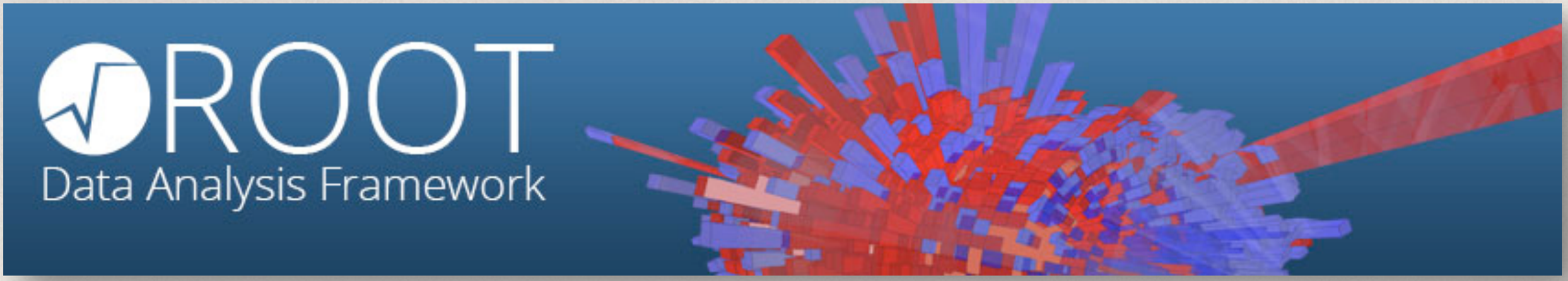


Data Processing in HEP: ROOT

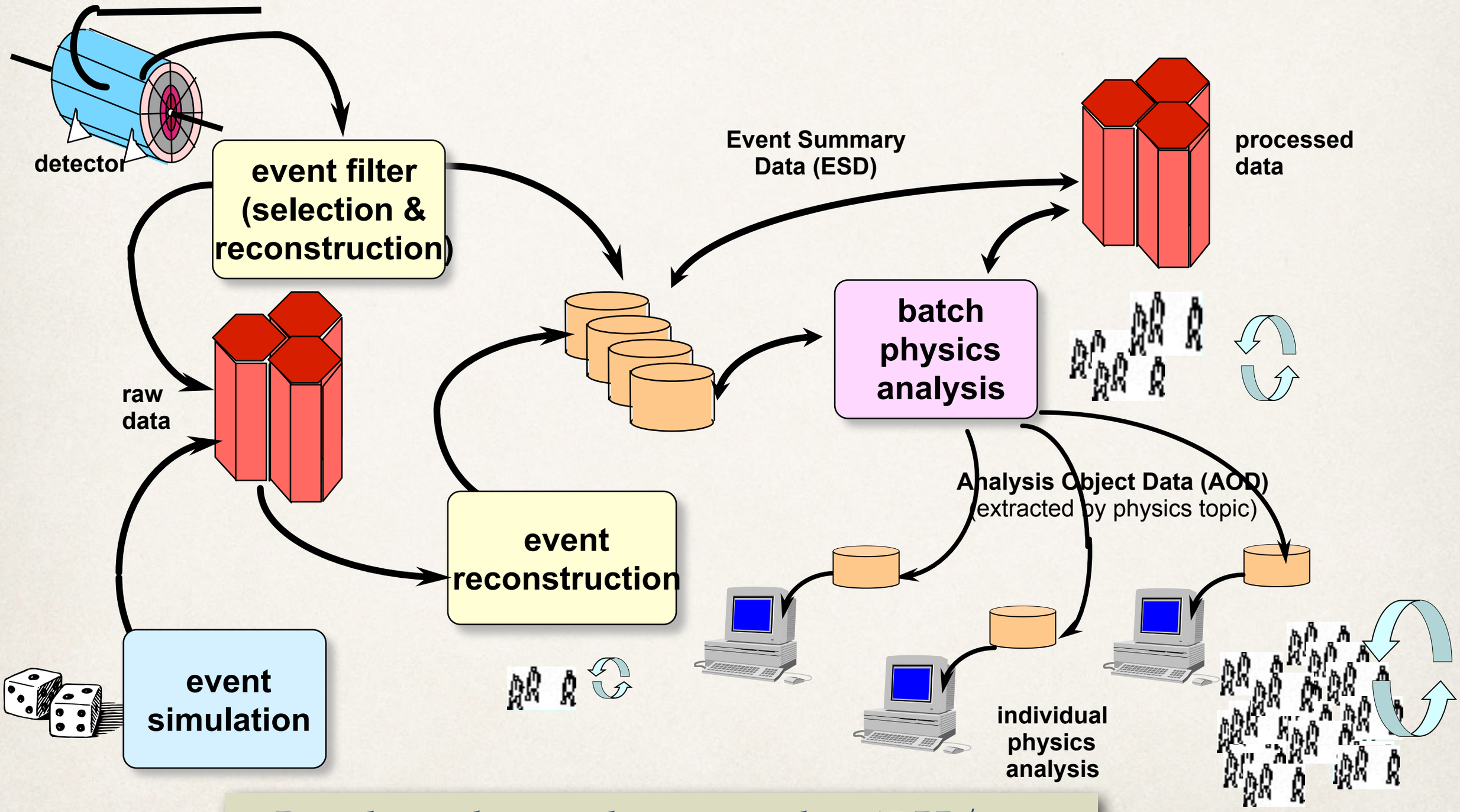
Future Trends in Nuclear Physics Computing, March 16-18 2016, Jefferson Lab

Pere Mato/CERN

Outline

- HEP Data Processing Software and Computing
  - LHC Computing Grid, LHC software stack
- ROOT: a modular scientific software framework
  - Main functionalities
  - What's new in ROOT 6
  - Development beyond ROOT 6
- Collaborations and the HEP Software Foundation
- Conclusions

