Processamento de Dados em Larga Escala

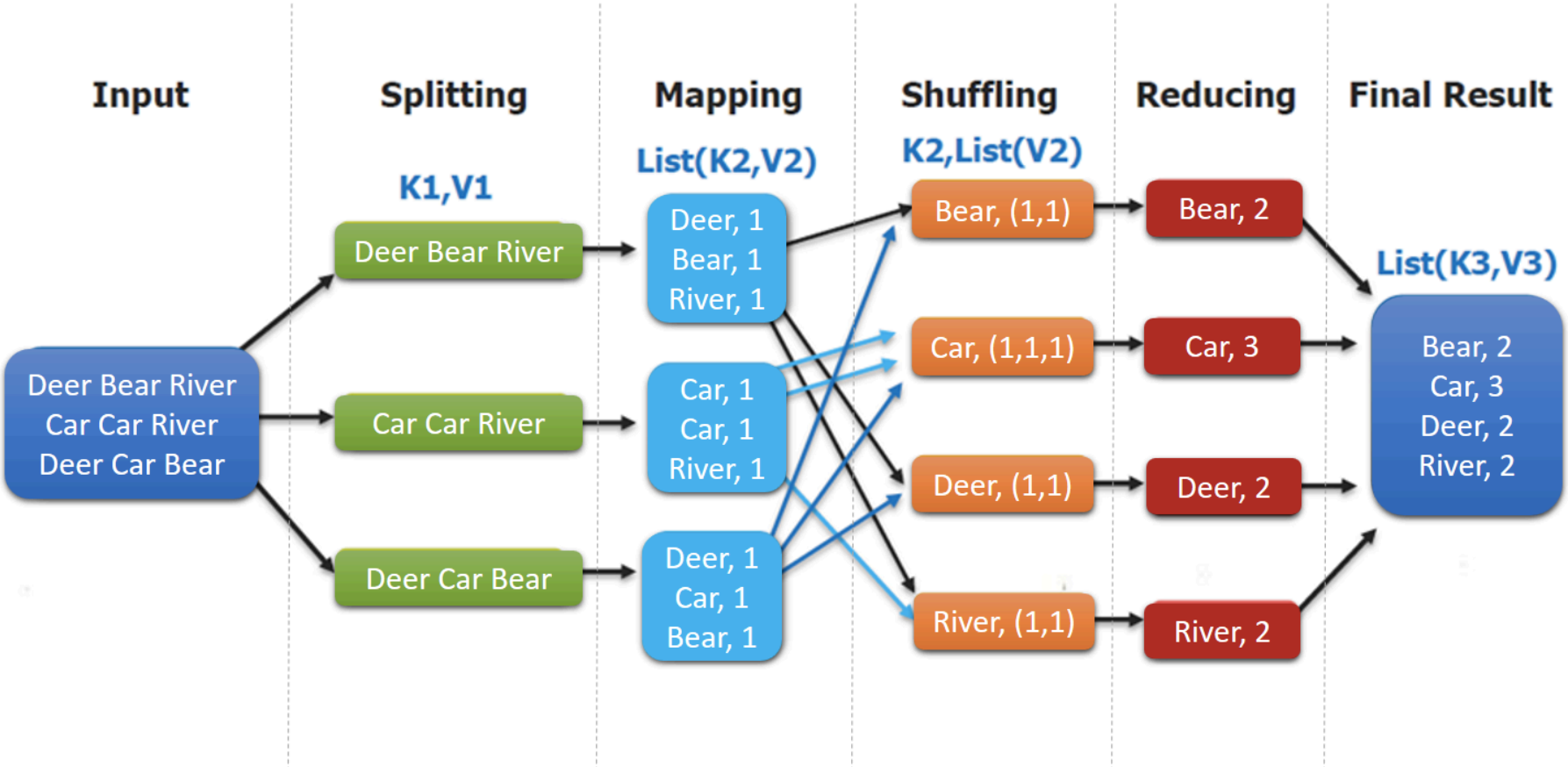
Luciano Barbosa

Parte do material adaptado dos slides de
Harold Liu, Matei Zaharia, Min-Yuh Day e Altigran Silva

