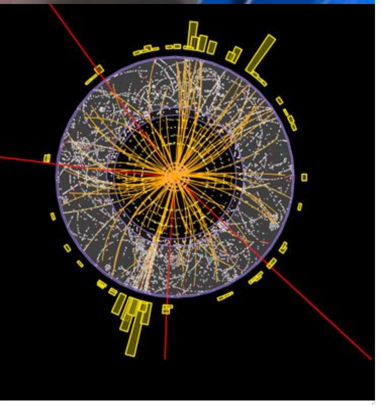
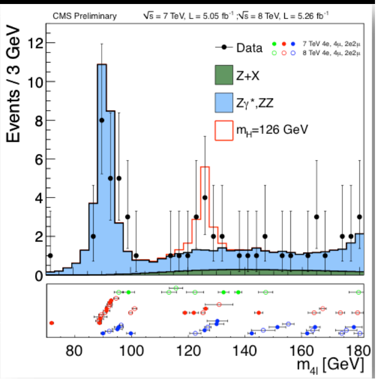
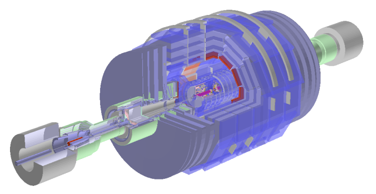





ROOT I/O Review and Future Plans

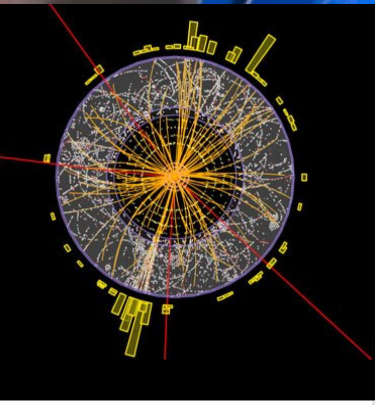
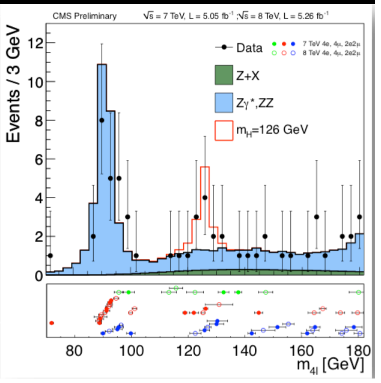
Philippe Canal
Fermilab

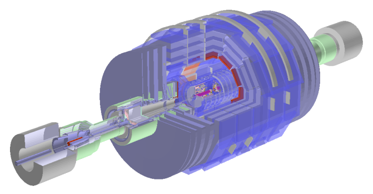
Overview

- What happened last year
  - Besides ***ROOT 6***

- Priorities
  - Multi-processing / Multi-threading
  - Performances improvements
  - Interface Simplification and Clarification
  - Interoperability
  - Statistics and feedback

- Challenges, outlook, discussions



Since Last Year

- New ***TClass*** state
- Checksum Updates
  - Still need bug fix and adding std
  - Fixed support for base class versions
- Added ***TTreeCache***::***LearnPrefill***
- ***TTreeCache*** enabled by env variable
- ***TTreeReader***
- Progress on Runtime gen. of ***CollectionProxy***
- New ***S3*** support class.
- Full support conversion to/from any ***STL*** *coll*
- Improved reading ***std::list<int>*** branch by 25%
- Add ***ROOT::Selection*** (for genreflex)

Rescheduled for after v6

- *Implement support for **I/O** for private classes*

- Last CheckSum updates
- *Type with template arguments that are enums*
- *Renaming rules fixes*

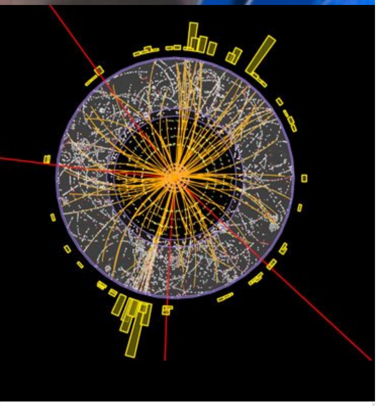
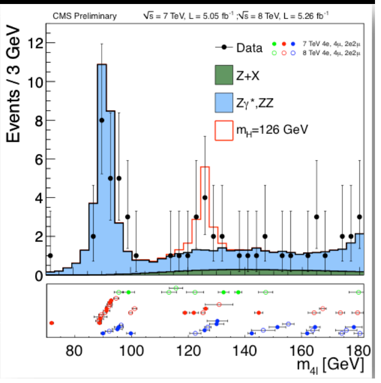
- ***TTreeCache***
  - Add missing global enable/disable API
  - Turn on by default
  - Install the new ***OptimizeBasket*** proposals

