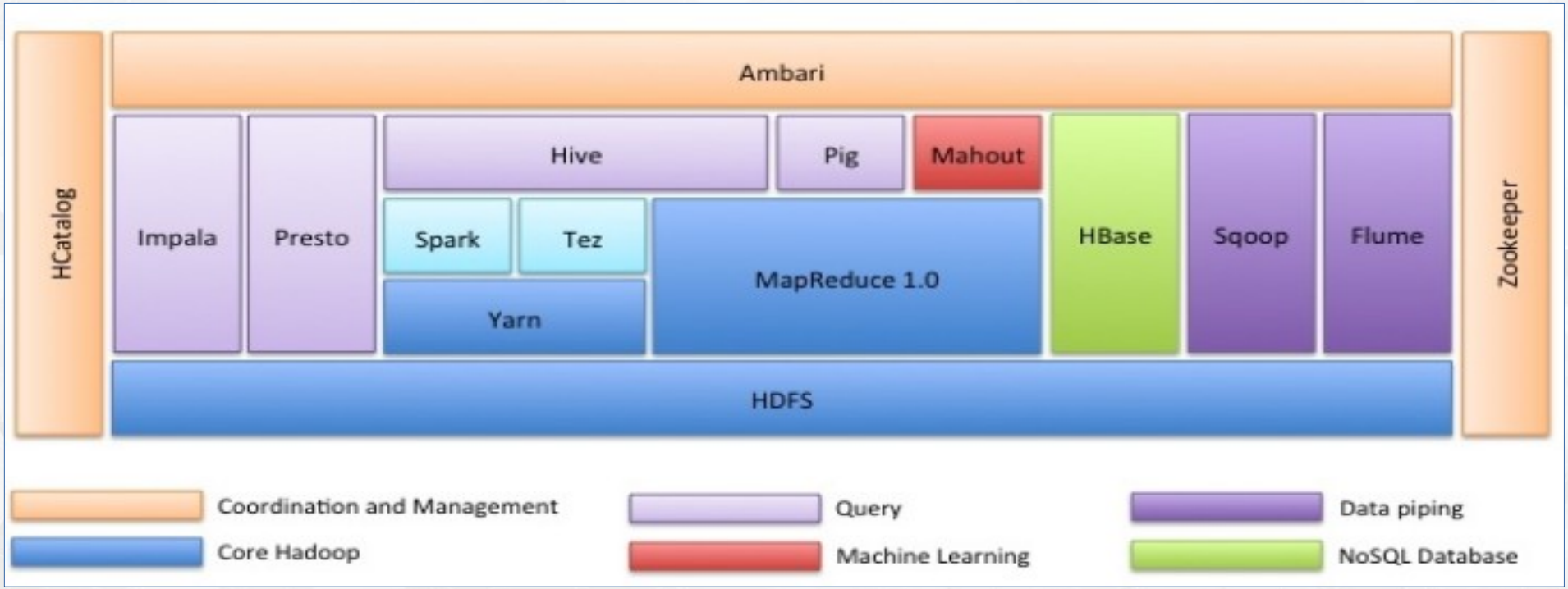
Spark
Programação Funcional para Big Data

2023

Agenda

- Spark
- Histórico
- Componentes
- Deployment
- RDDs, Dataframes, Datasets
- Configuração local
- Exemplos

Ecossistema Spark e Hadoop

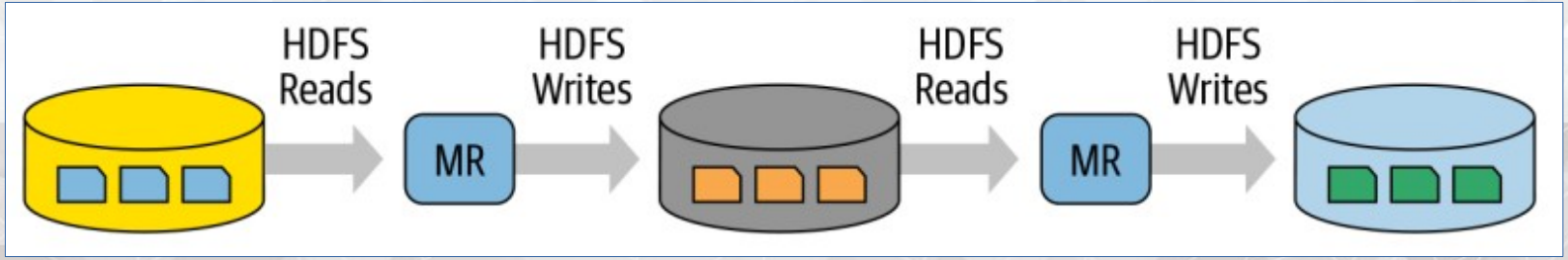


Programação para Big Data

- Hadoop usado extensivamente pela indústrica
  - Modelo de programação simples (MapReduce)
  - Escalável, flexível, tolerante a falhas, econômico
- Mas algumas questões sobre manter desempenho no processamento de datasets muito grandes
  - Tempo de espera entre as queries
  - Tempo de espera para iniciar programas
  - Google passou a usar Big Table por conta disso
  - API Map/Reduce nem sempre adequada a todos os casos

Programação para Big Data

- Hadoop adotado no Yahoo!
  - E logo disseminado para diversos outros ambientes
  - Doado à Apache Software Foundation em 2006
- MapReduce framework tinha algumas questões
  - Difícil administração, complexidade operacional
  - Muito código para operações simples e padronizadas
    - Boilerplate code
  - Leituras e gravações intermitentes



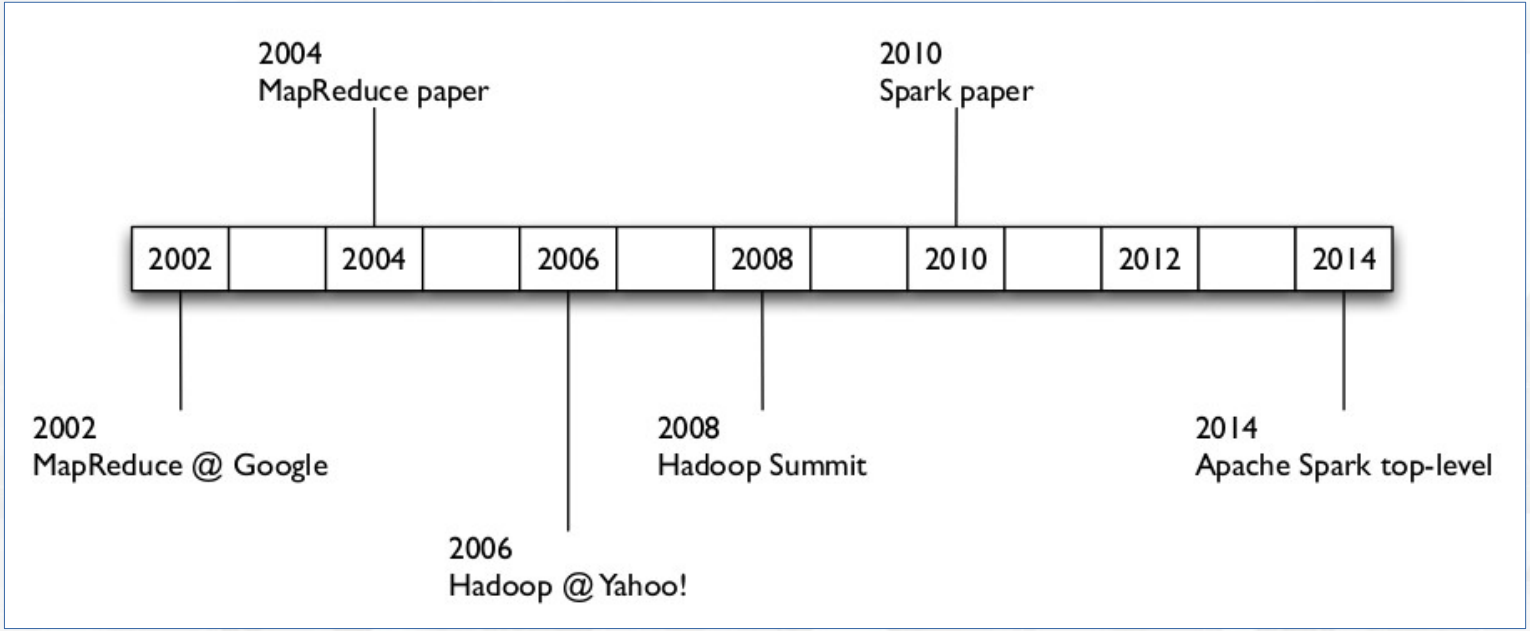
Spark

- AMPLab
  - https://amplab.cs.berkeley.edu/
  - UC Berkeley
  - Hoje RISELab
- Projeto Spark
  - Iniciado em 2009
  - Desenvolvido por Matei Zaharia, orientado por Ion Stoica
  - Reconhecimento que MR pode ser ineficiente para jobs interativos
    - Além da complexidade do framework
  - Papers iniciais mostraram desempenho 10 a 20 x melhor

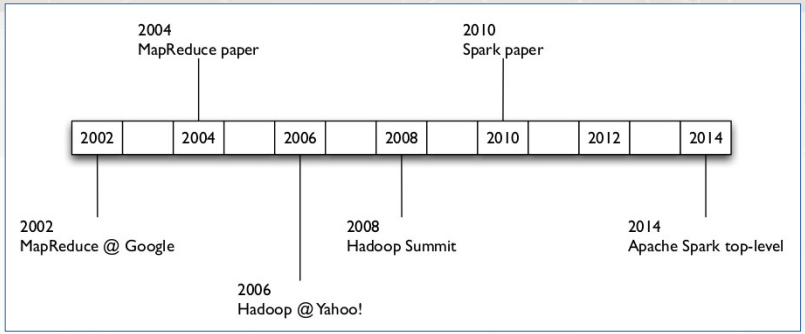
Spark

- Código aberto em licença BSD em 2010
- Passou a ser amplamente utilizado em torno de 2013
  - Quando foi doado para a Apache Software Foundation
  - Passou a projeto top-level em 2014
- Um grupo dos criadores originais fomentou a doação para a ASF
  - Matei Zaharia, Ali Ghodsi, Reynold Xin, Patrick Wendell, Ion Stoica e Andy Konwinski
  - E fundaram a Databricks no mesmo ano
- Apache Spark 1.0 foi lançado em 2014
  - Desenvolvedores da comunidade e da Databricks

Spark – Histórico



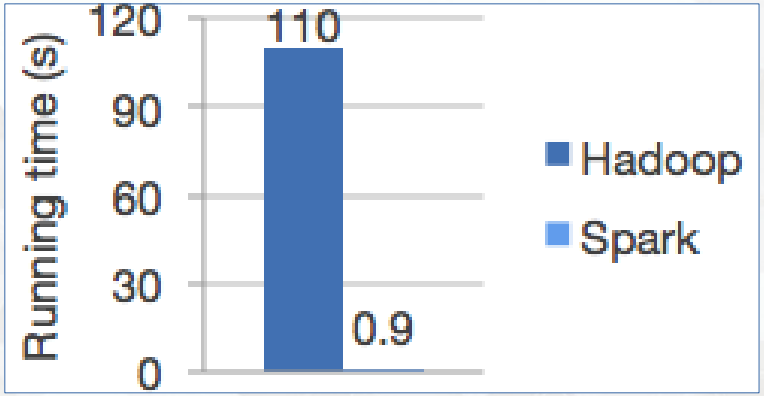
Spark – Papers seminais



- Spark: Cluster Computing with Working Sets
  - Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica
  - University of California, Berkeley
- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing
  - Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
  - University of California, Berkeley

O que é o Apache Spark?

