Karlsruhe Institute of Technology

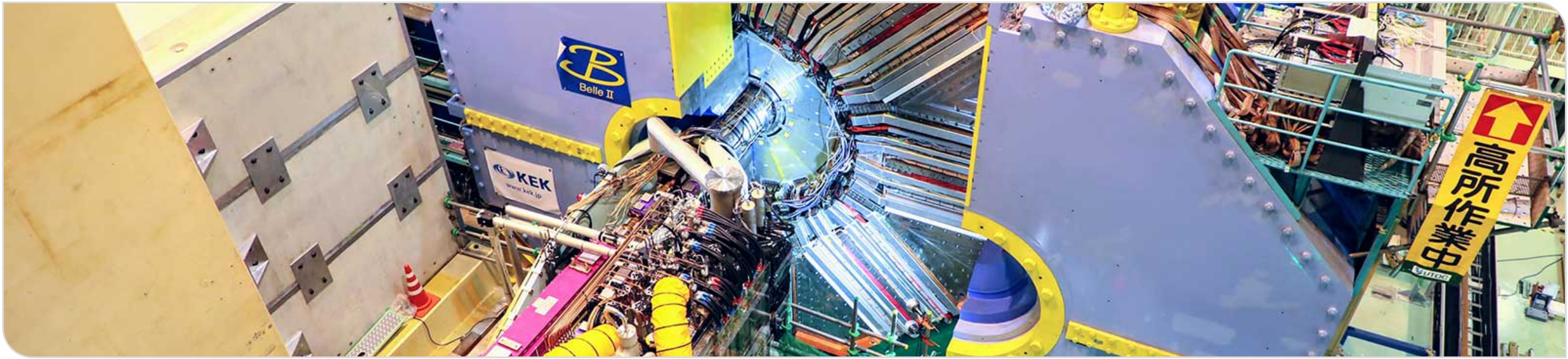
Institute of Experimental Particle Physics

# b2luigi - Bringing Batch 2 luigi!

**PyHEP 2024 - "Python in HEP" Users Workshop**

**Alexander Heidelbach**, Jonas Eppelt, Giacomo De Pietro

alexander.heidelbach@kit.edu



# Outline

- Why Workflow Management
- Basics of luigi
- b2luigi - bring batch 2 luigi!
- Examples at Belle II
- Summary & Discussion

## b2luigi

`b2luigi` — bringing batch 2 luigi!

`b2luigi` is a helper package for `luigi` for scheduling large `luigi` workflows on a batch system. It is as simple as

```python
import b2luigi


class MyTask(b2luigi.Task):
    def output(self):
        return b2luigi.LocalTarget("output_file.txt")

    def run(self):
        with self.output().open("w") as f:
            f.write("This is a test\n")


if __name__ == "__main__":
    b2luigi.process(MyTask(), batch=True)
```

Jump right into it with out Quick Start.

If you have never worked with `luigi` before, you may want to have a look into the luigi documentation. But you can learn most of the nice features also from this documentation!

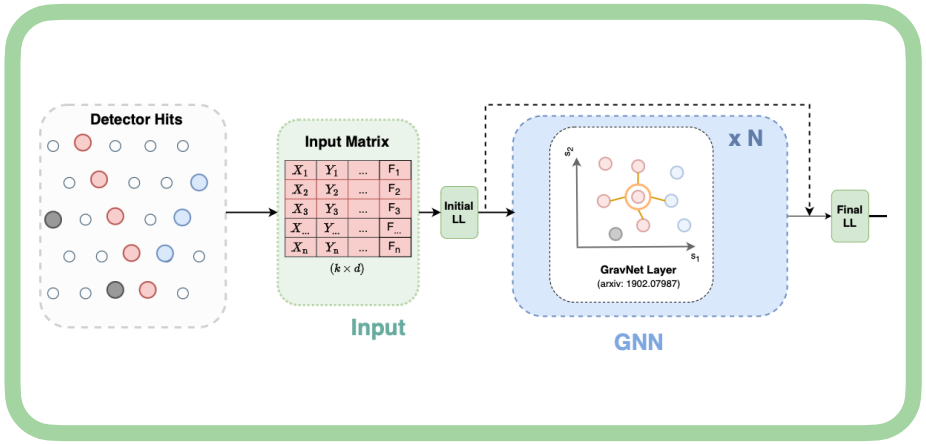


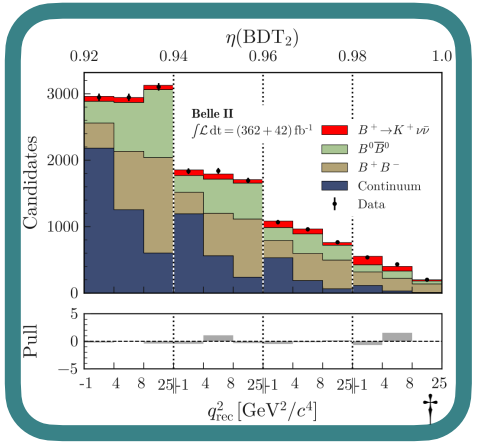
Belle II Group,
ETP, Karlsruhe Institute of Technology

