Persistent Object References in ROOT I/O

Status & Proposal

LCG meeting

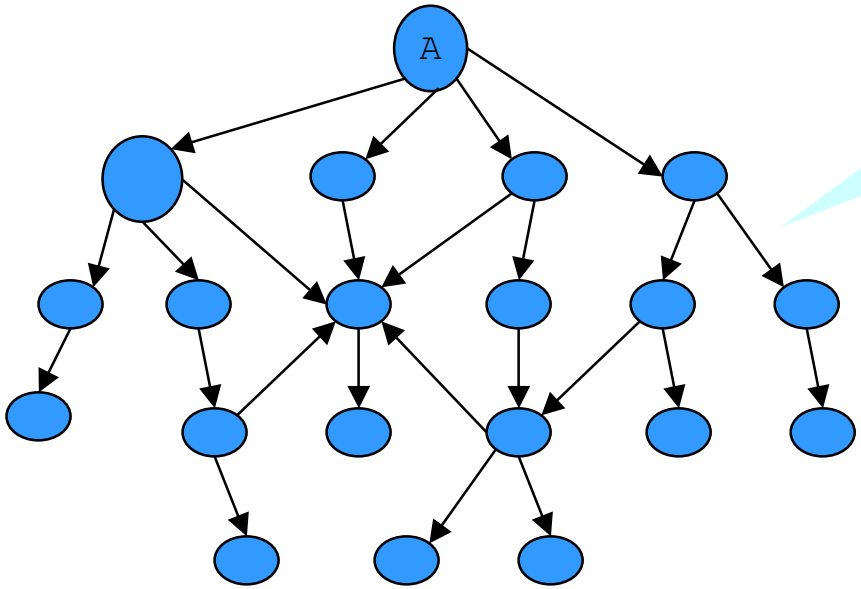
CERN- 5 June

René Brun

`ftp://root.cern.ch/root/longrefs.ppt`

Normal Streaming mode

References using C++ pointers



Only one copy of each object in the graph saved to buffer

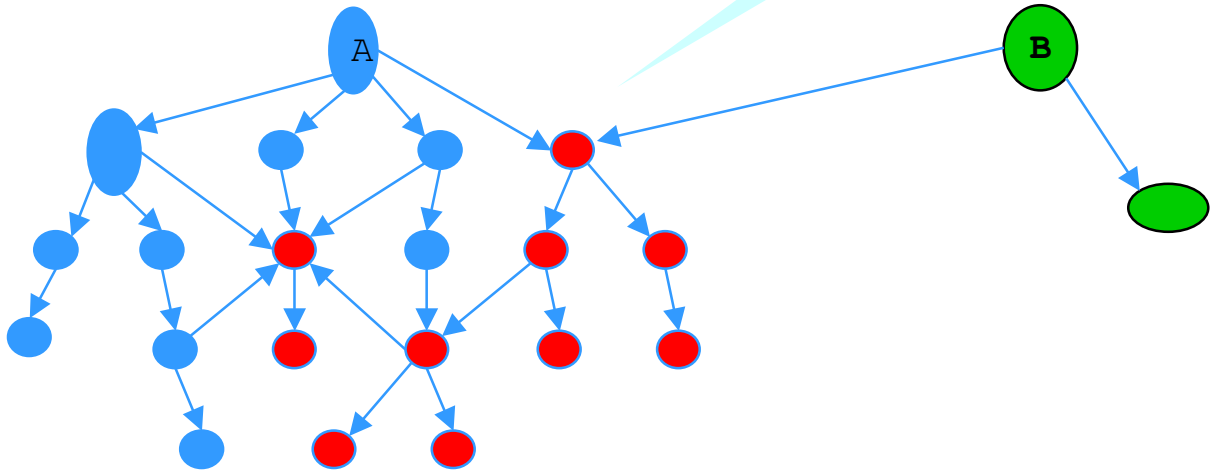
```
TBuffer b;
A.Streamer(b)
```

Normal Streaming mode

References using C++ pointers

```
TBuffer b1;
A.Streamer(b1)
TBuffer b2;
B.Streamer(b2)
```