- Computação em cluster rápida
  - “lightning-fast”
- Projetado para tratar diversos tipos de workloads
  - Aplicações batch, algoritmos interativos, queries interativas, streaming, machine learning, grafos
  - Reduz a sobrecarga de gerenciamento de manter diversas ferramentas separadas

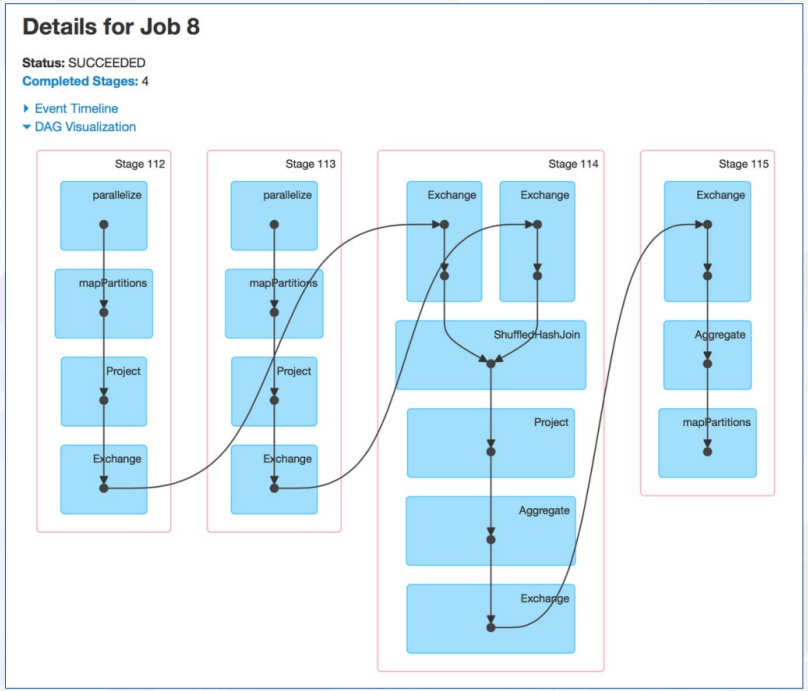


O que é o Apache Spark?

- Unified Engine
  - Processamento de dados distribuídos em larga escala
  - On premises ou na nuvem
- Processamento intermediário com armazenamento in-memory
  - Calcanhar de aquiles do framework MapReduce
- Diversas APIs disponíveis
  - MLlib, Spark SQL, stream processing, GraphX
- Filosofia de projeto centrada em 4 características
  - Desempenho
  - Facilidade de uso
  - Modularidade
  - Extensibilidade

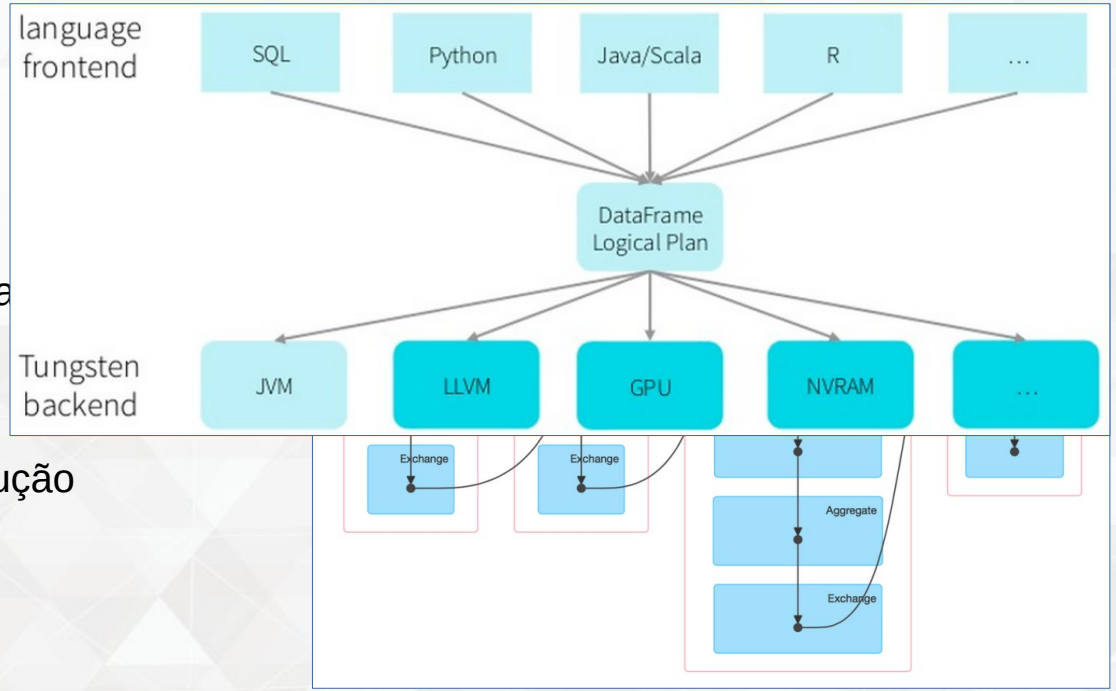
O que é o Apache Spark

- Desempenho
  - Uso de múltiplos cores e memória
  - DAG
    - Directed Acyclic Graph
    - DAG scheduler, otimização de queries, tarefas paralelas
  - Tungsten
    - Physical execution engine
    - Geração de código compacto para execução
  - In-memory
    - Resultados intermediários em memória
    - Redução de I/O de disco



O que é o Apache Spark

- Desempenho
  - Uso de múltiplos cores e memória
  - DAG
    - Directed Acyclic Graph
    - DAG scheduler, otimização de queries, ta
  - Tungsten
    - Physical execution engine
    - Geração de código compacto para execução
  - In-memory
    - Resultados intermediários em memória
    - Redução de I/O de disco



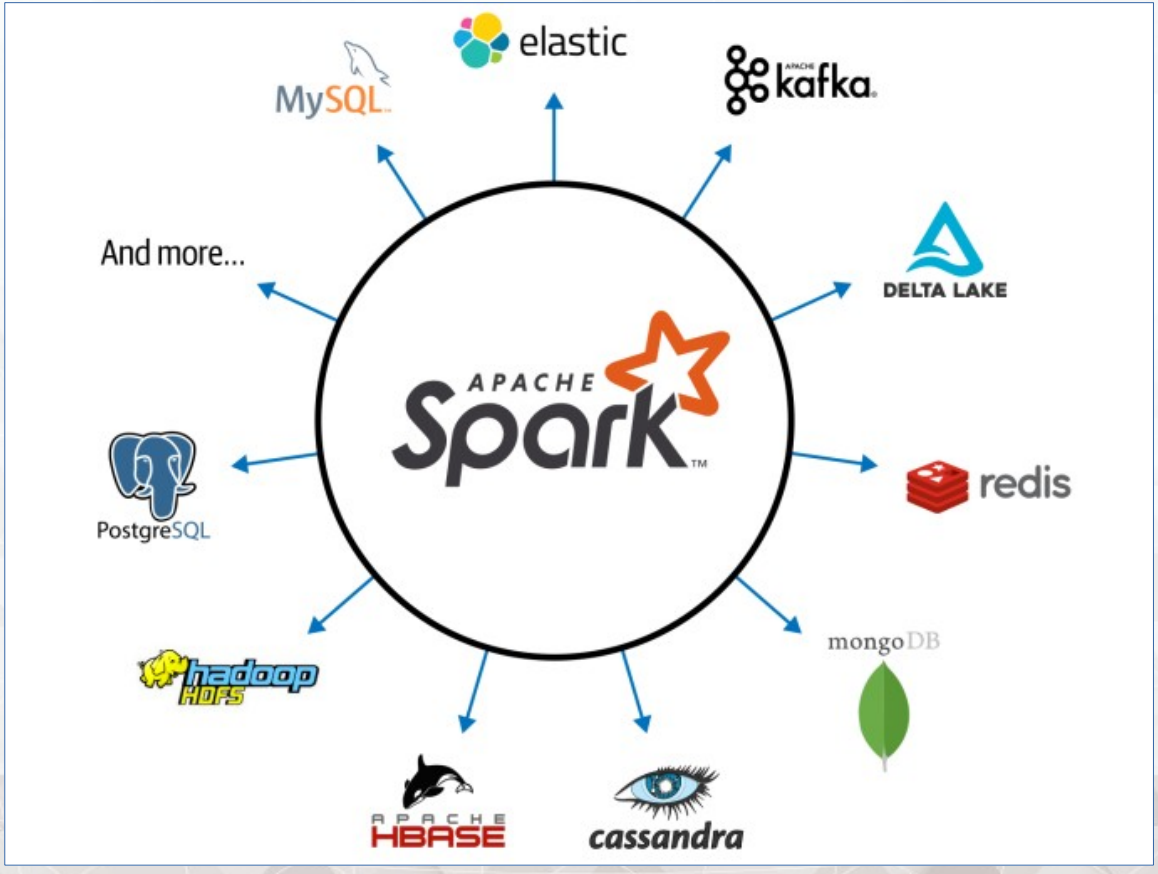
O que é o Apache Spark

- Facilidade de uso
  - Abstração fundamental: RDDs
    - Resilient Distributed Dataset
    - Estrutura lógica simples
  - Abstrações estruturadas de dados baseadas em RDDs
    - DataFrames e Datasets
  - Operações de transformações e ações
    - Modelo de programação simples
    - Fácil adaptação para os DAGs
    - Inclusão em diversas linguagens

O que é o Apache Spark

- Modularidade
  - Diversas linguagens, diversas bibliotecas
    - Scala, Java, Python, SQL, R
    - Spark SQL, Spark Structured Streaming, Spark MLlib, GraphX
  - Uma única aplicação Spark pode usar todas essas ferramentas
- Extensibilidade
  - Não é fortemente acoplado a componentes
  - Pode se conectar a Hadoop, Cassandra, HBase, MongoDB, Hive, bancos relacionais, etc
  - Fontes de dados em Kafka, Kinesis, Azure Storage, Amazon S3
  - Pacotes de terceiros podem ser incluídos
    - https://spark.apache.org/third-party-projects.html

O que é o Apache Spark



O que Spark não é?

