Curso 03: Administração de Dados Complexos em Larga Escala

-- Introdução ao Apache Spark --

Prof. Jose Fernando Rodrigues Junior

Objetivo: apresentar a solução Big Data Apache Spark

Níveis da Análise de Dados

- **Análise de dados básica**: contagens, somas, médias, máximo, mínimo, e ordenação;
- **Análise de dados estatística**: distribuição de dados, ajuste de modelo, teste de hipóteses, métricas, etc;
- **Análise de dados avançada**: aprendizado de máquina, classificação, regres o, clusterização, etc;
- **Aprendizado de máquina avançado**: arquiteturas de redes neurais visando inteligência artificial.

Business Intelligence - Tools

-DW: fontes de dados, na maioria das vezes heterogêneas organizadas em datamarts ou data warehouses - Apache Hive, Oracle Exadata, IBM Netezza, Microsoft Azure, e outras;

-DW/OLAP: software capaz de sumarizar grandes quantidades de dados de maneira rápida;

-Visualização: software para exibir os produtos finais do processamento de dados; Tableau, MS PowerBI, IBM Cognos, e outros;

***-Advanced Analytics*:**

+Estatística - R, Matlab, Saas;

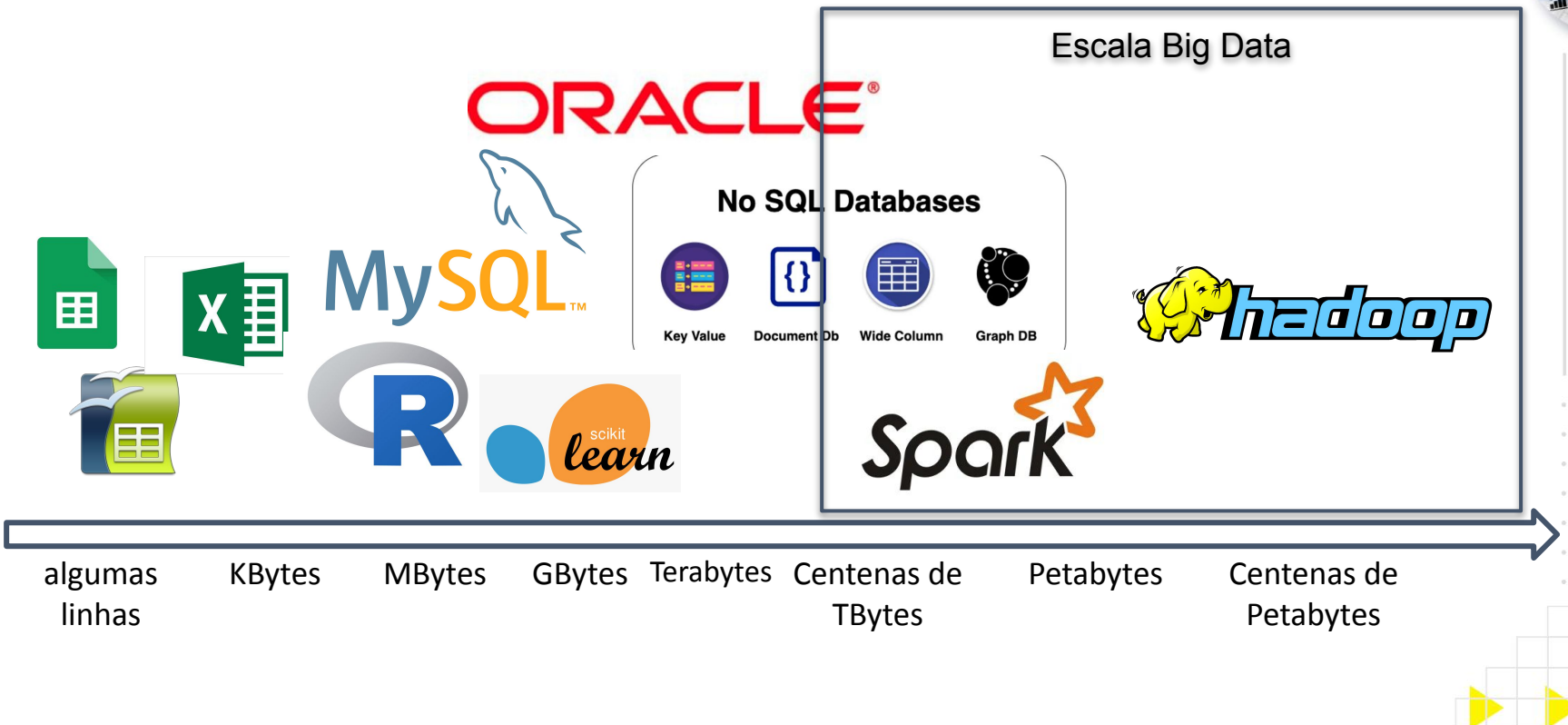
+**Aprendizado de Máquina - Scikit Learn, Saas, Scilab**;