Roteiro

- Map Reduce (Hadoop)
- Spark

Map Reduce



MapReduce Word Count Process

Map

- Aplica uma função a todos elementos de uma lista
- Ex: Computar a soma dos quadrados de uma lista
  - Lista L = [0,1,2,3]
  - Computar o quadrado de cada item
  - Saída: L = [0,1,4,9]

```
## For Loop
O=[]
for i in L:
    O.append(i*i)

## List Comprehension
[i*i for i in L]
```

Tradicional

```
map(lambda x:x*x, L)
```

Map-Reduce

Reduce

- Realiza a computação em uma lista e retorna um resultado
- Ex: Computar a soma dos quadrados de uma lista
  - Lista L = [0,1,2,3]
  - Calcular a soma
  - Saída: 16

```
## Use Builtin
sum(L)

## for loop
s=0
for i in L:
    s+=i
```

Tradicional

```
reduce(lambda (x,y): x+y, L)
```

Map-Reduce

Exemplo em Python

```
## For Loop
s=0
for i in L:
    s+= i*i
## List comprehension
sum([i*i for i in L])
```

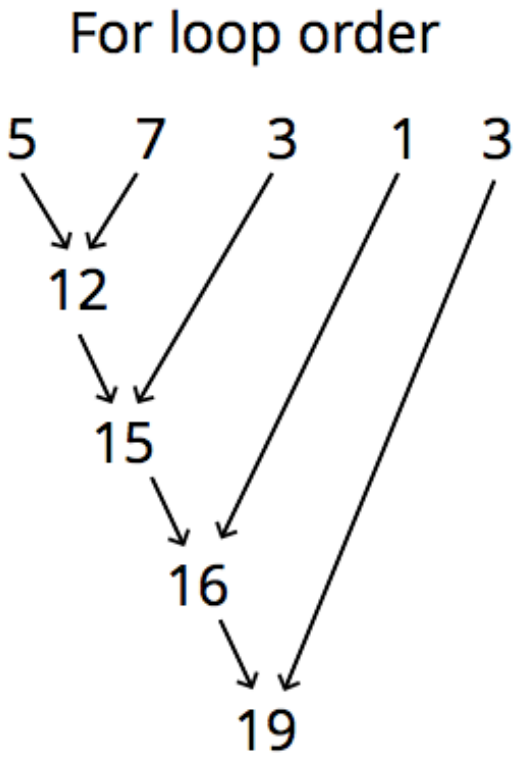
Tradicional

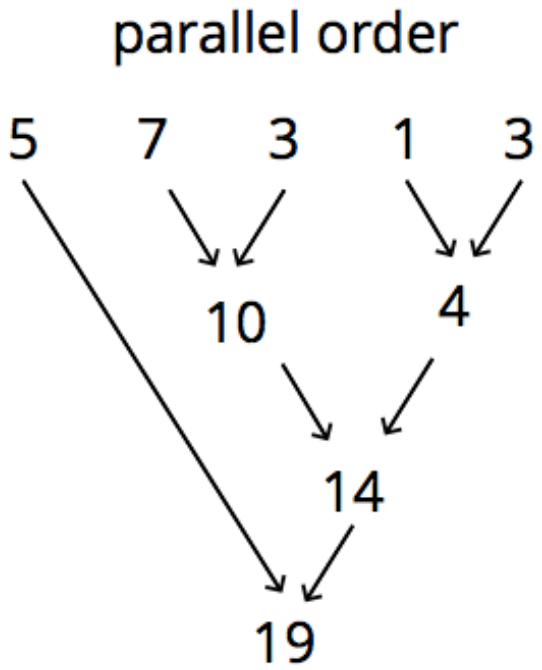
```
reduce(lambda x,y:x+y, \\
        map(lambda i:i*i,L))
```

