IMPERIAL

LArSoft TPC Simulation

Fantastic FHICLs and where to find them

Anyssa Navrer-Agasson

9th UK LArSoft Workshop - 28/10/2024

Goal of the lecture/tutorial

What will you (hopefully) know by the end?

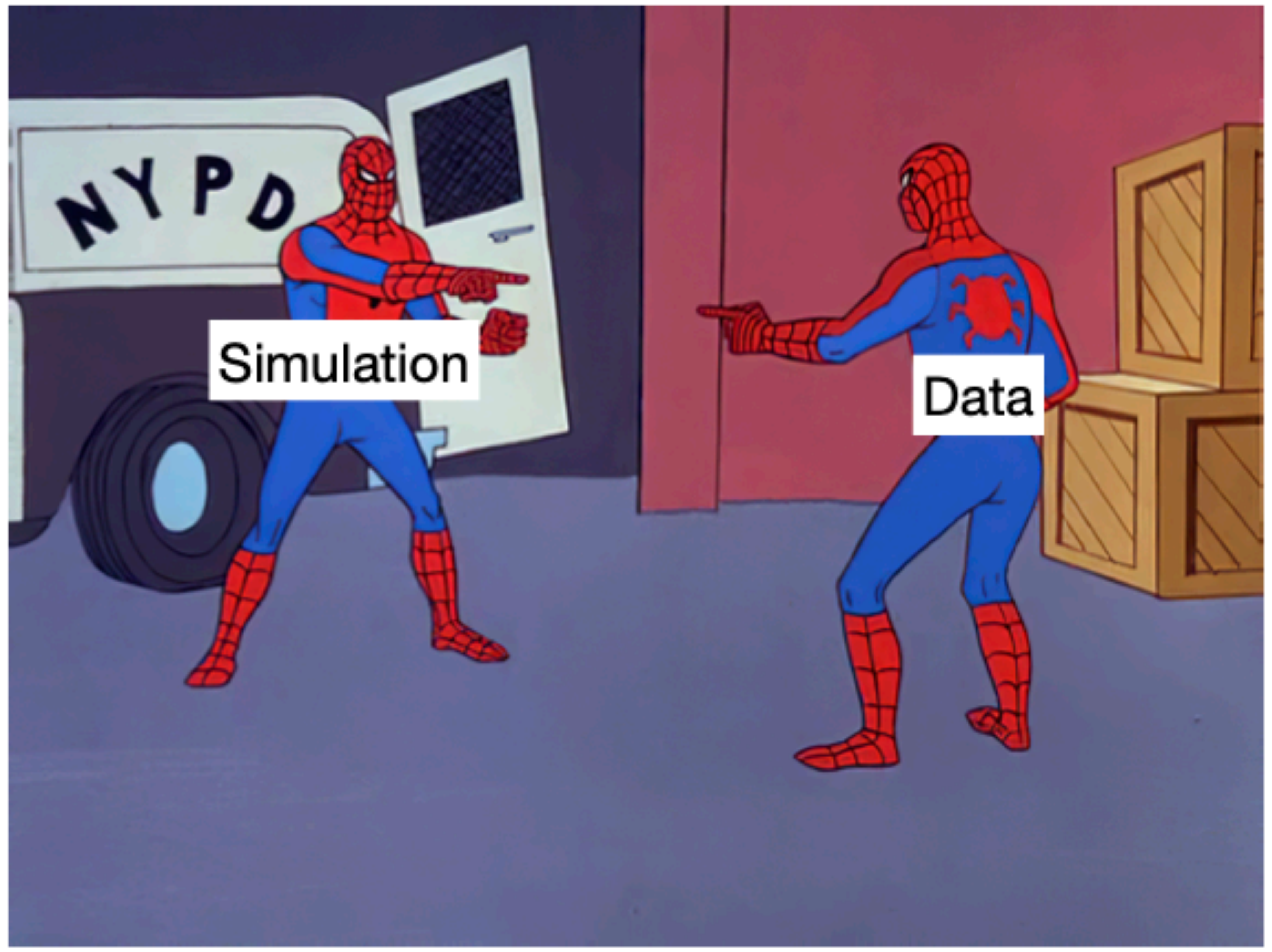
- What are the steps needed to generate events?
- What are the different tools used for each step?
- How do different part of the simulation communicate?
- What is the output of each step?

What is LArSoft?

- General FNAL LAr experiments simulation framework:
  - Only need to learn one framework, even if you're working on multiple experiments.
  - Need to have both common and experiment specific parts.
- In the following lectures/tutorials you will learn about how to reconstruct events. This lecture will help you understand how these events get generated.

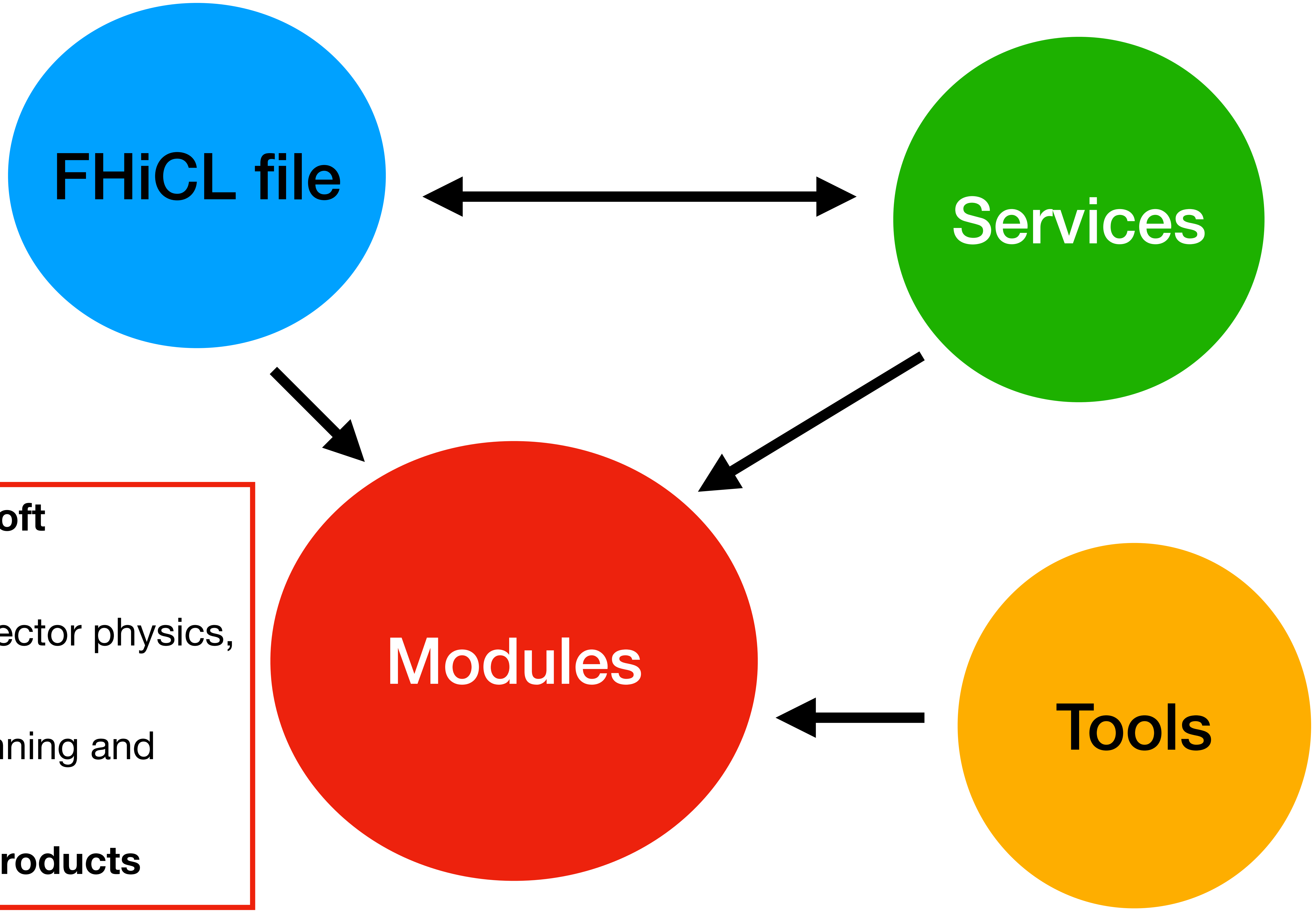
Why is LArSoft?

- 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



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

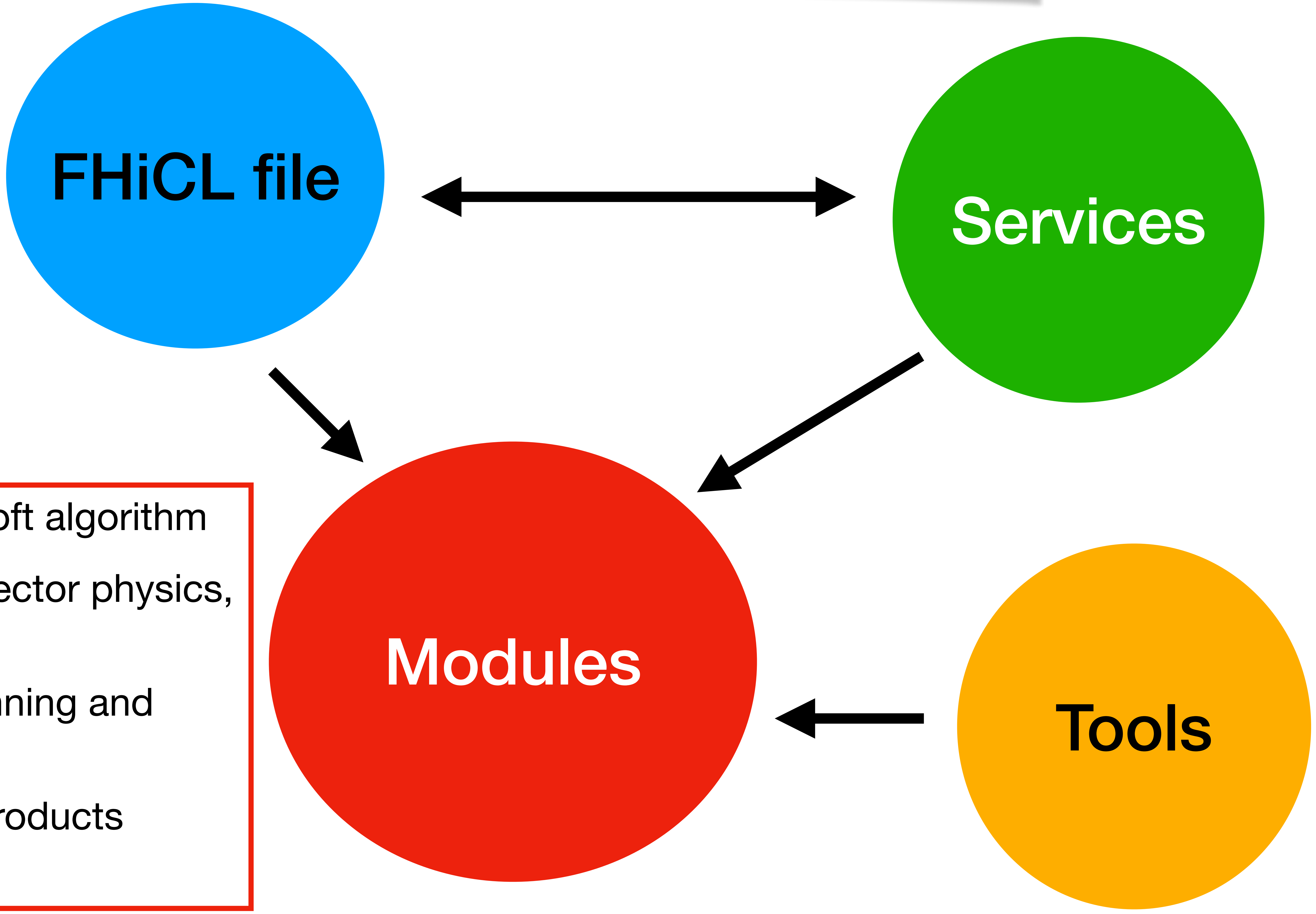
- Implementation of **LArSoft algorithms**
  - Event generator, detector physics, reconstruction, …
  - Has setup, input, running and output phase
  - Uses LArSoft **data products**

- 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!

