Engenharia de Computação

Especialização Lato Sensu em Ciência de Dados e Analytics

Soluções em Processamento para Big Data

{ SPARK }

Prof. Jairson Rodrigues
jairson.rodrigues@univasf.edu.br

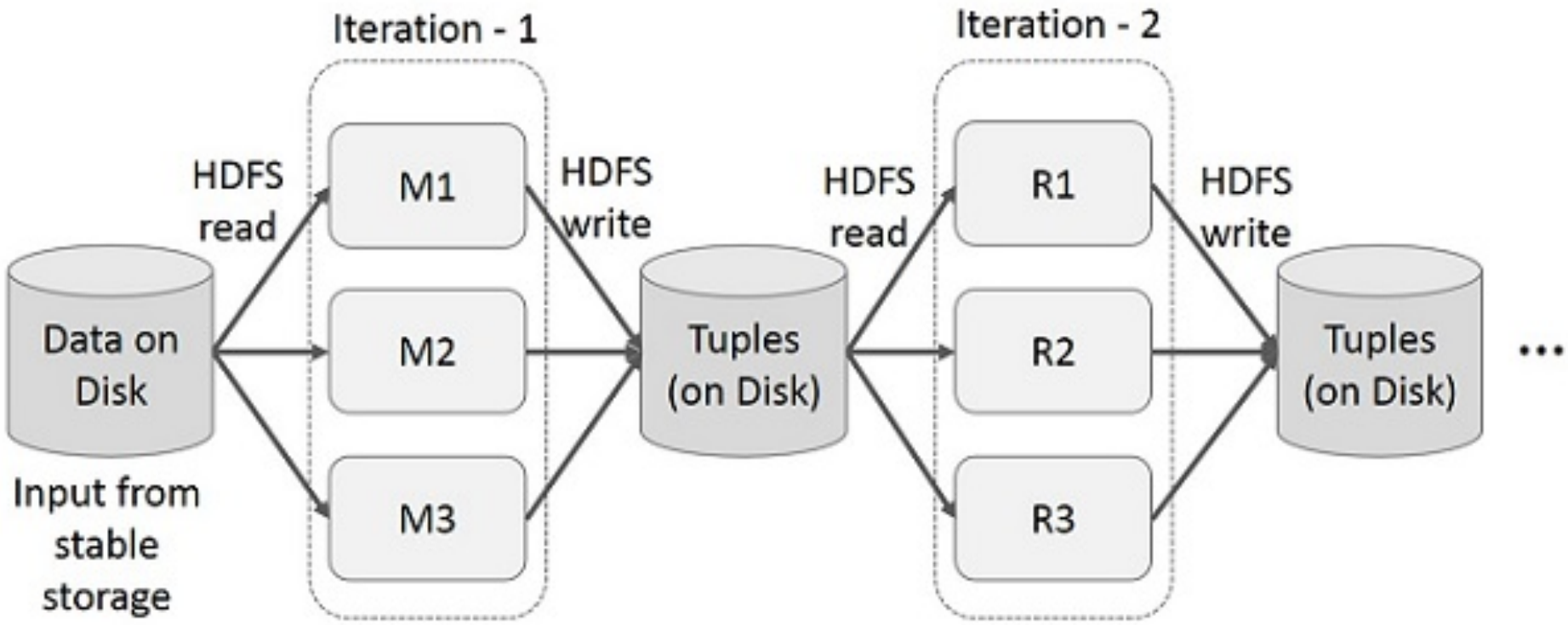
{ introdução }

AGENDA

Spark
Data Frames
Resilient Distributed Datasets
MLLib
GraphX

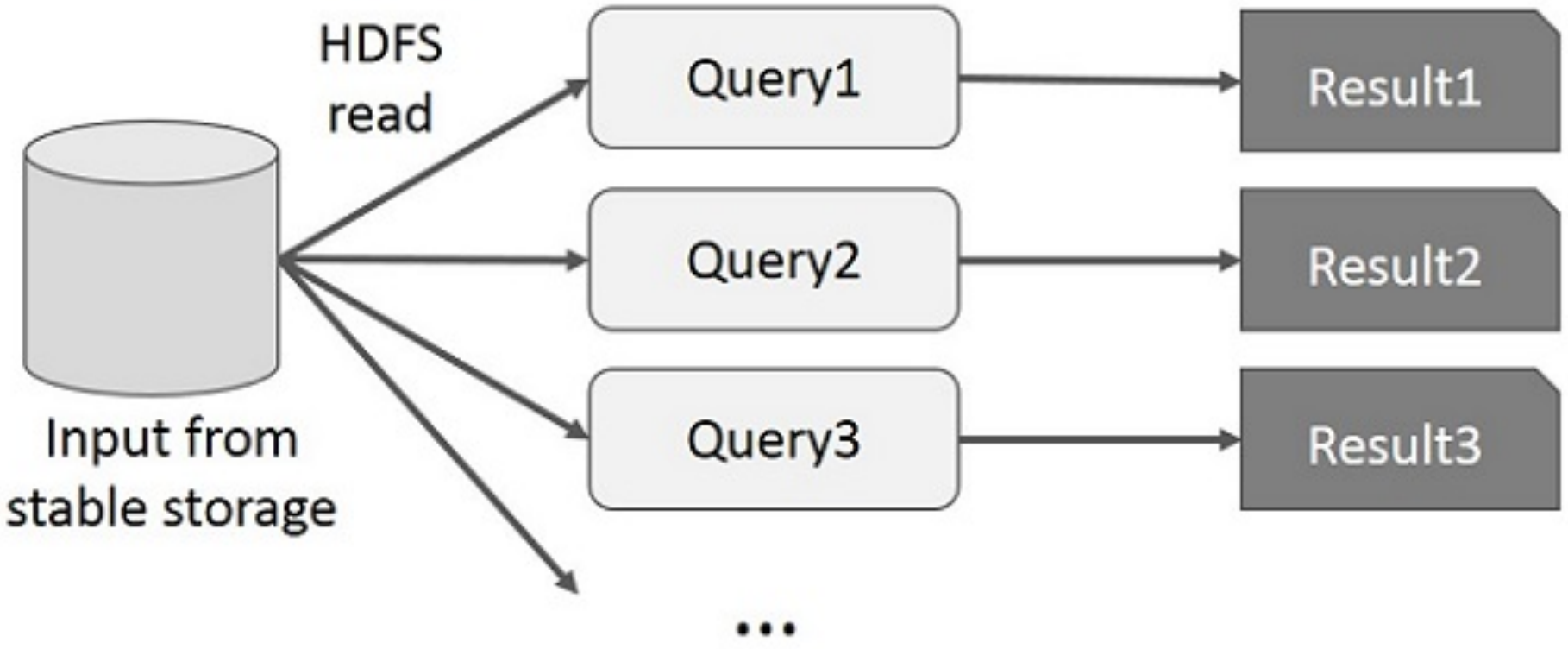
{ mapreduce * - aplicações iterativas }

TF ➡ DF ➡ NTDOC ➡ TFIDF ➡ SORT ➡ ...



* vide slides MapReduce

{ mapreduce * - aplicações interativas }



