Hands-On:
The Allpix Squared Silicon Detector Simulation Framework

Simon Spannagel, Paul Schütze,
Adriana Simancas, Manuel Del Rio Viera

10th Beam Telescope and Test Beams Workshop
20th - 24th June 2022

Scope of this Tutorial

- This tutorial will go step-by-step through setting up and running a simulation with Allpix Squared

- The main focus of the tutorial is the usage of Allpix Squared
  - Defining simple to more complicated simulation flows
  - Looking at what modules are doing and how to look at the output
  - You will get a “task” to work on

- In the second part we will include an electric field exported & converted from a TCAD simulation

Modality of this Tutorial

- The slides will contain most commands typed on the terminal
- Following along with your computer is **strongly encouraged!**
- Your options to do so:
  - We provide a Virtual Machine with an Allpix Squared installation
  - It’s possible to follow on lxplus / NAF via a CVMFS installation
  - You can also follow with a local installation, but we cannot debug local Geant4/ROOT6 installations during this session
- Please download the tutorial materials here

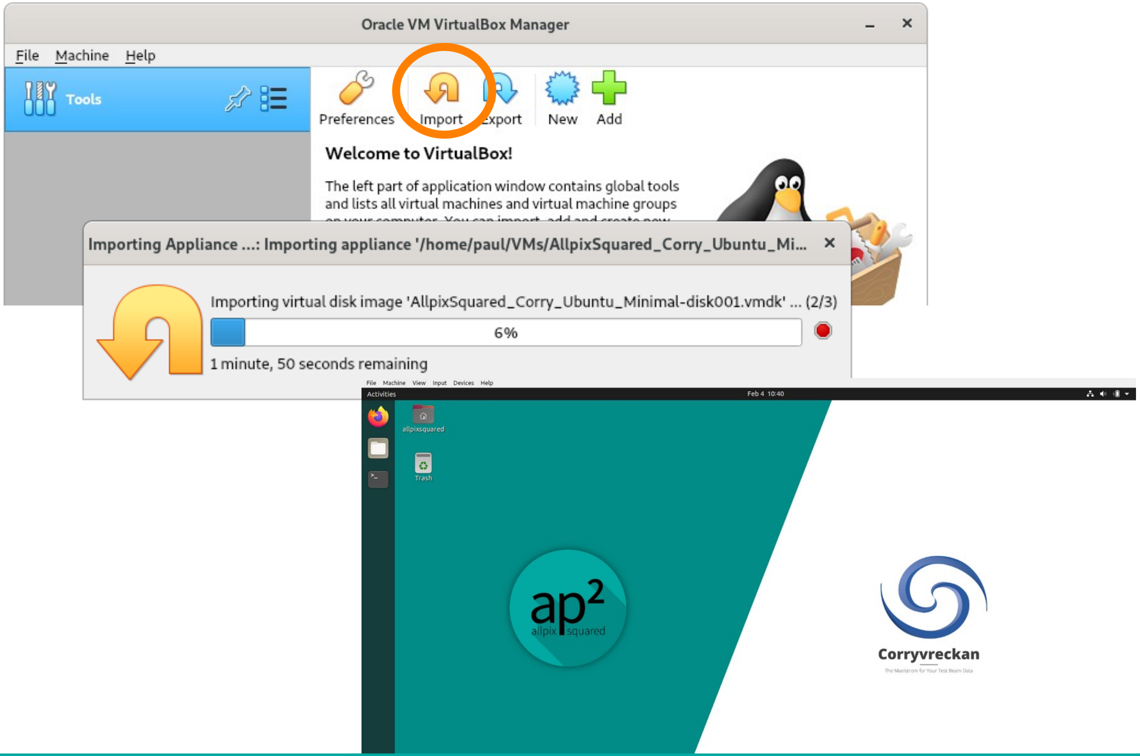
Installation
&
Sources

VirtualBox

- For the BTTB tutorials on *Corryvreckan* and *Allpix Squared* we created a Virtual Box
- Virtualisation software: Virtualises a physical machine and lets you run a PC on a PC
- Example:
  - Run a Linux machine as an application within your Windows PC
  - Host allocates configurable amount of memory/CPU/disk space to virtual machine
  - VirtualBox:
    - Free of charge
    - Plenty of useful features

VirtualBox

- Install VirtualBox (via package manager or https://www.virtualbox.org/wiki/Downloads)
- Import the downloaded virtual machine
  - Default options are typically fine – adjust if necessary
- Go! (double click on the new virtual box)



CVMFS – CernVM File System

- Central installation of software for CentOS7
- On any machine with CVMFS, simply *source* corresponding script and use the SW
- Many packages available: ROOT, Geant4, LCIO, Delphes, FastJet, …

… Allpix Squared

CVMFS – CernVM File System

```
drwxrwxr-x 4 cvmfs cvmfs   4  9. Jul 2019  [illegible]
drwxrwxr-x 4 cvmfs cvmfs   4 13. Sep 2019  1.4.1
drwxrwxr-x 4 cvmfs cvmfs   4 26. Nov 2019  1.4.2
drwxrwxr-x 4 cvmfs cvmfs   4 13. Jan 2020  1.4.3
drwxrwxr-x 4 cvmfs cvmfs   4 10. Mär 2020  1.4.4
drwxr-xr-x 4 cvmfs cvmfs  65 14. Apr 2020  1.5.0
drwxr-xr-x 4 cvmfs cvmfs  65 24. Jul 2020  1.5.1
drwxr-xr-x 4 cvmfs cvmfs  65 14. Sep 2020  1.5.2
drwxr-xr-x 4 cvmfs cvmfs  66 29. Okt 2020  1.6.0
drwxr-xr-x 4 cvmfs cvmfs  66 28. Jan 2021  1.6.1
drwxr-xr-x 6 cvmfs cvmfs 135  1. Apr 2021  1.6.2
drwxr-xr-x 6 cvmfs cvmfs 139 10. Jun 2021  2.0.0
drwxr-xr-x 6 cvmfs cvmfs 139  9. Jul 2021  2.0.1
drwxr-xr-x 6 cvmfs cvmfs 139 23. Sep 2021  2.0.2
drwxr-xr-x 6 cvmfs cvmfs 137 28. Okt 2021  2.0.3
drwxr-xr-x 6 cvmfs cvmfs 137 17. Nov 2021  2.1.0
drwxr-xr-x 6 cvmfs cvmfs 137 21. Dez 20:21 2.1.1
drwxr-xr-x 6 cvmfs cvmfs 137 20. Jan 14:42 2.1.2
drwxr-xr-x 6 cvmfs cvmfs 137 28. Feb 18:53 2.2.0
drwxr-xr-x 6 cvmfs cvmfs 137 22. Mär 13:58 2.2.1
drwxr-xr-x 6 cvmfs cvmfs 137  1. Apr 16:56 2.2.2
drwxr-xr-x 6 cvmfs cvmfs 137 16. Mai 17:29 2.3.0
drwxr-xr-x 6 cvmfs cvmfs 137  8. Jun 21:44 latest
/cvmfs/clicdp.cern.ch/software/allpix-squared%
```

- Using project space of CLICdp at *//cvmfs/clicdp.cern.ch/software/allpix-squared/*
- All versions since v1.1 available
  - Nightly build of master in “latest”
- Each version built for CentOS7/8 (gcc/clang)
  - /2.3.0/x86_64-centos8-gcc11-opt/
  - /2.3.0/x86_64-centos7-gcc11-opt/
- Load all dependencies, C++ libraries & set up $PATH using setup.sh file:

```
$ source /cvmfs/clicdp.cern.ch/software/allpix-squared/2.3.0/x86_64-centos7-gcc11-opt/setup.sh
$ allpix --version
Allpix Squared version v2.3.0
            built on 2022-05-16, 14:20:06 UTC
            ...
```

Compiling from Source I

- Satisfy **dependencies** first!
- Hard dependency:
  - ROOT6 - objects, object history, storing/reading from/to files
  - Boost.Random - system independent random number generator
- Soft dependencies:
  - Geant4 - particle interaction with matter & tracking *(charge deposition)*
  - Eigen - library for fast algebra, used by some modules (*propagation*)
- With access to CVMFS (as on LXPLUS / NAF), all is ready, simply do …

```
$ source etc/scripts/setup_lxplus.sh
```

Compiling from Source II

- Get the **code**!
- A reminder: all resources are linked to from the project page:

  https://cern.ch/allpix-squared/

- First of all, check out the Allpix-squared repository into a local directory “allpix-squared”
- Move to this directory, and source the setup script for lxplus

```
$ git clone https://gitlab.cern.ch/allpix-squared/allpix-squared.git allpix-squared
$ cd allpix-squared
```

Compiling from Source III

- We use **CMake** to configure the build
  - Cross-platform, same scripts on Linux & Mac OS
  - Weird-to-write but fairly easy to use/run
- Prefer out-of-source builds: make new folder to compile in

```
$ cmake /path/to/allpix-squared/
```

- Options can be set using “-D...=...” e.g.

```
$ cmake -DBUILD_VisualizationGeant4=OFF /path/to/allpix-squared/
```

- Graphical/ncurses tools can aid in configuration, try:

```
$ ccmake /path/to/allpix-squared/
```

Compiling from Source IV

- Now we can **build** it!

```
$ make -j16
$ make install
```

- Grab an espresso (but hurry up!)
- Run simulations!

Knock, knock – Who's there?

CMakeLists.txt	Instructions for cmake to prepare Allpix Squared compilation
CONTRIBUTING.md	A guide to developers for contributing code
LICENSE.md	The Allpix Squared licence (open source, MIT)
README.md	Instructions for getting started, installation locations, authors
3rdparty/	Included & required 3rd party software
cmake/	Macros for cmake, formatting tools to make code style consistent
doc/	Documentation including user manual (alternatively see website)
etc/	Selection of things like scripts for making new modules (see later), unit tests, etc
examples/	Documented examples, useful for setting up new simulations
models/	Detector models which can be included in geometry
src/	The main directory for c++ code, including the core software and all modules
tools/	External tools, for example to convert TCAD output, bundled with the framework

Simulation Modules

```
~/software/allpix-squared/src/modules $ ls
CapacitiveTransfer   DepositionPointCharge  GeometryBuilderGeant4  ROOTObjectWriter
CMakeLists.txt       DepositionReader       InducedTransfer        SimpleTransfer
CorryvreckanWriter   DetectorHistogrammer   LCIOWriter             TextWriter
CSADigitizer         DopingProfileReader    MagneticFieldReader    TransientPropagation
DatabaseWriter       Dummy                  ProjectionPropagation  VisualizationGeant4
DefaultDigitizer     ElectricFieldReader    PulseTransfer          WeightingPotentialReader
DepositionCosmics    GDMLOutputWriter       RCEWriter
DepositionGeant4     GenericPropagation     ROOTObjectReader
```

- Most important for today:
  - *GeometryBuilderGeant4* – Builds the geometry for Geant4 from configuration
  - *DepositionGeant4* – Uses Geant4 for particle propagation and energy deposition
  - *GenericPropagation* – Propagates charge carriers through the sensor volume
  - *DefaultDigitizer* – Describes the digitization in front end electronics
  - *DetectorHistogrammer* – Creates useful histogramms for detector charecterisation

Getting Started

Your first Simulation – The Configuration

- Let’s look at our first configuration file: **first-simulation.conf**
  - Hint: I usually create a new directory for a new simulation, either outside the source directory or in *allpix-squared/conf/*
- Syntax:
  - [Section] – This can be global parameters ([Allpix]) or a [module]
  - key = value – Key-value pairs that belong to the last mentioned section
  - Many different types can be input via the config files - strings, integers, doubles, vectors/arrays, etc
- Global parameters are always required
  - Number of events
  - Geometry file

```
[Allpix]
number_of_events = 1000
detectors_file = "tutorial-geometry.conf"
```

