Accelerated Starter Kit

2022 Belle II Physics Week

01.12.2022

R. Manfredi, S. Raiz, F. Tenchini,
S. Bilokin, M. Merola

Introduction

- This session is designed as an introduction to the basics of basf2 analysis.

- What we will cover:
  - What is basf2 and how to use it.
  - Signal particle reconstruction in the context of a Belle II analysis.
  - Grid submission and how to get your output back.
  - Workflow manager usage

- What we will not cover:
  - More advanced reconstruction tools e.g. Full Event Interpretation, Flavor Tagging, etc.
    - → Refer to the dedicated online book sections.
  - Offline analysis and how to make your advisor happy.

Foreword

- Due to time constraints, this starter kit is structured as lecture.

- Don’t be afraid to interrupt and ask questions.
  - There are no stupid questions.
  - Slides here are just seeds for conversation.

- You can follow along with the practical parts if you have a NAF or KEKCC account.

- You can also go over the material by yourself later by going through the online book, which covers everything we present here (and more).

Help! I'm stuck!

- Documentation: https://software.belle2.org
  - basf2 is fairly well documented, you can also find the **online book** there.
  - Glossary: https://confluence.desy.de/display/BI/Main+Glossary

- Ask somebody: https://questions.belle2.org
  - Forum for **any question**, not just software. Or send an email to an expert you know.

- Examples: `$BELLE2_RELEASE_DIR/<package>/examples`
  - For your specific package.

- The code: `$BELLE2_RELEASE_DIR`
  https://stash.desy.de/projects/B2/repos/basf2/browse/
  - If you really have to...

The basf2 code

- Belle II Analysis Software Framework (basf2)

- C++17 "under the hood"
    - Subdivided in **modules** to manipulate data.
    - Build particles, calculate physics quantities, apply cuts.
    - Handles the **heavy processing.**

- Python 3.8 for **steering**
    - Load and configure C++ modules.
    - Overall more user friendly and readable.
    - If you are not a developer you will do most of your work in Python.

Packages

- The software is organised into **packages**.
  - Tracking, simulation, various subdetectors...
  - During your time you might end up working on some of them.

- If you are doing a physics analysis, you mainly care about:

  analysis, mva, skim, b2bii

Convert Belle files to Belle II format.

Centrally run scripts to reduce data to usable size.

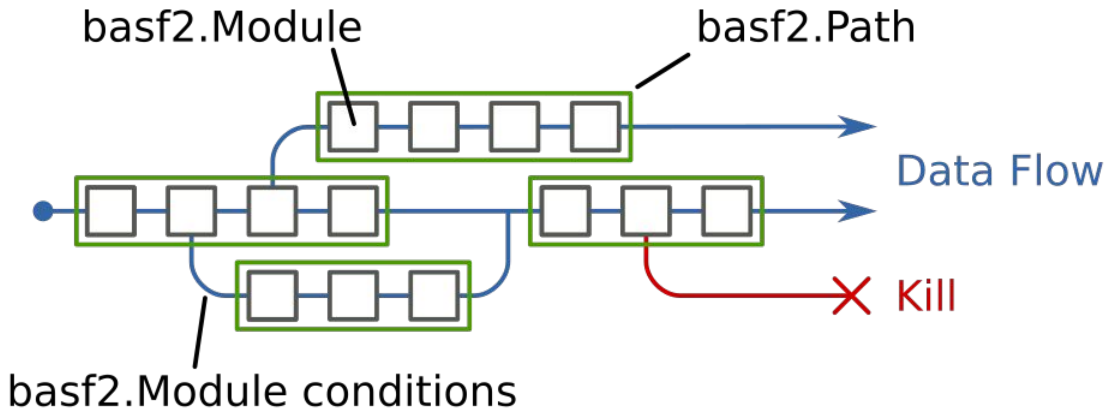
Multivariate analysis e.g. machine learning.

Steer most of the analysis steps and modules.
User interface for analysts.

Modules and Paths

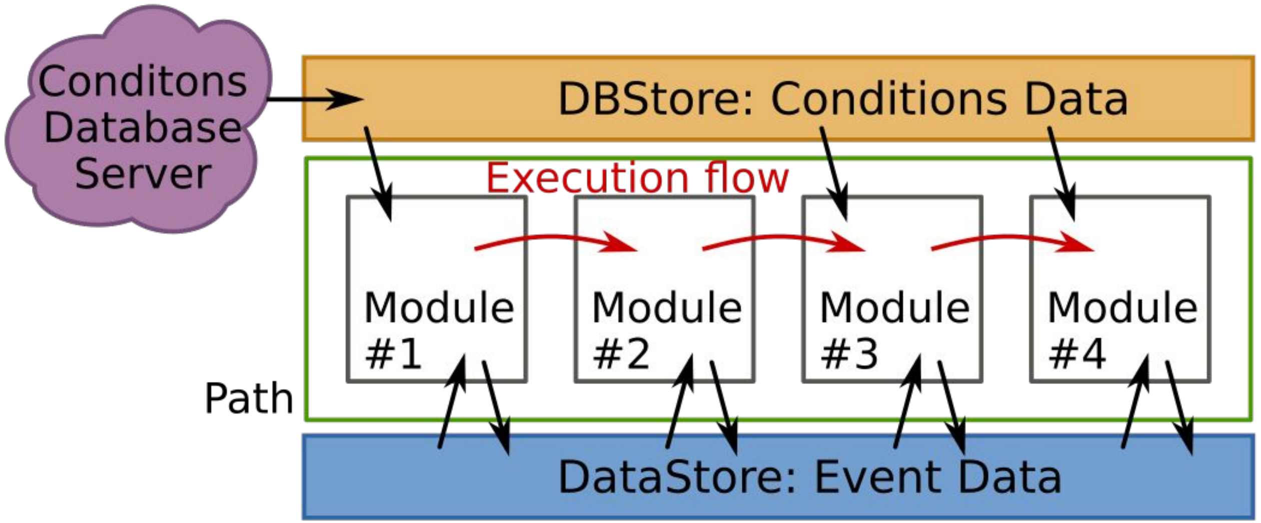
- A basf2 **module** is a piece of code that performs a specific task.

- A basf2 **path** is an ordered list of modules that will be used to process the data. Paths are built by chaining modules together.



Execution Flow

- **Event data** to be processed by modules is loaded into a common storage, the DataStore. These are the physics events (either real data or MC).
- **Non-event data** is loaded from the central conditions database to DBStore.
- When processing starts, the path(s) you defined are looped over the events.



Setting up basf2

YOU SHOULD NOT HAVE TO INSTALL ANYTHING!

Setting up basf2

YOU SHOULD NOT HAVE TO INSTALL ANYTHING!

Unless you are doing software development, a full local installation of basf2 is not necessary on KEK, NAF, or most other sites.

If you need to do limited code development, you can try an analysis setup.

(I will not cover it)

Setting up basf2

```
$ source /cvmfs/belle.cern.ch/tools/b2setup
Belle II software tools set up at: /cvmfs/belle.cern.ch/tools
```

Run this only once at the start of the session.

Setting up basf2

```
$ source /cvmfs/belle.cern.ch/tools/b2setup
Belle II software tools set up at: /cvmfs/belle.cern.ch/tools

$ b2setup --help  #check available releases
  ...
  The following releases are available:
  ...
    light-2210-devonrex
    prerelease-07-00-00b
    release-06-01-11
    prerelease-07-00-00c
```

Run this only once at the start of the session.

Multiple releases are available.

Setting up basf2

```
$ source /cvmfs/belle.cern.ch/tools/b2setup
Belle II software tools set up at: /cvmfs/belle.cern.ch/tools

$ b2setup --help  #check available releases
  ...
  The following releases are available:
  ...
    light-2210-devonrex
    prerelease-07-00-00b
    release-06-01-11
    prerelease-07-00-00c

$ b2help-releases --help
...
The recommended release is:
light-2210-devonrex
```

Run this only once at the start of the session

Multiple releases are available.

You can check the recommendation.
If you are new or unsure, use this.

Releases and backwards compatibility

- Typically we recommend the latest full release (**release-AA-BB-CC**) or light release (**light-YYMM-CODENAME**).

- A light release, such as **light-2210-devonrex** is made of a limited subset of packages (analysis, mdst, skim, b2bii) and is suitable for analysis.
  - Designed for faster version iteration, decoupled from other packages
  - No reconstruction or simulation available

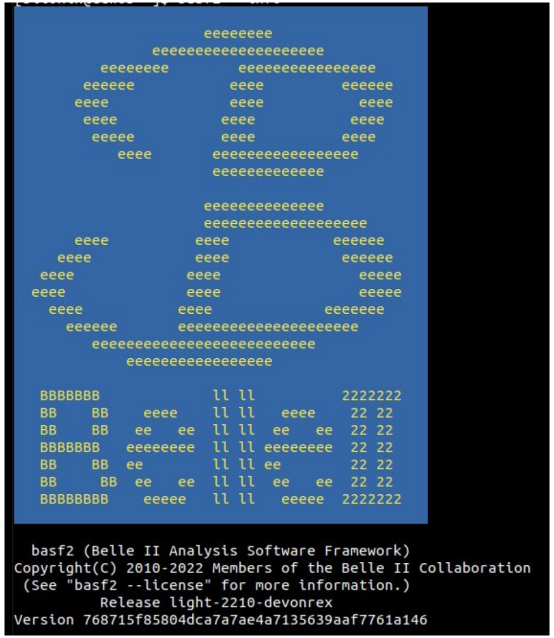
- Code is guaranteed to be backwards compatible across minor releases, E.g. from **release-AA-00-00** to **release-AA-01-00**, but not otherwise

Invoking basf2

```
$ source /cvmfs/belle.cern.ch/tools/b2setup

$ b2setup light-2210-devonrex
Environment setup for release: light-2210-devonrex
Central release directory   :
/cvmfs/belle.cern.ch/el7/releases/light-2210-devonrex

$ basf2 --info
```

Invoking basf2

```
$ source /cvmfs/belle.cern.ch/tools/b2setup

$ b2setup light-2210-devonrex
Environment setup for release: light-2210-devonrex
Central release directory   :
/cvmfs/belle.cern.ch/el7/releases/light-2210-devonrex

$ basf2 --info
```

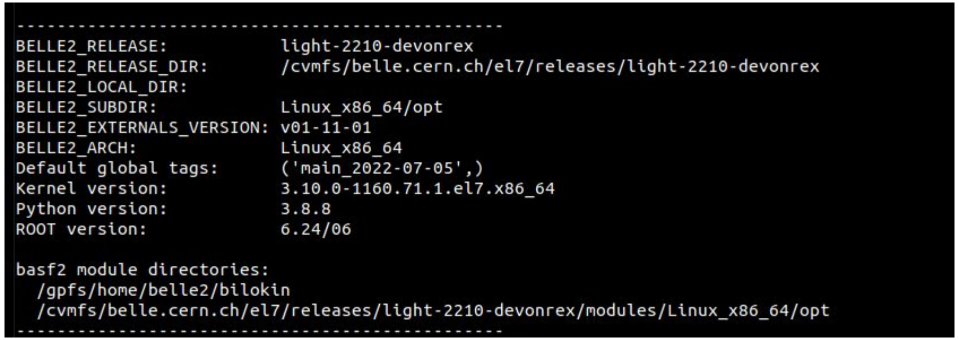


```
                      eeeeeeee
                eeeeeeeeeeeeeeeeeeee
          eeeeeeee        eeeeeeeeeeeeeeee
        eeeeee           eeee         eeeeee
       eeee              eeee           eeee
        eeee            eeee           eeee
         eeeee          eeee          eeee
            eeee       eeeeeeeeeeeeeeeee
                       eeeeeeeeeeeee

                      eeeeeeeeeeeeee
                      eeeeeeeeeeeeeeeeeee
       eeee          eeee            eeeeee
     eeee            eeee             eeeeee
   eeee             eeee                eeeee
  eeee              eeee                eeeee
    eeee           eeee             eeeeeee
      eeeeee       eeeeeeeeeeeeeeeeeeeee
         eeeeeeeeeeeeeeeeeeeeeeeeee
             eeeeeeeeeeeeeeeee

   BBBBBBB             ll ll          2222222
   BB    BB    eeee    ll ll   eeee    22 22
   BB    BB   ee   ee  ll ll  ee   ee  22 22
   BBBBBBB   eeeeeeee  ll ll eeeeeeee  22 22
   BB    BB  ee        ll ll ee        22 22
   BB     BB  ee   ee  ll ll  ee   ee  22 22
   BBBBBBBB    eeeee   ll ll   eeeee  2222222

  basf2 (Belle II Analysis Software Framework)
Copyright(C) 2010-2022 Members of the Belle II Collaboration
 (See "basf2 --license" for more information.)
          Release light-2210-devonrex
Version 768715f85804dca7a7ae4a7135639aaf7761a146
```

