Apache Spark

CARLOS EDUARDO MARTINS RELVAS

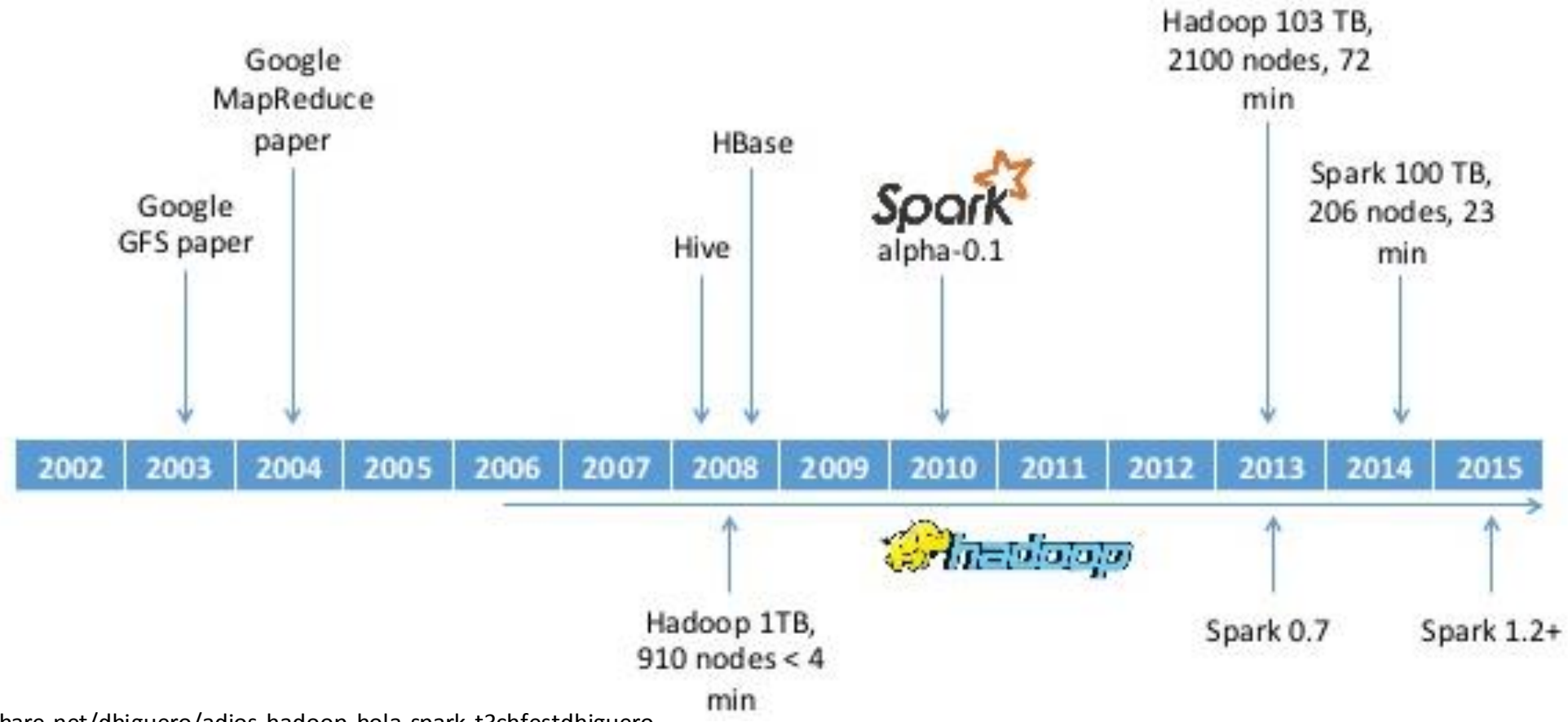
INTRODUÇÃO À COMPUTAÇÃO PARALELA E DISTRIBUÍDA

Apache Hadoop

- Mudou a forma de armazenamento e processamento de dados em um cluster de computadores, trazendo as seguinte propriedades:
  - Tolerância a falhas (replicação dos dados em diferentes nodes). Distribuição dos dados no momento do armazenamento.
  - Trazer a computação para onde o dado está armazenado (Map Reduce).
- Spark leva isto a outro patamar.
  - Dados são distribuídos em memória.

MapReduce é acíclico. Bom para gerenciamento de falhas, mas ruim reuso de dados (processos iterativos).

Time Line



Fonte: http://pt.slideshare.net/dhiguero/adios-hadoop-hola-spark-t3chfestdhiguero

Apache Spark

- Spark é uma engine rápida, escrita em Scala, para processamento de grandes volumes de dados em um cluster de computadores.

- Desenvolvido inicialmente pela AMPLab em Berkeley. Projeto iniciado em 2009. Os criadores (Matei Zaharia) fundaram Databricks para comercializar (consultoria e suporte) o Spark.

- Open Source Apache Project.
  - Mais de 400 desenvolvedores de 50 empresas diferentes.
  - Committers de mais de 16 organizações (Yahoo, Intel, Databricks, Cloudera, Hortonworks, etc).
  - Projeto Top Level Apache.
  - Um dos projetos mais ativos e com maior crescimento.

