IMPERIAL

LArSoft TPC Simulation

Fantastic FHICLs and where to find them

Anyssa Navrer-Agasson

a.navrer-agasson@imperial.ac.uk

10th UK LArSoft Workshop - October 2025

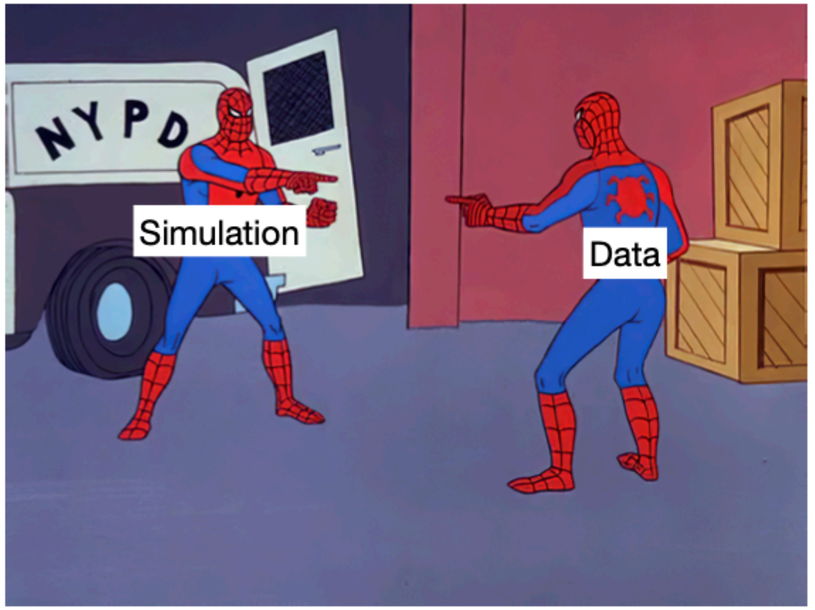
Goal of the lecture/tutorial

What will you (hopefully) know by the end?

- What are the steps needed to generate events?
- What are the different tools used for each step?
- How do different part of the simulation communicate?
- What is the output of each step?

What is LArSoft doing? (Simulation)

- Produce events that look like real data, but with “truth” information to check the behaviour of the reconstruction/analysis.
  - Output should have the same format and contain the same information as real data.
  - Simulation needs to be affected by the detector response.



How is LArSoft ~~(organised)~~?

- **One per module**
- **Configure your module**
  - List the services to use
  - Define parameter values

FHiCL file

Services

- **Provide information** about
  - Geometry
  - Physical properties
  - Physics simulation
- Accessed by modules.

- Implementation of **LArSoft algorithms**
  - Event generator, detector physics, reconstruction, …
  - Has setup, input, running and output phase
  - Uses LArSoft **data products**

Modules

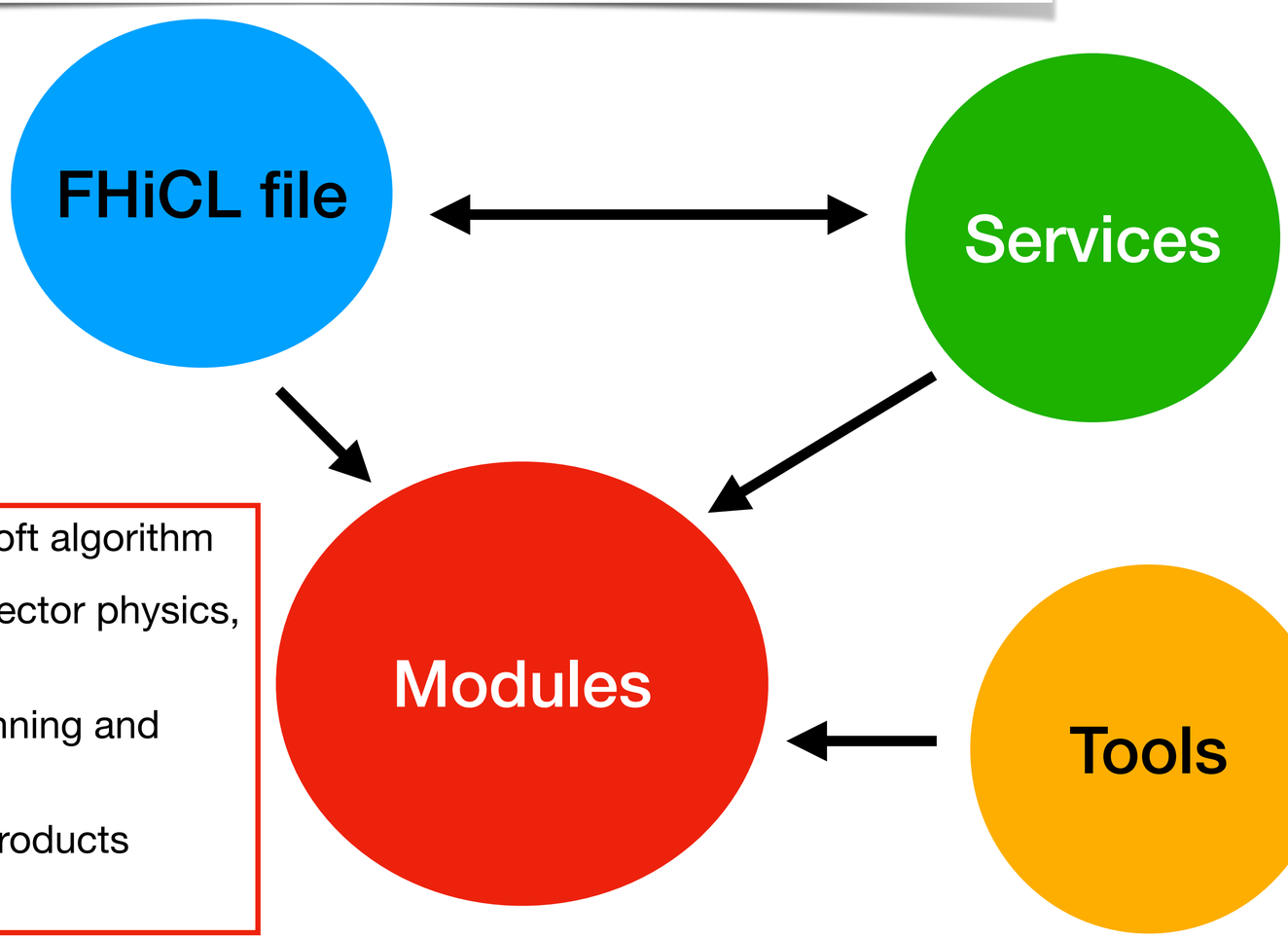
Tools

- Configurable, local utilities callable inside modules.

How is LArSoft ~~(organised)~~?

Most important thing in LArSoft: know the standard fhicl files and where to find them!!

- One per module
- Configure your module
  - List the services to use
  - Define parameter values



- Provide information about
  - Geometry
  - Physical properties
  - Physics simulation
- Accessed by modules.

- Implementation of LArSoft algorithm
  - Event generator, detector physics, reconstruction, …
  - Has setup, input, running and output phase
  - Uses LArSoft data products

- Configurable, local utilities callable inside modules.

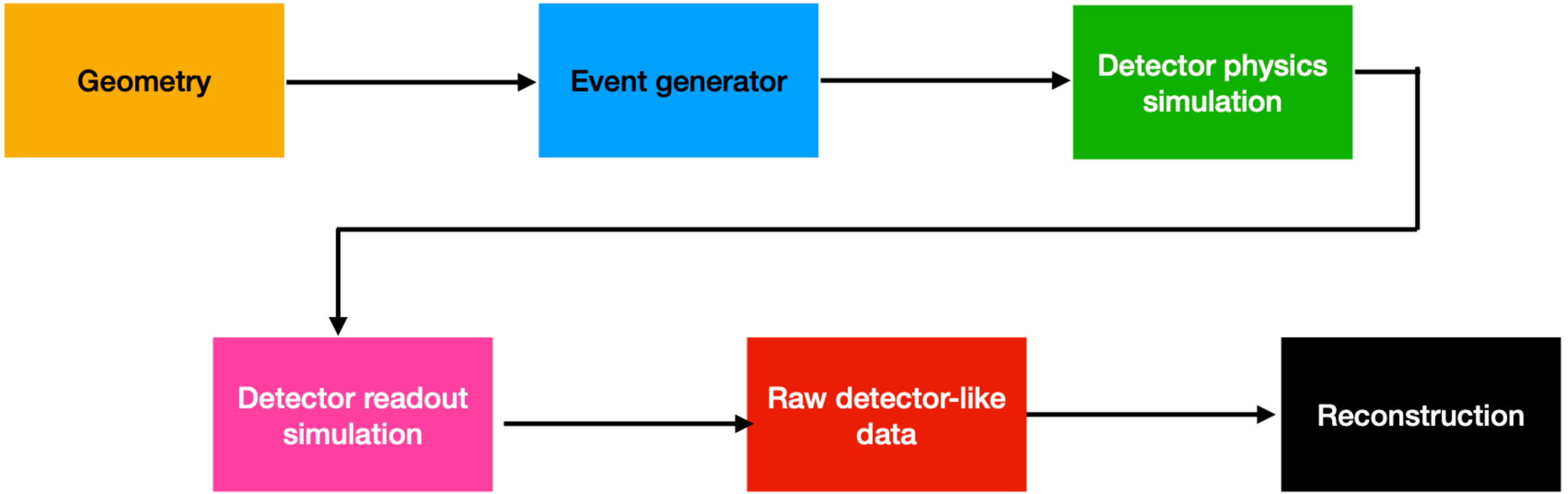
art, data products & *art*ROOT files

Not an acronym!

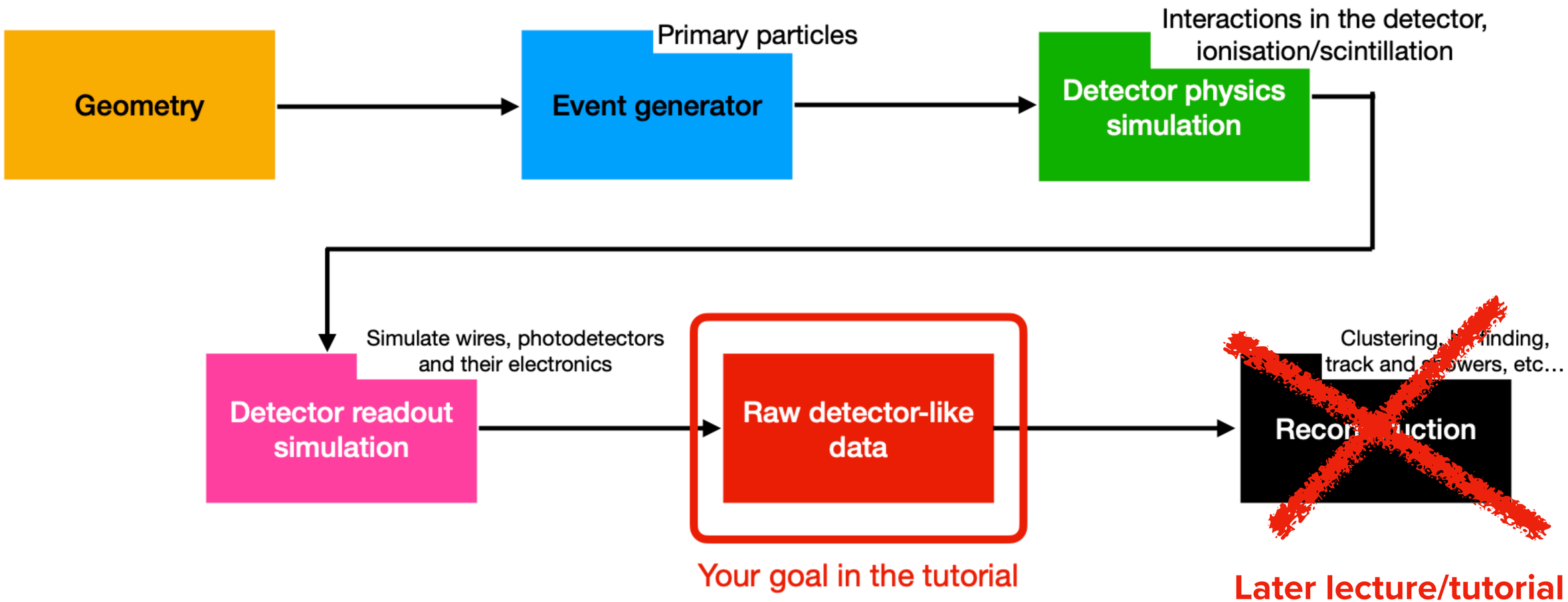
- LArSoft is based on the *art* framework developed by the Fermilab Scientific Computing division.
- ***Data products*** are units of information that modules may add to an event or retrieve from an event
- Output of the LArSoft modules (except analysers) are *art*ROOT files.
  - ***art*ROOT files are NOT usual ROOT files!**
  - Data products inside, not ntuples/trees

Click to access more info on art!

