Fermilab U.S. DEPARTMENT OF ENERGY | Office of Science

Spark and HPC for High Energy Physics Data Analyses

Marc Paterno, Jim Kowalkowski, and Saba Sehrish

2017 IEEE International Workshop on High-Performance Big Data Computing

Introduction

- High energy physics (HEP) data analyses are data-intensive; 3×10^{14} particle collisions at the Large Hadron Collider (LHC) were analyzed in Higgs boson discovery.
- Most analyses involve compute-intensive statistical calculations.
- Future experiments will generate significantly larger data sets.

Our question

Can “Big Data” tools (e.g. Spark) and HPC resources benefit HEP’s data- and compute-intensive statistical analysis to improve time-to-physics?

A physics use case: search for Dark Matter

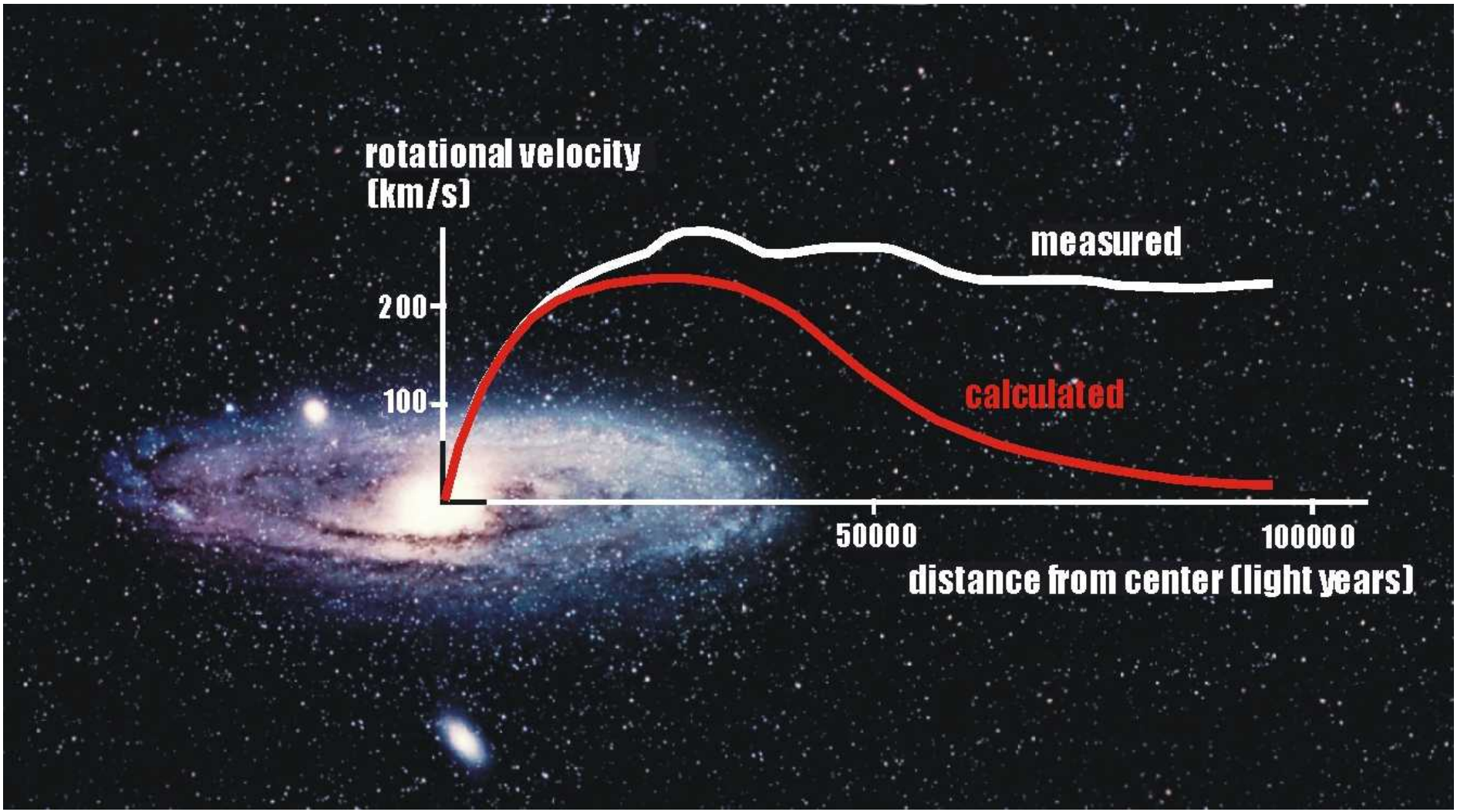
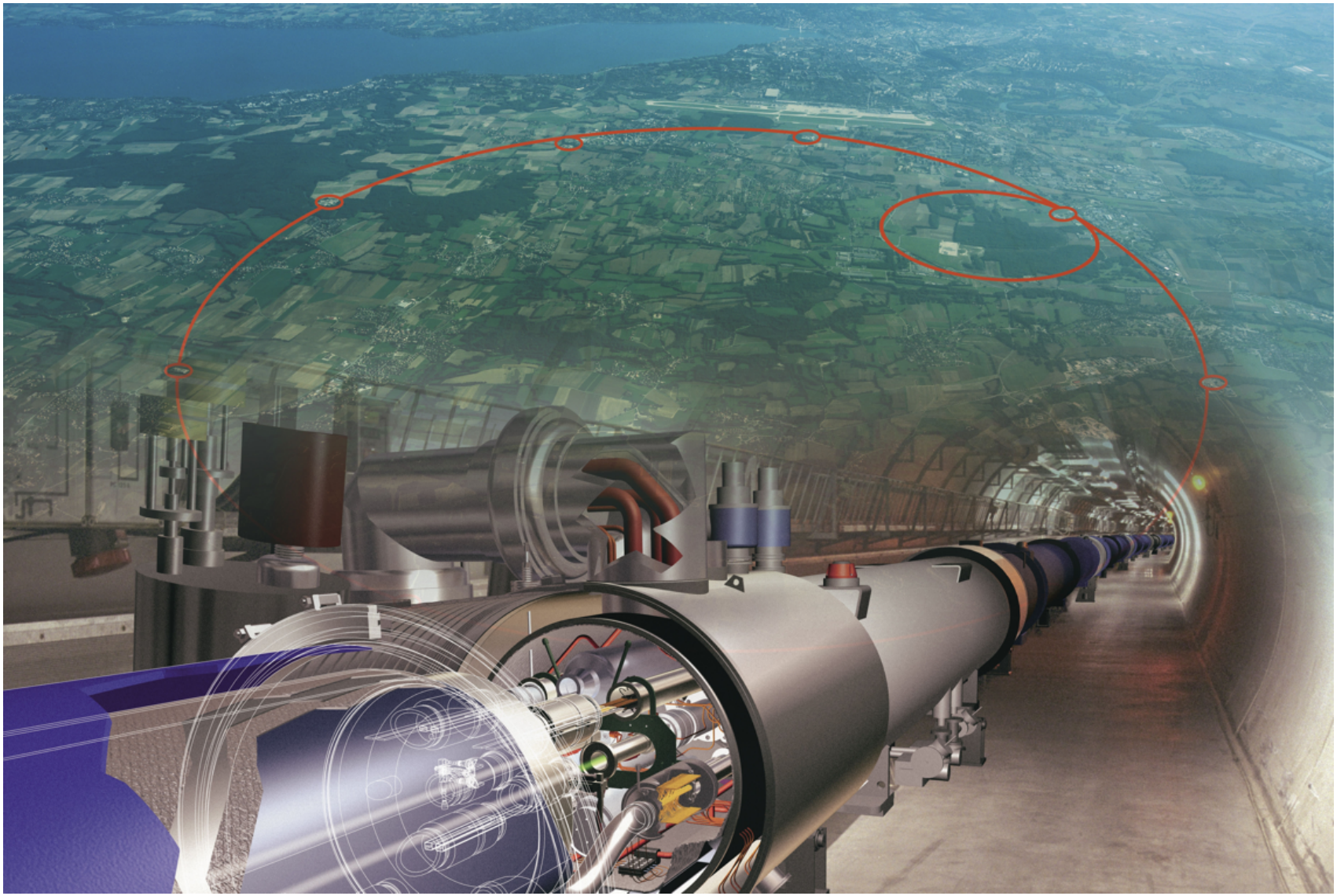
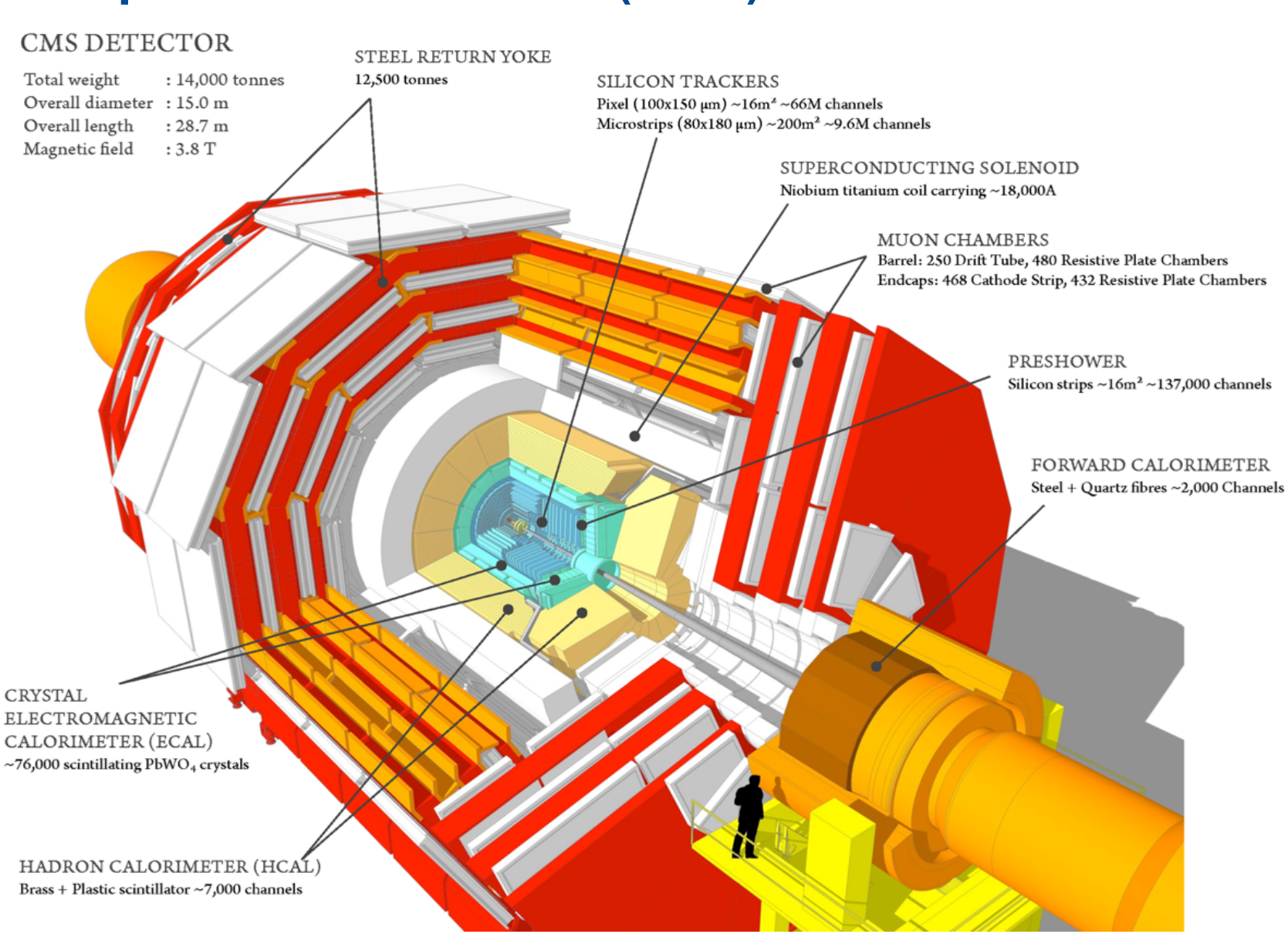


