

Deep learning at CERN's Large Hadron Collider with Analytics Zoo & BigDL

March 28th, 2019


Riccardo Castellotti, Matteo Migliorini, Luca Canali, Maria Girone



CERN

“Science for peace”

- International organisation close to Geneva, straddling Swiss-French border, founded 1954
- Facilities for fundamental research in particle physics
- 23 member states, 1.1 B CHF budget
- More than 3'000 staff, fellows, apprentices, ...
- About 15'000 associates



1954: 12 Member States

Members: Austria, Belgium, Bulgaria, Czech republic, Denmark, Finland, France, Germany, Greece, Hungary, Israel, Italy, Netherlands, Norway, Poland, Portugal, Serbia, Slovak Republic, Spain, Sweden, Switzerland, United Kingdom

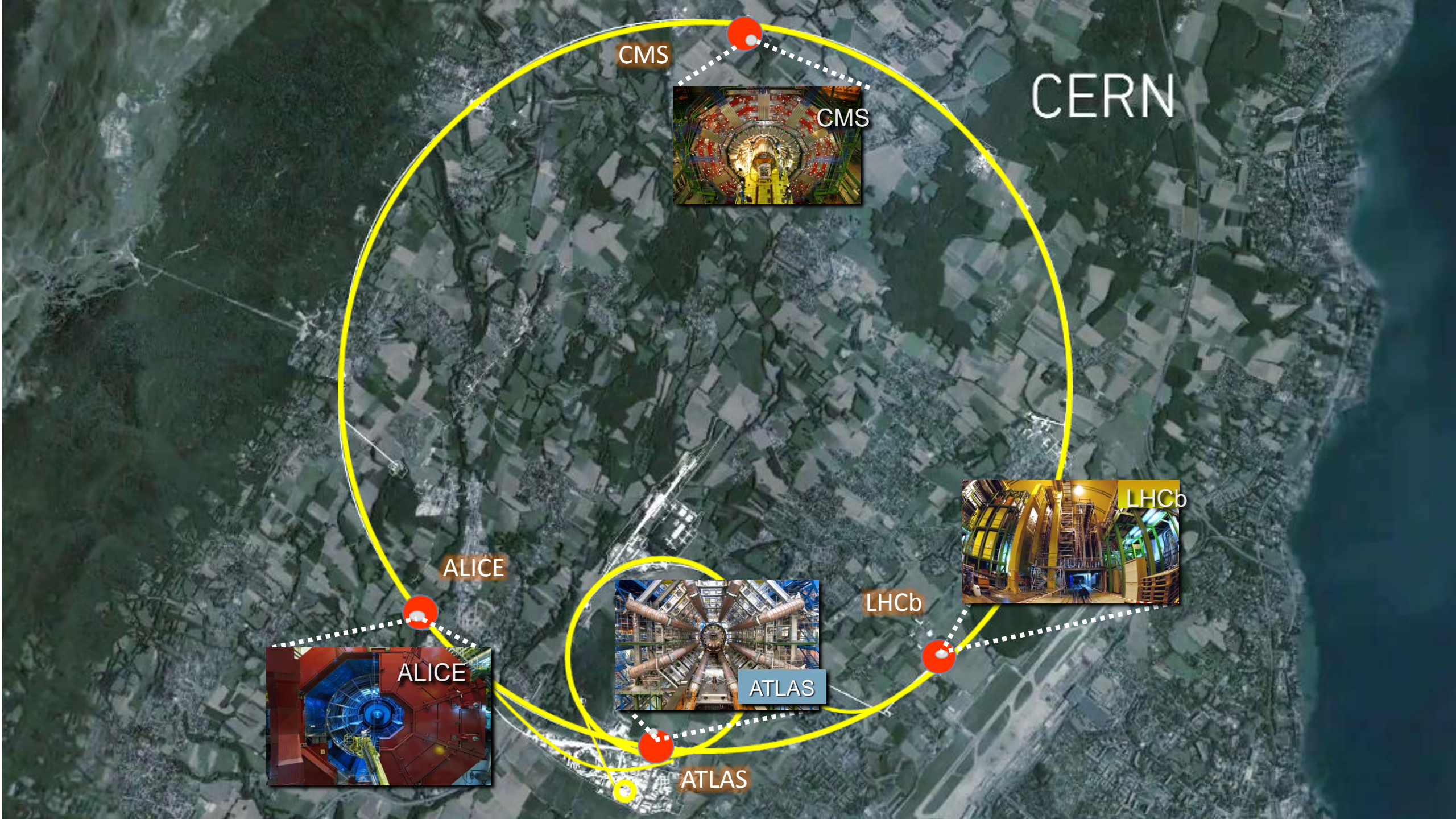
Candidate for membership: Cyprus, Slovenia

Associate members: India, Lithuania, Pakistan, Turkey, Ukraine (Croatia)

Observers: EC, Japan, JINR, Russia, UNESCO, United States of America

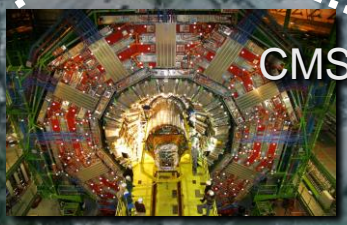
Numerous non-member states with collaboration agreement





CERN

CMS



CMS

ALICE



ALICE



ATLAS

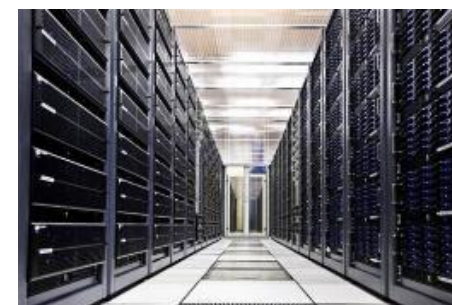
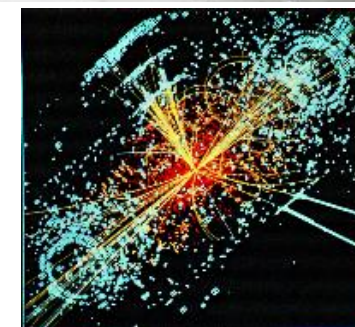
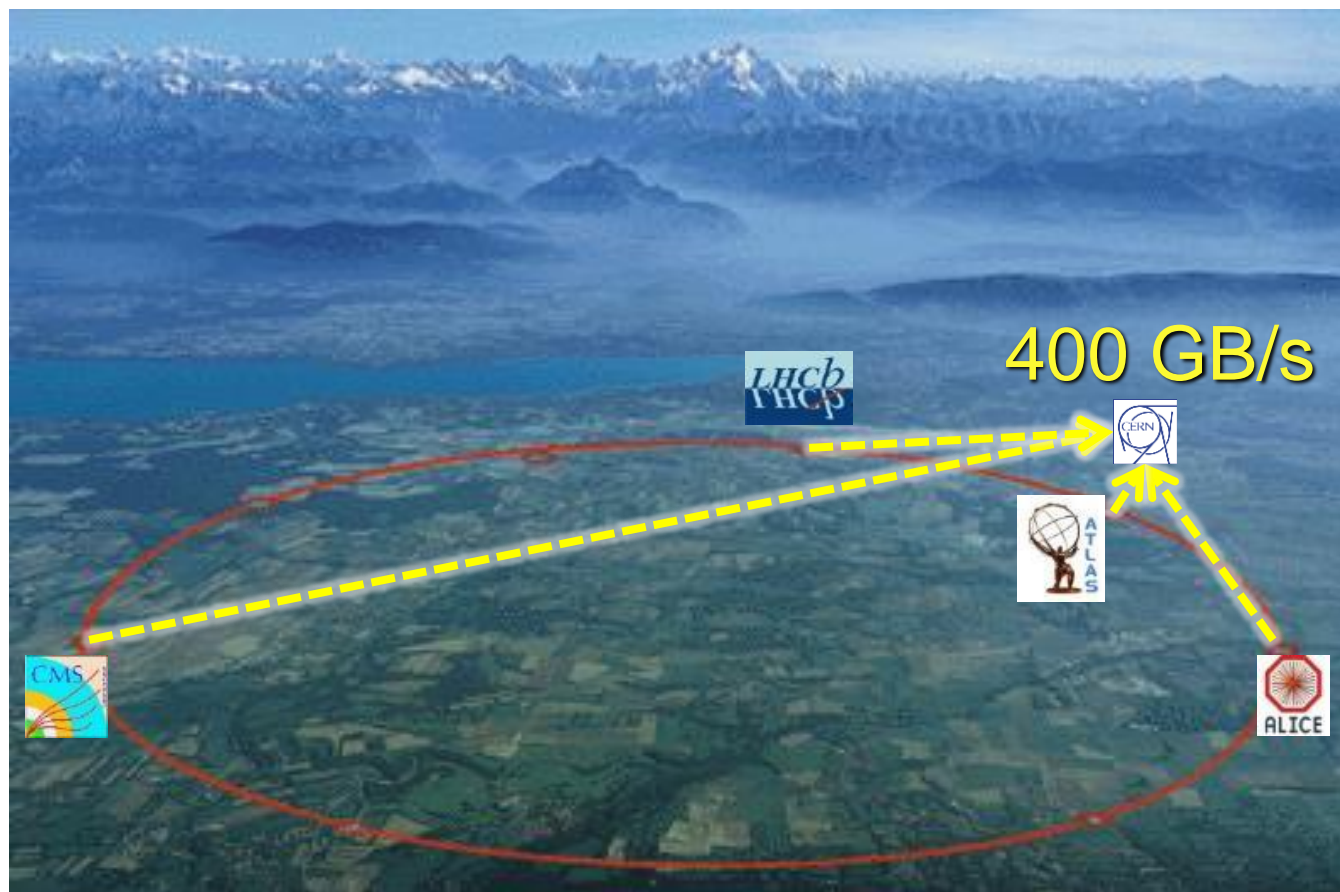
ATLAS

LHCb



LHCb

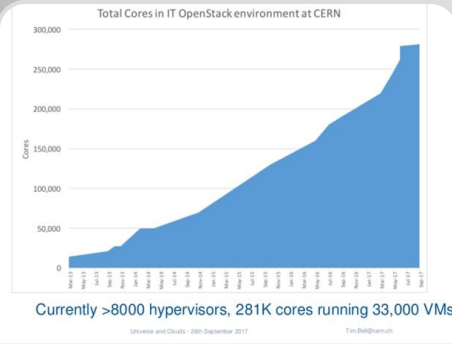
Storage, Reconstruction, Simulation, Distribution



