Developments in ROOT I/O and Trees

BRUN, René (CERN),
CANAL, Philippe (FERMILAB),
FRANK, Markus (CERN),
KRESHUK, Anna (CERN),
LINEV, Sergey (GSI),
RUSSO, Paul (FERMILAB),
RADEMAKERS, Fons (CERN)

ROOT I/O History

- Version 2.25 and older
  - Only hand coded and generated streamer function, Schema evolution done by hand
  - I/O requires : ClassDef, ClassImp and CINT Dictionary

- Version 2.26
  - **Automatic schema evolution**
  - **Use TStreamerInfo (with info from dictionary) to drive a general I/O routine.**

- Version 3.03/05
  - **Lift need for** ClassDef and ClassImp for classes not inheriting from TObject
  - **Any non TObject class** can be saved inside a TTree or as part of a TObject-class

- Version 4.00/08
  - Automatic versioning of 'Foreign' classes
  - Non TObject classes can be saved directly in TDirectory

- Version 4.04/02
  - Large TTrees, TRef autoload
  - TTree interface improvements, Double32 enhancements

- Version 5.08/00
  - Fast TTree merging, Indexing of TChains, **Complete STL support.**
- Version 5.12/00
  - Prefetching, TRef autoderefencing

- Version 5.16/00
  - Improved modularization (libRio)

Outline

- General I/O
  - *Major Enhancements*
  - libRIO
  - STL and Double32_t
  - File Utilities
  - Asynchronous Open
  - Consolidations
  - ROOT and SQL
- Trees
  - Autoderefencing
  - Fast Merging
  - Indexing of TChains
  - TTree Interface enhancements
  - New Features

Major Enhancements

- Improved Modularity
  - *"Booting ROOT with BOOT"* René Brun
  - libTree and libRio no longer loaded by default.
- Improvement in support TSQLFile and TXMLFile
  - Added support of ODBC
  - postgres support of new functionality upcoming
- TEntryList
  - A new class to support large and scalable event lists.
- Prefetching
  - *"Efficient Access to Remote Data in High Energy Physics."* Leandro Franco
- XROOTD
  - *"Data access performance through parallelization and vectored access. Some results."* Fabrizio Furano

libRIO

- New library containing all the ROOT classes to do basic Input/Output (ROOT 5.15/04 and above)
  - Includes TFile, TKey, TBufferFile, the Collection Proxies (for STL), etc.
  - TDirectory, TBuffer are now a pure abstract interface.
  - TDirectoryFile, TBufferFile are the concrete implementation
  - TFile derives from TDirectoryFile instead of TDirectory.
  - **These change may be backward incompatible:**
    - If you creates a TDirectory object, replace with TDirectoryFile
    - If you creates a TBuffer object, replace with TBufferFile
  - Dictionaries are **NO longer** dependent on any of the classes in libRIO

```
#if ROOT_VERSION_CODE >= ROOT_VERSION(5,15,0)
#include <TBufferFile.h>
#else
#include <TBuffer.h>
#endif
#if ROOT_VERSION_CODE >= ROOT_VERSION(5,15,0)
    TBufferFile b(TBuffer::kWrite,10000);
#else
    TBuffer b(TBuffer::kWrite,10000);
#endif
```

Streamer code update

- Change calls to TClass object into calls to TBuffer
- Removes direct dependency on TClass.

```
void Myclass::Streamer(TBuffer &R__b)
{
   // Stream an object of class Myclass.
   if (R__b.IsReading()) {
      Myclass::Class()->ReadBuffer(R__b,this);
   } else {
      Myclass::Class()->WriteBuffer(R__b,this);
   }
}
```

Old

New

```
void Myclass::Streamer(TBuffer &R__b)
{
   // Stream an object of class Myclass.
   if (R__b.IsReading()) {
      R__b.ReadClassBuffer(Myclass::Class(),this);
   } else {
      R__b.WriteClassBuffer(Myclass::Class(),this);
   }
}
```

Float, double and space…

- Math operations very often require double precision, but on saving single precision is sufficient…
- Data type: **Double32_t**

