

# Hadoop

CORSO DI BIG DATA  
a.a. 2021/2022

Prof. Roberto Pirrone

# Sommario

- Caratteristiche generali
- Architettura generale
  - HDFS
  - YARN
  - MapReduce
- Ecosistema Hadoop
  - Hive
  - Pig

Immagini e diagrammi da: [data-flair.training](http://data-flair.training)

# Caratteristiche generali

- Hadoop è un'architettura software per i Big Data, sviluppata da Apache Foundation ed orientata nativamente all'elaborazione batch, che consta di tre componenti
  - File system distribuito – HDFS
  - Resource manager – YARN
  - Batch processor – MapReduce

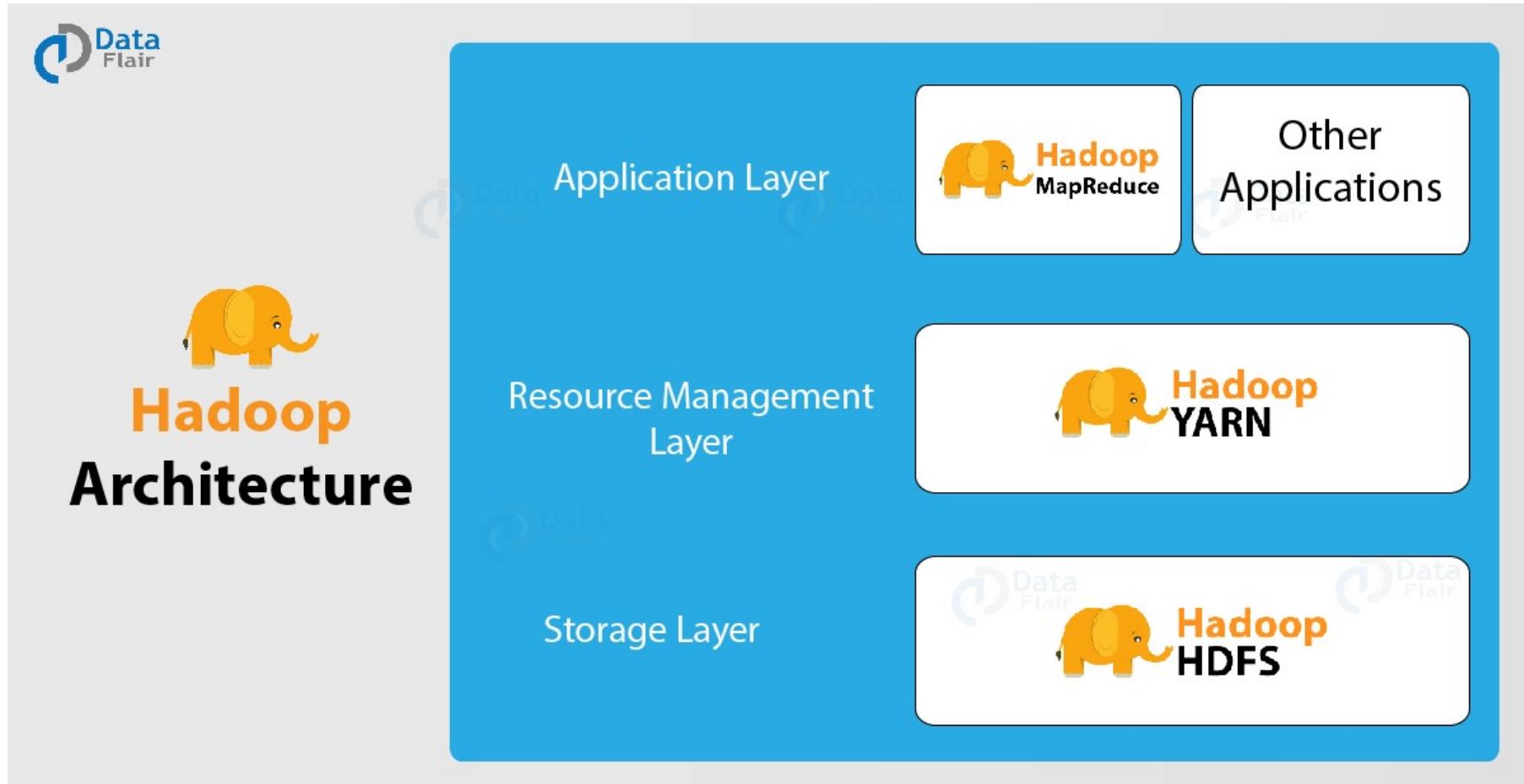
# Caratteristiche generali

- Hadoop si interfaccia con una serie di altri componenti software, sempre sviluppati da Apache, che ne estendono le funzionalità verso diversi campi di applicazione:
  - Stream processing
  - Workflow management
  - Interazione con RDBMS
  - Machine learning
  - ...
- Si parla di «ecosistema Hadoop» intendendo l'insieme di tutte queste componenti

# Caratteristiche generali

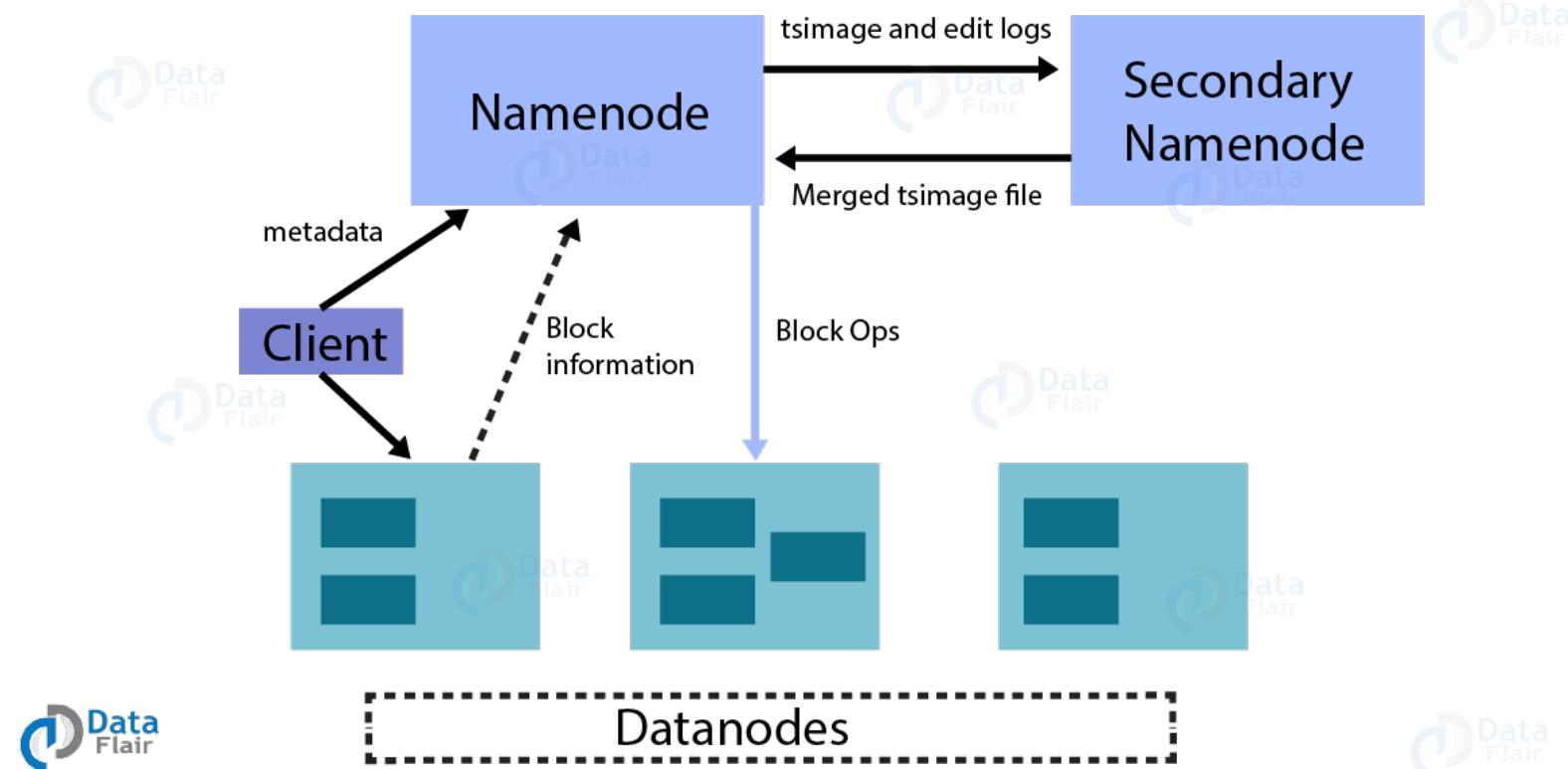


# Architettura generale



# Architettura generale

- HDFS
  - Distribuisce e replica i dati su un insieme di Datanode
  - Mantiene metadati e logo delle operazioni nei Namenode



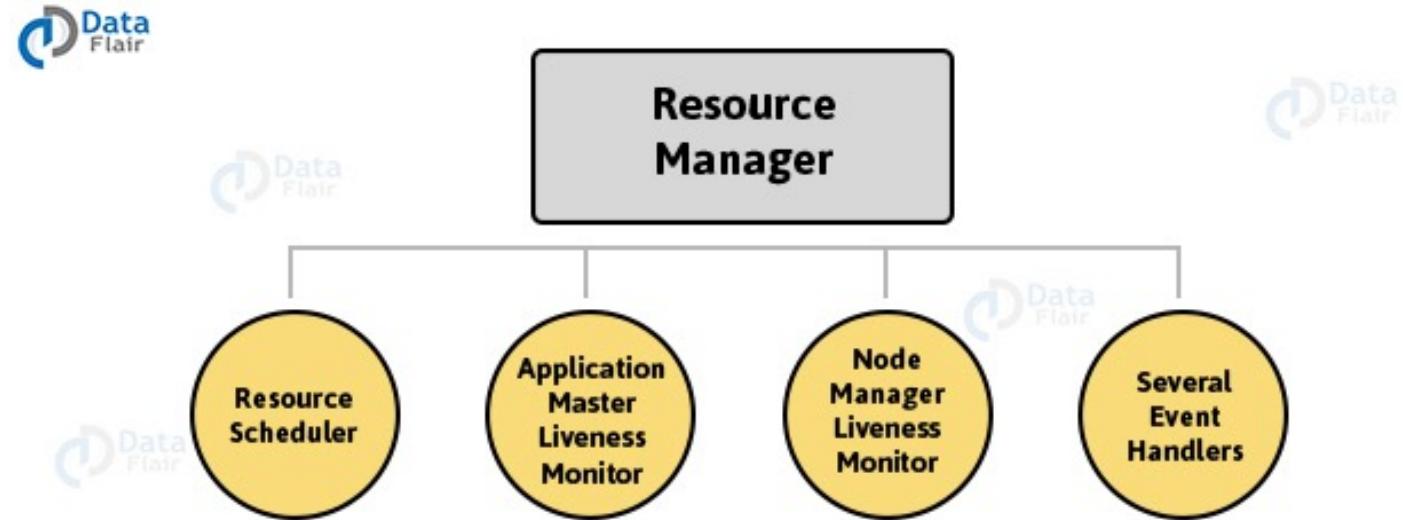
# Architettura generale

- YARN – Yet Another Resource Negotiator
- Tre componenti principali
  - Resource manager – gestore delle risorse del cluster
  - Node manager – gestore delle risorse del singolo nodo
  - Job submitter – esecutore del job

# Architettura generale

- YARN – Yet Another Resource Negotiator

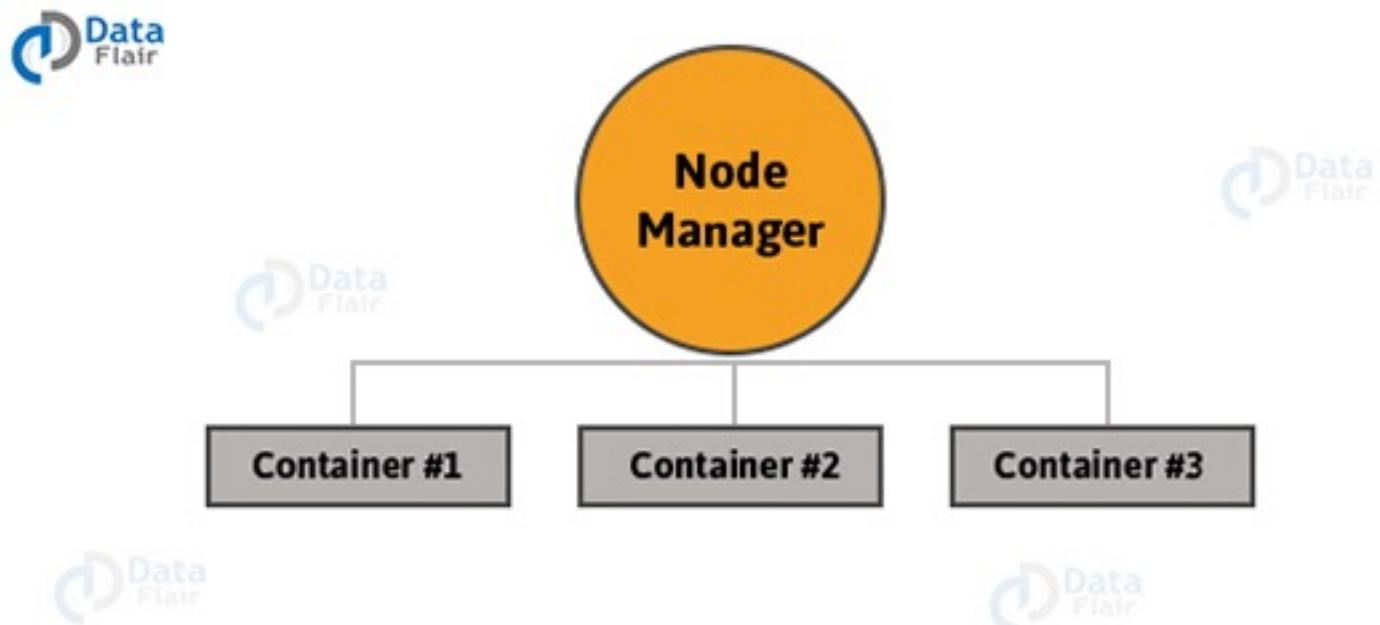
- Resource manager
  - Gestisce la rack awareness
  - Conosce le effettive risorse di ogni slave
  - Gestisce le risorse dei nodi slave (Res. Scheduler)
  - Il suo servizio di Application Manager negozia il primo *container* di ogni applicazione con i Node manager



# Architettura generale

- YARN – Yet Another Resource Negotiator

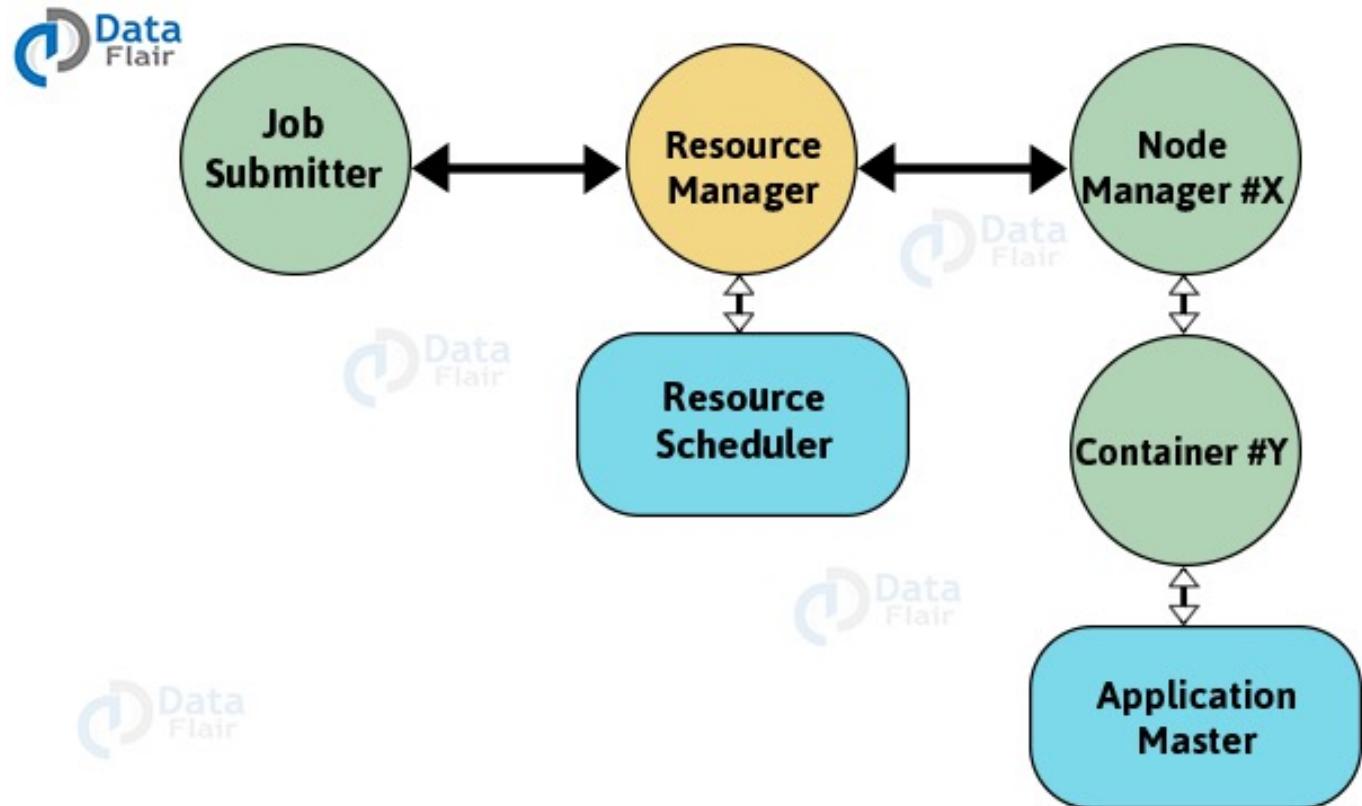
- Node manager
  - Esegue su ogni nodo
  - Gestisce i container: frazioni delle risorse computazionali del nodo
  - Controlla l'utilizzo delle risorse di ogni container
  - Invia il segnale di heartbeat al Resource manager



