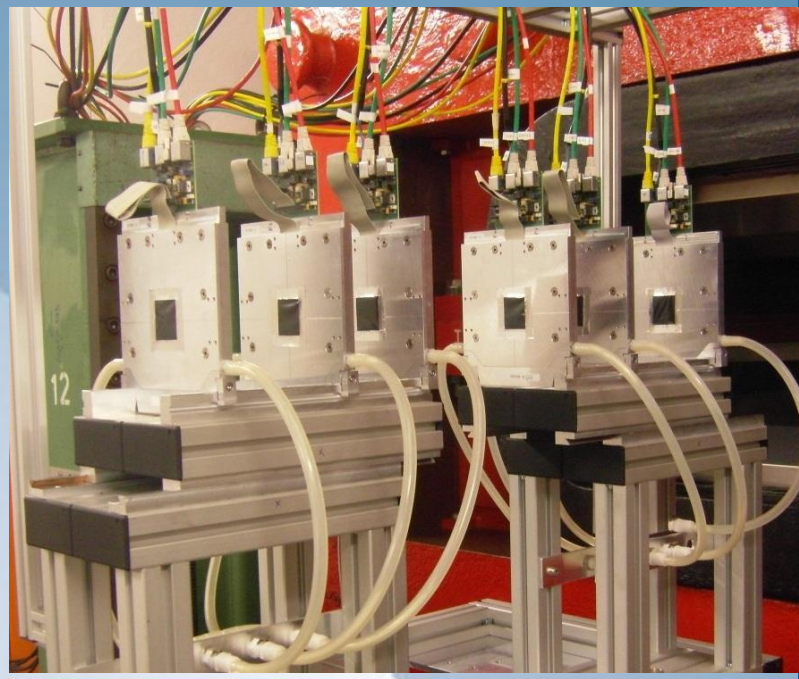
Pixel Beam Telescope - Online Monitor



Timing, Tests and Optimisation

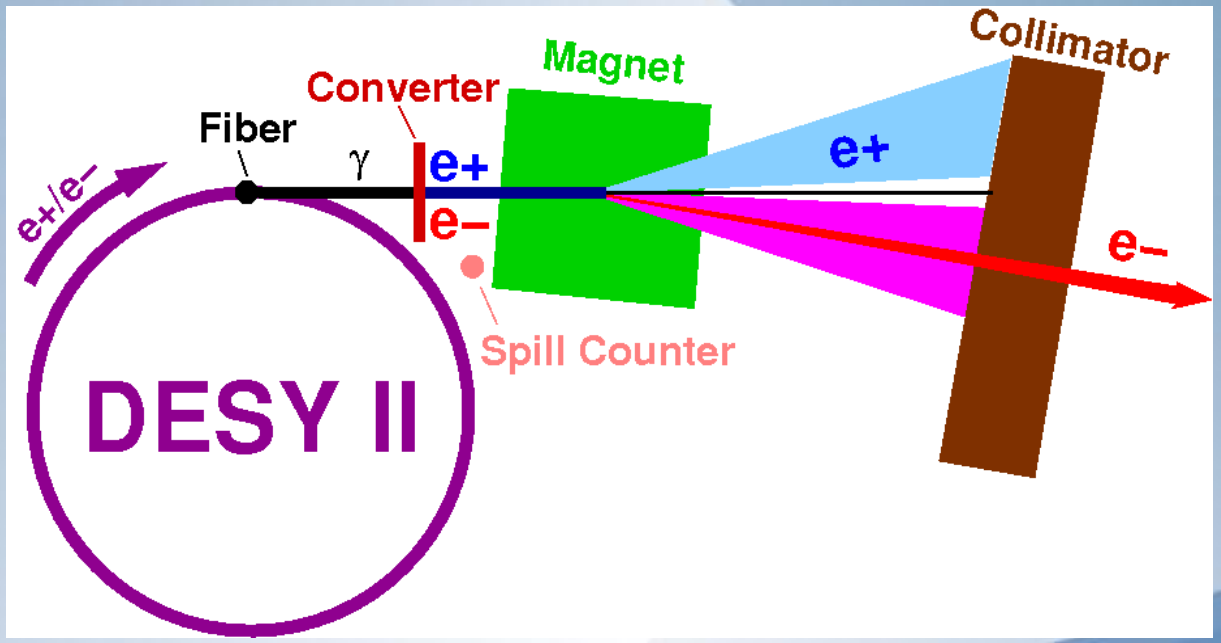
Michał Wysokiński

AGH University of Science and Technology, Krakow, Poland

Supervisor: *Igor Rubinsky*

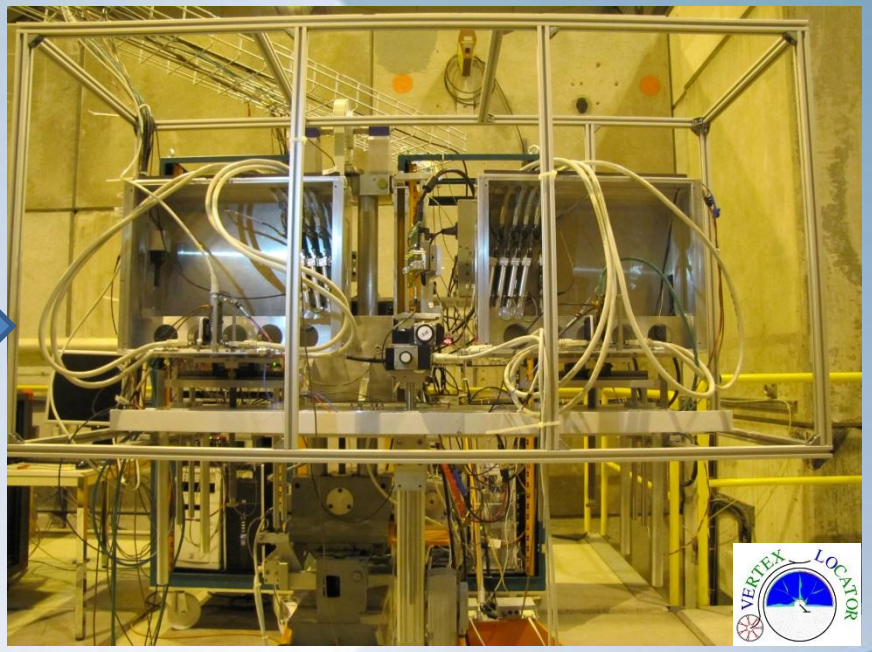
DESY test beam

- Test beams are used for determinig properties of new cutting edge detector systems
- They use devices called telescopes



