Hands-On:
The Allpix Squared Silicon Detector Simulation Framework

Simon Spannagel, Paul Schütze,
Sara Ruiz Daza, Stephan Lachnit,
Håkan Wennlöf, Manuel A. Del Rio Viera

11th Beam Telescope and Test Beams Workshop
17th - 21th April 2023

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)
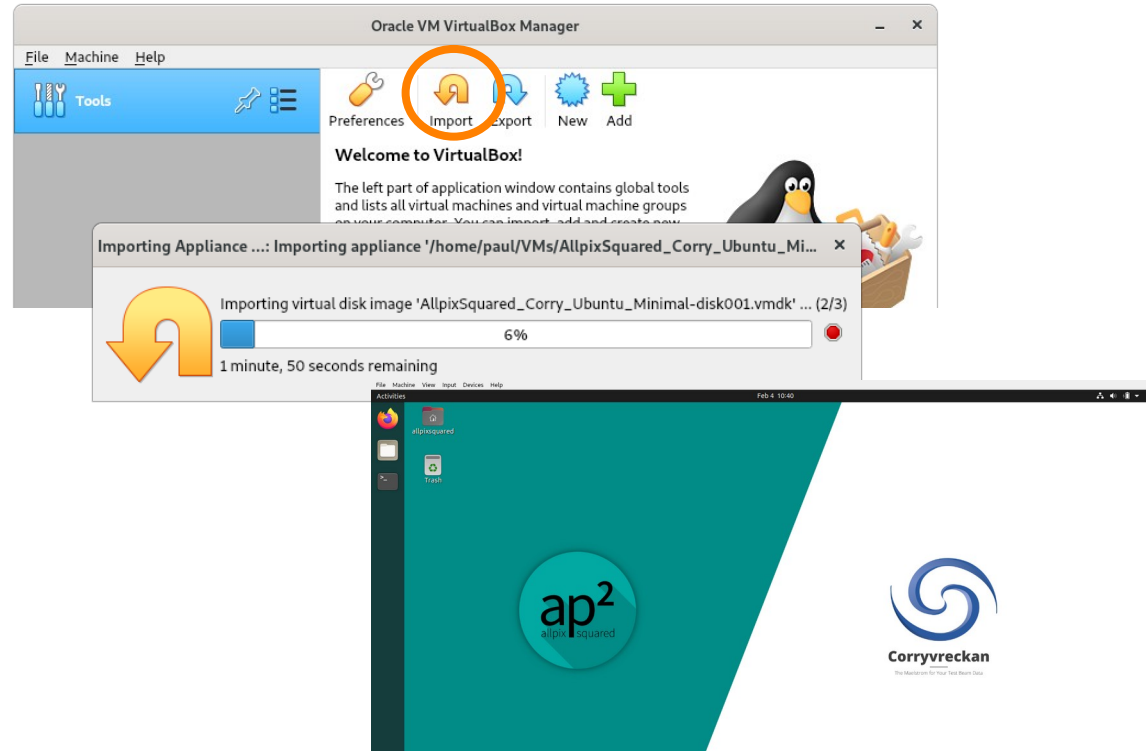
CVMFS – CernVM File System

- Central installation of software for CentOS7/8/9

- 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

- 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)
  - /2.4.1/x86_64-centos8-gcc11-opt/
  - /2.4.1/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/2.4.1/x86_64-centos7-gcc11-opt/setup.sh
$ allpix --version
Allpix Squared version v2.4.1
        ...
```

Simulation Modules

```
CapacitiveTransfer/    DepositionGenerator/     ElectricFieldReader/    ProjectionPropagation/  TransientPropagation/
CorryvreckanWriter/    DepositionLaser/         GDMLOutputWriter/       PulseTransfer/          VisualizationGeant4/
CSADigitizer/          DepositionPointCharge/   GenericPropagation/     RCEWriter/              WeightingPotentialReader/
DatabaseWriter/        DepositionReader/        GeometryBuilderGeant4/  ROOTObjectReader/
DefaultDigitizer/      DetectorHistogrammer/    InducedTransfer/        ROOTObjectWriter/
DepositionCosmics/     DopingProfileReader/     LCIOWriter/             SimpleTransfer/
DepositionGeant4/      Dummy/                   MagneticFieldReader/    TextWriter/
```

- 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

Other Methods

- Building from source, Docker, etc.
  See documentation: https://allpix-squared.docs.cern.ch/docs/02_installation/

  Documentation / Installation

  ## Installation

  Instructions on how to build and install Allpix Squared.

  This chapter aims to provide details and instructions on how to build and install Allpix Squared. An overview of possible build configurations is given. After installing and loading the required dependencies, there are various options to customize the installation of Allpix Squared. This chapter contains details on the standard installation process and information about custom build configurations.

  Alternatively, Allpix Squared can installed without building via a Docker image (see Section 2.7) or via CVMFS (see Section 2.8).

Getting Started

Your first Simulation – The Configuration

- Let's look at our first configuration file: **first-simulation.conf**

    - Hint: It is usually a good idea to create a new directory for a new simulation

- 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

```
1    [Allpix]
2    number_of_events = 1000
3    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

```
1   [detector1]
2   type = "timepix"
3   position = 0mm 0mm 0mm
4   orientation = 0deg 0deg 0deg
```

```
alpide.conf
clicpix2.conf
clicpix.conf
cmsp1.conf
diode.conf
fei3.conf
ibl_planar.conf
medipix3.conf
mimosa23.conf
mimosa26.conf
rd53a_25.conf
rd53a_50.conf
test.conf
timepix4.conf
timepix.conf
velopix.conf
:/opt/allpix-squared/share/Allpix/models
```

The Model: timepix.conf

```
 1  type = "hybrid"
 2
 3  number_of_pixels = 256 256
 4  pixel_size = 55um 55um
 5
 6  sensor_thickness = 300um
 7  sensor_excess = 1mm
 8
 9  bump_sphere_radius = 9.0um
10  bump_cylinder_radius = 7.0um
11  bump_height = 20.0um
12
13  chip_thickness = 700um
14  chip_excess_left = 15um
15  chip_excess_right = 15um
16  chip_excess_bottom = 2040um
17
18  [support]
19  thickness = 1.76mm
20  size = 47mm 79mm
21  offset = 0 -22.25mm
```
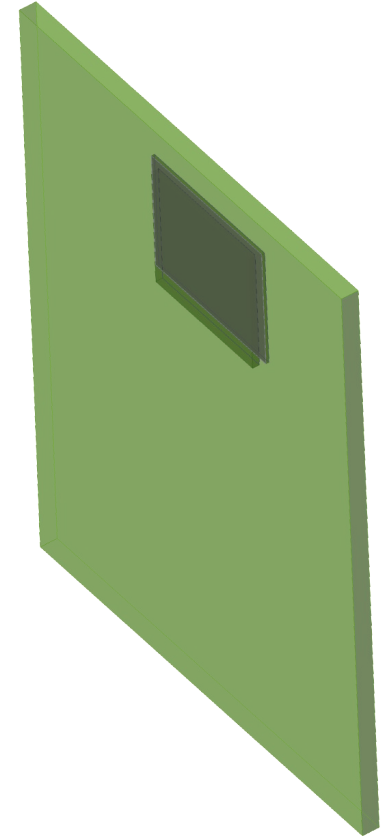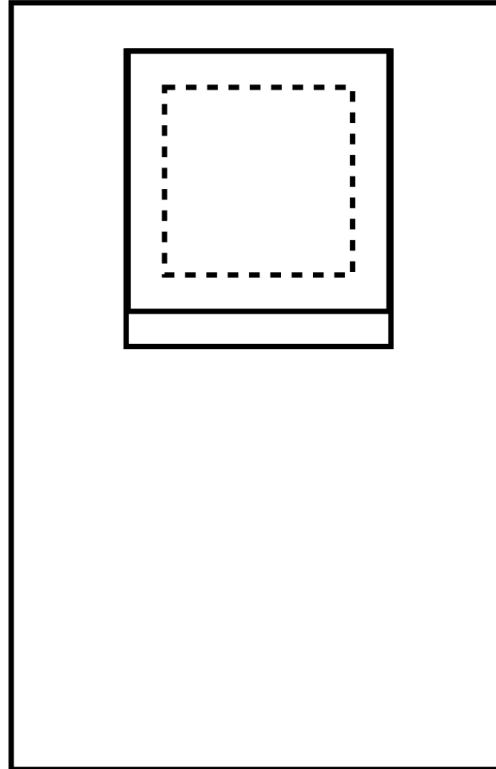
Adding Physics