# Analysis in a Nutshell

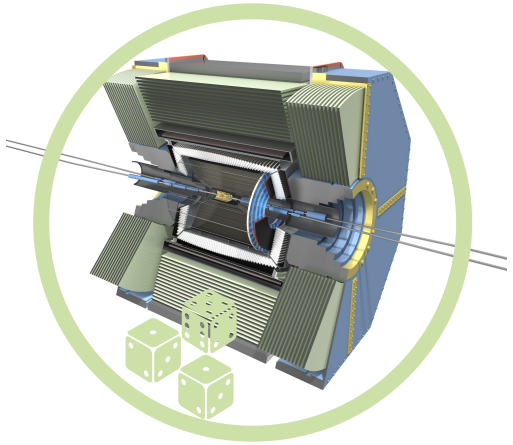
Institute of Experimental Particle Physics

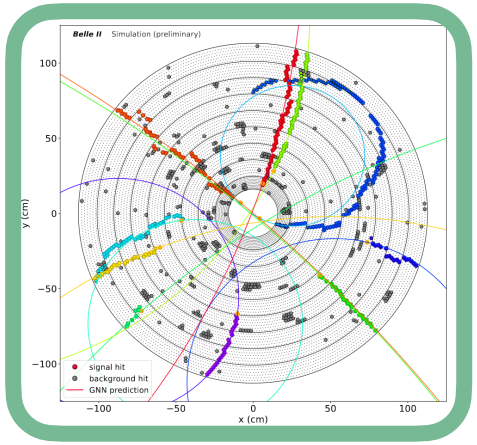
†PhysRevD.109.112006

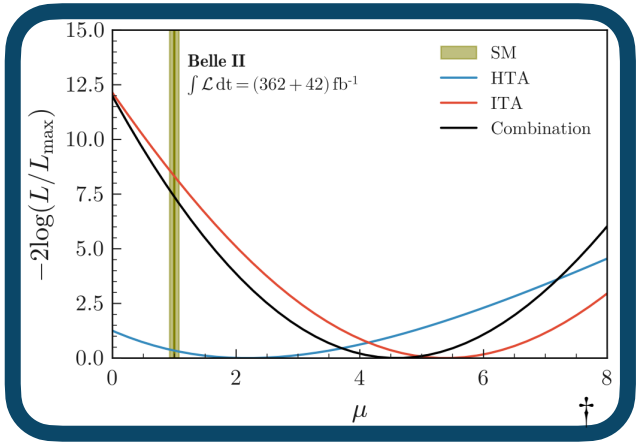



Data → Preprocessing → Reconstruction → Selection → Analysis



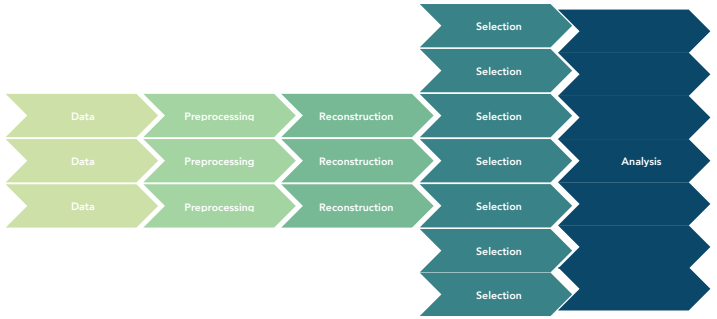






Belle II Group,
ETP, Karlsruhe Institute of Technology

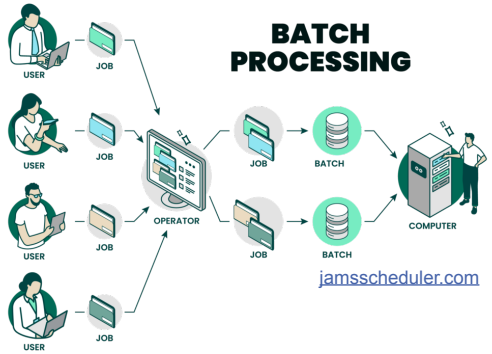

# Challenges

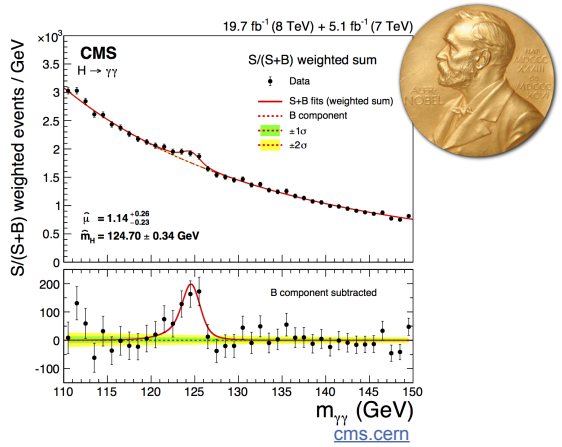
Complex Pipelines

Resource Management

Scalability and Parallel Processing

Reproducibility and Reliability






Belle II Group,
ETP, Karlsruhe Institute of Technology

# luigi in a Nutshell

- **Building a pipeline**
  - Dependency Resolution
  - Workflow Management
  - Visualisation
  - Handling Failures
  - Command Line Integration
  - …
- https://github.com/spotify/luigi