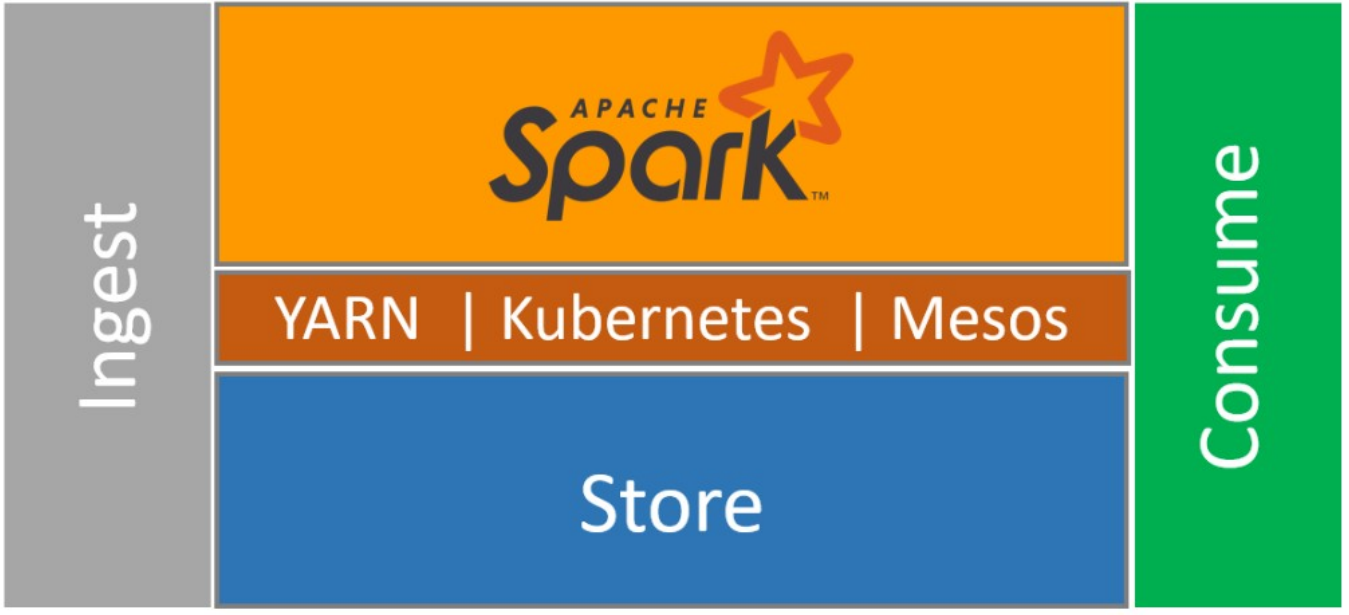
- Spark NÃO É uma versão modificada do Hadoop
  - Na realidade, nem sequer é dependente do Hadoop
    - Tem seu próprio gerenciador de clusters, baseado no Mesos
    - Pode ser implementado em outros gerenciadores também
- Pode usar Hadoop (ou partes dele) em duas maneiras
  - Storage (HDFS)
  - Processing (YARN)

Um novo ecossistema para DataLakes e BigData

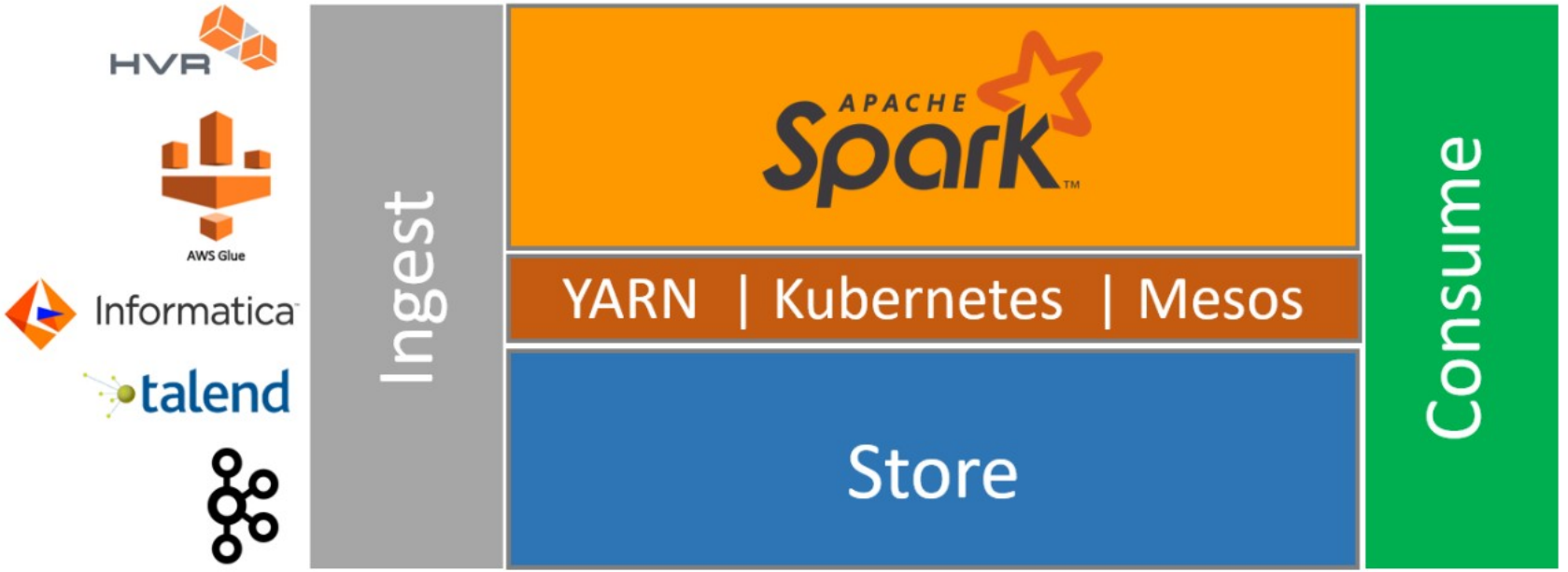
Um novo ecossistema para DataLakes e BigData

Um novo ecossistema para DataLakes e BigData

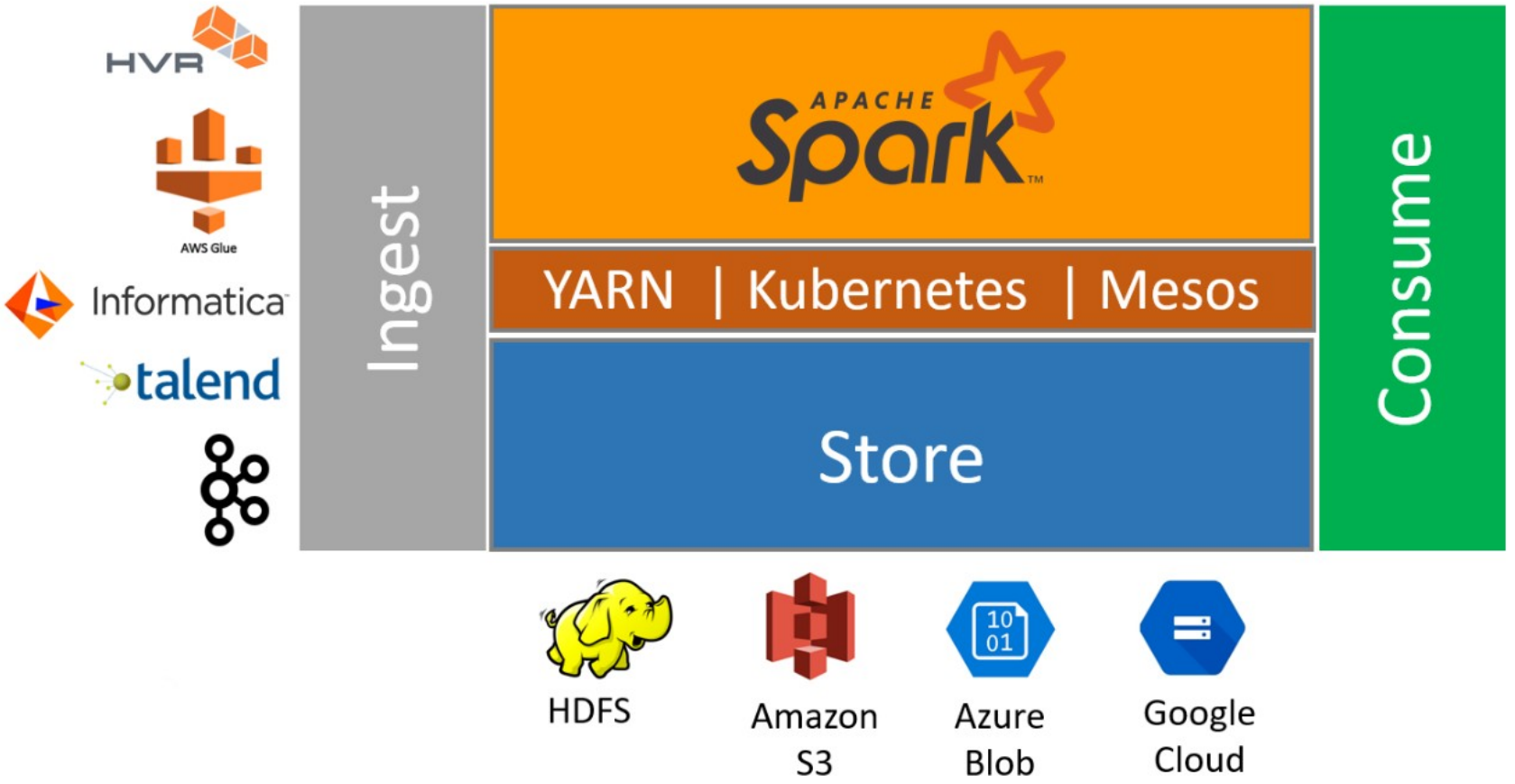
Um novo ecossistema para DataLakes e BigData