Offline Data Processing

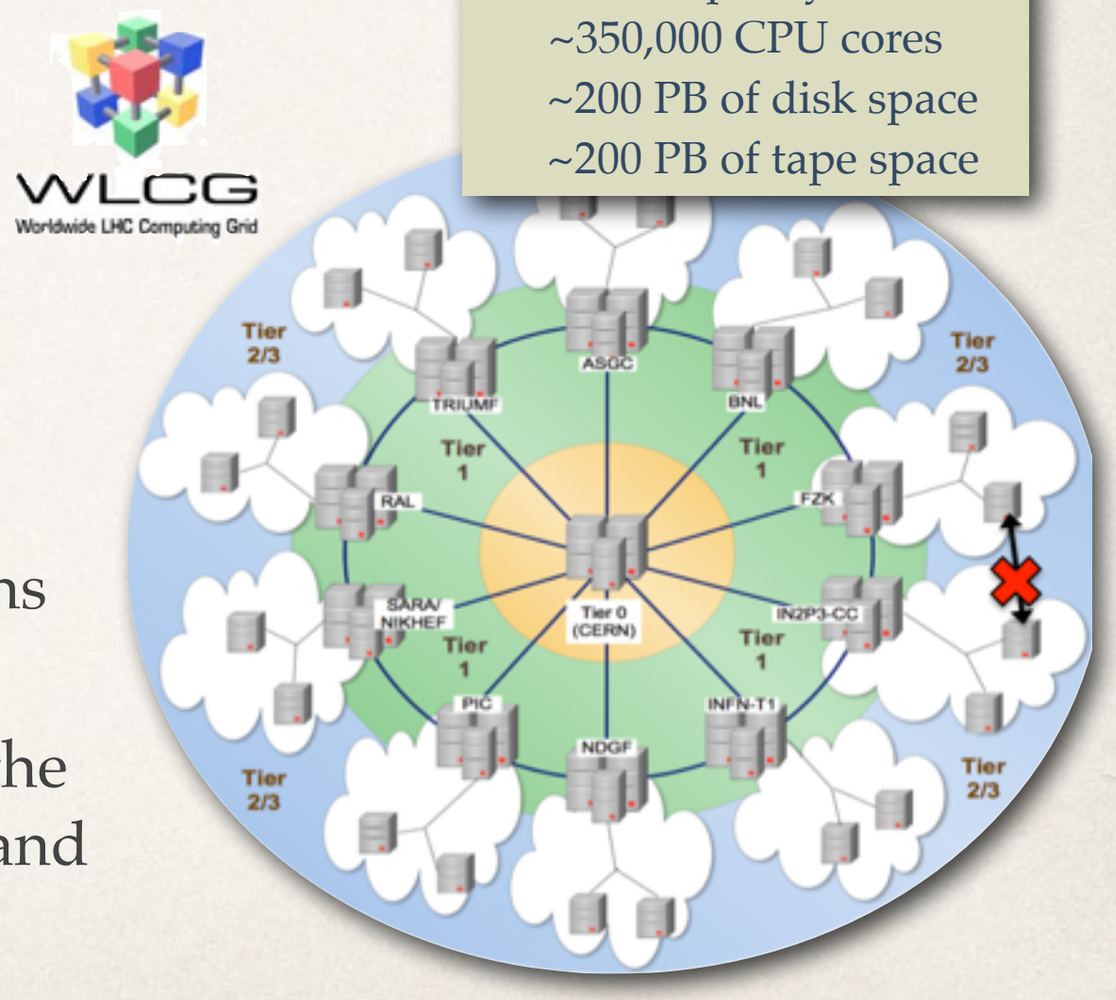


Raw data to be stored permanently: >15 PB/year

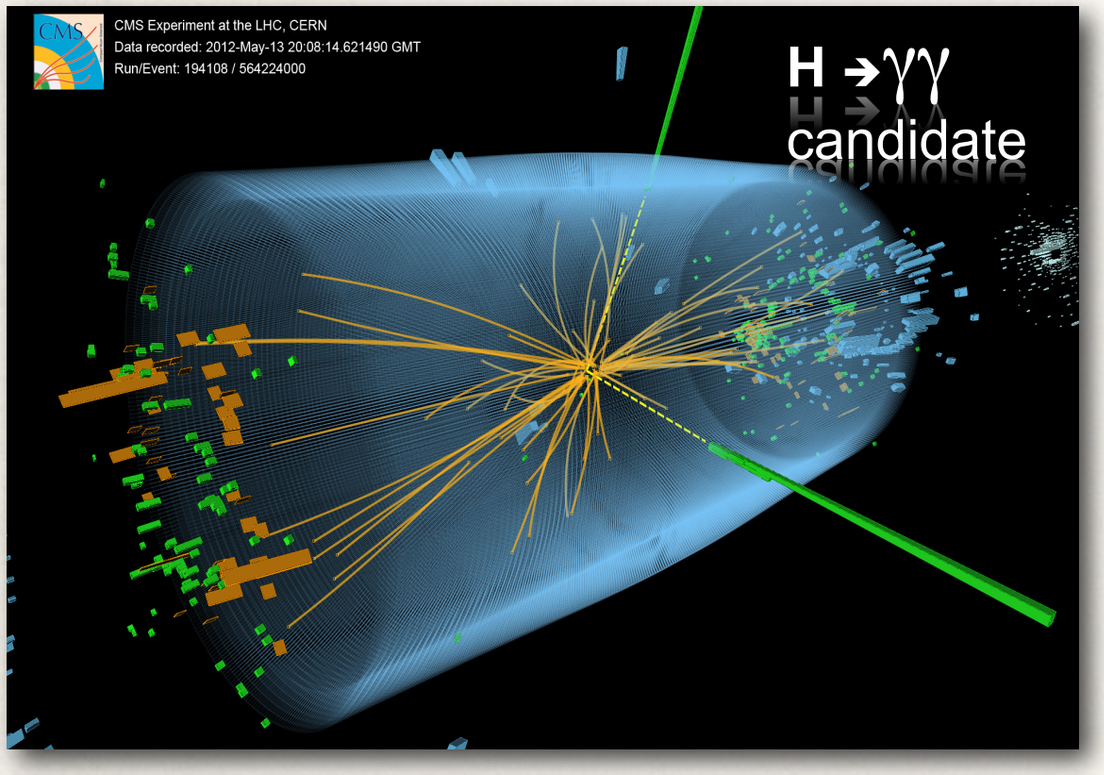
Big Data requires Big Computing

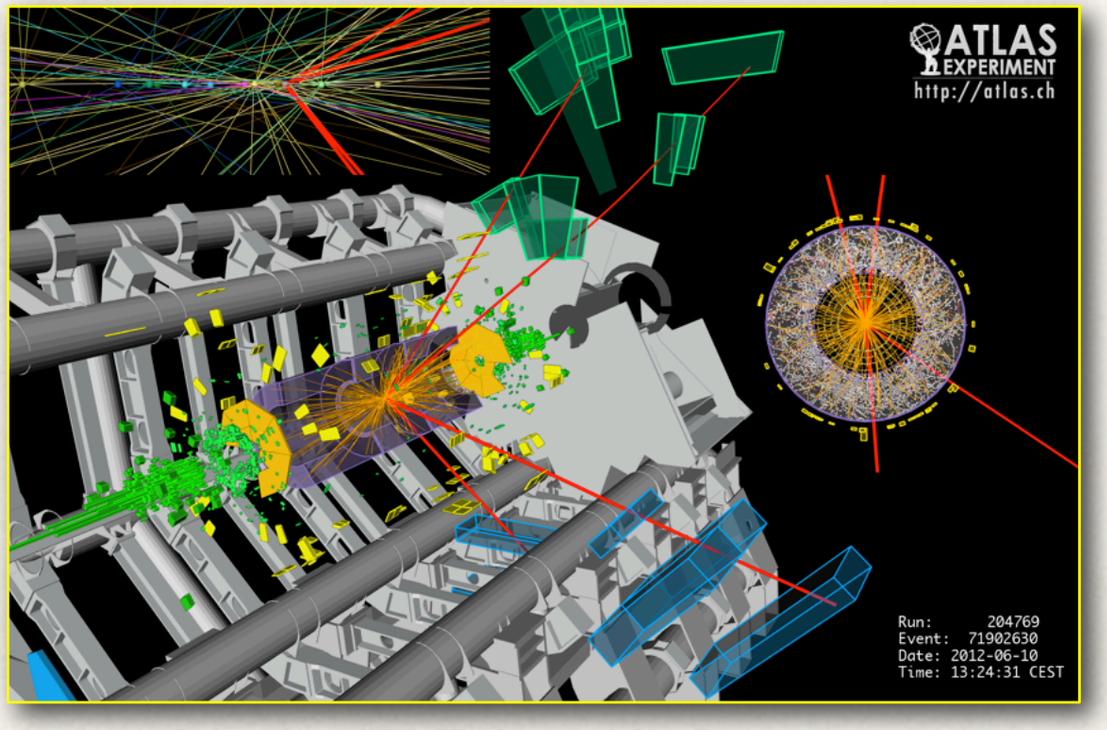
- The LHC experiments rely on distributed computing resources:
  - WLCG - a global solution, based on the Grid technologies/middleware.
    - distributing the data for processing, user access, local analysis facilities etc.
    - at time of inception envisaged as the seed for global adoption of the technologies
- Tiered structure
  - Tier-0 at CERN: the central facility for data processing and archival
  - 11 Tier-1s: big computing centers with high quality of service used for most complex/intensive processing operations and archival
  - ~140 Tier-2s: computing centers across the world used primarily for data analysis and simulation.

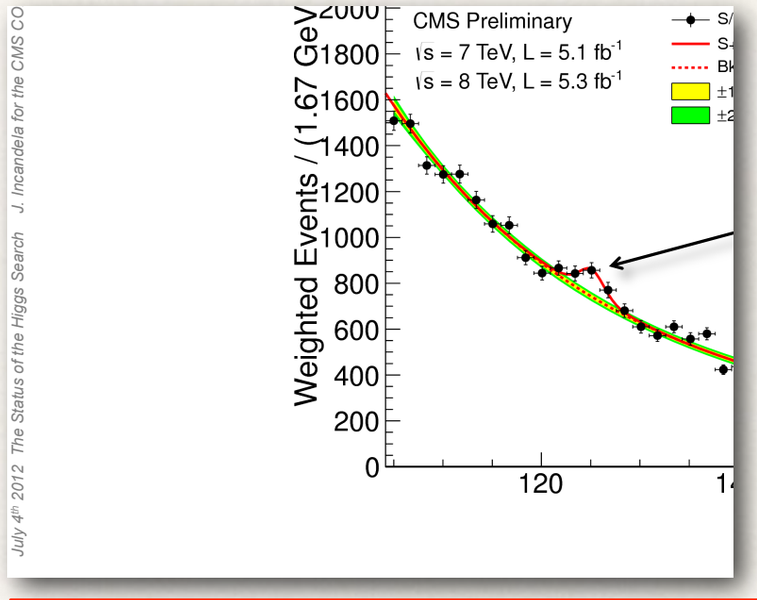
Capacity:
~350,000 CPU cores
~200 PB of disk space
~200 PB of tape space



A Success Story!







