

Software & Code Management

August 6th, 2018

Tokyo Analysis Workshop

Christian Graf



Max-Planck-Institut für Physik
(Werner-Heisenberg-Institut)



Software Overview



- CALICE Soft:
Collection of packages
- Relies on ILCSOft
- Some are outdated
- Structure of packages not always clear, code may be at different places - needs some clean up

Name	
2009	alignment OFFENTLICH
	calice_analysis OFFENTLICH
	calice_calib OFFENTLICH
	calice_cddata OFFENTLICH
	calice_cmake OFFENTLICH
2010	calice_daq OFFENTLICH
	calice_db_tools OFFENTLICH
	calice_dd_testbeams OFFENTLICH
2011	calice_lcioconv OFFENTLICH
2009	calice_maps OFFENTLICH
2011	calice_monitor OFFENTLICH
2009	calice_online OFFENTLICH
2011	calice_pandora OFFENTLICH
	calice_reco OFFENTLICH
	calice_ROOTmacros OFFENTLICH
	calice_run OFFENTLICH
	calice_sim OFFENTLICH
	calice_steering OFFENTLICH
	calice_tools OFFENTLICH
	calice_tutorial OFFENTLICH
	calice_userlib OFFENTLICH
	fitchalsoftware OFFENTLICH
	labview_converter OFFENTLICH
	RootTreeWriter OFFENTLICH
	TBTrackHelpers OFFENTLICH

- CALICE Soft:
Collection of packages

- Relies on ILCSoft

- Some are outdated

- Structure of packages not always clear, code may be at different places - needs some clean up

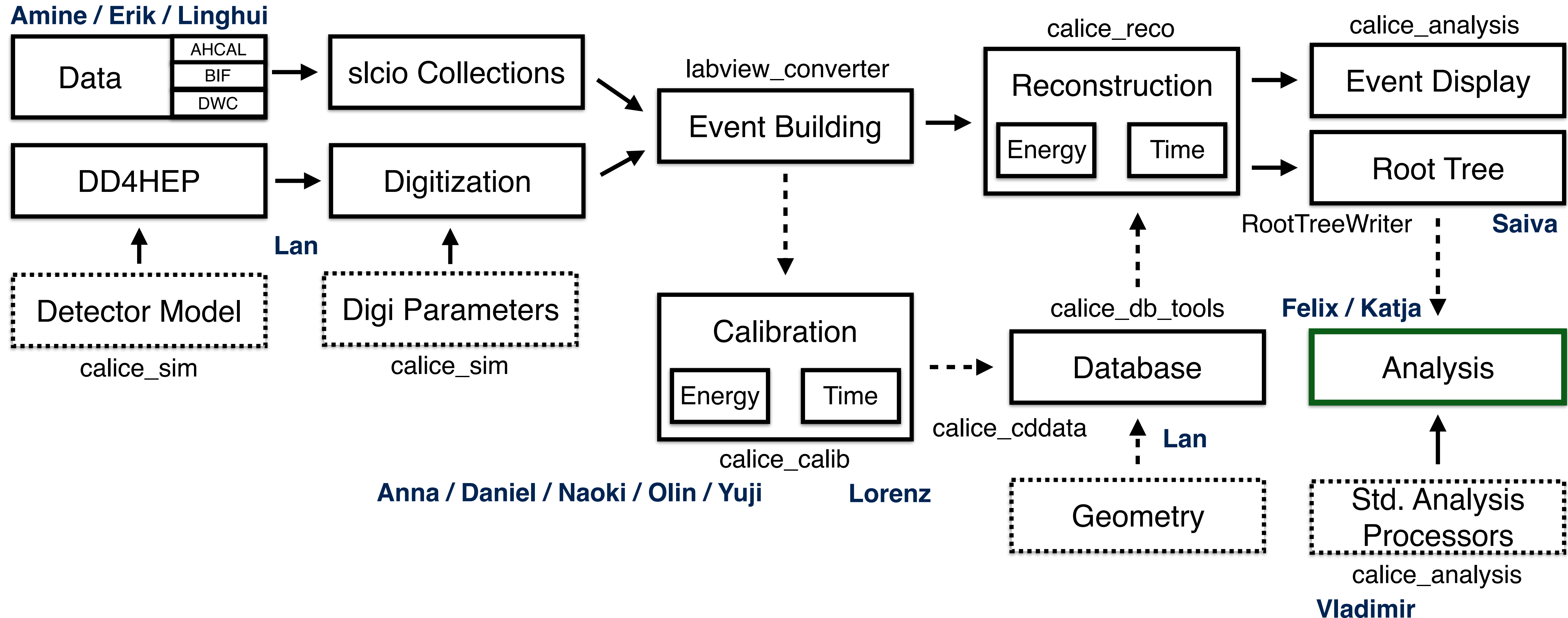
Where is software for pedestal extraction?

- calice_reco/recoSiPM/src/PedestalProcessor.cc
- calice_calib/calib/src/Ahc2PedestalCalibrator.cc
- calice_ROOTmacros/Ahc2_PedestalExtractor/
- flchcalsoftware/Extraction_Pedestal/

