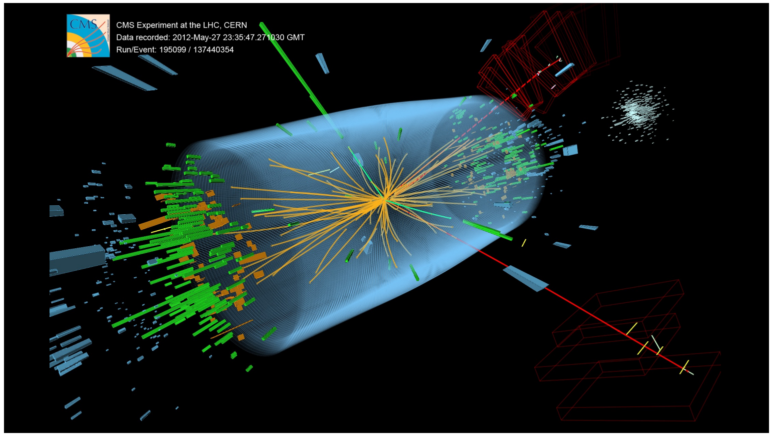
“Big Data In HEP” - Physics Data Analysis, Data Reduction and Machine Learning

Nicola De Filippis (Politecnico and INFN Bari)

LIP, September 7, 2022

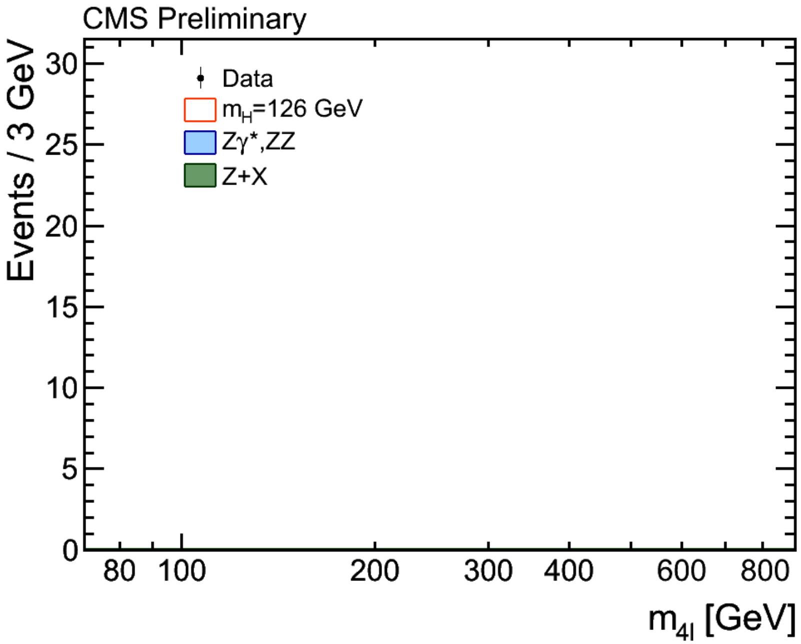
Experimental Particle Physics - the Journey

Particle Collisions

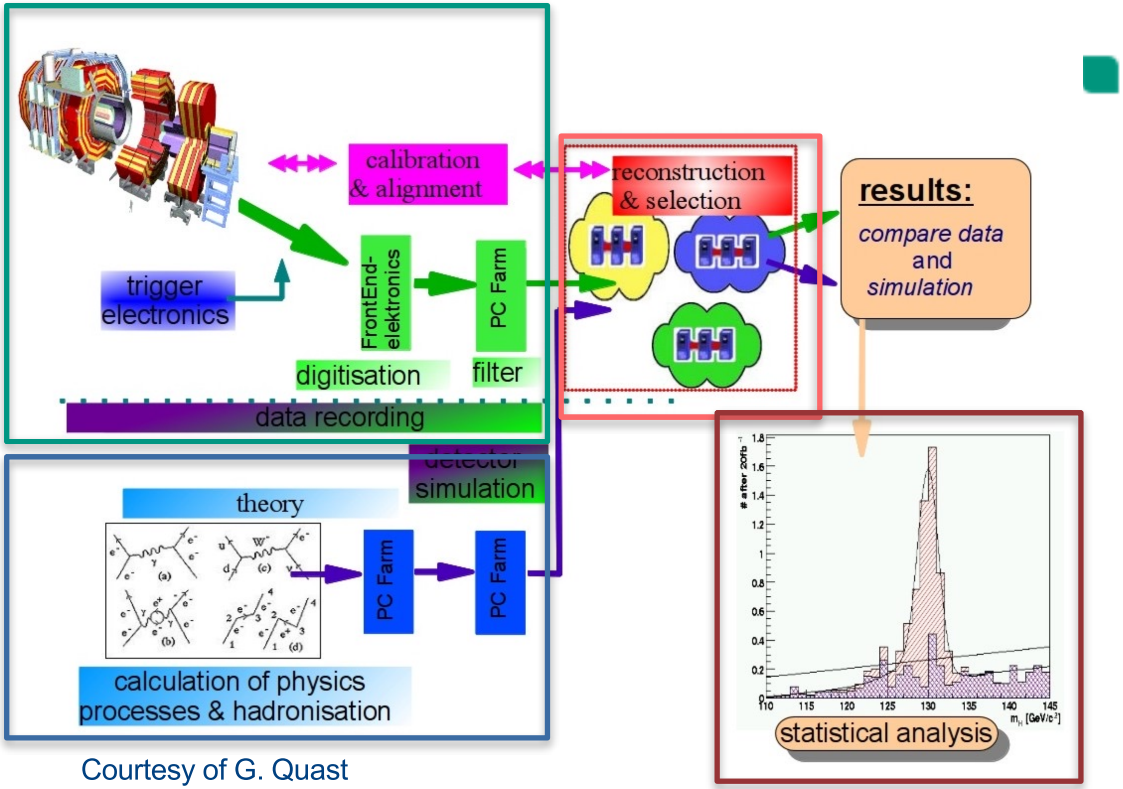

CMS Experiment at the LHC, CERN
Data recorded: 2012-May-27 23:35:47.271030 GMT
Run/Event: 195099 / 137440354

Large Scale Computing

Physics Discoveries



Detector & Analysis Chain



calibration & alignment

reconstruction & selection

results:

compare data and simulation

trigger electronics

FrontEnd-elektronics

PC Farm

digitisation

filter

data recording

detector simulation

theory

PC Farm

PC Farm

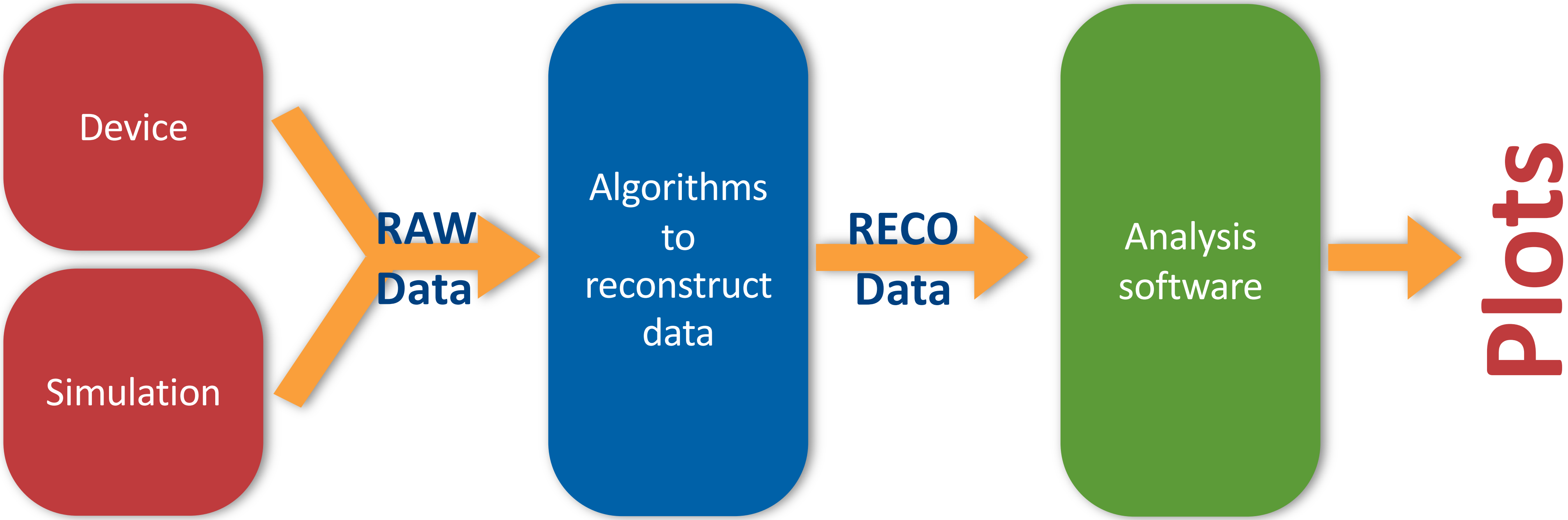
calculation of physics processes & hadronisation

statistical analysis

Courtesy of G. Quast

- Collider-based particle physics: complicated analysis chain
  - **Detector** input: calibrated digitized data, online selection (“trigger”)
  - **Theory** input: calculations and simulations
  - Physics object **reconstruction** and **selection**
  - **Statistical** analysis & comparison with theory

