Sapere utile

IFOA
Istituto Formazione Operatori Aziendali

BIG DATA e Analisi dei Dati

Lezione 4 – Introduzione a spark

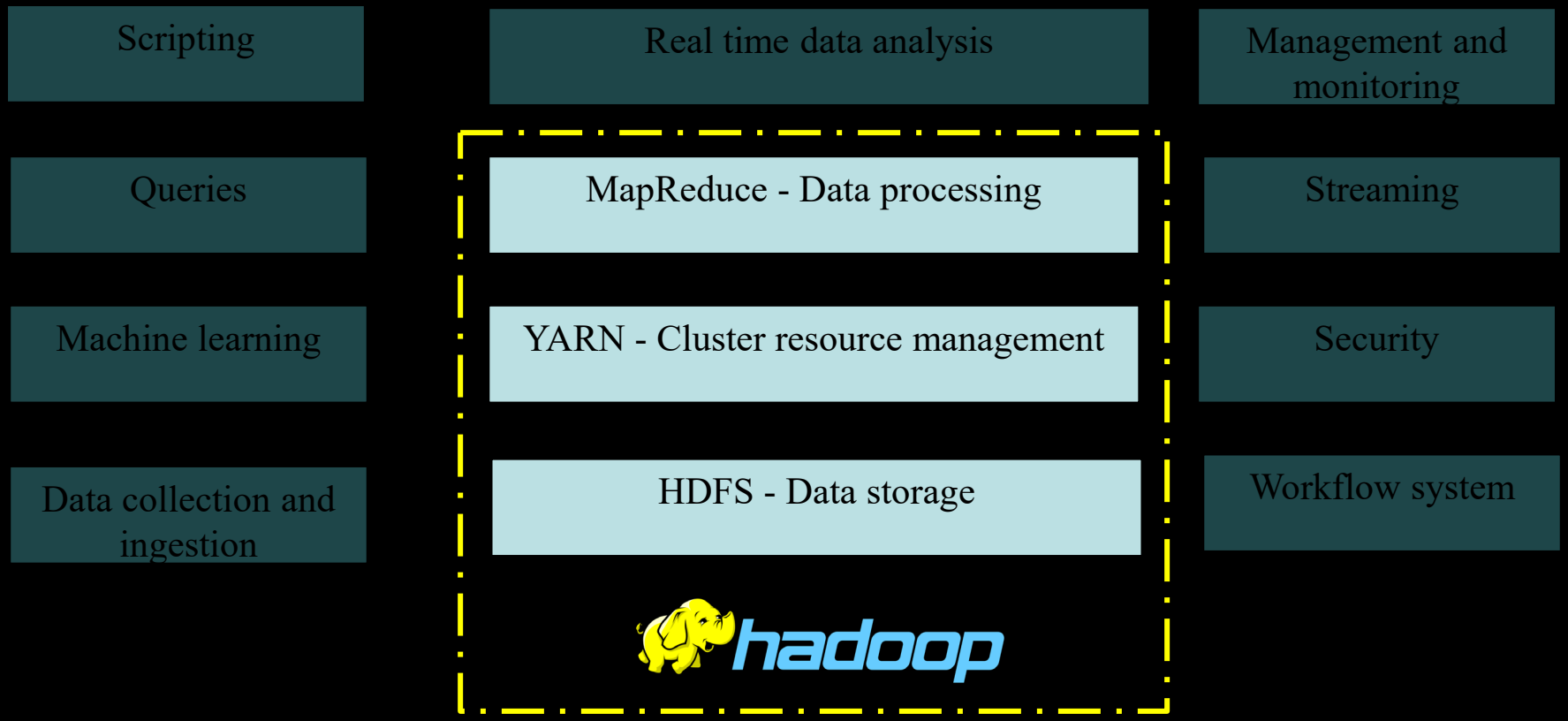
Mauro Bellone,
Robotics and AI researcher

bellonemauro@gmail.com
www.maurobellone.com

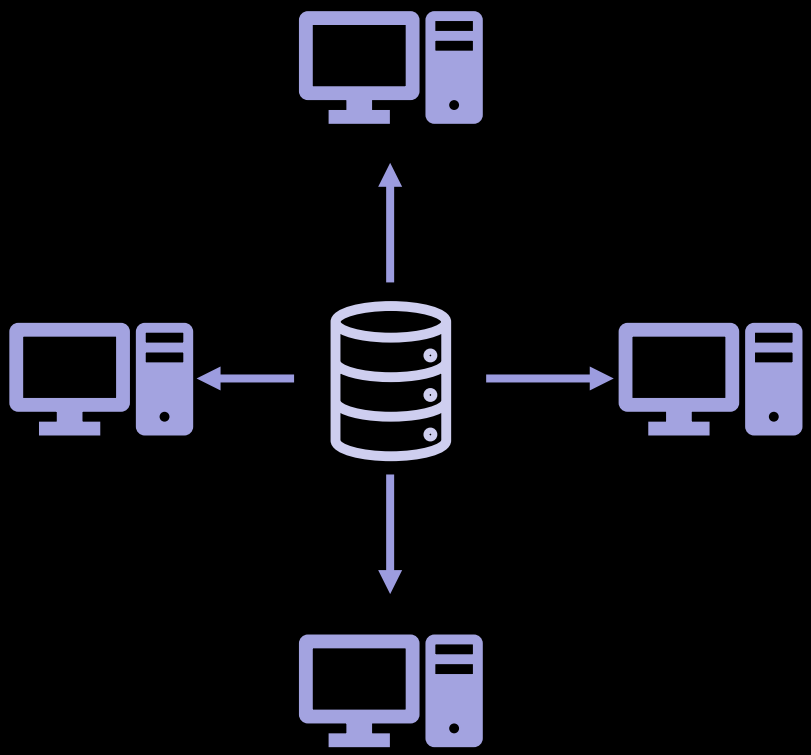
Obiettivo

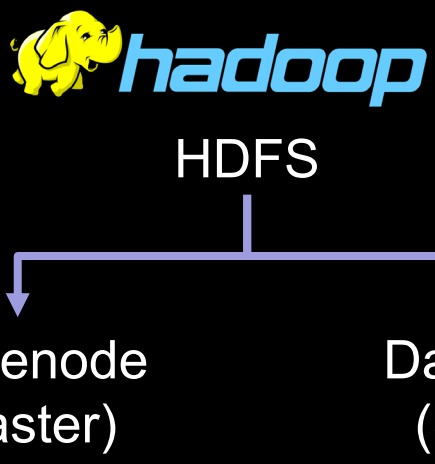
- ✓ Introduzione a Apachi SPARK
- ✓ RDD
- ✓ WordCount tutorial con il codice

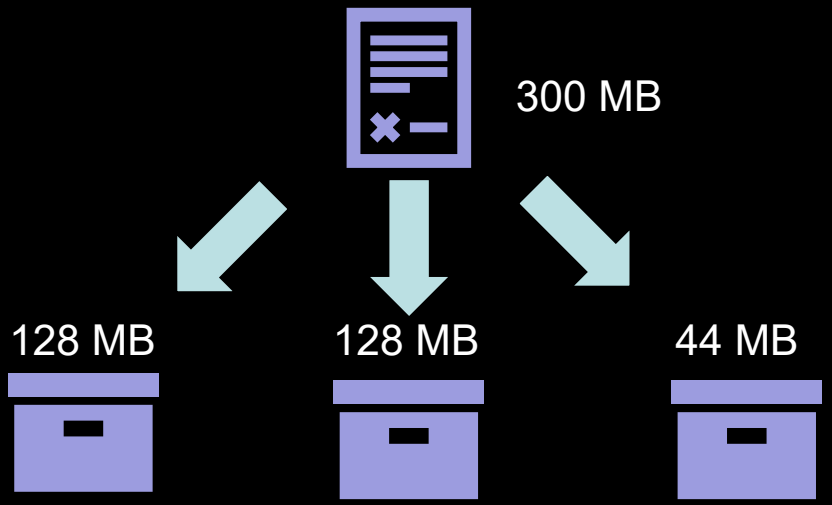
Recap - Hadoop - Ecosystem



Recap - HDFS – Hadoop distributed file system

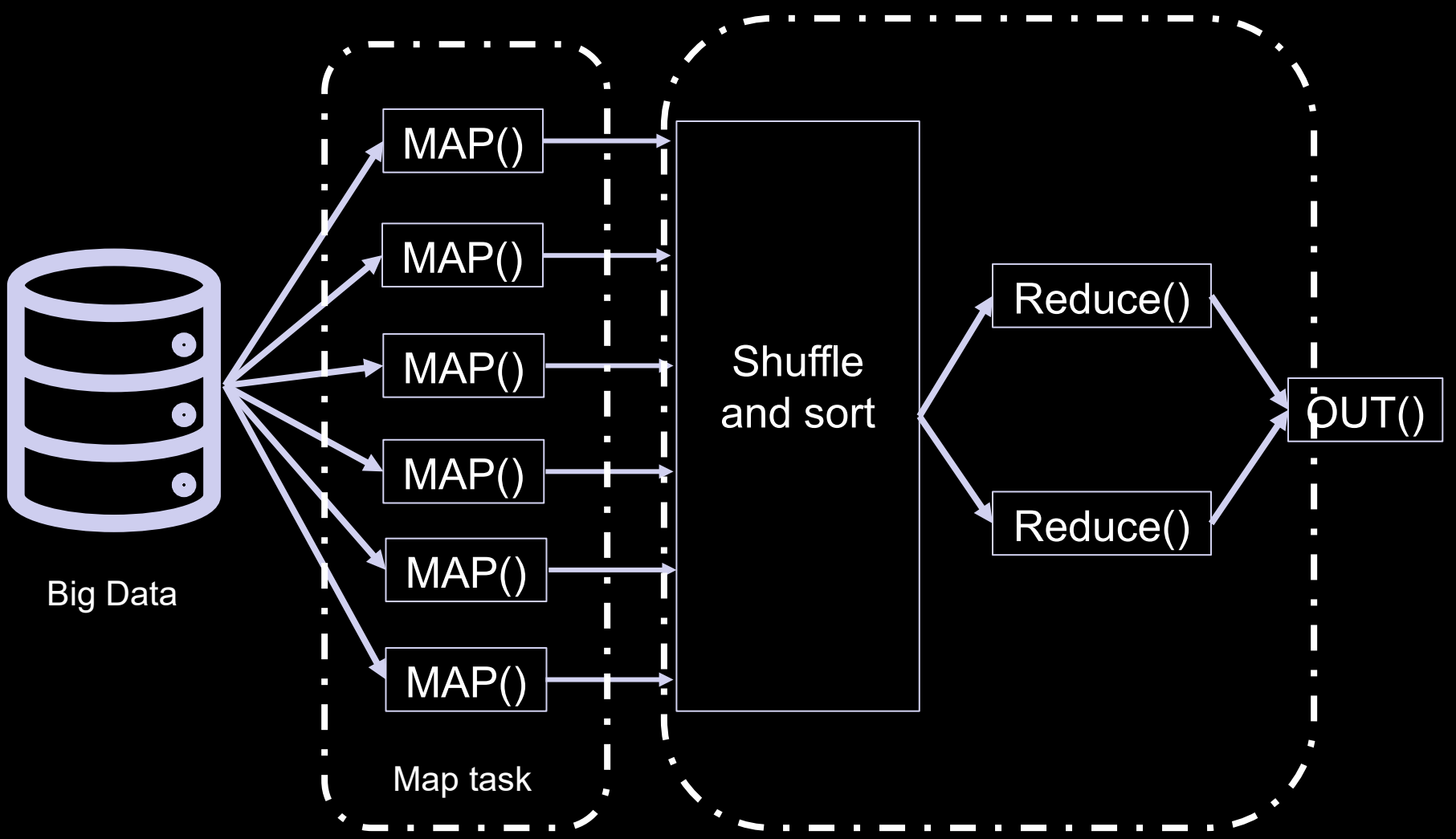






By default i dati sono divisi in blocchi di 128 MB

Recap - Data processing - MapReduce



I map task possono sfruttare il principio di data locality

I reduce task NON possono sftuttare il principio di data locality

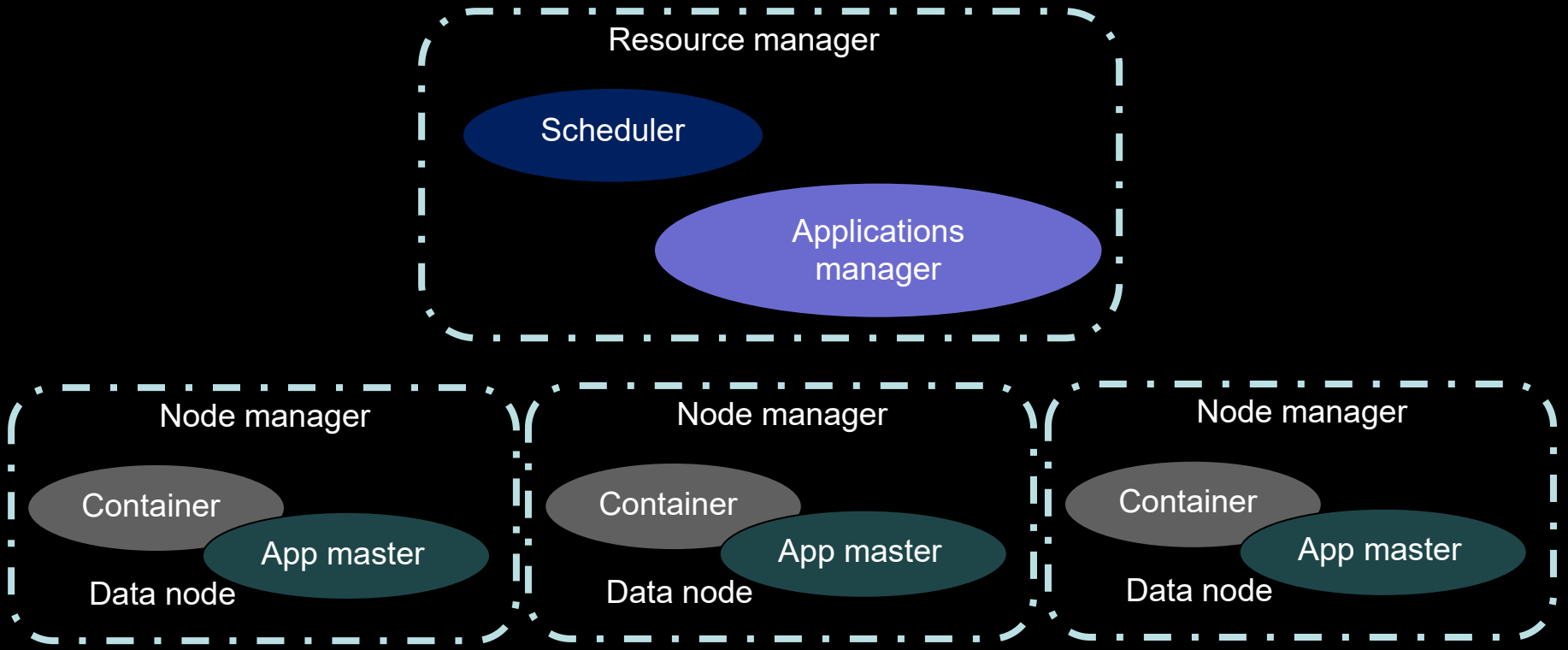
Recap - Architettura di YARN

L'architettura di YARN è costituita da tre elementi forndamentali:

- Resource manager
- Application master
- Node manager

YARN

Risorse di processamento dati



Limiti di mapReduce

Limiti di mapReduce

- ✓ Non adatto ad esecuzione in real-time

Limiti di mapReduce

- ✓ Non adatto ad esecuzione in real-time
- ✓ Non adatto per operazioni semplici

Limiti di mapReduce

- ✓ Non adatto ad esecuzione in real-time
- ✓ Non adatto per operazioni semplici
- ✓ Non lavora bene con grandi dati distribuiti sulla rete che hanno bisogno di operazioni comuni a causa della data locality

Limiti di mapReduce

- ✓ Non adatto ad esecuzione in real-time
- ✓ Non adatto per operazioni semplici
- ✓ Non lavora bene con grandi dati distribuiti sulla rete che hanno bisogno di operazioni comuni a causa della data locality
- ✓ Non funziona per le transazioni OLTP (online transaction processing)

Limiti di mapReduce

- ✓ Non adatto ad esecuzione in real-time
- ✓ Non adatto per operazioni semplici
- ✓ Non lavora bene con grandi dati distribuiti sulla rete che hanno bisogno di operazioni comuni a causa della data locality
- ✓ Non funziona per le transazioni OLTP (online transaction processing)
- ✓ Non adatto ad esecuzioni iterative

Limiti di mapReduce

- ✓ Non adatto ad esecuzione in real-time
- ✓ Non adatto per operazioni semplici
- ✓ Non lavora bene con grandi dati distribuiti sulla rete che hanno bisogno di operazioni comuni a causa della data locality
- ✓ Non funziona per le transazioni OLTP (online transaction processing)
- ✓ Non adatto ad esecuzioni iterative
- ✓ No grafi

Applicazioni con processamento in memoria

Le applicazioni con processamento in memoria sono veloci con applicazioni che hanno primitive in memoria tramite la struttura di database su colonne

Applicazioni con processamento in memoria

Le applicazioni con processamento in memoria sono veloci con applicazioni che hanno primitive in memoria tramite la struttura di database su colonne

- ✓ Tutta l'informazione è salvata in memoria (RAM) in formato compresso (binario), no indici, no key, etc.
- ✓ La compressione riduce la dimensione dei dati rispetto ai dischi
- ✓ Query semplici e rapide in memoria (no caching)
- ✓ Analisi dati flessibile, dati accessibili in tempo breve da molti utenti
- ✓ Accesso a dashboards ricche di risultati su dati esistenti

Operazioni in batch Vs. real-time

Batch

Real-time

Operazioni in batch Vs. real-time

Batch

- in una singola esecuzione possiamo processare grandi gruppi di dati e transazioni

Real-time

Operazioni in batch Vs. real-time

Batch

- in una singola esecuzione possiamo processare grandi gruppi di dati e transazioni
- è possibile eseguire jobs senza l'intervento manuale

Real-time

Operazioni in batch Vs. real-time

Batch

- in una singola esecuzione possiamo processare grandi gruppi di dati e transazioni
- è possibile eseguire jobs senza l'intervento manuale
- tutti i dati sono selezionati e dati in input al motore di processo usando parametri command-line

Real-time

Operazioni in batch Vs. real-time

Batch

- in una singola esecuzione possiamo processare grandi gruppi di dati e transazioni
- è possibile eseguire jobs senza l'intervento manuale
- tutti i dati sono selezionati e dati in input al motore di processo usando parametri command-line
- sono usati per eseguire operazioni multiple e gestire il caricamento dati, reporting e processamento offline

Real-time

Operazioni in batch Vs. real-time

Batch

- in una singola esecuzione possiamo processare grandi gruppi di dati e transazioni
- è possibile eseguire jobs senza l'intervento manuale
- tutti i dati sono selezionati e dati in input al motore di processo usando parametri command-line
- sono usati per eseguire operazioni multiple e gestire il caricamento dati, reporting e processamento offline

- Esempio: Rapporti regolari per eseguire operazioni di decision making

Real-time

Operazioni in batch Vs. real-time

Batch

- in una singola esecuzione possiamo processare grandi gruppi di dati e transazioni
- è possibile eseguire jobs senza l'intervento manuale
- tutti i dati sono selezionati e dati in input al motore di processo usando parametri command-line
- sono usati per eseguire operazioni multiple e gestire il caricamento dati, reporting e processamento offline

- Esempio: Rapporti regolari per eseguire operazioni di decision making

Real-time

- il processamento dati è eseguito in fase di caricamento dei dati, oppure quando il comando è ricevuto

Operazioni in batch Vs. real-time

Batch

- in una singola esecuzione possiamo processare grandi gruppi di dati e transazioni
- è possibile eseguire jobs senza l'intervento manuale
- tutti i dati sono selezionati e dati in input al motore di processo usando parametri command-line
- sono usati per eseguire operazioni multiple e gestire il caricamento dati, reporting e processamento offline

- Esempio: Rapporti regolari per eseguire operazioni di decision making

Real-time

- il processamento dati è eseguito in fase di caricamento dei dati, oppure quando il comando è ricevuto
- deve eseguire o rispondere entro un preciso limite temporale

Operazioni in batch Vs. real-time

Batch

- in una singola esecuzione possiamo processare grandi gruppi di dati e transazioni
- è possibile eseguire jobs senza l'intervento manuale
- tutti i dati sono selezionati e dati in input al motore di processo usando parametri command-line
- sono usati per eseguire operazioni multiple e gestire il caricamento dati, reporting e processamento offline

- Esempio: Rapporti regolari per eseguire operazioni di decision making

Real-time

- il processamento dati è eseguito in fase di caricamento dei dati, oppure quando il comando è ricevuto
- deve eseguire o rispondere entro un preciso limite temporale

- Esempio: rilevamento transazioni fraudolente

Operazioni in batch Vs. real-time

Batch

- in una singola esecuzione possiamo processare grandi gruppi di dati e transazioni
- è possibile eseguire jobs senza l'intervento manuale
- tutti i dati sono selezionati e dati in input al motore di processo usando parametri command-line
- sono usati per eseguire operazioni multiple e gestire il caricamento dati, reporting e processamento offline

- Esempio: Rapporti regolari per eseguire operazioni di decision making

Real-time

- il processamento dati è eseguito in fase di caricamento dei dati, oppure quando il comando è ricevuto
- deve eseguire o rispondere entro un preciso limite temporale

- Esempio: rilevamento transazioni fraudolente

Spark – Real time data analysis

SPARK è un motore di calcolo distribuito per il processamento e l'analisi di grandi quantità di dati in tempo reale.

https://spark.apache.org/

Spark – Real time data analysis

SPARK è un motore di calcolo distribuito per il processamento e l'analisi di grandi quantità di dati in tempo reale.

https://spark.apache.org/

- ✓ Velocità 100x e bassa latenza
- ✓ Effettua calcoli per dati direttamente in memoria
- ✓ È usato per processare dati in tempo reale come azioni e dati bancari
- ✓ Supporto per molti linguaggi di programmazione e per sistemi di analisi (advanced analytic)

Spark – Real time data analysis

SPARK è un motore di calcolo distribuito per il processamento e l'analisi di grandi quantità di dati in tempo reale.

https://spark.apache.org/

- ✓ Velocità 100x e bassa latenza
- ✓ Effettua calcoli per dati direttamente in memoria
- ✓ È usato per processare dati in tempo reale come azioni e dati bancari
- ✓ Supporto per molti linguaggi di programmazione e per sistemi di analisi (advanced analytic)

Spark – Real time data analysis

SPARK è un motore di calcolo distribuito per il processamento e l'analisi di grandi quantità di dati in tempo reale.

https://spark.apache.org/

- ✓ Spark Core
  - ✓ RDD Resilient Distributed dataset
- ✓ Spark SQL
- ✓ Spark streaming
- ✓ Spark Machine learning Mllib
- ✓ Spark GraphX

Spark – Core

Spark core è il motore di base per il processamento dati distribuito su larga scala

Spark – Core

Spark core è il motore di base per il processamento dati distribuito su larga scala

é responsabile di:

- ✓ Gestione della memoria
- ✓ Ripristino da eventuali guasti
- ✓ Scheduling, distribuzione e monitoraggio jobs su cluster
- ✓ Interazione con i sistemi di storage (HDFS)

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

Resiliente = capace di resistere ad uno shock senza una mutazione permanente di deformazione o rottura

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- **Resilient**, i.e. fault-tolerant con l'aiuto di un grafo è in grado di ripristinare parti mancanti o parti danneggiate a causa di un node failure

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- **Resilient**, i.e. fault-tolerant con l'aiuto di un grafo è in grado di ripristinare parti mancanti o parti danneggiate a causa di un node failure
- **Distributed** quindi i dati sono allocati su nodi diversi in un cluster

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- **Resilient**, i.e. fault-tolerant con l'aiuto di un grafo è in grado di ripristinare parti mancanti o parti danneggiate a causa di un node failure
- **Distributed** quindi i dati sono allocati su nodi diversi in un cluster
- **Dataset** è un gruppo di dati partizionati con le loro primitive e i loro valori.

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- **Resilient**, i.e. fault-tolerant con l'aiuto di un grafo è in grado di ripristinare parti mancanti o parti danneggiate a causa di un node failure
- **Distributed** quindi i dati sono allocati su nodi diversi in un cluster
- **Dataset** è un gruppo di dati partizionati con le loro primitive e i loro valori.

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- In-Memory

I dati in RDD sono allocati in memoria per la maggior parte (dimensione) e il maggior tempo possible

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- In-Memory
- Immutabili o Read-Only

I dati in RDD non cambiano una volta creati, possono essere trasformati creando nuove istanze di RDD

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- In-Memory
- Immutabili o Read-Only
- Lazy evaluated

I dati in RDD non sono disponibili o trasformati fino al termine dell'azione che da inizio all'operazione

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- In-Memory
- Immutabili o Read-Only
- Lazy evaluated
- Cacheable

E' possible mantenere dei dati in una memoria persistente (disco), tuttavia è sempre preferibile usare la memoria (RAM)

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- In-Memory
- Immutabili o Read-Only
- Lazy evaluated
- Cacheable
- Parallel

I dati possono essere processati in parallelo

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- In-Memory
- Immutabili o Read-Only
- Lazy evaluated
- Cacheable
- Parallel
- Typed

Solo alcuni tipi di dati sono permessi

es. `RDD[long]` **o** `RDD[(int, string)]`

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- In-Memory
- Immutabili o Read-Only
- Lazy evaluated
- Cacheable
- Parallel
- Typed
- Partitioned

I dati in RDD sono partizionati, quindi divisi n parti e distribuiti sui nodi del cluster

Spark – RDD Resilient Distributed Dataset

Spark core è integrato con RDDs (resilient distributed datasets) che rappresenta una collezione di oggetti distribuita, immutabile e robusta che può essere operata in parallelo

- In-Memory
- Immutabili o Read-Only
- Lazy evaluated
- Cacheable
- Parallel
- Typed
- Partitioned

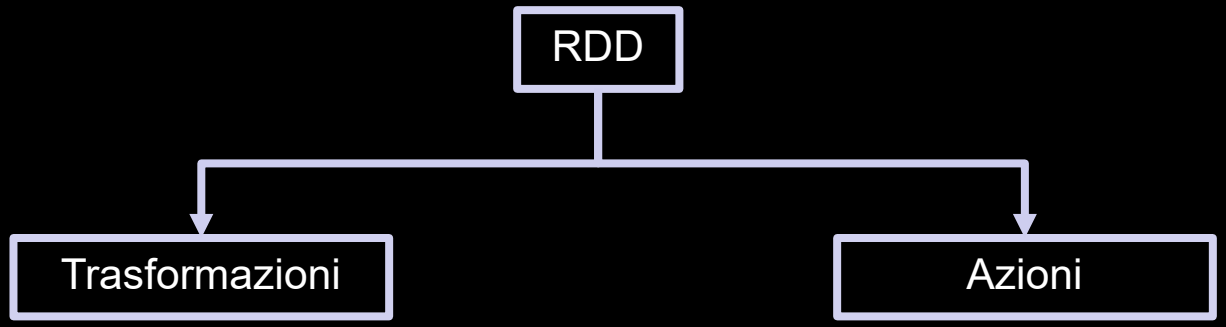
Spark – RDD Resilient Distributed Dataset

Limite:

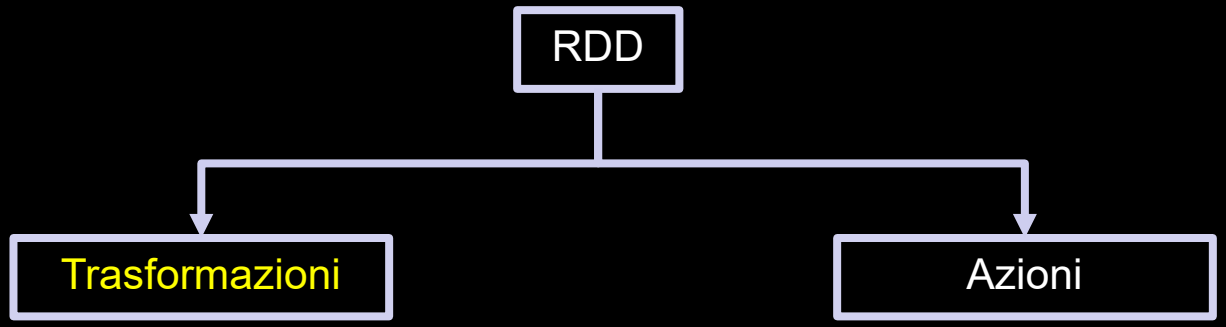
Spark RDDs non sono adatti ad applicazioni che aggiornano costantemente lo stato nella memoria fisica (storage) in maniera asincrona.

Per queste applicazioni, il modo più efficiente è quello di usare sistemi tradizionali che aggiornano il logging e creano checkpoints (database tradizionali).

