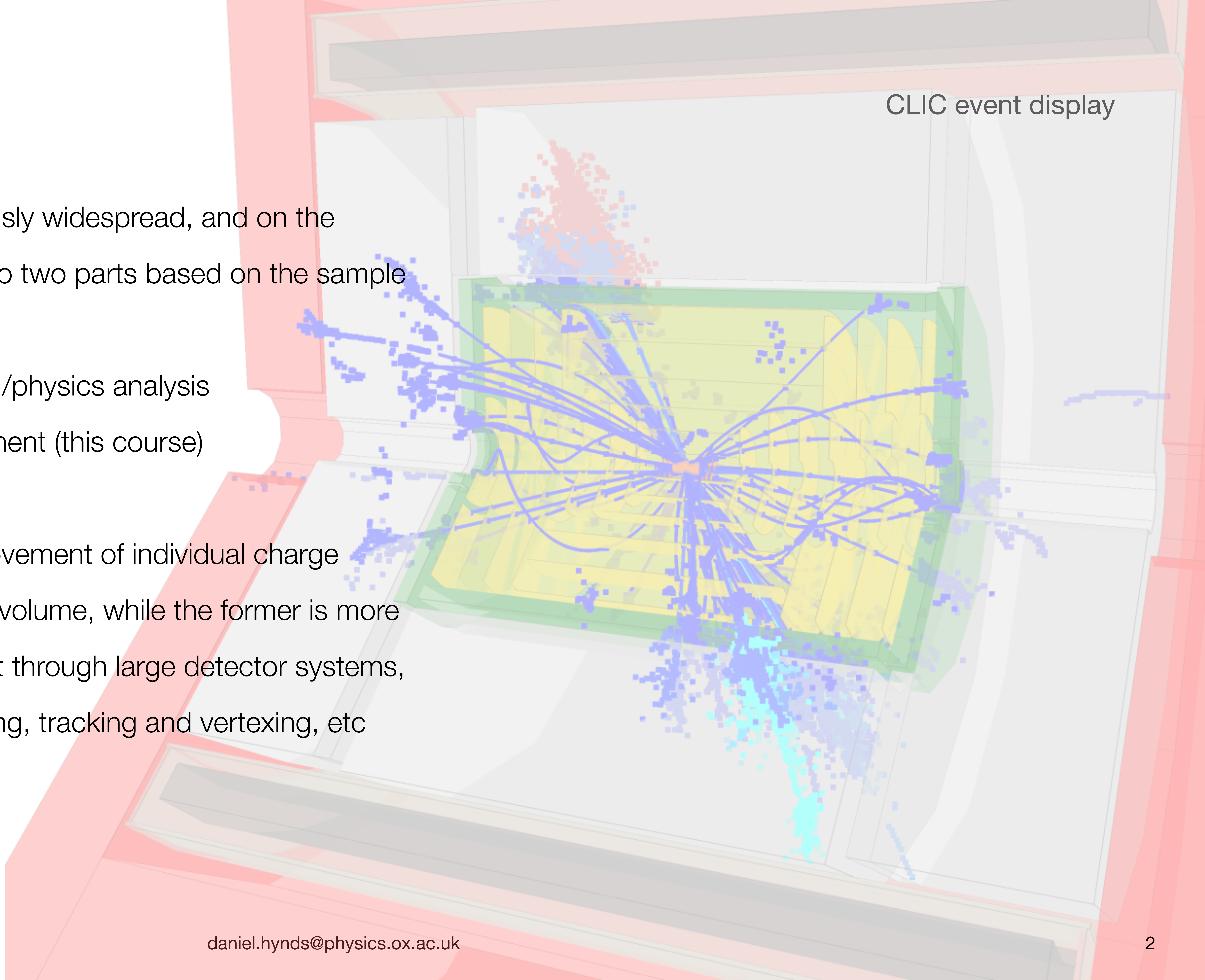
UNIVERSITY OF OXFORD

Monte Carlo simulations - Allpix^2

Daniel Hynds

Monte Carlo for detector physics



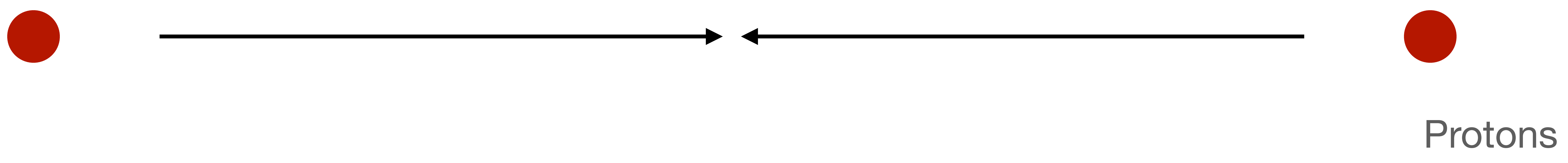
CLIC event display

Monte Carlo methods are obviously widespread, and on the detector side are roughly split into two parts based on the sample size and level of detail required:

- MC for experiment design/physics analysis
- MC for detector development (this course)

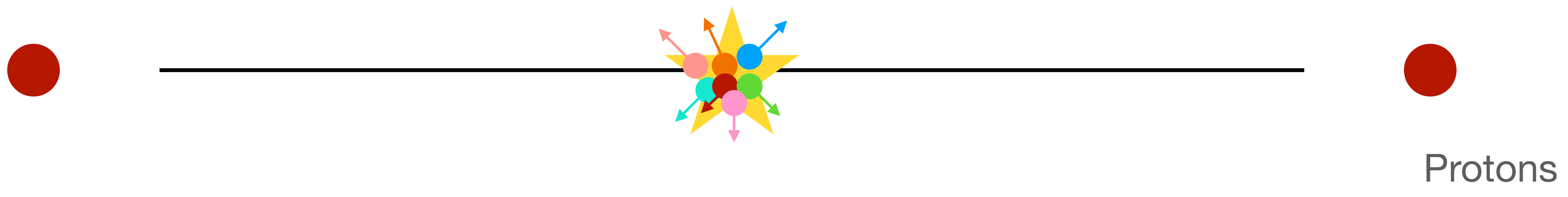
The latter typically covers the movement of individual charge carriers throughout the sensitive volume, while the former is more concerned with particle transport through large detector systems, covering layout, multiple scattering, tracking and vertexing, etc