Pixel telescopes

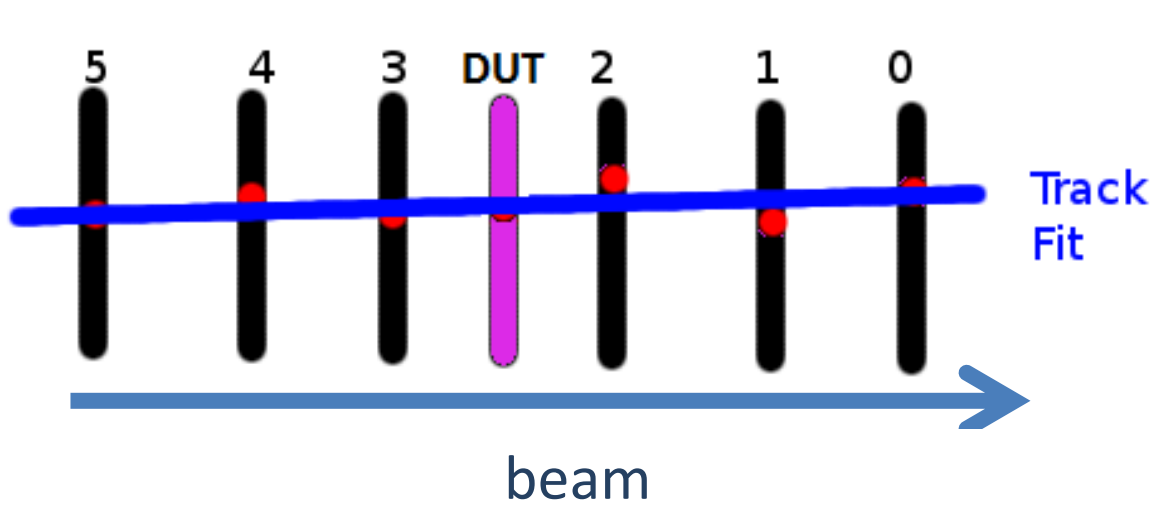
- Telescopes work like a normal camera



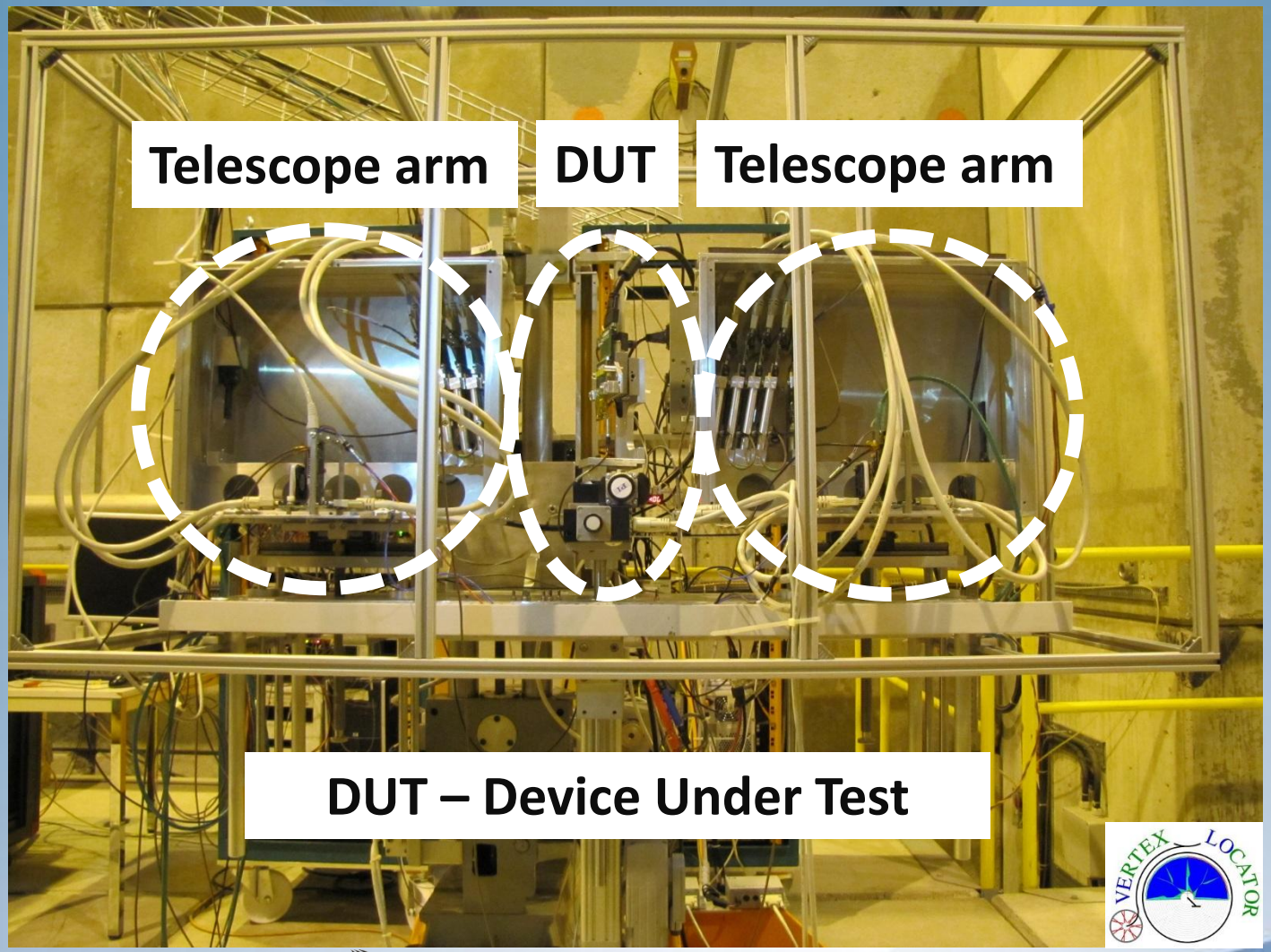
AGH DESY ATLAS EXPERIMENT

What do we want to measure with the telescope?

Check whether a device works or not and measure its properties:
- resolution
- etc...



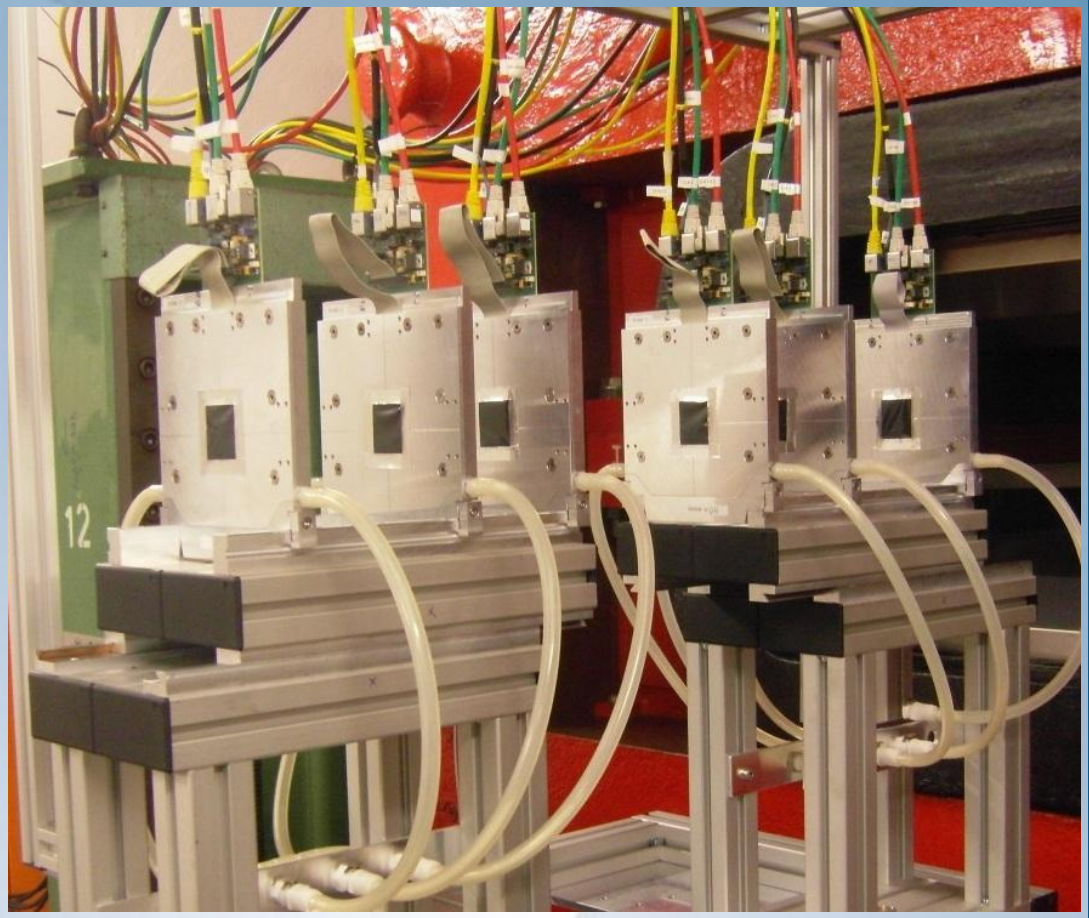
Pixel telescopes



AGH DESY ATLAS EXPERIMENT

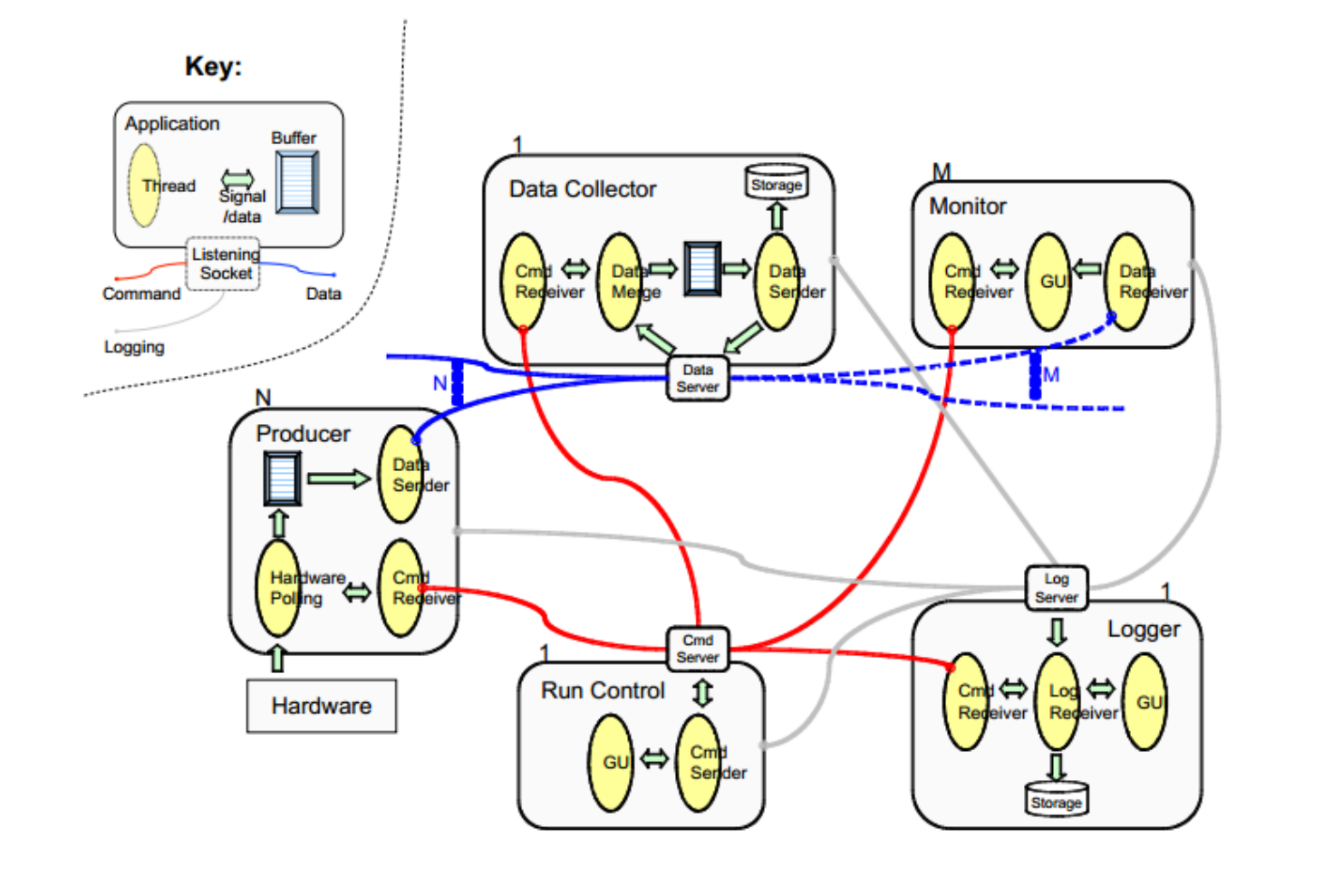
„Datura” telescope

- 6 telescope planes
- MIMOSA 26 sensors:
  - 1152x576 pixels = 0.7 Mpix
  - „pictures” are taken every 115.2 µs
  - binary readout
- Sensors are usually cooled to 15°C (to stabilize their performance in time)