The CERN Data Centre in Numbers



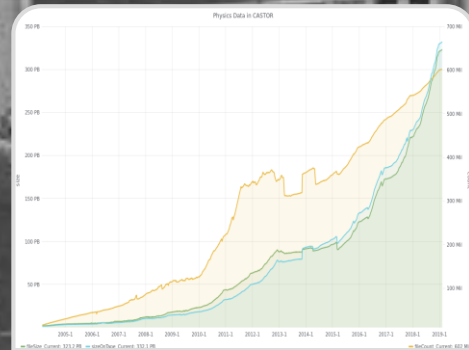
15 000
Servers



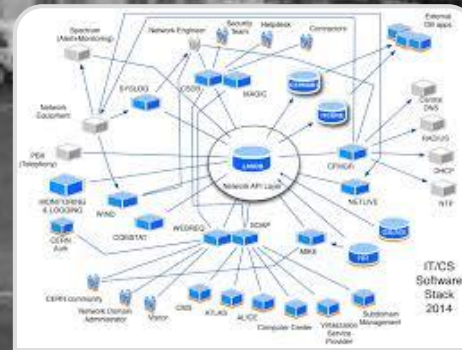
280 000
Cores



280 PB Hot
Storage



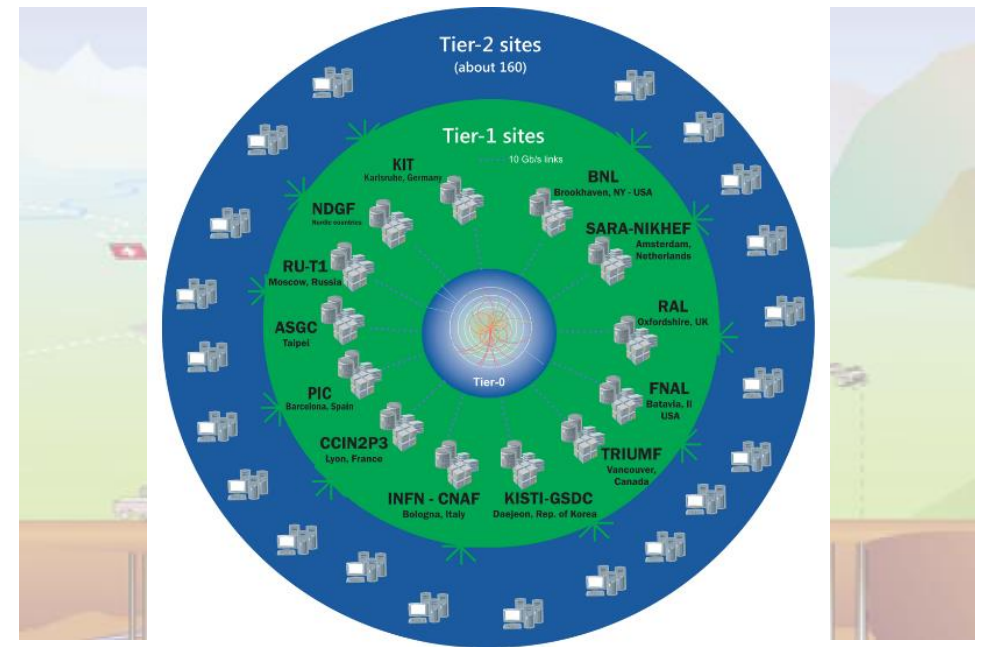
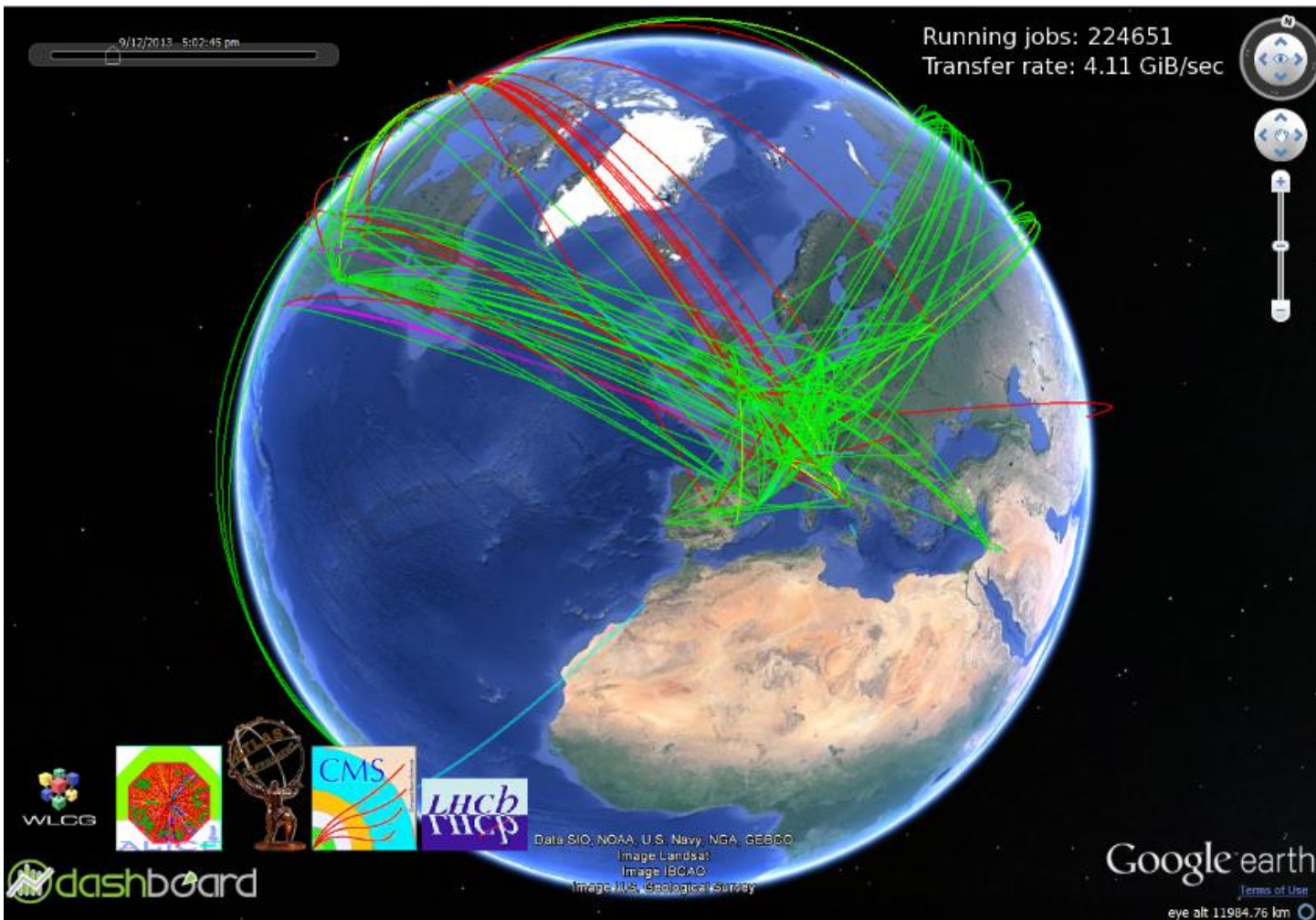
350 PB Cold
Storage



35 000 km
Fiber Optics



Worldwide LHC Computing Grid



Tier-0 (CERN):

- Data recording
- Initial data reconstruction
- Data distribution

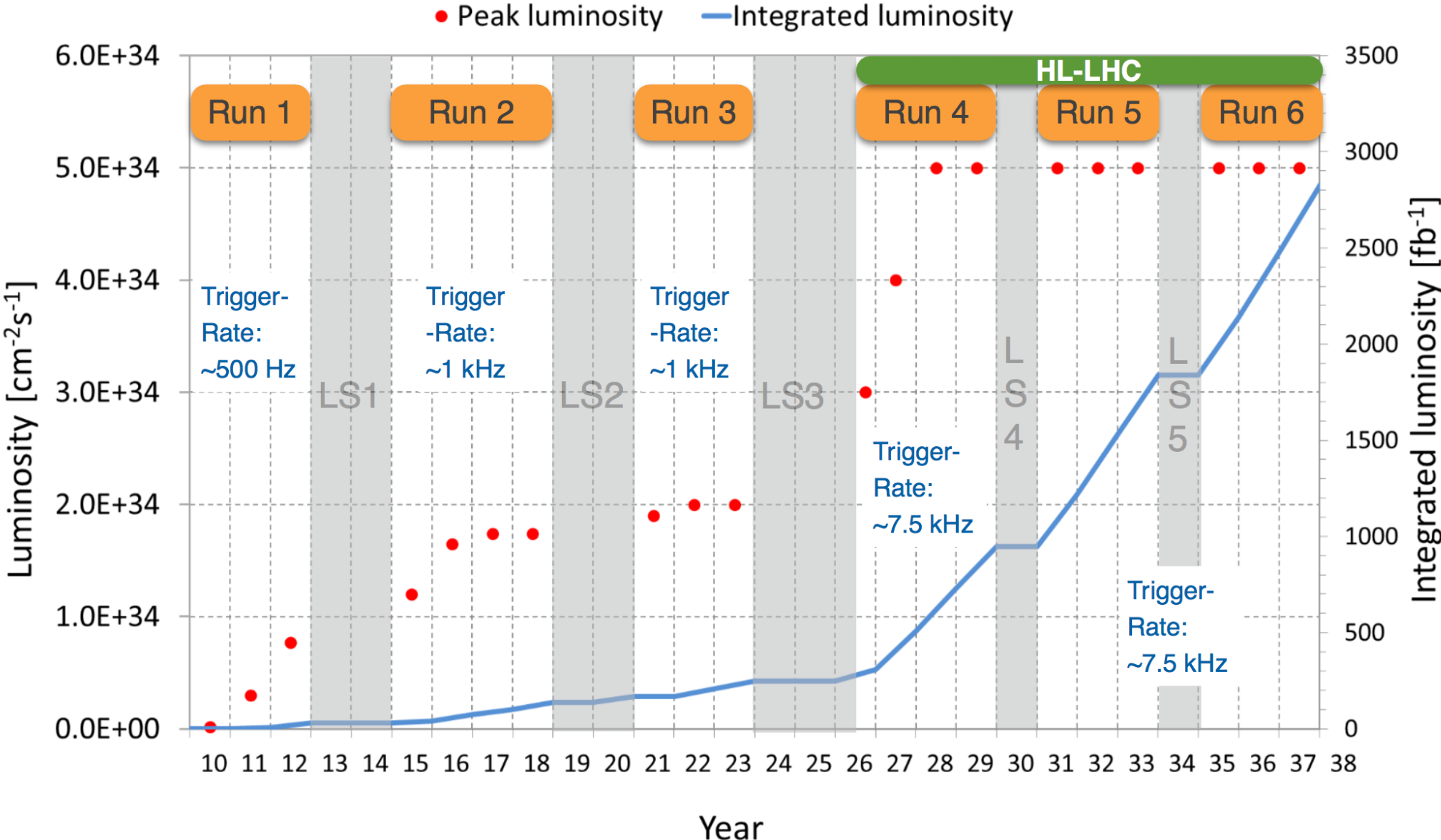
Tier-1 (14 centres):