* vide slides MapReduce

{ apache spark }

lightning-fast cluster computing technology

- Desenvolvido desde de 2009 pelo AMPLab da Universidade da Califórnia em Berkeley
- Em 2010 seu código foi aberto como projeto da fundação Apache
- Idealizado para aumentar a velocidade do processo de computacão Hadoop

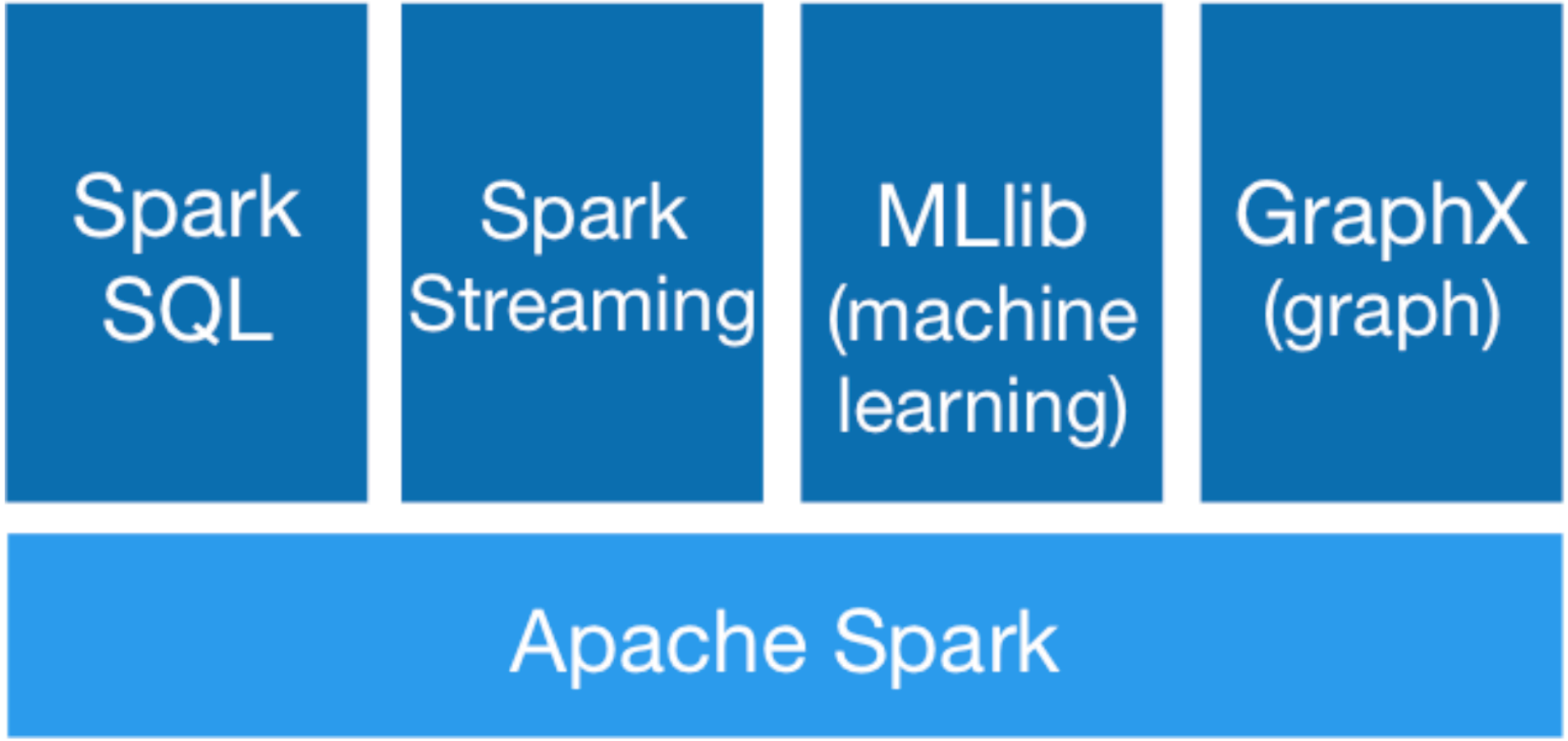
- O que Spark **não** é:
  - Spark não é uma versão modificada do Hadoop
  - Não é dependente do Hadoop, sendo o Hadoop apenas uma das alternativas de uso para Spark

