

Parte 2: Introdução ao Spark

- Introdução
- Spark em um ecossistema Big Data
- Primeiro contato com Spark

Desafios no processamento de dados em Big Data

- **MapReduce** possui algumas limitações.
 - Gargalos de performance
 - Suporte a **programação funcional** com linguagens não tão **verbosas**
- **Spark**: Uma ferramenta que veio para resolver esses problemas

O que é o Spark?

- **História do Spark:**

- Começou como um projeto de pesquisa no UC Berkeley AMPLab em 2009
- Se tornou open-source em 2010
- Se tornou um projeto Apache em 2013
- Versão atual 3.0
 - Lançada em 18/06/2020

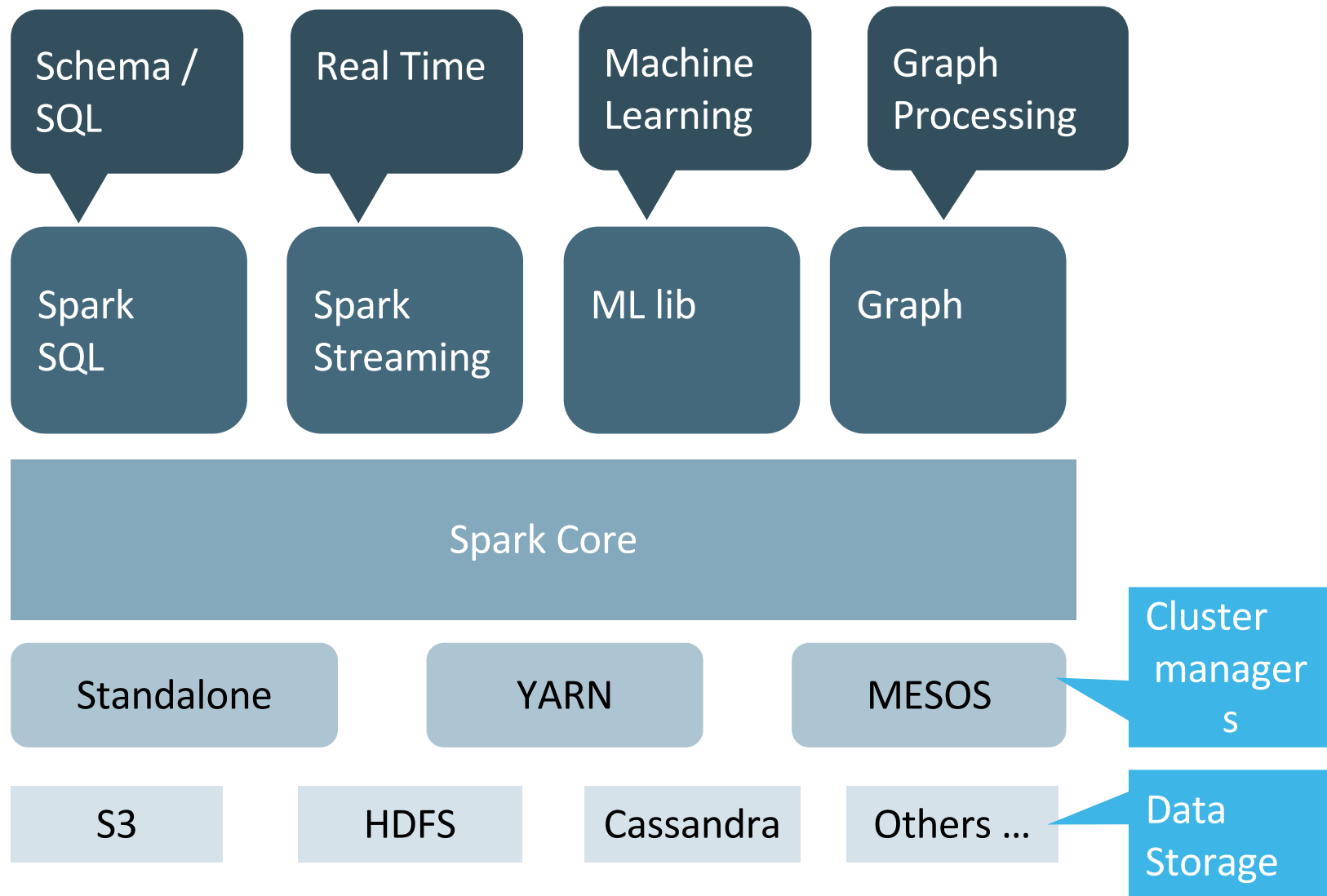
- É uma **ferramenta de computação baseada em cluster de forma distribuída**

- **Velocidade:** Processamento em memória, 100x mais rápido que o MR
 - Para operações em disco rígido é 10x mais rápido que o MR
- **De propósito geral:** MR, SQL, streaming, machine learning, analytics
- **Integrado:** Executa em clusters Hadoop/YARN, Mesos, standalone
- **Simples:** Scripts básicos em poucos minutos

Ecosystems Spark

- **Apache**: Distribuição open-source
- **Databricks**: Suporte e desenvolvimento do Spark
- Distribuições **Hadoop**(Hortonworks / Cloudera)
- Repositório **Spark packages** : Plug-ins criados pela comunidade
 - <http://spark-packages.org/>

Spark



Estrutura do Spark

- **Data Storage:** Componente de gerenciamento de fontes de armazenamento de dados.
- **Cluster Manager:** Responsável pelo gerenciamento dos nós do cluster
 - Trabalha com Mesos, YARN e Standalone
- **Spark Core:** Recurso de computação distribuída do Spark
- **Spark Components:** Componentes que suportam finalidades específicas dentro do Spark (Spark SQL, ML lib e etc)

Spark Core

- **RDD** (Resilient Distributed Dataset)
 - Coleção de dados distribuídos
 - Processados em paralelo
 - Criados a partir de N fontes de dados
- **Spark API**: Scala, Python, Java, and R
 - Responsável por abrir um canal de comunicação programática com o cluster

Interação com o Spark

- Modos de executar o spark:
 - Local
 - Cluster (Hadoop ou Standalone)
- **Spark shell** para programação interativa, utilizando o REPL do Scala
- **Spark API**: Interface de comunicação programática com os recursos do Spark

Spark Components

- **Spark SQL/DataFrames/Datasets:** Dados estruturados
 - Suporte para SQL e HQL (HiveQL)
 - Fontes de dados em arquivos ou bases de dados
- **Spark Streaming:** Streaming de dados em real-time
 - Baixa latência e alta taxa de transferência
- **ML/MLlib:** Machine Learning em escala
- **GraphX/GraphFrames:** Manipulação e computação em grafos de forma distribuída

Spark : Stack unificada

- Os componentes do Spark suportam **múltiplos modelos de programação**
 - Map / Reduce
 - Streaming / processamento real-time
 - SQL
 - Machine learning
- Todos os módulos são fortemente integrados
- Spark é independente!
 - Não depende de nenhum framework ou cluster para executar.

Otimizações do Spark para processamento rápido

- Múltiplas operações em um **Pipeline** de execução
 - Lazy evaluation
 - Executar um processo apenas quando o resultado dele é necessário.
- **Cachear resultados intermediários** de operações em memória
 - Provê o aumento significativo de performance.
 - Sem operações de I/O no HD.

Spark Use Cases

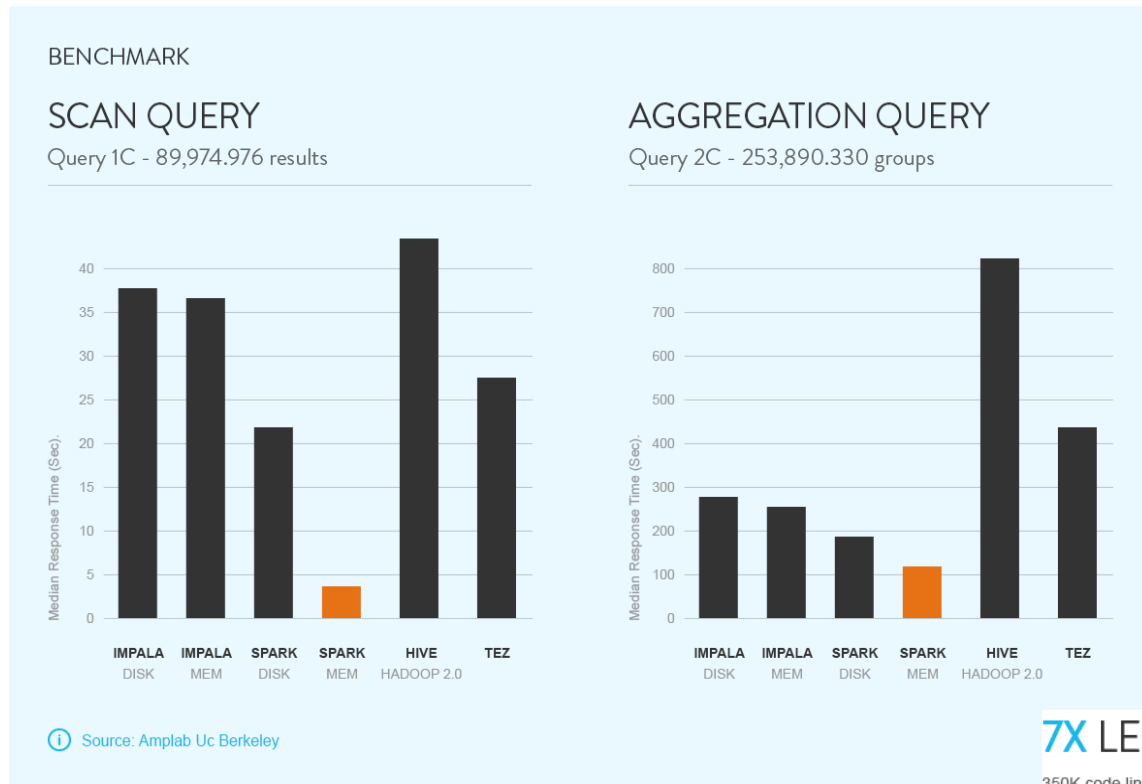
Data Scientist	Data Engineer
<ul style="list-style-type: none">• Exploração Ad-hoc• Construção de modelos	<ul style="list-style-type: none">• Processamento de dados em pipeline.
Shell para análises ad-hoc	Shell para teste e debug de scripts
Multi-linguagem	idem
Desenvolvimento interativo	idem
Stack unificada	idem

Exemplos de casos de uso do Spark

- **Facebook:** 60 TB+ em produção
 - Redução de tempo na execução de Jobs gerais. De 60 horas para 10 horas
- **Yahoo:** Novas personalizações
 - Um programa de 120 linhas escrito em Scala com MLlib substituiu 15.000 linhas de C++
 - 30 minutos para processar um data set de 100 milhões de registros
- **Netflix:** Analisar eventos a partir de Streaming
 - 450 bilhões de eventos por dia

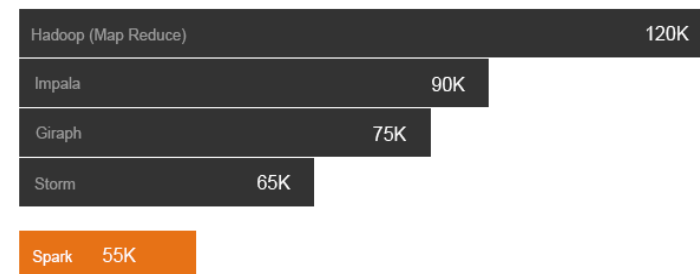
Parte 2.2: Spark em um ecossistema Big Data

Mais performance com um codebase menor



7X LESS CODE SIZE

350K code lines comparing to 55k.



Comparação com Map Reduce

Hadoop e MapReduce	Spark
Armazenamento e Processamento Distribuído	Apenas processamento distribuído
Apenas a ferramenta de MapReduce	Processamento generalizado (Streaming, ML, SQL e etc)
Dado apenas no HD	Dados no HD ou Memória
Não é o ideal para carga de dados com muita iteração	Ótimo para cargas de dados com muita iteração (machine learning, etc.)
Processamento em lotes	<ul style="list-style-type: none">- 2x — 10x mais rápido para operações com dados no HD- 100x faster para operações com dados em memória
Stack 100% em Java, sem suporte para outras linguagens	Compact code Java, Python, Scala, R supported
Não possui um shell interativo	Shell interativo para exploração, debug e testes.