+**Inteligência Artificial** - em amadurecimento.

Contextualização

algumas linhas | KBytes | MBytes | GBytes | Terabytes | Centenas de TBytes | Petabytes | Centenas de Petabytes

Contextualização



Spark

- Surgiu na academia, na **Universidade de Berkeley**, em 2009;

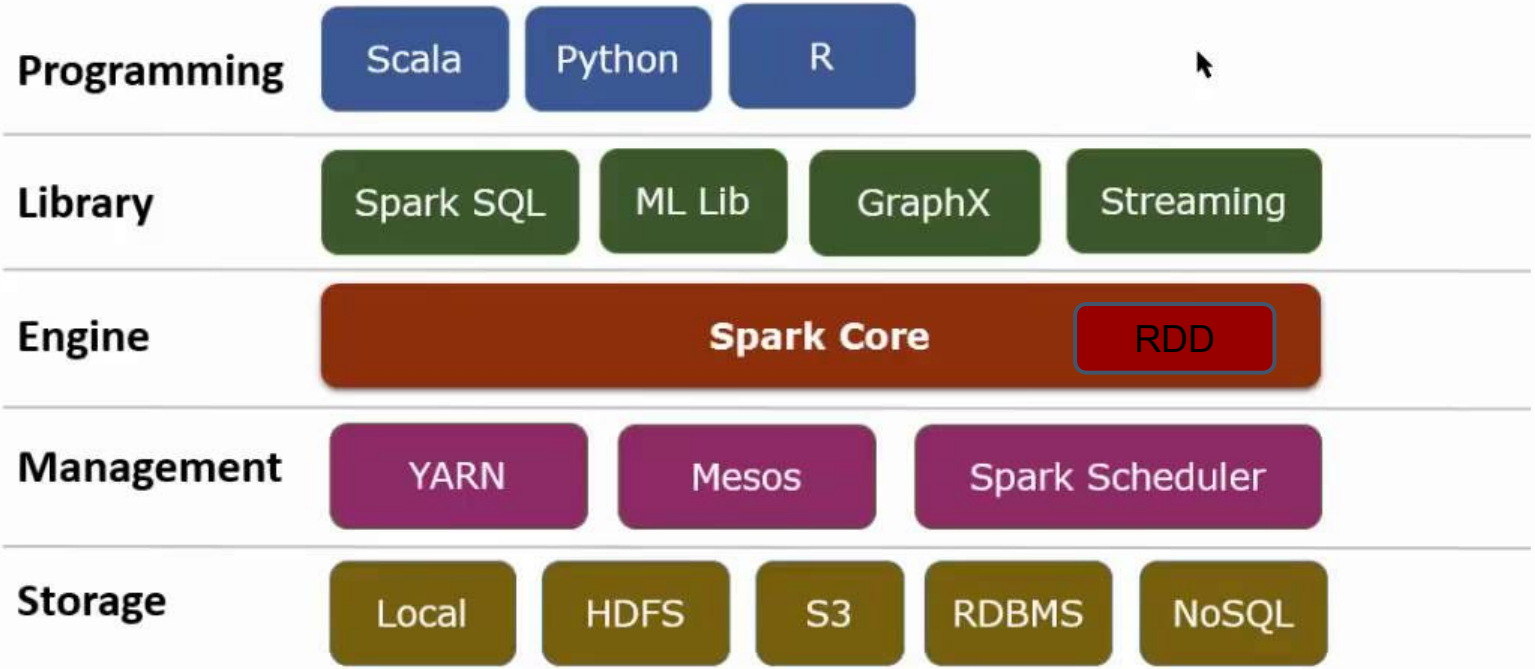
- Em 2013, tornou-se um projeto **Apache;**