Objects in red are in b1 and b2

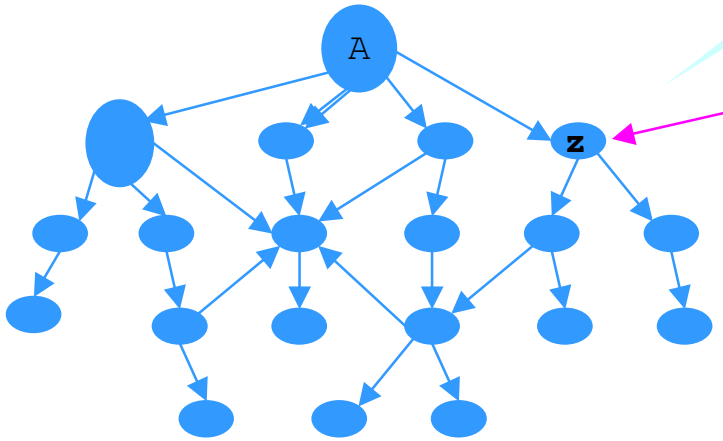


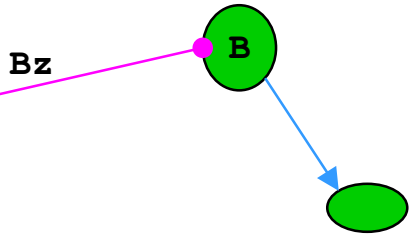
Normal Streaming mode

References using TRef pointers

```
TBuffer b1;
A.Streamer(b1)
TBuffer b2;
B.Streamer(b2)
```

Objects in blue are only in b1





C++ pointer

TRef

```
Set pointer to z with: TRef Bz = z;
Get pointer to z with: z = Bz.GetObject()
```

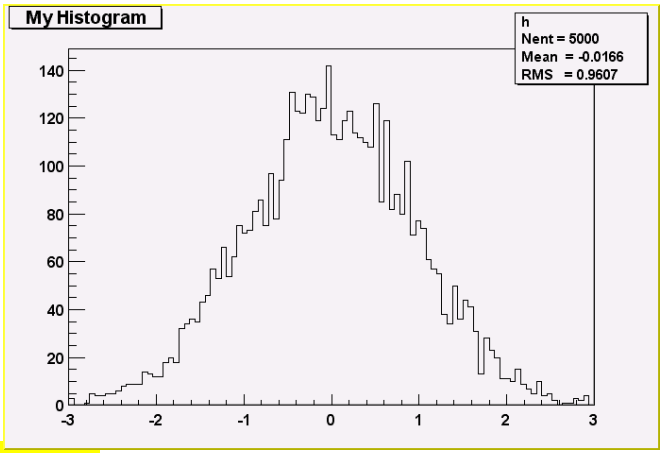
ROOT I/O : An Example

Program Writing

```
TFile f("example.root","new");
TH1F h("h","My histogram",100,-3,3);
h.FillRandom("gaus",5000);
h.Write();
```

