CMS Analysis and Data Reduction with Apache Spark

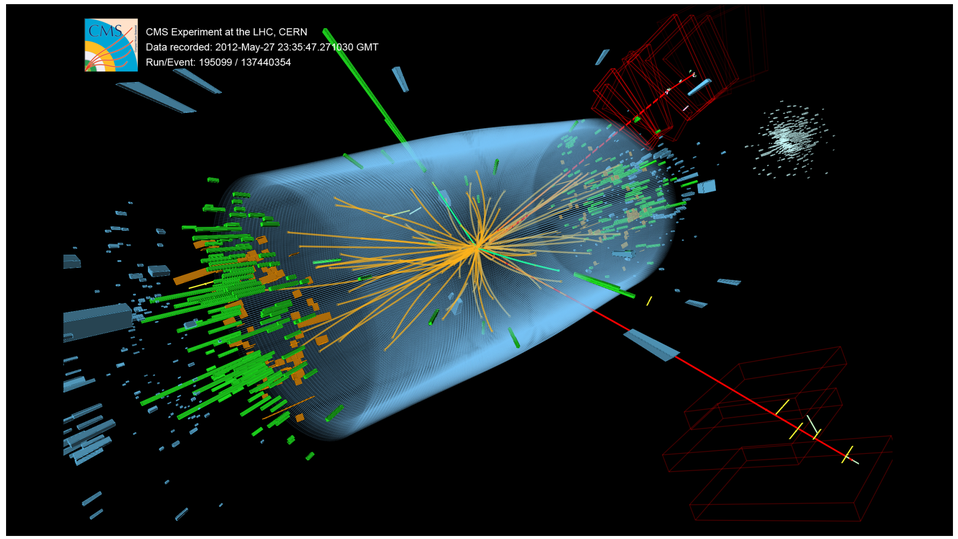
Oliver Gutsche, Matteo Cremonesi, Bo Jayatilaka, Jim Kowalkowski, Saba Sehrish - Fermilab
Peter Elmer, Jim Pivarski, Alexey Svyatkovskiy - Princeton University
Maria Girone, Luca Canali, Kacper Surdy, Vaggelis Motesnitsalis - CERN
Illia Cremer - Intel
Ian Fisk - Simons Foundations
Viktor Khristenko - University of Iowa

18th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2017)
Track 2: Data Analysis - Algorithms and Tools

22. August 2017

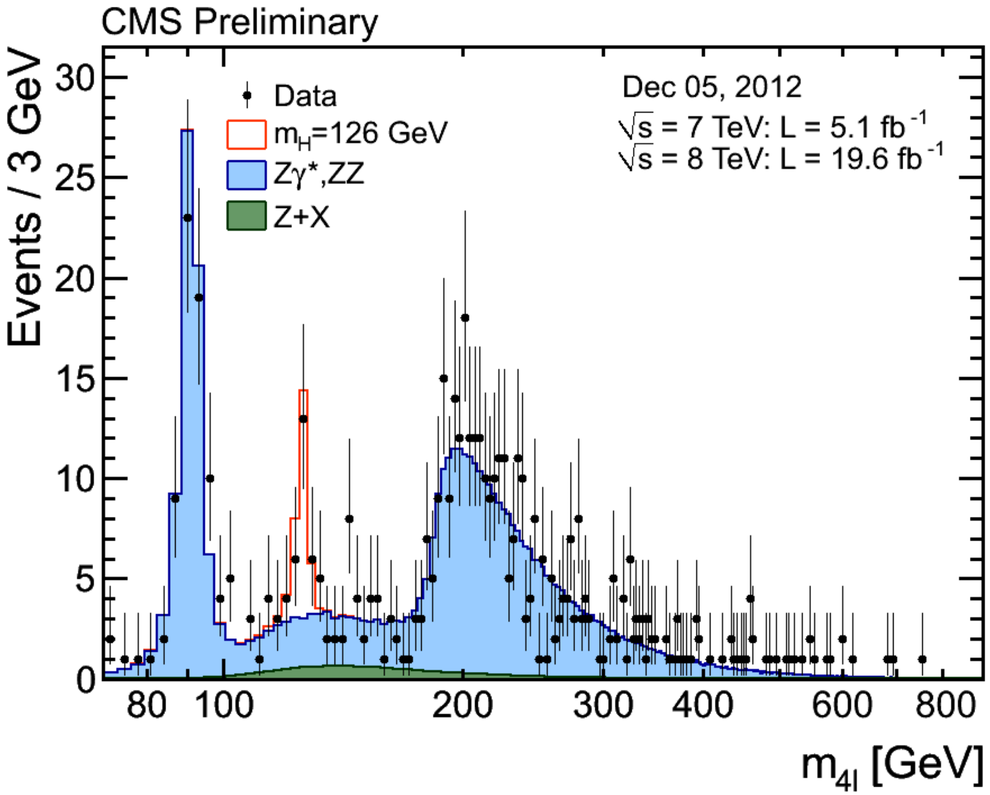
Experimental Particle Physics - the Journey

