

# Optical Monte Carlo in LarSoft : Update

Ben Jones, MIT

# Long term goals for optical MC

- Complete full geant4 simulation of optical photons produced in LAr interactions
- Use this to create a parameterized fast sim
- Understand phototube trigger efficiency – and dependence on position, energy, etc.
- Perform optimization of optical components – phototube reflectors, waveguides, etc
- Investigate utilization of optical data in reconstruction

# The Agenda

- Allow new physics outside the default physics list to be inserted into LarG4
- Take input of the new parameters required for optical simulation – wavelength dependent absorption spectra, etc
- Insert optical photons and optical processes (scintillation, cerenkov, reflections, absorptions, rayleigh scatters) into the physics list

Done!

- 
- Insert all parameters we already know, and find out which ones we need to know
  - Sanity check analysis of some sample events

Doing!

- 
- Insert sensitive detectors and optical components (phototubes, wavelength shifters, etc)
  - Investigate flux to phototube per event as function of event location, energy, etc
  - Parameterize phototube response per event
  - Create PropagatePhotons to provide representative phototube response for further studies

To do!

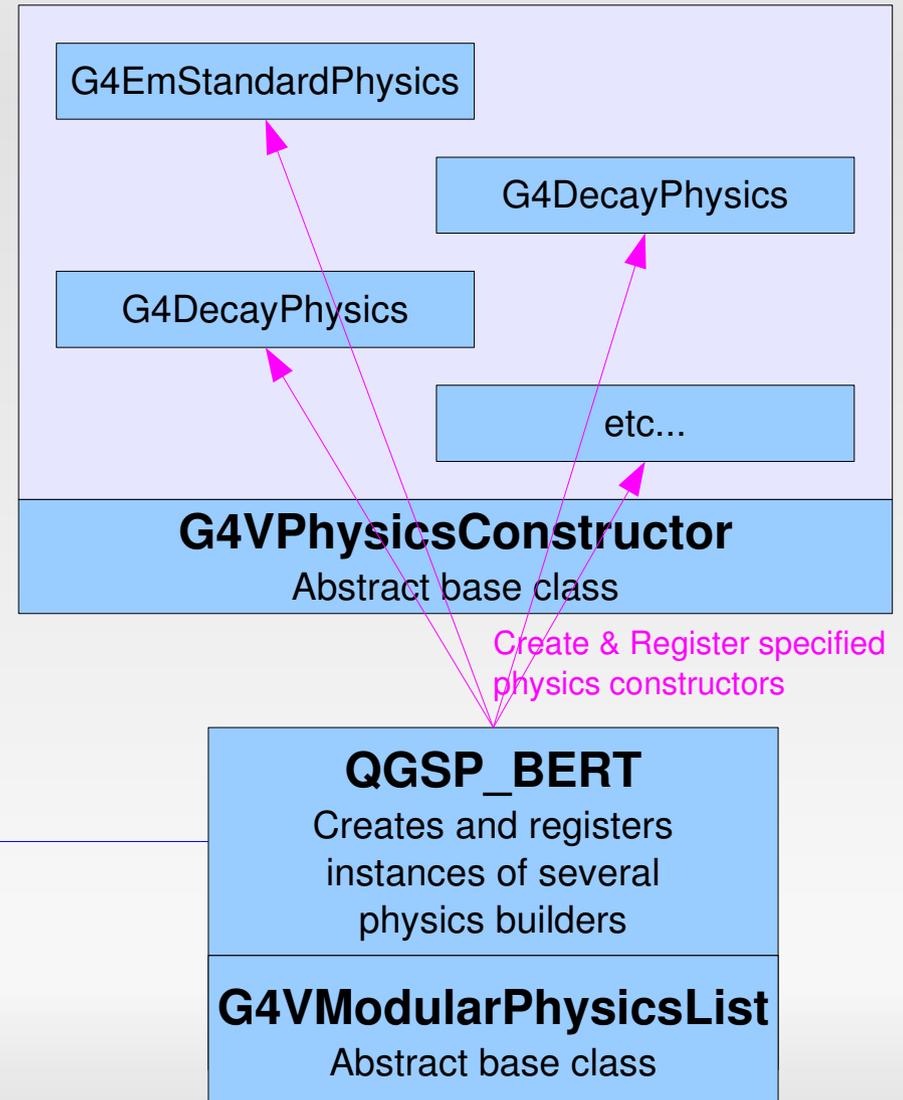
# Physics Lists in LarSoft - Before

LArG4

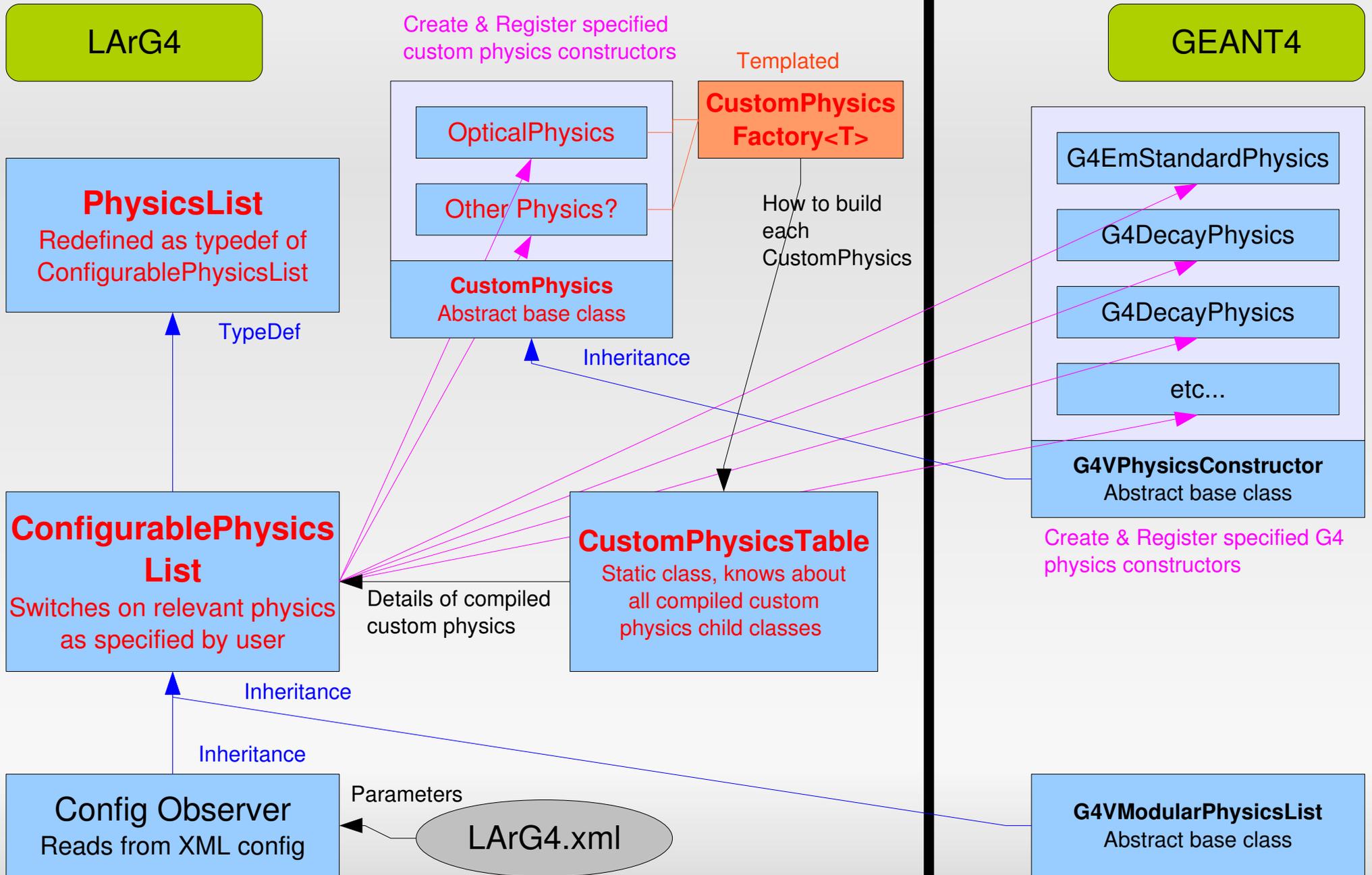
**PhysicsList**  
Defined in LarSoft as a  
typedef of the QGSP\_BERT  
G4 physics list

Typedef

GEANT4



# Physics Lists in LarSoft - After



# New Physics Constructors

- To create a new physics constructor, eg to register optical physics processes, create a new child class of CustomPhysics. In the header, create a CustomPhysicsTable with a templated CustomPhysicsFactory<OpticalPhysics> at global scope.
- The CustomPhysicsFactory tells the CustomPhysicsTable how to build its templated type by feeding it the name and constructor. A CustomPhysicsFactory is static so only created new upon the first call.
- CustomPhysicsList provides a list of compiled custom physics modules and pointers to their constructors to ConfigurablePhysicsList
- Hence any compiled CustomPhysics module can be switched on by specifying its name in LArG4.xml. This provides complete flexibility in registering physics processes, and allows the user to create new physics modules and run them in the simulation without directly changing the physics list class.