Map-Reduce

Independência de Ordem

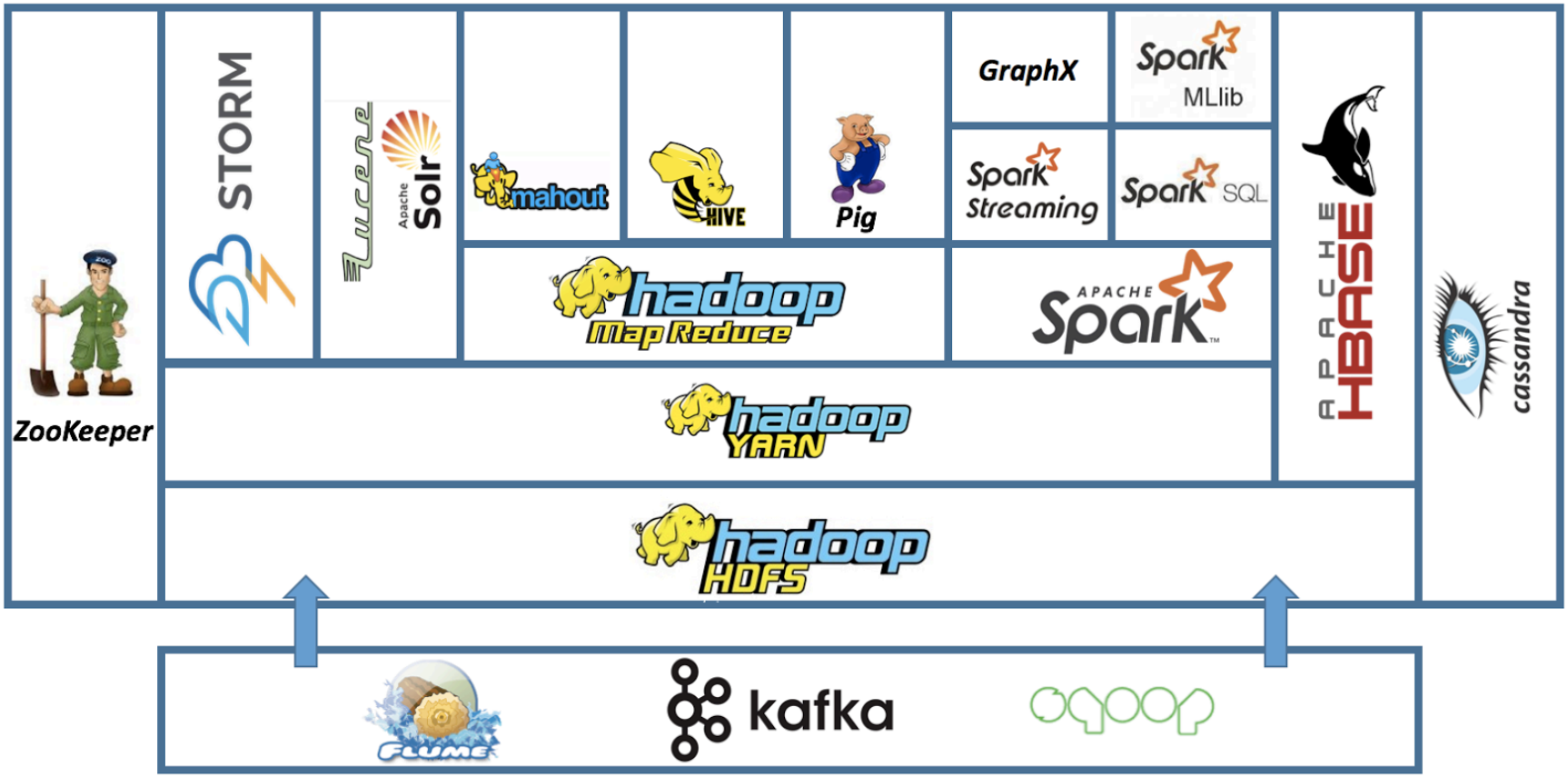
- O resultado não pode depender da ordem





