ROOT's Place and its Capabilities in the HEP Analysis Ecosystem

HEP analysis ecosystem workshop
22-24 May 2017, Amsterdam

P. Mato for the ROOT Core Team

Introduction

- ROOT is at the root of the software of basically every HEP experiments
  - 100s of PB of data of the LHC experiments are stored in ROOT format
  - Data Analysis is performed using ROOT functionality (TTree, RooStat, TMVA, etc.)
  - The majority of the HEP plots are made with ROOT
- ROOT is more than 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 (modern C++, use existing libraries, etc.)
  - Modernize C++ and Python APIs to improve robustness
- Require the collaboration of the HEP community to ensure its evolution and sustainability
  - Need to facilitate contributions to ROOT

ROOT Development Team

- Project initiated by Rene Brun and Fons Rademakers back in 1995
- Many authors have contributed to ROOT since then
  - From large packages, to new plugins, to additional features, many bug fixes, etc.
  - For example: about **90 different authors in the last 12 months** have contributed to ROOT
- Core team responsibilities
  - Design and development of the ROOT core libraries
  - Integration, testing, maintenance, releases, etc.
  - User support
- Today's members
  - G. Amadio, B. Bellenot, P. Canal (FNAL), O. Couet, G. Ganis, E. Guiraud, R. Isemann, P. Mato, L. Moneta, A. Naumann, D. Piparo, M. Storo Nyflott, E. Tejedor, X. Valls, V. Vassilev (Princeton),

ROOT Main Components

- **Core** (133 kLOC)
  - Base classes, utilities, containers, etc.
  - Users encouraged to use std:: equivalent if possible
  - C++ introspection classes
- **Cling** (31 kLOC)
  - C++ interpreter based on LLVM/Clang
- **Graf2** (370 kLOC)
  - Core 2D graphics classes
  - Modules: cocoa, asimage, fitsio, freetype, gviz, mathtext, postscript, qt, quartz, win32gdt, x11
  - Deprecating some parts: qt
- **Graf3** (83 kLOC)
  - Core 3D graphics classes
  - Modules: ftgl, gl, g3d, gviz3d, x3d

ROOT Main Components (2)

- **Hist** (107 kLOC)
  - Histograms classes, spectrum class, painters
- **Gui** (114 kLOC)
  - GUI, GUIBuilder, interface to Qt (deprecated)
  - To be replace in the long-term (see later)
- **Eve** (30 kLOC)
  - Event display classes
- **PyROOT** (20 kLOC)
  - Python bindings to any C++ class
- **JSROOT** (32 kLOC)
  - Re-implementation of ROOT graphics for web browsers (JavaScript)

ROOT Main Components (3)

- **I/O** (48 kLOC)
  - Basic ROOT I/O classes
  - Storage plugins: castor, chirp, dcache, gfal, hdfs, rfio, sql, xml
- **TTree** (62 kLOC)
  - The heart of the ROOT based analysis
- **Math** (148 kLOC)
  - Libraries: MathCore, MathMore(based on GSL), fttw, foam, fumili, generic, genvector, matrix, minuit, mlp, physics, quadp, rtools, smatrix, splot, unuran
- **RooFit** (111 kLOC)
  - Toolkit for modeling expected distributions and fitting
- **TMVA** (76 kLOC)
  - Multivariate analysis toolkit

ROOT Main Components (4)

- **Proof** (78 kLOC)
  - Parallel ROOT
  - Not developed anymore
- **Sql** (76 kLOC)
  - Interfaces to relational databases: mysql, odbc, oracle, pgsql, sapdb, sqlite
- **Net** (63 kLOC)
  - Interfaces to network protocols: xroot, davix, http, alien, auth, bonjour, glite, globus, krb5auth, ldap, monalisa, rootd

Language Bindings and Interfaces

- Python
  - Use any C++ class from Python as well as calling Python from ROOT.
  - P2 and P3 are supported
- R
  - Call any R function from C++. Opens the possibility to access the very large set of mathematical and statistical tools provided by R
- Ruby
  - Not maintained anymore
- Mathematica
  - Basic call out capability (C interface)
- JavaScript
  - Able to read ROOT data files from a web browser
  - Re-implementation of ROOT graphics.