- Permanent storage
- Re-processing
- Analysis

Tier-2 (72 Federations, ~150 centres):

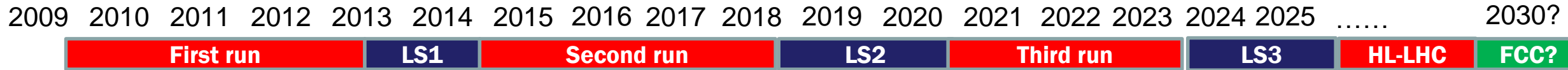
- Simulation
- End-user analysis
- 760,000 cores
- 700 PB

LHC Schedule



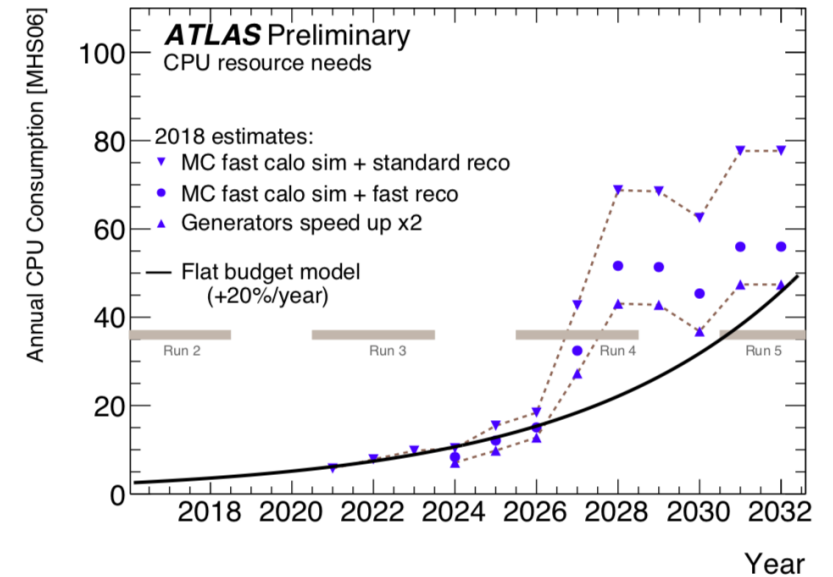
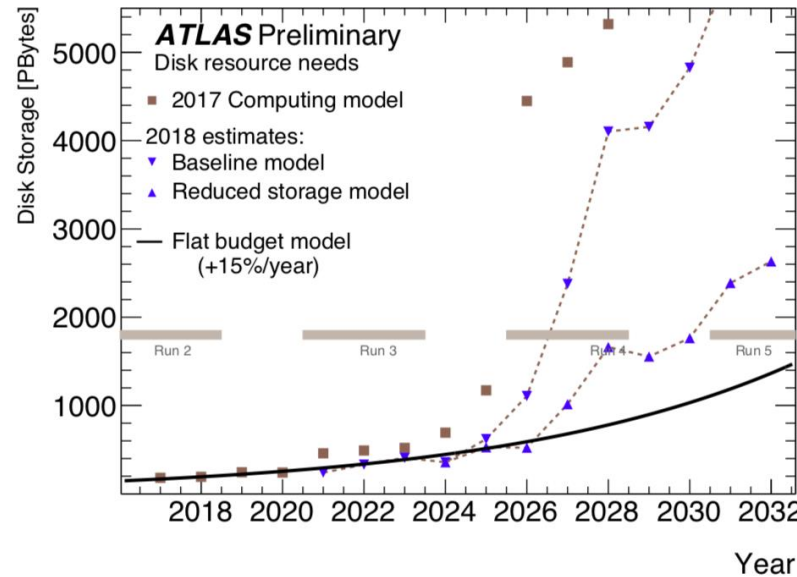
LHC Run3 and Run4

Scale and Challenges



Raw data volume for LHC increases exponentially and with it processing and analysis load

Estimates of resource needs at HL-LHC are factors above what is realistic to expect from technology with reasonably constant cost



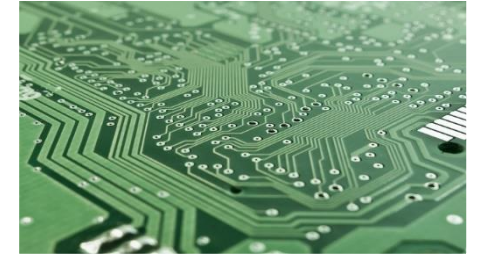
Technology revolutions are needed

Three Main Areas of R&D



COMPUTING
CHALLENGES

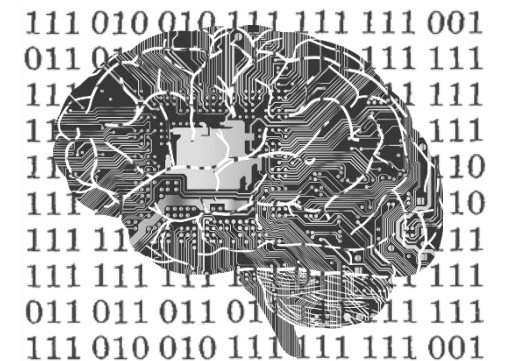
Increase **data centre performance** with hardware accelerators (FPGAs, GPUs, ..) optimized software



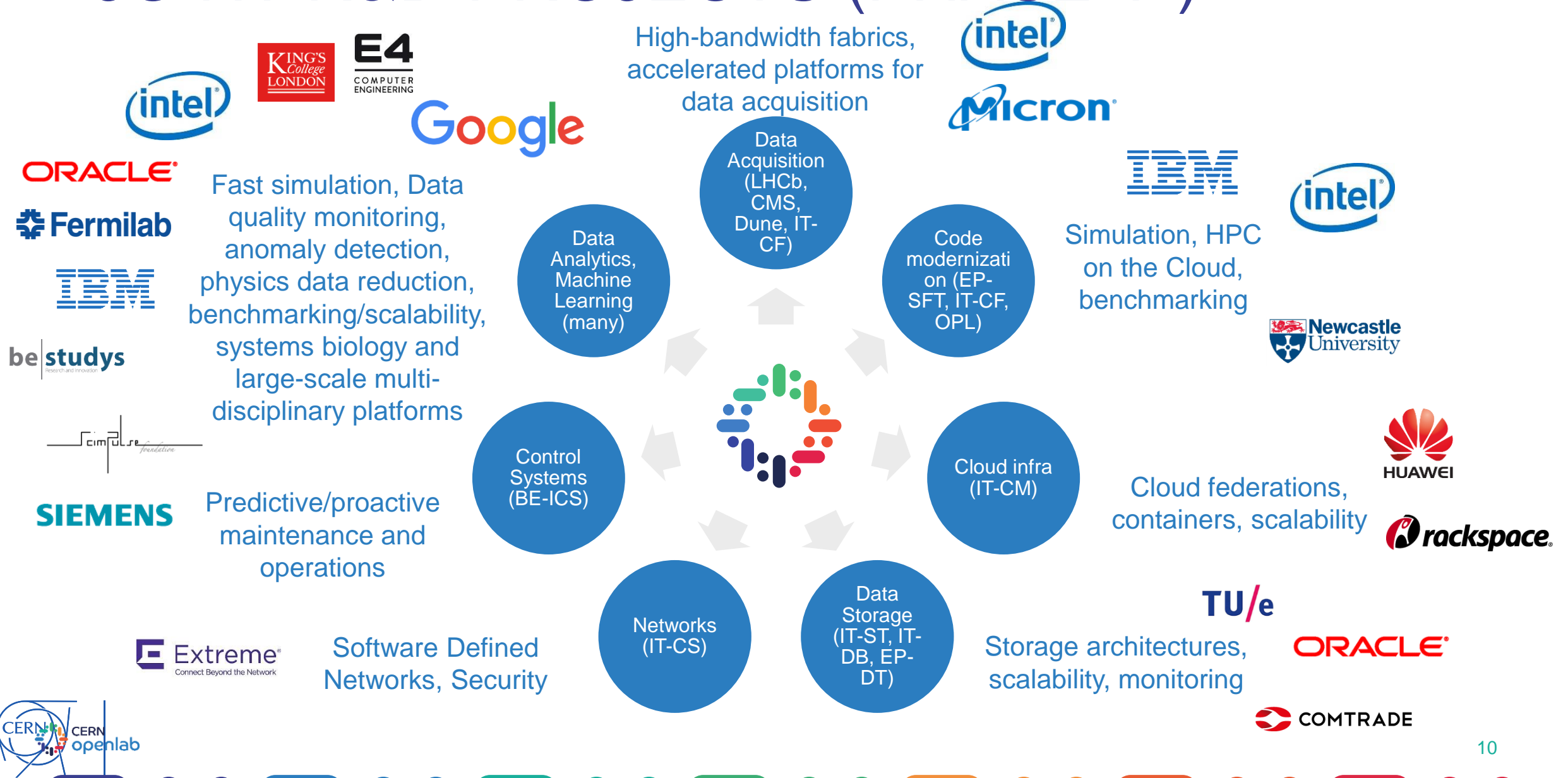
Scale out capacity with public clouds, HPC, new architectures



Change the computing paradigms with new technologies like **Machine Learning, Deep Learning, Advanced Data Analytics, Quantum Computing**



JOINT R&D PROJECTS (PHASE VI)



Hadoop, Spark and Kafka Service at CERN

- Setup and run the infrastructure for scale-out analytics solutions
- Today primarily for the components from Apache Hadoop framework and Big Data Ecosystem
- Support user community
 - Provide consultancy
 - Ensure knowledge sharing
 - Train on the technologies
 - Build the community



