Bundesministerium
für Bildung
und Forschung

Belle II

The Belle II Analysis Framework

PyHEP 2018 Workshop – Sofia, Bulgaria

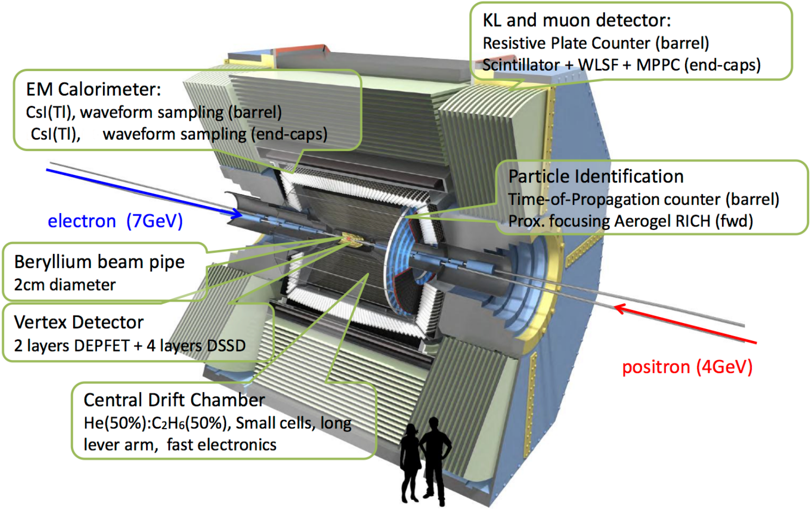
Thomas Hauth for the Belle II Collaboration
Institute of Experimental Particle Physics (ETP), KIT, Germany

The Belle II Experiment and its Goals

- KEKB was an electron-positron collider at KEK in Tsukuba/Japan which studied the decay of B mesons at the Υ(4S) resonance
- Nobel Prize in Physics 2008 to Kobayashi and Maskawa
- The SuperKEKB collider and the Belle II detector will build on the previous success:
  - Study the B meson system in far greater precision
  - Probe for new physics in a wide range of interesting topologies
  - Spectroscopy of Quarkonium systems
- The Belle II Collaboration: 756 members from 104 institutes in 25 countries

	KEKB	Super KEKB	Factor
Instantaneous Luminosity	$2 \times 10^{34}\,\mathrm{cm}^{-2}\mathrm{s}^{-1}$	$8 \times 10^{35}\,\mathrm{cm}^{-2}\mathrm{s}^{-1}$	40
Integrated Luminosity	$1\,\mathrm{ab}^{-1}$	$50\,\mathrm{ab}^{-1}$	50
Runtime	1998 to 2010	start in 2017	
Detector	Belle	Belle II	
Raw Data	1 PB	100 PB (projected)	100

Belle II Detector



KL and muon detector:
Resistive Plate Counter (barrel)
Scintillator + WLSF + MPPC (end-caps)

EM Calorimeter:
CsI(Tl), waveform sampling (barrel)
CsI(Tl), waveform sampling (end-caps)

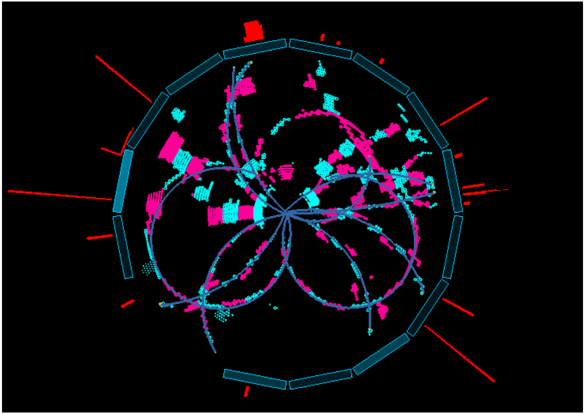
Particle Identification
Time-of-Propagation counter (barrel)
Prox. focusing Aerogel RICH (fwd)

Beryllium beam pipe
2cm diameter

Vertex Detector
2 layers DEPFET + 4 layers DSSD

Central Drift Chamber
He(50%):C_2H_6(50%), Small cells, long lever arm, fast electronics

First Collisions !



