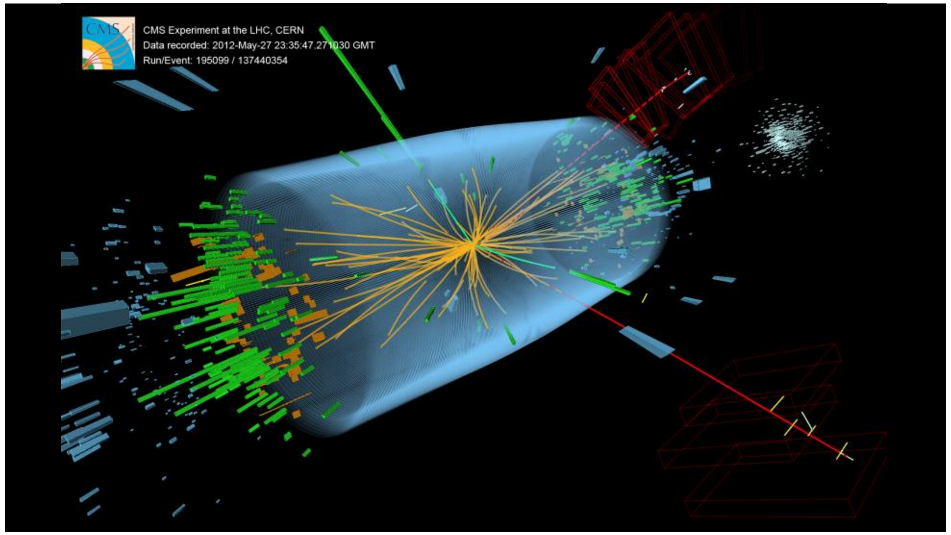
“Big Data In HEP” - Physics Data Analysis, Machine Learning and Data Reduction at Scale with Apache Spark

Luca Canali (CERN), Vaggelis Motesnitsalis (CERN),
Oliver Gutsche (Fermilab)

IXPUG Annual Conference 2019
September 24th, 2019

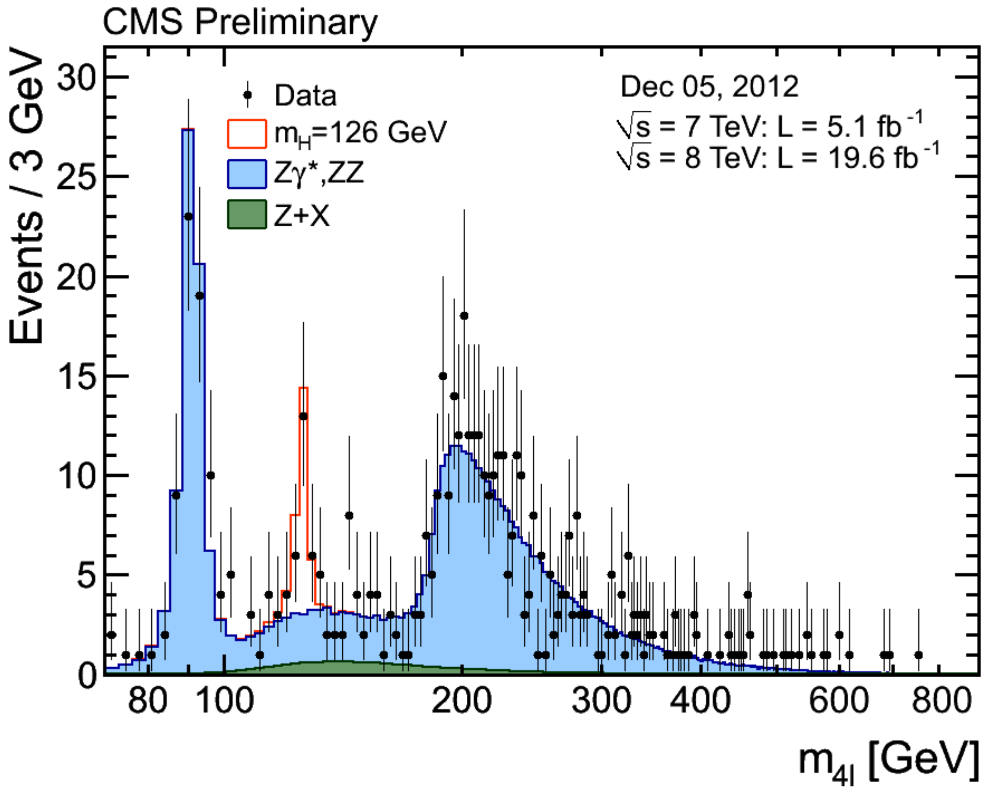
Experimental Particle Physics - the Journey

