TEntryList

A new class to support large and scalable event lists

Outline

- Motivation
- Internal structure of **`TEntryList`**
  - **`TEntryListBlock`** class
  - Lists for a **`TTree`** and for a **`TChain`**
  - Combining lists
- **`TEntryListFromFile`**
- **`TEntryList`** and **`TEventList`**
- Conclusions

Motivation

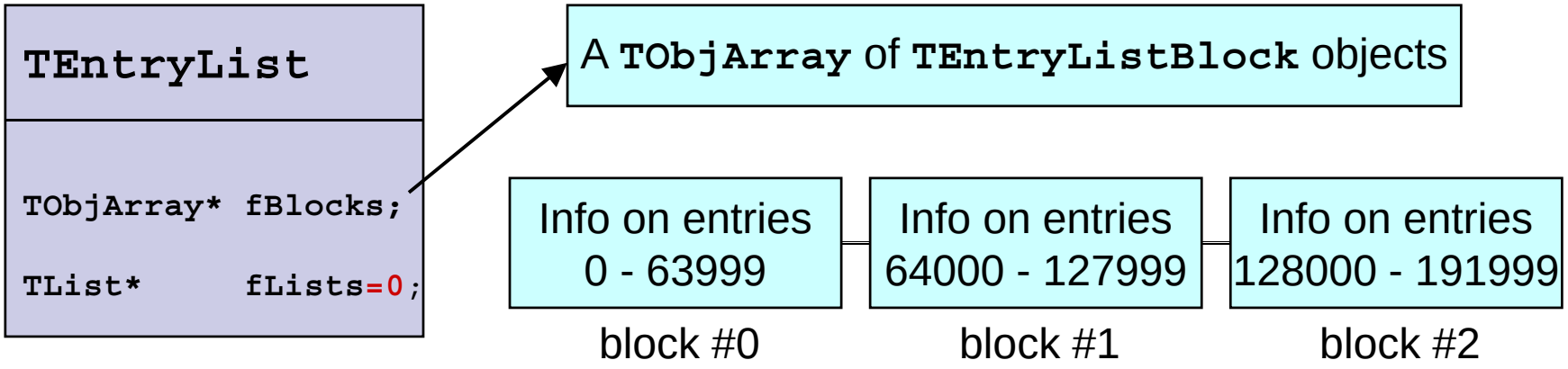
- When processing a **`TTree`** or a **`TChain`**, we need an object to store the indices of entries, passing the selection criteria, so that next time we can limit the processing to those entries.
- **`TEventList`** is already doing it. So, why change **`TEventList`**?

  To make it better!
  - Scalable
  - Modular
  - Small
  - Only partially loaded in memory

  These changes are especially important for Proof. We want an object, from which partial information can be extracted and processed independently.

`TEntryList` and `TTree`

- **`TEntryList`** for a **`TTree`**:
  - Stores information on the tree entries passing or not passing the selection criteria.
  - **`TEntryListBlock`** objects are used for storage. Each block contains information on 64000 entries.

