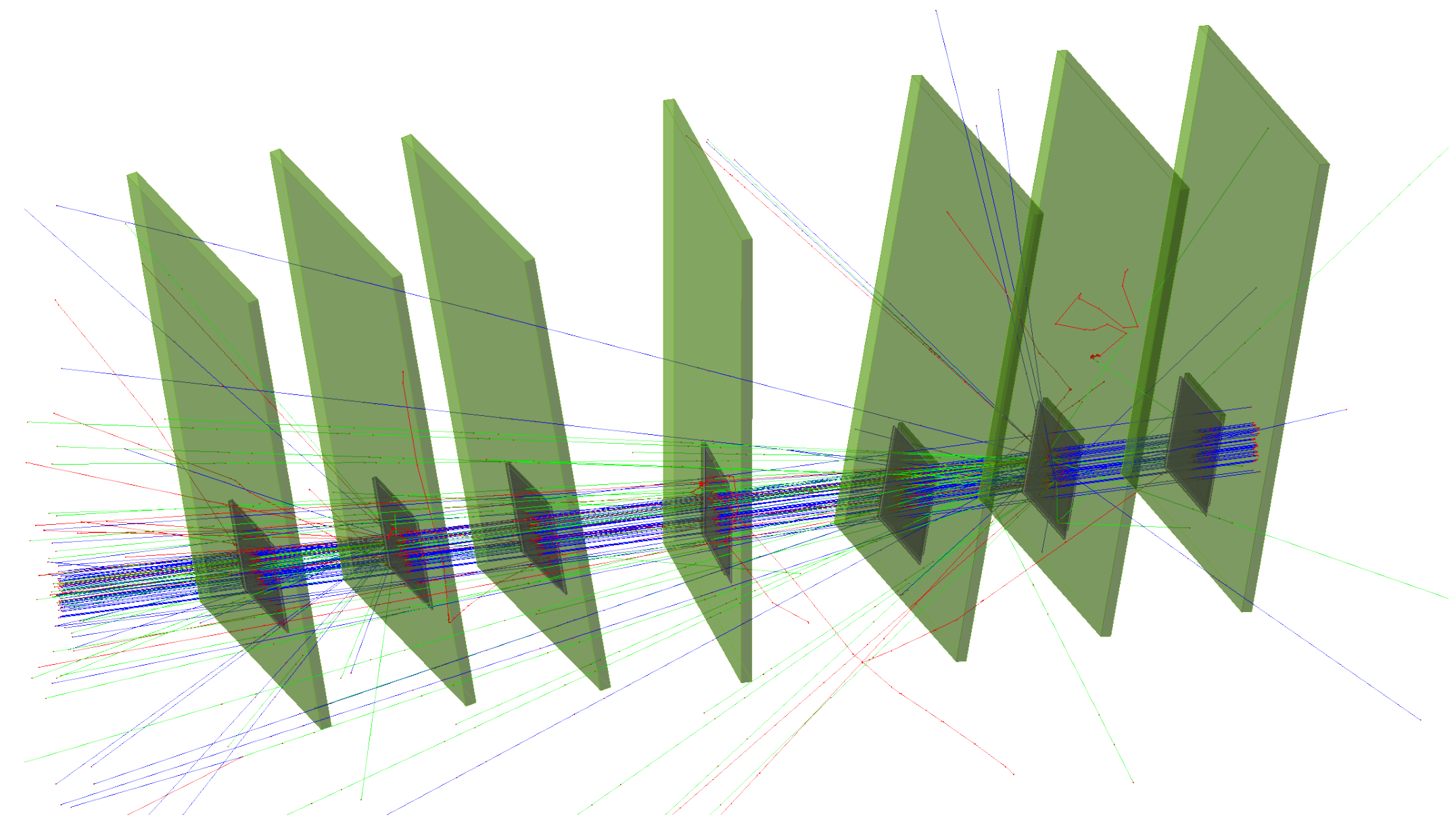


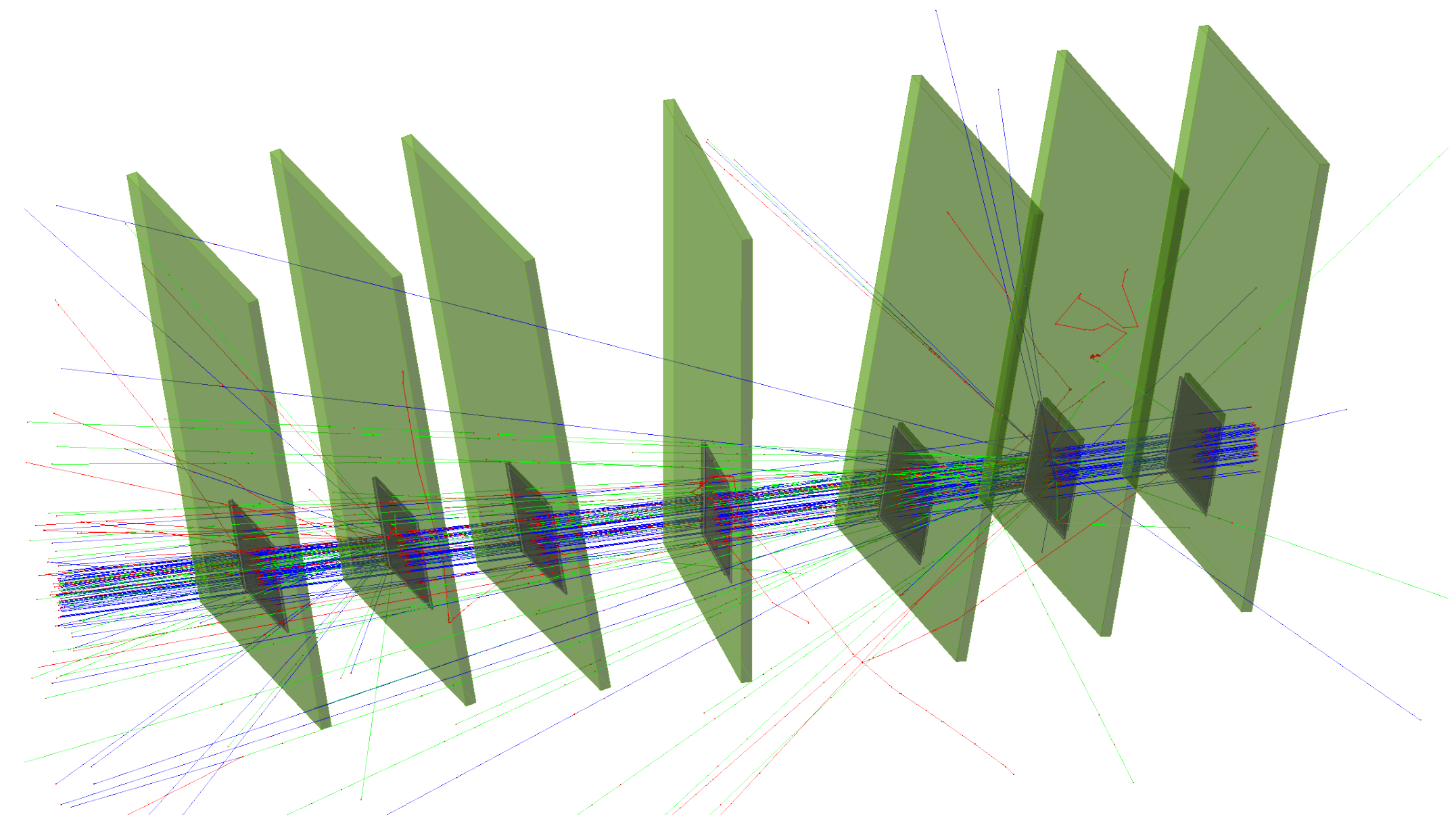
Allpix² step-by-step tutorial

Daniel Hynds

- Hopefully most of you already know what allpix-squared is
 - “A Modular Simulation Framework for Silicon Detectors”
- Some of the main advantages involve not having to code directly in Geant4, the modular style of the software so that code can be easily re-used, and the heavy Continuous Integration + documentation available on the repository
 - <https://project-allpix-squared.web.cern.ch>
 - <https://gitlab.cern.ch/allpix-squared/allpix-squared>
- Allpix-squared handles the description of typical detector geometries, the interface to Geant4 for particle interactions, description of the sensor electric field, propagation of charge deposits through the sensor, description of the Front End (FE) electronics response...

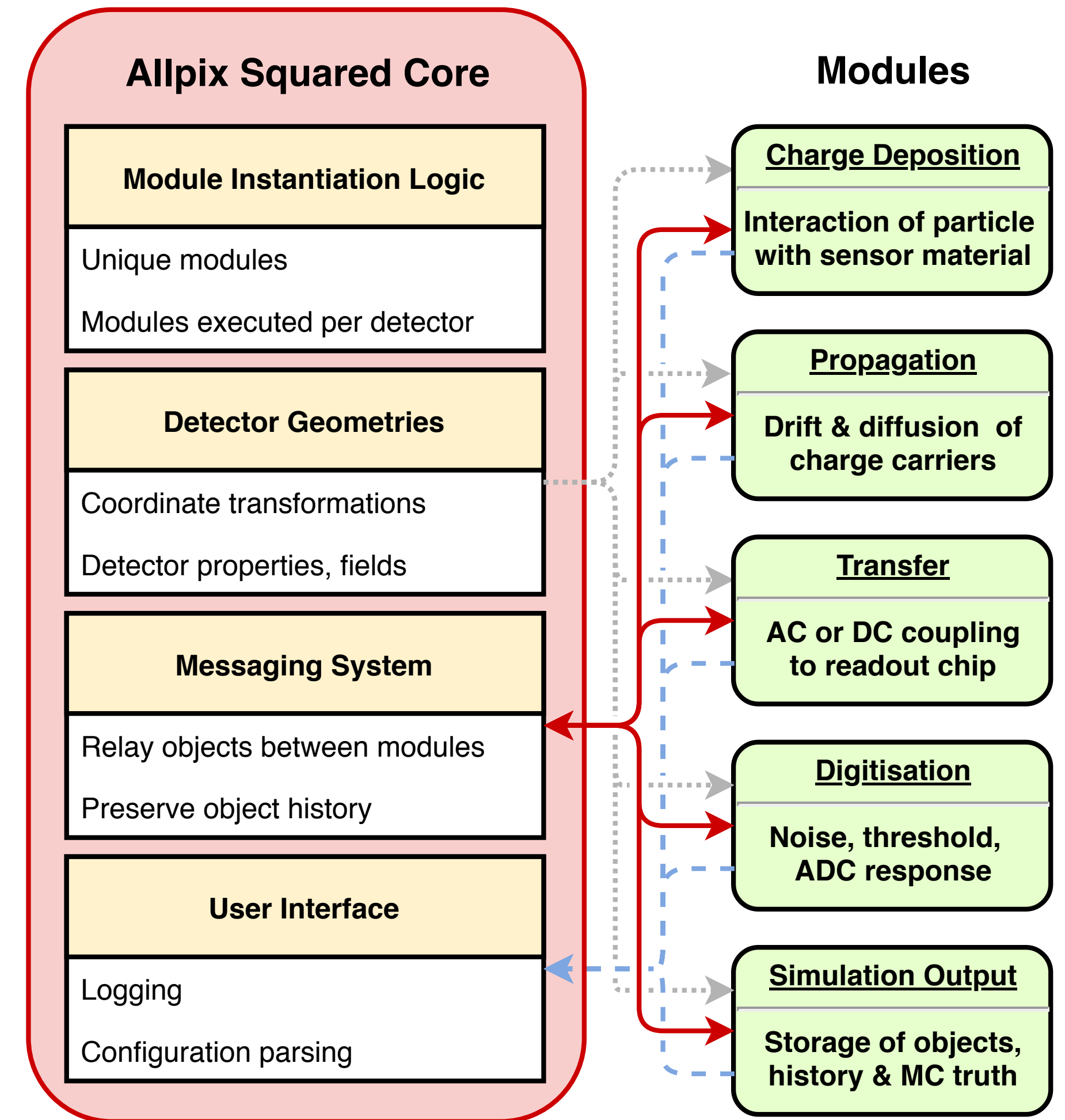


- Hopefully most of you already know what allpix-squared is
 - “A Modular Simulation Framework for ~~Silicon~~ Detectors”
- Some of the main advantages involve not having to code directly in Geant4, the modular style of the software so that code can be easily re-used, and the heavy Continuous Integration + documentation available on the repository
 - <https://project-allpix-squared.web.cern.ch>
 - <https://gitlab.cern.ch/allpix-squared/allpix-squared>
- Allpix-squared handles the description of typical detector geometries, the interface to Geant4 for particle interactions, description of the sensor electric field, propagation of charge deposits through the sensor, description of the Front End (FE) electronics response...

