

# Users entry point

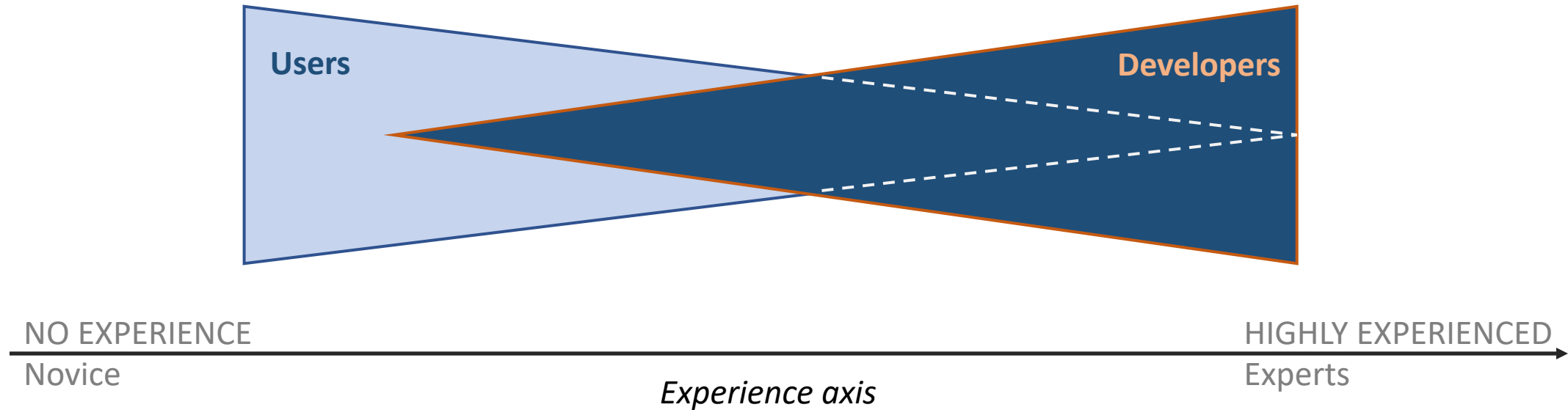


For community reference reconstruction and simulation

# Three talks about our efforts

1. Interfacing with users (general discussion ~~on good and evil~~ on where things are going)
2. **Entry point for users** ← (this talk)  
(with live demo and description)
3. Community reference reconstruction for developers  
(with in-depth details on backends, frontends and their connection)

# Users and developers



- Users solve problems through coding
- Some users are not so experience developers, but still have to code
- Treat users as developers

# Importance of modularity

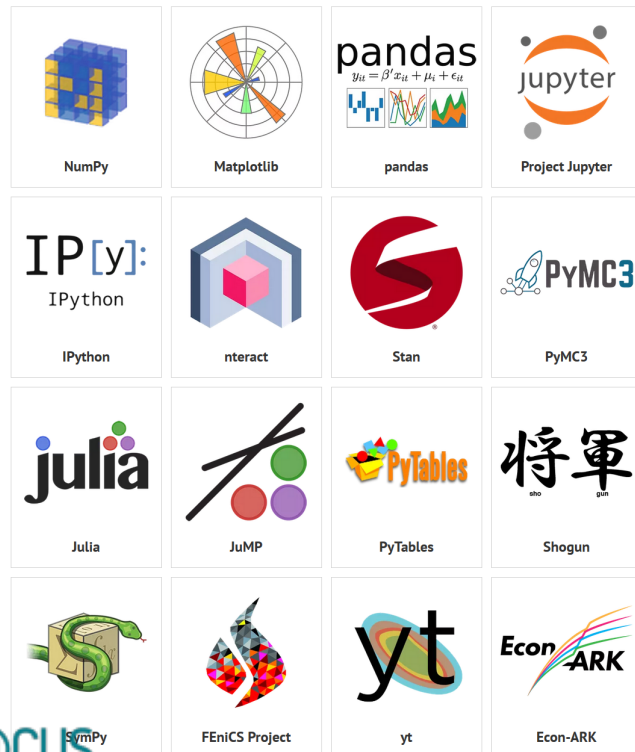


## Software shell be modular!

- It is easier for users to switch parts of software
- It is easier for developers to update parts
- (!) It makes developers to produce better code (!)

**The only way to maintain reusable code for decades**

# Stack of both HEP+NP and DataScience tools



Nowadays software shell work with both words:

- HEP & NP stack like ROOT, Geant4, etc
- Modern DataScience tools like Pandas, Numpy, R,

# Where are frameworks are going?

```

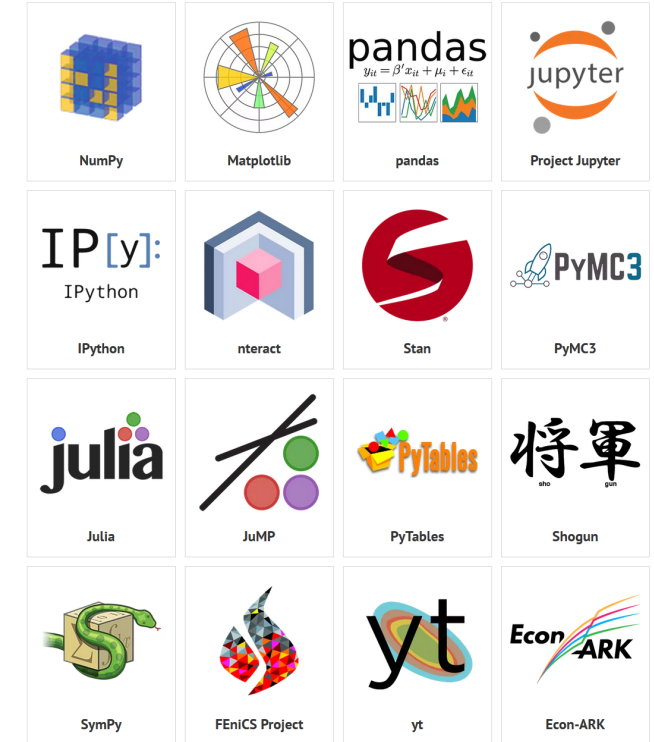
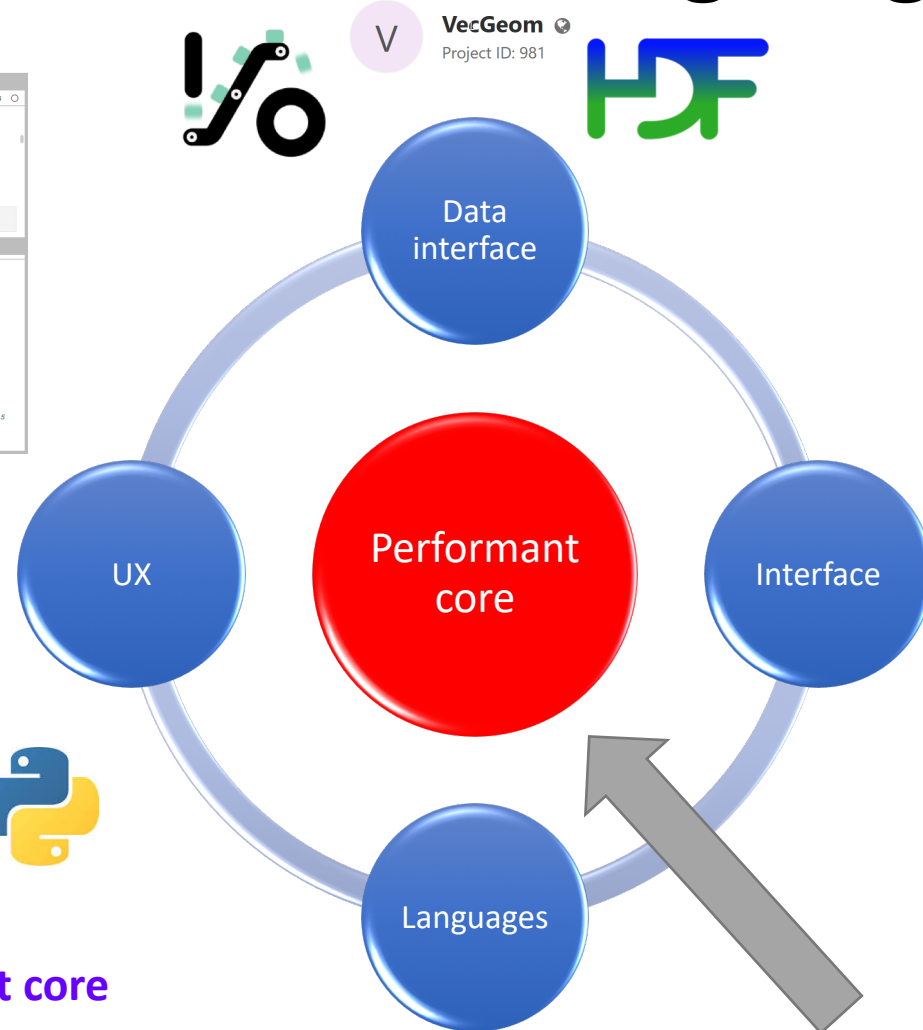
In this Notebook we explore the Lorenz system of differential equations:

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy \end{aligned}$$

Let's call the function once to view the solutions. For this set of parameters, we see the trajectories swirling around two points, called attractors.

In [4]: from lorenz import solve_lorenz
        t, x, z = solve_lorenz(N=10)

Output View
sigma: 10.00
beta: 2.87
rho: 28.00
    
```



Software must have performant core  
 Software must have interfaces

Most probably C++

# e<sup>JANA</sup> - JANA with plugins & deps for EIC community

## e<sup>JANA</sup> - stands for EIC JANA

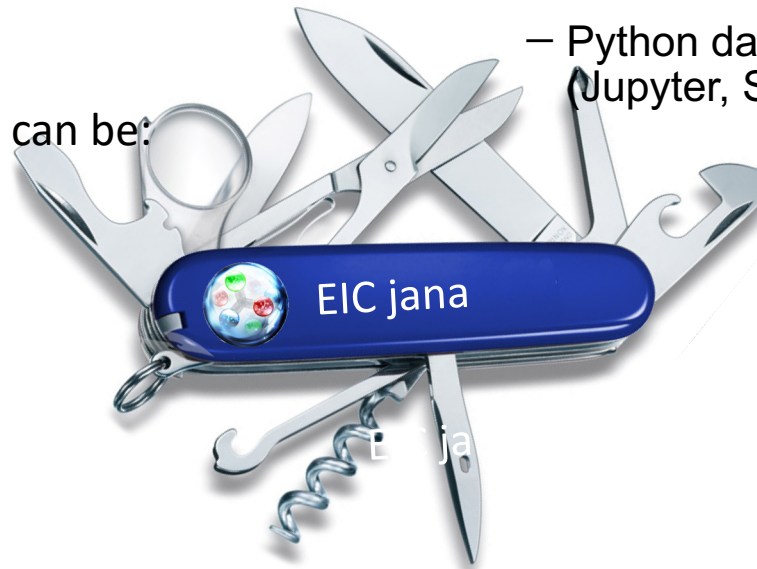
- Basic reconstruction
- Physics analysis
- **Users detector codebase integration**

## Reconstruction

- **Tracking** - Genfit
- **Vertex finding** – Rave
- **Physical analysis:**
  - ROOT C++ or
  - Python data science tools (Jupyter, Seaborn, Pandas, etc.)

Any existing C++ (or even others) code can be:

- compiled as JANA plugin
- run parallelized in eJANA
- accessed by other plugins



# Complexity scaling explained

Some complex system



We try to make complex - simple

Failed complexity scaling:

- **Complex** → **Simple**
- **Simple** → **Complex**

Will fail most of the times  
because of  
**complexity scaling**  
problem





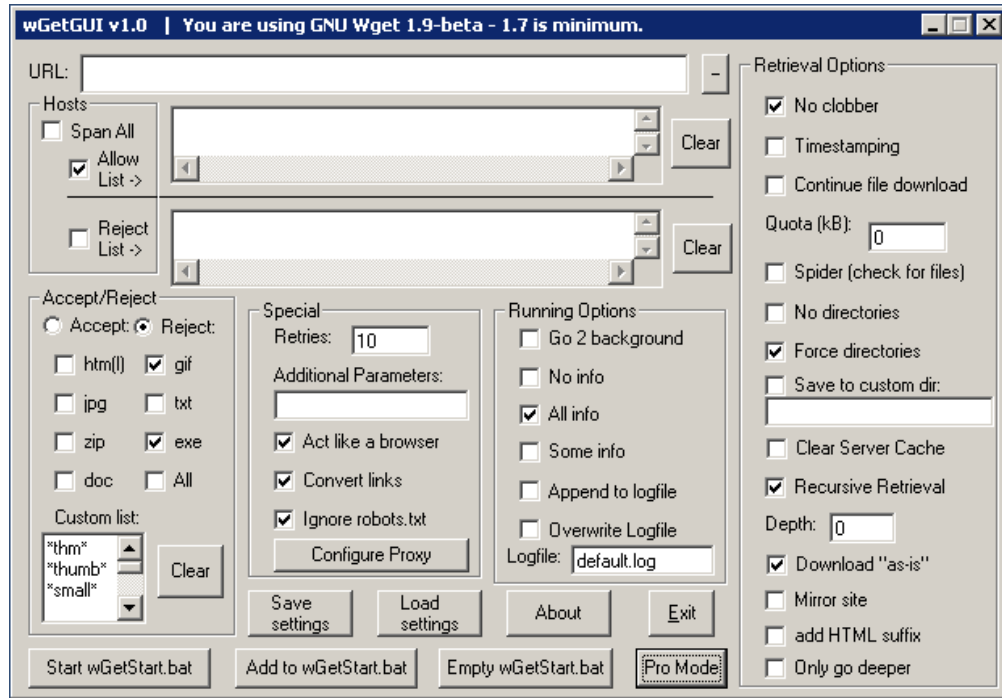
# Escaping complexity scaling trap

- Provide interfaces to internal complexity  
*(and better for everything)*  
*(and even better – each step must be replaceable)*
- Interaction between layers must be clear



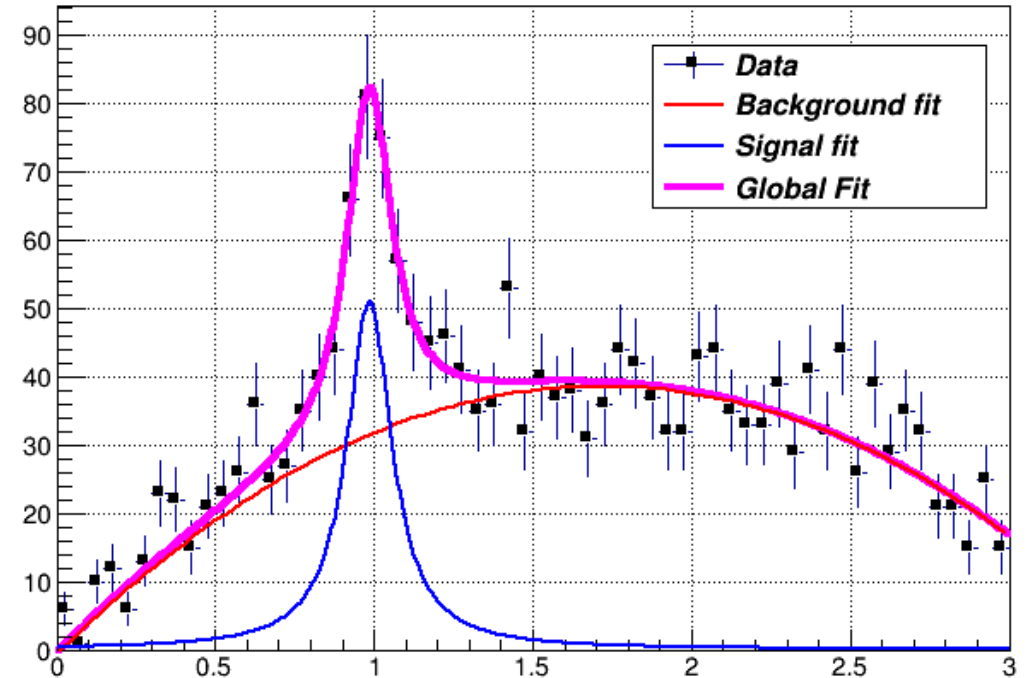
*(And working on modularity... true modularity is the way to make it)*

