

How Simulation (particularly of Scintillation Light) Works in LArSoft

Andrzej Szelc

(based somewhat on slides
by Diego Garcia-Gamez)

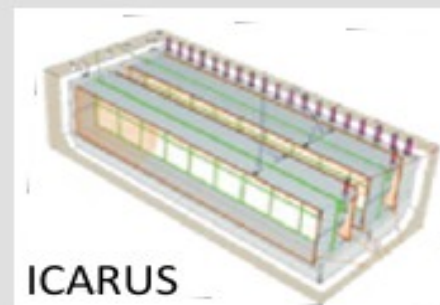
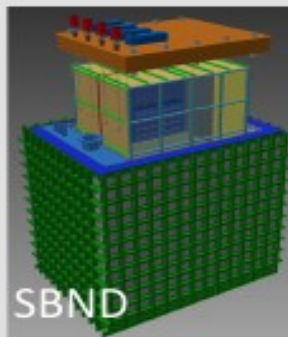
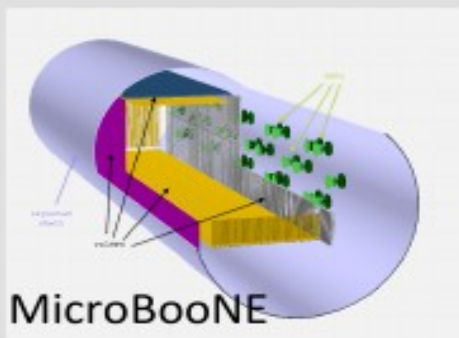
Simulation in LArSoft

- Its job is to simulate and reconstruct events from LArTPCs (as you've seen in the previous slides tutorials).
- The simulation usually runs in steps, with each step providing data products that can be used by the subsequent step.
- We are supposed to be focusing on scintillation light, but I will do a quick run through the TPC part and what goes before, because it gets used.
- I will then talk about details of how scintillation works and how you can tweak things.
- Discussion of electronics effects will be in the reconstruction session (even though it is simulation).

Geometry

<https://cdcv.s.fnl.gov/redmine/projects/larsoftsvn/wiki/Geometry>

- ✓ Each detector just needs to add a new geometry description



etc.

(ArgoNeut, 35t prototype, LArIAT, protoDUNE, DUNE)

- ✓ Simulation/Reconstruction knows how to access different geometries, but not dependent on any one
- ✓ Uses gdml (Geometry Description Markup Language)
- ✓ Detectors have two versions of Geometry (with wires and _nowires):
 - Former is used to determine wire location and properties (angles, pitch)
 - Latter is actually used in simulation (save time and memory)

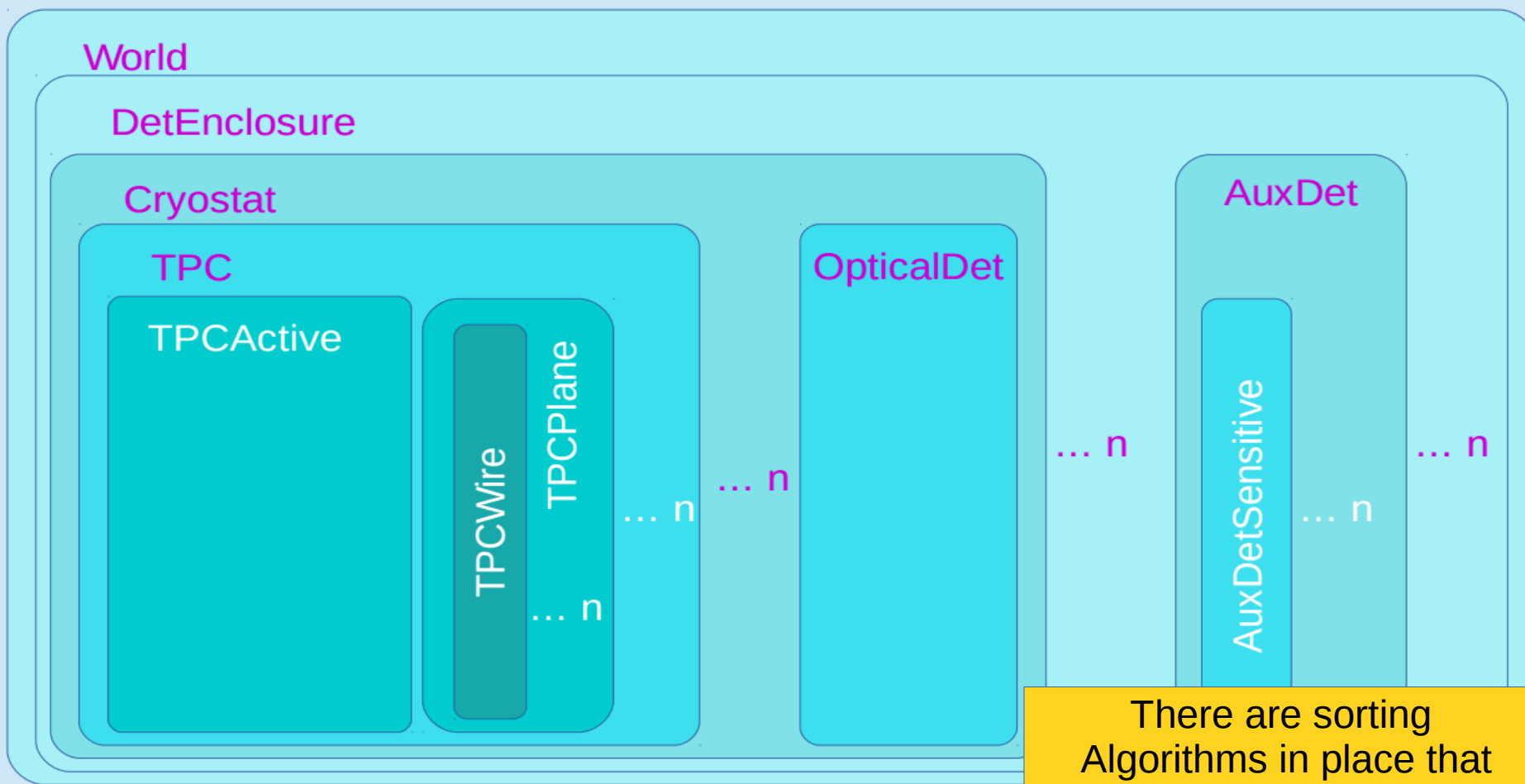
D. Garcia-Gamez

Geometry model in LArSoft

The geometry description is hierarchically organized:

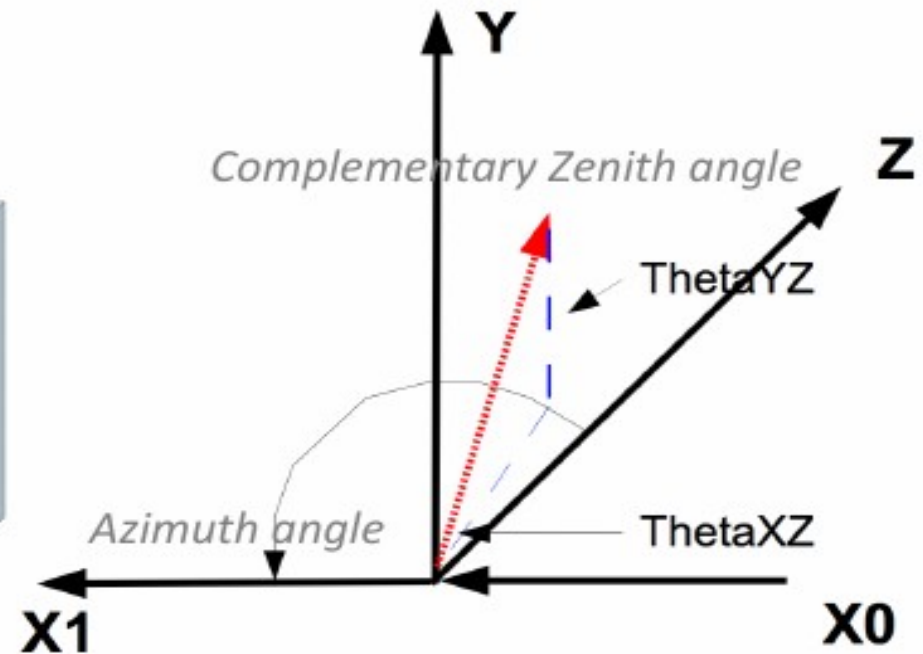
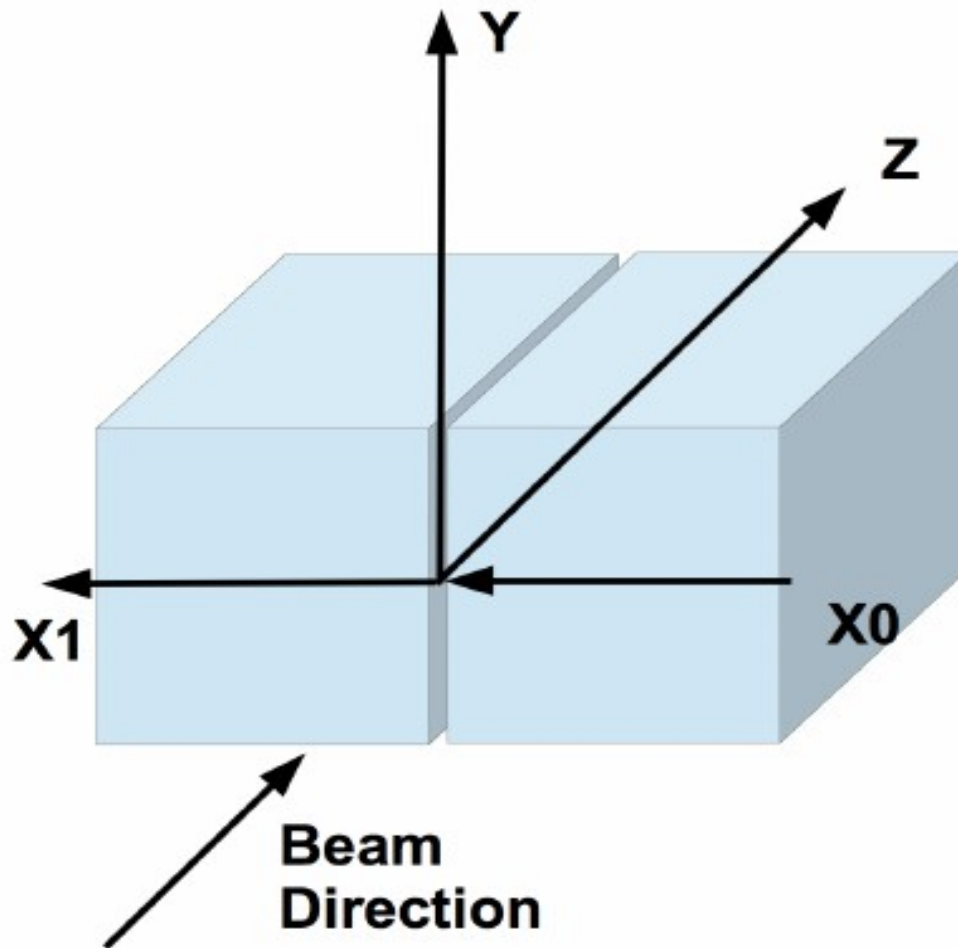
https://cdcv.s.fnal.gov/redmine/projects/larsoft/wiki/Geometry_Package

Hierarchy of geometry volumes



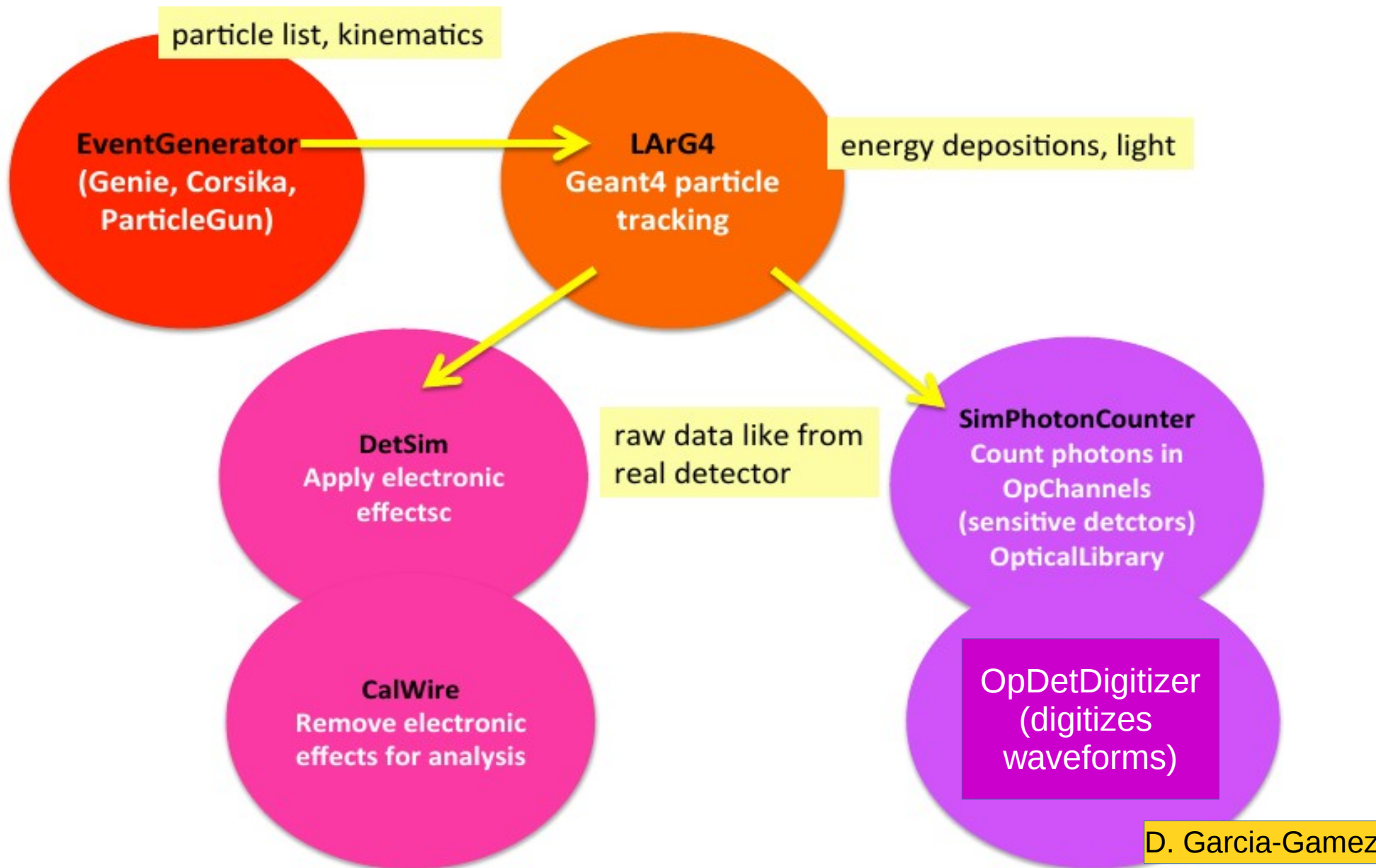
There are sorting Algorithms in place that Determine which one goes first in the code

