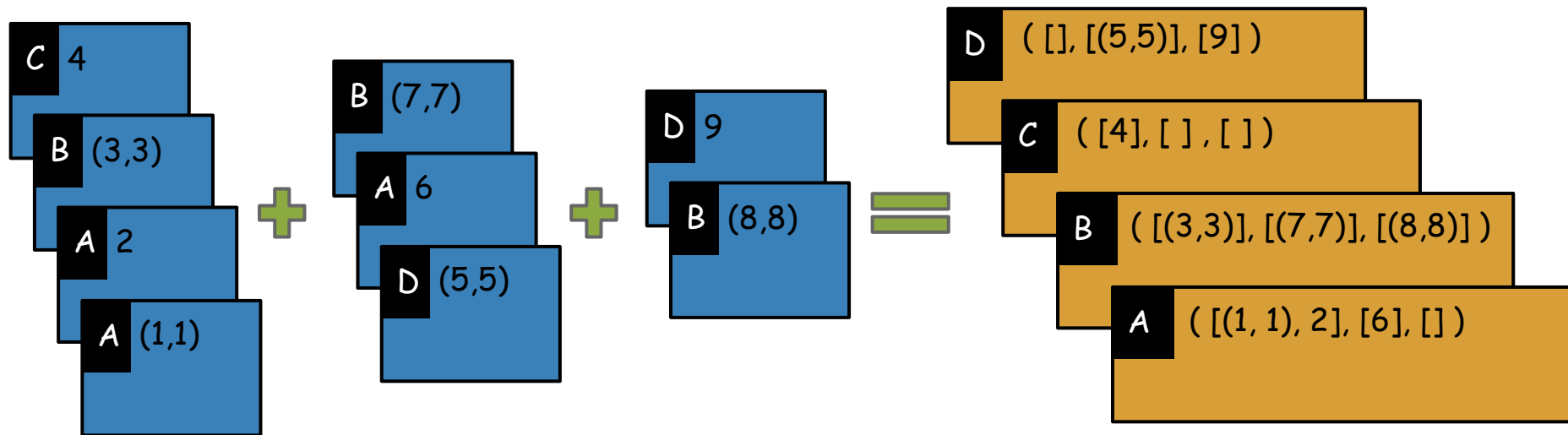
flatMapValues



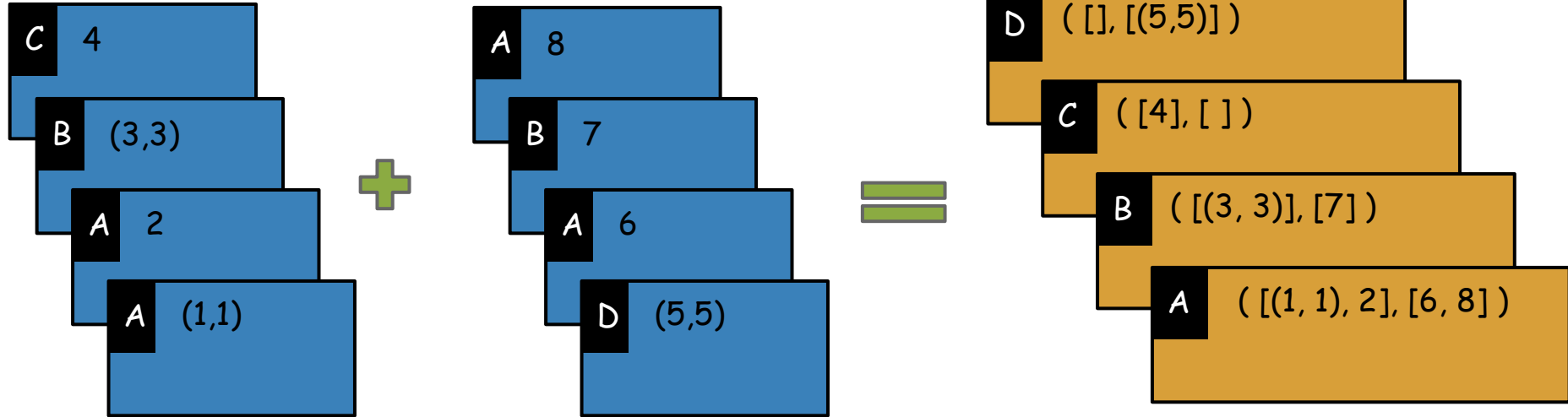
mapValues



groupWith