Spark vs. MapReduce

- Spark é mais **simples** que MapReduce
 - Menos código com suporte para outras linguagens
- **Amigável** para analistas e cientistas
 - Shell interativo
 - Testes, análises e debugs.
 - Componentes especializados (Grafos, Machine Learning, SQL e etc...)
- Suporte para outras linguagens na API
 - Java, Scala, Python, R

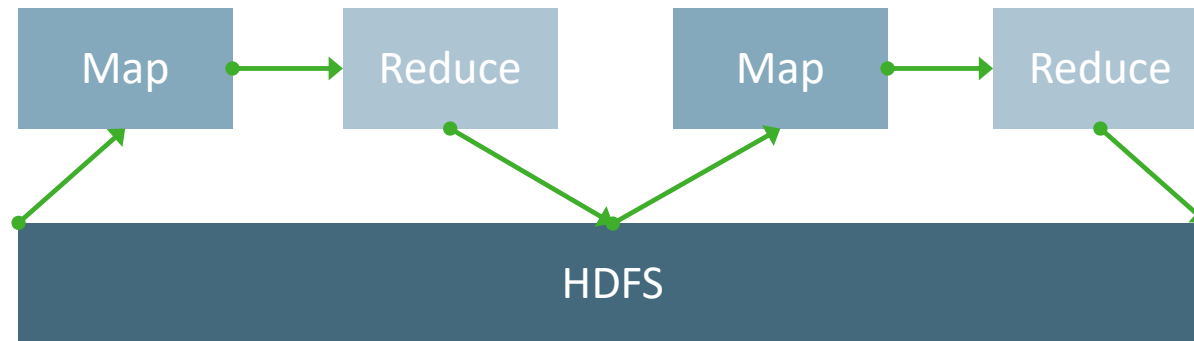
Spark: Recorde de ordenação de dados em larga escala

- Ordenou 100TB em 23 min com 206 nós

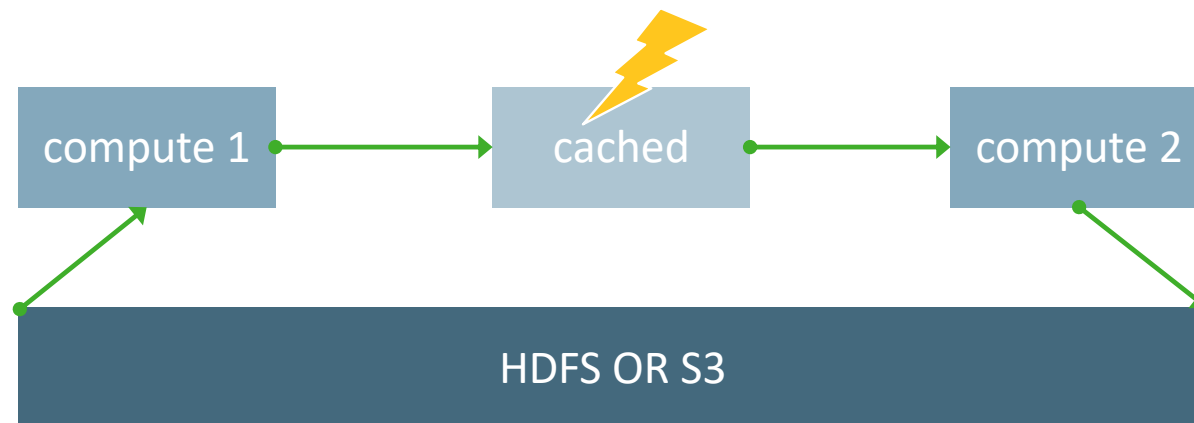
	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Sort rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min

Spark: Melhor escolha para processamento iterativo

Hadoop Map Reduce

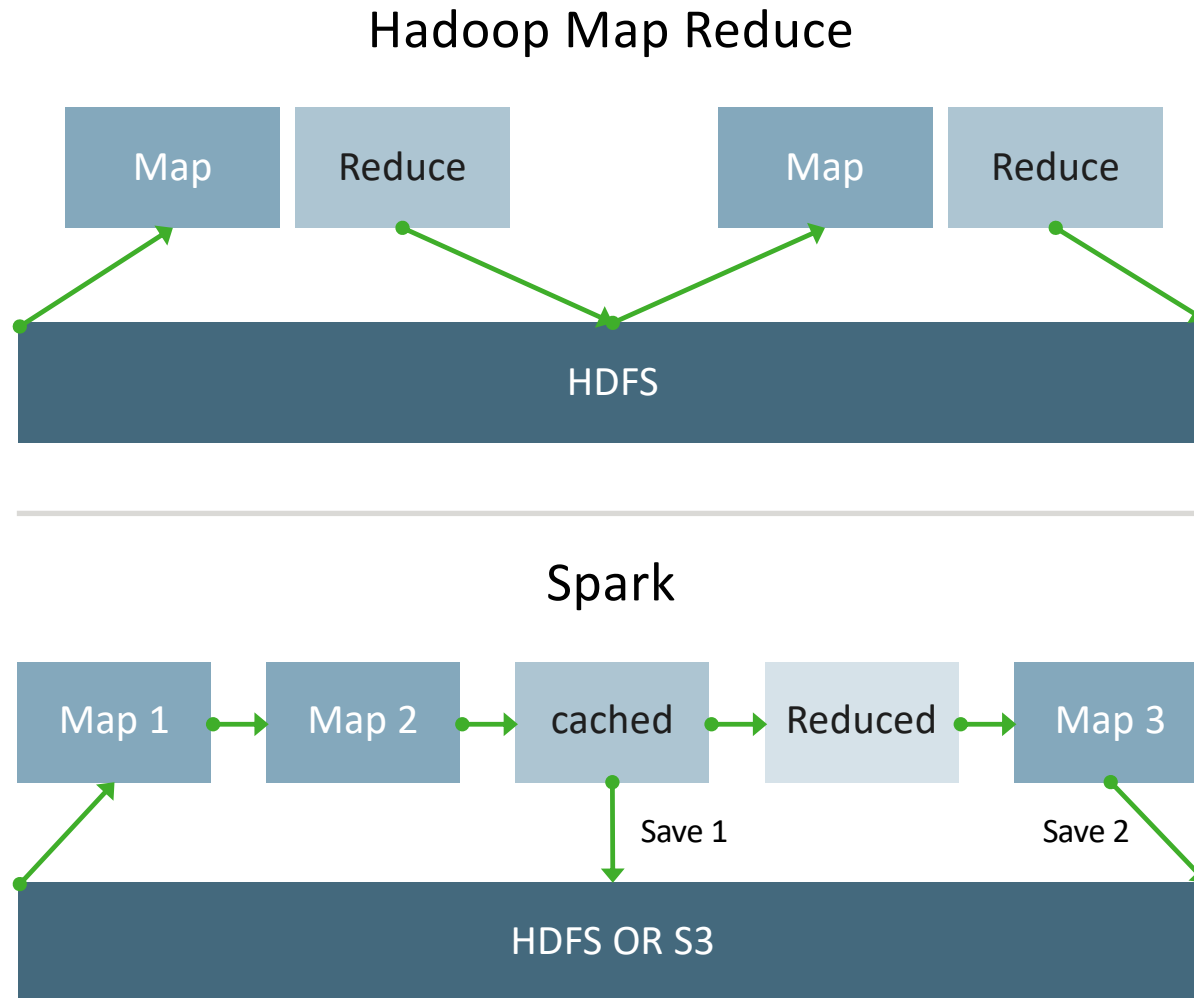


Hadoop Map Reduce



Spark: Um modelo de programação mais genérico

- Reduz a dependência de escritas no HD



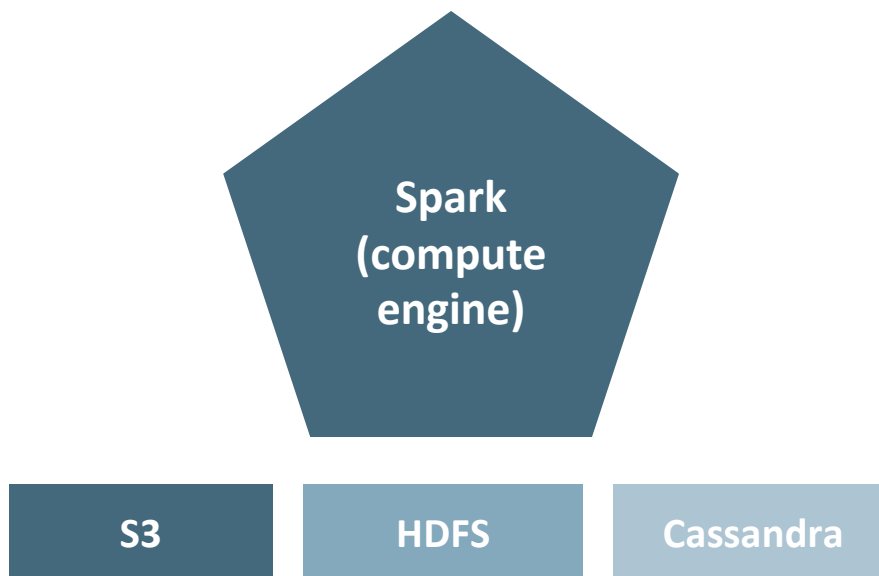
Hadoop vs. Spark

- A stack unificada do Spark suporta todas as necessidades

Use Case	Hadoop	Spark
Armazenamento	HDFS	<ul style="list-style-type: none">• HDFS / S3 / Cassandra• Tachyon (armazenamento distribuído em memória)
Cluster Manager	YARN	YARN ou Mesos
Batch processing	MapReduce (Java, Pig, Hive)	Spark MR
SQL querying	Hive	Spark SQL (suporta também Hive/HQL)
Stream / Real Time	Storm	Spark Streaming
Machine Learning	Mahout	Spark MLlib
Visualizações real-time	NoSQL (HBase, Cassandra, etc)	Spark não possui um componente Mas realiza queries em bancos NoSQL

Spark substitui o Hadoop?

- Spark atualmente pode se hospedar no ecossistema Hadoop / YARN
 - Pode ser visto como um modelo genérico de MapReduce
- O Spark atende muito bem quando o tamanho da carga dos dados cabe em memória (algumas centenas de gigas)
 - E ainda suporta o processamento utilizando o HD



Mais casos de uso de Spark

- Cluster
 - 8000 nós
 - 400 TB+ de dados
 - @ Tencent
- Single job
 - 1 PB
 - Processamento de imagens @ Alibaba
- Streaming
 - 1 TB por hora
 - Analise de imagens médicas @ Jenelia farm
- Recorde mundial de ordenação