Spotify®

"Hello World" in luigi:

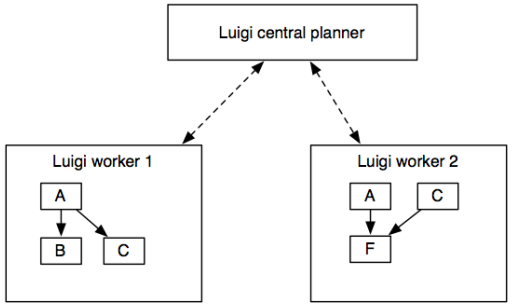
```
class MyTask(luigi.Task):
        parameter = luigi.Parameter()

        def run(self):
                do_smth(self.parameter)

        def output(self):
                return Target("some/file")

        def requires(self):
                yield OtherTask()
```

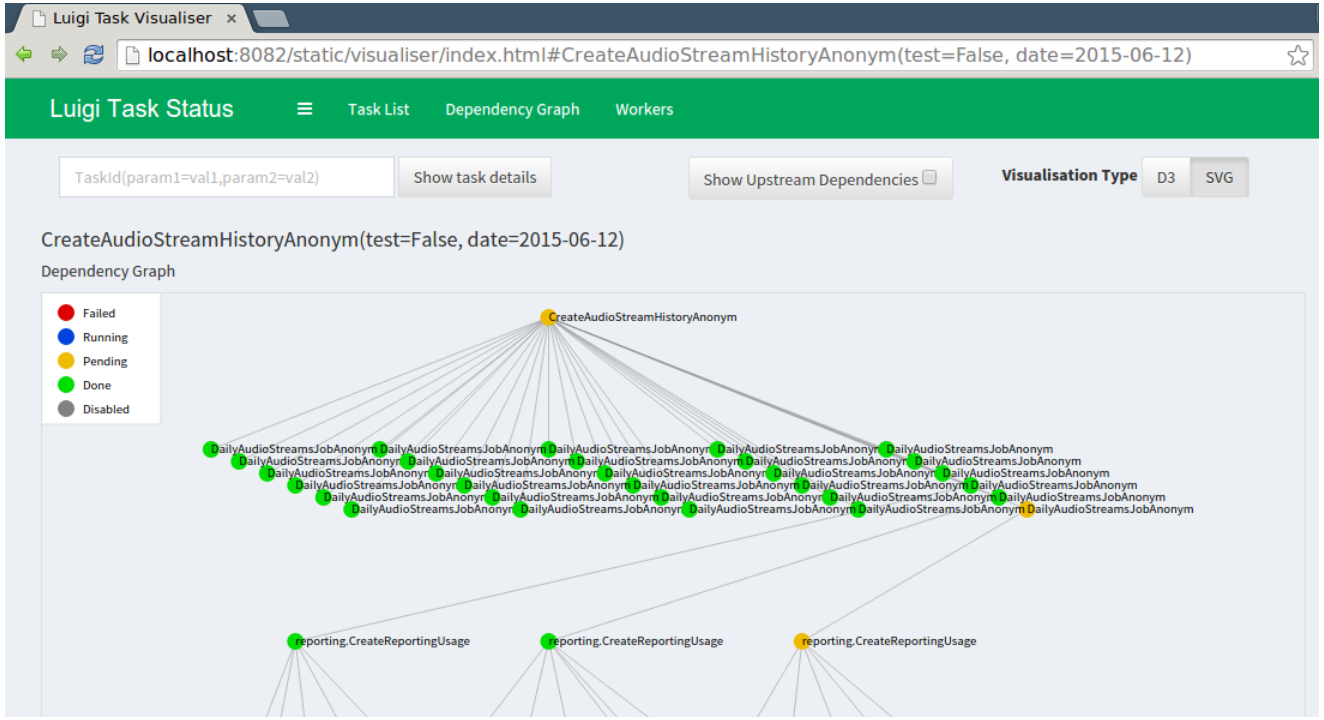


Belle II Group,
ETP, Karlsruhe Institute of Technology

# The luigi Scheduler

- User encodes dependencies of tasks
- The scheduler builds dependency graph and makes sure that multiple workers don't execute the same job
- Graphical visualisation provided



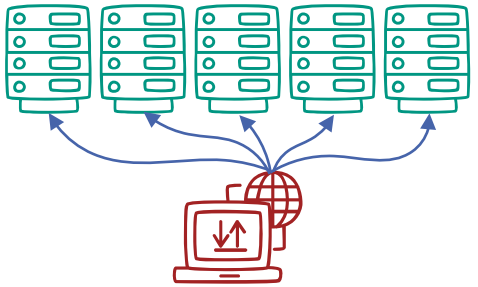



Belle II Group,
ETP, Karlsruhe Institute of Technology

# Why b2luigi?

Institute of Experimental Particle Physics

- **Many many many jobs!**
  - Running batch job = running process
  - **Problem**: Limitation on the number of processes per user
  - **Solution**: Single process on submission machine



