CERN School of Computing

Big Data Technologies and Physics Analysis with Apache Spark

Inverted CERN School of Computing 2019

Evangelos Motesnitsalis

4-6 March 2019

CERN openlab

Learning Goals

- Important big data concepts
- Main architecture characteristics of big data technologies
- Popular big data frameworks such as Apache Hadoop and Apache Spark
- Basic physics analysis concepts and frameworks
- Connecting tools between big data and High Energy Physics
- Example of physics analysis with Apache Spark

Contents

Who am I?

I am…

Technical Coordinator for 2 months
(still Data Engineer at heart)

Former Big Data Devops Support Engineer
and Escalation Engineer @AWS

Research Fellow in 2017 – today
Technical Student in 2014 – 2015
Summer Student in 2013

MSc Distributed Systems @Imperial College London

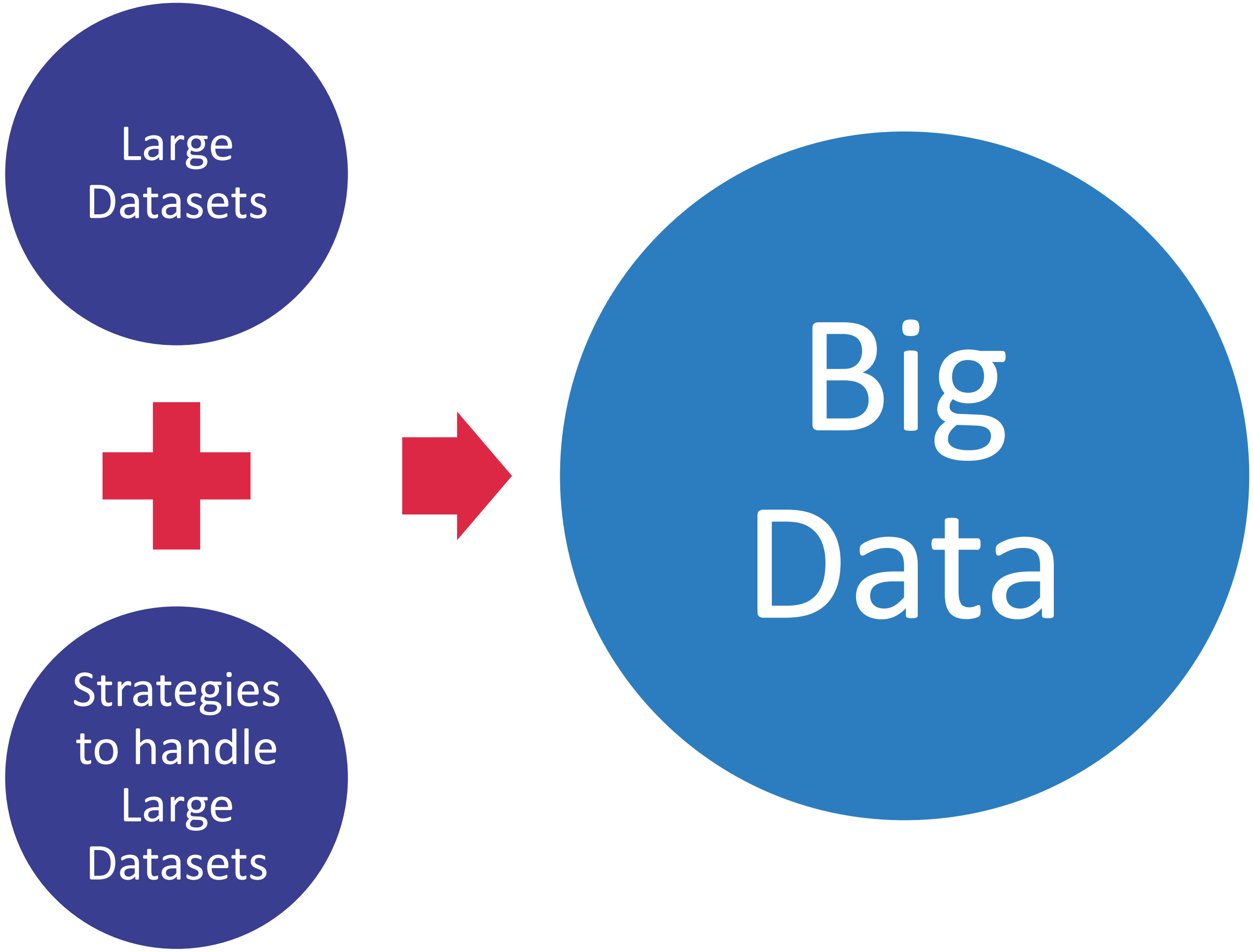
5+ years of Big Data experience

Studies at King's College London and Aristotle University of Thessaloniki

emotes@cern.ch

Introduction to Big Data

Introduction to Big Data



Introduction to Big Data

A bit of history…

18.000 BC: earliest signs of humans storing and analyzing data (Ishango bone)

2004: «MapReduce: Simplified Data Processing on Large Clusters» by Google

2005: The term “Big Data” emerges

2006: Introduction of Apache Hadoop

2010: «The amount of data from the beginning of human civilization to 2003 is generated every 2 days» E. Schmidt

Today: ~15 ZB per year

Big Data Systems Architecture

Architecture Overview

Top Level Abstractions

Distributed Processing Frameworks

Cluster Resource Manager

Distributed Filesystem

Cluster Node	Cluster Node	Cluster Node	Cluster Node	Cluster Node	Cluster Node

Architecture Principles

Resource Pooling: combining storage space, CPU and memory for different use cases and frameworks

High Availability: fault tolerance for source code execution and hardware components

Scalability: horizontal with new machines, vertical with bigger machines

Data Persistence: high replication factor and automatic restoration

Data Ingestion: ability to import raw data, RDBMS data, etc.

Parallelization: follow programming patterns that allow batch processing

Distributed File System

A software framework that allows users to acess and process distributed data

Same semantics and interfaces as Local Filesystems

Transparency everywhere: access, location, concurrency, failure, migration

Heterogeneity and Scalability

Usually centralized yet highly available metadata

Typical examples:

- Hadoop FileSystem (HDFS),
- EOS,
- Windows DFS, etc.

Cluster Manager

A software framework that runs distributely on cluster nodes

Multiple software components, multiple execution locations

Resource allocation and service configurations

High Availability, Scalability, and Resource Pooling are usually handled by Cluster Managers

Usually follows master – slave architecture principles

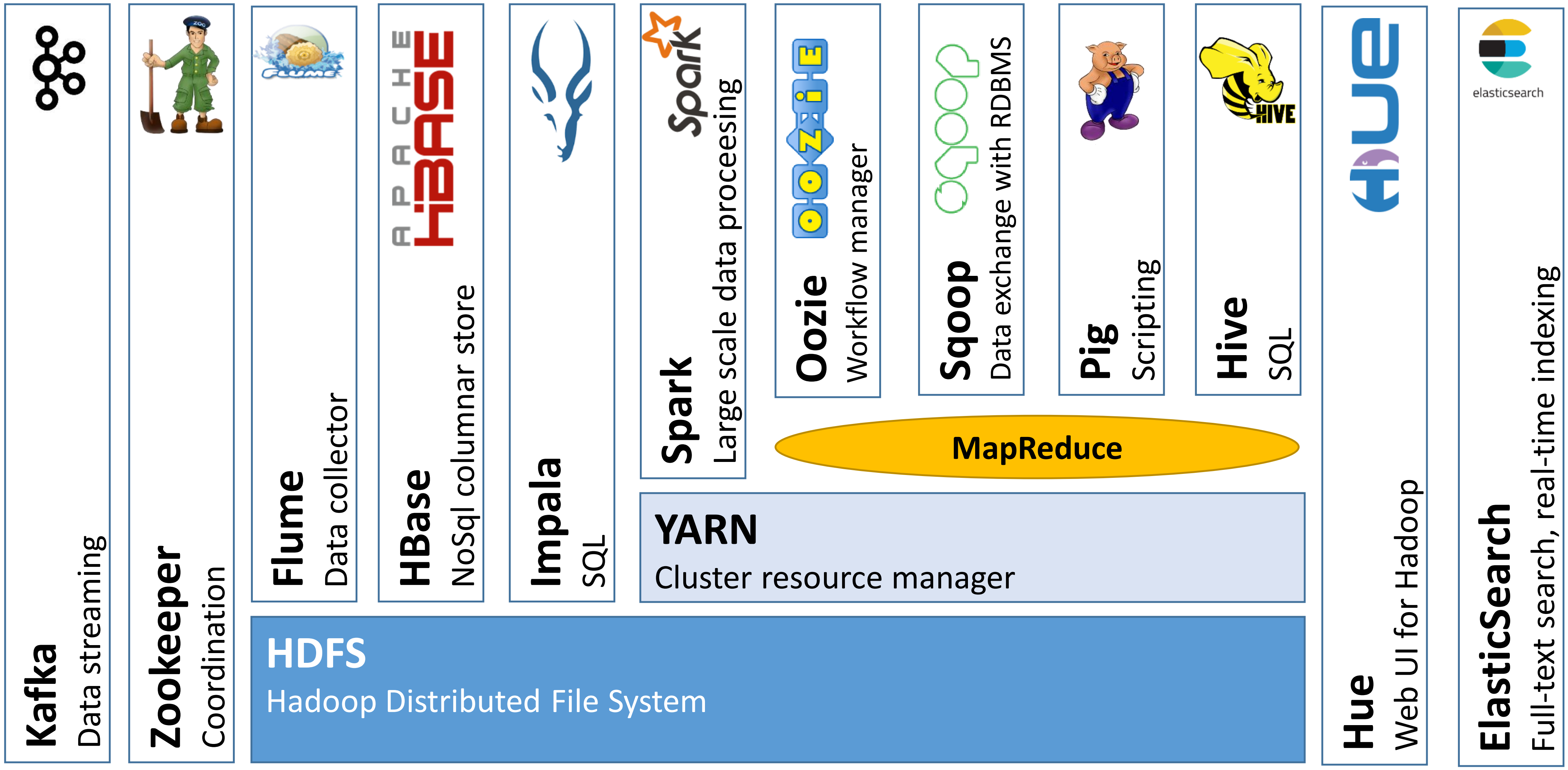
Typical open source examples:
- Apache YARN,
- Apache Mesos,
- Kubernetes, etc.

Distributed Processing Framework

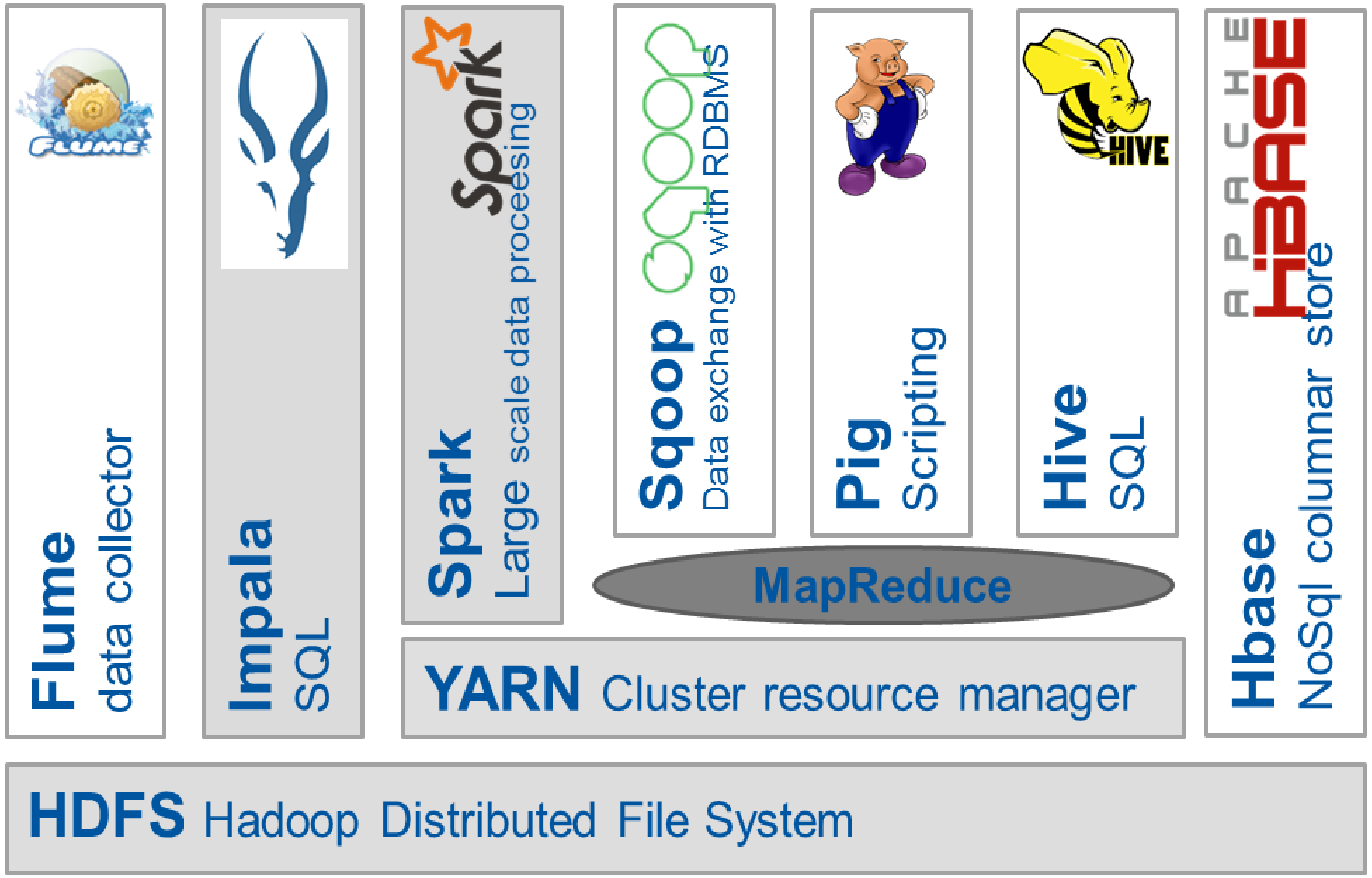
- A software framework that allows distributed computations
- Usually follows specific programming models (e.g. MapReduce)
- Code is automatically deployed in multiple locations
- Heterogeneity and Scalability
- Most distributed processing frameworks are highly interconnected (especially in the Hadoop Ecosystem)
- Typical examples:
  - Hadoop MapReduce,
  - Apache Spark, etc.