Spark – RDD Resilient Distributed Dataset

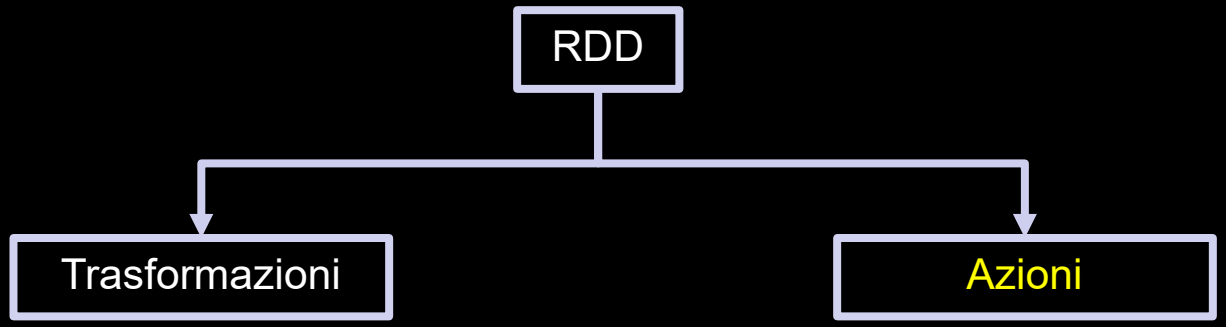


Spark – RDD Resilient Distributed Dataset



operazioni (lazy) effettuate su RDD che portano ad un nuovo risultato memorizzato su RDD

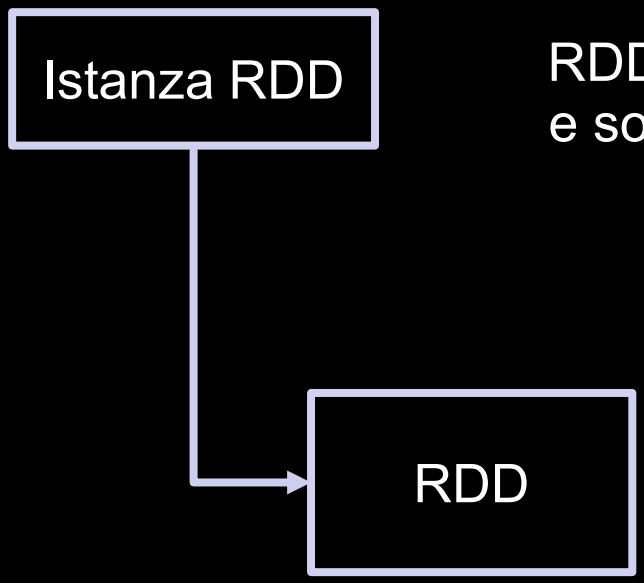
Spark – RDD Resilient Distributed Dataset



operazioni (lazy) effettuate su RDD che portano ad un nuovo risultato memorizzato su RDD

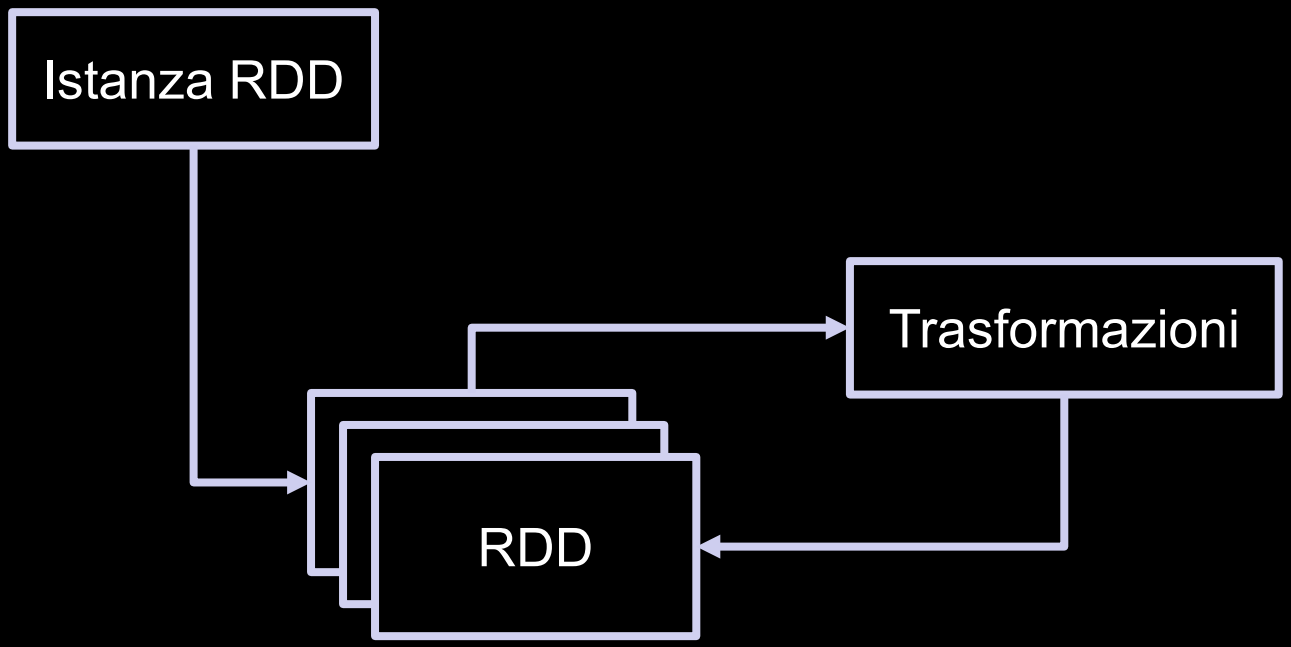
Operazioni che ritornano un valore a valle di una computazione su RDD

Spark – RDD Resilient Distributed Dataset



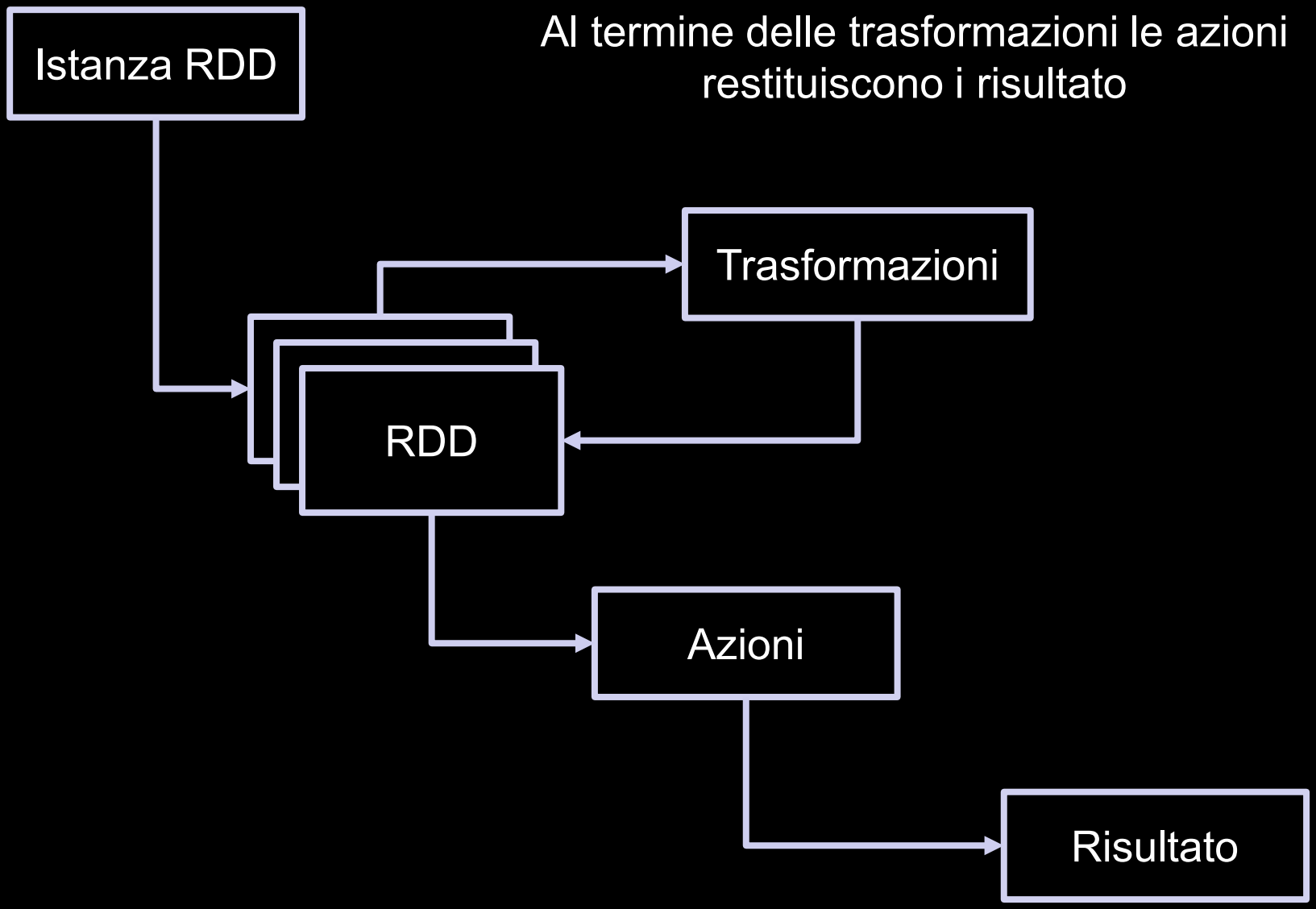
RDD sono i building blocks per ogni applicazione spark e sono istanziati tramite `sparkContext()` che crea un primo blocchetto di memoria

Spark – RDD Resilient Distributed Dataset

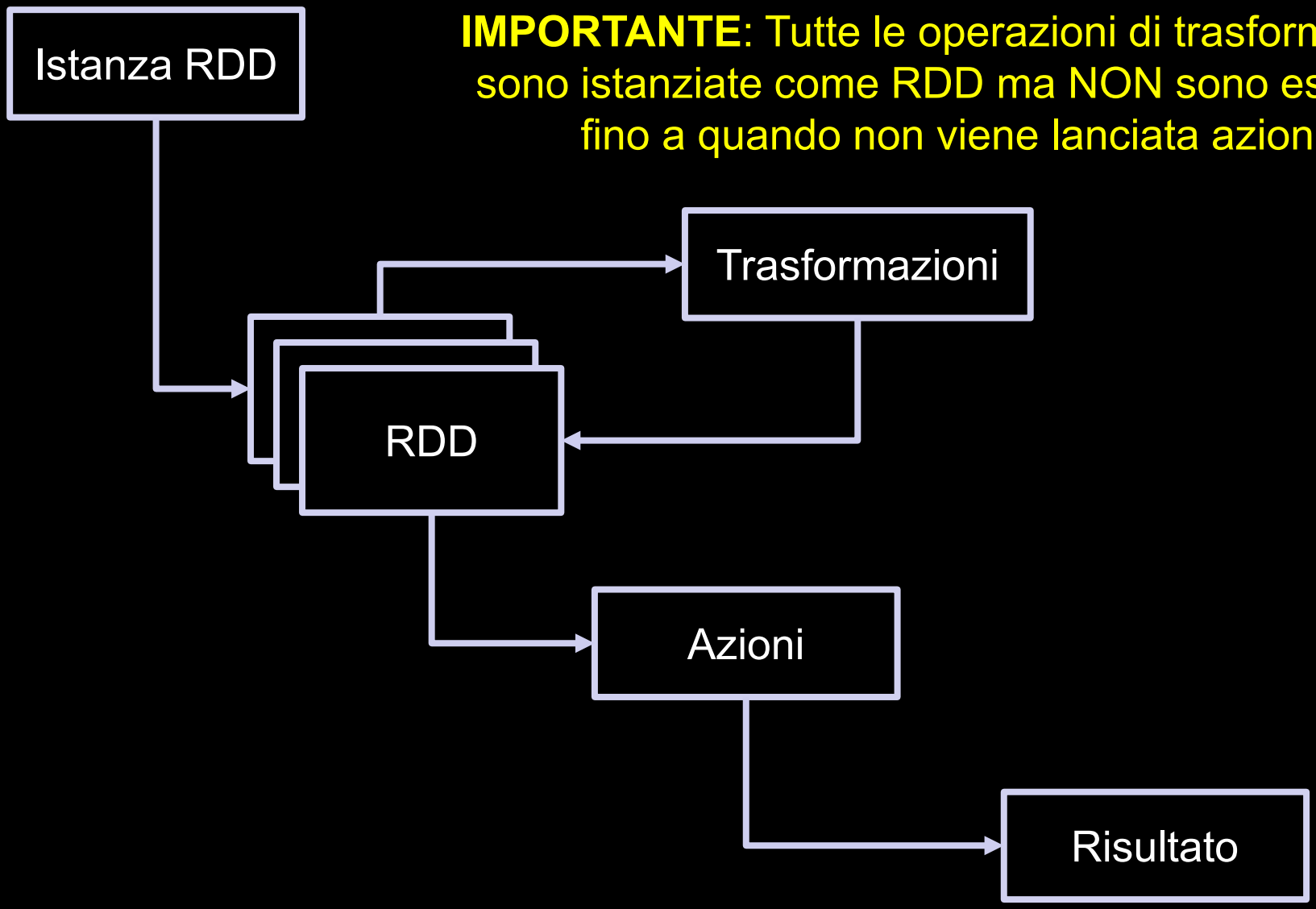


Ogni trasformazione crea un nuovo blocchetto di memoria (i dati sono immutable), possibili operazioni sono mappatura, filtraggio, unione etc.

