03 - Core Spark

Demi Ben-Ari

Panorays

Spark Languages

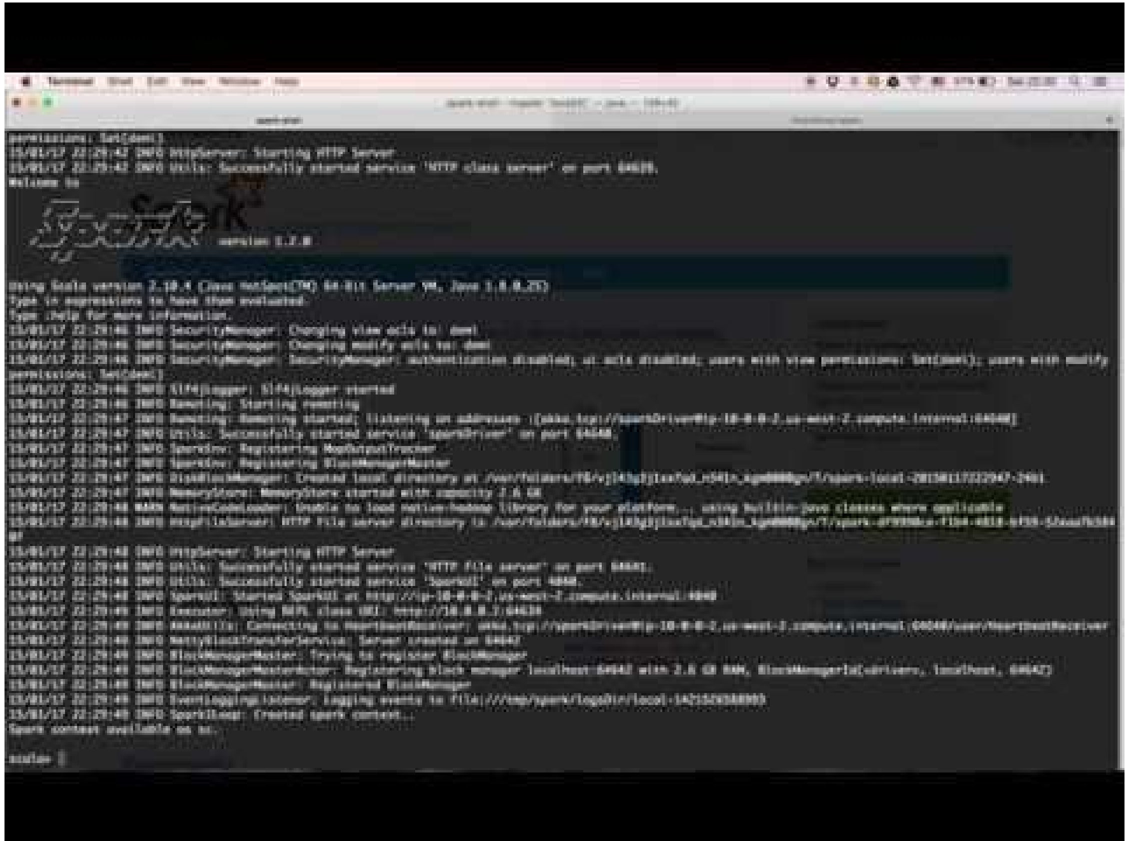
Scala

python™

Java™

R

Spark Word Count example - Spark Shell

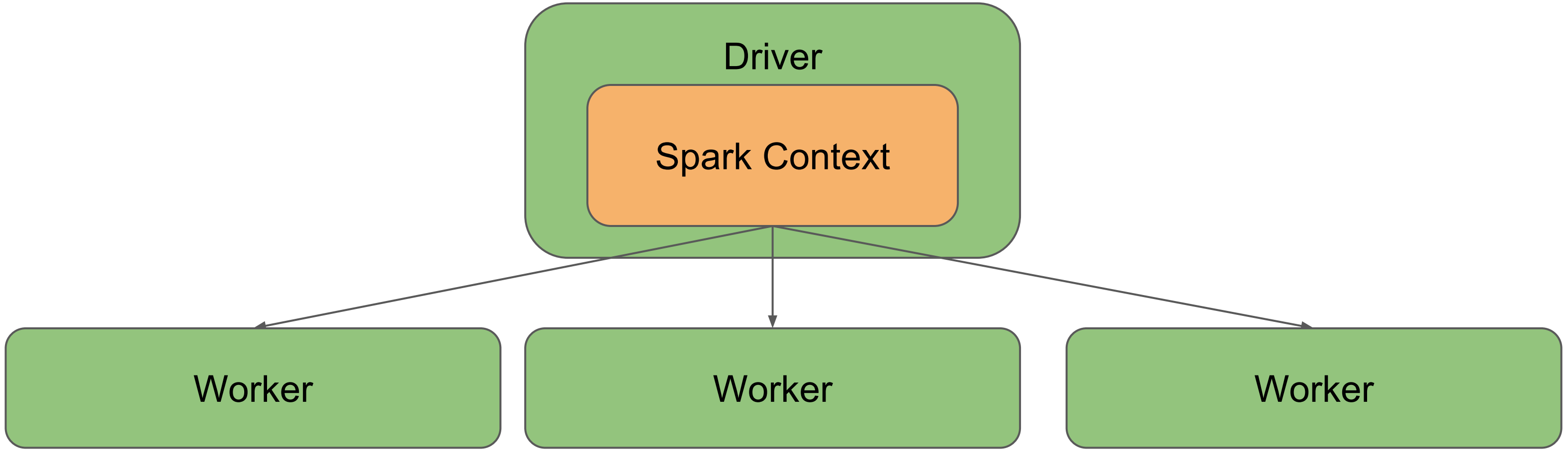


```
permissions: Set(doml)
15/01/17 22:29:42 INFO HttpServer: Starting HTTP Server
15/01/17 22:29:42 INFO Utils: Successfully started service 'HTTP class server' on port 64639.
Welcome to
      Spark  version 1.2.0

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_25)
Type in expressions to have them evaluated.
Type :help for more information.
15/01/17 22:29:46 INFO SecurityManager: Changing view acls to: doml
15/01/17 22:29:46 INFO SecurityManager: Changing modify acls to: doml
15/01/17 22:29:46 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(doml); users with modify
permissions: Set(doml)
15/01/17 22:29:46 INFO Slf4jLogger: Slf4jLogger started
15/01/17 22:29:46 INFO Remoting: Starting remoting
15/01/17 22:29:47 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriver@ip-10-0-0-2.us-west-2.compute.internal:64640]
15/01/17 22:29:47 INFO Utils: Successfully started service 'sparkDriver' on port 64640.
15/01/17 22:29:47 INFO SparkEnv: Registering MapOutputTracker
15/01/17 22:29:47 INFO SparkEnv: Registering BlockManagerMaster
15/01/17 22:29:47 INFO DiskBlockManager: Created local directory at /var/folders/f8/[illegible]/T/spark-local-20150117222947-2461
15/01/17 22:29:47 INFO MemoryStore: MemoryStore started with capacity 2.6 GB
15/01/17 22:29:48 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
15/01/17 22:29:48 INFO HttpFileServer: HTTP File server directory is /var/folders/f8/[illegible]/T/spark-[illegible]
ef
15/01/17 22:29:48 INFO HttpServer: Starting HTTP Server
15/01/17 22:29:48 INFO Utils: Successfully started service 'HTTP file server' on port 64641.
15/01/17 22:29:48 INFO Utils: Successfully started service 'SparkUI' on port 4040.
15/01/17 22:29:48 INFO SparkUI: Started SparkUI at http://ip-10-0-0-2.us-west-2.compute.internal:4040
15/01/17 22:29:49 INFO Executor: Using REPL class URI: http://10.0.0.2:64639
15/01/17 22:29:49 INFO AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp://sparkDriver@ip-10-0-0-2.us-west-2.compute.internal:64640/user/HeartbeatReceiver
15/01/17 22:29:49 INFO NettyBlockTransferService: Server created on 64642
15/01/17 22:29:49 INFO BlockManagerMaster: Trying to register BlockManager
15/01/17 22:29:49 INFO BlockManagerMasterActor: Registering block manager localhost:64642 with 2.6 GB RAM, BlockManagerId(<driver>, localhost, 64642)
15/01/17 22:29:49 INFO BlockManagerMaster: Registered BlockManager
15/01/17 22:29:49 INFO EventLoggingListener: Logging events to file:///tmp/spark/logDir/local-1421526588993
15/01/17 22:29:49 INFO SparkILoop: Created spark context..
Spark context available as sc.

scala>
```

Module Overview

- RDD
- Transformations
- Actions

Spark Mechanics



Panorays

Spark Mechanics

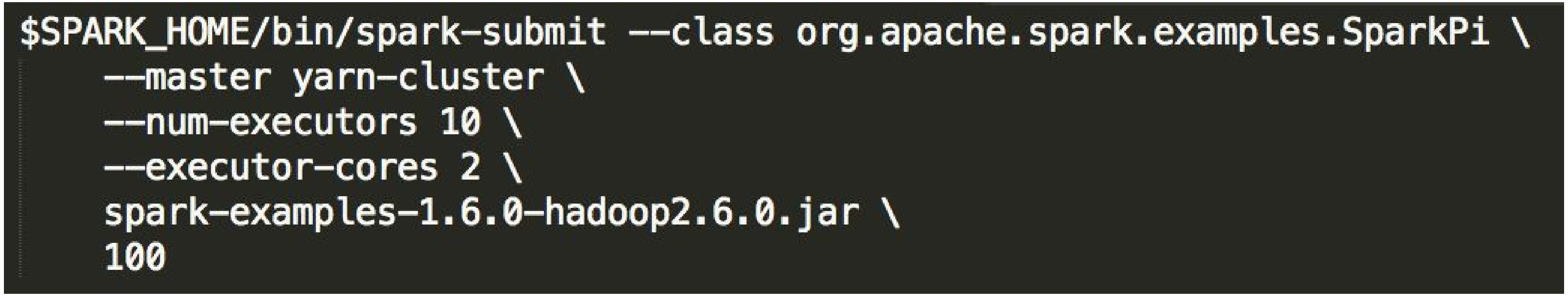
- Task Creator
  - Builds execution graph to be sent to each worker
- Scheduler
  - Scheduling all of the task across the nodes
- Data locality
  - Sending the work to the data to avoid moving data across the cluster
- Fault tolerance
  - Monitoring the tasks for any failures so it can trigger the task on a different node

Spark Application Configuration

Priority and Hierarchy (From most final to the most general):

1) Code
2) spark-submit --Flags
3) app.properties file ([app])
4) defaults (spark-default.sh)

spark-submit example



```
$SPARK_HOME/bin/spark-submit --class org.apache.spark.examples.SparkPi \
    --master yarn-cluster \
    --num-executors 10 \
    --executor-cores 2 \
    spark-examples-1.6.0-hadoop2.6.0.jar \
    100
```

RDD - Resilient Distributed Dataset

- … Collection of elements partitioned across the nodes of the cluster that can be operated on it in parallel…
  - http://spark.apache.org/docs/latest/programming-guide.html#overview
- RDD - Resilient Distributed Dataset
  - Collection similar to a List / Array (Abstraction)
  - It’s actually an Interface (Behind the scenes it’s distributed over the cluster)
- DAG - Directed Acyclic Graph
- Are Immutable!!!

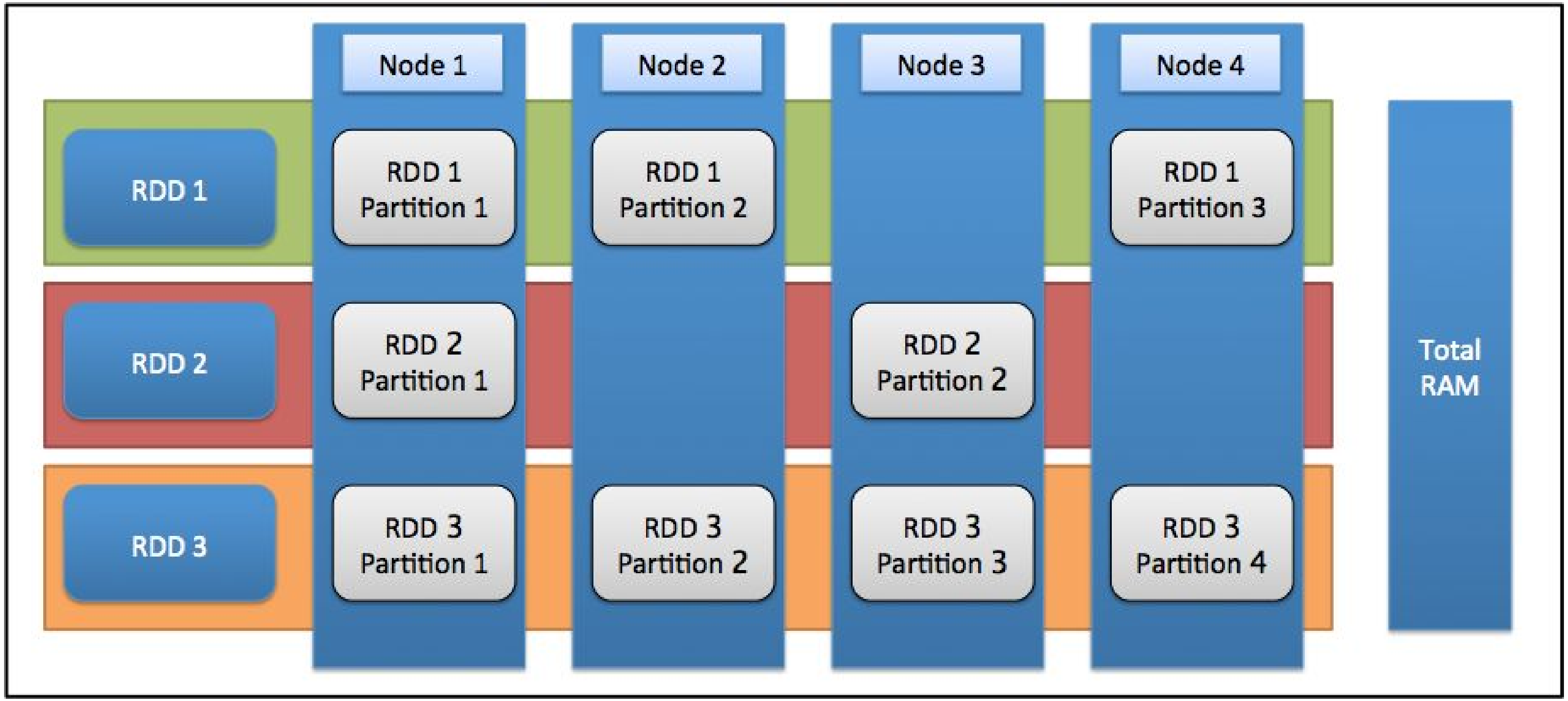
Panorays

RDD - Resilient Distributed Dataset

- Transformations are Lazy evaluated
  - map
  - filter
  - .....
- Actions - Triggers DAG computation
  - collect
  - count
  - reduce

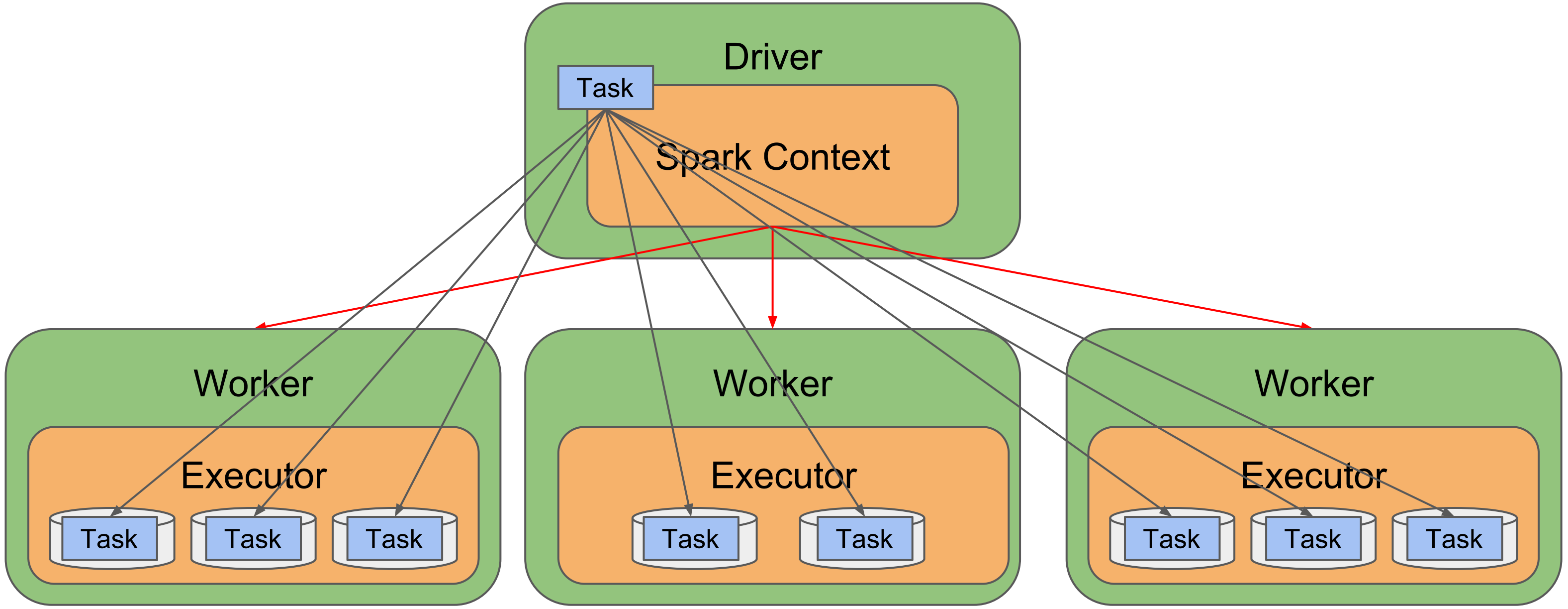
Panorays

What’s really an RDD???



Panorays

Spark Mechanics



Panorays

Lambdas - Anonymous Functions

- How It’s done in Java
- Why not to use it :)
    - Not testable
    - Results in a very verbose main class (inner classes)

Input

Panorays

Input methods

- Local FileSystem
- HDFS
- Cassandra
- Avro
- Parquet

Input methods

- sc.parallelize(<Collection>, <number of partitions>)
- sc.textFile(<path>)
- sc.sequenceFile()
  - Hadoop format
- sc.newAPIHadoopFile()
  - instead of partitioning it accepts Hadoop configuration

Panorays

Transformations

Thanks to Jeffrey Thompson for many of the schemas
https://github.com/jkthompson/pyspark-pictures

Official Documentation

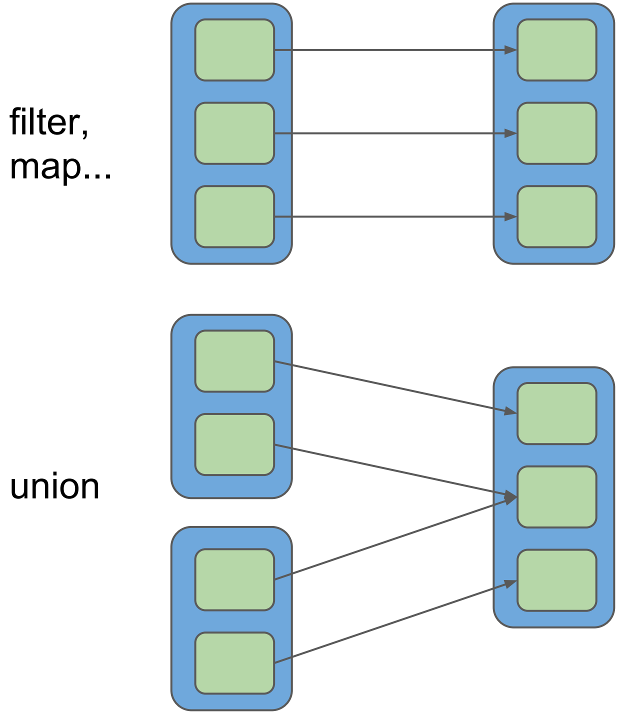
Panorays

Wide and Narrow Transformations

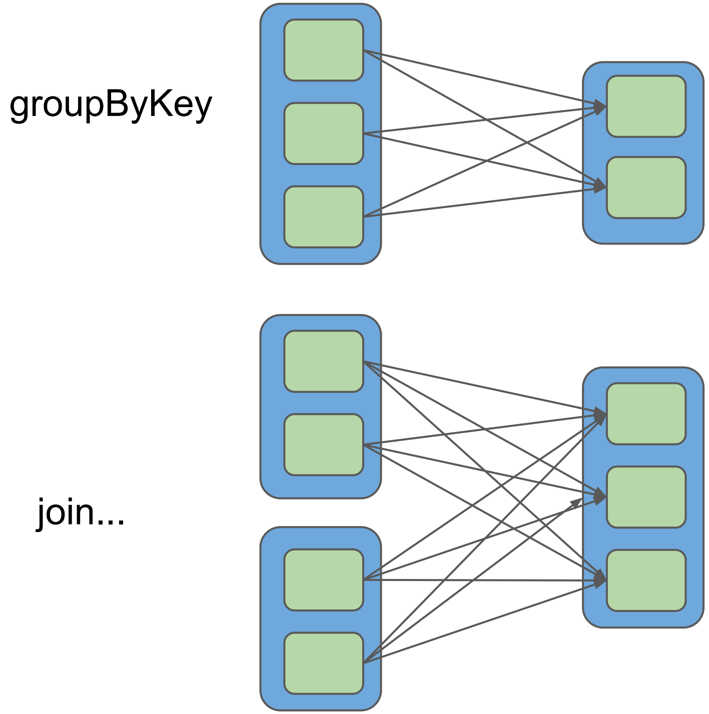
- **Narrow dependency**: each partition of the parent RDD is used by at most one partition of the child RDD. This means the task can be executed locally and we **don't have to shuffle.** (Eg: map, flatMap, Filter, sample)
- **Wide dependency**: multiple child partitions may depend on one partition of the parent RDD. **This means we have to shuffle data** unless the parents are hash-partitioned (Eg: sortByKey, reduceByKey, groupByKey, cogroupByKey, join, cartesian)
- You can read a good blog post about it.