- 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 …
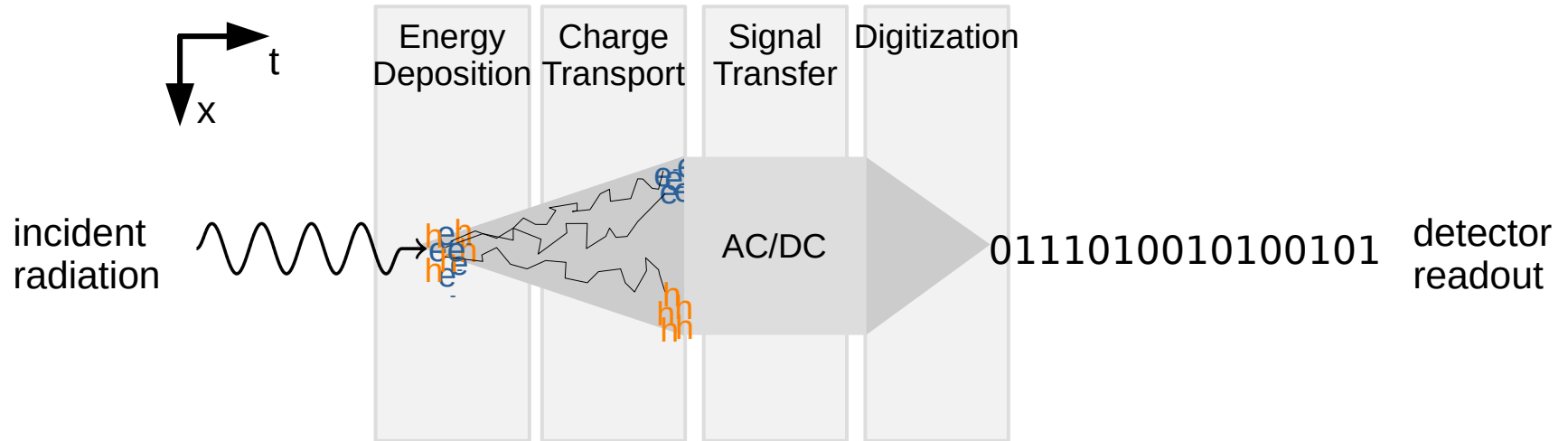
Adding Physics

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

Geometry Construction	Electric Field Config.	Energy Deposition	Charge Transport	Signal Transfer	Digitization	Monitoring	Writing Output Data
All detectors Construction of the Geant4 geometry	**Detector 1** Linear e-field	**All detectors** Charge deposition with Geant4	**Detector 1** Project charges	**Detector 1** Transfer charges	**Detector 1** Digitisation	**Detector 1** Monitoring histograms	**All detectors** Write simulation results to file

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]

[GenericPropagation]

[SimpleTransfer]

[DefaultDigitzer]

[DetectorHistogrammer]
```

Module Parameters

- All modules are thoroughly described in the user manual
  https://allpix-squared.docs.cern.ch/docs/08_modules/

  - List of available parameters for corresponding module with default values

  - Usage examples

SimpleTransfer

Transfers propagated charges to the pixel front-end

Description

Combines individual sets of propagated charges together to a set of charges on the sensor pixels and thus prepares them for processing by the detector front-end electronics. The module does a simple direct mapping to the nearest pixel, ignoring propagated charges that are too far away from the implants or outside the pixel grid. Timing information for the pixel charges is currently not yet produced, but can be fetched from the linked propagated charges.

If one or more implants are defined for the respective detector model, the `collect_from_implant` option can be turned on in order to only pick charge carriers from the implant volume and ignore everything outside this region. Only charge carriers from front-side implants are collected, charge carriers on back-side implants are dropped. Since this will lead to unexpected and undesired behavior when using linear electric fields, this option can only be used when using fields with an x/y dependence (i.e. field maps imported from TCAD). In case no implants are defined, charge carriers are collected from the pixel surface and the parameter `max_depth_distance` can be used to control the depth from which charge carriers are taken into account.

A histogram of charge carrier arrival times is generated if `output_plots` is enabled. The range and granularity of this plot can be configured.

Parameters

- `max_depth_distance` : Maximum distance in depth, i.e. normal to the sensor surface at the implant side, for a propagated charge to be taken into account in case the detector has no implants defined. Defaults to `5um` .
- `collect_from_implant` : Only consider charge carriers within the implant region of the respective detector instead of the full surface of the sensor. Should only be used with non-linear electric fields and defaults to `false` .
- `output_plots` : Determines if output plots should be generated. Disabled by default.
- `output_plots_step` : Bin size of the arrival time histogram in units of time. Defaults to `0.1ns` .
- `output_plots_range` : Total range of the arrival time histogram. Defaults to `100ns` .

The World

- **GeometryBuilderGeant4**

    `[GeometryBuilderGeant4]`

- Translates and defines the geometry to Geant4

- No required parameters
    - By default the setup is placed in air

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
output_plots = true
```

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
output_plots = true
```
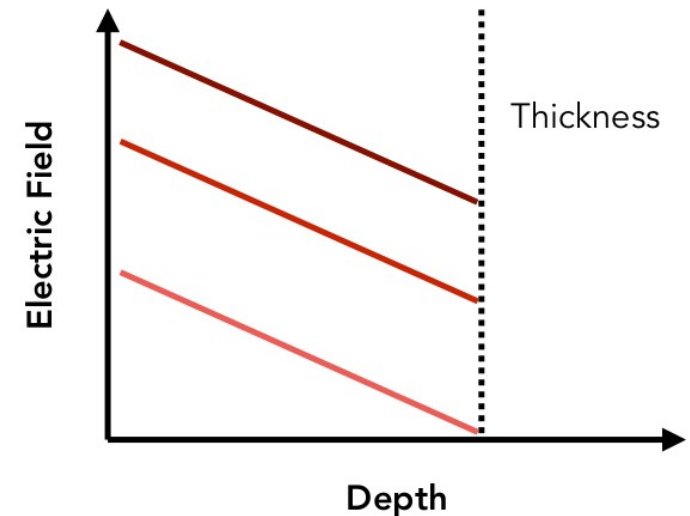
The Propagation

- **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
output_plots = true
```

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**

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

```
[SimpleTransfer]
output_plots = true
```

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
output_plots = true
```

The Histogrammer

- **DetectorHistogrammer**

  [DetectorHistogrammer]

- Creation of typical detector performance plots including a simple clustering technique and comparison the Monte Carlo truth, like …

  [DetectorHistogrammer]

  - 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*

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 …