Image from http://cdms.phy.queensu.ca/PublicDocs/DM_Intro.html.

The Large Hadron Collider (LHC) at CERN

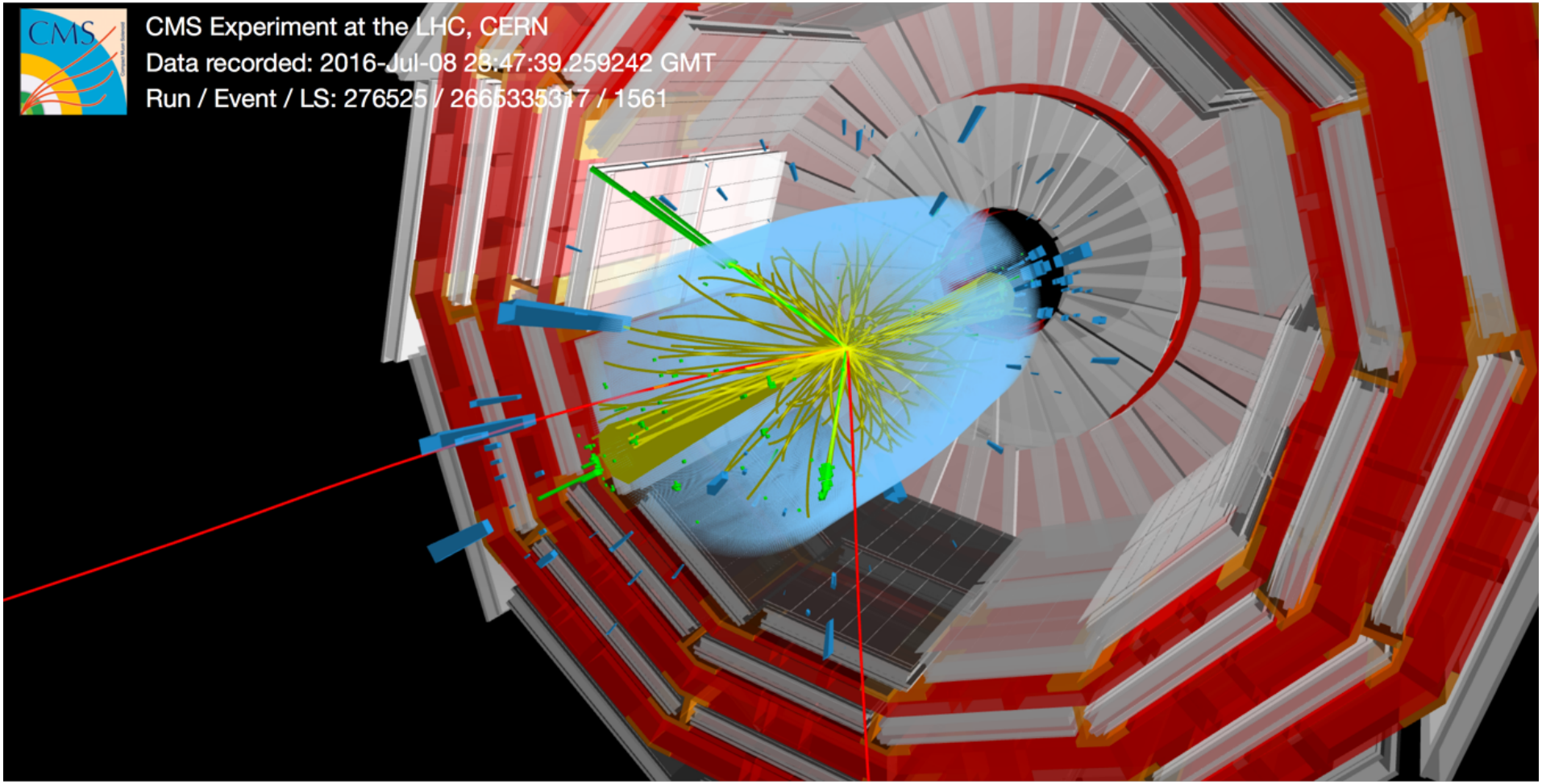