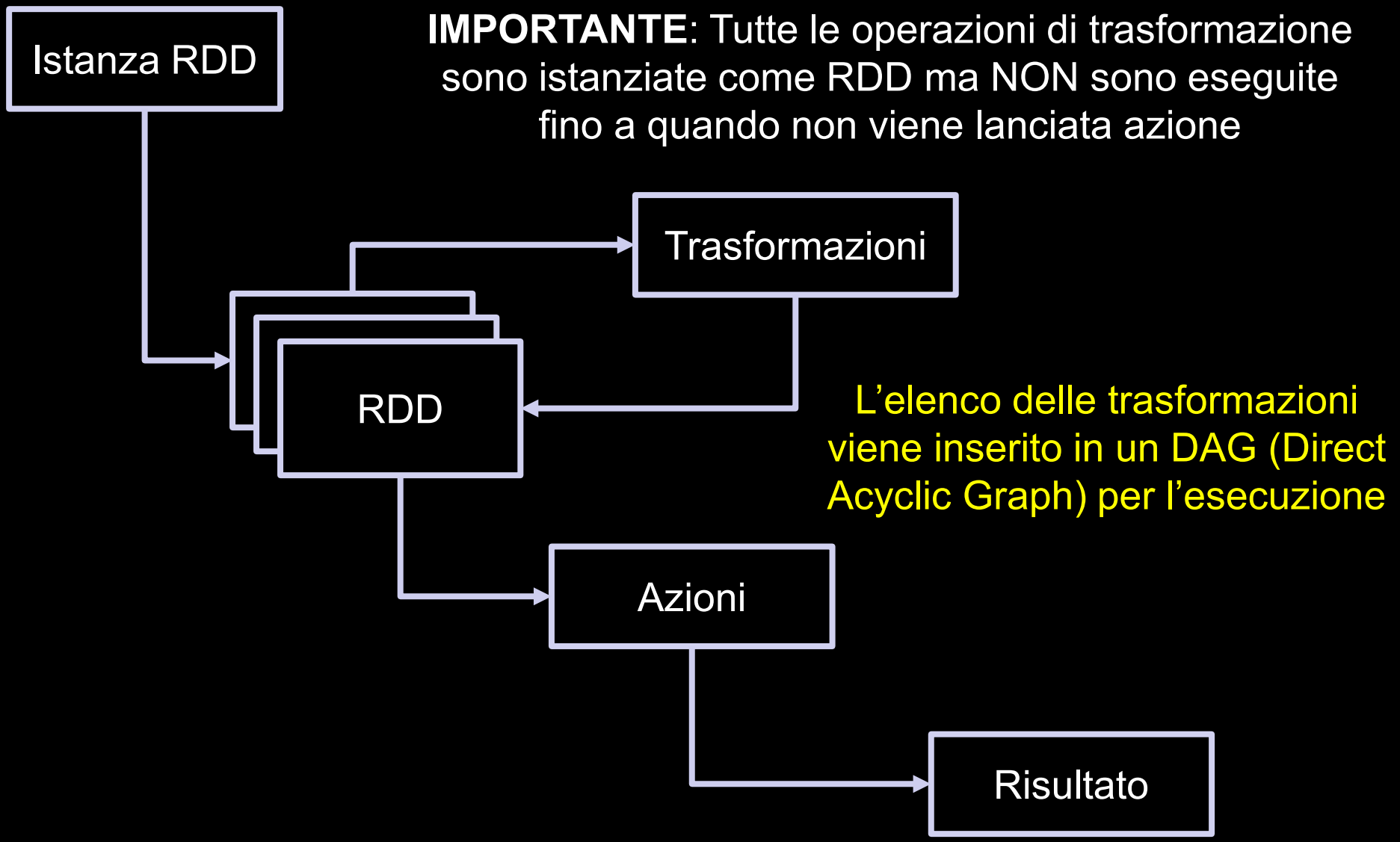
Spark – RDD Resilient Distributed Dataset



Spark – RDD Resilient Distributed Dataset

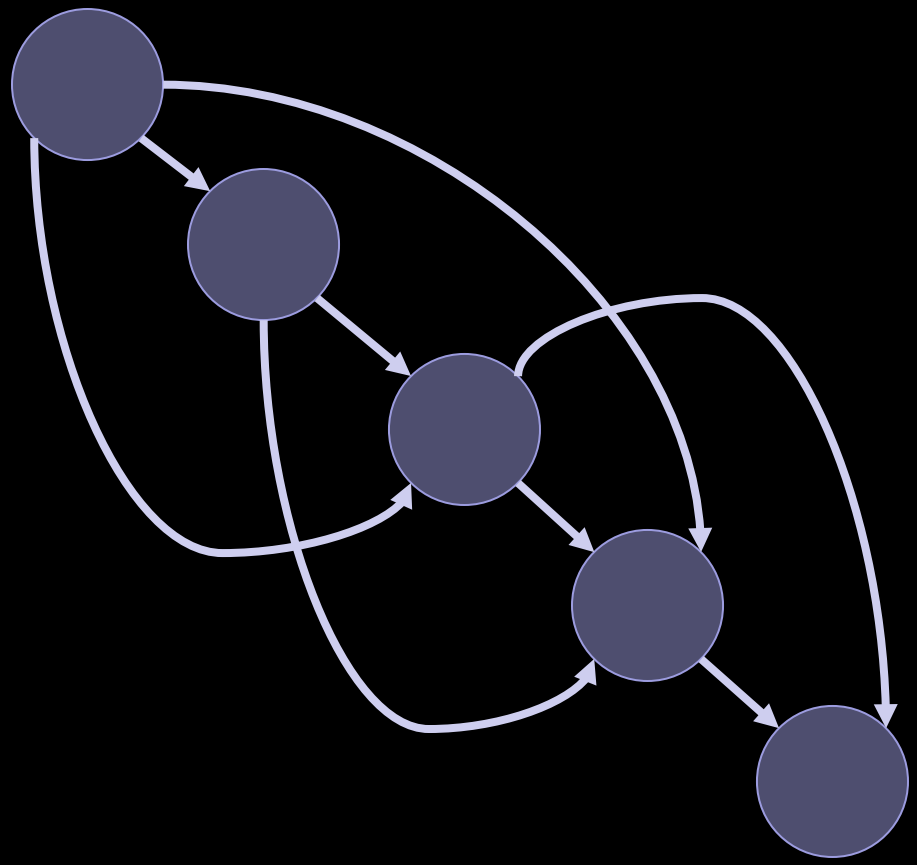


Spark – RDD Resilient Distributed Dataset



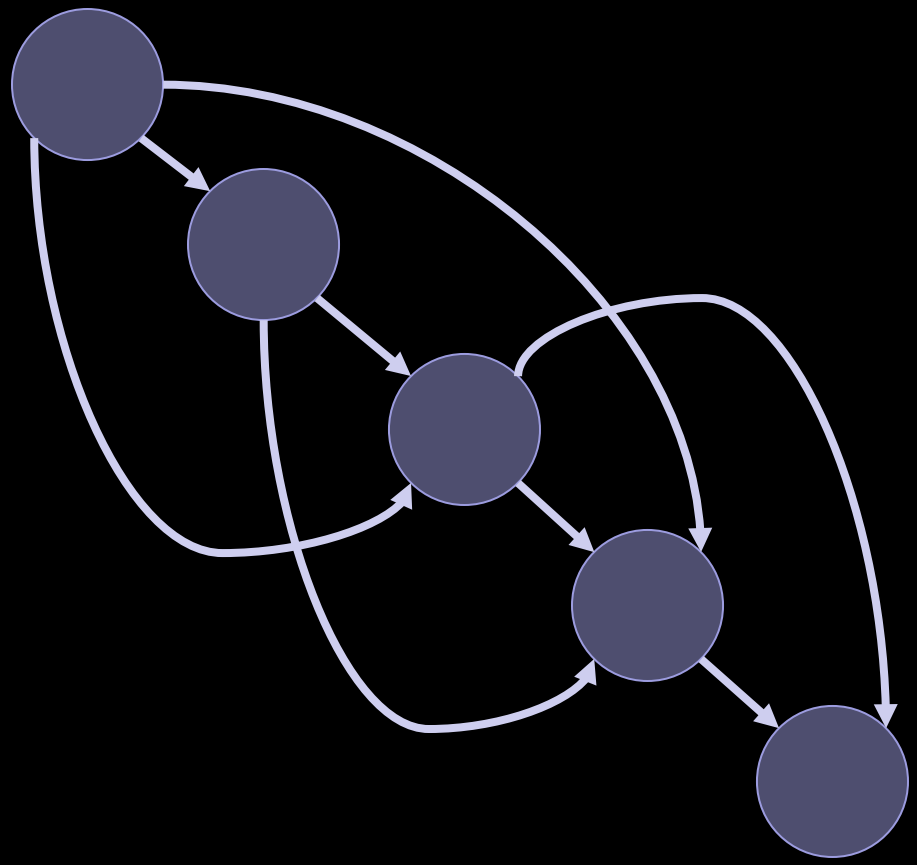
DAG – Direct Acyclic Graph

Possiamo considerarlo come un ordine topologico (grafo aciclico direzionato) nel quale ogni ramo va da una operazione ad una delle successive.



DAG – Direct Acyclic Graph

Possiamo considerarlo come un ordine topologico (grafo aciclico direzionato) nel quale ogni ramo va da una operazione ad una delle successive.



Un grafo è aciclico solo se ha un ordine topologico.

Nel nosto caso significa una procedura di esecuzione delle operazioni

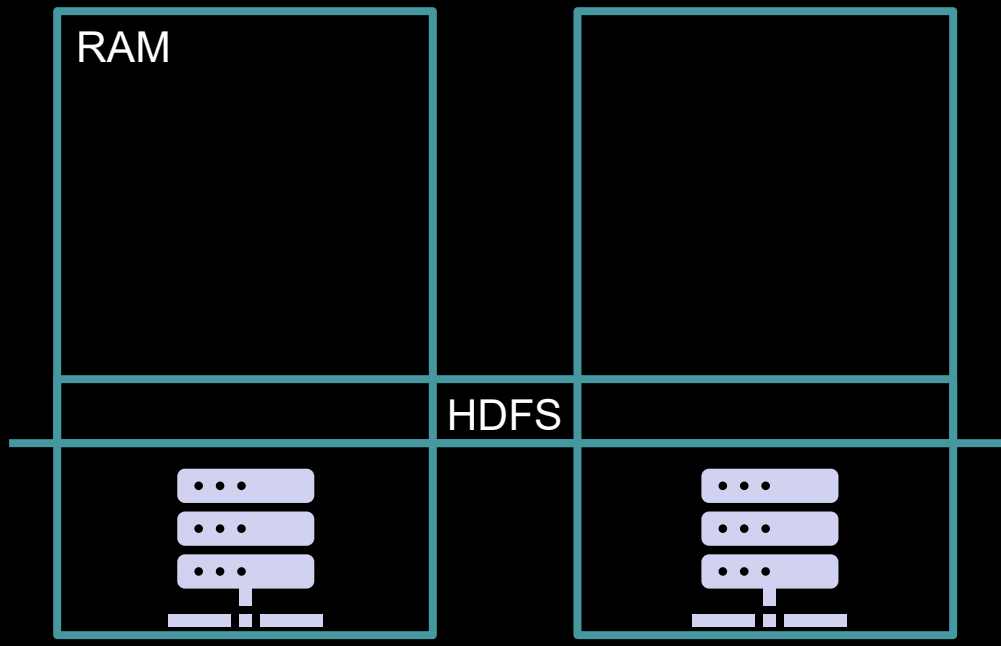
Spark – RDD Resilient Distributed Dataset

val x = sparkContext.textFile(_path_) ===> crea una istanza di RDD

Spark – RDD Resilient Distributed Dataset

val x = sparkContext.textFile(_path_) ===> crea una istanza di RDD

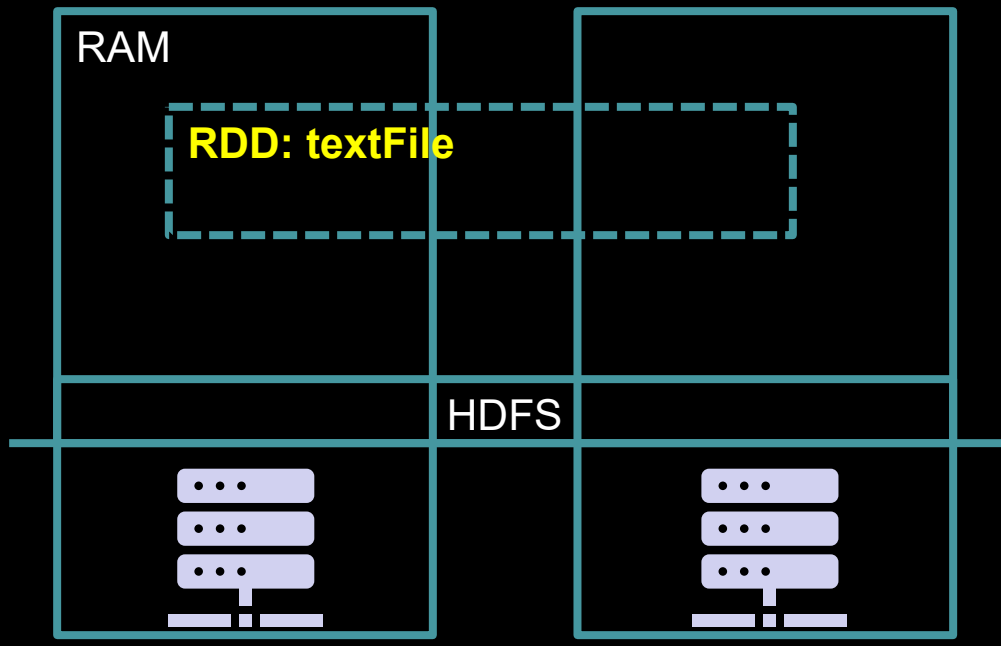
stiamo creando x, una variabile immutable in RDD che usiamo tramite uno sparkContext per leggere un file, questo crea un elemento nel DAG di esecuzione



Spark – RDD Resilient Distributed Dataset

val x = sparkContext.textFile(_path_) ===> crea una istanza di RDD

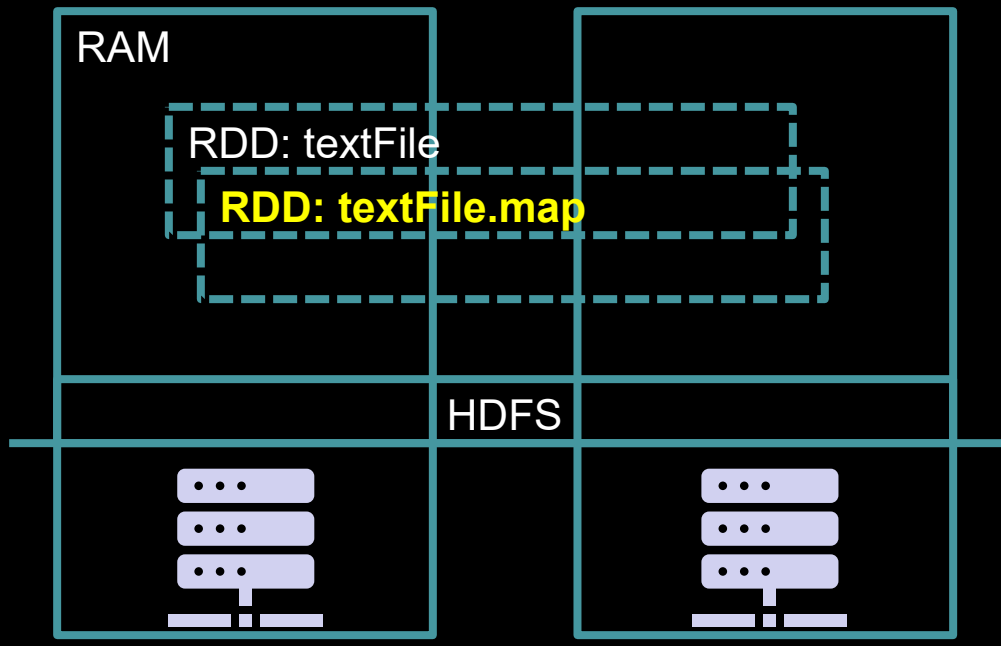
Un RDD è un dataset logico distribuito sui diversi PC che fanno parte del cluster