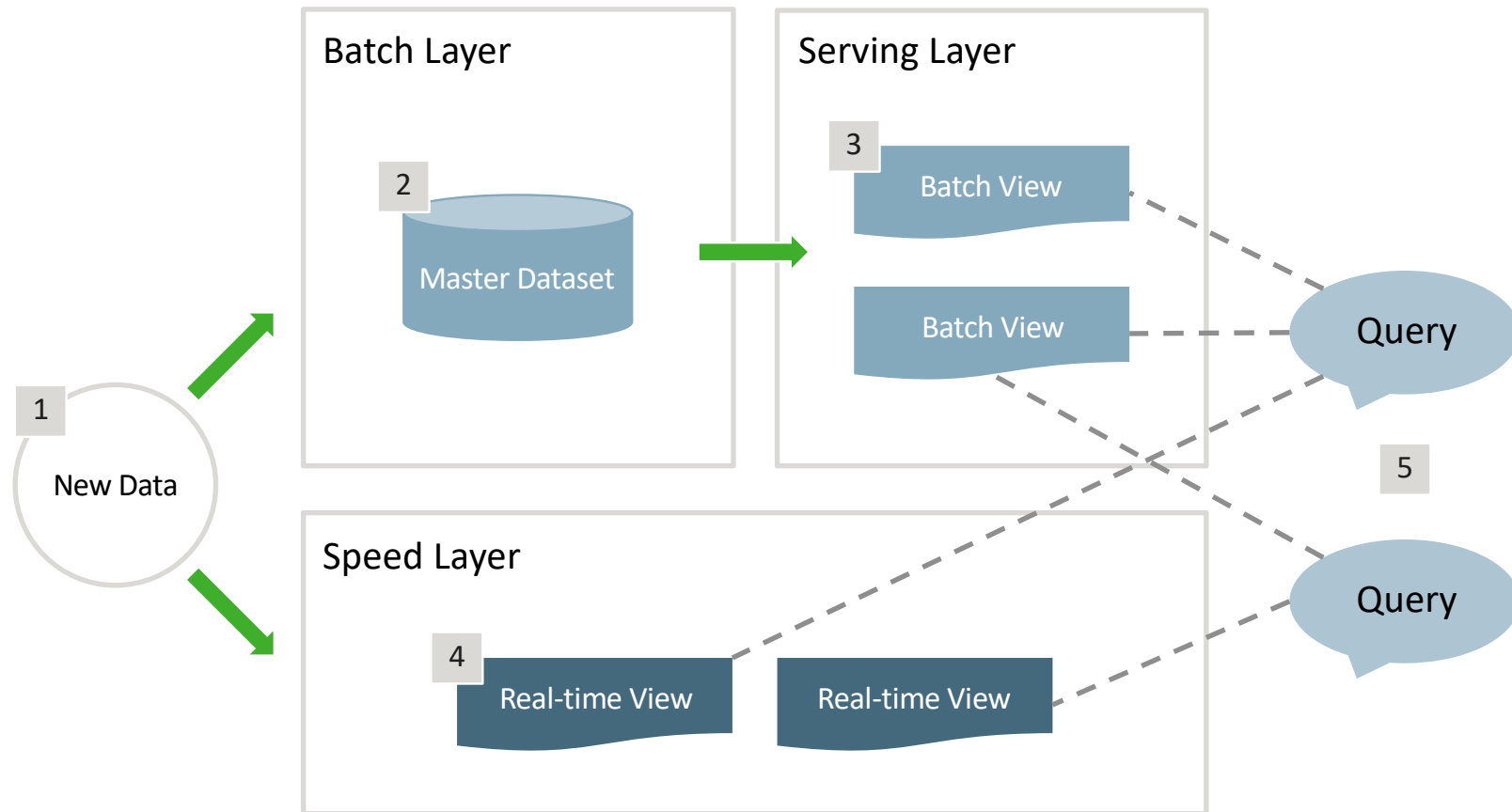
Spark New and Noteworthy

- Data Science
 - DataFrames / Datasets (inspired by R / Python)
 - Machine Learning pipelines (inspired by SciKit Learn)
 - R language support (!)
- Platform
 - Unified access across multiple datasources (HDFS/noSQL, etc.)
 - Spark packages : community libraries
 - <http://spark-packages.org>
- Core / Engine
 - **Tungsten**: Off-heap memory
 - **Succinct**: Direct queries on compressed data
 - Visualization / debugging tools

Spark para Arquitetura Lambda (LA)

- LA: Padrão de Design para infraestrutura de dados
 - Atender necessidades do negócio de forma escalável.
 - **Tolerância a falhas** de hardware e humanos.
- A abordagem em camadas (layers) atende todas as necessidades
 - **Batch Layer**: Gerencia o dataset master
 - Imutável, incrementável e com dados “raw”
 - Prepara visualizações dos dados “batch”
 - **Serving Layer**: Indexa as visualizações dos dados batch para atender queries com baixa latência.
 - **Speed Layer**: Atende as requisições que precisam ser processadas com uma baixa latência.
 - Dados recentes que usam algoritmos rápidos e incrementais

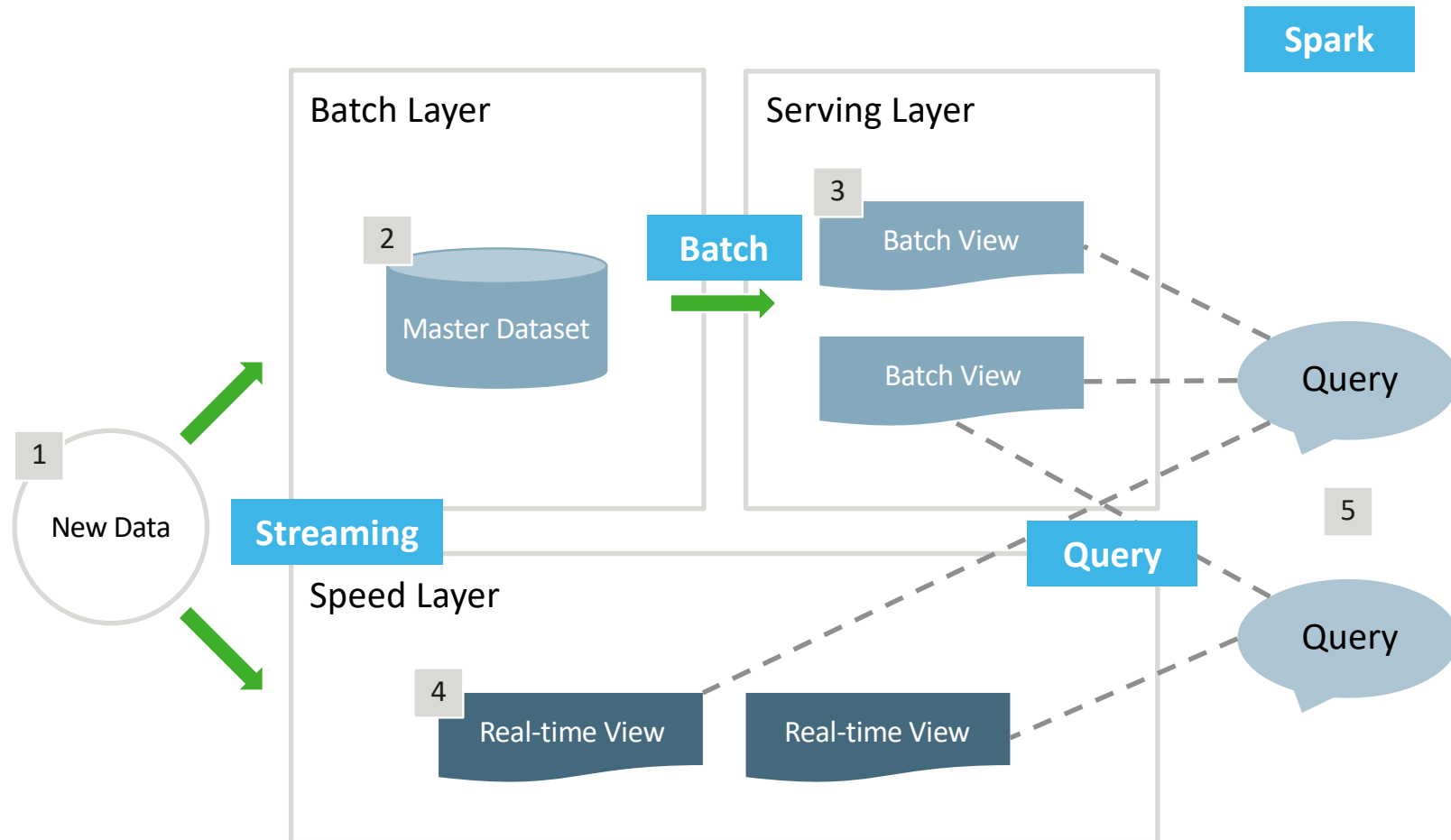
Arquitetura Lambda



Spark e LA

- LA necessita de componentes de processamento real-time e em lotes (batch)
 - **Spark** atende ambas as necessidades em um único sistema.
- Usar a funcionalidade “core” do Spark para processar os dados (batch)
 - Exemplo: Extração e armazenamento de eventos de logs condicionadas pelo tipo de Log (Erro, Sucesso, Info e etc).
- Criar agregações dos dados e gerar bases “view” indexadas para consultas rápidas (batch)
 - Exemplo: Numero de eventos por tipo de Log
- Usar **Spark streaming** para suportar visualizações e processamento rápido dos dados (real-time)

Spark + Arquitetura Lambda



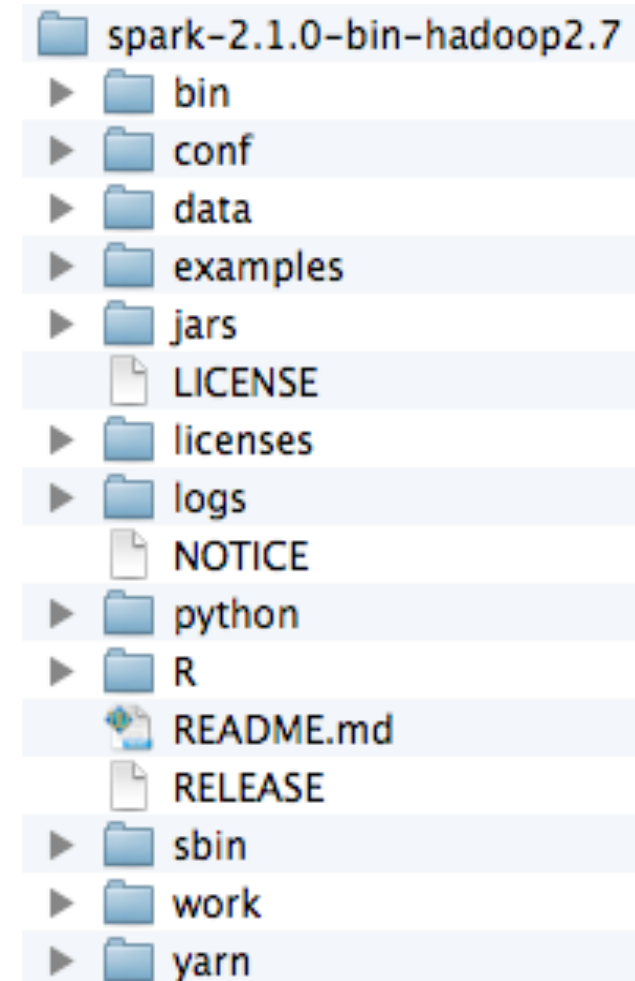
Parte 2.3: Primeiro contato com Spark

Apache Spark / Download

- Apache project (open-source)
 - <http://spark.apache.org/>
- Escrito em Scala
- Tarballs de binários estão disponíveis no website do projeto
 - Possui as bibliotecas do Spark e Scala, shell interativo e scripts de execução
 - Spark com distribuição Hadoop é opcional.
- Pode ser usado em alguns modelos de cluster
 - Standalone:
 - Usando um gerenciador de cluster externo
 - Hadoop/YARN and Mesos

