ap²
allpix squared

cern.ch/allpix-squared

Hands-On: The Allpix Squared Simulation Framework

Simon Spannagel, CERN

7th BTTB Workshop

CERN, 14 – 18 January 2019

About This Tutorial

- This tutorial will go step-by-step through setting up and running a simulation with Allpix Squared
  - The slides will contain all commands typed on the terminal/show all changes to configuration files
  - Following along with your computer on lxplus is **strongly encouraged**!
  - You can also follow with a local installation, but we do not want to start debugging local Geant4/ROOT6 installations during this session

- 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

- The latter part will move towards developing your own modules to provide custom output/functionality

CVMFS – CernVM File System

"provides a scalable, reliable and low-maintenance software distribution service. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on the worldwide-distributed computing infrastructure used to run data processing applications"

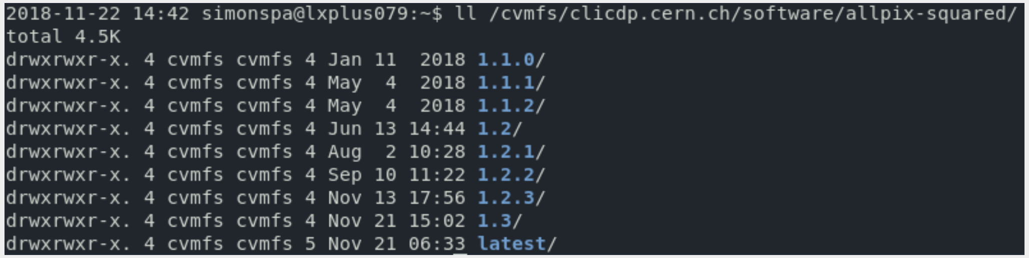
https://cernvm.cern.ch/portal/filesystem

- Central installation of software for SLC6 and CC7
- 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 SLC6 and CC7:
  /1.3/x86_64-centos7-gcc7-opt/
  /1.3/x86_64-slc6-gcc7-opt/



```
2018-11-22 14:42 simonspa@lxplus079:~$ ll /cvmfs/clicdp.cern.ch/software/allpix-squared/
total 4.5K
drwxrwxr-x. 4 cvmfs cvmfs 4 Jan 11  2018 1.1.0/
drwxrwxr-x. 4 cvmfs cvmfs 4 May  4  2018 1.1.1/
drwxrwxr-x. 4 cvmfs cvmfs 4 May  4  2018 1.1.2/
drwxrwxr-x. 4 cvmfs cvmfs 4 Jun 13 14:44 1.2/
drwxrwxr-x. 4 cvmfs cvmfs 4 Aug  2 10:28 1.2.1/
drwxrwxr-x. 4 cvmfs cvmfs 4 Sep 10 11:22 1.2.2/
drwxrwxr-x. 4 cvmfs cvmfs 4 Nov 13 17:56 1.2.3/
drwxrwxr-x. 4 cvmfs cvmfs 4 Nov 21 15:02 1.3/
drwxrwxr-x. 4 cvmfs cvmfs 5 Nov 21 06:33 latest/
```

- Load all dependencies, C++ libraries & set up $PATH using *setup.sh* file:

```
$ source /cvmfs/clicdp.cern.ch/software/allpix-squared/1.3/x86_64-centos7-gcc7-opt/setup.sh
$ allpix --version
Allpix Squared version v1.3
        built on 2018-11-21, 14:00:57 UTC
```

“Docker is a computer program that performs operating-system-level virtualization, also known as *containerization*”

https://en.wikipedia.org/wiki/Docker_(software)

- Not a traditional virtualization, where full machine is emulated
- Run software in container, encapsuled from operating system
  - All dependencies met inside Docker container
  - Very little performance penalty compared to native execution
- Software distributed via “Docker images”
  - Hosted in “docker registries”

- CERN's GitLab instance provides Docker registries for each project:
  https://gitlab.cern.ch/allpix-squared/allpix-squared/container_registry
- How to use this? The user manual has it:
  - Start an interactive shell inside the Docker:

```
$ docker run --interactive --tty                                   \
        --volume "$(pwd)":/data                                    \
        --name=allpix-squared                                      \
        gitlab-registry.cern.ch/allpix-squared/allpix-squared:v1.3 \
        bash
```

  - Directly start a simulation:

```
$ docker run --tty  --rm                                           \
        --volume "$(pwd)":/data                                    \
        --name=allpix-squared                                      \
        gitlab-registry.cern.ch/allpix-squared/allpix-squared:v1.3 \
        "allpix -c my_simulation.conf"
```

Cheat Sheet:

--tty	Allocate a pseudo-TTY
--rm	Automatically remove container when it exits
--interactive	Keep STDIN open even if not attached
--volume	Bind mount a volume
--name	Assign a name to the container

Compiling from Source #1 – Satisfy Dependencies

- Allpix Squared depends on
  - ROOT6 - objects, object history, storing/reading from/to files
  - Eigen - library for fast algebra, used by some modules
- Standard installation will require
  - Geant4 - particle interaction with matter & tracking
- On SLC6/CC7 machines, all is ready:

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

Compiling from Source #2 – Configuring the Build

- 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 #3 – Building

- Well…

```
$ make -j20
$ make install
```

(assuming you have a 20-core machine…)

- Wait a bit
- Run.

A Note on Visualization

- We currently use the Qt visualization viewer by Geant4
- Requires the Geant4 version used to be built with options

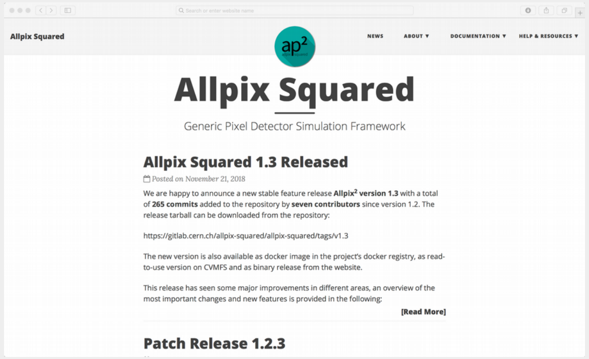
  ```
  -DGEANT4_USE_QT=ON
  -DGEANT4_USE_OPENGL_X11=ON
  ```

  enabled
- Unfortunately currently not available on CVMFS installation of Geant4
- Working on resolving this issue

Obtaining the Code

- A reminder: all resources are linked to from the project page:

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



Allpix Squared

Generic Pixel Detector Simulation Framework

Allpix Squared 1.3 Released

Posted on November 21, 2018

We are happy to announce a new stable feature release **Allpix^2 version 1.3** with a total of **265 commits** added to the repository by **seven contributors** since version 1.2. The release tarball can be downloaded from the repository:

https://gitlab.cern.ch/allpix-squared/allpix-squared/tags/v1.3

The new version is also available as docker image in the project's docker registry, as read-to-use version on CVMFS and as binary release from the website.

This release has seen some major improvements in different areas, an overview of the most important changes and new features is provided in the following:

[Read More]

Patch Release 1.2.3

- We will work on lxplus for this tutorial
  - 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
$ source etc/scripts/setup_lxplus.sh
```

Looking around - what’s there?

```
$ ls -l
```

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
cmake	→ Macros for cmake, formatting tools to make code style consistent
doc	→ Documentation including user manual (see website for easy-to-use version)
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

Looking around - Modules

```
$ ls -l src/modules
CapacitiveTransfer
CorryvreckanWriter
DefaultDigitizer
DepositionGeant4
DetectorHistogrammer
ElectricFieldReader
GenericPropagation
GeometryBuilderGeant4
LCIOWriter
MagneticFieldReader
ProjectionPropagation
RCEWriter
ROOTObjectReader
ROOTObjectWriter
SimpleTransfer
TextWriter
VisualizationGeant4
```

The most important ones (for this tutorial):

- GeometryBuilderGeant4
  - Builds the geometry that will be used by Geant4
- DepositionGeant4
  - Calls Geant4 to step particles through the geometry
- GenericPropagation
  - Propagates charges through the sensor – the main physics sim!
- DefaultDigitizer
  - Describes the digitization by FE electronics

Compiling the code

- Compilation of the code is straightforward using cmake
- Install command will place all libraries and executables in the right place
  - Libraries placed in allpix-squared/lib
  - Executables placed in allpix-squared/bin