Experiment-scale generation

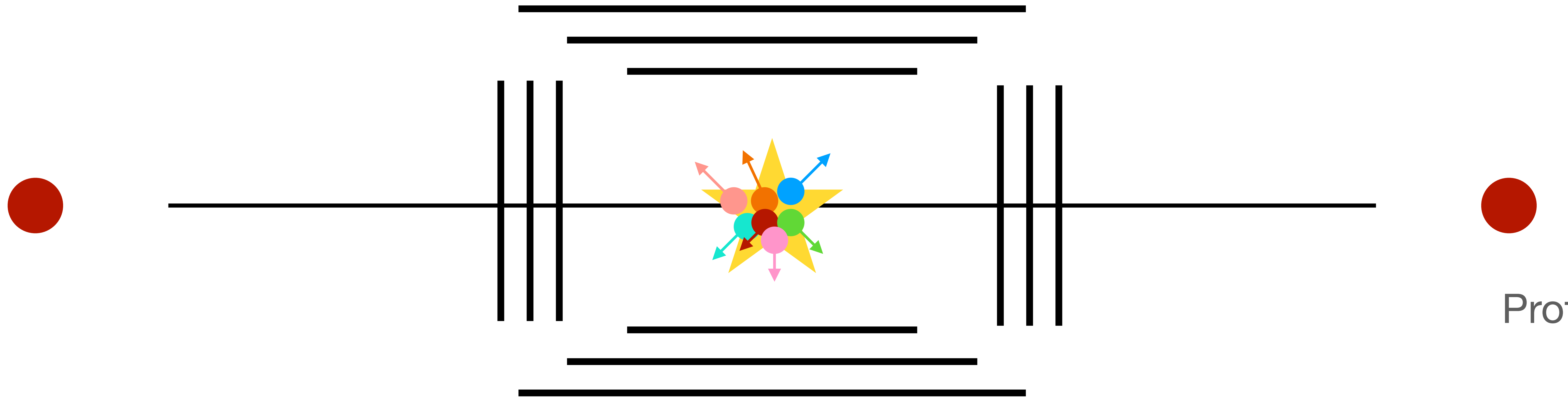


daniel.hynds@physics.ox.ac.uk

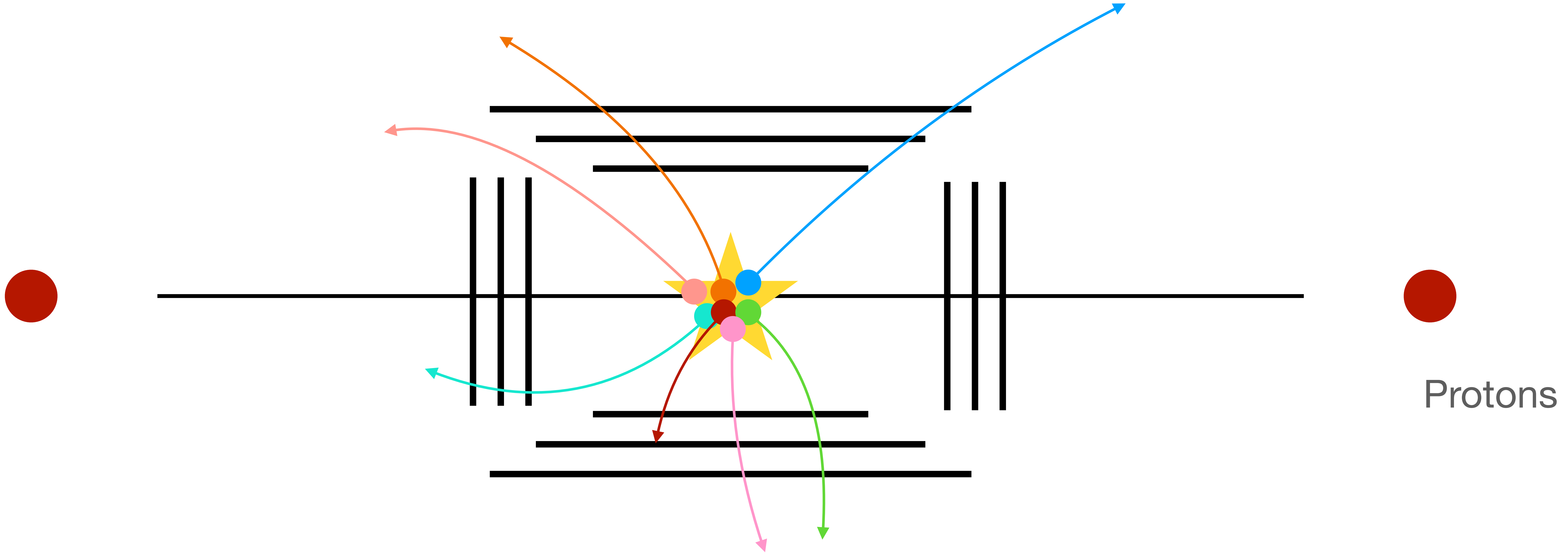
Experiment-scale generation



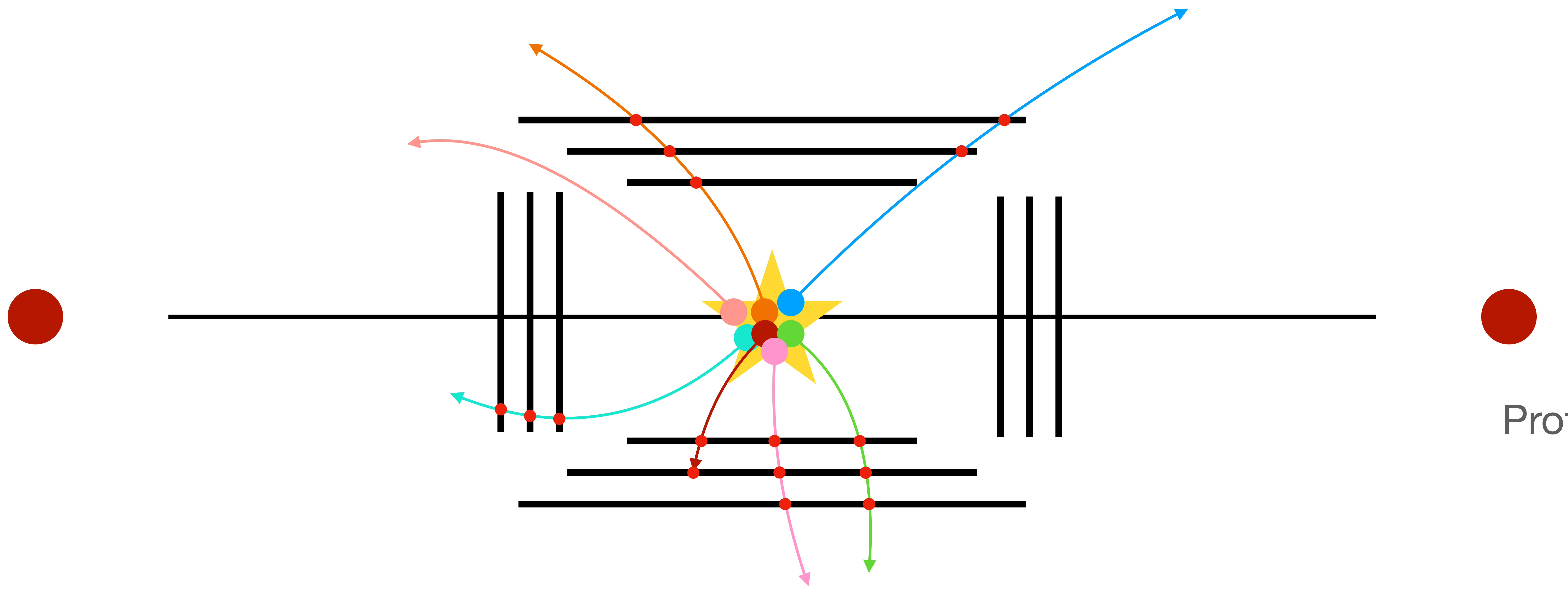
Experiment-scale generation



Experiment-scale generation



Experiment-scale generation



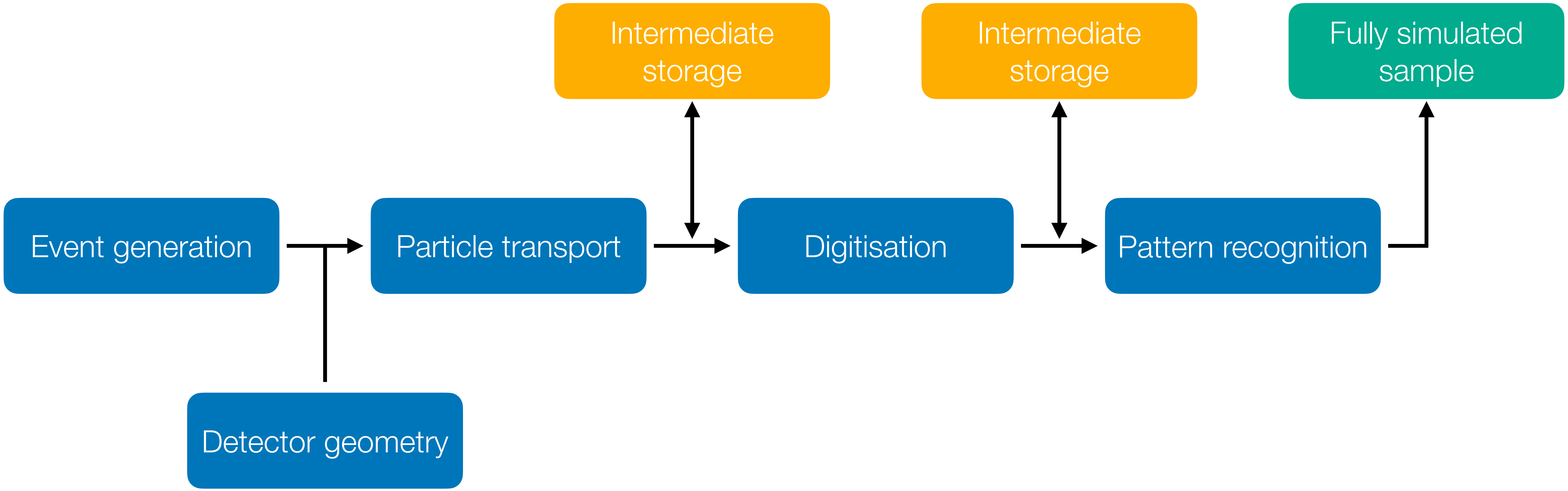
Protons

daniel.hynds@physics.ox.ac.uk

Experiment-scale generation

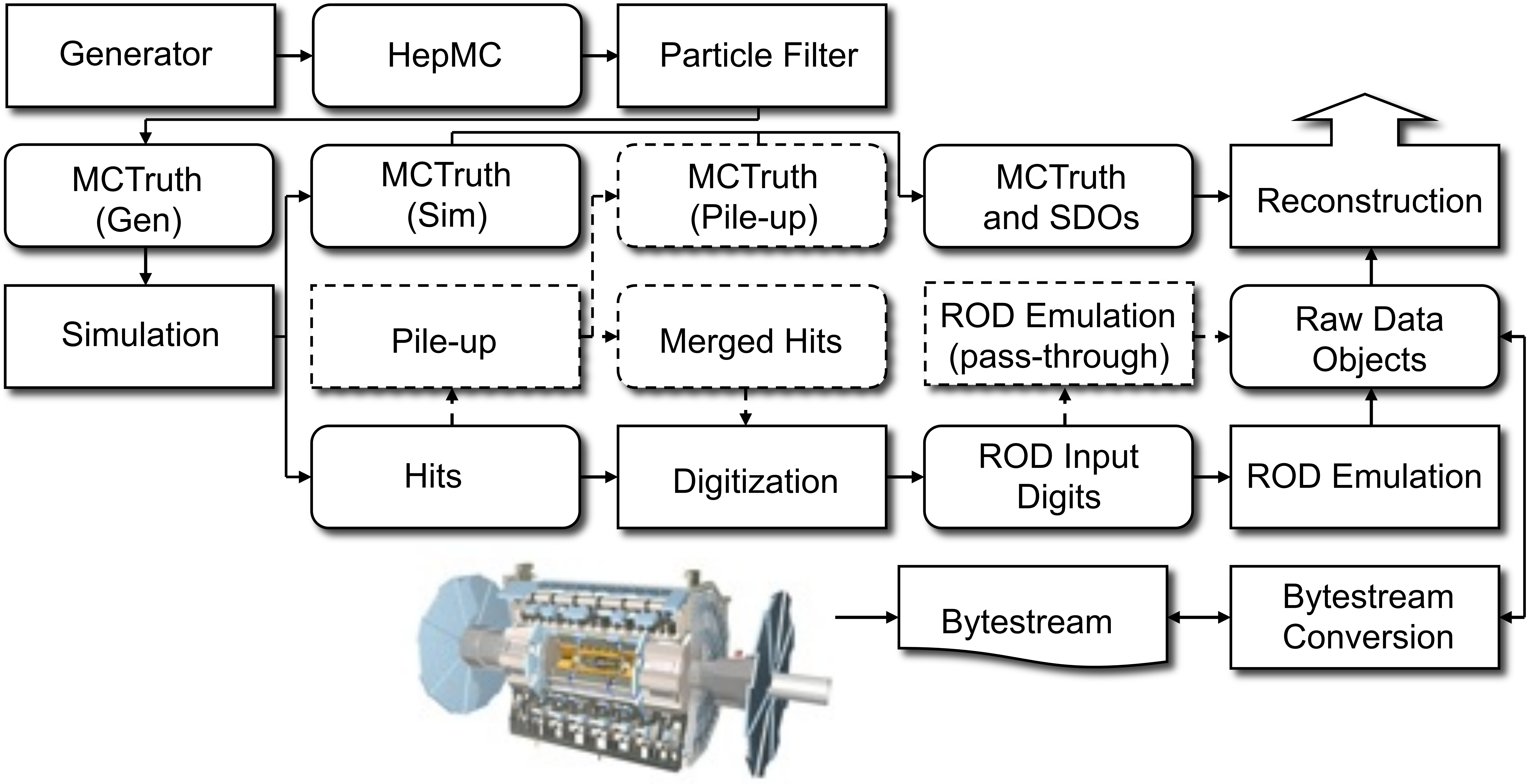
On the scale of LHC experiments the MC chain is rather long and optimised to minimise re-running expensive tasks

- Intermediate outputs after major steps
- For ATLAS in 2015 it took **1 hour** to fully simulate a single minimum bias event at the LHC