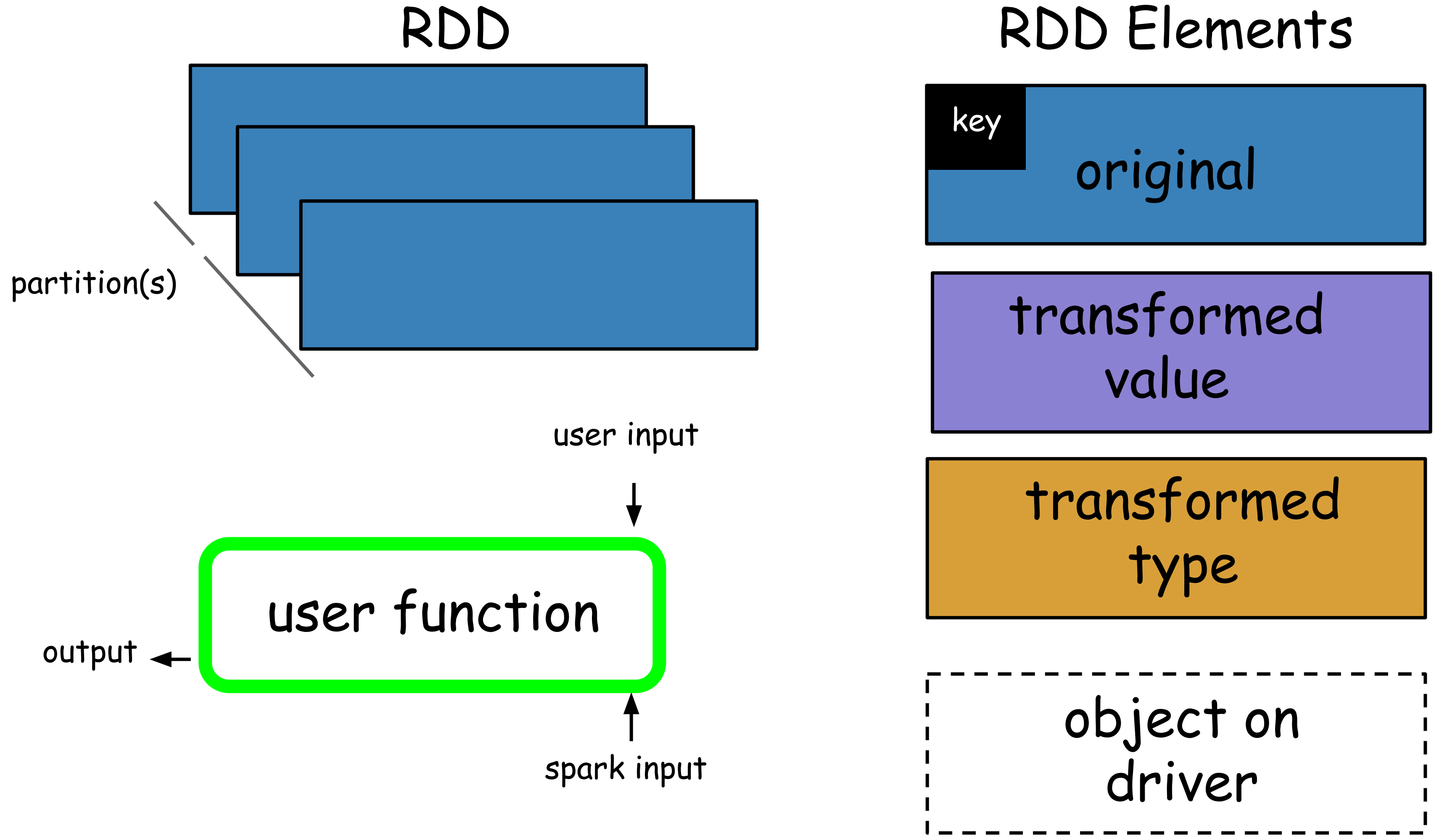
Basic Terms - Wide and Narrow Transformations

Narrow Dependencies:

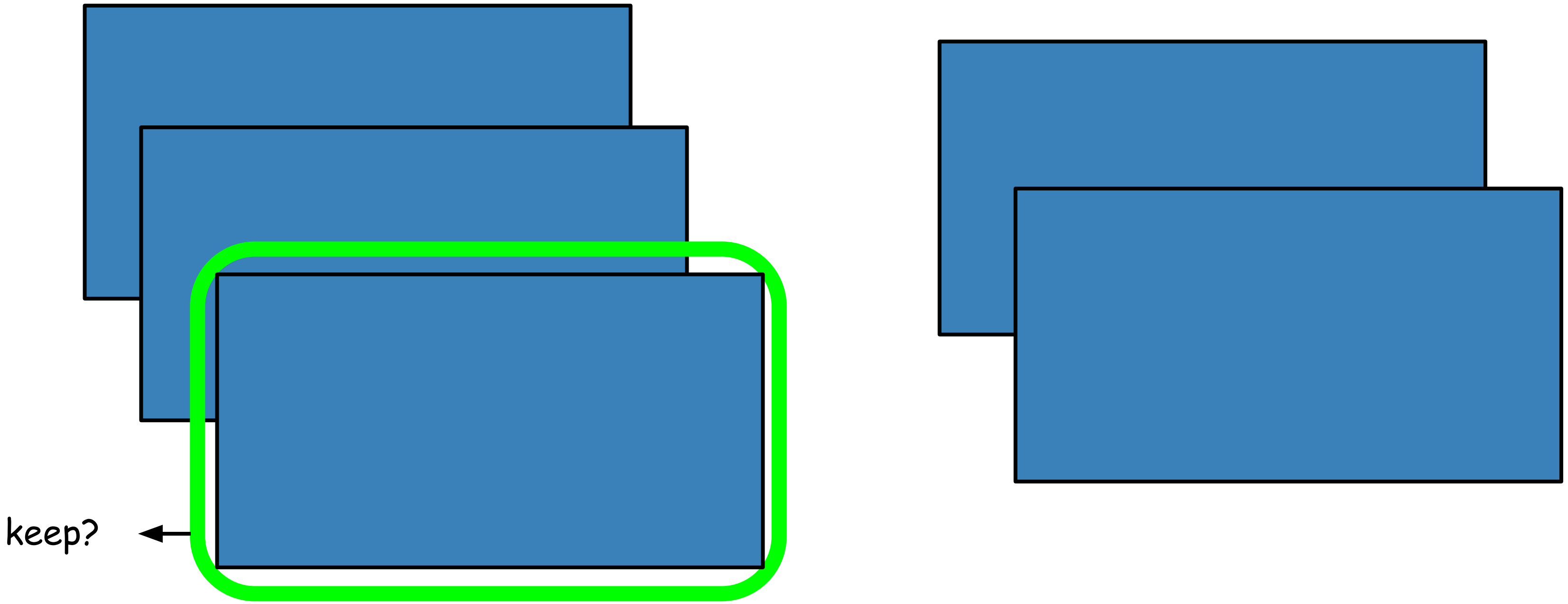


Wide (Shuffle) Dependencies:



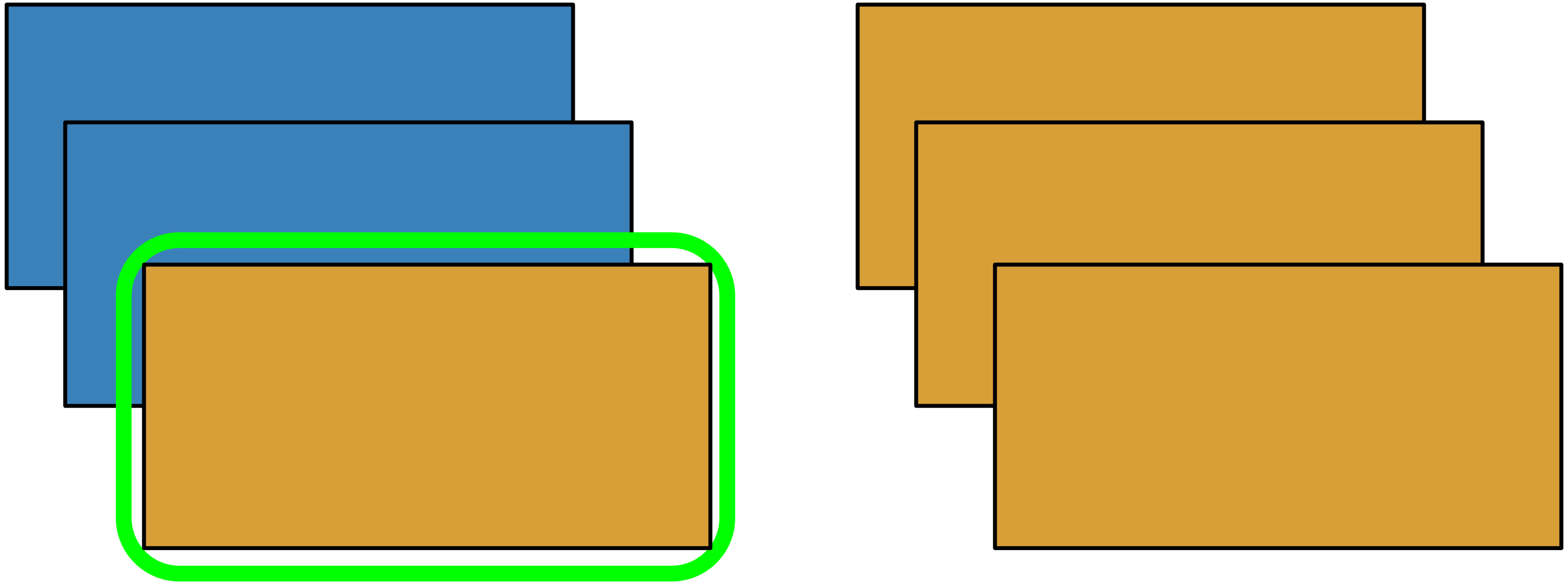

RDD
partition(s)
user input
user function
output
spark input
RDD Elements
key
original
transformed value
transformed type
object on driver