Big Data Frameworks

The Big Data Ecosystem



The Hadoop Ecosystem



Apache Hadoop

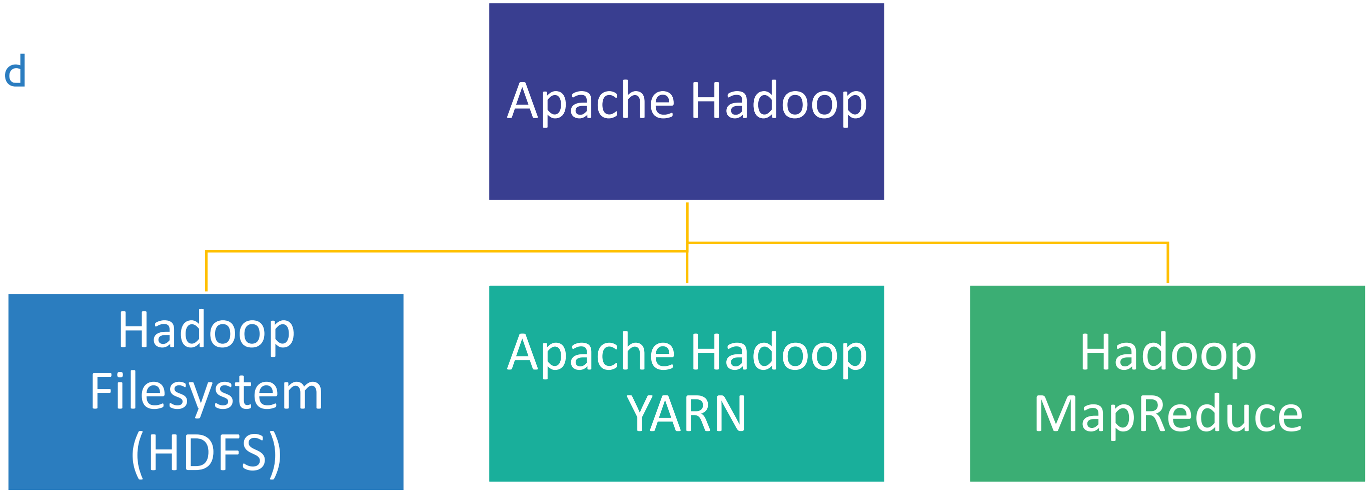
Apache Hadoop

A Framework for Large-scale Distributed Data Processing

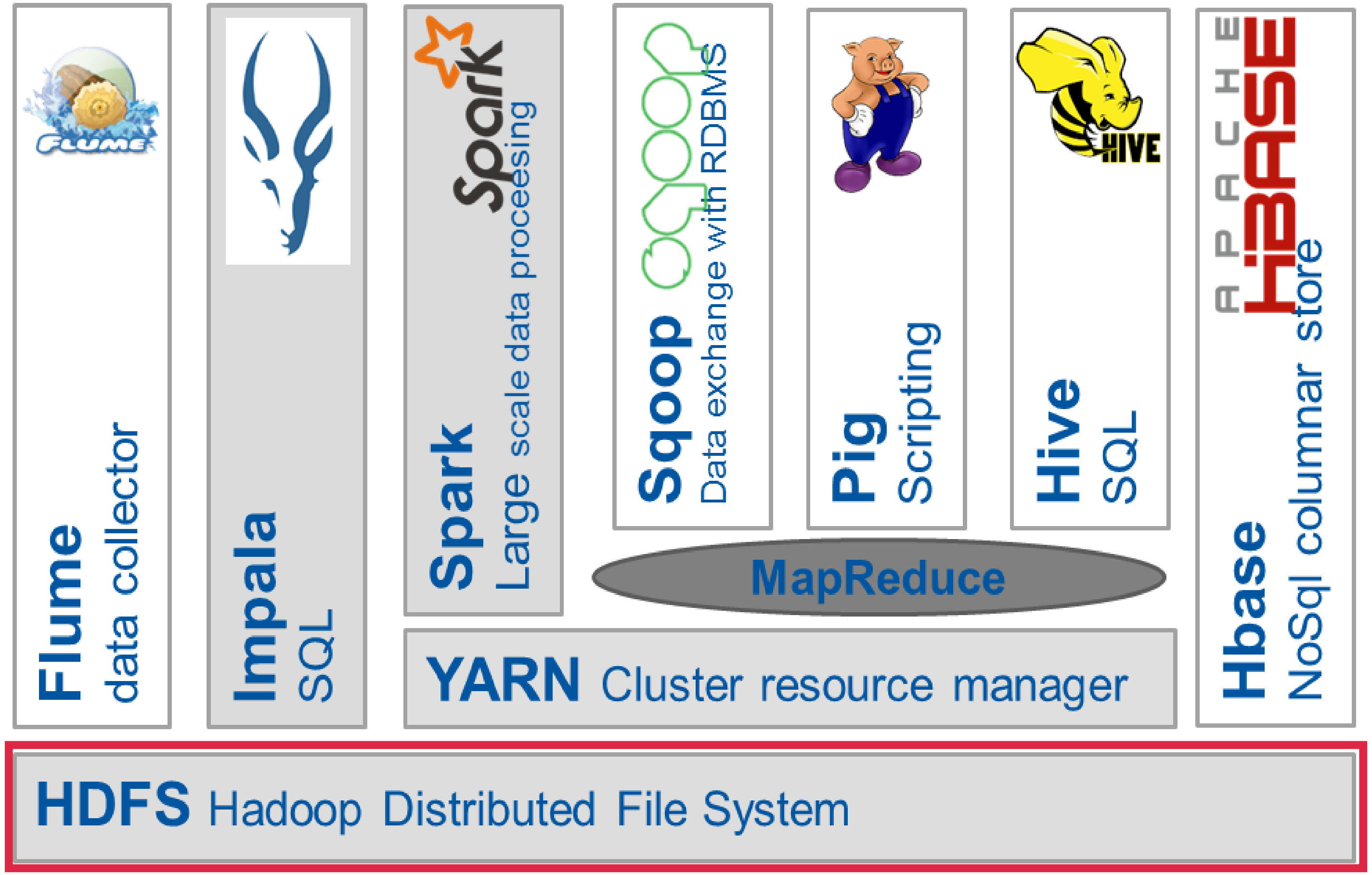
4 Vs: Volume, Variety, Velocity, Veracity

Runs on commodity hardware and based on Data Locality concepts

Open source



The Hadoop Ecosystem



Hadoop Distributed File System

The Filesystem of Hadoop

Fault tolerant – multiple replicas

Rack awareness

Scalable – design for high throughput

Minimal data motion and rebalance

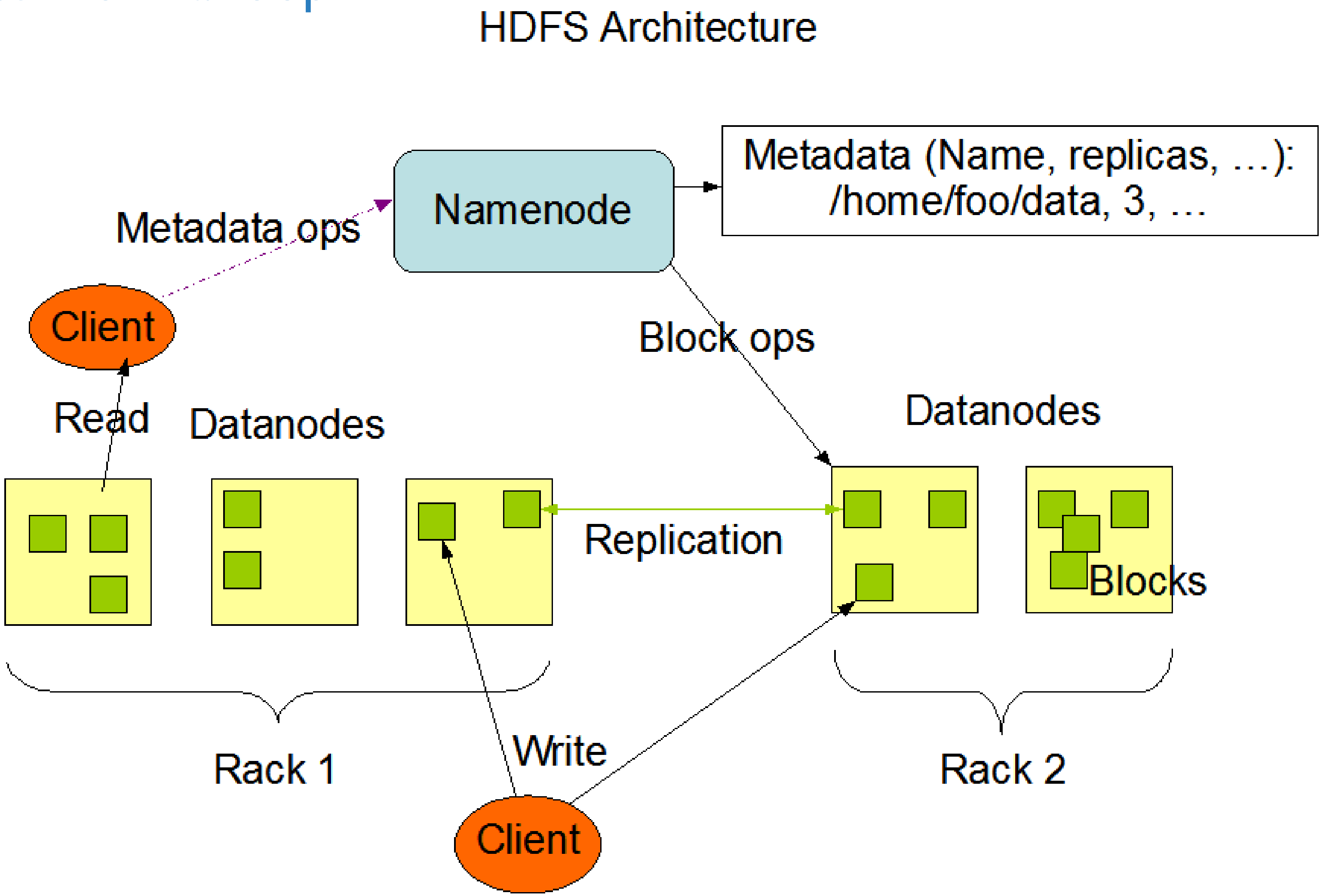
Files **cannot** be modified –
“Write Once – Read Many”

Consists of:

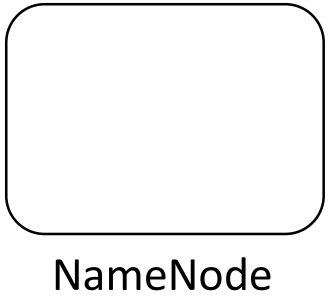
- 1 or 2 Namenodes (2 in HA)
- 1 Datanode per cluster node