Coordinate System



For all detectors: Z increases in the direction of the beam, Y increases away from the center of the Earth, and X increases so as to make a right handed coordinate system

Simulation flowchart



D. Garcia-Gamez

Event Generator

- First step in generating particles in LarSoft (majority of cases). Different cases, all 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 files generated by someone else...
- Each generator will create a collection of `simb::MCTruth` objects, which will be picked up by Geant4 and simulated through the detector. Main info: PDG, position, momentum (there is more, but these are generally enough)

SingleGen (single particle mode)

- Single Particle Gun equivalent in LArSoft. Very useful for debugging code and understanding simple features of what's going on.
- You can define the particle type (PDG code), position, momentum and how they vary (uniform, gaussian).
- There is an option to run with different/multiple particles either randomly between events or in the same event.
 - This is a bit tricky, because you need to specify parameters for all particles (tiresome). But there is a trick – you can ask larsoft to “PadOutVectors”. Your arrays then need to be 1 or N particles (where N is max number).

larsim/EventGenerator/singles.fcl

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
}

```

p, Campinas

21/1

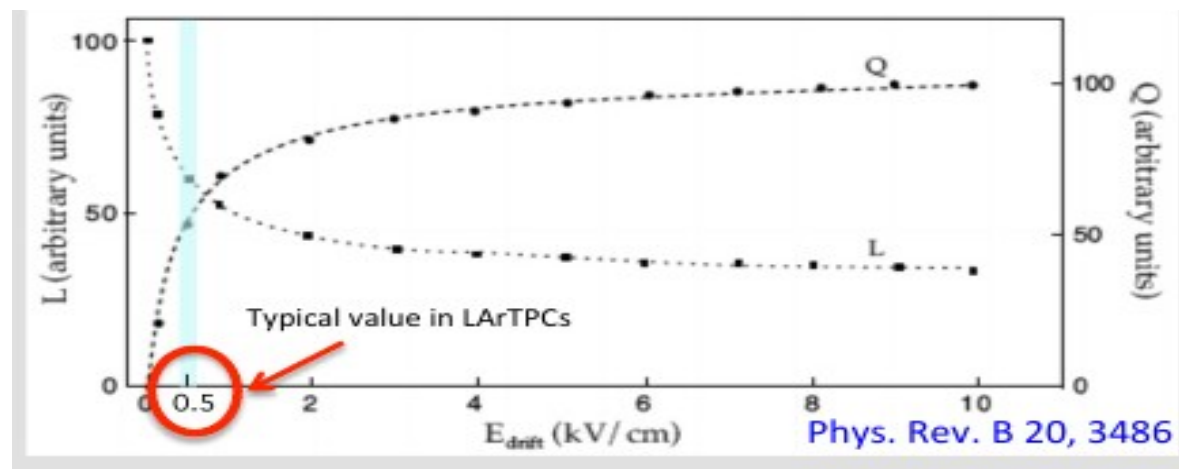
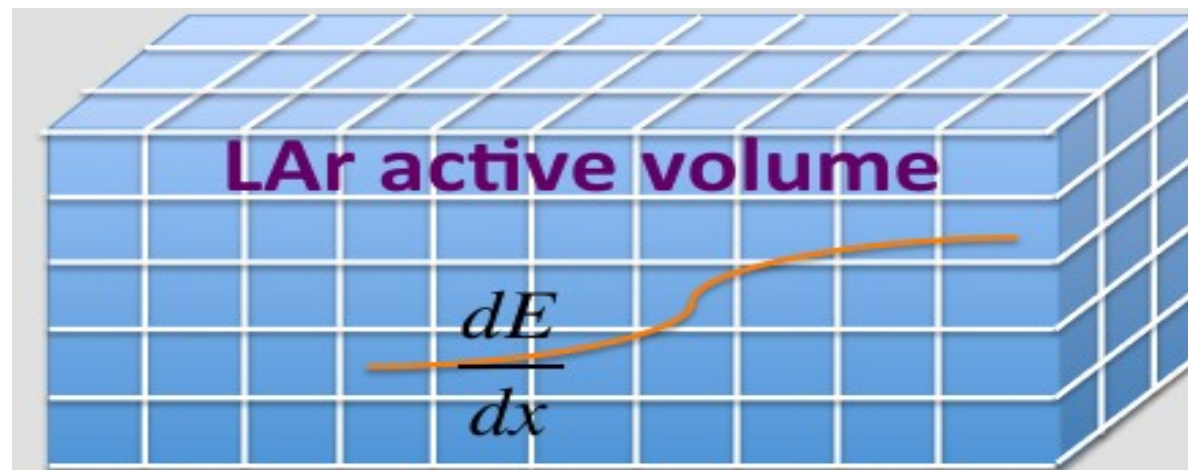
- 10

LArG4 - Geant4

- LArSoft uses Geant4 mechanics to simulate the particle propagation in the LArTPC.
- It does this through a module called LArG4 which is a wrapper that tells Geant4 to run things and harvests the results.
- It sets most of the relevant parameters needed by G4 (that you may have seen in the Geant4 tutorial earlier).
- It does so through .fcl files.

Simulation strategy

- Use Geant4 physics list (QGSP_BERT) to simulate particle propagation in materials.
- Harvest the energy depositions (dE/dx)
- Perform the charge drift and light scintillation via LArSoft (most of the time).



Electron “drift”

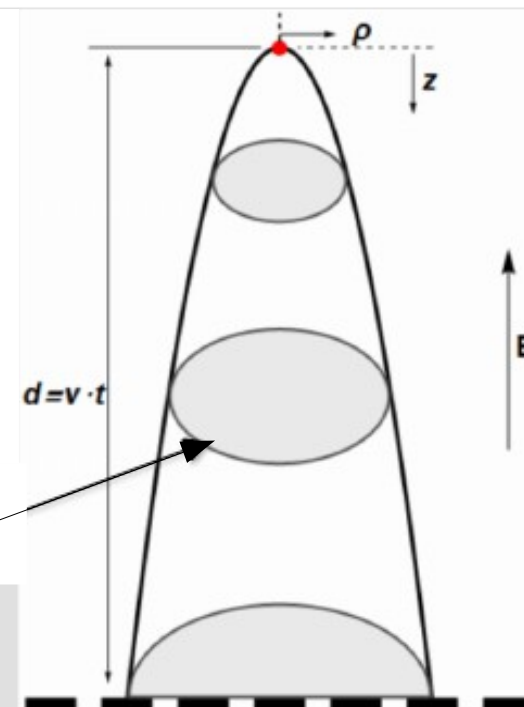
`LarVoxelReadout::DriftIonizationElectrons()`
 $dE/dx \rightarrow$ recombination, lifetime correction [impurities] \rightarrow nelectrons (charge)

The way this then works is:

- Electrons are split in groups (default 600)
- They are projected to a Y,Z position at the position of the wire planes
- This 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 to the x coordinate.

For 3.6 meter drift (DUNE): 1.8 mm longitudinal and 2.5 mm transverse to the electric field at 500 V/cm.

$$\sigma_{L/T} = \sqrt{2 D_{L/T} X_{\text{drift}} / v_X}$$



These charge clusters are then saved in `sim::SimChannel` objects (a protowire, if you will). These contain `sim::IDE` objects – these contain the information about the true deposition (position, charge, trackID)

DetSim – electronics etc...

a bunch of charge
(electrons) assigned to
channels (wires)

`SimWire<Experiment>` (simulates signal on a wire in the TPC)
but is outsourced to

`Utilities::SignalShaping<Experiment>` (all the detector
specific signal
shaping)

electronics response function \otimes field shape + noise

expected electronics
signal from the
detector

Each experiment should
implement its own version
of this module to simulate
electronics response

D. Garcia-Gamez

Glossary of ART objects or “who makes what”

- **Generator:** `std::vector <simb::MCTruth>`
- **LArG4:** `std::vector <simb::MCParticle>`
`std::vector <sim::SimChannel>`
`MCTruthParticle Info`
- **DetSim:** `std::vector <raw::RawDigit>`
- **CalWire:** `std::vector <recob::Wire>`

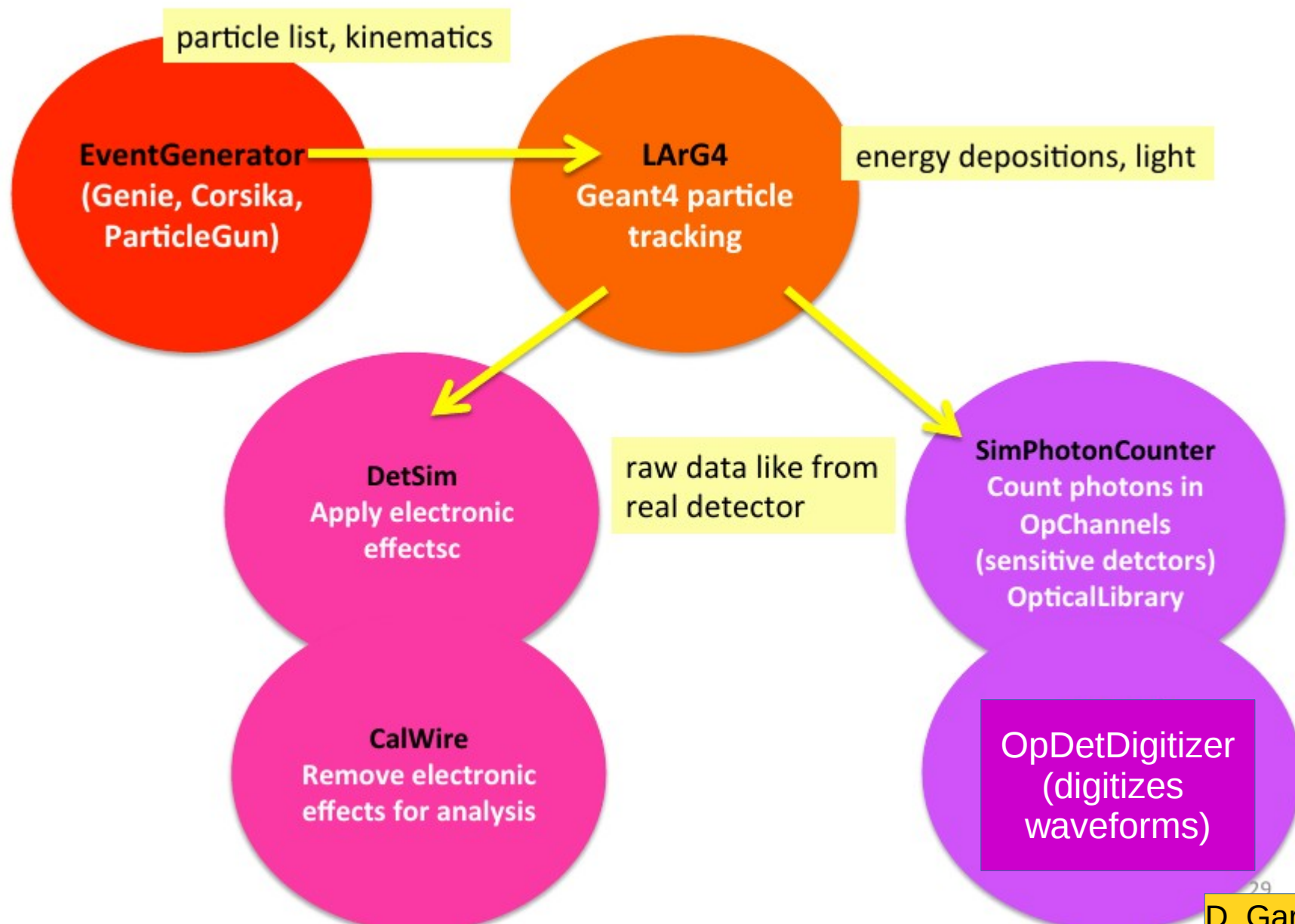
Begin processing the 5th record. run: 1 subRun: 0 event: 5 at 29-Aug-2018 07:14:19 PYT

PRINCIPAL TYPE: Event

PROCESS NAME	MODULE_LABEL..	PRODUCT INSTANCE NAME	DATA PRODUCT TYPE.....	SIZE
SinglesGen..	largeant.....	<code>std::vector<sim::SimPhotonsLite>.....</code>	..60
SinglesGen..	generator.....	<code>std::vector<simb::MCTruth>.....</code>	..1
