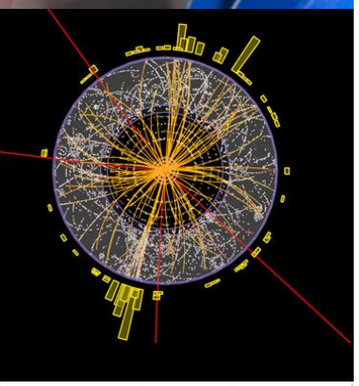
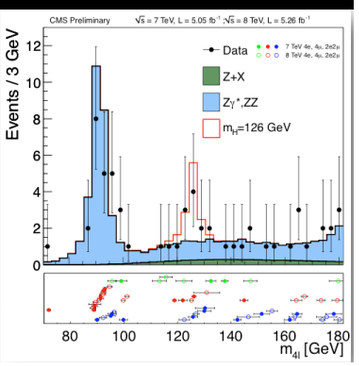
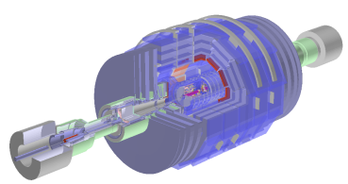


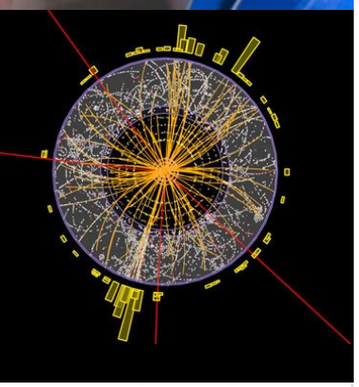


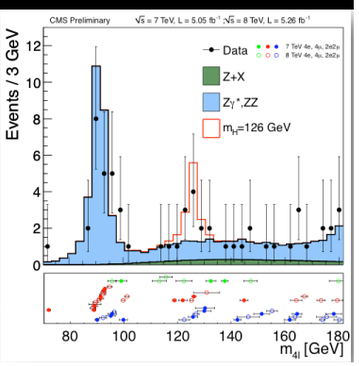



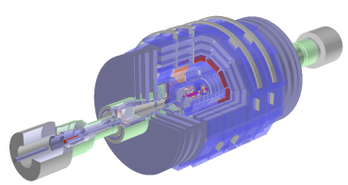
# ROOT I/O Workshop
# June 12th 2017

Philippe Canal
Fermilab
On behalf of ROOT Team.

# Contributors



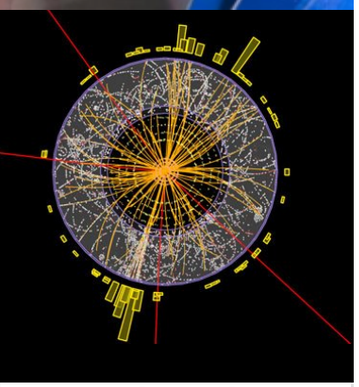


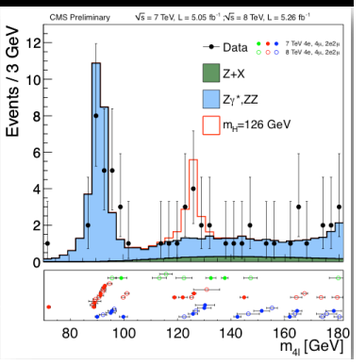



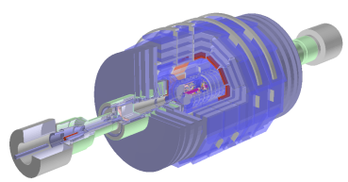
- Axel Naumann
- Bertrand Bellenot
- Brian Bockelman
- Danilo Piparo
- David Abdurachmanov
- Enric Tejedor
- Enrico Guiraud
- Luca Giommi
- Peter Van Gemmeren
- Philippe Canal
- Sergey Linev
- Xavier Valls
- Zhe Zhang