Apache Hadoop

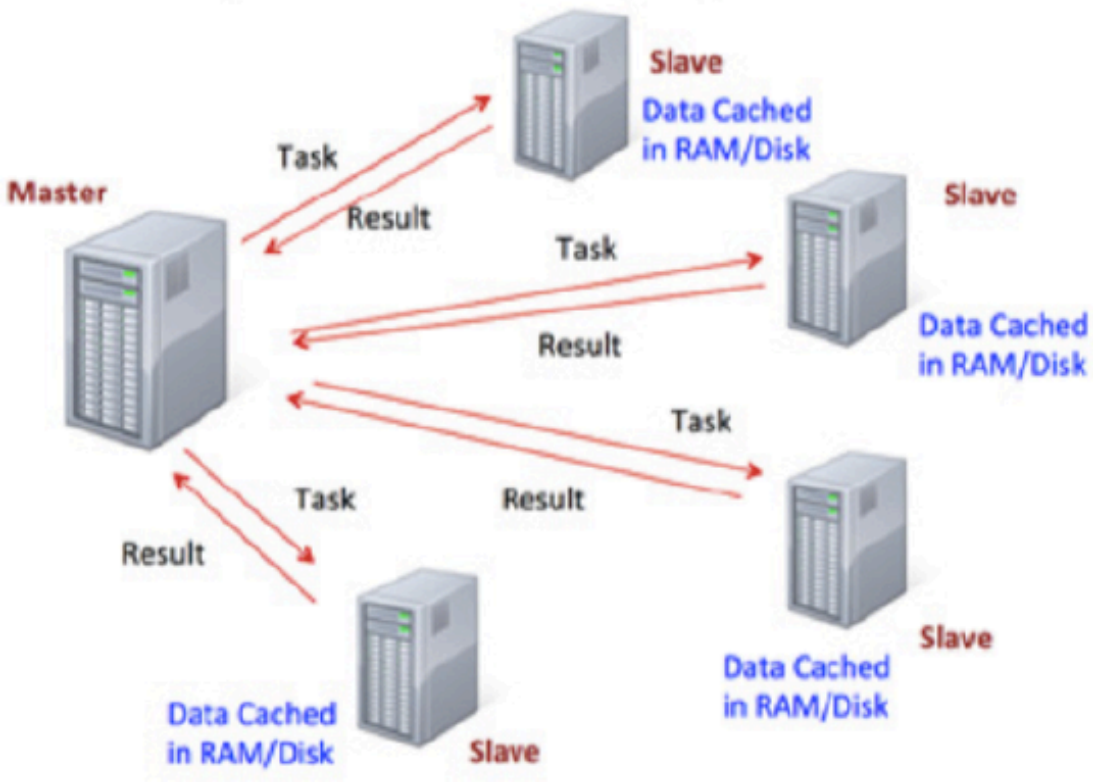
- Implementação open-source do map-reduce