The Compact Muon Solenoid (CMS) detector at the LHC

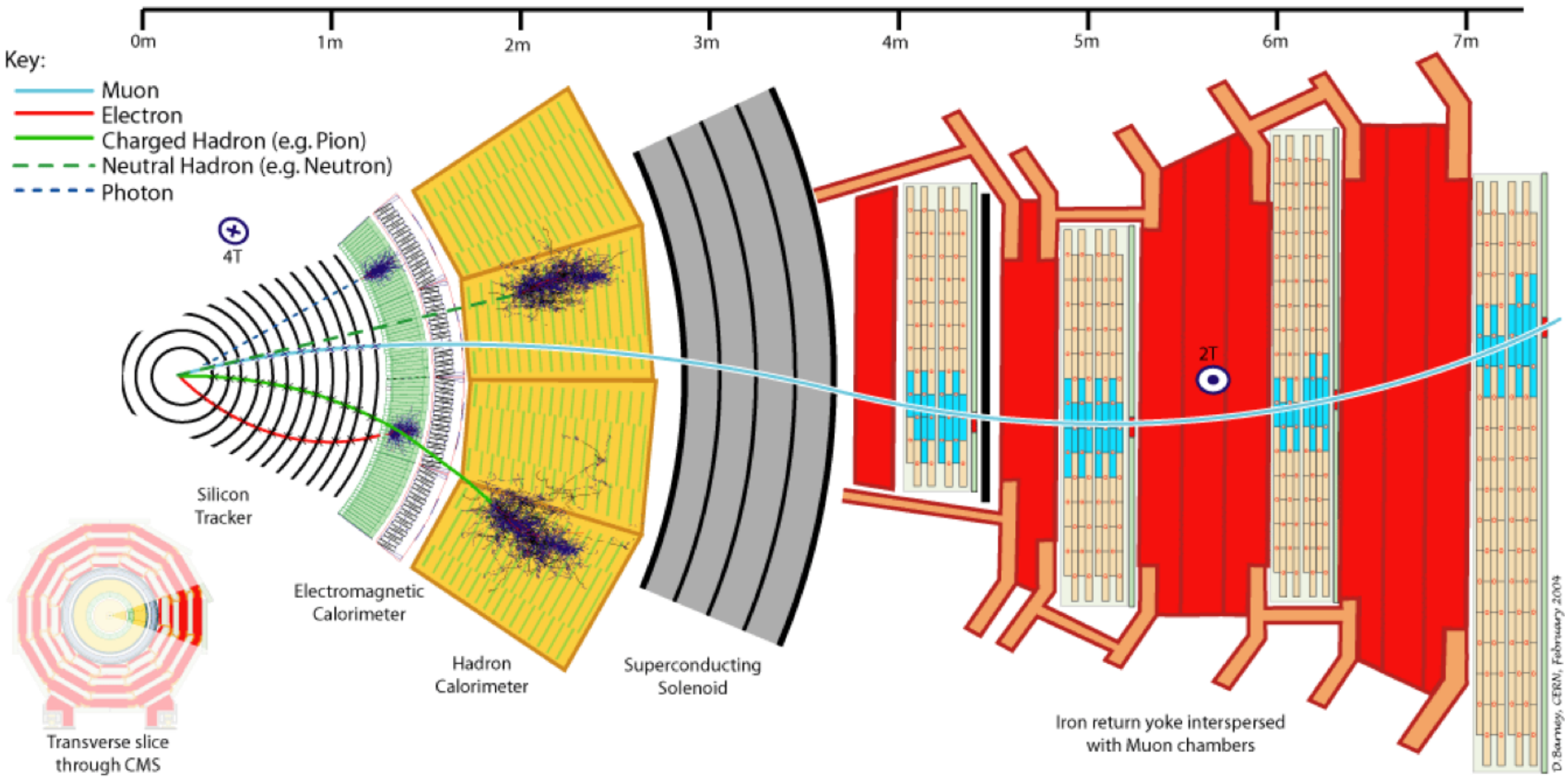


Fermilab

A particle collision in the CMS detector

