



UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386



GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung



Hands-On: Corryvreckan

The Maelstrom for Your Test Beam Data



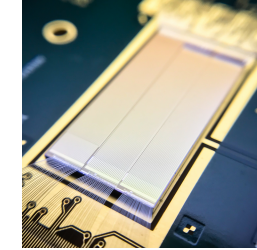
9th BTTB Workshop (virtual)
9th February 2021

Jens Kröger, Heidelberg University & CERN
on behalf of the Corryvreckan Developers

What are we going to learn today?

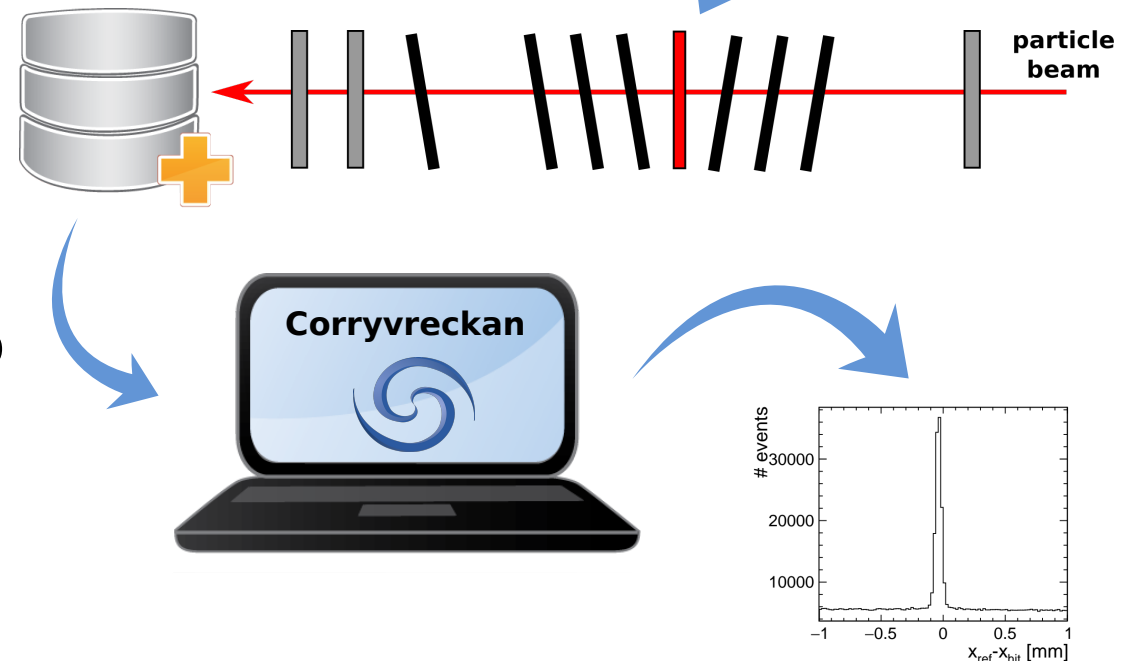
- **Introduction**

- What is Corryvreckan?
- What's new since BTTB8?



- **Hands-on:**

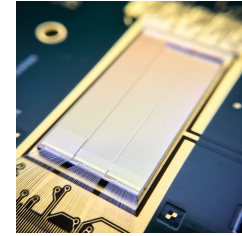
- installation options
- setting up an analysis step-by-step



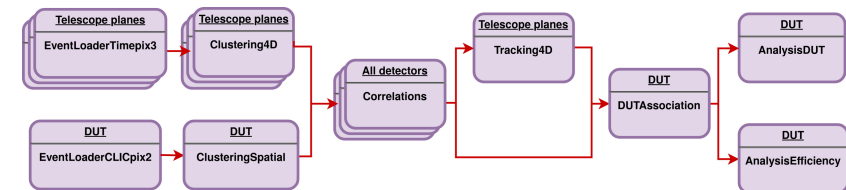
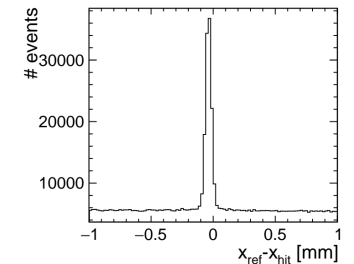
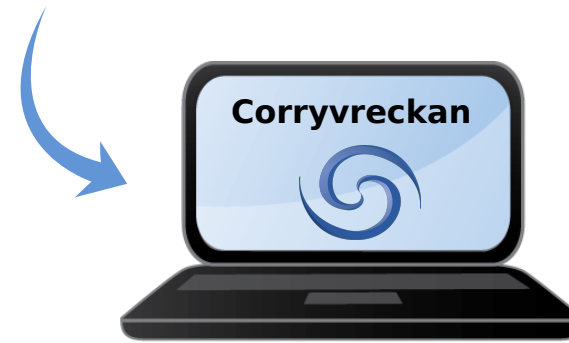
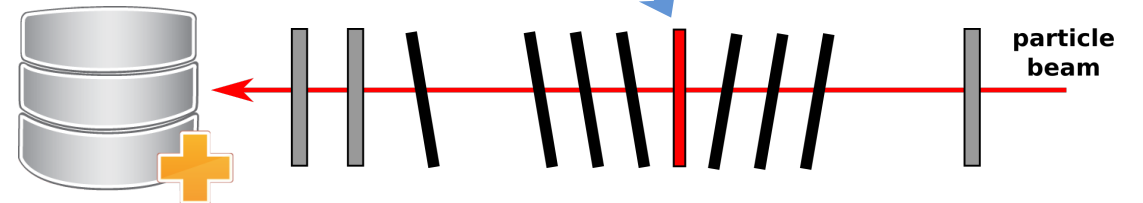
- Please ask questions at any time!

What is Corryvreckan?

A **reconstruction and analysis** tool for pixel sensor test-beam data

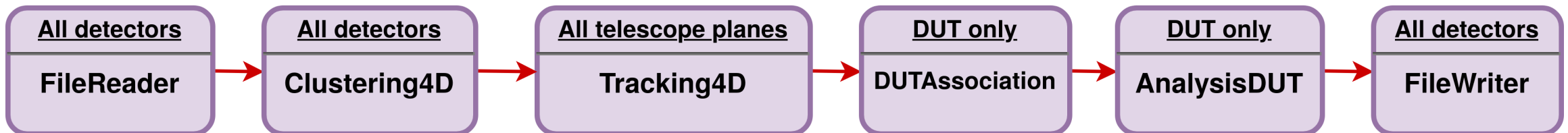


- modular structure
 - framework core
 - modules for specific tasks
- highly flexible and configurable
- easy to understand
 - written in modern C++
 - comprehensive documentation (> 120 pages!)



The Modular Approach

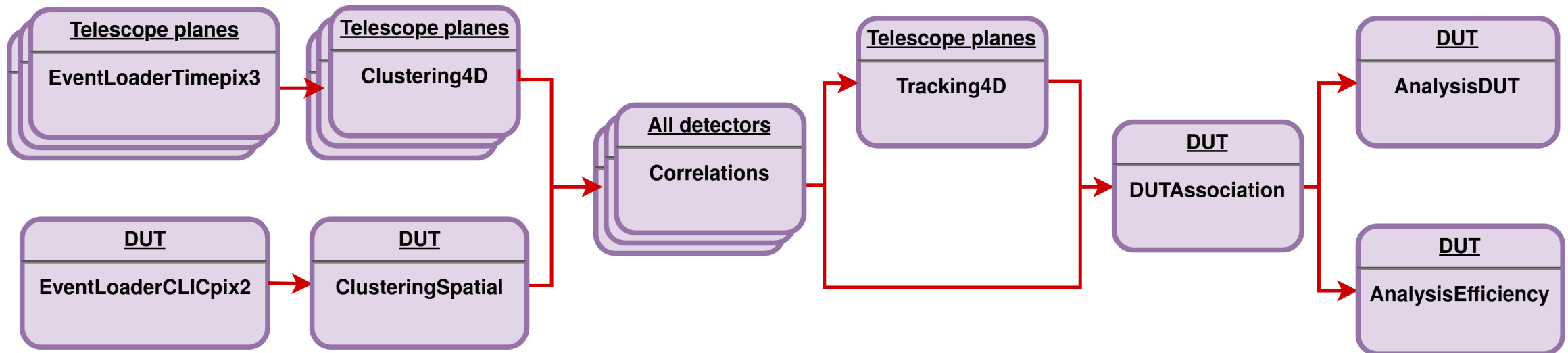
- modular structure:
 - framework core
 - implementation of algorithms → **[Modules]**
- modules:
(user) algorithms for specific tasks
- objects are stored temporarily:
 - events, pixels, clusters, tracks
- select suitable modules for
 - event building
 - clustering
 - tracking
 - analysis (also multiple DUTs)
 - ...
- quick to set up and easy to configure



The Modular Approach

- can create more **complex reconstruction chains**
- apply **different modules to different devices** in the same reconstruction

We will build this reco chain in **Example 2**:

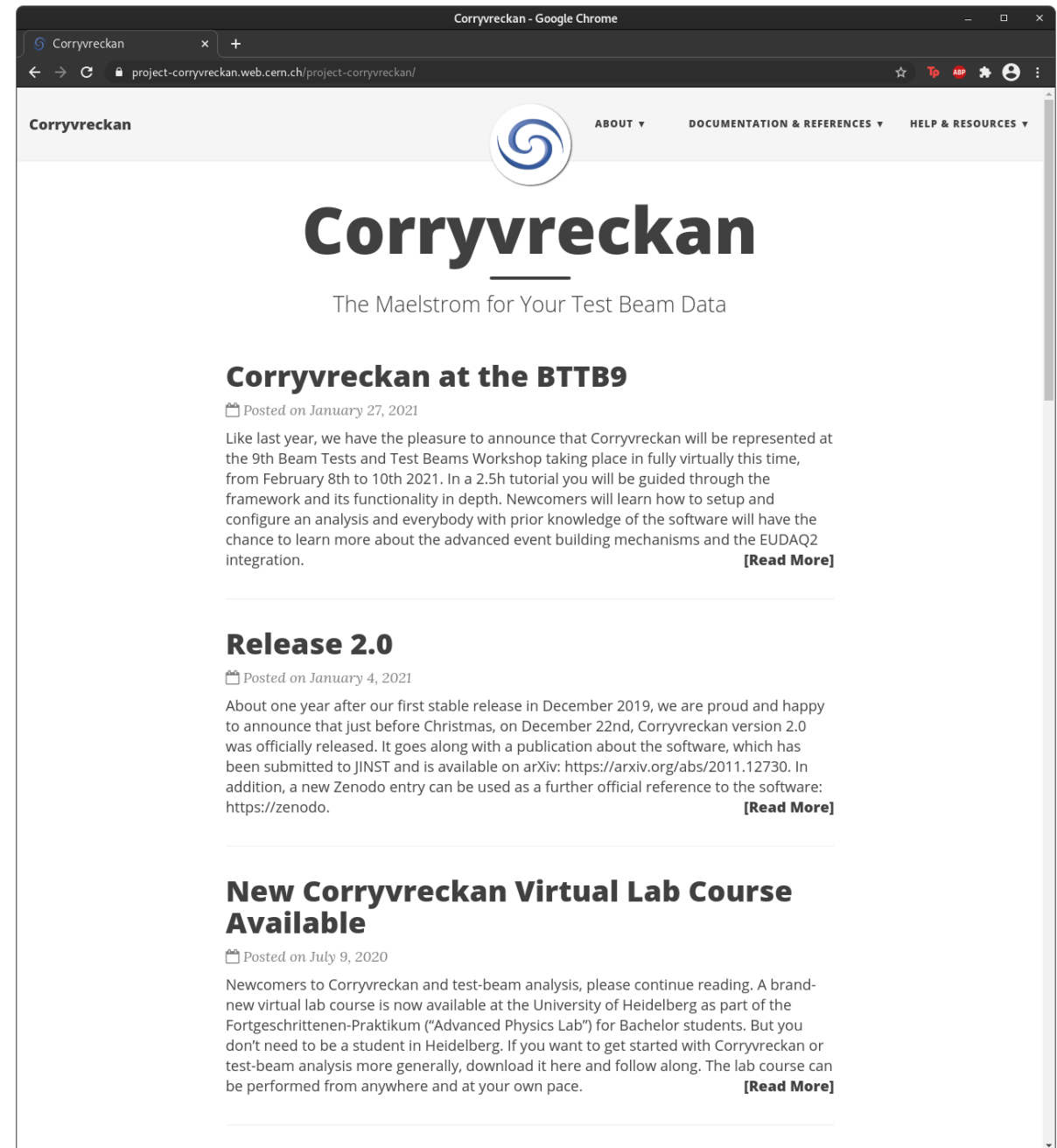


Project Website

First place to go:

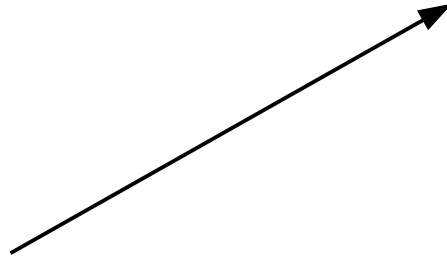
<https://cern.ch/corryvreckan>

- News on releases
- Installation/Getting Started
- **Links:**
 - code repository
 - issue tracker
 - forum
 - ...



Documentation

- **online documentation** in repo
<https://gitlab.cern.ch/corryvreckan/corryvreckan>
 - every modules has a README
- **extensive user manual**
<https://cern.ch/corryvreckan/usermanual/corryvreckan-manual-v2.0.pdf>
 - full description of framework
 - installation instructions
 - “Getting started”, FAQs
 - module descriptions (fetched from repo)
- **Doxygen code reference**
<https://cern.ch/corryvreckan/reference/>
 - more details on code



Corryvreckan > Corryvreckan > Repository

master corryvreckan / src / modules / Clustering4D / +

History Find file Web IDE

Fix mixup of coordinates
Simon Spannagel authored 1 month ago cab17a6d

Name	Last commit	Last update
..		
CMakeLists.txt	Rename Timepix3Clustering -> Clustering4D	1 year ago
Clustering4D.cpp	Spatial cuts: backwards compatibility for older names, print...	1 month ago
Clustering4D.h	Implementing simultaneous relative and absolute time cuts ...	2 months ago
README.md	Fix mixup of coordinates	1 month ago

README.md

Clustering4D

Maintainer: Daniel Hynds (daniel.hynds@cern.ch)
Module Type: DETECTOR
Detector Type: all
Status: Functional

Description

This module performs clustering for detectors with valid individual hit timestamps. The clustering method is either an arithmetic mean or a charge-weighted centre-of-gravity calculation, using a positional cut and a timing cut on proximity. If the pixel information is binary (i.e. no valid charge-equivalent information is available), the arithmetic mean is calculated for the position. Also, if one pixel of a cluster has charge zero, the arithmetic mean is calculated even if charge-weighting is selected because it is assumed that the zero-reading is false and does not represent a low charge but an unknown value. Thus, the arithmetic mean is safer.

Split clusters can be recovered using a larger search radius for neighbouring pixels.

Parameters

- `time_cut_rel`: Number of standard deviations the `time_resolution` of the detector plane will be multiplied by. This value is then used as the maximum time difference allowed between pixels for association to a cluster. By default, a relative time cut is applied. Absolute and relative time cuts are mutually exclusive. Defaults to `3.0`.
- `time_cut_abs`: Specifies an absolute value for the maximum time difference allowed between pixels for association to a cluster. Absolute and relative time cuts are mutually exclusive. No default value.
- `neighbour_radius_col`: Search radius for neighbouring pixels in column direction, defaults to `1` (do not allow split clusters)
- `neighbour_radius_row`: Search radius for neighbouring pixels in row direction, defaults to `1` (do not allow split clusters)
- `charge_weighting`: If true, calculate a charge-weighted mean for the cluster centre. If false, calculate the simple arithmetic mean. Defaults to `true`.