# Architettura generale

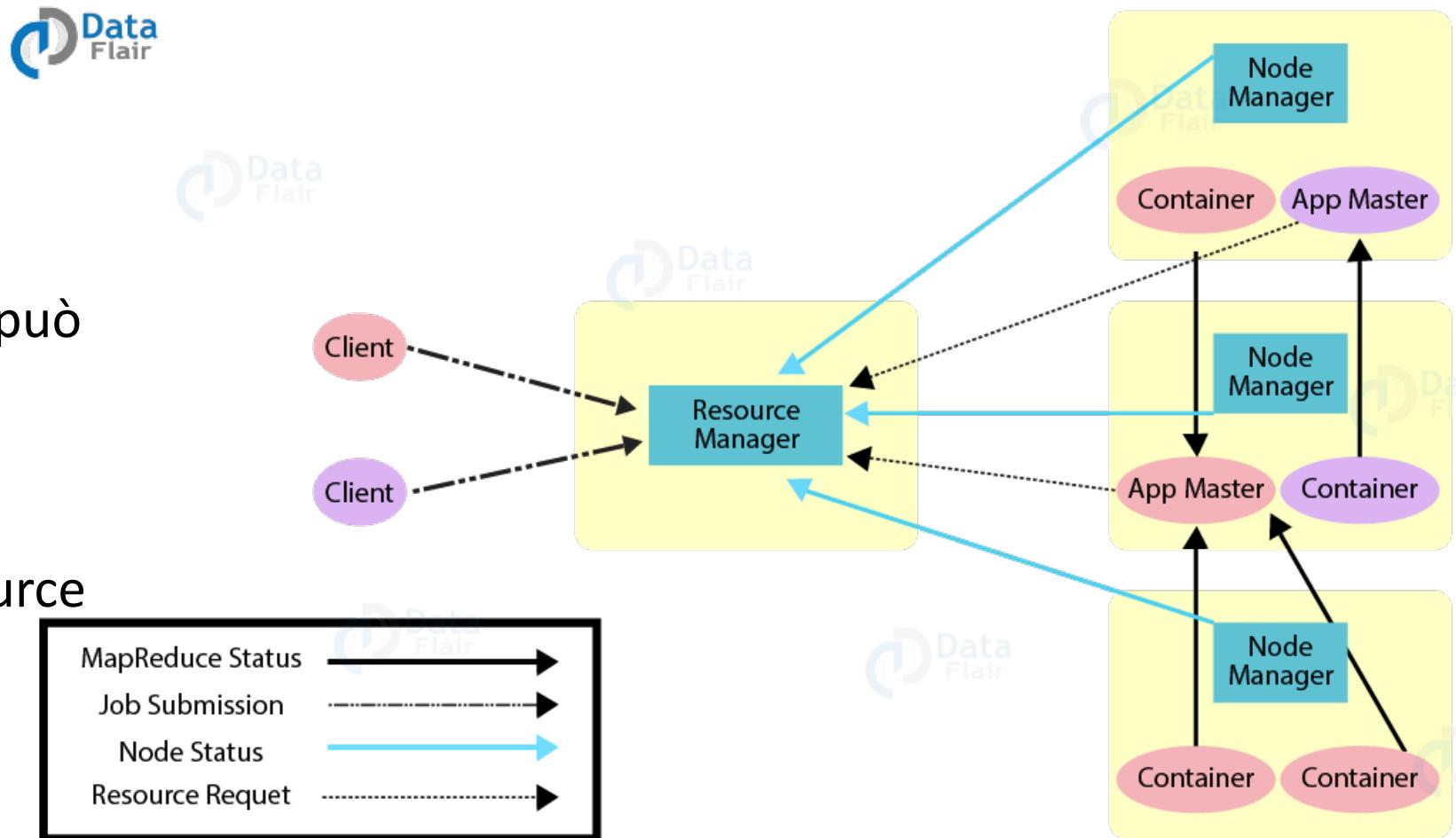
- YARN – Yet Another Resource Negotiator

- Job submitter
  - Sottomette il job al Resource manager
  - Resource manager e Resource scheduler negoziano il primo container per eseguire l'Application Master
  - Il Resource Manager contatta il nodo interessato per lanciare il container



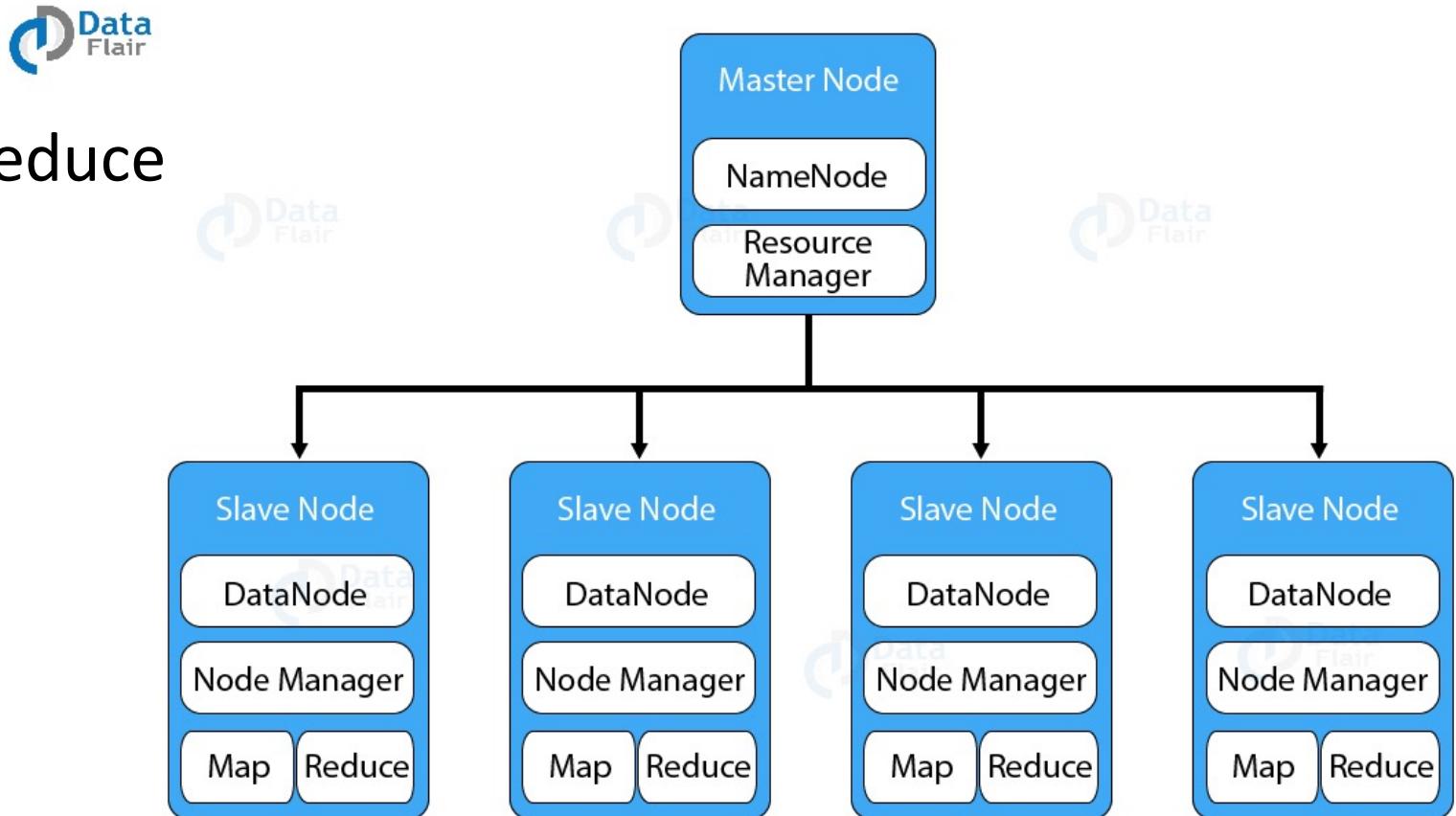
# Architettura generale

- YARN – Yet Another Resource Negotiator
  - L'Application Master può gestire più container
  - Le richieste vengono sempre fatte al Resource Manager



# Architettura generale

- YARN/HDFS/MapReduce

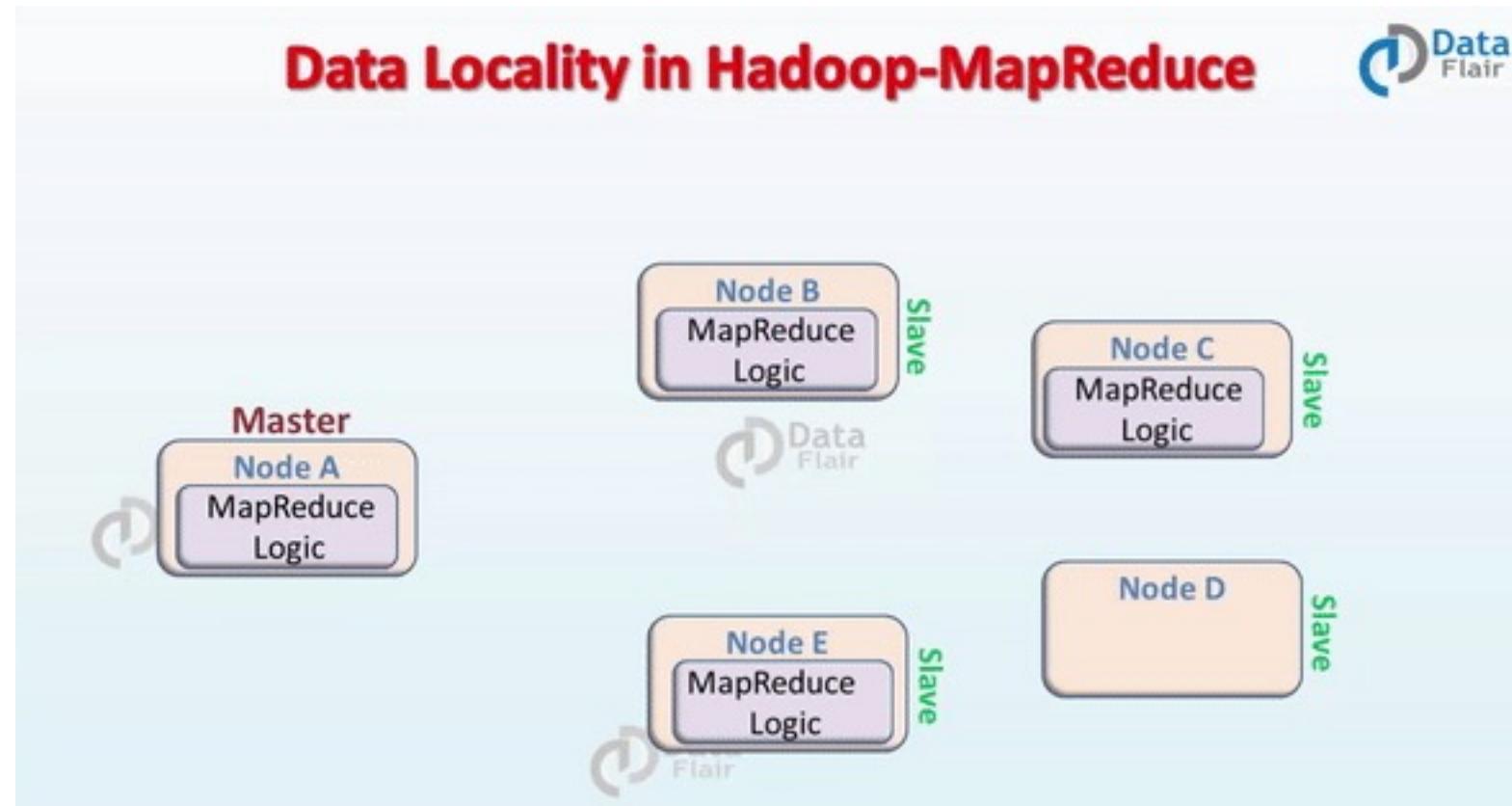


# Architettura generale

- MapReduce gestisce l'esecuzione di *programmi* sui dati presenti in HDFS e salva i risultati di nuovo in HDFS
- Ogni programma è costituito da due componenti
  - Mapper
  - Reducer
- Un task è l'esecuzione del mapper o del reducer su una partizione dei dati presenti su un nodo nel rispetto del principio di *data locality*

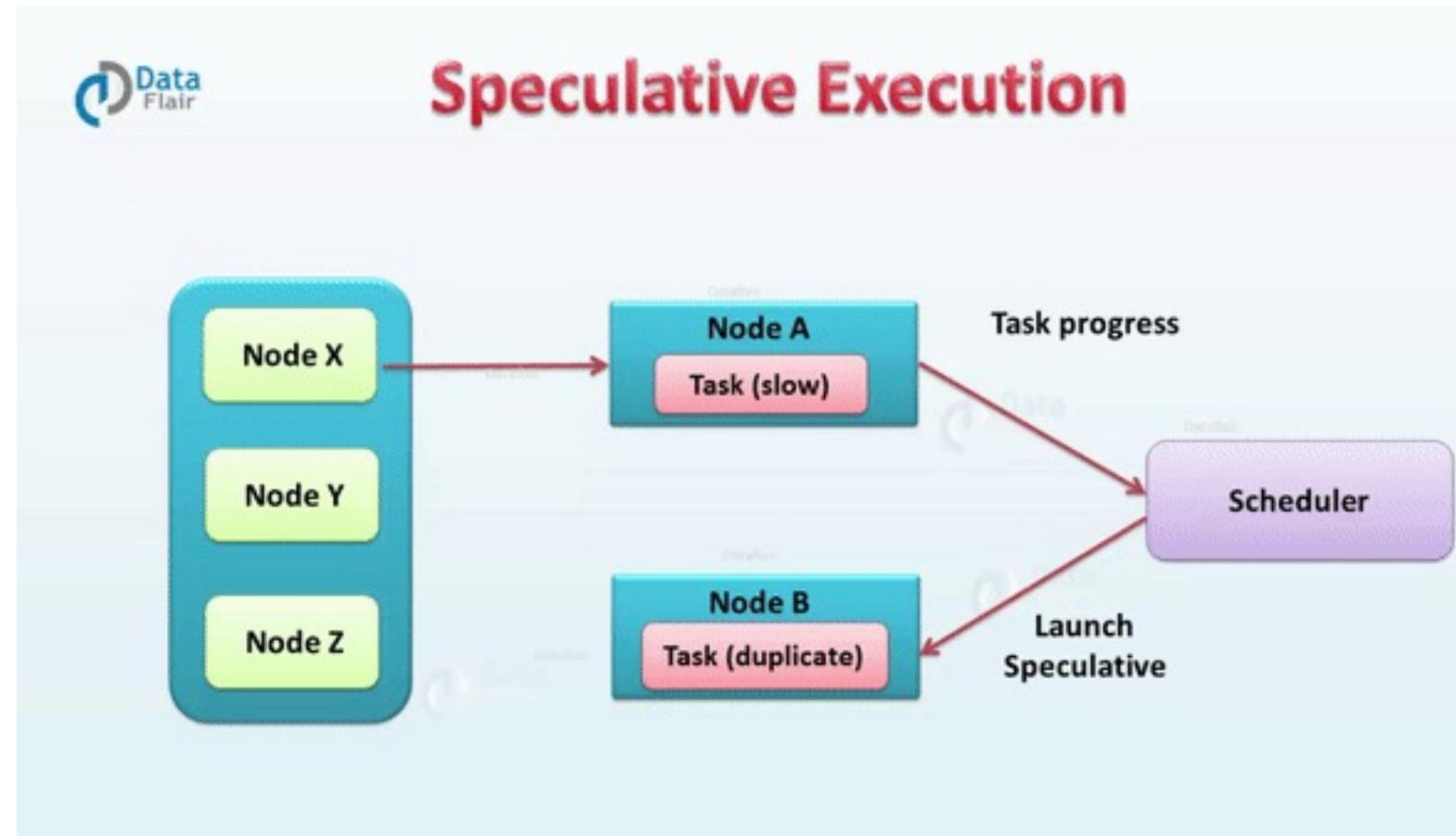
# Architettura generale

- I singoli task si generano perché il codice MapReduce viene fatto migrare sui vari nodi slave in cui si trovano i dati
- L'esecuzione di un task viene tentata (*TaskAttempt*) al più 4 volte e, se fallisce, il job viene considerato fallito



