HEP Data Processing with Apache Spark

Viktor Khristenko (CERN Openlab)

Outline

- HEP Data Processing
- ROOT I/O
- Apache Spark
- Data Ingestion
- Data Processing
- What's supported?!
- Internals and Optimizations
- Summary
- General Outlook

Important Note

- This talk is not about comparing ROOT File Format vs others (hdf5, parquet, avro, etc.).

- The goal of this work is to experiment with the available off-shell general purpose processing engines.

DEEP-EST Project

- DEEP - Extreme Scale Technologies.
- European Project aiming to build Modular Supercomputing Architecture.
- Exascale HPC.
- CERN Openlab is a collaborating partner.

HEP Data Processing

- c++ / python based
- ROOT I/O
- ROOT Histogramming Functionality
- Batch Processing - Custom Workload Distribution

ROOT I/O

- Columnar Data Format
- Very flexible and efficient!
- Self-descriptive - takes very few classes to bootstrap
- Storage of Arbitrary UDF classes
- Has both vector (SoA) and object (AoS) like layout for AoS depending on the internals.

Apache Spark

- General Purpose Processing Engine for both Batch and Streaming Processing
- lazy execution.
  - JVM bytecode codegen and execution per query.
- scala / java / python / R APIs
- Very similar API to TDataFrame, Panda's Dataframes.
- Easy scale-out of workflows.
- No additional boiler plate for managing batches.
  - Important for ML usually.

Data Ingestion: spark-root

0.1.15 on Maven Central!

- ROOT I/O for JVM.
  - A completely separate code base. Huge Thanks to ROOT Team: Axel/Danilo/Philippe!
  - There is almost 20-25 years old history of the JVM code base…
- Extends Spark's Data Source API.
- Represents ROOT TTree as DataFrame (Dataset[Row]) upon entry.
  - A single TTree => Dataset[Row]
- Parallelization = # files
  - Partitioning could be improved
- Implementation (Data Source) is modeled after parquet implementation.

Data Ingestion: spark-root

0.1.15 on Maven Central!

- Download spark's tar: https://spark.apache.org/downloads.html and unzip
- Start a scala shell:
  - ./bin/spark-shell --packages org.diana-hep:spark-root_2.11:0.1.15
- Or start a python shell:
  - ./bin/pyspark --packages org.diana-hep:spark-root_2.11:0.1.15
- Start analyzing/processing
- Straight-forward integration with Jupyter/Zeppelin Notebooks (any other ones..)

Data Ingestion: spark-root

<u>0.1.15 on Maven Central!</u>

Scala

```scala
// import the implicit DataFrameReader
import org.dianahep.sparkroot.experimental._

// read in a ROOT file
// select a TTree by name [optional]
// infer the schema
// Actual Data in the TTree is not read!
val df = spark
      .sqlContext
      .read
      .option("tree", "<treeName>")
      .root("<file,hdfs,root>:/path/to/files/*.root")
      //.parquet()
      //.csv()
      //.......
```

Python

```python
# read in a ROOT file
# select a TTree by name [optional]
# infer the schema
# Actual Data in the TTree is not read!
df = sqlContext\
      .read\
      .format("org.dianahep.sparkroot.experimental")\
      .load("<file,hdfs,root>:/path/to/files/*.root")
```

Data Ingestion: spark-root

0.1.15 on Maven Central!