Your first Simulation – The Geometry

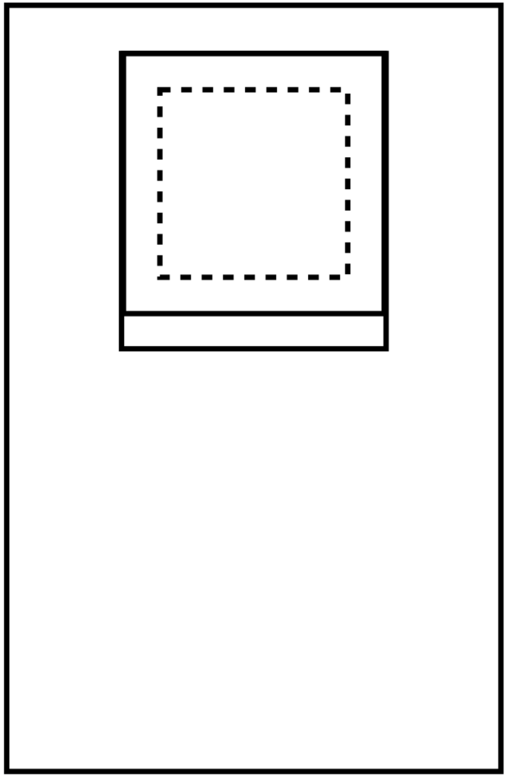
- The second required file is a geometry description
- Let's look at **tutorial-geometry.conf**
- The geometry configuration file determines which detector is used
  - Each detector is given a unique name (*detector1* here) and placed in the global coordinate system at a certain position with a given rotation
- The currently known detectors are listed in the *models* path
  - A new detector model can be built, or an existing detector used
  - For this example, we will pick the **timepix** model

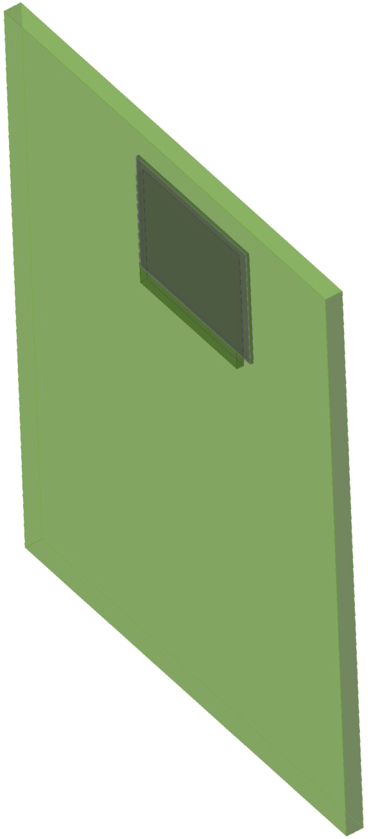
```
[detector1]
type = "timepix"
position = 0mm 0mm 0mm
orientation = 0 0 0
```

```
~/software/allpix-squared/models $ ls
alpide.conf
alpide_excess.conf
alpide_no_support.conf
alpide_no_support_rot.conf
clicpix2.conf
clicpix.conf
CMakeLists.txt
cmspl.conf
diode.conf
fei3.conf
ibl_planar.conf
medipix3.conf
mimosa23.conf
mimosa26.conf
mimosa26_no_support.conf
test.conf
timepix.conf
~/software/allpix-squared/models $
```

The Model: timepix.conf

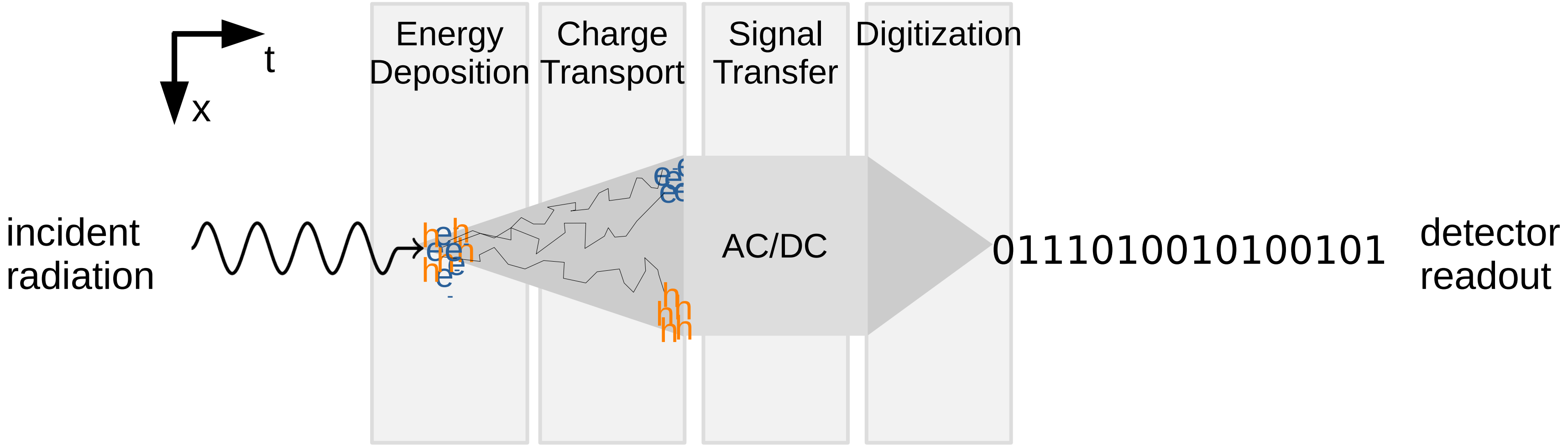
```
type = "hybrid"

number_of_pixels = 256 256
pixel_size = 55um 55um

sensor_thickness = 300um
sensor_excess = 1mm

bump_sphere_radius = 9.0um
bump_cylinder_radius = 7.0um
bump_height = 20.0um

chip_thickness = 700um
chip_excess_left = 15um
chip_excess_right = 15um
chip_excess_bottom = 2040um

[support]
thickness = 1.76mm
size = 47mm 79mm
offset = 0 -22.25mm
```





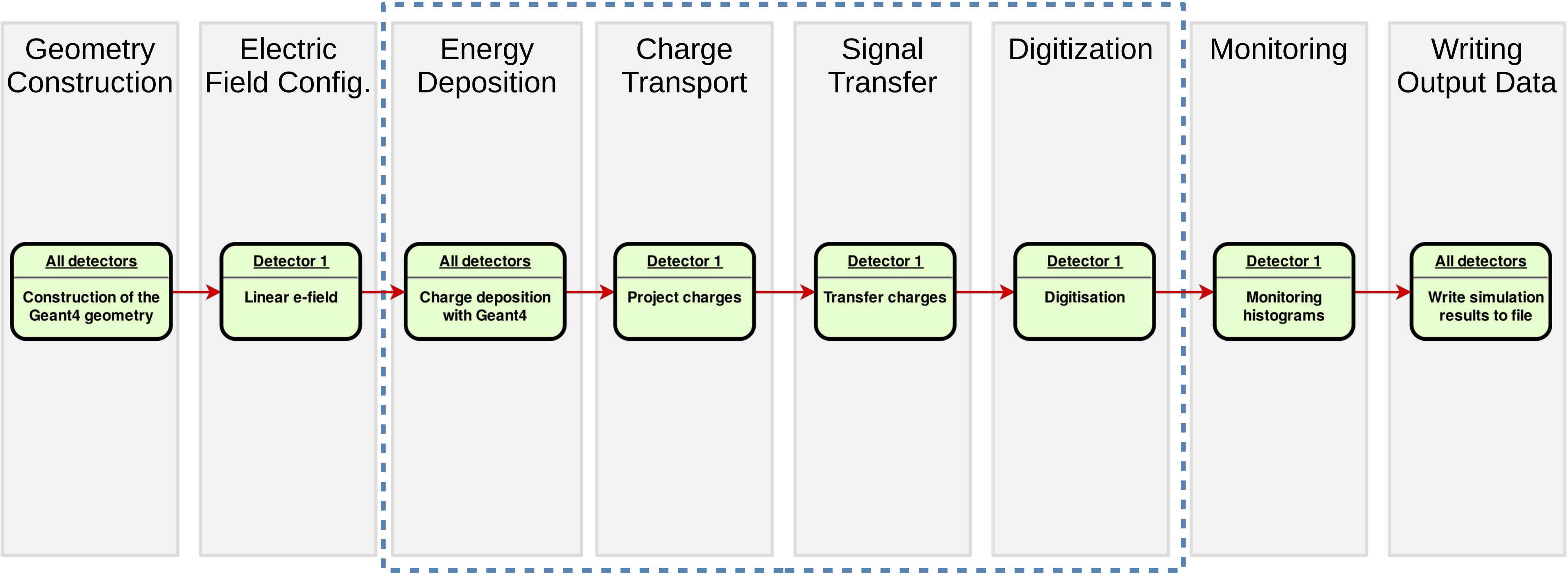
Adding Physics

- We can now start to add algorithms, or *modules*
  - Simply done by including a **[section]** in the main configuration file
  - Parameters for each algorithm are added within the corresponding section block
- Most simulations involve the same concepts …



Adding Physics

- Allpix Squared: one *module* for each *task*



Your first Simulation

- Let’s include some modules in our **first-simulation.conf** ...

- This won’t run yet as not all parameters have default values …

```
[Allpix]
number_of_events = 1000
detectors_file = "tutorial-geometry.conf"

[GeometryBuilderGeant4]

[DepositionGeant4]

[ElectricFieldReader]

[ProjectionPropagation]

[SimpleTransfer]

[DefaultDigitizer]

[DetectorHistogrammer]
```

Module Parameters

- All modules are thoroughly described in the user manual https://cern.ch/allpix-squared/usermanual/allpix-manualch7.html
    - List of available parameters for corresponding module with default values
    - Usage examples

Description

Converts all object data stored in the ROOT data file produced by the ROOTObjectWriter module back in to messages (see the description of ROOTObjectWriter for more information about the format). Reads all trees defined in the data file that contain Allpix objects. Creates a message from the objects in the tree for every event.

If the requested number of events for the run is less than the number of events the data file contains, all additional events in the file are skipped. If more events than available are requested, a warning is displayed and the other events of the run are skipped.

Currently it is not yet possible to exclude objects from being read. In case not all objects should be converted to messages, these objects need to be removed from the file before the simulation is started.

Parameters

- file_name : Location of the ROOT file containing the trees with the object data. The file extension .root will be appended if not present.
- include : Array of object names (without allpix:: prefix) to be read from the ROOT trees, all other object names are ignored (cannot be used simultaneously with the exclude parameter).
- exclude: Array of object names (without allpix:: prefix) not to be read from the ROOT trees (cannot be used simultaneously with the include parameter).
- ignore_seed_mismatch: If set to true, a mismatch between the core random seed in the configuration file and the input data is ignored, otherwise an exception is thrown. This also covers the case when the core random seed in the configuration file is missing. Default is set to false.