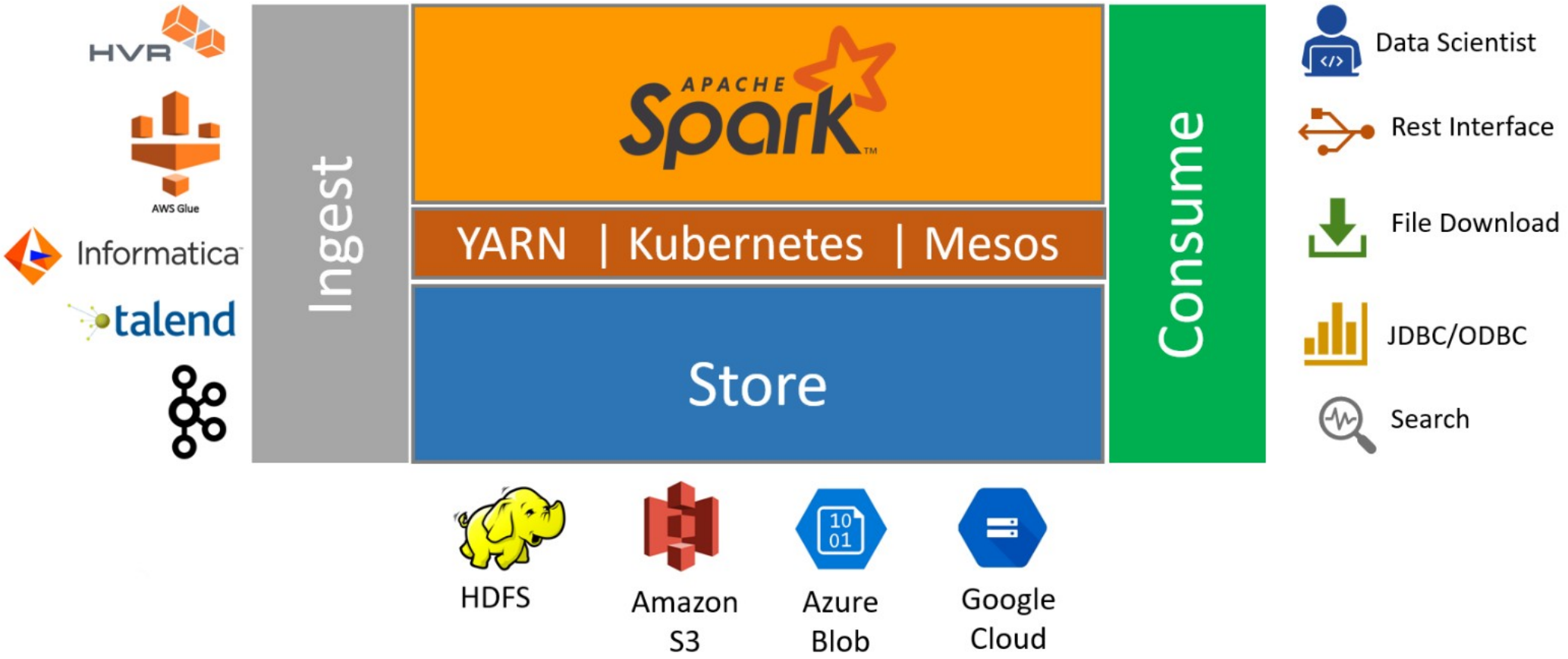
Um novo ecossistema para DataLakes e BigData



Um novo ecossistema para DataLakes e BigData



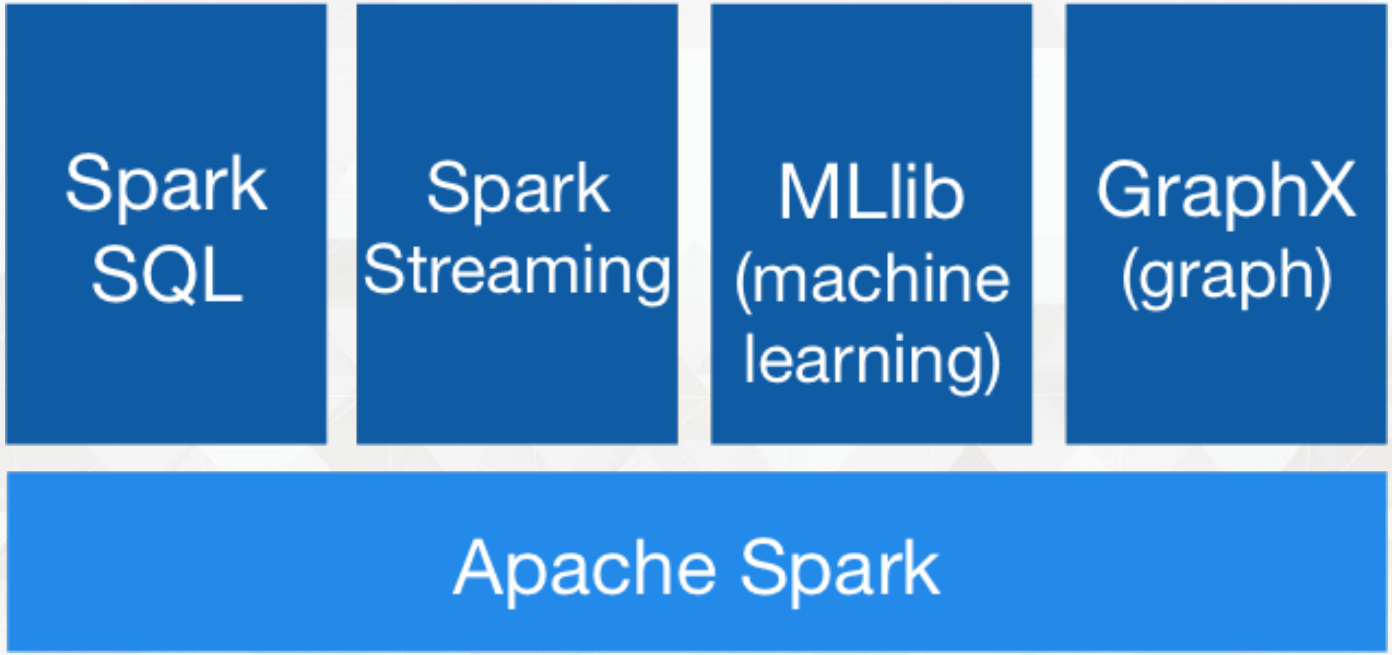
Um novo ecossistema para DataLakes e BigData



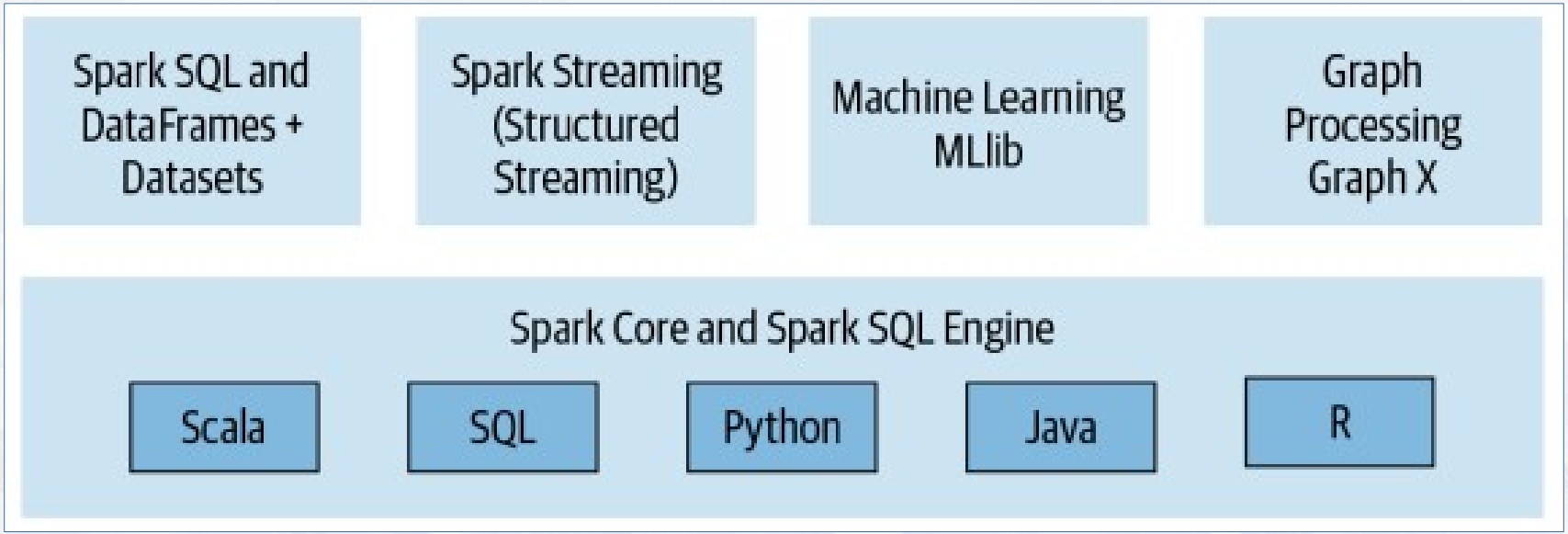
Unified Analytics

- Em 2016, os criadores do Spark receberam o ACM Award
  - Association for Computing Machinery
  - Pelo paper descrevendo Spark como um “Unified Engine for Big Data Processing”
- Vários diferentes componentes de Big Data podem ser unificados com o Spark
  - Menor complexidade, custos de manutenção, operação, treinamento, etc.

Componentes



Componentes



Componentes

- Spark SQL
  - Usado em Java, Python, Scala ou R
  - Dados estruturados
  - Formatos
    - RDBMs, NoSQL (MongoDB, Cassandra)
    - Arquivos
      - Texto puro, CSV, JSON, Avro, ORC, Parquet
    - Também combina dados de fontes estruturadas com DataFrames existentes
  - SQL ANSI 2003

Componentes

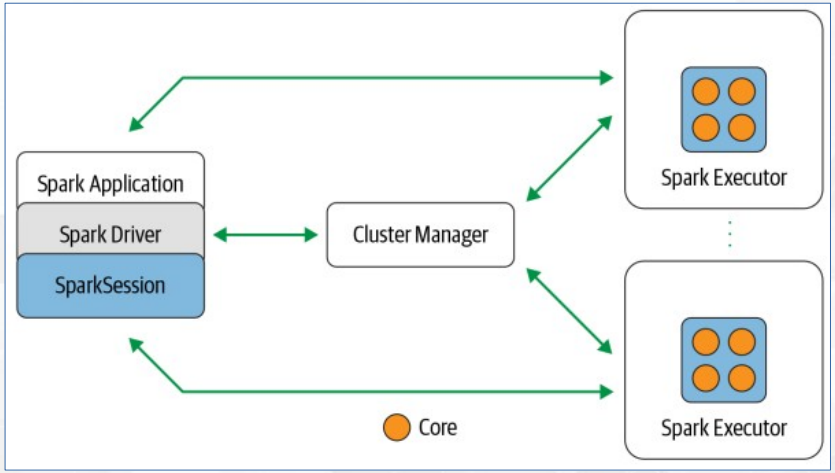
- Spark MLlib
  - MLlib Project
    - Dois packages
      - spark.mllib até versão 1.6
        - Baseada em RDD
      - spark.ml após isso
        - Baseada em DataFrames
  - Desempenho melhorou bastante após a versão 2.x do Spark
  - API em desenvolvimento
    - Extrair e transformar features, construir pipelines, persistir modelos, deployment, etc
  - Maior fomento a desenvolvimento após versão 2.x