LArSoft simulation flowchart

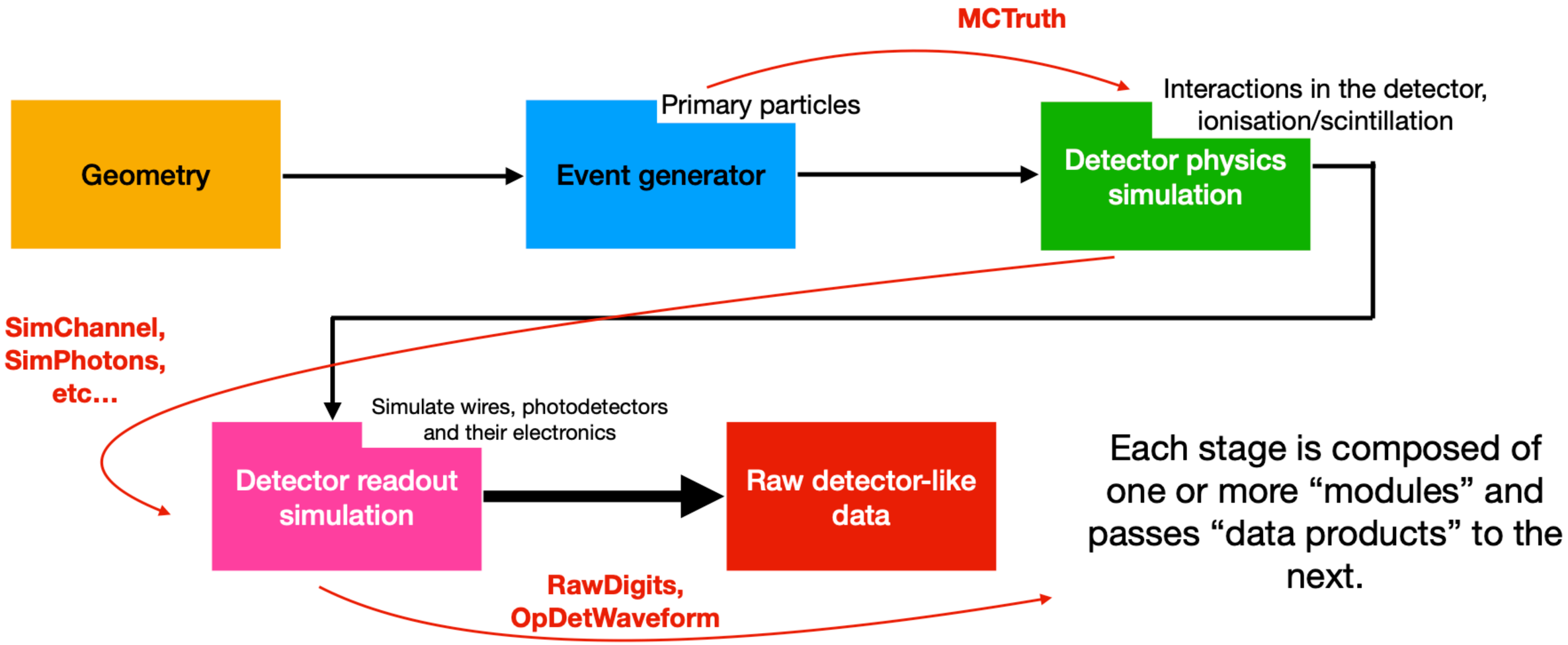


LArSoft simulation flowchart



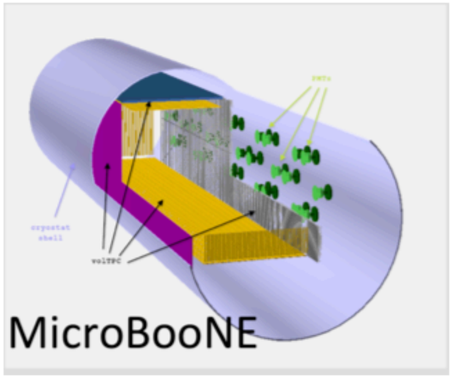
LArSoft simulation flowchart

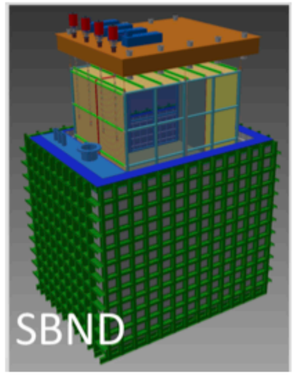
Data products: classes saved in the output artROOT file

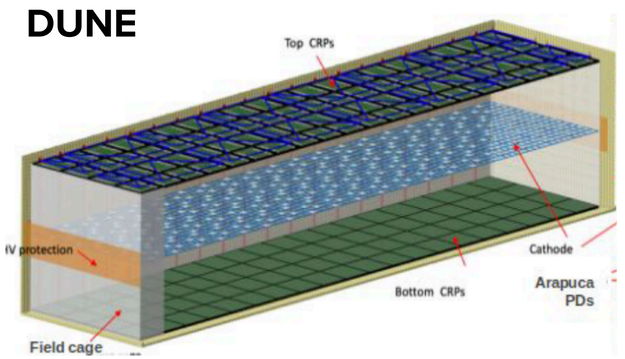


Each stage is composed of one or more “modules” and passes “data products” to the next.

Step 1: Build-A-Detector



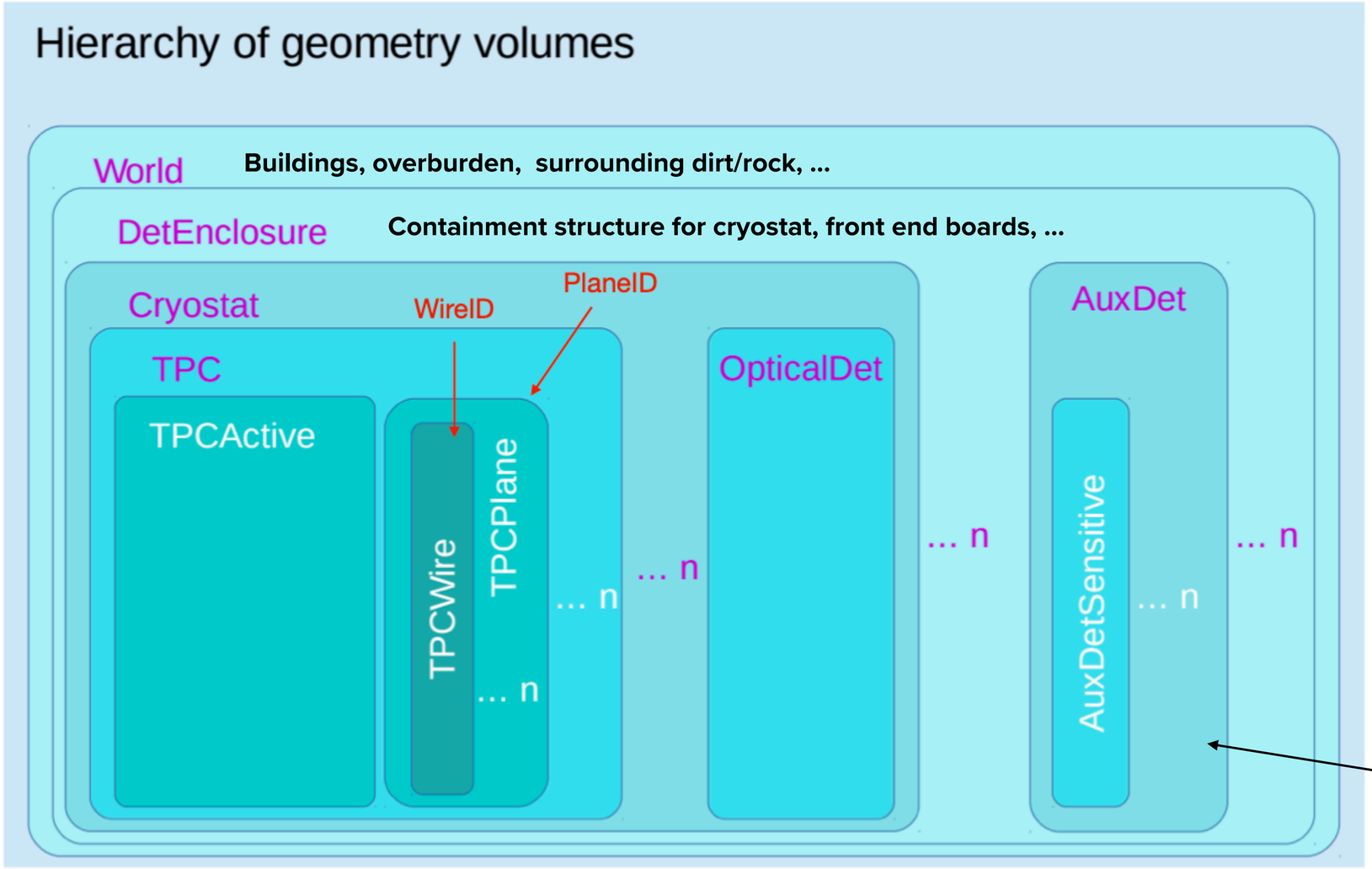




and more!

- Core LArSoft software is common but each experiment needs to define its own geometry
- Simulation/reconstruction knows how to access the different geometries
- Geometries are stored in GDML (Geometry Description Markup Language) files
  - SBND geometry files can be found in `sbndcode/Geometry/gdml`
  - DUNE geometry files can be found in `dunecore/Geometry/gdml`
  - Also contains Perl scripts to make the GDML files

Geometries in LArSoft

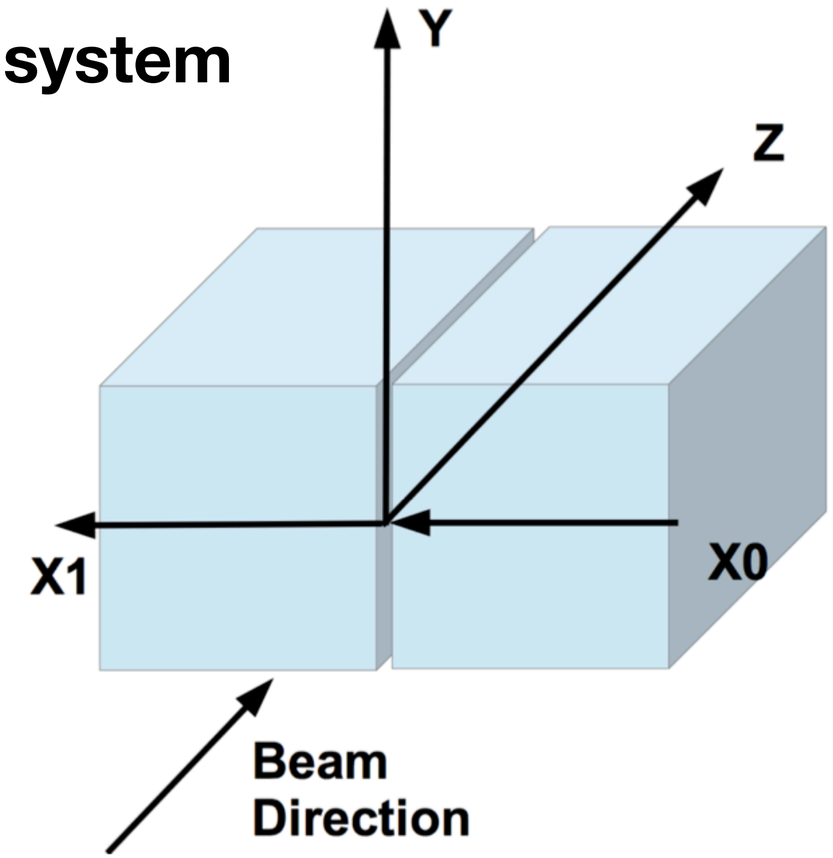


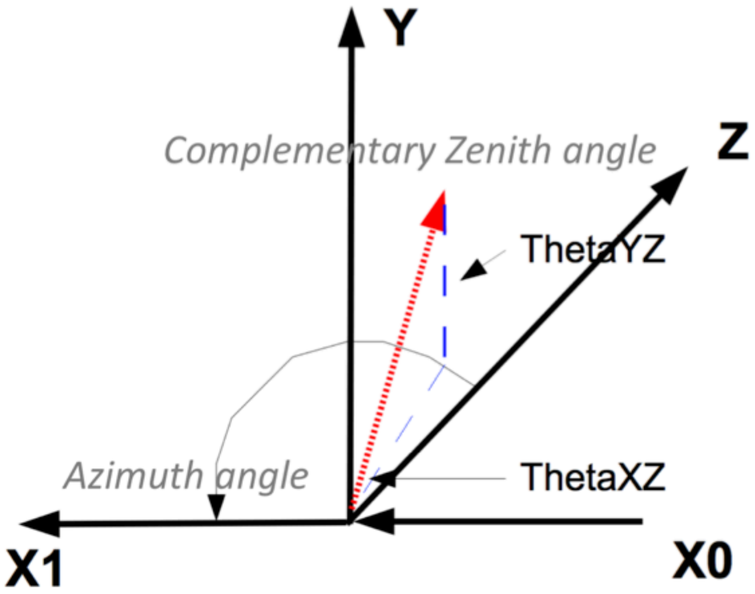
- Use ID objects to specify which instance of TPC geometry objects you want
- There are sorting algorithms in place that determine which one goes first in the code