```
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]
```
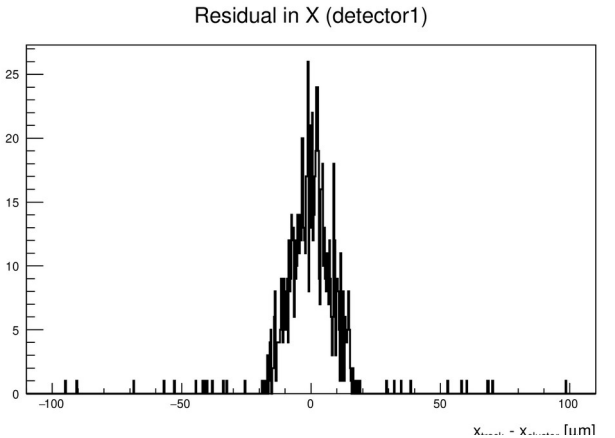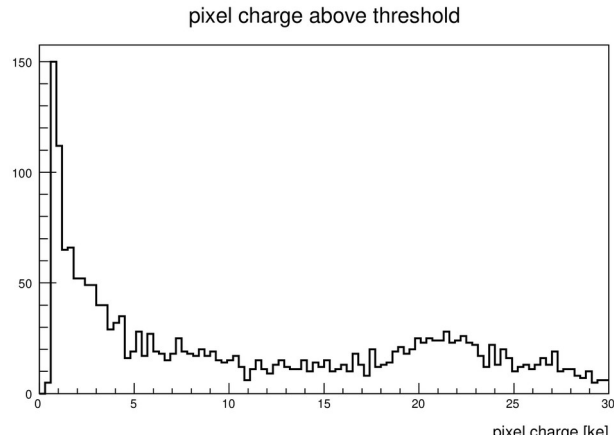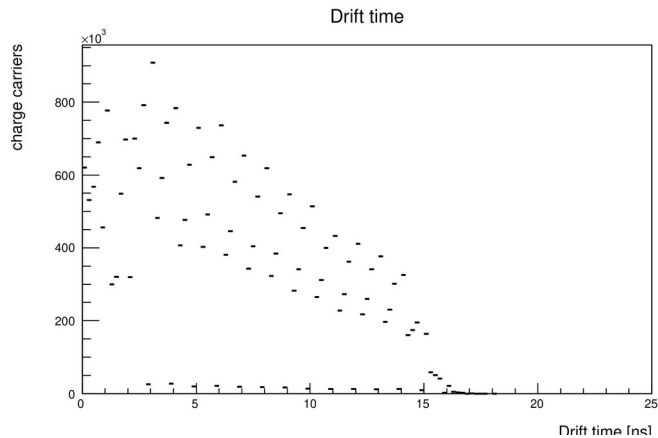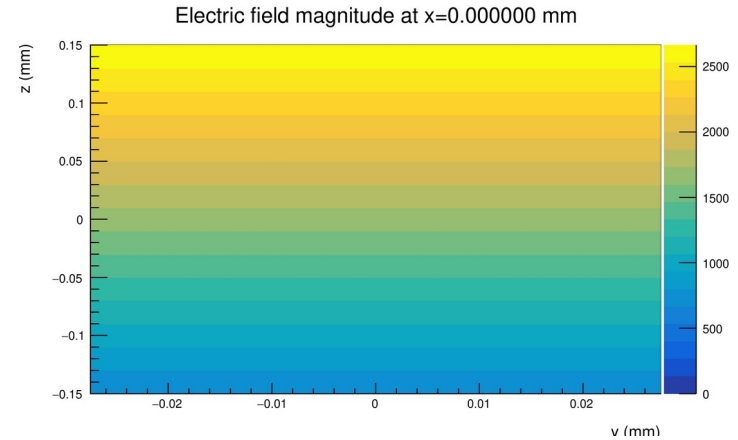
Plots, plots, plots ...

- The output is (if not configured otherwise) located in *output/modules.root*

- Let's have a look …

  $ rootbrowse output/modules.root

Electric field magnitude at x=0.000000 mm

Drift time

pixel charge above threshold

Residual in X (detector1)

Task: Optimize the Resolution

- Determine the resolution and **improve it** by …

  - Changing the incidence angle of particles (*orientation*)

  - Changing the magnetic field
    (https://allpix-squared.docs.cern.ch/docs/08_modules/magneticfieldreader/)

- Hint for speeding up the process:

```
$ 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

```
1  [detector1]
2  type = "timepix"
3  position = 0mm 0mm 0mm
4  orientation = 0 0 0
5
6  [detector2]
7  type = "timepix"
8  position = 0mm 0mm 20mm
9  orientation = 0 0 0
10
11 [detector3]
12 type = "timepix"
13 position = 0mm 0mm 40mm
14 orientation = 0 0 0
15
16 [detector4]
17 type = "timepix"
18 position = 0mm 0mm 60mm
19 orientation = 0 0 0
20
21 [detector5]
22 type = "timepix"
23 position = 0mm 0mm 80mm
24 orientation = 0 0 0
25
26 [detector6]
27 type = "timepix"
28 position = 0mm 0mm 100mm
29 orientation = 0 0 0
```

Visualising the Setup
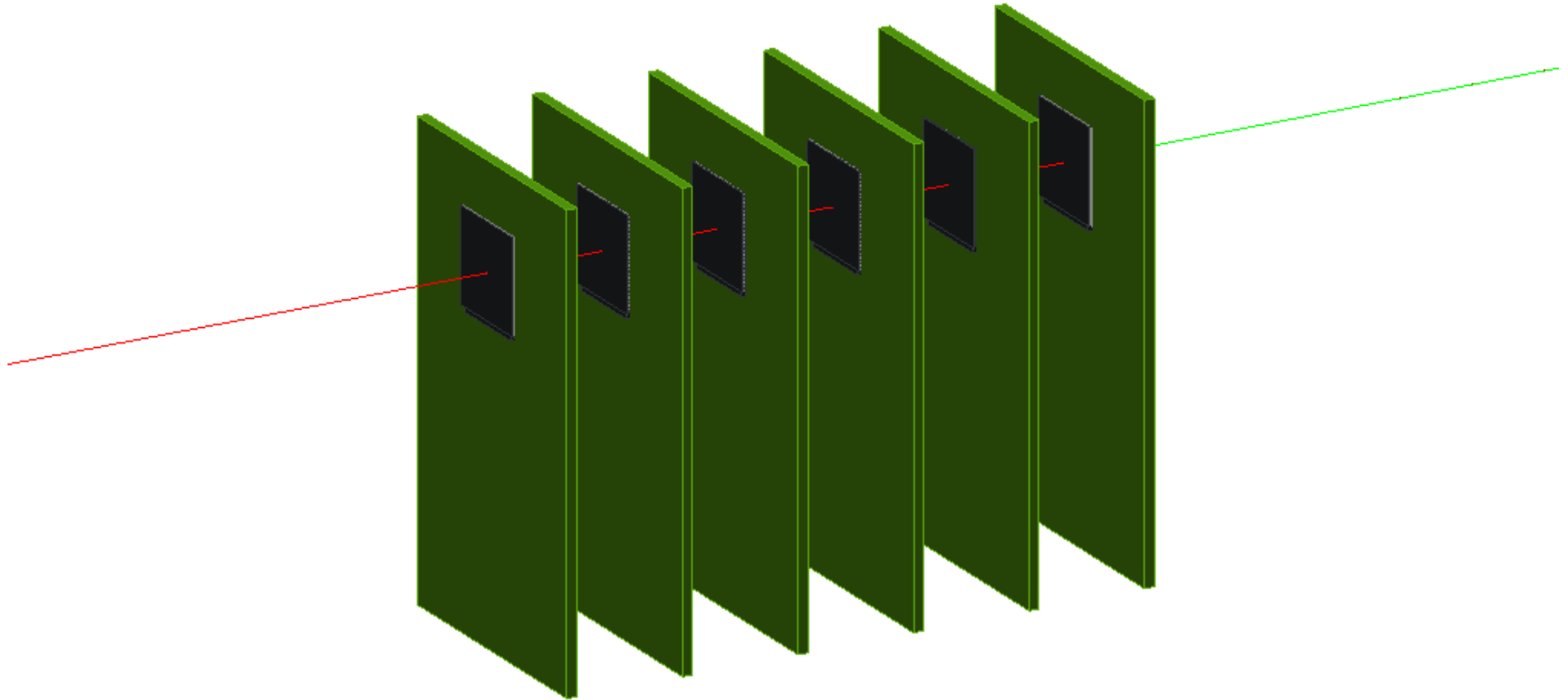
Treating Detectors in Different Ways