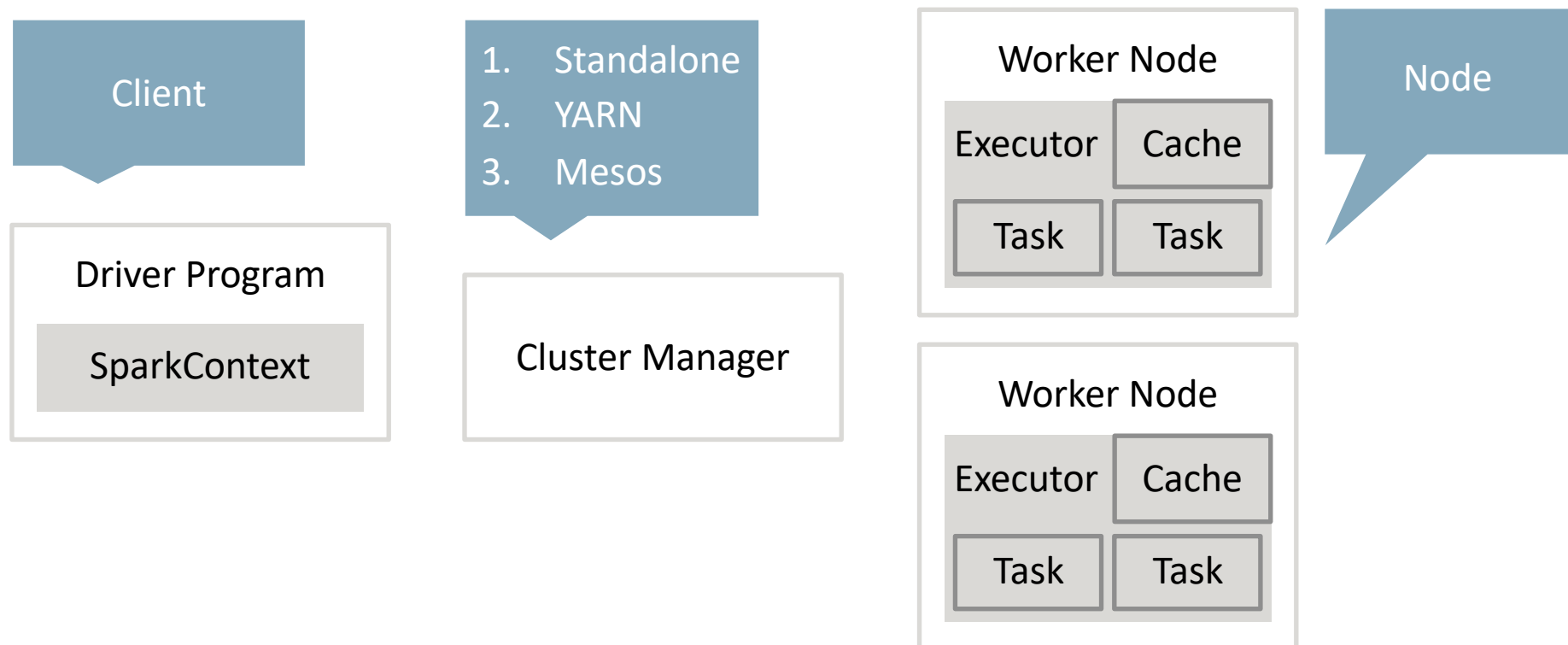
Requisitos do Sistema e instalação

- Java 7+ (Spark 1.5+), Java 8+ (Spark 2.0)
- Descompactar Spark tarball.
- Estrutura de arquivos:
 - **bin**: executáveis (Spark/Python shells, utilities, etc.)
 - **conf**: templates de configuração (e.g. spark-env.sh.template)
 - **data**: Dados de exemplo
 - **ec2**: Scripts para execução em ec2
 - **examples**: Programas de exemplo (Scala, Python, Java)
 - **lib**: Arquivos jar
 - **logs**: Arquivos de log
 - **python**: Códigos Python
 - **R**: SparkR (interface em R)
 - **sbin**: Scripts shell



Arquitetura de execução

- **Worker nodes** executam as tasks
- **Cluster manager** gerenciam os worker nodes
- **Driver program** inicia as tasks e envia os códigos para elas



Executando o Spark

- Usando o **gerenciador de cluster “standalone”** para iniciar o cluster
 - Iniciar todos os nós em uma única máquina
 - `<spark>/sbin/start-all.sh` inicia um cluster simples (um master um worker) — `stop-all.sh` para todas elas.
 - Pode precisar de algumas configurações para acesso ssh — mesmo executando o cluster em uma única máquina
- Usando algum shell interativo
 - Funciona com o shell interativo do Scala e do Python
 - **Scala:** `<spark>/bin/spark-shell`
 - **Python:** `<spark>/bin/pyspark`
 - Por padrão, executam uma instância **embutida** do Spark
 - Mas é possível se conectar com um cluster.

Gerenciador de cluster “Standalone”

- Cluster simples de gerenciamento próprio
 - Sem a necessidade de um gerenciador externo (e.g. YARN)
 - Configuração e execução simples.
- Inicialização simples com os scripts
 - `<spark>/sbin/start-all.sh` inicia o master e o(s) worker(s)
 - Padrão de um master e um worker na máquina local
 - Serve uma página web no endereço: `hostname:8080`
- Outros scripts de inicialização e parada
 - `start-master.sh`: Inicia apenas o master na máquina local
 - `start-slaves.sh`: Inicia os worker nas máquinas listadas nos arquivos de configuração.
 - `stop-*.sh`: Variações para parar os tipos de nós

Configurações no Spark

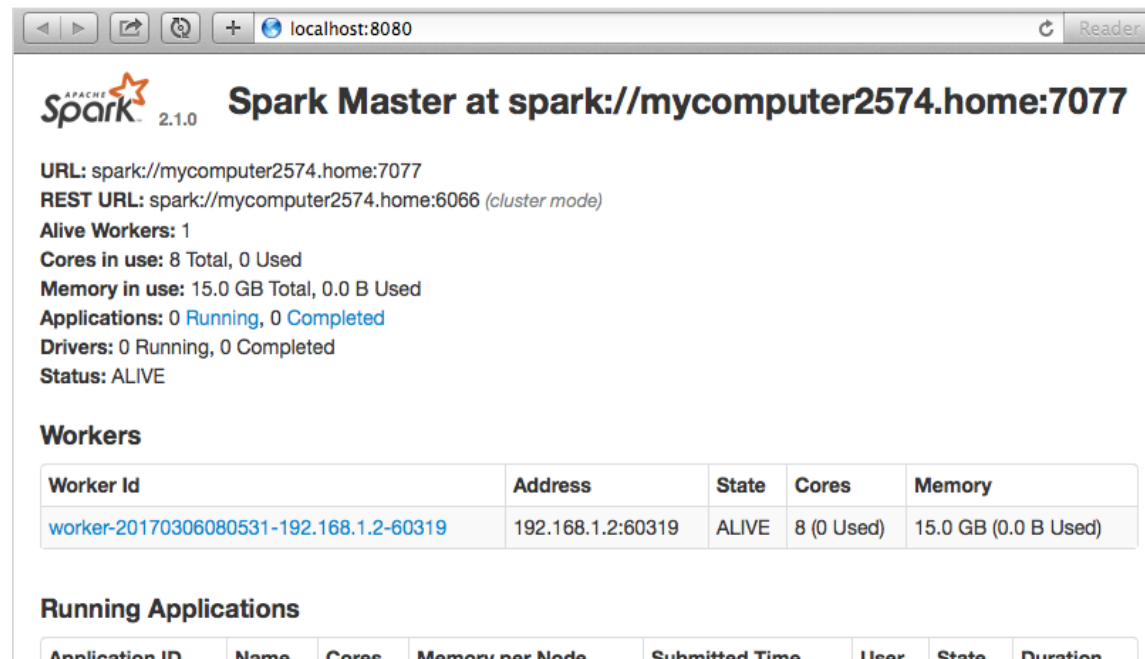
- *conf/log4j.properties*: Configuração de logs
 - log4j.properties.template é um exemplo para tomar como base
- *conf/slaves*: Listas dos endereços dos workers
 - O valor padrão é localhost
 - slaves.template é um exemplo para tomar como base
- *conf/spark-env.sh*: Configurações gerais
 - Recursos de execução, endereços e etc.
 - *spark-env.sh.template* é um exemplo para tomar como base
 - Exemplos de parâmetros para uma configuração “Standalone”:
 - **SPARK_MASTER_HOST / PORT / WEBUI_PORT** : Parâmetros da master
 - **SPARK_WORKER_CORES / INSTANCES** : Parâmetros dos workers
 - Entre outros

Iniciando o Spark e acessando a WebView

```
$ ./sbin/start-all.sh
```

```
starting org.apache.spark.deploy.master.Master, logging to  
/home/student/spark/logs/spark-student-org.apache.spark.deploy.master.Master-  
1-localhost.localdomain.out
```

```
localhost: starting org.apache.spark.deploy.worker.Worker, logging to  
/home/student/spark/logs/spark-student-org.apache.spark.deploy.worker.Worker-  
1-localhost.localdomain.out
```



Spark Master at spark://mycomputer2574.home:7077

URL: spark://mycomputer2574.home:7077
REST URL: spark://mycomputer2574.home:6066 (cluster mode)
Alive Workers: 1
Cores in use: 8 Total, 0 Used
Memory in use: 15.0 GB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20170306080531-192.168.1.2-60319	192.168.1.2:60319	ALIVE	8 (0 Used)	15.0 GB (0.0 B Used)

Running Applications

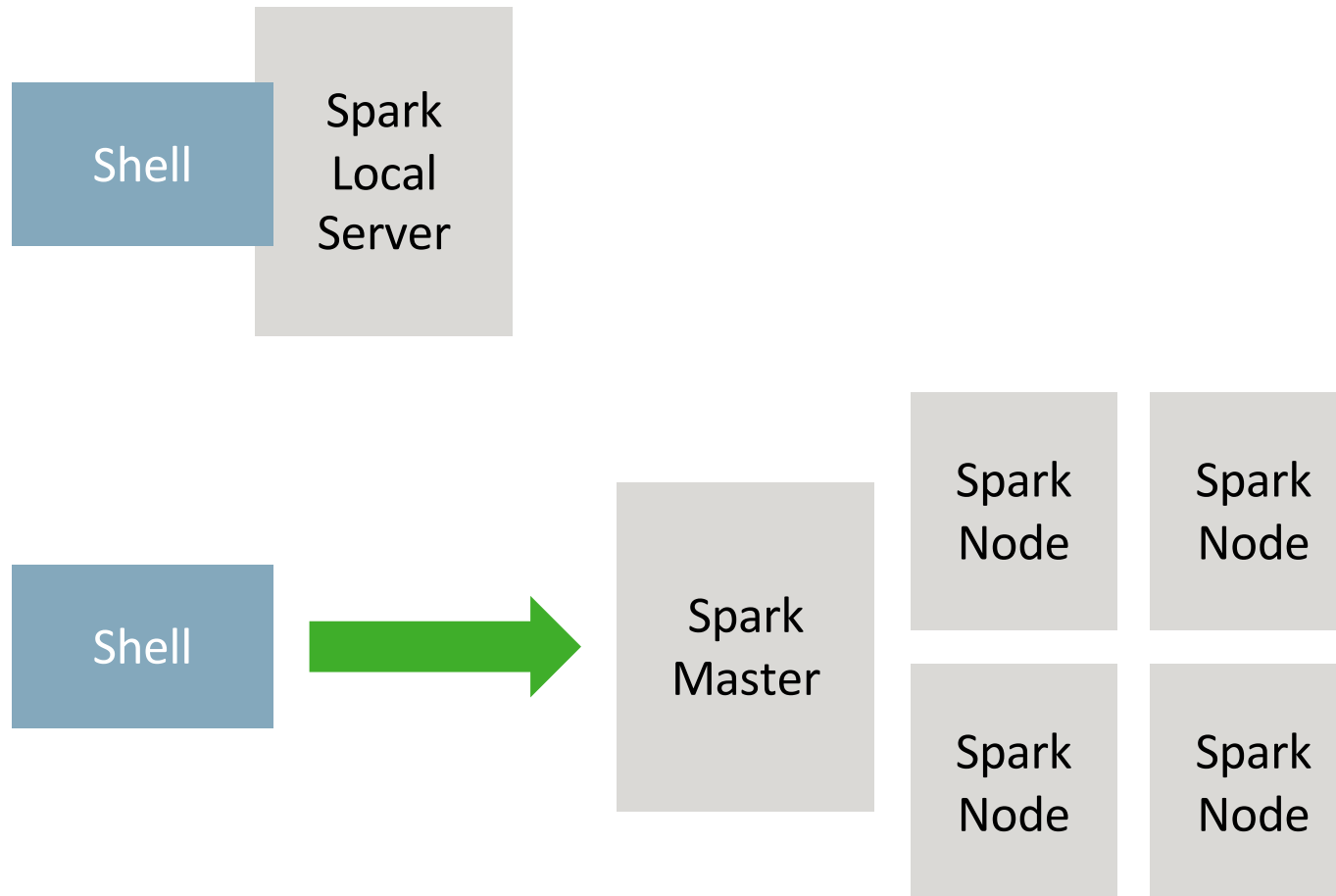
Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

Lab 2.1: Primeiro contato com Spark

Shells do Spark

- Shells **interativos** atendem execuções ad-hoc
 - Podem se conectar em grandes clusters para terem uma execução rápida
- **Scala** shell: `bin/spark-shell`
- **Python** shell: `bin/pyspark`
- Shells podem executar nos modos:
 - **Local**: Utiliza um cluster embutido
 - Não é o mesmo cluster do gerenciador “Standalone”
 - **Cluster**: Conecta em um cluster utilizando a URL do nó master

Modos de execução do Shell



Exemplos de inicialização do Spark Shell

- **spark-shell**: Inicialização utilizando o cluster embutido, com uma única thread.
 - Modo padrão
- **spark-shell --master local[4]**: Inicialização utilizando o cluster embutido, com 4 threads.
 - Geralmente limitado pela quantidade de cores que você possui
- **spark-shell --master spark://my-computer.home:7077**
 - Conectar em um cluster a partir da URL do nó master
- **spark-shell --help**: Mostrar sessão de ajuda

Outras opções do Spark Shell

- Opções de **Classpath**
 - **--jars**: Lista de jars locais para incluir na execução
 - **--packages**: Lista de coordenadas maven de pacotes jar para incluir na execução
 - **--exclude-packages**: Lista de groupId:artifactId para excluir enquanto resolve as dependências do --packages
- Opções de **Configuração**
 - **--conf PROP=VALUE**: Define único par de propriedade/valor
 - **--properties-file FILE**: Define propriedades a partir de um arquivo
- Opções da **JVM**:
 - **--driver-memory MEM**: Define a memória alocada para o driver
 - Entre outras.
- Para mais informações, utilize o **--help**

Opções de Master URL

Master URL	Details
local	O Spark executa localmente considerando todas as thread disponível
local[k]	O Spark executa em uma única instância com K threads worker – que deve ser menor ou igual a quantidade de cores que você possui.
local[*]	Quando você não sabe a quantidade de cores que você possui, pode-se usar o “*” que representa a quantidade total.
spark://HOST:PORT	Conectar em um cluster Standalone. A porta padrão é a 7077.
mesos://HOST:PORT	Conectar em um cluster Mesos. A porta padrão é a 5050.
yarn-client	Conectar no YARN no modo client.
yarn-cluster	Conecta no YARN no modo cluster.