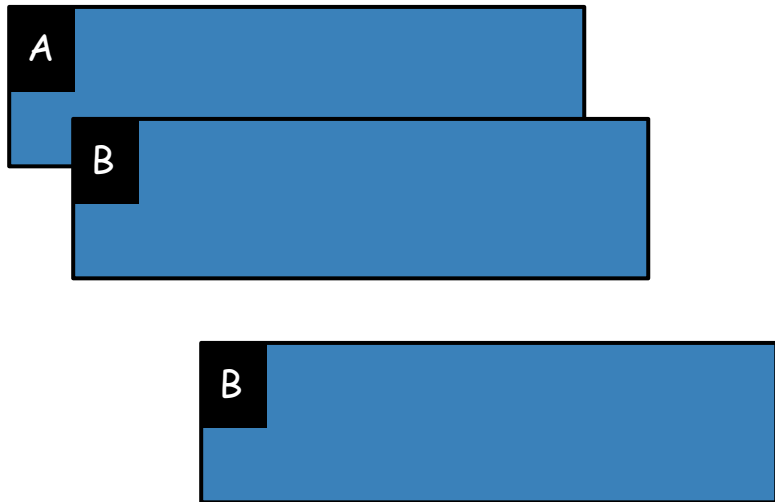
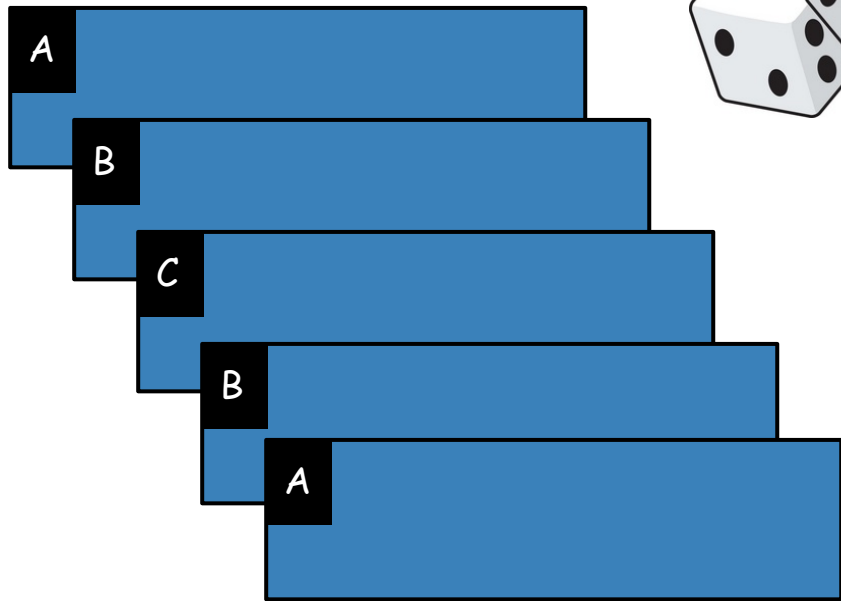
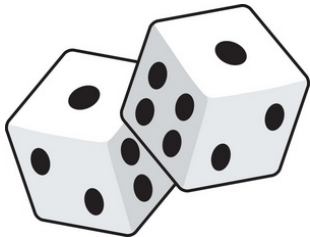


