ROOT I/O
recent improvements
Bottlenecks and Solutions

Rene Brun

December 02 2009

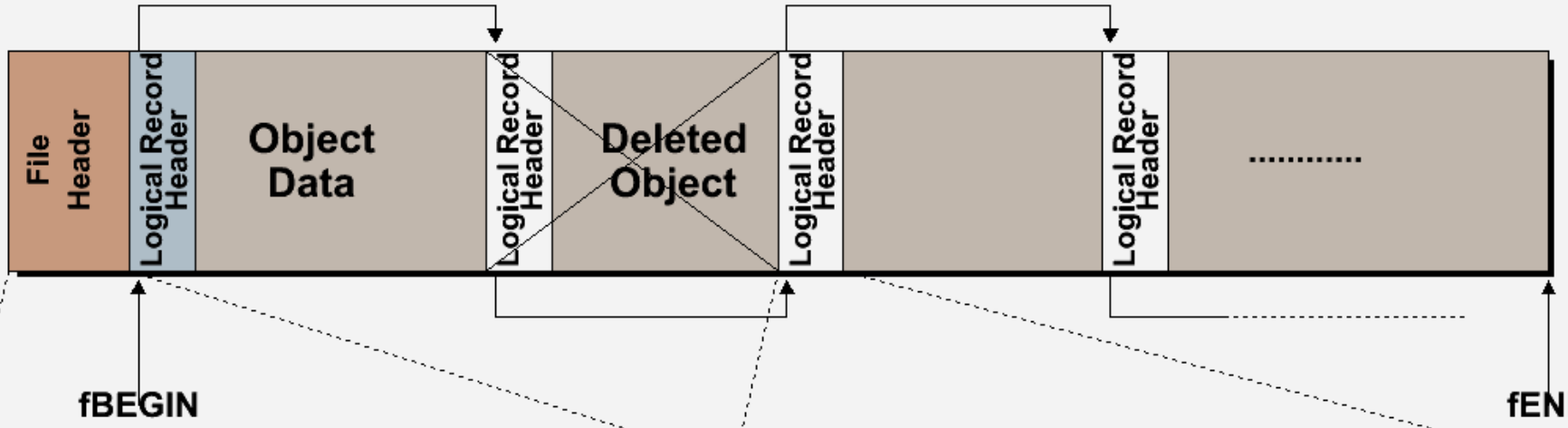
Main Points

- ROOT IO basics
- Bottlenecks analysis
- Solutions

ROOT IO

Basics

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
20070713/195532  At:806023  N=43823     TTree           CX =  1.84
20070713/195532  At:849846  N=6340      TH2F            CX = 100.63
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

TFile::ls

```
root [3] falice->ls()
  KEY: TH1D     logTRD_backfit;1
  KEY: TH1D     logTRD_refit;1
  KEY: TH1D     logTRD_clSearch;1
  KEY: TH1D     logTRD_X;1
  KEY: TH1D     logTRD_ncl;1
  KEY: TH1D     logTRD_nclTrack;1
  KEY: TH1D     logTRD_minYPos;1
  KEY: TH1D     logTRD_minYNeg;1
  KEY: TH2D     logTRD_minD;1
  KEY: TH1D     logTRD_minZ;1
  KEY: TH1D     logTRD_deltaX;1
  KEY: TH1D     logTRD_xCl;1
  KEY: TH1D     logTRD_clY;1
  KEY: TH1D     logTRD_clZ;1
  KEY: TH1D     logTRD_clB;1
  KEY: TH1D     logTRD_clG;1
  KEY: TTree    esdTree;1       Tree with ESD objects
  KEY: TTree    HLTesdTree;1    Tree with HLT ESD objects
  KEY: TH2F     TOFDig_ClusMap;1
  KEY: TH1F     TOFDig_NClus;1
  KEY: TH1F     TOFDig_ClusTime;1
  KEY: TH1F     TOFDig_ClusToT;1
  KEY: TH1F     TOFRec_NClusW;1
  KEY: TH1F     TOFRec_Dist;1
  KEY: TH2F     TOFDig_SigYVsP;1
  KEY: TH2F     TOFDig_SigZVsP;1
  KEY: TH2F     TOFDig_SigYVsPWin;1
  KEY: TH2F     TOFDig_SigZVsPWin;1
```

Shows the list of objects
In the current directory
(like in a file system)

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++");
```

TFile::MakeProject

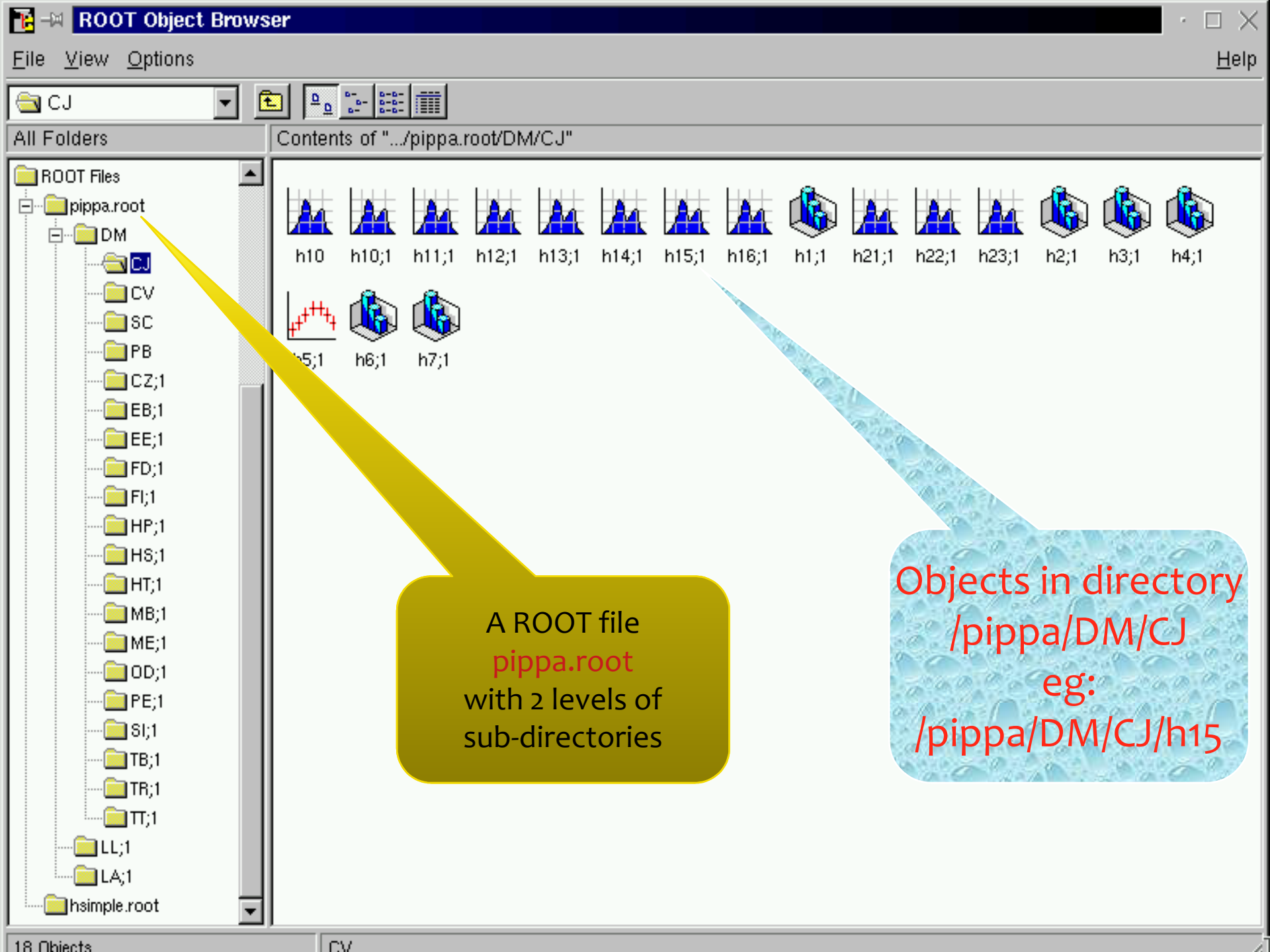
Generate C++ header files
and shared lib for the cla

```
(macbrun2) [253] root
root [0] TFile *falice = TFile::Open("http://ro
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
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]
```

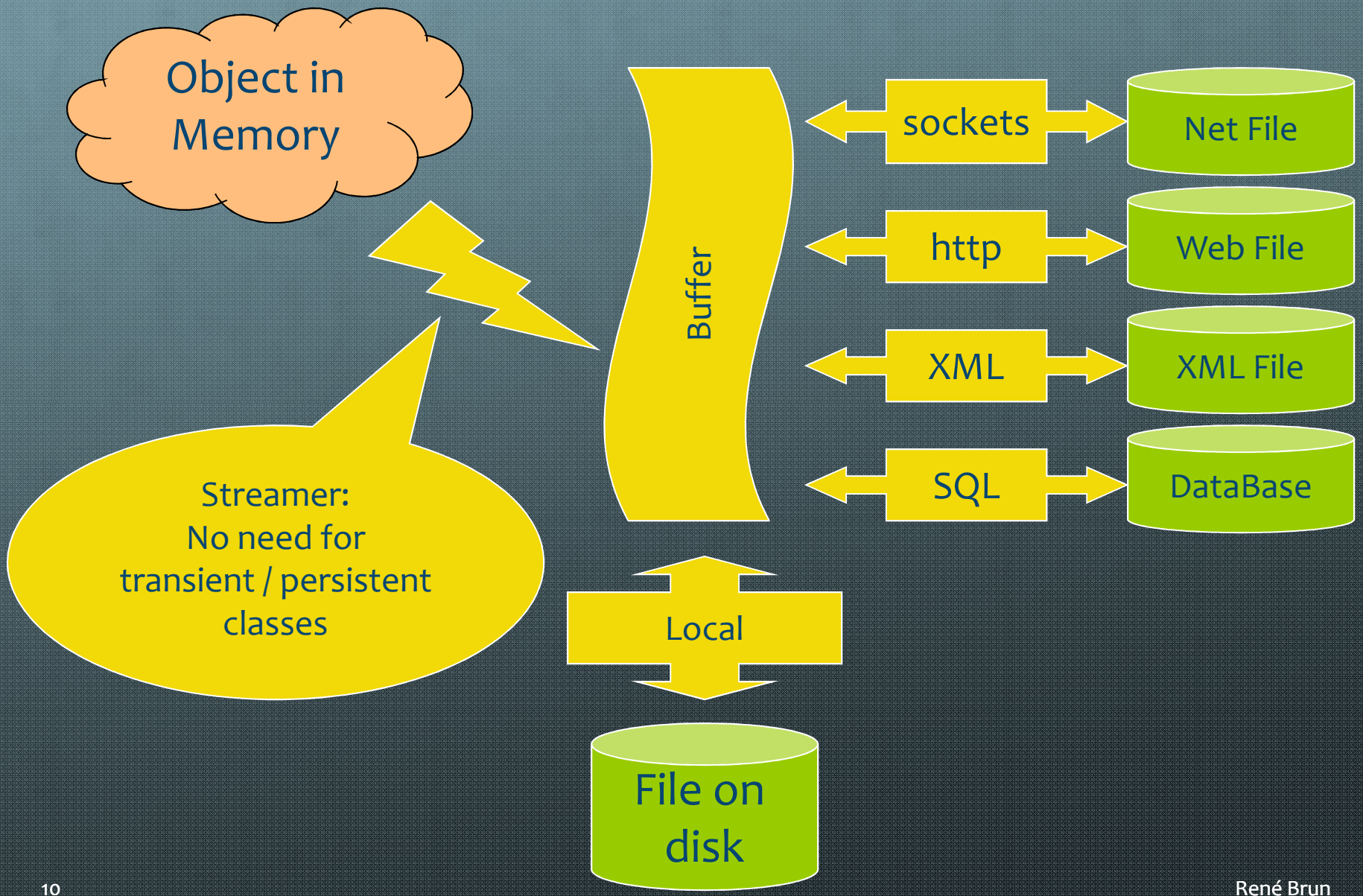
AliESDCaloCluster.h

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.
//////////////////////////////////////////////////////////


#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
   Float_t   fM20;               //2-nd moment along the main eigen axis
   Float_t   fM02;               //2-nd moment along the second eigen axi
   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 t
   Float_t   fDistToBadChannel;  //Distance to nearest bad channel
   TArrayS*  fTracksMatched;     //Index of tracks close to cluster. Firs
   TArrayS*  fLabels;            //list of primaries that generated the c
   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.
};
```

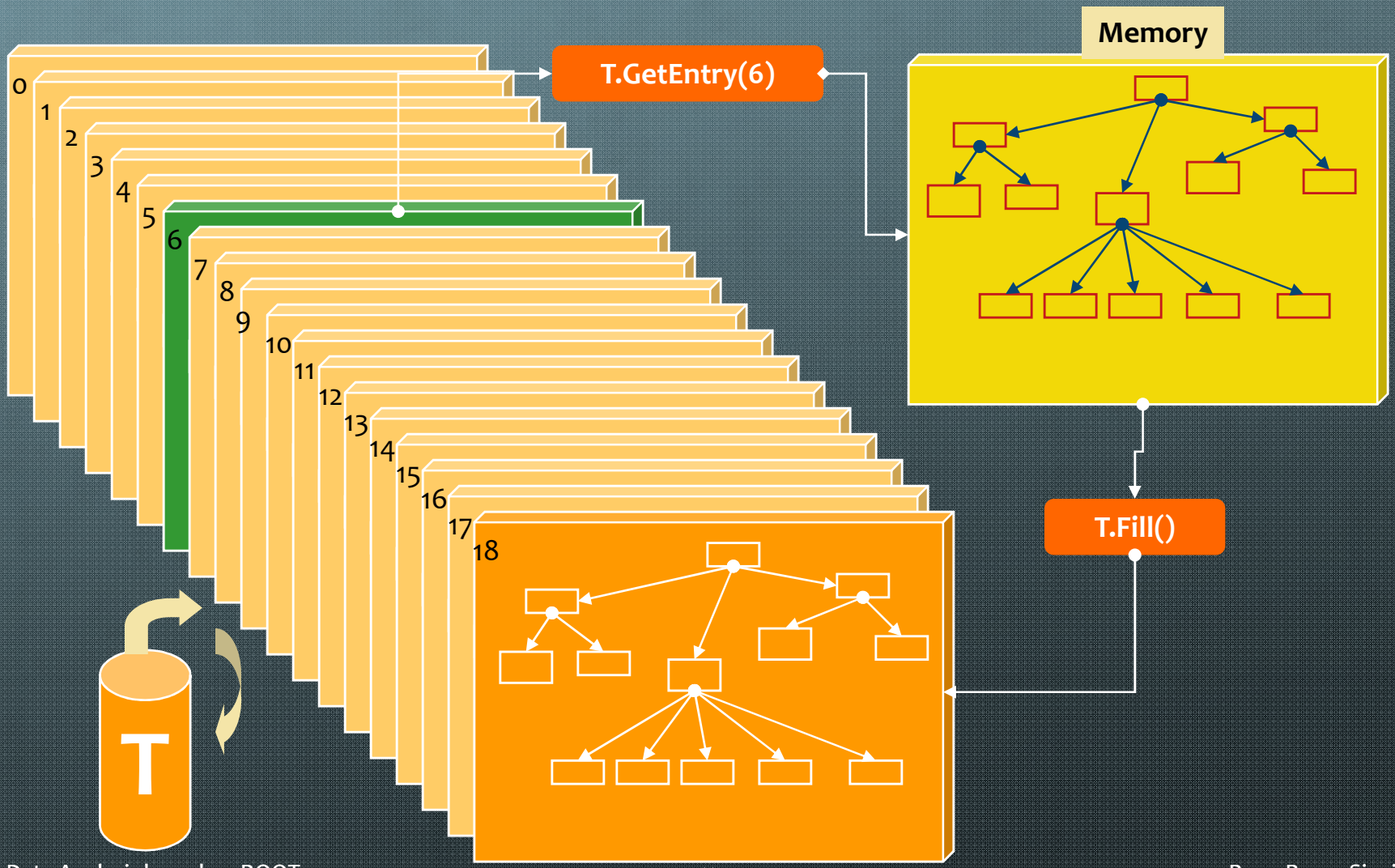

ROOT Object Browser
File View Options Help
CJ
All Folders
Contents of ".../pippa.root/DM/CJ"
ROOT Files
pippa.root
DM
CJ
CV
SC
PB
CZ;1
EB;1
EE;1
FD;1
FI;1
HP;1
HS;1
HT;1
MB;1
ME;1
OD;1
PE;1
SI;1
TB;1
TR;1
TT;1
LL;1
LA;1
hsimple.root
h10
h10;1
h11;1
h12;1
h13;1
h14;1
h15;1
h16;1
h1;1
h21;1
h22;1
h23;1
h2;1
h3;1
h4;1
h5;1
h6;1
h7;1
A ROOT file
pippa.root
with 2 levels of
sub-directories
Objects in directory
/pippa/DM/CJ
eg:
/pippa/DM/CJ/h15
18 Objects
CV