- 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
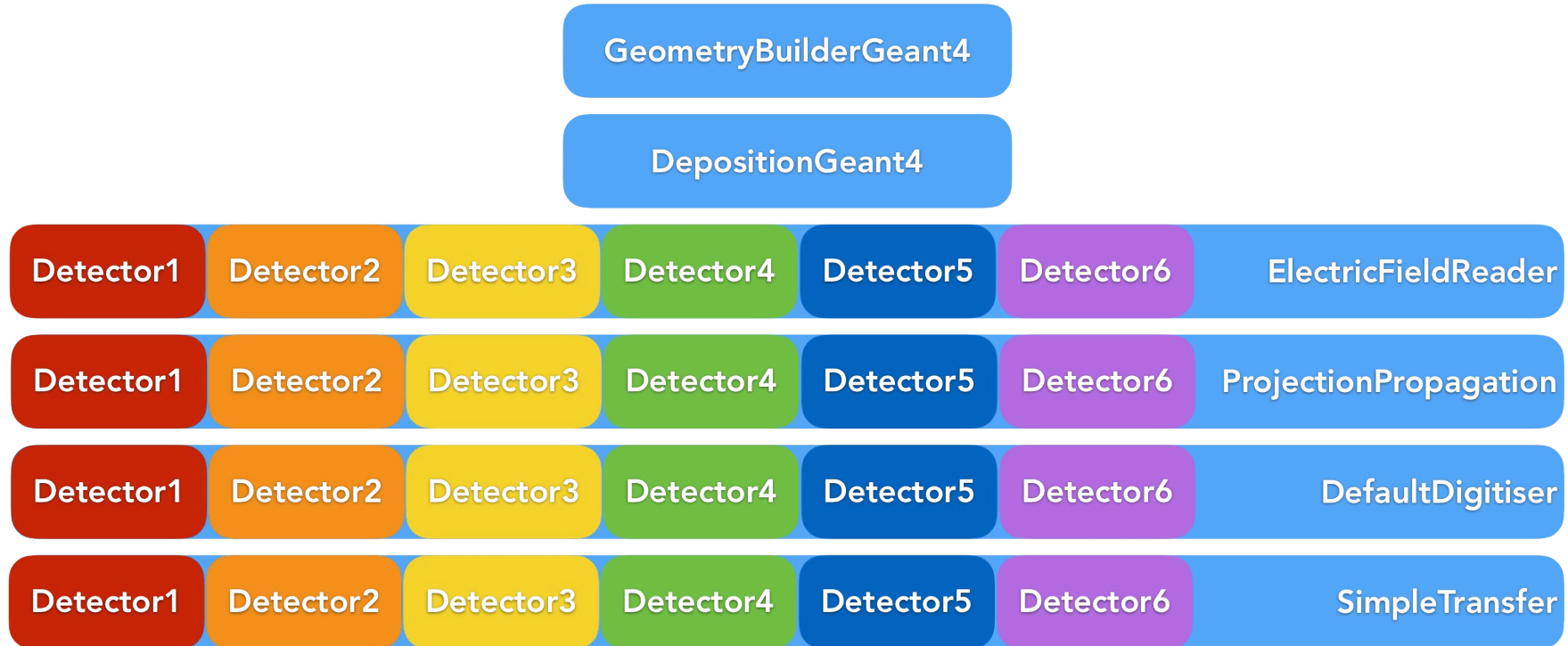

GeometryBuilderGeant4

DepositionGeant4

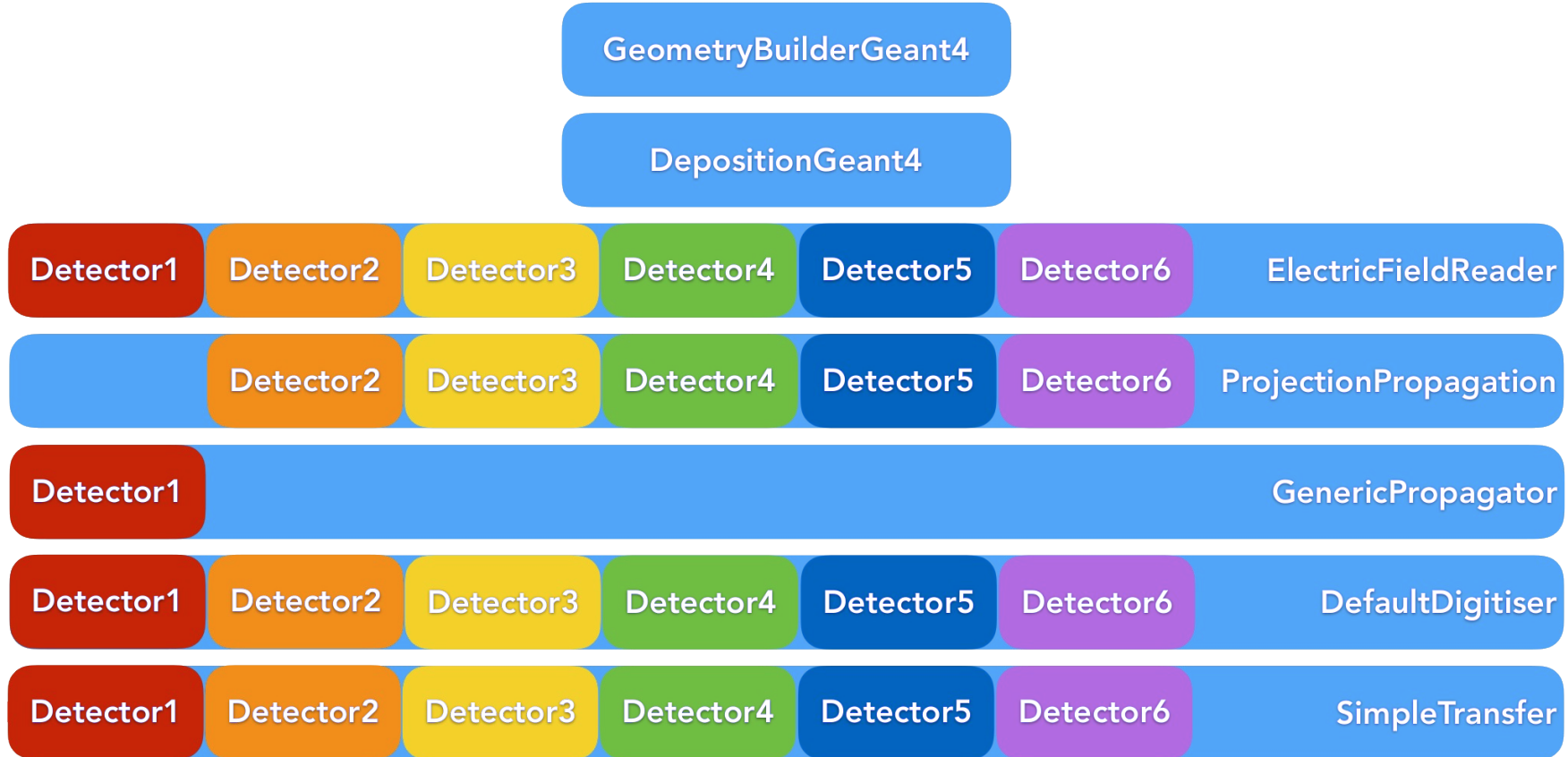ElectricFieldReader

ProjectionPropagation

DefaultDigitiser

SimpleTransfer

A Multi-Detector Simulation Chain

GeometryBuilderGeant4

DepositionGeant4

Detector1	Detector2	Detector3	Detector4	Detector5	Detector6	ElectricFieldReader
Detector1	Detector2	Detector3	Detector4	Detector5	Detector6	ProjectionPropagation
Detector1	Detector2	Detector3	Detector4	Detector5	Detector6	DefaultDigitiser
Detector1	Detector2	Detector3	Detector4	Detector5	Detector6	SimpleTransfer

Now what if I … ?