Large Scale Computing



Device

Simulation

RAW Data

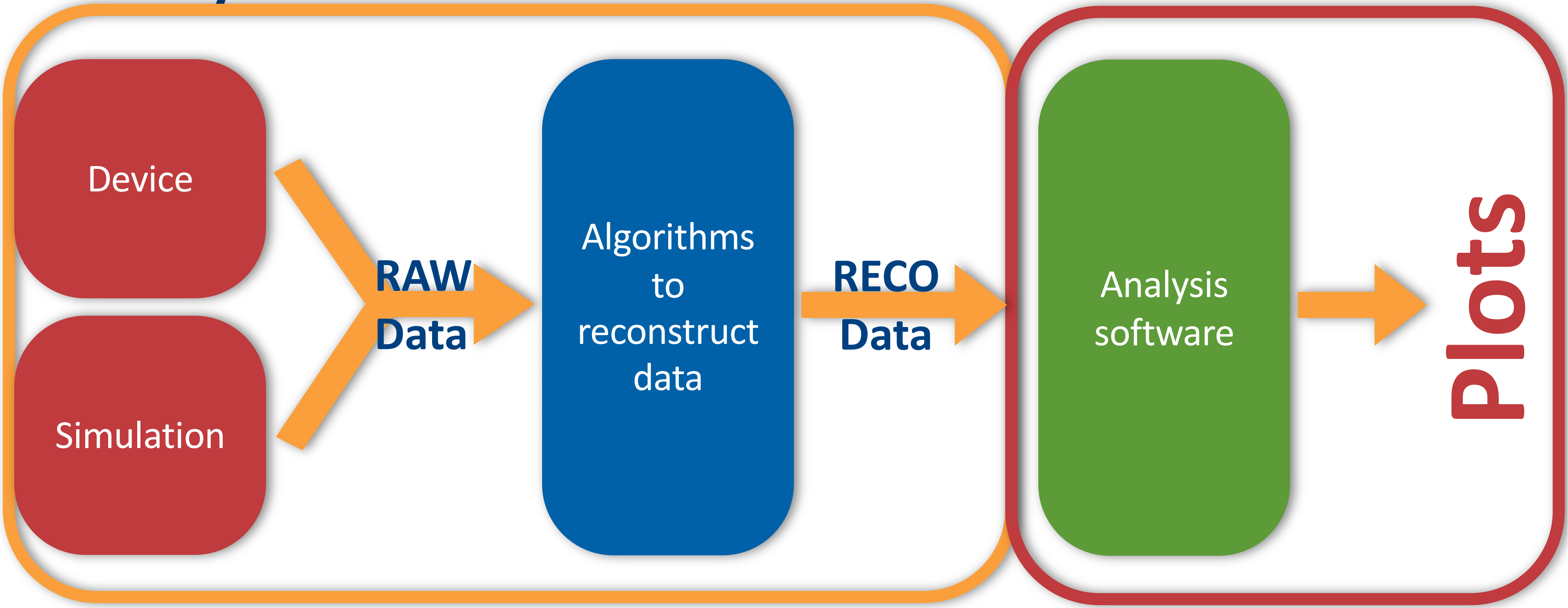
Algorithms to reconstruct data

RECO Data

Analysis software

Plots

Analysis in CMS



Device

Simulation

RAW Data

Algorithms to reconstruct data

RECO Data

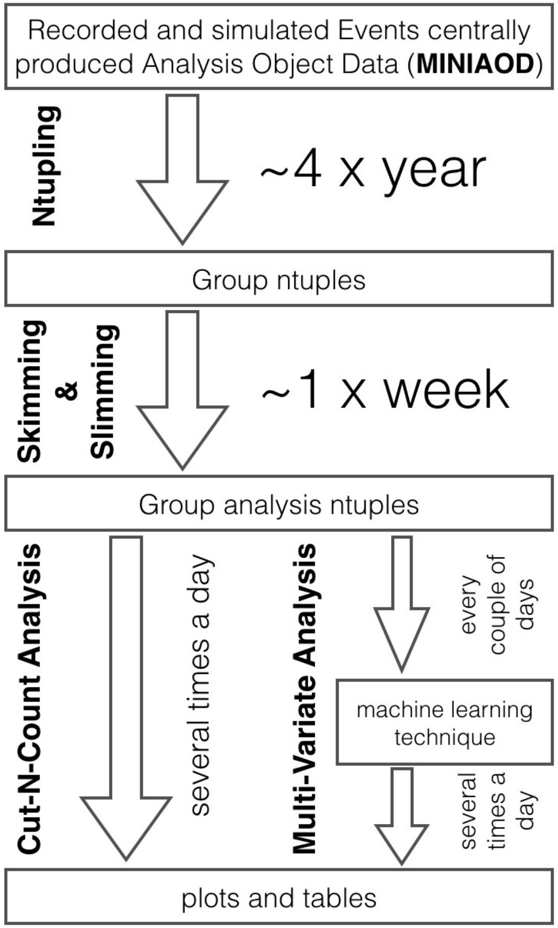
Analysis software

Plots

Central

Hundreds of physicists analyze the data with different goals at the same time

Analysis: A multi-step Process

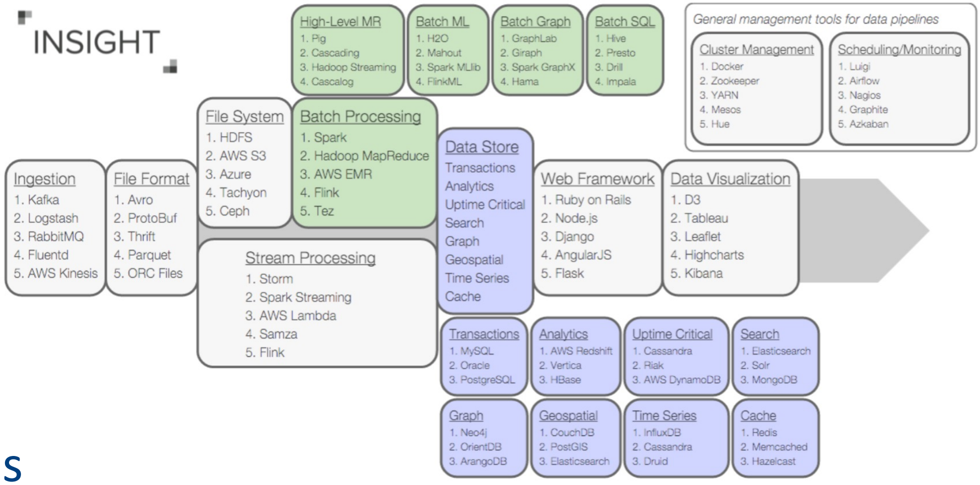


- Minimize **Time to Insight**
  - Analysis is a conversation with data - Interactivity is key
- Many different physics topics concurrently under investigation
  - Different slices of data are relevant for each analysis
- Programmatically same analysis steps
  - Skimming (dropping events in a disk-to-disk copy)
  - Slimming (dropping branches in a disk-to-disk copy)
  - Filtering (selectively reading events into memory)
  - Pruning (selectively reading branches into memory)

Big Data

- New toolkits and systems collectively called “Big Data” technologies have emerged to support the analysis of PB and EB datasets in industry.

- Our goals in applying these technologies to the HEP analysis challenge:
  - Reduce Time to Insight
  - Educate our graduate students and post docs to use industry-based technologies
    - Improves chances on the job market outside academia
    - Increases the attractiveness of our field
  - Be part of an even larger community



INSIGHT

Ingestion
1. Kafka
2. Logstash
3. RabbitMQ
4. Fluentd
5. AWS Kinesis

File Format
1. Avro
2. ProtoBuf
3. Thrift
4. Parquet
5. ORC Files

File System
1. HDFS
2. AWS S3
3. Azure
4. Tachyon
5. Ceph

Stream Processing
1. Storm
2. Spark Streaming
3. AWS Lambda
4. Samza
5. Flink

Batch Processing
1. Spark
2. Hadoop MapReduce
3. AWS EMR
4. Flink
5. Tez

High-Level MR
1. Pig
2. Cascading
3. Hadoop Streaming
4. Cascalog

Batch ML
1. H2O
2. Mahout
3. Spark MLlib
4. FlinkML

Batch Graph
1. GraphLab
2. Giraph
3. Spark GraphX
4. Hama

Batch SQL
1. Hive
2. Presto
3. Drill
4. Impala

Data Store
Transactions
Analytics
Uptime Critical
Search
Graph
Geospatial
Time Series
Cache