Analytics Pipelines – Use Cases

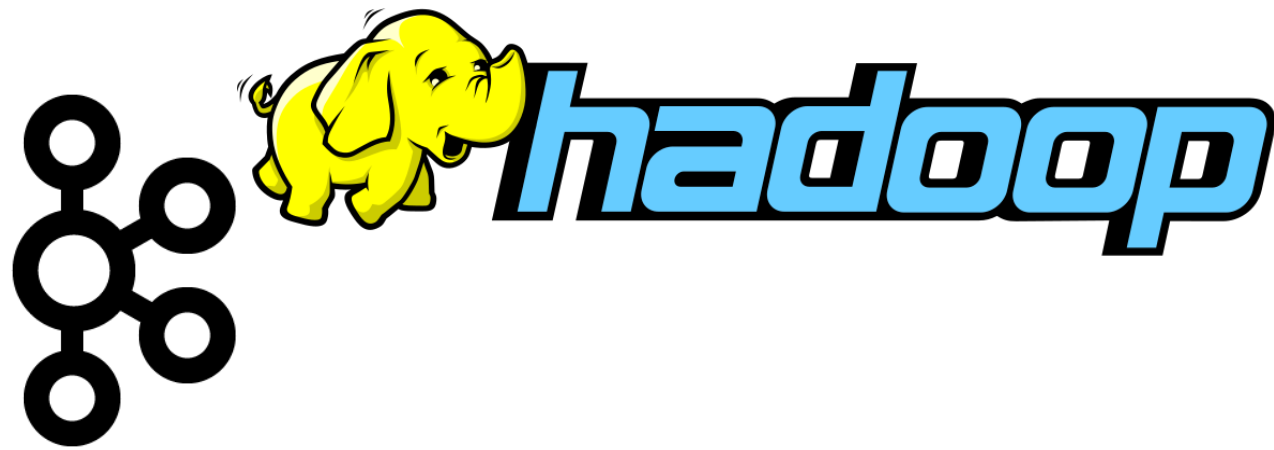
- Many use cases at CERN for analytics
 - Data analysis, dashboards, plots, joining and aggregating multiple data, libraries for specialized processing, machine learning, ...
- Communities
 - **Physics:**
 - Analysis on computing metadata (e.g. studies of popularity, grid jobs, etc) (CMS, ATLAS)
 - Development of new ways to process **Physics** data, e.g.: data reduction and analysis with Spark-ROOT by CMS Bigdata project, Root team and TOTEM experiment
 - **IT:**
 - Analytics on IT monitoring data
 - Computer security
 - **BE (Accelerators):**
 - NX CALS – next generation accelerator logging platform
 - BE controls data and analytics
 - More:
 - Many tools provided in our platforms are popular and readily available, likely to attract **new** projects, notably the analytics platform with hosted notebooks **SWAN_Spark**
 - E.g. Starting investigations on data pipelines for IoT (Internet of Things)

“Big Data”: Not Only Analytics

- Data **analytics** is a key use case for the platforms
- **Deep Learning/AI** is integrating with data analytics and pipelines
- Scalable workloads and parallel **computing**
- **Database**-type workload also important
 - Use Big Data tools instead of RDBMS
- Data pipelines and **streaming**
 - Datacenter monitoring and Computer security
 - IoT

Highlights of “Big Data” Components

- Apache Hadoop clusters with YARN and HDFS
 - Also HBase, Impala, Hive,...
- Apache Spark for analytics
- Apache Kafka for streaming
- Data: Parquet, JSON, ROOT
- UI: Python notebooks



Hadoop and Spark production deployment

✧ Software distribution

- ✧ Cloudera (since 2013)
- ✧ Vanilla Apache (since 2017)



✧ Rolling change deployment

- ✧ no service downtime
- ✧ transparent in most of the cases



✧ Installation and configuration

- ✧ CERN CentOS 7.6
- ✧ custom Puppet module



✧ Host monitoring and alerting

- ✧ via CERN IT Monitoring infrastructure



✧ Security

- ✧ authentication Kerberos
- ✧ fine-grained authorization integrate with e-groups



✧ Service level monitoring

- ✧ metrics integrated with: Elastic + Grafana
- ✧ custom scripts for availability and alerting



✧ High availability

- ✧ automatic master failover for HDFS, YARN and HBASE



✧ HDFS backups

- ✧ Daily incremental snapshots
- ✧ Sent to tapes (CASTOR)



Hadoop and Spark Clusters

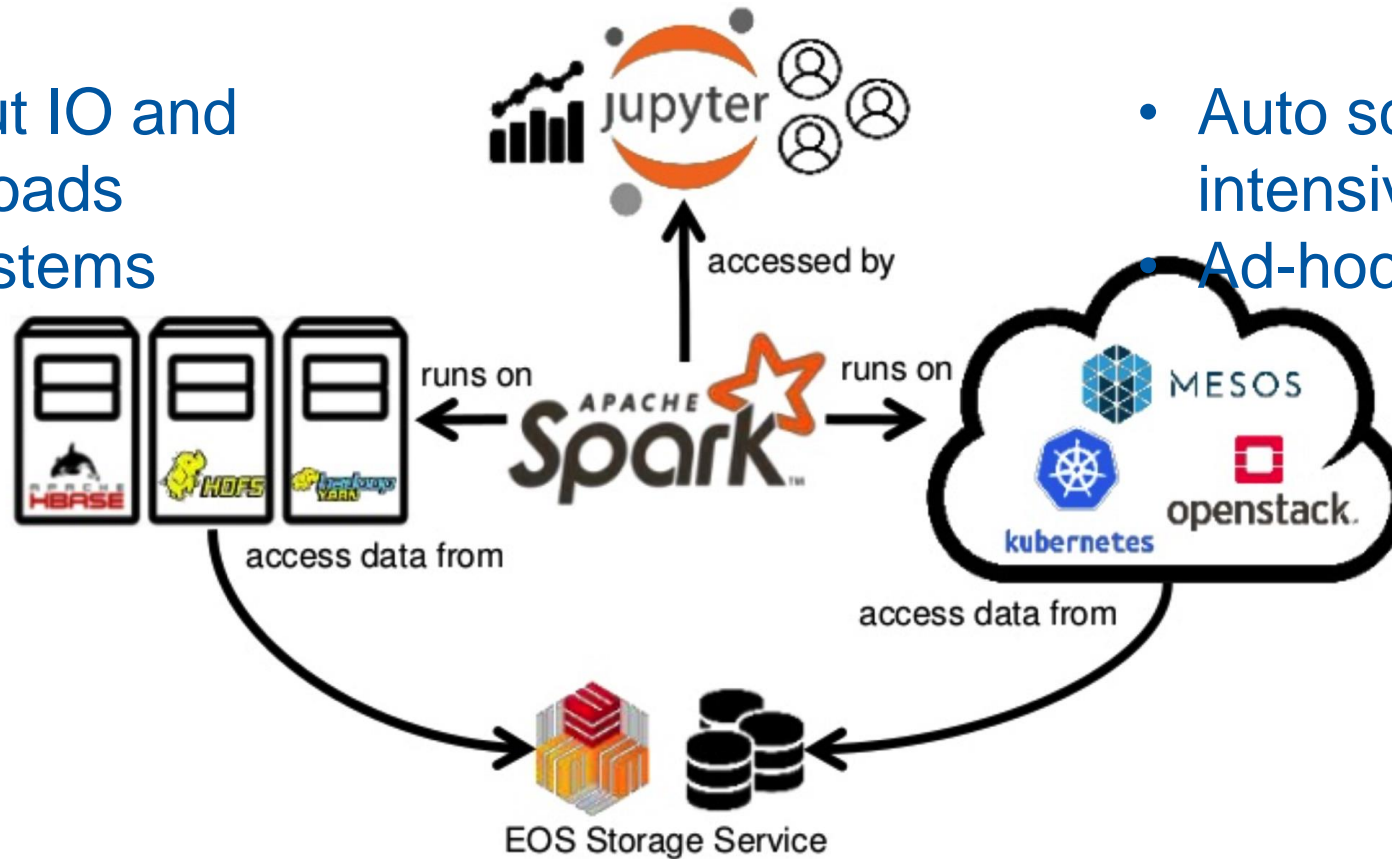
- Clusters: **YARN**/Hadoop and Spark on **Kubernetes**
- Hardware: Intel based servers, continuous refresh and capacity

Cluster Name	Configuration	Cluster type
Accelerator logging, NXCALS	20 nodes (Cores - 480, Mem - 8 TB, Storage – 5 PB) Upgrades in Q2 2019: will add 10 nodes	YARN/hadoop_cern
General Purpose	52 nodes (Cores – 900, Mem – 14 TB, Storage – 9 PB)	YARN/CDH
Development cluster	12 nodes (Cores – 196, Mem – 800 GB, Storage – 2 PB)	YARN/CDH
ATLAS Event Index	18 nodes (Cores – 288, Mem – 900 GB, Storage – 1.3 PB)	YARN/CDH
QA cluster	10 nodes	YARN/hadoop_cern
Cloud containers	60 VMs (Cores - 240, Mem – 480 GB) Notes: Storage is external (HDFS, EOS, S3/Ceph) + cluster can be easily grown and shrunk depending on needs	Spark on Kubernetes

Analytics platform outlook

- High throughput IO and compute workloads
- Established systems

- Auto scale for compute intensive workloads
- Ad-hoc users





SWAN – Jupyter Notebooks On Demand

- SWAN - Service for Web based ANalysis
 - Developed at CERN, provides Jupyter notebooks on demand with relevant CERN integration for data and compute
- Fully integrated with Spark and Hadoop clusters at CERN
 - Python on Spark (PySpark) at scale
 - Modern, powerful and scalable platform for data analysis
 - Web-based: no need to install any software
- <https://cern.ch/swan>