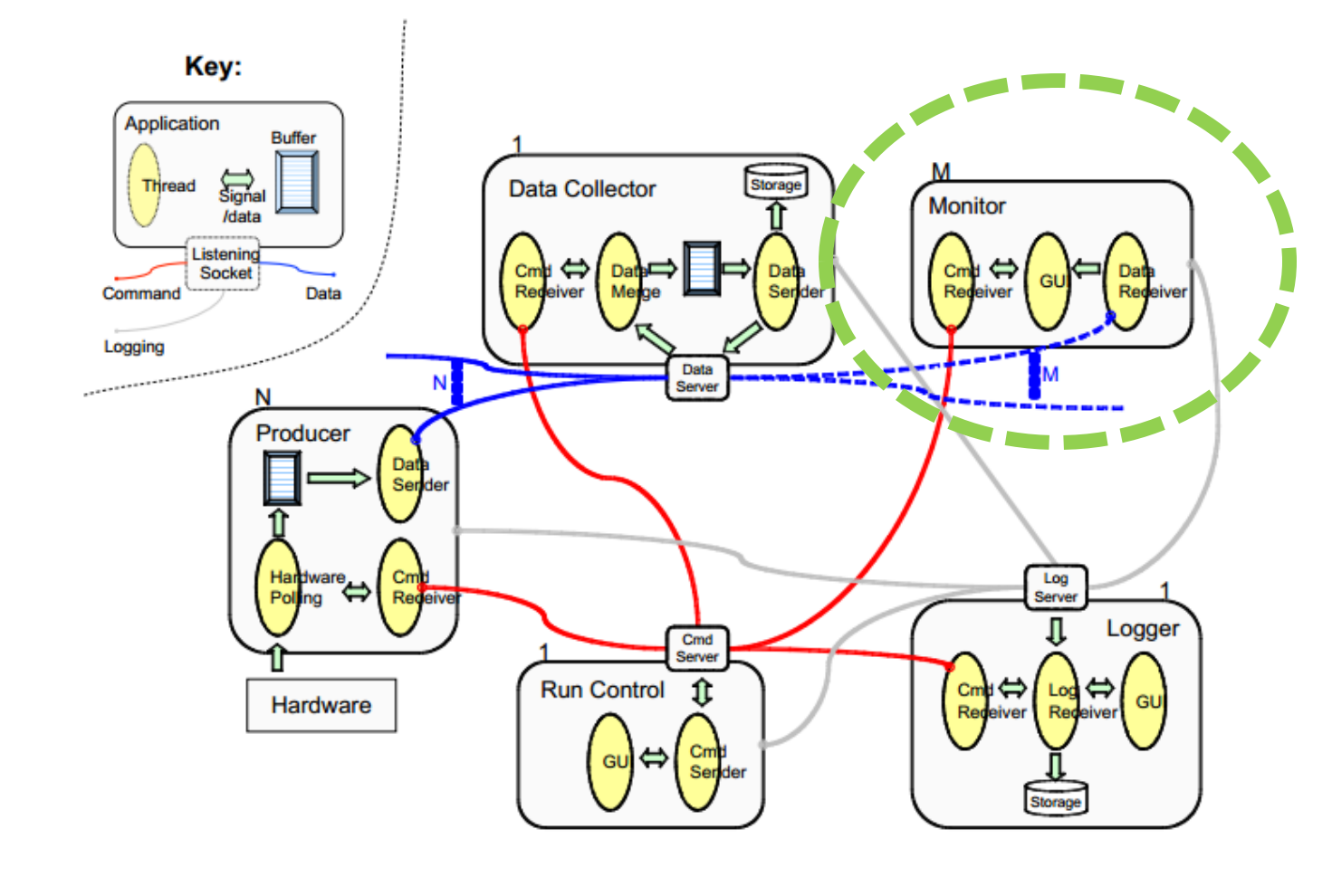


AGH DESY ATLAS EXPERIMENT

Architecture

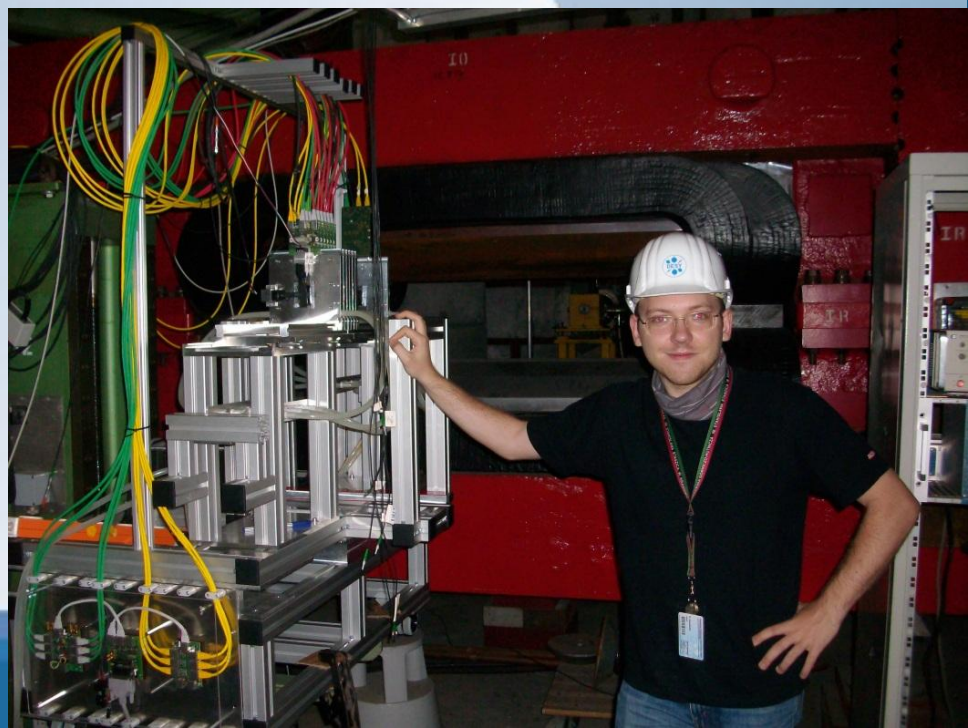


Architecture

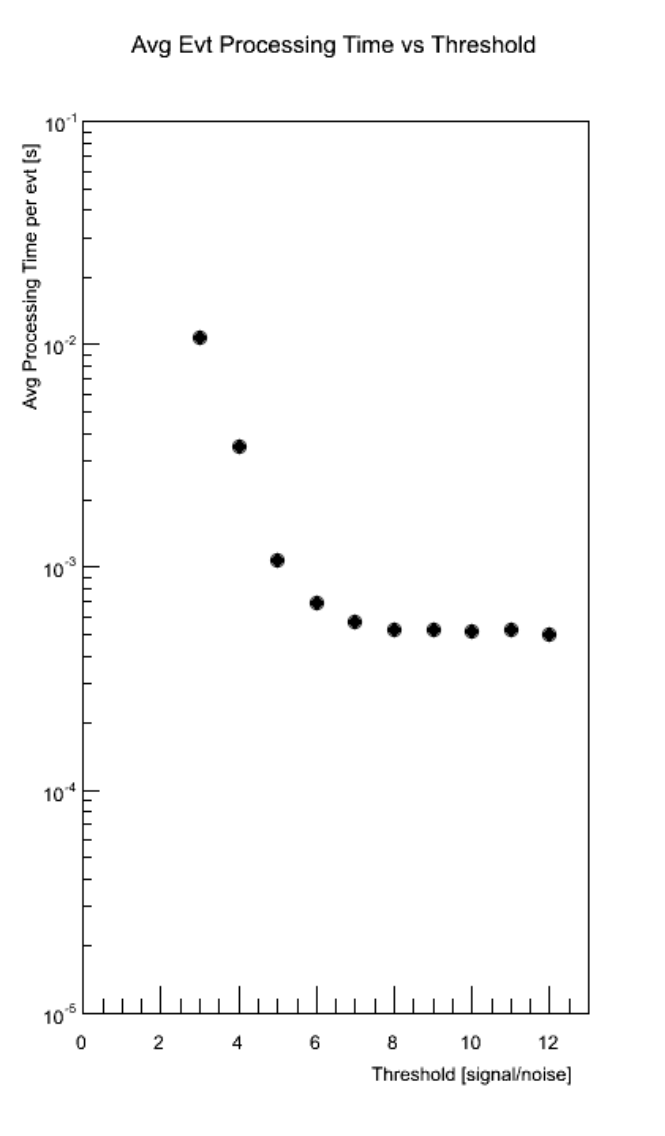


Goals

1) Make the Online Monitor as robust and fast as possible (make it a real-time tool).
2) Add new features for users' convenience.
3) Test it with the real data.