filter

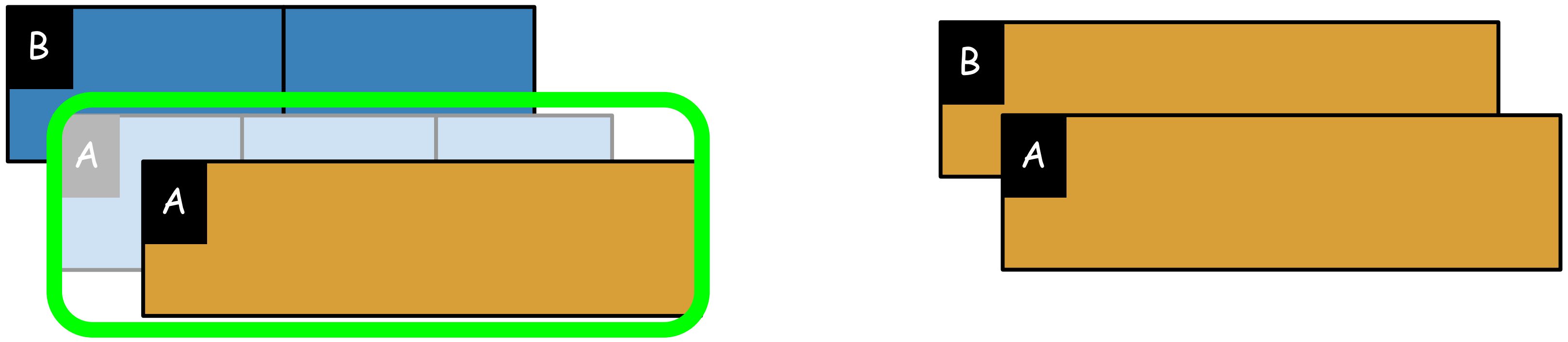


Panorays

map

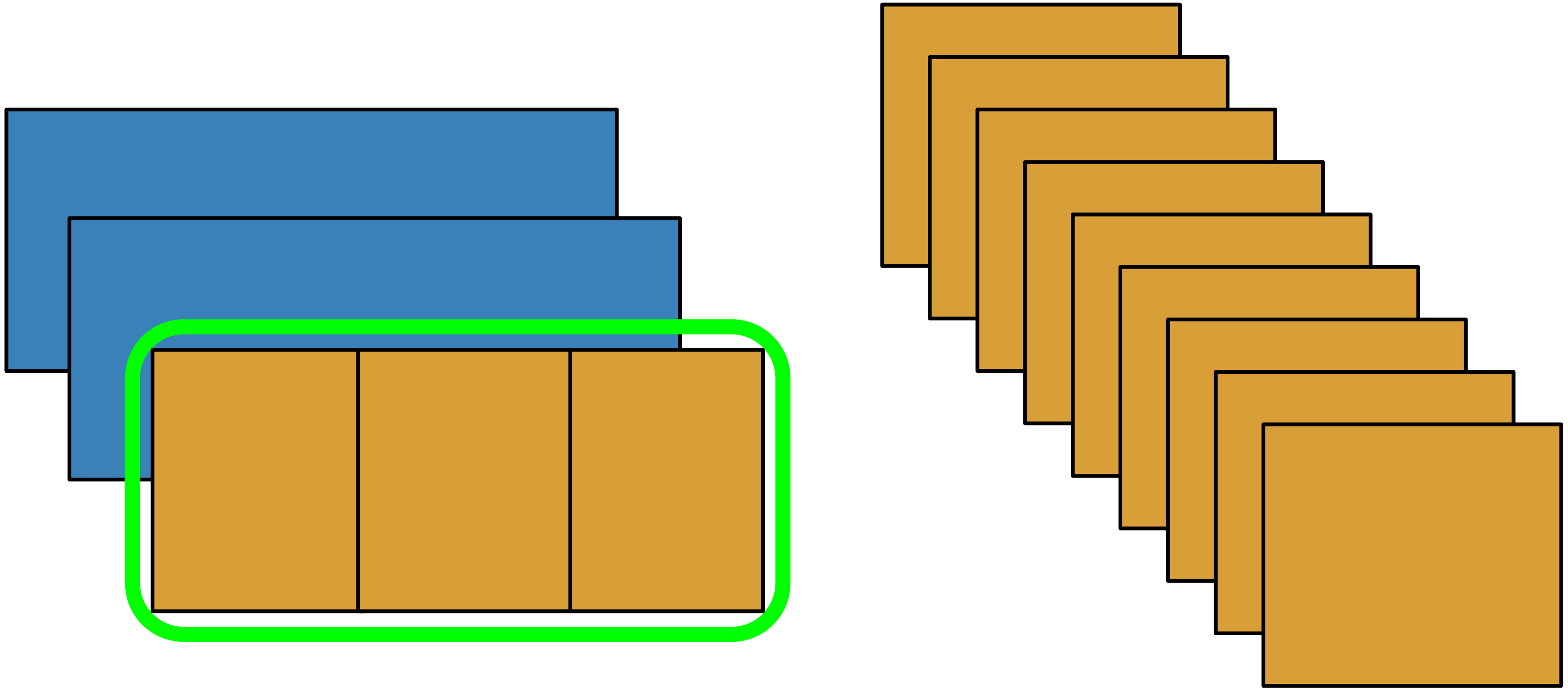


mapValues



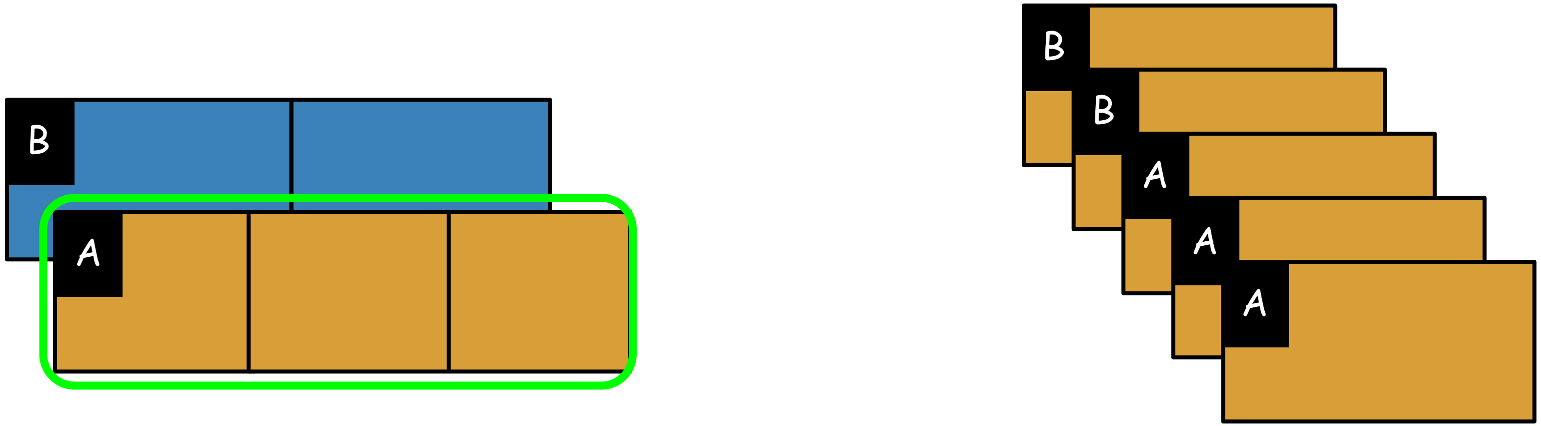
Panorays

flatMap



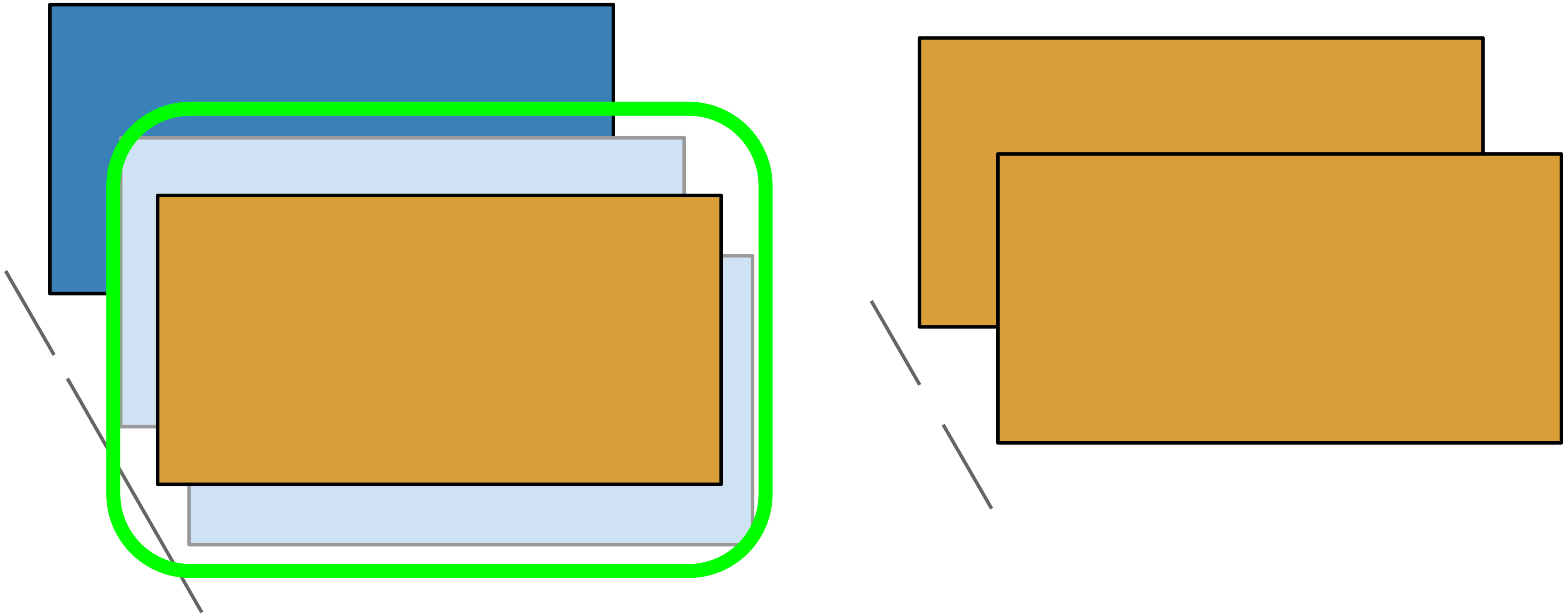
Panorays

flatMapValue



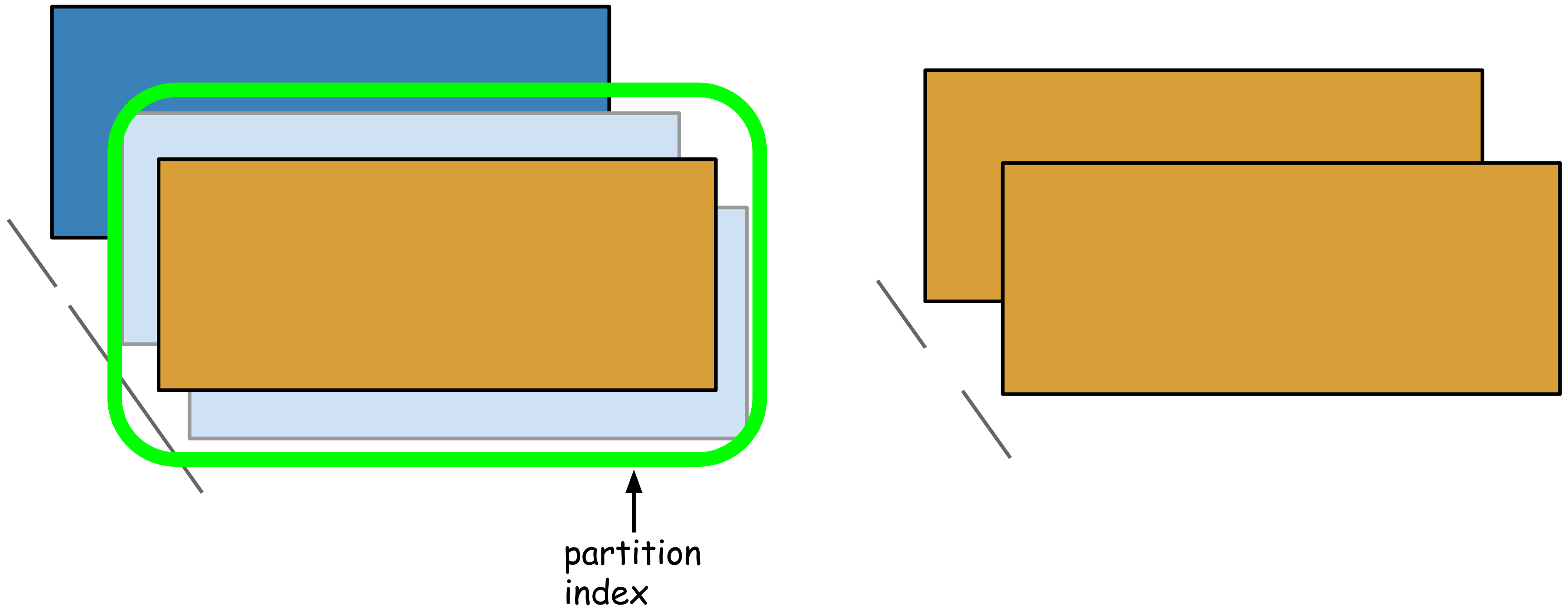
Panorays

mapPartitions

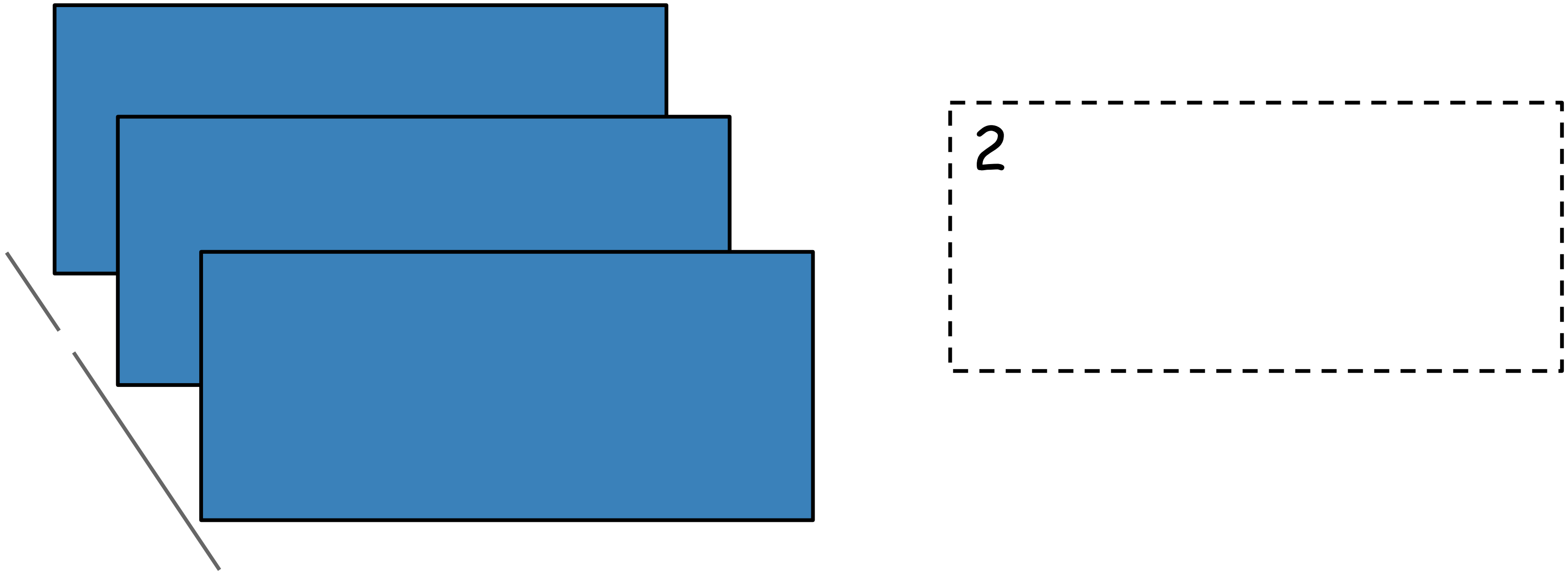


Panorays

mapPartitionsWithIndex

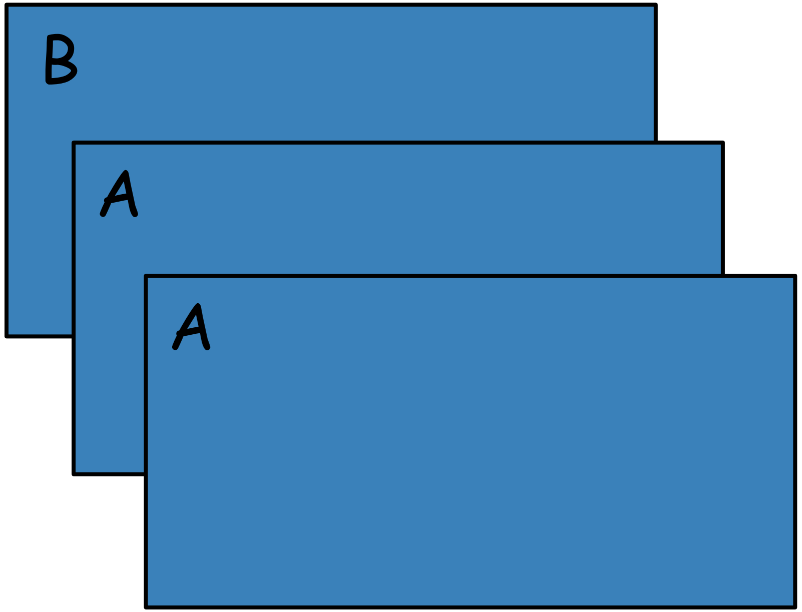


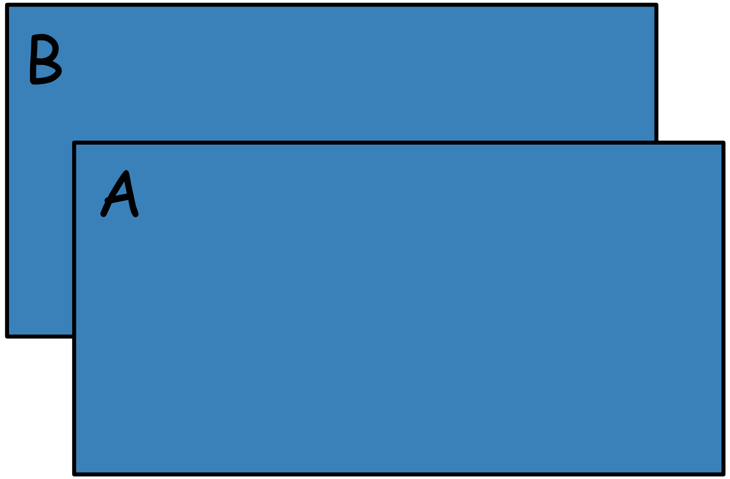
getNumPartitions



Panorays

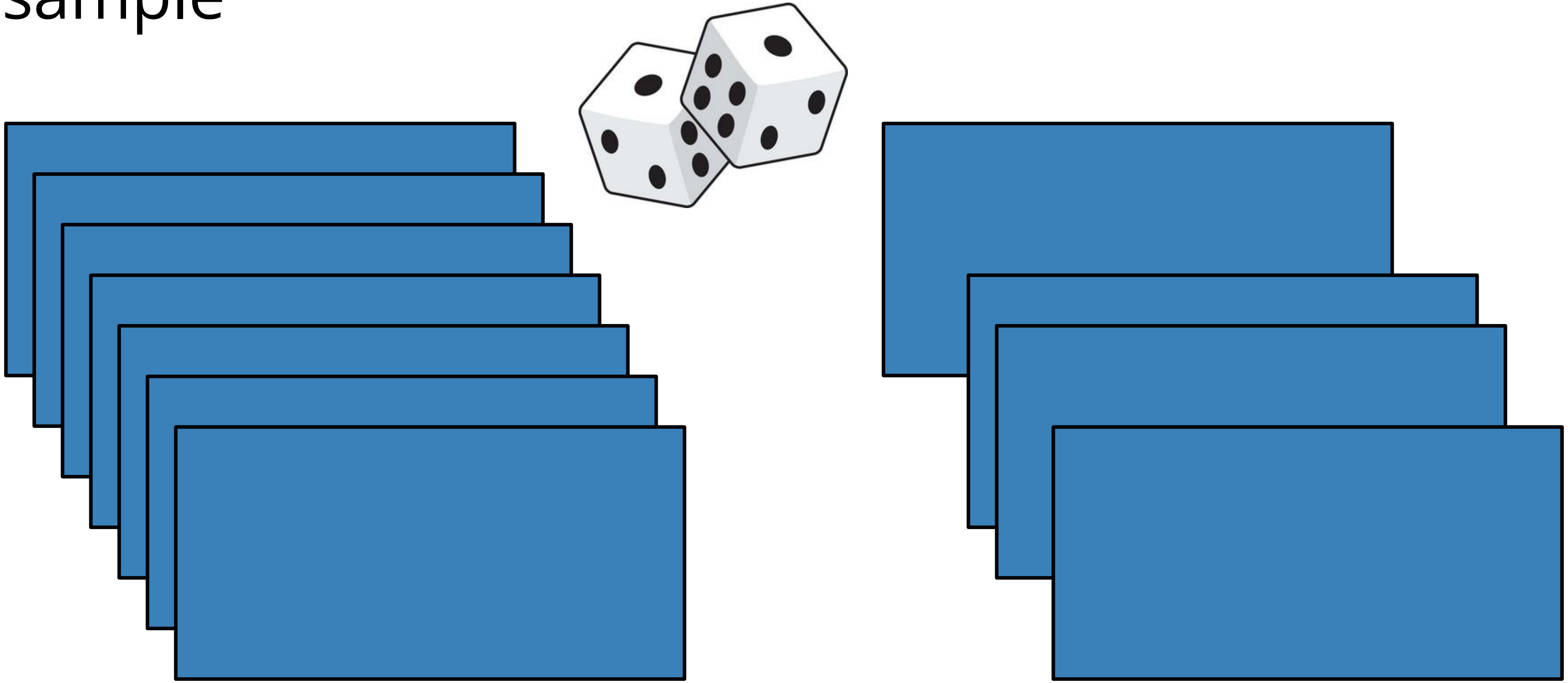
distinct





Panorays

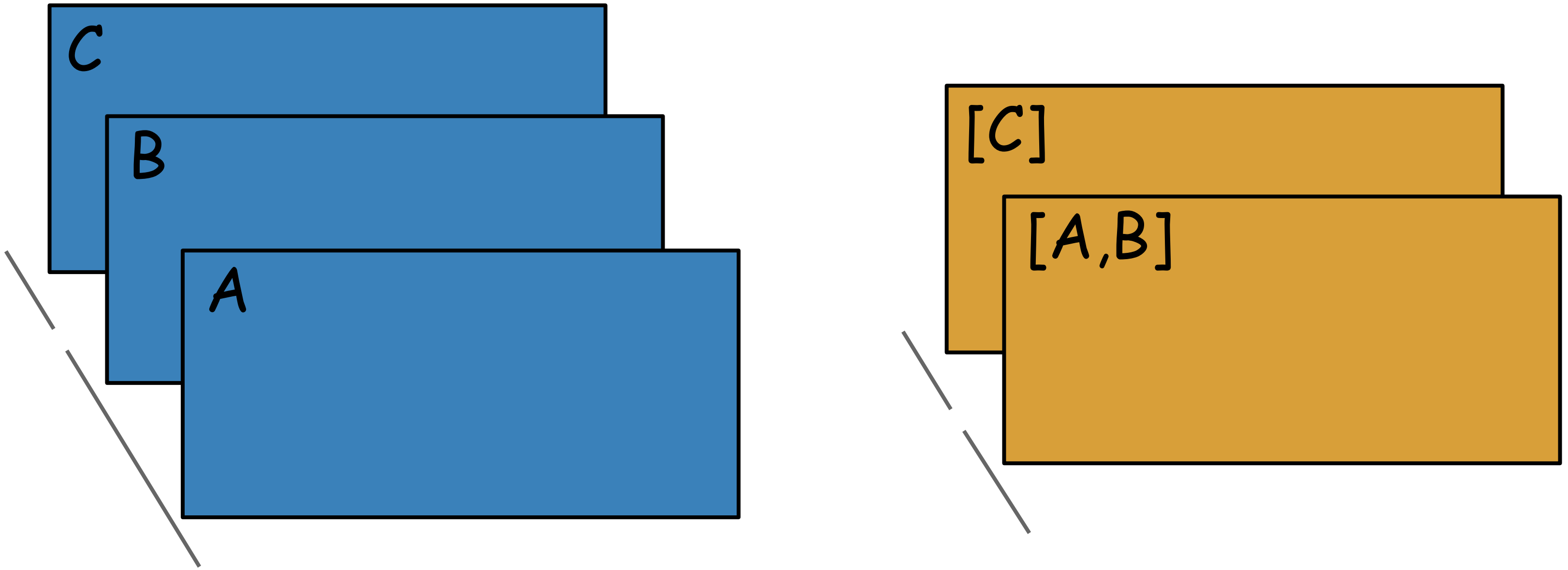
sample



Panorays

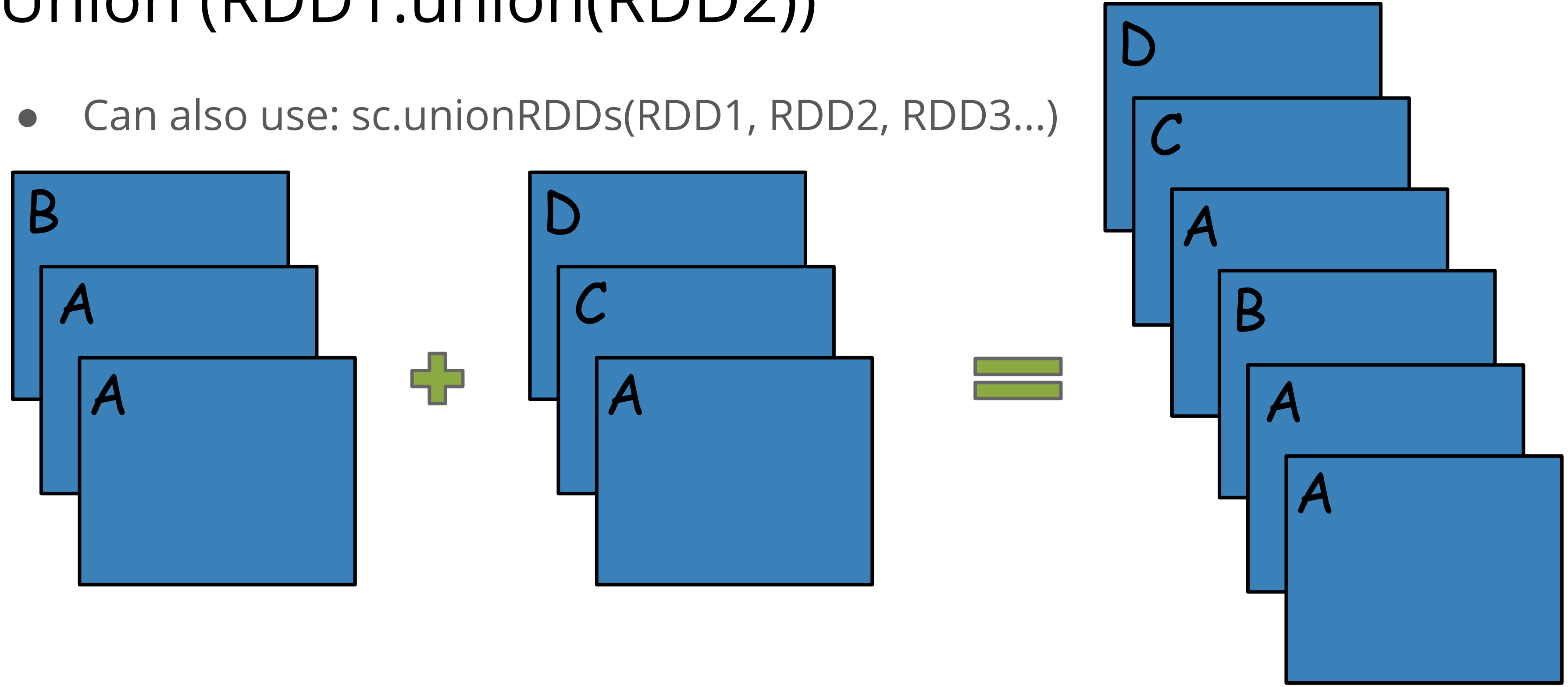
glom

- I have never used it :)

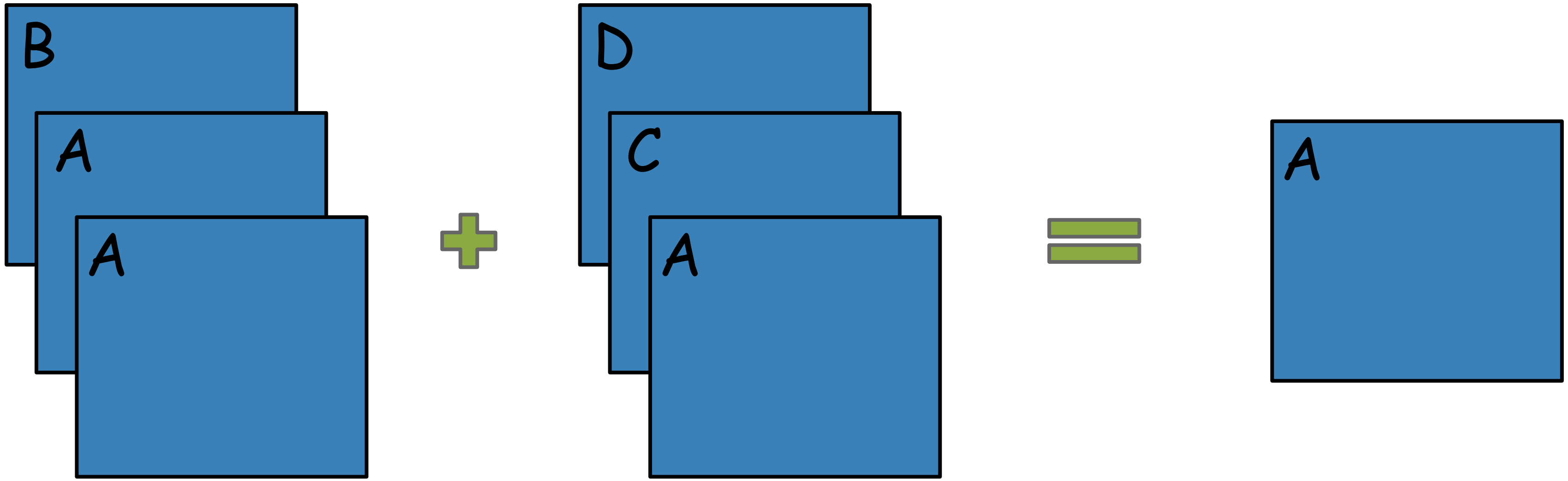


Union (RDD1.union(RDD2))

- Can also use: sc.unionRDDs(RDD1, RDD2, RDD3...)



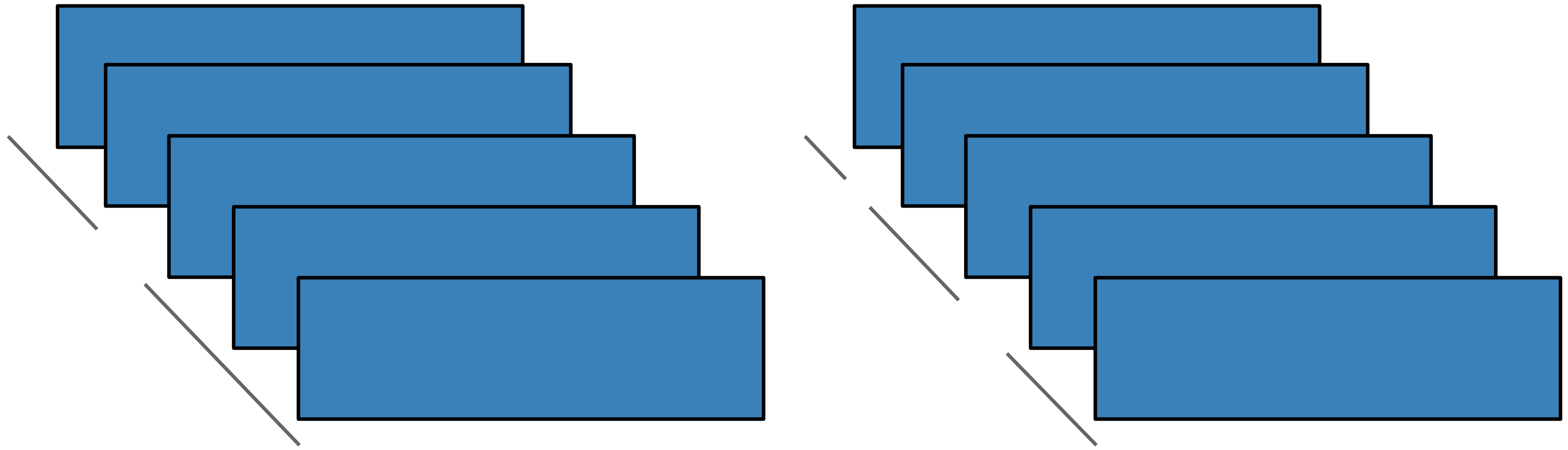
Intersection (RDD1.intersection(RDD2))