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()
```

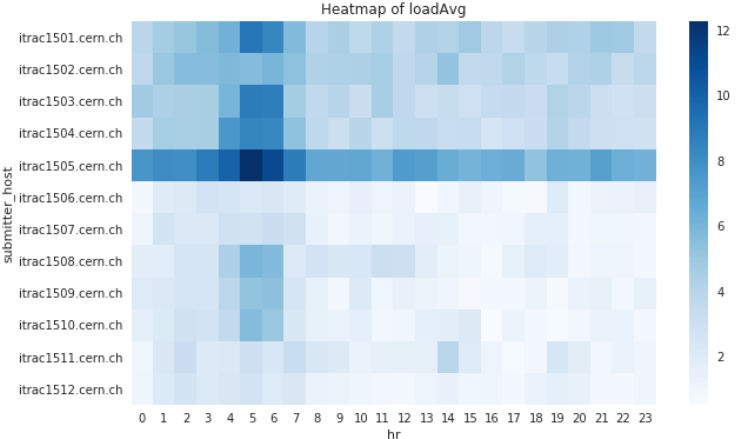
Apache Spark: 90 EXECUTORS 180 CORES Jobs: 1 COMPLETED

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')



Heatmap of loadAvg

Text

Code

Monitoring

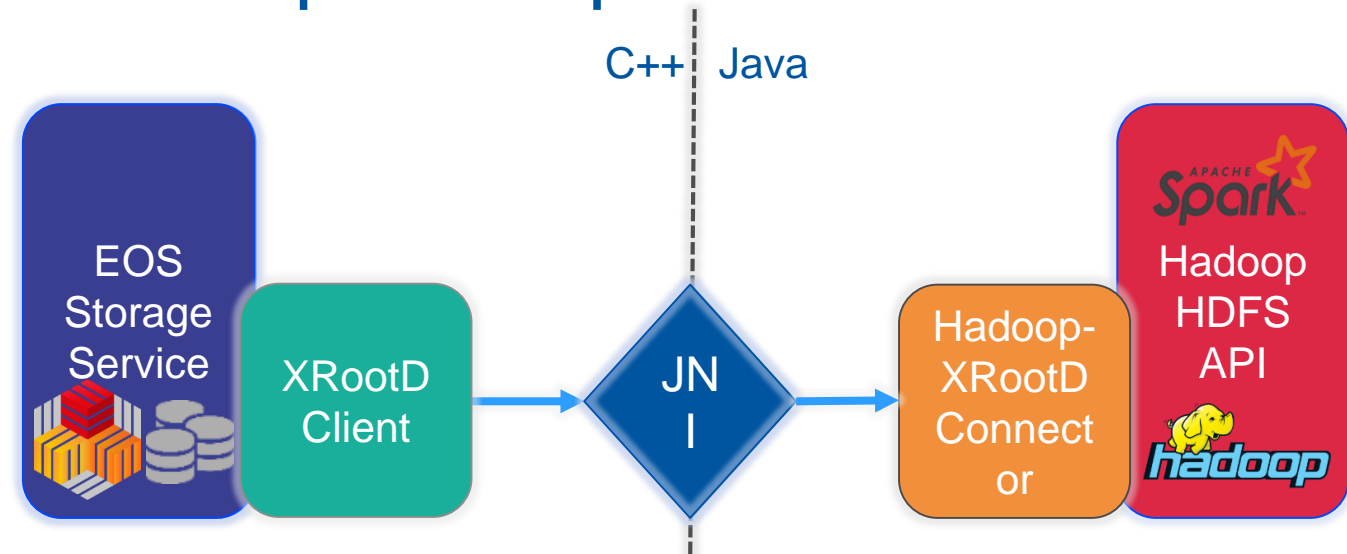
VISUALIZATION

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

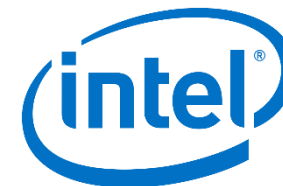
Extending Spark to read physics data

- 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
- +50 PBs per year of operation



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



Deep Learning Pipeline for Physics Data

Code at: <https://github.com/cerndb/SparkML>

Engineering Efforts to Enable Effective ML

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

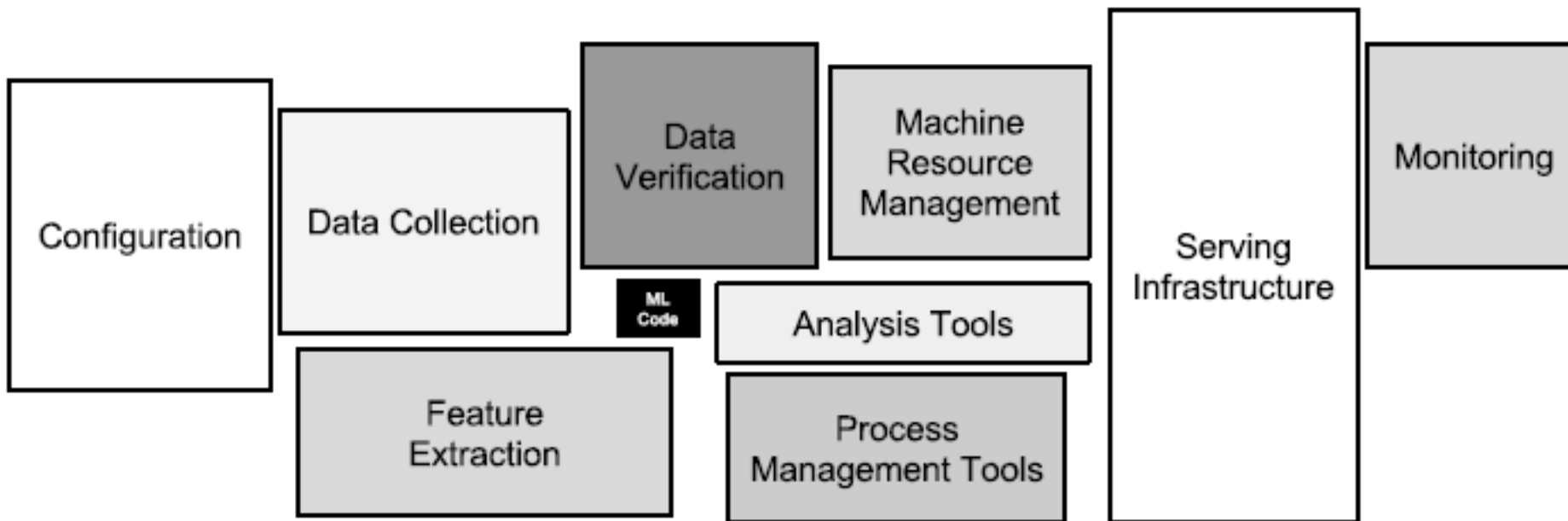


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

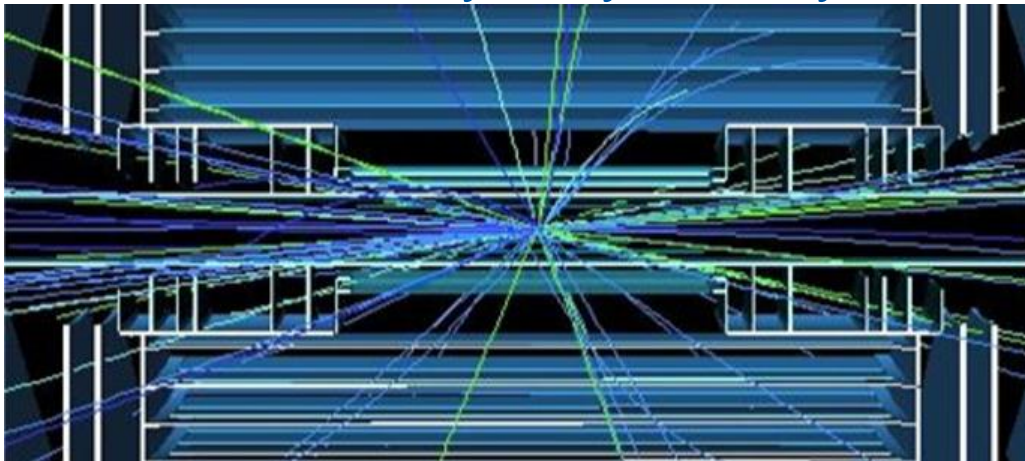
Analytics Zoo & BigDL

- 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



Data challenges in physics

- Proton-proton collisions in LHC happen at 40MHz.
 - Hundreds of TB/s of electrical signals that allow physicists to investigate particle collision events.
- Storage, limited by bandwidth
 - Currently, only 1 every 40K events stored to disk (~10 GB/s).



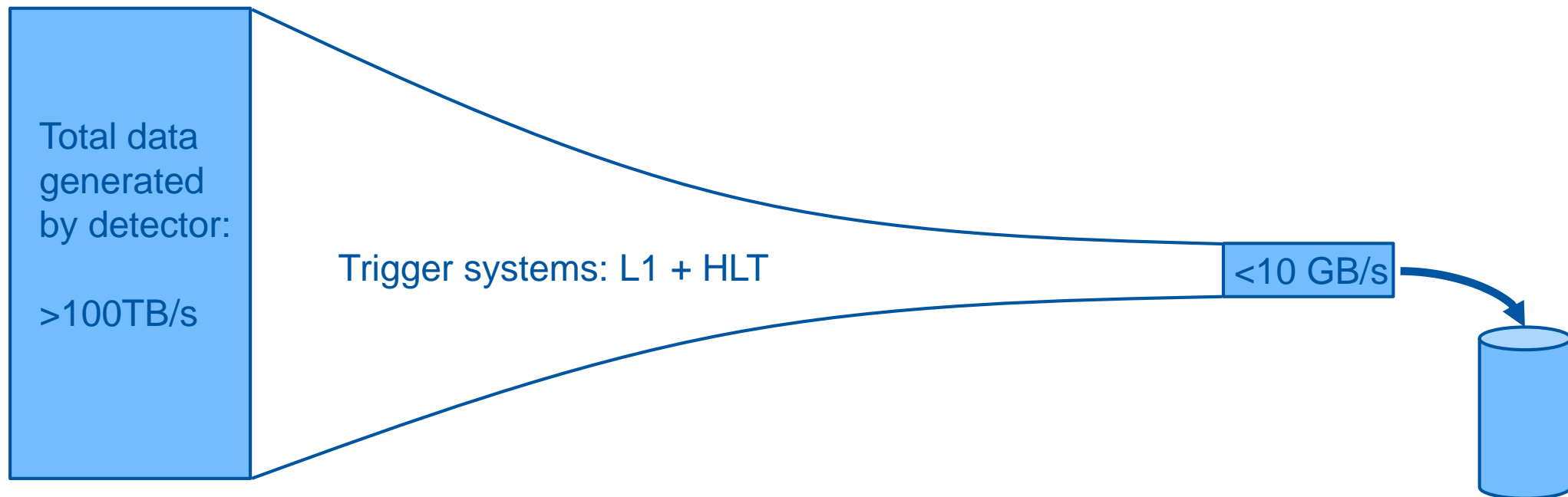
2018: 5 collisions/beam cross
LHC



2026: 400 collisions/beam cross
High Luminosity-LHC

Filtering

- How is the event filtering done (2018)?
- Two stages:
 - L1 trigger: 40 MHz \rightarrow 100 KHz ASICs/FPGAs rule-based algorithms
 - High Level trigger 100KHz \rightarrow 1KHz CPU farm

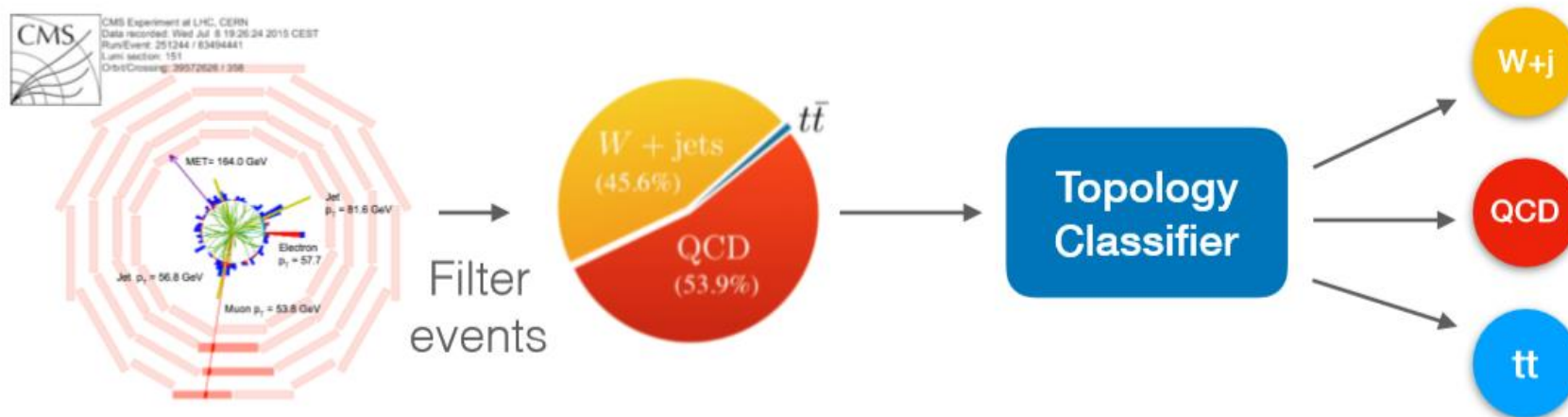


R&D

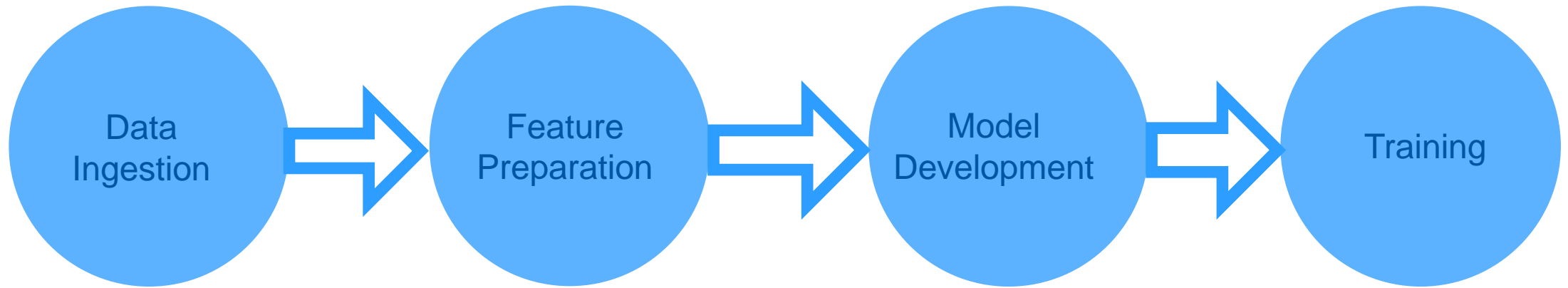
- **Improve the quality of filtering systems:** all the recorded events should be relevant for research
 - Moving from rule-based algorithms to Deep Learning classifiers
- Increase the analytics “at the edge”
 - Avoid wasting resources in offline computing
 - Reduction of operational costs
- Inference: very limited time budget for classification -> FPGAs

Particle classifier use case

- “Topology classification with deep learning to improve real-time event selection at the LHC”
[arXiv:1807.00083v2](https://arxiv.org/abs/1807.00083v2)



Data Pipeline



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

Leveraging Apache Spark and Analytics Zoo in 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'  
]
```