cogroup

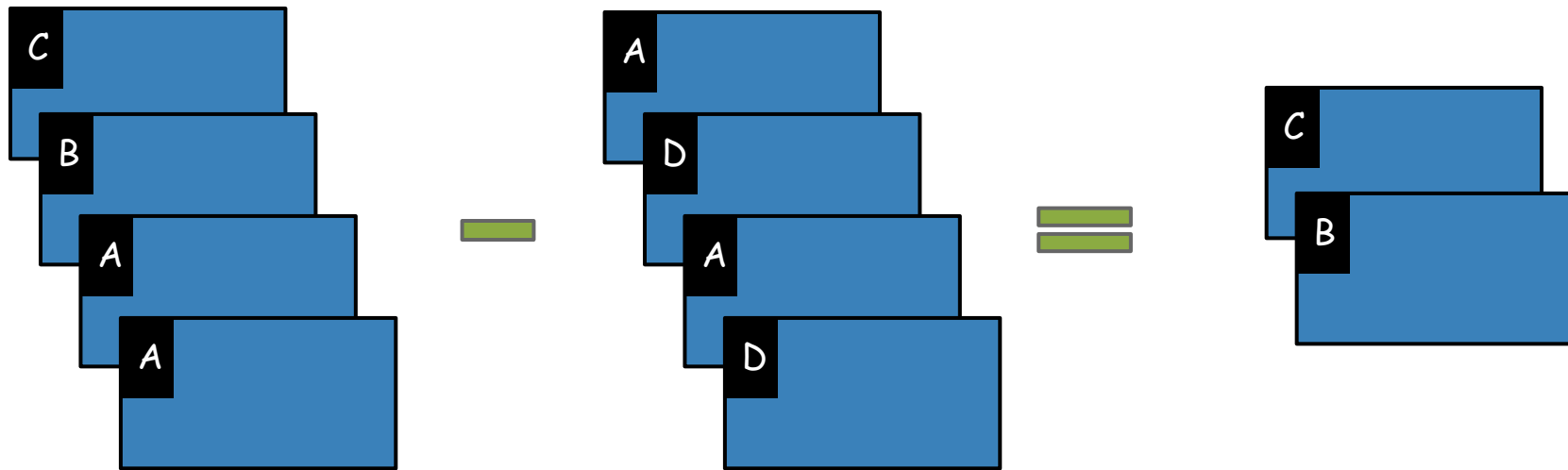


sampleByKey

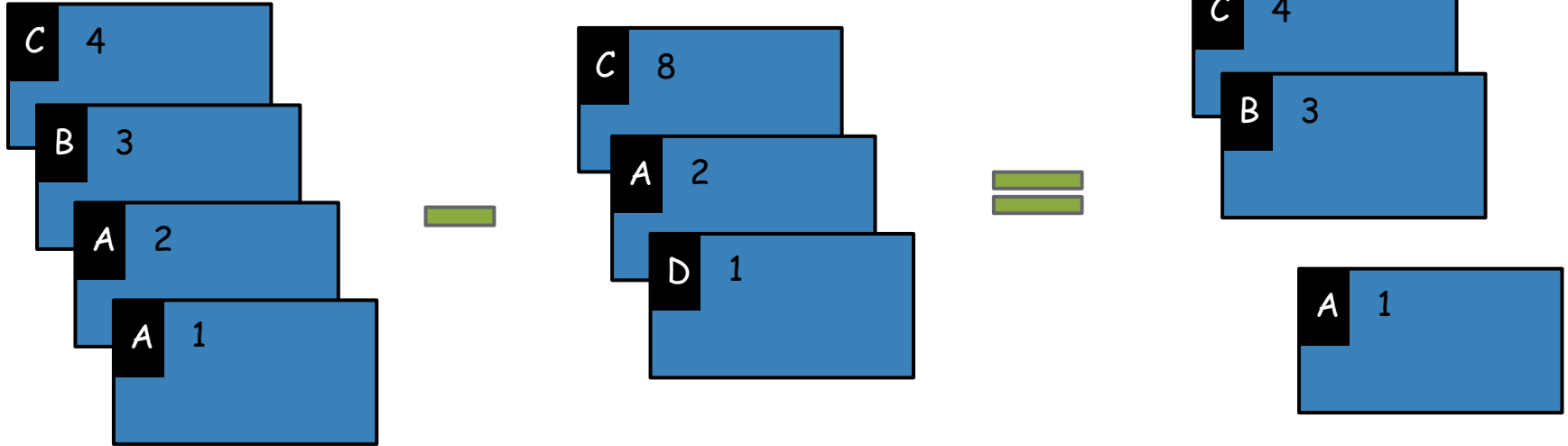
fractions = {'A': 0.5, 'B': 1, 'C': 0.2}