e. g. Cosmic Ray Tagger

Step 1: Build-A-Detector

Coordinate system



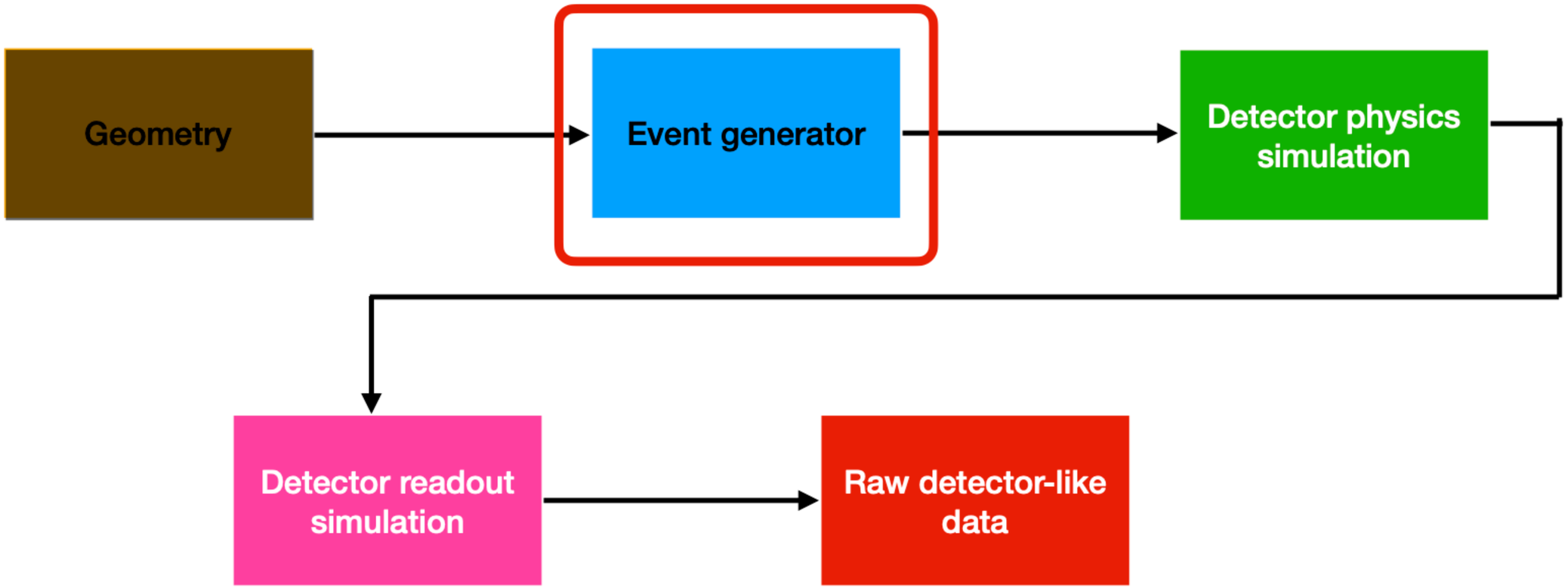


For all detectors: Z increases in the direction of neutrino travel, Y increases away from the centre of the Earth and X increases so as to make a right-handed coordinate system.

Origin is experiment-specific

Step 2: Let there be particles!

Now that you have a detector, you can generate some particles!

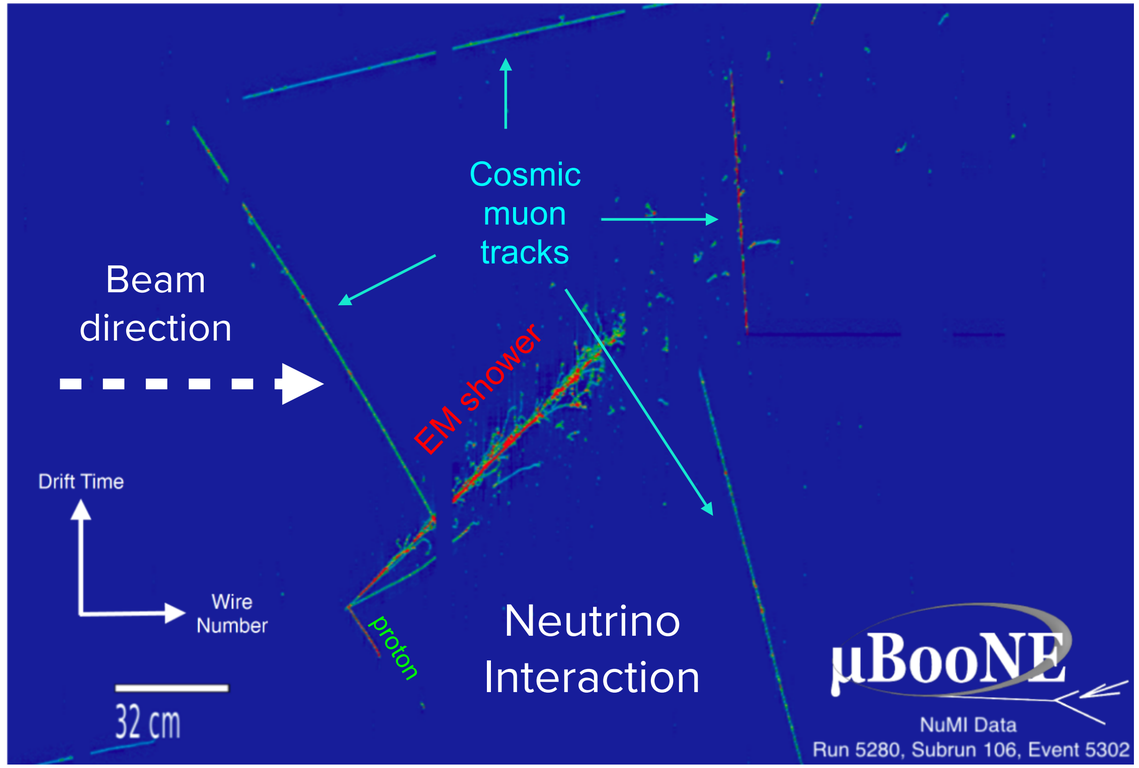


Step 2: Let there be particles!

Where we create particles from nothingness

- Actual TPC events are made of two main elements:
  - Neutrino interactions (sometimes)
  - Cosmic muon tracks

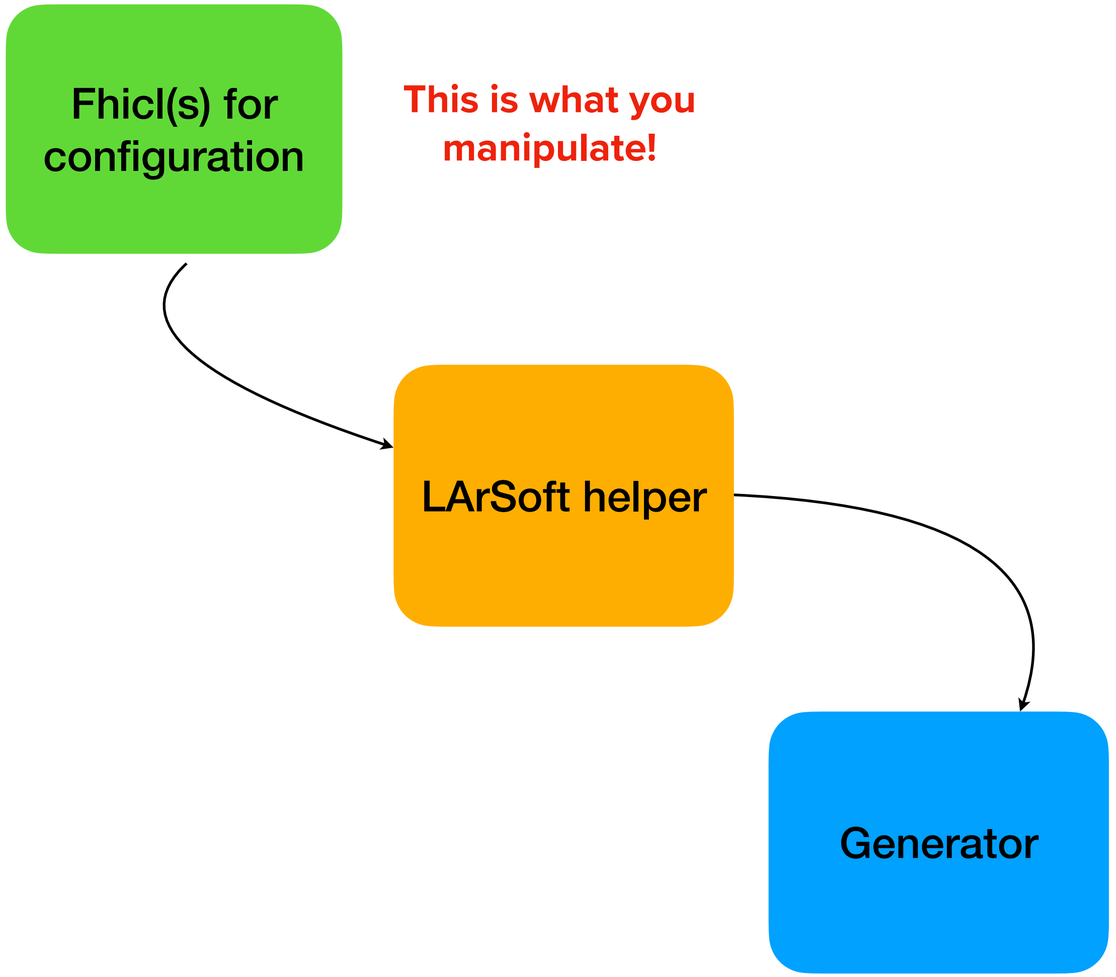
- Simulation has to be able to generate many types of particles and interactions
- Relies on more than one generator
- **Let's explore!**



Event Generators in LArSoft

- Most event generators are not native LArSoft tools
  - Have to be interfaced with the general framework
  - Usually via a helper module to run the external generator/exploit flux files
  - Helpers also fill the MCFlux/MCTruth data products

- Helpers live in larsim/EventGenerator



Before we start, some useful information

Click for more information about PDG codes

- PDG (Particle Data Group) defined a set of number/particle associations widely in particle physics simulations
- Particles lists in the simulation will be lists of PDG codes
- Example for leptons

LEPTONS	
e^-	11
ν_e	12
μ^-	13
ν_μ	14
τ^-	15
ν_τ	16
τ'^-	17
$\nu_{\tau'}$	18

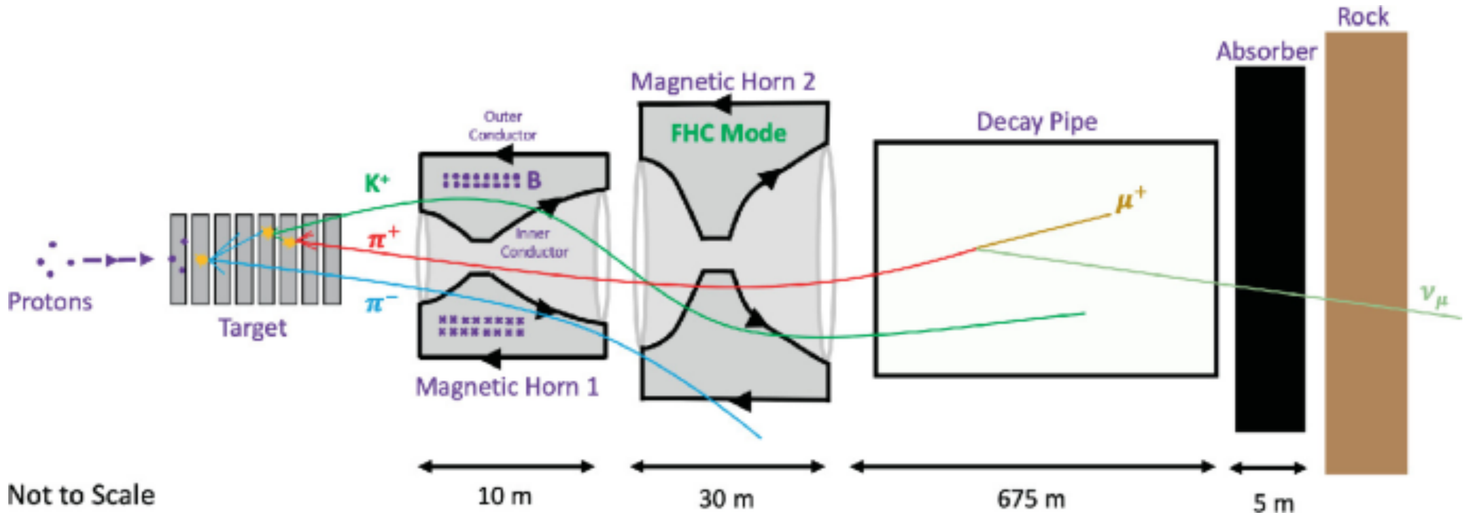
Click for more information about Geant4

- Geant4 is the toolkit to simulate the passage of particles through matter
- Widely used in high-energy physics
  - Beamline simulations
  - Particle propagation in LArSoft
- Includes a complete list of physics processes across energy ranges (eV to TeV)

The real beginning: beam line simulations

- The real start of the simulation isn't the neutrino interaction
- Neutrino beams are complicated environments, requiring their own simulation packages
- Usually Geant4-based (G4BNB, G4NuMI)
- Output flux files at different detector locations

NuMI beam line

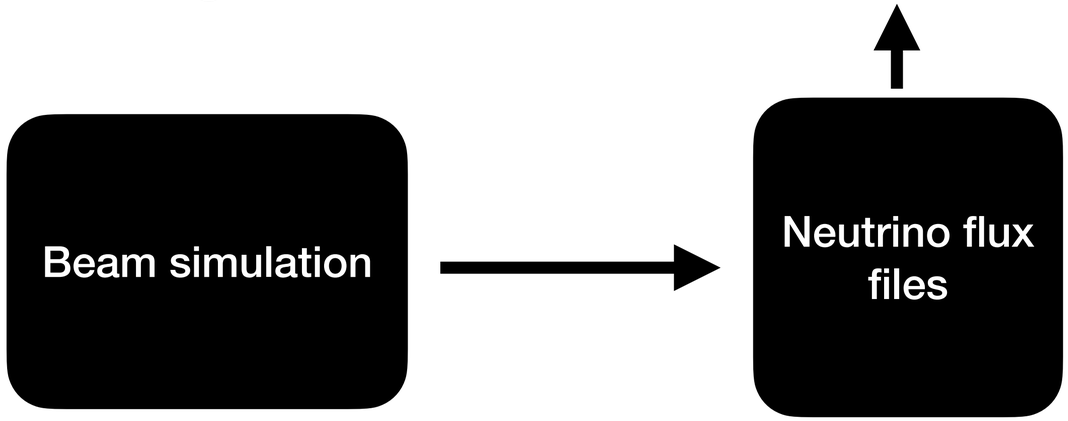


GENIE

- GENIE is the most popular neutrino interactions generator
- Takes as input the neutrino flux corresponding to your beam and detector
- Produces neutrino secondaries using accurate physics modelling
  - You can specify the type(s) of interaction you want to consider (CCQE, MEC, DIS, ...)
- GENIE also keeps track of the exposure (in POT) for the generated sample

