



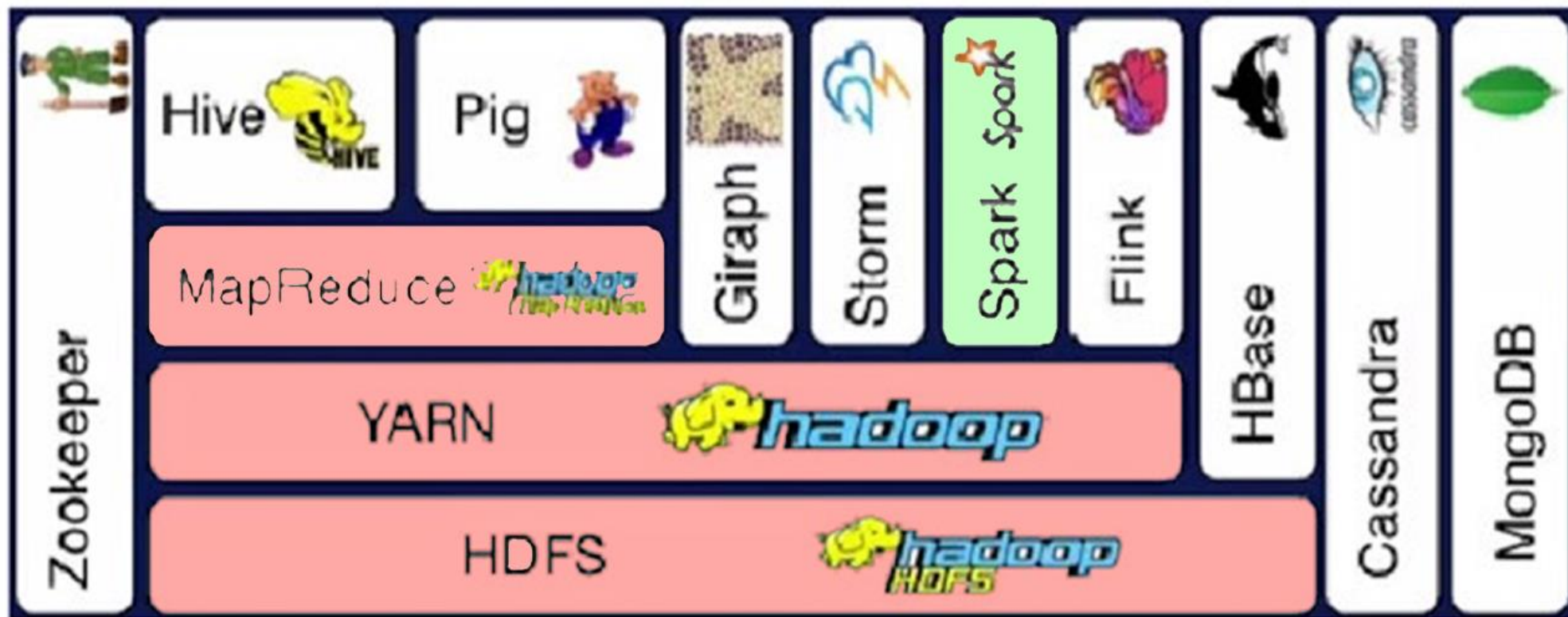
Odisee  
DE CO-HOGESCHOOL

# Big Data - Spark



Jens Baetens

# Hadoop Ecosystem





# Wat is Spark?



# Probleem met Hadoop

## ▣ Voordelen Hadoop

- ▬ Eenvoudig programmeermodel
- ▬ Schaalbaar
- ▬ Goedkoop
- ▬ Flexibel
- ▬ Fout-tolerantie