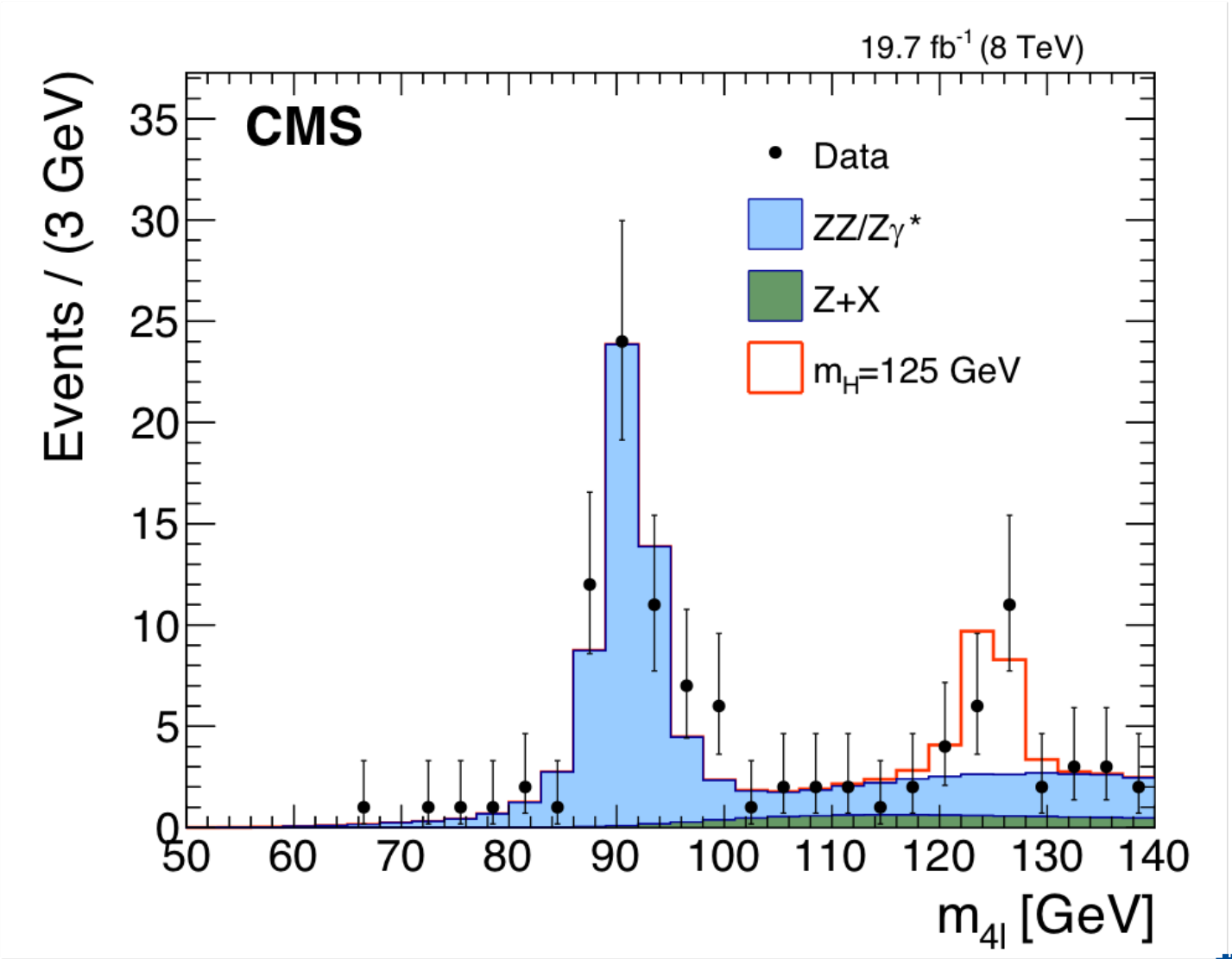


How particles are detected



Fermilab

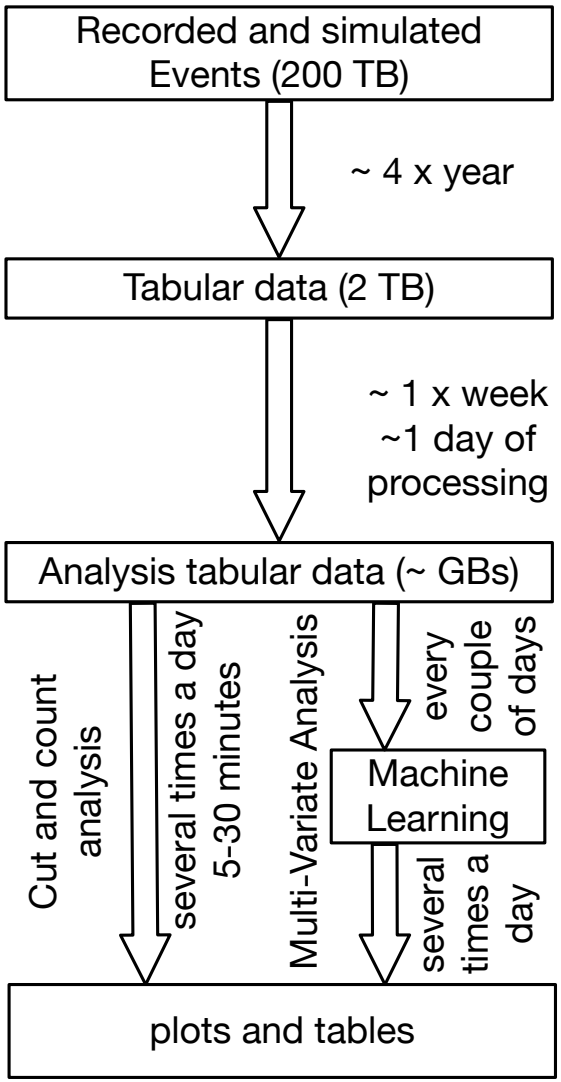
Statistical analysis: a search for new particles