Where to find the right FHiCLs? (examples)

- **General LArSoft simulation fhicls** can be found in: /cvmfs/larsoft.opensciencegrid.org/products/larsim/<larsim_version>/job
- In **sbndcode**, it's pretty straightforward: /cvmfs/sbnd.opensciencegrid.org/products/sbnd/sbndcode/<sbndcode_version>/fcl
- In **uboonecode**, everything is in: `/cvmfs/uboone.opensciencegrid.org/products`
- **BUT**
  - Most general fhicls are in: `uboonecode/job`
  - Event generators in: `ubcore/job`
  - Analysis fhicls in: `ubana/job`

Side note: 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
  - …

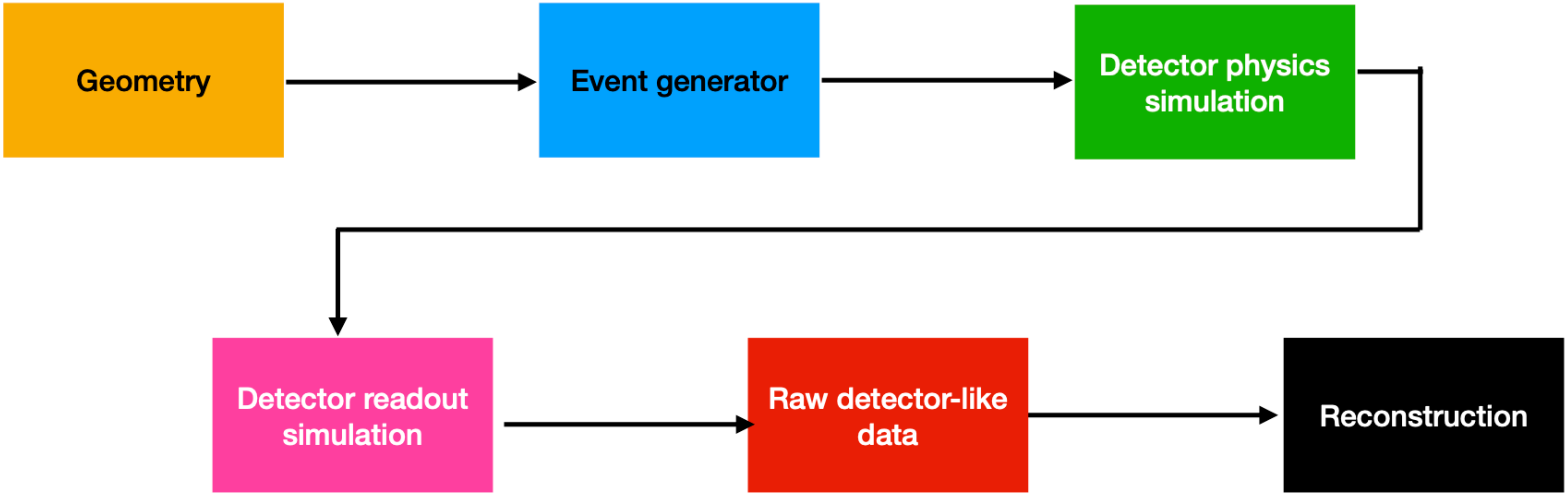
```
#!/bin/bash

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
```

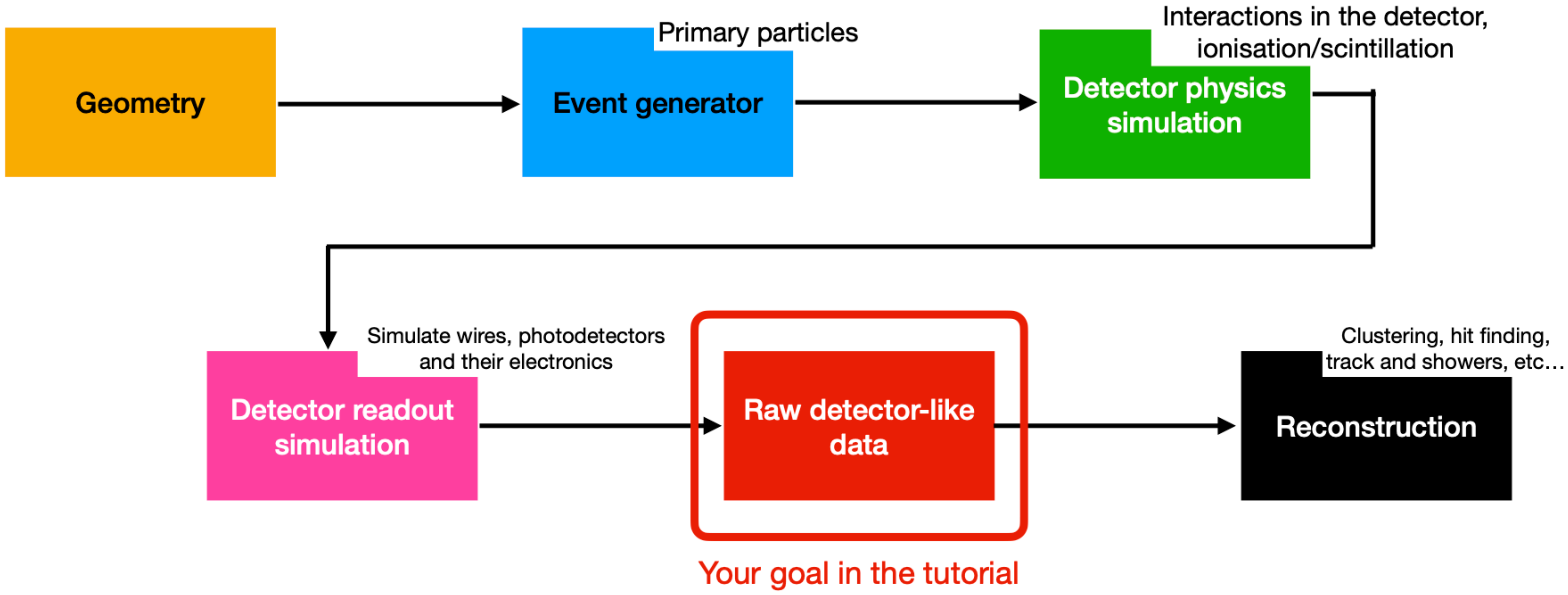
Usage:

```
./find_fhicl.sh <name_of_fhicl_file.fcl>
```

LArSoft simulation flowchart?

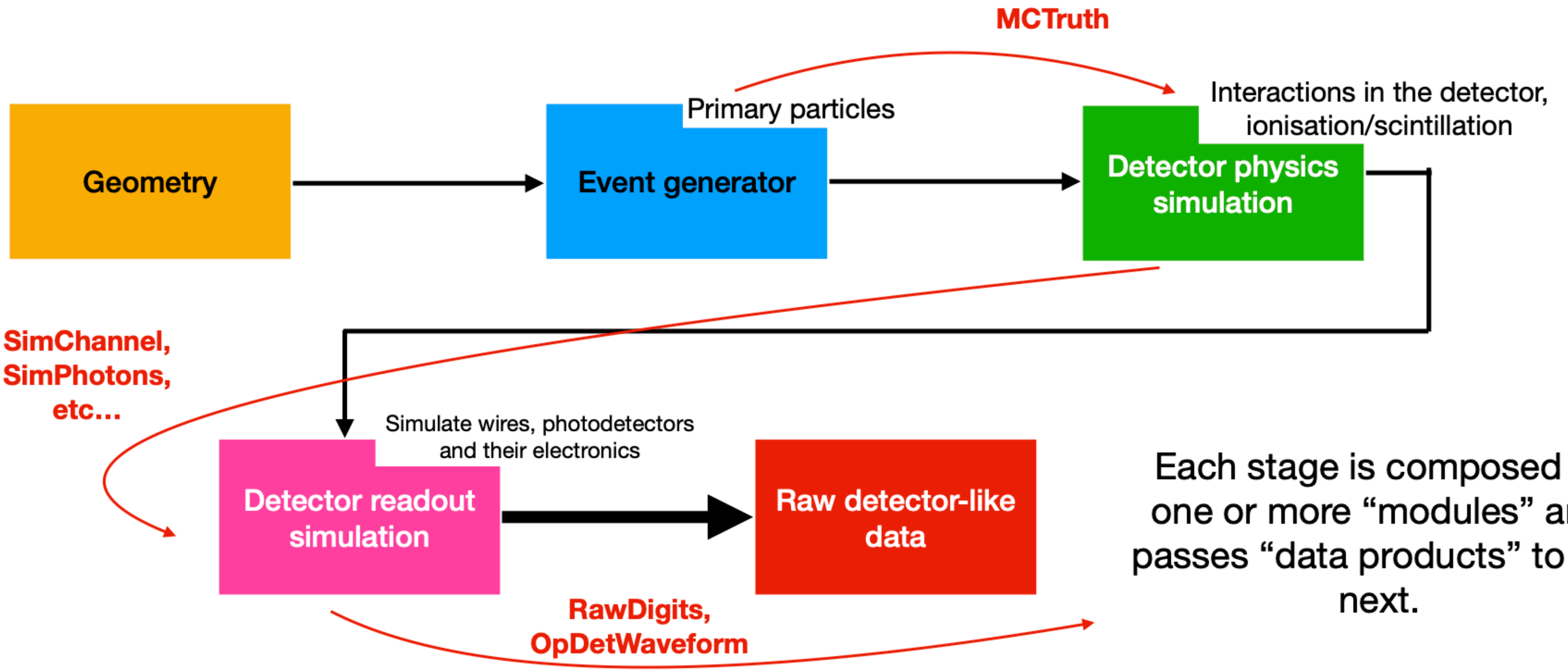


LArSoft simulation flowchart?



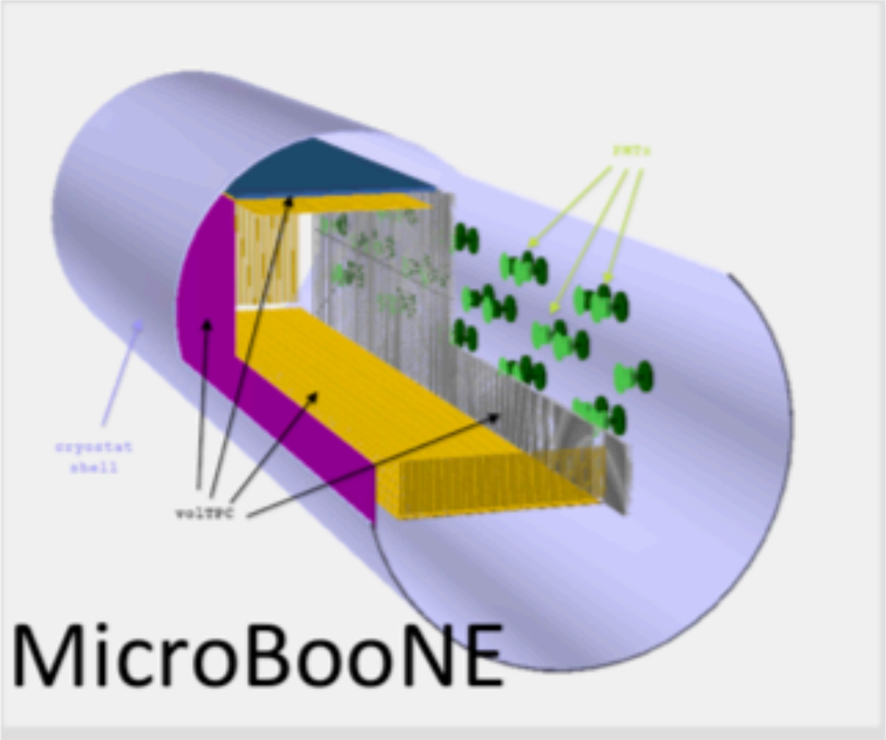
LArSoft simulation flowchart?