Apache Spark

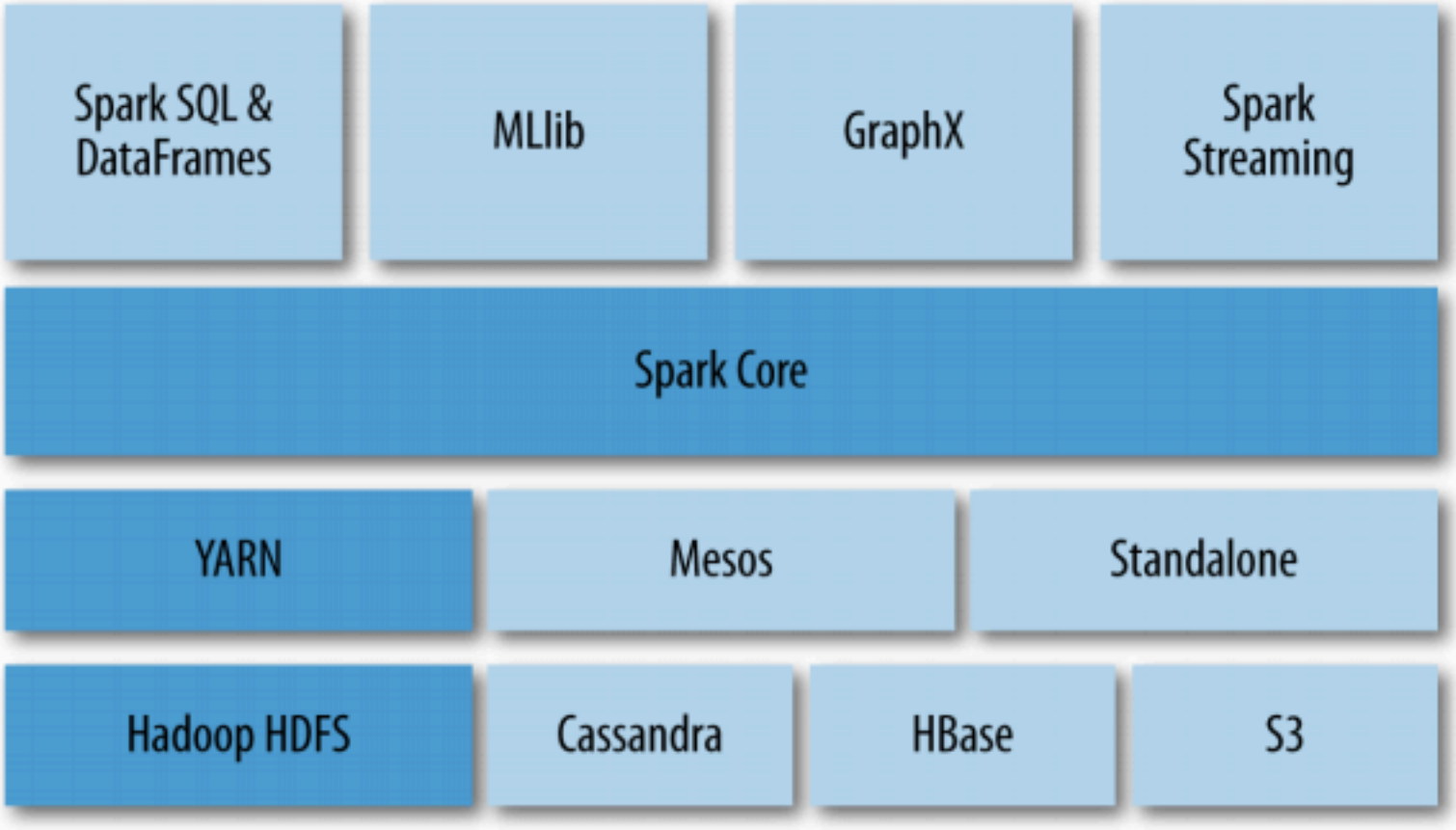
- Hadoop: Inadequado para computações iterativas (passam pelos mesmos dados muitas vezes)
- Diferença do Hadoop: memória distribuída ao invés de arquivos
- Java é a linguagem nativa do Hadoop
- Scala: linguagem nativa para Spark
  - Problema: poucas pessoas usam
- PySpark
  - Extensão de Python
  - Nem sempre possui a mesma eficiência de Scala
  - Mais fácil de aprender

Ambiente de Execução



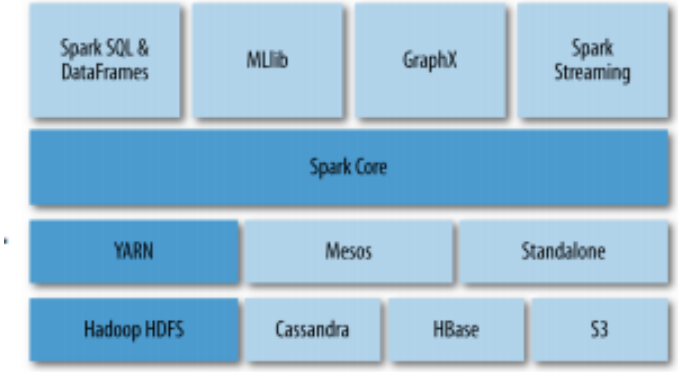
- Em máquina única: cores da cpu servem como master e slaves