```
$ mkdir build && cd build/
$ cmake ..
$ make install -j 4
$ cd ../examples/
```

Starting up a simulation

- Will make a new configuration from scratch
  - Create file **tutorial-simulation.conf**

- Configuration files are based on **[sections]** and use key-value pairs
  - Each section is related to an individual module, with the exception of the **[Allpix]** section which contains the global simulation configuration - most importantly the number of events and the geometry
  - Without these two global objects, Allpix Squared will not run
  - Many different types can be input via the config files - strings, integers, doubles, vectors/arrays, etc

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

Geometry definition

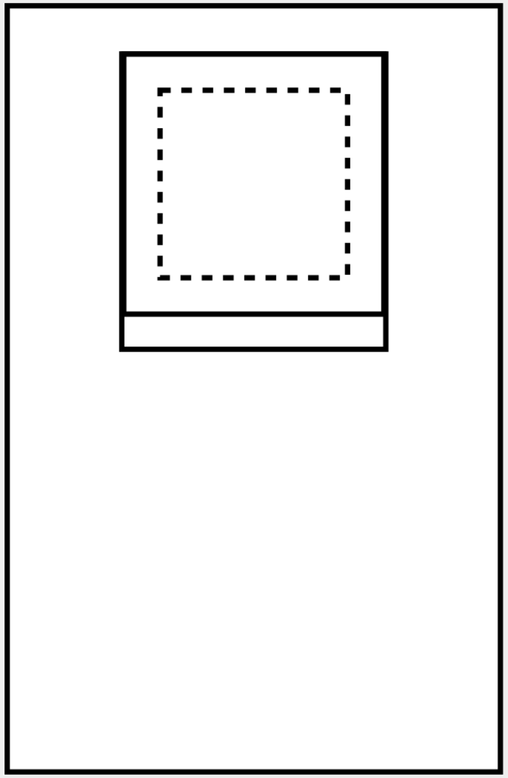
- Looking in the models folder the list of currently known detectors can be seen
  - A new detector model can be built, or an existing detector used
  - For this example, we will pick the **timepix** model
- The geometry configuration file determines which detector are used
  - Each detector is given a unique name (detector1 here) and placed in the global co-ordinate system at a certain position with a given rotation
  - Create geometry file **tutorial-geometry.conf**

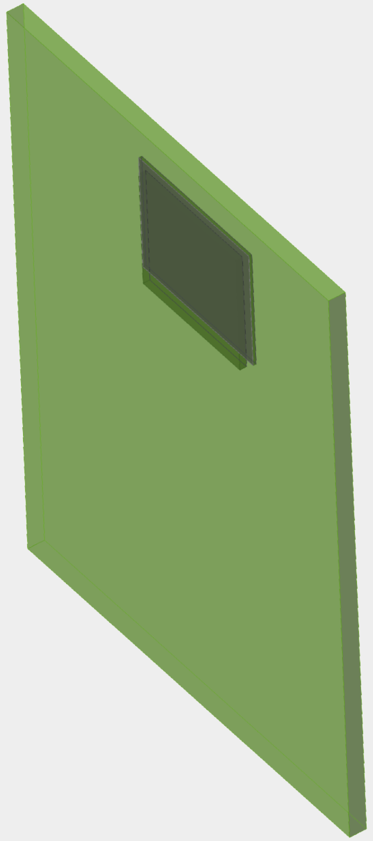
```
[detector1]
type = "timepix"
position = 0mm 0mm 0mm
orientation = 0 0 0
```

```
$ ls ../models/
clicpix.conf
clicpix2.conf
cmsp1.conf
diode.conf
fei3.conf
ibl_planar.conf
medipix3.conf
mimosa23.conf
mimosa26.conf
test.conf
timepix.conf
velopix.conf
```

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 algorithms

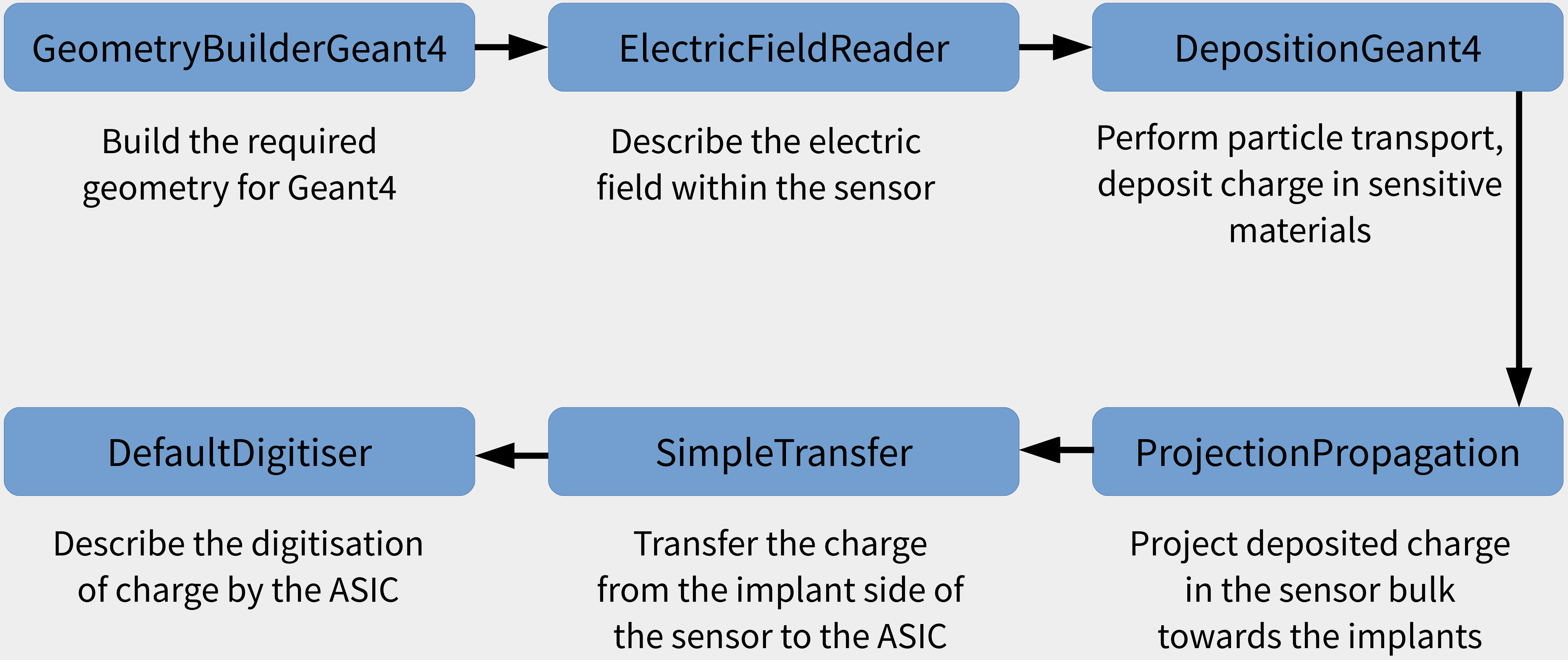
- We now have a simulation setup that doesn't do anything

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

- Can now start to add algorithms
  - 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
  - Creation of the Geant4 geometry, description of the electric field in the sensor
  - Generation and transport of particles through the geometry
  - Propagation of the deposited charges
  - Transfer of these charges to the electronics
  - Description of the electronics

Simple simulation flow



Build the required geometry for Geant4

Describe the electric field within the sensor

Perform particle transport, deposit charge in sensitive materials

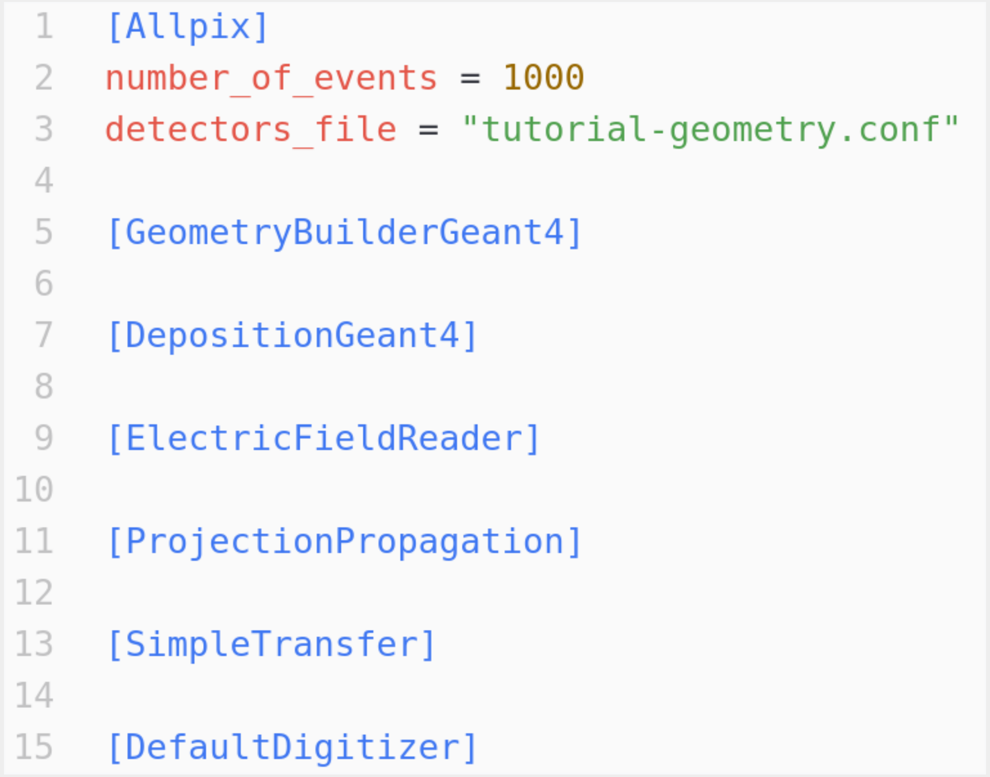
Project deposited charge in the sensor bulk towards the implants

Transfer the charge from the implant side of the sensor to the ASIC

Describe the digitisation of charge by the ASIC

Simple simulation flow

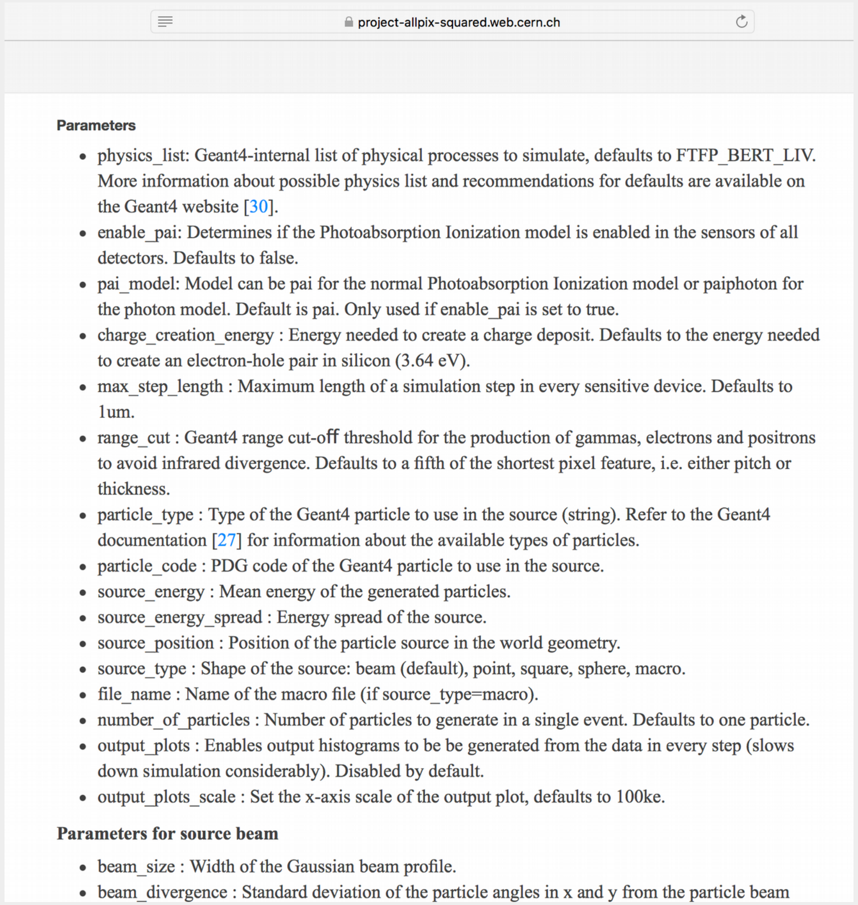
- Edit **tutorial-simulation.conf** to include the list of algorithms that we want to use



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

[GeometryBuilderGeant4]

[DepositionGeant4]

[ElectricFieldReader]

[ProjectionPropagation]

[SimpleTransfer]

[DefaultDigitizer]
```

