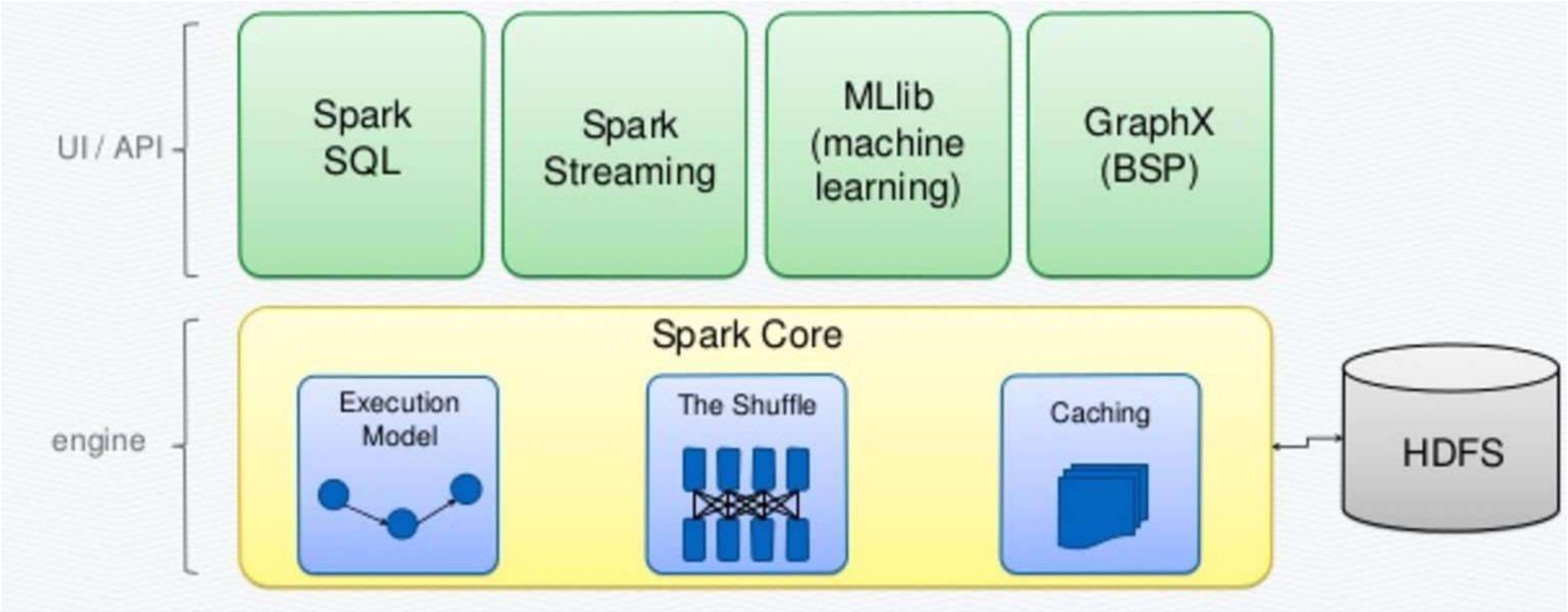
Spark Components



Spark SQL

- Spark package designed for working with structured data which is built on top of Spark Core
- Provides an SQL-like interface for working with structured data

DataFrames in Spark

- A DataFrame is a Dataset organized into named columns
- It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood
- DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs
- The DataFrame API is available in Scala, Java, Python, and R

DataFrame example in Spark 1.6.0

```
-----------------------------------------------------------
ordersRDD = sc.textFile('/FileStore/tables/orders')

from pyspark.sql import Row

ordersDF = ordersRDD.map(lambda x: Row(order_id = int(x.split(',')[0]), order_date = x.split(',')[1],
order_customer_id = int(x.split(',')[2]), order_status = x.split(',')[3])).toDF()

ordersDF.registerTempTable('orders')

sqlContext.sql("SELECT order_status, count(*) FROM orders GROUP BY order_status").show()
-----------------------------------------------------------
```