Program Reading

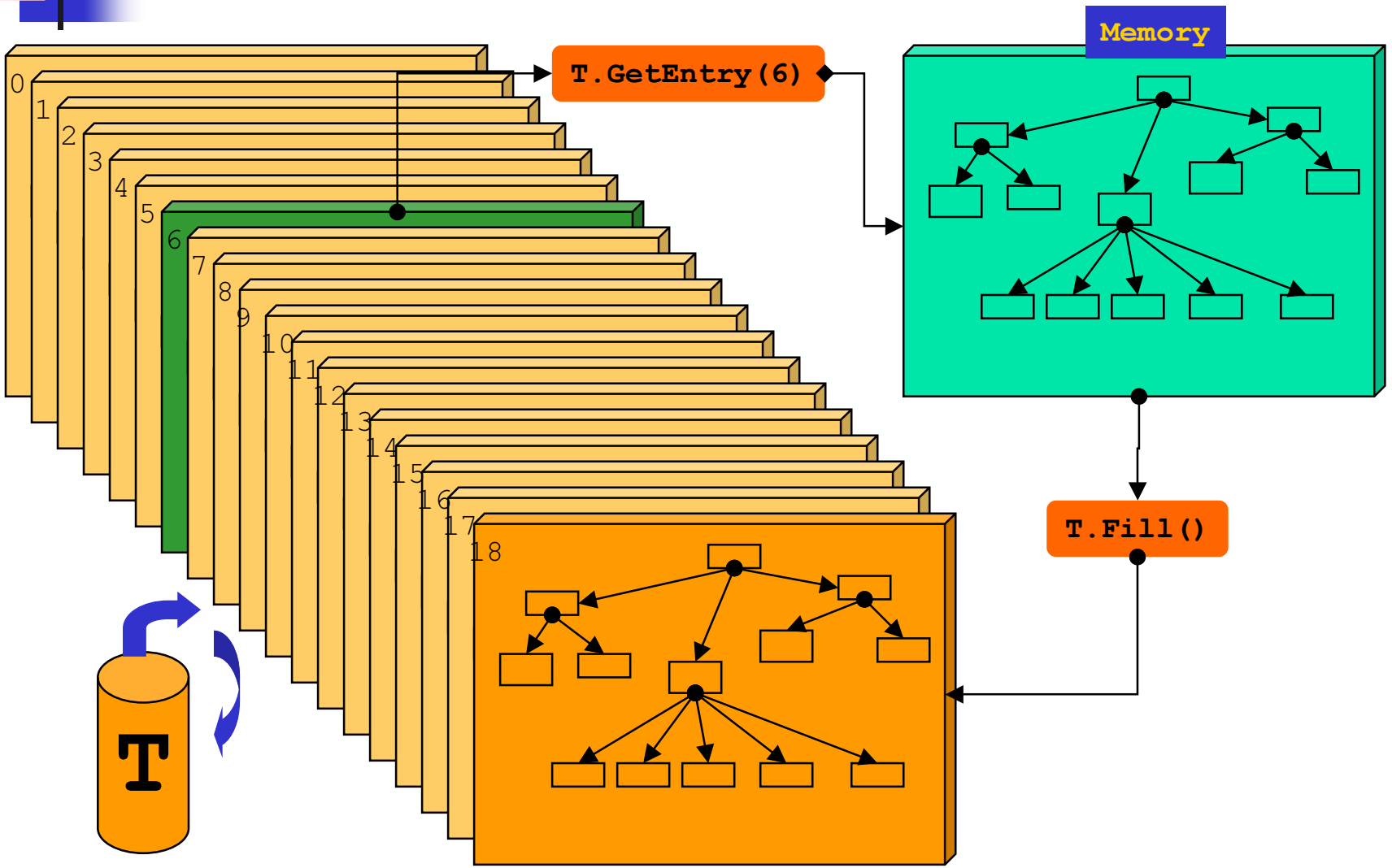
```
TFile f("example.root");
TH1F *h = (TH1F*)f.Get("h");
h->Draw();
f.Map();
```