Iniciando o Spark Shell no modo Local

```
$ bin/spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
// WARN logging messages omitted ...
Spark context Web UI available at http://192.168.1.2:4040
Spark context available as 'sc' (master = local[*], app id = local-
1488813130684).
Spark session available as 'spark'.
Welcome to

      /_/_/ _ _ _ _ _/_/_/
     /_/_/ _/_/ _/_/ _/_/ _/_/
    /_/_/ _/_/ _/_/ _/_/ _/_/
   /_/_/ _/_/ _/_/ _/_/ _/_/
  /_/_/ _/_/ _/_/ _/_/ _/_/
 /_/_/ _/_/ _/_/ _/_/ _/_/
/_/_/ _/_/ _/_/ _/_/ _/_/

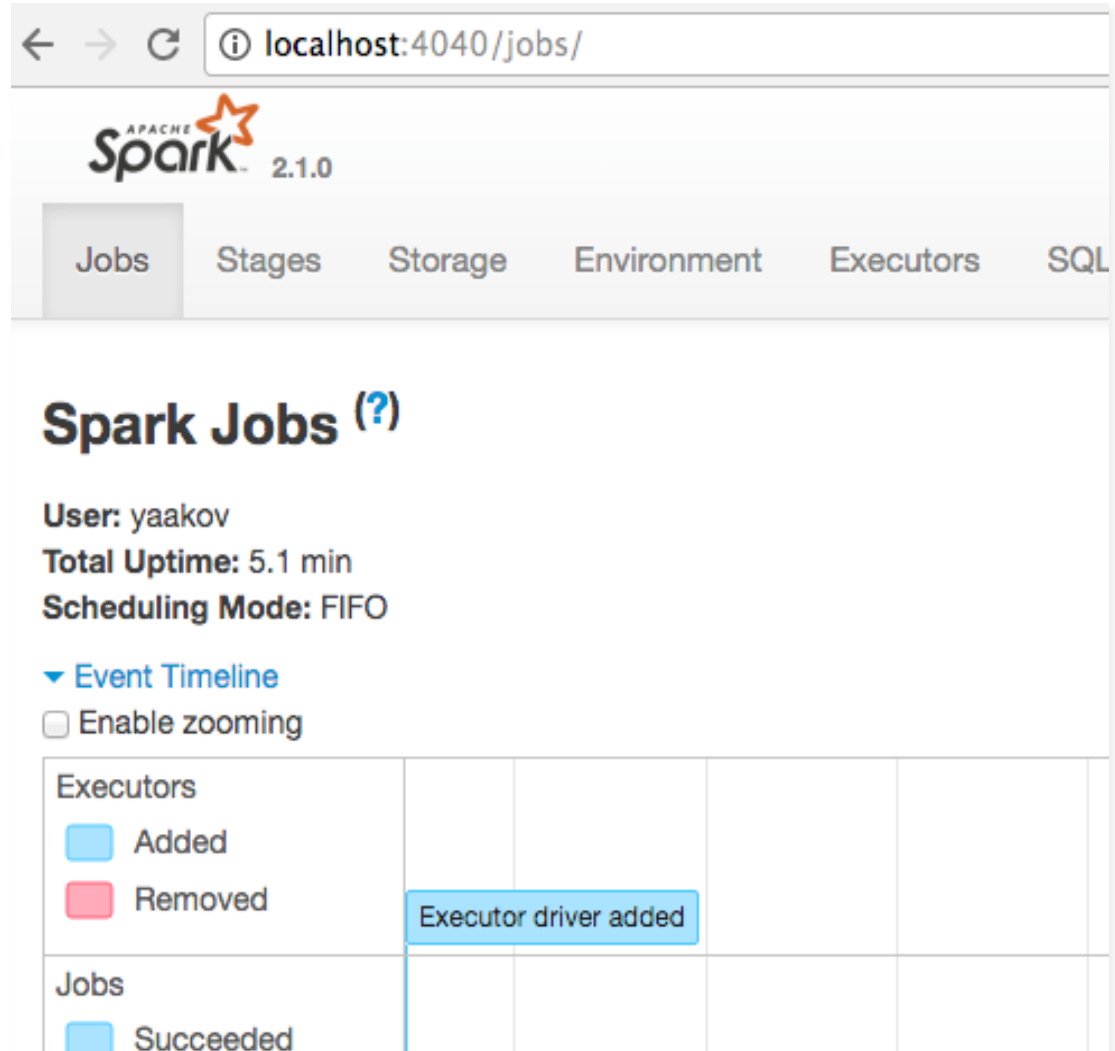
version 2.1.0

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_45)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Web UI do Spark Shell

- No endereço **<shell-host>:4040**
 - O número da porta é configurável
 - A UI é integrada ao SparkContext
 - Toda instância do SparkContext inicializa a Web UI
 - Serve informações sobre o driver do programa e sobre os jobs do spark



The screenshot shows the Apache Spark 2.1.0 Web UI. The browser address bar displays 'localhost:4040/jobs/'. The page features a navigation bar with tabs: 'Jobs' (selected), 'Stages', 'Storage', 'Environment', 'Executors', and 'SQL'. Below the navigation bar, the title 'Spark Jobs (?)' is displayed. The user 'yaakov' is logged in, and the total uptime is 5.1 minutes. The scheduling mode is FIFO. An 'Event Timeline' section is visible, with a checkbox for 'Enable zooming'. Below this, there are two main sections: 'Executors' and 'Jobs'. The 'Executors' section includes a legend with 'Added' (blue square) and 'Removed' (red square). A blue box labeled 'Executor driver added' is positioned over the first cell of the 'Executors' table. The 'Jobs' section includes a legend with 'Succeeded' (blue square).

Executors				
Added				
Removed				

Jobs				
Succeeded				

Abas do Web UI

- **Jobs:** Status de todos os jobs da Aplicação Spark (SparkContext)
 - Para os Jobs que estão sendo executados ou para os que estão agendados.
 - Pode-se obter informações específicas de cada Job.
- **Stages:** Estado atual de todos os estágios dos Jobs
- **Storage:** Informações sobre uso dos recursos de armazenamento (Memória e Disco)
- **Environment:** Informações sobre propriedades gerais, classpath, execução e etc.
- **Executor:** Processamento e armazenamento de cada executor
 - Pode-se obter informações granulares de cada executor (thread dump)
- **SQL:** Execuções de queries SQL
 - Pode-se obter detalhes de resultados e execuções das queries

SparkContext — Primeiro contato

- O acesso ao recursos do Spark é feito por meio do SparkContext
 - Representa a conexão com o cluster
 - Pré criado nos Shells de execução, armazenado na variável **sc**

```
> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@2c01a0dd

> sc.[tab] // Usando o tab para ver as possíveis operações
accumulable      accumulableCollection accumulator      addFile      addJar
addSparkListener appName      applicationId  asInstanceOf
binaryFiles
binaryRecords    broadcast      cancelAllJobs  cancelJobGroup
clearCallSite
clearFiles        clearJars      clearJobGroup  defaultMinPartitions
defaultMinSplits
... Remaining detail omitted

> sc.isLocal
res1: Boolean = true

> sc.master
res2: String = local[*]
```

RDD—Primeiro contato

- **Resilient Distributed Dataset:** Coleção distribuída do Spark
 - **Abstração de dados centrais** da ferramenta de processamento.
 - Operações em RDD's são paralelizadas entre o cluster
 - APIs de níveis altos (e.g. Dataset) não precisam do uso de RDD's
- Mais detalhes a frente.

Exemplo de usabilidade de RDD

- Ler um arquivo
- Obter uma amostra dos dados do arquivo
- Filtrar dados do arquivo

```
> val myfile = sc.textFile("README.md") # Cria uma RDD a partir do conteúdo do arquivo
myfile: org.apache.spark.rdd.RDD[String] = README.md MapPartitionsRDD[1] at textFile at
<console>:27

> val first = myfile.first()
first: String = # Apache Spark

> val all = myfile.collect()
all: Array[String] = Array(# Apache Spark, "", Spark is a fast and general cluster comp
# ... Remaining detail omitted

> val scalaLines = myfile.filter(line => line.contains("Scala"))
scalaLines: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at
<console>:29

