dimajix

Big Data | Analytics | Algorithms

PySpark Optimizations

dimajix | Dr. Kaya Kupferschmidt | k.kupferschmidt@dimajix.de

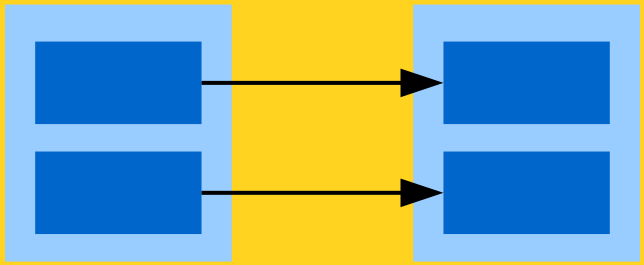
Spark Shuffles

Required Atoms

dimajix

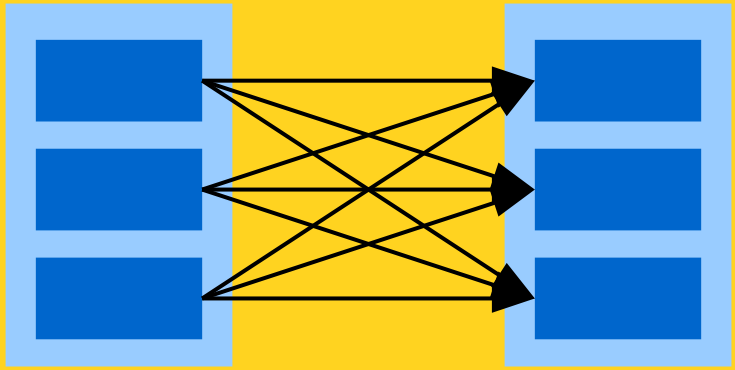
Map Operation

- Reads one record
- Applies any transformation
- Emits 0..n records
- Each output record depends on exactly one input record