Available parameters

- All modules described in detail in the Allpix Squared manual
  - Also shows list of available parameters, along with default values and typical use example
  - https://cern.ch/allpix-squared/usermanual/allpix-manualch7.html



Parameters

- physics_list: Geant4-internal list of physical processes to simulate, defaults to FTFP_BERT_LIV. More information about possible physics list and recommendations for defaults are available on the Geant4 website [30].
- enable_pai: Determines if the Photoabsorption Ionization model is enabled in the sensors of all detectors. Defaults to false.
- pai_model: Model can be pai for the normal Photoabsorption Ionization model or paiphoton for the photon model. Default is pai. Only used if enable_pai is set to true.
- charge_creation_energy : Energy needed to create a charge deposit. Defaults to the energy needed to create an electron-hole pair in silicon (3.64 eV).
- max_step_length : Maximum length of a simulation step in every sensitive device. Defaults to 1um.
- range_cut : Geant4 range cut-off threshold for the production of gammas, electrons and positrons to avoid infrared divergence. Defaults to a fifth of the shortest pixel feature, i.e. either pitch or thickness.
- particle_type : Type of the Geant4 particle to use in the source (string). Refer to the Geant4 documentation [27] for information about the available types of particles.
- particle_code : PDG code of the Geant4 particle to use in the source.
- source_energy : Mean energy of the generated particles.
- source_energy_spread : Energy spread of the source.
- source_position : Position of the particle source in the world geometry.
- source_type : Shape of the source: beam (default), point, square, sphere, macro.
- file_name : Name of the macro file (if source_type=macro).
- number_of_particles : Number of particles to generate in a single event. Defaults to one particle.
- output_plots : Enables output histograms to be be generated from the data in every step (slows down simulation considerably). Disabled by default.
- output_plots_scale : Set the x-axis scale of the output plot, defaults to 100ke.

Parameters for source beam

- beam_size : Width of the Gaussian beam profile.
- beam_divergence : Standard deviation of the particle angles in x and y from the particle beam

Defining particles

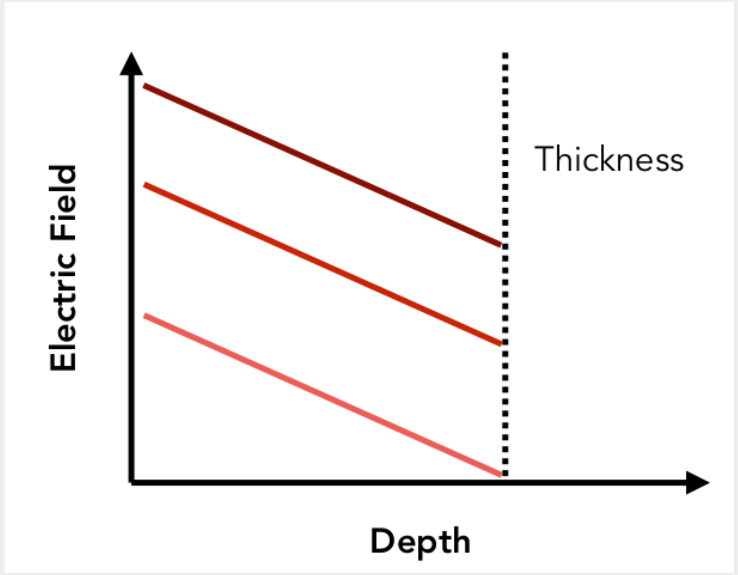
- DepositionGeant4 has several parameters, and is used as the source of particles in addition to interfacing geant4
  - Choose the type and energy of the particles that we want
  - Define the starting point and direction of the beam, in addition to the size of the beam
  - Pick a suitable physics list

```
[DepositionGeant4]
particle_type = "Pi+"
source_energy = 120GeV
source_type = "beam"
beam_size = 3mm
source_position = 0um 0um -200mm
beam_direction = 0 0 1
physics_list = FTFP_BERT_EMZ
```

Electric field definition

- ElectricFieldReader can generate electric fields for the sensor in several ways
- The simplest is a linear field approximation, using a user-defined depletion voltage and applied bias voltage
  - Higher bias voltages increase the electric field as expected
  - No attempt is made to describe focusing effects around the implants
- A more complete field can be added by converting the output of FE simulations such as TCAD

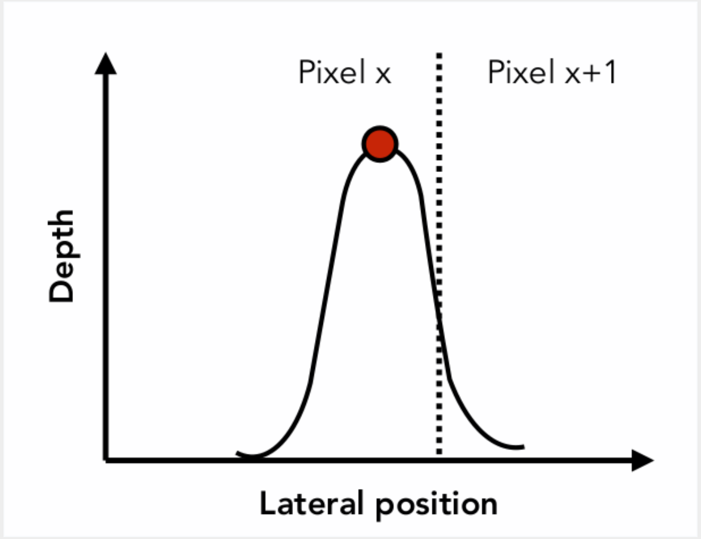
```
[ElectricFieldReader]
model="linear"
bias_voltage=-50V
depletion_voltage=-30V
```



Propagation of deposited charges

- ProjectionPropagation is a relatively simple way to propagate deposited charges towards the collection implants
  - Charges are picked up in discrete groups
  - The diffusion constant is calculated, after calculation of the drift time given the current position and electric field
  - Charge is smeared according to a Gaussian distribution, using the calculated diffusion constant
  - Pixel boundaries are used to determine how much charge is deposited in each pixel

```
[ProjectionPropagation]
temperature = 293K
```



Transferring charge

- Not all charge that is propagated will necessarily end up on the collection implant
  - For under-depleted sensors there could be charge still in the low-field region
  - For sensors with radiation damage charge trapping will occur in the bulk
- For this we use the concept of transferring the charge from the sensor to the input of the electronics
  - Also allows for simple extension to capacitive coupling between sensor and electronics
- The default module for simple DC-coupled detectors is SimpleTransfer
  - All charges with x microns of the implant are considered collected - defaults to 5 µm

```
[SimpleTransfer]
```