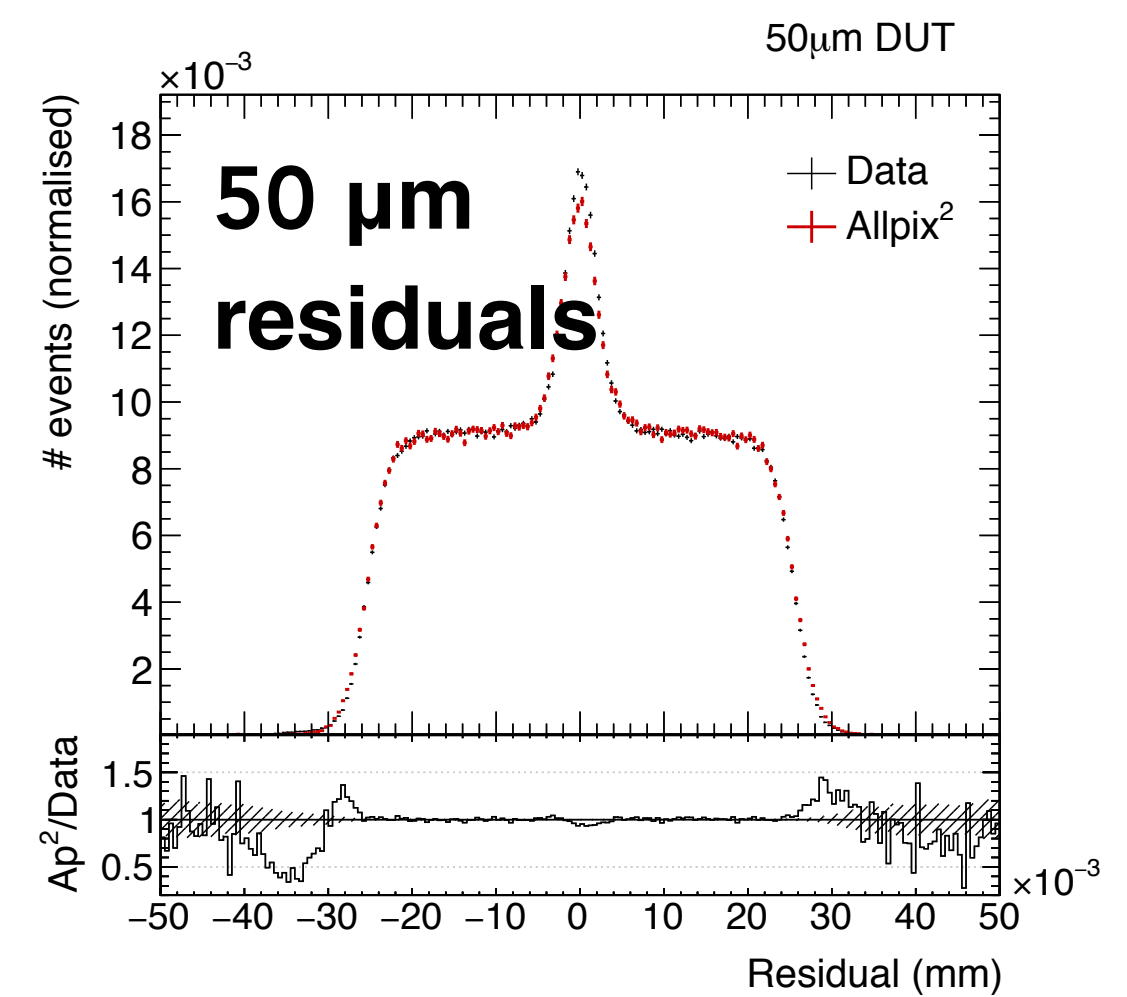
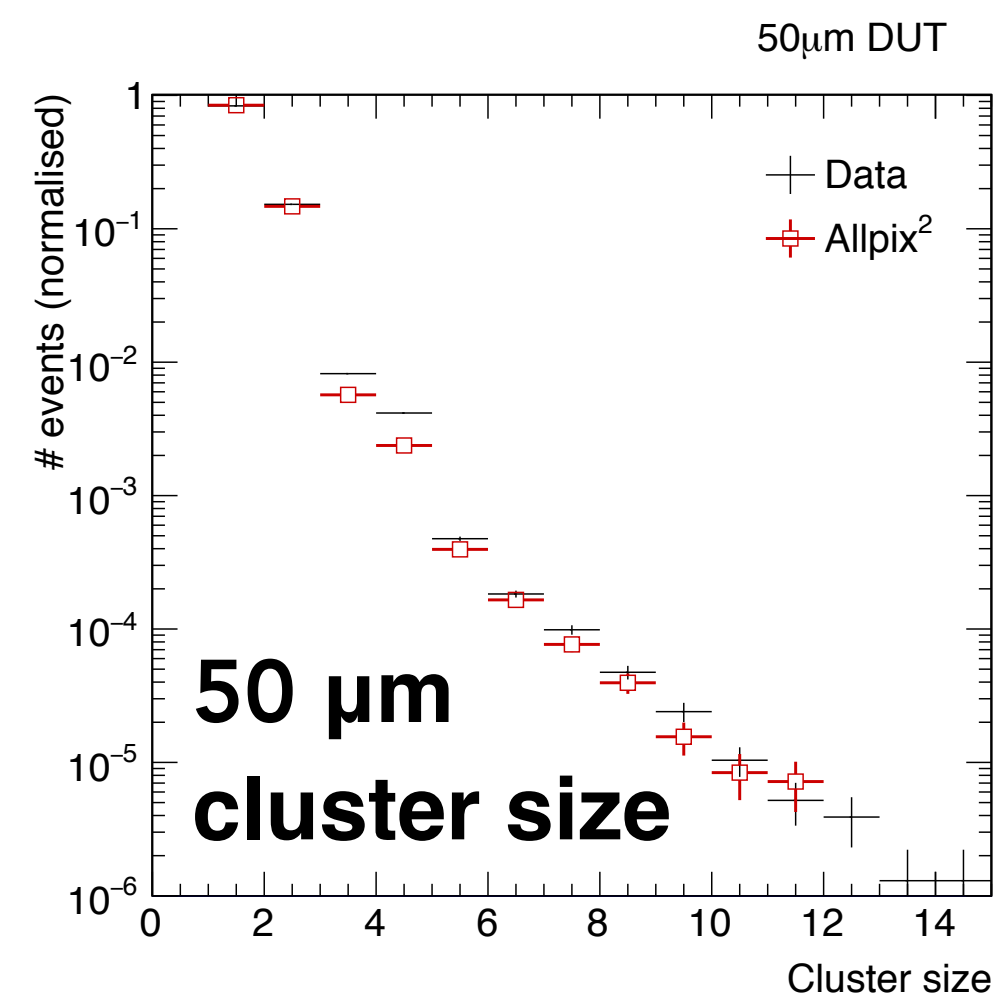
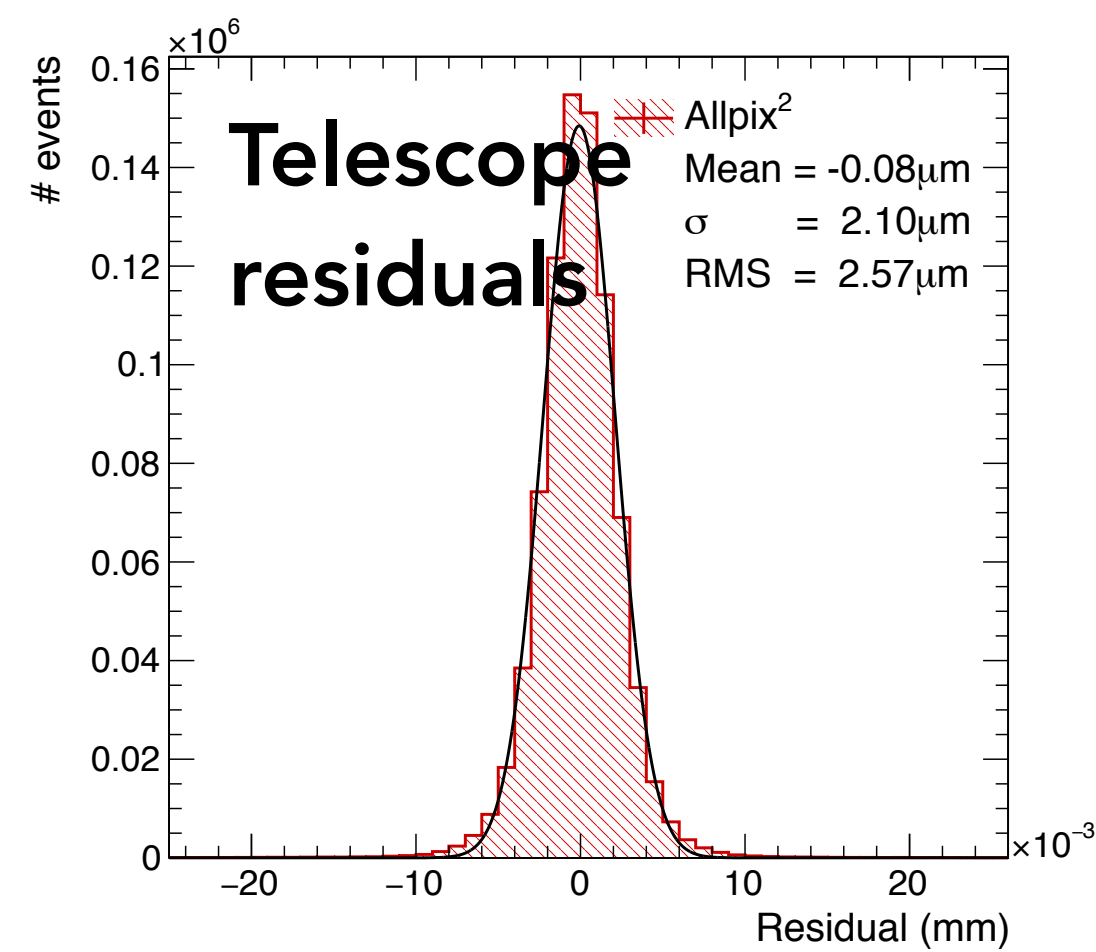
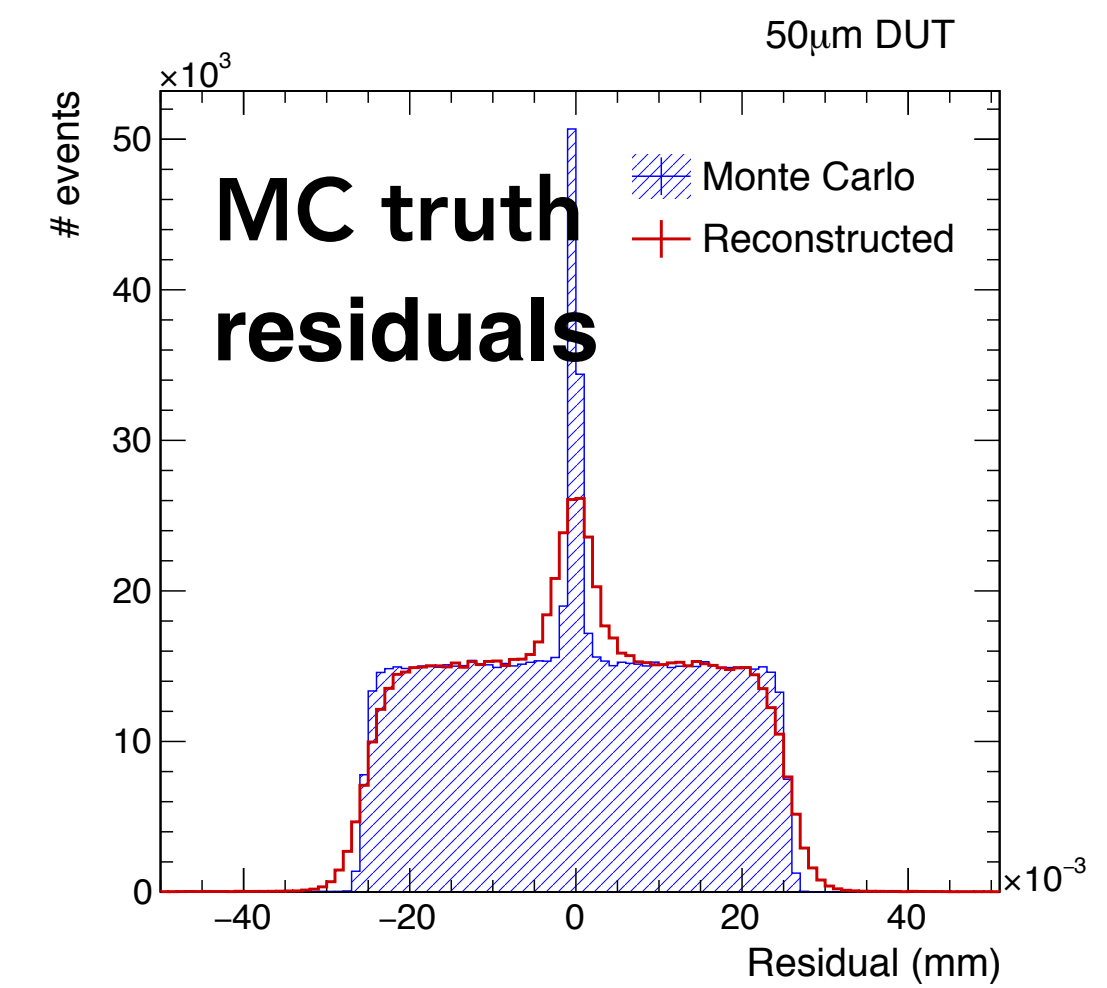
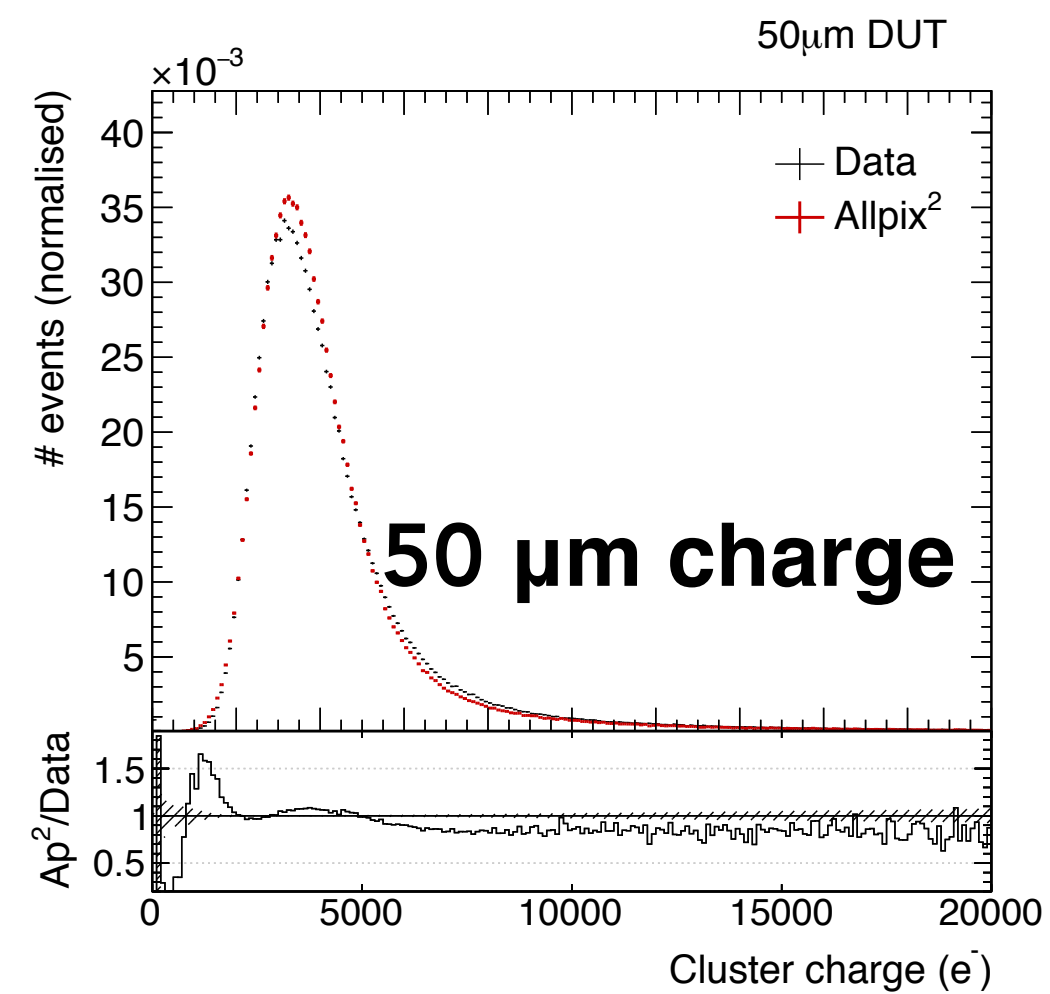
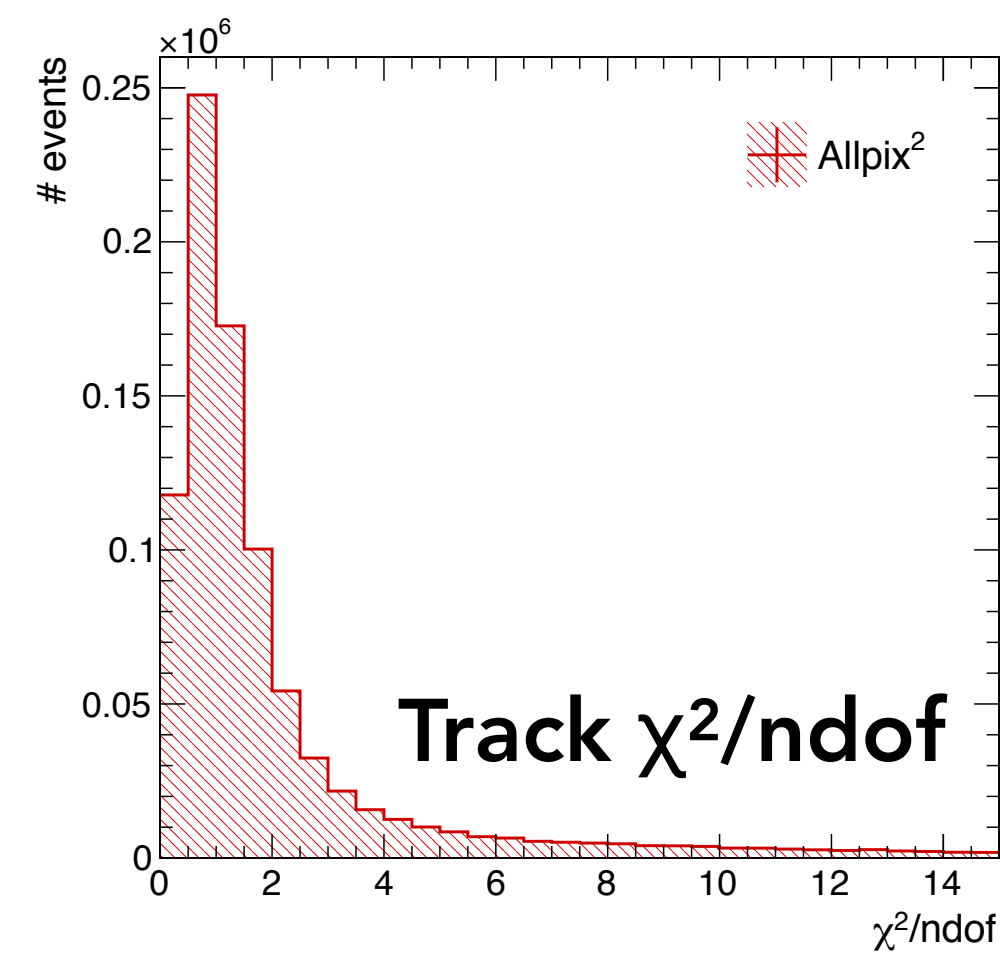