Plots produced

For each detector the following plots are produced:

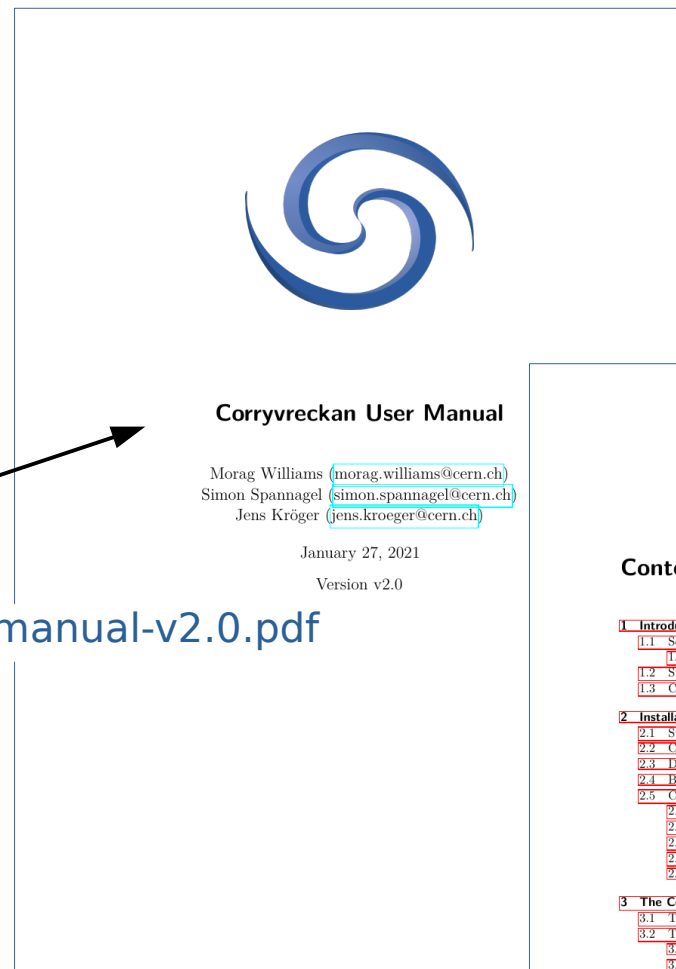
- Histograms for cluster size, seed charge, width (columns/X and rows/Y)
- Cluster charge histogram
- 2D cluster positions in global coordinates
- Cluster times
- Cluster multiplicity

Usage

```
[Clustering4D]
time_cut_rel = 3.0
```

Documentation

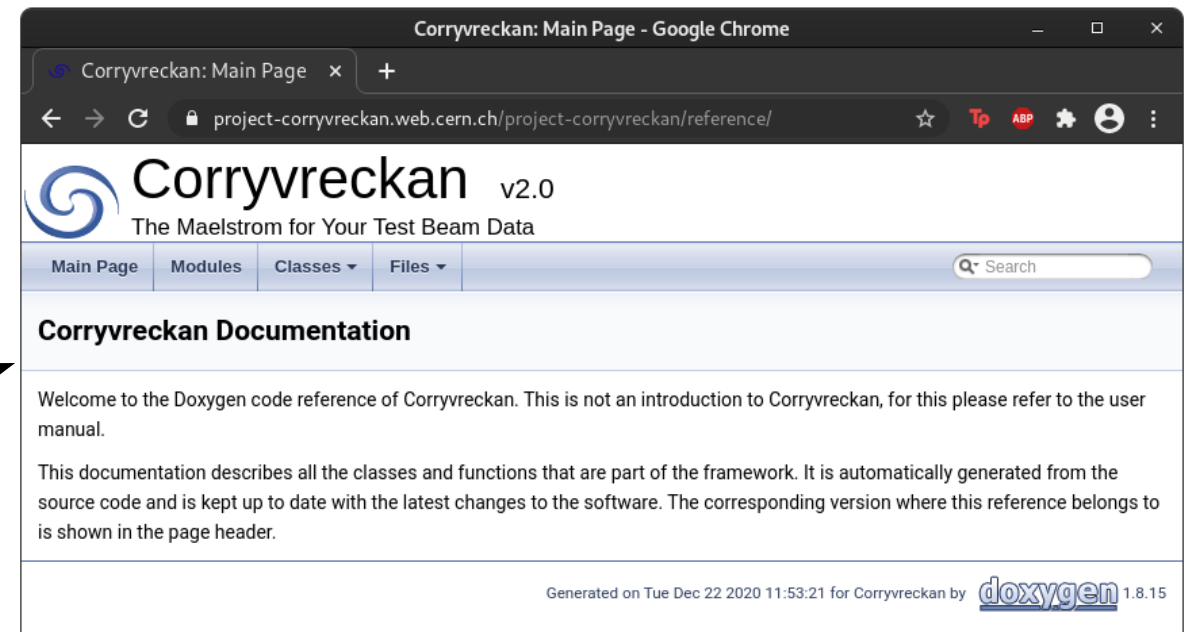
- online documentation in repo
<https://gitlab.cern.ch/corryvreckan/corryvreckan>
 - every modules has a README
- **extensive user manual**
<https://cern.ch/corryvreckan/usermanual/corryvreckan-manual-v2.0.pdf>
 - full description of framework
 - installation instructions
 - “Getting started”, FAQs
 - module descriptions (fetched from repo)
- Doxygen code reference
<https://cern.ch/corryvreckan/reference/>
 - more details on code



Contents	
1 Introduction	1
1.1 Scope of this Manual	1
1.1.1 Getting Started	2
1.2 Support and Reporting Issues	2
1.3 Contributing Code	2
2 Installation	3
2.1 Supported Operating Systems	3
2.2 CMVFS	3
2.3 Docker	3
2.4 Binaries	4
2.5 Compilation from Source	4
2.5.1 Prerequisites	5
2.5.2 Downloading the source code	5
2.5.3 Configuration via CMake	5
2.5.4 Compilation and installation	6
2.5.5 macOS	7
3 The Corryvreckan Framework	9
3.1 The corry Executable	9
3.2 The Clipboard	10
3.2.1 The Event	11
3.2.2 Temporary Data Storage	11
3.2.3 Persistent Storage	11
3.3 Global Framework Parameters	12
3.4 Modules and the Module Manager	13
3.4.1 Module Status Codes	13
3.4.2 Execution Order	14
3.4.3 Module instantiation	14
3.5 Logging and Verbosity Levels	15
3.6 Coordinate Systems	17
4 Configuration Files	19
4.1 Parsing types and units	19
4.1.1 File format	22
4.1.2 Accessing parameters	23
4.2 Main configuration	24
4.3 Detector configuration	25
4.3.1 Masking Pixels Offline	28

Documentation

- online documentation in repo
<https://gitlab.cern.ch/corryvreckan/corryvreckan>
 - every modules has a README
- extensive user manual
<https://cern.ch/corryvreckan/usermanual/corryvreckan-manual-v2.0.pdf>
 - full description of framework
 - installation instructions
 - “Getting started”, FAQs
 - module descriptions (fetched from repo)
- **Doxygen code reference**
<https://cern.ch/corryvreckan/reference/>
 - more details on code

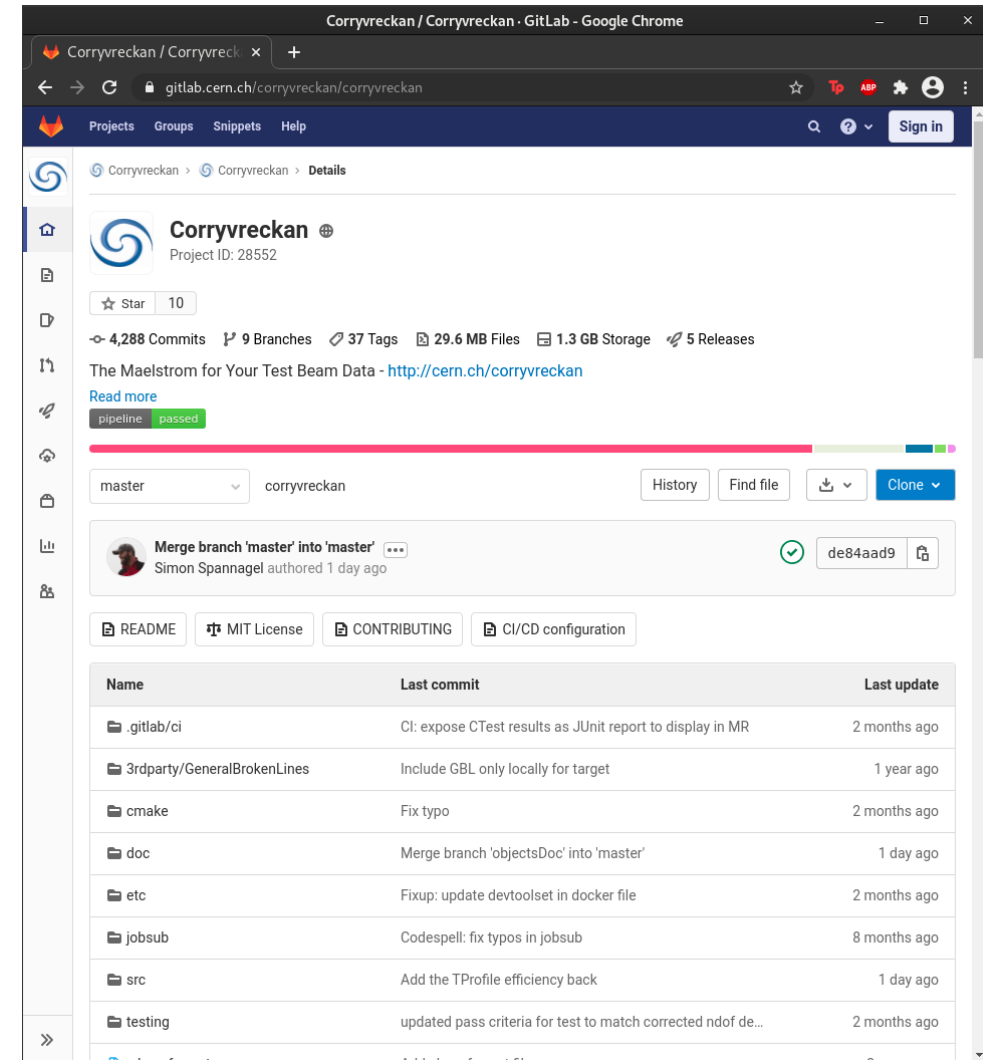


The Corryvreckan Repository

Let's explore the **repository**

<https://gitlab.cern.ch/corryvreckan/corryvreckan>

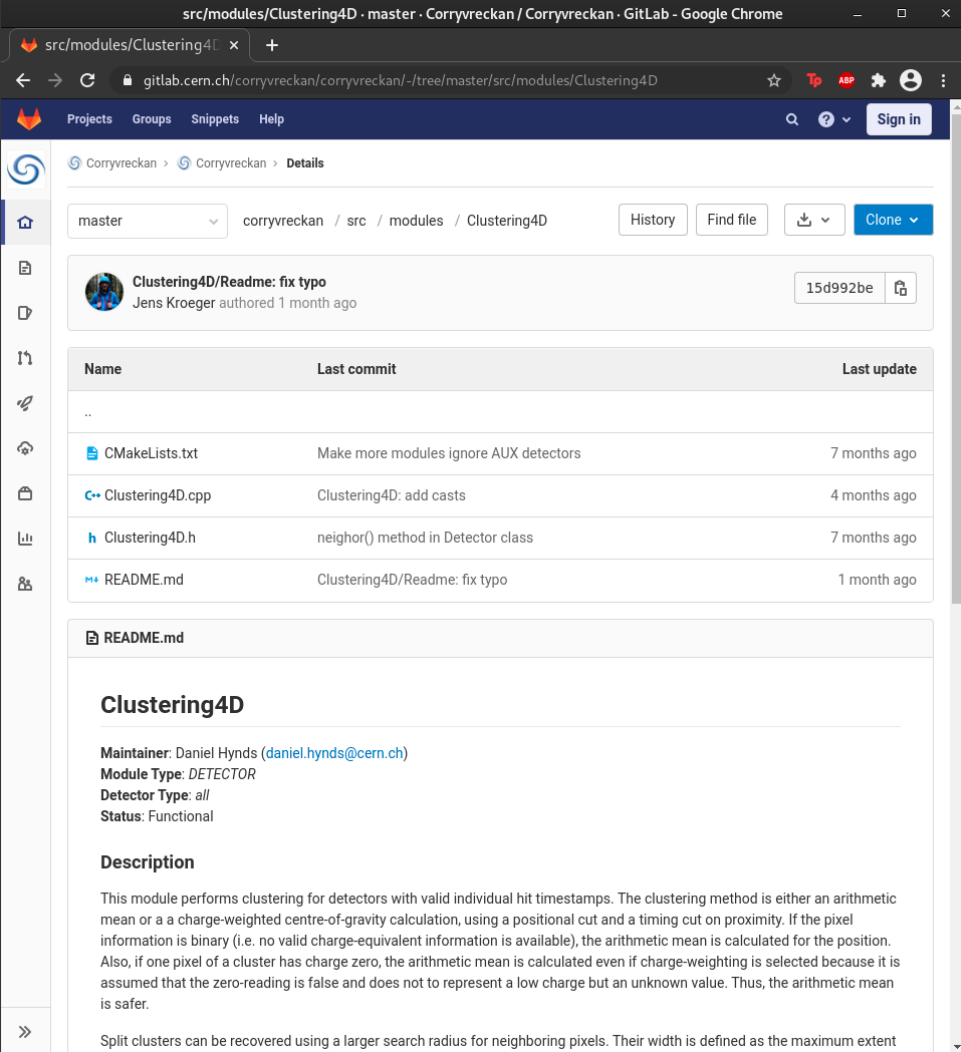
- **doc/** → documentation/user manual
- **etc/** → Ixplus setup script, etc.
- **jobsub/** → job submission tool + README
- **src/** → all the source code
 - **modules/** → code and READMEs of all modules
- **testing/** → CI test configs
- **README.md** → short introduction/getting started
- **Contributing.md** → contribution guidelines



The Corryvreckan Repository

Read the **documentation**,
explore the **source code**:

- **src/** → all the source code
→ **modules/** → code and READMEs of all modules
- e.g. Clustering4D (click here)



The screenshot shows the GitLab web interface for the repository 'Corryvreckan / Corryvreckan'. The browser address bar shows the URL 'gitlab.cern.ch/corryvreckan/corryvreckan/-/tree/master/src/modules/Clustering4D'. The page displays the file tree for the 'Clustering4D' directory, including files like 'CMakeLists.txt', 'Clustering4D.cpp', 'Clustering4D.h', and 'README.md'. A commit history table is visible, showing the most recent commit for 'README.md' by Jens Kroeger. Below the table, the 'README.md' content is displayed, including the title 'Clustering4D', maintainer information (Daniel Hynds), module type (DETECTOR), detector type (all), and status (Functional). The description explains the clustering method and its safety features.

Name	Last commit	Last update
..		
CMakeLists.txt	Make more modules ignore AUX detectors	7 months ago
Clustering4D.cpp	Clustering4D: add casts	4 months ago
Clustering4D.h	neighbor() method in Detector class	7 months ago
README.md	Clustering4D/Readme: fix typo	1 month ago

Clustering4D

Maintainer: Daniel Hynds (daniel.hynds@cern.ch)
Module Type: *DETECTOR*
Detector Type: *all*
Status: Functional

Description

This module performs clustering for detectors with valid individual hit timestamps. The clustering method is either an arithmetic mean or a charge-weighted centre-of-gravity calculation, using a positional cut and a timing cut on proximity. If the pixel information is binary (i.e. no valid charge-equivalent information is available), the arithmetic mean is calculated for the position. Also, if one pixel of a cluster has charge zero, the arithmetic mean is calculated even if charge-weighting is selected because it is assumed that the zero-reading is false and does not to represent a low charge but an unknown value. Thus, the arithmetic mean is safer.

Split clusters can be recovered using a larger search radius for neighboring pixels. Their width is defined as the maximum extent

What's new since BTTB8?

Official **Release v2.0** in December 2020 (and **first patch release v2.0.1** on 4th February)

- new **reference paper** accepted by JINST:
Corryvreckan: A Modular 4D Track Reconstruction and Analysis Software for Test Beam Data
[arXiv:2011.12730](https://arxiv.org/abs/2011.12730)
- Zenodo entry: <https://zenodo.org/record/4384186>

New Features:

(for a full description, see [Release Notes](#))

- full **General Broken Lines** (GBL) support
- new **track model**: multiplet
→ two straight-line-tracks with one kink

New Modules:

- AnalysisEdgeSensor
- AnalysisMaterialBudget
- EventDefinitionM26 → **Example 3.3**
- JSONWriter
- TrackingMultiplet

Let's get started

Installing Corryvreckan...

- 1) **Virtual Machine**
- 2) Download **Binary**
- 3) Compile from **Source**
- 4) Use on **LXPLUS**
- 5) **Docker** Image

Installation – 1) Download the Virtual Machine

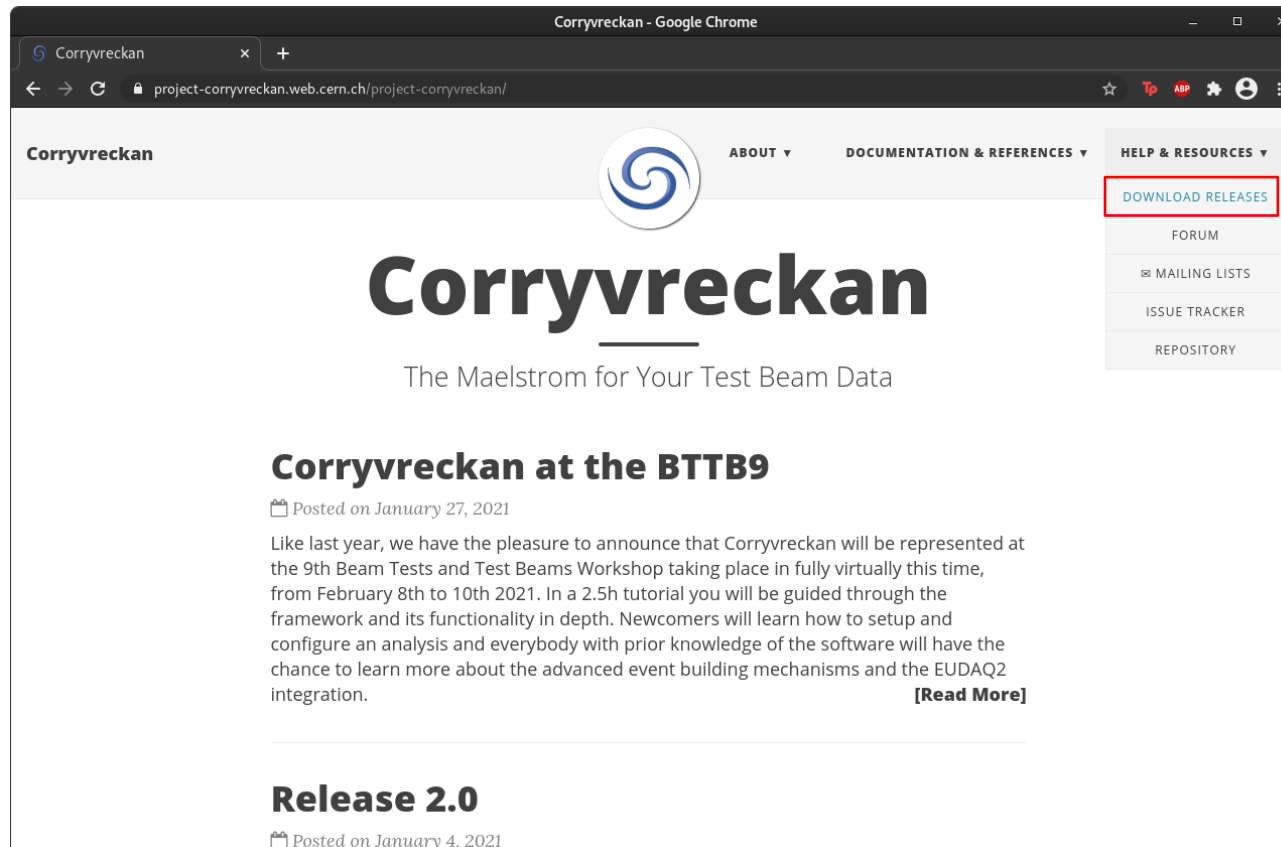
- works on any operating system with **Virtual Box 6.1**
 - <https://www.virtualbox.org/>
 - setup instructions [here](#)
- virtual machine image contains everything needed for this tutorial:
 - Corryvreckan v2.0.1
 - all dependencies
 - data