Particle Collisions

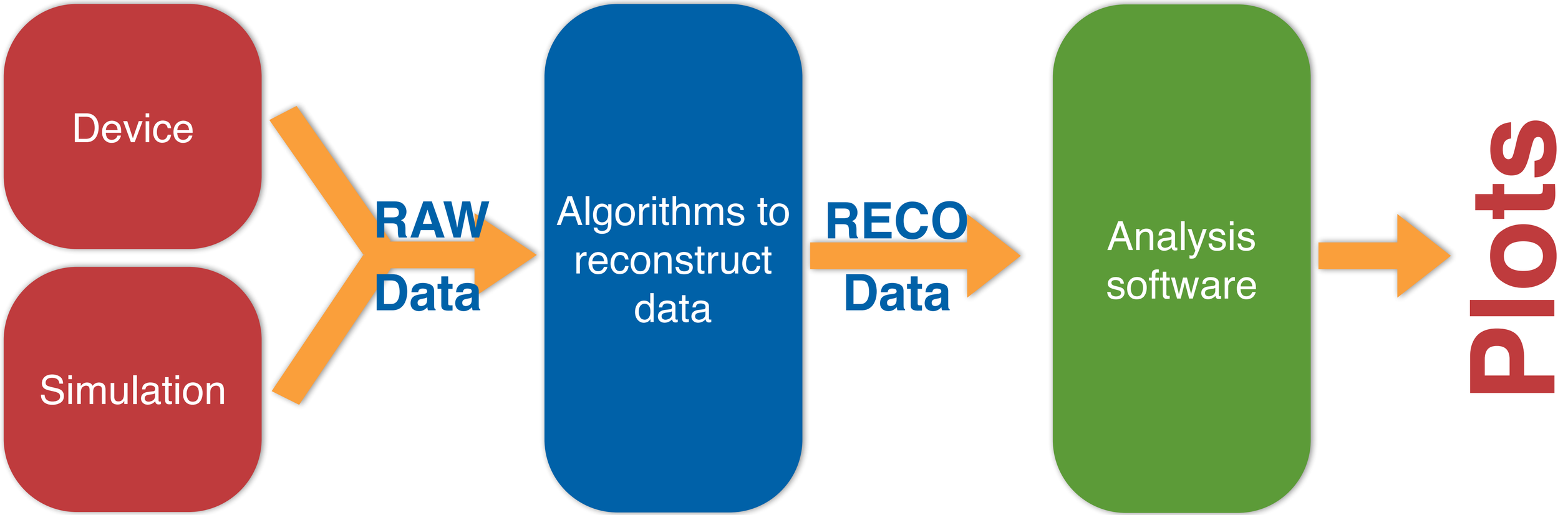


Big Computing

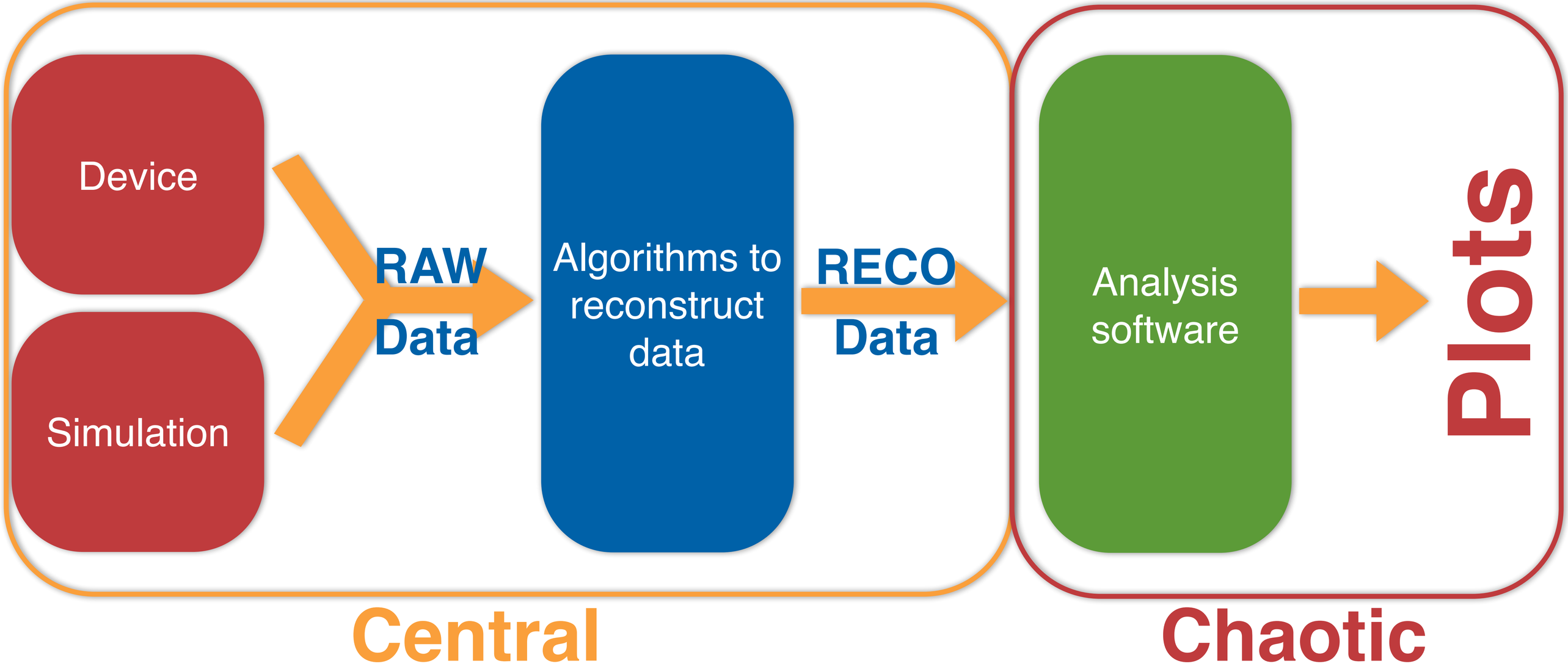
Physics Discoveries



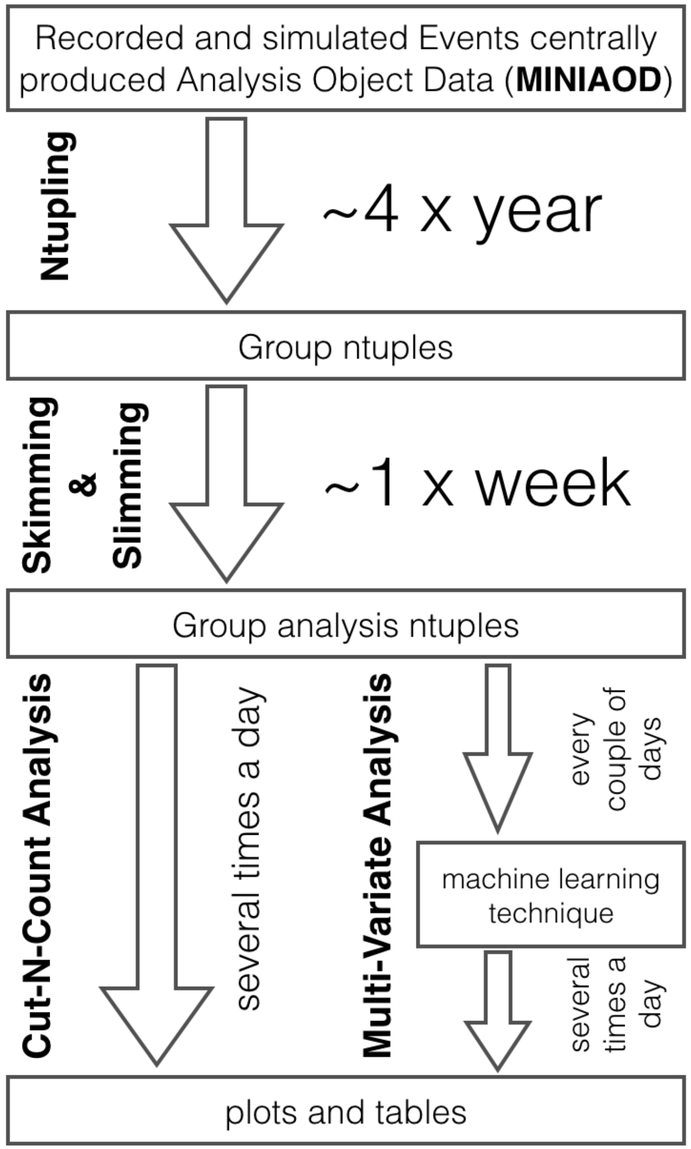
Big Computing



Analysis in CMS



Analysis - A multi-step process

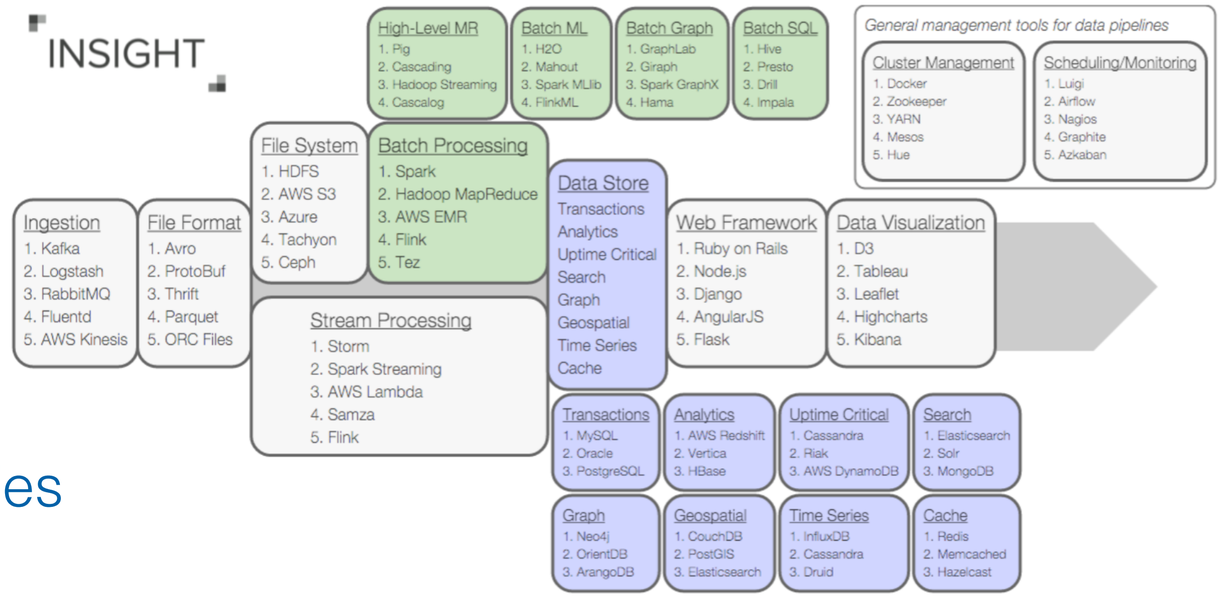


- Minimize **Time to Insight**
  - Analysis is a conversation with data - Interactivity is key
- Many different physics topics concurrently under investigation
  - Different slices of data are relevant for each analysis
- Programmatically same analysis steps
  - Skimming (filter specific collisions)
  - Slimming (reduce content per collision)
  - Thinning (partial read of data, Dynamic Tree Traversal in CS)

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



Two Investigation Thrusts - Apache Spark

- Thrust 1: Usability study
  - End-to-end investigation to conduct CMS physics analyses in Apache Spark
  - Produce publication quality plots and tables from CMS data

- Thrust 2: CMS Data Reduction Facility
  - CERN openlab / Intel project
  - Demonstrate reduction capabilities producing analysis ntuples using Apache Spark
    - Ambitious goal: reduce 1 PB input to 1 TB output in 5 hours

Status @ CHEP 2016

- Usability Study using Apache Spark
  - Conversion of data to AVRO format and upload to HDFS
  - Analysis implemented in Scala
  - Processing in Apache Spark
  - Result:
    - Spark analysis simpler to structure (functional programming) and easier to port
    - Performance comparison challenging (apples-to-apples comparison)