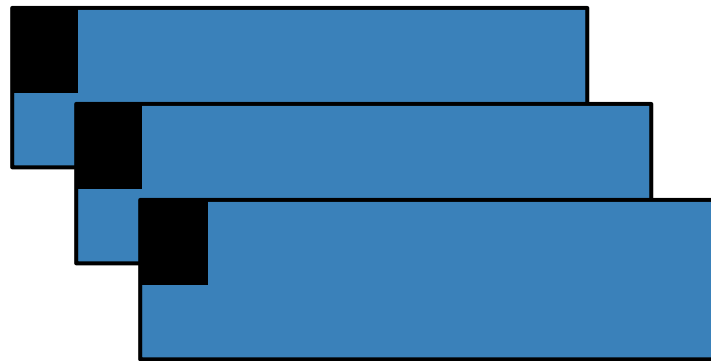
subtractByKey



subtract

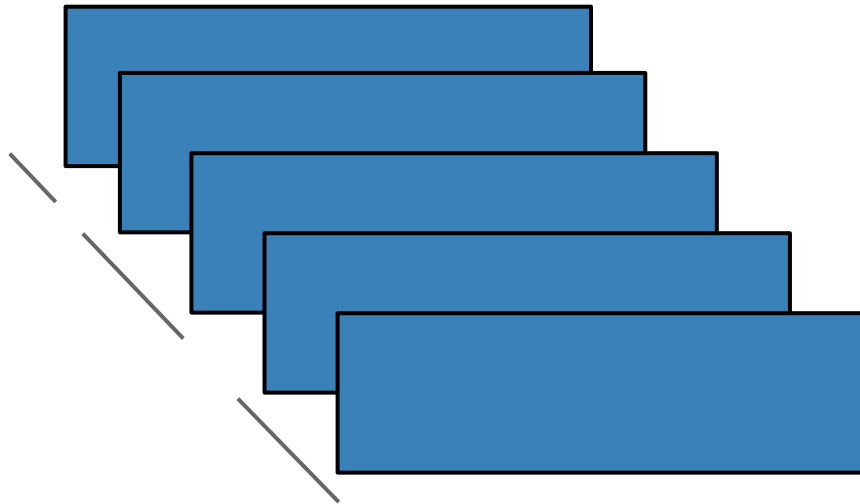
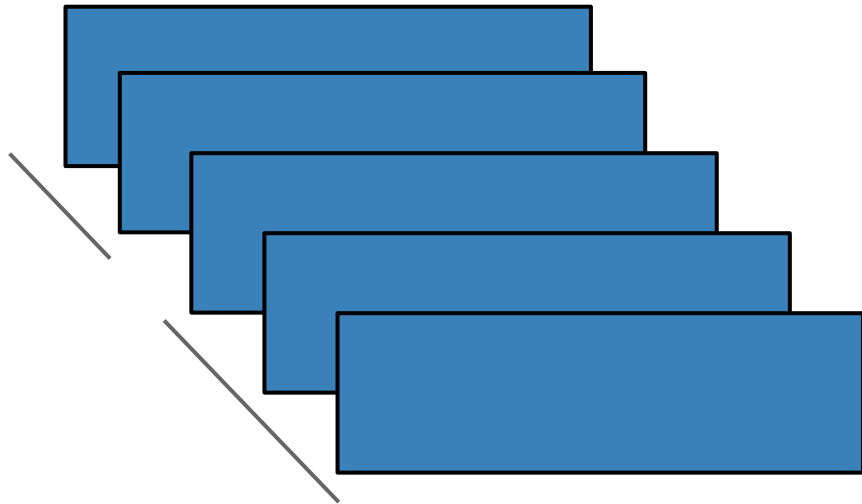


keyBy



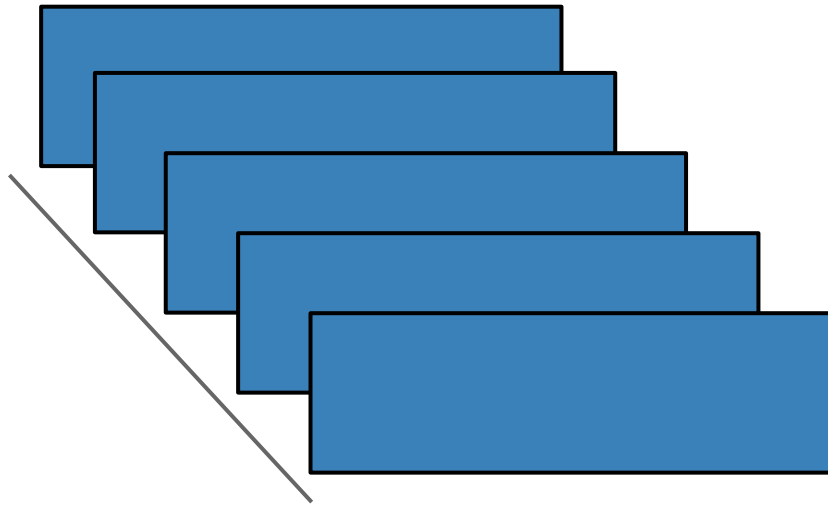
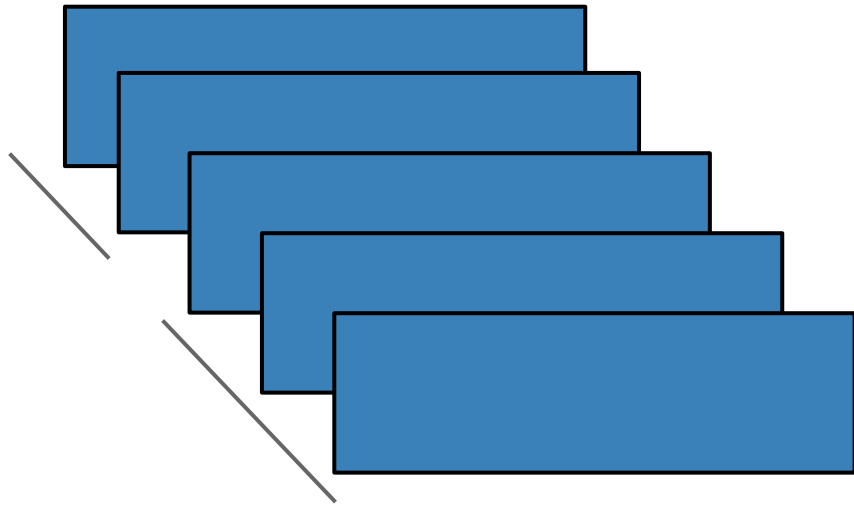
repartition