→ **ideal to get started**



Installation – 2) Download the binary

- for **Centos7** and **SLC6**: download binary release tarballs
 - <https://cern.ch/corryvreckan> → Help & Resources → Download Releases



Corryvreckan

ABOUT ▾ DOCUMENTATION & REFERENCES ▾ HELP & RESOURCES ▾

DOWNLOAD RELEASES

FORUM

MAILING LISTS

ISSUE TRACKER

REPOSITORY

Corryvreckan

The Maelstrom for Your Test Beam Data

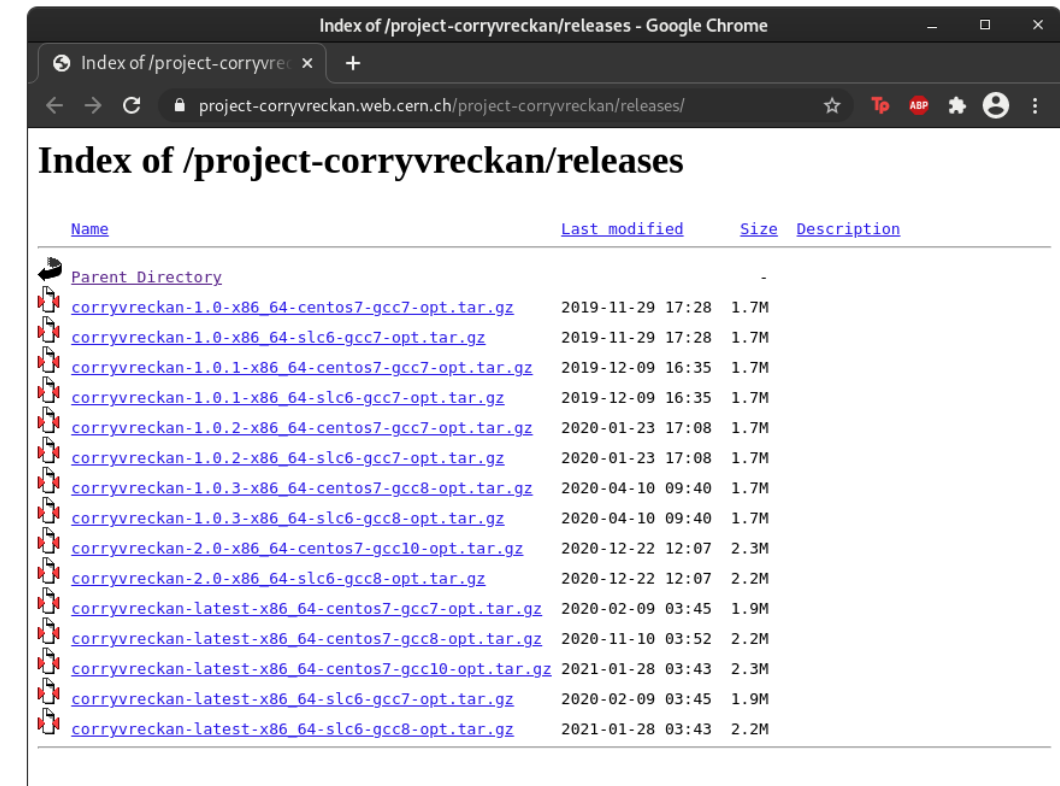
Corryvreckan at the BTTB9

Posted on January 27, 2021

Like last year, we have the pleasure to announce that Corryvreckan will be represented at the 9th Beam Tests and Test Beams Workshop taking place in fully virtually this time, from February 8th to 10th 2021. In a 2.5h tutorial you will be guided through the framework and its functionality in depth. Newcomers will learn how to setup and configure an analysis and everybody with prior knowledge of the software will have the chance to learn more about the advanced event building mechanisms and the EUDAQ2 integration. [\[Read More\]](#)

Release 2.0

Posted on January 4, 2021



Index of /project-corryvreckan/releases

Name	Last modified	Size	Description
Parent Directory	-	-	-
corryvreckan-1.0-x86_64-centos7-gcc7-opt.tar.gz	2019-11-29 17:28	1.7M	
corryvreckan-1.0-x86_64-slc6-gcc7-opt.tar.gz	2019-11-29 17:28	1.7M	
corryvreckan-1.0.1-x86_64-centos7-gcc7-opt.tar.gz	2019-12-09 16:35	1.7M	
corryvreckan-1.0.1-x86_64-slc6-gcc7-opt.tar.gz	2019-12-09 16:35	1.7M	
corryvreckan-1.0.2-x86_64-centos7-gcc7-opt.tar.gz	2020-01-23 17:08	1.7M	
corryvreckan-1.0.2-x86_64-slc6-gcc7-opt.tar.gz	2020-01-23 17:08	1.7M	
corryvreckan-1.0.3-x86_64-centos7-gcc8-opt.tar.gz	2020-04-10 09:40	1.7M	
corryvreckan-1.0.3-x86_64-slc6-gcc8-opt.tar.gz	2020-04-10 09:40	1.7M	
corryvreckan-2.0-x86_64-slc6-gcc8-opt.tar.gz	2020-12-22 12:07	2.3M	
corryvreckan-latest-x86_64-centos7-gcc7-opt.tar.gz	2020-02-09 03:45	1.9M	
corryvreckan-latest-x86_64-centos7-gcc8-opt.tar.gz	2020-11-10 03:52	2.2M	
corryvreckan-latest-x86_64-centos7-gcc10-opt.tar.gz	2021-01-28 03:43	2.3M	
corryvreckan-latest-x86_64-slc6-gcc7-opt.tar.gz	2020-02-09 03:45	1.9M	
corryvreckan-latest-x86_64-slc6-gcc8-opt.tar.gz	2021-01-28 03:43	2.2M	

Installation – 3) Compile from source code

- requires local **ROOT 6** installation
- recommended if you want to **develop your own code**

```
$ git clone https://gitlab.cern.ch/corryvreckan/corryvreckan.git
$ cd corryvreckan
$ mkdir build && cd build/
$ cmake .. # change CMake flags as needed
$ make install -j<number_of_cores>
```

- **corry executable** available in installation directory:

```
$ path/to/corryvreckan/bin/corry
```


Installation - 4) Use Corryvreckan on LXPLUS

- requires a **CERN LXPLUS** account
- 2 options:
 - **source CVMFS version**

```
$ source /cvmfs/clicdp.cern.ch/software/  
corryvreckan/<version>/x86_64-  
<slc6/centos7>-gcc7-opt/setup.sh
```

Note:

- only default modules available
- e.g. not [**EventLoaderEUDAQ2**]
(needed for Example 3)

- **install on LXPLUS**

```
$ git clone https://gitlab.cern.ch/  
corryvreckan/corryvreckan.git  
$  
$ cd corryvreckan  
$ source etc/setup_lxplus.sh  
$  
$ # continue as for local installation
```

→ rest as for local installation

Installation - 5) Use docker image



- run software in container including all dependencies
→ like “light-weight virtual machine”
- needs “docker” executable
 - install via `$ sudo snap install docker`
- start docker container:
`$ sudo ./start_docker_container.sh`
- then use like usual terminal

Note:

- only **default** modules active are available (like CVMFS version)
 - e.g. not **[EventLoaderEUDAQ2]** (needed for Example 3)
- docker does **not** support **graphics**
 - for **TBrowser** to inspect output: external ROOT installation

Installation - 5) Use docker image



- run software in container including all dependencies
→ like “light-weight virtual machine”

- needs “docker” executable

- install via `$ sudo snap install docker`

- start docker container:

- `$ sudo ./start_docker_container.sh`

If you run docker the **first time**:

```
$ sudo groupadd docker
```

```
$ sudo usermod -aG docker $USER
```

Then log out and back in and try:

```
$ docker run hello-world
```

Small docker cheat sheet:

1. Start corry in docker container:

```
$ ./start_docker_container.sh
```

2. Exit by typing

```
$ exit
```

3. List all containers using

```
$ docker container ls -a
```

4. Re-start exited container

```
$ docker start <container_name>
```

5. Re-attach to interactive container:

```
$ docker attach <container_name>
```

6. Removed (stopped) container:

```
$ docker rm <container_name>
```

Let's get started

Setting up an analysis...

Download the example data and configs



→ Skip this if using the virtual machine!

- example **configs** and **scripts**:

```
$ git clone https://gitlab.cern.ch/jekroege/bttb9_tutorial_corryvreckan.git
```

- In this repo we've got **3 examples** we'll go through:

```
01_atlaspix/      ← simple
02_clicpix2/     ← simple
03_desy/         ← advanced
```

- Go to **data/** and download the example data files:

```
$ cd data/
$ ./download_example_data_01.sh ← only this if connection is slow
$ ./download_example_data_02.sh
$ ./download_example_data_03.sh
```

Reminder: Configuring Corryvreckan

- TOML style = easy to read
- support of physical units (e.g. 25um)
- **configuration** file:
 - global parameters
 - specifies modules and parameters
 - sets input/output paths
 - defines log level
- **geometry** file:
 - number, types and positions/rotations of sensors
 - role of sensors (DUT, reference, auxiliary)

example *configuration*

```
1 [Corryvreckan]
2 log_level = "WARNING"
3 detectors_file = "geometry.conf"
4 number_of_tracks = 900000
5
6 [EventLoaderEUDAQ2]
7 file_name = "data/run0000456.raw"
8 inclusive = false
9 buffer_depth = 1000
10 shift_triggers = -1
```

example *geometry*

```
1 [W0013_D04]
2 number_of_pixels = 256, 256
3 orientation = 10.9deg, 17.2deg, -1.3deg
4 orientation_mode = "xyz"
5 pixel_pitch = 55um, 55um
6 position = 886.5um, 270um, 0
7 spatial_resolution = 4.8um, 4.8um
8 time_resolution = 1.56ns
9 type = "Timepix3"
10 role = "reference"
```

Example 1 - Data and configuration files

- All **configuration** and **geometry** files in

`01_atlaspix/`

- `geometries/`
- `01_analysis_telescope.conf`
- `02_analysis_withDUT.conf`
- `03_online_monitor.conf`
- `04_filewriter.conf`
- `05_filereader.conf`

- The **data** is in

`data/`

- `01_data_atlaspix/`
- `01_data_telescope/`

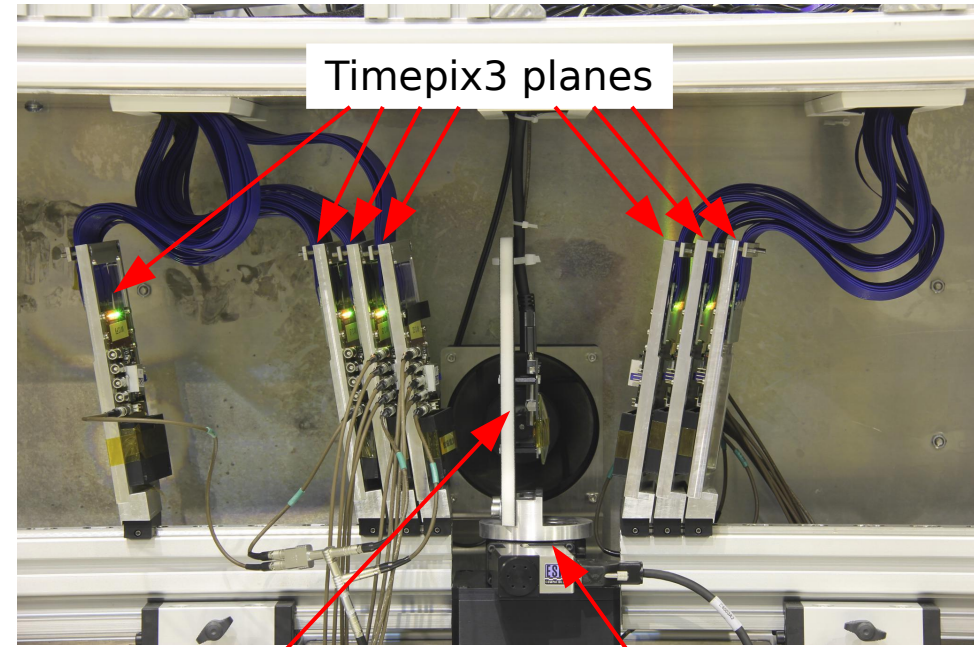
Disclaimer:

The shown analyses are chosen to be instructive and illustrate the framework functionality.

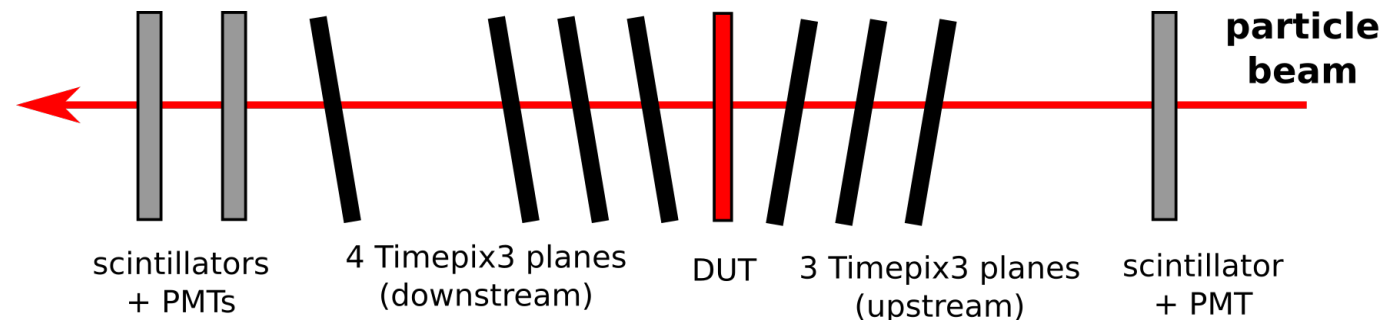
Please don't infer any sensor performance estimates.

Example 1 - Let's have a look at the setup

- Timepix3 **telescope**:
 - data-driven readout
 - pixel-by-pixel timestamp
- ATLASpix as **device-under-test**:
 - data-driven readout
 - pixel-by-pixel timestamp



device-under-test (DUT) translation + rotation stage



Example 1.1 - Let's start with the **telescope only**

01_analysis_telescope.conf

- **configuration file:**
 - global parameters
 - specifies **[Modules]** and **parameters**
 - sets input/output paths
 - defines log level
- all detectors are data-driven
→ need **[Metronome]** to define event

```
[Corryvreckan]
output_directory = "output"
detectors_file =
    "geometries/01_alignment_telescope.geo"
histograms_file =
    "01_histograms_analysis_telescope.root"

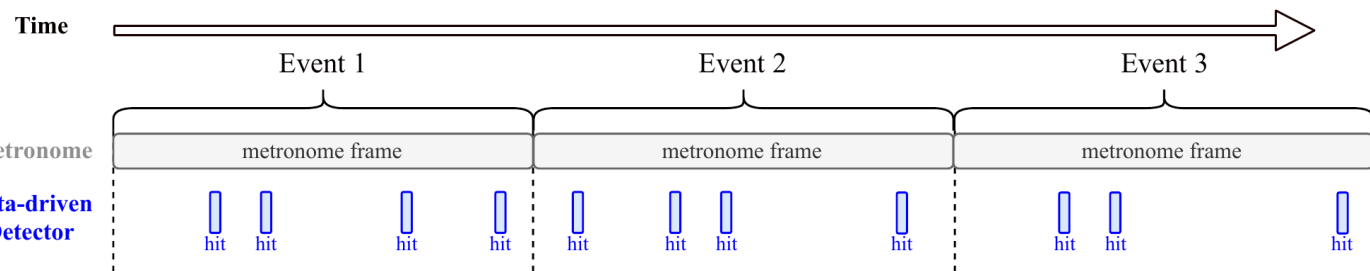
[Metronome]
event_length = 20us

[EventLoaderTimepix3]
input_directory = "../data/01_data_telescope"

[Clustering4D]
time_cut_abs = 200ns

[Correlations]

[Tracking4D]
min_hits_on_track = 6
spatial_cut_abs = 200um, 200um
time_cut_abs = 50ns
```



Example 1.1 - Let's check the geometry

geometries/01_alignment_telescope.geo

- **geometry** file:
 - number/size of pixels
 - detector type
 - position/rotation
 - role of particular sensors:
 - DUT
 - reference
 - auxiliary

```
[Timepix3_0]
number_of_pixels = 256,256
orientation = 9.94299deg,187.403deg,-1.36501deg
orientation_mode = "xyz"
pixel_pitch = 55um,55um
position = 939.743um,285.05um,0
spatial_resolution = 4um,4um
time_resolution = 1.5ns
type = "Timepix3"

[Timepix3_1]
...

[Timepix3_2]
...
role = "reference"
```

Example 1.1 - Let's run the analysis

- run analysis:

```
$ corry -c 01_analysis_telescope.conf
```

- event loop is updated continuously:

```
|16:24:49.775| (STATUS) =====| Event loop |=====
|16:25:12.339| (STATUS) Ev: 564.9k Px: 310.4k Tr: 17.6k (0.0312/ev) t = 11.298s
```

- You can interrupt gently by hitting **Ctrl+C**

- alternatively use

```
$ corry -c 01_analysis_telescope.conf -o number_of_events=25000
```

```
$ corry -c 01_analysis_telescope.conf -o number_of_tracks=250000
```

We'll look at this in more detail in a minute!