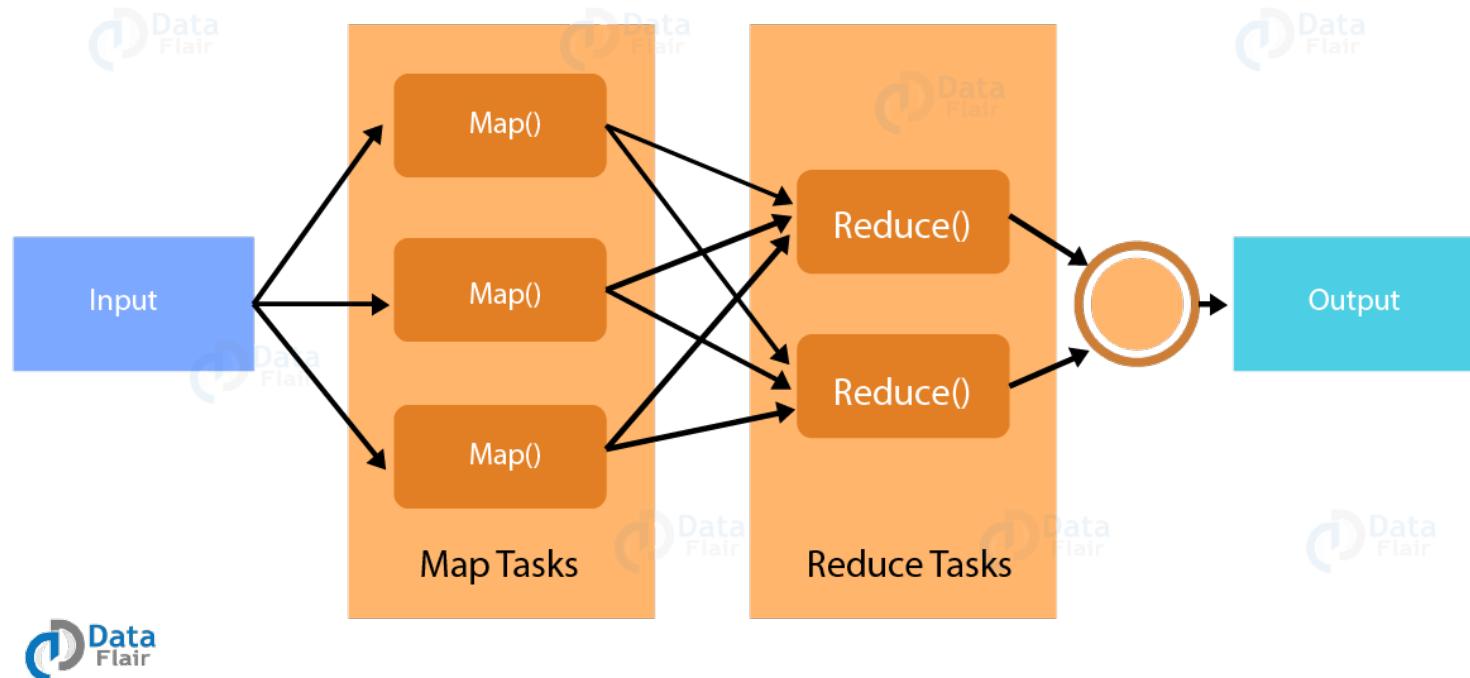
# Architettura generale

- Al fine di incrementare la performance, l'esecuzione speculativa rileva i task lenti e genera dei task di backup
- Quando il più veloce dei due termina l'altro viene cancellato



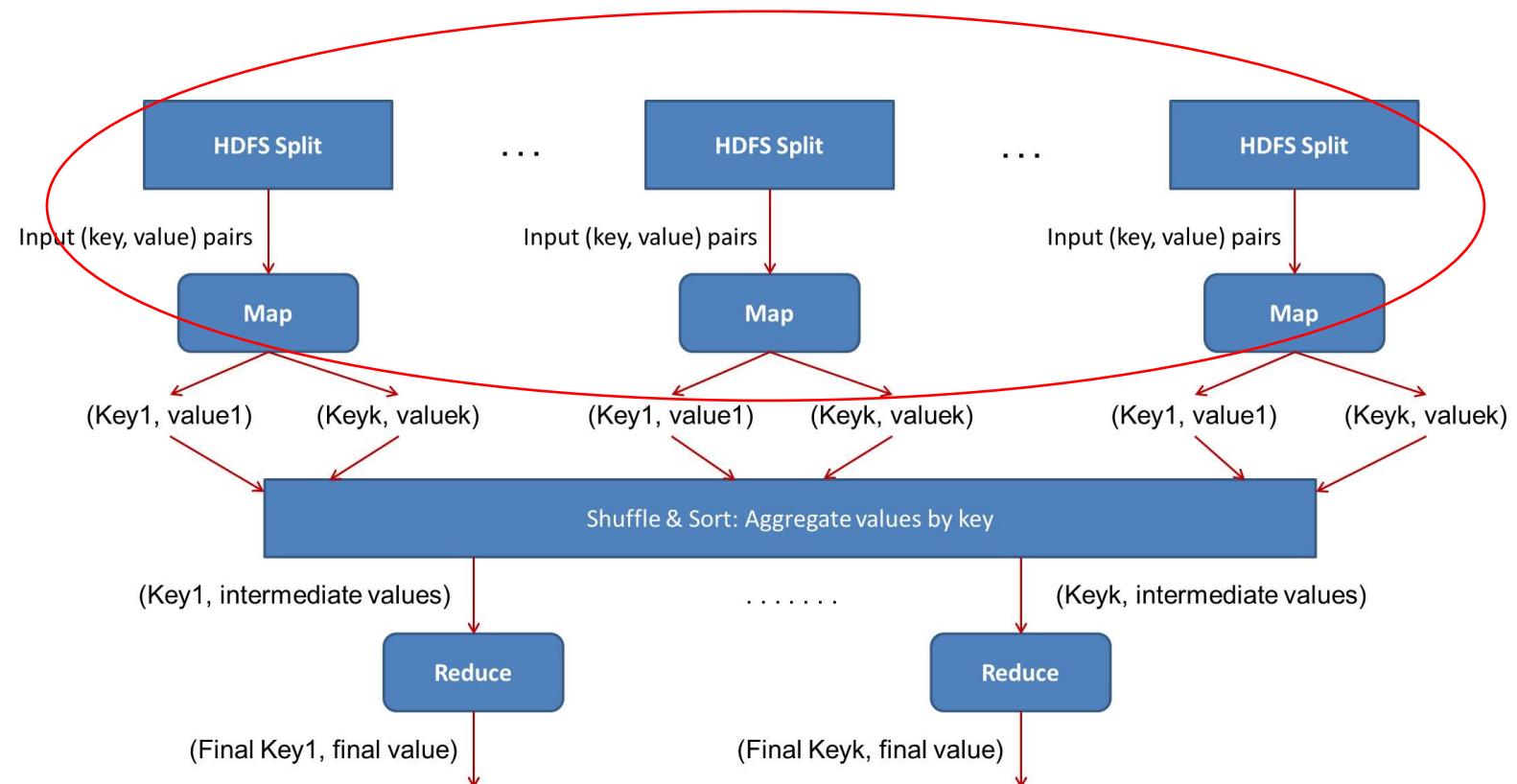
# Architettura generale

- I task di map e di reduce sono in genere su nodi distinti
- L'intero processo si svolge elaborando coppie chiave valore



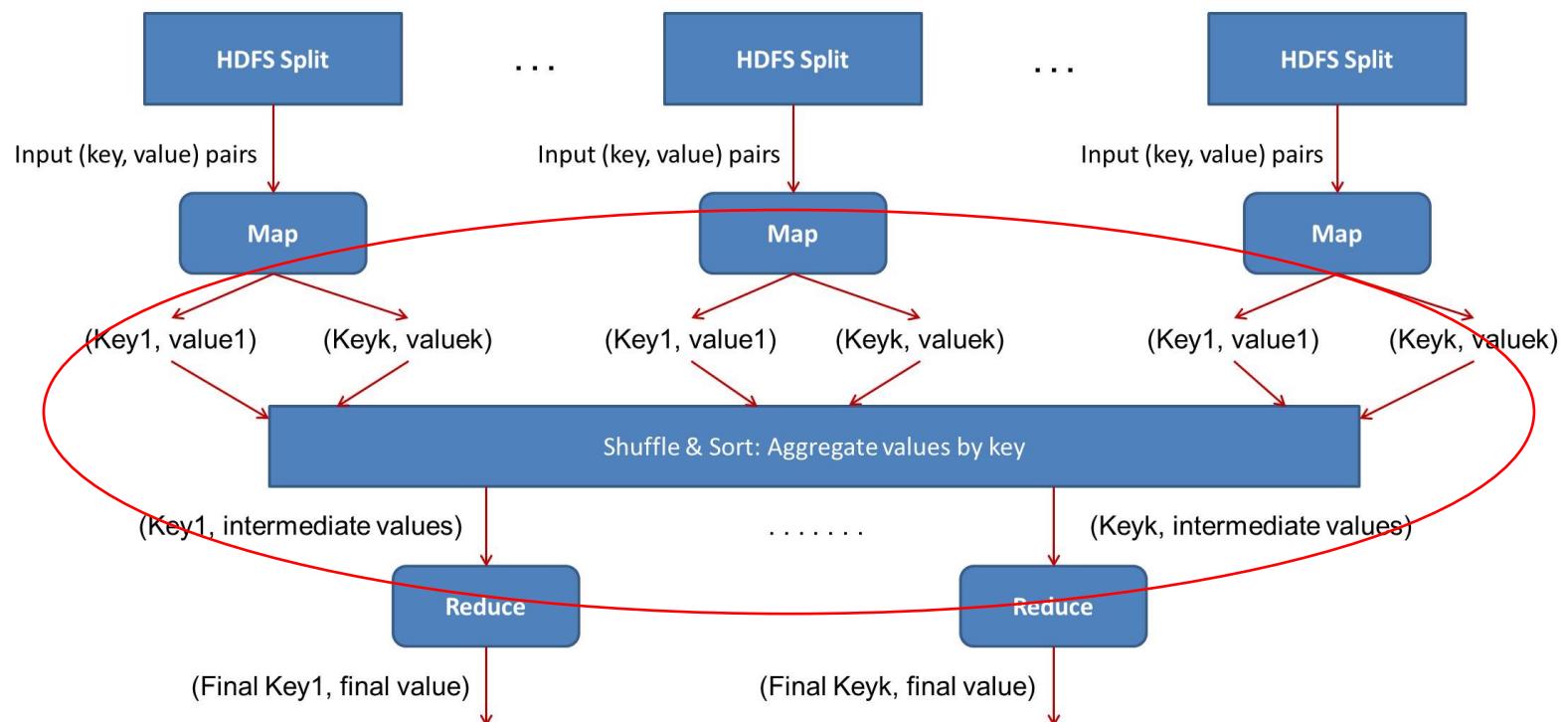
# Architettura generale

- HDFS esegue lo split dei dati, un blocco da 128MB per volta
- Ogni task di Map lavora su tutto il proprio input costituito da coppie chiave-valore che vengono trasformate in nuove coppie, secondo la logica programmata



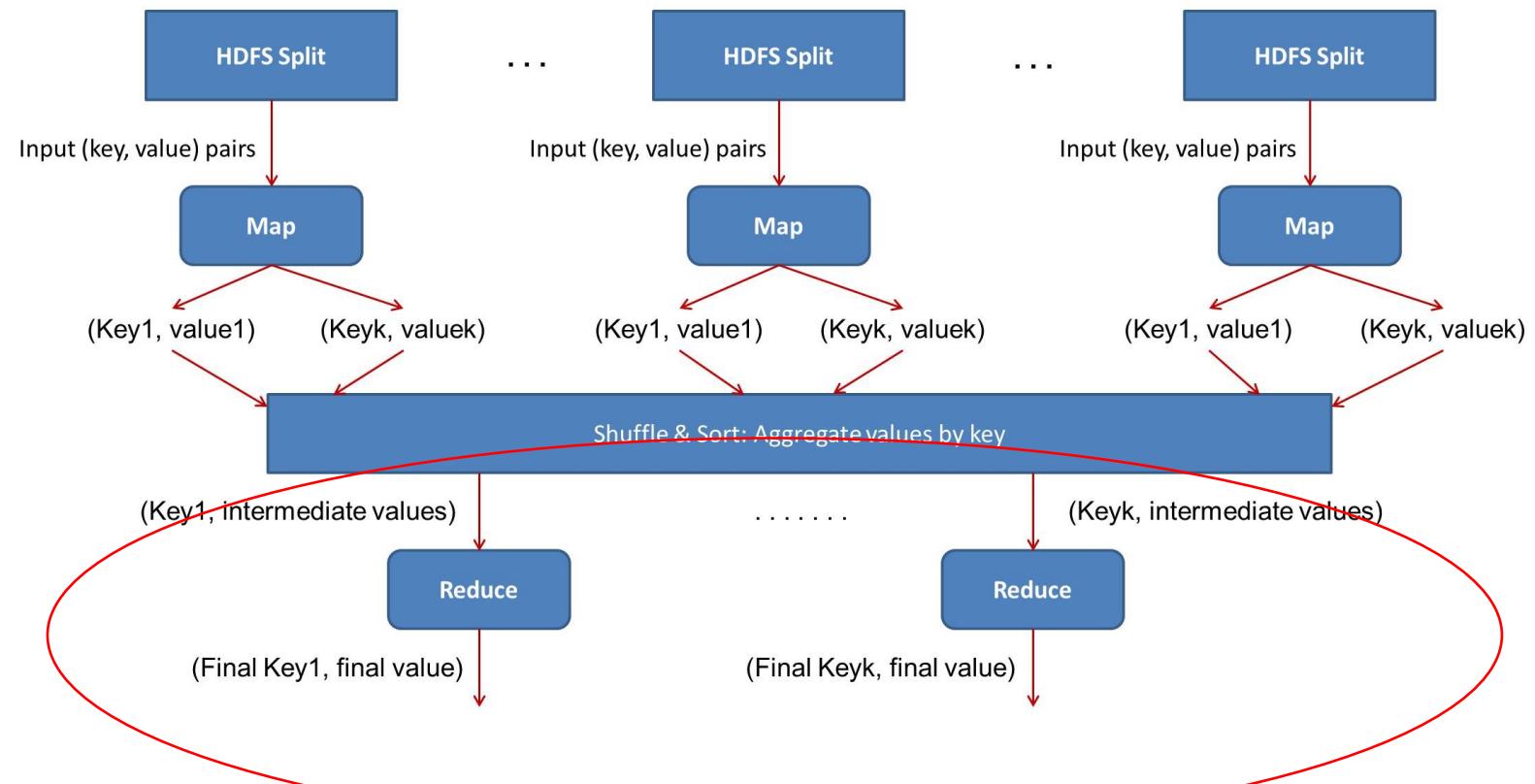
# Architettura generale

- I valori in uscita da tutti i task di Map sono processati da un layer di shuffle/sort che aggregano i dati per chiave
- Lo scopo è ridurre i nodi deputati ai task di Reduce



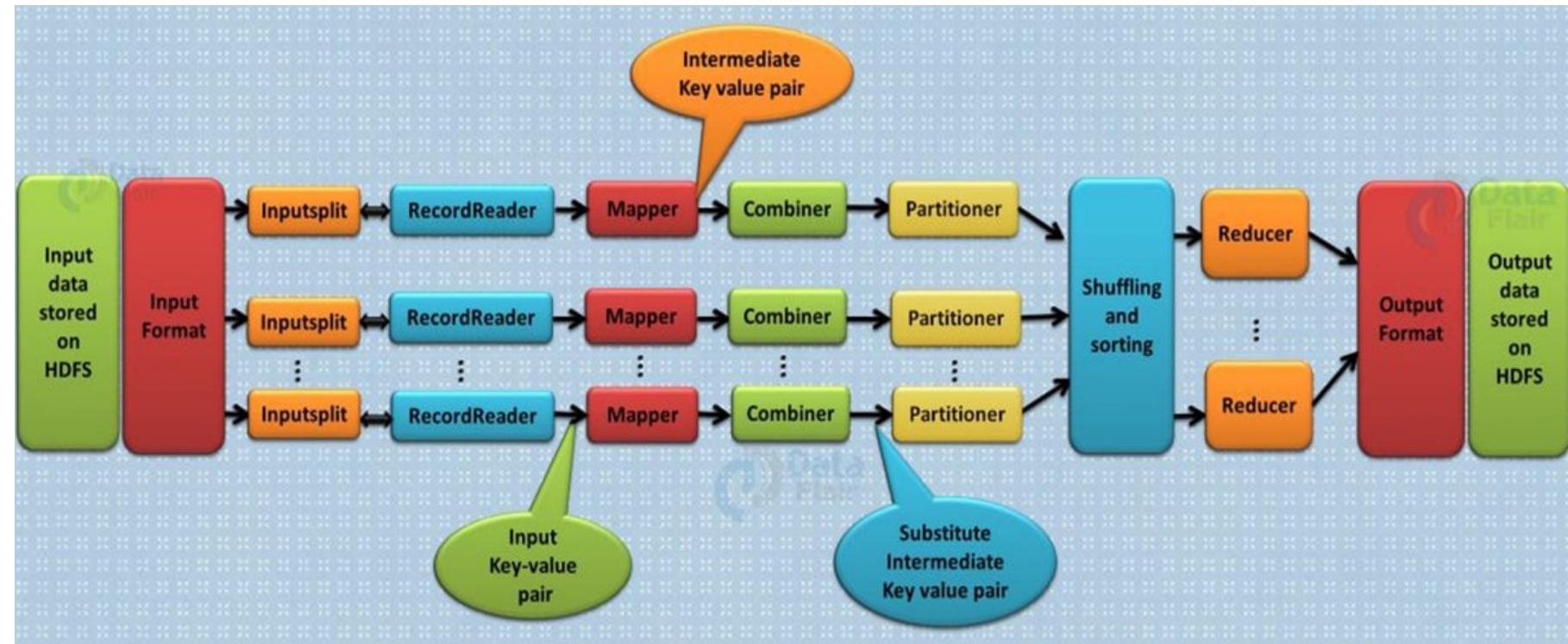
# Architettura generale

- Ogni task di Reduce genera una nuova lista di coppie secondo la propria logica programmata
- Il tipo delle coppie finali è diverso, in genere, dalle coppie intermedie
- Il risultato è salvato di nuovo in HDFS