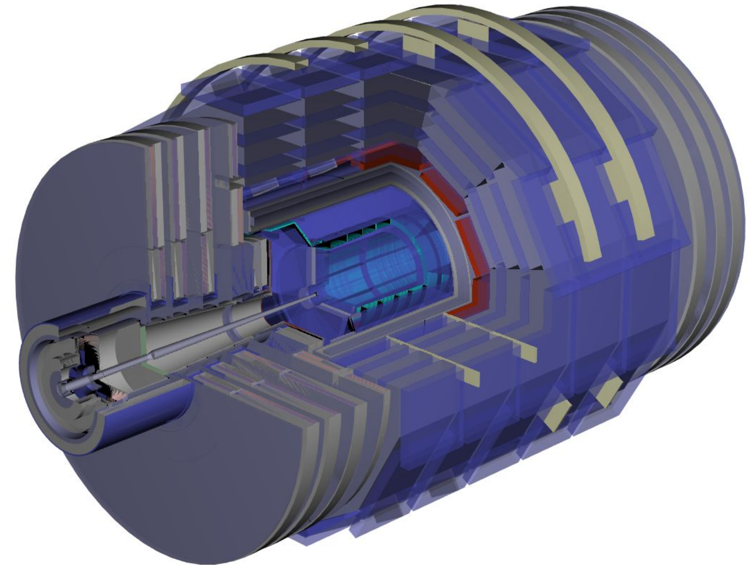
PyROOT

- The ROOT Python extension module (PyROOT) allows users to interact with any C++ class from Python
  - Generically, without the need to develop specific bindings using the C++ reflection information
  - Much less work than using other binding tools (e.g. boost, swig)
- Uniform mapping of C++ idioms to Python equivalent
  - Same behavior everywhere
- Standard “Pythonizations” of C++ classes and constructs
  - E.g. containers, smart pointers, etc.
- Give access to the whole Python ecosystem
  - ROOT classes and functionality first class elements in the ecosystem

```python
# 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()
```

JSROOT

- Implementation of the ROOT graphics for web browsers in JavaScript
    - Possibility to interactively open ROOT files and draw objects like histograms or canvas
    - Different implementation, similar look-and-feel
- Display of TGeo objects
    - Visualization of detectors
- Able to interact with **THttpServer** that provides access to all objects in a running ROOT application
    - Interchange format: JSON
- Graphics for the **Jupyter notebooks**



ROOT Versions

- 5.34 - current legacy production version
  - ROOT 5 is frozen except for critical bug fixes
- 6.08 - current production
  - Released November 2016
  - Current patch 6.08/06 (March 2017)
- 6.10 - next production version
  - Expected release June 2017
  - Highlights
    - New TDataFrame class to allow interaction with data stored in columnar format in a functional and intuitive way in order to perform data analysis
    - New classes to express parallelism (MT, MP)

Platforms and Compilers

- Portability across platforms has been one of the important requirements
  - Larger user reach
  - Improved software quality
- ROOT 5 (frozen)
  - Linux (32, 64), openbsd, freebsd7, MacOS (64), Windows (32)
  - Intel, arm64, ppc32
  - GCC >= 4.8, Clang, ICC >= 13, VisualStudio >= 2008,
- ROOT 6
  - Linux (32, 64), openbsd, freebsd7, MacOS (64)
  - Intel, arm64, ppc64le (in preparation)
  - GCC >= 4.8, Clang ICC >= 16 (17 in preparation), Clang

Management of ROOT Externals

- Supporting two main use-cases
  - **Easy building of ROOT standalone**
  - **Easy integration of ROOT in larger software stacks**
- Basic ROOT requires a limited number of external libraries (e.g. pcre, lzma, zlib, freetype, libafterimage, x11, cocoa, etc.)
- Most of the optional components and plugins require additional externals (e.g. python, graphviz, libxml2, mysql, occ, vecore, tbb, gsl, xrootd, davix, fftw3, openssl, unurun, qt, …)
- ROOT will try to locate these external libraries in the system if the corresponding option is enabled
  - For some of them, a ‘builtin_xxx’ option is available to build ROOT with the external XXX bundled together (standable installations)