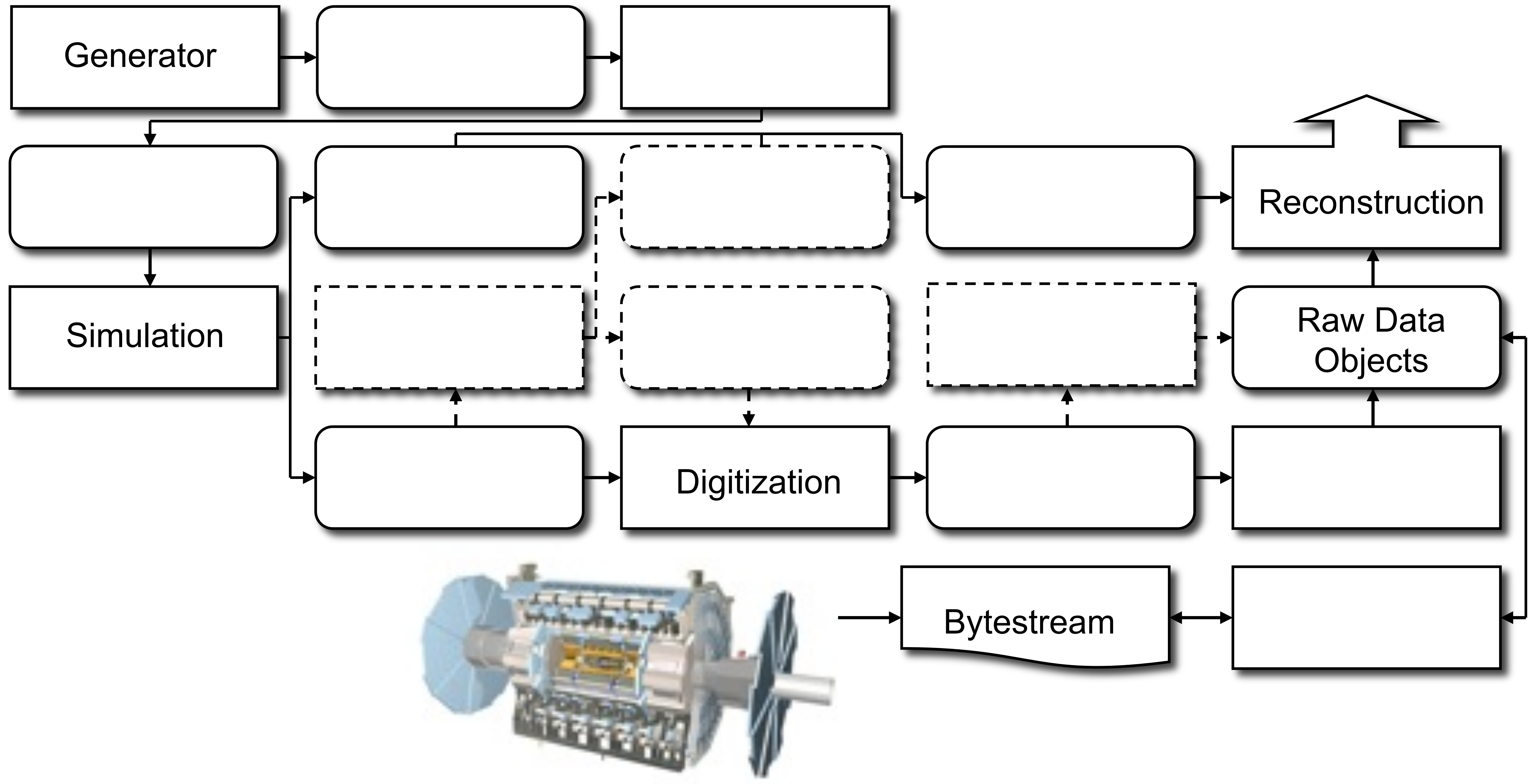
Experiment-scale generation - ATLAS

Picture courtesy Z. Marshall



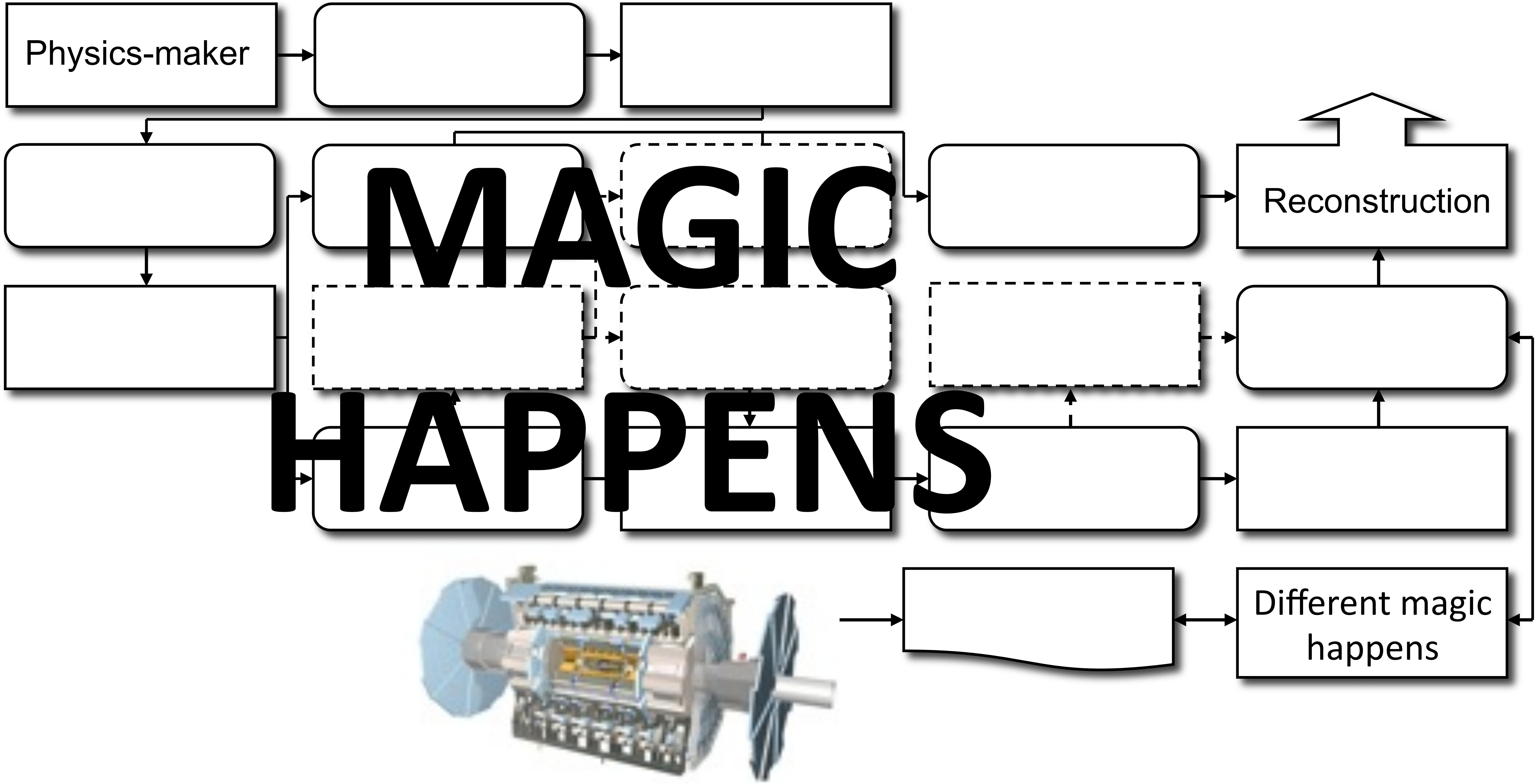
Experiment-scale generation - ATLAS

Picture courtesy Z. Marshall



Experiment-scale generation - ATLAS

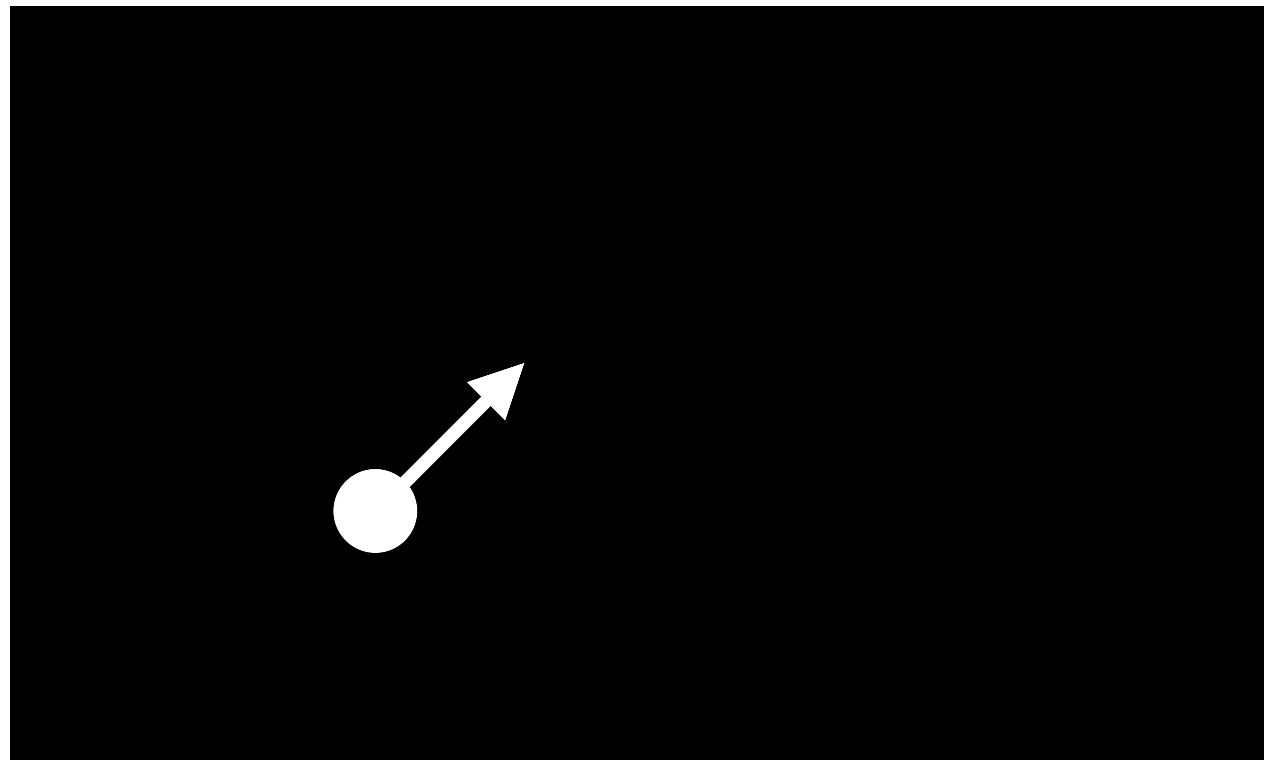
Picture courtesy Z. Marshall



Geant4 - the particle interaction toolkit

What we are concerned with in this course starts with Geant4

- It deals with particle transport, with detailed modelling of interactions with matter, including energy loss, scattering, showering, etc
- Geant4 starts with a collection of particles, and needs to have the world geometry defined (and everything in it)
- The level of detail that Geant4 goes into can be tailored by choosing how far to step through different materials - obviously one 300 um step through a silicon sensor will not give the same results as 1 um steps



What Geant4 doesn't do

http://opendata.atlas.cern/release/2020/documentation/datasets/mc.html

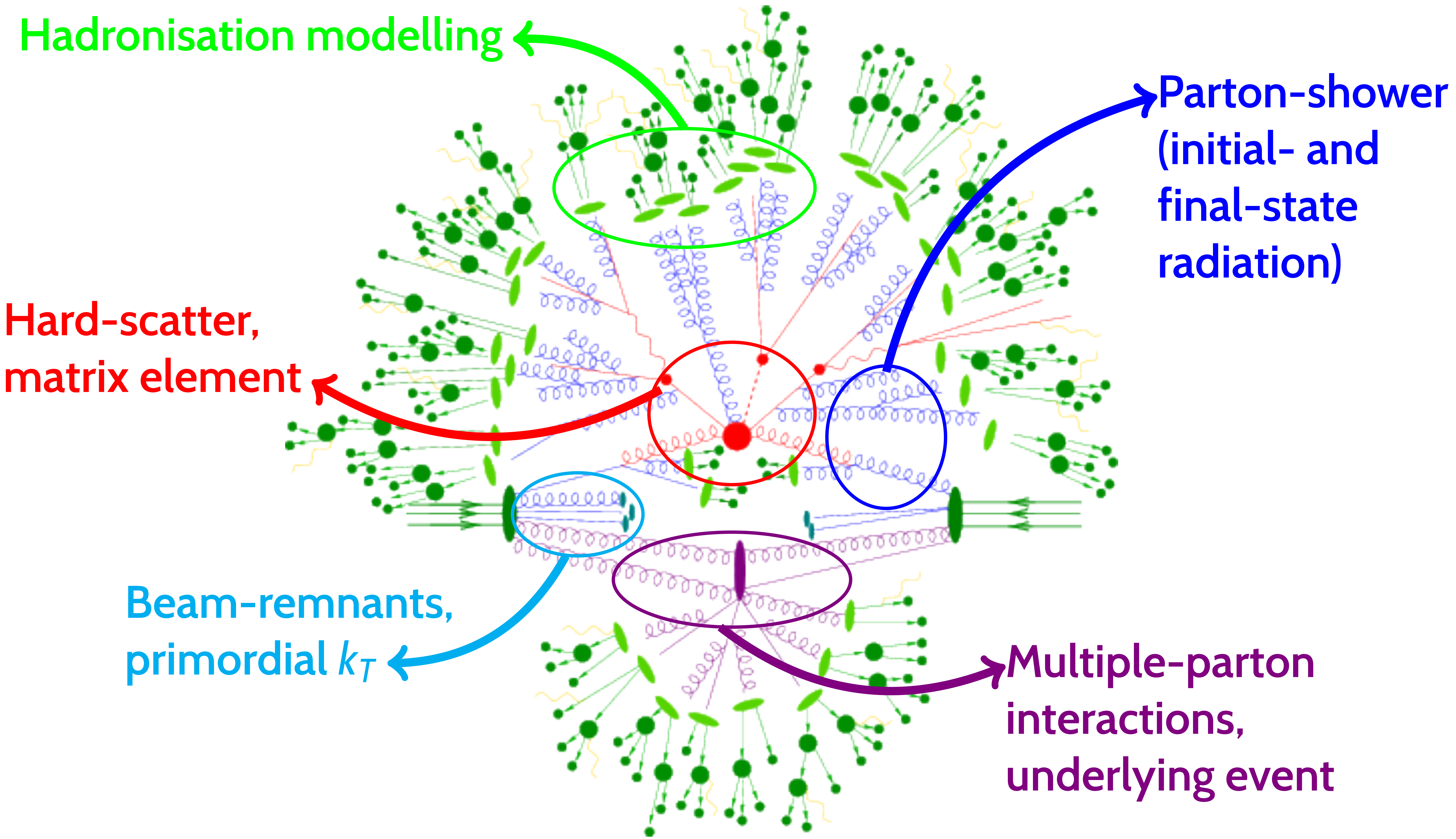
It is not a particle physics generator! There are many of these in HEP:

- Herwig
- Jimmy
- MadGraph
- MG5_aMC@NLO
- POWHEG
- Pythia
- Sherpa
- Whizard
- …

Process	Unique "channelNumber"	Generator, hadronisation	Additional information
		Top-quark production	
$t\bar{t}$+jets	410000	POWHEG-BOX v2 [68] + PYTHIA 8 [69]	only 1ℓ and 2ℓ decays of $t\bar{t}$-system
single (anti)top t-channel	(410012) 410011	POWHEG-BOX v1 + PYTHIA 6 [70]	
single (anti)top Wt-channel	(410014) 410013	POWHEG-BOX v2 + PYTHIA 6	
single (anti)top s-channel	(410026) 410025	POWHEG-BOX v2 + PYTHIA 6	
		W/Z (+ jets) production	
$Z \rightarrow ee,\ \mu\mu,\ \tau\tau$	361106 – 361108	POWHEG-BOX v2 + PYTHIA 8	LO accuracy up to $N_{\text{jets}} = 1$
$W \rightarrow e\nu,\ \mu\nu,\ \tau\nu$	361100 – 361105	POWHEG-BOX v2 + PYTHIA 8	LO accuracy up to $N_{\text{jets}} = 1$
$W \rightarrow e\nu,\ \mu\nu,\ \tau\nu$ + jets	364156 – 364197	SHERPA 2.2 [71]	LO accuracy up to 3-jets final states
$Z \rightarrow ee,\ \mu\mu,\ \tau\tau$ + jets	364100 – 364141	SHERPA 2.2	LO accuracy up to 3-jets final states
		Diboson production	
WW	363359, 363360	SHERPA 2.2	$qq'\ell\nu$ final states
WW	363492	SHERPA 2.2	$\ell\nu\ell'\nu'$ final states
ZZ	363356	SHERPA 2.2	$qq'\ell^+\ell^-$ final states
ZZ	363490	SHERPA 2.2	$\ell^+\ell^-\ell'^+\ell'^-$ final states
WZ	363358	SHERPA 2.2	$qq'\ell^+\ell^-$ final states
WZ	363489	SHERPA 2.2	$\ell\nu qq'$ final states
WZ	363491	SHERPA 2.2	$\ell\nu\ell^+\ell^-$ final states
WZ	363493	SHERPA 2.2	$\ell\nu\nu\nu'$ final states
		SM Higgs production ($m_{\text{H}} = 125$ GeV)	
ggF, $H \rightarrow WW$	345324	POWHEG-BOX v2 + PYTHIA 8	$\ell\nu\ell'\nu'$ final states
VBF, $H \rightarrow WW$	345323	POWHEG-BOX v2 + PYTHIA 8	$\ell\nu\ell'\nu'$ final states
ggF, $H \rightarrow ZZ$	345060	POWHEG-BOX v2 + PYTHIA 8	$\ell^+\ell^-\ell'^+\ell'^-$ final states
VBF, $H \rightarrow ZZ$	344235	POWHEG-BOX v2 + PYTHIA 8	$\ell^+\ell^-\ell'^+\ell'^-$ final states
ZH, $H \rightarrow ZZ$	341947	PYTHIA 8	$\ell^+\ell^-\ell'^+\ell'^-$ final states
WH, $H \rightarrow ZZ$	341964	PYTHIA 8	$\ell^+\ell^-\ell'^+\ell'^-$ final states
ggF, $H \rightarrow \gamma\gamma$	343981	POWHEG-BOX v2 + PYTHIA 8	
VBF, $H \rightarrow \gamma\gamma$	345041	POWHEG-BOX v2 + PYTHIA 8	
$WH(ZH)$, $H \rightarrow \gamma\gamma$	345318, 345319	POWHEG-BOX v2 + PYTHIA 8	
$t\bar{t}H$, $H \rightarrow \gamma\gamma$	341081	aMC@NLO [72] + PYTHIA 8	
		BSM production	
$Z' \rightarrow t\bar{t}$	301325	PYTHIA 8	$m_{Z'} = 1$ TeV
$\tilde{\ell}\tilde{\ell}' \rightarrow \ell\tilde{\chi}_1^0\ell'\tilde{\chi}_1^{0\prime}$	392985	aMC@NLO + PYTHIA 8	$m_{\tilde{\ell}} = 600$ GeV, $m_{\tilde{\chi}_1^0} = 300$ GeV

Event generation - ATLAS

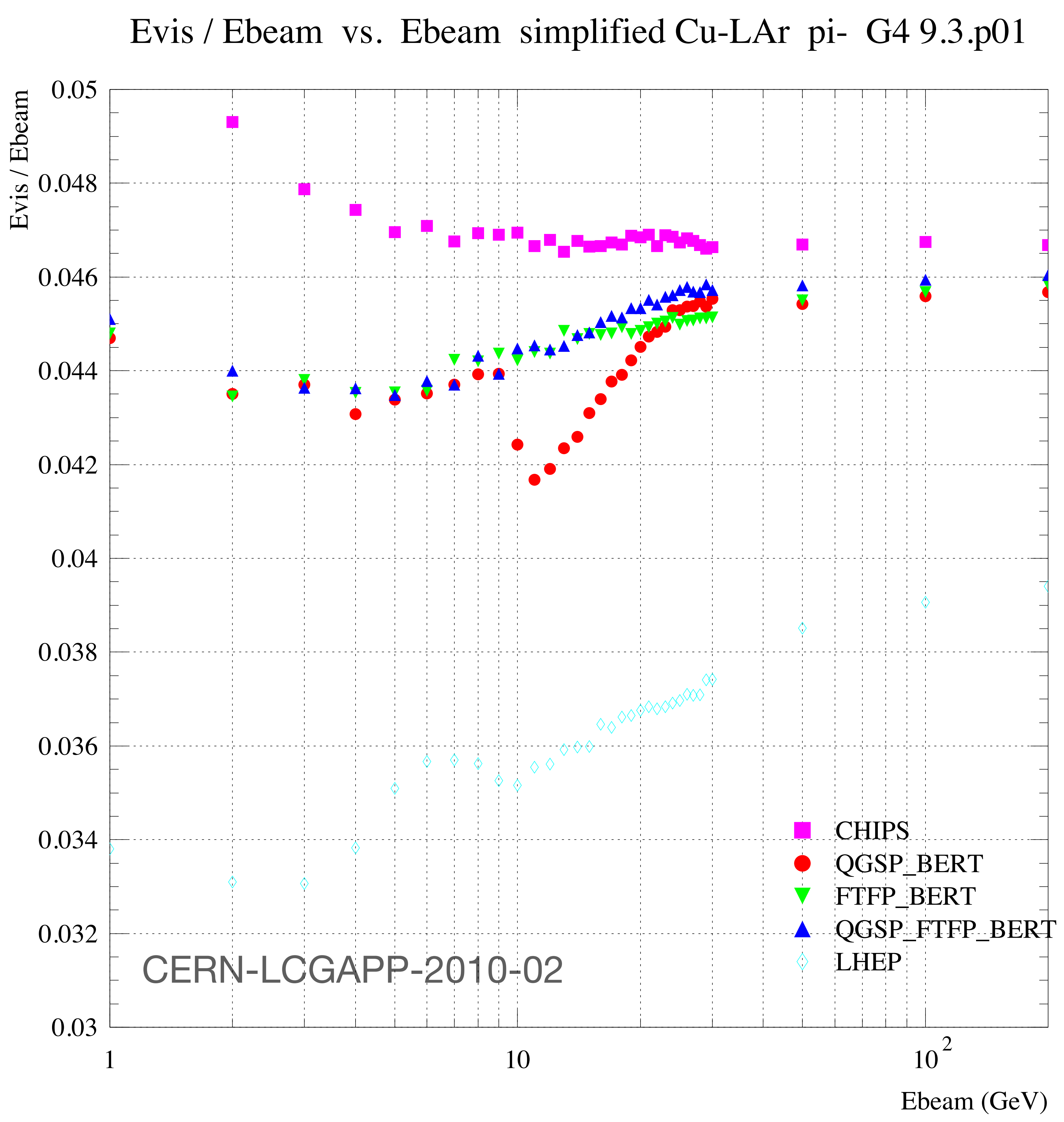
Picture courtesy J. Robinson



Things to be aware of

Geant4 developers are not mystics - simulation output is only as good as the data going into it!