Apache Spark

- High Level API. Desenvolvimento mais fácil e rápido. Programadores podem focar na lógica.
  - API para Scala, Java e Python.
  - API para R e dataframes previsto no Spark 1.4 (Junho/Julho de 2015).
- Arquitetura Lambda.
- Baixa Latência – near real time.

```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

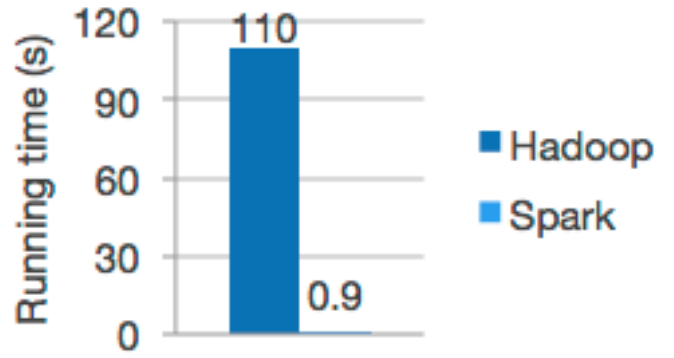
Word count in Spark's Python API

Apache Spark

- Benchmarks com hadoop. 100 vezes mais rápido em memória e 10 vezes mais rápido em disco.

- Spark pode rodar:
  - Local (sem processamento distribuído). Útil para testes.
  - Local com múltiplos processadores (threads).
  - Em um cluster. Gerenciadores: Spark Standalone, Hadoop Yarn, Apache Mesos e EC2.

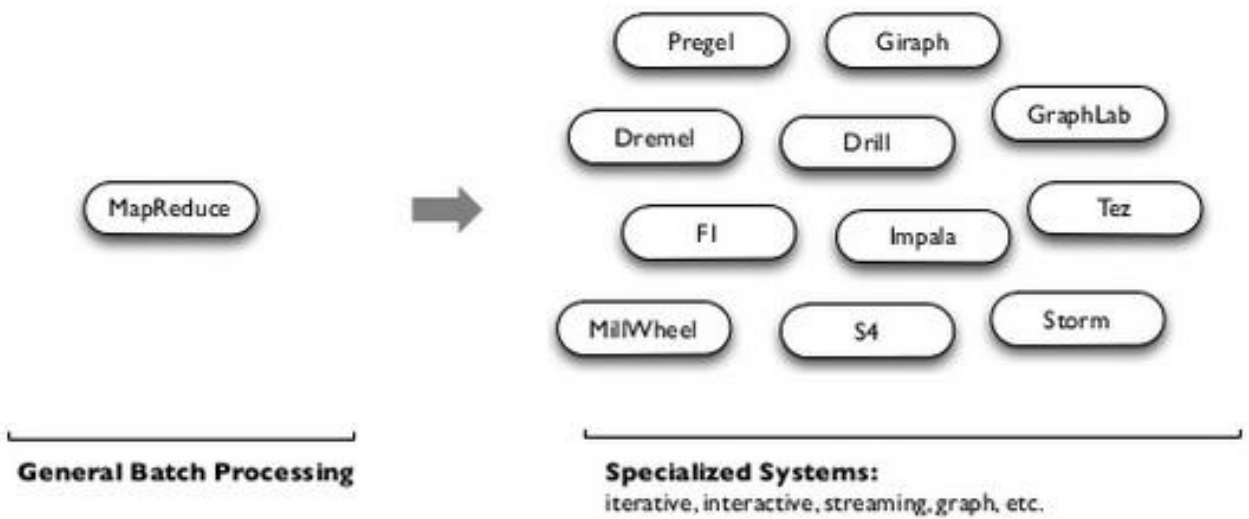
- Acessa dados de diversos lugares:
  - HDFS, Cassandra, Hbase, Hive, Tachyon, etc.



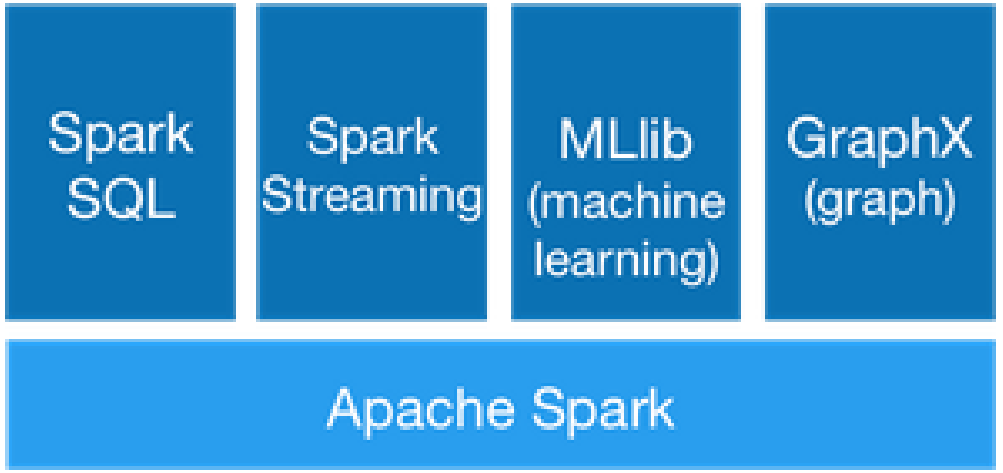
Logistic regression in Hadoop and Spark

Fonte: https://spark.apache.org/

Apache Spark



Map Reduce não é a solução para tudo. Logo, diversos sistemas especializados surgiram.



Hadoop	Spark
Hive	SparkSQL
Apache Mahout	MLLib
Impala	SparkSQL
Apache Giraph	Graphax
Apache Storm	Spark streaming

Resilient Distributed Datasets

- RDDs são a unidade fundamental de dados em Spark. São imutáveis.
  - Resilient: se dados na memória são perdidos, podem ser recriados.
  - Distributed: armazenados na memória por todo o cluster.
  - Datasets: dados iniciais podem vir de um arquivo ou ser criado programaticamente.
- A maioria dos programas em Spark consistem em manipular RDDs.
- RDDs são criados por meio de arquivos, de dados na memória ou de outras RDDs.

Resilient Distributed Datasets

- 2 tipos de operações, de transformação ou de ação.

- Exemplos de operações de transformação.
  - map(function) -> cria um novo RDD processando a função em cada registro do RDD.
  - filter(function) -> cria um novo RDD incluindo ou excluindo cada elemento de acordo com um função booleana.
  - outros: distinct, sample, union, intersection, subtract, cartesian, combineByKey, groupByKey, join, etc.

- Exemplos de operações de ações.
  - count() -> retorna o número de elementos.
  - take(n) -> retorna um array com os primeiros n elementos.
  - collect() -> retorna um array com todos os elementos.
  - saveAsTextFile(file) -> salva o RDD no arquivo.

Resilient Distributed Datasets

- Lazy Evaluation: Nada é processado até uma operação de ação.

File: test.txt

Eu gosto de hadoop.
Eu também gosto de Spark.
Mllib é ótimo, mas
eu acho que precisa de mais algoritmos.

```
>
```

Resilient Distributed Datasets

- Lazy Evaluation: Nada é processado até uma operação de ação.

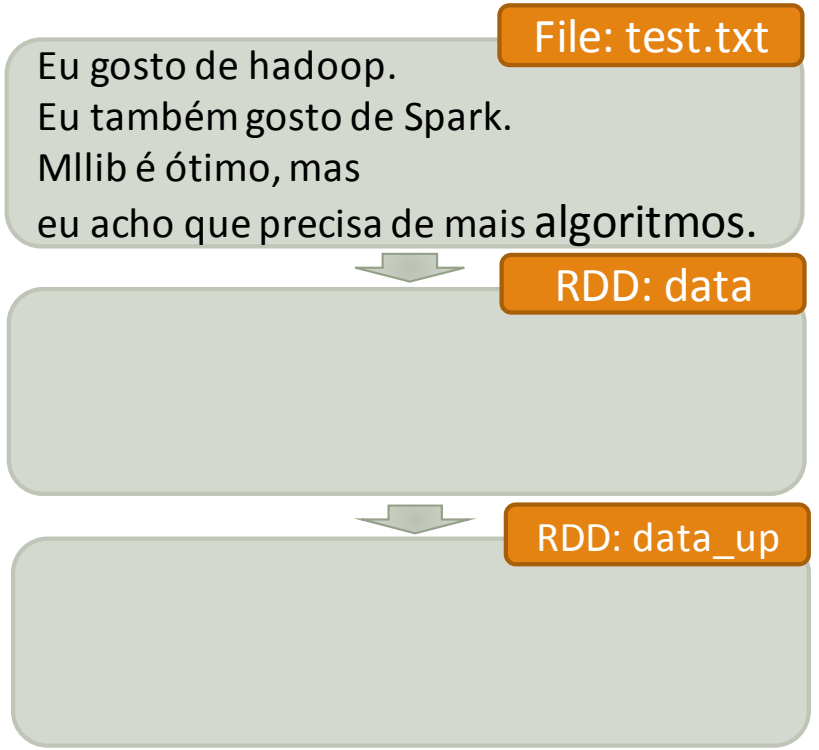
```
> data = sc.textfile("test.txt")
```

Resilient Distributed Datasets

- Lazy Evaluation: Nada é processado até uma operação de ação.