Particle Collisions

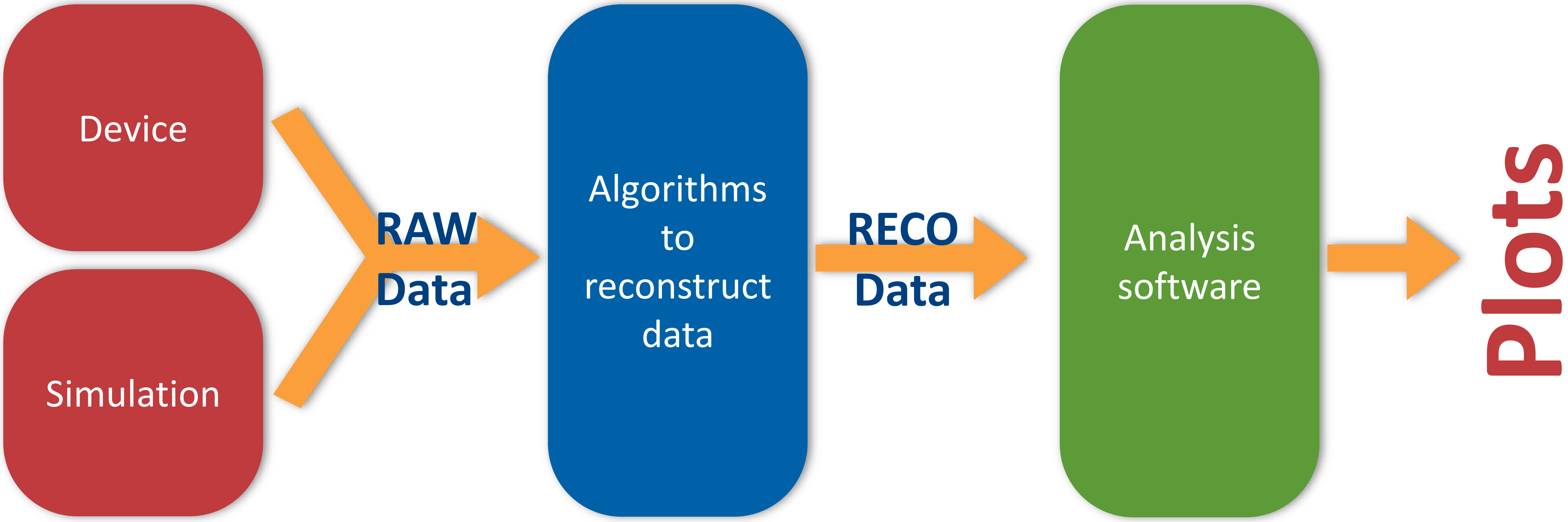


Large Scale Computing

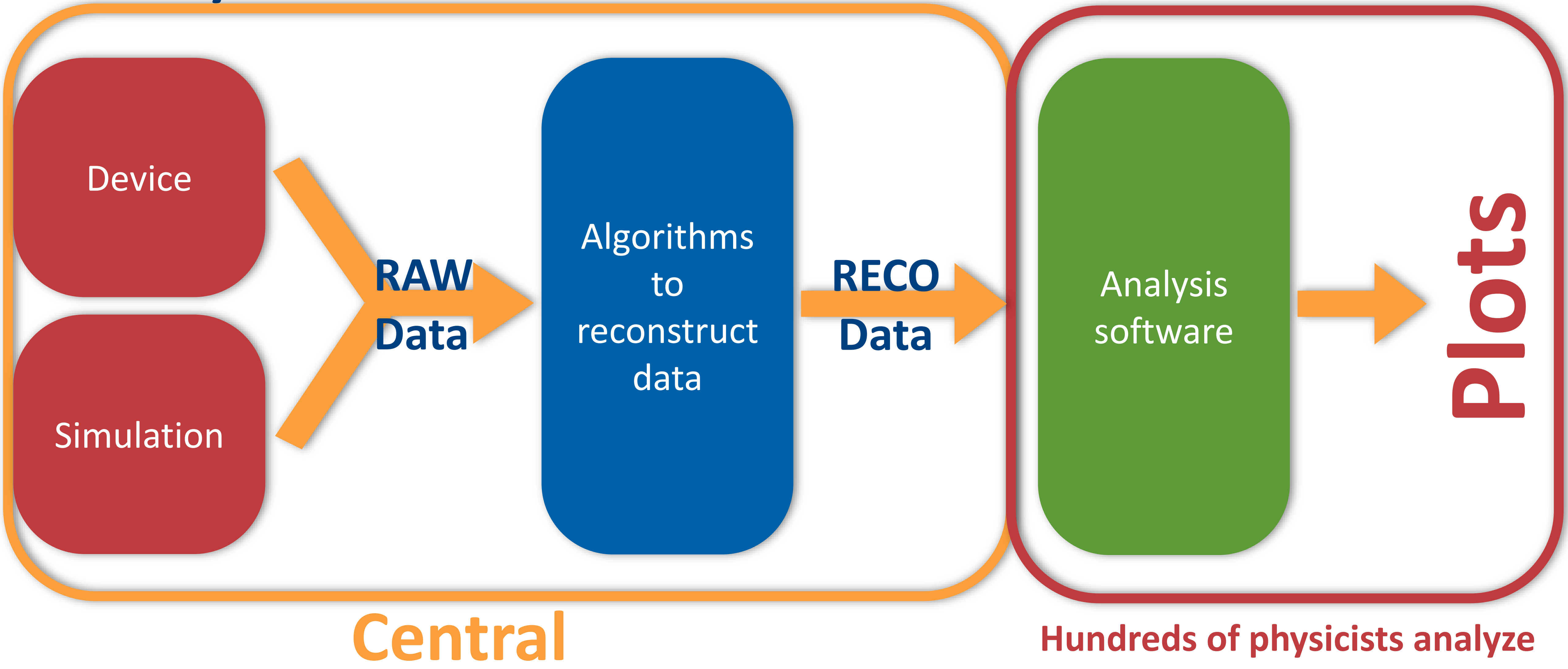
Physics Discoveries



Large Scale Computing

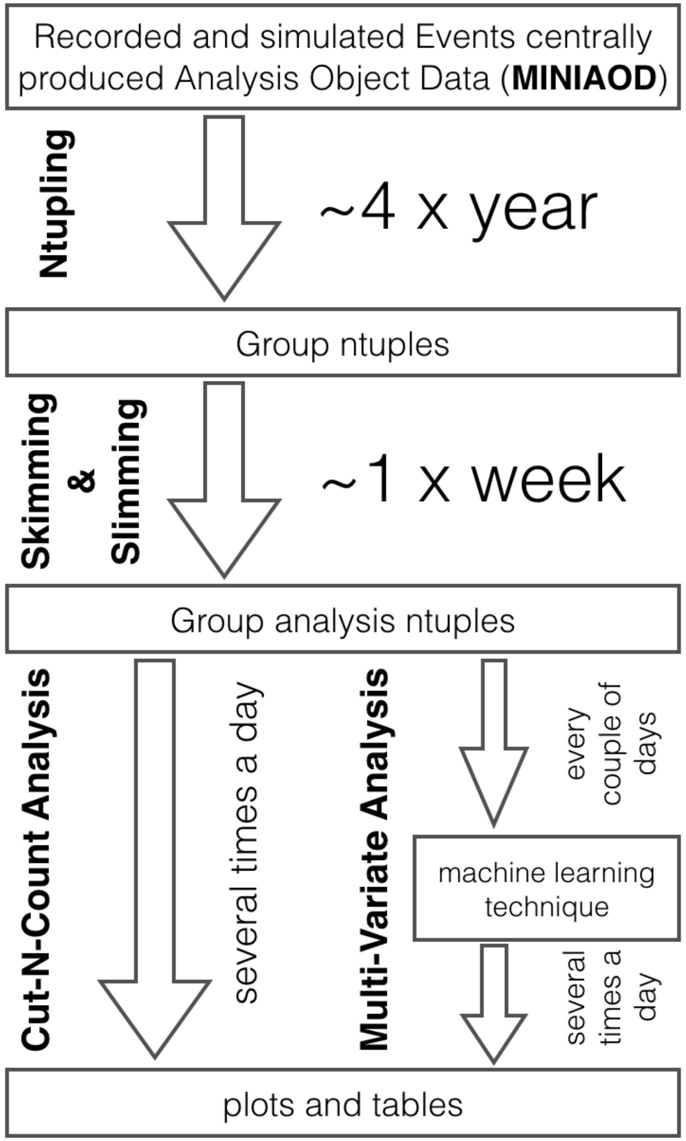


Analysis in CMS



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

Analysis: A multi-step Process

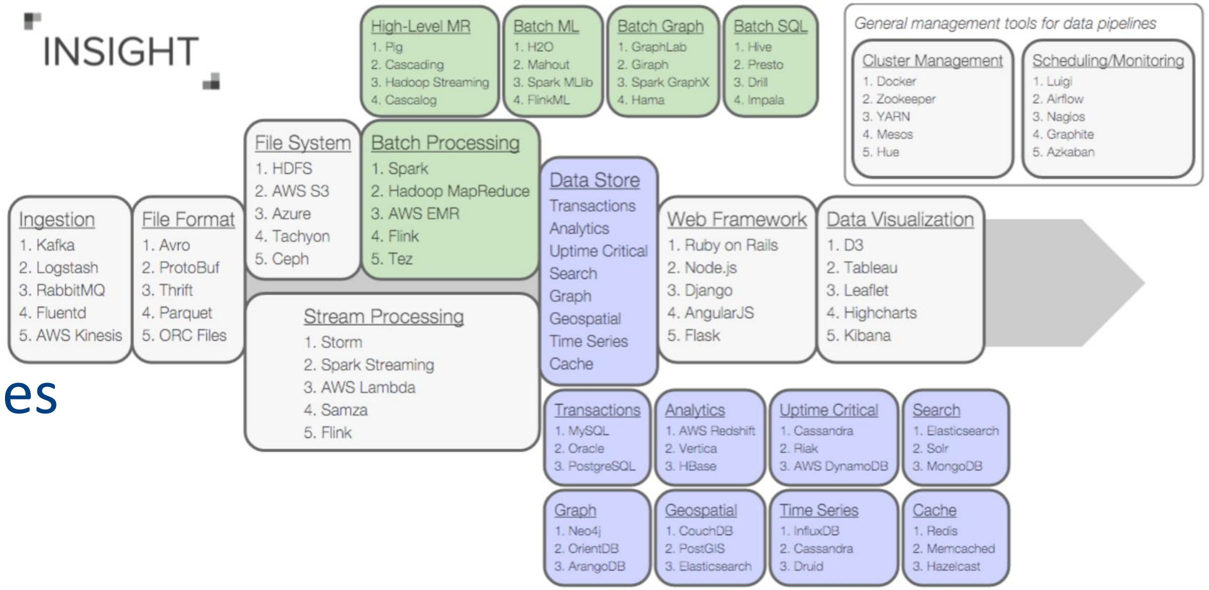


- Minimize **Time to Insight**
  - Analysis is a conversation with data - Interactivity is key
- Many different physics topics concurrently under investigation
  - Different slices of data are relevant for each analysis
- Programmatically same analysis steps
  - Skimming (dropping events in a disk-to-disk 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



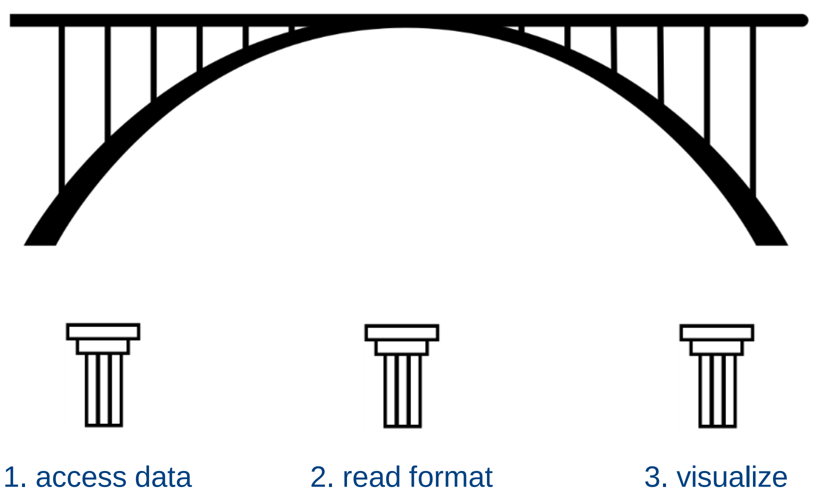
Bridging the Gap

- 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.