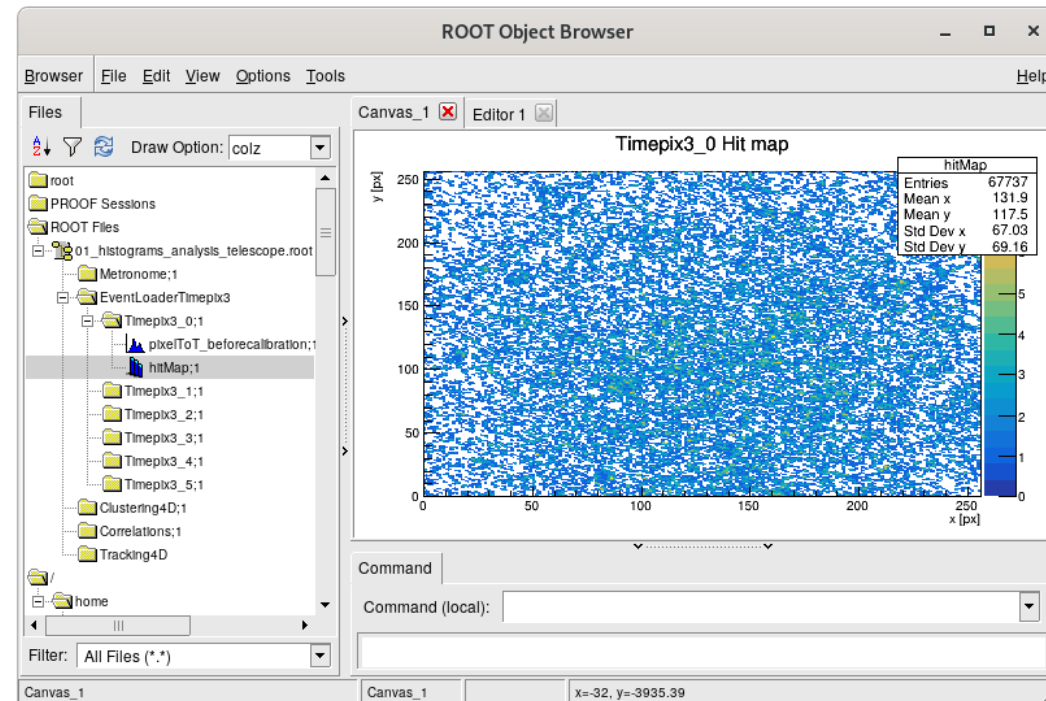


Example 1.1 - Let's check the results

- terminal output:

```
|16:37:52.337| (STATUS) =====| Finalising modules |=====
|16:37:53.317| (STATUS) Wrote histogram output file to /home/jens/Documents/bttb8_tutorial_corryvreckan/01_atlaspix/output/01_histograms_analysis_telescope.root
```

- ROOT file:**
histograms of all modules



Example 1.2 - Now let's add a device-under-test

- still all detectors are data-driven
→ keep **[Metronome]** to define event
- in **configuration** file:
 - add **[EventLoaderATLASpix]**
 - add **[DUTAssociation]** to associate tracks to hits on the device-under-test
 - add analysis module(s)

02_analysis_telescope_withDUT.conf

```
[EventLoaderTimepix3]
input_directory = "../data/01_data_telescope"

[EventLoaderATLASpix]
input_directory = "../data/01_data_atlaspix"

[Clustering4D]

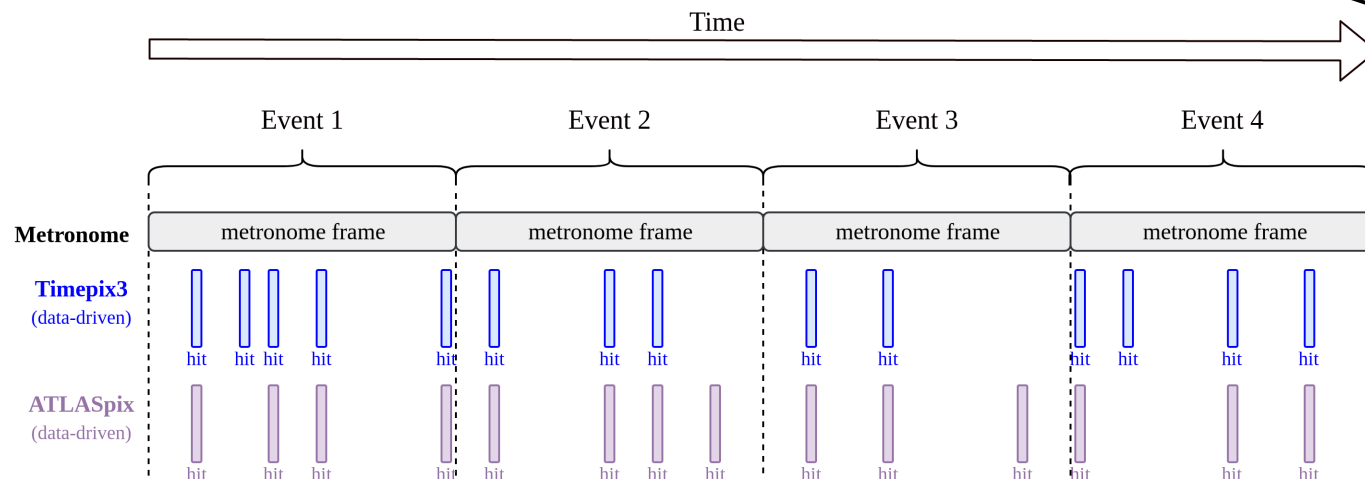
[Correlations]
time_cut_abs = 200ns

[Tracking4D]
min_hits_on_track = 6
spatial_cut_abs = 200um, 200um
time_cut_abs = 50ns

[DUTAssociation]
time_cut_abs = 200ns
spatial_cut_abs = 300um, 150um

[AnalysisDUT]

[AnalysisEfficiency]
```



Example 1.2 - Now let's add a device-under-test

- in **geometry** file:
 - add **[ATLASpix_0]**
 - mark as device-under-test ("**dut**")
 - optionally define region-of-interest ("**roi**") for analysis modules

geometries/02_alignment_telescope_withDUT.geo

```
[Timepix3_2]
...
[ATLASpix_0]
number_of_pixels = 25,400
orientation = -0.0124905deg,0.0684685deg,-0.52609deg
orientation_mode = "xyz"
pixel_pitch = 130um,40um
position = 335.278um,-1.55396mm,105mm
spatial_resolution = 40um,10um
time_resolution = 200ns
role = "dut"
type = "ATLASpix"
roi = [1,1], [1, 398], [23, 398], [23, 1]

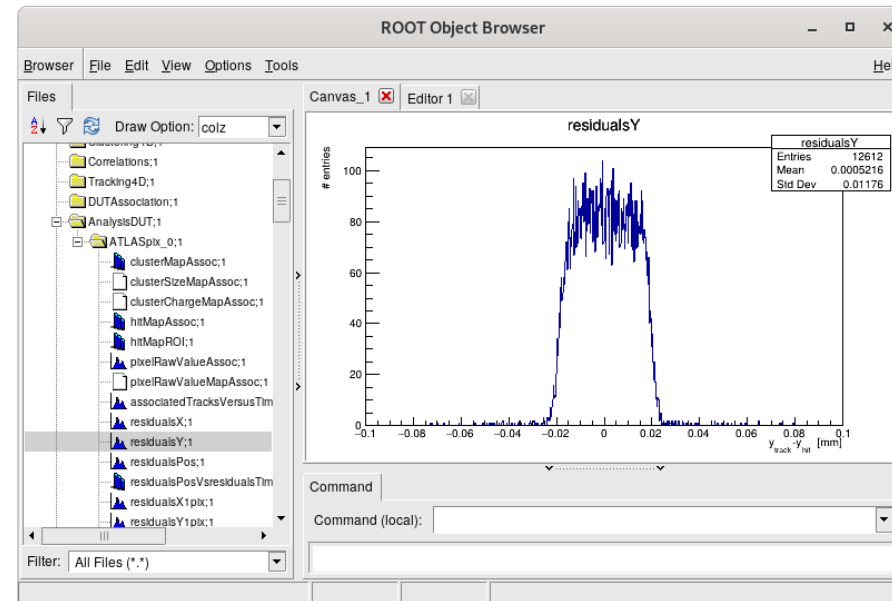
[Timepix3_3]
...
```

Example 1.2 - Let's check the result again

- now the **terminal** has some more information:

```
|17:21:35.912| (STATUS) =====| Finalising modules |=====
|17:21:38.781| (STATUS) [F:DUTAssociation:ATLASpix_0] In total, 28385 clusters are associated to 28323 tracks.
|17:21:38.781| (INFO) [F:DUTAssociation:ATLASpix_0] Number of tracks with at least one associated cluster: 28323
|17:21:40.187| (STATUS) [F:AnalysisEfficiency:ATLASpix_0] Track selection flow:      86598
                                     * rejected by chi2          -45801
                                     * track outside ROI          -529
                                     * track outside DUT          -28712
                                     * track close to masked px   -0
                                     Accepted tracks:            12073
|17:21:40.187| (STATUS) [F:AnalysisEfficiency:ATLASpix_0] Total efficiency of detector ATLASpix_0: 99.9255%, measured with 12064/12073 matched/total tracks
|17:21:40.329| (STATUS) Wrote histogram output file to /home/jens/Documents/bttb8_tutorial_corryvreckan/01_atlaspix/output/02_histograms_analysis_telescope_withDUT.root
```

- **ROOT file:**
histograms of all modules



Example 1.2 - Ways to quit the framework

- **Ctrl+C (SIGINT):**

- **gently** quit corry
- finish processing current event and finalize properly
- recommended option

- **Ctrl+\ (SIGQUIT):**

- **forcefully** terminate corry
- leads to loss of all generated data
- generally not recommended

```
|11:12:18.931| (STATUS) =====| Event loop |=====
|11:12:21.607| (STATUS) Ev: 122.1k Px: 0.3k Tr: 0.0k (0/ev) t = 2.442s^C
|11:12:21.610| (STATUS) Interrupted! Finishing up current event...
|11:12:21.610| (STATUS) Ev: 122.2k Px: 0.3k Tr: 0.0k (0/ev) t = 2.444s
|11:12:21.610| (STATUS) =====| Finalising modules |=====
|11:12:22.077| (STATUS) Wrote histogram output file to /home/jens/Documents/bttb8_tutoria
nalysis_telescope.root
|11:12:22.077| (STATUS) =====| Wall-clock timing (seconds) |=====
|11:12:22.077| (STATUS) Metronome -- 0.04757s = 0.000389ms/evt
|11:12:22.077| (STATUS) EventLoaderTimepix3 : Timepix3_0 -- 0.28141s = 0.002303ms/evt
|11:12:22.077| (STATUS) EventLoaderTimepix3 : Timepix3_1 -- 0.27699s = 0.002267ms/evt
|11:12:22.077| (STATUS) EventLoaderTimepix3 : Timepix3_2 -- 0.30479s = 0.002494ms/evt
|11:12:22.077| (STATUS) EventLoaderTimepix3 : Timepix3_3 -- 0.27372s = 0.002240ms/evt
|11:12:22.078| (STATUS) EventLoaderTimepix3 : Timepix3_4 -- 0.27328s = 0.002236ms/evt
|11:12:22.078| (STATUS) EventLoaderTimepix3 : Timepix3_5 -- 0.27130s = 0.002220ms/evt
|11:12:22.078| (STATUS) Clustering4D : Timepix3_0 -- 0.06246s = 0.000511ms/evt
|11:12:22.078| (STATUS) Clustering4D : Timepix3_1 -- 0.05449s = 0.000446ms/evt
|11:12:22.078| (STATUS) Clustering4D : Timepix3_2 -- 0.05425s = 0.000444ms/evt
|11:12:22.078| (STATUS) Clustering4D : Timepix3_3 -- 0.05352s = 0.000438ms/evt
|11:12:22.078| (STATUS) Clustering4D : Timepix3_4 -- 0.05330s = 0.000436ms/evt
|11:12:22.078| (STATUS) Clustering4D : Timepix3_5 -- 0.06170s = 0.000505ms/evt
|11:12:22.078| (STATUS) Correlations : Timepix3_0 -- 0.05661s = 0.000463ms/evt
|11:12:22.078| (STATUS) Correlations : Timepix3_1 -- 0.05492s = 0.000449ms/evt
|11:12:22.078| (STATUS) Correlations : Timepix3_2 -- 0.05488s = 0.000449ms/evt
|11:12:22.078| (STATUS) Correlations : Timepix3_3 -- 0.05459s = 0.000447ms/evt
|11:12:22.078| (STATUS) Correlations : Timepix3_4 -- 0.05364s = 0.000439ms/evt
|11:12:22.078| (STATUS) Correlations : Timepix3_5 -- 0.05346s = 0.000437ms/evt
|11:12:22.078| (STATUS) Tracking4D -- 0.10156s = 0.000831ms/evt
|11:12:22.078| (STATUS) =====
```

```
|11:12:46.911| (STATUS) [I:Tracking4D] Initialising "Tracking4D"
|11:12:46.911| (STATUS) =====| Event loop |=====
|11:12:49.362| (STATUS) Ev: 113.4k Px: 0.3k Tr: 0.0k (0/ev) t = 2.268s ^\
|11:12:49.362| (FATAL) Aborting!
Aborted (core dumped)
```


Example 1.2 - Let's play with the command line

- parameters from config file can be **overwritten** in the command line:

`corry -c config.cfg` → path to configuration file
`-l log.txt` → write terminal output to file

`-v "FATAL", "STATUS", "ERROR", "WARNING", "INFO", "DEBUG"` → logging verbosity level
also per module: `-o MyModule.log_level="DEBUG"`

`-o histogram_file="hists.root"` → global framework parameter

`-o AnalysisDUT.chi2ndof_cut=3` → module configuration
specified by dot (.) between module and key

`-g mydut.orientation=0deg,5deg,0deg` → detector geometry parameter

→ Play around for a few minutes!

Example 1.3 - the Online Monitor

- we can also see the results **while** the analysis is running
- **configuration** file:
 - add `[OnlineMonitor]`
 - specify which plots you want to see
 - supported keywords: `%DETECTOR%`, `%DUT%`, `%REFERENCE%`

`03_online_monitor.conf`

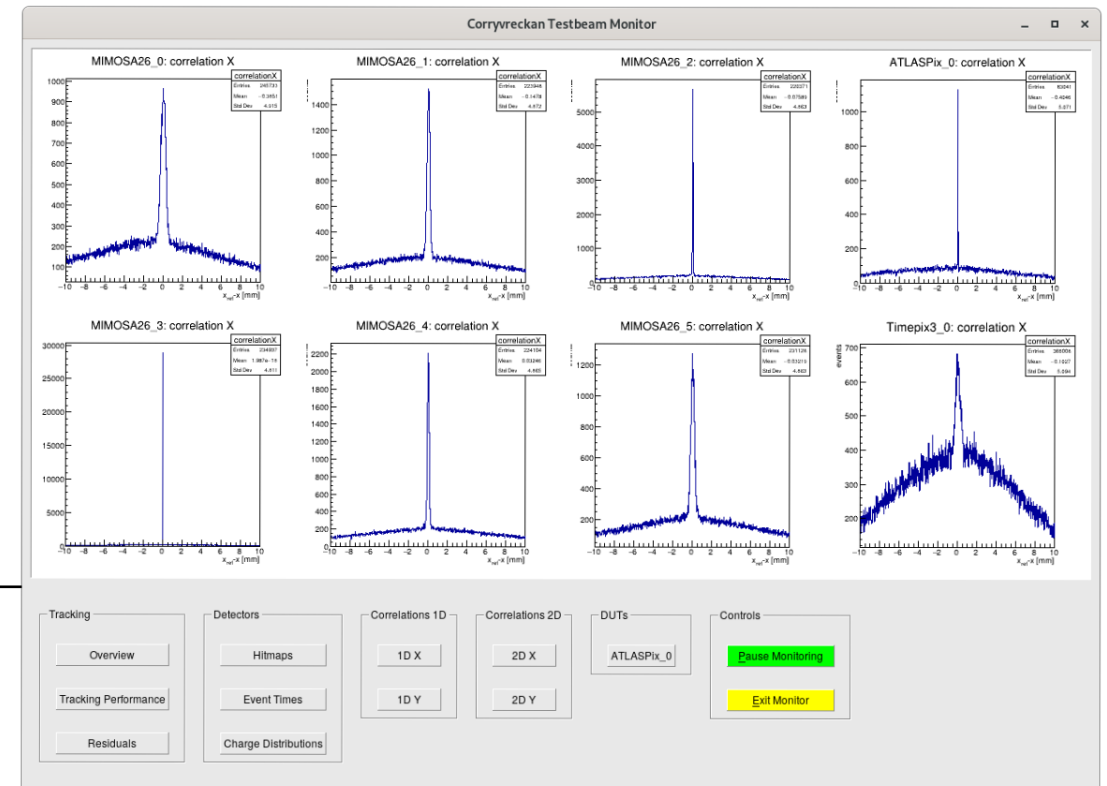
```
[OnlineMonitor]
```

```
update = 200
```

```
dut_plots =
```

```
[["EventLoaderATLASpax/%DUT%/hitMap", "colz"],  
 ["EventLoaderATLASpax/%DUT%/pixelToT"],  
 ["EventLoaderATLASpax/%DUT%/pixelMultiplicity", "log"],  
 ["EventLoaderATLASpax/%DUT%/hPixelTimes_long"]]
```

- opens interactive window
(**Note:** does not work in docker container!)