```
> data = sc.textfile("test.txt")

> data_up = data.map(lambda line: line.upper())
```



Resilient Distributed Datasets

- Lazy Evaluation: Nada é processado até uma operação de ação.

```
> data = sc.textfile("test.txt")

> data_up = data.map(lambda line: line.upper())

> data_filtr = data_up.filter(lambda line: line.startswith("E"))
```

Resilient Distributed Datasets

- Lazy Evaluation: Nada é processado até uma operação de ação.

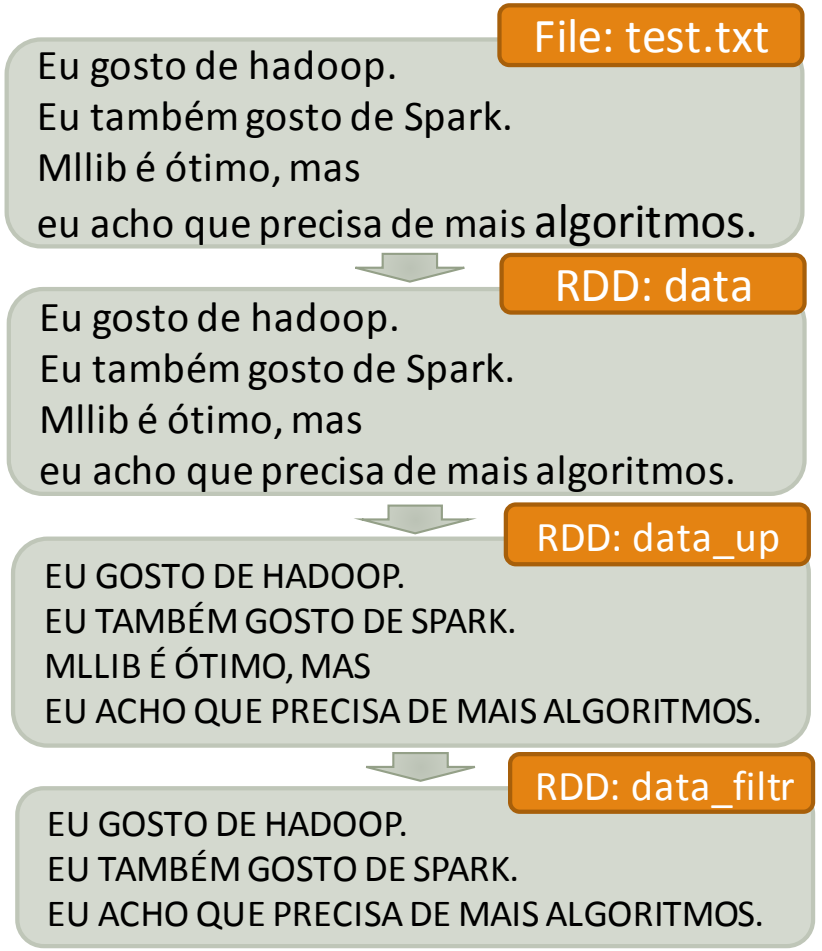
```
> data = sc.textfile("test.txt")

> data_up = data.map(lambda line: line.upper())

> data_filtr = data_up.filter(lambda line: line.startswith("E"))

> data_filtr.count()

 3
```



Resilient Distributed Datasets

- Caching.

```
>
```

File: test.txt

Eu gosto de hadoop.
Eu também gosto de Spark.
Mllib é ótimo, mas
eu acho que precisa de mais algoritmos.

Resilient Distributed Datasets

- Caching.

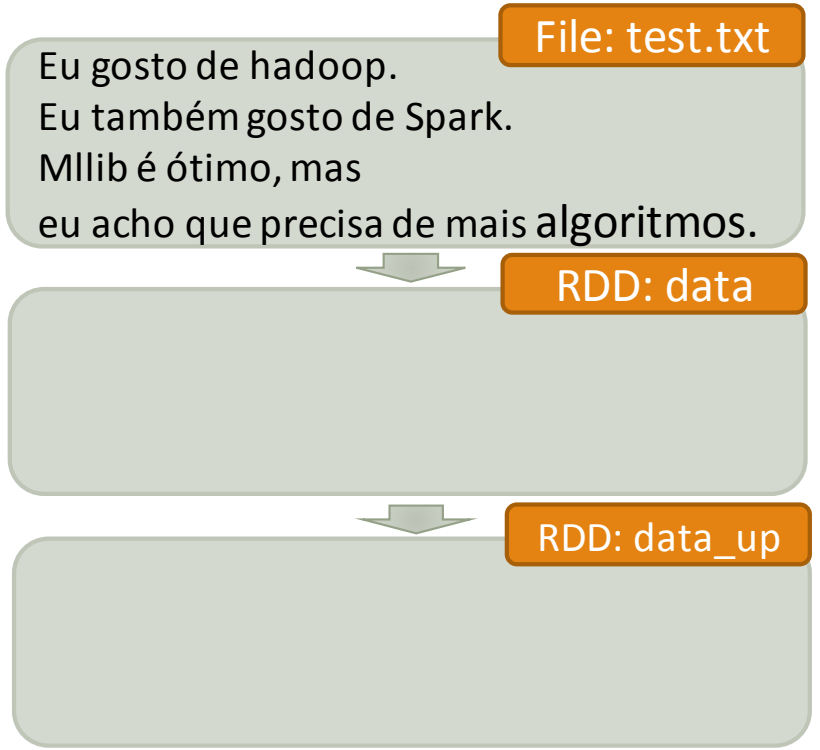
```
> data = sc.textfile("test.txt")
```

Resilient Distributed Datasets

- Caching.