- Lesson's learned
  - Analysis tools (especially plots) not easily transferrable to map-reduce style processing
  - Conversion seen as a big impediment to use new technology

Big Data in HEP: A comprehensive use case study

Oliver Gutsche[1], Matteo Cremonesi[1], Peter Elmer[2], Bo Jayatilaka[1], Jim Kowalkowski[1], Jim Pivarski[2], Saba Sehrish[1], Cristina Mantilla Surez[3], Alexey Svyatkovskiy[2], Nhan Tran[1]

[1]Fermi National Accelerator Laboratory, Batavia, IL, USA
[2]Princeton University, Princeton, NJ, USA
[3]Fermi National Accelerator Laboratory, Batavia, IL, USA; now Johns Hopkins University, Baltimore, MD, USA

E-mail: gutsche@fnal.gov

Abstract. Experimental Particle Physics has been at the forefront of analyzing the worlds largest datasets for decades. The HEP community was the first to develop suitable software and computing tools for this task. In recent times, new toolkits and systems collectively called Big Data technologies have emerged to support the analysis of Petabyte and Exabyte datasets in industry. While the principles of data analysis in HEP have not changed (filtering and transforming experiment-specific data formats), these new technologies use different approaches and promise a fresh look at analysis of very large datasets and could potentially reduce the time-to-physics with increased interactivity.

In this talk, we present an active LHC Run 2 analysis, searching for dark matter with the CMS detector, as a testbed for Big Data technologies. We directly compare the traditional NTuple-based analysis with an equivalent analysis using Apache Spark on the Hadoop ecosystem and beyond. In both cases, we start the analysis with the official experiment data formats and produce publication physics plots. We will discuss advantages and disadvantages of each approach and give an outlook on further studies needed.

1. Introduction

In 2012, Particle Physics entered a new age. With the discovery of the Higgs Boson, the Standard Model was extended with the missing mechanism that gives rise to particle masses. This theory, developed since the 1960's and constrained by numerous experiments before discovery, followed a predictable path. Now that the Higgs Boson has been discovered, the way forward is wide open. Many different theories that could explain the shortcomings of the Standard Model need to be investigated.

Particle physics has always been at the forefront of analyzing the world's largest datasets. Although we constrain ourselves in this paper to High Energy Physics (HEP), where known particles are made to collide at the highest energies possible, the underlying data organization holds for all sub-fields of particle physics. The most basic concept of how data in HEP is organized is an event: all detector signals associated with a single beam crossing and high-energy collision. Events are the atomic unit of HEP data and may be processed separately, which is why the computational problems of particle physics can be easily parallelized.

Events must be reconstructed to convert detector signals into measurements of particles produced in collisions. This is usually done centrally by each experiment. The reconstructed

https://arxiv.org/abs/1703.04171

DIANA: Histogrammar

- Spark manages concurrency (no event loop)
- Histogrammar designed for map-reduce environment
  - Functional interface
    - Fill histograms by passing lambda functions
    - Same as transformations in Spark
  - Histogrammar fills histogram data structures ➜ afterwards convert into favorite plotting tool (for example ROOT)

histo·*grammar*
/histō,ˈgɹæm.ər/

MAKING HISTOGRAMS FUNCTIONAL

ROOT:

```
histogram = ROOT.TH1F("name", "title", 100, 0, 10)
for muon in muons:
    if muon.pt > 10:
        histogram.fill(muon.mass)
```

Histogrammar:

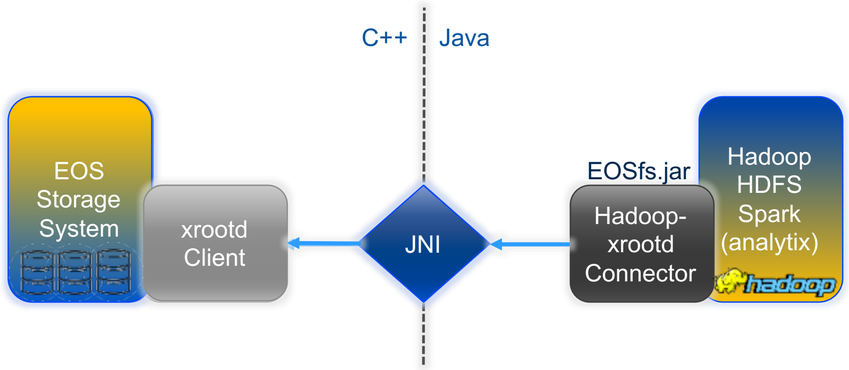
```
histogram = Select(lambda mu: mu.pt > 10,
                   Bin(100, 0, 10, lambda mu: mu.mass,
                       Count()))
for muon in muons:
    histogram.fill(muon)
```