DEEP Projects

Scala

// pretty print of the schema
df.printSchema

```
|-- Particle: array (nullable = true)
 |     |-- element: struct (containsNull = true)
 |     |     |-- fUniqueID: integer (nullable = true)
 |     |     |-- fBits: integer (nullable = true)
 |     |     |-- PID: integer (nullable = true)
 |     |     |-- Status: integer (nullable = true)
 |     |     |-- IsPU: integer (nullable = true)
 |     |     |-- M1: integer (nullable = true)
 |     |     |-- M2: integer (nullable = true)
 |     |     |-- D1: integer (nullable = true)
 |     |     |-- D2: integer (nullable = true)
 |     |     |-- Charge: integer (nullable = true)
 |     |     |-- Mass: float (nullable = true)
 |     |     |-- E: float (nullable = true)
 |     |     |-- Px: float (nullable = true)
 |     |     |-- Py: float (nullable = true)
 |     |     |-- Pz: float (nullable = true)
 |     |     |-- PT: float (nullable = true)
 |     |     |-- Eta: float (nullable = true)
 |     |     |-- Phi: float (nullable = true)
 |     |     |-- Rapidity: float (nullable = true)
 |     |     |-- T: float (nullable = true)
 |     |     |-- X: float (nullable = true)
 |     |     |-- Y: float (nullable = true)
 |     |     |-- Z: float (nullable = true)
 |-- Particle_size: integer (nullable = true)
```

Python

# pretty print of the schema
df.printSchema()

```
|-- Particle: array (nullable = true)
 |     |-- element: struct (containsNull = true)
 |     |     |-- fUniqueID: integer (nullable = true)
 |     |     |-- fBits: integer (nullable = true)
 |     |     |-- PID: integer (nullable = true)
 |     |     |-- Status: integer (nullable = true)
 |     |     |-- IsPU: integer (nullable = true)
 |     |     |-- M1: integer (nullable = true)
 |     |     |-- M2: integer (nullable = true)
 |     |     |-- D1: integer (nullable = true)
 |     |     |-- D2: integer (nullable = true)
 |     |     |-- Charge: integer (nullable = true)
 |     |     |-- Mass: float (nullable = true)
 |     |     |-- E: float (nullable = true)
 |     |     |-- Px: float (nullable = true)
 |     |     |-- Py: float (nullable = true)
 |     |     |-- Pz: float (nullable = true)
 |     |     |-- PT: float (nullable = true)
 |     |     |-- Eta: float (nullable = true)
 |     |     |-- Phi: float (nullable = true)
 |     |     |-- Rapidity: float (nullable = true)
 |     |     |-- T: float (nullable = true)
 |     |     |-- X: float (nullable = true)
 |     |     |-- Y: float (nullable = true)
 |     |     |-- Z: float (nullable = true)
 |-- Particle_size: integer (nullable = true)
```

Data Processing: Simple Example

- 50K events (rows) of 100 x 100 matrix
- Perform a total reduction
- 4GB uncompressed. ROOT file is ~106MB!

```
root
 |-- darr: array (nullable = true)
 |    |-- element: array (containsNull = true)
 |    |    |-- element: double (containsNull = true)
```

Scala

```scala
import org.dianahep.sparkroot.experimental._

// read in the file
val df = spark.sqlContext.read.root(inputFileName)

// cast each Row to a 2D Array
val ds = df.as[Seq[Seq[Double]]]

// Perform the reduction
ds.flatMap({case l => l.flatMap({case v => v})})
      .reduce(_ + _)
```

Python

```python
# read in the file
df = sqlContext.read\
       .format("org.dianahep.sparkroot.experimental")\
       .load(fileName)
# define a function to sum up
def sumUp(row):
    total = 0
    for arr in row.darr:
        total += sum(arr)
    return total
# perform map (transformation) and reduce (action)
df.rdd.map(sumUp).reduce(lambda x,y: x+y)
```

Data Processing: CMS Open Data Example

- CMS Public 2010 Muonia Dataset
- Hundreds of top columns
- Very complicated nestedness: AoS of AoS
- Tested on TBs of data across > 1K input files
  - on CERN's Analytix Cluster
- Transparent for scale-out. Just a glob operation
- http://opendata.cern.ch/record/10