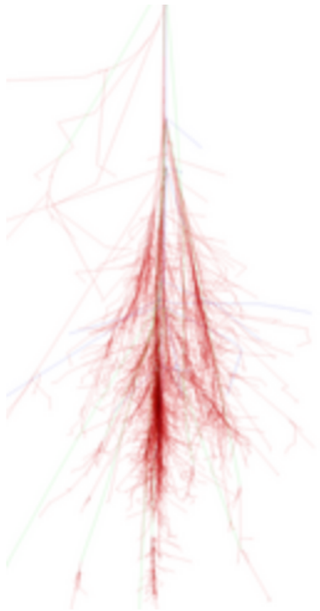
(Protons on Target)

- GENIE can also generate non-neutrino (BSM) interactions



CORSIKA & friends

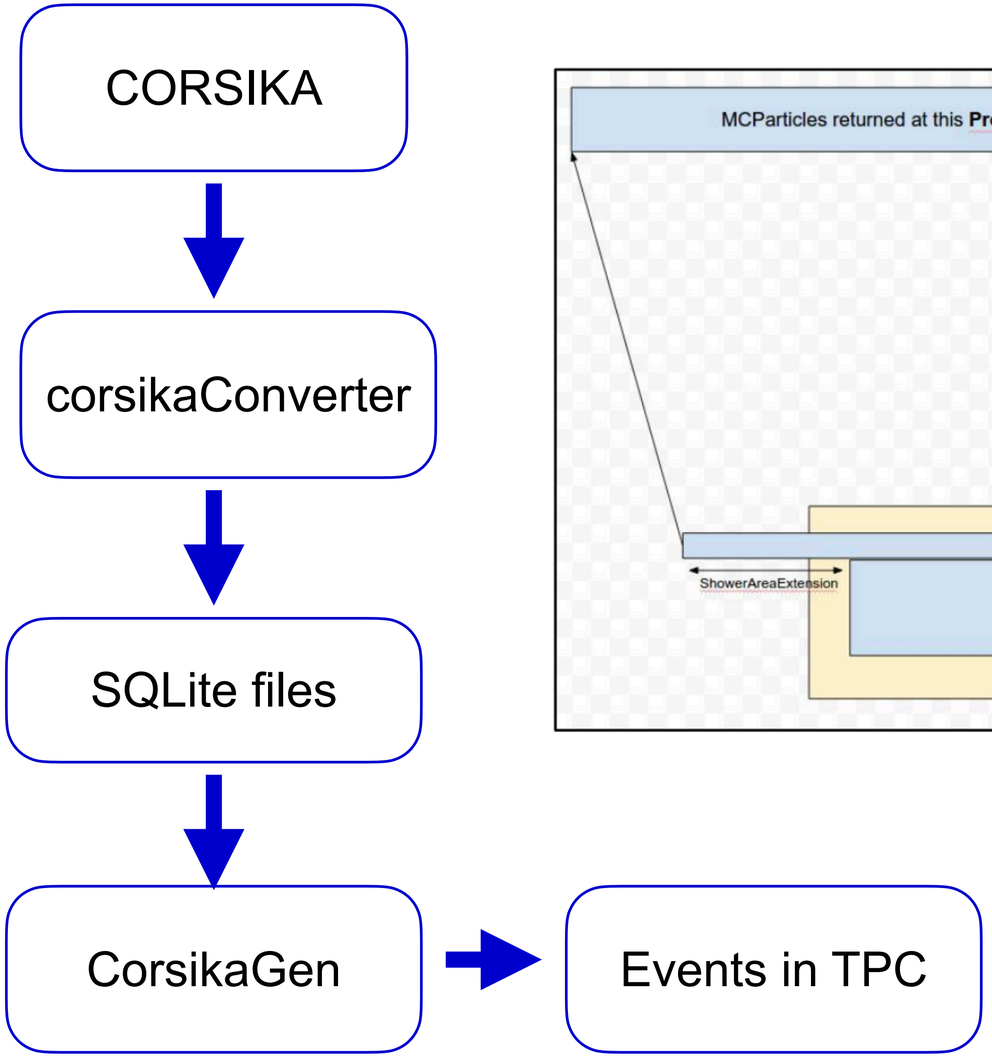
CORSIKA

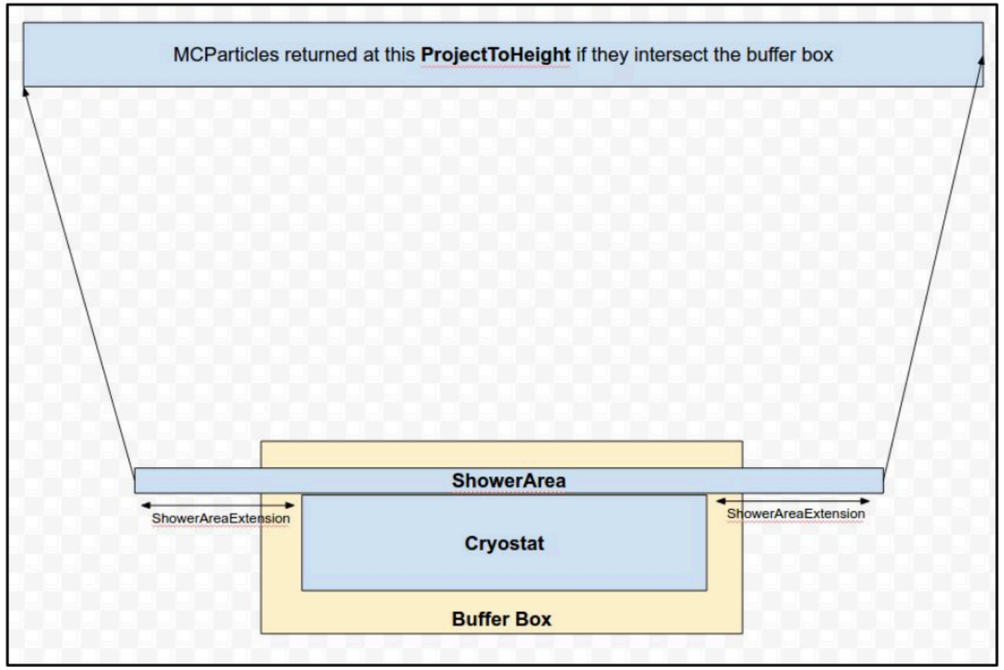


Observation level (228 m for Fermilab)

- CORSIKA generates **extended air showers**
- Can configure the primary particle
- User sets up an observation level
- **Particles reaching this level are stored**
- Output is stored in binary files

CORSIKA & friends



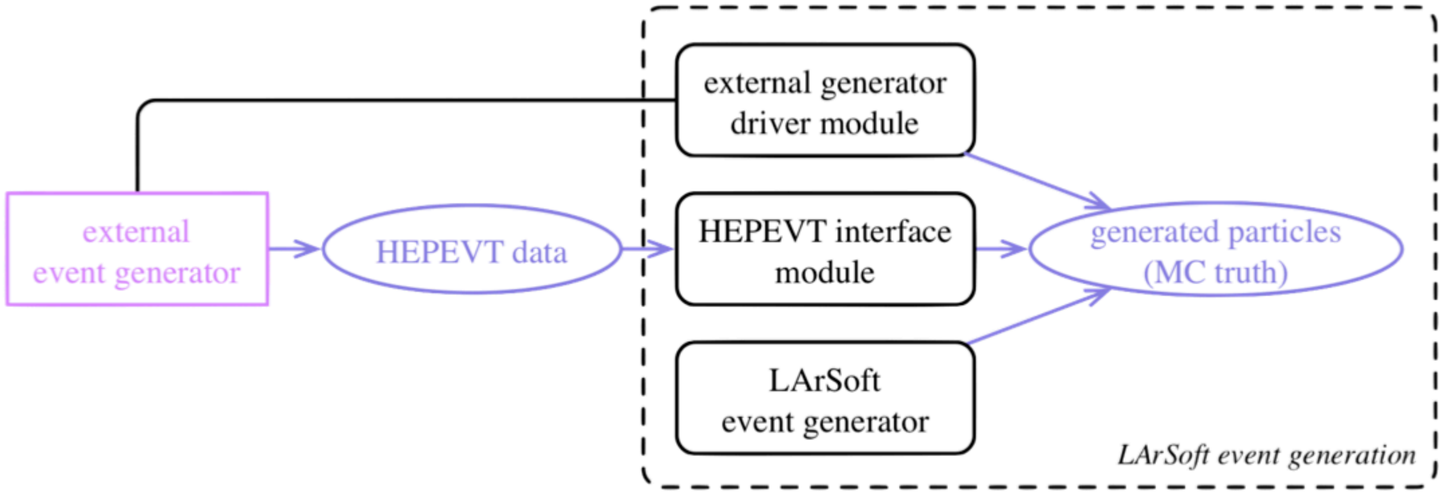


- **LArSoft** module **CorsikaGen** handles the creation of detector events
- **Assembles events** by randomly sampling from the showers table and arrange particles in space/time using:

$$I_N(E) \approx 1.8 \times 10^4 \, (E/1\ \mathrm{GeV})^{-\alpha} \, \frac{\mathrm{nucleons}}{\mathrm{m}^2\ \mathrm{s}\ \mathrm{sr}\ \mathrm{GeV}}$$

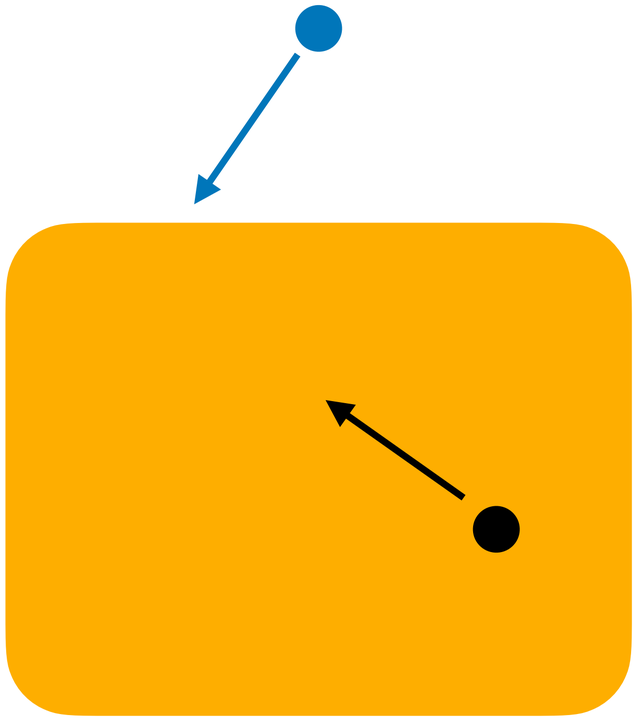
Event generators: TextFileGen

- To use every time a generator isn't interfaced with LArSoft (#BSM)
- Can generate primary particles from a file containing a list of PDG codes, position, momentum & energies
- Takes HEPEVT files as input
- Can be tricky to use



Simplest: Single Particle Gun

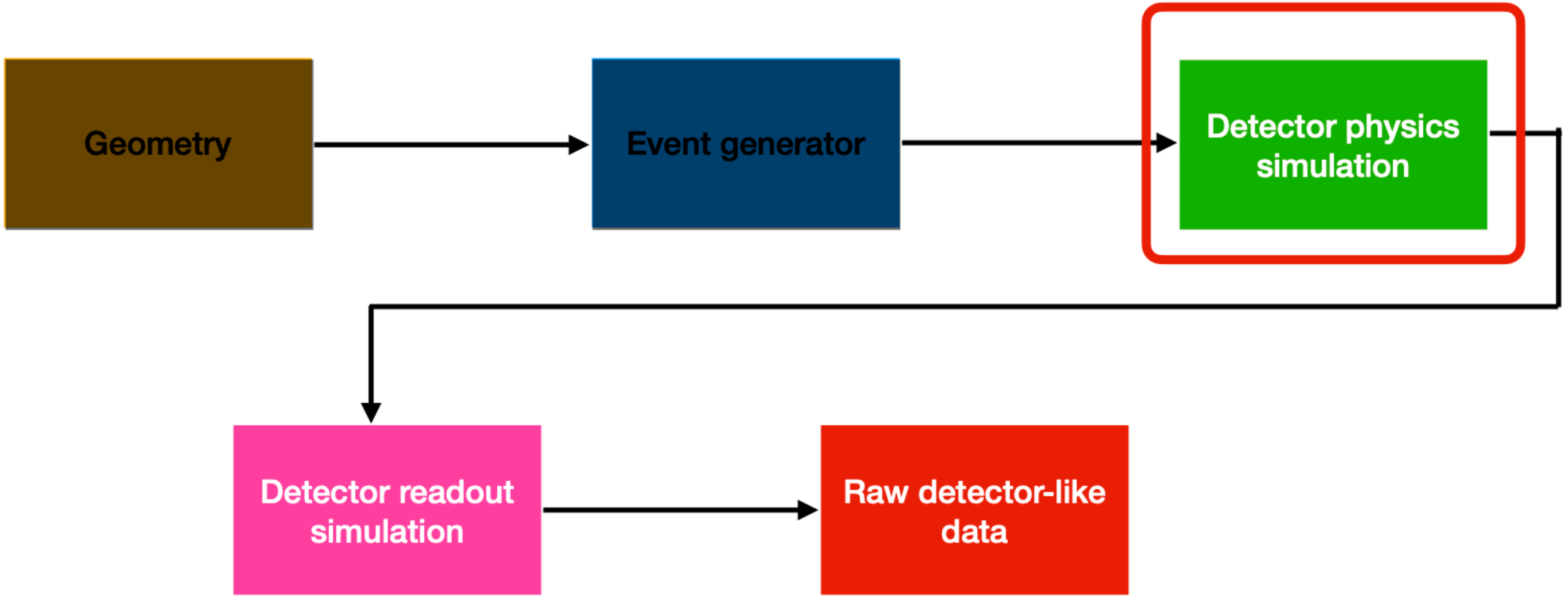
- Simplest event generator
- Used to generate individual particles or very simple interactions
- You can define particle type (PDG code), position, momentum and the way they vary (uniform, gaussian)
- There is an option to generate multiple particles:
  - Randomly pick one from a list at each event
  - Generate all particles within the same event
- **This is what you will use in the tutorial!**