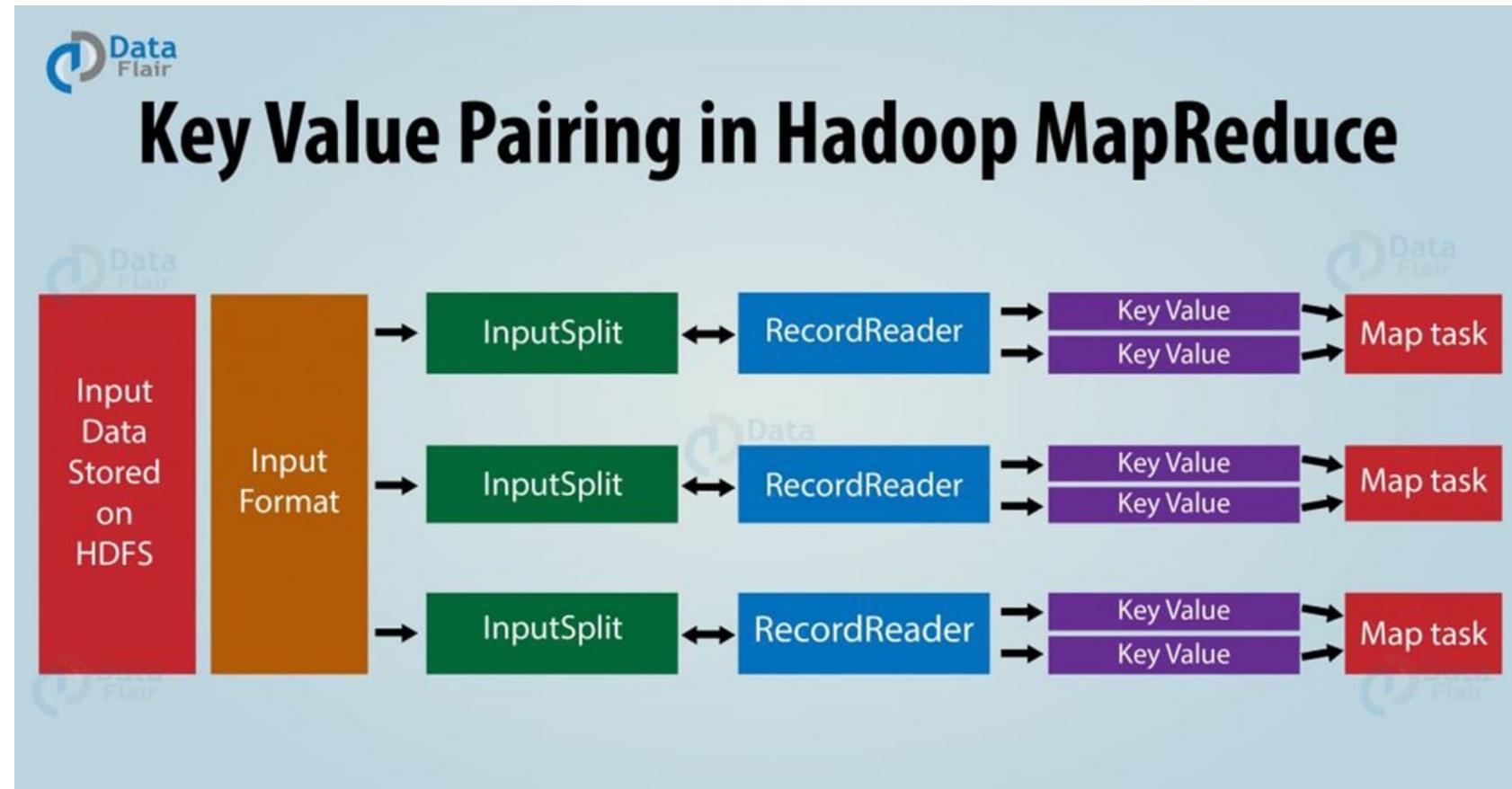
# Architettura Generale

- La pipeline di esecuzione viene mappata in un serie di classi Java
- Un programma MapReduce ha quindi una sua struttura fissa all'interno della quale si introduce la logica vera e propria



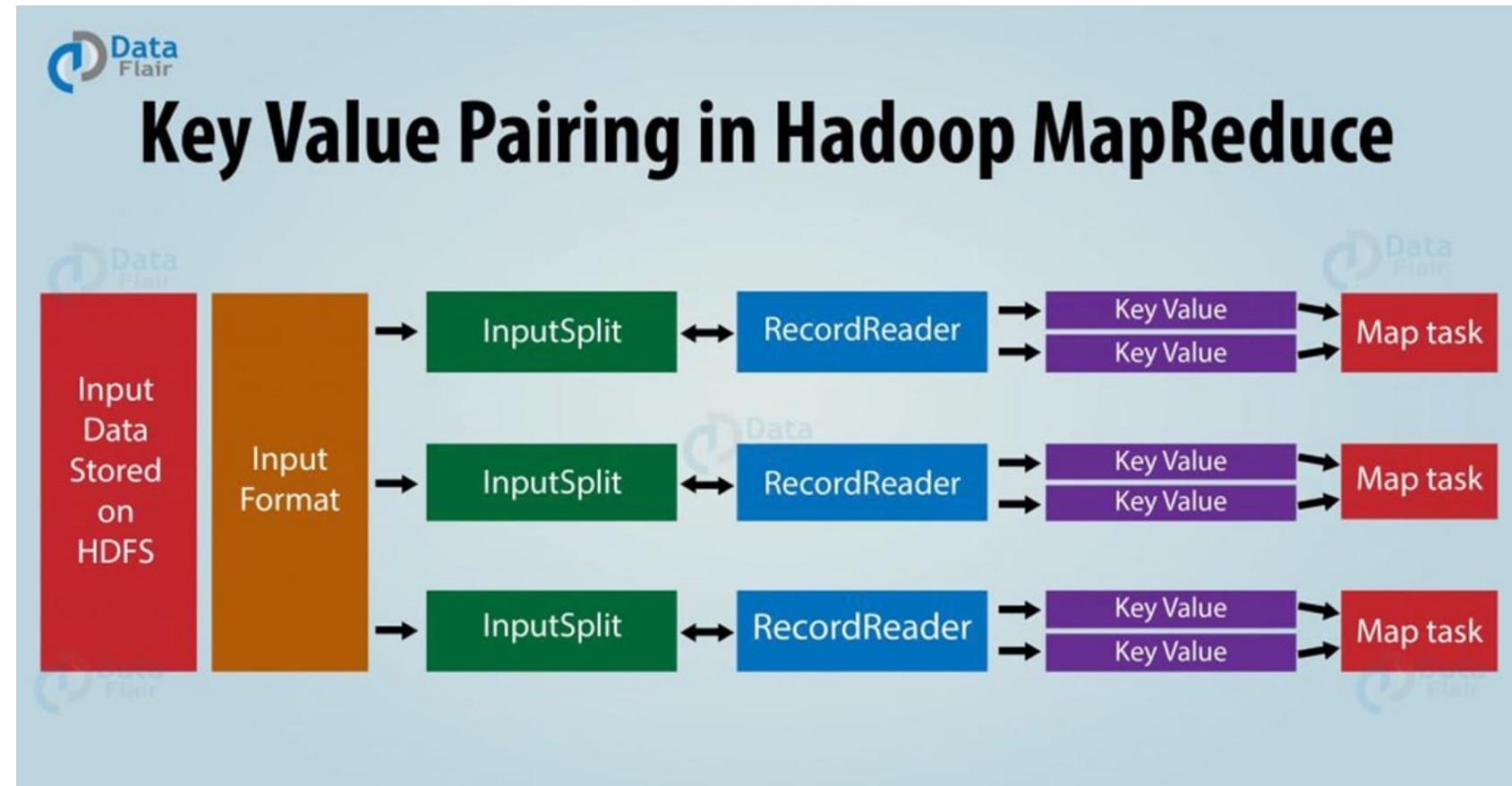
# Architettura generale

- Le coppie chiave-valore sono il meccanismo su cui si basa l'elaborazione MapReduce
- Sono definite dall'utente al fine di analizzare i dati, ***non sono*** proprietà intrinseche dei dati stessi
- Ci sono diversi formati di input ed output gestiti da MapReduce che possono essere usati dall'utente



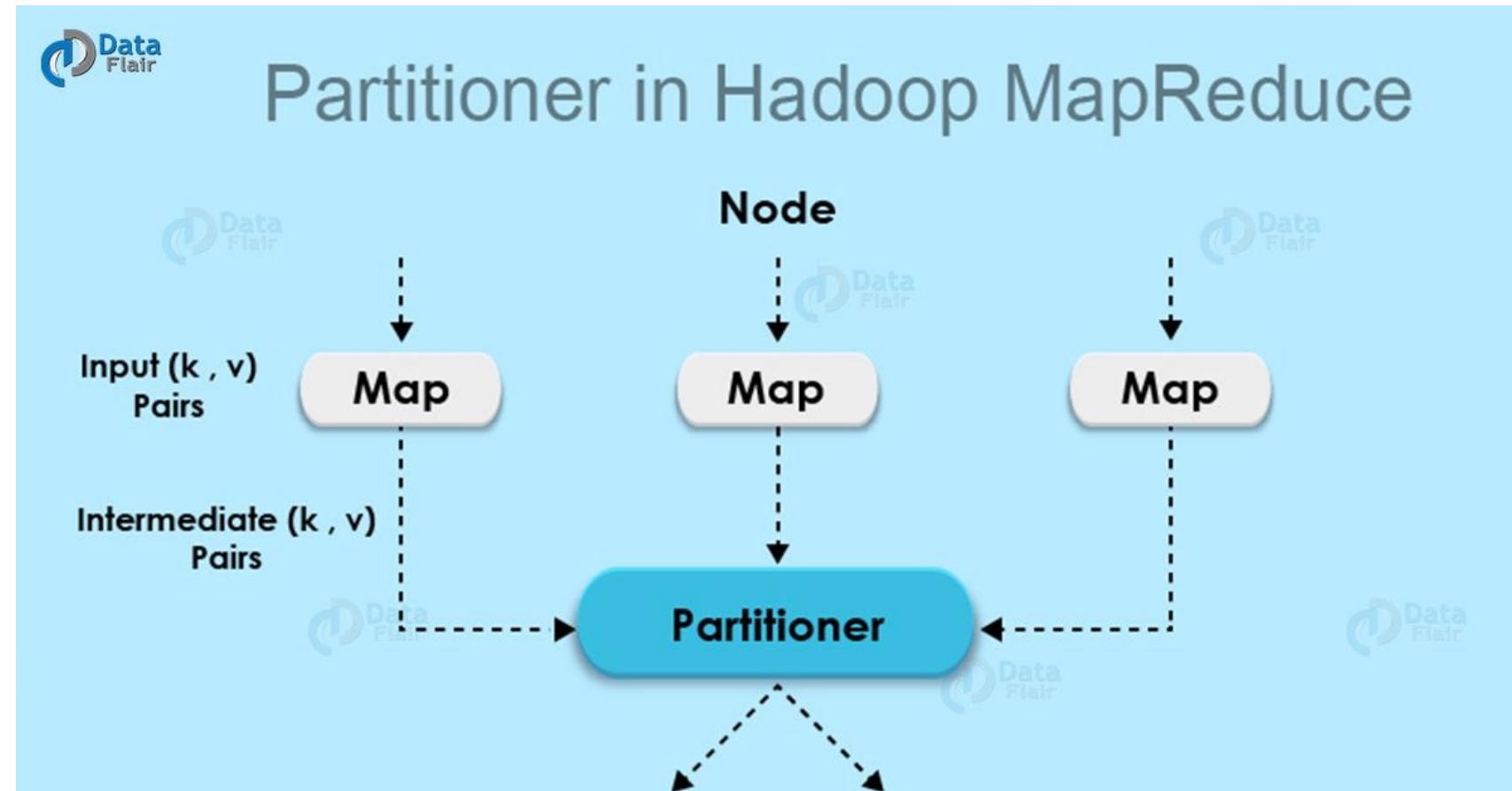
# Architettura generale

- Map
  - Input:  $(K_1, V_1)$
  - Output: list( $K_2, V_2$ )
- Reduce
  - Input:  $\{(K_2, \text{list}(V_2))\}$
  - Output: list( $K_3, V_3$ )



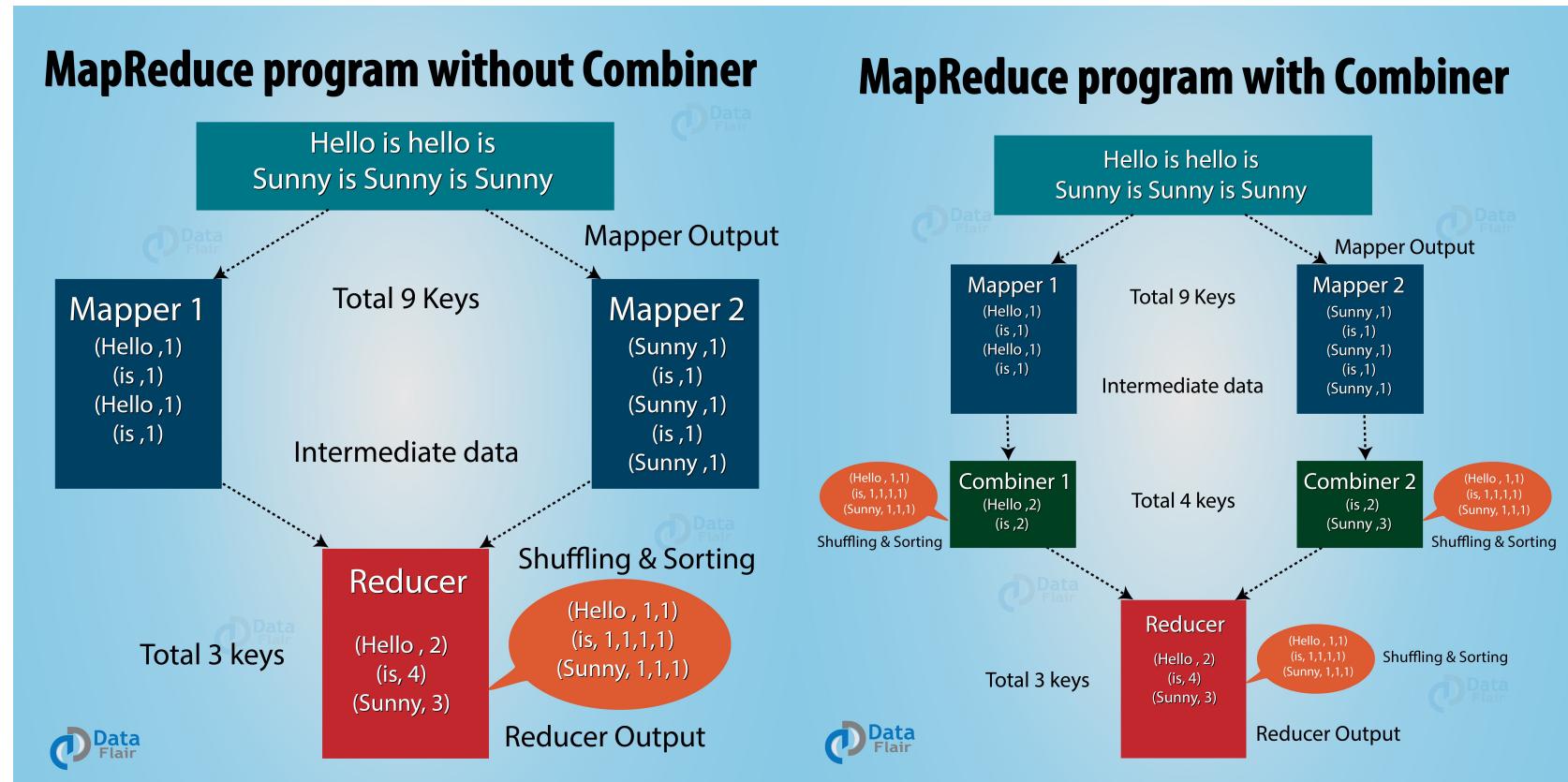
# Architettura generale

- Il Partitioner è il componente che crea le partizioni di valori da inviare ai task di Reduce usando una tecnica di hashing sulla chiave



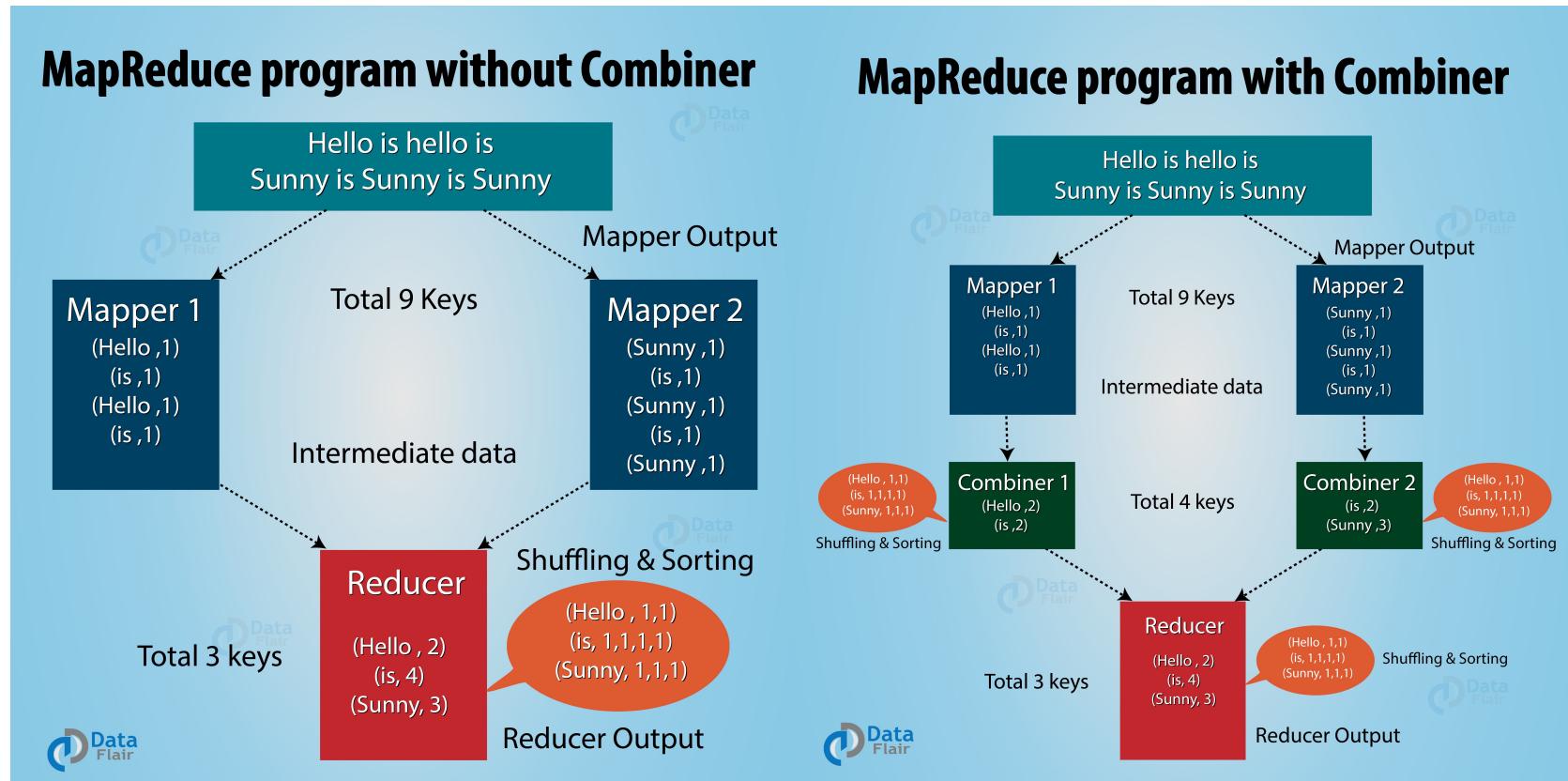
# Architettura generale

- Il Combiner è un reducer locale che esegue lo shuffling & sorting delle coppie generate dal mapper
- Diminuisce il traffico di rete e il carico del reducer



