Belle II Software Roadmap

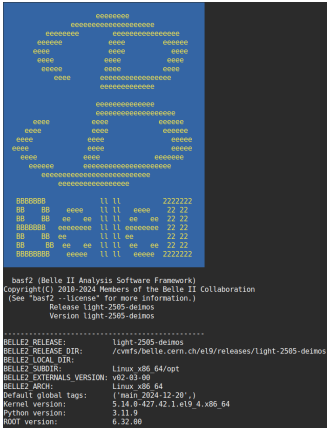
Frank Meier

2025 Belle II Summer Workshop
23 June 2025

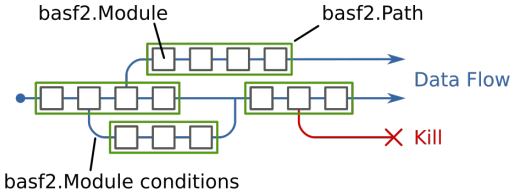
Research supported by

Introduction

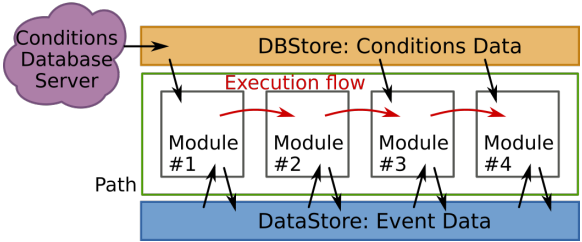
- Belle II Analysis Software Framework (basf2) divided into 35 packages
  - One for each subdetector: arich, cdc, ecl klm, pxd, svd, top, vxd
  - Core packages: framework, reconstruction, tracking
  - Data taking: daq, hlt, rawdata, trg
  - Data quality: alignment, calibration, dqm, validation
  - Data storage tables: mdst, skim
  - MC: decfiles, generators, geometry, simulation, structure
  - Background: background, beast, ir
  - Offline analysis: analysis, b2bii, mva
  - Documentation / outreach: display, masterclass, online_book
- Each package has one or two librarians (total number of librarians: 40)
- Code written in C++
- One shared library created per package and installed in top-level lib directory
- Build system based on SCons



Modular structure

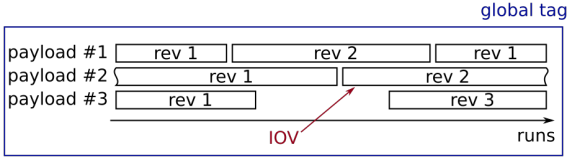


- Linear arrangement of C++ modules in a path
- Core functions of modules
  - initialize
  - beginRun
  - event
  - endRun
  - terminate
- Python steering script to set up path



Conditions Database

- Storage place of additional data needed to interpret and analyze the data that can change over time, *e.g.*, detector configuration or calibration constants
- Payloads: binary objects (usually ROOT files) identified by name and revision number
- Each payload has defined intervals of validity (iov), *i.e.*, experiment and run range
- Globaltag: collection of payloads and iovs for a certain dataset, identified by unique name



- Once prepared globaltag is immutable and cannot be modified any further to ensure reproducibility of analyses
- Different processing iterations use different globaltags
- Globaltag of reconstruction stored in metadata and automatically applied
- Chain of globaltags possible

Data-taking

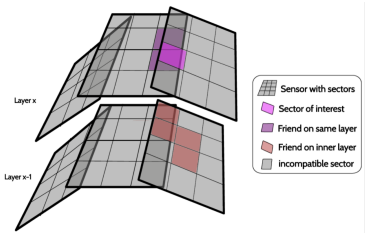
- basf2 runs on 12 high-level trigger nodes (14 from fall 2025)
- ZeroMQ
  - Acts like concurrency framework while looking like an embeddable networking library
  - Sockets that carry atomic messages across various transports like in-process, inter-process, TCP, and multicast
  - Fast
  - Asynchronous I/O model $\rightarrow$ scalable to multi-core operation

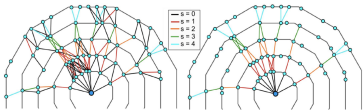
Reconstruction chain

- RootInputModule
- Geometry
- Clustering of calorimeter
- Clustering of pixel and silicon vertex detectors
- Track finding
- Track fitting
- Track extrapolation
- Track-cluster matching
- Software trigger
- Post-filter tracking
- PID
- RootOutputModule