Invoking basf2

```
$ source /cvmfs/belle.cern.ch/tools/b2setup

$ b2setup light-2210-devonrex
Environment setup for release: light-2210-devonrex
Central release directory    :
/cvmfs/belle.cern.ch/el7/releases/light-2210-devonrex

$ basf2 --info
```

Includes some useful information



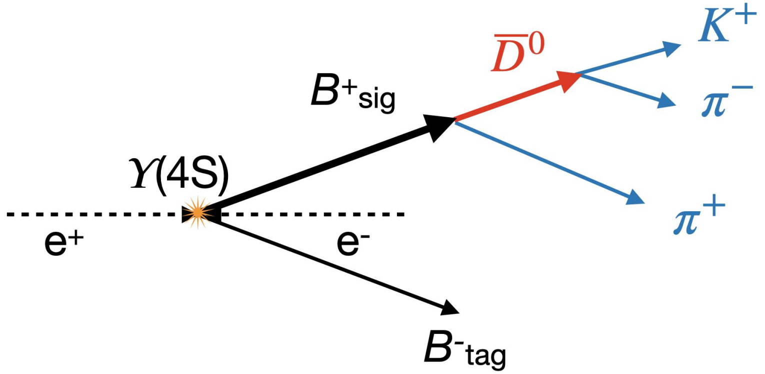
```
-----------------------------------------------
BELLE2_RELEASE:            light-2210-devonrex
BELLE2_RELEASE_DIR:        /cvmfs/belle.cern.ch/el7/releases/light-2210-devonrex
BELLE2_LOCAL_DIR:
BELLE2_SUBDIR:             Linux_x86_64/opt
BELLE2_EXTERNALS_VERSION:  v01-11-01
BELLE2_ARCH:               Linux_x86_64
Default global tags:       ('main_2022-07-05',)
Kernel version:            3.10.0-1160.71.1.el7.x86_64
Python version:            3.8.8
ROOT version:              6.24/06

basf2 module directories:
  /gpfs/home/belle2/bilokin
  /cvmfs/belle.cern.ch/el7/releases/light-2210-devonrex/modules/Linux_x86_64/opt
-----------------------------------------------
```

basf2: Practical Example

https://software.belle2.org/development/sphinx/online_book/basf2.html

Reconstruct a B decay

We will now reconstruct $\mathbf{B^+ \rightarrow \bar{D}^0 \pi^+}$, with $\mathbf{\bar{D}^0 \rightarrow K^+\pi^-}$.



You can find the scripts at:

```
KEKCC:/home/belle2/bilokin/public/valencia-tutorial
NAF:  /afs/desy.de/user/b/bilokin/public/valencia-tutorial
https://stash.desy.de/projects/B2A/repos/starterkit-physics-week/browse
```

Script structure

```
#!/usr/bin/env python3

import basf2 as b2
import modularAnalysis as ma

# Creates the main path
my_path = b2.create_path()

# Import the input file
ma.inputMdst(environmentType='default',
     filename='input.root', path=my_path)

# MAIN CODE GOES HERE

# Processes the path
b2.process(my_path)
```

Script structure

```
#!/usr/bin/env python3

import basf2 as b2
import modularAnalysis as ma

# Creates the main path
my_path = b2.create_path()

# Import the input file
ma.inputMdst(environmentType='default',
     filename='input.root', path=my_path)

# MAIN CODE GOES HERE

# Processes the path
b2.process(my_path)
```

Module containing wrapper functions for analysis modules.
Documentation: modularAnalysis

Script structure

```
#!/usr/bin/env python3

import basf2 as b2
import modularAnalysis as ma

# Creates the main path
my_path = b2.create_path()

# Import the input file
ma.inputMdst(environmentType='default',
     filename='input.root', path=my_path)

# MAIN CODE GOES HERE

# Processes the path
b2.process(my_path)
```

Initialize and process the path.
Your code goes in the middle.

Script structure

```
#!/usr/bin/env python3

import basf2 as b2
import modularAnalysis as ma

# Creates the main path
my_path = b2.create_path()

# Import the input file
ma.inputMdst(environmentType='default',
     filename='input.root', path=my_path)

# MAIN CODE GOES HERE

# Processes the path
b2.process(my_path)
```

Standard way to input mDSTs.
Documentation: inputMdst

Script structure

```
#!/usr/bin/env python3

import basf2 as b2
import modularAnalysis as ma

# Creates the main path
my_path = b2.create_path()

# Import the input file
ma.inputMdst(environmentType='default',
     filename='input.root', path=my_path)

# MAIN CODE GOES HERE

# Processes the path
b2.process(my_path)
```

File Types in Belle II analysis

DSTs are files containing post-reconstruction objects used to populate the DataStore. They are based on the ROOT file format.

- **DST: d**ata **s**ummary **t**able containing basf2 objects.
- **cDST: c**alibration DST, which includes necessary information for calibration and alignment.
- **mDST: m**ini **DST** with less information, optimised for file size.
  - Detector hits are dropped, contains only high-level objects, tracks and clusters.
  - mDST backward-compatibility is guaranteed for **the last two major releases**.
- **uDST: u**ser or micro (**μ**) DST, so-called skimmed format which contains only certain events and more analysis objects per event. ← Recommended for your analysis

Ntuples are the files that are produced via basf2 execution on _DST files. Typically, this is a flat table data, where columns are the variables, like momentum, energy, etc., and rows are particles or events.

- ROOT: the most popular format to go (yet). Contains TTree or TH1D, TH2D objects
- H5: not used as much (yet)

Define lists of stable particles

```
track_quality_cut = 'abs(dr) < 0.5 and abs(dz) < 3'
# Fill pion lists
ma.fillParticleList(decayString='pi+:myPions',
     cut=track_quality_cut, path=my_path)

# Equivalent of doing
# pload = register_module('ParticleLoader')
# pload.param('decayStringsWithCuts',
#    [('pi+:myPions', track_quality_cut)])
# mypath.add_module(pload)


sc.stdK('all', path=my_path)
ma.cutAndCopyList('K-:myKaons', 'K-:all',
     track_quality_cuts, path=my_path)
```

Define lists of Final State Particles

```
track_quality_cut = 'abs(dr) < 0.5 and abs(dz) < 3'
# Fill pion lists
ma.fillParticleList(decayString='pi+:myPions',
     cut=track_quality_cut, path=my_path)

# Equivalent of doing
# pload = register_module('ParticleLoader')
# pload.param('decayStringsWithCuts',
#    [('pi+:myPions', track_quality_cut)])
# mypath.add_module(pload)


sc.stdK('all', path=my_path)
ma.cutAndCopyList('K-:myKaons', 'K-:all',
     track_quality_cuts, path=my_path)
```

Creates lists of basic objects to reconstruct decays.
Documentation: fillParticleList
Documentation: Standard lists

Define lists of Final State Particles

```
track_quality_cut = 'abs(dr) < 0.5 and abs(dz) < 3'
# Fill pion lists
ma.fillParticleList(decayString='pi+:myPions',
     cut=track_quality_cut, path=my_path)

# Equivalent of doing
# pload = register_module('ParticleLoader')
# pload.param('decayStringsWithCuts',
#    [('pi+:myPions', track_quality_cut)])
# mypath.add_module(pload)


sc.stdK('all', path=my_path)
ma.cutAndCopyList('K-:myKaons', 'K-:all',
     track_quality_cuts, path=my_path)
```

These are convenience functions which are equivalent to several lines of code.

More legible and user friendly.

Define lists of Final State Particles

```
track_quality_cut = 'abs(dr) < 0.5 and abs(dz) < 3'
# Fill pion lists
ma.fillParticleList(decayString='pi+:myPions',
    cut=track_quality_cut, path=my_path)

# Equivalent of doing
# pload = register_module('ParticleLoader')
# pload.param('decayStringsWithCuts',
#    [('pi+:myPions', track_quality_cut)])
# mypath.add_module(pload)


sc.stdK('all', path=my_path)
ma.cutAndCopyList('K-:myKaons', 'K-:all',
    track_quality_cuts, path=my_path)
```

Creates lists of basic objects to reconstruct decays.
Documentation: fillParticleList
Documentation: Standard lists

Need to specify:
- particle name
- list label
- selection criteria

Which selection criteria to use? Consult the physics performance group, or your own physics working group.

Define lists of Final State Particles

```
track_quality_cut = 'abs(dr) < 0.5 and abs(dz) < 3'
# Fill pion lists
ma.fillParticleList(decayString='pi+:myPions',
    cut=track_quality_cut, path=my_path)

# Equivalent of doing
# pload = register_module('ParticleLoader')
# pload.param('decayStringsWithCuts',
#    [('pi+:myPions', track_quality_cut)])
# mypath.add_module(pload)


sc.stdK('all', path=my_path)
ma.cutAndCopyList('K-:myKaons', 'K-:all',
    track_quality_cuts, path=my_path)
```

Creates lists of basic objects to reconstruct decays.
Documentation: fillParticleList
Documentation: Standard lists

Need to specify:
- particle name
- list label
- selection criteria

No need to create charge conjugate, basf2 does it automatically.

Building charged particleLists

- Charged **f**inal **s**tate **p**articles (FSP) are built from reconstructed Track objects which have (among other things) a well defined momentum *p*.
- The mass *m* (and therefore energy E) is assigned based on your hypothesis.
- The label distinguishes lists of the same particles type, for your own benefit.
    - Maybe choose a useful name.

```
ma.fillParticleList(decayString='pi-:tomato'  ,cut='',path)
ma.fillParticleList(decayString='K+:potato'   ,cut='',path)
ma.fillParticleList(decayString='K-:breakfast',cut='abs(dz) < 3',path)
```

Building charged particleLists

- Charged **f**inal **s**tate **p**articles (FSP) are built from reconstructed Track objects which have (among other things) a well defined momentum *p*.
- The mass *m* (and therefore energy E) is assigned based on your hypothesis.
- The label distinguishes lists of the same particles type, for your own benefit.
  - Maybe choose a useful name.

```
ma.fillParticleList(decayString='pi-:tomato'   ,cut='',path)
ma.fillParticleList(decayString='K+:potato'    ,cut='',path)
ma.fillParticleList(decayString='K-:breakfast',cut='abs(dz) < 3',path)
```

- Lists 1 & 2 are built from the exact same tracks, but assuming different mass.
  - One need to use particle identification (PID) selection to select kaon or pion like particles

Building charged particleLists

- Charged **f**inal **s**tate **p**articles (FSP) are built from reconstructed Track objects which have (among other things) a well defined momentum *p*.
- The mass *m* (and therefore energy E) is assigned based on your hypothesis.
- The label distinguishes lists of the same particles type, for your own benefit.
  - Maybe choose a useful name.

```
ma.fillParticleList(decayString='pi-:tomato'  ,cut='',path)
ma.fillParticleList(decayString='K+:potato'   ,cut='',path)
ma.fillParticleList(decayString='K-:breakfast',cut='abs(dz) < 3',path)
```

- Lists 1 & 2 are built from the exact same tracks, but assuming different mass.
  - One need to use particle identification (PID) selection to select kaon or pion like particles
- The third list is a subset of the second.