Hadoop Distributed File System

The Filesystem of Hadoop



Hadoop Distributed File System

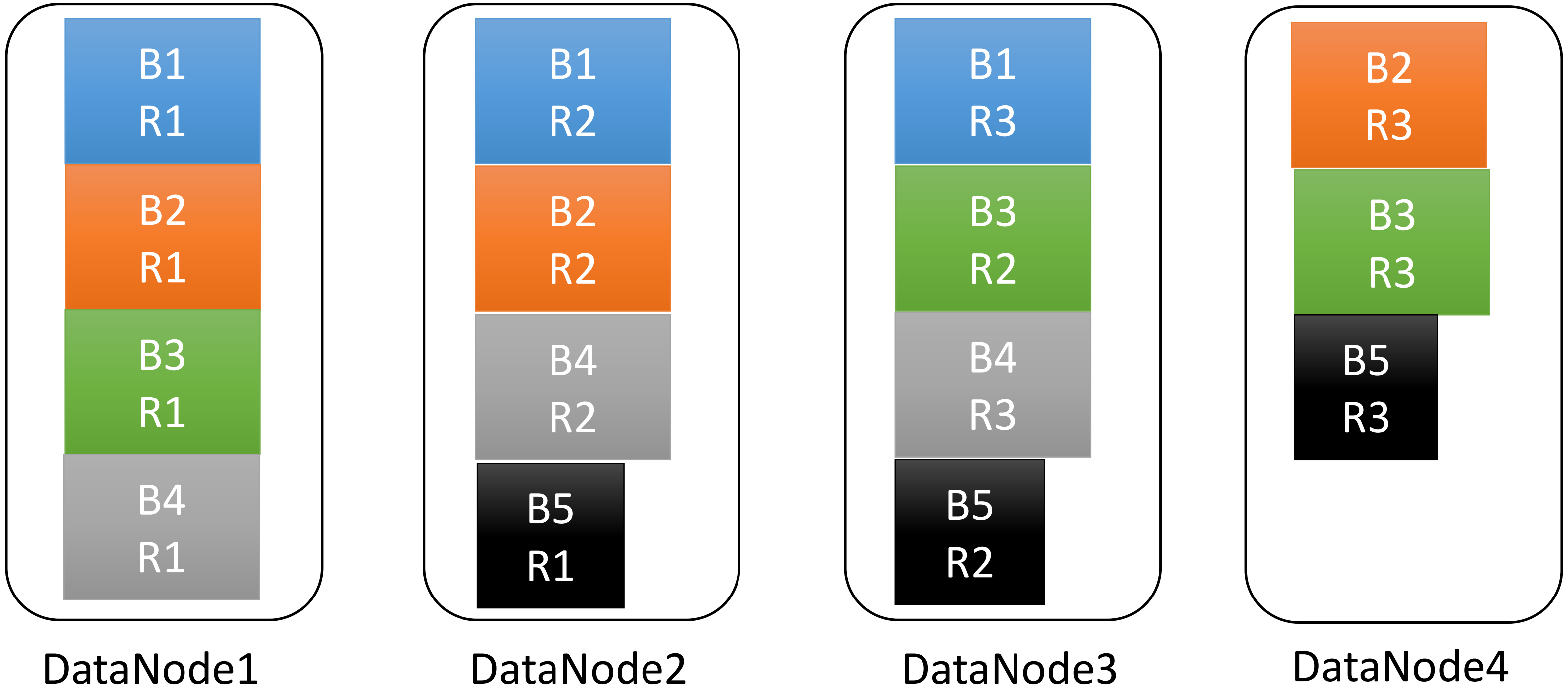


1)One 1176 MB File to be stored on HDFS

2) Splitting into 256MB blocks

3) Ask NameNode where to put them

4) Blocks with their replicas (by default 3) are distributed across Data Nodes



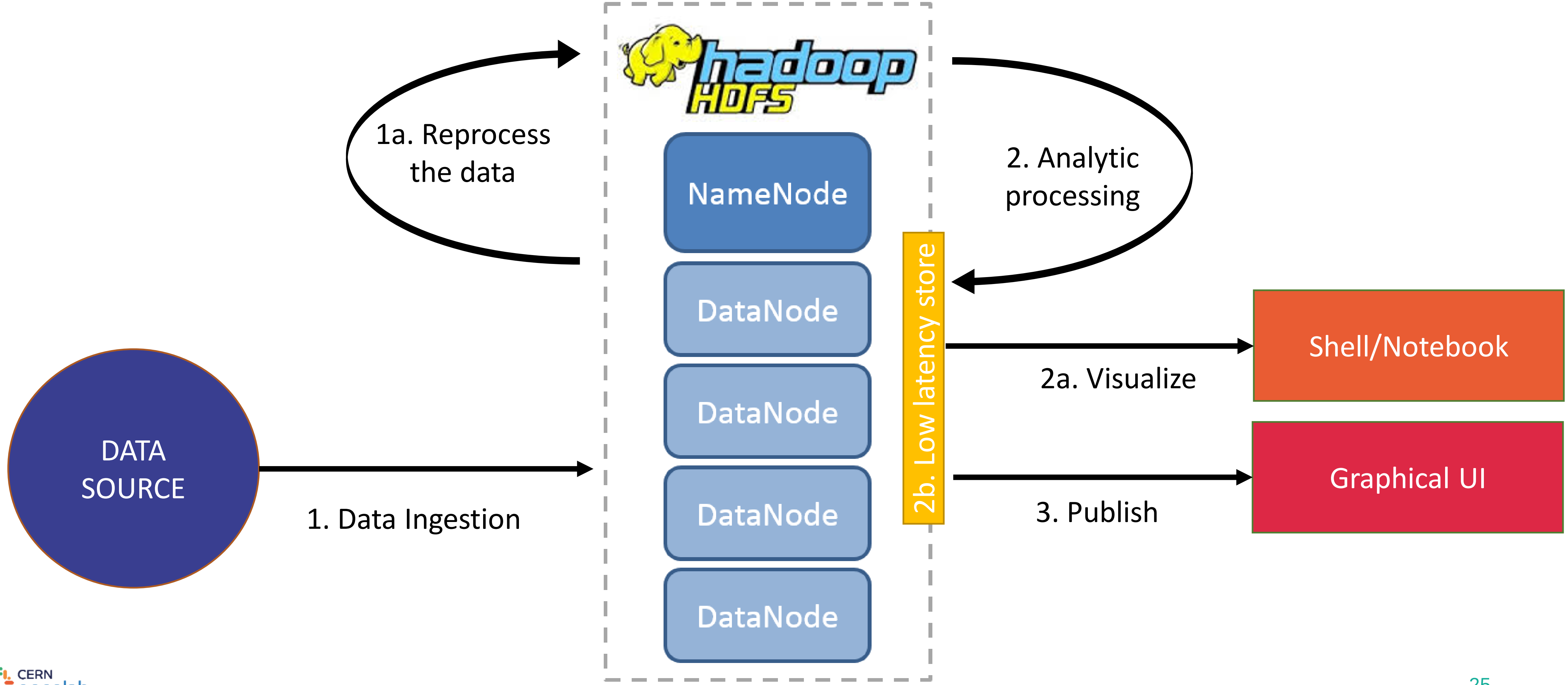
Hadoop Distributed File System

Interacting with HDFS

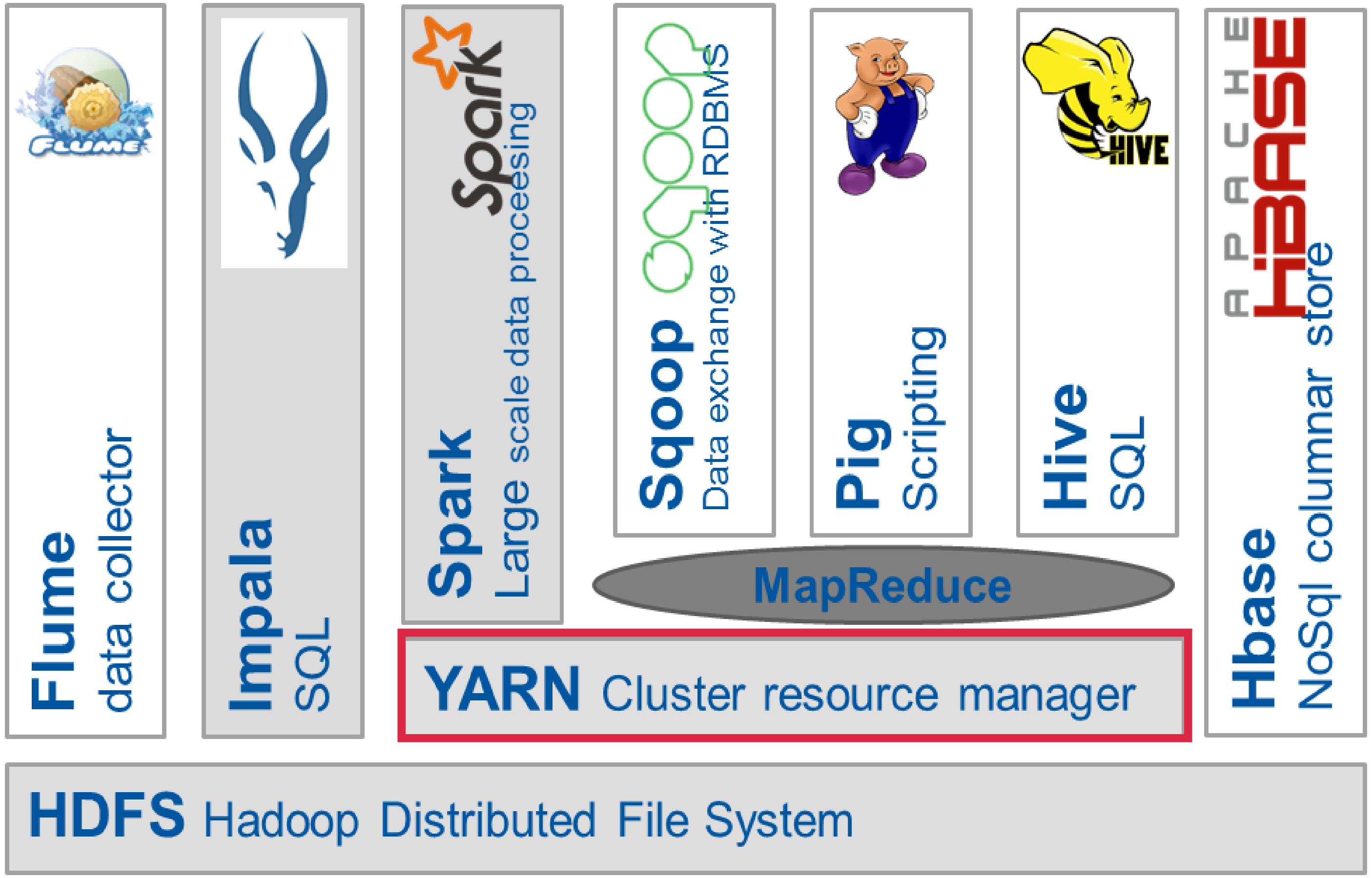
```
hdfs dfs -ls                   #listing home dir
hdfs dfs -ls /user             #listing user dir…
hdfs dfs -du -h /user          #space used
hdfs dfs -mkdir newdir         #creating dir
hdfs dfs -put myfile.csv .     #storing a file on HDFS
hdfs dfs -get myfile.csv .     #getting a file fr HDFS
```

Hadoop Distributed File System

Data Flow in HDFS



The Hadoop Ecosystem



Apache Hadoop YARN

Yet Another Resource Negotiator

Manages cluster computing resources in Hadoop

Creates the environment for Hadoop applications and deploys them

Negotiates with the applications the CPU and Memory resources that will be assigned to them

Utilizes different user queues and different schedulers: Fair, Capacity, and FIFO

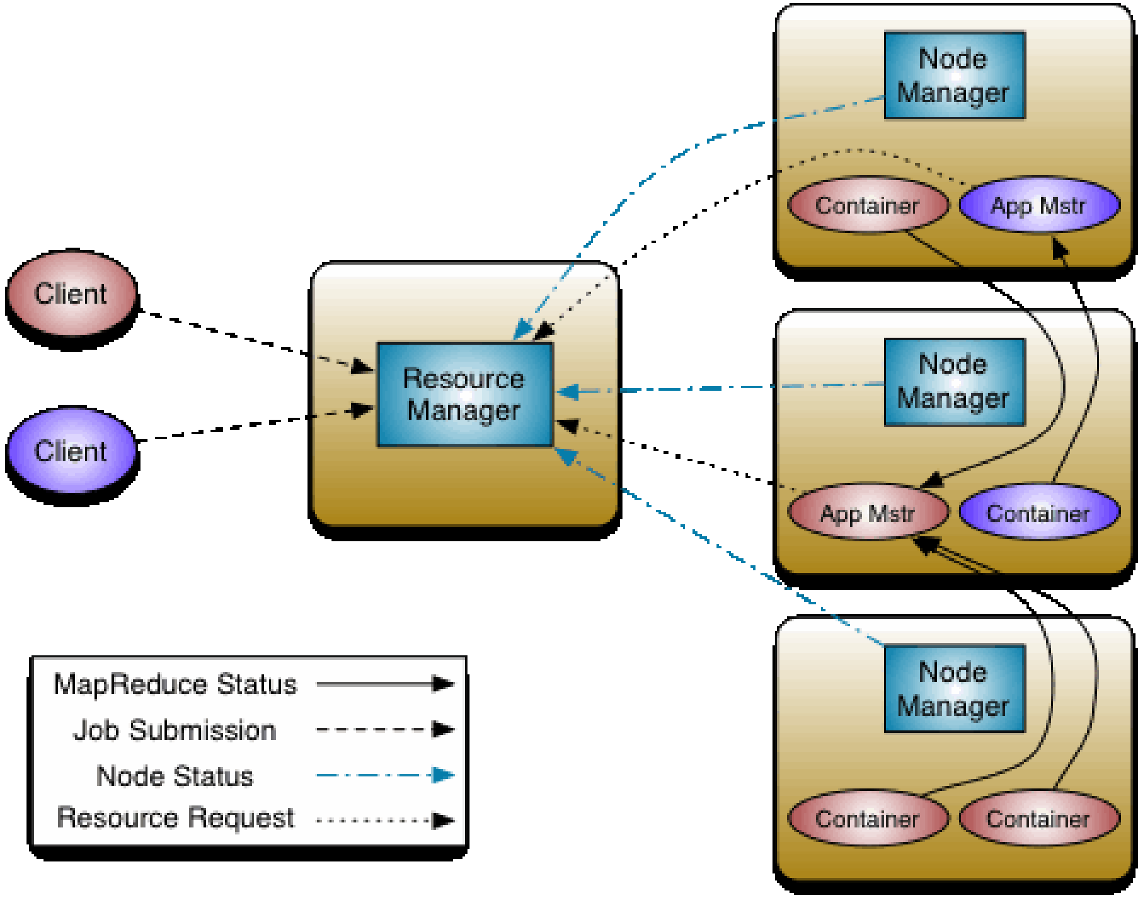
Each application relies on an Application Master

Consists of:

- 1 Resource Manager in the master node
- 1 Node Manager per cluster node

Apache Hadoop YARN

Yet Another Resource Negotiator

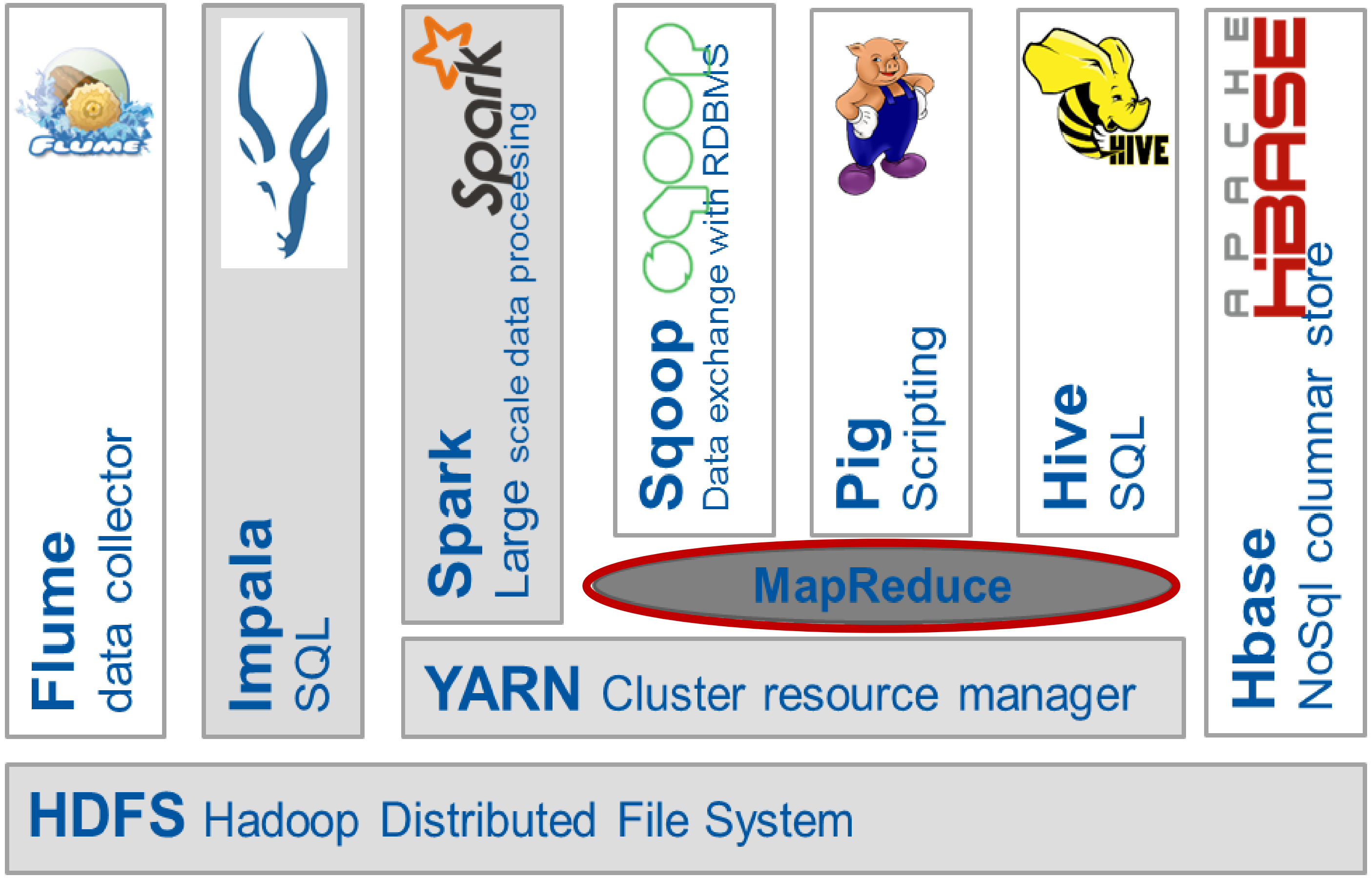


Apache Hadoop YARN

Interacting with YARN

```
yarn application –list             #listing apps submited
yarn application -status <id>      #details about app
yarn application –kill <id>        #kill running app
```