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)
- A sorting metric to create a sequence of particles to be fed to a sequence based classifier

Feature preparation

- All features need to be converted to a format consumable by the 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])

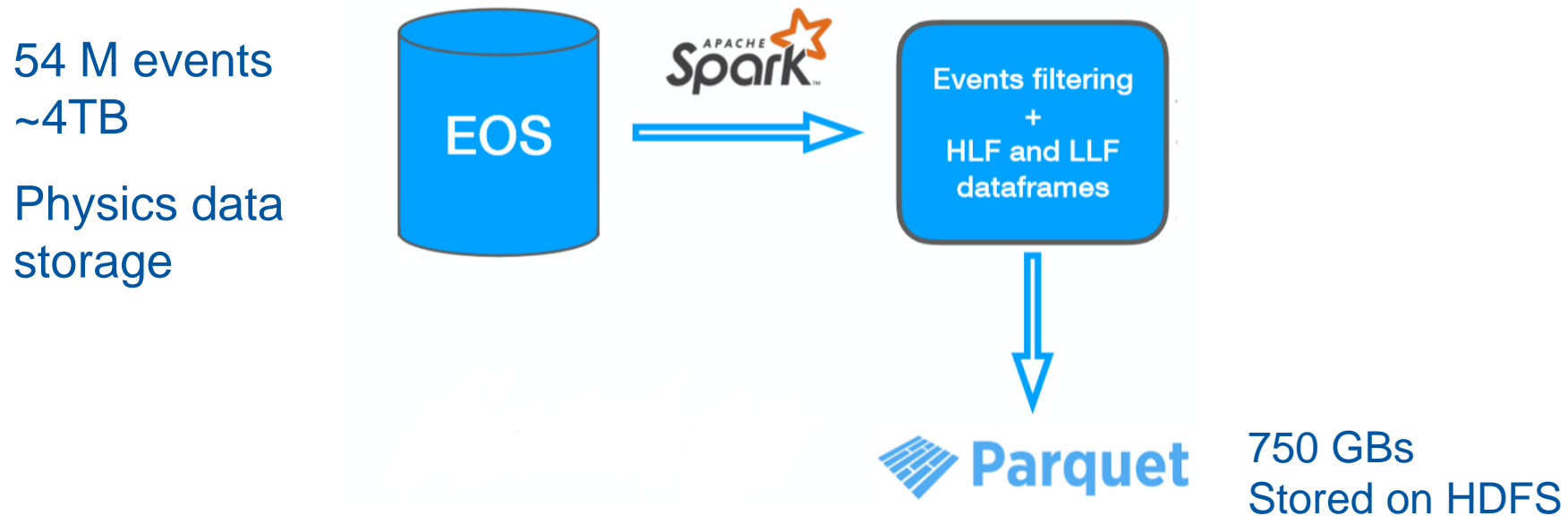
         %time fitted_pipeline = pipeline.fit(data)

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

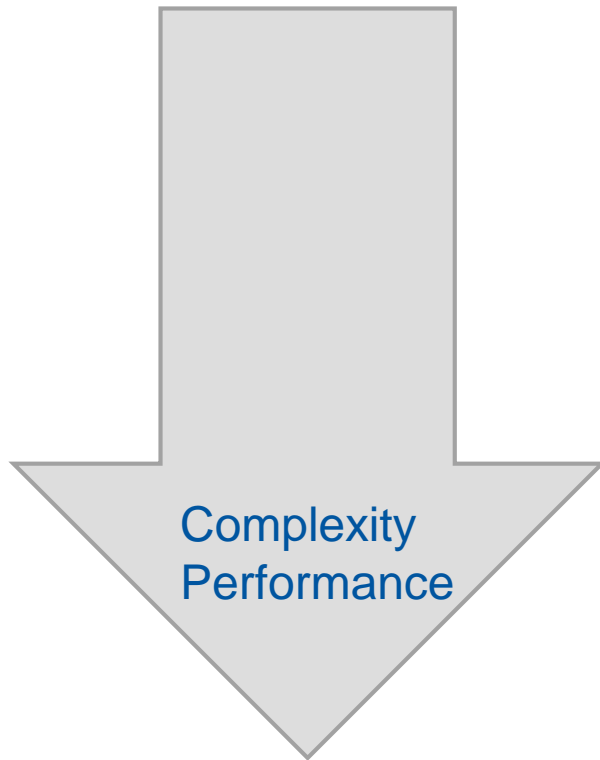
```
In [12]: data = fitted_pipeline.transform(data)
```

Data ingestion

- Read input files (4 TB) from custom format
- Compute physics-motivated features
- Store to parquet format



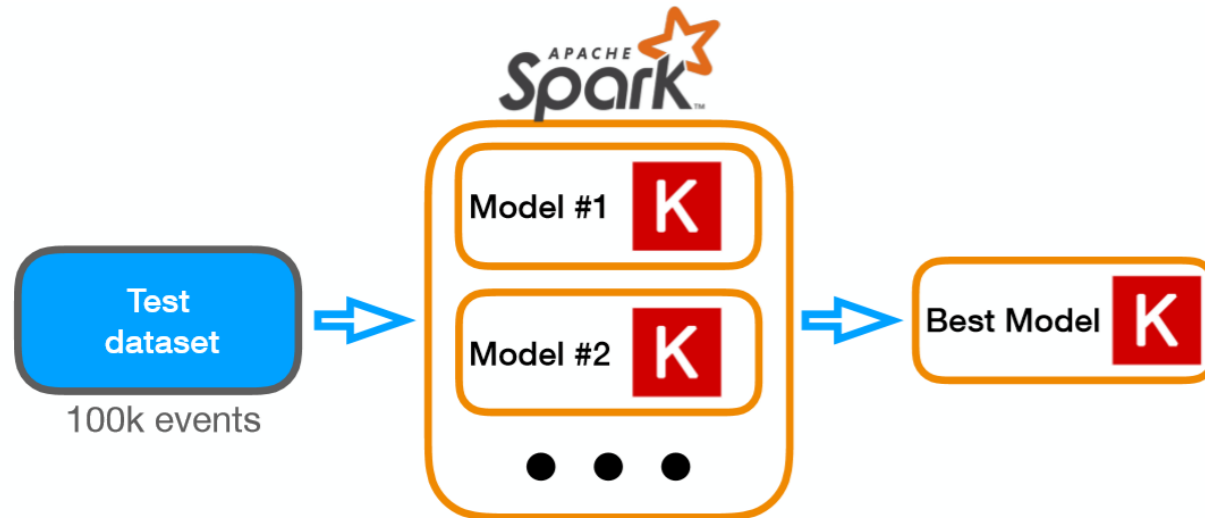
Models investigated



1. Fully connected feed-forward DNN
2. DNN with a recursive layer (typical for sequence-based problems such as Natural Language Processing)
3. Recursive DNN from 2. and feature engineering (domain specific)