Digitization

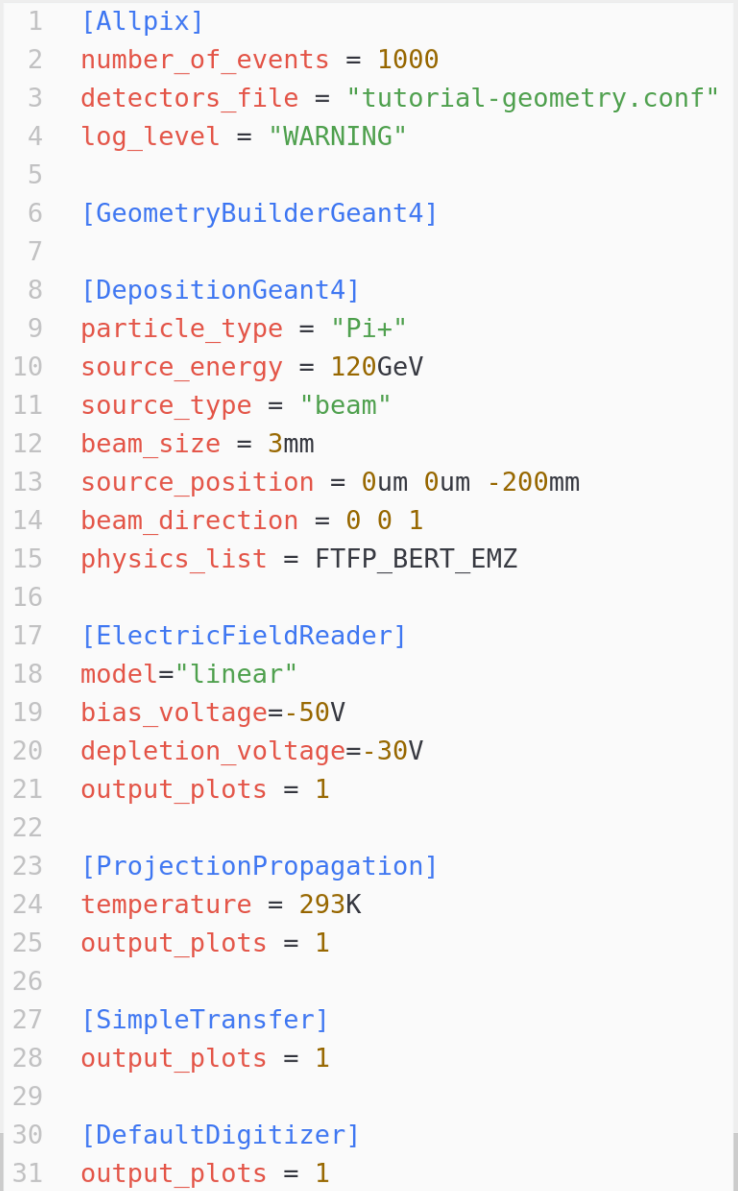
- Many front-end chips feature similar kinds of effects
  - Gaussian noise on the collected charge
  - A threshold level
  - An ADC with a certain gain
- All of these features, with additional features such as threshold dispersion/gain variation are implemented in the DefaultDigitizer
  - Can be easily configured to produce a Time-over-Threshold style (ToT) digitisation

```
[DefaultDigitizer]
```

Updated simulation configuration

- Now we have a simulation set up that will shoot 120 GeV pions at a timepix detector, propagate charges through the sensor with our desired electric field, and digitize the resulting collected charge

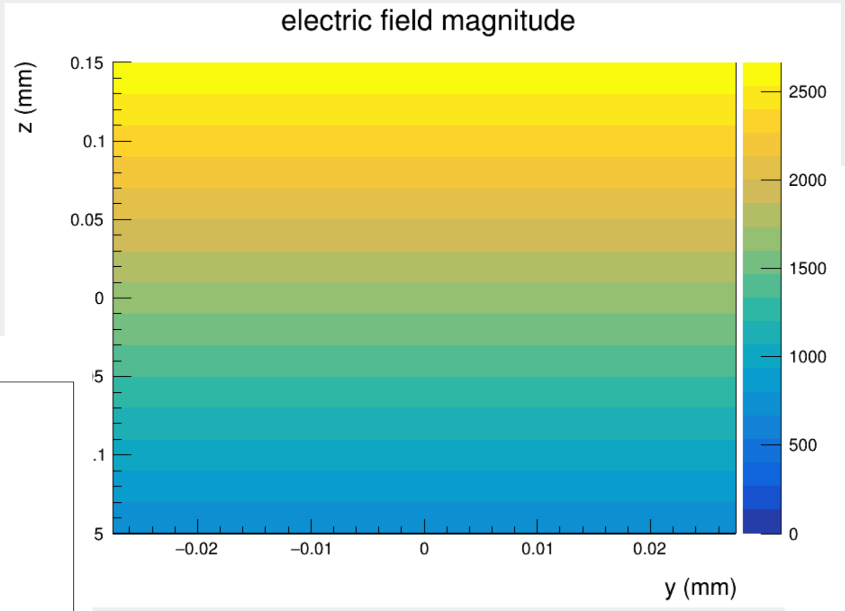
- A few tips make running the simulation easier:
    - The **log_level** flag, which changes the quantity of information output by modules,
    - The **output_plots** flag, which can be set per module in order to get additional debug output

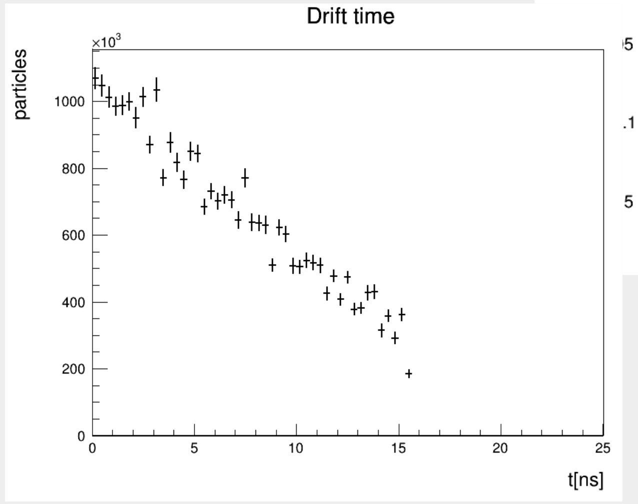


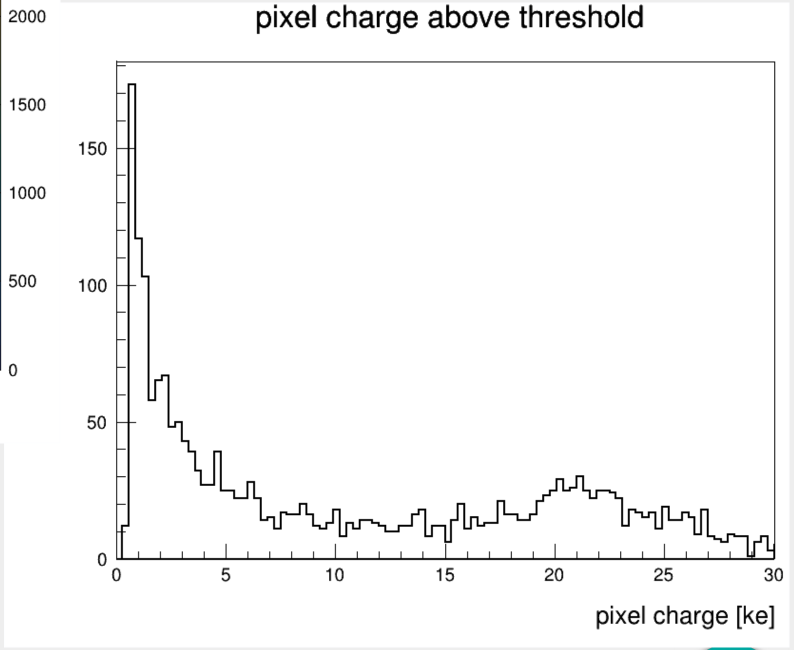
```
[Allpix]
number_of_events = 1000
detectors_file = "tutorial-geometry.conf"
log_level = "WARNING"

[GeometryBuilderGeant4]

[DepositionGeant4]
particle_type = "Pi+"
source_energy = 120GeV
source_type = "beam"
beam_size = 3mm
source_position = 0um 0um -200mm
beam_direction = 0 0 1
physics_list = FTFP_BERT_EMZ

[ElectricFieldReader]
model="linear"
bias_voltage=-50V
depletion_voltage=-30V
output_plots = 1

[ProjectionPropagation]
temperature = 293K
output_plots = 1

[SimpleTransfer]
output_plots = 1

[DefaultDigitizer]
output_plots = 1
```