ROOT Data Analysis Framework

EOS Storage Service

WLCG Worldwide LHC Computing Grid

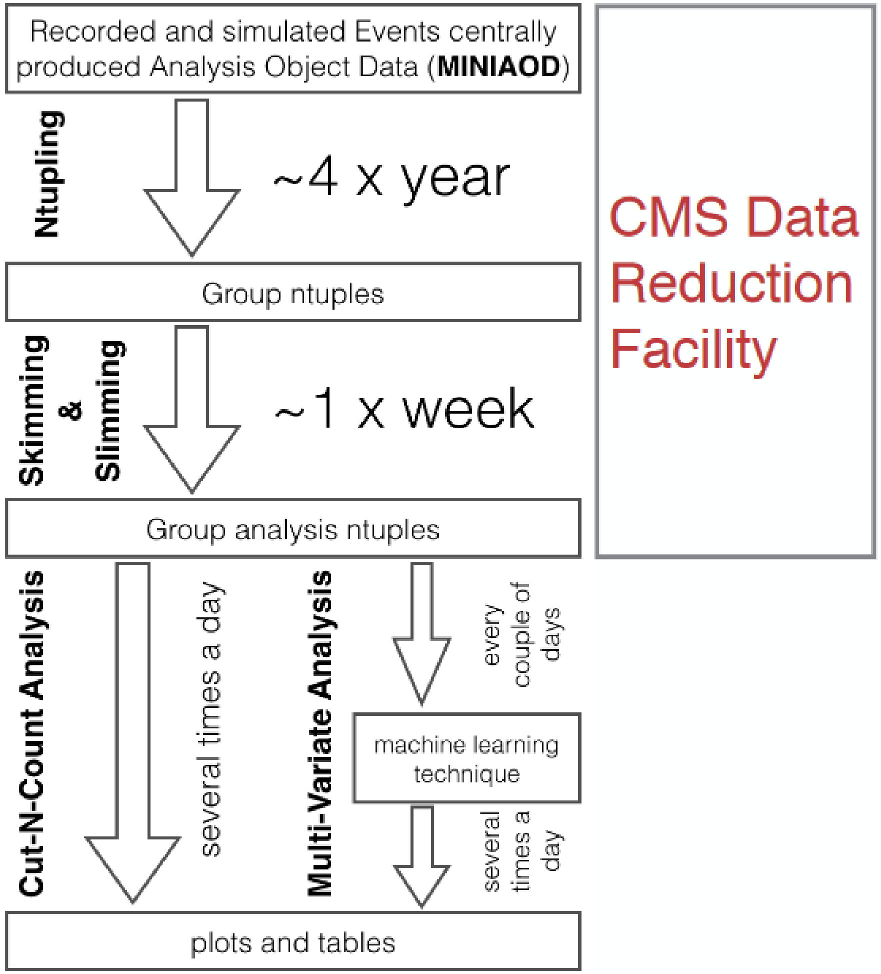


APACHE Spark

hadoop

jupyter

CMS Data Reduction and Analysis Facility



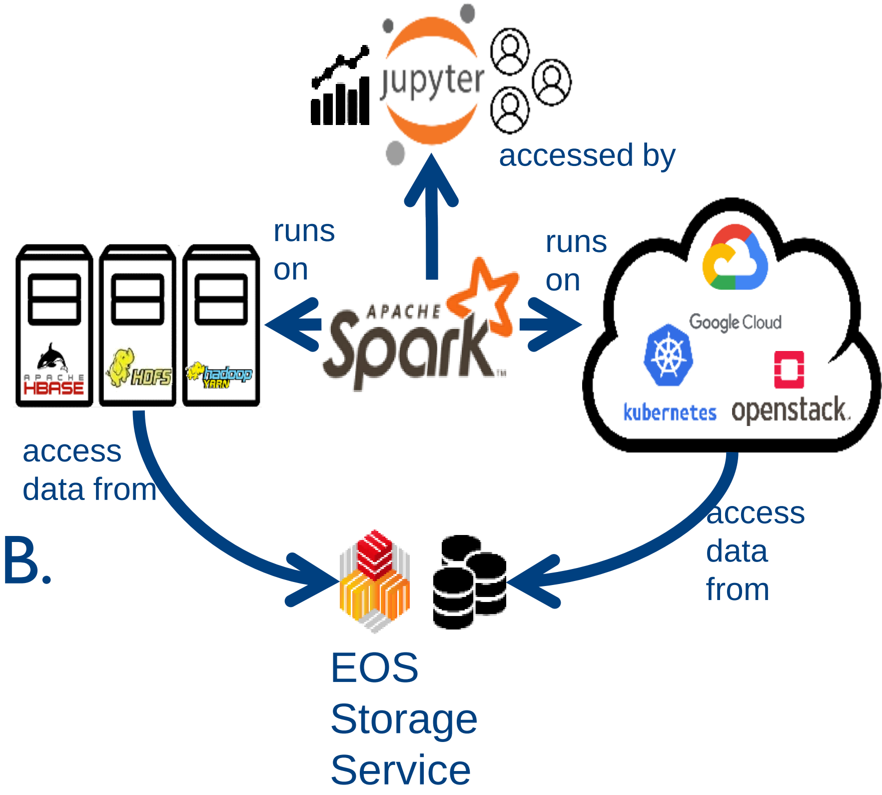
- CERN openlab / Intel project
- Demonstrate reduction capabilities producing analysis ntuples using Apache Spark
  - Demonstrator's goal: reduce 1 PB input in 5 hours

Milestones and Achievements

- We solved two important data engineering challenges:

1. Read files in ROOT Format using Spark
2. Access files stored in EOS directly from Hadoop/Spark

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



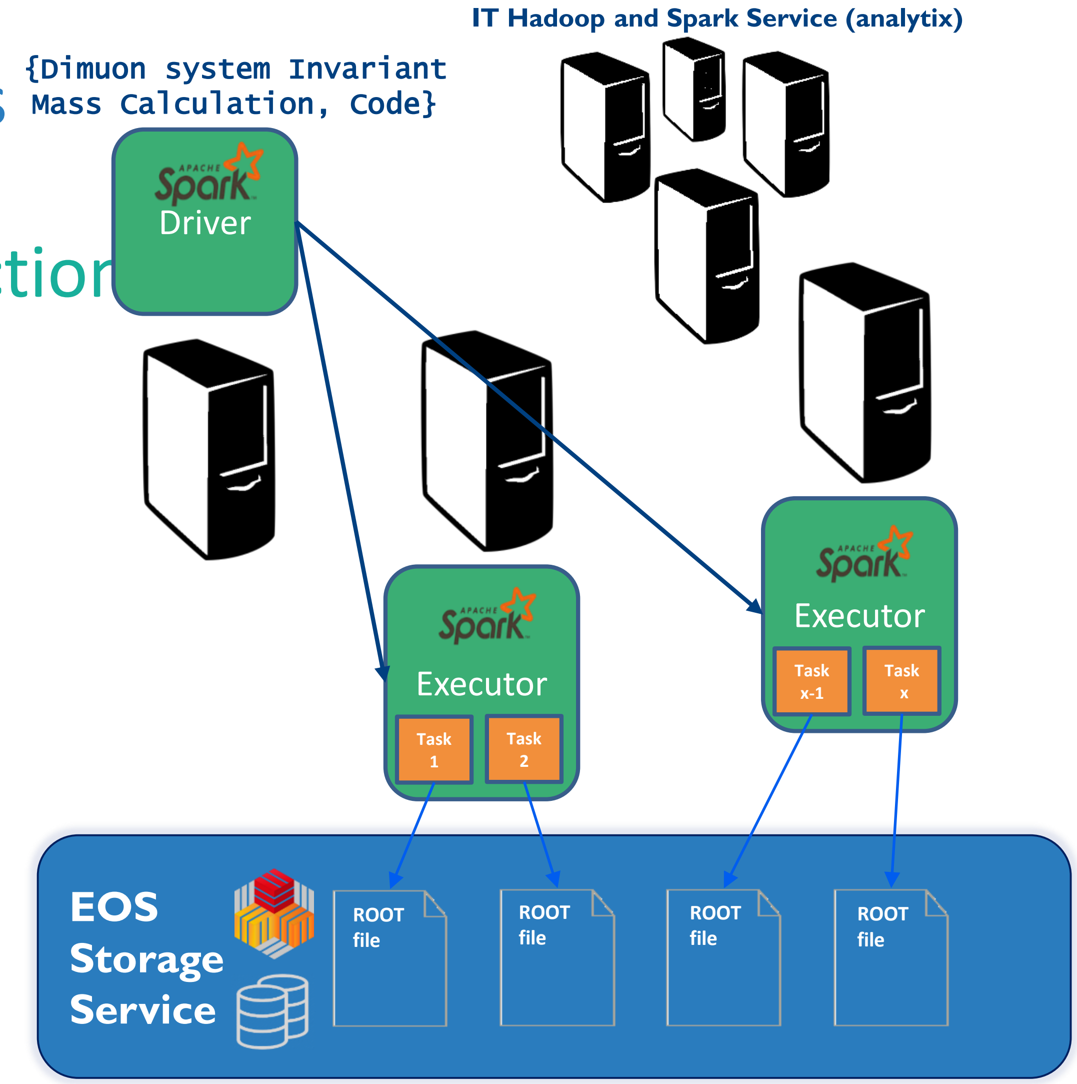
Scalability Tests

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
  - Hadoop YARN
  - Kubernetes and Openstack
  - Collaborated with Intel for Spark jobs optimizations: using Intel CoFluent Cluster Simulation Technology