Allpix² structure

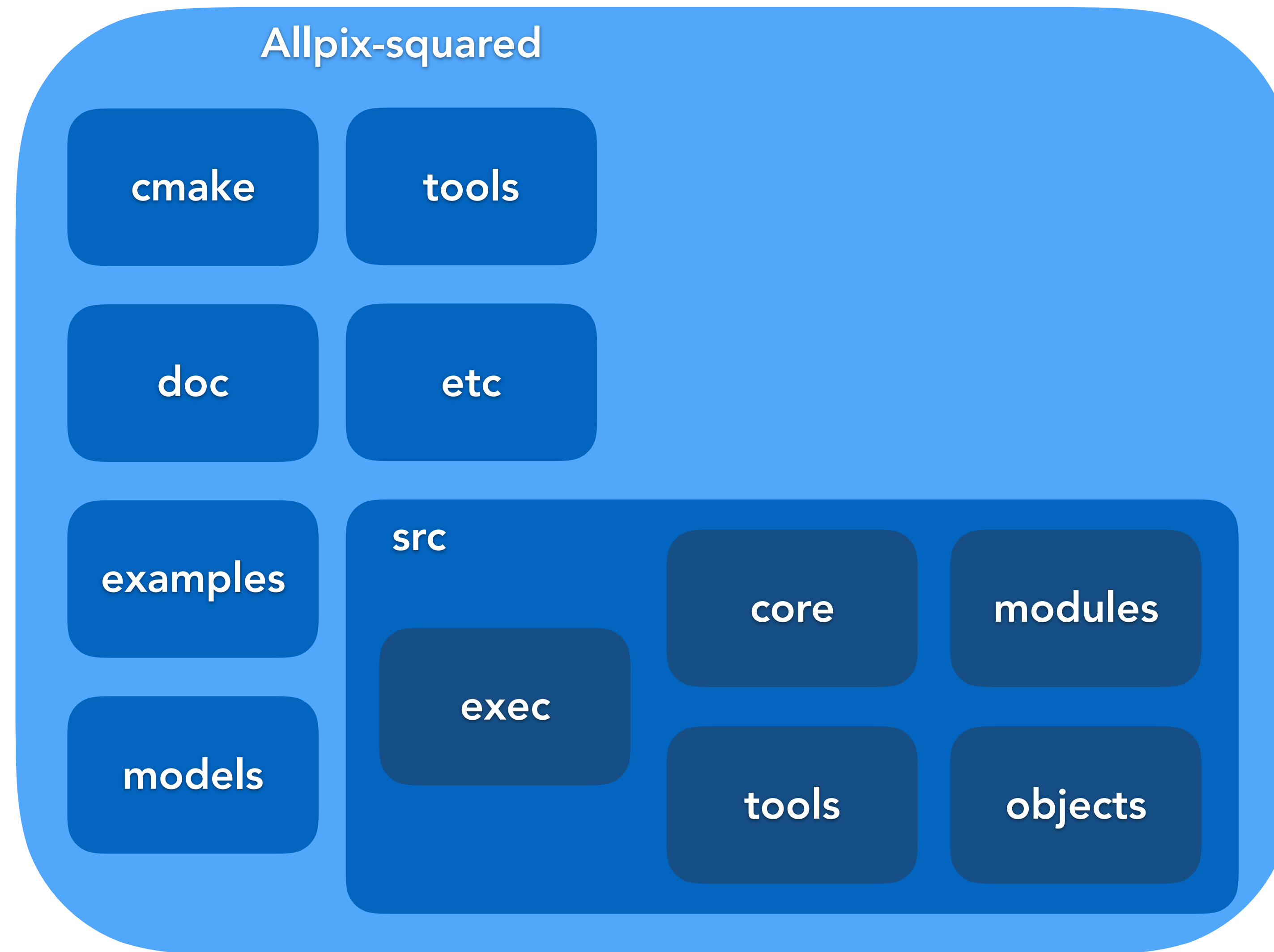
- Core of the software handles all of the main infrastructure
 - ❑ Creating instantiations of each module and checking that the program is configured properly
 - ❑ Interpreting the configuration files (TOML-style, human-readable) and passing these options to the modules
 - ❑ Passing information to and from modules
 - ❑ Logging module output
- Modules are the work-horses which handle all of the real detector simulation
 - ❑ Modular approach means modules are entirely independent
 - ❑ Defined input objects on which they act, and output objects which will be produced



Allpix² validation



Allpix² navigation

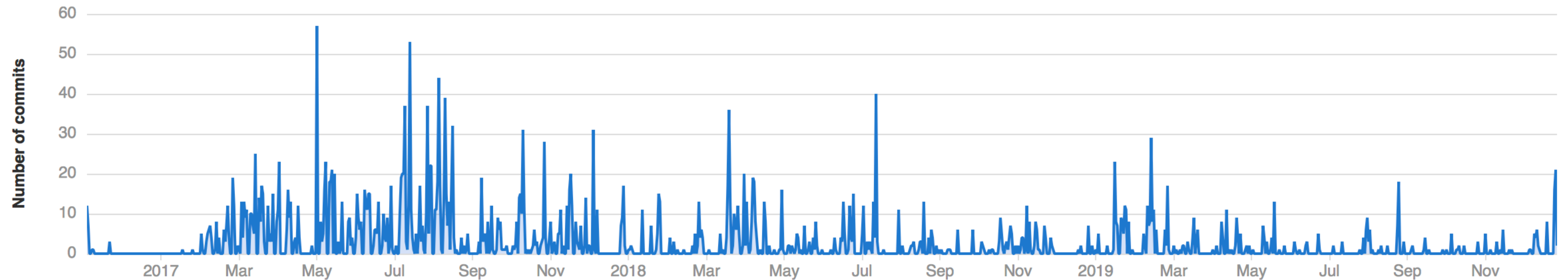


Allpix² timeline

- Allpix-Squared website went live July 20, 2017
- First stable release in September 2017
- Latest patch 1.4.3 from January 10 this year, version 2.0 in the coming months

Commits to master

Excluding merge commits. Limited to 6,000 commits.

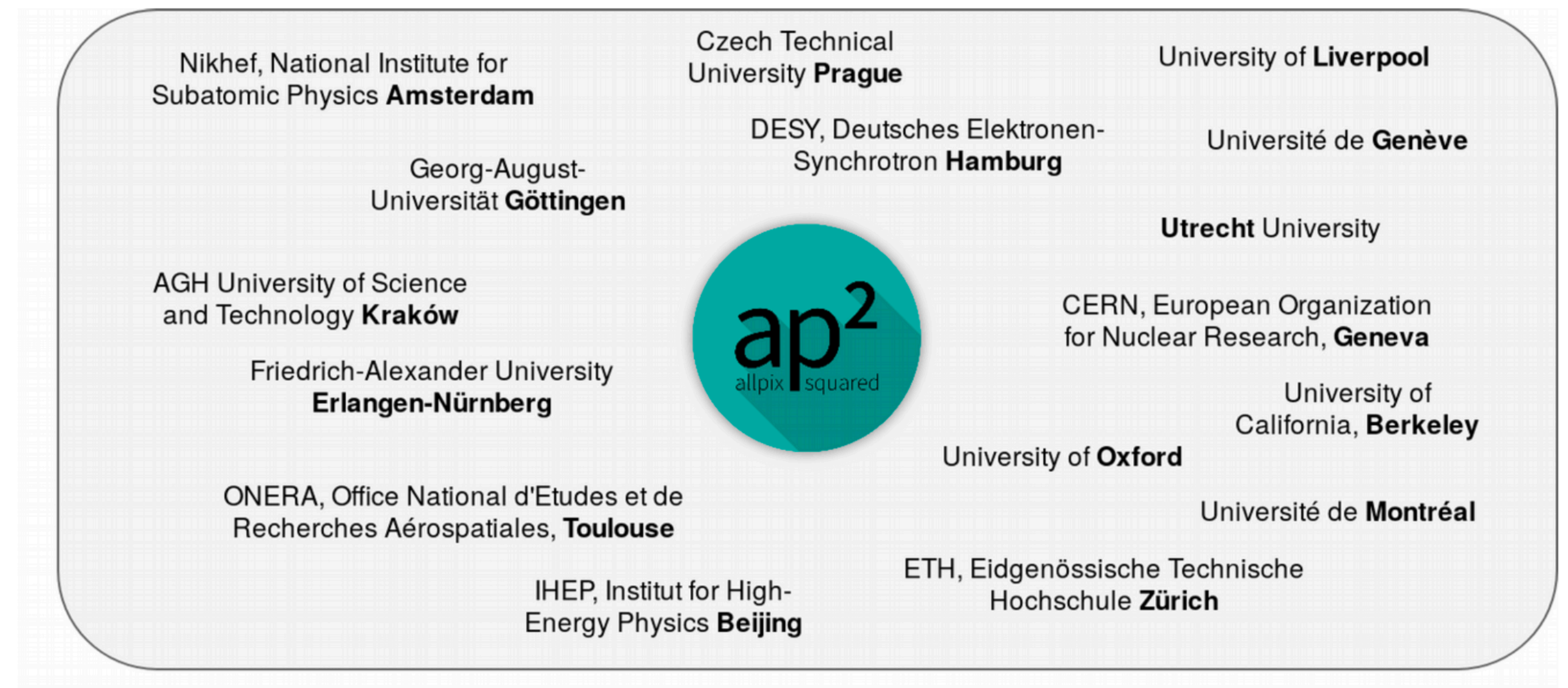


Allpix² developers

- > 20 members registered in project on gitlab
 - Varying from a few commits to several hundreds
 - A lot of effort provided by students
 - Technical student effort (K.Wolters) to write a lot of the original code
 - GSoC 2018 student (V.Sonesten) developing multi-event processing
 - GSoC 2019 student (V.Sonesten) completing this work
 - 2019-2020 master student (K. van den Brandt) extending passive materials and adding scintillator support
 - Rest of effort provided on a best-effort basis
 - Infrastructure maintained by S.Spannagel
 - Code review still draws on experience of K.Wolters
- **Andreas Nurnberg**
 - **Daniel Hynds**
 - **Dominik Dannheim**
 - **Edoardo Rossi**
 - **Joern Schwandt**
 - **Katharina Dort**
 - **Koen Wolters**
 - **Mateus Vicente**
 - **Mathieu Benoit**
 - **Matthew Daniel Buckland**
 - **Moritz Kiehn**
 - **Neal Gauvin**
 - **Niloufar Alipour Tehrani**
 - **Paul Schutze**
 - **Ruth Magdalena Munker**
 - **Salman Maqbool**
 - **Sebastien Murphy**
 - **Simon Spannagel**
 - **Thomas Billoud**
 - **Tobias Bisanz**
 - **Xin Shi**

Allpix² use cases

- Adoption of allpix-squared by many groups, covering a lot of applications not originally conceived of
 - ❑ Particle physics tracking detector R&D
 - ❑ Spin-off companies (neutron scanners, new detector types)
 - ❑ Space applications
 - ❑ Dosimetry
 - ❑ New sensor materials
 - ❑ Calorimetry
 - ❑ Photon science and imaging
 - ❑ ???