# GUI you hate



# GUI you love

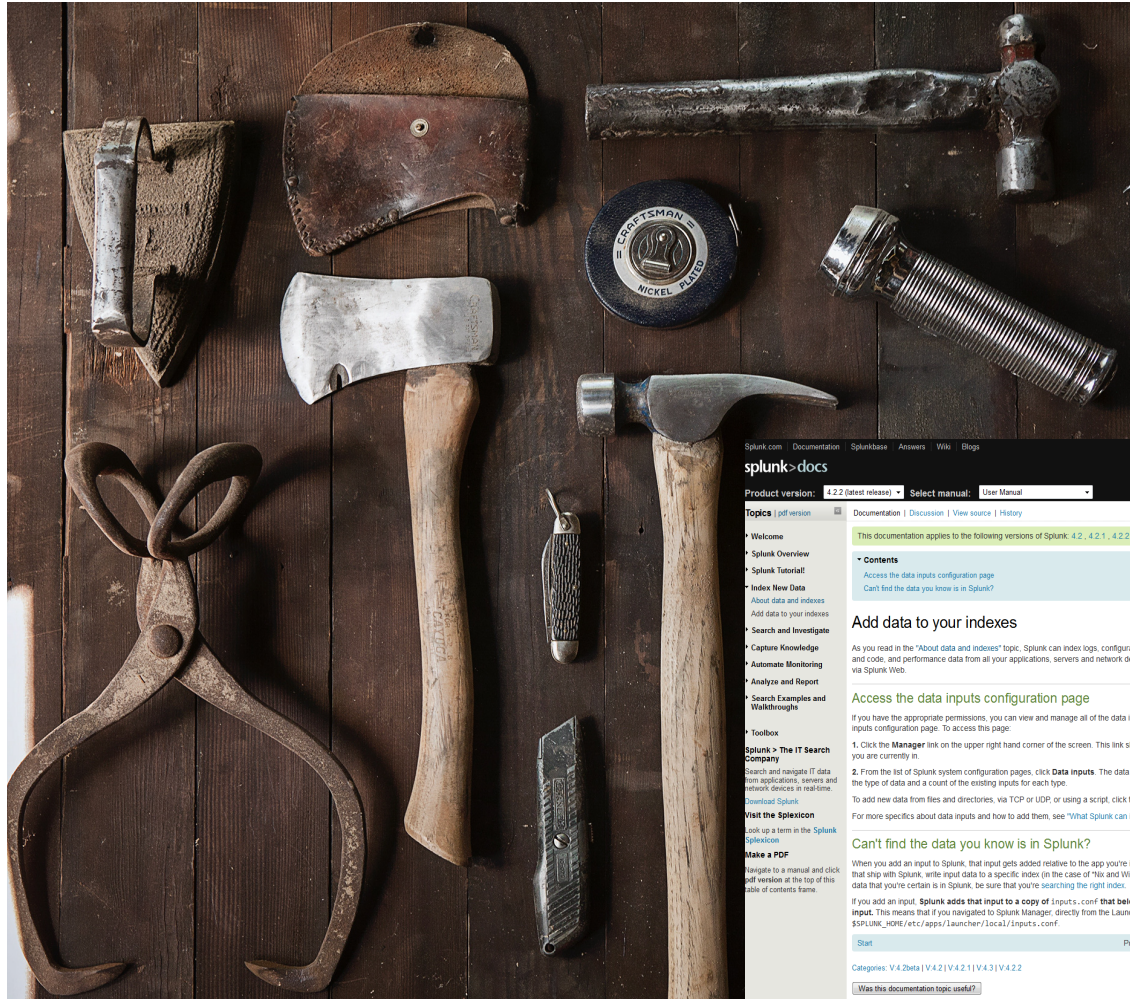
Lorentzian Peak on Quadratic Background



Wrapping something into a GUI to something is controversial:

- GUIs are pretty easily fall into Complexity scaling trap
- For some tasks GUI is not only the best solution, but the only acceptable solution (Plots are GUI too!)

# We provide just tools

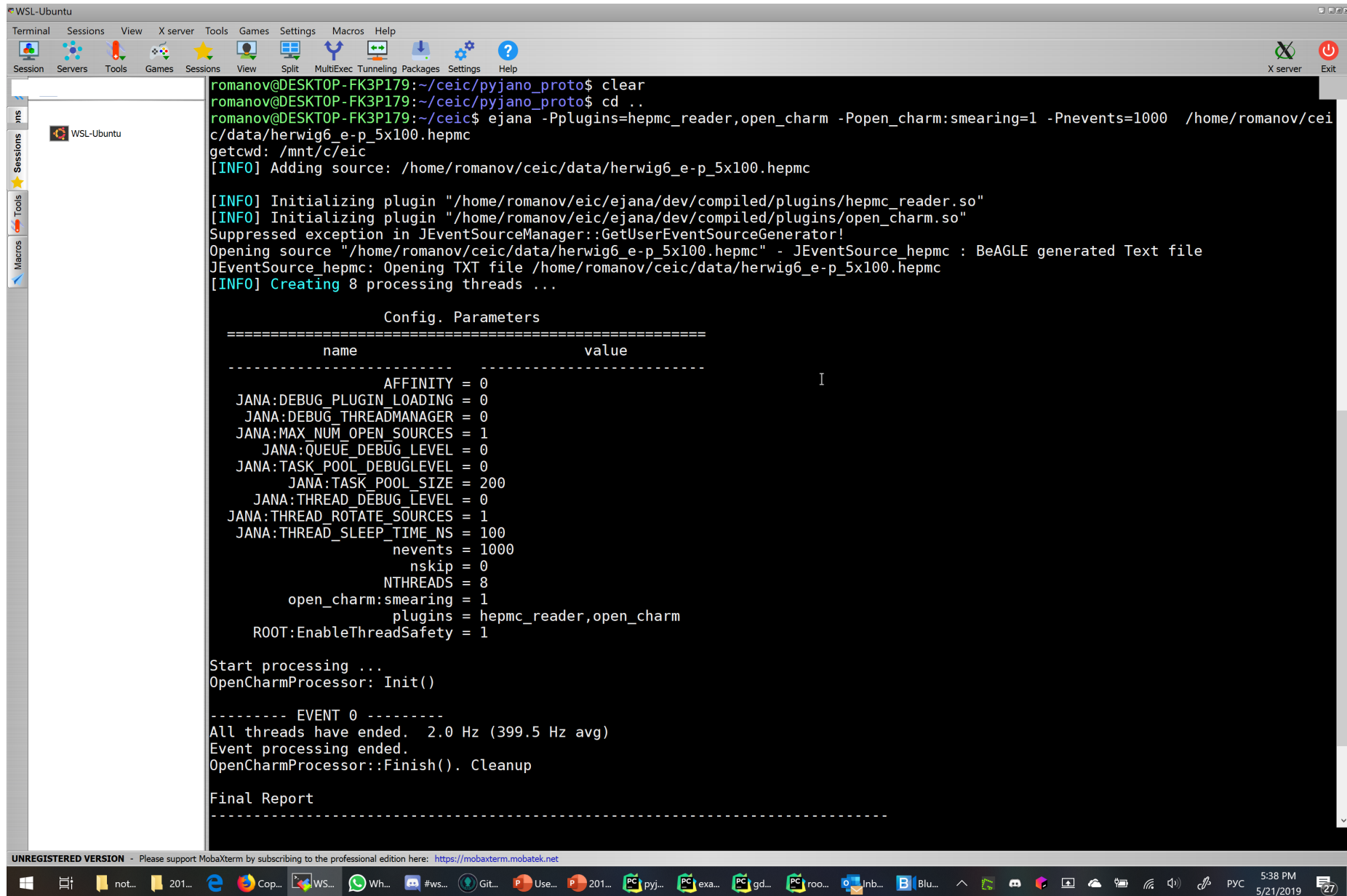


# Users also need workflows



- Providing just tools and documentation in form of wikis/sites to it make learning curve steep and prevents users from doing their job effectively
- Workflows of how users achieve their tasks must be overthought on each step of software development

# Example of how docker container provides tools but not workflow



```
romanov@DESKTOP-FK3P179:~/ceic/pyjano_proto$ clear
romanov@DESKTOP-FK3P179:~/ceic/pyjano_proto$ cd ..
romanov@DESKTOP-FK3P179:~/ceic$ ejana -Pplugins=hepmc_reader,open_charm -Popen_charm:smearing=1 -Pnevents=1000 /home/romanov/cei
c/data/herwig6_e-p_5x100.hepmc
getcwd: /mnt/c/eic
[INFO] Adding source: /home/romanov/ceic/data/herwig6_e-p_5x100.hepmc

[INFO] Initializing plugin "/home/romanov/eic/ejana/dev/compiled/plugins/hepmc_reader.so"
[INFO] Initializing plugin "/home/romanov/eic/ejana/dev/compiled/plugins/open_charm.so"
Suppressed exception in JEventManager::GetUserEventSourceGenerator!
Opening source "/home/romanov/ceic/data/herwig6_e-p_5x100.hepmc" - JEventSource_hepmc : BeAGLE generated Text file
JEventSource_hepmc: Opening TXT file /home/romanov/ceic/data/herwig6_e-p_5x100.hepmc
[INFO] Creating 8 processing threads ...

Config. Parameters
=====
name                               value
-----
AFFINITY = 0
JANA:DEBUG_PLUGIN_LOADING = 0
JANA:DEBUG_THREADMANAGER = 0
JANA:MAX_NUM_OPEN_SOURCES = 1
JANA:QUEUE_DEBUG_LEVEL = 0
JANA:TASK_POOL_DEBUGLEVEL = 0
JANA:TASK_POOL_SIZE = 200
JANA:THREAD_DEBUG_LEVEL = 0
JANA:THREAD_ROTATE_SOURCES = 1
JANA:THREAD_SLEEP_TIME_NS = 100
nevents = 1000
nskip = 0
NTHREADS = 8
open_charm:smearing = 1
plugins = hepmc_reader,open_charm
ROOT:EnableThreadSafety = 1

Start processing ...
OpenCharmProcessor: Init()

----- EVENT 0 -----
All threads have ended.  2.0 Hz (399.5 Hz avg)
Event processing ended.
OpenCharmProcessor::Finish(). Cleanup

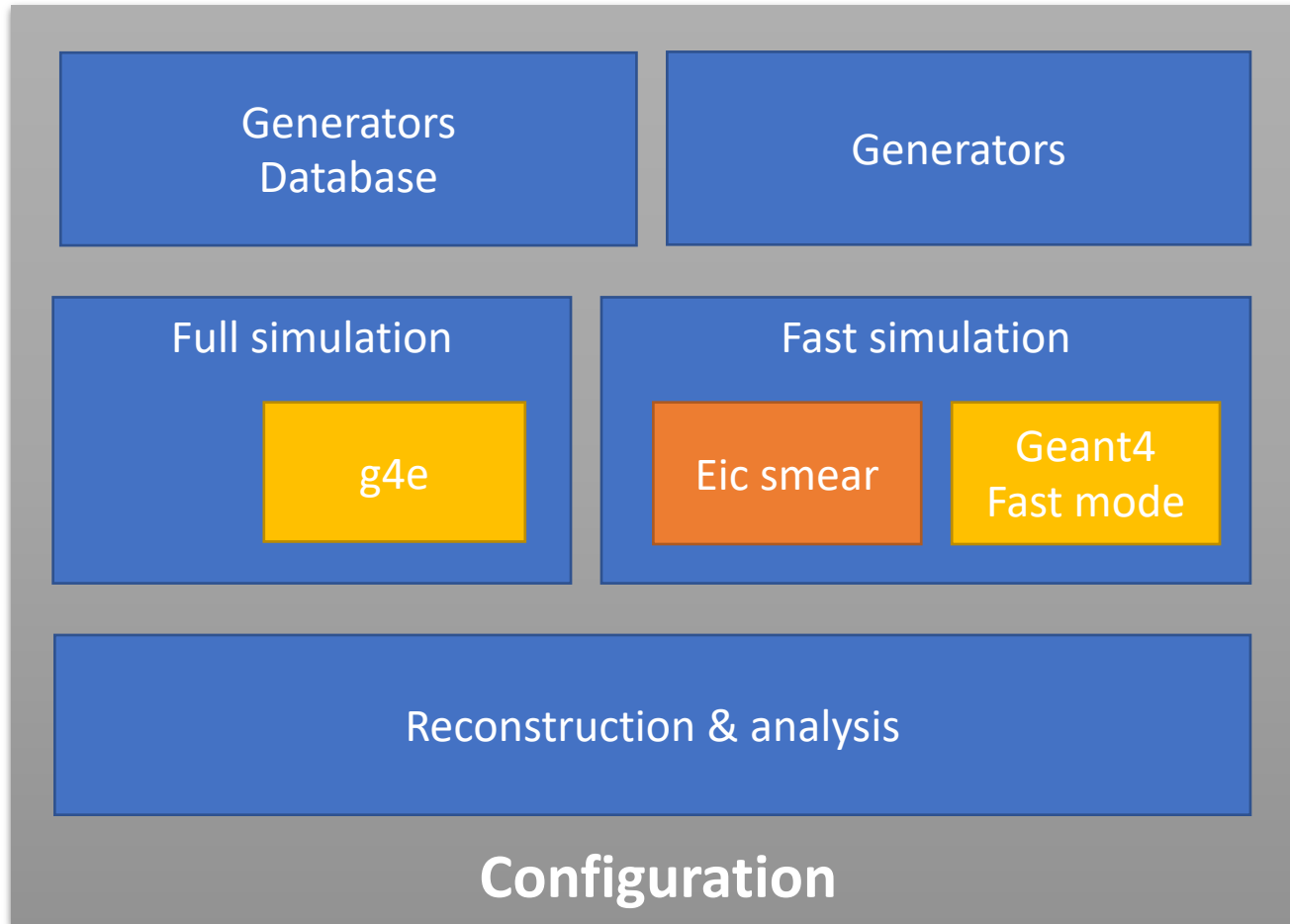
Final Report
-----
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>

# Target audience

- Users from specific collaborations
- Detector design groups
- General audience  
(users, who wants to do physics studies)

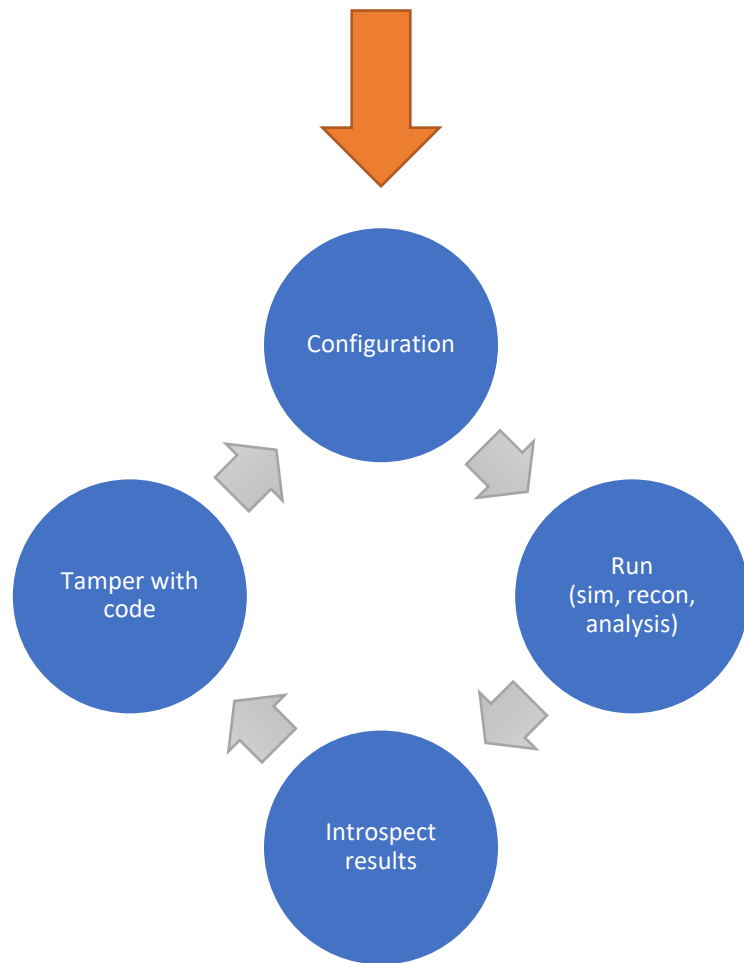
# MC Chain



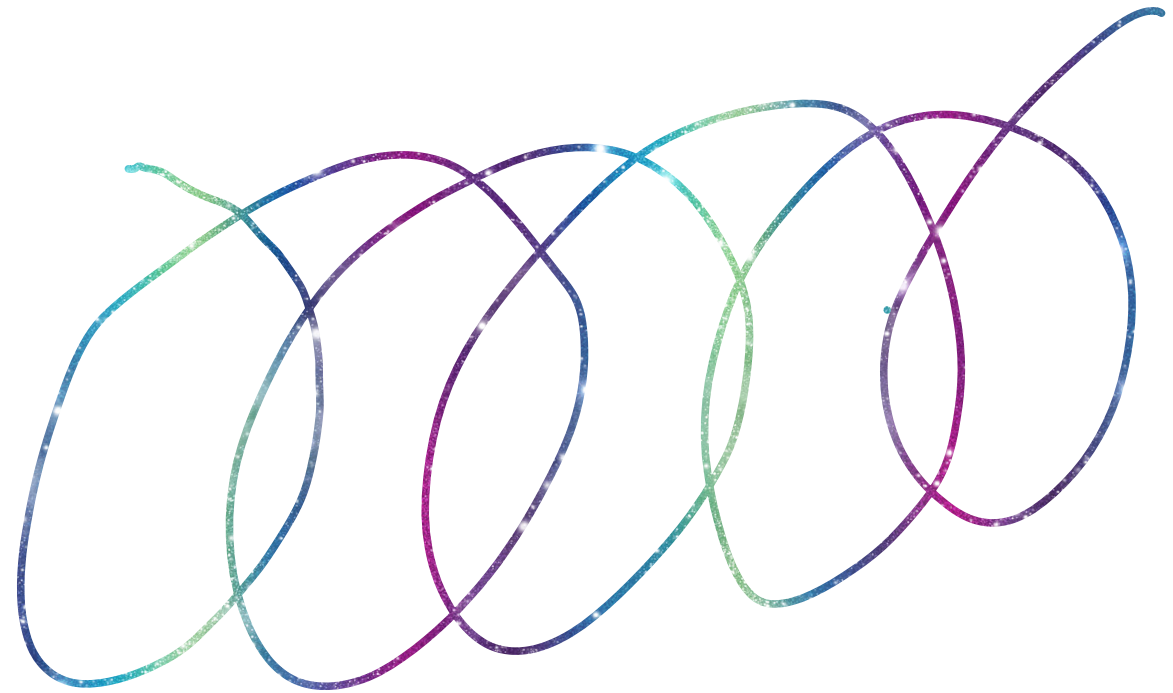
## How easy to configure Generator + Detector simulation + Reconstruction?

- Each module require its own configuration
- Configuration of the whole system is cumbersome
- We present new package that thinks of a workflow of how you configure and run such stack

# Can we identify entry point workflows



How it really works



\* (The Spiral depicts that in reality user experience changes on each cycle. Users knowledge of the system grows and their understanding of what is required to make the job done develops)

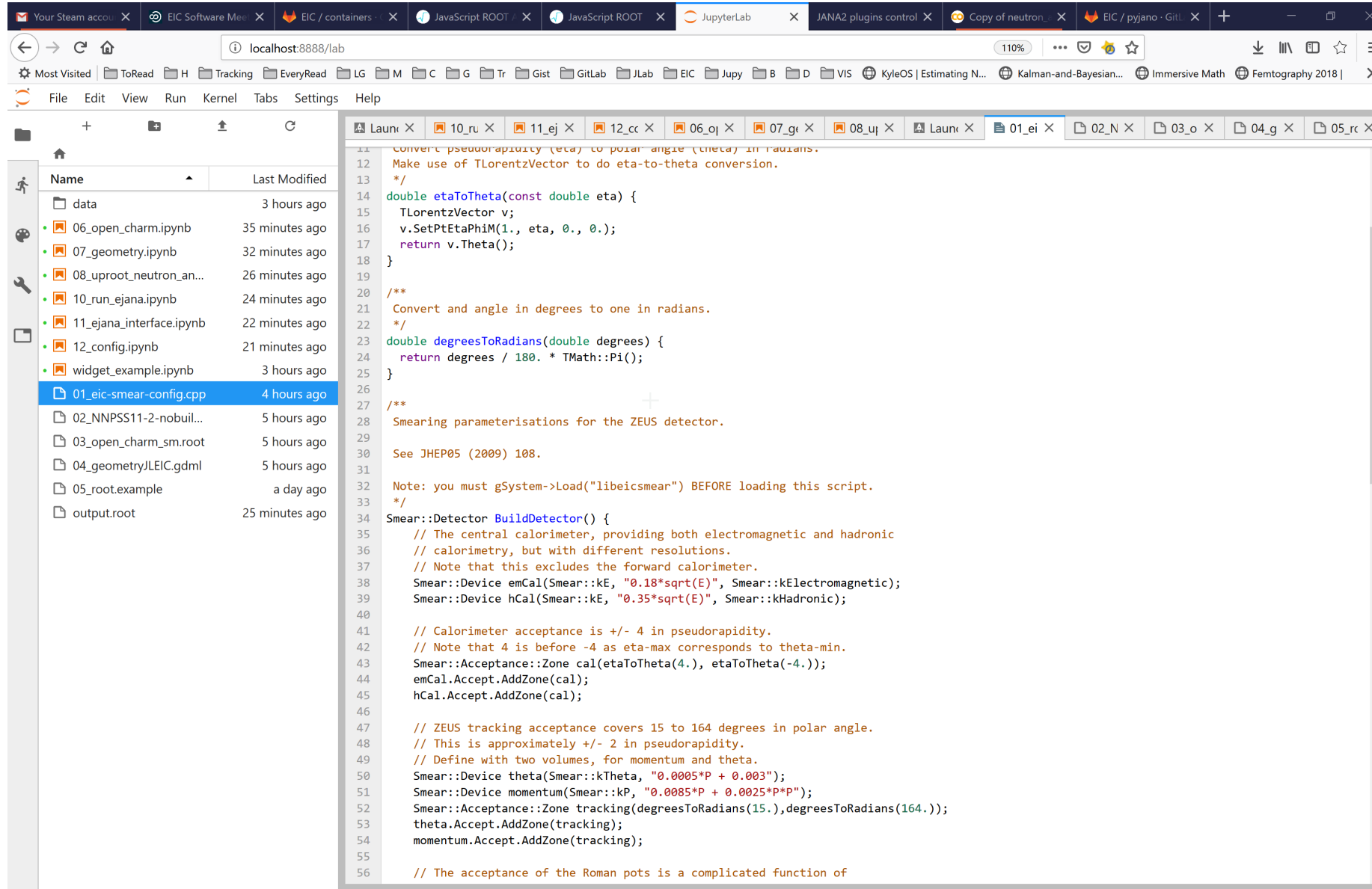
*The next part of this presentation shows how such workflow can be implemented convenient for users*

# JupyterLab for EIC

Presentation of developed user environment  
based on jupyterlab.



# Env. allows to edit code of many languages with syntax highlighting and even with autocompletion



The screenshot shows a JupyterLab interface with a file explorer on the left and a code editor on the right. The file explorer lists files such as `01_eic-smear-config.cpp`, `02_NNPS11-2-nobuil...`, `03_open_charm_sm.root`, `04_geometryJLEIC.gdml`, `05_root.example`, and `output.root`. The code editor displays C++ code for a detector configuration, including comments and function definitions like `etaToTheta` and `degreesToRadians`. The code is syntax-highlighted and includes autocompletion suggestions.