What's in your output file? (1)

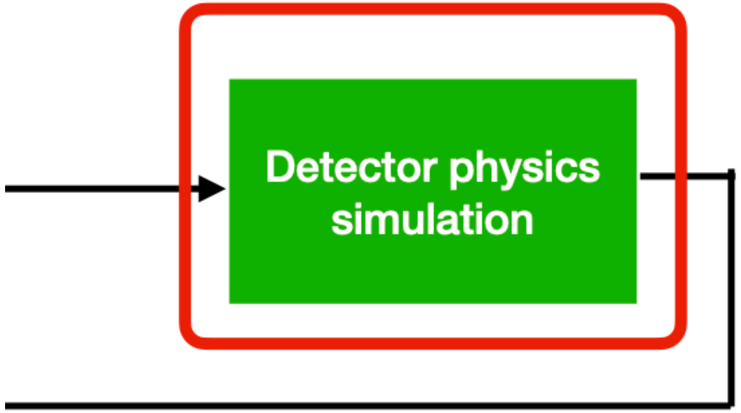
- `simb::MCTruth` objects (usually one per generator used) which will be picked up by the next stage
- Contains:
  - Information about the generator
  - List of particles (`simb::MCParticle`) with PDG code, position, momentum, ...
  - Information about the neutrino interaction (if any)

Step 3: the tribulations of particles in LAr



Step 3: the tribulations of particles in LAr

- Interactions of the primary particles with the detector
- Handles energy depositions and creation of secondary particles

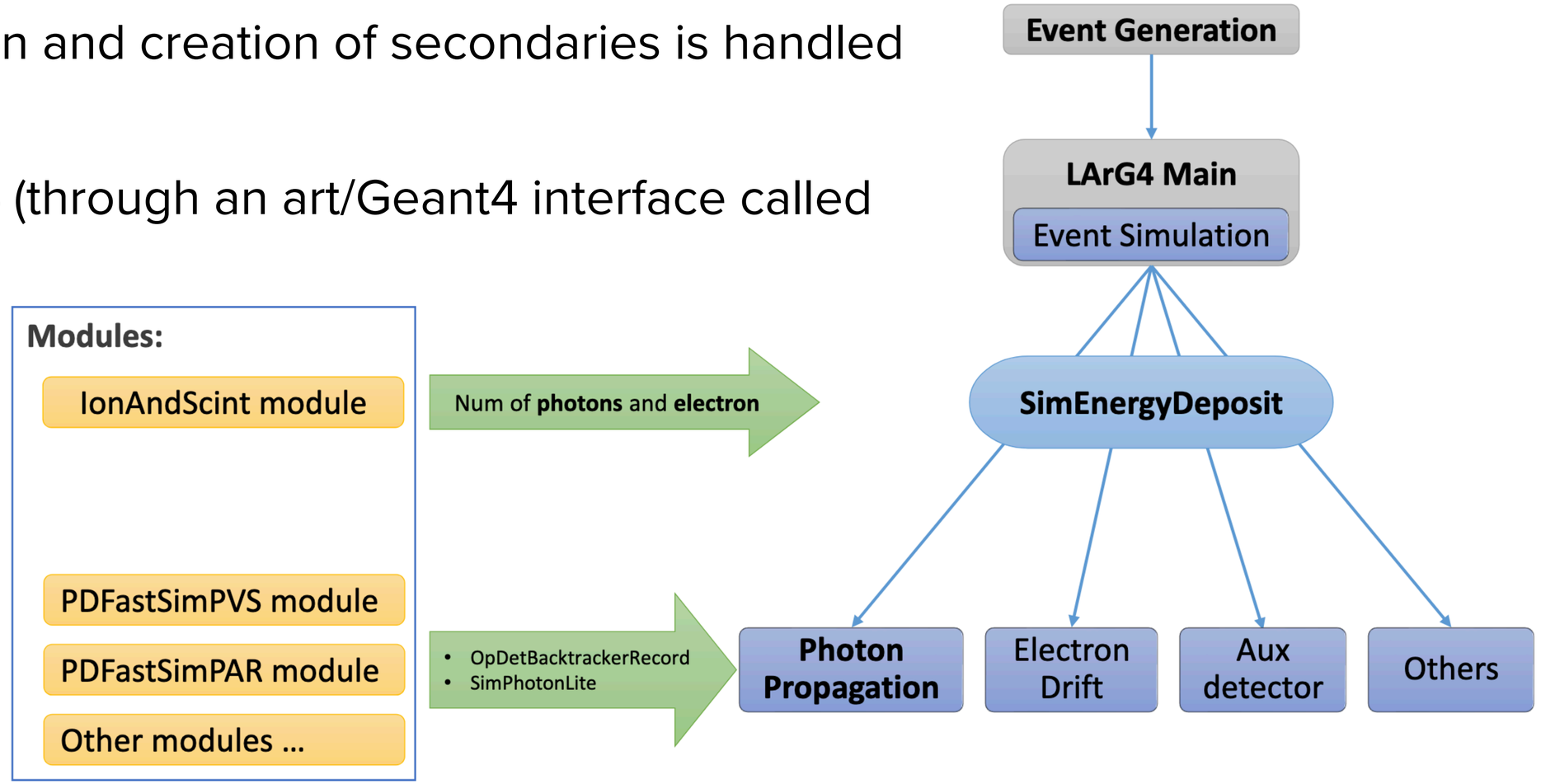


Parameters for simulation can be found in `larsim/simulation/simulationservices.fcl`

Step 3: the tribulations of particles in LAr

Where we make particles interact and see what comes out

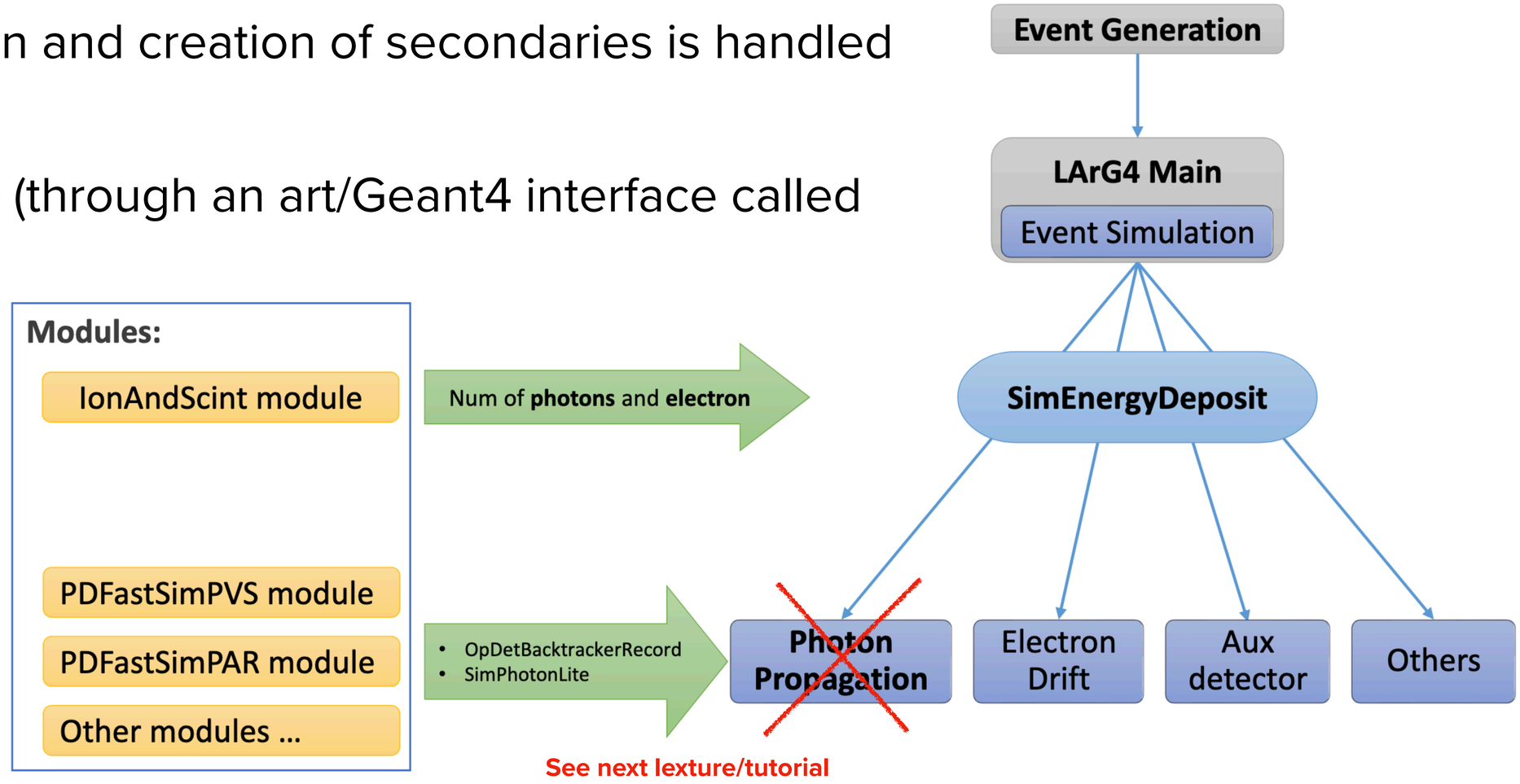
- Particle propagation and creation of secondaries is handled by LArG4
  - Relies on Geant4 (through an art/Geant4 interface called artg4tk)



Step 3: the tribulations of particles in LAr

Where we make particles interact and see what comes out

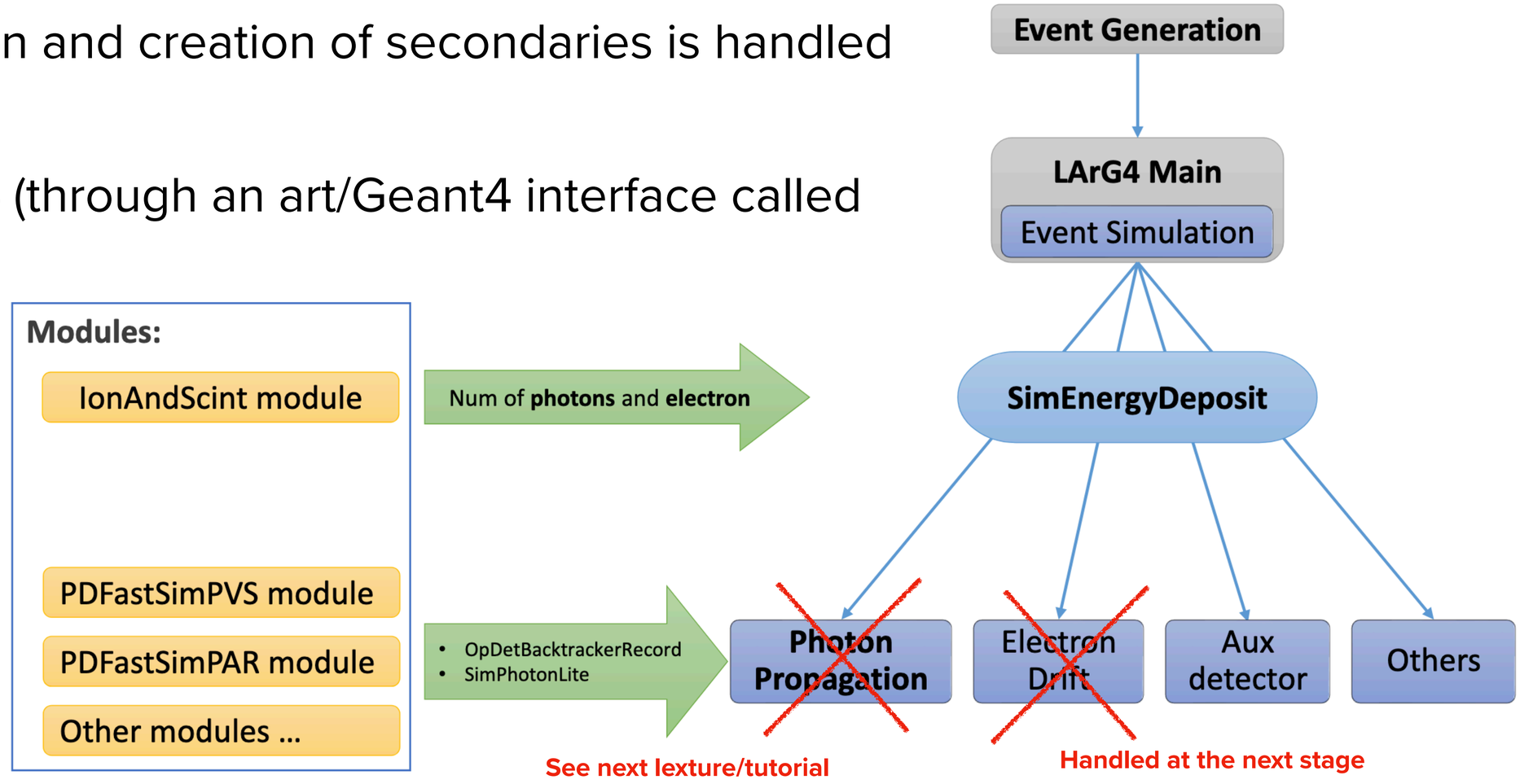
- Particle propagation and creation of secondaries is handled by LArG4
  - Relies on Geant4 (through an art/Geant4 interface called artg4tk)