I/O


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

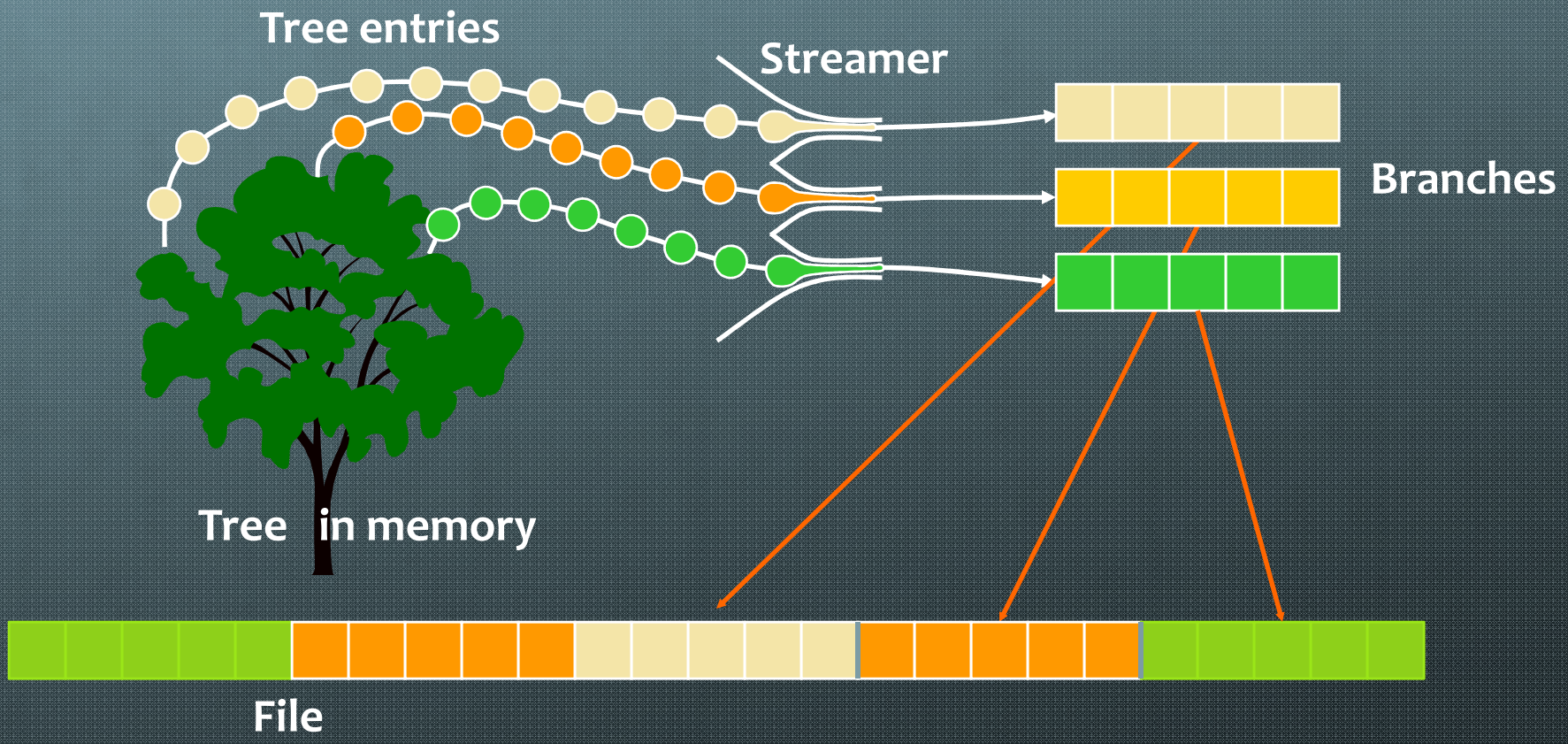
Memory <--> Tree

Each Node is a branch in the Tree

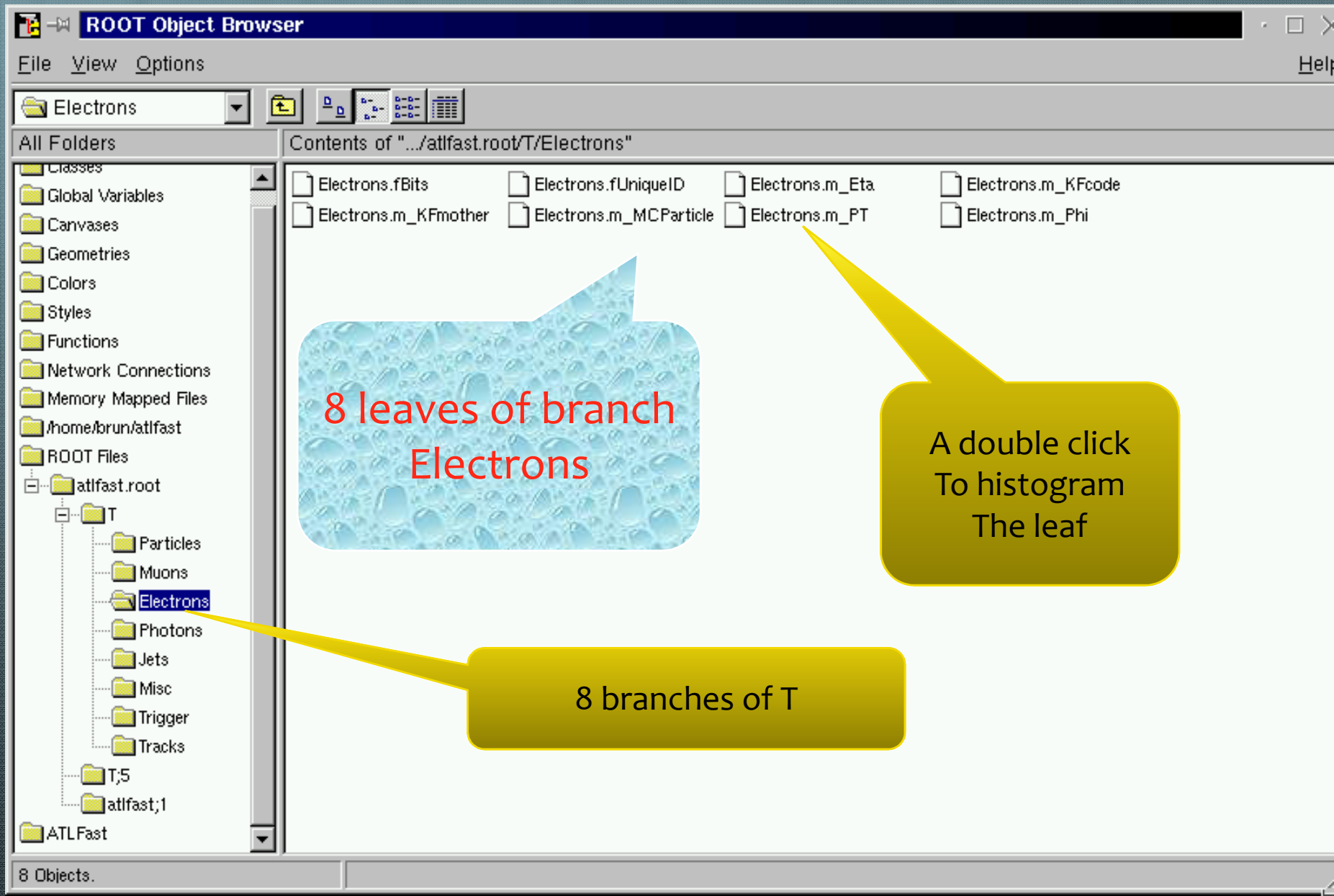


ROOT I/O -- *Split/Cluster*

Tree version



Browsing a TTree with TBrowser



Trees Split Mode

Object-wise

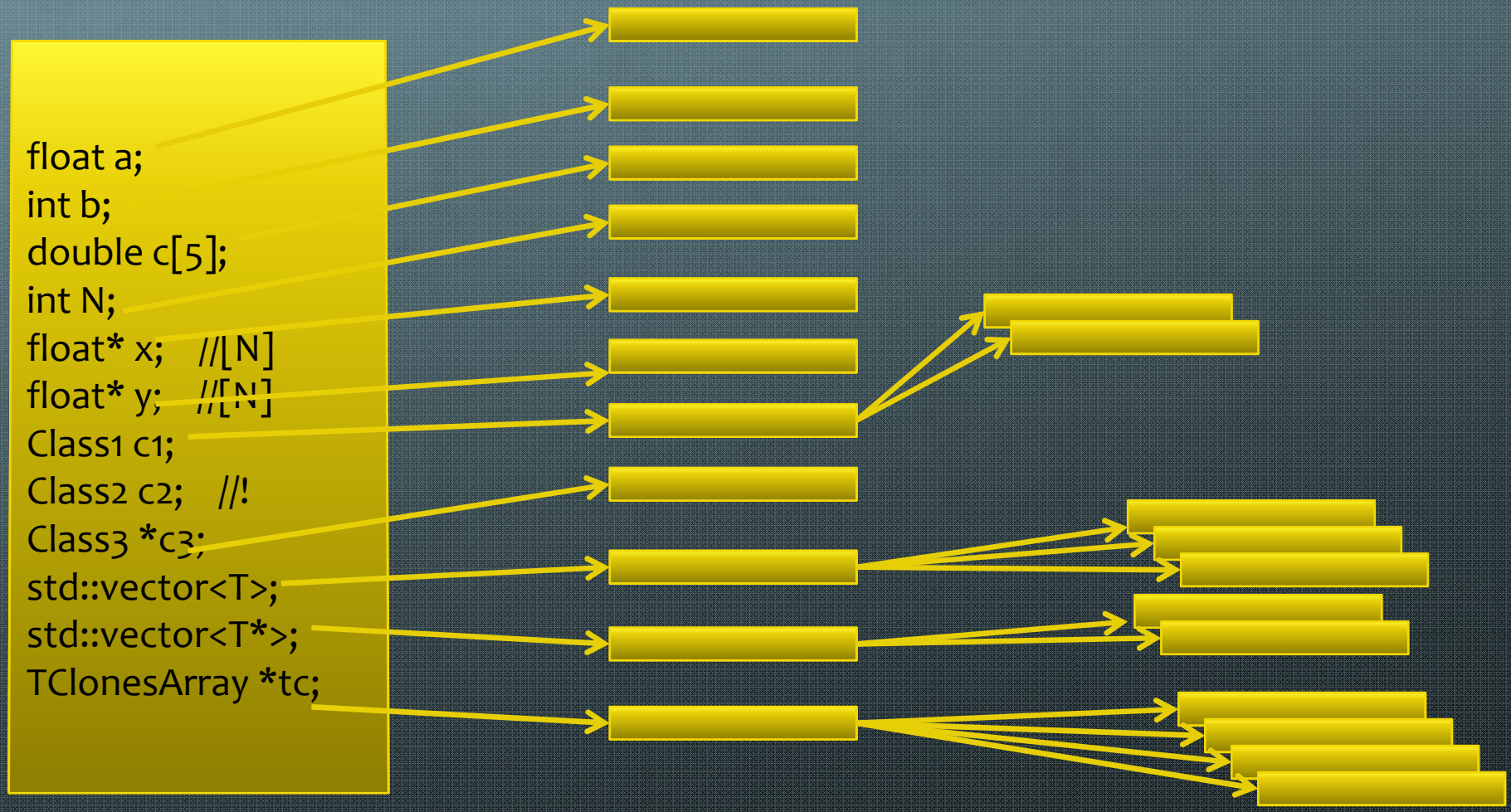
Member-wise

streaming

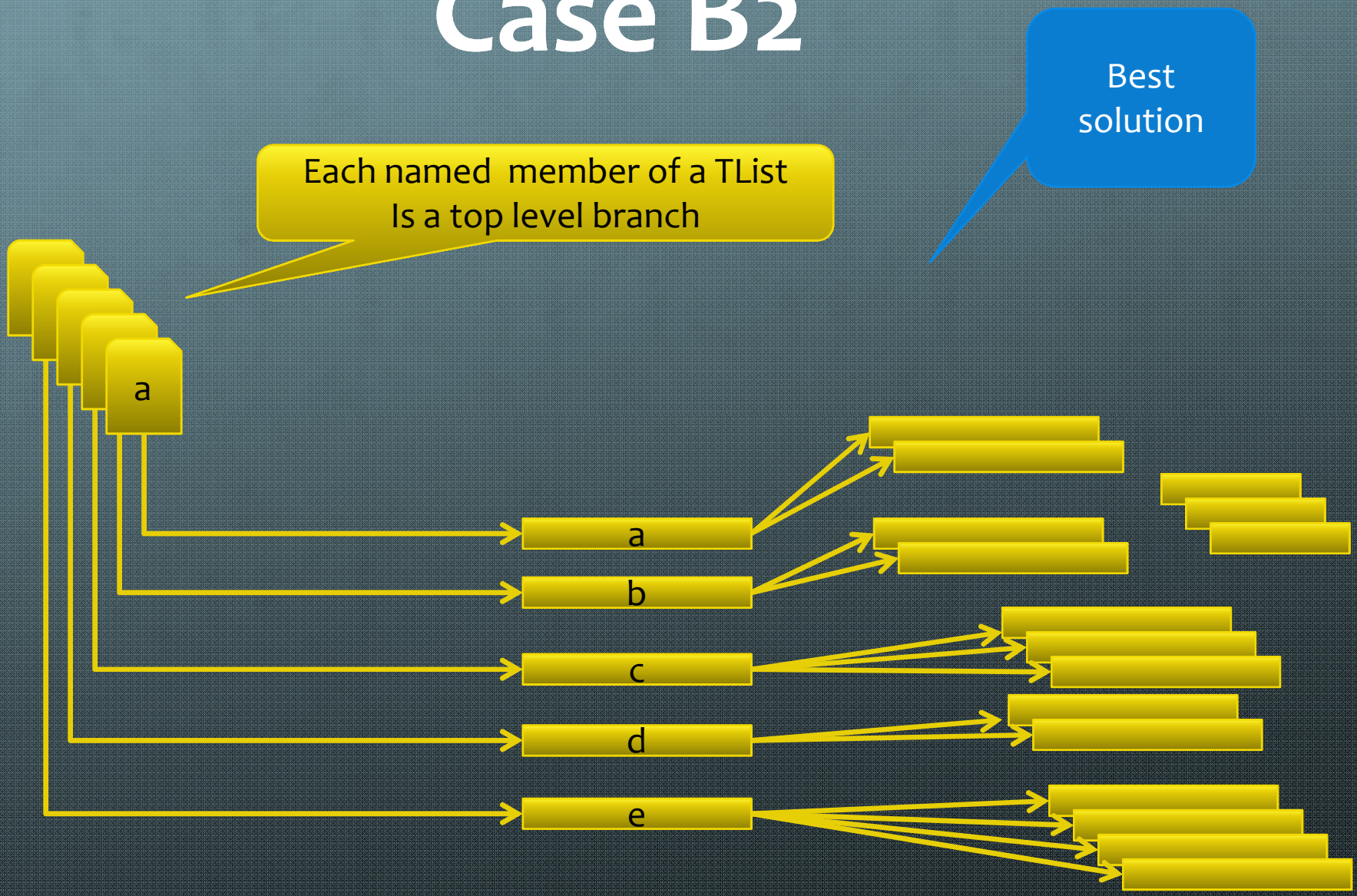
Creating branches

- B1: Branches can be created automatically from the natural hierarchy in the top level object.
- B2: Branches can be created automatically from a dynamic list of top level objects (recommended).
- B3: Branches can be created manually one by one specifying a top level object per top branch.
- B4: Sub-branches are automatically created for STL collections and TClonesArray.

