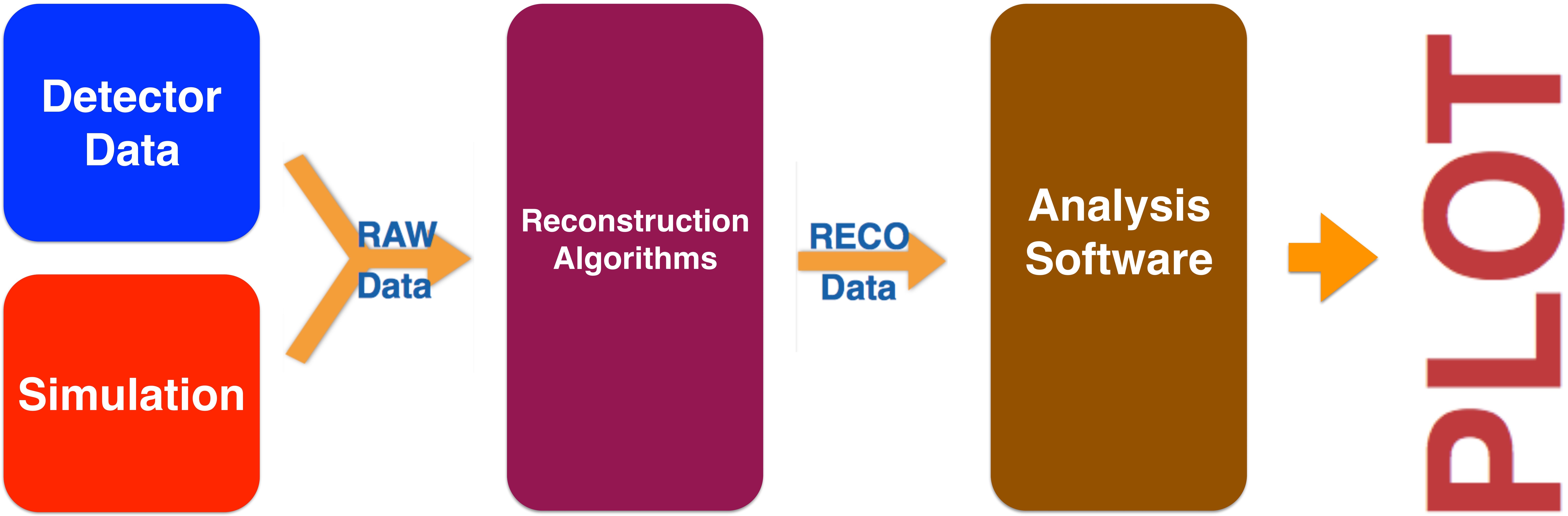
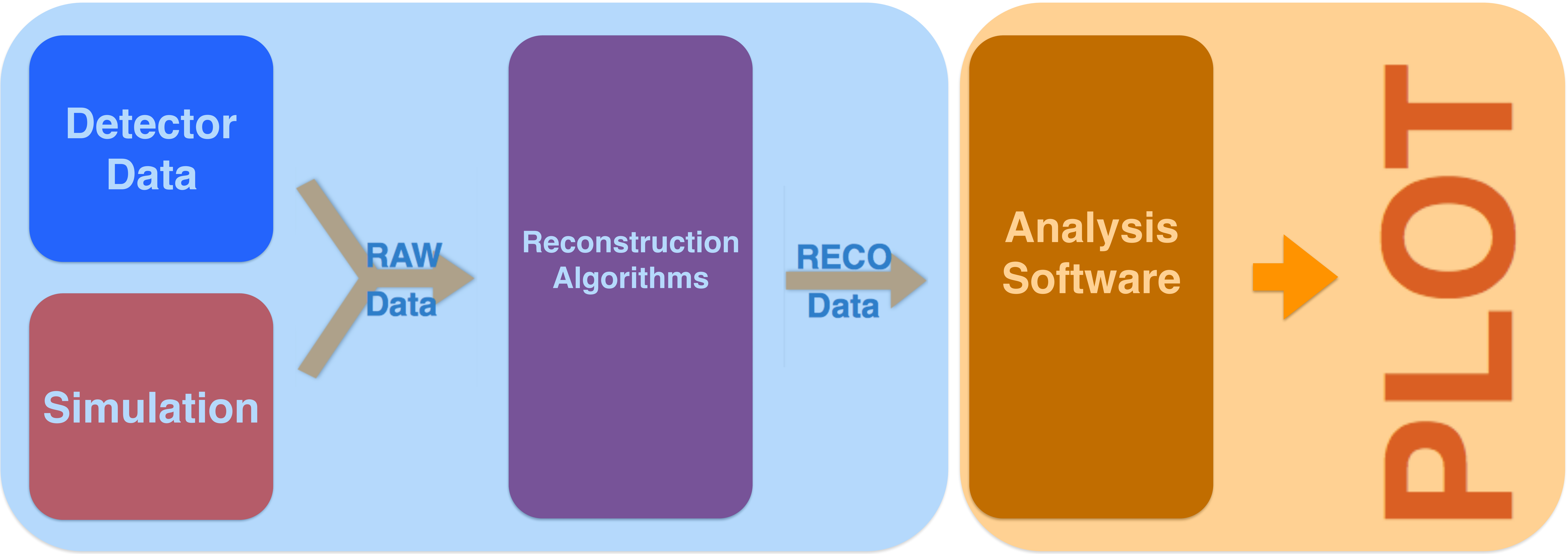
The Coffea Project: Introduction and Experience with Different Data Delivery Systems

The Coffea Development Team:

M. Cremonesi, L. Gray, A. Hall, I. Mandrichenko, N. Smith **[FNAL]**

J. Pivarski **[Princeton]**

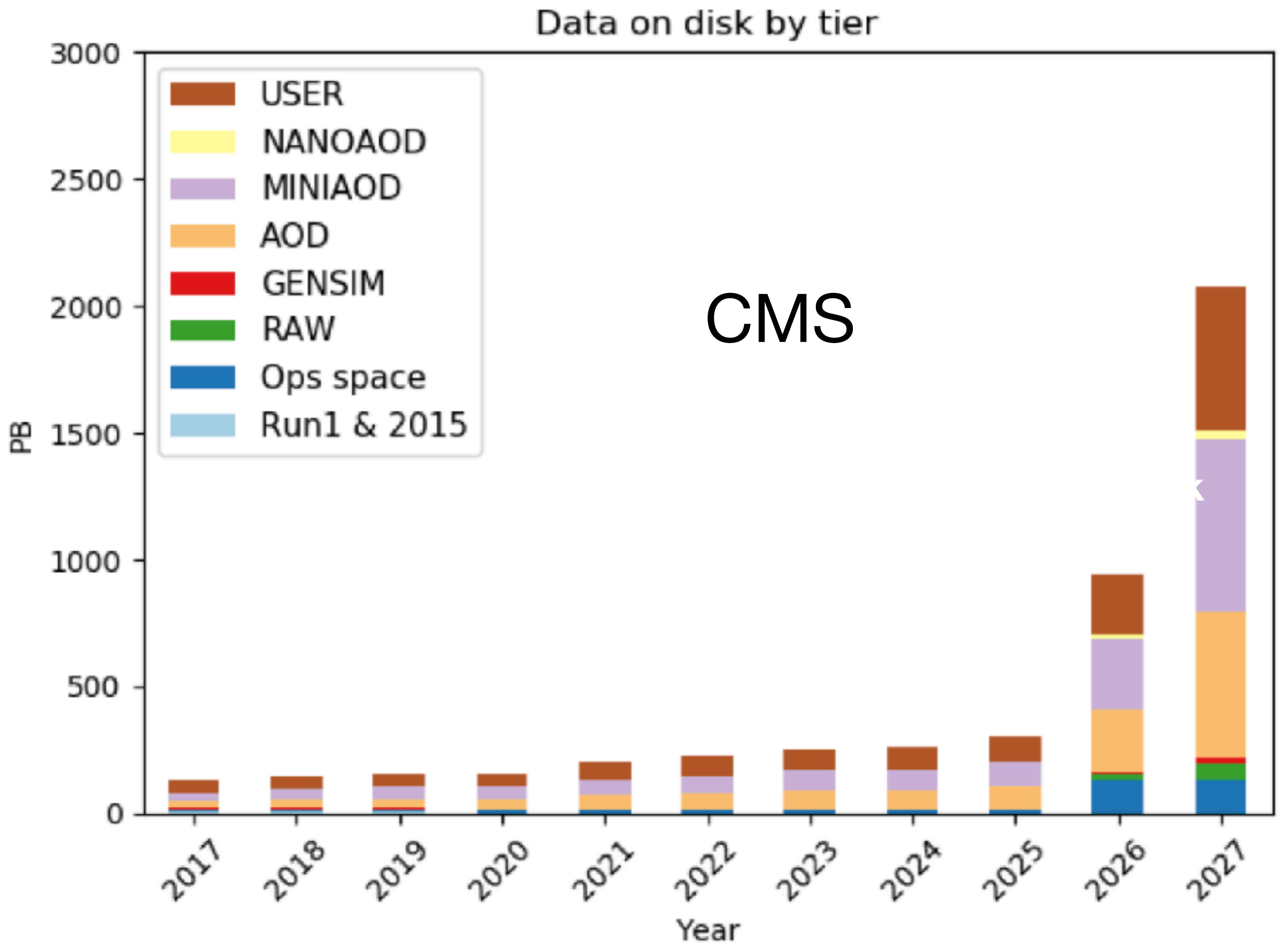

Detector Data
Simulation
RAW Data
Reconstruction Algorithms
RECO Data
Analysis Software
PLOT