Step 3: the tribulations of particles in LAr

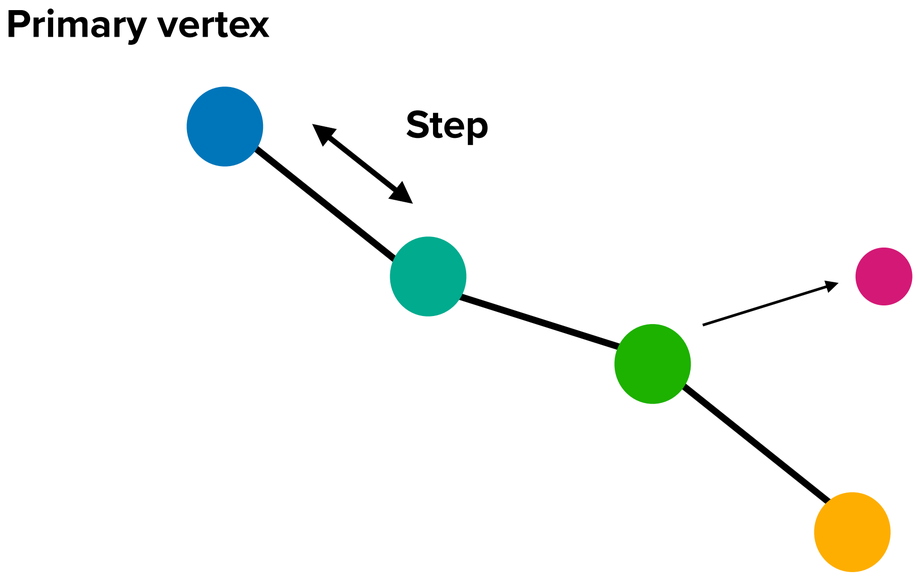
Where we make particles interact and see what comes out

- Particle propagation and creation of secondaries is handled by LArG4
  - Relies on Geant4 (through an art/Geant4 interface called artg4tk)



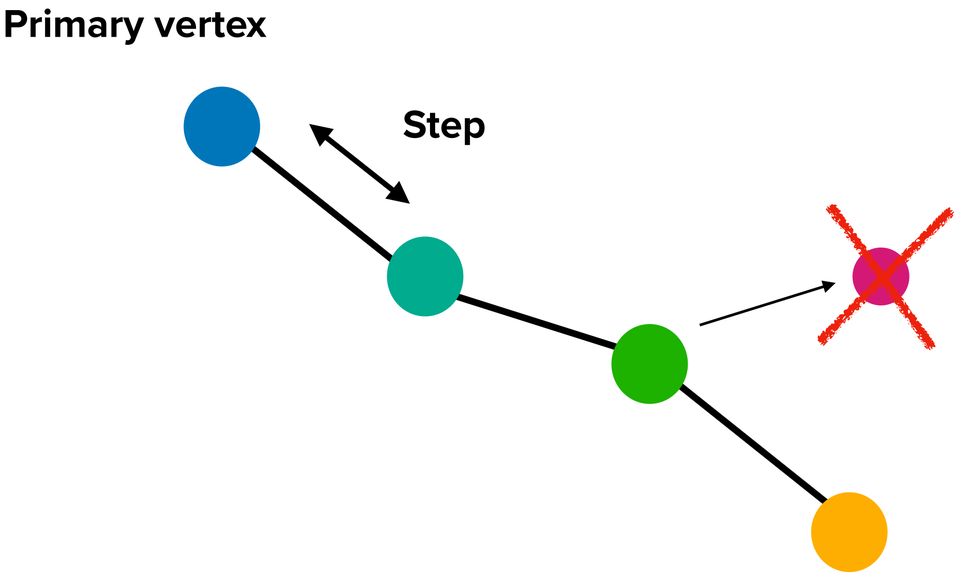
Step 3: the tribulations of particles in LAr

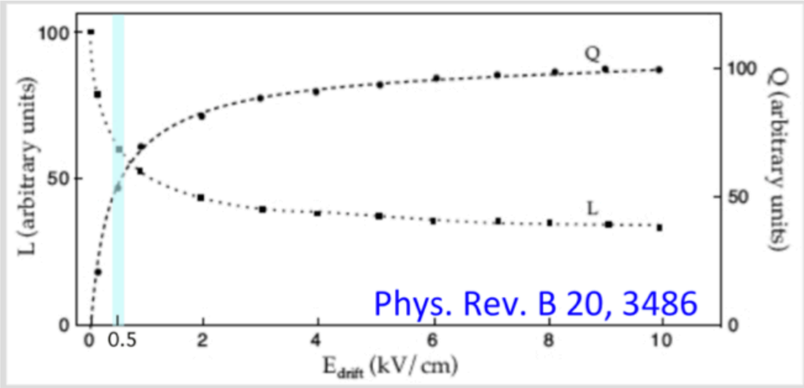
- **Loop through the particles in MCTruth**
  - **For each particle:**
    - Generate primary interaction vertex
    - Track particle through a series of steps
      - **For each step:**
        - Evaluate energy deposition
        - Determine physics process happening
        - Add secondary particles created (if any) to the particle list



Step 3: the tribulations of particles in LAr

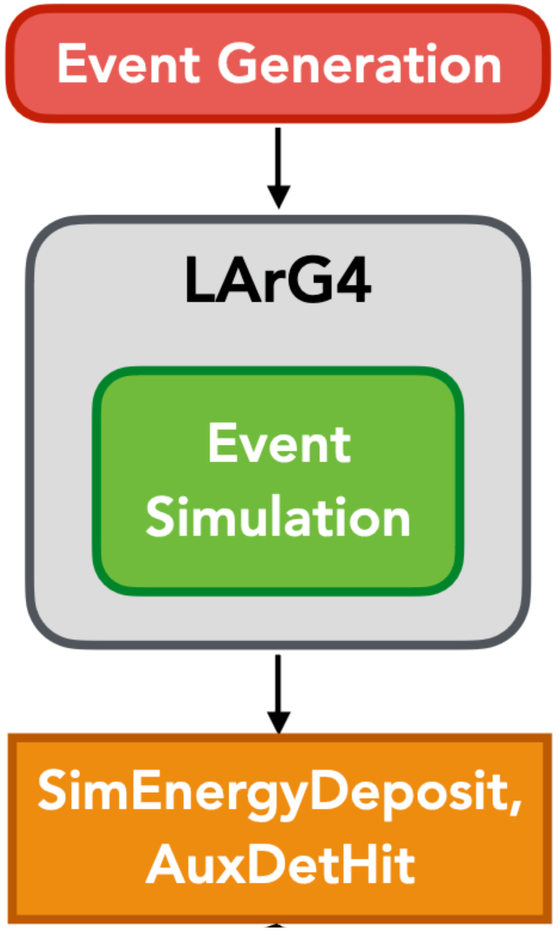
- The simulation is configured via fhicl
  - Physics list (which processes to consider)
  - Additional physics parameters
  - Which particles to track
  - …
- Possibility to impose cuts on particles type, energy, …
- Energy deposition then fed to other modules to compute scintillation & ionisation yields





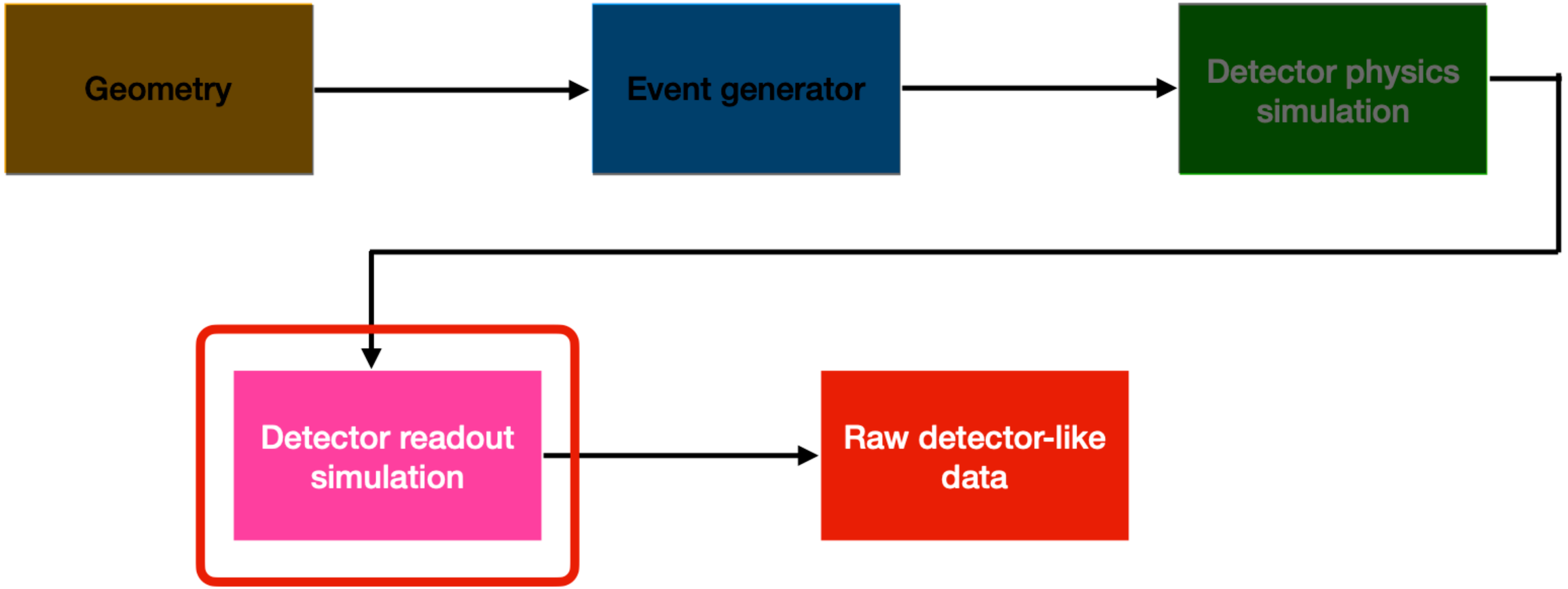
What's in your output file? (2)

Illustration credit: Marco Del Tutto



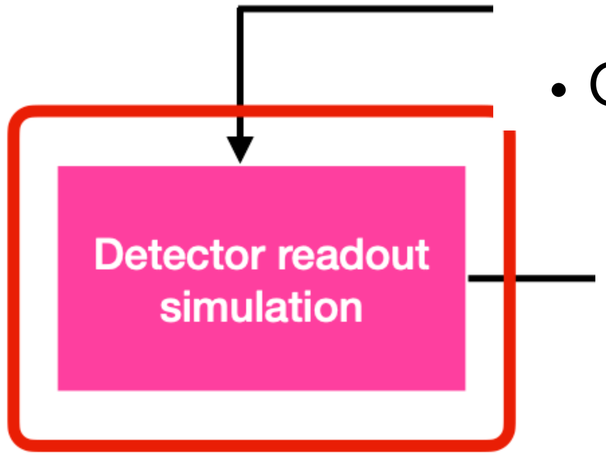
- `simb::MCTruth` object from the previous stage
- New collection of `simb::MCParticle` containing secondary particles created at this stage
- Collections of `sim::SimEnergyDeposit` containing the energy depositions
  - Information on the position, scintillation/ionisation yield of the deposition
- A `sim::ParticleAncestryMap` containing the particle ancestry
- Associations between MCTruth, MCParticle and GeneratedParticleInfo
- Collections of MCTrack and MCShower objects from truth-level reconstruct
- Photon-related stuff you'll learn about tomorrow

Step 4: make some noise!



Step 4: make some noise!

- Drifts electrons to the wire planes
- Transforms the physics information (electrons) into digitised detector response
- Includes the simulation of signal shaping and electronic noise
- Output is detector-like data



Look for detsimmodules_sbnd.fcl

Step 4: make some noise!

