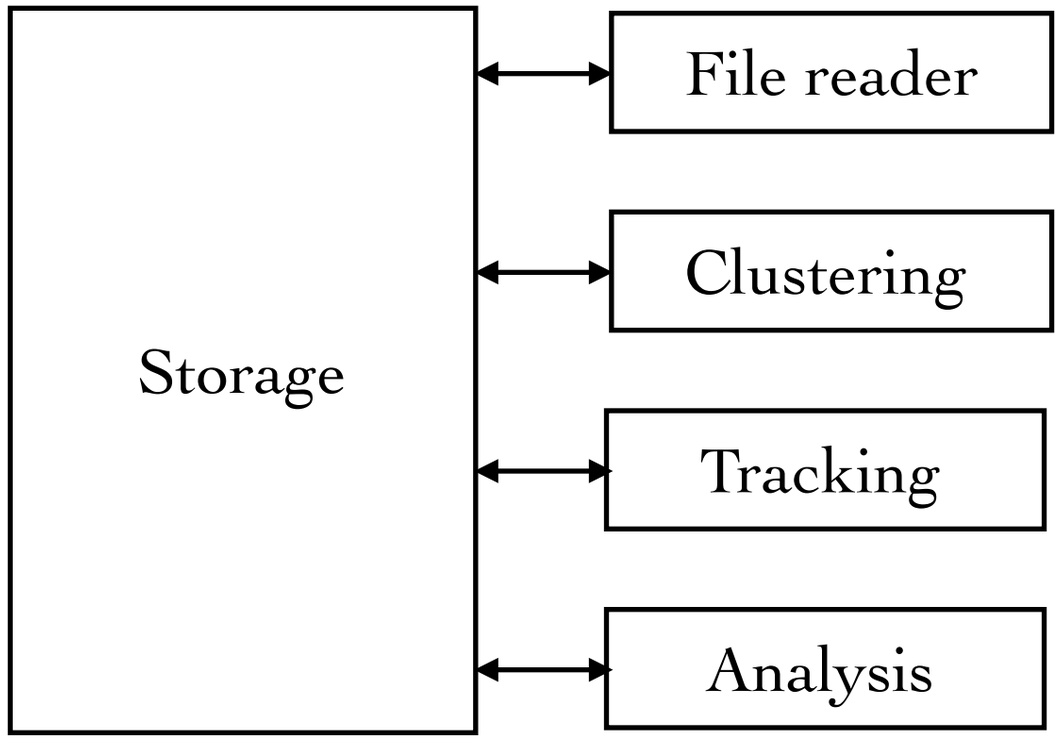
Testbeam software overview

Philosophy

- As per Guadi, Marlin, a chunk of data (event) is analysed sequentially through several algorithms, which are able to pass information to each other

- At the end of each event processing, objects can be removed from the storage element (all, or selectively ie. based on time)



Core

- There are a few core files:

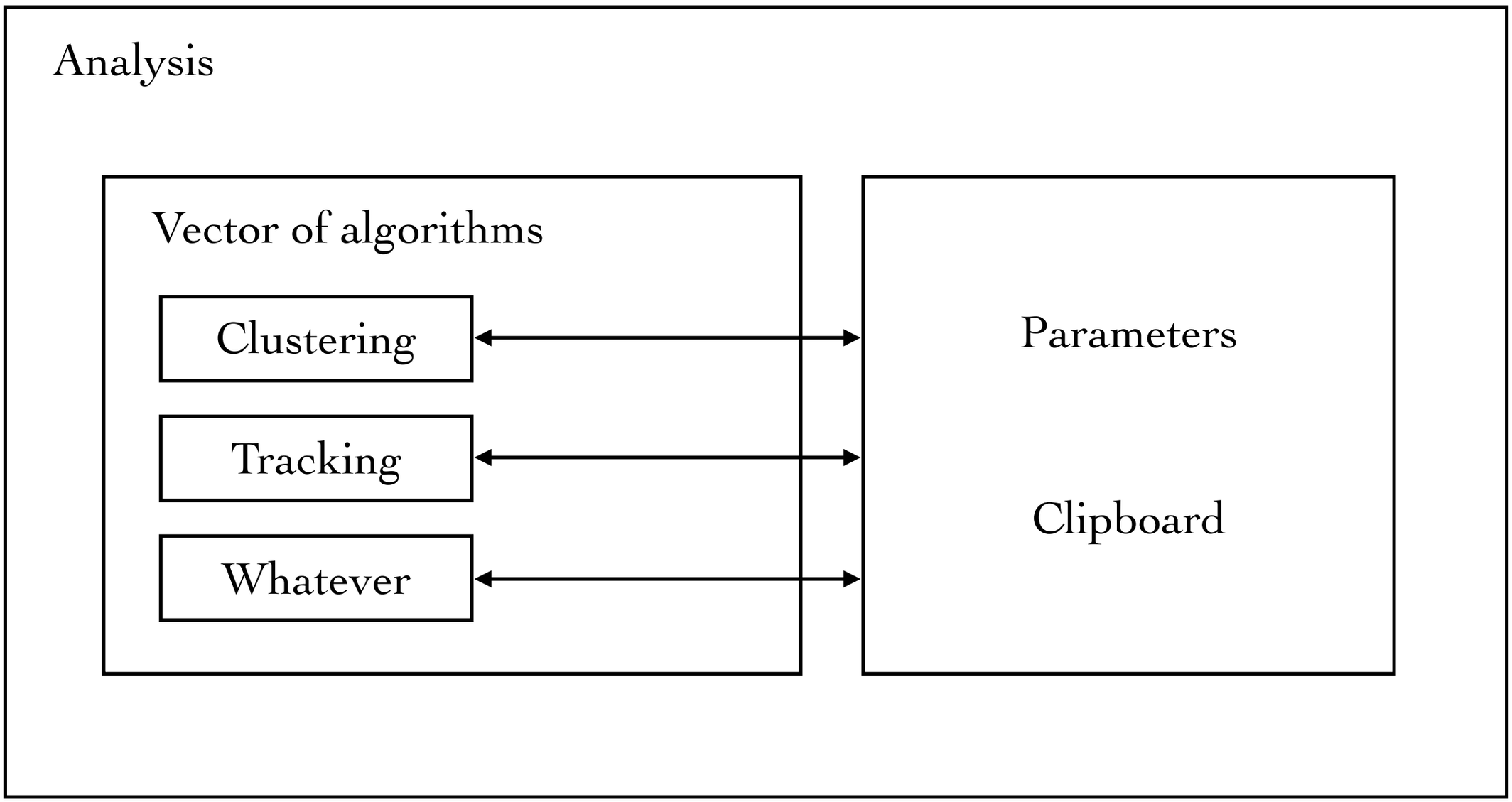
  Analysis - this is simply a class which holds two things, a set of **parameters** and a vector of **algorithms**. For each algorithm, it runs an initialise function at the start, a run function for each event, and a finalise function at the end of processing

