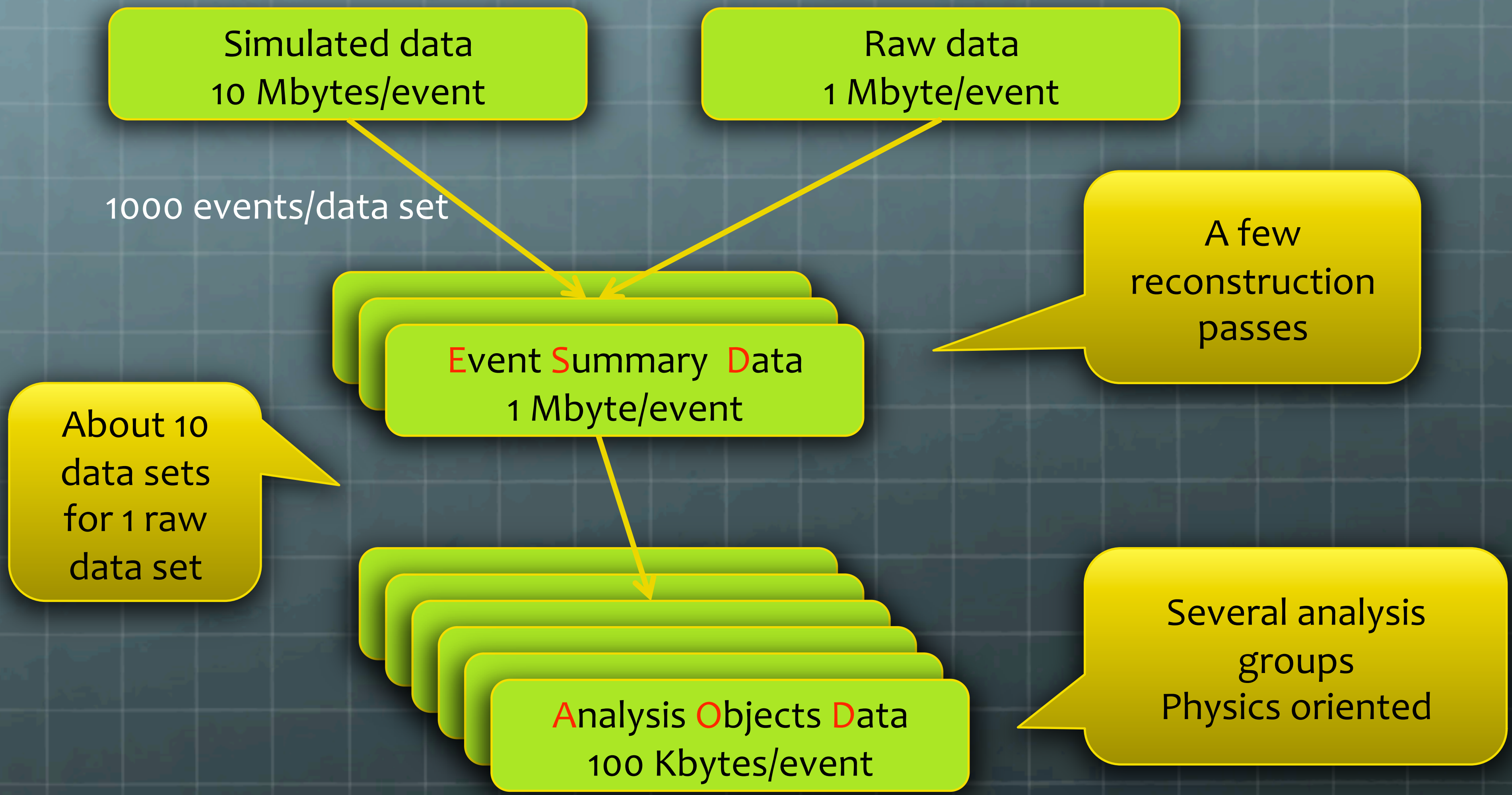
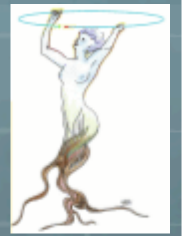
How ROOT can help
to store and analyze data in the short/medium/long term

DPHEP workshop
CERN December 7
Rene Brun/CERN

Data Sets types





Data Sets Total Volume

- Each experiment will take about 1 billion events/year
- 1 billion events → 1 million raw data sets of 1 Gbyte
- ===→10 million data sets with ESDs and AODs
- ==→ 100 million data sets with the replica on the GRID
- All event data are C++ objects streamed to ROOT files

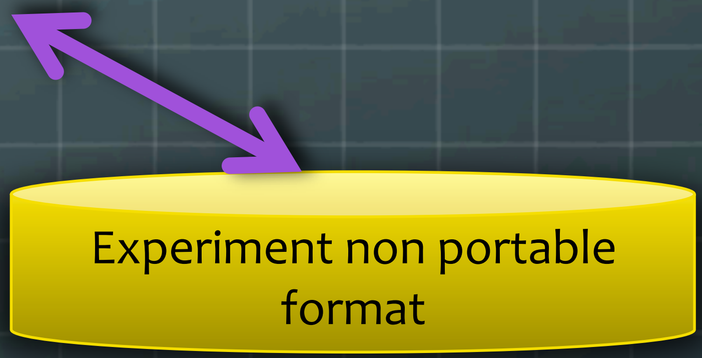
How did we reach this point

- Today's situation with ROOT + RDBMS was reached circa 2002 when it was realized that an alternative solution based on an object-oriented data base (Objectivity) was not optimum.
- It took us a long time to understand that a file format alone was not sufficient and that an automatic object streaming for any C++ class was fundamental.
- One more important step was reached when we understood the importance of self-describing files and automatic class schema evolution.

The situation in the 70s

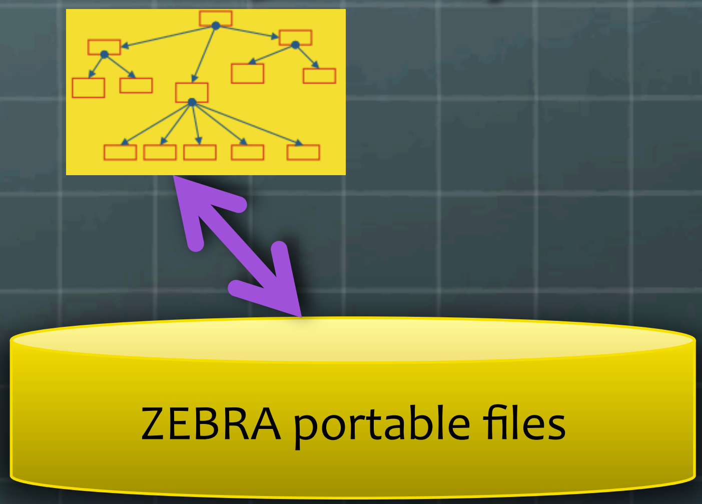
- Fortran programming. Data in common blocks written and read by user controlled subroutines.
- Each experiment has his own format. The experiments are small (10 to 50 physicists) with short life time (1 to 5 years).