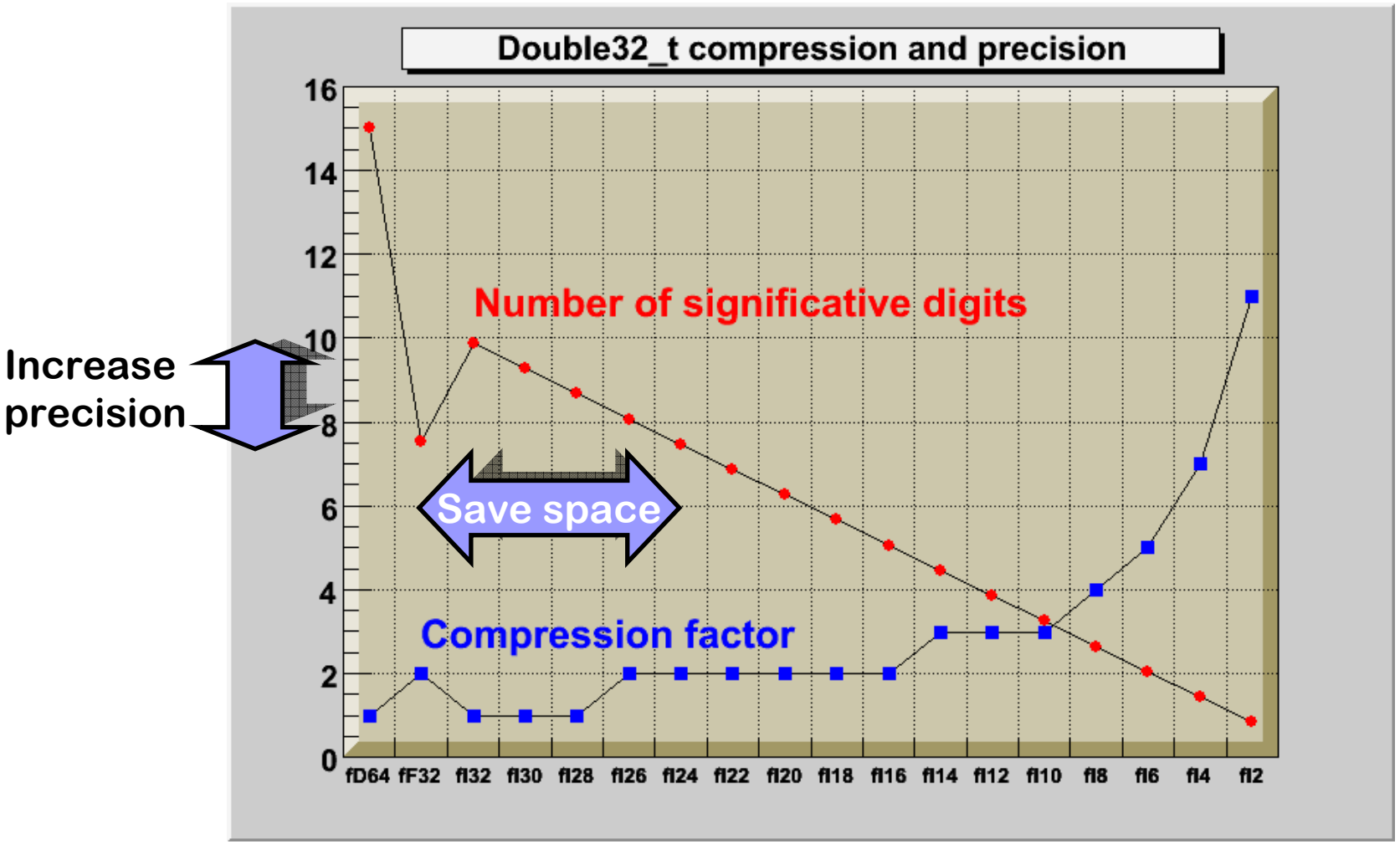
  In memory: double
  On disk: float or integer

- Usage (see tutorials/io/double32.C):

```
Double32_t m_data; //[min,max<,nbits>]
```

  - No nbits,min,max
    - saved as float
  - min, max
    - saved as int 32 bits precision explicit values or expressions of values known to Cint (e.g. “pi”)
  - nbits present
    - saved as int with nbit precision higher precision than float for same persistent space

Float, double and space… (2)



STL containers and Double32_t

- Support for Double32_t extended to the case where it is a template parameter.
  - Allow storing of the content of vector<Double32_t> as float instead of double (and any other STL containers).
  - Supported only for data members and when going via the "*string based*" interfaced.
    - Compilers and C++ RTTI can not distinguish between a mytemp<double> and a mytemp<Double32_t>.
    - Restriction could be lifted with the new C++ feature 'opaque typedef'
    - Dictionary for mytemp<double> and mytemp<Double32_t> must be in two different dictionary files.

```
Event* eventptr; std::vector<Double32_t> *myvect;
tree->Branch("event",&eventptr);
tree->Branch("myvect","vector<Double32_t>",&myvect);
```

- Support schema evolution from a container of double to the ***same*** container of Double32_t and vice et versa.