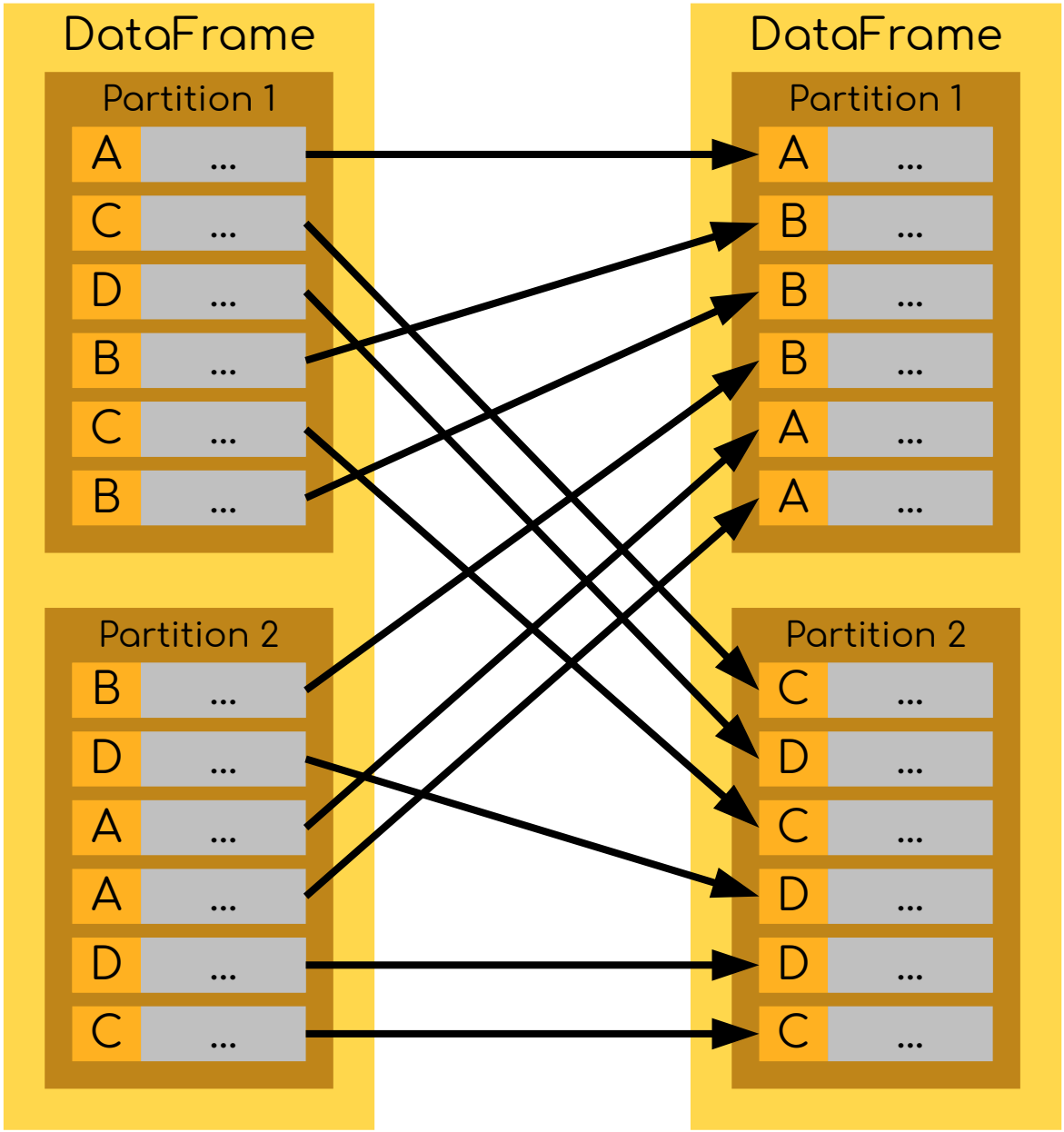


Shuffle Operation

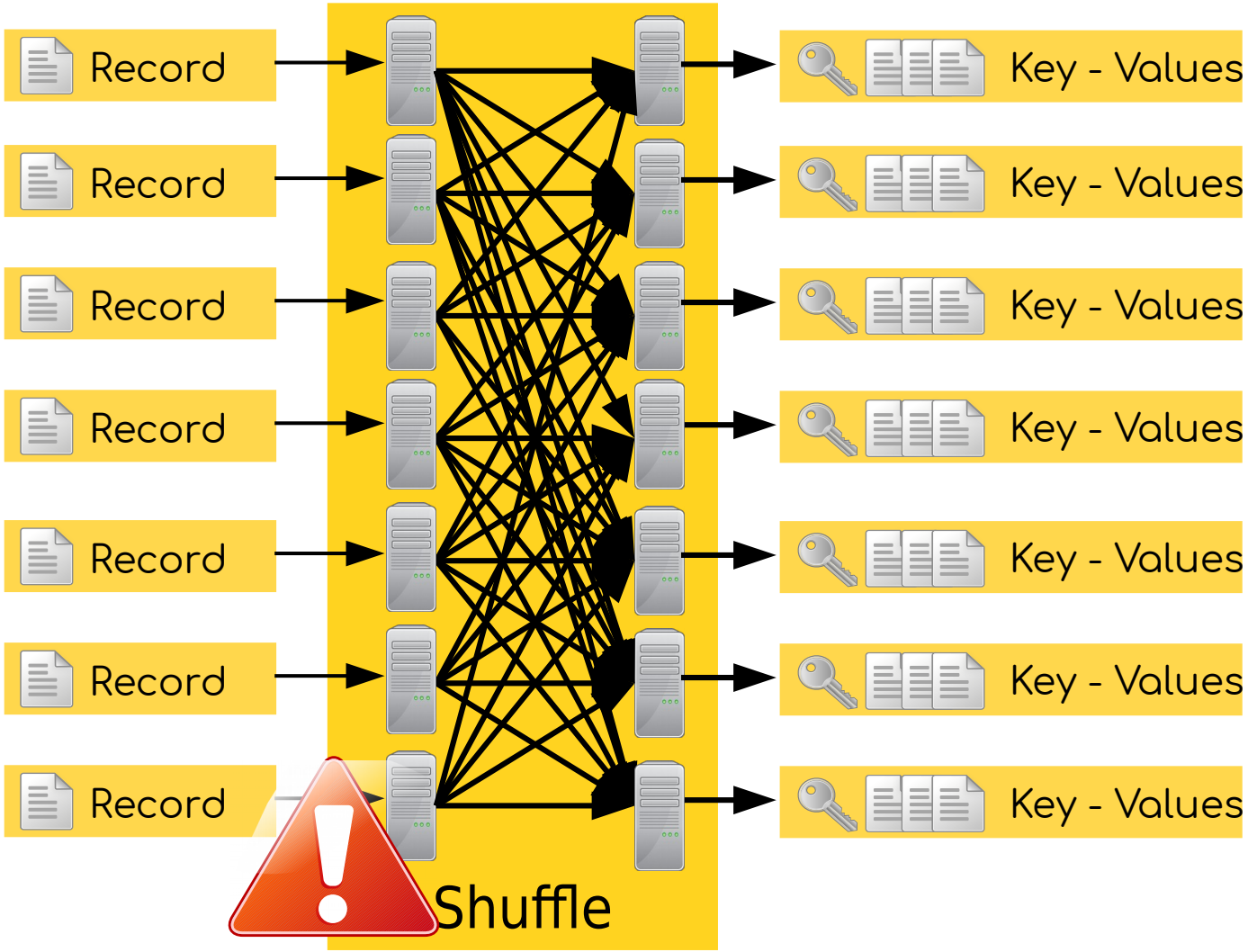
- Collects multiple records with same key
- Results are groups of records