Building charged particleLists

- Charged **f**inal **s**tate **p**articles (FSP) are built from reconstructed Track objects which have (among other things) a well defined momentum *p*.
- The mass *m* (and therefore energy E) is assigned based on your hypothesis.
- The label distinguishes lists of the same particles type, for your own benefit.
  - Maybe choose a useful name.

```
ma.fillParticleList(decayString='pi-:tomato'  ,cut='',path)
ma.fillParticleList(decayString='K+:potato'   ,cut='',path)
ma.fillParticleList(decayString='K-:breakfast',cut='abs(dz) < 3',path)
```

- Lists 1 & 2 are built from the exact same tracks, but assuming different mass.
  - One need to use particle identification (PID) selection to select kaon or pion like particles
- The third list is a subset of the second.
- Both positive and negative tracks are included, no matter the sign used.

Neutral lists

- Neutrals are also built with `fillParticleList()` or via `stdPhotons`, `stdKlongs`, `stdPi0s` modules

- Photons are generally built from ECLClusters.
- K_L are generally built from KLMClusters.
- Some composites, like K_S and Λ, are generally built from an object called V0, etc.

- I will not delve into the details; you can find many examples in the docs.

- **NEVER** do your own event loop over ECLClusters, etc.
  - Just use `fillParticleList()`

Pythonization of the script

- If we want to reuse the steering script it is better to reorganize it:

```
def get_Bmeson_path():
    # Create a path
    my_path = b2.create_path()
    # MAIN CODE HERE
    return my_path

# Condition to run only when the script is called directly:
if __name__ == "__main__":
    # Call our function:
    my_path = get_Bmeson_path()
    # Processes the path
    b2.process(my_path)
```

- This is useful for the script reusability

Reconstruct D0 and then B+ candidates

```
# Form D0 candidates
ma.reconstructDecay(decayString='D0:Kpi -> K+:myKaons pi-:myPions',
     cut='1.7 < M < 2.1', path=my_path)


# Form B+ candidates
ma.reconstructDecay(decayString='B+:D0pi -> D0:Kpi pi+:myPions',
     cut='5.2 < Mbc < 5.3 and abs(deltaE) < 0.3', path=my_path)
```

Reconstruct D0 and then B+ candidates

```
# Form D0 candidates
ma.reconstructDecay(decayString='D0:Kpi -> K+:myKaons pi-:myPions',
     cut='1.7 < M < 2.1', path=my_path)


# Form B+ candidates
ma.reconstructDecay(decayString='B+:D0pi -> D0:Kpi pi+:myPions',
     cut='5.2 < Mbc < 5.3 and abs(deltaE) < 0.3', path=my_path)
```

Forms and selects composite particle candidates.
Documentation: reconstructDecay

Reconstruct D0 and then B+ candidates

```
# Form D0 candidates
ma.reconstructDecay(decayString='D0:Kpi -> K-:myKaons pi+:myPions',
     cut='1.7 < M < 2.1', path=my_path)


# Form B+ candidates
ma.reconstructDecay(decayString='B+:D0pi -> anti-D0:Kpi pi+:myPions'
     cut='5.2 < Mbc < 5.3 and abs(deltaE) < 0.3', path=my_path)
```

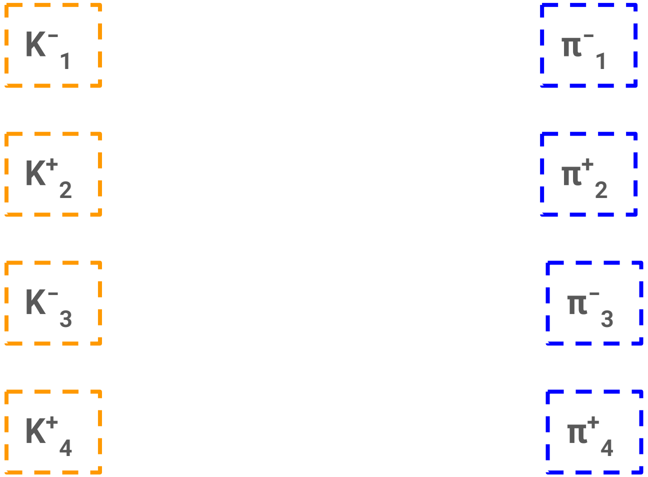
Forms and selects composite particle candidates.
Documentation: reconstructDecay

Needs decayString as input.
Documentation: decayString

ReconstructDecay internals

```
# Form D0 candidates
ma.reconstructDecay(decayString='D0:Kpi -> K-:all pi+:all',cut='', path=my_path)
```

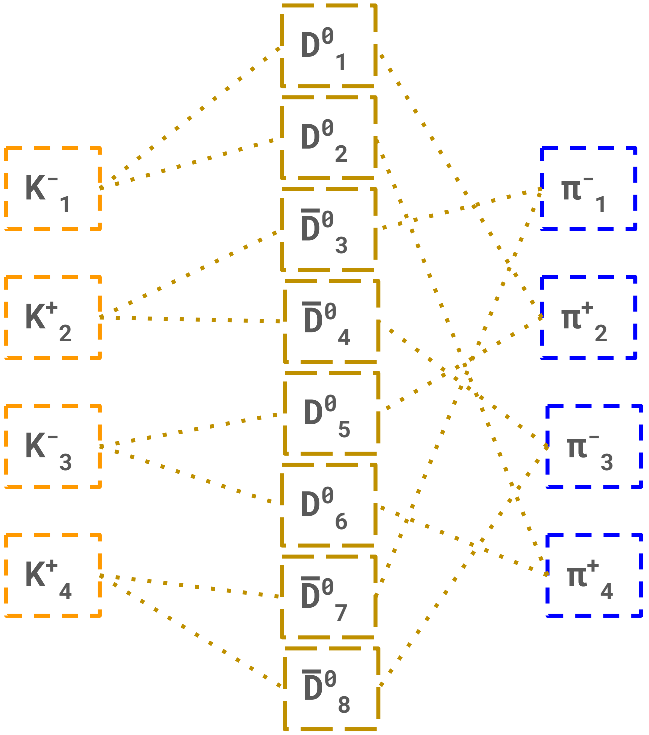
Imagine that there are no selection criteria applied



ReconstructDecay internals

```
# Form D0 candidates
ma.reconstructDecay(decayString='D0:Kpi -> K-:all pi+:all',cut='', path=my_path)
```

Imagine that there are no selection criteria applied



The `reconstructDecay` function will produce all possible combinations of daughter particles

This is an origin of the multiple candidates per event

Candidate-based analysis

- Take particle lists.
- Build up decay parents from daughters.
- Make candidates for your decay of interest.
- Filter/cut/keep/process.

- You might have more than one candidate per event.
- We deal with this after the fact. **This is fine.**
  - See for example https://arxiv.org/abs/1703.01128

We will now perform a vertex fit

- What does it mean?
  - Perform a minimisation to combine particle measurements, under the assumption that they originate from a common point (or a set of points).
  - **Inputs:** track helix, energy deposits, measurement covariances
  - **Outputs:** vertex position, improved 4-momentum of composites, covariance matrix

- Why?
  - Combinatorial background rejection.
  - Decay vertex position measurement → lifetime measurement.
  - Improved knowledge of kinematics.

- More tutorials here.

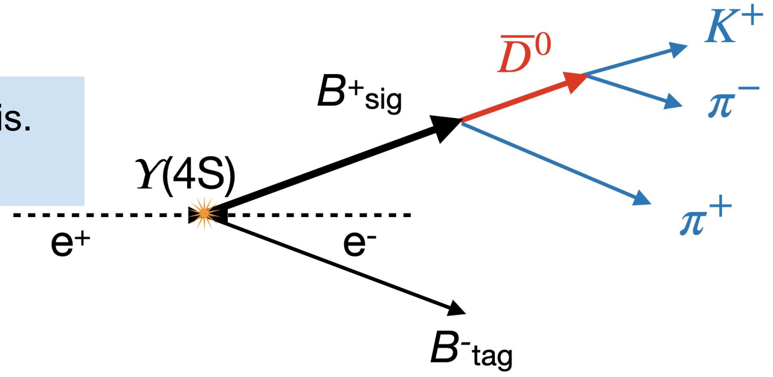
Vertex fit

```
import vertex as vx

# Vertex fit of the whole decay chain
vx.treeFit(list_name='B+:D0pi', conf_level=-1, path=my_path)
```

Vertex fit

```
import vertex as vx

# Vertex fit of the whole decay chain
vx.treeFit(list_name='B+:D0pi', conf_level=-1, path=my_path)
```

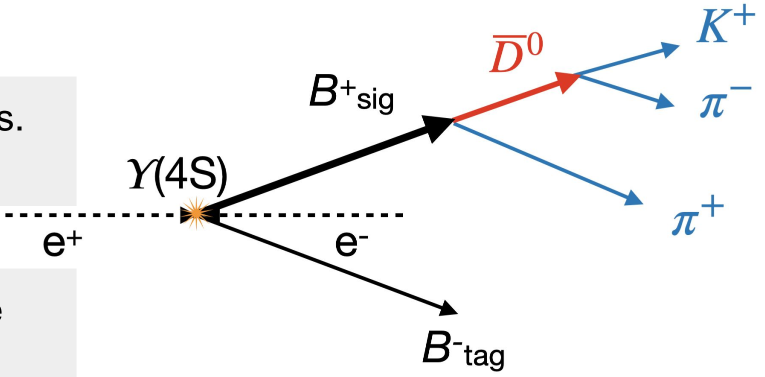
Fits the full decay chain according to our **B⁺** hypothesis.
Documentation: TreeFitter



Vertex fit

```
import vertex as vx

# Vertex fit of the whole decay chain
vx.treeFit(list_name='B+:D0pi', conf_level=-1, path=my_path)
```

Fits the full decay chain according to our **B⁺** hypothesis.
Documentation: TreeFitter



Vertex fit algorithms may change the kinematics of the daughter particles which might affect the high-level variables such as M_{bc} and ΔE

TreeFitter is also fine for fitting single vertices.
However, for specific cases, other fitters are available.

Match MC information

```
# Match reconstructed particles with MC particles
ma.matchMCTruth(list_name='B+:D0pi', path=my_path)
```

How do I know if my reconstruction is correct?
If we are working on simulation, I can compare my **candidate** with the MC event **truth.**
MCMatching tells me if a decay is correctly reconstructed, or why it is not.
Documentation: MCMatching

This line allows us to use `isSignal, mcPDG` and other MC-based variables

The flavor of daughter particles matters:
`decayString='B+:D0pi -> anti-D0:Kpi pi+:myPions'` and
`decayString='B+:D0pi -> D0:Kpi pi+:myPions'` will give different results!

Saving outputs: Variables

- Variables are either:
  - Physical quantities: invariant mass, beam-constrained mass, E, p, p_T , ...
  - Counters: event number, experiment number ...