Panorays

repartition

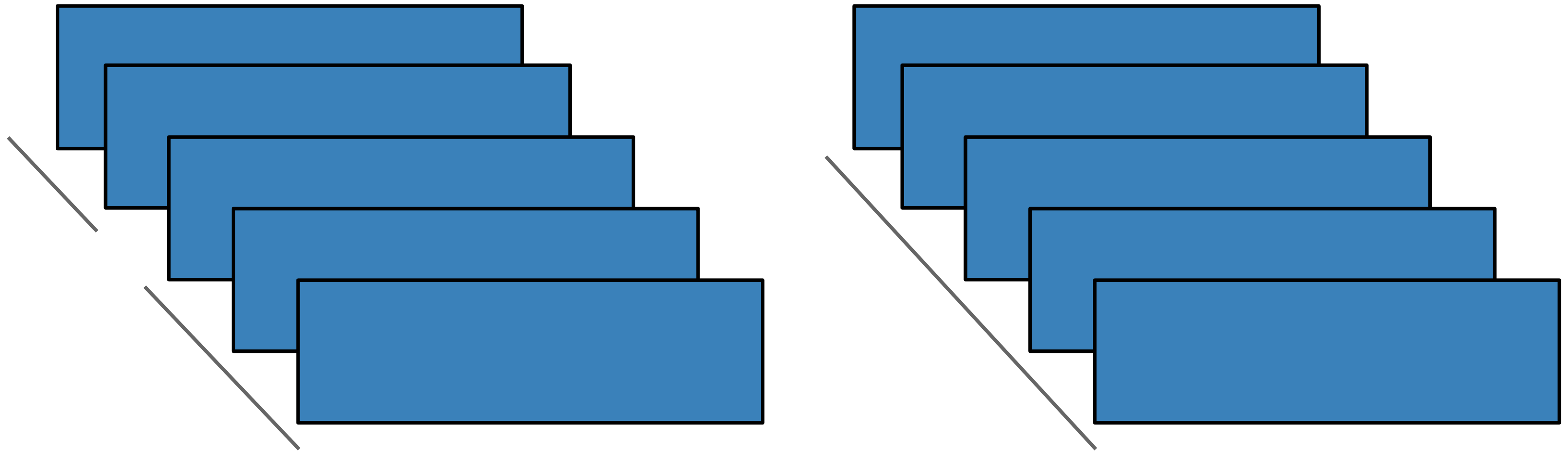
numPartitions = 3



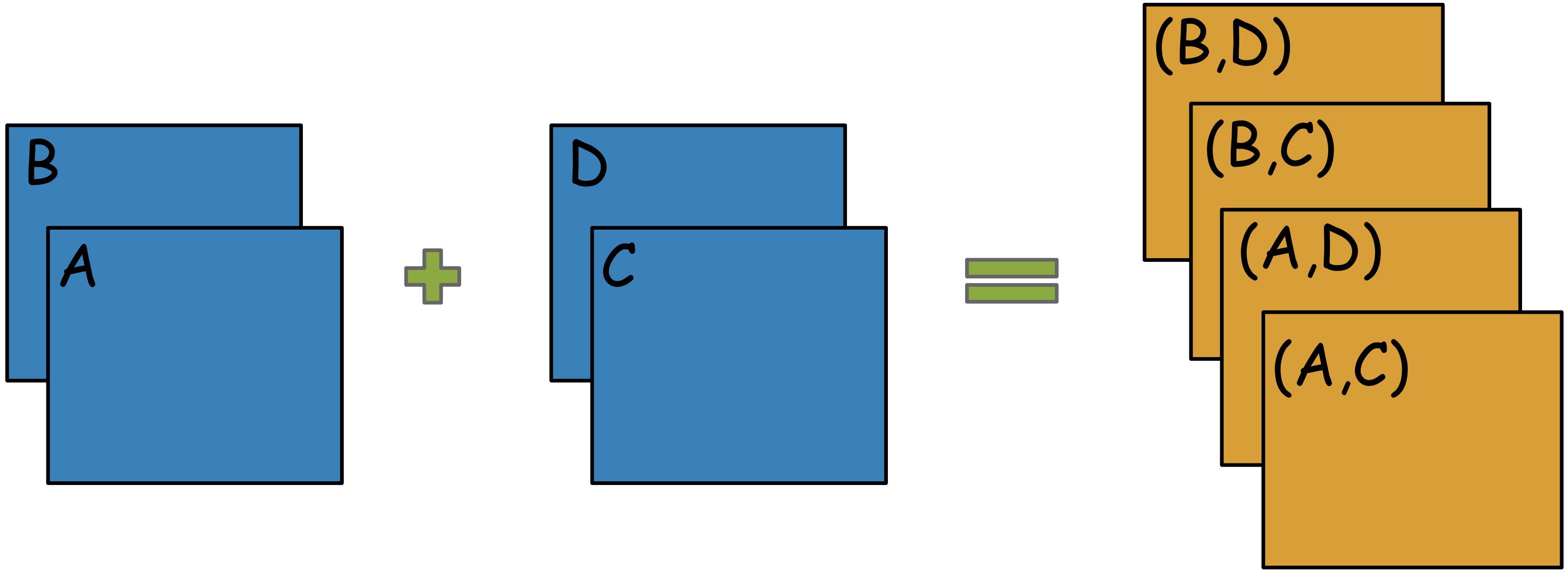
Panorays

coalesce

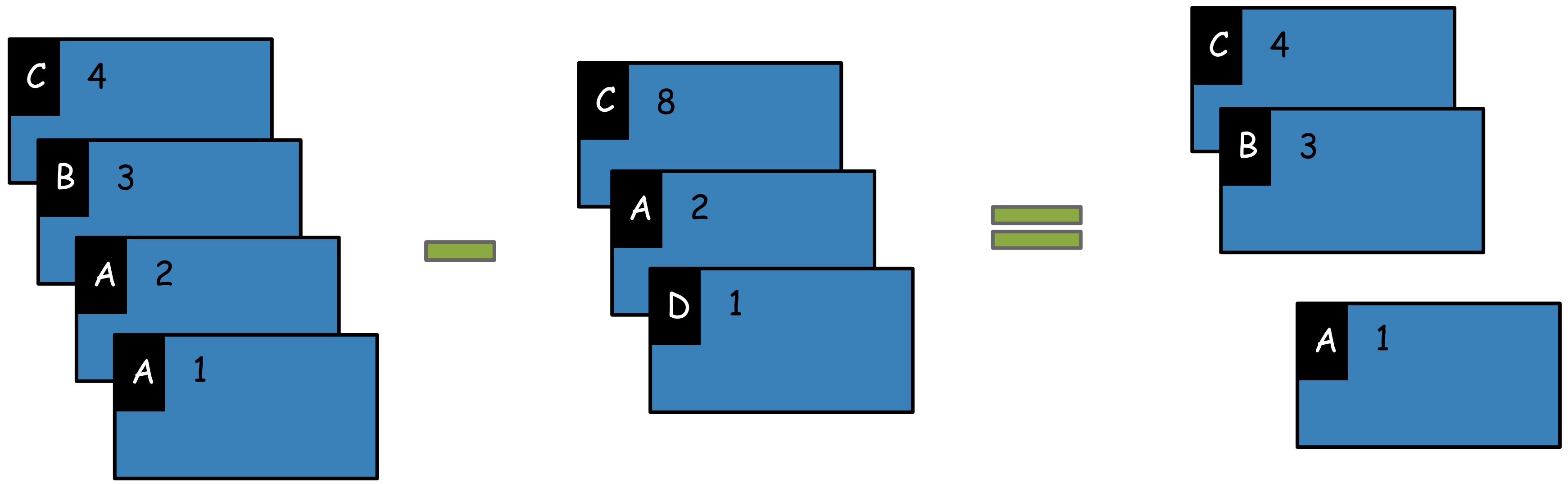
numPartitions = 1



cartesian

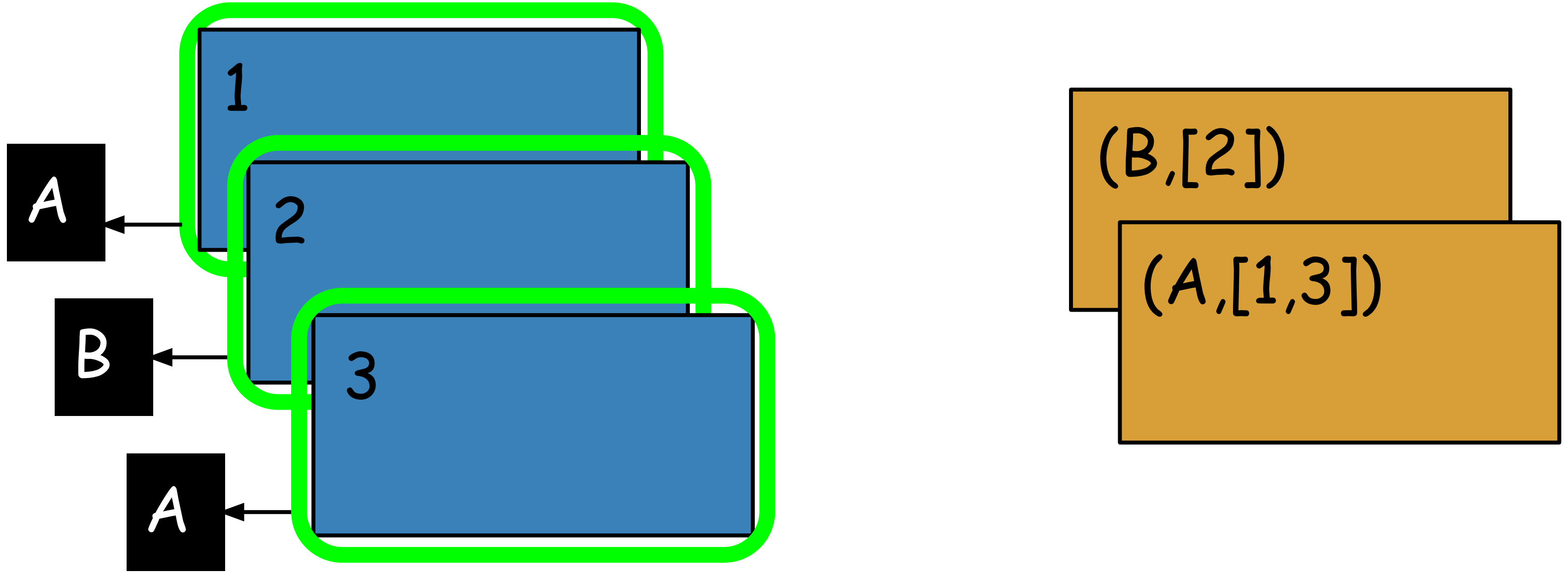


substruct (RDD1.substruct(RDD2))



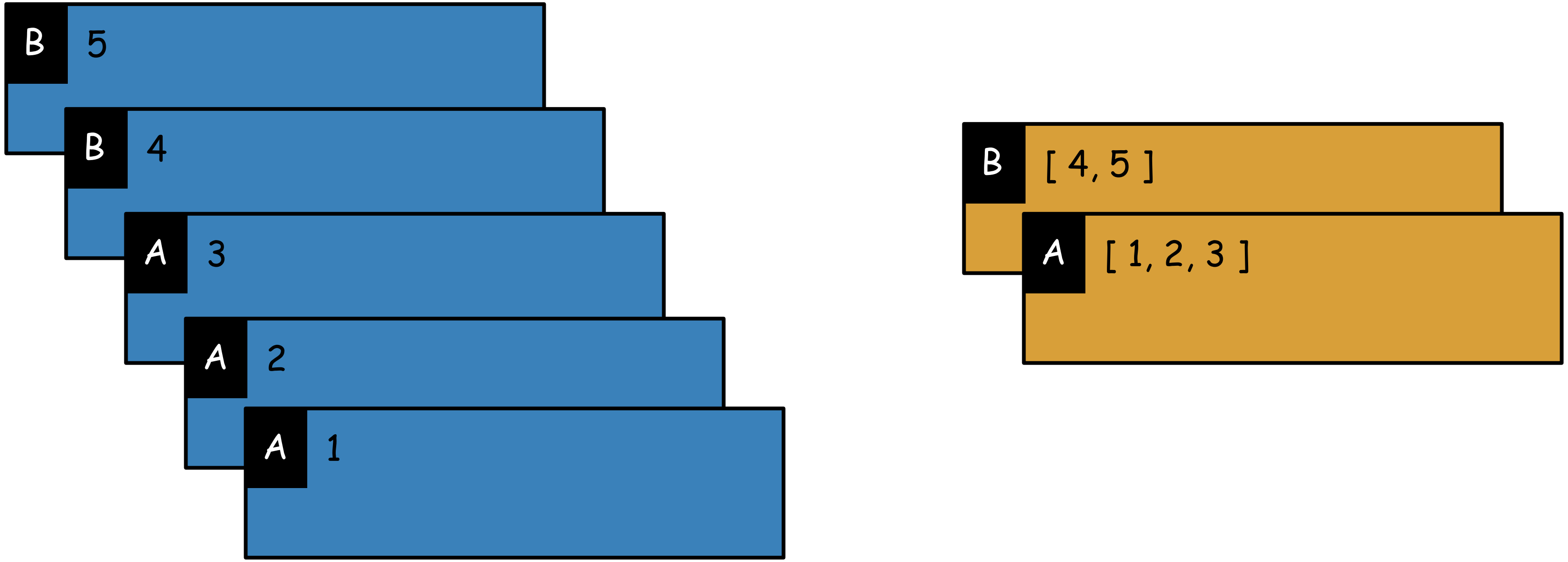
Panorays

groupBy

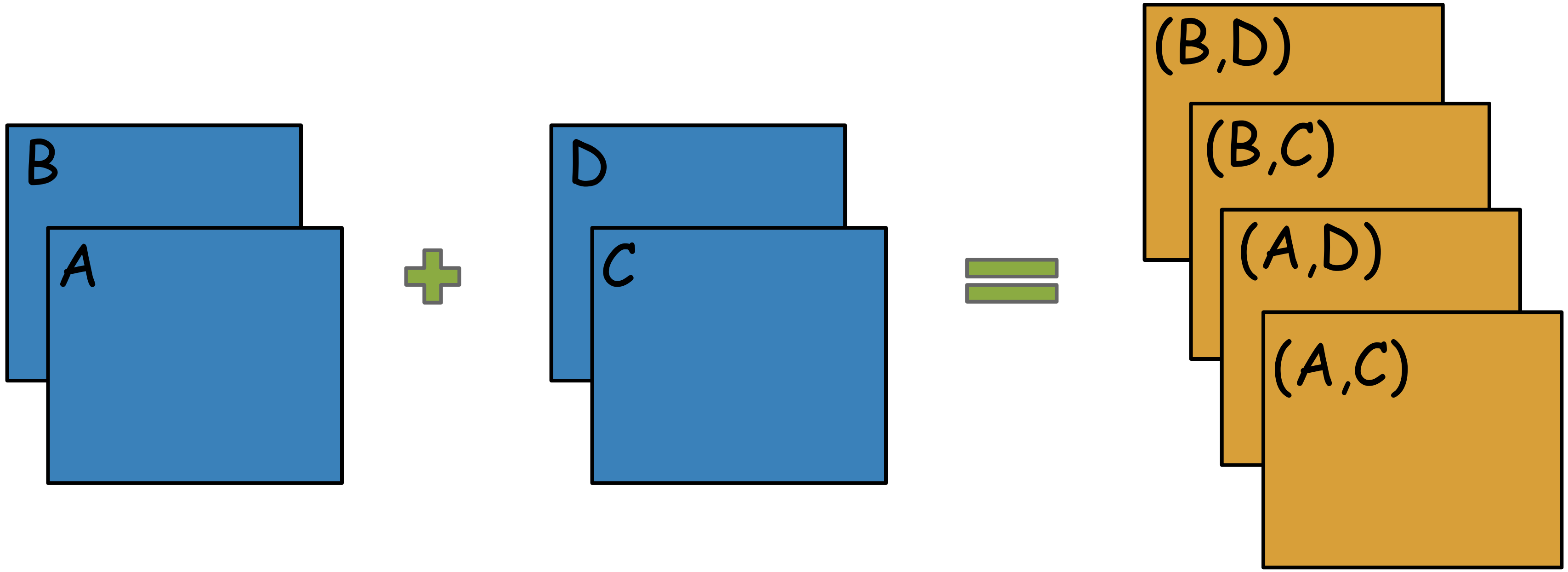


Panorays

groupByKey

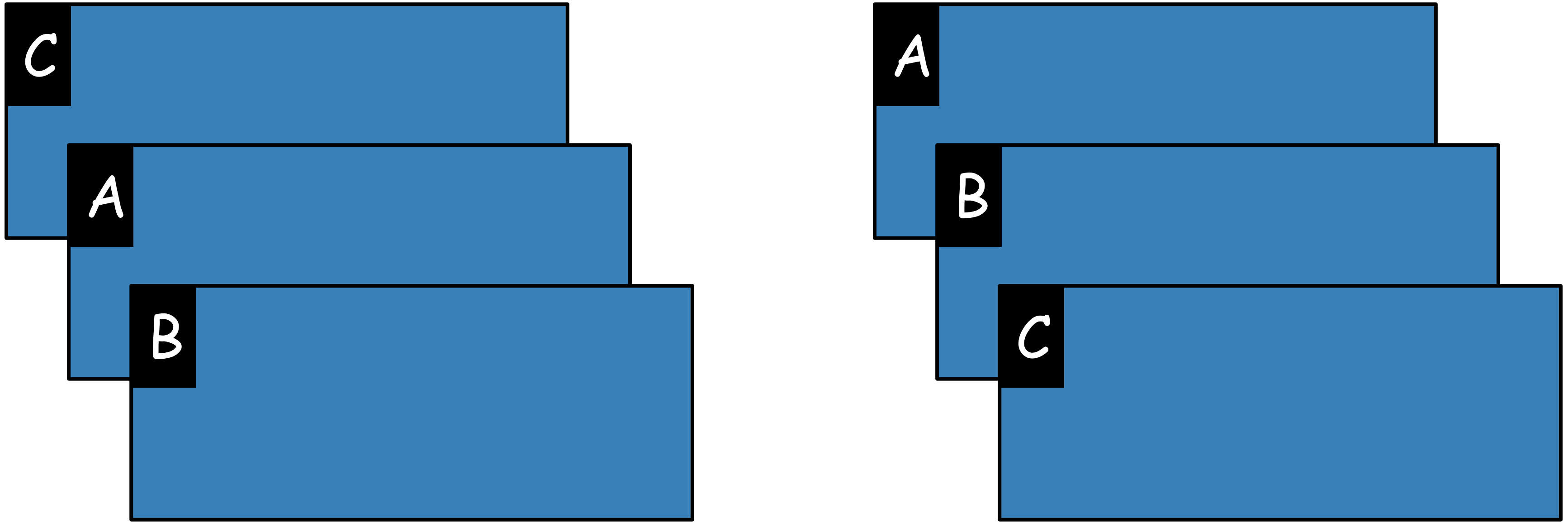


cartesian (RDD1.cartesian(RDD2))



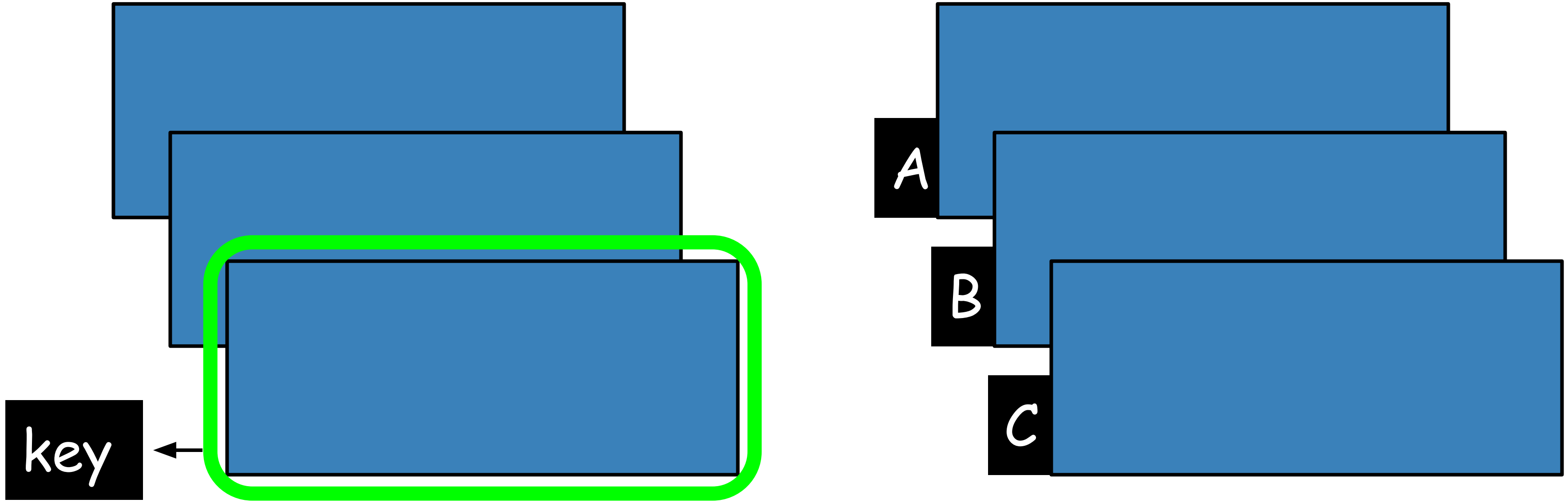
Panorays

sortByKey

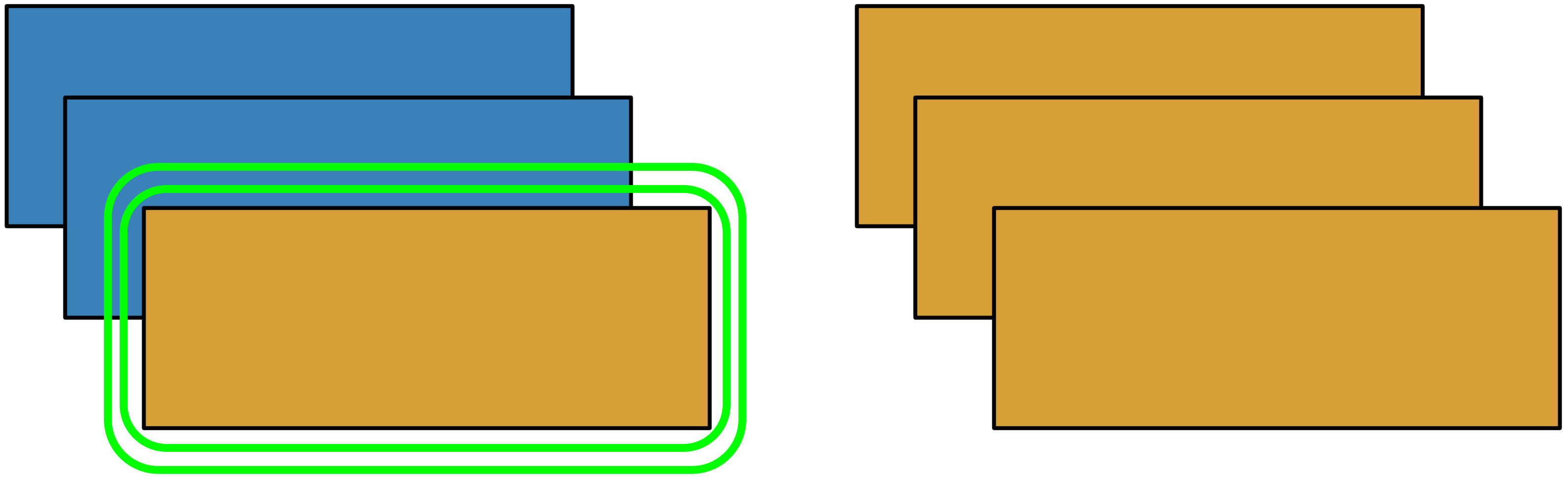


Panorays

sortBy



pipe

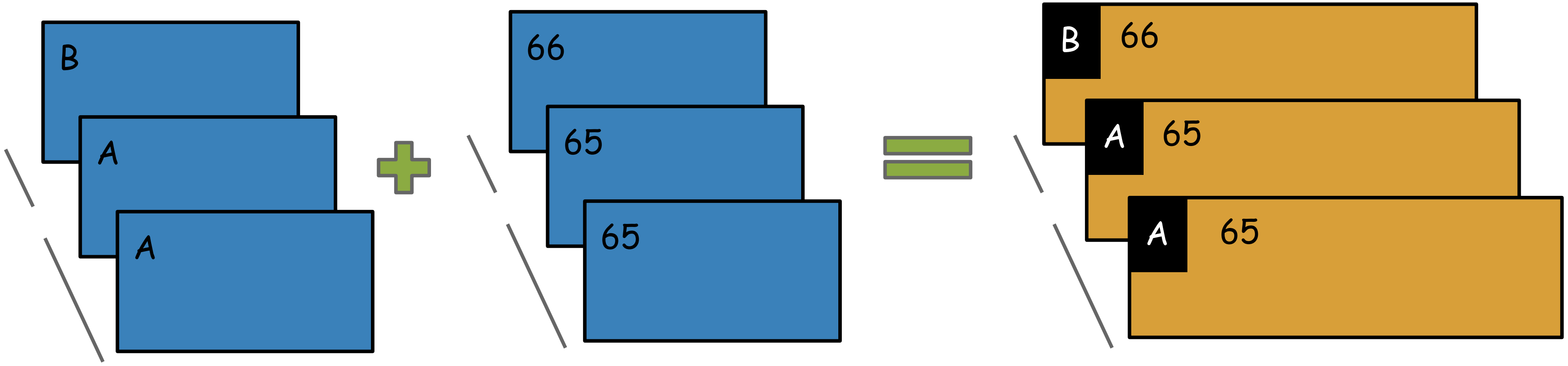


external command

- Pipe each partition of the RDD through a shell command, e.g. a Perl or bash script. RDD elements are written to the process's stdin and lines output to its stdout are returned as an RDD of strings.
- (I’ve never used it)