One of the first hadronic events recorded with the Belle II Detector at 2:27 a.m. JST on the 26th April 2018

... not much python involved here, sorry ...

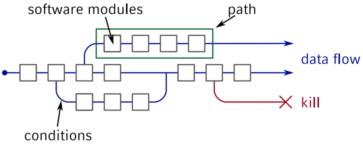
The `basf2` Framework

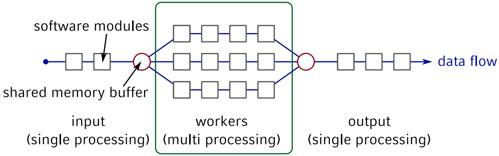
Mainly written from scratch using experiences from Belle and other experiments

- utilizes new technologies: C++14 (GCC 7.3), ROOT 6, Geant 4.10, Python 3.6
- Python 3 as steering/scripting language
- ROOT for input/output (also raw data)
- parallel processing support using `fork`

Framework Design:

- *Modules* are individual units of processing and use a common data store to read event data and write back results (not to be confused with python modules)
- All processing steps for recorded and simulated events are implemented in `basf2`: Event generation, simulation, digitization, online trigger, reconstruction and analysis
- Important libraries are bundled into *externals*: ROOT, gcc, Geant4, . . .





Python usage in the basf2 Framework

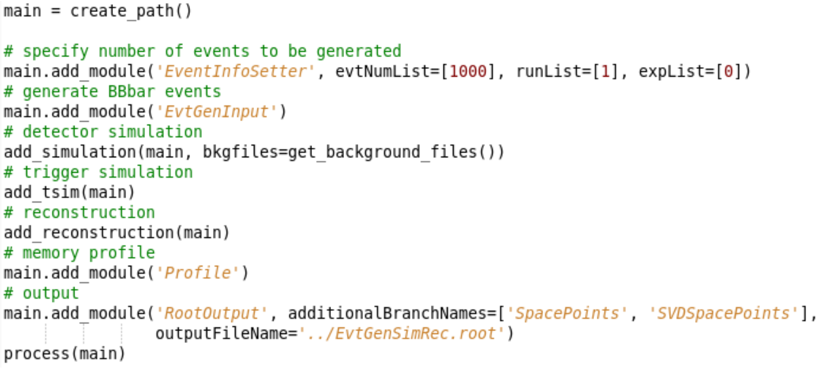
- `python 3.6.1` included in the externals of our framework - ensures the same modern python environment on all supported platforms (even Scientific Linux 6)

Python is a first-class citizen in our framework:

- Steering files connect modules to paths and are written in Python 3
- Framework Modules can be written in C++ and Python
- Framework functionality is exported to Python via `boost::python`
- User classes and all objects within the framework's **DataStore** are available in Python via **PyROOT**

Python Steering files

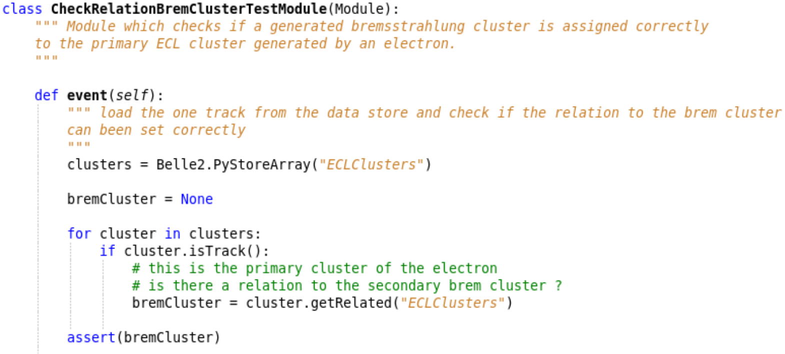
Steering files configure the execution modalities (which calbration database to use etc.) and the modules executed for each event:



```
main = create_path()

# specify number of events to be generated
main.add_module('EventInfoSetter', evtNumList=[1000], runList=[1], expList=[0])
# generate BBbar events
main.add_module('EvtGenInput')
# detector simulation
add_simulation(main, bkgfiles=get_background_files())
# trigger simulation
add_tsim(main)
# reconstruction
add_reconstruction(main)
# memory profile
main.add_module('Profile')
# output
main.add_module('RootOutput', additionalBranchNames=['SpacePoints', 'SVDSpacePoints'],
                outputFileName='../EvtGenSimRec.root')
process(main)
```

