A users view on exploiting the control system in MDs

G. Iadarola with input from:

G. Arduini, T. Argyropoulos, V. Baggiolini, H. Bartosik, R. De Maria,

J. Gonzalez Cobas, K. Fuchsberger, M. Hostettler, T. Levens,

E. Metral, T. Persson, R. Tomas, M. Solfaroli Camillocci,

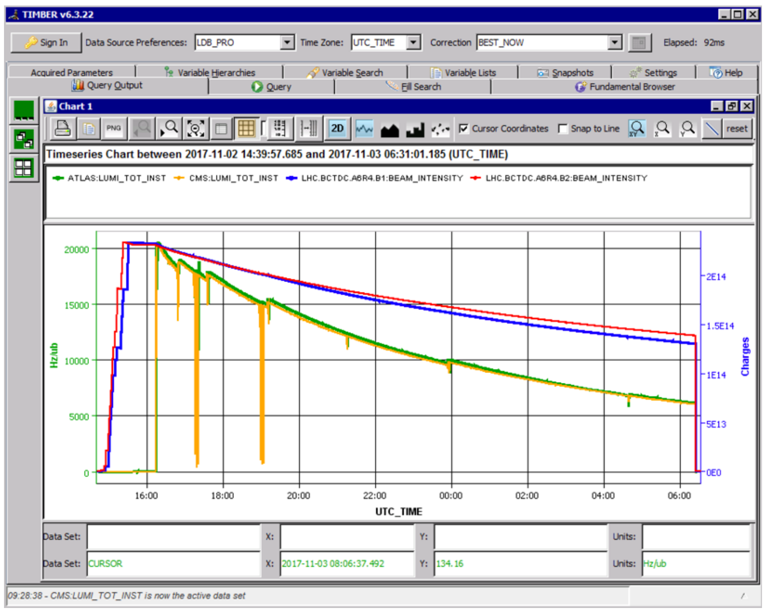
G. Sterbini, J. Wenninger

8th Evian Workshop - December 2017

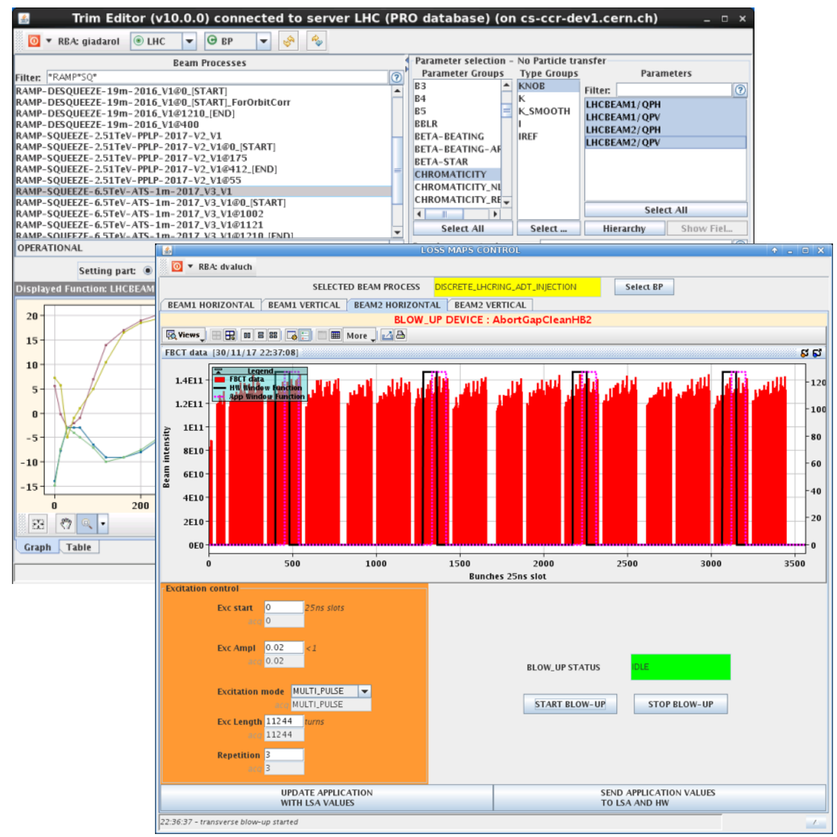
Introduction

Focusing on two types of activity

Extraction and analysis of data from the logging service (CALS)

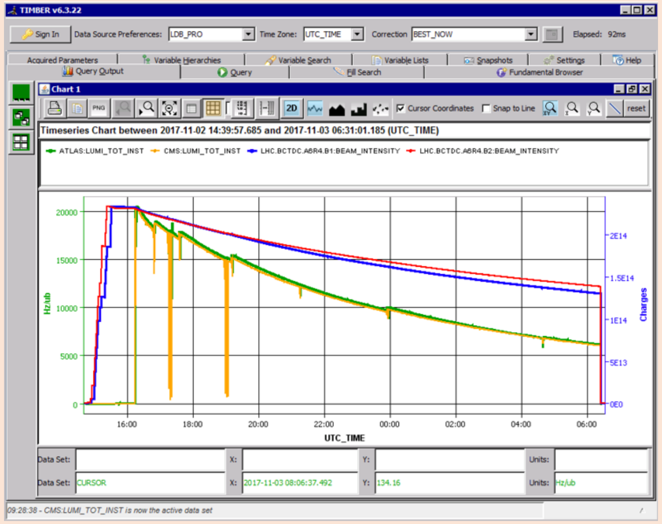


Interaction with LHC equipment



Analysis of logged data: the “classical paradigm”

1. **Extract the data** from Logging System and store on local drive (e.g. using Timber)