Remote File Utilities

- New static function TFile::Cp()
  - Allows ***any*** files (including non-ROOT files) to be copied via any of the many ROOT remote file access plugins.
- New Class TFileMerger
  - Similar to *hadd*
  - Easy copying and merging of two or more files using the many TFile plugins (i.e. it can copy from **Castor** to **dCache**, or from **xrootd** to **Chirp**, etc.).

```
TFileMerger m;
m->AddFile("url1");
m->AddFile("url2")
m->Merge();
```

  - The AddFile() and Merge() use the Cp() to copy the file locally before making the merge, and if the output file is remote the merged file will be copied back to the remote destination.

Remote File Utilities

- "CACHEREAD" option for TFile::Open()
  - First use TFile::Cp() to copy the file locally to a cache directory
  - Open the local cache file.
  - If the remote file already exists in the cache this file will be used directly, unless the remote file has changed.

```
root [] TFile::SetCacheFileDir("/tmp/userid");
root [] TFile *f = TFile::Open("http://root.cern.ch/files/aleph.root",
"CACHEREAD");
[TFile::Cp] Total 0.11 MB       |====================| 100.00 % [8.8 MB/s]
Info in : using local cache copy of http://root.cern.ch/files/aleph.root
[/tmp/userid/files/aleph.root]
root [] f->GetName();
(const char* 0x41dd2d0)"/tmp/userid/files/aleph.root"
root [] TFile::ShrinkCacheFileDir();
```

- New interface TFileStager defining the interface to a generic stager.

```
stg = TFileStager::Open("root://lxb6046.cern.ch")
stg->Stage("/alice/sim/2006/pp_minbias/121/168/root_archive.zip")
stg->IsStaged("/alice/sim/2006/pp_minbias/121/168/root_archive.zip")
```

Asynchronous Open

- TFile::AsyncOpen never blocks
  - returns an opaque handle (a TFileOpenHandle).
  - Also support string base lookup.
  - Active only for **xrootd** connection.

```
TFile::AsyncOpen(fname);

// Do something else while waiting for open readiness
EAsyncOpenStatus aos = 0;
while ((aos = TFile::GetAsyncOpenStatus(fname)) == TFile::kAOSInProgress) {
    // Do something else
    ...
}
// Attach to the file if ready ... or open it if the asynchronous
// open functionality is not supported
if (aos == TFile::kAOSSuccess || aos == TFile::kAOSNotAsync) {
    // Attach to the file
    f = TFile::Open(fname);
}
```

Consolidations

- Improvement in *hadd*
  - Compression level selections
  - Option to copy only histogram (and no TTree).
  - Use the new fast merge by default
- Thread safety tweaks
  - Reduced reliance on gFile/gDirectory in the ROOT I/O inner code so that only the first level routine (directly called by user code) access gFile and gDirectory.
  - We enhanced the STL container streaming code to make it thread-safe.

Consolidations

- Extended support for operator new in the dictionaries
- Implemented a proper 'destructor' for 'emulated objects'.
  - This changes allow for proper allocation and deallocation of emulated objects in all cases.
- Enabled I/O for C-style array of polymorphic array

```
Int_t fN;
MyClass** fAry; //[fN]
fAry = new MyClass*[fN];
fAry[0] = new MyClass;
fAry[1] = new DerivedFromMyClass;
```

- Enabled I/O for C-style array of strings.
- Add support for TBuffer's operator<< and operator>> from the CINT command line.

ROOT and SQL

- TSQLStatement
  - Related SQL prepared statements
  - Works with native data types: integer, double, date/time, string, null
  - Introduces binary data support (BLOBs)
  - Useful not only for SELECT, but also for INSERT queries
  - Implementations for:
    - *MySQL, Oracle, PostgreSQL, SapDB*
  - Significant improvement in performance, especially for bulk operations, especially for Oracle (factor of 25 - 100)
- Added support for *ODBC*
- TFileSQL
  - Allow access to table via the well known TFile interface
- Supports ***both*** classes with and without custom streamer.