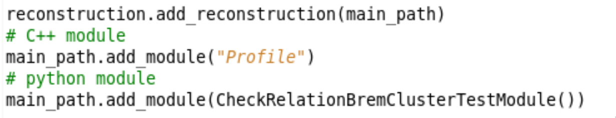
Python Modules

Python modules use the same entry methods as our C++ modules (`beginRun()`, `event()` for each processed event etc.)



```
class CheckRelationBremClusterTestModule(Module):
    """ Module which checks if a generated bremsstrahlung cluster is assigned correctly
    to the primary ECL cluster generated by an electron.
    """

    def event(self):
        """ load the one track from the data store and check if the relation to the brem cluster
        can been set correctly
        """
        clusters = Belle2.PyStoreArray("ECLClusters")

        bremCluster = None

        for cluster in clusters:
            if cluster.isTrack():
                # this is the primary cluster of the electron
                # is there a relation to the secondary brem cluster ?
                bremCluster = cluster.getRelated("ECLClusters")

        assert(bremCluster)
```

Python modules can be added to the processing path as easy as C++ modules:



```
reconstruction.add_reconstruction(main_path)
# C++ module
main_path.add_module("Profile")
# python module
main_path.add_module(CheckRelationBremClusterTestModule())
```

Jupyter Notebooks

Jupyter inherits all benefits available to Python in our framework, plus:

- Notebooks
    - Save expressions and corresponding results in one place
    - Include comments, documentation, pictures, drawings, LaTeX, videos
    - Send notebooks, including all results, to someone else (use-cases: software examples, bug reports)
    - Perform analyses interactively
- Clickable widgets in HTML and JavaScript
- Sections of a notebook can be executed individually
- Tab-completion and syntax highlighting
- Server–client structure
    - Access the Jupyter service from your home computer, smartphone, tablet, etc. but run the calculations on a high-performance machine
    - No need to rely on X forwarding or other technologies
- Many data science tools with Jupyter integration: ROOT, matplotlib, pandas
- Not only for Python (Haskell, Julia, C++, ROOT, Terminal)

Under the Hood: Integrating basf2

We developed the Python library `hep_ipython_tools` [1] which simplifies the integration of HEP Frameworks with jupyter notebooks.

Core component for seamless Jupyter integration of `basf2`:
Process handler for background framework execution

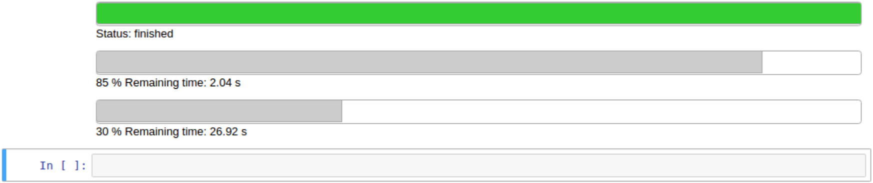
- Creates a separate worker process for `basf2`
- Transfers path configuration and starts processing
- Monitors running framework process
- Installs a message queue between jupyter and basf2 processes to transfer status information (current event number, performance statistics etc.)
- Can support multiple basf2 Instances to concurrently scan a parameter space
- Implementation is generic and can be easily adapted to support other frameworks

Better User Experience: Widgets

- Jupyter Widgets are graphical exentsions to notebooks to better view use-case specific contents
- Written in Python and JavaScript, running interactively in the user's browser
- Allows to use rich library ecosystem of Python and web-development world (jQuery, HTML5, CSS etc.)