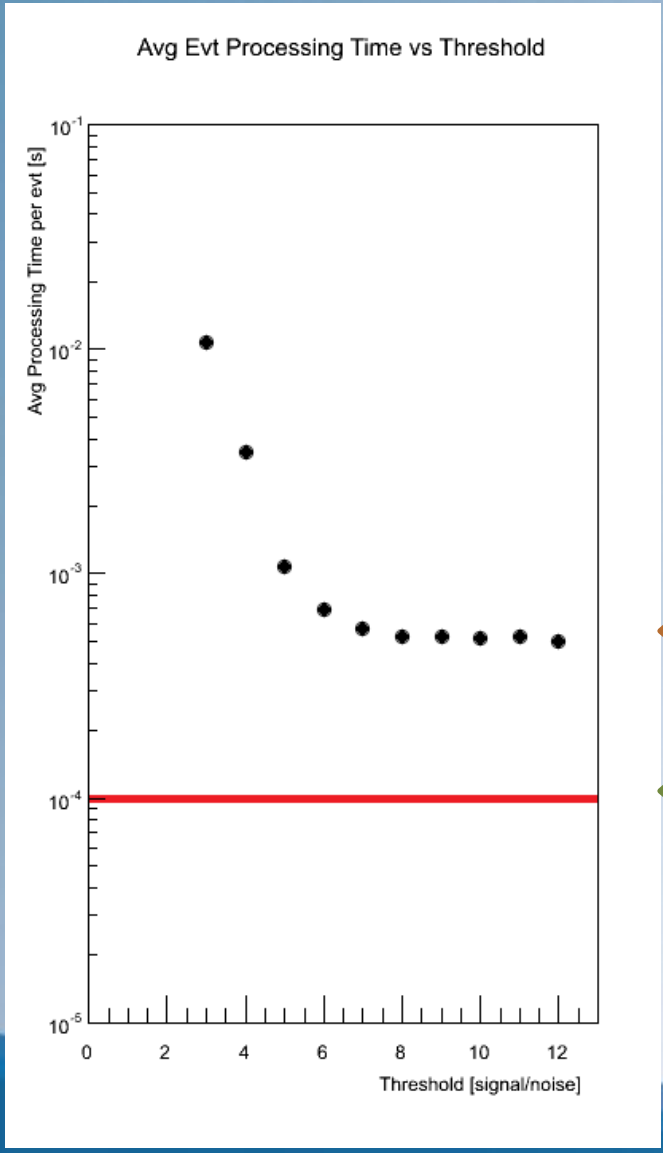


Preliminary analysis



We are here

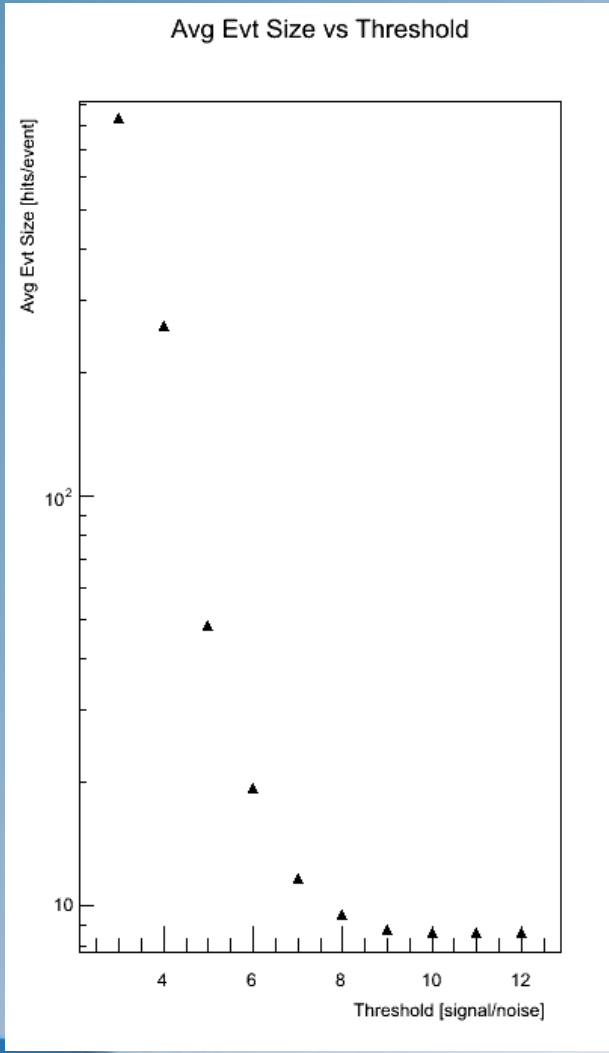
Preliminary analysis



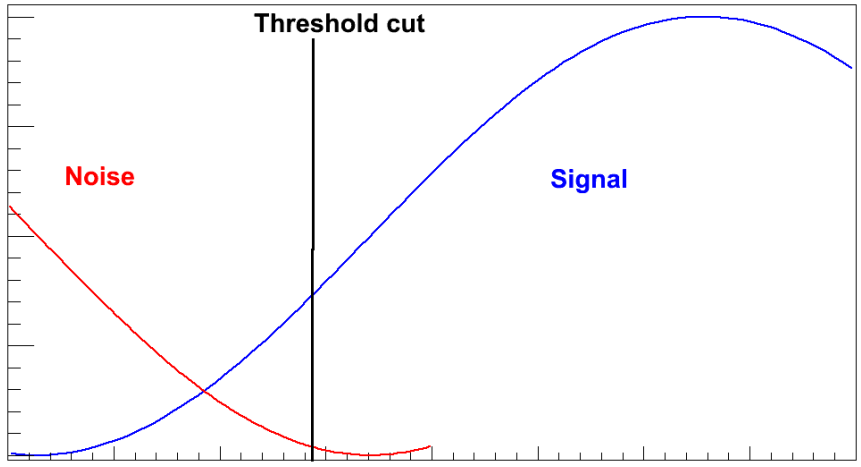
We are here

We want to be here

Preliminary analysis

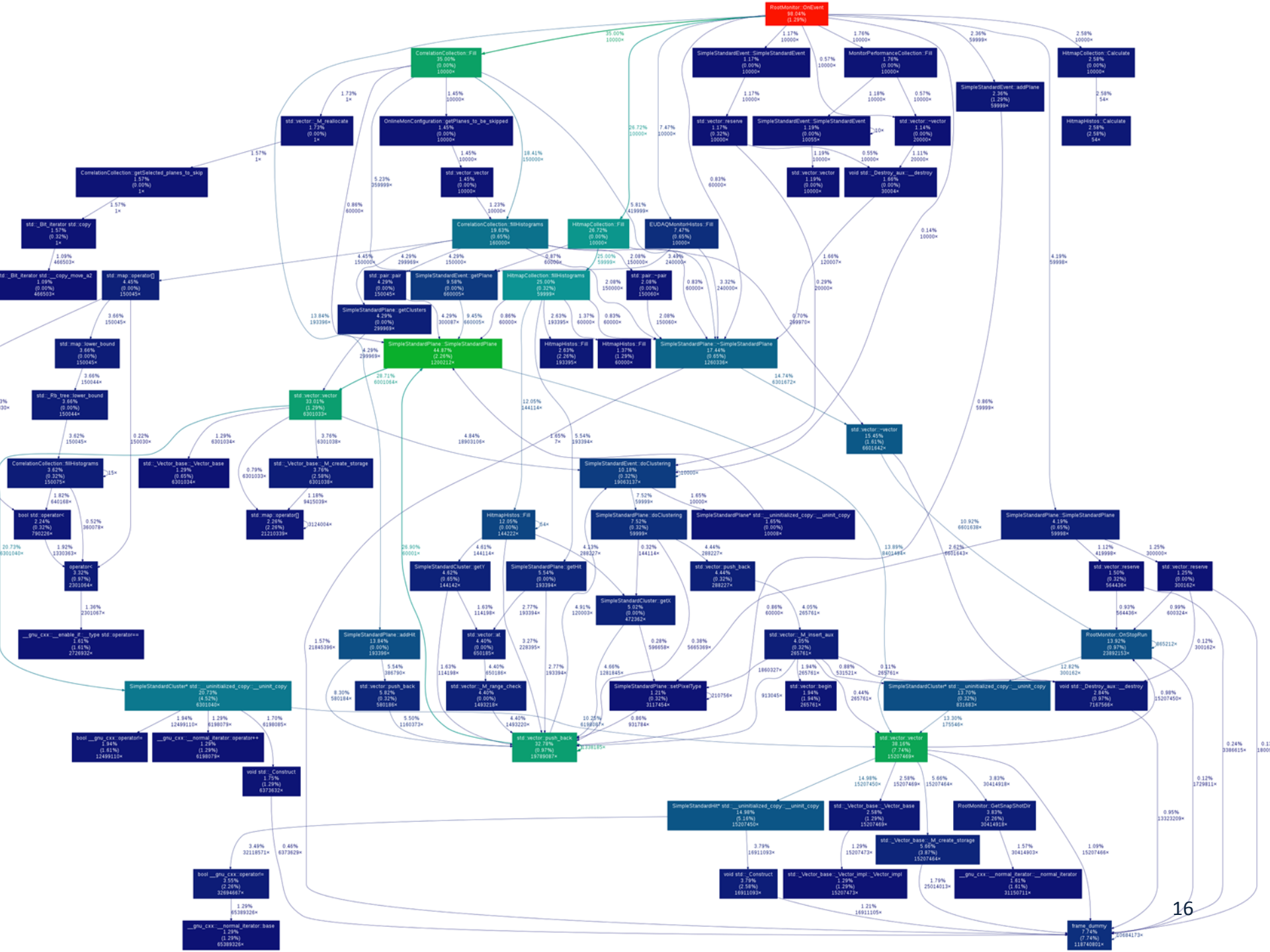


Threshold impacts the event size [hits/event] significantly.



Profiling

- I started with profiling...
  - **Profiling** – analysis of how much time is spent in different functions and how many times they are being called
- ...and I was about to find out that my start would not be very simple...