Central **Chaotic**

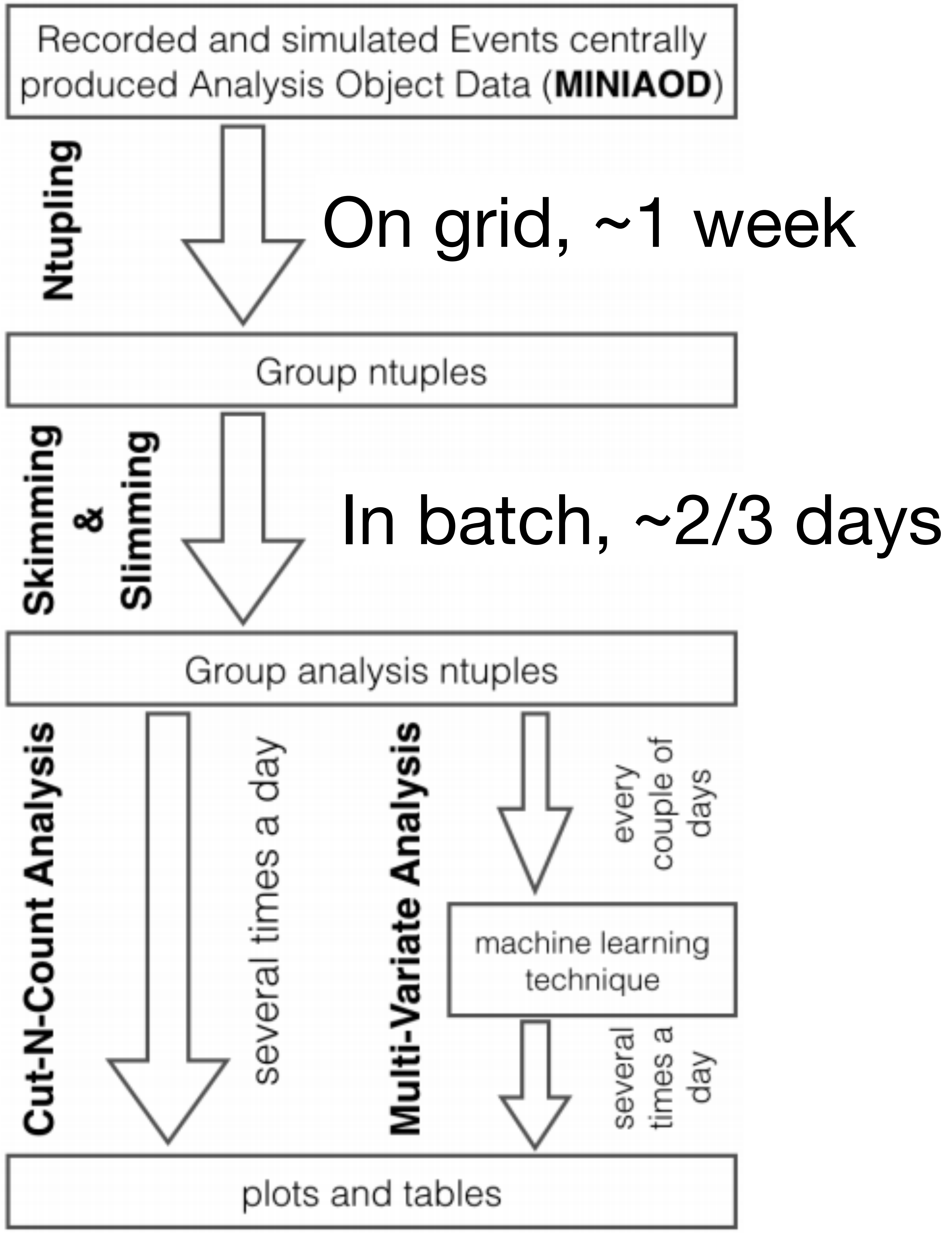
Data Volume

- Extract physics results will require to handle/analyze a lot more data
  - Must optimize further



Current CMS Analysis Workflow

- **MiniAOD**:
  - Centrally produced output of reconstruction software
  - O(100 TB), ROOT (C++) format, nested structure with branches
- **Ntupling**:
  - disk-to-disk copy, to modify the event content
  - duplicate immutable, add needed, and remove unused branches
- **Group ntuples**:
  - O(100 TB), ROOT (C++) format, nested structure with branches
  - dump of the content of the original files into a moderately specialized version
- **Skimming & Slimming**:
  - disk-to-disk copy, to limit latencies
  - dropping events/branches
- **Analysis ntuples:**
  - O(10 TB), ROOT (C++) format, flat



Current Limitations

The current file-based data representation and data management systems do not allow:

- To add/remove branches from the original data representation (*therefore we ntuple*)
- To extract branches efficiently from nested ROOT files (*therefore we skim&slim*)

Current Limitations

The current file-based data representation and data management systems do not allow:

- To add/remove branches from the original data representation (*therefore we ntuple*)
- To extract branches efficiently from nested ROOT files (*therefore we skim&slim*)

As a result we have:

- convoluted approach that **limits interactivity**
- Group/analysis-specific, often hardware-specific, **limits portability**
- unneeded duplication of immutable branches with **significant storage space**

Current Limitations

The current file-based data representation and data management systems do not allow:

- To add/remove branches from the original data representation (*therefore we ntuple*)
- To extract branches efficiently from nested ROOT files (*therefore we skim&slim*)

As a result we have:

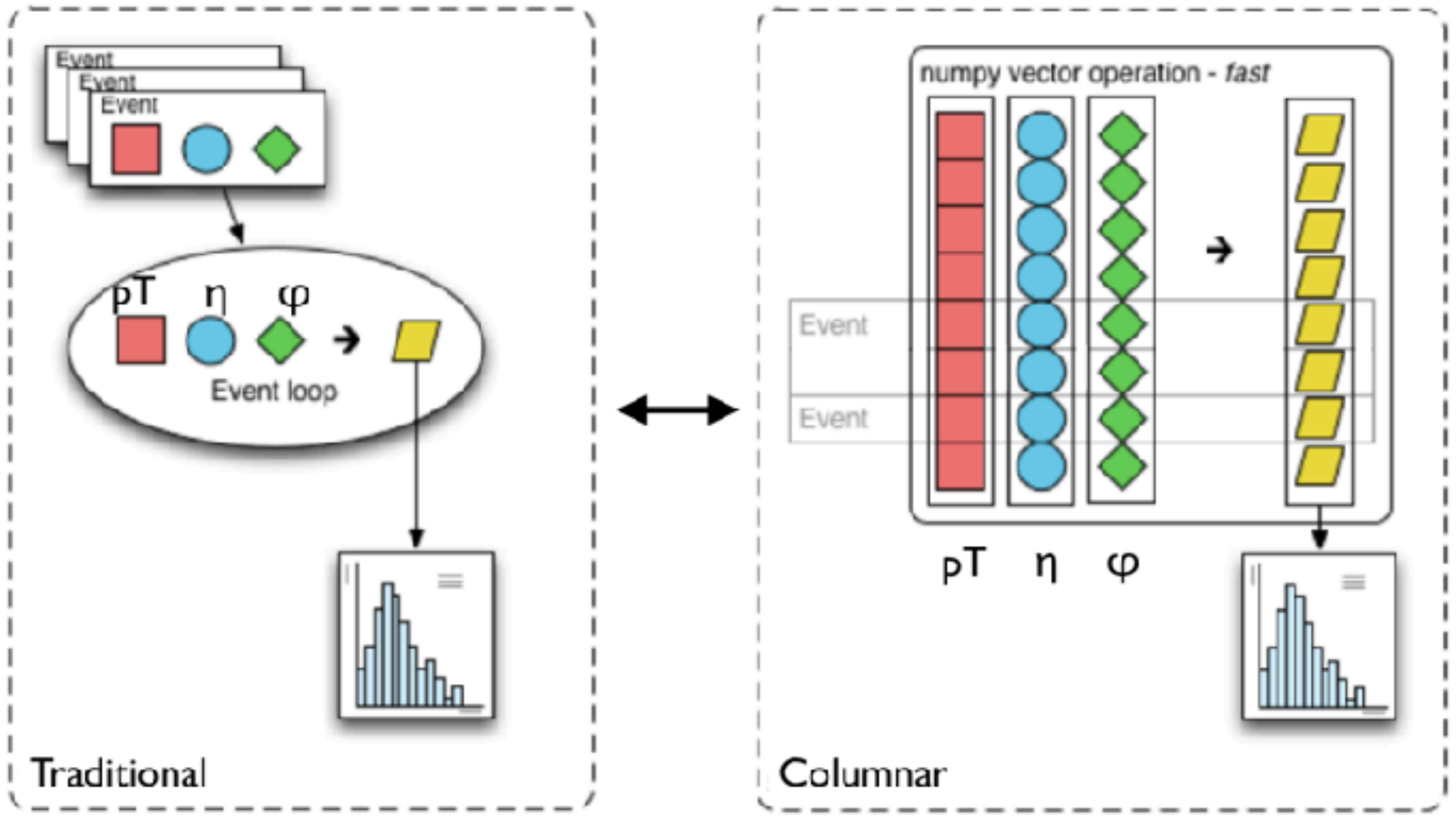
- convoluted approach that **limits interactivity**
- Group/analysis-specific, often hardware-specific, **limits portability**
- unneeded duplication of immutable branches with **significant storage space**