- Calculate the invariant mass of a di-muon system and histogram

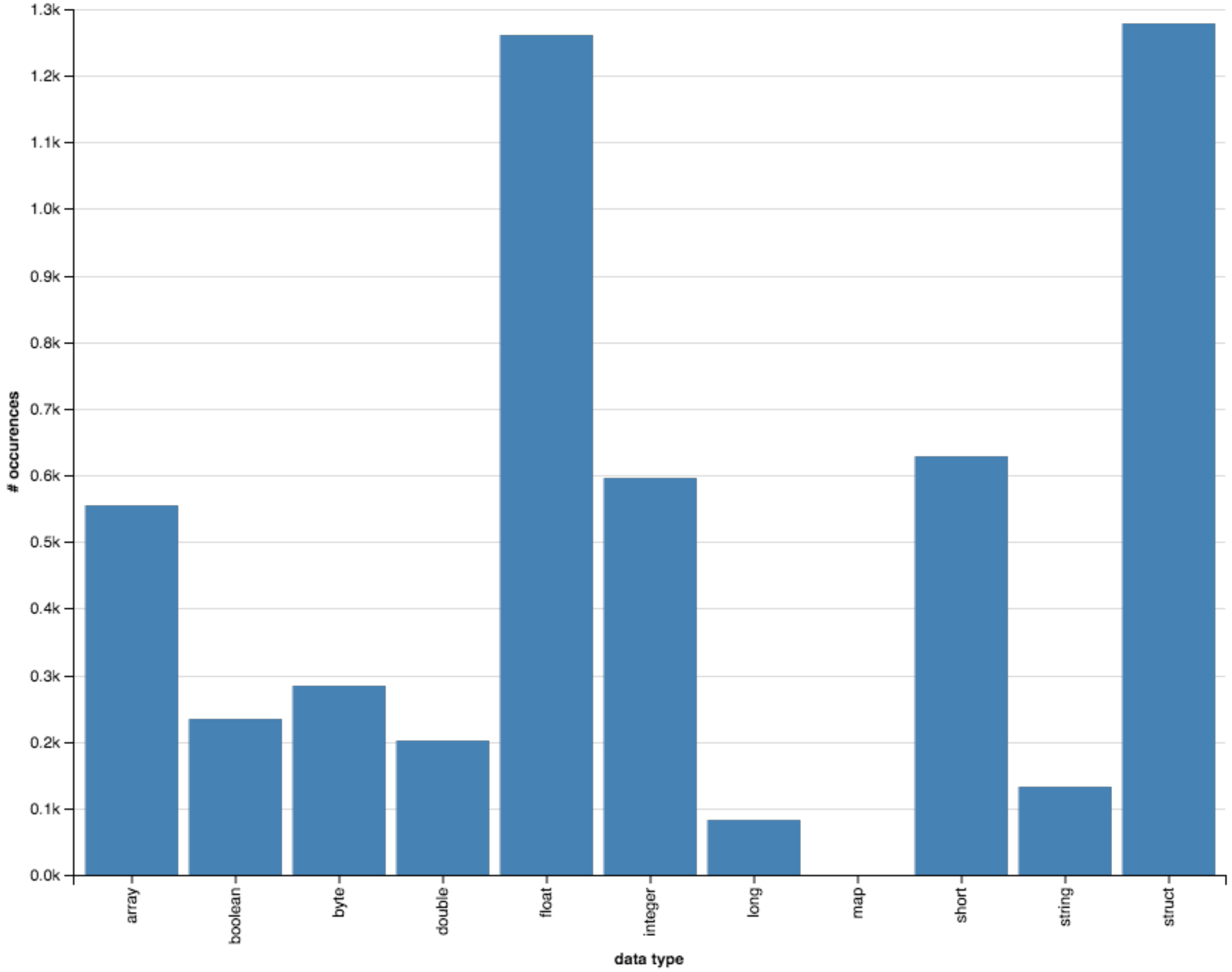
```
|-- patMuons_slimmedMuons__RECO_: struct (nullable = true)
 |    |-- present: boolean (nullable = true)
 |    |-- patMuons_slimmedMuons__RECO_obj: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- m_state: struct (nullable = true)
 |    |    |    |    |-- vertex_: struct (nullable = true)
 |    |    |    |    |    |-- fCoordinates: struct (nullable = true)
 |    |    |    |    |    |    |-- fX: float (nullable = true)
 |    |    |    |    |    |    |-- fY: float (nullable = true)
 |    |    |    |    |    |    |-- fZ: float (nullable = true)
 |    |    |    |    |-- p4Polar_: struct (nullable = true)
 |    |    |    |    |    |-- fCoordinates: struct (nullable = true)
 |    |    |    |    |    |    |-- fPt: float (nullable = true)
 |    |    |    |    |    |    |-- fEta: float (nullable = true)
 |    |    |    |    |    |    |-- fPhi: float (nullable = true)
 |    |    |    |    |    |    |-- fM: float (nullable = true)
 |    |    |    |    |-- qx3_: integer (nullable = true)
 |    |    |    |    |-- pdgId_: integer (nullable = true)
 |    |    |    |    |-- status_: integer (nullable = true)
```