- Every variable takes at least a `Particle*` as input and return a single number, such as double or integer, or a boolean.

- There are many variables available for analysis use.
  You can run **`b2help-variables`** to output a list, or check here.

Saving outputs: Writing to file

- Variables can be saved to ROOT trees:

```
ma.variablesToNtuple('B+:D0pi', ['Mbc'],
     filename='outputTree.root',treename='tree', path=my_path)
```

$$\mathrm{M_{bc}} = \sqrt{E_{\mathrm{beam}}^2 - \mathbf{p}_B^{*2}}$$

- … or directly to histogram:

```
ma.variablesToHistogram('B+:D0pi', ('Mbc', 60, 5.2, 5.3)],
     filename='outputHisto.root', path=my_path)
```

- Note: It is a <u>bad</u> idea to use different folder in the output path, given that every analysis script will be run on the Grid at some point

Variable Collections

- For practical reasons, variables are grouped in **collections.**

```
import variables.collections as vc

ma.variablesToNtuple('B+:D0pi', vc.deltae_mbc,
     filename='outputTree.root',treename='tree', path=my_path)
```

- The above is equivalent to:

```
ma.variablesToNtuple('B+:D0pi', ['Mbc',’deltaE’],
     filename='outputTree.root',treename='tree', path=my_path)
```

Aliases

- In addition to the head of the decay (in this case, the B+) you will probably want to save some quantities for the daughter particles.
- We do this with the `daughter(i,var)` meta-function.
- This can lead to unwieldy variable names, so it's useful to define aliases:

```
#Energy of the K+ in B+ -> [D0 -> K+ pi-] pi+
vm.addAlias("K_E", "daughter(0, daughter(0, E))")
```

- This is also useful for other meta-functions and long variable names to improve readability.

Aliases

- If you have many variables, this can quickly become cumbersome.
- Thankfully we can automate alias creation.

```
# Create aliases for variables

import variables.collections as vc
import variables.utils as vu

track_vars = vu.create_aliases_for_selected(
     list_of_variables = vc.kinematics,
     decay_string='B+ -> [D0 -> ^K+ ^pi-] ^pi+',
     prefix=['D_K', 'D_pi', 'B_pi'])
```

Aliases

- If you have many variables, this can quickly become cumbersome.
- Thankfully we can automate alias creation.

```
# Create aliases for variables

import variables.collections as vc
import variables.utils as vu

track_vars = vu.create_aliases_for_selected(
    list_of_variables = vc.kinematics,
    decay_string='B+ -> [D0 -> ^K+ ^pi-] ^pi+',
    prefix=['D_K', 'D_pi', 'B_pi'])
```

Contains functions to manage variable names and define aliases.
Documentation: variables.utils

Aliases

- If you have many variables, this can quickly become cumbersome.
- Thankfully we can automate alias creation.

```
# Create aliases for variables

import variables.collections as vc
import variables.utils as vu

track_vars = vu.create_aliases_for_selected(
     list_of_variables = vc.kinematics,
     decay_string='B+ -> [D0 -> ^K+ ^pi-] ^pi+',
     prefix=['D_K', 'D_pi', 'B_pi'])
```

Adds prefixes to all input variables, according to the particles selected in the decay string, avoiding ambiguities.

Optional

Aliases

- If you have many variables, this can quickly become cumbersome.
- Thankfully we can automate alias creation.

```
# Create aliases for variables

import variables.collections as vc
import variables.utils as vu

track_vars = vu.create_aliases_for_selected(
     list_of_variables = vc.kinematics,
     decay_string='B+ -> [D0 -> ^K+ ^pi-] ^pi+',
     prefix=['D_K', 'D_pi', 'B_pi'])
```

```
['px','py','pz','pt','p','E']
```

Adds prefixes to all input variables, according to the particles selected in the decay string, avoiding ambiguities.

Aliases

- If you have many variables, this can quickly become cumbersome.
- Thankfully we can automate alias creation.

```
# Create aliases for variables

import variables.collections as vc
import variables.utils as vu

track_vars = vu.create_aliases_for_selected(
     list_of_variables = vc.kinematics,
     decay_string='B+ -> [D0 -> ^K+ ^pi-] ^pi+',
     prefix=['D_K', 'D_pi', 'B_pi'])
```

```
['px','py','pz','pt','p','E']
```

Adds prefixes to all input variables, according to the particles selected in the decay string, avoiding ambiguities.

```
['D_K_px','D_K_py',...,'D_K_E','D_pi_px',...]
```

Saving outputs: Writing to file

- We can put it all together:

```
ma.variablesToNtuple('B+:D0pi',
     ['charge', 'nTracks'] + vc.deltae_mbc + track_vars,
     filename='outputTree.root',
     treename='tree',
     path=my_path)
```

- I trimmed this down for the sake of the lecture.
- Check out the example script for a slightly more extensive example:
    - `/afs/desy.de/user/b/bilokin/public/valencia-tutorial/steering_0X_final.py`

Run the script!

- First set up basf2:

```
$ source /cvmfs/belle.cern.ch/tools/b2setup
$ b2setup light-2210-devonrex
```

- Then run it on a test mdst file which you can find in the same folder:

```
basf2 /afs/desy.de/user/b/bilokin/public/valencia-tutorial/steering_0X_final.py
  -i
/afs/desy.de/user/b/bilokin/public/valencia-tutorial/mdst/charged_testmdst_r6.root
  -o output.root
```

- Feel free to take it, play around with it, and modify it to suit your analysis!

GBASF2

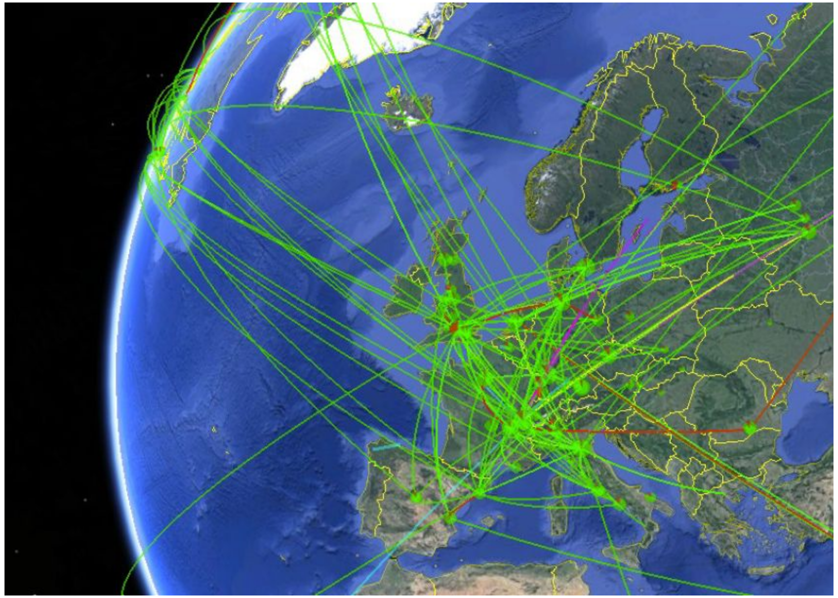
https://software.belle2.org/development/sphinx/online_book/computing/gbasf2.html

The Grid

- The Grid is a distributed computing system utilised by Belle II (and other particle physics experiments), to make use of the computing resources of the many institutions worldwide.

- Modern particle physics experiments collect and analyze tens of petabytes of data and MC.

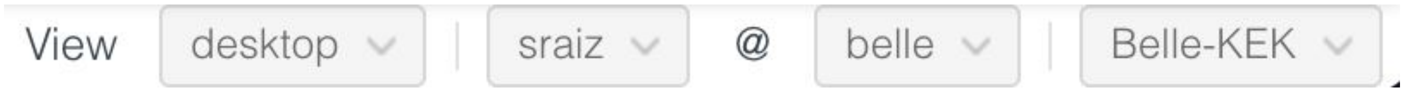
- Processing all of it at a single site is not feasible.



Gbasf2

- Gbasf2 is the command-line client for submitting grid-based basf2 jobs.
- The **same steering files** are used for both basf2 and gbasf2.
- The steps to gain access are described here: https://confluence.desy.de/display/BI/Computing+GettingStarted
    - Be aware they might take some time to complete.

- If everything is fine, travel to the Dirac webpage (https://dirac.cc.kek.jp:8443/) and you should see your username at the bottom right:



Setting up gbasf2

- You *used* to need a local installation. It can still be done; follow the steps here: https://confluence.desy.de/display/BI/Computing+GBasf2
- However, nowadays you can also use a pre-installed version. Simply run:

```
$ source /cvmfs/belle.kek.jp/grid/gbasf2/pro/tools/setup.sh
$ gb2_proxy_init -g belle
```

- Attention: at this moment the basf2 and gbasf2 environments are not compatible. This means gbasf2 will require a fresh ssh session.

Check gbasf2 release

- Run gb2_check_release to see what version gbasf2 you are running:

```
$ gb2_check_release
Your installation is up-to-date: v5r6
Available gbasf2 releases:
Production: v5r6
Newer than prod:
Older than prod: v5r4,v5r4p1,v5r4p2,v5r5,v5r5p2
Available basf2 releases:
release-06-01-10
release-06-01-09
release-06-00-03


...
```

This will also tell you which basf2 releases are enabled for grid use.

Locating files on the grid

- Files (both MC and data) are stored around the world. A file catalog keeps the record of where the files are located.
- To access them, we use a logical file name (**LFN**) or logical path name (**LPN**) which is a unique identifier taking the form of a unix-like path:

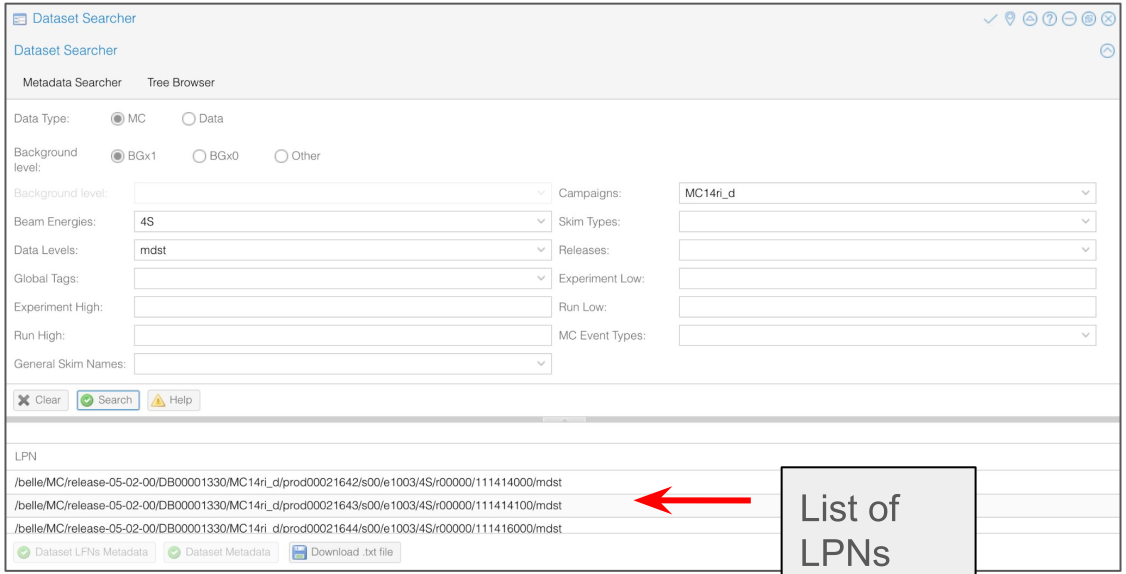
```
/belle/data_type/some_more_directories/dataset/datablock/file
```

- By design, each datablock contains a maximum of 1000 files.
    - If a dataset contains more than 1000 files, it will be subdivided into several blocks.