> scalaLines.collect()
res1: Array[String] = Array(high-level APIs in Scala, Java, and Python, and an optimized
engine that, ## Interactive Scala Shell, The easiest way to start using Spark is through
the Scala shell:)
```


Lab 2.2: Primeiro contato com o Spark Shell

Revisão

- O que é Spark, e quais são as suas competências?
- Como o Spark se compara ao Hadoop / MapReduce?

Resumo

- Spark é uma ferramenta de código aberto para processamento de dados em cluster.
 - **Core**: Processamento distribuído de grandes quantidades de dados.
 - Componentes adicionais:
 - **Spark SQL**: Dados estruturados
 - **Spark Streaming**: Streaming de dados em tempo real
 - **ML Lib**: Machine learning
 - **GraphX**: Computação em grafos
 - Rápido, flexível e com codebase de desenvolvimento relativamente simples
 - Suporta outras linguagens (Scala, Java, Python, R e etc)
 - Projeto Apache suportado pela Databricks

Resumo

- Clusters executam no modelo “Standalone” ou gerenciados por uma ferramenta externa (Mesos, Yarn)
 - Instalação simples pelo pacote de binários.
 - Incluso em algumas distribuições do ecossistema Hadoop (Cloudera/Hortonworks)
- O processamento do Spark executa de forma distribuída em nós.
- O **Spark Shell** suporta operações ad-hoc e interativas.
 - Usando Scala ou Python
 - Um **SparkContext** serve a conexão com o cluster
- **Resilient Distributed Datasets** (RDDs) representam os dados no cluster
 - Coleção distribuída dos dados
 - Processados em paralelo

Resumo

- Spark supera muitas limitações do MapReduce (Hadoop)
 - Atende as atuais necessidades de Big Data
 - Rápido e simples para desenvolver
 - Stack unificada
- Crescimento acelerado no mundo de Big Data
 - Adoção cada vez mais rápida
 - Comunidade ativa e dedicada
 - Não é um ecossistema tão maduro quanto o Hadoop
 - e.g. soluções de gerenciamento e monitoramento ainda não são tão robustas

Parte 3: Conceitos de RDD

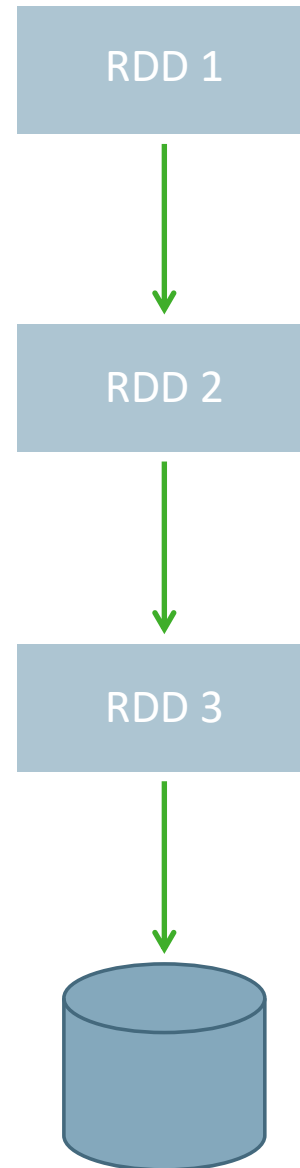
- Introdução
- Trabalhando com RDDs

O que é um RDD?

- **Resilient Distributed Dataset: Abstração de dados do Spark Core**
 - **Coleção distribuída** de elementos
 - **Particionado** — preferencialmente **entre diferentes nós**
 - **Imutável**
 - Operações executam **em paralelo** nas partições
 - **Resiliente**: Tem a habilidade de se recuperar caso haja a perda de alguma partição
 - **Eficiente** — Re-processados, não armazenados.
- **Dois tipos de operações**
 - **Transformação**: Lazy operation que cria um novo RDD
 - e.g. `map()`, `filter()`
 - **Ação**: Retorna um resultado ou o armazena em algum lugar
 - e.g. `take()`, `save()`

Ciclo de vida de um RDD

- RDD é **criado**:
 - Uma carga de dados externa
 - Uma distribuição de dados locais
- RDD é **transformado**:
 - e.g. filtrar elementos
 - Resultado: **Um novo RDD**
 - Muitas vezes passa por uma sequência de transformações
- Dados são eventualmente **extraídos**:
 - Por uma **ação** em um RDD
 - e.g. salvar os dados



Create: Read a log file
(e.g. from HDFS)
`sc.textFile("server.log")`

Filter: Keep lines
starting with "ERROR"

Filter: Keep lines
containing "mysql"

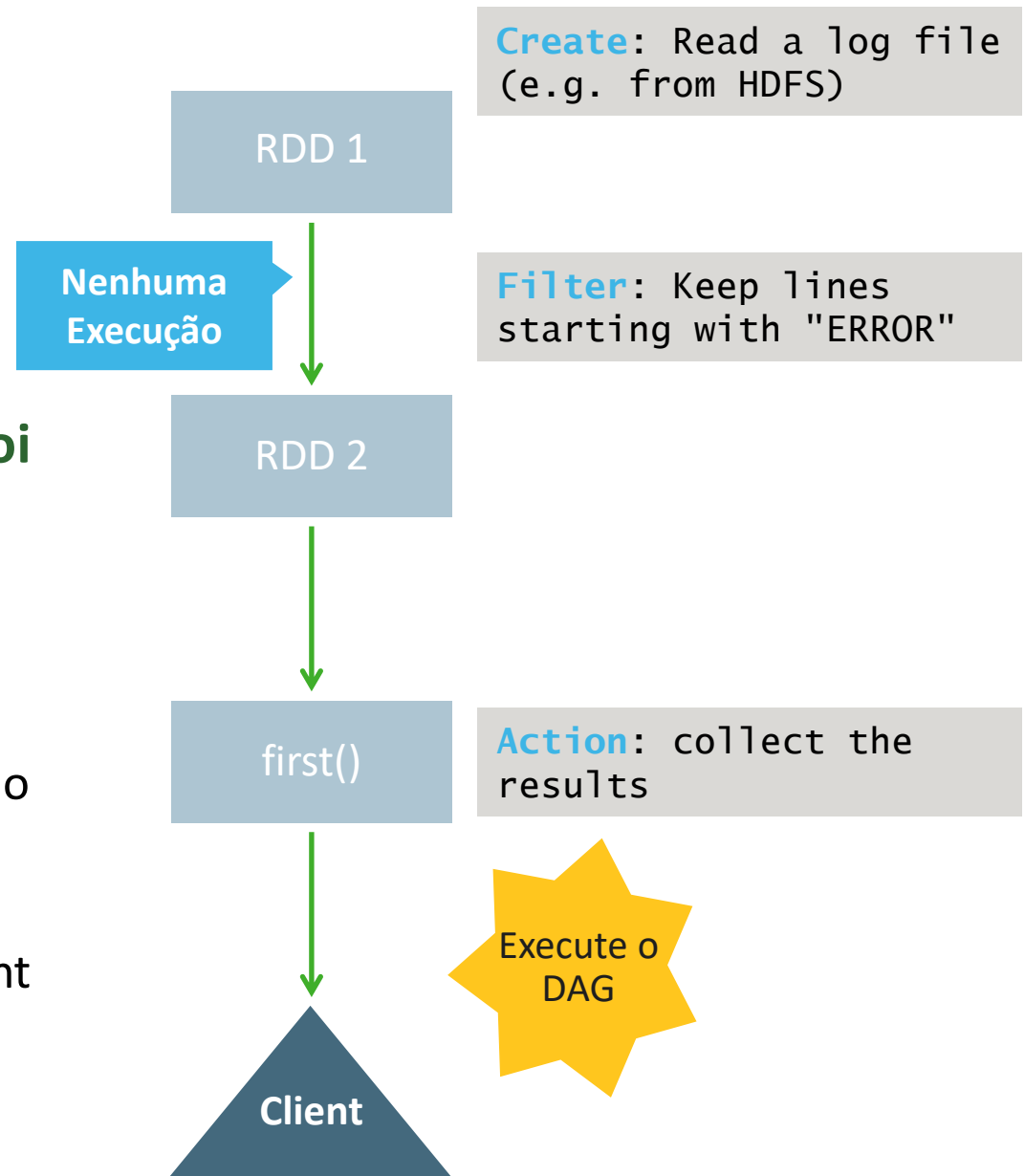
Action: Write result to
storage

Todas as transformações são “Lazy”

- O Spark não **não processa os resultados imediatamente**
 - Transformações são armazenadas como um grafo (DAG) de uma RDD base
- O DAG é executado quando uma ação acontece
 - Quando ele precisa obter dados
- Permite que o Spark:
 - **Otimize** os cálculos necessários.
 - **Recupere com eficiência** os RDDs no caso de falha de nó

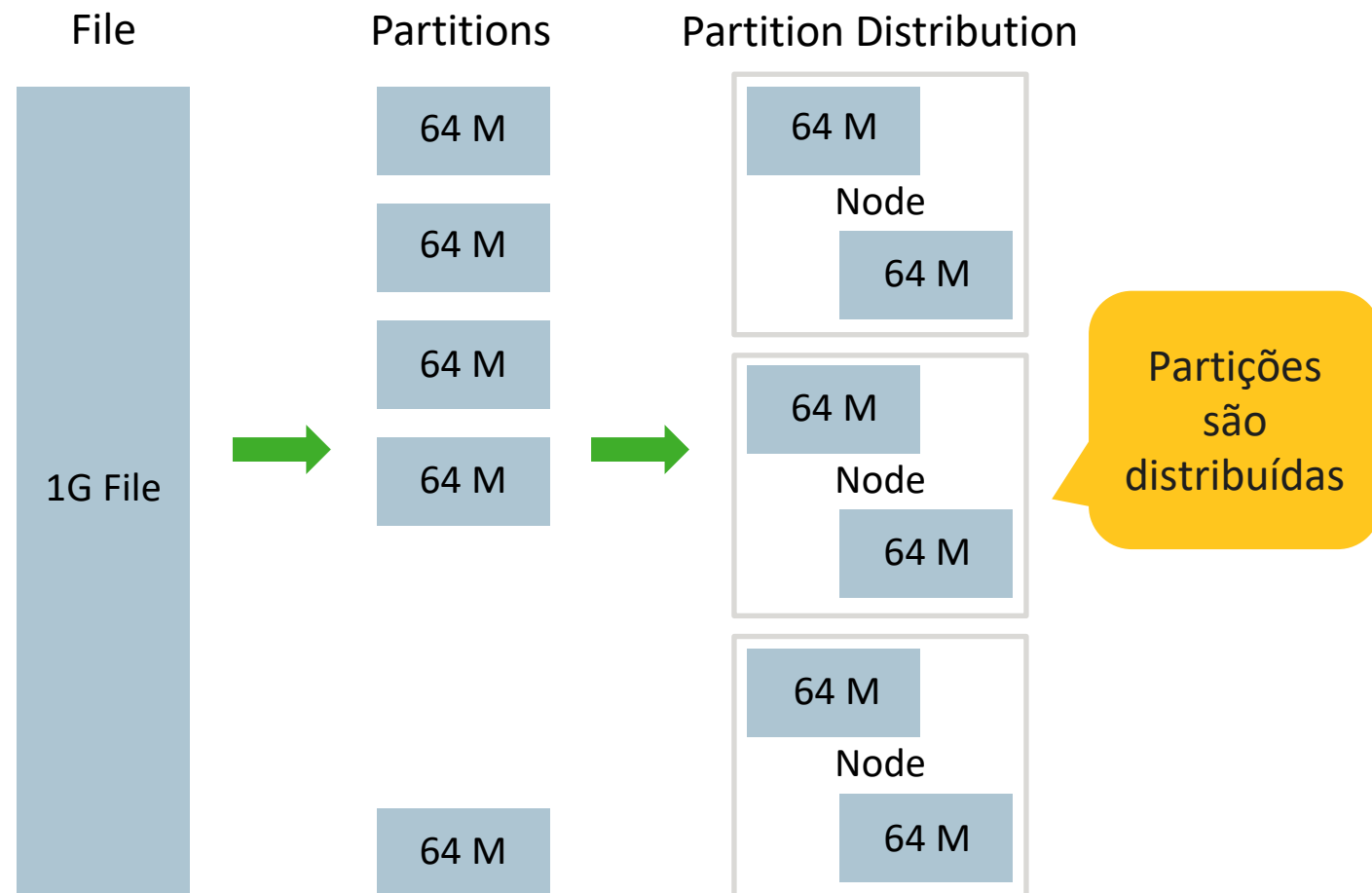
Lazy Evaluation

- Este exemplo faz a leitura de um arquivo de log
 - E filtra as linhas com informações de erro
- Até aqui, **nenhum trabalho foi feito**
- O Client solicita a primeira linha
 - Aciona o gatilho de execução do DAG
 - **Aqui**, o trabalho é feito