Fermilab

The current computing solution

- Whole-event based processing, sequential file-based solution
- Batch processing on distributed computing farms
- 28,000 CPU hours to generate 2 TB tabular data, $\sim$ 1 day of processing to generate GBs of analysis tabular data, 5–30 minutes to run end-user analysis
- Filters used on analysis data to:
  - Select interesting events
  - Reduce the event to a few relevant quantities
  - Plot the relevant quantities



Why Spark might be an attractive option

- In-memory large-scale distributed processing: *Resilient distributed datasets* (RDDs): collections of data partitioned across nodes, operated on in parallel
- Able to use parallel and distributed file system
- Write code in a high level language, with implicit parallelism
  - Spark SQL: a Spark module for structured data processing.
  - DataFrame: a distributed collection of rows organized into named columns, an abstraction for optimized operations for selecting, filtering, aggregating and plotting structured data.
- Good for repeated analysis performed on the same large data
- Lazy evaluation used for transformations, allowing Spark's Catalyst optimizer to optimize the whole graph of transformations before any calculation
  - *Transformations* map input RDDs into output RDDs; *actions* return the final result of an RDD calculation
- Tuned installation available on (some) HPC platforms

HDF5: essential features

- Tabular data representable as columns (datasets) in tables (groups).
- HDF5 is a widely-used format for the HPC systems; this allows us to use traditional HPC technologies to process these files.
- Parallel reading supported

Overview: computing solution using Spark and HDF5

- Read HDF5 files into multiple DataFrames, one per particle type.
  - First, we had to translate from the standard HEP format to HDF5.
- Define filtering operations on a DataFrame as a whole (as opposed to writing loops over events).
- Data are loaded once in memory and processed several times.
- Make plots, repeat as needed.

Simplified example of data

Standard HEP event-oriented data organization.

Event ID	Event Info		Electrons		Taus	
1	met	weight	pt	eta	pt	eta
	150	0.5	130	0.4	17	0.1
			50	1.3	55	0.3
					44	1.9
2	met	weight	pt	eta	pt	eta
	210	0.65	67	-0.5	34	1.5
			87	1.9	44	0.3