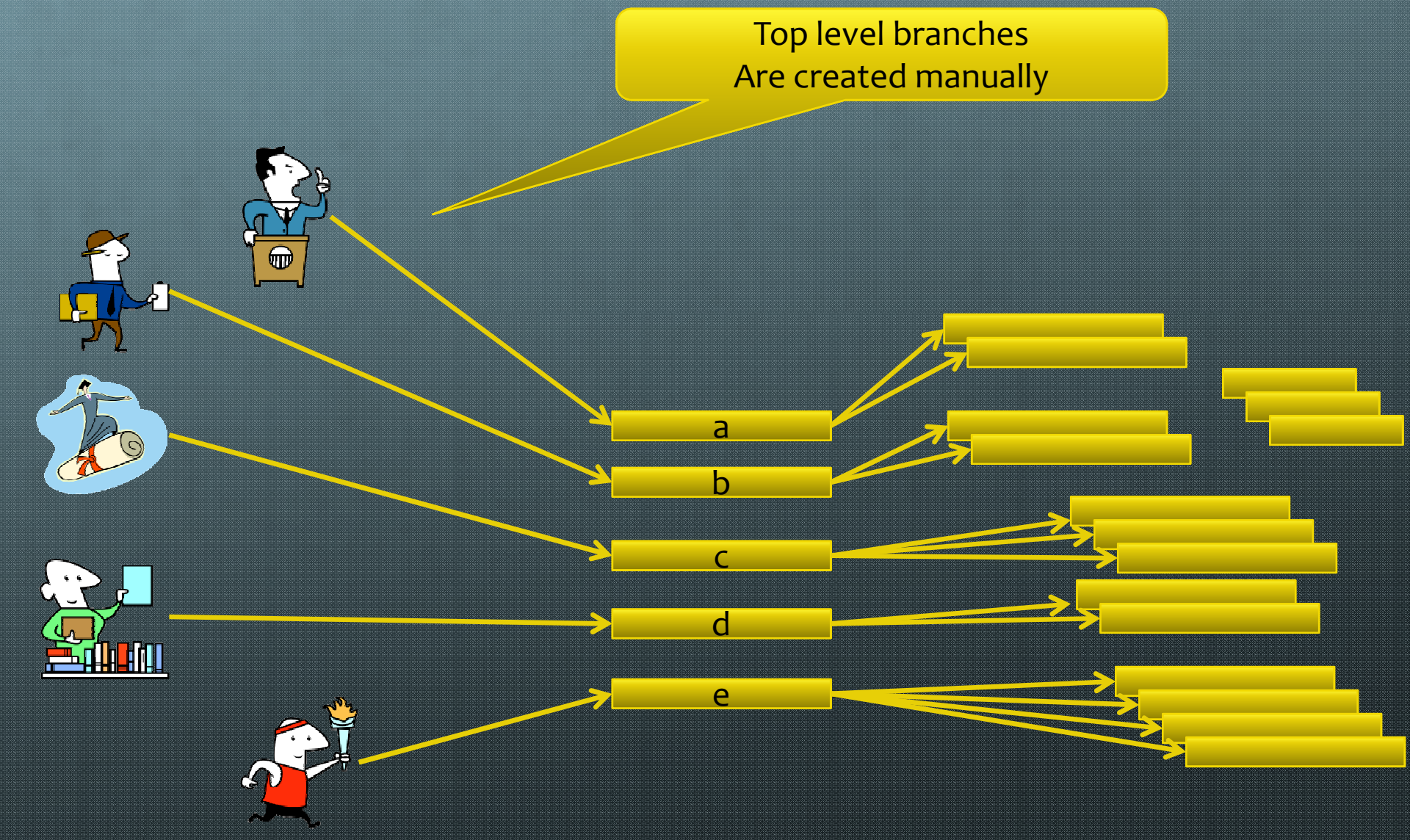
Case B1


float a;
int b;
double c[5];
int N;
float* x; //[N]
float* y; //[N]
Class1 c1;
Class2 c2; //!
Class3 *c3;
std::vector<T>;
std::vector<T*>;
TClonesArray *tc;

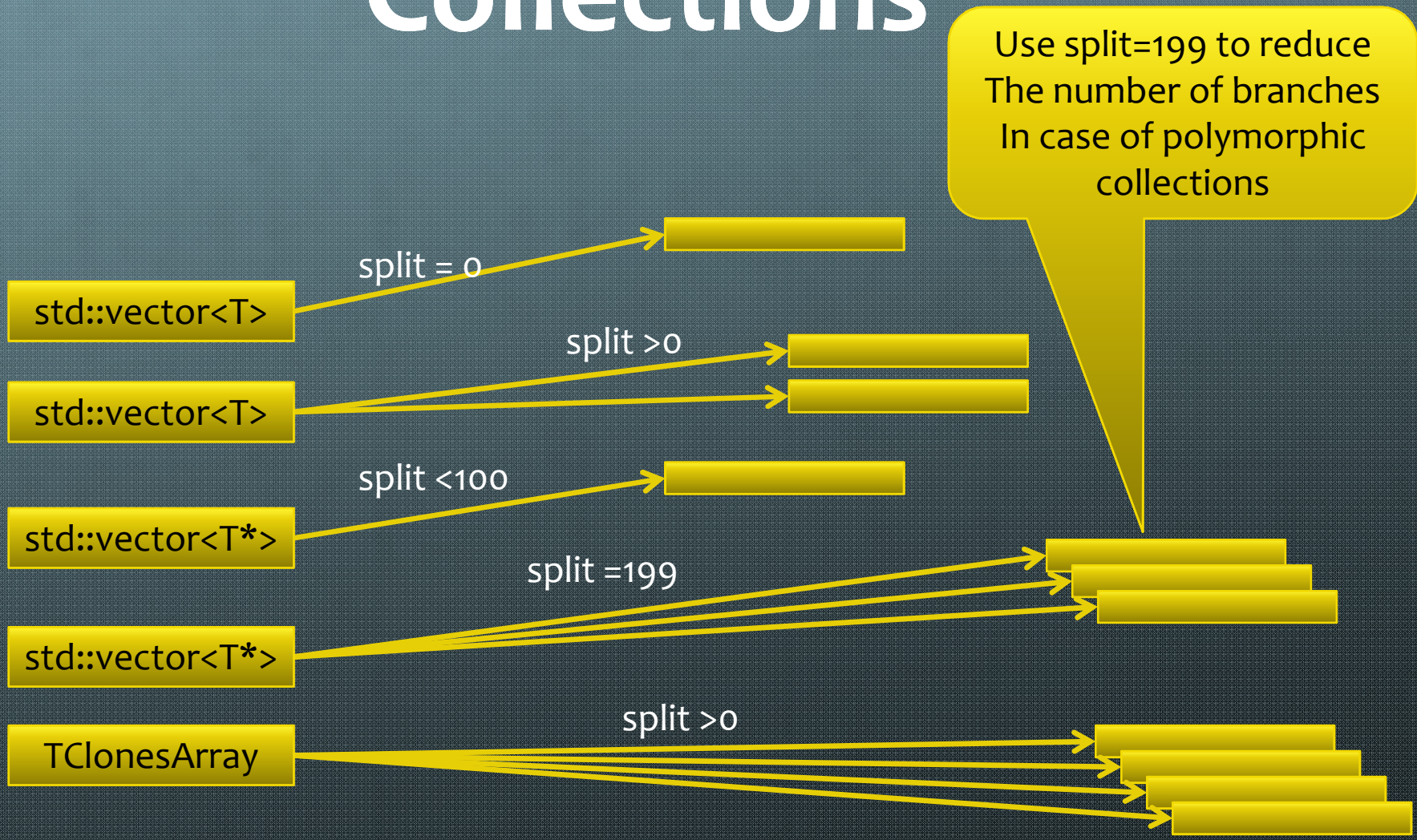
Case B2


Best solution
Each named member of a TList
Is a top level branch
a
a
b
c
d
e

Case B3



Case B4 Collections



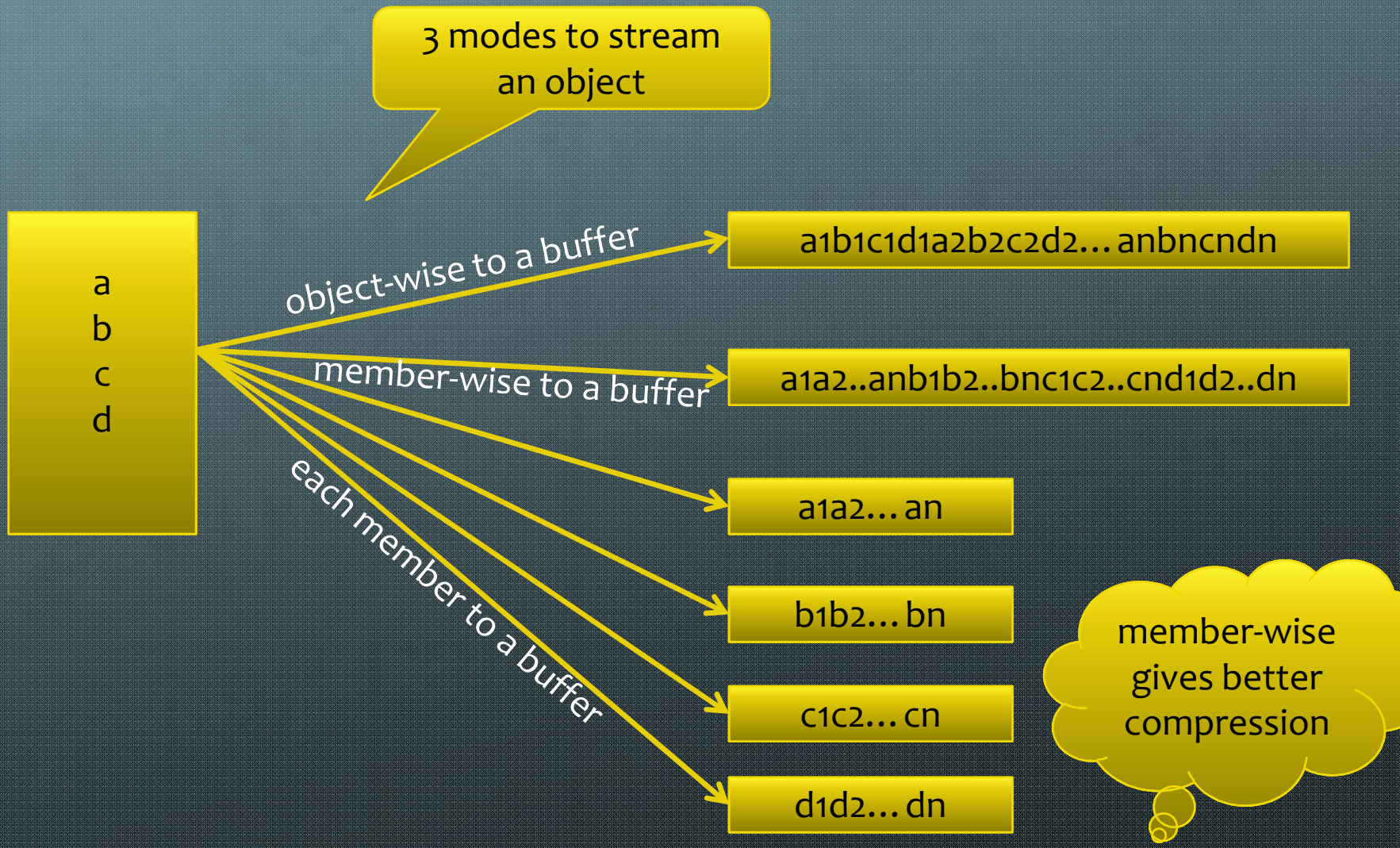
Collections are not equal

Time to read in seconds	comp 0	comp 1
TClonesArray(TObjHit) level=99	0.14	0.45
TClonesArray(TObjHit) level= 0	0.17	0.63
vector<THit> level=99	0.32	0.52
vector<THit> level= 0	0.48	0.87
vector<THit> level= 0 MW	0.23	0.65
list<THit> level=99	0.52	0.64
list<THit> level= 0	0.73	0.95
list<THit> level= 0 MW	0.49	0.62
deque<THit> level=99	0.31	0.58
deque<THit> level= 0	0.55	0.93
deque<THit> level= 0 MW	0.36	0.62
set<THit> level=99	0.46	0.81
set<THit> level= 0	0.74	1.06
set<THit> level= 0 MW	0.44	0.69
multiset<THit> level=99	0.46	0.84
multiset<THit> level= 0	0.72	1.06
multiset<THit> level= 0 MW	0.45	0.70
map<THit> level=99	0.45	0.76
map<THit> level= 0	0.97	1.12
map<THit> level= 0 MW	0.84	1.14

Time to read in seconds	comp 0	comp 1
multimap<THit> level=99	0.45	0.78
multimap<THit> level= 0	0.89	1.11
multimap<THit> level= 0 MW	0.79	1.12
vector<THit*> level=25599	0.34	0.61
list<THit*> level=25599	0.41	0.81
deque<THit*> level=25599	0.49	0.62
set<THit*> level=25599	0.59	0.76
multiset<THit*> level=25599	0.49	0.73
map<THit*> level=99	1.41	1.55
multimap<THit*> level=99	1.15	1.54
vector<THit*> level=99 (NS)	1.17	1.38
list<THit*> level=99 (NS)	1.33	1.48
deque<THit*> level=99 (NS)	1.36	1.39
set<THit*> level=99 (NS)	1.02	1.48
multiset<THit*> level=99 (NS)	1.28	1.45

For more details run $ROOTSYS/test/bench

ObjectWise/MemberWise Streaming



More branches is better

- Better compression
- Faster when reading (in particular a subset of branches)
- Good for parallelism (for the future)

10000 is may be too much :☺

Split mode	File size (MBytes)	Comp factor	Write time (seconds)	Read time (seconds)
0 1 branch	177.45	2.23	38.7	12.6
1 20 branches	174.7	2.26	37.9	12.0
99 56 branches	144.9	2.53	38.3	10.8