 - O resultado é entregue ao client



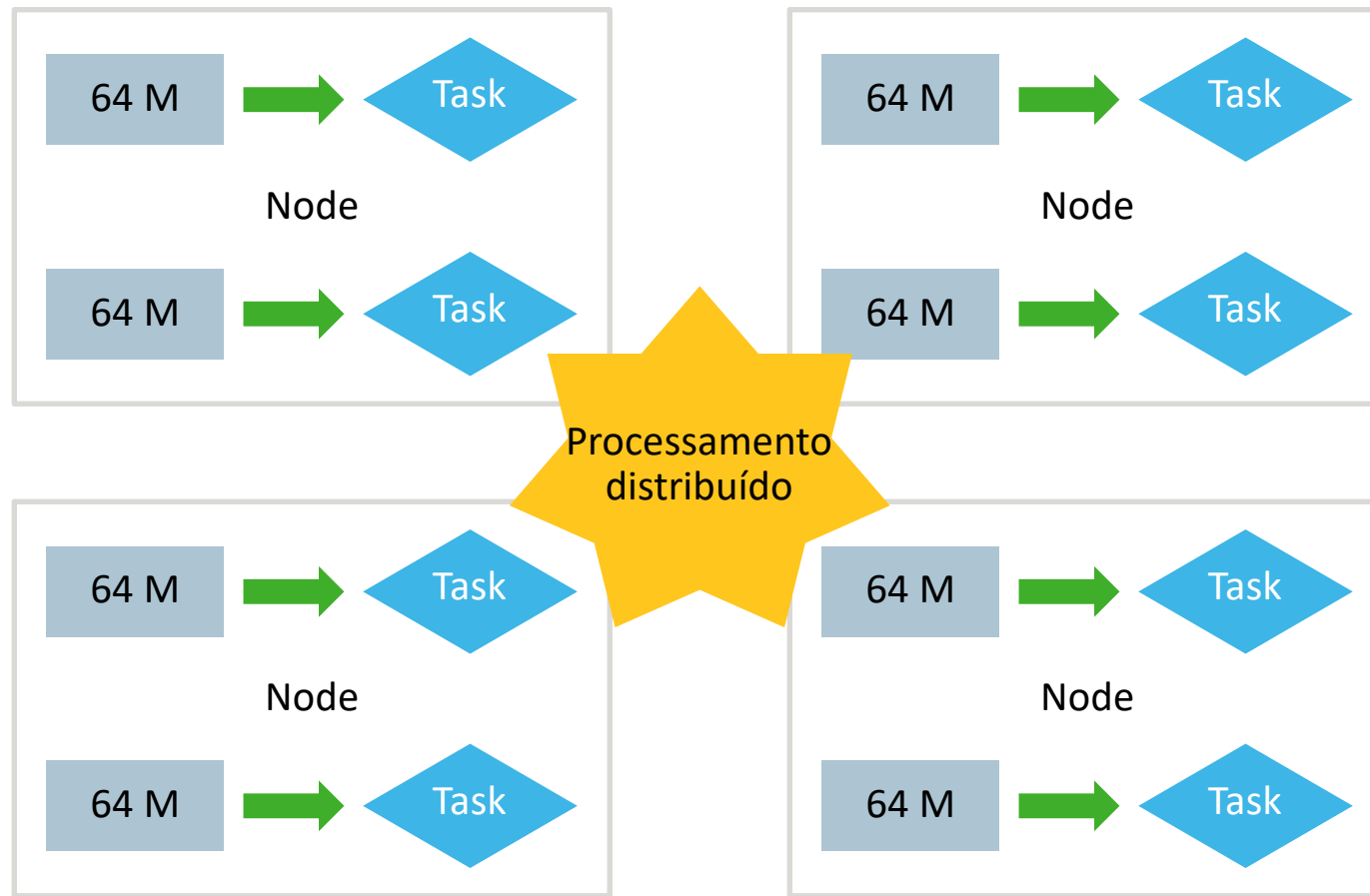
Particionamento de RDD

- Os dados de uma RDD são distribuídos pelo cluster
 - e.g. Com HDFS, o Spark cria partições do RDD a partir dos blocos do HDFS



Partições do RDD e Processamento Distribuído

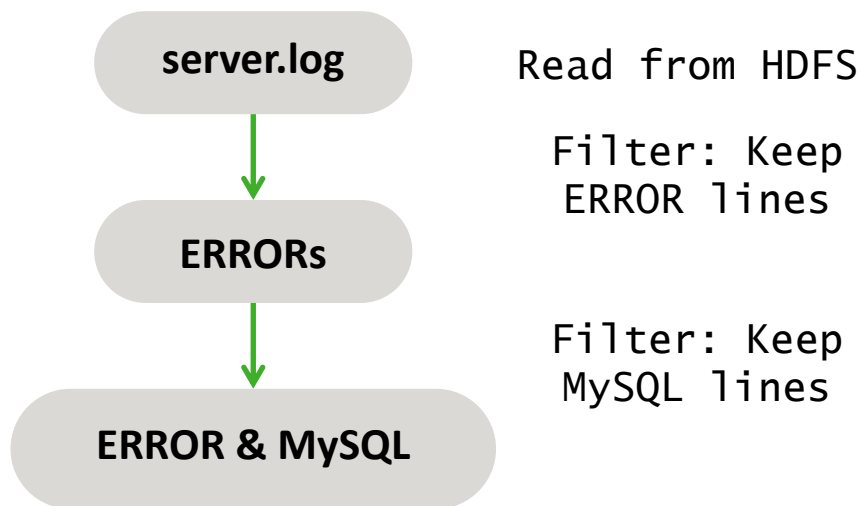
- Os nós executam tasks em paralelo nas partições
 - O Spark colocará tarefas em conjunto com seus dados



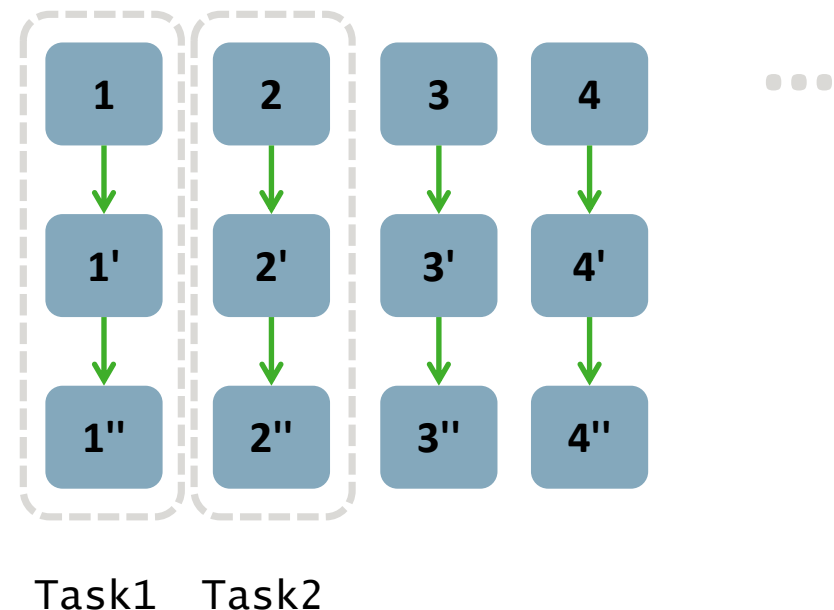
Transformações Criam Novas Partições

- Uma transformação em uma partição cria uma nova partição de uma nova RDD
- Transformações em sequência podem ser feitas no modelo de pipeline em uma task
- Geralmente, isso pode ser feito com dados na memória (mais rápido)
- Algumas transformações precisam de shuffling
 - Veremos mais sobre isso

Visualização do RDD



Visualização da Partição



Exemplo: Partições do RDD

- Ilustração de particionamento de uma RDD de um arquivo de logs

Partition 1	Partition 2	Partition 3
INFO, (msg1 ...MySQL...)	INFO (msg6 ...)	ERROR, (msg11 ...MySQL ...)
ERROR, (msg2 ...)	ERROR, (msg7 ...)	WARN, (msg12, ...)
ERROR, (msg3 ...MySQL ...)	ERROR, (msg8 ...)	INFO, (msg13 ...)
INFO, (msg4 ...)	WARN, (msg9 ...)	WARN, (msg14 ...)
ERROR, (msg5 ...MySQL...)	ERROR, (msg10 ...)	INFO, (msg15 ...)

Exemplo: Transformação do RDD

- Nós transformamos (filtramos) o RDD — cada partição processa o seu dado

Partition 1	Partition 2	Partition 3	RDD
INFO, (msg1 ...MySQL...)	INFO (msg6 ...)	ERROR, (msg11 ...MySQL ...)	
ERROR, (msg2 ...)	ERROR, (msg7 ...)	WARN, (msg12, ...)	
ERROR, (msg3 ...MySQL ...)	ERROR, (msg8 ...)	INFO, (msg13 ...)	
INFO, (msg4 ...)	WARN, (msg9 ...)	WARN, (msg14 ...)	
ERROR, (msg5 ...MySQL...)	ERROR, (msg10 ...)	INFO, (msg15 ...)	



Filter: Keep
ERROR lines

Partition 1'	Partition 2'	Partition 3'	RDD '
		ERROR, (msg11 ...MySQL ...)	
ERROR, (msg2 ...)	ERROR, (msg7 ...)		
ERROR, (msg3 ...MySQL ...)	ERROR, (msg8 ...)		
ERROR, (msg5 ...MySQL...)	ERROR, (msg10 ...)		

Exemplo: Transformação do RDD

- Nós transformamos (filtramos) o RDD novamente

Partition 1'	Partition 2'	Partition 3'	RDD '
		ERROR, (msg11 ...MySQL ...)	
ERROR, (msg2 ...)	ERROR, (msg7 ...)		
ERROR, (msg3 ...MySQL ...)	ERROR, (msg8 ...)		
ERROR, (msg5 ...MySQL...)	ERROR, (msg10 ...)		

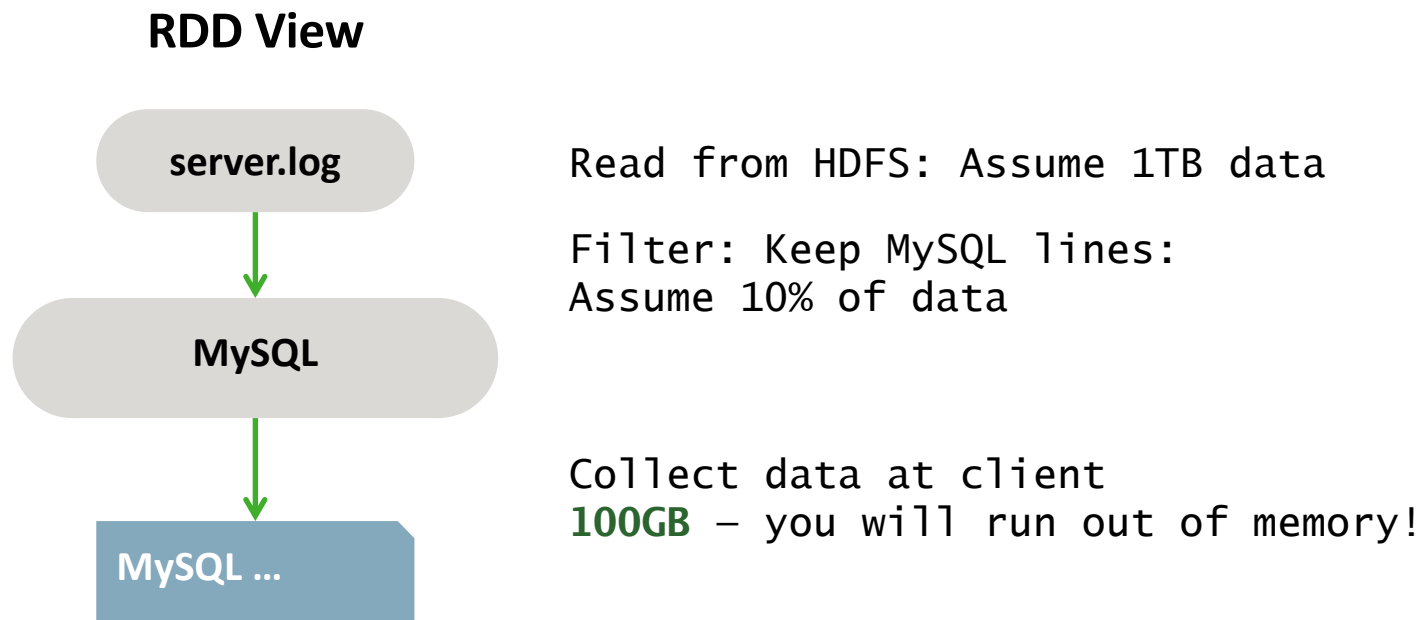


Filter: Keep
MySQL lines

Partition 1''	Partition 2''	Partition 3''	RDD ''
		ERROR, (msg11 ...MySQL ...)	
ERROR, (msg3 ...MySQL ...)			
ERROR, (msg5 ...MySQL...)			

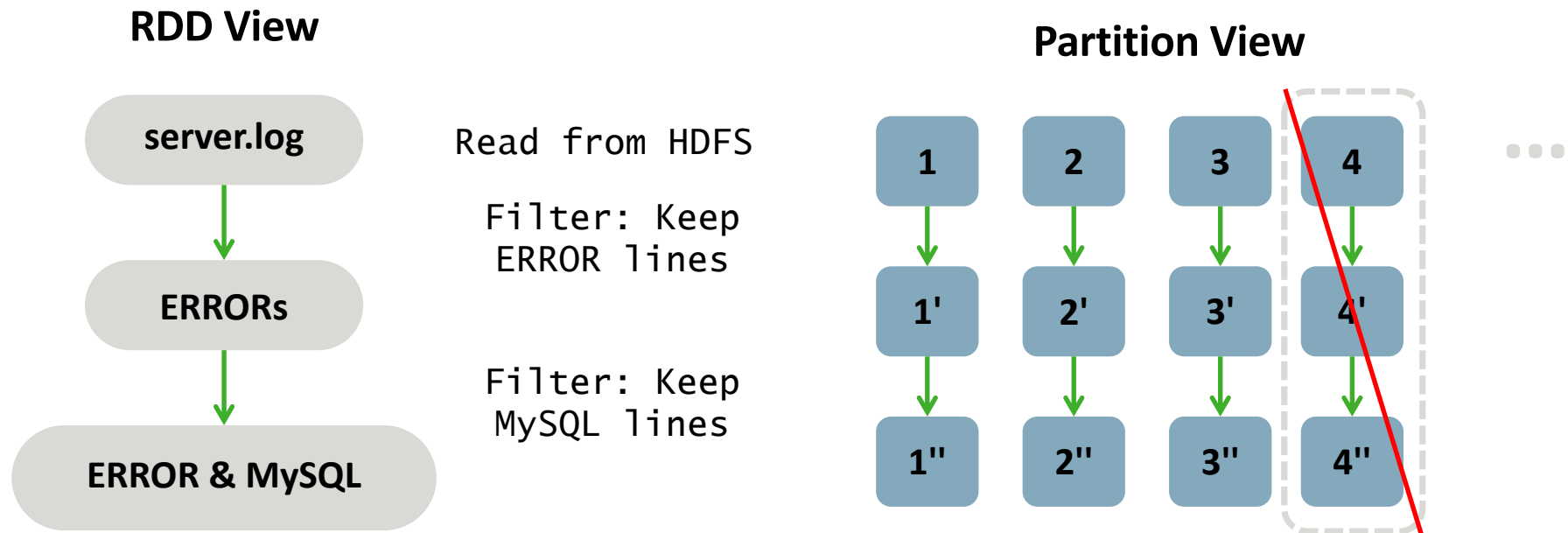
Cuidado com os retornos para o client

- Não retorne dados demais para o seu client!
 - Isso pode causar um overhead de memória para o client
 - Pode-se salvar esses dados em um armazenamento distribuído



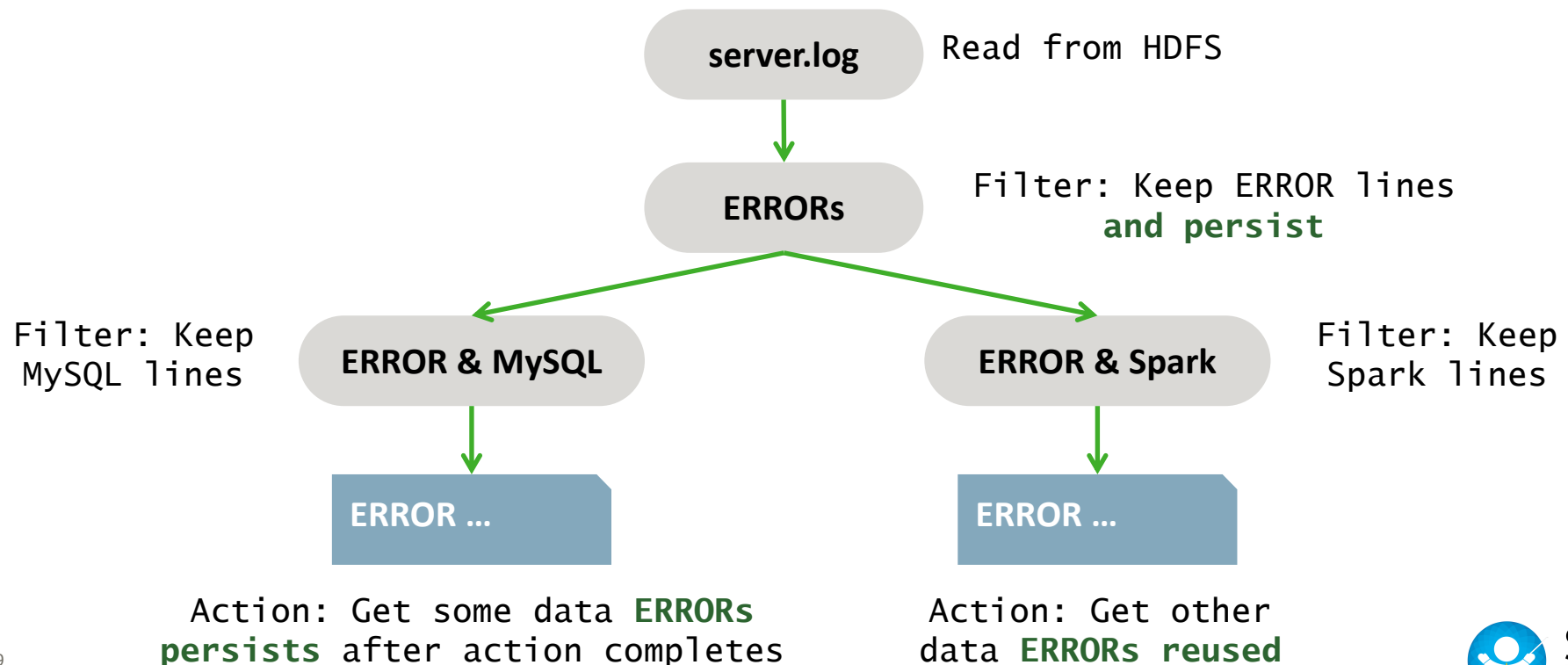
Tolerância a falhas

- O Spark rastreia as transformações que criam um RDD
 - **Lineage**: Uma série de transformações que criaram a RDD
- Uma partição perdida pode ser reconstruída pelo seu “Lineage”
 - e.g. Se a partição 4 é perdida, o Spark pode ler o bloco no HDFS novamente e refazer todas as transformações a partir do “Lineage”, e assim, recuperar o dado



RDDs são transientes por padrão

- Quando uma ação é concluída, sua RDD desaparece.
 - Se você precisa dela novamente, então ela é re-processada
- Você pode solicitar ao Spark para persistir a RDD e mantê-la em memória
 - Muito útil para RDDs que precisam de muitos recursos para serem re-processadas



Parte 3.2: Trabalhando com RDDs

Criando um RDD

- Duas formas de criar:
 - **Carregar arquivos:** De um file system local ou distribuído
 - **Paralelizar uma coleção:** Para poucos dados ou dados de teste
 - Tudo deve caber na memória do seu nó

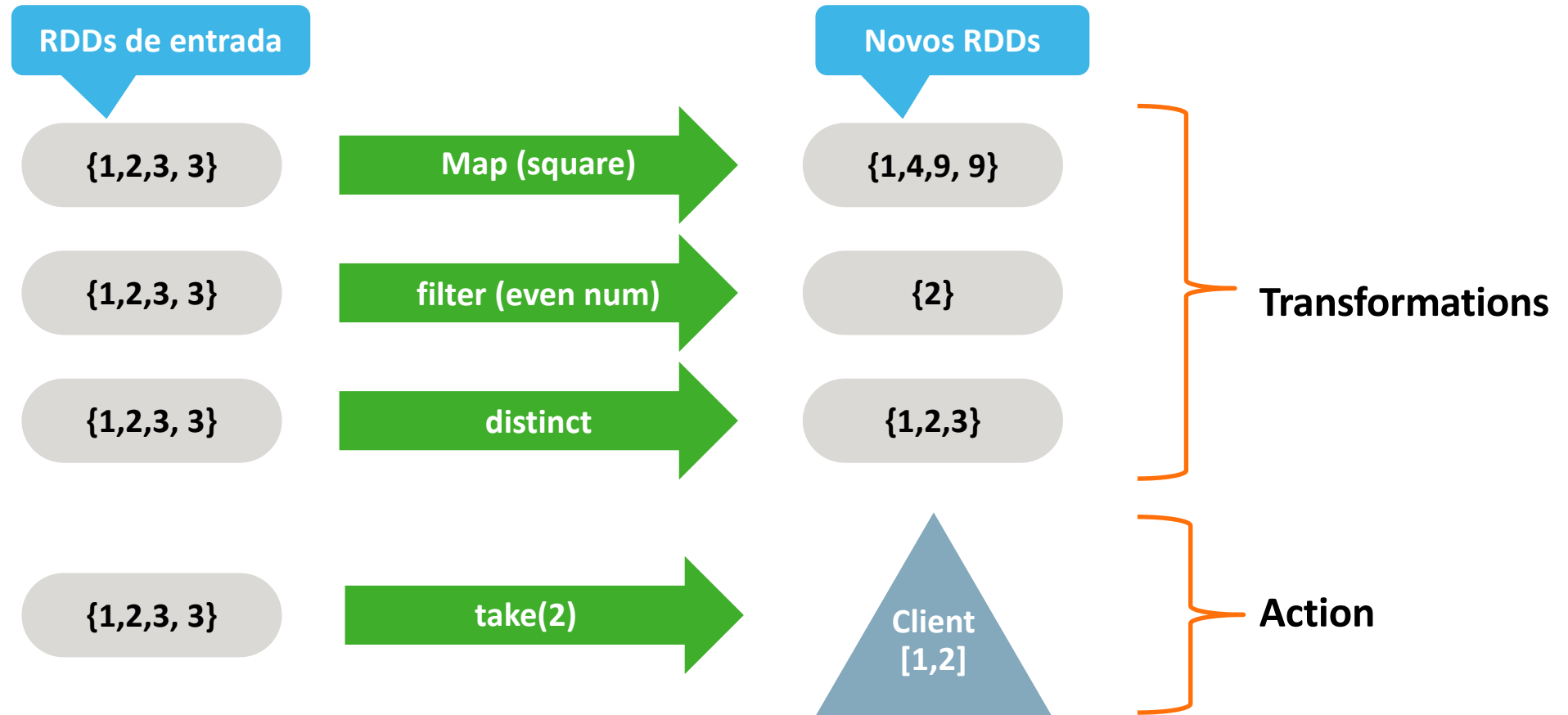
```
// Criando um RDD a partir de uma Coleção do Scala
> val numbers = sc.parallelize (List(1,2,3,3))