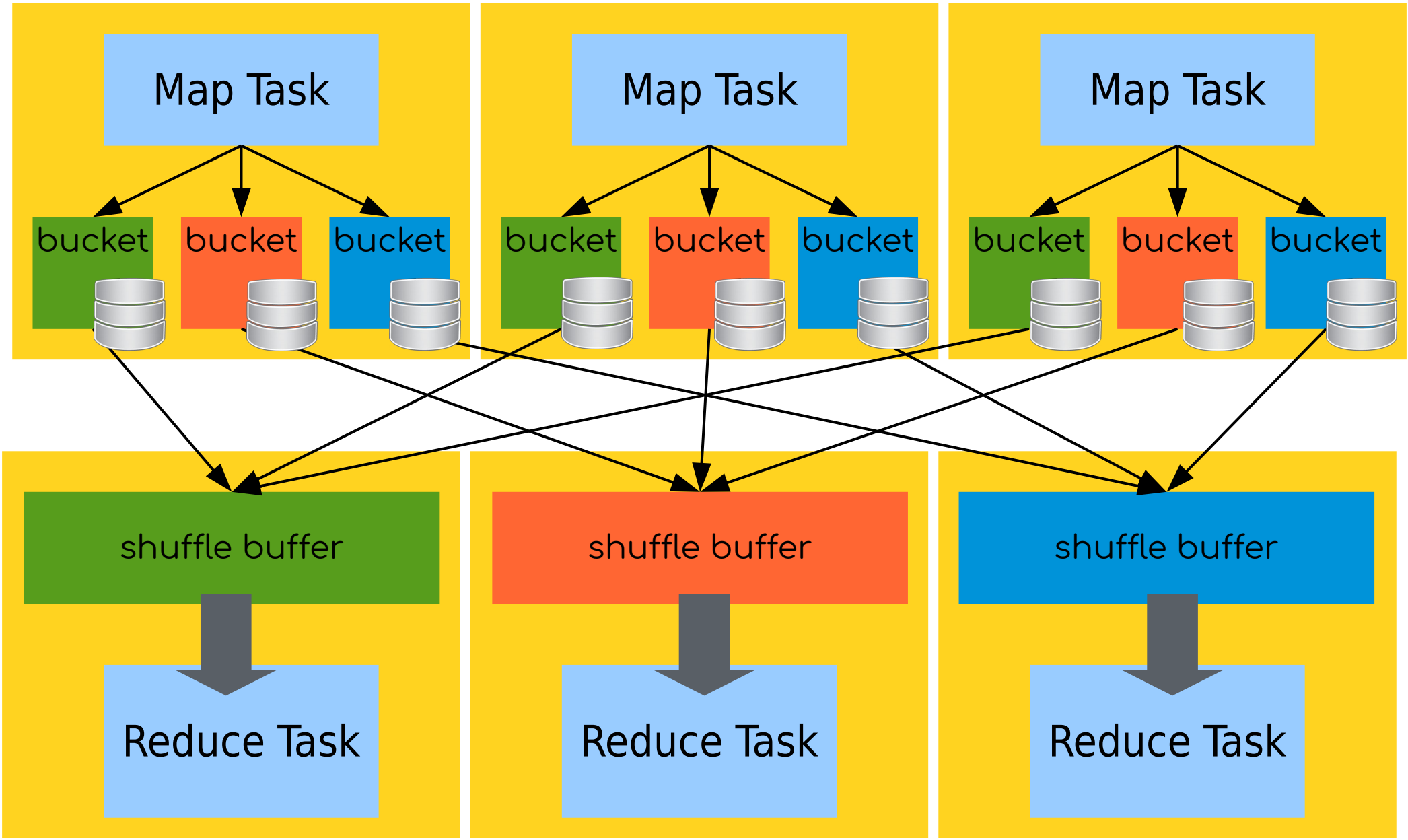
Shuffle Operations



Shuffles in Clusters

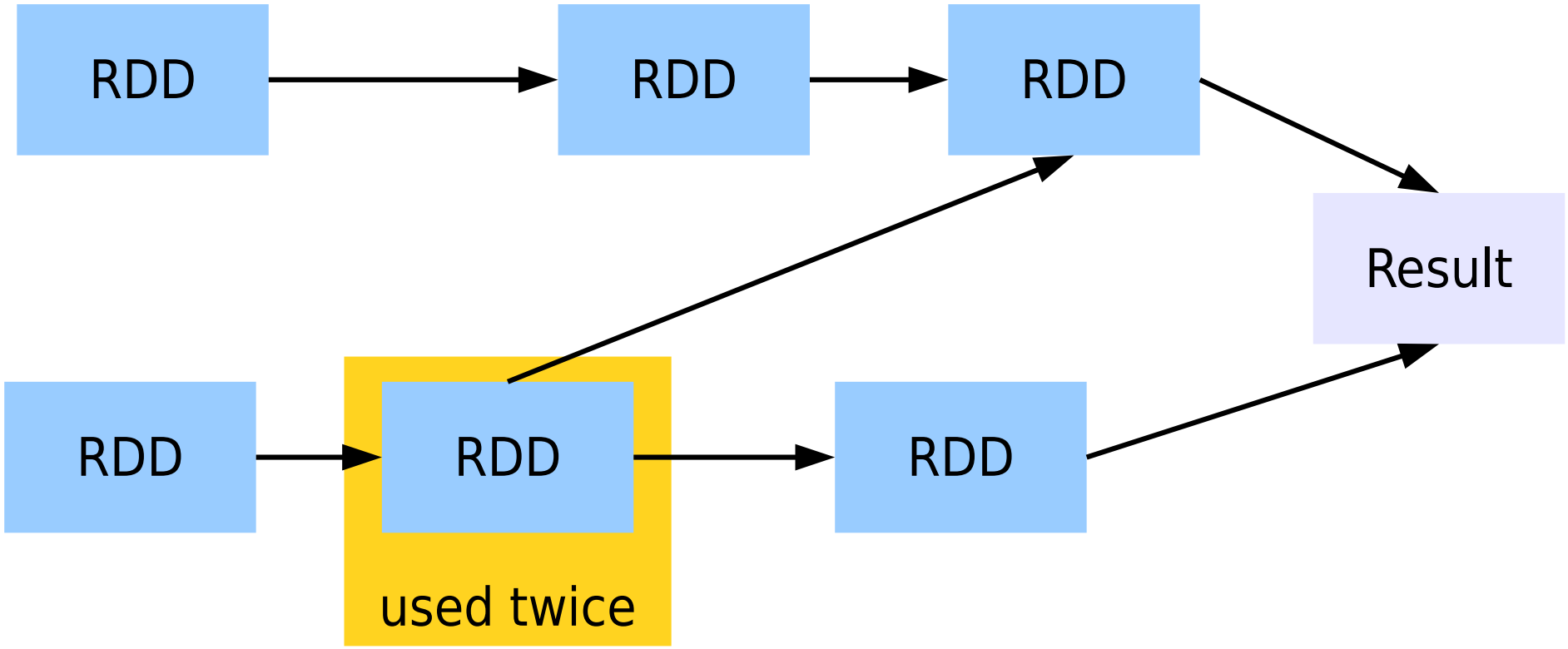