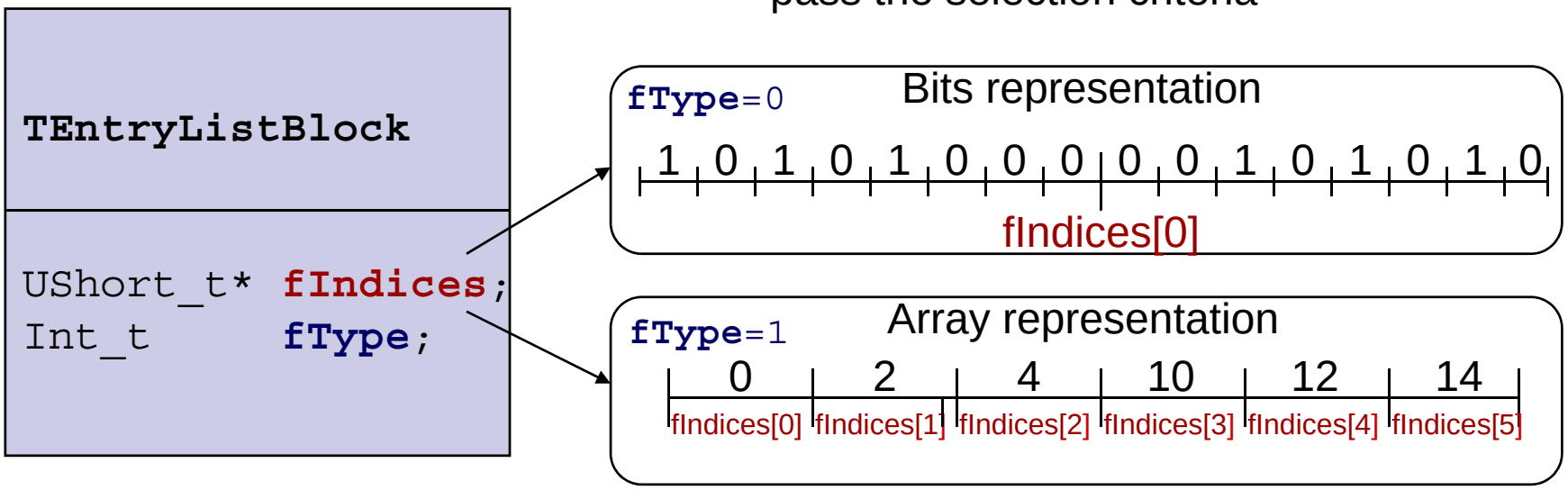


TEntryListBlock - 0

- A utility class, used by **`TEntryList`** to store indices of the **`TTree`** entries, passing the selection criteria
- A UShort_t array is used for storage
- There are 2 ways to store the indices: as bits or as a regular array (see next slide)
- **`TEntryListBlock::OptimizeStorage()`** checks if it makes sense to switch to the other representation
  - When the entry list is filled via **`TEntryList::Enter(Long64_t entry)`** function, it optimizes the storage in the previous block before filling the next block

TEntryListBlock - 1

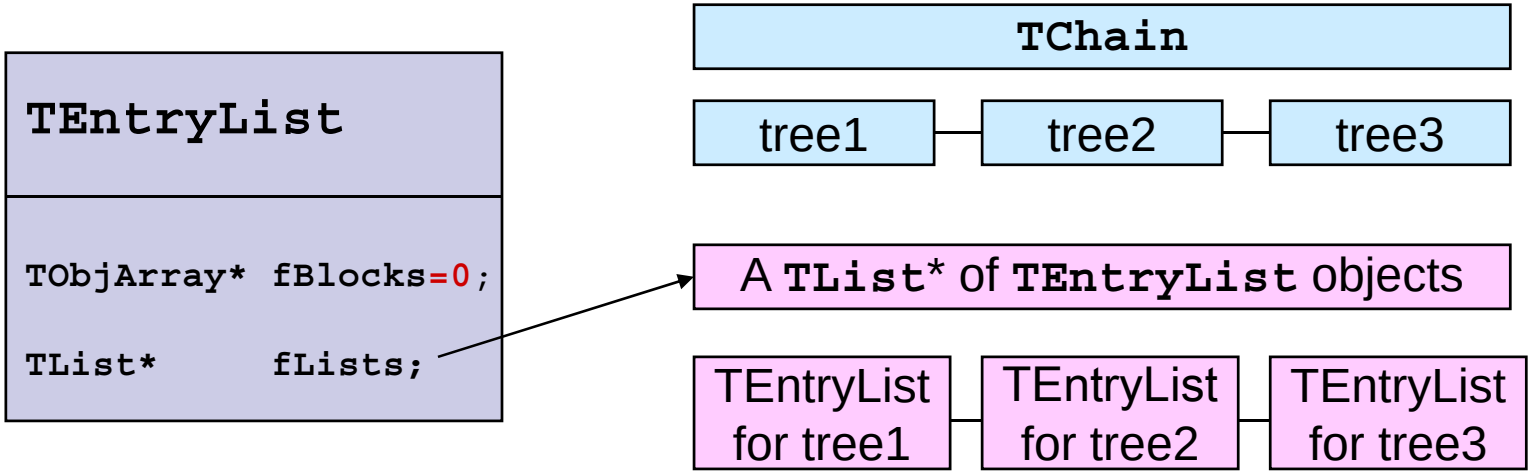
Suppose that this block stores information that entries
0, 2, 4, 10, 12, 14
pass the selection criteria