Usage

This module should be placed at the beginning of the main configuration. An example to read only PixelCharge and PixelHit objects from the file data.root is:

```
[ROOTObjectReader]
file_name = "data.root"
include = "PixelCharge", "PixelHit"
```

The World

- **GeometryBuilderGeant4**
- Translates and defines the geometry to Geant4
- No required parameters
  - By default the setup is placed in air

```
[GeometryBuilderGeant4]
```

The Particles

- **DepositionGeant4**
- Interface to Geant4
- Definition of particles and tracking through the setup
    - Energy deposition in sensitive material
- Pick …
    - the type of particles
    - the particle energy
    - the origin and direction of the beam
    - the shape and size of the beam
    - a suitable physics list

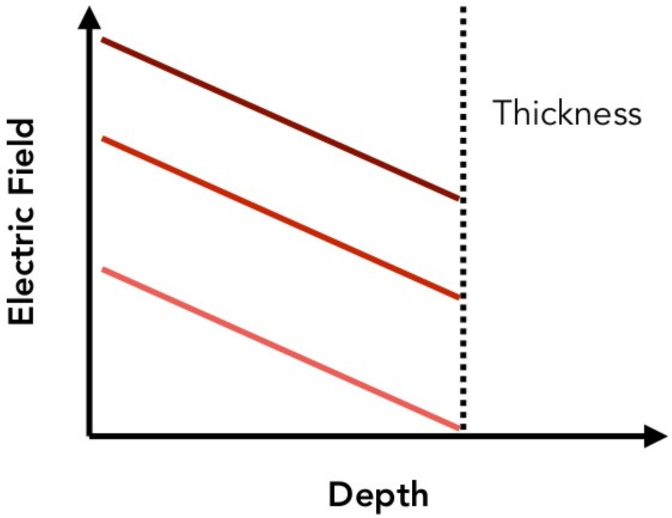
```
[DepositionGeant4]
particle_type = "e-"
source_energy = 5GeV
source_type = "beam"
beam_size = 3mm
source_position = 0 0 -200mm
beam_direction = 0 0 1
physics_list = FTFP_BERT_EMZ
```

Don't forget the units!

The Electric Field

- **ElectricFieldReader**
- Generation of an electric field in the sensor
- A *Linear* field is the simplest approximation, using a user-defined depletion and bias voltage
  - Higher bias voltages increase the electric field
  - No focussing effects around the implants
- More realistic fields can be added by converting the output of e.g. TCAD simulations

```
[ElectricFieldReader]
model = "linear"
bias_voltage = -150V
depletion_voltage = -30V
```



The Propagation

- **GenericPropagation**
- Propagate deposited charges towards the electrodes
- Charges are propagated to the sensor surface in discrete groups (default: 10)
  - 5th order Runge-Kutta-Fehlberg integration
  - Per step:
    - Drift depending on electric/magnetic field
    - Diffusion depending on step time
    - Determination of recombination and/or trapping probability

```
[GenericPropagation]
temperature = 293K
```

The Transfer

- Not all propagated charge carriers will necessarily end up on the collection implant due to …
  - Low-field regions
  - Linear field approximation
- Transferring the charge from the sensor to the input of the electronics
- **SimpleTransfer**

```
[SimpleTransfer]
```

- In this approximation, all charges within x microns to the implant are considered as collected (default: 5 µm)

The Digitisation

- Many front-end chips feature similar kinds of effects
  - Gaussian noise on the collected charge
  - A threshold level
  - A QDC with a certain gain & resolution
- **DefaultDigitizer**
- Implementation of above features plus …
  - Threshold dispersion
  - QDC smearing
  - TDC / ToA / ToT calculation if pulses are simulated

```
[DefaultDigitizer]
qdc_slope = 200e
qdc_resolution = 8
```

The Histogrammer

- **DetectorHistogrammer**
- Creation of typical detector performance plots including a simple clustering technique and comparison the Monte Carlo truth, like …
  - Hit map
  - Pixel & cluster charge distribution
  - Cluster size
  - Efficiency (vs MC truth)
  - Residuals (vs MC truth)
- These plots can give a good and quick overview over the detector performance, *but they do not replace a thorough analysis*

```
[DetectorHistogrammer]
```

Updated Configuration

- Now we have a simulation setup that will shoot *5 GeV electrons* at a *timepix* detector, *project* the charges to the surface based on a *linear field approximation* and *digitises* the collected charges
- Two hints before we start …
  - The **log_level** flag changes the detail (and quantity) of information output by modules (global and/or module parameter)
  - The **output_plots** flag can be set individually for modules to get additional plots
- Run the simulation …

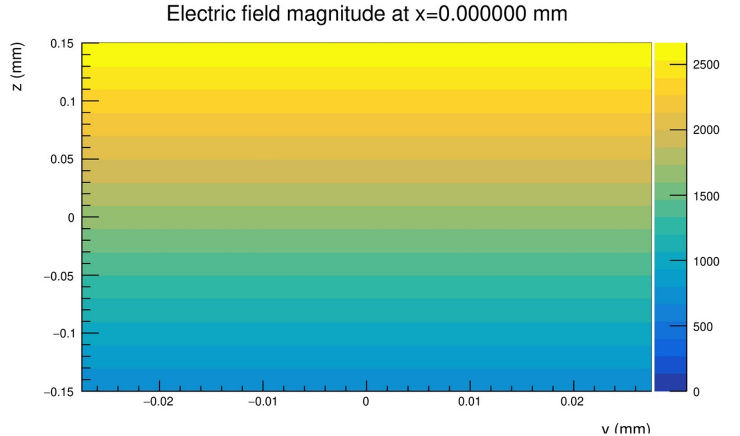
```
$ ../bin/allpix -c first-simulation.conf
```

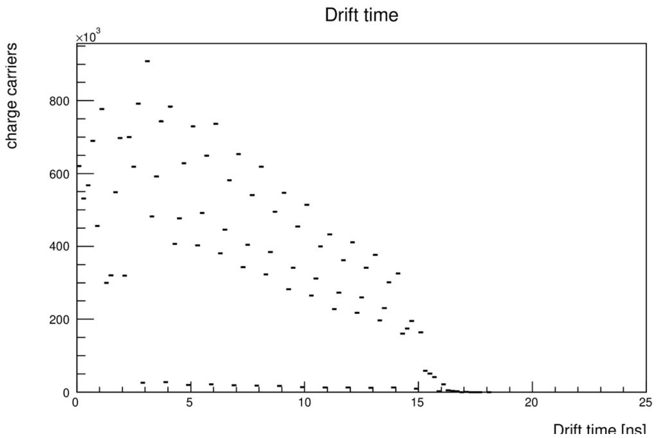
```
[Allpix]
number_of_events = 1000
detectors_file = "tutorial-geometry.conf"
log_level = "STATUS"

[GeometryBuilderGeant4]

[DepositionGeant4]
particle_type = "e-"
source_energy = 5GeV
source_type = "beam"
beam_size = 3mm
source_position = 0 0 -200mm
beam_direction = 0 0 1
physics_list = FTFP_BERT_EMZ
output_plots = true

[ElectricFieldReader]
model = "linear"
bias_voltage = -150V
depletion_voltage = -30V
output_plots = true

[GenericPropagation]
temperature = 293K
output_plots = true

[SimpleTransfer]
output_plots = true

[DefaultDigitizer]
qdc_slope = 200e
qdc_resolution = 8
output_plots = true

[DetectorHistogrammer]
```

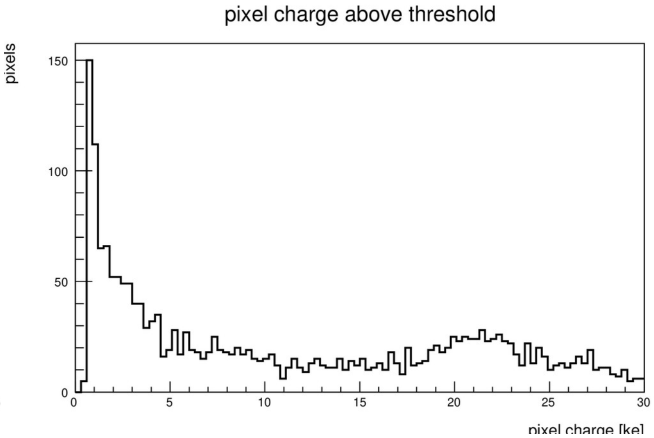
Plots, plots, plots ...

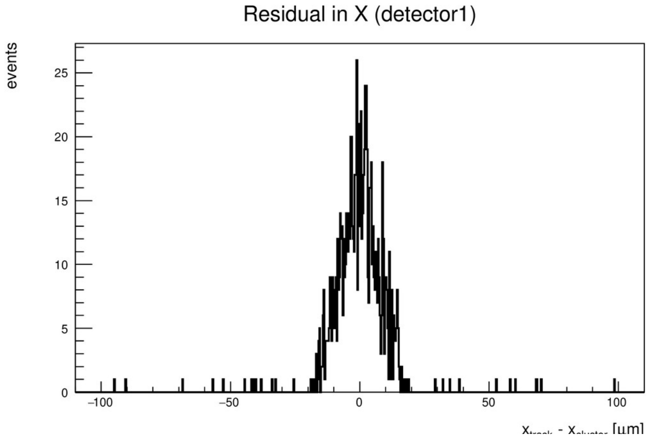
- The output is (if not configured otherwise) located in *output/modules.root*
- Let's have a look …

```
$ root -l output/modules.root
```









Task: Optimize the Resolution

- Determine the resolution and **improve it** by …
  - Changing the incidence angle of particles (*orientation*)
  - Changing the magnetic field (*magnetic_field*)

- Hint for speeding up the process:
  Change individual parameters from the command line …

```
$ allpix -c config.conf -o MyModule.your_parameter=2GeV -g detector5.position=0,0,10m
```