- Other languages:
  - Viktor Khristenko: JAVA
  - Sebastian Binet: GO

# Recent Additions

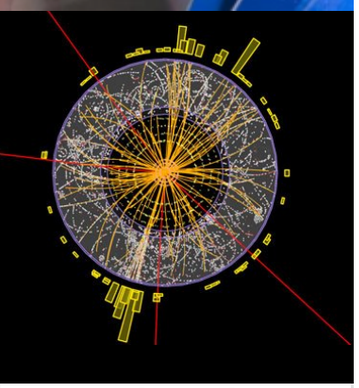


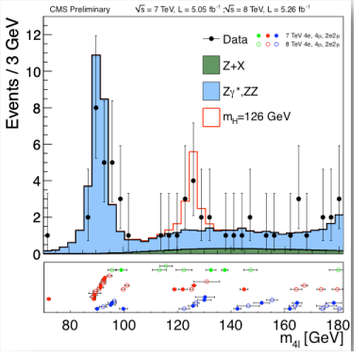


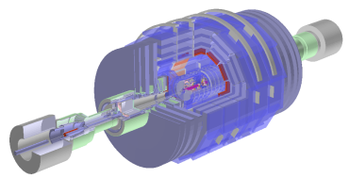



- Support I/O for
  - std::array
  - std::tuple
  - pointer to std::string
  - TMemFile larger than 2GB
  - LZ4 compression algorithm – Fast decompression.
- Significant enhancement of JSON read/write
  - Support for Collections
  - SaveAs(jsonfilename)
  - Started separating the text and binary StreamerInfoActions to improve output of the first and performance of the second (and first step towards coding the write-actions)
- Thread safety enhancements in Core, I/O, TTree

# Recent Additions

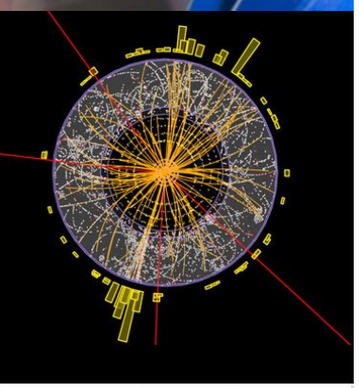


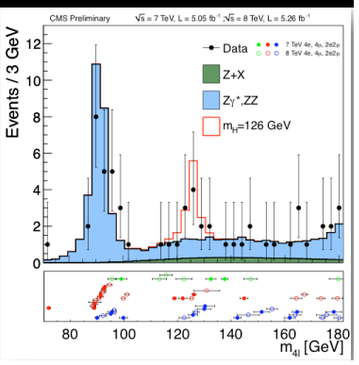


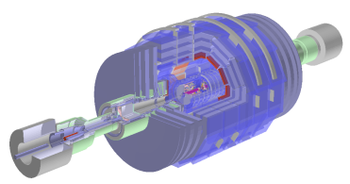



- TTree
  - Fast cloning now uses its own prefetch technique, i.e. fast cloning is now also fast over low latency links!
  - Command line histogramming via new command:
    - rootdrawtree (See also TSimpleAnalysis)
  - Implicitly parallel implementation of TTree::GetEntry
  - TTree::ReadStream (similar to TTree::ReadFile)
- ***TDataFrame***
- ***In-process parallel TTree merging***
- And "Elsewhere"
  - Resurrection of Java version of ROOT I/O
    - Of course completely independent from the one in JavaScript
  - GO implementation of ROOT I/O

# Continuing Progress on



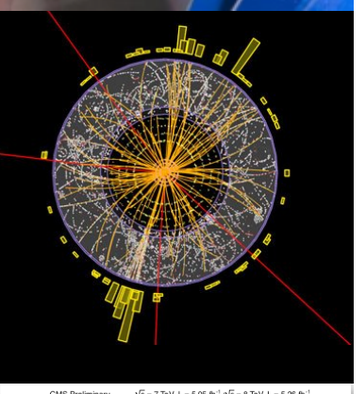


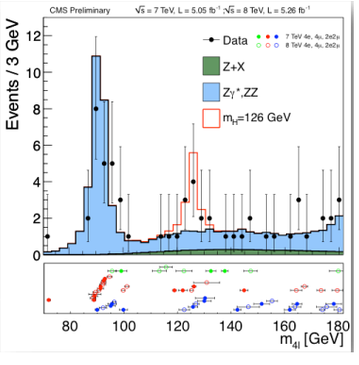



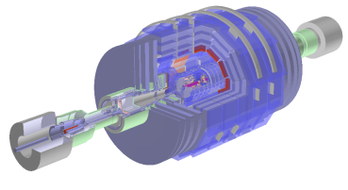
- ***Bulk I/O***
- ***TDataFrame***
- ***Compression***
  - LZ4
  - ***Hardware based***
  - ***In parallel***
- ***New TClonesArray (v7)***
- ***In Process Parallel Merge***

- ”Final” big thread-safety hurdle
  - ListOfCleanups (solution uses a read-write lock)
  - (Optional) Multi-access detector for Tcollection