DataFrame example in data bricks Spark 2.4.4

```
# File location and type
file_location = '/FileStore/tables/athletes.csv'
file_type = 'csv'

# CSV options
infer_schema = 'False'
first_row_is_header = 'True'
delimiter = ','

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

df.printSchema()
```

DataFrame example in data bricks Spark 2.4.4

```
from pyspark.sql.types import StructField, IntegerType, FloatType, StructType, StringType, DoubleType

df_final = df.withColumn("ID", df["ID"].cast(IntegerType())) \
.withColumn("Age", df["Age"].cast(IntegerType())) \
.withColumn("Height", df["Height"].cast(FloatType())) \
.withColumn("Weight", df["Weight"].cast(FloatType())) \
.withColumn("Year", df["Year"].cast(IntegerType()))

df_final.printSchema()
```

DataFrame example in data bricks Spark 2.4.4

```
df_final.createOrReplaceTempView("athletes_final")
```

```
%sql
select * from athletes_final
```

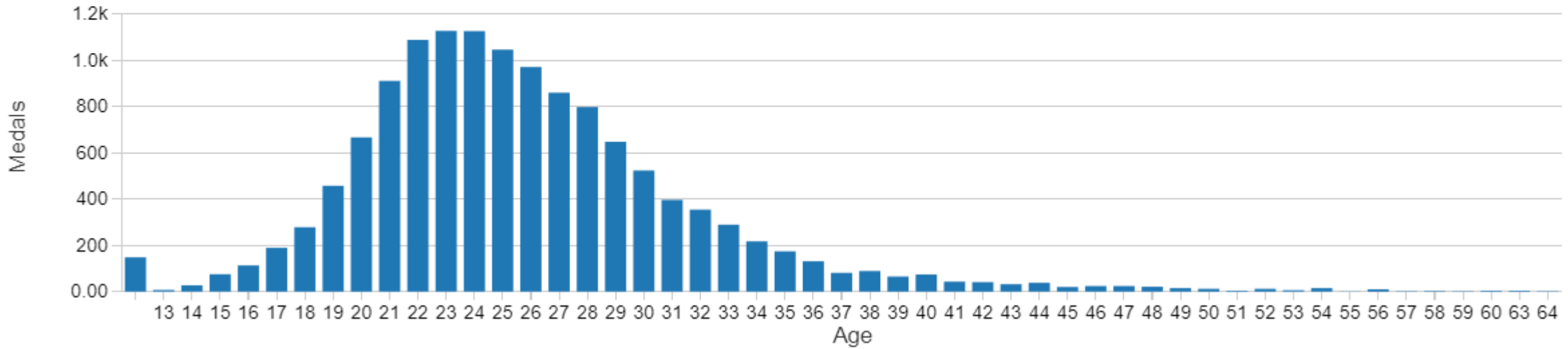
DataFrame example in data bricks Spark 2.4.4

Distribution of the age of gold medalists

```
%sql

select count(Medal) as Medals, Age from athletes_final where Medal = 'Gold' group by Age order by Age asc;
```