SinglesGen..	largeant.....	<code>std::vector<simb::MCParticle>.....</code>	..368
SinglesGen..	opdigi.....	<code>std::vector<raw::OpDetWaveform>.....</code>	..167
SinglesGen..	daq.....	<code>std::vector<raw::RawDigit>.....</code>	4173
SinglesGen..	largeant.....	<code>art::Assns<simb::MCTruth,simb::MCParticle,sim::GeneratedParticleInfo>.....</code>	..368
SinglesGen..	rns.....	<code>std::vector<art::RNGsnapshot>.....</code>	..4
SinglesGen..	largeant.....	<code>std::vector<sim::SimChannel>.....</code>	2143
SinglesGen..	largeant.....	<code>std::vector<sim::AuxDetSimChannel>.....</code>	..512
SinglesGen..	opdigi.....	<code>std::vector<sim::OpDetDivRec>.....</code>	..60
SinglesGen..	largeant.....	<code>std::vector<sim::OpDetBacktrackerRecord>.....</code>	..60
SinglesGen..	TriggerResults	<code>art::TriggerResults.....</code>	..-

Total products (present, not present): 12 (12, 0).

Simulation flowchart



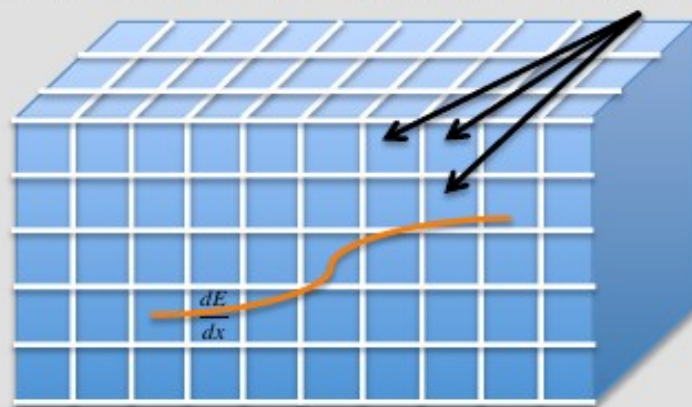
Optical Simulation in LArSoft

- Liquid argon is very good at scintillating – 24000 photons/MeV of deposited energy at 500V/cm, 40000/MeV at zero field.
- Simulating this for say a few hundred MeV muon in a large detector is **very** slow and **very** cpu consuming.
- We need to be clever – one way to do this is to use an “optical library”. A lookup table that tells us what is the probability of detecting a photon given its position in x,y,z.
- We still need to use the full optical simulation to generate the lookup library. But only once. Well, only once per new configuration.

How does Fast OpSim work?

- i. The G4Scintillation \rightarrow OpFastScintillation process models the production of some number of photons along a particle step (determined by particle type + scintillation yield + quenching) in the detector:

Detector volume divided in voxels (3D pixels)

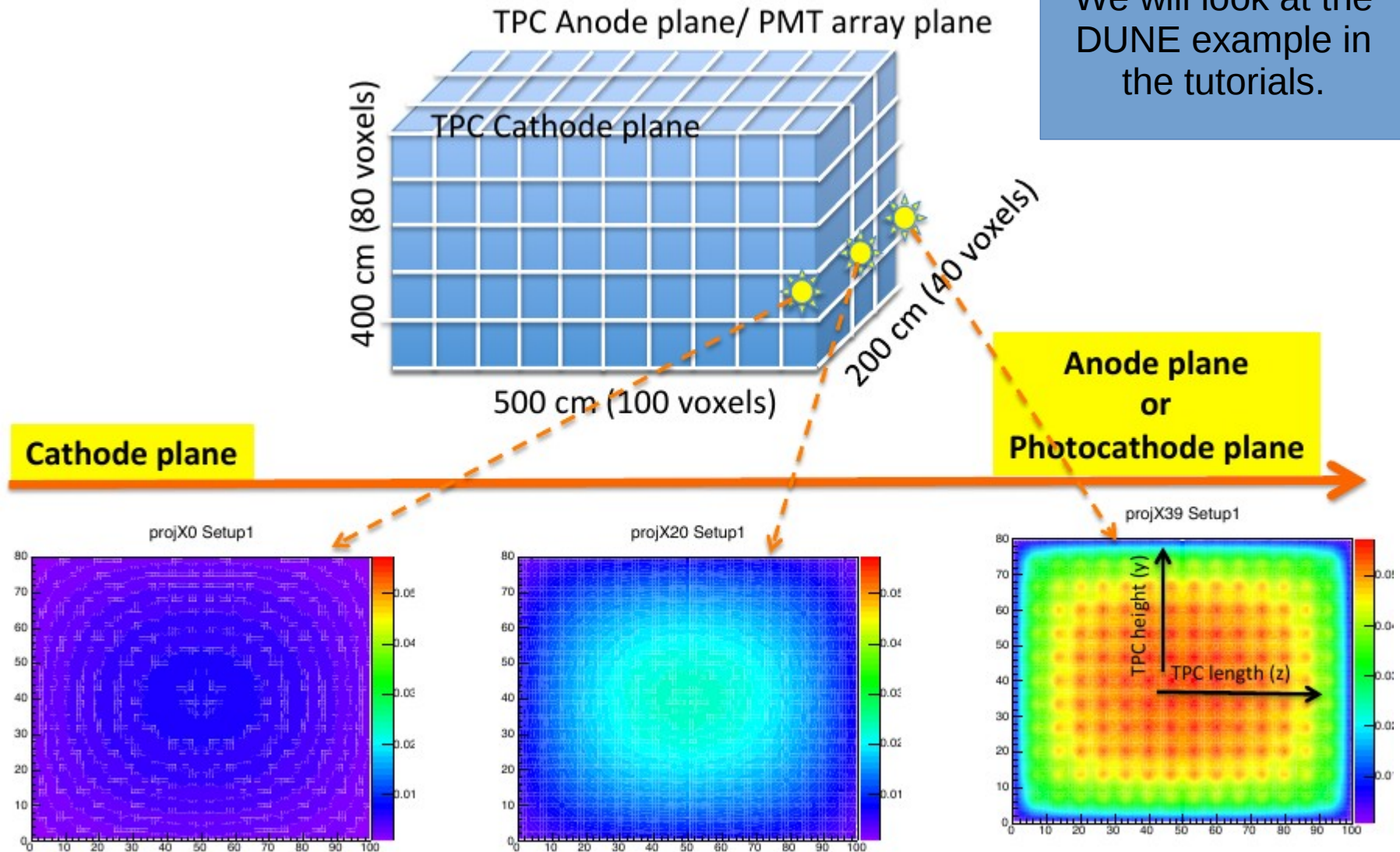


$$\langle N \rangle_{PMT-hits} = \left(\frac{dE}{dx_{step}} \cdot Length_{step} \right) \cdot LY \cdot visibility_{step}^{PMT}$$

- ii. Arrival time distribution of the photons is still needed: Storing arrival time histograms for every voxel/PMT pair takes up memory

Example of a Light Library

We will look at the DUNE example in the tutorials.



Relevant Pieces of Code

Most of the relevant code for scintillation simulation lives in the larsim repository

• Full Optical Simulation

- LArG4/OpticalPhysics.cxx
- LArG4/OpBoundaryProcessSimple.cxx
- LArG4/OpDet*

• Fast Optical Simulation

- LArG4/OpFastScintillation.cxx
- PhotonPropagation/PhotonVisibilityService*
- PhotonPropagation/PhotonLibrary.cxx

How to turn on OpFast?

- `services.LArG4Parameters.UseCustomPhysics: true`
`services.LArG4Parameters.EnabledPhysics: ["Em",
 "FastOptical",
 "SynchrotronAndGN",
 "Ion",
 "Hadron",
 "Decay",
 "HadronElastic",
 "Stopping",
 "NeutronTrackingCut"]`
- The full Optical simulation is called `"G4Optical"`

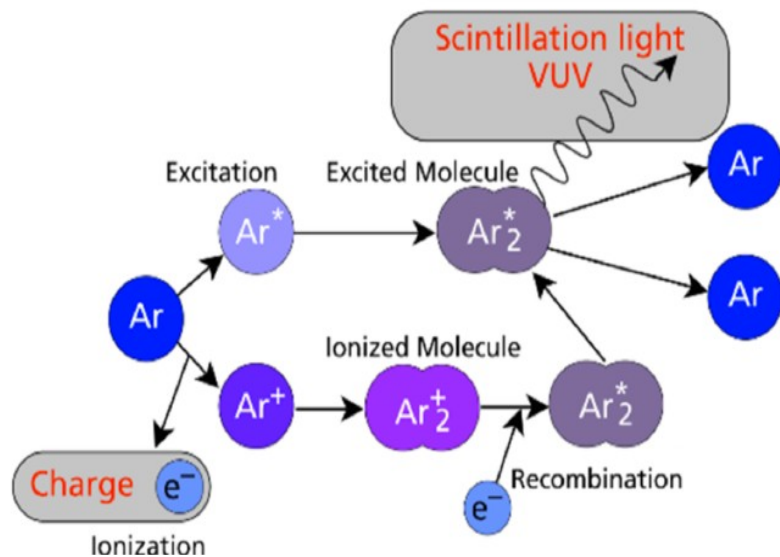
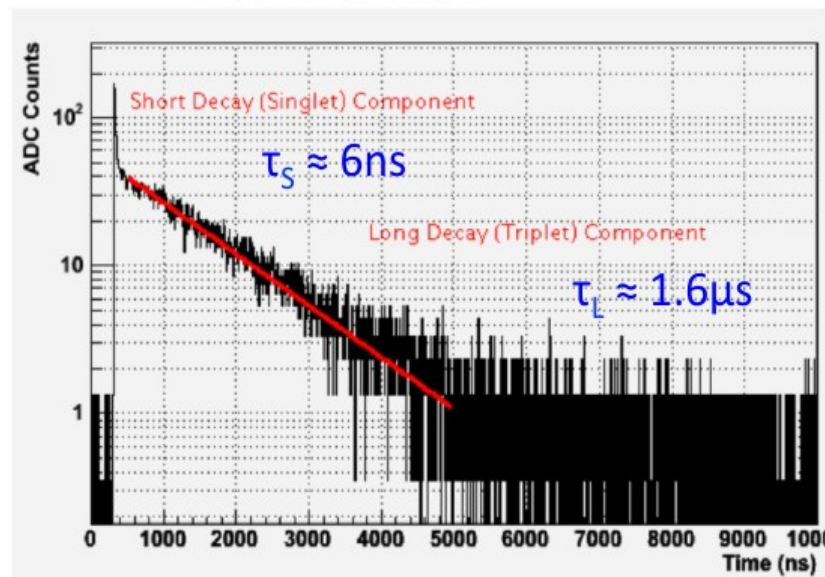
Tunable knobs in Optical Simulation

- Timing Constants
- Ionization/Scintillation yields
- Rayleigh Scattering
- Timing Parametrization
- OnePhoton vs LitePhotons
- Material Properties
- Adding Reflected Light

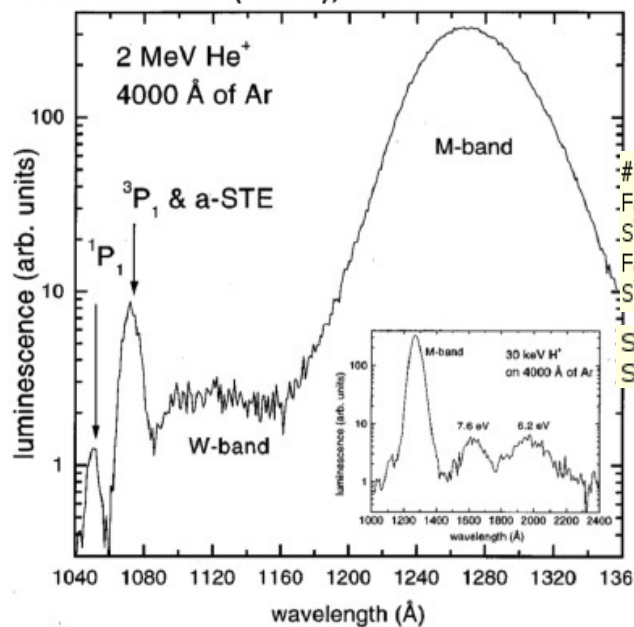
Most of these are tunable
via FHiCL parameters

Energy spectrum and timing Constants

2010 JINST 5 P06003



Ph. Rev. B 56 (1997), 6975



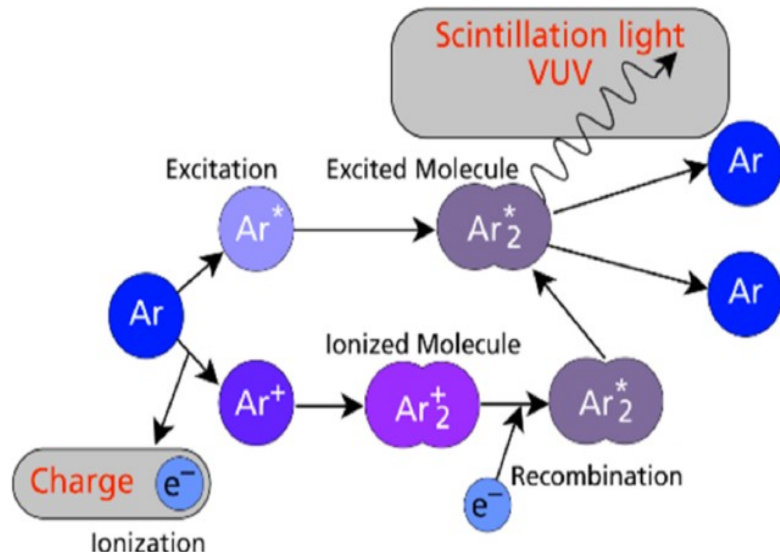
• `lardataalg` → `larproperties.fcl`

```
# Fast and slow scintillation emission spectra, from [J Chem Phys vol 91 (1989) 1469]
FastScintEnergies: [ 6.0, 6.7, 7.1, 7.4, 7.7, 7.9, 8.1, 8.4, 8.5, 8.6, 8.8, 9.0, 9.1, 9.4, 9.8, 10.4, 10.7]
SlowScintEnergies: [ 6.0, 6.7, 7.1, 7.4, 7.7, 7.9, 8.1, 8.4, 8.5, 8.6, 8.8, 9.0, 9.1, 9.4, 9.8, 10.4, 10.7]
FastScintSpectrum: [ 0.0, 0.04, 0.12, 0.27, 0.44, 0.62, 0.80, 0.91, 0.92, 0.85, 0.70, 0.50, 0.31, 0.13, 0.04, 0.01, 0.0]
SlowScintSpectrum: [ 0.0, 0.04, 0.12, 0.27, 0.44, 0.62, 0.80, 0.91, 0.92, 0.85, 0.70, 0.50, 0.31, 0.13, 0.04, 0.01, 0.0]