Distributed Shuffle



Caching

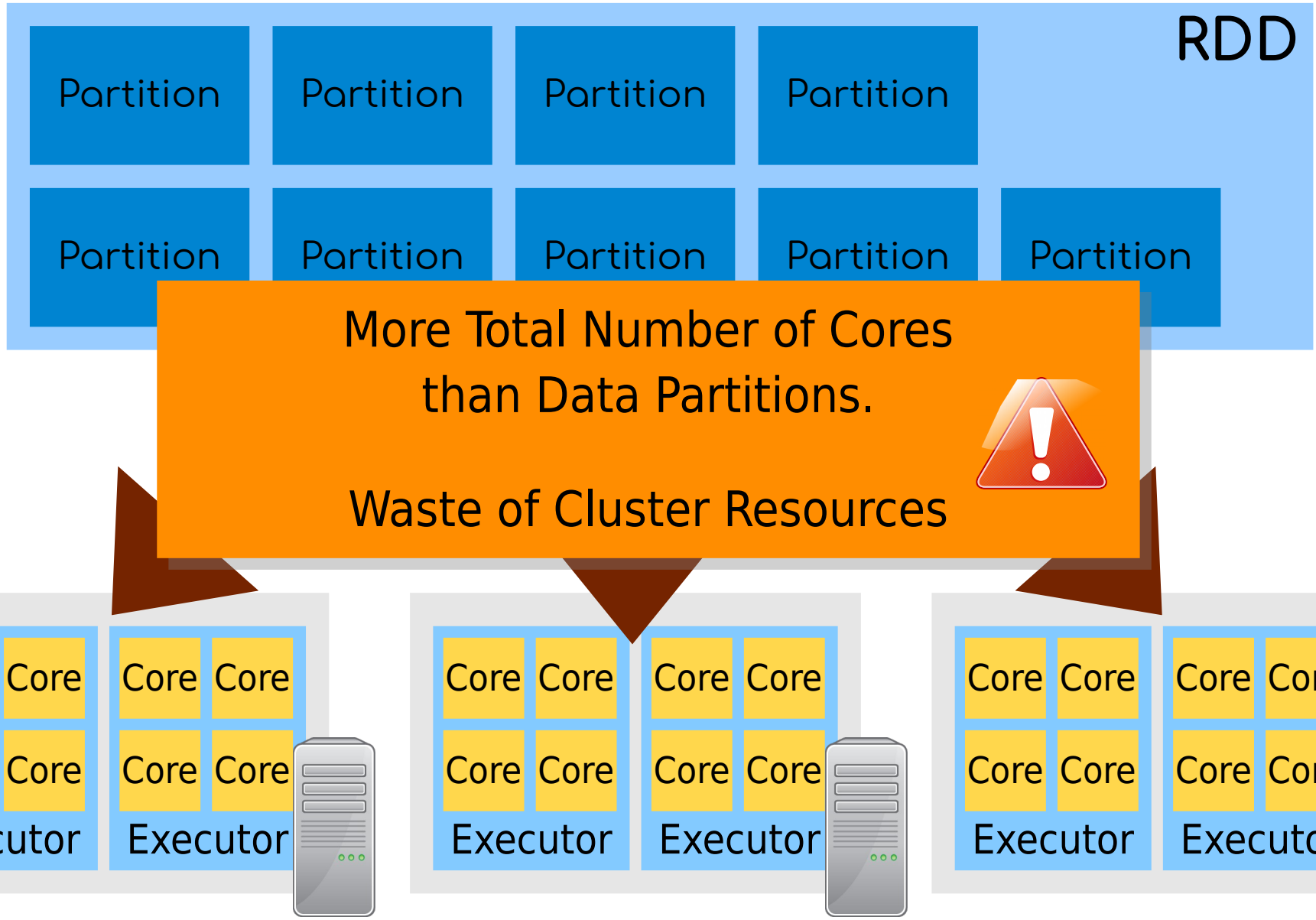
Caching

