

Usability Test of Big Data Technologies Applied to HEP Data Analysis

The CMS Big Data Project:

M. Cremonesi, O. Gutsche, B. Jayatilaka, J. Kowalkowski, **Y. Ou**, S. Sehrish [FNAL]

L. Canali, V. Dimakopoulos, M. Girone, V. Khristenko, E. Motesnitsalis [CERN-IT]

S.-Y. Hoh, J. Pazzini, M. Zanetti [Padova]

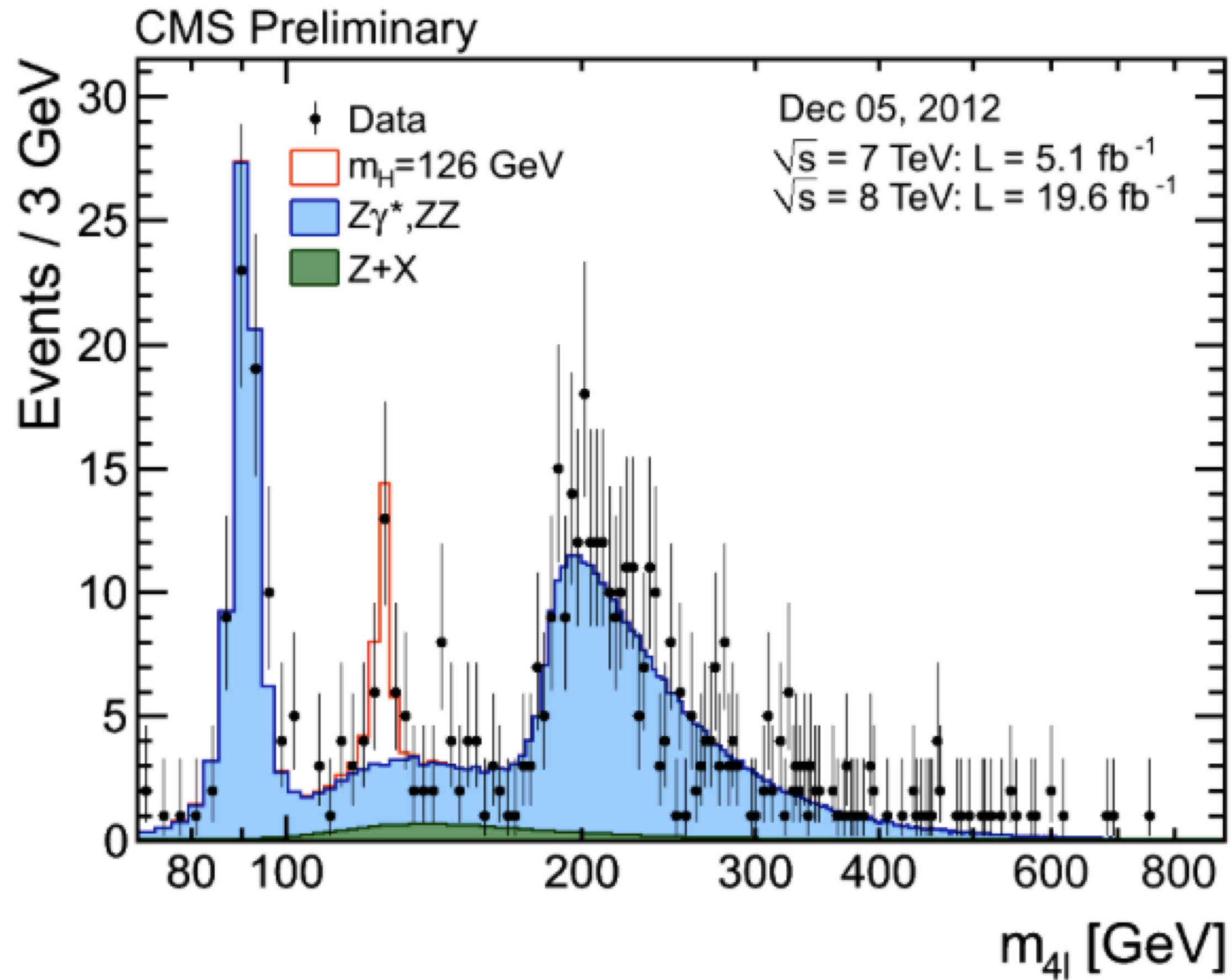
P. Elmer, J. Pivarski, A. Svyatkovskiy [Princeton]

I. Fisk [Simons Foundation]

A. Melo [Vanderbilt]

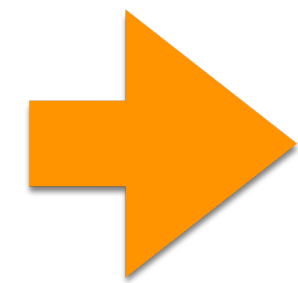


Experimental Particle Physics from Computing Perspective

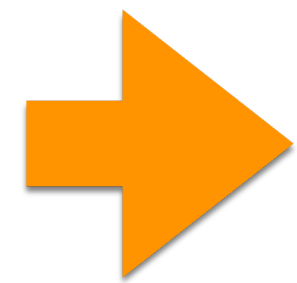


- Detect particle interactions (data), compare with theory predictions (simulation)
 - Black dots: recorded data
 - Blue shape: simulation
 - Red shape: simulation of new theory (in this case the Higgs)