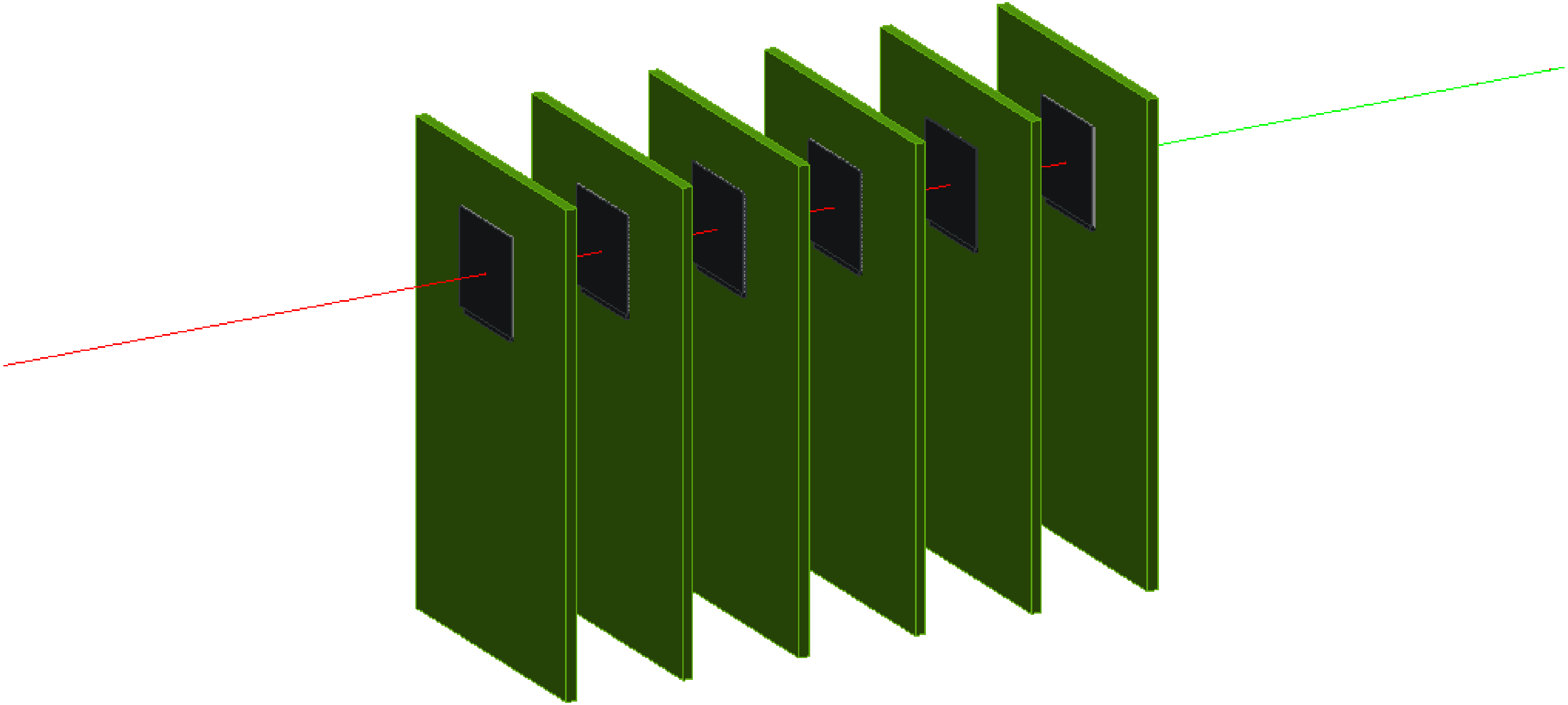
Augmenting Simulations

Adding Detectors

- Similar to the single detector geometry, subsequent detectors can be added
  - Avoid overlaps between detectors/materials
- Here: Let's simulate a timepix telescope with a spacing of 20 mm in z
- For more complex geometries, a visualisation of the setup using the Geant4 visualisation tools is extremely useful `[VisualizationGeant4]`
  - Unfortunately these do not run on lxplus – tools not included in the default CVMFS installation

```
[detector1]
type = "timepix"
position = 0mm 0mm 0mm
orientation = 0 0 0

[detector2]
type = "timepix"
position = 0mm 0mm 20mm
orientation = 0 0 0

[detector3]
type = "timepix"
position = 0mm 0mm 40mm
orientation = 0 0 0

[detector4]
type = "timepix"
position = 0mm 0mm 60mm
orientation = 0 0 0

[detector5]
type = "timepix"
position = 0mm 0mm 80mm
orientation = 0 0 0

[detector6]
type = "timepix"
position = 0mm 0mm 100mm
orientation = 0 0 0
```

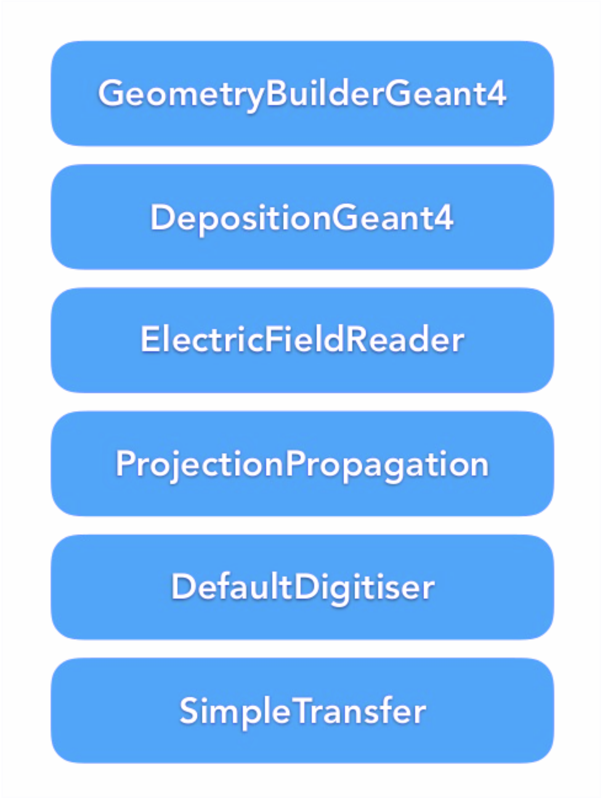
Visualising the Setup



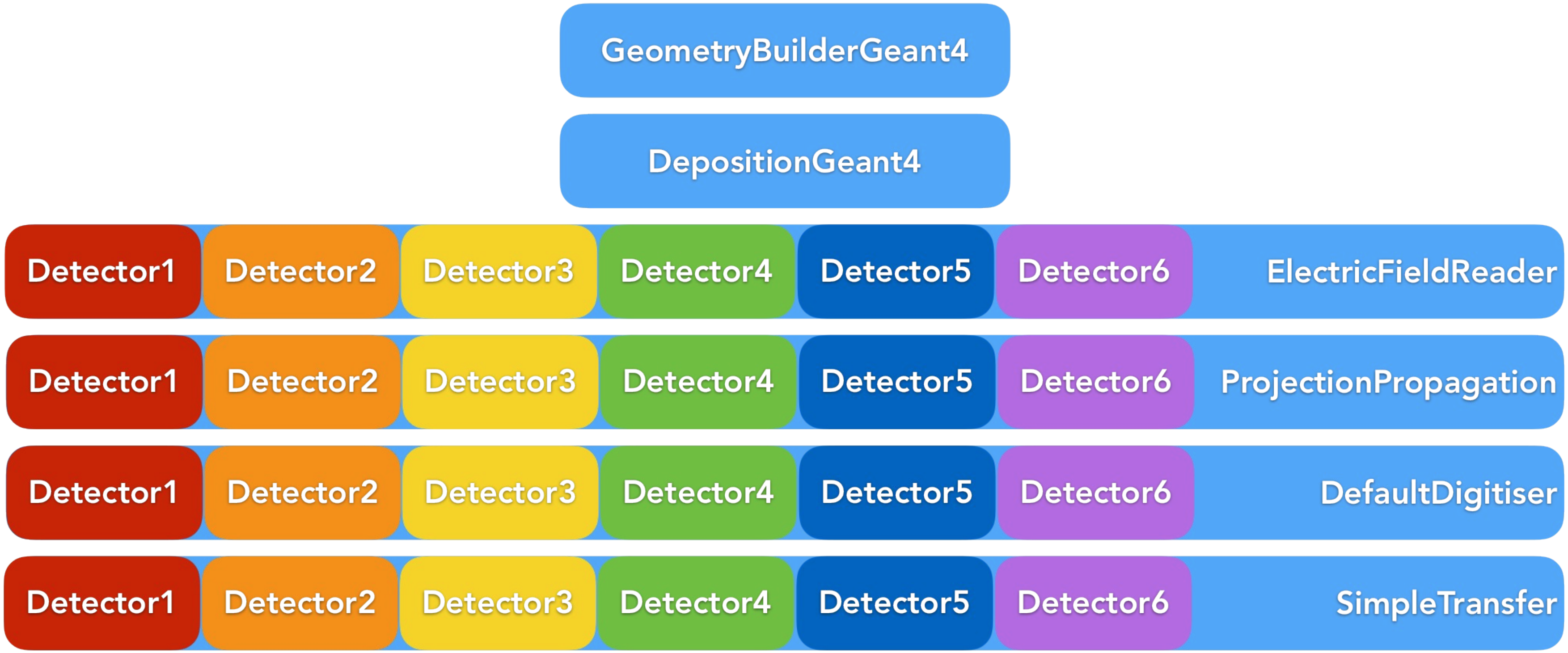
Treating Detectors in Different Ways

- When we added more detectors to the geometry file, everything was taken care of under the hood
- What is happening in the background?
  - One instance of a module is created per detector
  - This allows e.g. to use multi-threading, running the same module for different detectors in parallel
- Background information: modules can be either **unique** or specific to one **detector**
- What if we would like to use different parameters for different detectors, though?

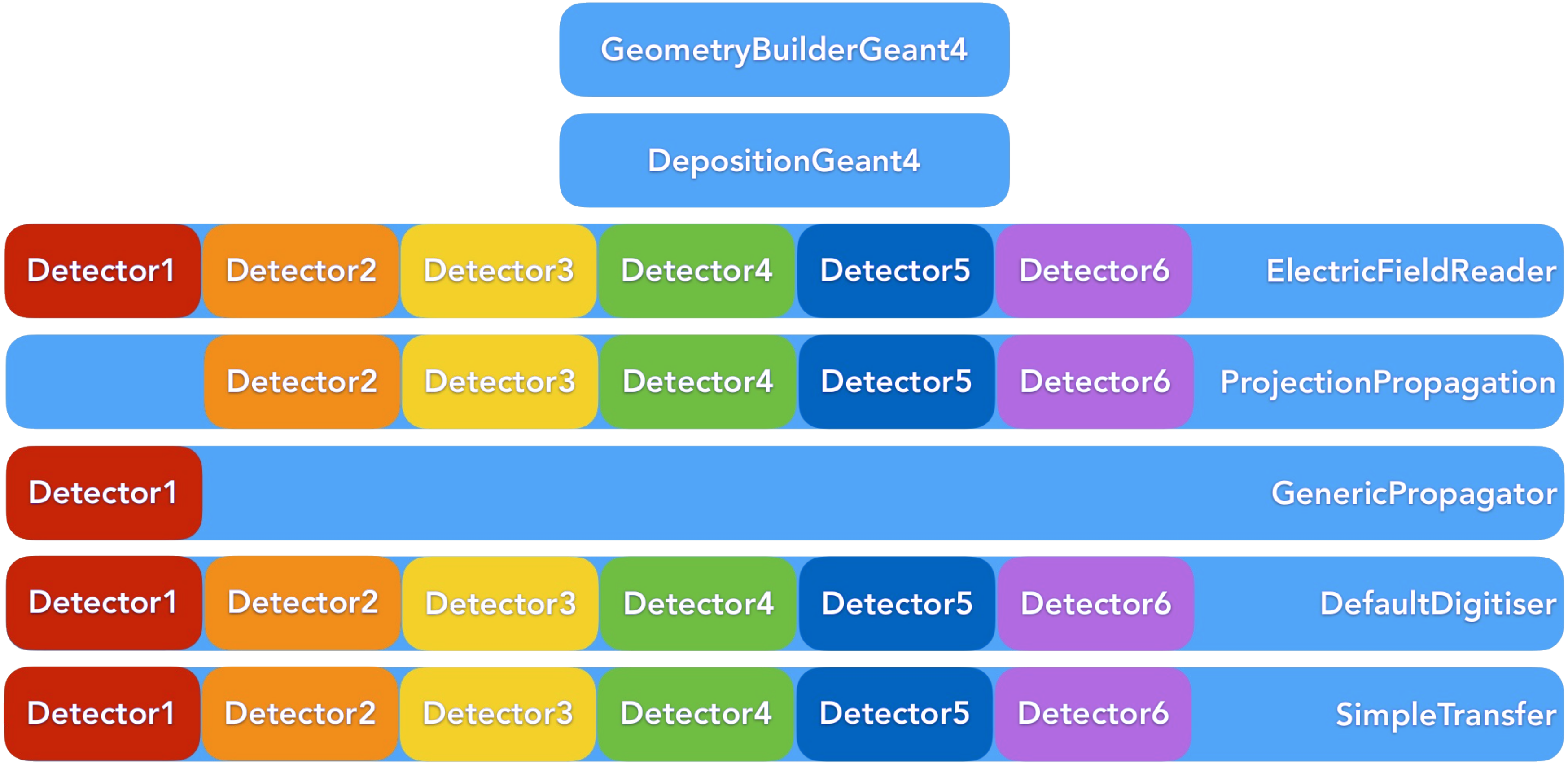
A Single Detector Simulation Chain



A Multi-Detector Simulation Chain



Now what if I … ?