## ▣ Maar: Niet snel!





# Spark

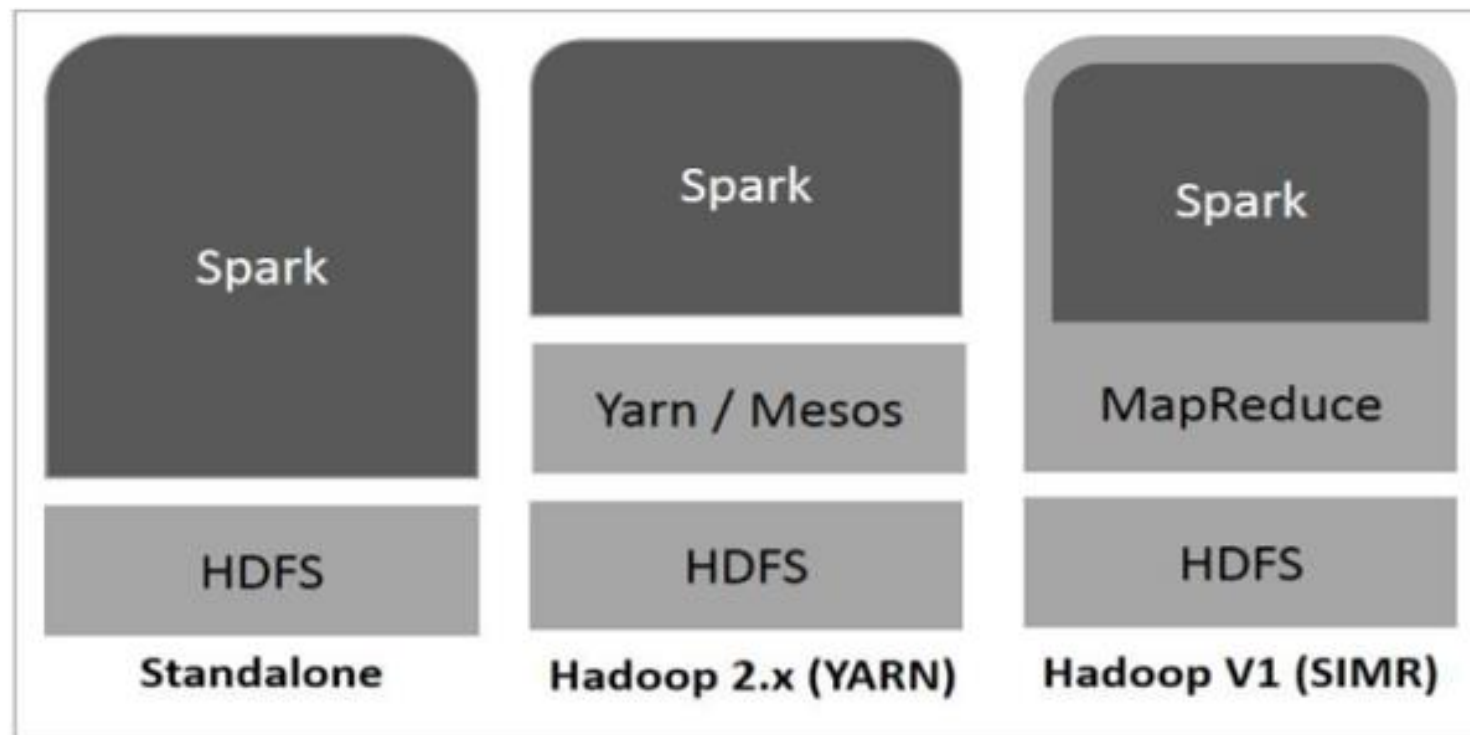
- ▣ Geïntroduceerd om de rekenlaag van Hadoop te versnellen
- ▣ Heeft zijn eigen cluster-beheer
  - ▬ Geen andere versie van Hadoop
  - ▬ Niet afhankelijk van Hadoop
- ▣ 2 delen
  - ▬ Storage                      -> Hiervoor kan Hadoop gebruikt worden
  - ▬ Processing



## Features

- ▣ Speed: in memory-processing
- ▣ Ondersteuning voor verscheidene talen (bvb: Python, Java, Scala)
- ▣ Geavanceerde Analytics
  - MapReduce
  - SQL
  - Streaming
  - Machine Learning
  - Graph algoritmes