Higgs boson-like particle discovery claimed at LHC

COMMENTS (1665)

By Paul Rincon
Science editor, BBC News website, Geneva

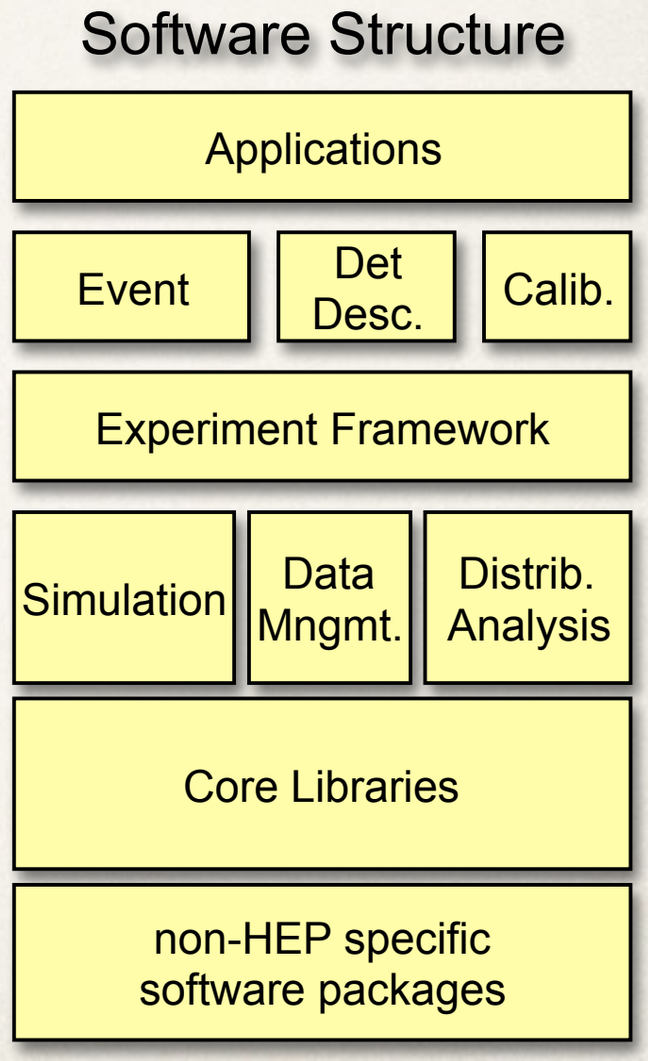
The moment when Cern director Rolf Heuer confirmed the Higgs results

Cern scientists reporting from the Large Hadron Collider (LHC) have claimed the discovery of a new particle consistent with the Higgs boson.

Relat

The LHC Software Stack

- Each experiment maintains its own application software
  - A total of ~50M lines of code; mainly written in C++
  - Few hundred part-time developers; mostly PhD students
  - More a bazaar than a cathedral
- Common packages and infrastructure:
  - Fundamental common software products and libraries
    - data storage, histograms, simulation toolkits etc.
  - Consistent stack of external software packages
    - GSL, Minuit, Qt, python, … (~100 in total)
  - Development infrastructure
    - procedures and tools



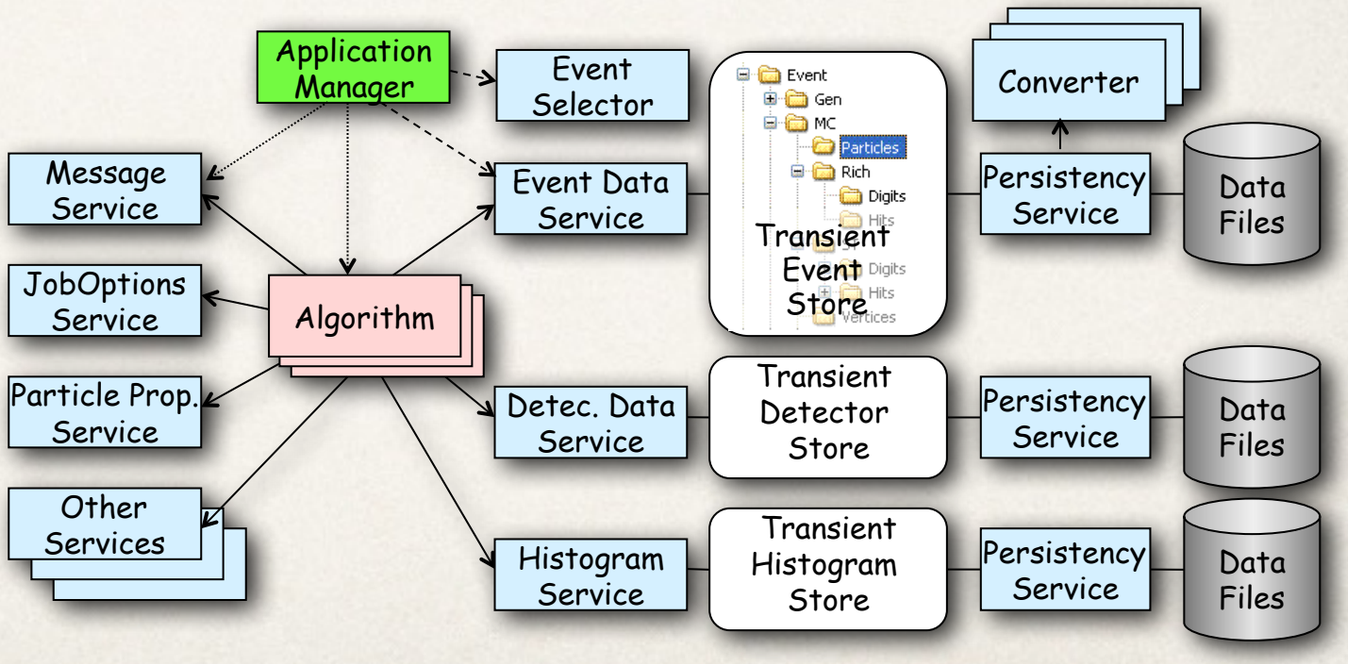
Software Components

- Foundation Libraries
  - Basic types
  - Utility libraries
  - System isolation libraries
- Mathematical Libraries
  - Special functions
  - Minimization, Random Numbers
- Data Organization
  - Event Data
  - Event Metadata (Event collections)
  - Detector Conditions Data
- Data Management Tools
  - Object Persistency
  - Data Distribution and Replication
- Simulation Toolkits
  - Event generators
  - Detector simulation
- Statistical Analysis Tools
- Histograms, N-tuples
  - Fitting
  - Interactivity and User Interfaces
- GUI, Scripting
  - Interactive analysis
  - Data Visualization and Graphics
- Event and Geometry displays
- Distributed Applications
- Parallel processing
- Grid computing

