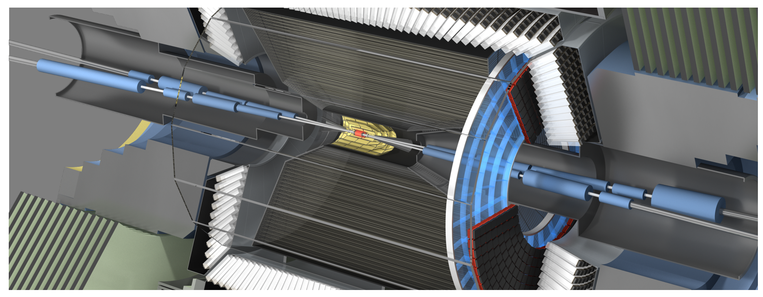
Jupyter Notebook Support of the Belle II Software

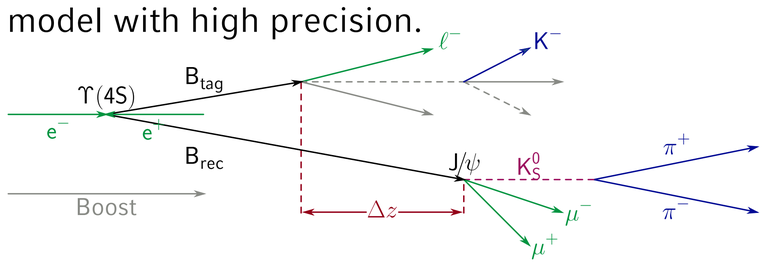


19th International Workshop on Advanced Computing and Analysis Techniques in Physics Research

M. Ritter[1], N. Braun[2], T. Hauth[2] and T. Kuhr[1] for the Belle II Collaboration
[1]LMU, Munich, Germany [2]KIT, Karlsruhe, Germany

The Belle II Experiment

An electron positron collider with asymmetric energies located in Japan to test the standard model with high precision.

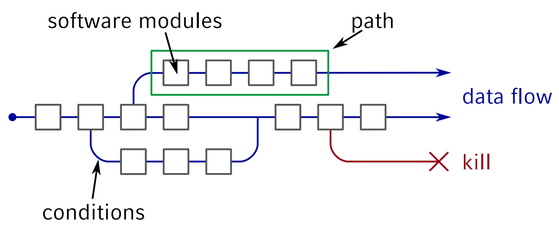


- started in 2018, collect 50 ab^{-1} until 2027
- plan to record 4×10^{11} events, 60 PB of data
- begin of data taking with full detector in about one week

Software Framework

Software framework written from scratch using experience from Belle and other HEP experiments.

- core framework implemented in C++17 and including the boost libraries
- use ROOT 6 framework for serialization of event data, Geant4 for simulation
- Python 3 interface for configuration and high level program steering
- different algorithms (called modules) are executed sequentially for each event



Analysis Concept

For analysts we provide high level constructs to work in a candidate based analysis scheme

- provide centralized and tested analysis tools
- grammar based final state reconstruction
- text based cuts on “variables”
- basically no C++ necessary for analysts

```
# create Ks -> pi+ pi- list from V0
# keep only candidates with 0.4 < M(pipi) < 0.6 GeV
fillParticleList('K_S0:pipi', '0.4 < M < 0.6')

# reconstruct J/psi -> mu+ mu- decay
# keep only candidates with 3.0 < M(mumu) < 3.2 GeV
reconstructDecay('J/psi:mumu -> mu+:loose mu-:loose',
                 '3.0 < M < 3.2')

# reconstruct B0 -> J/psi Ks decay
# keep only candidates with 5.2 < M(J/PsiKs) < 5.4 GeV
reconstructDecay('B0:jspiks -> J/psi:mumu K_S0:pipi',
                 '5.2 < M < 5.4')

# perform B0 kinematic vertex fit using only the mu+ mu-
# keep all candidates (C.L. of fit >=0)
vertexRave('B0:jspiks', 0.0, 'B0 -> [J/psi -> ^mu+ ^mu-] K_S0')

# build the rest of the event associated to the B0
buildRestOfEvent('B0:jspiks')

# perform MC matching (MC truth asociation). Always before TagV
matchMCTruth('B0:jspiks')

# calculate the Tag Vertex and Delta t (in ps)
# breco: type of MC association.
TagV('B0:jspiks', 'breco')

# save candidates to ntuple
variablesToNtuple("B0:jspiks", ["Mbc", "dE", "DeltaT"])
```

➡ Try to support the full stack of modern python tools

Jupyter Notebooks

- interactive python in a web browser
- mix code, documentation and visualization
- useful for data exploration
- great for teaching

Distributed with the Belle II Software

We provide Jupyter as part of the Belle II Software along with a large subset of the scientific Python ecosystem.