Panorays

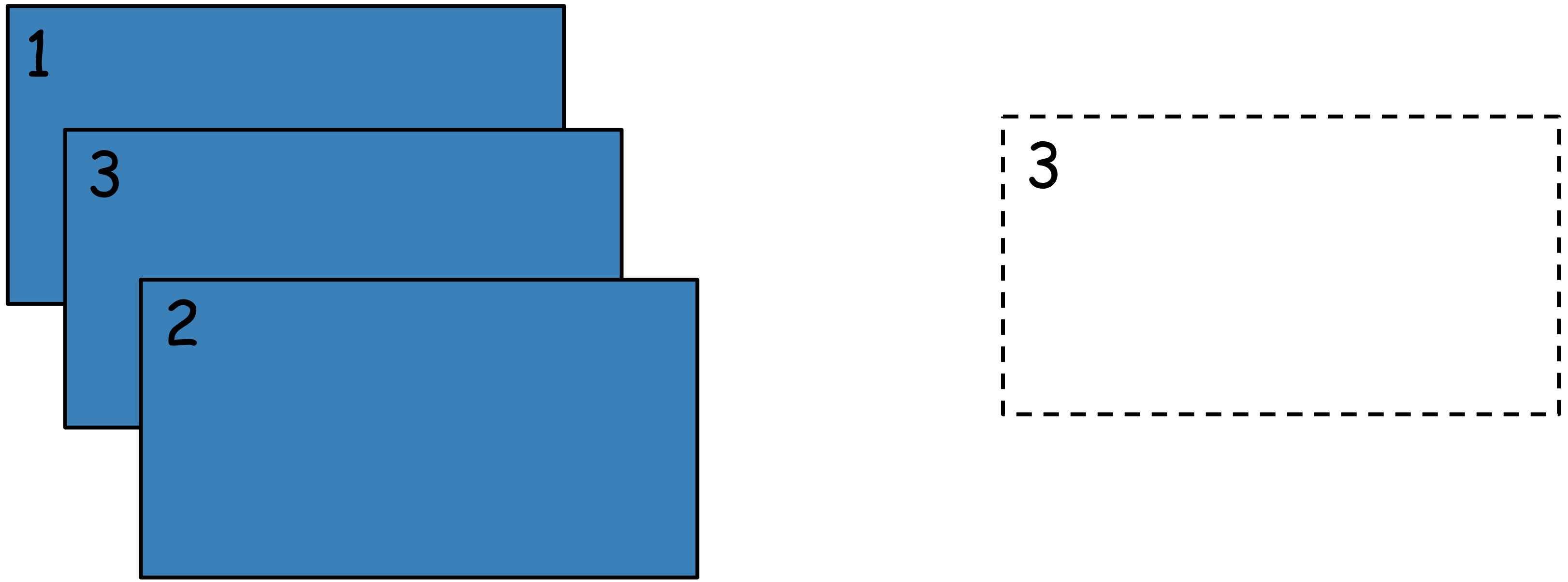
zip (RDD1.zip(RDD2))



Panorays

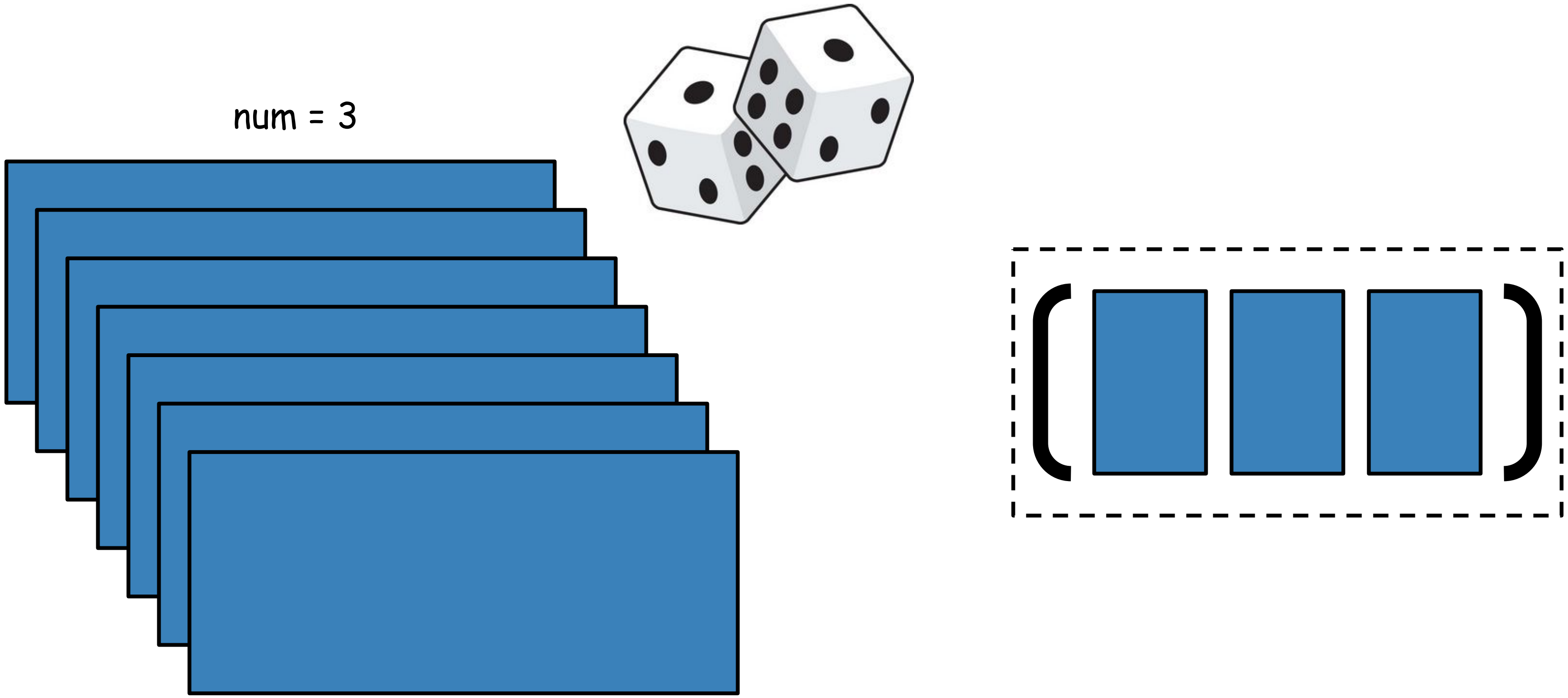
Actions

count



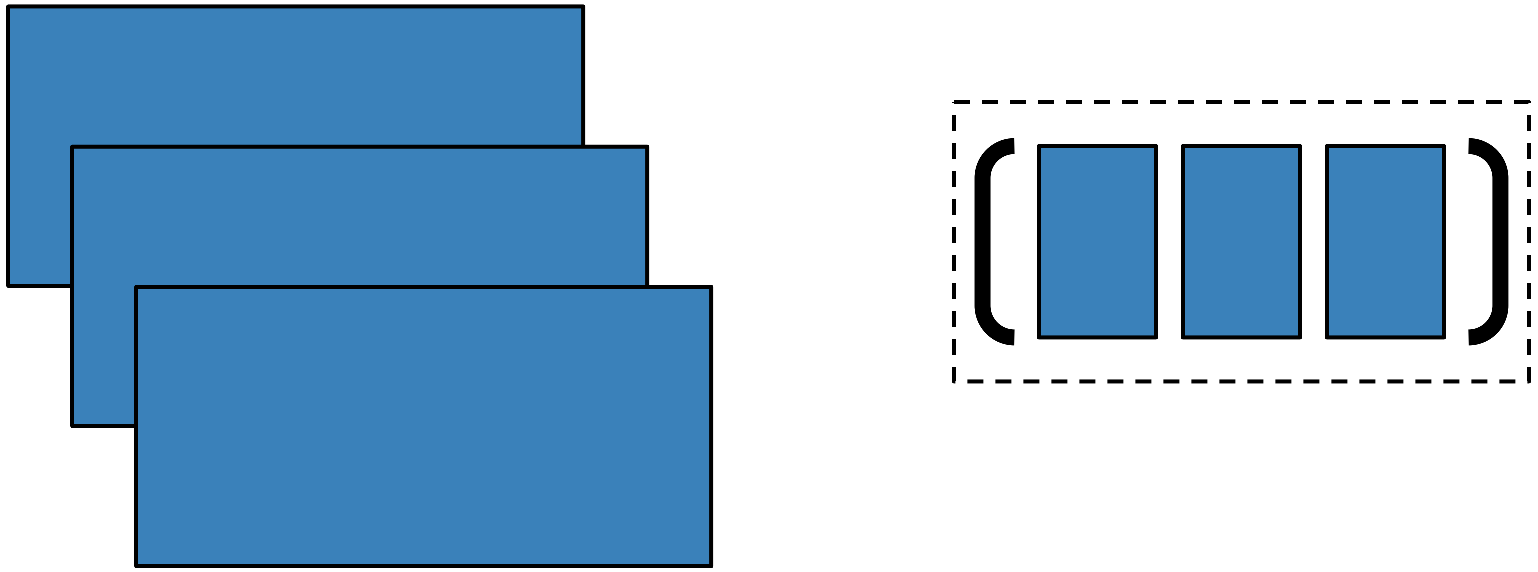
Panorays

takeSample



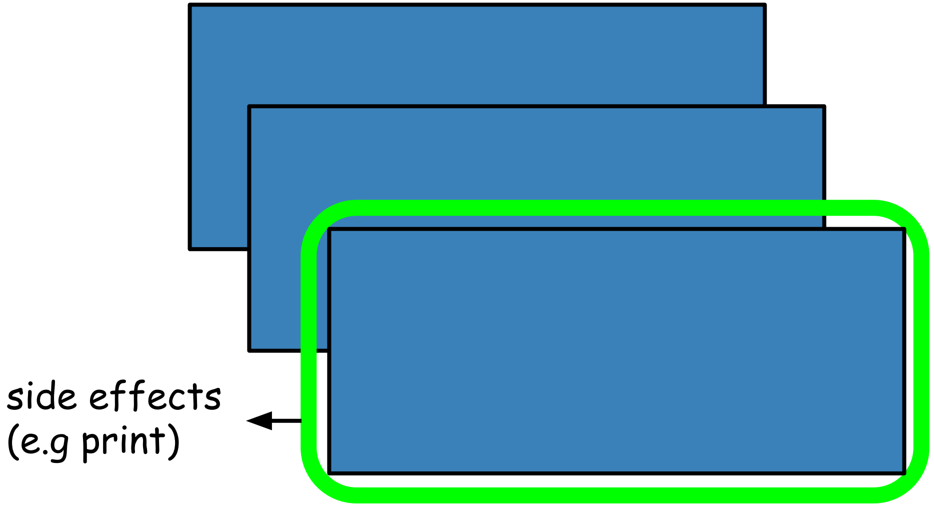
Panorays

Collect



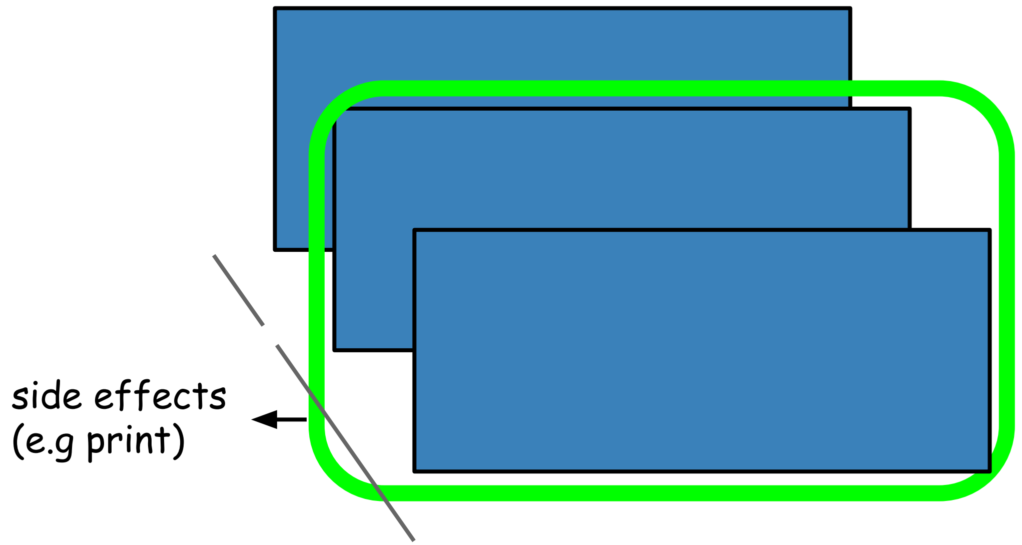
Panorays

foreach



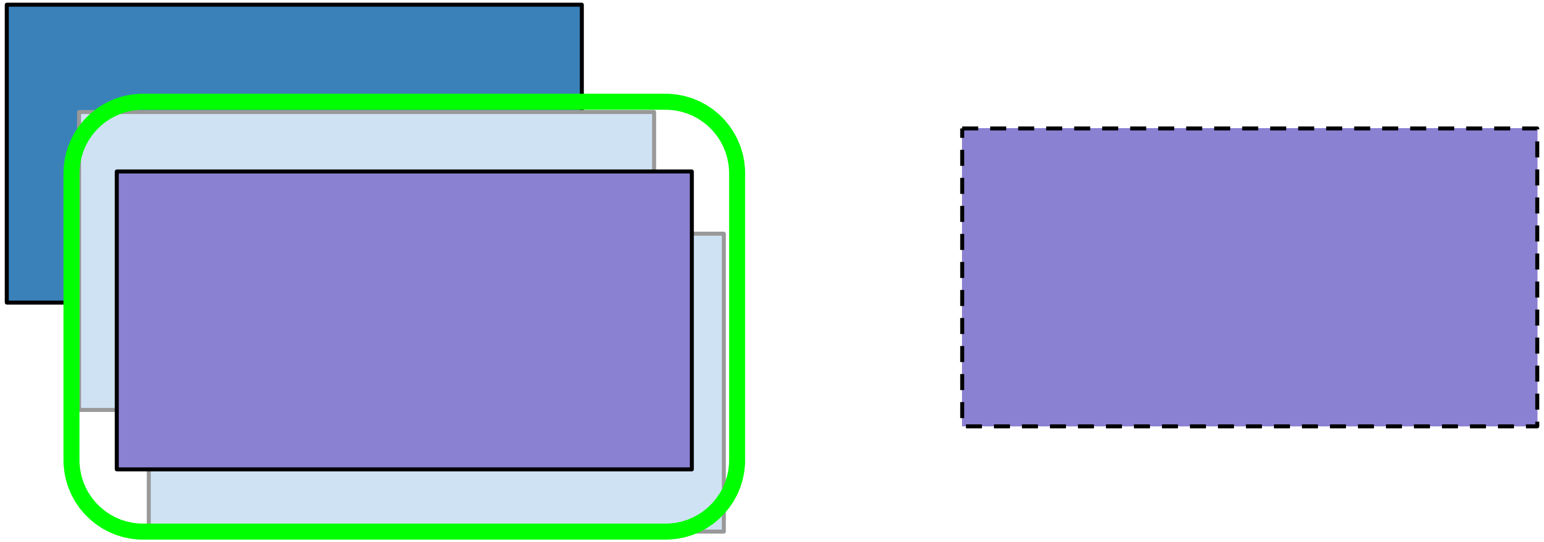
*no return value,
original RDD unchanged

foreachPartition



*no return value,
original RDD unchanged

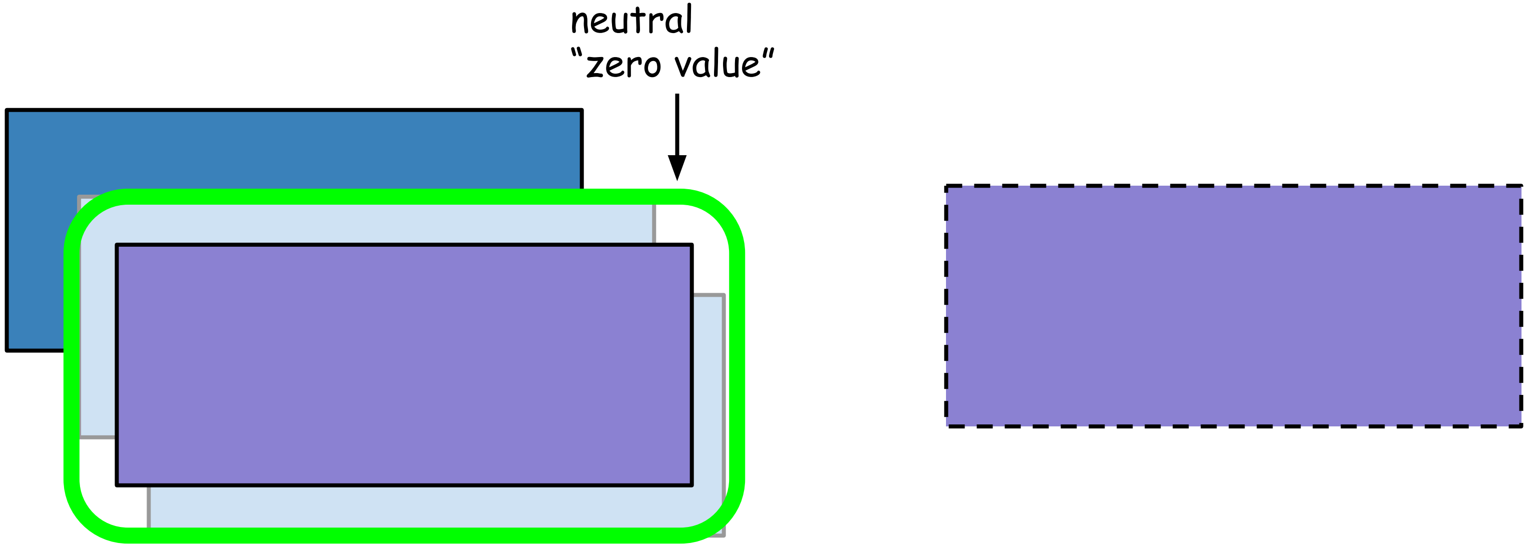
reduce



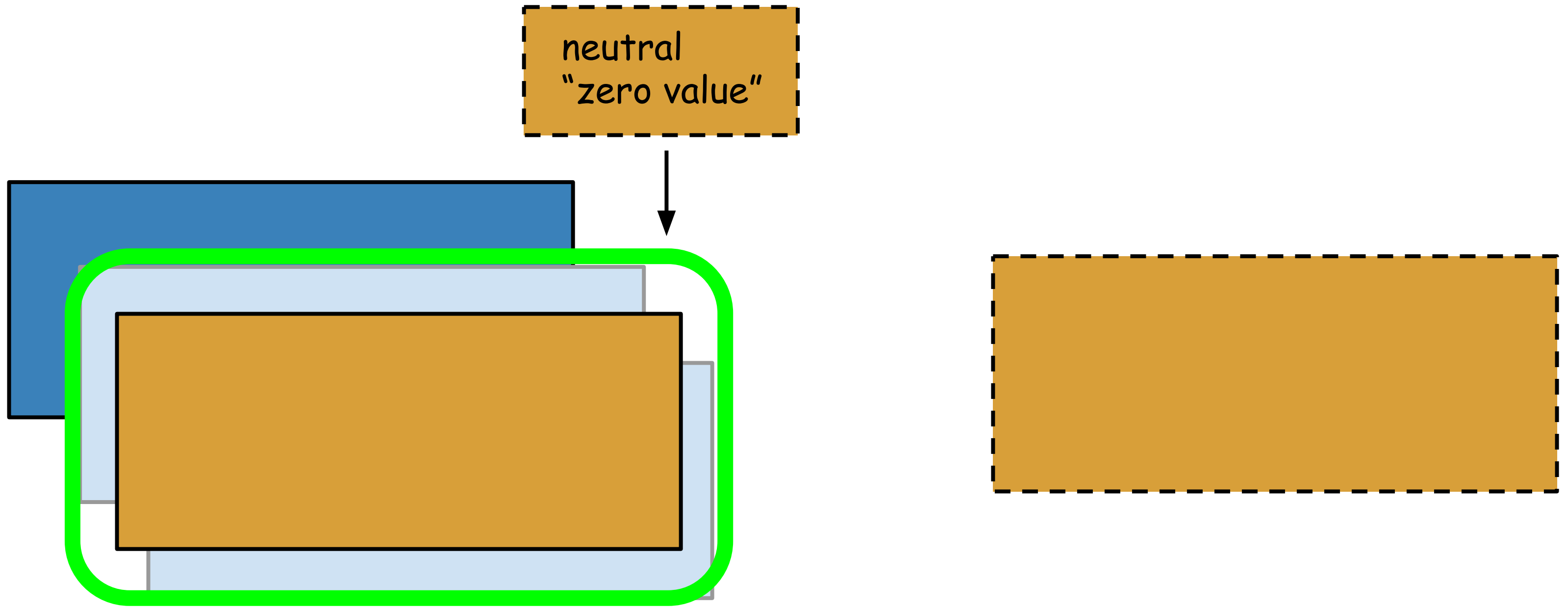
Panorays

fold

- If you have an empty RDD, it’s a replacement to reduce.