HEP Software Frameworks

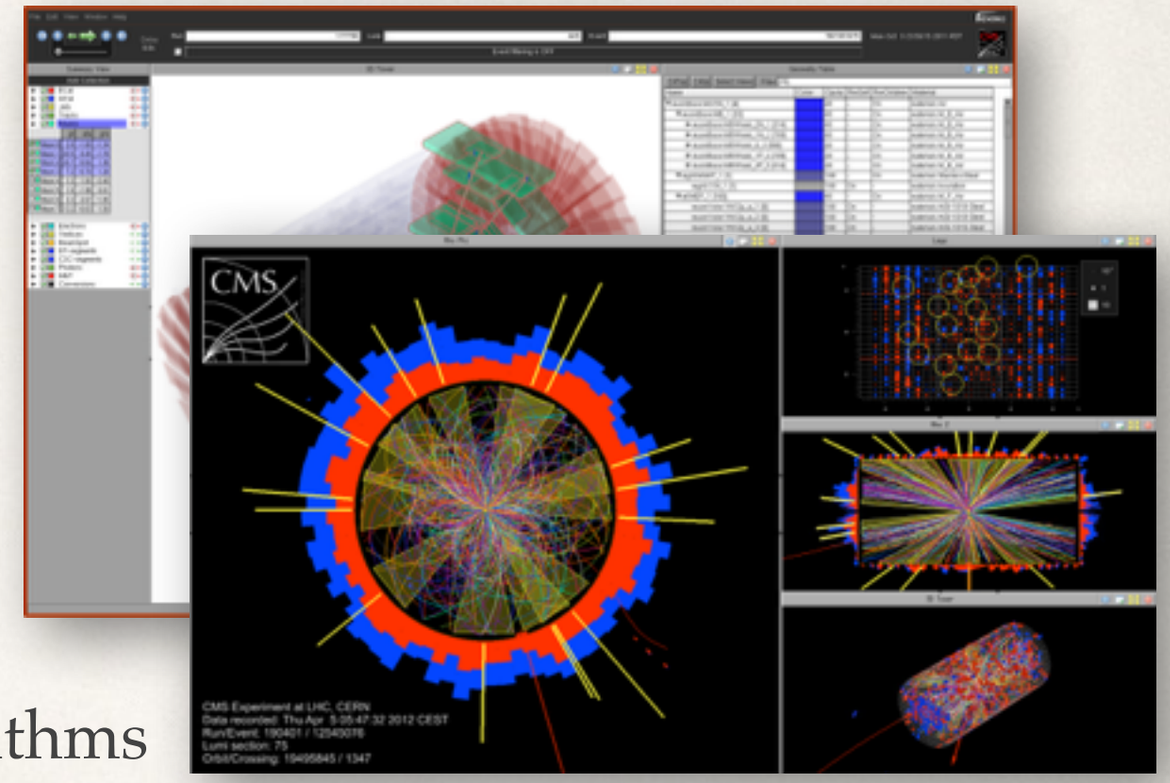
- HEP Experiments develop Software Frameworks
  - General Architecture of the Event processing applications
  - To achieve coherency and to facilitate software re-use
  - Hide technical details to the end-user Physicists (providers of the *Algorithms*)
- Applications are developed by customizing the Framework
  - By composition of elemental *Algorithms* to form complete applications
  - Using third-party components wherever possible and configuring them

- Example the Gaudi Framework used by ATLAS and LHCb among others



ROOT

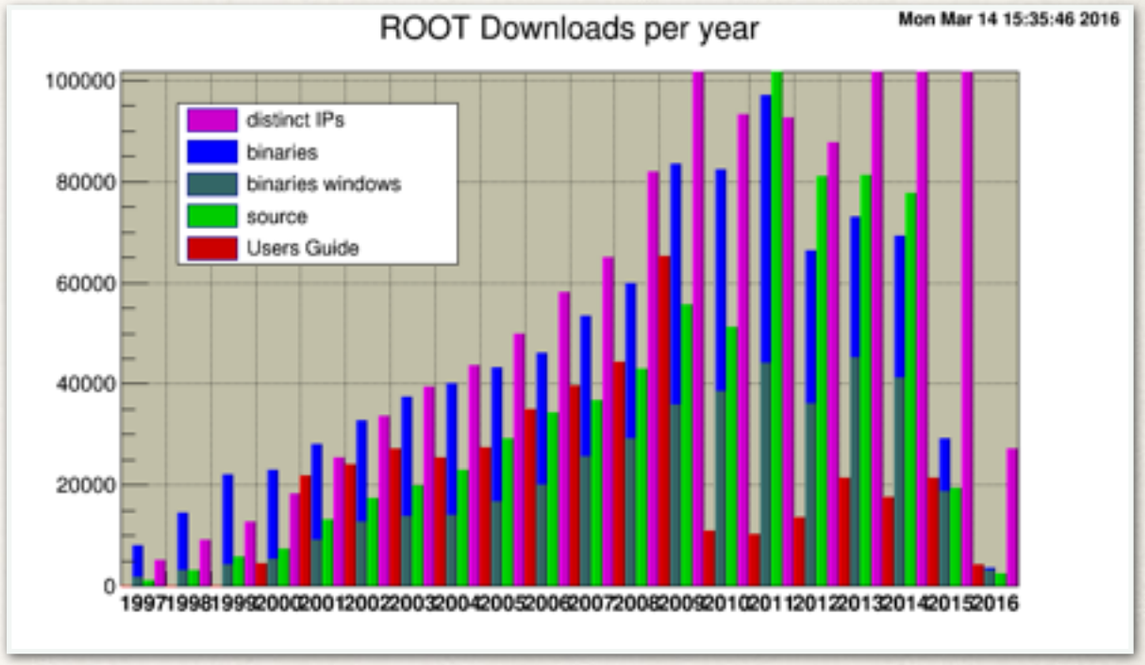
- "At the root of the experiments", project started in 1995
- Open Source project (LGPL3)
  - mainly written in C++; 4 MLOC
- ROOT provides (amongst other things):
  - C++ interpreter, Python bindings
  - Efficient data storage mechanism
  - Advanced statistical analysis algorithms
    - histogramming, fitting, minimization, statistical methods …
  - Multivariate analysis, machine learning methods
  - Scientific visualization: 2D/3D graphics, PDF, Latex
  - Geometrical modeler
  - PROOF parallel query engine



http://root.cern.ch

ROOT in Numbers

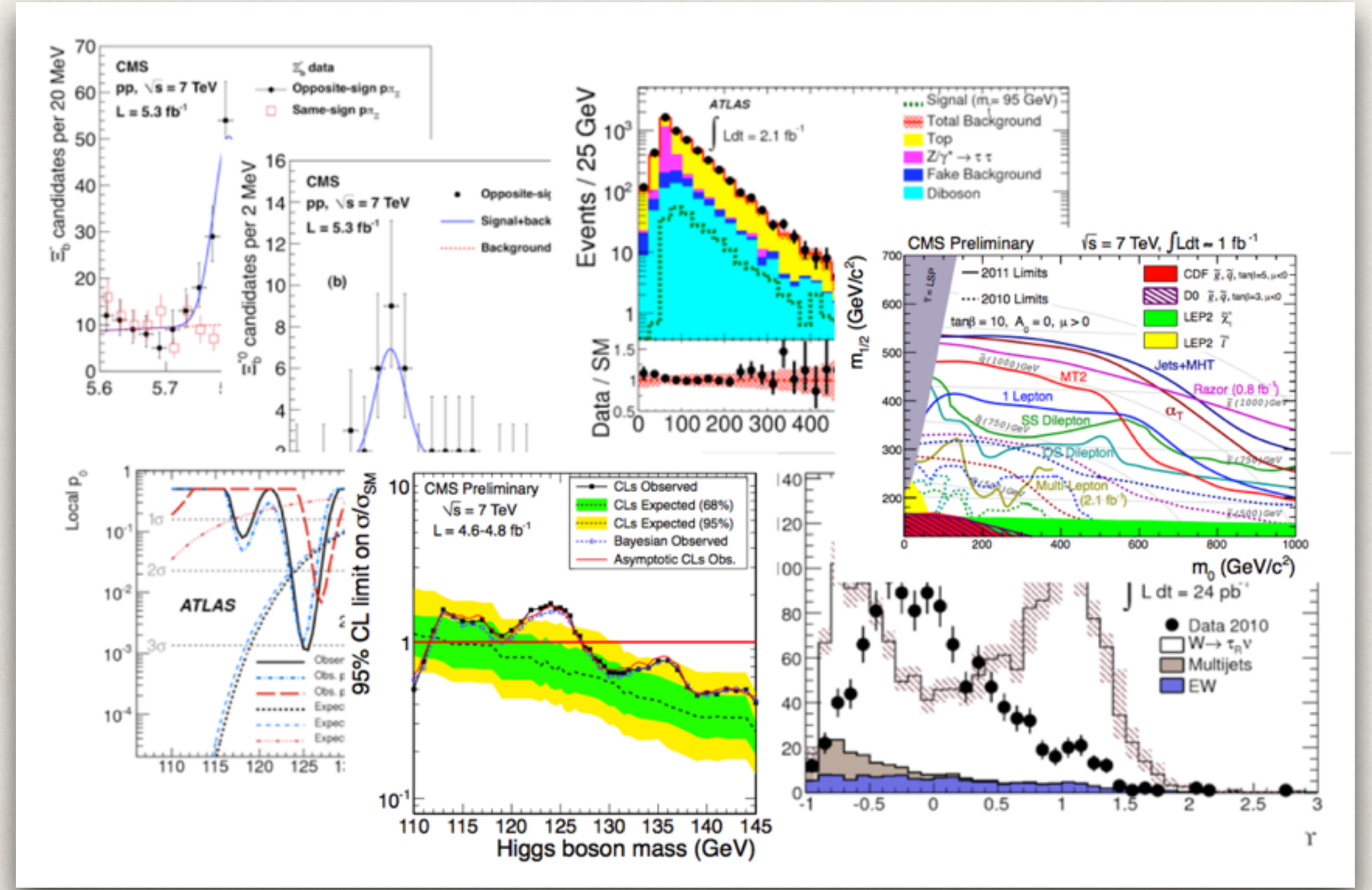
- Ever increasing number of users
  - 9336 forum members, 91582 posts, 1300 on mailing list
  - Used by basically all HEP experiments and beyond