- In particular, cross-sections for processes involving neutrons are very difficult to measure - output from Geant4 may vary from data by ~ order of magnitude
- The selection of physics processes to include in the simulation is set by the **physics list** - using the wrong one can give drastically different results



Working directly with Geant4

Geant4 is a large and complex package, which was first released in 1998 as a successor to GEANT (and was the first iteration to use c++)

- There is quite some work involved in understanding and creating detector geometries
- Geant4 will **not** carry out things like charge carrier propagation in semiconducting devices, or front-end electronics
- Geant4 typically wants to be in charge (it has its own run manager) and any post-particle-transport steps are added at the end as ‘post action hooks’

You can write your own native Geant4 scripts, but these are not terribly friendly and if you want to re-run eg. your charge carrier transport you have to start your own reading/writing of intermediate files, etc.

- Typical development loop follows PhD student life cycle: start, hack, get some results, hack, spaghetti, unintelligible when PhD student leaves, start from scratch with next student…

Allpix^2 - “A Modular Simulation Framework for Silicon Detectors”

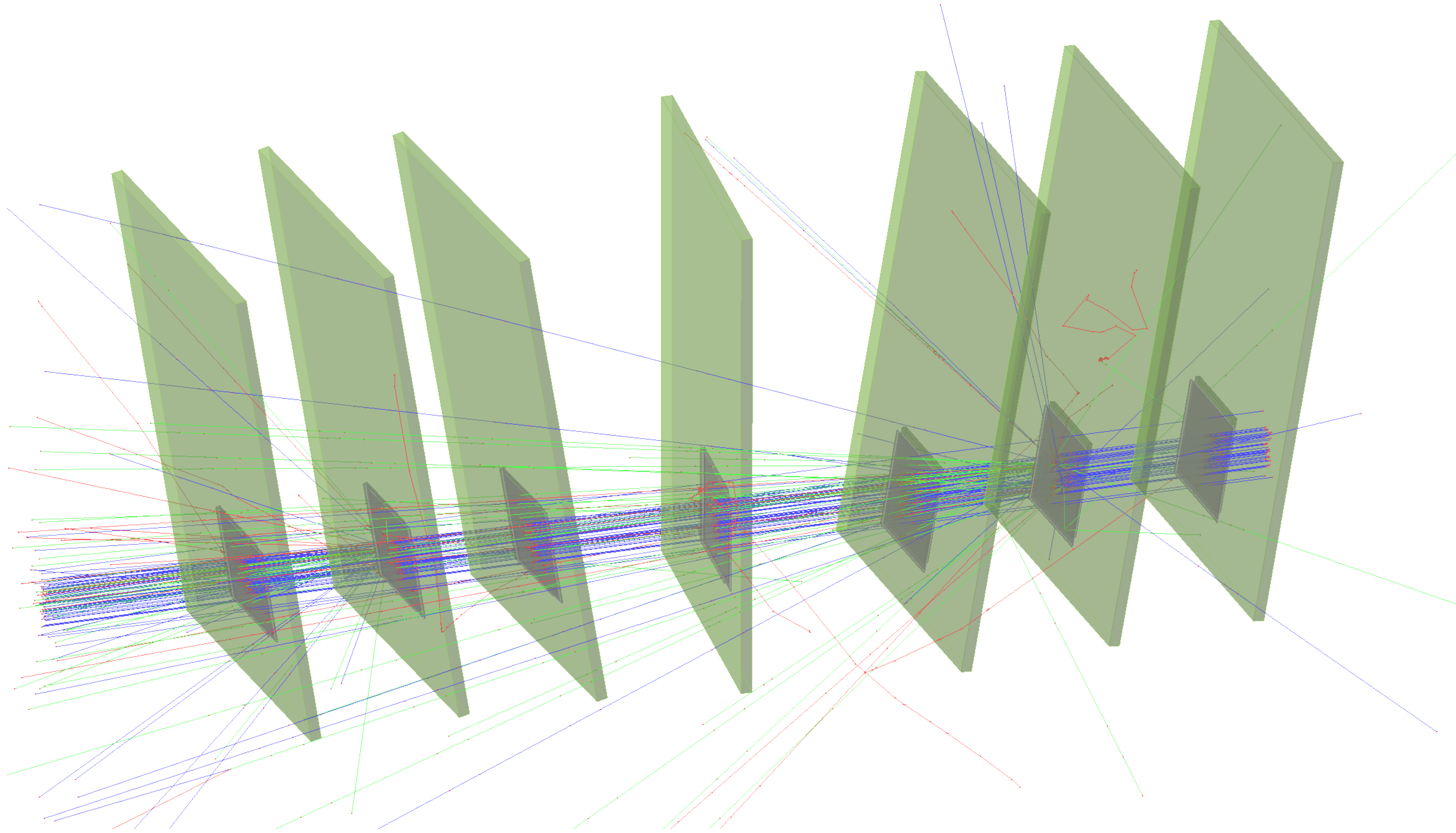
To counter this method of Italian cooking, allpix^2 was developed: modern, modular software that encapsulates Geant4 and adds the semiconductor physics + electronics descriptions

It is extensively documented

- https://project-allpix-squared.web.cern.ch
- https://gitlab.cern.ch/allpix-squared/allpix-squared

One of the main advantages is in making it very quick to simulate typical setups found in semiconductor R&D, without having to spend time defining geometries, materials, etc.

- Effort can be spent on implementing solid-state physics models



Allpix² structure

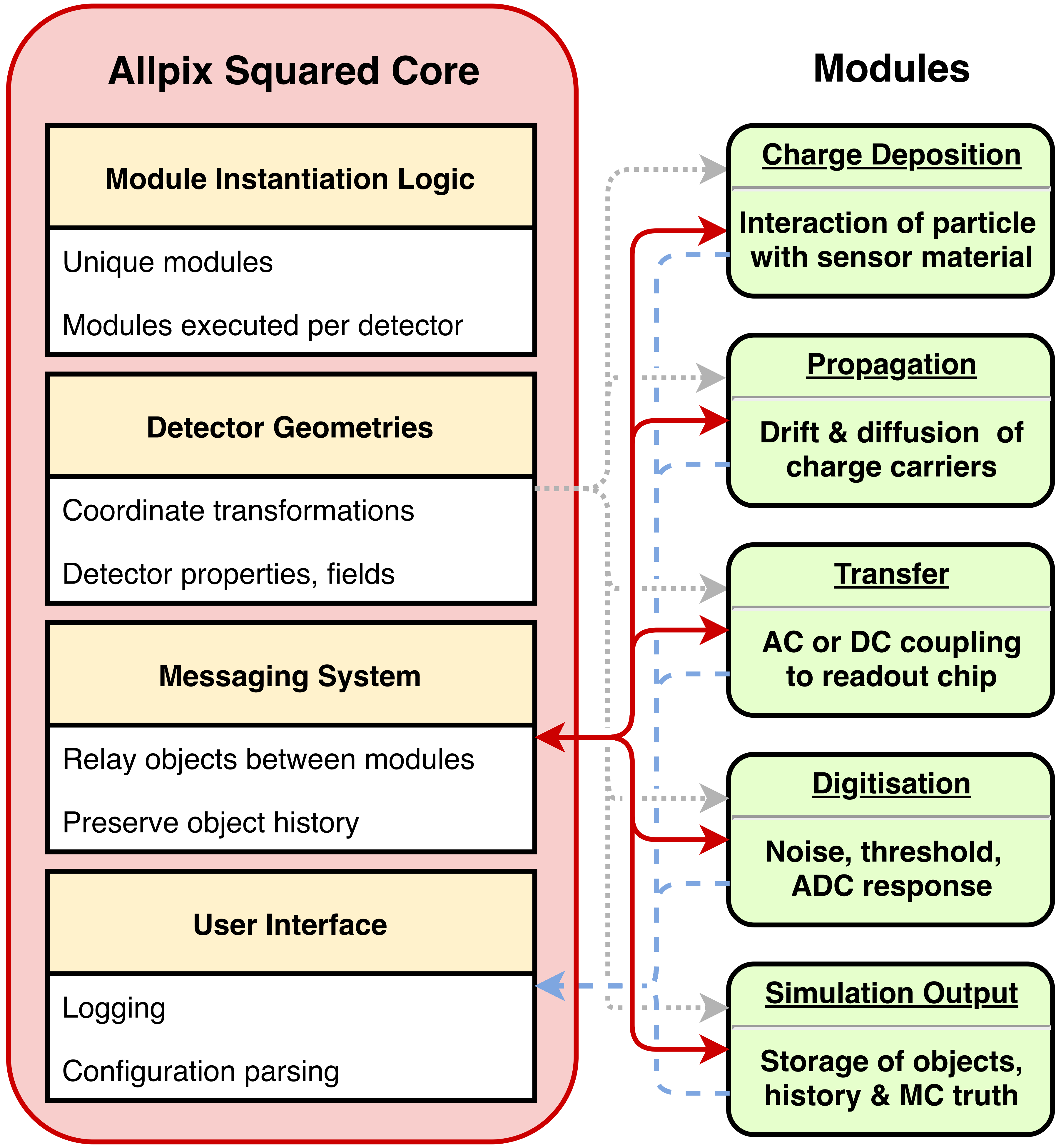
Core of the software handles all of the main infrastructure

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

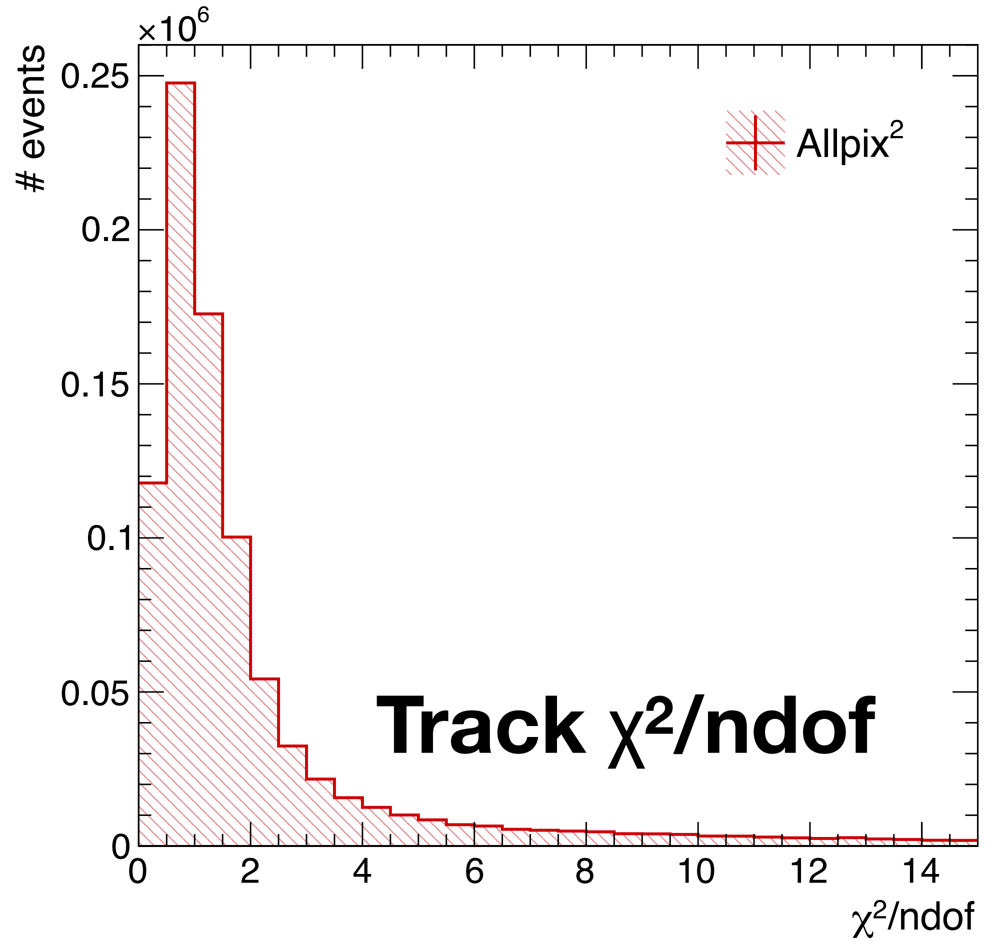
Modules are the work-horses which handle all of the real detector simulation

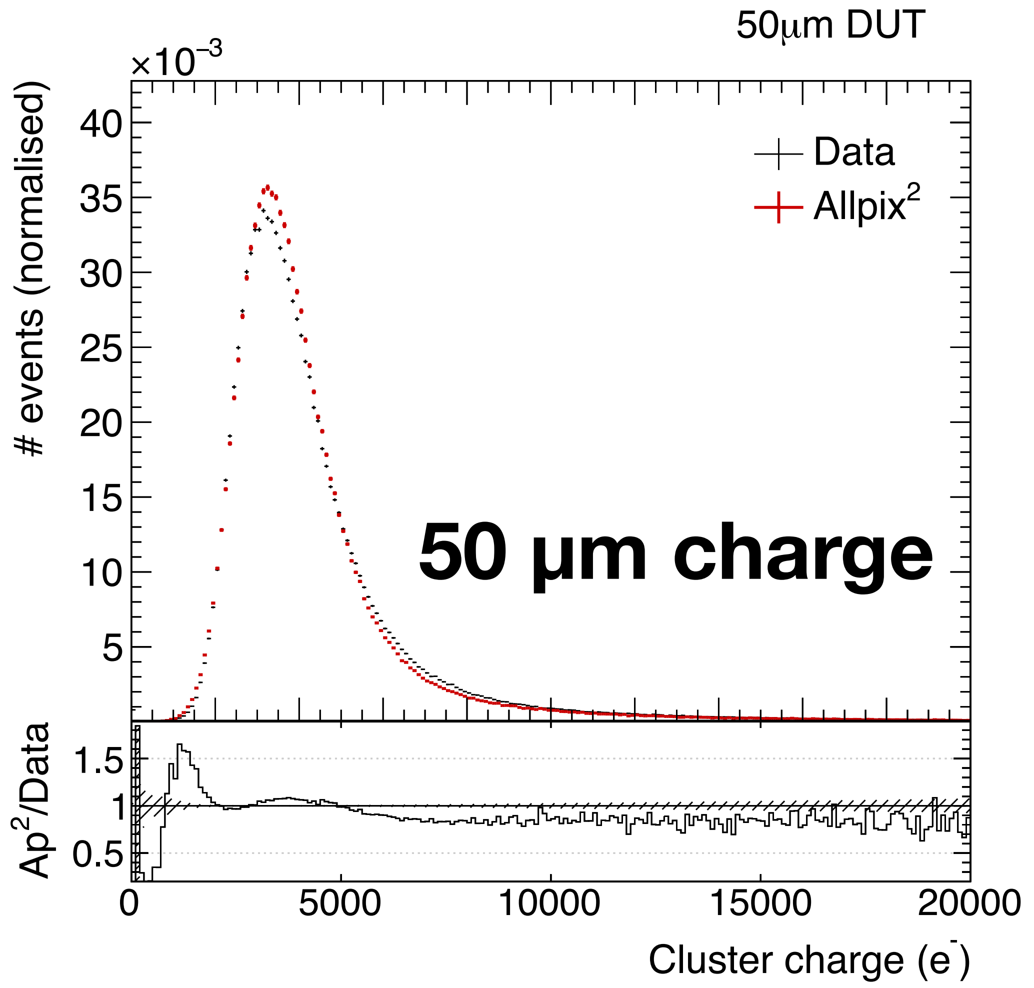
- Modular approach means modules are entirely independent
- Defined input objects on which they act, and output objects which will be produced