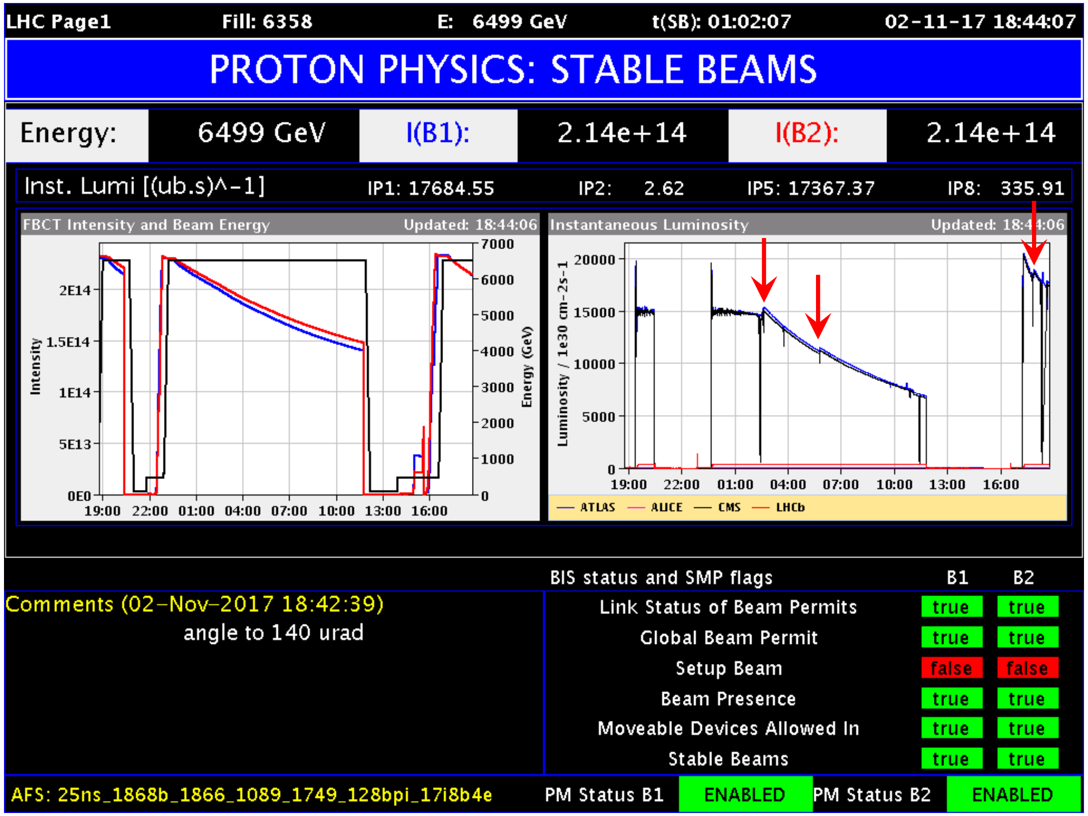


2. **Parse** downloaded file
3. **Perform “the analysis”** (data manipulation, correlations, plotting, etc.)

But this does not work efficiently when the analysis needs to interact with the data extraction…

A practical example: analysis of Xing angle anti-leveling

- For **all physics fills in 2017** we want to know:
  - After how much **time in Stable Beams** (SB) the **first angle reduction** was applied
  - By **how much** was the crossing angle changed
  - What was the **average bunch intensity** at the moment of the change
- With the approach described before we would need to **download** and **store** the required data for the **entire run**, **parse** the files, do the **processing** → **>1 day** of work



A practical example: analysis of Xing angle anti-leveling

- For **all physics fills in 2017** we want to know:
  - After how much **time in Stable Beams** (SB) the **first angle reduction** was applied
  - By **how much** was the crossing angle changed
  - What was the **average bunch intensity** at the moment of the change
- With the approach described before we would need to **download** and **store** the required data for the **entire run**, **parse** the files, do the **processing** → **>1 day** of work
- Of course this can be made much **more efficient** with an **intelligent data retrieval**

A possible algorithm combining **data retrieval** and with **some logics**

For all fills of the 2017 p-p run:
- Download beam modes info
- Check if SB declared (if not skip fill)
- Get start-end of SB timestamps
- Get crossing angle at start SB
- Get crossing angle during SB
- Identify time of first step (t_{step})
- Save Xing angle before and after t_{step}
- Extract beam intensity at t_{step}
- Extract number of bunches
- Compute average bunch intensity
- Save average bunch intensity at t_{step}

A lot less data: download one intensity point per fill instead of the full run!

But this cannot be done with Timber…

A practical example: analysis of Xing angle anti-leveling

- For **all physics fills in 2017** we want to know:
  - After how much **time in Stable Beams** (SB) the **first angle reduction** was applied
  - By **how much** was the crossing angle changed
  - What was the **average bunch intensity** at the moment of the change
- With the approach described before we would need to **download** and **store** the required data for the **entire run**, **parse** the files, do the **processing** → **>1 day** of work
- Of course this can be made much **more efficient** with an **intelligent data retrieval**

A possible algorithm combining **data retrieval** and with **some logics**

For all fills of the 2017 p-p run:
- Download beam modes info
- Check if SB declared (if not skip fill)
- Get start-end of SB timestamps
- Get crossing angle at start SB
- Get crossing angle during SB
- Identify time of first step (t_{step})
- Save Xing angle before and after t_{step}

A lot less data: download one intensity point per fill instead of the full run!

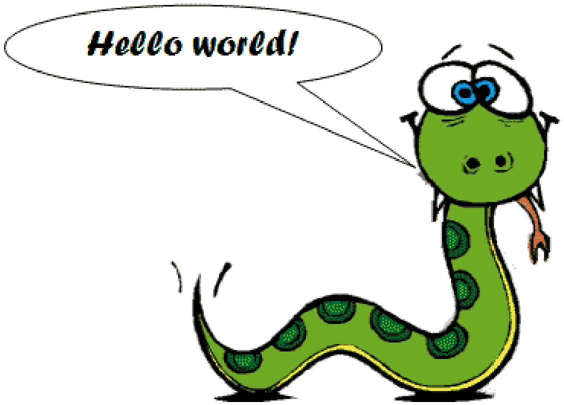
But this cannot be done with Timber…

- The conventional/supported way of interacting with the logging system with full flexibility is to use the **Java API** (Application Program Interface) provided by BE-CO
- Unfortunately Java is not part of the background for the average physicist and has a quite long learning curve…

An interesting middle ground: Python

Python is an open-source general-purpose language and is a **very popular choice for scientific computing** (e.g. data analysis, calculations, numerical simulations)