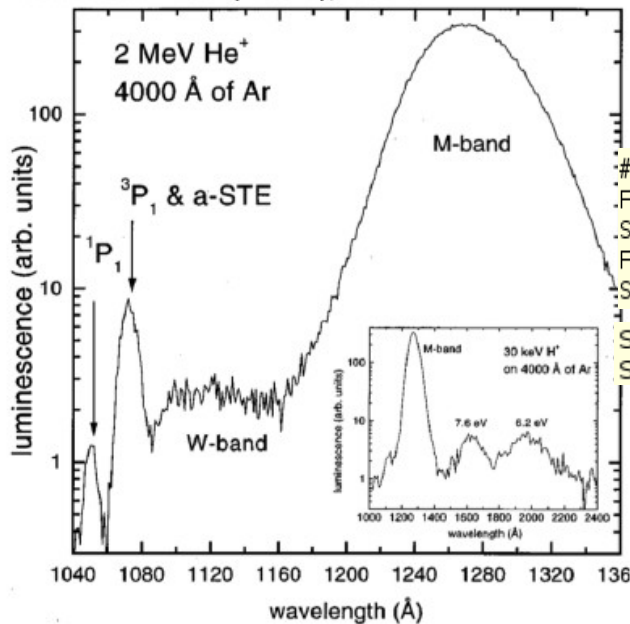
ScintFastTimeConst: 6.      # fast scintillation time constant (ns)
ScintSlowTimeConst: 1590.  # slow scintillation time constant (ns)
```

Found preparing these slides.
Is everything wrong?!

Energy spectrum and timing Constants



Ph. Rev. B 56 (1997), 6975



larsim → lightsource.fcl

Fast and slow scin
FastScintEnergies:
SlowScintEnergies:
FastScintSpectrum:
SlowScintSpectrum:
ScintFastTimeC
ScintSlowTimeC

```

21 standard_lightsource_scanmode:
22 {
23   module_type:      "LightSource"
24
25   SourceMode:       1      # Indicates scan mode
26
27   FillTree:         true   # Whether to make a TTree of photon information
28                           # in the TFileService
29
30   PosDist:           0      # Flags specifying random distribution
31   PDist:             1      # 0 = Uniform
32   TDist:             0      # 1 = Gaussian
33
34   P:                 9.7    # Central photon momentum in eV (arxiv:1511.07718)
35   SigmaP:            0.25   # Width of momentum distribution (arxiv:1511.07718)
36   T0:                0.0    # Central time for photon production
37   SigmaT:            0.0    # Extent in time
38   N:                 10000  # Number of photons to make per voxel
39
40   FirstVoxel:        0      # First and last voxel IDs to populate
41   LastVoxel:         -1     # (LastVoxel=-1 means run over all voxels)
42
43   UseCustomRegion:   false  # Use the voxel params from PhotonVisibilityService
44                           # (false) or those supplied below (true)
45
46   # Custom voxelization parameters
47
48   XSteps:            1
49   YSteps:            10
50   ZSteps:            20
51   RegionMin:         [ -120.0, -120.0, 0.0 ]
52   RegionMax:         [ 120.0, 120.0, 1400.0 ]
53 }
54
55

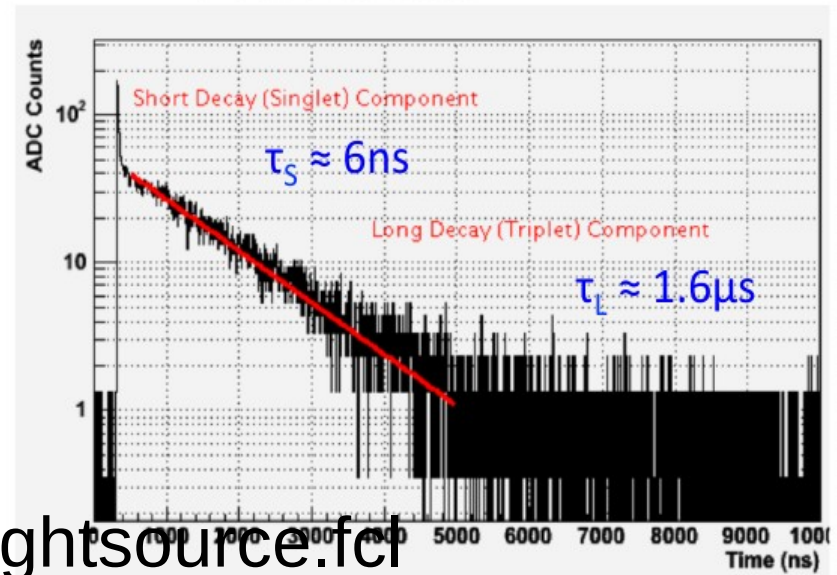
```

9.1, 9.4, 9.8, 10.4, 10.7]
9.1, 9.4, 9.8, 10.4, 10.7]
0, 0.31, 0.13, 0.04, 0.01, 0.0]
0, 0.31, 0.13, 0.04, 0.01, 0.0]
ns)
ns)

ie slides.

Fortunately not.
(if you're using the library)

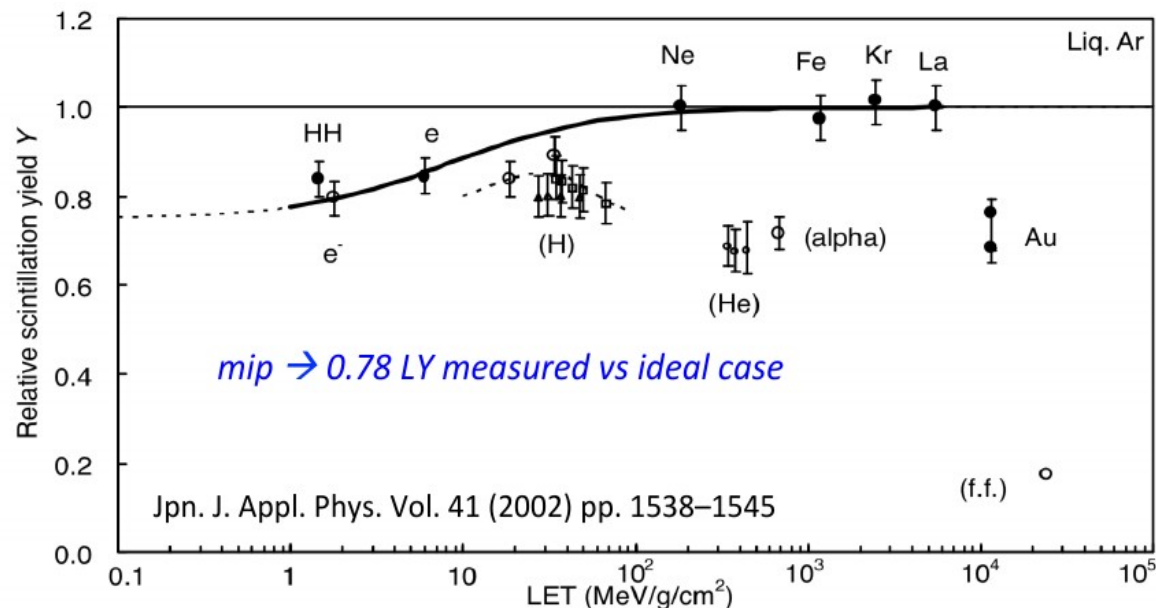
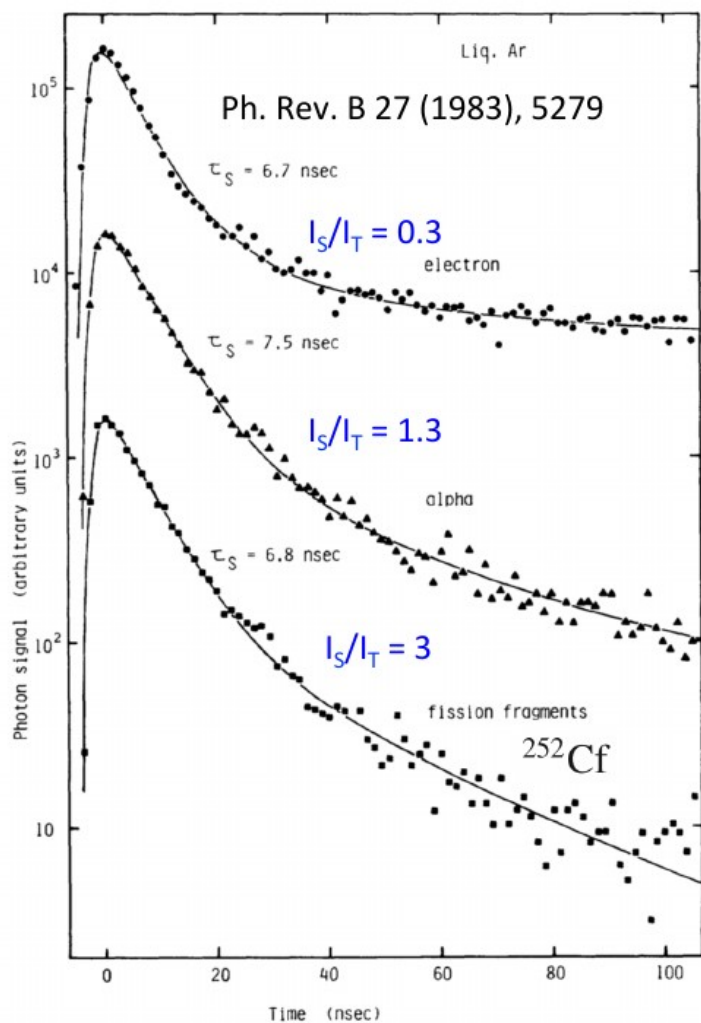
2010 JINST 5 P06003



Before we restart

- Go to your working dir. If you were logged out, then:
 - `source /cvmfs/...`
 - `source localProducts...`
 - `cd $MRB_BUILDDIR`
 - `mrbsenv`
 - `make install -j4`
- If your compilation is finished, we may also need, larsim.
- ```
$ cd srcs/
$ git clone /home/andrzej/larsoft/srcs/larsim/.git
$ mrbs uc
$ cd larsim
$ git checkout feature/andrzej_workshop
$ cd $MRB_BUILDDIR
$ mrbsenv
$ mrbs i -j8
```

# Ionization/Scintillation Yields



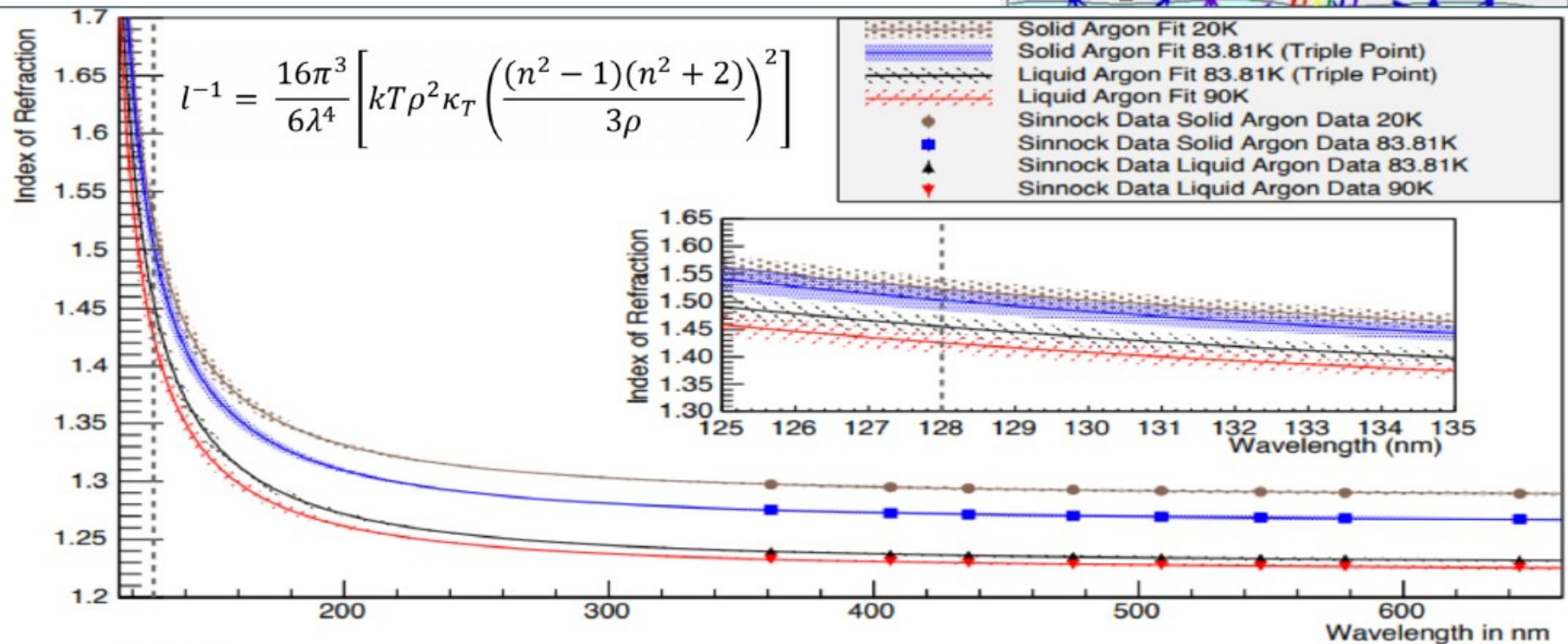
```
ScintYield: 24000. # total scintillation yield (ph/Mev)
ScintPreScale: 0.03 # Scale factor to reduce number of photons simulated
 # Later QE should be corrected for this scale
ScintYieldRatio: 0.3 # fast / slow scint ratio (needs revisitting)
ScintByParticleType: false # whether to use different yields and
 # quenching per particle in fast op sim
EnableCerenkovLight: true # whether to switch on cerenkov light (slow)