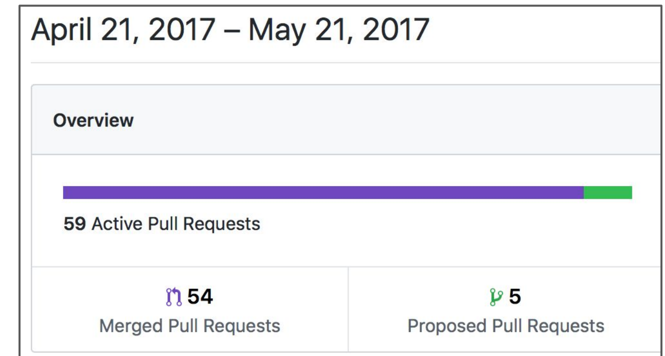
Building on top of ROOT

- ROOT is a set of libraries used by larger software stacks
- Several artifacts are delivered to facilitate this task (e.g. root-config, ROOTConfig.cmake)

```
cmake_minimum_required(VERSION 3.0 FATAL_ERROR)
project(event)
#---Tell CMake where to find the ROOT installation.
list(APPEND CMAKE_PREFIX_PATH $ENV{ROOTSYS})
#---Locate the ROOT package and defines a number of variables (e.g. ROOT_INCLUDE_DIRS)
find_package(ROOT REQUIRED)
#---Define useful ROOT functions and macros (e.g. ROOT_GENERATE_DICTIONARY)
include(${ROOT_USE_FILE})
#---Create a shared library with geneated dictionary
ROOT_GENERATE_DICTIONARY(EventDict Event.h MODULE Event LINKDEF EventLinkDef.h)
add_library(Event SHARED Event.cxx EventDict.cxx)
add_dependencies(Event EventDict)
target_link_libraries(Event ROOT::Hist ROOT::Tree)
#---Create  a main program using the library
add_executable(Main MainEvent.cxx)
target_link_libraries(Main Event)
```

Development Process

- Programme of work
  - Discussed yearly and added to JIRA
- Release schedule
  - Discussed with experiments and agreed (1 or 2 main releases/year)
- Quality assurance
  - Many tests implemented (unit, integration, tutorials, regression, etc.)
  - CI (Jenkins, GitHub)
  - Static analysis (coverity), test coverage, etc.
- Contributions
  - Since migration to GitHub as the master repository, **Pull Requests** are the prefered way to get contributions
  - Automatically checked (next slide)

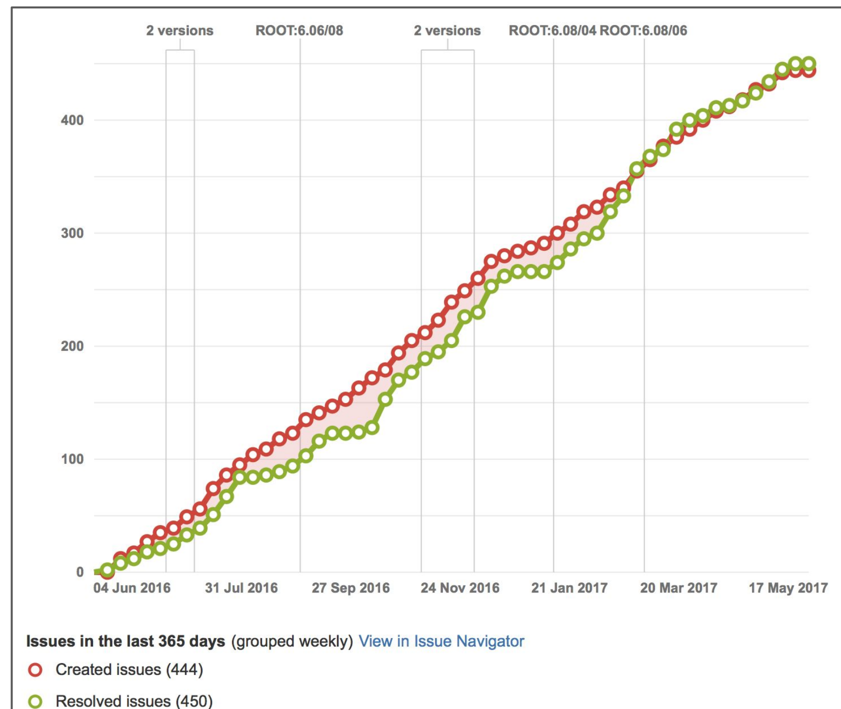


Contributions Quality Assurance

- The ROOT migration to GitHub made contributing easier!
- Adopted an tool-aided contribution review model to automatically give feedback to contributors if there are defects
    - **Clang-tidy** - static analysis tool, currently detects problems such as null pointer dereferences dead code. Soon code modernisation, performance checks,
    - **Clang-format** - coding conventions checker tool
    - **Jenkins Pipelines** - checking that modifications build successfully and pass all tests on a number of OS and compiler combinations
    - **In-tree Google Test** - tool helping develop unit tests easier