GeometryBuilderGeant4
DepositionGeant4
Detector1
Detector2
Detector3
Detector4
Detector5
Detector6
ElectricFieldReader
Detector2
Detector3
Detector4
Detector5
Detector6
ProjectionPropagation
Detector1
GenericPropagator
Detector1
Detector2
Detector3
Detector4
Detector5
Detector6
DefaultDigitiser
Detector1
Detector2
Detector3
Detector4
Detector5
Detector6
SimpleTransfer

Split Module Configurations

- By default, all instances of a module will apply to all detectors
- We can overwrite this by specifying either the *name* or the *type* of the detectors
- Make a module operate **on one detector only** or on a subset of detectors

Default electric field configuration

```
[ElectricFieldReader]
model = "linear"
bias_voltage = -50V
depletion_voltage = -30V
```

Overwritten for *detector1*

```
[ElectricFieldReader]
name = "detector1"
model = "linear"
bias_voltage = -100V
depletion_voltage = -30V
```

Works great on *Telescope + DUT* simulations!

Split Module Configurations

- By default, all instances of a module will apply to all detectors
- We can overwrite this by specifying either the *name* or the *type* of the detectors
- Make a module operate on one detector only or **on a subset of detectors**

Subset given as list of detectors

```
[ProjectionPropagation]
name = "detector2", "detector3", "detector4", "detector5", "detector6"
temperature = 293K
charge_per_step = 10
```

Specific for *detector1*

```
[GenericPropagation]
name = "detector1"
temperature = 293K
charge_per_step = 10
```

Updated Simulation Configuration

```
[Allpix]
number_of_events = 1000
detectors_file = "tutorial-geometry.conf"
log_level = "WARNING"

[GeometryBuilderGeant4]

[DepositionGeant4]
particle_type = "e-"
source_energy = 5GeV
source_type = "beam"
beam_size = 3mm
source_position = 0 0 -200mm
beam_direction = 0 0 1
physics_list = FTFP_BERT_EMZ
output_plots = true

[ElectricFieldReader]
model = "linear"
bias_voltage = -50V
depletion_voltage = -30V
output_plots = true

[ElectricFieldReader]
name = "detector1"
model = "linear"
bias_voltage = -100V
depletion_voltage = -30V
output_plots = true

[ProjectionPropagation]
name = "detector2", "detector3", "detector4", "detector5", "detector6"
temperature = 293K
charge_per_step = 10
output_plots = true

[GenericPropagation]
name = "detector1"
temperature = 293K
charge_per_step = 10
output_plots = true

[SimpleTransfer]
output_plots = true

[DefaultDigitizer]
output_plots = true

[DetectorHistogrammer]
```

Simulation Replays

- Imagine:
  "I would like to perform a threshold scan for my detector, but running the full chain 20 times is just time consuming…"
- In each simulation step, *objects* are generated: *MCParticle, DepositedCharge, PropagatedCharge, PixelHit, …*
- The modules *RootObjectWriter* and *-Reader* store and read these objects
  - This allows to pick up on your simulation at a later point in time

Simulation Replays

```
[detector1]
type = "timepix"
position = 0mm 0mm 0mm
orientation = 0 0 0
```

```
[Allpix]
number_of_events = 1000
detectors_file = "tutorial-geometry.conf"
log_level = "WARNING"
random_seed_core = 1234

[GeometryBuilderGeant4]

[DepositionGeant4]
particle_type = "e-"
source_energy = 5GeV
source_type = "beam"
beam_size = 3mm
source_position = 0 0 -200mm
beam_direction = 0 0 1
physics_list = FTFP_BERT_EMZ
output_plots = true

[ElectricFieldReader]
model = "linear"
bias_voltage = -50V
depletion_voltage = -30V
output_plots = true

[GenericPropagation]
name = "detector1"
temperature = 293K
charge_per_step = 10
output_plots = true

[SimpleTransfer]
output_plots = true

[ROOTObjectWriter]
file_name = "prepared_data.root"
include = "MCParticle", "PixelCharge"
```

```
[Allpix]
number_of_events = 1000
detectors_file = "tutorial-geometry.conf"
log_level = "WARNING"
random_seed_core = 1234

[ROOTObjectReader]
file_name = "output/prepared_data.root"
include = "MCParticle", "PixelCharge"

[DefaultDigitizer]
threshold = 400e
output_plots = true

[DetectorHistogrammer]
```

```
[DefaultDigitizer]
threshold = 800e
output_plots = true
```

```
[DefaultDigitizer]
threshold = 1200e
output_plots = true
```

Or try …

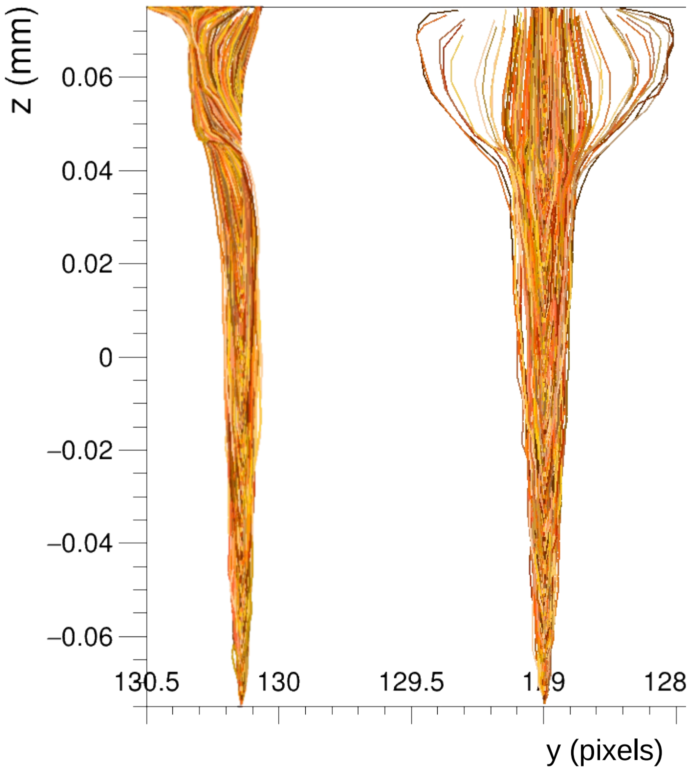
```
$ allpix -c tutorial-replay.conf
         -o DefaultDigitizer.threshold=1200e
```

TCAD Interface

Adding Precision

- Importing results from TCAD simulations can drastically improve the precision of a sensor simulation

- Electric fields → Propagation
- Weighting potentials → Signal induction
- Doping profiles → Recombination



Mesh Converter

- Converter available:
  TCAD results → Allpix Squared readable
  - TCAD results can have an irregular mesh
  - ➔ Computing-intense, often not needed
  - ➔ Convert to regular mesh
- Input formats
  - DF-ISE file format (*.grd* and *.dat* required)
  - Silvaco TCAD
- Output formats
  - APF: Allpix Squared Field (binary)
  - INIT: ASCII text file

13.2 Mesh Converter

This code takes adaptive meshes from finite-element simulations and transforms them into a regularly spaced grid for faster field value lookup as reuqired by Monte Carlo simulations tools such as Allpix Squared. The input consists of two files, one containing the vertex coordinates of each input mesh node, the other providing the relevant field values associated to each of these vertices. One output file containing the regular interpolated mesh is produced.

A new regular mesh is created by scanning the model volume in regular X Y and Z steps (not necessarily coinciding with original mesh nodes) and using a barycentric interpolation method to calculate the respective electric field vector on the new point. The interpolation uses the four closest, no-coplanar, neighbor vertex nodes such, that the respective tetrahedron encloses the query point. For the neighbors search, the software uses the Octree implementation [85].

13.2.1 File Formats

Input Data

Currently, this tool supports the TCAD DF-ISE data format and requires the .grd and .dat files as input. Here, the .grd file contains the vertex coordinates (3D or 2D) of each mesh node and the .dat file contains the value of each electric field vector component for each mesh node, grouped by model regions (such as silicon bulk or metal contacts). The regions are defined in the .grd file by grouping vertices into edges, faces and, consecutively, volumes or elements.

Output Data

This tools can produce output in two different formats, with the file extensions .init and .apf. Both file formats can be imported into Allpix Squared.

The APF (Allpix Squared Field) data format contains the field data in binary form and is therefore a bit more compact and can be read much faster. Whenever possible, this format should be preferred.

The INIT file is an ASCII text file with a format used by other tools such as PixelAV. Its header therefore contains several fields which are not used by Allpix Squared but need to be present nevertheless. The following example shows such a file header, important variables are marked with <...> while other fields are not interpreted and can be left untouched:

Mesh Converter

- Driven bei configuration file
  Example: ***convert.conf***

- Provide …
  - Output format
  - Region of the mesh that should be converted (data often contains non-silicon parts that can be omitted)
  - Units & name of observable to convert
  - Orientation of the axes
  - Dimensions of output mesh
  - Multithreading?

```
model = apf
region = "n-bulk"
observable = "ElectricField"
xyz = x y  z
divisions = 220 600
observable_units = V/cm


workers = 4
```

Converting an Electric Field

- Execute like this …

```
$ mesh_converter -c convert.conf -f timepix_like_tcad
```

  - *-f* provides the input file name prefix
  - ➔ Grid read from *timepix_like_tcad.grd*
  - ➔ Data read from *timepix_like_tcad.dat*

- Let's have a look at the field:

```
$ mesh_plotter -f timepix_like_tcad_ElectricField.apf
```

Converting an Electric Field

- Execute like this …

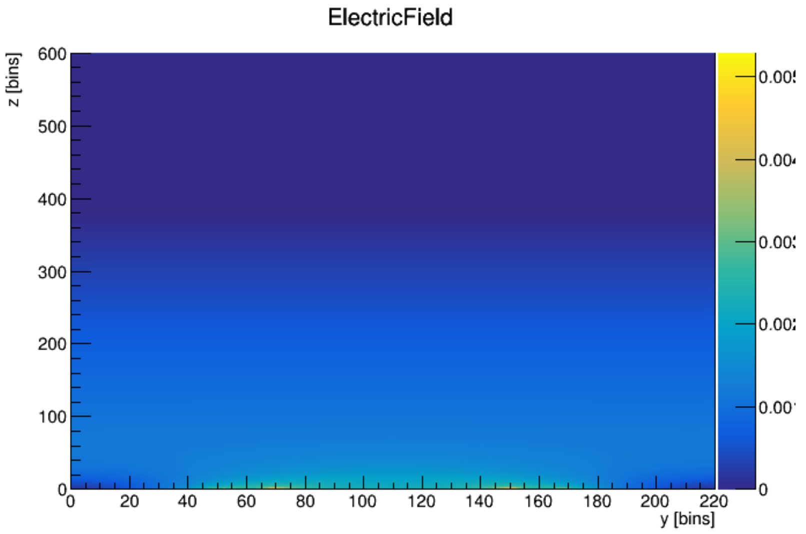
```
$ mesh_converter -c convert.conf -f timepix_like_tcad
```

  - *-f* provides the input file name prefix
  - ➔ Grid read from *timepix_like_tcad.grd*
  - ➔ Data read from *timepix_like_tcad.dat*

- Let's have a look at the field:

```
$ mesh_plotter -f timepix_like_tcad_ElectricField.apf
```

➔ Umm… Well…



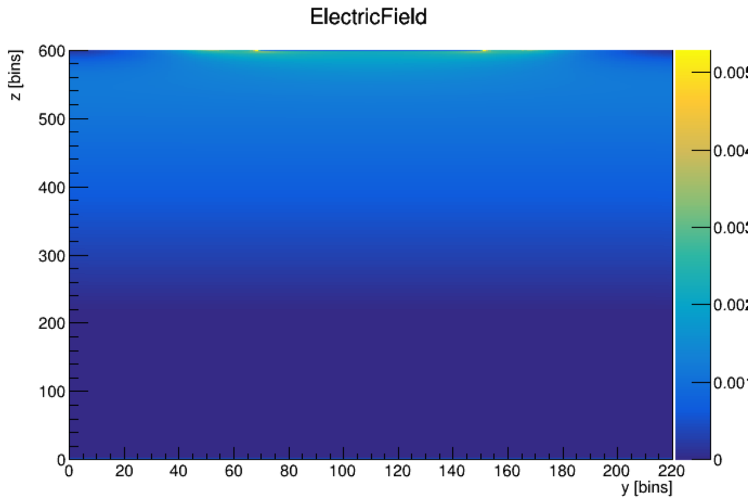
Mesh Converter

- ➔ In Allpix Squared, the implant is always defined to be at positive *z*!