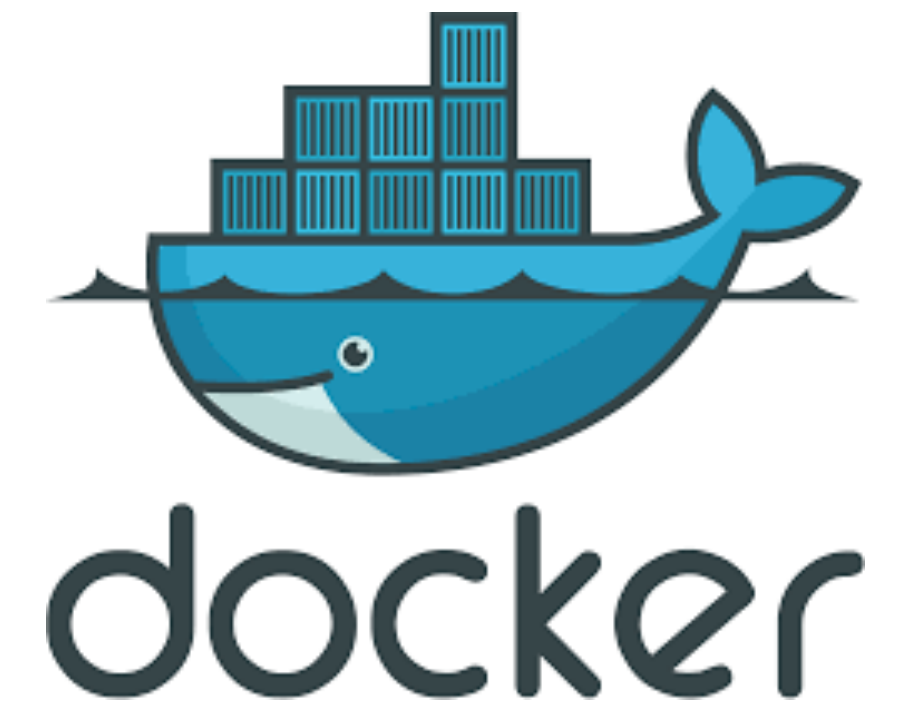
Approach

- 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 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

Installation options

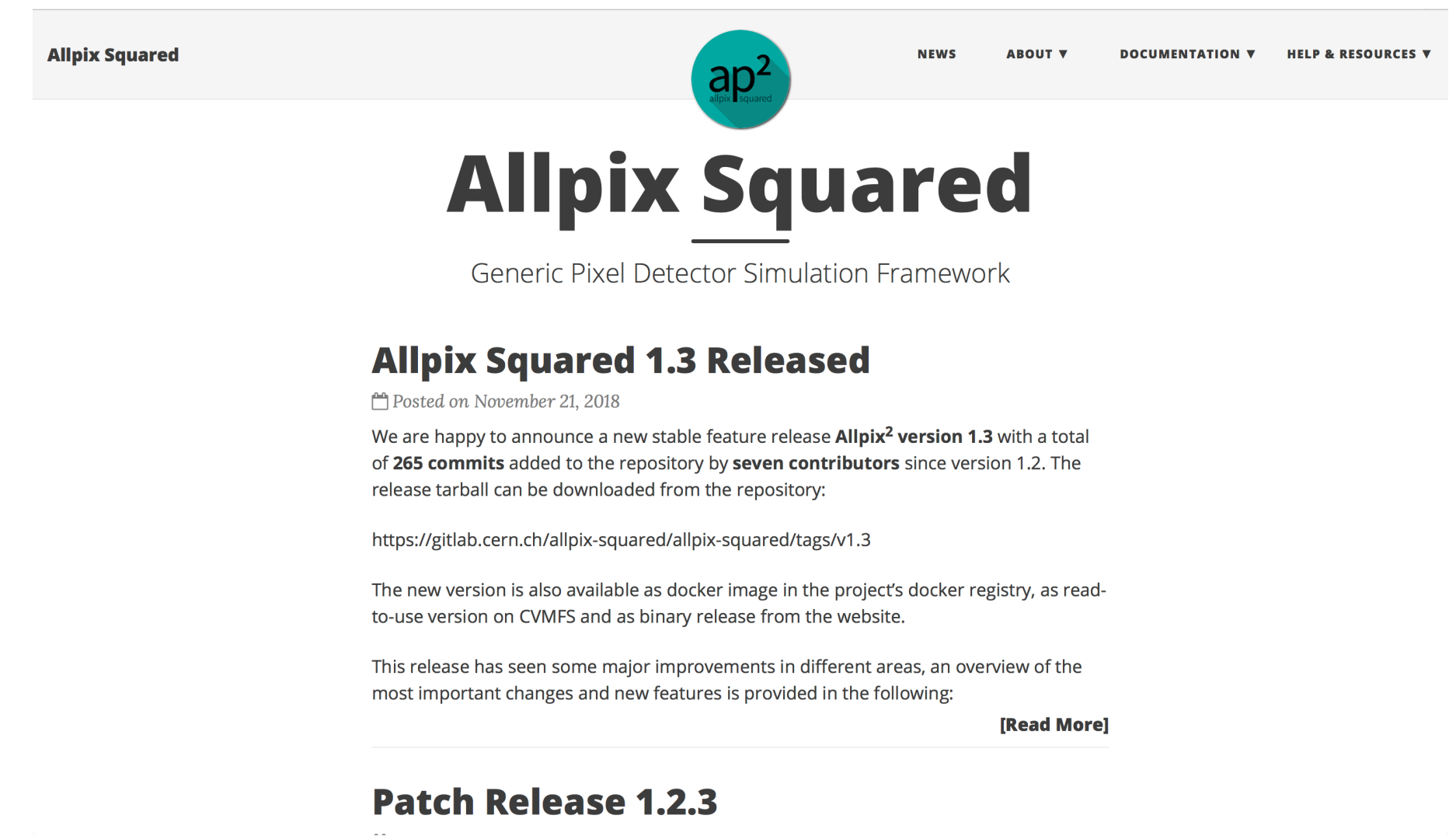
- There are many ways that allpix squared can be run, depending on what you want to do
 - Local checkout and installation on your laptop
 - Remote checkout on server with cvmfs access (lxplus)
 - Direct running on server with cvmfs access
 - Download and run a docker image
 - Download a binary tarball (slc6 and centos7 only)

- The most commonly used versions of this are to check out the software and compile it yourself - either locally or on a system with cvmfs access
 - This tutorial assumes that you are working on lxplus/a remote system with cvmfs access, and you will check out the code yourself
 - Checkout is with https access, use ssh if you have a gitlab account with ssh keys



Check out on lxplus

- A reminder, all resources are linked to from the project page:
 - <https://project-allpix-squared.web.cern.ch/project-allpix-squared/>
- 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
```

3rdparty	—————	Small external objects which can be used (eg. Fast iterators)
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
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

Modules

```
$ ls -l src/modules
```

```
CMakeLists.txt  
CapacitiveTransfer  
CorryvreckanWriter  
DefaultDigitizer  
DepositionGeant4  
DepositionPointCharge  
DetectorHistogrammer  
Dummy  
ElectricFieldReader  
GDMLOutputWriter  
GenericPropagation  
GeometryBuilderGeant4  
InducedTransfer  
LCIOWriter  
MagneticFieldReader  
ProjectionPropagation  
PulseTransfer
```

GeometryBuilderGeant4

Builds the geometry that will be used by Geant4

DepositionGeant4

Calls Geant4 to step particles through the geometry

GenericPropagation

Propagates charges deposited by Geant4 through the sensor

DefaultDigitiser

Describes the digitisation 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 8
$ 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 -l ../models/
```

```
CMakeLists.txt
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
```

```
type = "hybrid"

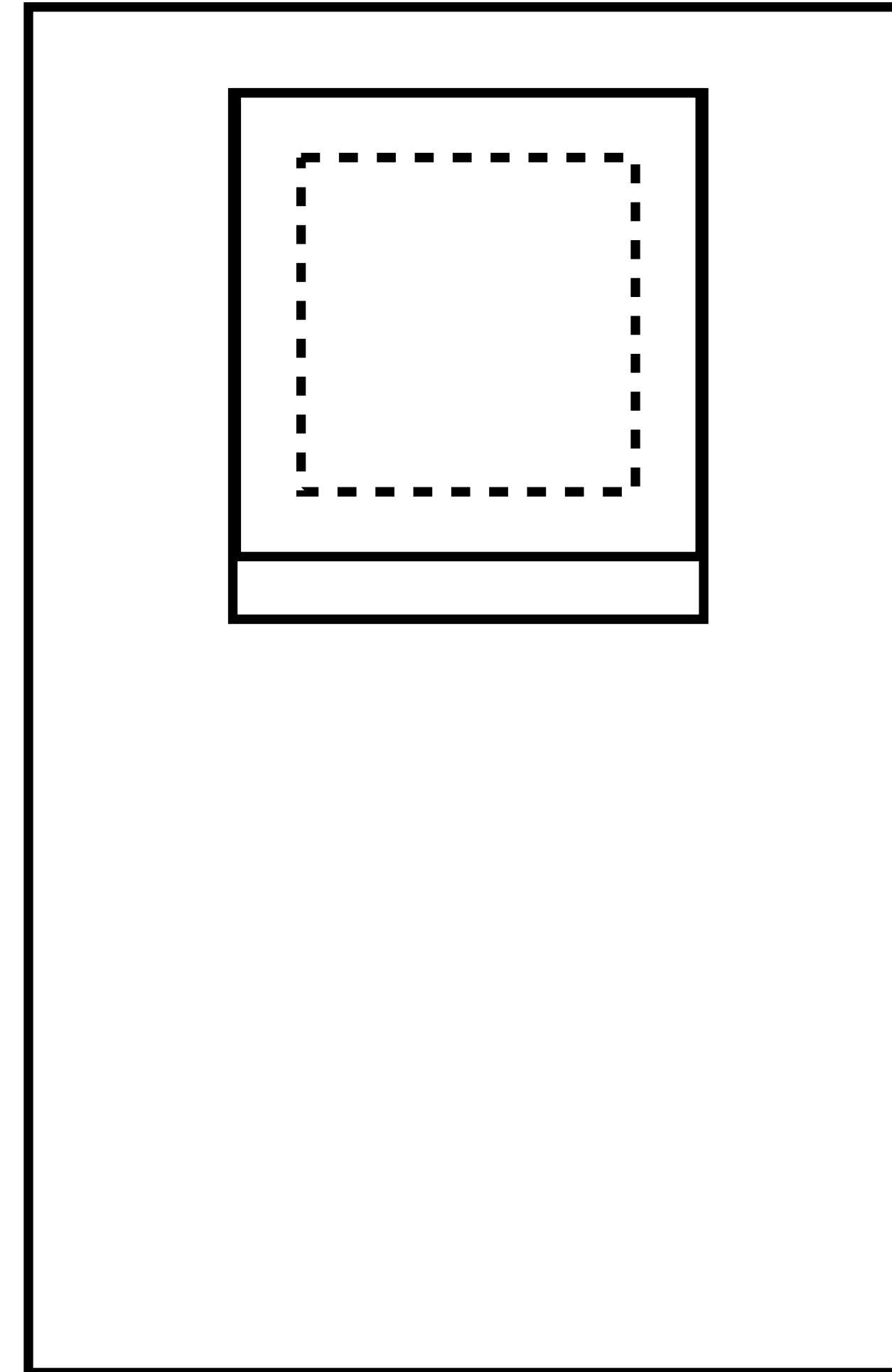
number_of_pixels = 256 256
pixel_size = 55um 55um

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
```