ROOT IO

Bottlenecks and Solutions

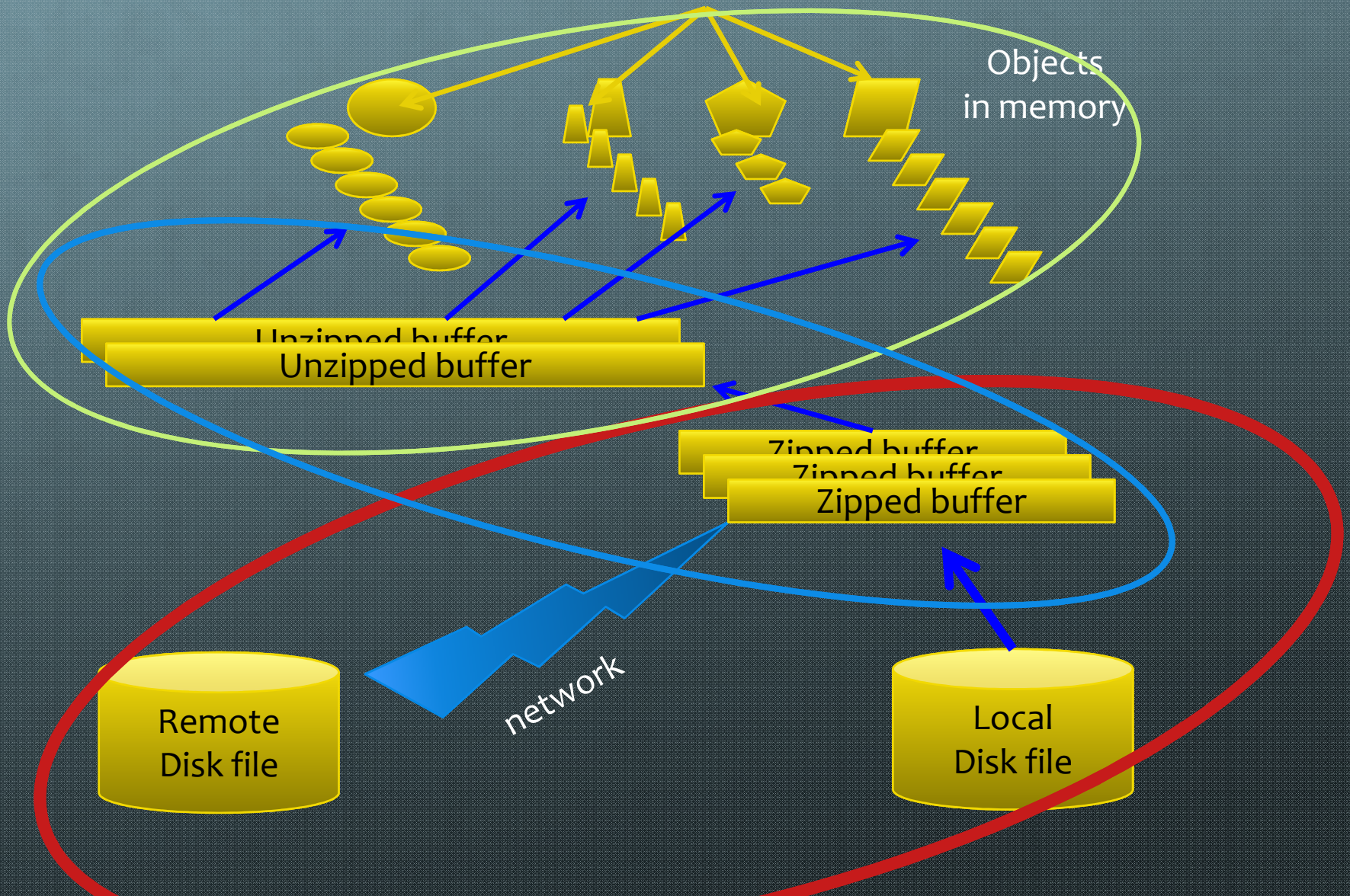
Buffering effects

- Branch buffers are not full at the same time.
- A branch containing one integer/event and with a buffer size of 32Kbytes will be written to disk every 8000 events, while a branch containing a non-split collection may be written at each event.
- This may give serious problems when reading if the file is not read sequentially.

problems

- In the past few months we have analyzed Trees from Alice, Atlas, CMS and LHCb and found problems in all cases.
- Some of these problems have solutions with old versions of ROOT, eg
  - Using effectively the TreeCache
  - Optimizing basket buffers
- ROOT version 5.25/04 contains several enhancements helping to improve size and performance

Important factors



What is the TreeCache

- It groups into one buffer all blocks from the used branches.
- The blocks are sorted in ascending order and consecutive blocks merged such that the file is read sequentially.
- It reduces typically by a factor 10000 the number of transactions with the disk and in particular the network with servers like xrootd or dCache.
- The small blocks in the buffer can be unzipped in parallel on a multi-core machine.
- The typical size of the TreeCache is 10 Mbytes, but higher values will always give better results. If you have no memory problem, set large values like 200 Mbytes.

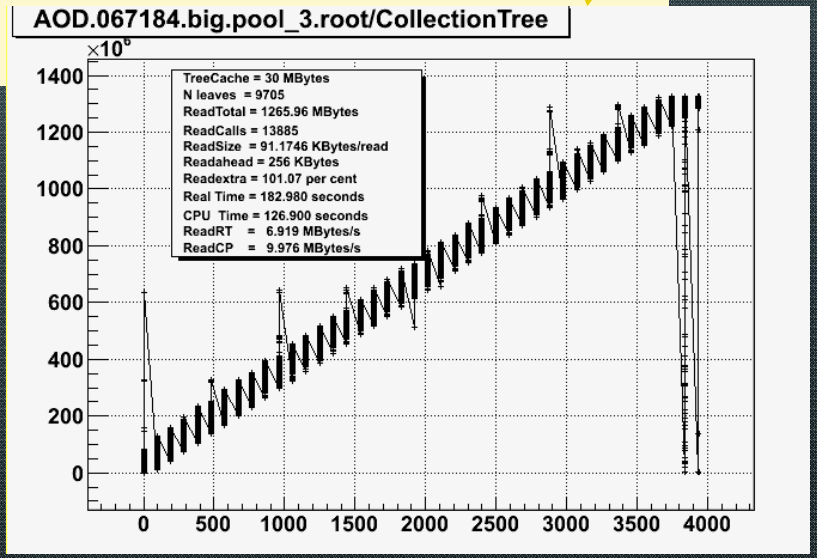
TreeCache: new interface

- Facts: Most users did not know if they were using or not the TreeCache.
- We decided to implement a simpler interface from TTree itself (no need to know about the class TTreeCache anymore).
- Because some users claimed to use the TreeCache and the results clearly showing the contrary, we decided to implement a new IO monitoring class TTreePerfStats.

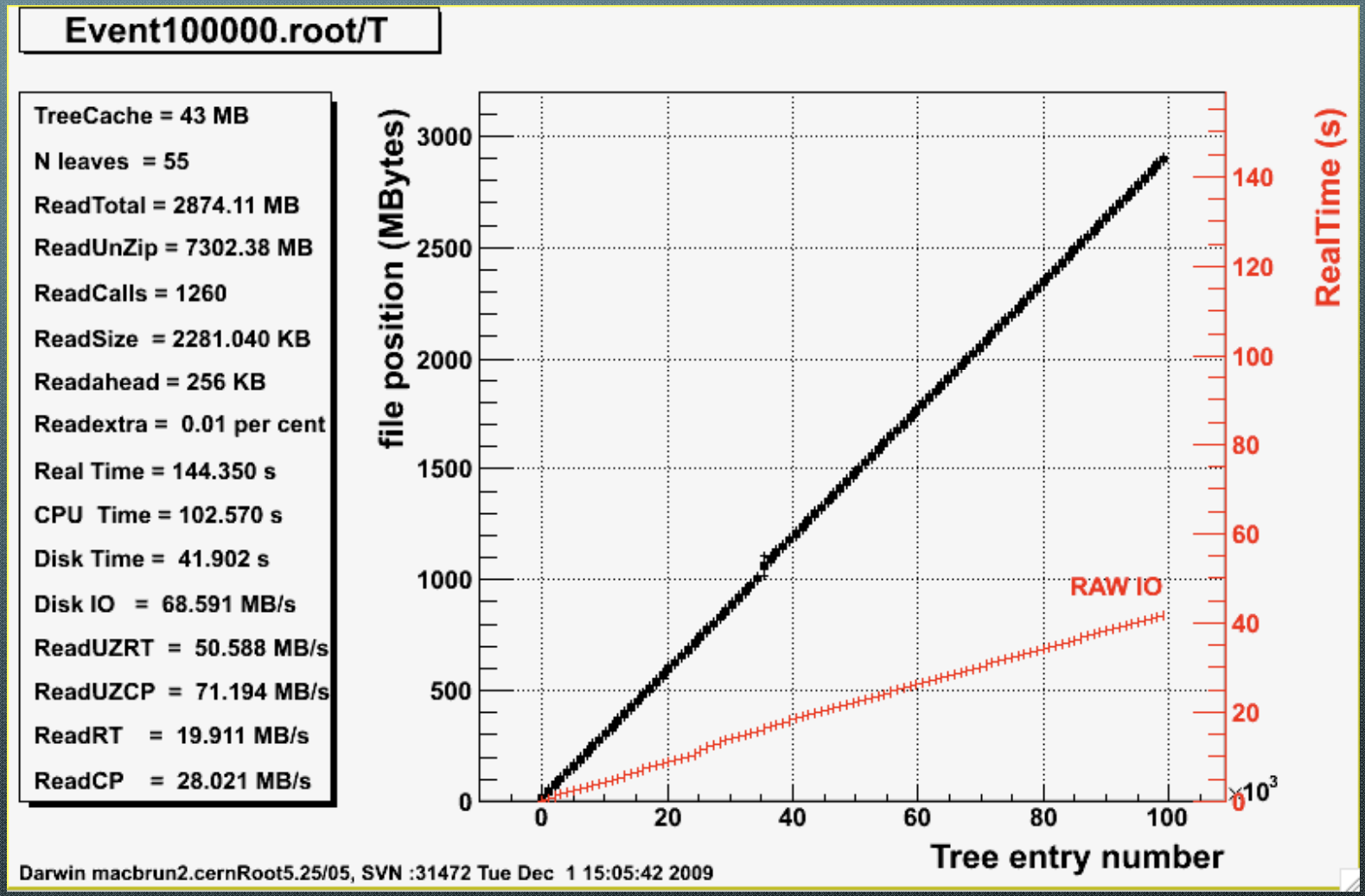
Use TTreePerfStats

```
void taodr(Int_t cachesize=10000000) {
   gSystem->Load("aod/aod");  //shared lib generated with TFile::MakeProject
   TFile *f = TFile::Open("AOD.067184.big.pool.root");
   TTree *T = (TTree*)f->Get("CollectionTree");
   Long64_t nentries = T->GetEntries();
   T->SetCacheSize(cachesize);
   T->AddBranchToCache("*",kTRUE);

   TTreePerfStats ps("ioperf",T);

   for (Long64_t i=0;i<nentries;i++) {
      T->GetEntry(i);
   }
   ps.SaveAs("aodperf.root");
}
```

Root > TFile f("aodperf.root")
Root > ioperf.Draw()



Performance with standard ROOT example



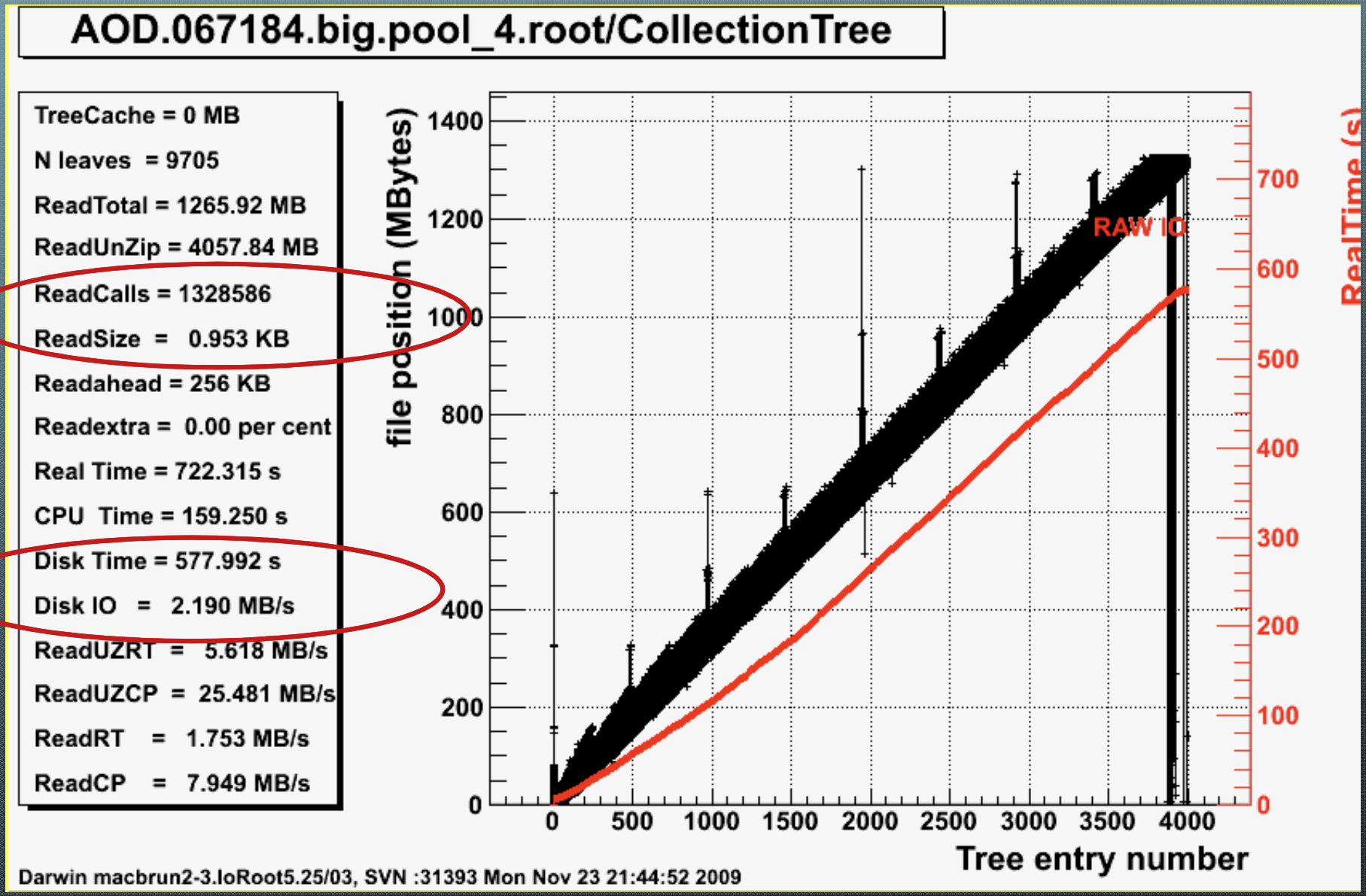
Test conditions

- Because both the TreeCache and Readahead are designed to minimize the difference RealTime-CpuTime, care has been taken to run the tests with “cold” files, making sure that system buffers were dropped before running a new test.
- Note that increasing the TreeCache size reduces also the CpuTime.
- Note that running OptimizeBaskets also reduces substantially the CpuTime because the number of baskets is in general reduced by several factors.