Spark – RDD Resilient Distributed Dataset

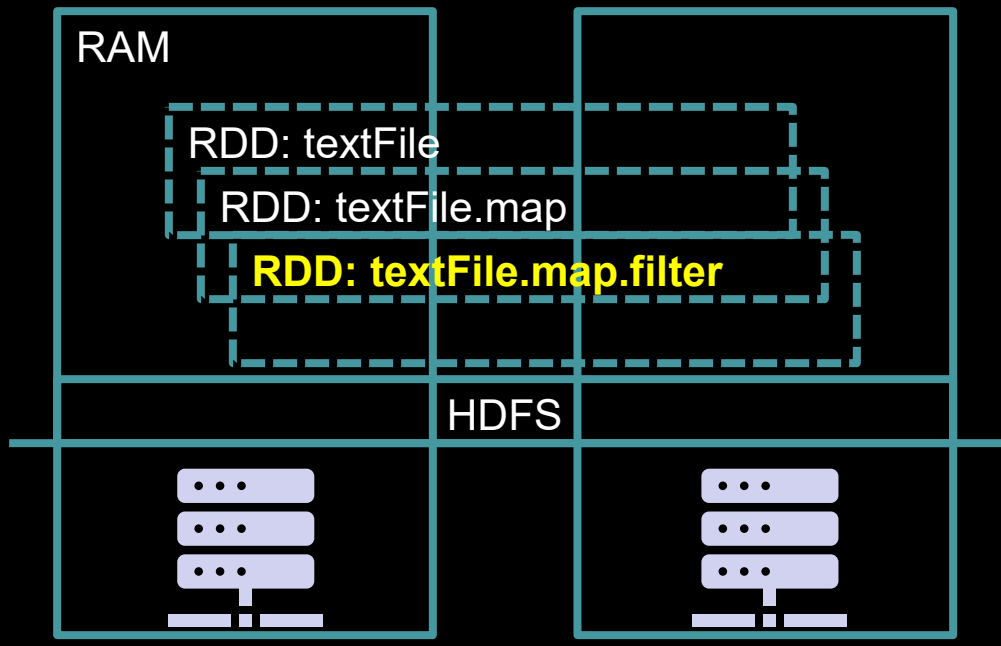
val x = sparkContext.textFile(_path_) ===> crea una istanza di RDD
val y = x.map(____) ===> trasformazione

Con x.map creo una nuova istanza di RDD che dipende dalla prima istanza



Spark – RDD Resilient Distributed Dataset

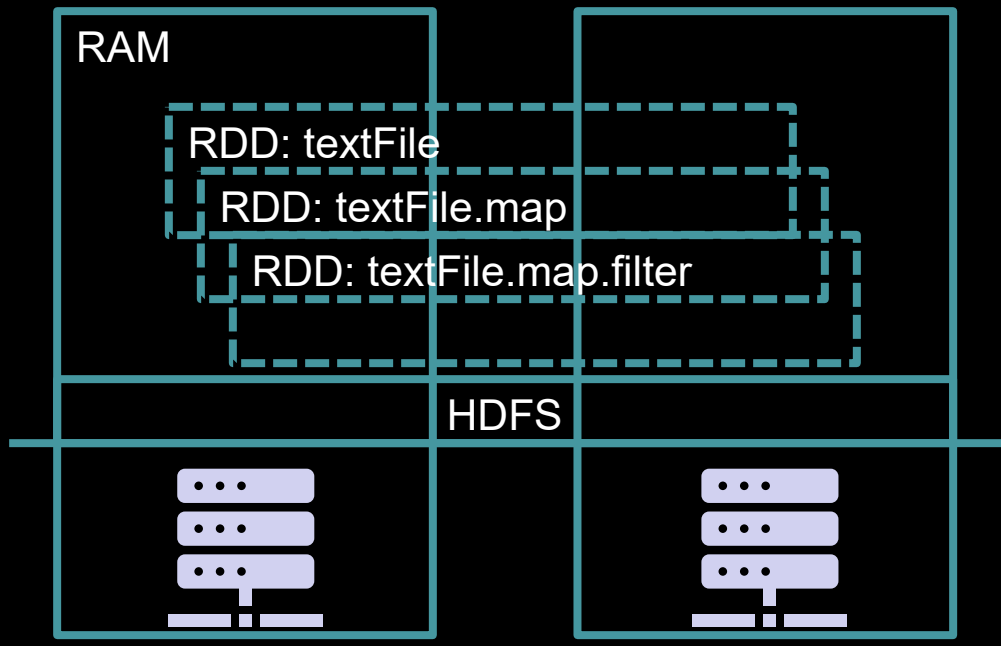
val x = sparkContext.textFile(_path_) ===> crea una istanza di RDD
val y = x.map(____) ===> trasformazione
val z = y.filter (____) ===> trasformazione



Spark – RDD Resilient Distributed Dataset

val x = sparkContext.textFile(_path_) ===> crea una istanza di RDD
val y = x.map(____) ===> trasformazione
val z = y.filter (____) ===> trasformazione

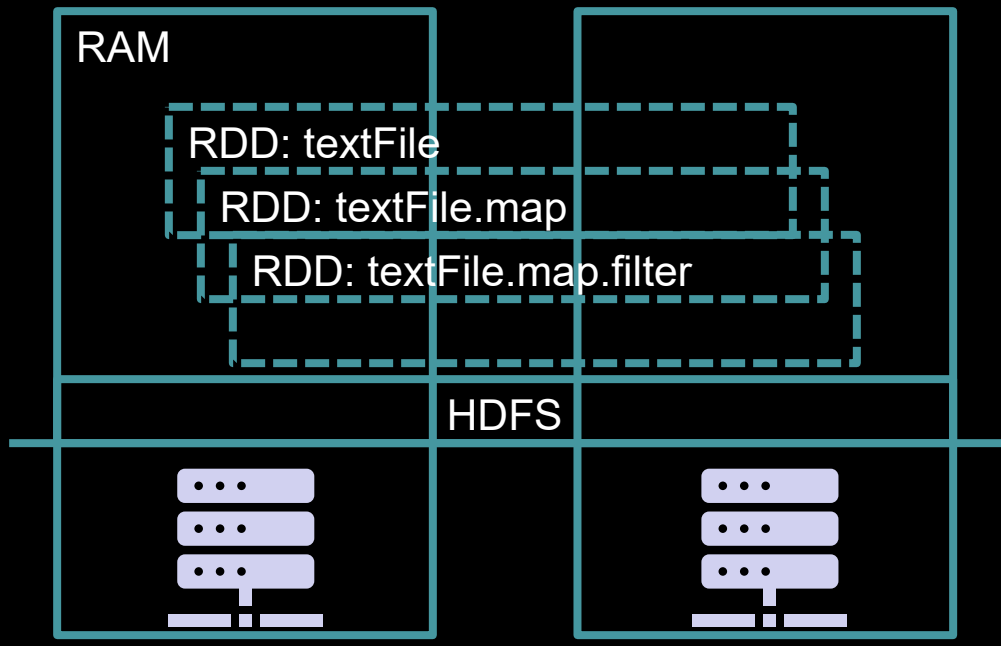
Tutte queste sono trasformazioni applicate al dataset, questo semplicemente crea una logica, i dati non sono di fatto processati



Spark – RDD Resilient Distributed Dataset

val x = sparkContext.textFile(_path_) ===> crea una istanza di RDD
val y = x.map(____) ===> trasformazione
val z = y.filter (____) ===> trasformazione

Il processamento avviene nella fase di azione

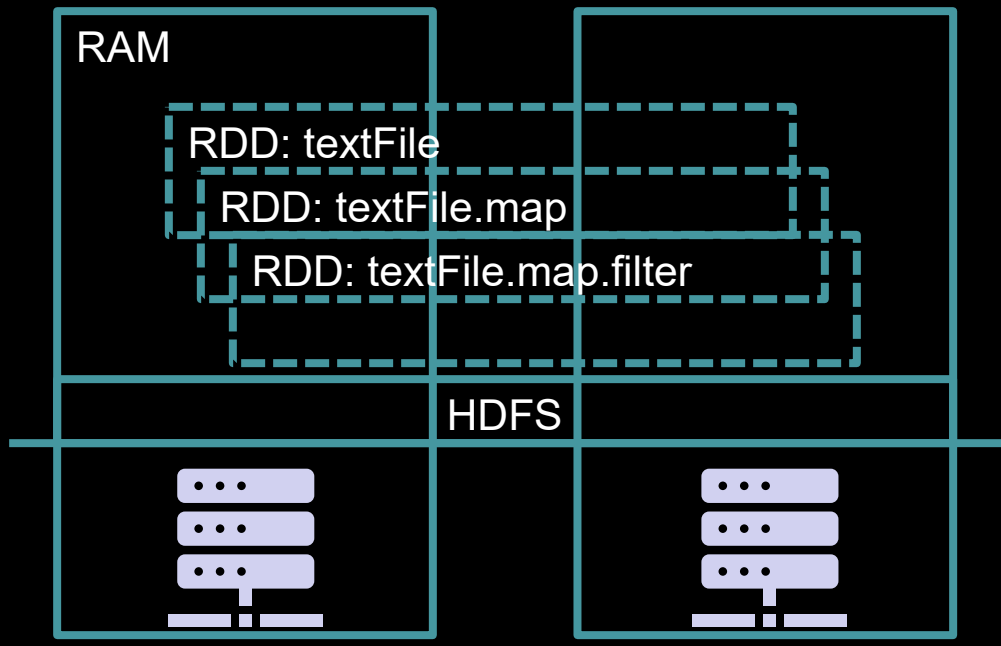


Spark – RDD Resilient Distributed Dataset

val x = sparkContext.textFile(_path_) ===> crea una istanza di RDD
val y = x.map(____) ===> trasformazione
val z = y.filter (____) ===> trasformazione

val c = z.count() ===> azione

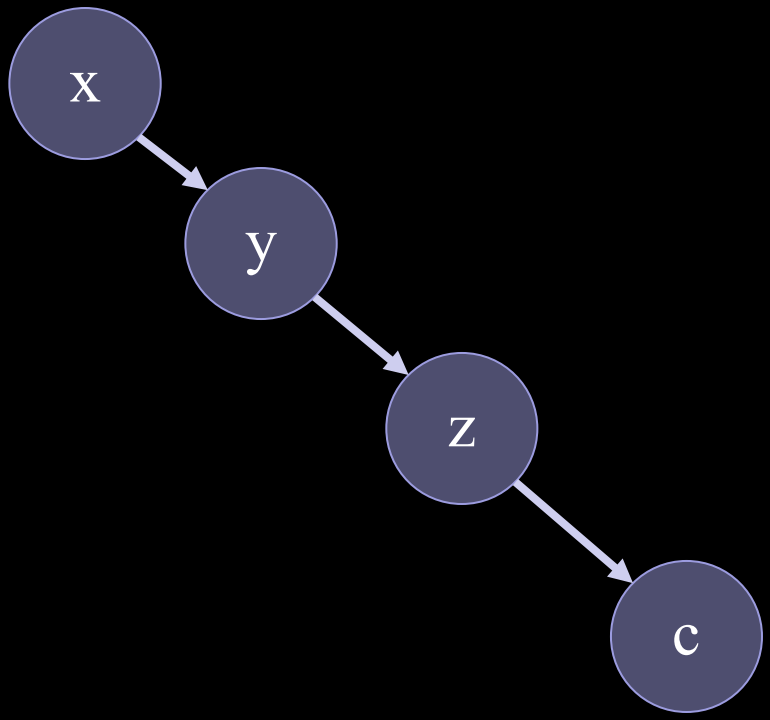
Il processamento avviene nella fase di azione

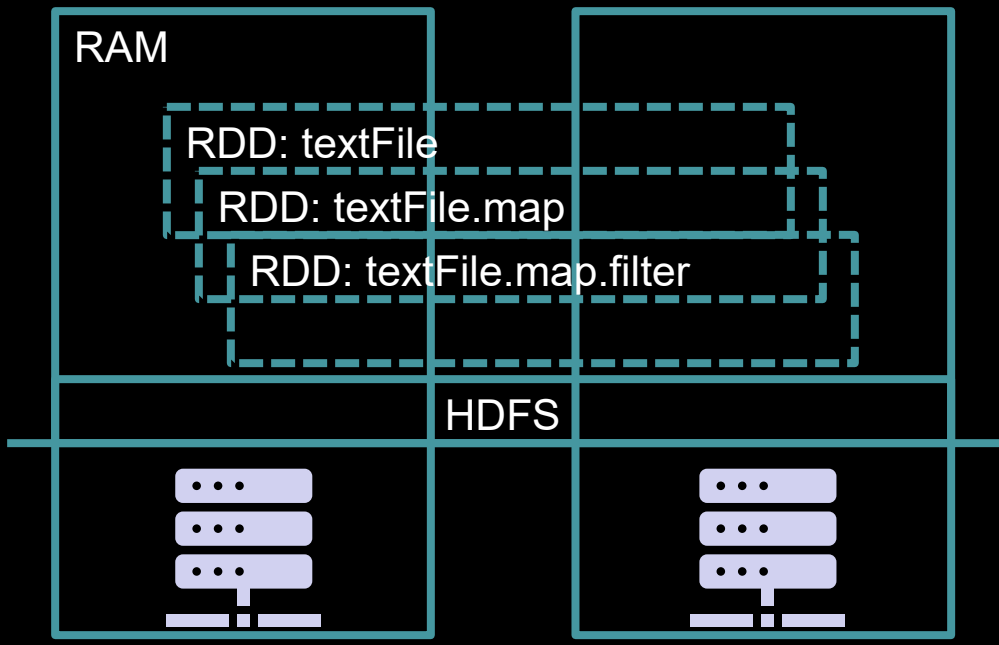


Spark – RDD Resilient Distributed Dataset

val x = sparkContext.textFile(_path_) ===> crea una istanza di RDD
val y = x.map(____) ===> trasformazione
val z = y.filter (____) ===> trasformazione