- 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 that we 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

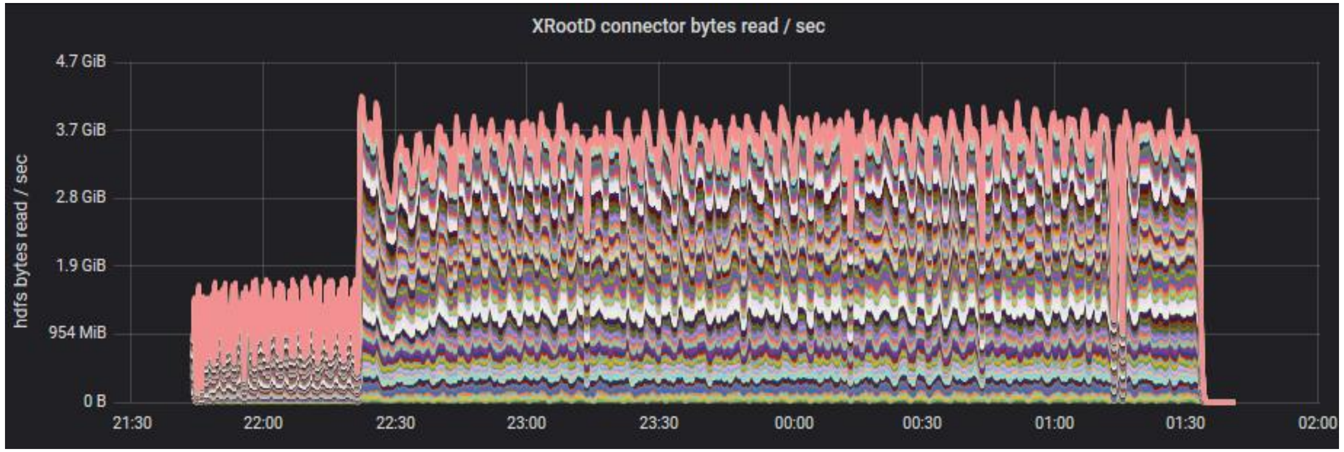
- Clusters:
  - YARN/Hadoop
  - Spark on Kubernetes
- Hardware: Intel based servers, continuous refresh and capacity expansion

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**

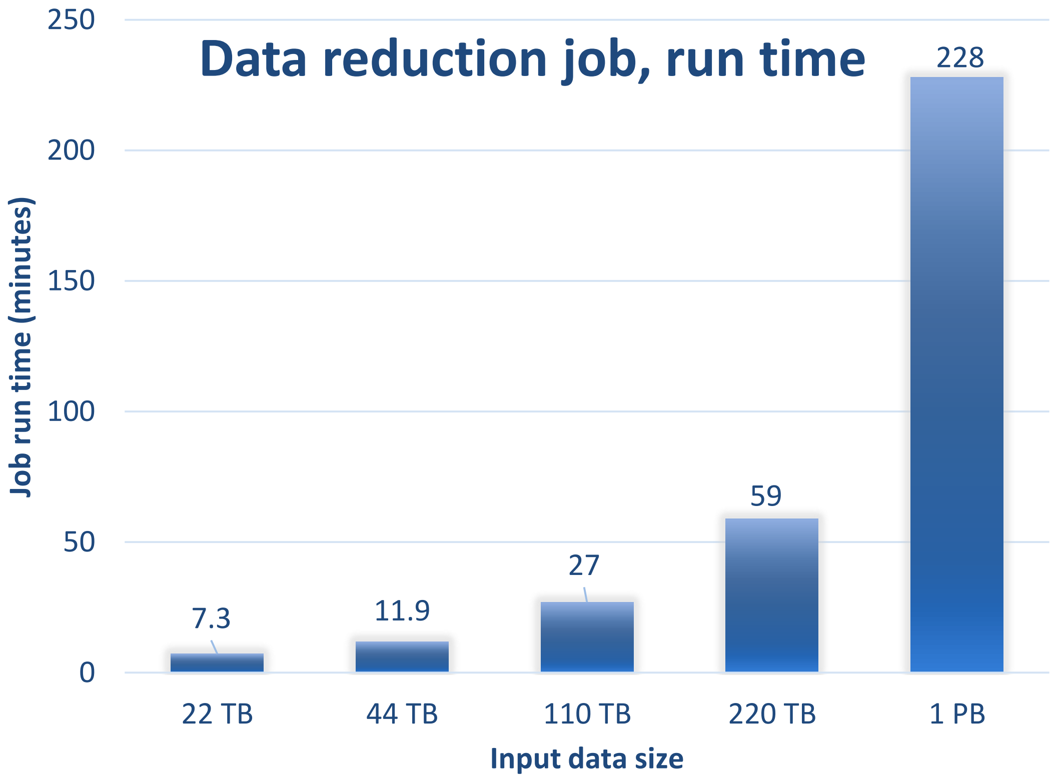


- **Read Throughput in GB/s**
- **Measure throughout during job execution for 1 PB of input with, 100 Spark executors, each using 8 logical cores.**

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)

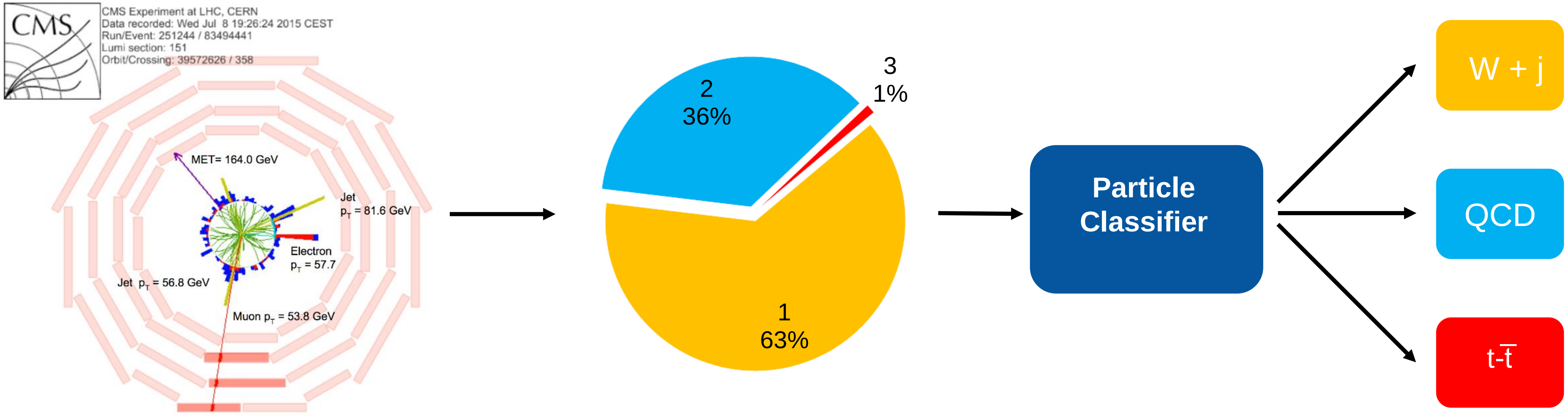


- Can we reduce 1 PB in 5 hours (original project milestone)? YES.
  - We even dropped to 4 hours in our latest tests