RootMonitor::OnEvent
98.04%
(1.29%)
CorrelationCollection::Fill
35.00%
(0.00%)
10000×
SimpleStandardEvent::SimpleStandardEvent
1.17%
(0.00%)
10000×
MonitorPerformanceCollection::Fill
1.76%
(0.00%)
10000×
SimpleStandardEvent::addPlane
2.36%
(1.29%)
59999×
HitmapCollection::Calculate
2.58%
(0.00%)
10000×
std::vector::_M_reallocate
1.73%
(0.00%)
1×
OnlineMonConfiguration::getPlanes_to_be_skipped
1.45%
(0.00%)
10000×
std::vector::reserve
1.17%
(0.32%)
10000×
SimpleStandardEvent::SimpleStandardEvent
1.19%
(0.00%)
10055×
std::vector::~vector
1.14%
(0.00%)
20000×
HitmapHistos::Calculate
2.58%
(2.58%)
54×
CorrelationCollection::getSelected_planes_to_skip
1.57%
(0.00%)
1×
std::vector::vector
1.45%
(0.00%)
10000×
std::vector::vector
1.19%
(0.00%)
10000×
void std::_Destroy_aux::__destroy
1.66%
(0.00%)
30004×
std::_Bit_iterator::std::copy
1.57%
(0.32%)
1×
CorrelationCollection::fillHistograms
19.83%
(0.65%)
160000×
HitmapCollection::Fill
26.72%
(0.00%)
10000×
EUDAQMonitorHistos::Fill
7.47%
(0.65%)
10000×
std::_Bit_iterator::std::__copy_move_a2
1.09%
(0.00%)
466503×
std::map::operator[]
4.45%
(0.00%)
150045×
std::pair::pair
4.29%
(0.00%)
150045×
SimpleStandardEvent::getPlane
9.58%
(0.00%)
660005×
HitmapCollection::fillHistograms
25.00%
(0.32%)
59999×
std::pair::~pair
2.08%
(0.00%)
150060×
SimpleStandardPlane::getClusters
4.29%
(0.00%)
299969×
std::map::lower_bound
3.66%
(0.00%)
150045×
SimpleStandardPlane::SimpleStandardPlane
44.87%
(2.26%)
1200212×
HitmapHistos::Fill
2.83%
(2.26%)
193395×
HitmapHistos::Fill
1.37%
(1.29%)
60000×
SimpleStandardPlane::~SimpleStandardPlane
17.44%
(0.65%)
1260336×
std::_Rb_tree::lower_bound
3.66%
(0.00%)
150044×
std::vector::vector
33.01%
(1.29%)
6301033×
std::vector::~vector
15.45%
(1.61%)
6601642×
CorrelationCollection::fillHistograms
3.62%
(0.32%)
150075×
std::_Vector_base::_Vector_base
1.29%
(0.65%)
6301034×
std::_Vector_base::_M_create_storage
3.76%
(2.58%)
6301038×
SimpleStandardEvent::doClustering
10.18%
(0.32%)
19063137×
bool std::operator<
2.24%
(0.32%)
790226×
std::map::operator[]
2.26%
(2.26%)
21210339×
HitmapHistos::Fill
12.05%
(0.00%)
144222×
SimpleStandardPlane::doClustering
7.52%
(0.32%)
59999×
SimpleStandardPlane* std::__uninitialized_copy::__unint_copy
1.65%
(0.00%)
10008×
SimpleStandardPlane::SimpleStandardPlane
4.19%
(0.65%)
59998×
operator<
3.32%
(0.97%)
2301064×
SimpleStandardCluster::getY
4.62%
(0.65%)
144142×
SimpleStandardPlane::getHit
5.54%
(0.00%)
193394×
std::vector::push_back
4.44%
(0.32%)
288227×
std::vector::reserve
1.50%
(0.32%)
564436×
std::vector::reserve
1.25%
(0.00%)
300162×
__gnu_cxx::__enable_if::__type std::operator==
1.61%
(1.61%)
2726932×
SimpleStandardCluster::getX
5.02%
(0.00%)
472362×
SimpleStandardPlane::addHit
13.84%
(0.00%)
193396×
std::vector::at
4.40%
(0.00%)
650185×
std::vector::_M_insert_aux
4.05%
(0.32%)
265761×
RootMonitor::OnStopRun
13.92%
(0.97%)
23892153×
SimpleStandardCluster* std::__uninitialized_copy::__unint_copy
20.73%
(4.52%)
6301040×
std::vector::push_back
5.82%
(0.32%)
580186×
std::vector::_M_range_check
4.40%
(0.00%)
1493218×
SimpleStandardPlane::setPixelType
1.21%
(0.32%)
3117454×
std::vector::begin
1.94%
(1.94%)
265761×
SimpleStandardCluster* std::__uninitialized_copy::__unint_copy
13.70%
(0.32%)
831683×
void std::_Destroy_aux::__destroy
2.84%
(0.97%)
7167566×
bool __gnu_cxx::operator!=
1.94%
(1.61%)
12499110×
__gnu_cxx::__normal_iterator::operator++
1.29%
(1.29%)
6198079×
std::vector::push_back
32.78%
(0.97%)
19789087×
std::vector::vector
38.16%
(7.74%)
15207469×
void std::_Construct
1.75%
(1.29%)
6373632×
SimpleStandardHit* std::__uninitialized_copy::__unint_copy
14.98%
(5.16%)
15207450×
std::_Vector_base::_Vector_base
2.58%
(1.29%)
15207469×
RootMonitor::GetSnapShotDir
3.83%
(2.26%)
30414918×
std::_Vector_base::_M_create_storage
5.66%
(3.87%)
15207464×
bool __gnu_cxx::operator!=
3.55%
(2.26%)
32694667×
void std::_Construct
3.79%
(2.58%)
16911093×
std::_Vector_base::_Vector_impl::_Vector_impl
1.29%
(1.29%)
15207473×
__gnu_cxx::__normal_iterator::__normal_iterator
1.61%
(1.61%)
31150711×
__gnu_cxx::__normal_iterator::base
1.29%
(1.29%)
65389326×
frame_dummy
7.74%
(7.74%)
118740801×

Will structural optimisations be enough? Or new algorithms will have to be developed?

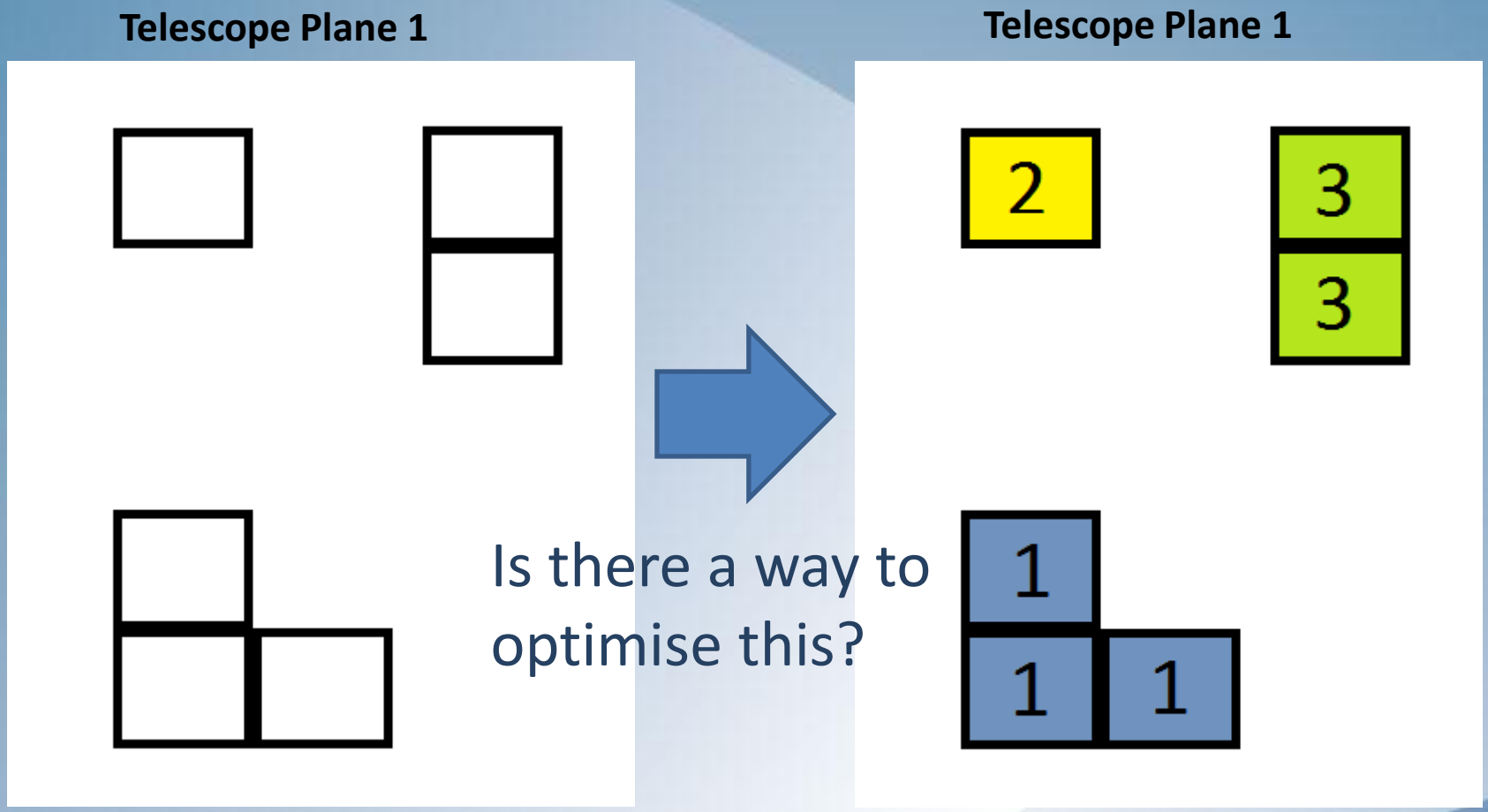
- This version of the Online Monitor was originally developed within the ATLAS pixel test beam group
- It is very good but there is always something to improve,
  so I decided to investigate the algorithms

Previous clusterisation algorithm

- How does the previous clusterization algorithm work?

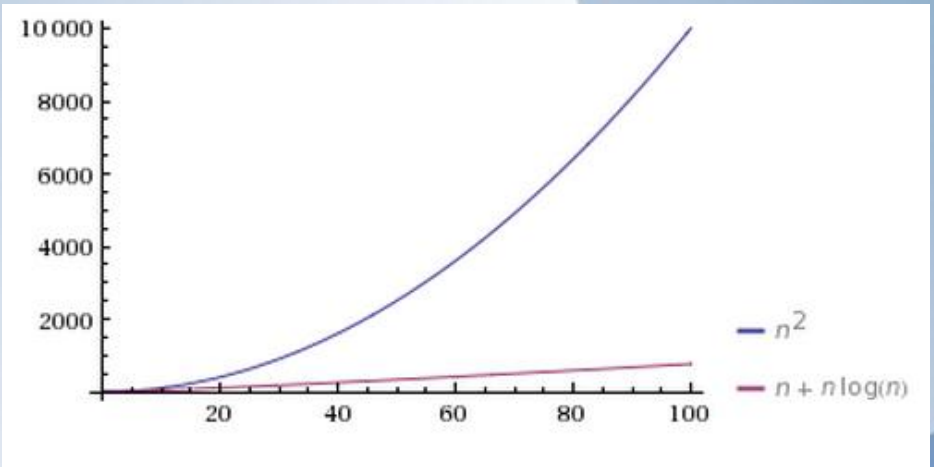
It compares every single hit with the rest from the same plane - $O(n^2)$

How does the previous clusterisation algorithm work?