As of today 177 PB of LHC data stored in ROOT format

ALICE: 30PB, ATLAS: 55PB, CMS: 85PB, LHCb: 7PB

ROOT in Plots



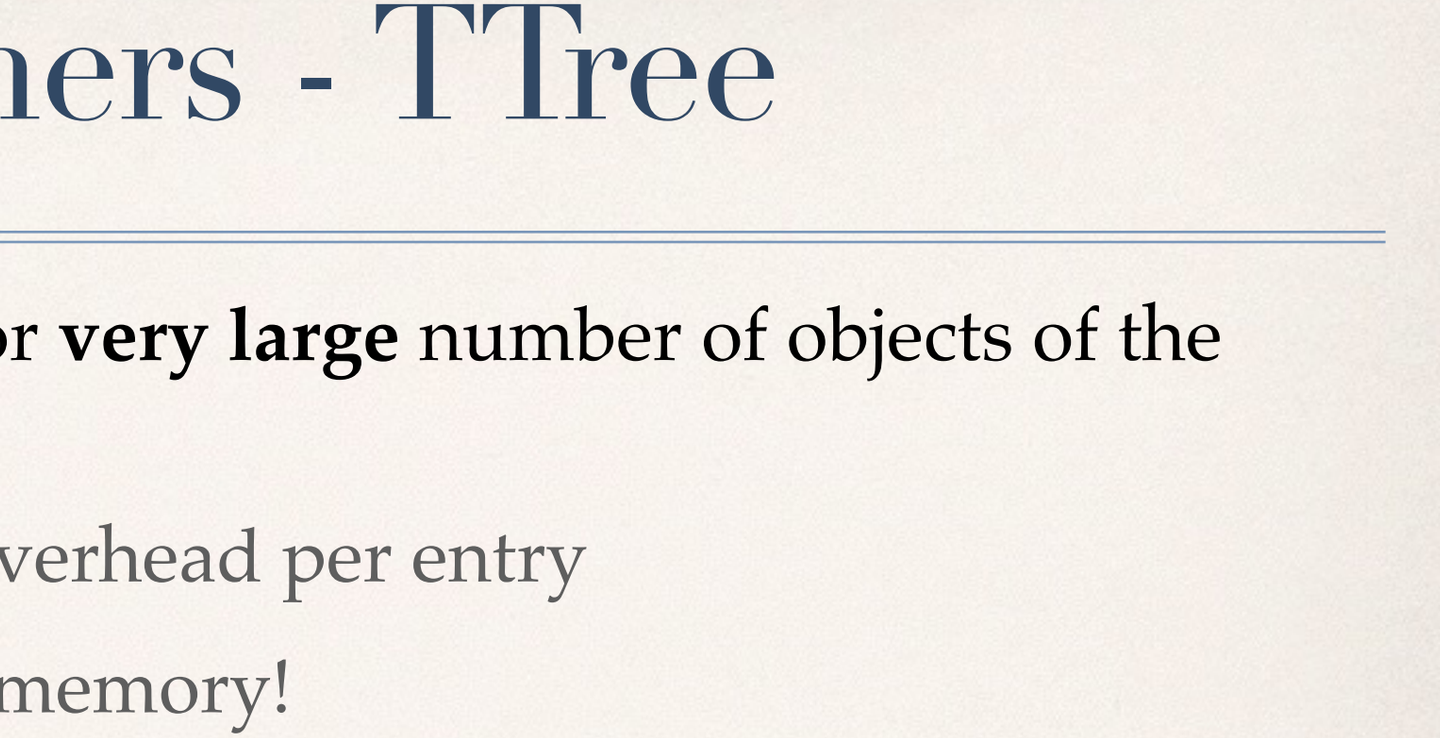
ROOT Object Persistency

- Scalable, efficient, machine independent, orthogonal to data model
- Based on object serialization to a buffer
- Object versioning
- Automatic schema evolution (backward and forward compatibility)
- Compression
- Easily tunable granularity and clustering
- Remote access
  - XROOTD, HTTP, HDFS, Amazon S3, …
- Self describing file format (stores reflection information)
- Used to store all LHC data (basically all HEP data)

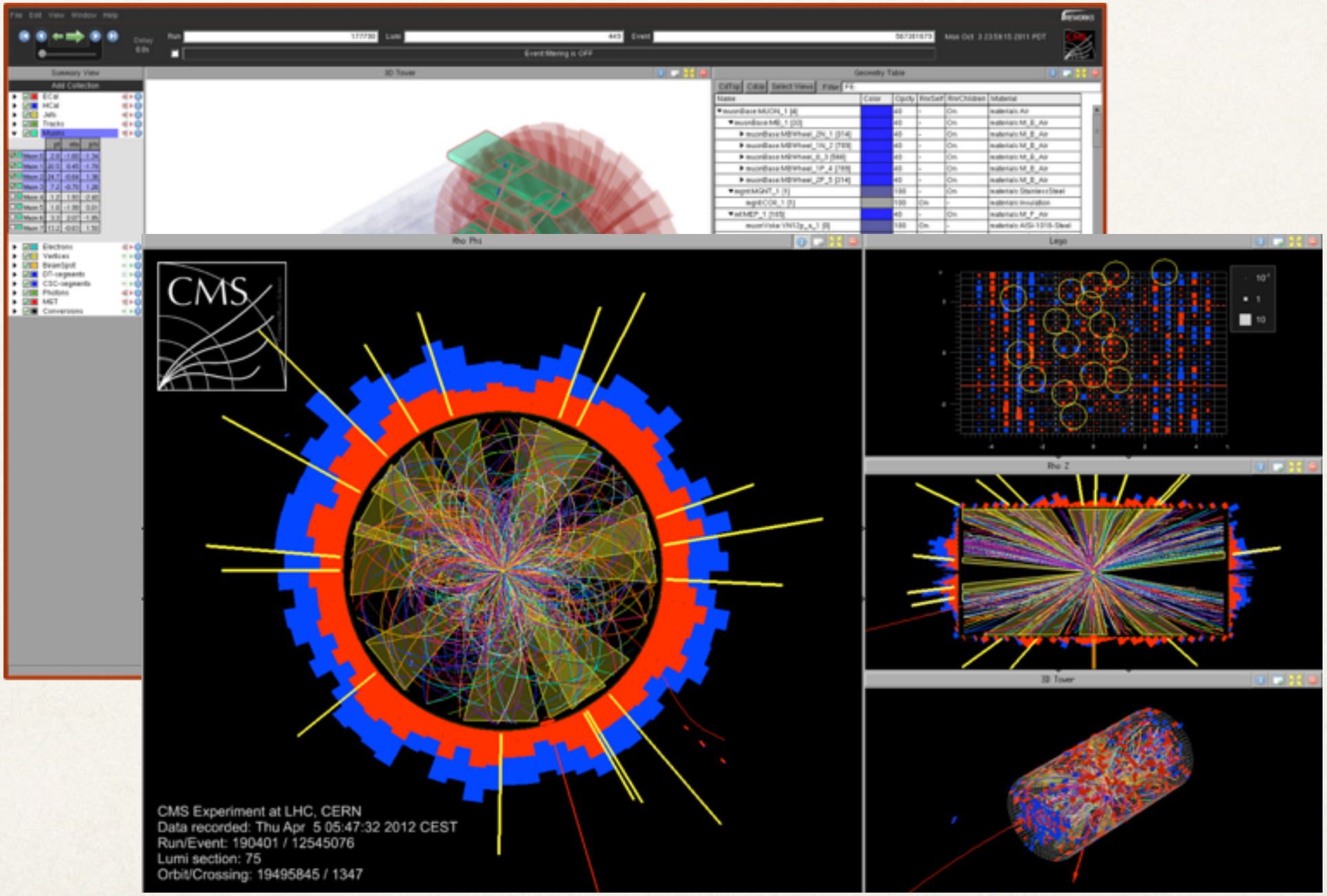
Object Containers - TTree

- Column-wise container for **very large** number of objects of the same type (events)
  - Minimum amount of overhead per entry
  - Typically not fitting in memory!
- Objects can be clustered per sub object or even per single attribute (clusters are called branches)
- Each branch can be read individually
  - A branch is a column