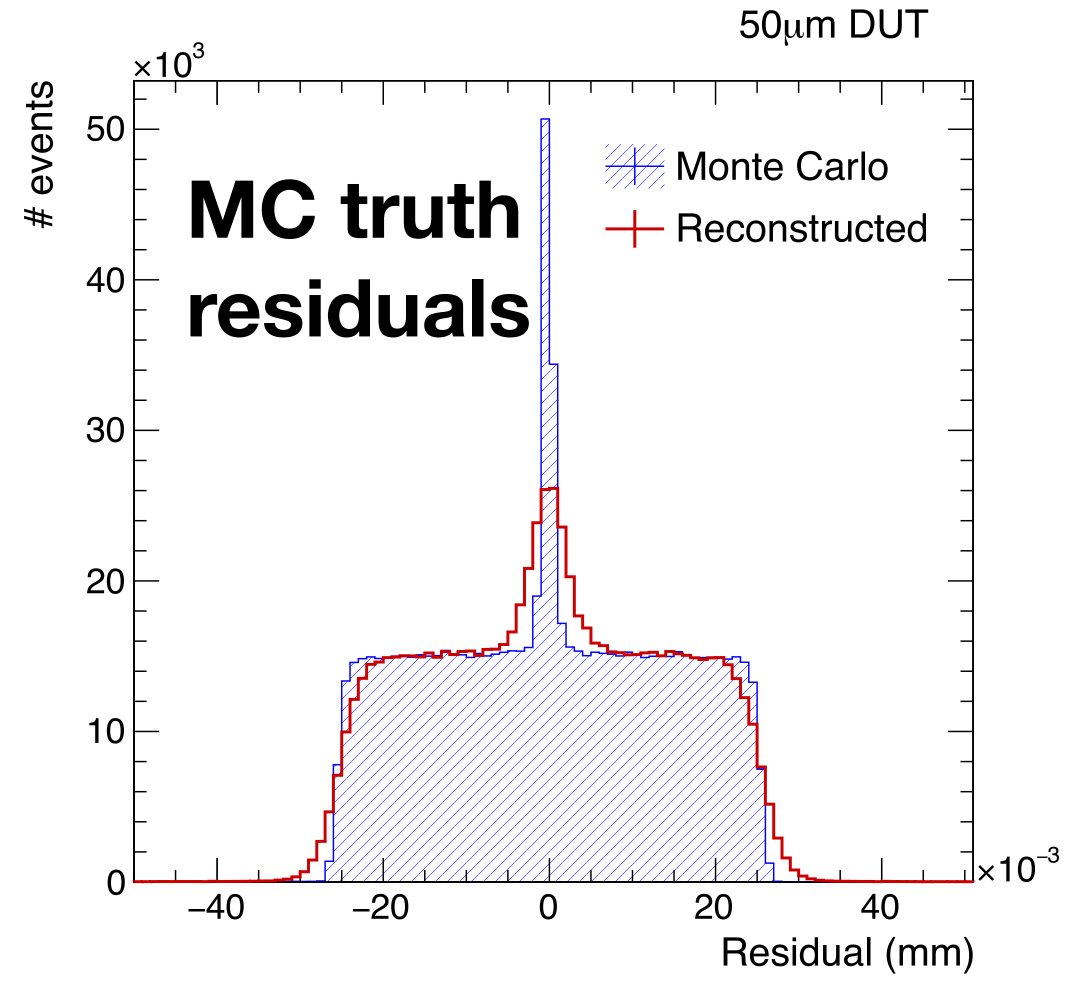


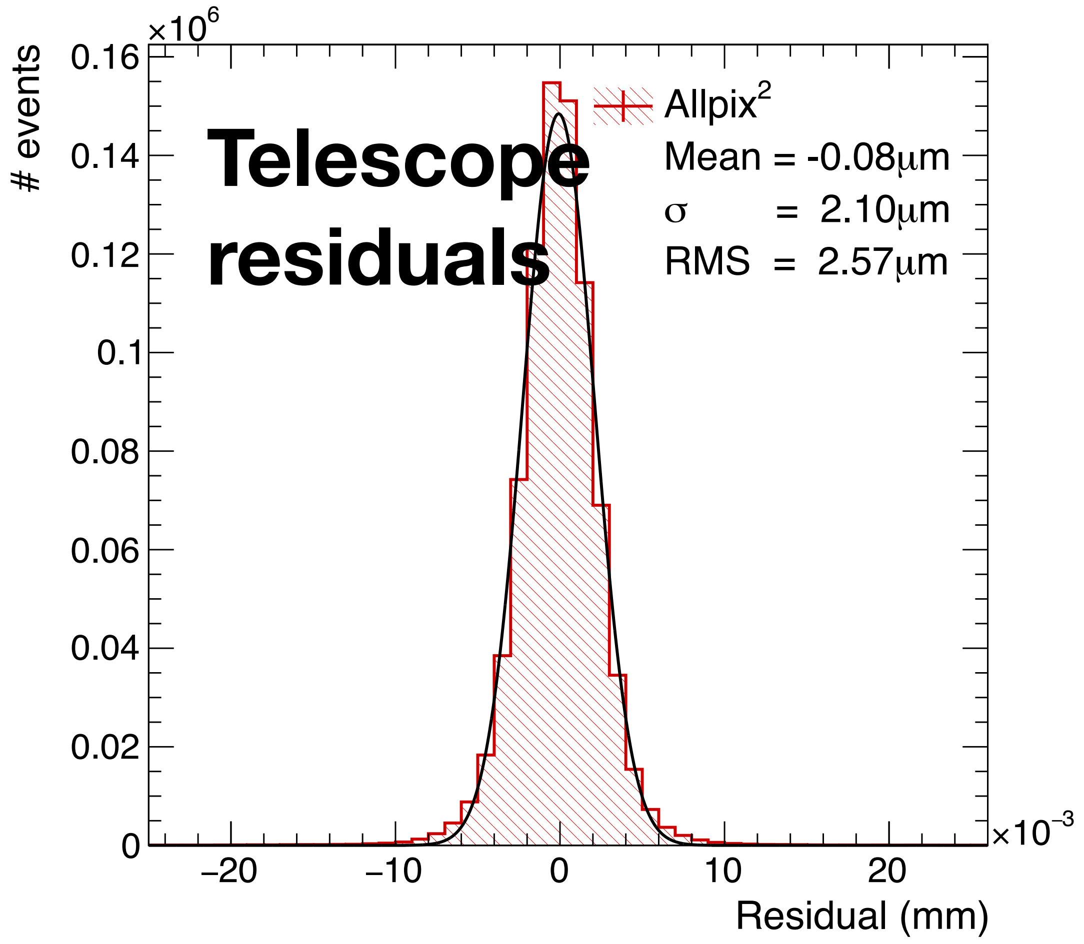
Allpix² validation

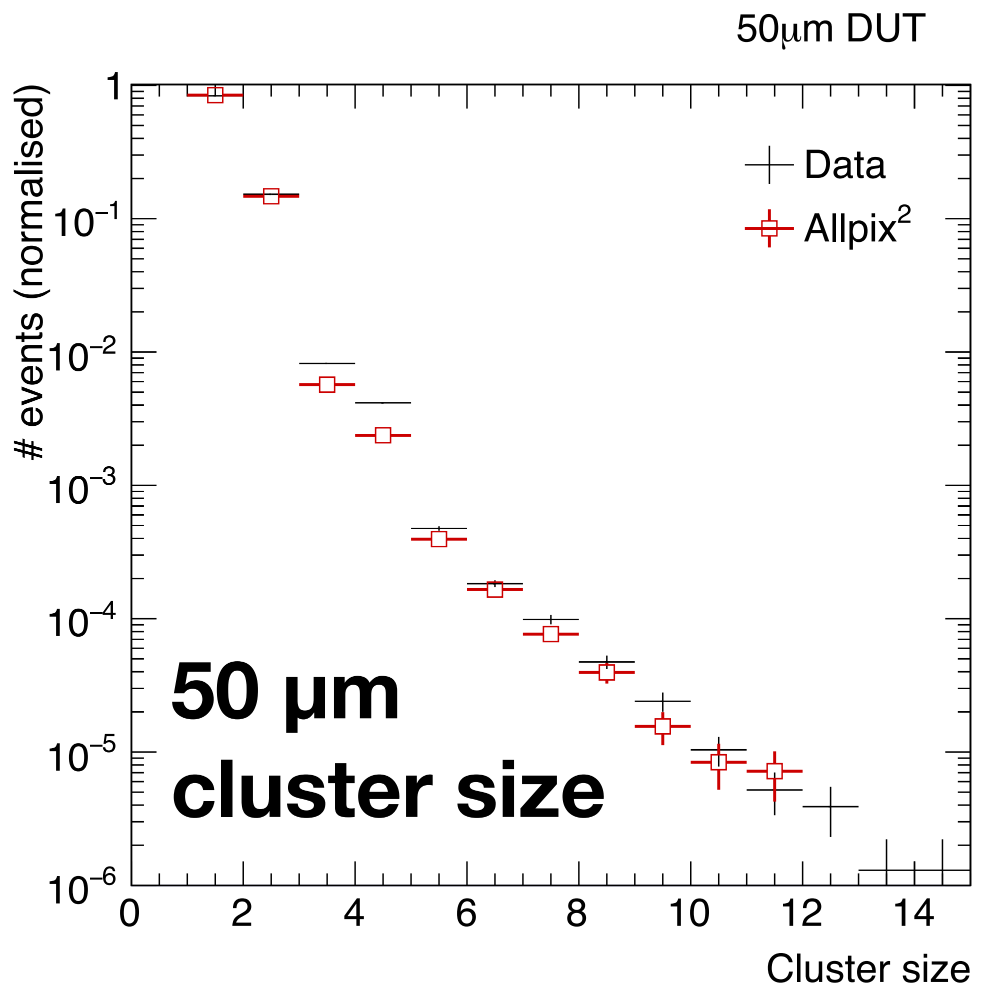
S. Spannagel et al., NIM A 901 (2018)

