CMS and ROOT6

David J Lange

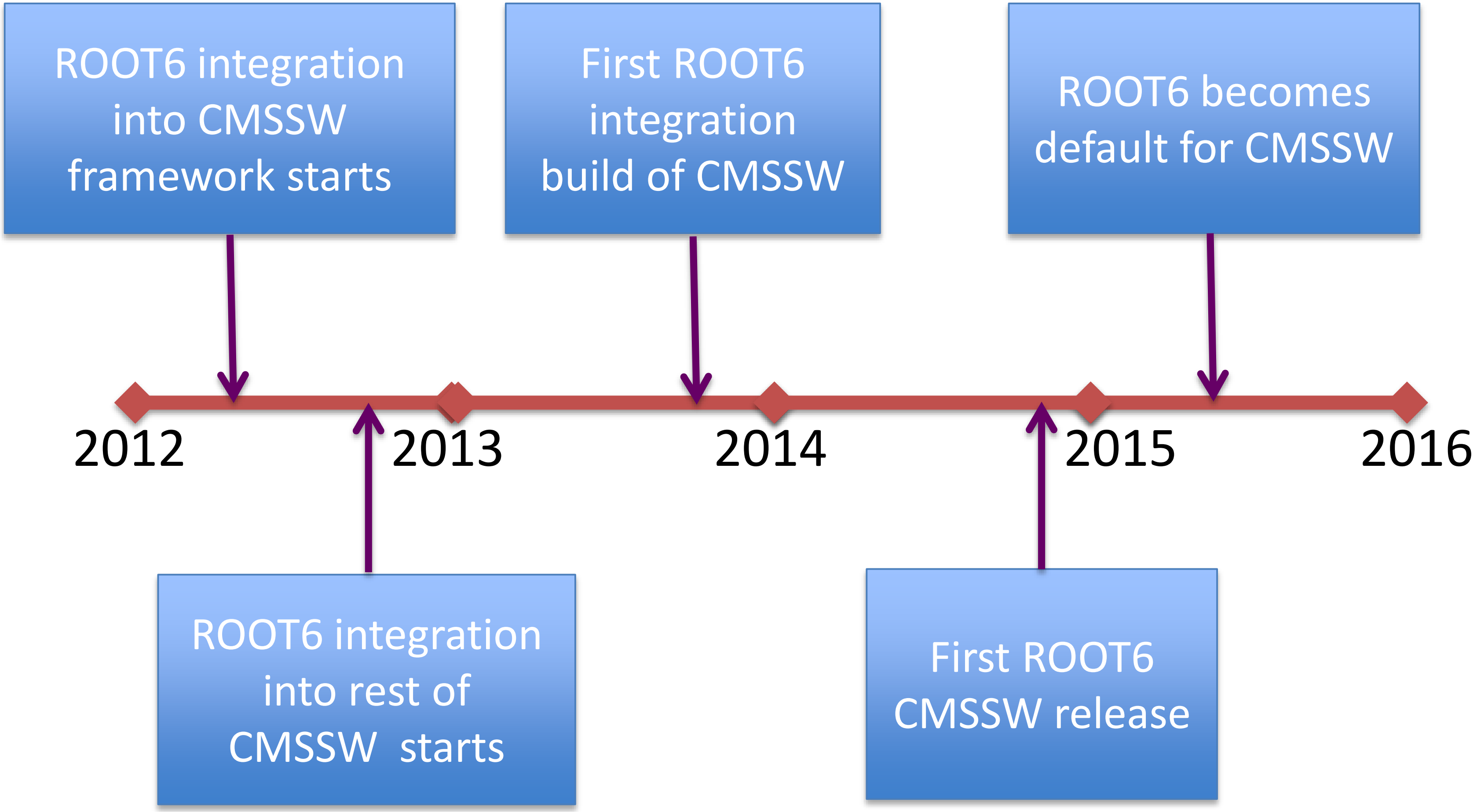
LLNL

September 15, 2015

CMS and ROOT

- CMS relies on ROOT from start to finish
  - ROOT persistency to store simple and complex objects for archiving and analysis
  - ROOT histograming capability for quality assurance and data certification plots
  - User analysis (tuples, fitting, plotting)
- Like other externally developed packages, we build ROOT ourselves (still using configure ☹) to have a flexible and consistent tool set
- CMS made the transition to ROOT6 during the long shutdown of LHC
  - We maintain a mirror of the ROOT GitHub so we can retain the flexibility to follow (or not) the latest changes in the ROOT patches branches and to fix bugs we discover quickly.