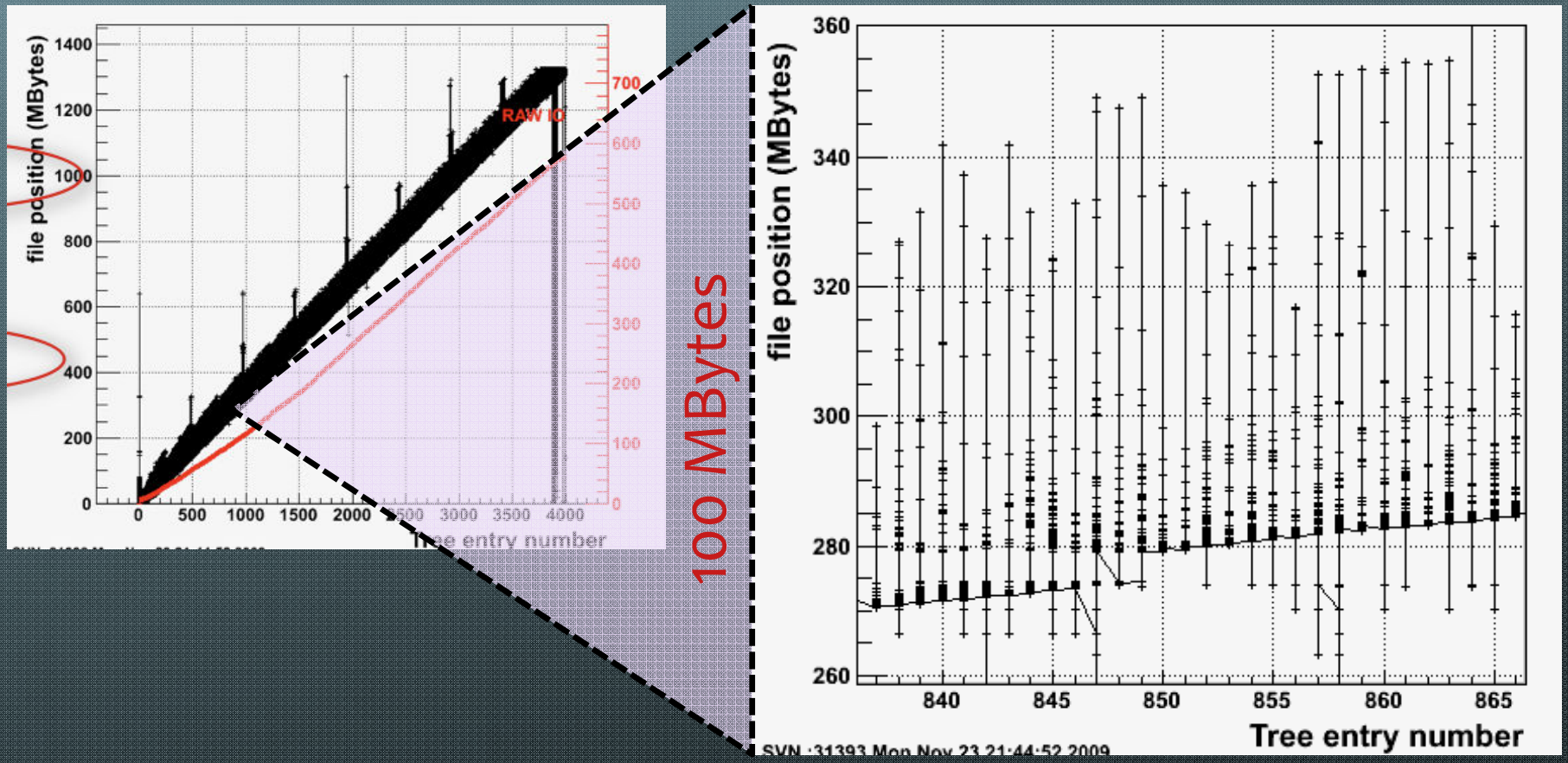
Test conditions 2

- Using one of the AOD files the class headers have been generated automatically via TTree::MakeProject.
- The corresponding shared library is linked such that the same object model is used in my tests and in Atlas persistent model.
- The tests read systematically all entries in all branches. Separate tests have been run to check that the optimal performance is still obtained when reading either a subset of branches, a subset of entries or both. This is an important remark because we have seen that sometimes proposed solutions are good when reading everything and very bad in the other mentioned use cases that are typical of the physics analysis scenarios.

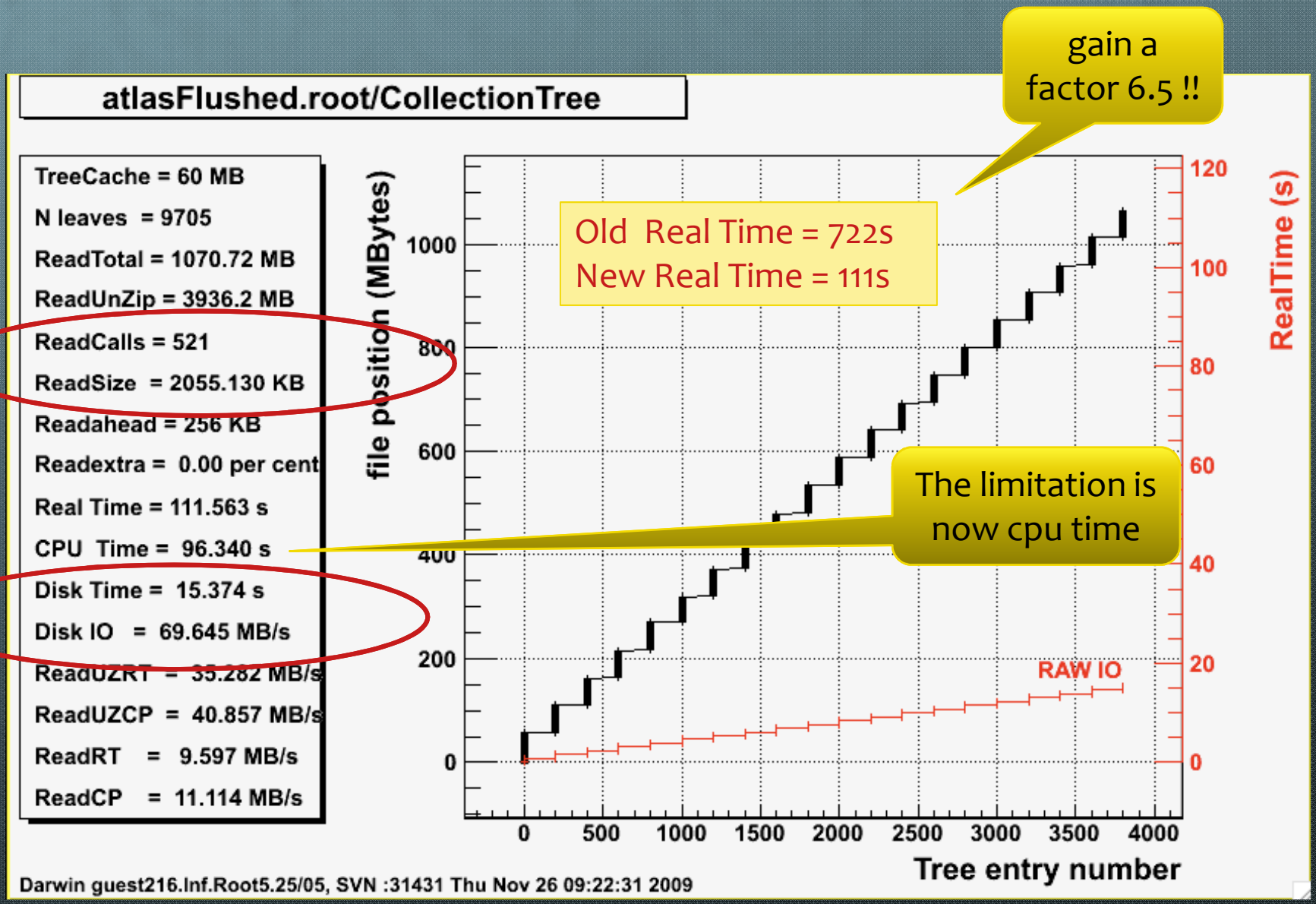
See Doctor


AOD.067184.big.pool_4.root/CollectionTree
TreeCache = 0 MB
N leaves = 9705
ReadTotal = 1265.92 MB
ReadUnZip = 4057.84 MB
ReadCalls = 1328586
ReadSize = 0.953 KB
Readahead = 256 KB
Readextra = 0.00 per cent
Real Time = 722.315 s
CPU Time = 159.250 s
Disk Time = 577.992 s
Disk IO = 2.190 MB/s
ReadUZRT = 5.618 MB/s
ReadUZCP = 25.481 MB/s
ReadRT = 1.753 MB/s
ReadCP = 7.949 MB/s
file position (MBytes)
RealTime (s)
RAW IO
Tree entry number
Darwin macbrun2-3.loRoot5.25/03, SVN :31393 Mon Nov 23 21:44:52 2009

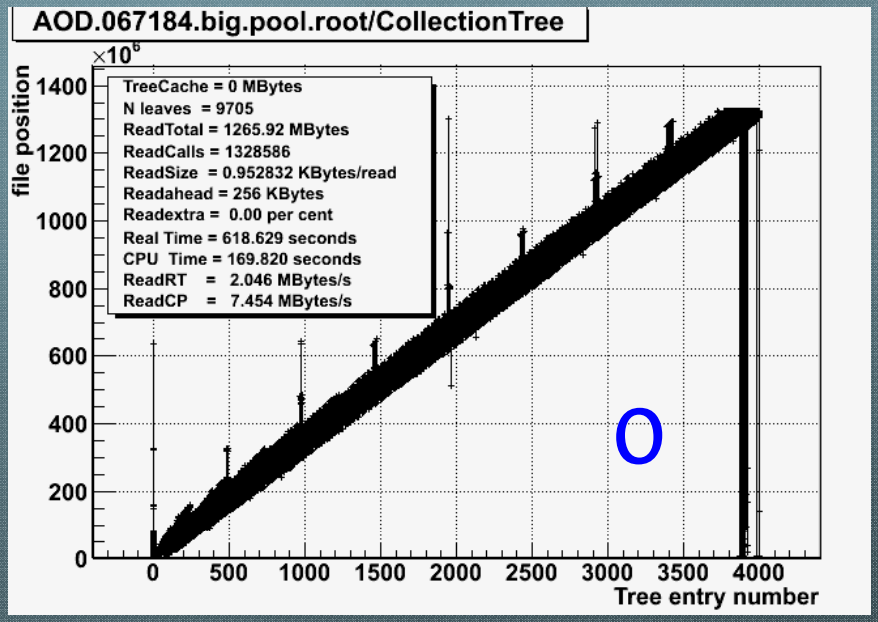
Overlapping reads

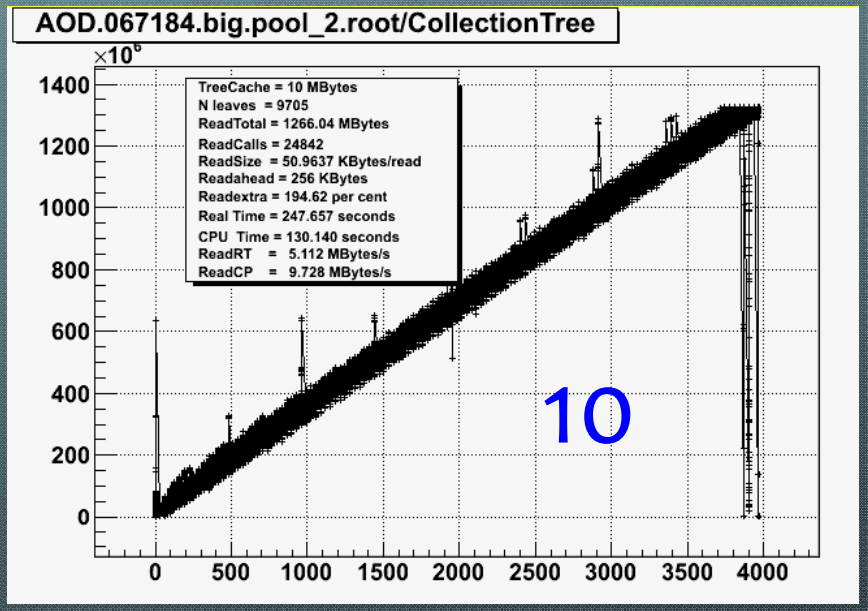


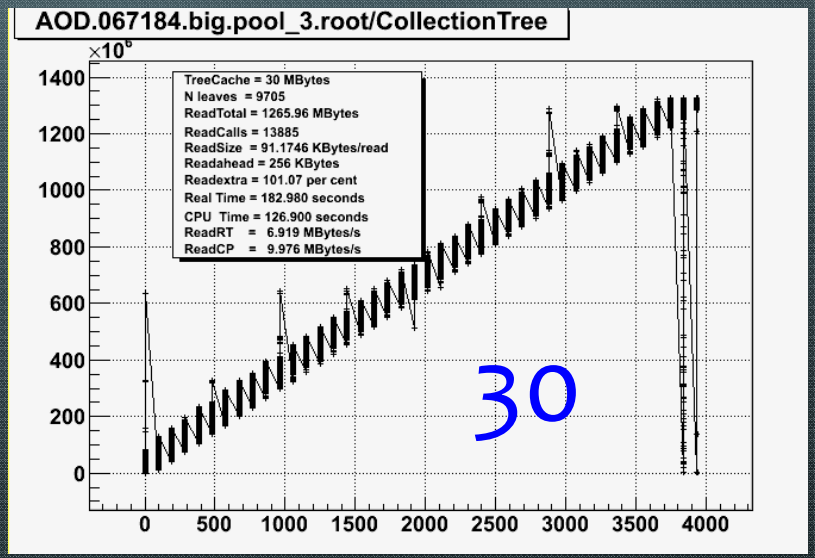
After doctor

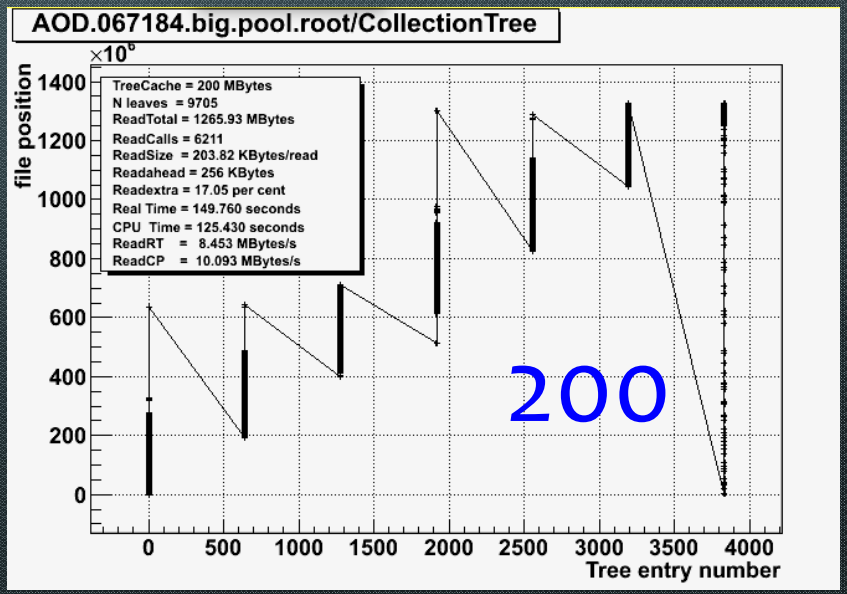


TreeCache size impact









TreeCache results table

Original Atlas file (1266MB), 9705 branches split=99

Cache size (MB)	readcalls	RT pcbrun4 (s)	CP pcbrun4 (s)	RT macbrun (s)	CP macbrun (s)
0	1328586	734.6	270.5	618.6	169.8
LAN 1ms 0	1328586	734.6+1300	270.5	618.6+1300	169.8
10	24842	298.5	228.5	229.7	130.1
30	13885	272.1	215.9	183.0	126.9
200	6211	217.2	191.5	149.8	125.4