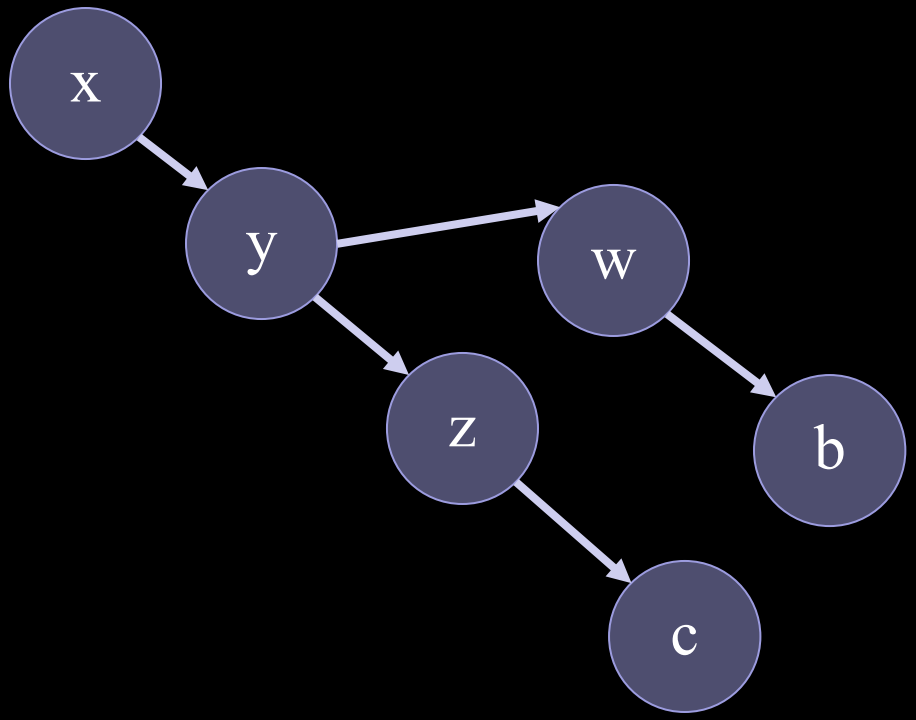
val c = z.count() ===> azione





Spark – RDD Resilient Distributed Dataset

val x = sparkContext.textFile(_path_)	===>	crea una istanza di RDD
val y = x.map(____)	===>	trasformazione
val z = y.filter (____)	===>	trasformazione
val w = y.filter2(____)	**===>**	**trasformazione**
val c = z.count()	===>	azione
val b = w.count()	**===>**	**azione**



val w = y.filter2(____)

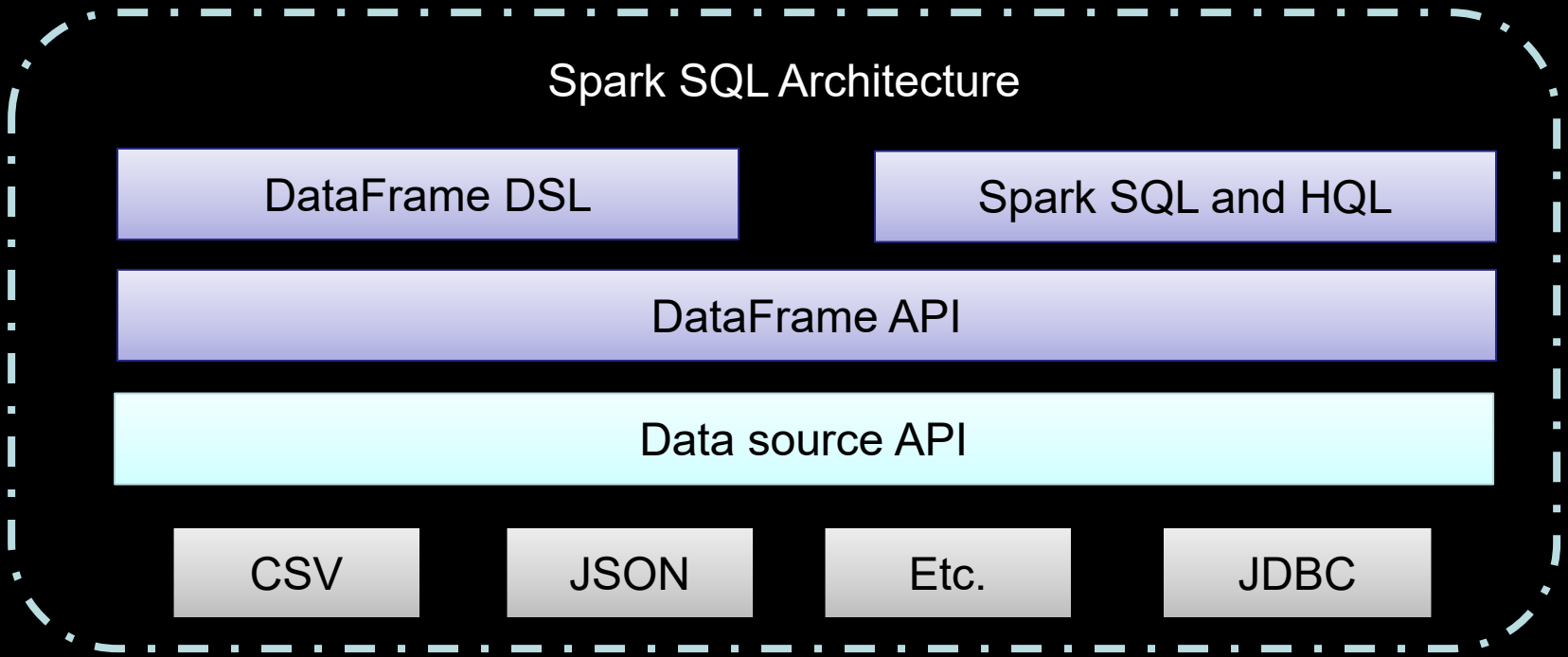
viene eseguita quando
chiamo l'azione w.count()

Spark – SQL

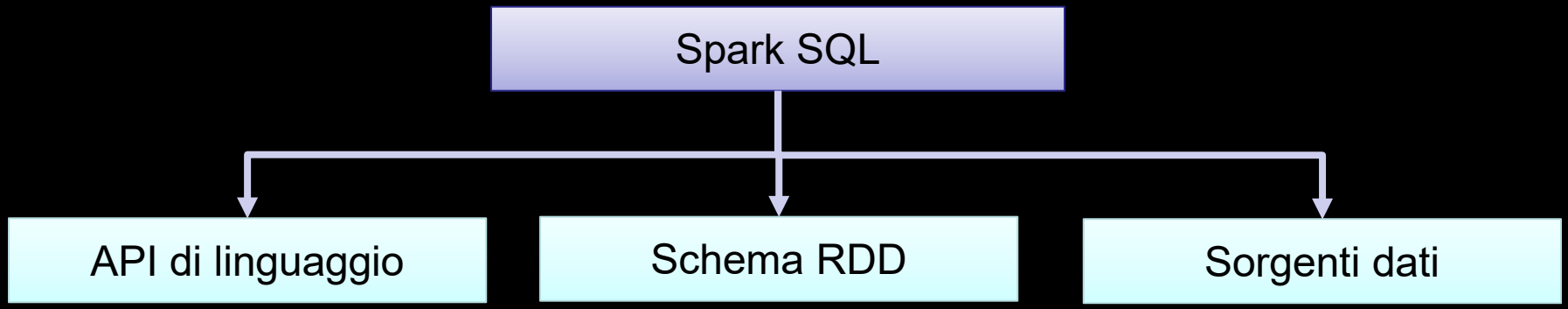
Spark SQL è un componente usato per processare dati strutturati e semi-strutturati in Spark.

Spark – SQL

Spark SQL è un componente usato per processare dati strutturati e semi-strutturati in Spark.

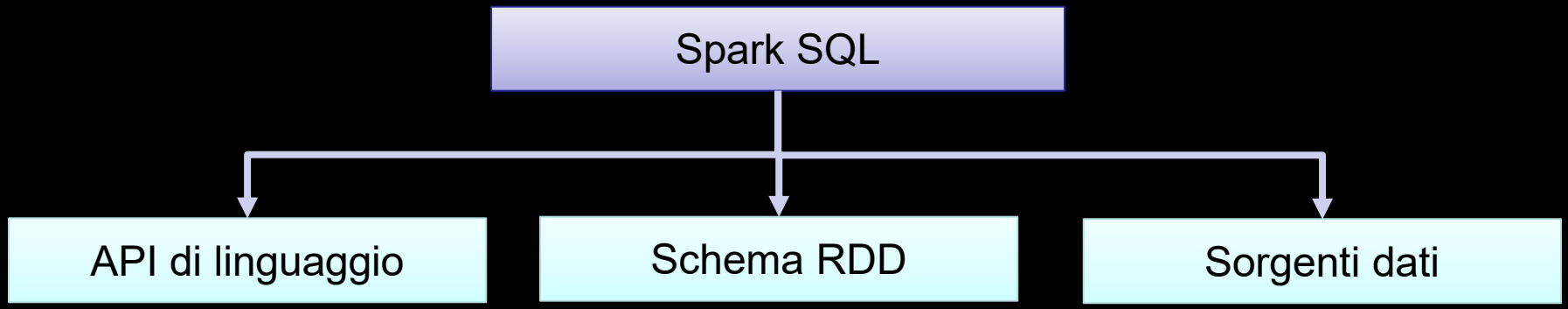


Livelli di spark – SQL



Interfaccia di Java, python, scala etc.

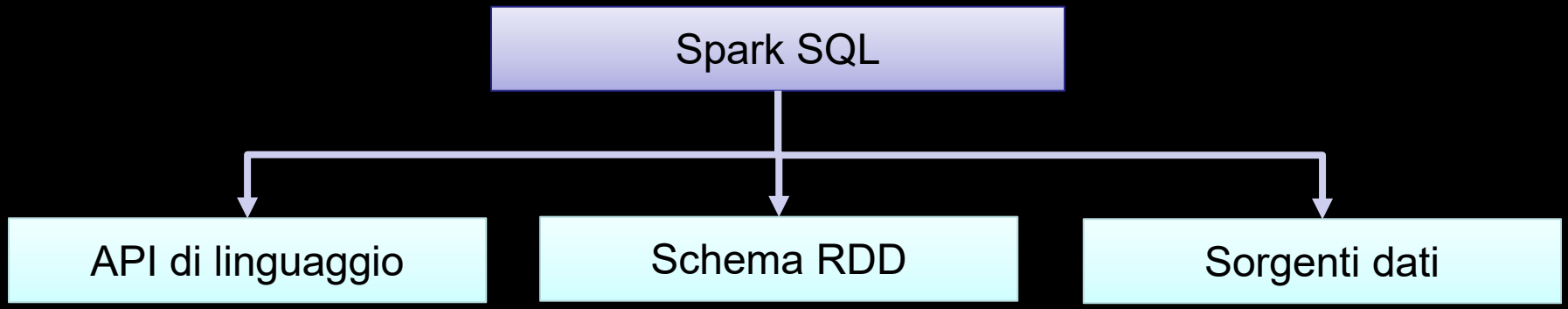
Livelli di spark – SQL



Funziona su schemi come tabelle e voci che possono essere usate per creare data frame e tabelle

SchemaRDD è un sistema di astrazione dati che funziona su dati strutturati e semi-strutturati

Livelli di spark – SQL



Diverse sorgenti di dati, CSV, JSON etc.

Spark – Streaming

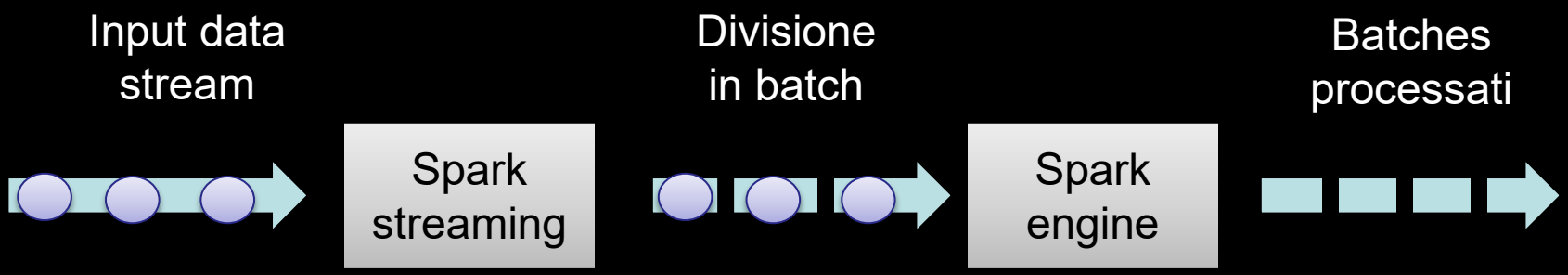
Spark streaming è una API che permette il processo di batch in tempo reale

Divide i dati in piccoli batch e invia il flusso permettendo il processamento a cadenza temporale definita o per evento

Spark – Streaming

Spark streaming è una API che permette il processo di batch in tempo reale

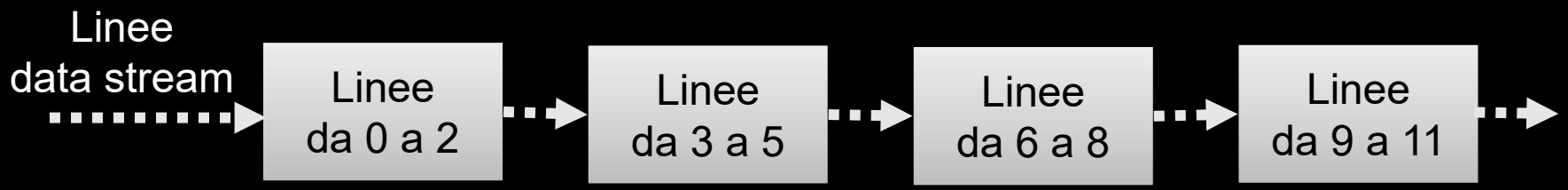
Divide i dati in piccoli batch e invia il flusso permettendo il processamento a cadenza temporale definita o per evento