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

`TEntryList` and `TChain`

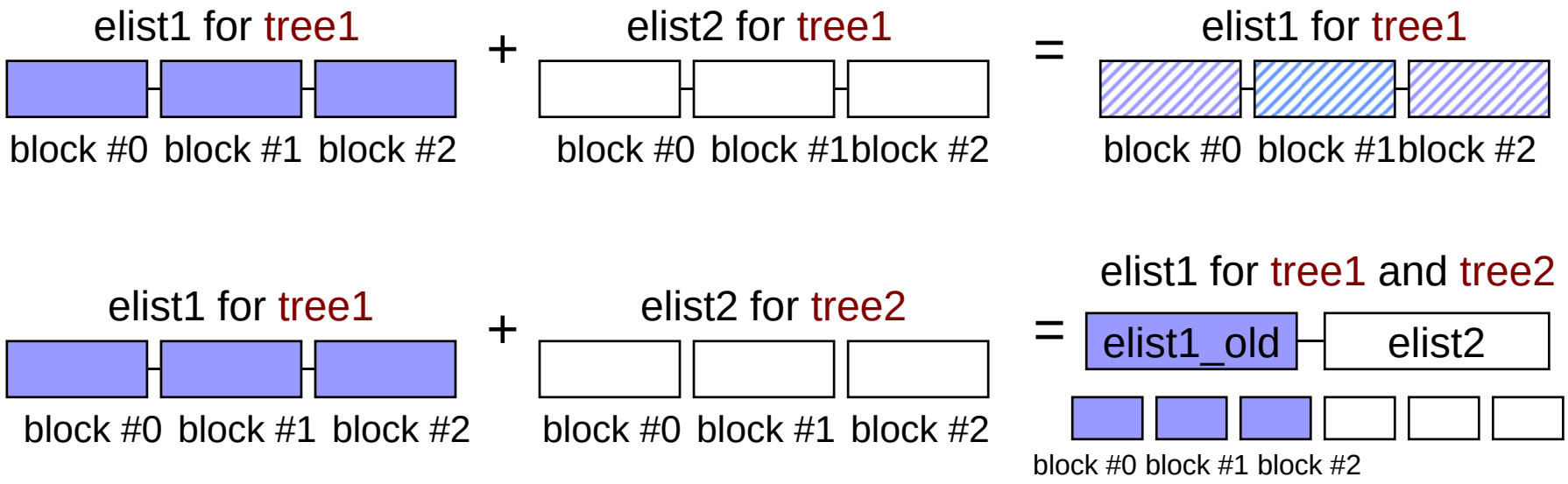
- Keeps a `TList*` of `TEntryLists` for individual `TChainElements`
- `TEntryList::GetEntryList(treename, filename)` function allows to extract those “sublists” and use them as ordinary `TEntryLists` for a `TTree`



Combining `TEntryLists` - 0

- **`TEntryList::Add(TEntryList *elist)`**
  - If this entry list is for a **`TTree`**:
    - If elist is for the same **`TTree`**, their contents are merged (like it was for **`TEventLists`**)
    - If elist is for a different **`TTree`**, this entry list is turned into an entry list for a **`TChain`** of this 2 **`TTrees`**
  - If this entry list is for a **`TChain`**:
    - Lists, for **`TTrees`** present in both this list and elist, are merged.
    - Lists for **`TTree`** not yet in this list are added to the **`TList`** of sub-lists.

Combining TEntryLists - 1



TTree::Draw

- Option “entrylist” should be used to write the results into an entry list:

```
tree->Draw(“>>elist”, “x<0”, “entrylist” );
TEntryList *el = (TEntryList*)gDirectory->Get(“elist”);
```

- To limit the processing to the entries in a TEntryList:

```
tree->SetEntryList(elist);
```

- **TTree::SetEventList()** function is still available, but internally the **TEventList** is transformed into a **TEntryList** object. All tree headers have to be loaded for this transformation!