```
> data = sc.textfile(“test.txt”)

> data_up = data.map(lambda line: line.upper())
```



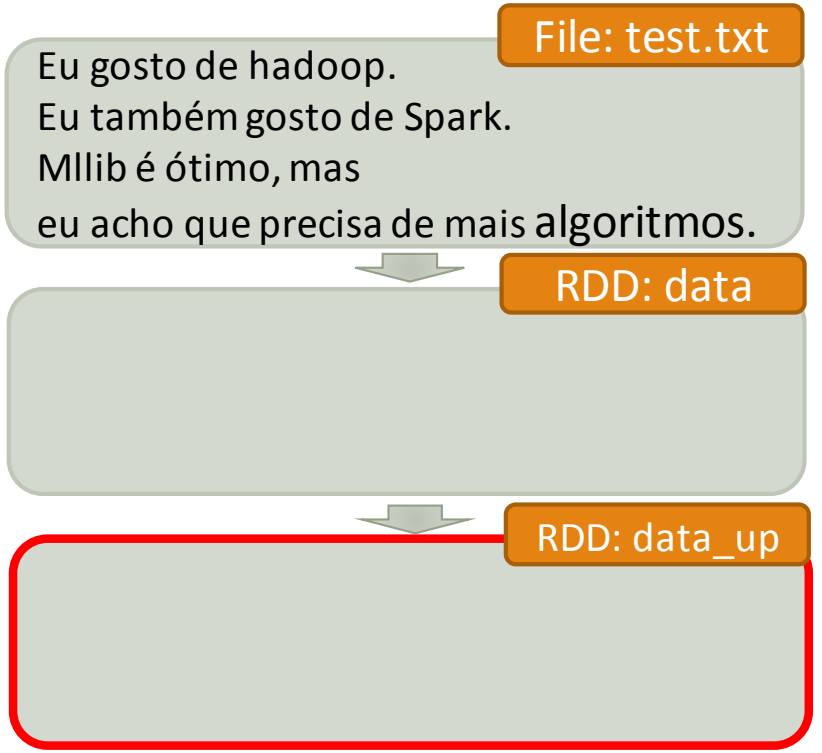
Resilient Distributed Datasets

- Caching.

```
> data = sc.textfile(“test.txt”)

> data_up = data.map(lambda line: line.upper())

> data_up.cache
```



Resilient Distributed Datasets

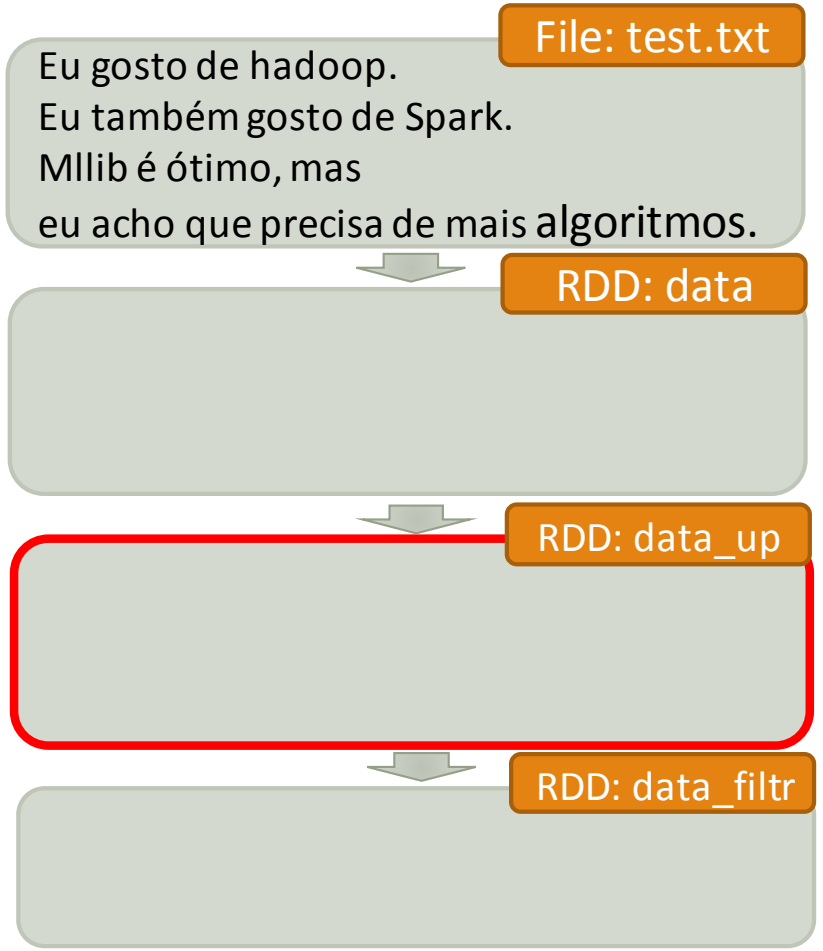
- Caching.

```
> data = sc.textfile("test.txt")

> data_up = data.map(lambda line: line.upper())

> data_up.cache

> data_filtr = data_up.filter(lambda line: line.startswith("E"))
```



Resilient Distributed Datasets

- Caching.

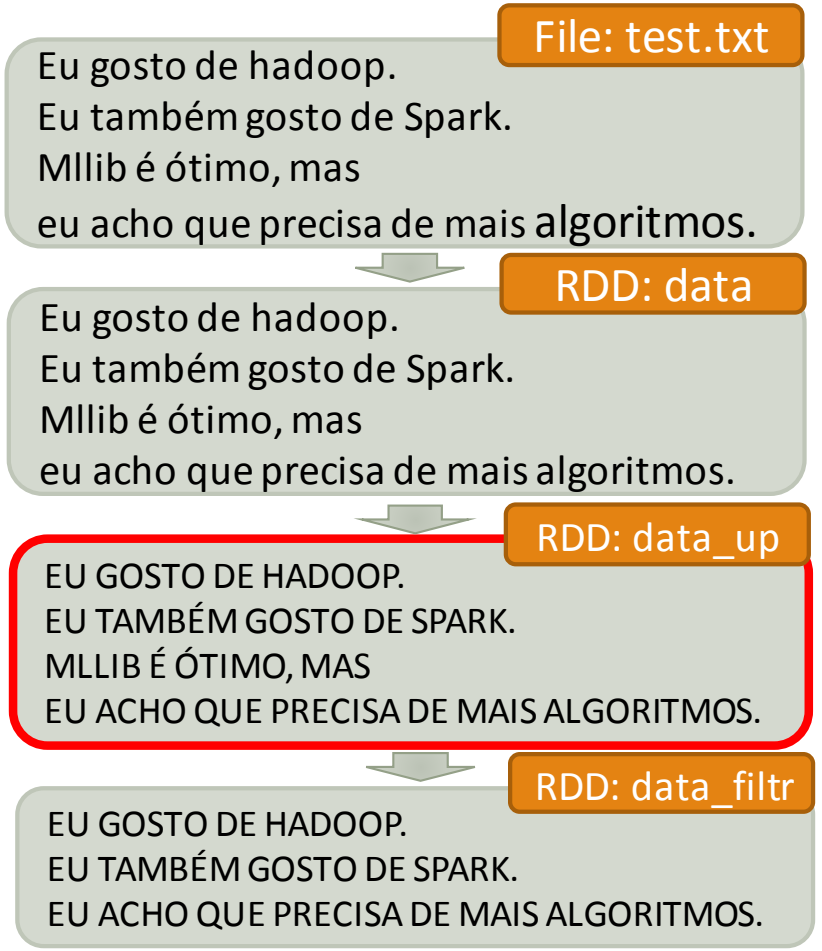
```
> data = sc.textfile("test.txt")