→ In fact many of us are already using it



- Very **fast learning curve**
- Simple, flexible, and **concise** → development generally faster than with other languages: ideal for quick **prototyping**, **testing** new ideas
- Prone to **interactive development**
- Used and supported by a **large community** (if you have a question, you just have to write it on google)
- It comes with solid and complete set of **tools for numerical analysis and plotting** (numpy, scipy, matplolib, pandas)

"Hello world" in Java

```
public class HelloWorld
{
  public static void main(String[] args)
   {
     System.out.println("Hello, World!");
   }
}
```

"Hello world" in Python

```
print "Hello, World!"
```

PyTimber

Users developed a python module (**PyTimber**) to **access the logging via python**

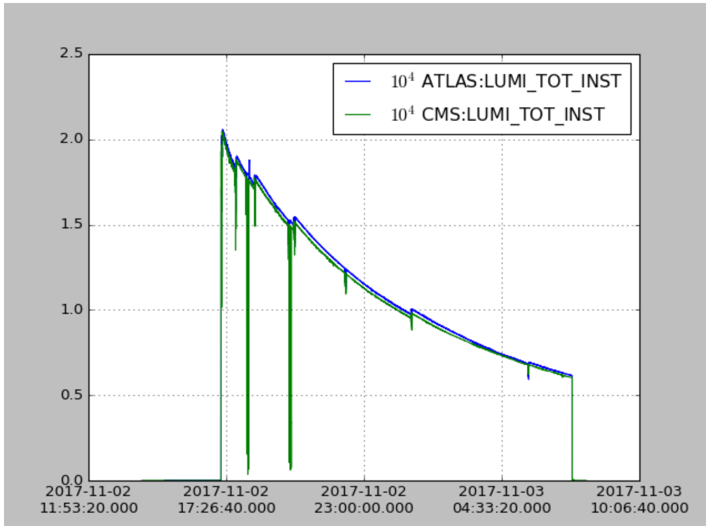
→ Proved to be very handy as many of us are already using python for data analysis, simulations etc.

It is a python **wrapper of the CALS Java API** (made using Jpype)

- Hides most of the "java technicalities" providing a **user-friendly** but **scriptable** interface
- Started like a "personal" project, it spread very fast (>100 users across CERN) and further evolved by **community development** (based on GitHub)
- Made available and regularly used within the **SWAN environment**, developed by the LHC experiment for **interactive data analysis** using **cloud computing** resources (you can do everything in your web browser)

Example: download and plot LHC luminosities

```
import pytimber, pylab
ldb = pytimber.LoggingDB()

data = pytimber.DataQuery(ldb,
     ["ATLAS:LUMI_TOT_INST","CMS:LUMI_TOT_INST"],
     "2017-11-02 14:00:00", “2017-11-0308:00:00”)

data.plot_2d()

pylab.show()
```



Authors: R. De Maria, T. Levens, C. Hernalsteens, M. Betz, M. Fitterer, R. Castellotti

For more info: *github repository*

Xing angle example: PyTimber implementation

For all fills of the 2017 p-p run:

- Download beam modes info
- Check if SB declared (if not skip fill)
- Get start-end of SB timestamps
- Get crossing angle at start SB
- Get crossing angle during SB
- Identify time of first step (t_{step})
- Save Xing angle before and after t_{step}
- Extract beam intensity at t_{step}
- Extract number of bunches
- Compute average bunch intensity
- Save average bunch intensity at t_{step}

With PyTimber these 11 steps can be implemented in ~30 lines of python ☺

```
for filln in xrange(first_fill, last_fill):

    fill_data = ldb.getLHCFillData(filln)
    bmode_dict = build_dict_bmodes(fill_data)

    if 'STABLE' not in bmode_dict.keys():
        print 'No stable beams'; continue

    t_start_stable = bmode_dict['STABLE']['startTime'][0]
    t_end_stable = bmode_dict['STABLE']['endTime'][0]

    ang_var_start = ldb.get([ang_varname],
            t1 = t_start_stable , t2 = 'last')

    ang_var_duringSB = ldb.get([ang_varname],
            t1 = t_start_stable, t2 = t_end_stable)

    if len(ang_var_duringSB[ang_varname][0])==0:
        print 'No crossing change'; continue

    t_1st_change = ang_var_duringSB[ang_varname][0][0]
    t_1st_change_h_list.append((t_1st_change - t_start_stable)/3600.)

    ang_1st_change_list.append(ang_var_duringSB[ang_varname][1][0])
    ang_start_fill_list.append(ang_var_start[ang_varname][1][0])

    inten_vars_start_SB = ldb.get([intenB1_varname, intenB2_varname,
                               nbunB1_varname, nbunB2_varname],
                                t1 = t_start_stable, t2 = 'last')

    inten_vars_at_change = ldb.get([intenB1_varname, intenB2_varname],
                                 t1 = t_1st_change, t2 = 'last')

    avg_bint_start_SB = inten_vars_start_SB[intenB1_varname][1][0]\
                        /inten_vars_start_SB[nbunB1_varname][1][0]
    avg_bint_at_change = inten_vars_at_change[intenB1_varname][1][0]\
                        /inten_vars_start_SB[nbunB1_varname][1][0]

    avg_bint_start_SB_list.append(avg_bint_start_SB)
    avg_bint_at_change_list.append(avg_bint_at_change)
    nbun_list.append(inten_vars_start_SB[nbunB1_varname][1][0])
    filln_list.append(filln)
```