- Can correct for the orientation in the conversion configuration file

```
model = apf
region = "n-bulk"
observable = "ElectricField"
xyz = x y -z
divisions = 220 600
observable_units = V/cm

workers = 4
```

- Note:
  The *mesh_plotter* uses framework internal units.
  Electric field: *MV/mm*



Loading APF Files

- Adapt the *ElectricFieldReader*

```
[ElectricFieldReader]
model = "mesh"
file_name = "../fields2/timepix3_fields/timepix_like_tcad_ElectricField.apf"
output_plots = true
```

- Note:
  If the electric field spans only the top part of the sensor, the parameter *depletion_depth* can be used to constrain the electric field to this part of the sensor

Example Configuration

- Let’s make a few adjustments to the configuration:
  tcad-field-simulation.conf

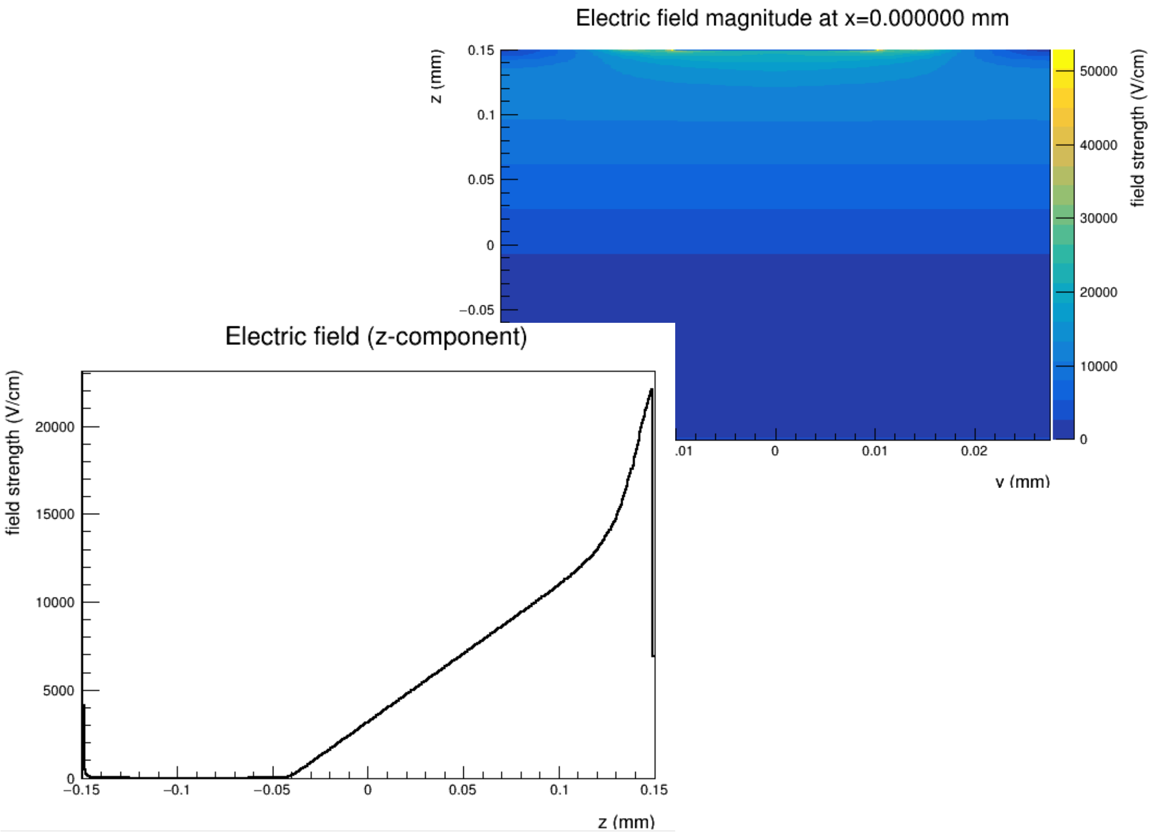
- *DepositionPointCharge*:
  Deposits charge carriers along a linear path (in this case);
  control over the incidence position & simulate “ideal” particles

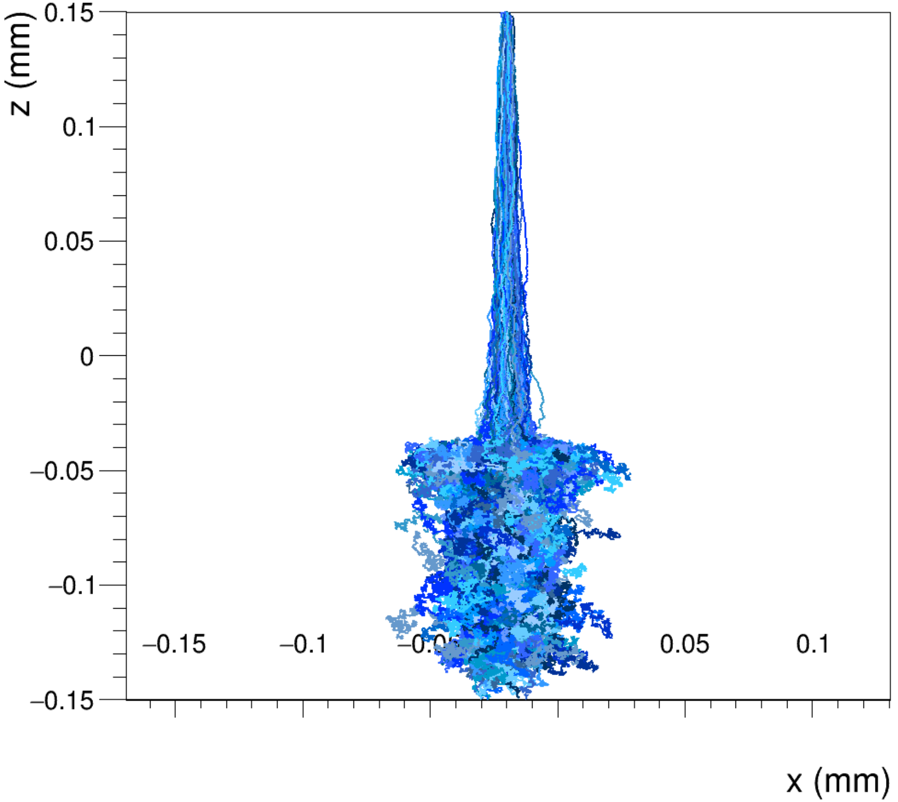
- *GenericPropagation*:
  Enable linegraphs, reduce timesteps, add mobility model

- Note:
  Linegraphs are rather computing & memory intensive.
  We recommend enabling these for individual events only

```
[Allpix]
number_of_events = 1
detectors_file = "tutorial-geometry.con
log_level = "STATUS"

[GeometryBuilderGeant4]

[DepositionPointCharge]
source_type = "mip"
model = "fixed"
position = -20um 20um
number_of_steps = 10

[ElectricFieldReader]
model = "mesh"
file_name = "../TCAD_fields/timepix_lik
output_plots = true

[GenericPropagation]
temperature = 293K
charge_per_step = 20
output_linegraphs = true
output_linegraphs_collected = true
timestep_min = 10ps
timestep_max = 20ps

mobility_model = "canali"

output_plots = true

[SimpleTransfer]
output_plots = true

[DefaultDigitizer]
output_plots = true

[DetectorHistogrammer]
```

Plots again ...


Electric field magnitude at x=0.000000 mm
z (mm)
y (mm)
field strength (V/cm)
Electric field (z-component)
field strength (V/cm)
z (mm)

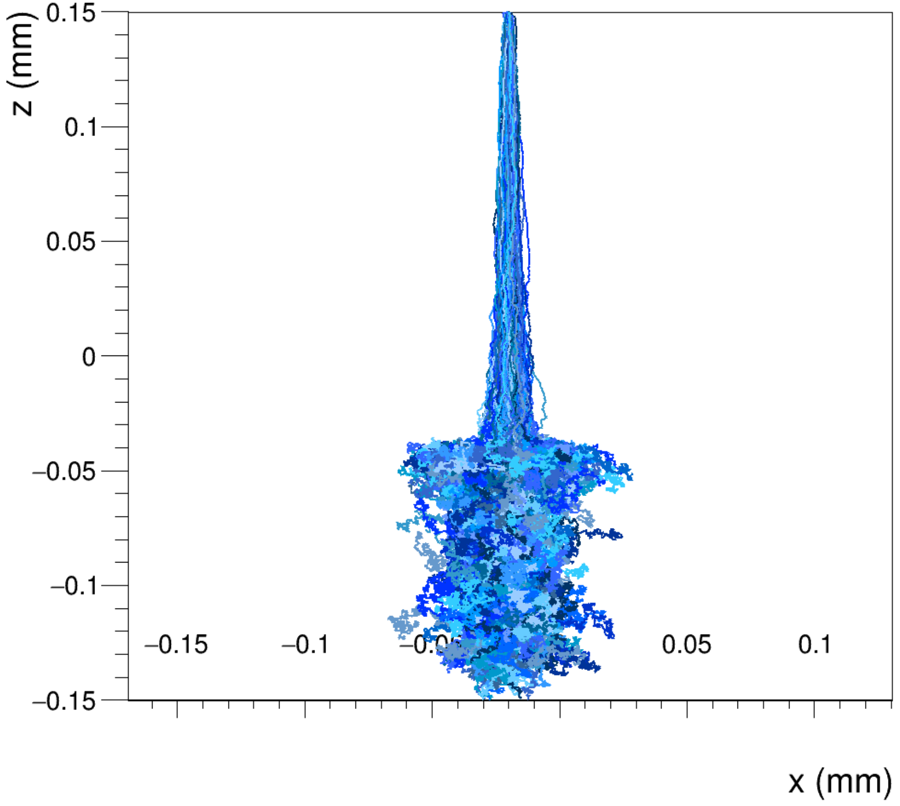

z (mm)
x (mm)