{ apache spark }

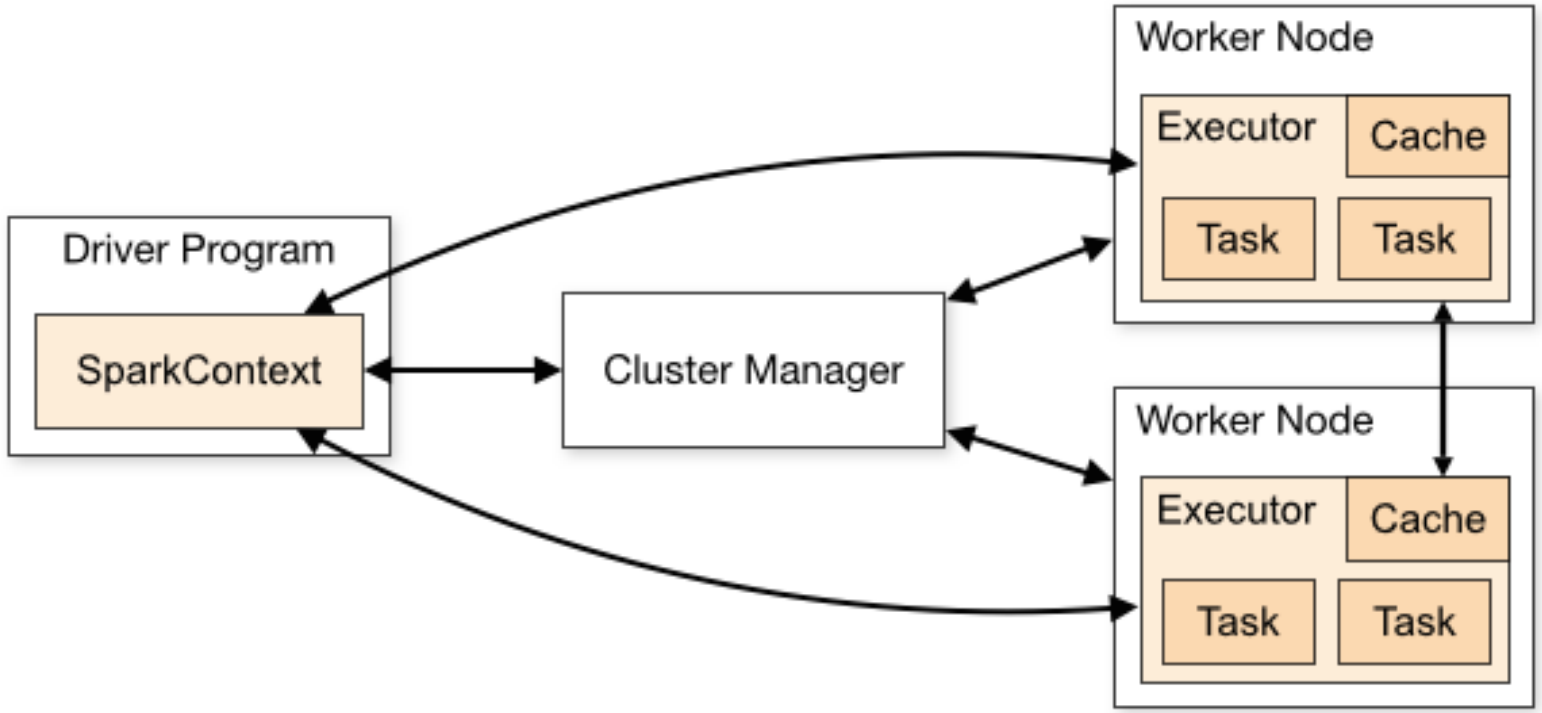
lightning-fast cluster computing technology

- Gerenciador de cluster projetado para processamento rápido de grandes volumes de dados
  - in-memory cluster computing
- Cobertura
  - aplicações em batch
  - algoritmos iterativos
  - consultas interativas
  - streaming

{ spark - componentes }



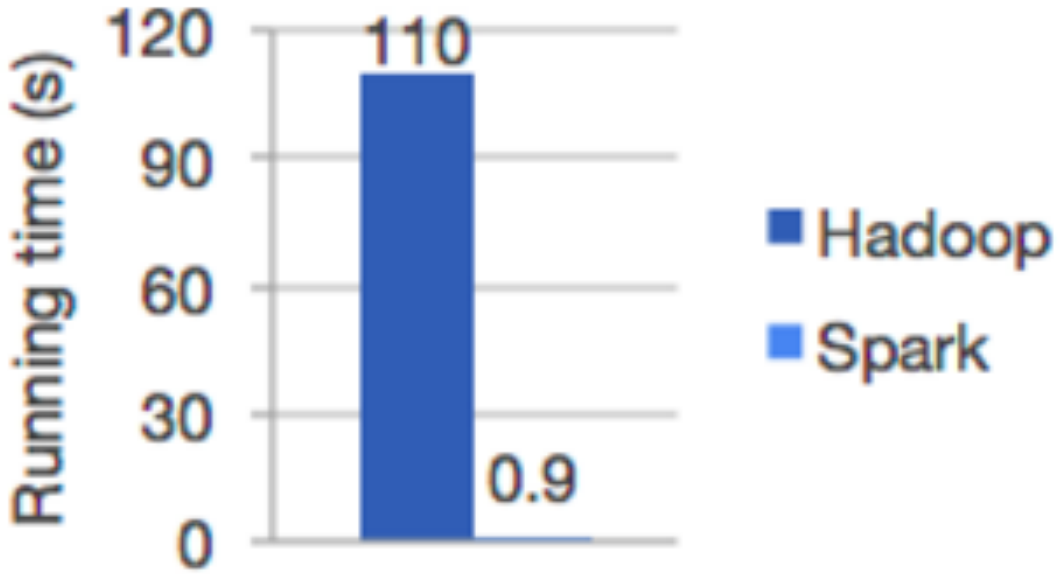
{ spark - visão do funcionamento }



{ spark - componentes }

- Spark Core - engine de computação "in-memory" e funcionalidades centrais do framework
- Spark SQL - suporte para dados estruturados e não-estruturados
- Spark Streaming - operações sobre fluxos de dados
- Spark MLLIb - bibliotecas para aprendizagem de máquina
- Spark GraphX - processamento distribuído de grafos