Wire-Cell

Electron drift and detector response is handled by the Wire-Cell Toolkit

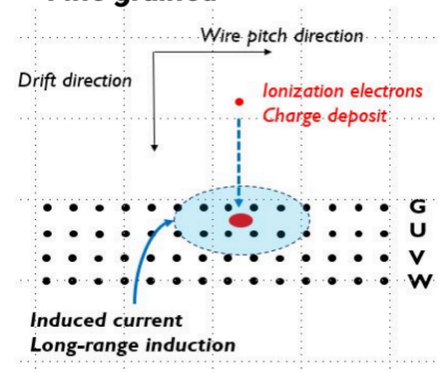
1) Drift and diffusion

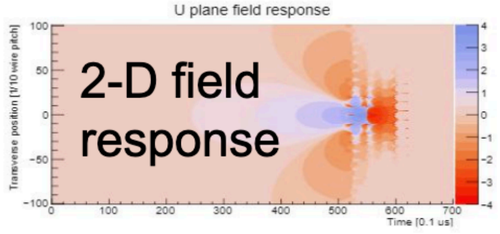
<Energy deposit>
* x, y, z, t, # of e

- Ionized electron absorption (lifetime in LAr)
- Gaussian random diffusion (longitudinal/transverse) $\sigma^2 = 2Dt$
- Fluctuation in electron absorption

2) Field response convolution:

- ***Long range effects***
- ***Fine grained***





3) "Raw" waveform

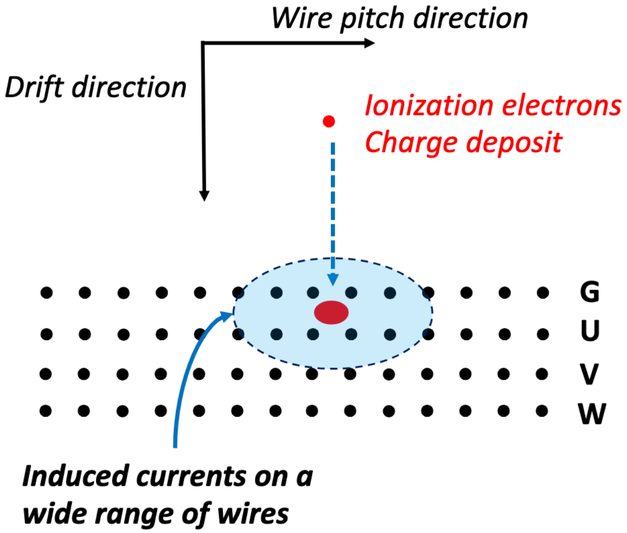
- Electronics response
- Preamp shaping
- AC coupling
- Noise
- Digitizer

The digital response is simulated as a convolution of the electron distribution, field response and electronics response.

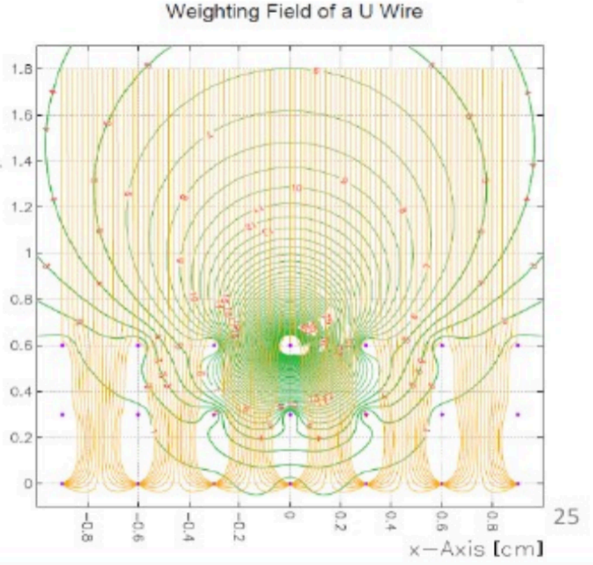
Step 4: make some noise!

Electron drift & field response

- Handles the ‘drift’ of ionisation electrons to the wire planes
- Computes the induced current according to an energy deposition
- Takes into account long-range effects and construct a field response file (pre-calculated 2D Garfield)
- Accounts for diffusion, electron lifetime, …



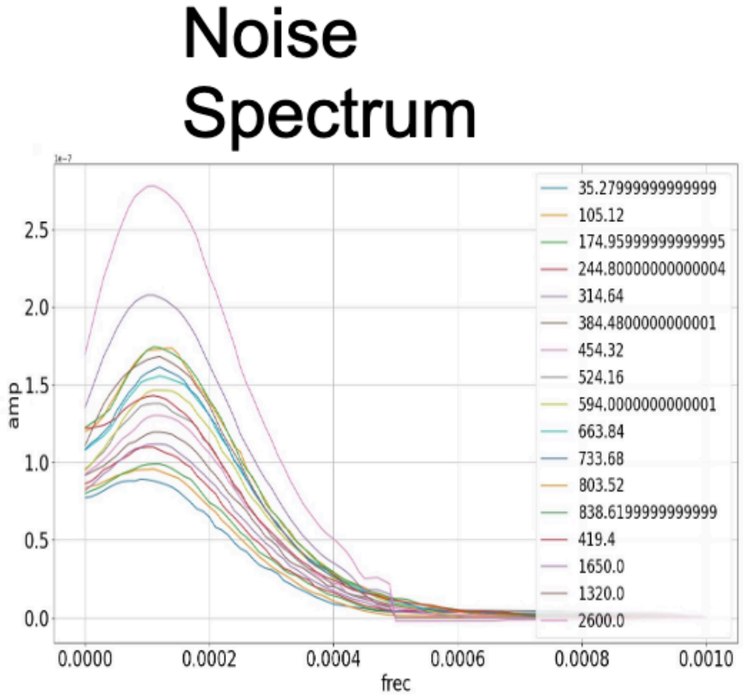
2D GARFIELD simulation : The field extends beyond a single wire region



Step 4: make some noise!

Noise

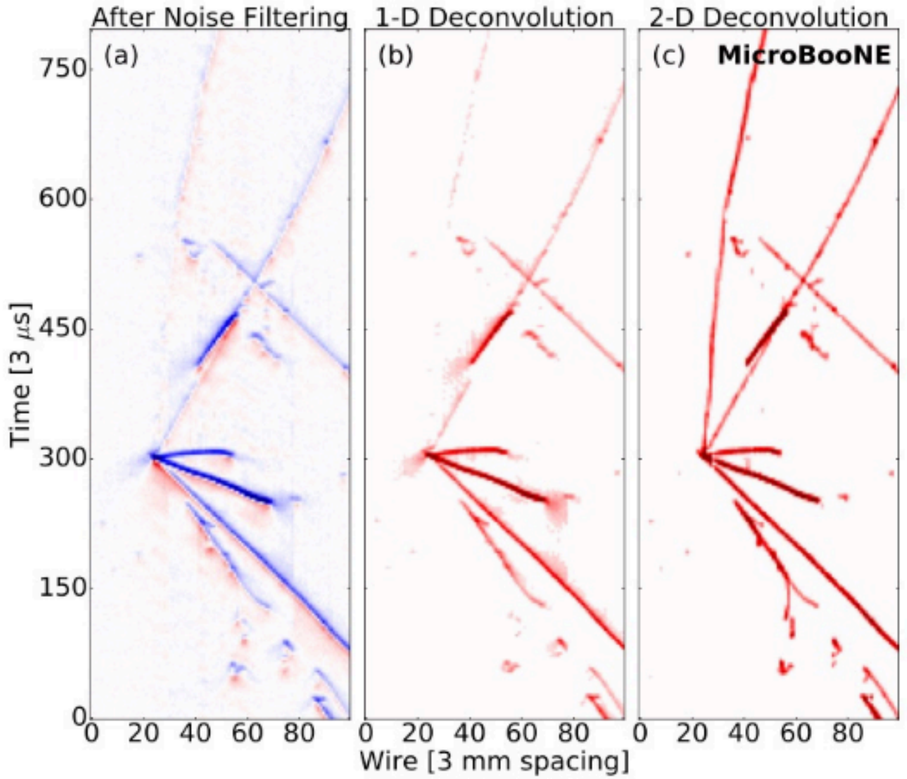
- Wire-Cell provides a “sampled spectrum noise model” which is parametrised by two types of information:
  - A set of “mean noise spectra”.
  - An associative mapping between channel and spectrum.
- The WCT implementation is structured into two layers
  - A “model” component provides a mean noise spectrum given a key (channel or group).
  - An “adder” component generates a noise waveform and associates it to one or more channels.



Step 4: make some noise!

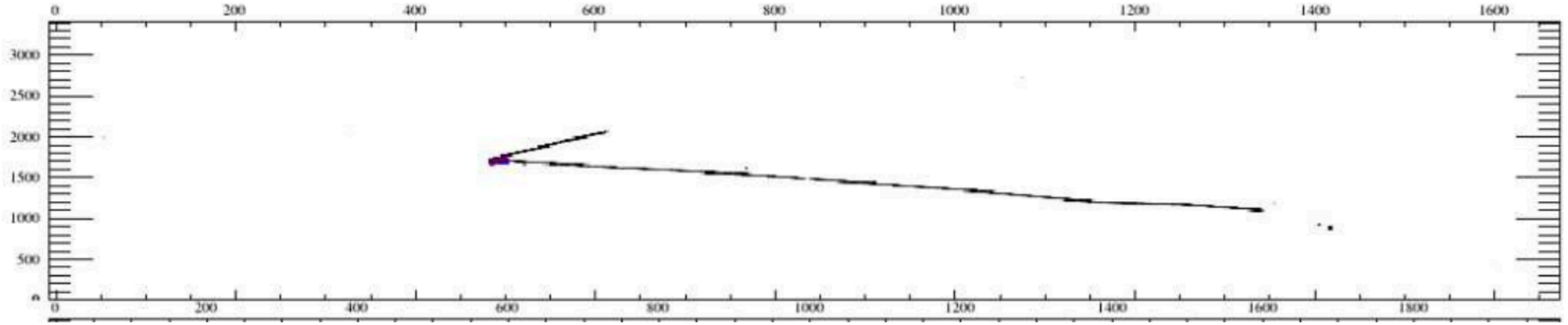
Signal processing

- Now that we have the raw data, we want to remove the field and electronics response we just added and recover the ionisation charge
- Wire-Cell also provides signal processing to convert raw signal to arriving charge vs. time
  - Noise filtering
  - Signal deconvolution



What’s in your output file? (3)

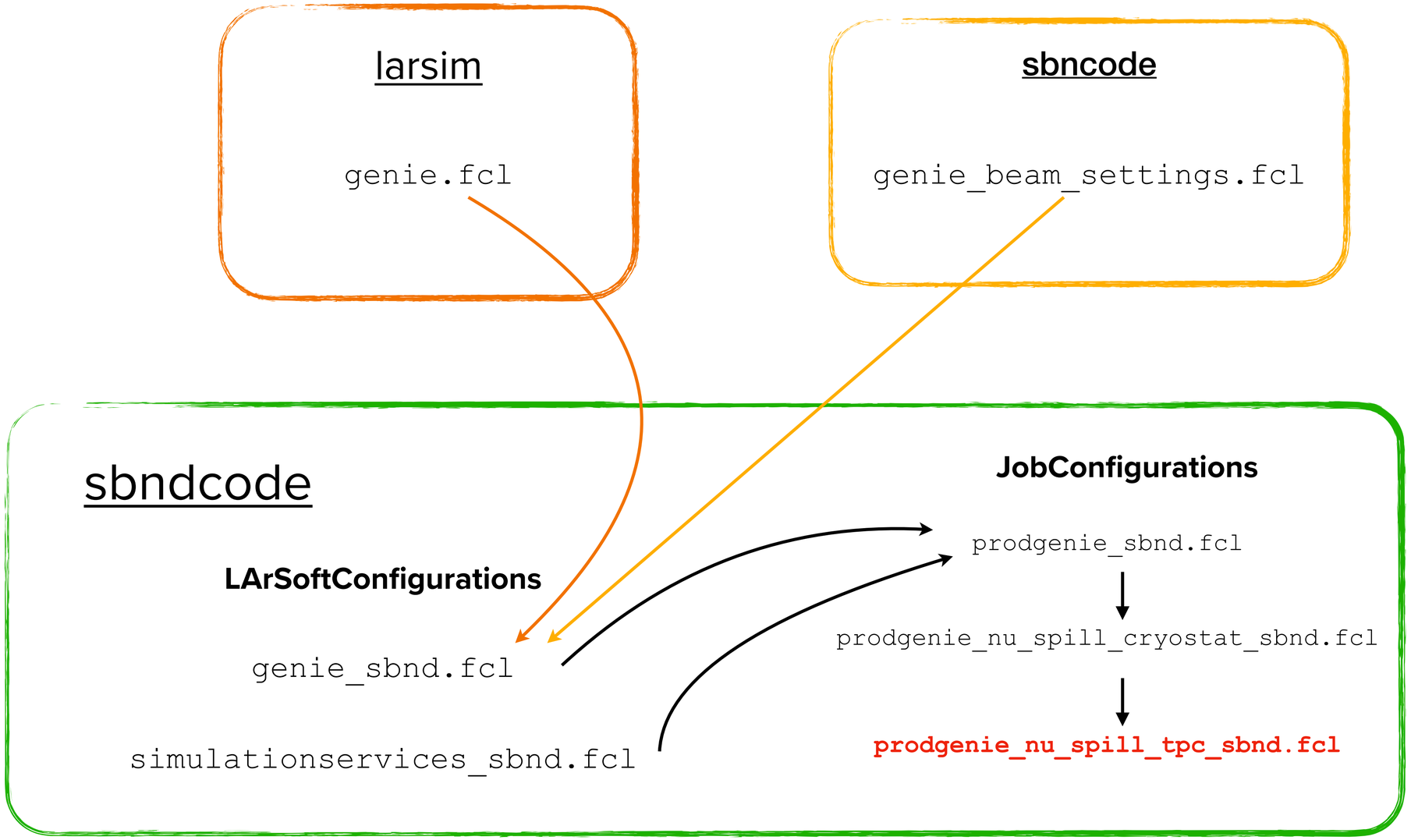
- Objects from the previous stages
- Collection of recob::Wire containing the simulated wire waveforms
- Collection of raw::OpDetWaveform containing the photon-detector waveforms



How to actually figure out where things are

LArSoft simulation FhiCL map (GENIE)

- Figuring out where things are defined is not trivial
- See this example of the FhiCL dependency to generate events with GENIE



A useful ally: find_fhicl.sh

- Script to help you find a particular fhicl file
  - Gives you the path to said fhicl
- Very helpful when trying to understand
  - where a particular parameter is set
  - which fhicl file(s) you're supposed to include
  - ...