Hyper-parameter tuning– DNN

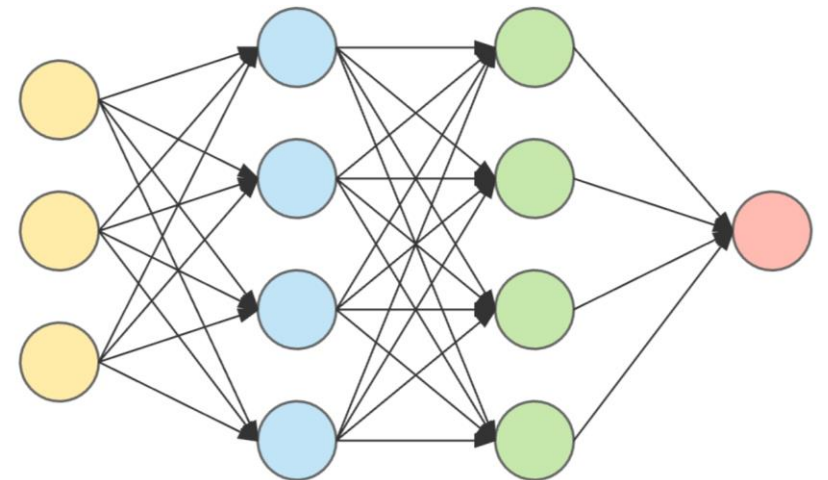
- Once the network topology is chosen, hyper-parameter tuning is done with scikit-learn+Keras and parallelized with Spark