Scintillation yields and fast/slow ratios per particle type
MuonScintYield: 24000
MuonScintYieldRatio: 0.23
PionScintYield: 24000
PionScintYieldRatio: 0.23
ElectronScintYield: 20000
ElectronScintYieldRatio: 0.27
KaonScintYield: 24000
KaonScintYieldRatio: 0.23
ProtonScintYield: 19200
ProtonScintYieldRatio: 0.29
AlphaScintYield: 16800
AlphaScintYieldRatio: 0.56
```



# Rayleigh Scattering

- RS -> Elastic scattering of photon with medium of particle  $\sim 1/10$  size of the wavelength; change of angle/direction (blue colour of the sky)
- Parametric process: initial state = final state

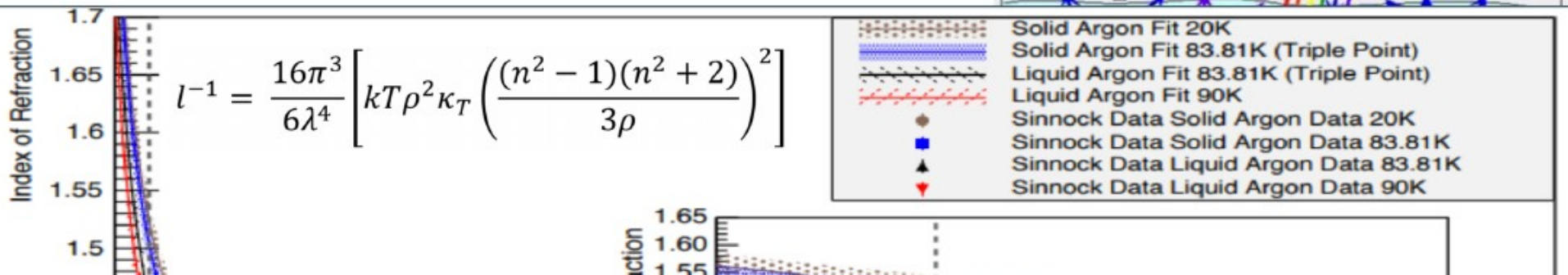


LARSOTT)

- Late different measurements suggest a larger value  $\langle \lambda_{RS} \sim 100 \text{ cm} \rangle$
- For a complete review see *sbn-doc-3590*

# Rayleigh Scattering

- RS -> Elastic scattering of photon with medium of particle  $\sim 1/10$  size of the wavelength; change of angle/direction (blue colour of the sky)
- Parametric process: initial state = final state



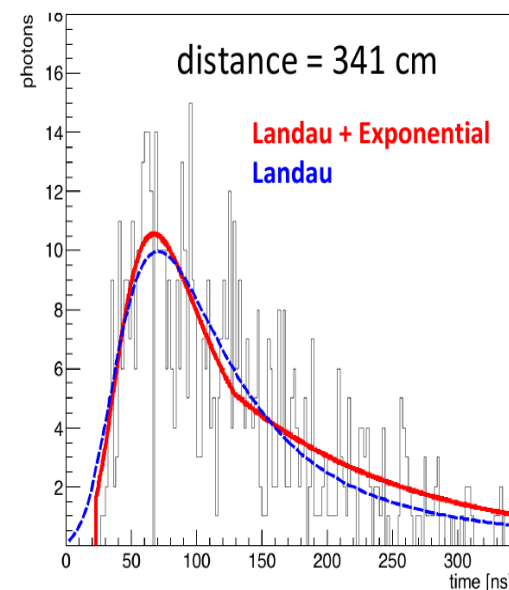
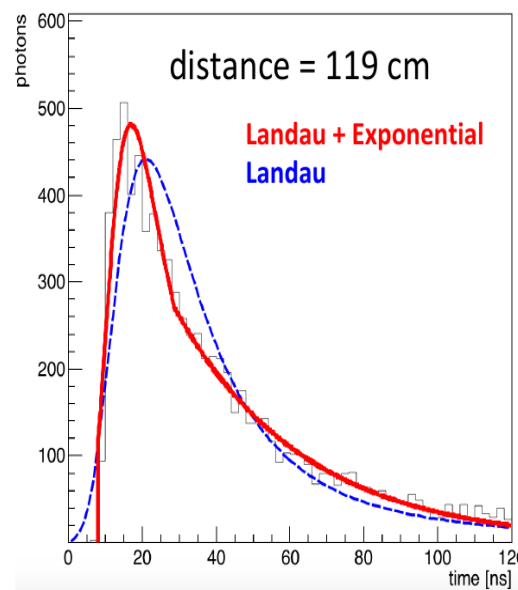
- `lardataalg` → `larproperties.fcl`

```
Rayleigh scattering length (cm) @ 90K as a function of energy (eV) from arXiv:1502.04213
RayleighEnergies: [2.80, 3.00, 3.50, 4.00, 5.00, 6.00, 7.00, 8.00, 8.50, 9.00, 9.20, 9.40, 9.50, 9.60, 9.70, 9.80, 9.90, 10.0, 10.2, 10.4, 10.6, 10.8]
RayleighSpectrum: [47923., 35981., 18825., 10653., 3972., 1681., 750.9, 334.7, 216.8, 135.0, 109.7, 88.06, 78.32, 69.34, 61.06, 53.46, 46.50, 40.13, 28.91, 19.81, 12.61, 7.20]
```

```
Refractive index as a function of energy (eV) from arXiv:1502.04213v1
RIndexEnergies: [1.900, 2.934, 3.592, 5.566, 6.694, 7.540, 8.574, 9.044, 9.232, 9.420, 9.514, 9.608, 9.702, 9.796, 9.890, 9.984, 10.08, 10.27, 10.45, 10.74, 10.92]
RIndexSpectrum: [1.232, 1.236, 1.240, 1.261, 1.282, 1.306, 1.353, 1.387, 1.404, 1.423, 1.434, 1.446, 1.459, 1.473, 1.488, 1.505, 1.524, 1.569, 1.627, 1.751, 1.879]
```

# Timing Effects

- Rayleigh scattering can affect photon arrival times.
- This is accounted for automatically in full sim, but needs to be introduced in fastSim.



larsim/PhotonPropagation  
/photpropservices.fcl

```
----- Direct/VUV component modeled with a Landau + Exponential function ----- #
parameters are parametrized as a function of the distance

Direct_landauNormpars: [7.85903, -0.108075, 0.00110999, -6.90009e-06, 2.52576e-08, -5.39078e-11, 6.20863e-14, -2.97559e-17]
Direct_landauMPVpars: [1.20259, 0.0582674, 0.000308053, -2.71782e-07, -3.37159e-10]
Direct_landauWidthpars: [0.346667, -0.00768231, 0.000211825, -3.81361e-07]
Direct_expoCtepars: [13.6592, -0.188798, 0.00192431, -1.10689e-05, 3.38425e-08, -5.20737e-11, 3.17657e-14]
Direct_expoSlopepars: [-0.57011, 0.0156393, -0.000197461, 1.34491e-06, -5.24544e-09, 1.1703e-11, -1.38811e-14, 6.78368e-18]

At long distances we extrapolate the behaviour of the parameters:

Direct_landauNormpars_far: [2.23151, -0.00627503]
Direct_landauMPVpars_far: [-3.04952, 0.128638]
Direct_expoCtepars_far: [3.69578, -0.00989582]

Direct_functions: ["pol7", "pol4", "pol3", "pol6", "pol7", "expo", "pol1", "expo"]

range of distances where the parametrization is valid [~10 - 500cm], then:

D_break: 500.

farther are extrapolations

D_max: 750.
```

# OnePhoton vs LitePhotons

- Two objects that can be saved in the simulation.
- Differences are:

```
// This structure contains all the information per photon
// which entered the sensitive OpDet volume.
```

```
class OnePhoton
{
public:
 OnePhoton();

 bool SetInSD;
 TVector3 InitialPosition;
 TVector3 FinalLocalPosition; // in cm
 float Time;
 float Energy;
 int MotherTrackID;
};
```

One of these is much lighter  
In terms of disk space and  
Memory.

Guess which one?

```
class SimPhotonsLite
{
public:
 SimPhotonsLite();
 SimPhotonsLite(int chan)
 : OpChannel(chan)
 {}

 int OpChannel;
 std::map<int, int> DetectedPhotons;

 SimPhotonsLite& operator+=(const SimPhotonsLite &rhs);
 const SimPhotonsLite operator+(const SimPhotonsLite &rhs) const;

 bool operator==(const SimPhotonsLite &other) const;
};
```



# Material Properties

- Some definitions in:  
lardataalg → larproperties.fcl

```
Surface reflectivity data - vector of energy spectrum per
surface type
ReflectiveSurfaceEnergies: [7, 9, 10]
ReflectiveSurfaceNames: ["STEEL_STAINLESS_Fe7Cr2Ni"]
ReflectiveSurfaceReflectances: [[0.25, 0.25, 0.25]]
ReflectiveSurfaceDiffuseFractions: [[0.5, 0.5, 0.5]]

Information related with the simulation of the Wavelength Shifter (TPB)
LoadExtraMatProperties: false

TPB - WLS
TpbTimeConstant: 2.5 #wls time constant in s J. Lumin 81(1999) 285

WLS - TPB properties original tpb [0.0, 0.0, 0.0, 0.0588,0.235, 0.853, 1.0,1.0,0.9259,0.704,0.0296,0.011, 0.0,0.0, 0.]
TpbEmmisionEnergies: [0.05,1.0,1.5, 2.25, 2.481, 2.819, 2.952,2.988,3.024, 3.1, 3.14,3.1807, 3.54, 5.5, 50.39]
TpbEmmisionSpectrum: [0.0, 0.0, 0.0, 0.0588,0.235, 0.853, 1.0,1.0,0.9259,0.704,0.0296,0.011, 0.0,0.0, 0.]
TpbAbsorptionEnergies: [0.05,1.77,2.0675, 7.42, 7.75, 8.16, 8.73, 9.78,10.69, 50.39]
TpbAbsorptionSpectrum: [100000.0,100000.0, 100000.0,0.001,0.00000000001,0.00000000001, 0.00000000001, 0.00000000001, 0.00000000001, 0.00000000001]
}
```

Not much there really.

The TPB is usually not implemented unless using TPB-covered foils.

- Some hardcoded in  
larsim/LArG4/MaterialPropertyLoader.cxx

# Adding Reflected Light

- SBND is installing WLS coated reflector foils on the CPA to enhance light collection. The idea has been proposed for DUNE. This leads to visible light being seen from the cathode.

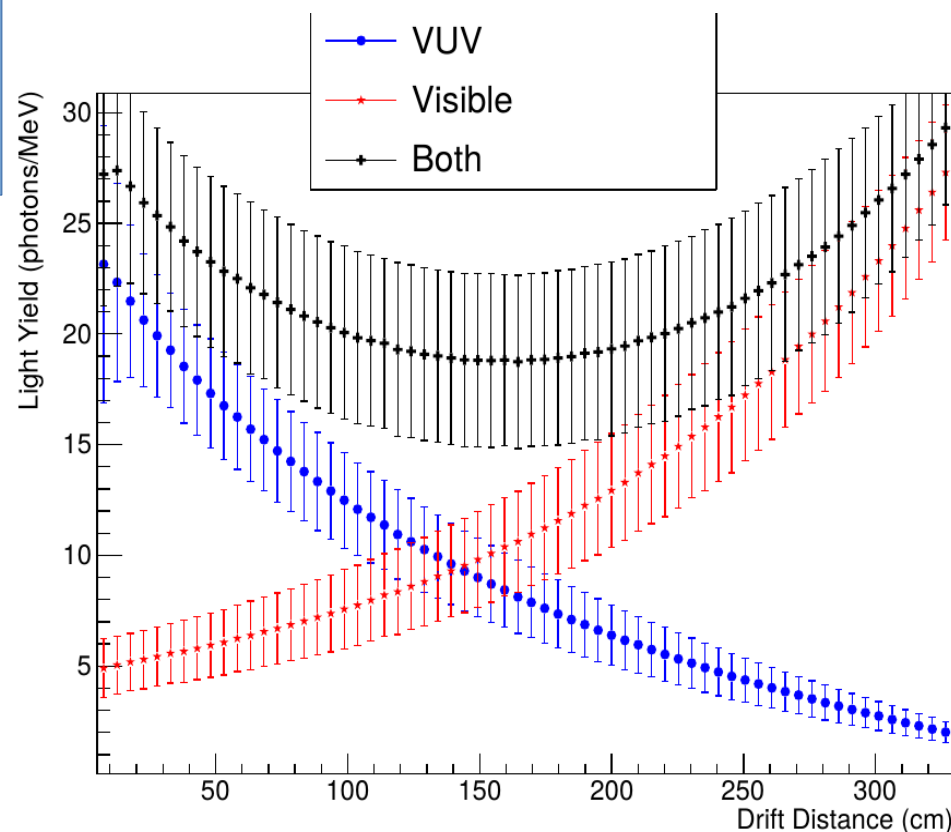
```
<material name="TPB" formula="TPB">
 <D value="1.40" unit="g/cm3"/>
 <fraction n="1.0000" ref="argon"/>
</material>
```

In-gdml defintion  
Of TPB is very  
"Argon-like"

