Histogram and Concurrency

- Histograms created for every thread
  - faster to fill
  - memory can be an issue
  - merge can become a bottleneck
- Shared histograms between the threads
  - locking the fill can degrade performances

- Multiple copies of the histogram preferable
- Shared histograms only in case of memory problem (many histograms or multi-dimensional)
  - small probability of having threads filling the same bins

Rene Use Case and Wishes

- Multi-threaded CPU intensive applications filling several (~ 100-1000) histograms continuously
  - monitor results by looking at histograms frequently (e.g. every minute)
- wish to have:
  - histogram management tool in a multi thread environment
    - histogram booked automatically in every thread and register in a list/thread
    - histograms from the lists are merged regularly and efficiently.
      - do not want to spend more than needed in TH1::Merge
      - do not want to waste time in navigating the lists to find corresponding histograms with the same name to merge
      - want to merge in memory (not going through I/O)

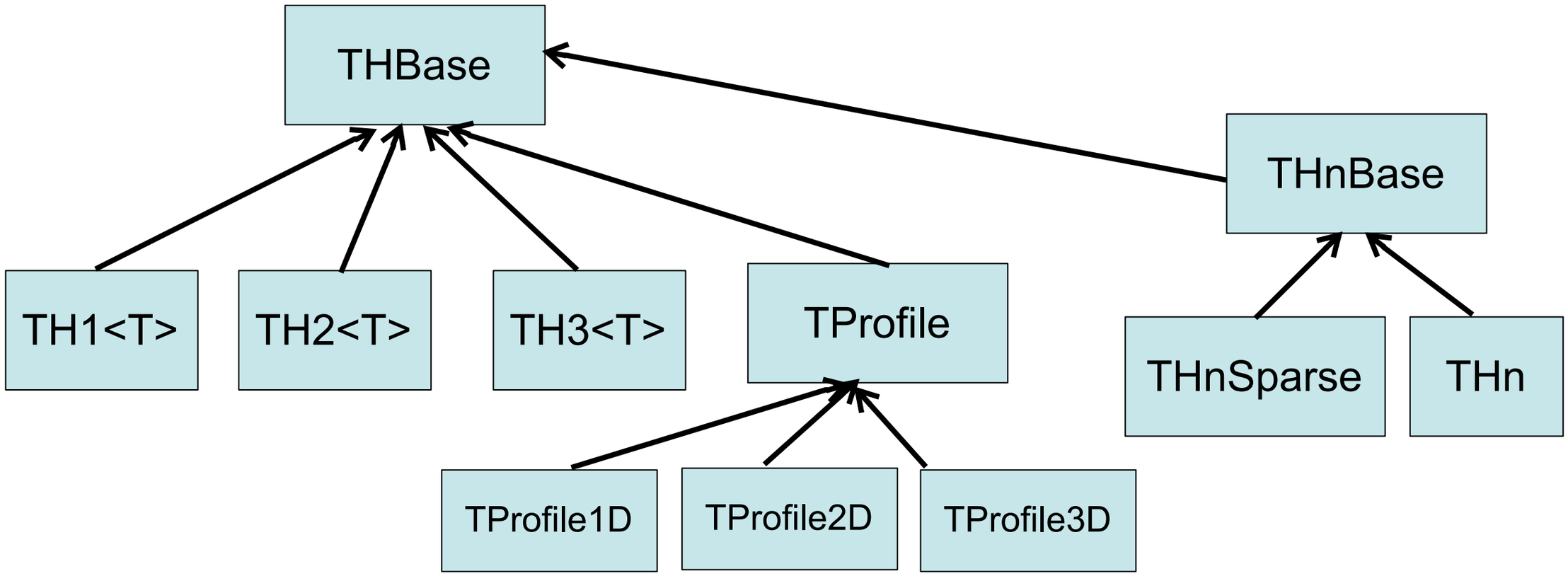
Personal Thoughts on Histograms

- Management histograms in a concurrent environment is certainly needed (for the single users not for frameworks)
  - be part of a general tool in ROOT and not of histogram library
- Having a new Proof-Lite working both in multi-process and multi-threaded environment
  - this tool manages histograms and does the merge

- Histogram library should provide:
  - super efficient Merge
  - a thread-safe histogram for other use cases

Histogram Re-Design

- Improve histograms classes
  - new histogram design to fix hierarchy structure



  - maintain as much as possible backward compatibility in the interfaces

Histogram Re-Design (2)

- Being more ambitious and change histograms to have:
  - Very light histogram class (no graphics or other attributes)
  - Different histogram types (e.g. using template policies) for:
    - fixed bins
    - variable bins
    - label bins
    - no bins (just array of entries)
    - atomic bins (thread safe histogram)

Histogram Re-Design (3)

- TH1 class as a wrapper to these light objects
  - constructor from light histograms
- Conversion between histograms
  - eg. a no-bin histogram can be converted to a binned histogram
- Specialised operations (e.g. Merge) for the different objects
  - no virtual inheritance
    - current TH1::Merge is dominated by TH1::Get/SetBinContent
  - can vectorise some of the operations

Multi-Thread and Fitting

- Need to add support for multi-threads when fitting histograms/trees or whatever data set
  - parallelize likelihood/ chi2 calculations
- Not difficult to add once we have defined an high level interface to run parallel tasks in ROOT
  - e.g. a new Proof Lite

New Function/Formula classes

- With new TFormula extend possibility of building TF1 objects
  - support combinations of TF1 (e.g. additions)

- Providing now support for re-normalising the TF1 objects (i.e. scale the value by the integral in its range)

- Support of I/O for TF1 based on functor objects
  - need I/O for interpreted classes

Vectorization

- Vectorize fitting (chi2/likelihood calculations)
  - add a vectorized function interface to be used for fitting
  - can use VDT and/or Vc

```
• double F(const double * xobs, const double * params);
• void F(const double * xvec, const double * params, double * f);
• template <typename T>
  T F( T x, cont double * params);
```

  - what is the best object to use ?
    - a structure with big array is good for SIMD but not convenient for cache and multi-threads environment
    - data structure with Vc types
      - std::vector<double_v> x ?

Documentation

- Reference Guide
  - migrate to Doxygen and revise/update class documentation
- User Guide
  - Look at the various chapters
  - revise and plan to update/re-write them
  - Common effort in the team and/or hire a documentation expert
    - some more updated doc exists in Drupal but needs to be put in the User Guide
    - we can probably remove also some existing material