We developed a set of Jupyter widgets to improve the user experience of basf2 in Notebooks

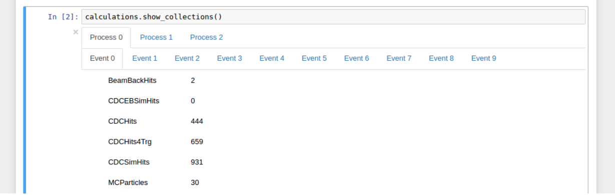
Progress Bar



```
Status: finished

85 % Remaining time: 2.04 s

30 % Remaining time: 26.92 s

In [ ]:
```

Better User Experience: Widgets

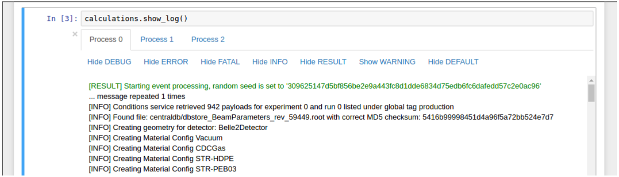
Collection Viewer



```
In [2]: calculations.show_collections()
```

BeamBackHits	2
CDCEBSimHits	0
CDCHits	444
CDCHits4Trg	659
CDCSimHits	931
MCParticles	30

Log Parser



```
In [3]: calculations.show_log()
```

```
[RESULT] Starting event processing, random seed is set to '309625147d5bf856be2e9a443fc8d1dde6834d75edb6fc6dafedd57c2e0ac96'
... message repeated 1 times
[INFO] Conditions service retrieved 942 payloads for experiment 0 and run 0 listed under global tag production
[INFO] Found file: centraldb/dbstore_BeamParameters_rev_59449.root with correct MD5 checksum: 5416b99998451d4a96f5a72bb524e7d7
[INFO] Creating geometry for detector: Belle2Detector
[INFO] Creating Material Config Vacuum
[INFO] Creating Material Config CDCGas
[INFO] Creating Material Config STR-HDPE
[INFO] Creating Material Config STR-PEB03
```

basf2 + Python + Jupyter + Hub = Epic Training

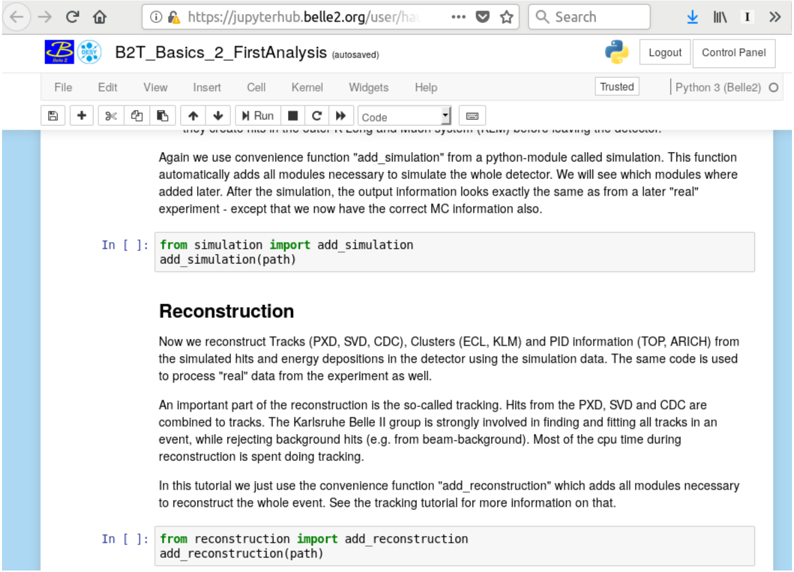
The Belle II collaboration hosts a Jupyterhub instance for all it's members:

`https://jupyterhub.belle2.org/`

Main purpose of this instance is for training workshops we hold at least 3 times per year.