Physicists perform final data analysis processing large TTrees

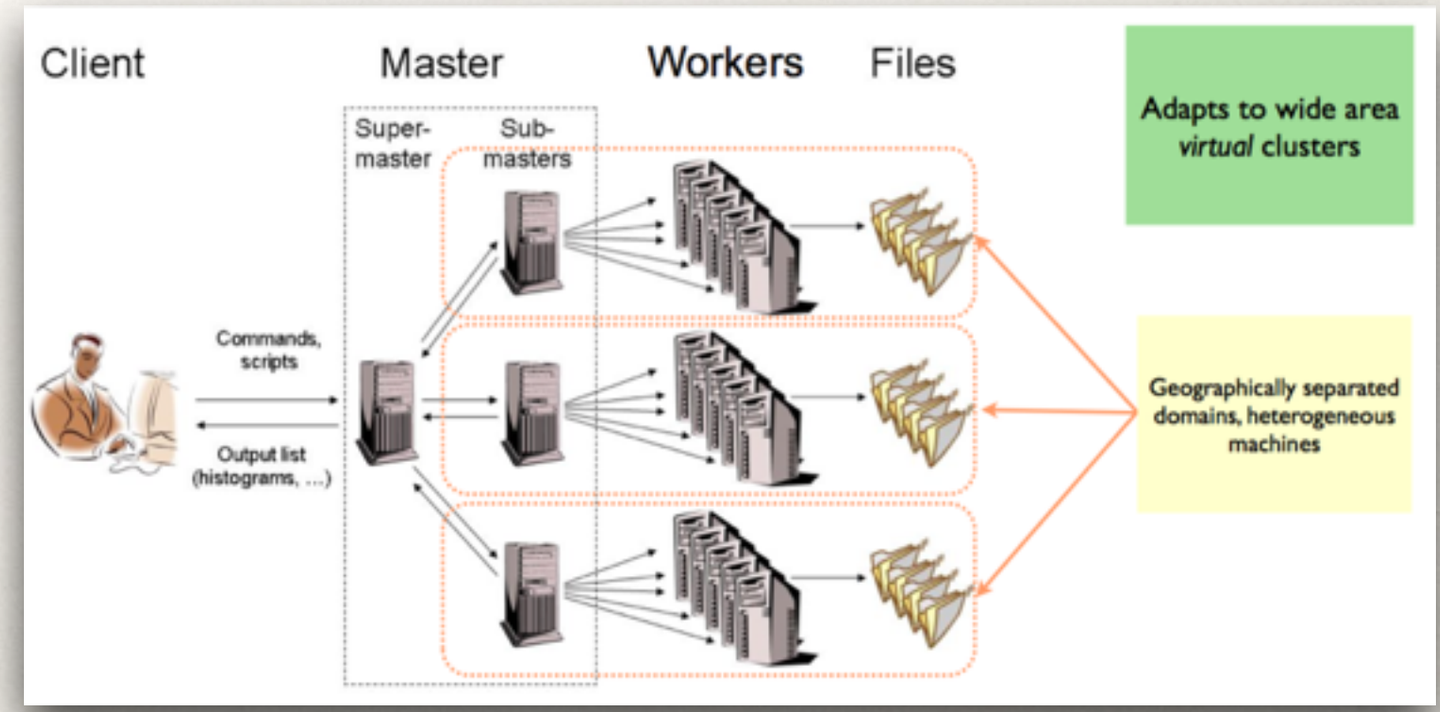


EVE Event Display



PROOF-The Parallel Query Engine

- A system for running ROOT queries in parallel on a large number of distributed computers or many-core machines
- PROOF is designed to be a transparent, scalable and adaptable extension of the local interactive ROOT analysis session
- For optimal CPU load it needs fast data access (SSD, disk, network) as queries are often I/O bound
- The packetizer is the heart of the system
  - It runs on the client/master and hands out work to the workers
  - It takes data locality and storage type into account
  - Tries to avoid storage device overload
  - It makes sure all workers end at the same time



Various Flavors of PROOF

- PROOF-Lite (optimized for single many-core machines)
  - Zero configuration setup (no config files and no daemons)
  - Workers are processes and not threads for added robustness
  - Once your analysis runs on PROOF Lite it will also run on PROOF
  - Works with exactly the same user code as PROOF
- Dedicated PROOF Analysis Facilities (multi-user)
  - Cluster of dedicated physical nodes
    - E.g. department cluster, experiment dedicated farm
  - Some local storage, sandboxing, basic scheduling, basic monitoring
- PROOF on Demand (single-user)
  - Create a temporary dedicated PROOF cluster on batch resources (Grid or Cloud)
  - Uses an resource management system to start daemons
  - Each user gets a private cluster
  - Sandboxing, daemon in single-user mode (robustness)

Current Version: ROOT 6

- An evolution of ROOT5
  - "Like before, but better"
- Old functionalities preserved, new ones added
- ROOT6 and ROOT5 are compatible:
  - Old ROOT files are readable with the 6 version
  - New ROOT files are readable with the 5 version
  - Old macros can be executed with ROOT6 (if written in proper C++, see next slides)

CLING

- Replaces CINT: a radical change at the core of ROOT
- Based on LLVM and CLANG libraries.
  - Piggy back on a production quality compiler rather than using an old C parser
  - Future-safe - CLANG is an active C++ compiler
  - Full support for C++11/14 with carefully selected extensions
  - Script's syntax is much stricter (proper C++)
  - Use a C++ just in time compiler (JIT)
  - A C++11 package (e.g. needs at least gcc 4.8 to build)
- Support for more architectures (ARM64, PowerPC64)

Proper C++ Interpreter

```
root [0] auto vars = gROOT->GetListOfGlobals()
(class TCollection *) 0x7f8df2b36690
root [1] for (auto && var : *vars) cout << var->GetName() << endl;
gROOT
gPad
gInterpreter
...
```

C++11

```
root [0] auto f=[](int a){return a*2;}
(class (lambda at ROOT_prompt_0:1:8) &) @0x109b7751c
root [1] f(2)
(int) 4

root [2] #include <random>
root [3] #include <functional>
root [4] std::mt19937_64 myEngine;
root [5] std::normal_distribution<float> myDistr(125.,12.);
root [6] auto myGaussian = std::bind(myDistr,myEngine);
root [7] myGaussian()
(typename __bind_return<_Fd, _Td, tuple<> >::type) 1.344781e
```