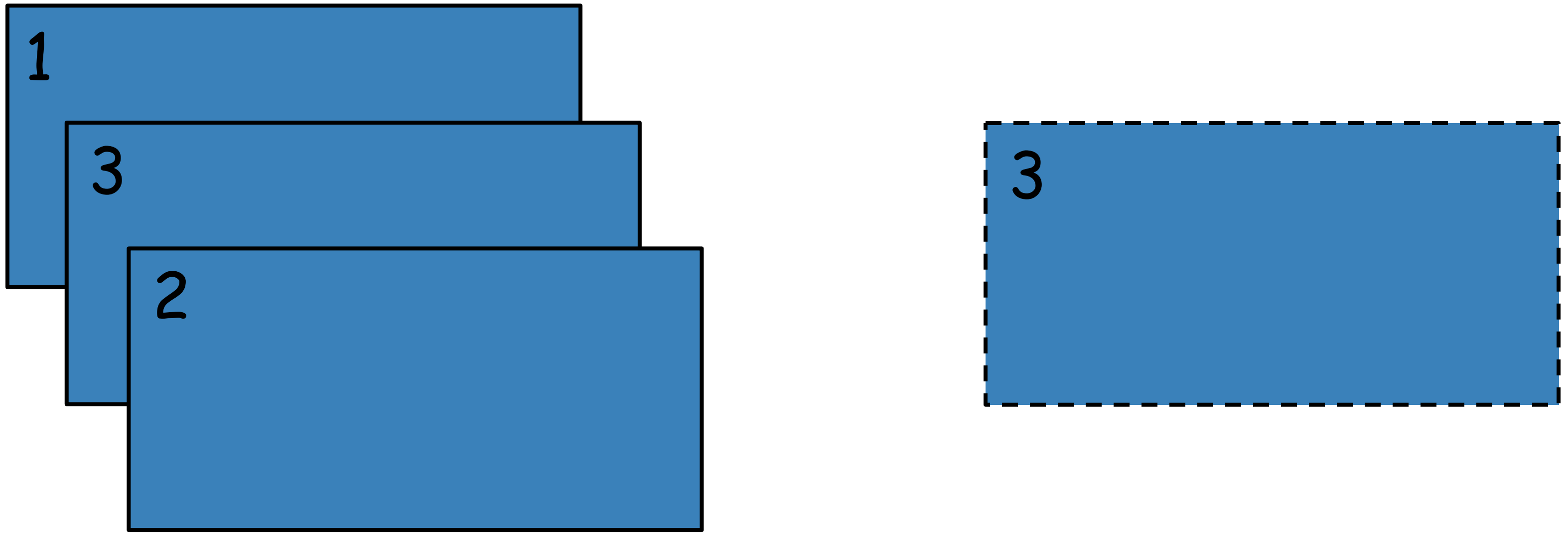


aggregate()



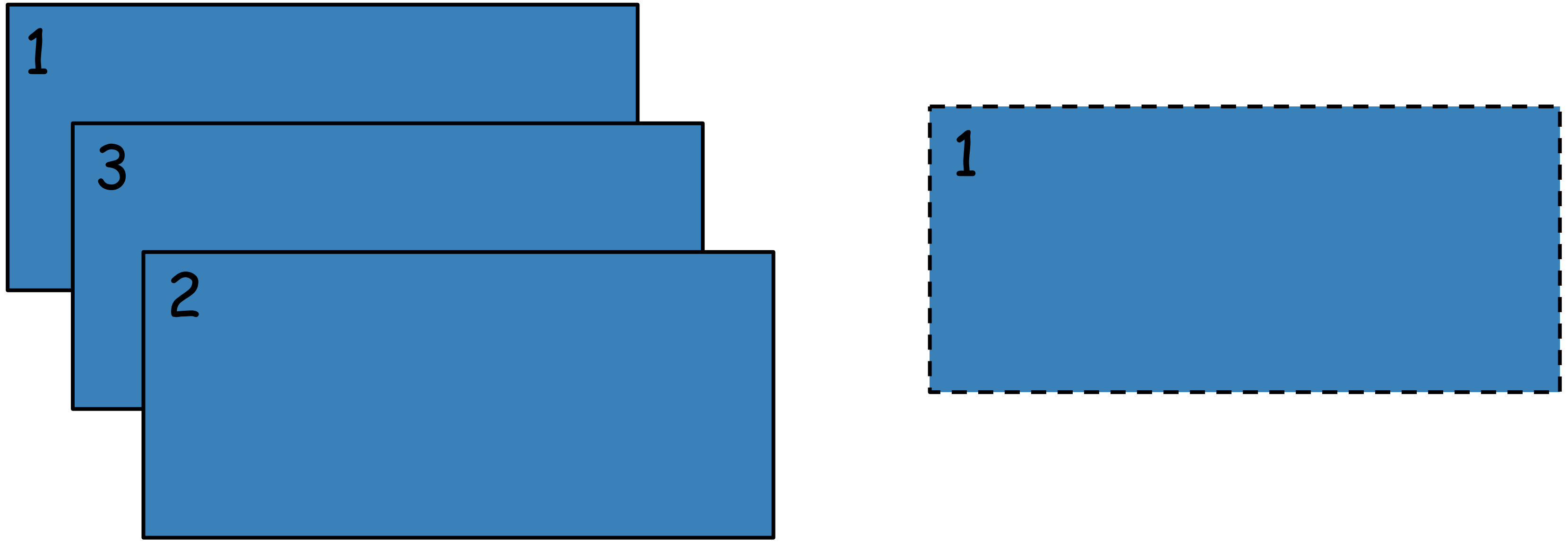
Panorays

max



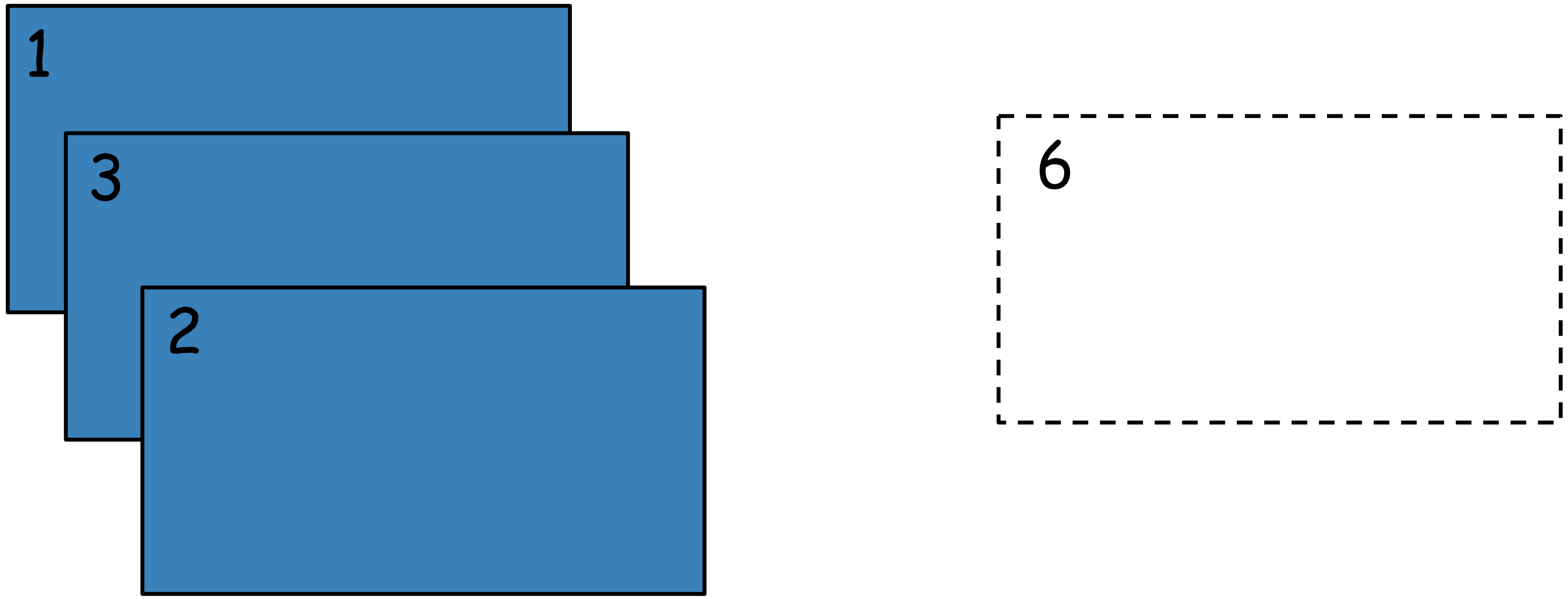
Panorays

min

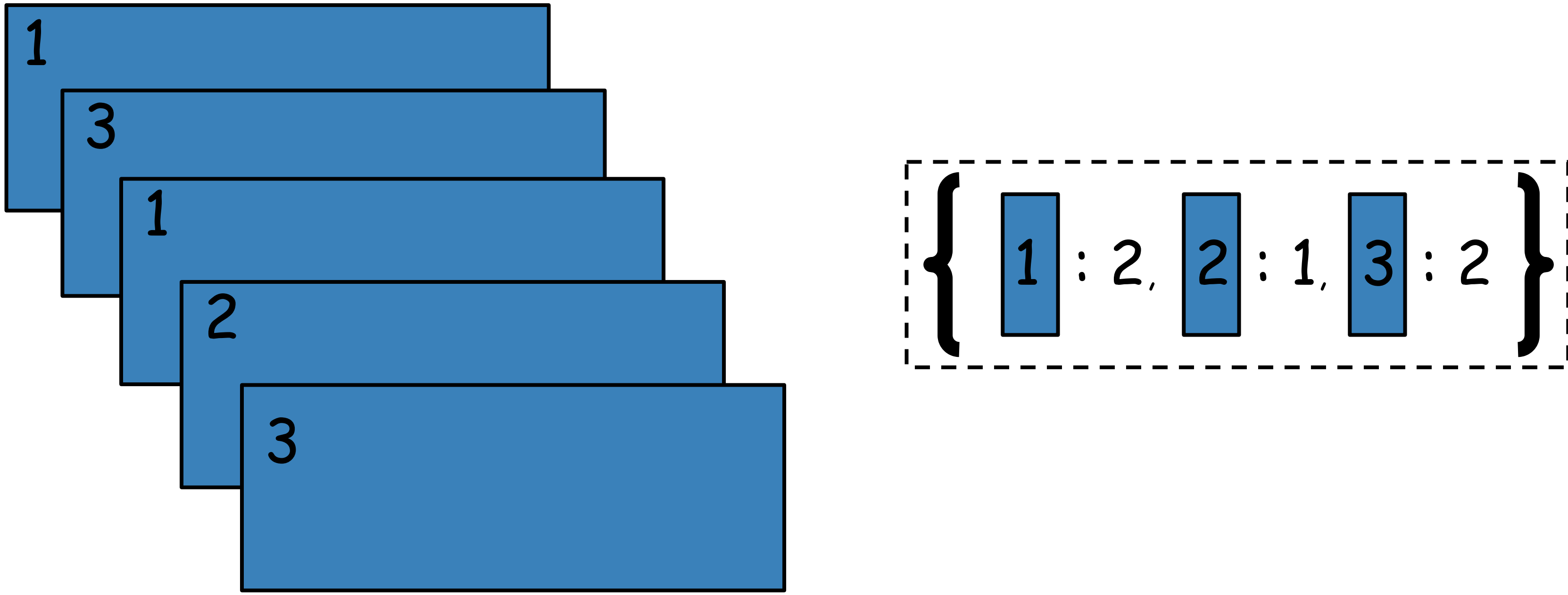


Panorays

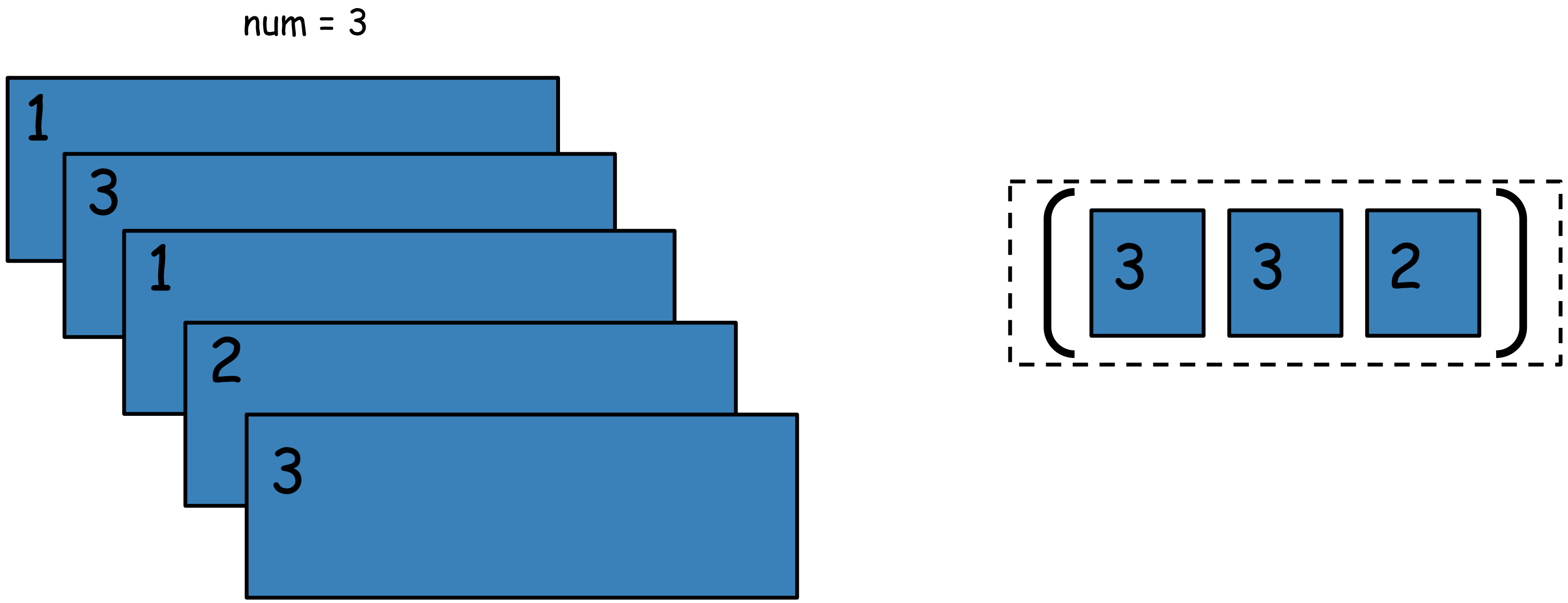
sum



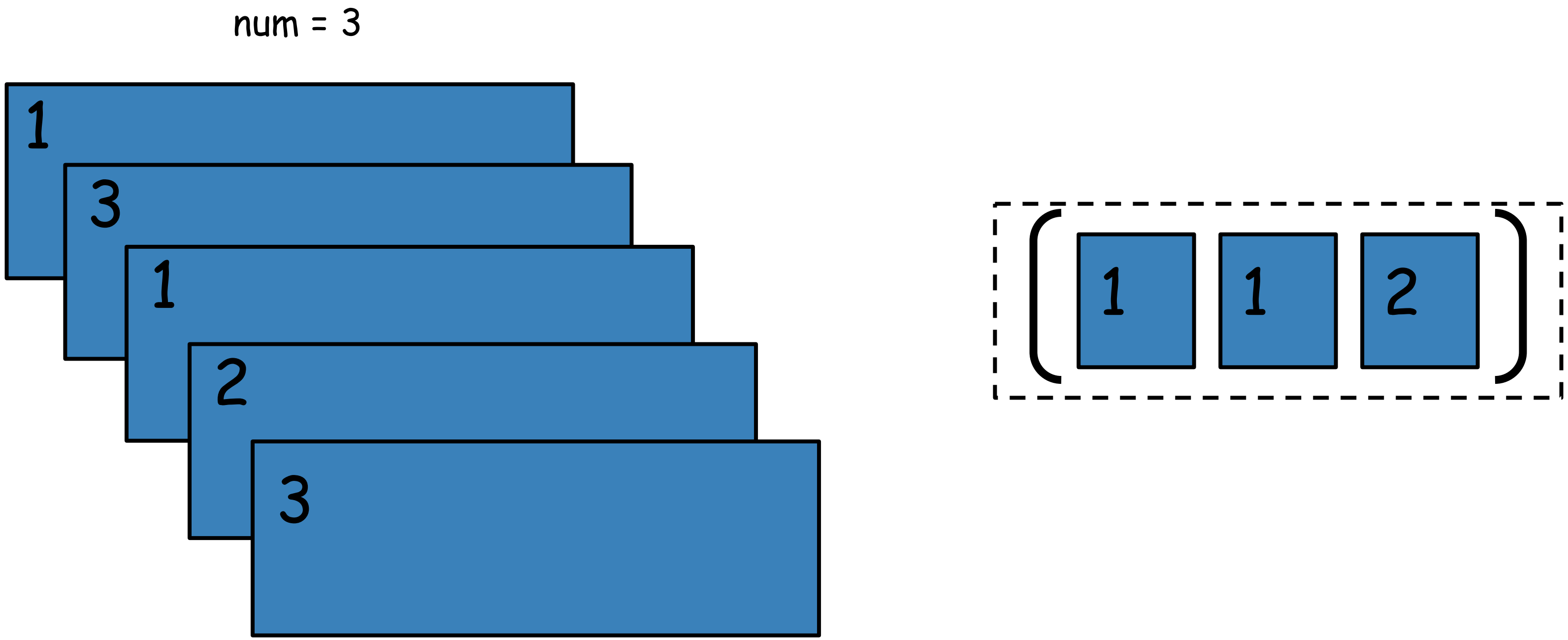
countByValue()



top

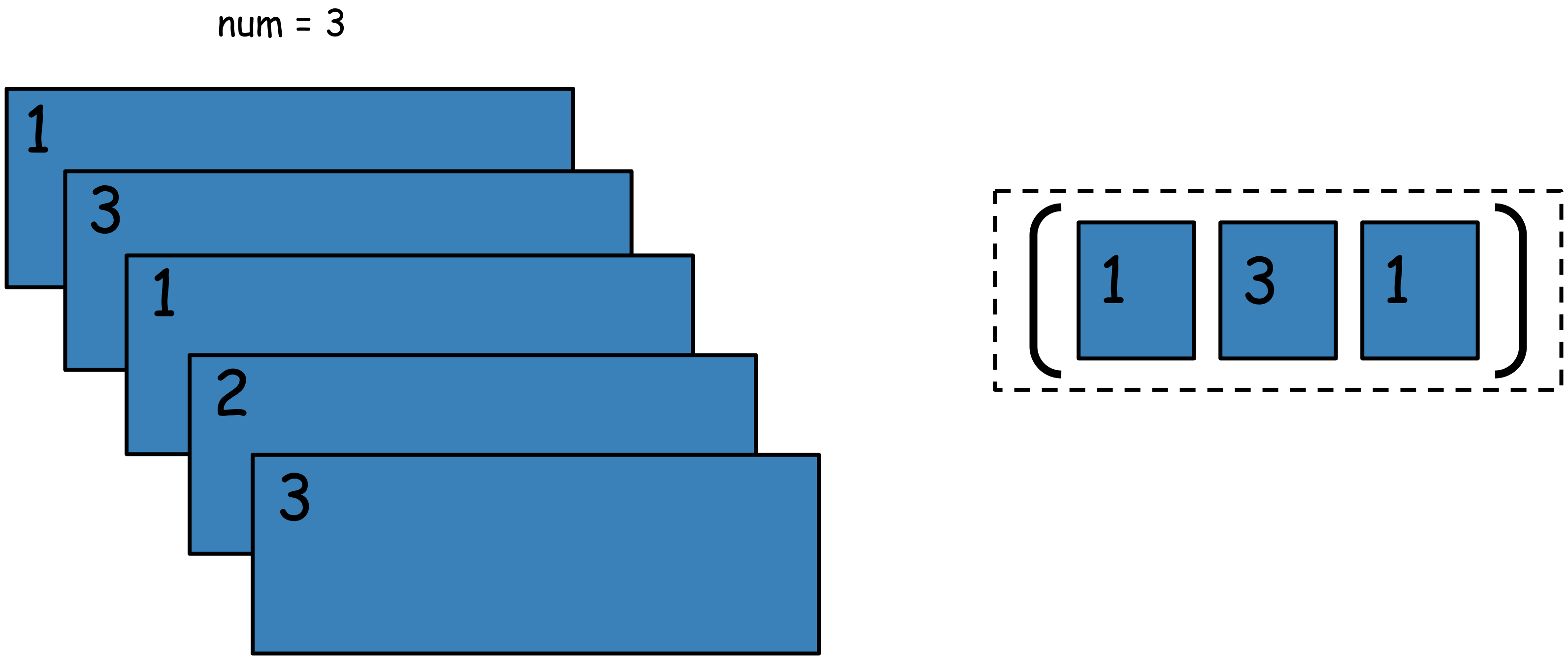


topOrdered

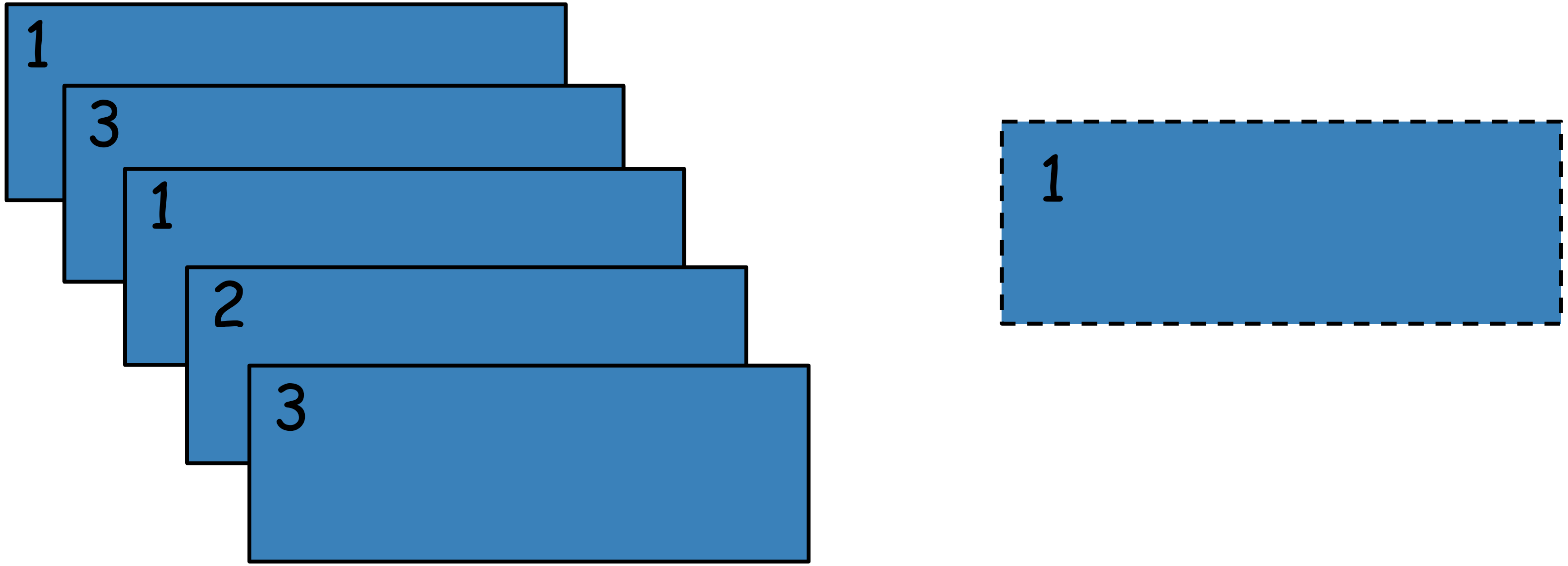


Panorays

take



first



More Actions

- takeSample()
- mean()
- variance()
- tsdev()
- histogram()
- sampleStdev()
- sampleVariance()

Again, to tell you the truth, I’ve never used these.