Evolution of ROOT6 in CMS


ROOT6 integration into CMSSW framework starts
First ROOT6 integration build of CMSSW
ROOT6 becomes default for CMSSW
2012
2013
2014
2015
2016
ROOT6 integration into rest of CMSSW starts
First ROOT6 CMSSW release

- Moving CMS to ROOT6 took a considerable effort on the part of developers (and now users)

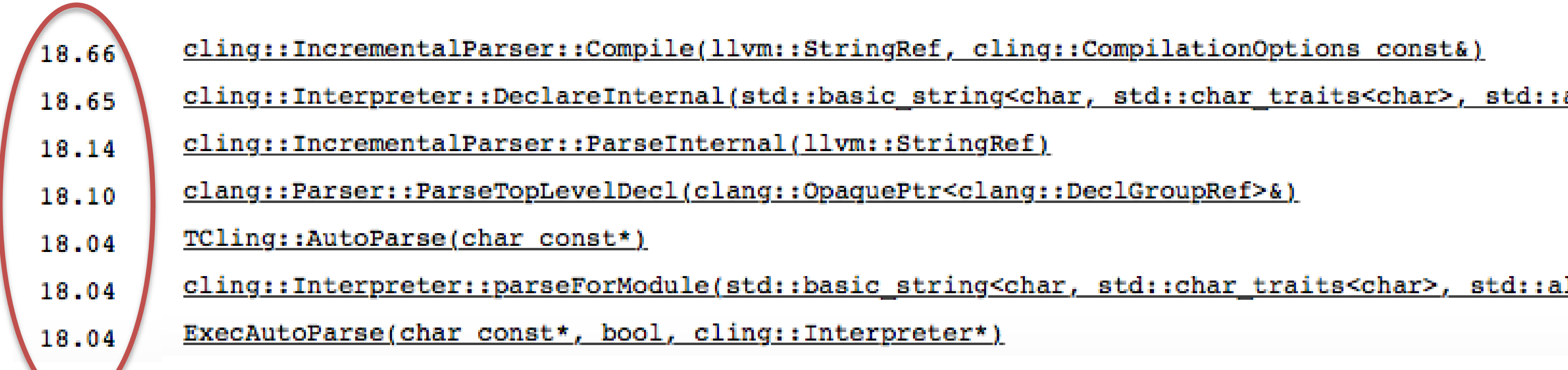
Where are we now?

- 2015 release of CMSSW:
  - Using ROOT 6.02.06+patches
  - We will move soon to tip of 6.02 branch now that problem blocking us for nearly 2 months is fixed
- Development release cycle
  - Two versions
    - Tip of 6.02 branch (+patches)
    - Tip of 6.04 branch (+patches)
  - Given the current status of our integration tests, we expect to use ROOT 6.04 when this release cycle becomes production
- We are trying to stay up to date!

Some issues we encountered on the way to ROOT6.02….