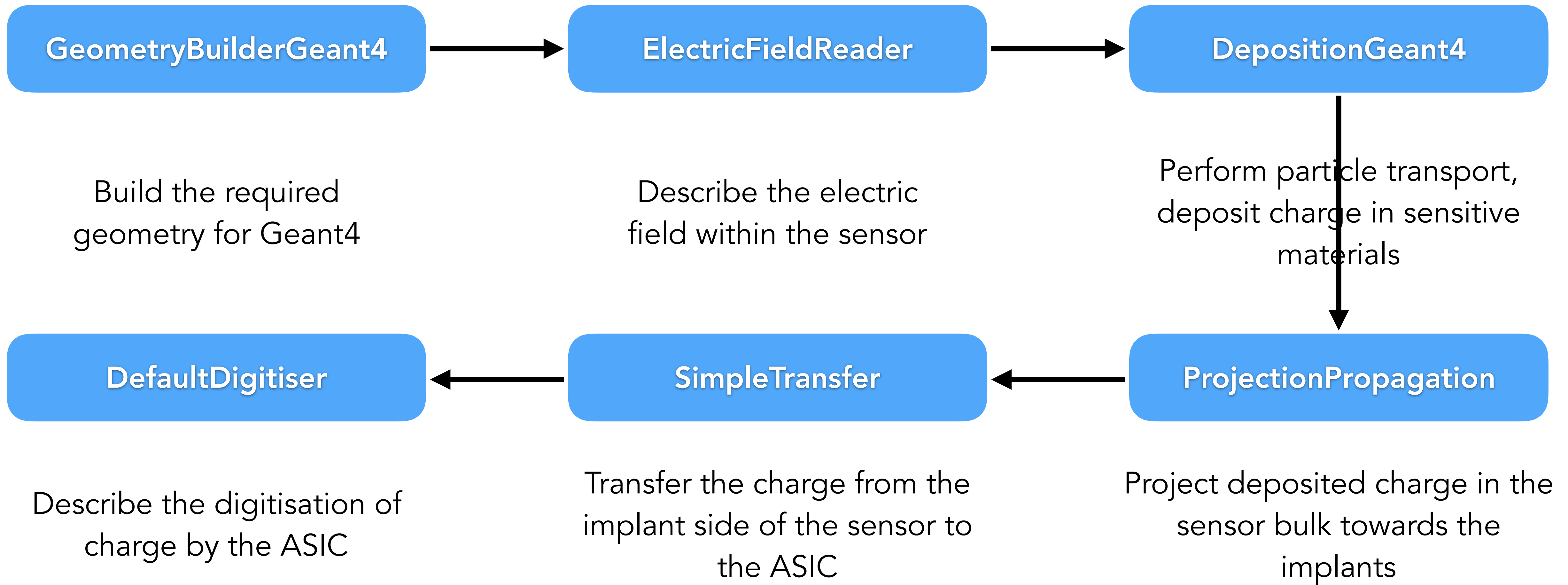
Adding algorithms

- 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



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]
```

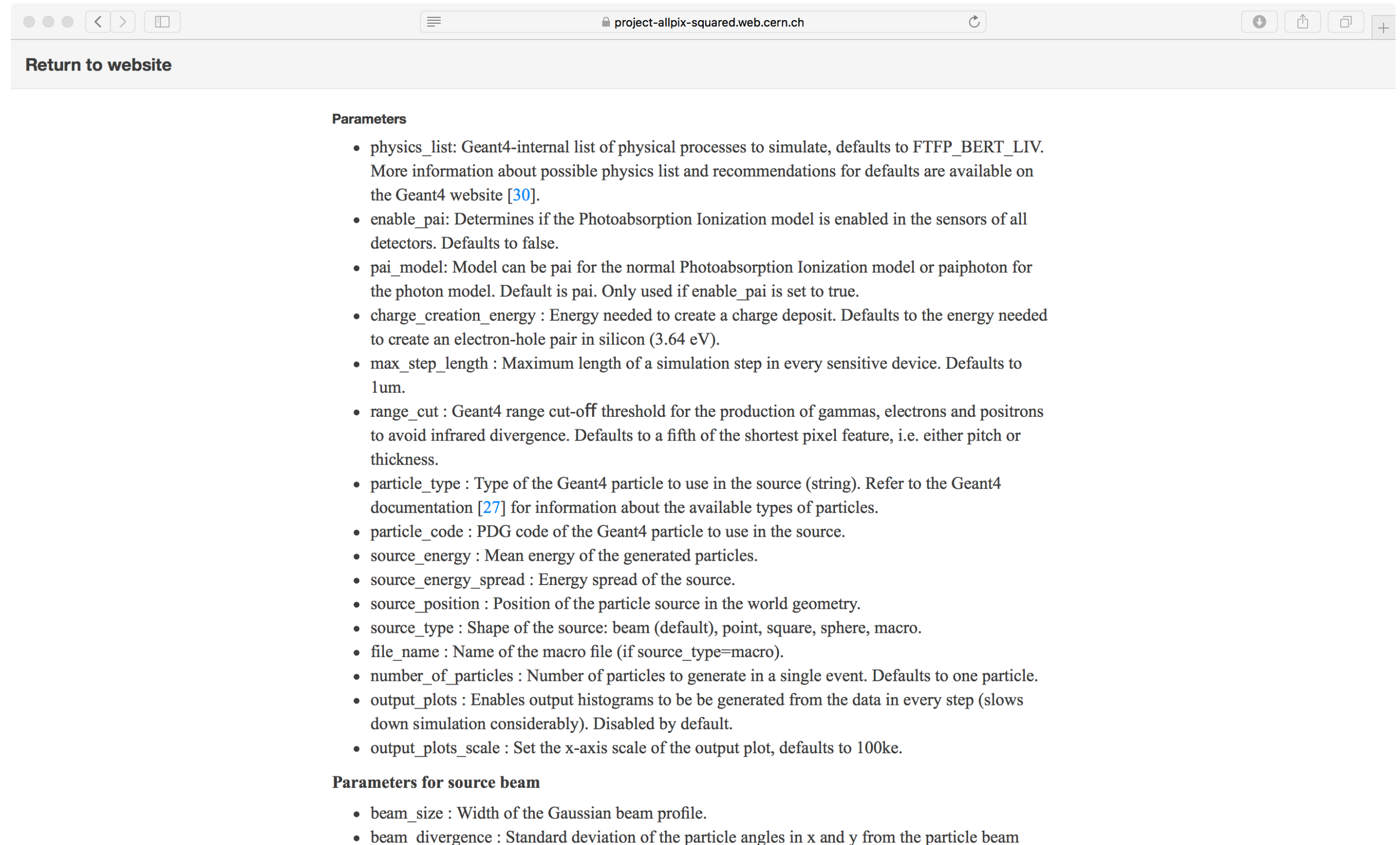
```
[ProjectionPropagation]
```

```
[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://project-allpix-squared.web.cern.ch/project-allpix-squared/usermanual/allpix-manualch7.html>



The screenshot shows a web browser window with the address bar displaying "project-allpix-squared.web.cern.ch". Below the address bar, there is a button labeled "Return to website". The main content area is titled "Parameters" and lists various simulation parameters with their default values and descriptions.

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 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

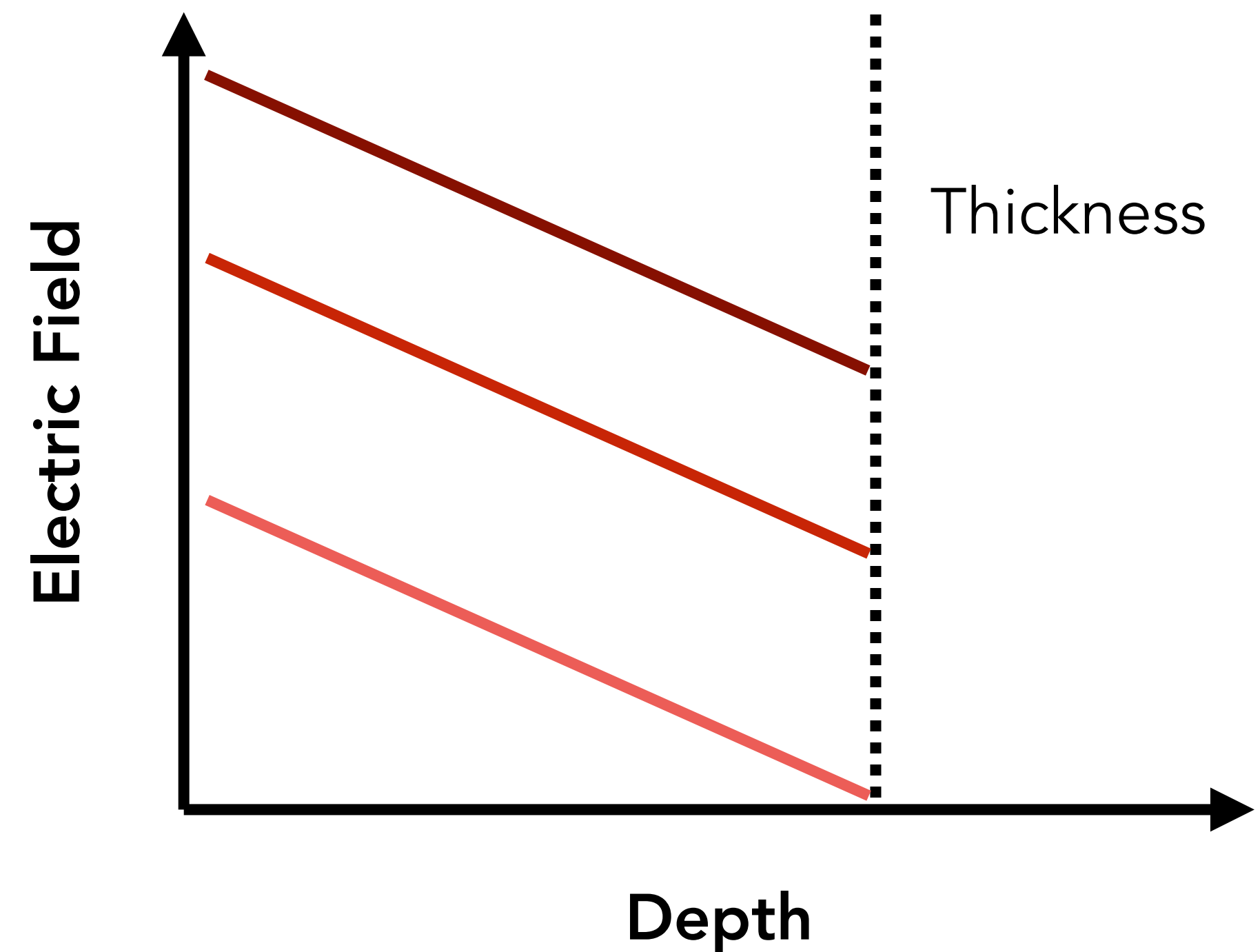
- DepositionGeant4 has several parameters, and is used as the source of particles in addition to interfacing geant4
 - Choose the type and energy of the particles that we want
 - Define the starting point and direction of the beam, in addition to the size of the beam
 - Pick a suitable physics list

```
[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 focussing effects around the implants
- A more complete field can be added by converting the output of FE simulations such as TCAD

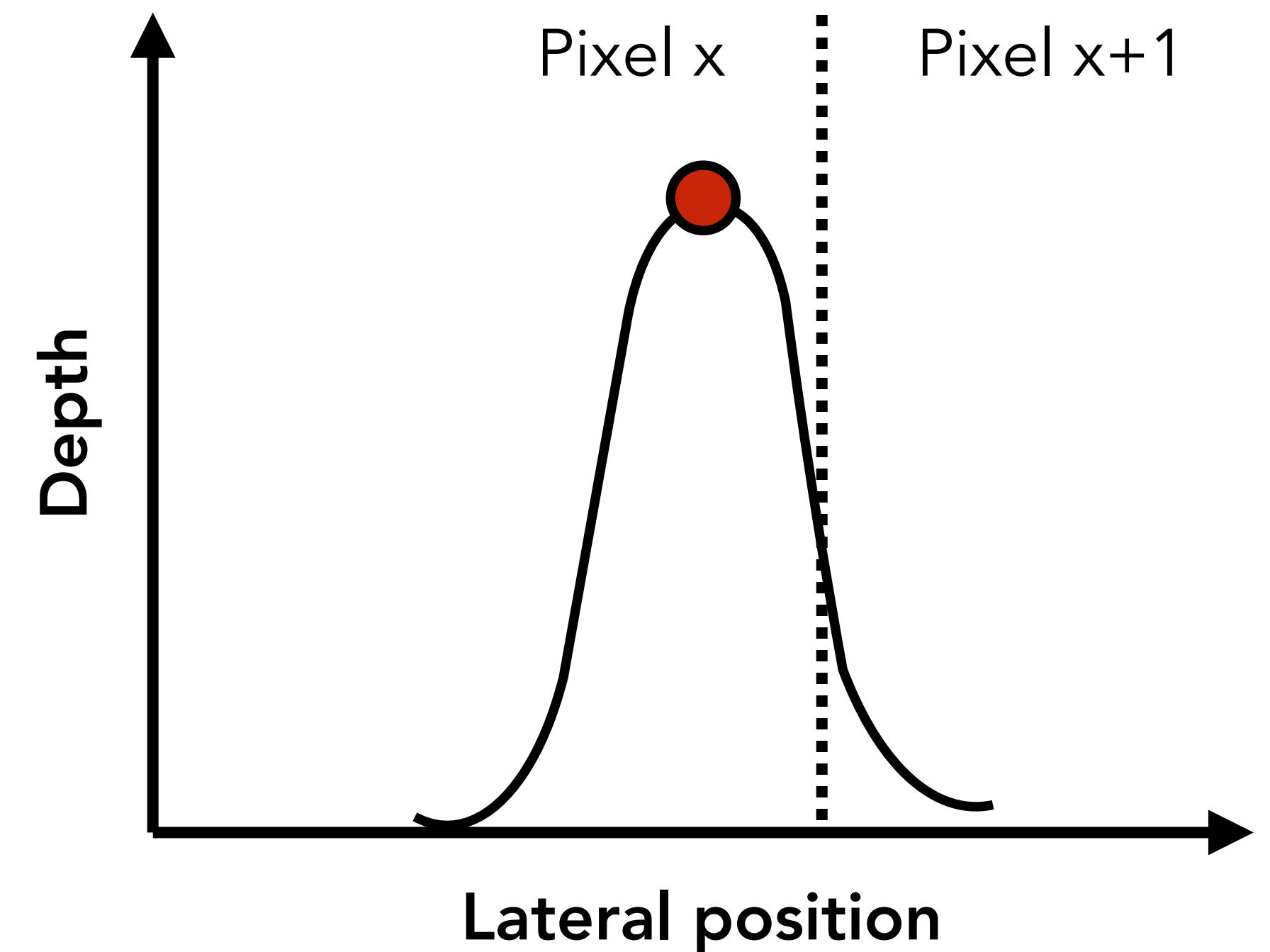
```
[ElectricFieldReader]  
model="linear"  
bias_voltage=-50V  
depletion_voltage=-30V
```



Propagation of deposited charges

[ProjectionPropagation]
temperature = 293K

- 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



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]

Digitisation

- 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 digitise 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