http://histogrammar.org

Example from usability study using histogrammar

DIANA: spark-root

- Read ROOT files directly in Apache Spark
  - Connect ROOT to ApacheSpark to be able to read ROOT TTrees and infer the schema
  - Manipulate the data via Spark's DataFrames/Datasets/RDDs
- Read from HDFS and (new) from CERN EOS




```
df = sqlContext.read.format("org.dianahep.sparkroot").option("tree", "Events").load("hdfs:/cms/bigdatasci/vkhriste/data/publiccms_muionia_aod")
#df1 = sqlContext.read.format("org.dianahep.sparkroot").option("tree", "Events").load("hdfs:/cms/bigdatasci/vkhriste/data/publiccms_muionia_aod/0000/FEEFB039-0978-E011-BB60-E41F131815BC.root")
df.printSchema()
```

```
root
 |-- EventAuxiliary: struct (nullable = true)
 |    |-- processHistoryID_: struct (nullable = true)
 |    |    |-- hash_: string (nullable = true)
 |    |-- id_: struct (nullable = true)
 |    |    |-- run_: integer (nullable = true)
 |    |    |-- luminosityBlock_: integer (nullable = true)
 |    |    |-- event_: integer (nullable = true)
 |    |-- processGUID_: string (nullable = true)
 |    |-- time_: struct (nullable = true)
 |    |    |-- timeLow_: integer (nullable = true)
 |    |    |-- timeHigh_: integer (nullable = true)
 |    |-- luminosityBlock_: integer (nullable = true)
 |    |-- isRealData_: boolean (nullable = true)
 |    |-- experimentType_: integer (nullable = true)
 |    |-- bunchCrossing_: integer (nullable = true)
 |    |-- orbitNumber_: integer (nullable = true)
 |    |-- storeNumber_: integer (nullable = true)
 |-- EventBranchEntryInfo: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- branchID_: struct (nullable = true)
 |    |    |    |-- id_: integer (nullable = true)
 |    |    |-- productStatus_: byte (nullable = true)
 |    |    |-- parentageID_: struct (nullable = true)
 |    |    |    |-- hash_: string (nullable = true)
 |    |    |-- transients_: struct (nullable = true)
 |-- EventSelections: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- hash_: string (nullable = true)
 |-- BranchListIndexes: array (nullable = true)
 |    |-- element: short (containsNull = true)
 |-- L1GlobalTriggerObjectMapRecord_hltL1GtObjectMap__HLT_: struct (nullable = true)
 |    |-- edm::EDProduct: struct (nullable = true)
```

```
In [6]: df.count()
Out[6]: 12058887

In [7]: slimmedEvents = df.select("recoMuons_muons__RECO_.recoMuons_muons__RECO_obj.reco::RecoCandidate.reco::LeafCandidate")
```

```
slimmedEvents.show()
```

```
+--------------------+
| reco::LeafCandidate|
+--------------------+
|[[[],-3,3.085807,...|
|[[[],3,4.1558356,...|
```

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

Test on stand-alone infrastructure

FILE FORMAT:	EOS TO HDFS COPY FOLDER (Disk)	HDFS TO SPARK-SHELL 2 EXECUTORS 4 CORES (Memory)	EOS TO SPARK-SHELL 2 EXECUTORS 4 CORES (Memory)
TEXT 200 GB CERNBOX	1 Gbit/s		300 Mbit/s
PARQUET 200 GB CERNBOX	800 Mbit/s		6 Gbit/s max 9 Gbit/s (full scan)
ROOT 200 GB CERNBOX			2.3 Gbit/s

- Dedicated Hadoop Cluster
  - 3 Nodes in total, 1 Namenode, 2 Datanodes
    - 3 x 10 Gb/s Network
    - 3 x 128 GB RAM
    - 3 x 32 cores Intel(R) Xeon(R) CPU E5-2650 @ 2.00GHz
- spark-root from HDFS and EOS works reasonable well
  - no show stoppers visible in stand-alone tests
  - But effects from different parts of the infrastructure (Spark, network, local disk, memory, number of files, …) not easy to disentangle