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 = "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 = -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]
qdc_slope = 200e
qdc_resolution = 8
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

```
1  [detector1]
2  type = "timepix"
3  position = 0mm 0mm 0mm
4  orientation = 0 0 0
```

```
1  [Allpix]
2  number_of_events = 1000
3  detectors_file = "tutorial-geometry.conf"
4  log_level = "WARNING"
5  random_seed_core = 1234
6
7  [GeometryBuilderGeant4]
8
9  [DepositionGeant4]
10 particle_type = "e-"
11 source_energy = 5GeV
12 source_type = "beam"
13 beam_size = 3mm
14 source_position = 0 0 -200mm
15 beam_direction = 0 0 1
16 physics_list = FTFP_BERT_EMZ
17 output_plots = true
18
19 [ElectricFieldReader]
20 model = "linear"
21 bias_voltage = -50V
22 depletion_voltage = -30V
23 output_plots = true
24
25 [GenericPropagation]
26 name = "detector1"
27 temperature = 293K
28 charge_per_step = 10
29 output_plots = true
30
31 [SimpleTransfer]
32 output_plots = true
33
34 [ROOTObjectWriter]
35 file_name = "prepared_data.root"
36 include = "MCParticle", "PixelCharge"
```

```
1  [Allpix]
2  number_of_events = 1000
3  detectors_file = "tutorial-geometry.conf"
4  log_level = "WARNING"
5  random_seed_core = 1234
6
7  [ROOTObjectReader]
8  file_name = "output/prepared_data.root"
9  include = "MCParticle", "PixelCharge"
10
11 [DefaultDigitizer]
12 threshold = 400e
13 output_plots = true
14
15 [DetectorHistogrammer]
```

```
11 [DefaultDigitizer]
12 threshold = 800e
13 output_plots = true
```

```
11 [DefaultDigitizer]
12 threshold = 1200e
13 output_plots = true
```
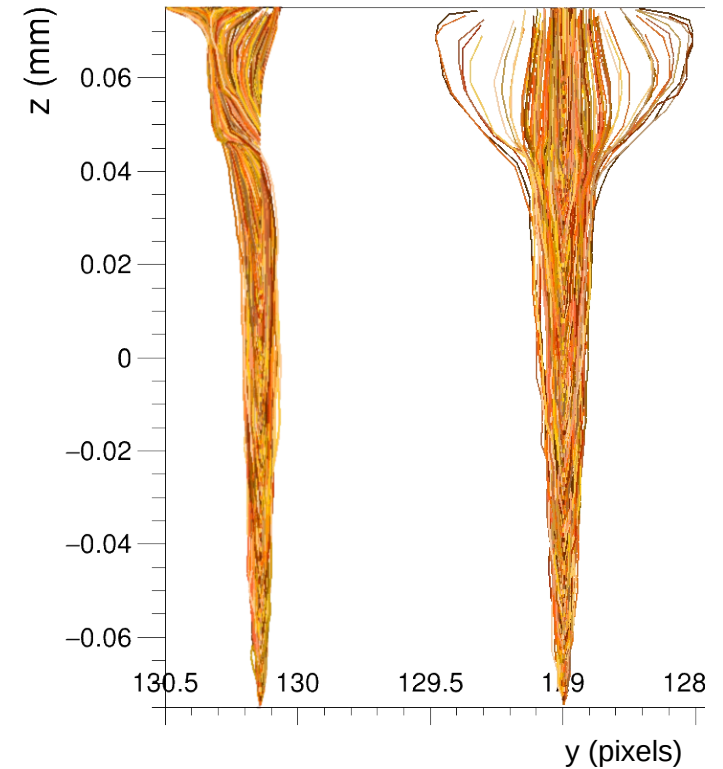
Or try …

```
$ 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

Documentation / Additional Tools & Resources / Mesh Converter

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 required 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 mesh is produced. This tool can work in two different modes, the closest-neighbor mode and interpolation mode:

Simple Closest-Neighbor Search

In this mode, selected by setting the parameter `interpolate = false`, no interpolation of field values is performed, but for every output mesh point, the values from the closest neighbor of the input mesh are taken. In most cases this approach should produces reasonably precise results with a granularity similar to the respective adaptive mesh granularity in the respective region. The tool uses the Octree `findNeighbor` algorithm [@octree] to find the closest neighbor to the query point.

Barycentric Interpolation Method

In this mode, the regular mesh is created by scanning the model volume in regular X, Y and Z steps 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 tool uses the Octree `radiusNeighbors` neighbor search algorithm [@octree].

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.

Mesh Converter

- Driven by 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`
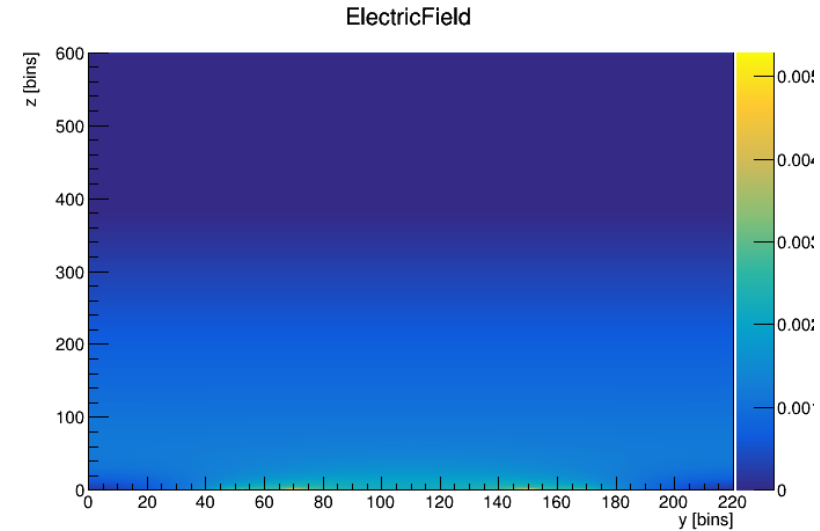
Converting an Electric Field

- 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
  ```
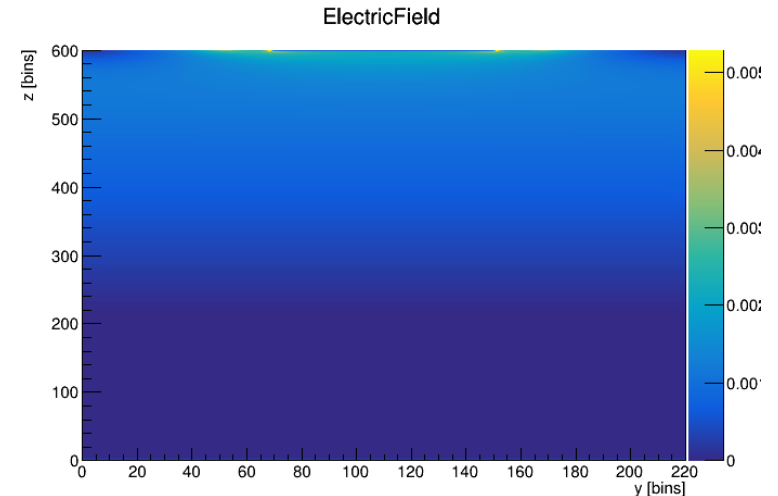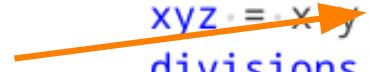
➔ Umm… Well…