Spark – Streaming

Spark streaming è una API che permette il processo di batch in tempo reale

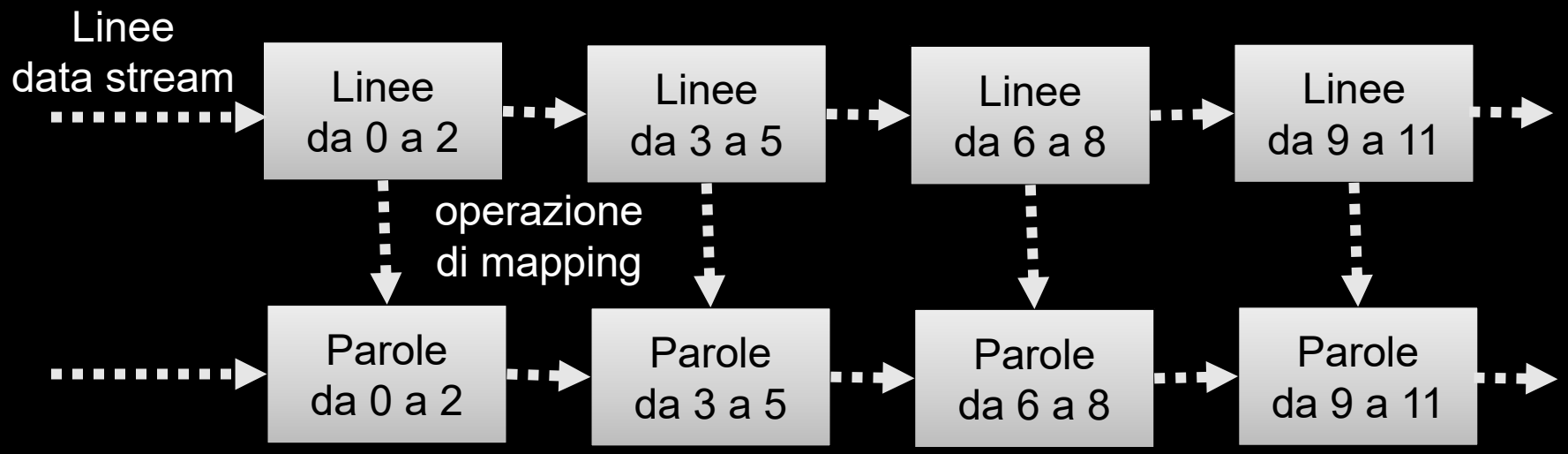
Esempio: operazione RDD di estrazione di parole da una linea di testo per uno streaming



Spark – Streaming

Spark streaming è una API che permette il processo di batch in tempo reale

Esempio: operazione RDD di estrazione di parole da una linea di testo per uno streaming



Spark – MLlib

Mllib è una libreria di machine learning di basso livello che opera su dati distribuiti sotto il framework spark

é responsabile per:

- ✓ è 9 volte più veloce della sua versione hadoop
- ✓ possiamo fare classificazione, clusterizzazione o filtraggio sui grandi dati

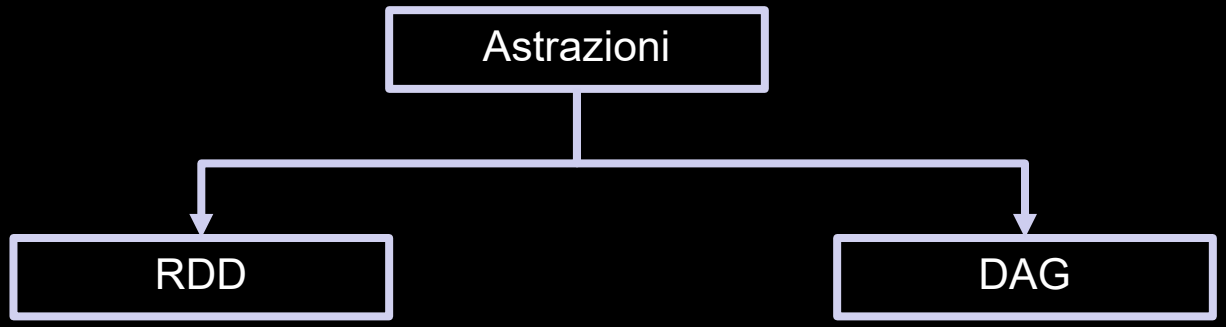
Spark – GraphX

GraphX è un framework di processamento dati su spak che fornisce una API ottimizzata per il calcolo e l'analisi dei dati (analytic)

é responsabile di:

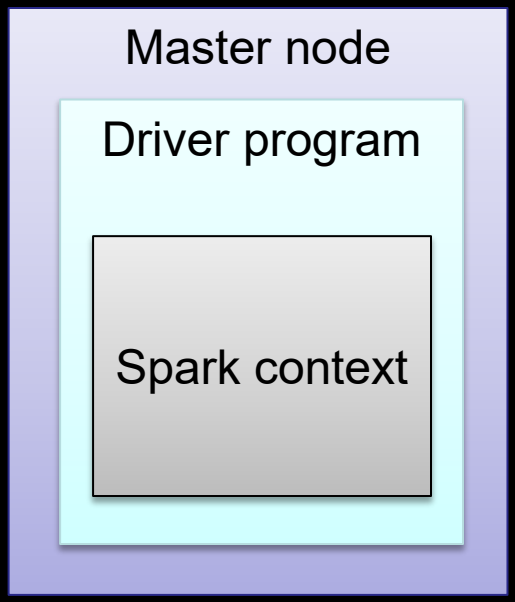
- ✓ Analisi dati esploratoria
- ✓ Computazione grafici interattivi
- ✓ Analisi delle connessioni tra i dati
- ✓ Strumento unificato per ETL (Extract, Transform and Load)

Spark – Architettura



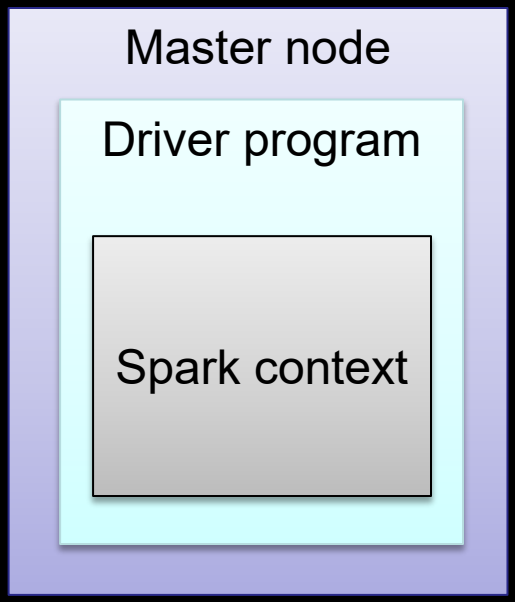
Spark – Architettura

Spark usa una architettura master-slave che consiste in un driver, eseguito sul master node, e un numero indefinito di esecutori (slave) distribuiti sui nodi del cluster



Spark – Architettura

Spark usa una architettura master-slave che consiste in un driver, eseguito sul master node, e un numero indefinito di esecutori (slave) distribuiti sui nodi del cluster



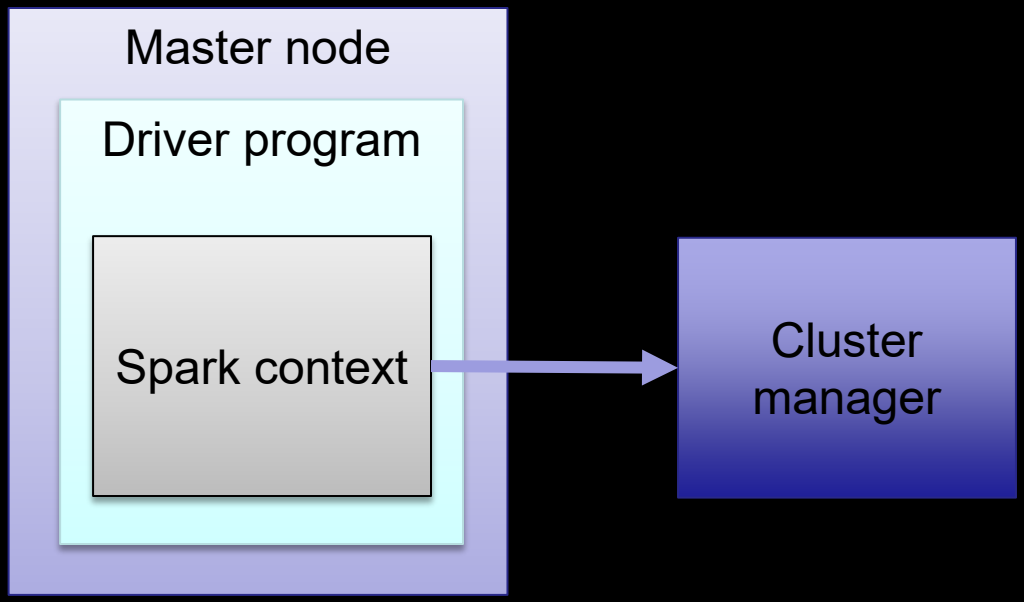
Il master node ha un driver program (obbligatorio)

il codice spark si comporta come un driver program e crea un context che rappresenta una interfaccia per tutte le funzionalità

Spark – Architettura

Le applicazioni spark eseguono insiemi di processi indipendenti sul cluster

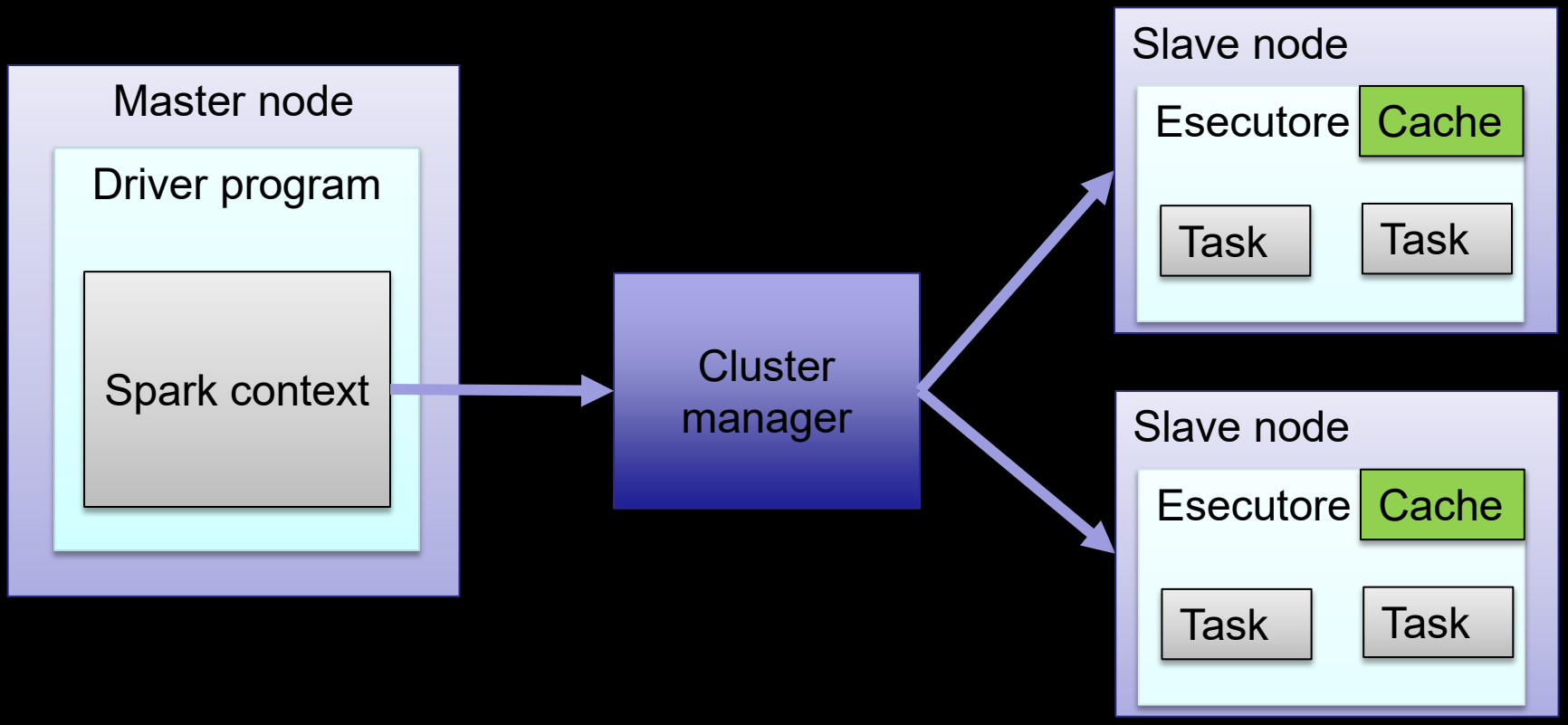
Il driver program e lo spark context si occupano dell'esecuzione dei jobs nel cluster