- Each participant gets the correct software version with zero hassle (we use Jupyterhub's docker spawner and our externals and framework in a Docker image)
- The Notebook contain explanations, instructions and code fragments in one place: solves problem of disconnect between documentation and code.

Juypterhub Training



they create hits in the outer K-Long and Muon system (KLM) before leaving the detector.

Again we use convenience function "add_simulation" from a python-module called simulation. This function automatically adds all modules necessary to simulate the whole detector. We will see which modules where added later. After the simulation, the output information looks exactly the same as from a later "real" experiment - except that we now have the correct MC information also.

In [ ]:

```
from simulation import add_simulation
add_simulation(path)
```

Reconstruction

Now we reconstruct Tracks (PXD, SVD, CDC), Clusters (ECL, KLM) and PID information (TOP, ARICH) from the simulated hits and energy depositions in the detector using the simulation data. The same code is used to process "real" data from the experiment as well.

An important part of the reconstruction is the so-called tracking. Hits from the PXD, SVD and CDC are combined to tracks. The Karlsruhe Belle II group is strongly involved in finding and fitting all tracks in an event, while rejecting background hits (e.g. from beam-background). Most of the cpu time during reconstruction is spent doing tracking.

In this tutorial we just use the convenience function "add_reconstruction" which adds all modules necessary to reconstruct the whole event. See the tracking tutorial for more information on that.

In [ ]:

```
from reconstruction import add_reconstruction
add_reconstruction(path)
```

Juypterhub Training

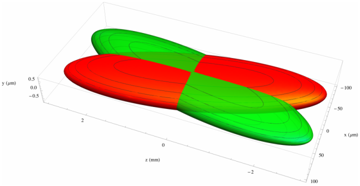
Over the last year, we have compiled Notebooks to a *Belle II Starterkit* which we can use during Training Workshops or people can use on their own time. Currently, the following topics are covered:

- Basic Python usage
- Writing basf2 steering files (event generation, simulation and reconstruction)
- Analysis of ROOT N-Tuples with pandas
- Plotting with pandas and matplotlib
- Usage of basf2's multi-variate tools for analysis
- Flavor Tagger and Continuum Suppression in user analysis

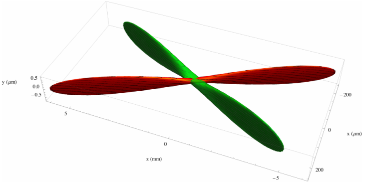
Very positive feedback from the two workshop we had so far from participants. Nota Bene: We also explain how to use basf2 without jupyter notebooks, some users prefer this.

Python and First Collision Results

Fast feedback to the SuperKEKB-Accelerator group is important during the early commissioning phase of the accelerator.
One novel feature to achieve the 40 times higher luminosity is the nano-beam scheme which allows for a very strong focusing at the beam interaction point up to 10μm (20 times smaller than KEKB)



KEKB Interaction region beam overlap



SuperKEKB Interaction region beam overlap

Source: Study of the collision point properties. N.Braun et al.[2]

Python and First Collision Results

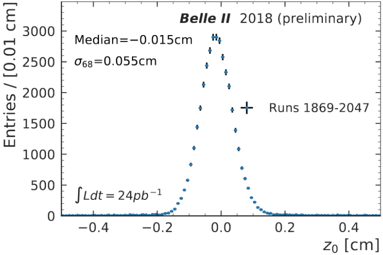


Figure: Longitudinal component of the interaction vertex estimated using single tracks originating from the interaction vertex in early Belle II events.[2]

