Improvements in the I/O Area (*)

- General I/O related improvements
- Tree related issues
- Plans

ROOT 2005
Users Workshop
CERN-September 28,29,30

Workshop Topics

- Use of ROOT as a general framework
- Feedback from experiments
- Progress with objects persistency
- Merge with SEAL Reflex
- Progress with the new version of CINT
- Progress with the Python interface
- What is now with the Math libraries
- Distributed Data Analysis with PROOF
- Progress with GUIs and Graphics
- Progress with the new GL viewer
- Progress with the Geometry classes

http://root.cern.ch

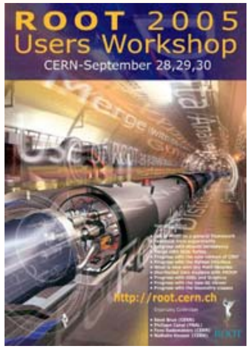
Organizing Committee

- René Brun (CERN)
- Philippe Canal (FNAL)
- Fons Rademakers (CERN)
- Nathalie Knoors (CERN)

(*) I present here, I did not develop it all myself
Hence: forgotten credits built-in…

I/O Improvements – Outline

- STL collections
- Data compression using reduced precision
- Alternatives to default constructors
- Mixed items

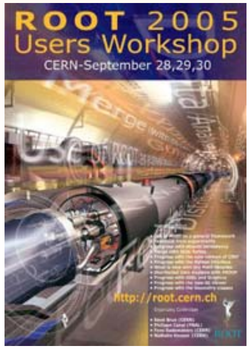


ROOT I/O: STL Collections

- ROOT now supports I/O of all STL containers
  - std::vector<T> std::list<T> std::set<T> std::deque<T> std::map<K,T> std::multimap<K,T>
  - And implicitly (through std::deque) std::queue<T> std::stack<T>

- Containers not in the C++ standard
  - If the dictionaries are translated from reflex...
  - hash_map<K, T>, hash_multimap<K,T>
    hash_set<T>, hash_multiset<T>
  - But be aware: these are **NOT portable**:

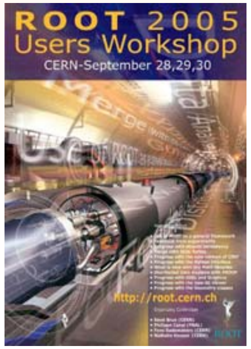
gcc:	namespace __gnu_cxx
VC++:	namespace stdext
Intel:	namespace std