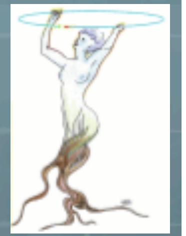
```
common /data1/np, px(100),py(100),pz(100)…
common/data2/nhits, adc(50000), tdc(10000)
```



The situation in the 80s

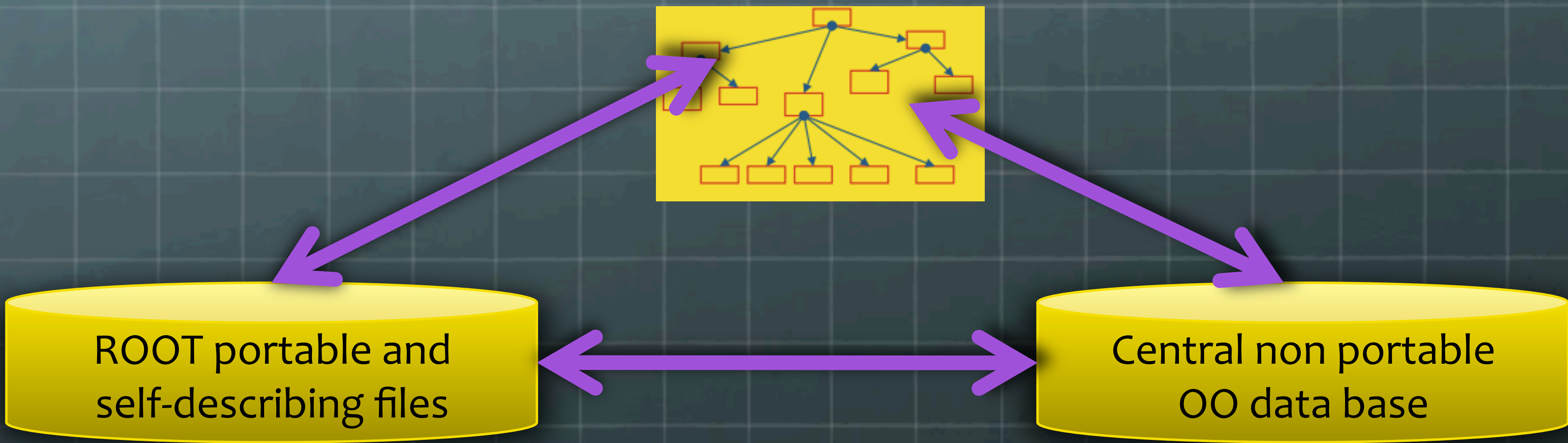
- Fortran programming. Data in banks managed by data structure management systems like **ZEBRA**, BOS writing portable files with some data types description.
- The experiments are bigger (100 to 500 physicists) with life time between 5 and 10 years.



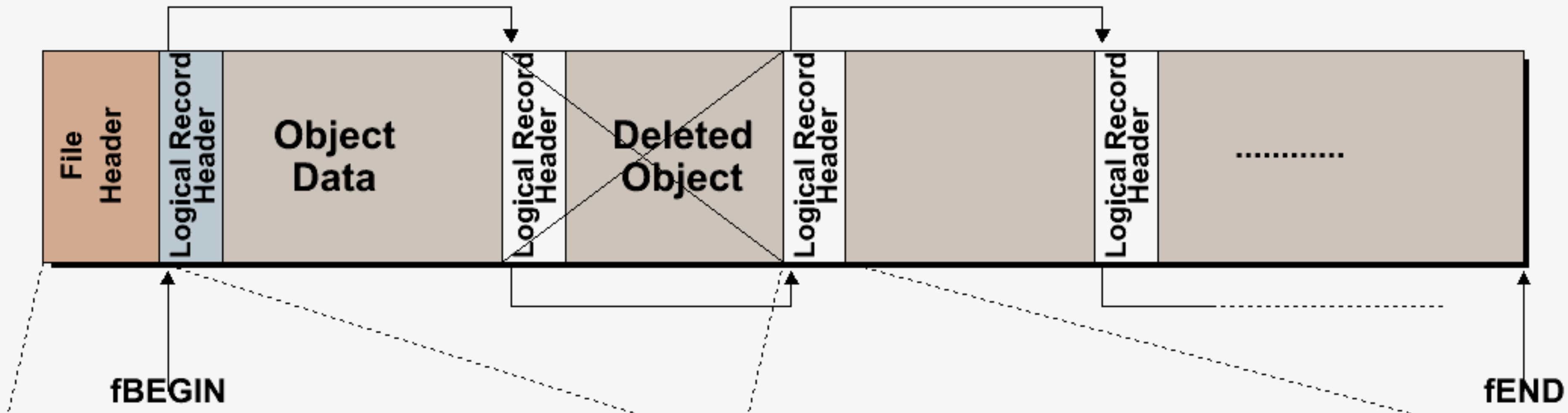


The situation in the 90s

- Painful move from Fortran to C++.
- Drastic choice between HEP format (ROOT) or a commercial system (Objectivity).
- It took more than 5 years to show that a central OO data base with transient object = persistent object and no schema evolution could not work
- The experiments are huge (1000 to 2000 physicists) with life time between 15 and 25 years.



ROOT File description



File Header

"root": Root File Identifier
fVersion: File version identifier
fBEGIN: Pointer to first data record
fEND: Pointer to first free word at EOF
fSeekFree: Pointer to FREE data record
fNbytesFree: Number of bytes in FREE
fNfree: Number of free data records
fNbytesName: Number of bytes in name/title
fUnits: Number of bytes for pointers
fCompress: Compression level

Logical Record Header (TKEY)

fNbytes: Length of compressed object
fVersion: Key version identifier
fObjLen: Length of uncompressed object
fDatime: Date/Time when written to store
fKeylen: Number of bytes for the key
fCycle : Cycle number
fSeekKey: Pointer to object on file
fSeekPdir: Pointer to directory on file
fClassName: class name of the object
fName: name of the object
fTitle: title of the object

TFile::Map

```
root [0] TFile *falice = TFile::Open("http://root.cern.ch/files/alice_ESDs.root")
root [1] falice->Map()
20070713/195136  At:100     N=120       TFile
20070713/195531  At:220     N=274       TH1D           CX =   7.38
20070713/195531  At:494     N=331       TH1D           CX =   2.46
20070713/195531  At:825     N=290       TH1D           CX =   2.80
….
20070713/195531  At:391047  N=1010      TH1D           CX =   3.75
Address = 392057        Nbytes = -889   =====G A P===========
20070713/195519  At:392946  N=2515      TBasket        CX = 195.48
20070713/195519  At:395461  N=23141     TBasket        CX =   1.31
20070713/195519  At:418602  N=2566      TBasket        CX = 10.40
20070713/195520  At:421168  N=2518      TBasket        CX = 195.24
20070713/195532  At:423686  N=2515      TBasket        CX = 195.48
20070713/195532  At:426201  N=2023      TBasket        CX = 15.36
20070713/195532  At:428224  N=2518      TBasket        CX = 195.24
20070713/195532  At:430742  N=375281    TTree          CX =   4.2[illegible]
20070713/195532  At:806023  N=43823     TTree          CX = 1[illegible].84
20070713/195532  At:849846  N=6340      TH2F           CX = [illegible]
20070713/195532  At:856186  N=951       TH1F           CX =   9.02
20070713/195532  At:857137  N=16537     StreamerInfo   CX =   3.74
20070713/195532  At:873674  N=1367      KeysList
20070713/195532  At:875041  N=1         END
root [2]
```