- You can list the content of a directory on the grid using `gb2_ds_list`
    - This is similar to using `ls` on your local system

The Dataset Searcher

This application is how we find datasets on the grid.



Dataset Searcher

Dataset Searcher

Metadata Searcher | Tree Browser

Data Type: [✓ MC] Data

Background level: [✓ BGx1] BGx0 Other

Background level:		Campaigns:	MC14ri_d
Beam Energies:	4S	Skim Types:	
Data Levels:	mdst	Releases:	
Global Tags:		Experiment Low:	
Experiment High:		Run Low:	
Run High:		MC Event Types:	
General Skim Names:			

Clear | Search | Help

LPN
/belle/MC/release-05-02-00/DB00001330/MC14ri_d/prod00021642/s00/e1003/4S/r00000/111414000/mdst
/belle/MC/release-05-02-00/DB00001330/MC14ri_d/prod00021643/s00/e1003/4S/r00000/111414100/mdst
/belle/MC/release-05-02-00/DB00001330/MC14ri_d/prod00021644/s00/e1003/4S/r00000/111416000/mdst

Dataset LFNs Metadata | Dataset Metadata | Download .txt file

List of LPNs

- Console version: `gb2_ds_search dataset ...`

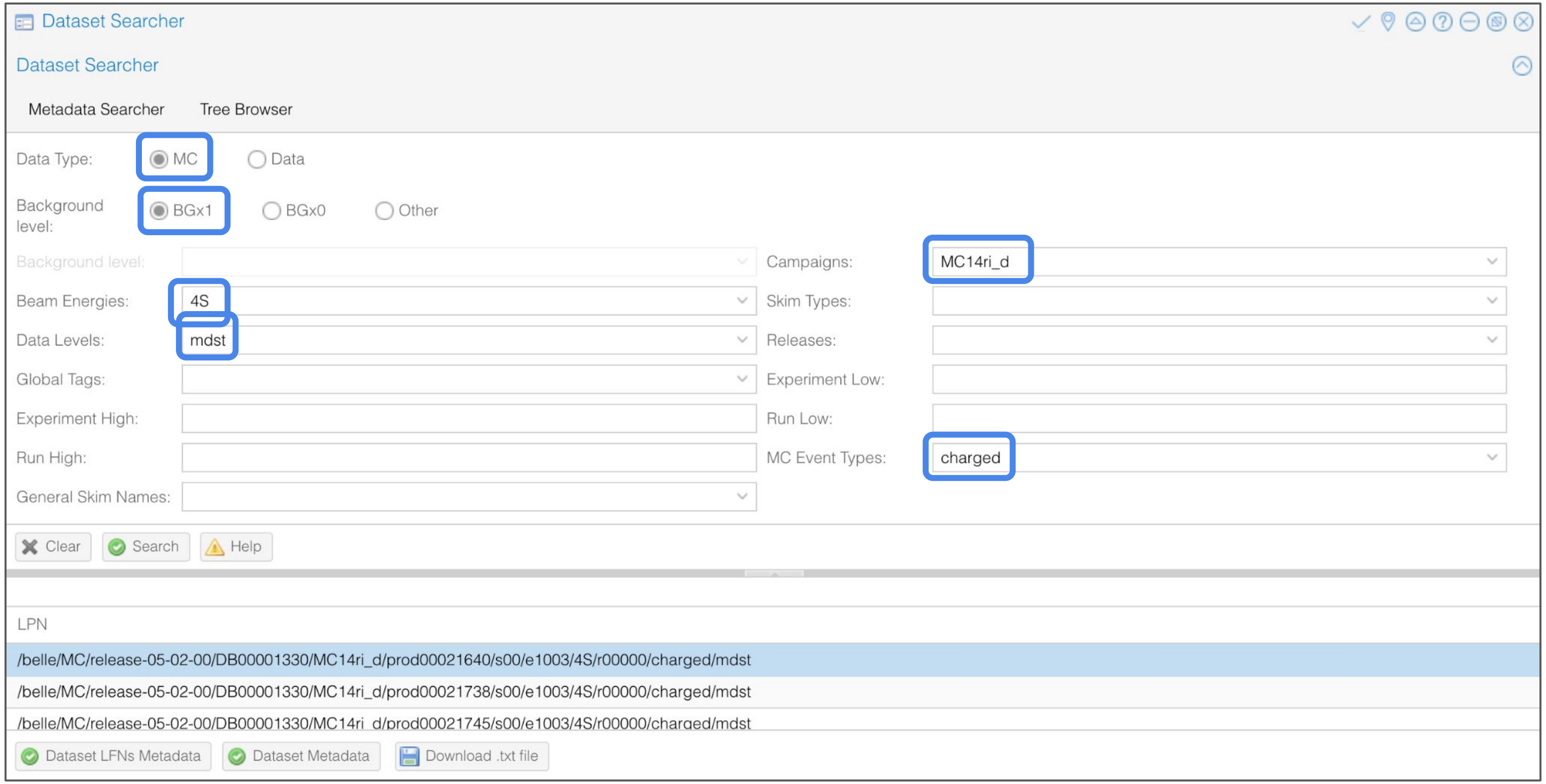
Try it yourself!

- Go to the DIRAC webportal and open (at the bottom left)
  → Menu
  → BelleDIRACApps
  → Dataset Searcher.

- … and play around with the fields.

- For example if I want to find the mdst files produced for MC type “charged” during the MC14ri_d campaign, with beam background (BGx1), I would do...

Try it yourself!



Dataset Searcher

Dataset Searcher

Metadata Searcher | Tree Browser

Data Type:	[✓ MC] Data		
Background level:	[✓ BGx1] BGx0 Other		
Background level:		Campaigns:	MC14ri_d
Beam Energies:	4S	Skim Types:	
Data Levels:	mdst	Releases:	
Global Tags:		Experiment Low:	
Experiment High:		Run Low:	
Run High:		MC Event Types:	charged
General Skim Names:			

Clear | Search | Help

LPN

/belle/MC/release-05-02-00/DB00001330/MC14ri_d/prod00021640/s00/e1003/4S/r00000/charged/mdst
/belle/MC/release-05-02-00/DB00001330/MC14ri_d/prod00021738/s00/e1003/4S/r00000/charged/mdst
/belle/MC/release-05-02-00/DB00001330/MC14ri_d/prod00021745/s00/e1003/4S/r00000/charged/mdst

Dataset LFNs Metadata | Dataset Metadata | Download .txt file

Submitting your first jobs

- The basic usage is

```
$ gbasf2 <your_steering_file.py> -p <project_name>
-s <available_basf2_release> -i <LPN of the input file>
```

Submitting your first jobs

This needs to be unique and cannot be reused. Figure out a good naming scheme for your jobs.

- The basic usage is

```
$ gbasf2 <your_steering_file.py> -p <project_name>
-s <available_basf2_release> -i <LPN of the input file>
```

- **You should test your steering file locally before submitting it!**

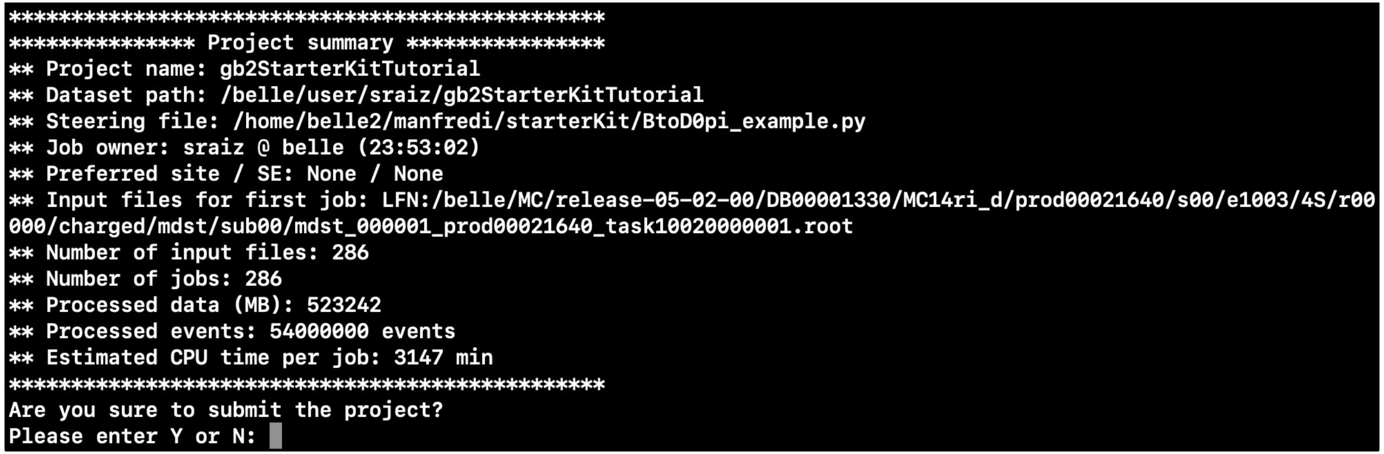
- In our case, if we wanted to run our steering file on the LPN we just found:

```
$ gbasf2 /home/belle2/manfredi/starterKit/BtoD0pi_example.py \
      -p gb2StarterKitTutorial -s light-2110-tartarus \
      -i /belle/MC/release-05-02-00/.../charged/mdst/sub00
```

Submitting your first jobs

- In our case, if we wanted to run our steering file on the LPN we just found:

```
gbasf2 /afs/desy.de/user/b/bilokin/public/valencia-tutorial/steering_0X_final.py \
       -p gb2StarterKitTutorial -s light-2210-devonrex \
       -i /belle/MC/release-06-00-08/.../charged/mdst/sub00
```



```
************************************************
*************** Project summary ****************
** Project name: gb2StarterKitTutorial
** Dataset path: /belle/user/sraiz/gb2StarterKitTutorial
** Steering file: /home/belle2/manfredi/starterKit/BtoD0pi_example.py
** Job owner: sraiz @ belle (23:53:02)
** Preferred site / SE: None / None
** Input files for first job: LFN:/belle/MC/release-05-02-00/DB00001330/MC14ri_d/prod00021640/s00/e1003/4S/r00
000/charged/mdst/sub00/mdst_000001_prod00021640_task10020000001.root
** Number of input files: 286
** Number of jobs: 286
** Processed data (MB): 523242
** Processed events: 54000000 events
** Estimated CPU time per job: 3147 min
************************************************
Are you sure to submit the project?
Please enter Y or N:
```

Submitting your first jobs

- You can also provide a Belle II collection

```
gbasf2 -p gb2StarterKitTutorial -s light-2210-devonrex \
/afs/desy.de/user/b/bilokin/public/valencia-tutorial/steering_0X_final.py -i
/belle/collection/Data/proc13_had_4S_v3
```

This will run on the proc13 hadronic skim.

- For now, to run on the full Moriond2023 dataset you need to run on two separate collections: “proc13 Moriond2023” and “prompt Moriond2023”.