Example plots

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







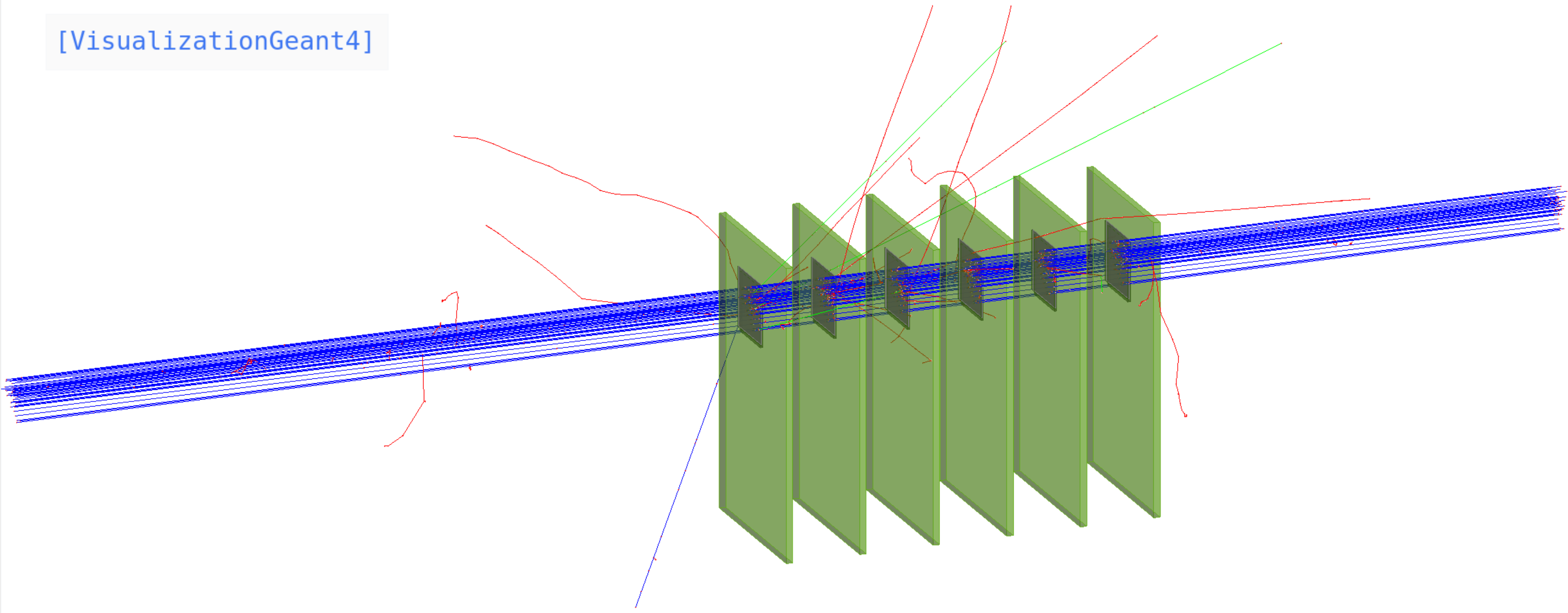
Adding more detectors

- In the same way that we had a single detector in our geometry file, it is trivial to add subsequent detectors
  - Each detector is simply placed in the same way in the global coordinate system
- For our purposes, we can add a further 5 timepix detectors to produce a telescope setup, with the detectors spaced by 20 mm in z
- For such scenarios, it is extremely useful to visualize the setup using some of the built-in Geant4 viewing tools
  - Unfortunately these do not run on lxplus - default installation is without these tools compiled

```
[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
```

Visualizing the setup

```
[VisualizationGeant4]
```



Processing detectors in different ways

- When we added more detectors to the geometry file, everything took care of things under the hood
  - No need to add additional information to the simulation configuration file

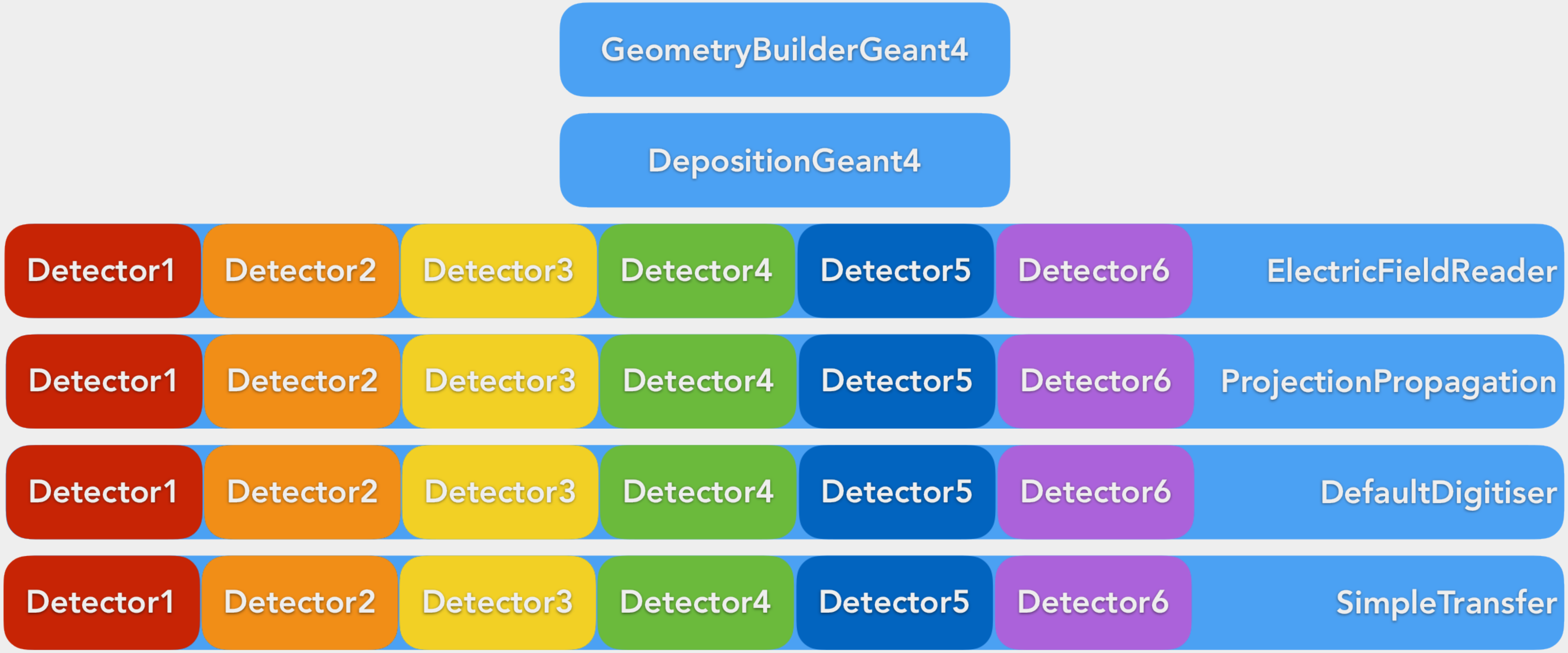
- What is happening? A separate instance of each module is created per detector
  - This allows some measure of multi-threading to be used to improve simulation times - all detectors can be run in parallel

- This behavior is controlled by the module type, either it is **unique** or **detector**-specific

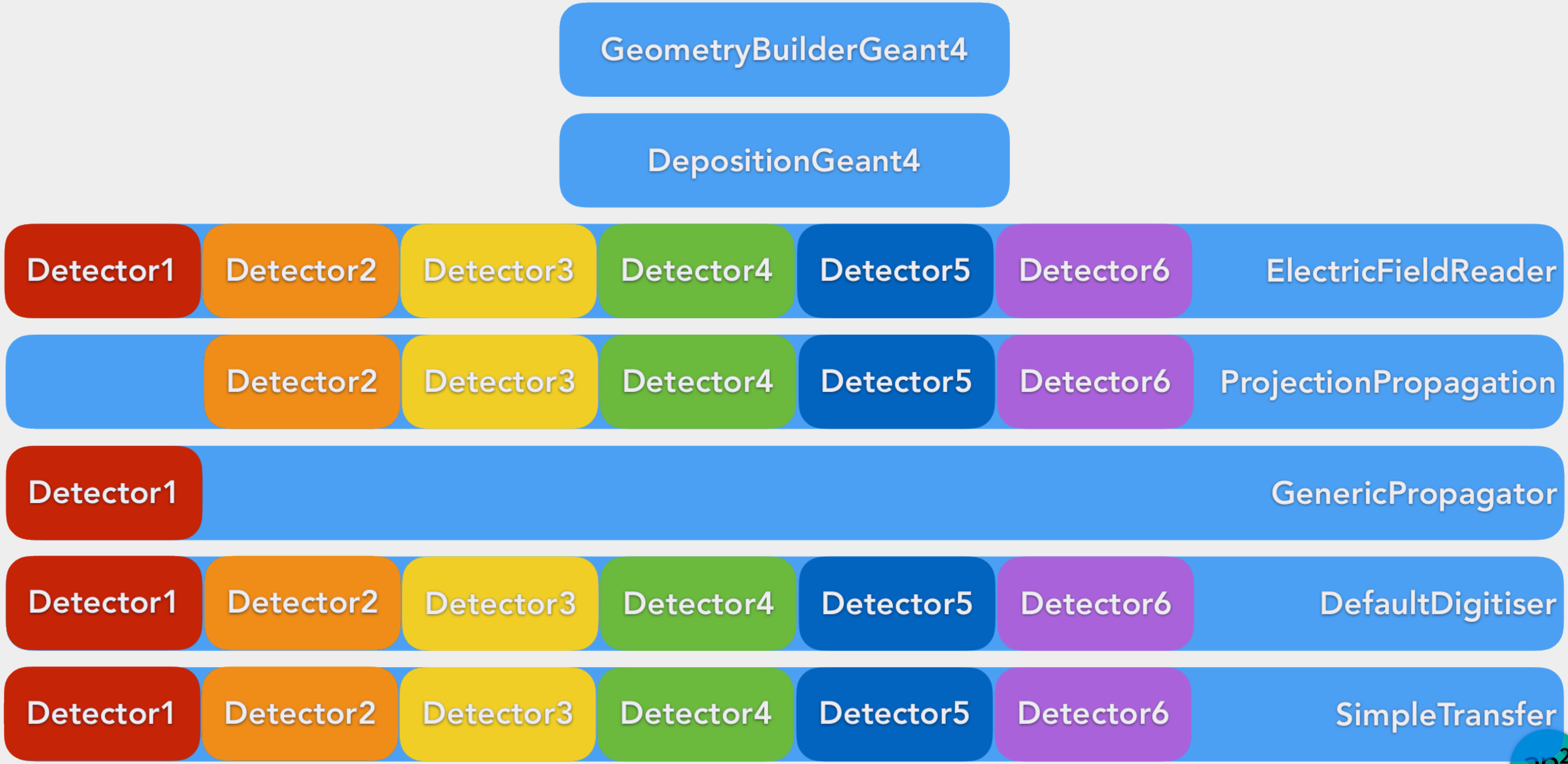
A single detector chain

- GeometryBuilderGeant4
- DepositionGeant4
- ElectricFieldReader
- ProjectionPropagation
- DefaultDigitiser
- SimpleTransfer

A multi-detector chain



What if ?