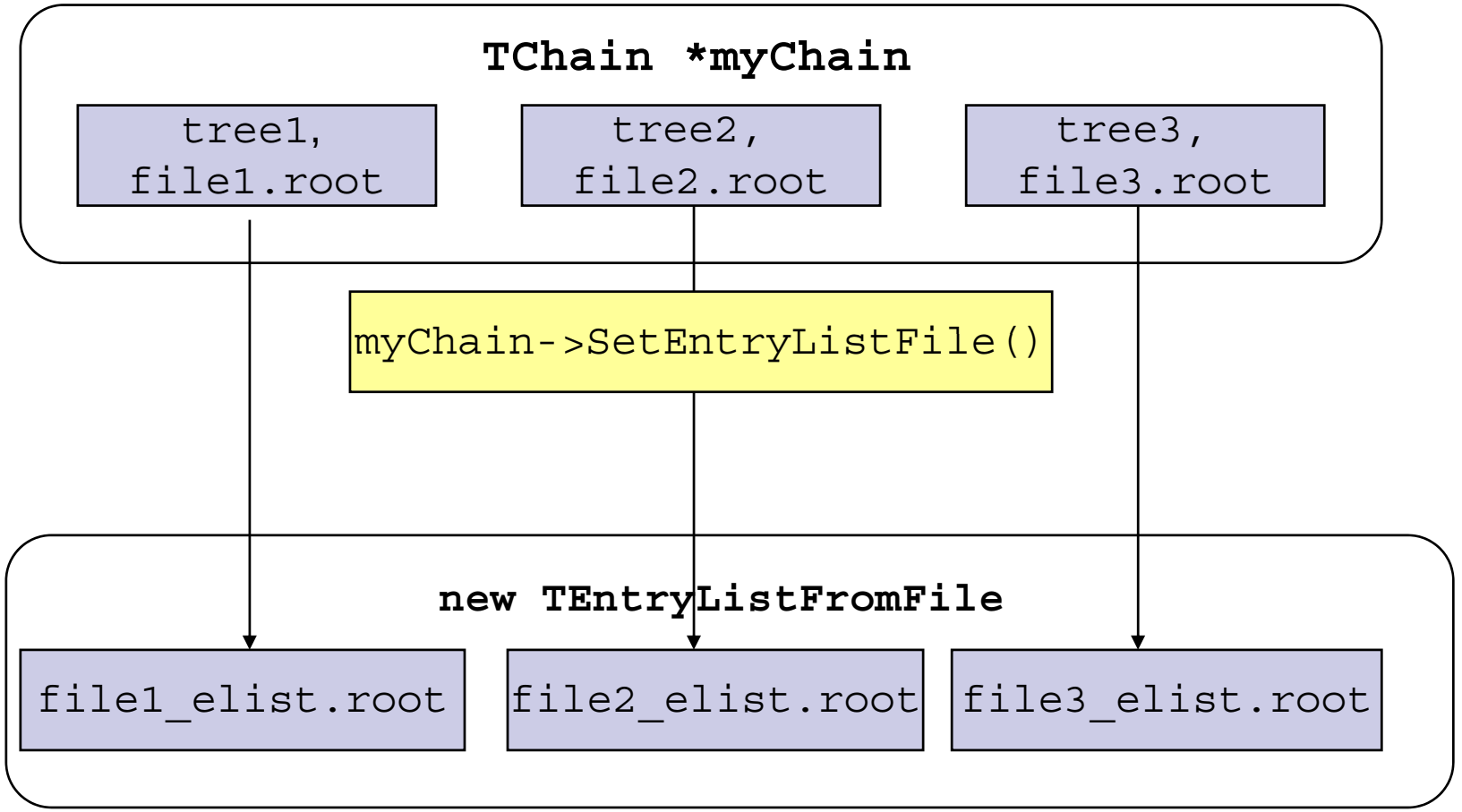
TEntryListFromFile - 0

- A utility class managing **TEntryLists** from different files.
  Use case: the user has entry lists for the trees of the chain, stored in separate files. We don't want to load all of them in memory at the same time.
- This class is called by