Spark Stack


Spark SQL & DataFrames
MLlib
GraphX
Spark Streaming
Spark Core
YARN
Mesos
Standalone
Hadoop HDFS
Cassandra
HBase
S3

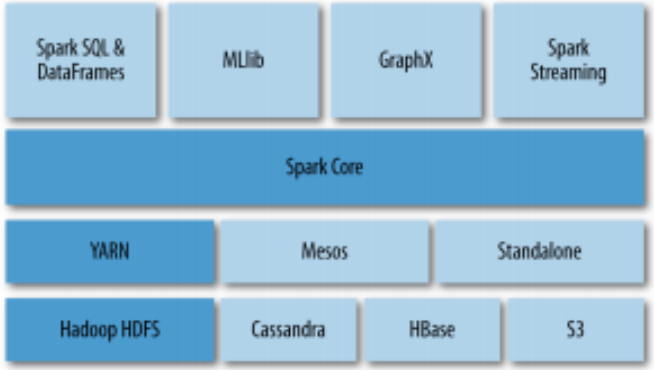
Spark Core

- Implementa a funcionalidade básica do Spark
- Componentes:
  - Escalonamento de tarefas
  - Gerenciamento de memória
  - Recuperação de falhas
- Implementa APIs baseada em RDDs
  - Resilient Distributed Datasets - Conjuntos Resilientes de Dados Distribuídos (RDDs)
  - Principal abstração de programação do Spark
  - Representam coleções de itens distribuídos pelos nós do cluster
  - Podem ser manipulados em paralelo



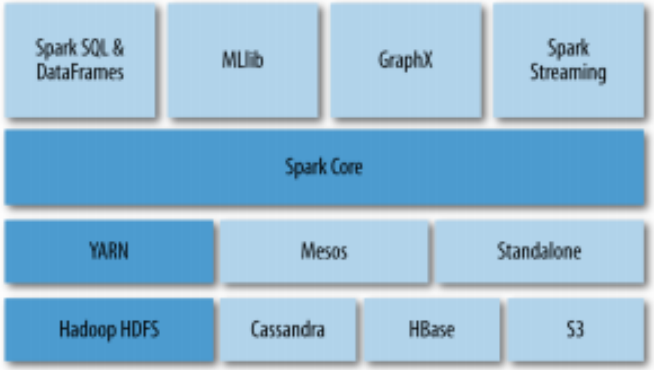
Pacotes

- Spark SQL: criação de RDD a partir de diferentes formatos: tabelas Hive, Parquet, JSON, etc..
- Spark Streaming: manipular fluxos de dados como RDDs em tempo real
  - Ex., logs gerados por servidores da web filas de mensagens, etc.
- MLlib: biblioteca de algoritmos de machine learning
- GraphX: biblioteca para manipulação de grafos
  - Ex., PageRank, centralidade em grafos etc.



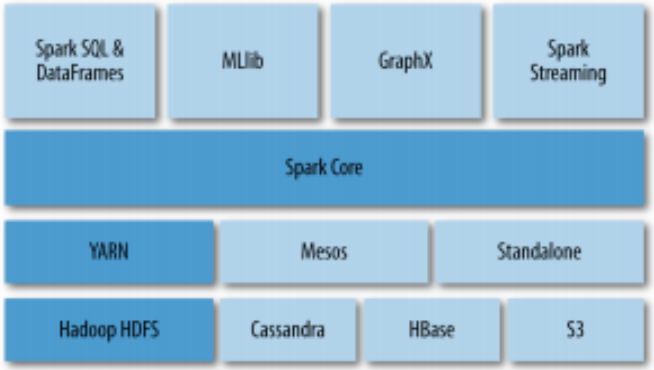
Gerenciadores de Cluster

- Standalone (próprio)
  - Pequenos clusters, dedicado ao spark
- HadoopYARN (Yet Another Resource Negotiator)
  - Clusters maiores, projetado para cargas MapReduce
  - Escalonamento em nível de aplicação
- Apache Mesos
  - Clusters maiores, projetado para diversos tipos de carga
  - Escalonamento em nível de SO
- Kubernets - baseado em containers