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

Specifying detector type/name

- When we added more detectors to the geometry file, everything was taken care of
  - No need to specify which detector each module was being applied to
- By default, all modules will apply to all detectors. Can overwrite this behavior by specifying either the name or type of detector to run over
  - We can use this to either make a module have different parameters per-detector, or operate on a subset of detectors

Instantiation to operate on all detectors

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

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

Instantiation for detector1 will be overwritten by this one, since it is the same type of module and specified only for detector1

Specifying detector type/name

- When we added more detectors to the geometry file, everything took care of things under the hood
    - No need to specify which detector each module was being applied to
- By default, all modules will apply to all detectors. Can overwrite this behavior by specifying either the name or type of detector to run over
    - We can use this to either make a module have different parameters per-detector, or **operate on a subset of detectors**

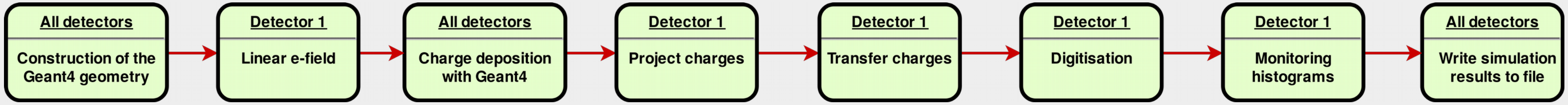
```
[ProjectionPropagation]
name = "detector2", "detector3", "detector4", "detector5", "detector6"
temperature = 293K

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

Specifying detector type/name

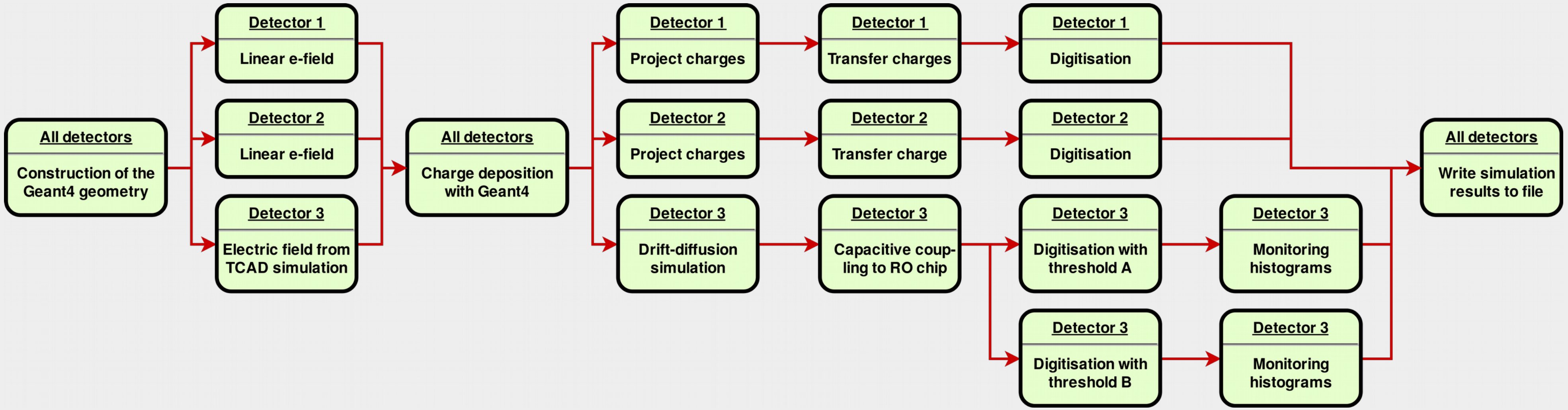
```
[Allpix]
number_of_events = 1000
detectors_file = "tutorial-geometry.conf"
log_level = "WARNING"

[GeometryBuilderGeant4]

[DepositionGeant4]
particle_type = "Pi+"
source_energy = 120GeV
source_type = "beam"
beam_size = 3mm
source_position = 0um 0um -200mm
beam_direction = 0 0 1
physics_list = FTFP_BERT_EMZ

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

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

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

[GenericPropagation]
name = "detector1"
temperature = 293K

[SimpleTransfer]
output_plots = 1

[DefaultDigitizer]
output_plots = 1
```

Single detector chain



Multi-detector chain

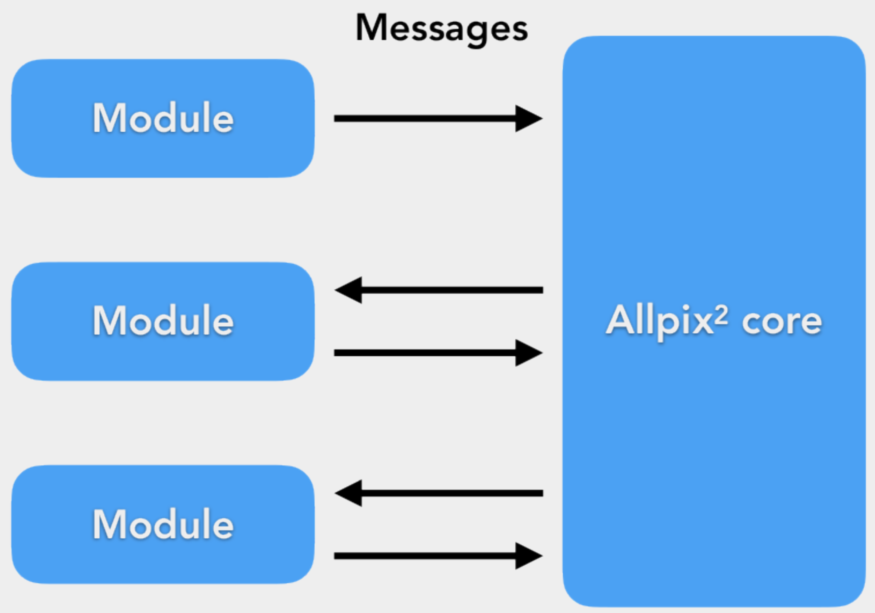


Making your own module

- Until now, setting up a simulation and configuring different modules for different detectors
  - No need to touch c++ code, only config files
- Next step is developing a custom module - keep in mind that modules may already be implemented/can be configured in a way that you need (cf. Digitisation is reasonable generic)
  - Consider that making your module generic will benefit other users - no point in implementing 10 times a module to apply time walk effects to an ASIC
- Useful script comes with the software to make it easy to develop new modules: **make_modules.sh**
  - Define the name of the module
  - Whether the module is **unique** or operates per-**detector**
  - The type of message that the module accepts

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.



Producing your own module

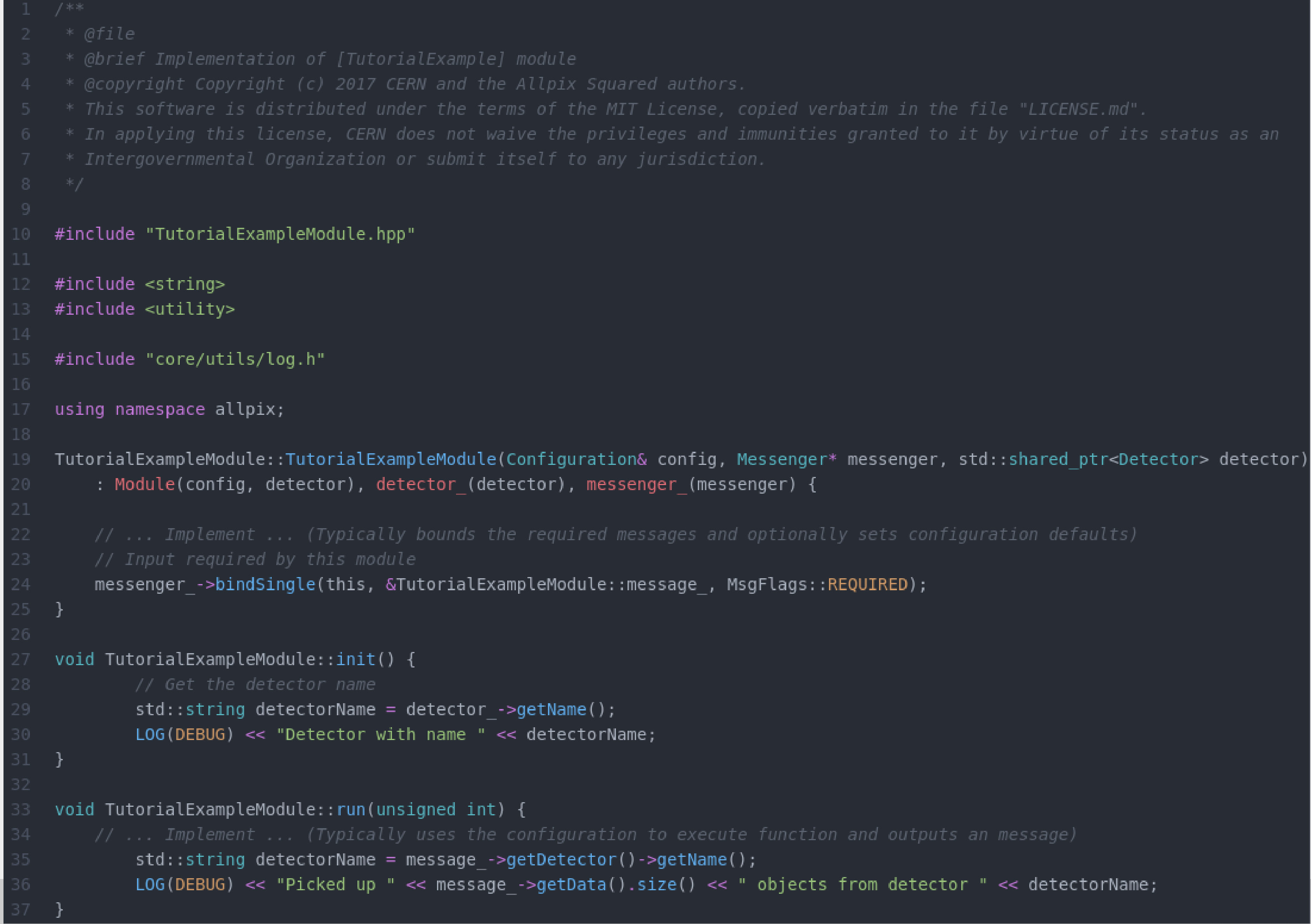
```
$ cd ../etc/scripts/
$ ./make_module.sh
Preparing code basis for a new module:
Name of the module? TutorialExample
Type of the module?
1) unique
2) detector
#? 2
Input message type? DepositedCharge
Creating directory and files...