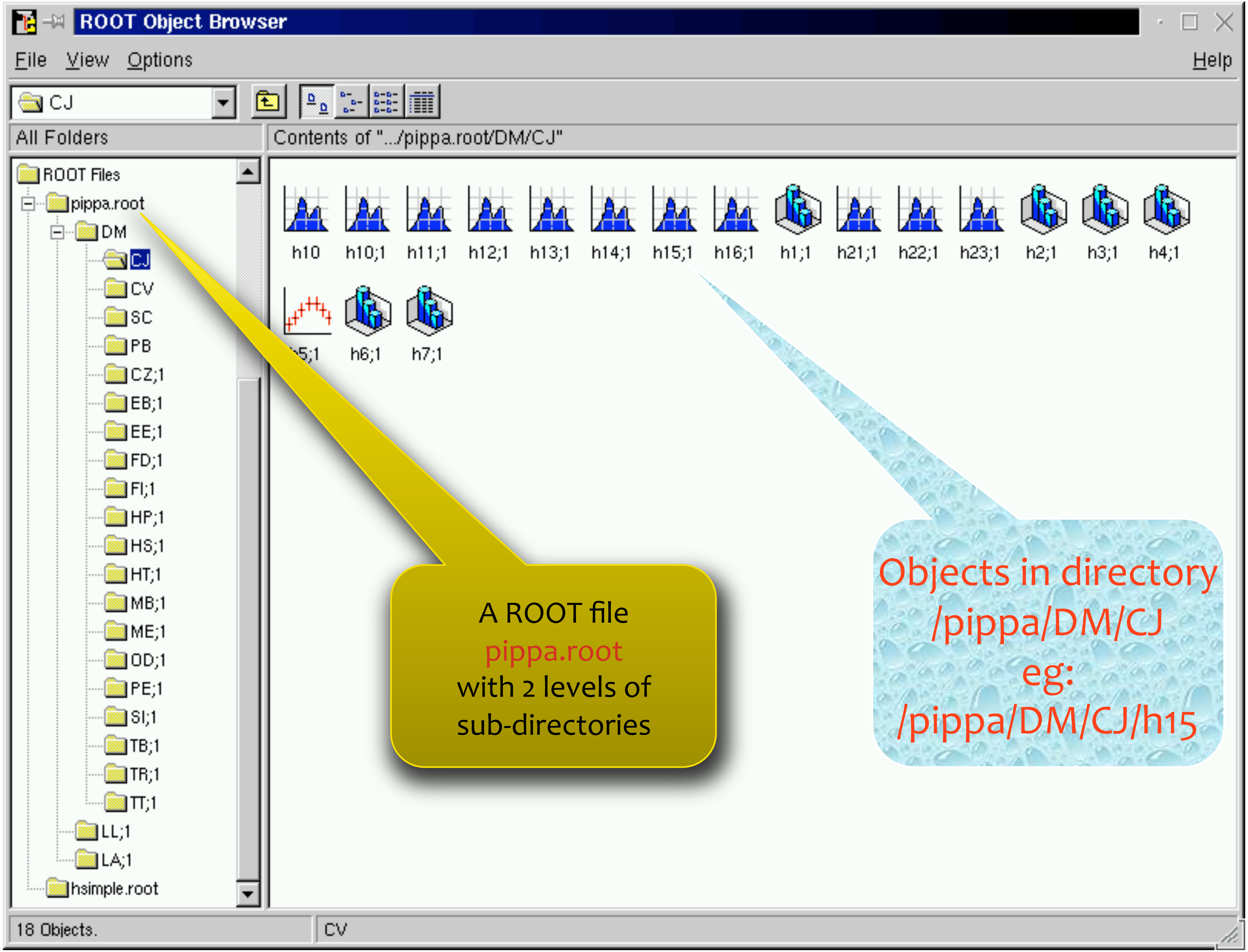
Classes dictionary

List of keys

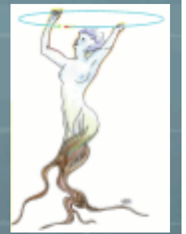
Self-describing files

- Dictionary for persistent classes written to the file.
- ROOT files can be read by foreign readers
- Support for Backward and Forward compatibility
- Files created in 2001 must be readable in 2015
- Classes (data objects) for all objects in a file can be regenerated via TFile::MakeProject

```
Root >TFile f("demo.root");
Root > f.MakeProject("dir","*","new++");
```


ROOT Object Browser
File View Options Help
CJ
All Folders
Contents of ".../pippa.root/DM/CJ"
ROOT Files
pippa.root
DM
CJ
CV
SC
PB
CZ;1
EB;1
EE;1
FD;1
FI;1
HP;1
HS;1
HT;1
MB;1
ME;1
OD;1
PE;1
SI;1
TB;1
TR;1
TT;1
LL;1
LA;1
hsimple.root
h10 h10;1 h11;1 h12;1 h13;1 h14;1 h15;1 h16;1 h1;1 h21;1 h22;1 h23;1 h2;1 h3;1 h4;1
h6;1 h7;1
18 Objects.
CV
A ROOT file
pippa.root
with 2 levels of
sub-directories
Objects in directory
/pippa/DM/CJ
eg:
/pippa/DM/CJ/h15