Xing angle example: PyTimber implementation

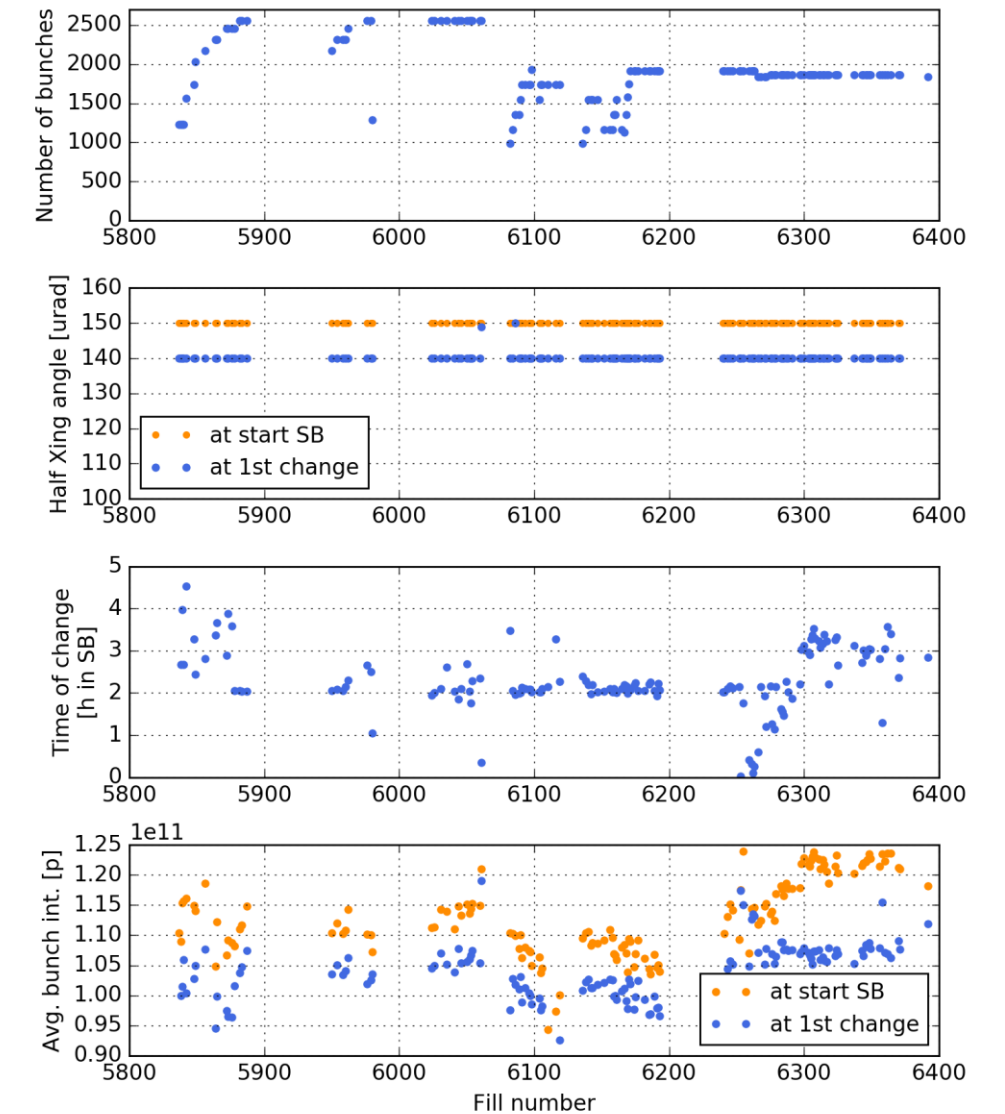
For all fills of the 2017 p-p run:

- Download beam modes info
- Check if SB declared (if not skip fill)
- Get start-end of SB timestamps
- Get crossing angle at start SB
- Get crossing angle during SB
- Identify time of first step (t_{step})
- Save Xing angle before and after t_{step}
- Extract beam intensity at t_{step}
- Extract number of bunches
- Compute average bunch intensity
- Save average bunch intensity at t_{step}

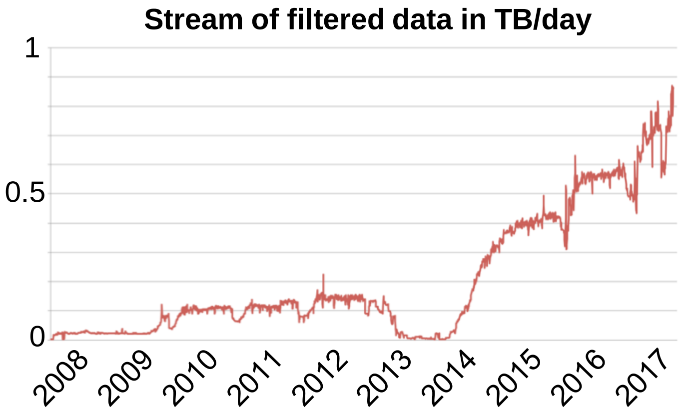
With PyTimber these 11 steps can be implemented in ~30 lines of python ☺

```
for filln in xrange
    fill_data = ldb
    bmode_dict = bu

    if 'S
        p

    t_sta
    t_end

    ang_v

    ang_v

    if le
        p

    t_1st
    t_1st

    ang_1
    ang_s

    inten

    inten

    avg_b

    avg_b

    avg_b
    avg_b
    nbun
    filln
```

Code put together in **~1.5h**
Execution time: **3.5 mins**



The future of the Logging System: NXCALS



BE-CO presently working on **full renovation of the Logging System** to fulfill growing needs:

→ New **NXCALS** system is under development

→ Based on **Hadoop**/**Spark** technology (open source, leading players in the "Big Data" world)

→ **Migration of data** from the present system and **analysis** of all present **use-cases** are **ongoing**

→ Message from the developers: "Give us more people, it will be faster!" :-)

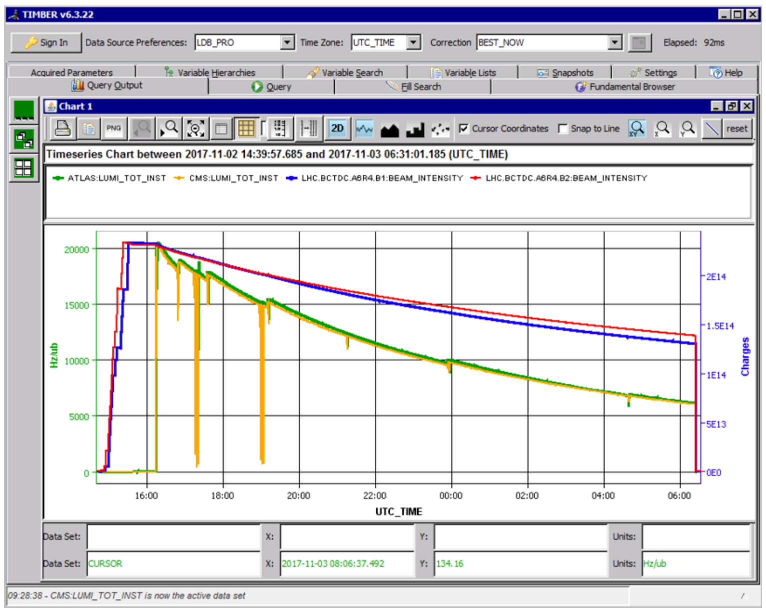
Features of the new system:

- Better **horizontal scalability** (good performance in spite of growing size of stored data, >1 PB)
- Possibility of using **"Big Data" toolset**. Change of paradigm:
  - User does not download data to his local machine but **sends analysis code** to be executed directly by the distributed storage/computing resources
- **Present Java API will be maintained** (present applications, including PyTimber, will still work)
- Plus other and more advanced ways of interacting with the system (python, jupyter, spark, swan…)

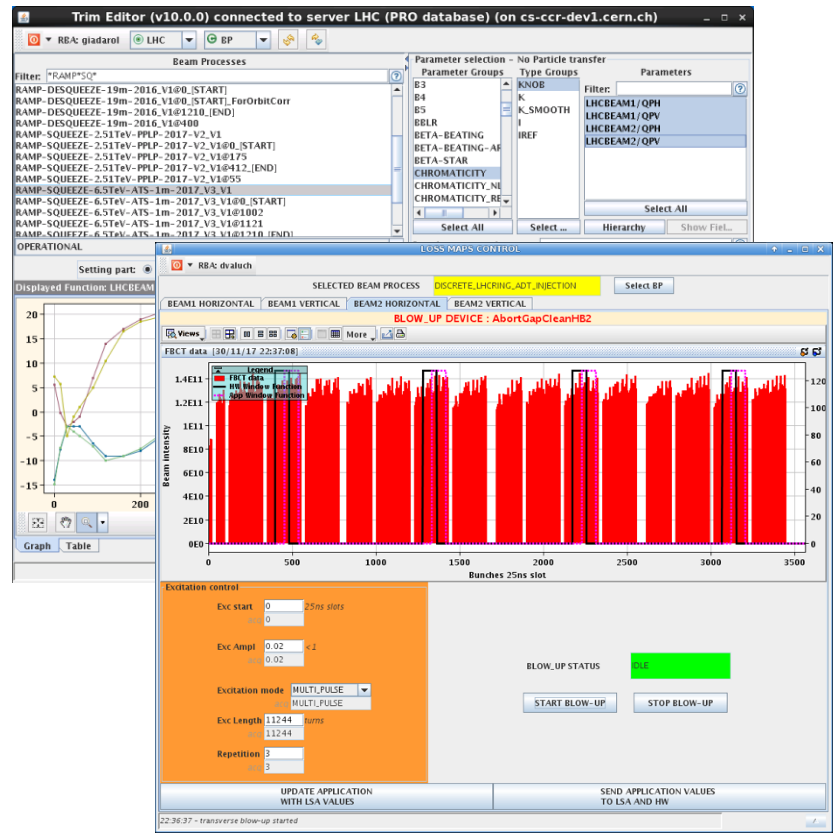
For more info: *gitlab repository*, wiki

Focusing on two types of activity

Extraction and analysis of data from the logging service (CALS)



Interaction with LHC equipment



Interaction with LHC equipment

Machine Studies often require **interacting with LHC equipment**, in particular when using diagnostics from beam instrumentation, RF, etc.

→ Often this is not only a passive observation of published data but requires sending commands, settings, triggers …

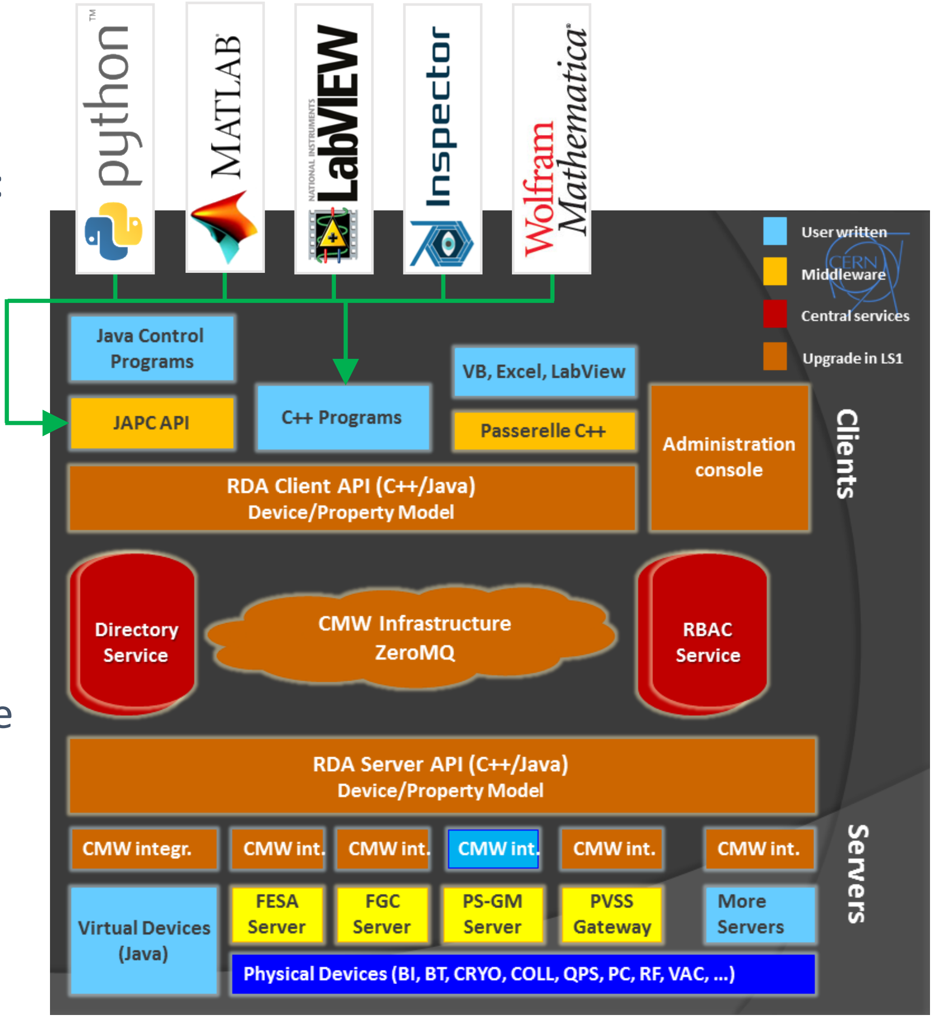
The convectional way is to use the existing **applications** (mostly written in Java), but this does **not cover many cases** of interest for MDs, notably:

- Commissioning of **new devices**
- Experiments with **unconventional usage** of existing devices

Several approaches adopted:

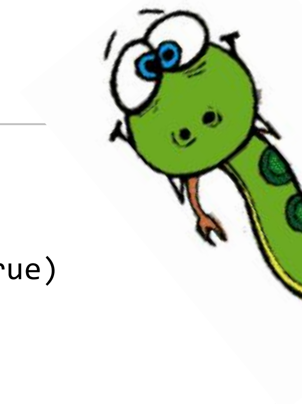
- Develop **ad-hoc java applications** (requires time and expertise)
- Use **scripting language:**

  → **Tools** have been developed to hide some of the complexity and allow for more agile development, common choices: **PyJAPC** and **Inspector**



More info on the different options available at: https://wikis.cern.ch/display/ST/Libraries+Available

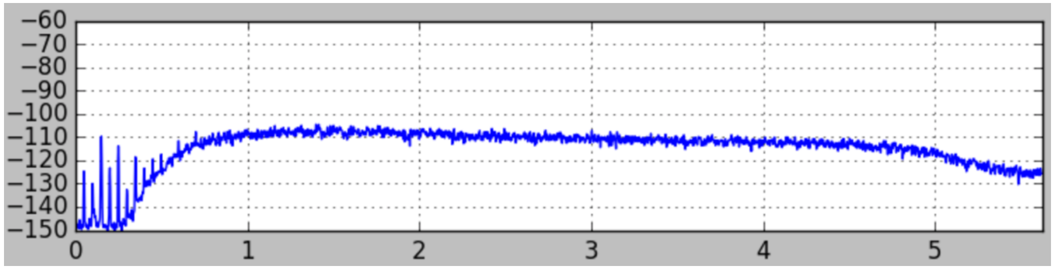
PyJAPC



- **PyJAPC** is a simplified **python interface to accelerator hardware** (e.g. FESA)
- Implementation makes use of JPype to call functions of the "Java API for Parameter Control" (JAPC) directly from Python
- It can be used without knowing anything about the underlying JAPC API
- For more complex functionality it is possible to manually call the relevant JAPC functions from Python

Example: plot BBQ spectrum

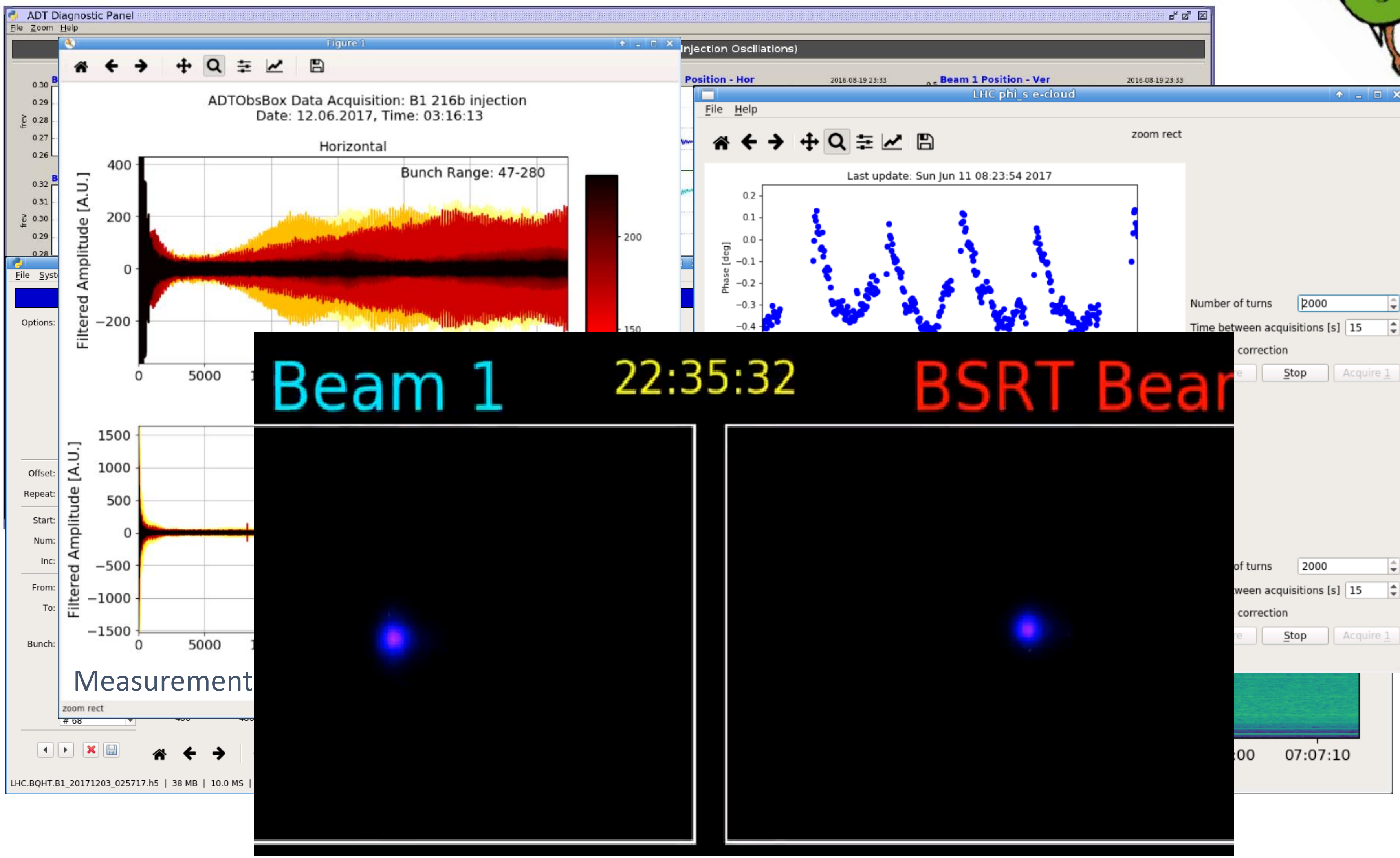
```
# instantiate pyjapc object
import pyjapc
japc = pyjapc.PyJapc(selector="LHC.USER.ALL",
           incaAcceleratorName="LHC", noSet=True)

# RBAC login
japc.rbacLogin(loginDialog=True)

# Get vector data from LHC BBQ
v = japc.getParam(
 "LHC.BQ.ONDEMAND.B1/SummaryMeasurement#averageMagnitudeH")

# plot
import numpy as np; import pylab as pl
xVect = np.linspace(0, 11e3/2, len(v), endpoint=False )
pl.plot(xVect/1e3, v, label=par)

# RBAC Logout
japc.rbacLogout()
```