Additional limitations are introduced by manual bookkeeping:

- Tedious and time-consuming
- Results in an inefficient job splitting, with suboptimal parallelization

Solutions

- Develop a system that:
  - adopt a **columnar database** concept for input data representation
    - physics quantities are columns
    - Add/remove columns to modify the event content, **no ntupling**
    - Allow for "structural sharing" of immutable data, **no duplication**



Solutions

- Develop a system that:

  - adopt a **columnar database** concept for input data representation

    - physics quantities are columns

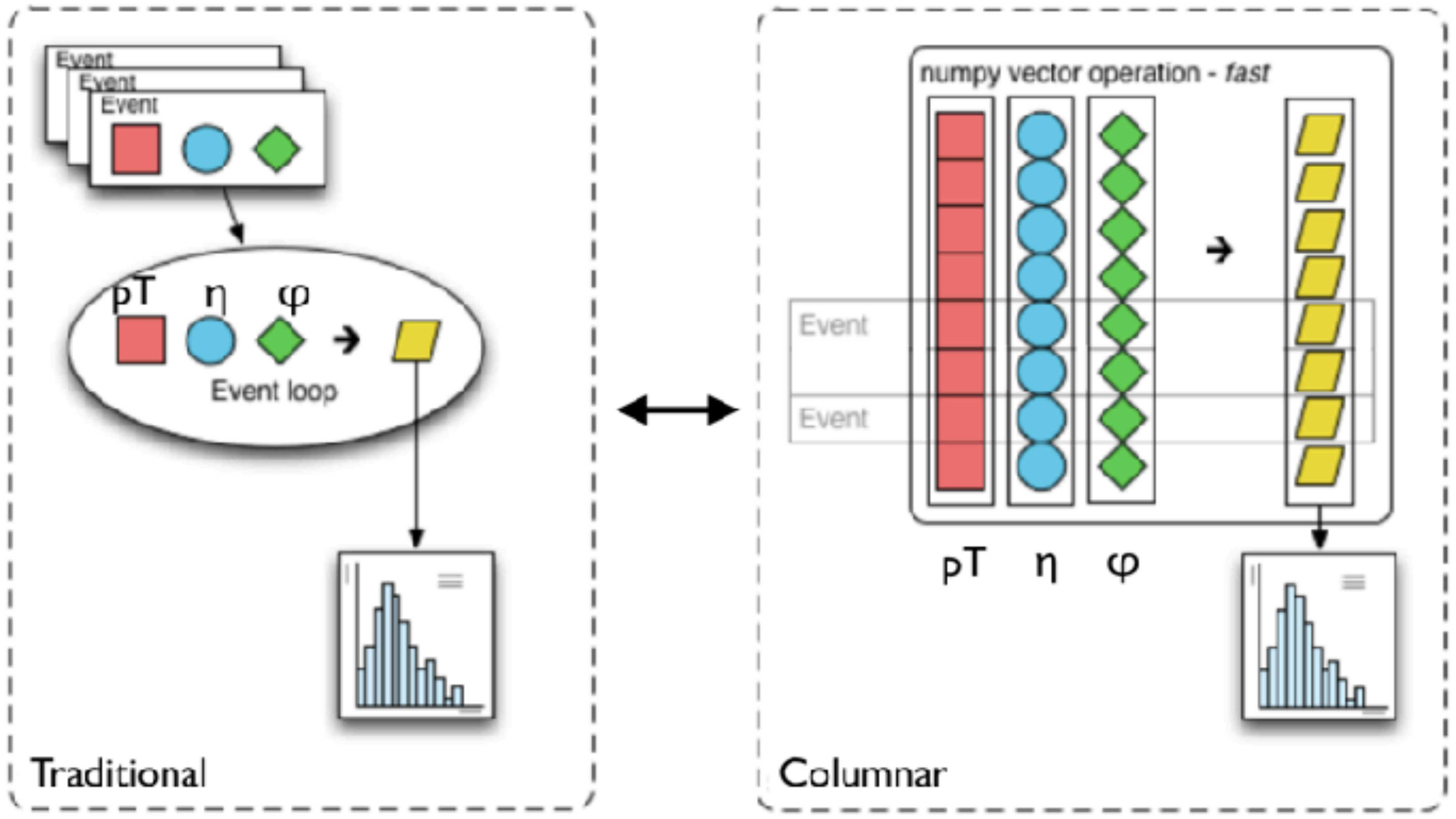
    - Add/remove columns to modify the event content, **no ntupling**

    - Allow for "structural sharing" of immutable data, **no duplication**

  - Does **caching** and **indexing** of the inputs to replace skimming/slimming

    - No stage-out of intermediate steps

  => Produce plots directly from the inputs, is fully portable and use-case independent