{ spark - características - desempenho }



Regressão Logística, Spark x Hadoop

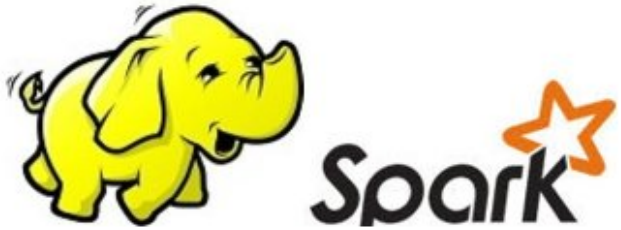
Até 100 vezes mais rápido em memória e 10 vezes mais rápido em disco. Cenário possível graças ao reduzido número de E/S do framework

{ spark - características - frameworks }

Executa em Hadoop YARN, Mesos ou modo standalone

Acessa diversas fontes de dados, incluindo HDFS, Cassandra, HBase, S3 etc

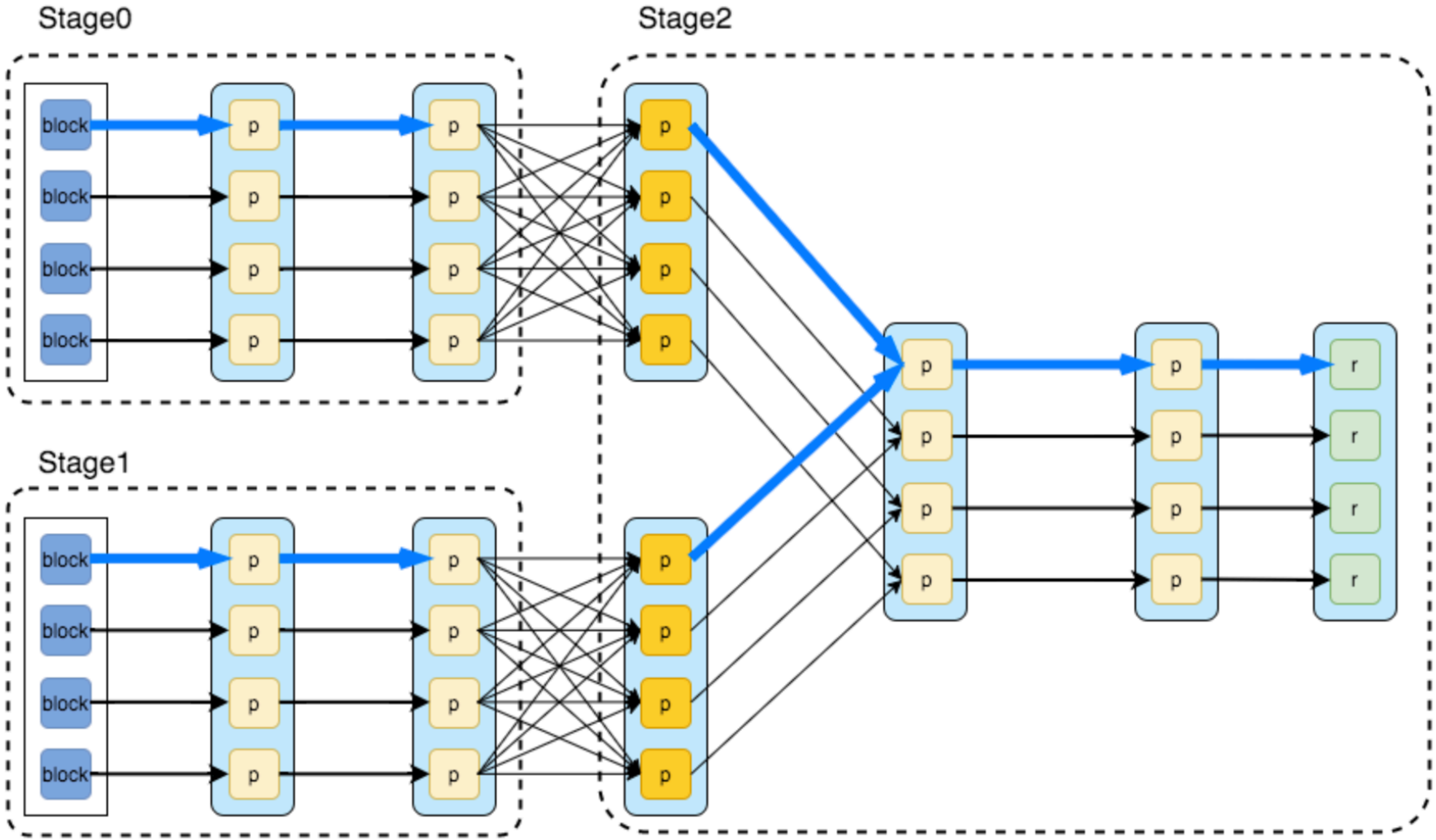
{ spark - características - linguagens }