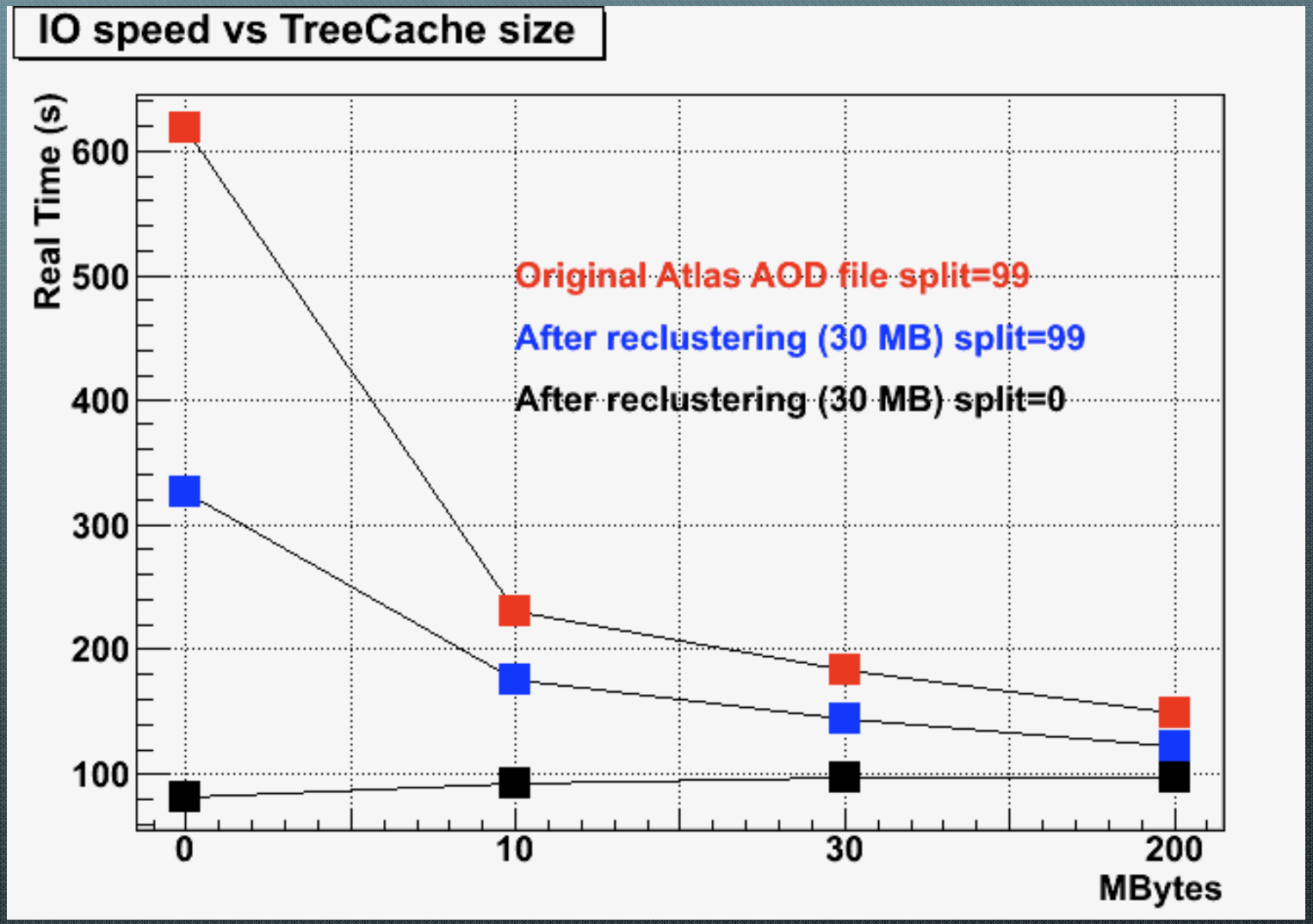
Reclust: OptimizeBaskets 30 MB (1147 MB), 203 branches split=0

Cache size (MB)	readcalls	RT pcbrun4 (s)	CP pcbrun4 (s)	RT macbrun (s)	CP macbrun (s)
0	15869	148.1	141.4	81.6	80.7
LAN 1ms 0	15869	148.1 + 16	141.4	81.6 + 16	80.7
10	714	157.9	142.4	93.4	82.5
30	600	165.7	148.8	97.0	82.5
200	552	154.0	137.6	98.1	82.0

Reclust: OptimizeBaskets 30 MB (1086 MB), 9705 branches split=99

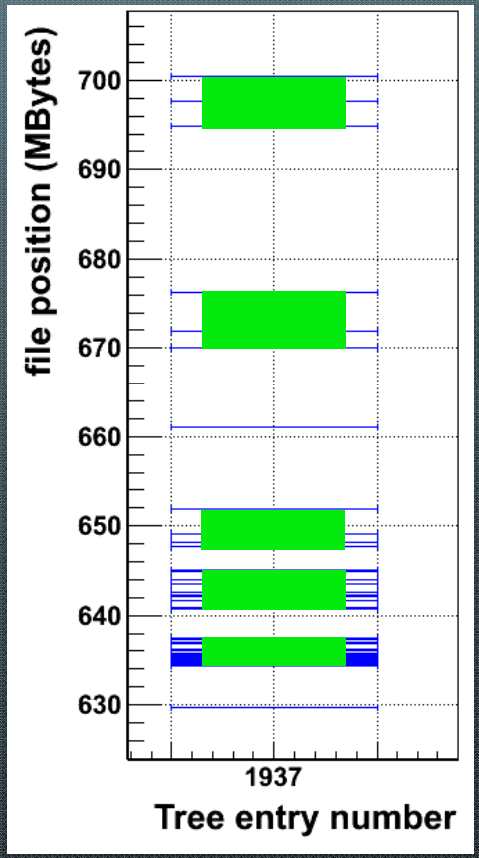
Cache size (MB)	readcalls	RT pcbrun4 (s)	CP pcbrun4 (s)	RT macbrun (s)	CP macbrun (s)
0	515350	381.8	216.3	326.2	127.0
LAN 1ms 0	515350	381.8 + 515	216.3	326.2 +515	127.0
10	15595	234.0	185.6	175.0	106.2
30	8717	216.5	182.6	144.4	104.5
200	2096	182.5	163.3	122.3	103.4

TreeCache results graph

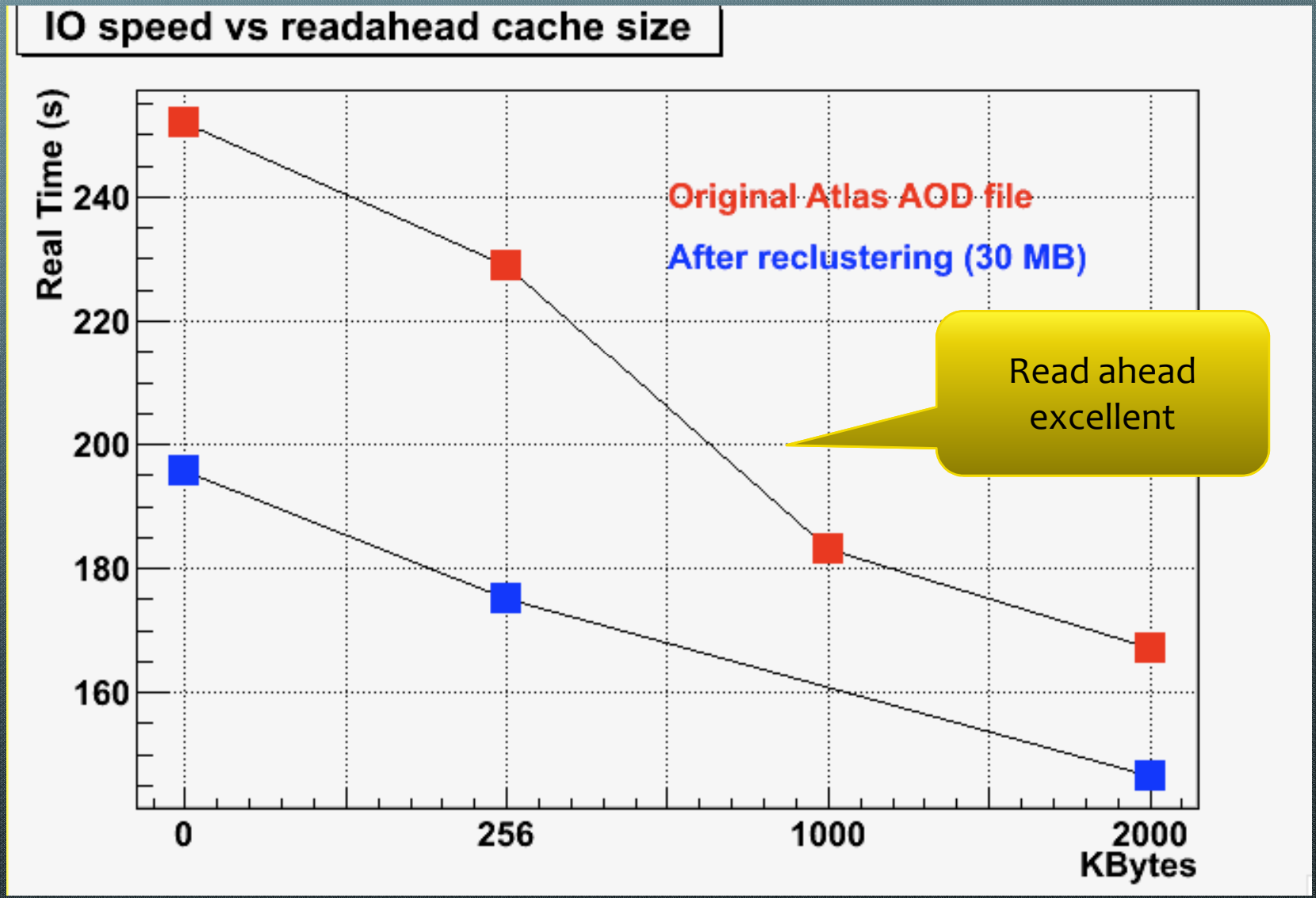


What is the readahead cache

- The readahead cache will read all non consecutive blocks that are in the range of the cache.
- It minimizes the number of disk access. This operation could in principle be done by the OS, but the fact is that the OS parameters are not tuned for many small reads, in particular when many jobs read concurrently from the same disk.
- When using large values for the TreeCache or when the baskets are well sorted by entry, the readahead cache is not necessary.
- Typical (default value) is 256 Kbytes, although 2 Mbytes seems to give better results on Atlas files, but not with CMS or Alice.



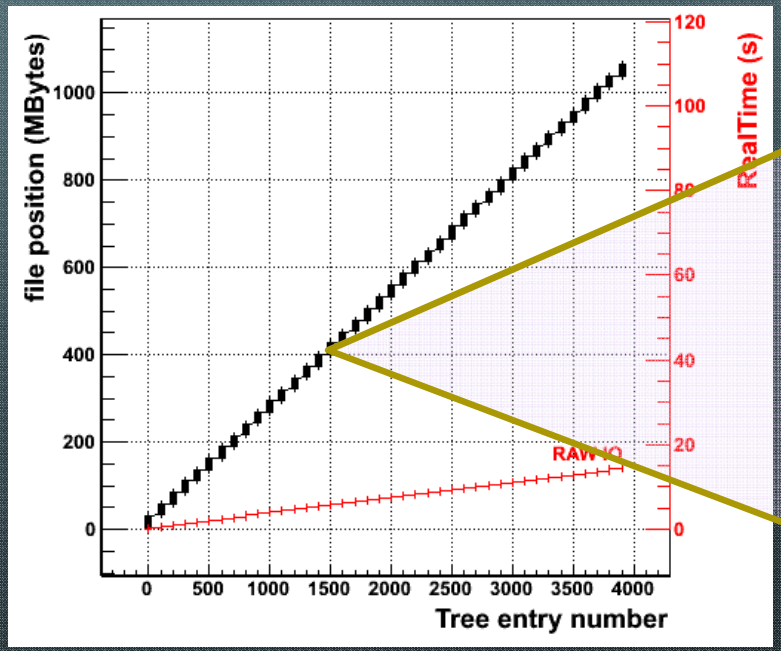
Readahead
reading all branches, all entries

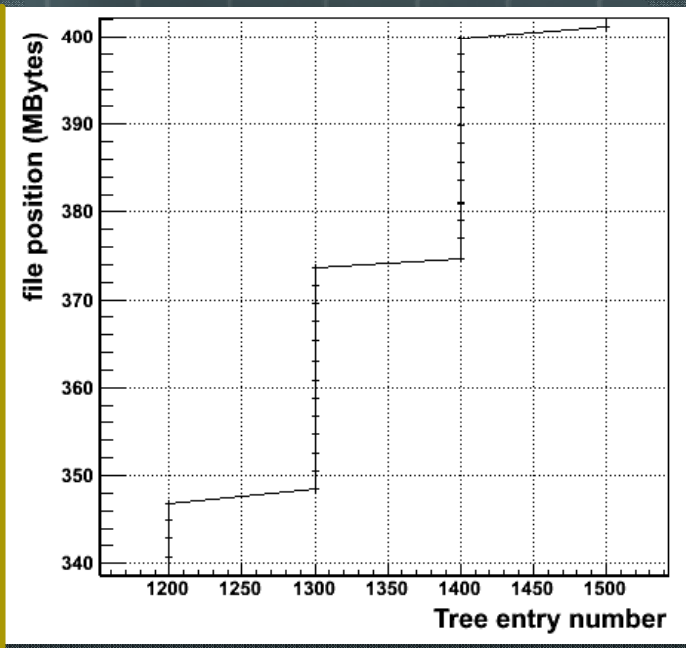


OptimizeBaskets, AutoFlush

New in v5.25/04!

- Solution, enabled by default:
  - Tweak basket size!
  - Flush baskets at regular intervals!





OptimizeBaskets

- Facts: Users do not tune the branch buffer size
- Effect: branches for the same event are scattered in the file.
- TTree::OptimizeBaskets is a new function that will optimize the buffer sizes taking into account the population in each branch.
- You can call this function on an existing read only Tree file to see the diagnostics.