Example 1.4 - The FileWriter

04_filewriter.conf

- we can **write out reconstructed data** at **any stage** during the reconstruction

- in **configuration** file:

- add **[FileWriter]**
- objects of **all previous modules** will be written to file

```
[Tracking4D]
min_hits_on_track = 6
spatial_cut_abs = 200um, 200um
time_cut_abs = 50ns

[FileWriter]
file_name = "04_output_tuples.root"

[DUTAssociation]
time_cut_abs = 200ns
spatial_cut_abs = 300um, 150um

[AnalysisDUT]

[AnalysisEfficiency]
```

- explore output file with ROOT TBrowser
- read in for further analysis via the **[FileReader]** (next slide)

Example 1.5 - The FileReader

05_filereader.conf

- we can **read in** reconstructed data and **continue** the reconstruction

- in **configuration** file:

- add **[FileReader]**
- all objects from modules above will be written to file

```
# [Tracking4D] ← not needed anymore!  
# min_hits_on_track = 6  
# spatial_cut_abs = 200um, 200um  
# time_cut_abs = 50ns  
  
[FileReader]  
file_name = "output/04_output_tuples.root"  
  
[DUTAssociation]  
time_cut_abs = 200ns  
spatial_cut_abs = 300um, 150um  
  
[AnalysisDUT]  
  
[AnalysisEfficiency]
```

- we can also read in **simulated** data from **Allpix Squared**

- [Talk: What's new on Allpix Squared?](#)
- [Hands-On: Silicon Detector Monte-Carlo Simulations with Allpix Squared - Beginners](#)
- [Hands-On: Everything you would like to know about Allpix Squared - Advanced](#)



Example 2 - Data and configuration files

- All **configuration** and **geometry** files in

`02_clicpix2/`

- `geometries/`
- `01_analysis_telescope.conf`
- `02_analysis_withDUT.conf`

- The **data** is in

`data/`

- `02_data_clicpix2/`
- `02_data_telescope/`

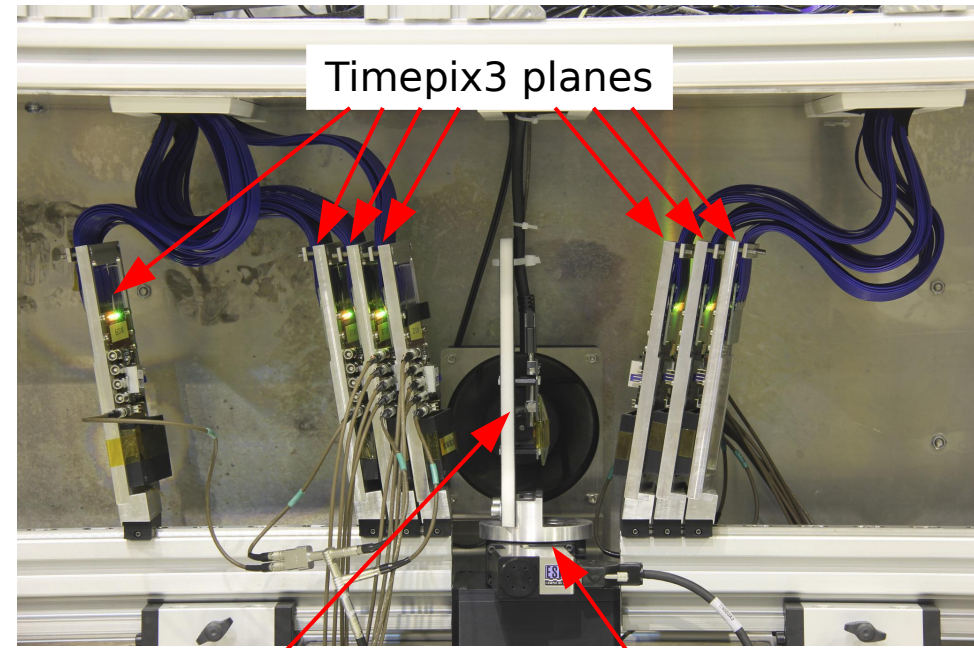
Disclaimer:

The shown analyses are chosen to be instructive and illustrate the framework functionality.

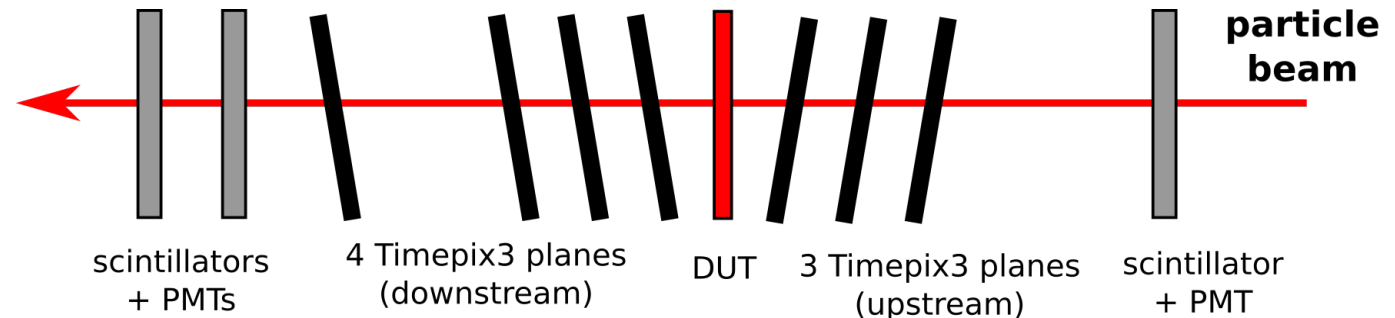
Please don't infer any sensor performance estimates.

Example 2 - Let's have a look at the setup

- Timepix3 telescope: same as in Example 1
 - data-driven readout
 - pixel-by-pixel timestamp
- CLICpix2 as device-under-test:
 - **frame-based readout:**
→ **shutter open/close**
 - no pixel-by-pixel timestamp
(in this operation mode)



device-under-test (DUT) translation + rotation stage



Example 2.1 - Again let's start with the **telescope only**

01_analysis_telescope.conf

- telescope detectors are data-driven
→ need **[Metronome]** to define event
- configuration** file as before
(only path/to/data is updated)

```
[Corryvreckan]
detectors_file =
    "geometries/01_alignment_telescope.geo"
histograms_file =
    "01_histograms_telescope.root"

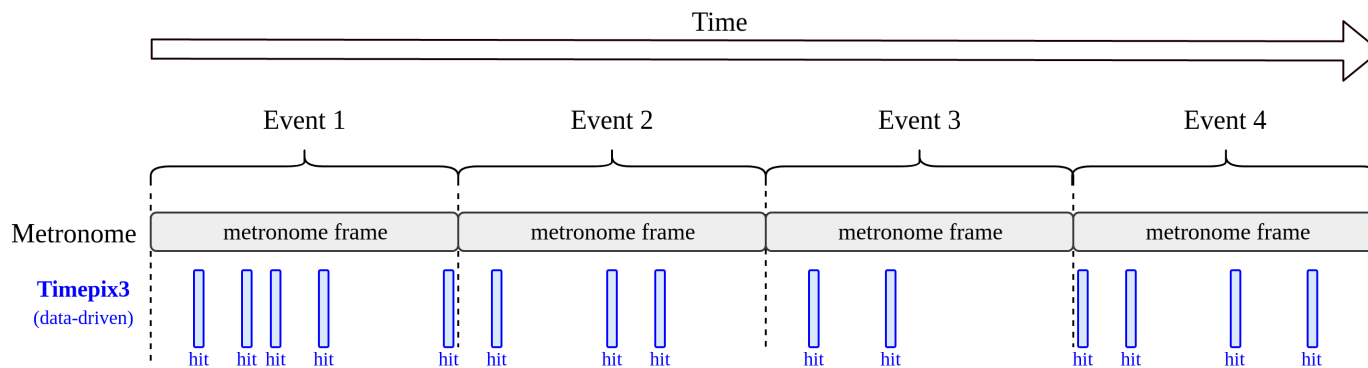
[Metronome]
event_length = 100us

[EventLoaderTimepix3]
input_directory = "../data/02_data_telescope"

[Clustering4D]

[Correlations]

[Tracking4D]
min_hits_on_track = 6
spatial_cut_abs = 200um, 200um
time_cut_abs = 60ns
```



Example 2.2 - Now let's add the device-under-test

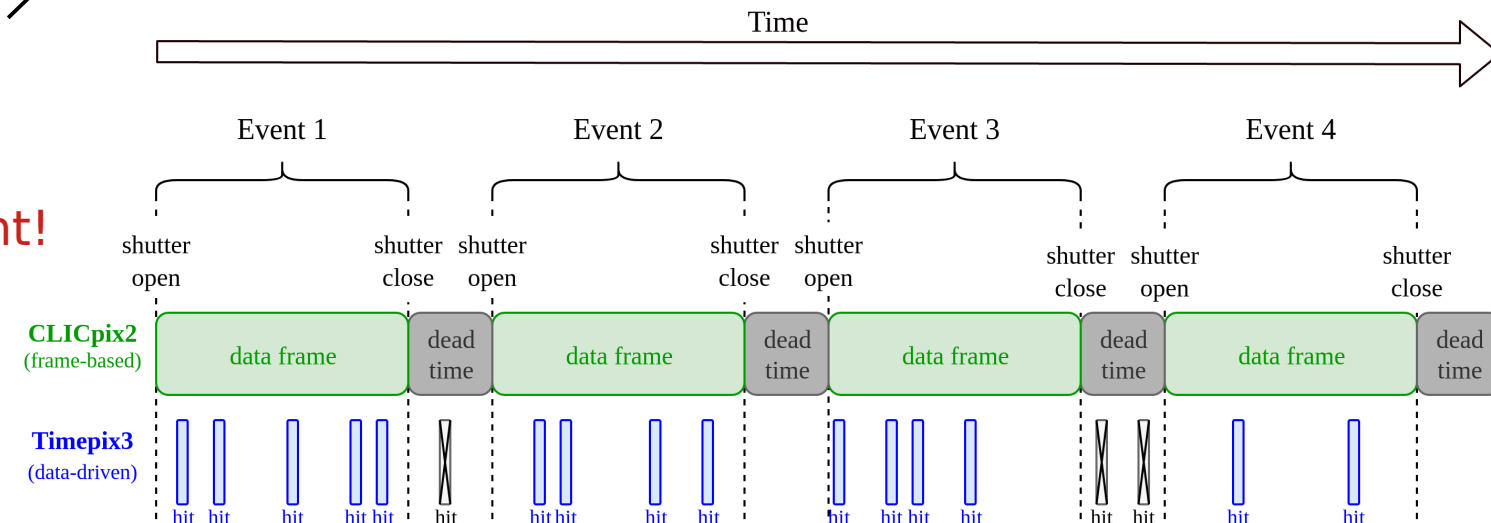
02_analysis_telescope_withDUT.conf

- CLICpix2 is frame-based
 - this detector needs to define the events
 - data from Timepix3 are filled in

```
# Remove this module:  
# [Metronome]  
# event_length = 20us  
  
# Now the CLICpix2 defines the frames:  
[EventLoaderCLICpix2]  
input_directory = "../data/01_data_clicpix2"  
  
[EventLoaderTimepix3]  
input_directory = "../data/01_data_telescope"
```

- **configuration** file:

- remove [Metronome]
- place [EventLoaderCLICpix2] above Timepix3
- **Remember:**
The 1st module defines the event!



Example 2.2 - Now let's add the device-under-test

- CLICpix2 does **not** have **pixel timestamps** (in this operation mode)

- **configuration** file:

- add `[ClusteringSpatial]`
- now need to specify which clustering for which detector:
hierarchy: name > type > nothing
- also add DUT association + analysis module

02_analysis_telescope_withDUT.conf

```
[Clustering4D]
type = "Timepix3"

[ClusteringSpatial]
name = "CLICpix2_0"

[Correlations]

[Tracking4D]

[DUTAssociation]

[AnalysisDUT]
```

Example 3 - Data and configuration files

- All **configuration** and **geometry** files in
 - 03_desy/
 - geometries/
 - 01_tlu_mimosa26_timepix3.conf
 - 02_tlu_mimosa26_timepix3_atlaspix.conf
 - 03_EventDefinitionM26.conf
 - 04_clictd_tlu_mimosa26_timepix3.conf
- The **data** is in
 - data/
 - 03_data_desy/

Note:

This example uses the modules

- [EventLoaderEUDAQ2]
- [EventDefinitionM26]

which require EUDAQ2:

<https://github.com/eudaq/eudaq>

Use:

- VirtualBox (already prepared)
- Compilation from source

Disclaimer:

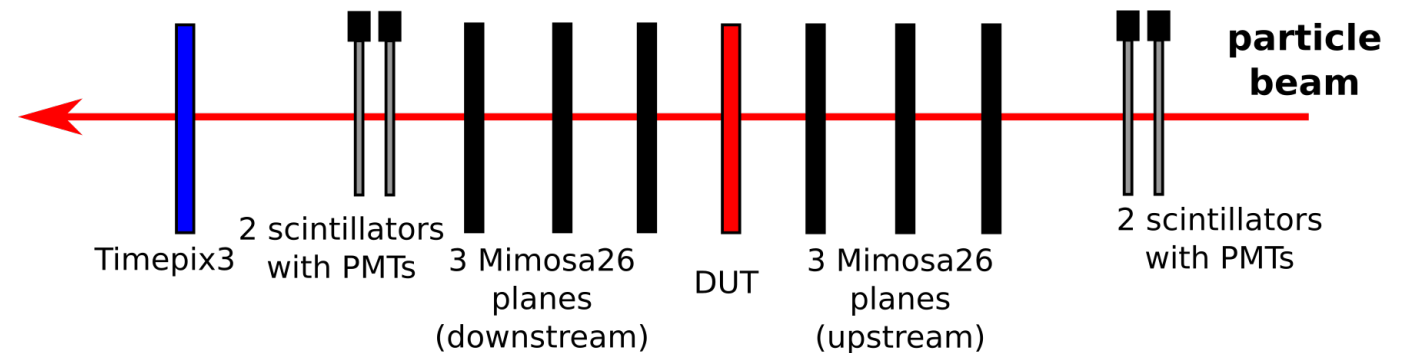
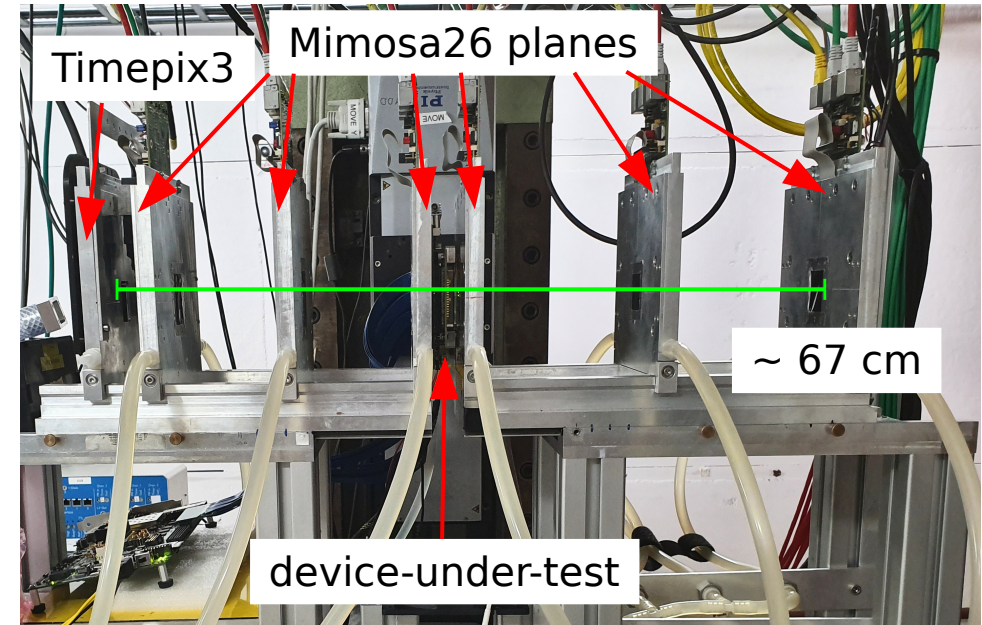
The shown analyses are chosen to be instructive and illustrate the framework functionality.

Please don't infer any sensor performance estimates.