## Modes van Spark





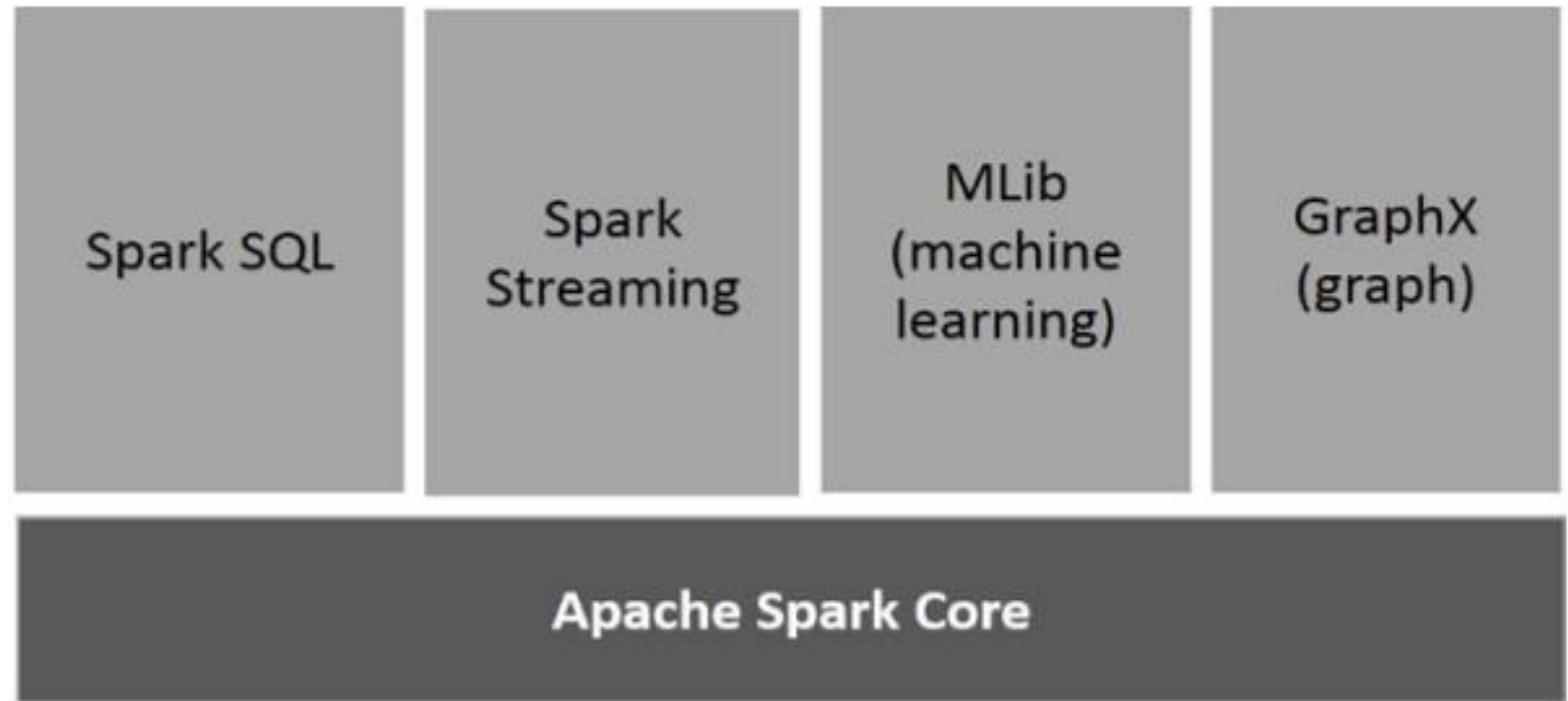
## Componenten

### ▣ Spark Core

- ▬ Execution engine
- ▬ In-memory computing
- ▬ External Datasets (HDFS)

### ▣ Spark SQL

- ▬ (Semi) structured Data
- ▬ Distributed Database
- ▬ RDD – Resilient Distributed Datasets



