University of Glasgow | Experimental Particle Physics

LHCb

Configuration using python – feedback from VETRA (a happy user perspective)

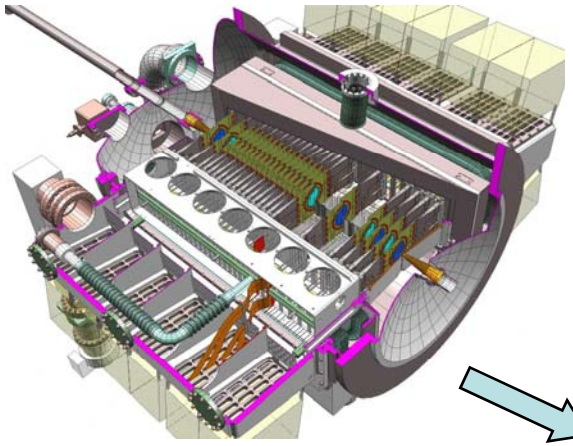
1) VETRA why do we (i.e. VELO) need it?
2) VTERA tasks – modularization
3) Configurable classes for VETRA
4) Options files
5) TED run – real test with real data

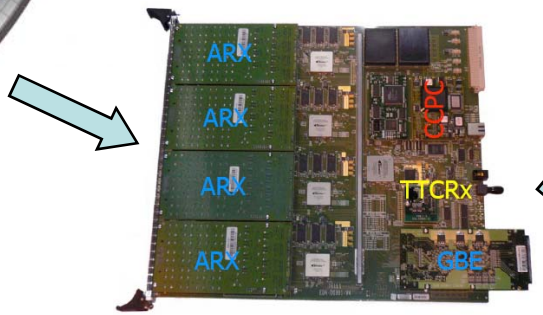
Tomasz Szumlak, Eduardo Rodrigues

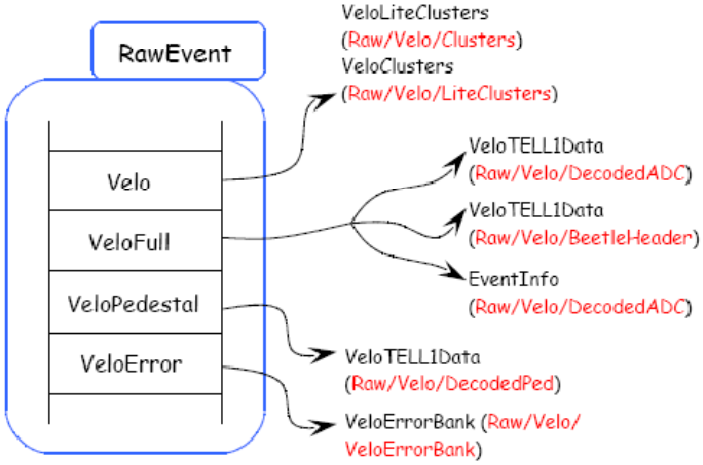
Why

VELO is a very precise and very complicated tool – a lot of things can go wrong with it, thus, we need some means to control it

Let's take a look what we can get out of the VELO







Raw data types

• Zero Suppressed (ZS) Raw Bank – the main input for the tracking reconstruction phase; it contains encoded reconstructed hits (cluster objects). One ZS bank is produced for each active sensor on the acquisition board (TELL1) that handles data from the sensor

•Non-Zero Suppressed (NZS) Raw Bank – contains, so called, full data that was read out of one sensor (2048 samples). This data makes the main input for emulation

•Error Bank (EB) – contains informations on synchronization errors produced by the TELL1 board

•Pedestal Bank (PB) – very similar to the NZS bank, contains 2048 pedestals

•ProcFull Bank (PF) – specialized bank for debugging purposes, identical with ZS bank except no thresholds are applied during the clusterization process