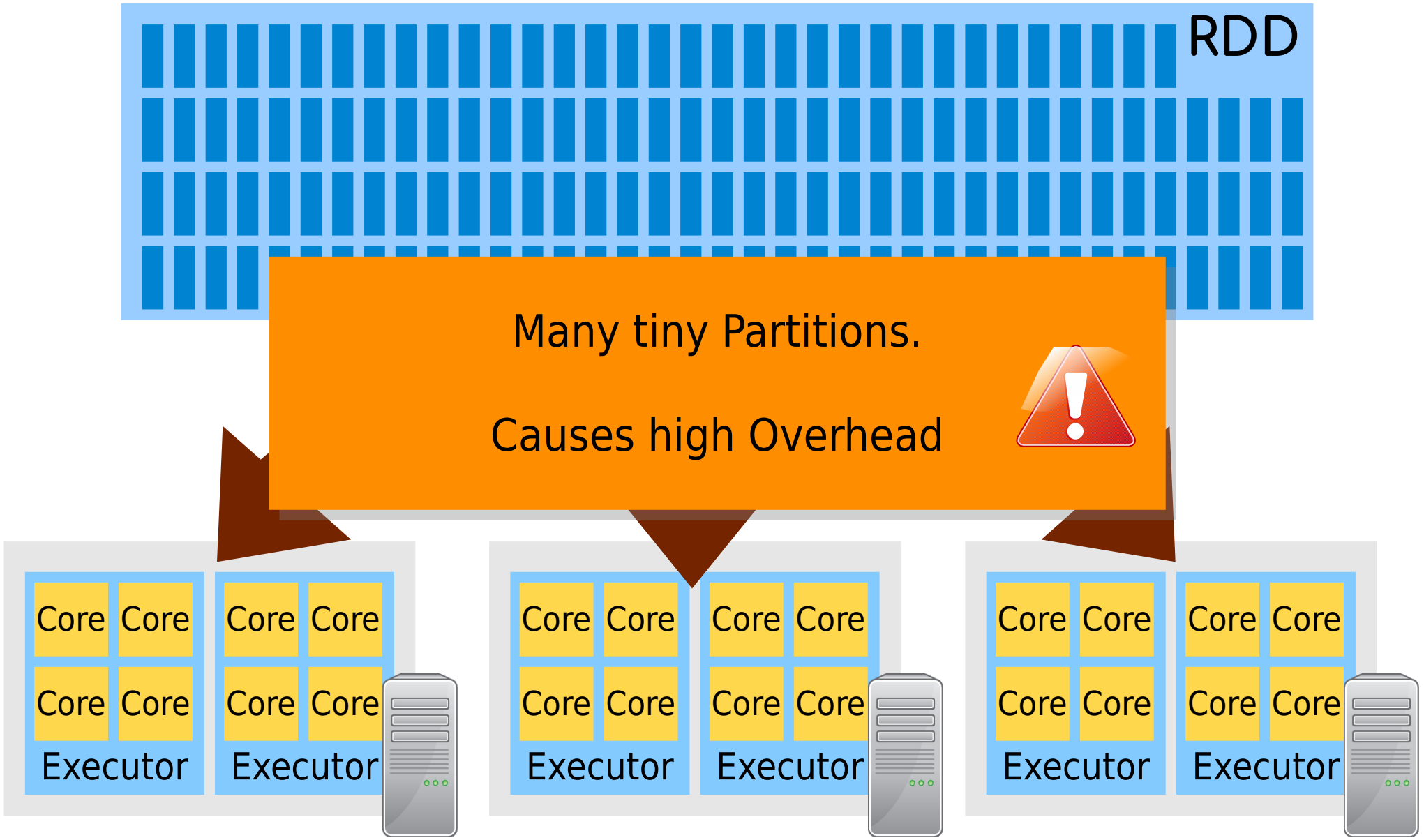


Use **df.cache()** or **df.persist()** for
keeping DataFrames that are repeatedly used.