> data_up = data.map(lambda line: line.upper())

> data_up.cache

> data_filtr = data_up.filter(lambda line: line.startswith("E"))

> data_filtr.count()
 3
```



Resilient Distributed Datasets

- Caching.

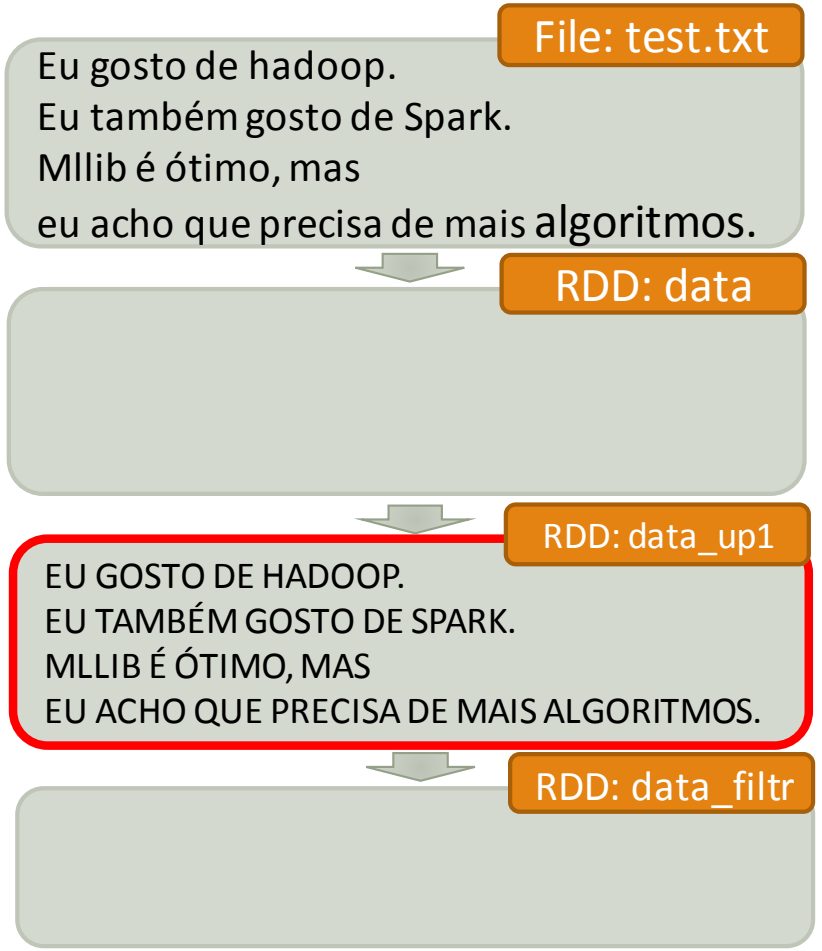
```
> data = sc.textfile("test.txt")

> data_up1 = data.map(lambda line: line.upper())

> data_up1.cache

> data_filtr = data_up1.filter(lambda line: line.startswith("E"))

> data_filtr.count()
 3
> data_filtr.count()
```



Resilient Distributed Datasets

- Caching.

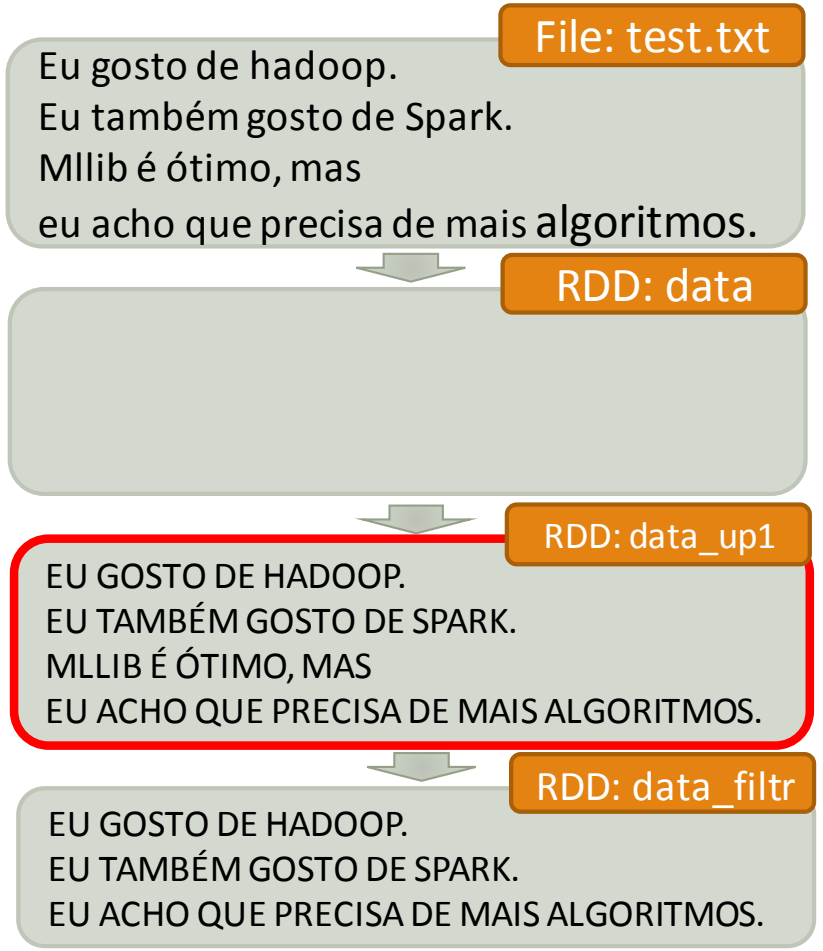
```
> data = sc.textfile("test.txt")

> data_up1 = data.map(lambda line: line.upper())

> data_up1.cache

> data_filtr = data_up1.filter(lambda line: line.startswith("E"))