- **Many many many tasks!**
  - **Problem**: Tasks in luigi need to adjust for batch execution specifically
  - **Solution**: Abstract batch submission away from the task

```
b2luigi.process(...,batch=True)
```

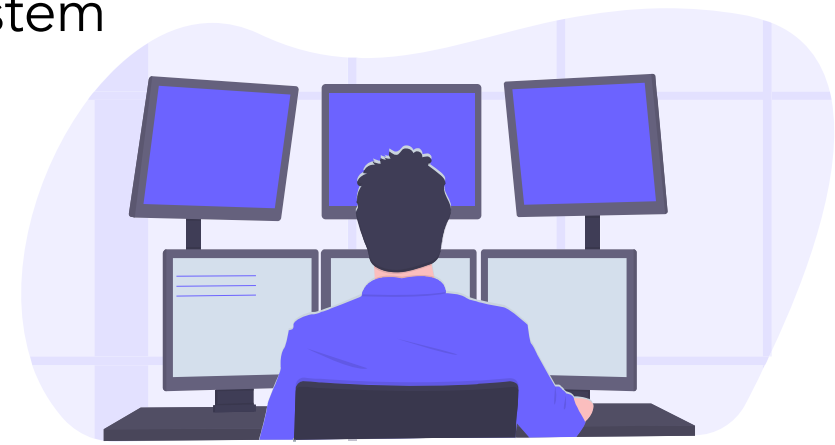
- **Which batch system?**
  - **Problem**: Batch system usage defined by task instance
  - **Solution**: Batch system usage only defined by config variable

HTCondor Software Suite ?

?

# Before we dive into it...

- b2luigi was written by a group of **PhD students** for **their analyses**
  - **Goal:** Make everyday work a little bit easier
  - **Not a goal:** Invent the next workflow management system
    - A lot of helper functions on top of luigi
    - Easy transition between luigi and b2luigi



- Since this year Belle II has been the official maintainer

  → Completely new team of developers

  → Targeting collaboration (and beyond…) wide use

# Something to Click

## b2luigi

pip install b2luigi

sphinx latest | license GPL-3.0 | pypi v1.0.1 | DOI 10.5281/zenodo.11207742

`b2luigi` is a helper package constructed around `luigi` that helps you schedule working packages (so-called tasks) locally or on a batch system. Apart from the very powerful dependency management system by `luigi`, `b2luigi` extends the user interface and has a built-in support for the queue systems, e.g. LSF and HTCondor.

You can find more information in the documentation. Please note that most of the core features are handled by `luigi`, which is described in the separate luigi documentation, where you can find a lot of useful information.

If you find any bugs or want to add a feature or improve the documentation, please send me a pull request! Check the development documentation on information how to contribute.

Contributors are listed here.

This project is in still beta. Please be extra cautious when using in production mode.

To get notified about new features, (potentially breaking) changes, bugs and their fixes, I recommend using the `Watch` button on GitHub to get notifications for new releases and/or issues or to subscribe the releases feed (requires no GitHub account, just a feed reader).

Github

Documentation

Zenodo

PyPi



Belle II Group,
ETP, Karlsruhe Institute of Technology

# A Simple Task

```python
import b2luigi
import random

class MyNumberTask(b2luigi.Task):
    some_parameter = b2luigi.IntParameter()

    def output(self):
        yield self.add_to_output("output_file.txt")

    def run(self):
        random_number = random.random()

        with open(self.get_output_file_name("output_file.txt"), "w") as f:
            f.write(f"{random_number}\n")

if __name__ == "__main__":
    b2luigi.set_setting("result_dir", "results")
    b2luigi.process([MyNumberTask(some_parameter=i) for i in range(100)], workers=200)
```

Execution:

```
python script.py --batch
```

- Add for batch mode!
- Batch jobs are only scheduled when all dependencies are fulfilled
- On your local machine runs only the scheduling mechanism