Componentes

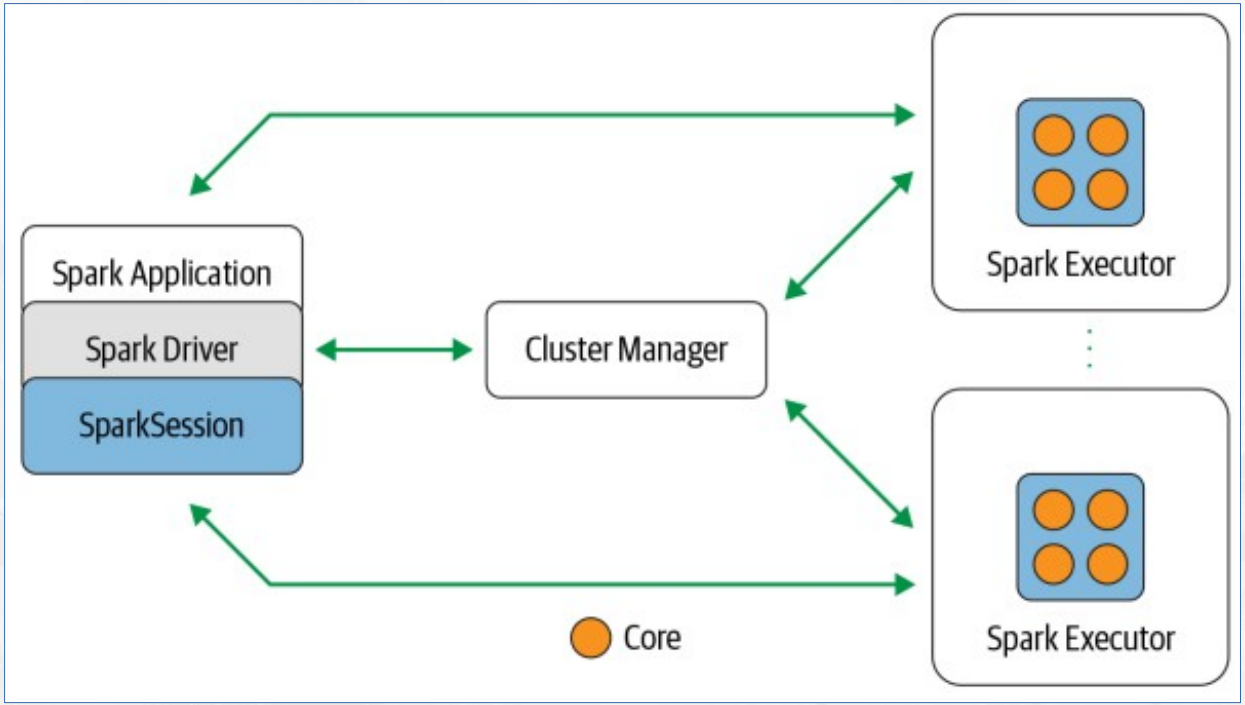
- Spark Structured Streaming
  - Iniciado no Spark 2.0
  - Modelo de streaming contínuo
    - Construído sobre Spark SQL e DataFrames
    - Vê o streaming como uma tabela que cresce continuadamente
- GraphX
  - Manipulação de grafos
    - Análise de redes sociais, rotas, pontos de conexão, topologias de rede, etc
  - Operações paralelizadas em grafos
  - Algoritmos padronizados
    - Page Rank, Connected Components, Triangle Couting, centralidades, cliques, etc

Execução distribuída

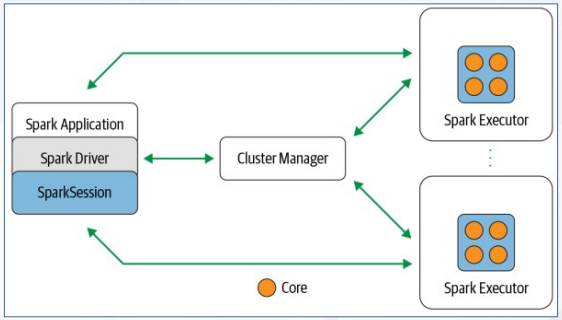
- Aplicação Spark consiste em:
  - Driver program
    - Orquestração das tarefas
    - Acessa componentes distribuídos no cluster
  - Spark executors
    - Workers
  - Cluster Manager
    - Master
  - Acesso através de uma SparkSession



Execução distribuída



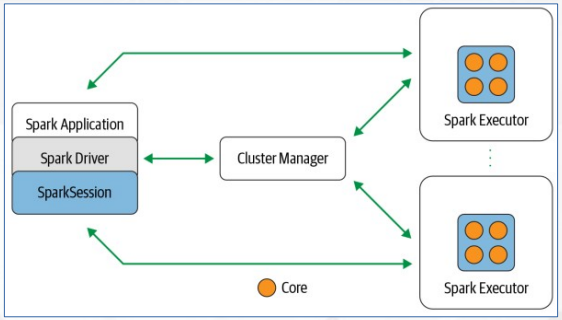
Execução distribuída



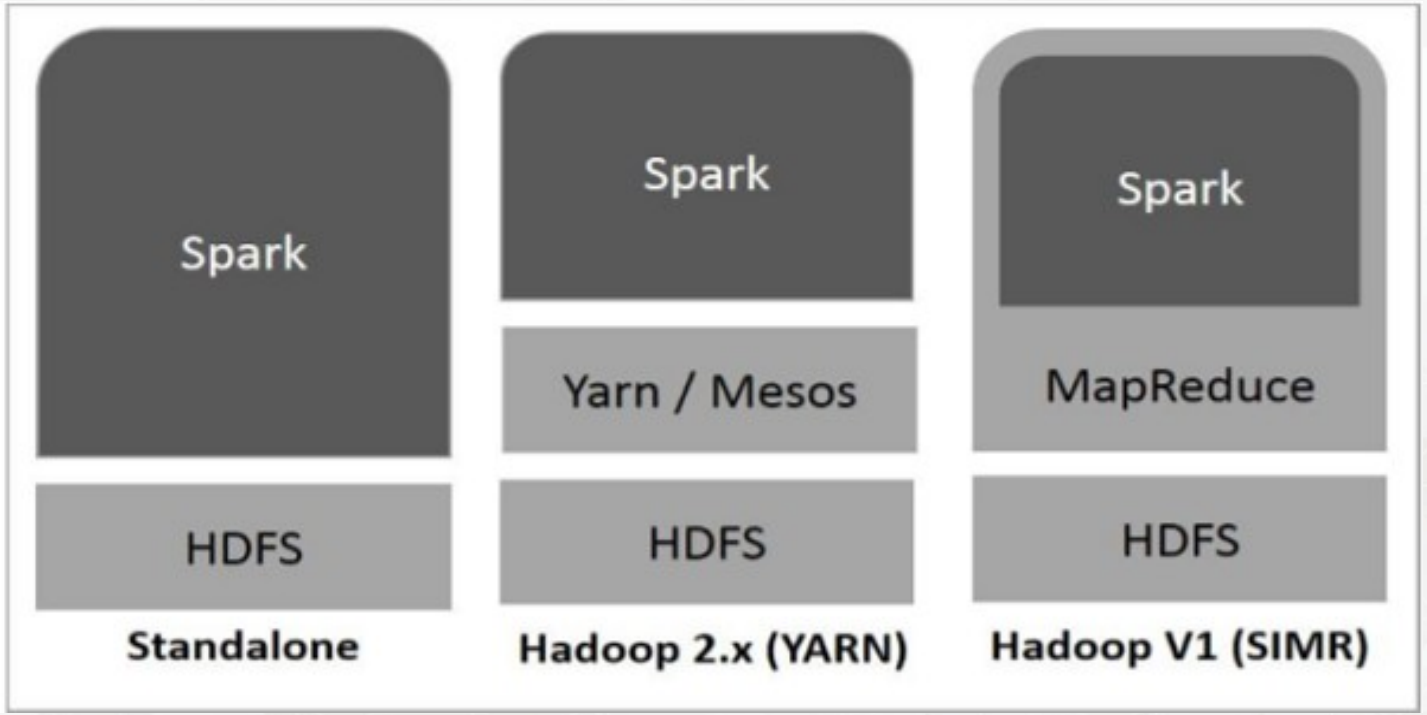
- Spark Driver
  - Inicia a SparkSession
  - Se comunica com o cluster manager
  - Requisita recursos (CPU, memória)
  - Transforma operações em tarefas do DAG, escalona e distribui as tarefas nos workers
- SparkSession
  - Acesso unificado aos recursos de operações e dados do Spark
    - Em versões anteriores, tarefa era dividida entre SparkContext, SQLContext, HiveContext, SparkConf, StreamingContext
      - Classes da versão 1.x continuam a funcionar, por componentes da SparkSession
  - Operações das aplicações são feitas através da SparkSession
  - Criada por operações das APIs
    - Spark shell cria uma session por default