Armazenamento

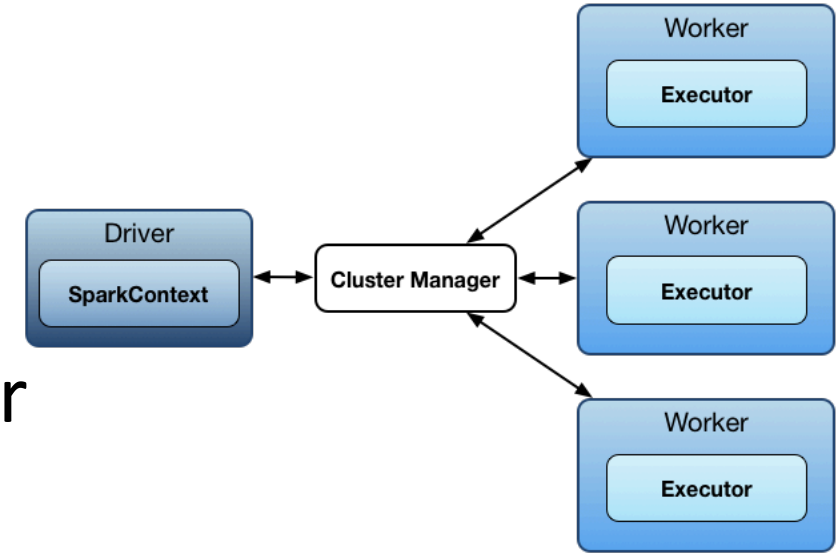
- Podem-se gerar RDDs a partir de qualquer arquivo armazenado no HDFS
- Suporte a vários meios de armazenamento
  - Sistemas de Arquivo: arquivos local, Amazon S3, etc
  - SGBDs: Cassandra, Hive, Hbase, etc
- Suporta arquivos de texto, SequenceFiles, Avro, Parquet e qualquer outro Hadoop InputFormat



Spark: Driver

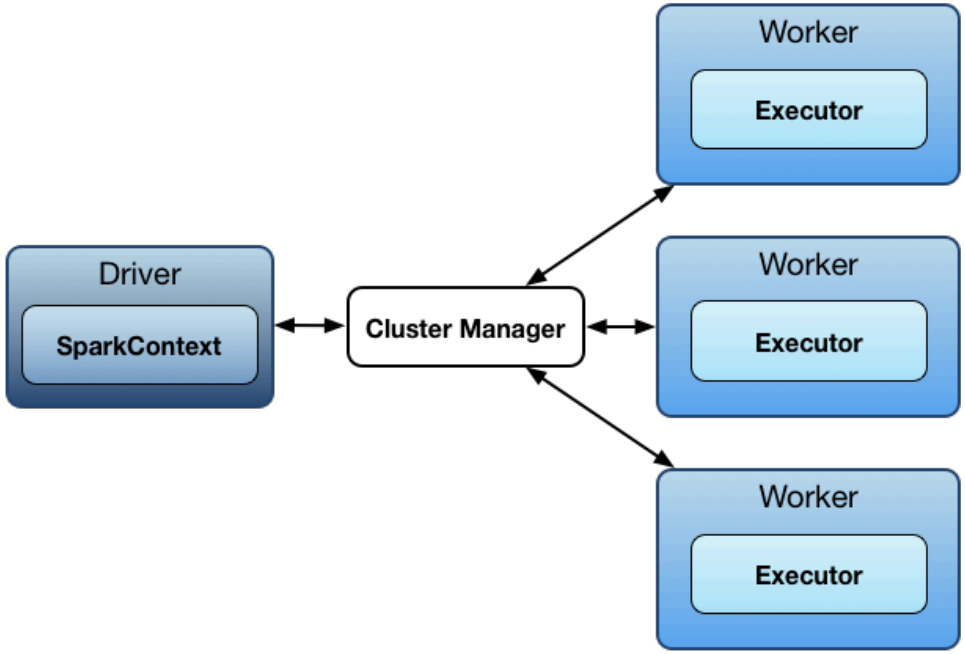
- Toda aplicação Spark consiste um programa (driver)
- Dispara várias operações paralelas em um cluster
- Define os RDDs no cluster

  e aplica operações a eles
- Utiliza um objeto SparkContext para conectar-se com o cluster