Dateien

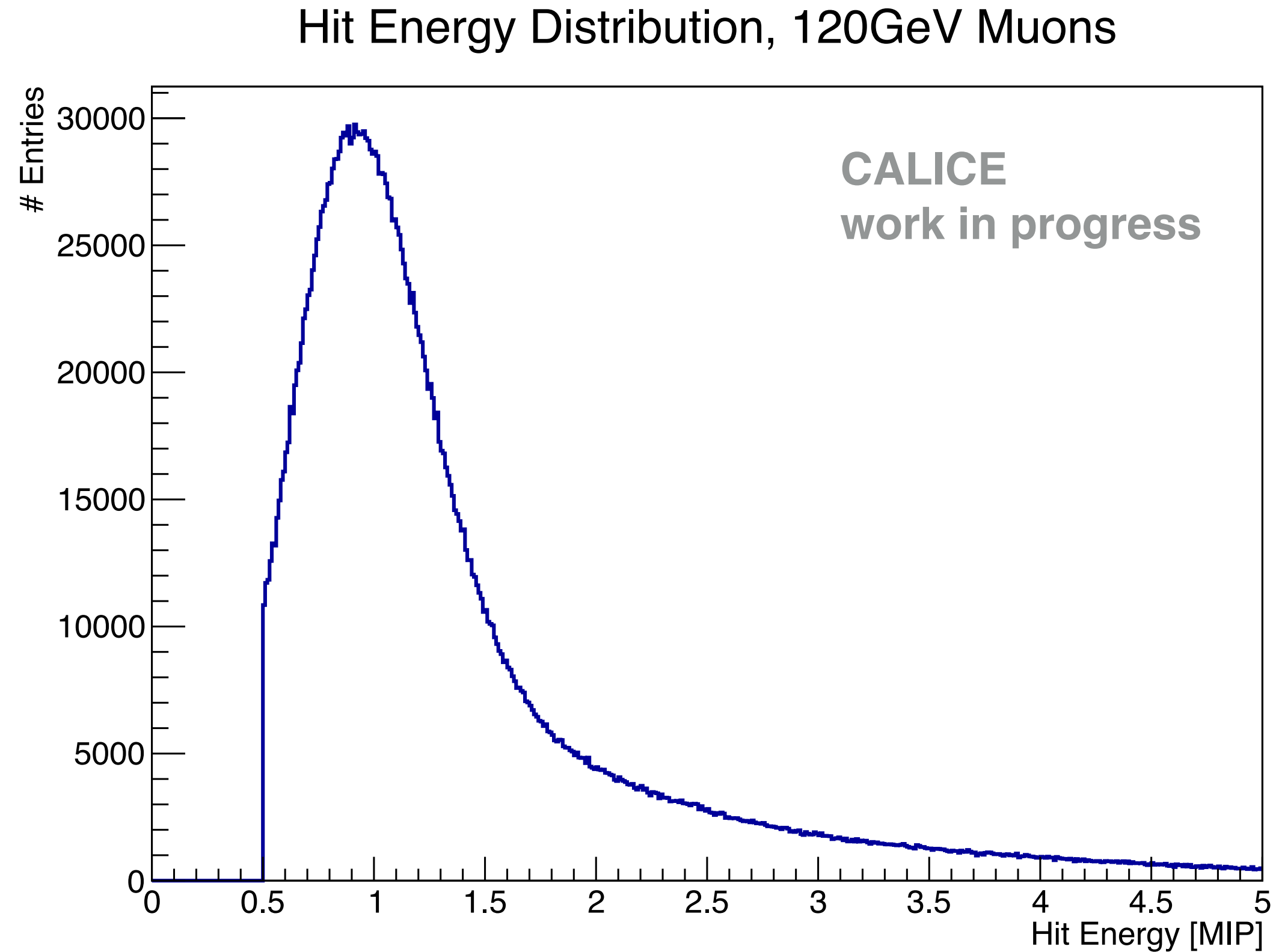
master	...	labview_converter / raw2lcio / src /
..		
CMakeLists.txt	Cleaned up	
EUDAQEventBuilder.cc	Cleaned up	
EUDAQEventBuilder2016.cc	Allowing for	
EUDAQEventBuilder2016_woBIF.cc	Comment o	
EUDAQEventBuilder2018_cosmics.cc	Comment o	
EventChecker.cc	Cleaned up	
LabviewConverter.cc	Cleaned up	
LabviewConverter2.cc	Corrected b	
RootTreeGenerator.cc	Cleaned up	
RootTreeGenerator2.cc	Cleaned up	
RootTreeGenerator3.cc	Cleaned up	
RootTreeGeneratorEUDAQ2016.cc	Modified to	
TempRootTreeGenerator.cc	Cleaned up	

Overview - Software Workflow



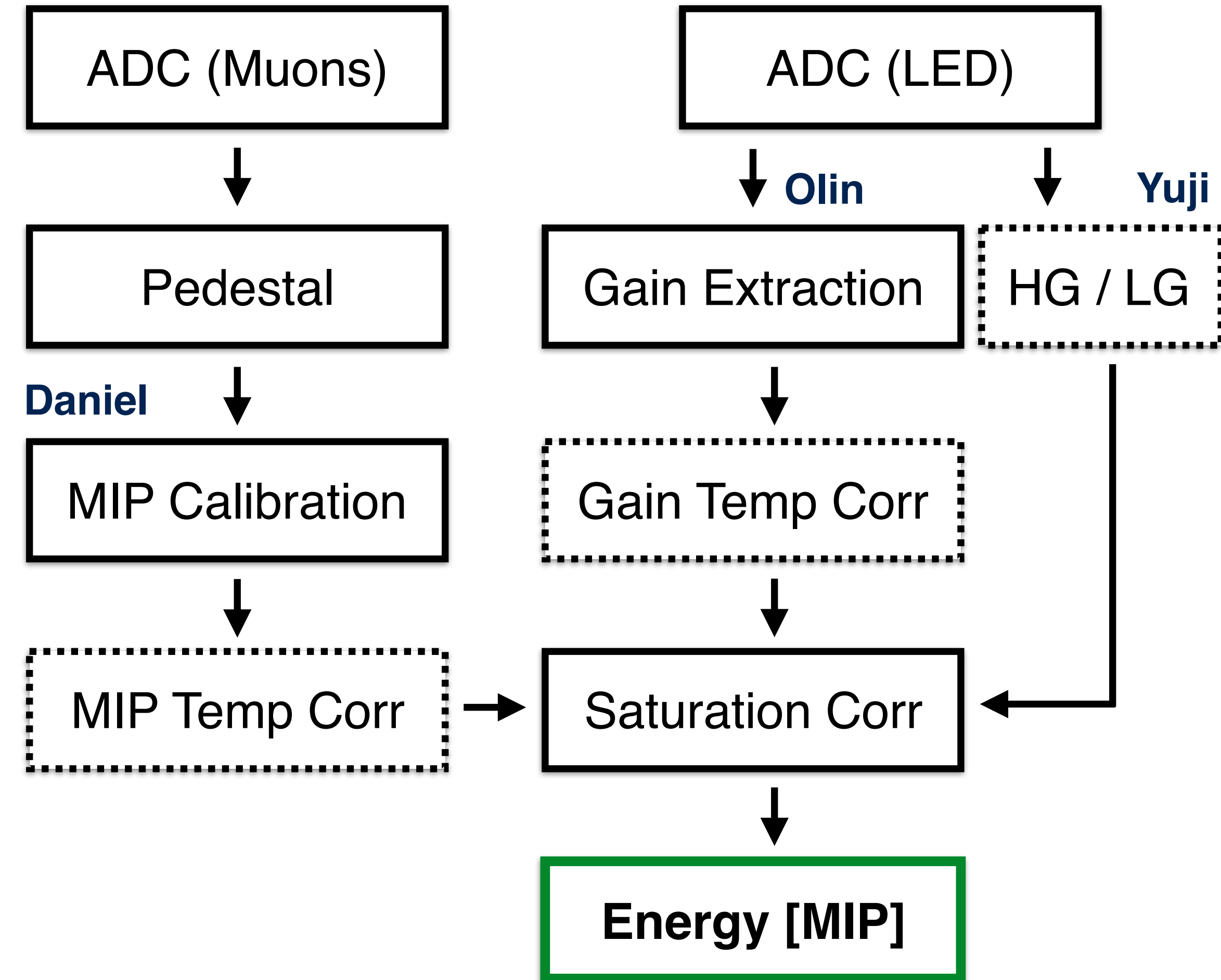
- Data and simulation are processed by the same pipeline

Energy / Time Reconstruction



[calice_reco/recoSIPM/Ahc2CalibrateProcessor]

$$E_{\text{calibrated}} = \frac{f_{\text{saturation}} [(ADC - \text{Pedestal}) * IC / \text{Gain}]}{IC / \text{Gain} * \text{MIP}}$$



$$t[\text{ns}] = \text{TDC} * \text{slope} + \text{Offset} - T_{\text{reference}} \quad \text{Lorenz}$$

How to work together?