Detector



DAQ
&
Trigger

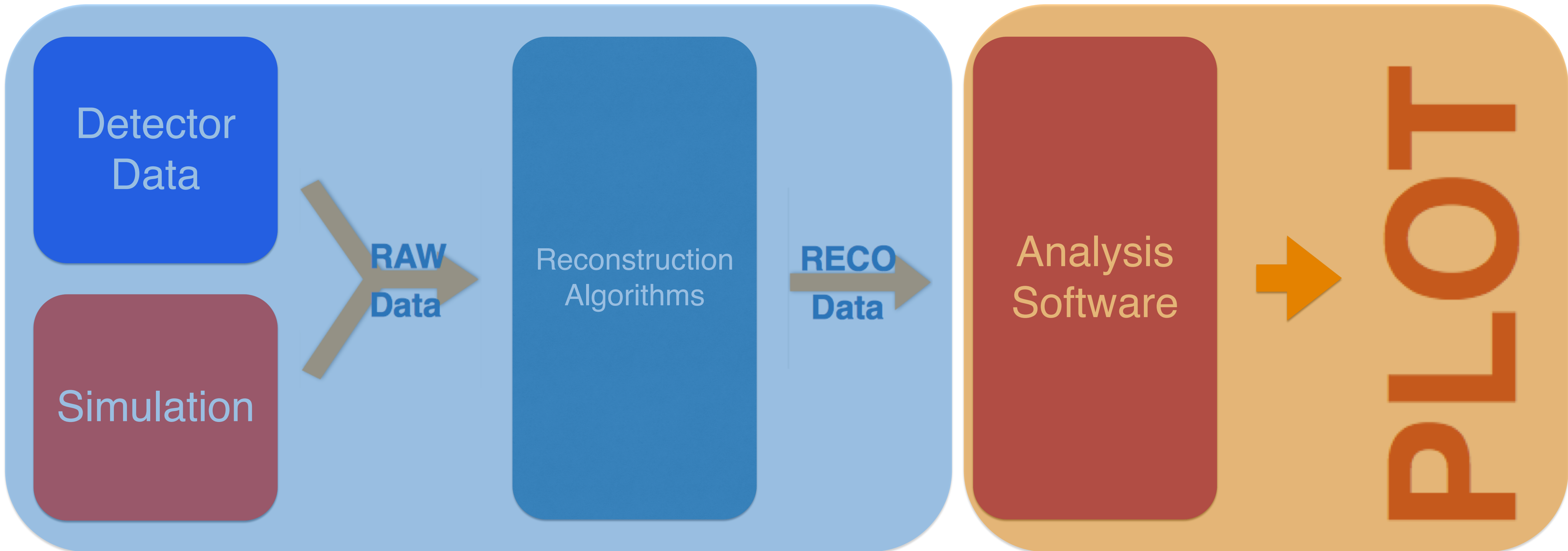


Software
&
Computing



TOLE



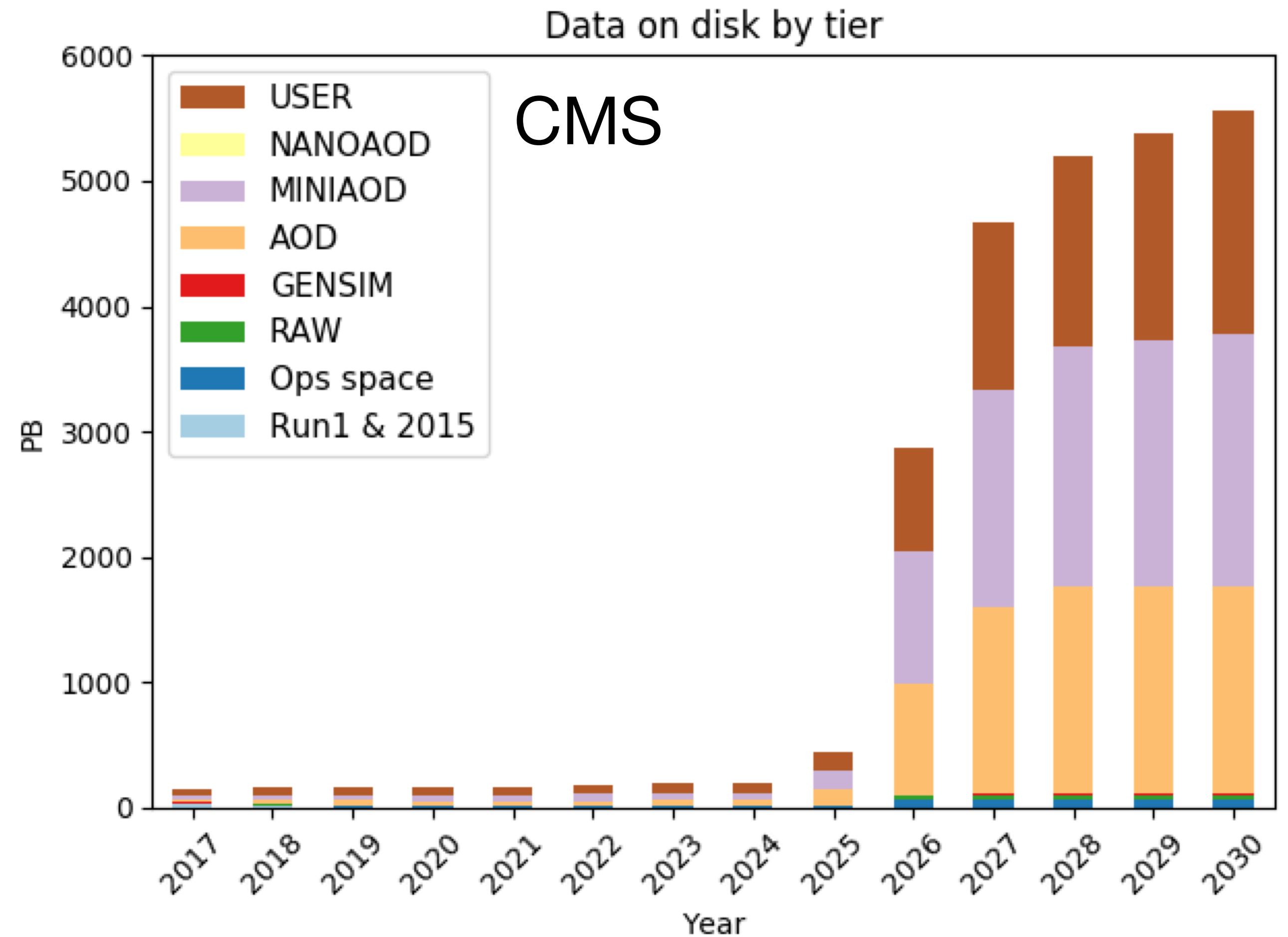


Central

Chaotic

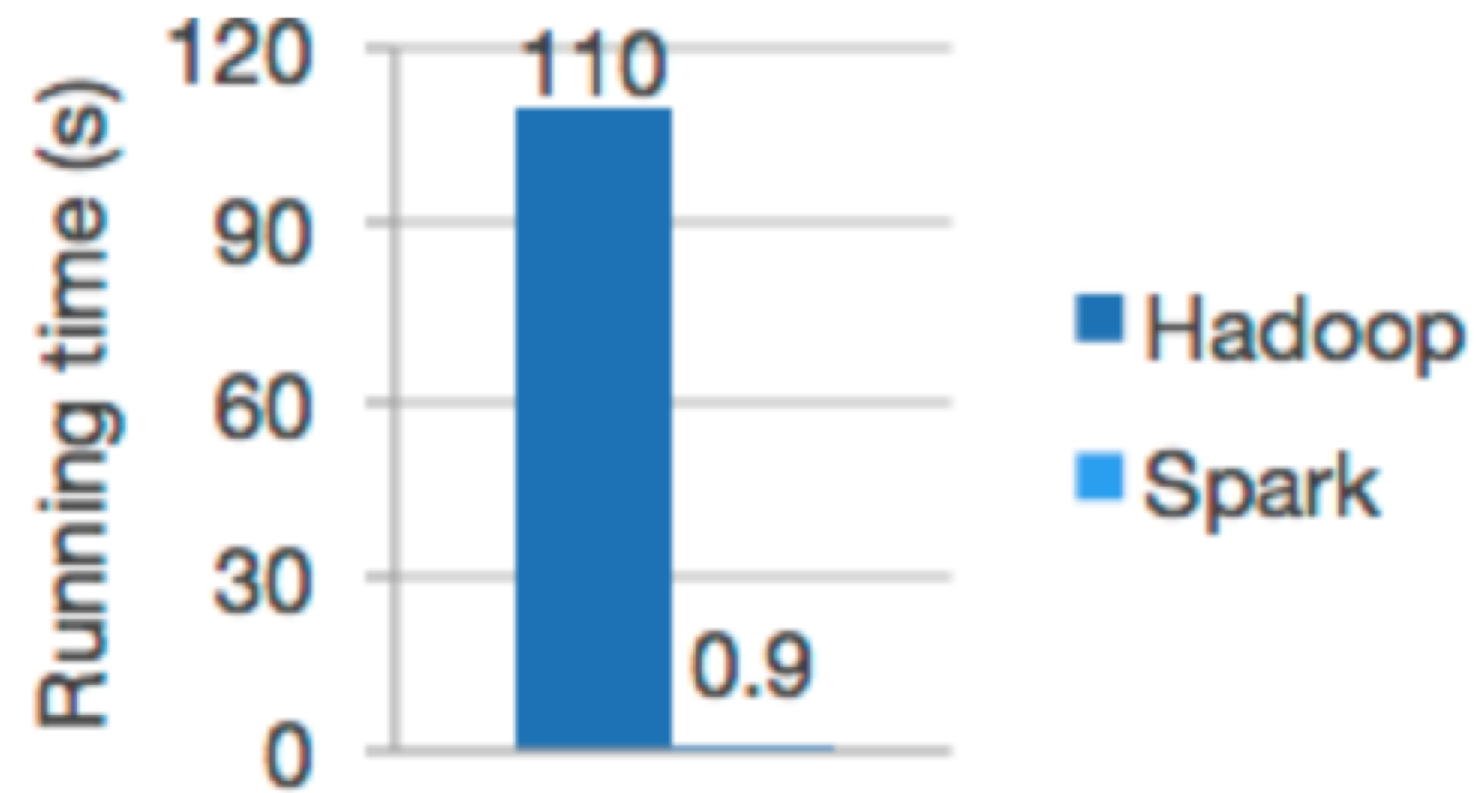
CMS Data Volume @ HL-LHC

- Extract physics results will require to handle/analyze a lot more data
 - must trim inefficiencies
- Explore industry technologies as suitable candidates for user analysis



Apache Spark

- Spark is a unified analytics engine for



Logistic regression in Hadoop and Spark

		Spark 100 TB *	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark Daytona Rules	Yes	Yes	No
Environment	dedicated data center	EC2 (i2.8xlarge)	EC2 (i2.8xlarge)