```
log_level = "Warning"
```

```
output_plots = 1
```

```
[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
```

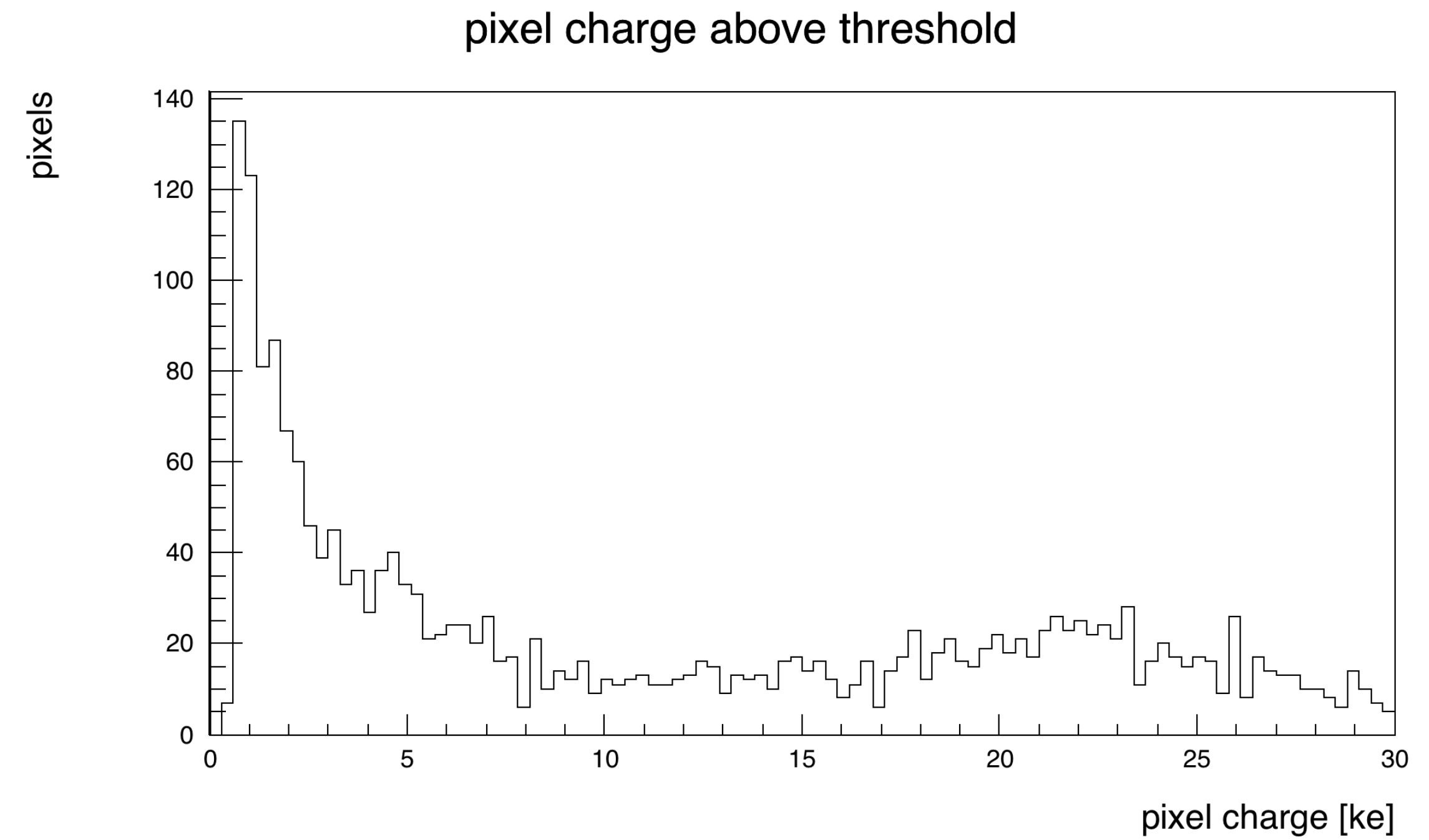
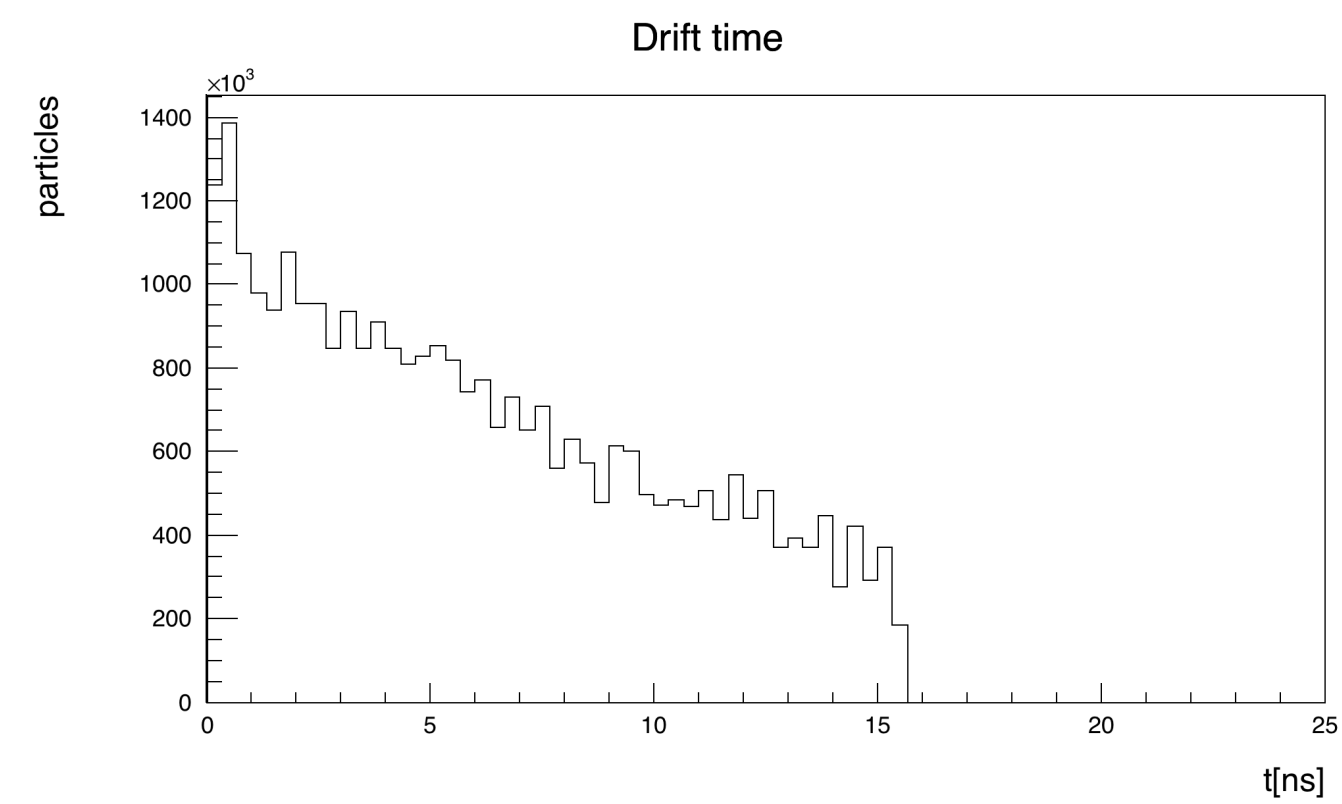
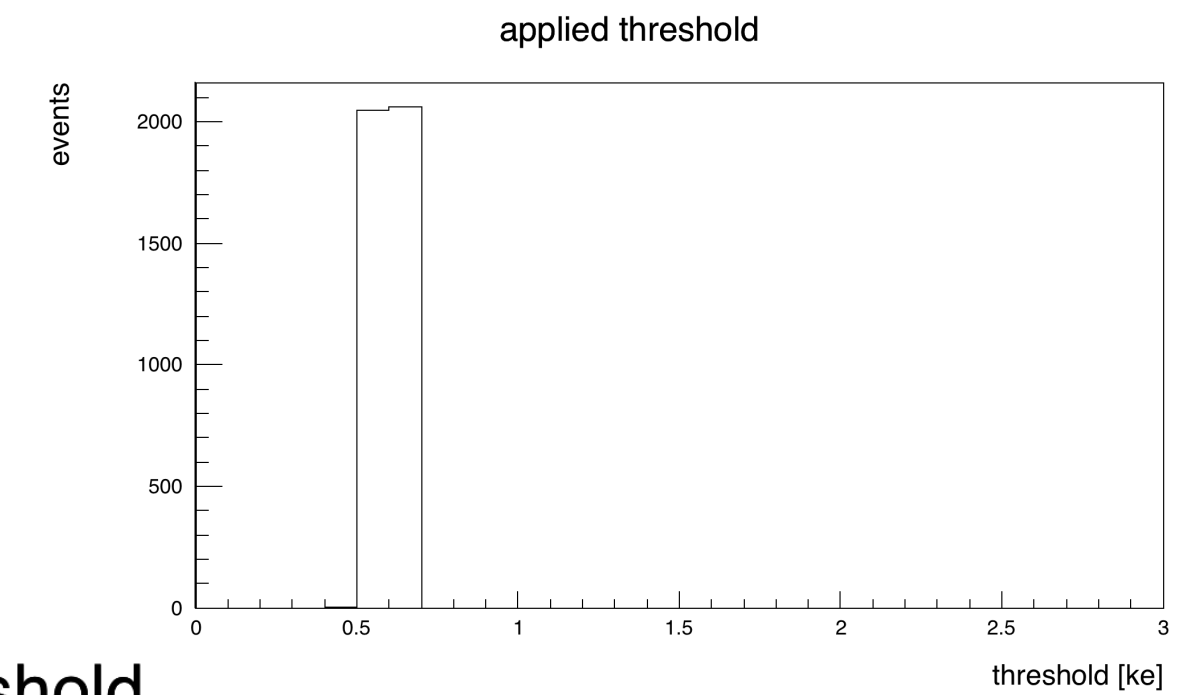
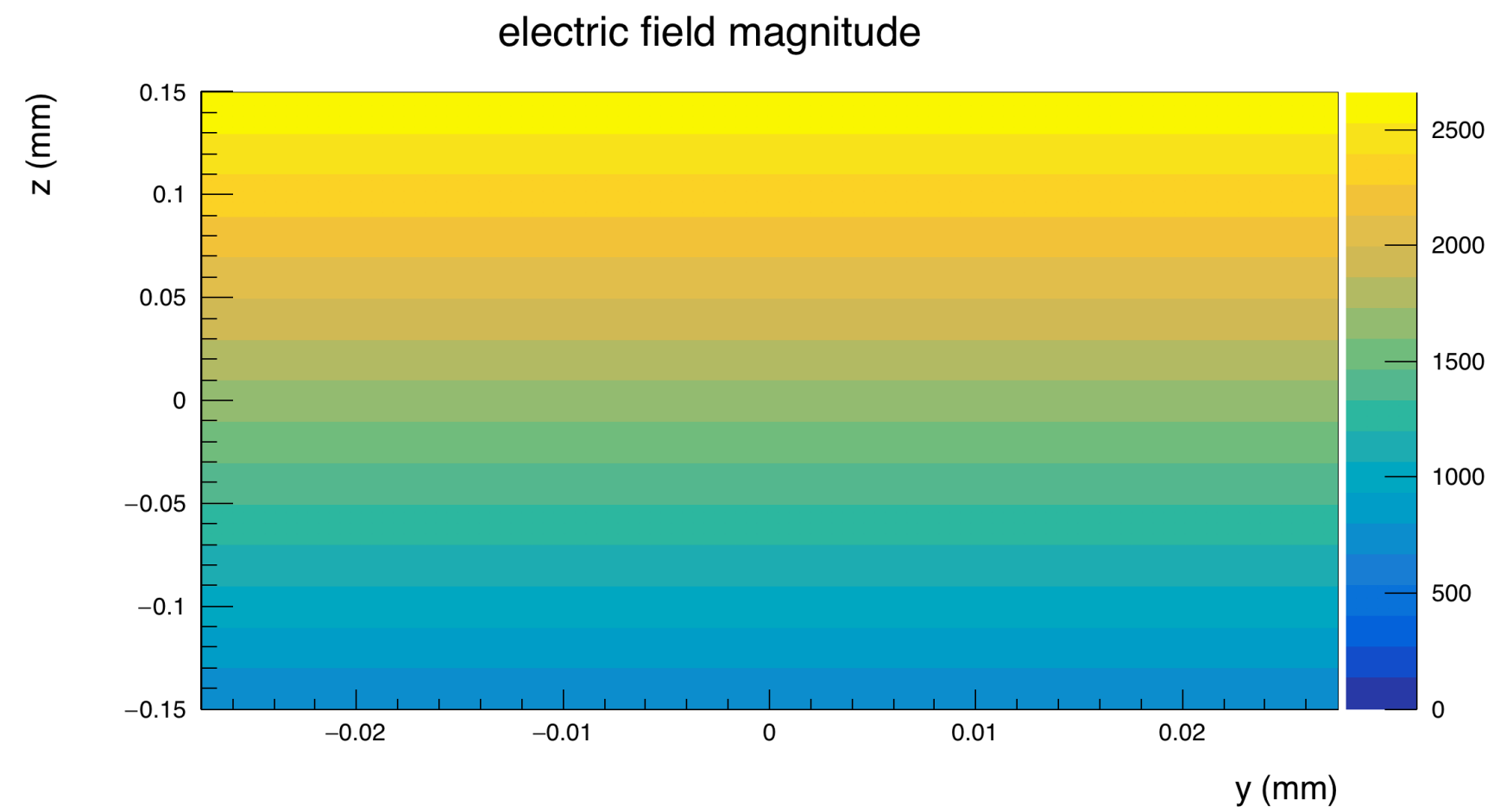
```
[ProjectionPropagation]  
temperature = 293K  
output_plots = 1
```

```
[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 visualise 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
 - Can show an example run from my laptop, where QT is installed

```
[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
```