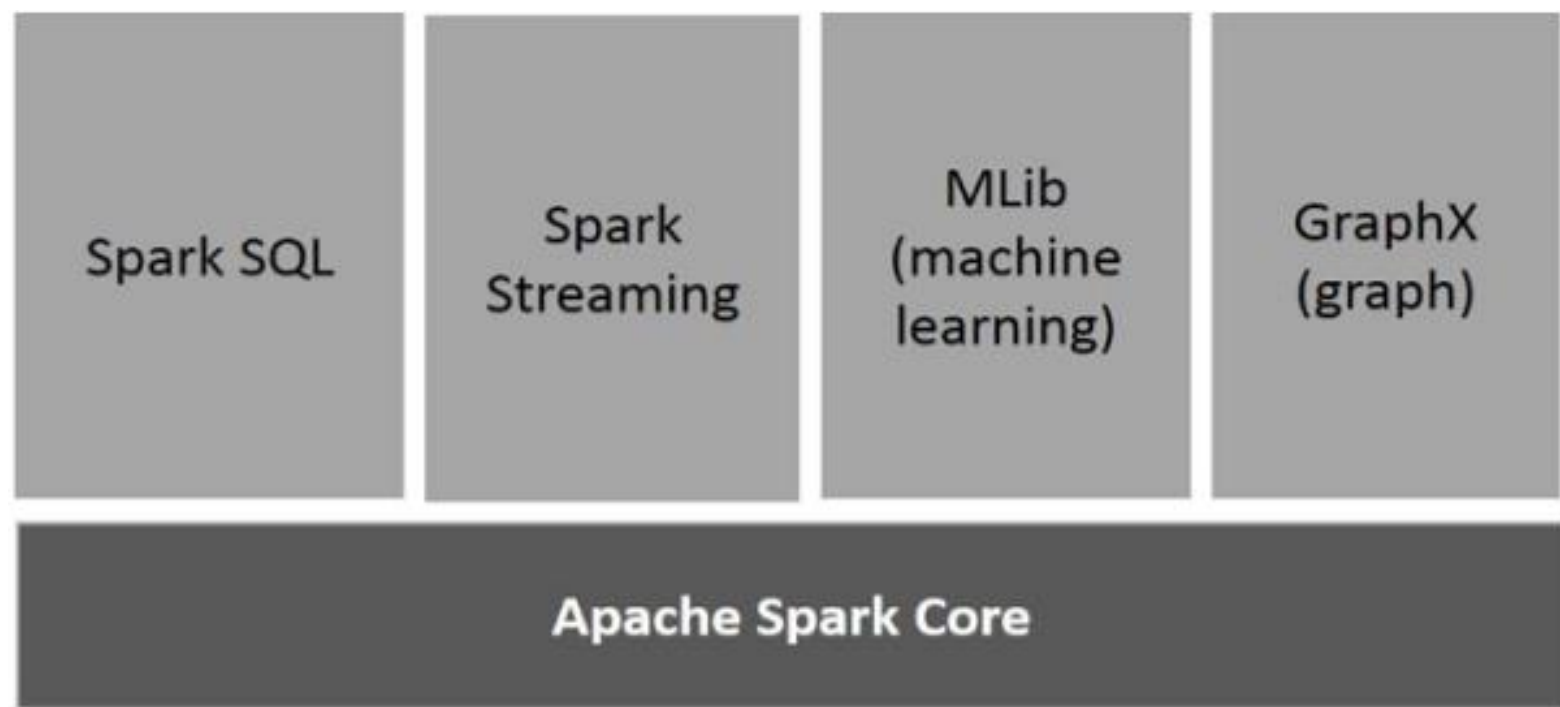
## Componenten

### ▣ Spark Streaming

- ▬ Streaming analytics
- ▬ Verwerk data in mini-batches
- ▬ Voer RDD transformaties uit