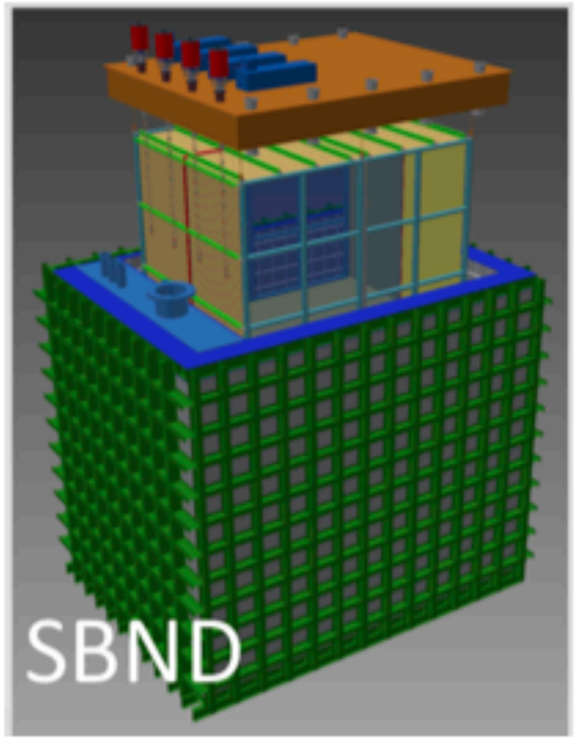
Data products: classes saved in the output artROOT file

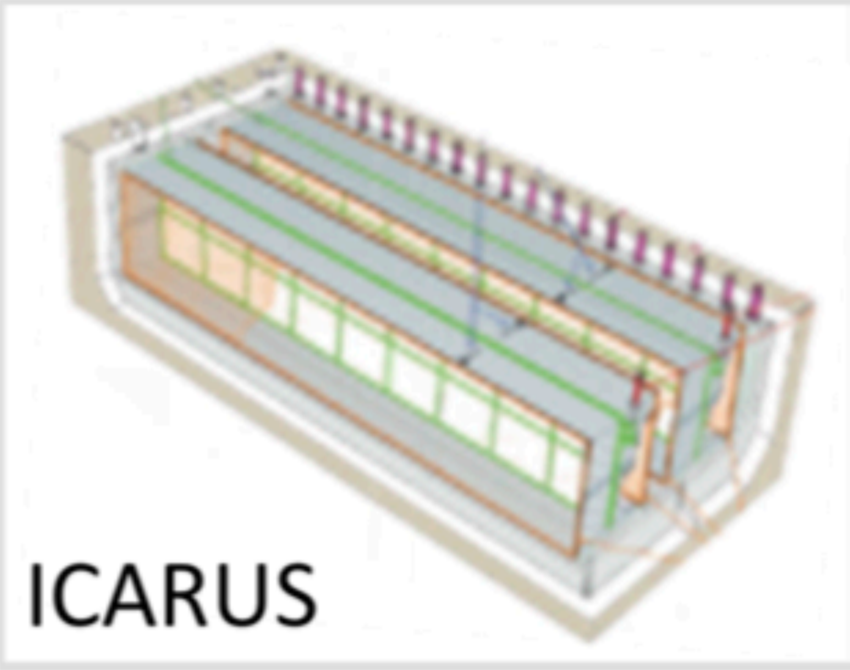


Each stage is composed of one or more “modules” and passes “data products” to the next.

Step 1: Build-A-Detector





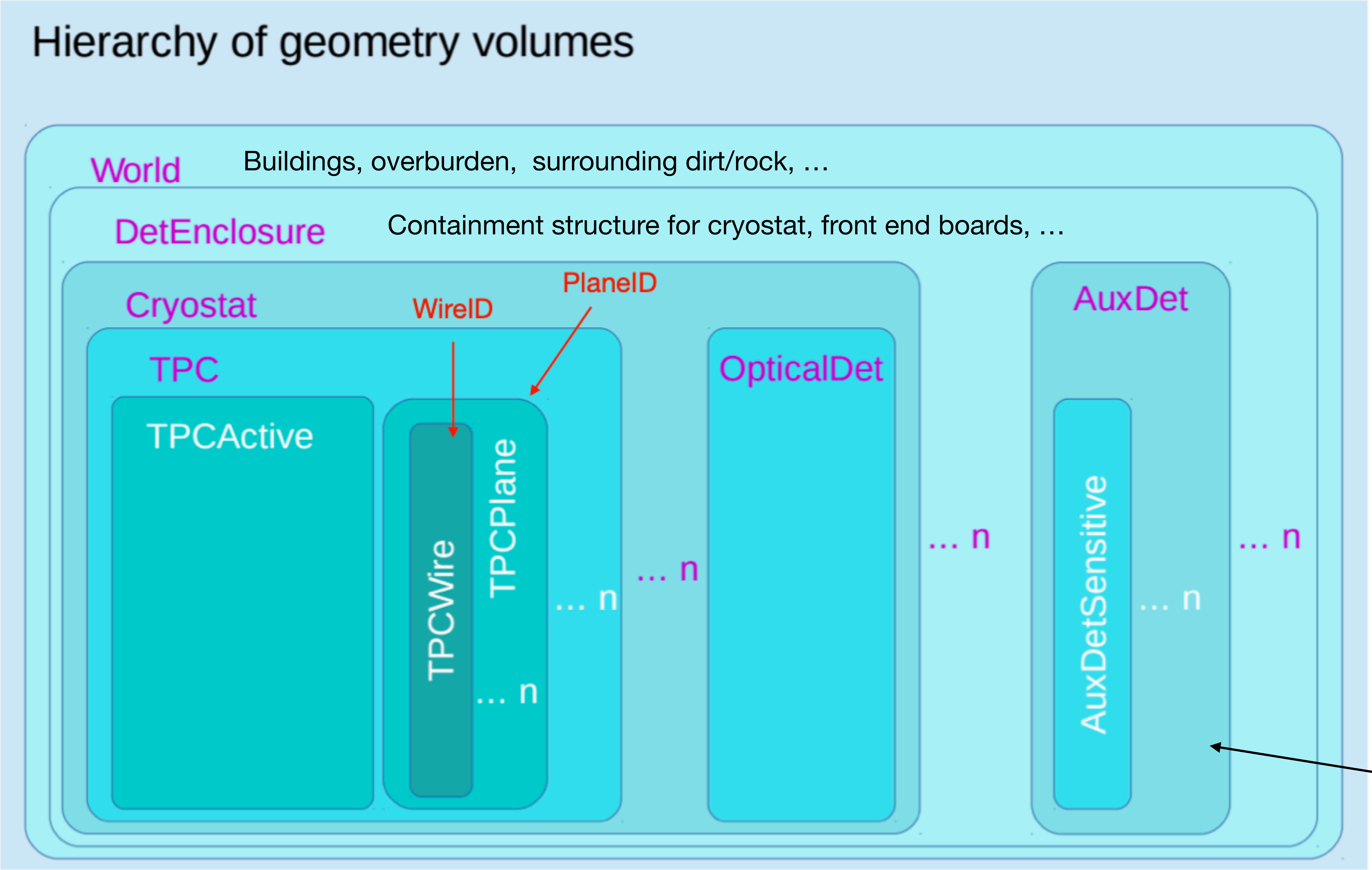


and more!

- Each detector just needs to add a new geometry description
- Simulation/reconstruction knows how to access different geometries, but are not dependent on any one
- Uses GDML (Geometry Description Markup Language)

Step 1: Build-A-Detector

SBND geometry files in sbndcode/Geometry/gdml

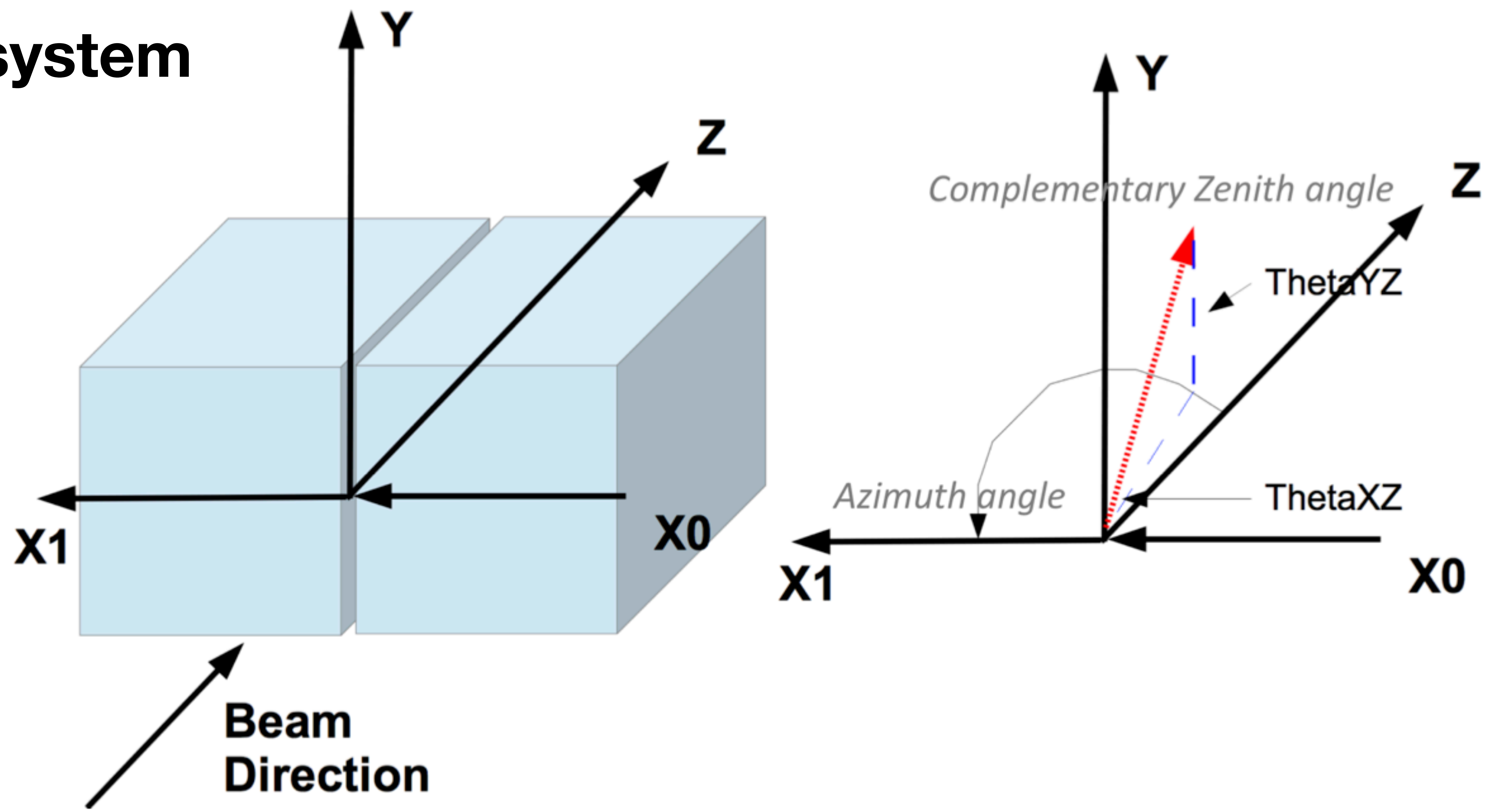


- Use ID objects to specify which instance of TPC geometry objects you want
- There are sorting algorithms in place that determine which one goes first in the code

e. g. Cosmic Ray Tagger

Step 1: Build-A-Detector

Coordinate system

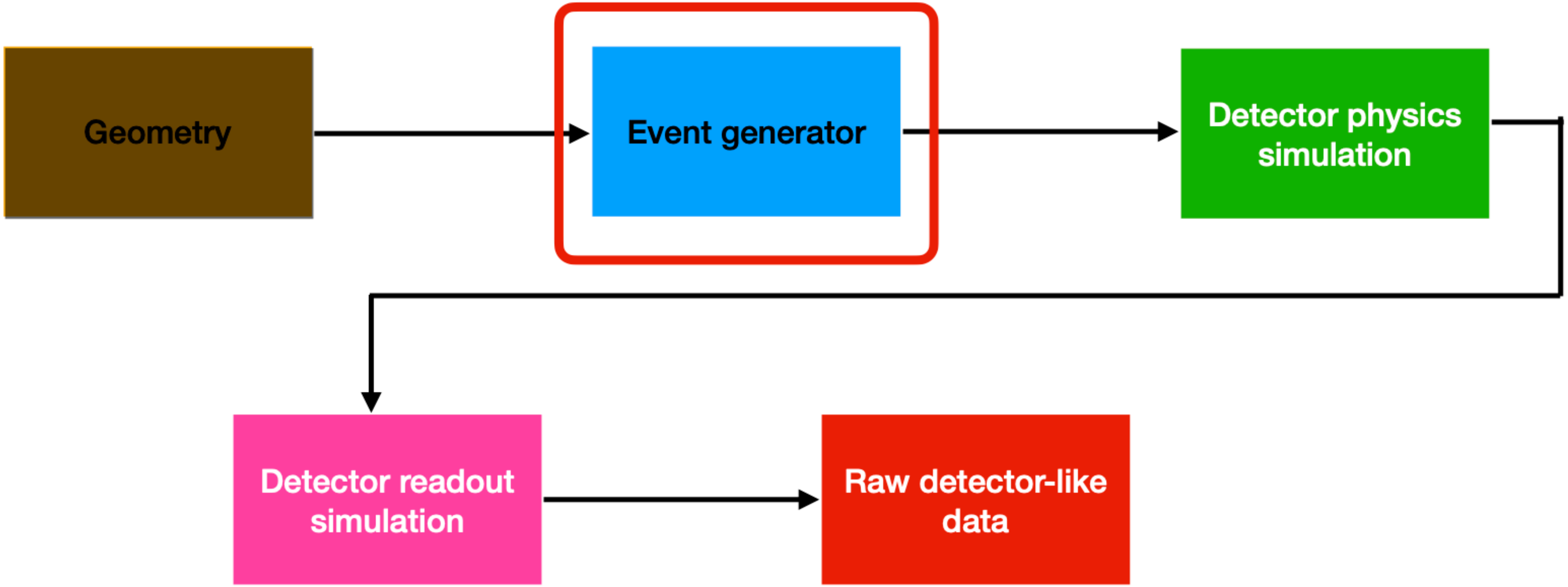


For all detectors: Z increases in the direction of neutrino travel, Y increases away from the centre of the Earth and X increases so as to make a right-handed coordinate system.