- not an official sort benchmark record



CHEP 2016: Proof of Principle

arXiv:1711.00375

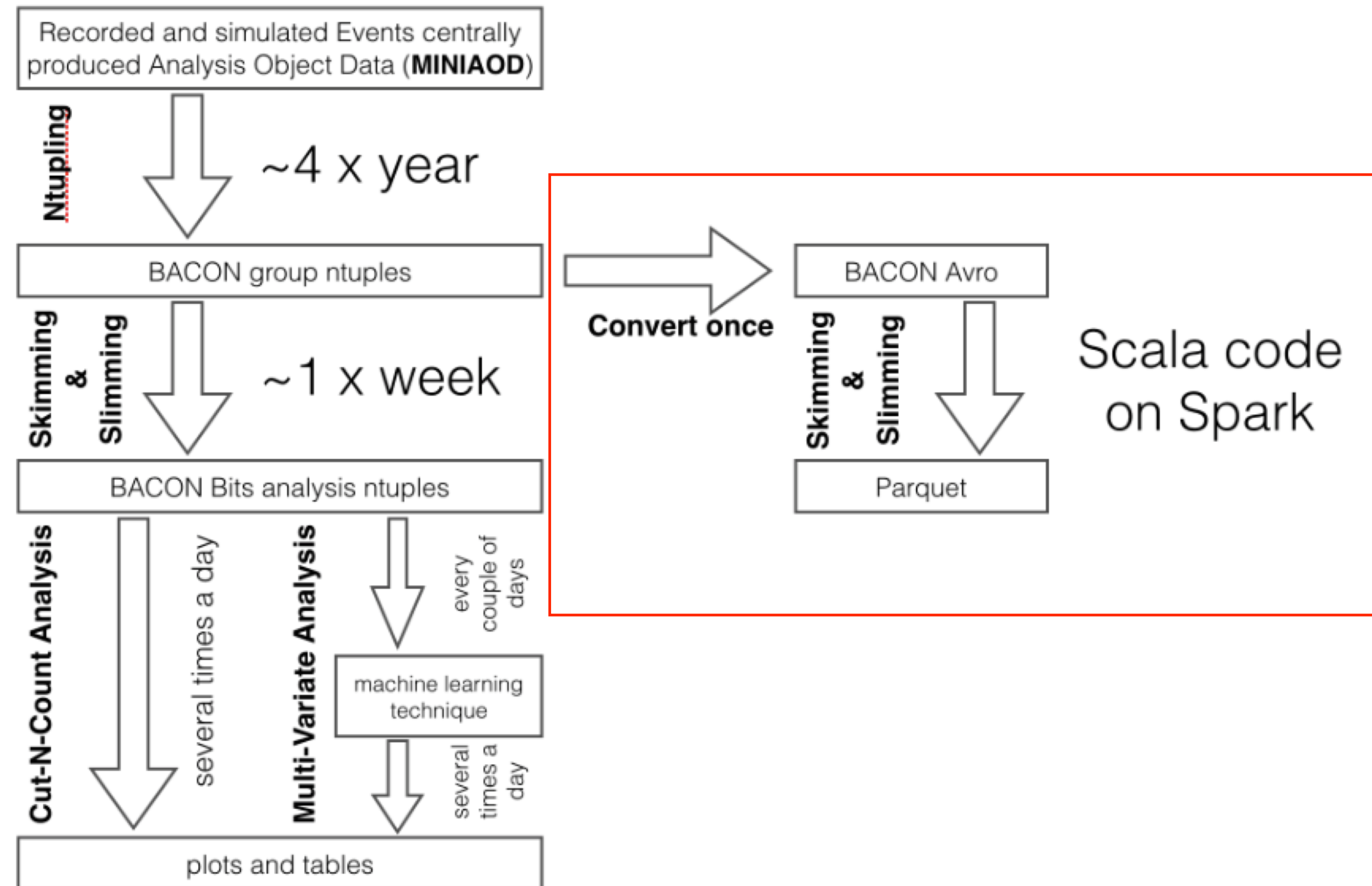
- Usability Study using Apache

Spark:

- Analyzer code in Scala
- Input converted in Avro:

<https://github.com/diana-hep/rootconverter>

- Improved user experience with optimized bookkeeping



ACAT 2017: Steps Forward

arXiv:1703.04171

Several technical advancements:

- stability to read root files in Spark: <https://github.com/diana-hep/spark-root>, eliminating the need to convert in a more suitable format
- Capability to read input files remotely using XRootD (e.g. from EOS at CERN): <https://github.com/cerndb/hadoop-xrootd>, eliminating the need to store files on HDFS

Analysis Use-case @ Vanderbilt/Padova

Analysis workflow:

- Load standard ROOT files as DataFrames (DFs)
- Open files over XRootD
- Use Spark to transform DFs
- Aggregate DFs into histograms
- Produce plots, tables, etc.. from histograms

Tools used:

- Spark-ROOT - ROOT in Spark
- Hadoop-XRootD - XRootD FS support for Hadoop
- Histogrammar - Data aggregation
- Matplotlib - Python-based plotting

<https://github.com/FNALLPC/spark-hats>

Identical physics use cases, using similar strategy, same tools, but different infrastructure

History Server Spark History S Apache Spark - Apache Spark™ spark comparec Physics forum c Apache Spark t GraphX | Apach BIGDATA-1/Zpe Zpeak_Nan