TELL1 Emulation

Calibration Data stream needed to optimize processing

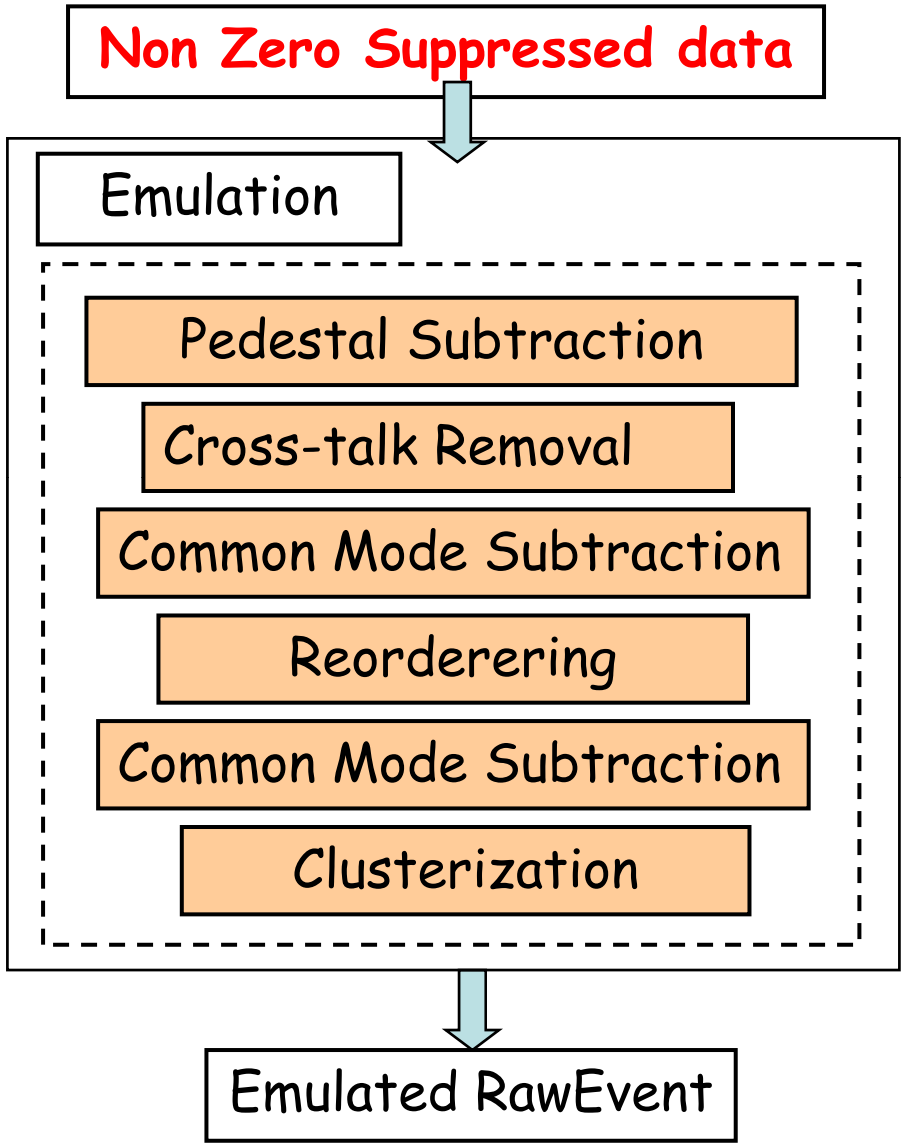
Non-zero suppressed data at 0.1 Hz rate

Run emulation of TELL1 board processing off-line to optimise algorithm performance

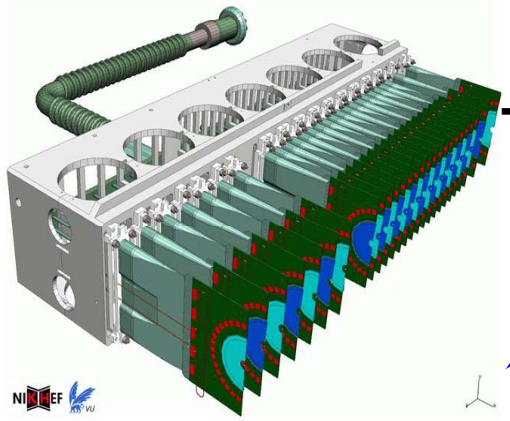
6 algorithms
~ 10^6 parameters
Bit perfect emulation of FPGA performance