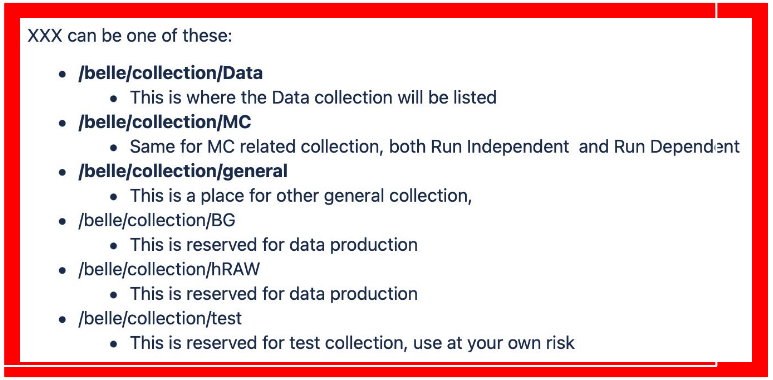
Belle II Collections

- You can check the list of all available collections using

```
$ gb2_ds_search collection --list_all_collections /belle/collection/XXX/*
```

or at https://confluence.desy.de/pages/viewpage.action?spaceKey=BI&title=Collection+summary

Recommended


XXX can be one of these:

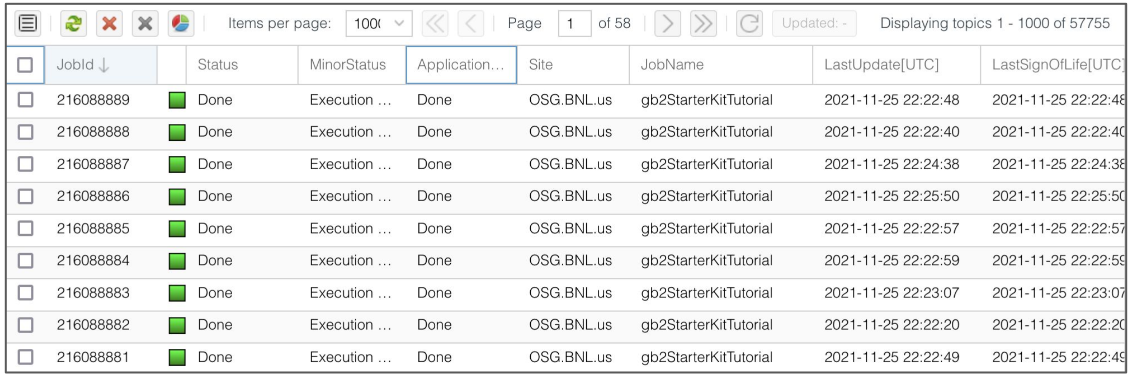
- **/belle/collection/Data**
  - This is where the Data collection will be listed
- **/belle/collection/MC**
  - Same for MC related collection, both Run Independent and Run Dependent
- **/belle/collection/general**
  - This is a place for other general collection,
- /belle/collection/BG
  - This is reserved for data production
- /belle/collection/hRAW
  - This is reserved for data production
- /belle/collection/test
  - This is reserved for test collection, use at your own risk

Monitoring jobs

- There are two ways to monitor your jobs on the grid.
- By command line:

```
gb2_project_summary -p gb2StarterKitTutorial
       Project          Owner   Status    Done    Fail    Run    Wait    Submission Time(UTC)    Duration
===========================================================================================================
gb2StarterKitTutorial    sraiz   Running   285     0       1      0       2021-11-25 21:55:40     00:25:08
```

- Or on the web portal:
  → Menu
   → Applications
    → Job Monitor



Items per page: 100(Page 1 of 58 Updated: - Displaying topics 1 - 1000 of 57755

	JobId		Status	MinorStatus	Application...	Site	JobName	LastUpdate[UTC]	LastSignOfLife[UTC
☐	216088889	🟩	Done	Execution ...	Done	OSG.BNL.us	gb2StarterKitTutorial	2021-11-25 22:22:48	2021-11-25 22:22:48
☐	216088888	🟩	Done	Execution ...	Done	OSG.BNL.us	gb2StarterKitTutorial	2021-11-25 22:22:40	2021-11-25 22:22:40
☐	216088887	🟩	Done	Execution ...	Done	OSG.BNL.us	gb2StarterKitTutorial	2021-11-25 22:24:38	2021-11-25 22:24:38
☐	216088886	🟩	Done	Execution ...	Done	OSG.BNL.us	gb2StarterKitTutorial	2021-11-25 22:25:50	2021-11-25 22:25:50
☐	216088885	🟩	Done	Execution ...	Done	OSG.BNL.us	gb2StarterKitTutorial	2021-11-25 22:22:57	2021-11-25 22:22:57
☐	216088884	🟩	Done	Execution ...	Done	OSG.BNL.us	gb2StarterKitTutorial	2021-11-25 22:22:59	2021-11-25 22:22:59
☐	216088883	🟩	Done	Execution ...	Done	OSG.BNL.us	gb2StarterKitTutorial	2021-11-25 22:23:07	2021-11-25 22:23:07
☐	216088882	🟩	Done	Execution ...	Done	OSG.BNL.us	gb2StarterKitTutorial	2021-11-25 22:22:20	2021-11-25 22:22:20
☐	216088881	🟩	Done	Execution ...	Done	OSG.BNL.us	gb2StarterKitTutorial	2021-11-25 22:22:49	2021-11-25 22:22:49

Downloading the output

- If your jobs finished successfully (status ‘Done’), you can download the output.

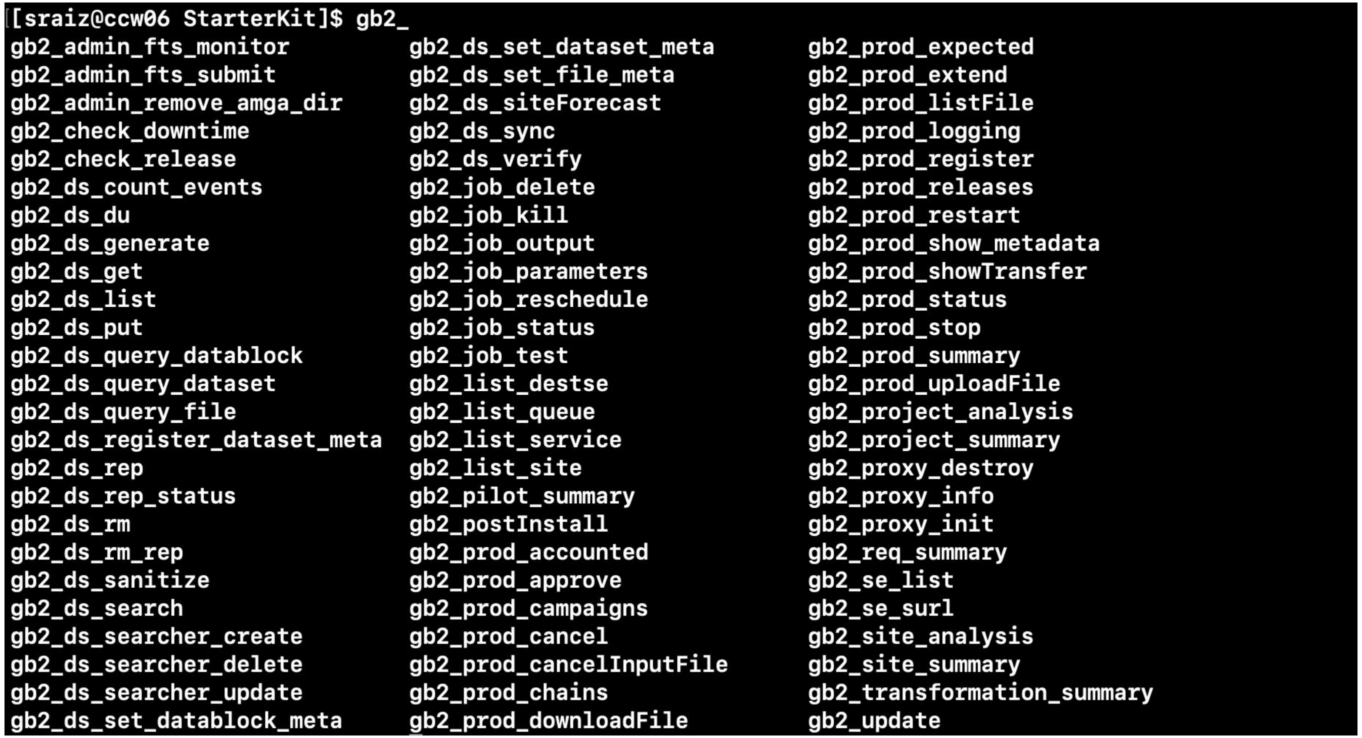
- This is located below your user space:
  `/belle/user/<username>/<project_name>`
- You can check the output using `gb2_ds_list <project_name>` :

```
$ gb2_ds_list gb2StarterKitTutorial/sub00
/belle/user/sraiz/gb2StarterKitTutorial/sub00/ntuple_00000_job216088604_00.root
/belle/user/sraiz/gb2StarterKitTutorial/sub00/ntuple_00001_job216088605_00.root
...
```

- … and download them with `gb2_ds_get <project_name>`

Other useful commands

All the gbasf2 commands start with `gb2_`: you can use tab completion to see a list.



```
[sraiz@ccw06 StarterKit]$ gb2_
gb2_admin_fts_monitor         gb2_ds_set_dataset_meta       gb2_prod_expected
gb2_admin_fts_submit          gb2_ds_set_file_meta          gb2_prod_extend
gb2_admin_remove_amga_dir     gb2_ds_siteForecast           gb2_prod_listFile
gb2_check_downtime            gb2_ds_sync                   gb2_prod_logging
gb2_check_release             gb2_ds_verify                 gb2_prod_register
gb2_ds_count_events           gb2_job_delete                gb2_prod_releases
gb2_ds_du                     gb2_job_kill                  gb2_prod_restart
gb2_ds_generate               gb2_job_output                gb2_prod_show_metadata
gb2_ds_get                    gb2_job_parameters            gb2_prod_showTransfer
gb2_ds_list                   gb2_job_reschedule            gb2_prod_status
gb2_ds_put                    gb2_job_status                gb2_prod_stop
gb2_ds_query_datablock        gb2_job_test                  gb2_prod_summary
gb2_ds_query_dataset          gb2_list_destse               gb2_prod_uploadFile
gb2_ds_query_file             gb2_list_queue                gb2_project_analysis
gb2_ds_register_dataset_meta  gb2_list_service              gb2_project_summary
gb2_ds_rep                    gb2_list_site                 gb2_proxy_destroy
gb2_ds_rep_status             gb2_pilot_summary             gb2_proxy_info
gb2_ds_rm                     gb2_postInstall               gb2_proxy_init
gb2_ds_rm_rep                 gb2_prod_accounted            gb2_req_summary
gb2_ds_sanitize               gb2_prod_approve              gb2_se_list
gb2_ds_search                 gb2_prod_campaigns            gb2_se_surl
gb2_ds_searcher_create        gb2_prod_cancel               gb2_site_analysis
gb2_ds_searcher_delete        gb2_prod_cancelInputFile      gb2_site_summary
gb2_ds_searcher_update        gb2_prod_chains               gb2_transformation_summary
gb2_ds_set_datablock_meta     gb2_prod_downloadFile         gb2_update
```

Troubleshooting

- Things do not always go well. One or several of your jobs might fail.
  - If it’s only a few jobs, you can reschedule them and attempt again.
  - If it’s all of them, it’s probably an issue you need to debug.
- The online book provides info on what to do in these situations.

- You can also find more information in the documentation:
  - https://confluence.desy.de/display/BI/Computing+GBasf2
  - https://gbasf2.belle2.org/

- If that doesn’t work, you can send an email to the user forum:
  - Subscribe at https://lists.belle2.org/sympa/info/comp-users-forum
  - (If you are using gbasf2 you should be subscribed anyway.)