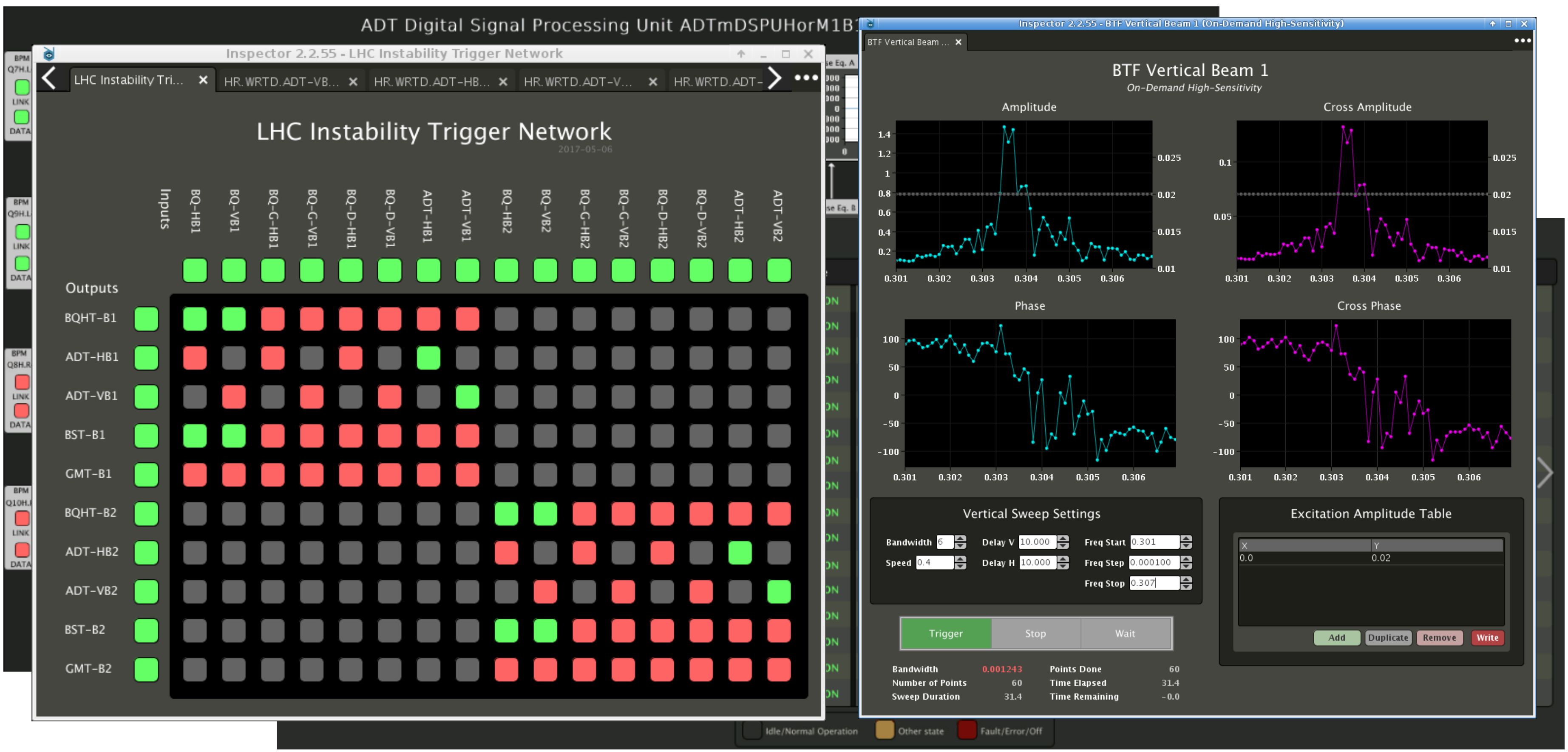
Authors: M. Betz, T. Levens

For more info: documentation, wikis, gitlab repository

- Under the hood of many familiar CCC windows



- Tool for the development of control applications using **graphical programming**
- Based on java API (JAPC)
- Allows fast development of **Graphical User Interfaces (GUIs) and displays**
- Used for several **expert interfaces** and **MD tools**



Authors: B. Lefort et al.

For more info: wiki

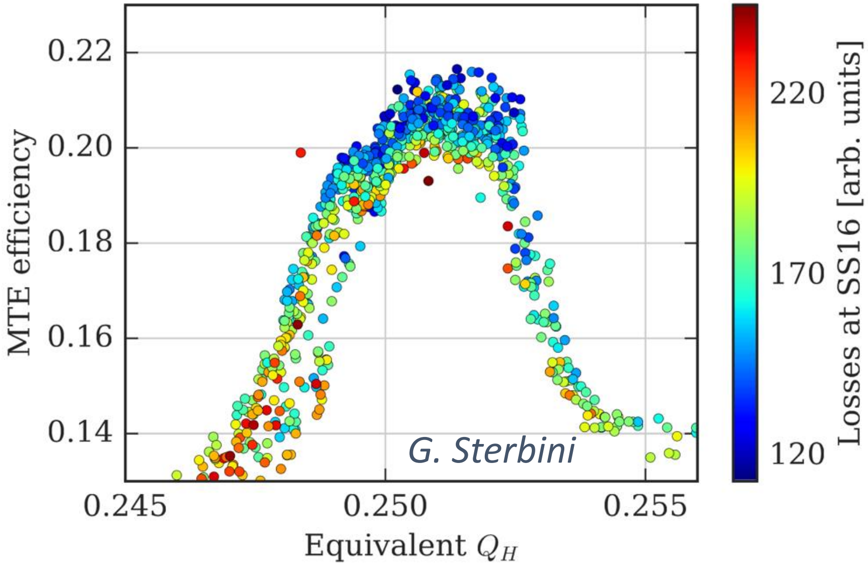
A lot of useful information stored in the **LSA database** (functions, trim history, knob definitions)