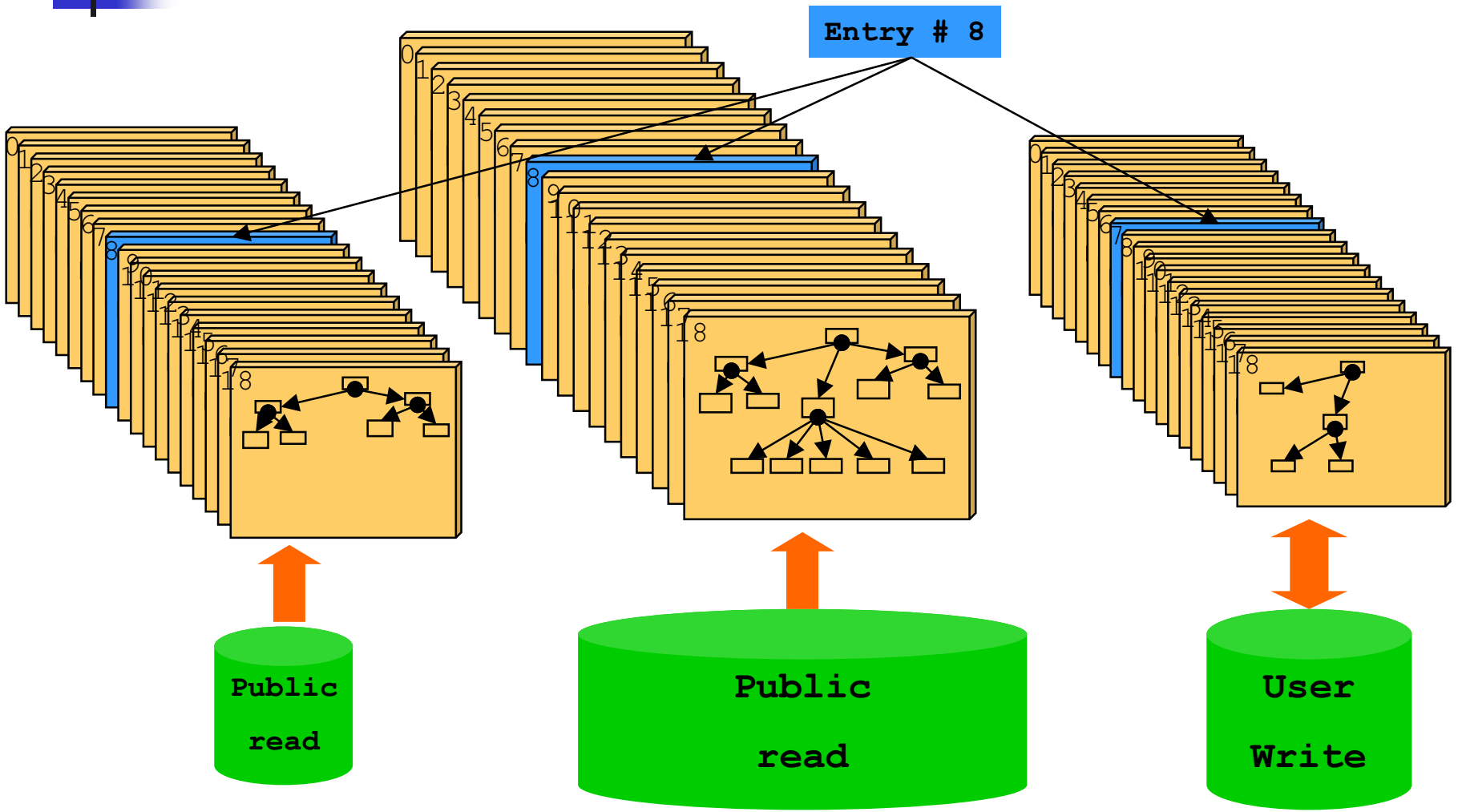

```
20010831/171903  At:64      N=90      TFile
20010831/171941  At:154     N=453     TH1F          CX =  2.09
20010831/171946  At:607     N=2364    StreamerInfo  CX =  3.25
20010831/171946  At:2971    N=96      KeysList
20010831/171946  At:3067    N=56      FreeSegments
20010831/171946  At:3123    N=1       END
```