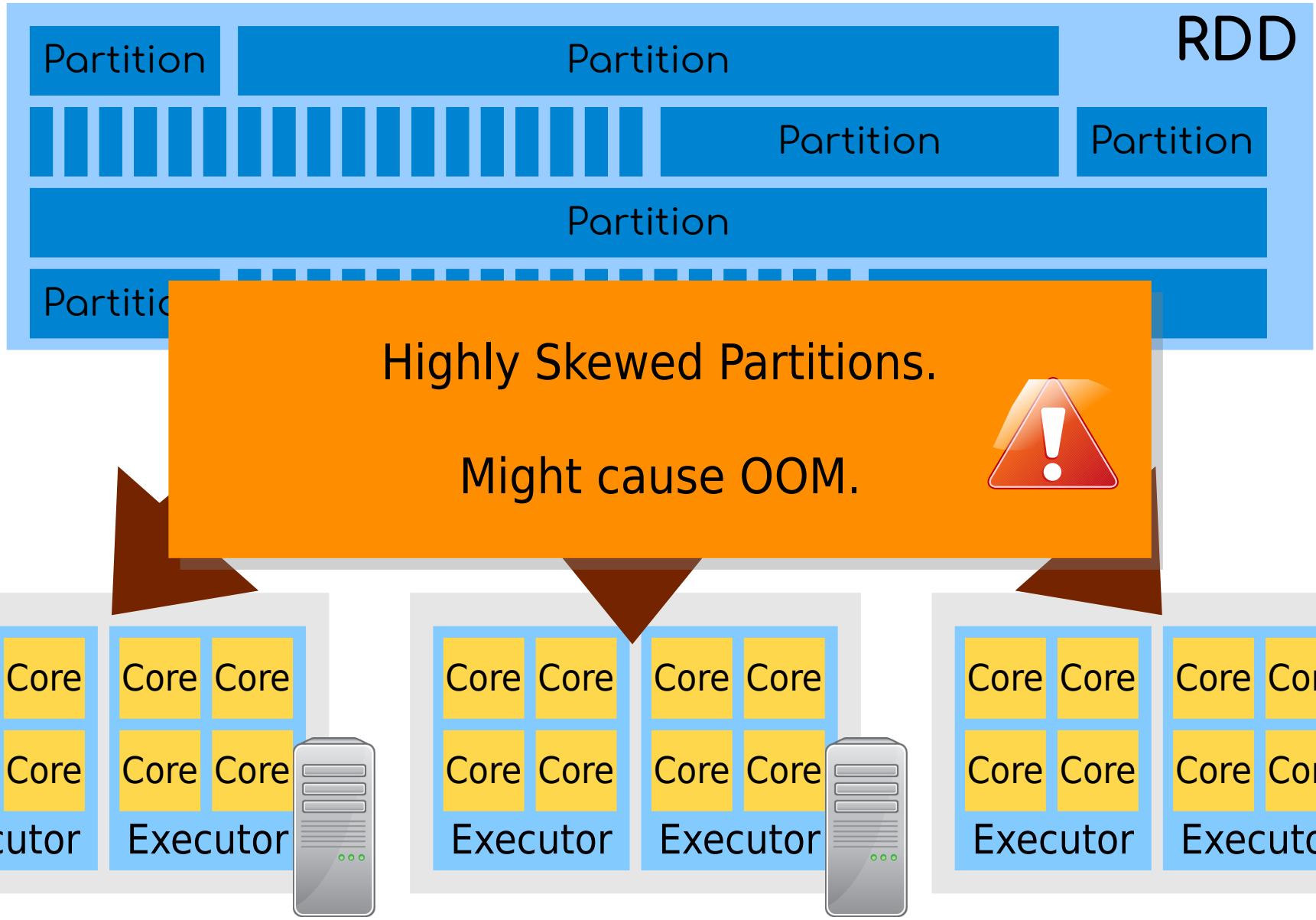
Partitioning

Partitioning




RDD
Many tiny Partitions.
Causes high Overhead
Core Core Core Core
Core Core Core Core
Executor
Executor
Core Core Core Core
Core Core Core Core
Executor
Executor
Core Core Core Core
Core Core Core Core
Executor
Executor