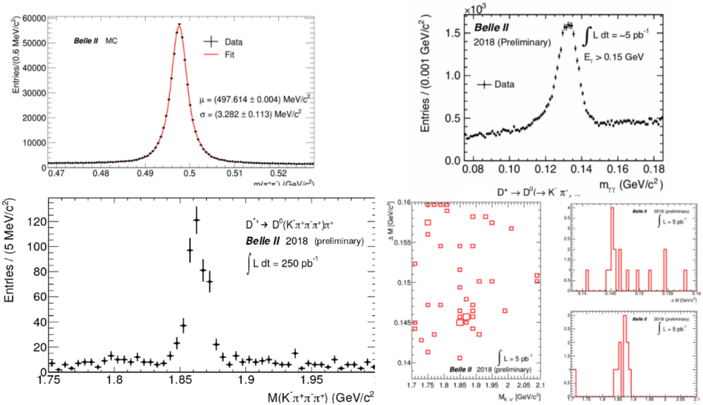
Workflow:

- Event-reconstruction with `basf2`
- Python-based analysis running as module in `basf2` writes ROOT NTuples
- Using the `root_pandas` library to load the NTuples as into one pandas dataframe
- Additional selections using the pandas Dataframe
- Final plotting with matplotlib and custom Belle II style configuration

Within days of first data, we were able to generate publication quality plots.

Python and First Collision Results

In the first weeks of data-taking, many more plots for publication have been created using either ROOT plotting, matplotlib and pandas



Underlines the flexibility and efficiency of user analyses options available in the the Belle II software ecosystem.

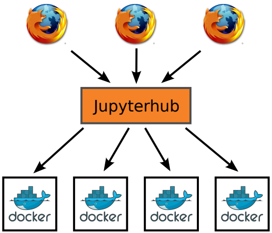
https://docs.belle2.org/search?cc=Belle+II+Notes+%3A+Plots

Wishlist for better Python @ HEP

- **Python 3 compatibility checked as part of the ROOT release procedure**
  Currently, we have to patch some things ourselves before building ROOT, but they are in the ROOT repository now.
- **Some ROOT-based libraries are difficult to use from python (for example RooFit) due to conceptual differences how python handles object lifetime**
  This results in weird side-effects which are hard to understand for the average user.
- **More common development effort on HEP-tools, specifically for Python**
  The `scikit-hep` project is a good start
  We also should work more to ”enrich” existing python data analysis tools like *scipy* with HEP-specific features (where applicable)
- **Compile documentation and training of *best-practices* for data analysis with existing python tools for HEP**

The Big Picture

A^{3}S



Your browser
is the UI

Central
webservice

Distributed
Computing

- Analysis in the cloud (analysis as a service)
- The notebooks can be used for outreach (e.g. tutorials in universities and schools)
- Jupyterhub provides a jupyter notebook server with authentication, user management, distributed computation/cluster support.
- Prototype and evaluation setup is running successfully

We are currently in discussion with data centers hosting large parts of our user base to provide a production-grade Jupyterhub with access to storage and batch farm.

Conclusion and Outlook

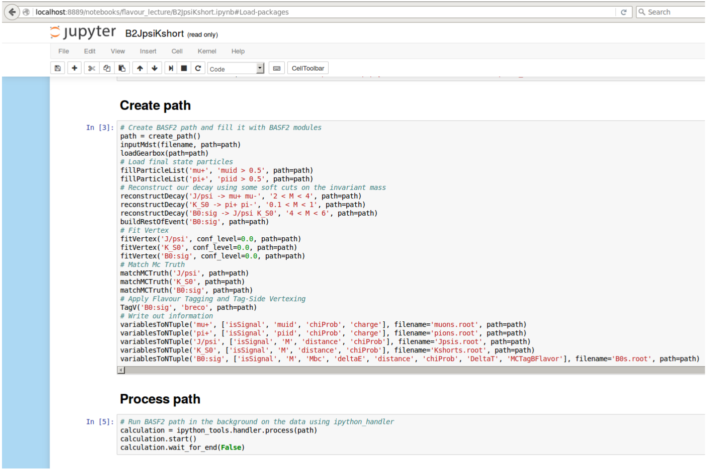
- Perform Python calculations with Jupyter notebooks to have all benefits of Python together with the interactivity.
- The lightweight software layer provided by `hep_ipython_tools` allows a **seamless integration of HEP frameworks** (here basf2) with interactive jupyter notebooks