Example 3 - Let's have a look at the setup

Setup at **DESY** using **EUDAQ2**:
<https://github.com/eudaq/eudaq>

- **AIDA TLU** (“trigger logic unit”)
 - generates trigger signal from scintillator coincidence
- **Mimosa26 telescope**:
 - triggered by TLU
 - rolling-shutter readout
 - no pixel-by-pixel timestamp
- **additional** Timepix3 plane:
 - data-driven
 - pixel-by-pixel timestamp



Example 3 - Let's have a look at the event building

Setup at **DESY** using **EUDAQ2**:
<https://github.com/eudaq/eudaq>

- **AIDA TLU** (“trigger logic unit”)

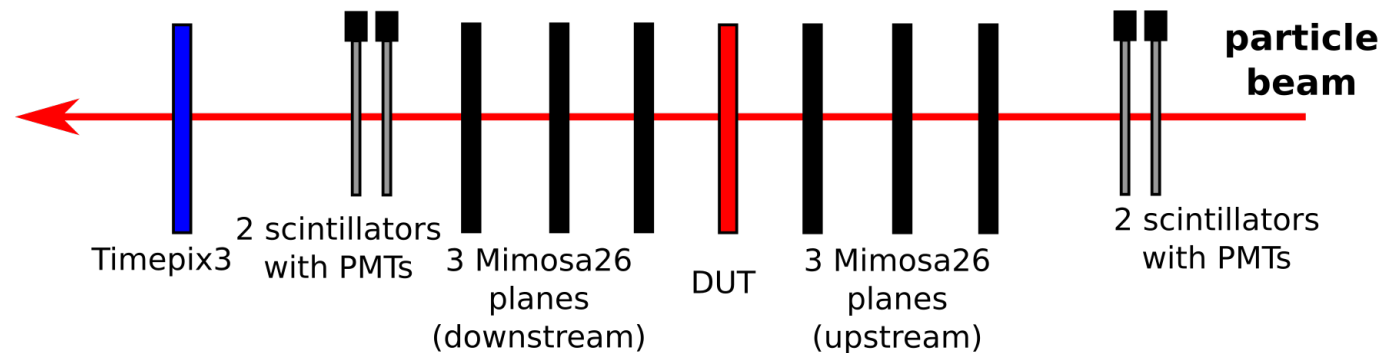
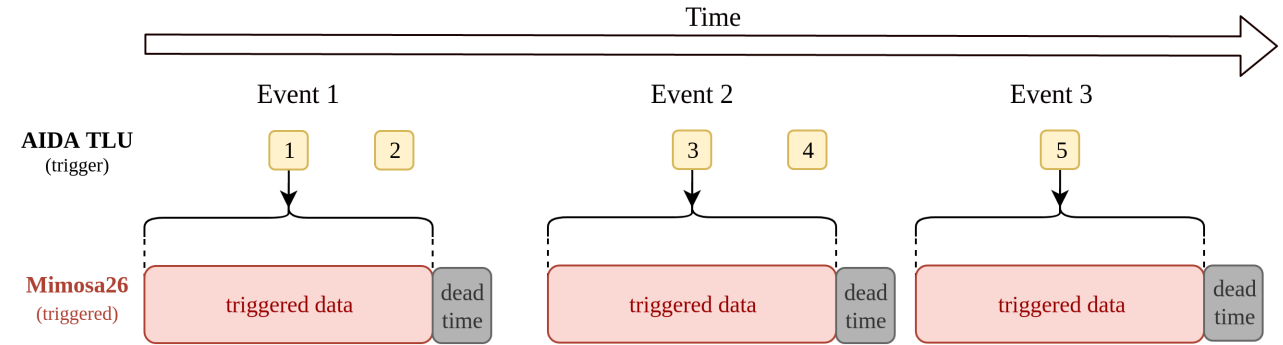
- generates trigger signal from scintillator coincidence

- **Mimosa26 telescope**:

- triggered by TLU
- rolling-shutter readout
- no pixel-by-pixel timestamp

- **additional** Timepix3 plane:

- data-driven
- pixel-by-pixel timestamp



Example 3 - Let's have a look at the event building

Setup at **DESY** using **EUDAQ2**:
<https://github.com/eudaq/eudaq>

- **AIDA TLU** (“trigger logic unit”)

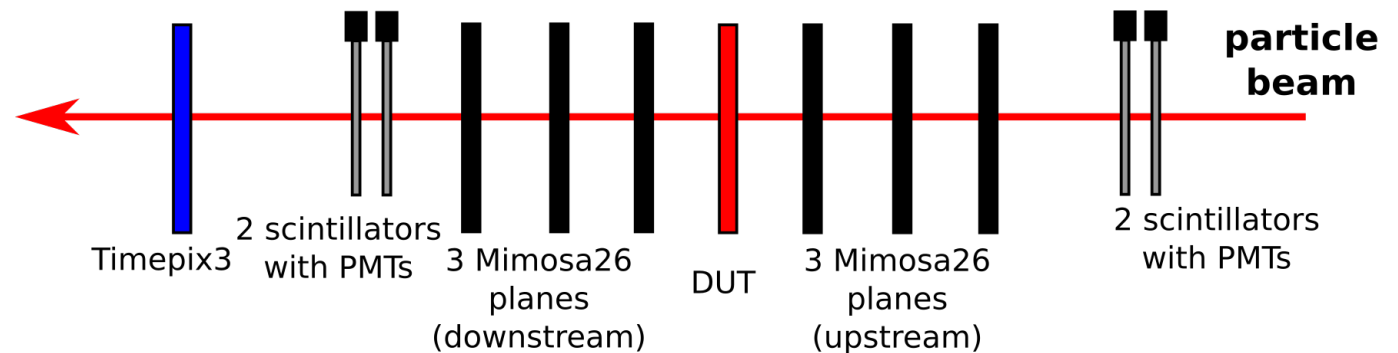
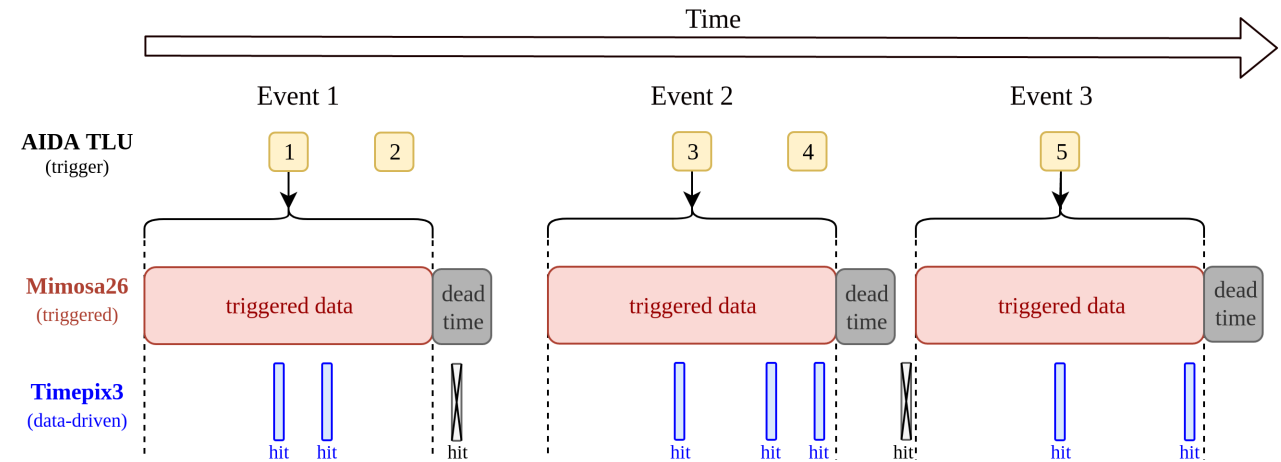
- generates trigger signal from scintillator coincidence

- **Mimosa26 telescope**:

- triggered by TLU
- rolling-shutter readout
- no pixel-by-pixel timestamp

- **additional** Timepix3 plane:

- data-driven
- pixel-by-pixel timestamp



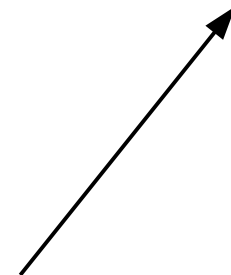
Example 3.1 - Let's have a look at the geometry

- new situation:
 - **AIDA TLU is not a pixel detector**
 - use as “auxiliary” detector
- **geometry** file:
 - **no** pixels/spatial resolution
 - but can have position, time resolution
 - define its role as “**auxiliary**” or “**aux**”

alignment_tlu_mimosa26_timepix3_atlaspix.geo

```
[TLU_0]
...
type = "TLU"
role = "aux"

[Mimosa26_0]
...
```



Example 3.1 - Now let's check the config

- **configuration file:**

- start with the TLU and “expand” the event to match time slice of triggered data
- add data from Mimosa26 based on **trigger ID**
- add hits from Timepix3 based on **pixel timestamp**

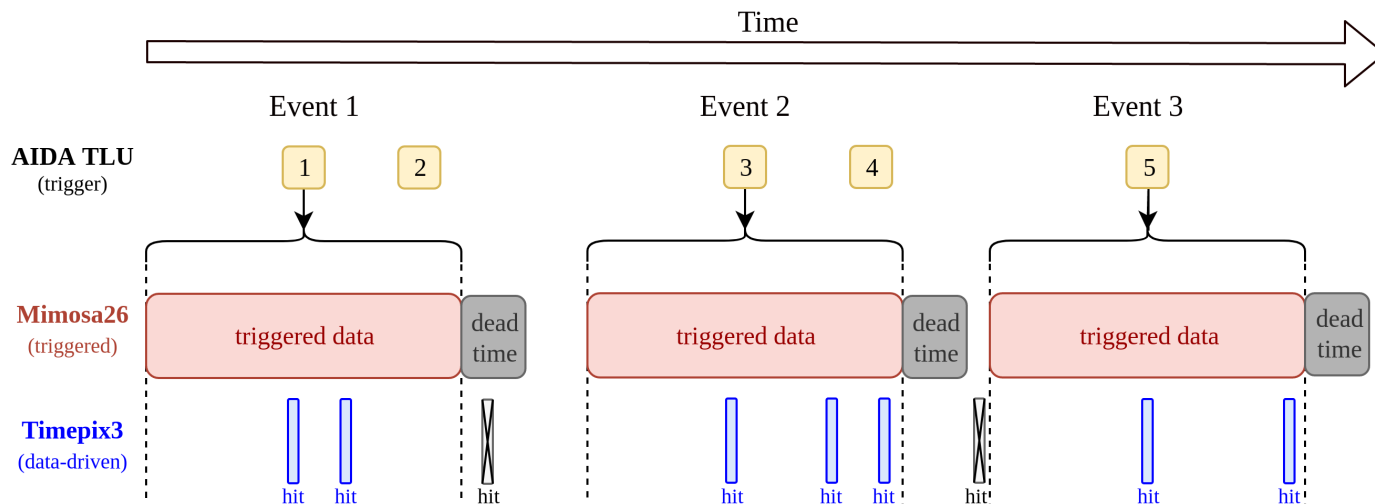
The 1st module defines the event!

01_tlu_mimosa26_timepix3.conf

```
[EventLoaderEUDAQ2]
name = "TLU_0"
adjust_event_times =
    [ ["TluRawDataEvent", -115us, +230us] ]

[EventLoaderEUDAQ2]
type = "Mimosa26"

[EventLoaderEUDAQ2]
type = "Timepix3"
```



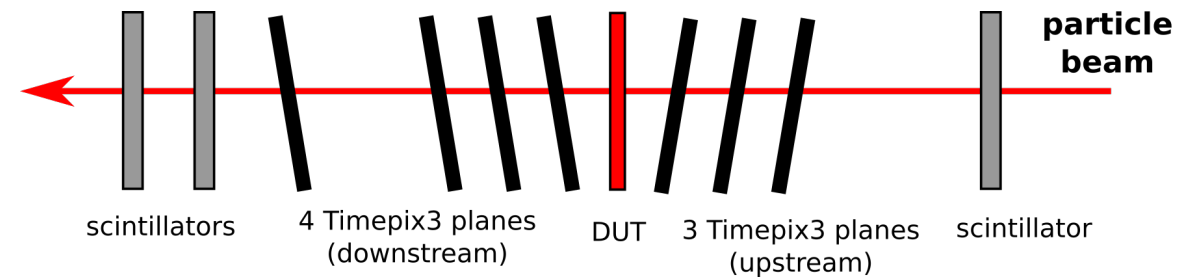
Example 3 - Differences in Tracking

- Examples 1-2: **SPS**

- 7 Timepix3 hits with precise timestamp
- track timestamp = **average** TPX3 timestamp

SPS:

all sensors provide hit timestamps for tracking

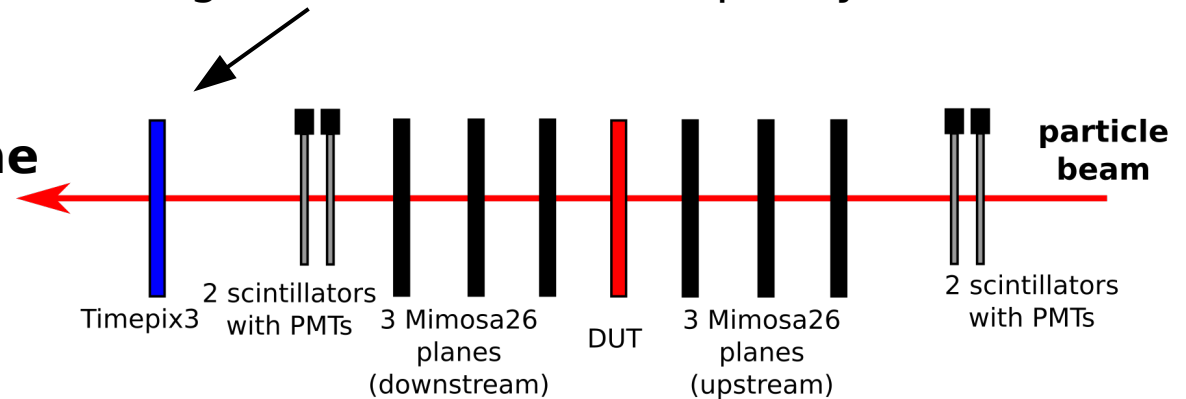


- Example 3: **DESY**

- Mimosa26 hits (3x 115 μ s) with multiple trigger timestamps
- require Timepix3 for **unambiguous track time**
- track timestamp = TPX3 timestamp

DESY:

unambiguous track timestamp only with TPX3



Example 3 - Differences in Tracking

- **DESY**

- **Mimosa26** hits (3x 115 μ s) with multiple trigger timestamps
- require **Timepix3** for unambiguous track time

→ track timestamp = TPX3 timestamp



01_tlu_mimosa26_timepix3.conf

```
[ClusteringSpatial]
type = "Mimosa26"
use_trigger_timestamp = true

[Clustering4D]
type = "Timepix3"

[Tracking4D]
require_detector = "Timepix3_0"
timestamp_from = "Timepix3_0"
```

Example 3 - Differences in Tracking

- one more major difference:
 - SPS beam: 120 GeV pions
 - DESY beam: 5.4 GeV electrons
- much lower momentum
- use **general-broken-line** tracking instead of straight tracks
- in **configuration**:
 - add momentum and track model
- in **geometry**:
 - add material budget in (X/X_0)

01_tlu_mimosa26_timepix3.conf

```
[Tracking4D]
require_detector = "Timepix3_0"
timestamp_from = "Timepix3_0"
momentum = 5.4GeV
track_model = "gb1"
```

alignment_tlu_mimosa26_timepix3_atlaspix.geo

```
[Mimosa26_0]
material_budget = 0.00075
...

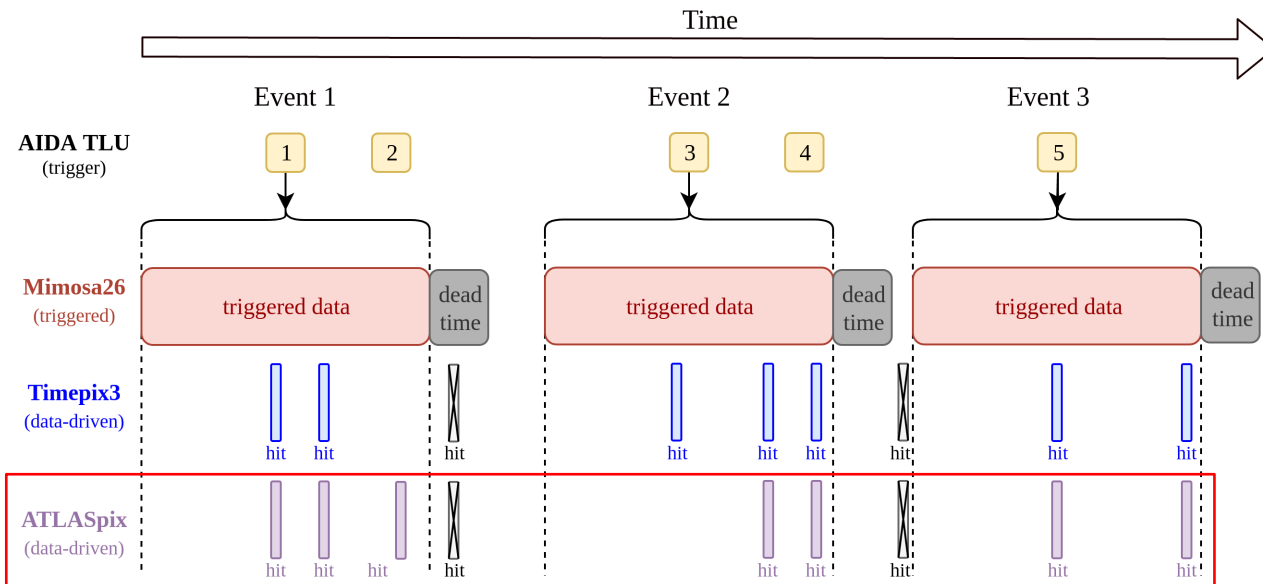
[Timepix3_0]
material_budget = 0.01068
...
```

Example 3.2 - Adding a device-under-test

- ATLASpix as **device-under-test**:

- data-driven
- pixel-by-pixel timestamp

→ simply add another event loader



01_tlu_mimosa26_timepix3_atlaspix.conf

```
[EventLoaderEUDAQ2] # the 1st event loader
name = "TLU_0"      # defines the event!
adjust_event_times =
  [ ["TluRawDataEvent", -115us, +230us] ]

[EventLoaderEUDAQ2]
type = "Mimosa26"

[EventLoaderATLASpix]
type = "ATLASpix"

[EventLoaderEUDAQ2]
type = "Timepix3"
```

Here the order
doesn't matter!