← → ↻ 🏠 https://swan003.cern.ch/user/yiou/notebooks/SWAN_projects/CMS-PD/Zpeak_Nanoaod-SPARK.ipynb 📖 ☆

To see favorites here, select ☆ then ☆, and drag to the Favorites Bar folder. Or import from another browser. [Import favorites](#)

FILE EDIT VIEW INSERT CELL KERNEL HELP Not Trusted | Py

📁 + ✂️ 📄 📄 ⬆️ ⬇️ ⏪ ⏹️ ⏩ Code 🗂️ ⌘ ☆

Access DataFrame content


Return a list of columns in one of the DataFrame, a column corresponds to branche in ROOT TTree.

```
In [3]: from pyspark.sql.functions import lit
        from samples import *

        DFList = []

        for s in samples:
            print 'Loading {0} sample from EOS file'.format(s)
            dsPath = "root://eospublic.cern.ch/eos/opstest/cmspd-bigdata/"+samples[s]['filename']
            tempDF = sqlContext.read \
                .format("org.dianahep.sparkroot") \
                .option("tree", "Events") \
                .load(dsPath)\
                .withColumn("pseudoweight", lit(samples[s]['weight'])) \
                .withColumn("sample", lit(s))
            DFList.append(tempDF)
```

```
Loading ZH sample from EOS file
Loading TT sample from EOS file
Loading WW sample from EOS file
Loading SingleMuon sample from EOS file
Loading ZZ sample from EOS file
Loading DYJetsToLL sample from EOS file
Loading WZ sample from EOS file
```

To see favorites here, select  then , and drag to the Favorites Bar folder. Or import from another browser. [Import favorites](#)

Return statistical information about the data

Exploit pySparkSql functions to get statistical insights on the DataFrame

```
In [13]: from pyspark.sql.functions import *

print 'Number of events, pre-selection level'

DF.groupBy("sample").count().show()

print 'Number of events, Dimuon invariant mass in [70-110] GeV'

DF.where( (col("Dimuon.mass") > 70) & (col("Dimuon.mass") < 110) ).groupBy("sample").count().show()

print 'Mean of Dimuon mass, evaluated in [70-110] GeV range'

DF.where( (col("Dimuon.mass") > 70) & (col("Dimuon.mass") < 110) ).groupBy('sample').mean('Dimuon.mass').show()

print 'Description of Dimuon mass features for SingleMuon dataset only, evaluated in [70-110] GeV range'

DF.where( (col("Dimuon.mass") > 70) & (col("Dimuon.mass") < 110) & (DF["sample"] == "SingleMuon") ).describe('Dimuon.mass').show()
```

Number of events, pre-selection level

```
+-----+-----+
|  sample|  count|
+-----+-----+
|      ZZ| 480144|
|      WW| 348954|
|      ZH|1039542|
|DYJetsToLL| 549790|
|SingleMuon|2394268|
|      WZ| 424048|
|      TT| 382077|
+-----+-----+
```

Plotting the Zpeak mass

Finally, the interesting variables are plotted using accumulator from Histogrammar package and the graphic object is handled by backend matplotlib.

```
In [14]: # add histogrammar to driver python path
sc.addPyFile("/eos/project/s/swan/public/PhysicsAnalysis/histogrammar.zip")
```

```
In [15]: # Load libraries, and append histogrammar functionalities to dataframe
import matplotlib.pyplot as plt
%matplotlib inline
import histogrammar as hg
import histogrammar.sparksql
import numpy as np

DF = DF.where( (col("Dimuon.mass") > 70) & (col("Dimuon.mass") < 110) )

hg.sparksql.addMethods(DF)

plots = hg.UntypedLabel(
    # 1d histograms
    LeadPt      = hg.Bin(50, 30, 180, DF['Dimuon.mu1_pt'],hg.Sum(DF['pseudoweight'])),
    LeadPtEta   = hg.Bin(48, -2.4, 2.4, DF['Dimuon.mu1_eta'],hg.Sum(DF['pseudoweight'])),
    SubLeadPt   = hg.Bin(100, 0, 200, DF['Dimuon.mu2_pt'],hg.Sum(DF['pseudoweight'])),
    SubLeadPtEta = hg.Bin(48, -2.4, 2.4, DF['Dimuon.mu2_eta'],hg.Sum(DF['pseudoweight'])),
    InvMass     = hg.Bin(80, 70, 110, DF['Dimuon.mass'],hg.Sum(DF['pseudoweight'])),
    DeltaR      = hg.Bin(50, 0, 5, DF['Dimuon.dPhi'],hg.Sum(DF['pseudoweight'])),
    DeltaPhi    = hg.Bin(64, -3.2, 3.2, DF['Dimuon.dR'],hg.Sum(DF['pseudoweight'])),
)

# Make a set of histograms, categorized per-sample
bulkHisto = hg.Categorize(quantity = DF['sample'], value = plots)

# Fill from spark
bulkHisto.fillsparksql(df=DF)
print 'Filling histogrammar done'
```