Web Framework
1. Ruby on Rails
2. Node.js
3. Django
4. AngularJS
5. Flask

Data Visualization
1. D3
2. Tableau
3. Leaflet
4. Highcharts
5. Kibana

General management tools for data pipelines

Cluster Management
1. Docker
2. Zookeeper
3. YARN
4. Mesos
5. Hue

Scheduling/Monitoring
1. Luigi
2. Airflow
3. Nagios
4. Graphite
5. Azkaban

Transactions
1. MySQL
2. Oracle
3. PostgreSQL

Analytics
1. AWS Redshift
2. Vertica
3. HBase

Uptime Critical
1. Cassandra
2. Riak
3. AWS DynamoDB

Search
1. Elasticsearch
2. Solr
3. MongoDB

Graph
1. Neo4j
2. OrientDB
3. ArangoDB

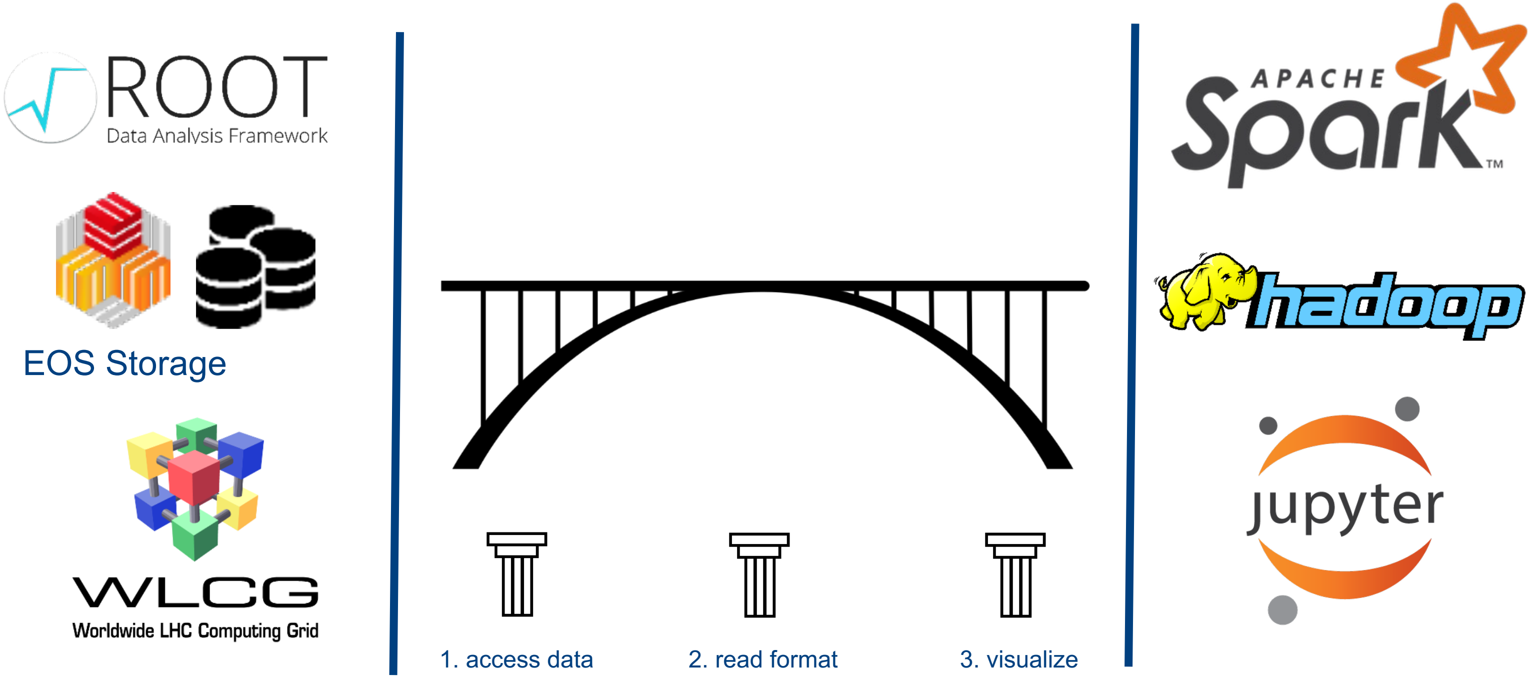
Geospatial
1. CouchDB
2. PostGIS
3. Elasticsearch

Time Series
1. InfluxDB
2. Cassandra
3. Druid

Cache
1. Redis
2. Memcached
3. Hazelcast

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.



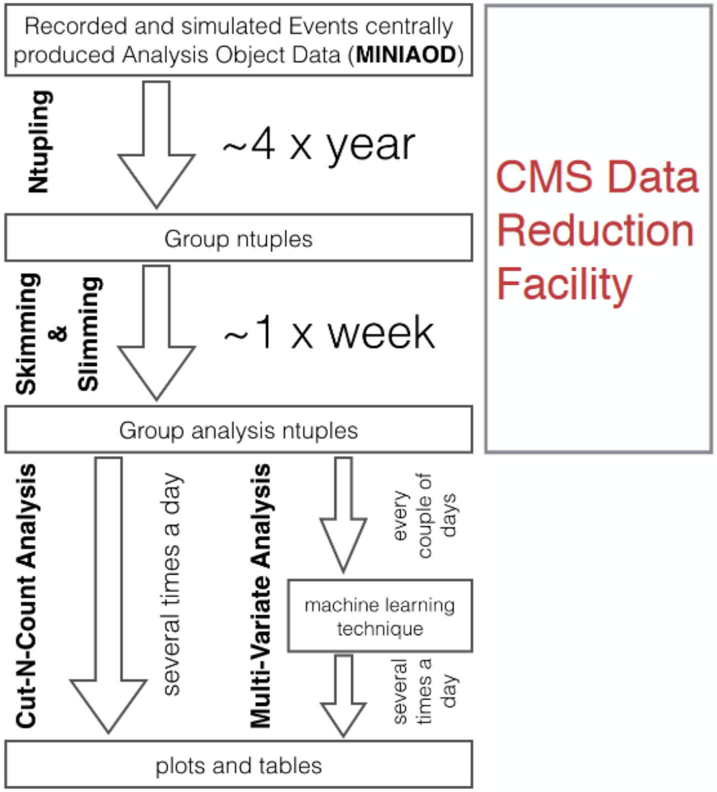
EOS Storage

1. access data

2. read format

3. visualize

CMS Data Reduction and Analysis Facility



- CERN openlab / Intel project

- Apache Spark is a unified analytics engine for large-scale data processing with built-in modules for SQL, streaming, machine learning, and graph processing. Spark can run on Apache Hadoop, Apache Mesos, Kubernetes, on its own, in the cloud and for diverse data sources.

- Demonstrate reduction capabilities producing analysis ntuples using Apache Spark

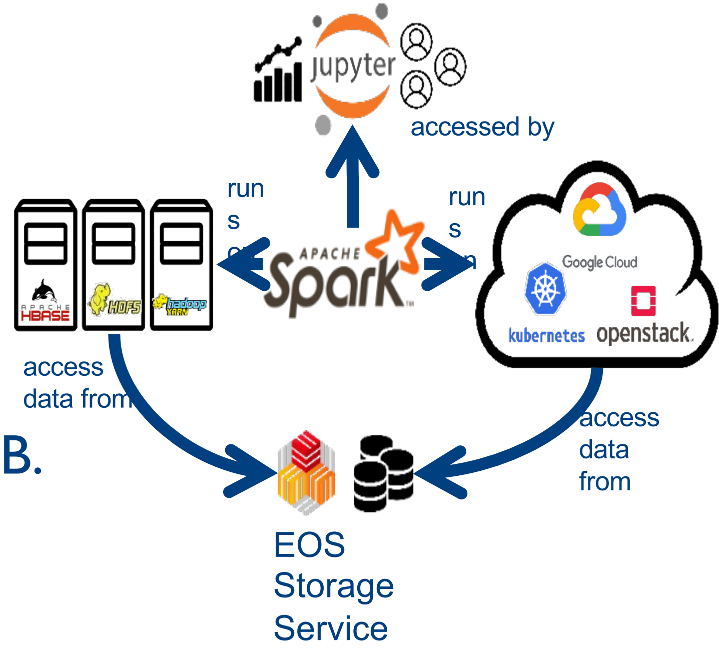
Demonstrator’s goal: data reduction of 1 PB input data in 5 hours

Milestones and Achievements

- Two important data engineering challenges were solved:

1. Read files in ROOT Format using Spark

2. Access files stored in EOS directly from Hadoop/Spark


accessed by

run
s
o

run
s
o

access
data from

access
data
from

EOS
Storage
Service

- This enabled us to produce, scale up, and optimize Physics Analysis Workloads with data input up to 1 PB.

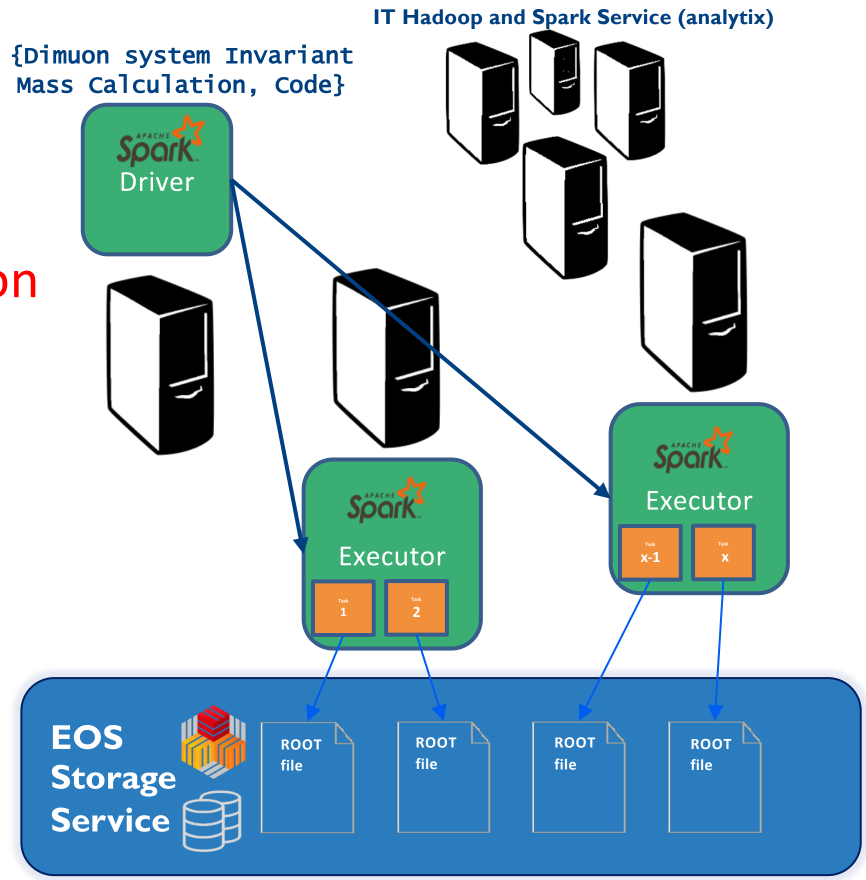
Scalability Tests

The data processing job of this project was developed in Scala by CMS members.