TFile::MakeProject



Generate C++ header files
and shared lib for the cla

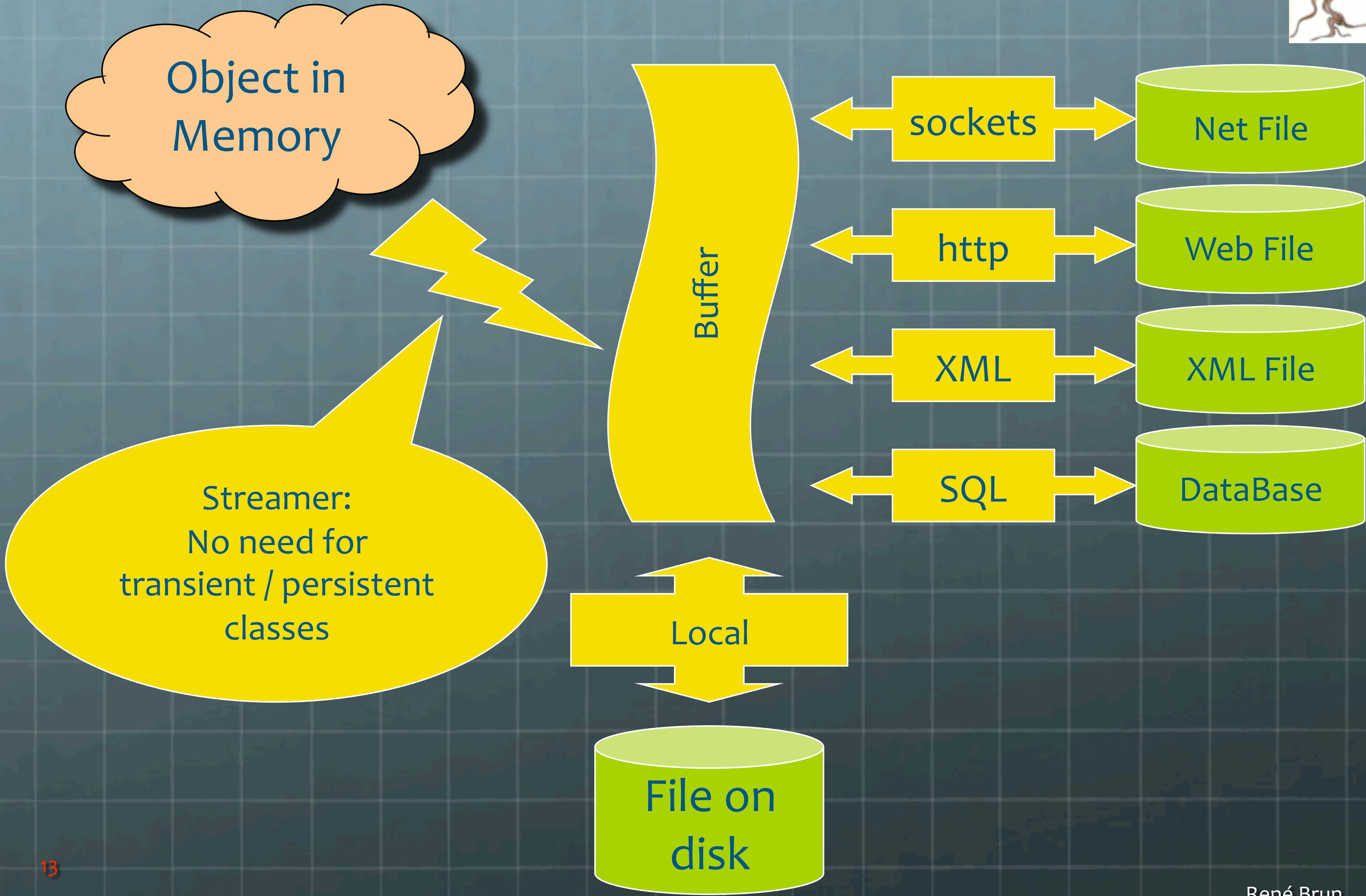
```
(macbrun2) [253] root
root [0] TFile *falice = TFile::Open("http://roo
root [1] falice->MakeProject("alice","*","++");
MakeProject has generated 26 classes in alice
alice/MAKEP file has been generated
Shared lib alice/alice.so has been generated
Shared lib alice/alice.so has been dynamically l
root [2] .!ls alice
AliESDCaloCluster.h       AliESDZDC.h
AliESDCaloTrigger.h       AliESDcascade.h
AliESDEvent.h             AliESDfriend.h
AliESDFMD.h               AliESDfriendTrack.h
AliESDHeader.h            AliESDkink.h
AliESDMuonTrack.h         AliESDtrack.h
AliESDPmdTrack.h          AliESDv0.h
AliESDRun.h               AliExternalTrackParam.h
AliESDTZERO.h             AliFMDFloatMap.h
AliESDTrdTrack.h          AliFMDMap.h
AliESDVZERO.h             AliMultiplicity.h
AliESDVertex.h            AliRawDataErrorLog.h
root [3]
```

AliESDCaloCluster.h

AliESDCaloCluster.h:2 <No selected symbol>

```
//////////////////////////////////////////////////////////
//   This class has been generated by TFile::MakeProject
//     (Sat Jan 24 15:24:51 2009 by ROOT version 5.23/01)
//      from the StreamerInfo in file http://root.cern.ch/files/alice_ESDs.root
//////////////////////////////////////////////////////////


#ifndef AliESDCaloCluster_h
#define AliESDCaloCluster_h
class AliESDCaloCluster;

#include "TObject.h"
#include "TArrayS.h"

class AliESDCaloCluster : public TObject {

public:
   Int_t      fID;                   //Unique Id of the cluster
   Int_t      fClusterType;          //Flag for different clustering versions
   Bool_t     fEMCALCluster;         //Is this is an EMCAL cluster?
   Bool_t     fPHOSCluster;          //Is this is a PHOS cluster?
   Float_t    fGlobalPos[3];         //position in global coordinate system
   Float_t    fEnergy;               //energy measured by calorimeter
   Float_t    fDispersion;           //cluster dispersion, for shape analysis
   Float_t    fChi2;                 //chi2 of cluster fit
   Float_t    fPID[10];              //"detector response probabilities" (for the PID)
   Float_t    fM20;                  //2-nd moment along the main eigen axis
   Float_t    fM02;                  //2-nd moment along the second eigen axis
   Float_t    fM11;                  //2-nd mixed moment Mxy
   UShort_t   fNExMax;               //number of (Ex-)maxima before unfolding
   Float_t    fEmcCpvDistance;       //the distance from PHOS EMC rec.point to the closes
   Float_t    fDistToBadChannel;     //Distance to nearest bad channel
   TArrayS*   fTracksMatched;        //Index of tracks close to cluster. First entry is t
   TArrayS*   fLabels;               //list of primaries that generated the cluster, orde
   TArrayS*   fDigitAmplitude;       //digit energy (integer units)
   TArrayS*   fDigitTime;            //time of this digit (integer units)
   TArrayS*   fDigitIndex;           //calorimeter digit index

   AliESDCaloCluster();
   virtual ~AliESDCaloCluster();

   ClassDef(AliESDCaloCluster,5); // Generated by MakeProject.
};
#endif
```