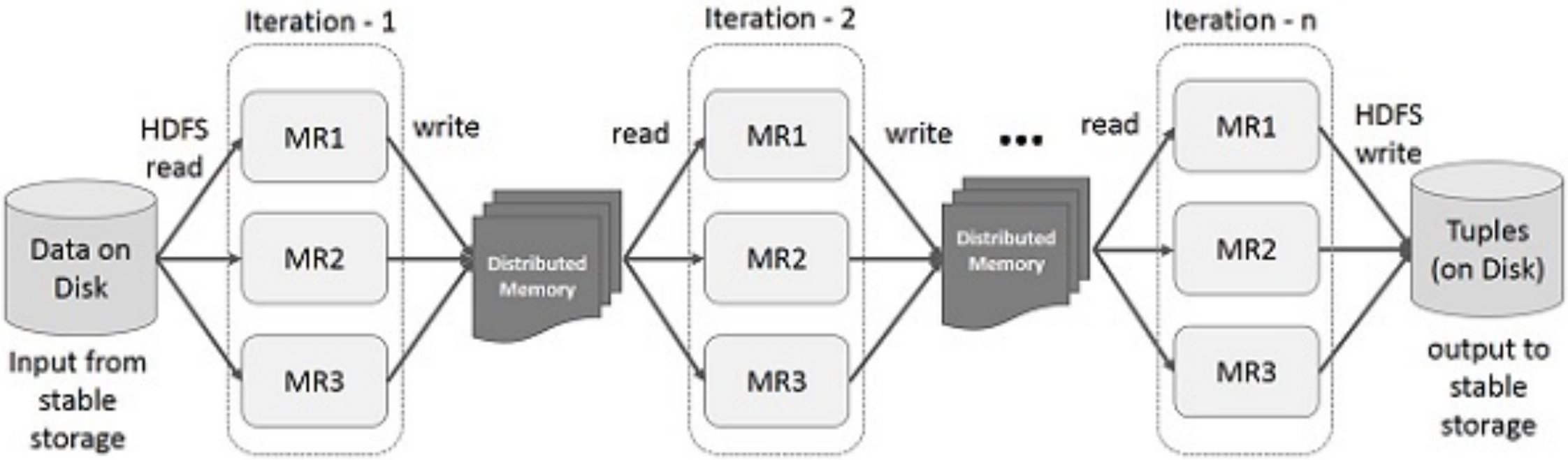
{ resilient distributed datasets - RDD }

- Coleção imutável de dados, dividida em partições
- Cada partição lógica pode ser computada por diferentes nós
- São coleções de dados tolerantes à falha
- Criação de RDD
  - a partir da "paralelização" de um conjunto de dados
  - ao referenciar um conjunto de dados do HDFS, HBase, S3 etc

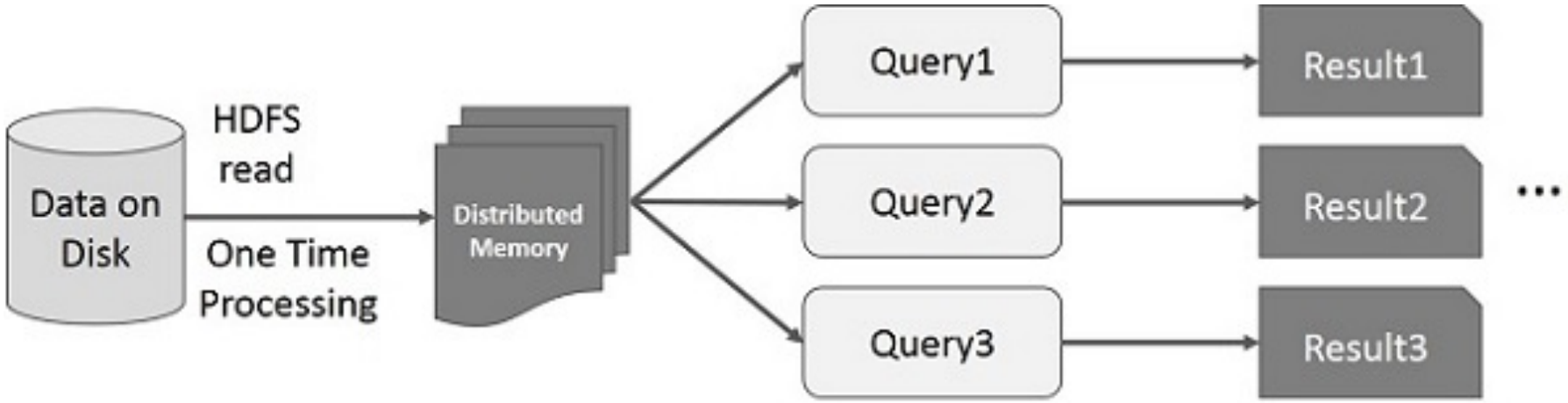
{ resilient distributed datasets - RDD }



{ spark - aplicações iterativas }



{ spark - aplicações interativas }



{ primeiros passos / shell }

- Análise interativa com shell para spark
- Ferramenta para analisar dados
- Executa scripts Scala
  - **$ spark-shell**
- Executa scripts Python
  - $ pyspark

{ primeiros passos }

Texto base [hdfs://namenode:9000/poli/base.txt]
Comandos do exercício => http://bit.ly/2kpQL0h

A UPE fornece o curso de ciência dos dados que possui na grade curricular a disciplina soluções para processamento de grandes volumes de dados
A disciplina possui conteúdo voltado para armazenamento e processamento de dados em escala muito alta utilizando frameworks como Hadoop MapReduce e Spark