User Support

- ROOT Forum (Discourse)
  - Main channel to provide user support
  - 23k topics, about 10 new topics/day, ~100 new posts/day
- Bug Tracker (JIRA)
  - Bugs and Tasks are tracked
  - Large backlog of issues (~769 bugs)
- GitHub PR
  - ~600 PR has been created
- Other User Feedback
  - Fortnightly planning meetings (critical bugs, release schedule, progress reports, etc.)
  - 2015 ROOT Users' Survey (prepared for the Saas-Fee Workshop)



Main development directions in 2017

- C++ modules
- I/O performance
- Parallelization
- SIMD, MT and MP in Math and Statistical Libraries
- Machine Learning
- Modularization
- Windows

See for more details https://root.cern.ch/program-work

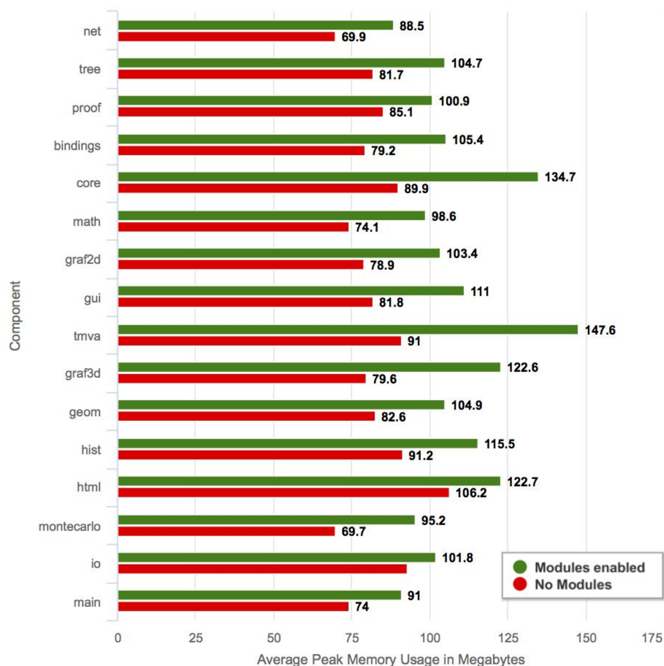
Exploration Work in 2017

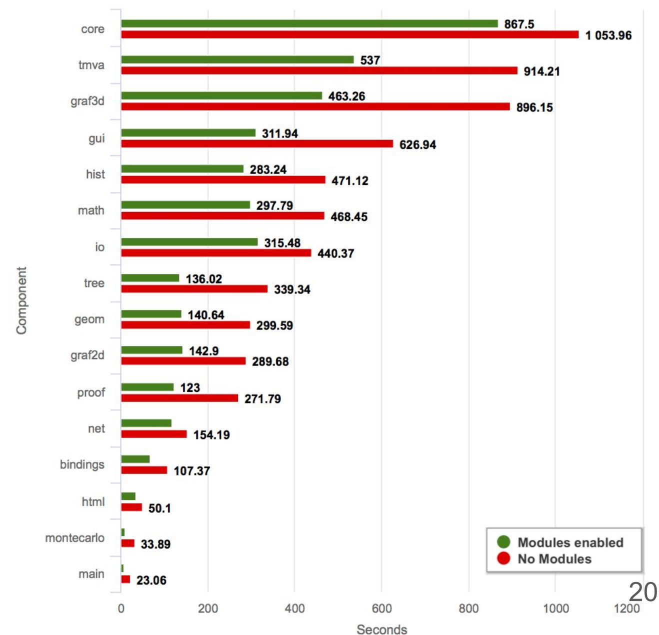
- New C++ interfaces
- New analysis approaches: functional chains, Spark clusters
- Web-based graphics (JS) to replace ROOT graphics and GUI