# Progress on – Themes



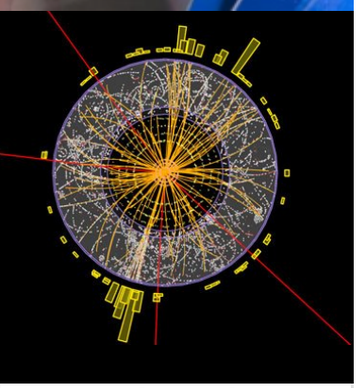


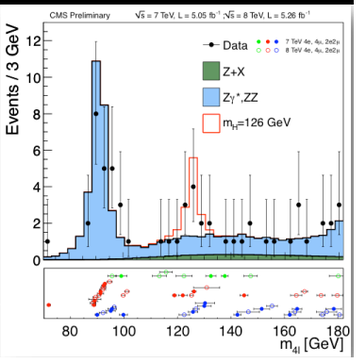



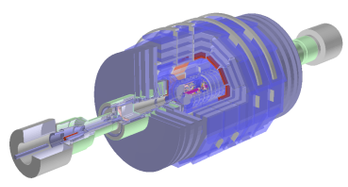
Two main themes:

- Performance
  - Both through optimization and new interfaces.
  - Multi-threading
- Usability
  - TDataFrame, etc.

# Interlude



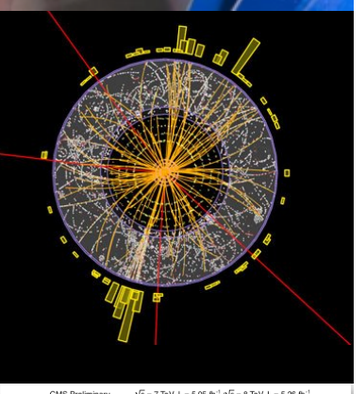


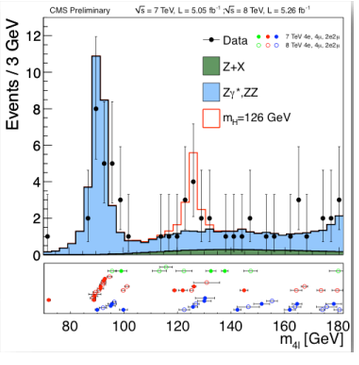



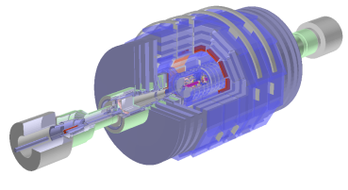
[Data Formats in HEP Analyses]
by Jakob Blomer

https://tinyurl.com/cern002

# Coming Soon

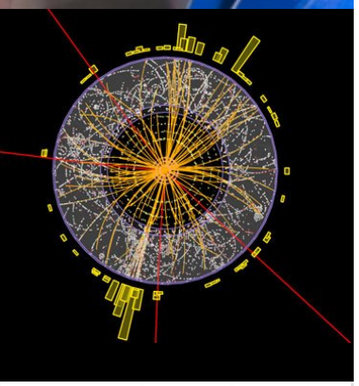


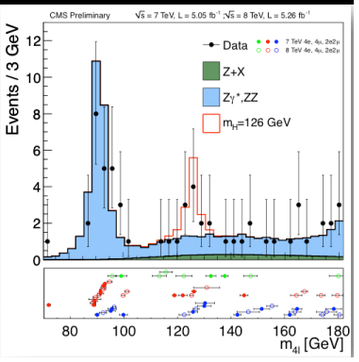


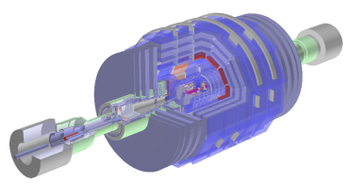



- LZ4 as the default compression algorithm
  - As it useful for the ‘quick’ analysis case, i.e. file is smallish and turn around matter most
  - 20% more disk space
  - **2 to 3** times **faster** decompression
- std::array in collections.
- std::shared_ptr
- ***I/O*** for ‘interpreted’ classes