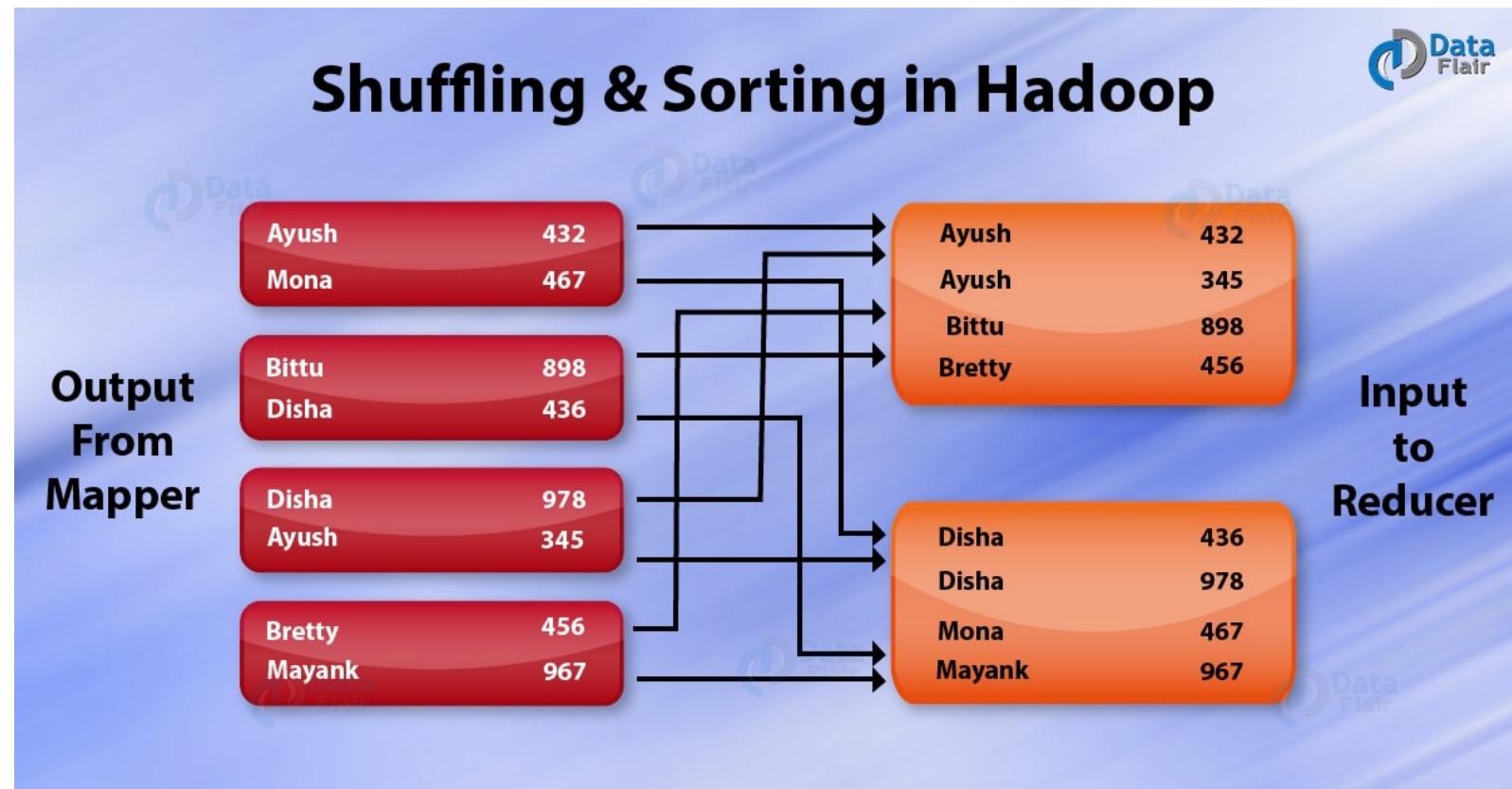
# Architettura generale

- Il combiner non evita la fase di shffling & sorting del reducer
- Può portare a un eccesso di uso del disco locale del mapper



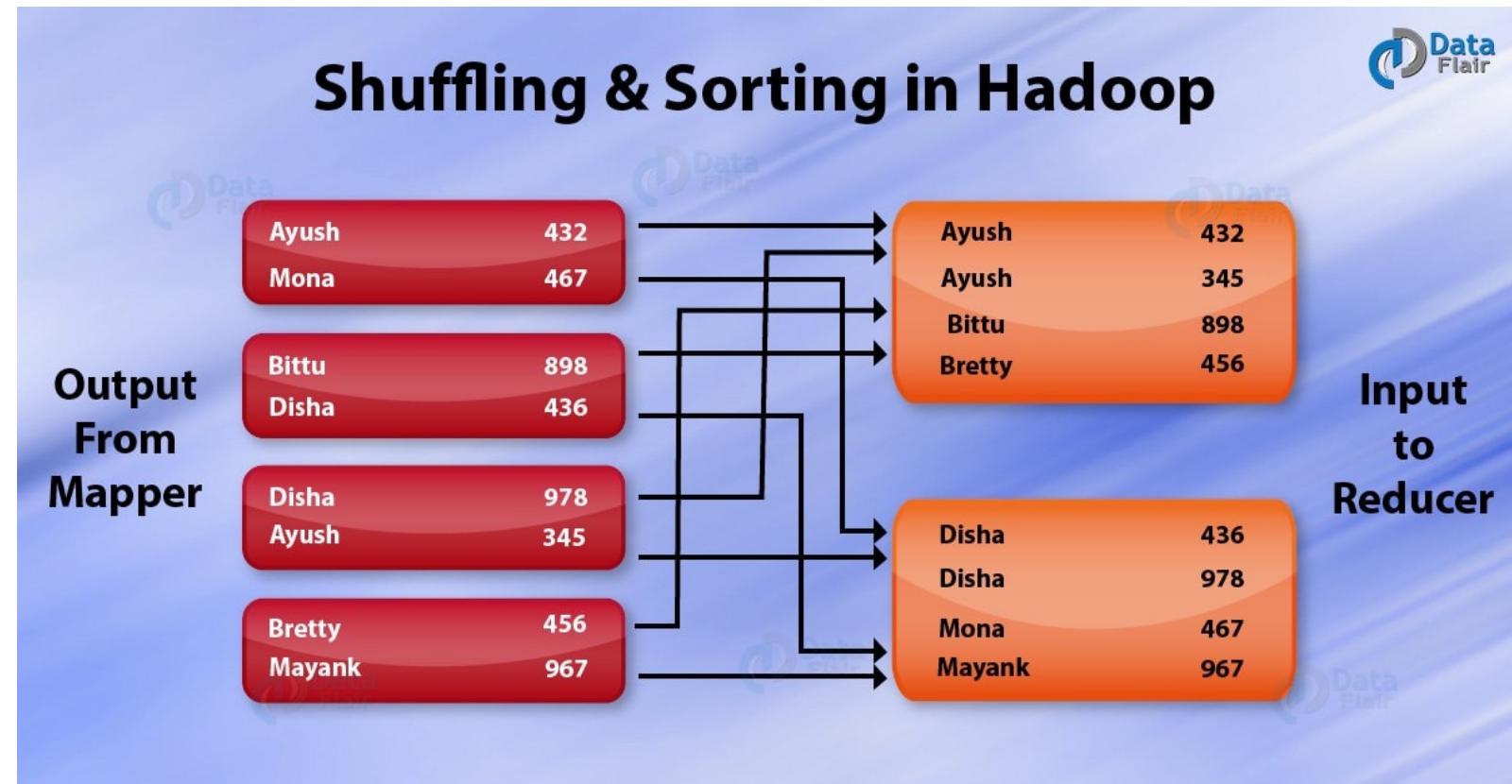
# Architettura generale

- Le operazioni di shuffling e sorting sono eseguite dal framework
- Lo shuffling ridistribuisce i valori generati dai mapper per passarli ai reducer



# Architettura generale

- Il sorting avviene *per chiave* insieme allo shuffling e prima dell'avvio dei reducer
- Ogni reducer non necessita di fare ordinamento
- L'ordinamento è usato dal framework per stabilire il numero dei reducer



# Architettura Generale

- Classi Hadoop MapReduce
- `InputFormat` gestisce la lettura dei dati da HDFS e la creazione degli split appartenenti alla classe `InputSplit` che vengono letti dalle istanze di `RecordReader` per generare le coppie chiave-valore di input

```
public abstract class InputFormat<K, V>
{
    public abstract List<InputSplit>
        getSplits(JobContext context) throws IOException, InterruptedException;

    public abstract RecordReader<K, V> createRecordReader(InputSplit split,
        TaskAttemptContext context) throws IOException,
        InterruptedException;
}
```

# Architettura Generale

- Classi Hadoop MapReduce
- Classi che implementano `InputFormat`
  - `TextInputFormat`, per linee di testo
    - Chiave: offset in byte della linea dall'inizio del file
    - Valore: contenuto della linea
  - `KeyValueTextInputFormat`, spezza le linee di testo in base al carattere '\t'
    - Chiave: offset in byte dall'inizio del file fino al carattere '\t'
    - Valore: contenuto della linea dopo il carattere '\t'

# Architettura Generale

- Classi Hadoop MapReduce
- Classi che implementano `InputFormat`
  - `SequenceFileInputFormat`, per file binari che serializzano qualunque tipo di dato
    - Chiave: definito dall'utente
    - Valore: definito dall'utente
  - `SequenceFileAsTextInputFormat`, come il precedente, ma richiama direttamente `toString()` per ottenere il formato testo di chiave e valore

# Architettura Generale

- Classi Hadoop MapReduce
- Classi che implementano `InputFormat`
  - `SequenceFileAsBinaryInputFormat`, restituisce chiave e valore come oggetti binari
  - `NLineInputFormat`, gestisce la possibilità di inviare al mapper un certo numero N di linee di testo gestite con coppie chiave-valore di tipo `TextInputFormat` ovvero `KeyValueTextInputFormat`

# Architettura Generale

- Classi Hadoop MapReduce
- Classi che implementano `InputFormat`
  - `DBInputFormat`, carica chiavi e valori da un database relazionale usando il driver JDBC

# Architettura Generale

- Classi Hadoop MapReduce
- `InputSplit` e `RecordReader` non vanno in genere utilizzate esplicitamente dall'utente perché i metodi `getSplits()` e `createRecordReader()` sono già implementati dalle classi di input
  - Il client calcola il numero di split del job attraverso `getSplits()` e li invia al JobTracker che gestisce la ripartizione degli split dove sono i dati colloquiando con i TaskTracker di nodo
  - I `RecordReader` sono creati per eseguire l'effettiva generazione delle coppie chiave-valore per i task di Map

# Architettura Generale

- Classi Hadoop MapReduce
- OutputFormat gestisce la scrittura dei risultati su HDFS
- E' una classe astratta da cui si derivano diversi tipi duali rispetto a quelli che implementano InputFormat

# Architettura Generale

- Classi Hadoop MapReduce
- `OutputFormat` gestisce la scrittura dei risultati su HDFS
  - `TextOutputFormat` è la classe di default in uscita dai processi di reduce: l'output è costituito da coppie chiave valore riportate come linee di testo separate dal carattere '\t'
  - `SequenceFileOutputFormat` definisce un formato file per la serializzazione degli output dei reducer, di qualunque tipo. E' usato nel caso di job Mapreduce in pipeline: il corrispondente `SequenceFileInputFormat` del mapper in cascata deserializza i dati negli stessi tipi di uscita del reducer precedente.

# Architettura Generale

- Classi Hadoop MapReduce
- OutputFormat gestisce la scrittura dei risultati su HDFS
  - SequenceFileAsBinaryOutputFormat analogo al precedente, ma crea dei file binari.
  - MapFileOutputFormat produce l'output come map file: si tratta di una cartella che contiene un file di dati, con chiavi ordinate e relativi valori, oltre ad un file di indici alle chiavi. E' necessario che il reducer emetta le chiavi in ordine

# Architettura Generale

- Classi Hadoop MapReduce
- OutputFormat gestisce la scrittura dei risultati su HDFS
  - MultipleOutputs consente la scrittura su più file di output oltre a quello di default del reducer
  - LazyOutputFormat una variante di OutputFormat che garantisce la scrittura del output per una data partizione solo se un record è stato effettivamente emesso per quella partizione

# Architettura Generale

- Classi Hadoop MapReduce
- OutputFormat gestisce la scrittura dei risultati su HDFS
  - DBOutputFormat consente di inviare l'output del reducer ad una tabella SQL

# Architettura Generale

- Classi Hadoop MapReduce
- La gestione complessiva del Job è demandata all'esecuzione di un `JobClient` per il quale è stata impostata una configurazione definita dalla classe `JobConf` che contiene le seguenti informazioni:
  - Nome del job
  - Nomi delle classi che implementano il mapper ed il reducer
  - Tipo della chiave e dei valori restituiti dal mapper
  - Tipo della chiave e dei valori restituiti dal reducer
  - Tipo del formato di input
  - Tipo del formato del output

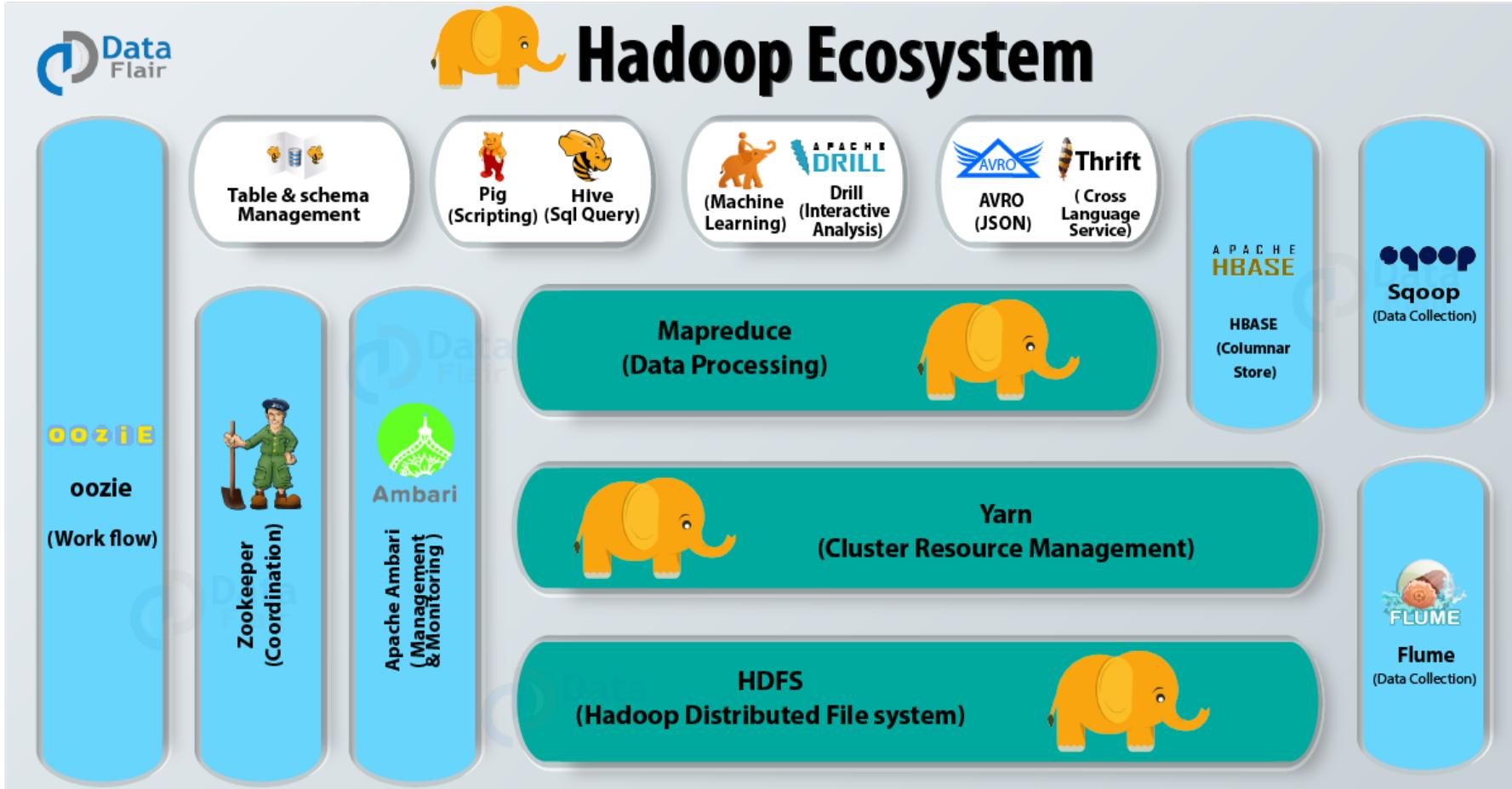
# Architettura Generale

- Classi Hadoop MapReduce
- Il mapper ed il reducer sono due classi che devono derivare da uno dei diversi modelli di applicazione MapReduce disponibili: il più semplice è `MapReduceBase`. Esse implementano rispettivamente le interfacce `Mapper` e `Reducer`.
- Entrambi emettono le coppie chiave valore tramite l'utilizzo del `OutputCollector`, il quale va parametrizzato con i tipi del chiave e del valore e deve coincidere con quanto riportato nella specifica della configurazione del job.