- Performs event selection (i.e. Data Reduction)
- Uses the filtered events to compute the dimuon invariant mass

- On a single thread/core and one single file as input, the workload reads one branch and calculates the dimuon invariant mass in approximately 10 mins for a 4GB file

Test Workload Architecture and File-Task Mapping



Scalability Tests: Technology

Apache Spark used:

- Hadoop YARN cluster
- Kubernetes and Openstack (cloud resources)

Services/Tools Used:

- EOS Public, CERN open data
- Hadoop-XRootD Connector (allows Spark to access the CERN EOS storage system)
- spark-root (Spark data source for ROOT format)
- sparkMeasure (spark instrumentation)
- Spark on Kubernetes Service

Issues tackled:

- Network bottleneck at scale: “readAhead” buffer size configuration of the Hadoop-XRtooD connector
- Running tests on a shared clusters and share infrastructure in IT datacenter

Hadoop and Spark Clusters at CERN

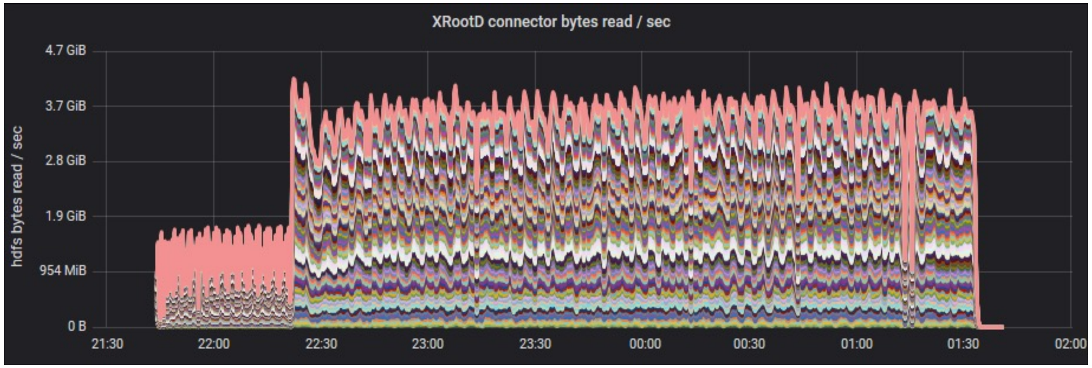
- Clusters:
  - YARN/Hadoop
  - Spark on Kubernetes
- Hardware: Intel based servers, continuous refresh and capacity expansion

Accelerator logging (part of LHC infrastructure)	Hadoop - YARN - 30 nodes (Cores - 800, Mem - 13 TB, Storage – 7.5 PB)
General Purpose	Hadoop - YARN, 65 nodes (Cores – 1.3k, Mem – 20 TB, Storage – 12.5 PB)
Cloud containers	Kubernetes on Openstack VMs, Cores - 250, Mem – 2 TB Storage: remote HDFS or EOS (for physics data)

Scalability Tests – Optimization Results

Metric Name	**Total Time Spent (Sum Over al Executors)**	**% (Compared to Execution Time)**
Total Execution Time	~3000 - 3500 hours	1
CPU Time	~1200 hours	40%
EOS Read Time	~1200 - 1800 hours, depending on readAhead size	40-50%
Garbage Collection Time	~200 hours	7-8 %

- **Key workload metrics and time spent, measured with Spark custom instrumentation for 1 PB of input with 804 logical cores, 8 logical cores per Spark executor**

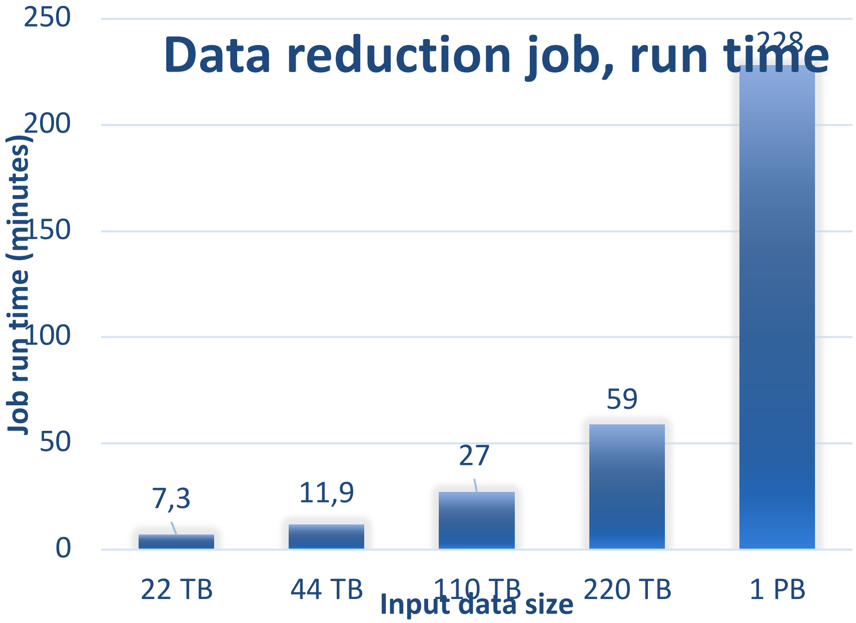


- **Read Throughput in GB/s**
- **Measure throughout during job execution for 1 PB of input with, 100 Spark executors, each using 8 logical cores.**

Scalability Tests - Results

- Performance and Scalability of the tests for different input size in minutes, 800 logical cores, and 8 logical cores per Spark executor

Input Data	Time for EOS Public
22 TB	7.3 mins
44 TB	11.9 mins
110 TB	27 mins (±2)
220 TB	59 mins (±5)
1 PB	228 mins (±10) (~3.8 hours)

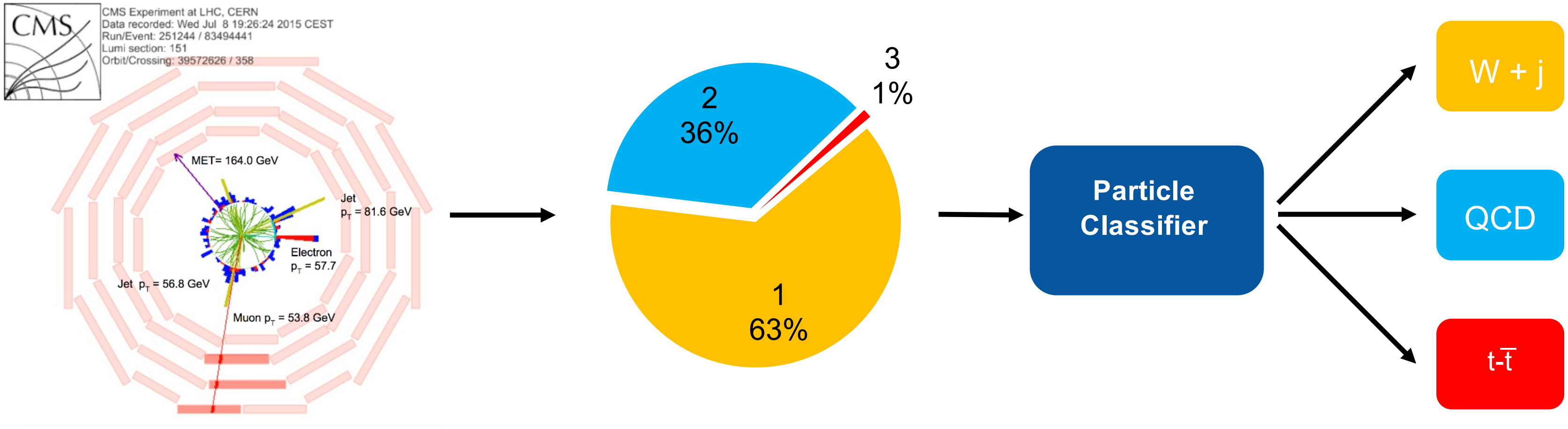


- Is it possible to reduce 1 PB in 5 hours (original project milestone)? YES.
  - It was even dropped to 4 hours in our latest tests

Machine Learning Use Case

Deep Learning Pipeline for Physics Data

- R&D to improve the **quality of filtering systems**
  - Develop a "Deep Learning classifier" to be used by the filtering system
  - Goal: Reduce false positives → do not store nor process uninteresting events
  - "Topology classification with deep learning to improve real-time event selection at the LHC", Nguyen et al. **Comput.Softw.Big Sci. 3 (2019) no.1, 12**



Particle Classifier

W + j

QCD

t-$\bar{t}$

Engineering Efforts to Enable Effective ML

- From “Hidden Technical Debt in Machine Learning Systems”, D. Sculley at al. (Google), paper at NIPS 2015