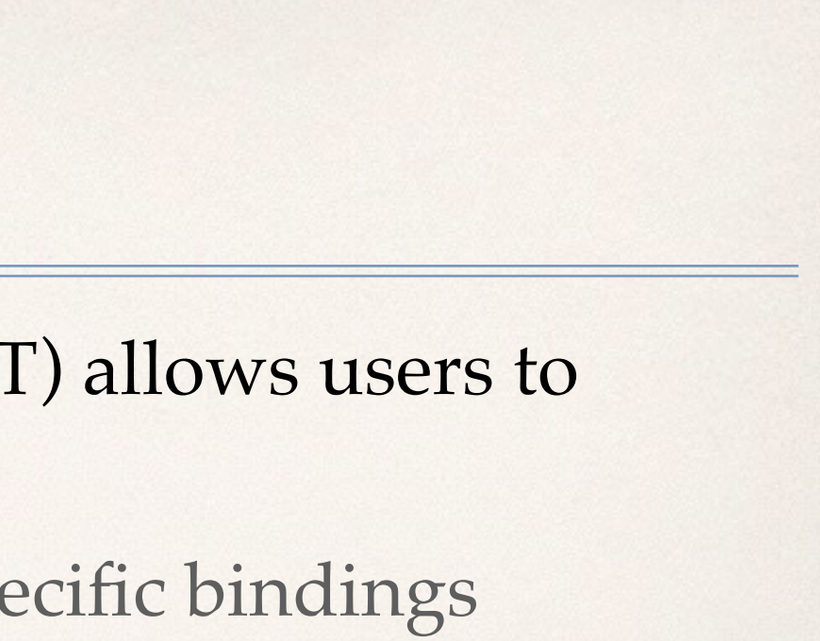
C++11

PyROOT

- The ROOT Python extension module (PyROOT) allows users to interact with any C++ class from Python
  - Generically, without the need to develop specific bindings
  - Mapping C++ constructs to Python equivalent
  - Give access to the whole Python ecosystem

```
# Example: displaying a ROOT histogram from Python
from ROOT import gRandom,TCanvas,TH1F
c1 = TCanvas('c1','Example',200,10,700,500)
hpx = TH1F('hpx','px',100,-4,4)
for i in xrange(25000):
   px = gRandom.Gaus()
   i = hpx.Fill(px)

hpx.Draw()
```



ROOT
Data Analysis Framework

Python without Dictionaries

A.h

```
#include <iostream>
class A {
 public:
  A(const char* n) : m_name(n){}
  void printName() {std::cout<< m_name
                    << std::endl;}
 private:
  const std::string m_name;
};
int dummy() {return 42;}
```

python

```
>>> import ROOT
>>> ROOT.gInterpreter.ProcessLine('#include "A.h"')
0L
>>> a = ROOT.A('my name')
>>> a.printName()
my name
>>> ROOT.dummy()
42
```

- Great potential with many 3rd party libraries!!

ROOT
Data Analysis Framework

Exploiting JIT: New TFormula

- Example of using the new JIT capability of LLVM/CLANG
- Pre-parsing of expressions (as before) but now using a real compiler
- I/O backward compatibility has been preserved

```
auto f = new TFormula(“F”,”[0]+[1]*x”);

auto f = new TFormula("F","[](double *x, double *p)
                          {return p[0]+p[1]*x[0];}”,1,2);
```

ROOT-R Interfaces

- R functions available in ROOT
- Wrapper classes for R data objects (e.g. TRDataFrame)
- R plugins for minimization
- R plugins for TMVA
- decision tree libraries (xgboost), neural network, and support vector machine packages

```
template<class T> T fun(T x) { return x+x; }

auto r = TRInterface::Instance();
r[“func”] << fun<TVector>;
r << “print(fun(c(0.5,1,1.5,2,2.5)))”;

1 2 3 4 5
```

MultiProc package

- Developed a new lightweight framework for multi-process applications
  - Inspired by the Python *multiprocessing* module
  - Idea to **re-implement Proof-Lite** using it
  - Distribute work to a number of fork()'d *workers,* then collect results
    - Main advantage: workers have access to complete 'master' state

```
TPool pool(8)

auto result = pool.Map(fun, args);

auto result = pool.MapReduce(fun, args, redfun);
```

defaults to # of cores

C/C++ function loaded macro std::function lambda

std::container initializer list TCollection& unsigned N

std::vector or TObjArray

Reduce function

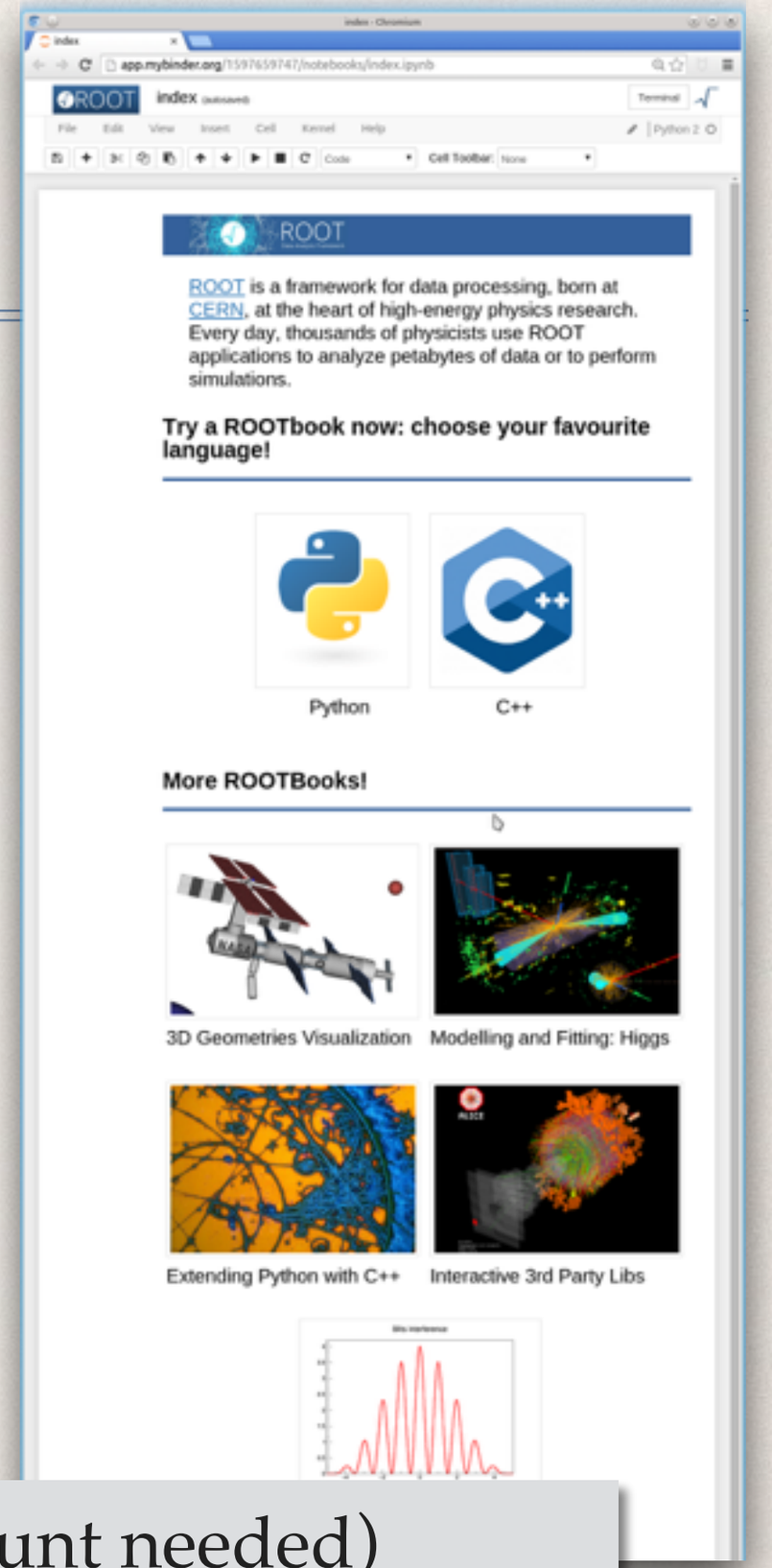
ROOT Data Analysis Framework

Build and Testing System

- ROOT uses the **CMake** cross-platform build-generator tool as a primary build system
  - Native windows builds, support for many build tools: GNU make, Ninja, Visual Studio, Xcode, etc
  - See instructions at https://root.cern.ch/building-root
    - Classic **configure/make** will still be maintained, but it will not be upgraded with new functionality, platforms or modules.
- Unit and Integration tests (~1400) have been migrate to **CTest**
- Binary installations are packaged with **CPack**
- Nightly and Continuous integration builds are automated and scheduled with **Jenkins,** as well as all the release procedures

ROOTbooks

- The integration of ROOT with the **Jupyter notebook technology** (ROOTBook) is well advanced
  - Ideal for training material
  - Possible way to document and share analysis
- Still some work to do in the following areas:
  - Disseminate the use of ROOTbooks
  - Make the JS visualization the default
  - Add R



Take a tour with Binder (no account needed)
http://root.cern.ch/notebooks/rootbinder.html