Data Processing: CMS Open Data Example

- CMS Public 2010 Muonia Dataset
- Hundreds of top columns
- Very complicated nestedness: AoS of AoS
- Tested on TBs of data across > 1K input files
  - on CERN's Analytix Cluster
- Transparent for scale-out. Just a glob operation
- http://opendata.cern.ch/record/10

- Calculate the invariant mass of a di-muon system and histogram

Histogram of the Types present in the Schema



Data Processing: CMS Open Data Example

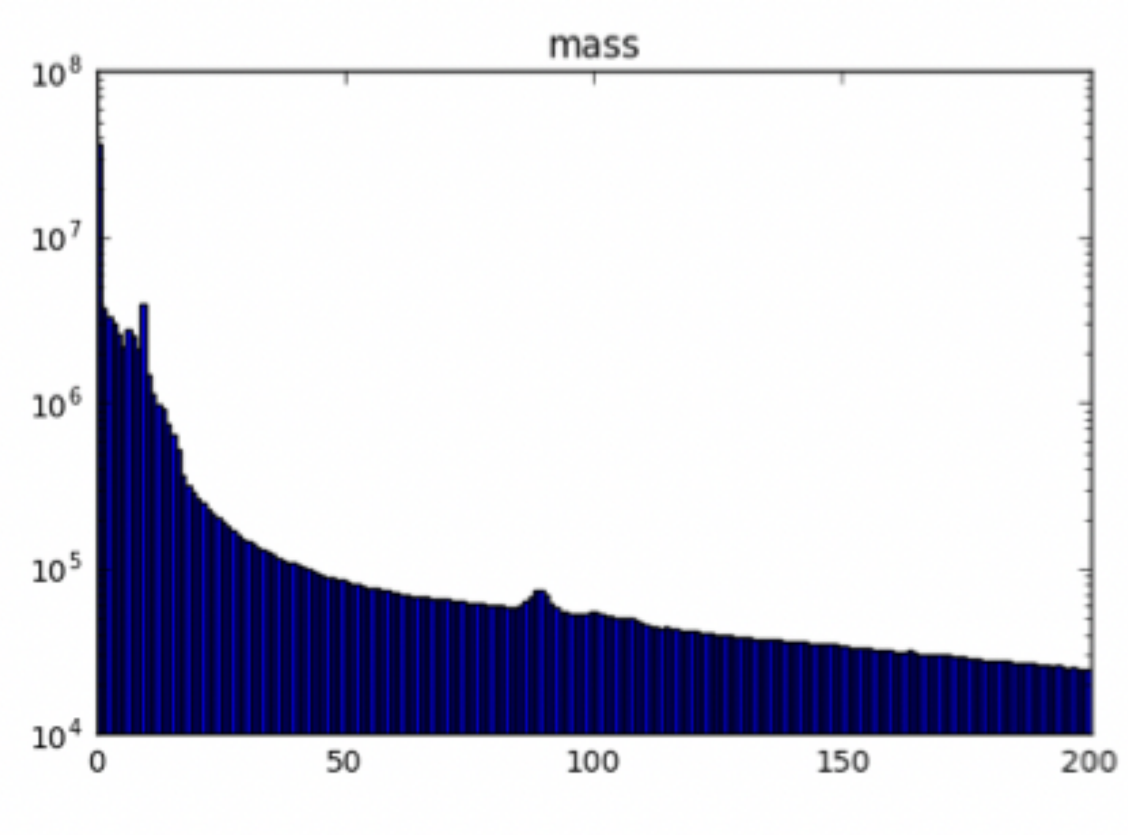
```
# read in the data
df = sqlContext.read\
        .format("org.dianahep.sparkroot.experimental")\
        .load("hdfs:/path/to/files/*.root")

# count the number of rows:
df.count()

# select only muons
muons =
df.select("patMuons_slimmedMuons__RECO_.patMuons_slimme
dMuons__RECO_obj.m_state").toDF("muons")

# map each event to an invariant mass
# inv_masses = muons.rdd.filter(lambda row: row.muons.size==2)
inv_masses = muons.rdd.map(toInvMass)

# Use histogrammar to perform aggregations
empty = histogrammar.Bin(200, 0, 200, lambda row: row.mass)
h_inv_masses = inv_masses.aggregate(empty,
        histogrammar.increment,
        histogrammar.combine)
```