numPartitions = 3

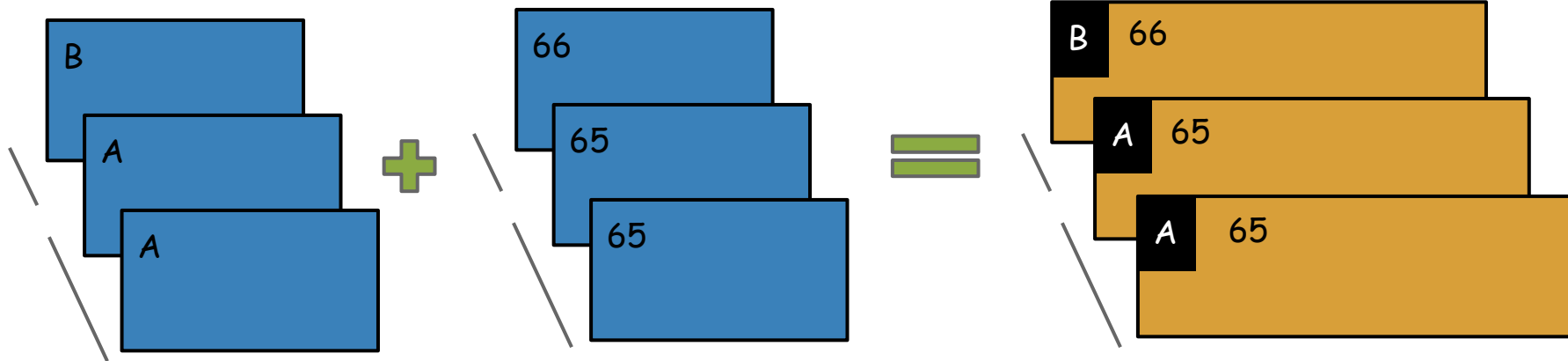


coalesce

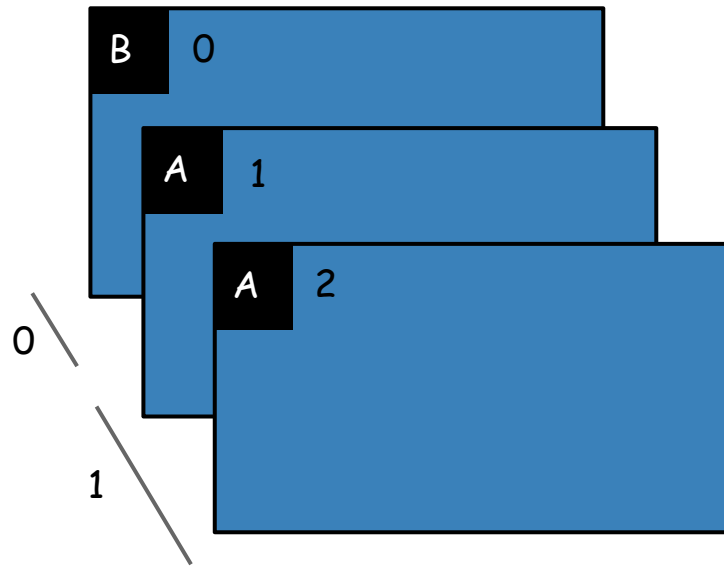
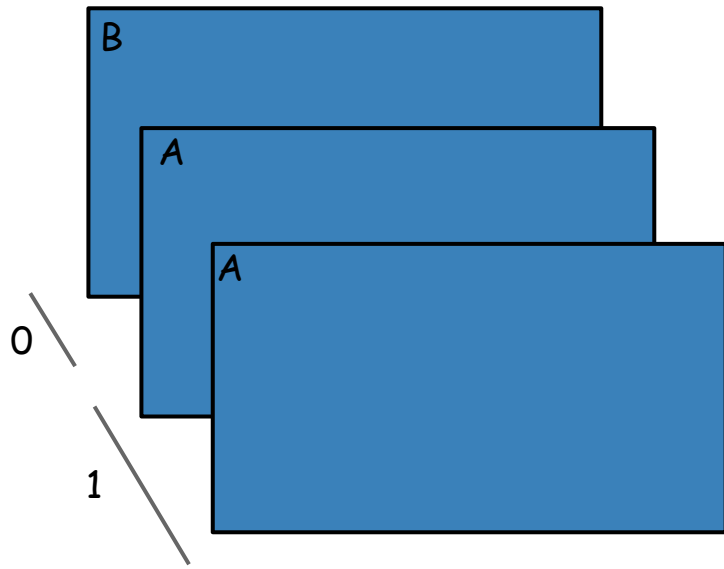
numPartitions = 1