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**



Engineering Efforts to Enable Effective ML

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

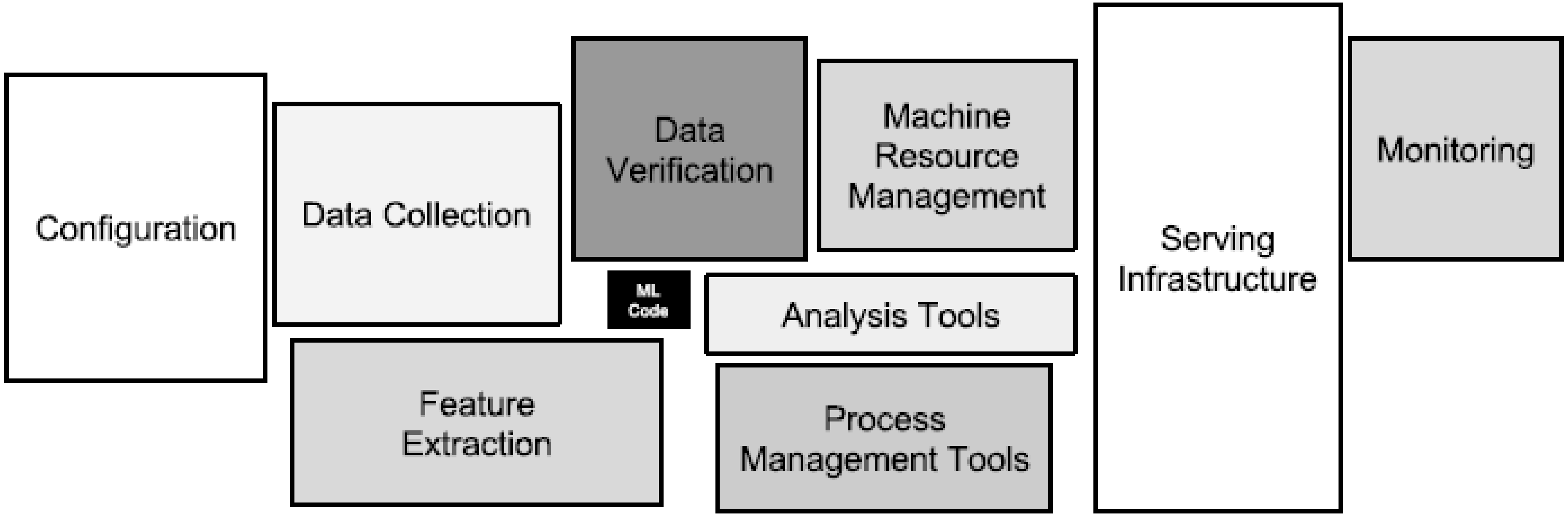
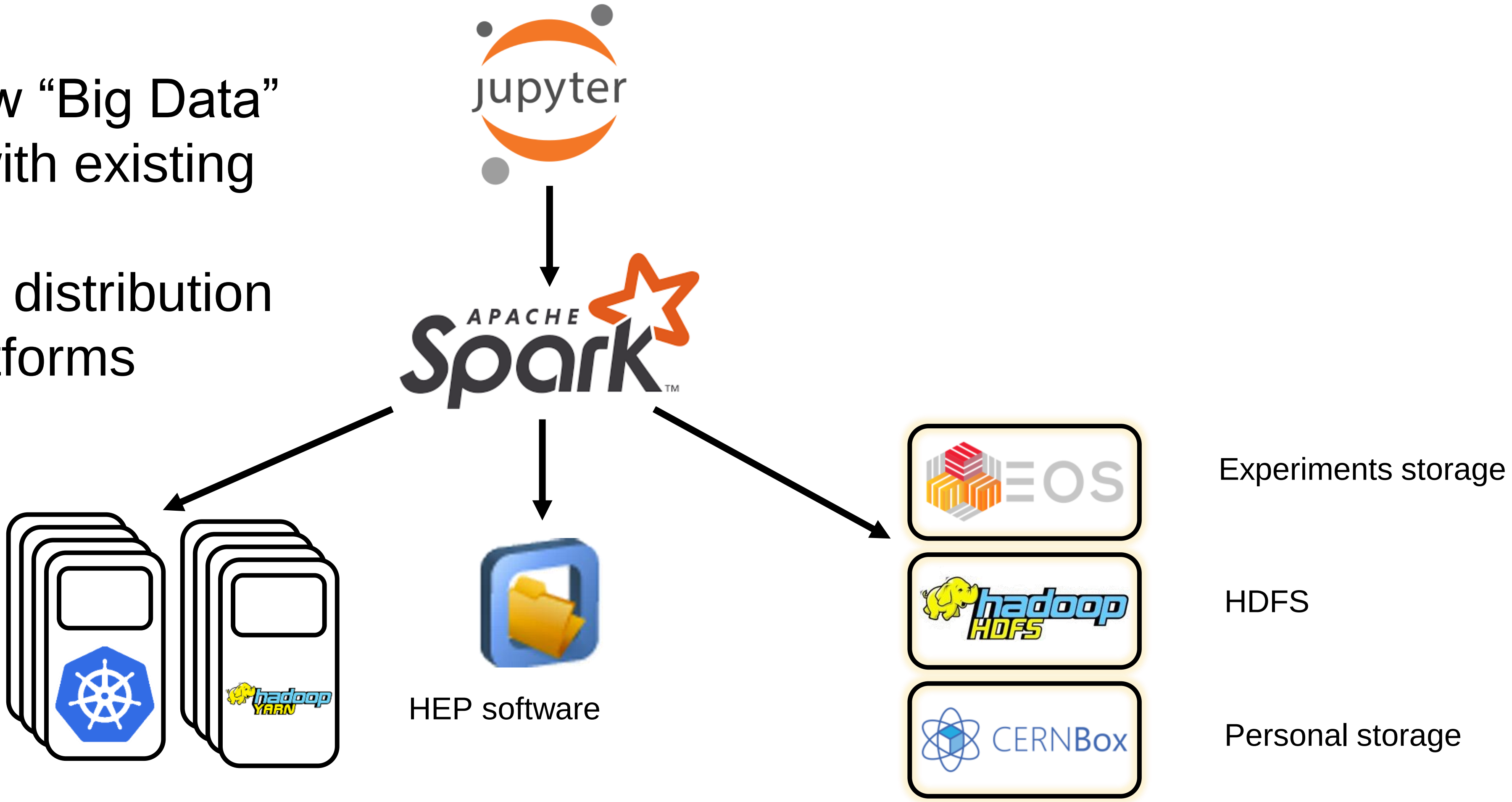


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Analytics Platform at CERN

Integrating new “Big Data” components with existing infrastructure:

- Software distribution
- Data platforms



Analytics with SWAN

FILE EDIT VIEW INSERT CELL KERNEL HELP | Trusted | Python 2

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()
```

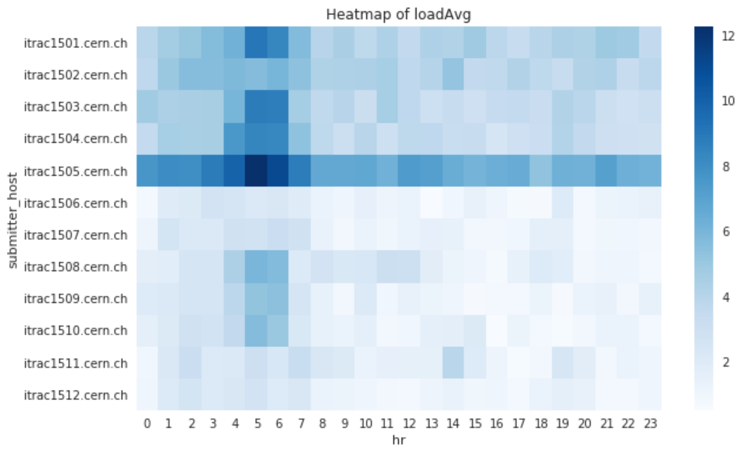
Apache Spark: 90 EXECUTORS 180 CORES Jobs: 1 COMPLETED

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

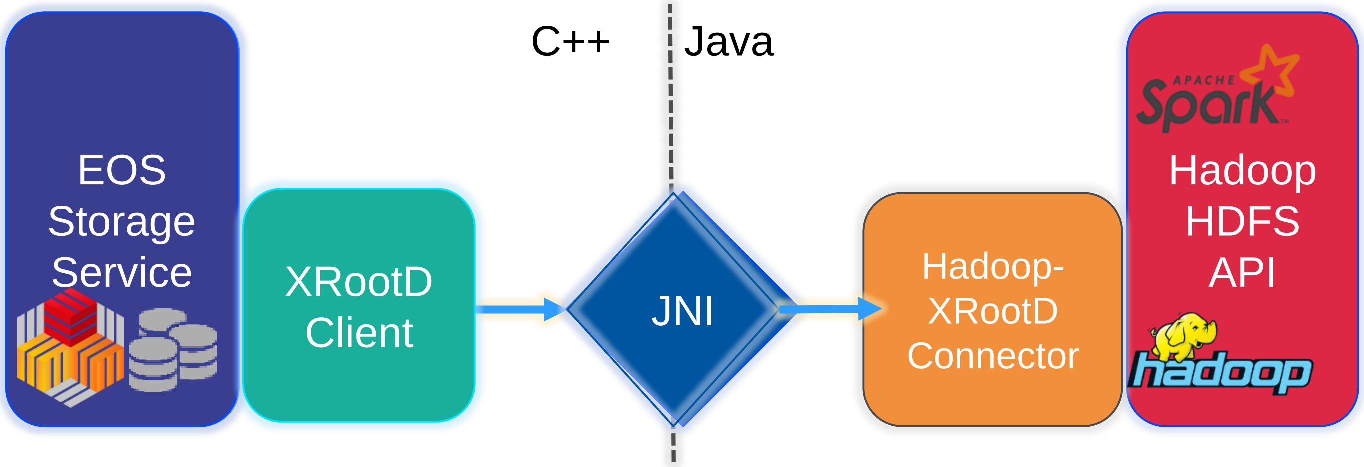
Visualizations

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

Data Ingestion → Feature Preparation → Model Development → Training

Data Ingestion

Read physics data and feature engineering

Feature Preparation

Prepare input for Deep Learning network

Model Development

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

Training

Train the best model

Built with Apache Spark + Analytics Zoo + Python Notebooks

The Dataset

- Software simulators generate events and calculate the detector response
- Every event is a 801x19 matrix: for every particle momentum, position, energy, charge and particle type are given

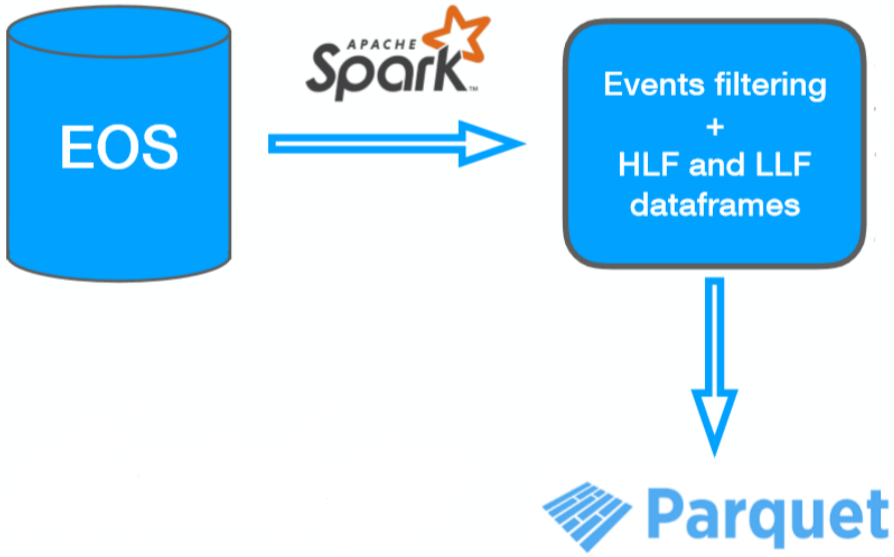
```
features = [
    'Energy', 'Px', 'Py', 'Pz', 'Pt', 'Eta', 'Phi',
    'vtxX', 'vtxY', 'vtxZ', 'ChPFIso', 'GammaPFIso', 'NeuPFIso',
    'isChHad', 'isNeuHad', 'isGamma', 'isEle', 'isMu', 'Charge'
]
```

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



750 GBs
Stored on HDFS

Features Engineering

- From the 19 features recorded in the experiment:
  - 14 more are calculated based on domain specific knowledge: these are called High Level Features (HLF)
- Order the sequence of particles to be fed to a sequence based classifier
  - The final sequence is ordered using custom Python code implementing physics

Feature Preparation

- All features need to be converted to a format consumable by the neural network
  - One Hot Encoding of categories
  - Sort the particles for the sequence classifier with a UDF
- Executed in PySpark using Spark SQL and ML

Feature preparation

Elements of the `hfeatures` column are list, hence we need to convert them into `Vectors.Dense`

In [10]:
```
from pyspark.ml.linalg import Vectors, VectorUDT
from pyspark.sql.functions import udf

vector_dense_udf = udf(lambda r : Vectors.dense(r),VectorUDT())
data = data.withColumn('hfeatures_dense',vector_dense_udf('hfeatures'))
```

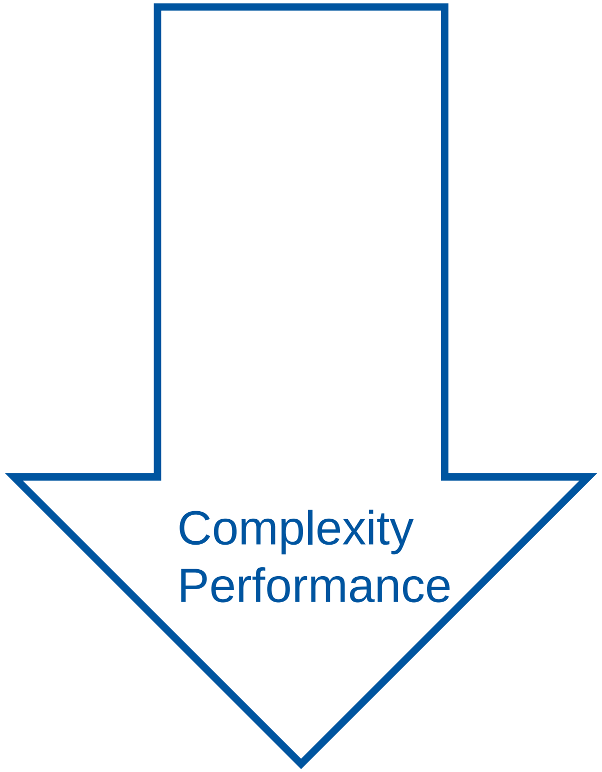
Now we can build the pipeline to scale HLF and encode the labels

In [11]:
```
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoderEstimator
from pyspark.ml.feature import MinMaxScaler

## One-Hot-Encode
encoder = OneHotEncoderEstimator(inputCols=["label"],
                                 outputCols=["encoded_label"],
                                 dropLast=False)

## Scale feature vector
scaler = MinMaxScaler(inputCol="hfeatures_dense",
                      outputCol="HLF_input")

pipeline = Pipeline(stages=[encoder, scaler])