# Switching physics on and off

```
<config name="LArG4" version="optical" base="default">  
  
  <param name="UseCustomPhysics">  
    <bool> true </bool>  
  </param>  
  
  <param name="EnabledPhysics">  
    A list of physics builders to include. All physics builders  
    which register at compile time are available only read if  
    UseCustomPhysics is set to true.  
    <string> 'Em' 'Optical' 'SynchotronAndGN' 'Ion' 'Hadron'  
      'Decay' 'HadronElastic' 'Stopping' 'NeutronTrackingCut'  
    </string>  
  </param>  
  
</config>
```

# Optical Physics

- The optical physics constructor registers one new particle, Photon - `G4OpticalPhoton`
- and several new physics processes

Cerenkov Radiation – `G4Cerenkov (/watchable)`

Absorption – `G4OpAbsorption (/watchable)`

Rayleigh Scattering – `G4OpRayleigh (/watchable)`

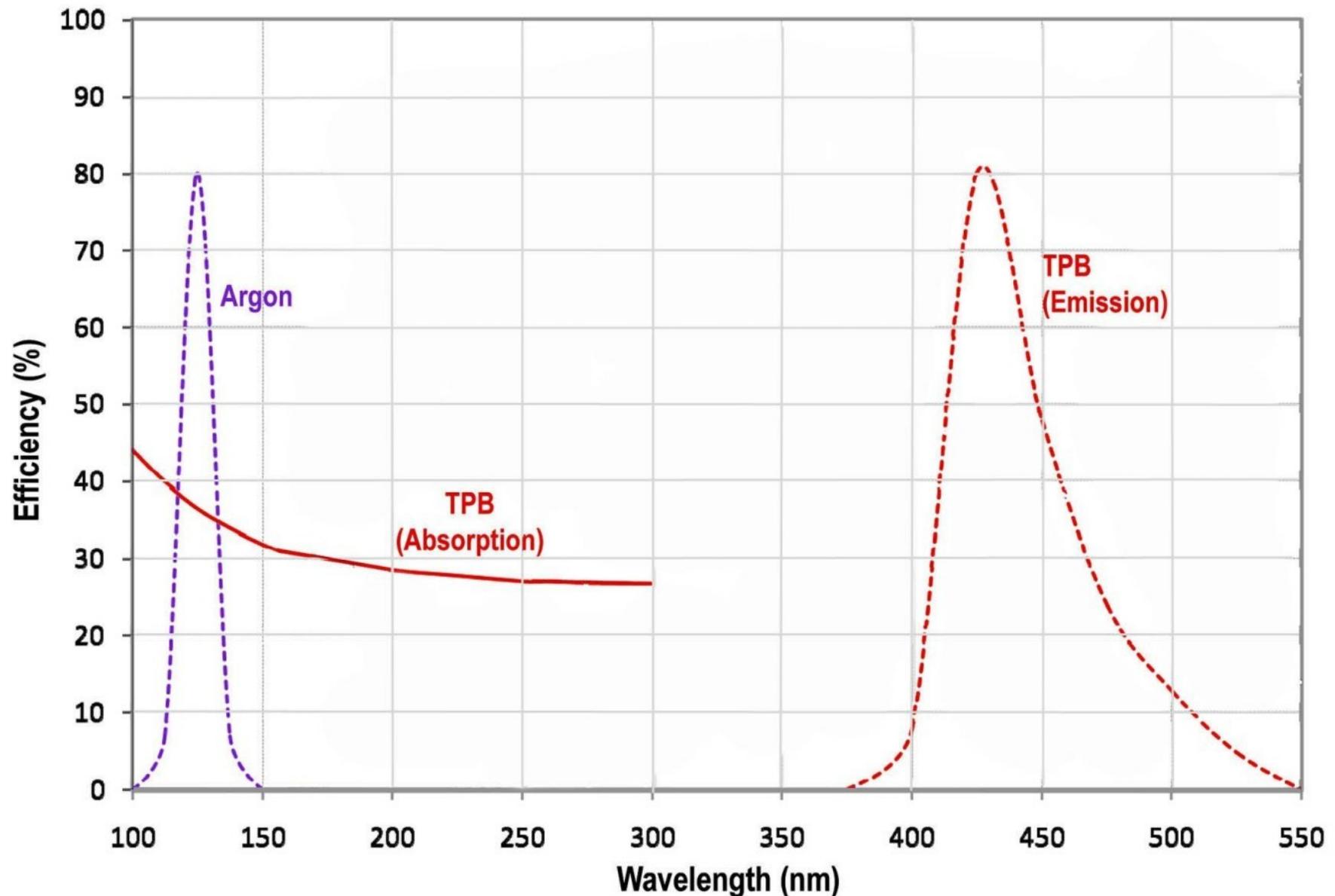
Reflections - `G4OpBoundarySimple`

Scintillation – `G4Scintillation (/watchable)`

Customised for  
LArSoft

Needs  