> data_filtr.count()
 3
> data_filtr.count()
 3
```



Resilient Distributed Datasets

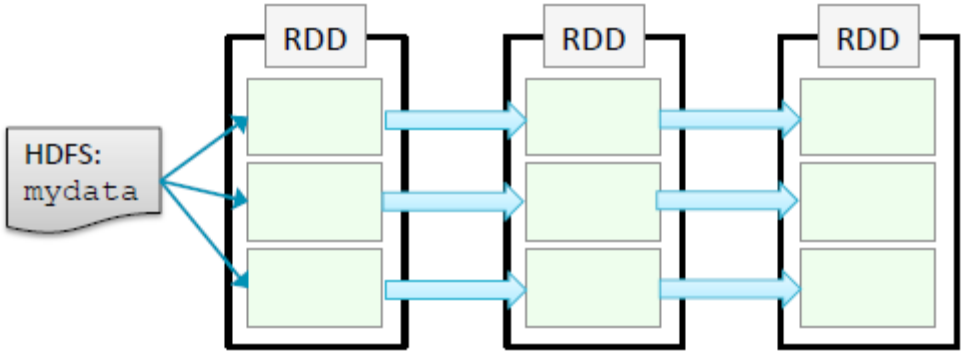
- Níveis de persistência.

Level	Space Used	CPU time	In memory	On Disk	Nodes with data	Comments
MEMORY_ONLY	High	Low	Y	N	1	
MEMORY_ONLY_2	High	Low	Y	N	2	
MEMORY_ONLY_SER	Low	High	Y	N	1	
MEMORY_ONLY_SER_2	Low	High	Y	N	2	
MEMORY_AND_DISK	High	Medium	Some	Some	1	Spills to disk if there is too much data to fit in memory.
MEMORY_AND_DISK_2	High	Medium	Some	Some	2	Spills to disk if there is too much data to fit in memory.
MEMORY_AND_DISK_SER	Low	High	Some	Some	1	Spills to disk if there is too much data to fit in memory.
MEMORY_AND_DISK_SER_2	Low	High	Some	Some	2	Spills to disk if there is too much data to fit in memory.
DISK_ONLY	Low	High	N	Y	1	
DISK_ONLY_2	Low	High	N	Y	2	

Applications, Jobs, Stages e Tasks

Dados estão particionados em diferentes nós. Quando possível, as tasks são executadas nos nós onde os dados estão na memória.

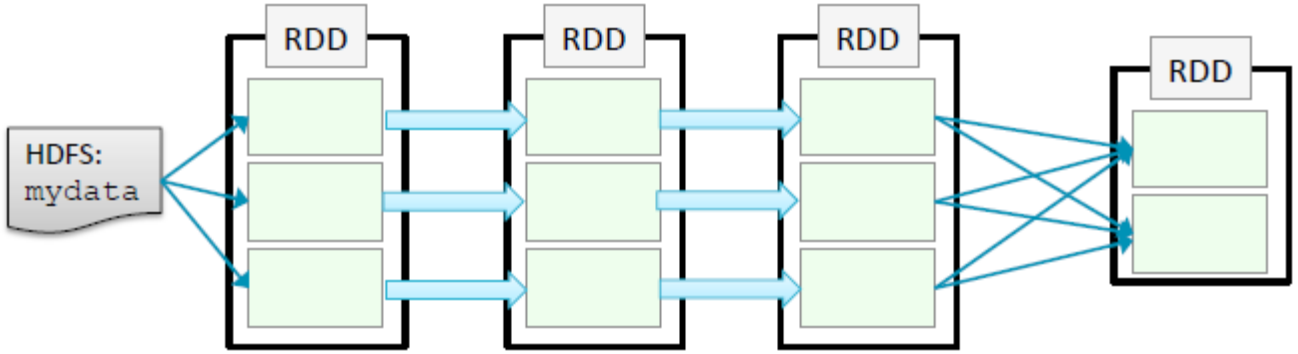
```
> media_palavra = sc.textfile("/user/hive/warehouse/mydata/") \
    .flatMap(lambda line: line.split()) \
    .map(lambda word: (word[0], len(word)))
```



Applications, Jobs, Stages e Tasks

Dados estão particionados em diferentes nós. Quando possível, as tasks são executadas nos nós onde os dados estão na memória.

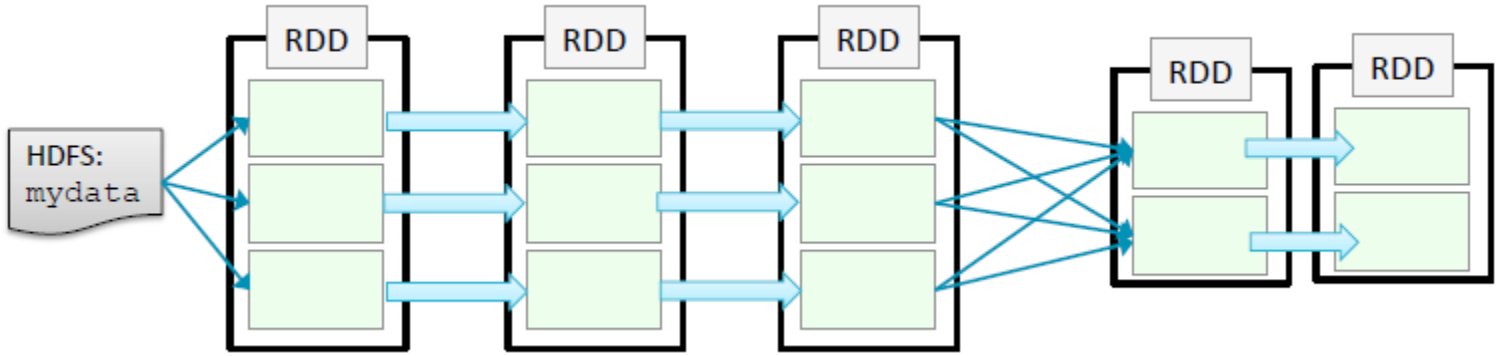
```
> media_palavra = sc.textfile("/user/hive/warehouse/mydata/") \
    .flatMap(lambda line: line.split()) \
    .map(lambda word: (word[0], len(word))) \
    .groupByKey()
```



Applications, Jobs, Stages e Tasks

Dados estão particionados em diferentes nós. Quando possível, as tasks são executadas nos nós onde os dados estão na memória.

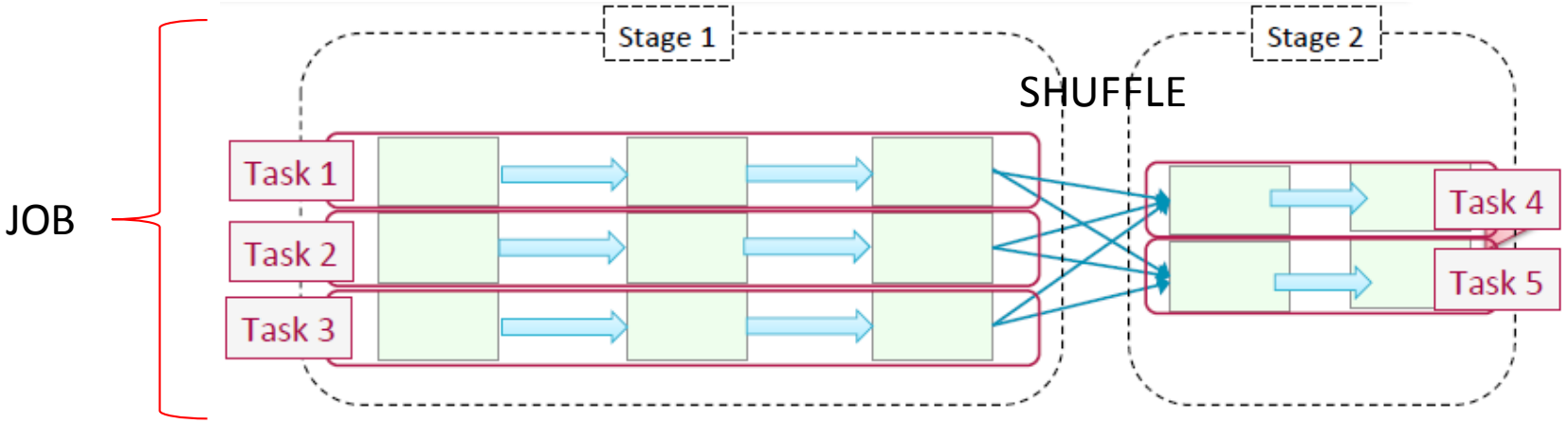
```
> media_palavra = sc.textfile("/user/hive/warehouse/mydata/") \
    .flatMap(lambda line: line.split()) \
    .map(lambda word: (word[0], len(word))) \
    .groupByKey() \
    .map(lambda (k, values): (k, sum(values)/len(values)))
```



Applications, Jobs, Stages e Tasks

Dados estão particionados em diferentes nós. Quando possível, as tasks são executadas nos nós onde os dados estão na memória.