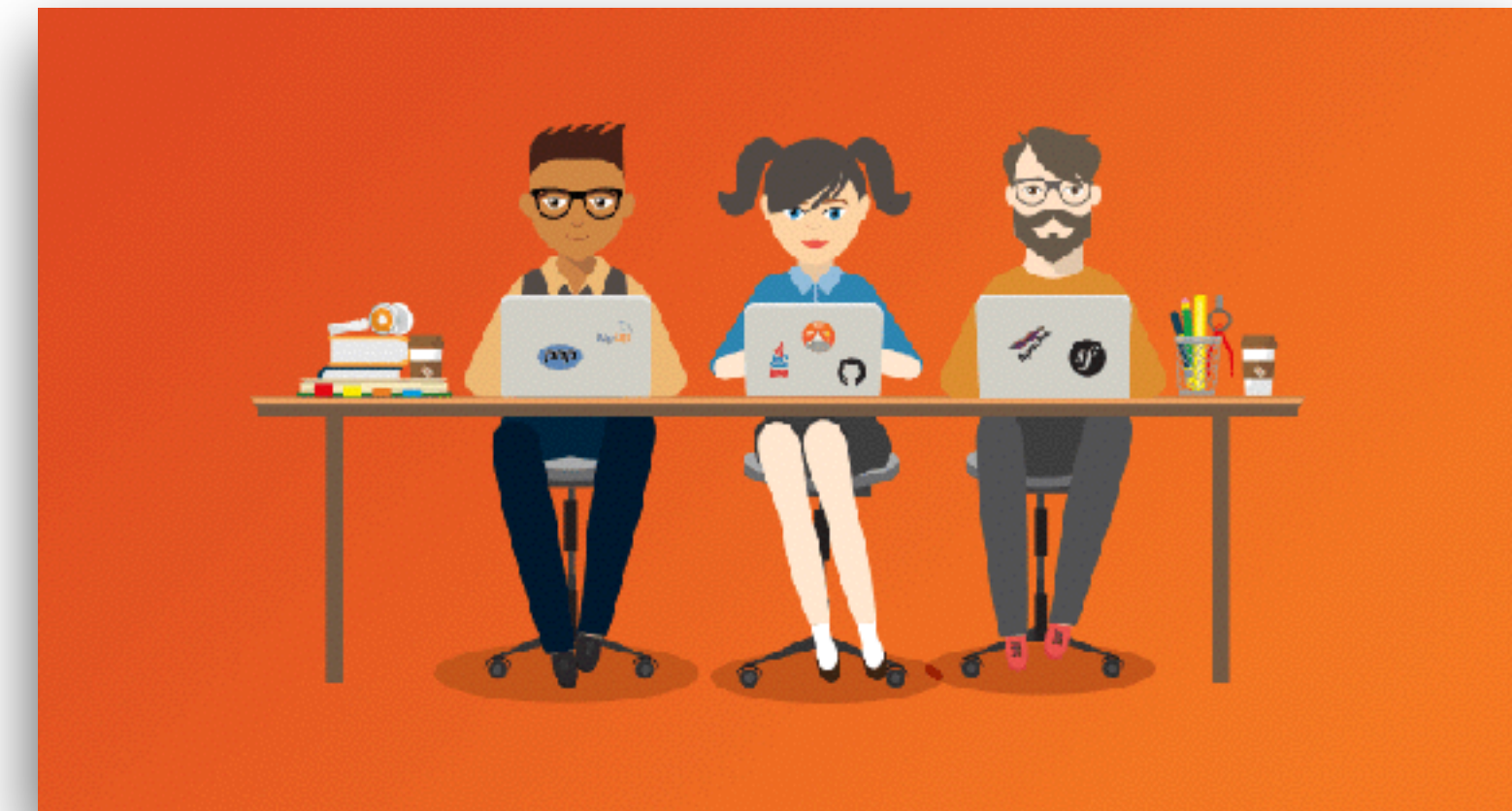
- Working with several people on the same software imposes some challenges:
- How to share the code? **SVN**
- How to effectively develop features / fix bugs? **Problematic**
- How to maintain several versions of the software **Problematic**
- How to ensure working code while developing? **Problematic**
- How to track issues (bugs / missing features)? **Pen & Paper**
- How to organize documentation? **Documentation?**
- How to guarantee a working software? **Hope for the best**



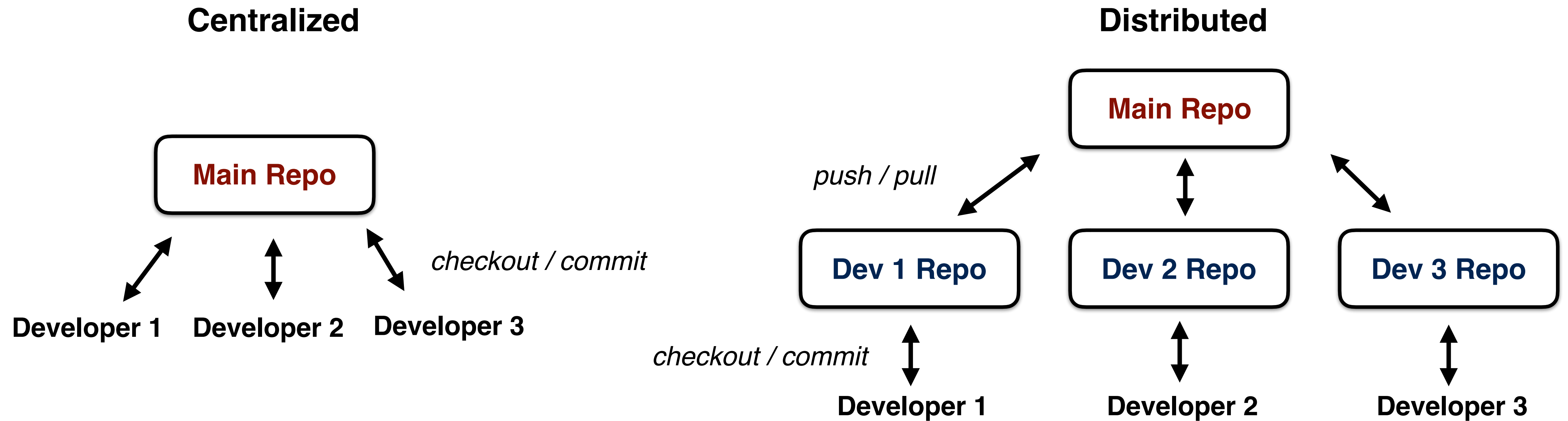
How to work together?



- Working with several people on the same software imposes some challenges:
- How to share the code? **Git**
- How to effectively develop features / fix bugs? **Branching**
- How to maintain several versions of the software **Branching**
- How to ensure working code while developing? **Pull requests**
- How to track issues (bugs / missing features)? **Issue tracker**
- How to organize documentation? **Confluence**
- How to guarantee a working software? **Continuous integration**



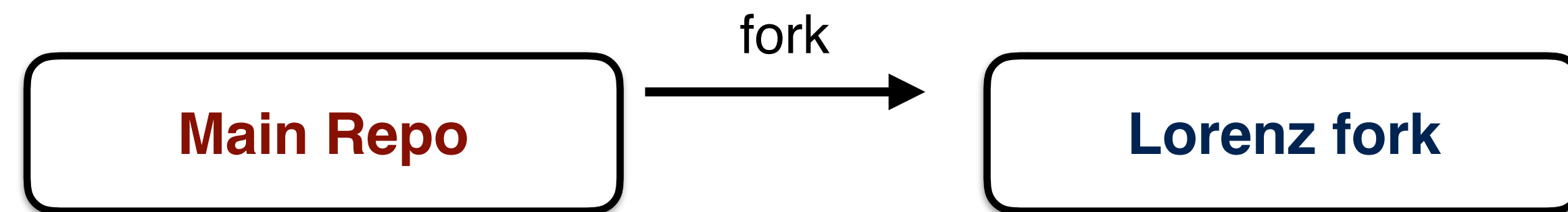
Centralized vs Distributed Version Control