▸ (1) Spark Jobs

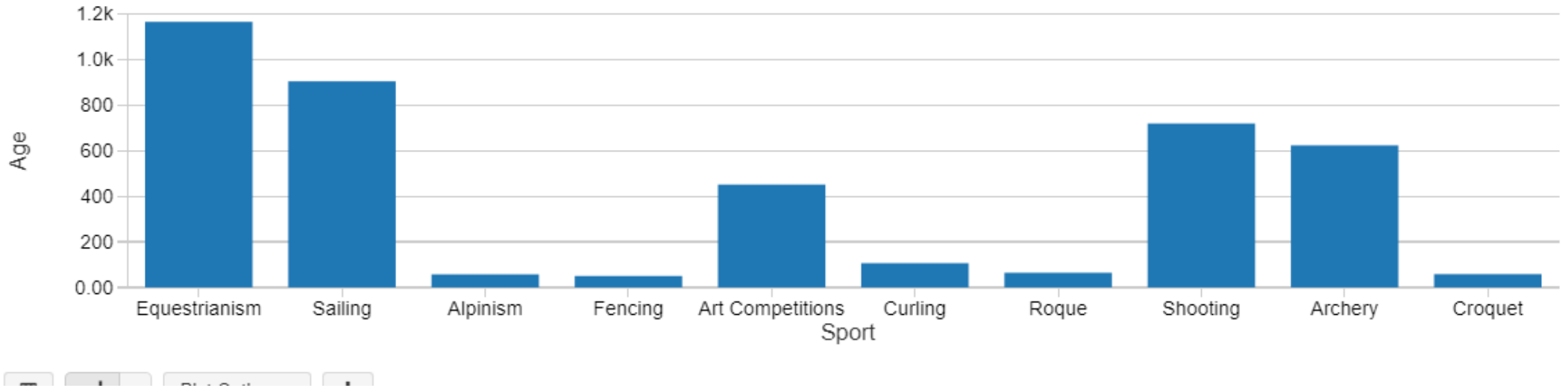


DataFrame example in data bricks Spark 2.4.4

```
%sql

select Sport, Age from athletes_final where Medal = 'Gold' and Age >= 50;
```

▸ (3) Spark Jobs

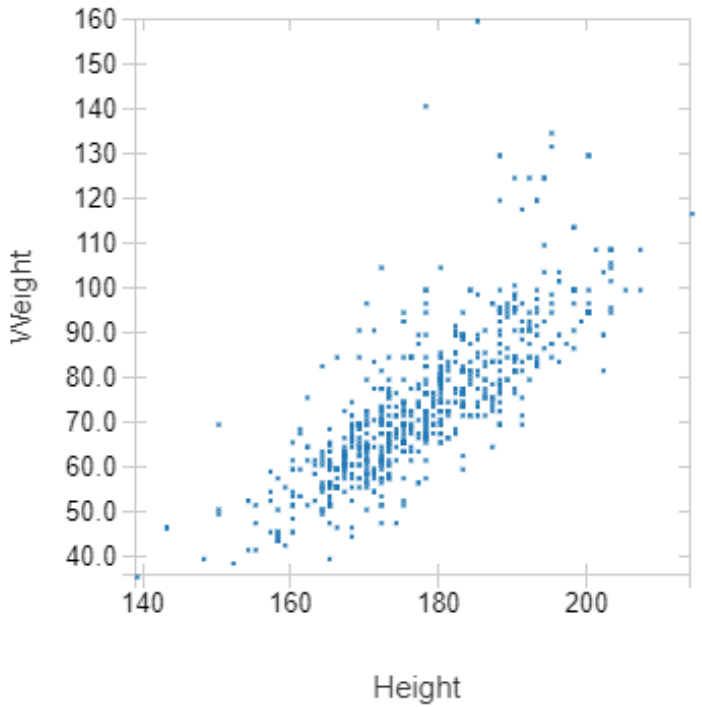


DataFrame example in data bricks Spark 2.4.4

Height vs Weight of Olympic Medalists

```
%sql
select Weight, Height from athletes_final where  Medal = 'Gold';
```

▸ (1) Spark Jobs



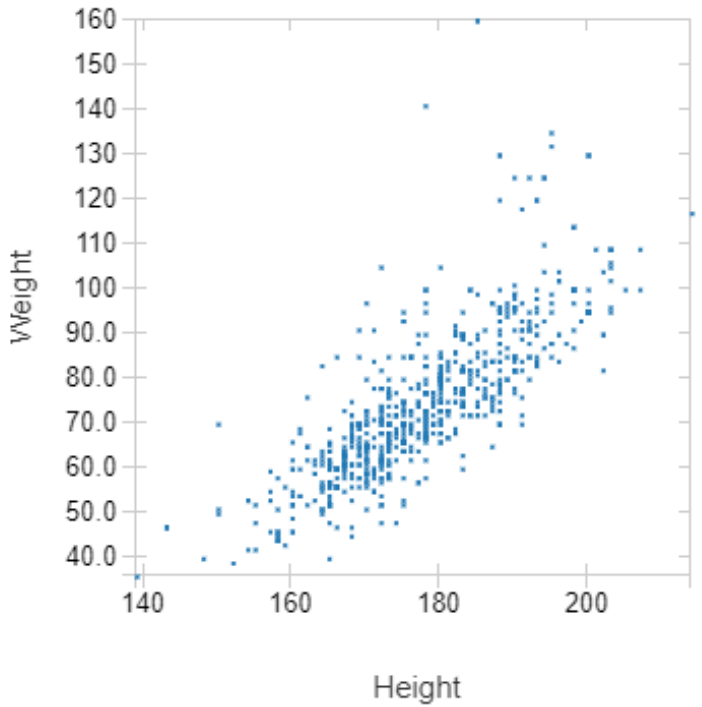
Showing sample based on the first 1000 rows.