Solutions

- Develop a system that:

  - adopt a **columnar database** concept for input data representation

    - physics quantities are columns

    - Add/remove columns to modify the event content, **no ntupling**

    - Allow for "structural sharing" of immutable data, **no duplication**

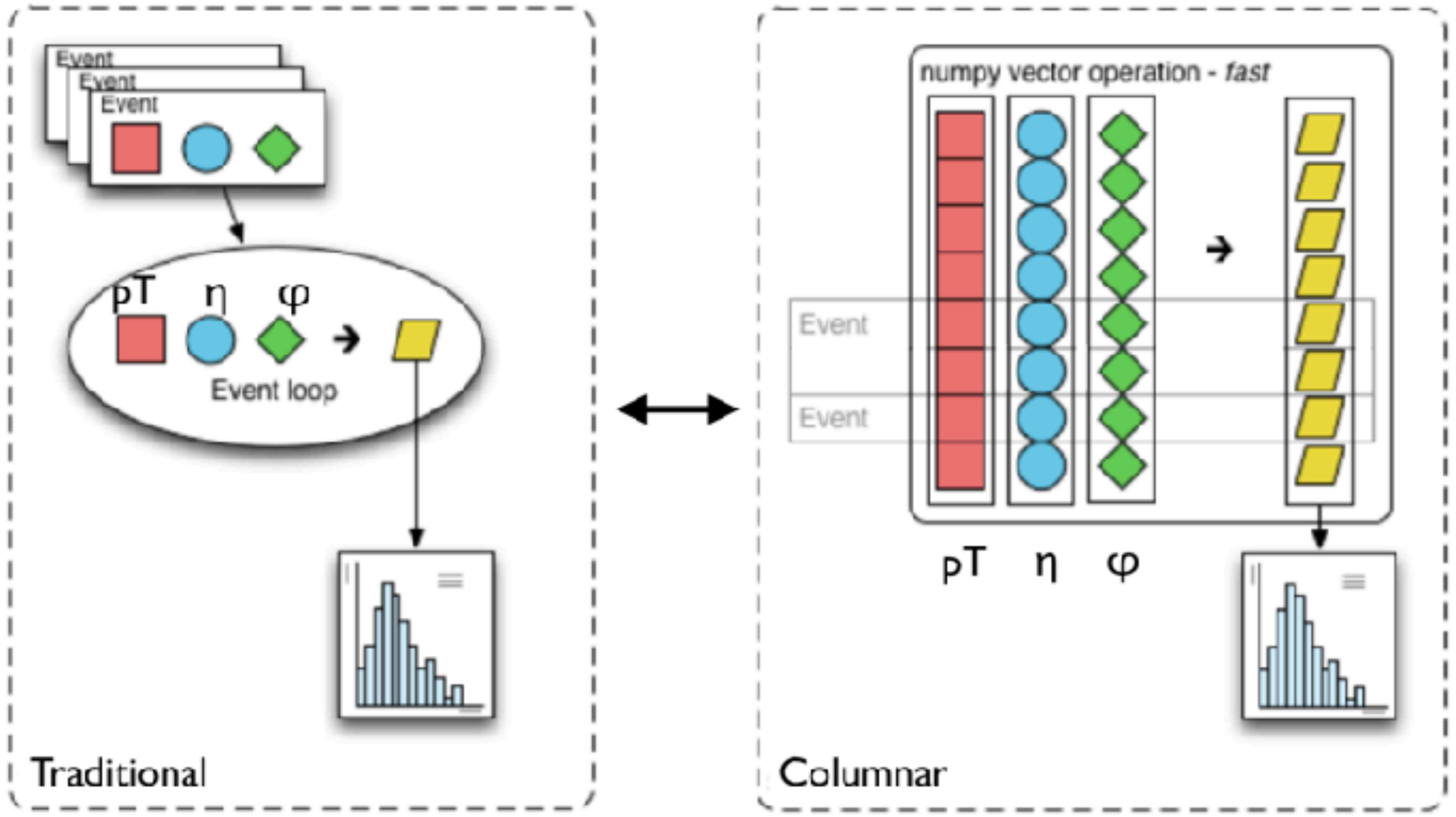
  - Does **caching** and **indexing** of the inputs to replace skimming/slimming

    - No stage-out of intermediate steps

  => Produce plots directly from the inputs, is fully portable and use-case independent

- Use **Apache-Spark/Striped** as general-purpose engines for large-scale data processing to deliver columns

  - Solves by construction the problem of the manual bookkeeping



Coffea

https://github.com/CoffeaTeam

Columnar **O**bject **F**ramework **F**or **E**fficient **A**nalysis

- **CoffeaHarvester:** delivers HEP data in columnar form
- **coffeabeans:** columnar datasets and metadata
- **CoffeaGrinder:** fast, understandable columnar analysis code
- **coffeapods:** histograms aggregated into plots or fitting ntuples
- **CoffeaMaker:** interface to CMS Combine fitting package