Por simplicidade, ignoramos pontos, vírgulas e correlatos no exemplo

{ primeiros passos }

- Apontando para o arquivo

```
$ scala> val arquivo = sc.textFile("hdfs://hadoop-master:8020/poli/base.txt")
```

```
arquivo: org.apache.spark.rdd.RDD[String] = /poli/base.txt MapPartitionsRDD[1]
```

- Contando linhas do arquivo

```
$ scala> arquivo.count()
```

```
$ res0: Long = 6
```

- Recuperando o primeiro item

```
$ scala> arquivo.first()
```

```
$ res1: String = "A UPE fornece o curso de ciência dos dados que possui na grade "
```

{ primeiros passos }

- Aplicando filtros

```
$ scala> val filtro = arquivo.filter(line =>
line.contains("dados"))
$ filtro: org.apache.spark.rdd.RDD[String]

$ scala> filtro.collect()
$ Array[String] = Array("A UPE fornece o curso de ciência dos dados que possui na grade ", volumes de dados, "processamento de dados em escala muito alta utilizando ")

$ filtro.first()
$ filtro.take(5)
$ filtro.collect()
```

{ primeiros passos }

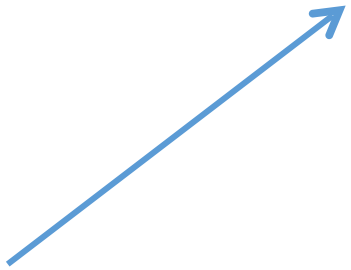
- Quantas palavras há na maior linha?

```
$ scala > arquivo.map(line => line.split(" ").size)
                    .reduce((a, b) => Math.max(a, b))
```

```
Array[Int] = Array(13, 8, 3, 8, 8, 6)

$ Res3: Int = 13

Max(13,8) = 13
Max(Max(13,8),3) = 13
Max(Max(Max(13,8),8) = 13
...
```



{ primeiros passos }

- Word Count

```
$ scala> val palavras = arquivo.flatMap(line => line.split(" "))
                                .map(word => (word, 1))
                                .reduceByKey((a, b) => a + b)
palavras: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[6]
```

- Visualizando os resultados

```
$ scala> palavras.collect()
res3: Array[(String, Int)] = Array((processamento,2), (escala,1),
(armazenamento,1), (para,2), (e,2), (curso,1), (que,1), (em,1), (volumes,
1), (disciplina,2), (fornece,1), (Spark,1), (voltado,1), (muito,1), (a,1),
(de,4), (A,2), (MapReduce,1), (dos,1), (o,1), (alta,1), (ciência,1),
(frameworks,1), (possui,2), (curricular,1), (grade,1), (UPE,1), (soluções,
1), (grandes,1), (dados,3), (conteúdo,1), (como,1), (na,1), (utilizando,
1), (Hadoop,1))
```

{ cache }

- RDD.cache()
  - Armazena os resultados intermediários em memória distribuída
- Imagine descobrir quantas palavras há na menor linha?
  - Operação map não precisaria ser repetida

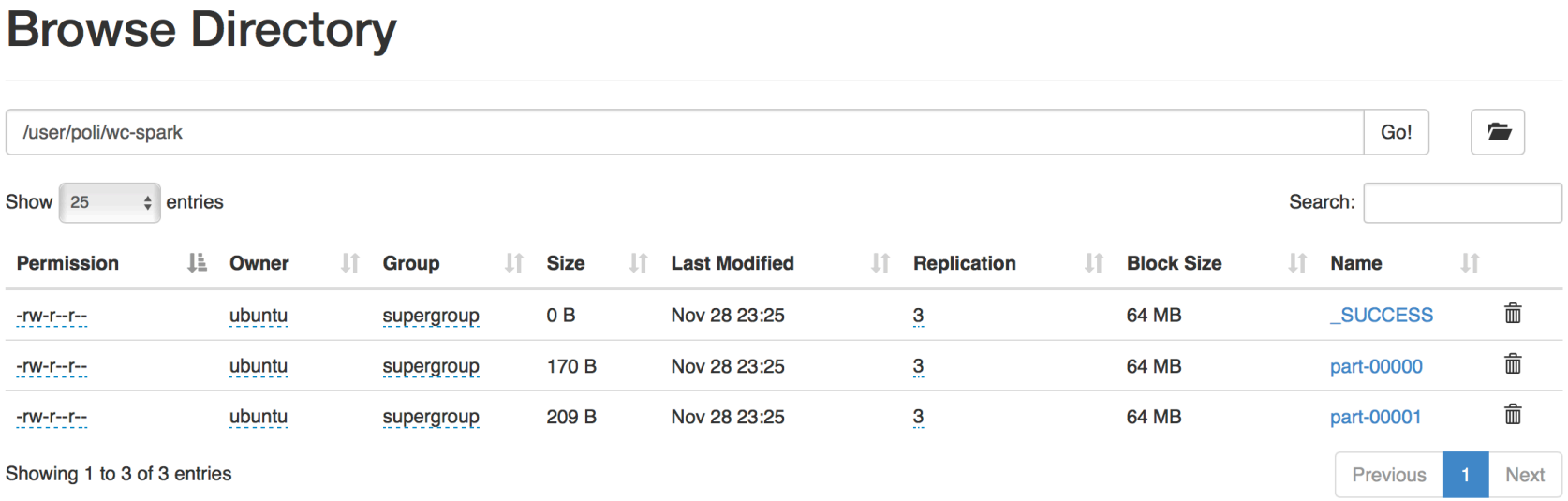
```
$ scala > arquivo.map(line => line.split(" ").size).cache()
                    .reduce((a, b) => Math.max(a, b))
$ Res4: Int = 13

$ scala > arquivo.map(line => line.split(" ").size)
                    .reduce((a, b) => Math.min(a, b))
$ Res5: Int = 3
```

{ persistência }

- RDD.saveAsTextFile(path)
  - Persiste o conteúdo do RDD em armazenamento secundário

```
$ scala> arquivo.flatMap(line => line.split(" "))
                .map(word => (word, 1))
                .reduceByKey((a, b) => a + b)
                .saveAsTextFile("hdfs://hadoop-master:
8020/poli/wc-spark")
```



{ operações de filtragem }

- Quantas linhas do texto base possuem a palavra dados?