- integration of Jupyter tested as part of the release cycle
- possible to have Jupyter notebooks as unit tests in the software
- allows usage of Jupyter notebooks for all stages of the analysis: from processing data files to obtaining the final results and documentation of the analysis

Possible Problems with Jupyter

Jupyter notebooks work great with python code but most of our programs are compiled code with python interfaces

- Jupyter displays python output by replacing `stdout`/`stderr` objects. This doesn’t work with `std::cout`.
- event frameworks are not usually executed multiple times in one session
- event processing can take a long time blocking the notebook

By default event loop is executed in a supervised subprocess

```
In [*]: import basf2
path = basf2.Path()
path.add_module("EventInfoSetter", evtNumList=[10000])
path.add_module("EvtGenInput")
basf2.process(path)
```

```
Welcome to JupyROOT 6.14/06
```

35 % Remaining time: 12 seconds

Eventloop in Supervised Subprocess

While we allow for the event loop to executed multiple times we cannot guarantee that this works reliable in all cases. But repeated execution is a key feature of Jupyter. So by default we execute the event loop in a supervised subprocess when inside Jupyter.

- guaranteed side effect free execution
- allow asynchronous execution
- provide widgets to monitor asynchronous execution

Adaptive Belle II Logging System

The Belle II Software now detects when loaded inside a Jupyter notebook and will use `sys.stdout` for all outputs in this case

- correctly intersperse python and C++ output
- works for fatal errors which kill the kernel

Provide Rich Output for Objects

The Jupyter protocol allows rich output of objects is very simple to implement.

- extended custom objects like Modules and Path to be visualized easily

We can provide Rich Output for custom objects so that users can very easly inspect them

```
In [1]: import basf2
path = basf2.Path()
path.add_module("EventInfoSetter", evtNumList=[5000], expList=[0])
```

Out[1]: basf2.Module **EventInfoSetter**

Sets the event meta data information (exp, run, evt). You must use this module to tell basf2 about the number of events you want to generate, unless you have an input module that already does so. Note that all experiment/run combinations specified must be unique.

parameter	type	default	current	is default	is required
evtNumList	list(unsigned int)	[1]	[5000]	no	no
expList	list(int)	[0]	[0]	no	no
runList	list(int)	[0]	[0]	yes	no
skipNEvents	unsigned int	0	0	yes	no
skipToEvent	list(int)	[]	[]	yes	no

```
In [2]: path.add_module("PrintBeamParameters").set_name("A Module named Foo 😃")
path
```

Out[2]: basf2.Path object containing the following Modules

1. Module **EventInfoSetter**

   Sets the event meta data information (exp, run, evt). You must use this module to tell basf2 about the number of events you want to generate, unless you have an input module that already does so. Note that all experiment/run combinations specified must be unique.

   Changed Parameters:
   - `evtNumList = [5000]`
     List of the number of events which should be processed. Can be overridden via -n argument to basf2.

2. Module **A Module Named Foo** 😃 (type "PrintBeamParameters")

   Print the BeamParameters everytime they change

Use of Notebooks For Training

We have moved most of our introduction material for new users to Jupyter Notebooks to allow users to play around with all the examples in the tutorials.

- we provide training and documentation on how to open Jupyter notebooks remotely from analysis facilities
- centrally hosted JupyterHub notebook server to explore introduction material without the need to setup anything
- easy to verify that examples and training is not outdated

Now we reconstruct the decay channel $D^0 \to K^+\pi^-$ and again apply a cut on the created Particle objects directly during their creation. The charge-conjugated decay is always automatically reconstructed as well.

```
In [ ]: reconstructDecay('D0:Kpi -> K-:higheff pi+:higheff',
                          '1.7 < M < 1.9', path=path)
```

By now you should grasped the basic concept of all analysis convenience functions. The first parameter is always a decay-string consisting of a particle name and optionally its decay products. The second parameter is often a cut, which supports logical (and, or) and numeric (>, <, ==, !=, ...) operations, and features so-called variables. A variable is a small C++ function which can be identified by a name e.g. 'M' for the invariant mass. This C++ function takes a Particle object and returns a floating point number.

Hands-on
Take a look at the available variables by executing 'basf2 variables.py'. Can you find the description for all the variables we used so far?

Let's try something more advanced. We fit the decay vertex of each D meson candidate, and throw away all D candidates, which have a χ^2 fit probability below a certain threshold.

Everything Degrades Gracefully

After exploring the interface and data in Jupyter notebooks many users might want to switch to a more traditional script to submit to the grid. We took care to make this as painless as possible.

- users can easily convert notebooks to plain scripts
- in most cases the conversion is automatic and doesn’t require user intervention
- all data files produced in a notebook contain the script to reproduce them as metadata

Belle II
LMU TUM ORIGINS Excellence Cluster
Bundesministerium für Bildung und Forschung