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

- First step in generating events in LArSoft (majority of cases).
  All generators live in `larsim/EventGenerator`
- We may be interested in different sources of particles:
  - Single particle gun (SingleGen)
  - Neutrino interactions (GENIE)
  - Cosmic rays (CORSIKA)
  - Supernova neutrinos (MARLEY)
  - Read in from text file (TextFileGen)
- Possibility to combine generators to create complex events

Event generators: Single Particle Gun

- Used to generate individual particles or very simple interactions
- You can define the particle type (PDG code), position, momentum and their how they vary (uniform, gaussian)
- There is an option to run with different/multiple particles either randomly between events or within the same event.
  - This is a bit tricky because you need to specify parameters for all particles. But there is a trick: you can ask LArSoft to "PadOutVectors".

More on that in the tutorial!

```
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

argoneut_singlep: @local::standard_singlep

microboone_singlep: @local::standard_singlep
microboone_singlep.Theta0YZ: [ 0.0 ]  # beam is along the z axis.
microboone_singlep.X0:    [125]       # in cm in world coordinates, ie x = 0 is at the wire plane
microboone_singlep.Z0:    [50]        # in cm in world coordinates
```

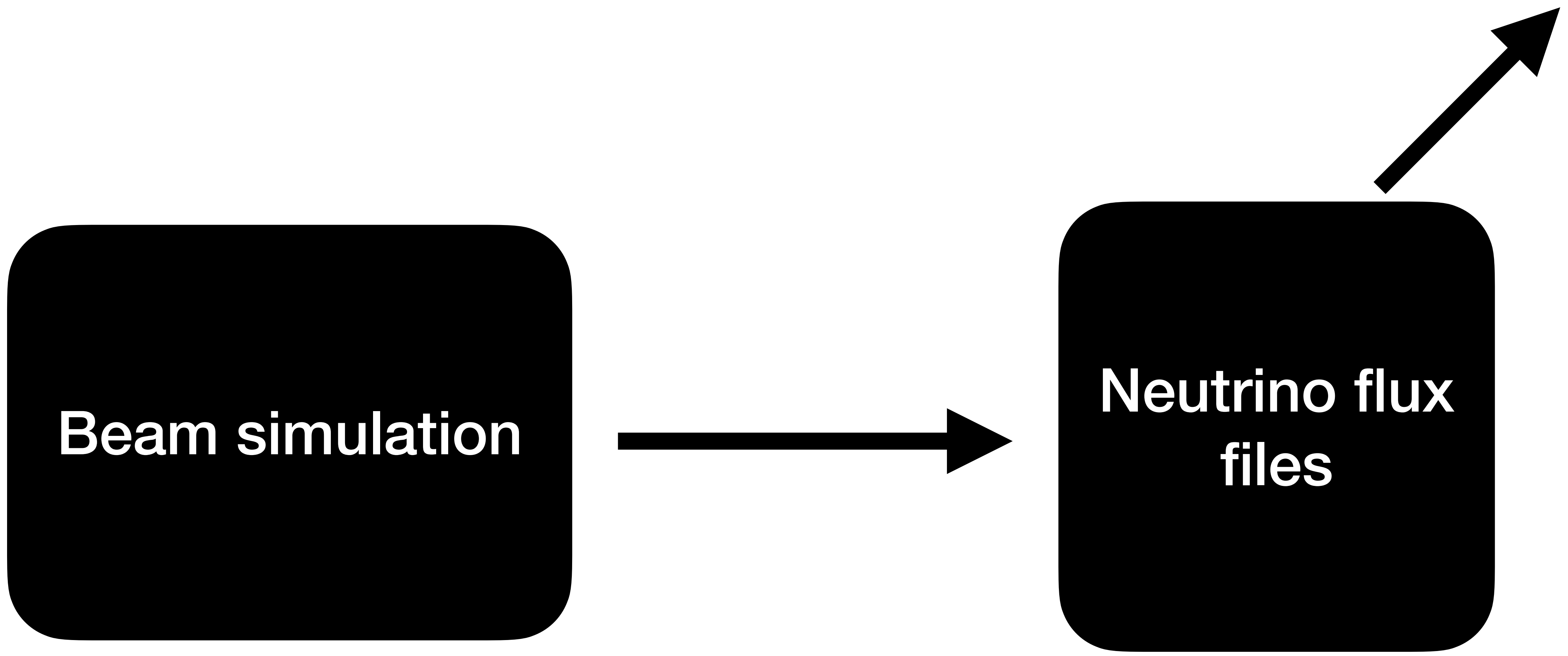
larsim/EventGenerator/singles.fcl

Event generators: GENIE

- GENIE is the most popular neutrino event generator.
- You provide the flux files and specify where you want the neutrino to interact.
- It produces neutrino secondaries according to flux files appropriate to the detector under study.
- You can specify the type of interaction (CCQE, RES, DIS, etc…).
- GENIE is able to calculate the POT exposure for the generated sample.

(Protons on Target)

UNIVERSAL NEUTRINO GENERATOR & GLOBAL FIT

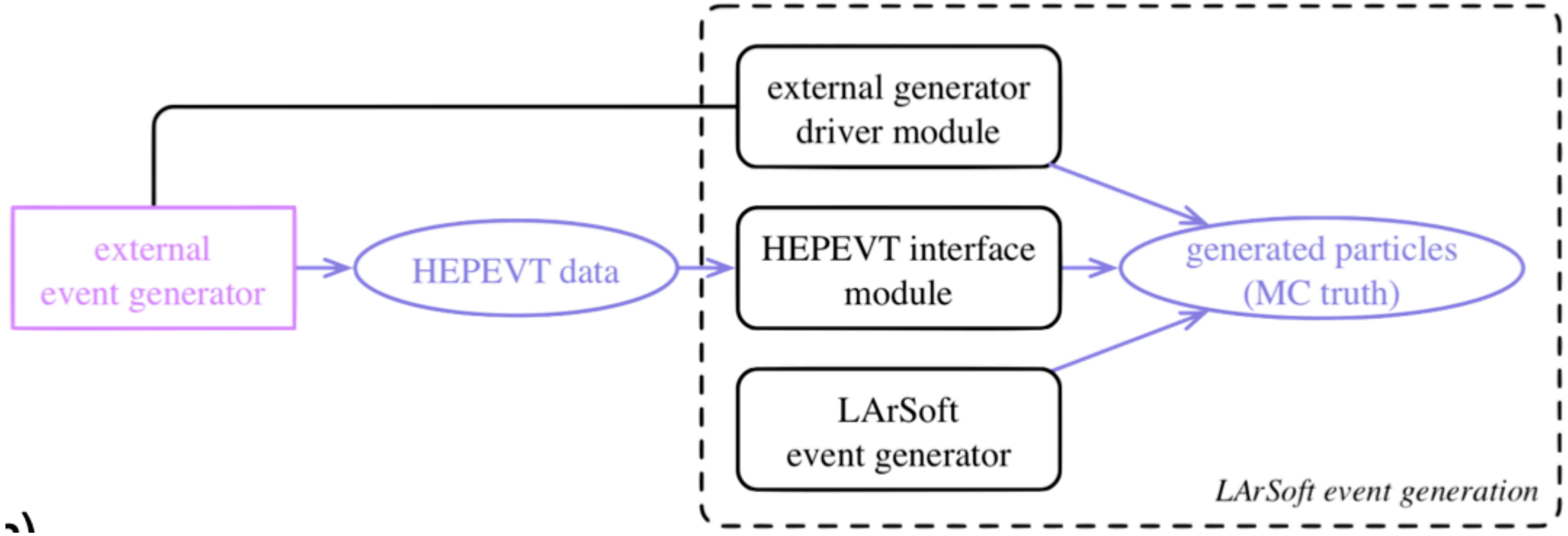


Event generators: TextFileGen

- To use every time a generator isn't interfaced with LArSoft (#BSM)
- Can generate primary particles from a file containing a list of particles, with PDG code, position, momentum, etc…
- Only takes HEPEVT files as input
- Very simple FHICL file!
- Can be tricky to use…

```
standard_textfilegen:
{
 module_type:   "TextFileGen"
 InputFileName: "input.txt"   #name of file containing events in hepevt format to
                              #put into simb::MCTruth objects for use in LArSoft
}
```

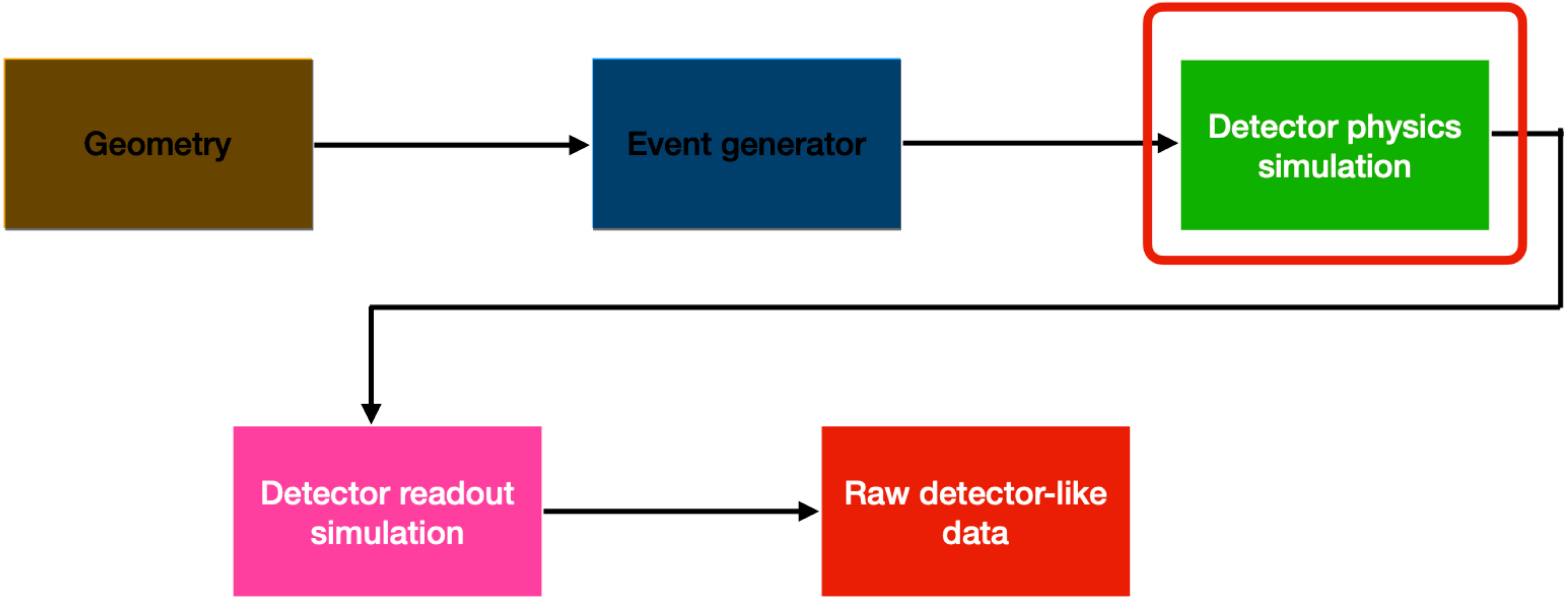
larsim/EventGenerator/textfilegen.fcl



What's in your output file? (1)