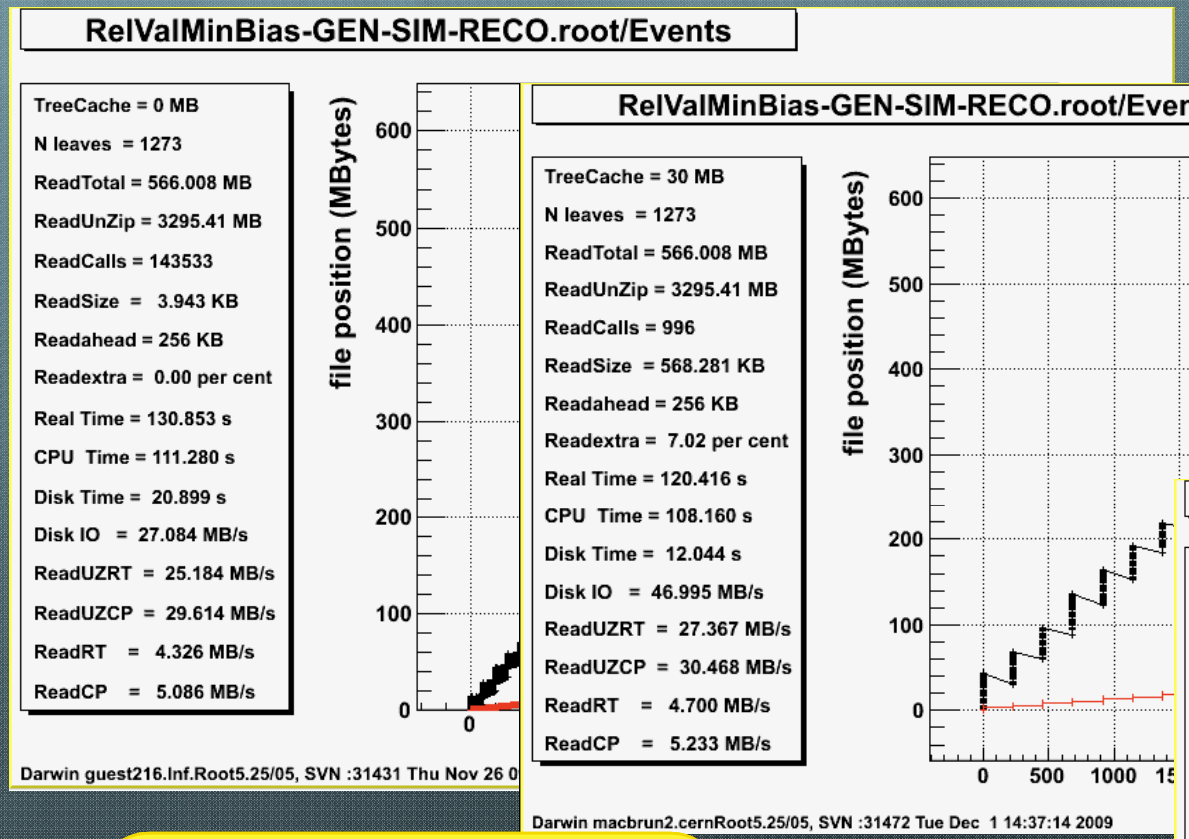
FlushBaskets

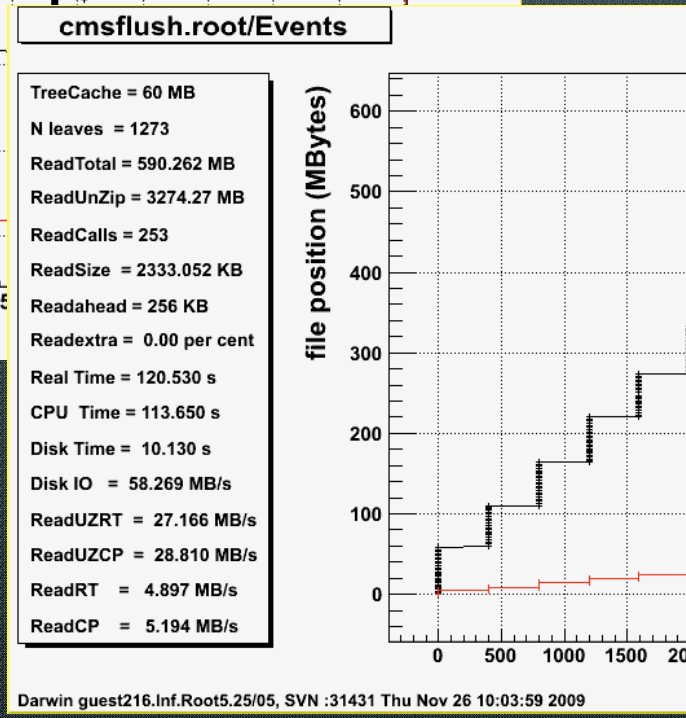
- TTree::FlushBaskets was introduced in 5.22 but called only once at the end of the filling process to disconnect the buffers from the tree header.
- In version 5.25/04 this function is called automatically when a reasonable amount of data (default is 30 Mbytes) has been written to the file.
- The frequency to call TTree::FlushBaskets can be changed by calling TTree::SetAutoFlush.
- The first time that FlushBaskets is called, we also call OptimizeBaskets.

FlushBaskets 2

- The frequency at which FlushBaskets is called is saved in the Tree (new member fAutoFlush).
- This very important parameter is used when reading to compute the best value for the TreeCache.
- The TreeCache is set to a multiple of fAutoFlush.
- Thanks to FlushBaskets there is no backward seeks on the file for files written with 5.25/04. This makes a dramatic improvement in the raw disk IO speed.

Similar pattern with CMS files

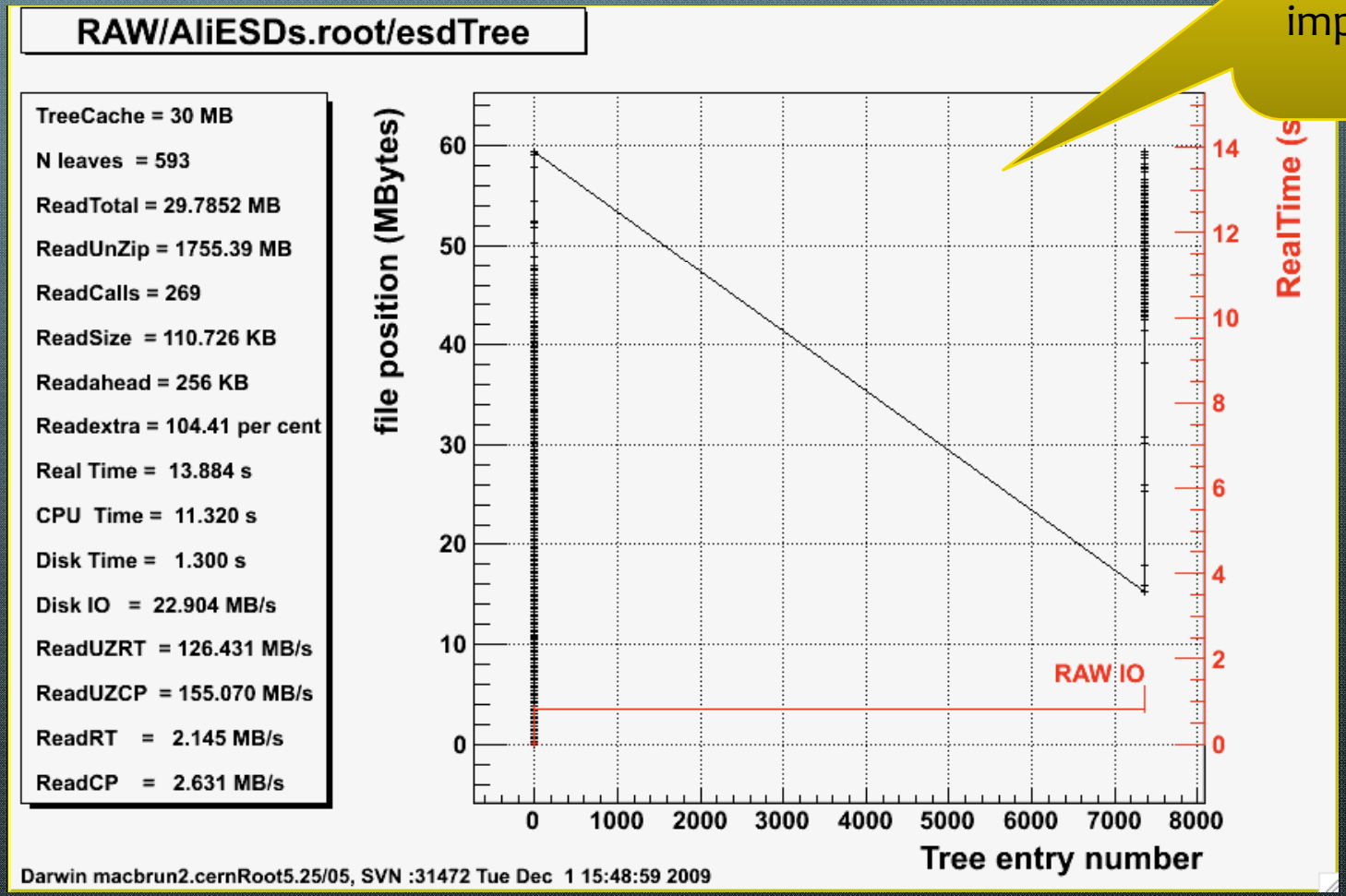




CMS : mainly CPU problem due to a complex object model

Alice files

Only small files used in the test. Performance improvement with 5.26



LHCb files

Performance must be very poor !!

- One LHCb file contains about 45 Trees !!
- Each Tree should be a top level branch in the main Tree
  - To take advantage of the TreeCache
  - To take advantage of FlushBaskets

TreeCache

Interface for
Different use patterns

Use Case 1
Reading all branches

```
void taodr(Int_t cachesize=10000000) {
   TFile *f = TFile::Open("AOD.067184.big.pool.root");
   TTree *T = (TTree*)f->Get("CollectionTree");
   Long64_t nentries = T->GetEntries();
   T->SetCacheSize(cachesize);
   T->AddBranchToCache("*",kTRUE);

   for (Long64_t i=0;i<nentries;i++) {
      T->GetEntry(i);
   }
}
```

Use Case 2

Reading only a few branches in consecutive events

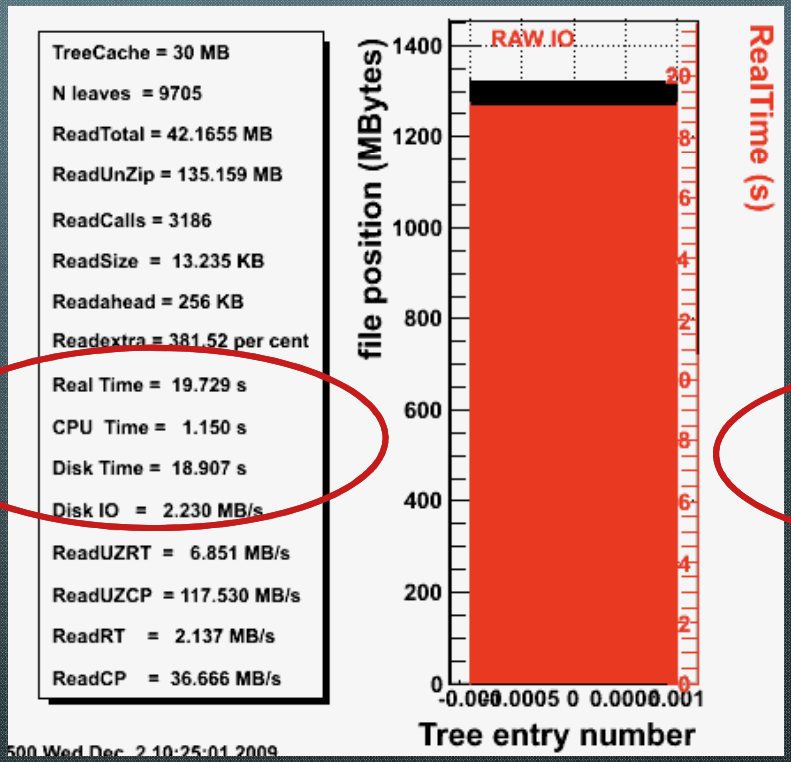
```
void taodr(Int_t cachesize=10000000) {
   TFile *f = TFile::Open("AOD.067184.big.pool.root");
   TTree *T = (TTree*)f->Get("CollectionTree");
   Long64_t nentries=1000;
   Long64_t efirst= 2000;
   Long64_t elast = efirst+nentries;
   T->SetCacheSize(cachesize);
   T->SetCacheEntryRange(efirst,elast);
   TBranch *b_m_trigger_info = T->GetBranch("m_trigger_info");
   TBranch *b_m_compactData  = T->GetBranch("m_compactData");
   T->AddBranchToCache(b_m_trigger_info,kTRUE);
   T->AddBranchToCache(b_m_compactData, kTRUE);
   T->StopCacheLearningPhase();

   for (Long64_t i=0;i<nentries;i++) {
      T->LoadTree(i);
      b_m_trigger_info->GetEntry(i);
      b_m_compactData->GetEntry(i);
   }
}
```