Name:   TutorialExample
Author: Simon Spannagel (simon.spannagel@cern.ch)
Path:   /home/simonspa/software/allpix-squared/src/modules/TutorialExample
This module listens to "DepositedCharge" messages from one detector

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

Code

Constructor:
Bind to messages

Initialize variables/
histograms etc.

Main code, executed
each event



```cpp
/**
 * @file
 * @brief Implementation of [TutorialExample] module
 * @copyright Copyright (c) 2017 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 "TutorialExampleModule.hpp"

#include <string>
#include <utility>

#include "core/utils/log.h"

using namespace allpix;

TutorialExampleModule::TutorialExampleModule(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, &TutorialExampleModule::message_, MsgFlags::REQUIRED);
}

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

void TutorialExampleModule::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;
}
```

Compiling and including your module

- CMake set up to compile all modules in the corresponding directory
  - Just need to rerun cmake from the build directory and compile
- Module can then be added in the simulation configuration file in the same way as any other module

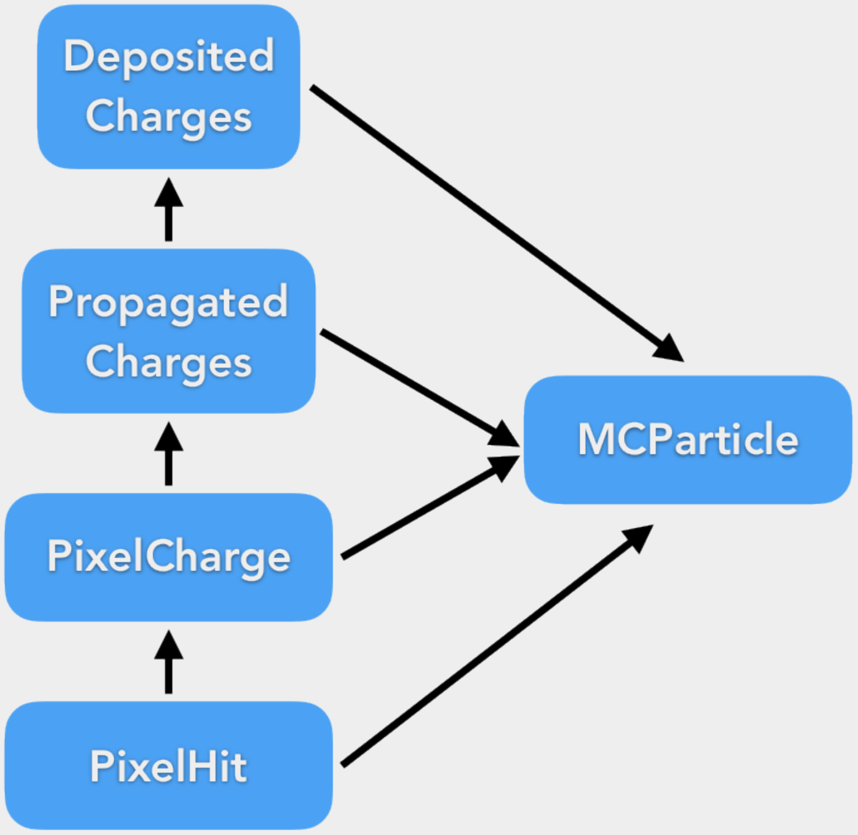
```
$ cd ../../build/
$ cmake ..
$ make install -j 4

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

```
...

[TutorialExample]