%time fitted_pipeline = pipeline.fit(data)
```

```
CPU times: user 294 ms, sys: 293 ms, total: 587 ms
Wall time: 1min 34s
```

In [12]:
```
data = fitted_pipeline.transform(data)
```

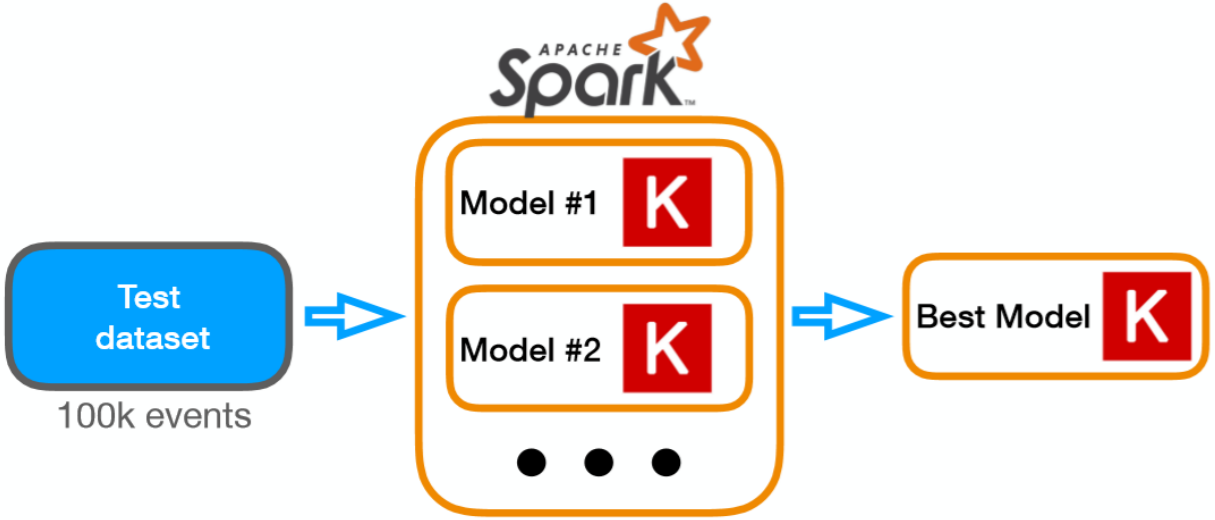
Models Investigated



1. Fully connected feed-forward DNN with High Level Features
2. DNN with a recursive layer (based on GRUs)
3. Combination of (1) + (2)

Hyper-Parameter Tuning– DNN

- Once the network topology is chosen, hyper-parameter tuning is done with scikit-learn + Keras and parallelized with Spark



Analytics Zoo & BigDL

- 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

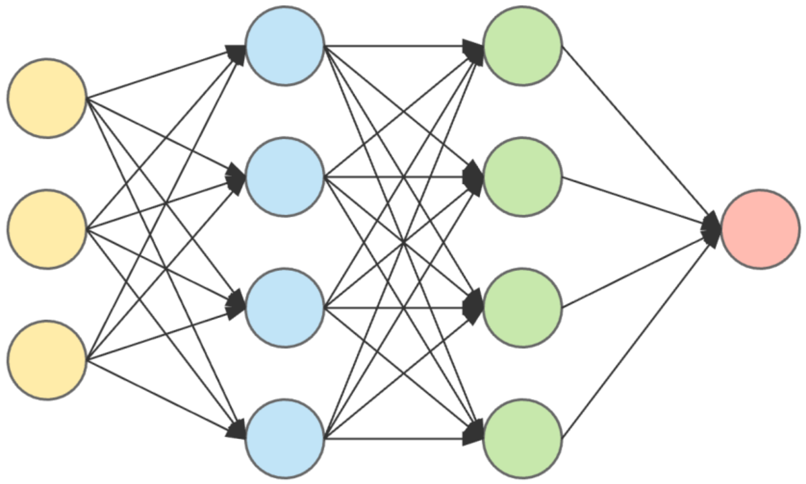
- BigDL is a distributed deep learning framework for Apache Spark

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'))
```

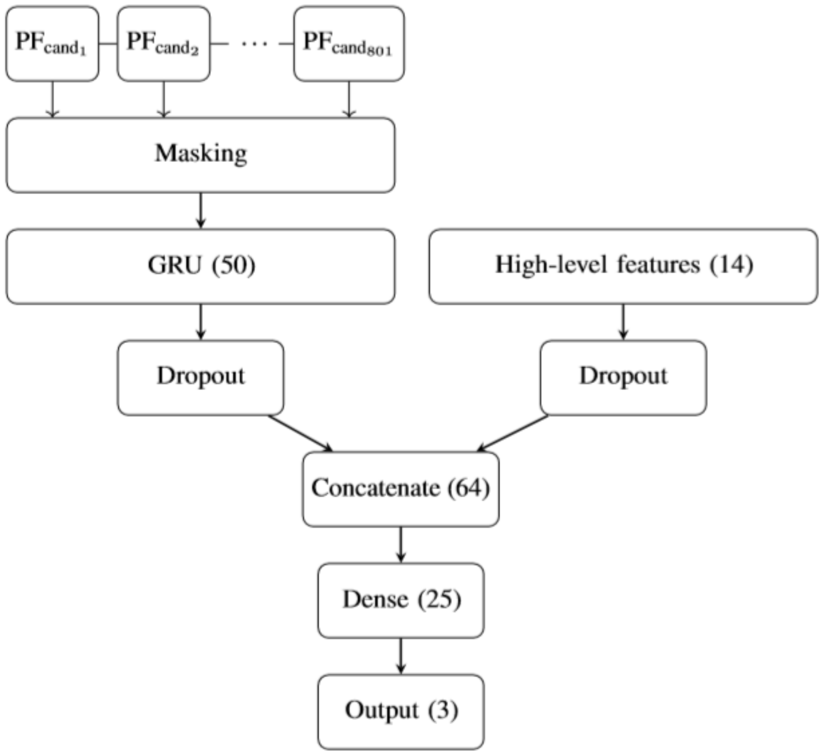
```
creating: createZooKerasSequential
creating: createZooKerasDense
creating: createZooKerasDense
creating: createZooKerasDense
creating: createZooKerasDense
```



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'))
```



Distributed Training

Instantiate the estimator using Analytics Zoo / BigDL

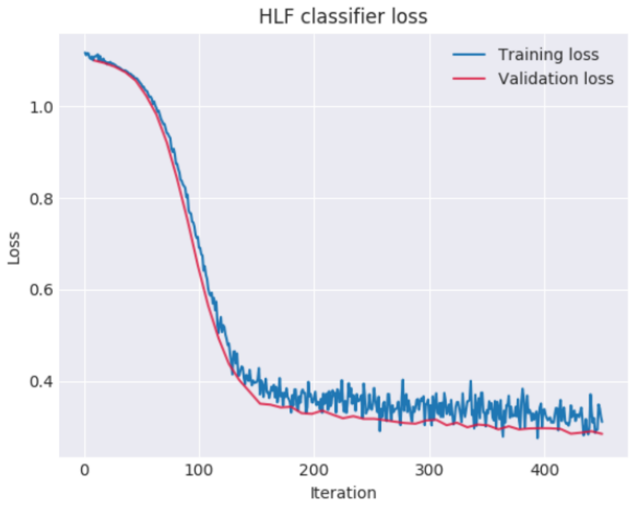
```
# Create SparkML compatible estimator for deep learning training