Autoderefencing

- TRef and TRefArray are now auto-dereferenced when used in TTree::Draw
  - Requires to enable TRef autoloading (by calling TTree::BranchRef)
  - For collections of references either a specific element of the collections may be specified or the entire collection may be scanned. (example 2.)
- Same framework can be used for any ***Reference*** classes (eg. POOL Ref)

- The TTreeFormula operator @ applied to a reference object allows to access internals of the reference object itself (example 3.)
- The dereference mechanism even works across distributed files (if supported by the reference type) (example 4.)
- Caveat for TRefArray and TRef:
  - To know the underlying type, the first entry of the TTree is read.
- Special Thanks To Markus Frank
- Special Thanks to DZero for testing the limits of the TRef mechanism

```
1: T->Scan("fLastTrack.GetPx()");
2: T->Scan("fMuons.GetPx():fMuons[0].GetPx()”);
3: T->Scan("fLastTrack.GetUniqueID():fLastTrack@.GetUniqueID()&0xFFFF");
4: T->Scan("fWebHistogram->GetRMS()");
```

Autoderefencing (2)

- New Abstract interface TVirtualRefProxy
  - Generic interface to return referenced objects and their types.
  - Support both single references and collection of references.
- Concrete implementation must be attached to the corresponding TClass.

```
TClass::GetClass("TRef")->AdoptReferenceProxy(new TRefProxy());
```

```
void* TRefProxy::GetObject(TFormLeafInfoReference* info, void* data, int)
{
   if ( data )  {
      TRef*      ref    = (TRef*)((char*)data + info->GetOffset());
      // Dereference TRef and return pointer to object
      void* obj = ref->GetObject();
      if ( obj )  { return obj; }
      ... else handle error or implement failover ....
    }
}
```

Fast Merge of TTrees.

- New option, "fast“ for CloneTree and Merge.
  - No unzipping, no un-streaming.
  - Direct copy of the raw bytes from one file to the other.
  - Much higher performance.
  - Only available if the complete TTree is being copied.
  - Can also sort the baskets by branch or by sequential read order

```
myChain->CloneTree(-1,"fast");
myChain->Merge(filename,"fast");
```

New TTree Features

- Importing ASCII data
  - Long64_t TTree::ReadFile(filename,description)
  - 'description' gives column layout following 'leaflist' format

```
TTree *T = new TTree("ntuple","data from ascii file");
Long64_t nlines = T->ReadFile("basic.dat","x:y:z");
```

- TTree::GetEntries
  - Number of entries passing the selection

```
Long64_t nevents = T->GetEntries("fPx>2.5");
```

TTree Drawing

- TString and std::string can now be plotted directly.

```
tree->Draw("mybr.mystring");
tree->Draw("mybr.mystring.c_str()");
tree->Draw("mybr.mytstring.Data()");
```

- Object plotting
  - If a class has a method named either AsDouble or AsString (AsDouble has priority), it will be used for plotting.
  - For example with a TTimeStamp object:

```
tree->Draw("myTimeStamp");
tree->Draw("myTimeStamp.AsDouble()");
```

- Allow more formatting options for TTree::Scan.

```
tree->Scan("val:flag:flag:c:cstr", "", "col=::#x:c:");

************************************************************************
*    Row   *       val *      flag *      flag *         c *      cstr *
************************************************************************
*        0 *         0 *         1 *       0x1 *         a *       i00 *
*        1 *       1.1 *         2 *       0x2 *         b *       i01 *
```

TTree::Draw extensions

- TTree::Draw can execute scripts in a context where the name of the branches can be used as a C++ variable.

```
// File hsimple.C
double hsimple()
{
   return px;
};
```

```
// File track.C
double track()
{
  int ntrack = event->GetNTracks();
  if (ntrack>2) {
    return fTracks.fPy[2];
  }
  return 0;
};
```

```
tree->Draw("hsimple.C");
```

```
tree->Draw("track.C");
```

TTree::MakeProxy

- Enables `tree->Draw("hsimple.C");`
- Generates a skeleton analysis class inheriting from TSelector and using TBranchProxy.
  - TBranchProxy is the base class of a hierarchy implementing an indirect access to the content of the branches of a TTree.
- Main Features:
  - **on-demand** loading of branches
  - ability to use the 'branchname' as if it was a data member
  - protection against array out-of-bound
  - ability to use the branch data as an **object** (when the user code is available)
  - Gives access to all the functionality of TSelector
- Example in $ROOTSYS/tutorials:
  h1analysisProxy.cxx , h1analysProxy.h and h1analysisProxyCut.C