  Notebooks can be used for:

  - Interactive development of framework module algorithms
  - Working on analyses with fast feedback via inlined plots
  - Self-describing Notebooks for tutorials and outreach
- Using jupyter(hub) with basf2 is a full environment for physics analysis!
- In the future: possibilities for interactive Belle 2 physics analysis via the web browser, centrally hosted at data centers

Backup Slides

Full Analysis Example



Create path

In [3]:

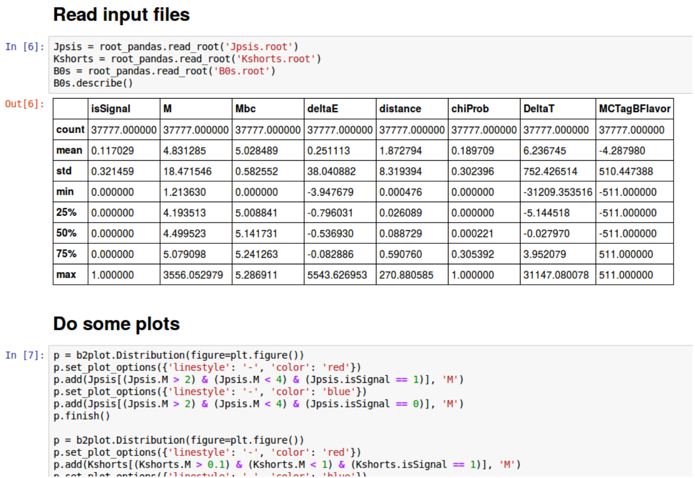
```
# Create BASF2 path and fill it with BASF2 modules
path = create_path()
inputMdst(filename, path=path)
loadGearbox(path=path)
# Load final state particles
fillParticleList('mu+', 'muid > 0.5', path=path)
fillParticleList('pi+', 'piid > 0.5', path=path)
# Reconstruct our decay using some soft cuts on the invariant mass
reconstructDecay('J/psi -> mu+ mu-', '2 < M < 4', path=path)
reconstructDecay('K_S0 -> pi+ pi-', '0.1 < M < 1', path=path)
reconstructDecay('B0:sig -> J/psi K_S0', '4 < M < 6', path=path)
buildRestOfEvent('B0:sig', path=path)
# Fit Vertex
fitVertex('J/psi', conf_level=0.0, path=path)
fitVertex('K_S0', conf_level=0.0, path=path)
fitVertex('B0:sig', conf_level=0.0, path=path)
# Match Mc Truth
matchMCTruth('J/psi', path=path)
matchMCTruth('K_S0', path=path)
matchMCTruth('B0:sig', path=path)
# Apply Flavour Tagging and Tag-Side Vertexing
TagV('B0:sig', 'breco', path=path)
# Write out information
variablesToNTuple('mu+', ['isSignal', 'muid', 'chiProb', 'charge'], filename='muons.root', path=path)
variablesToNTuple('pi+', ['isSignal', 'piid', 'chiProb', 'charge'], filename='pions.root', path=path)
variablesToNTuple('J/psi', ['isSignal', 'M', 'distance', 'chiProb'], filename='Jpsis.root', path=path)
variablesToNTuple('K_S0', ['isSignal', 'M', 'distance', 'chiProb'], filename='Kshorts.root', path=path)
variablesToNTuple('B0:sig', ['isSignal', 'M', 'Mbc', 'deltaE', 'distance', 'chiProb', 'DeltaT', 'MCTagBFlavor'], filename='B0s.root', path=path)
```

Process path

In [5]:

```
# Run BASF2 path in the background on the data using ipython_handler
calculation = ipython_tools.handler.process(path)
calculation.start()
calculation.wait_for_end(False)
```