# Ecosistema Hadoop

- Con il termine ecosistema Hadoop ci si riferisce all'insieme di progetti software che si complementano con Apache Hadoop per realizzare applicazioni di analisi di dati eterogenei, sia batch che streaming
  - Machine learning
  - Workflow management
  - Monitoraggio
  - Datawarehouse
  - ...

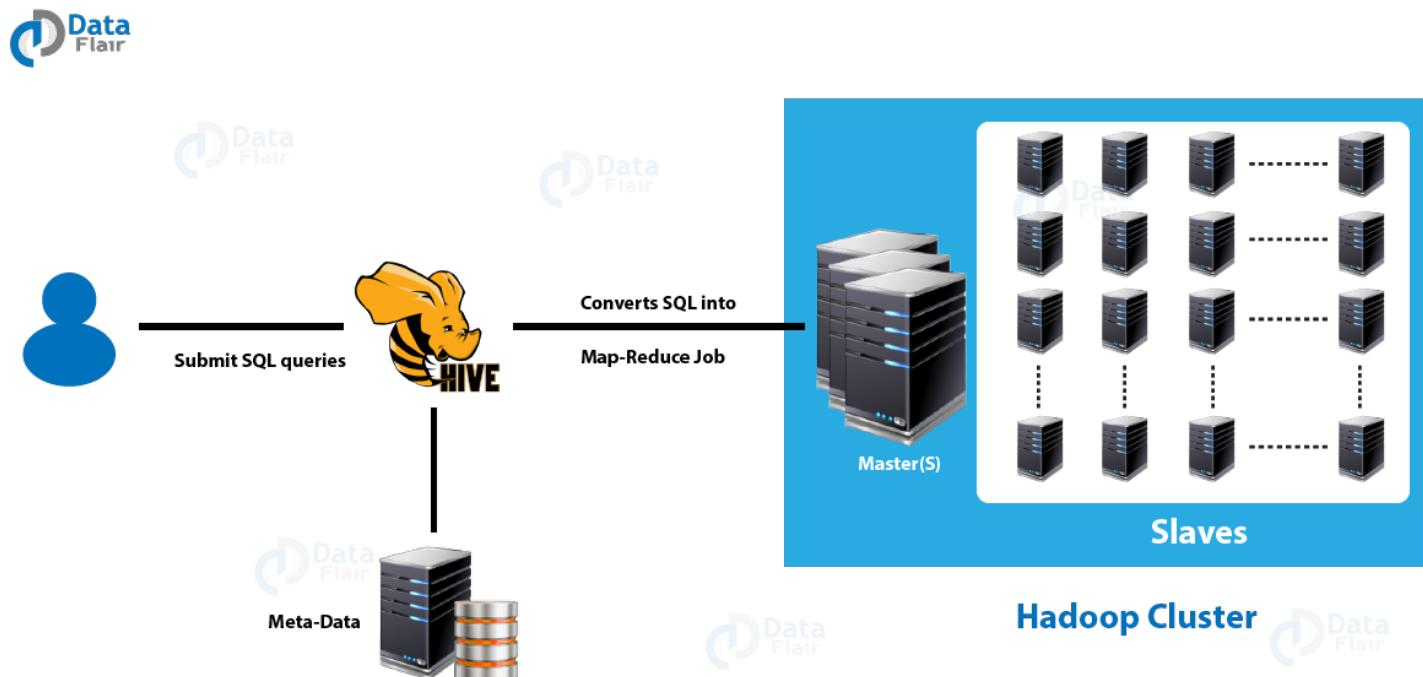
# Ecosistema Hadoop



# Ecosistema Hadoop

- Hive

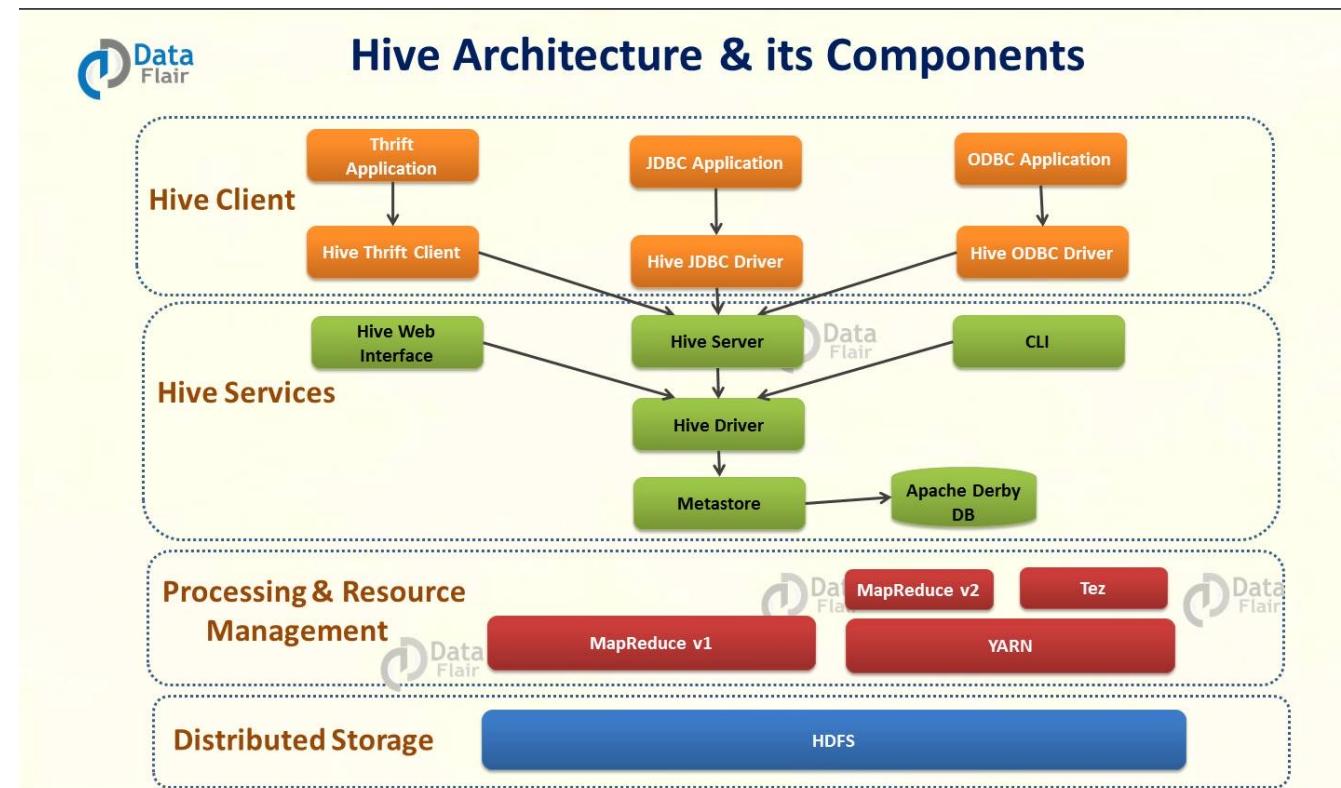
- È uno strumento di datawarehouse che ha un proprio linguaggio di query (Hive Query Language – HQL) per eseguire job MapReduce ed eseguire analisi dei dati



# Ecosistema Hadoop

- Hive

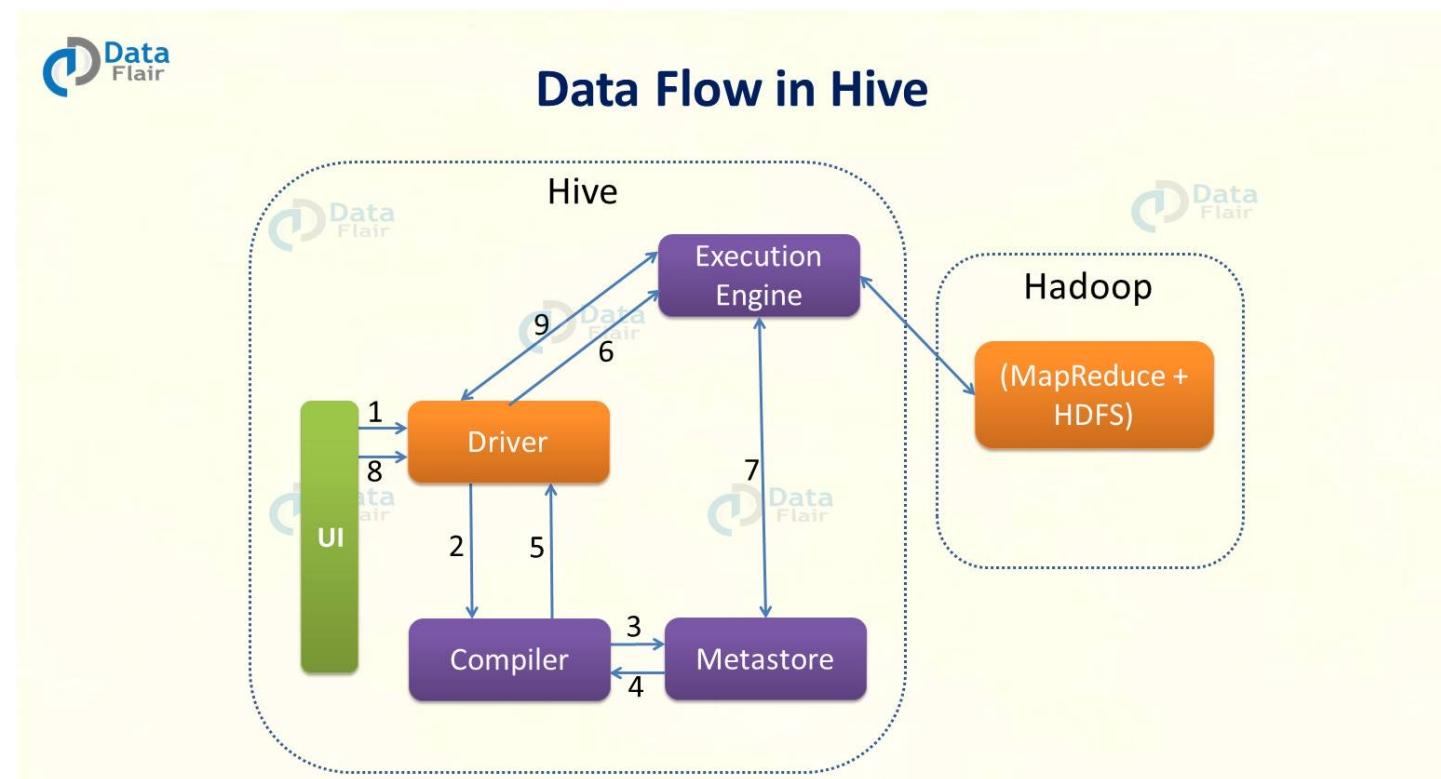
- Il job può essere sottomesso da un client ODBC/JDBC, per esempio in Python, ovvero da shell interattiva o via web
- Il Driver di Hive utilizza il Metastore che è il repository dei metadati per gli schemi e le tabelle Hive



# Ecosistema Hadoop

- Hive

- Il processo di compilazione genera un «piano» cioè un grafo diretto aciclico (DAG) di operazioni:
  - MapReduce
  - Accesso ai metadati
  - Accesso a HDFS
- La vera e propria esecuzione attraverso componenti Hadoop è delegata al execution engine



# Ecosistema Hadoop

- Hive
  - Comandi di shell del client `beeline` nella forma `!<nome comando>`
    - `!connect jdbc:hive2://localhost:1000/<nome db> <user> <pass>`  
– connessione al server Hive
    - `!dbinfo` – lista dei metadati del database
    - `!tables` – lista delle tabelle del database corrente
    - `!columns` – informazioni sulle colonne della tabella
    - `!sql` – esecuzione di un comando SQL
    - `!list` – elenco delle connessioni attive
    - `!quit` – uscita
    - ...

# Ecosistema Hadoop

- Hive
  - Comandi di shell `hive`
    - `set <key>` / `set <key> = <val>` -- imposta dei valori di configurazione
    - `reset` / `reset <key>` -- reset della configurazione
    - `dfs <comando hdfs>` -- esecuzione di comandi HDFS
    - `source <nome file>` -- esecuzione di script
  - Gestione delle risorse nella cache
  - `ADD { FILE[S] | JAR[S] | ARCHIVE[S] } <filepath1> [<filepath2>]*`
  - `LIST { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]`
  - `DELETE { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]`

# Ecosistema Hadoop

- Hive
  - Comandi CRUD – creazione/cancellazione/uso/descrizione di un database
    - CREATE SCHEMA <nome db>
    - CREATE DATABASE [ IF NOT EXISTS ] <nome db>
    - DROP DATABASE [ IF EXISTS ] <nome db>
    - USE <nome db>
    - DESCRIBE DATABASE / SCHEMA <nome db>

# Ecosistema Hadoop

- Hive
  - Tipi di dato semplice
    - TINYINT/SMALL/INT/BIGINT
    - FLOAT/DOUBLE/DECIMAL
    - TIMESTAMP/DATE/INTERVAL
    - STRING/VARCHAR/CHAR
    - BOOLEAN
    - BINARY
  - interi con segno da 8 a 64 bit
  - virgola mobile a 32/64 bit e a precisione infinita
  - tempo con precisione al nanosecondo
  - stringhe a lunghezza non fissata ovvero a lunghezza fissa
  - sequenza di byte

# Ecosistema Hadoop

- Hive
  - Tipi di dato complessi
    - ARRAY<tipo del dato>
    - MAP<tipo della chiave, tipo del dato>
    - STRUCT<nome colonna: tipo [COMMENT 'commento'], ... >
    - UNIONTYPE<tipo, tipo, tipo ... >

# Ecosistema Hadoop

- Hive
  - Comandi CRUD – creazione/alterazione/cancellazione/descrizione di una tabella

```
CREATE TABLE [ IF NOT EXISTS ] <nome tabella>(<nome colonna>
<tipo colonna>[ , <nome colonna> <tipo colonna>] , ...);
```

# Ecosistema Hadoop

- Hive
  - Comandi CRUD – creazione/alterazione/cancellazione/descrizione di una tabella
    - ALTER TABLE <nome tabella> RENAME TO <nuovo nome tabella>
    - ALTER TABLE <nome tabella> [PARTITION partition\_spec] ADD|REPLACE COLUMNS (<nome colonna> <tipo colonna> [COMMENT 'commento'], ...) [CASCADE|RESTRICT]

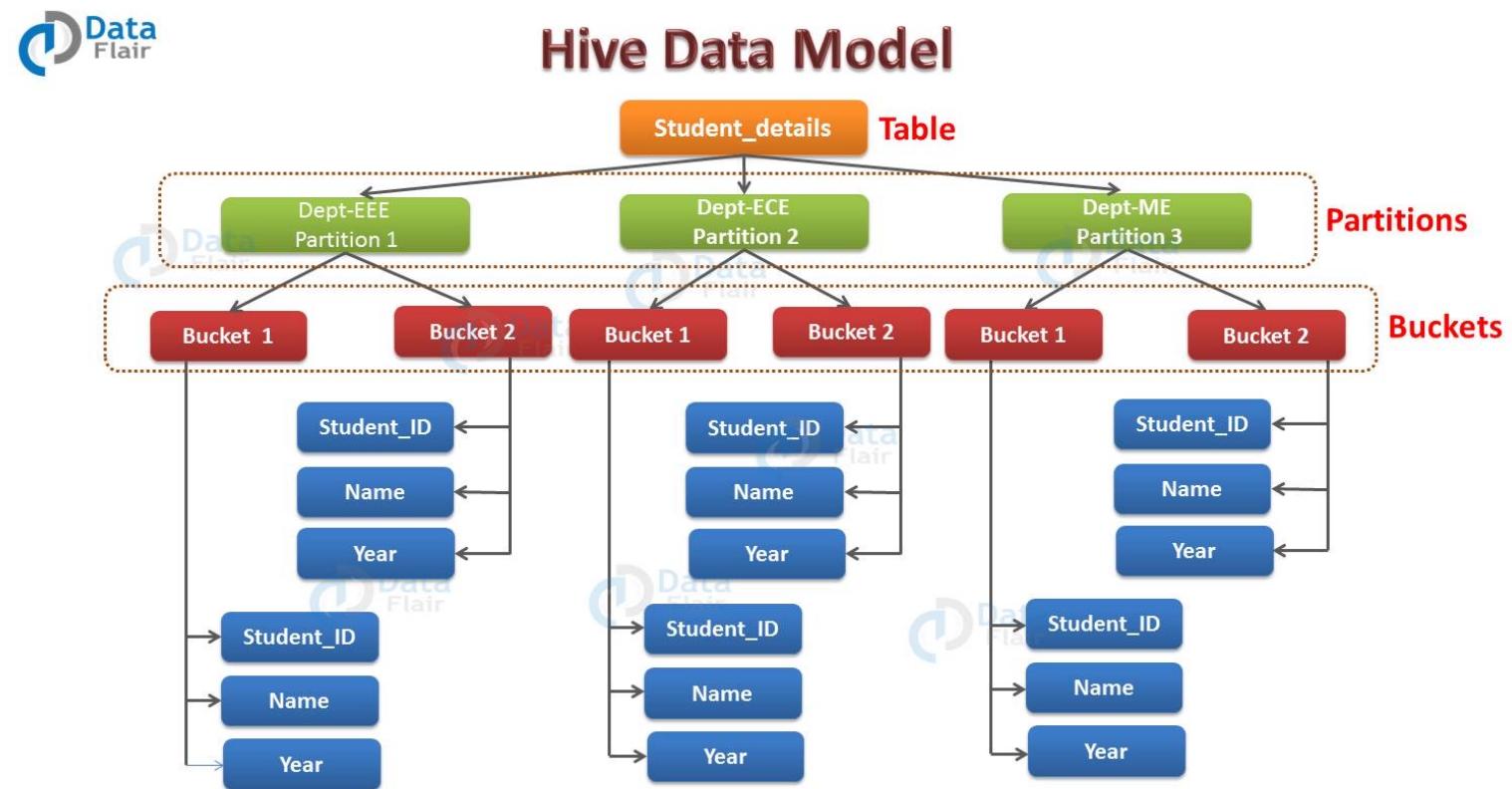
# Ecosistema Hadoop

- Hive
  - Comandi CRUD – creazione/alterazione/cancellazione/descrizione di una tabella
    - ALTER TABLE <nome tabella> [PARTITION partition\_spec] CHANGE [COLUMN] <vecchio nome col> <nuovo nome col> <tipo colonna> [COMMENT 'commento'] [FIRST|AFTER <nome colonna>] [CASCADE|RESTRICT]
    - DROP TABLE [ IF EXISTS ] <nome tabella>
    - DESCRIBE <nome tabella>