- In a distributed version control system everybody has a full copy of the repositories history
- This allows for more complex workflows

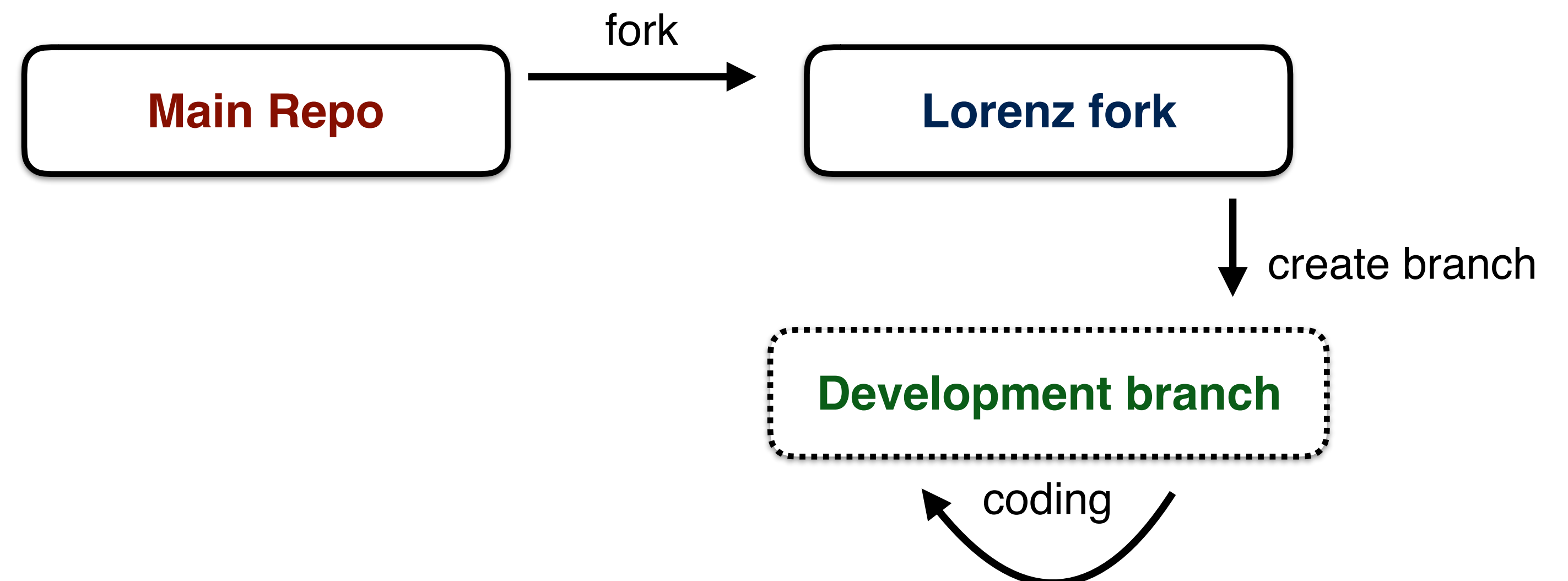
Let's say Lorenz wants to implement an amazing feature

1. He creates **his personal fork** (a copy of the repo on the server)
2. He creates a **separate branch** on his fork for the development
3. He merges his **development** to his **master branch**
4. He creates a pull request for the **main repo**
5. The pull request will be merged in the **main repo**



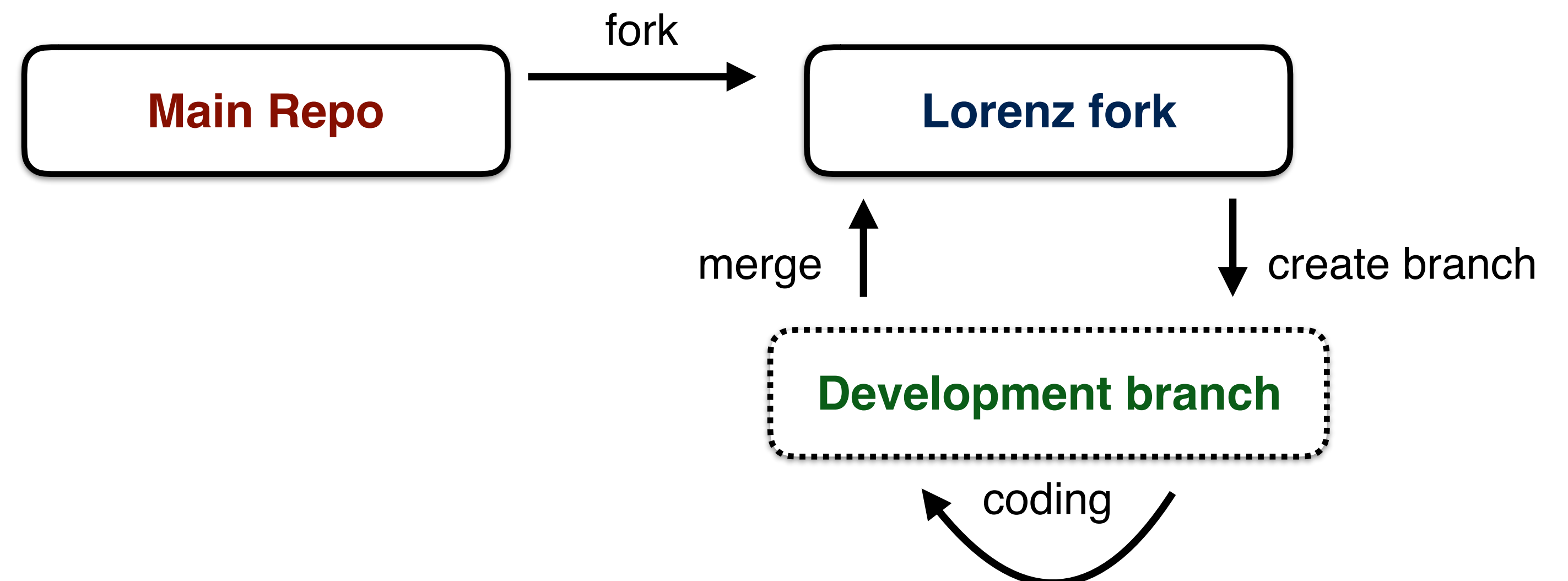
Let's say Lorenz wants to implement an amazing feature

1. He creates **his personal fork** (a copy of the repo on the server)
2. He creates a **separate branch** on his fork for the development
3. He merges his **development** to his **master branch**
4. He creates a pull request for the **main repo**
5. The pull request will be merged in the **main repo**