Complex processing
Strip lengths, radiation damage…

Parameters used for readout board fixed in emulation



Processing parameters



Noise Data

VELOCOND

Gaudi Job (Vetra)

Tell1 Emulation

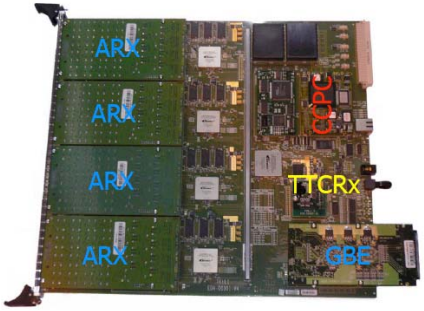
Monitoring/
verification job

Analysis

Tuned Processing Parameters

XML file

XML2PVSS



Tell1

VETRA

All of this can be done using VETRA project

At the moment this is a joint-venture of VELO and ST

I will concentrate on the VELO part only

There is a LHCb note available: CERN-LHCb-2008-022
The official web page: http://lhcb-release-area.web.cern.ch/LHCb-release-area/DOC/vetra/

VETRA

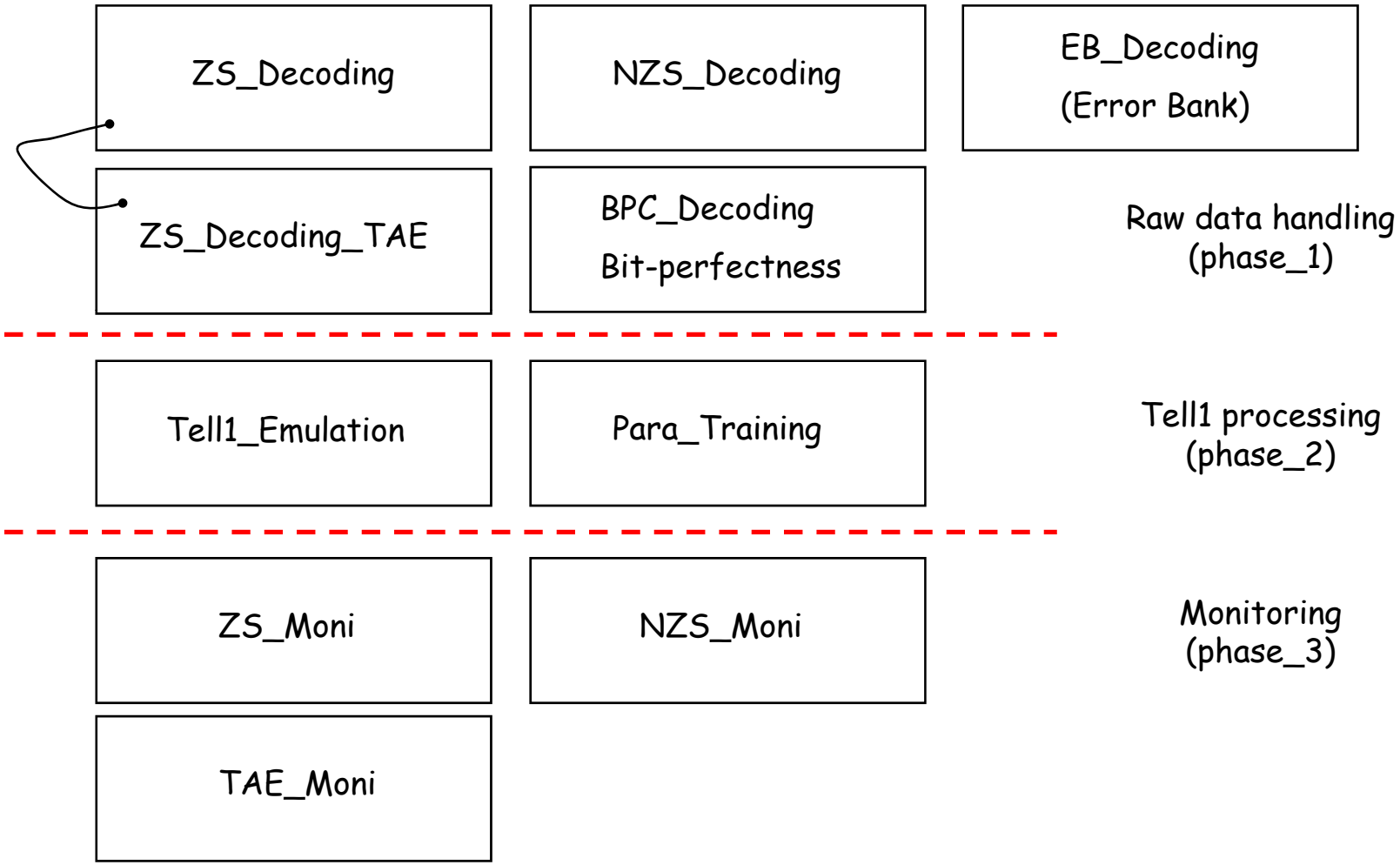
There are great expectation for VETRA – wide range of application