https://github.com/diana-hep/spark-root/blob/master/ipynb/publicCMSMuonia_exampleAnalysis_wROOT.ipynb

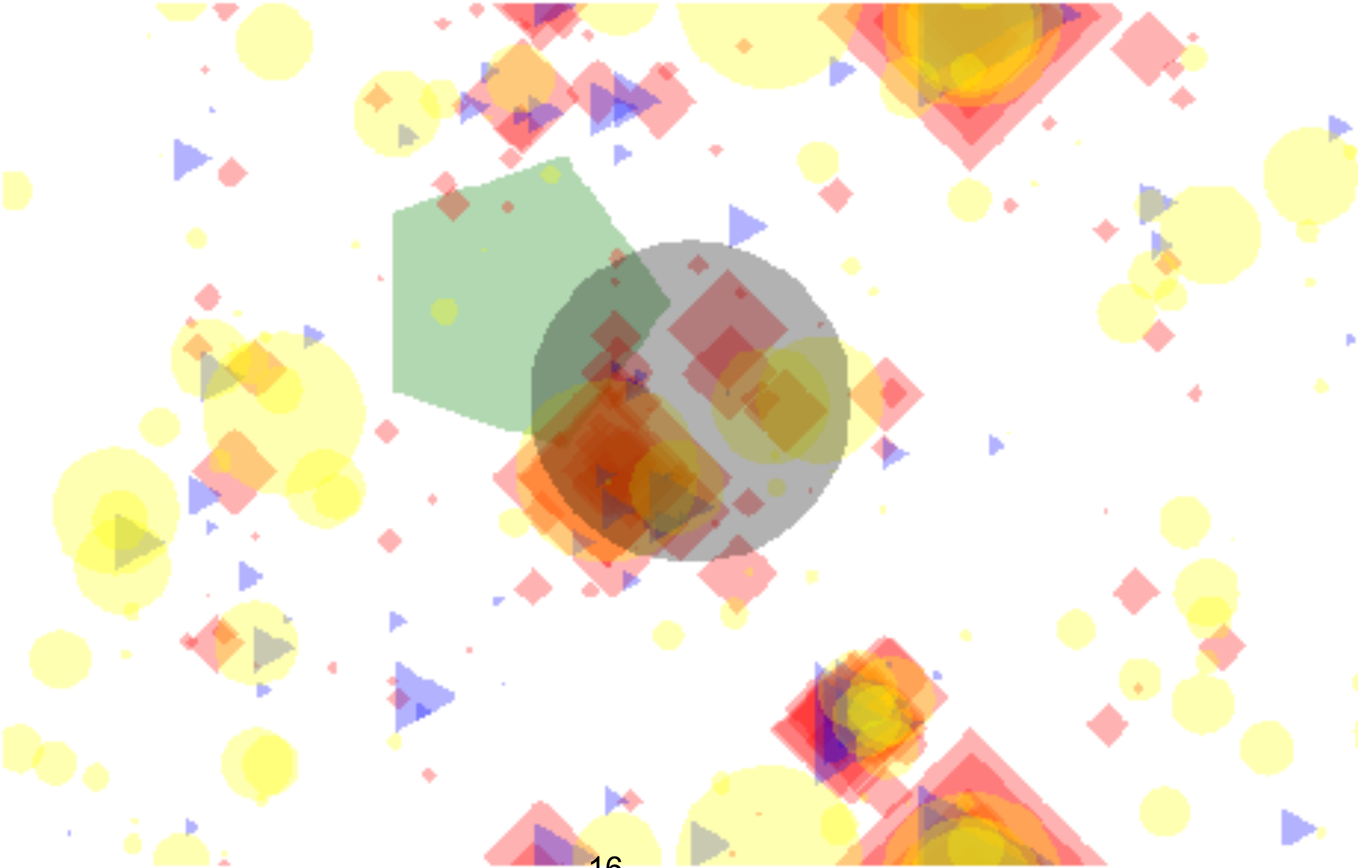
Data Processing: Feature Engineering

- Simulated Events with:
  - Tracks, Hadrons, Photons, Electrons, Muons
- A glimpse of the input schema:

```
|-- Particle: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- fUniqueID: integer (nullable = true)
 |    |    |-- fBits: integer (nullable = true)
 |    |    |-- PID: integer (nullable = true)
 |    |    |-- Status: integer (nullable = true)
 |    |    |-- IsPU: integer (nullable = true)
 |    |    |-- M1: integer (nullable = true)
 |    |    |-- M2: integer (nullable = true)
 |    |    |-- D1: integer (nullable = true)
 |    |    |-- D2: integer (nullable = true)
 |    |    |-- Charge: integer (nullable = true)
 |    |    |-- Mass: float (nullable = true)
 |    |    |-- E: float (nullable = true)
 |    |    |-- Px: float (nullable = true)
 |    |    |-- Py: float (nullable = true)
 |    |    |-- Pz: float (nullable = true)
 |    |    |-- PT: float (nullable = true)
 |    |    |-- Eta: float (nullable = true)
 |    |    |-- Phi: float (nullable = true)
 |    |    |-- Rapidity: float (nullable = true)
 |    |    |-- T: float (nullable = true)
 |    |    |-- X: float (nullable = true)
 |    |    |-- Y: float (nullable = true)
 |    |    |-- Z: float (nullable = true)
 |-- Particle_size: integer (nullable = true)
```

https://github.com/vkhristenko/MPJRPipeline/blob/master/ipynb/preprocessing_python_noudfs.ipynb

- For each event, build a 2D matrix of features from
  - N tracks/hadrons/photons/1lepton
- For each such matrix, build an image and train:



https://github.com/vkhristenko/MPJRPipeline/blob/master/ipynb/convert2images_python.ipynb

Data Processing: Feature Engineering

- Simulated Events with:
  - Tracks, Hadrons, Photons, Electrons, Muons
- Pipeline is quite simple:

Step1:

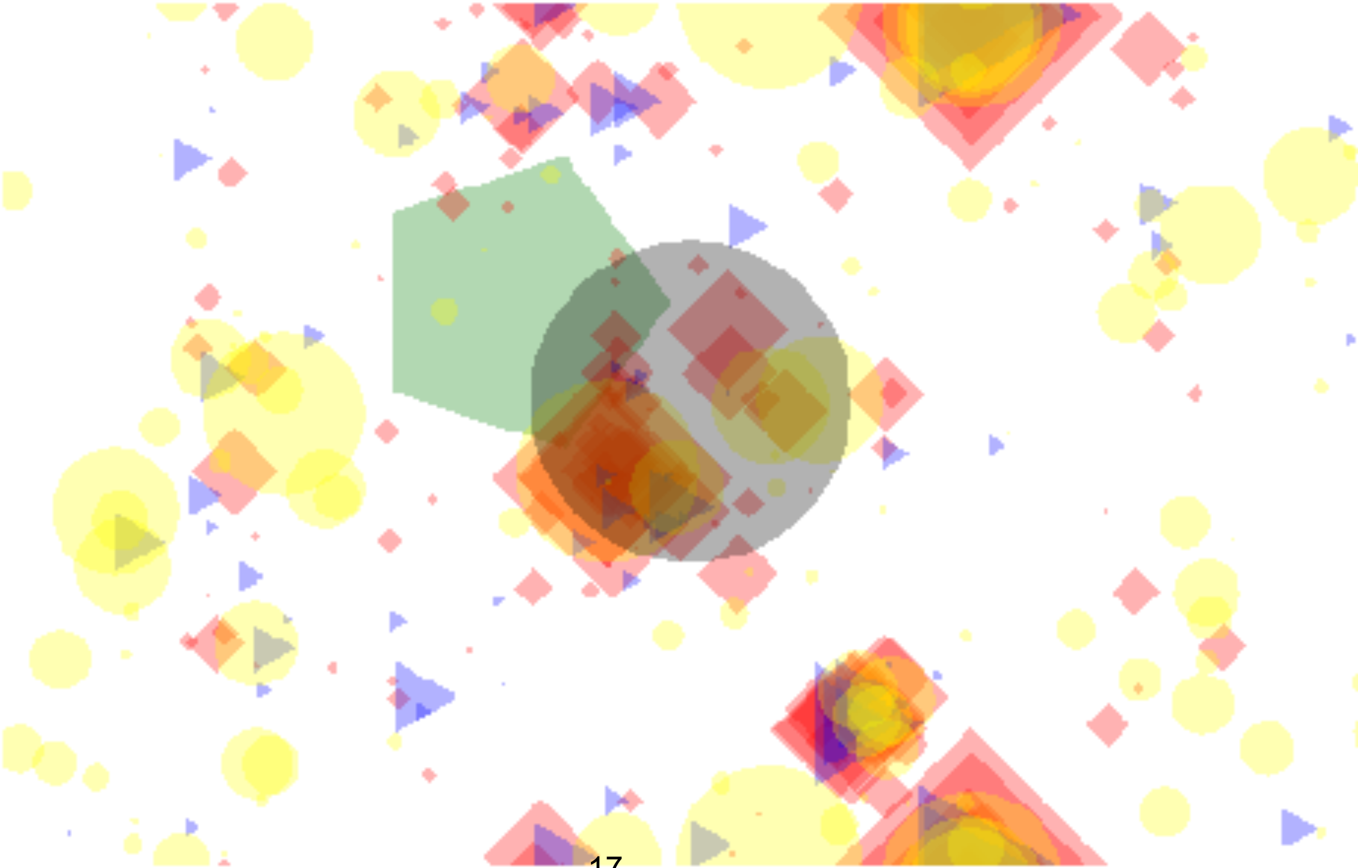
```
features = events\
    .limit(1000)\
    .rdd\
    .map(convert)\
    .filter(lambda row: len(row) > 0)\
    .toDF()
```

Step2:

```
images = features\
    .rdd\
    .map(convert2image)\
    .toDF()
```

https://github.com/vkhristenko/MPJRPipeline/blob/master/ipynb/preprocessing_python_noudfs.ipynb

- Step1: For each event, build a 2D matrix of features from
  - N tracks/hadrons/photons/1lepton
- Step2: For each such matrix, build an image and train:



https://github.com/vkhristenko/MPJRPipeline/blob/master/ipynb/convert2images_python.ipynb