Increased memory footprint from header parsing

- Increased memory from header parsing was a big part of the work. Fixed both by ROOT changes and by CMSSW changes (to avoid the most troublesome syntaxes).
- Fragile situation: on the CMSSW side, nothing prevents users from re-introducing a “bad” syntax
- We still hope to do better, as header parseing a big part of our RSS



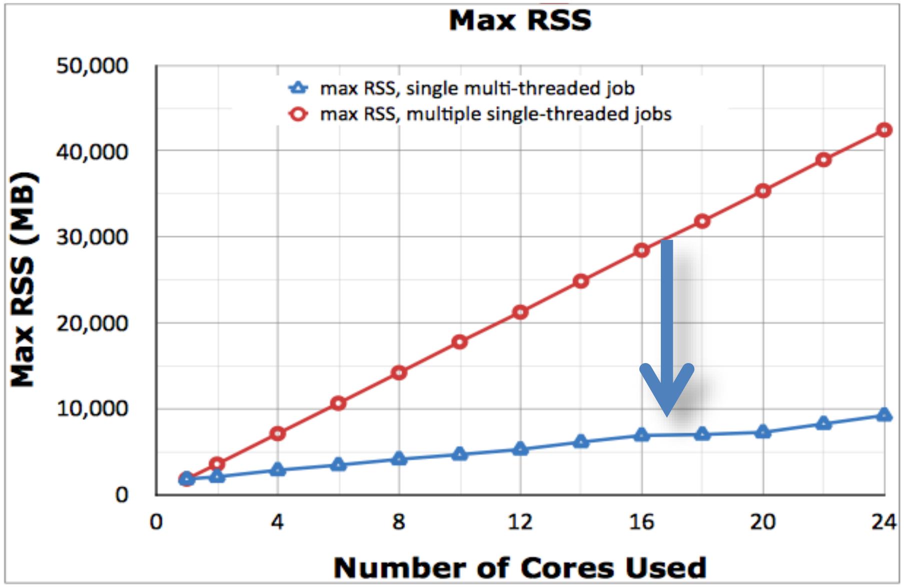
```
18.66   cling::IncrementalParser::Compile(llvm::StringRef, cling::CompilationOptions const&)
18.65   cling::Interpreter::DeclareInternal(std::basic_string<char, std::char_traits<char>, std::a
18.14   cling::IncrementalParser::ParseInternal(llvm::StringRef)
18.10   clang::Parser::ParseTopLevelDecl(clang::OpaquePtr<clang::DeclGroupRef>&)
18.04   TCling::AutoParse(char const*)
18.04   cling::Interpreter::parseForModule(std::basic_string<char, std::char_traits<char>, std::a
18.04   ExecAutoParse(char const*, bool, cling::Interpreter*)
```

RSS fraction of RECO application

- **Awaiting real PCMs**: Reducing the memory from parsing can bring a real improvement over our CMSSW+ROOT5

Importance of threading: The CMS Multi-threaded Framework now in production

- We have developed next-generation framework for CMSSW based on a multi-threading approach
  - This gives CMS a natural platform to investigate highly parallelizable algorithms
- Short term focus: This Framework allows us to process higher Run 2 trigger rates efficiently and to adapt to computing technology trends
- Current results:
  - Good scaling in CPU performance beyond where we need for Run 2
  - Substantial application memory savings in CMS reconstruction
- A development plan is in place to modify the FWK to scale up the threading performance to much higher levels over the next years.



ROOT and thread-friendliness

- To complete the transition of our production applications to the threaded CMSSW framework, we needed to improve the thread safety of a number of HEP products (“external” to CMSSW)
- This work started with ROOT5 as our transition to the threaded Framework started before ROOT6 was used by default in CMSSW.

- For ROOT, the largest issues affecting us were with I/O
  1. Read multiple TFiles on different threads
  2. Write multiple TFiles on different threads
  3. Calls to other ROOT functions on other threads should not interfere with I/O
- We did not set reading/writing one TFile on multiple threads as an initial need (or goal)

Implementation

- Approach: Use static analyzer, helgrind, simple test case
- Solutions applied
  - thread_local
  - Std::atomic<>
  - Mutex locks
- Most of this work is now part of ROOT
  - Exception: a lock in TROOT:GetListOfCleanups
  - The ROOT changes were a critical component for the efficient use of our threaded FWK