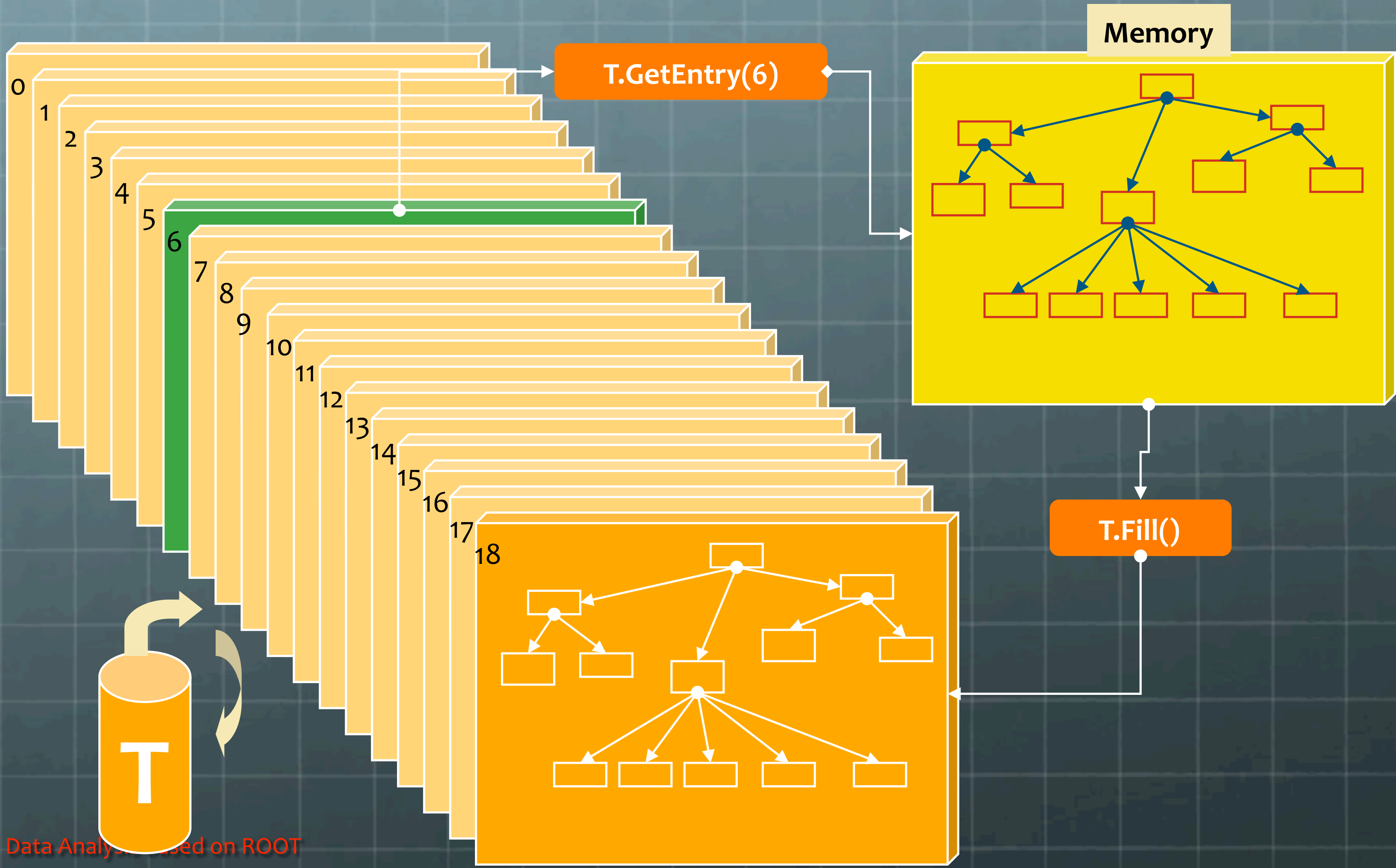
I/O



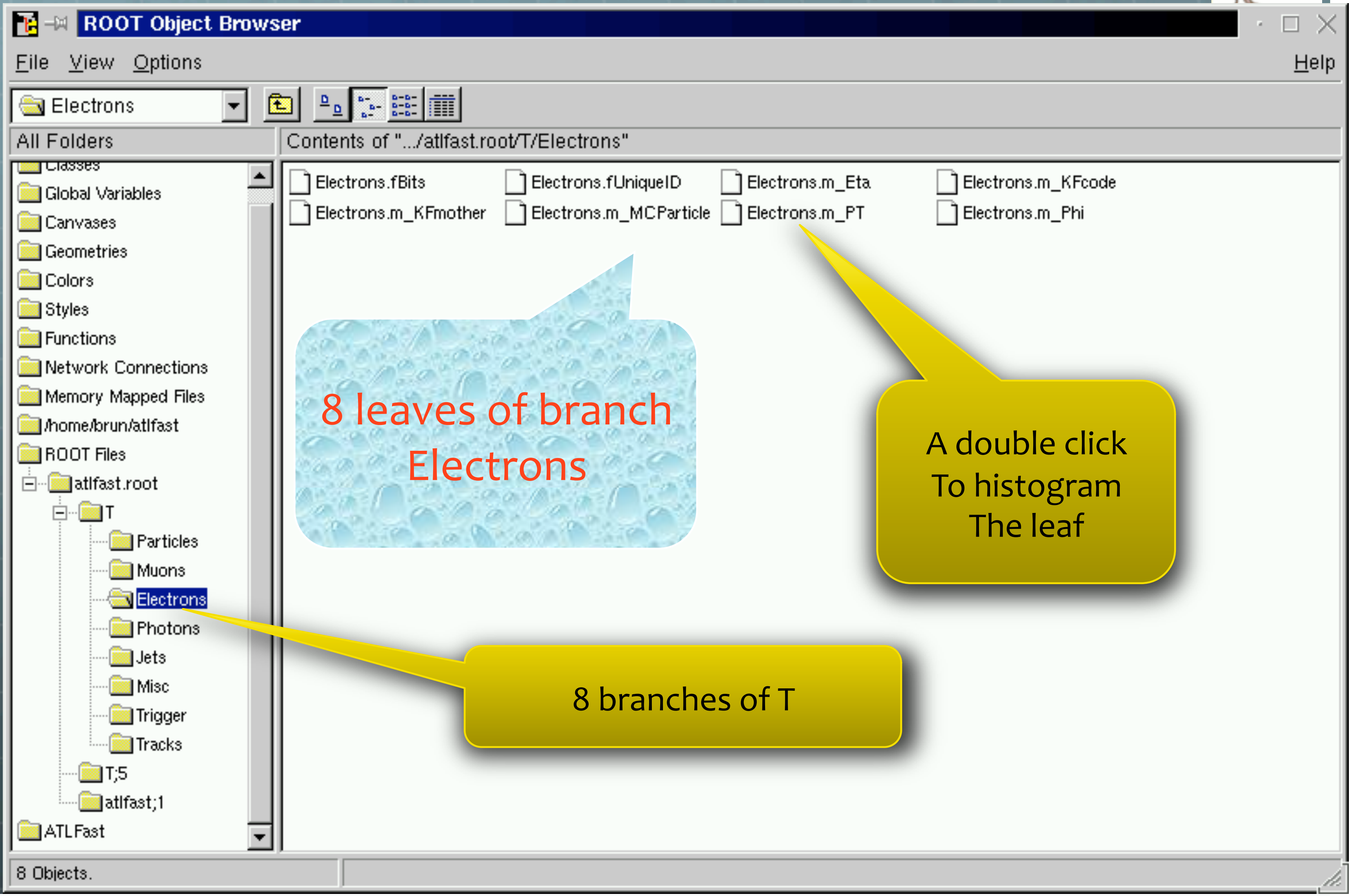
Object in Memory
Streamer:
No need for
transient / persistent
classes
Buffer
sockets
http
XML
SQL
Net File
Web File
XML File
DataBase
Local
File on disk