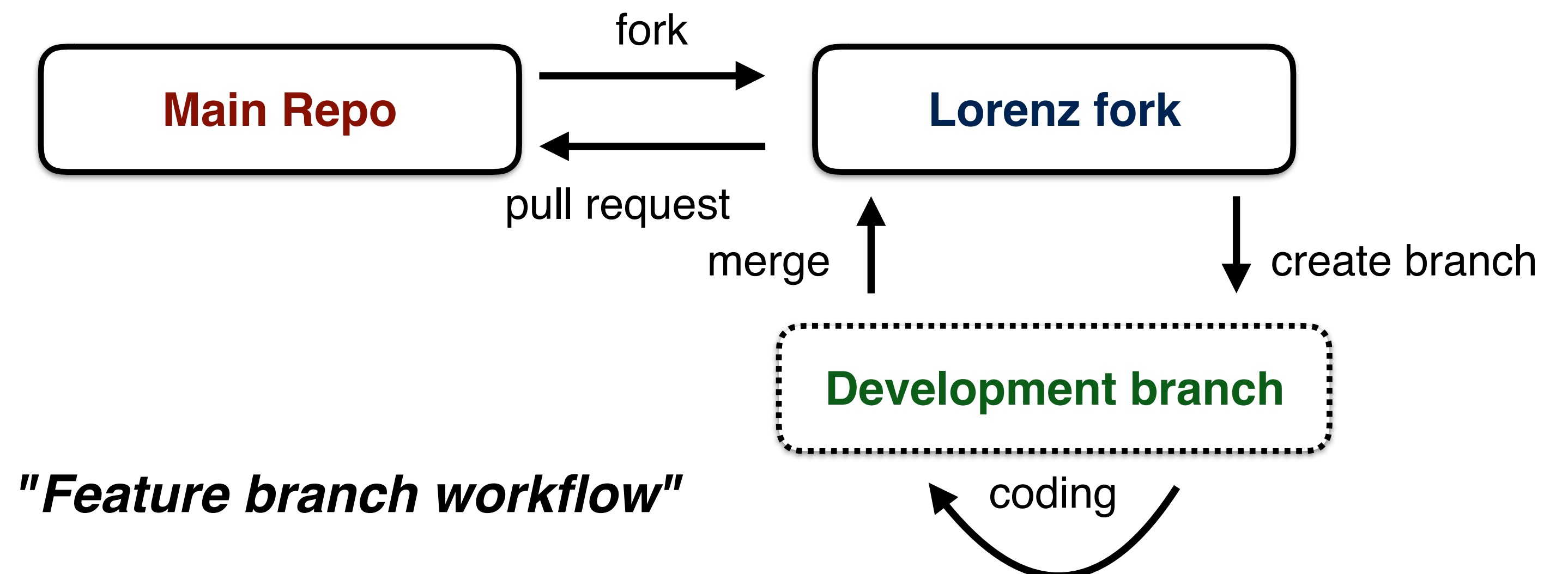
Let's say Lorenz wants to implement an amazing feature

1. He creates **his personal fork** (a copy of the repo on the server)
2. He creates a **separate branch** on his fork for the development
3. He merges his **development** to his **master branch**
4. He creates a pull request for the **main repo**
5. The pull request will be merged in the **main repo**



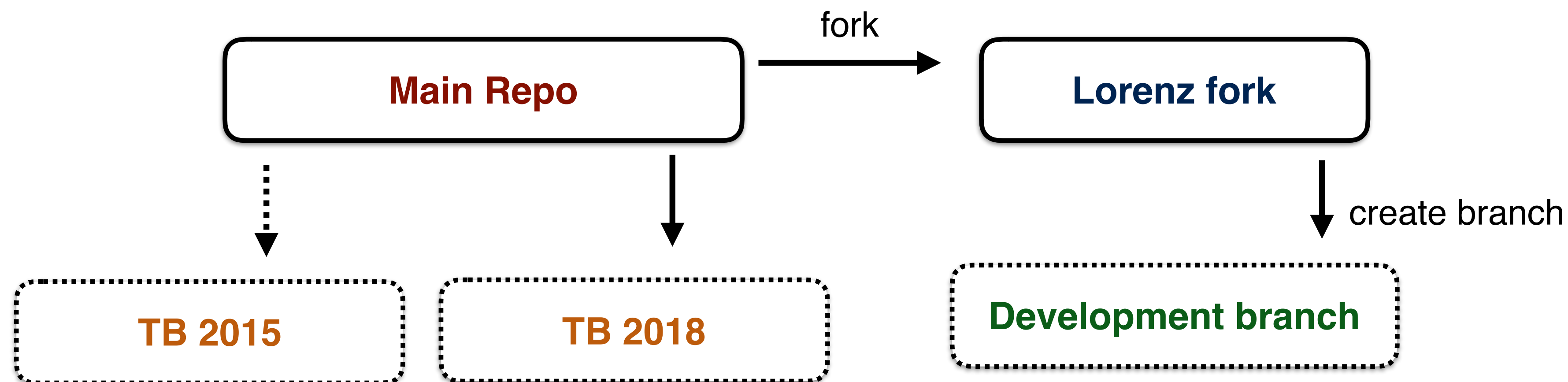
Let's say Lorenz wants to implement an amazing feature

1. He creates **his personal fork** (a copy of the repo on the server)
2. He creates a **separate branch** on his fork for the development
3. He merges his **development** to his **master branch**
4. He creates a pull request for the **main repo**
5. The pull request will be merged in the **main repo**



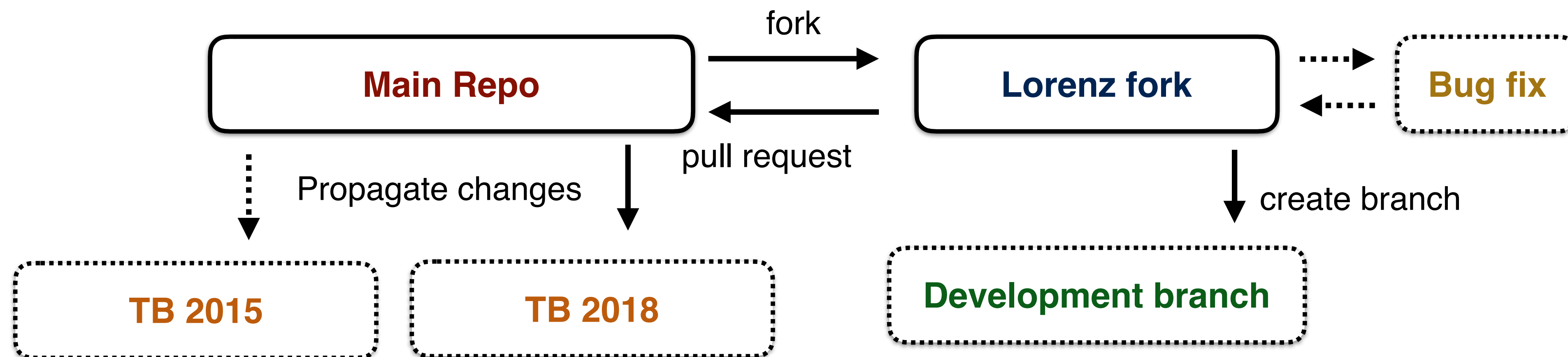
Why the hassle?