Development Beyond ROOT 6

Main Challenges

- ROOT is 20 years old, and some parts **require re-engineering and modernization**
  - Need to **exploit modern hardware** (many-core, GPU, etc.) to boost performance
  - **Modernize implementations** (C++11 constructs, use existing libraries, etc.)
  - **Modernize C++ API** to improve robustness eventually giving up on backward/forward compatibility
- Require the collaboration of the community to **ensure evolution and sustainability**
  - **Facilitate contributions** to ROOT without engaging our responsibility in the maintenance and user support
  - Layered software modules or plugins that can bring new functionality to the end-users

Development Main Directions

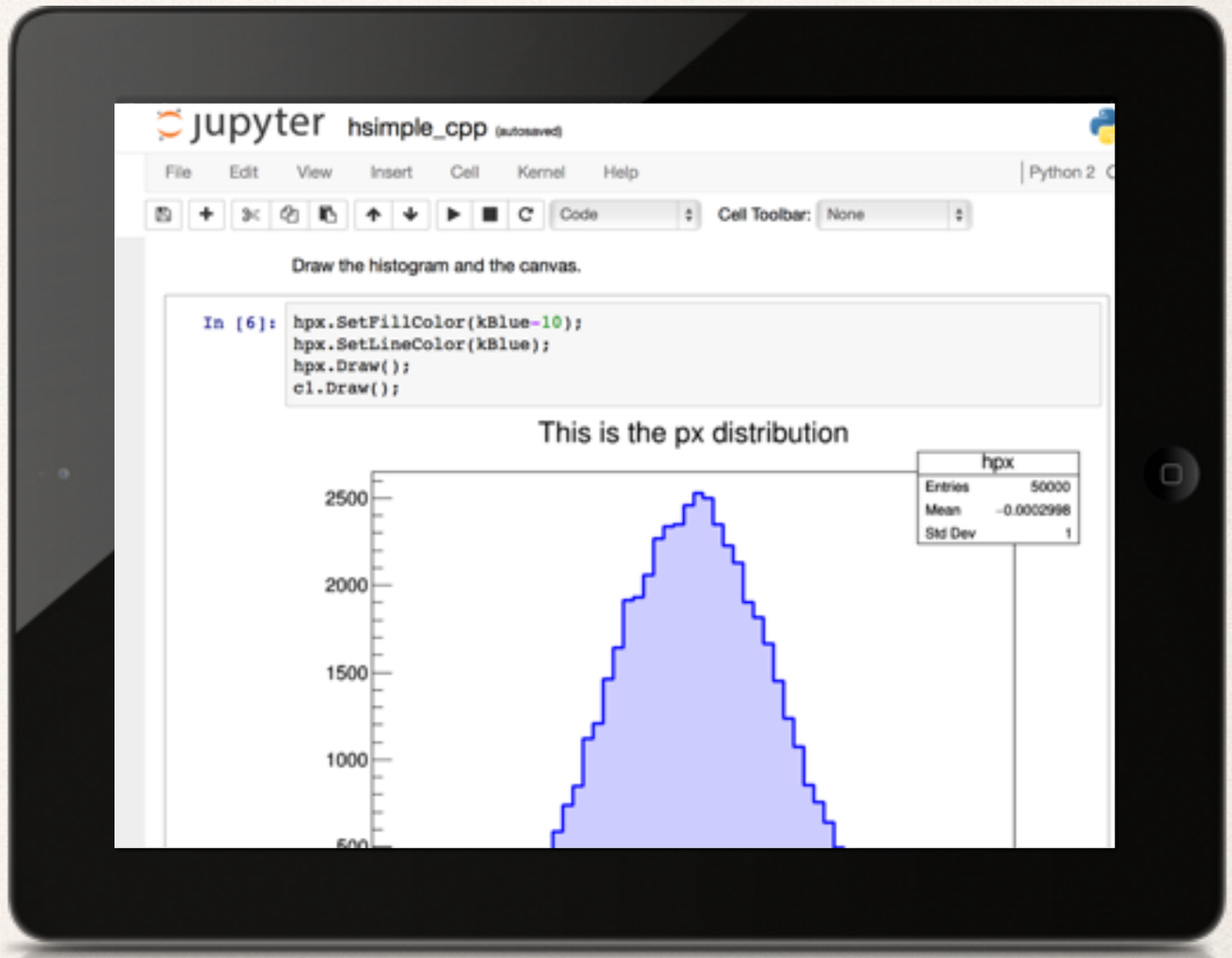
- **Cling Interpreter and its full exploitation**
  - C++11/14, JIT compilation opens many possibilities (automatic differentiation, improved interactivity, etc.)
- **Modern C++ and Python interfaces**
  - Make ROOT simpler and more robust. Explore better C++ interfaces making use of new standards (C++14, C++17)
- **Machine Learning**
  - Modernize and improve TMVA with new and more flexible design and integration of new methods
- **Parallelization**
  - Seek for any opportunity in ROOT to do things in parallel to better exploit the new hardware (e.g. Ntuple processing, I/O, Fitting, etc.)

Development Main Directions (2)

- **Packaging and modularization**
  - Incorporate easily third party packages (e.g. VecGeom in TGeom)
  - Build/install modules and plugins on demand. Facilitate contributors to provide new functionality
- **Re-thinking user interface**
  - New ways to provide thin-client web-based user interfaces
  - Javascript and ROOTbooks (Jupyter notebooks)
- **ROOT as-a-service**
  - Thin client plugged directly into a ROOT supercomputing cloud, computing answers quickly, efficiently, and without scalding your lap

ROOT as-a-Service

- Combines naturally the work on **parallelization** to exploit many cores and nodes together with the new **web-based interface** to provide a modern and satisfying user experience
  - Build on top of cloud security, storage and compute services
- Working towards a **Pilot Service** that is able to serve requests for a medium number of users (O(100))
  - Implemented using the CERN palette of IT services



See for example presentation at CS3, ETH Zurich

ROOT
Data Analysis Framework

ROOT Versions

- **5.34 - current legacy production version**
  - ROOT 5 is frozen except for critical bug fixes
- 6.02
  - First functionally complete ROOT 6 version. Superseded by 6.04
- 6.04
  - Used by the LHC experiments in production
- **6.06 - current production**
  - Released beginning of December 2015
  - 6.06/02 - latest patch release
- 6.07/04 - development release
  - Testing latest features

ROOT
Data Analysis Framework

Collaborations

- ROOT has currently many external contributors:
  - in 2015 we had commits from 57 different authors
- Active Collaborate with us and Ideas web page
- Interest of other projects to collaborate with ROOT
  - DIANA
    - Data Intensive ANAlysis, 4-year NSF funded
    - Focus on analysis software, including ROOT and its ecosystem
    - Three primary goals: performance, interoperability, support for collaborative analysis
  - Other initiatives in the pipeline
- Need to integrate these collaborations and achieve a coherent program of work

HEP Software Foundation

- Goals
  - Share expertise, raise awareness of existing software and solutions
  - Catalyze new common projects, promote commonality
  - Aid in creating, discovering, using and sustaining common software
  - Support training career development for software and computing
  - Framework for attracting effort and support to common projects
  - Provide a structure to set priorities and goals for the work
  - Facilitate wider connections; while it is a HEP community effort, it should be open to form the basis for collaboration with other sciences

http://hepsoftwarefoundation.org

Activities and Working Groups

Working Group	Objectives	Forum - Mailing list
Communication and information exchange	Address communication issues and building the knowledge base Technical notes	hep-sf-tech-forum
Training	Organization of training and education, learning from similar initiatives	hep-sf-training-wg
Software Packaging	Package building and deployment, runtime and virtual environments	hep-sf-packaging-wg
Software Licensing	Recommendation for HSF licence(s)	hep-sf-tech-forum
Software Projects	Define incubator and other project membership or association levels. Developing templates	hep-sf-tech-forum
Development tools and services	Access to build, test, integration services and development tools	hep-sf-tech-forum

HSF Workshop, Paris May 2-4, 2016

Conclusion

- ROOT is one of the main ingredients in the software of each HEP experiment
  - Development started in 1995 and being improved continuously
  - Data storage, statistical data analysis and scientific graphics are the main features
  - Used in many other scientific fields, like astrophysics and biology but also in finance
- Development ideas following the axes:
  - Modernization of C++ & Python API, machine learning, parallelization, packaging, web interfaces and ROOT as a service.
- Collaborations and contributions most welcome
  - Major player in the HEP Software Foundation