  $ scala > val result = arquivo.filter(line => line.contains("dados"))

  $ scala > result.count()

- Ou, simplesmente

  $ scala > arquivo.filter(line => line.contains("dados")).count()

  Res6 = 3

{ recuperando arquivo salvo }

FORA DO SPARK-SHELL

```
vagrant@spark-node1:~$ hadoop fs -text /poli/wc-spark/part*
(armazenamento,1)
(para,2)
(curso,1)
(em,1)
(disciplina,2)
(fornece,1)
(muito,1)
(dos,1)
(MapReduce,1)
(alta,1)
(curricular,1)
(UPE,1)
 ...
```

{ spark - transformações }

Transformação	Significado
`map(func)`	Retorna um novo RDD a partir da aplicação de cada elemento do RDD origem à função func
`filter(func)`	Seleciona os elementos do RDD origem cujos critérios atendem à função func (TRUE)
`flatMap(func)`	Similar a map(), “achata” o resultado para retorno unidimensional
`sample(withReplacement, fraction, seed)`	Retorna uma amostra aleatória do conjunto de dados
`union(dataset)`	União entre dois conjuntos de dados
`intersection(dataset)`	Interseção entre dois conjuntos de dados
`groupByKey([numTasks])`	Dado um conjunto (K,V) => conjunto (K, Iterable <V>). Opcionalmente pode-se informar o nível de paralelismo desejado (numTasks)