Memory <--> Tree

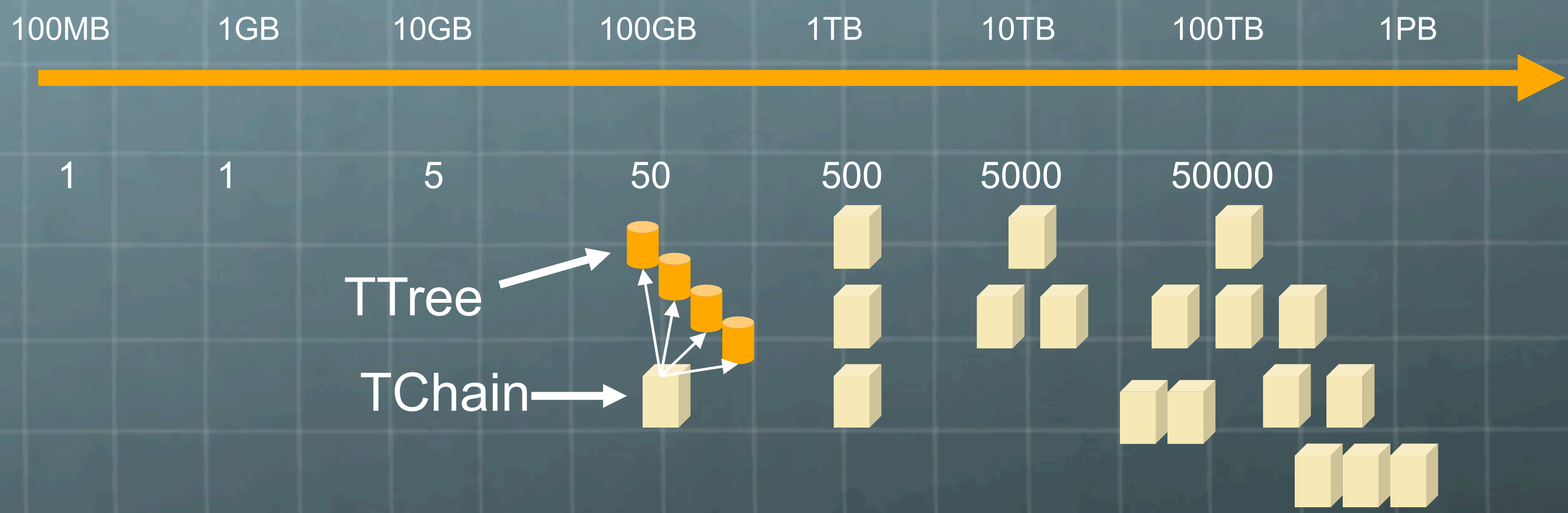
Each Node is a branch in the Tree



Browsing a TTree with TBrowser



Data Volume & Organization



A TFile typically contains 1 TTree

A TChain is a collection of TTrees or/and TChains

A TChain is typically the result of a query to the file catalogue

The situation in 2000-a

- Following the failure of the OODBMS system, an attempt to store event data in a relational data base fails also quite rapidly when we realized that RDBMS systems are not designed to store petabytes of data.
- The ROOT system is adopted by the large US experiments at FermiLab and BNL. This version is based on object streamers specific to each class and generated automatically by a preprocessor.

The situation in 2000-b

- Although automatically generated object streamers were quite powerful, they required the class library containing the streamer code at read time.
- We realized that this will not fly in the long term as it is quite obvious that the streamer library used to write the data will not likely be available when reading the data several years later.

The situation in 2000-c

- A system based on class dictionaries saved together with the data was implemented in ROOT. This system was able to write and read objects using the information in the dictionaries only and did not required anymore the class library used to write the data.

- In addition the new reader is able to process in the same job data generated by successive class versions.

- This process, called automatic class-schema-evolution proved to be a fundamental component of the system.

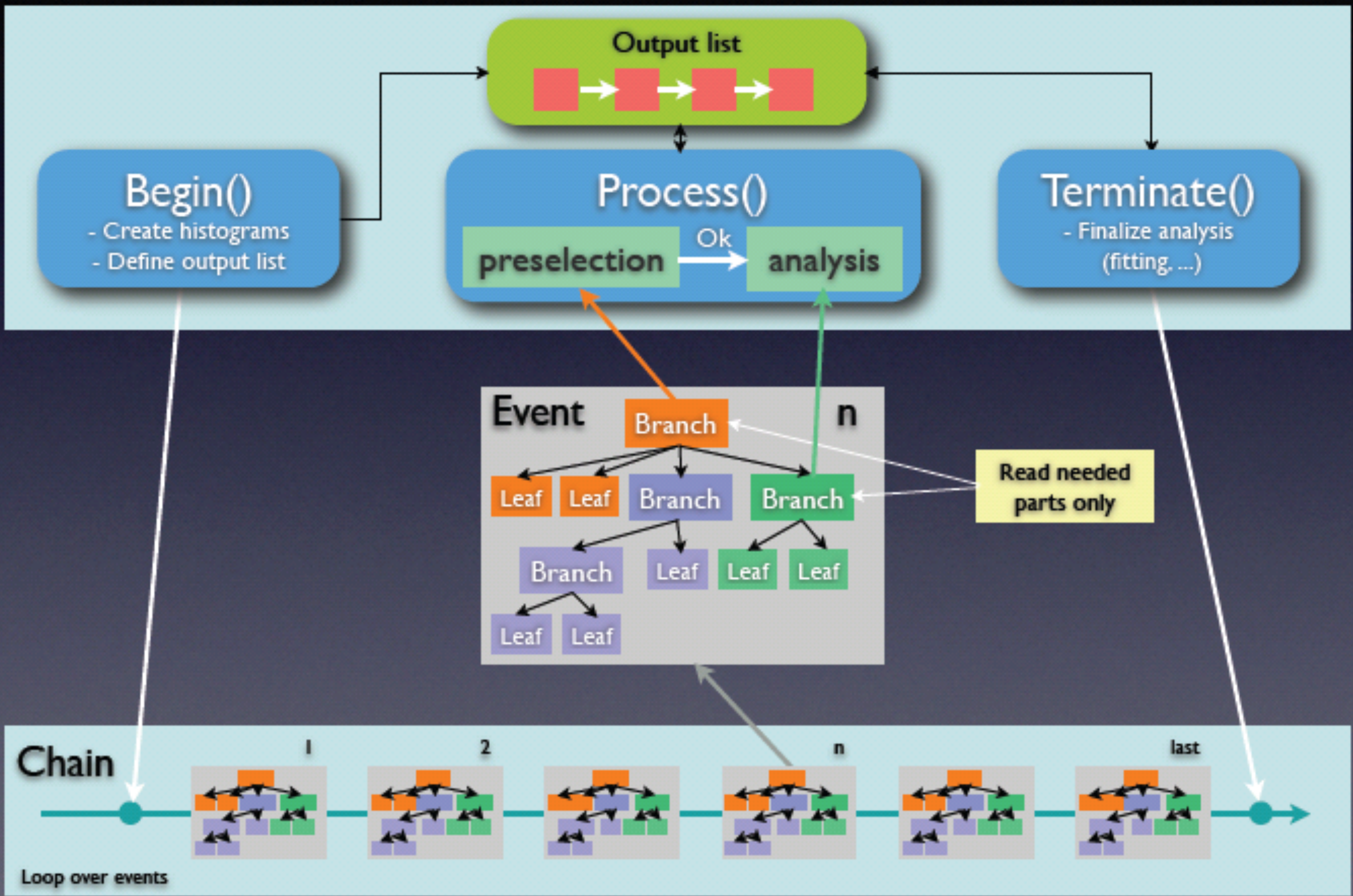
- It was a huge difference with the OODBMS and RDBMS systems that forced a conversion of the data sets to the latest class version.