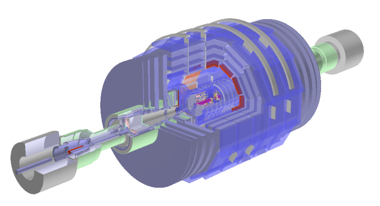
Here comes cling

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

- Will allow:
  - ***I/O*** for 'interpreted' classes
  - Runtime generation of ***CollectionProxy***
    - Dictionary ***no longer*** needed for collections! ***[Summer Student]***
  - Run-time compilation of ***I/O*** Customization rules
    - including those carried in ***ROOT*** file.
  - Derivation of 'interpreted' class from compiled class
    - In particular ***TObject***
  - 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
    - New, simpler TTree interface (***TTreeReader***) ***[Summer Contributor]***

Priorities



- Multi-processing / Multi-threading
- Performances improvements
  - Amdahl, File Format, Streaming, Vectorization
- Interface Simplification and Clarification
  - Leverage ***C++11*** for ease of use/documentation
- Interoperability
  - ***HDF5, R, Python, Blaze, numpy***, etc.
- Additional statistics and Feedback on I/O Perf.

Multi-Processing

- Import Chris' changes to ***v5.34*** and port to ***v6.02***
- Extend the ability to disable auto-add
  - Limited to ***TH**** so far
  - Remove use of ***I/O*** in ***TH*::Clone***
- Resolve parallelism limitations
  - As shown in the **CMS** condition database example

Multi-Processing

- ***Histogram*** and multi-threading
  - Need to start prototyping & testing asap
  - New interface to incrementally merge histograms from multiple threads

- Read/Write ***TTree*** branches in multiple user thread
  - Need to start prototyping/testing asap
  - Do we need new/simpler interface?
  - Need to design the limit and semantics
  - Extra complexity/cost to conserve basket clustering
  - Require ***TFile*** synchronization

Thread Safety

- ***Cling*** enables support for robust multi-thread ***I/O***
  - ***Cling*** has clear separation of database engine and execution engine allowing to lock them independently

- Chris' changes allow multi-threaded ***I/O*** as long as
  - Each ***TFile*** and ***TTree*** objects are accessed by only one thread (or the user code is explicitly locking the access to them)
  - Interpreter is *not* the top level entry point.
  - ***Cling*** will allow to remove the second limitation.

- More has to be done to optimize
  - Some object layout leads to poor performance and poor scalability
  - Reduce number of 'class/version/checksum' searches
    - To reduce the number of atomic and thread local uses

Parallel Merge Challenges

- Need official daemon/thread ***parallelMergeServer***
  - Could use ***Zero MQ*** as underlying transport.
- Need to efficiently deal with many histograms
  - Each of them still need to be merged at the end
- Lack of ordering of the output of the workers
  - No enforcing of luminosity block boundaries for example
  - Support for ordering increases worker/server coupling
  - Space reservation is challenging (variable entry)

- Need a new concept (an ***Entry Block***)
  - 'Set of entries that are semantically related'
  - To be used to gather those entries together 'automatically'
  - Need flexible/customizable marker
  - Is it really worth the extra complexity?

Parallel merge

- Fully tested and performing version requires
  - Parallel Merge Thread
  - Parallel Merge Daemon (authorization, auto-start, error handling)
  - ***Parallel Merge for Histogram*** (proper set of benchmarks, performance improvement, etc.)

- Benchmarks
  – Still to be designed
  – Based on existing example (some multithread) and new example based of the ***Event*** test.
  – Based on experiment uses cases.

Other Possible Parallel Processing

- Read/Write branches using *internals* thread/tasks
  - Need to partially back out memory optimization
  - Require ***TFile*** synchronization
- Offload work (compression) to separate thread
  - Need to work well with task based scheduler
- Thread safe version of ***TFile***
  - Not quite sure of semantic
  - Need to be cost-neutral for traditional uses

- Support for 'multiple' interpreter state
  - Decide on need / interface / use limitations
  - shared libraries (their PCMs) shared between interpreters?