= in this talk

= next talk from Nick

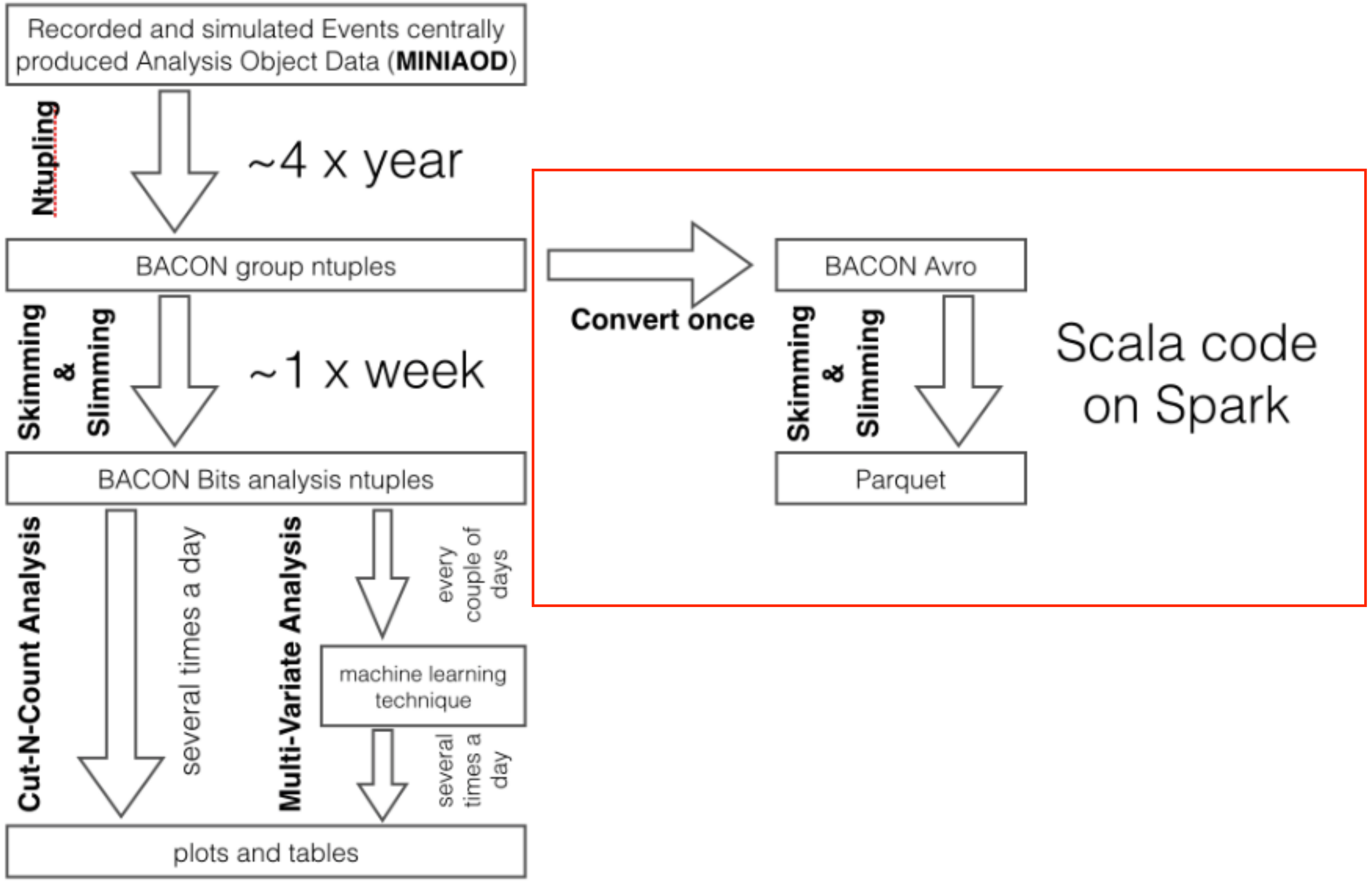
Spark experience: the CMS Big Data Project

- Group created end of 2015
  - tight collaboration with Diana-HEP at Princeton and CERN-IT
  - website: https://cms-big-data.github.io
- Partnerships with industry through CERN openlab:
  - Fermilab joined CERN openlab in 2017
  - **Intel** actively taking part in the project
    - CERN fellow sponsored by Intel