- Atualmente, um **arcabouço para processamento** em real time e, também, para batch em escala Big Data.

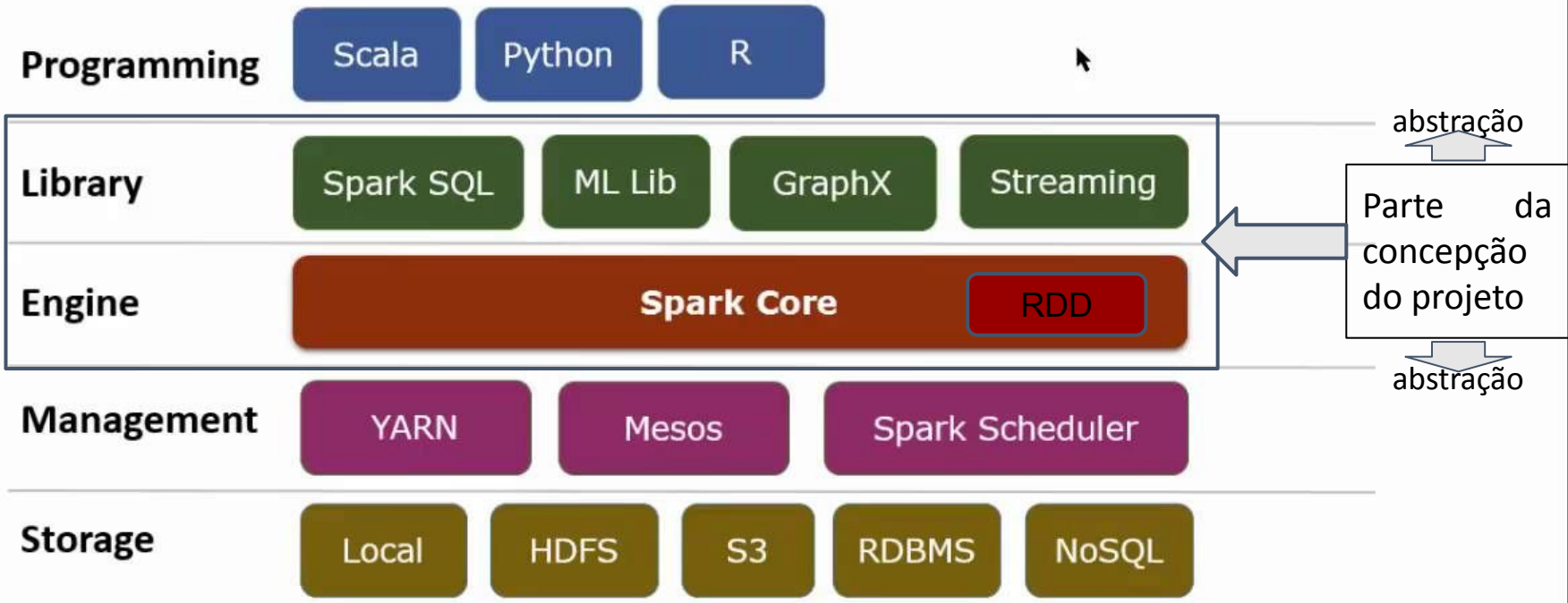
Real time x Batch

Real time	Batch
O processamento é iniciado **imediatamente** quando da chegada de novos dados, ou comandos	**Dados pré-selecionados** e carregados usando *scripts* em linha de comando
A execução precisa ser concluída dentro de **restrições estritas de tempo**	Um conjunto, usualmente grande, de dados/transações é processado em uma **única execução**
	Execução completa **sem nenhuma intervenção do usuário**
	Usado para **múltiplas operações sequenciais** em larga escala, geração de relatórios, e fluxo de dados offline
Ex.: detecção de transações fraudulentas	Ex.: relatórios gerenciais

Spark Framework



Spark Framework



Spark Resilient Distributed Dataset (RDD)

- Conceito básico para:
  - input e output;
  - **execução distribuída e paralela de modo abstrato**;
  - transformação de dados como map, join, reduce, e filter;
  - processamento **in-memory**.
- Processamento in-memory:
  - dados em memória (128 MB por padrão), **reduzindo a dependência de otimizações** como índices, pré-agregações, SGBDs, esquemas estrela e cubos; modelo de processamento mais simples;
  - **compactação de dados**, aumentando a capacidade da memória;
  - acesso a dados 10.000 a 1.000.000 de vezes **mais rápida** do que via acesso a disco;
  - aderência a ferramentas de **visualização**.

Spark Resilient Distributed Dataset (RDD)

- Por que Resilient (resiliente)?