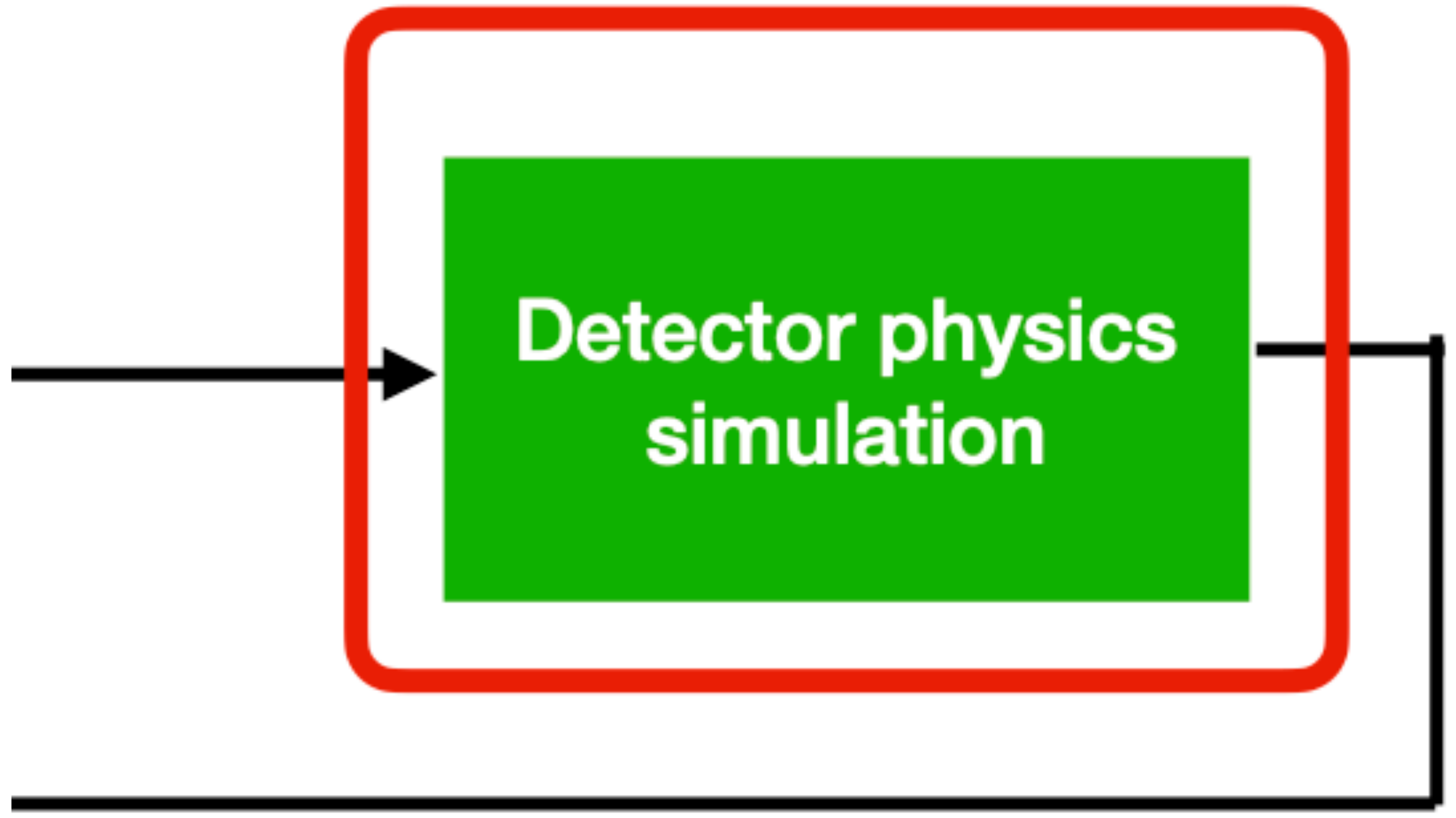
- `simb::MCTruth` objects (usually one per generator used), which will be picked up by GEANT4 and propagated though the detector.
- Contains:
  - Information about the generator
  - List of particles (`simb:MCParticle`) with PDG code, position, momentum, etc…
  - Information about neutrino interaction (if any)

Step 3: the tribulations of particles in LAr



Step 3: the tribulations of particles in LAr

- Interactions of the generated particles with the detector and energy depositions
- Transportation of ionisation electrons and scintillation photons to the readout
- Includes TPC and auxiliary detectors (e.g. CRT)



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

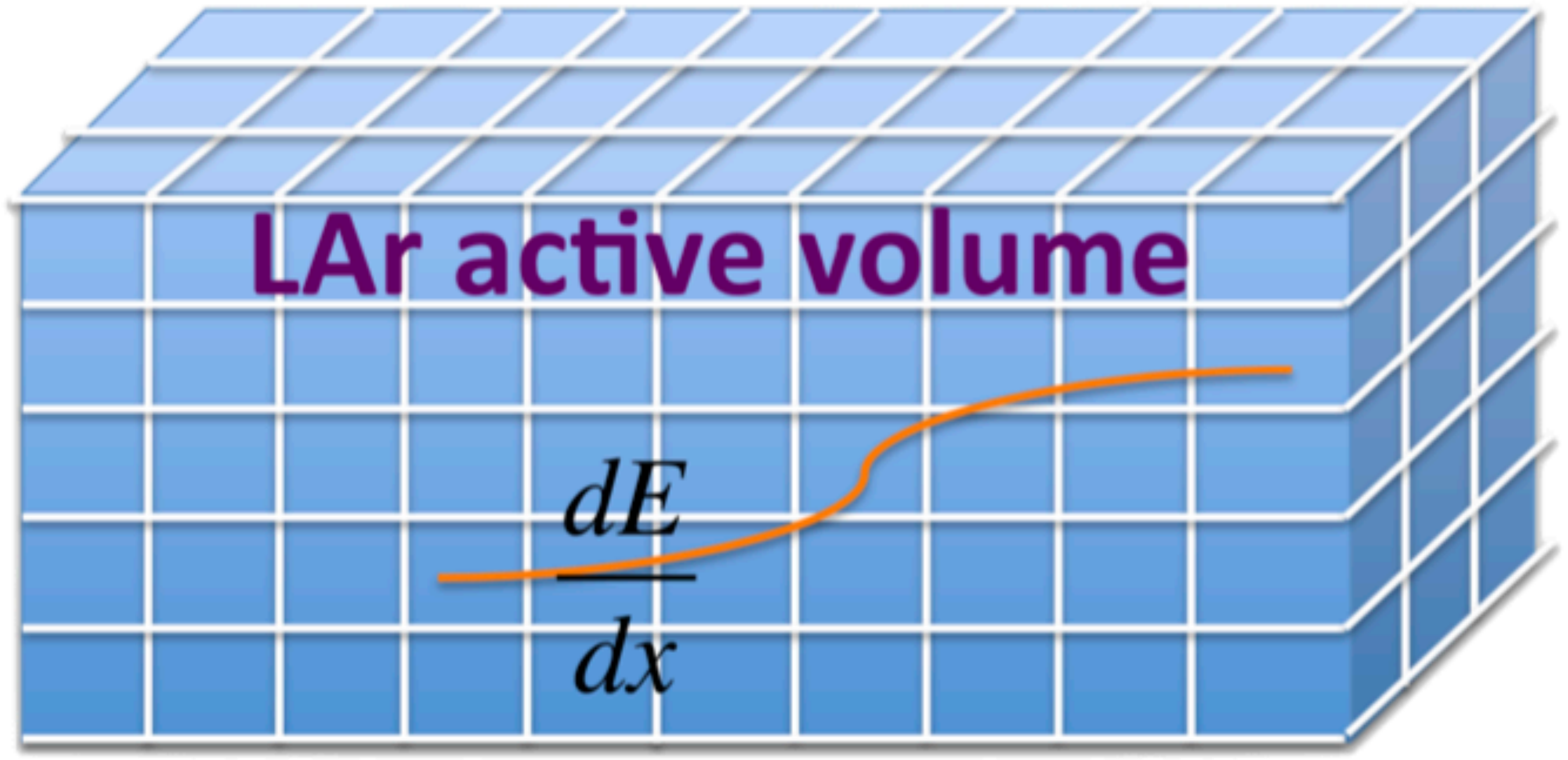
Step 3: the tribulations of particles in LAr

Where we make our particles interact and see what comes out

- Relies on GEANT4 for particle transportation and energy depositions
- Takes the MCTruth objects from generator stage and passes the primary particles to Geant4 to calculate the energy depositions along propagation though LAr
- Particles are stepped one after the other (oblivious to each other’s existence)
    - A step is a ‘delta’ in the particle trajectory, particle information (energy, position, etc..) is evaluated at each step
    - Step length is calculated based on the physics list (all processes and models to consider for particle interactions)

Step 3: the tribulations of particles in LAr

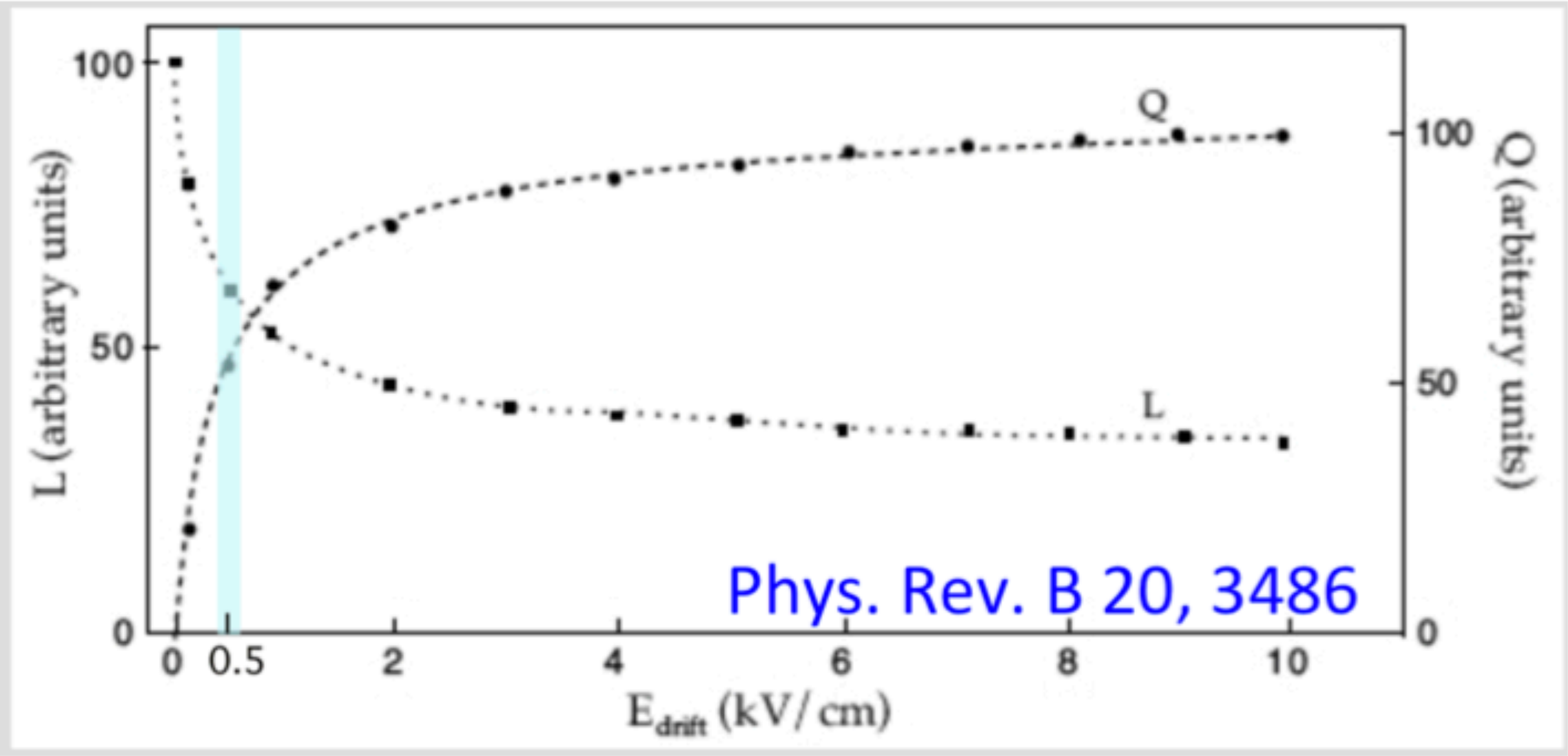
Simulation strategy



Voxel size defined in `simulationservices.fcl`

- Detector volume divided into voxels (3D pixels)
- Geant4 deposits energy in each voxel

- Number of ionisation electrons and scintillation photons produced depends on the electric field