Fast introduction to workflow managers

Introduction to workflow managers

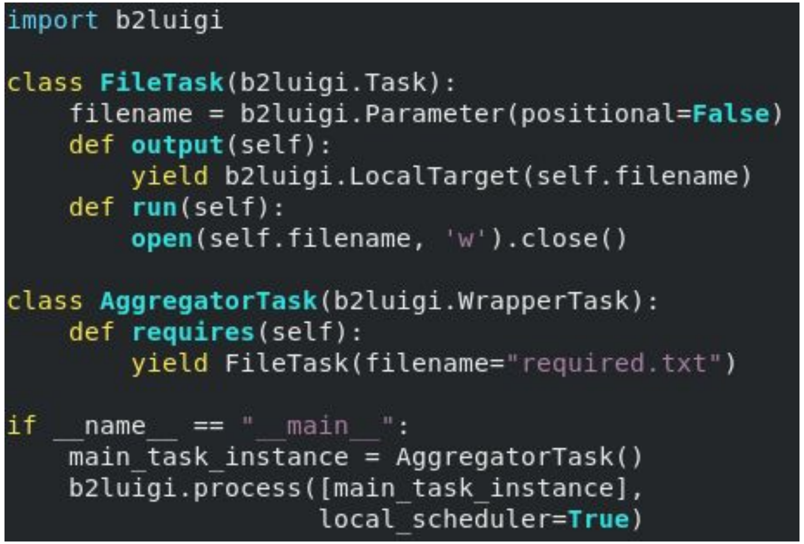
- Our physics analyses typically have several stages:
  - Running gbasf2 processing to produce ntuples
  - Merging the resulting ntuples
    - Train / test / validation split
  - Reweighting
  - Training of ML algorithms
  - Fitting
  - Performing systematic studies
- These stages has to be repeated for primary and control channel, data and MC, etc., which makes it even more complicated
- A collection of bash scripts is not easily configurable and cannot be parallelized, outputs no reports,
  no error handling, etc.
- Implementation of these features in bash scripts is just reinventing workflow management systems

Available solutions

- snakemake: https://snakemake.readthedocs.io/
  - Binds separate analysis scripts using a compact python configuration file
  - Make/Cmake based, relies on filesystem
  - Supports cluster systems like HTCondor (NAF)*, LSF (KEKCC)
    - Experimental gbasf2 support
  - Very steep learning curve, requires knowledge of regex
- b2luigi: https://b2luigi.readthedocs.io/
  - Design any pipeline in python using Task and Target classes
  - Build your own framework
  - Supports HTCondor, LSF and **gbasf2**
  - Quite easy to master, but code intensive
- Other popular frameworks:
  - Apache Airflow
  - Nextflow

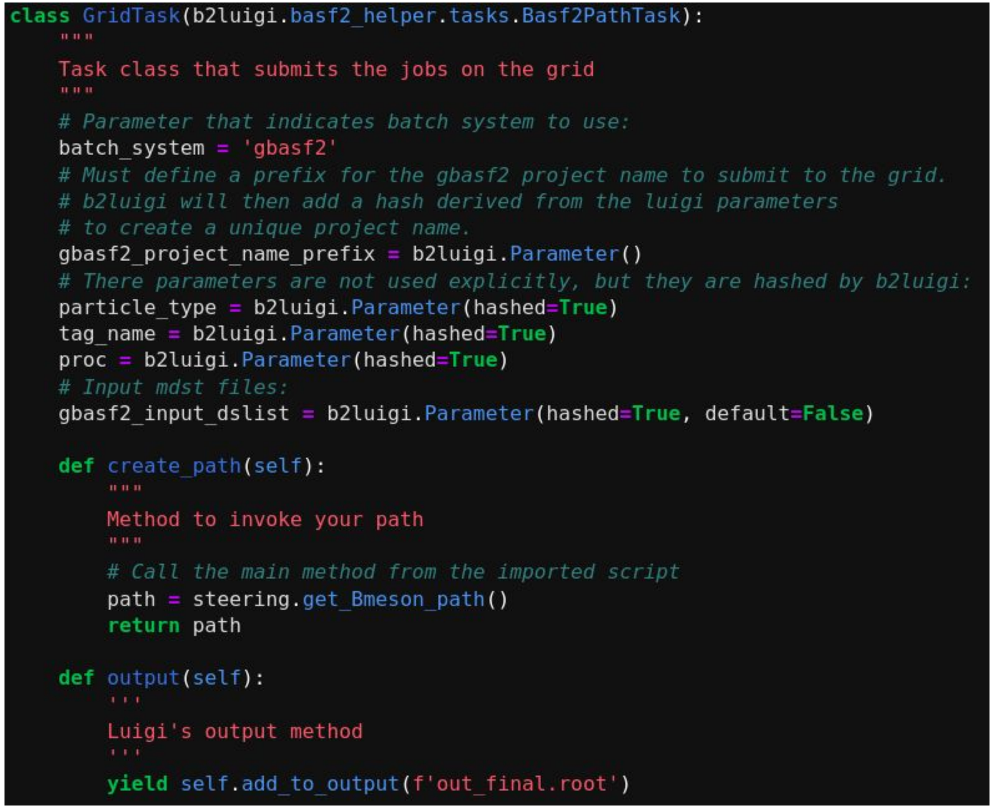
B2luigi

- Building of b2luigi pipeline requires writing a chain of python classes
  - Basic logic element is Task class:
    - `requires()`: yields a list of required tasks
    - `outputs()`: yields a list of Targets
    - `run()`: main logic method
  - LocalTarget object contains output file path
- The tasks are chained output to requires
- Similarly to basf2 one has to run a “process” command:
  - `b2luigi.process([task],local_scheduler=True)`
- b2luigi checks the availability of the Targets and it decides which chain to run
- Tasks are required to raise Exceptions properly in order to catch the errors



```python
import b2luigi

class FileTask(b2luigi.Task):
    filename = b2luigi.Parameter(positional=False)
    def output(self):
        yield b2luigi.LocalTarget(self.filename)
    def run(self):
        open(self.filename, 'w').close()

class AggregatorTask(b2luigi.WrapperTask):
    def requires(self):
        yield FileTask(filename="required.txt")

if __name__ == "__main__":
    main_task_instance = AggregatorTask()
    b2luigi.process([main_task_instance],
                    local_scheduler=True)
```

B2luigi gbasf2 example



```
class GridTask(b2luigi.basf2_helper.tasks.Basf2PathTask):
    """
    Task class that submits the jobs on the grid
    """
    # Parameter that indicates batch system to use:
    batch_system = 'gbasf2'
    # Must define a prefix for the gbasf2 project name to submit to the grid.
    # b2luigi will then add a hash derived from the luigi parameters
    # to create a unique project name.
    gbasf2_project_name_prefix = b2luigi.Parameter()
    # There parameters are not used explicitly, but they are hashed by b2luigi:
    particle_type = b2luigi.Parameter(hashed=True)
    tag_name = b2luigi.Parameter(hashed=True)
    proc = b2luigi.Parameter(hashed=True)
    # Input mdst files:
    gbasf2_input_dslist = b2luigi.Parameter(hashed=True, default=False)

    def create_path(self):
        """
        Method to invoke your path
        """
        # Call the main method from the imported script
        path = steering.get_Bmeson_path()
        return path

    def output(self):
        '''
        Luigi's output method
        '''
        yield self.add_to_output(f'out_final.root')
```

- No need to setup gbasf2 environment in different terminal!
- To use gbasf2 functionality one has to use a `Basf2PathTask` with `batch_system='gbasf2'` parameter
- The task will submit the project, monitor, reschedule jobs and download the output
- Additional settings can be provided in `settings.json` file:
  - `gbasf2_install_directory`
  - `gbasf2_release`
  - `gbasf2_max_retries`
  - `gbasf2_release`

Workflow managers summary

- Workflow managers will simplify your interaction with gbasf2
  - The example will submit, monitor, reschedule jobs and download the output
  - Merge output into one file

- The analysis workflow can scale beyond the Grid
  - It is possible to create a chain that will produce the result, value±uncertainties, using the data on the grid as an input
  - Important for the analysis repetition and preservation

Closing words and Acknowledgements

Closing words and Acknowledgements

- Almost everything in these slides has been taken from previous lectures and StarterKits on the subject, as well as from the online book.

- Huge effort by the Training and Documentation group to provide easily accessible material.

- However, nothing is perfect. You might find mistakes or missing information.

- Help us fix it. You can make the difference!
  https://software.belle2.org/development/sphinx/online_book/join_us.html

QUESTIONS?

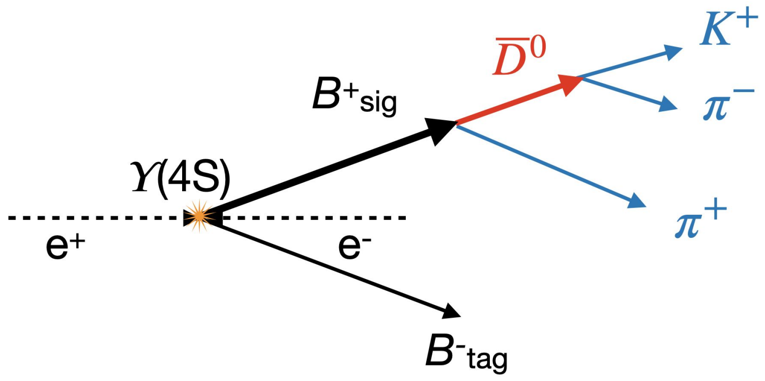
Reconstruct a simple B decay

We will now reconstruct **B- → D0 π-**, with **D0 → K- π+** .

We'll start selecting final state particles and then combining them to form D and then B candidates, perform a kinematic fit, associate to MC information, and save everything in a ROOT output file.

The next slides show step by step the composition of a script that performs all these operations using basf2.

The script is available on KEKCC at
`/home/belle2/manfredi/starterKit/BtoD0pi_example.py`



Save output -- define variables to save

```
# Create aliases for variables

import variables.collections as vc
import variables.utils as vu

track_vars = vu.create_aliases_for_selected(
     list_of_variables=vc.kinematics + vc.mc_truth,
     decay_string='B+ -> [D0 -> ^K+ ^pi-] ^pi+',
     prefix=['D_K', 'D_pi', 'B_pi'])

meson_vars = vu.create_aliases_for_selected(
     list_of_variables=vc.kinematics + vc.mc_truth +
     vc.inv_mass + vc.vertex,
     decay_string='^B+ -> [^D0 -> K+ pi-] pi+',
     prefix=['B', 'D'])
```

Save output -- define variables to save

```
# Create aliases for variables

import variables.collections as vc
import variables.utils as vu

track_vars = vu.create_aliases_for_selected(
     list_of_variables=vc.kinematics + vc.mc_truth,
     decay_string='B+ -> [D0 -> ^K+ ^pi-] ^pi+',
     prefix=['D_K', 'D_pi', 'B_pi'])

meson_vars = vu.create_aliases_for_selected(
     list_of_variables=vc.kinematics + vc.mc_truth +
     vc.inv_mass + vc.vertex,
     decay_string='^B+ -> [^D0 -> K+ pi-] pi+',
     prefix=['B', 'D'])
```

Contains predefined collections of variables.
Documentation: variables.collections

Save output -- define variables to save

```
# Create aliases for variables

import variables.collections as vc
import variables.utils as vu

track_vars = vu.create_aliases_for_selected(
    list_of_variables=vc.kinematics + vc.mc_truth,
    decay_string='B+ -> [D0 -> ^K+ ^pi-] ^pi+',
    prefix=['D_K', 'D_pi', 'B_pi'])

meson_vars = vu.create_aliases_for_selected(
    list_of_variables=vc.kinematics + vc.mc_truth +
    vc.inv_mass + vc.vertex,
    decay_string='^B+ -> [^D0 -> K+ pi-] pi+',
    prefix=['B', 'D'])
```