```
<!-- For Foils -->
<material name="vm2000" formula="vm2000">
 <D value="1.2" unit="g/cm3"/>
 <composite n="2" ref="carbon"/>
 <composite n="4" ref="hydrogen"/>
</material>
```

If PhotonVisiblityService.StoreReflected: true  
Then library gets a second component, direct  
VUV and reflected visible light are  
in principle separate.

Although for SimPhotonsLite, can't  
tell the difference.





# FastSim vs OpSim Knobs

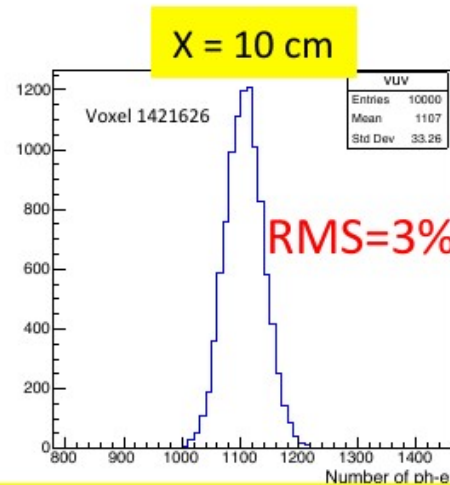
	Full Optical Sim	Fast Optical
Timing Constants	Tunable	Tunable
Energy Spectrum	Tunable	Tunable (although affects transport)
Ionization/Scintillation Yield	Tunable (handwavy implemented)	Tunable (handwavy implemented)
Rayleigh Scattering	Tunable	“Burned in”
Timing Parametrization	Not needed	tunable
Material Properties	Tunable	“Burned In”
OnePhoton vs LitePhotons	chooseable	chooseable

# Random bits of knowledge

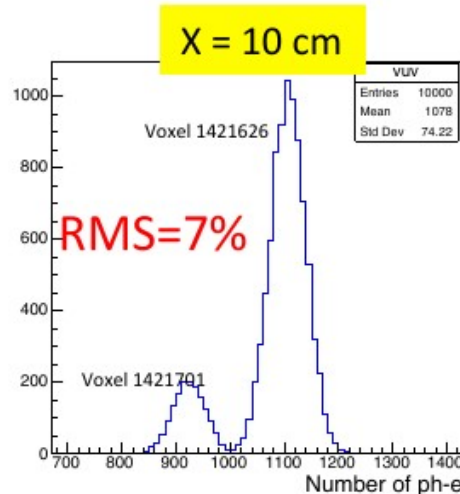
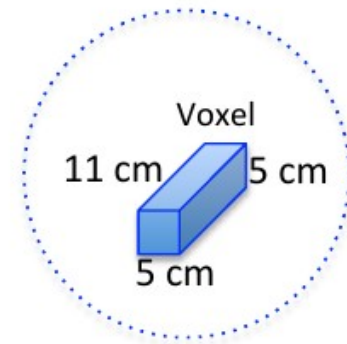
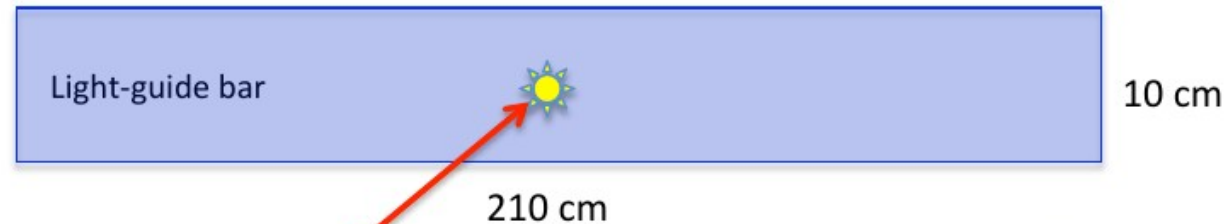
# Generating the Optical Library

- Here is an advanced tutorial, that covers most of the details:  
[https://cdcv.sfnal.gov/redmine/projects/dunetpc/wiki/How\\_to\\_make\\_a\\_photon\\_library](https://cdcv.sfnal.gov/redmine/projects/dunetpc/wiki/How_to_make_a_photon_library)
- The Library is generated by using a special event generator called LightSource (lives in larsim).
- The volume needs to be divided into voxels – the size of the voxels affects memory footprint of the library very quickly. Remember, we need to store each x,y,z,PD combination.
  - Currently we can do 5x5x11cm in half of the 1x2x6 detector.
- The larger the number of photons, the better. Current DUNE way of working is 50k/voxel. IMHO this is on low border of what you can trust. More photons = more memory, though. :(
- Basically, you cannot get out of doing this on the grid.

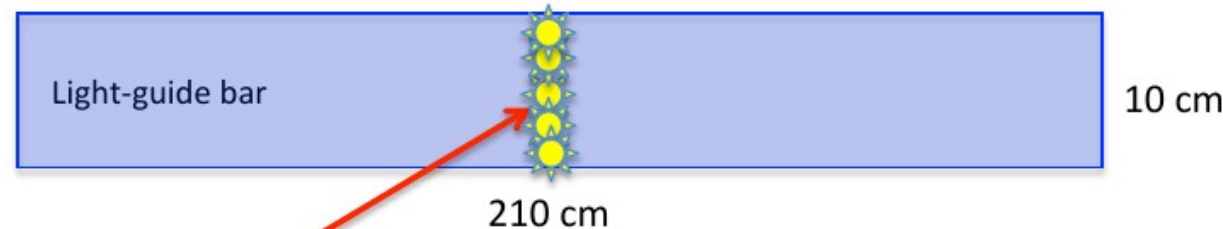
# Why the uncertainty of the direct light is so big close to the photo-cathode plane?



Scintillation points fixed in the center of a bar



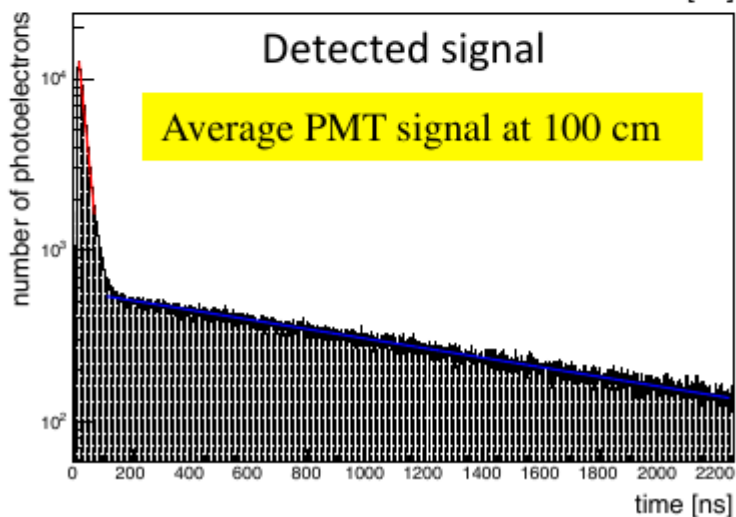
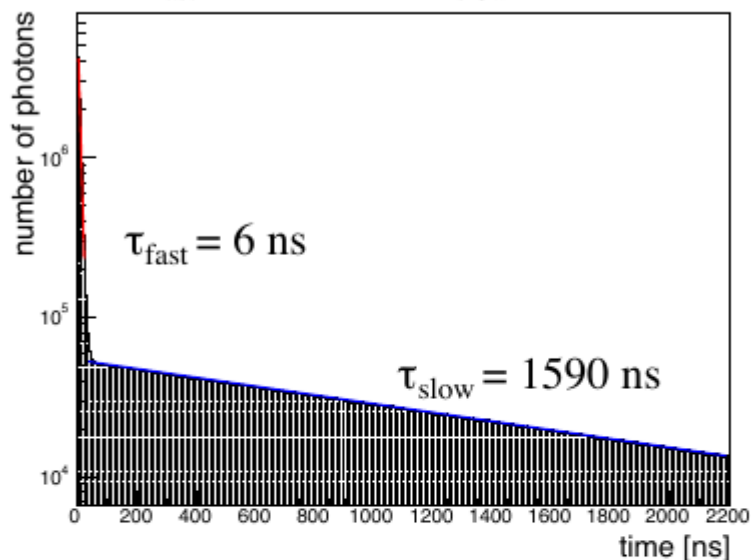
Scintillation points varying within the bar Y-width



# Why care about timing?

## Scintillation (emission):

$$0.3 \times \tau_{\text{fast}} (6 \text{ ns}) + 0.7 \times \tau_{\text{slow}} (1590 \text{ ns})$$

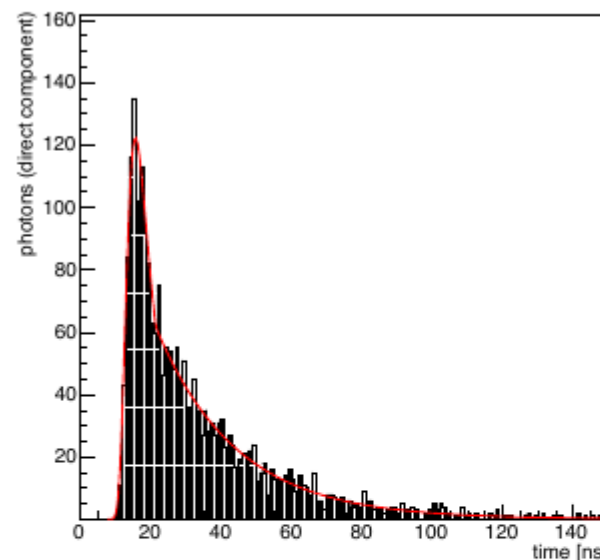


## Propagation:

Direct transportation + Rayleigh Scattering

+

+



=

In "large" detectors transport effects will affect the effective time structure of the detected scintillation light

D. Garcia-Gomez

(SBND geometry used on these plots)

33

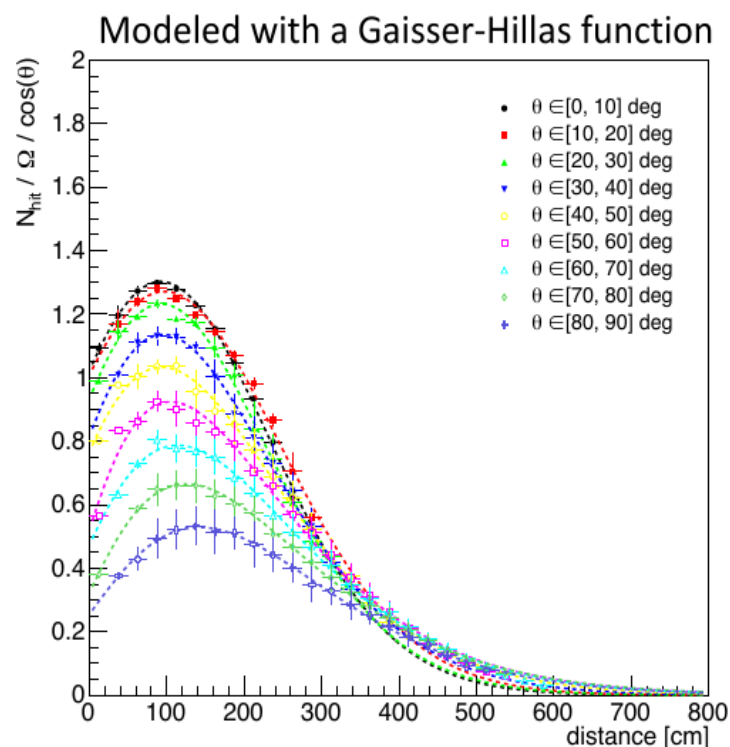
# Some general comments

- Given the large scale of the detectors and the high energies deposited, LArSoft will embrace any simplification that it can get away with. And probably some it can't.