Step 3: the tribulations of particles in LAr

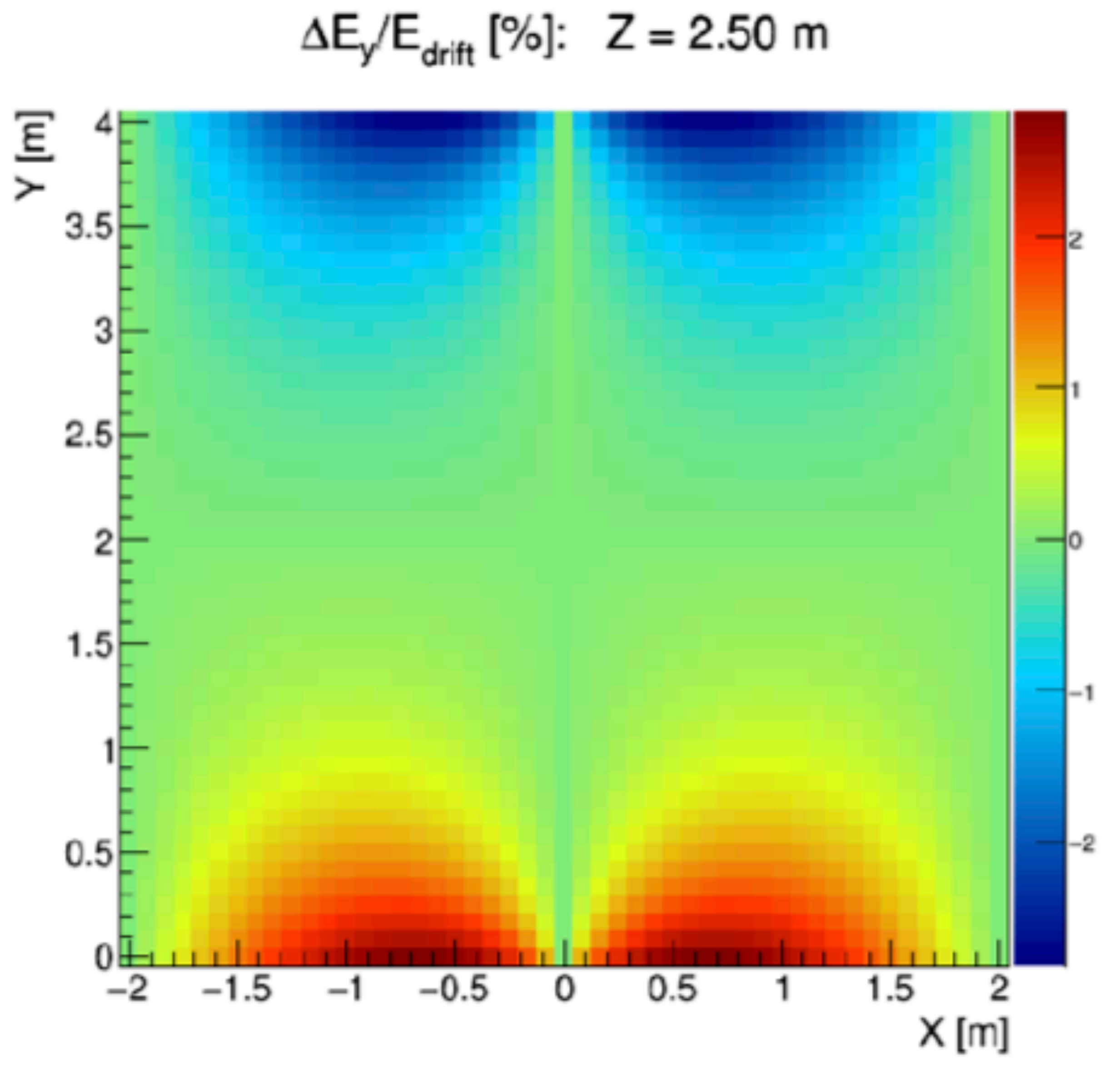
Scintillation photons

Not my problem! See Patrick's/ Andrzej's lecture/tutorial ;)

Step 3: the tribulations of particles in LAr

Electron drift

- Number of ionisation electrons computed from energy deposition
    - dE/dx -> [recombination, lifetime correction (impurities)] -> n_electrons
- Electrons are split in groups (default 600)
- They are projected to a Y, Z position at the position of the wire planes.
- The position is then smeared using transverse diffusion coefficients - this results in an effective diffusion of the whole deposition.
- Longitudinal diffusion is applied the same way
- Generates sequence of arrival times for each channel



Corrections due to field distortions (space charge effect) are applied

Parameters in: `larsim/simulation/simulationservices.fcl`

Step 3: the tribulations of particles in LAr

LArG4 is dead! Long live larg4!

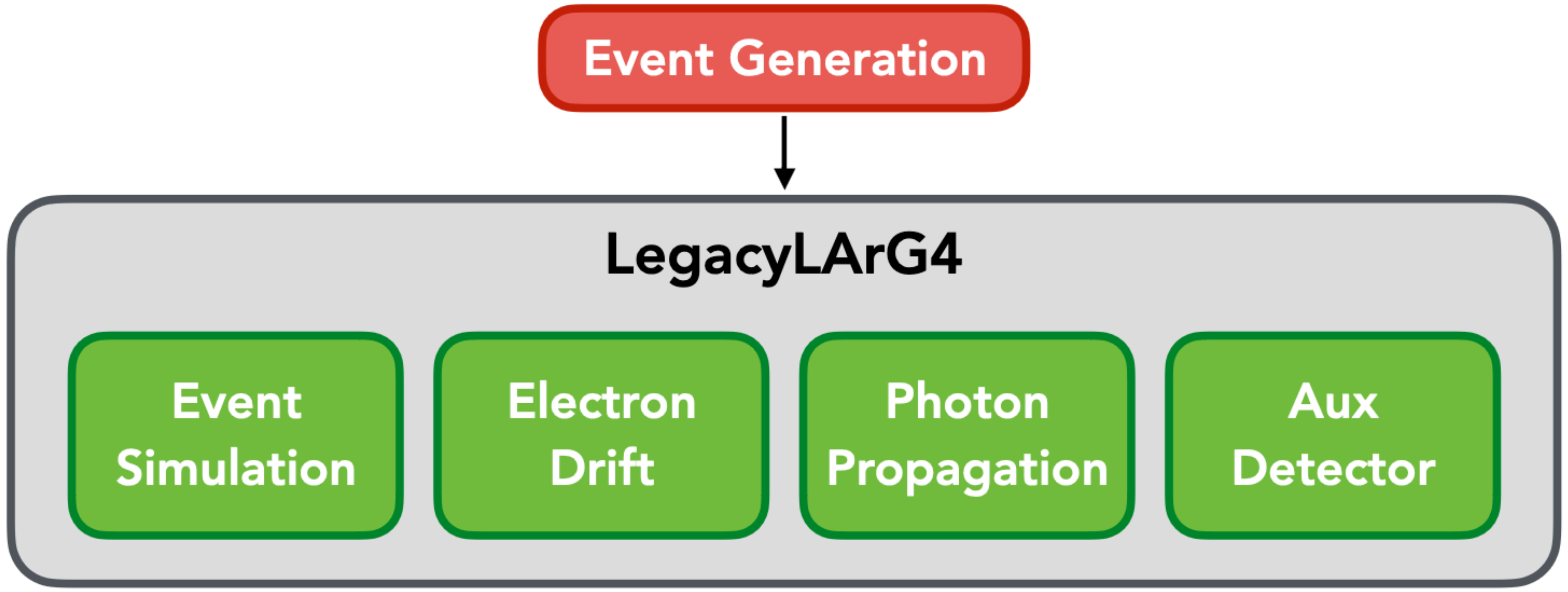
Legacy	Refactored
• Depends on nutools • To change physics list, one needs to change ConfigurablePhysicsList.hh in larsim and recompile • There can only be LAr as scintillating material	• Depends on artg4k • Can change physics list from fhicl file, or extend it, or create our own physics list in sbndcode and register it via G4 physics stamper class • Can have different scintillating materials in the detector

MicroBooNE uses legacy, SBND/DUNE use refactored

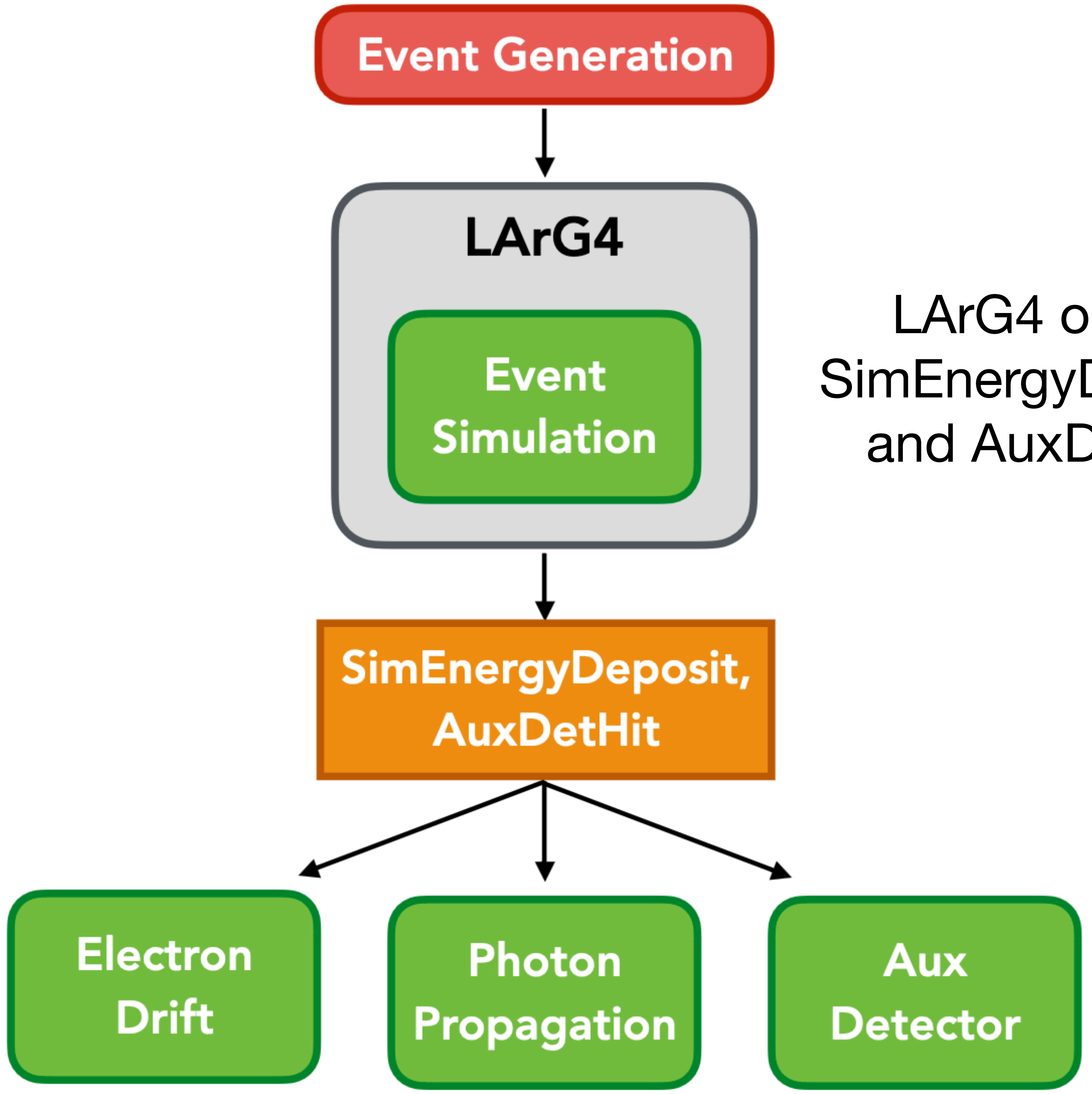
Step 3: the tribulations of particles in LAr

Illustration credit: Marco Del Tutto

LArG4 is dead! Long live larg4!



One module to rule them all



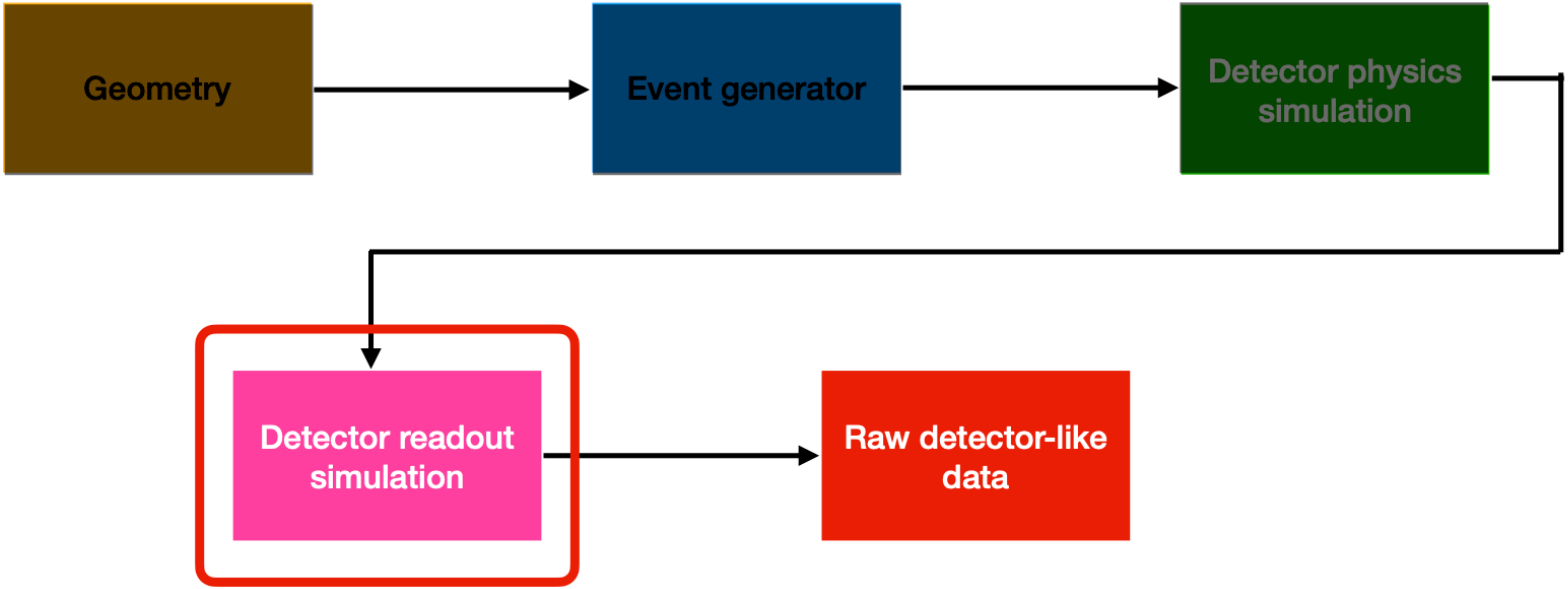
LArG4 output SimEnergyDeposits and AuxDetHits