DataFrame example in data bricks Spark 2.4.4

Height vs Weight of Olympic Medalists

```
%sql
select Weight, Height from athletes_final where  Medal = 'Gold';
```

▸ (1) Spark Jobs



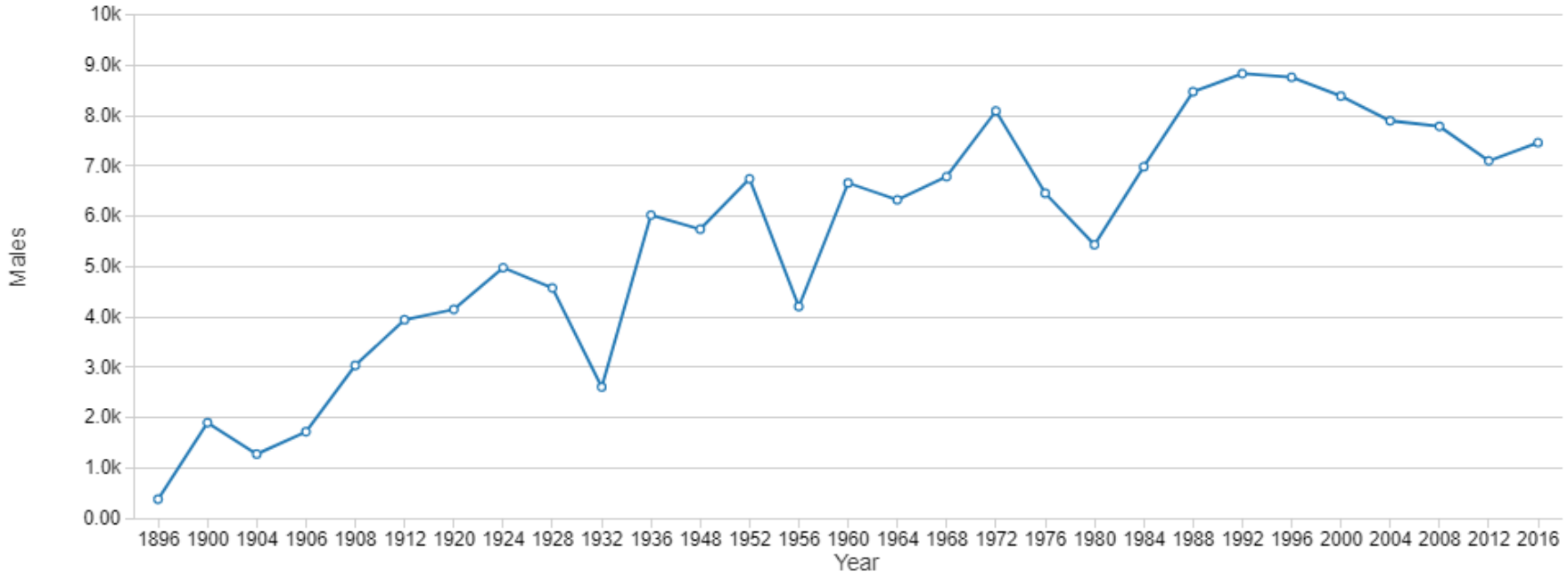
Showing sample based on the first 1000 rows.