Contains functions to manage variable names and define aliases.
Documentation: variables.utils

Save output -- define variables to save

```
# Create aliases for variables

import variables.collections as vc
import variables.utils as vu

track_vars = vu.create_aliases_for_selected(
    list_of_variables=vc.kinematics + vc.mc_truth,
    decay_string='B+ -> [D0 -> ^K+ ^pi-] ^pi+',
    prefix=['D_K', 'D_pi', 'B_pi'])

meson_vars = vu.create_aliases_for_selected(
    list_of_variables=vc.kinematics + vc.mc_truth +
    vc.inv_mass + vc.vertex,
    decay_string='^B+ -> [^D0 -> K+ pi-] pi+',
    prefix=['B', 'D'])
```

Adds prefixes for all input variables, for all the particles selected in a decay string, avoiding ambiguities.
documentation:createAliasesForSelected

Save output -- write in tree/histogram

```
# Saves variables for each candidate in a ROOT tree
ma.variablesToNtuple('B+:D0pi', ['charge', 'isContinuumEvent', 'nTracks'] +
     vc.deltae_mbc + meson_vars + track_vars,filename='outputTree.root',
     treename='myTree', path=my_path)


# Saves variables for each candidate in a ROOT histogram
ma.variablesToHistogram('B+:D0pi', [('deltaE', 40, -0.3, 0.3),
     ('Mbc', 60, 5.2, 5.3)], path=my_path)
```

Save output -- write in tree/histogram

```
# Saves variables for each candidate in a ROOT tree
ma.variablesToNtuple('B+:D0pi', ['charge', 'isContinuumEvent', 'nTracks'] +
    vc.deltae_mbc + meson_vars + track_vars,filename='outputTree.root',
    treename='myTree', path=my_path)


# Saves variables for each candidate in a ROOT histogram
ma.variablesToHistogram('B+:D0pi', [('deltaE', 40, -0.3, 0.3),
    ('Mbc', 60, 5.2, 5.3)], path=my_path)
```

Save selected variables of a particle list into ROOT trees or histograms.
documentation:variablesToNtuple
documentation:variablesToHistogram

Outline

- Setup of gbasf2
- Introduction on the grid
- Develop a basf2 steering file. (DONE)
- Test it locally. (DONE)
- Locate your input files.
- Submit jobs to the grid with the same steering file.
- Download the output to perform the offline analysis (plots, fits, etc.)

Converting p12 to PEM

- Copy your certificate, e.g. `myCert.p12`, to the computer (e.g. KEKCC) where you will run `gb2_proxy_init`.
- If this is the first time, you may not have the directory .globus, then make it:
```
mkdir -p $HOME/.globus
```
- Extract the certificate. You need to execute the two commands to extract both `usercert.pem` and `userkey.pem`:
```
openssl pkcs12 -in myCert.p12 -clcerts -nokeys -out
$HOME/.globus/usercert.pem
openssl pkcs12 -in myCert.p12 -nocerts -out $HOME/.globus/userkey.pem
```
- You must set the mode of your `userkey.pem` file to read/write only by the owner, otherwise `voms-proxy-init` will not use it:
```
chmod go-rw $HOME/.globus/userkey.pem
```
- Delete the myCert.p12 file from KEKCC to avoid security issues

Installing gbasf2 (see also https://confluence.desy.de/display/BI/Computing+GBasf2)

- This exercise assumes you will run gbasf2 on KEKCC.
- Make sure you do not have basf2 setup (you cannot run basf2 and gbasf2 from the same terminal session).

```
$ mkdir gbasf2 && cd gbasf2
$ wget -N http://belle2.kek.jp/~dirac/dirac-install.py
$ python dirac-install.py -V Belle-KEK
$ source bashrc # or cshrc if you use csh.
$ dirac-proxy-init -x # enter certificate password
$ dirac-configure defaults-Belle-KEK.cfg
```

- Once the above installation is done, you only need to execute two commands every time that you open a new terminal:

```
$ source ~/gbasf2/BelleDIRAC/gbasf2/tools/setup
$ gb2_proxy_init -g belle # enter certificate password
```

Troubleshooting

Sometimes, things do not go well. A few jobs can fail because a large list of reasons, like

- A timeout in the transfer of a file between sites.
- A central service not available for a short period of time.
- An issue in the site hosting the job.
- etc.

To reschedule failed jobs, you can use `gb2_job_reschedule -p <project name>`:

```
gb2_job_reschedule --usage
Resubmit failed jobs or projects.
Only jobs which have fatal status (Failed, Killed, Stalled) are affected.
Exact same sandbox and parameters are reused. Thus you may need to submit different job if they are wrong.
By default, select only your jobs in current group.
Please switch group and user name by options.
All user's jobs are specified by '-u all'.
Examples:
% gb2_job_reschedule -j 723428,723429
% gb2_job_reschedule -p project1 -u user
```

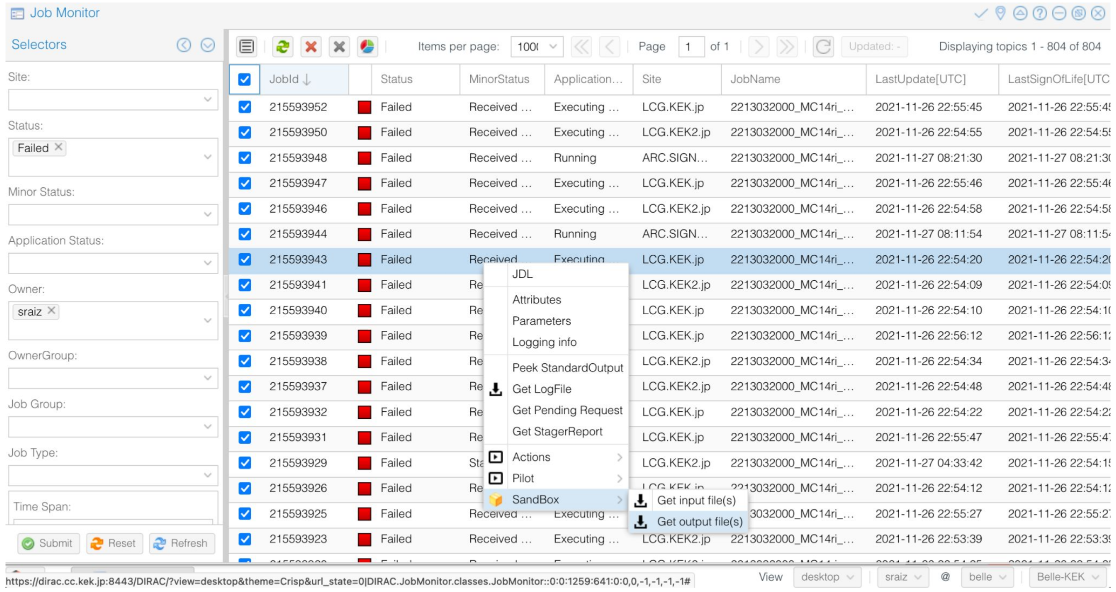
Or you can use the job monitor in the DIRAC web portal, selecting the failed jobs and clicking the ‘Reschedule’ button.

Troubleshooting

What if **all** your jobs failed?

Most probably there is something wrong with the steering file or the gbasf2 arguments.

A useful way to track which was the problem is (if possible) downloading the output sandbox. It contains the logs related to your job.



Site:

Status: Failed ×

Minor Status:

Application Status:

Owner: sraiz ×

OwnerGroup:

Job Group:

Job Type:

Time Span:

Items per page: 100(| Page 1 of 1 | Updated: - | Displaying topics 1 - 804 of 804

	JobId ↓		Status	MinorStatus	Application...	Site	JobName	LastUpdate[UTC]	LastSignOfLife[UTC
✓	215593952	🟥	Failed	Received ...	Executing ...	LCG.KEK.jp	2213032000_MC14ri_...	2021-11-26 22:55:45	2021-11-26 22:55:4
✓	215593950	🟥	Failed	Received ...	Executing ...	LCG.KEK2.jp	2213032000_MC14ri_...	2021-11-26 22:54:55	2021-11-26 22:54:5
✓	215593948	🟥	Failed	Received ...	Running	ARC.SIGN...	2213032000_MC14ri_...	2021-11-27 08:21:30	2021-11-27 08:21:3
✓	215593947	🟥	Failed	Received ...	Executing ...	LCG.KEK.jp	2213032000_MC14ri_...	2021-11-26 22:55:46	2021-11-26 22:55:4
✓	215593946	🟥	Failed	Received ...	Executing ...	LCG.KEK2.jp	2213032000_MC14ri_...	2021-11-26 22:54:58	2021-11-26 22:54:5
✓	215593944	🟥	Failed	Received ...	Running	ARC.SIGN...	2213032000_MC14ri_...	2021-11-27 08:11:54	2021-11-27 08:11:5
✓	215593943	🟥	Failed	Received ...	Executing ...	LCG.KEK.jp	2213032000_MC14ri_...	2021-11-26 22:54:20	2021-11-26 22:54:2
✓	215593941	🟥	Failed	Re		LCG.KEK2.jp	2213032000_MC14ri_...	2021-11-26 22:54:09	2021-11-26 22:54:0
✓	215593940	🟥	Failed	Re		LCG.KEK.jp	2213032000_MC14ri_...	2021-11-26 22:54:10	2021-11-26 22:54:1
✓	215593939	🟥	Failed	Re		LCG.KEK.jp	2213032000_MC14ri_...	2021-11-26 22:56:12	2021-11-26 22:56:1
✓	215593938	🟥	Failed	Re		LCG.KEK2.jp	2213032000_MC14ri_...	2021-11-26 22:54:34	2021-11-26 22:54:3
✓	215593937	🟥	Failed	Re		LCG.KEK2.jp	2213032000_MC14ri_...	2021-11-26 22:54:48	2021-11-26 22:54:4
✓	215593932	🟥	Failed	Re		LCG.KEK.jp	2213032000_MC14ri_...	2021-11-26 22:54:22	2021-11-26 22:54:2
✓	215593931	🟥	Failed	Re		LCG.KEK.jp	2213032000_MC14ri_...	2021-11-26 22:55:47	2021-11-26 22:55:4
✓	215593929	🟥	Failed	Sta		LCG.KEK2.jp	2213032000_MC14ri_...	2021-11-27 04:33:42	2021-11-26 22:54:1
✓	215593926	🟥	Failed	Re		LCG.KEK.jp	2213032000_MC14ri_...	2021-11-26 22:54:12	2021-11-26 22:54:1
✓	215593925	🟥	Failed	Received ...	Executing ...		3032000_MC14ri_...	2021-11-26 22:55:27	2021-11-26 22:55:2
✓	215593923	🟥	Failed	Received ...	Executing ...	LCG.KEK2.jp	2213032000_MC14ri_...	2021-11-26 22:53:39	2021-11-26 22:53:3

https://dirac.cc.kek.jp:8443/DIRAC/?view=desktop&theme=Crisp&url_state=0|DIRAC.JobMonitor.classes.JobMonitor::0:0:1259:641:0:0,0,-1,-1,-1,-1#

Where to get help

https://questions.belle2.org/questions/

You can always email: comp-users-forum@belle2.org

It is a forum for discussion between users, please feel free to participate.

Sign up to receive emails:

https://lists.belle2.org/sympa/info/comp-users-forum

chat.belle2.org, e.g. https://chat.belle2.org/channel/starterkit-workshop

Documentation on confluence:

e.g. https://confluence.desy.de/display/BI/Computing+GBasf2

Other issues, look here:

https://confluence.desy.de/display/BI/Belle+II+Support+Contacts