Use Case 2 results

reading 33 Mbytes out of 1100 MBytes

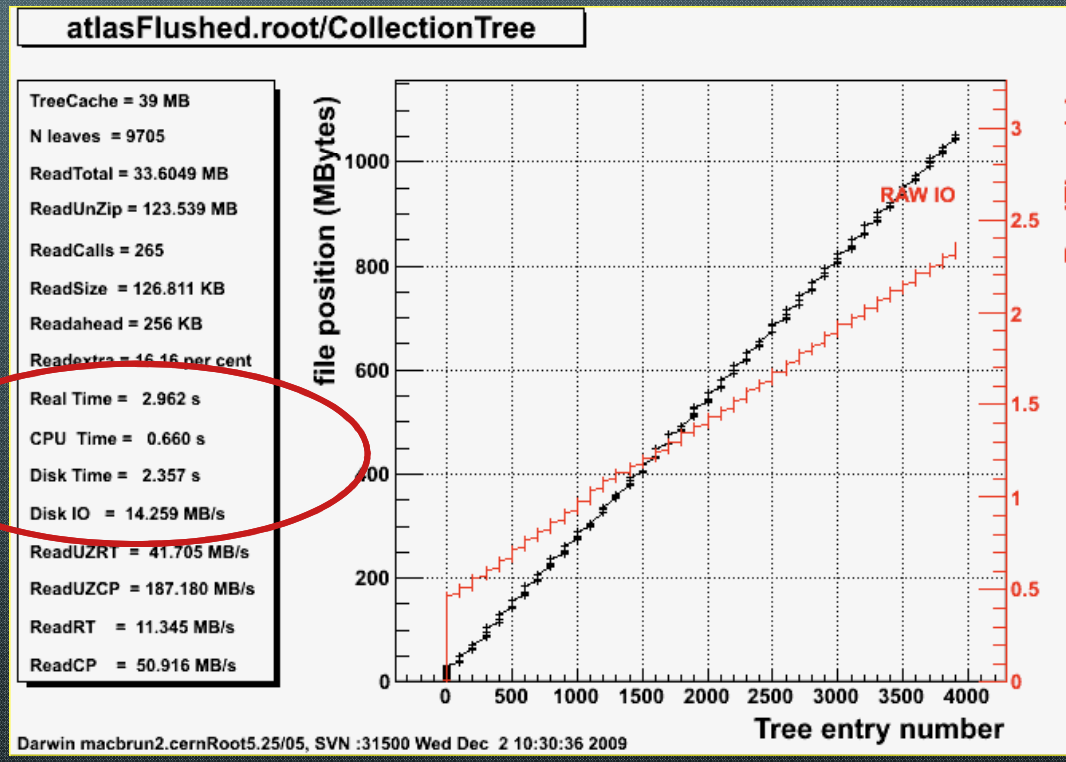
Seek time = 3186*5ms = 15.9s

Old ATLAS file



Seek time = 265*5ms = 1.3s

New ATLAS file



Use Case 3

Select events with one branch then read more branches

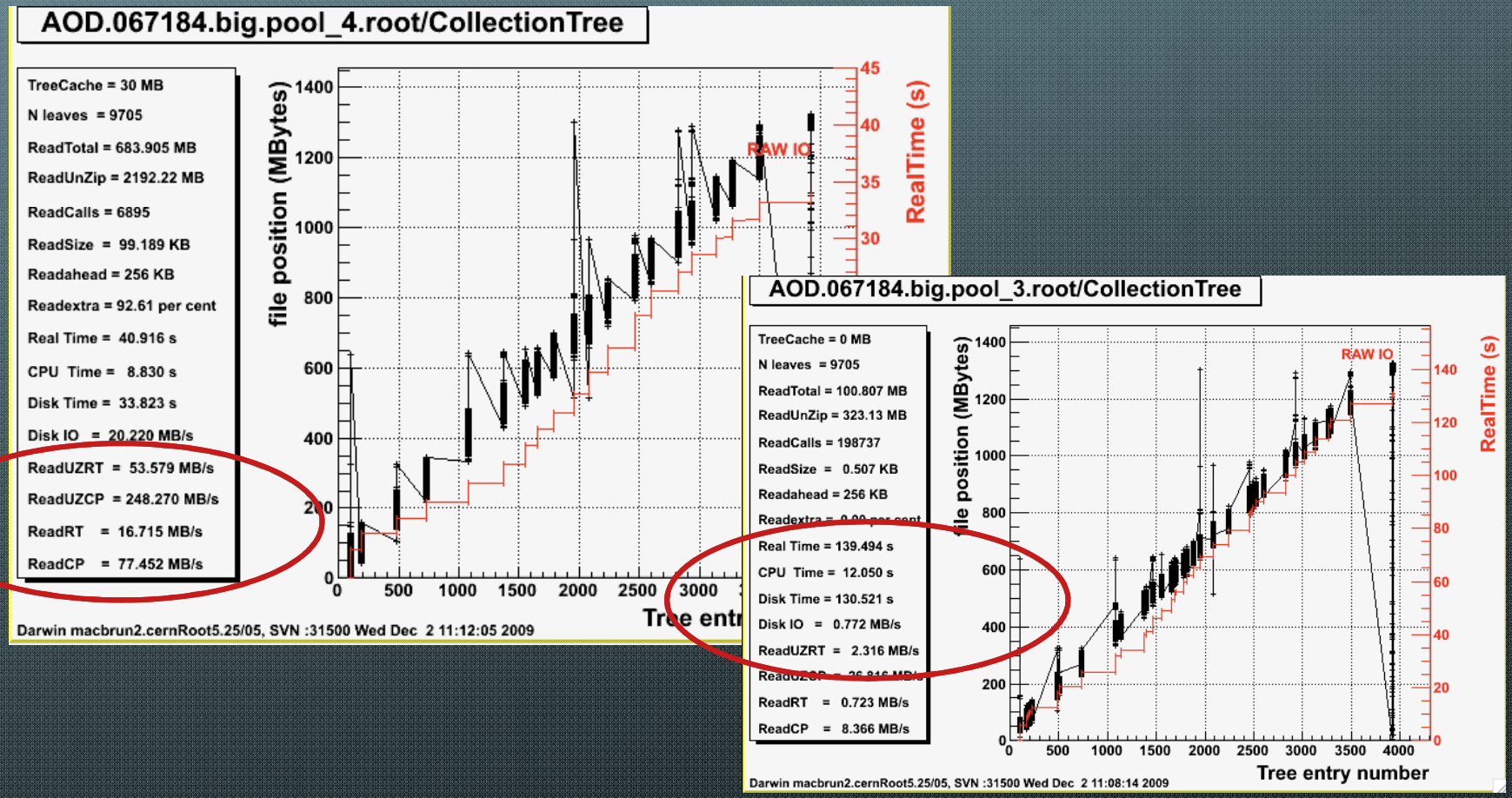
```
void taodr(Int_t cachesize=10000000) {
   TFile *f = TFile::Open("AOD.067184.big.pool.root");
   TTree *T = (TTree*)f->Get("CollectionTree");
   Long64_t nentries=1000;
   Long64_t efirst= 2000;
   Long64_t elast = efirst+nentries;
   T->SetCacheSize(cachesize);
   T->SetCacheEntryRange(efirst,elast);
   T->SetCacheLearnEntries(10);
   TBranch *b_m_trigger_info = T->GetBranch("m_trigger_info");
   T->AddBranchToCache(b_m_trigger_info,kTRUE);

   for (Long64_t i=0;i<nentries;i++) {
      T->LoadTree(i);
      b_m_trigger_info->GetEntry(i);
      if (somecondition) readmorebranches();
   }
}
```

All branches used in the first 10 entries will be cached

Use Case 3 results

reading 1% of the events

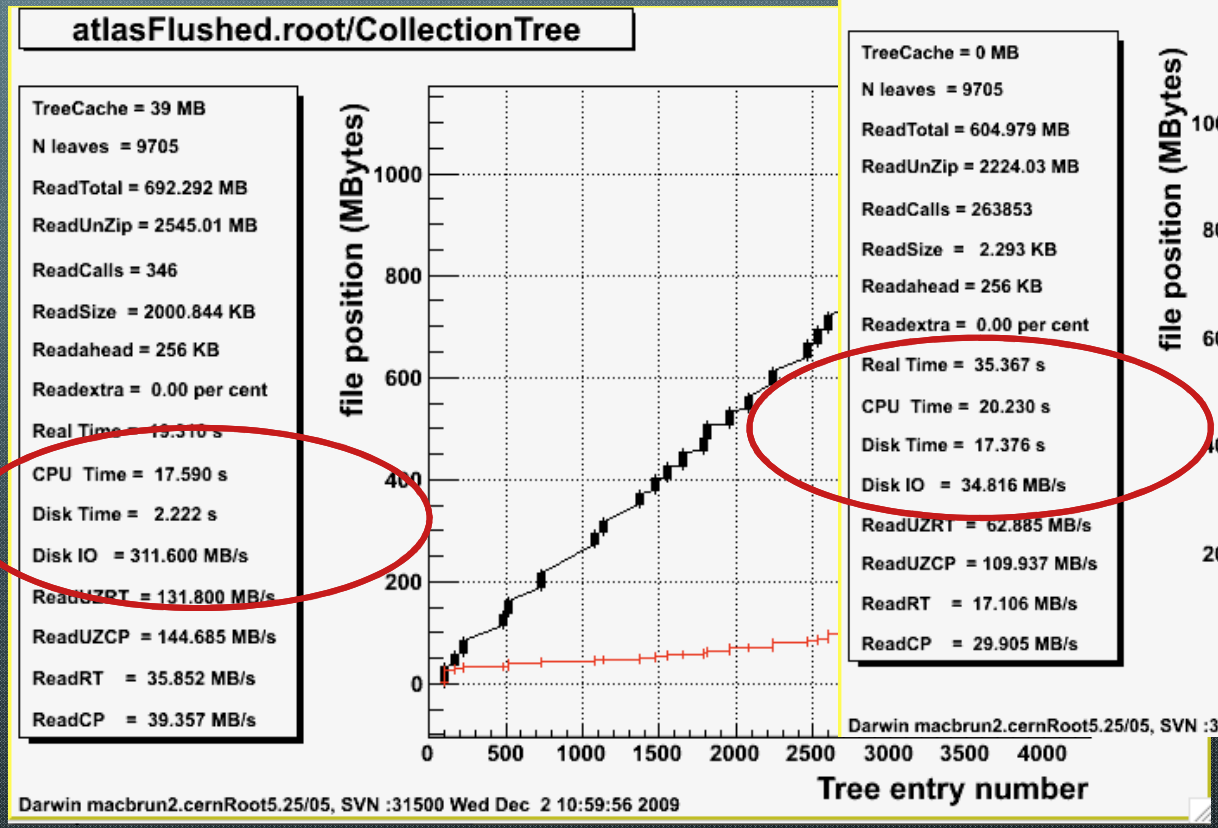
Even in this difficult case cache is better

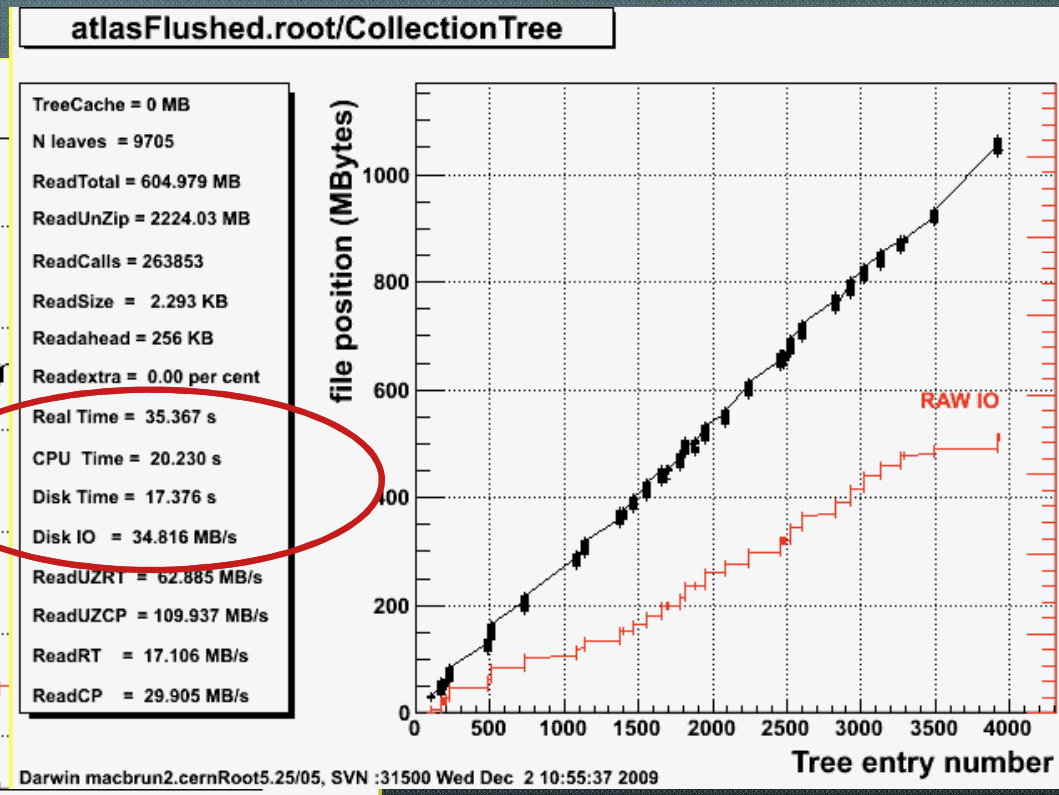


Use Case 3 results

reading 1% of the events

Even in this difficult case cache is better





Reading network files

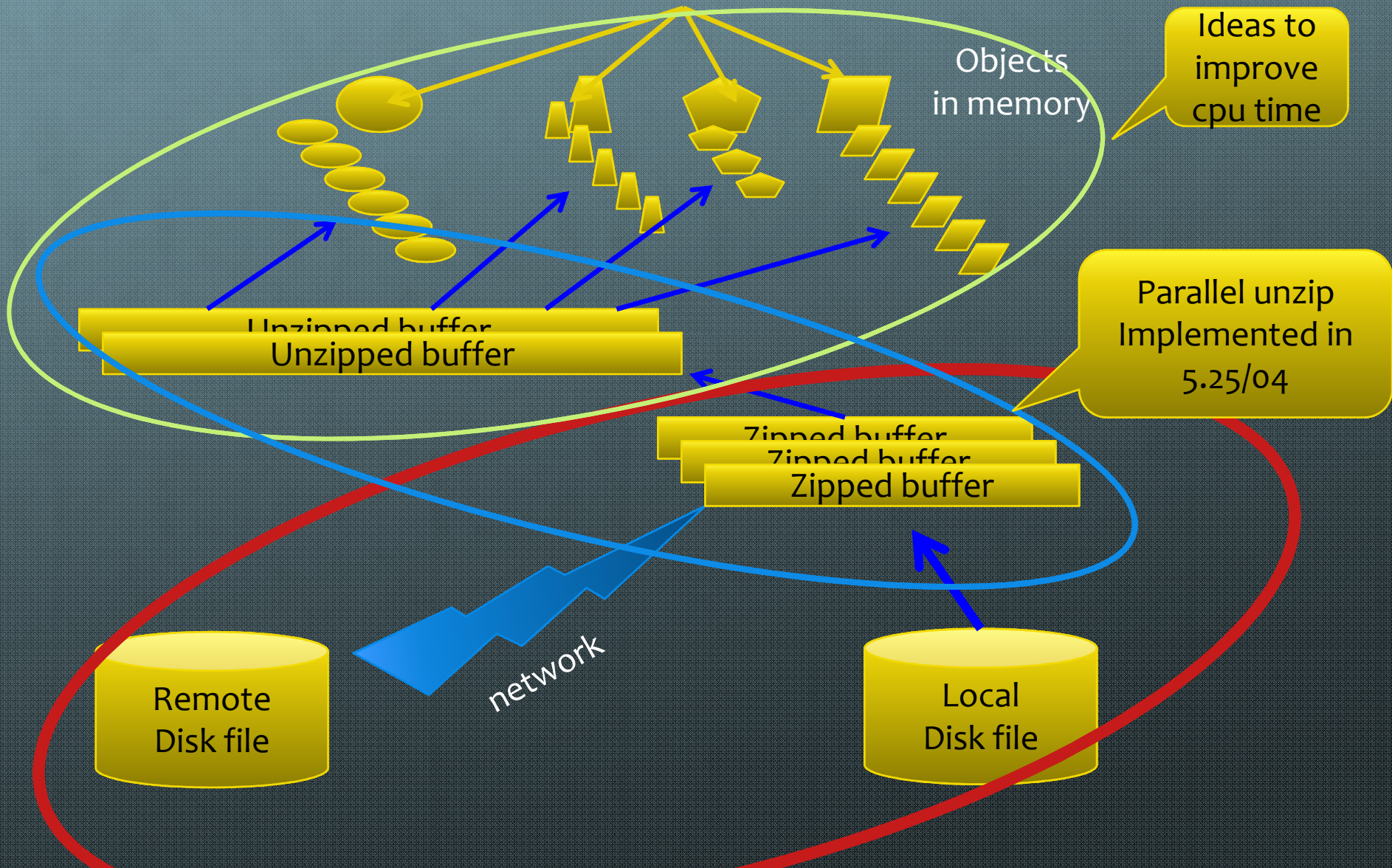
f = TFile::Open("http://root.cern.ch/files/AOD.067184.big.pool_4.root")
f = TFile::Open("http://root.cern.ch/files/atlasFlushed.root")

Same results with xrootd

File type Cache on/off	Local file	LAN =0.3ms 1 Gbit/s	WLAN = 3ms 100 Mbits/s	ADSL=72ms 8 Mbits/s
Atlas orig CA=OFF	RT=720s TR=1328587	RT=1156s NT=400+12s	RT=4723s NT=4000+120s	RT > 2 days NT>2 days
Atlas orig CA=ON	RT=132s TR=323	RT=144s NT=0.1+12s	RT=253s NT=1+120s	RT=1575s NT=223+1200s
Atlas flushed CA=OFF	RT=354s TR=486931	RT=485s NT=120+11s	RT=1684 NT=1200+110s	RT= NT>1 day
Atlas flushed CA=ON	RT=106s TR=45	RT=117s NT=0.03+11s	RT=217s NT=0.1+110s	RT=1231s NT=25+1100s

TR=Transactions
NT=Network Time (latency + data transfer)

Important factors



Summary

- Use of the TreeCache is essential, even with local files
- Flushing buffers at regular intervals help a lot
- In ROOT 5.25/04, FlushBaskets is ON by default and also an automatic basket buffer optimization.
- In ROOT 5.25/04 a simplified interface to TreeCache is provided via TTree.
- Files written with 5.25/04 will be faster to read

Summary 2

- Version 5.26 will be released on December 15.
- This version is back compatible with 5.22 and 5.24.
- We hope that collaborations will move soon to this new version.