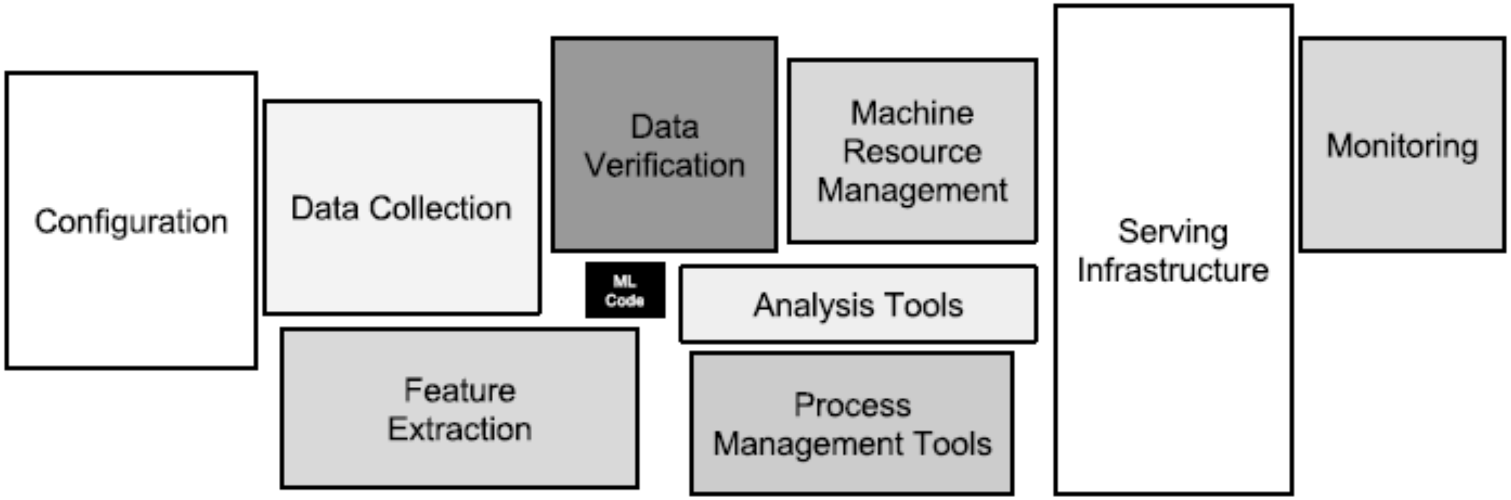
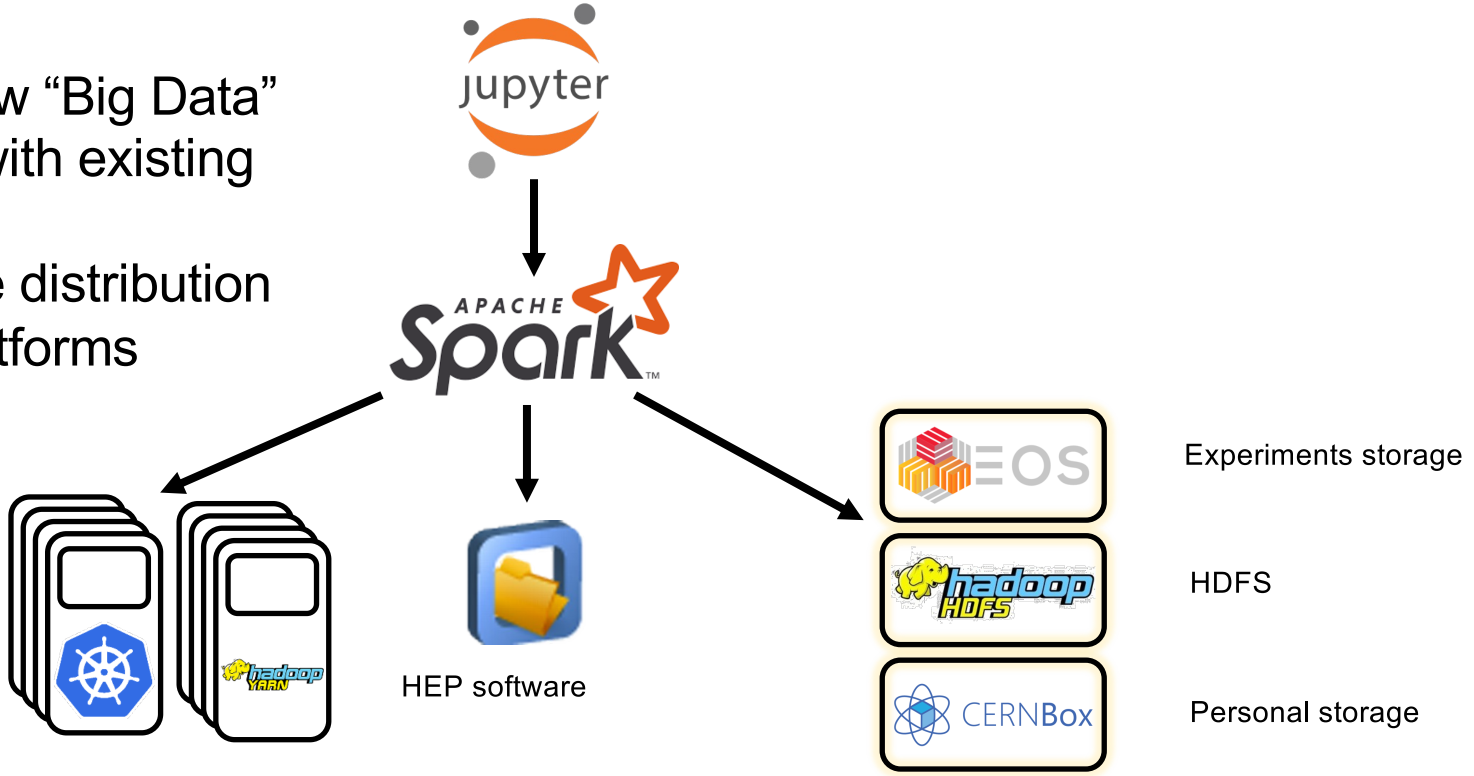


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

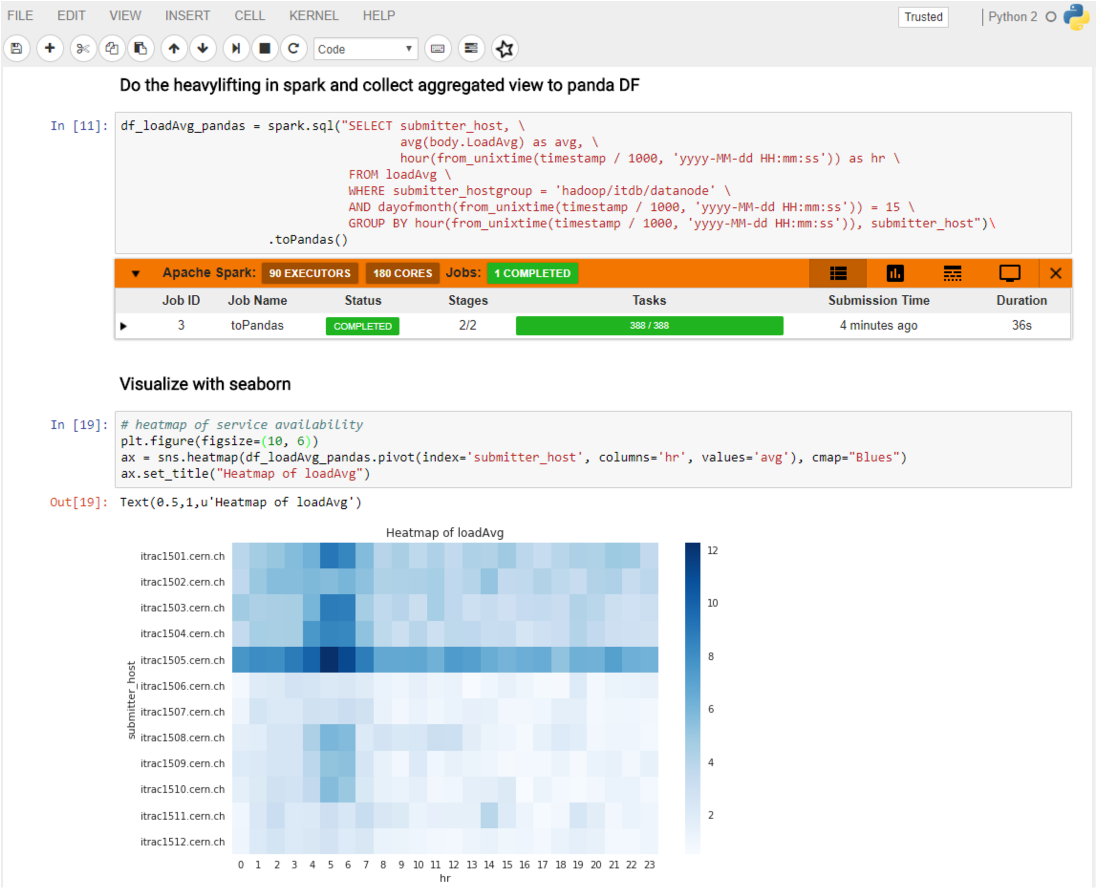
Analytics Platform at CERN

Integrating new “Big Data” components with existing infrastructure:

- Software distribution
- Data platforms



Analytics with SWAN



Do the heavylifting in spark and collect aggregated view to panda DF

```
In [11]: df_loadAvg_pandas = spark.sql("SELECT submitter_host, \
                                          avg(body.LoadAvg) as avg, \
                                          hour(from_unixtime(timestamp / 1000, 'yyyy-MM-dd HH:mm:ss')) as hr \
                                   FROM loadAvg \
                                   WHERE submitter_hostgroup = 'hadoop/itdb/datanode' \
                                   AND dayofmonth(from_unixtime(timestamp / 1000, 'yyyy-MM-dd HH:mm:ss')) = 15 \
                                   GROUP BY hour(from_unixtime(timestamp / 1000, 'yyyy-MM-dd HH:mm:ss')), submitter_host")\
                   .toPandas()
```

Apache Spark: 90 EXECUTORS 180 CORES Jobs: 1 COMPLETED

	Job ID	Job Name	Status	Stages	Tasks	Submission Time	Duration
▸	3	toPandas	COMPLETED	2/2	388 / 388	4 minutes ago	36s

Visualize with seaborn

```
In [19]: # heatmap of service availability
         plt.figure(figsize=(10, 6))
         ax = sns.heatmap(df_loadAvg_pandas.pivot(index='submitter_host', columns='hr', values='avg'), cmap="Blues")
         ax.set_title("Heatmap of loadAvg")

Out[19]: Text(0.5,1,u'Heatmap of loadAvg')
```

Text

Code

Monitoring

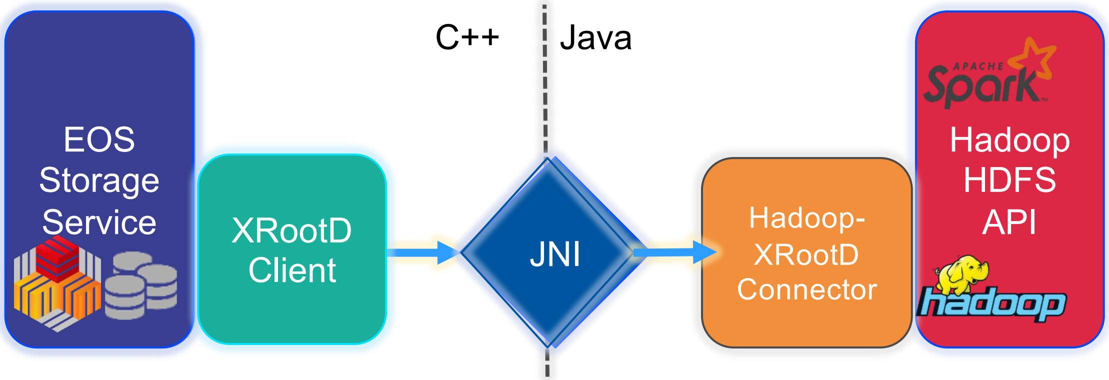
Visualizations

All the required tools, software and data available in a single window!