- ZS data analysis/monitoring
- Full bank analysis/monitoring/emulation
- Error bank analysis
- Pedestal monitoring
- Bit-perfectness checking
- The Tell1 processing parameters training/verification
- Time alignment of the VELO
- Beetle headers analysis (and the X-Talk)

…
- this list goes on…

Vetra is not 'linear application', all of these tasks requires quite different configuration – try to come up with a number of 'modules' that can be used to build any task – this way we can shield end-user from a serious head ache (i.e., what's inside VETRA)

VETRA – basics tasks – a model



Implementation

Based on the simple task model we implemented a number of configurables that can be used to configure any composite task
Description of the model and implementation can be found at:

E. Rodrigues, T. Szumlak LHCb-INT-2009-006

VETRA configurable classes:

1) The big boss – Configuration.py
2) VetraTELL1ProcessingConf.py (decoders, emulation)
3) VetraTELL1ParaTraining.py (the processing parameters tuning)
4) VetraMoniConf.py (monitors)
5) VetraOutputConf.py (want to write digi and run tracking?)

Implementation


```
# Steering options
__slots__ = { 'DataType' : '2008' ,
              'DataBanks' : 'NZS+ZS' ,
              'Context' : 'Offline' ,
              'DDDBtag' : '' ,
              'CondDBtag' : '' ,
              'Simulation' : False ,
              'RunEmulation' : False ,
              'EmuMode' : 'Static' ,
              'EmuConvLimit' : 4096 ,
              'CheckEBs' : False ,
              'CheckTAEs' : False ,
              'ParaTraining' : False ,
              'BitPerfectness' : False ,
              'OutputType' : 'NONE' ,
              'HistogramFile' : 'Vetra_histos.root' ,
              'EvtMax' : -1 ,
              'SkipEvents' : 0 ,
              'MainSequence' : [ 'ProcessPhase/TELL1Processing',
                                 'ProcessPhase/Moni',
                                 'ProcessPhase/Output' ]
            }
```

Problems handling

Is that all? Let's be smart and use the python actually…

Two more classes to make things nicer:

Check options – consistency checking – if someone wants to do some nasty things (e.g., wrong option name) an exception will be thrown:

VetraOptionError

Handle the exceptions – should there is problem with the options – handle the exception accordingly:

VetraRuntimeError

Checker in action

```
from Configurables import Vetra
from Configurables import DecodeVeloFullRawBuffer, VeloTELL1Reordering

from VeloDAQ.DefaultVeloRawBufferDecoders import DefaultDecoderToVeloClusters

vetra = Vetra()

vetra.DataBanks     = 'MZS'
```

```
# applying configuration of Vetra
 Allowed values for job option "DataBanks" :
['NZS', 'ZS', 'NZS+ZS', 'ProcFull', 'NONE']

Traceback (most recent call last):
  File "/afs/cern.ch/sw/Gaudi/releases/GAUDI/GAUDI_v21r0/InstallArea/scripts/gaudirun.py", line 106, in <module>
    applyConfigurableUsers()
  File "/afs/cern.ch/sw/Gaudi/releases/GAUDI/GAUDI_v21r0/InstallArea/python/GaudiKernel/Configurable.py", line 1295, in appl
yConfigurableUsers
    c.__apply_configuration__()
  File "/afs/cern.ch/sw/Gaudi/releases/GAUDI/GAUDI_v21r0/InstallArea/python/GaudiKernel/Configurable.py", line 1236, in __ap
ply_configuration__
    return self.applyConf()
  File "/afs/cern.ch/user/s/szumlat/cmtuser/Vetra_v7r1/InstallArea/python/Vetra/Configuration.py", line 173, in applyConf
    self.checkOptions()
  File "/afs/cern.ch/user/s/szumlat/cmtuser/Vetra_v7r1/InstallArea/python/Vetra/Configuration.py", line 100, in checkOptions
    optionsChecker( 'DataBanks'  , self.getProp( 'DataBanks'  ) )
  File "/afs/cern.ch/user/s/szumlat/cmtuser/Vetra_v7r1/InstallArea/python/Vetra/VetraOptionsChecker.py", line 82, in options
Checker
    raise VetraOptionError( name, value )
Vetra.VetraOptionsChecker.VetraOptionError: invalid Vetra "DataBanks" option value specified: MZS
```

Composite task

A typical composite VETRA task would be to run a job over a mdf file with NZS and ZS data, decode and monitor the ZS data (clusters), run pedestal training and the emulation and monitor the noise at different processing stages

How would the options file look like? (skiped module import lines)

```
vetra = Vetra()

vetra.RunEmulation = True

EventSelector().PrintFreq = 100

# set the appropriate DB tags
LHCbApp().DDDBtag   = 'head-20080905'
LHCbApp().CondDBtag = 'head-20080905'
```

Looks easy and comprehensible…

Some 'cool' features

Some tasks are very similar, except a really tiny detail – no problem with python options whatsoever!

```
# no "prepare" algorithm for simulation !
if Vetra().getProp( 'Simulation' ) :
    decoding_NZS.Members.remove( prepareVeloFullRawBuffer )
```

For many tasks we need to specify, so called convergence limit, its value depends on what we want to do and needs to be distributed to many places, again, no problem with python

```
# --> convergence limit, this will be propagated to all TELL1 algos
VeloTELL1EmulatorInit().ConvergenceLimit = Vetra().getProp( 'EmuConvLimit' )
```

Using python for the '09 TED run

TED run 6th and 7th June
Shots on the TED beam dump provide real tracks that in turn can be use to calibrate the VELO

Helps the VELO to be ready for the collision data taking

Almost the last opportunity to test the hardware/software before the LHC start

During the last year TED runs we were able to collect ~2000 tracks
In the proposal for the 2009 TED we were dreaming about having ~10 k real tracks (the processing paramters tuning needed)

Actually we were able to get almost 60 k

Great success of the TED data taking for the VELO!

Using python for the '09 TED run

A very important lesson that we got during 2008 TED - proper configuration is critical for stable run

With so many goals to achieve and no less people around eager to do stuff we had a real mess with options files in no time (a large number of specific tasks)