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

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

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


ElectricField

Loading APF Files

- Adapt the *ElectricFieldReader*

```
[ElectricFieldReader]
model = "mesh"
file_name = "../example_field/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 or *field_depth* can be used to constrain the electric field to this part
  of the sensor

- From versions after 2.4.1, there is an additional required parameter:

  field_mapping=PIXEL_FULL , indicating that the map spans the full 2D plane and the
  field is centered around the pixel center

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.conf"
log_level = "STATUS"

[GeometryBuilderGeant4]

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

[ElectricFieldReader]
model = "mesh"
file_name = "../example_field/timepix_like_tcad_ElectricField.apf"
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]
```
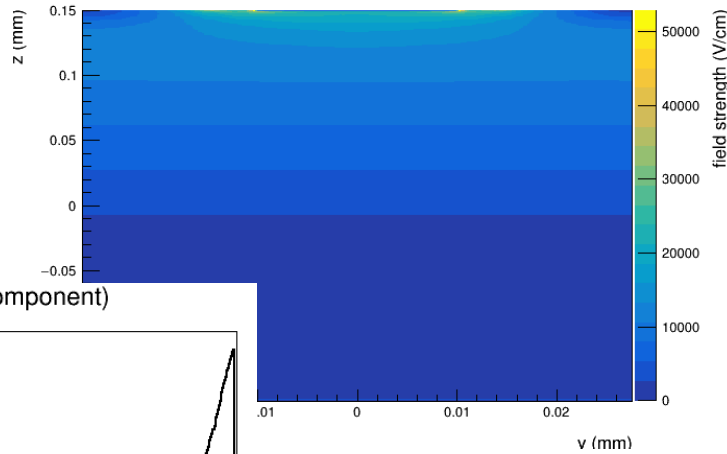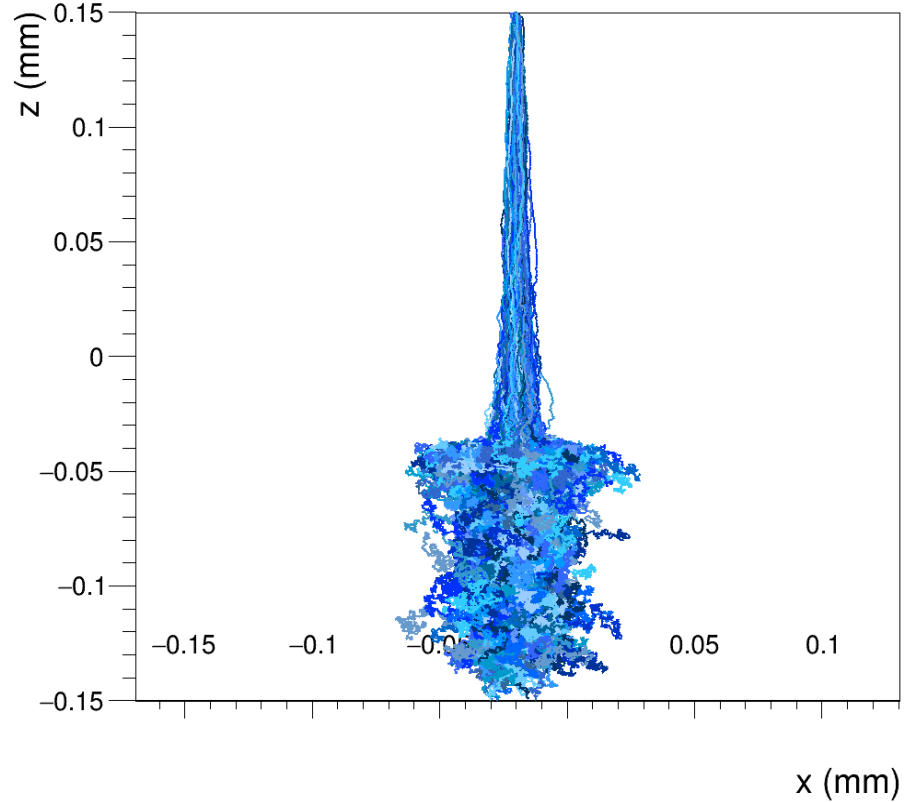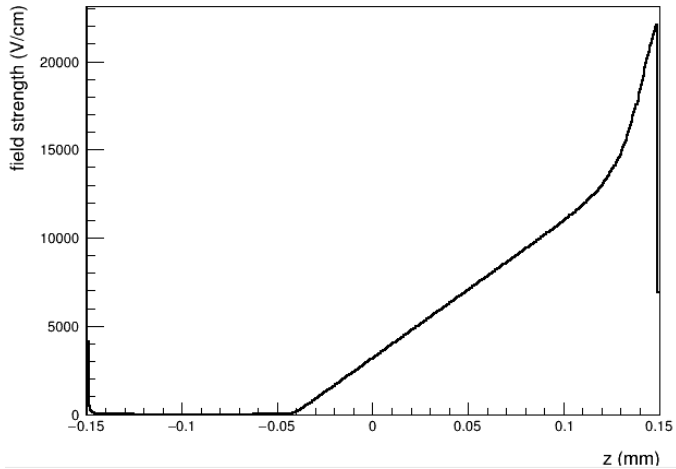
Plots again …

Electric field magnitude at x=0.000000 mm

Electric field (z-component)

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
```
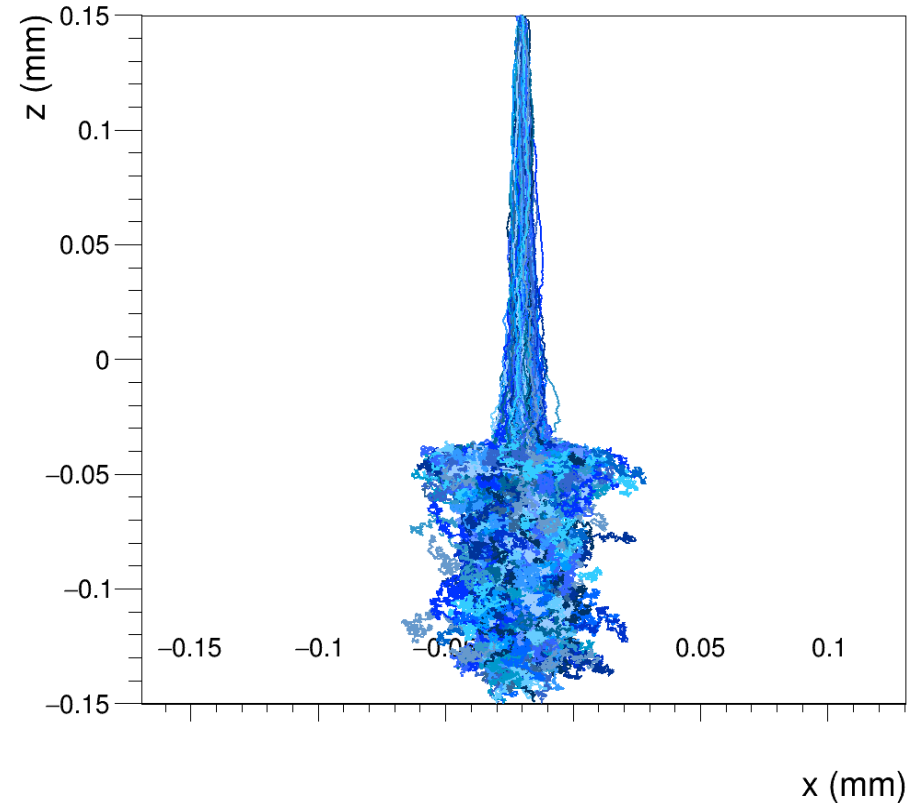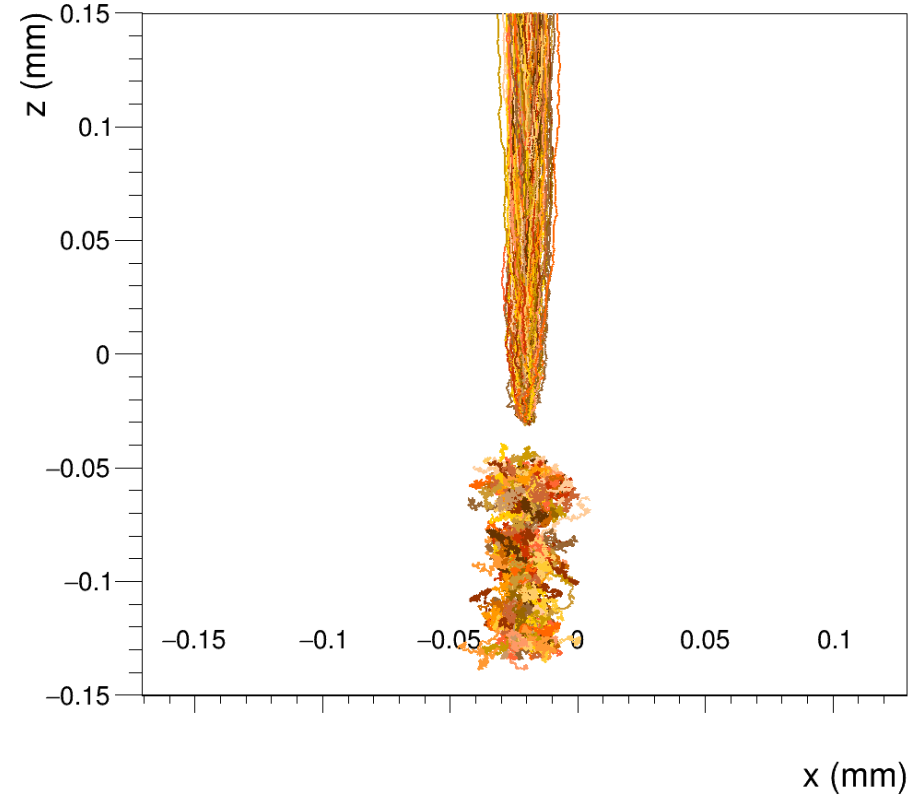
Plots again …

- 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.

Messages

Module →

Module ←→

Module ←→

Allpix² core
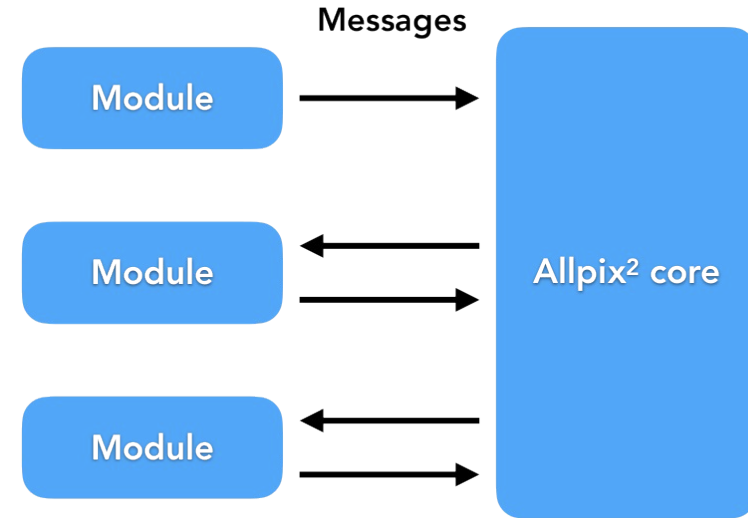
Making your own Module III

Constructor:
Configuration &
Bind to messages

Initialisation:
Variables / Histograms

Run Loop:
Main code,
executed for each event