```
> media_palavra = sc.textfile("/user/hive/warehouse/mydata/") \
    .flatMap(lambda line: line.split()) \
    .map(lambda word: (word[0], len(word))) \
    .groupByKey() \
    .map(lambda (k, values): (k, sum(values)/len(values)))
```



Como o Spark calcula as Stages?

- Spark constrói uma grafo acíclico direcionado (DAG) das dependências entre as RDDs.

- Operações Narrow
  - Apenas um filho depende do RDD.
  - Shuffle não é necessário entre os nós.
  - Colapsada em um estágio único.
  - Ex: map, filter, union.

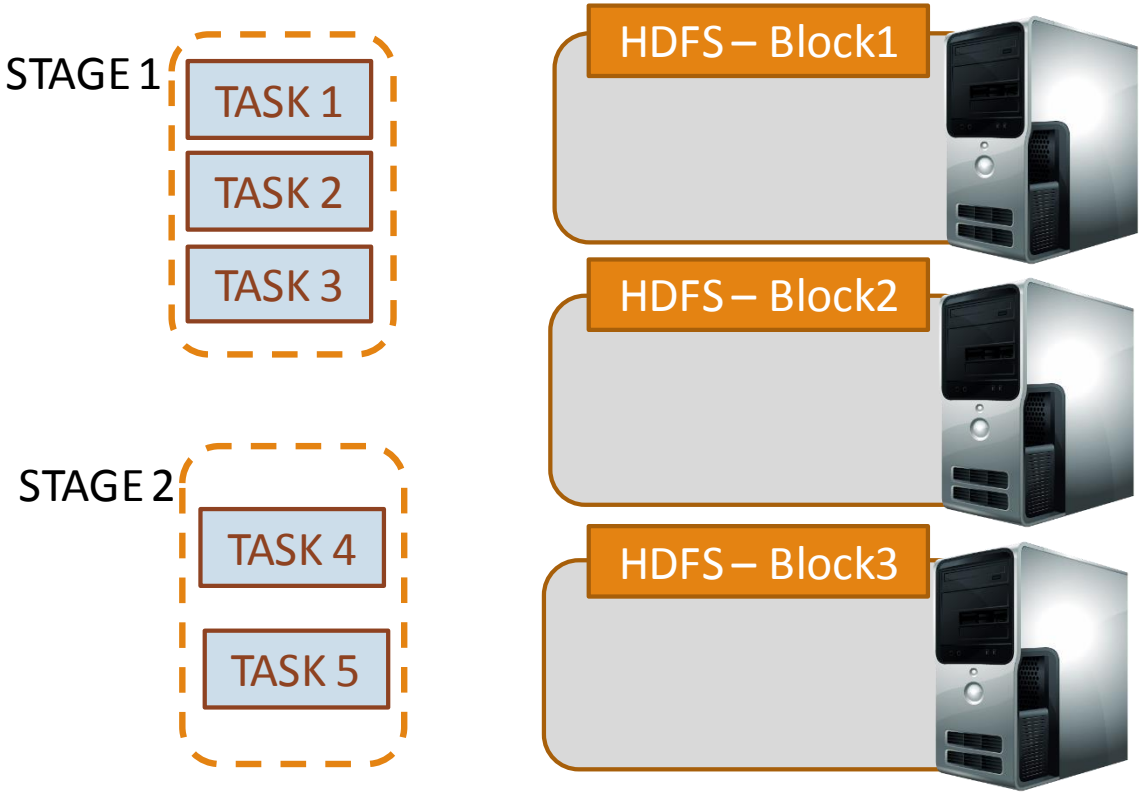
- Operações Wide
  - Múltiplos filhos dependem do mesmo RDD.
  - Define um novo stage.
  - Ex: reduceByKey, join,groupByKey

Como o Spark calcula as Stages?

- Spark constrói uma grafo acíclico direcionado (DAG) das dependências entre as RDDs.

- Operações Narrow
  - Apenas um filho depende do RDD.
  - Shuffle não é necessário entre os nós.
  - Colapsada em um estágio único.
  - Ex: map, filter, union.

- Operações Wide
  - Múltiplos filhos dependem do mesmo RDD.
  - Define um novo stage.
  - Ex: reduceByKey, join,groupByKey

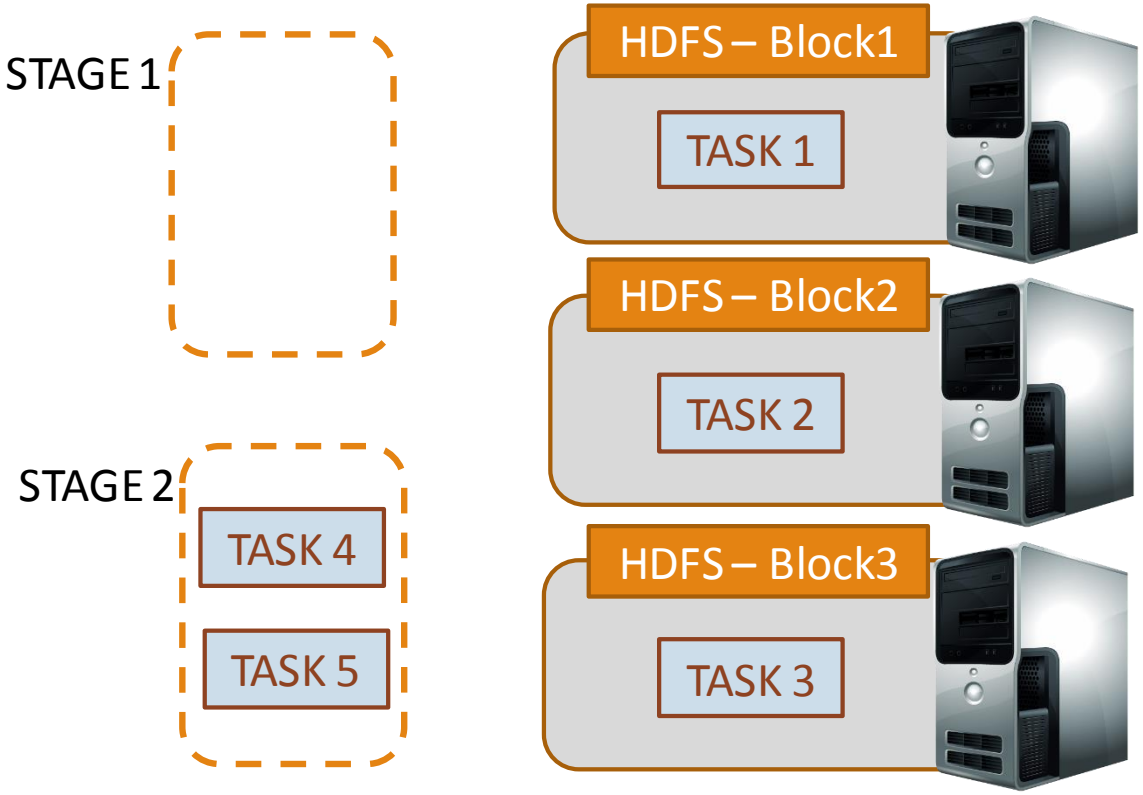


Como o Spark calcula as Stages?

- Spark constrói uma grafo acíclico direcionado (DAG) das dependências entre as RDDs.

- Operações Narrow
  - Apenas um filho depende do RDD.
  - Shuffle não é necessário entre os nós.
  - Colapsada em um estágio único.
  - Ex: map, filter, union.