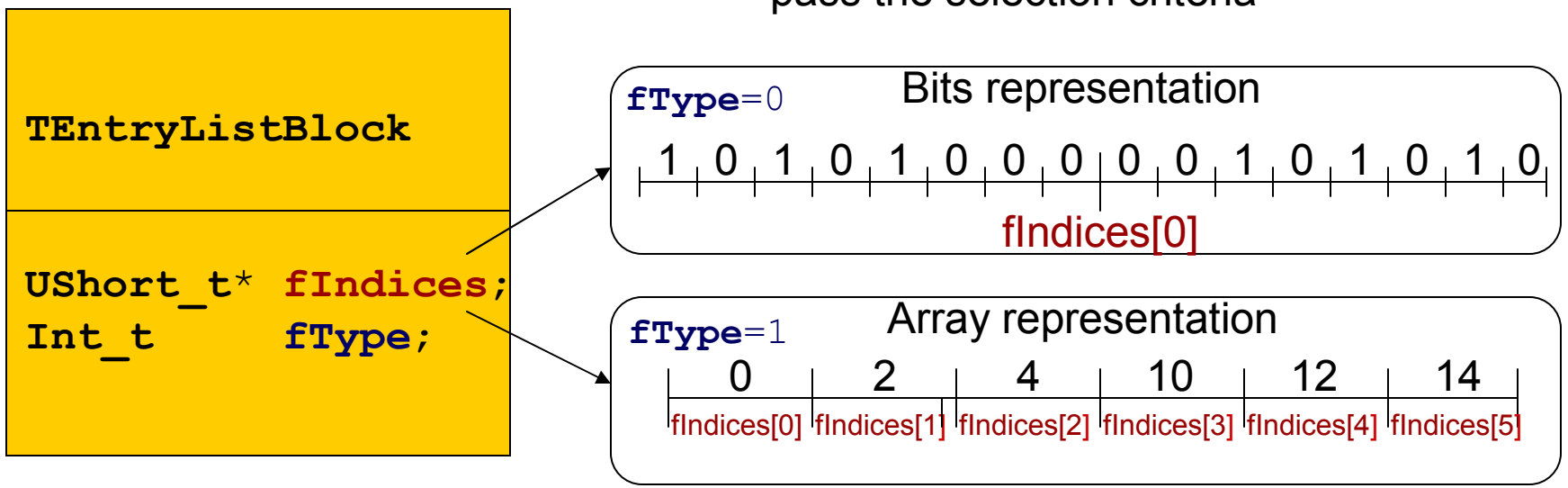
TEntryList

```
tree->Draw(">>elist", "x<0", "entrylist" );
…
tree->SetEntryList(elist);
```

- Goals:
  - Replace TEventList
    - TEventList is a simple list of the entries numbers
      - Scale linearly with the number of entries selected!
      - Not well suited for Proof
  - Scalable, Modular, Small, Only partially loaded in memory
- Strategy:
  - Use ‘block’ holding information on 64000 entries
  - Information stored ***either*** as a bit field or a regular array of entry number
- Features:
  - Can be stored/restored easily from (independent) files
  - Can be combined and split
    - To handle trees independently from their chain (This is essential for Proof)

TEntryList Storage Strategy

Suppose that this block stores information that entries
0, 2, 4, 10, 12, 14
pass the selection criteria



It makes sense to switch to the array representation when less than 1/16 of entries in the block pass the selection criteria

Upcoming Features

- Continue Consolidations ☺

- Splitting STL collection of pointers
- Schema Evolution
  - Provision for Data Model Evolution
  - To and From more combinations of containers (Double32 vs. double, ROOT containers).
- MakeProxy
  - Add support for CINT-interpretation
- TTree
  - Indexing using bitmap algorithm (TBitMapIndex) from LBL.
  - TTree::Draw performance

Conclusions

- Even after 12 years of ROOT:
- The I/O area is still improving
- There were quite a number of developments
  - Remote file performance
  - SQL Suport
  - Tree I/O from ASCII, tree indices
  - Auto dereferencing
  - Fast Merge
- There will be certainly some developments in the I/O area
- The "*classical*" stuff however is intended to be kept stable
- Main focus:
  Consolidation (*Thread Safety***)**, Data Model Evolution and more STL support.