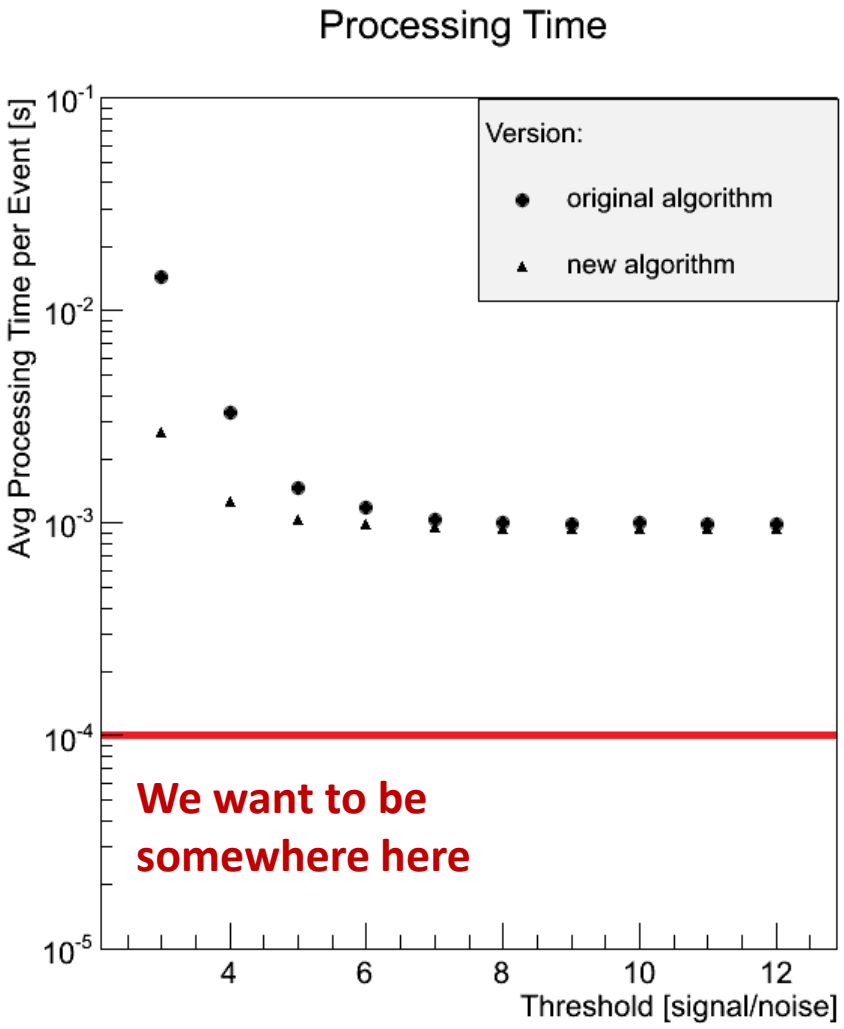
New clusterisation algorithm

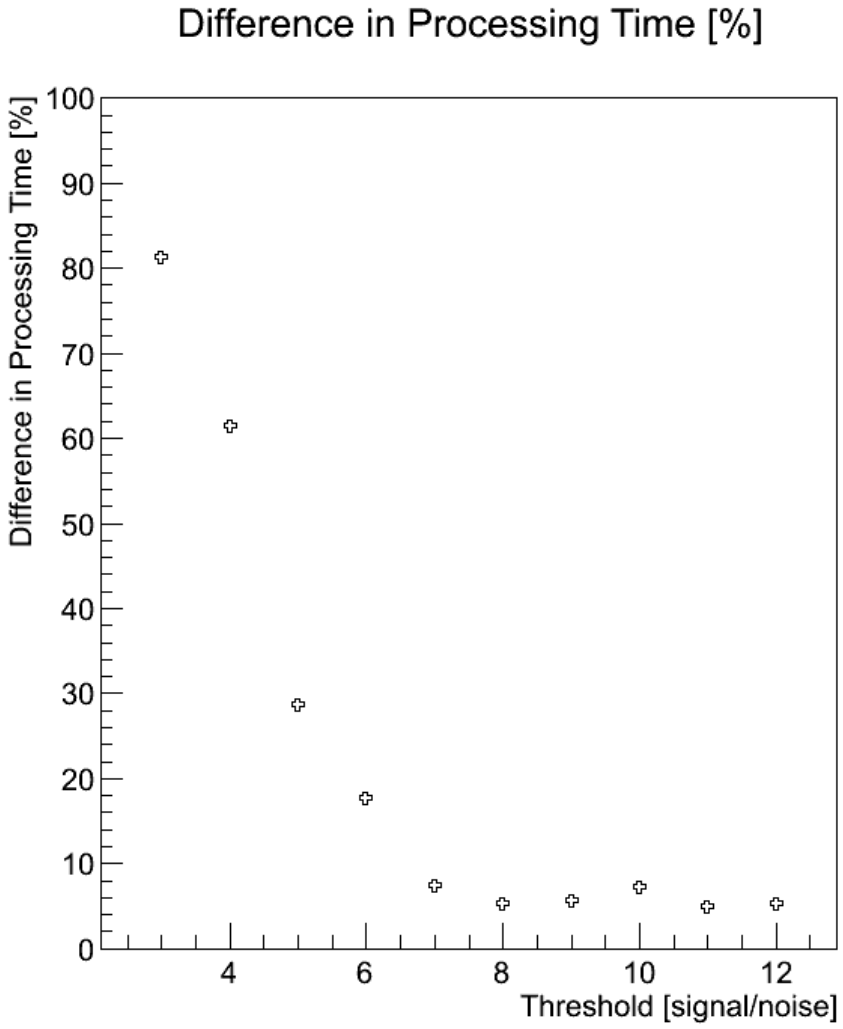
1. *Let's sort all hits*
2. *Now we can look only for the nearest neighbours*
3. *O(n*log(n) + n)*



Performance

- Did it help?



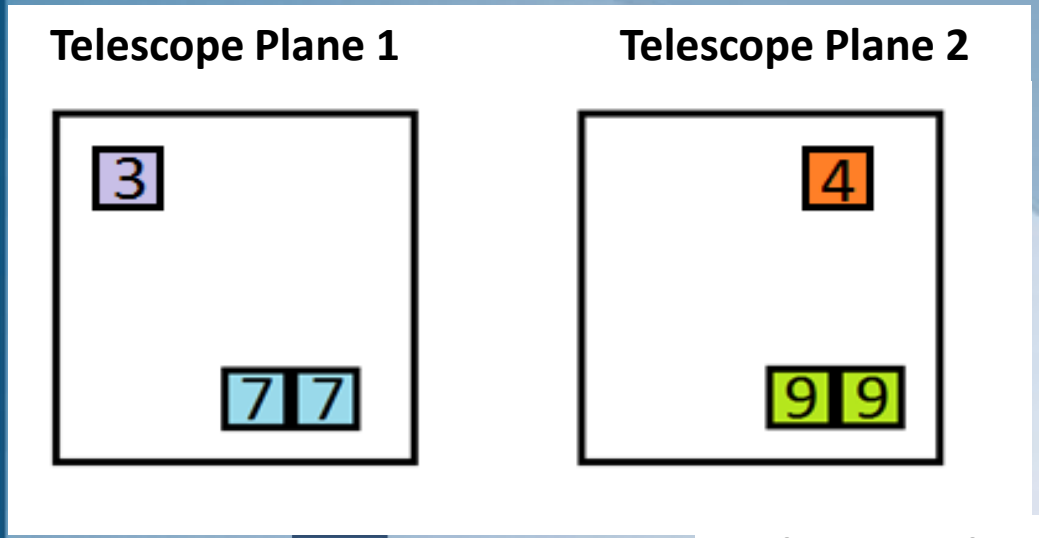


Previous correlation algorithm

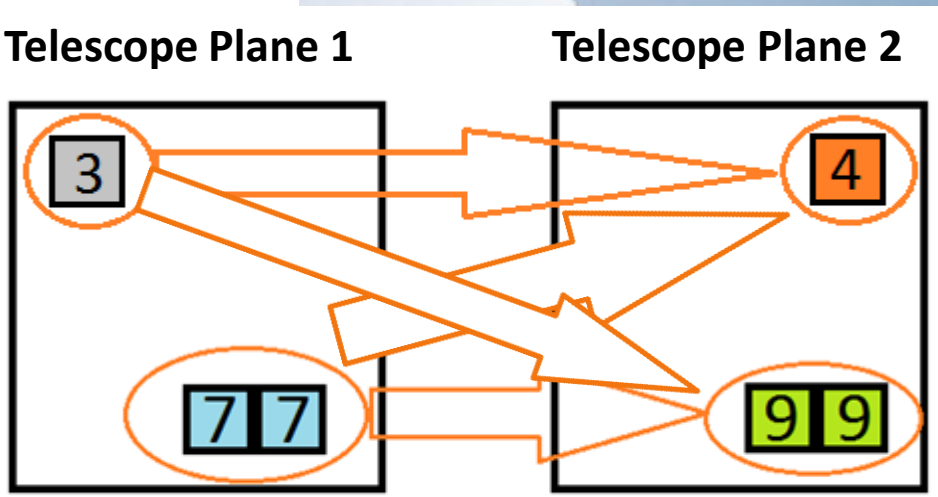
- How does the previous correlation algorithm work?

It correlates all clusters between every two planes - $O(n^2)$

How does the previous correlation algorithm work ?

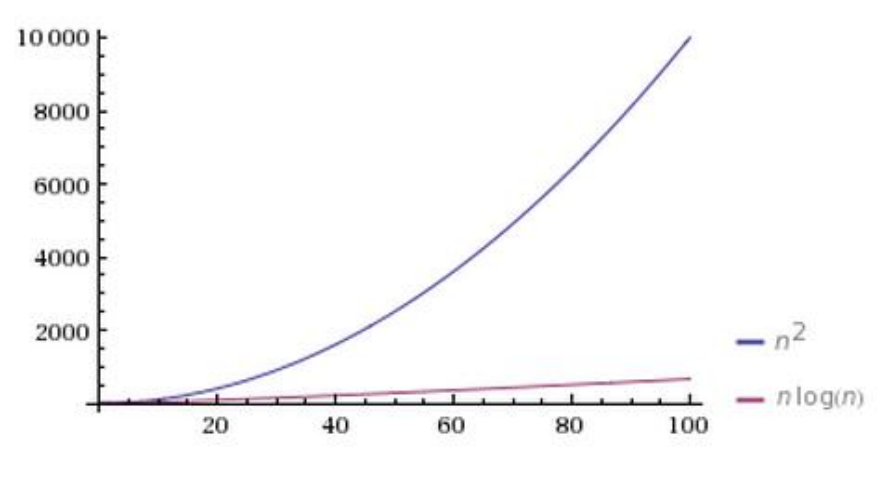


Is there a way to optimise this?