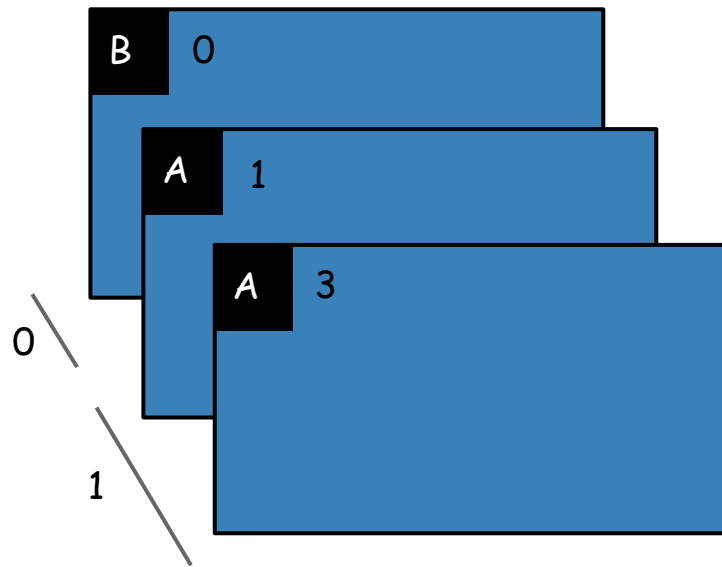
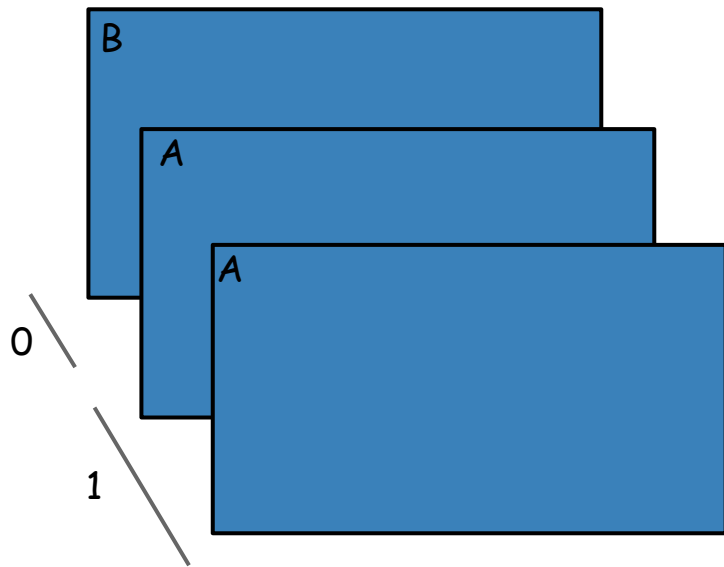
zip



zipWithIndex



zipWithUniqueID



$$\text{uniqueId} = \text{element index} * \text{\#partitions} + \text{partition index}$$