Cluster test - single executor

1 executor, 1 core/ executor, 1 task	HDFS to spark-submit		EOS to spark-submit	
Data Input	0.5 TB	1 TB	0.5 TB	1 TB
Time			~22,000 s ~6 h	~45,000 s ~12 h

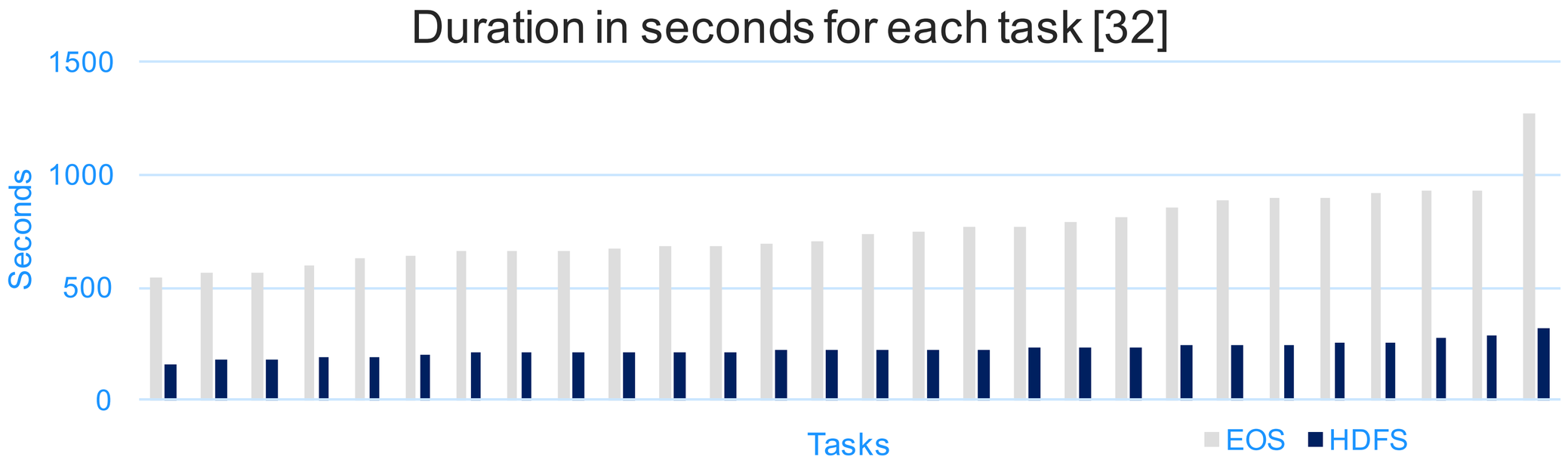
- Using shared analytix cluster at CERN
  - repository for monitoring data from IT end experiment dashboards, format conversion utilities, data analysis using Spark and map/ reduce (Pig) for dashboards, predictive traffic analysis
  - <some info about size of cluster>
- Single executor/thread test shows reasonable input volume scaling

Cluster test - data reduction

8 executor, 4 cores/ executor, 32 tasks	HDFS to spark-submit	EOS to spark-submit
Data Input	0.5 TB	0.5 TB
Total Task Time		6.1 h
Job Duration (= longest task duration)		21.2 min
Shortest task duration		9 min

- Cluster mode shows reasonable parallel scaling
  - Each task gets equal number of ROOT files
  - Variation in file size cause uneven task duration distribution

Cluster test - data reduction



$$r_{eos} = \frac{Average\ EOS\ Data}{sec} = \sim 22.2\ MB/s \qquad r_{hdfs} = \frac{Average\ HDFS\ Data}{sec} = \sim 78.8\ MB/s$$

- Conclusion: reading ROOT files in Spark works well
  - Throughput currently factor ~3 smaller than HDFS access
- Need to start disentangling impact of network, task fragmentation ➜ performance tuning

Summary & Outlook

- Solutions for the two largest caveats from CHEP 2016 study
  - Histogrammar allows filling histograms map-reduce style
  - spark-root allows to read ROOT files directly from Spark

- Next steps
  - Performance tuning of spark-root from EOS and HDFS
  - Investigate scaling behavior for larger and larger input volumes (goal is 1 PB)