Tracking

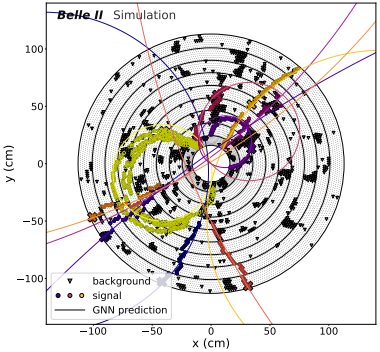
- Pattern recognition / track finding
  - Finding hits belonging to the same charged particle
  - SectorMaps
    ↓
    Segment Network
    ↓
    Cellular Automaton to find longest paths
  - SVD-only pattern recognition and CDC-only pattern recognition
  - Inter-detector track finding via Combinatorial Kalman Filter
- Track fit
  - Extracting track parameters from fit to collection of hits
  - Deterministic Annealing Filter (DAF)
  - Currently use GenFit package (DOI:10.5281/zenodo.10301439)
- Track refining
  - Flip and Refit
- More details in sphinx documentation





Tracking

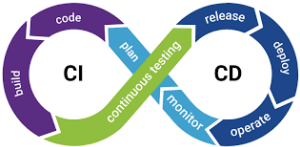
- Pattern recognition / track finding
  - Finding hits belonging to the same charged particle
  - SectorMaps
    ↓
    Segment Network
    ↓
    Cellular Automaton to find longest paths
  - SVD-only pattern recognition and CDC-only pattern recognition
  - Inter-detector track finding via Combinatorial Kalman Filter
- Track fit
  - Extracting track parameters from fit to collection of hits
  - Deterministic Annealing Filter (DAF)
  - Currently use GenFit package (DOI:10.5281/zenodo.10301439)
- Track refining
  - Flip and Refit
- More details in sphinx documentation
- End-to-End Multi-Track Reconstruction using Graph Neural Networks at Belle II Comput. Softw. Big Sci. 9, 6 (2025)



CAT Finder

Unit-tests

- First layer of software validation
- Run full test suite for each commit of open merge requests and each merge into main or release branch
- Unit-tests intended to catch non-trivial dependencies and implications of code changes
- Currently about 1000 unit-tests of C++ code using GoogleTest
  - Check basic functionality of modules, return values of functions and variables
- About 350 additional python tests
  - Make sure that standard scripts do not crash
  - Compare output of certain scripts with reference expectation, *e.g.*, for mdst backward compatibility
- Running all tests (in 16 parallel processes) takes 15 - 20 min



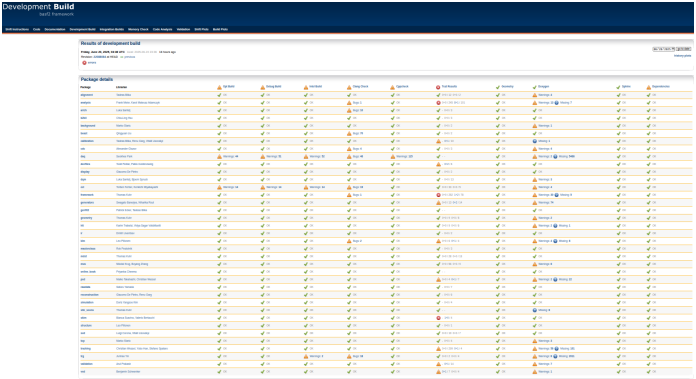
Nightly validation

- Run once per day (night)
- Workflow of nightly validation
  1. Generate smallish samples
  2. Run validation scripts
  3. Create output histograms
  4. Comparison with reference
     - Calculate p value of histogram compatibility
     - Calculate performance numbers, *e.g.*, width of distribution
- Plots of various software releases uploaded to web server
- Email notifications sent out to assigned contacts