  Parameters - this is simply a global object where parameters can be placed (minimum no. clusters on track, etc.). For each detector in the conditions file, it holds an entry with the number of pixels, pitch, global position, local to global transform

  Algorithm - all algorithms inherit from algorithm, allowing the analysis class to run each of them despite not knowing what they are. They have a name, and the three functions mentioned

  Clipboard - the object which passes objects between algorithms. It contains a map of objects, which you can get by their name

Core


Analysis
Vector of algorithms
Clustering
Tracking
Whatever
Parameters
Clipboard

Objs

- All of the objects which you want to pass between algorithms/write to file

  All objects of this type inherit from **TestBeamObject**, which is simply a TObject (required by root in order to write to file)

  Currently implemented are Timepix3Pixel, Timepix3Cluster, Timepix3Track

Algorithms

- Where the real work is done. Each algorithm inherits from Algorithm, and can do whatever it likes. Currently there are a few there:

  Timepix3EventLoader - very self-explanatory, decodes input files and puts pixels onto the clipboard.

  Timepix3MaskCreator - picks up pixels from the clipboard, and if any pixel fires more times than X then it will be masked. At the end, it reads in the current mask file, masks the pixels that it found to be noisy, and writes out a new mask file

  Timepix3Clustering - very stupid algorithm that looks at all neighbouring pixels, and if they have a time difference of < 100 ns clusters them. Calculates the centre of gravity (ToT weighted at the moment) and puts the clusters on the clipboard

  BasicTracking - takes a hit on the reference plane, and looks for the closest hit (in time) on all other planes. If the hit is within ~100um then the hit is added to the track. Tracks are fitted and placed on the clipboard

Steering

- The last part, located in **core**. This is simply the "main" function for c++, which makes an **Analysis** object and tells it what algorithms to add

  All algorithm parameters (that you don't want to set from the command line) can be set here - defaults probably stay in the algorithm declaration, but if you want to change them then you can

  Command line arguments are read

  The analysis class is run

  Escape behaviour added - if you ^C while the code is running, it will tell the analysis class to finalise all of the algorithms so that the histograms are written out properly and the file closed

Size of the code

- The **core** section of the code, that handles passing objects, running algorithms, etc. consists of **3** .cpp files, and **5** header files
- Each algorithm has 1 .cpp file and 1 header file
- Each object has 1 header file

In total the code is probably 12 files, with an addition 2 for each algorithm implemented. It should be fast (no overhead), easy to install and only links against ROOT for compilation

Compilation is performed with a simple Makefile. If you add a new algorithm, it will attempt to compile it without you doing anything (simply looking for all .C files in the algorithms folder)

To include your algorithm in the analysis, just add the header in Steering.C and then:

```
MyAlgorithm* myAlgorithm = new MyAlgorithm();
analysis->add(myAlgorithm);
```

Trying it out

- The code is now committed to git:

  https://gitlab.cern.ch/CLICdp/tbAnalysis

- Once you set up git (blame/ask Adrian) then you can check it out:

  git clone ssh://git@gitlab.cern.ch:7999/CLICdp/tbAnalysis.git tbAnalysis

- Then type “make”

- An example set of data (from September ’15) is included on the repository, and if you go the macros folder and run **analyse.sh** then it will begin analysing the data