Optimizations

- ***OptimizeBasket***
  - There are a couple of new algorithm proposals
  - Need to be tested on wide range of cases
- 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
- ***TTreeCache***
  - Start using it in ***TTreeCloner***.
  - Allow alternative algorithm
  - Tests, tests and tests
  - Switch on by default

File Format Upgrades

- Switch to little-endian
  - Enable additional run-time optimization
- Support ***C++11*** entities
- Improve meta-data
  - Reduce cost of repeated [deep] hierarchies
- Space saving changes.
  - Improve compression of branch of unsplit collections
  - Reduce overhead for deep hierarchy
- Time saving changes
  - Compress each entry individually to improve random access
- Write-once files
  - Support for direct write to ***Hadoop*** file System
- ***SQLite*** within ***ROOT*** file
  - Support database (for meta-data) co-located with data

I/O Customization Framework

- Bug fixes
  - Class renaming
  - Rules execution in complex ***TTree***
- Continue development
  - Extend documentation
  - Implement Write rules
  - Enable Just-In-Time compilation of rules
- Extend automatic conversions
  - ***Derived**** <-> ***Base****
  - From object to pointer

TTree

- ***TTree***
  - Bug fixes
  - Interface simplification
    - Promote ***TTreeReader***
    - Make ***SetAddress*** and ***SetBranchAddress*** ‘smarter’
  - Optimizations
  - Improve documentation
  - Improve statistics gathering ***[Atlas]***

- ***TTree*** Draw/Scan
  - Leverage cling

Vectorization

- In *TTree*
  - Eg. *TTree::Draw* execute formula on more than one element at a time
  - New interface allowing retrieval of multiple entries at once.
- In Streaming
  - Changing endianess would also merging and vectorization of even more streaming actions.

Brainstorming Future Interfaces

- Lesson learned in industry:
  - deprecation does not work (***Google, Apple, etc.***)
  - but interface versioning *does* work: ***Windows***, ***Javascript***, ***libc++***,…
- Challenge
  - reduce duplication by making old interfaces use new implementations
- One example of a possible solution

```
namespace ROOT {
  namespace v6 {
    class TFile { current interface };  // ROOT::v6::TFile
  }
  inline namespace v7 {
    class TFile { better interface };   // ROOT::TFile
  }
}
// If backward compatibility is needed/wanted
using namespace ROOT::v6;               //  TFile <==> ROOT::v6::TFile
```

Brainstorming Future Interfaces

- Some possible examples:
  - Type safe interfaces: no more casting
  - No globals, minimal static caching, const == thread safe
  - From:

```
OwnOrNot(TWhatever* arg);
```

  - To:

```
OwnOrNot(std::unique_ptr<TWhatever> arg);
OwnOrNot(&myWhatever); // Compilation error!
```

  - Conscious inlining e.g. for vectorization
  - Improve data structure for vectorization
  - Revisit/Redesign all functions in ***ROOT/Meta*** in view of ***cling***
  - Further simplify and reduce dictionaries

Interoperability

- ***HDF5, R, Python, Blaze, numpy***, etc.
  - These ecosystems has their strengths and weaknesses as well some similarities and significant differences with ***ROOT***
  - What can we learn from them?
  - How can ***ROOT [I/O]*** can be leveraged to enhance them?
  - How could our workflows benefit from using directly or indirectly any part of these ecosystems?
  - Who can help?

Additional statistics and Feedback

- Standardize and expand statistics gather in ***TFile*** impl.
- Give *qualitative feedback* on user data model and customization choices:
  - Evaluate the deserialization speed of a given object or a given TTree organization.
  - Visualizing ROOT file format layout
  - Correlate RIO API calls with block IO activity in the kernel (eg. SystemTap)

Challenges

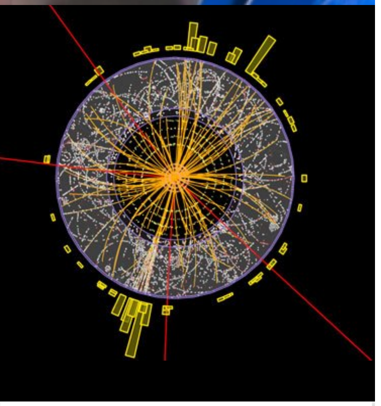
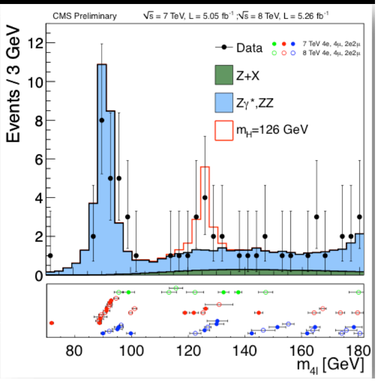
- Large program of work
  - 59 outstanding deficiencies
  - 63 improvements and new features
- Effort
  - My effort spread over ***ROOT I/O***, ***Cling*** and ***Geant/GPU***
    - Split 50/50 between ROOT and Geant
  - Extra effort required to make any real progress
    - Danilo will ramp up work on I/O
  - ***ROOT I/O*** Workshop helps coordinate direct effort from experiments
    - This comes and goes 'as needed' and competes with their own internal efforts.
  - Summer Students and other external contribution
    - ***MakeSelector*** *for* ***TTreeReader***
    - Runtime generation of ***CollectionProxy***