Write this in a new file called find_fhicl.sh

```
#!/bin/bash

if [ $# -ne 1 ]; then
  echo "Error: please pass a fcl file name (or regex)"
  exit 1
fi
if [ -z ${FHICL_FILE_PATH+x} ]; then
  echo "Error: FHICL_FILE_PATH has not been set!"
  exit 2
fi
SEARCH_PATHS=`echo $FHICL_FILE_PATH | sed 's/:/\n/g'`
for THIS_PATH in $SEARCH_PATHS; do
  if [ -d $THIS_PATH ]; then
    find $THIS_PATH -name $1
  fi
done
```

FHiCL File Adventure II: find the first file

- Let’s try to find a standard single particle generation file
- Run `./find_fhicl.sh  singles_sbnd.fcl`
- It should print the path to the `singles_sbnd.fcl` file
- Open that file in your favourite text editor

FHiCL file adventure III: clue to the treasure

- Defines some parameters for generating events, but not everything is in there

- There's an include at the top!
  - Remember: fhicl files can inherit from other fhicl files

- Let's look at the file included
  - Repeat the steps of the previous slide to find `singles.fcl`

```
#include "singles.fcl"

BEGIN_PROLOG

sbnd_singlep: @local::standard_singlep

# By default pad ot a vector if it is size 1
physics.producers.generator.PadOutVectors: true

# Particle generated at this time will appear in main drift window at trigger T0.
physics.producers.generator.T0:     [0] # us

physics.producers.generator.P0:     [ -1.0 ]  # GeV/c
physics.producers.generator.SigmaP: [ 0.0 ]  # GeV/c
physics.producers.generator.PDist:  0
physics.producers.generator.X0:     [ 150.0 ] # cm
physics.producers.generator.Y0:     [ 150.0 ] # cm
physics.producers.generator.Z0:     [  -50.0 ] # cm
physics.producers.generator.Theta0XZ:     [  15.0 ] # degrees
physics.producers.generator.Theta0YZ:     [ -15.0 ] # degrees
physics.producers.generator.SigmaThetaXZ: [   0.0 ] # degrees
physics.producers.generator.SigmaThetaYZ: [   0.0 ] # degrees

END_PROLOG
```

FHiCL file adventure IV: the treasure

- You've found the core FHiCL file for the single particle gun!
- It contains all the configurable parameters for this module
- We got there by looking through all the files included
- You're going to do a lot of that in your work!
- Keep this file in mind, it can be useful for the tutorial.

```
BEGIN_PROLOG

#no experiment specific configurations because SingleGen is detector agnostic

standard_singlep:
{
 module_type:           "SingleGen"
 ParticleSelectionMode: "all"       # 0 = use full list, 1 =  randomly select a single listed particle
 PadOutVectors:     false       # false: require all vectors to be same length
                                # true:  pad out if a vector is size one
 PDG:                   [ 13 ]      # list of pdg codes for particles to make
 P0:                    [ 6. ]      # central value of momentum for each particle
 SigmaP:                [ 0. ]      # variation about the central value
 PDist:                 "Gaussian"  # 0 - uniform, 1 - gaussian distribution
 X0:                    [ 25. ]     # in cm in world coordinates, ie x = 0 is at the wire plane
                                # and increases away from the wire plane
 Y0:                    [ 0. ]      # in cm in world coordinates, ie y = 0 is at the center of the TPC
 Z0:                    [ 20. ]     # in cm in world coordinates, ie z = 0 is at the upstream edge of
                                # the TPC and increases with the beam direction
 T0:                    [ 0. ]      # starting time
 SigmaX:                [ 0. ]      # variation in the starting x position
 SigmaY:                [ 0. ]      # variation in the starting y position
 SigmaZ:                [ 0.0 ]     # variation in the starting z position
 SigmaT:                [ 0.0 ]     # variation in the starting time
 PosDist:               "uniform"   # 0 - uniform, 1 - gaussian
 TDist:                 "uniform"   # 0 - uniform, 1 - gaussian
 Theta0XZ:              [ 0. ]      #angle in XZ plane (degrees)
 Theta0YZ:              [ -3.3 ]    #angle in YZ plane (degrees)
 SigmaThetaXZ:          [ 0. ]      #in degrees
 SigmaThetaYZ:          [ 0. ]      #in degrees
 AngleDist:             "Gaussian"  # 0 - uniform, 1 - gaussian
}

random_singlep: @local::standard_singlep
random_singlep.ParticleSelectionMode: "singleRandom"  #randomly select one particle from the list
```

FHiCL file adventure IV: more adventures!

- Congrats! You now know how to explore the FHiCL files dependencies starting from any file

- Below, you'll find a generic workflow including all the stages presented in this lecture
  - Explore the FHiCLs!
  - Careful: you may get lost in the process…

```
1. prodgenie_nu_spill_tpc_sbnd.fcl
2. standard_g4_sbnd.fcl
3. standard_detsim_sbnd.fcl
```

The parameter definition you need!

FHiCL file adventure V: the shortcut

- If you just want to see the values of all parameters defined for a module without having to open all the includes, there is a solution: `fhicl-dump`

- Let's see what it does!

- Run `fhicl-dump prodgenie_nu_spill_tpc_sbnd.fcl`

- The output may not fit in the terminal so you can also run

```
fhicl-dump prodgenie_nu_spill_tpc_sbnd.fcl > fcl_dump_genie_sbnd.txt
```

and open `fcl_dump_genie_sbnd.txt` in your text editor.

Summary

- **LArSoft simulation produces events made to look like raw detector data**
  - Accounts for both physics and detector effects

- **Simulation in LArSoft has many steps**
  - You likely won’t need to tweak all (if any) of them

- **It is highly modular and can handle a variety of geometries**

- **The software constantly evolves:**
    - Keep track of new developments!

Questions?

Backup

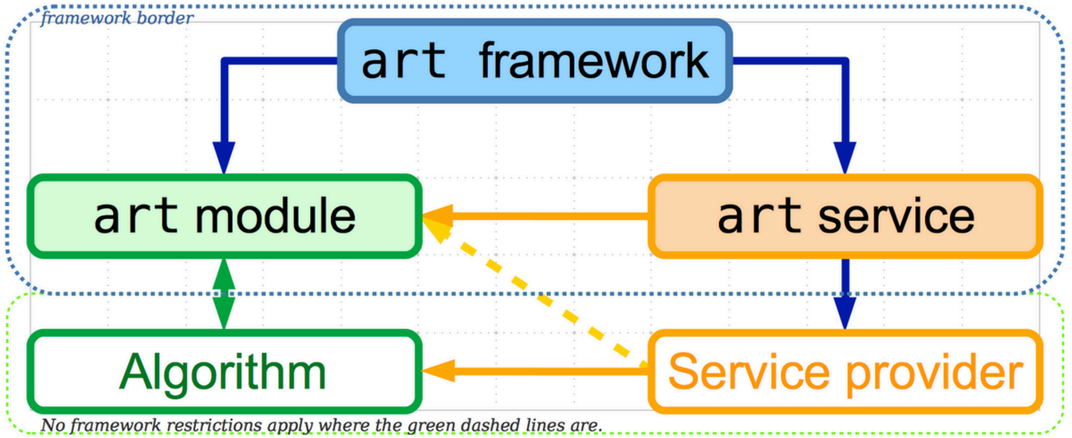
Wire-Cell configuration parameters

```
dunefd_horizdrift_1x2x6_sim_nfsp : {
    module_type : WireCellToolkit
    wcls_main: {
        tool_type: WCLS
        apps: ["TbbFlow"] # TbbFlow, Pgrapher
        logsinks: ["stdout:info", "wcls-sim-drift-simchannel-nf-sp.log:debug"]
        loglevels: ["debug"]
        plugins: ["WireCellPgraph", "WireCellGen","WireCellSio","WireCellRoot","WireCellLarsoft","WireCellTbb"]
        # needs to be found via your WIRECELL_PATH
        configs: ["pgrapher/experiment/dune10kt-1x2x6/wcls-sim-drift-simchannel-nf-sp.jsonnet"]
        # Contract note: these exact "type:name" must be used to identify
        # the configuration data structures for these components in the Jsonnet.
        inputers: ["wclsSimDepoSource:"]
        outputers: [
            "wclsSimChannelSink:postdrift",
            "wclsFrameSaver:spsignals"
        ]
        # Make available parameters via Jsonnet's std.extVar()
        params: {
            # Pgrapher, TbbFlow
            engine: "TbbFlow"
        }
        structs: {
          # number of time samples
          #nticks: @local::dunefd_detproperties.NumberTimeSamples
          # Longitudinal diffusion constant [cm2/ns]
          DL: @local::dunefd_largeantparameters.LongitudinalDiffusion
          # Transverse diffusion constant [cm2/ns]
          DT: @local::dunefd_largeantparameters.TransverseDiffusion
          # Electron lifetime [us]
          lifetime: @local::dunefd_detproperties.Electronlifetime
          # Electron drift speed, assumes a certain applied E-field [mm/us]
          driftSpeed: 1.60563
          # G4RefTime [us]
          G4RefTime: @local::dunefd_detectorclocks.G4RefTime


        }
    }
}
```

Modules

- Implemented as self-standing class with one header file and on implementation file
- Extracts the parameters to run from a FHICL file
- Has an input phase, running phase and output phase
- Produces data products and histograms
- Technically, there should be
  - An algorithm class to perform all the operations required for a task
  - A framework module class to manage coordinate algorithms
- Sometimes, algorithms are implemented within modules.



Communication in LArSoft: services

- Services are classes with only one instance managed by the framework and can be accessed by the different modules.
- They provide information about (non-exhaustive lists):
  - Geometry: TPC structure, optical detectors positions, auxiliary detectors (e.g. CRT)
  - Physical properties: LAr properties (e. g. radiation length), detector properties (e. g. drift velocity)
  - Physics simulation: GEANT4 parameters

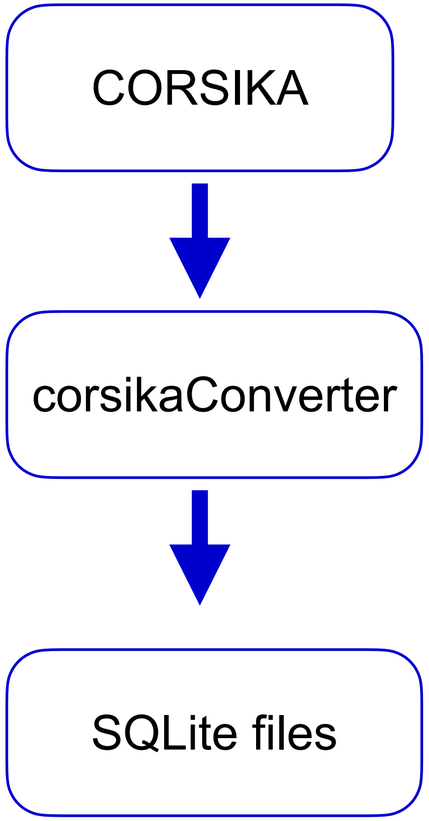
GENIE common fhicl file

```
standard_genie:
{
 module_type:      "GENIEGen"

 DefinedVtxHistRange: false
 VtxPosHistRange:  [0. , 0., 0., 0., 0., 0.] #if DefinedVtxHistRange is set to true VtxPosHistRange sets the hist range of the vertex positio
                                             #It is helpful for dual phase detector for which the range is asymmetric.
 PassEmptySpills:  false
 FluxType:         "mono"     #mono, histogram, ntuple, or simple_flux
 FluxFiles:         ["flugg_L010z185i_neutrino_mode.root"] #name of file with flux histos
 BeamName:         "numi"          #numi or booster at this point - really for bookkeeping
 TopVolume:        "volDetEnclosure"        #volume in which to produce interactions
 EventsPerSpill:   1.              #set != 0 to get n events per spill
 POTPerSpill:      5.e13           #should be obvious
 MonoEnergy:       2.              #in GEV
 BeamCenter:       [-1400., -350., 0.]  #center of the beam in cm relative to detector coordinate origin, in meters for GENIE
 BeamDirection:    [0., 0., 1.]    #all in the z direction
 BeamRadius:       3.              #in meters for GENIE
 SurroundingMass:  0.0             #mass surrounding the detector to use
 GlobalTimeOffset: 10000.          #in ns - 10000 means the spill appears 10 us into the readout window
 RandomTimeOffset: 10000.          #length of spill in ns
 FiducialCut:      "none"          #fiducial cut, see https://cdcvs.fnal.gov/redmine/projects/nusoft/wiki/GENIEHelper
 GenFlavors:       [12,14,-12,-14] #pdg codes of flux generator neutrino flavors
 Environment:      [  ]  # obsolete
 ProductionMode:   "yes"           #turn off the GENIE verbosity
 EventGeneratorList: "Default"
 DetectorLocation: "MINOS-NearDet" #location name for flux window
 MixerConfig:      "none"          #no flux mixing by default
 #MixerConfig:     "swap 12:16 14:16 -12:-16 -14:-16" # example flavor swapping
 MixerBaseline:    0.              #distance from tgt to flux window needs to be set if using histogram flx
 DebugFlags:       0               #no debug flags on by default
 XSecTable: "gxspl-FNALsmall.xml"  #default cross section
}
```

larsim/EventGenerator/genie.fcl

What is in the simulation?



- **corsikaConverter** takes the CORSIKA output and generates SQLite files containing 3 tables:
  - CORSIKA **input parameters**
  - **Showers:** all showers with the number of particles reaching the observation level
  - **Particles**: all particles reaching observation levels
- Files used by the current simulation can be found here:
  `/cvmfs/sbn.osgstorage.org/pnfs/fnal.gov/usr/sbn/persistent/stash/physics/cosmics/Fermilab/CORSIKA/standard/v01_00`