The Hadoop Ecosystem



Hadoop MapReduce

The First Batch Processing Framework

- MapReduce is a programming model for parallel processing
- Executes Java code in parallel
- Contains two stages: Map & Reduce
- Optimized for local data access
- Good for huge data sets and offline analysis but does not fit every use case
- Time consuming and not interactive

Hadoop MapReduce

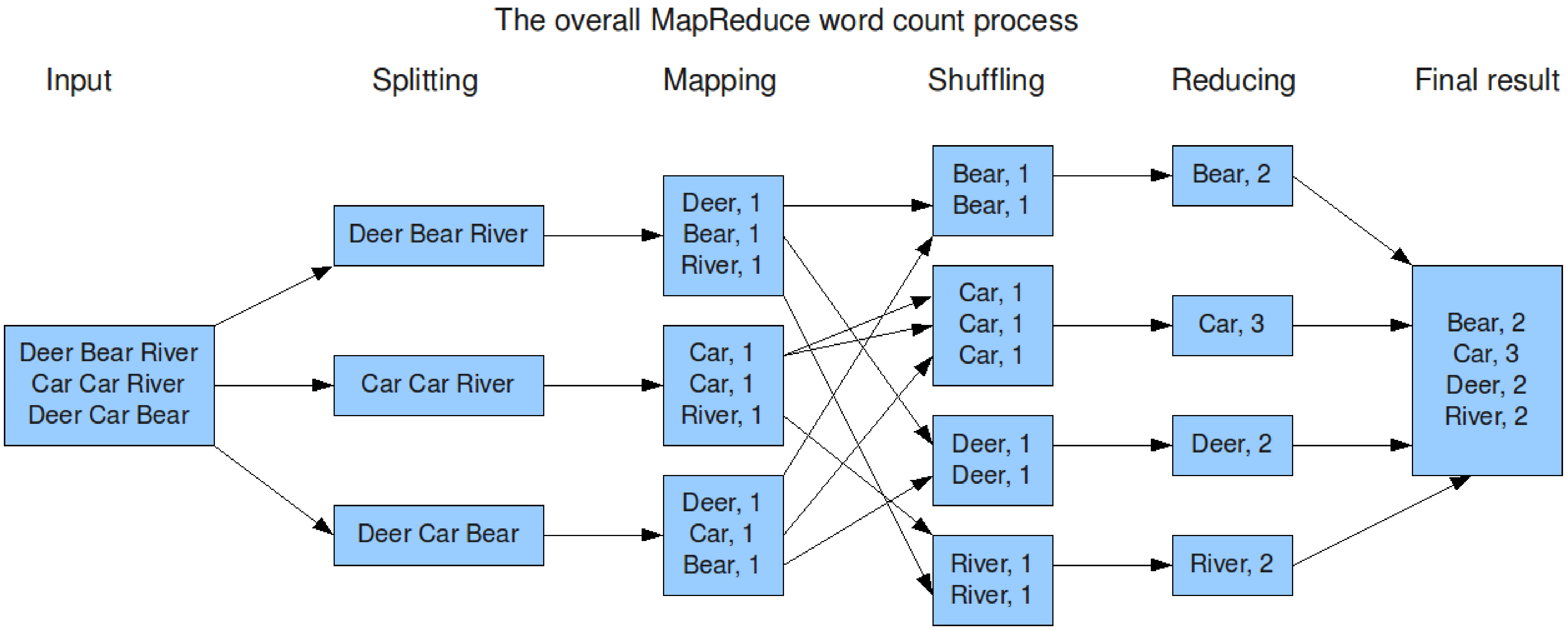
Hello World – aka « Wordcount »

```
//MAP method body
map(String key, String value)
// key: document name
// value: document contents
 for each word w in value
   EmitIntermediate(w, "1")
```

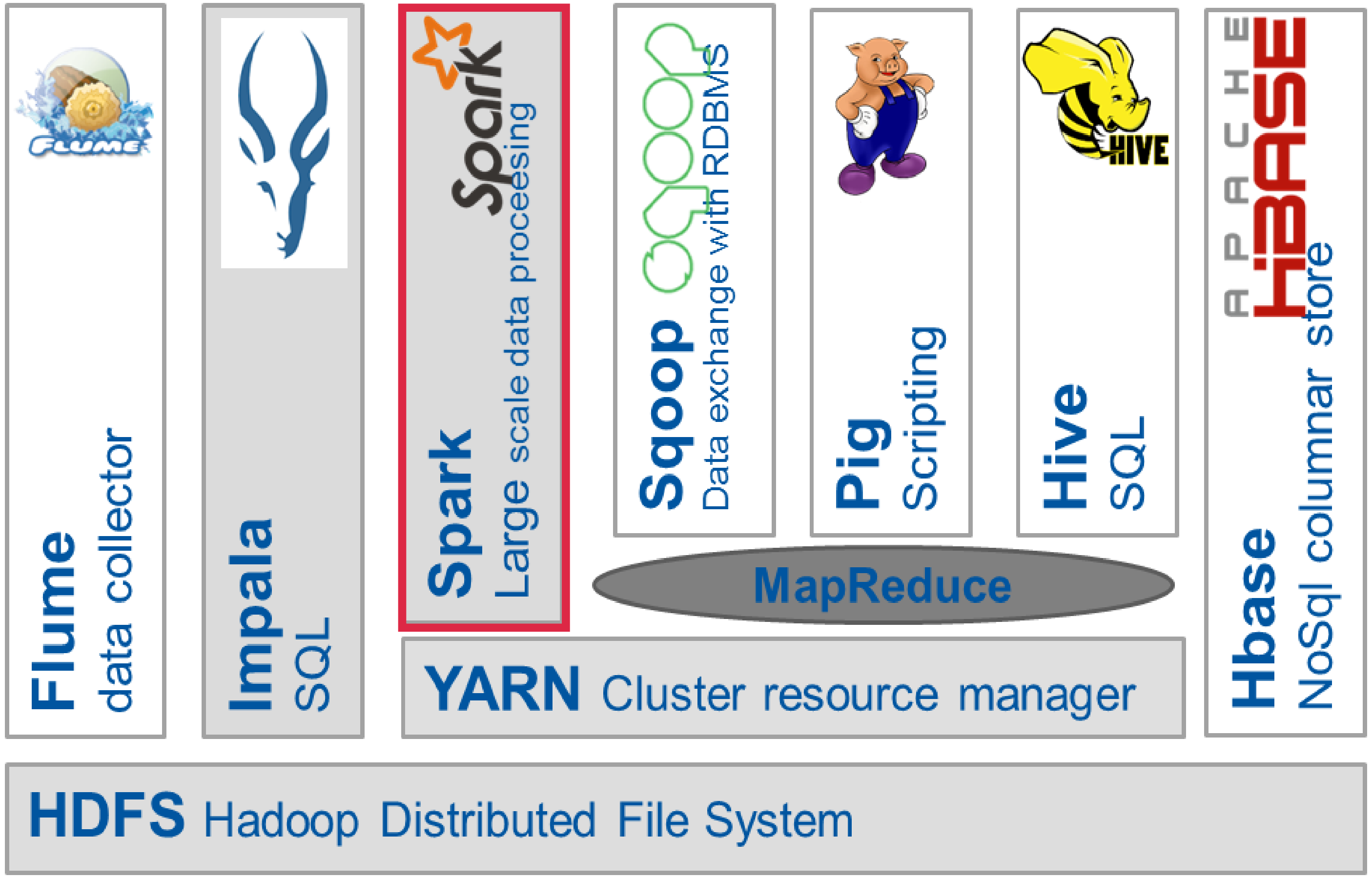
```
//REDUCER method body
reduce(String key, Iterator values):
// key: word
// values: a list of counts
  for each v in values:
    result += ParseInt(v);
    Emit(AsString(result))
```

Hadoop MapReduce

Hello World – aka « Wordcount »



The Hadoop Ecosystem



Apache Spark

Apache Spark

Overview

Apache Spark is an open source cluster computing framework

Brought fast, iterative, near real-time processing with no strict programming model – everything that MapReduce lacked.

APIs in Python, Scala, Java, R

Compatible with multiple cluster managers:

- Apache YARN
- Apache Mesos
- Kubernetes
- Standalone

Multiple File Formats and Filesystem Compatibility

Consists of multiple components:

- Spark SQL
- Spark Mlib
- Spark Graph
- Spark Structured Streaming

Apache Spark

Basic Concepts of Apache Spark

Spark supports complex processing patterns based on DAG

RDDS: Resilient Distributed Dataset, the basic abstraction of Spark is collection of partitioned data with primivite values

Directed Acyclic Graph: A finite directed graph with no directed cycles

Spark supports two types of operations: transformations and actions

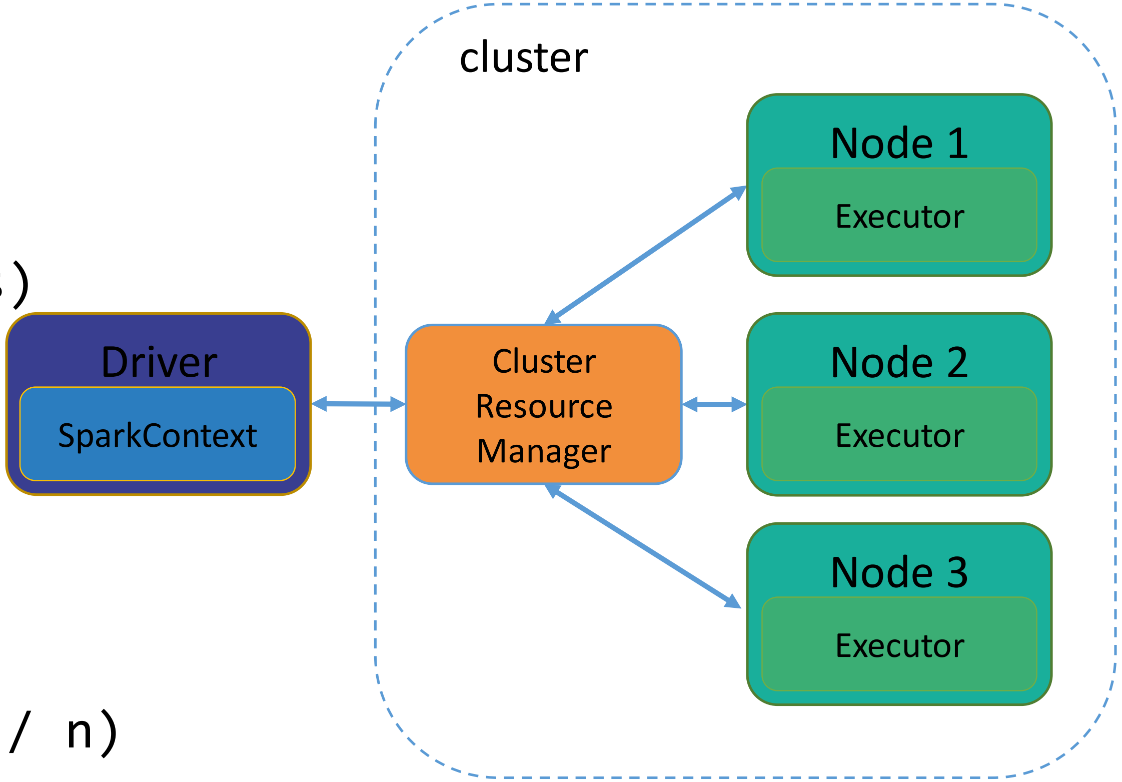
Staged Data are kept **in memory**

Transformations are lazy, they only get executed when we call an action

Apache Spark

Driver and Executors

```
import scala.math.random

val slices = 3
val n = 100000 * slices
val rdd = sc.parallelize(1 to n, slices)
val sample = rdd.map { i =>
        val x = random
        val y = random
        if (x*x + y*y < 1) 1 else 0
}
val count = sample.reduce(_ + _)
println("Pi is roughly " + 4.0 * count / n)
```



Apache Spark

Hello World – aka « Wordcount »