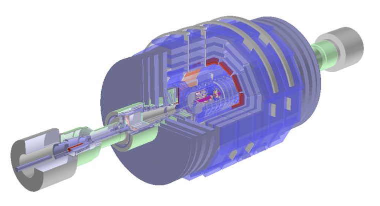
Conclusion

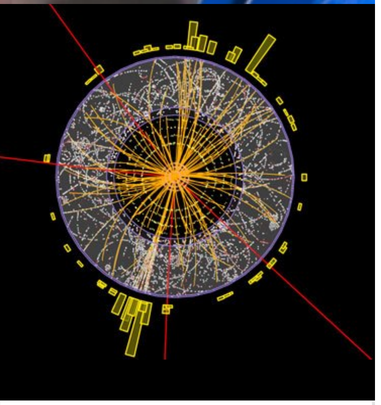
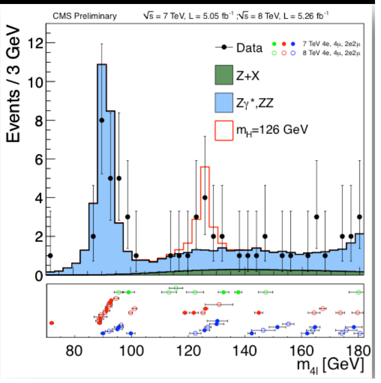
- Ambitious program to update ***ROOT*** for tomorrow's need
  - Update interfaces reflecting/solving usage problems
  - Use current ***C++***, code style and patterns
  - Allow more multi-processing uses
  - Reduce need for locks/atomics etc
  - **Improve performance and usability**
  - Extend use of vectorization
  - Extend reach of ***ROOT***

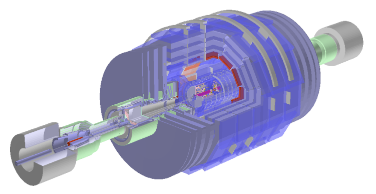
Priorities

- Multi-processing / Multi-threading
- Performances improvements
  - Amdahl, File Format, Streaming, Vectorization
- Interface Simplification and Clarification
  - Leverage C++11 for ease of use/documentation
- Interoperability
  - HDF5, R, Python, Blaze, numpy, etc.
- Additional statistics and Feedback on I/O Perf.





Backup slides

Backward Incompatibility

- ***rootcling*** no longer re-#defines the private and protected keywords to public.
  - ***ACLiC*** no longer breaks privacy!

- As a consequence I/O is ***currently*** not supported for private or protected classes
  - The major issue is access the constructor and destructor

Why one thread/schedule per TTree

- When reading TTree holds:
  - Static State:
    - List of branches, their types their data location on file.
  - Dynamic State:
    - Current entry number, ***TTreeCache*** buffer (per ***TTree***), User object ptr (one per (top level) branch), Decompressed basket (one per branch)
  - Separating both would decrease efficiency
- Advantages
  - Works now!
  - No need for locks or synchronization
  - Decoupling of the access patterns
- Disadvantages
  - Duplication of some data and some buffers.
    - However this is usually small compare to the dynamic state.
  - Duplication of work if access overlap

What's in a name …

- ***CINT*** and ***C++*** names are quite different
  - Implicit using namespace std statement in ***CINT***.
  - User typed spelling vs. 'real' spelling
    - ***vector<Int_t>*** vs ***std::vector<int, std::allocator<int> >***
    - User typed spelling not always available in Clang, especially for derived entities (data member of templates).
  - ***Clang*** does not propagate typedef to default template args
  - ***CINT*** template parsing bugs/shortcuts.
  - Opaque typedefs (***Double32_t***, ***std::string***, etc…)
- Almost sole source of 'risk' left for ***I/O***, handled by:
  - Adapt code to automatically discover the correct entity given the wrong (CINT) name.
    - Automatic matching of different spelling
  - Adapt checksum and schema checker to detect match due to variation in naming.
    - Added flexibility in checksum matching cross-checks