Example 3.3 - Using the EventDefinitionM26

- new module **EventDefinitionM26**
 - use **pivot pixel** information for more precise event definition
- add before all event loaders
 - **Remember:** The 1st module defines the event!

```
[EventDefinitionM26] # the 1st module
detector_event_time = "TLU" # defines the event!
file_timestamp = "path/to/TLUdata"
file_duration = "path/to/M26data"
```

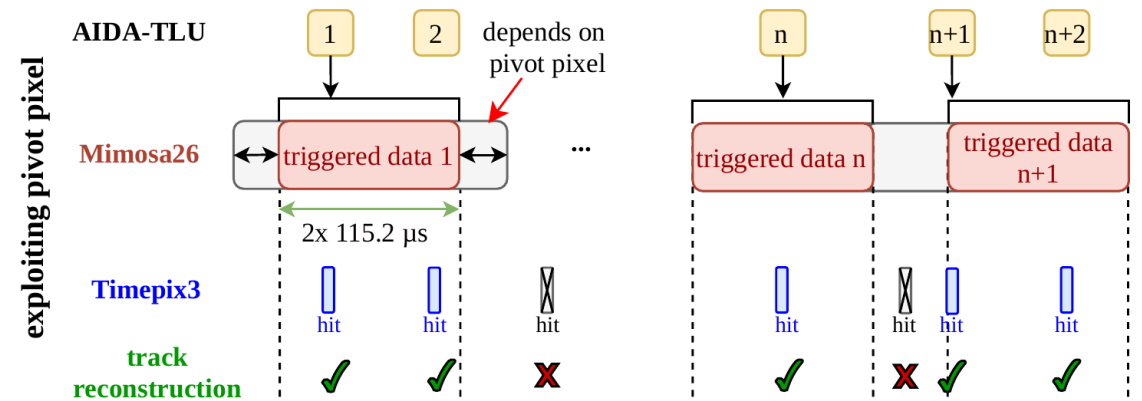
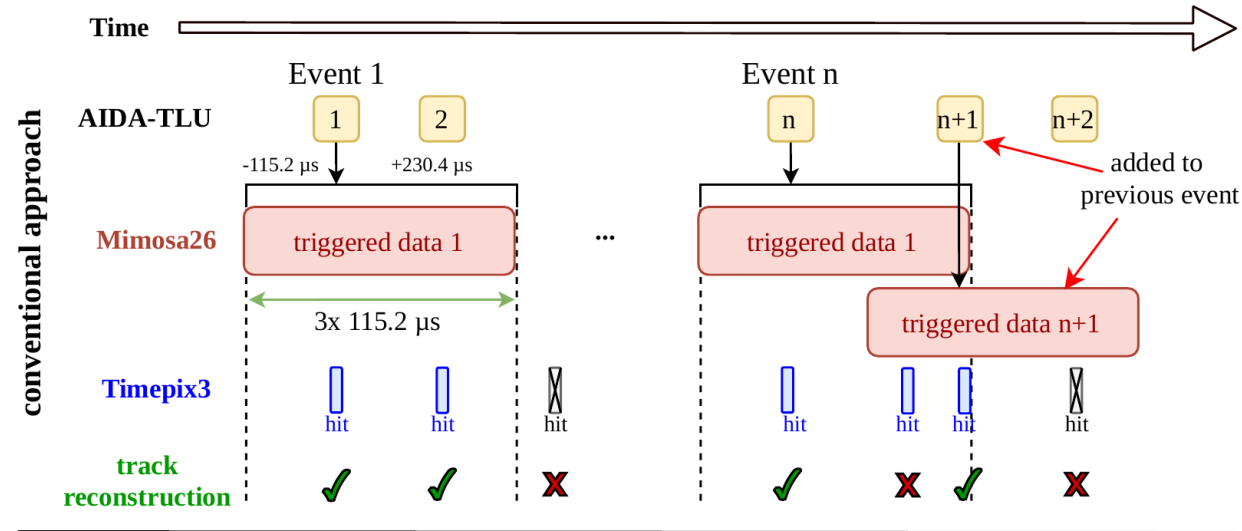
```
[EventLoaderEUDAQ2]
name = "TLU_0"
# Remove this parameter! Only add triggers now!
# adjust_event_times =
  [{"TluRawDataEvent", -115us, +230us}]
```

```
[EventLoaderEUDAQ2]
type = "Mimosa26"
```

```
[EventLoaderATLASpix]
type = "ATLASpix"
```

```
[EventLoaderEUDAQ2]
type = "Timepix3"
```

03_EventDefinitionM26.conf



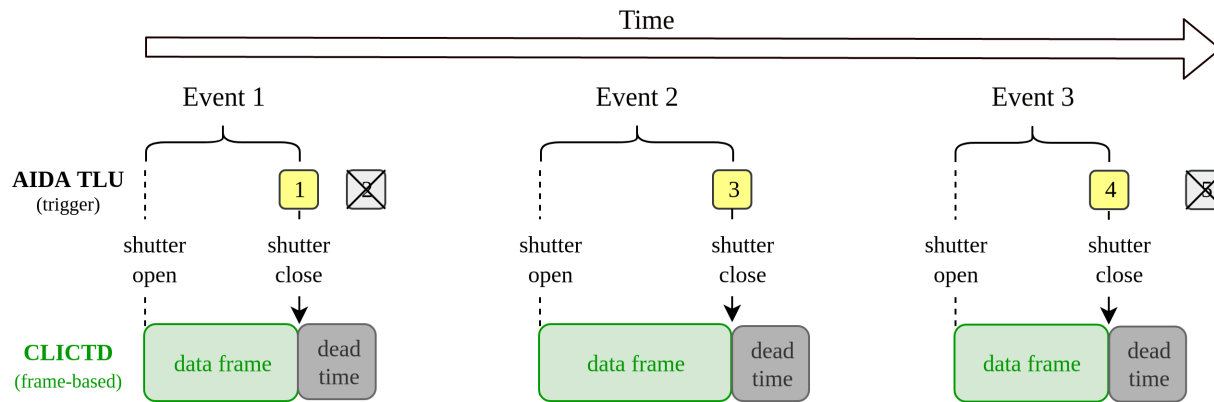
For more details see [arXiv:2011.09205](https://arxiv.org/abs/2011.09205)

Example 3.4 - Let's have a look at the event building

- CLICTD as **device-under-test**:

- no pixel-by-pixel timestamp
- frame-based
 - shutter opened randomly
 - shutter closed by trigger

- CLICTD should define events:



The 1st module defines the event!

04_clicpix2_tlu_mimosa26_timepix3.conf

```
[EventLoaderEUDAQ2]
name = "CLICTD_0"

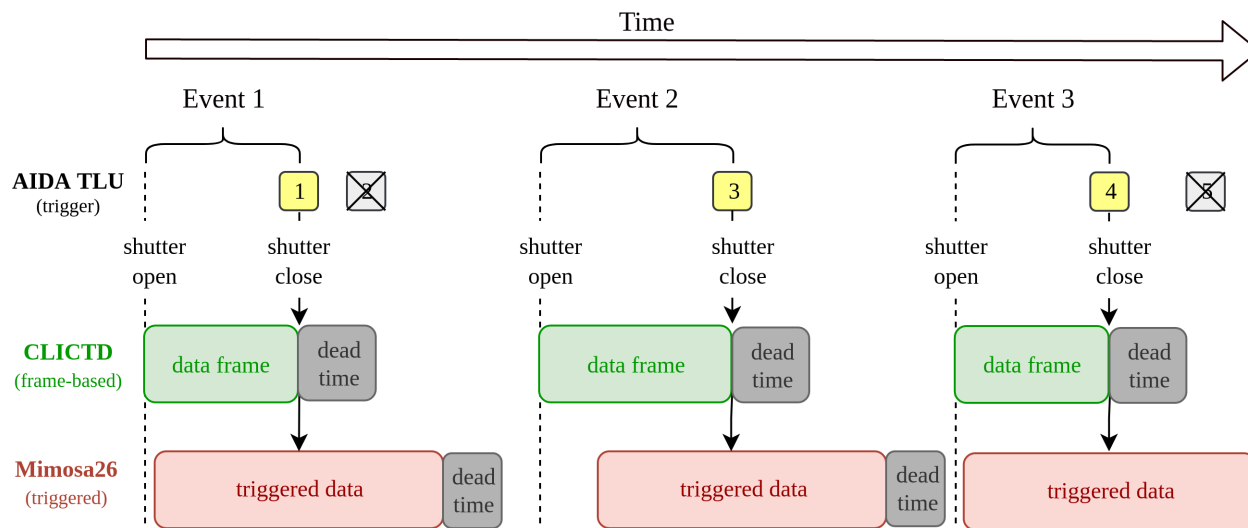
[EventLoaderEUDAQ2]
name = "TLU_0"
```

Example 3.4 - Let's have a look at the event building

- CLICTD as **device-under-test**:

- no pixel-by-pixel timestamp
- frame-based
 - shutter opened randomly
 - shutter closed by trigger

- CLICTD should define events:



The 1st module defines the event!

04_clicpix2_tlu_mimosa26_timepix3.conf

```
[EventLoaderEUDAQ2]
name = "CLICTD_0"

[EventLoaderEUDAQ2]
name = "TLU_0"

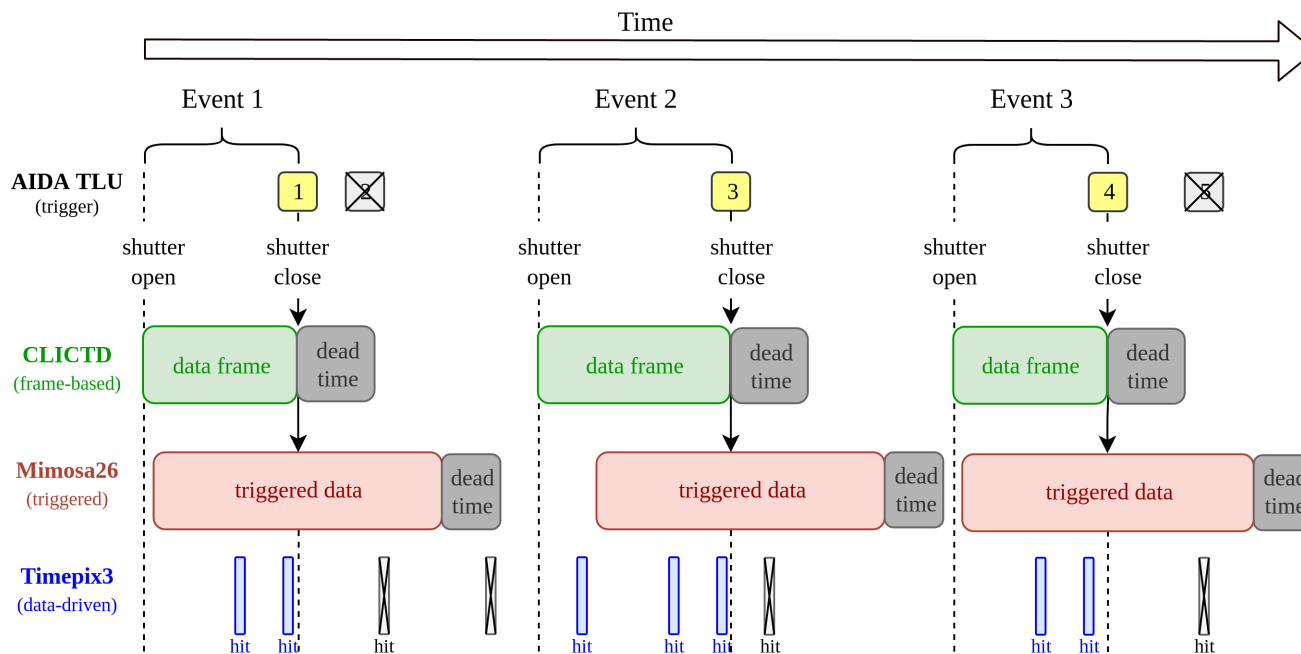
[EventLoaderEUDAQ2]
type = "Mimoso26"
```

Example 3.4 - Let's have a look at the event building

- CLICTD as **device-under-test**:

- no pixel-by-pixel timestamp
- frame-based
 - shutter opened randomly
 - shutter closed by trigger

- CLICTD should define events:



The 1st module defines the event!

04_clicpix2_tlu_mimosa26_timepix3.conf

```
[EventLoaderEUDAQ2]
name = "CLICTD_0"

[EventLoaderEUDAQ2]
name = "TLU_0"

[EventLoaderEUDAQ2]
type = "Mimoso26"

[EventLoaderEUDAQ2]
type = "Timepix3"
```

Corryvreckan in short

reconstruction and analysis tool

for pixel sensor test beam data

- highly flexible/configurable
 - separate modules for each reconstructions/analysis step
 - many different event building options
- comprehensive documentation + beginner-friendly tutorials
- growing number of users/contributors
→ **Thanks to all of you!**

Learn more:



Visit our website:

<https://cern.ch/corryvreckan>



Browse through our manual:

→ [Get the latest version here](#)



Try our other tutorial:

→ [Get Started](#) (no prior experience required)



Check out the repository:

<https://gitlab.cern.ch/corryvreckan/corryvreckan>



Discuss in the forum:

<https://corryvreckan-forum.web.cern.ch/>



Contact us:

corryvreckan.info@cern.ch

<https://mattermost.web.cern.ch/corryvreckan>

Backup

In case there are some questions...

Example Publications

for which Corryvreckan has been used

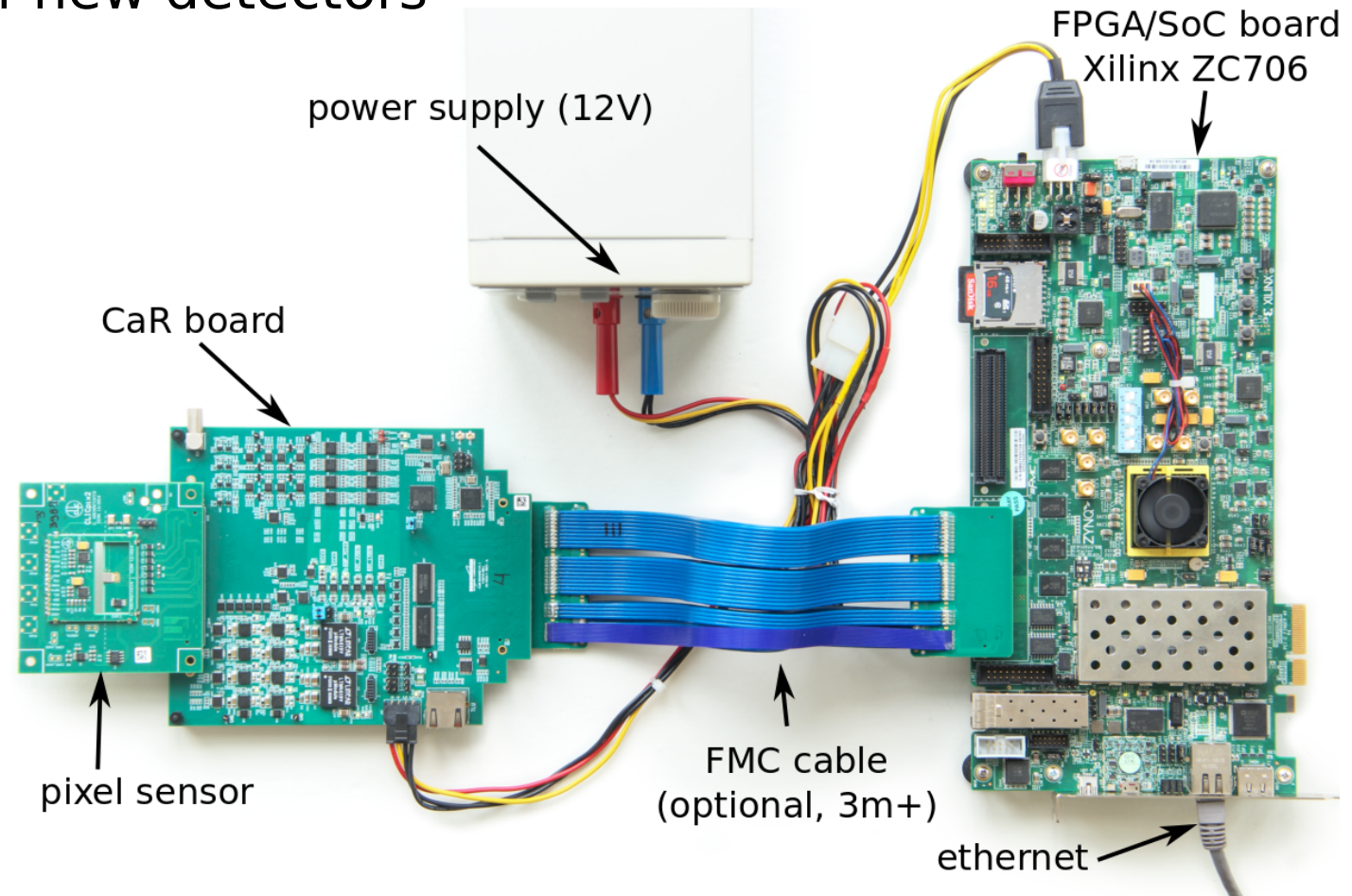
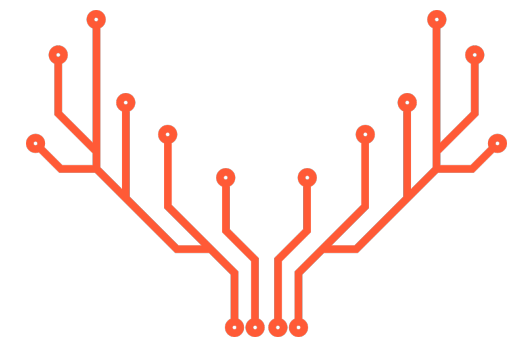
- Florian Pitters:
Time Resolution Studies with Timepix3 Assemblies with Thin Silicon Pixel Sensors
JINST 14 (2019) 05, P05022
DOI: 10.1088/1748-0221/14/05/P05022
- Mathieu Benoit:
Pixel detector R&D for the Compact Linear Collider
JINST 14 (2019) 06, C06003
DOI: 10.1088/1748-0221/14/06/C06003
- Magdalena Munker:
Vertex and tracking detector R&D for CLIC
Nucl.Instrum.Meth.A 980 (2020) 164475
DOI: 10.1016/j.nima.2020.164475
- Morag Williams:
R&D for the CLIC Vertex and Tracking detectors
JINST 15 (2020) 03, C03045
DOI: 10.1088/1748-0221/15/03/C03045
- Jens Kröger:
Silicon Pixel R&D for the CLIC Tracking Detector
JINST 15 (2020) 08, C08005
DOI: JINST 15 (2020) 03, C03045