Ionisation, electron drift, optical simulation, CRT simulation is done by different LArSoft modules

What's in your output file? (2)

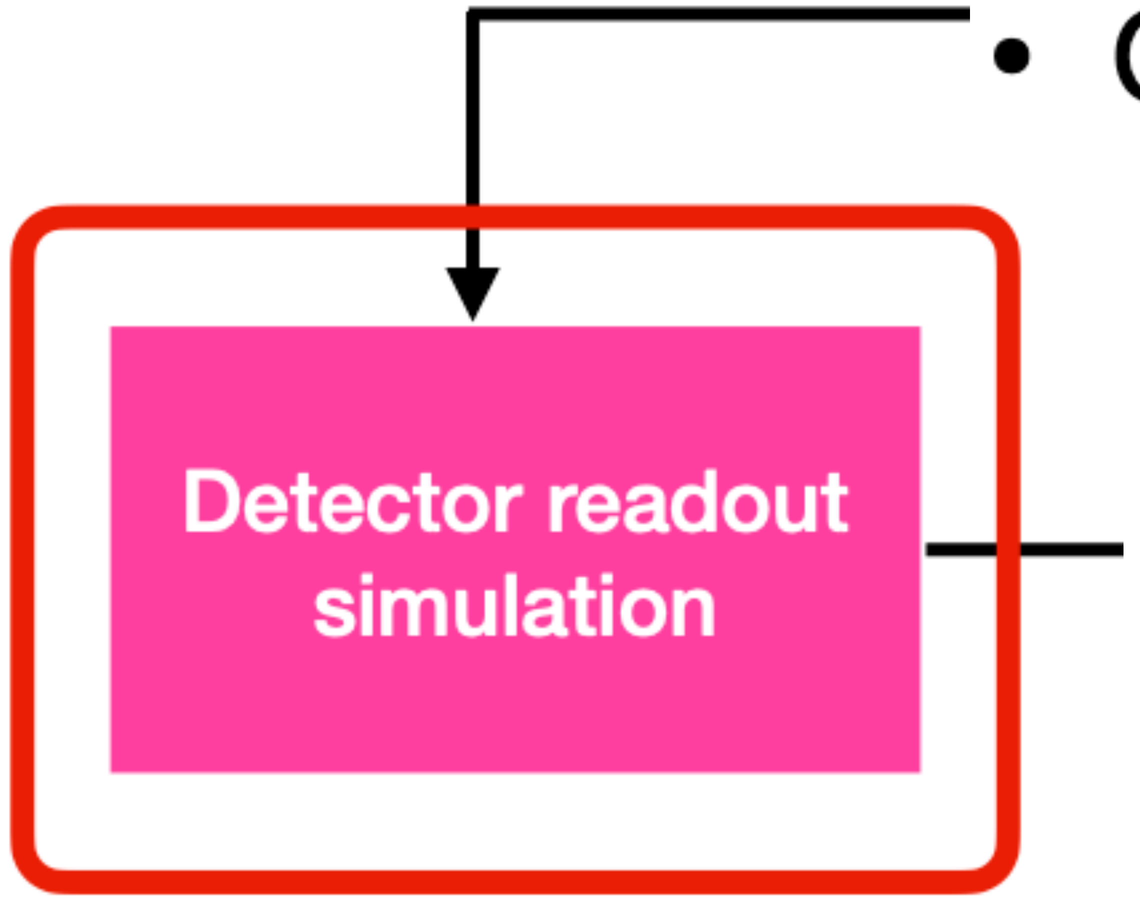
- `simb::MCTruth` objects from previous stage.
- New collection of `simb:MCParticle` for particles created during propagation.
- Collections of `sim::SimEnergyDeposit` containing the energy depositions
- Collections of `sim::SimChannel` (wires), `sim::SimPhotons` (optical detectors) and `sim::AuxDetSimChannel` (auxiliary detectors).
  - Contains electrons (photons) reaching the wires (optical detectors) as a function of time, connected to the generated particle that produced them
- With refactored LArG4, you can have more/different data products coming from the plugins.

Step 4: make some noise!



Step 4: make some noise!

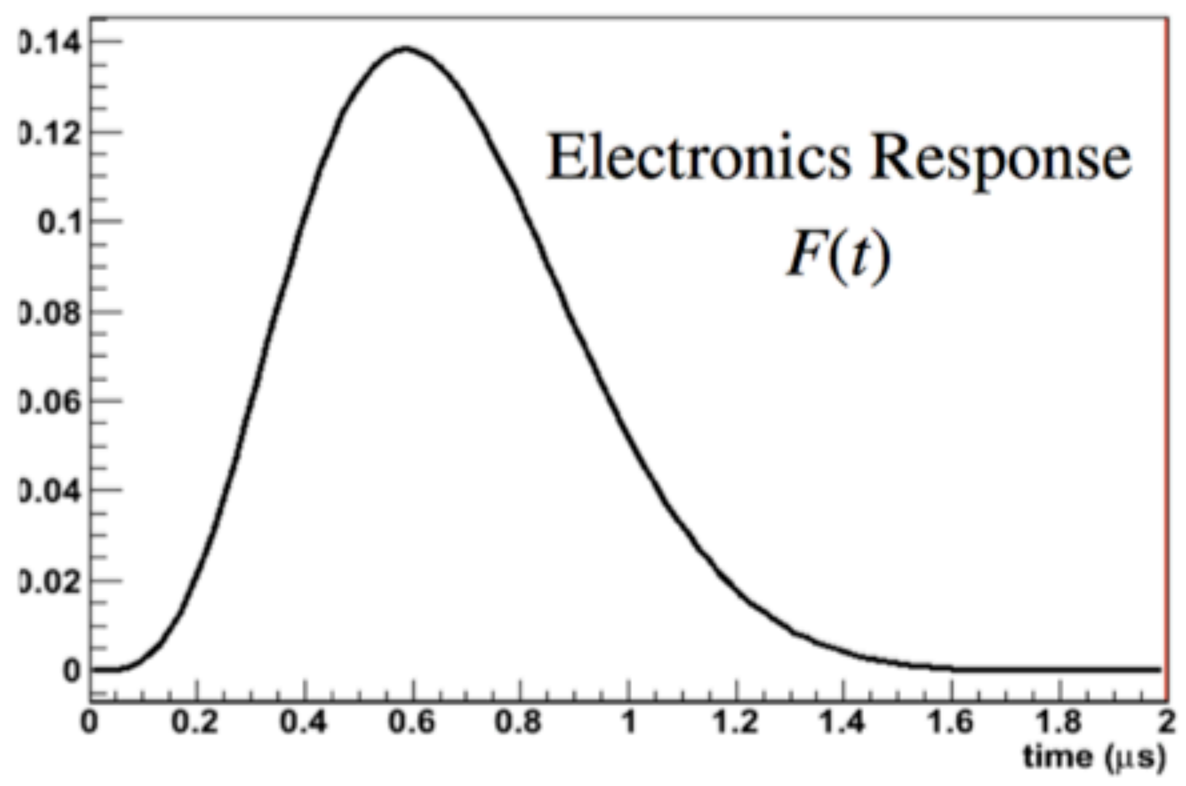
- Transforms the physics information (electrons and photons) into digitised detector response
- Includes the simulation of electronic noise and shaping
- Output is detector-like raw data



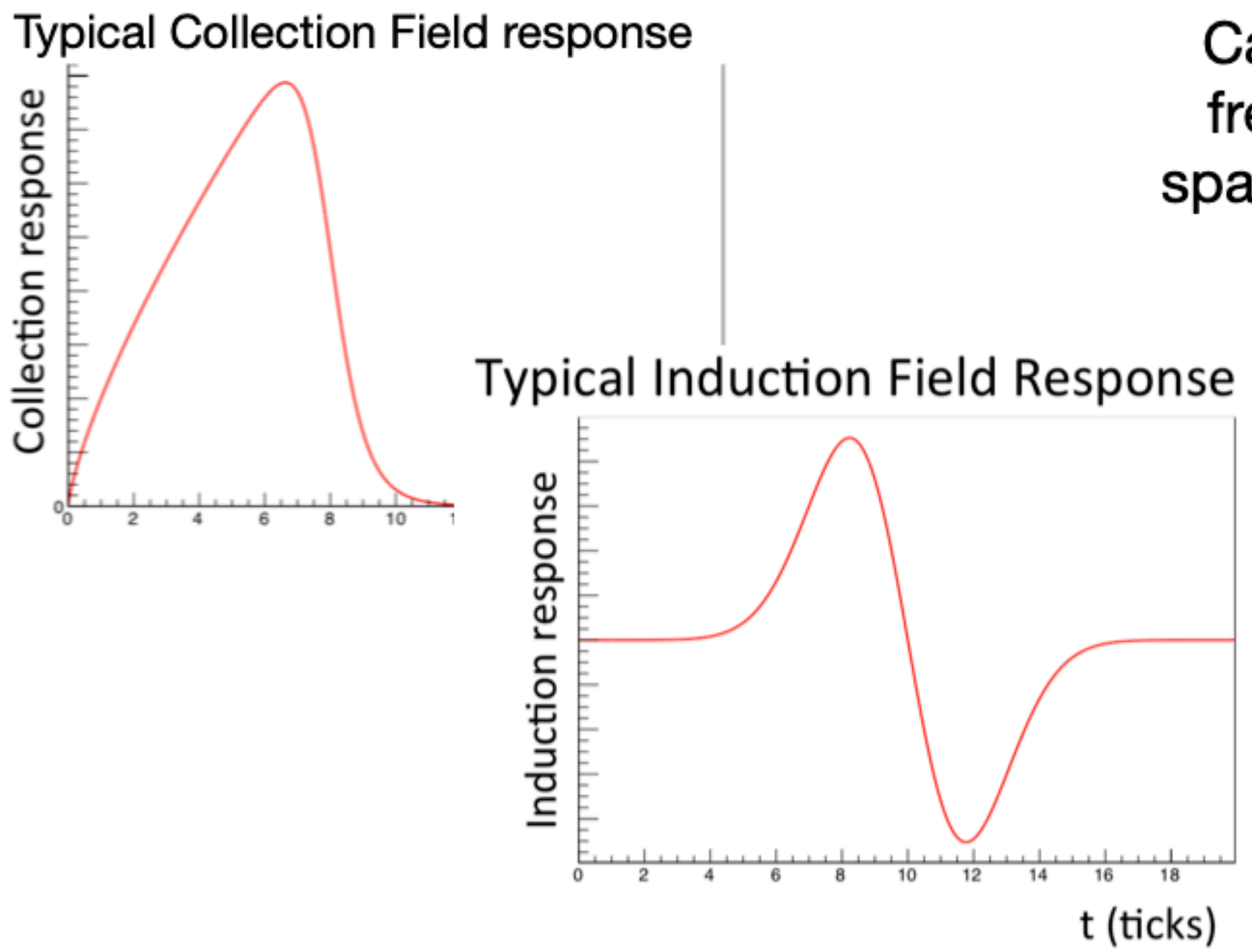
Look for detsimmodules_sbnd.fcl

Step 4: make some noise!