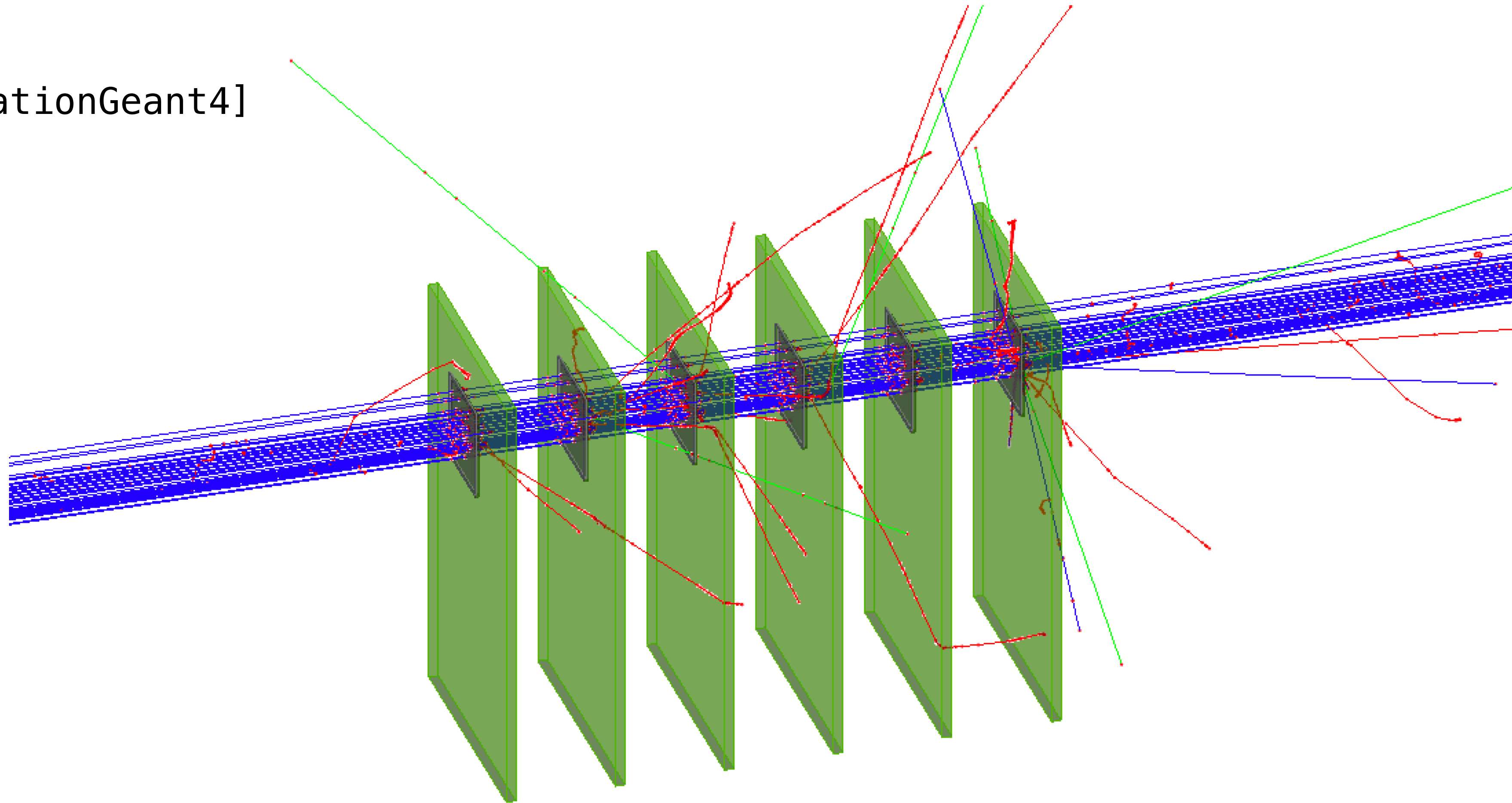
```
[detector4]
type = "timepix"
position = 0mm 0mm 60mm
orientation = 0 0 0
```

```
[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

[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 is that a separate instance of each module is created per detector
 - This allows some measure of multithreading to be used to improve simulation times - all detectors can be run in parallel
- This behaviour is controlled by the module type, either it is **unique** or **detector-specific**

A single detector chain

GeometryBuilderGeant4

DepositionGeant4

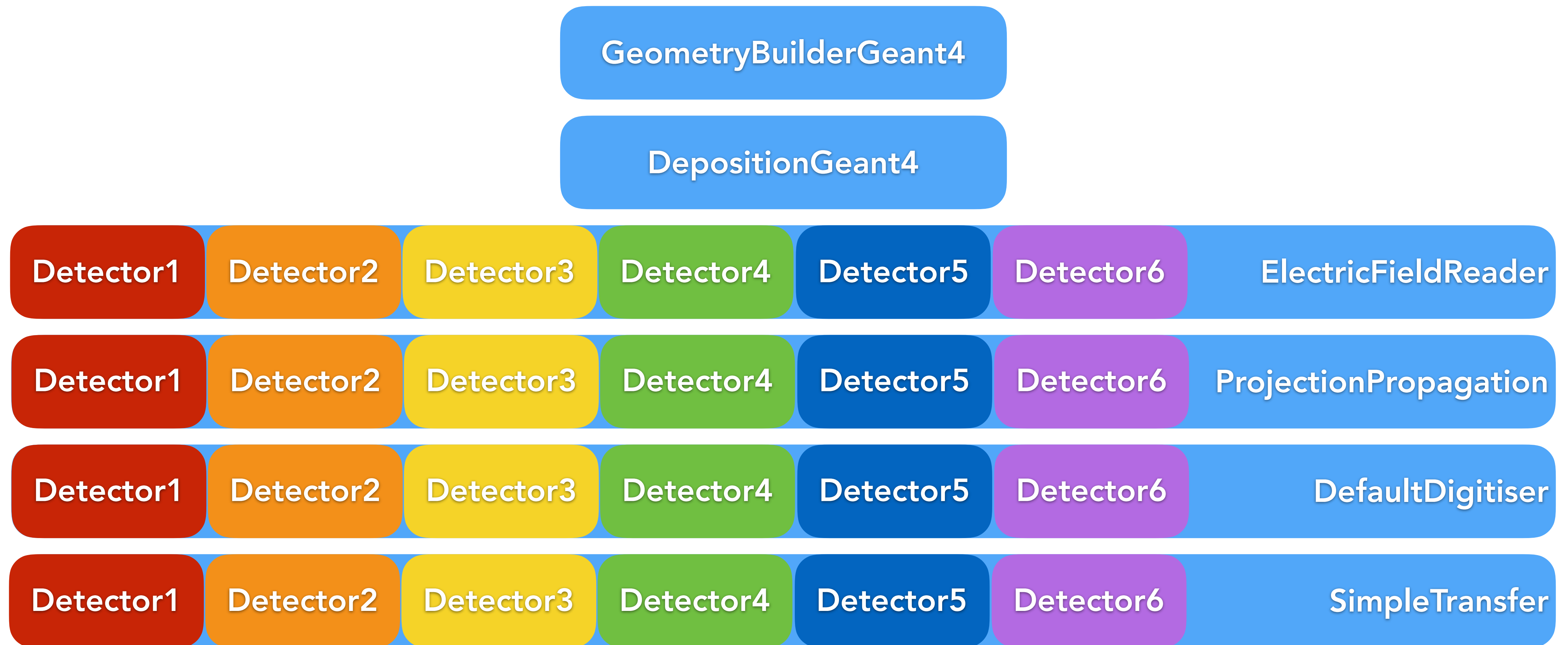
ElectricFieldReader

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 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 behaviour 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

Set 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 behaviour 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
output_plots = 1
```

```
[ElectricFieldReader]
model="linear"
name="detector1"
bias_voltage=-100V
depletion_voltage=-30V
output_plots = 1
```

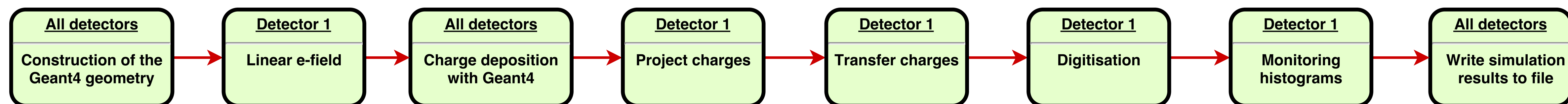
```
[ProjectionPropagation]
temperature = 293K
name = "detector2", "detector3", "detector4",
"detector5", "detector6"
output_plots = 1
```

```
[GenericPropagation]
name = "detector1"
temperature = 293K
output_plots = 1
```

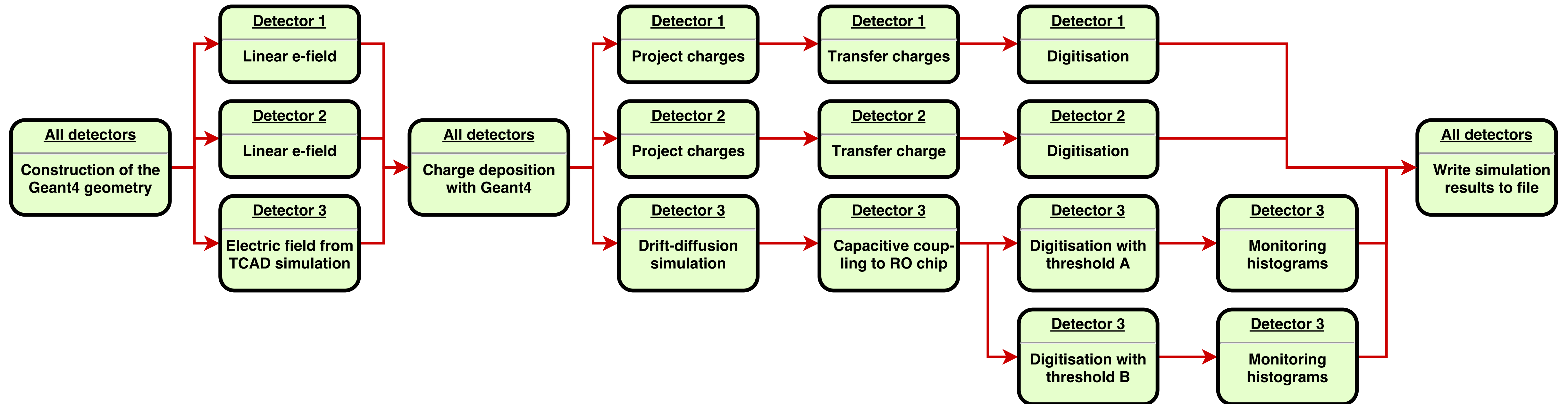
```
[SimpleTransfer]
output_plots = 1
```

```
[DefaultDigitizer]
output_plots = 1
```


Single detector chain



Multi-detector chain

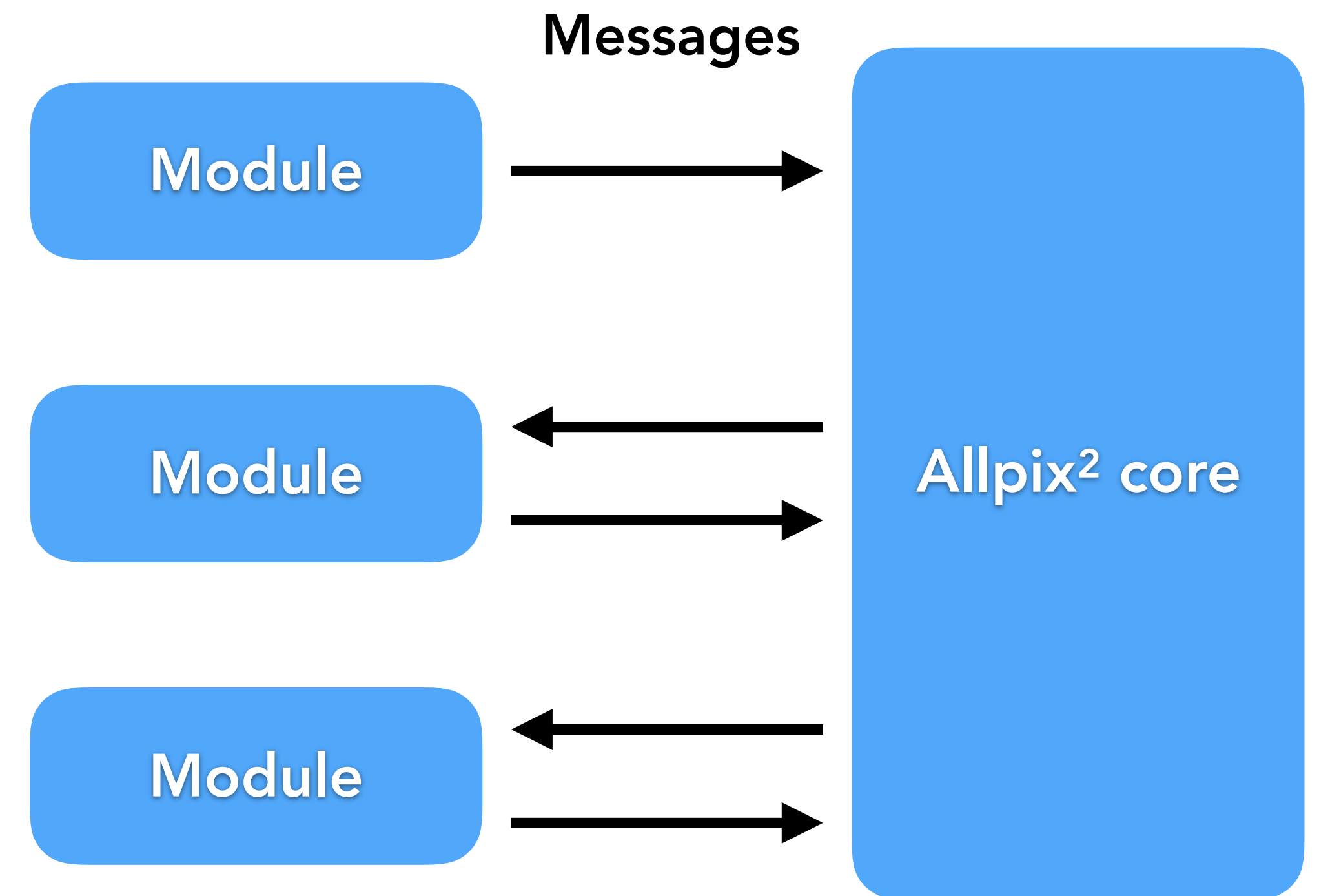


Making your own module

- 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
 - Check performed on start-up for configuration errors - check with 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 pick up
 - 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...
```

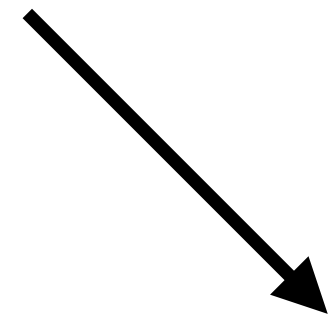
```
Name: TutorialExample  
Author: Daniel Hynds (daniel.hynds@cern.ch)  
Path:  
This module listens to "DepositedCharge" messages from one detector
```

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

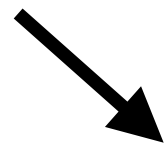
Writing your own module

```
1  /**
2   * @file
3   * @brief Implementation of [TutorialExample] module
4   * @copyright Copyright (c) 2017 CERN and the Allpix Squared authors.
5   * This software is distributed under the terms of the MIT License, copied verbatim in the file "LICENSE.md".
6   * In applying this license, CERN does not waive the privileges and immunities granted to it by virtue of its status as an
7   * Intergovernmental Organization or submit itself to any jurisdiction.
8   */
9
10 #include "TutorialExampleModule.hpp"
11
12 #include <string>
13 #include <utility>
14
15 #include "core/utils/log.h"
16
17 using namespace allpix;
18
19 TutorialExampleModule::TutorialExampleModule(Configuration& config, Messenger* messenger, std::shared_ptr<Detector> detector)
20     : Module(config, detector), detector_(detector), messenger_(messenger) {
21
22     // ... Implement ... (Typically bounds the required messages and optionally sets configuration defaults)
23     // Input required by this module
24     messenger_>bindSingle(this, &TutorialExampleModule::message_, MsgFlags::REQUIRED);
25 }
26
27 void TutorialExampleModule::init() {
28     // Get the detector name
29     std::string detectorName = detector_>getName();
30     LOG(DEBUG) << "Detector with name " << detectorName;
31 }
32
33 void TutorialExampleModule::run(unsigned int) {
34     // ... Implement ... (Typically uses the configuration to execute function and outputs an message)
35     std::string detectorName = message_>getDetector()>getName();
36     LOG(DEBUG) << "Picked up " << message_>getData().size() << " objects from detector " << detectorName;
37 }
```

Initialise variables/
histograms etc.



Main code, executed
each event



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
```

...