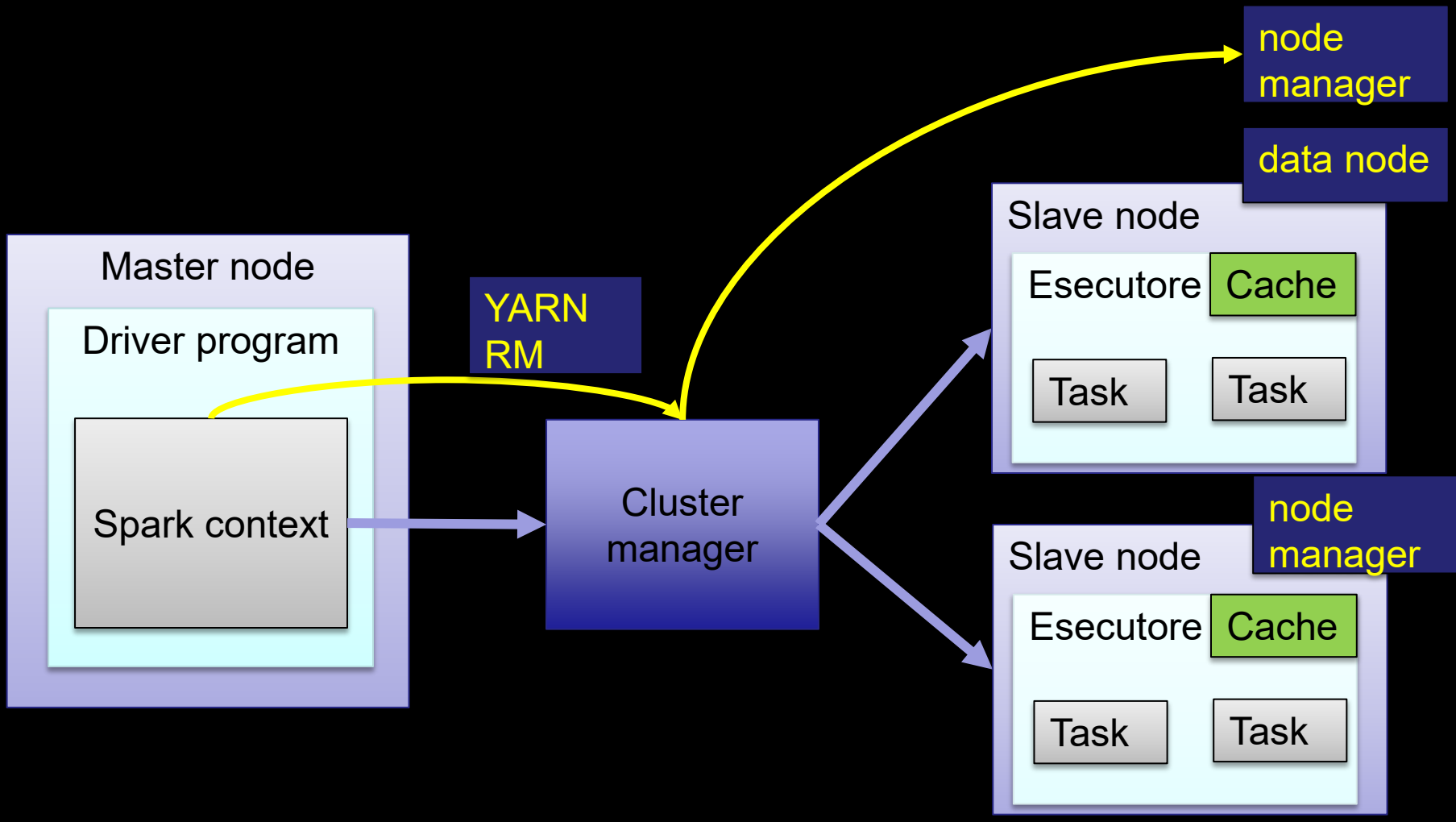
Spark – Architettura

Gli esecutori sono responsabili dell'esecuzione dei task

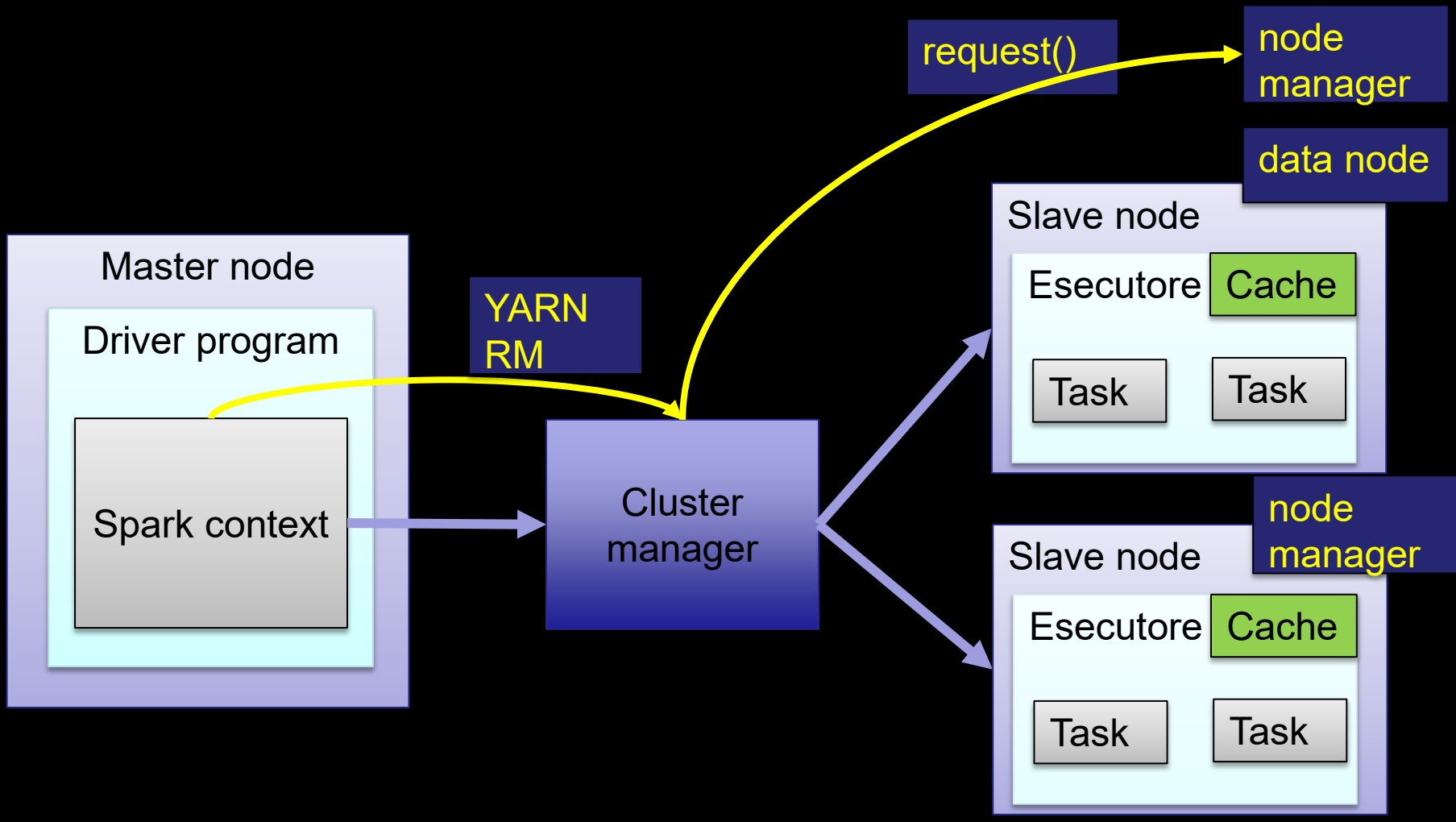
Gli slave nodes eseguono i task assegnati dal cluster manager e ritornano il risultato allo spark context



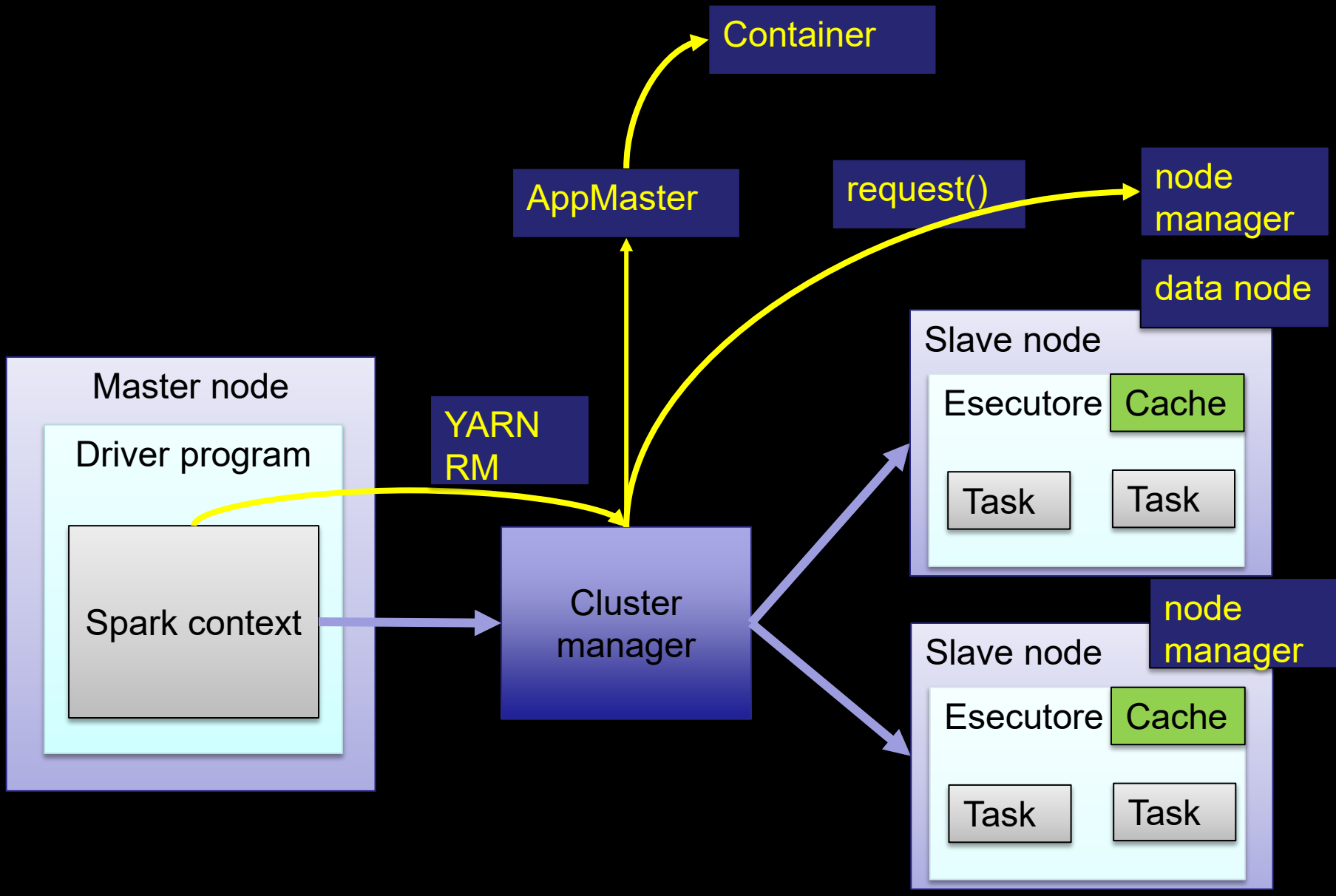
Spark – Interazione con YARN



Spark – Architettura



Spark – Architettura



Spark cluster managers

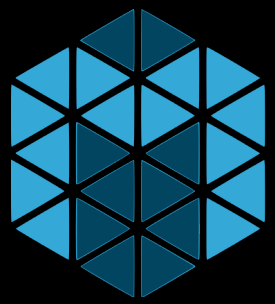
- ✓ Modalità Standalone

Tutte le applicazioni sottomesse in modalità standalone saranno eseguite in una coda FIFO e tutte le applicazioni proveranno ad usare tutte le risorse disponibili

Spark cluster managers

- ✓ Modalità Standalone
- ✓ Mesos

Mesos è un progetto open source per la gestione dei cluster e può funzionare anche con hadoop



Apache MESOS™

Spark cluster managers

- ✓ Modalità Standalone
- ✓ Mesos
- ✓ Hadoop YARN

YARN è usato come cluster resource manager su hadoop

Spark cluster managers

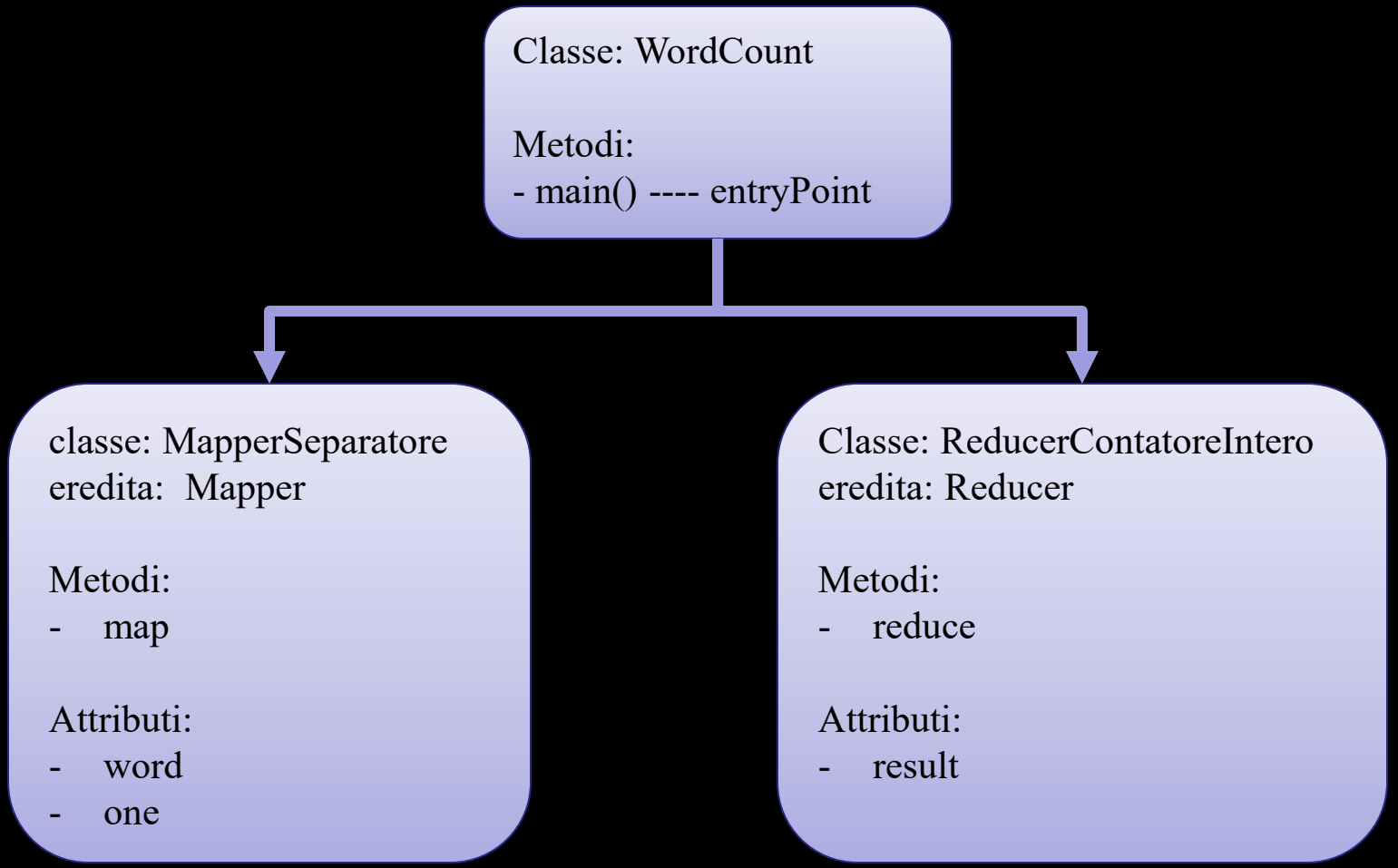
- ✓ Modalità Standalone
- ✓ Mesos
- ✓ Hadoop YARN
- ✓ Kubernetes

è un framework opensource per la gestione automatica di applicazioni in containers

Tutorial

- ✓ Conteggio parole

Esecuzione di un task map reduce – conteggio parole



Destrutturazione del codice

Esecuzione di un task map reduce – conteggio parole

Eseguiamo il file system di Hadoop e il relative resource manager

```
./sbin/start_dfs.sh
./sbin/start_yarn.sh
```

Compiliamo l'esempio di word count in codice

```
hadoop com.sun.tools.javac.Main WordCount.java
jar cf wc.jar ./WordCount*.class
hadoop jar wc.jar WordCount --/input --/output
```

Esaminiamo l'output

```
hdfs dfs -get output output
cat output/*
```

oppure

```
hdfs dfs -cat output/*
```

Tutorial – word count

SMS spam collection

https://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

Start SPARK

Iniziare il master (visible su http://localhost:8080)

start-master.sh

iniziare un worker

start-worker.sh spark://master:port

Start SPARK

Per assegnare specifiche risorse –c (core) –m (memoria)

```
start-worker.sh  -c 2 –m 512M spark://master:port
```

`pyspark` oppure `spark-shell`

aprono una shell nella quale è possible eseguire comandi diretti in python o scala