```
 * In applying this license, CERN does not waive the privileges and immunities g
ranted 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"
```

```cpp
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::REQUIRE
D);
}
```

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

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

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
```

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

```
-- 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/CM
ieldReader.dir/MagneticFieldReaderModule.cpp.o
[ 75%] Building CXX object src/modules/NewTransfer/CMakeFiles
ewTransferModule.cpp.o
```

```
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>;
```
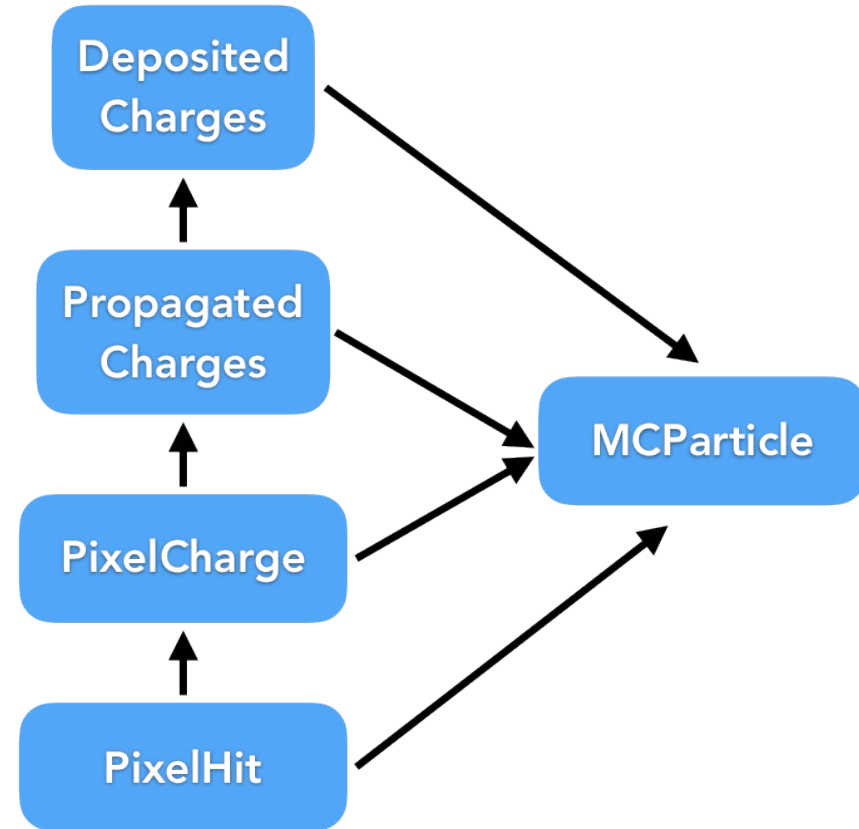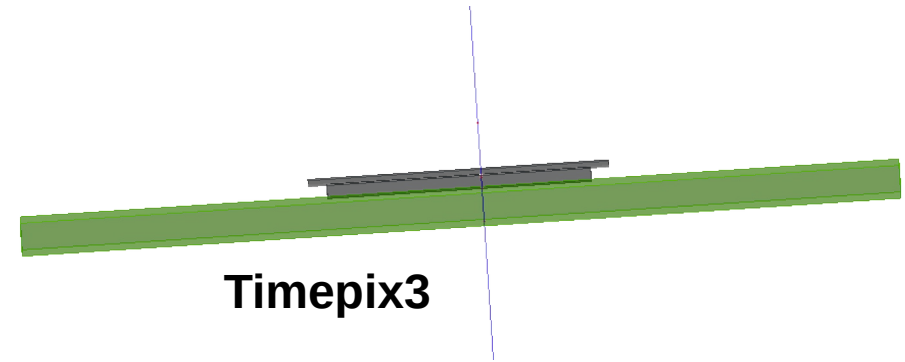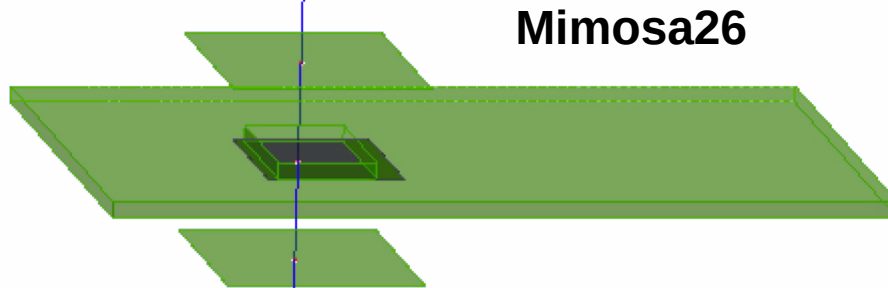
A few other Features – Output Writing