Full Analysis Example



Read input files

```
In [6]: Jpsis = root_pandas.read_root('Jpsis.root')
        Kshorts = root_pandas.read_root('Kshorts.root')
        B0s = root_pandas.read_root('B0s.root')
        B0s.describe()
```

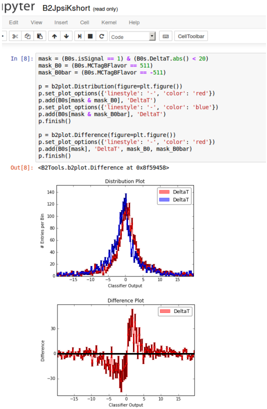
Out[6]:

	isSignal	M	Mbc	deltaE	distance	chiProb	DeltaT	MCTagBFlavor
count	37777.000000	37777.000000	37777.000000	37777.000000	37777.000000	37777.000000	37777.000000	37777.000000
mean	0.117029	4.831285	5.028489	0.251113	1.872794	0.189709	6.236745	-4.287980
std	0.321459	18.471546	0.582552	38.040882	8.319394	0.302396	752.426514	510.447388
min	0.000000	1.213630	0.000000	-3.947679	0.000476	0.000000	-31209.353516	-511.000000
25%	0.000000	4.193513	5.008841	-0.796031	0.026089	0.000000	-5.144518	-511.000000
50%	0.000000	4.499523	5.141731	-0.536930	0.088729	0.000221	-0.027970	-511.000000
75%	0.000000	5.079098	5.241263	-0.082886	0.590760	0.305392	3.952079	511.000000
max	1.000000	3556.052979	5.286911	5543.626953	270.880585	1.000000	31147.080078	511.000000

Do some plots

```
In [7]: p = b2plot.Distribution(figure=plt.figure())
        p.set_plot_options({'linestyle': '-', 'color': 'red'})
        p.add(Jpsis[(Jpsis.M > 2) & (Jpsis.M < 4) & (Jpsis.isSignal == 1)], 'M')
        p.set_plot_options({'linestyle': '-', 'color': 'blue'})
        p.add(Jpsis[(Jpsis.M > 2) & (Jpsis.M < 4) & (Jpsis.isSignal == 0)], 'M')
        p.finish()

        p = b2plot.Distribution(figure=plt.figure())
        p.set_plot_options({'linestyle': '-', 'color': 'red'})
        p.add(Kshorts[(Kshorts.M > 0.1) & (Kshorts.M < 1) & (Kshorts.isSignal == 1)], 'M')
        p.set_plot_options({'linestyle': '-', 'color': 'blue'})
```

Full Analysis Example



```
mask = (B0s.isSignal == 1) & (B0s.DeltaT.abs() < 20)
mask_B0 = (B0s.MCTagBFlavor == 511)
mask_B0bar = (B0s.MCTagBFlavor == -511)

p = b2plot.Distribution(figure=plt.figure())
p.set_plot_options({'linestyle': '-', 'color': 'red'})
p.add(B0s[mask & mask_B0], 'DeltaT')
p.set_plot_options({'linestyle': '-', 'color': 'blue'})
p.add(B0s[mask & mask_B0bar], 'DeltaT')
p.finish()

p = b2plot.Difference(figure=plt.figure())
p.set_plot_options({'linestyle': '-', 'color': 'red'})
p.add(B0s[mask], 'DeltaT', mask_B0, mask_B0bar)
p.finish()
```

```
<B2Tools.b2plot.Difference at 0x8f59458>
```

The shown notebook was already successfully tested with students in a tutorial for Belle II.

References I

“HEP IPython Tools.” `https://github.com/hep-ipython-tools/hep-ipython-tools` (28.9.2016).

N. Braun, A. Glazov, F. Metzer, and E. Paoloni, “Study of the collision point properties.,” May 2018.
Internal note describing selection is BELLE2-NOTE-PH-2018-006.