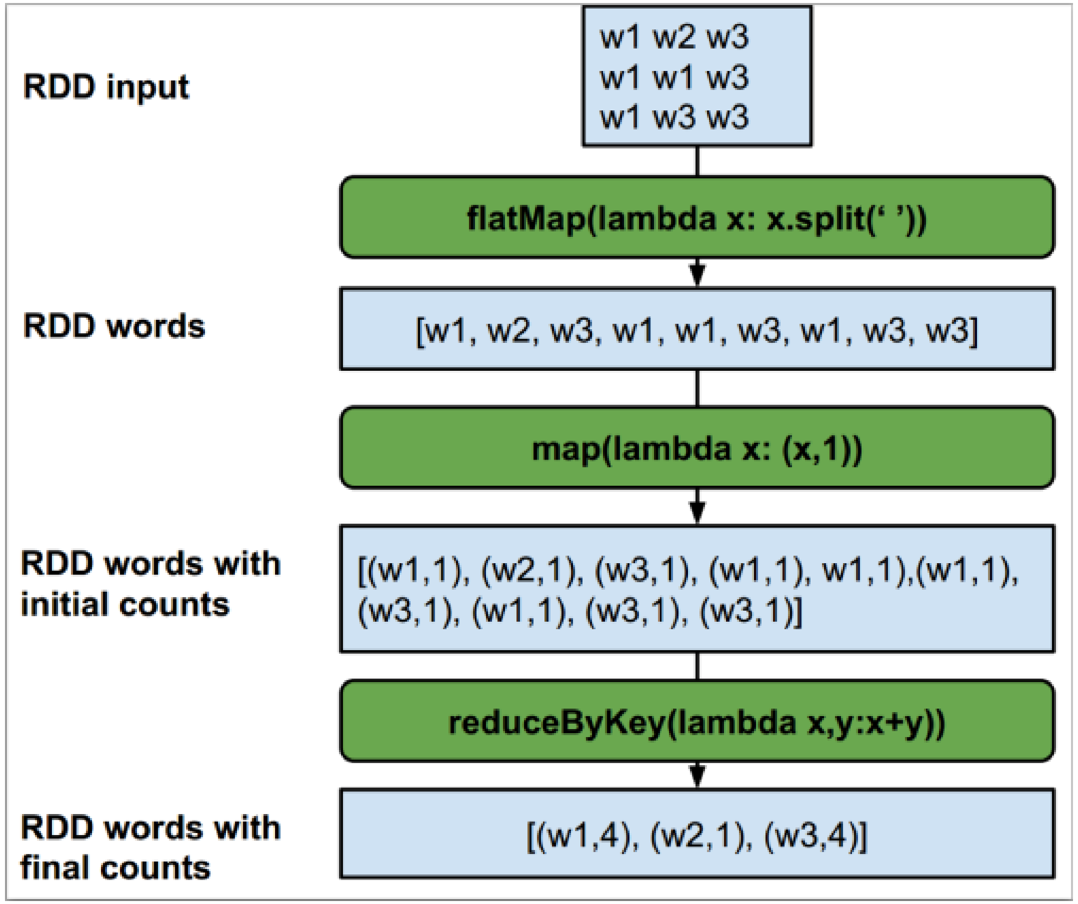
```
text_file = sc.textFile("/user/emotes/datasets/")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("/user/emotes/outputfolder/")
```

SQL on Spark

```
#defining dataframe with schema from parquet files
val df = spark.read.parquet("/user/emotes/datasets/")

#counting the number of pre-filtered rows with DF API
df.filter($"l1username".contains("emotes")).count

#counting the number of pre-filtered rows with SQL
df.registerTempTable("my_table")

spark.sql("SELECT count(*) FROM my_table where l1username like '%emotes%'").show
```

Apache Spark

Hello World – aka « Wordcount »



Standard Physics Analysis Procedures

HEP Data Processing

Physics Analysis is typically done with the ROOT Framework which uses physics data that are saved in ROOT format files.

At CERN these files are stored within the EOS Storage Service.

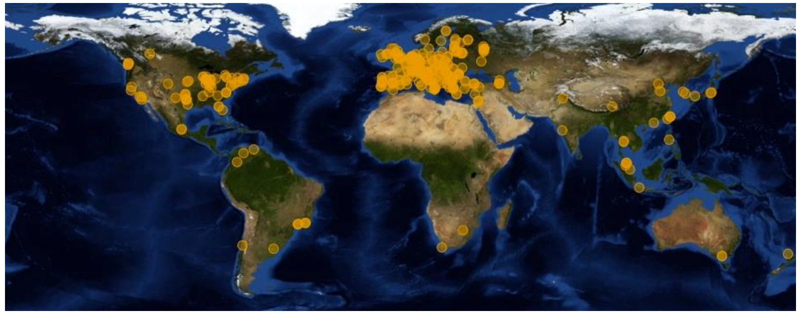
EOS Service

A disk-based, low-latency storage service with a highly-scalable hierarchical namespace, which enables data access through the XRootD protocol.

ROOT Data Analysis Framework

A modular scientific software framework which provides all the functionalities needed to deal with big data processing, statistical analysis, visualization and file storage.

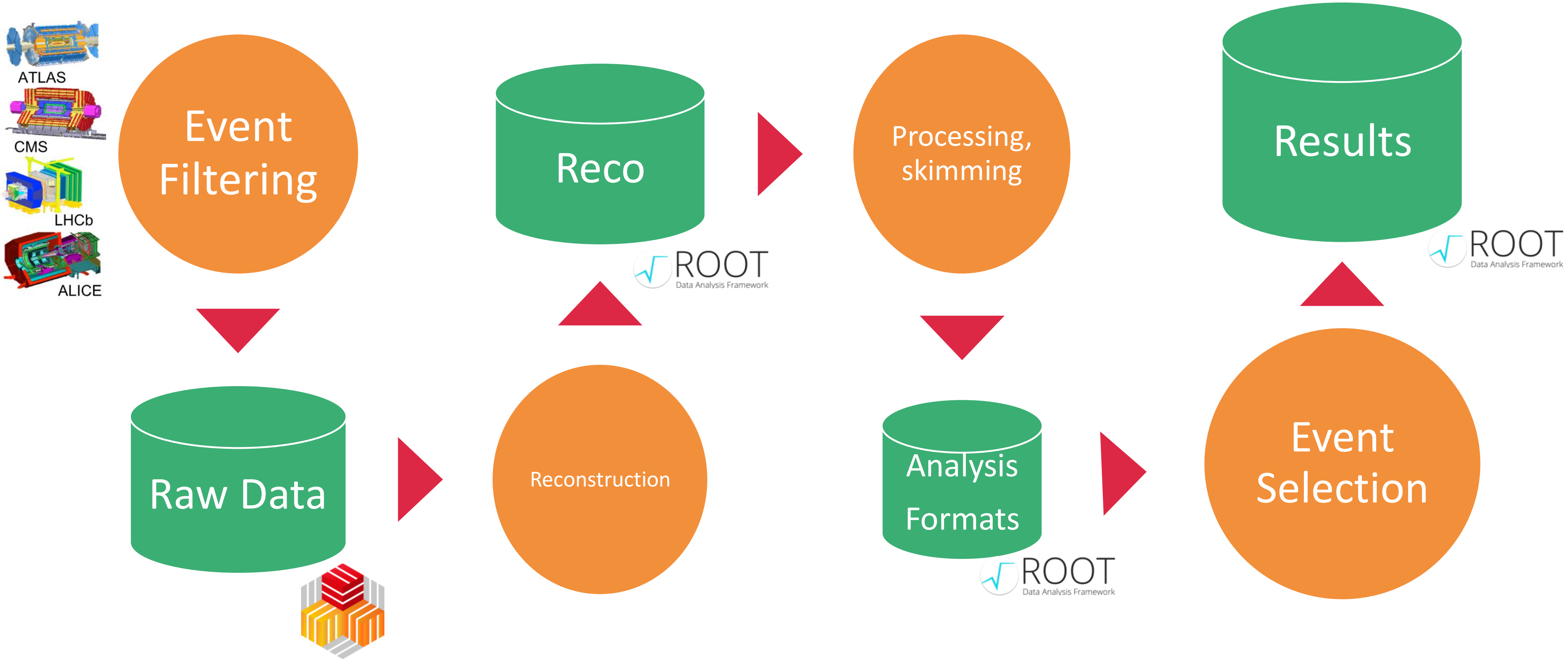
WLCG
Worldwide LHC Computing Grid



The Worldwide LHC Computing Grid (WLCG) is a global collaboration of more than 170 institutions in 42 countries which provide resources to store, distribute and analyse the PBs of LHC Data

WLCG
Worldwide LHC Computing Grid

LHC Data Flow at CERN

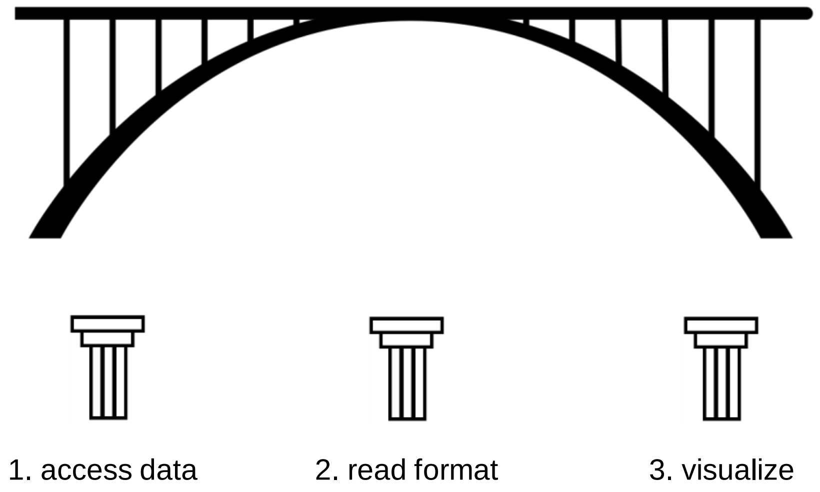


Big Data Tools
for High Energy Physics

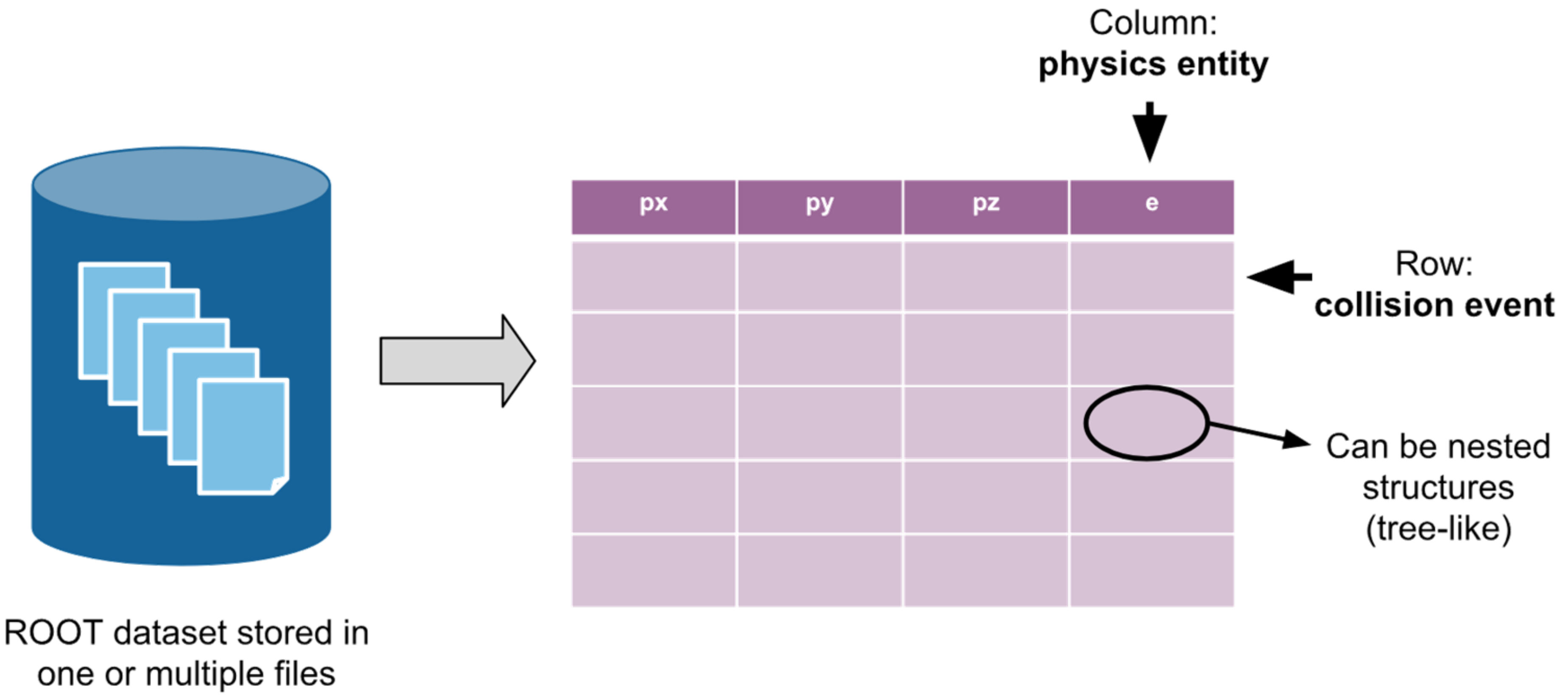
Bridging the Gap

Physics Analysis is typically done with the ROOT Framework which uses physics data that are saved in ROOT format files. At CERN these files are stored within the EOS Storage Service.