- The **main repo** is always in a production version ready to analyze data
- The **main repo** may have **different branches** e.g., for different test beam campaigns
- While working on his **nice feature**, Lorenz gets a mail, that he urgently has to fix a bug
- He switches back to his **master branch**, creates a **new branch**, fixes the bug, merges to the **master**, creates a pull request for the **main repo** and continues to work on **his feature**



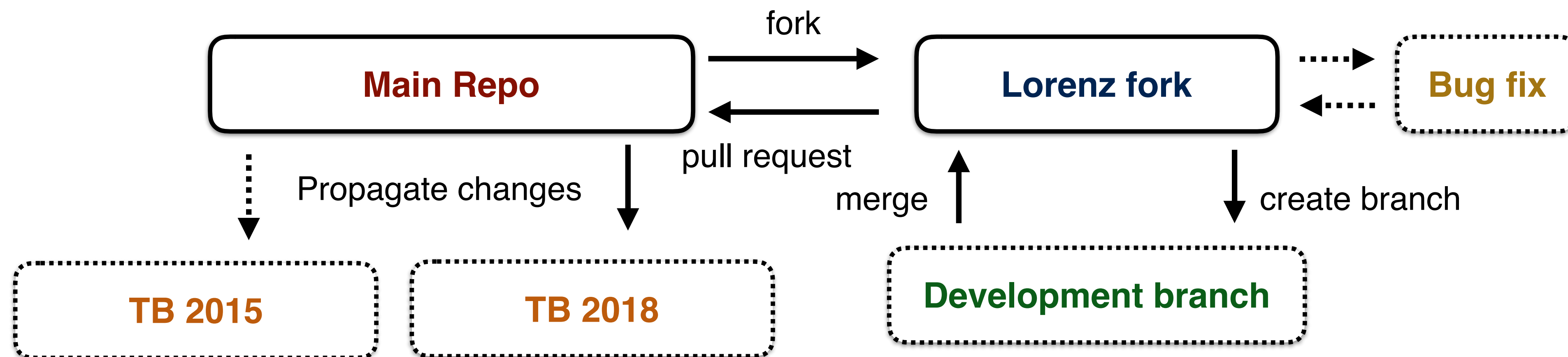
Why the hassle?

- The **main repo** is always in a production version ready to analyze data
- The **main repo** may have **different branches** e.g., for different test beam campaigns
- While working on his **nice feature**, Lorenz gets a mail, that he urgently has to fix a bug
- He switches back to his **master branch**, creates a **new branch**, fixes the bug, merges to the **master**, creates a pull request for the **main repo** and continues to work on **his feature**









Why the hassle?

- The **main repo** is always in a production version ready to analyze data
- The **main repo** may have **different branches** e.g., for different test beam campaigns
- While working on his **nice feature**, Lorenz gets a mail, that he urgently has to fix a bug
- He switches back to his **master branch**, creates a **new branch**, fixes the bug, merges to the **master**, creates a pull request for the **main repo** and continues to work on **his feature**



- Basic commands on confluence
- Many online tutorials

Seiten / CALICE / CALICE Software    Bearbeiten  Favorit  Beobachtung  Teilen ...

How to use git?


Eldwan Brianne posted on 11. Jul. 2018 11:41h - last edited by Eldwan Brianne on 16. Jul. 2018 18:36h


For people not familiar with git.

What is the advantage of Git over another version control system, e.g SVN?

In short terms, git is a distributed version control system (compared to svn which is a centralised system). In this way, each developer has a their own local repository, complete with a full history of commits, instead of a working copy. This also has the advantage that the production branch cannot be broken, each developer work on their own local repository. This creates a more reliable environment.

A detailed explanation can be seen [here](#).

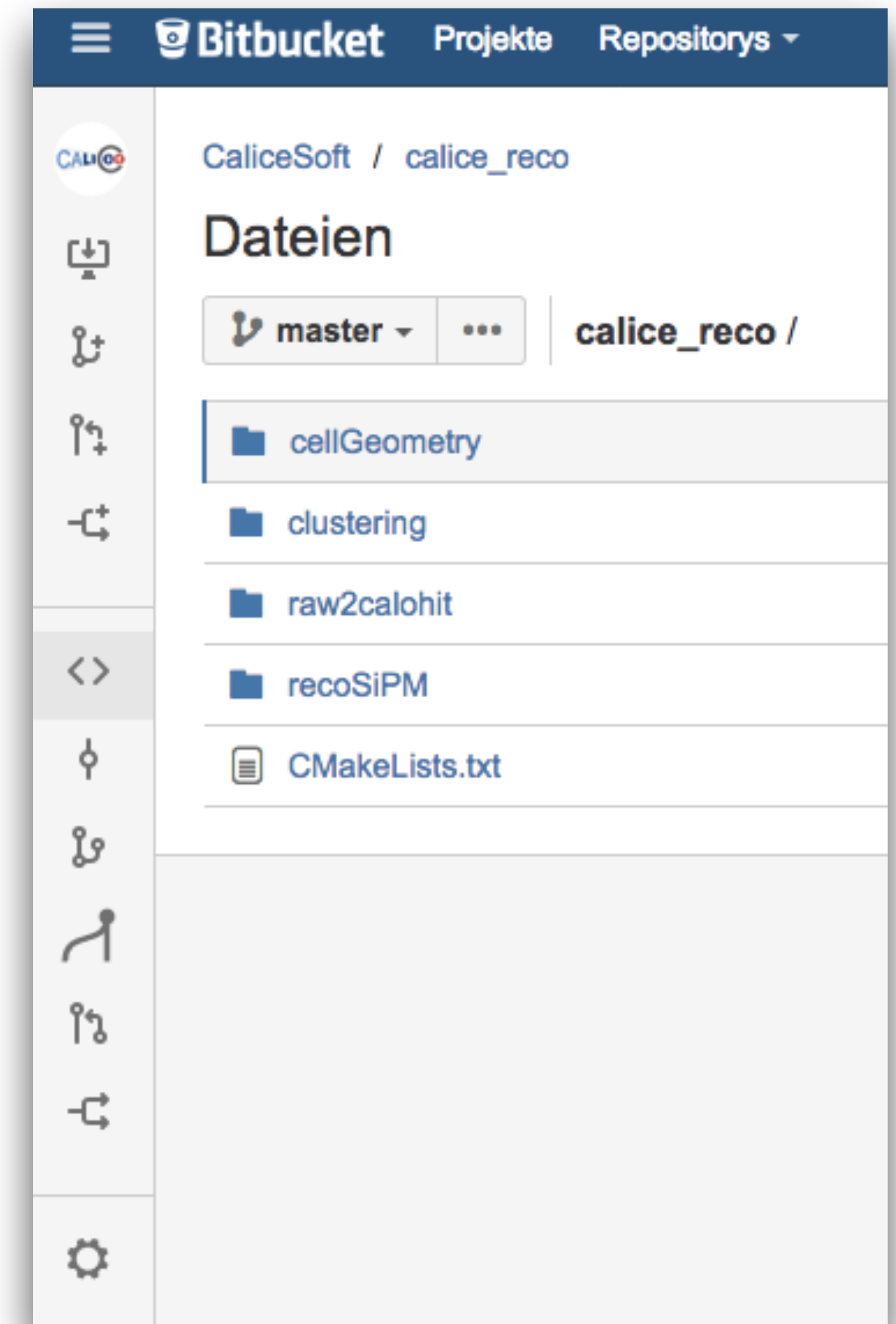
 These are the most simple commands. For more details, please consult the [git documentation](#).

 To be able to do pull requests on stash, access with your DESY account need to be added. Please request it at it-atlassian@desy.de.