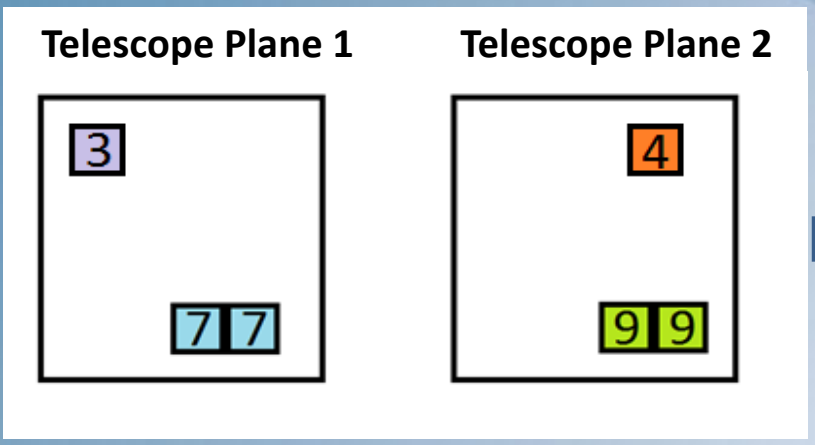


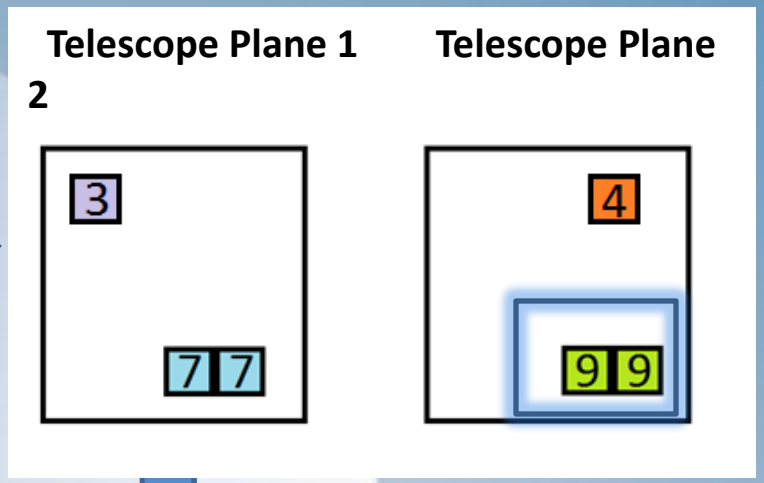
New correlation algorithm

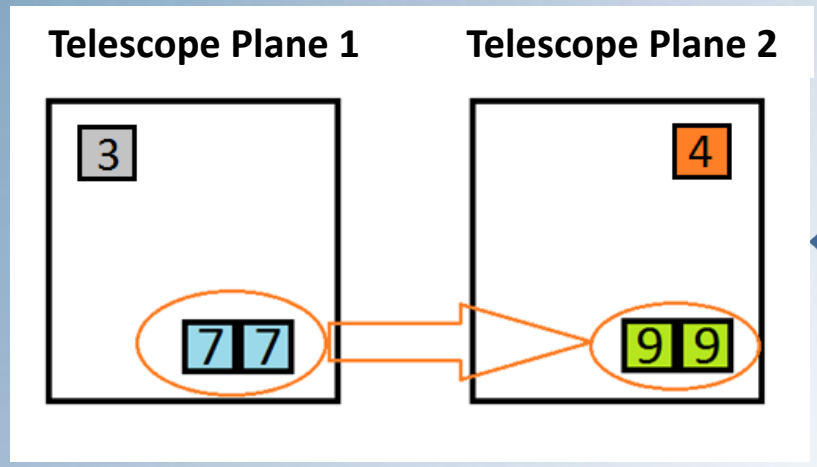
1. *Let's try to reconstruct all tracks first*
2. *Then we could use reconstructed tracks to get information about correlations*
3. *... and in addition to that we now know how many tracks (roughly) we have in a single event*
4. *O(n*log(n))*

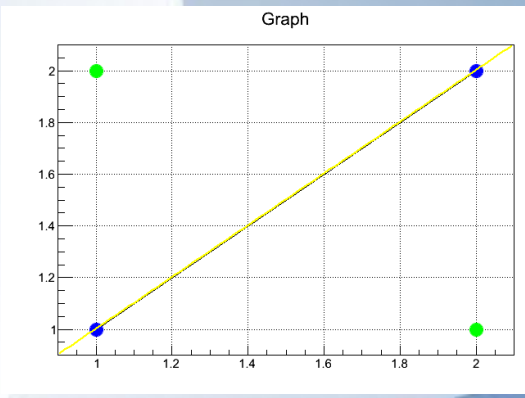


How the new correlation algorithm works ?



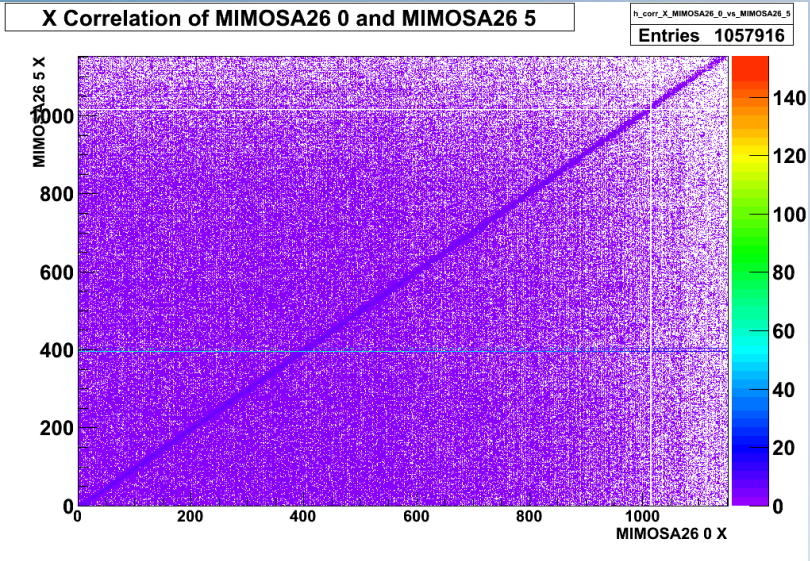






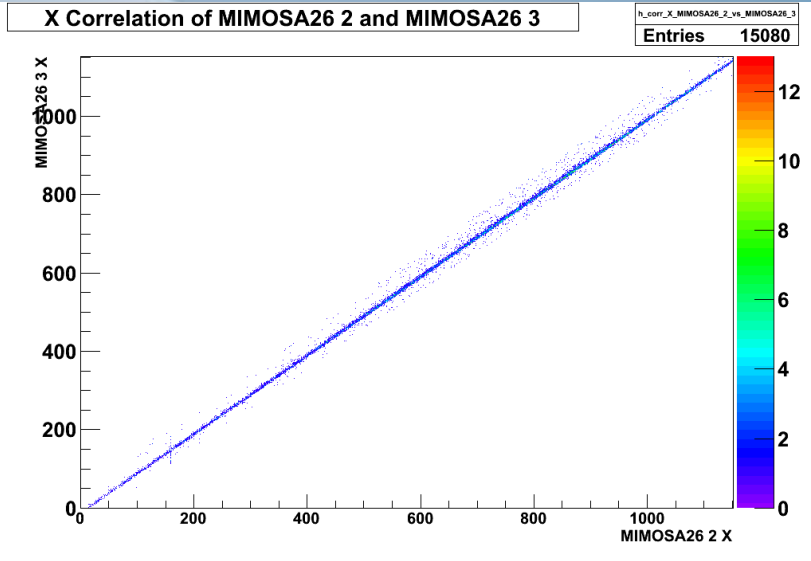
Changes in Plots

Old Correlation Plot



All clusters are being correlated.