- The library itself is an example, but by no means the only place where we're cutting corners.
- Most materials are assumed to be non-reflective to VUV light.
- You've built up a fancy simulation of your light detector that is super precise and sophisticated?
  - Unfortunately, it will probably end up being an efficiency-like number in the large detector simulation. Sorry.



# Getting around the library

- It would be great if we could get a quicker/more precise method of estimating the amount of light from scintillation.
- Geometric approximations are a natural candidate, but they don't work well because of Rayleigh scattering – modelling this is non-trivial.
- We think it can be done though, some first results look promising. Should make it into LArSoft in the next couple of months.



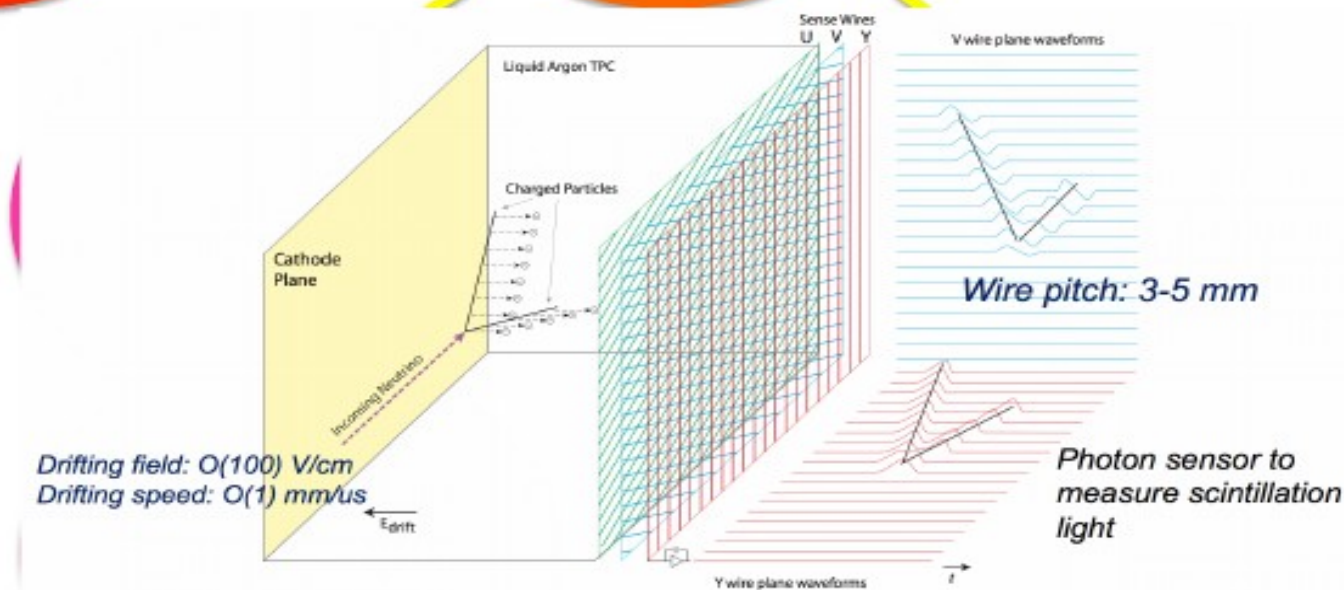
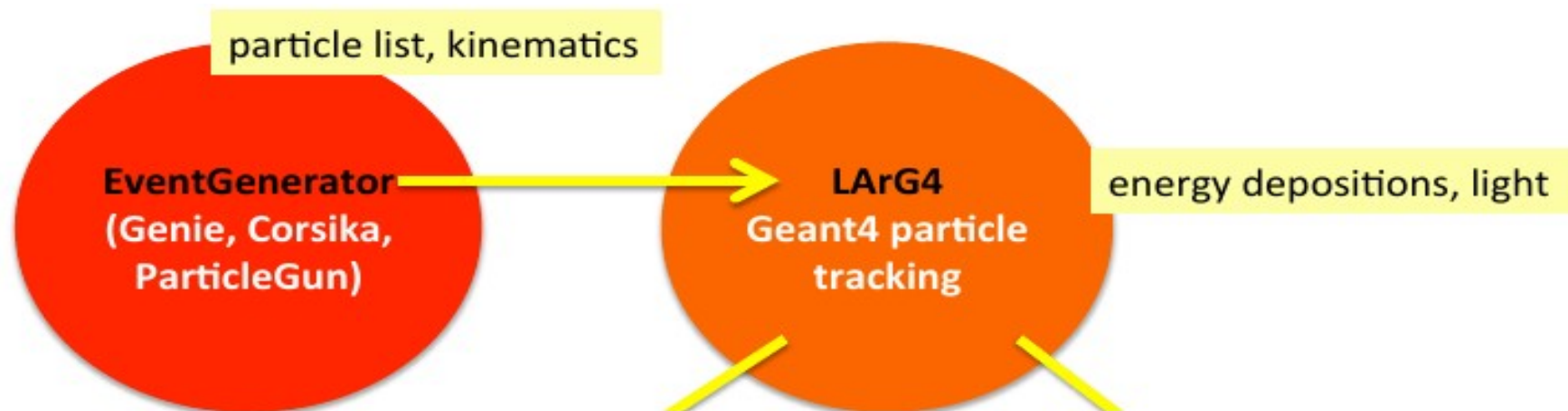
# Closing remarks

- Simulation of scintillation light in LArSoft is not easy.
- Some of these steps are still in development, so you need to keep track of what's going on. ;-)
- Tomorrow we will learn how to look at some of the effects of the simulation.



# Backup

# Simulation flowchart



enter  
s in  
s  
(ors)  
ry

ctor  
tons

into waveforms

D. Garcia-Gamez

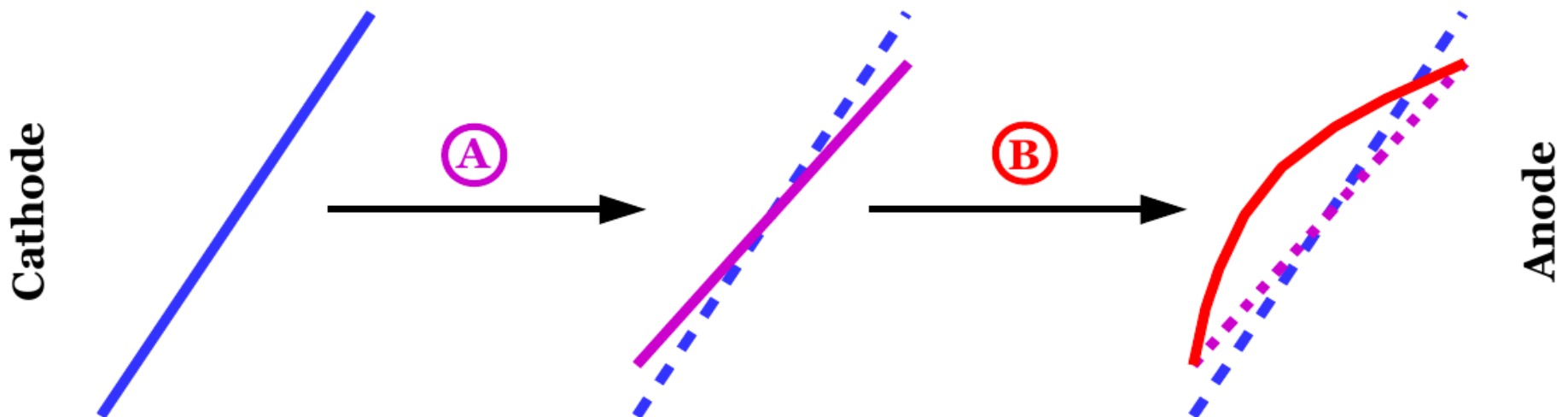
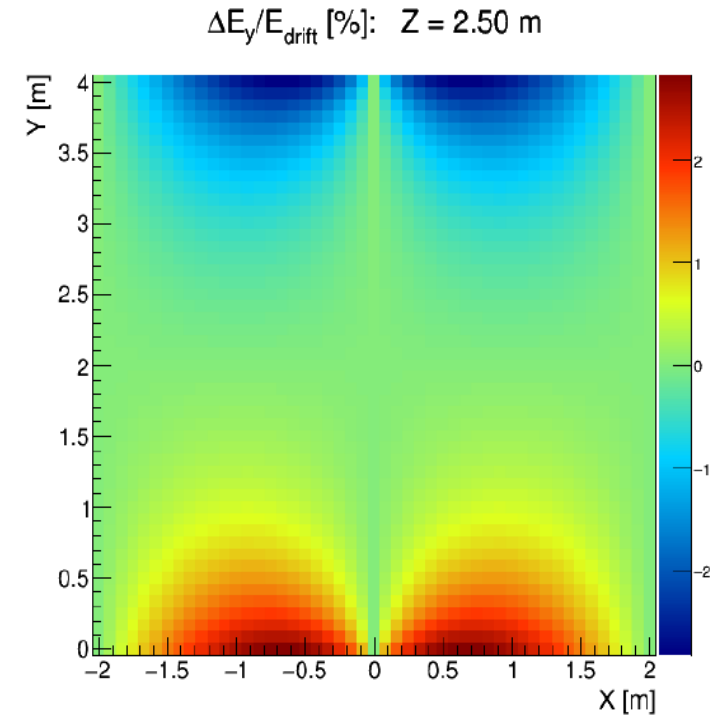
# GenieGen (neutrino interaction)

- ✓ GENIE is the most popular neutrino generator
- ✓ You provide the flux files and specify where you want the neutrinos to interact: *volWorld*, *volCryostat*, *volTPC*
- ✓ 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 event sample

D. Garcia-Gamez

# Modifying the electric field

- Until now, we assumed that the electric field is uniform.
- This could not be the case. A well known effect that can modify the E-field is space charge.
- A module exists to simulate this and correct for it.





# Services

Classes with a single instance managed by the framework

Services are provided in the areas of geometry, physical properties, physics simulation and miscellaneous.

***Geometry services*** provide a description of the physical and readout aspects of the detector such as:

- TPC structure
- optical detector structure
- readout channel mappings (e.g. readout channels to physical wires and optical sensors)
- auxiliary detectors

***Physical properties services*** such as:

- Detector Properties (e.g. drift velocity,  $dQ/dx \rightarrow dE/dx$ , electron lifetime)
- LArProperties about the liquid argon environment (i.g. radiation length, argon temperature)

***Physics simulation services*** such as:

- Voxel Calculator helps with computations in the virtual grid.
- G4 Parameters stores the parameters GEANT is configured with.
- Photon Visibility describes the simulation of the transportation of photons to the optical detectors.

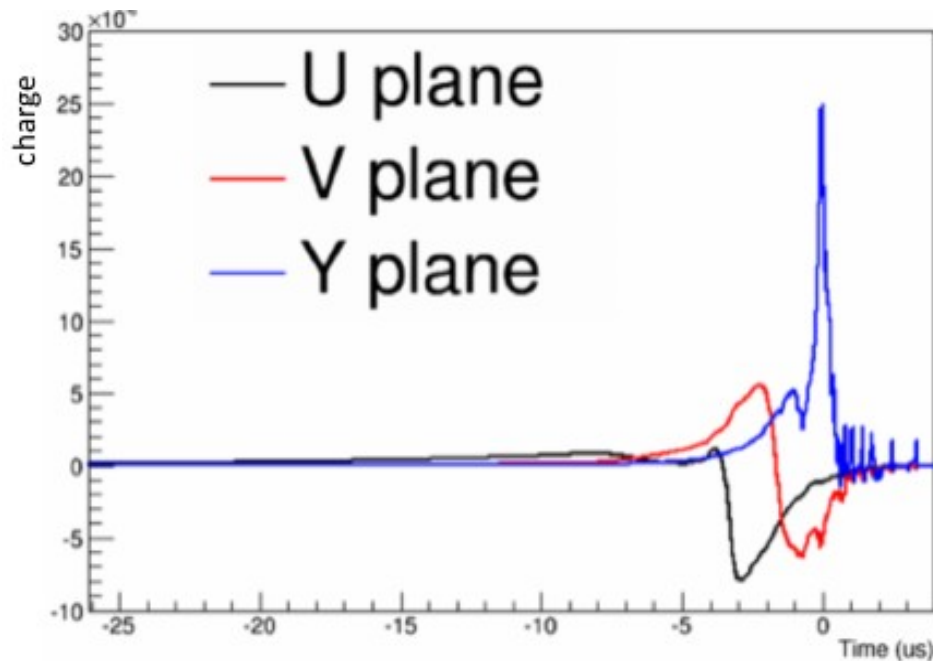
D. Garcia-Gamez

# larsim/simulation/ simulationsservices.fcl