Persistence

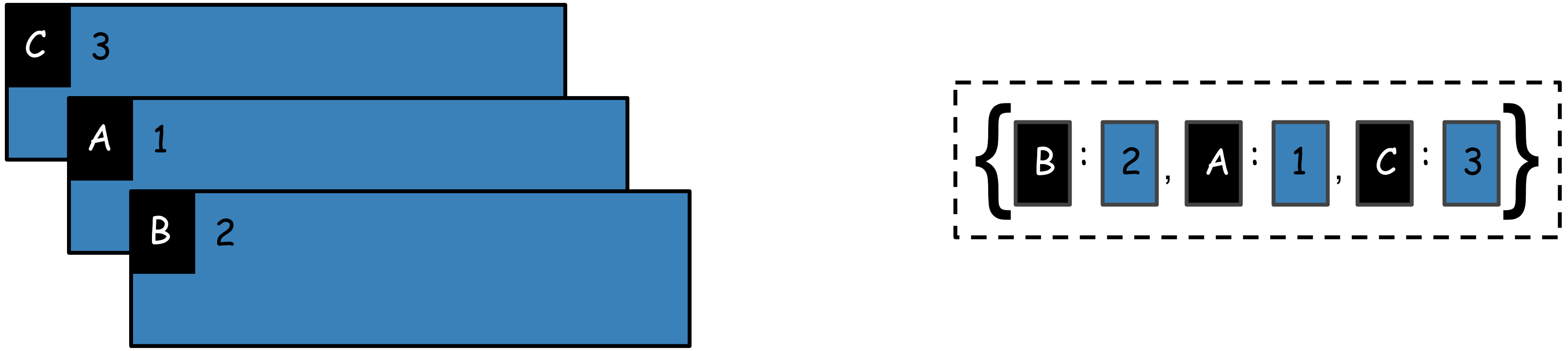
Actions

- saveAsObjectFile(path)
- saveAsTextFile(path)
- ExternalConnector
- foreach()
    - forEachPartition()

Key Value Methods

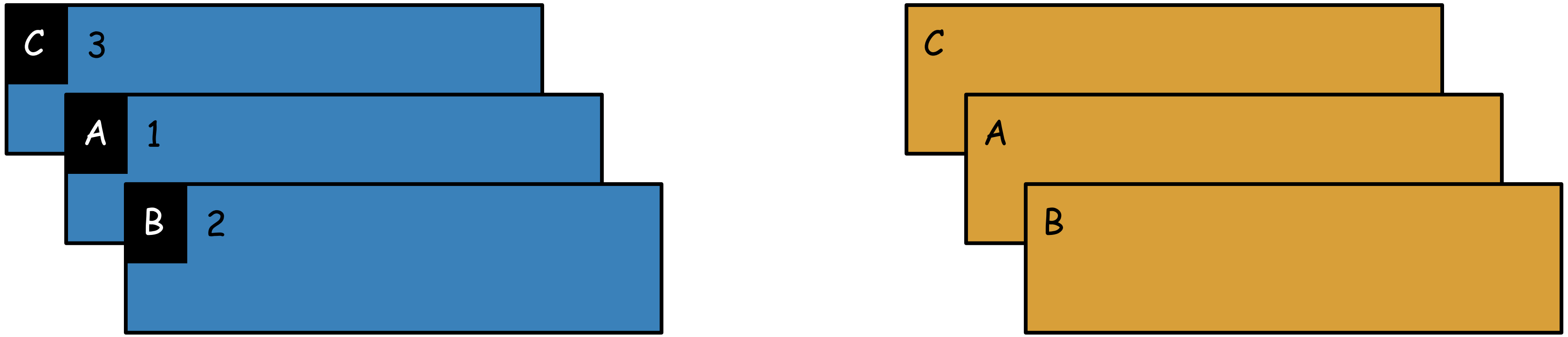
Panorays

collectAsMap

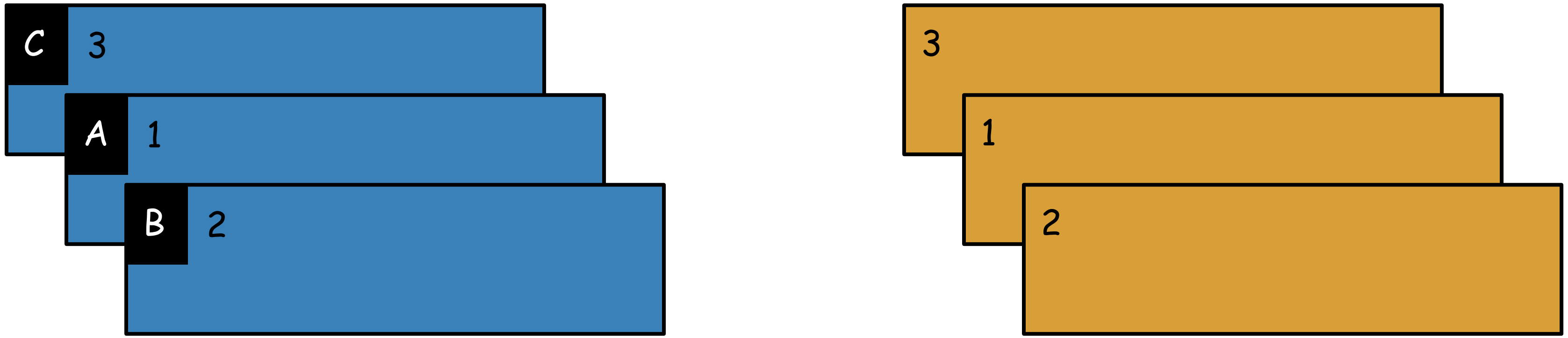


Panorays

keys

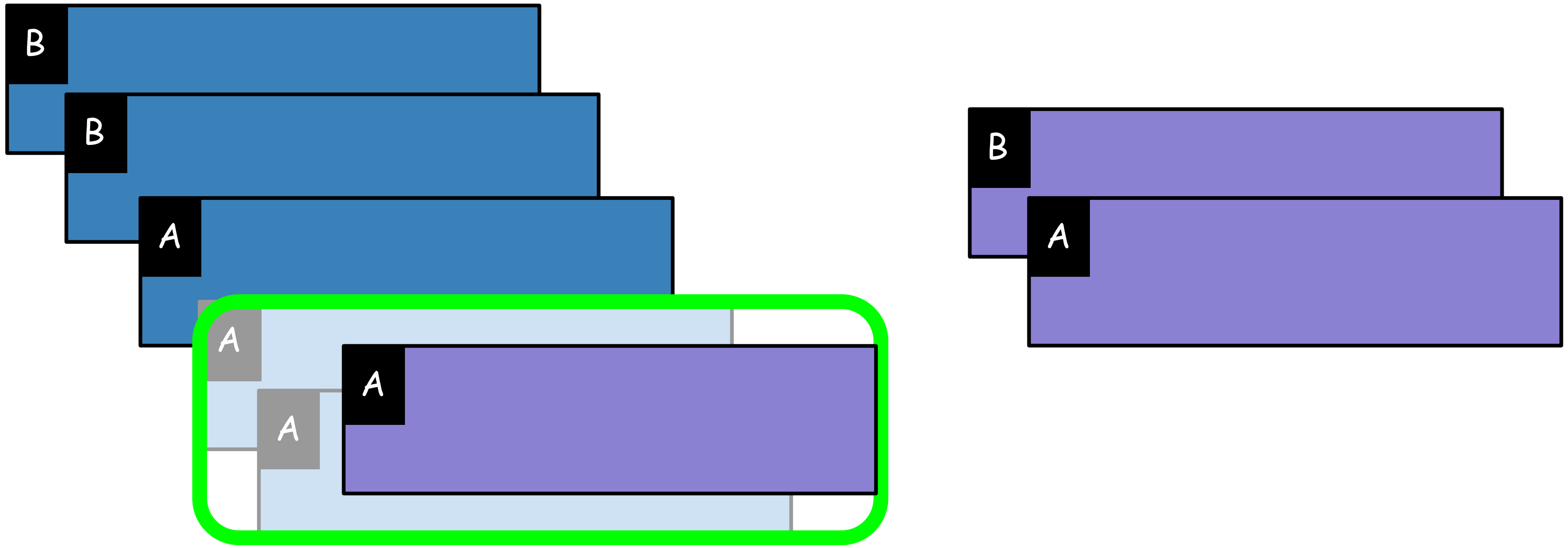


values



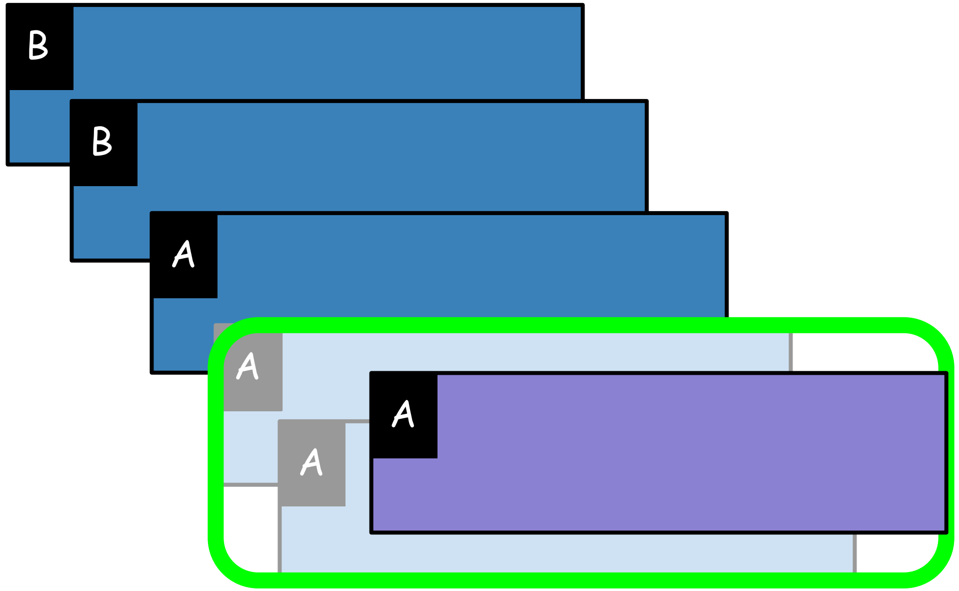
reduceByKey

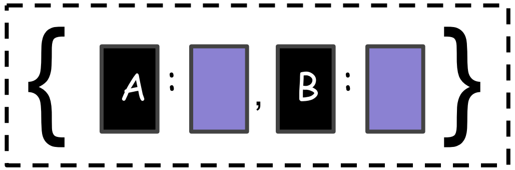
- Occurs locally



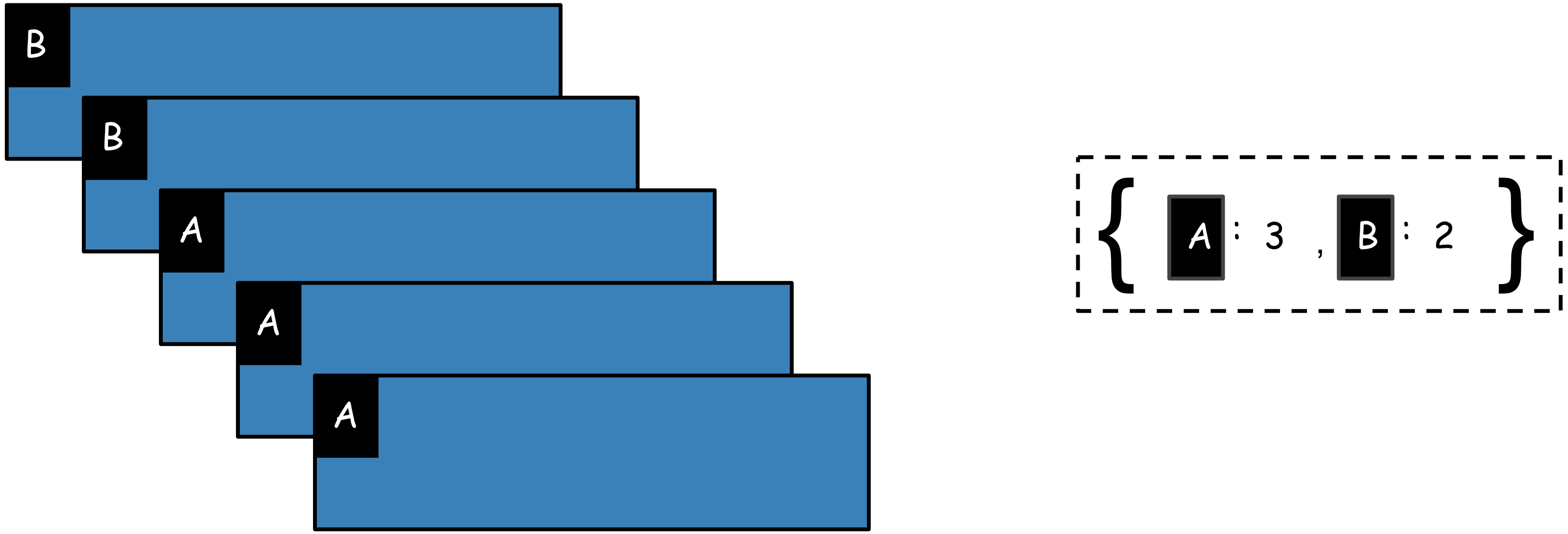
Panorays

reduceByKeyLocally





countByKey

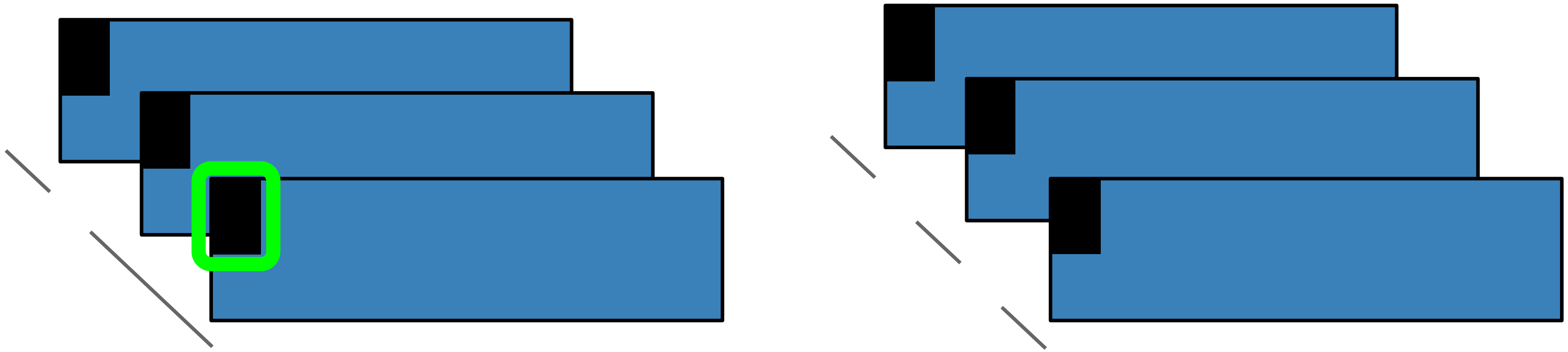


Panorays

partitionBy

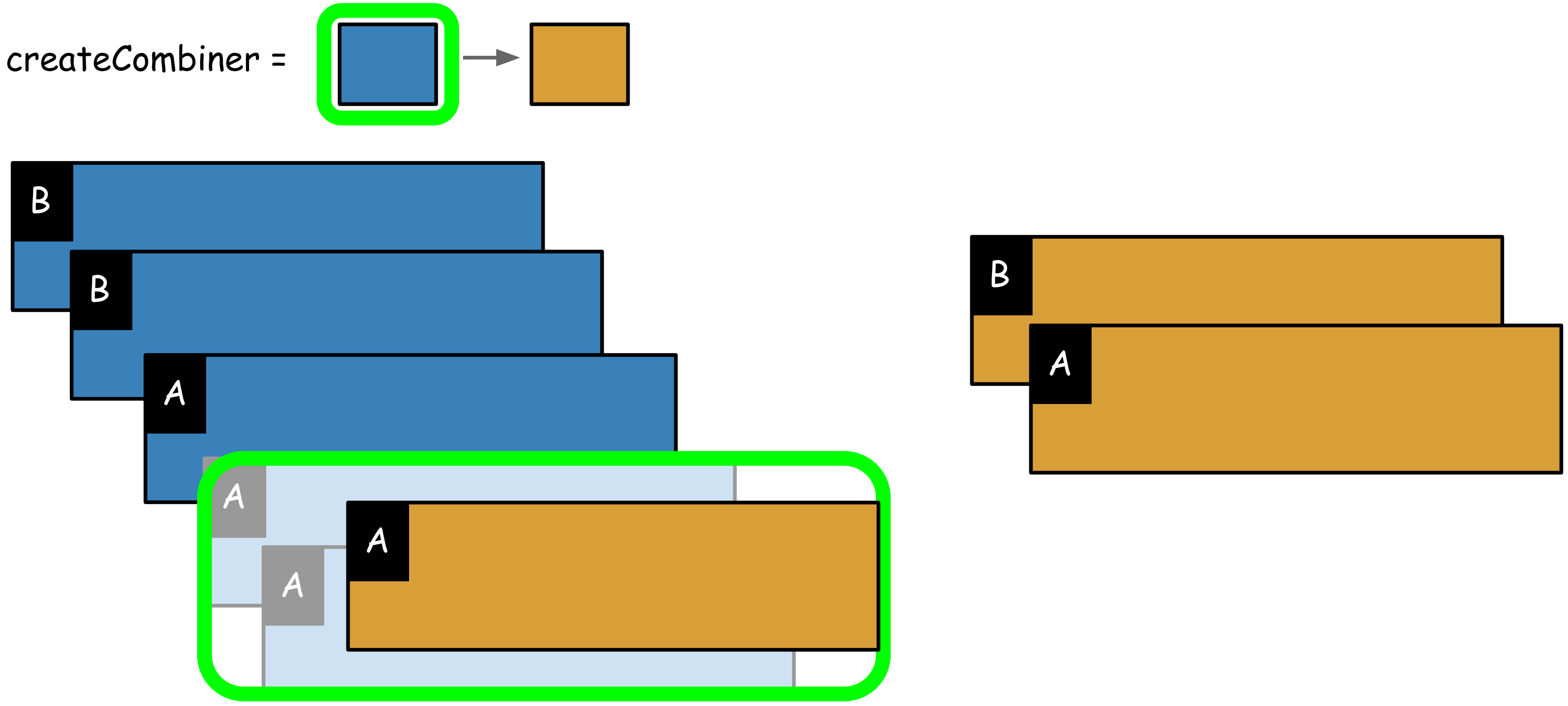
new partition index = key % numPartitions