1 year outlook

End Of		Philippe Only		Philippe and extra effort
			3798	*The various TTree::Branch functions are very hard to figure out*
			3992	*TSelector::Process() on TChain*
			5078	Update fast-merging to leverage the TTreeCache
			4549	TRefArray does not clean fUIDs array in Streamer
	4489	Memory leak when TTree::BuildIndex is called multiple times	4550	TMessage doesn't honour kIsOwner bit when compression is used
July	4549	TRefArray does not clean fUIDs array in Streamer	4489	Memory leak when TTree::BuildIndex is called multiple times
			5070	Parallel merging daemon
			4044	*Documentation of compress parameter of TFile::Open()*
				Genreflex replacement
August		Genreflex replacement	**5080**	**Develop a comprehensive test plan for OptimizeBasket, LearnPrefill, TTreeCache.**
			5079	*Update TTreePerfStats to support multiple cache per file (Peter)*
	5079	*Update TTreePerfStats to support multiple cache per file (Peter)*	*5085*	*TTreeIndex supporting Long64_t (Peter)*
	5085	*TTreeIndex supporting Long64_t (Peter)*	*5084*	*TTreeFormula calculation in Long64_t (Peter)*
	5084	*TTreeFormula calculation in Long64_t (Peter)*	**5071**	**Parallel merge of histograms**
September	114	Fix issues in the renaming of classes in split branches where it is the base classes	5075	Write only once files (Hadoop)
			4496	*TTree doc*
			5073	Explore changing the on-file byte format to little endian!
October	*5078*	*Update fast-merging to leverage the TTreeCache*	*4441*	*hadd crashes when merging ntuples with different formats*
		Release Cut off		
			114	Fix issues in the renaming of classes in split branches where it is the base classes
November	5070	Parallel merging daemon	4839	TTree::Refresh and TTree::GetEntry causing crash
			113	*Fix issues when the target of the rule is an 'unsigned int' and when it is a struct*
			3709	*Crash when writing object with schema rule*
December	*5073*	*Explore changing the on-file byte format to little endian!*	*5157*	*Enhance Documentation for I/O customization rules*
			5077	*Find a way to avoid storing the byte count and version number for deep hierarchy!*
	113	Fix issues when the target of the rule is an 'unsigned int' and when it is a struct	*5082*	*Upgrade SetAddress and SetBranchAddress!*
January	3709	Crash when writing object with schema rule	*131*	*Optimize Baskets*
			3078	*Schema evolution rules not applied when loading from TTree*
			4049	**Base class schema problem when using member wise streaming**
			5156	*TTree::Draw and existing histogram*
February	*5075*	*Write only once files (Hadoop)*	*5183*	*TTree c'tor should take TDirectory*
	4550	*TMessage doesn't honour kIsOwner bit when compression is used*	*5066*	*multi-threaded file compression (tree writing)*
March	4833	TMessage::ReadObjectAny returns non-null pointer even in case of errors	*4441*	*hadd crashes when merging ntuples with different formats*
			4444	*ROOT crashes reading bad.root file (II)*
			4576	*Error reading older version ROOT tree file after upgrading ROOT*
April	**4049**	**Base class schema problem when using member wise streaming**	*119*	*Implement Write rules*
		Release Cut off		
	4839	*TTree::Refresh and TTree::GetEntry causing crash*	*5076*	*In TBasket compress each entry individually (for large basket)!*
May	5173	Issue with collection proxy and emulated class	*5159*	*Improve TTree documentation about SetMakeClass()*

1 year outlook

- … Not counting unexpected but essential new issues ….
- Current effort
  - 20ish (mostly small) issues addressed
- Additional effort
  - at least 40ish (many large) issues addressed

Multi Processing Bottleneck

- Number of cores and nodes increasing dramatically
- Managing very large number of files is both hard and somewhat wasteful.
- Usual solution is to merge the files.
- In addition, the number of disks is not increasing as fast
  - Hidden serialization, for example when using whole node allocation and fork on write.

Priorities

- Multi-processing / Multi-threading
- Performances improvements
  - OptimizeBasket
  - Endianess of buffer
  - “fast path” deserialization
  - Cost of repeated [deep] hierarchies
  - Write I/O customization Rules
- Interface Simplification and Clarification
  - SetBranchAddress, TTree::Draw, etc.
  - Leverage C++11 for ease of use/documentation.
- Interoperability
  - HDF5, R, Python, Blaze, numpy, etc.
- Additional statistics and Feedback
  - tool to evaluate the deserialization speed of a given object on a scale of one to ten using a few heuristics (similar in spirit to how lint will evaluate C source code quality).
  - tools for visualizing ROOT file format layout
  - module for SystemTap which will allow us to log and correlate RIO API calls with block IO activity in the kernel.