EOS Storage Service



Different Approaches for Physics Analysis with Spark



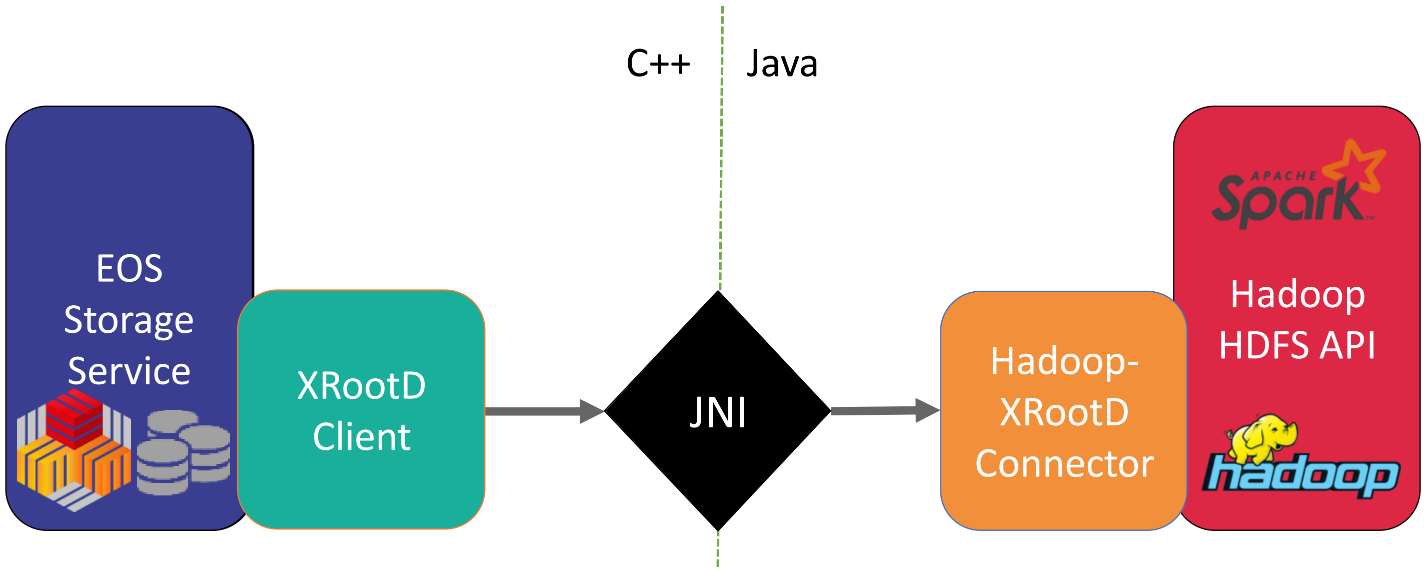
1: Apache Spark with Hadoop-XRootD Connector, spark-root, SWAN

2: Apache Spark with ROOT Rdataframe, SWAN

The 'Hadoop – XRootD Connector' Library

Connecting XRootD-based Storage Systems with Hadoop and Spark

- A Java library that connects to the XRootD client via JNI
- Reads files from the EOS Storage Service directly
- Makes all Physics Data available for processing with Spark
- Supports Kerberos and GRID Certificate Authentication



Open Source: https://github.com/cerndb/hadoop-xrootd

The 'Spark – Root' Library

A Scala library which implements DataSource for Apache Spark

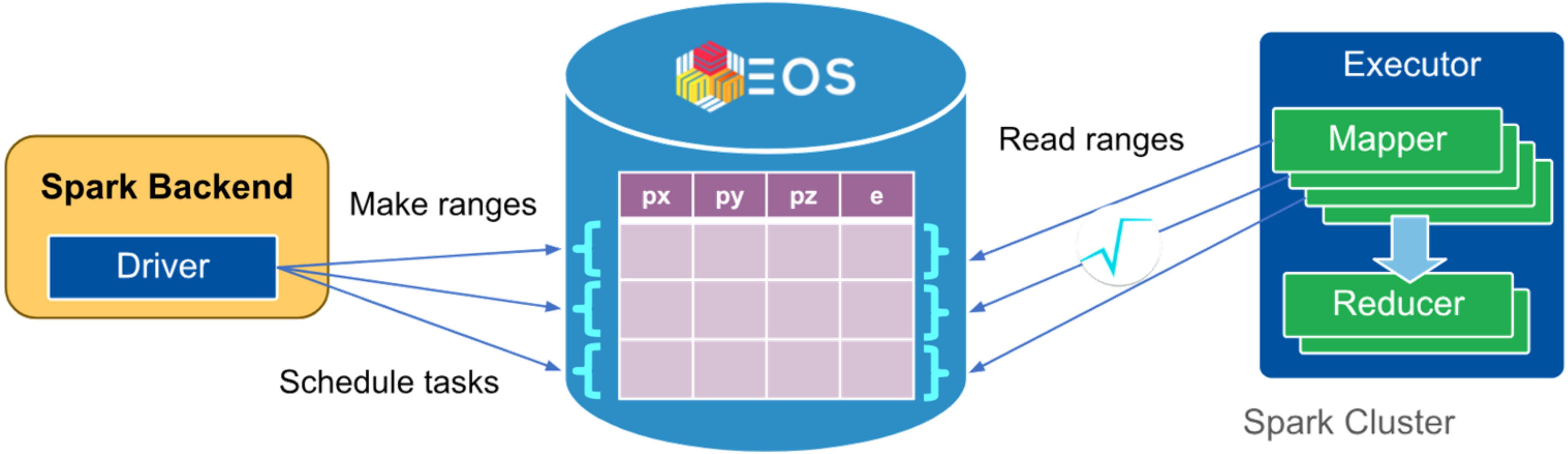
Spark can read ROOT TTrees and infer their schema

Root files are imported to Spark Dataframes/Datasets/RDDs

Developed by DIANA-HEP in collaboration with CERN openlab

Open Source: https://github.com/diana-hep/spark-root/

ROOT RDataframe



Implemented in C++ but also interfaced on Python

Exploratory work to parallelize RDataFrame computations with multiple backends

Spark Dataframes tailored for ROOT and HEP

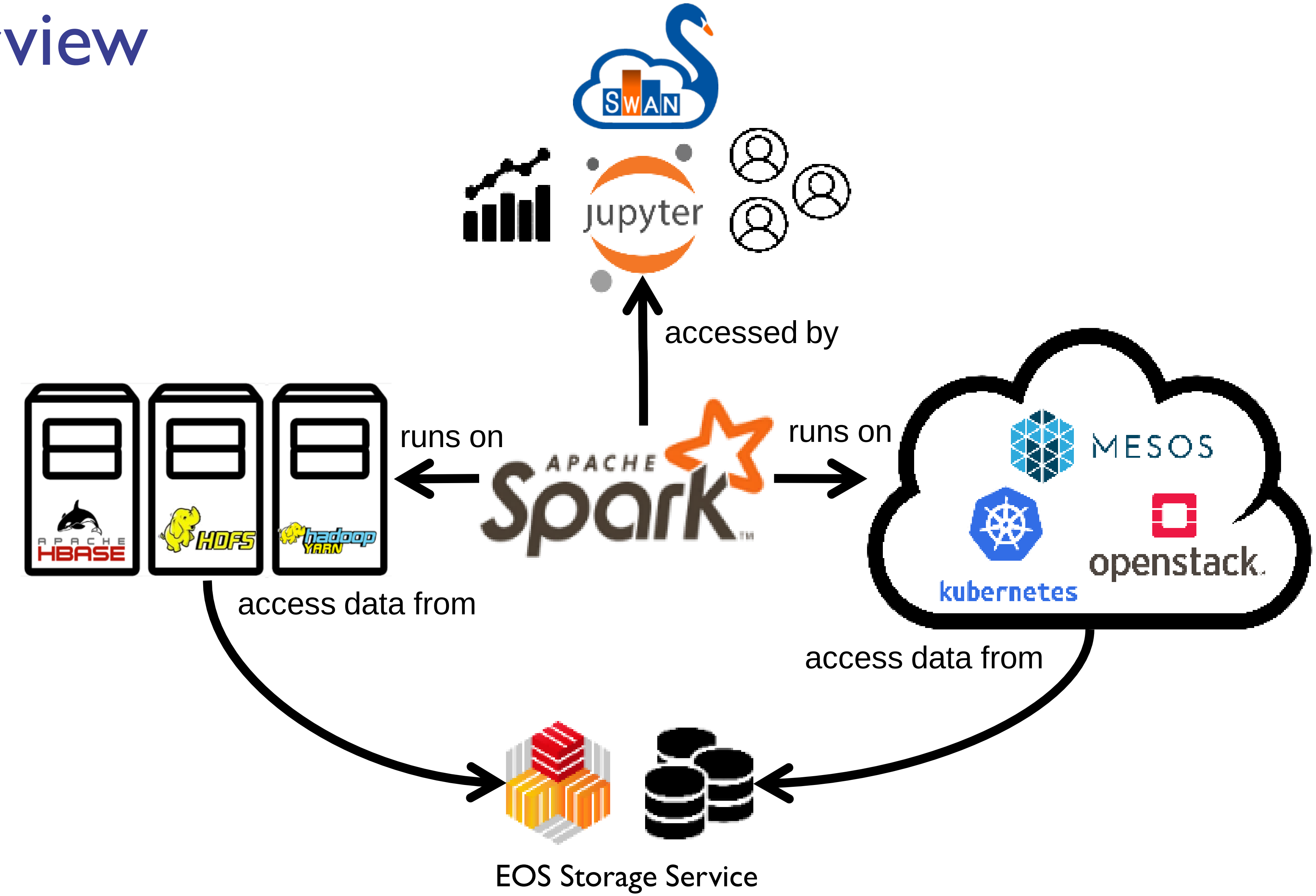
SWAN Service and Spark Integration

Hosted Jupyter Notebooks for Data Analysis

- Web-based interactive analysis using PySpark in the cloud
- No need to install software
- Cover the need for user-friendly environments that allow collaboration and sharing between researchers
- Combines code, equations, text and visualisations
- Direct access to the EOS and HDFS
- Fully Integrated with IT Spark and Hadoop Clusters

https://swan.web.cern.ch/

Overview


SWAN
jupyter
accessed by
runs on
runs on
APACHE Spark
APACHE HBASE
HDFS
hadoop YARN
MESOS
kubernetes
openstack
access data from
access data from
EOS Storage Service

Physics Analysis with Apache Spark

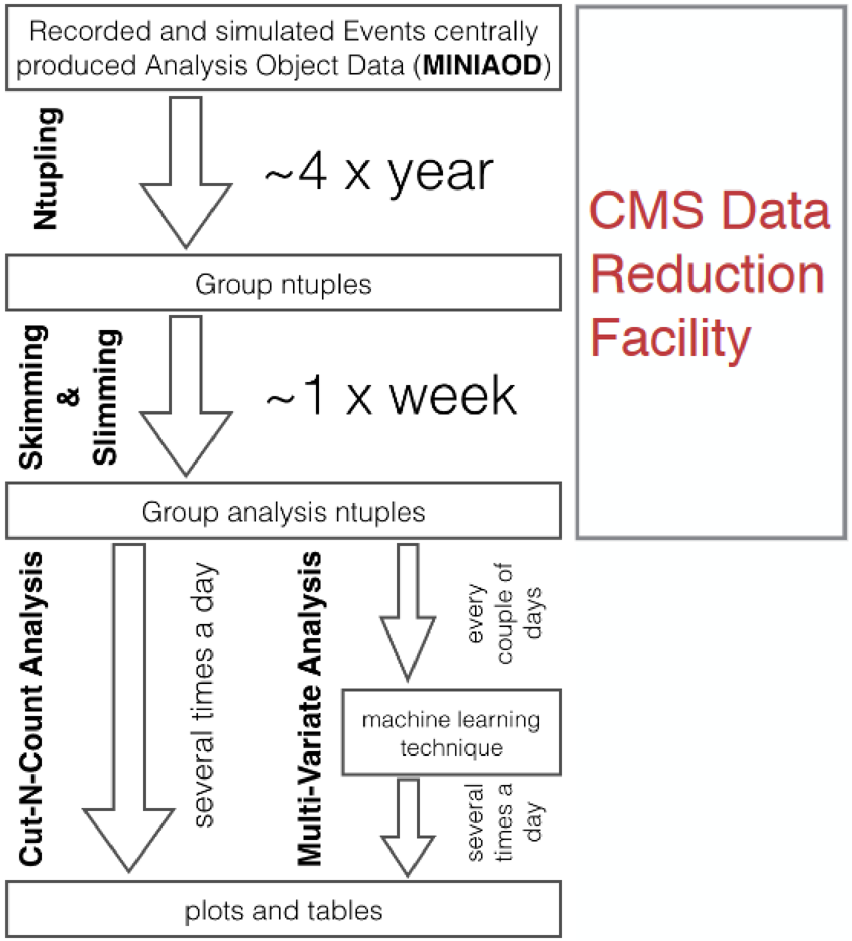
The Use Case of the CMS Data Reduction Facility

CMS Data Reduction and Analysis Facility

Performing Physics Analysis and Data Reduction with Apache Spark

- Investigate new ways to analyse physics data and improve resource utilization and time-to-physics
- Main goal was to be able to reduce 1 PB of data in 5 hours or less
- Data Reduction refers to event selection and feature preparation based on potentially complicated queries
- It offers an alternative for 'ad-hoc' data reduction for each research group
- We now have fully functioning Analysis and Reduction examples tested over CMS Open Data
- Bridge the gap between High Energy Physics and Big Data communities