```
standard_largeantparameters:
{
OpticalSimVerbosity: 0 #verbosity of optical simulation, soon to be deprecated
ParticleKineticEnergyCut: 0.01e-3 #in GeV
StoreTrajectories: true
VisualizationEnergyCut: 10.e-3 #deprecated, in GeV
VisualizeNeutrals: false #deprecated
UseCustomPhysics: false #Whether to use a custom list of physics processes or the default
KeepEMShowerDaughters: false #save secondary, tertiary, etc particles in EM showers
LongitudinalDiffusion: 6.2e-9 #in cm^2/ns
TransverseDiffusion: 16.3e-9 #in cm^2/ns
ElectronClusterSize: 600.0 #number of ionization electrons to drift in a unit
MinNumberOfElCluster: 0 #minimum number of electron clusters
EnabledPhysics: ["Em", "SynchrotronAndGN", "Ion", "Hadron",
 "Decay", "HadronElastic", "Stopping", "NeutronTrackingCut"]
CosmogenicKOBias: 0 # 0 is off. N is the number of secondaries to produce.
CosmogenicXSMNBiasOn: 0 # 0 is off. 1 works. 2 still in development.
CosmogenicXSMNBiasFactor: 1 # Not more than 5-ish cuz of numerical instabilities.
DisableWireplanes: false #if set true, charge drift simulation does not run - used for optical si
SkipWireSignalInTPCs: [] # put here TPC id's which should not receive ionization electrons - use
UseModBoxRecomb: true # use Modified Box recombination instead of Birks
}
```

# Field Modeling (MicroBooNE example)

- Simulating the field response function is the first step in the chain of signal processing
- The response of the channels to the drifting electrons is parameterized as a function of drift time, with separate response functions for collection and induction wires

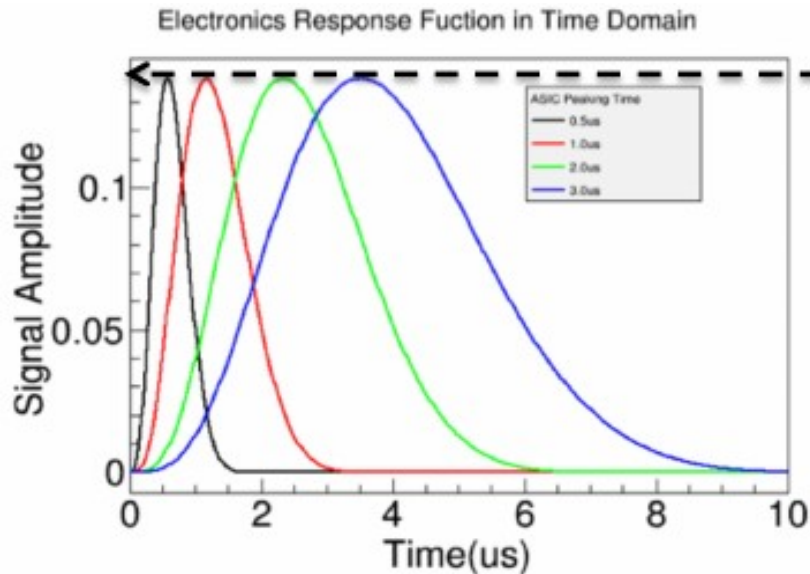


- 2-D GARFIELD(\*) simulated response to a single electron generated in the MicroBooNE detector
- Can be inserted via Tformula, or use basic step functions

D. Garcia-Gamez

# Modelling Electronics Response

- MicroBooNE front-end cold electronics designed to be programmable with 4 different gain settings (4.7, 7.8, 14, and 25 mV/fC) and 4 shaping time settings (0.5, 1, 2, and 3  $\mu$ s)  $\rightarrow$  the electronic response function varies according to these settings



- For a fixed gain setting, the peak is always at the same height independent of the shaping time

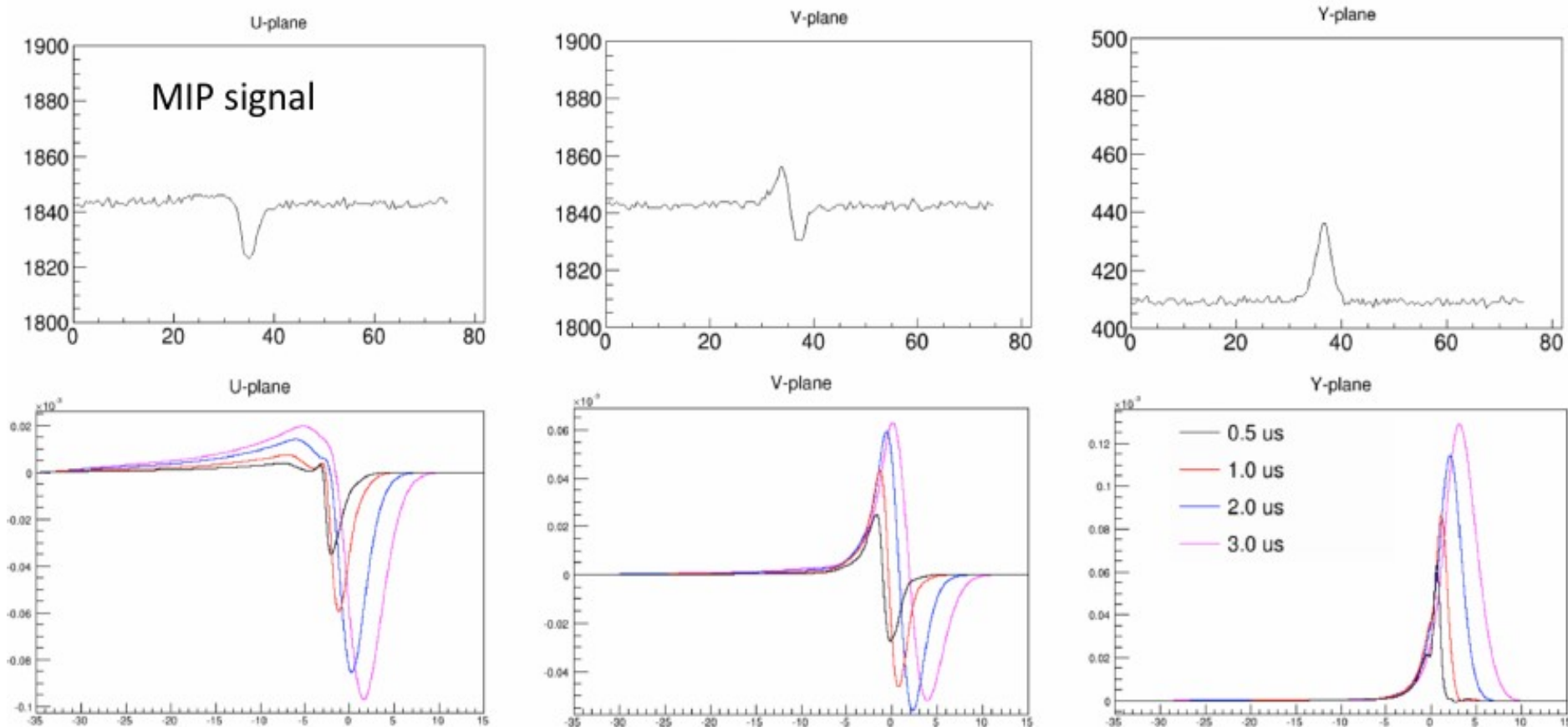
## Noise

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

D. Garcia-Gamez

# Examples of Raw and Convoluted Signals

Digitized signal after the ADC = ionization signal convoluted with the detector and the electronics response functions and then digitized at a fixed frequency



D. Garcia-Gomez



# CalWire

- ✓ So now, that we have the raw data event, we want to remove the field shape and electronics response that we just put in
- ✓ This is done in `CalWire<Experiment>` (again outsourced to `SingalShaping<Experiment>_service`)
- ✓ Filtering can also be applied here to get rid of noise
- ✓ What we want to get is a representation of the initial charge in ADC counts as accurate as possible (without the detector effects) → Which we could then convert to real charge via `DetectorProperties::ElectronsToADC`

D. Garcia-Gamez



# Deconvolution

- ✓ Deconvolution is needed to convert raw readout signals to arriving charge vs. time

Based on deconvolution kernels (precalculated and stored in a file or calculated on the fly, at job initialization) → `SignalShaping<Experiment>`

- ✓ **Ideally**, according to image processing theory (Wiener deconvolution), the optimal filter function is (minimizes the mean square error)

$$F(f) = |R(f)|^2 / (|R(f)|^2 + |N(f)|^2)$$

- $R(f)$  = Response function (convolution of field and electronics response)
- $N(f)$  = Noise spectrum

- ✓ In any case, the deconvolution kernel is calculated as the ratio of the filter function and the convolution kernel

→ **deconvolution kernel:  $K(f) = F(f)/R(f)$**

D. Garcia-Gamez

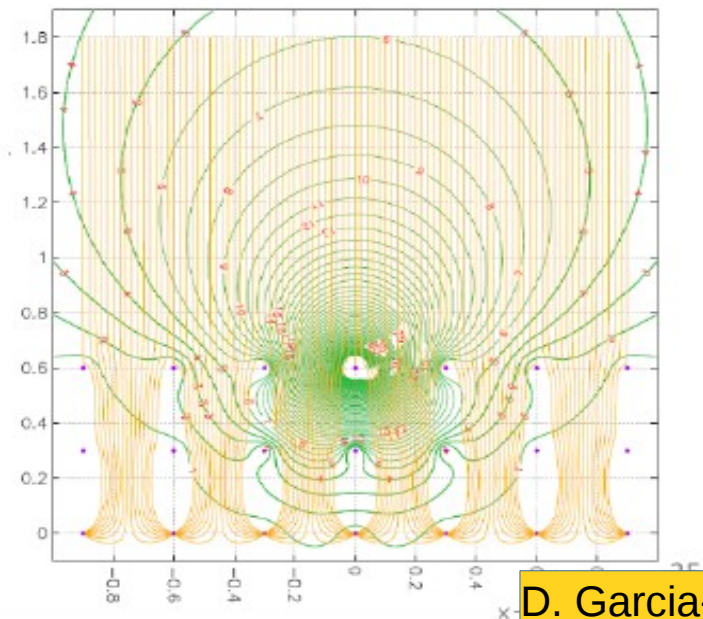
# Additional challenges in signal processing

- With the noise filtering and deconvolution, the full signal processing chain is complete
- But, there are still some additional challenges involved in the process:

- Ionized electrons traveling through the TPC wires induce signal not only on the closest wire but also on the adjacent wires → dynamic induced charge
- The field model described before does not take into account the charge contributions from the adjacent wires

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

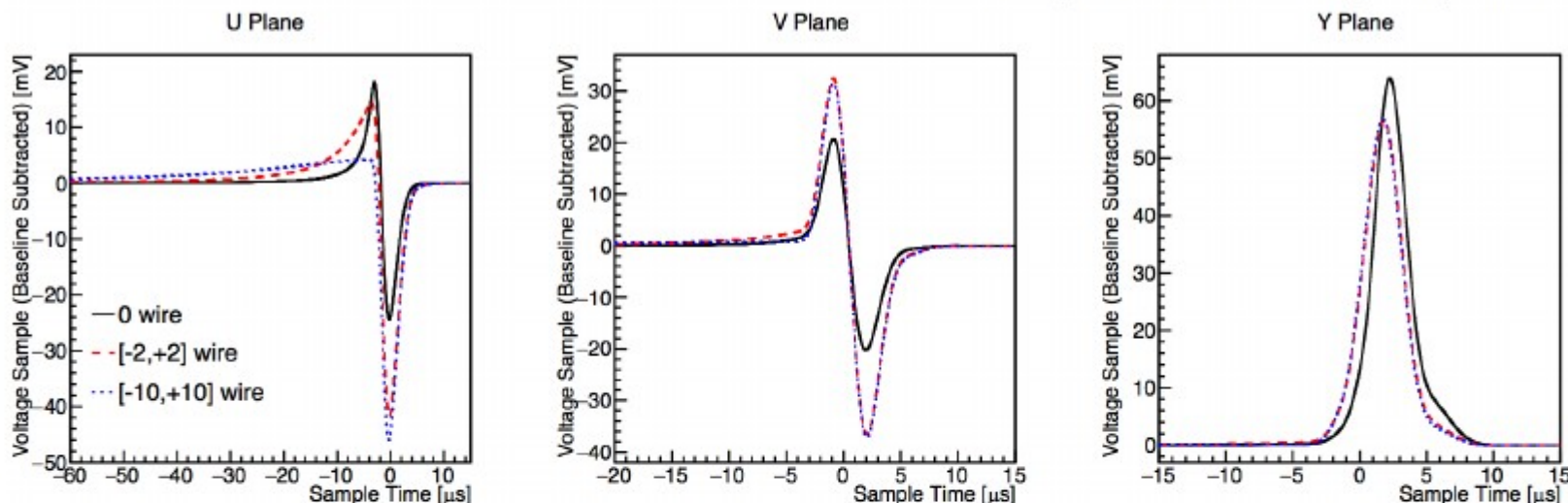
Weighting Field of a U Wire



D. Garcia-Gamez

# Field response - 2D Garfield calculation

Field response + Pre-amp response



## 2D Deconvolution

Significant contribution from adjacent wires

- Deconvolve with respect to time and wire dimensions
- $M_i(t') = \int_{-\infty}^{\infty} (... + R_1(t_0 - t) \cdot S_{i-1}(t) + R_0(t_0 - t) \cdot S_i(t) + R_1(t_0 - t) \cdot S_{i+1}(t) + \dots) dt$ 
  - $M_i(t')$  - measured signal from wire  $i$ ,
  - $S_i(t)$  - signal within the boundaries of wire  $i$ , where  $\pm$  a half pitch defines the wire boundaries
  - $R_n(t_0 - t)$  - average response of wire  $i$ , where  $n = ||i||$

This implementation of a double de-convolution method is a recent development in the LArSoft signal processing procedure

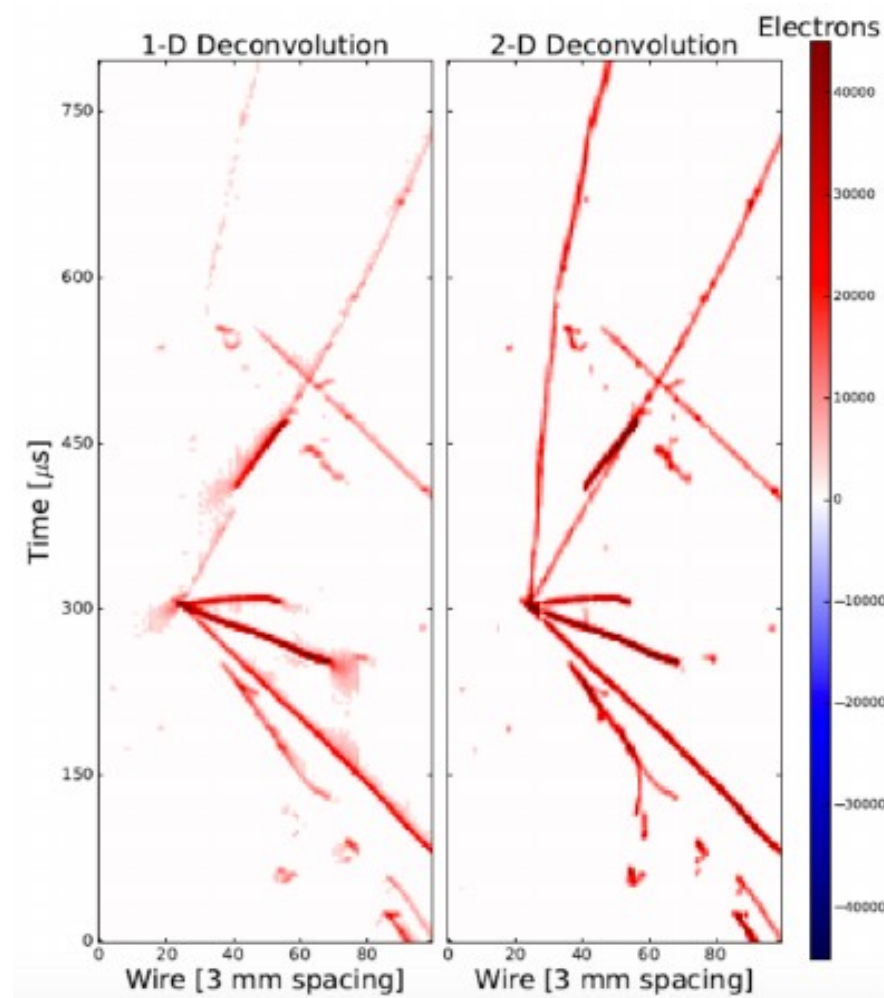
Courtesy of Brooke Russell

D. Garcia-Gamez

# Effects of 2D deconvolution

## Qualitative Performance

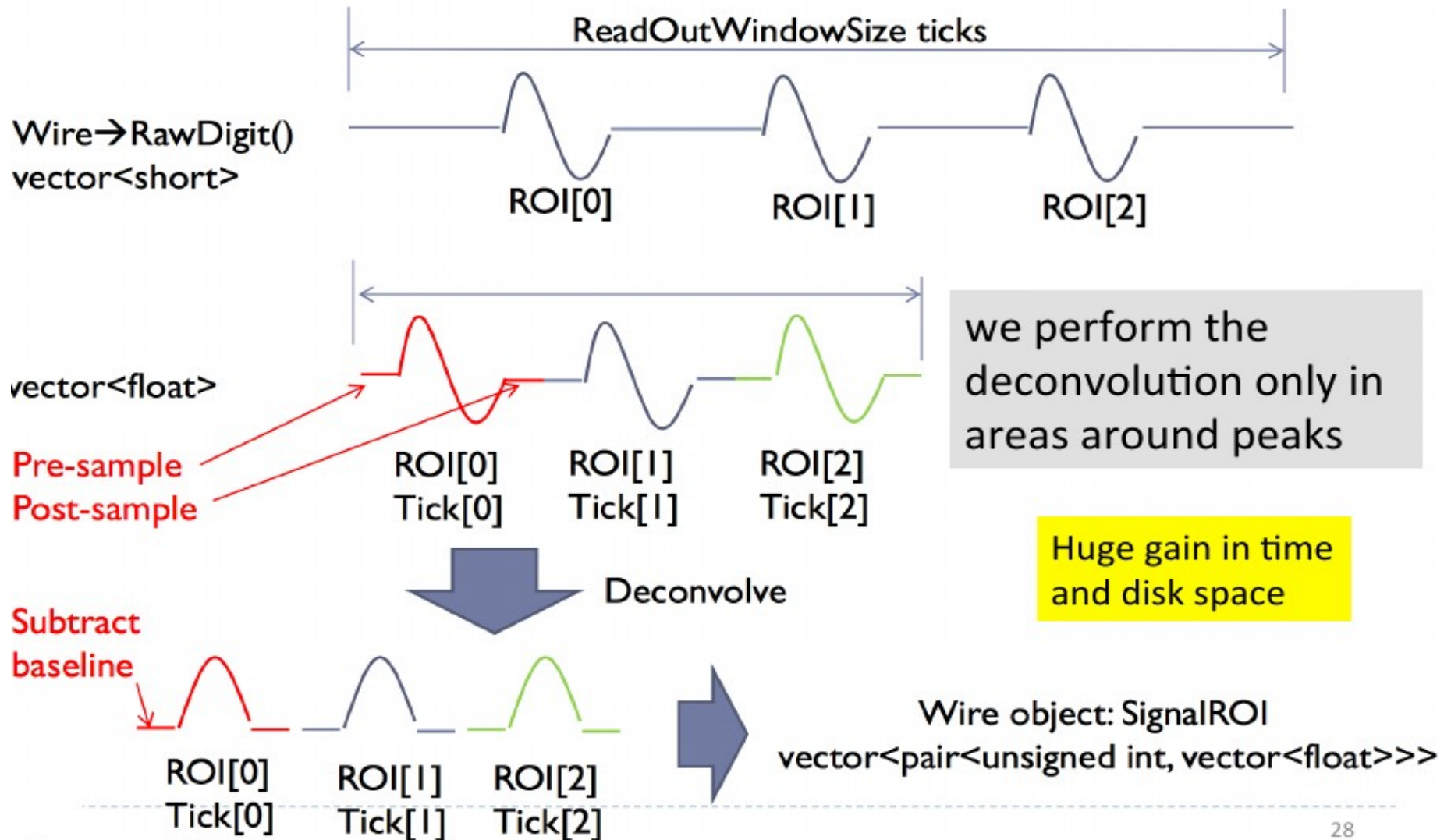
- 1D deconvolution
  - Signal smearing
  - Less efficiency for reconstructing charge for tracks at large angle with respect to wire plane
- 2D
  - Better recovers the true signal
  - More efficient recovery of charge for difficult topologies



Courtesy of Brooke Russell



# Using Regions of Interest



Courtesy of B. Baller

28