DataFrame example in data bricks Spark 2.4.4

Variation of Male Athletes over time

```
%sql
select count(Gender) as Males, Year from athletes_final where Gender = 'M' and Season = 'Summer' group by Year order by Year asc;
```

▸ (1) Spark Jobs



Apache Flume

- Flume is a distributed service for efficiently collecting large amount of unstructured data
- Developed at Cloudera, now open sourced and managed by Apache Software Foundation
- Flume's architecture is based on streaming data flow (Ingest live data)
- HDFS is our storage point at Hadoop, we would like to ingest our data into HDFS
- A Flume agent is a JVM process that hosts a bunch of components through which the data flows from an external system into Hadoop Storage

Apache Flume

- A Flume agent configuration file is a set of Java properties file, comprising of a Source, Channel and Sink
  - Source --> Defines where to get the data from
  - Sink --> Defines where the data needs to be stored
  - Channel --> Interim queue between source and sink

Flume Configuration File

```
logAgent.sources = src
logAgent.channels = ch
logAgent.sinks = sink

logAgent.sources.src.type = exec
logAgent.sources.src.command = tail -F /tmp/data_log
logAgent.sources.src.channels = ch

logAgent.channels.ch.type = memory
logAgent.sinks.sink.channel = ch
logAgent.sinks.sink.type = hdfs
logAgent.sinks.sink.hdfs.path = hdfs://localhost:8020/log_data
logAgent.sinks.sink.hdfs.fileType = DataStream
logAgent.sinks.sink.hdfs.writeFormat = Text
```

Spark Streaming

- Running on top of Spark, Spark Streaming provides an API for manipulating data streams that closely match the Spark Core’s RDD API
- Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams
- Data can be ingested from many sources like Kafka, Flume, Twitter, ZeroMQ, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window
- Finally, processed data can be pushed out to filesystems, databases, and live dashboards

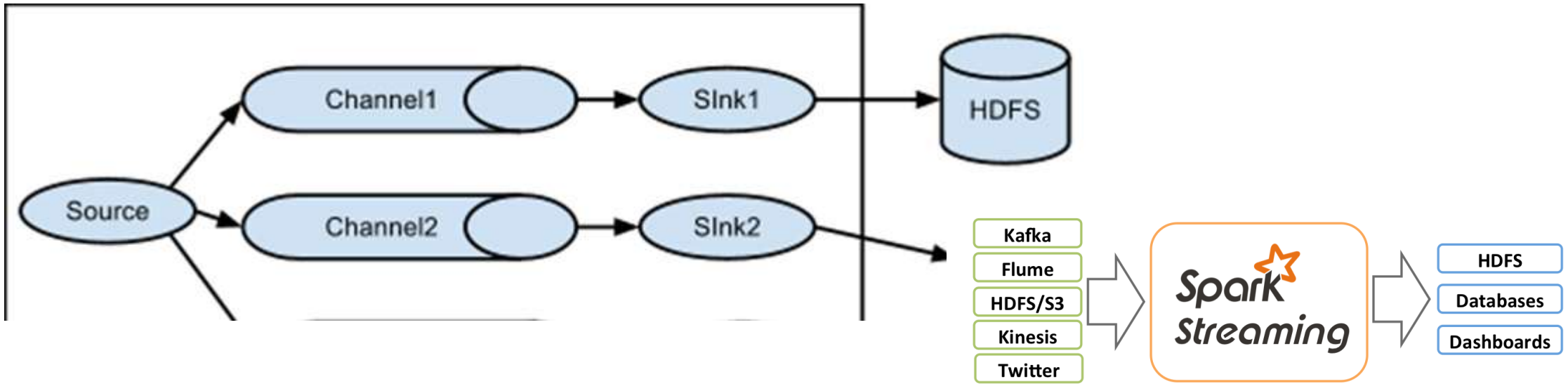
Spark Streaming

Spark Streaming

- Internally, it works as follows, Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches

Spark Streaming

Flume & Spark Streaming Integration



Spark Streaming Example

```
from pyspark import SparkConf, SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.flume import FlumeUtils
from operator import add

conf = SparkConf().setAppName("Streaming Log Count")
sc = SparkContext(conf=conf)

sc.setLogLevel("ERROR")

ssc = StreamingContext(sc, 30)

agents = [("localhost", 9999)]
pollingStream = FlumeUtils.createPollingStream(ssc, agents)

logsRDD = pollingStream.map(lambda log: log[1])

logCount = logsRDD.map(lambda x: (x.split(" ")[3].replace("[","").replace("]",""),1)).reduceByKey(add)

logCount.saveAsTextFiles("/LogCount/")

ssc.start()

ssc.awaitTermination()
```

Flume Sink to Spark Streaming

```
logAgent.sources = src
logAgent.channels = ch sparkmem
logAgent.sinks = sink spark

# Bind the source and sink to the channel
logAgent.sources.src.channels = ch sparkmem

logAgent.sources.src.type = exec
logAgent.sources.src.command = tail -F /tmp/data_log

# Describe the sink
logAgent.sinks.sink.channel = ch
logAgent.sinks.sink.type = hdfs
logAgent.sinks.sink.hdfs.path = hdfs://localhost:8020/log_data
logAgent.sinks.sink.hdfs.fileSuffix = .txt
logAgent.sinks.sink.hdfs.rollInterval = 120
logAgent.sinks.sink.hdfs.rollSize = 1048576
logAgent.sinks.sink.hdfs.rollCount = 100
logAgent.sinks.sink.hdfs.fileType = DataStream

logAgent.sinks.spark.channel = sparkmem
logAgent.sinks.spark.type = org.apache.spark.streaming.flume.sink.SparkSink
logAgent.sinks.spark.hostname = localhost
logAgent.sinks.spark.port = 9999

logAgent.channels.ch.type = memory
logAgent.channels.ch.capacity = 1000
logAgent.channels.ch.transactionCapacity = 100

logAgent.channels.sparkmem.type = memory
logAgent.channels.sparkmem.capacity = 1000
logAgent.channels.sparkmem.transactionCapacity = 100
```

Spark’s Machine Learning Library

- Enables powerful interactive and analytical applications across both streaming and historical data while inheriting Spark’s ease of use and fault tolerance characteristics
- Built on top of Spark, MLlib is a scalable machine learning library that delivers both high-quality algorithms and blazing speed

Spark’s Machine Learning Library

- Spark's ML-Lib has built-in modules for classification, regression, clustering, recommendations etc algorithms, and the library takes care of running these algorithms across a cluster behind the scenes
- This completely abstracts the programmer from implementing the ML algorithm or the complexities of running it across a cluster

Spark’s Machine Learning Library

- Spark's ML-Lib achieves the following purpose
  - Abstraction
  - Performance

Note: MLlib is on a shift to DataFrame based API

Spark's Machine Learning Library example

```
dataPath = '/FileStore/tables/user_artist_data.txt'
rawUserArtistData = sc.textFile(dataPath)
#rawUserArtistData.count()
#rawUserArtistData.take(10)

rawUserArtistData.map(lambda x: float(x.split( )[2])).stats()

from pyspark.mllib.recommendation import Rating,ALS

uaData = rawUserArtistData.map(lambda x: x.split()).filter(lambda y: float(y[2]) >= 20).map(lambda z: Rating(z[0], z[1], z[2]))

#uaData.take(10)
uaData.cache()

# Hidden factors, max iterations, lambda (control the quality of ALS results)
model=ALS.trainImplicit(uaData,10,5,0.01)

user = 1059637

# user, no of recommendations
recommendations = model.recommendProducts(user,5)

print(recommendations)
```