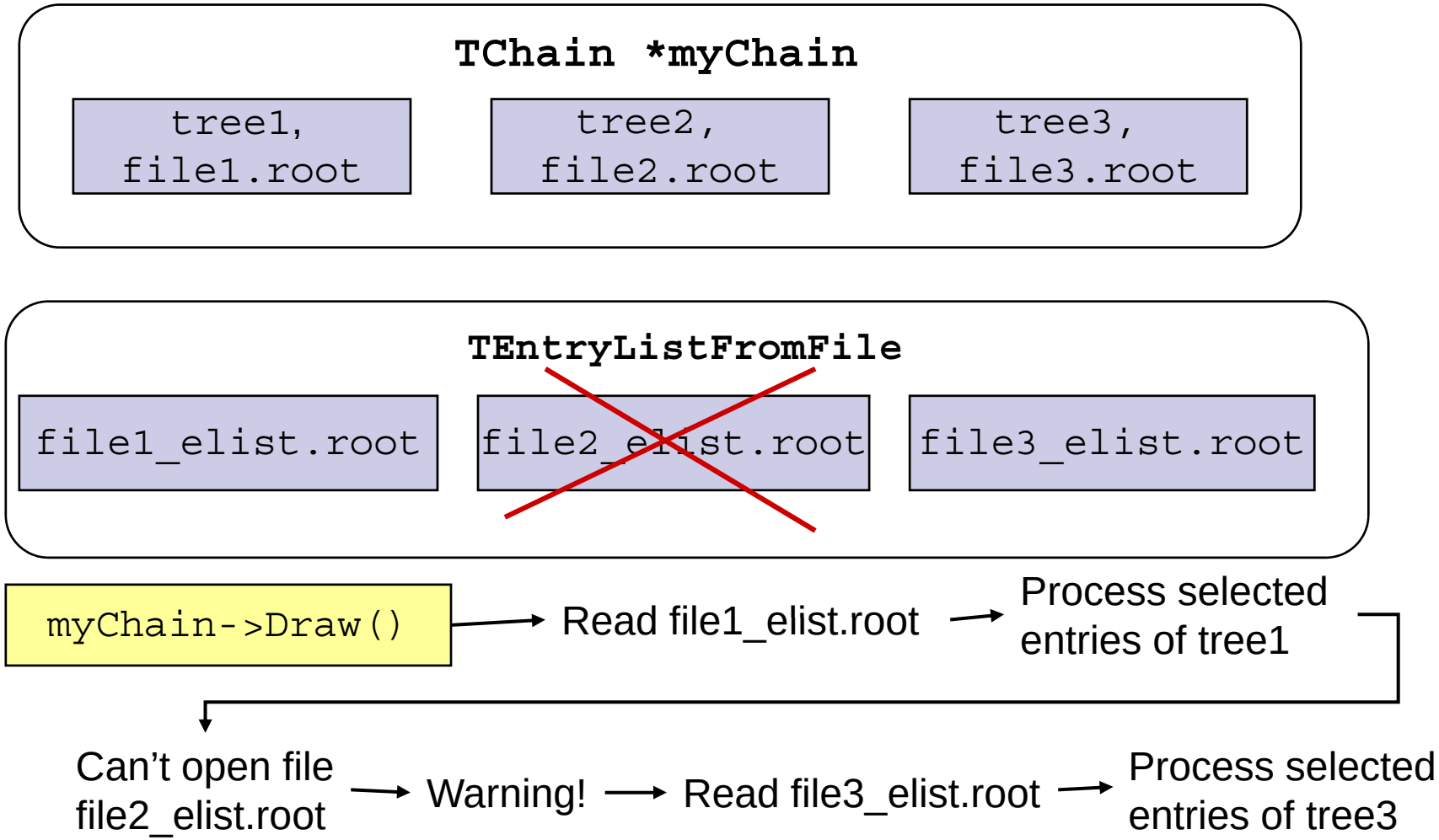
```
TChain::SetEntryListFile(const char *filename, Option_t *opt)
```

- It finds the entry list files, corresponding to **TChainElements**, and only loads a file in memory, when an entry from this **TEntryList** is requested (by **TChain::GetEntryNumber()**)
- If there is an error opening an entry list file, the tree, corresponding to this list, is not processed.

TEntryListFromFile - 1



TEntryListFromFile - 2



`TEntryList` and `TEventList` - 0

- Performance: 20 trees, 1.000.000 entries each, not selective cut:

```
chain->Draw(">>elist", "Entry$%2==0", "entrylist");
```

  - **`TEntryList`**: CPU time 11.098 **`TEventList`**: CPU time 17.075

More selective cut `Entry$%100==0`

  - **`TEntryList`**: CPU time 8.8 **`TEventList`**: CPU time 8.6

```
chain->SetEntryList(elist); //or SetEventList()
chain->Draw("x", "", "goff");
```

  - **`TEntryList`**: CPU time 15.3 **`TEventList`**: CPU time 13.4

More selective cut `Entry$%100==0`

  - **`TEntryList`**: CPU time 6.7 **`TEventList`**: CPU time 6.6

`TEntryList` and `TEventList` - 1

- Memory:
  - When less than 1/16 of entries pass the cut, **`TEntryList`** is ~8 times smaller than **`TEventList`** (UShort_t instead of Long64_t)
  - When more than 1/16 of entries pass the cut, the size of **`TEntryList`** stays the same (bits representation), while a **`TEventList`** gets bigger as more entries pass
  - The size of **`TEntryList`** is also very dependent on the distribution of passing entries in the entry range

Conclusions

- New classes **`TEntryList`**, **`TEntryListBlock`** and **`TEntryListFromFile`** have been added
  - Entry lists can be used with **`TTree::Draw`**, with selector-based analysis, and stand-alone
  - **`TTree::SetEventList(TEventList* el)`** now internally uses **`TEntryList`**
- Modular structure of **`TEntryList`** allows to extract sublists, that belong to specific trees, and process them independently. That, combined with its significantly smaller size, makes **`TEntryList`** much better suited for Proof than **`TEventList`**.
- Next step: full integration of **`TEntryList`** into PROOF
  Work in progress…