Extending Spark to Read Physics Data

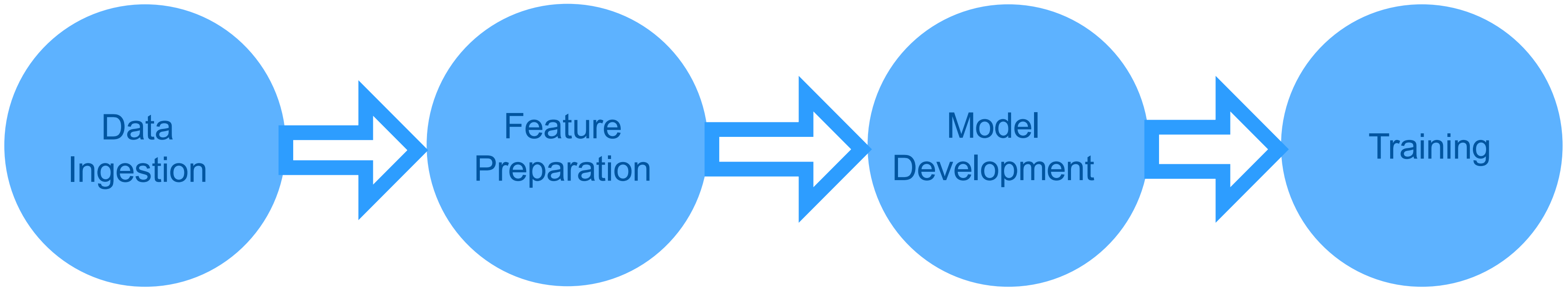
- Physics data is stored in EOS system, accessible with xrootd protocol: extended HDFS APIs
- Stored in ROOT format: developed a Spark Datasource

- Currently: 300 PBs
- Growing >50 PB/year



- https://github.com/cerndb/hadoop-xrootd
- https://github.com/diana-hep/spark-root

Deep Learning Pipeline for Physics Data



Read physics data and feature engineering

Prepare input for Deep Learning network

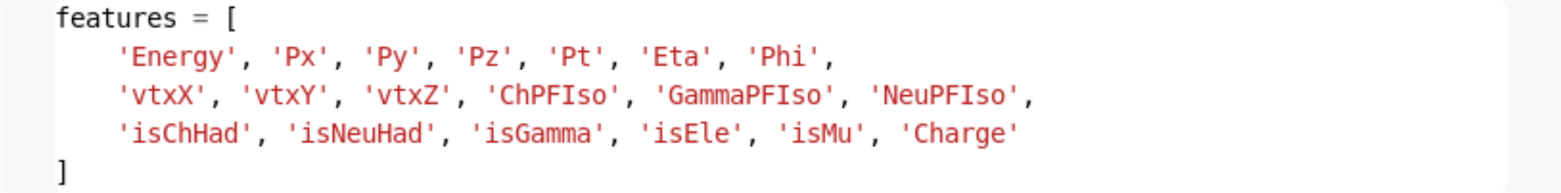
1. Specify model topology
2. Tune model topology on small dataset

Train the best model

Built with Apache Spark + Analytics Zoo + Python Notebooks

The Dataset

- Software simulators generate events (W+jets, tt, QCD) and calculate the detector response
- HEP knowledge to implement trigger selection:
  - all particles are then ranked in decreasing order of p_T
  - the isolated lepton is the first en-try of the list of particles
  - together with the isolated lepton, the first 450 charged particles, the first 150 photons, and the first 200 neutral hadrons, for a total of 801 particles with 19 features each
- Every event is a 801x19 matrix: for every particle momentum, position, energy, charge and particle type are given



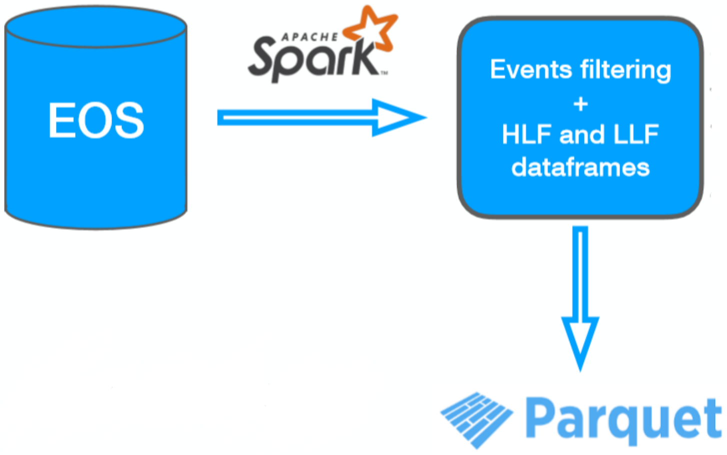
```
features = [
    'Energy', 'Px', 'Py', 'Pz', 'Pt', 'Eta', 'Phi',
    'vtxX', 'vtxY', 'vtxZ', 'ChPFIso', 'GammaPFIso', 'NeuPFIso',
    'isChHad', 'isNeuHad', 'isGamma', 'isEle', 'isMu', 'Charge'
]
```

Data Ingestion

- Read input files (4.5 TB) from ROOT format
- Compute physics-motivated features
- Store to parquet format

54 M events
4.5TB

Physics data
storage



EOS

Events filtering
+
HLF and LLF
dataframes

750 GBs
Stored on HDFS

Features Engineering

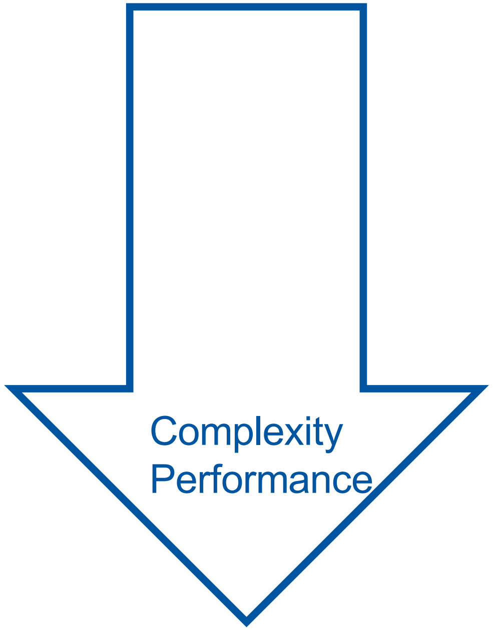
- From the 19 (low-level) features (LLF) recorded in the experiment:
  - 14 are calculated based on domain specific knowledge: these are called High Level Features (HLF)

- LLF and HLF datasets are saved in Apache Parquet format
  - the amount of training data is reduced at this point from the original 4.5 TB of ROOT files to 950 GB of snappy-compressed Parquet files.

- Order the sequence of particles to be fed to a sequence based classifier
  - The final sequence is ordered using custom Python code implementing physics

- The datasets, containing HLF and LLF features and labels, are split into training and test datasets (80% and 20% respectively) and saved in two separate Parquet files

Models Investigated



Complexity
Performance

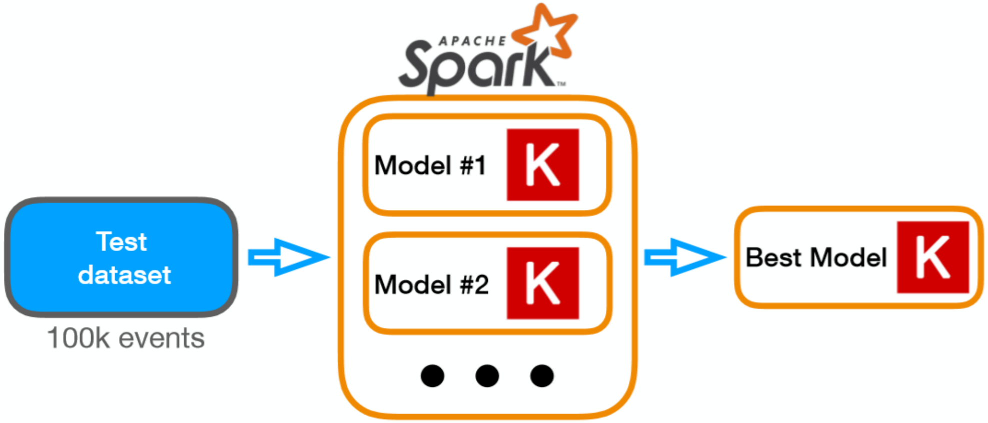
1. Fully connected feed-forward DNN with High Level Features (“HLF classifier”)

2. DNN with a recursive layer (based on GRUs) and used a “LLF/Particle Sequence classifier” with 801 particles

3. Combination of (1) + (2): “inclusive classifier”

Hyper-Parameter Tuning– DNN

- Once the network topology is chosen, hyper-parameter tuning is done with scikit-learn + Keras and parallelized with Spark
- the Area Under the ROC curve (AUC), as the performance metric to compare different classifiers
- fthe feed-forward DNN tuning done by changing the number of layers and units per layer, the activation function, the optimizer, etc.



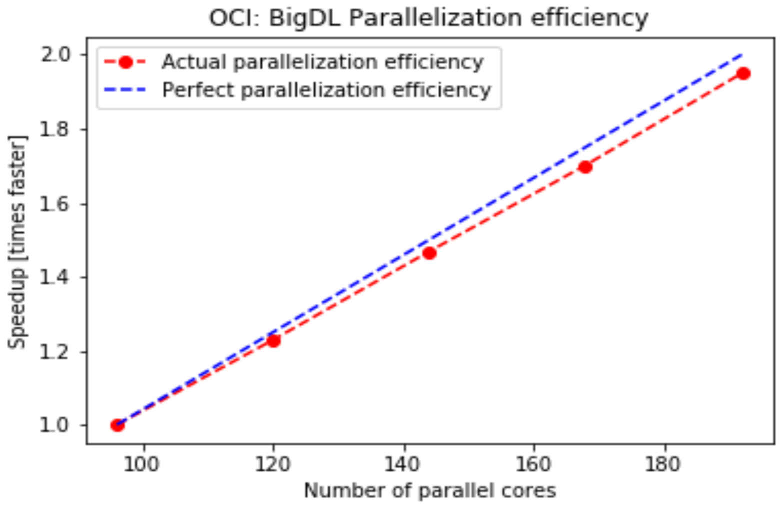
Analytics Zoo & BigDL tools for distributed training