Plots again …

- Wrong sensor type –
  should be collecting holes

- Let's select to propagate holes instead

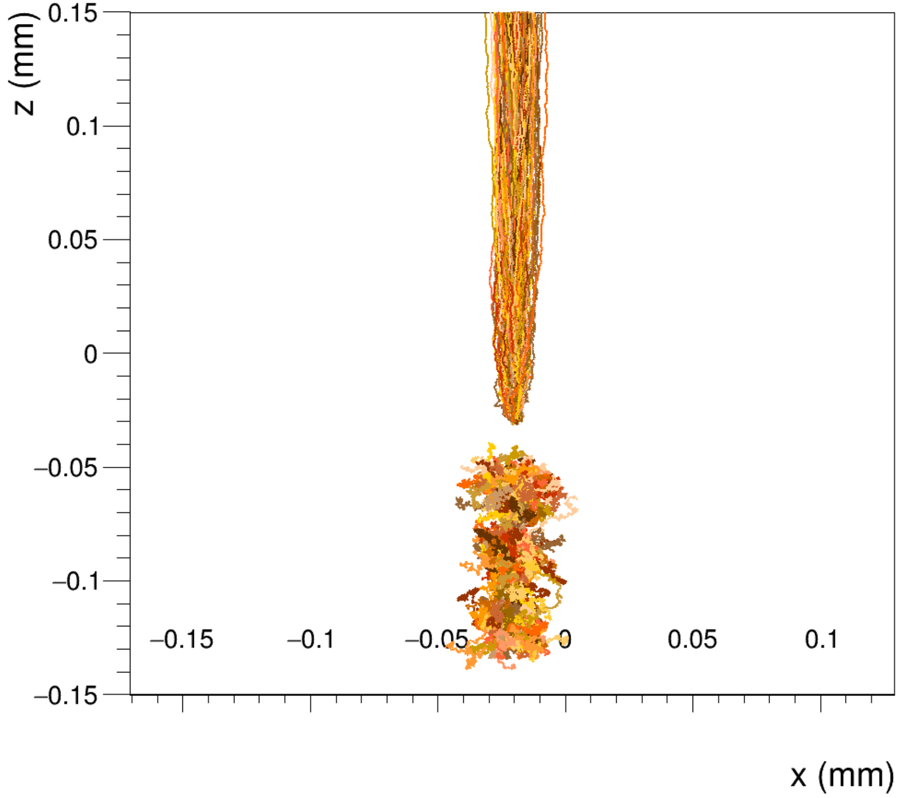
```
[GenericPropagation]
temperature = 293K
charge_per_step = 20
output_linegraphs = true
output_linegraphs_collected = true
timestep_min = 10ps
timestep_max = 20ps

propagate_holes = true
propagate_electrons = false

mobility_model = "canali"

output_plots = true
```



Plots again ...

- Wrong sensor type –
  should be collecting holes
- Let's select to propagate holes instead

```
[GenericPropagation]
temperature = 293K
charge_per_step = 20
output_linegraphs = true
output_linegraphs_collected = true
timestep_min = 10ps
timestep_max = 20ps

propagate_holes = true
propagate_electrons = false

mobility_model = "canali"

output_plots = true
```



Bonus Material

Allpix Squared
–
Development

Making your own Module I

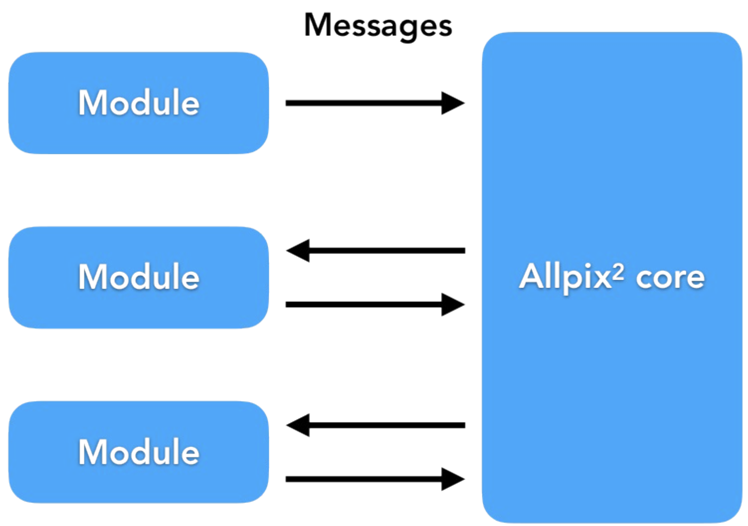
- Up to now:
  Setting up a simulation and configuring different modules for different detectors
  - No need to touch c++ code yet
- Next step:
  Developing a custom module
  - Keep in mind that modules may already be implemented / can be configured in a way that you need
  - Keep in mind that making your new module generic will benefit other users
- Useful script delivered in repository: **make_module.sh**

Making your own Module II

```
~/software/allpix-squared $ ./etc/scripts/make_module.sh

Preparing code basis for a new module:

Name of the module? NewTransfer
Type of the module?

1) unique
2) detector
#? 2

Input message type? PropagatedCharge
Creating directory and files...

Name:   NewTransfer
Author: Paul Schuetze (paul.schuetze@desy.de)
Path:   /home/paul/software/allpix-squared/src/modules/NewTransfer
This module listens to "PropagatedCharge" messages from one detector

Re-run CMake in order to build your new module.
```

A Word on Messages

- Modules exist entirely standalone in Allpix Squared
  - Information exchange by dispatching and receiving messages via the core of the software
  - Checks which messages each module is waiting for and whether messages being dispatched are subsequently used
- For per-detector modules, separate messages are dispatched for each detector, with the detector name used in the identification
- New modules need to decide what objects to request
  - DepositedCharges, PropagatedCharges, etc.



Making your own Module III

Constructor:
Configuration &
Bind to messages

Initialisation:
Variables / Histograms

Run Loop:
Main code,
executed for each event

```cpp
/**
 * @file
 * @brief Implementation of [NewTransfer] module
 * @copyright Copyright (c) 2017-2020 CERN and the Allpix Squared authors.
 * This software is distributed under the terms of the MIT License, copied verbatim in the file "LICENSE.md".
 * In applying this license, CERN does not waive the privileges and immunities granted to it by virtue of its status as an
 * Intergovernmental Organization or submit itself to any jurisdiction.
 */

#include "NewTransferModule.hpp"

#include <string>
#include <utility>

#include "core/utils/log.h"

using namespace allpix;

NewTransferModule::NewTransferModule(Configuration& config, Messenger* messenger, std::shared_ptr<Detector> detector)
    : Module(config, detector), detector_(detector), messenger_(messenger) {

    // ... Implement ... (Typically bounds the required messages and optionally sets configuration defaults)
    // Input required by this module
    messenger_->bindSingle(this, &NewTransferModule::message_, MsgFlags::REQUIRED);
}

void NewTransferModule::init() {
        // Get the detector name
        std::string detectorName = detector_->getName();
        LOG(DEBUG) << "Detector with name " << detectorName;
}

void NewTransferModule::run(unsigned int) {
    // ... Implement ... (Typically uses the configuration to execute function and outputs an message)
        std::string detectorName = message_->getDetector()->getName();
        LOG(DEBUG) << "Picked up " << message_->getData().size() << " objects from detector " << detectorName;
}
```

-:--- NewTransferModule.cpp   All (1,0)   (C++//l +1 hs Smrt Abbrev)

Making your own Module IV

- CMake is set up to compile all modules in the corresponding directory
  - Simply re-run CMake from the build directory and compile

```
$ cd build

$ cmake ..

$ make -j8 install

$ cd ../conf/

$ ../bin/allpix -c tutorial-simulation.conf
```

```
-- Building module ON    - NewTransfer
```

```
Scanning dependencies of target AllpixModuleNewTransfer
[ 75%] Building CXX object src/modules/NewTransfer/CMakeFiles/
_/__/core/module/dynamic_module_impl.cpp.o
[ 75%] Building CXX object src/modules/MagneticFieldReader/CMa
ieldReader.dir/MagneticFieldReaderModule.cpp.o
[ 75%] Building CXX object src/modules/NewTransfer/CMakeFiles/
ewTransferModule.cpp.o
```

- Module can be added to the simulation configuration in the same way as any other module

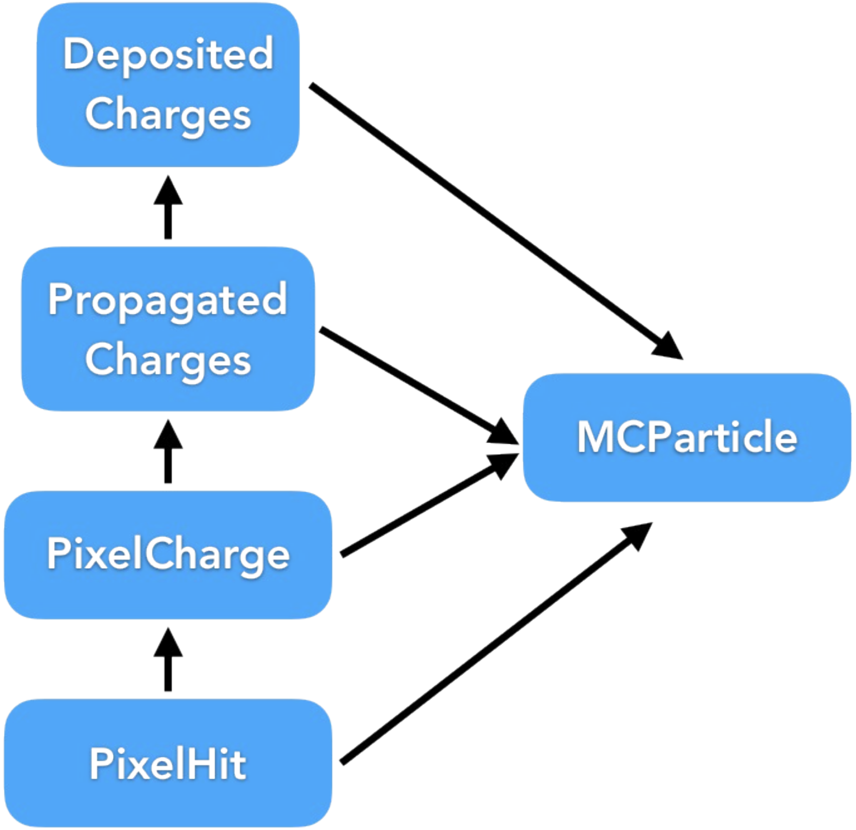
```
31 [NewTransfer]
```

A few other Features – MC History

- All objects contain information about where they come from
  - Direct link to preceding object
  - All objects link back to original *MCParticle*

- Messages templated in the code, so adding a new object is straight forward
  - Define the object, must inherit from *Object*
  - Add a definition for the message …

```
/**
 * @brief Typedef for message carrying propagated charges
 */
using PropagatedChargeMessage = Message<PropagatedCharge>;
```