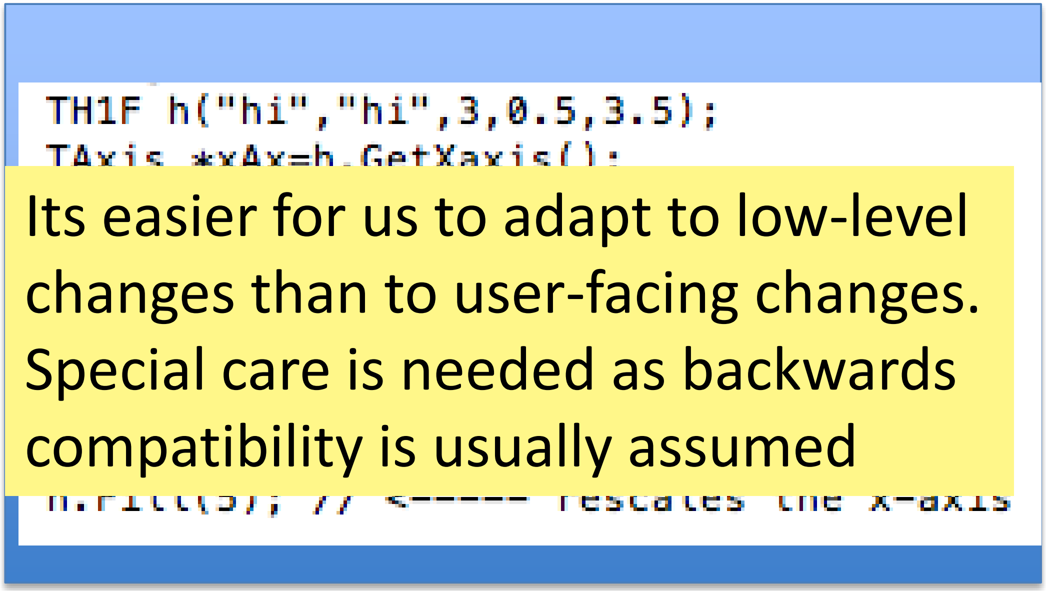
What about analysis users?

- We asked CMS analysts for comments on their experience moving from ROOT5 and ROOT6
  - Received very little feedback – can interpret this as a positive sign. No big troubles (despite some that were anticipated)
  - Some comments:
    - Went more smoothly that expected.
    - Compared to writing macros with ROOT5, ROOT6 forces you to become a better programmer (both good and bad)

- Likely we will get much more feedback about ROOT6 performance as the Run 2 datasets increase.

Changes to treatment of alpha-numeric histogram

- Of course overflows in a true alpha-numeric histogram make no sense.
- But that doesn't mean that users don't rely on the previous (ROOT5) behavior being maintained in ROOT6



```
TH1F h("hi","hi",3,0.5,3.5);
TAxis *xAx=h.GetXaxis();
h.Fill(5); // <----- rescales the x-axis
```

Its easier for us to adapt to low-level changes than to user-facing changes. Special care is needed as backwards compatibility is usually assumed

- Unfortunately it took looking at all of CMS Q/A histograms to discover the handful of cases where the new behavior makes a big difference
  - This is one example where we missed any advertisement of this important change.

Feedback on integration of bug fixes

- Bug fixes are often complex and may fix an issue for some but cause new issues for others
  - Impossible to catch all of these in a self contained ROOT test suite
- Suggestion: Can we work to more tightly couple experiment validation and propagation of bug fixes?
- CMS would benefit from having an opportunity to check/sign-off fixes in newest stable releases (eg, 6.04) before they go back to older ones (eg, 6.02)
  - The cost of problems getting all the way to stable releases is high and delays other fixes from getting to our users
  - We can volunteer to do this in a timely way (or not complain if we don’t manage to)

CMS data for Run 2: MiniAOD

- The “MiniAOD” has been created to increase agility and flexibility of the CMS data format structure for Run 2