See for more details https://root.cern.ch/program-work

C++ Modules

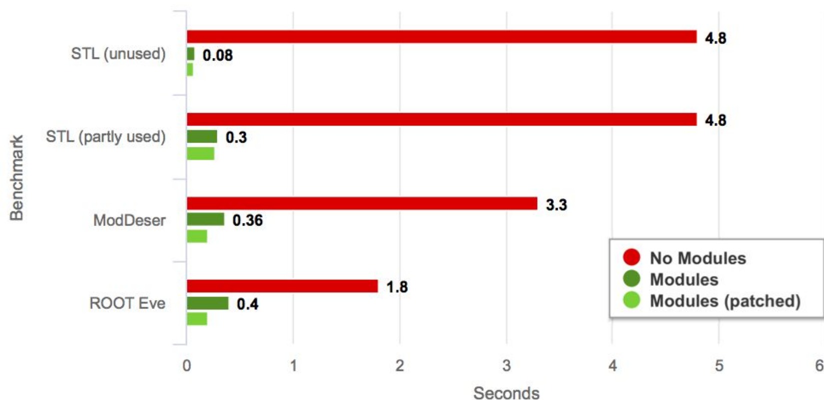
Reduce the memory size and improve performance when running third party (experiment code) in ROOT. Nightly builds performance:

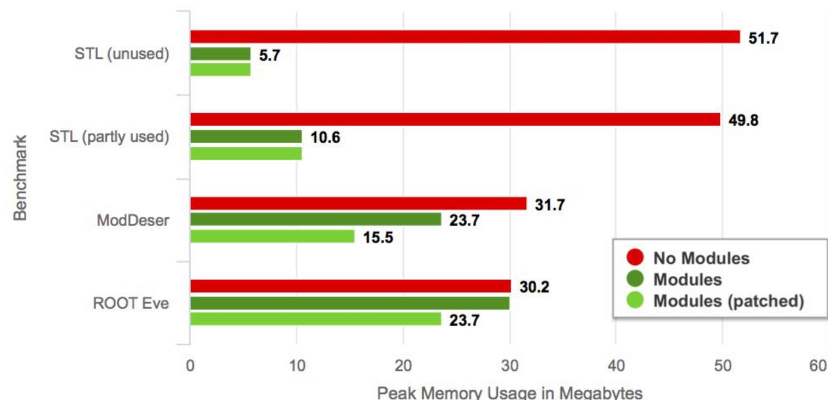




C++ Modules. Preliminary benchmarks

Work in progress to implement C++ Modules in cling, optimizing ROOT's runtime performance. Preliminary benchmarks:





I/O Performance

- Multi-threaded file operations
  - Introduce a new TFile class, potentially less performant than the existing one, but featuring thread safe methods
  - Parallel write
- Optimizations
  - Improved compression of branches holding a non-split collection
  - Per-entry compression
  - Reduction of overhead of deep inheritance chains
- Comparisons with other data formats (Parquet, Avro, Kudu, etc.)
  - See next talk

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.)
  - Parallelization under the hood
- Developing a new lightweight framework for both multi-process and multi-threading applications (TThreadExecutor, TProcessExecutor)
  - Task-oriented inspired by the Python multiprocessing module
  - Idea to re-implement Proof-Lite using it
- Take advantage of implicit multi-threading (IMT) for writing/reading large TTrees, etc.
- Merging efficiently the output objects produced by the parallel tasks
- Introduce thread-safety where needed (e.g. I/O)

SIMD, MT and MP in Math and Statistical Libraries

- Integration of VecCore in ROOT
  - Adding new vector types
- Extend Math functions and algorithm to have a vectorized interface
  - Support SIMD vectorization when fitting
  - Vectorize adaptive numerical integration
- Support for Multi-Thread and Multi-Process parallelization in fitting when evaluating least-square and log-likelihood functions
- Parallelize numerical integration based on Monte-Carlo methods (e.g. Vegas)
- Support in RooStats multi-process evaluations for statistics calculators requiring multiple fits (e.g. toy MC generation and fitting)