# Coming soon

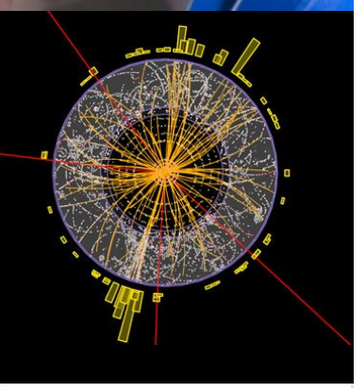


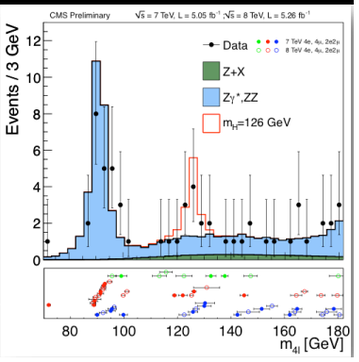


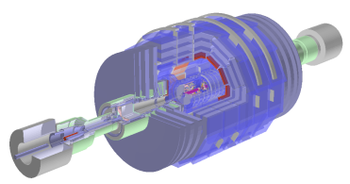



- ***OptimizeBasket***
  - There are a couple of new algorithm proposals
  - Need to be tested on wide range of cases
- ***TTreeCache:*** Allow alternative algorithm
- Read/WriteBuffer
  - 25% of the read code moved to optimized framework (function based) ; representing most of the use cases.
  - Write code still need to be similarly optimized

# Further Plans

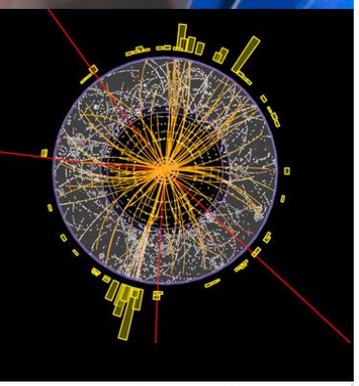


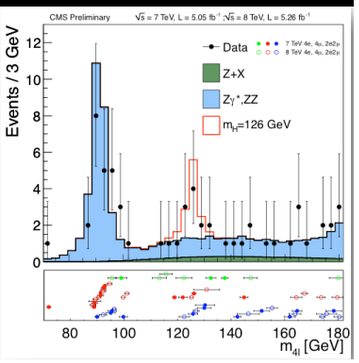


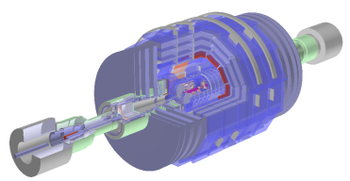



- ~~Write one direction files (for ***Hadoop*** )~~
- Enable Just-In-Time compilation of rules
- Extend automatic conversions
  - ***Derived**** <-> ***Base****
  - From object to pointer
  - From ROOT Collection to STL collection
- Design proper solution for experiment requiring sliding time frames instead of events

# Analysis Eco-System Workshop

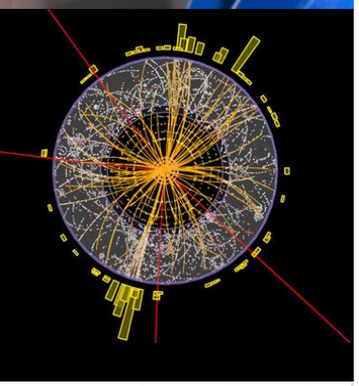


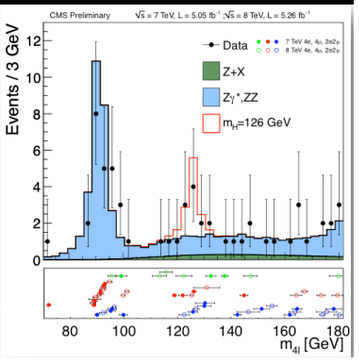


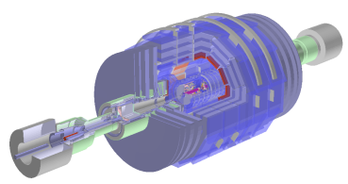



- **ROOT I/O will persist** as the key highly optimized data format and persistency technology for our I/O needs, but within an ecosystem in which others are important also. Accordingly, creating **data bridges** between formats, languages and tools is important for gaining full access to non-HEP tools.
- On fast storage, ROOT I/O performance is **dominated by deserialization**. Compression default may be reconsidered.
- Important for the end analysis is the I/O performance, **analyze where performance is lost** in current end analysis workflows.