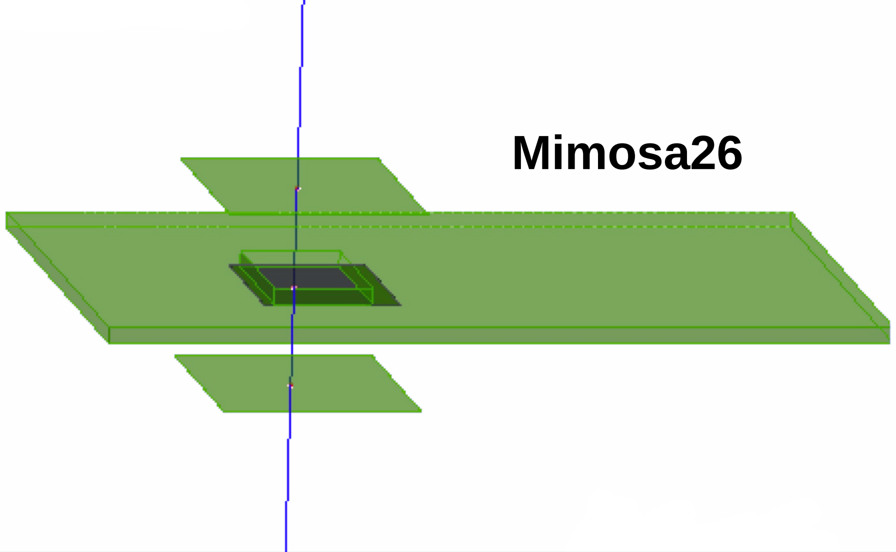
A few other Features – Output Writing

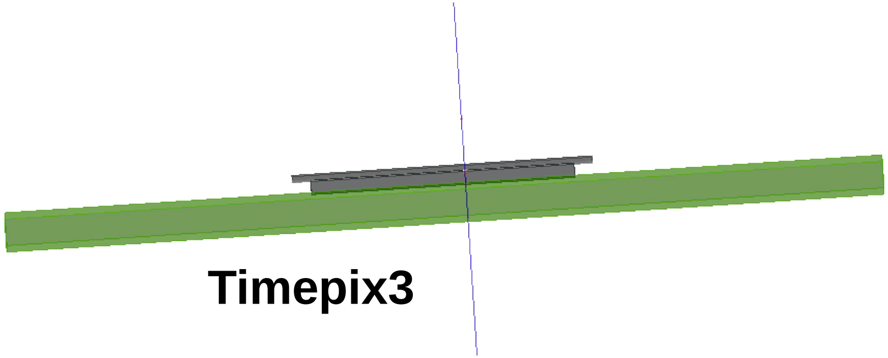
- Several output formats are already supported
  - LCIO – Linear collider community / EUTelescope
  - RCE – ATLAS pixel group data format
  - Corryvreckan – Test Beam Reconstruction framework
  - Text files – Human-readable
  - ROOTObjects – Allpix Squared data

- Allows to …
  - perform detailed analyses of individual sensors
  - replicate a test beam experiment and analyse the simulated data with the same software as the measured data

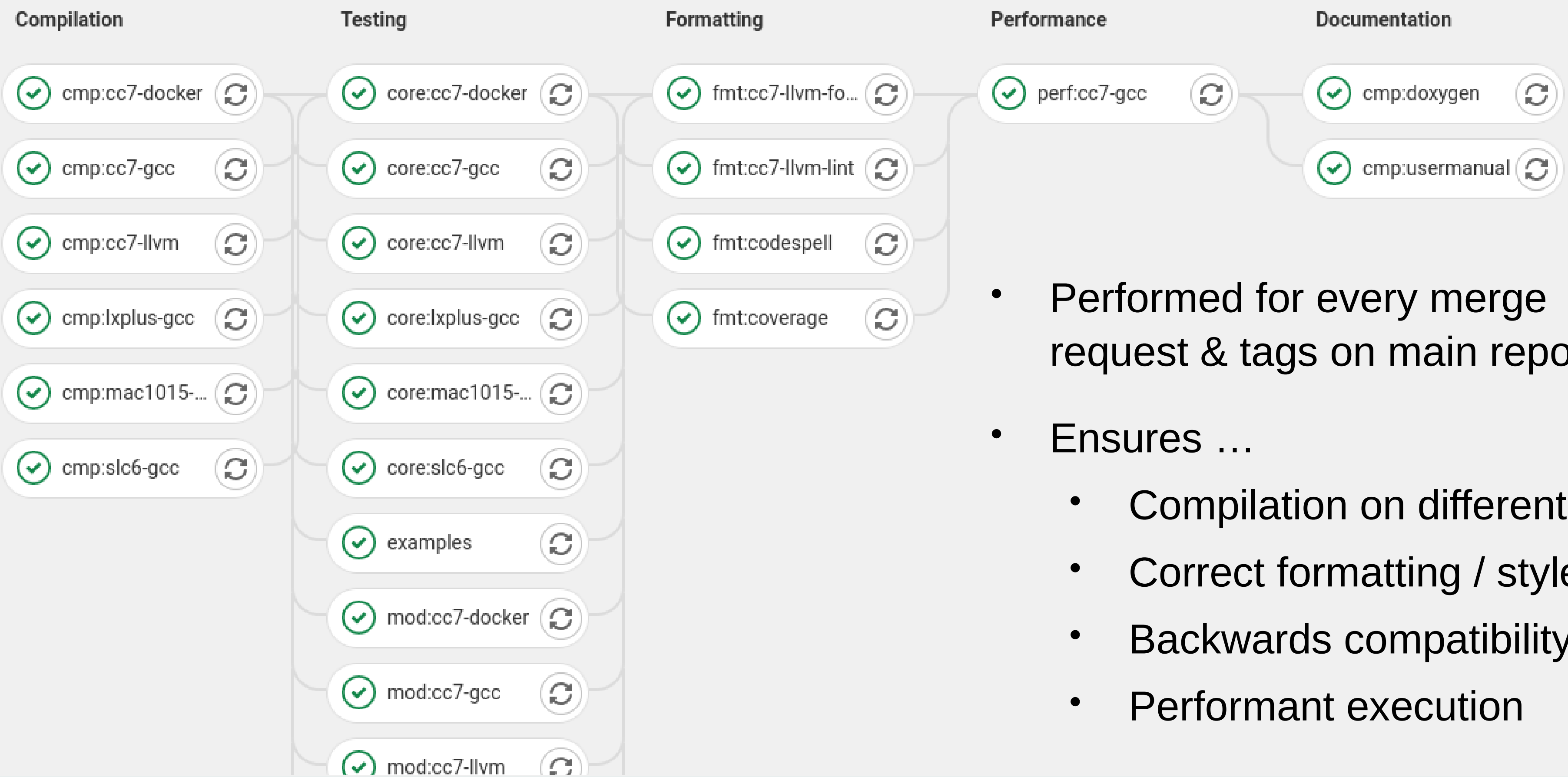
A few other Features – Geometry

- Currently geometries are implemented for **hybrid** and **monolithic** detectors
  - Monolithic can be used for strip detectors, with 1 by n “pixels” of appropriate size
- Geometry can be configured with cut-outs in the PCB, support materials (beam windows/physical supports), bump dimensions, etc.





Excursion: Continuous Integration

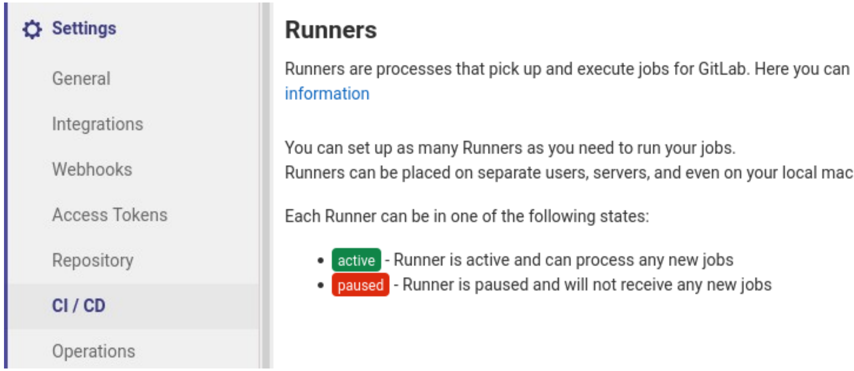


- Performed for every merge request & tags on main repository
- Ensures …
  - Compilation on different OS
  - Correct formatting / style
  - Backwards compatibility
  - Performant execution

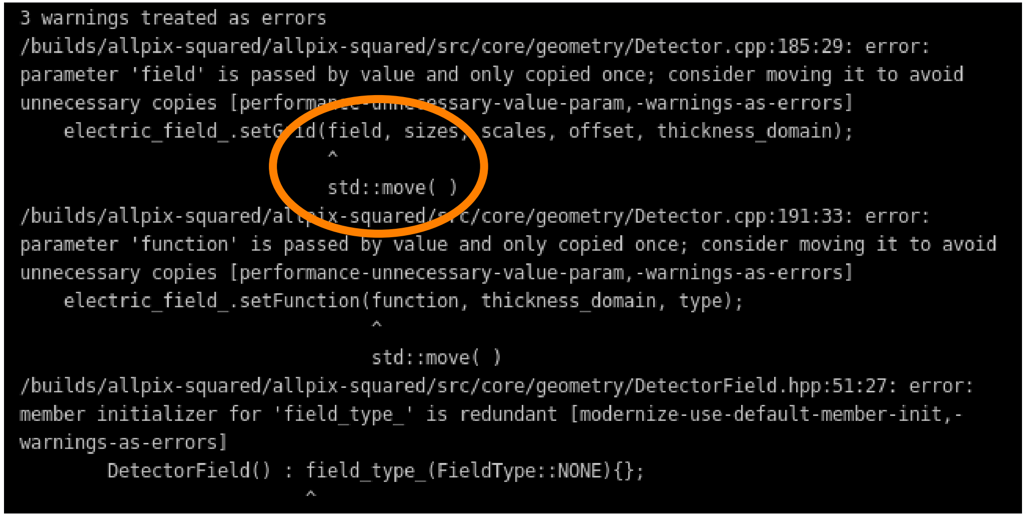
Excursion: Continuous Integration

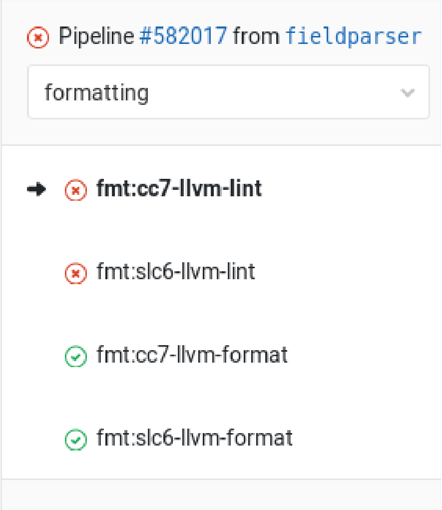
What? But how?!?

- Enable runners
  (machines to execute jobs)



- If Pipeline fails: read the output of the failing job





Where to go from here?

- Allpix Squared has many more features that we could not go through today
  - Transient propagation
    - Calculate pulse on readout electrode using electric and weighting field maps
    - Simulate response of charge sensitive amplifier
    - ➔ What’s new on Allpix Squared?, S. Spannagel, Wed., 17:50
  - Reading in of TCAD simulated electric and weighting fields
  - Magnetic field → primary particle and charge carrier propagation
  - Passive materials → replicate test beam setups or scattering measurements
  - Point charge / MIP deposition w/o Geant4
  - Source simulation
- Many of those are represented in one of the examples: `$ cd examples/`

Resources

Website
https://cern.ch/allpix-squared

Repository
https://gitlab.cern.ch/allpix-squared/allpix-squared

Docker Images
https://gitlab.cern.ch/allpix-squared/allpix-squared/container_registry

User Forum:
https://cern.ch/allpix-squared-forum/

Mailing Lists:
allpix-squared-users https://e-groups.cern.ch/e-groups/Egroup.do?egroupId=10262858
allpix-squared-developers https://e-groups.cern.ch/e-groups/Egroup.do?egroupId=10273730

User Manual:
https://cern.ch/allpix-squared/usermanual/allpix-manual.pdf