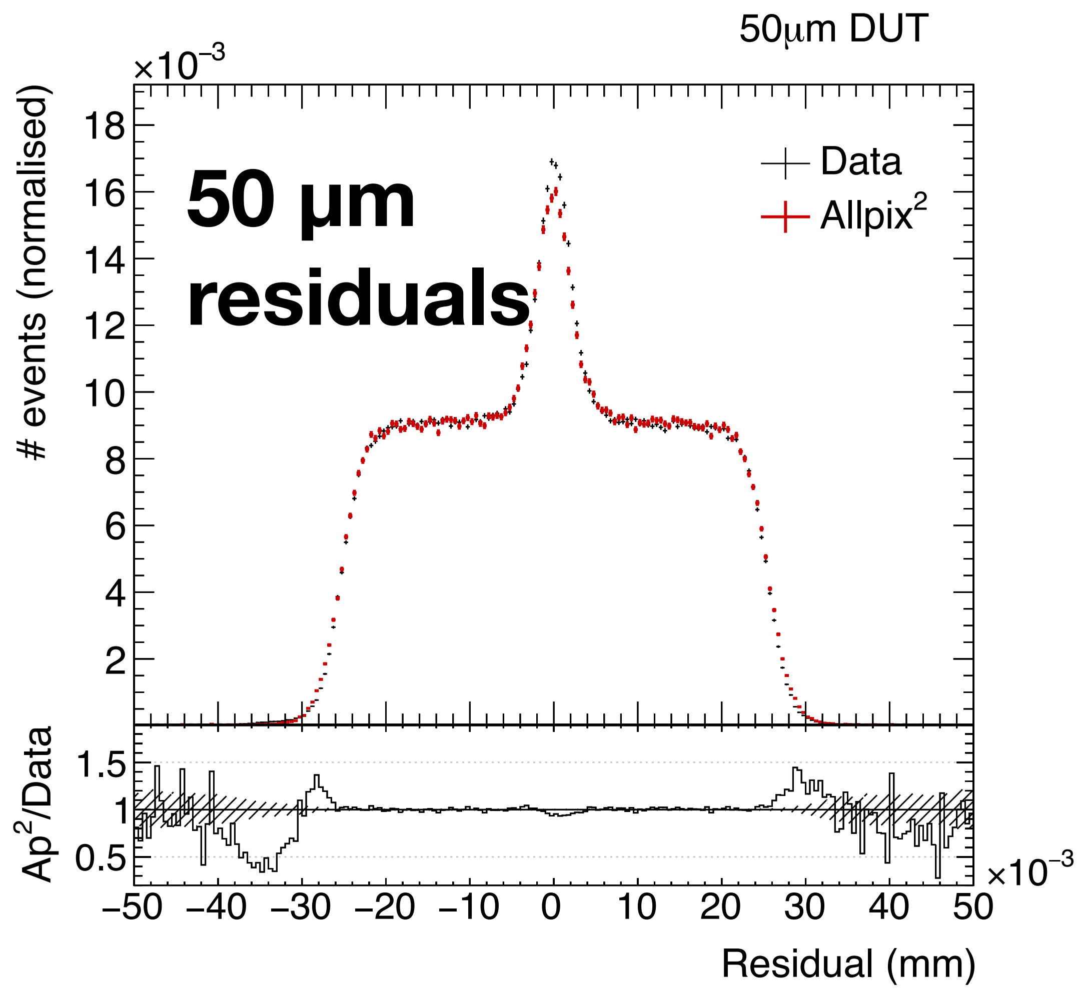








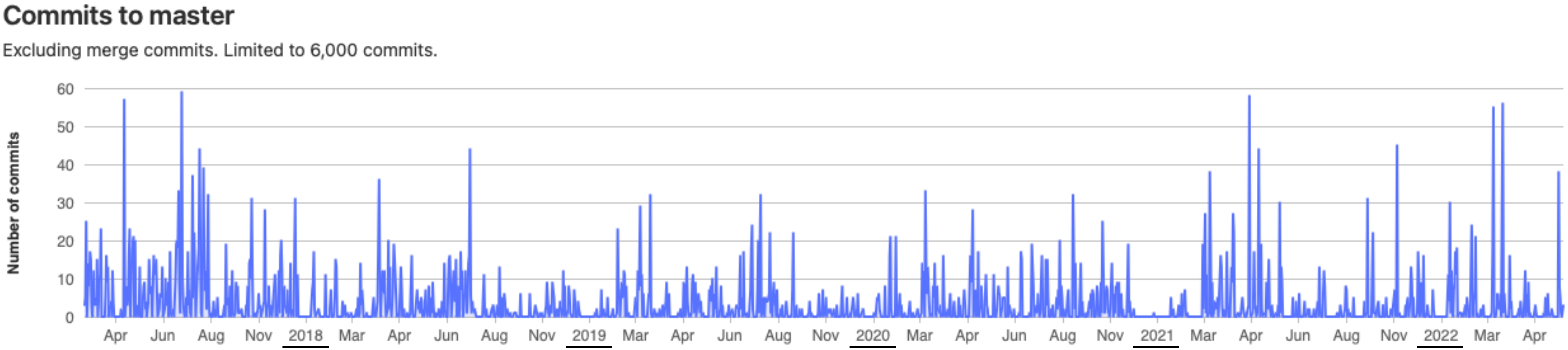




Allpix² timeline

Allpix-Squared website went live July 20, 2017

- First stable release in September 2017
- Latest patch 2.3 from May 16 this year



Allpix^2 developers

> 60 forks, with >80 contributors 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 (M.Ali) 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

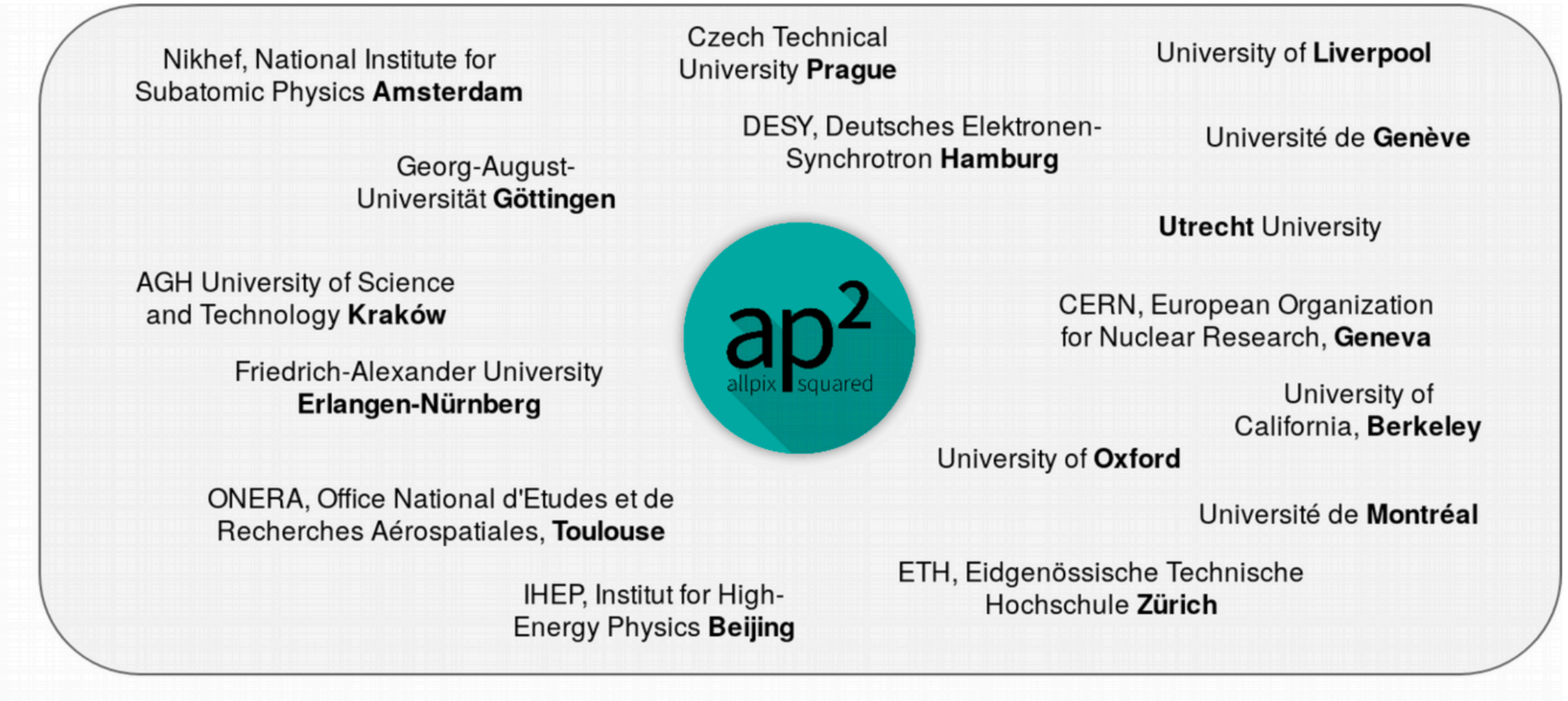
- Infrastructure maintained by S.Spannagel
- Code review still draws on experience of K.Wolters

- **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
- ???



daniel.hynds@physics.ox.ac.uk

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)

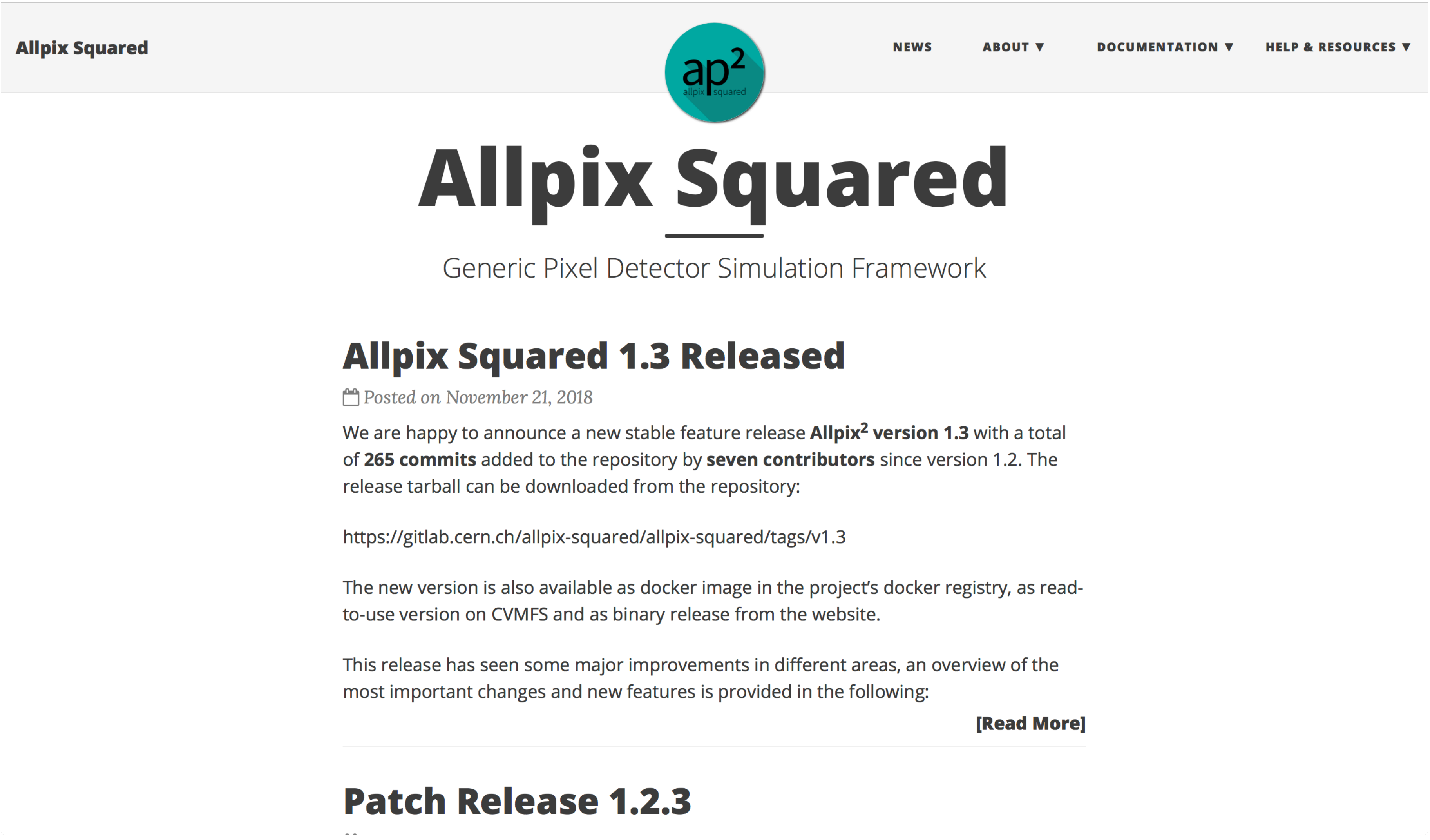
The most commonly used versions of this are to check out the software and compile it yourself - either locally or on a system with cvmfs access

- 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/

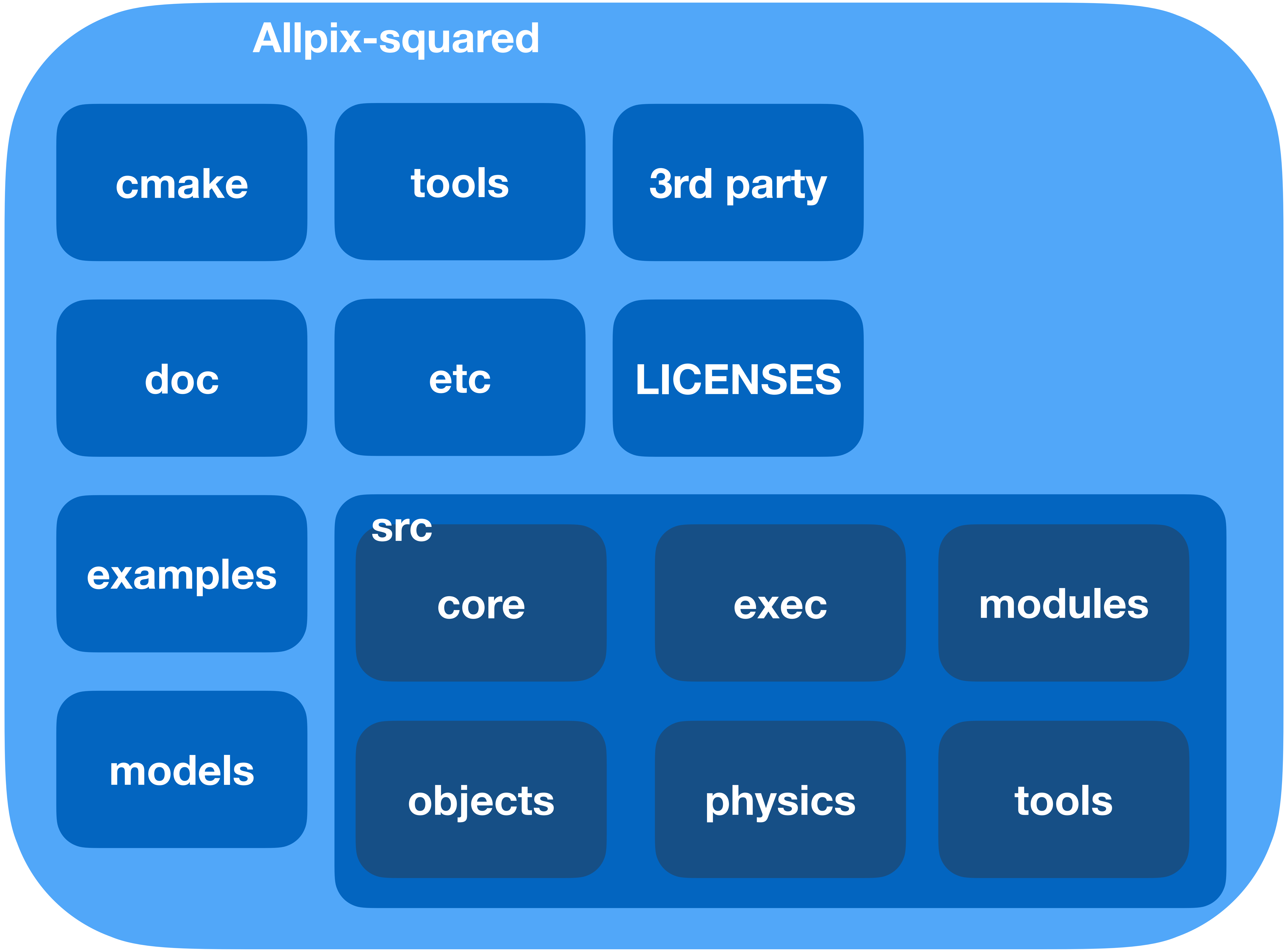


We will work on lxplus for this tutorial

- 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
```

Allpix² navigation



Looking around - what’s there