Filling histogrammar done

```

In [16]: # variable for plotting
VARIABLE = 'InvMass'

fig = plt.figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')

aHisto = bulkHisto("SingleMuon")(VARIABLE)
nBins = len(aHisto.values)
edges = np.linspace(aHisto.low, aHisto.high, nBins + 1)
width = (aHisto.high - aHisto.low) / nBins

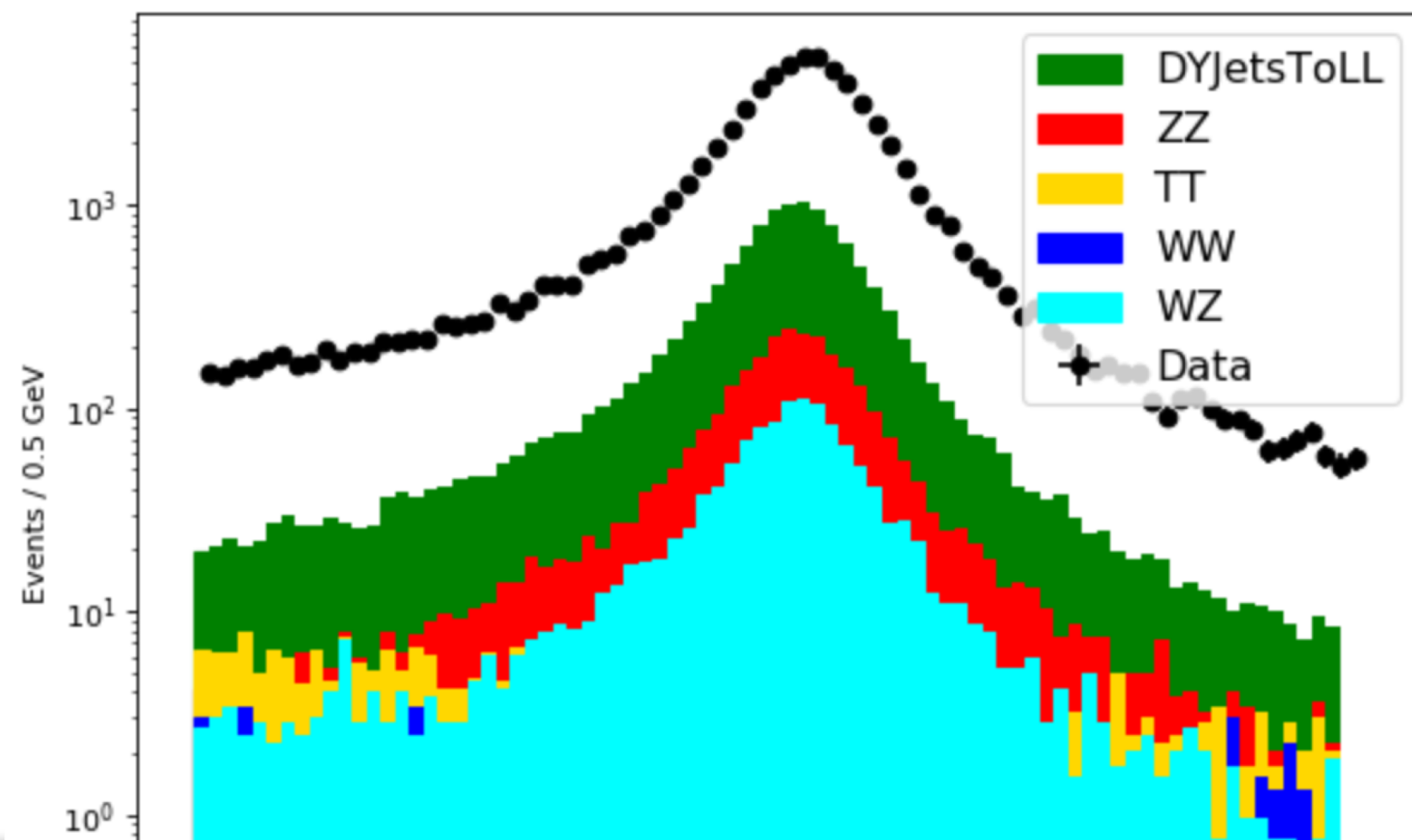
plotVals = {}
for k in ['DYJetsToLL', 'ZZ', 'TT', 'WW', 'WZ']:
    #plotVals[k] = [x.toJson()['data']*0.19 for x in bulkHisto(k)(VARIABLE).values]
    plotVals[k] = [x.sum*0.19 for x in bulkHisto(k)(VARIABLE).values]
    plt.bar(edges[:-1], plotVals[k], width=width, label=k, color=samples[k]['color'], edgecolor=samples[k]['color'], fill=True, log=True)

xdata = np.linspace(aHisto.low+0.5*width, aHisto.high+0.5*width, nBins)
#ydata = [x.toJson()['data'] for x in bulkHisto('SingleMuon')(VARIABLE).values]
ydata = [x.sum for x in bulkHisto('SingleMuon')(VARIABLE).values]
yerror = [x**0.5 for x in ydata]

plt.errorbar(xdata, ydata, fmt='ko', label="Data", xerr=width/2, yerr=yerror, ecolor='black')

plt.xlabel('Dimuon invariant mass m($\mu\mu$) (GeV)')
plt.ylabel('Events / 0.5 GeV')
#plt.yscale('log')
plt.legend(loc='upper right', fontsize='x-large', )

```



Running Padova Workflow at Vanderbilt

Obstacles:

- The xrootd connector is not packaged in an easily deployable way (like spark-root is, for instance)
- Notebook's spark configurations are “site-specific”
- Without root privilege, it is hard to install new packages or do some changes on configuration