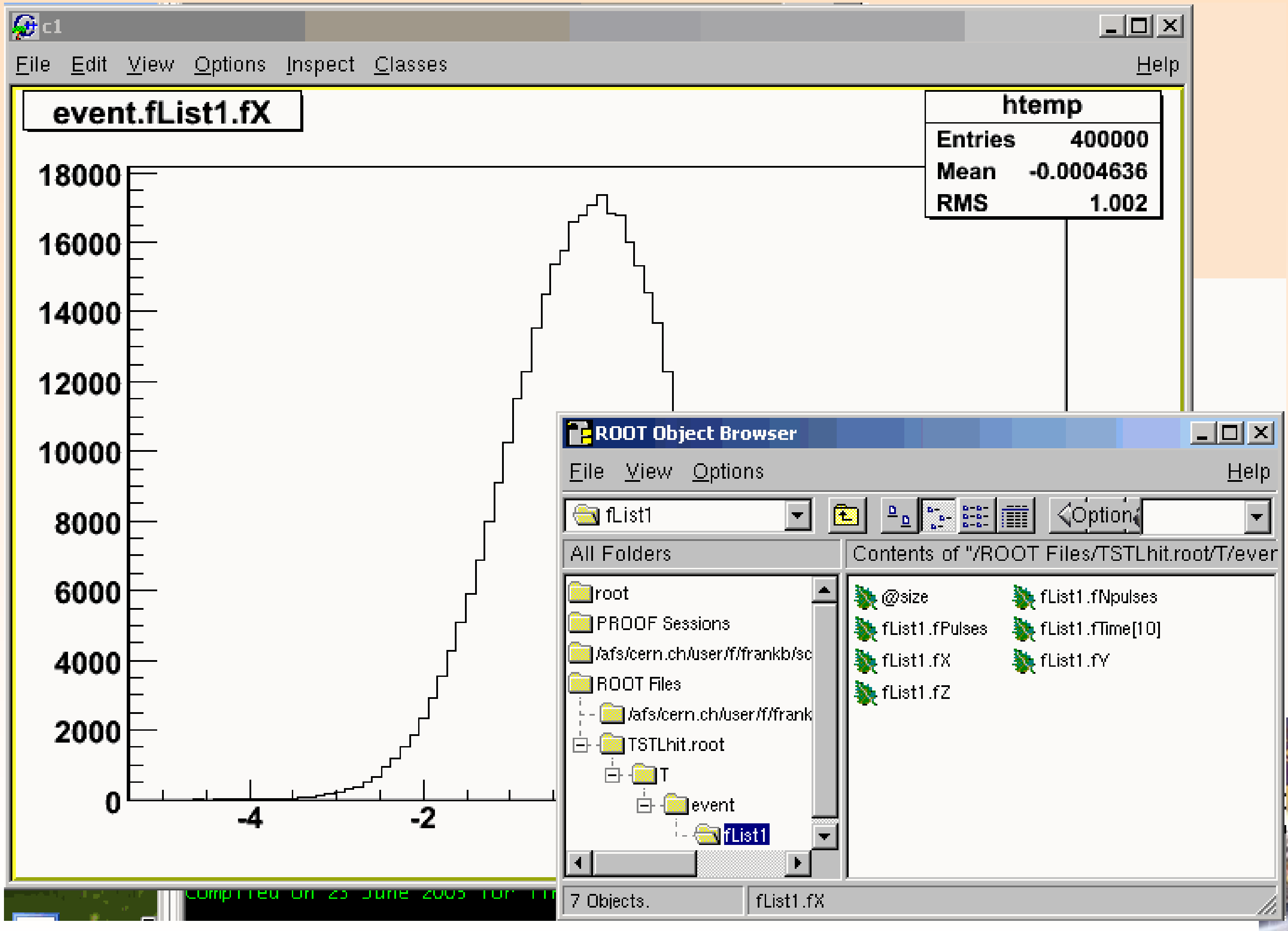
ROOT I/O: STL Collections (2)

- STL collections are saved in split mode
  - Objects are split (but: NOT if pointers)
  - Quick pre-selections on trees
  - Interactivity: Trees can be browsed
  - Save space (see $ROOTSYS/test/bench):

std::vector<THit>:	compression 5.38
std::vector<THit*>:	compression 3.37



ROOT I/O: STL Collections (3)



htemp	
Entries	400000
Mean	-0.0004636
RMS	1.002

Contents of "/ROOT Files/TSTLhit.root/T/ever

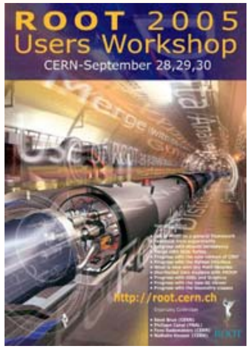
7 Objects.