# Ecosistema Hadoop

- Hive

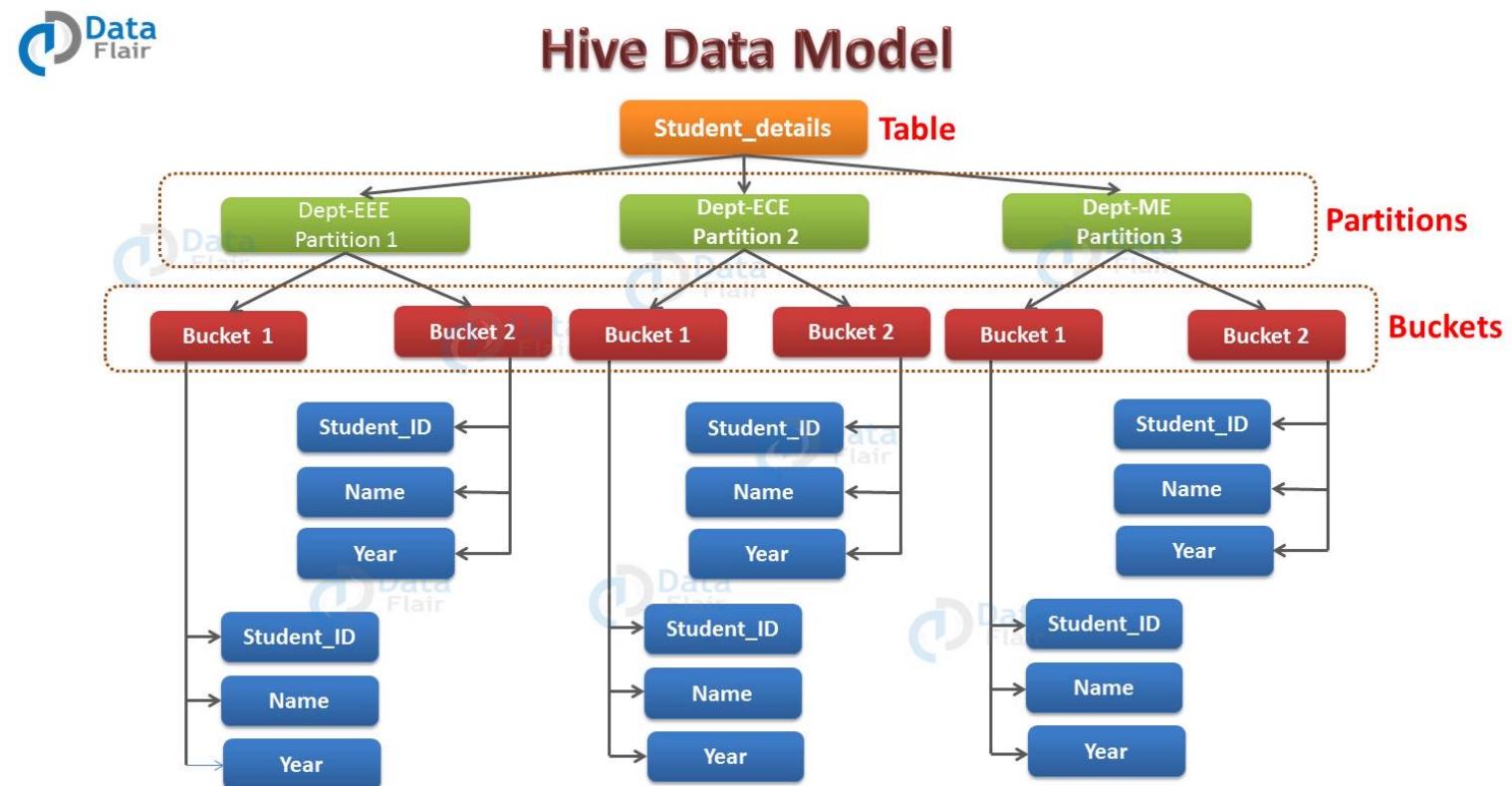
- I dati creati in tabella possono essere suddivisi in partizioni e/o raggruppamenti («buckets»)
- Una partizione separa i dati logicamente secondo un fine applicativo in base ad una o più chiavi il cui tipo coinciderà con quello delle colonne di interesse



# Ecosistema Hadoop

- Hive

- I bucket si creano per rendere efficace l'indicizzazione secondo certe chiavi
- Si utilizzano le direttive
  - CLUSTERED BY
  - SORTED BY



# Ecosistema Hadoop

- Hive
  - Comandi CRUD – creazione di una tabella, sintassi completa

```
CREATE TABLE [ IF NOT EXISTS ] <nome tabella>(<nome colonna> <tipo  
colonna>[ , <nome colonna> <tipo colonna>], ...)  
[ PARTITIONED BY (col_name data_type [ COMMENT col_comment ], ... )]  
[ CLUSTERED BY (col_name, col_name, ... )  
[ SORTED BY (col_name [ ASC|DESC ], ... )] INTO num_buckets BUCKETS ]  
[ ROW FORMAT row_format ]  
[ STORED AS file_format ]  
[ LOCATION hdfs_path ]  
[ TBLPROPERTIES (property_name=property_value, ... ) ]  
[ AS select_statement ];
```

# Ecosistema Hadoop

- Hive – esempio completo di creazione tabella

```
CREATE TABLE bucketed_user(
    firstname VARCHAR(64),
    lastname VARCHAR(64),
    address STRING,
    city VARCHAR(64),
    state VARCHAR(64),
    post STRING,
    phone1 VARCHAR(64),
    phone2 STRING,
    email STRING,
    web STRING
)
COMMENT 'A bucketed sorted user table'
PARTITIONED BY (country VARCHAR(64))
CLUSTERED BY (state) SORTED BY (city)
INTO 32 BUCKETS
STORED AS SEQUENCEFILE;
```

# Ecosistema Hadoop

- Hive
  - Query di selezione

```
SELECT [ALL | DISTINCT] espressione, espressione, ...
      FROM nome_tabella
      [WHERE condizione]
      [GROUP BY lista di colonne]
      [HAVING condizione]
      [CLUSTER BY lista di colonne | [DISTRIBUTE BY
          lista di colonne] [SORT BY lista di colonne]]
      [LIMIT num];
```

# Ecosistema Hadoop

- Hive
  - Caricamento e inserimento dati
    - LOAD DATA [LOCAL] INPATH 'percorso\_file' [OVERWRITE] INTO TABLE nome\_tabella [PARTITION (partcol1=val1, partcol2=val2 ...)] [INPUTFORMAT 'inputformat' SERDE 'serde']

La ricerca avviene nel  
filesystem locale

Specifica la serializzazione  
dei dati (JSON, CSV, testo ...)

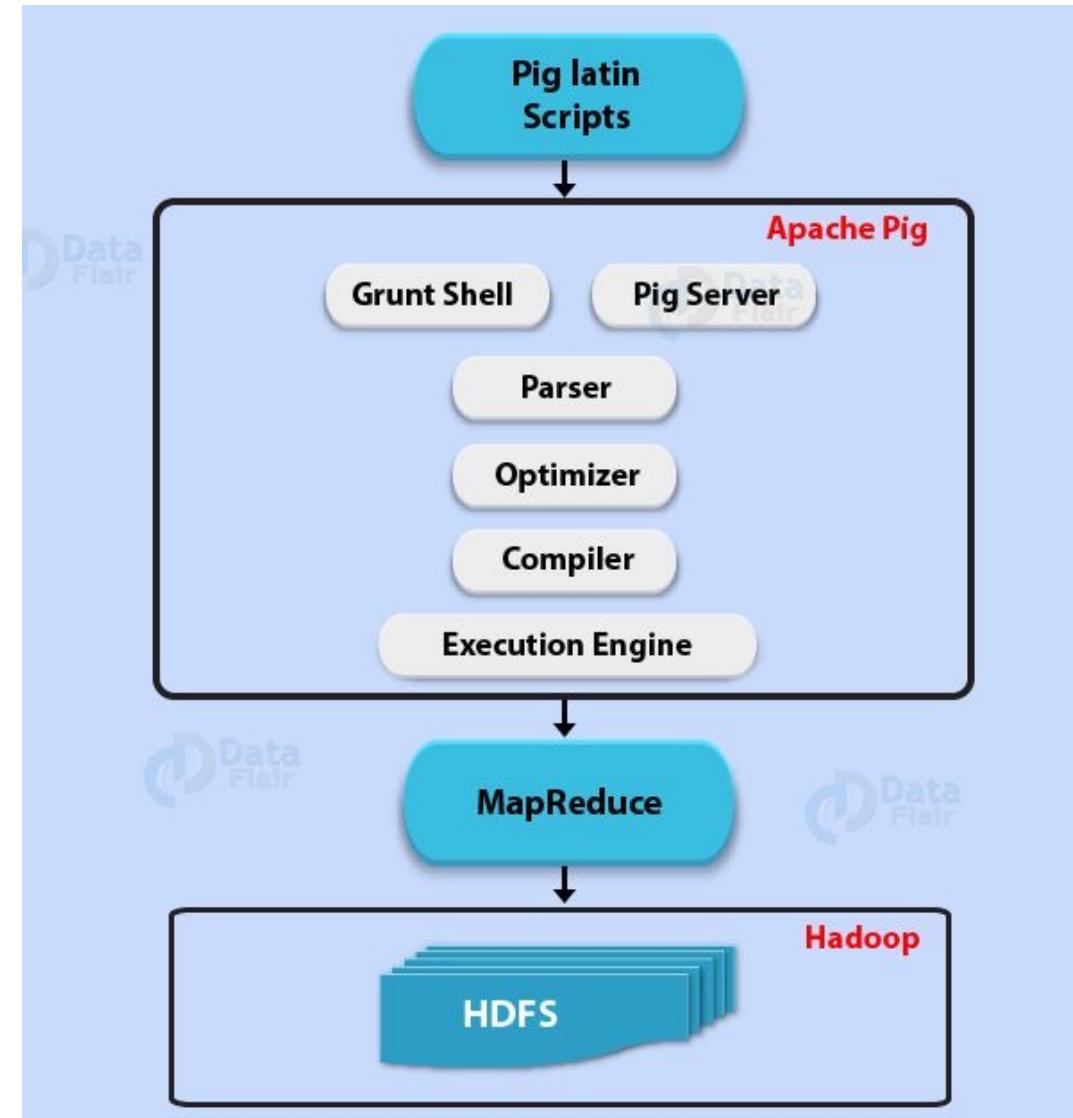
# Ecosistema Hadoop

- Hive
  - Caricamento e inserimento dati
    - `INSERT OVERWRITE TABLE nome_tabella [PARTITION (partcol1=val1, partcol2=val2 ...) [IF NOT EXISTS]] clausola_select FROM clausola_from;`
    - `INSERT INTO TABLE nome_tabella [PARTITION (partcol1=val1, partcol2=val2 ...)] clausola_select FROM clausola_from;`

# Ecosistema Hadoop

- Pig

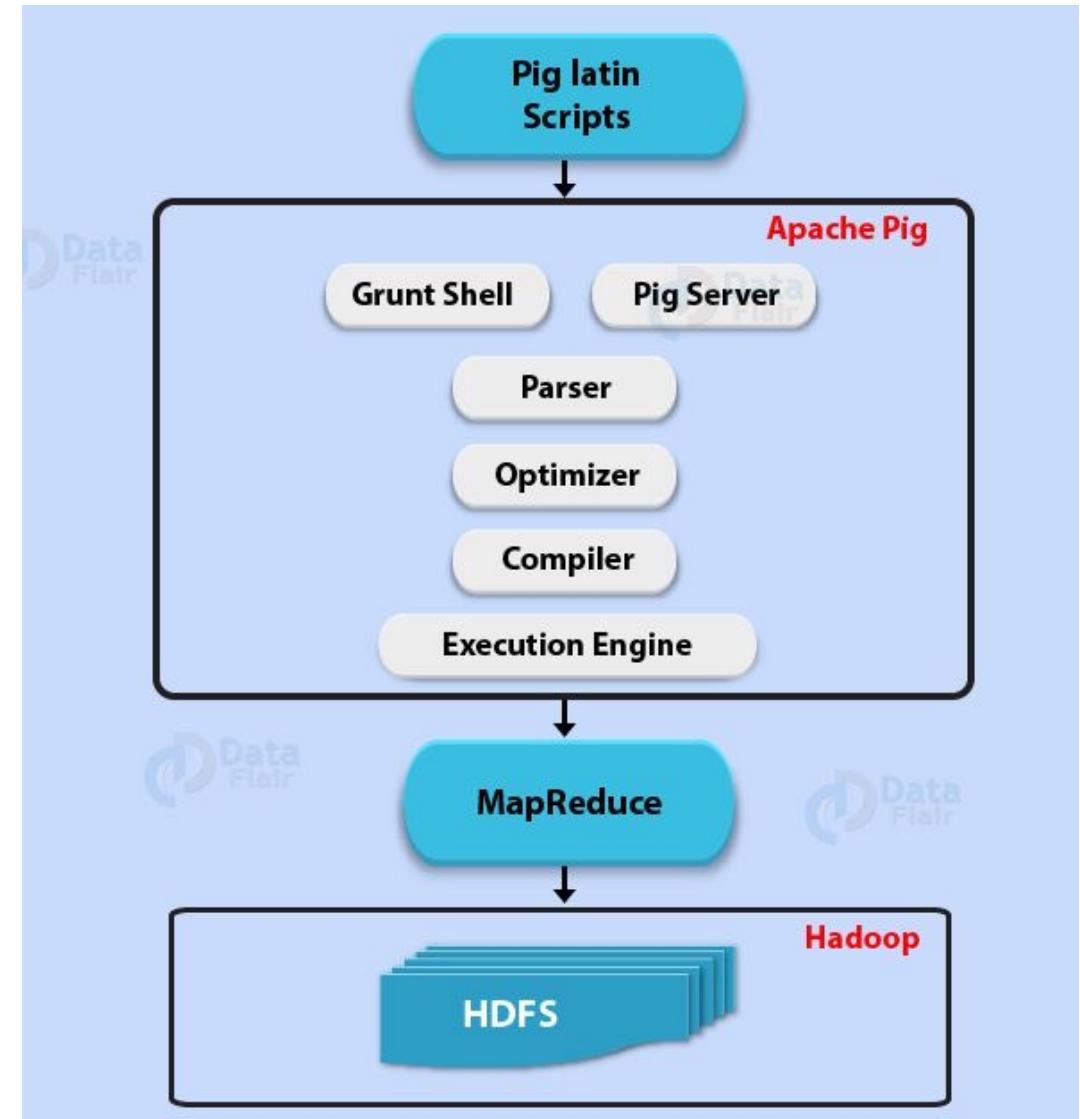
- Analogamente a Hive genera un piano in forma di DAG che innesca una esecuzione MapReduce su HDFS
- Si basa sugli script «Pig Latin» che è un linguaggio procedurale, a differenza di HQL che è un linguaggio di query