Nightly validation

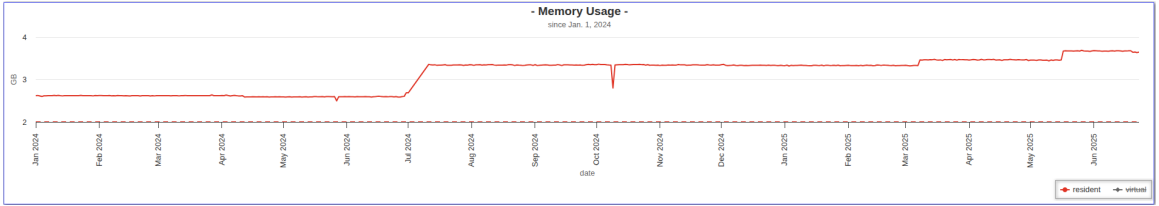
Monitoring

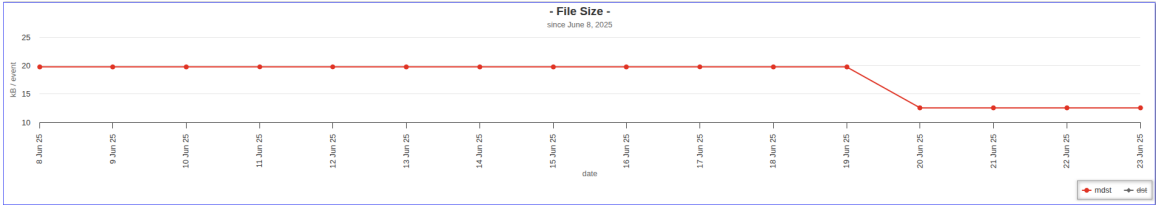
- Nightly build run with different configurations (debug, intel, clang)
- Many resource checks (memory consumption, execution time, output file size)
  - Summarize build warnings, cppcheck, doxygen check, dependency check, geometry overlap check
- History plots of warning and error counters as well as resource usage



Monitoring

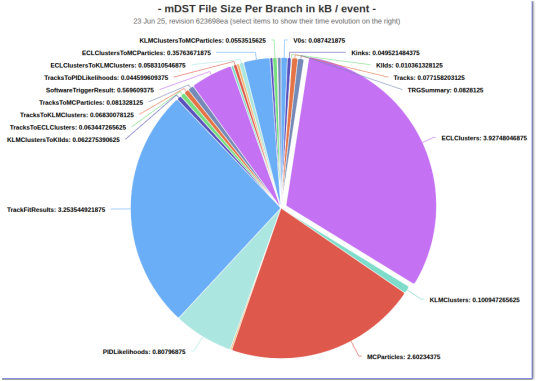
- Nightly build run with different configurations (debug, intel, clang)
- Many resource checks (memory consumption, execution time, output file size)
  - Summarize build warnings, cppcheck, doxygen check, dependency check, geometry overlap check
- History plots of warning and error counters as well as resource usage

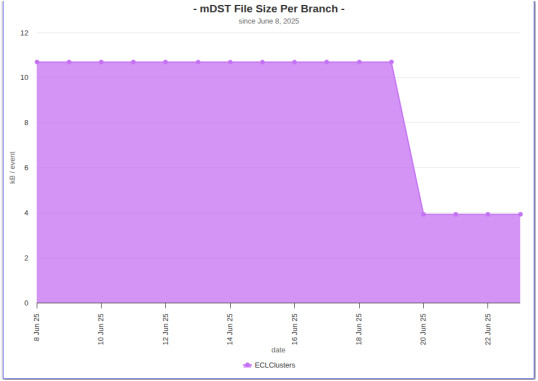




Monitoring

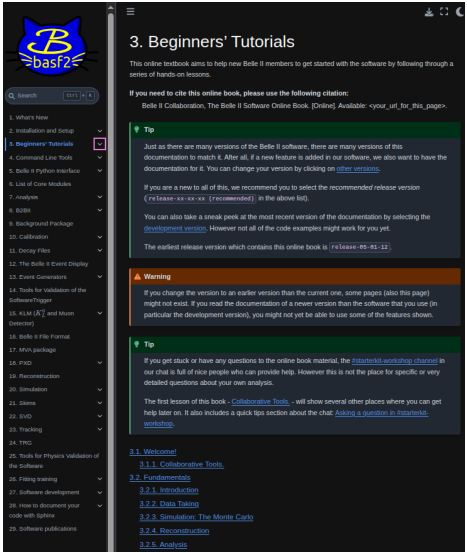
- Nightly build run with different configurations (debug, intel, clang)
- Many resource checks (memory consumption, execution time, output file size)
  - Summarize build warnings, cppcheck, doxygen check, dependency check, geometry overlap check
- History plots of warning and error counters as well as resource usage





Documentation

- Good documentation crucial to (recruiting) process of software development and maintenance
- Documentation publicly available at `https://software.belle2.org/`
- Sphinx
  - Use reStructuredText
  - Use sphinx's autodoc feature to conveniently create documentation based on python's docstrings
- Doxygen for C++ documentation
- Tests for (almost) all packages to ensure that everything is documented