Spark's Machine Learning Library example

LATENT FACTOR COLLABORATIVE FILTERING

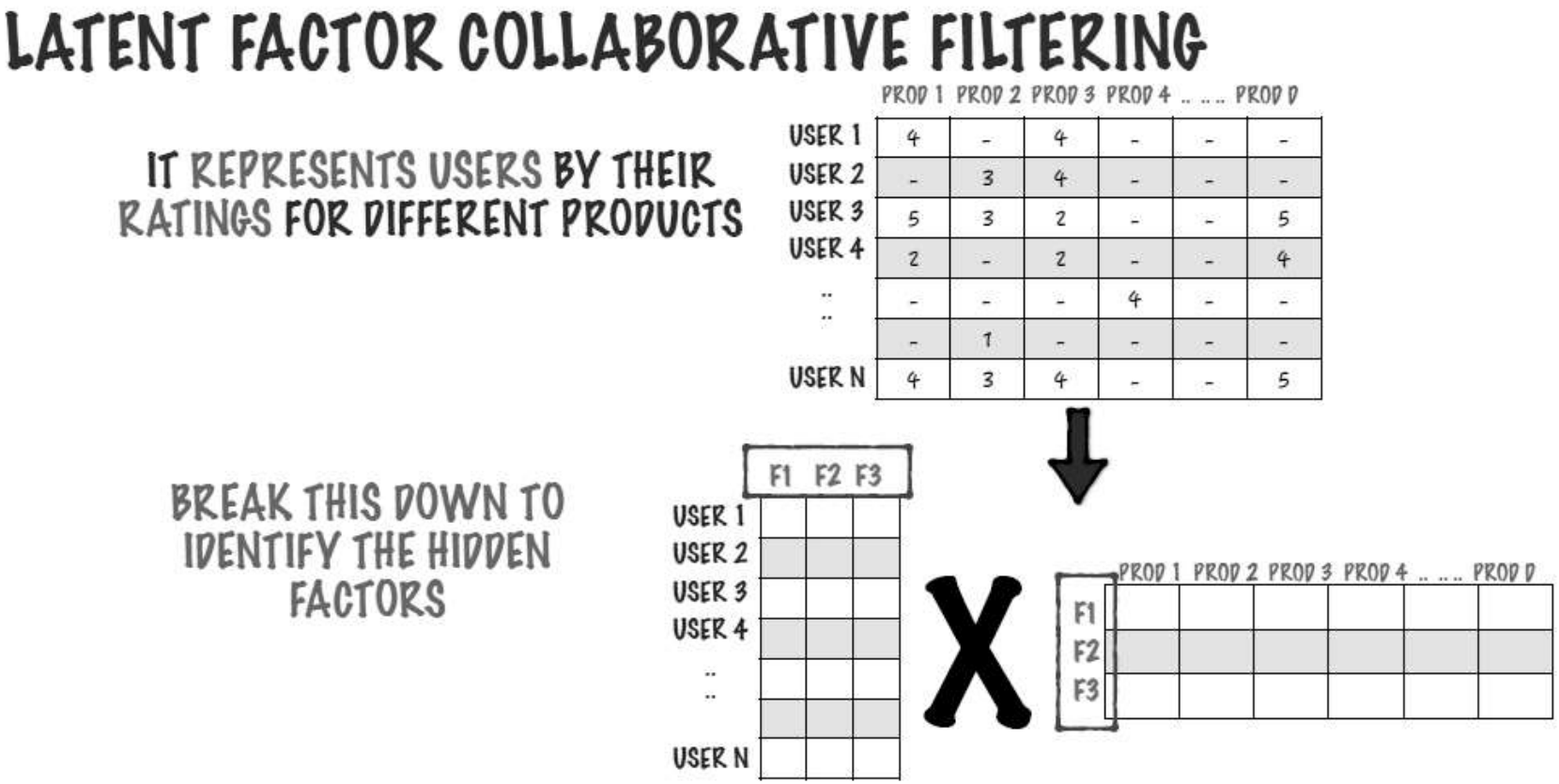
TO IDENTIFY HIDDEN FACTORS

YOU NEED A USER-PRODUCT-RATING MATRIX

	PROD 1	PROD 2	PROD 3	PROD 4	.. .. ..	PROD D
USER 1	4	-	4	-	-	-
USER 2	-	3	4	-	-	-
USER 3	5	3	2	-	-	5
USER 4	2	-	2	-	-	4
..	-	-	-	4	-	-
..	-	1	-	-	-	-
USER N	4	3	4	-	-	5

IT REPRESENTS USERS BY THEIR RATINGS FOR DIFFERENT PRODUCTS

Spark's Machine Learning Library example



Spark's Machine Learning Library example