### ▣ MLlib

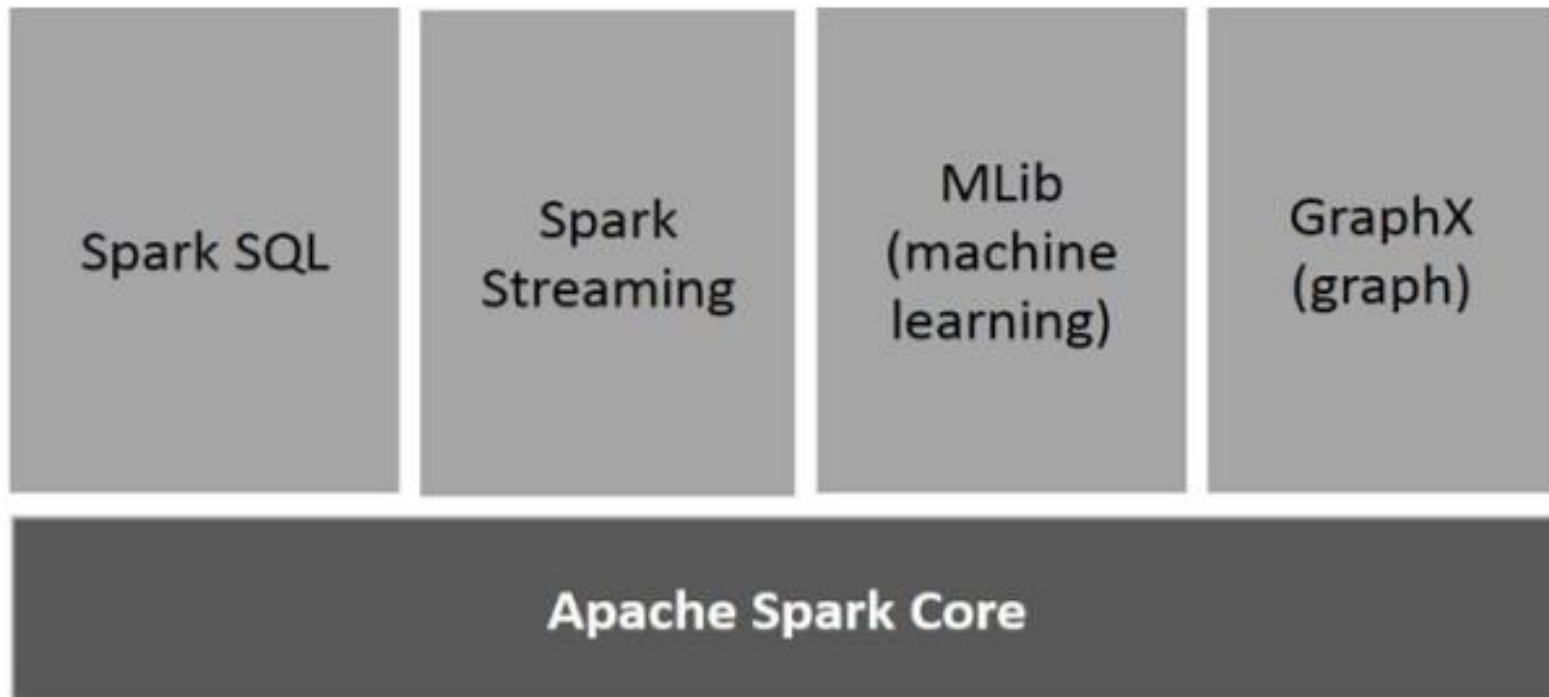
- ▬ Distributed machine learning framework
- ▬ 9 x sneller dan Mahout (ML via MapReduce)



## Componenten

### ▣ GraphX

- Graph Processing Framework





Storage

HDFS

MapReduce



Speed

Fast

Resource  
Management

YARN

Spark

Leverage  
Existing



10 - 100 X  
Faster

Standalone

[www.hadoopinrealworld.com](http://www.hadoopinrealworld.com)

# Onderdelen van Spark

1

Spark SQL

2

Spark Streaming

3

MLlib

4

GraphX

INTRODUCTION  
TO

APACHE  
Spark

The Apache Spark logo features a stylized orange star with a black outline, positioned to the right of the word 'Spark'.

<https://youtu.be/ymtq8yjmD9I>



# Spark SQL



## Resilient Distributed Datasets

- ▣ Distributed collectie van objecten
- ▣ Fout tolerante read-only gepartitioneerde collectie van gegevens
- ▣ Aangemaakt door
  - Parallelliseren van een bestaande collective
  - Inladen van een dataset in een externe opslagsysteem (Shared file system, HDFS, HBase, SQL, ...)