- Operações Wide
  - Múltiplos filhos dependem do mesmo RDD.
  - Define um novo stage.
  - Ex: reduceByKey, join,groupByKey

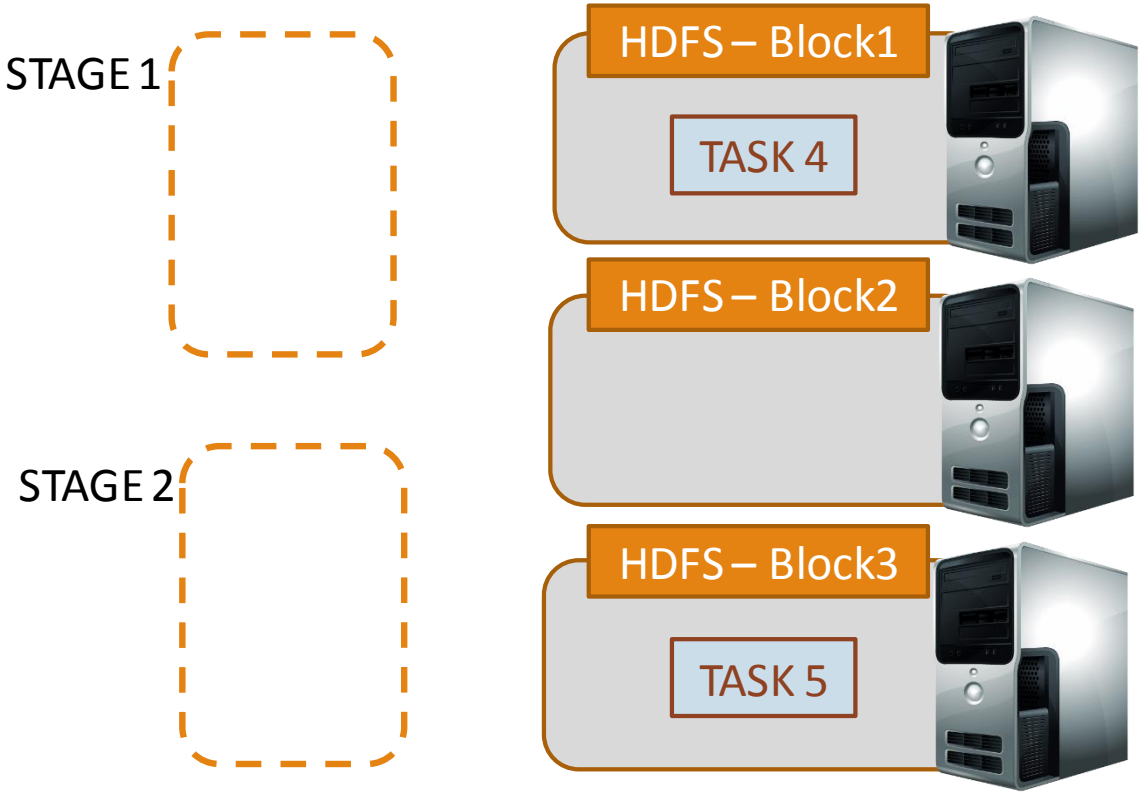


Como o Spark calcula as Stages?

- Spark constrói uma grafo acíclico direcionado (DAG) das dependências entre as RDDs.

- Operações Narrow
  - Apenas um filho depende do RDD.
  - Shuffle não é necessário entre os nós.
  - Colapsada em um estágio único.
  - Ex: map, filter, union.

- Operações Wide
  - Múltiplos filhos dependem do mesmo RDD.
  - Define um novo stage.
  - Ex: reduceByKey, join,groupByKey



Problemas com Spark

- Imaturidade (Bugs).
- Ainda não escala tão bem quanto Hadoop.
- Tempo maior para recuperação de falhas em relação ao Hadoop.
- Alto consumo de memória.
- Configuração e otimização mais difíceis.

Bibliografia

- Zaharia, Matei. An Architecture for Fast and General Data Processing on Large Clusters, EECS Department, University of California, Berkeley, 2014.
- Documentação do Apache Spark: https://spark.apache.org/documentation.html
- Tom White. Hadoop: The Definitive Guide. Yahoo Press.
- Chuck Lam. Hadoop in Action. Manning Publications
- Introduction to Spark Developer Training, http://pt.slideshare.net/cloudera/spark-devwebinarslides-final
- Introduction to Spark Training, http://pt.slideshare.net/datamantra/introduction-to-apache-spark-45062010?qid=36fd8e78-6b2b-4b5f-9c5b-e3ac1bfa87e4&v=qf1&b=&from_search=3