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

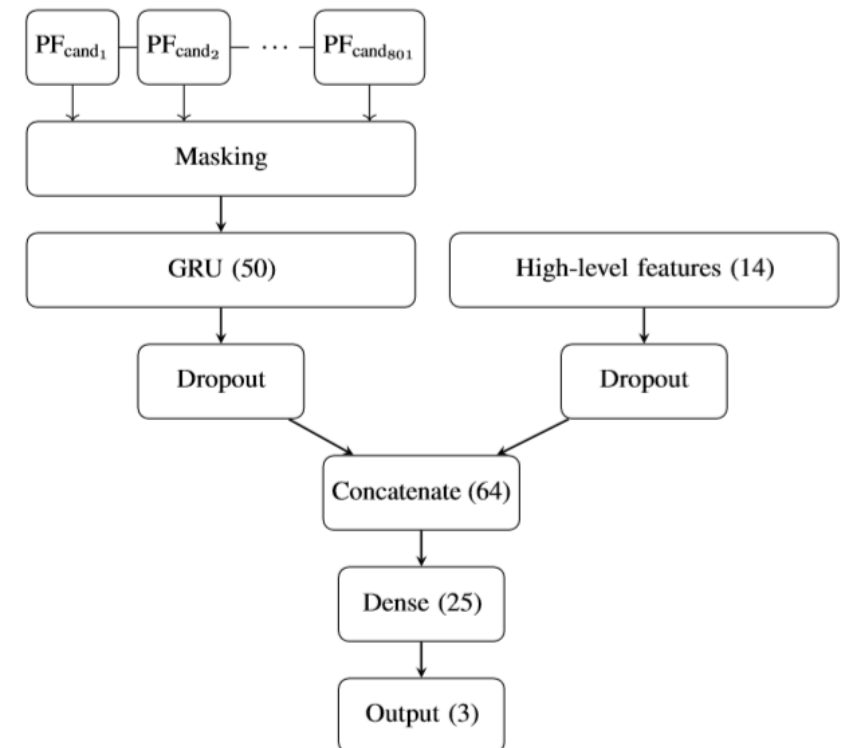
```
In [6]: from zoo.pipeline.api.keras.models import Sequential
from zoo.pipeline.api.keras.layers.core import *
from zoo.pipeline.api.keras.layers.torch import Select
from zoo.pipeline.api.keras.layers.normalization import BatchNormalization
from zoo.pipeline.api.keras.layers.recurrent import GRU
from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

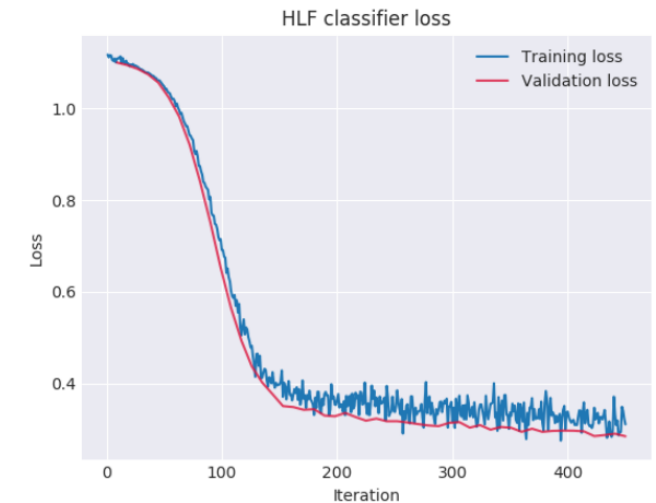
estimator = NNEstimator(model, CategoricalCrossEntropy())\
    .setOptimMethod(Adam()) \
    .setBatchSize(BDLbatch) \
    .setMaxEpoch(numEpochs) \
    .setFeaturesCol("HLF_input") \
    .setLabelCol("encoded_label") \
    .setValidation(trigger=EveryEpoch(), val_df=testDF,
                  val_method=[Loss(CategoricalCrossEntropy()),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

The actual training is distributed using Spark executors

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

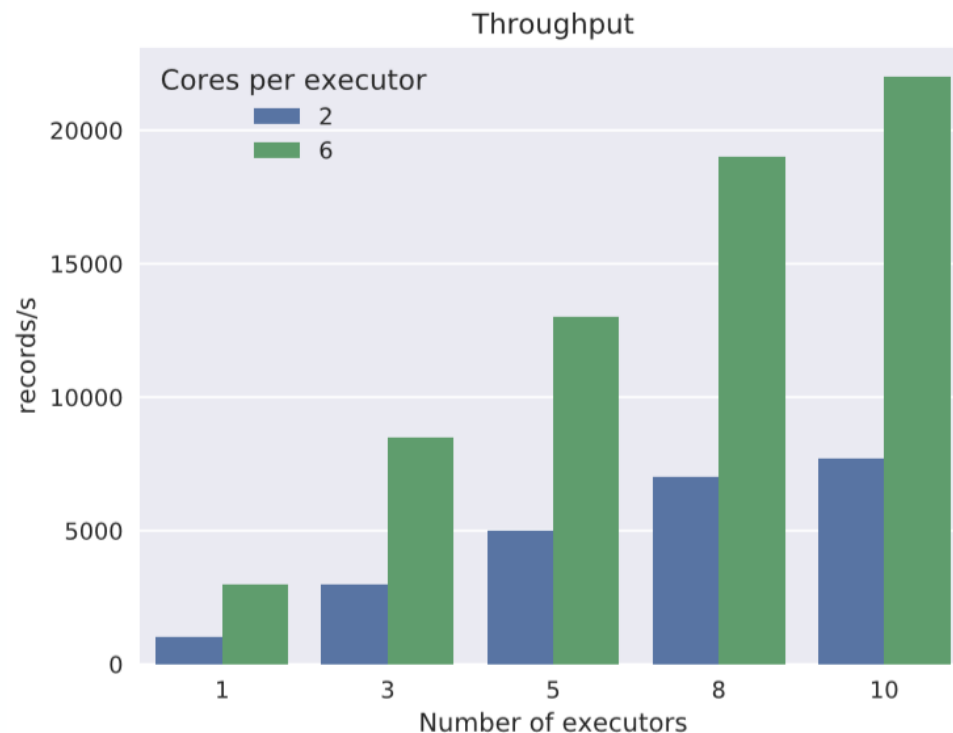
Storing the model for later use

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



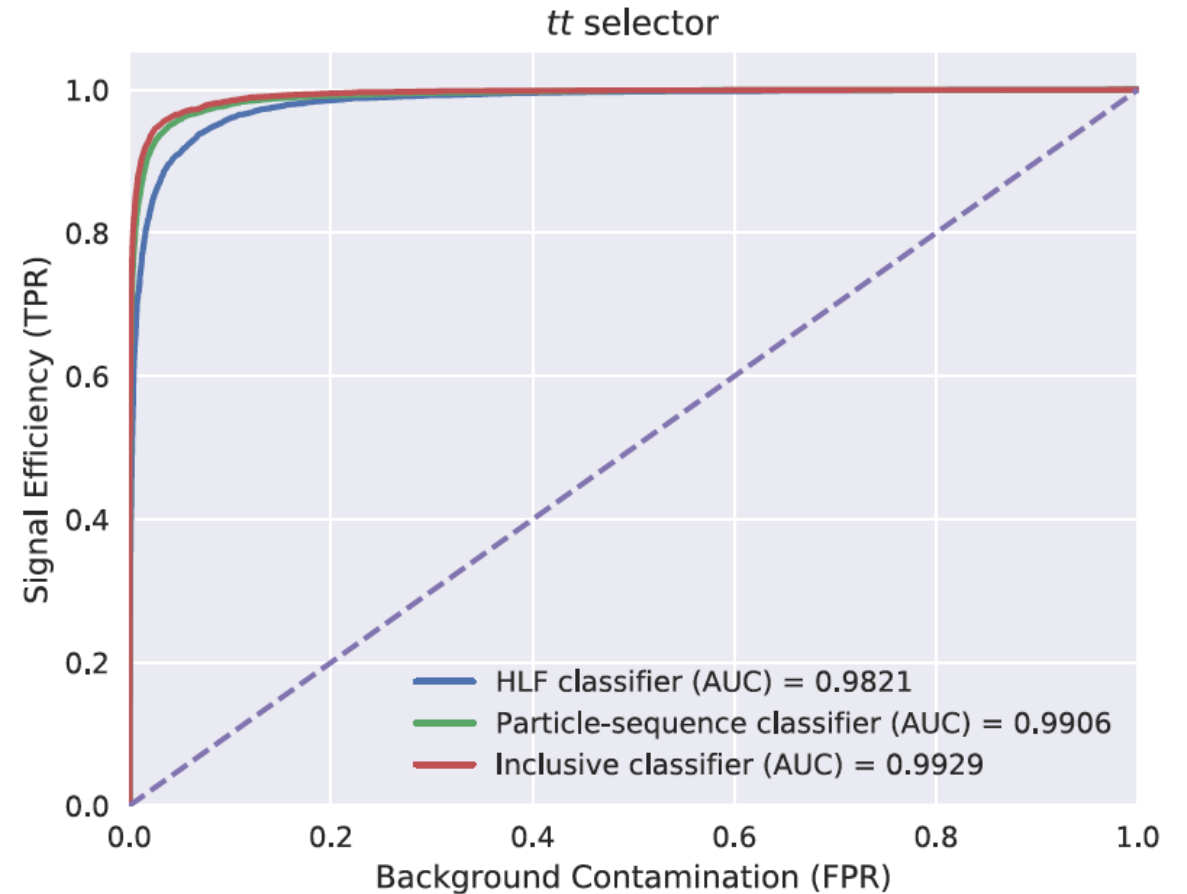
Performance and Scalability of Analytics Zoo & BigDL training

Analytics Zoo & BigDL scales very well in the range tested



Results

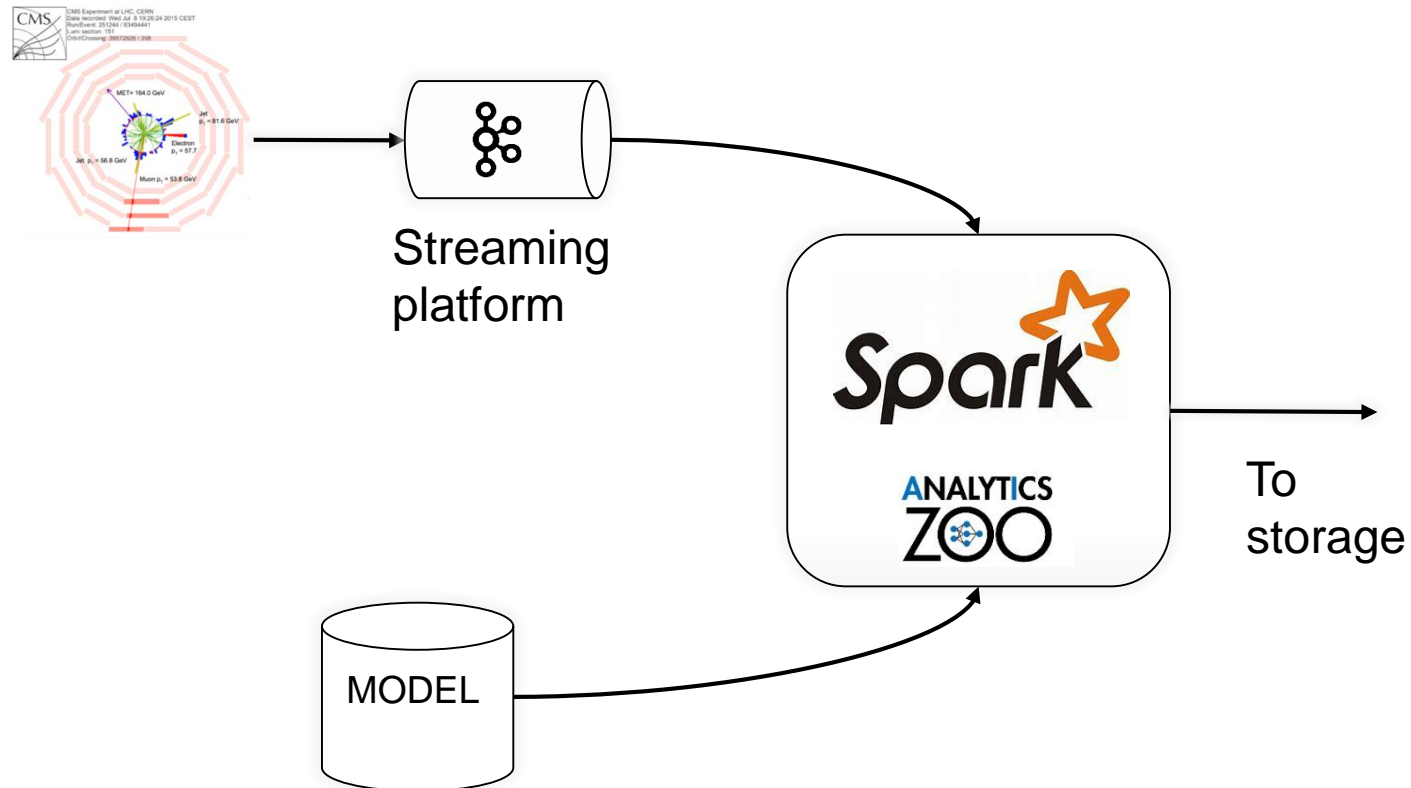
- Trained models with Analytics Zoo and BigDL
- Met with the expected accuracy results



Future work on inference

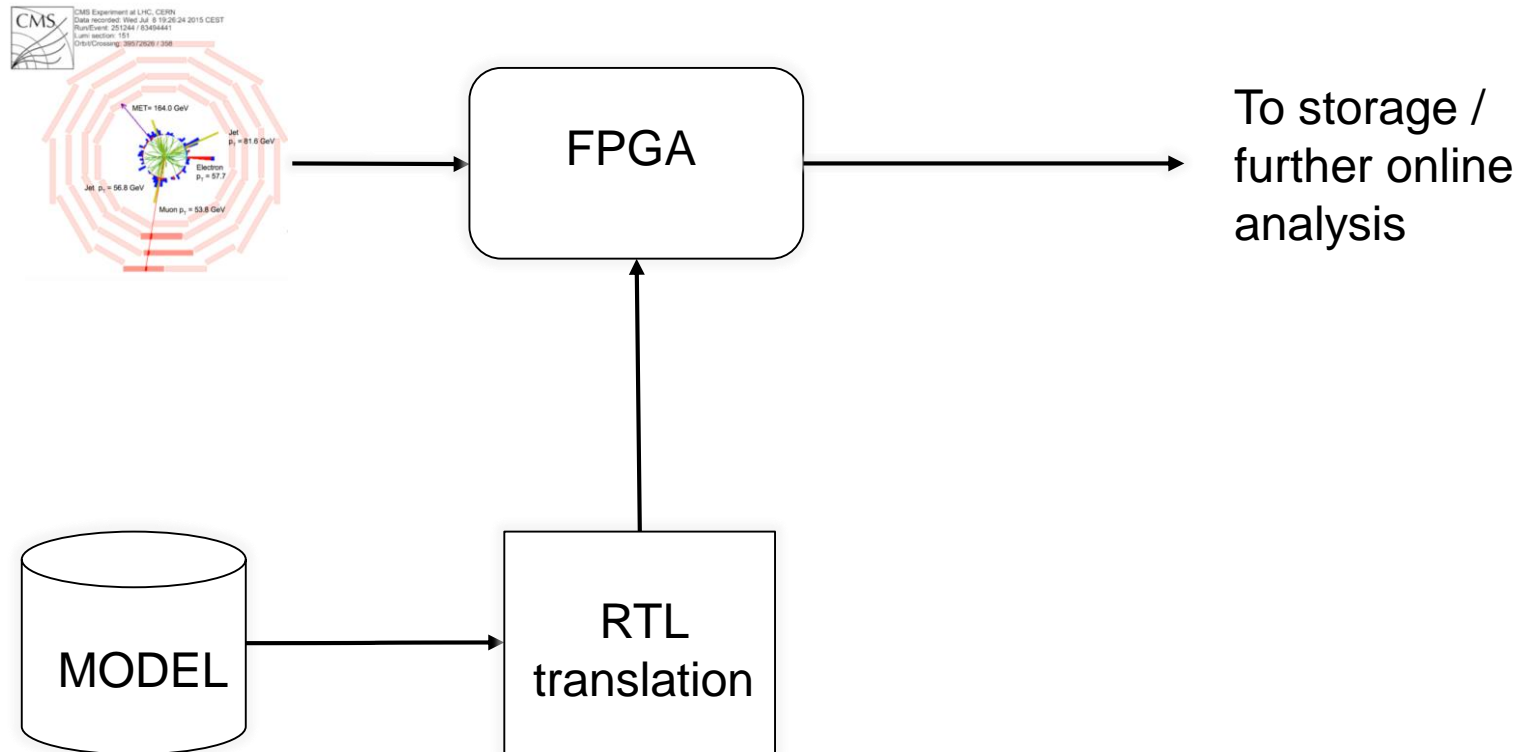
Inference and Streaming - plans

- Using Apache Kafka and Spark



Inference and FPGA - plans

- In FPGA replacing/integrating current rule-based algorithms



Summary

- We have successfully developed a Deep Learning pipeline using **Apache Spark** and **Analytics Zoo** on **Intel Xeon servers**
 - The use case developed addresses the needs for higher efficiency in event filtering at LHC experiments
 - Spark, Python notebooks and Analytics Zoo provide **intuitive APIs** for data preparation at scale on existing Hadoop cluster and cloud
 - Analytics Zoo & BigDL solve the problems of **scaling DL** on Spark clusters running on Intel Xeon servers, and offering familiar APIs to researchers
- Future work
 - Development of serving pipelines using streaming technologies /FPGAs
 - Investigate scale-out on public cloud

Acknowledgements

- CERN Colleagues
 - Hadoop, Spark and Streaming service
- CERN Openlab Data analytics project with Intel and CMS Big Data project
- Intel BigDL team, Sajan Govindan, Jennie Wang
- Colleagues from physics, authors of “Topology classification with deep learning to improve real-time event selection at the LHC”, <https://arxiv.org/abs/1807.00083> - for discussions and sharing data