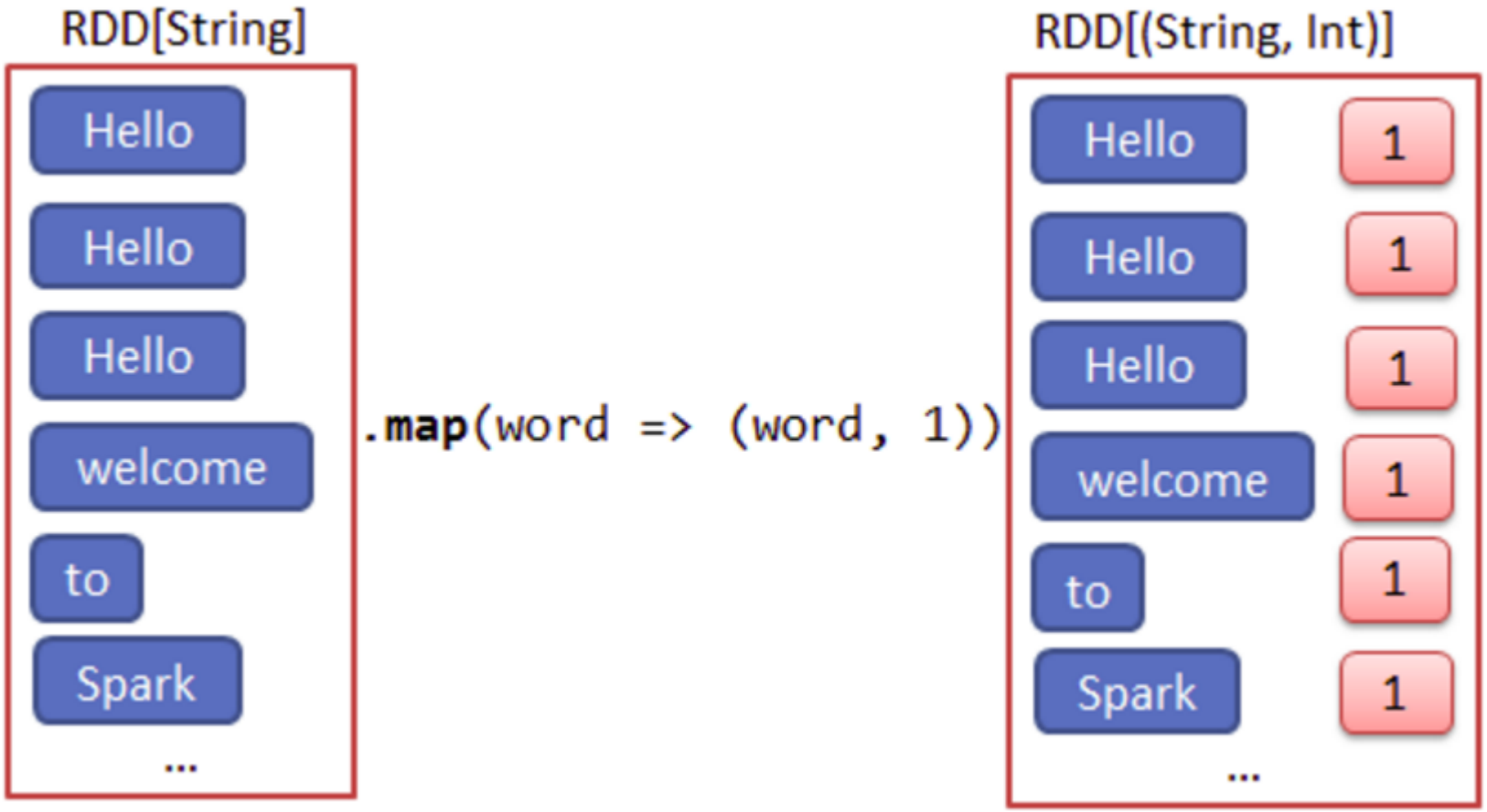
{ spark - transformações }

Transformação	Significado
`reduceByKey(func, [numTasks])`	Dado um conjunto (K,V) retorna um novo conjunto (K,V) onde os valores de cada chave são agregados usando a função de redução func. Paralelismo configurado por numTasks
`sortByKey([ascending], [numTasks])`	Dado um conjunto (K,V) retorna outro conjunto (K,V) ordenado por K, opções para ordenações ASC ou DESC
`join(otherDataset, [numTasks])`	Dados dois conjuntos (K,V) e (K,W), retorna um conjunto (K, (V,W))
`coalesce(numPartitions)`	Diminui o número de partições no RDD para a quantidade numPartitions informada. Útil para ganhar mais eficiência após filtragem de grandes volumes
`repartition(numPartitions)`	Reorganiza a distribuição de partições no RDD randomicamente, podendo gerar mais ou menos partições.
`...`	...

Referência completa em http://spark.apache.org/docs/latest/programming-guide.html#transformations

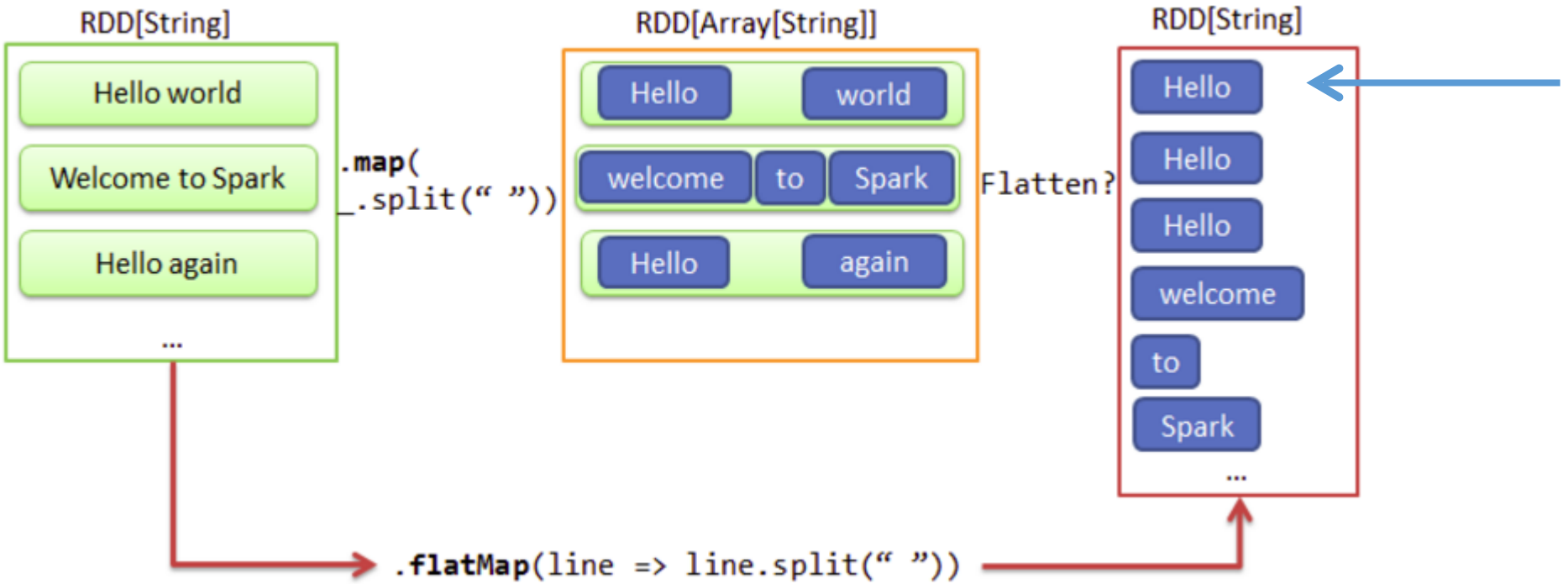
{ uma nota sobre `map` e `flatMap` }

map(f: T => U)



{ uma nota sobre `map` e `flatMap` }

flatmap(f: T => [U])



{ spark - ações }

Transformação	Significado
`reduce(func)`	`Agrega elementos de um conjunto de dados usando a função func.`
`collect()`	Retorna os elementos de um conjunto de dados. É usualmente utilizado após operações filter que asseguram um subconjunto pequeno dos dados como retorno
`count()`	Número de elementos do conjunto de dados
`first()`	Retorna o primeiro elemento do conjunto de dados
`take(n)`	Retorna os primeiros n elementos do conjunto de dados
`...`	...

Referência completa em http://spark.apache.org/docs/latest/programming-guide.html#actions

{ quadro geral de ações e transformações spark [3] }

Transformations	*map*(*f* : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
	filter(*f* : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
	flatMap(*f* : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(*f* : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(*f* : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(*c* : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(*p* : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	:	RDD[T] ⇒ Long
	collect()	:	RDD[T] ⇒ Seq[T]
	reduce(*f* : (T, T) ⇒ T)	:	RDD[T] ⇒ T
	lookup(*k* : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS

{ referências }

[1] 2016 Hadoop Maturity Survey Results
http://info.atscale.com/2016-hadoop-maturity-survey-results-report

[2] Landset, Sara, et al. "A survey of open source tools for machine learning with big data in the Hadoop ecosystem." Journal of Big Data 2.1 (2015): 24.

[3] Zaharia, Matei, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Technical Report UCB/EECS-2011-82, EECS Department, University of California, Berkeley, 2011.