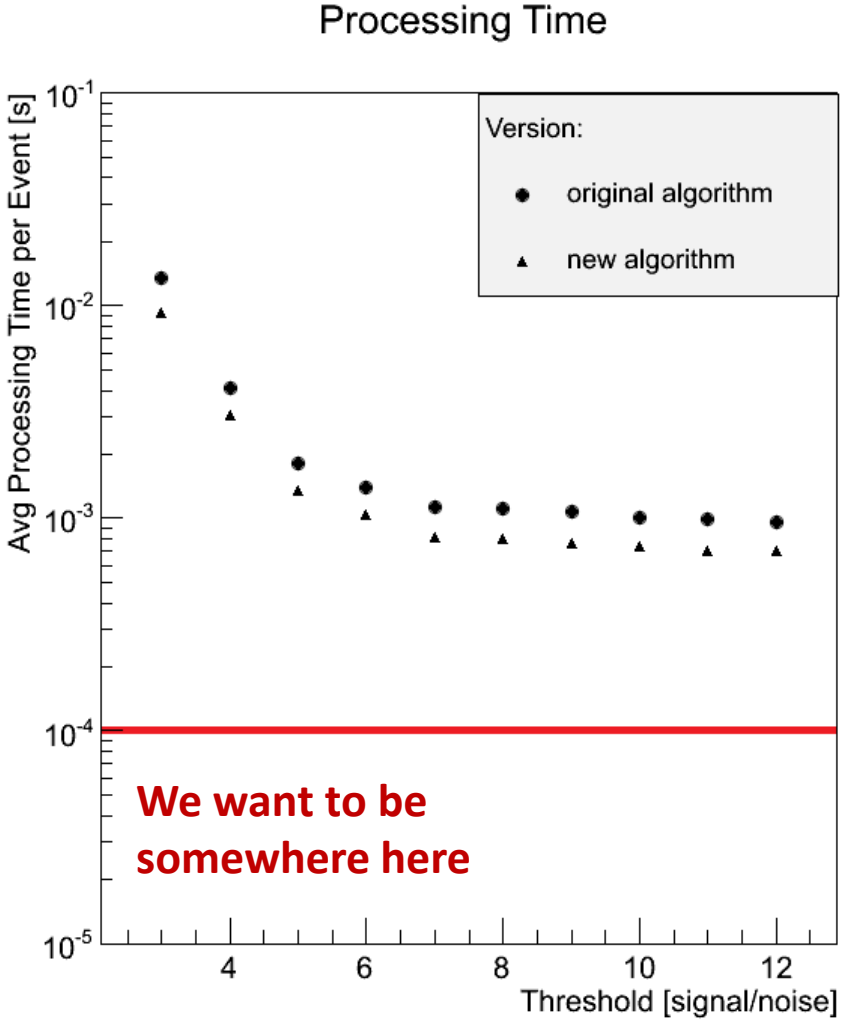
New Correlation Plot

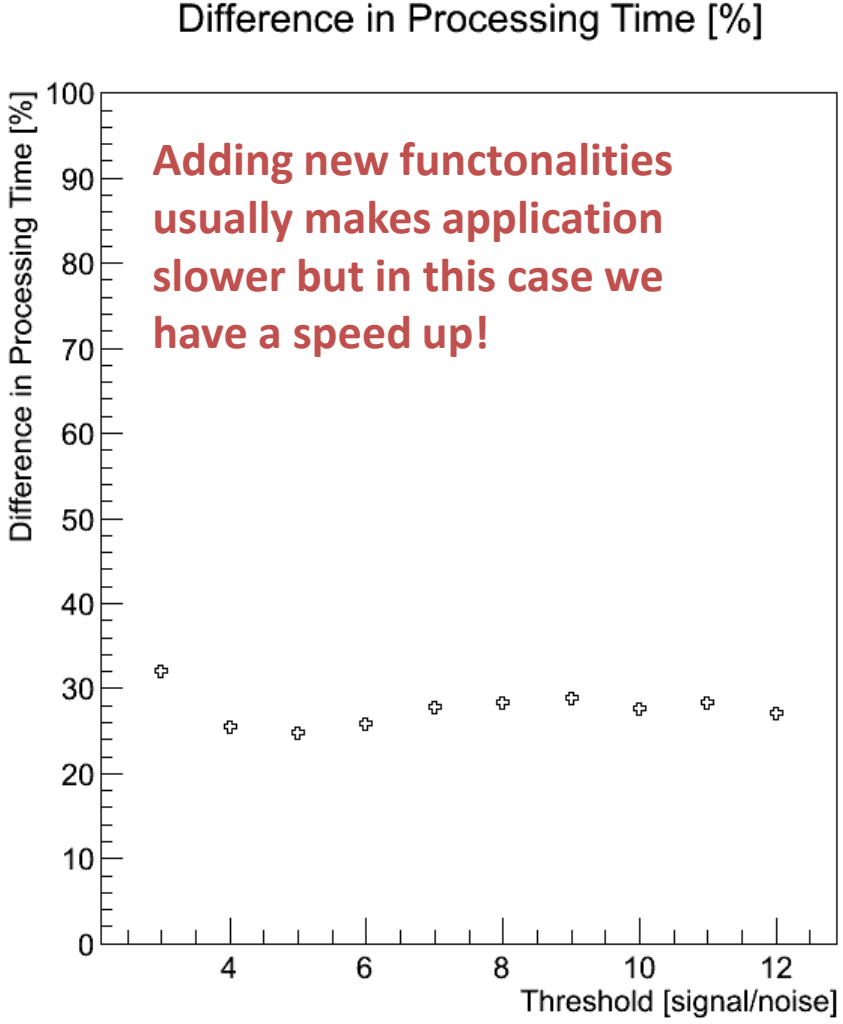


Only clusters with associated tracks are being correlated.

Performance

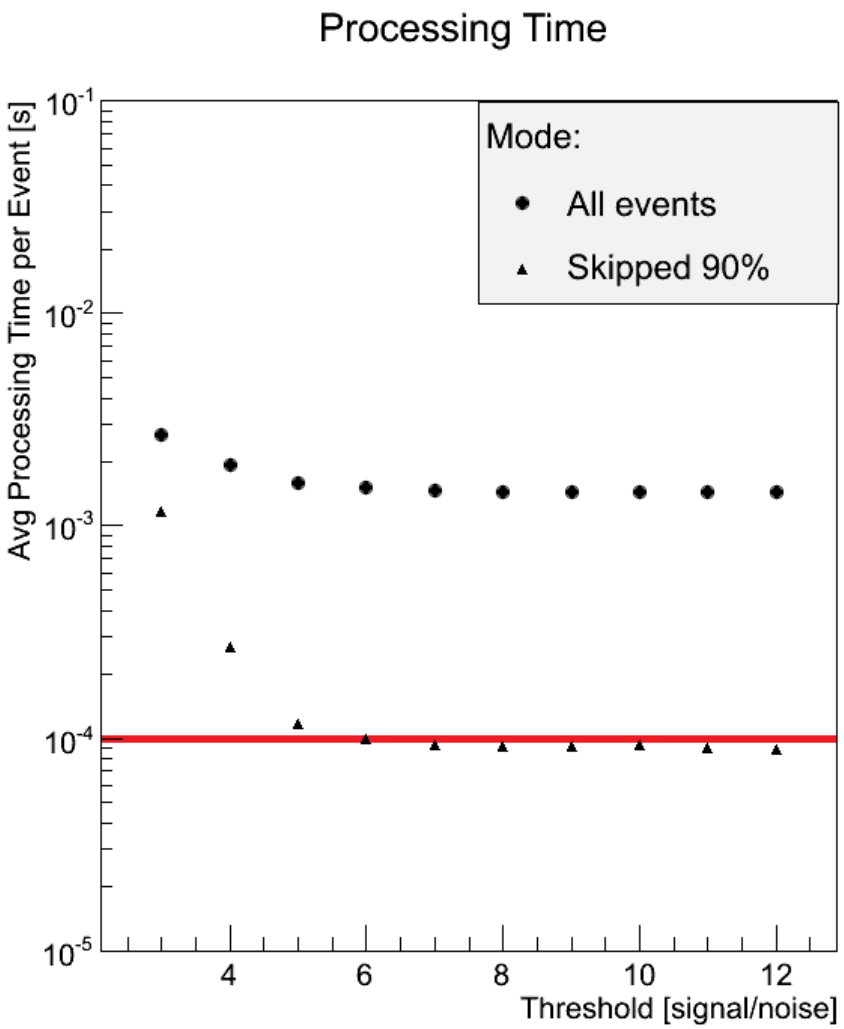
- Did it help? *2GeV beam (2 kHz)*

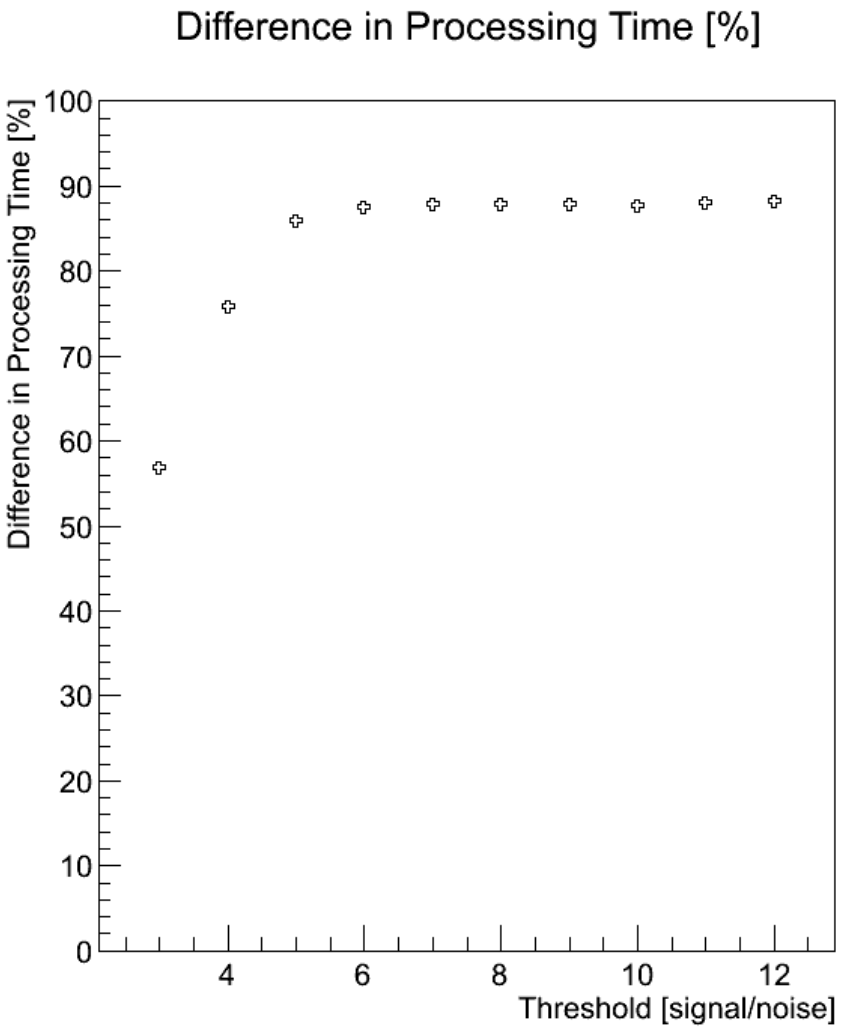




The only solution left was to skip the incoming events.

Performance with skipping 90 % of events





Summary

The Online Monitor:

- became a real-time tool
- has new data quality monitoring functionalities
- conducts self-analysis about its performance

We took data necessary for the performance study of the telescope.

BACKUPS

Tracks per Event plot in the Online Monitor