Tabular organization

Event Info

Event ID	met	weight
1	150	0.5
2	210	0.65

Electrons

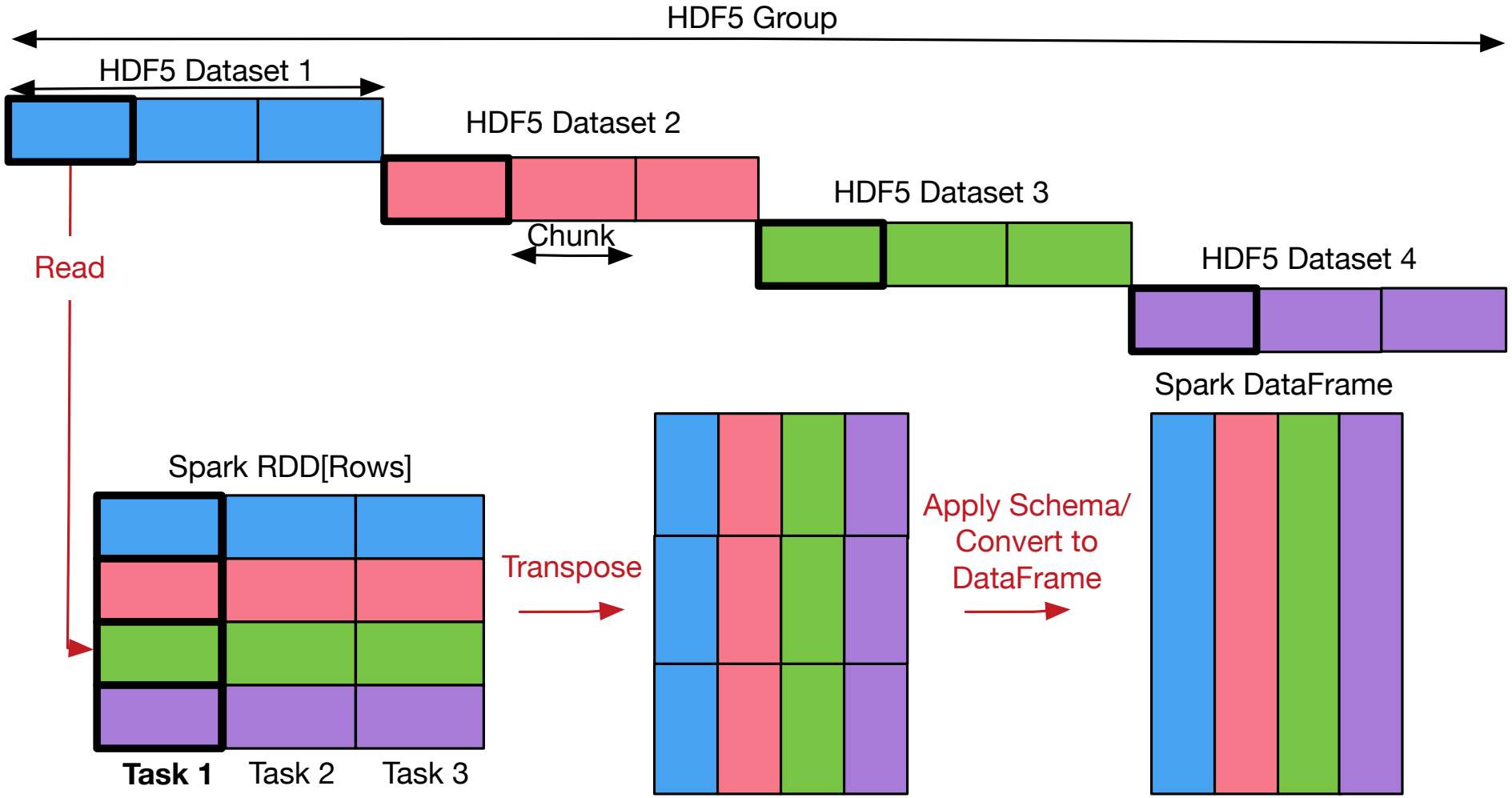
Event	pt	eta
1	130	0.4
1	50	1.3
2	67	-0.5
2	87	1.9

Taus

Event	pt	eta
1	17	0.1
1	55	0.3
1	44	1.9
2	34	1.5
2	44	0.3

Reading HDF5 files into Spark

The columns of data are organized as we want them in the HDF5 file, but Spark provides no API to read them directly into DataFrames.



Fermilab

An example analysis

- Find all the events that have:
  - missing E_T (an event-level feature) greater than 200 GeV
  - one or more electrons candidates with
    - $p_T > 200$ GeV
    - eta in the range of -2.5 to 2.5
    - good "electron quality": qual > 5
- For each selected event record:
  - missing E_T
  - the leading electron p_T

Some observations

- Range queries across multiple variables are very frequent
- Hard to describe by just using SQL declarative statements
- Relational databases that we are familiar with are unable to efficiently deal with these types of queries