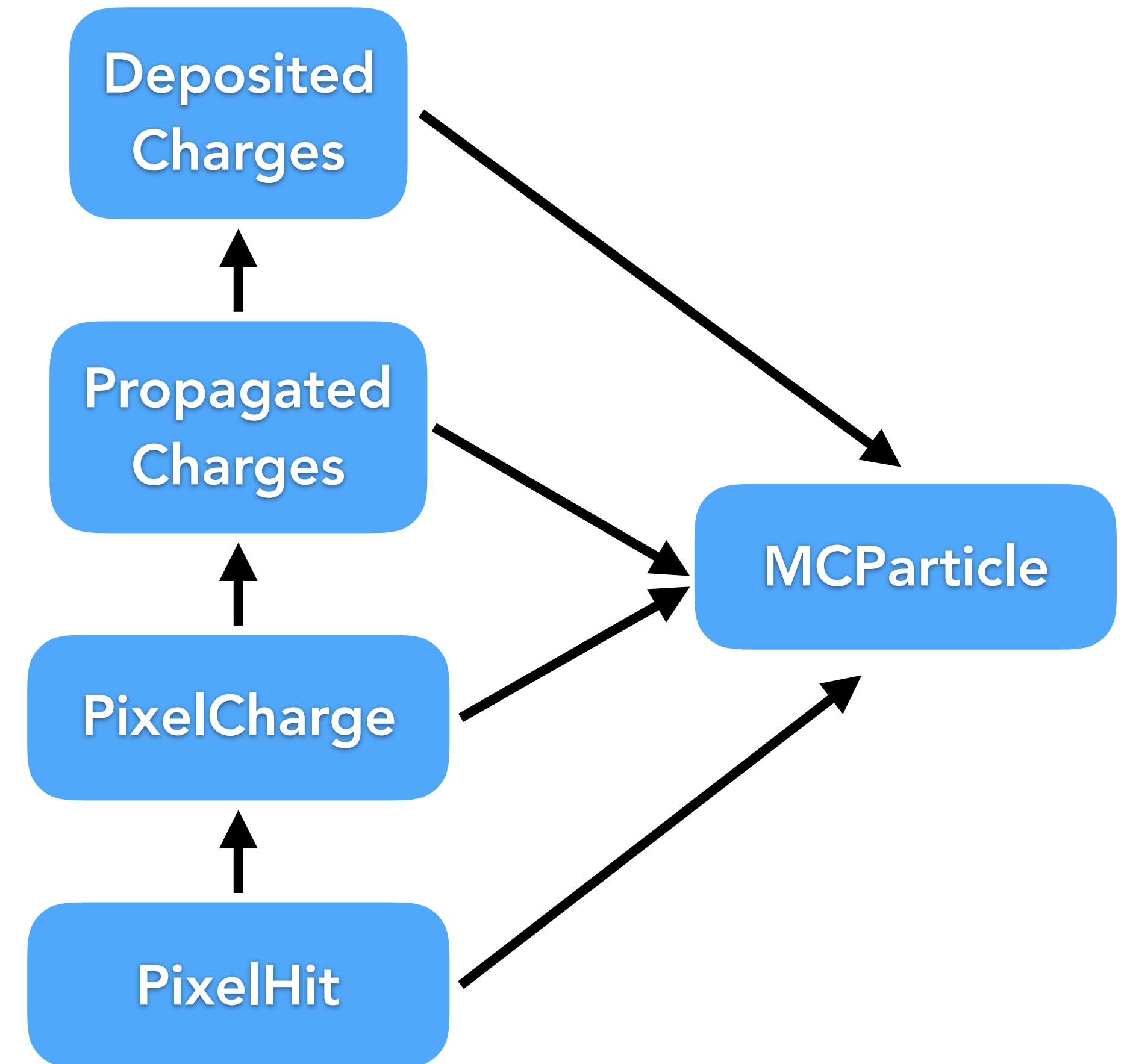
[TutorialExample]

...

A few other features - MC history

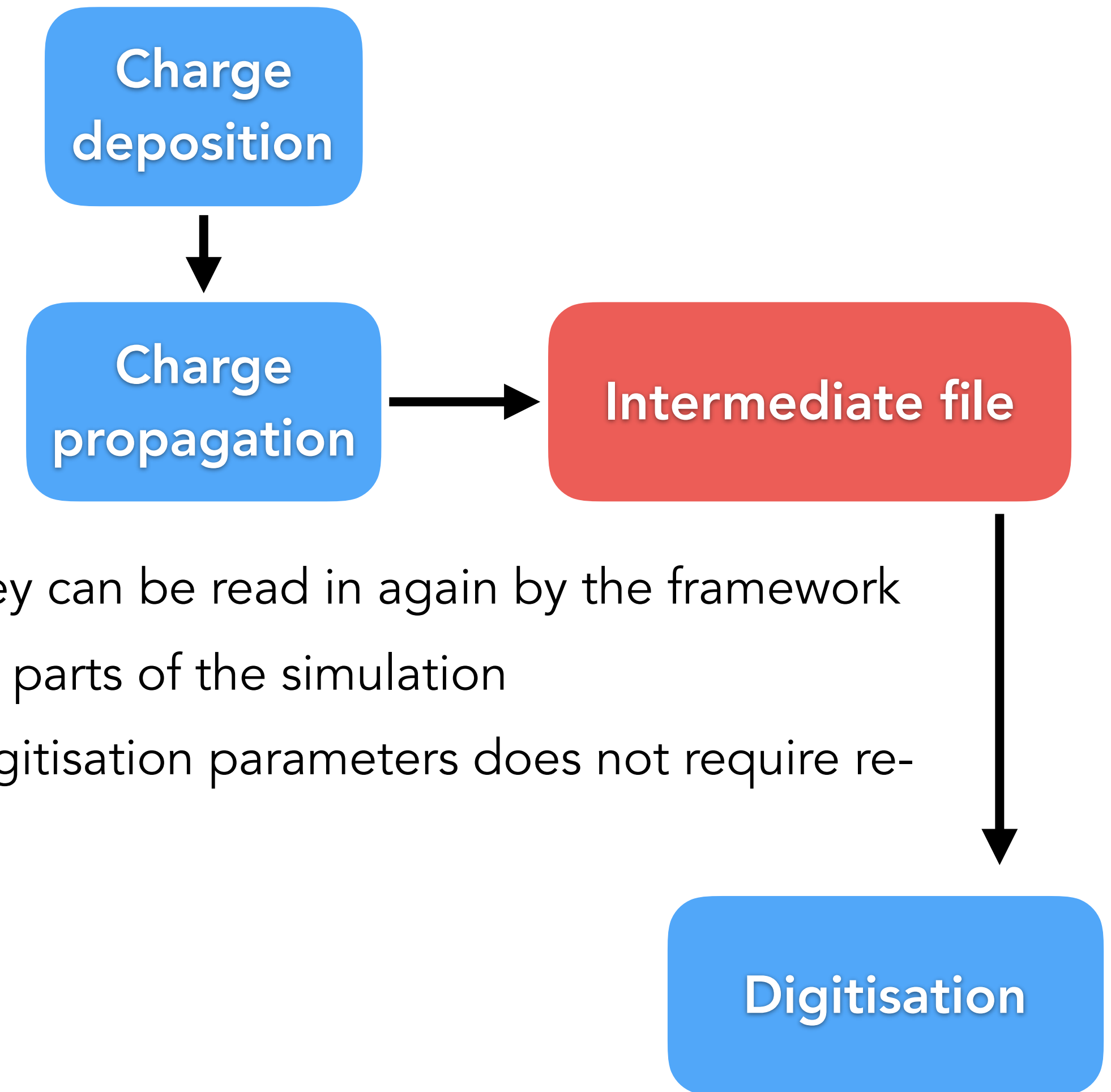
- All objects contain information about where they come from
 - Direct link to the preceding object
 - All objects link back to original MC particle
- Messages templated in the code, so adding a new object is straightforward
 - Define the object, must inherit from Object
 - Add a definition for the message

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



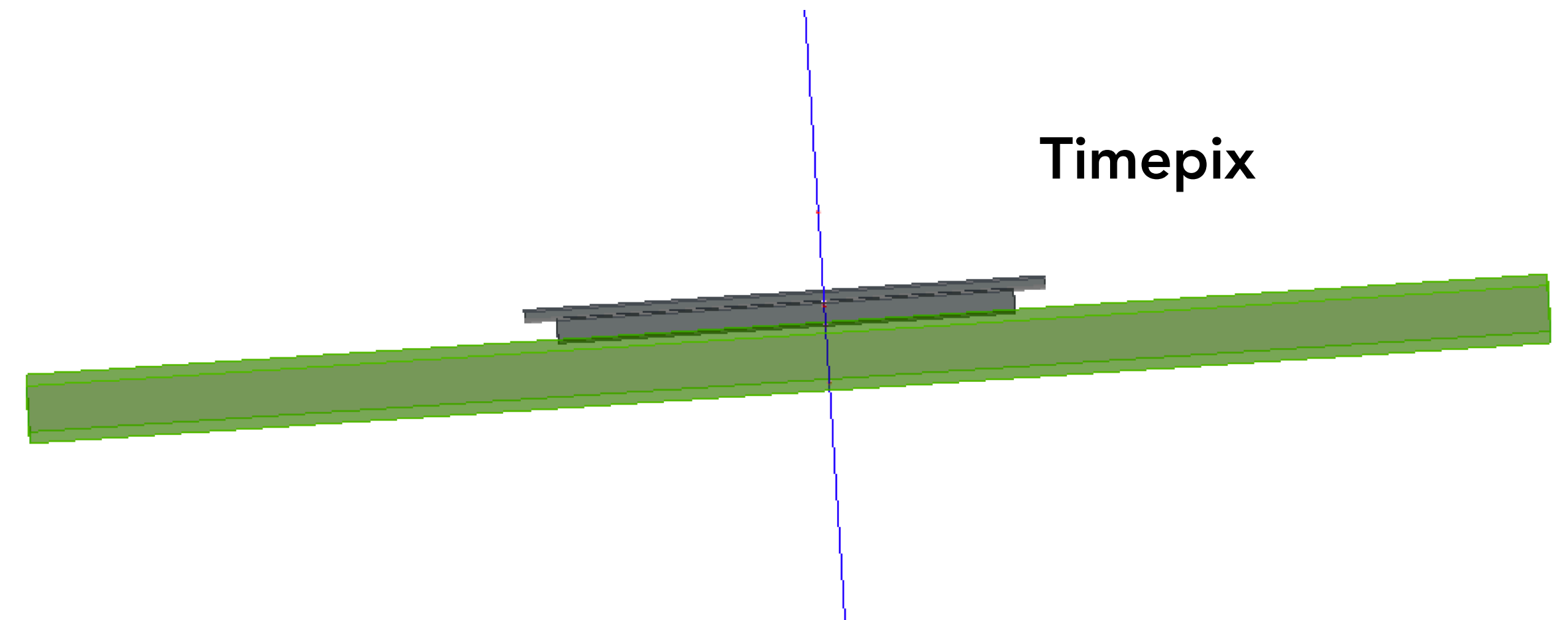
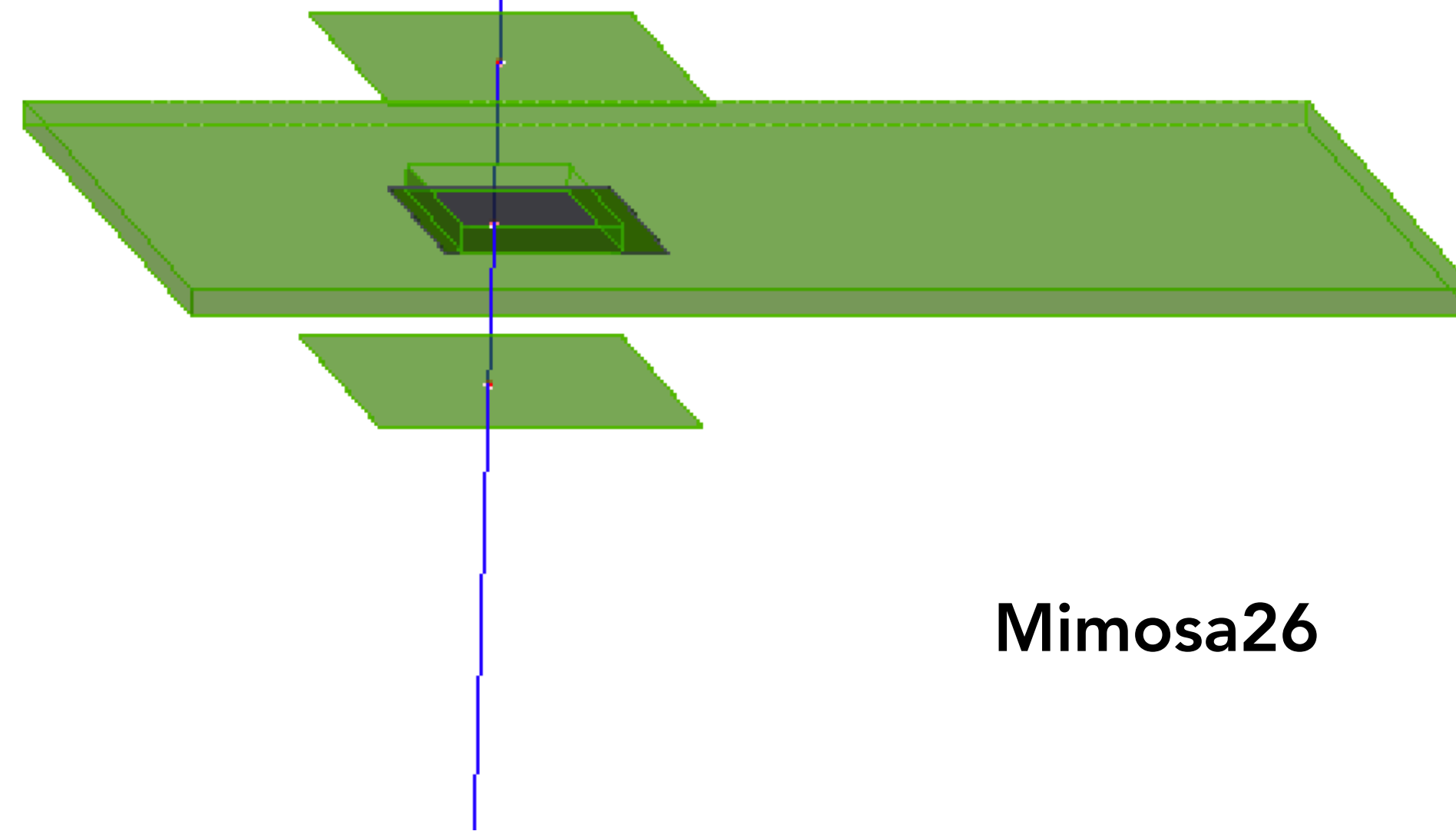
A few other features - output writing

- 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
 - eg. 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.



Summary

- Simple to get started with allpix-squared and set up a simulation for multiple detectors and varying parameters
 - Installations on cvfms, easy to set up local installations for development work

- First point of reference is typically the user manual
 - <https://project-allpix-squared.web.cern.ch/project-allpix-squared/usermanual/allpix-manual.html>

- Support available via email on the dedicated mailing list
 - allpix-squared-users@cern.ch