~ 3 weeks to make it work

```
'Jet_bReg',  
### Weight
```

Spark UI

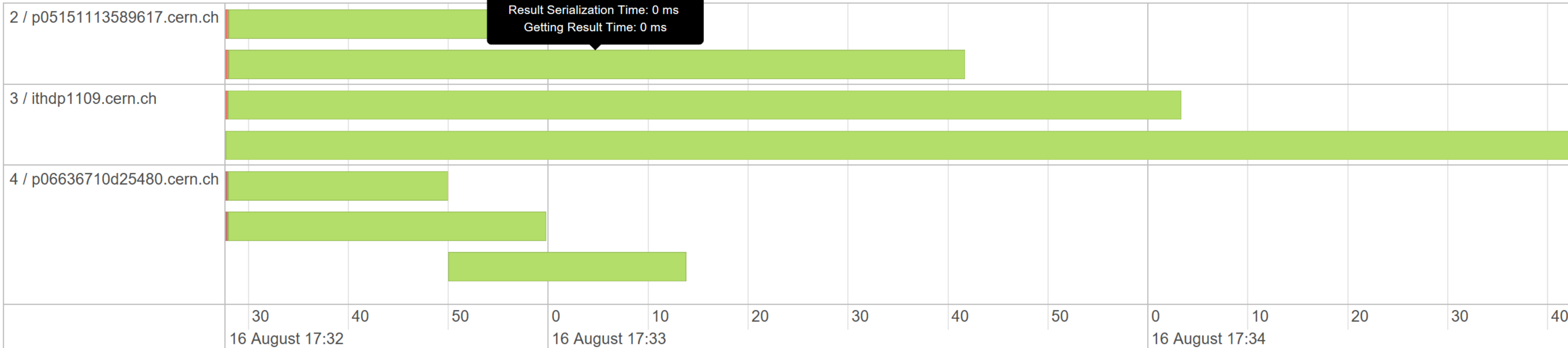
Details for Stage 4 (Attempt 0)

Total Time Across All Tasks: 4.9 min
Locality Level Summary: Process local: 7
Shuffle Write: 354.0 B / 6

- ▶ DAG Visualization
- ▶ Show Additional Metrics
- ▼ Event Timeline
- Enable zooming

- Scheduler Delay
- Task Deserialization Time
- Shuffle Read Time
- Executor Computing Time
- Shuffle Write Time
- Result Serialization Time

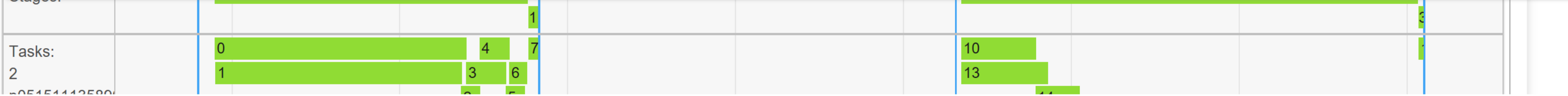
Task 4 (attempt 0)
Status: SUCCESS
Launch Time: 2018/08/16 17:32:27
Finish Time: 2018/08/16 17:33:41
Scheduler Delay: 12 ms
Task Deserialization Time: 0.3 s
Shuffle Read Time: 0 ms
Executor Computing Time: 1.2 min
Shuffle Write Time: 0 ms
Result Serialization Time: 0 ms
Getting Result Time: 0 ms



In [7]:

Summary Metrics for 6 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	22 s	24 s	45 s	1.2 min	1.6 min
GC Time	0.4 s	0.6 s	1 s	1 s	3 s
Shuffle Write Size / Records	59.0 B / 1	59.0 B / 1	59.0 B / 1	59.0 B / 1	59.0 B / 1



Running Padova Workflow with VC3

What is it?

- The **Virtual Clusters for Community Computation** (VC3) is a service that shares custom software across multiple sites through the establishment of a virtual cluster
- Users interface to the system by providing:
 - credentials to access different computing centers
 - details to set up custom software
 - The desired size of the requested virtual cluster
- VC3 creates the virtual cluster
 - generating a head node in a private cloud
 - Allocating resources from the remote facilities
- The user simply sees a single private cluster that they can access and share with collaborators to run science applications
 - It allows users to share custom software across multiple sites on the fly i.e. do not need root privilege



Running Padova Workflow with VC3

Able to deploy to the virtual cluster

- Spark
- Hadoop-xrootd connector
- Matplotlib, Histogrammar libraries

=> Able to run Padova workflow on resources allocated from the UChicago Midwest Tier 2 cluster (2 executors, 8 cores, 12GB RAM)