Partitioning



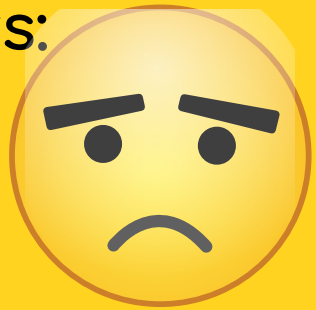
dimajix

Importance of Good Partitiniong

- Large Partitions help to amortize Overhead
- Many Small Partitions help to Utilize Cluster
- But number of Partitions increase overhead!
- Skewed Partitions can cause OOM

No general rule due to many factors:

- Cluster (#Nodes ,RAM, Cores)
- Program (Caching RDDs)
- Data (Skewed Data)



Partitioning RDDs

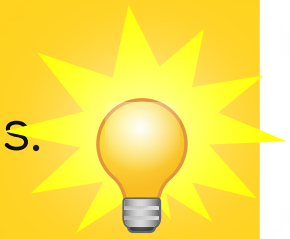
rdd.repartition(numPartitions)

Returns a new RDD which has exactly numPartitions partitions.

rdd.coalesce(numPartitions, doShuffle=False)

Returns a new RDD that is reduced to numPartitions partitions.
Returns a narrow dependency, i.e. no shuffle will occur.

coalesce is useful for limiting number output files.



Partitioning: DataFrames

dimajix

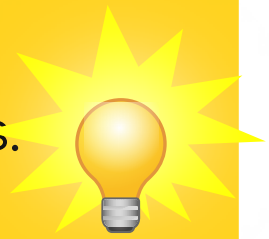
rdd.repartition(numPartitions, *cols)

Returns a new DataFrame which has exactly numPartitions partitions using specified columns for hashing.

rdd.coalesce(numPartitions)

Returns a new DataFrame that is reduced to numPartitions partitions by logically concatenating partitions.
Returns a narrow dependency, i.e. no shuffle will occur.

coalesce is useful for limiting number output files.



Partitioning: Configuration

spark.default.parallelism=2

Default number of partitions of a manually created RDD

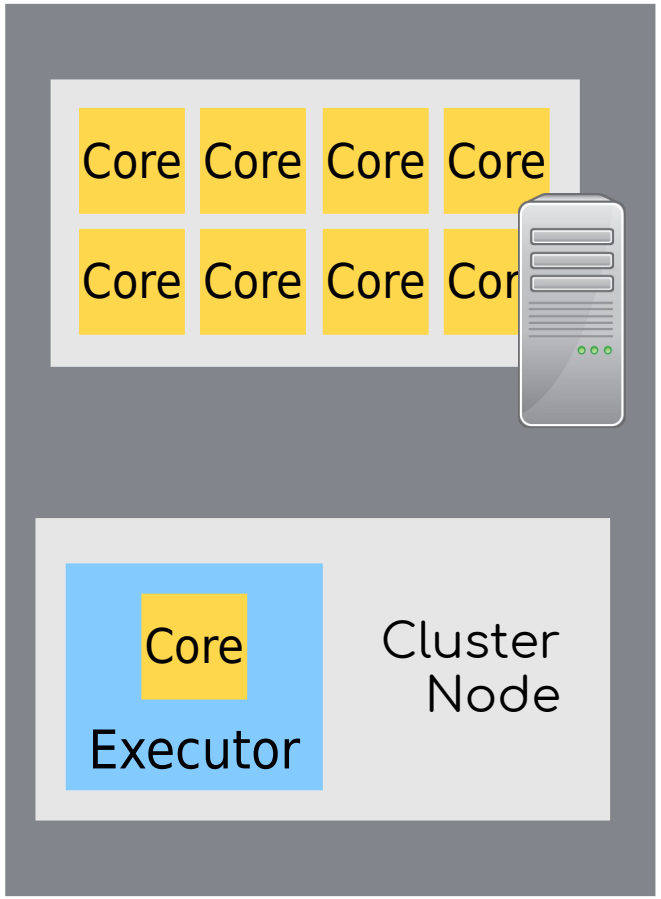
spark.sql.shuffle.partitions=200

Number of partitions returned by a DataFrame shuffle

spark.sql.shuffle.partitions
also controls number output files of SQL operations

Executors

Executor Configuration



Submit Configuration


```
spark-submit \
  <spark options> \
  <my_program.py> \
  <my_program_parameters>
```