- 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

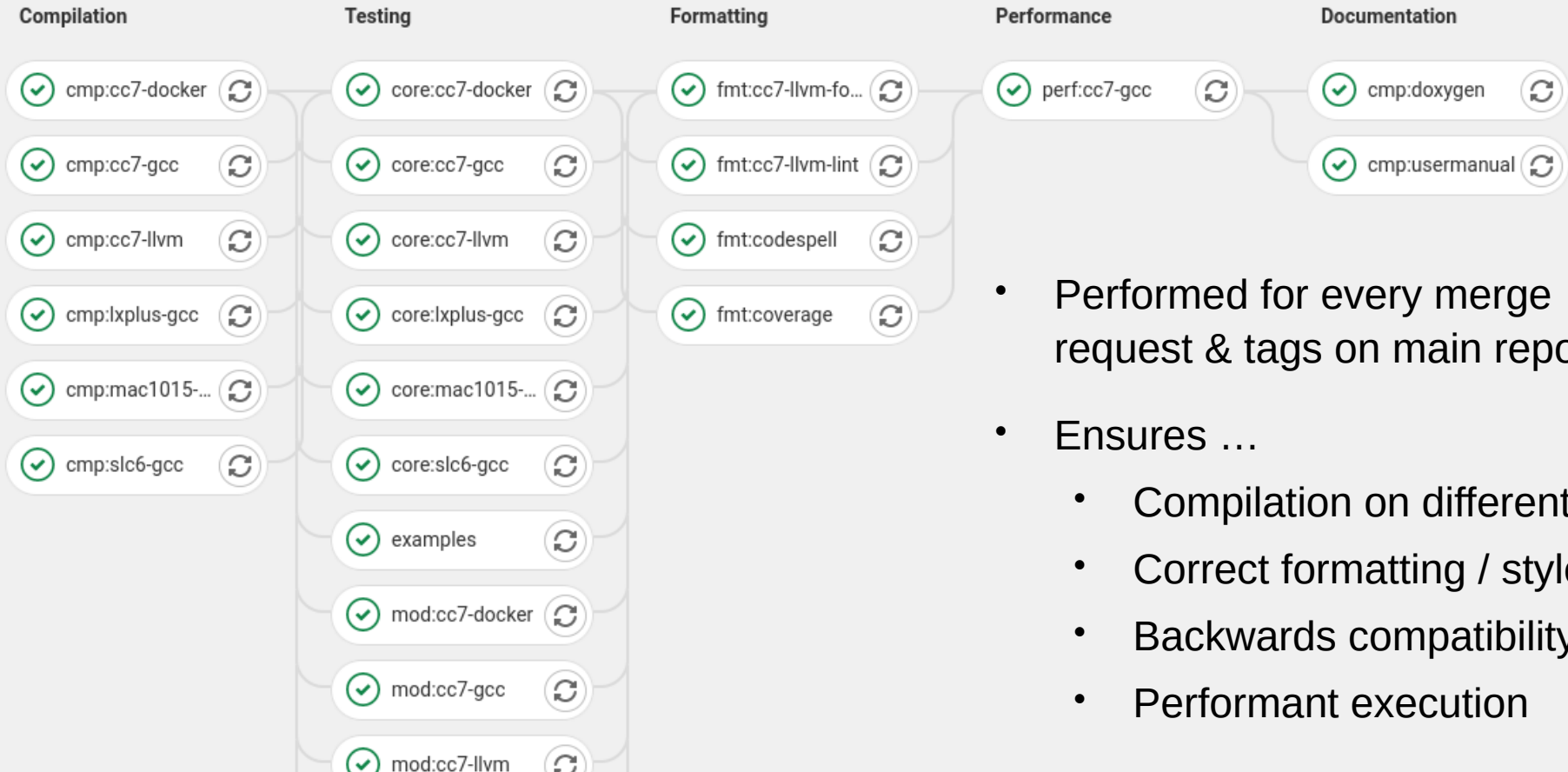 - ROOTObjects – Allpix Squared data

- 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.

Mimosa26

Timepix3

Excursion: Continuous Integration

Compilation

- ✓ cmp:cc7-docker ↻
- ✓ cmp:cc7-gcc ↻
- ✓ cmp:cc7-llvm ↻
- ✓ cmp:lxplus-gcc ↻
- ✓ cmp:mac1015-... ↻
- ✓ cmp:slc6-gcc ↻

Testing

- ✓ core:cc7-docker ↻
- ✓ core:cc7-gcc ↻
- ✓ core:cc7-llvm ↻
- ✓ core:lxplus-gcc ↻
- ✓ core:mac1015-... ↻
- ✓ core:slc6-gcc ↻
- ✓ examples ↻
- ✓ mod:cc7-docker ↻
- ✓ mod:cc7-gcc ↻
- ✓ mod:cc7-llvm ↻

Formatting

- ✓ fmt:cc7-llvm-fo... ↻
- ✓ fmt:cc7-llvm-lint ↻
- ✓ fmt:codespell ↻
- ✓ fmt:coverage ↻

Performance

- ✓ perf:cc7-gcc ↻

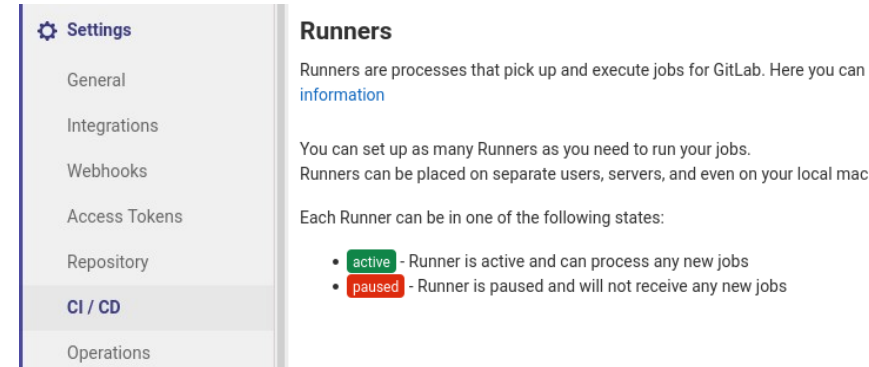**Documentation**

- ✓ cmp:doxygen ↻
- ✓ cmp:usermanual ↻

- 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

Runners

Runners are processes that pick up and execute jobs for GitLab. Here you can
information

You can set up as many Runners as you need to run your jobs.
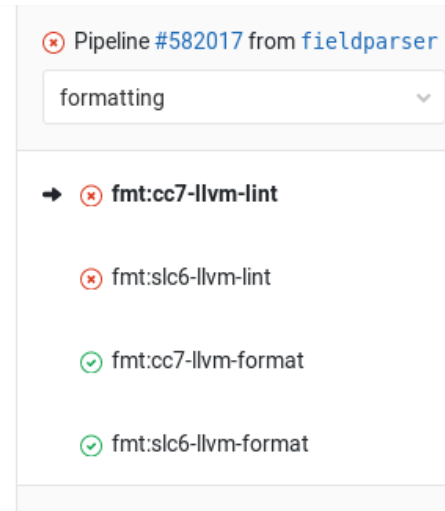Runners can be placed on separate users, servers, and even on your local mac

Each Runner can be in one of the following states:

- **active** - Runner is active and can process any new jobs
- **paused** - Runner is paused and will not receive any new jobs

```
3 warnings treated as errors
/builds/allpix-squared/allpix-squared/src/core/geometry/Detector.cpp:185:29: error:
parameter 'field' is passed by value and only copied once; consider moving it to avoid
unnecessary copies [performance-unnecessary-value-param,-warnings-as-errors]
    electric_field_.setGrid(field, sizes, scales, offset, thickness_domain);
                            ^
                            std::move( )
/builds/allpix-squared/allpix-squared/src/core/geometry/Detector.cpp:191:33: error:
parameter 'function' is passed by value and only copied once; consider moving it to avoid
unnecessary copies [performance-unnecessary-value-param,-warnings-as-errors]
    electric_field_.setFunction(function, thickness_domain, type);
                                ^
                                std::move( )
/builds/allpix-squared/allpix-squared/src/core/geometry/DetectorField.hpp:51:27: error:
member initializer for 'field_type_' is redundant [modernize-use-default-member-init,-
warnings-as-errors]
        DetectorField() : field_type_(FieldType::NONE){};
                          ^
```

Pipeline #582017 from `fieldparser`

formatting

→ ⊗ **fmt:cc7-llvm-lint**

⊗ fmt:slc6-llvm-lint

⊘ fmt:cc7-llvm-format

⊘ fmt:slc6-llvm-format

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