```
$ ls -l
```

3rdparty	Small external objects which can be used (eg. Fast iterators)
Authors.md	List of authors
CITATION.cff	Software citation
cmake	Macros for cmake, formatting tools to make code style consistent
CMakeLists.txt	Instructions for cmake to prepare allpix-squared compilation
CONTRIBUTING.md	A guide to developers for contributing code
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
LICENSE.md	The allpix-squared licence (open source, MIT)
LICENSES	The full list of licences (MIT, BSL, etc.)
models	**Detector models which can be included in geometry**
README.md	Instructions for getting started, installation locations
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

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

GeometryBuilder Geant4 — Builds the geometry that will be used by Geant4

DepositionGeant4 — Calls Geant4 to step particles through the geometry

Projection Propagation — 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/
```


daniel.hynds@physics.ox.ac.uk

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"
```


daniel.hynds@physics.ox.ac.uk

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

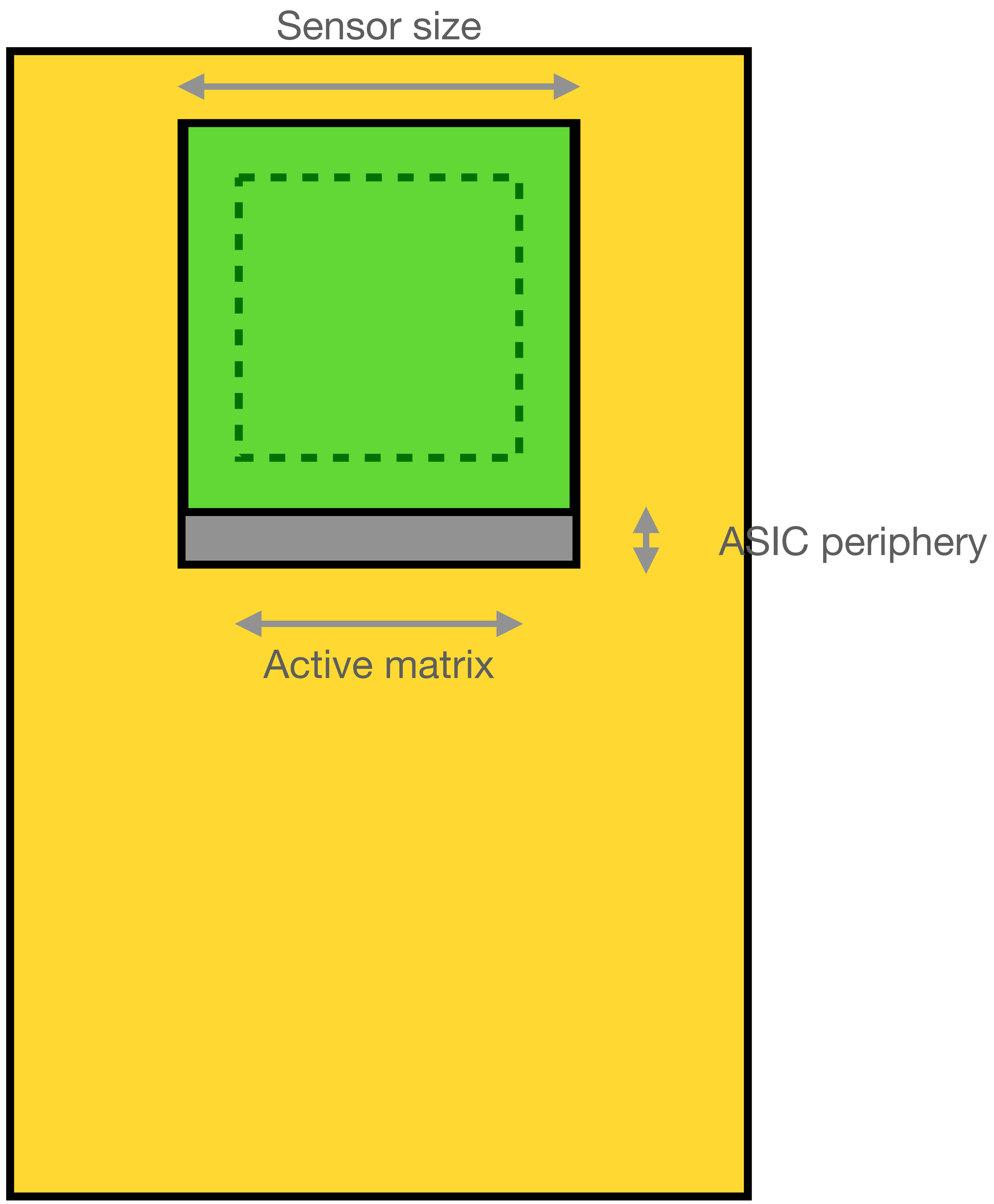
- Each detector is given a unique name (detector1 here) and placed in the global co-ordinate system at a certain position with a given rotation
- Create geometry file tutorial-geometry.conf

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

```
$ ls -l ../models/

CMakeLists.txt
alpide.conf
atlas_itk_r0.conf
clicpix.conf
clicpix2.conf
cmsp1.conf
diode.conf
fei3.conf
ibl_planar.conf
medipix3.conf
mimosa23.conf
mimosa26.conf
rd53a_25.conf
rd53a_50.conf
test.conf
timepix.conf
timepix4.conf
velopix.conf
```

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
```

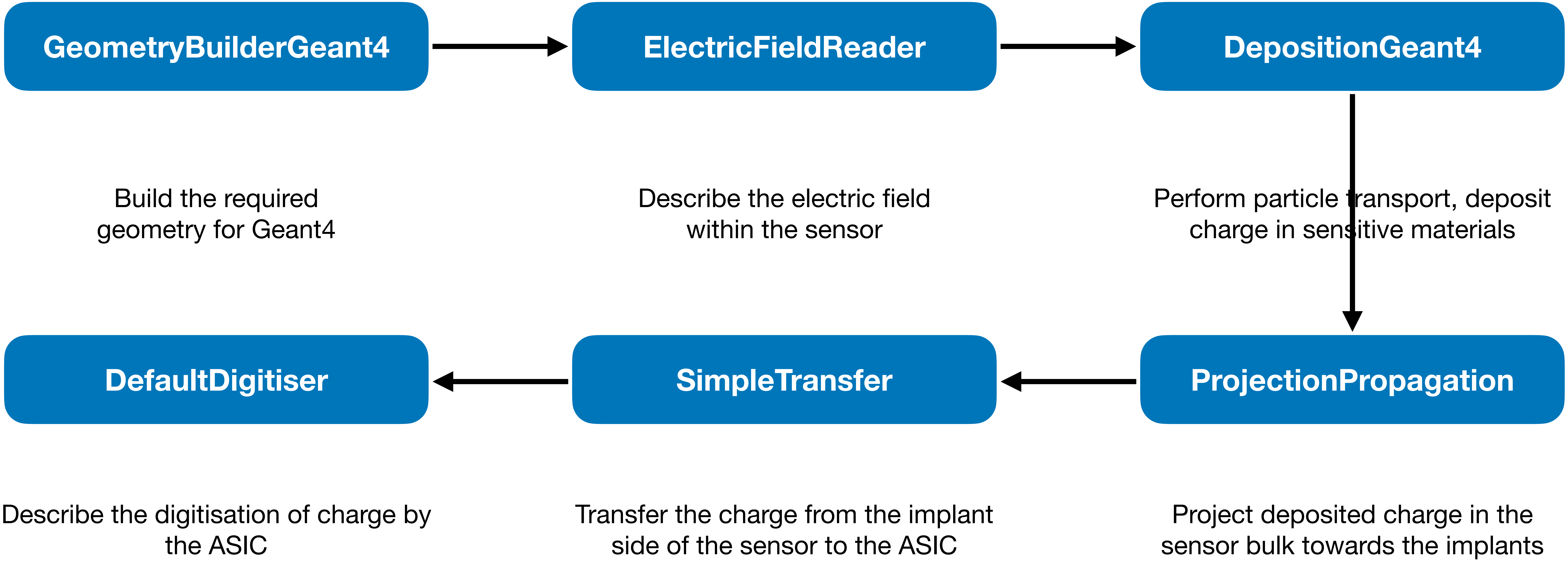
Can now start to add algorithms

- 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



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]
```

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]
```

If you have not defined a variable that is required, then the algorithm will complain! Sensible default **should** exist

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-manualch8.html

project-allpix-squared.web.cern.ch

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
```


daniel.hynds@physics.ox.ac.uk

Electric field definition

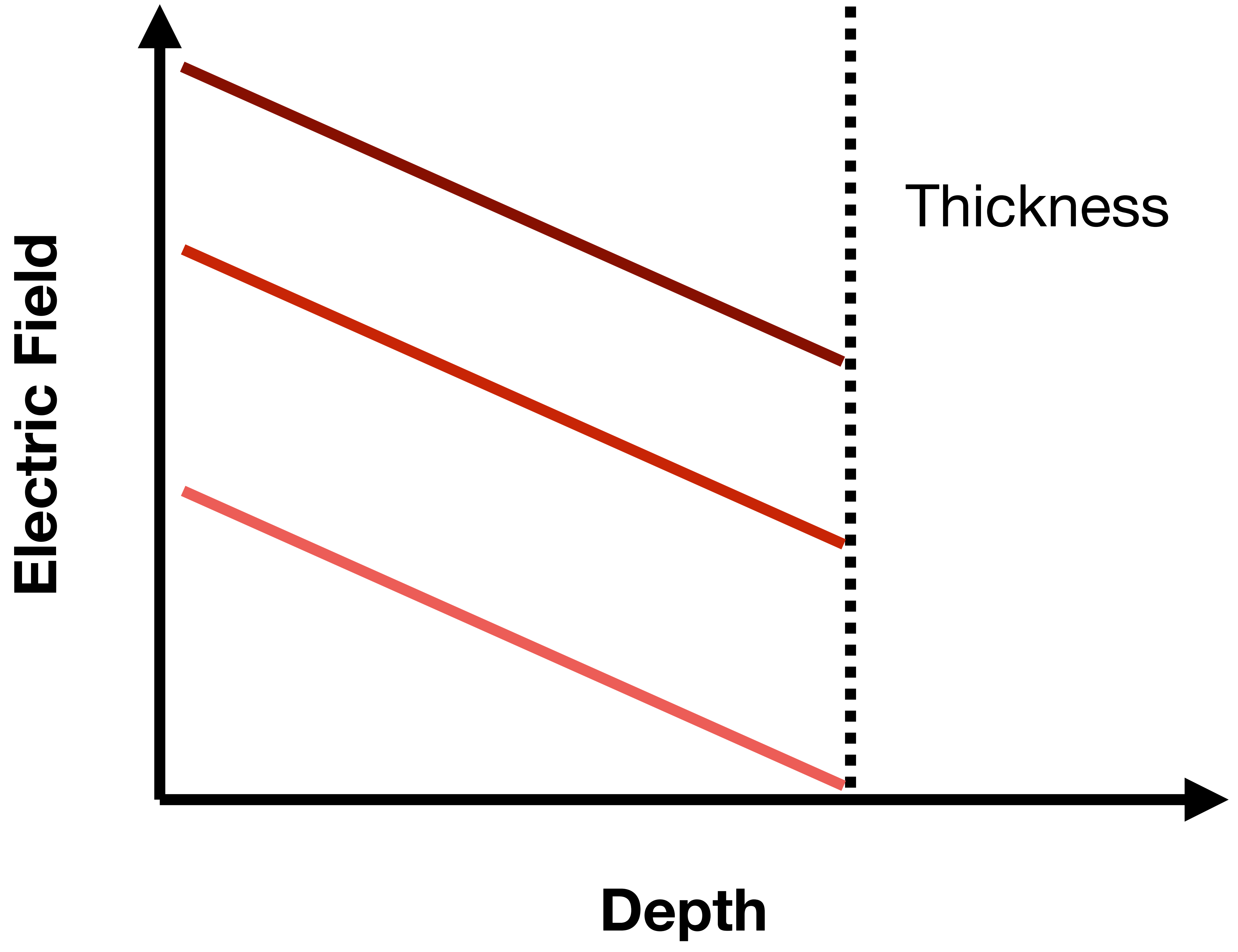
ElectricFieldReader can generate electric fields for the sensor in several ways

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
```