Coding a physics analysis with the DataFrame API

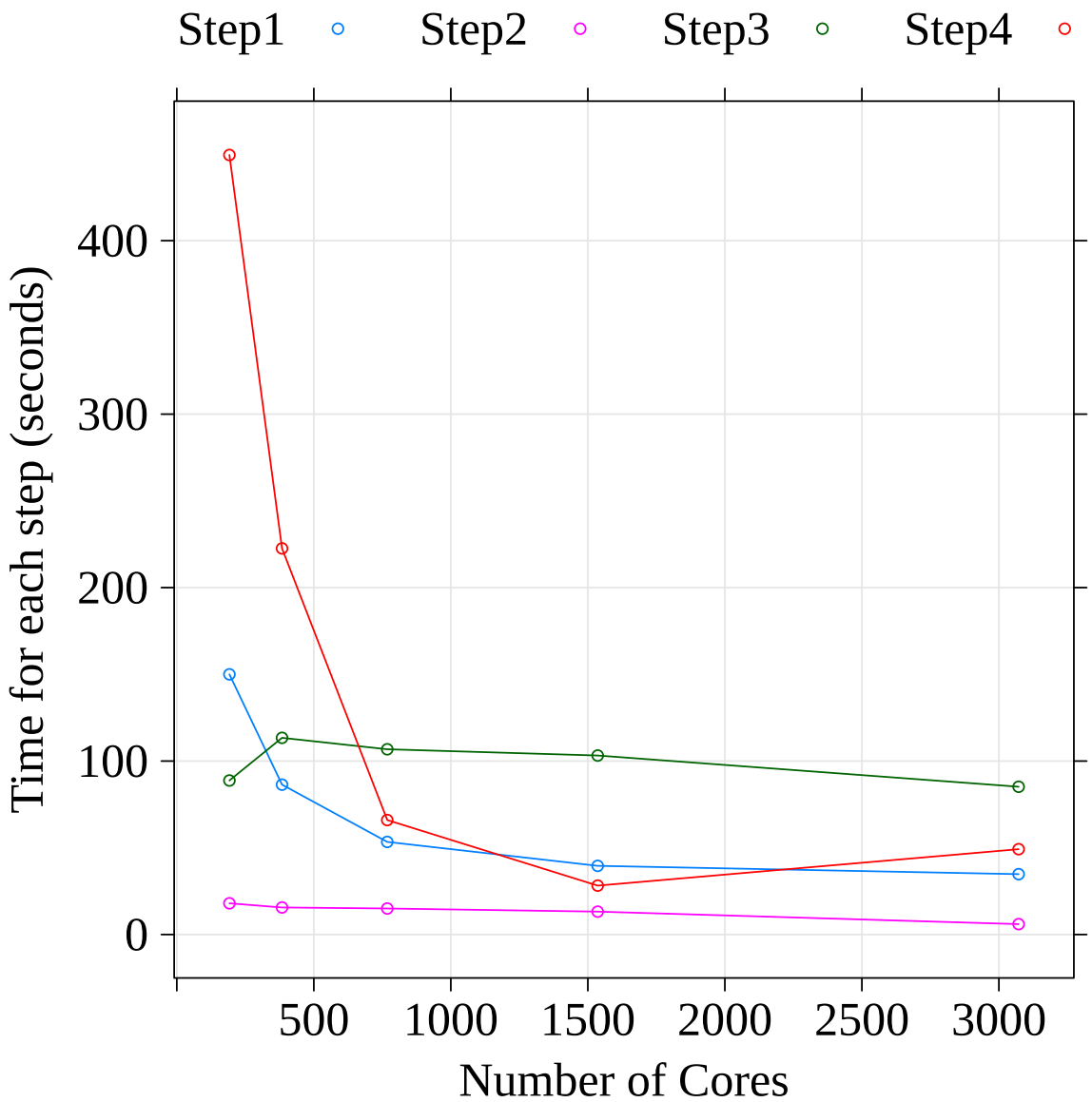
```
val good_electrons =
  electrons.filter("pt">200)
           .filter(abs("eta")<2.5)
           .filter("qual">5)
           .groupBy("event")
           .agg(max("pt"),"eid")

val good_events =
  events.filter("met">200)

val result_df =
  good_events.join(good_electrons)
```

Using `result`, make a histogram of the p_T of the "leading electron" for each good event.

Measuring the performance

- The real analysis we implemented involves much more complicated selection criteria, and many of them. It required the use of *user defined functions* (UDFs).
- In order to understand where time is spent by Spark, we determined
  - the time to read from HDF5 file into RDDs (*step 1*)
  - the time to transpose RDDs (*step 2*)
  - the time to create DataFrames from RDDs (*step 3*)
  - the time to run analysis code (*step 4*)
- Tests run on Edison at NERSC, using Spark v2.0.
- Tested using 8, 16, 32, 64, 128 and 256 nodes.
- Input data consists of
  - 360 million events
  - 200 million electrons
  - 0.5 TB in memory

Scaling results



- *Steps 1–3* read the files and prepare the data in memory. Different steps exhibit different (or no) scaling.
- *Step 4* is performing the analysis on in-memory data.

Lessons learned

The goal of our explorations is to shorten the time-to-physics for analysis.

- We have observed good scalability and task distribution.
- However, absolute performance does not yet meet our needs.
- It is hard to tune a Spark system:
  - Optimal number of executor cores, executor memory, *etc*.
  - Optimal data partitioning to use with the parallel file system, *e.g.*, Lustre File System stripe size, OST count.
  - Difficult to isolate slow performing stages due to lazy evaluation.
- *pySpark* and *SparkR* high-level APIs may be appealing to the HEP community.
- Our understanding of Scala and Spark best practices is still evolving.
- Documentation and error reporting could be improved.

Future work

- Scale up to multi-TB data sets.
- Compare performance with a Python+MPI approach.
- Improve our HDF5/Spark middleware.
- Evaluate the I/O performance of different file organizations, *e.g.* all backgrounds in one HDF5 file.
- Optimize the workflow to filter the data: try to remove UDFs, which prevents Catalyst from performing optimizations.

References

1. Kowalkowski, Jim, Marc Paterno, Saba Sehrish:*Exploring the Performance of Spark for a Scientific Use Case*. In IEEE International Workshop on High-Performance Big Data Computing.In conjunction withThe 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2016).
2. LHC http://home.cern/topics/large-hadron-collider
3. CMS http://cms.web.cern.ch
4. HDF5 https://www.hdfgroup.org
5. Spark at NERSC
   http://www.nersc.gov/users/data-analytics/data-analytics/spark-distributed-analytic-framework
6. Traditional analysis code:
   https://github.com/mcremone/BaconAnalyzer
7. Our approach:
   https://github.com/sabasehrish/spark-hdf5-cms

Acknowledgments

- We would like to thank Lisa Gerhardt for guidance in using Spark optimally at NERSC.
- This research supported through the Contract No. DE-AC02-07CH11359 with the United States Department of Energy 2016 ASCR Leadership Computing Challenge award titled “An End- Station for Intensity and Energy Frontier Experiments and Calculations”.
- This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02- 05CH11231.