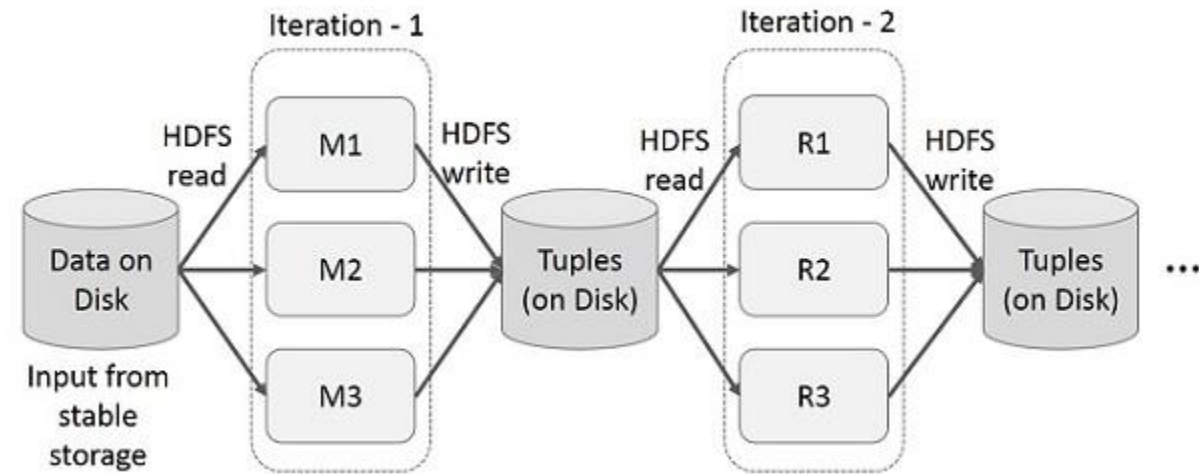
# Probleem met map-reduce

## ▣ Delen van data is traag

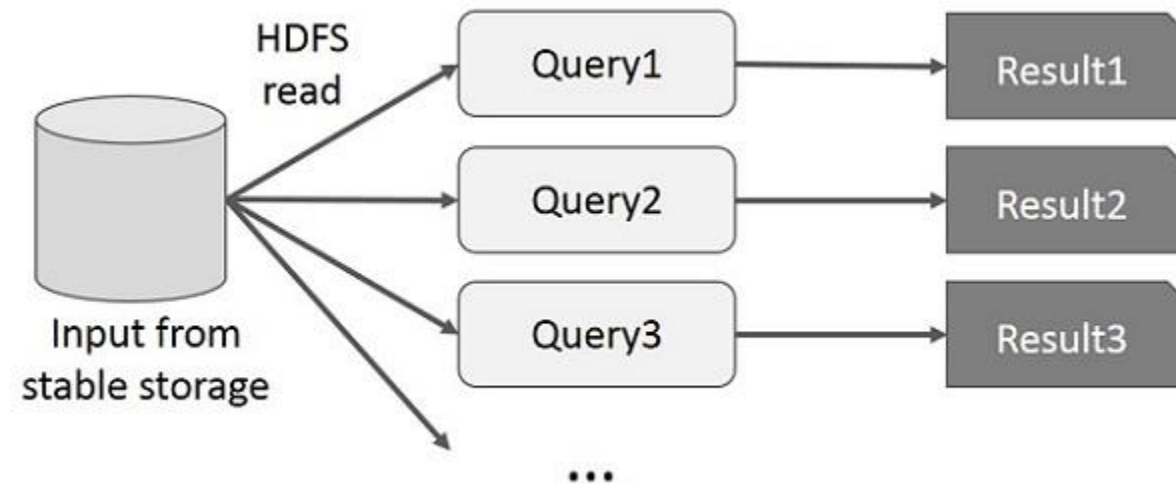
- Replication
- Serialization
- Disk IO

## ▣ 90% van de tijd in HDFS read-write

Iterative



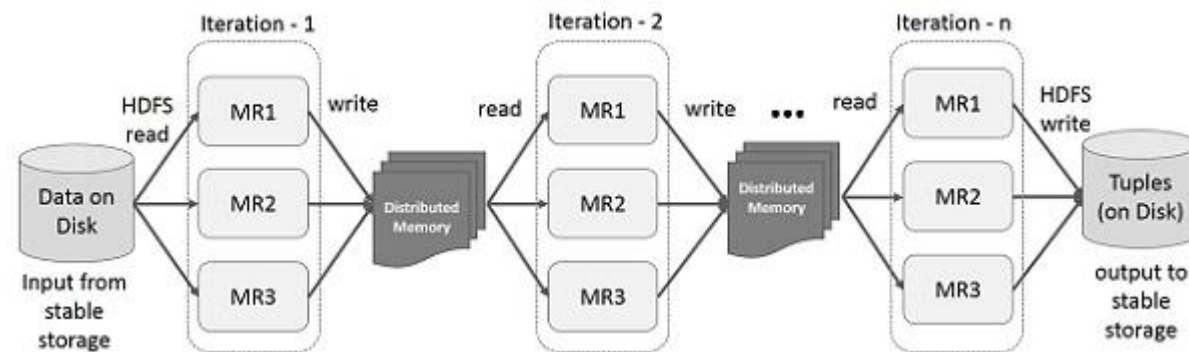
Interactive



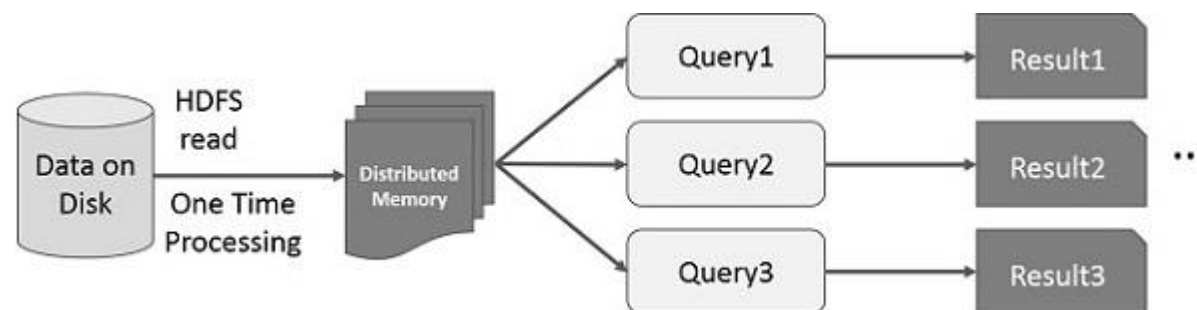
## RDD is sneller

- In-memory sharing is 10 tot 100 keer sneller

### Iterative



### Interactive



# Installatie

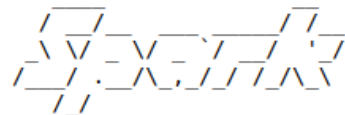
- De python-installatie kan eenvoudig gebeuren via

```
(base) bigdata@bigdata-VirtualBox:~$  
(base) bigdata@bigdata-VirtualBox:~$ pip install pyspark
```

- Hierna kan het gebruikt worden in een notebook

```
In [2]: !pyspark --version
```

```
21/11/08 14:56:32 WARN Utils: Your hostname, bigdata-VirtualBox resolves to a loopback address: 127.0.1.1; using 10.  
0.2.15 instead (on interface enp0s3)  
21/11/08 14:56:32 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address  
WARNING: An illegal reflective access operation has occurred  
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/home/bigdata/anaconda3/lib/python3.8/s  
ite-packages/pyspark/jars/spark-unsafe_2.12-3.2.0.jar) to constructor java.nio.DirectByteBuffer(long,int)  
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform  
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations  
WARNING: All illegal access operations will be denied in a future release  
Welcome to
```



version 3.2.0

```
Using Scala version 2.12.15, OpenJDK 64-Bit Server VM, 11.0.11  
Branch HEAD  
Compiled by user ubuntu on 2021-10-06T12:46:30Z  
Revision 5d45a415f3a29898d92380380cfd82bfc7f579ea  
Url https://github.com/apache/spark  
Type --help for more information.
```



## Installatie

- ▣ Indien Spark gecombineerd wordt met YARN moeten er nog drie zaken toegevoegd worden aan `.bashrc`
  - `export SPARK_LOCAL_IP="127.0.0.1"`
  - `export HADOOP_CONF_DIR="$HADOOP_HOME/etc/hadoop"`
  - `export YARN_CONF_DIR="$HADOOP_HOME/etc/hadoop"`



