In this poster, we present our experience of evaluating Apache Spark for High Energy Physics (HEP) analyses. Our goal is to understand how well this technology performs for HEP-like analyses in both HPC and Hadoop ecosystem. We use an example from the Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider (LHC) in Geneva, Switzerland, the highest energy particle collider in the world. The CMS detector consists of detector components that measure different properties of the particles produced in a collision, such as tracks left by charged particles and energy deposits from all particles that interact via photons and gluons. Our use case focuses on searching for new types of elementary particles explaining Dark Matter in the universe. In particular, this search is looking for a signature in the events commonly referred to as "mono-X" where "X" can be a light quark or gluon, a vector boson, or a heavy quark such as a bottom or top quark. We focus our search on the “monoTop” signature, where the detectable particle is a single, unbalanced top quark.

Two C++ frameworks are used for the traditional end user analysis: CMSSW, specially designed for analyzing CMS data, and ROOT framework provides statistical tools and a serialization format to persist data. Although these C++ frameworks can be very efficient, using them efficiently remains challenging for end-users.

Most data analysts or analysis groups start by translating the class structure of the event data into a "flat ntuple," in which events are rows of a table with primitive numbers or arrays of numbers as columns representing different particles (photons, electrons, taus, etc) and their properties. Often, the ntuples are still too big for interactive analysis (~ 2 TB). Most analysis groups therefore introduce additional steps in which the ntuples themselves are skimmed and slimmed (~GBs). In the last step of the analysis, quantities from the final ntuple are aggregated and plotted as histograms. The time scale of the complete Dark Matter workflow, shown in Fig.~\ref{tadaaa}, can range from days to weeks, depending on how many reconstructed and simulated events are needed for the analysis.

We provide different implementations of this analysis workflow; one using Spark on the Hadoop ecosystem, and the other using Spark on high performance computing platforms.

*Spark on Hadoop:* We developed a library to convert the official experiment data files in ROOT to Apache Avro row-based format readable by Spark. The skimming and slimming on the input data is implemented by using Spark’s map, flatMap and filter transformations. We use Apache Parquet columnar format to store the intermediate results between stages of the calculation in HDFS, allowing to easily ingest data into Dataframes and Datasets. We use modern Dataset and Dataframe   
APIs taking advantage of advanced Catalyst query optimizer and direct operations on serialized data, available in Spark 2.0.

*Spark on HPC:* We wrote a converter in Python to convert the official experiment data files in ROOT to the HDF5 format, the main reason is that HDF5 is well supported at many HPC platforms including NERSC, and will allow us to use other HPC programming interfaces (e.g. MPI) on the same set of files. Our approach was to convert each branch representing a particle (e.g. Tau, Photon, etc) in an event to a group in HDF5, and each leaf representing a property (e.g. pt, eta, phi) with in a branch to 1D dataset in HDF5. We stored the highly structured data into flat tables and column-oriented structure to allow distributed processing across groups as needed. We implemented a customized HDF5 reader in Spark/Scala to read in an HDF5 group with specified datasets into a Spark DataFrame. The data partition is decided based on the number of elements in each HDF5 dataset per group across all the input files; we use HDF5 hyper slabs to read in chunks to allow maximal parallelism while reading the data into DataFrames. We implemented the skimming and slimming code in Scala for the converted HDF5 files; we directly operate on the Spark DataFrames and use SQL queries.

We ran the following test on each platform: to keep the number of processing cores same across NERSC and Princeton big data cluster, in the preliminary testing we only operated on 140 cores. The analysis is multi-pass, and requires sum of event weights as input to the analysis code. The input data we used was about ~170 GB ROOT and Avro, however only 46 GB in HDF5 because only the branches needed in this analysis were converted and compressed in the HDF5 format.

On Edison, we used 7 nodes, 20 cores, each node is 64GB and the executor memory is specified as 58GB. We ran 10 iterations: after caching and the first iteration, on average it is taking about 2.0 seconds to calculate the sums. The same test on Princeton big data cluster takes about 14 seconds. The sample data set we used has 37937742 events, and it takes about 1.7 seconds to count the number of events on Edison. On Edison, we were generating 7 files, one per particle and two for the event info. It takes about 1-6 minutes to write all the DataFrames after all the filtering (cuts) for this data sample.

Spark is relatively new and emerging technology, and its use especially in the HEP community is in exploratory stages. The learning curve involved with the use of this new technology especially using Scala cannot be ignored. However, the availability of APIs in R and Python improves the user experience to get started with the system. Other advantages include automatic data and task distribution based on the number of partitions of data sets; users can also control the partitioning. We have seen good scaling behavior of Spark applications with increase in dataset size and the number of nodes with no extra work. It is important to choose among RDDs or DataFrame or DataSet APIs for the optimal and clear design of analysis task. Working with several large DataFrames is inconvenient, and involves operations such as join, which have performance penalties. As with any new technology, the inadequate documentation, unhelpful error messages, lack of expert consultations remain critical issues. However, the ease of use, reasonable performance and good scalability makes Spark a viable candidate for our future work.