Execução distribuída

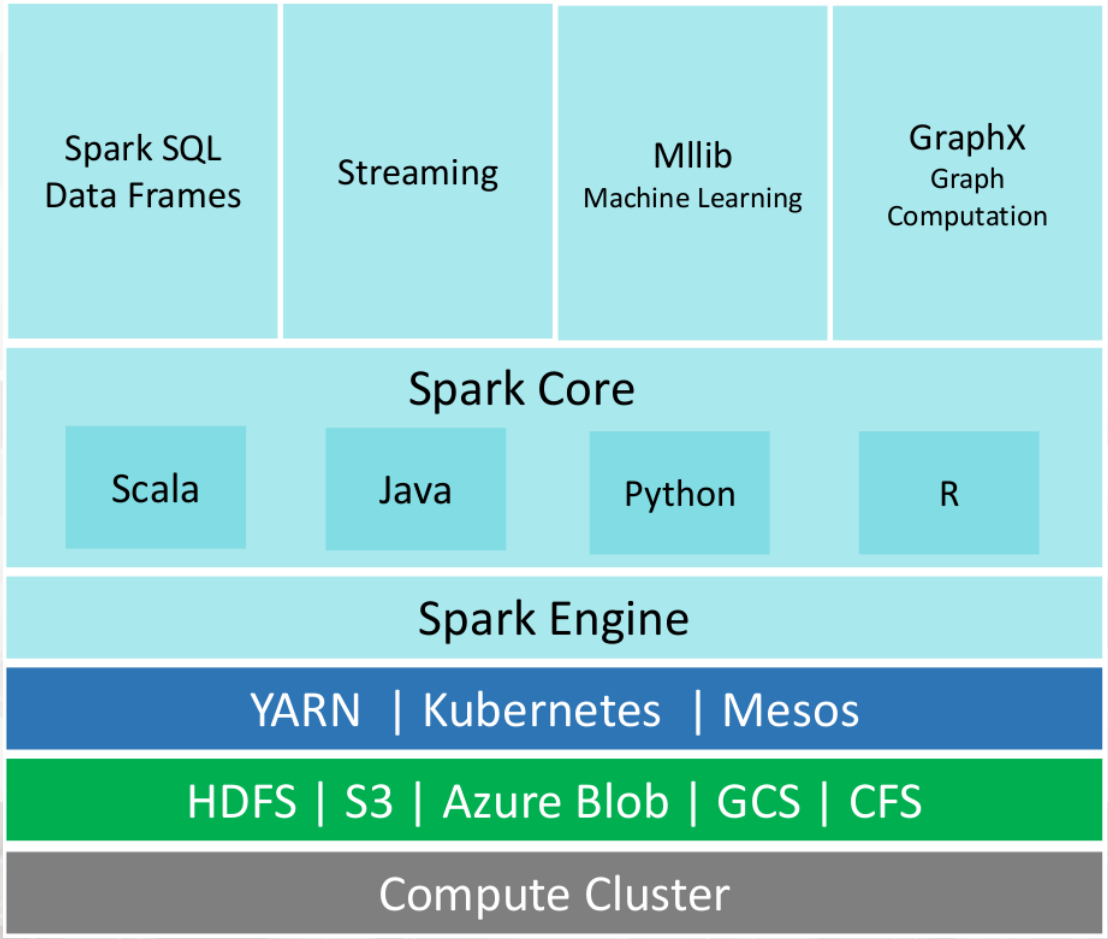
- Cluster manager
  - Gerenciamento e alocação de recursos
  - Spark suporta diversos cluster managers
    - Built-in
    - Apache Hadoop Yarn
    - Apache Mesos
    - Kubernetes
- Spark executor
  - Ativo em cada nó do cluster
  - Executa as tarefas agendadas pelo Driver Program



Spark – deployment

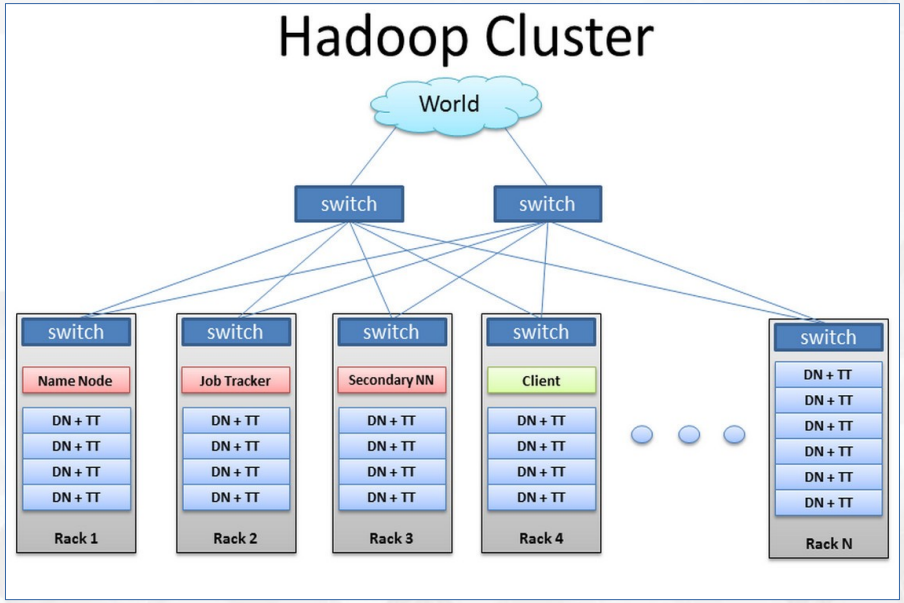


Spark – deployment



Modos de Deployment

- Local
  - Driver e executor rodam em uma mesma JVM local
  - Geralmente em um só computador (ou nó)
- Standalone
  - Executa em qualquer nó, em JVM separada
  - Lança ao menos um master e um worker
  - Conexão e submissão de jobs por protocolo próprio
- YARN
  - Cliente ou cluster
    - Driver Executando em um nó do cluster ou no YARN Application Master
  - Workers executando nos containers alocados pelos NodeManagers
- Kubernetes
  - Usa o Kubernetes Master como cluster manager
  - Driver executa em um pod Kubernetes
  - Cada worker executa em seu próprio pod



Modos de Deployment

- Acesso a armazenamento
  - Local e standalone
    - Armazenamento local
  - YARN
    - HDFS
  - Kubernetes
    - HDFS
  - Nuvem
    - Armazenamento distribuído em nuvem

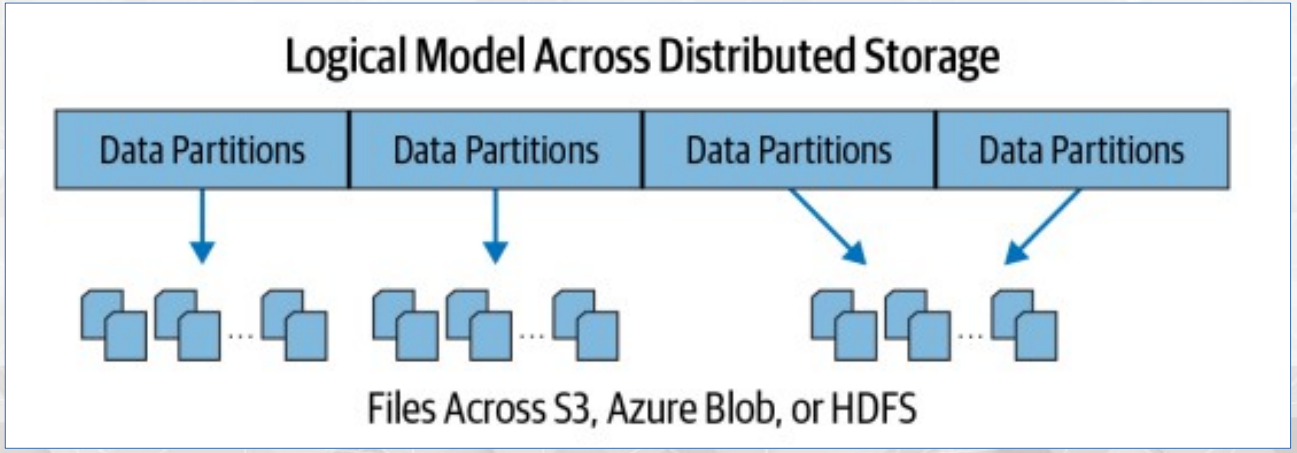
Modos de Deployment

- Acesso a armazenamento
  - Local e standalone
    - Armazenamento local
  - YARN
    - HDFS
  - Kubernetes
    - HDFS
  - Nuvem
    - Armazenamento distribuído em nuvem

Databricks oferece sua própria nuvem

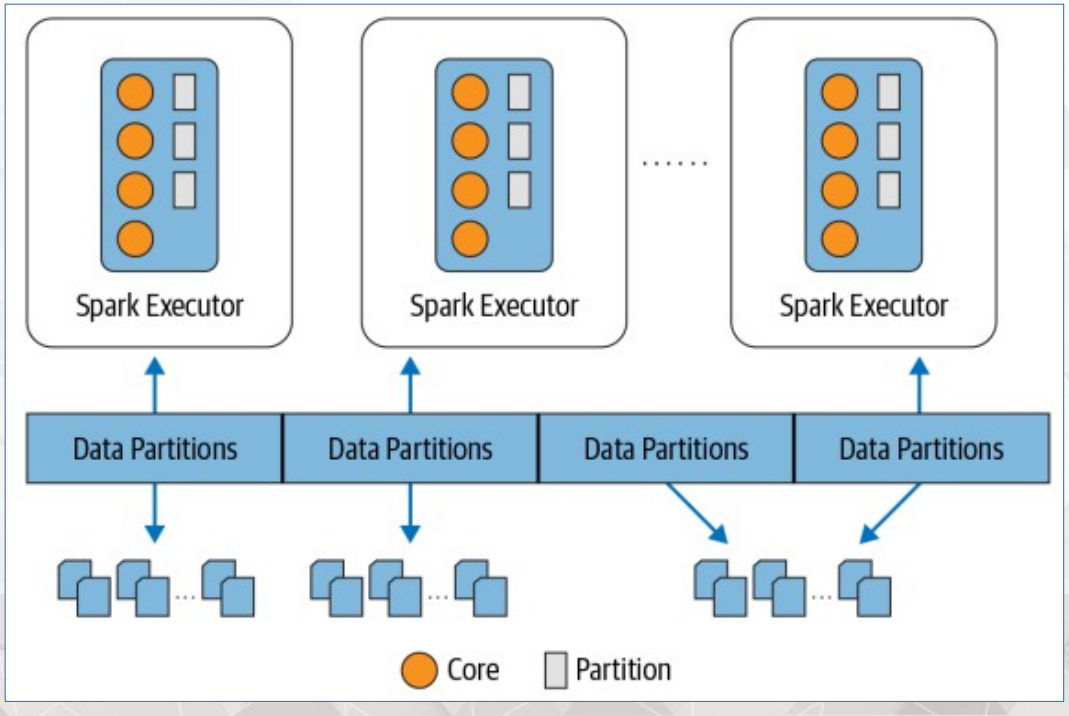
Dados distribuídos e partições

- Dados particionados
  - Em HDFS ou armazenamento em nuvem
  - Permite eficiência em paralelismo
    - Cada worker procura tratar uma partição local



Dados distribuídos e partições

- Método repartition em DataFrames
  - Permite acesso local, minimizando uso de rede



Utilização e comunidade

- Criado para tarefas de manipulação de dados e Big Data
  - Data science
    - Limpeza, exploração de dados
    - Descoberta de padrões, modelos de ML
    - Data wrangle
  - Grandes volumes de dados
    - Project Hydrogen
      - Modelos de deep learning distribuídos
  - Spark 3.0 suporta GPUs em cluster

Utilização e comunidade

- Engenharia de dados
  - Pipelines
    - Acesso e processamento de dados, streams, modelos ML
- Comunidade ativa
  - Mais de 600 Apache Spark Meetup groups
    - Globalmente
  - Spark + AI Summit
    - Conferência para uso de Spark em ML e data science
  - 1500 contribuidores para código no projeto