  → saving local resources



Belle II Group,
ETP, Karlsruhe Institute of Technology

# A Simple Task Output

```
===== Luigi Execution Summary =====

Scheduled 100 tasks of which:
* 100 ran successfully:
    - 100 MyTask(some_parameter=0,1,10,11,12,13,14,15,16,17,18,...)

This progress looks :) because there were no failed tasks or missing dependencies

===== Luigi Execution Summary =====
```

```
❯ ls
results  simple-example.py
❯ ls results
'some_parameter=0'   'some_parameter=27'  'some_parameter=45'  'some_parameter=63'  'some_parameter=81'
'some_parameter=1'   'some_parameter=28'  'some_parameter=46'  'some_parameter=64'  'some_parameter=82'
'some_parameter=10'  'some_parameter=29'  'some_parameter=47'  'some_parameter=65'  'some_parameter=83'
'some_parameter=11'  'some_parameter=3'   'some_parameter=48'  'some_parameter=66'  'some_parameter=84'
'some_parameter=12'  'some_parameter=30'  'some_parameter=49'  'some_parameter=67'  'some_parameter=85'
'some_parameter=13'  'some_parameter=31'  'some_parameter=5'   'some_parameter=68'  'some_parameter=86'
'some_parameter=14'  'some_parameter=32'  'some_parameter=50'  'some_parameter=69'  'some_parameter=87'
'some_parameter=15'  'some_parameter=33'  'some_parameter=51'  'some_parameter=7'   'some_parameter=88'
'some_parameter=16'  'some_parameter=34'  'some_parameter=52'  'some_parameter=70'  'some_parameter=89'
'some_parameter=17'  'some_parameter=35'  'some_parameter=53'  'some_parameter=71'  'some_parameter=9'
'some_parameter=18'  'some_parameter=36'  'some_parameter=54'  'some_parameter=72'  'some_parameter=90'
'some_parameter=19'  'some_parameter=37'  'some_parameter=55'  'some_parameter=73'  'some_parameter=91'
'some_parameter=2'   'some_parameter=38'  'some_parameter=56'  'some_parameter=74'  'some_parameter=92'
'some_parameter=20'  'some_parameter=39'  'some_parameter=57'  'some_parameter=75'  'some_parameter=93'
'some_parameter=21'  'some_parameter=4'   'some_parameter=58'  'some_parameter=76'  'some_parameter=94'
'some_parameter=22'  'some_parameter=40'  'some_parameter=59'  'some_parameter=77'  'some_parameter=95'
'some_parameter=23'  'some_parameter=41'  'some_parameter=6'   'some_parameter=78'  'some_parameter=96'
'some_parameter=24'  'some_parameter=42'  'some_parameter=60'  'some_parameter=79'  'some_parameter=97'
'some_parameter=25'  'some_parameter=43'  'some_parameter=61'  'some_parameter=8'   'some_parameter=98'
'some_parameter=26'  'some_parameter=44'  'some_parameter=62'  'some_parameter=80'  'some_parameter=99'
❯ ls results/some_parameter=0
output_file.txt
```

- The scheduler **tracks** the **state** of each job and **takes action**
- **Job is done = Output file exists**
  - Unique output names per job
  - Add parameters to job output name
  - Create a folder structure for each parameter
  - b2luigi does this automatically
  - **No check of content!**

# A Simple Task Settings

```
class MyTask(b2luigi.Task):
    some_setting = “some value”
```

```
class MyTask(b2luigi.Task):
    @property
    def some_setting(self):
        return “some value”
```

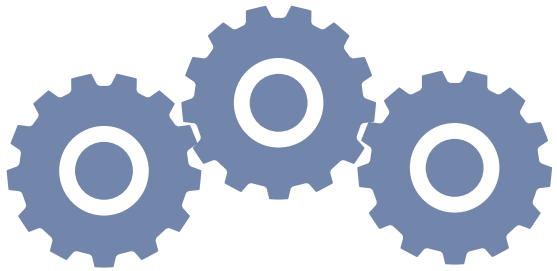
```
b2luigi.set_setting(“some_setting”, "some value")
```

**settings.json**

```
{
    “some_setting”: “some value”
}
```

Priority

- **Settings** are handled by b2luigi!
- **Control**:
  - batch system choice
  - output and log path
  - environment
  - workflow specific settings



# A Simple Task Continued

```python
class MyAverageTask(b2luigi.Task):
    def requires(self):
        for i in range(100):
            yield self.clone(MyNumberTask, some_parameter=i)

    def output(self):
        yield self.add_to_output("average.txt")

    def run(self):
        summed_numbers = 0
        counter = 0
        for input_file in self.get_input_file_names("output_file.txt"):
            with open(input_file, "r") as f:
                summed_numbers += float(f.read())
                counter += 1

        average = summed_numbers / counter

        with open(self.get_output_file_name("average.txt"), "w") as f:
            f.write(f"{average}\n")
```

- More **helper functions**:
  - b2luigi.Task.get_input_file_names()
  - b2luigi.Task.get_output_file_name()
- Jobs that are done will not be run again!

```
› python3 simple-example.py --show-output
average.txt
        /home/aheidelbach/b2luigi/results/average.txt

output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=0/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=1/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=2/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=3/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=4/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=5/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=6/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=7/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=8/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=9/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=10/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=11/output_file.txt
        /home/aheidelbach/b2luigi/results/some_parameter=12/output_file.txt
```

# Batch Processing

- Initial motivation: **Make batch submission easy!**
- Currently **fully supported**:
  - HTCondor
  - LSF
  - Gbasf2 (BelleII@WLCG)
- **Challenges**:
  - Using your environment
    - **Settings**: env_script, env,..
  - Ensuring consistent locations
    - **Settings**: working_dir, result_dir,…

```
class MyTask(b2luigi.Task):
    batch_system = "htcondor"
```

```
class MyTask(b2luigi.Task):
    @property
    def htccondor_settings(self):
        return {"request_memory": 4096}
```

```
b2luigi.set_setting("result_dir", "path/to/result")
```