fList1.fX

Compiled on 23 June 2005 for

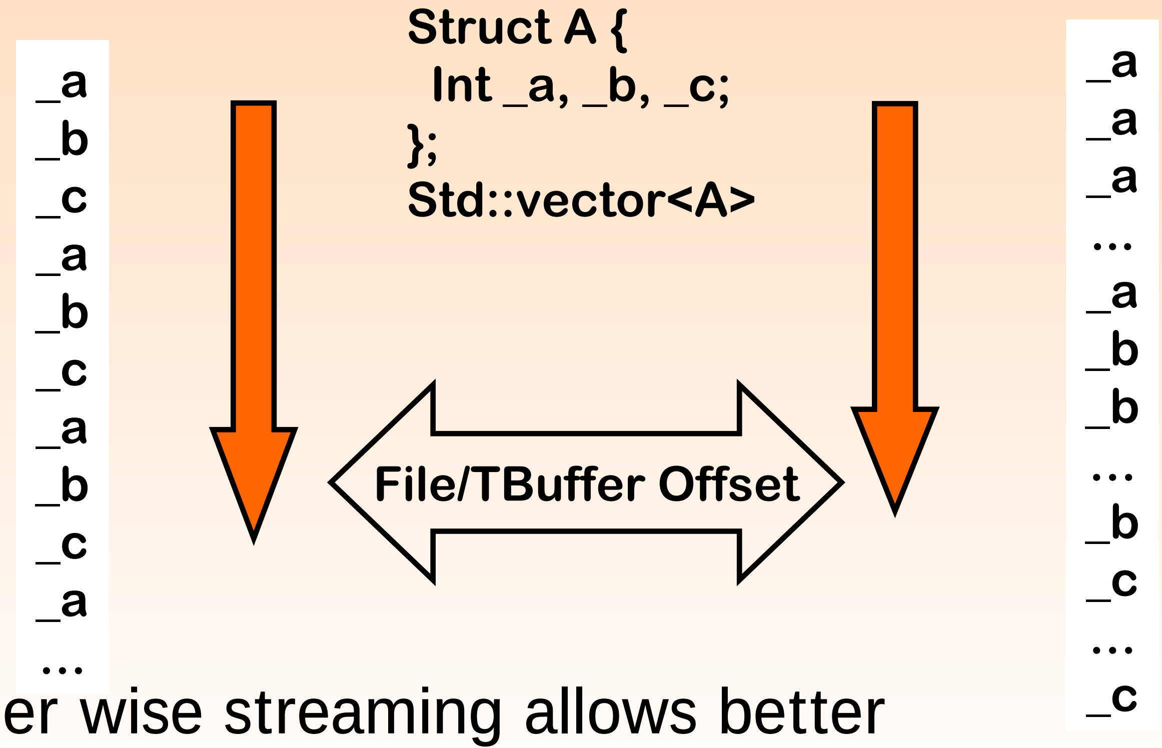
ROOT I/O: STL Collections (4)

- STL collections which can be split
  - Collections of objects ... not collections of pointers
- Can be saved either object wise
  - As ROOT always did it.
- Or member wise



ROOT I/O: STL Collections (5)

- **Streaming: Object- & member wise**

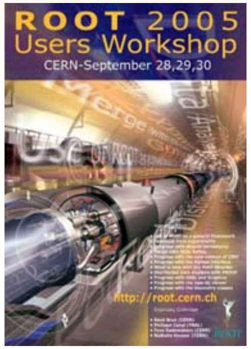


```
Struct A {
  Int _a, _b, _c;
};
Std::vector<A>
```

_a
_b
_c
_a
_b
_c
_a
_b
_c
_a
…

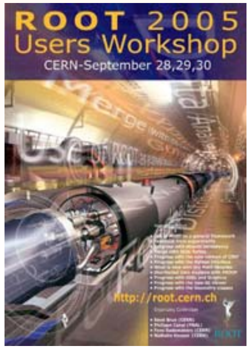
_a
_a
_a
…
_a
_b
_b
…
_b
_c
…
_c

- Member wise streaming allows better compression (zip works more efficient)
- **Bool_t TStreamerInfo::SetStreamMemberWise(Bool_t enable)**



ROOT I/O: STL Collections (6)

- Schema evolution of STL containers
  - As your classes change evolve ROOT can switch to new container types at reading time
TClonesArray	<->	std::vector<T>
TClonesArray	<->	std::list<T>
std::vector<T>	<->	std::list<T>
...		
  - Conversion between any non-associative Container



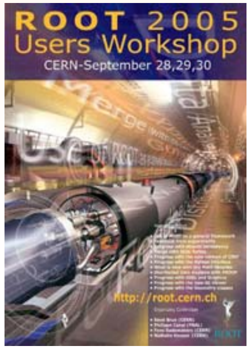
Float, double and space...[1]

- Math operations very often require double precision, but on saving single precision is sufficient...