# Ecosistema Hadoop

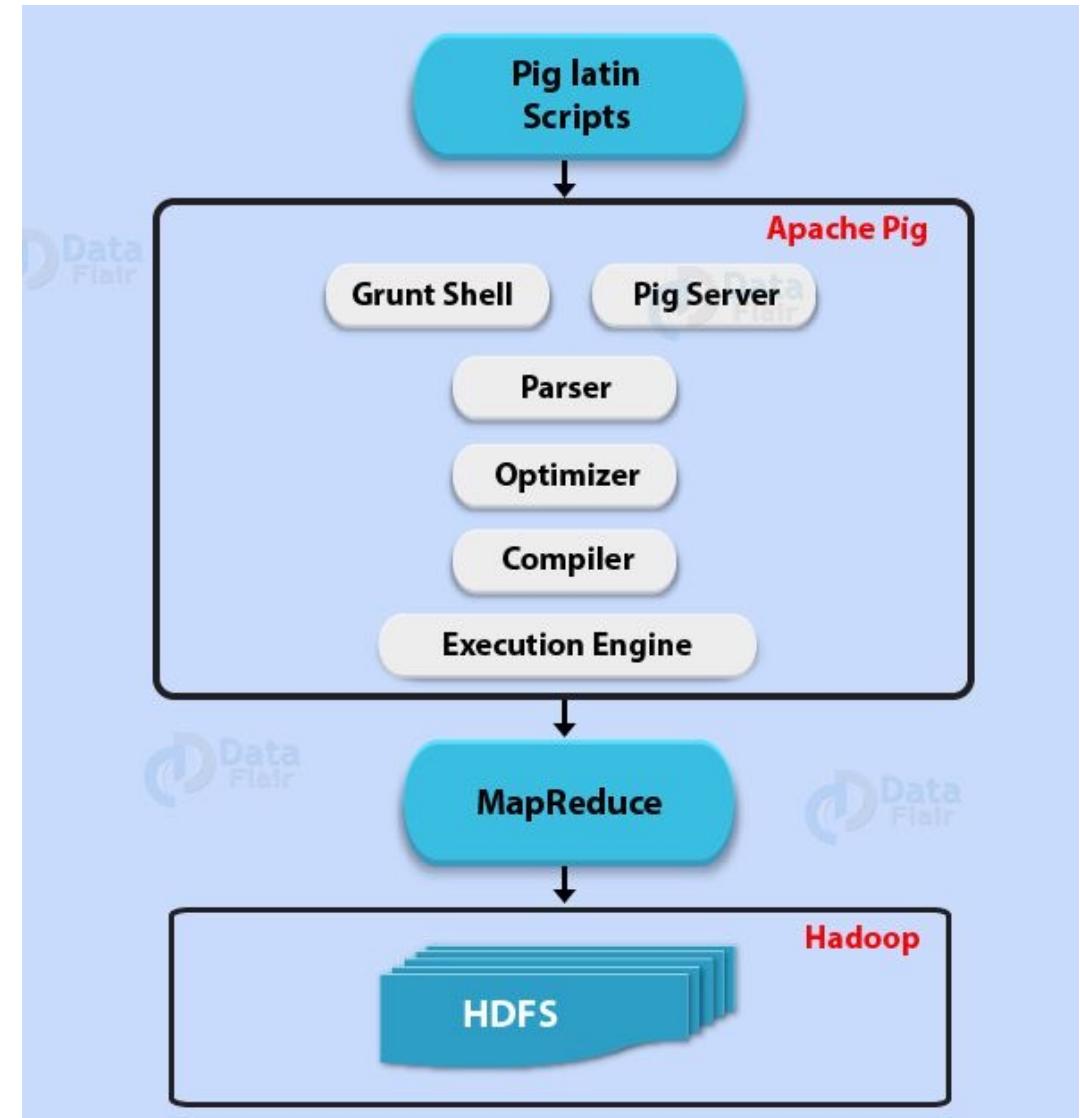
- Pig

- Pig Latin possiede un set molto ampio di funzioni e operatori, ma consente anche la definizione di User Defined Functions (UDF)
- Il modello dei dati si basa su diversi tipi di aggregazione



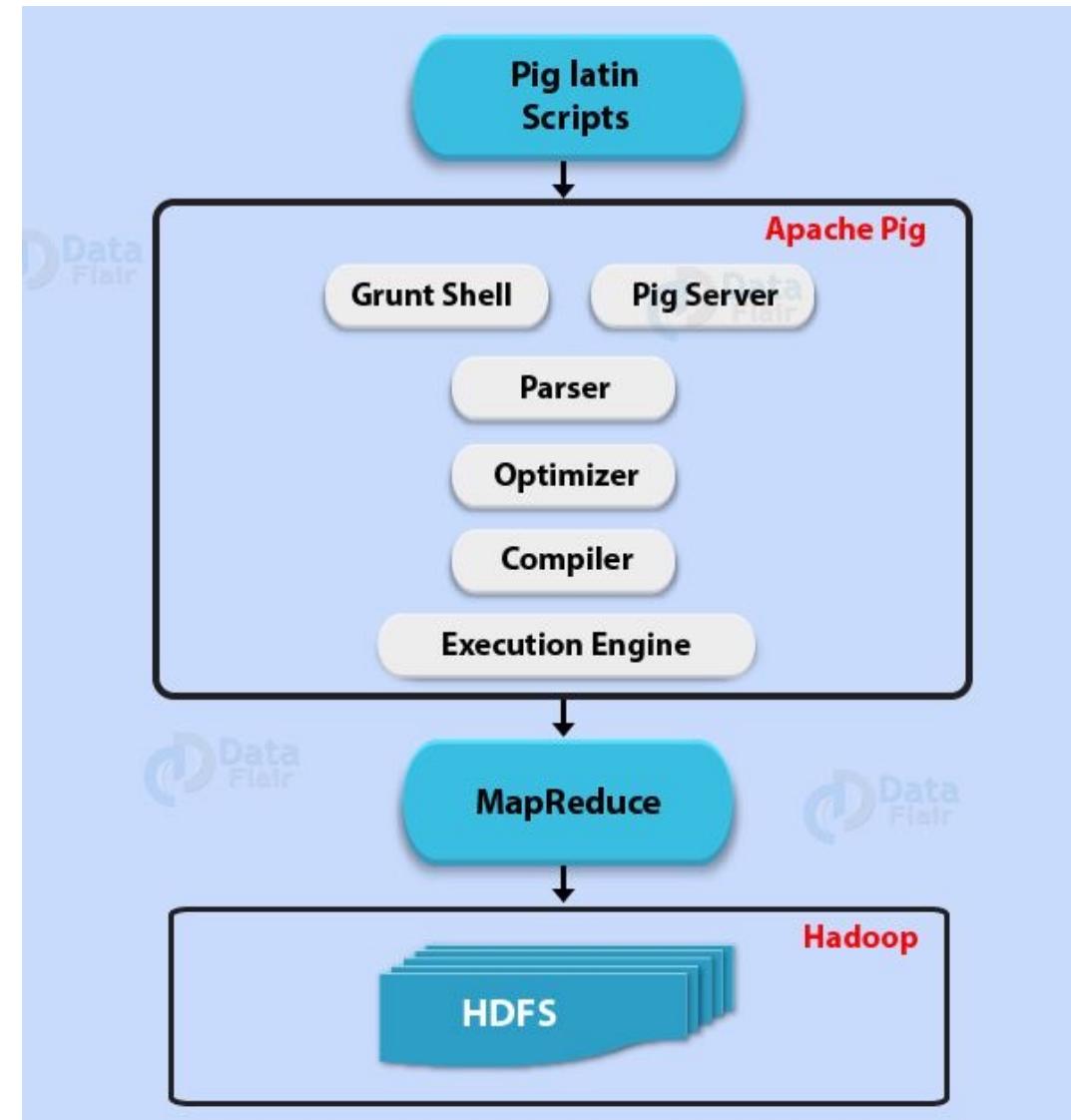
# Ecosistema Hadoop

- Pig
  - Tipi di dato semplice o atomico
    - int
    - long
    - float
    - double
    - chararray
    - bytearray
    - boolean
    - datetime
    - biginteger
    - bigdecimal



# Ecosistema Hadoop

- Pig
  - Tipi di dato complesso
    - Tupla: come la riga di un database  
(891, 1, 'Mrs. Anne Smith',  
'female', 32.0F)
    - Map: coppia chiave valore  
[ name#John, age#25 ]

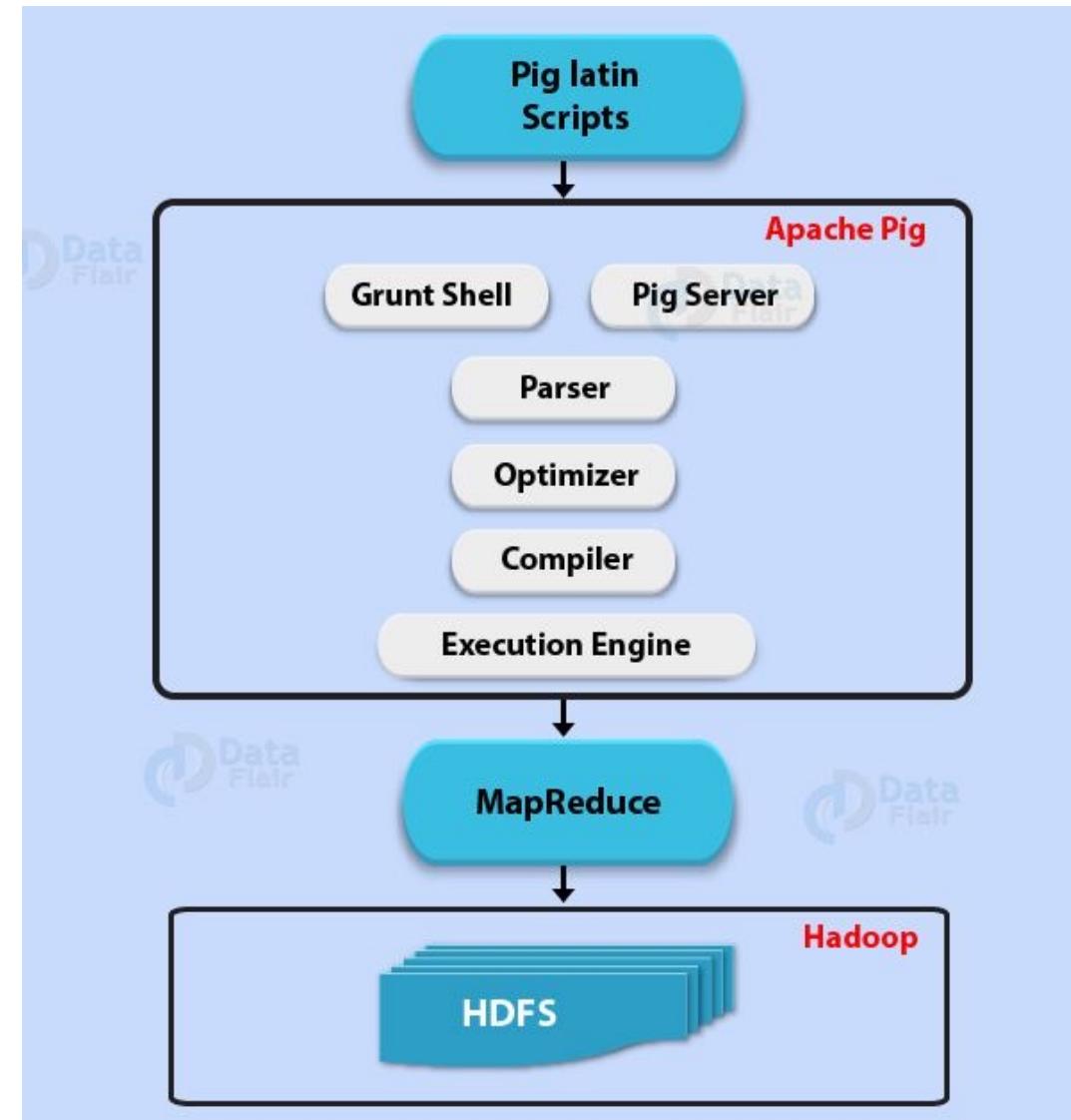


# Ecosistema Hadoop

- Pig

- Tipi di dato complesso

- Bag: un insieme non ordinato di tuple
- ```
{(891, 1, 'Mrs. Anne Smith',  
'female', 32.0F),  
 (567, 0, 'Mr. George  
Duncan', 'male', 58.0F),  
 ... }
```



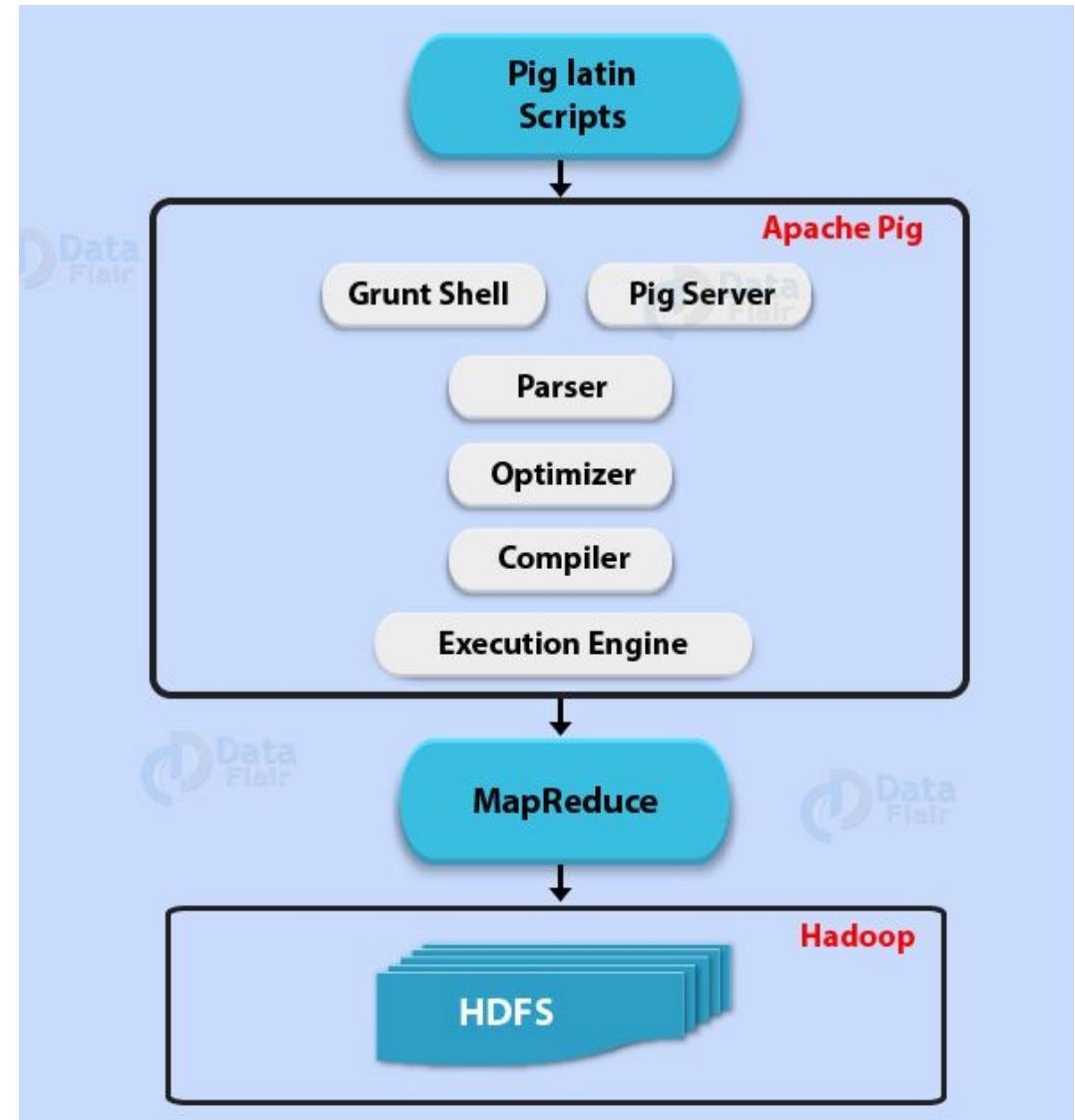
# Ecosistema Hadoop

- Pig

- Tipi di dato complesso

- Si parla in generale di «relazioni», riferendosi a bag che possono anche contenerne altre al loro interno

```
{(1, {('John',1,false),('Mary',1,true)}),  
 (2, {('Jack',2,true)}),  
 (3,{('Will',3,true),('Dave',3,true),  
 ('Danny',3,false)})}
```



# Ecosistema Hadoop

- Pig
  - Operatori base
    - Aritmetici, binari, booleani, di casting, di confronto
    - \$<num posizione> per accedere ai campi di una tupla
    - Dereference: `tupla.campo`, `bag.campo`, `mappa#chiave`,
    - :: è il dereference per disambiguazione

# Ecosistema Hadoop

- Pig
  - Funzioni di load/store
    - <alias> = load 'path (locale o hdfs)'  
using <funzione di storage>  
[as (<nome campo>:<tipo> [, <nome campo>:<tipo>, ... ] )]
    - store <alias> into 'path (locale o hdfs)'  
using <funzione di storage>
    - La funzione di storage definisce il formato dello storage: PigStorage( [<delim>] )  
gestisce il testo csv

# Ecosistema Hadoop

- Pig
  - Descrizione dei dati
    - DUMP <alias> – esegue il piano MapReduce per generare l'aggregato e mostra le tuple
    - EXPLAIN <alias> -- mostra la struttura del piano e la sua decomposizione in azioni semplici
    - DESCRIBE <alias> -- mostra lo schema dell'aggregato
    - ILLUSTRATE <alias> -- come describe, ma inserisce esplicitamente alcuni campioni

# Ecosistema Hadoop

- Pig
  - Operatori relazionali
    - alias = ORDER alias BY { \* [ASC|DESC] | field\_alias [ASC|DESC] [, field\_alias [ASC|DESC] ...] } [PARALLEL n];
      - Ordinamento classico
    - alias = RANK alias [ BY { \* [ASC|DESC] | field\_alias [ASC|DESC] [, field\_alias [ASC|DESC] ...] } [DENSE] ];
      - Inserisce un campo con il numero d'ordine mentre ordina
    - alias = FILTER alias BY expression; -- filtra per righe

# Ecosistema Hadoop

- Pig
  - Operatori relazionali
    - `alias = LIMIT alias n;`
      - Taglia alle prime n righe
    - `alias = GROUP alias { ALL | BY expression} [, alias ALL | BY expression ...] [USING 'collected' | 'merge'] [PARTITION BY partitioner] [PARALLEL n];`
      - Retituisce un aggregato in cui ogni tupla contiene la chiave di raggruppamento e una inner bag con tutte le tuple raggruppate

# Ecosistema Hadoop

- Pig
  - Operatori relazionali
    - alias = FOREACH alias GENERATE expression [AS schema] [expression [AS schema], ...];
      - Opera sulle colonne attraverso gli operatori e/o le funzioni predefinite di Pig ovvero quelle definite dall'utente e restituisce un nuovo aggregato secondo lo schema voluto
    - alias = JOIN alias BY expr, alias BY expr [USING join\_strategy] [PARTITION BY partitioner] [PARALLEL n];
    - alias = JOIN left-alias BY left-alias-column [LEFT|RIGHT|FULL] [OUTER], right-alias BY right-alias-column [USING join\_strategy] [PARTITION BY partitioner] [PARALLEL n];