Electronics response function ⊗ Field shape + Noise



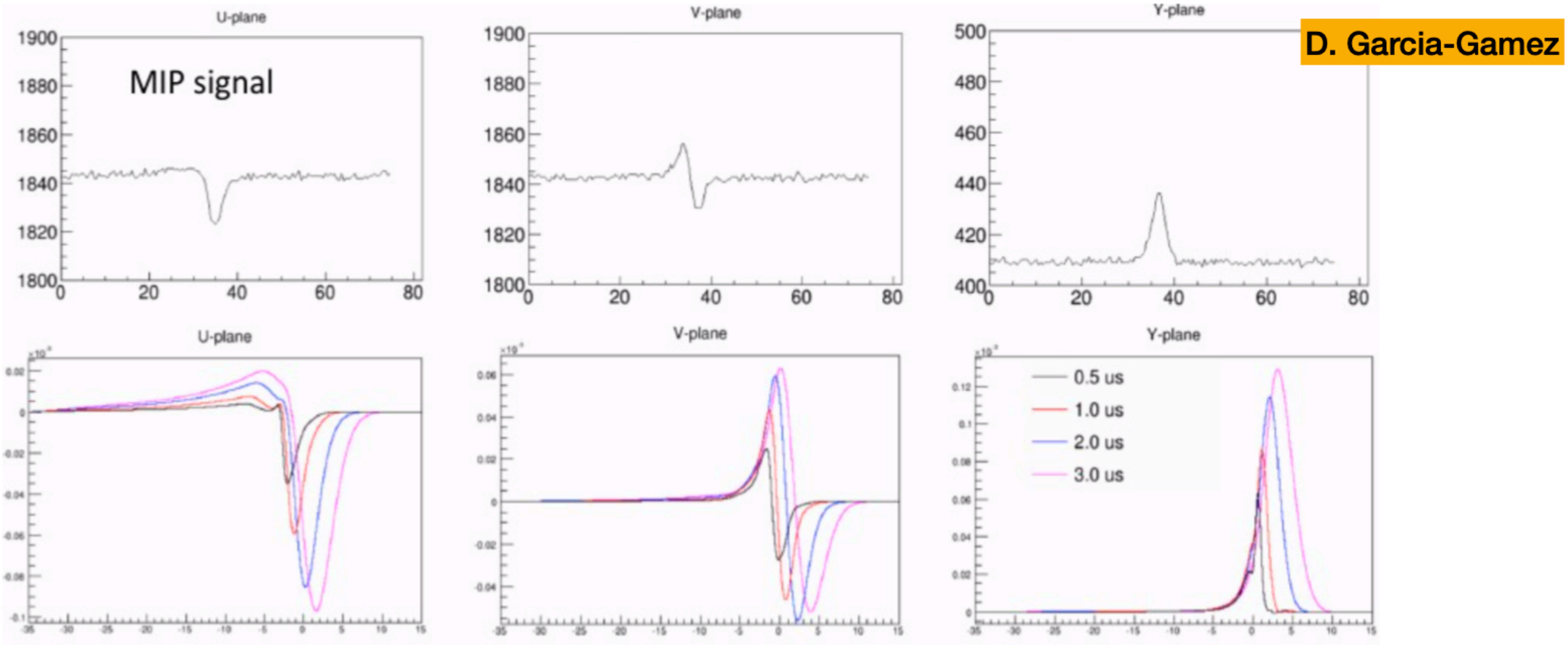
Depends on gain and shaping time



Response to channels to drifting electrons as a function of time

Can be inserted as a histogram (of freq. spectrum), generated in freq. space or with Gaussian distribution in time-domain

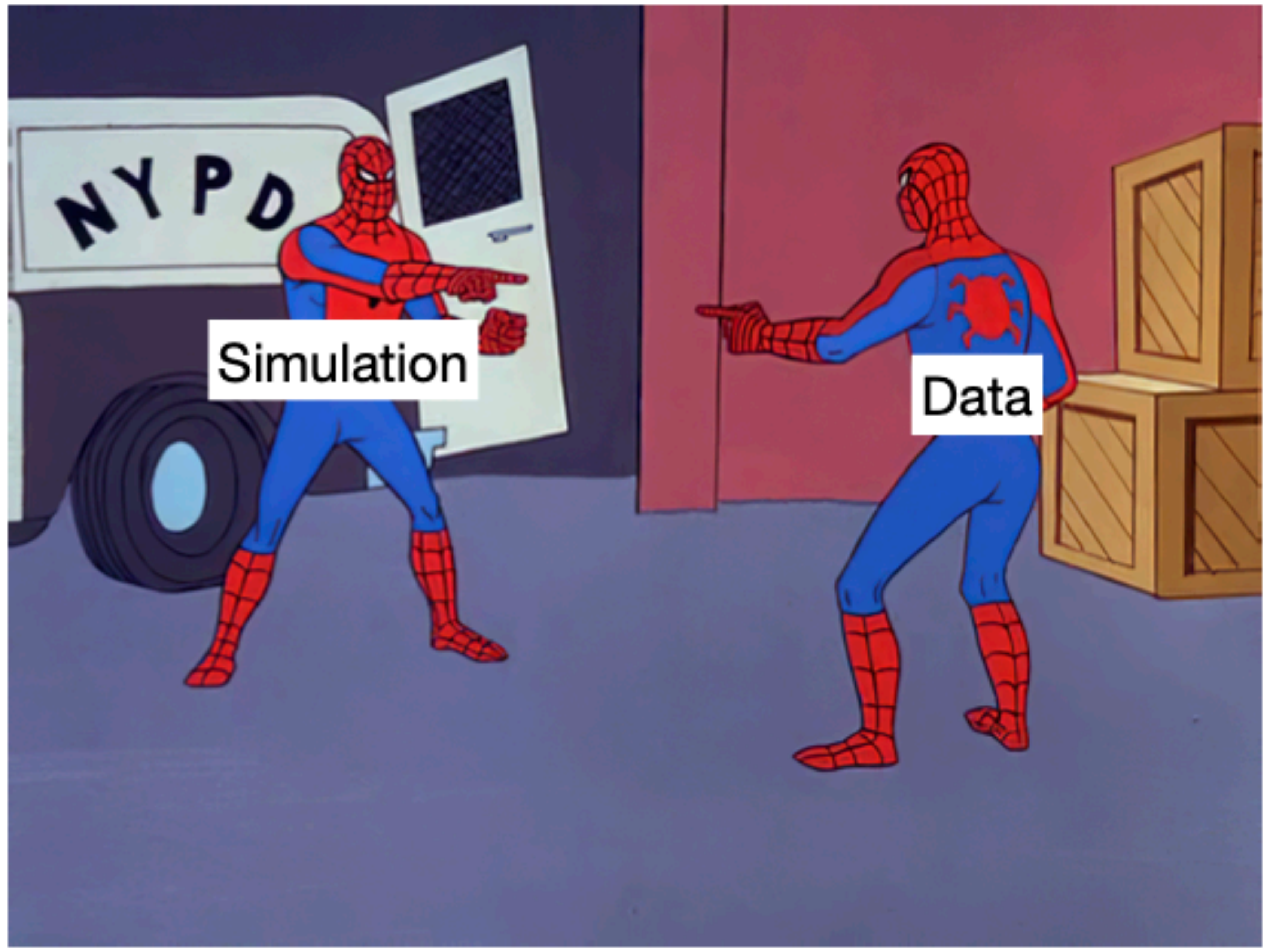
Step 4: make some noise!



Digitised signal after the ADC = ionisation signal convoluted with the detector and electronics response functions then digitised at a fixed frequency

What's in your output file? (3)

- Objects from the previous stages
- Collection of `raw::RawDigit` and `raw::OpDetWaveform` containing the data-like digitised waveforms



Summary

- Simulation in LArSoft is composed of many steps.
  - It can be scary but you'll learn!
- Offers a lot of possibilities.
- LArSoft is an ever-changing landscape, so you'll have to keep track of new developments.

Now, let's generate some events!

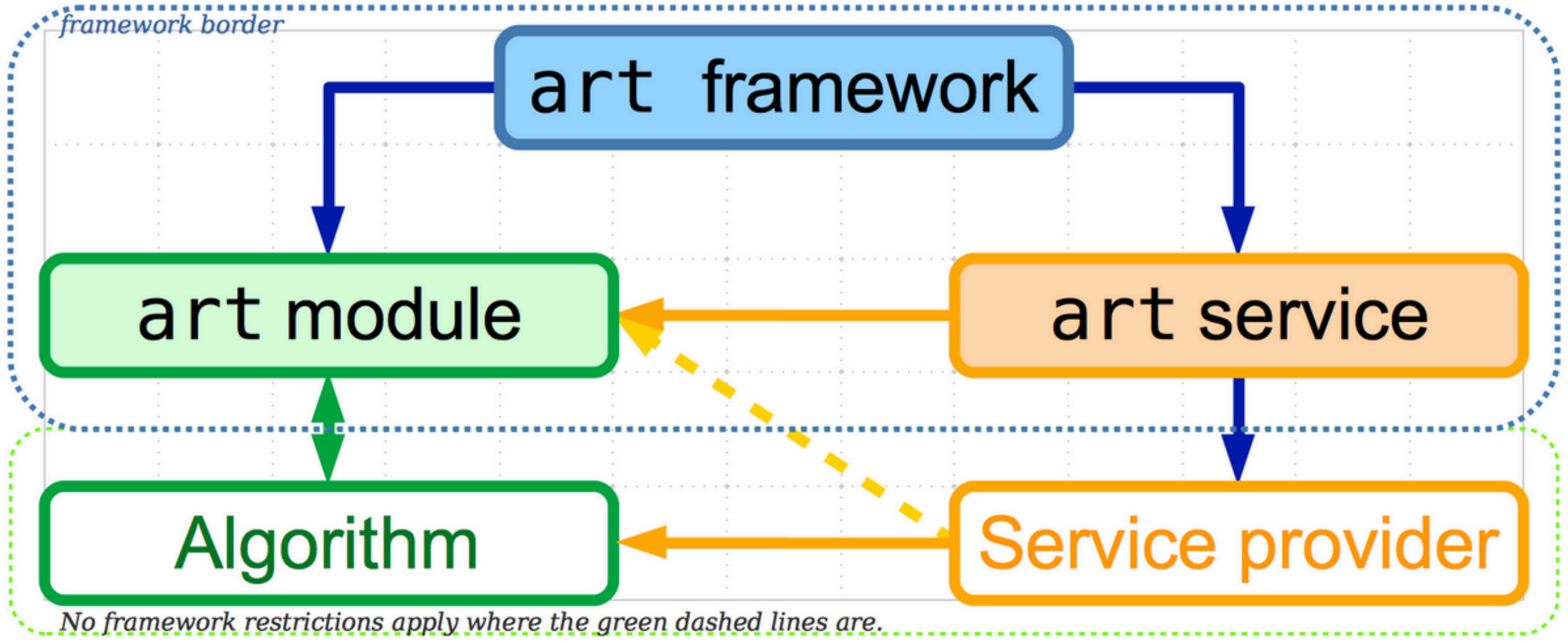
Additional resources

- **LArSoft website:** https://larsoft.org
- **LArSoft wiki:** https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki
- **LArG4 wiki:** https://cdcvs.fnal.gov/redmine/projects/larg4/wiki
- **List and documentation of LArSoft data products:** https://larsoft.org/important-concepts-in-larsoft/data-products
- **Refactored LArG4**: https://indico.fnal.gov/event/18681/contributions/48530/attachments/30244/37222/Dune.pdf
- **Geant4 website:** https://geant4.web.cern.ch

Backup

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

Event generators: Single Particle Gun

- Used to generate individual particles or very simple interactions
- You can define the particle type (PDG code), position, momentum and their how they vary (uniform, gaussian)
- There is an option to run with different/multiple particles either randomly between events or within the same event.
  - This is a bit tricky because you need to specify parameters for all particles. But there is a trick: you can ask LArSoft to "PadOutVectors". Your array then needs to be 1 or N particles (where N is max number)

```
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

argoneut_singlep: @local::standard_singlep

microboone_singlep: @local::standard_singlep
microboone_singlep.Theta0YZ: [ 0.0 ]  # beam is along the z axis.
microboone_singlep.X0:   [125]        # in cm in world coordinates, ie x = 0 is at the wire plane
microboone_singlep.Z0:   [50]         # in cm in world coordinates
```

larsim/EventGenerator/singles.fcl

Step 4: make some noise!

detsimmodules.fcl

```
standard_simwire:
{
  module_type:         "SimWire"
  DriftEModuleLabel:   "largeant"
  NoiseFact:            0.0132      # Noise Scale
  NoiseWidth:          62.4         # Exponential Noise width (kHz)
  LowCutoff:            7.5         # Low frequency filter cutoff (kHz)
  FieldBins:           75
  Col3DCorrection:     2.5
  Ind3DCorrection:     1.5
  ColFieldRespAmp:     0.0354
  IndFieldRespAmp:     0.018
  ShapeTimeConst:      [ 3000., 900. ]
  CompressionType:     "none"
}
```

```
microboone_simwire:
{
  module_type:         "SimWireMicroBooNE"
  DriftEModuleLabel:   "largeant"
  NoiseFact:           0.0132       #Noise Scale
  #NoiseFact:           0.15       #Noise Scale to use with histogram
  NoiseWidth:          62.4         #Exponential Noise width (kHz)
  LowCutoff:           7.5          #Low frequency filter cutoff (kHz)
  CompressionType:     "none"       #could also be none
  GetNoiseFromHisto:    false
  NoiseFileFname:       "uboone_noise_v0.1.root"
  NoiseHistoName:       "NoiseFreq"

}
```

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