## Features

- ▣ In memory processing
- ▣ Distributed processing via parallellisatie
- ▣ Werkt met verschillende clusterbeheerders (Spark, Yarn, ...)
- ▣ Fout-tolerant
- ▣ Immutable/onverranderlijk -> nieuwe objecten ipv updates
- ▣ Lazy evaluation -> reken uit wanneer het nodig is
- ▣ Cache & persistentie
- ▣ Optimalisatie via dataframes
- ▣ Ondersteund een vorm van SQL



## Sparksession

- ▣ Entry point voor de pyspark applicatie
- ▣ Kan aangemaakt worden via
  - Builder ()
  - newSession()
- ▣ Maakt intern een sparkcontext aan
- ▣ Maximum 1 context per JVM maar meerdere sessions mogelijk



## RDD

- ▣ Op basis van de huidige sessie kan een RDD aangemaakt worden
  - ▬ Basisobject in Spark
  - ▬ Kan verdeeld worden in logische delen en uitgevoerd worden op verschillende nodes
- ▣ Aangemaakt door
  - ▬ Nieuwe objecten -> `parallelize()`
  - ▬ Inlezen van files -> `textFile / .read....`
  - ▬ Opgevraagd uit (NO)SQL Databases



## RDD - operaties

- ▣ Steeds in parallel

- ▣ Transformaties

- ▬ Lazy operations (berekenen nog niets)
- ▬ Geen updates maar een nieuw RDD gereturned
- ▬ flatMap(), map(), reduceByKey(), filter(), sortByKey(), ...

- ▣ Actions

- ▬ Starten een berekening
- ▬ Count(), collect(), first(), max(), reduce(), ...



# DataFrames

- ▣ Gelijkaardig aan pandas dataframe
- ▣ Werken volledig in parallel

```
In [60]: data = [('Harry', 'Potter', '1980-07-31', 'M', 1000000000),
                 ('Ronald', 'Wemel', '1980-04-01', 'M', 10),
                 ('Hermelijn', 'Griffel', '1979-09-19', 'F', 4000)]

columns = ["firstname", "lastname", "dob", "gender", "budget"]
df = spark.createDataFrame(data=data, schema = columns)
df.show()
```

firstname	lastname	dob	gender	budget
Harry	Potter	1980-07-31	M	1000000000
Ronald	Wemel	1980-04-01	M	10
Hermelijn	Griffel	1979-09-19	F	4000



## Shared variables

- ▣ Normaal heeft elke node zijn eigen variabelen
- ▣ Shared variabelen met read-write access is zeer inefficient
- ▣ Twee types shared variabelen aanwezig
  - Broadcast variables
  - Accumulators



## Broadcast variables

- ▣ Read-only variable cached on each machine
- ▣ Eenmalig verstuurd ipv elke nieuwe job
- ▣ Aangemaakt via `SparkContext.broadcast()`
- ▣ Delete
  - ▬ Temporary: `unpersist()`
  - ▬ Permanent: `destroy()`
- ▣ Toepassingen
  - ▬ Grote input dataset op elke node krijgen



## Accumulators

- ▣ Write-only shared variabelen waar enkel kan aan toegevoegd worden
  - Schrijven via `.add()`
- ▣ Enkel de driver kan het lezen via `.value`
- ▣ Applicaties
  - Counters
  - Sums
  - ...
- ▣ Aangemaakt via `SparkContext.accumulator()`



## Accumulators

- ▣ Standaard support voor numerieke accumulators, andere zelf te implementeren (overerven van AccumulatorParam)
- ▣ Let op dat Spark Lazy Evaluation doet
  - Schrijven naar de accumulator gebeurt enkel bij acties
  - Transformations kunnen dit ook doen maar worden slechts bij een actie uitgevoerd.

## PySpark SQL

- ▣ Veel gebruikte module om sql-queries uit te voeren op dataframes.

```
In [65]: df.createOrReplaceTempView("test")
df_male = spark.sql("select * from test where gender='M'")
df_male.show()

df_num_gender = spark.sql('select gender, count(*) from test group by gen
df_num_gender.show()
```

firstname	lastname	dob	gender	budget
Harry	Potter	1980-07-31	M	1000000000
Ronald	Wemel	1980-04-01	M	10

gender	count(1)
F	1
M	2