numPartitions = 3

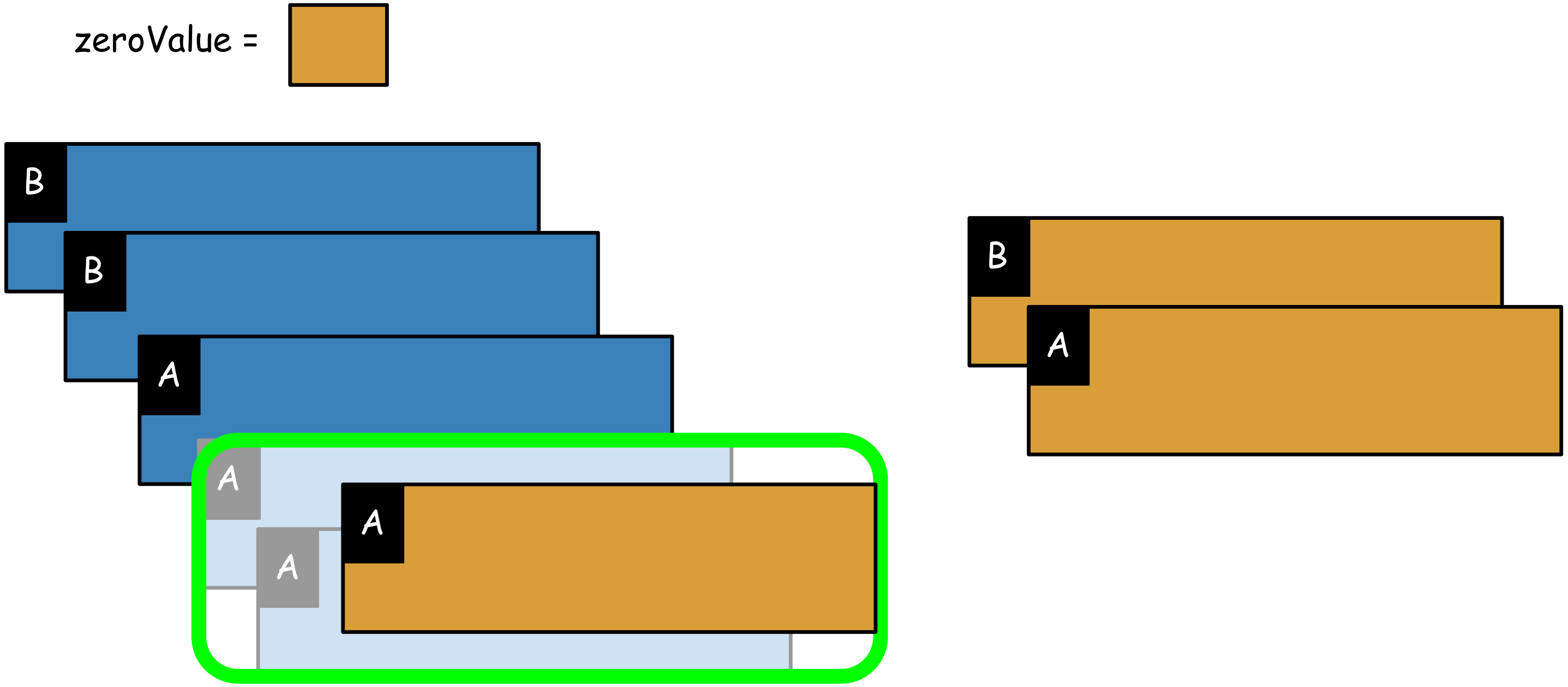


Panorays

combineBy

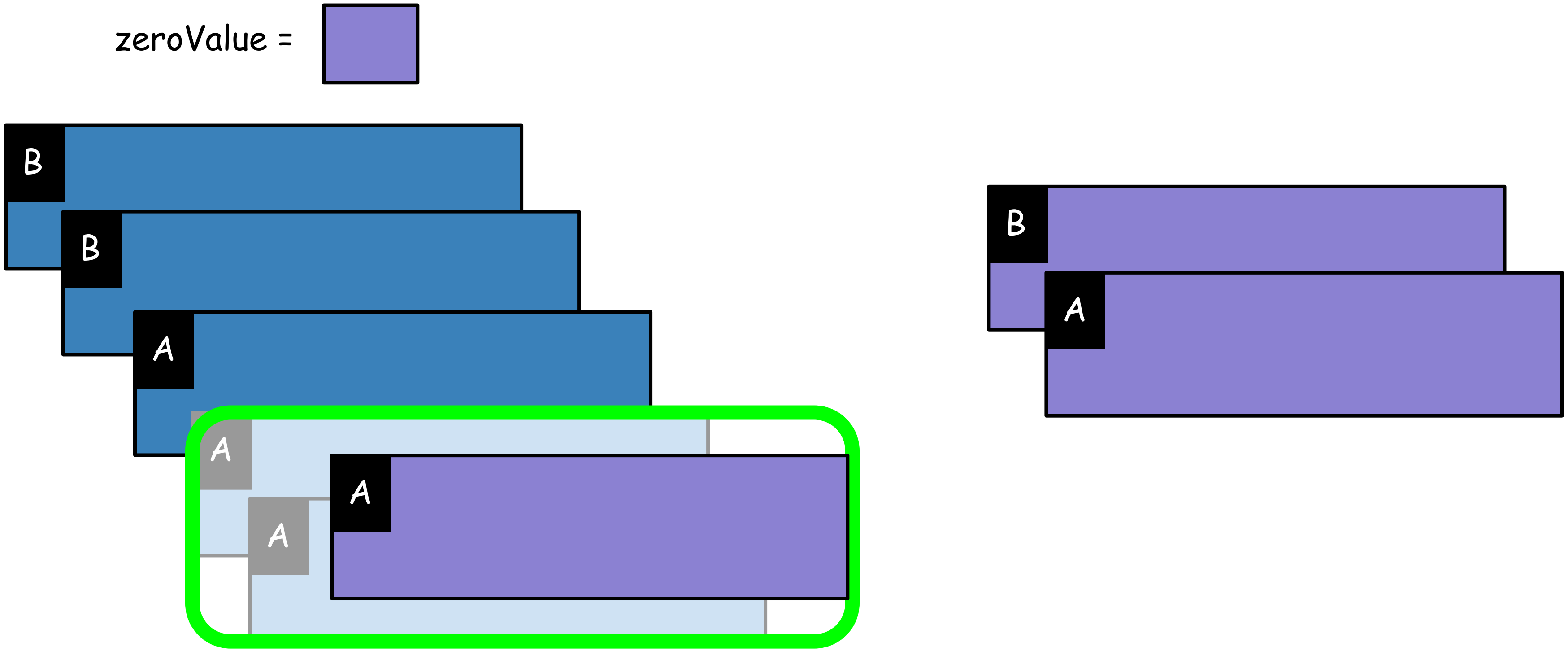


aggregateByKey



Panorays

foldByKey



Panorays

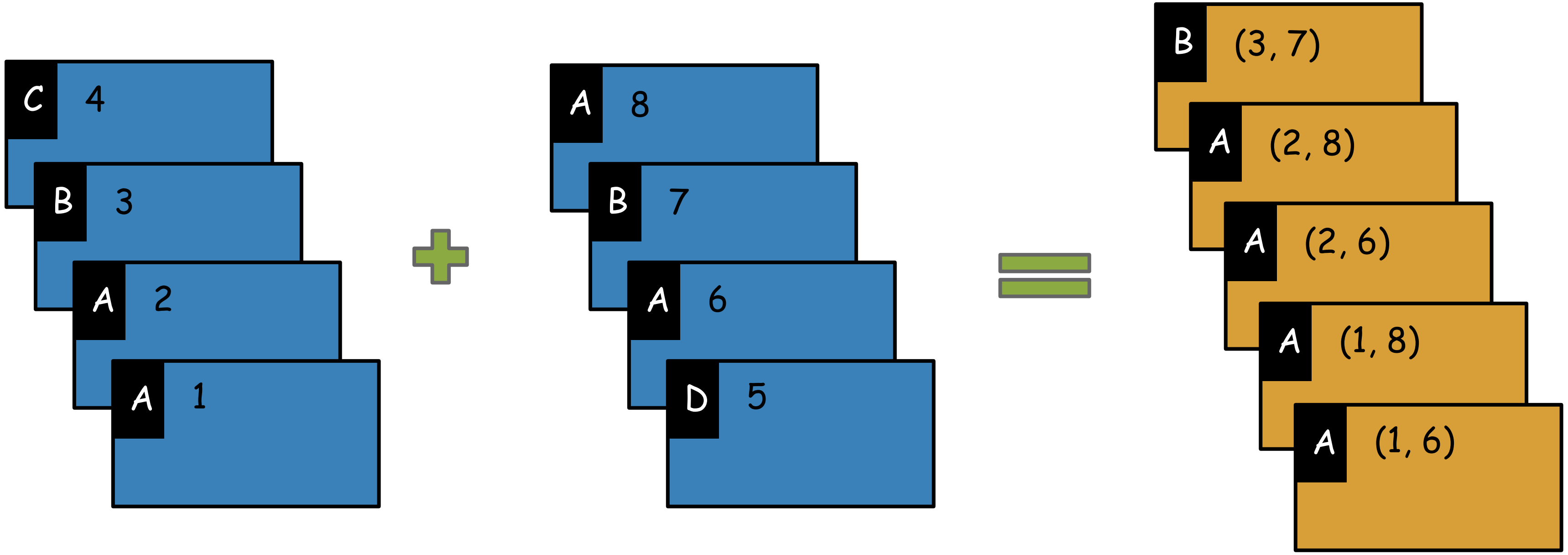
Regular Methods

- collectAsMap
- mapValues
- reduceByKey
  - Occurs locally
- foldByKey
- aggregateByKey
- combineByKey
- groupByKey
- countByKey
- sampleByKey
- substractByKey
- sortByKey

SQL-like Pairings

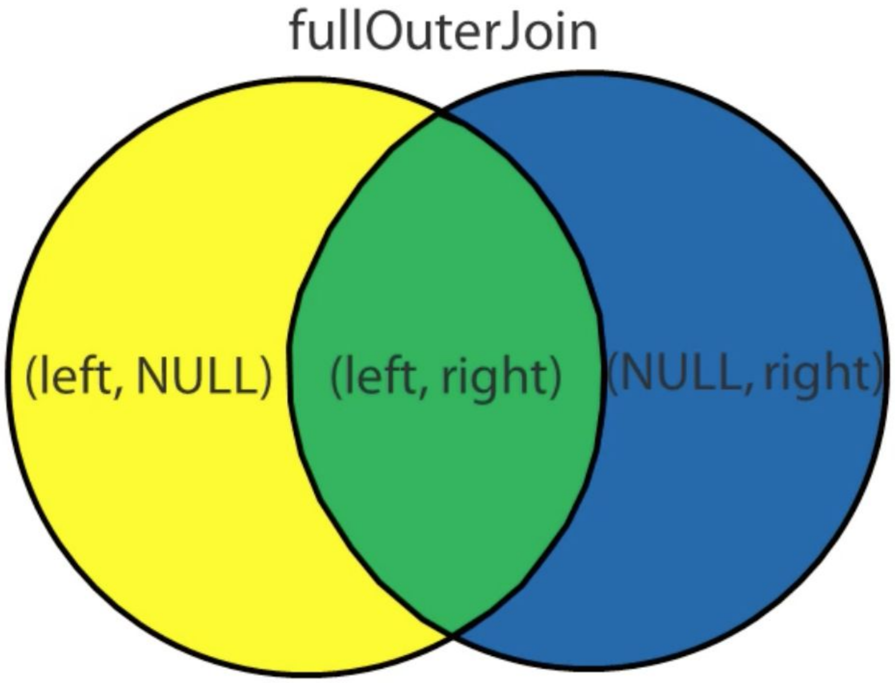
Panorays

join



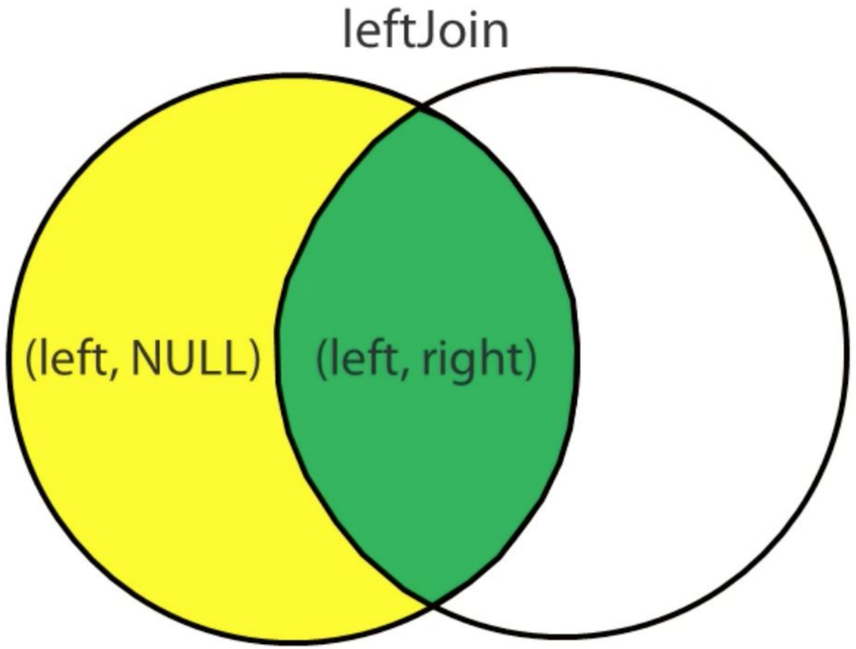
Panorays

fullOuterJoin

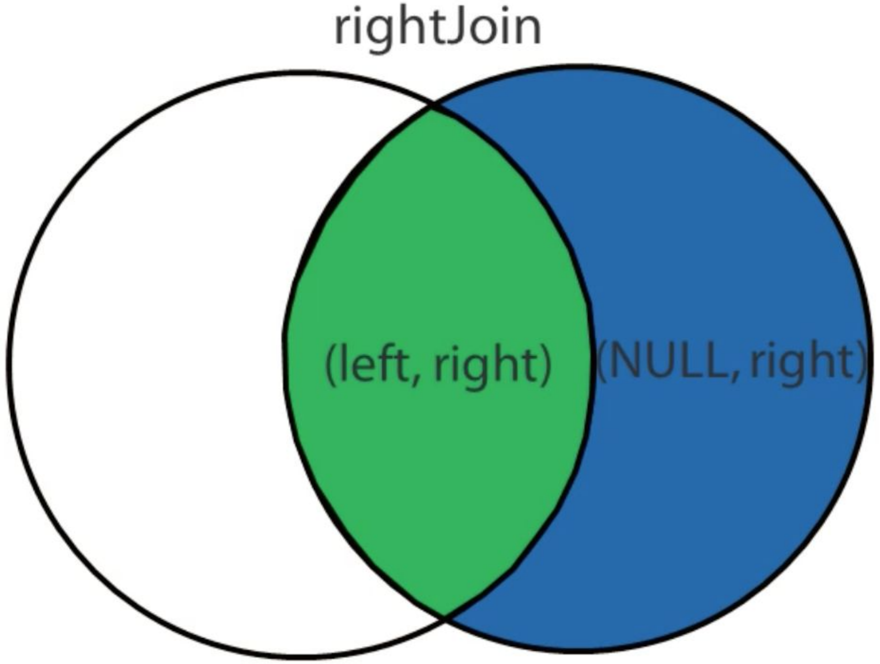


Panorays

leftJoin



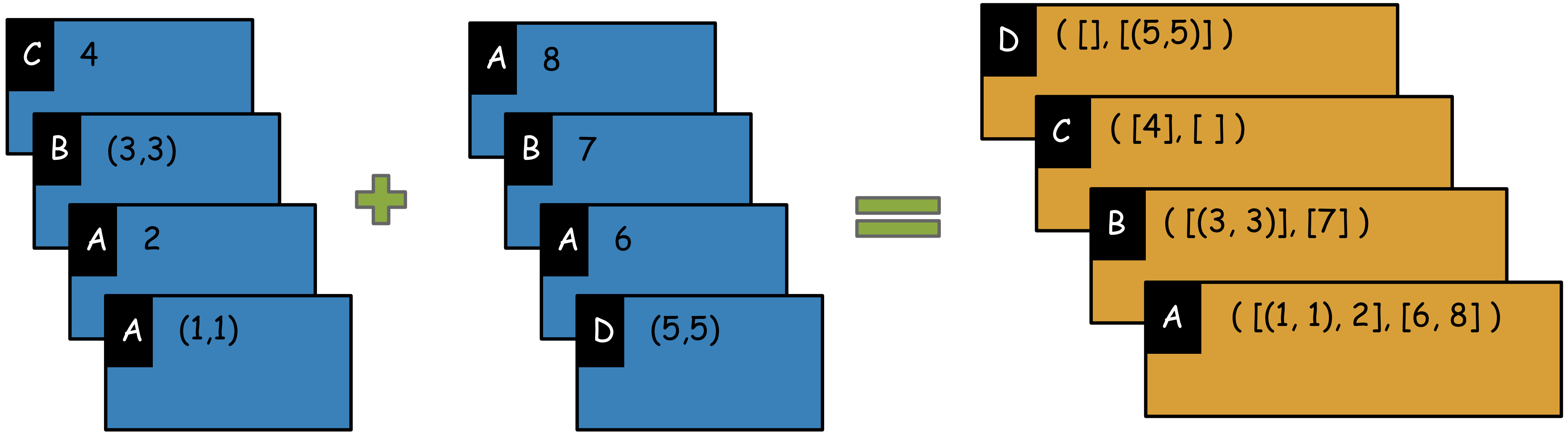
rightJoin



Panorays

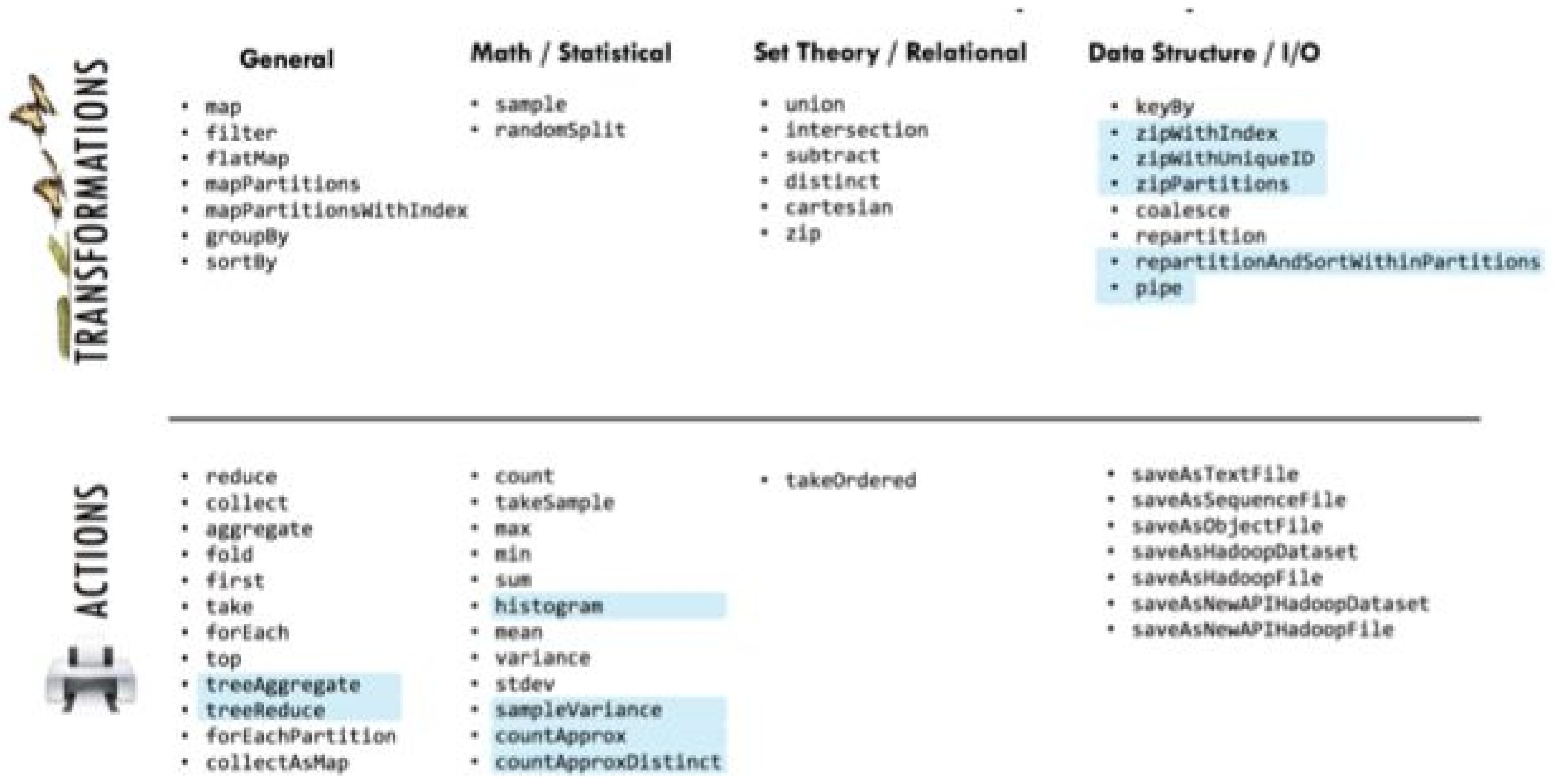
cogroup

- When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W>)) tuples. This operation is also called `groupWith`.



Panorays

List of Transformations and Actions



	General	Math / Statistical	Set Theory / Relational	Data Structure / I/O
TRANSFORMATIONS	• map • filter • flatMap • mapPartitions • mapPartitionsWithIndex • groupBy • sortBy	• sample • randomSplit	• union • intersection • subtract • distinct • cartesian • zip	• keyBy • zipWithIndex • zipWithUniqueID • zipPartitions • coalesce • repartition • repartitionAndSortWithinPartitions • pipe
ACTIONS	• reduce • collect • aggregate • fold • first • take • forEach • top • treeAggregate • treeReduce • forEachPartition • collectAsMap	• count • takeSample • max • min • sum • histogram • mean • variance • stdev • sampleVariance • countApprox • countApproxDistinct	• takeOrdered	• saveAsTextFile • saveAsSequenceFile • saveAsObjectFile • saveAsHadoopDataset • saveAsHadoopFile • saveAsNewAPIHadoopDataset • saveAsNewAPIHadoopFile

Caching Data

Panorays

Cache

- Cache / Persist
  - org.apache.spark.storage.StorageLevel.MEMORY_ONLY
- persist(newLevel: StorageLevel)
  - MEMORY_ONLY
  - MEMORY_AND_DISK
  - DISK_ONLY
  - MEMORY_ONLY_SER
  - MEMORY_AND_DISK_SER
  - …_2 (Can be replicated to another node)
  - OFF_HEAP
- unpersist(blocking: boolean = true)

Panorays

Caching pitfalls

- Always unpersist your cached RDDs, but not too soon.
- Try to cache right after wide transformations
    - To avoid the shuffle in case of failure

Accumulator & Broadcast

Accumulator

- val accumulator = sc.accumulator(0, “Accumulator Name”)
- rdd.foreach(x=> {

  doSomething();

  accumulator += 1;

  })

- In the driver:
  - val accumulatorValue = accumulator.value;

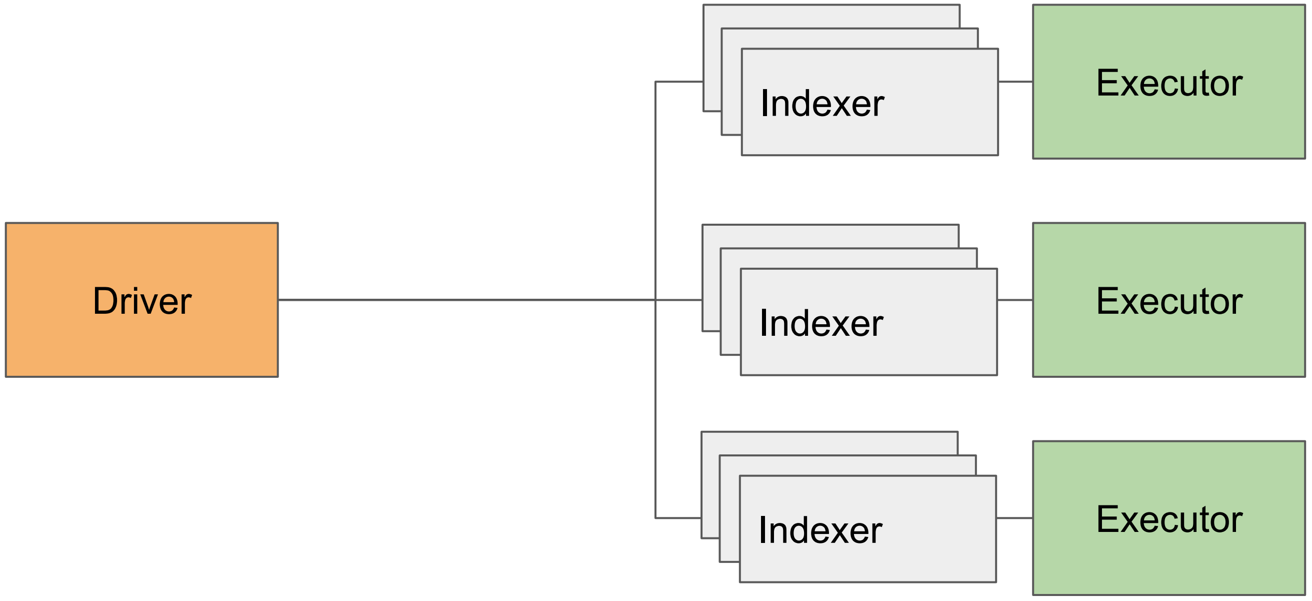
Panorays

Broadcast

```
val indexer = Map(...)

rdd.flatMap(rddVal => indexer.get(rddVal))
```

- What will happen?

Broadcast



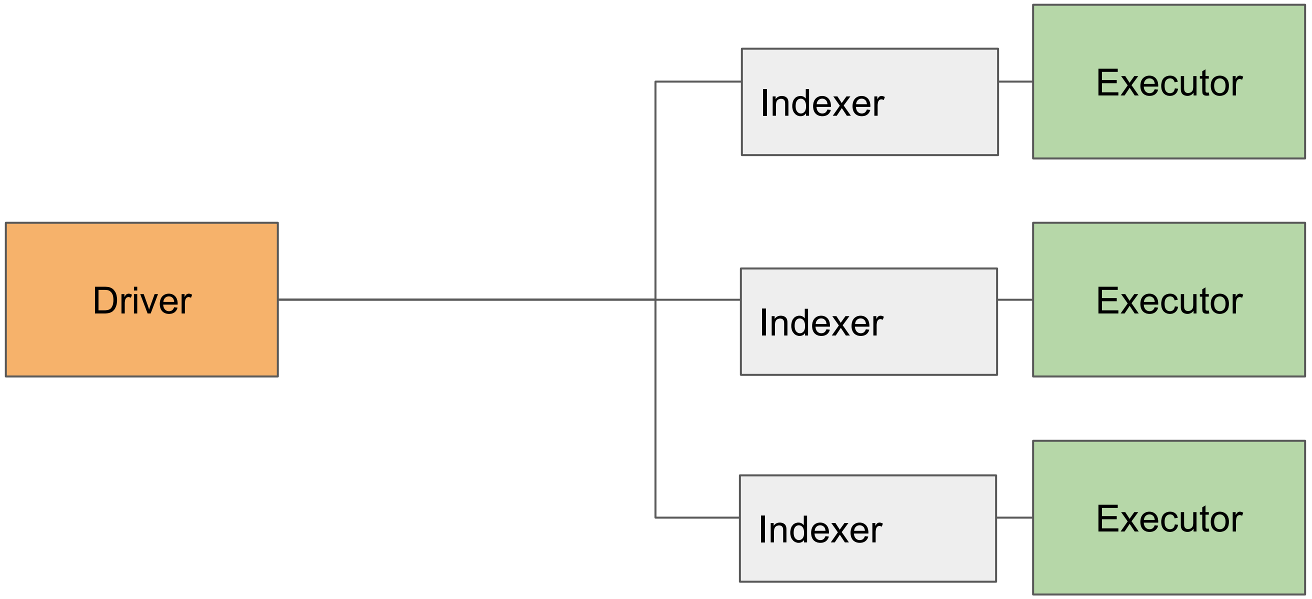
Panorays

Broadcast

```
val indexer = sc.broadcast((Map(....))) // Map = 10Mb; indexer < 10Mb

rdd.flatMap(rddVal => indexer.value.get(rddVal))
```

Panorays

Broadcast



Panorays

Java API

- All implemented with:
  - Org.apache.spark.api.java.function
- JavaPairRDD
  - mapToPair
- Almost all of the API is pretty identical, all of the changes can be found in the documentation

Resources

- RDD Research Paper
  - https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf
- Lambdas
  - https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
- Official Documentation
  - http://spark.apache.org/docs/latest/programming-guide.html

Panorays

Panorays

Mind the
Attack Surface