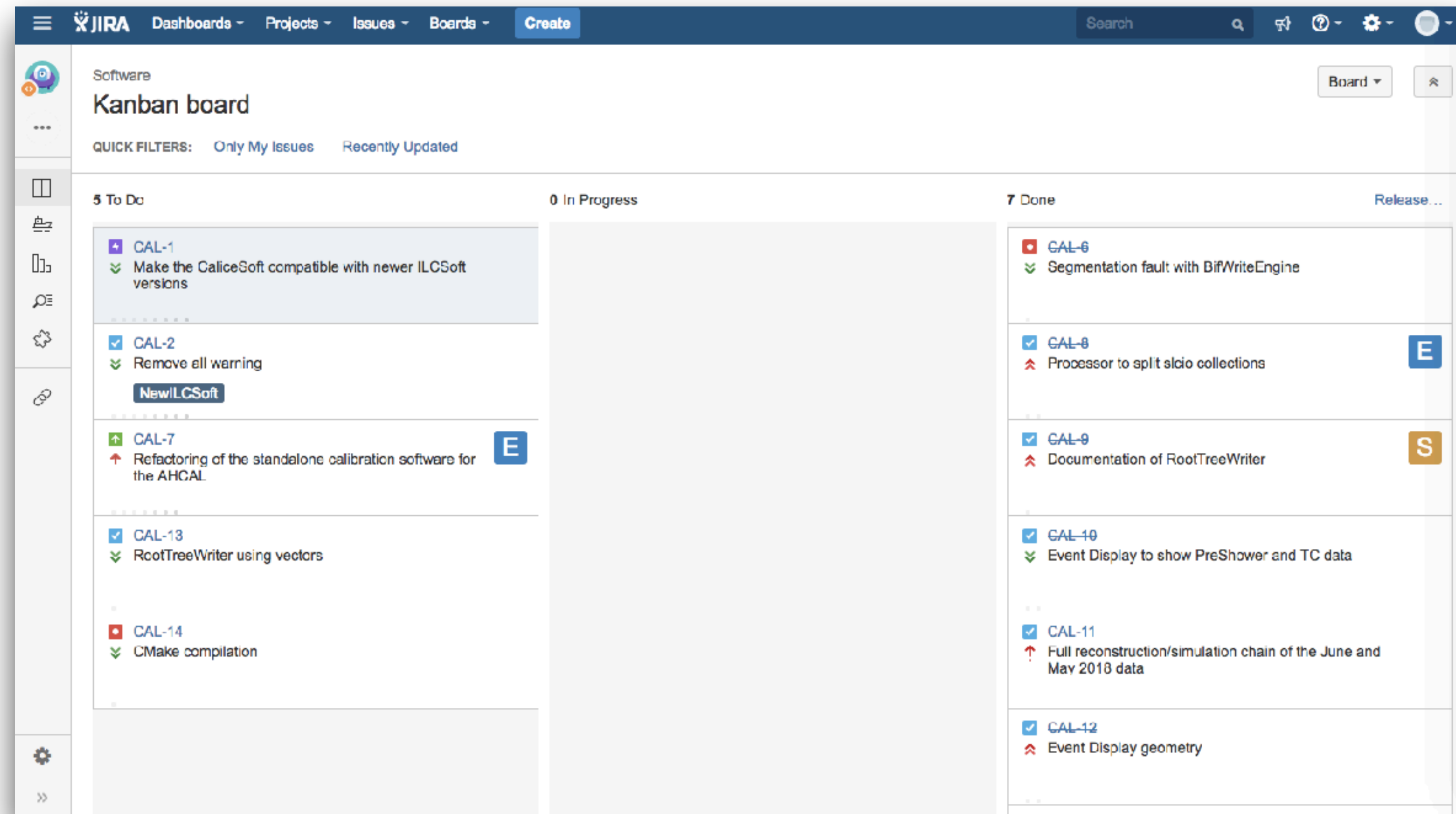
- [Git configuration](#)
- [Cloning a repository](#)
- [Workflow](#)
 - [Make a fork](#)
 - [Add a downstream remote pointing to your fork](#)
 - [Working, updating](#)
 - [Staging and commit](#)
 - [Before a pull request](#)



- CALICE Soft on Atlassian Stash (Thanks to Eldwan!)
- Git repository hosted on DESY Servers
- Fork - Pull Request workflow
- Easy branching
- Integration with other DESY Atlassian tools