PySpark: Spark Context

- Executado no nó central
- Controla os outros nós
- Necessário somente 1

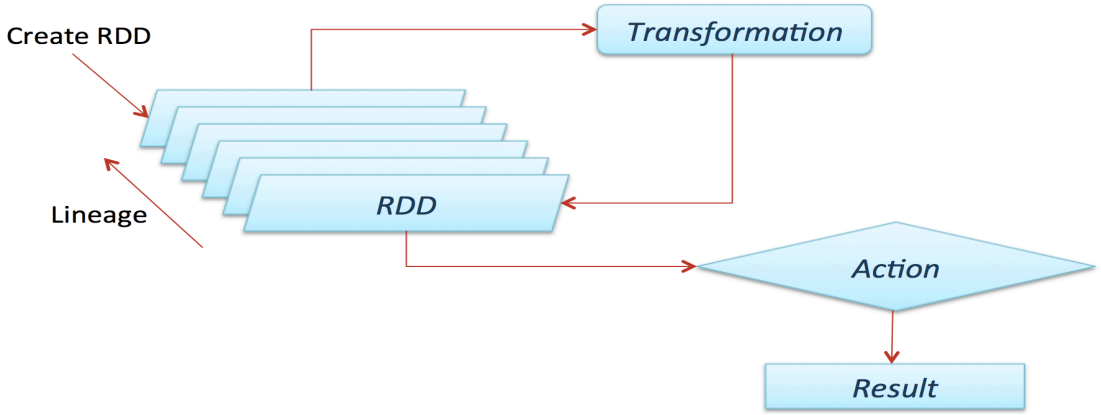


PySpark: RDD

- RDD: Resilient Distributed Dataset
- Paraleliza o processamento: clusters/cores
- Tolerante à falha
- Mantido em memória
- Imutável: somente leitura
- Lazy evaluation: dados só são materializados quando feita uma operação para isso

PySpark: RDD

- Permite três tipos de operações
  - Creation: lê dados em arquivos ou banco de dados
  - Transformation: aplicadas a um RDD para gerar outro
  - Action: computa um resultado e o retorna para o drive ou salva em um sitema de armazenamento



Exemplos de Transformações

- map: forma um RDD distribuído a partir de um RDD
- filter: novo RDD a partir de dados filtrados de um RDD
- sample: novo RDD a partir de amostra de um RDD
- union: novo RDD pela união de RDDs
- groupByKey: (k,v),…,(k,v) ⇒(k,seq[v])
- join: (k,v),…,(k,w) ⇒(k,<v,w>)

Transformação: Map

- Aplica uma operação a cada elemento do RDD
- Operações executadas em paralelo em todos executores
- Cada executor opera em dados locais dele
- Ex: B = A.map(lambda x: x-2)

Exemplos de Ações

- reduce: agregação em um RDD usando uma função
- collect: cria um objeto com todos os itens de um RDD
- count: número de elementos em um RDD
- countByKey: (k,v1),…,(k,vn) ⇒ (k,n)

Ação: Reduce

- Entrada: RDD
- Saída: único valor
- Os resultados de todos executores são combinados
- Ex: A.reduce(lambda x,y: x+y)

Diferenças entre DataFrame em Pandas e PySpark

- Pandas
  - Não roda em paralelo
  - Resultado das operações disponíveis qdo ela termina
  - Suporta mais operações
  - Operações complexas mais fáceis
- PySpark
  - Roda em paralelo em nós do cluster
  - Operações são lazy