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

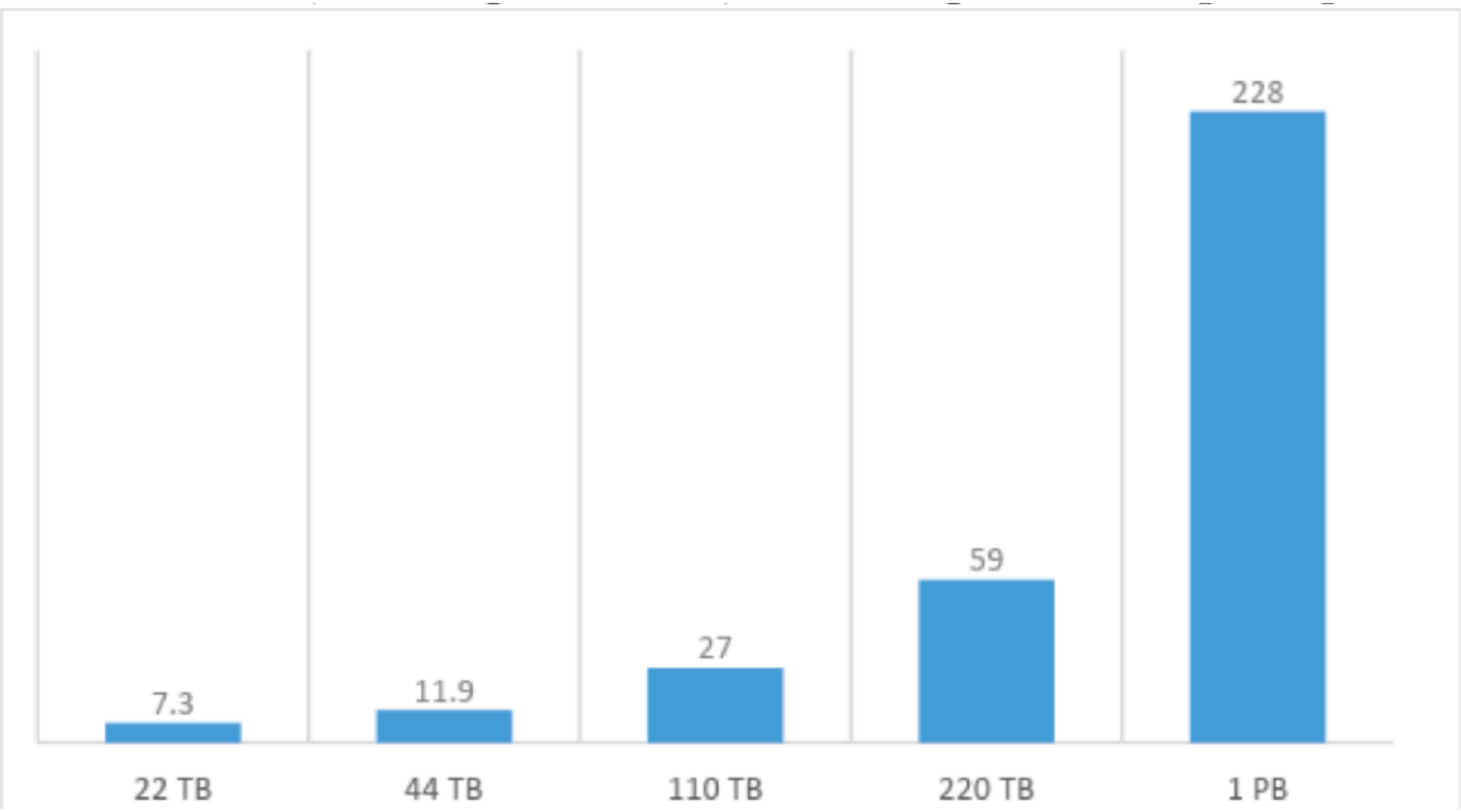
Fermilab

CERN Infrastructure

- Spark cluster:
  - **analytix @ CERN**: shared infrastructure with **~1300 cores, 7 TB RAM**
- Storage:
  - Remote EOS Public
- Simple physics analysis use case is applied to select events and reduce the datasets

Scalability Test

Increasing the input size while maintaining the same amount of resources

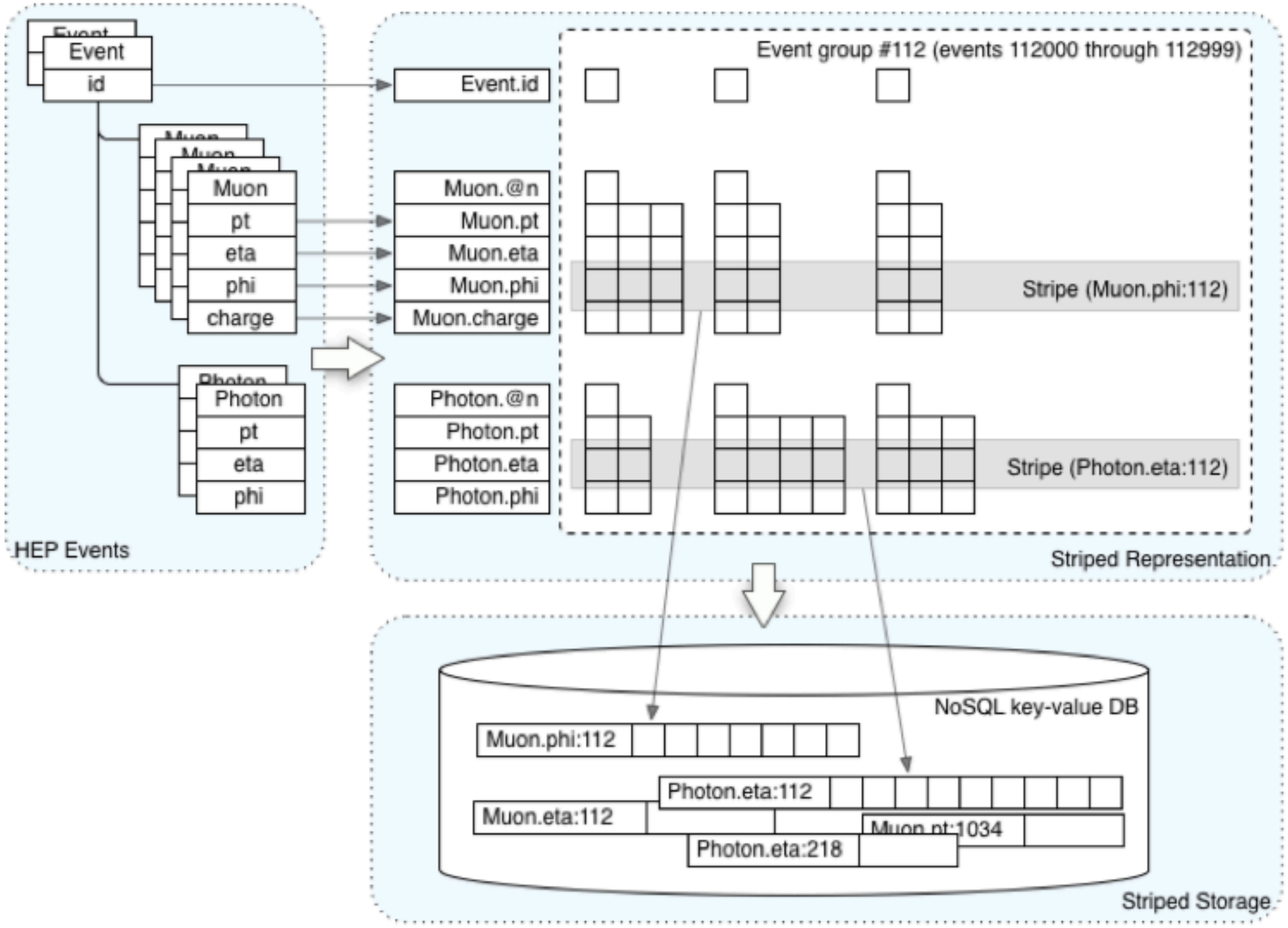


Input	EOS PUBLIC
22 TB	7.3 mins
44 TB	11.9 mins
110 TB	27 mins
220 TB	59 mins
1 PB	3.8 hours

Initial configuration: 804 logical cores, and 8 logical cores per Spark executor

The Striped Server

- HEP data rearranged into **stripes,** stored in a noSQL database, served to worker nodes
- Only the columns requested for the analysis are sent
- Input data are cached for quick re-use, and columns may be updated