What’s __not__ well supported for ROOT I/O

- Pointers: Anything that requires Run (read time) Time Type Inference!
  - e.g. TClonesArray that do not occupy a “splitted" branch
- Most prominent example:

```
class Base {…};
class Derived : public Base {…};
std::vector<Base*> someP2BaseVector;
```

- Most of the STL containers are supported (e.g. bitset).
- **Apache Spark requires that the schema is known before the actual Query Plan is built!**

Avoiding what's not supported

- CMSSW RECO/AOD/MINIAOD are one of the most complex examples of ROOT files.
- Typical content is a bunch of UDF Classes + STL Containers.
  - std::vector<framework::Particle>
  - class Particle : public Parent { … std::map<std::string, std::vector<framework::Hits> > };
  - All of that works!
- <u>Pointers are present but rare.</u>

- A set of optimizations were included to prune away __RunTime__ Types.

Internals: spark-root

- Bootstrapping - a set of classes with predefined streaming logic.
  - TKey, TFile…
- Byte Code Engineering Library (bcel) is used for JIT compilation of ROOT classes
- root4j is the java code base that implements above
  - Created by Tony Johnson
  - >20 years of history - very old code base.
- Has been revived and bug fixed for proper reading of ROOT files
- spark-root builds on top of root4j and implements the proper TTree reading.
  - scala code-base.

Optimizations: spark-root

- Internally:
  - TTree => IR schema => Spark Schema (Struct Type)
- Several Optimizations are performed on the IR schema
  - Nested Column Pruning (with https://github.com/apache/spark/pull/16578)
    - once this PR is in, we will need to push an update on top to spark's master.
    - PR assumes parquet usage only, but has been tested to apply to our Data Source as well
  - Empty Rows Removal (parquet does not allow empty Groups!)
  - Flatten out Base Classes
  - Removal of Run Time Types (pointers) and Unknown/Null types.
    - It's possible that some types are not available: enums, hard-coded streaming logic.

Anyone using spark-root?

- Given ROOT files => you can use it… no installation of anything.
    - No need for Class Dictionaries…
    - For Spark Applications - no special compilation procedures.
- Jars are on Maven Central.

- CMS Big Data Project
    - Applying Apache Spark for processing of CMS Data
    - Open Data Muonia Example Workflow

- Feature Engineering / ML Training
    - Experimenting myself with using Apache Spark + ML Frameworks on top
    - dist-keras, BigDL - anything that plugs on top.

Summary

- spark-root - Spark's Data Source for ROOT File Format.

- Works!
  - but currently has limitations.

- Very easy to use - no special knowledge - just use standard Apache Spark API.

- Very easy to get started - no installation.
  - You do not have to install Scala or SBT!

- Very easy to scale out

General Outlook

- Nothing has been said about __current__ Apache Spark performance.
  - Good scale-out
  - Bad single thread performance

- Apache Spark is (<u>seems to be</u>) optimized for simple table structure
  - For deeply nested structures like collection of physics objects -> <u>not optimal. A lot of overhead!</u>
  - Databricks have additions to SQL for High Order Functions
  - But they are not in spark/master…

- Very easy to port python based analyses (w/ or w/o ROOT)
  - copy/paste and run!
  - On Analytix we could even use ROOT Physics Classes since it’s visible across all the nodes.
    - TLorentzVector…

General Outlook

- Apache Spark is young technology
- Quite Flexible Codebase
- Flare: flaredata.github.io
  - Native Compilation of the Query Plan!
  - No JVM overheads!
- scala-native: https://github.com/scala-native/scala-native
  - scala-native = clang on top of LLVM - FrontEnd Compiler for Scala.
  - Runs as fast as c++ based processing.
  - Early stages of dev - but does work! Developed by Scala Center at EPFL!
  - scala Language -> Multiple Compier FrontEnds: scala-js (JS in Browser) / scala-native (Native Executable) / scala (JVM)

The DEEP projects DEEP, DEEP-ER and DEEP-EST have received funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under grant agreement no ICT-610476 and no ICT-287530 as well as the Horion2020 funding framework under grand agreement no. 754304.