Need to test with resources with other remote sites (FNAL?)

```
--keytab KEYTAB          secure HDFS.  
                          The full path to the file that contains the keytab for the  
                          principal specified above. This keytab will be copied to  
                          the node running the Application Master via the Secure  
                          Distributed Cache, for renewing the login tickets and the  
                          delegation tokens periodically.
```

```
sh-4.2$ spark-submit --executor-memory 2G --master spark://128.135.158.228:7077 --verbose --conf spark.sql.caseSensitive=true --co  
nf spark.serializer=org.apache.spark.serializer.KryoSerializer --packages org.diana-hep:spark-root_2.11:0.1.16,org.diana-hep:histo  
grammar-sparksql_2.11:1.0.4 Zpeak_Nanoaod-SPARK.py  
Using properties file: null
```

```
Parsed arguments:  
master                spark://128.135.158.228:7077  
deployMode            null  
executorMemory        2G  
executorCores         null  
totalExecutorCores   null  
propertiesFile        null  
driverMemory          null  
driverCores           null  
driverExtraClassPath null  
driverExtraLibraryPath null  
driverExtraJavaOptions null  
supervise             false  
queue                 null  
numExecutors          null  
files                 null  
pyFiles               null  
archives              null  
mainClass             null  
primaryResource       file:/home/yiyangou/Zpeak_Nanoaod-SPARK.py  
name                  Zpeak_Nanoaod-SPARK.py  
childArgs             []  
jars                  null  
packages              org.diana-hep:spark-root_2.11:0.1.16,org.diana-hep:histogrammar-sparksql_2.11:1.0.4  
packagesExclusions   null  
repositories          null  
verbose              true
```

```
18/08/14 17:45:48 INFO TaskSetManager: Starting task 0.0 in stage 12.0 (TID 46, 10.50.182.4, executor 5, partition 5, PROCESS_LOCAL, 4957 bytes)
18/08/14 17:45:48 INFO BlockManagerInfo: Added broadcast_12_piece0 in memory on 10.50.182.137:34664 (size: 41.7 KB, free: 950.6 MB)
18/08/14 17:45:48 INFO BlockManagerInfo: Added broadcast_12_piece0 in memory on 10.50.182.4:40424 (size: 41.7 KB, free: 943.7 MB)
18/08/14 17:45:49 INFO TaskSetManager: Finished task 0.0 in stage 12.0 (TID 46) in 285 ms on 10.50.182.4 (executor 5) (1/2)
18/08/14 17:45:49 INFO TaskSetManager: Finished task 1.0 in stage 12.0 (TID 45) in 532 ms on 10.50.182.137 (executor 3) (2/2)
18/08/14 17:45:49 INFO TaskSchedulerImpl: Removed TaskSet 12.0, whose tasks have all completed, from pool
18/08/14 17:45:49 INFO DAGScheduler: ResultStage 12 (showString at NativeMethodAccessorImpl.java:0) finished in 0.533 s
18/08/14 17:45:49 INFO DAGScheduler: Job 7 finished: showString at NativeMethodAccessorImpl.java:0, took 0.548779 s
```

```
+-----+-----+-----+-----+-----+-----+
| sample|nMuon| Muon_pt| Muon_eta| Muon_phi|Muon_charge|
+-----+-----+-----+-----+-----+-----+
|DYJetsToLL| 1| [34.75507]| [-1.3212891]| [-1.0375977]| [1]|
|DYJetsToLL| 0| [ ]| [ ]| [ ]| [ ]|
|DYJetsToLL| 1|[3.3921983]| [-1.5285645]| [-0.2514038]| [-1]|
|DYJetsToLL| 0| [ ]| [ ]| [ ]| [ ]|
|DYJetsToLL| 0| [ ]| [ ]| [ ]| [ ]|
+-----+-----+-----+-----+-----+-----+
```

only showing top 5 rows

```
18/08/14 17:45:49 INFO ContextCleaner: Cleaned accumulator 394
18/08/14 17:45:49 INFO BlockManagerInfo: Removed broadcast_11_piece0 on 128.135.158.228:42302 in memory (size: 41.7 KB, free: 366.2 MB)
18/08/14 17:45:49 INFO BlockManagerInfo: Removed broadcast_11_piece0 on 10.50.182.137:40651 in memory (size: 41.7 KB, free: 944.1 MB)
18/08/14 17:45:49 INFO BlockManagerInfo: Removed broadcast_11_piece0 on 10.50.182.4:44555 in memory (size: 41.7 KB, free: 713.8 MB)
18/08/14 17:45:49 INFO BlockManagerInfo: Removed broadcast_11_piece0 on 10.50.182.137:44506 in memory (size: 41.7 KB, free: 932.2 MB)
18/08/14 17:45:49 INFO BlockManagerInfo: Removed broadcast_11_piece0 on 10.50.181.72:38015 in memory (size: 41.7 KB, free: 961.1 MB)
18/08/14 17:45:49 INFO BlockManagerInfo: Removed broadcast_12_piece0 on 128.135.158.228:42302 in memory (size: 41.7 KB, free: 366.3 MB)
18/08/14 17:45:49 INFO BlockManagerInfo: Removed broadcast_12_piece0 on 10.50.182.137:34664 in memory (size: 41.7 KB, free: 950.6 MB)
18/08/14 17:45:49 INFO BlockManagerInfo: Removed broadcast_12_piece0 on 10.50.182.4:40424 in memory (size: 41.7 KB, free: 943.7 MB)
```



Summary

User experience with the new tools

- first-year CS undergraduate student can learn to run the workflow in one day
 - No knowledge of physics whatsoever, limited computing knowledge

Portability

- Run the Padova code at Vanderbilt
- Major showstopper: environment setup
- With the acquired knowledge, able to run at CERN in a few hours
 - SWAN provides a better UI
- VC3 eliminates the site configuration issue
 - Need to be tested by using the service over different remote sites

Backup

What's Next: Coffea Development

- Generalized version of the code used so far by Padova/Vanderbilt
 - Fully portable (no configuration issues)
 - Achievable through VC3
 - Use-case independent (in principle it already is)
- COmpact Framework For Elaborate Algorithms
- Consist in:
 - List of centrally-produced dataset in experiment-specific format needed for analysis => coffeabeans
 - Custom-made version of the experiment software to produce privately datasets in the experiment-specific format => CoffeaGrinder (this step may be needed to add information)
 - List of privately/centrally produced dataset in experiment-specific format => coffeapowder
 - Apache Spark analysis code => CoffeaMaker
 - Reduced datasets/analysis plots => coffeacups
 - Interface with the experiment statistical packages => CoffeaDrinker