- "Conventional" access via **operational applications** (e.g. LHC Trim) or through the **Java API** (only within the Technical Network)
- Recently a **python wrapper** (via Jpype) has been developed → **PjLSA**
  - Only read functions available for now
  - **Accessible** also from the General Purpose Network (**GPN**) within CERN
  - Can be easily **combined with PyTIMBER and PyJAPC** obtaining a **complete scriptable toolbox** for machine studies

In the **injectors** similar tools are used also to automatically send settings and perform **automatic scans**

→ What is the potential for the LHC?

Example: Optimization of transverse excitation for **PS multi-turn extraction**



For more info: github repository

Authors: R. De Maria, M. Hostettler, V. Baggiolini

From MD and expert tools to more general usage

Many of the tools developed for Machine Studies and commissioning of new equipment have **potential to become useful in the LHC daily life**

→ Common perception that that this should be further pursued

See for example D. Jacquet at Evian 2016

From MD and expert tools to more general usage

Many of the tools developed for Machine Studies and commissioning of new equipment have **potential to become useful in the LHC daily life**

→ Common perception that that this should be further pursued

What can we practically do to facilitate this process?

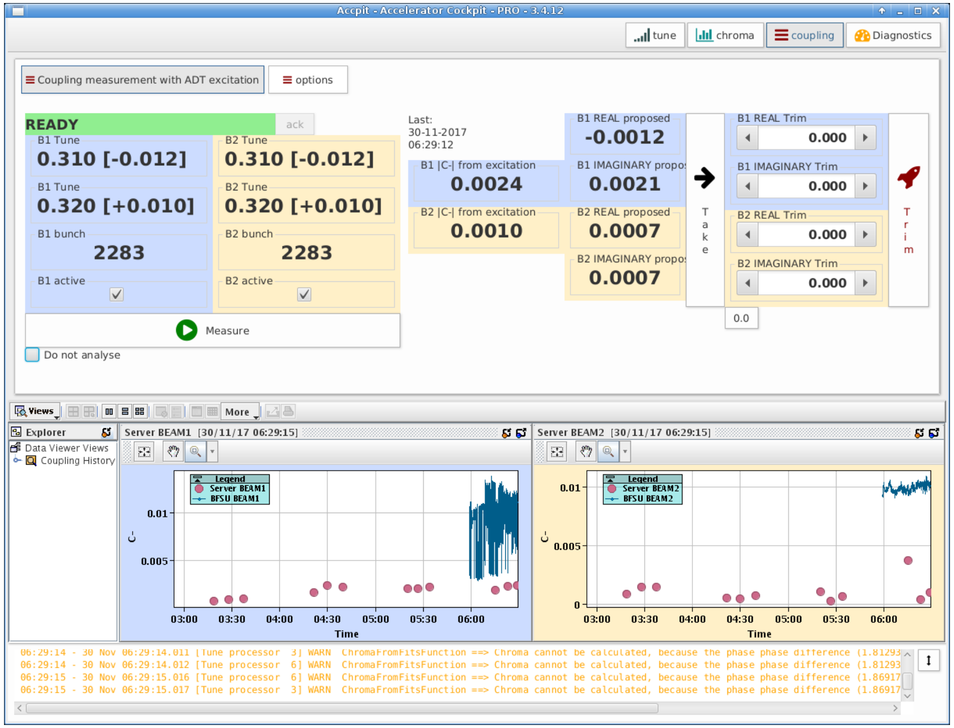
From the BE-CO and BE-OP:

- **Guidelines** and **feedback** on how to develop and maintain (semi-)operational tools within the LHC software ecosystem
- **Accept the python invasion**: this is already happening to some extent (see next slide)
- An agile way to develop **Graphical User Interfaces** (GUIs), e.g. inspector-like

From the MD/equipment groups:

- Code written respecting **basic programming good practices** (a basic training course is often a very good investment ☺)
- Understand and **respect guidelines** defined above
- Minimal commitment for **maintenance** and **support**

Example: **Linear Coupling Measurement**
(ABP/RF development embedded in OP application)



Evolution of python support

Until recently python **not officially supported by BE-CO** (e.g. no official python installation available in Technical Network) ☹

Some **infrastructure built by the users** themselves:

- Different **python installations** available in AFS/NSF user public folders (e.g. BI install.)
- For python tools based on java APIs (pytimber, pyjapc, pjlsa) **CommonBuild Dependency Manager** has been developed (by T. Levens) to automatically identify **java dependencies**, downloads required jars, and setup the required Java Virtual Machine (JVM) within python

Growing **interest and support from BE-CO** in the latest period ☺:

- Python usage (via web-server) **supported in the new CommonBuild** (CBNG) developed by BE-CO-APS
- Contribution in the development of **PjLSA**
- Setup of the **Python Focus Group** (chaired by J. Gonzales Cobas), first objectives:
  - Provide a common forum for python
  - Setup **supported python installation(s)** in the Technical Network (based on LHC Computing Grid) → should be available for the 2018 run
  - Setup basic environment for deployment and maintenance of python applications

Summary

- Several **solutions developed by the MD community** to fully exploit the potential of the LHC hardware and control system in a **flexible** and experimental **way**

- In particular, we start having a **complete and powerful python toolbox** (PyTimber, PyJPAC, PjLSA, PyLogbook) developed **collaboratively** across different teams
  - → Interact with logging, LSA, LHC hardware and perform advanced data analysis all in the same environment
  - → We have only scratched the surface of the potential behind that

- Tools developed for equipment commissioning and Machine Studies **can evolve and prosper the LHC software ecosystem**…

…if we work together to make it happen!

Thanks for your attention!