Memory <--> Tree

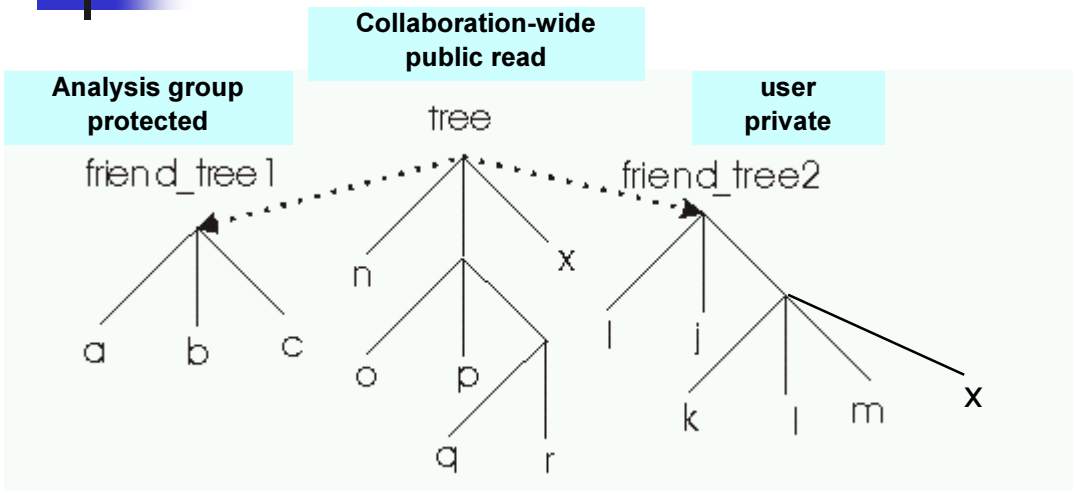
The Tree entry serial number



Tree Friends



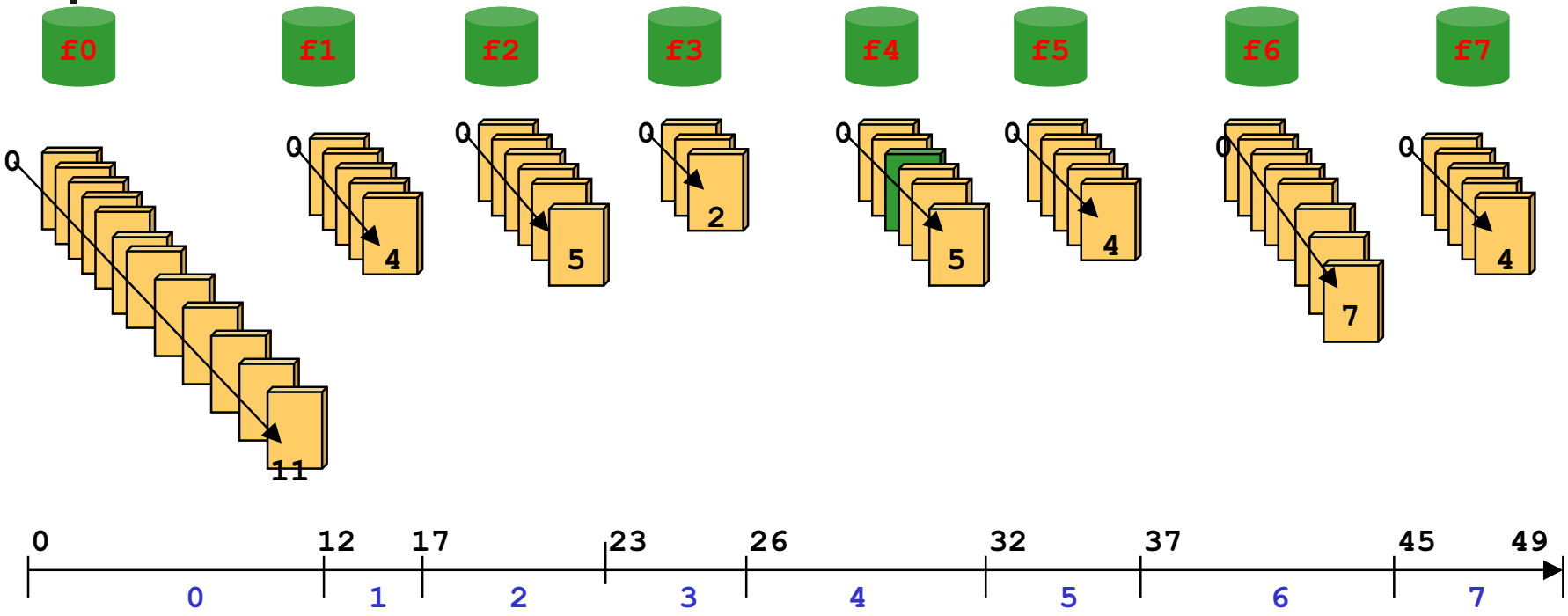
Tree Friends



Processing time independent of the number of friends unlike table joins in RDBMS

```
Root > TFile f1("tree1.root");
Root > tree.AddFriend("tree2","tree2.root")
Root > tree.AddFriend("tree3","tree3.root");
Root > tree.Draw("x:a","k<c");
Root > tree.Draw("x:tree2.x","sqrt(p)<b");
```

Chains of Trees



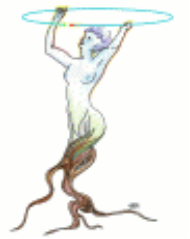

```
TChain ch("T");
ch.Add("f0.root");
ch.Add("f1.root");
ch.Add("f2.root");
ch.Add("f3.root");
ch.Add("f4.root");
ch.Add("f5.root");
ch.Add("f6.root");
ch.Add("f7.root");
```

```
ch.GetEntry(28);
```

Binary search in table above
find slot 4, local entry 2
T.GetEntry(2) in f4.root

```
ch.GetEntryWithIndex(12,567);
```

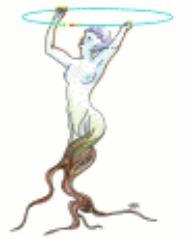
Existing TRef, TRefArray



- Designed as light weight entities
- Assume large number of TRefs per event
- Very fast dereferencing (direct access tables)
- Cannot (not designed for) find an object in a file

TLongRef, TLongID classes proposed for references with load on demand

TRef/TRefArray advantages



- TRef is perfect for referencing objects like hits, clusters, tracks that may be > 10000.
- You would not like to have the size of a TRef bigger than the size of its referenced object !
- A TRef occupies in average 2.5 bytes in the file
- There is no point in providing load on demand for one single hit, cluster or track.

TRef example: Event.h

```
class Event : public TObject {

private:
   char           fType[20];          //event type
   char          *fEventName;         //run+event number in character format
   int            fNtrack;            //Number of tracks
   int            fNseg;              //Number of track segments
   int            fNvertex;
   int            fMeasures[10];
   float          fMatrix[4][4];
   float         *fClosestDistance;   //[fNvertex]
   EventHeader    fEvtHdr;
   TClonesArray  *fTracks;            //->array with all tracks
   TRefArray     *fHighPt;            //array of High Pt tracks only
   TRefArray     *fMuons;             //array of Muon tracks only
   TRef           fLastTrack;         //reference pointer to last track
   TRef           fWebHistogram;      //EXEC:GetWebHistogram
   TH1F          *fH;                 //->


public:
   ...
   TH1F         *GetHistogram() const {return fH;}
   TH1F         *GetWebHistogram(Bool_t reload=kFALSE)  const {
                   return (TH1F*)fWebHistogram.GetObject(reload);}
```