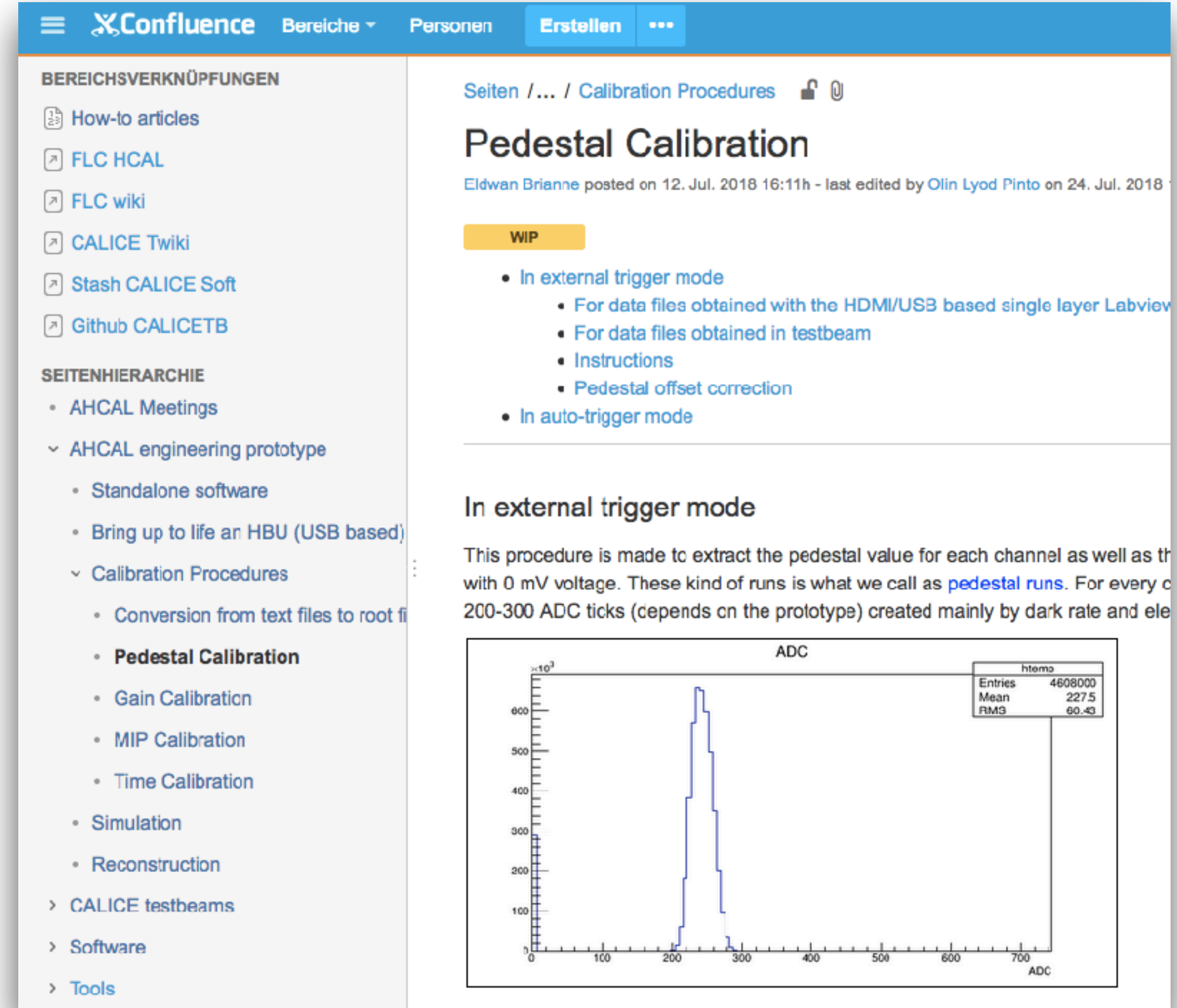


- Issue tracker
- Allows to list bugs / missing features
- Keeps track on what is currently being worked on
- Responsible persons can be assigned



We need documentation on several layers

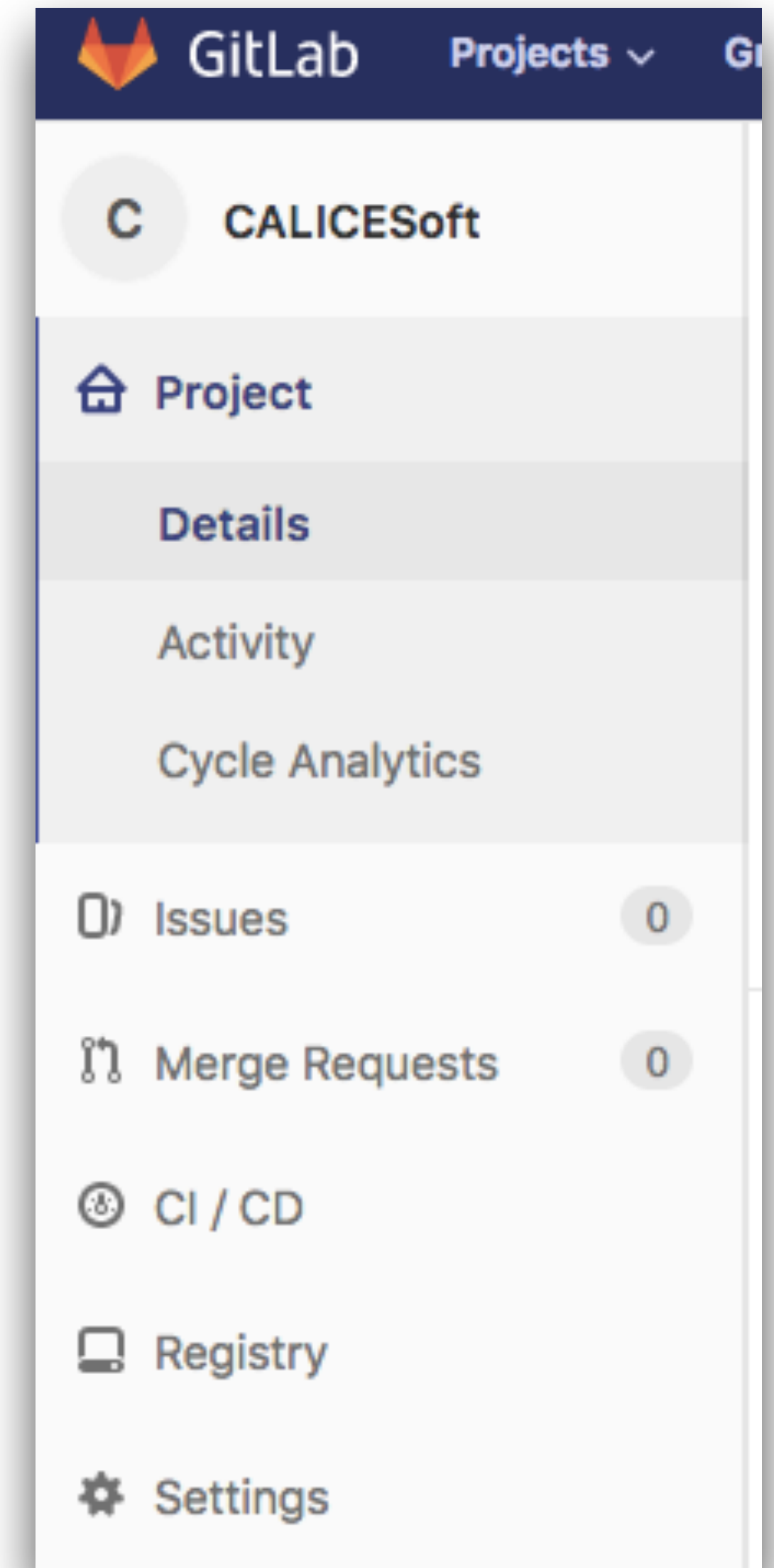
1. Comments to explain non trivial parts of the code
2. Each processor should have basic documentation on what it does, what input parameter it needs and what the output is
3. Each calice package / subfolder should have a README that clearly states what the scope of this package is
4. High level documentation on e.g. how to install the software, how to perform a pedestal calibration ...
—> Confluence



The screenshot shows a Confluence page titled "Pedestal Calibration". The left sidebar contains a navigation menu with sections like "BEREICHsverknüpfungen" (How-to articles, FLC HCAL, FLC wiki, CALICE Twiki, Stash CALICE Soft, Github CALICETB) and "SEITENHIERARCHIE" (AHCAL Meetings, AHCAL engineering prototype, Standalone software, Bring up to life an HBU (USB based), Calibration Procedures, Conversion from text files to root files, Pedestal Calibration, Gain Calibration, MIP Calibration, Time Calibration, Simulation, Reconstruction, CALICE testbeams, Software, Tools). The main content area shows the page title "Pedestal Calibration", a "WIP" status, and a list of topics: "In external trigger mode" (For data files obtained with the HDMI/USB based single layer Labview, For data files obtained in testbeam, Instructions, Pedestal offset correction) and "In auto-trigger mode". Below this, there is a section "In external trigger mode" with a description: "This procedure is made to extract the pedestal value for each channel as well as the pedestal offset with 0 mV voltage. These kind of runs is what we call as pedestal runs. For every channel 200-300 ADC ticks (depends on the prototype) created mainly by dark rate and electronics noise." At the bottom, there is a histogram plot titled "ADC" showing a distribution of ADC values. The x-axis is labeled "ADC" and ranges from 0 to 700. The y-axis is labeled "Entries" and ranges from 0 to 600. The histogram shows a sharp peak around 227.5. A table in the top right corner of the plot area provides summary statistics: Entries: 4608000, Mean: 227.5, RMS: 60.43.

ADC	
Entries	4608000
Mean	227.5
RMS	60.43

- Github - data protection concerns
- CERN Gitlab
- CERN Computing account required
- Git Repository / Issue Tracker / CI in one place
- Straight forward to use



- Complex software as a base for every analysis
- Several new tools may help us to effectively work together
- Analysis workshop is the perfect environment to try out the tools and give feedback

Happy Coding!