```
11 Convert pseudorapidity (eta) to polar angle (theta) in radians.
12 Make use of TLorentzVector to do eta-to-theta conversion.
13 */
14 double etaToTheta(const double eta) {
15     TLorentzVector v;
16     v.SetPtEtaPhiM(1., eta, 0., 0.);
17     return v.Theta();
18 }
19
20 /**
21 Convert and angle in degrees to one in radians.
22 */
23 double degreesToRadians(double degrees) {
24     return degrees / 180. * TMath::Pi();
25 }
26
27 /**
28 Smearing parameterisations for the ZEUS detector.
29
30 See JHEP05 (2009) 108.
31
32 Note: you must gSystem->Load("libeicsmear") BEFORE loading this script.
33 */
34 Smear::Detector BuildDetector() {
35     // The central calorimeter, providing both electromagnetic and hadronic
36     // calorimetry, but with different resolutions.
37     // Note that this excludes the forward calorimeter.
38     Smear::Device emCal(Smear::kE, "0.18*sqrt(E)", Smear::kElectromagnetic);
39     Smear::Device hCal(Smear::kE, "0.35*sqrt(E)", Smear::kHadronic);
40
41     // Calorimeter acceptance is +/- 4 in pseudorapidity.
42     // Note that 4 is before -4 as eta-max corresponds to theta-min.
43     Smear::Acceptance::Zone cal(etaToTheta(4.), etaToTheta(-4.));
44     emCal.Accept.AddZone(cal);
45     hCal.Accept.AddZone(cal);
46
47     // ZEUS tracking acceptance covers 15 to 164 degrees in polar angle.
48     // This is approximately +/- 2 in pseudorapidity.
49     // Define with two volumes, for momentum and theta.
50     Smear::Device theta(Smear::kTheta, "0.0005*P + 0.003");
51     Smear::Device momentum(Smear::kP, "0.0085*P + 0.0025*P*P");
52     Smear::Acceptance::Zone tracking(degreesToRadians(15.), degreesToRadians(164.));
53     theta.Accept.AddZone(tracking);
54     momentum.Accept.AddZone(tracking);
55
56     // The acceptance of the Roman pots is a complicated function of
```

PDF files are rendered. Latex, markdown. Good opportunity for docs

The screenshot displays a JupyterLab environment with a file browser on the left and a rendered PDF slide in the main area. The browser tabs include 'Your Steam account', 'EIC Software Meeting', 'EIC / containers', 'JavaScript ROOT', 'JupyterLab', 'JANA2 plugins control', 'Copy of neutron...', 'EIC / pyjano - GitLab', and others. The file browser shows a list of files with their last modified times, with '02\_NNPSS11-2-nobuil...' selected. The rendered PDF slide is titled 'Deep-inelastic scattering' and features a diagram of an electron ( $e$ ) scattering off a proton via a virtual photon ( $\gamma^*$ ). The diagram includes a box labeled 'The virtual photon and  $Q^2$ ' and a logo for 'hermes'. Text on the slide explains that in relativistic quantum mechanics (quantum field theory), scattering is treated as if a virtual particle were exchanged between the beam and target. At the bottom, two boxes labeled 'force' and 'carrier' are shown. The system tray at the bottom indicates the time is 5:24 PM on 5/21/2019.

One can introspect root files. Everything is interactive and “rootish”

The screenshot displays a JupyterLab environment with a browser window at localhost:8888/lab. The interface is divided into several sections:

- File Browser (Left):** Shows a directory structure with files like '03\_open\_charm\_sm.root' selected, which was modified 5 hours ago.
- ROOT online server (Middle-Left):** Displays the JSROOT version (6.16.00) and a file hierarchy tree. The selected file '03\_open\_charm\_sm.root' is expanded to show its contents, including 'opencharm' and various histograms like 'hD0\_KaonPvsEta;1'.
- Plot (Middle-Right):** A 2D histogram titled 'Kaons from D0 ptot vs eta'. The x-axis is labeled  $\eta$  (ranging from -4 to 4) and the y-axis is labeled  $P_{tot} [GeV]$  (ranging from 0 to 20). A tooltip for a specific bin shows:   
open\_charm\_sm.root/opencharm/hD0\_KaonPvsEta;1  
x = [3.100, 3.200]  
y = [11, 11.20]  
bin = 81, 55  
entries = 1
- Statistics Panel (Top-Right):** A table for 'hD0\_KaonPvsEta' with the following values:  
Entries: 27070  
Mean x: 1.421  
Mean y: 2.534  
Std Dev x: 1.222  
Std Dev y: 2.496

# Render GDML geometry

The screenshot shows a JupyterLab environment with the following components:

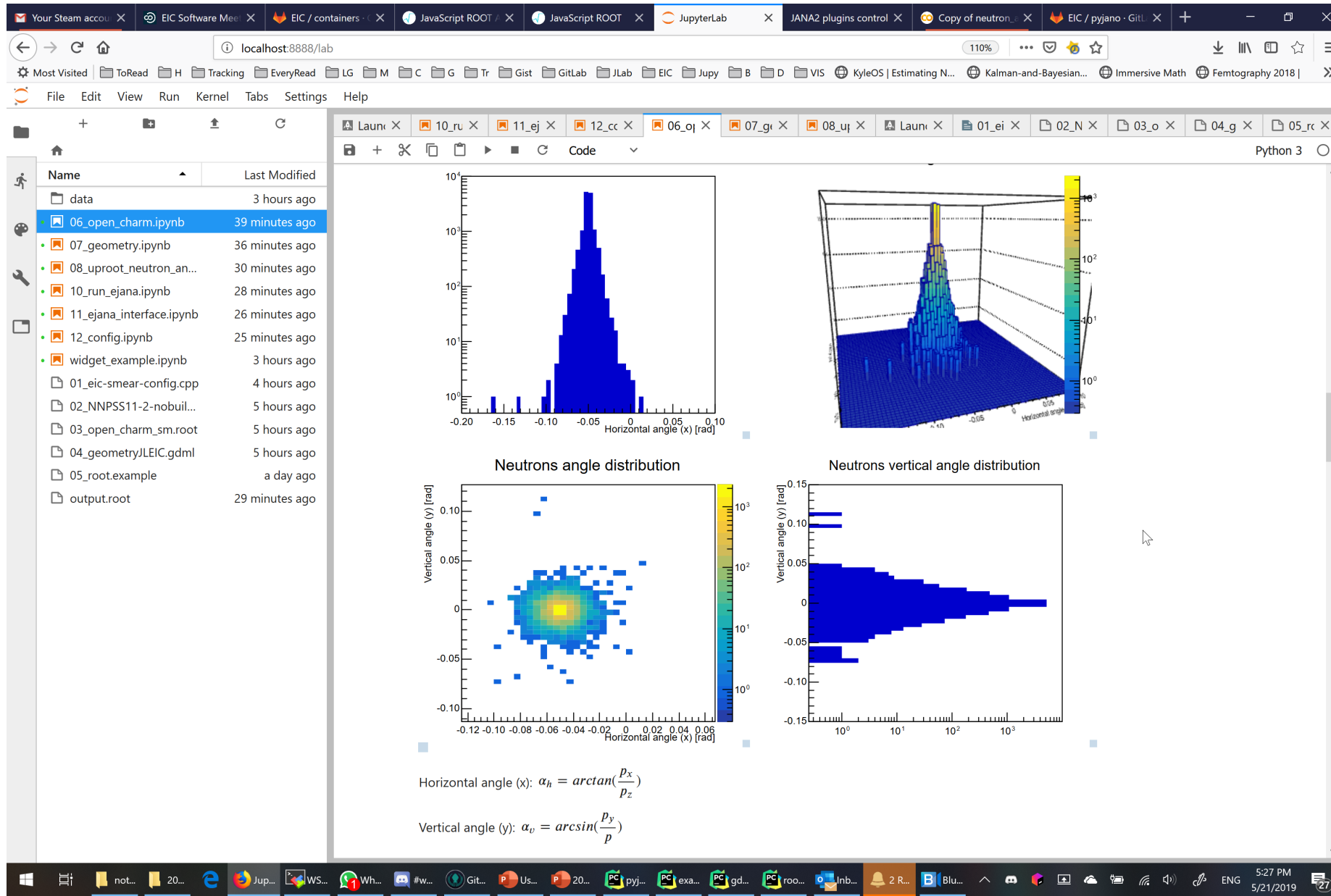
- Browser Tabs:** Includes "Your Steam account", "EIC Software Meet", "EIC / containers", "JavaScript ROOT", "JavaScript ROOT", "JupyterLab", "JANA2 plugins control", "Copy of neutron", and "EIC / pyjano - Git".
- Address Bar:** Shows the URL "localhost:8888/lab".
- File Browser (Left):** Lists files and folders with their last modified times. The file "04\_geometryJLEIC.gdml" is selected.
- Terminal (Top):** Displays the "ROOT online server" interface, including the JSROOT version (6.16.00), hierarchy in json and xml format, and a file tree showing "ROOT" > "geometryJLEIC.root" > "VGM Root geometry;1".
- Control Panel (Right):** Contains options for "Clipping", "Appearance", and "Advanced", with a "Close Controls" button.
- 3D View (Center):** Shows a 3D rendering of a particle detector component, likely a beam pipe or detector element, rendered in a light gray color.

# (While not yet fully implemented for EIC), example of event viewer browser from CERN

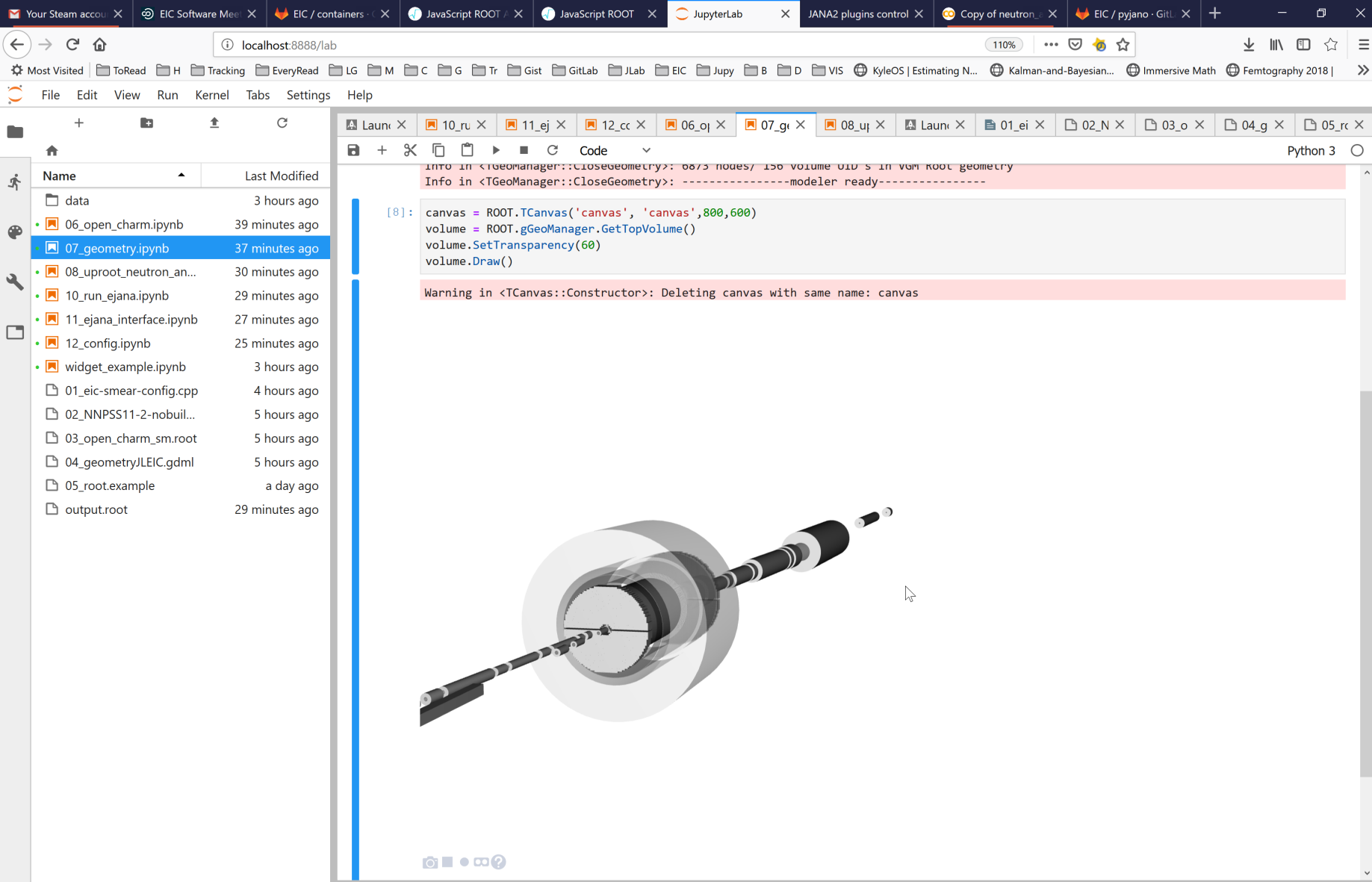
The screenshot displays a JupyterLab environment running in a browser at localhost:8888/lab. The interface includes a file browser on the left, a central workspace with a 3D visualization, and a top toolbar with various application tabs. The 3D visualization shows a complex detector geometry with a central orange cylindrical component. Numerous colored lines (purple, green, blue, red) represent particle tracks originating from the center and extending outwards. The tracks are overlaid on a semi-transparent blue and purple detector structure. The file browser on the left shows a list of files and folders, with '05\_root.example' selected. The top toolbar shows several open tabs, including 'JupyterLab', 'JANA2 plugins control', and 'Copy of neutron...'. The bottom status bar shows the system time as 5:26 PM on 5/21/2019.

Name	Last Modified
data	3 hours ago
06_open_charm.ipynb	37 minutes ago
07_geometry.ipynb	35 minutes ago
08_uproot_neutron_an...	29 minutes ago
10_run_ejana.ipynb	27 minutes ago
11_ejana_interface.ipynb	25 minutes ago
12_config.ipynb	23 minutes ago
widget_example.ipynb	3 hours ago
01_eic-smear-config.cpp	4 hours ago
02_NNPSS11-2-nobuil...	5 hours ago
03_open_charm_sm.root	5 hours ago
04_geometryLEIC.gdml	5 hours ago
05_root.example	a day ago
output.root	27 minutes ago

# Interactive root plots in jupyter notebooks



# Geometry can be opened right in jupyter notebook



The screenshot displays a JupyterLab environment with a file browser on the left and a code editor on the right. The file browser shows a list of files and folders, with `07_geometry.ipynb` selected. The code editor shows the following Python code:

```
Info in <TGeoManager::CloseGeometry>: 6875 nodes/ 136 volume UID's in vsm root geometry
Info in <TGeoManager::CloseGeometry>: -----modeler ready-----

[8]: canvas = ROOT.TCanvas('canvas', 'canvas', 800, 600)
      volume = ROOT.gGeoManager.GetTopVolume()
      volume.SetTransparency(60)
      volume.Draw()

Warning in <TCanvas::Constructor>: Deleting canvas with same name: canvas
```

Below the code, a 3D visualization of a complex geometry is shown. The geometry consists of a central cylindrical component with a smaller cylindrical section extending from its top, surrounded by a larger, semi-transparent cylindrical shell. The visualization is rendered in a perspective view.

# Why 76 degrees?

- At apple stores laptop screens must all be set at exactly 76 degrees.

Why?

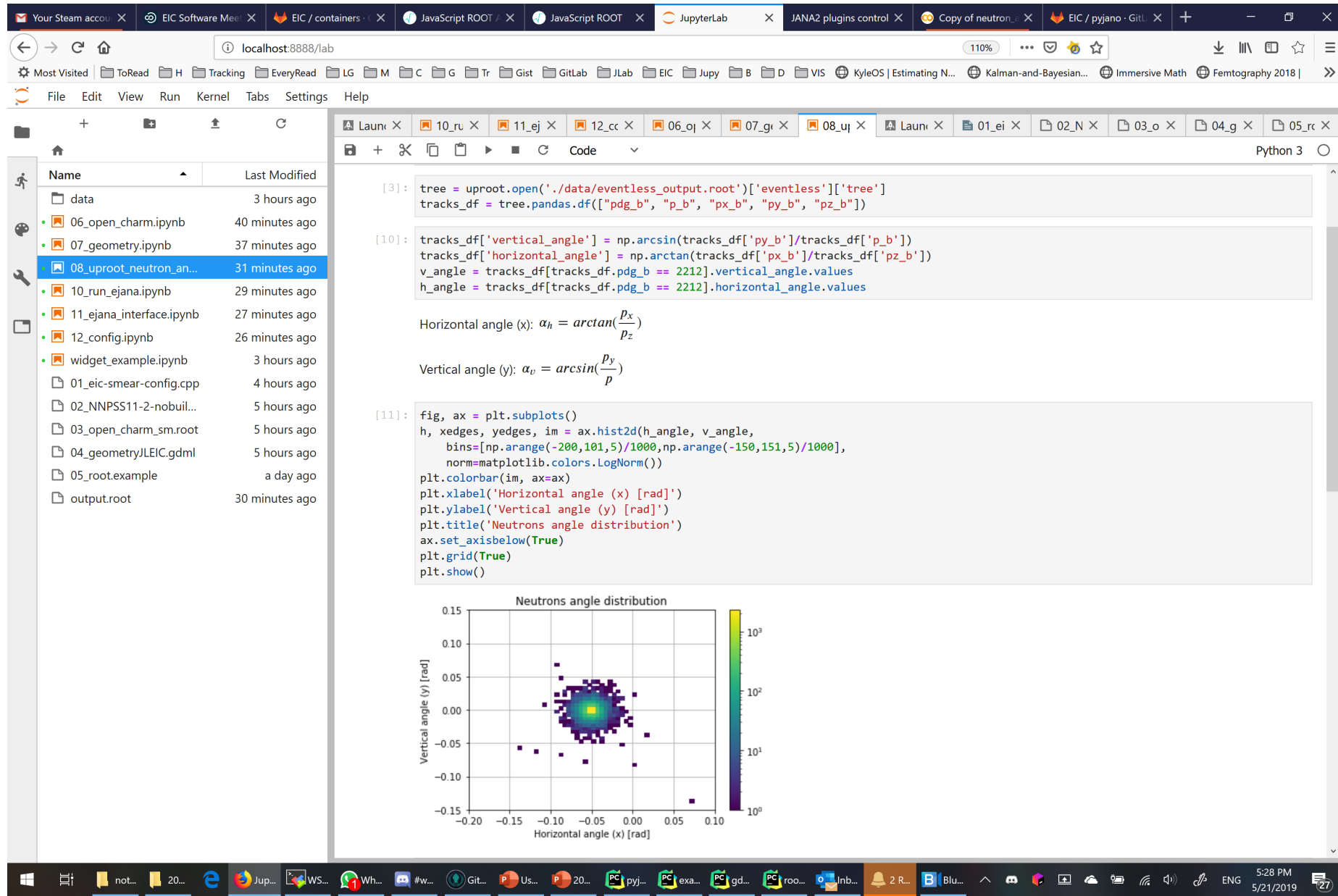


*It is the most uncomfortable angle to look at a display which makes store visitors to adjust the screen. By this they begin interacting with a laptop.*

**Python Notebooks is the excellent way to allow users to interact with analysis**



# Using usual DataScience tools to do analysis and plot data (without pyROOT)



The screenshot displays a JupyterLab environment with a file browser on the left and a code editor on the right. The file browser shows a directory structure with files like 'data', '06\_open\_charm.ipynb', '07\_geometry.ipynb', '08\_uproot\_neutron\_an...', '10\_run\_ejana.ipynb', '11\_ejana\_interface.ipynb', '12\_config.ipynb', 'widget\_example.ipynb', '01\_eic-smear-config.cpp', '02\_NNPSS11-2-nobuil...', '03\_open\_charm\_sm.root', '04\_geometryLEIC.gdml', '05\_root.example', and 'output.root'. The code editor shows the following code:

```
[3]: tree = uproot.open('./data/eventless_output.root')['eventless']['tree']
tracks_df = tree.pandas.df(["pdg_b", "p_b", "px_b", "py_b", "pz_b"])

[10]: tracks_df['vertical_angle'] = np.arcsin(tracks_df['py_b']/tracks_df['p_b'])
tracks_df['horizontal_angle'] = np.arctan(tracks_df['px_b']/tracks_df['pz_b'])
v_angle = tracks_df[tracks_df.pdg_b == 2212].vertical_angle.values
h_angle = tracks_df[tracks_df.pdg_b == 2212].horizontal_angle.values

Horizontal angle (x):  $\alpha_h = \arctan\left(\frac{p_x}{p_z}\right)$ 

Vertical angle (y):  $\alpha_v = \arcsin\left(\frac{p_y}{p}\right)$ 

[11]: fig, ax = plt.subplots()
h, xedges, yedges, im = ax.hist2d(h_angle, v_angle,
bins=[np.arange(-200,101,5)/1000,np.arange(-150,151,5)/1000],
norm=matplotlib.colors.LogNorm())
plt.colorbar(im, ax=ax)
plt.xlabel('Horizontal angle (x) [rad]')
plt.ylabel('Vertical angle (y) [rad]')
plt.title('Neutrons angle distribution')
ax.set_axisbelow(True)
plt.grid(True)
plt.show()
```

The plot, titled "Neutrons angle distribution", shows a 2D histogram of the horizontal angle (x) and vertical angle (y) in radians. The x-axis ranges from -0.20 to 0.10, and the y-axis ranges from -0.15 to 0.15. The plot shows a central cluster of data points, with a color bar on the right indicating the density of points, ranging from  $10^0$  to  $10^3$ .

# Pyjano allows to configure and run “performant core” from python

The screenshot shows a JupyterLab interface with a terminal window displaying the execution of the Pyjano core. The terminal output is as follows:

```
[2]: !ejana -Pplugins=hepmc_reader,open_charm -Popen_charm:smearing=1 -Pnvents=1000 /home/romanov/ceic/data/herwig6_e-p_5x100.hepmc

getcwd: /mnt/c/eic/pyjano_proto/notebooks
[INFO] Adding source: /home/romanov/ceic/data/herwig6_e-p_5x100.hepmc

[INFO] Initializing plugin "/home/romanov/eic/ejana/dev/compiled/plugins/hepmc_reader.so"
[INFO] Initializing plugin "/home/romanov/eic/ejana/dev/compiled/plugins/open_charm.so"
Suppressed exception in JEventManager::GetUserEventSourceGenerator!
Opening source "/home/romanov/ceic/data/herwig6_e-p_5x100.hepmc" - JEventSource_hepmc : BeAGLE generated Text file
JEventSource_hepmc: Opening TXT file /home/romanov/ceic/data/herwig6_e-p_5x100.hepmc
[INFO] Creating 8 processing threads ...

-----
Config. Parameters
-----
name                value
-----
AFFINITY              = 0
JANA:DEBUG_PLUGIN_LOADING = 0
JANA:DEBUG_THREADMANAGER = 0
JANA:MAX_NUM_OPEN_SOURCES = 1
JANA:QUEUE_DEBUG_LEVEL = 0
JANA:TASK_POOL_DEBUGLEVEL = 0
JANA:TASK_POOL_SIZE    = 200
JANA:THREAD_DEBUG_LEVEL = 0
JANA:THREAD_ROTATE_SOURCES = 1
JANA:THREAD_SLEEP_TIME_NS = 100
nevents              = 1000
nskip                 = 0
NTHREADS              = 8
open_charm:smearing  = 1
plugins               = hepmc_reader,open_charm
ROOT:EnableThreadSafety = 1

Start processing ...
OpenCharmProcessor: Init()

----- EVENT 0 -----
All threads have ended.
Event processing ended.
OpenCharmProcessor::Finish(). Cleanup

Final Report
-----
Source                Nevents  Queue  NTasks
```

# You can use python API in notebook or pure python

The screenshot displays a JupyterLab environment. The left sidebar shows a file browser with a list of files and folders, including 'data', '06\_open\_charm.ipynb', '07\_geometry.ipynb', '08\_uproot\_neutron\_an...', '10\_run\_ejana.ipynb', '11\_ejana\_interface.ipynb' (selected), '12\_config.ipynb', 'widget\_example.ipynb', '01\_eic-smear-config.cpp', '02\_NNPSS11-2-nobuil...', '03\_open\_charm\_sm.root', '04\_geometryLEIC.gdml', '05\_root.example', and 'output.root'. The main area shows a code editor with the following Python code:

```
[ ]: import pyjano

[5]: jana = pyjano.Jana()
      jana.configure(
          plugins=[
              'beagle_reader',           # a list of plugins to use:
              {'open_charm': [          # plugin name, no additional parameters
                  {'verbose': 1},       # add vmeson plugin & set '-Pvmeson:verbose=2' parameter
                  {'smearing': 1}]     # Set verbose mode for that plugin
              # Set smearing mode
          ]),
          in_files="/home/romanov/ceic/data/herwig6_e-p_5x100.hepmc", # or [list, of, files]

          params={'nthreads':4, 'nevents':2000} # for parameters that don't follow <plugin>:<name> naming
                                                    # Smart enough to run it like --nthreads=8
                                                    # instead of -P...
      )

JANA loaded...