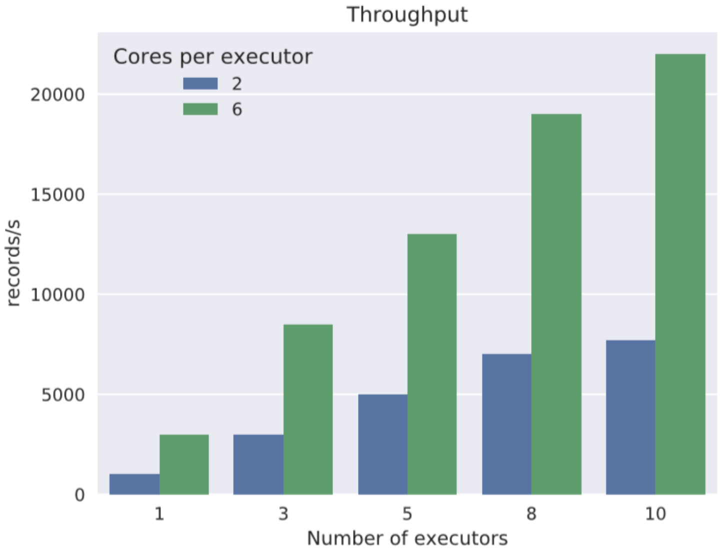
- Analytics Zoo is a platform for **unified** analytics and AI on Apache Spark leveraging BigDL / Tensorflow
  - For service developers: integration with the existing distributed and scalable analytics infrastructure (hardware, data access, data processing, configuration and operations)
  - For users: Keras APIs to run user models, integration with Spark data structures and pipelines

- BigDL is a distributed deep learning framework for Apache Spark

Performance and Scalability of Analytics Zoo & BigDL

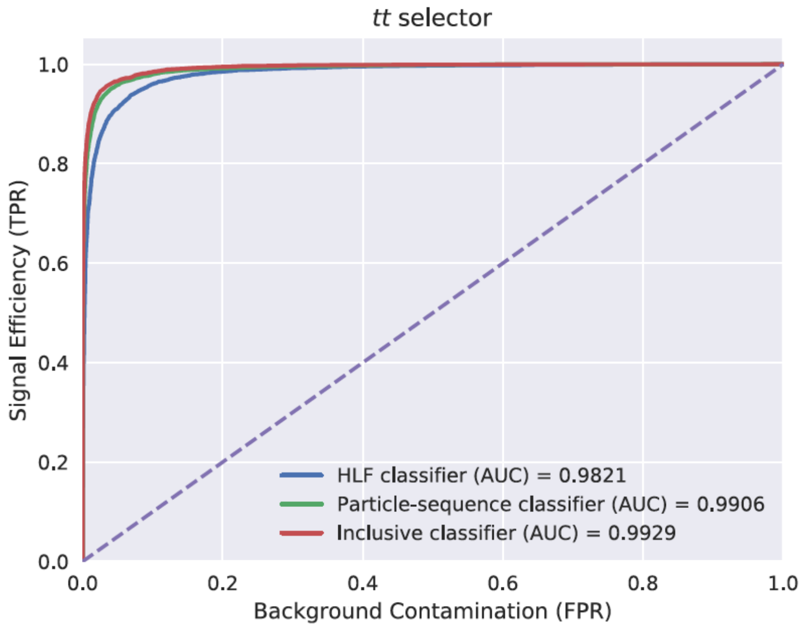
Analytics Zoo & BigDL scales very well in the ranges tested



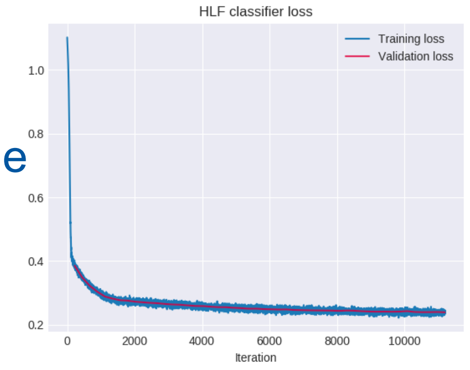


Results

- trained models with Analytics Zoo and BigDL
- results using the test dataset evaluated
- each of the models returns as output the probability that an input event is associated with a given topology: y_{QCD}, y_{W+jets} or y_{tt}.
- this can be used to define classifer, for example, by applying a threshold requirement on y_{tt} or y_{W} to define a W or a tt classifier
- performance of classifiers by comparing the ROC (receiver operating characteristic curve) curves and AUC

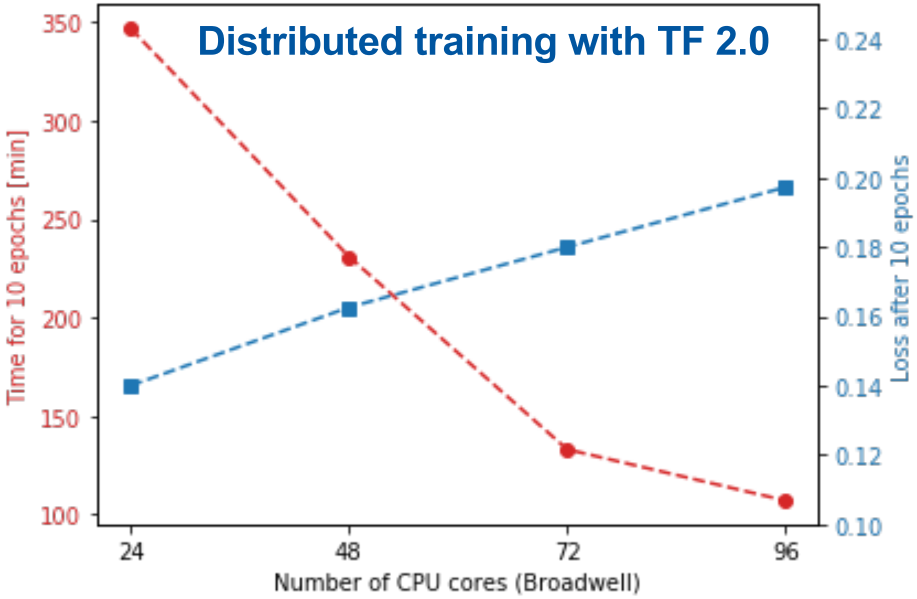


- the 3 models perfomed very well with comparable result (slightly better for the Particle Sequence classifier)
- smooth training convergence for the HLF classifier with the distributed training tools, reproducing original results

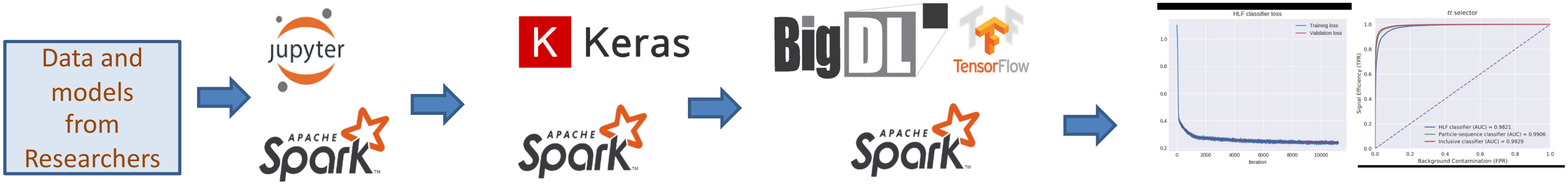


TensorFlow on Kubernetes

- Additional results using TensorFlow 2.0 on Kubernetes
  - CERN Cloud on Openstack
  - TF.distribute Multi Worker Strategy on K8S: https://github.com/cerndb/tf-spawner
  - Data transformed from Parquet to TFRecord using Spark, then fed to TF.Data



Machine Learning with Spark and Keras



Data and models from Researchers

Input: labeled data and DL models

Feature engineering at scale

Hyperparameter optimization (Random/Grid search)

Distributed model training

Output: particle selector model

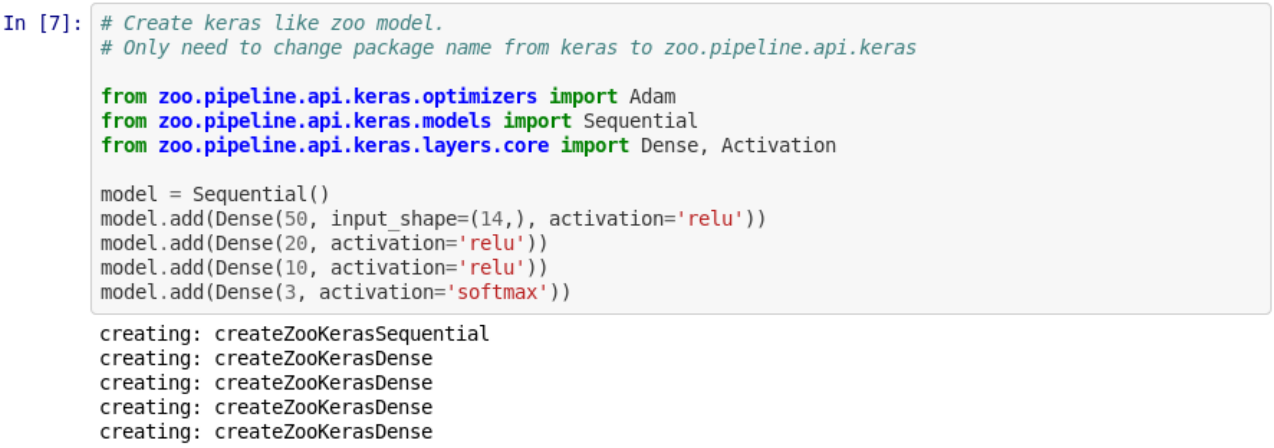
Conclusions

- Spark and “Big Data”-based analysis platforms can improve High Energy Physics data pipelines
  - Industry-standard APIs
  - Run natively on “data lakes” and cloud
  - Profit from large communities in industry and open source
- Two use cases developed
  - CMS Data reduction at scale with Apache Spark
  - Deep learning pipeline with Spark + BigDL and TensorFLow
- Analytics platform at CERN
  - Open for access to CERN community, notably users in Physics, Beams and Accelerators, IT.
- References:
  - Using Big Data Technologies for HEP Analysis https://doi.org/10.1051/epjconf/201921406030
  - Machine Learning Pipelines with Modern Big Data Tools for High Energy Physics http://arxiv.org/abs/1909.10389