The situation in 2000-d

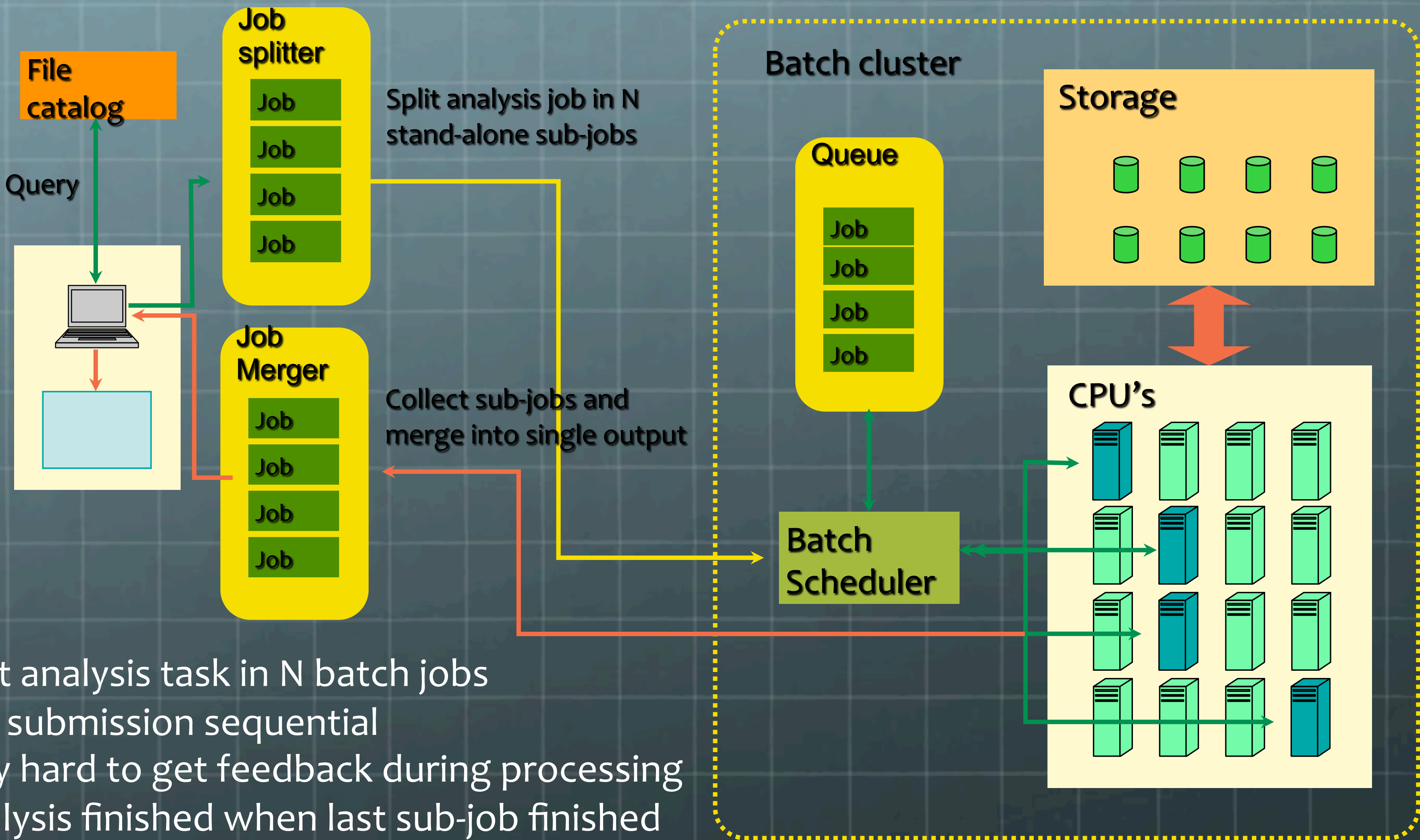
- Circa 2000 it was also realized that streaming objects or objects collections in one single buffer was totally inappropriate when the reader was interested to process only a small subset of the event data.
- The ROOT Tree structure was not only a Hierarchical Data Format, but was designed to be optimal when
  - The reader was interested by a subset of the events, by a subset of each event or both.
  - The reader has to process data on a remote machine across a LAN or WAN.
- The TreeCache minimizes the number of transactions and also the amount of data to be transferred.

The ROOT Data Model

Trees & Selectors

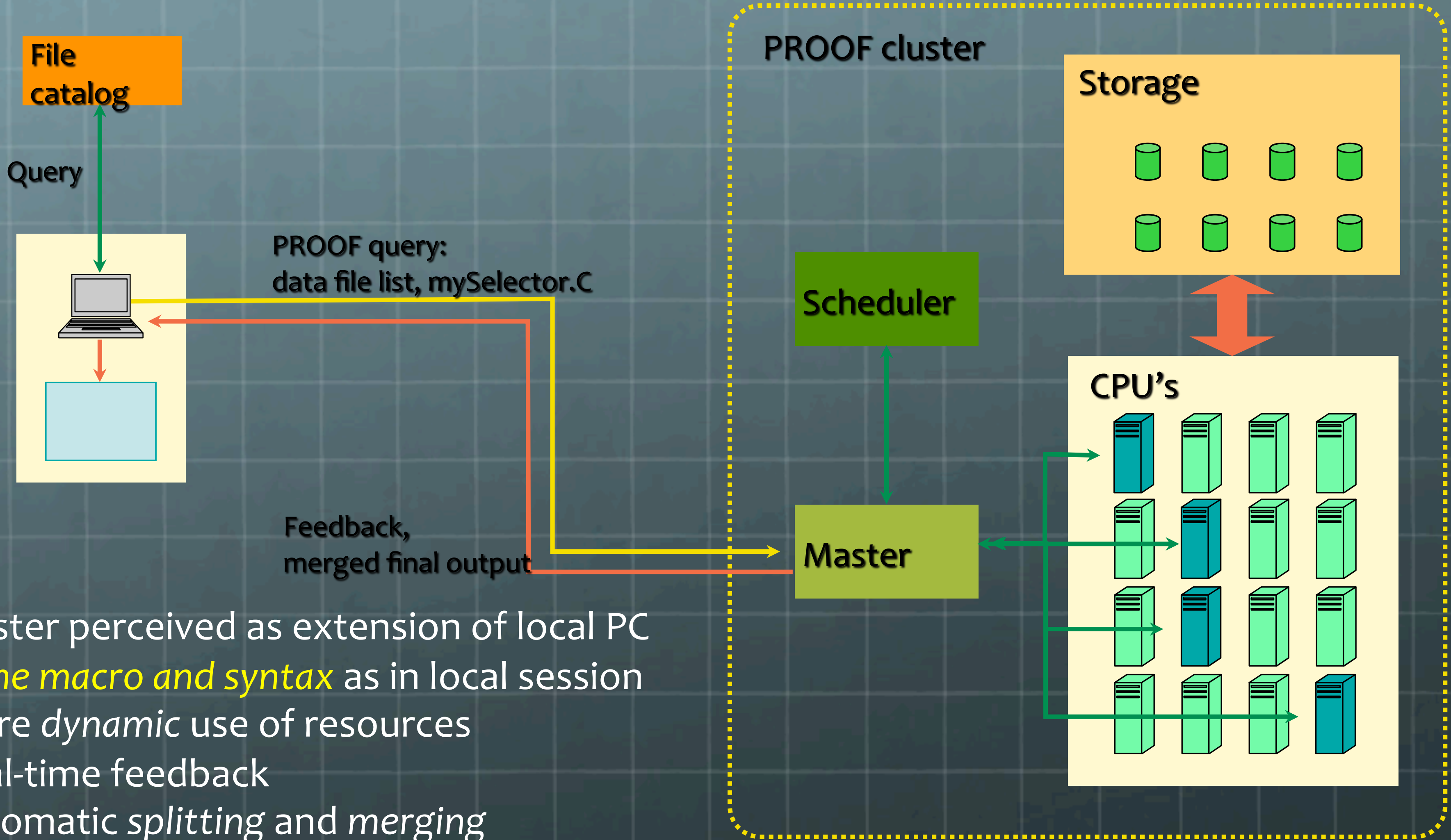


Traditional Batch Approach



- Split analysis task in N batch jobs
- Job submission sequential
- Very hard to get feedback during processing
- Analysis finished when last sub-job finished