- New data type: **Double32_t**
  In memory: double
  On disk: float or integer[2]

[1] **Implemented by R.Brun**

[2] **Inspired by O.Callot (LHCb)**

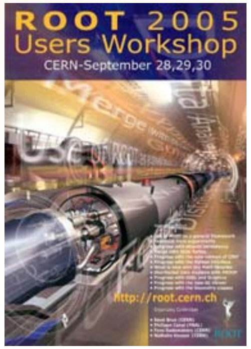


Float, double and space… (2)

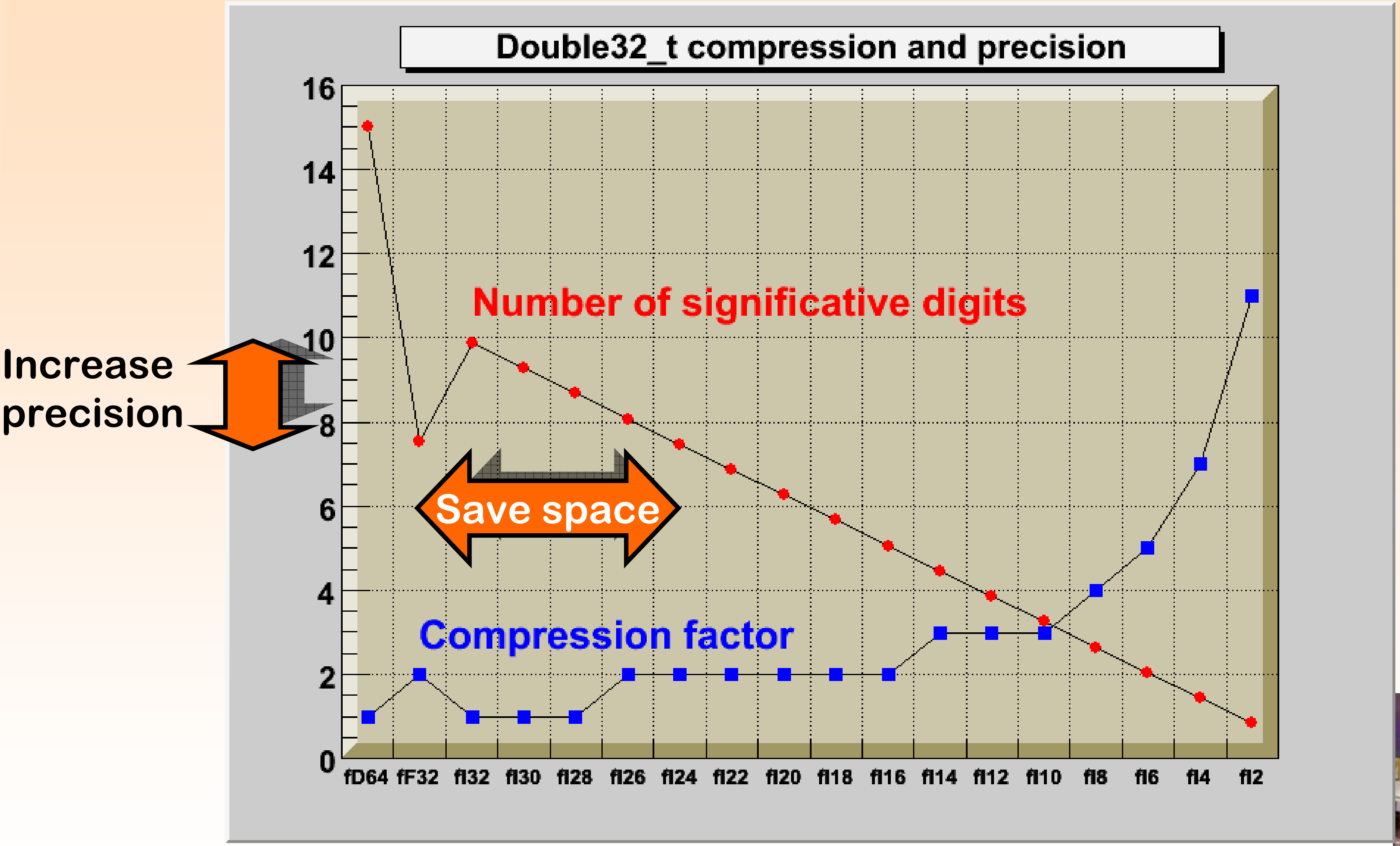
- Usage (see tutorials/double32.C):

```
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… (3)



Increase precision

Default Constructors

- ROOT requires a default constructor for reading
- Not all libraries provide such default constructors (e.g. Geant4)
- Alternative: I/O constructor customization

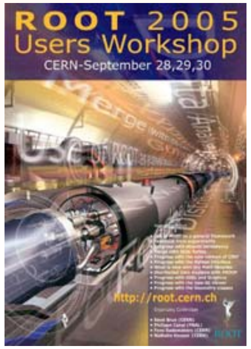
```
#pragma link C++ class MyClass;
#pragma link C++ ioctortype UserClass1;
#pragma link C++ ioctortype UserClass2;
```

Constructor search:

```
MyClass(UserClass1*);
MyClass(UserClass2*);
MyClass(TRootIOCtor*);
MyClass(); // Or constructor with all args defaulted.
```

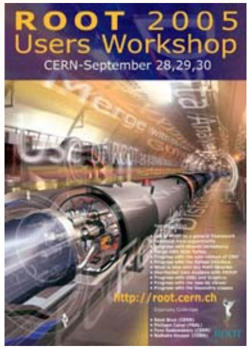
Bug Fix: bool Data Type

- Bool data type was handled as "unsigned char"
- However: on some architectures (MAC) the size of a bool is not 1 byte
- Needed proper handling to read/write "bool*"



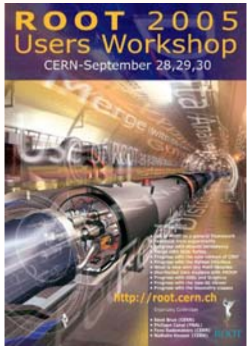
TTree extensions - Outline

- Large Trees (> 2 GB)
- Circular buffers
- Importing ASCII data
- Indices
- Binding of Objects to Trees



Large Trees

- Sequence of files:
  - myFile.root -> myFile_1.root -> myFile_2.root -> myFile_N.root
  - Define file size using the functions (Default 1.9 GByte):
    - `TTree::GetMaxTreeSize(), TTree::SetMaxTreeSize(Long64_t)`
  - Note: Maximum File size is no longer 2GB !
- User guide (Chapter 12 – pg.172)

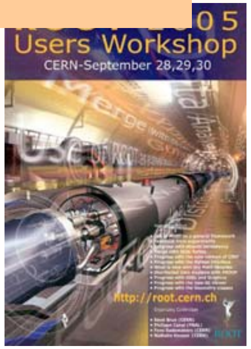


Circular TTree buffers

- For memory resident Trees
- Tree buffers wrap after specified number of entries
  - Currently for basic types
  - Extension for objects to come in the next release
- Monitoring