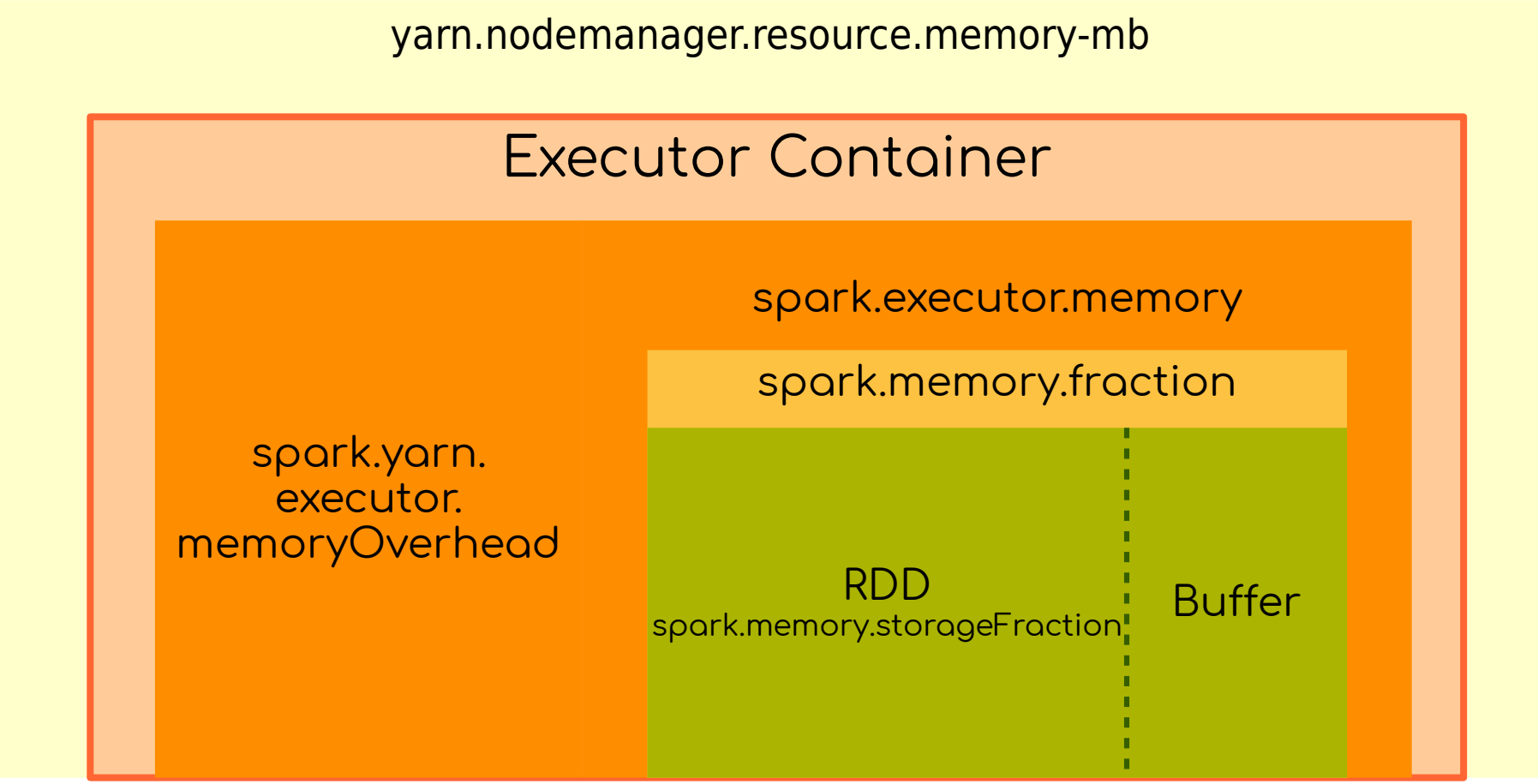
Command Line Option	Description
`--master <master>`	Master for distributing work in the cluster. Normally „yarn“
`--deploy-mode <mode>`	Deploy driver application into cluster or run it on the client. Normally „client“
`--driver-memory <size>`	Amount of memory for driver program. I.e. 1g, 4g,256m
`--executor-memory <size>`	Amount of memory per executor. I.e. 1g, 4g,32g
`--executor-cores <number>`	Number of cores per executor process.
`--conf <variable>=<value>`	Additional configuration properties.

Executor Configuration

Configuration Option	Default	Description
`spark.executor.instances`	`2`	The number of executors.
`spark.executor.cores`	`1`	Number of Cores per Executor Process
`spark.driver.maxResultSize`	`1g`	
`spark.executor.memory`		Amount of memory for driver program. I.e. 1g, 4g,256m
`spark.python.worker.memory`	`512m`	Amount of memory to use per python worker process
`spark.python.worker.reuse`	`true`	Reuse Python worker or not.

Memory Management

Memory Management



Memory Configuration

Configuration Option	Default	Description
`spark.driver.maxResultSize`	`1g`	
`spark.executor.memory`		Amount of memory for driver program. I.e. 1g, 4g,256m
`spark.memory.fraction`	`0.6`	Fraction of (heap space - 300MB) used for execution and storage.
`spark.memory.storageFraction`	`0.5`	Storage space immune against eviction by execution
`spark.yarn.memory.overhead`		The amount of off-heap memory (in megabytes) to be allocated per executor.
`spark.python.worker.memory`	`512m`	Amount of memory to use per python worker process
`spark.python.worker.reuse`	`true`	Reuse Python worker or not.