Backup

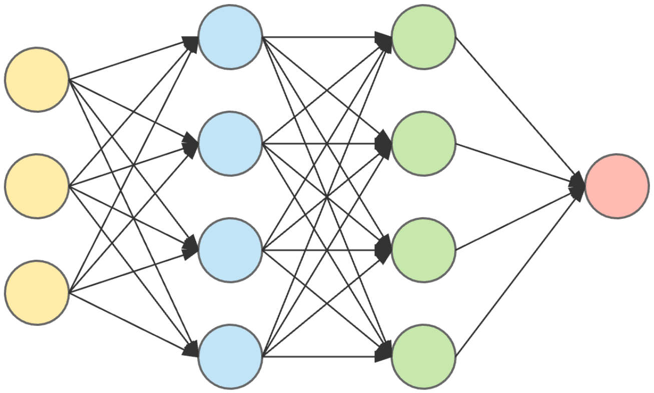
Model Development – DNN

- Model is instantiated with the Keras-compatible API provided by Analytics Zoo



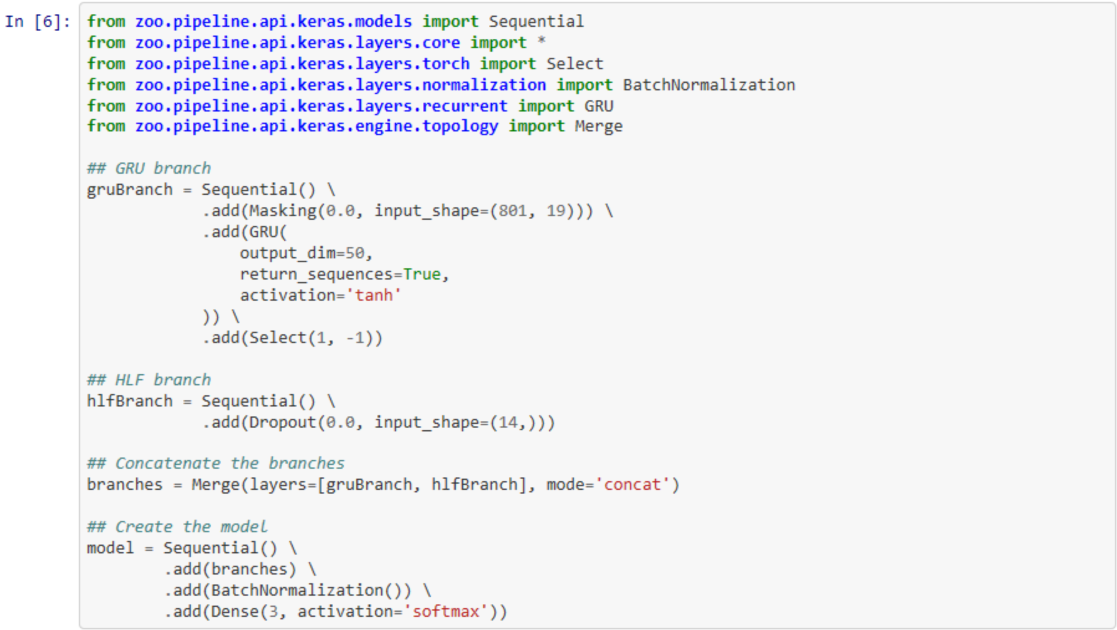
```
In [7]: # Create keras like zoo model.
        # Only need to change package name from keras to zoo.pipeline.api.keras

        from zoo.pipeline.api.keras.optimizers import Adam
        from zoo.pipeline.api.keras.models import Sequential
        from zoo.pipeline.api.keras.layers.core import Dense, Activation

        model = Sequential()
        model.add(Dense(50, input_shape=(14,), activation='relu'))
        model.add(Dense(20, activation='relu'))
        model.add(Dense(10, activation='relu'))
        model.add(Dense(3, activation='softmax'))
```

```
creating: createZooKerasSequential
creating: createZooKerasDense
creating: createZooKerasDense
creating: createZooKerasDense
creating: createZooKerasDense
```

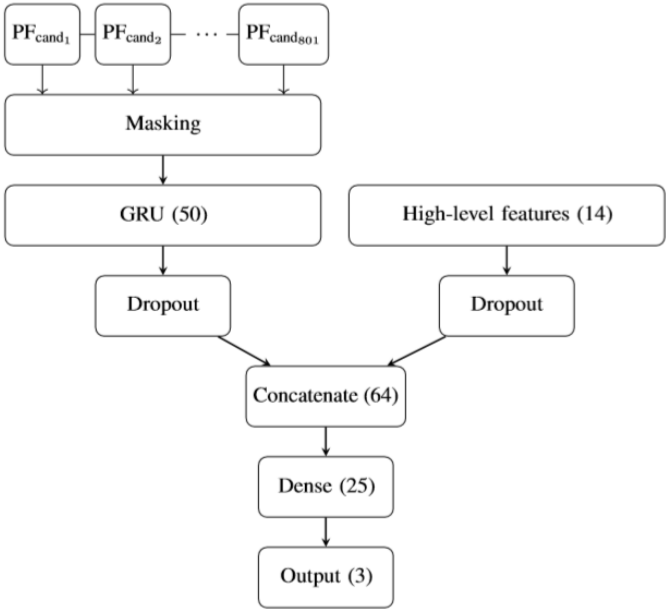


Model Development – GRU+HLF

A more complex topology for the network

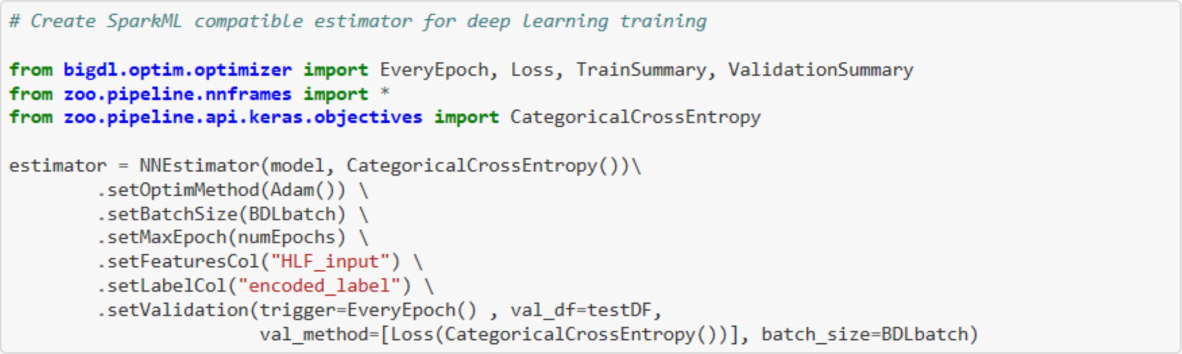


```
In [6]: from zoo.pipeline.api.keras.models import Sequential
        from zoo.pipeline.api.keras.layers.core import *
        from zoo.pipeline.api.keras.layers.torch import Select
        from zoo.pipeline.api.keras.layers.normalization import BatchNormalization
        from zoo.pipeline.api.keras.layers.recurrent import GRU
        from zoo.pipeline.api.keras.engine.topology import Merge

        ## GRU branch
        gruBranch = Sequential() \
                    .add(Masking(0.0, input_shape=(801, 19))) \
                    .add(GRU(
                        output_dim=50,
                        return_sequences=True,
                        activation='tanh'
                    )) \
                    .add(Select(1, -1))

        ## HLF branch
        hlfBranch = Sequential() \
                    .add(Dropout(0.0, input_shape=(14,)))

        ## Concatenate the branches
        branches = Merge(layers=[gruBranch, hlfBranch], mode='concat')

        ## Create the model
        model = Sequential() \
                .add(branches) \
                .add(BatchNormalization()) \
                .add(Dense(3, activation='softmax'))
```


$\mathrm{PF}_{\mathrm{cand}_1}$
$\mathrm{PF}_{\mathrm{cand}_2}$
· · ·
$\mathrm{PF}_{\mathrm{cand}_{801}}$
Masking
GRU (50)
High-level features (14)
Dropout
Dropout
Concatenate (64)
Dense (25)
Output (3)

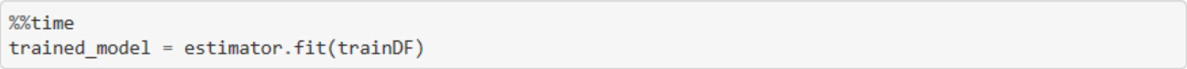
Distributed Training

Instantiate the estimator using Analytics Zoo / BigDL

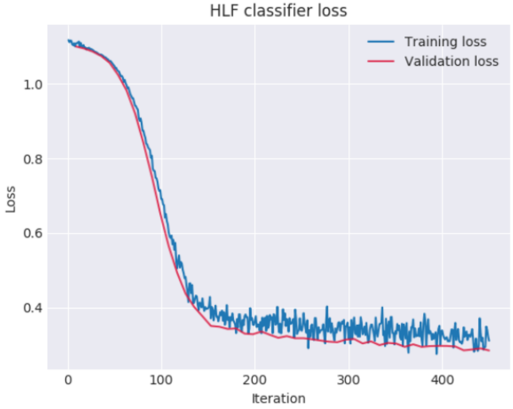


```
# Create SparkML compatible estimator for deep learning training

from bigdl.optim.optimizer import EveryEpoch, Loss, TrainSummary, ValidationSummary
from zoo.pipeline.nnframes import *
from zoo.pipeline.api.keras.objectives import CategoricalCrossEntropy

estimator = NNEstimator(model, CategoricalCrossEntropy())\
        .setOptimMethod(Adam()) \
        .setBatchSize(BDLbatch) \
        .setMaxEpoch(numEpochs) \
        .setFeaturesCol("HLF_input") \
        .setLabelCol("encoded_label") \
        .setValidation(trigger=EveryEpoch() , val_df=testDF,
                       val_method=[Loss(CategoricalCrossEntropy())], batch_size=BDLbatch)
```

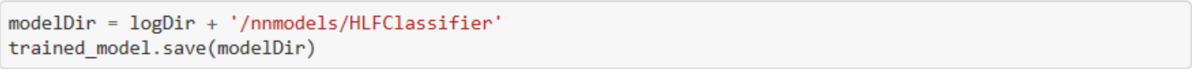
The actual training is distributed to Spark executors



```
%%time
trained_model = estimator.fit(trainDF)
```



Storing the model for later use



```
modelDir = logDir + '/nnmodels/HLFClassifier'
trained_model.save(modelDir)
```