



# An Interactive and Comprehensive Working Environment for High-Energy Physics Software with Python and Jupyter Notebooks

CHEP 2016 – San Francisco, United States of America

Nils Braun<sup>1</sup>, [Thomas Hauth](#)<sup>1</sup>, Christian Pulvermacher<sup>2</sup>, Martin Ritter<sup>3</sup>

<sup>1</sup> Institute for Experimental Nuclear Physics (IEKP), KIT, Germany

<sup>2</sup> KEK, Tsukuba, Japan

<sup>3</sup> Ludwig-Maximilians-University, Munich, Germany



# SuperKEKB and the Belle II Experiment



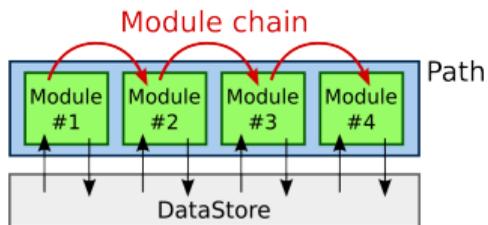
- KEKB was an electron–positron collider in Tsukuba/Japan which studied the decay of B mesons at the  $Y(4S)$  resonance
- Discovery of CP violation in the B system by the Belle experiment at KEKB and by the BaBar experiment at SLAC
  - Nobel Prize in Physics to Kobayashi and Maskawa in 2008
- The SuperKEKB collider and the Belle II detector will build on the previous success:
  - Study B meson system in far greater precision
  - Probe for new physics in a wide range of interesting topologies

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

# The basf2 Framework



- Event processing framework for Belle II written in C++11
- Modules use a common data store to read event data and write back results
- All processing steps for recorded and simulated events are implemented in basf2: Event generation, simulation, digitization, HLT, reconstruction and analysis
- Important libraries are bundled into *externals*: ROOT, gcc, Geant4, ...



Python is a first-class citizen in our framework:

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

# 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)

# Jupyter Notebooks - Example

A screenshot of a web browser displaying a Jupyter Notebook. The browser tab is titled 'first-compute' and the address bar shows 'localhost:8888/notebooks/first-compute'. The Jupyter interface includes a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', and 'Help', and a language selector set to 'Python 3'. Below the menu is a toolbar with icons for saving, adding, deleting, and running cells, along with a dropdown menu set to 'Code' and a 'Cell Toolbar' dropdown set to 'None'. The main content area shows a code cell with the following text:

```
In [1]: import math  
print(math.pi)
```

3.141592653589793

# 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



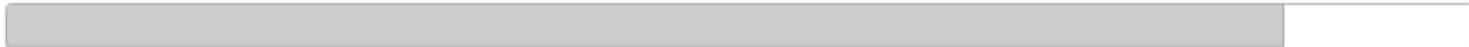
- Jupyter Widgets are graphical extensions to notebooks to better view use-case specific contents
- Written in Python and JavaScript, running interactively in the user's browser
- Allows to use rich library ecosystem of Python and web-development world (jQuery, HTML5, CSS etc.)

**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()`

× Process 0 Process 1 Process 2

Event 0	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Event 7	Event 8	Event 9
BeamBackHits	2								
CDCEBSimHits	0								
CDCHits	444								
CDCHits4Trg	659								
CDCSimHits	931								
MCParticles	30								

## Log Parser

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

× Process 0 Process 1 Process 2

Hide DEBUG Hide ERROR Hide FATAL Hide INFO Hide RESULT Show WARNING Hide DEFAULT

```
[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/tbstore_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
```

# Full Analysis Example



localhost:8889/notebooks/flavour\_lecture/B2JpsiKshort.ipynb#Load-packages

jupyter B2JpsiKshort (read only)

File Edit View Insert Cell Kernel Help

Code CellToolbar

## 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
variablesToTuple('mu+', ['isSignal', 'muid', 'chiProb', 'charge'], filename='muons.root', path=path)
variablesToTuple('pi+', ['isSignal', 'piid', 'chiProb', 'charge'], filename='pions.root', path=path)
variablesToTuple('J/psi', ['isSignal', 'M', 'distance', 'chiProb'], filename='Jpsis.root', path=path)
variablesToTuple('K S0', ['isSignal', 'M', 'distance', 'chiProb'], filename='Kshorts.root', path=path)
variablesToTuple('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()
```

# Full Analysis Example



ipyter B2JpsiKshort (read only)

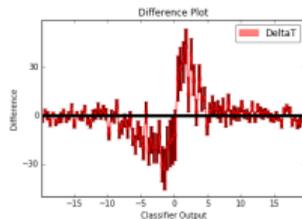
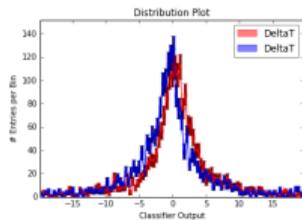
```
Edit View Insert Cell Kernel Help
Code CellToolbar

In [8]: 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()

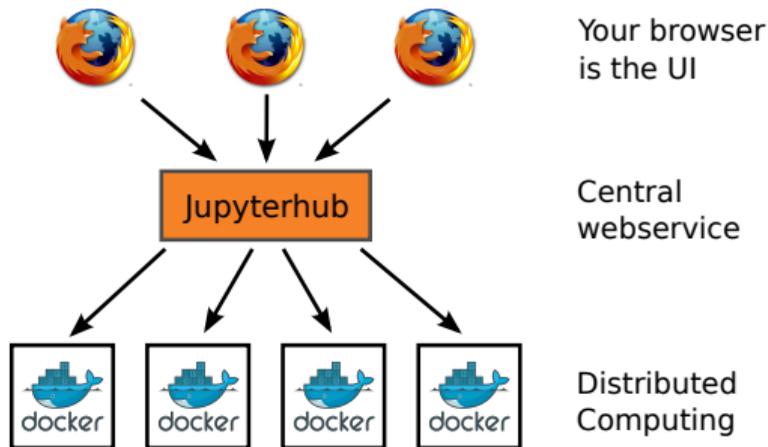
p = b2plot.Difference(figure=plt.figure())
p.set_plot_options({'linestyle': '--', 'color': 'red'})
p.add(B0s[mask], 'DeltaT', mask_B0, mask_B0bar)
p.finish()
```

Out [8]: <B2Tools.b2plot.Difference at 0x8f59458>



The shown notebook was already successfully tested with students in a tutorial at KIT.

# The Big Picture



- 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 on a machine at KIT
- In the future, we hope that data centers offer to host this kind of services

# 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 physics analysis via the web browser, centrally hosted at data centers

# References I



“HEP IPython Tools.”

<https://github.com/hep-ipython-tools/hep-ipython-tools> (28.9.2016).