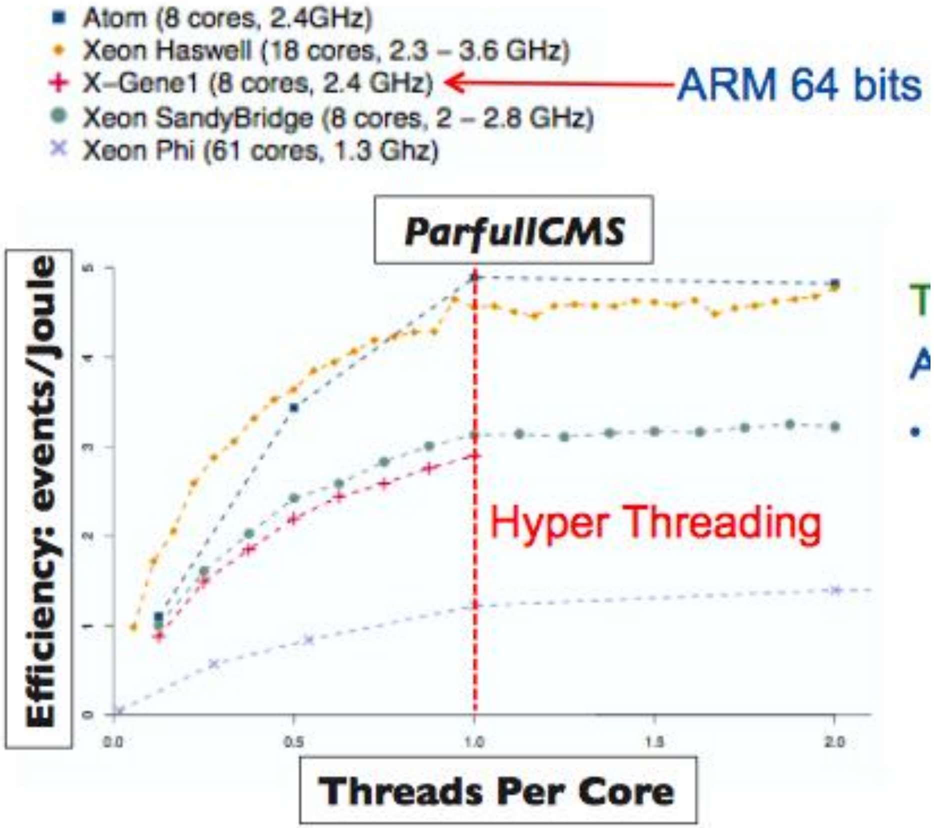
- We retain easy to use physics objects (e.g., complex classes for jets,electrons,muons,etc) .
- Instead the big size gains in the MiniAOD come from
  - Dropping objects used by a minority of analyses
  - Adding tighter physics requirements on all objects
  - Reduce precision where possible
- Goals were to keep only 20-40 kB per event while being useful for 80-90% of CMS analyses
  - While our format continues to evolve, these goals have been achieved for Run 2 startup

Optimizing miniAOD

- We investigated options for achieving better compression
  - Focused on storage size and readback time
  - **The results of our AOD optimization years ago still hold. No major improvements were found without changing data formats (given current set of hooks available)**
  - On the other hand we found ways to potentially improve our data formats
- One catch:
  - We merge together MiniAOD from smaller files.
    - Reconstruction time per event is too long to make a MiniAOD file that is several GB (goes away with threaded MC jobs)
  - Via fast cloning, the compression is much worse than if we had run one long job

Its important for ROOT to lead the way on new platforms (with input from users of course)

- Low-power architectures are one of the big R+D focus points in CMS
  - Collaborating on performance measurements with both reduced benchmarks and full CMSSW
  - Entire software stack up to anaysis job submission working in some cases (eg, AArch64)
- Being an underlying component of CMSSW, we need ROOT support for platforms
  - Especially interested in AArch64 and IBM POWER8



Example CHEP15 Results

Conclusion

- ROOT has proven an extremely successful toolkit for both CMS developers and users
- We find the weekly meeting with the ROOT team essential
  - We have ROOT6 for Run2 because of the long collaboration between CMS and ROOT developers
  - Should this become a more widely advertised meeting for “customers” of ROOT?