**settings.json**

```
{
    "env_script": "setup.sh"
}
```



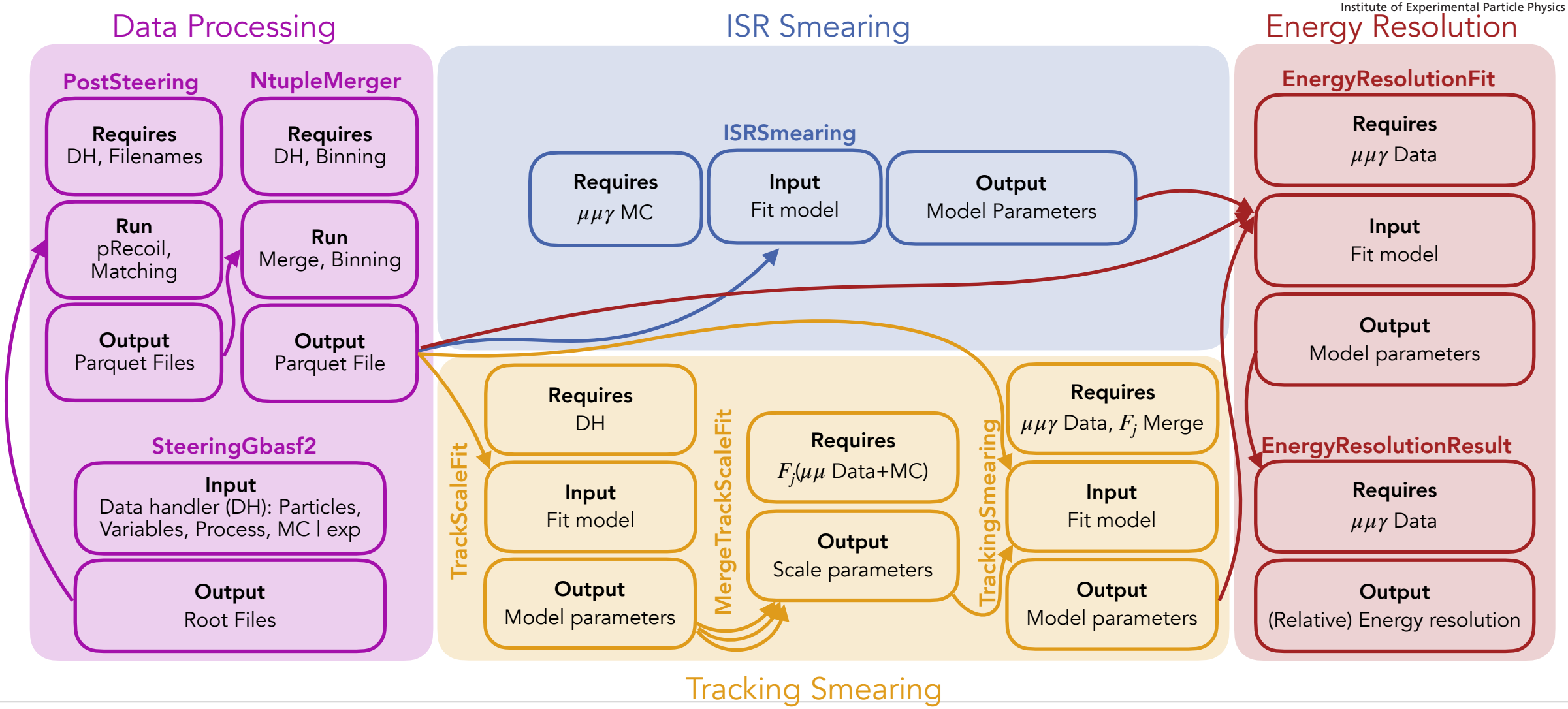
Belle II Group,
ETP, Karlsruhe Institute of Technology

# Example: HTCondor

```python
class MyTask(b2luigi.Task):
    parameter = b2luigi.IntParameter()
    batch_system = "htcondor"

    @property
    def executable(self):
        return ["$MY_PYTHON"]

    def output(self):
        yield self.add_to_output("test.txt")

    def run(self):
        with open(self.get_output_file_name("test.txt"), "w") as f:
            f.write(f"Test {self.parameter}")

class Wrapper(b2luigi.WrapperTask):
    def requires(self):
        for i in range(10):
            yield MyTask(parameter=i)

if __name__ == "__main__":
    b2luigi.set_setting("env_script", "setup.sh")

    b2luigi.set_setting("result_dir", "results")

    b2luigi.process(Wrapper(), batch=True, workers=100)
```

- Script executed on the batch job side
- Location needs to be accessible on the batch job side
- Wrapper tasks need no output
- Definition of the batch system
- Executable for this specific task
  - E.g. environment variable set in setup.sh