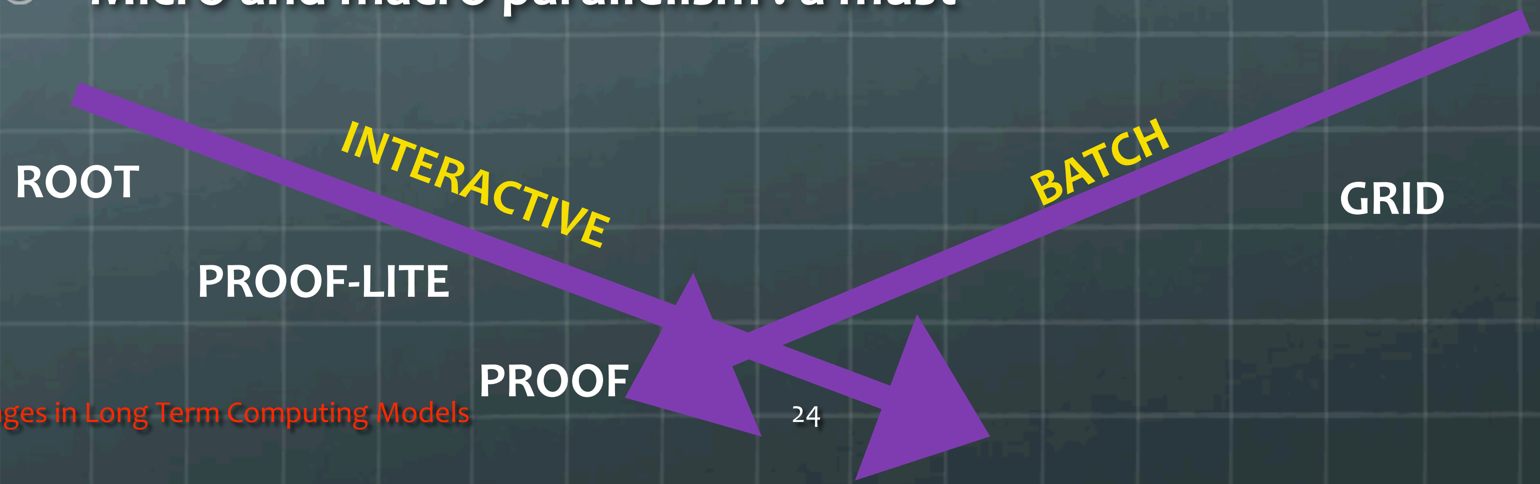
The PROOF Approach



- Cluster perceived as extension of local PC
- *Same macro and syntax* as in local session
- More *dynamic* use of resources
- Real-time feedback
- Automatic *splitting* and *merging*

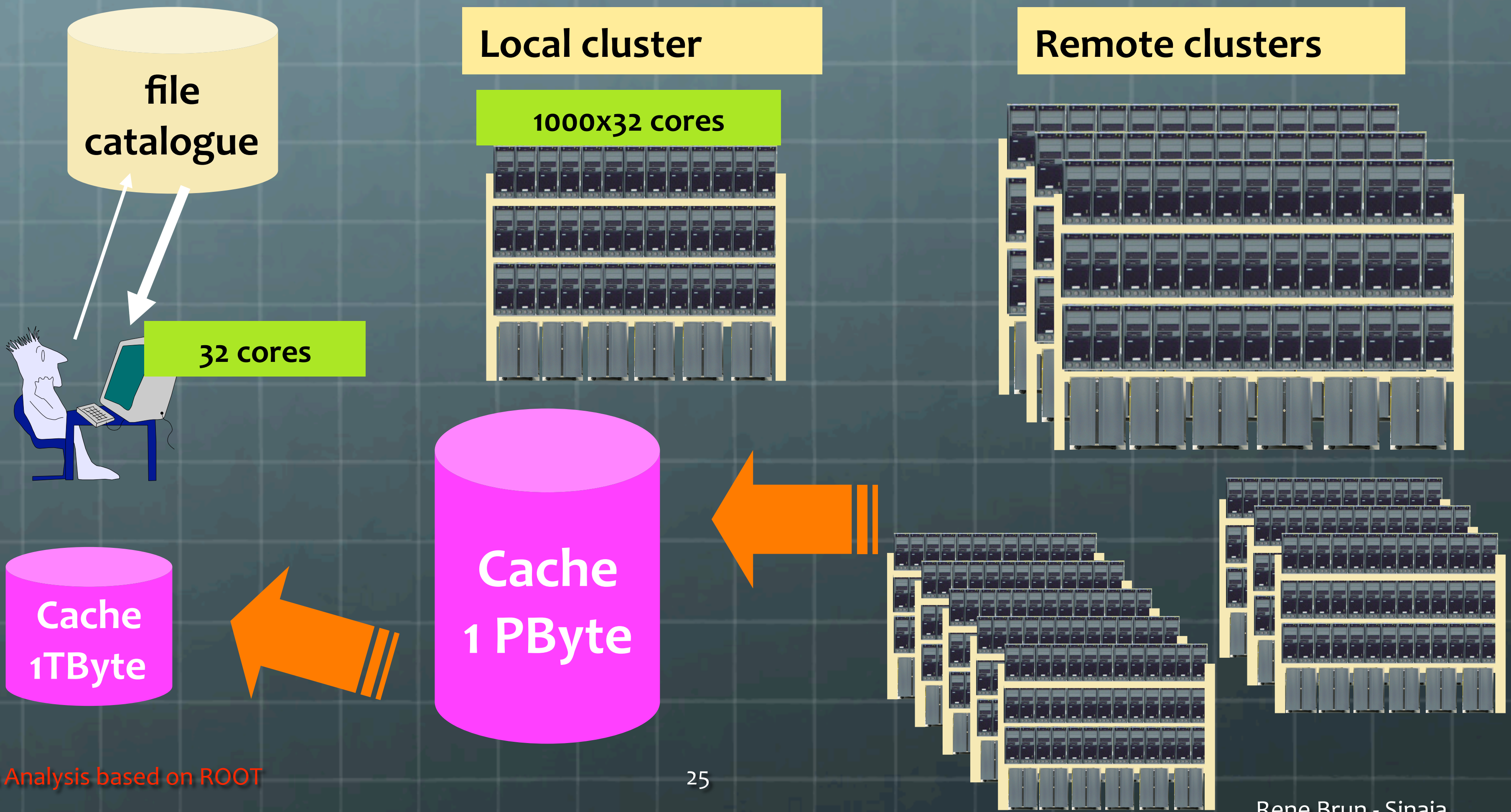
Stability, Robustness

- Substantial manpower effort
  - 8 FTE at CERN + 1.5 at FNAL
- Support in place for the LHC era
- Still many developments
  - I/O robustness for LHC is vital
  - I/O performance improvements
- Micro and macro parallelism : a must



Parallelism everywhere

- What about this picture in the very near future?



Summary

- The ROOT system is the result of 15 years of cooperation between the development team and thousands of heterogeneous users.
- ROOT is not only a file format, but mainly a general object I/O storage and management designed for rapid data analysis of very large shared data sets.
- It allows concurrent access and supports parallelism in a set of analysis clusters (LAN and WAN).