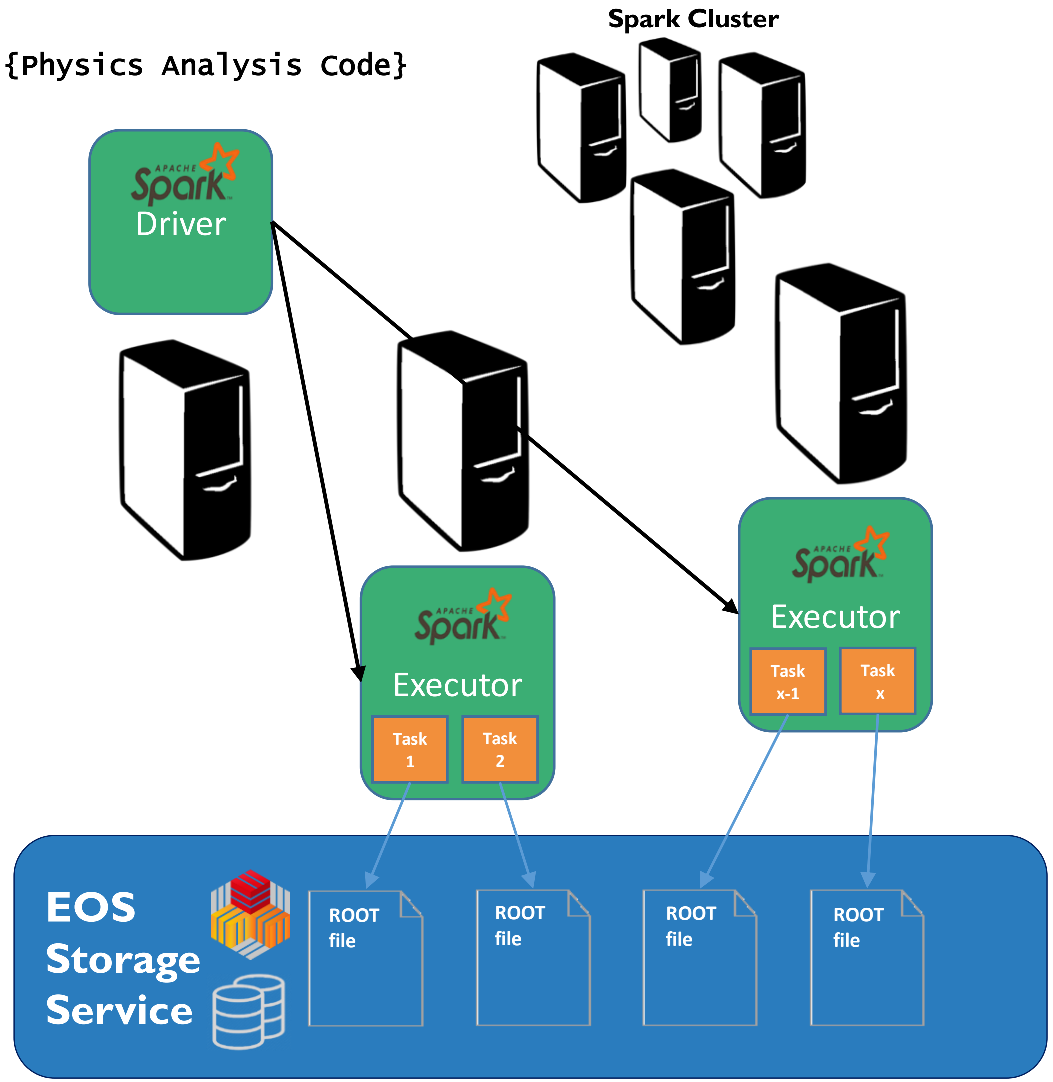
CMS Data Reduction and Analysis Facility



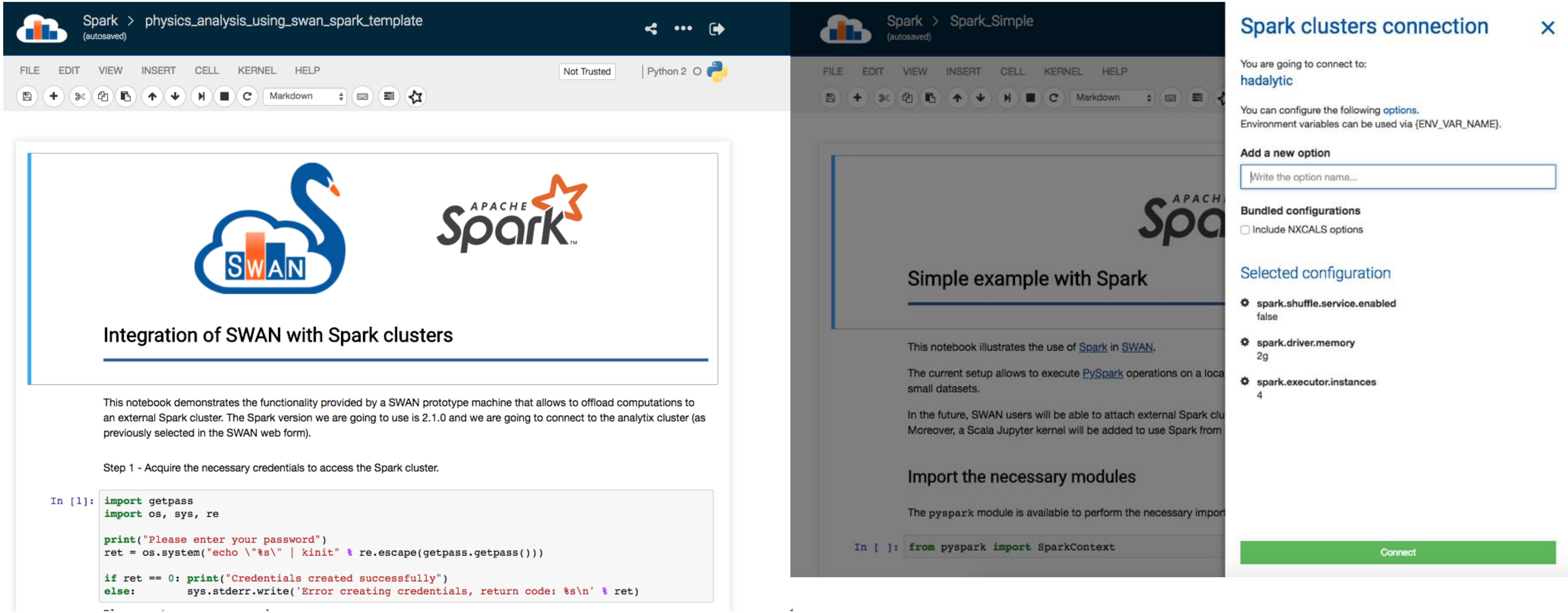
Examples

Example on Physics Analysis

Test Workload Architecture and File-Task Mapping



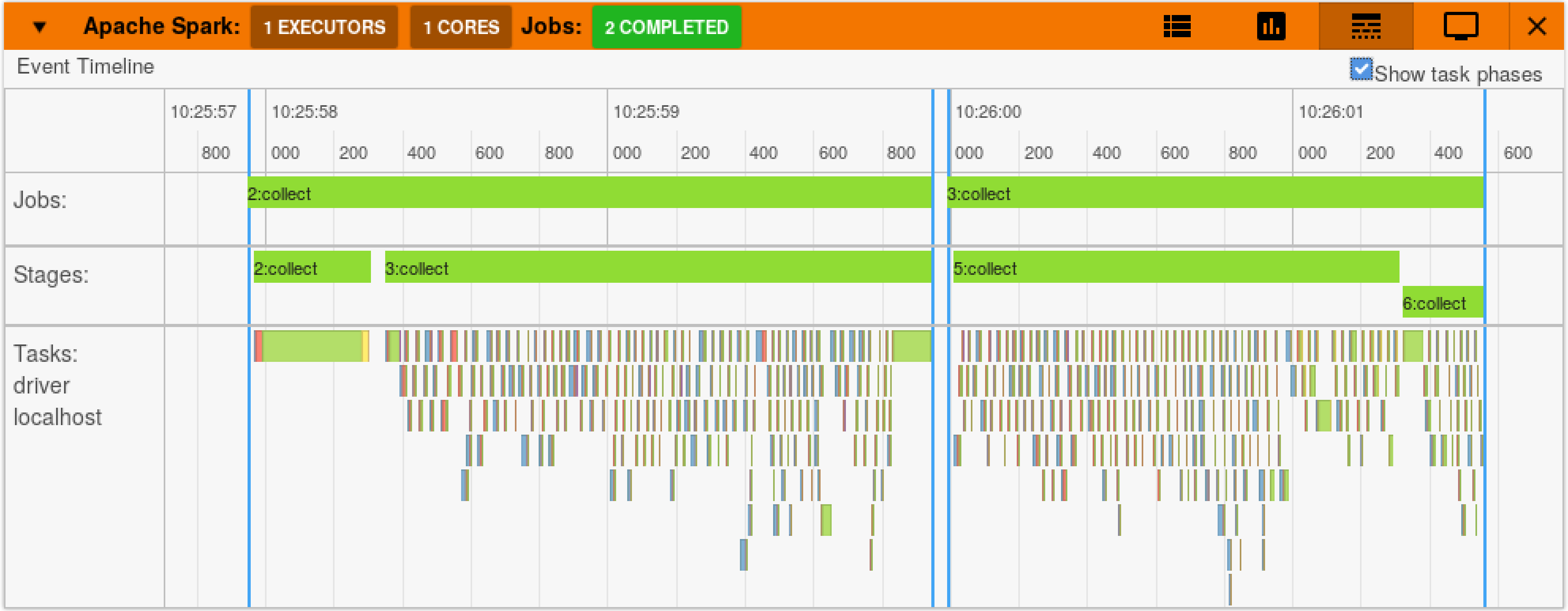
Example on Physics Analysis with SWAN



Example on Physics Analysis with SWAN

In [11]:

```
val h = df.filter(_.muons.length >= 2).flatMap({e: Event => for (i <- 0 until e.muons.length; j <- 0 until
e.muons.length) yield buildDiCandidate(e.muons(i), e.muons(j))}).rdd.aggregate(emptyDiCandidate)(new Increment, new
Combine);
```



Example on Physics Analysis with SWAN

Apache Spark: 1 EXECUTORS 4 CORES Jobs: 2 COMPLETED

Job ID	Job Name	Status	Stages	Tasks	Submission Time	Duration
2	reduce	COMPLETED	2/2	48 / 48	5 minutes ago	3s

Stage Id	Stage Name	Status	Tasks	Submission Time	Duration
5	reduce	COMPLETED	32 / 32	5 minutes ago	2s
4	coalesce	COMPLETED	16 / 16	5 minutes ago	0s

Job ID	Job Name	Status	Stages	Tasks	Submission Time	Duration
3	foreach	COMPLETED	1/1 (1 skipped)	32 / 32	5 minutes ago	1m:20s

Stage Id	Stage Name	Status	Tasks	Submission Time	Duration
6	coalesce	SKIPPED	0 / 16	Unknown	-
7	foreach	COMPLETED	32 / 32	5 minutes ago	1m:20s

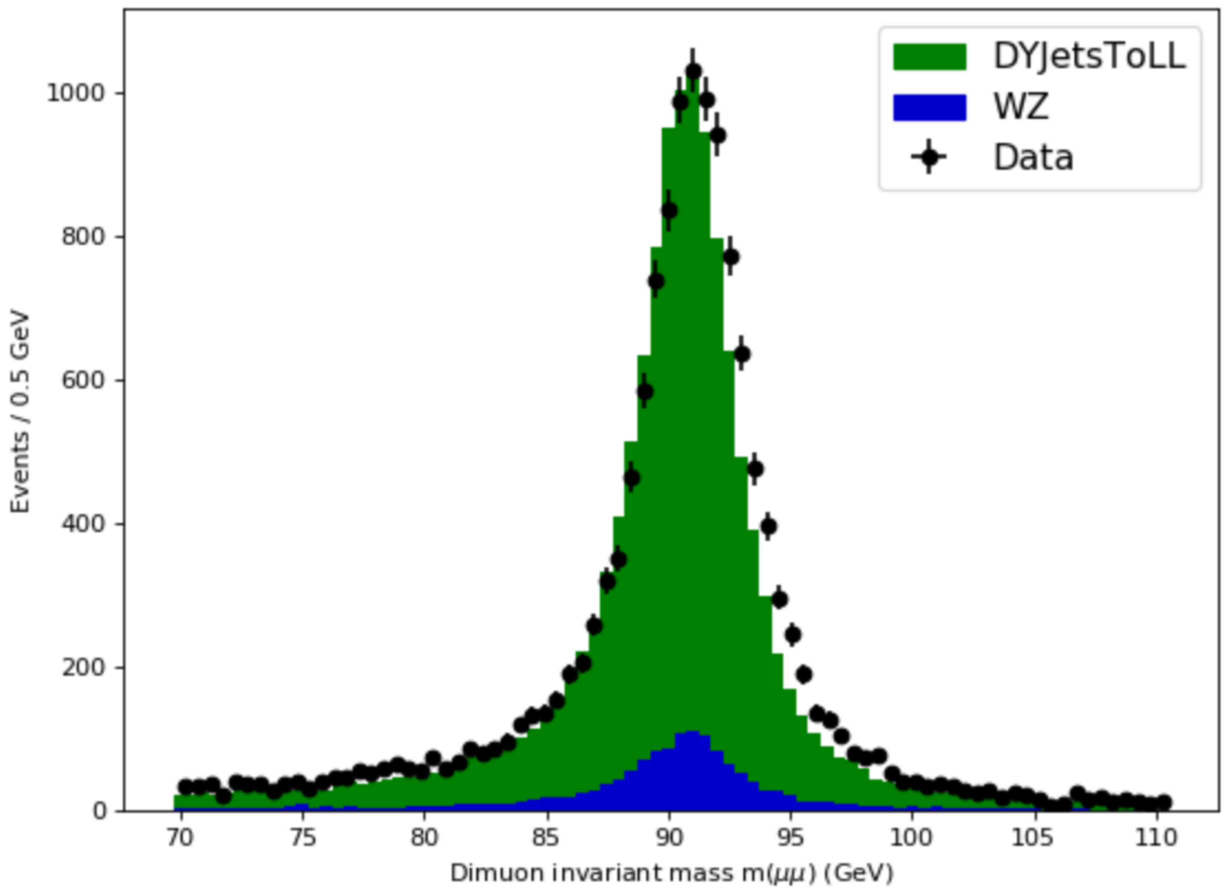
Final Result

```
scala> val empty = Bin(40, 0, 100, {x: Float => x})
empty: org.dianahep.histogrammar.Binning[Float,org.dianahep.histogrammar.Counting,org.dianahep.histogrammar.Counting,org.dianahep.histogrammar.Counting,org.dianahep.histogrammar.Counting] = <Binni
ng num=40 low=0.0 high=100.0 values=Count underflow=Count overflow=Count nanflow=Count>

scala> val histo = muons.as[Seq[Float]].flatMap({case x => x}).rdd.aggregate(empty)(new Increment, new Combine)
recoMuons_muons__RECO_
[Lorg.apache.spark.sql.sources.Filter;@798f8f25
17/09/20 15:06:45 WARN ClosureCleaner: Expected a closure; got org.dianahep.histogrammar.Increment
17/09/20 15:06:45 WARN ClosureCleaner: Expected a closure; got org.dianahep.histogrammar.Combine
histo: org.dianahep.histogrammar.Binning[Float,org.dianahep.histogrammar.Counting,org.dianahep.histogrammar.Counting,org.dianahep.histogrammar.Counting,org.dianahep.histogrammar.Counting] = <Binni
ng num=40 low=0.0 high=100.0 values=Count underflow=Count overflow=Count nanflow=Count>

scala> histo.println
                      0                                                         8869.30
                      +-----------------------------------------------------------+
underflow       0     |                                                           |
[  0   ,  2.5 ) 8063  |*******************************************************    |
[  2.5 ,  5   ) 5173  |***********************************                        |
[  5   ,  7.5 ) 1629  |***********                                                |
[  7.5 ,  10  ) 379   |***                                                        |
[  10  ,  12.5) 130   |*                                                          |
[  12.5,  15  ) 60    |                                                           |
[  15  ,  17.5) 26    |                                                           |
[  17.5,  20  ) 19    |                                                           |
[  20  ,  22.5) 8     |                                                           |
[  22.5,  25  ) 5     |                                                           |
[  25  ,  27.5) 4     |                                                           |
[  27.5,  30  ) 7     |                                                           |
[  30  ,  32.5) 2     |                                                           |
[  32.5,  35  ) 3     |                                                           |
[  35  ,  37.5) 2     |                                                           |
[  37.5,  40  ) 1     |                                                           |
[  40  ,  42.5) 1     |                                                           |
[  42.5,  45  ) 3     |                                                           |
[  45  ,  47.5) 1     |                                                           |
[  47.5,  50  ) 0     |                                                           |
[  50  ,  52.5) 0     |                                                           |
[  52.5,  55  ) 1     |                                                           |
[  55  ,  57.5) 2     |                                                           |
[  57.5,  60  ) 0     |                                                           |
[  60  ,  62.5) 0     |                                                           |
[  62.5,  65  ) 0     |                                                           |
[  65  ,  67.5) 0     |                                                           |
[  67.5,  70  ) 0     |                                                           |
[  70  ,  72.5) 1     |                                                           |
[  72.5,  75  ) 0     |                                                           |
[  75  ,  77.5) 0     |                                                           |
[  77.5,  80  ) 0     |                                                           |
[  80  ,  82.5) 0     |                                                           |
[  82.5,  85  ) 0     |                                                           |
[  85  ,  87.5) 0     |                                                           |
[  87.5,  90  ) 1     |                                                           |
[  90  ,  92.5) 1     |                                                           |
[  92.5,  95  ) 0     |                                                           |
[  95  ,  97.5) 0     |                                                           |
[  97.5,  100 ) 0     |                                                           |
overflow        7     |                                                           |
nanflow         0     |                                                           |
                      +-----------------------------------------------------------+

scala>
```