// Criando um RDD a partir de um arquivo no File System local
> val oneFile = sc.textFile("README.md")

// Criando um RDD a partir de multiplos arquivos no File System local
> val multiFile = sc.textFile("data/ml1ib/*.txt")

// Criando um RDD a partir de um arquivo no HDFS
> val hdfsFile =
  sc.textFile("hdfs://namehost:9000/logs/log20170301.txt")
```

Exemplos de Transformações/Ações



Detalhes do método map()

- **map(f: (T)=>U)**: Aplica a função “f” para todos os elementos
 - f() recebe um argumento e retorna um valor mapeado
 - **Retorno**: Um novo RDD de elementos mapeados



- Exemplo:
 - Cria um novo RDD a partir do quadrado dos elementos do RDD de entrada
 - O argumento do método map() é uma função anônima

```
> val numbers = sc.parallelize (List(1,2,3,3))

// Mapeie os números do RDD com o quadrado de cada elemento
> val squares=numbers.map(x=> x*x)

// Colete todos os dados do RDD
> squares.collect()
res0: Array[Int] = Array(1, 4, 9, 9)
```


Detalhes do método filter()

- **filter(f(T=>Bool))**: filtrar os elementos na função f()
 - f() Recebe um argumento e retorna um boolean
 - **Retorno**: Uma nova RDD cujo f(elemento)==true

- Exemplo

- Cria um novo RDD com os números que são ímpares do RDD de entrada



```
> val numbers = sc.parallelize (List(1,2,3,3))

// Filtrar números ímpares
> val odds=numbers.filter(x=> x%2 == 1)
// Ou numbers.filter(_%2 == 1)

> odds.collect()
res1: Array[Int] = Array(1, 3, 3)
```

Visão geral de Ações

- Exemplos de algumas ações e seus resultados

```
> val numbers = sc.parallelize (List(1,2,3,3))  
> val odds=numbers.filter(x=> x%2 == 1)
```

```
// Contagem de um RDD
```

```
> numbers.count()
```

```
res2: Long = 4
```

```
> odds.count()
```

```
res3: Long = 3
```

```
// Obter os 2 primeiros elementos
```

```
> odds.take(2)
```

```
res4: Array[Int] = Array(1, 3)
```