  - No modelo de processamento Spark, os dados de entrada são particionados e carregados **em memória** nos nós de processamento;
  - Dados em memória são **voláteis** - podem se perder;
  - O nome Resilient vem do fato de que os dados são **replicados em diversas máquinas** segundo o parâmetro inteiro "*Replication Factor*", de modo que há reduzidas chances de perda de dados.
  - Além disso, os RDDs são **imutáveis**, o que permite uma replicação simplificada - operações que gerem novos dados, resultam em novos RDDs.

RDD: abstração de dados distribuída

Considere um arquivo com 384 MB, "Dados.txt", e o comando:

```
RDD arquivo = sc.textFile("Dados.txt")/*código Scala*/
```

⇒ Automaticamente:

- o arquivo será **particionado** em 3 RDDs de 128 MB cada;
- cada RDD será **distribuído** e carregado em memória;
- haverá *Replication Factor* **cópias** dos RDDs.

Considere o comando:

```
RDD contagemDePalavras = arquivo.map(....lógica para contar palavras…)
```

⇒ A operação de map irá:

- criar 3 **novos RDDs**, cada um contendo o resultado da lógica do map para um dos 3 RDDs originais;
- haverá *Replication Factor* **cópias** dos RDDs.

RDD: abstração de dados distribuída

Isto é, o RDD:

- **abstrai** o particionamento e a distribuição dos dados;
- **gerencia** a replicação dos dados;
- permite **operações** sobre os RDDs sem que o usuário precise saber das partições e das réplicas;

RDD: abstração de dados distribuída

O poder da abstração RDD:

- Se eu tenho um arquivo "meus_dados.txt" com **10 Gigabytes**, é possível distribuir este arquivo em um cluster computacional com o comando:

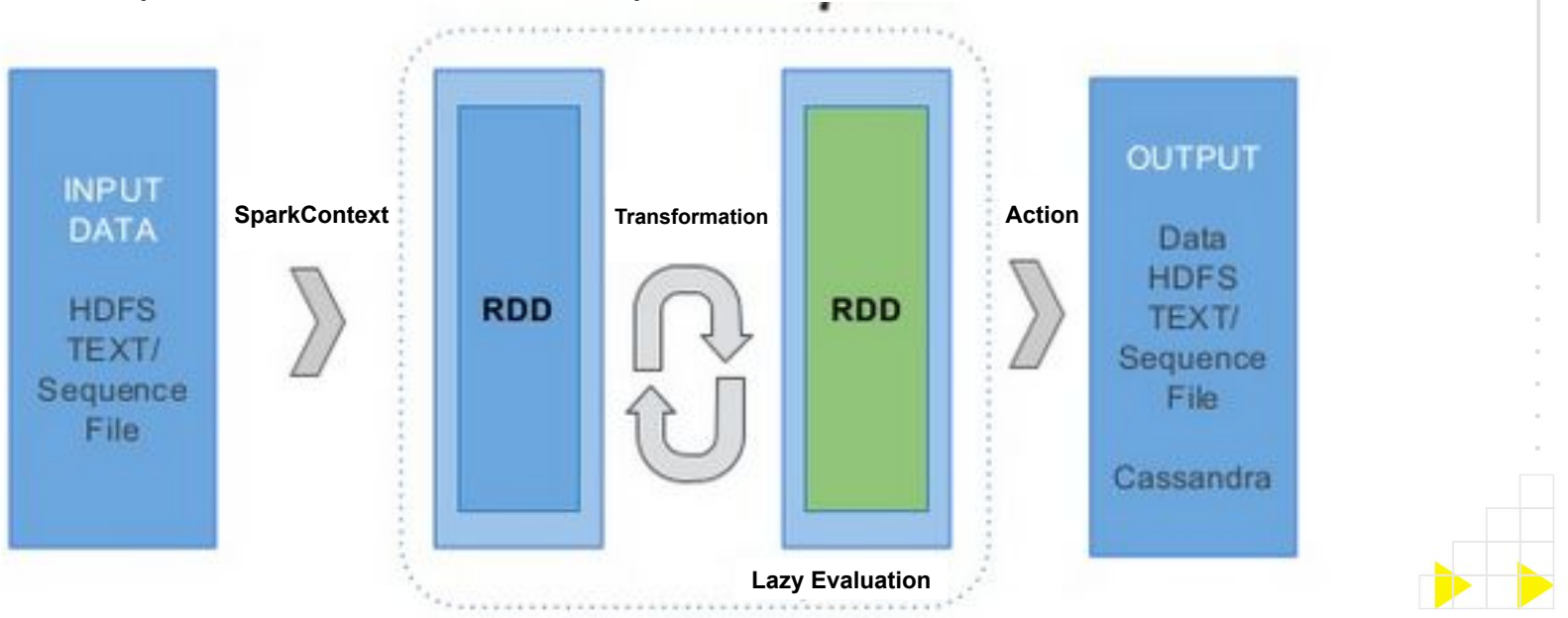
```
RDD arquivo = sc.textFile("meus_dados.txt")
```

- Se eu tenho um arquivo com **10 Terabytes**, é possível distribuir este arquivo em um cluster computacional com o comando:

```
RDD arquivo = sc.textFile("meus_dados.txt")
```

Operações com RDDs

- **Transformation**: a partir de um RDD, executa um processamento, e cria um novo RDD;
- **Action**: a partir de um RDD, computam um resultado.