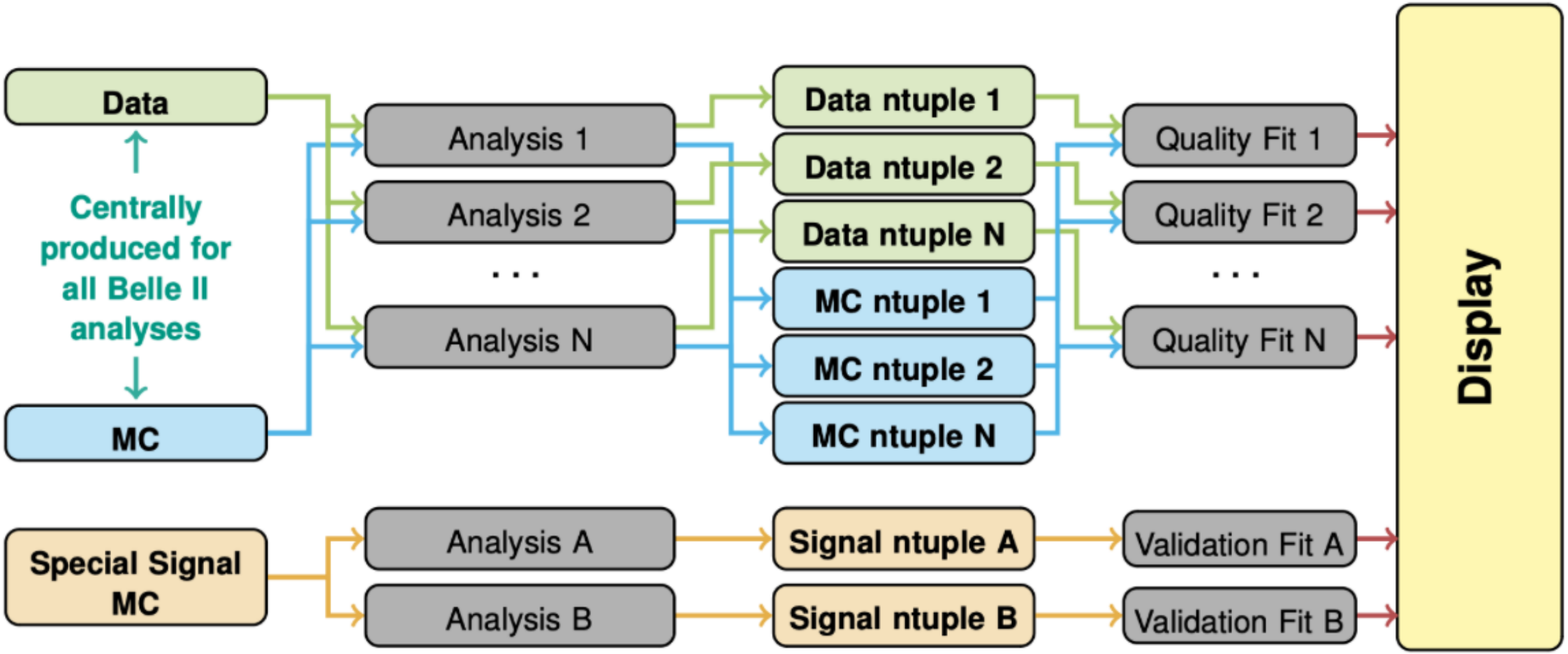
# A typical Analysis






Belle II Group,
ETP, Karlsruhe Institute of Technology

# Validation Interface for the Belle II Experiment



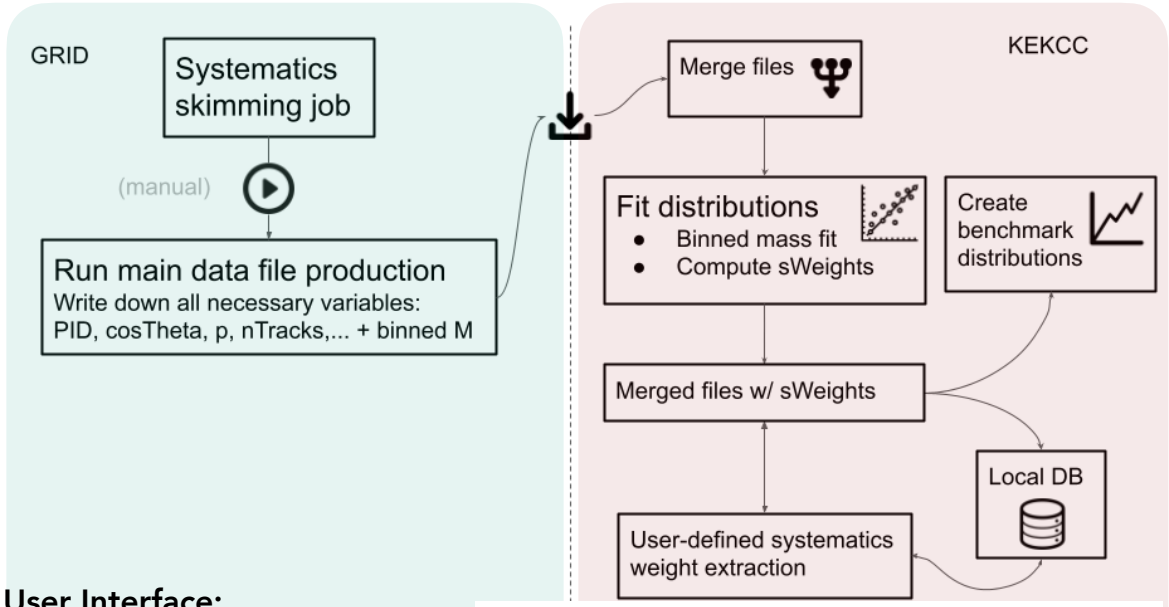



Belle II Group,
ETP, Karlsruhe Institute of Technology

# Systematic Corrections Framework

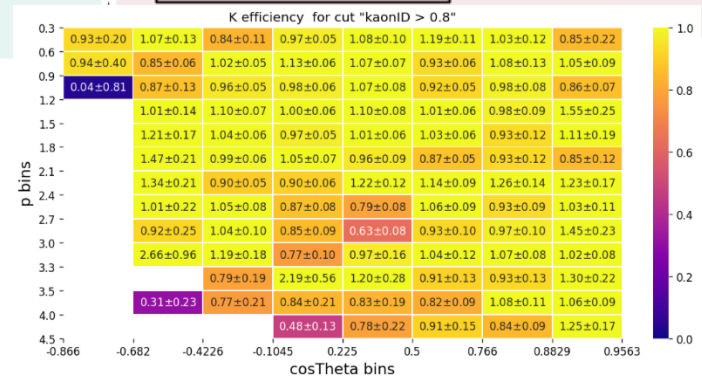
Institute of Experimental Particle Physics

- 2 Stage Algorithm
  - Ntuple Production
    - Centrally run for every campaign
    - Running: Gbasf2
  - Data/MC Corrections
    - User runs their specific selection
    - Running: Locally, HTCondor, LSF
- Also: Validation of different datasets




**User Interface:**

```
# ================================== #
#  Hadron ID weight configuration file  #
# ================================== #
weight_dir: 'fixed_weights/'
remove_tmp_files: True
weight_cfg_list:
-
  prefix_name: Rdtmc_v1
  efficiency particle type: 'K'
  fakerate particle type: 'pi'
  binning: [[0.5, 2.5, 4.5],
            [-0.8, 0.2, 0.9563]]
  track variables: [ "p", "cosTheta" ]
  cuts: [ "kaonID > 0.2", "kaonID > 0.8"]
  precuts: [ "", "charge > 0", "charge < 0" ]
  mc proc query: [ "MC14ri_1" ]
  data_proc_query: [ "proc12e7" ]
```

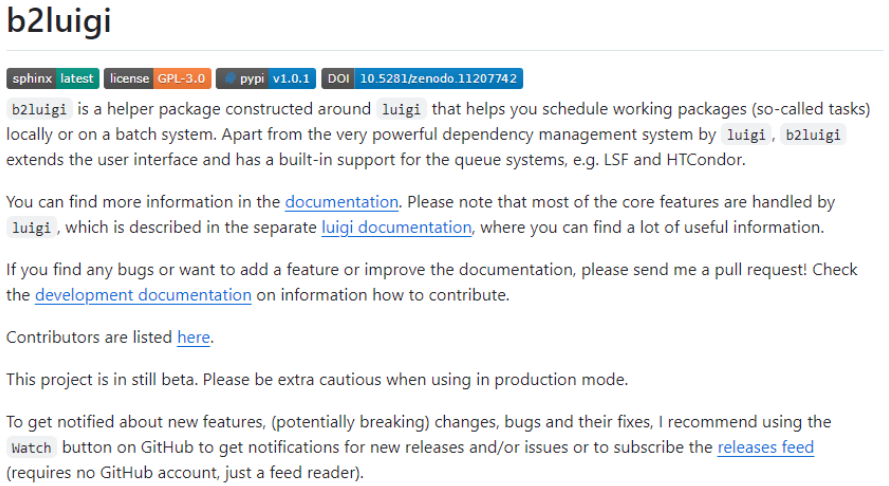



[Documentation]

# Summary & Discussion

Institute of Experimental Particle Physics

- b2luigi provides a simple and flexible implementation to run your workflow on batch systems!
- The abstraction of the batch processing to global settings allows for:
  - Quick change in the submission strategy
  - Simple code and execution
- **Discussion**
  - There are many other workflow management libraries! Do we want a common HEP standard?
  - How can we make the workflow management libraries fully experiment agnostic?



b2luigi

sphinx latest | license GPL-3.0 | pypi v1.0.1 | DOI 10.5281/zenodo.11207742

`b2luigi` is a helper package constructed around `luigi` that helps you schedule working packages (so-called tasks) locally or on a batch system. Apart from the very powerful dependency management system by `luigi`, `b2luigi` extends the user interface and has a built-in support for the queue systems, e.g. LSF and HTCondor.

You can find more information in the documentation. Please note that most of the core features are handled by `luigi`, which is described in the separate luigi documentation, where you can find a lot of useful information.

If you find any bugs or want to add a feature or improve the documentation, please send me a pull request! Check the development documentation on information how to contribute.

Contributors are listed here.

This project is in still beta. Please be extra cautious when using in production mode.

To get notified about new features, (potentially breaking) changes, bugs and their fixes, I recommend using the `Watch` button on GitHub to get notifications for new releases and/or issues or to subscribe the releases feed (requires no GitHub account, just a feed reader).

Github zenodo Zenodo

SPHINX Documentation PyPi