[6]: jana.run()

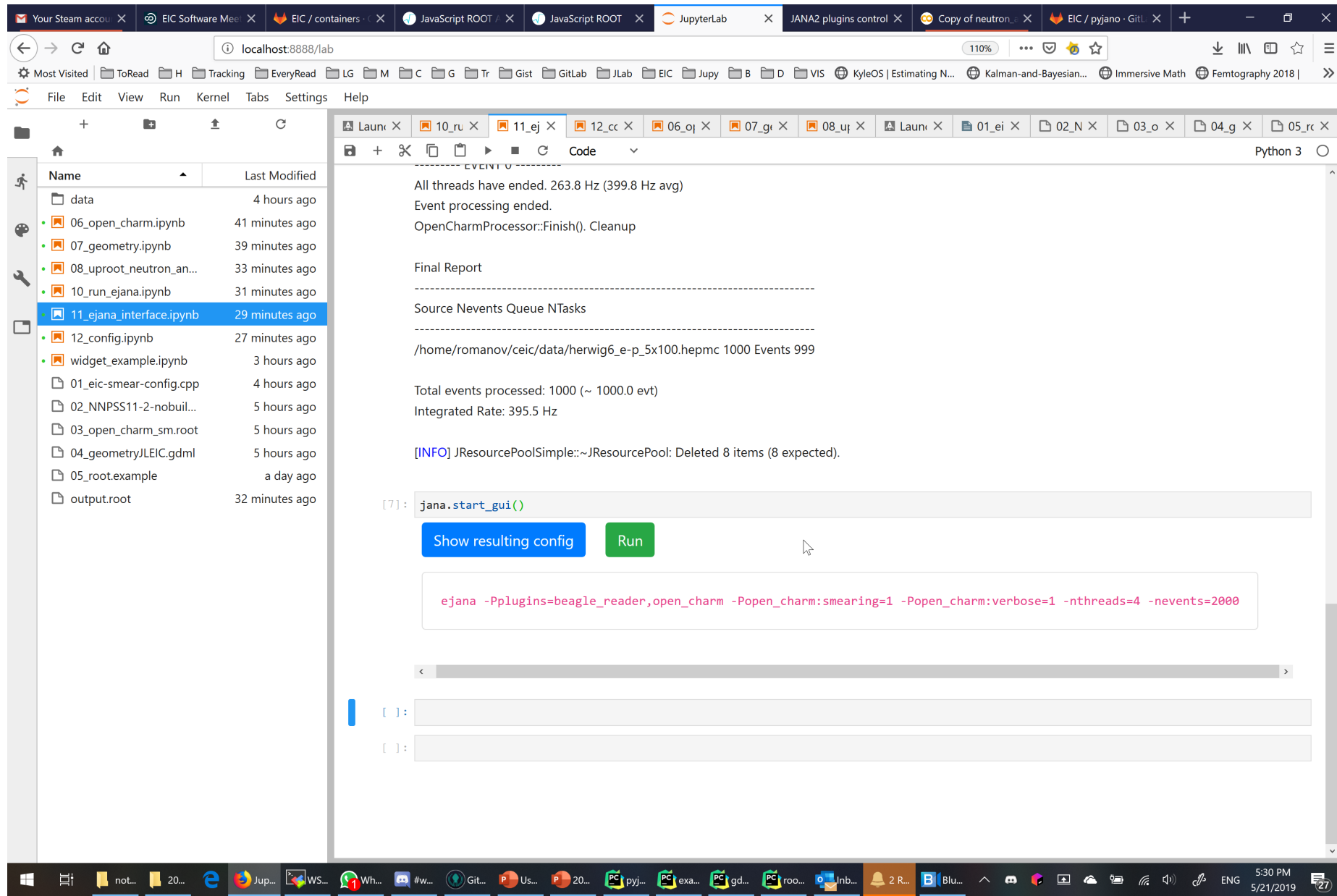
ejana -Pplugins=hepmc_reader,open_charm -Popen_charm:smearing=1 -Pnevents=1000 /home/romanov/ceic/data/herwig6_e-p_5x100.hepmc
getcwd: /mnt/c/eic/pyjano_proto
[INFO] Adding source: /home/romanov/ceic/data/herwig6_e-p_5x100.hepmc

[INFO] Initializing plugin "/home/romanov/eic/ejana/dev/compiled/plugins/hepmc_reader.so"
[INFO] Initializing plugin "/home/romanov/eic/ejana/dev/compiled/plugins/open_charm.so"
Suppressed exception in JEventManager::GetUserEventSourceGenerator!
Opening source "/home/romanov/ceic/data/herwig6_e-p_5x100.hepmc" - JEventManager_hepmc: BeAGLE generated Text file
JEventManager_hepmc: Opening TXT file /home/romanov/ceic/data/herwig6_e-p_5x100.hepmc
[INFO] Creating 8 processing threads ...

Config. Parameters
=====
name value
```

The bottom of the image shows the Windows taskbar with various application icons and the system clock indicating 5:29 PM on 5/21/2019.

# It provides ipywidgets with GUI inside notebooks to run “PC”



The screenshot displays a JupyterLab environment. The browser address bar shows `localhost:8888/lab`. The left sidebar contains a file explorer with a list of files and folders, including `data`, `06_open_charm.ipynb`, `07_geometry.ipynb`, `08_uproot_neutron_an...`, `10_run_ejana.ipynb`, `11_ejana_interface.ipynb` (highlighted), `12_config.ipynb`, `widget_example.ipynb`, `01_eic-smear-config.cpp`, `02_NNPSS11-2-nobuil...`, `03_open_charm_sm.root`, `04_geometryLEIC.gdml`, `05_root.example`, and `output.root`.

The main area shows a notebook with a code cell containing the following text:

```
-----EVENT U-----  
All threads have ended. 263.8 Hz (399.8 Hz avg)  
Event processing ended.  
OpenCharmProcessor::Finish(). Cleanup  
  
Final Report  
-----  
Source Nevents Queue NTasks  
-----  
/home/romanov/ceic/data/herwig6_e-p_5x100.hepmc 1000 Events 999  
  
Total events processed: 1000 (~ 1000.0 evt)  
Integrated Rate: 395.5 Hz  
  
[INFO] JResourcePoolSimple::~JResourcePool: Deleted 8 items (8 expected).
```

Below the code cell, there is a GUI element with a text input field containing the command:

```
jana.start_gui()
```

Two buttons are visible: a blue "Show resulting config" button and a green "Run" button. Below the input field, the command is displayed in a pink font:

```
ejana -Pplugins=beagle_reader,open_charm -Popen_charm:smearing=1 -Popen_charm:verbose=1 -nthreads=4 -nevents=2000
```

The bottom of the image shows the Windows taskbar with various application icons and the system clock indicating 5:30 PM on 5/21/2019.

# GUI to configure “preformant core” inside jupyter notebooks

The screenshot displays a JupyterLab environment within a web browser. The browser's address bar shows `localhost:8888/lab`. The interface includes a file browser on the left, a code editor in the center, and a terminal at the bottom.

The code editor shows the following Python code:

```
[6]: from pyjano import Jana
     jana = Jana()

[7]: jana.plugins_gui()
```

The `plugins_gui()` call has rendered a GUI with the following sections:

- IO plugins:**
  - lund\_reader
  - beagle\_reader
  - hepmc\_reader
  - jleic\_geant\_reader
  - jleic\_gemc\_reader
- Process & Analysis:**
  - trk\_fit
  - trk\_eff
  - jleic\_iff
  - jleic\_occupancy
  - vmeson
  - open\_charm
- verbose (int):**  2

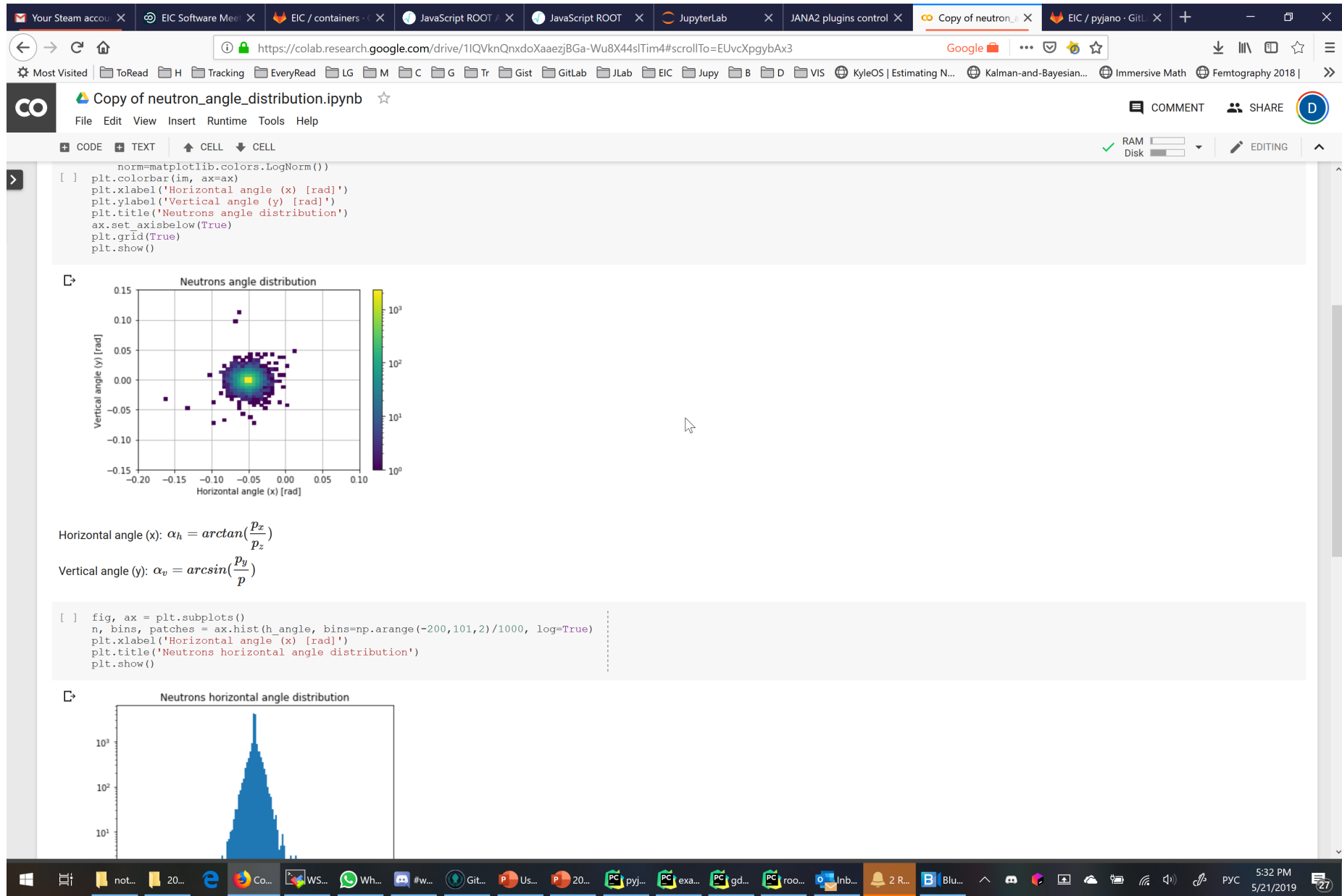
Below the GUI, there is a description for the `beagle_reader` plugin:

Plugin **beagle\_reader**: Opens files from BeAGLE event generator as a data source  
**BeAGLE - Benchmark eA Generator for LEptoproduction**  
[Documentation](#)

The bottom of the screen shows a Windows taskbar with various application icons and a system tray displaying the time as 5:30 PM on 5/21/2019.



# Can work in clouds (Google colab as example), central servers and Jlab jupyter lab servers. And other such things



The screenshot displays a Google Colab notebook interface. The browser tabs at the top include 'Your Steam account', 'EIC Software Meeting', 'EIC / containers', 'JavaScript ROOT', 'JupyterLab', 'JANA2 plugins control', 'Copy of neutron...', and 'EIC / pyjano - GitLab'. The notebook title is 'Copy of neutron\_angle\_distribution.ipynb'. The code cell contains the following Python code:

```
norm=matplotlib.colors.LogNorm()  
plt.colorbar(im, ax=ax)  
plt.xlabel('Horizontal angle (x) [rad]')  
plt.ylabel('Vertical angle (y) [rad]')  
plt.title('Neutrons angle distribution')  
ax.set_axisbelow(True)  
plt.grid(True)  
plt.show()
```

The output of this code is a 2D heatmap titled 'Neutrons angle distribution'. The x-axis is labeled 'Horizontal angle (x) [rad]' and ranges from -0.20 to 0.10. The y-axis is labeled 'Vertical angle (y) [rad]' and ranges from -0.15 to 0.15. A color bar on the right indicates a logarithmic scale from  $10^0$  to  $10^3$ . The plot shows a central cluster of data points with a color gradient from purple to yellow.

Below the heatmap, the following mathematical definitions are provided:

Horizontal angle (x):  $\alpha_h = \arctan\left(\frac{p_x}{p_z}\right)$

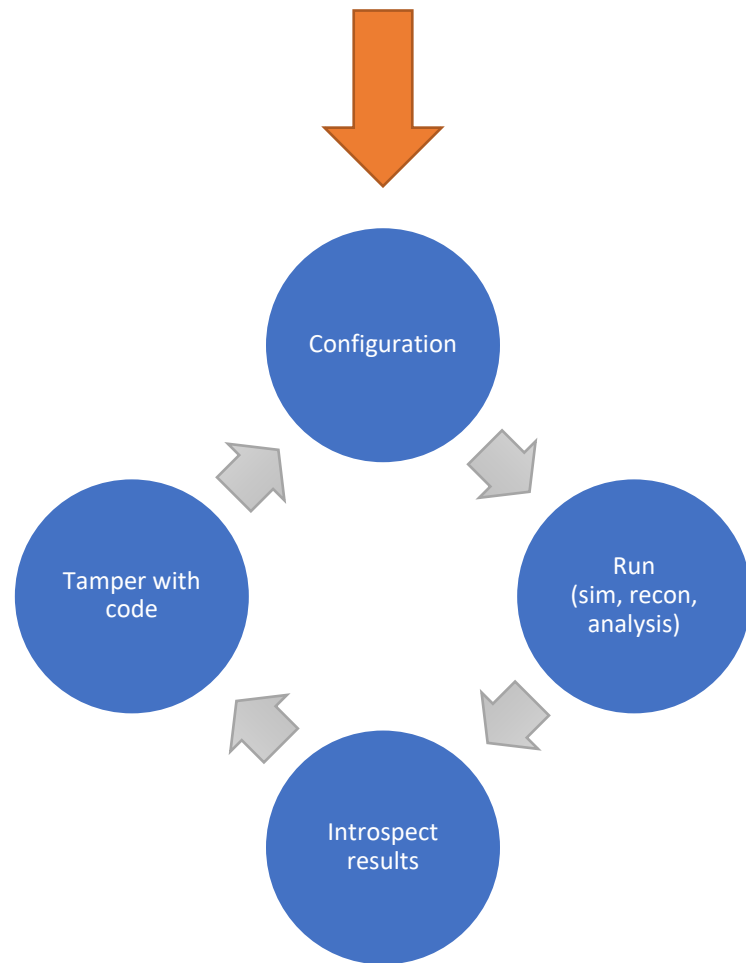
Vertical angle (y):  $\alpha_v = \arcsin\left(\frac{p_y}{p}\right)$

The next code cell contains the following Python code:

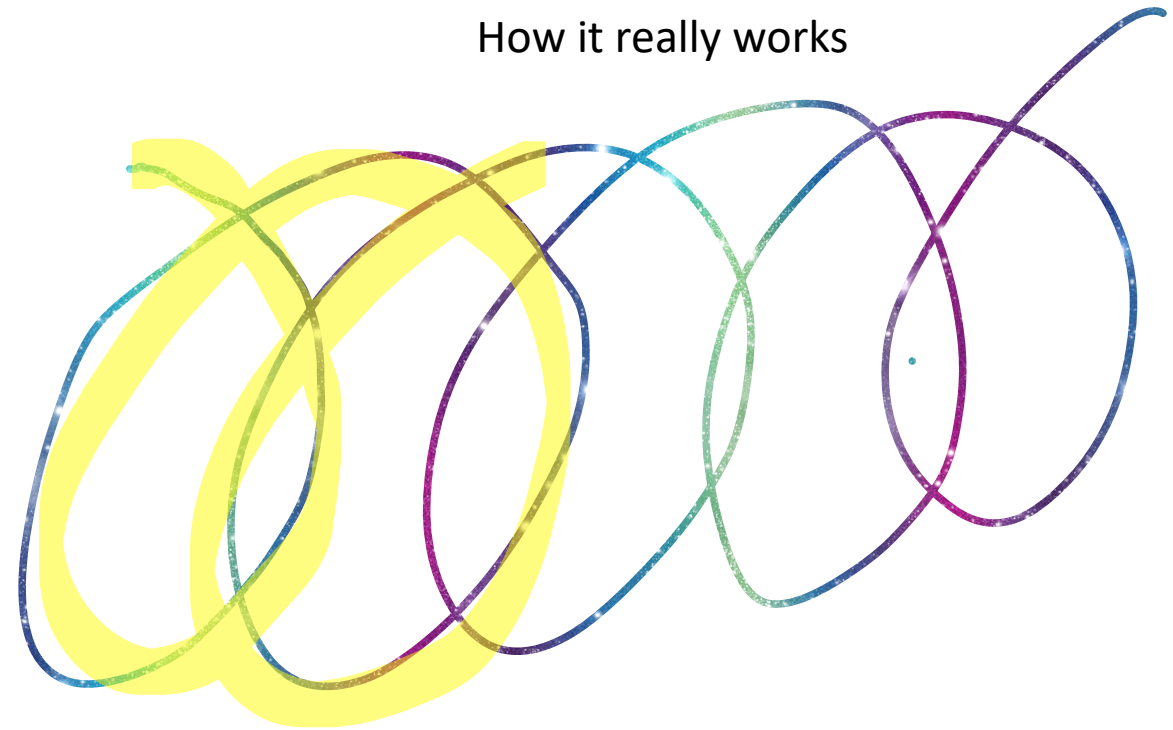
```
fig, ax = plt.subplots()  
n, bins, patches = ax.hist(h angle, bins=np.arange(-200,101,2)/1000, log=True)  
plt.xlabel('Horizontal angle (x) [rad]')  
plt.title('Neutrons horizontal angle distribution')  
plt.show()
```

The output of this code is a histogram titled 'Neutrons horizontal angle distribution'. The x-axis is labeled 'Horizontal angle (x) [rad]' and ranges from -0.20 to 0.10. The y-axis is labeled 'Frequency' and ranges from  $10^1$  to  $10^3$ . The histogram shows a distribution of data points centered around 0.00, with a color gradient from purple to yellow.

# Can we identify entry point workflows



How it really works

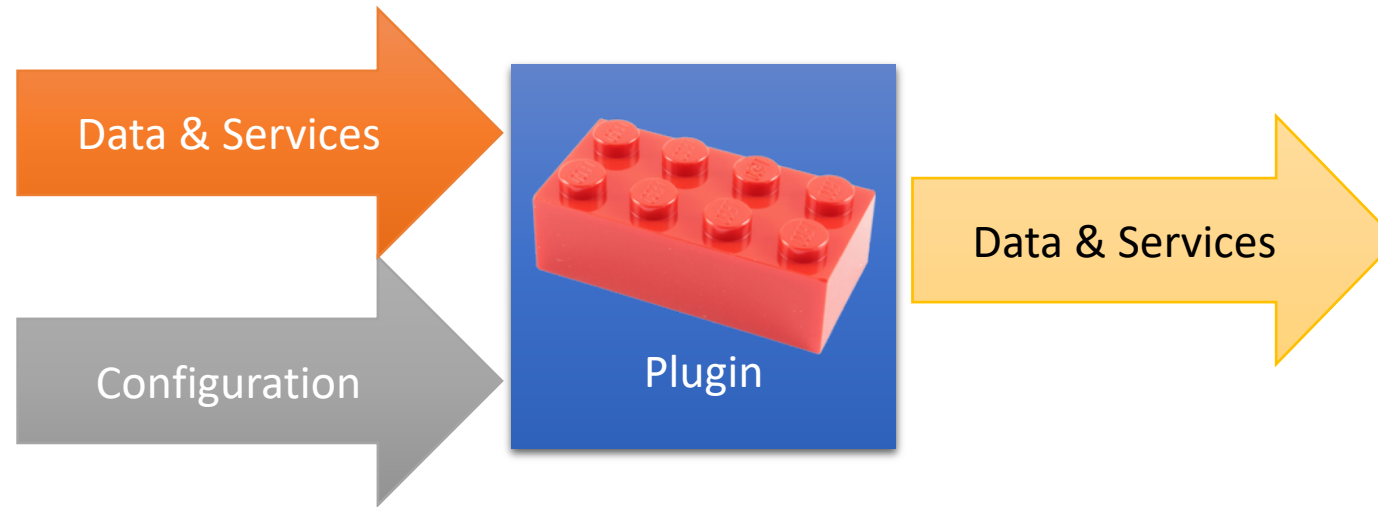


\* (Yellow lines depict how such example notebooks and documentation can guide users through cycles of analysis development)



# JANA2 modularity

*(what is important for this talk)*



Pretty standalone .so library

# Do we hide the complexity?

Jupyter lab, GUI,

Python, scripts, analysis

C++, eJANA, plugins

JANA, eic-smear, ROOT, Geant4

# EJPM

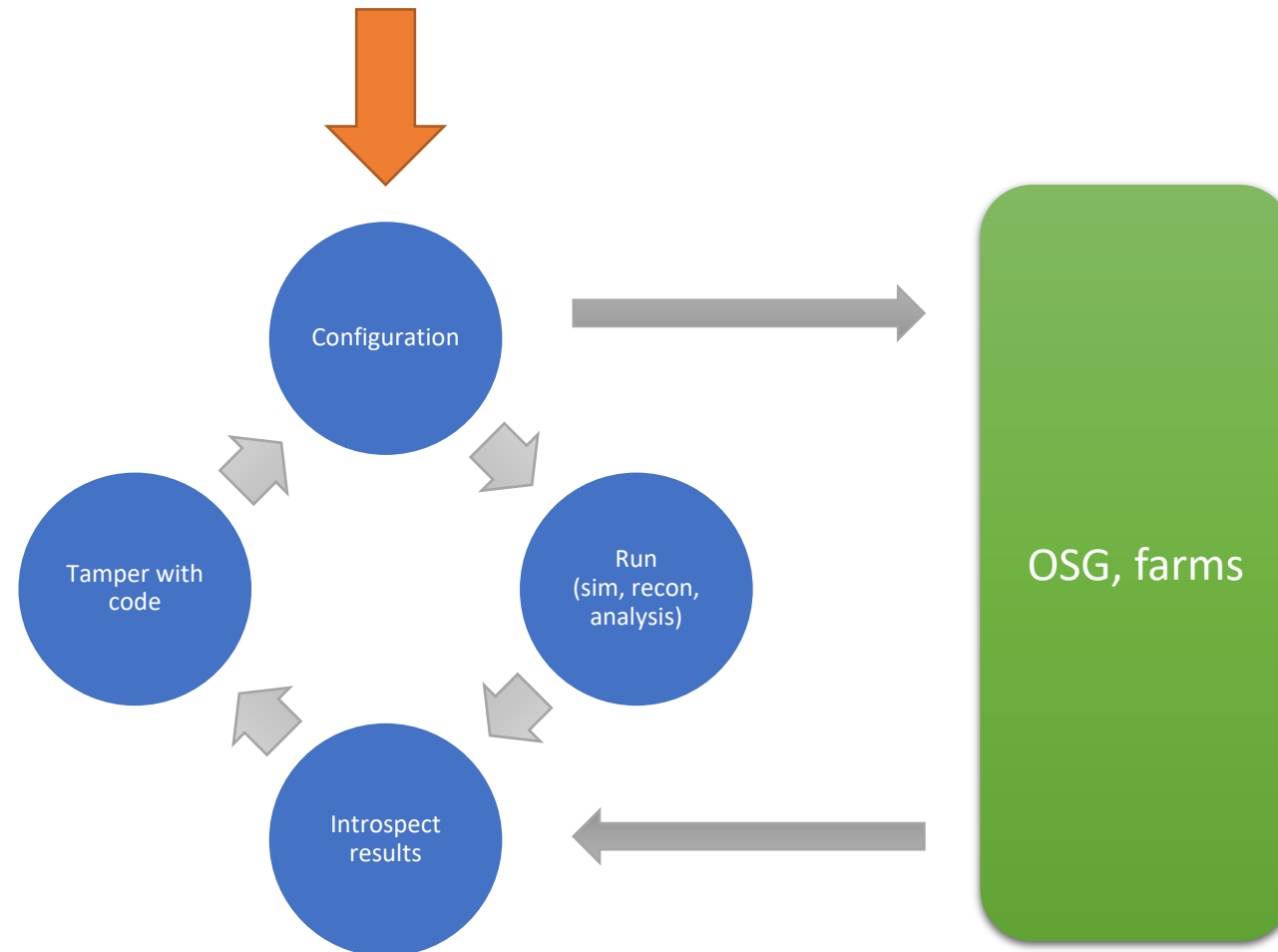
- EJPM stands for eJana ~~P~~acket-Build Manager