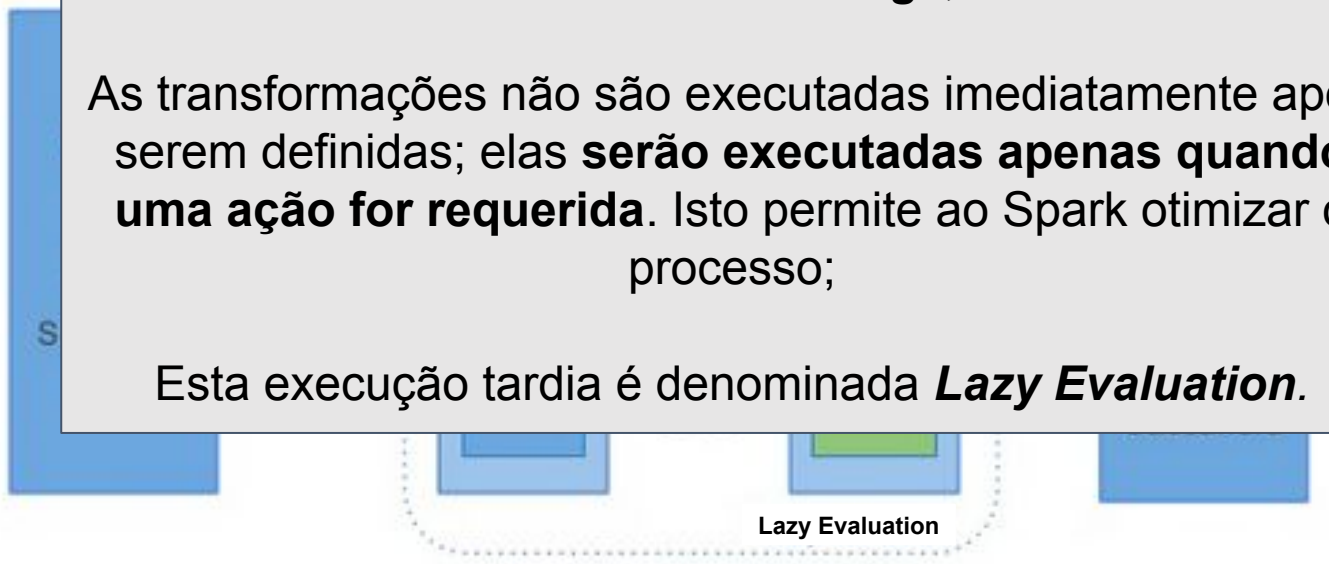
Operações com RDDs

- **Transformation**: a partir de um RDD, executa um processamento, e cria um novo RDD;
- **Action**

Uma sequência de transformações define um **grafo acíclico** denominado ***lineage;***

As transformações não são executadas imediatamente após serem definidas; elas **serão executadas apenas quando uma ação for requerida**. Isto permite ao Spark otimizar o processo;