Spark RDD

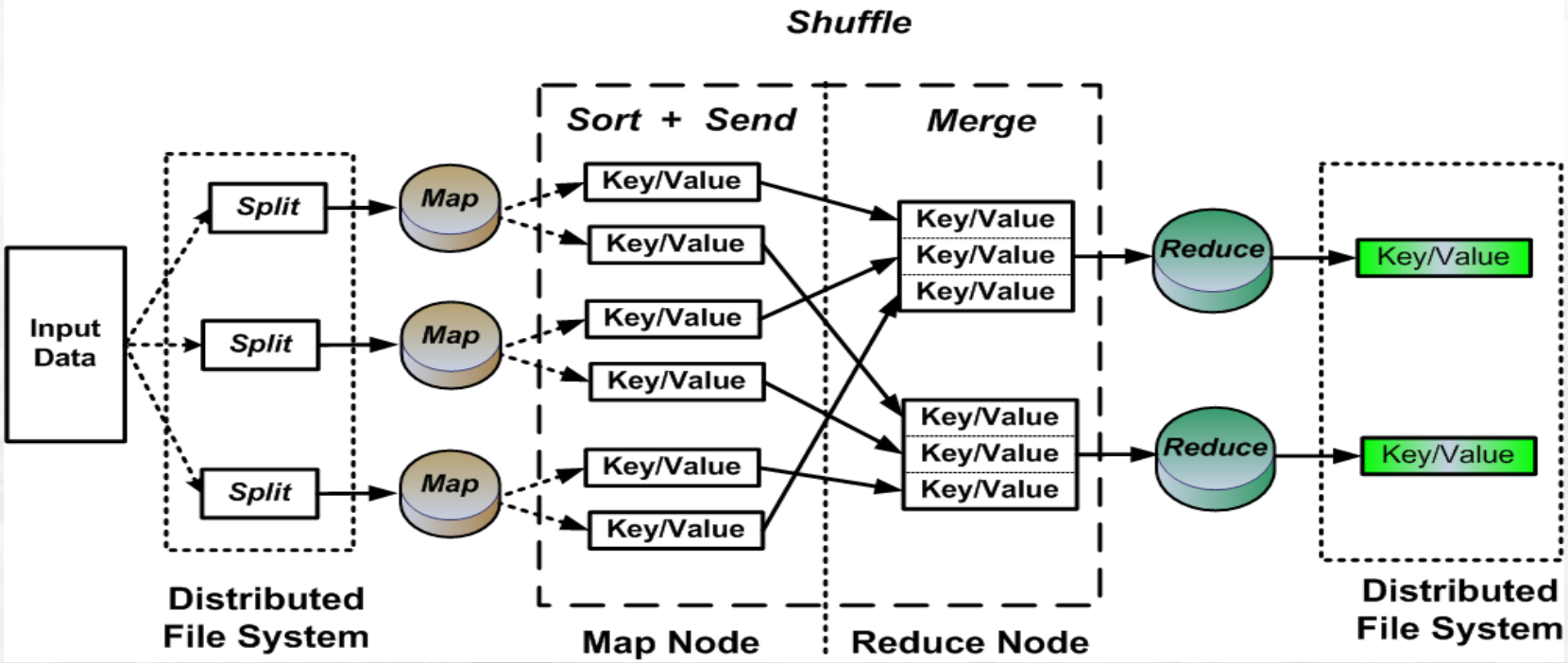
- RDD – Resilient Distributed Datasets
  - Estrutura de dados fundamental do Spark
- Coleção imutável de objetos
  - Pode conter qualquer tipo de objetos Scala, Java ou Python
    - Incluindo user-defined classes
  - Cada dataset em RDD é dividido em partições lógicas
  - Cada partição pode ser tratada em diferentes nós do cluster
- Formalmente: coleção particionada de registro somente para leitura
  - read-only, partitioned collection of records
    - Criada através de operações determinísticas sobre dados em armazenamento ou outros RDDs
    - Coleção tolerante a falhas que pode ser tratada com paralelização

Porque RDDs?

- Ou qualquer outra collection...
  - Já não temos opções demais disponíveis?
- MapReduce (e soluções similares) é muito adotado
  - Processar datasets grandes com um algoritmo distribuído e paralelizado em um cluster
  - API de alto nínel
    - Não é necessário se preocupar com distribuição, tolerância a falhas, etc
- Mas...
  - No Hadoop MapReduce, a única maneira de se reusar dados entre operações é gravando para um armazenamento estável externo (como HDFS)

Porque RDDs?

MapReduce workflow

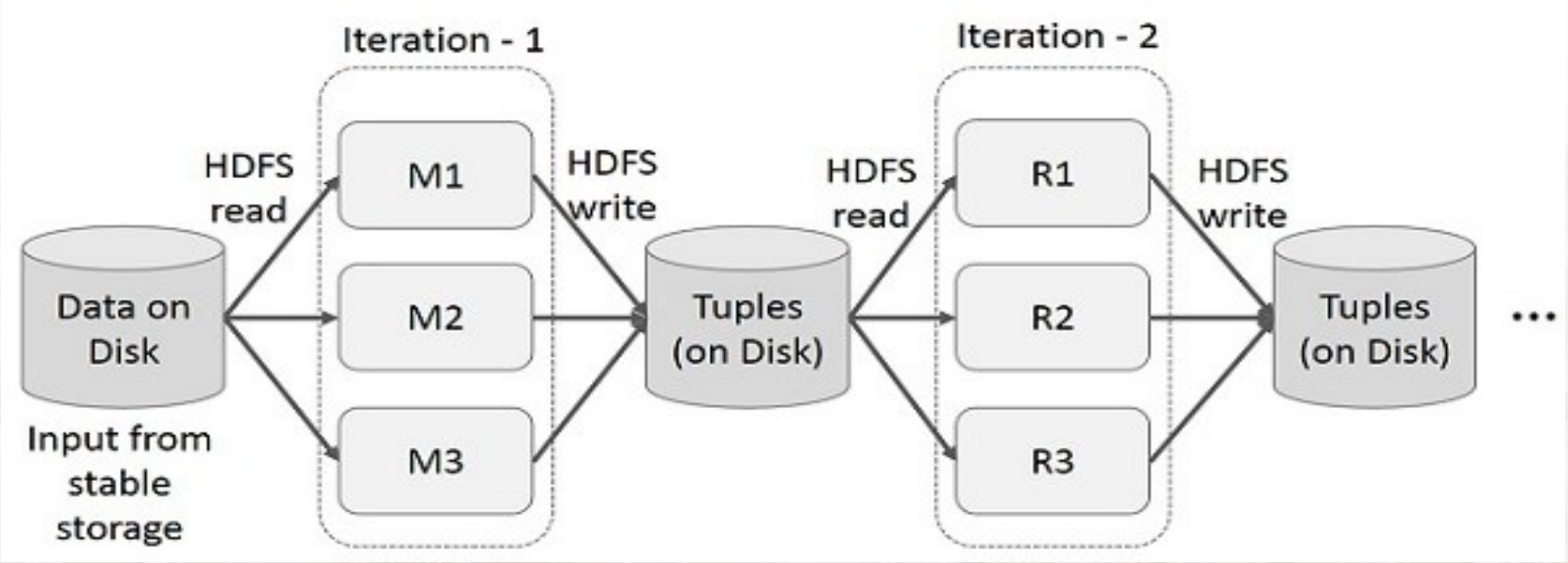


Porque RDDs?

- Vários frameworks já fornecem acesso rápido para dados em trânsito
  - Com sucesso limitado
- Compartilhamento de dados é lento em MapReduce devido as mesmas características que fazem o framework ser útil
  - Replicação
  - Serialização
  - Disk IO
- Aplicações Hadoop gastam mais de 90% do tempo em operações read/write em HDFS

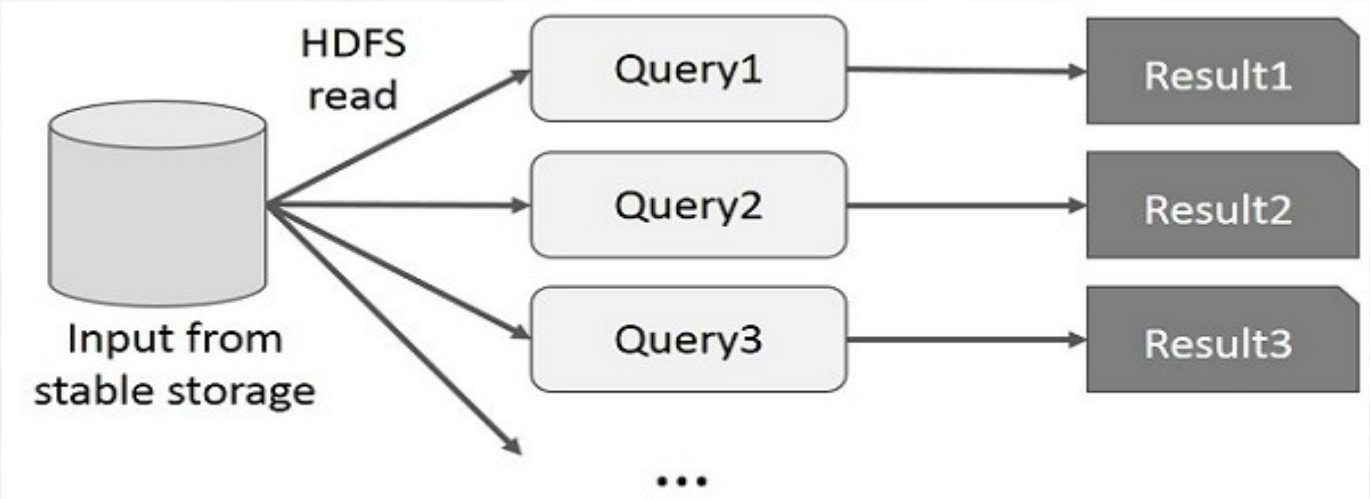
Operações iterativas em MapReduce

Reutilizando resultados intermediários ao longo de múltiplas operações



Operações iterativas em MapReduce

Queries ad-hoc nos mesmos dados

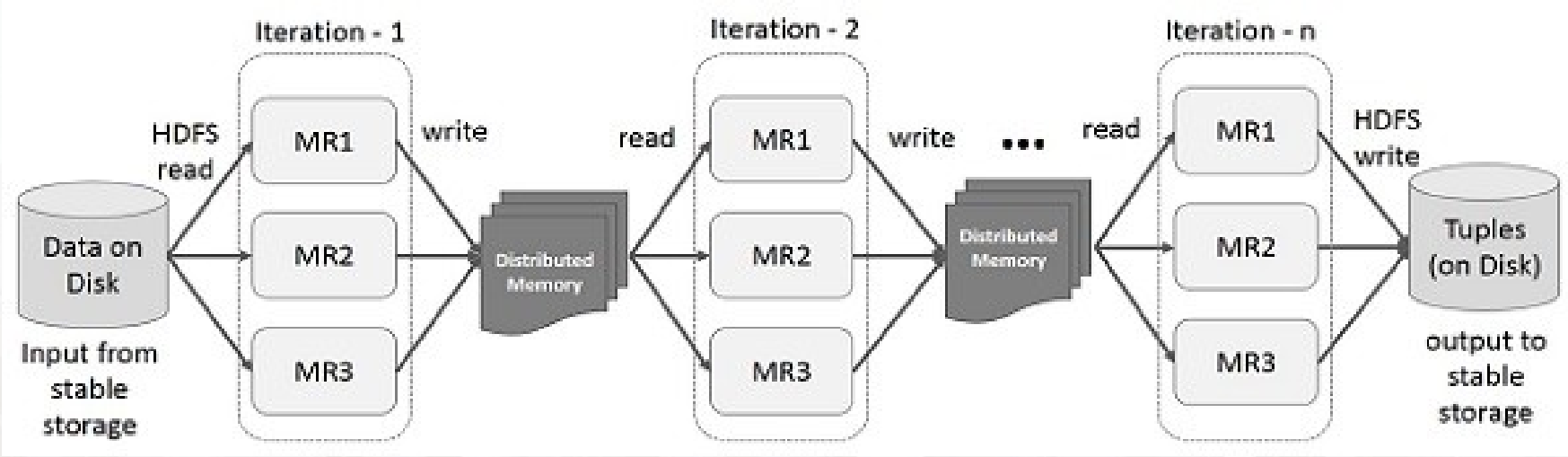


Usando Spark RDDs

- Compartilhamento de dados é lento em MapReduce
  - Replicação, serialização, I/O de disco
- Spark procura tratar essas questões
- Resilient Distributed Dataset
  - Ideia principal
  - Suporta “in memory processing computation”
- Armazena o estado da memória como objetos ao longo da execução dos jobs
  - Objeto é compartilhável entre os jobs
- Compartilhamento de memória é de 10 a 100 vezes mais rápido que rede e disco