Can also do load on demand

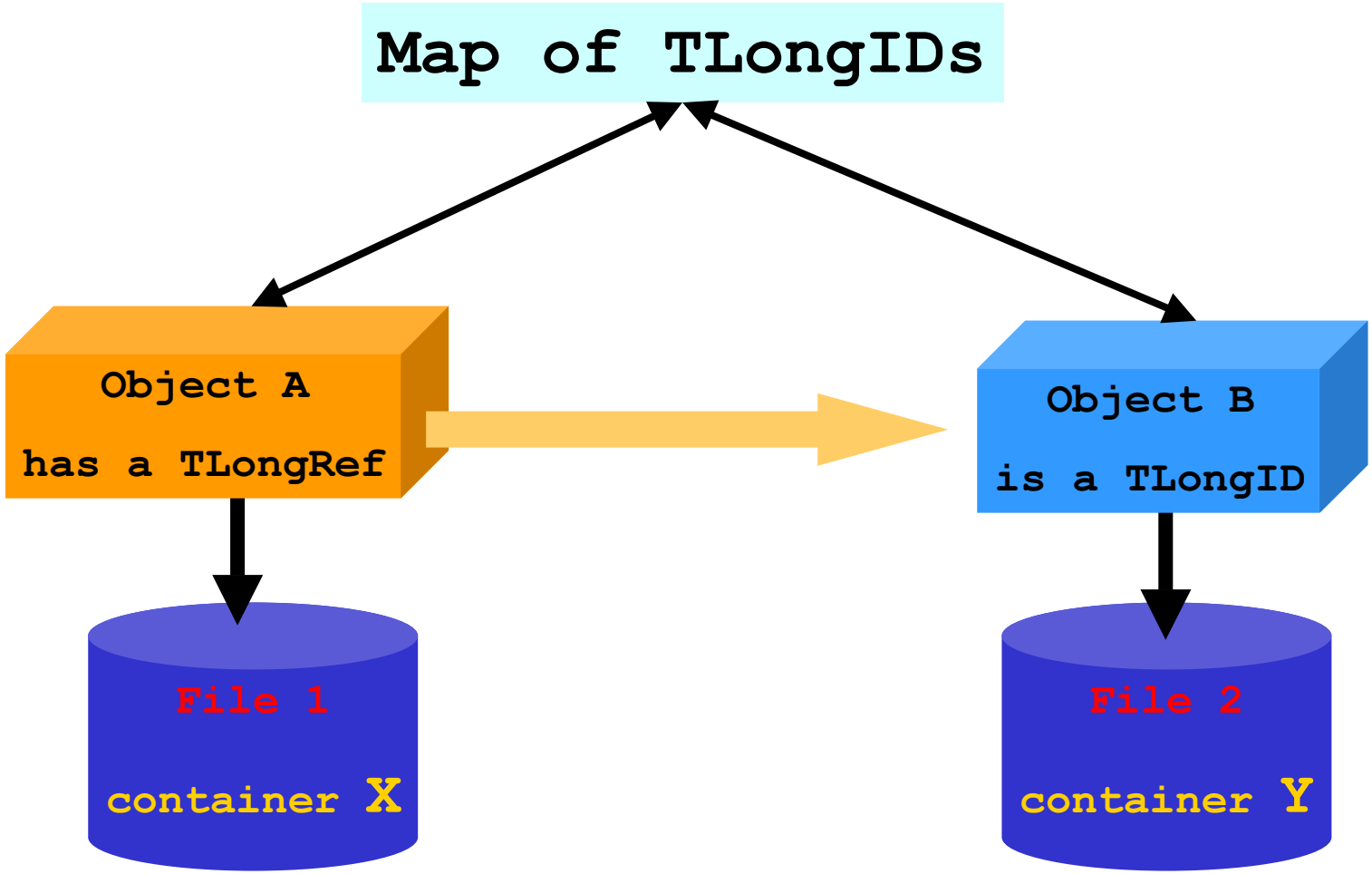
Load on demand

- It makes sense for objects like
  - large collections of hits, clusters, tracks
  - files
  - mag field
  - geometry
- Assuming that an event will contain < 100 such objects to be requested on demand, there is no problem in having fat references (eg 50 bytes)