OK, OK, one with better graphics

Final Result

Out[16]: <matplotlib.legend.Legend at 0x7fd0a7f308d0>



That is what we will do in the exercises ☺

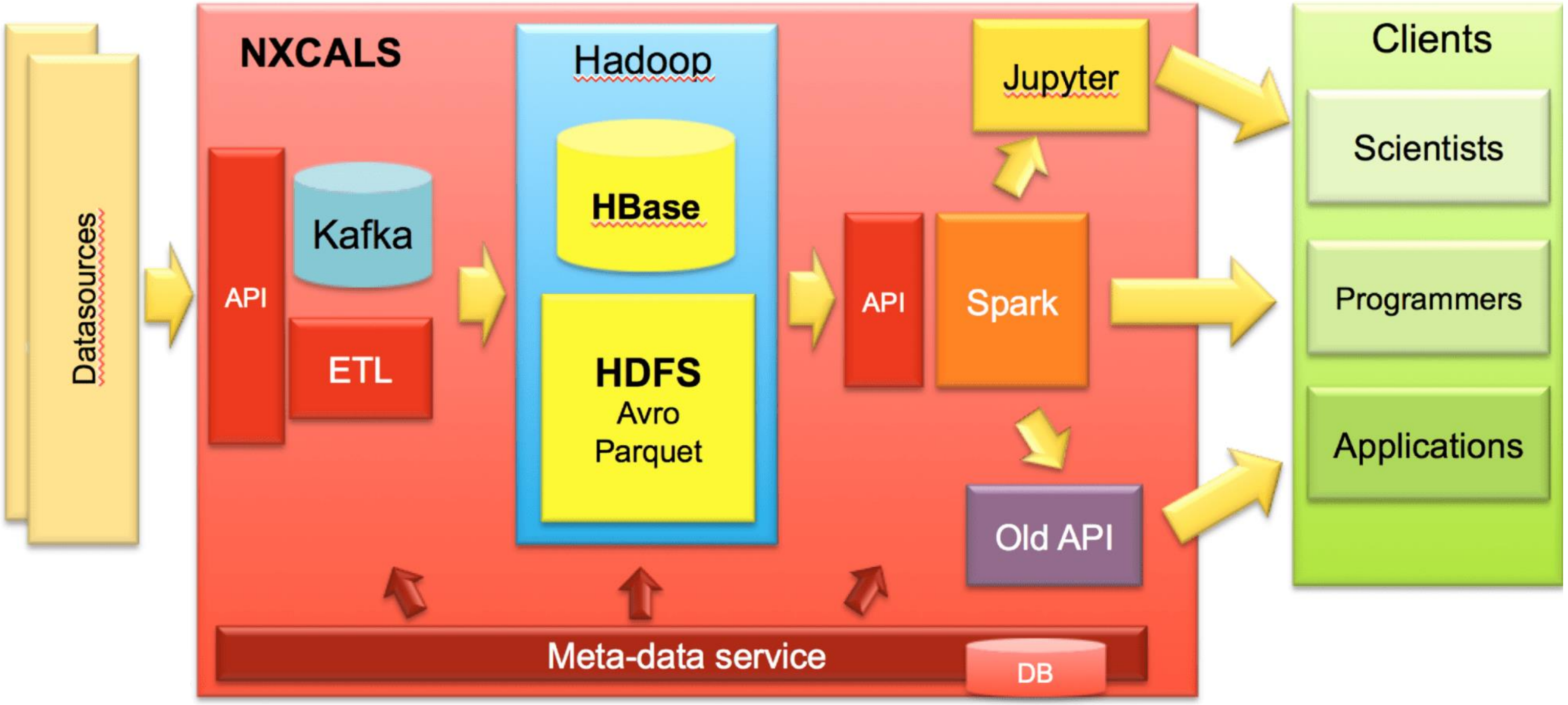
Projects beyond Physics Analysis

Next Accelerator Logging Service (NXCALS)

A control system with:
- Streaming
- Online System
- API for Data Extraction

Critical for LHC Operations

Runs on a dedicated cluster

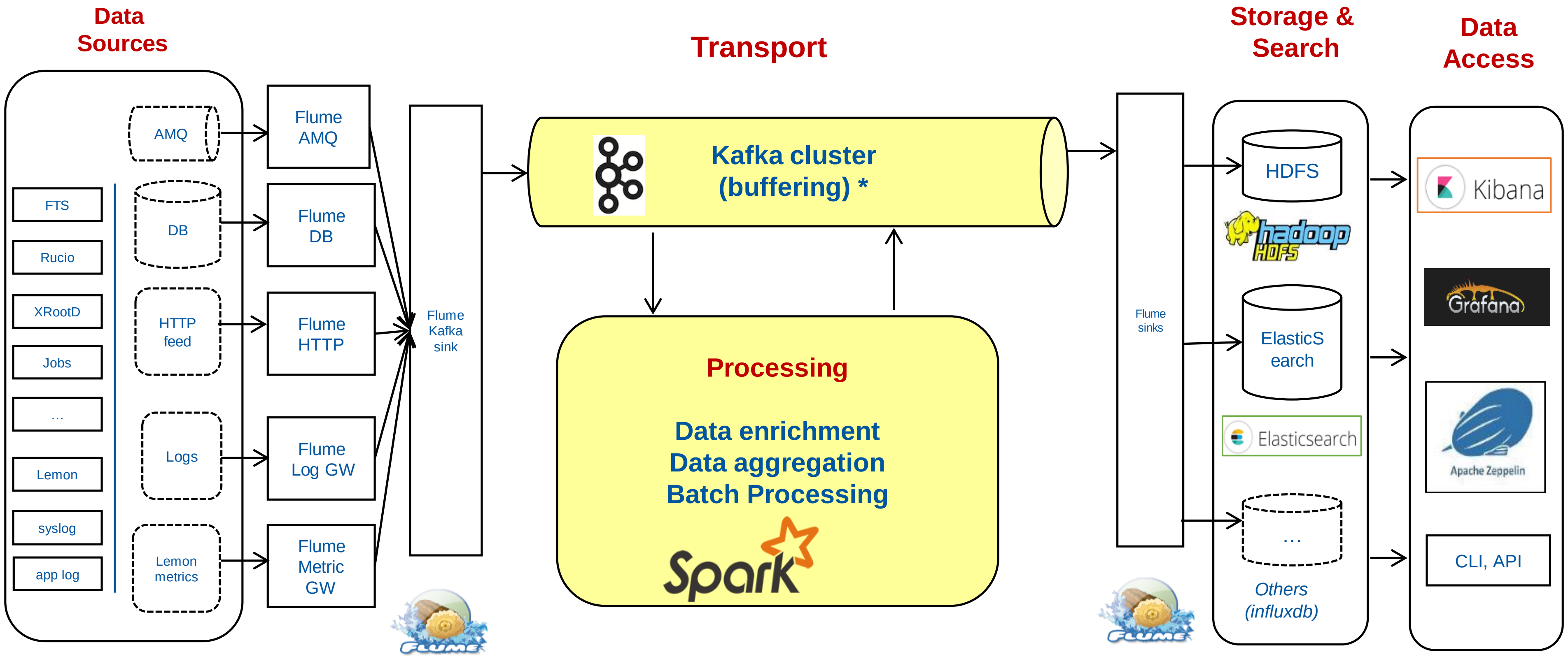


Credits: BE-CO-DS

Data Center and WLCG Monitoring Systems

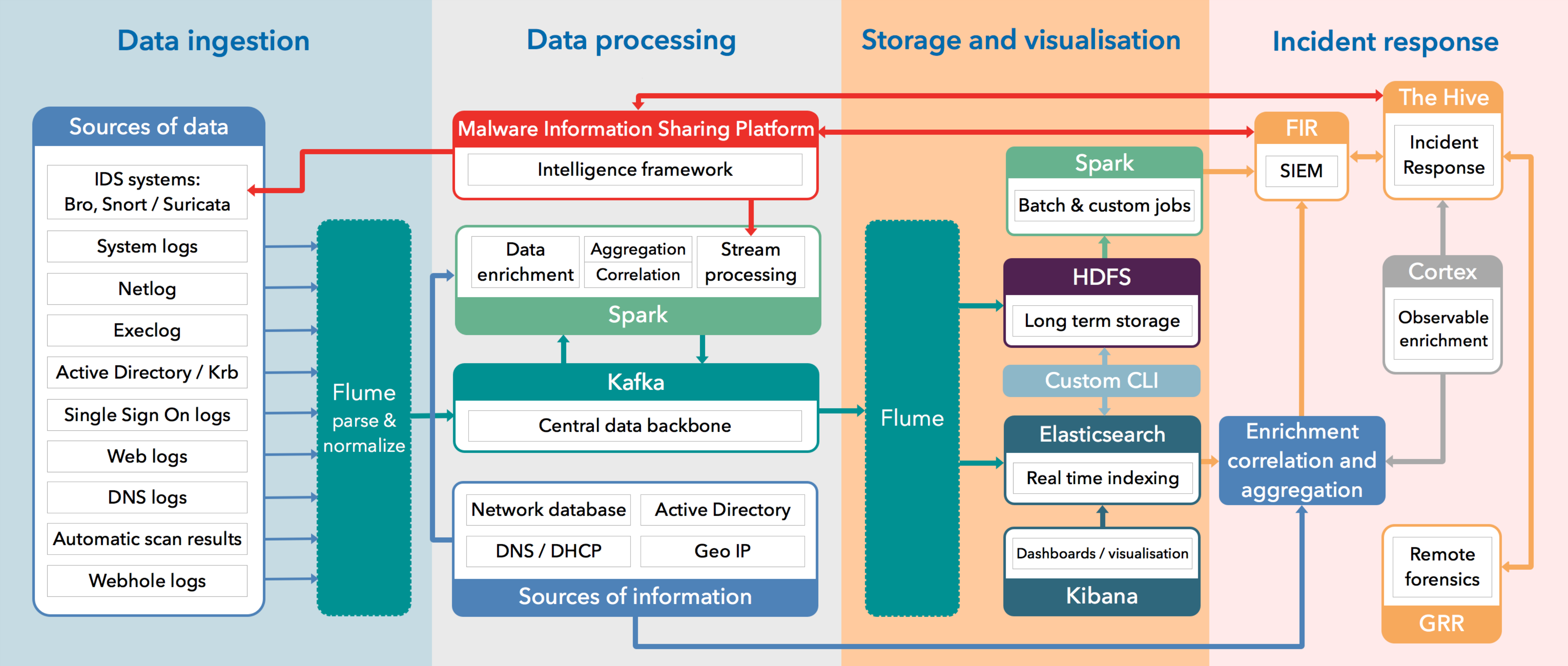
Critical for Data Center operations and WLCG

200M events/day
500 GB/day



Credits: IT-CM-MM

Computer Security Intrusion Detection



Credits: CERN security team, IT-DI

Conclusions

Conclusions

- There is a broad ecosystem of Big Data Frameworks, most of which share the same architecture principles such as resource pooling, high availability, fault tolerance, etc.

- Popular Big Data Frameworks such as Apache Spark show great potential in bridging the gap between the High Energy Physics community and the Big Data community.

- There are now available tools and services to use these big data technologies in order to perform analytics on physics, infrastructure, and accelerator data.

Acknowledgements

My mentors, Sebastian Lopienski and Enric Tejedor Saavedra

Colleagues at the CERN Hadoop, Spark, and streaming services who kindly helped with material and feedback

The CSC Team, especially Joelma and Nikos

CMS members of the Big Data Reduction Facility, DIANA/HEP, Fermilab

Thank you

emotes@cern.ch