```
> pip install ejpm # self descriptive CLI interface
```

- Never designed to be a real packet manager
- Provides unified build and deployment tool for
  - workstations
  - containers
  - cloud deployment

Cloud?

# Can we identify entry point workflows



# Distribution way

Cloud

Containers

Conda

Workstation  
Compilation  
**EJPM**

NO EFFORT AT ALL

Novice

*Efforts required axis*

Some effort

Experts



# What is ready?

**JANA2** – C++ framework for NP data processing - **beta**

**e<sup>JANA</sup>** – EIC community reference reconstruction - **beta** (June)

**g4e** – Geant 4 EIC – **alpha-> beta** (June)

**ejpm** – Packet manager builder - **beta**

**deic** – Docker helper for EIC users - *in project*

**Eic-smear** – Integrated into ejpm,  
(full integration into ejana in progress) - **beta** (June)

**pyjano** – Python JANA Orchestrator – **alpha**

**eic-jupyterlab** – GUI for users entry point – *prototype-> alpha*

## Future announces?

Follow EIC Software consortium & User Groups meetings:

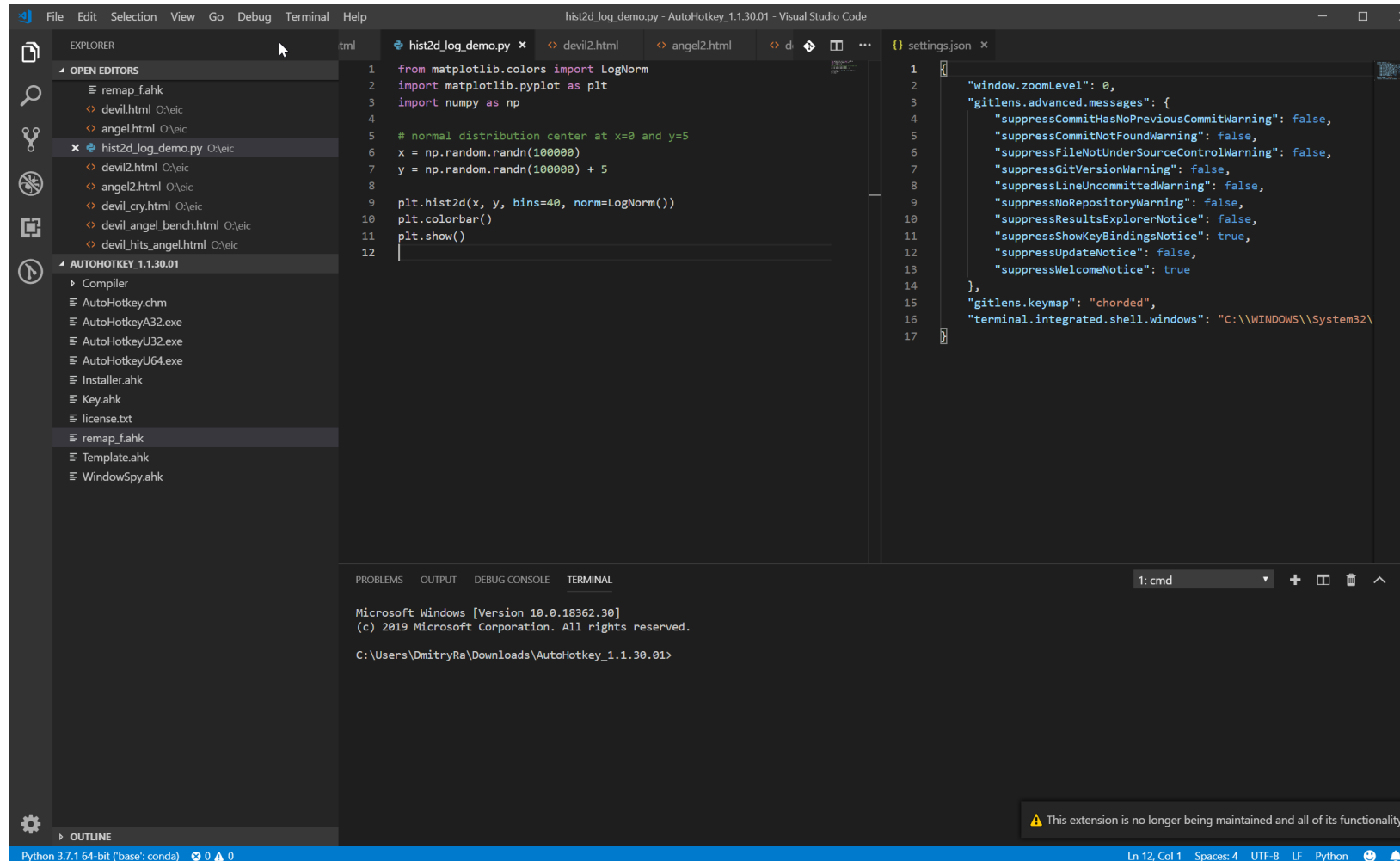
*May UG meeting: <https://agenda.infn.it/event/17249/>*



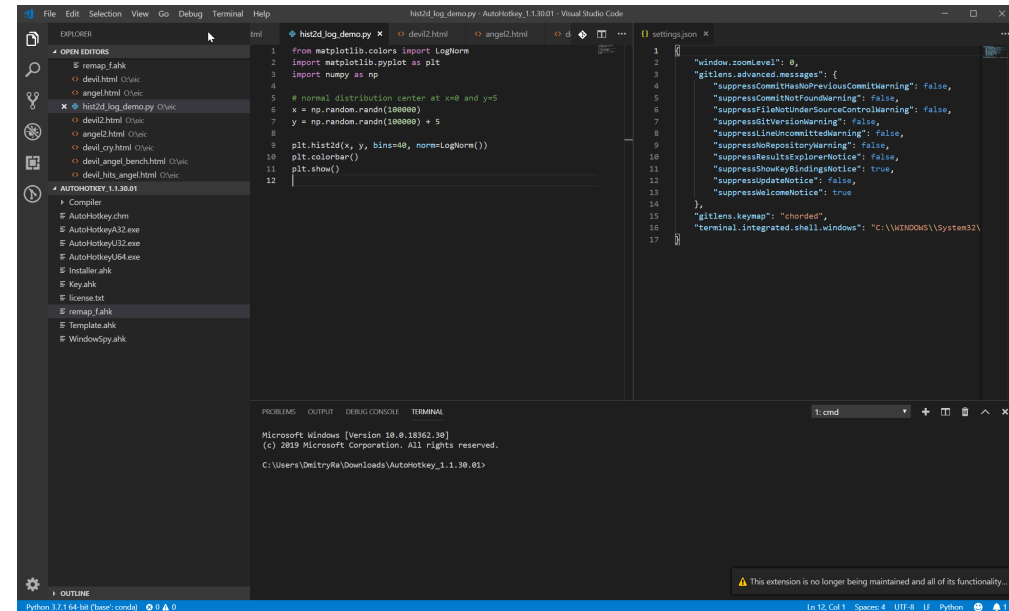
Thank you!



# GUI for CODE related work (success story)



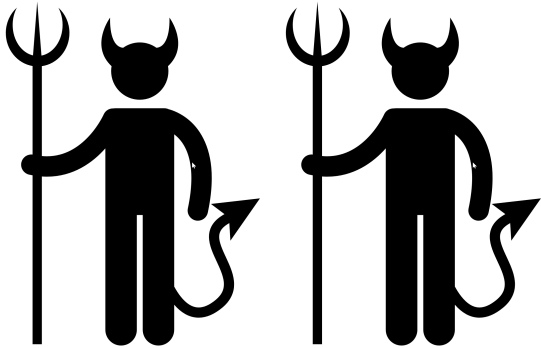
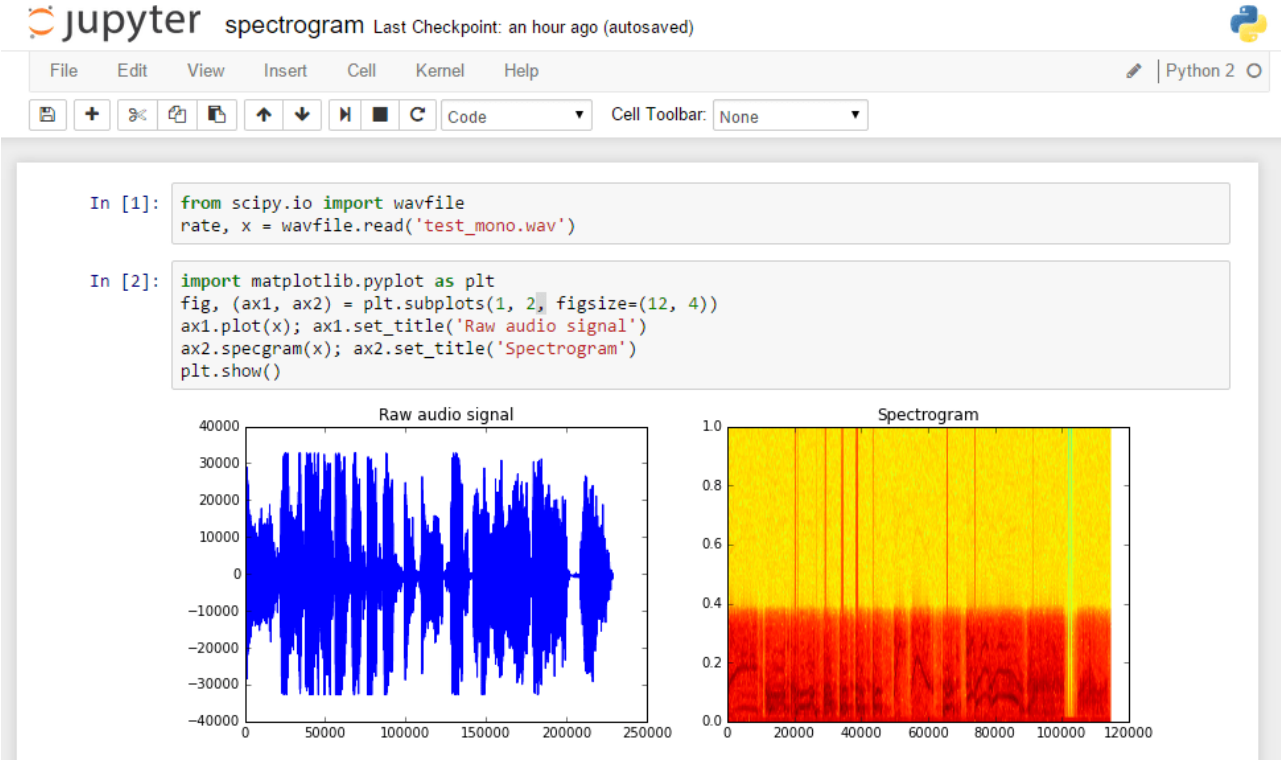
# Web GUI vs Native GUI



More like WEB gui

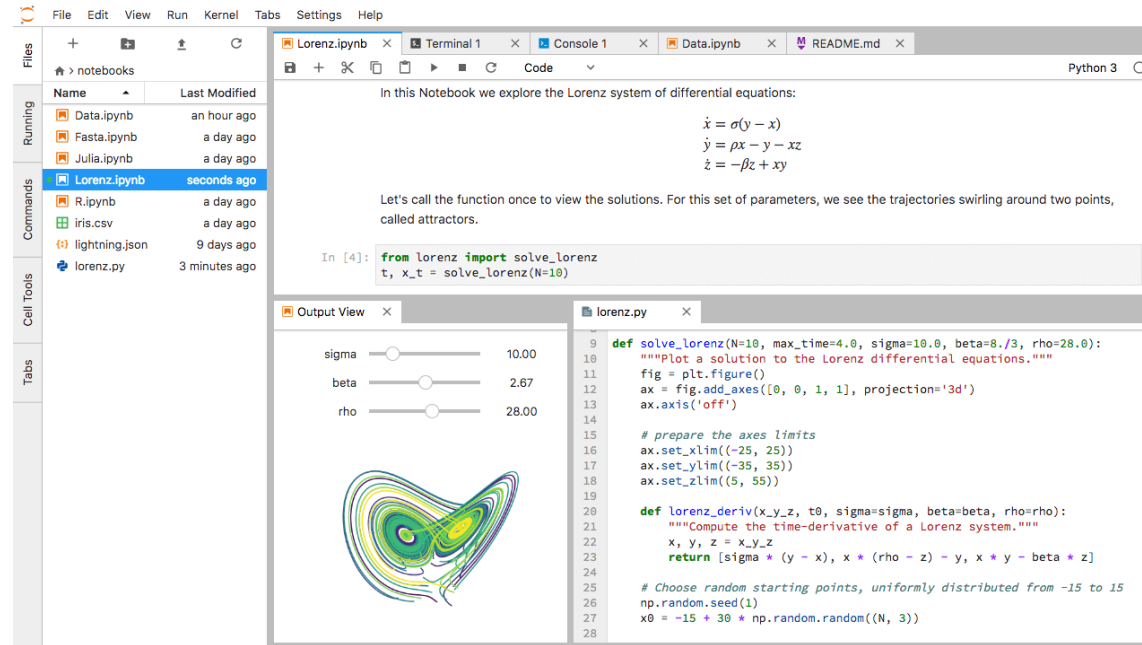
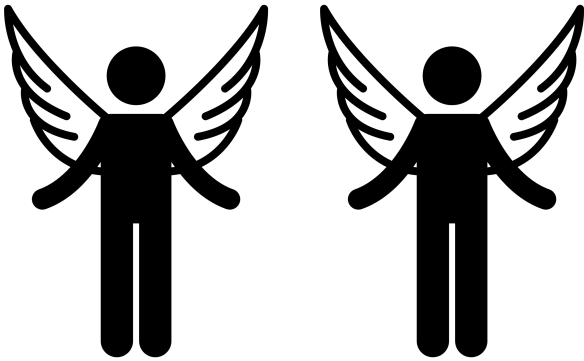


# Jupyter notebooks



# Jupyter lab

- Modular + plugins
- One place to tamper with scripts and see the output
- Better interacts with system
- Can be used as SAAS



The screenshot displays the Jupyter Lab environment. On the left, a sidebar shows a file browser with notebooks like 'Data.ipynb', 'Fasta.ipynb', 'Julia.ipynb', and 'Lorenz.ipynb'. The main area shows a notebook with the following content:

In this Notebook we explore the Lorenz system of differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy\end{aligned}$$

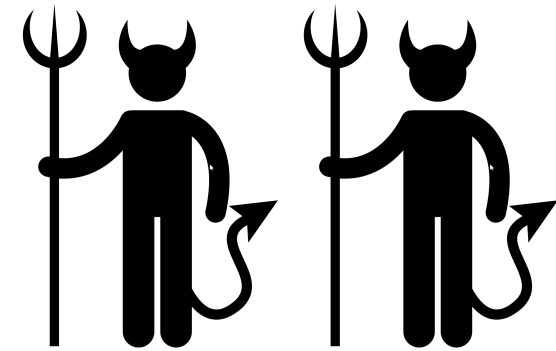
Let's call the function once to view the solutions. For this set of parameters, we see the trajectories swirling around two points, called attractors.

```
In [4]: from lorenz import solve_lorenz
t, x_t = solve_lorenz(N=10)
```

The 'Output View' shows a 3D plot of the Lorenz attractor with sliders for parameters: sigma = 10.00, beta = 2.67, and rho = 28.00. The 'Code' view shows the following Python code:

```
9 def solve_lorenz(N=10, max_time=4.0, sigma=10.0, beta=8./3, rho=28.0):
10     """Plot a solution to the Lorenz differential equations."""
11     fig = plt.figure()
12     ax = fig.add_axes([0, 0, 1, 1], projection='3d')
13     ax.axis('off')
14
15     # prepare the axes limits
16     ax.set_xlim((-25, 25))
17     ax.set_ylim((-35, 35))
18     ax.set_zlim((5, 55))
19
20     def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
21         """Compute the time-derivative of a Lorenz system."""
22         x, y, z = x_y_z
23         return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]
24
25     # Choose random starting points, uniformly distributed from -15 to 15
26     np.random.seed(1)
27     x0 = -15 + 30 * np.random.random((N, 3))
28
```

- Still in development
- Some workflows are there
- Have many limitations compared to a native GUI



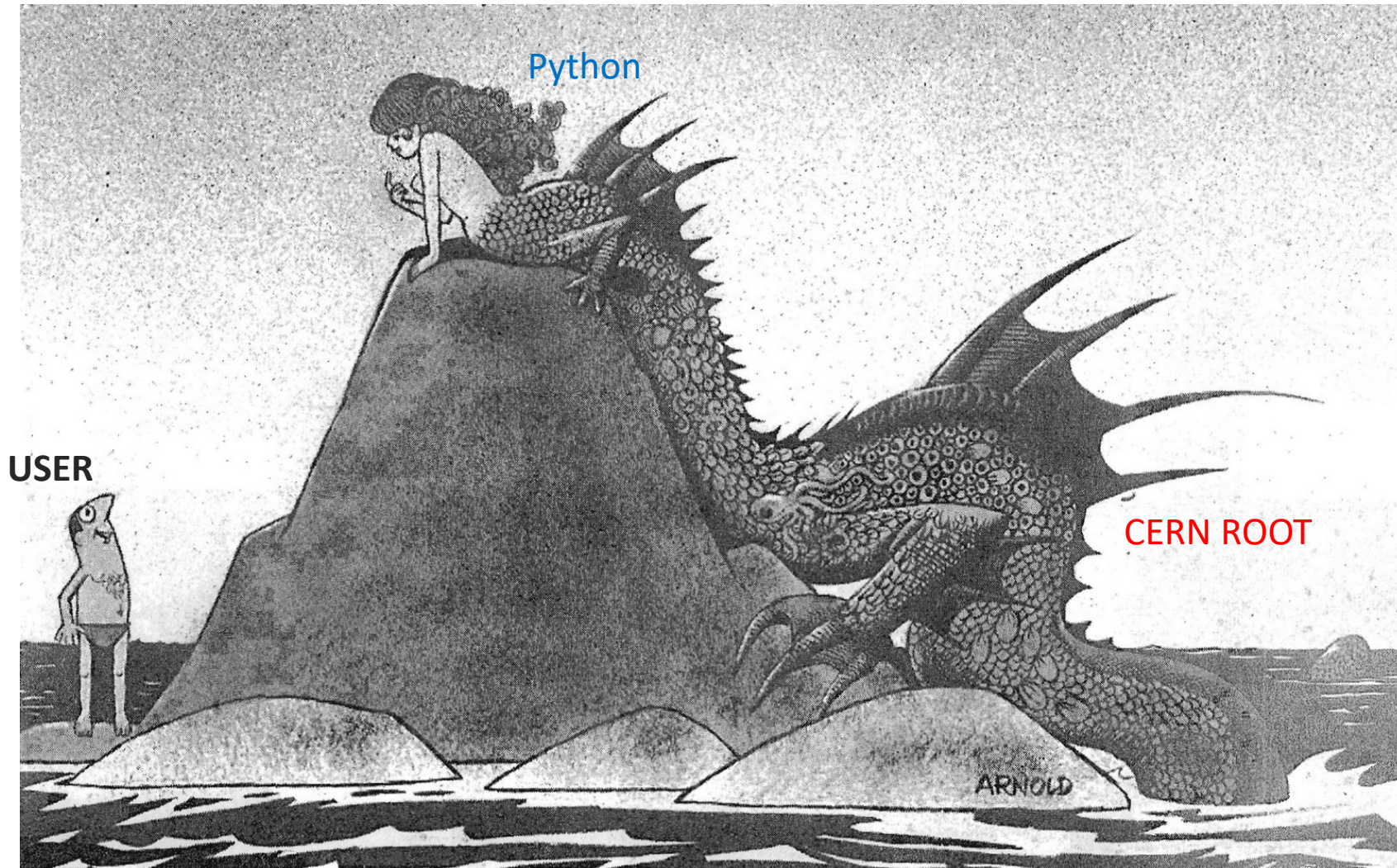
# Conclusions

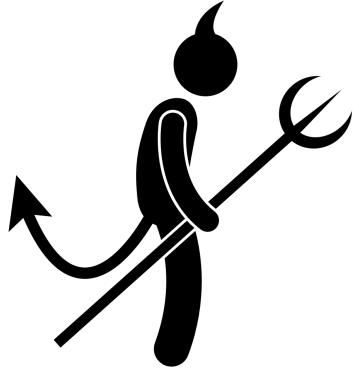
A heated discussion should conclude this talk!



*We will present our thoughts on users entry point in terms of  
working prototype  
in the next talk*

# Root with a python interface





Good workflow

VS

Good tool

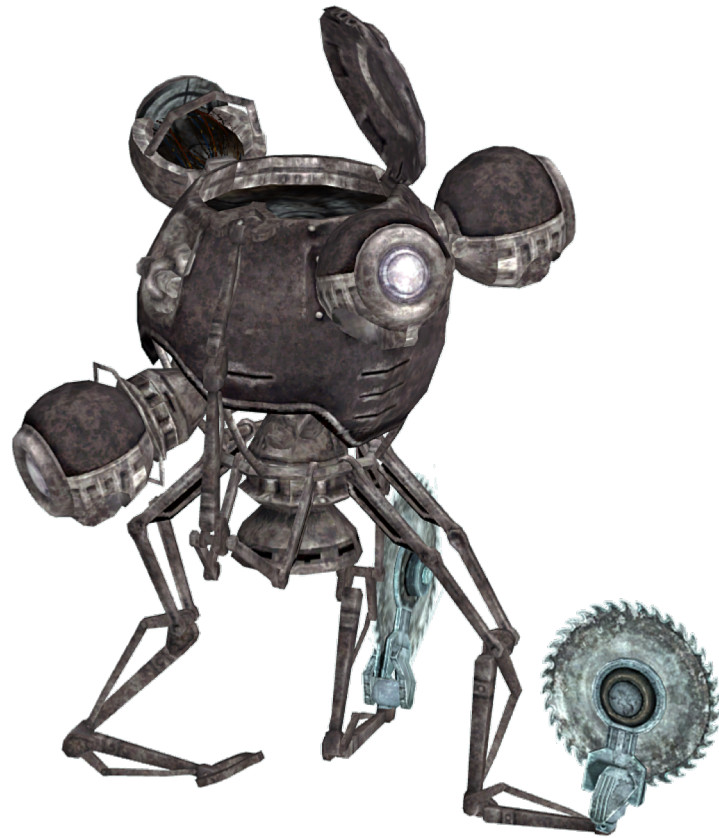


Even for code and libs:

Github -> Tutorial -> Selling you workflow of your future work



# Backup slides



# C++ everywhere is a strange choice

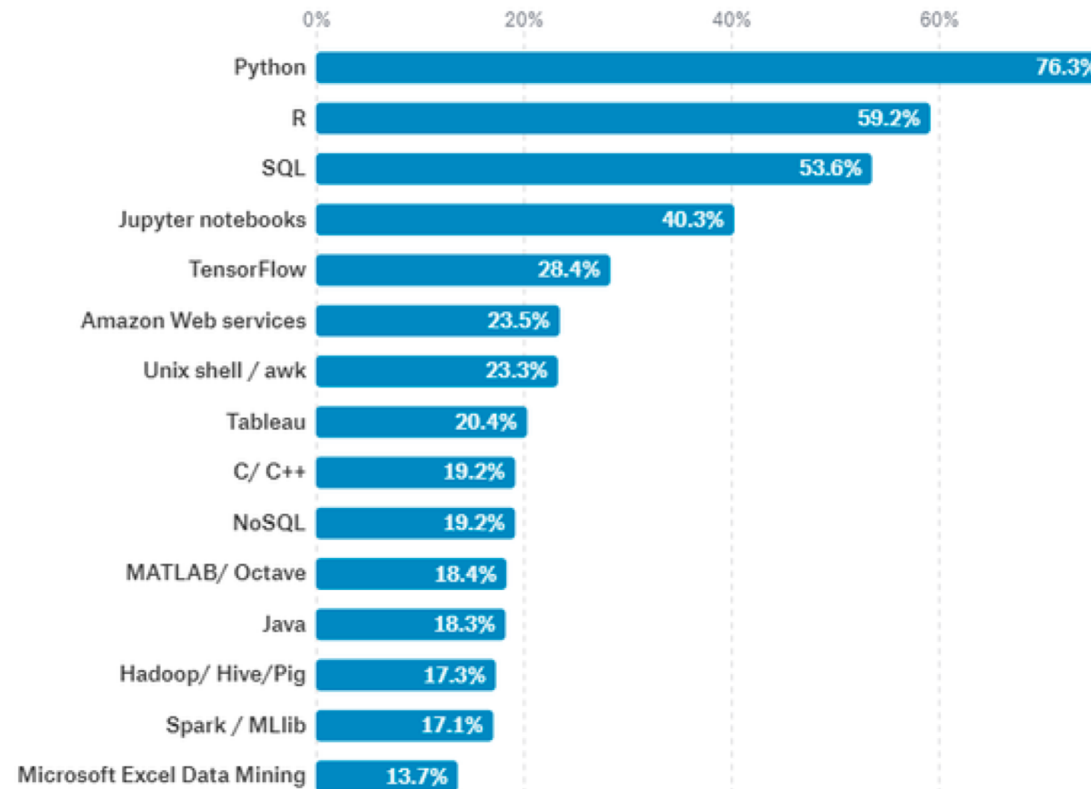


**Mario Alemi**

Physicist

... So I started using ROOT. I was eager to learn C++. I took courses, read Bjarne Stroustrup's book. And then asked myself –did ROOT people *first* get a good dose of LSD (lysergic acid) and *after* decided to use C++ as scripting language?

# Data science languages



# Lets start with ROOT



[Download](#) [Documentation](#) [News](#) [Support](#) [About](#) [Development](#) [Contribute](#)



**Getting Started**



**Reference Guide**



**Forum**



**Gallery**

## ROOT is ...

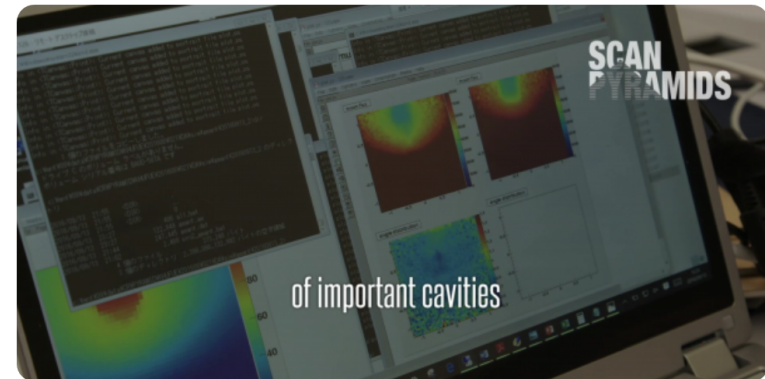
A modular scientific software toolkit. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualisation and storage. It is mainly written in C++ but integrated with other languages such as Python and R.

[Start from examples](#) or [try it in your browser!](#)



**Download ROOT**

or [Read More ...](#)



[Previous](#) [Pause](#) [Next](#)

**Our major elephant is the software room**

# STUPID

VS

# SOLID

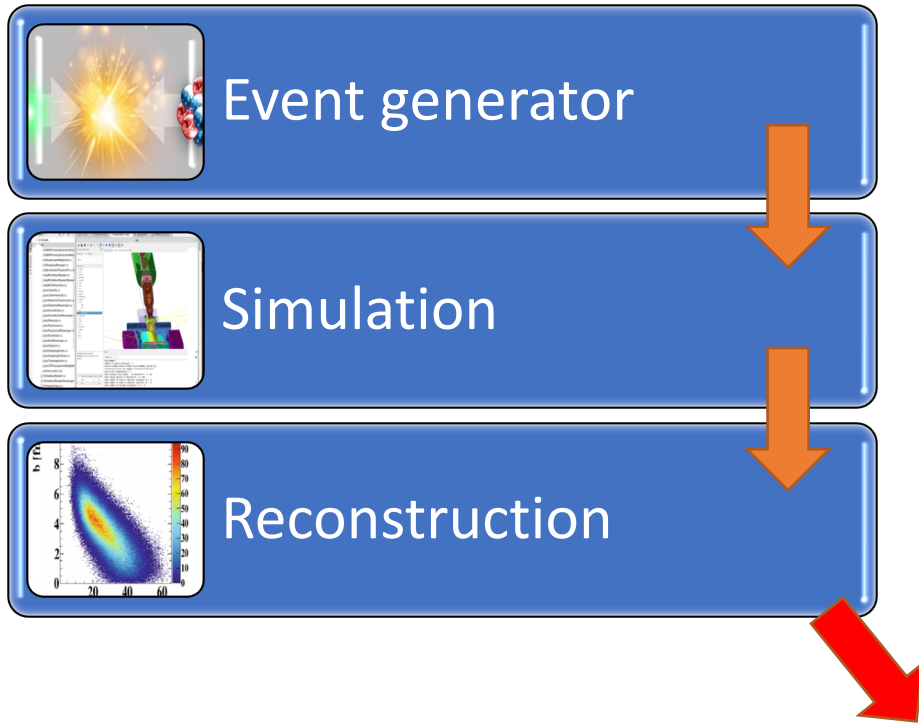
- +++ • **S**ingleton
- +++ • **T**ight coupling
- +++ • **U**ntestability
- • **P**remature Optimization
- + - • **I**ndescriptive Naming
- + • **D**uplication

- - • **S**ingle responsibility
- • **O**pen/closed
- + - • **L**iskov substitution
- + - • **I**nterface segregation
- - • **D**ependency inversion



# Thinking in workflows

## Docker containers



## Your own system install

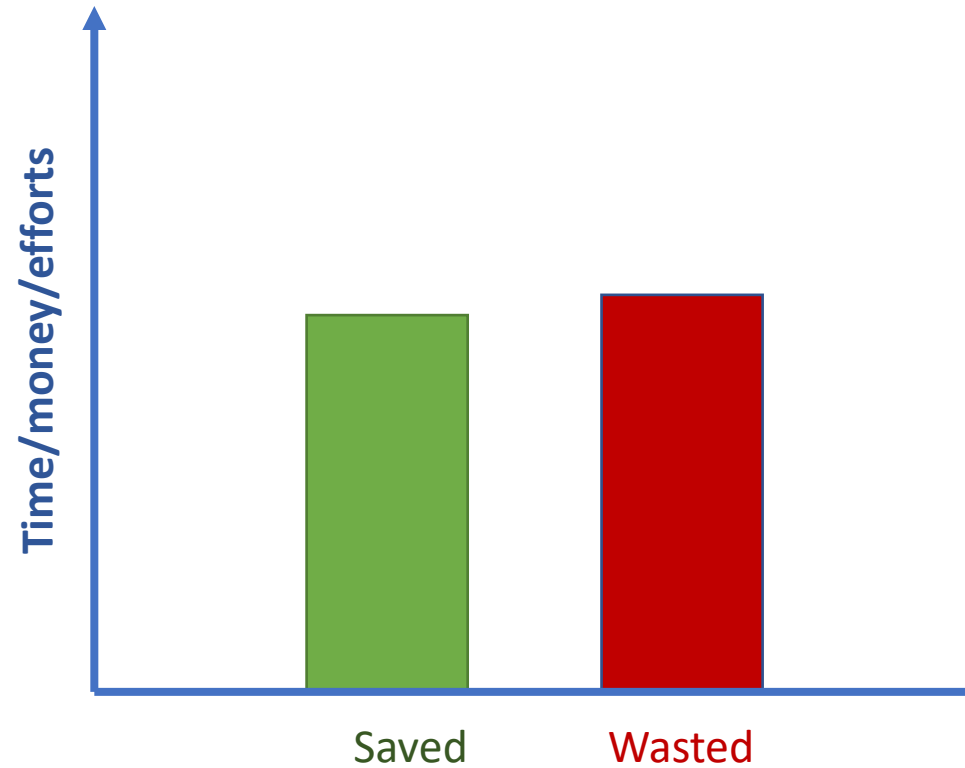


Installs Geant, g4e, Jana2, Rave...

User

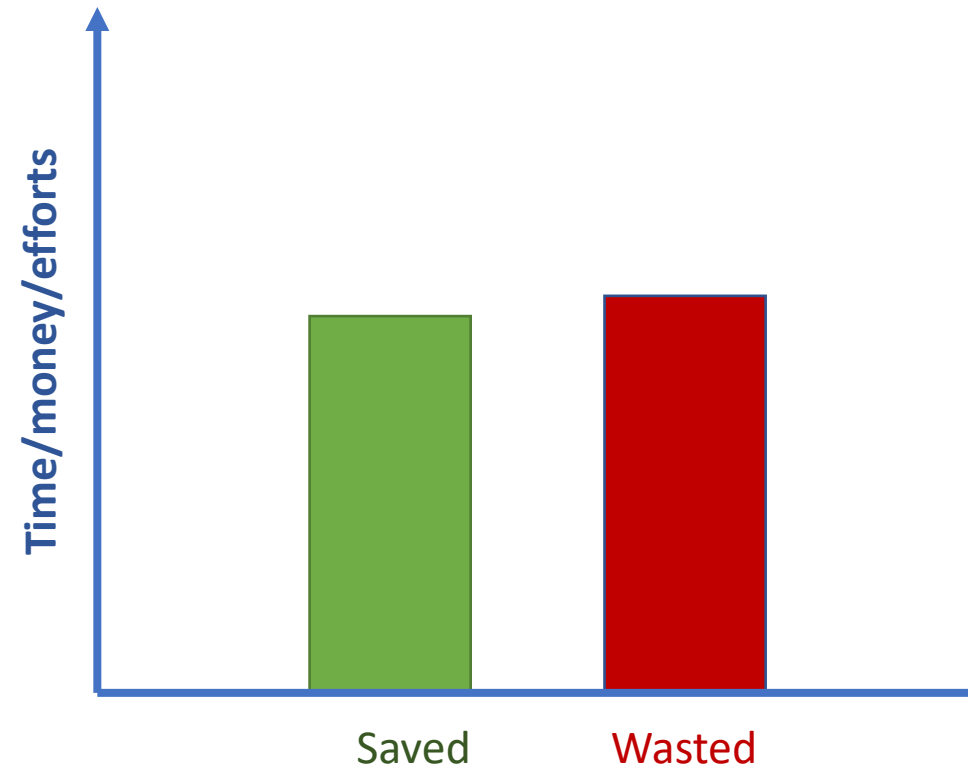
# Unexperienced developers

Time & money saved on team of professional C++ developers



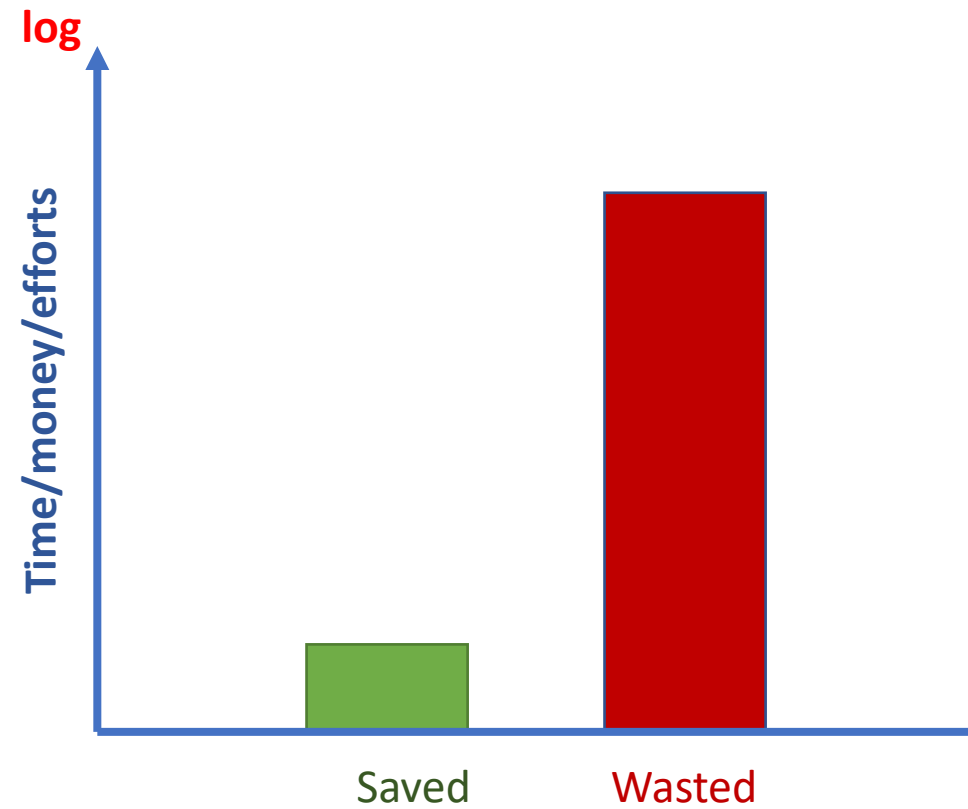
Time & money wasted by students, postdocs, professors and scientist on ROOT failures

# Unexperienced developers





# Unexperienced developers





# ROOT 7 will save us!

- MUCH better API
- Professional team (love their users)
- Work being done on improving modularity

```
Fill a series of histograms
New interfaces
1. #include "ROOT/TH1s.h"
2. #include "ROOT/TFile.h"
3. #include "ROOT/TH2s.h"
4. #include "ROOT/TH3s.h"
5. #include "TTreeReader.h"
6. #include "TTreeReaderArray.h"
7. #include "TTree.h"
8. // Another function. Who knows what it does in a month from now.
9. void someOtherFunction()
10.
11. void fill(TTree* tree)
12. {
13.     using namespace ROOT::Experimental;
14.     // Define the axes, we will re-use them
15.     TAxisConfig axisMUP("Muon pT [GeV]", {0, 1., 10., 100.});
16.     TAxisConfig axisJES("Jet ET [GeV]", {0, 0., 1000.});
17.     TAxisConfig axisTag("tag value", {0, 0., 1.});
18.
19.     // Create the histograms
20.     TH2F hMUPTag("muon pT versus tag value", axisMUP, axisTag);
21.     TH2F hJESTag("Jet ET versus tag value", axisJES, axisTag);
22.
23.     // Set up reading from the TTree
24.     TTreeReader reader(tree);
25.     TTreeReaderArray<TObject> jets(reader, "jet_eta_mu_pt");
26.     TTreeReaderArray<TObject> muPs(reader, "jet_eta_mu_pt");
27.     TTreeReaderArray<TObject> tagReader("jet_tag");
28.
29.     // Fill the histograms
30.     while (reader.Next()) {
31.         for (int iJet = 0; iJet < jets.GetSize(); ++iJet) {
32.             hMUPTag.Fill(muPs[iJet], tagReader[iJet]);
33.             hJESTag.Fill(jets[iJet], tagReader[iJet]);
34.         }
35.     }
36.
37.     someOtherFunction();
38.
39.     // Store the result, can create the file where really needed.
40.     TFile* file = TFile::Open("jet_muon_tag.root");
41.     file->Write("hMUPTag", hMUPTag);
42.     file->Write("hJESTag", hJESTag);
43. }
```

## ROOT 7 problem

It is for tomorrow while we need it **yesterday**



Can we live without CERN.ROOT?

**NO**

At least not today

Yes. We have trapped ourselves



- ROOT is dead. Long live the ROOT!

# CERN root-7

- Example (and what could go wrong)  
<https://root.cern.ch/root-7>

# Things to consider

## **DISCLAIMER:**

- Some things in this talk are **subjective** and **opinionated**. I want this talk to be a discussion. You don't agree – excellent, lets discuss it.
- **We, us, our** = HEP & NP developers in general (most of the time)

## **ASSUMPTIONS:**

- Our software does physics
- Our software is technically OK
- Are users happy with our software?

# HOW 2019 impression



Users are not happy with  
our (*HEP&NP*) software

(especially in analysis and  
reconstruction part)

# Usual view on users vs developers