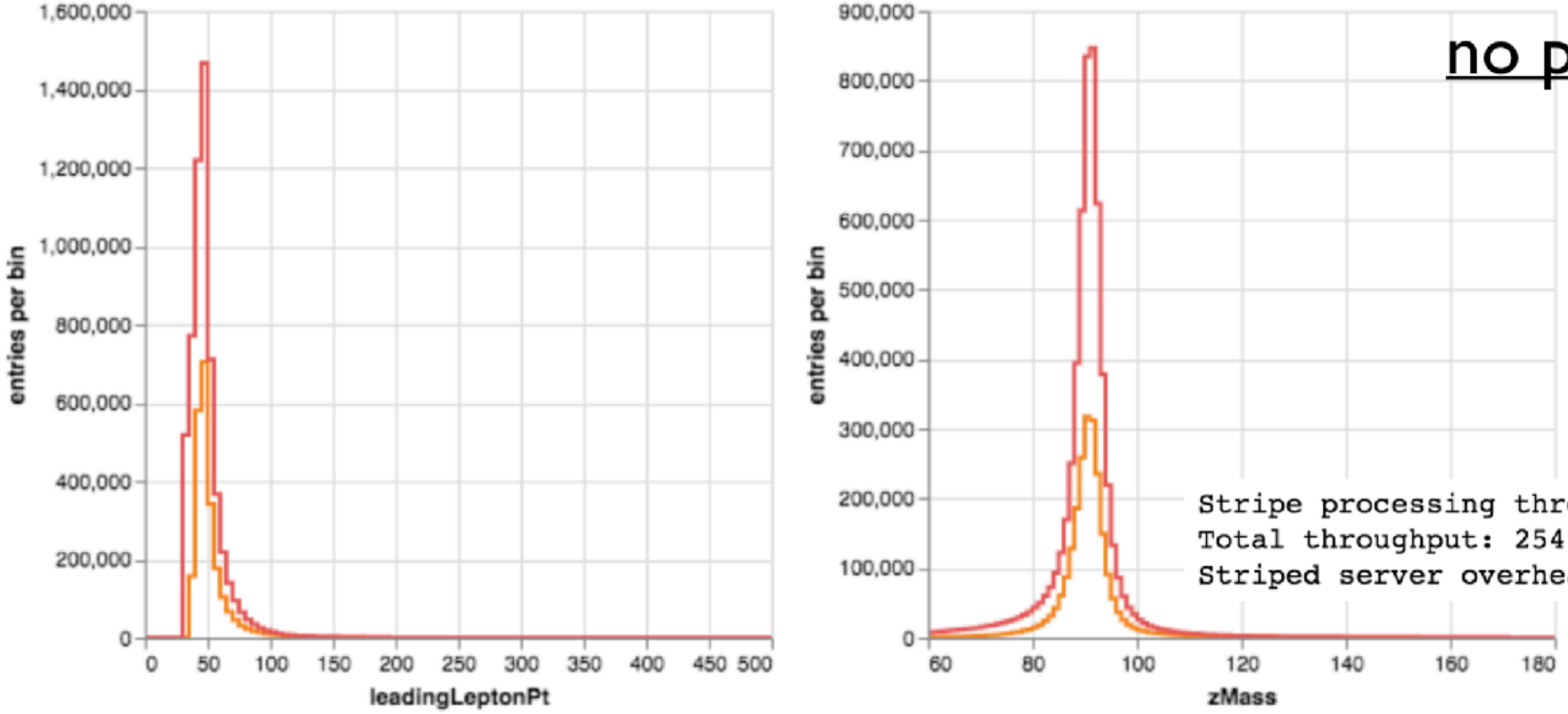


Fermilab

Striped Performances

```
Summer16.DYJetsToLL_M-50_TuneCUETP8M1_13TeV-madgraphMLM-pythia8        49.144 M events,   4.328 M
events/sec
Total events processed: 49144274 in 19.3 seconds -> 2.549672 million events/second
```

no preselection




```
Stripe processing throughput: 105576 evt/s
Total throughput: 2549672 evt/s
Striped server overhead: 86.6 %
```

(From CERN)

100 kHz per core, > 4 MHz total throughput with 280 workers from CERN

At FNAL, ~10 MHz with network overhead

Conclusions

- Spark
  - Scale as expected with the resources, proved the ability to reduce 1 PB of input in <4hrs
  - Slowness in talking to the JVM is the major bottleneck, there are solutions available
- Striped
  - 4 MHz with 280 cores from CERN, up to 10MHz from FNAL
  - Dataset uploading to the database requires significant time and manual bookeeping, a striped upload module for CMSSW can solve this

We will explore different data delivery systems to accomplish the “harvesting” step, working in tight connection with data engineers both from academia and industry

- Evaluating the possibility of a partnership with Databricks

Summary

- Developing a columnar analysis framework for HEP analysis
  - New analysis style (array programming, more in Nick's talk)
  - Fits nicely with big data processing engines
    - In this talk, Spark and Striped
- Currently exploring multiple solutions to find the optimal data delivery system for the Coffea framework

Backup

Analysis Use-case @ Vanderbilt/Padova

Analysis workflow:

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

Infrastructure:

- Padova
  - ~1000 cores with 5 TB of RAM
- Vanderbilt
  - 40 cores and 16 GB of RAM

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

Usability Test

- Make a first-year CS undergraduate student run the workflow
  - No knowledge of physics whatsoever, limited computing knowledge
  - Able to make the Vanderbilt workflow run in one day
- **Portability**
  - Run the Padova code at Vanderbilt
  - Major showstopper: environment setup

=> Solutions:

- shared library with site configuration towards **full generalization**
- packaging of the Hadoop-XRootD connector in order to make the tool more automatically deployable**, avoiding manual configuration**

Running Through VC3

- Virtual Clusters for Community Computation (VC3) is a service that:
  - shares custom software across multiple sites
  - creates a virtual cluster of the desired size
- Able to run Padova workflow on the UChicago Midwest Tier 2 cluster
  - 2 executors, 8 cores, 12GB RAM

Need to test with resources from other remote sites