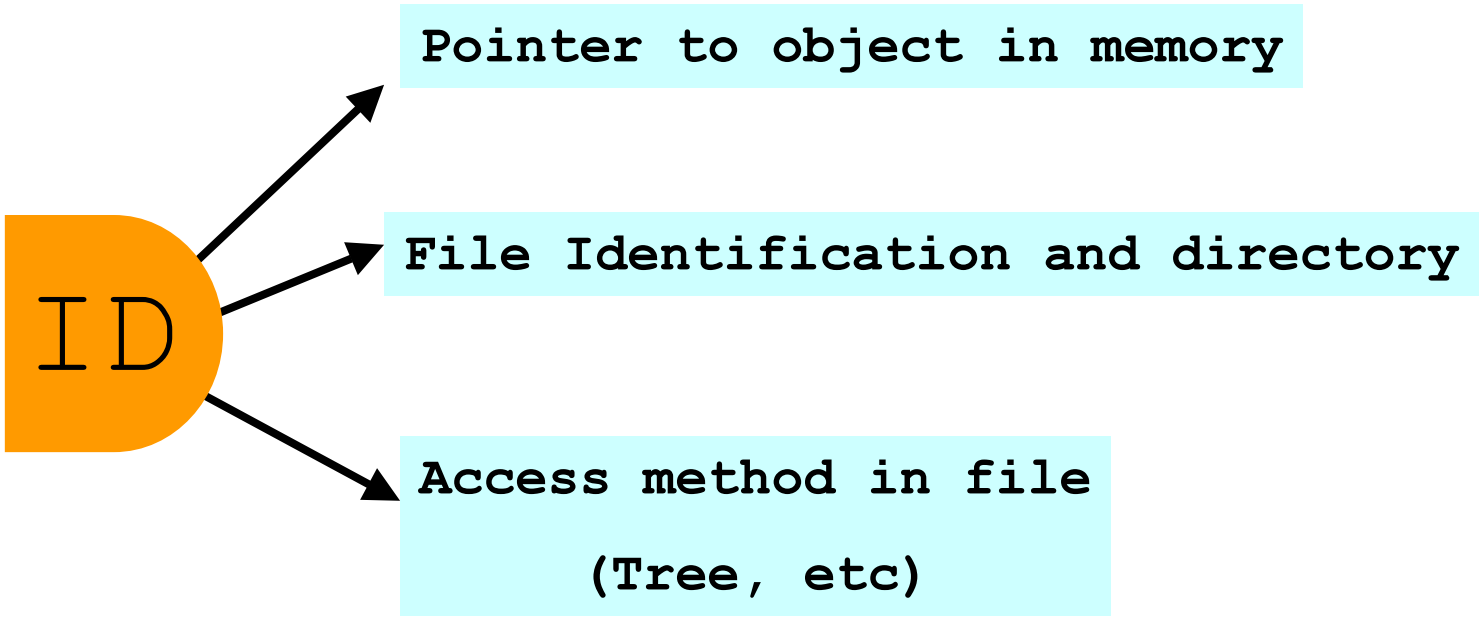

```
TLongRef

TLongID
```

TLongRef, TLongID



Object Identification



TLongID

- An object referenced by a TLongRef must inherit from class TLongID
- When a TLongID is created, it is added to one single table (map) of LongIDs
- When a TLongID is written to a file, its persistent components are written to the file.
- When a TLongID is read, it is added to the map of LongIDs.

TLongRef

- A TLongRef points to an object inheriting from TLongID.
- TLongRef attributes are identical to TLongID. In addition it includes a pointer to the object.
- When a TLongRef is written, its TLongID components are written.
- When a TLongRef is dereferenced, its pointer is computed (if not already there) by searching in the map of TLongIDs.

TLongID

Up to 128 bits

```
root [0] TUUID u
root [1] u.AsString()
(const char* 0x40476a80)"c62ad97a-78c9-11d6-9e58-4ed58a89beef"
```

```
class TLongID {
TUUID  fUUID;
TString fText1;
TString fText2;
etc…
}
```

TUUID unique in time (nanoseconds) and space

Additional info to be discussed

TLongID could be reduced to TUUID

TLongRef

```
class TLongRef {
TUUID  fUUID;
TString fWhere;
TString fHow;
etc…
}
```

A copy of the TUUID in TLongID

Additional info to be discussed

FileID

subdirectory

branch in Tree, etc