...
```

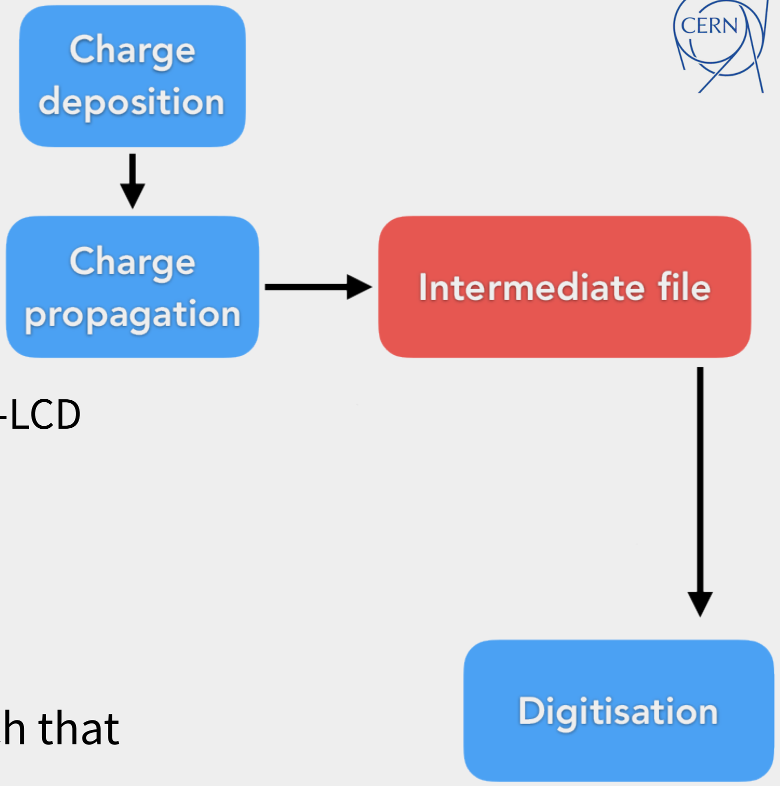
A few other features - MC history

- All objects contain information about where they come from
    - Direct link to the preceding object
    - All objects link back to original MC particle
- 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

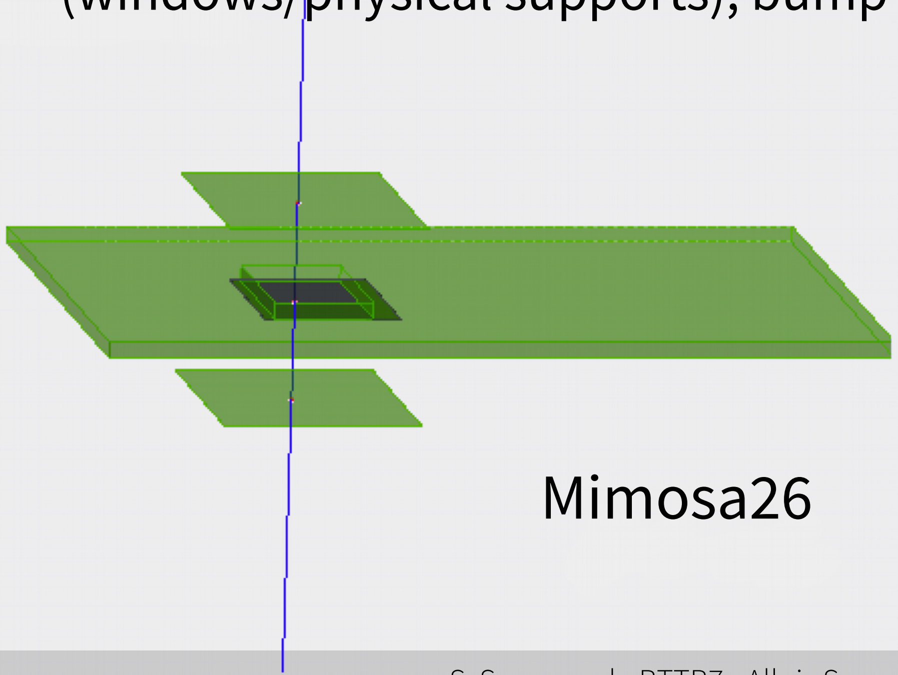


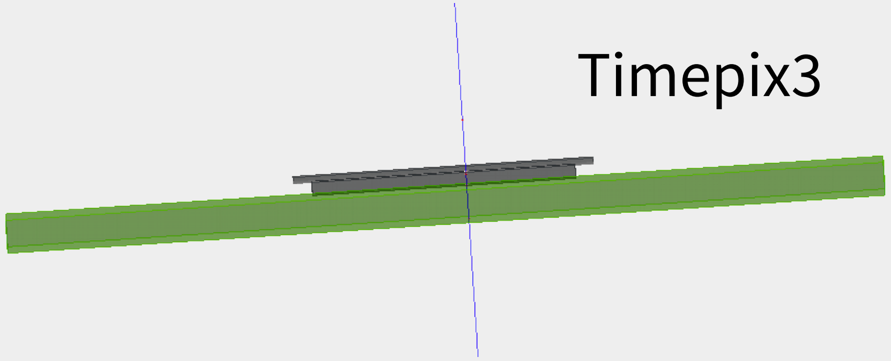
- Several output formats are already supported
  - LCIO - format used in linear collider community
  - RCE - ATLAS pixel group data format
  - Corryvreckan - reconstruction framework developed in EP-LCD
  - Text files - simple human-readable format
  - RootObjects – Allpix Squared data

- This last class writes out native Allpix Squared objects, such that they can be read in again by the framework
  - Allows intermediate file-writing to avoid repeating CPU-intensive parts of the simulation
  - Write out propagated charge objects so that tuning of the digitisation parameters does not require re-running Geant4 and propagation code

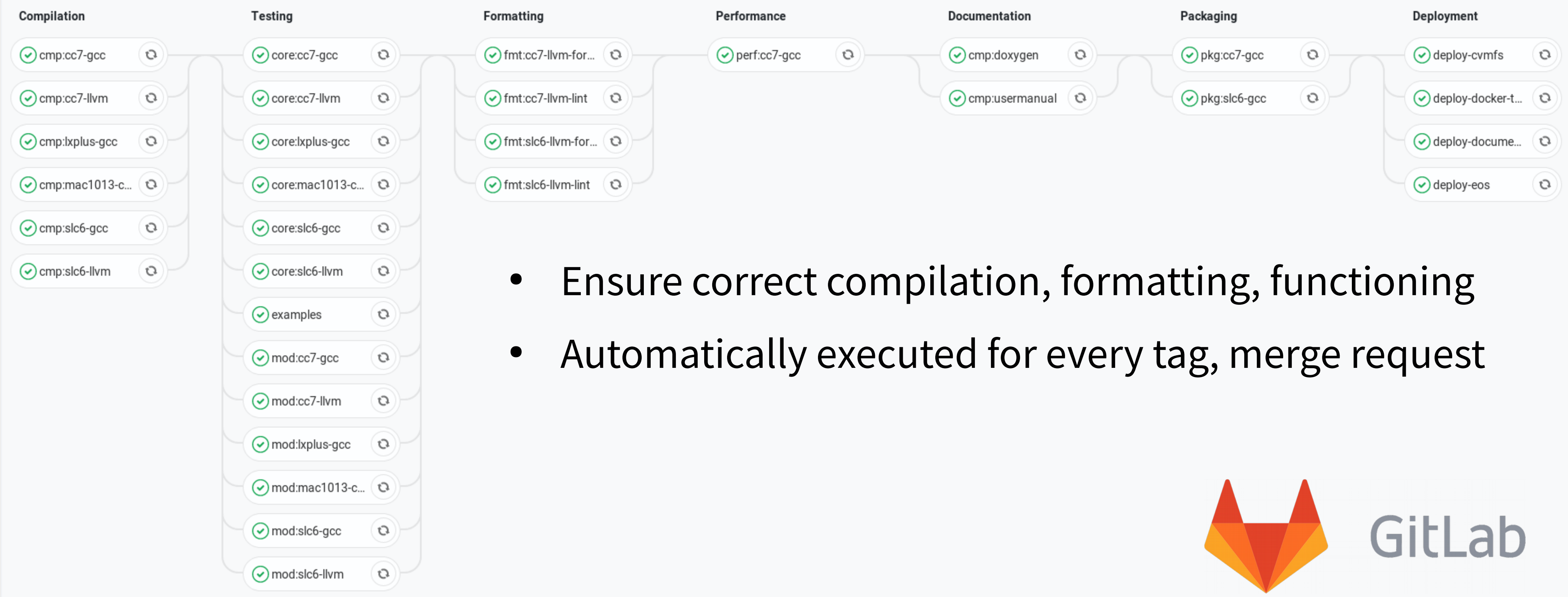
A few other features - geometry

- Currently-implemented geometries are 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 (windows/physical supports), bump dimensions, etc.


Mimosa26


Timepix3

Excursion: Continuous Integration



Compilation	Testing	Formatting	Performance	Documentation	Packaging	Deployment
✓ cmp:cc7-gcc	✓ core:cc7-gcc	✓ fmt:cc7-llvm-for...	✓ perf:cc7-gcc	✓ cmp:doxygen	✓ pkg:cc7-gcc	✓ deploy-cvmfs
✓ cmp:cc7-llvm	✓ core:cc7-llvm	✓ fmt:cc7-llvm-lint		✓ cmp:usermanual	✓ pkg:slc6-gcc	✓ deploy-docker-t...
✓ cmp:lxplus-gcc	✓ core:lxplus-gcc	✓ fmt:slc6-llvm-for...				✓ deploy-docume...
✓ cmp:mac1013-c...	✓ core:mac1013-c...	✓ fmt:slc6-llvm-lint				✓ deploy-eos
✓ cmp:slc6-gcc	✓ core:slc6-gcc					
✓ cmp:slc6-llvm	✓ core:slc6-llvm					
	✓ examples					
	✓ mod:cc7-gcc					
	✓ mod:cc7-llvm					
	✓ mod:lxplus-gcc					
	✓ mod:mac1013-c...					
	✓ mod:slc6-gcc					
	✓ mod:slc6-llvm					

- Ensure correct compilation, formatting, functioning
- Automatically executed for every tag, merge request

GitLab

CI Stages

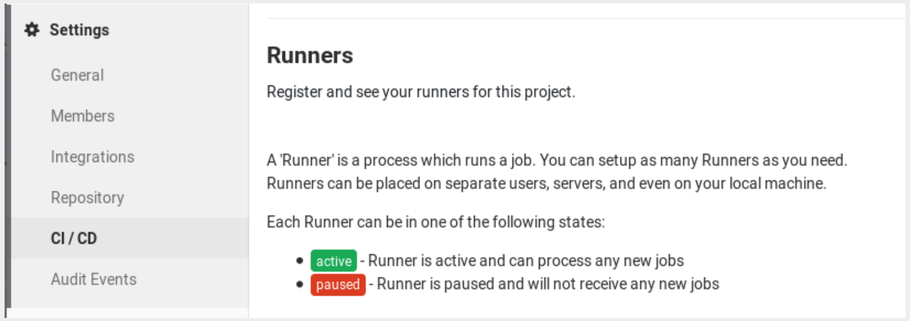
- **Compilation**
  builds the framework from source on Scientific Linux 6, CentOS7, and Mac OS X using GCC, Clang, AppleClang
- **Testing**
  executes functionality tests for the core (correct parsing of configs, coordinate transformations…) and modules (constant behavior, no change to physics)
- **Formatting**
  makes sure that changes concern code, not its style (white spaces, tabs…) and ensures proper usage of recent C++ features (linting, best practices)
- **Performance**
  Execute a few tests with several thousand events, measure execution time

CI Stages

- **Documentation**
  generate user manual from LaTeX sources, generate Doxygen code reference
- **Packaging**
  generate tarballs for deployment/publication
- **Deployment**
  - Publish new version on CVMFS
  - Publish new Docker image in registry
  - Publish user manual and code reference on website
  - Publish binary tarballs on website

Me? How?

- Enable runners (to execute jobs)



Settings

General

Members

Integrations

Repository

CI / CD

Audit Events

Runners

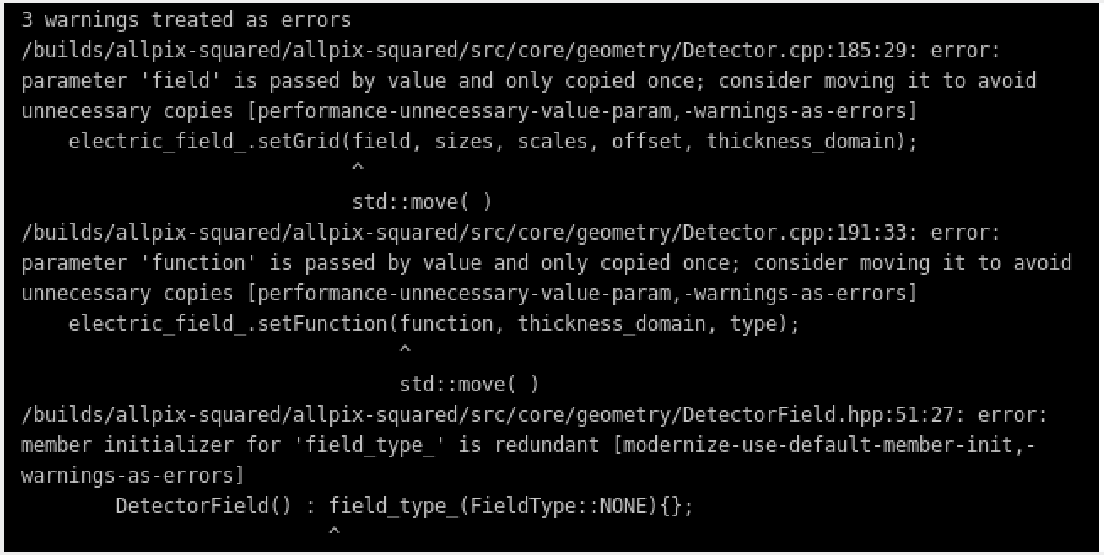
Register and see your runners for this project.

A 'Runner' is a process which runs a job. You can setup as many Runners as you need. Runners can be placed on separate users, servers, and even on your local machine.

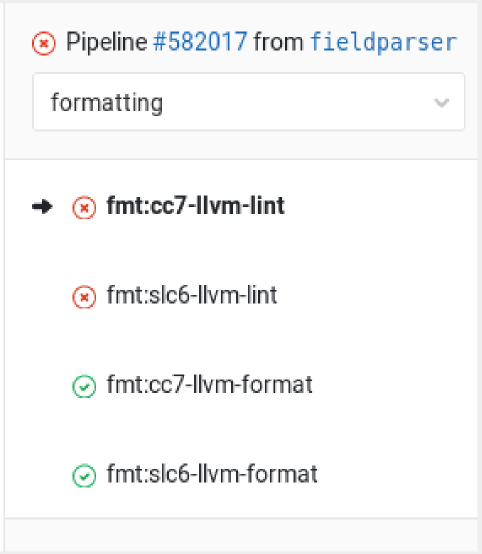
Each Runner can be in one of the following states:

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

- Pipeline fails: do read output of the failing job!



```
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

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

CERN