PhD Theses

- Florian Pitters:
Silicon Detector Technologies for Future Particle Collider Experiments
<https://cds.cern.ch/record/2714709?ln=en>
- Thorben Quast:
Qualification, Performance Validation and Fast Generative Modelling of Beam Test Calorimeter Prototypes for the CMS Calorimeter Endcap Upgrade
<https://cds.cern.ch/record/2725040?ln=en>
- Morag Williams:
Evaluation of Fine-Pitch Hybrid Silicon Pixel Detector Prototypes for the CLIC Vertex Detector in Laboratory and Test-Beam Measurements
(work-in-progress)

Caribou - the readout system

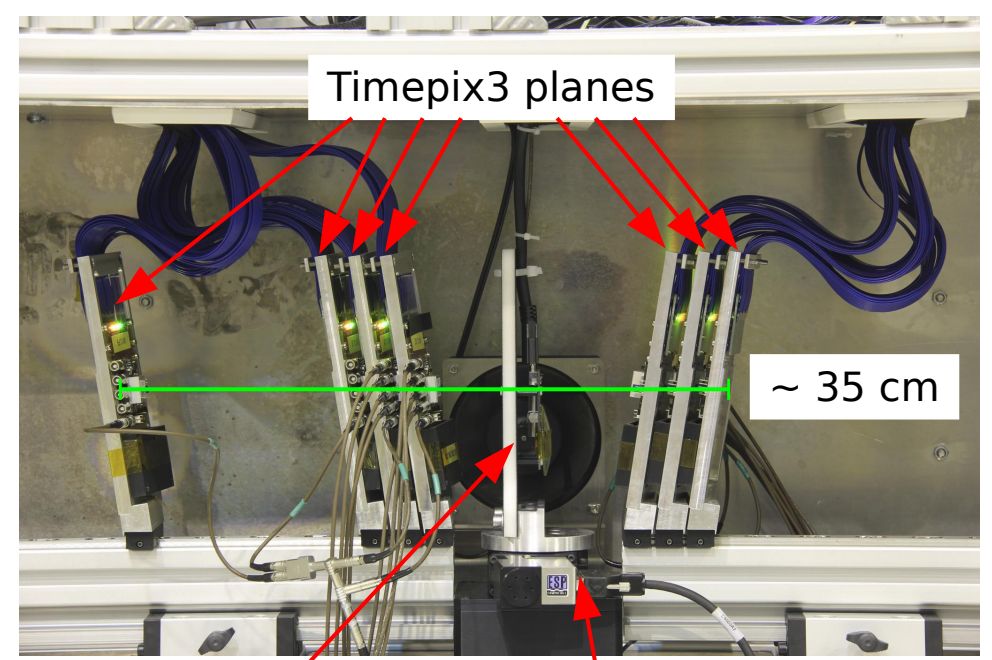
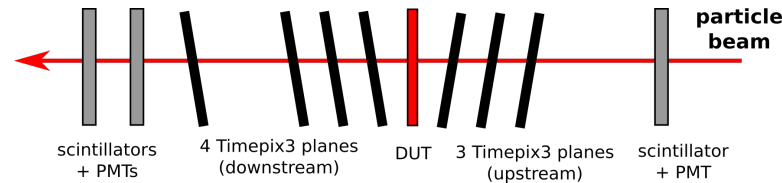
- versatile, open-source, linux-based
- fast & simple implementation of new detectors
→ “fast prototyping”
- **universal:**
 - FPGA board
 - Control & Readout (CaR) board
 - “most of the” firmware/software
- **chip-specific:**
 - chip board
 - “some” firmware/software blocks



SPS vs. DESY II

SPS:

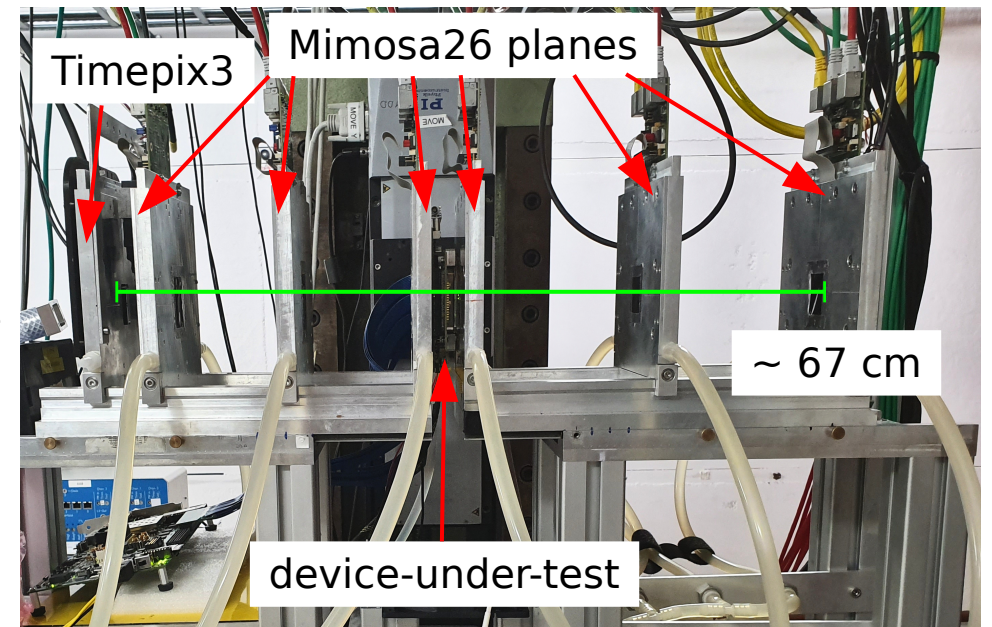
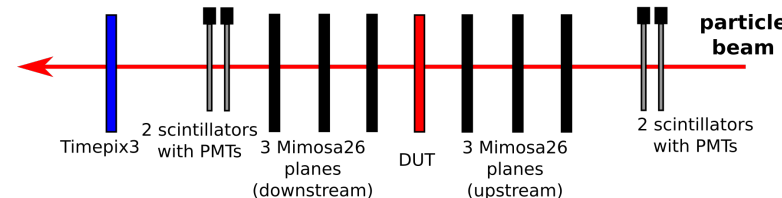
- typical beam condition:
120 GeV pions @ few MHz
- telescope in operation
2014-2018



device-under-test translation + rotation stage

DESY:

- typical beam condition:
5.4 GeV electrons @ few kHz
- use for CLICdp testbeam
campaigns during
LHC LS2 2019-2020



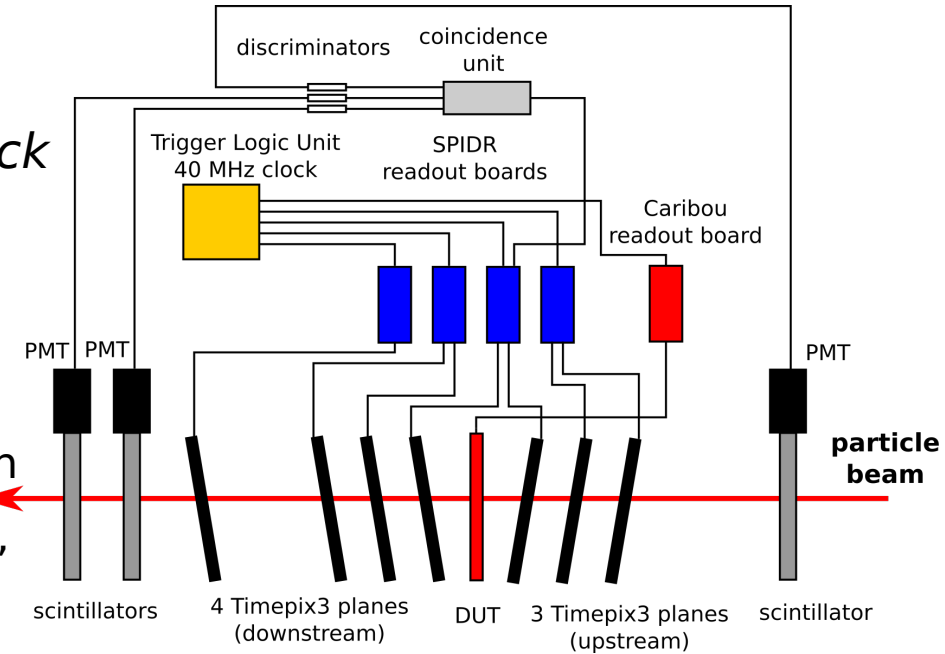
→ much lower rate & energy

SPS vs. DESY II - Readout

continuous readout, timestamps synchronous with global clock

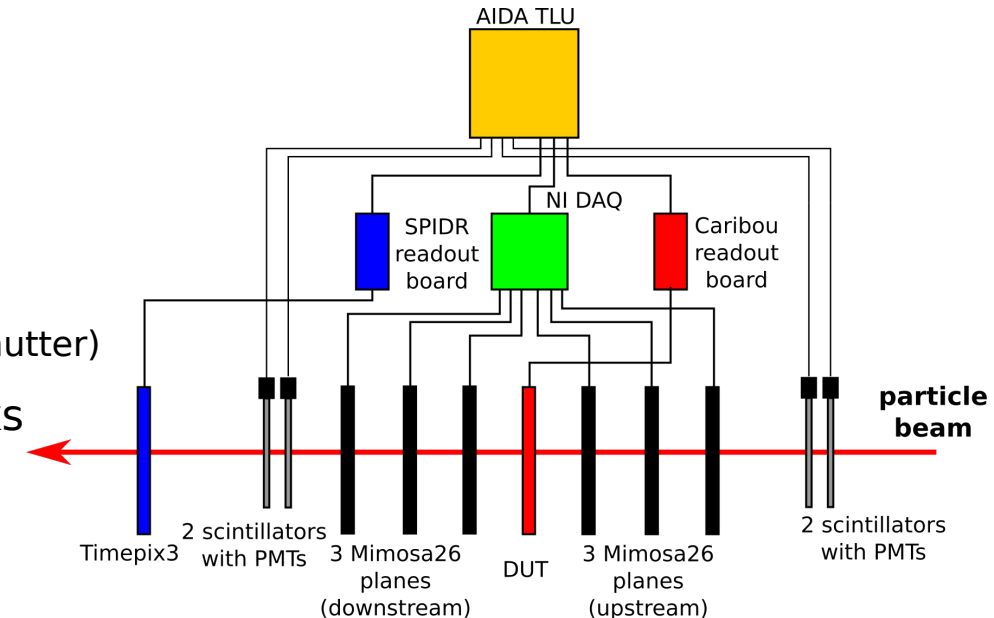
SPS:

- Trigger Logic Unit (TLU) → provides global clock (time sync.)
- 3 scintillators + PMTs → trigger timestamps
- 7 Timepix3 planes → excellent spatial + timing resolution
- DUT → Investigator, Cracow SOI, Timepix3, CLICpix, CLICpix2, ATLASpix



DESY:

- AIDA TLU → provides global clock (time sync.) + triggers Mimosa Readout
- 4 scintillators + PMTs → input to TLU
- 6 Mimosa26 planes → good spatial res. (2x 115μs bins rolling shutter)
- Timepix3 → used to assign ns timestamp to tracks
- DUT → CLICpix2, ATLASpix, CLICTD



→ **additional subsystem**

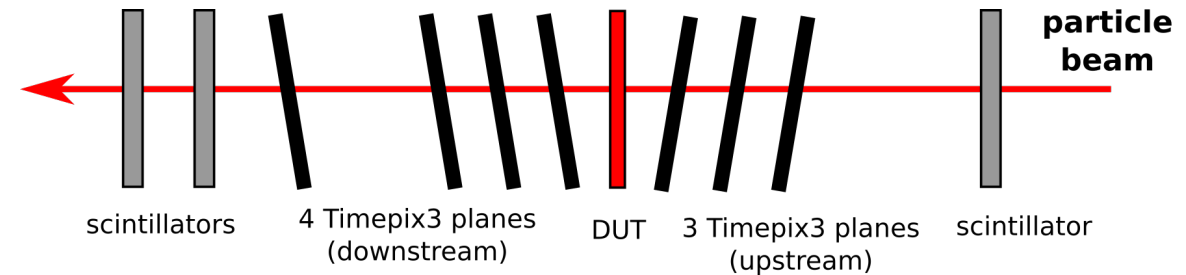
SPS vs. DESY II - Differences in the Analysis

Tracking:

- **SPS:**
 - 7 Timepix3 hits with precise timestamp
 - track timestamp = average TPX3 timestamp
- **DESY:**
 - Mimosa26 hits (3x 115 μ s) with multiple trigger timestamps
 - require Timepix3 for unambiguous track time
 - track timestamp = TPX3 timestamp

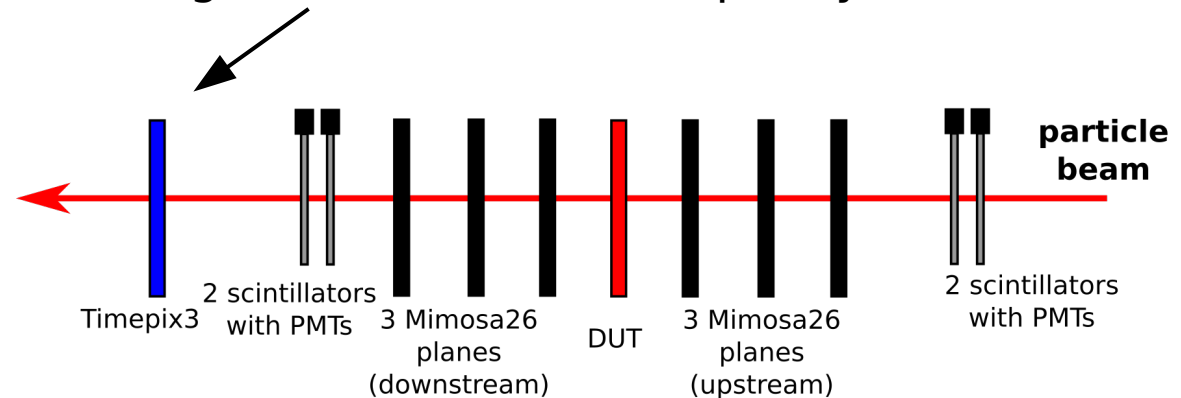
SPS:

all sensors provide hit timestamps for tracking



DESY:

unambiguous track timestamp only with TPX3

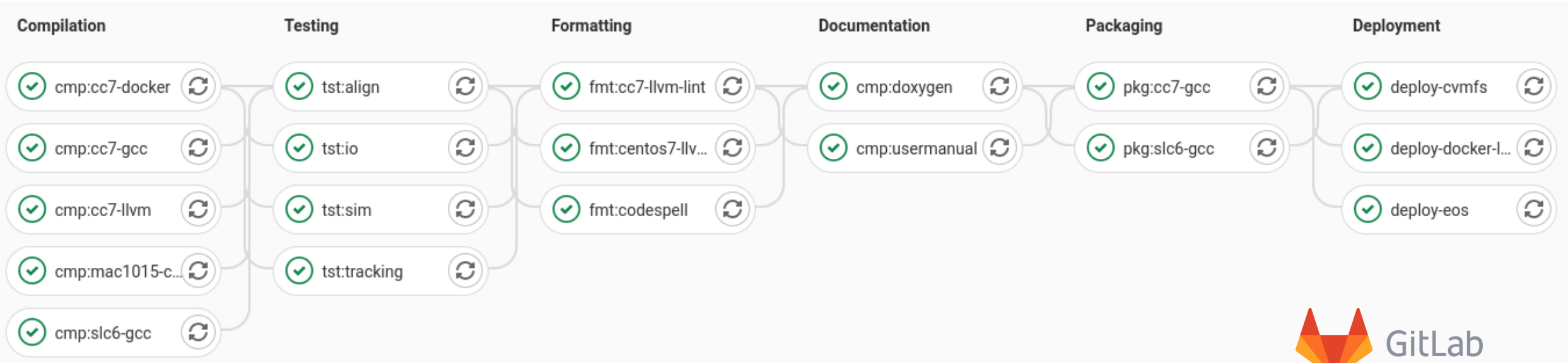


GitLab Continuous Integration

- ensures compilation, formatting, functionality (all stages explained in backup)
- pipeline runs through for every commit

for every commit

for every merge request or tag



GitLab Continuous Integration - all stages

- **Compilation**

- compile source code on Scientific Linux 6, CentOS7, and Mac OS X with GCC, Clang, and AppleClang

- **Testing**

- analyse test data sets and compare output to pass conditions

- **Formatting**

- check format against defined syntax rules (e.g. tabs ↔ whitespaces) to avoid changes caused e.g. by different indentation, and apply linting

- **Documentation**

- compile user manual from LaTeX sources and generate Doxygen code reference

- **Packaging**

- generate release tarballs

- **Deployment**

- publish new version of CVMFS, new docker image in registry, new user manual and code reference on the website and release tarballs