# Consequence



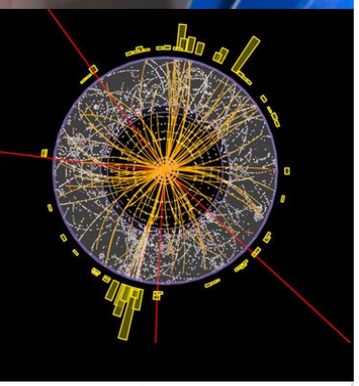


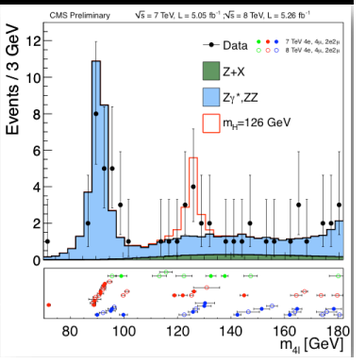



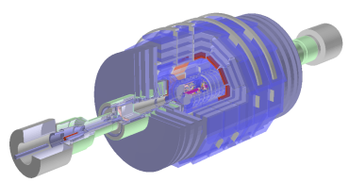
- Need to develop 'direct' access to TTree Data for NumPy arrays
  - Bulk I/O API likely a stepping stone

- Need to develop a HDF5 façade for ROOT Trees
  - Would allow use of HDF5 tools without conversion.

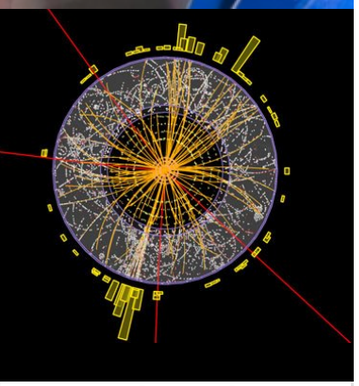
- Need to survey HDF5 converter landscape
  - Would help in making a recommendation if any.

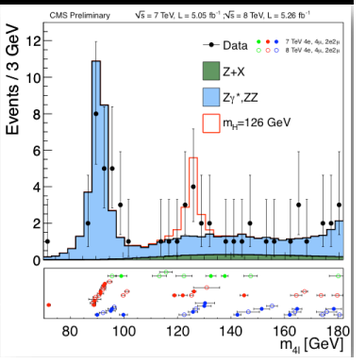


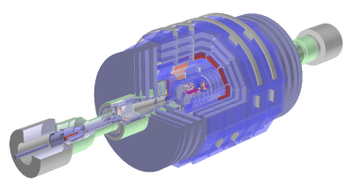


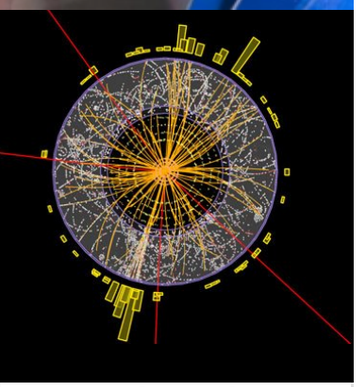
# Backup Slides

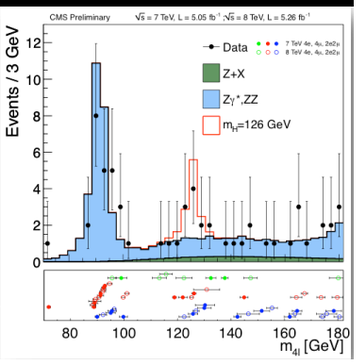


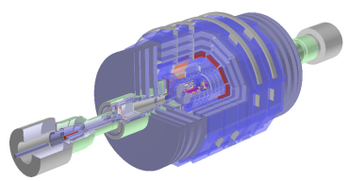


- Zero copy I/O
  - Essentially for significant performance boost
  - Prototype of TBuffer using little endian
  - First step in many
    - Extend to file. Handle/detect when I/O actions can be merged (deal with alignment etc.). Implement I/O action for the writing side.
- Compress each entry individually to improve random access
- Thread safe segfault handler





- ***TTree*** Draw/Scan
  - Leverage cling
- ***TTree***
  - Interface simplification
    - Make ***SetAddress*** and ***SetBranchAddress*** ‘smarter’
  - Optimizations
- In ***TTree***
  - Eg. ***TTree::Draw*** execute formula on more than one element at a time
  - New interface allowing retrieval of multiple entries at once

# Here comes cling

- ***Cling*** introduces binary compatible Just In Time compilation of script and code snippets.

- Will allow:
  - ***I/O*** for 'interpreted' classes
  - Runtime generation of ***CollectionProxy***
  - Run-time compilation of ***I/O*** Customization rules
    - including those carried in ***ROOT*** file.
  - Faster, smarter ***TTreeFormula***
  - Potential performance enhancement of ***I/O***
    - Optimize hotspot by generating/compiling new code on demand
  - Interface simplification thanks to full ***C++*** support