Machine Learning

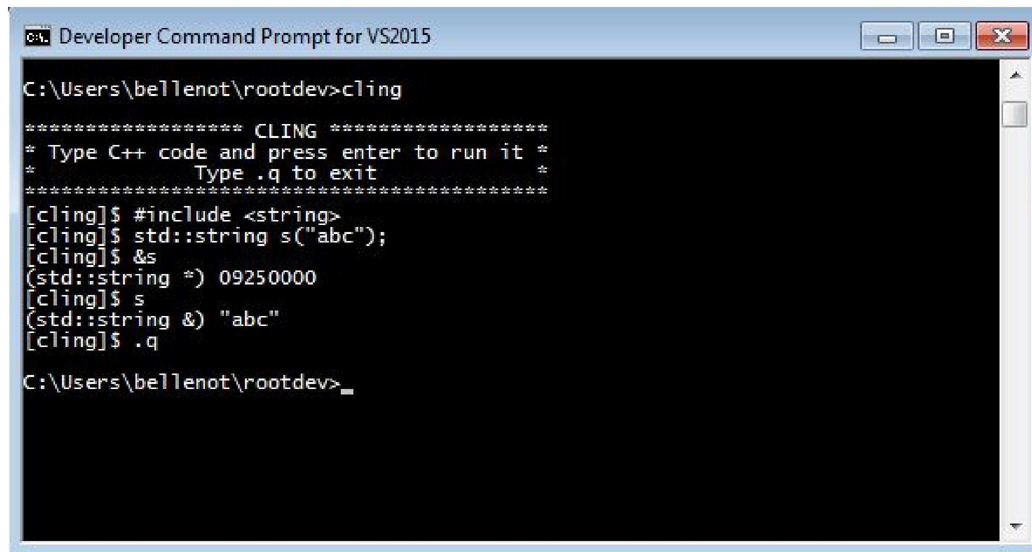
- Very active development in TMVA
  - E.g. added in 2016 a new DNN with parallelization support for CPU and GPU
  - Good contributions expected also this year (GSOC, doctoral and summer students)
- Upcoming features
  - Complete and fully support parallelization (e.g. in cross validation, hyper-parameter tuning..)
  - Integrate more pre-processing tools (auto-encoders, unsupervised training for DNN initialization, etc..)
  - Extend Deep Learning to Convolutional and Recurrent Neural Networks
  - Fully support multi-class classification
  - Introduce multi-class regression
- Performance improvements in ML methods (especially in their evaluation)
- Modernize TMVA interfaces
  - Integrate new Python API developed for notebooks to PyROOT
  - Facilitate interoperability with external ML tools written in Python

Modularization

- Essential for opening ROOT to collaborators willing to contribute with a large chunk of functionality
- Develop a model for building, distributing and deploying **ROOT modules** that can extend an existing installation of ROOT on demand
- Most of the non-core ROOT functionality can then be distributed as modules
  - E.g. package the current ROOT TFile plugins as ROOT modules
  - Evolve ROOT into `BOOT` (à la R)
- Cope with package/module dependencies by interfacing ROOT with popular package managers (e.g. Spack, Conda, …)

Windows Support

- Very good progress towards ROOT 6 running on Windows
  - Cling compiles and runs on Windows with Visual Studio 2015
  - Most of the cling tests passes
- What Next
  - Add missing Windows specific parts in CMake
  - Build ROOT
    - Rootcling (dictionaries)?
    - Implement Windows equivalent of some posix code
  - Run ROOT with its tests (CTest)



New C++ Interfaces

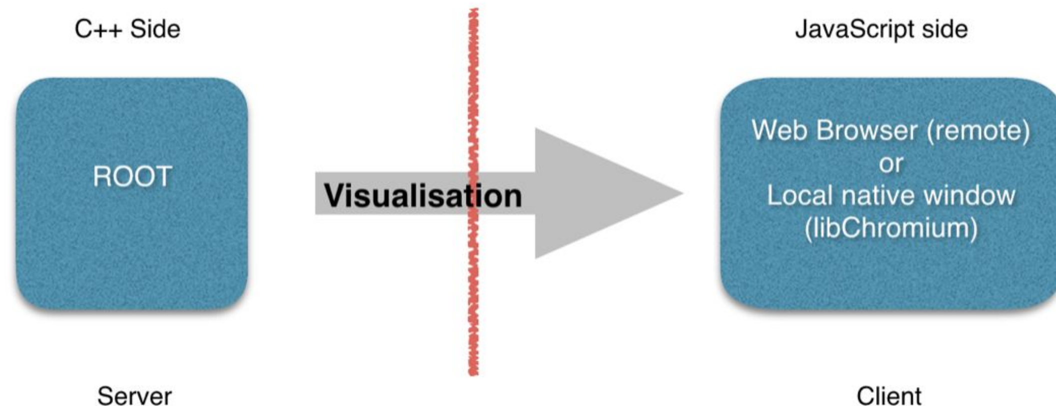
- The goal is to make using ROOT more robust and simple
  - think current PhD students versus PAW-style memory management
- Redesign interfaces using current C++ where it simplifies and clarifies
  - Be minimal (one trick pony) and explicit (threading, ownership, dependencies)
  - Use wording we know, to ease transition (+ tooling?)
- Backward incompatible change
  - We can do things we dreamt of but couldn't do. Unlocks development energ
- Continuous flow of interface upgrades started with THist, TFile; now TDataFrame; soon graphics