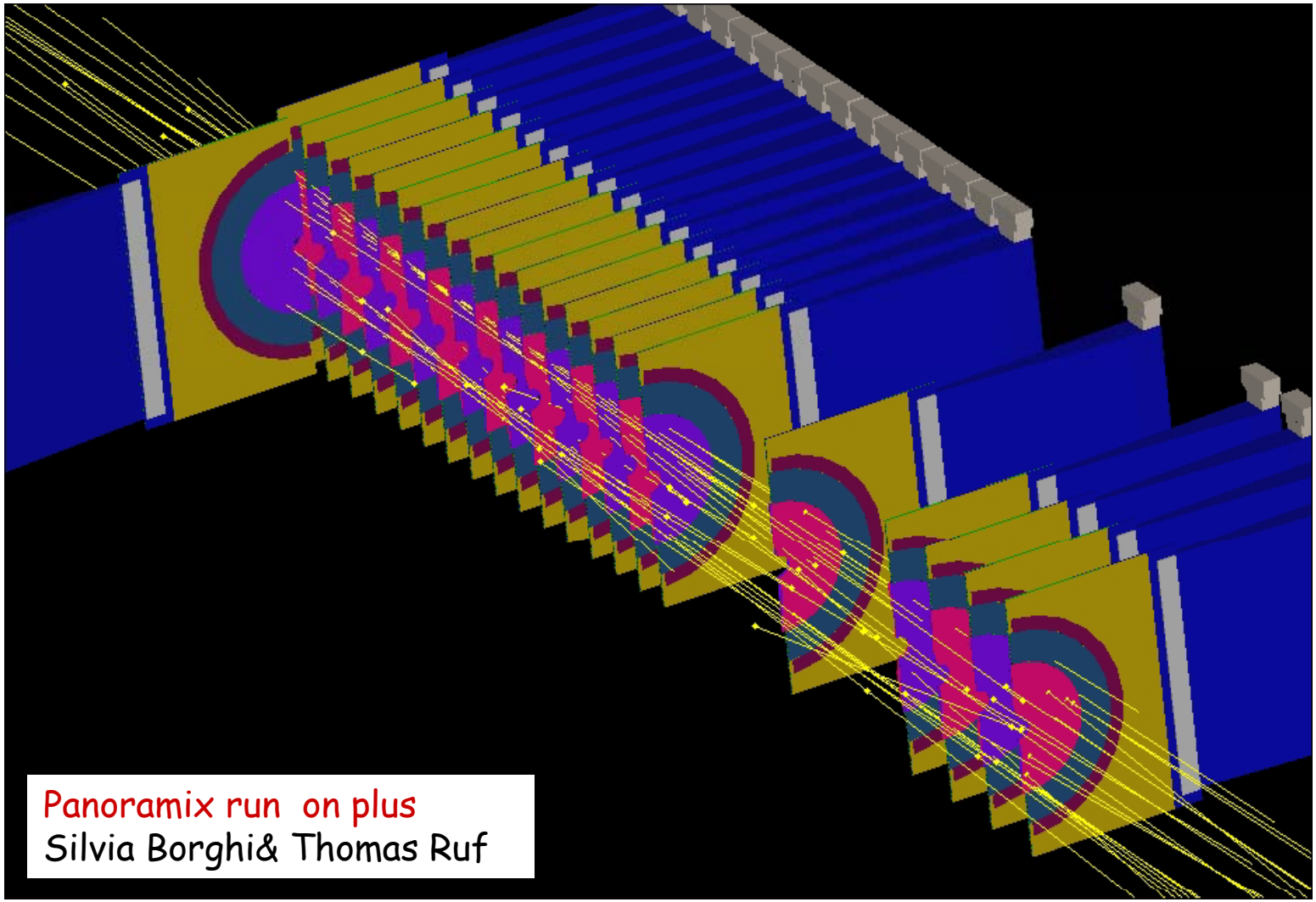
A lot of confusion – data base tags, adding/removing algorithms, etc.

This time we took our time to design something smarter - python configuration framework turned out to be a real help

We created all the needed specific options in 20 minutes and nobody was thinking about them for the next two days

That stability was really surprising (and nice of course) – full off-line analysis of the data was available minutes after each file was taken

Using python for the '09 TED run



Panoramix run on plus
Silvia Borghi& Thomas Ruf

Final remarks

VETRA project options have been pythonized
The python configurables provide very efficient and flexible framework
Allows to configure easily complicated tasks using the power of the python languge
End-user options files are easy to read/create

All in all we had a very positive experience with the configurables

A significant contribution to the successfull TED run!