Release policy

- Major releases
  - Once a year (feature freeze for release-10 in one week)
  - Very thorough validation
  - Contains all software changes that are merged to the main branch
- Minor releases
  - Frequency: one to two per major release
  - Limited amount of new features, usually for specific purpose
- Patch releases
  - Mostly for bug fixes, especially for data-taking and calibration
  - During data-taking synchronized with maintenance days
- Light releases
  - Every two months
  - For introduction of new offline data analysis features
  - Contain only framework, mdst, mva, analysis, skim, geometry, online_book, and b2bii packages
  - No unpacking or digitization ⇒ only mdst and udst can be processed
  - Currently named after celestial objects: aldebaran, betelgeuse, ceres, deimos

Supported environments

- basf2 meant to work on any recent 64bit Linux system but only tested and binaries provided for
  - Enterprise Linux 8 or CentOS 8
  - Enterprise Linux 9 or AlmaLinux 9
  - Ubuntu 22.04
  - Ubuntu 24.04
- basf2 distributed via cvmfs
- ARM version under development
- Central Buildbot instance connected via GitLab webhooks to code changes
  $\Rightarrow$ triggers builds on various workers

Externals

- Versioned set of external software packages used and linked against the Belle II software
- Dependency between packages considered and compatibility guaranteed
- C++ packages like
  - ROOT, XRootD
  - gcc, clang, gdb, cmake, Python
  - boost, Eigen, gsl
  - EvtGen, Geant4, clhep, PYTHIA
  - git, cppcheck, doxygen
- Includes patches
- Python packages like pandas, matplotlib, torch, tensorflow, jupyter, ...
- Source files uploaded to web server to never lose availability

Tools

- Collection of scripts to prepare environment for execution of Belle II software
- `b2setup`
  - Setting environment variables
- `b2code-create`, `b2code-style-check`, `b2code-style-fix`, `b2code-clean`
  - Creating local directory for core software development and fixing style issues
- `b2install-prepare`, `b2install-release`, `b2install-externals`, `b2install-data`
  - Installing pre-compiled software versions or example data on local machine
- `b2analysis-create`, `b2analysis-get`, `b2analysis-update`
  - Creating local directory for development of analysis code including preparation of build system and addition of repository to git

Links & License

- basf2 source code
  - Internal GitLab repository at `https://gitlab.desy.de/belle2/software/basf2`
- basf2 links against defined set of third-party libraries called **`externals`**
  - Internal GitLab repository at `https://gitlab.desy.de/belle2/software/externals`
- Repository with scripts to install and set up basf2 called **`tools`**
  - Internal GitLab repository at `https://gitlab.desy.de/belle2/software/tools`
- Repository with script for version managing (recommended releases and global tags)
  - Internal GitLab repository at `https://gitlab.desy.de/belle2/software/versioning`
- LGPL (GNU Lesser General Public License) version 3 or later
  - Header in each file:

```
basf2 (Belle II Analysis Software Framework)
Author: The Belle II Collaboration

See git log for contributors and copyright holders.
This file is licensed under LGPL-3.0, see LICENSE.md.
```

Links & License

- basf2 source code
  - Publicly available at `https://github.com/belle2/basf2`
- basf2 links against defined set of third-party libraries called **`externals`**
  - Publicly available at `https://github.com/belle2/externals`
- Repository with scripts to install and set up basf2 called **`tools`**
  - Publicly available at `https://github.com/belle2/tools`
- Repository with script for version managing (recommended releases and global tags)
  - Publicly available at `https://github.com/belle2/versioning`
- LGPL (GNU Lesser General Public License) version 3 or later
  - Header in each file:

```
basf2 (Belle II Analysis Software Framework)
Author: The Belle II Collaboration

See git log for contributors and copyright holders.
This file is licensed under LGPL-3.0, see LICENSE.md.
```

Conclusion

- Belle II software = C++ code with python interface
- Serial execution of dynamically loaded modules to process collection of events
- Conditions Database stores settings and calibration constants
- Unit-test ensure stability of software
- Nightly validation
- Documentation via sphinx and doxygen
- Please cite Comput. Softw. Big Sci. 3, 1 (2019) and DOI:10.5281/zenodo.5574115 in Belle II papers

Conclusion

- Belle II software = C++ code with python interface
- Serial execution of dynamically loaded modules to process collection of events
- Conditions Database stores settings and calibration constants
- Unit-test ensure stability of software
- Nightly validation
- Documentation via sphinx and doxygen
- Please cite Comput. Softw. Big Sci. 3, 1 (2019) and DOI:10.5281/zenodo.5574115 in Belle II papers

Thanks for your attention!