New Analysis Approaches

- Improve the current ROOT **notebook** interface
    - Tab-completion, definition of functions in code cells (without magic), JSROOT as default graphics, etc.
- Exploration and prototyping of **functional chains**
    - The User specifies the **What** and ROOT decides the **How** with all sort of optimizations
    - **TDataFrame**: Analyse data stored in TTrees with a higher level interface
    - Both C++ and Python interfaces
- Assess the Apache **Spark** as scheduler in combination with the PyROOT interface to complement PROOF for distributed calculations
    - Distributed processing of large TTrees
    - Interface with Spark clusters
    - Interface IT container service

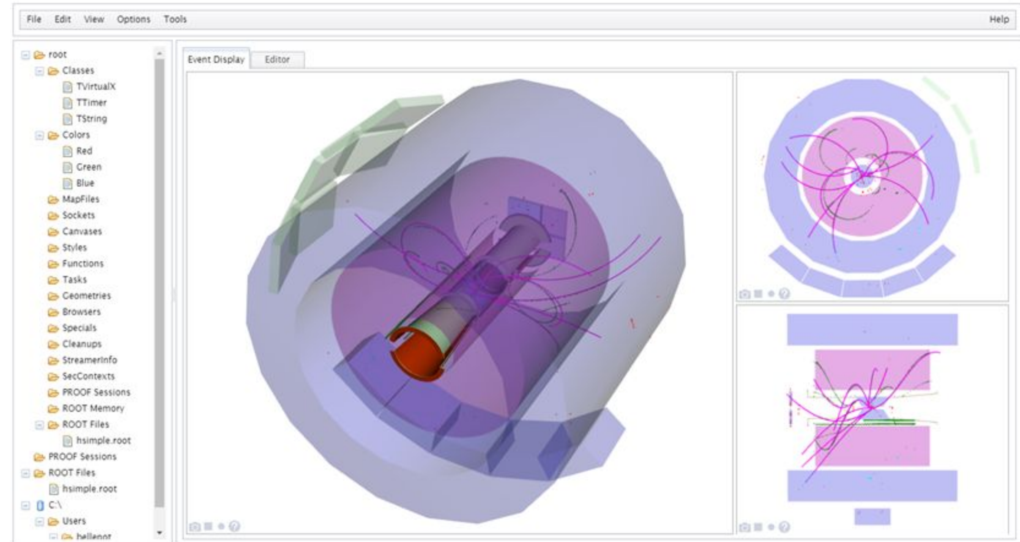
Web-based graphics (JavaScript)

- Replacement of the ROOT graphics making use of modern and powerful JavaScript based graphic libraries
- ROOT applications will generate graphics from the C++ side (server) and display them with JavaScript (client)
    - The client will be a remote web browser of an embedded browser in the ROOT application



Web-based GUI (JavaScript)

- Need to support future (and current) windowing systems; HTML just works
- The GUI (widgets) is on the client side (HTML, JavaScript, …).
- The GUI action triggers either a JavaScript function call (on the client side), or pass a formatted C++ method to be called on the server side.
- On the server side (C++) the ROOT THttpServer class is expected to handle the events.
- Users' GUI could be created using any web GUI toolkit



Summary

- Attempted to give a quick overview of the ROOT components and the current software development process
    - In particular the recent efforts to facilitate contributions from the community
- A lot of Core Team effort goes into support and maintenance
    - The team is aware that we need to do even more
- Exposed the main development directions in the near future
    - Modernization, performance, papalelization, modularity, new interfaces, etc.
- ROOT as a key player in the future Analysis Ecosystem requires eventually to refine and evolve some aspects
    - Language interoperability, openness, modularity, collaboration model, etc.

Guidelines - Overview

- Overview of the ROOT components meant for HEP analysis
  - Core, Hist, I/O, etc.
  - Which are the key components from the ROOT team's point of view?
- ROOT component modularity
  - What level of independence is there at the moment, and how is this going to evolve?
  - How a new component can be developed/provided now?
- Language/Build support
  - Programming languages supported for "ROOT analysis"
  - Support provided for building code against ROOT
- Machine learning support
- Near term plans
  - Including ROOT 7 of course