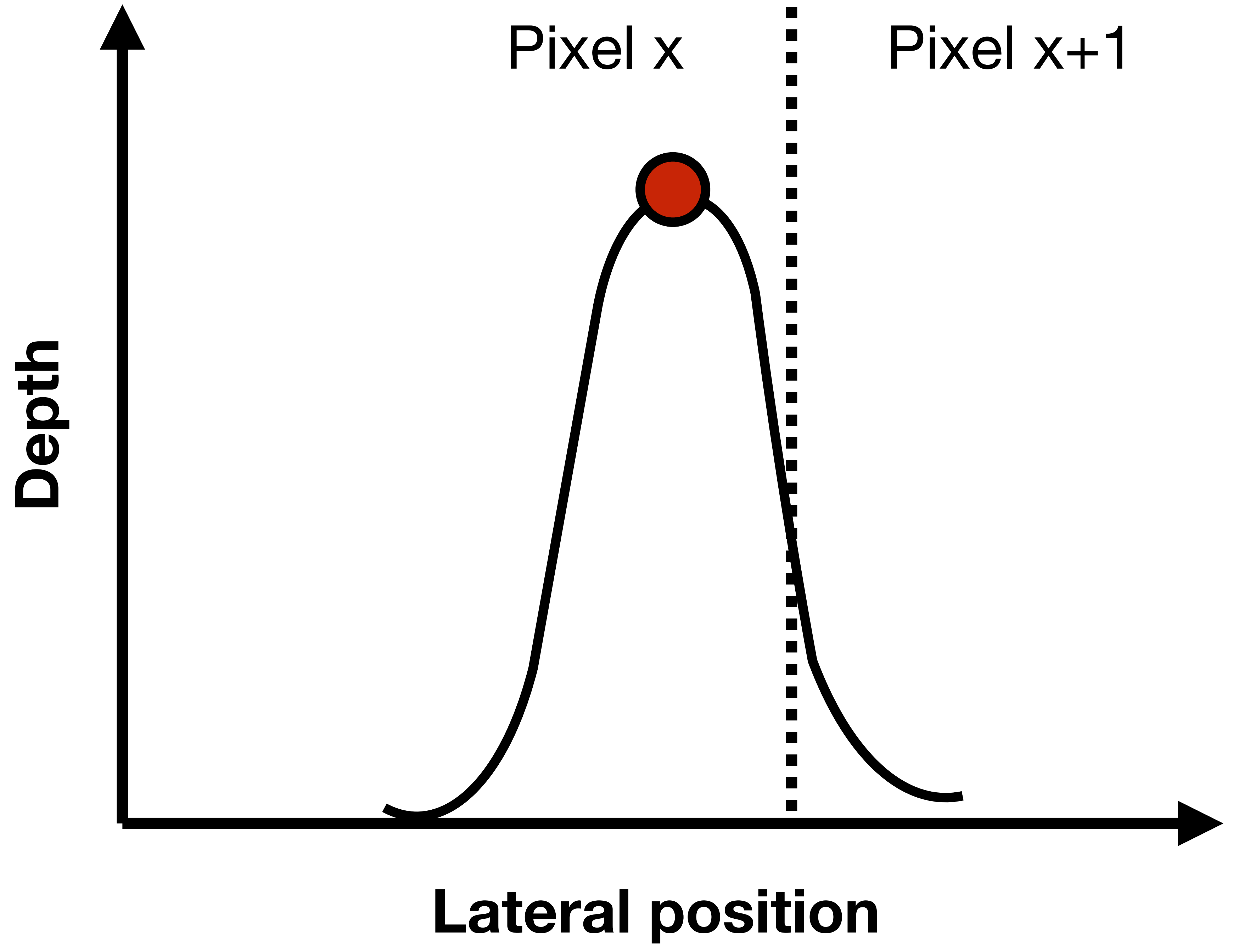


Propagation of deposited charges

ProjectionPropagation 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

`[SimpleTransfer]`

- 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

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

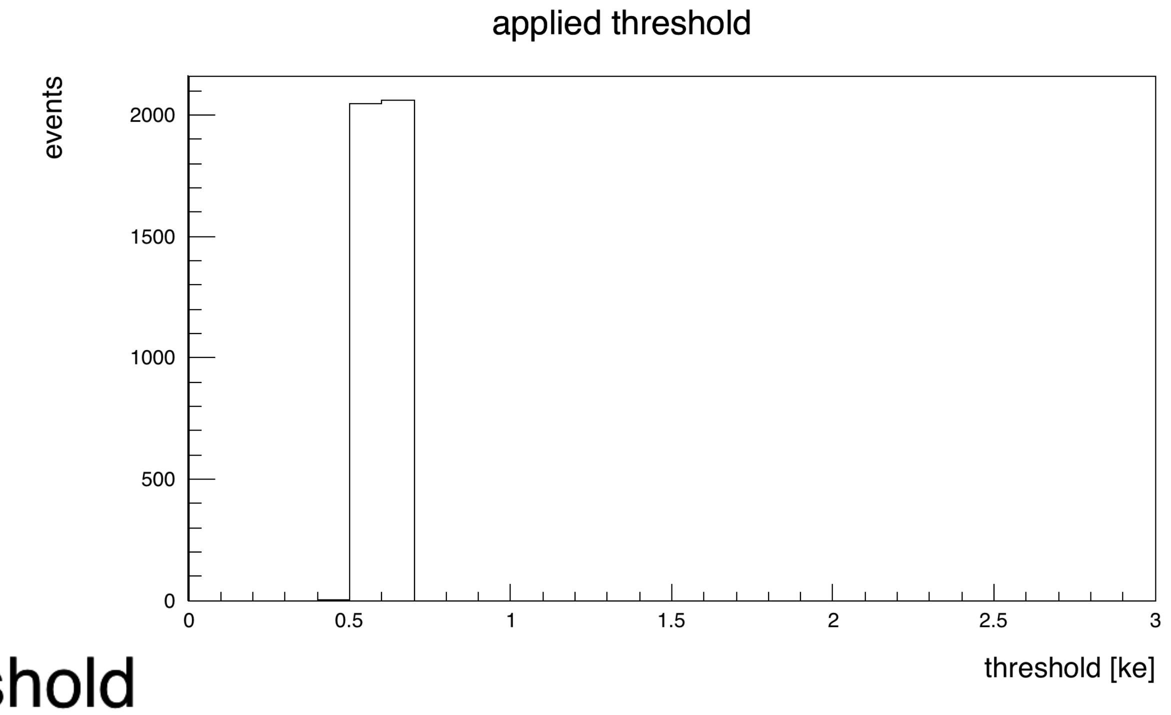
```
log_level = "Warning"

output_plots = 1
```

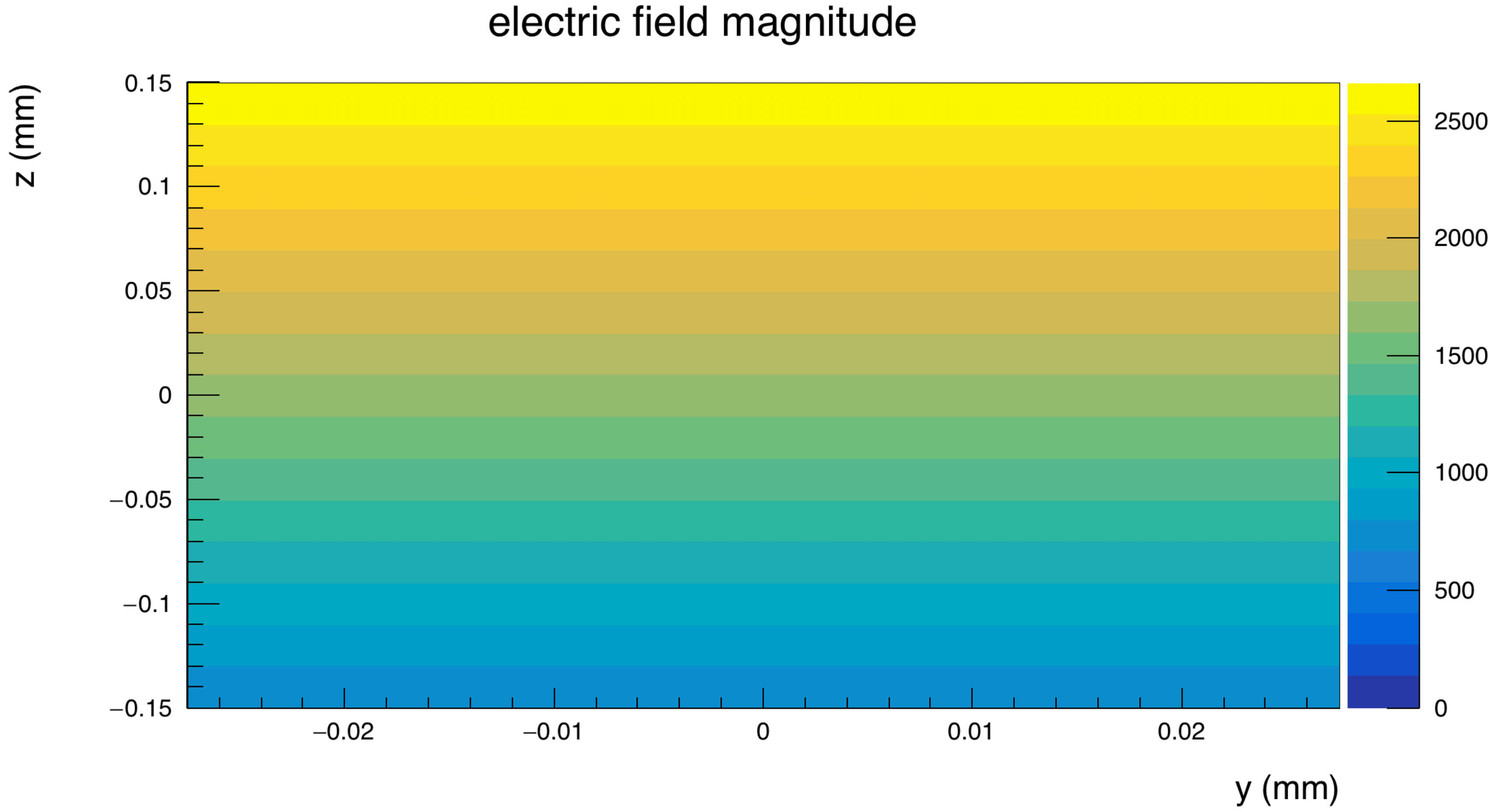
```
[Allpix]
number_of_events = 1000
detectors_file = "tutorial-geometry.conf"
log_level = "Warning"

[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
```

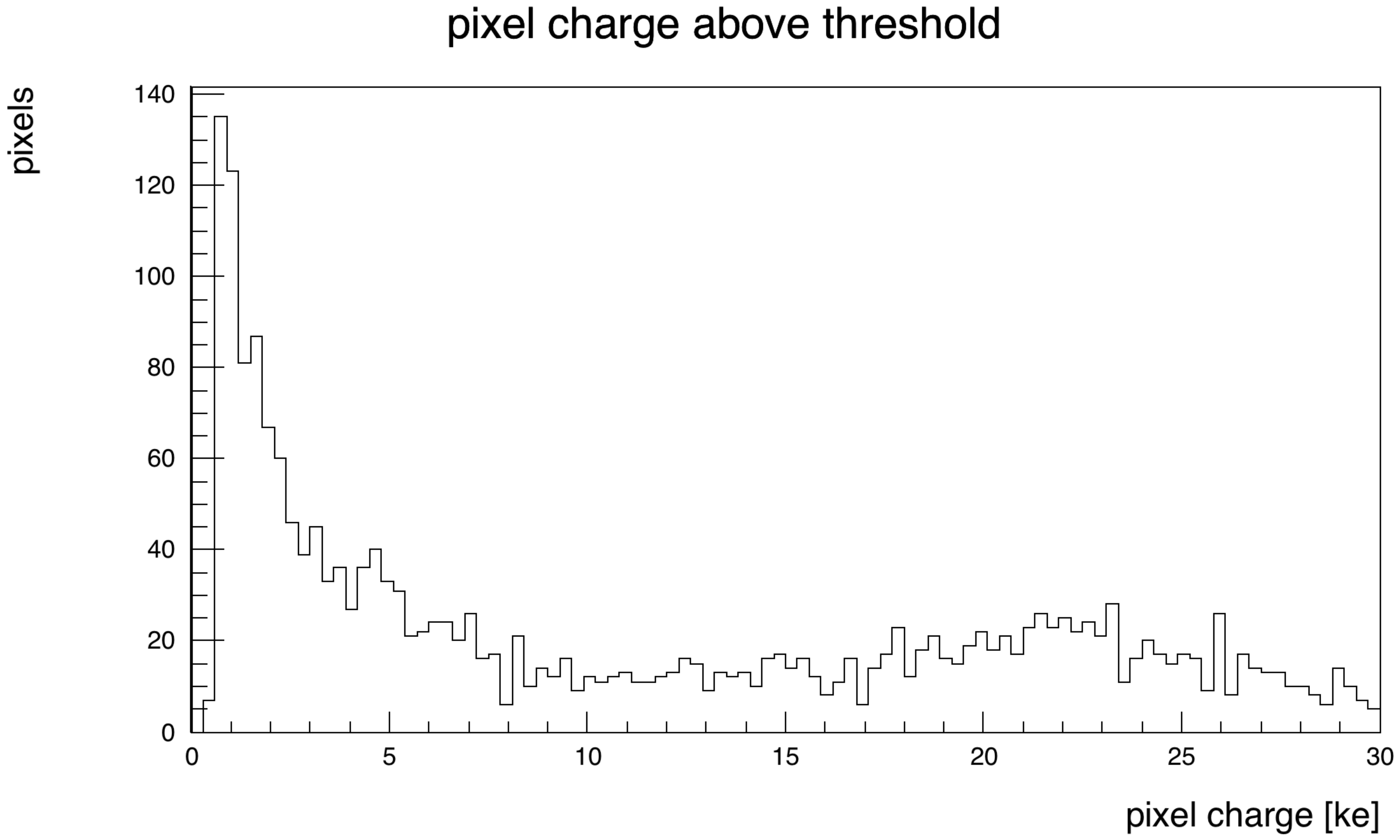

daniel.hynds@physics.ox.ac.uk

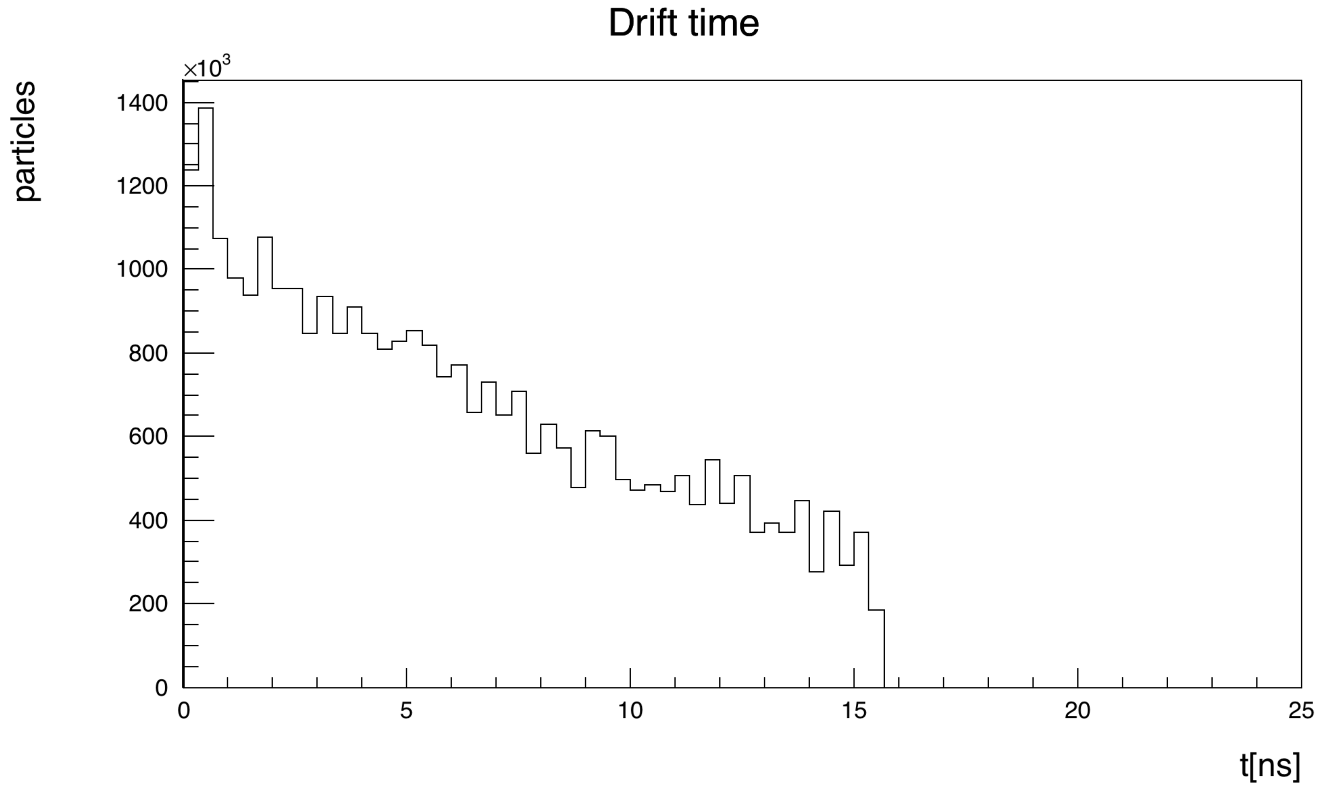
Example plots

./../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 co-ordinate 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]