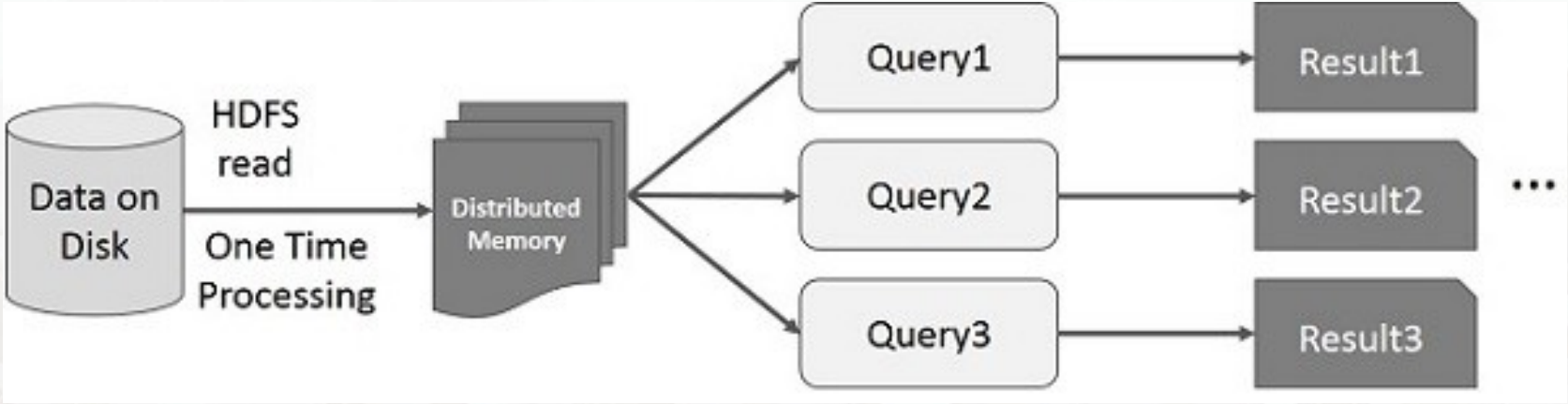
Operações iterativas em RDDs

Resultados intermediários armazenados em memória distribuída



Operações iterativas em RDDs

Se queries diferentes são executadas sobre o mesmo dataset repetidamente, esses dados podem ser mantidos em memória



Operações em RDDs

- Memória distribuída (RAM) pode não ser suficiente para armazenar resultados intermediários
  - Que serão eventualmente armazenados em disco
- A execução das operações em RDDs ocorrem somente quando a ação final é chamada
  - Lazy computation
- RDDs podem ser persistidos em memória
  - Podem ser também persistidos em disco e/ou replicados nos nós do cluster

Spark – Modelo de programação

- Programadores escrevem um ***driver program***
  - Implementa um fluxo de alto nível
- Duas abstrações principais para programação paralela
  - RDDs
    - E DataFrames e DataSets
  - Operações paralelas
- Suporta carregamento de dados de diversas fontes
  - Além de acesso a bancos de dados relacionais e NoSQL

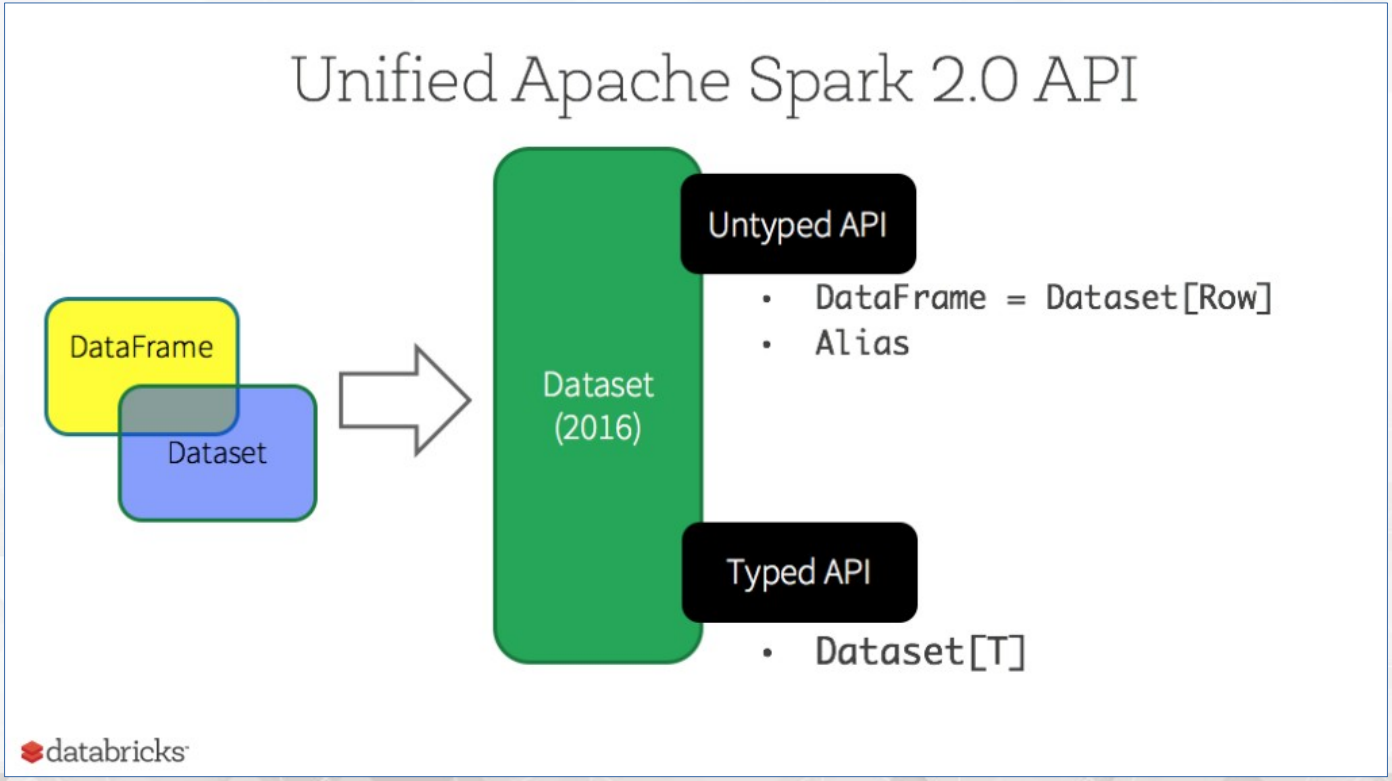
RDDs, DataFrames, DataSets

- RDD
  - API primária em Spark
  - Parecida com collections em Scala
  - Muito flexível
    - Unstructured data
- MAS...
  - Eventualmente, seus dados SÃO estruturados
    - XML, JSON, CSV
  - E você precisa de operações de alto nível
    - SQL

RDDs, DataFrames, DataSets

- DataFrames
  - Collection organizada em colunas
    - Determina estrutura
  - Fornece um API específica para um domínio específico
- DataSets
  - Spark 2.0
  - Strongly-typed API
    - Coleções de strongly-typed JVM objects
  - Integra DataFrame como DataSet[Row]

RDDs, DataFrames, DataSets



Spark – Configuração Local

- Mais adequado a sistemas baseados em Linux
- Passos:
  - Instalar JDK (8/11/17)
  - Instalar suporte a linguagem específica
    - Se necessário
    - Scala, Python
  - Baixar o pacote Spark (spark+hadoop)
  - Extrair o conteúdo do arquivo em um diretório
  - Verificar a instalação
    - Executar Spark shell
    - $SPARK_HOME/bin/spark-shell (ou pyspark, ou sparkR, ou spark-sql)
  - Usar a webconsole local
    - http://localhost:4040

Spark – Configuração Local

- Shells disponíveis
  - Acesso interativo
  - Exploração inicial
- Para programação
  - Em Java
    - Maven
    - Jars disponíveis na instalação
  - Em Python
    - PySpark instalável via pip install
    - Uso em notebooks ou IDEs

Exemplo – WordCount (em spark-shell)

```
# leitura de dados de um arquivo
texto = sc.textFile("/home/leandro/Documentos/datasets/2600-0.txt")

# quebrar todas as linhas em palavras
palavras = texto.flatMap(lambda line: line.split(" "))

# contar as ocorrencias de cada palavra
contadores = palavras.map(lambda palavra: (palavra, 1)).reduceByKey(lambda a,b:a+b)

# salvar os contadores para um diretorio
contadores.saveAsTextFile("/home/leandro/Documentos/datasets/contagem")
```

Exemplo – dados em CSV (em pyspark)

```
# carregar um CSV em um dataframe, usando o header e com o separador como ;
df = spark.read.csv("/home/leandro/datasets/employees/employees100.csv", header=True, sep=';')

# mostrar as primeiras linhas do dataframe
df.show()

# mostrar estatísticas sobre o dataframe
df.summary().show()

# filtrar por um campo
df_f = df.filter(df.gender == "F")
df_f.show()

df_m = df.where("gender = 'M'")
df_m.show()

# referenciar o dataframe como uma tabela relacional
df.createOrReplaceTempView("employees")

# executar uma consulta em SQL sobre a tabela
spark.sql("select * from employees where emp_no = 10002").show()
```

Obrigado

leandro@utfpr.edu.br

<lapti>