Esta execução tardia é denominada ***Lazy Evaluation.***



Operações com RDDs

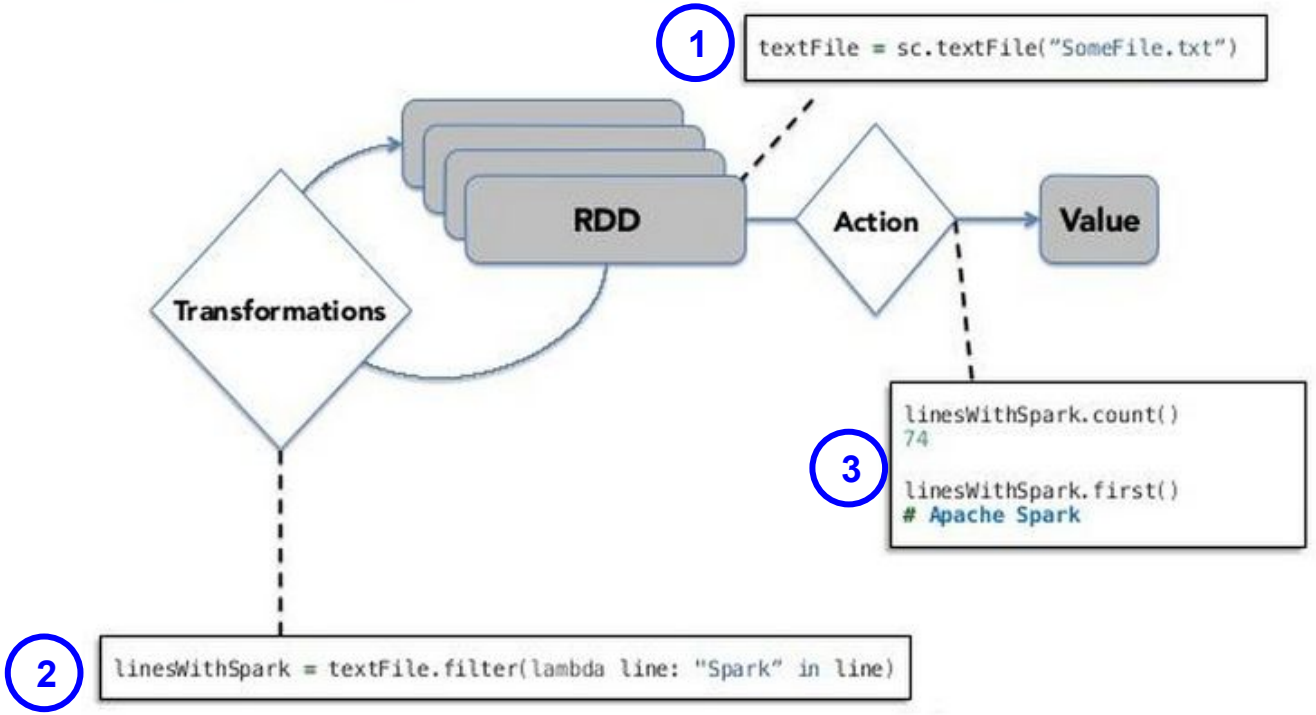
- **Transformation**: a partir de um conjunto de RDDs, executa um processamento, e cria um novo conjunto de RDDs;
- **Action**: a partir de um conjunto de RDDs, computam um resultado.

Transformations	Actions
map(func)	take(N)
flatMap(func)	count()
filter(func)	collect()
groupByKey()	reduce(func)
reduceByKey(func)	takeOrdered(N)
mapValues(func)	top(N)
…	…

⇒ Referência: [Transformations and actions]

Processamento simples

- **Problema:** contar quantas linhas contém a palavra "Spark"



Spark API

- O RDD é a **abstração distribuída de dados fundamental**
- A API Spark **evoluiu** ao longo do tempo para o Spark 2, e o Spark 3
- Novas abstrações de dados, mais elaboradas: **DataFrames e DataSets**

⇒ Para saber mais: RDDs vs DataFrames and Datasets

Problemas paralelizáveis

⇒ **Nem todos os problemas** podem ser resolvidos com processamento paralelo distribuído; apenas a classe de problemas definida como *Embarrassingly Parallel*. Alguns exemplos:

- consultas SQL
- integração numérica
- renderização gráfica
- reconhecimento facial
- algoritmos genéticos
- transformada de Fourier
- redes neurais artificiais
- entre outros

Problemas paralelizáveis

- Problemas considerados não paralelizáveis são resolvidos por meio de **algoritmos de alta complexidade**;
- São admitidos **resultados aproximados**, obtidos por processamento iterativo, por exemplo;
- O **Processamento Paralelo** é uma área de pesquisa atuante.

Hands on

- Docker para rodar PySpark em um Jupyter notebook

1) Instalar o Docker - https://docs.docker.com/get-docker/

2) Instalar/executar o container (Jupyter Notebook Python, Spark Stack)

```
docker run -p 8888:8888 jupyter/pyspark-notebook
```

3) Abrir o link http://127.0.0.1:8888/?token=... -> New notebook

- Passo a passo completo de um 1o. programa Spark em Python (PySpark)

⇒ First Steps With PySpark and Big Data Processing

Hands on

- Comparar o contador de palavras em MapReduce visto na Quinzena 01, Aula 03 com o contador de palavras Spark descrito no arquivo:

⇒ `Curso03-Quinzena03-Aula04-PrimeiroSpark.ipynb`