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)
```

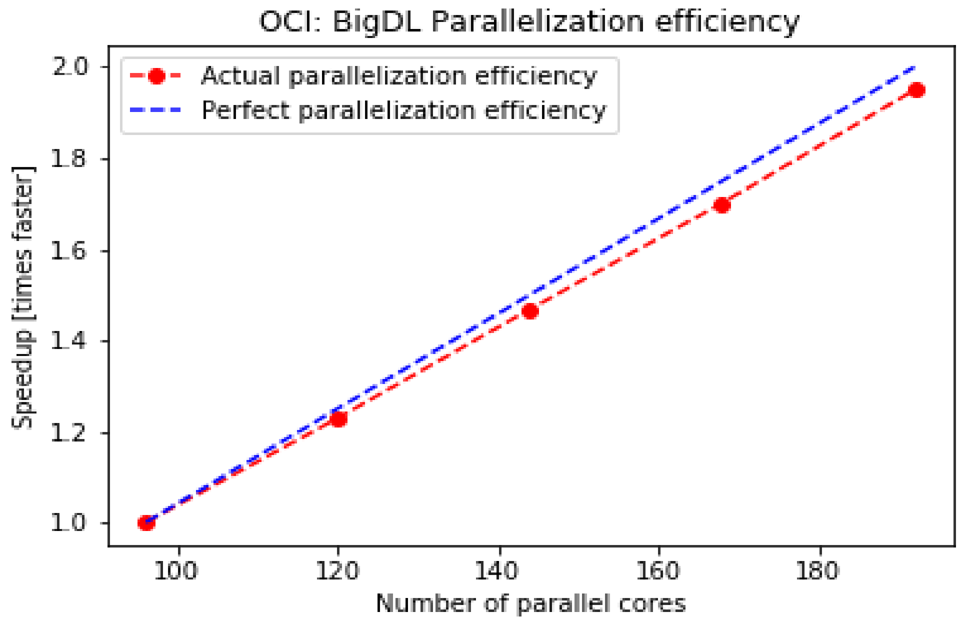
Storing the model for later use

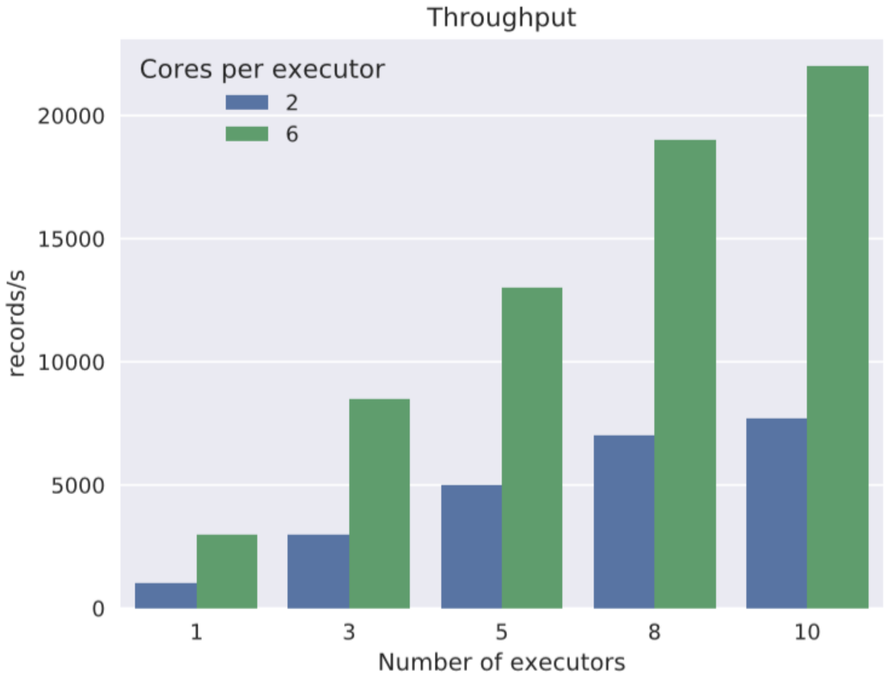
```
modelDir = logDir + '/nnmodels/HLFClassifier'
trained_model.save(modelDir)
```



Performance and Scalability of Analytics Zoo & BigDL

Analytics Zoo & BigDL scales very well in the ranges tested

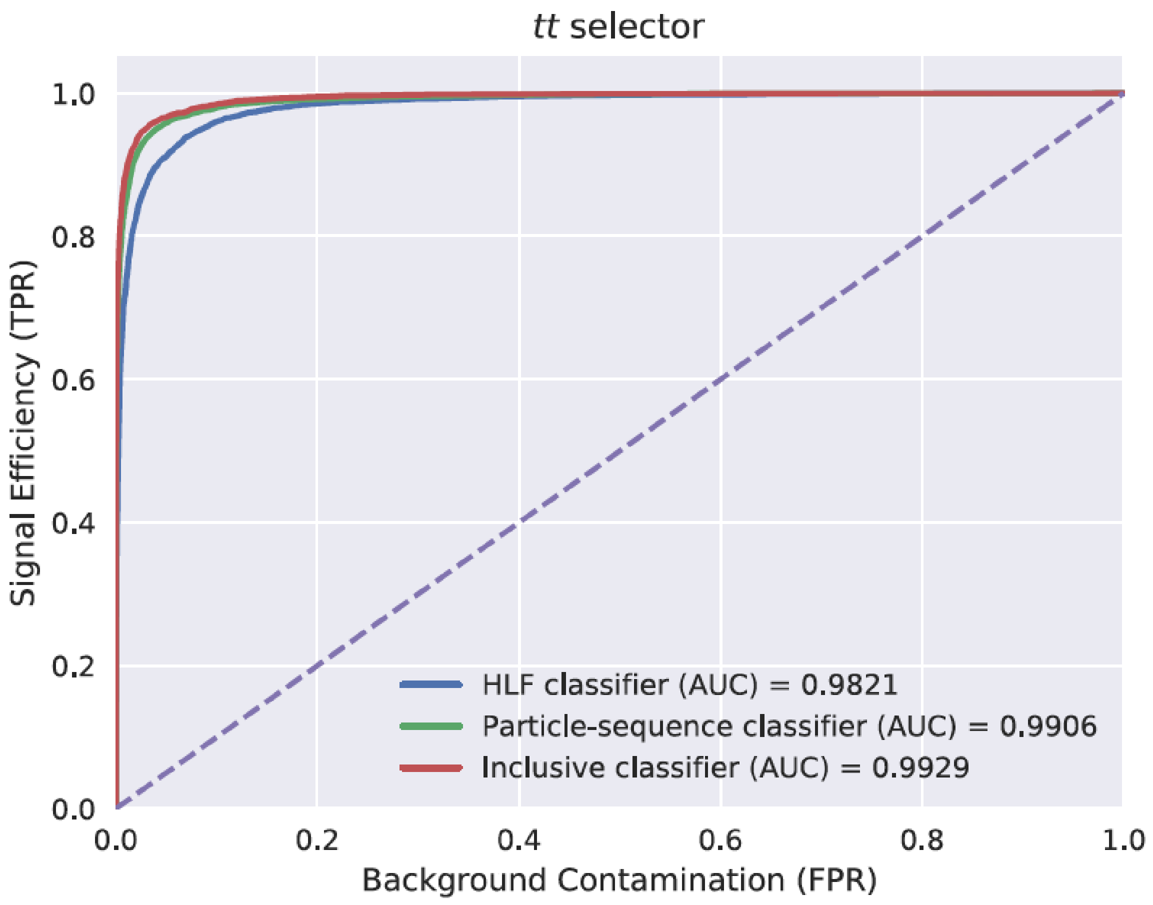




Results

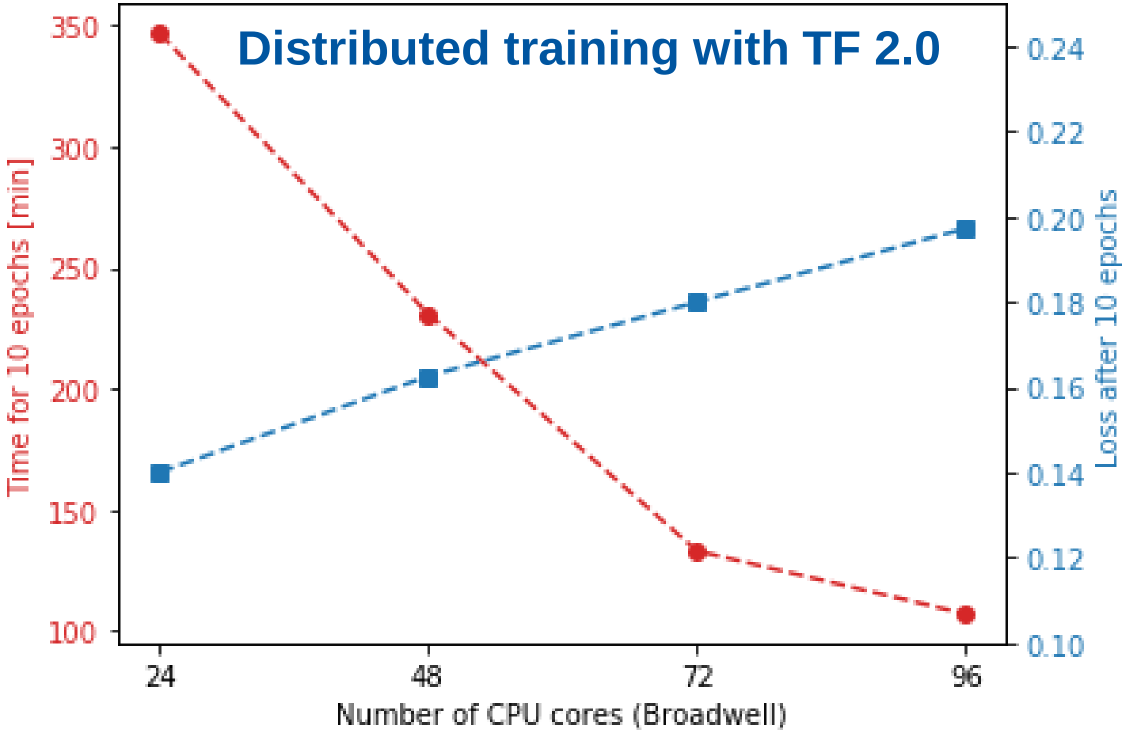
- Trained models with Analytics Zoo and BigDL

- Met the expected accuracy 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

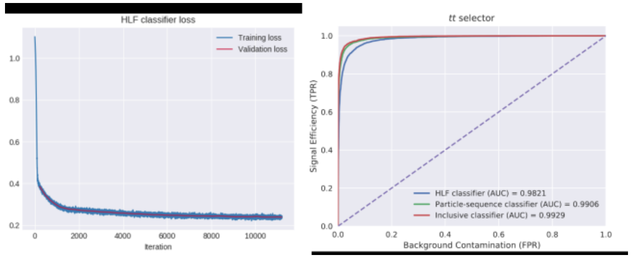
Data and models from Researchers

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.

Acknowledgments

- CERN openlab: Riccardo Castellotti, Michał Bień, Viktor Khristenko, Maria Girone
- CERN Spark and Hadoop service
- CMS, Bigdata team: Matteo Cremonesi, Jim Pivarski
- CMS, University of Padova: Matteo Migliorini, Marco Zanetti
- Intel team for BigDL and analytics Zoo: Jiao (Jennie) Wang, Sajan Govindan

- 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