```
gROOT->cd(); //make sure that the Tree is memory resident
TTree *T = new TTree("T","test circular buffers");
. . .
T->SetCircular(20000);
for (i = 0; i < 65000; i++) { . . . }
```

- User guide (Chapter 12 – pg.172)

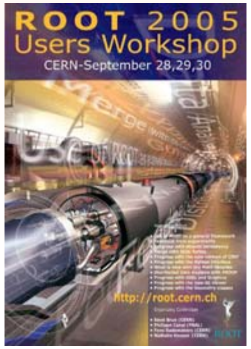


Importing ASCII data

- Long64_t TTree::ReadFile(fname,branchDesc)
  - Read formatted data from file <fname>
  - branchDesc gives column layout
    (Like for TTree::Branch(...leaflist...))

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

- User guide (Chapter 12 – pg.190)

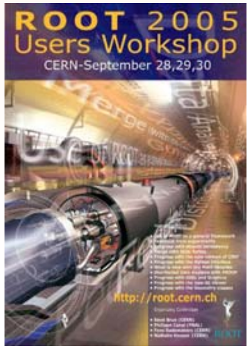


TTree indices

- Fast lookup fo entries
  - tree->BuildIndex(majorname, minorname);
  - Major/minorname are expressions using tree variables e.g. “Energy-3*E_miss”
- For TChains: Existing Tree indices can be reused

```
// to create an index using leaves Run and Event
tree.BuildIndex("Run","Event");
// to read entry corresponding to Run=1234 and Event=56789
tree.GetEntryWithIndex(1234,56789);
```

- User guide (Chapter 12 – pg.172)



Binding of Objects to Trees (1)

- TBranch::SetBranchAddress(object)
  - _was_ a very slow call
    usage was deprecated after initialization
  - Consequence: re-use of objects
    Splinter in the C++ purist’s eye
- Then after some investigation
  - Speed improvements by ~ factor 20

☺Purists no longer need to reuse objects
Objects can quickly bound for each Fill() call

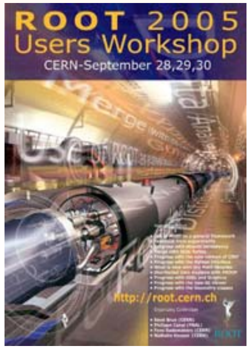
Binding of Objects to Trees (2)

- New overloaded call to TTree::Branch

```
template <class T>
TBranch *Branch(name, T **obj_address,…)
example:     MyObj* pObj = ….;
             myTree->Branch(“Branch”,&ptr);
```

- Better type safety
- Saves additional argument with the classname

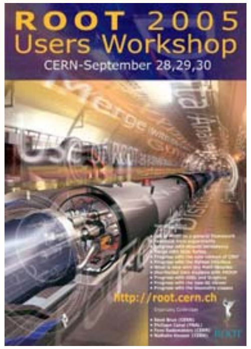
☺ No more typos of class names for templated classes



Ongoing: Object Reference Support

- ROOT and POOL support references to objects
  - ROOT: TRef
  - POOL: pool::Reference
- Need for automatic, implementation independent reference follow mechanism

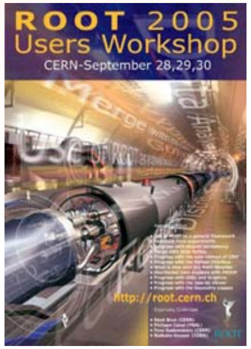
☺ TTree::Draw will automatically follow TRefs



Ongoing: Object References (TRef) [*]

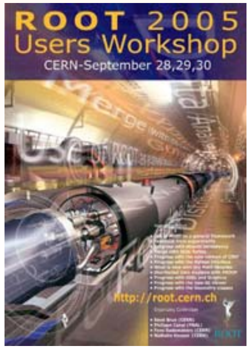
- TBranch* TTree::BranchRef()
- Creation of optional branch containing all information to find the branches of referenced objects.
- Enabling this branch at write time saves the additional info

[*] **courtesy of Rene Brun (CERN)**



Conclusions

- Event after 10 years of ROOT:
- The I/O area is still moving
- There were quite a number of developments
  - Full STL support
  - Data compression
  - Tree I/O from ASCII, tree indices



Conclusions (2)

- There will be certainly some developments in the I/O area
- The “classical” stuff however is intended to be kept stable
- Main focus:
  Generic Object Reference support
  - User defined reference objects supported by
  - User defined reference handlers (proxies)