customising

# Wavelength Dependences



# MaterialPropertyLoader

- Gdml does not allow parameters to be input to geant with a wavelength dependence
- Hence need a way to get absorption spectra, wavelength dependent reflectivities, etc, into the simulation
- The MaterialPropertyLoader class is a placeholder class which feeds these parameters into Geant4. Each implementation reads from a different source.
- **Currently in use** : DummyMaterialPropertyLoader, simply hard code parameters in
- **Almost Done** : xmMaterialPropertyLoader, read values from xml file
- **Maybe one day if required** : RootMaterialPropertyLoader, read from root file

# TextMaterialPropertyLoader

```
<materialdoc>
  <material name = "LAr">

    <property type = "variable" name = "RAYLEIGH" unit = "cm">
      Rayleigh scattering length in liquid argon
      <momenta>    9.5        9.7        9.9    </momenta>
      <value>      90.0      91.0      92.0    </value>
    </property>

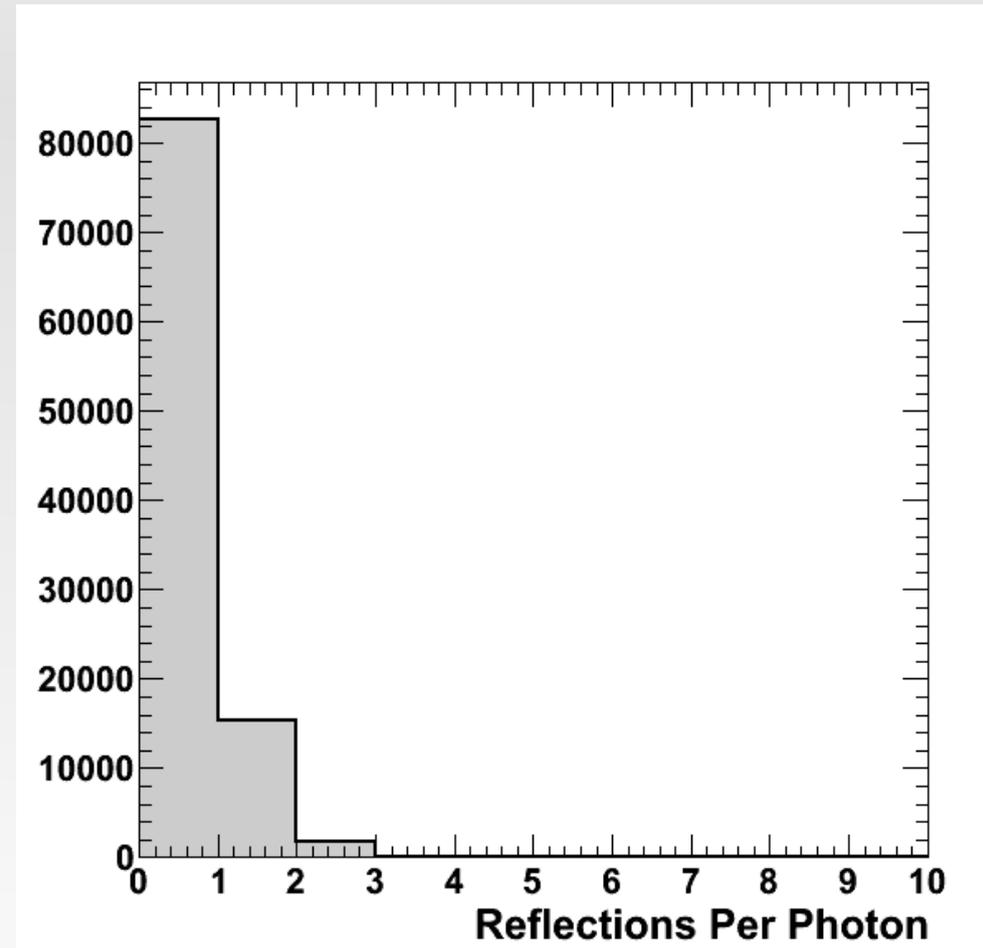
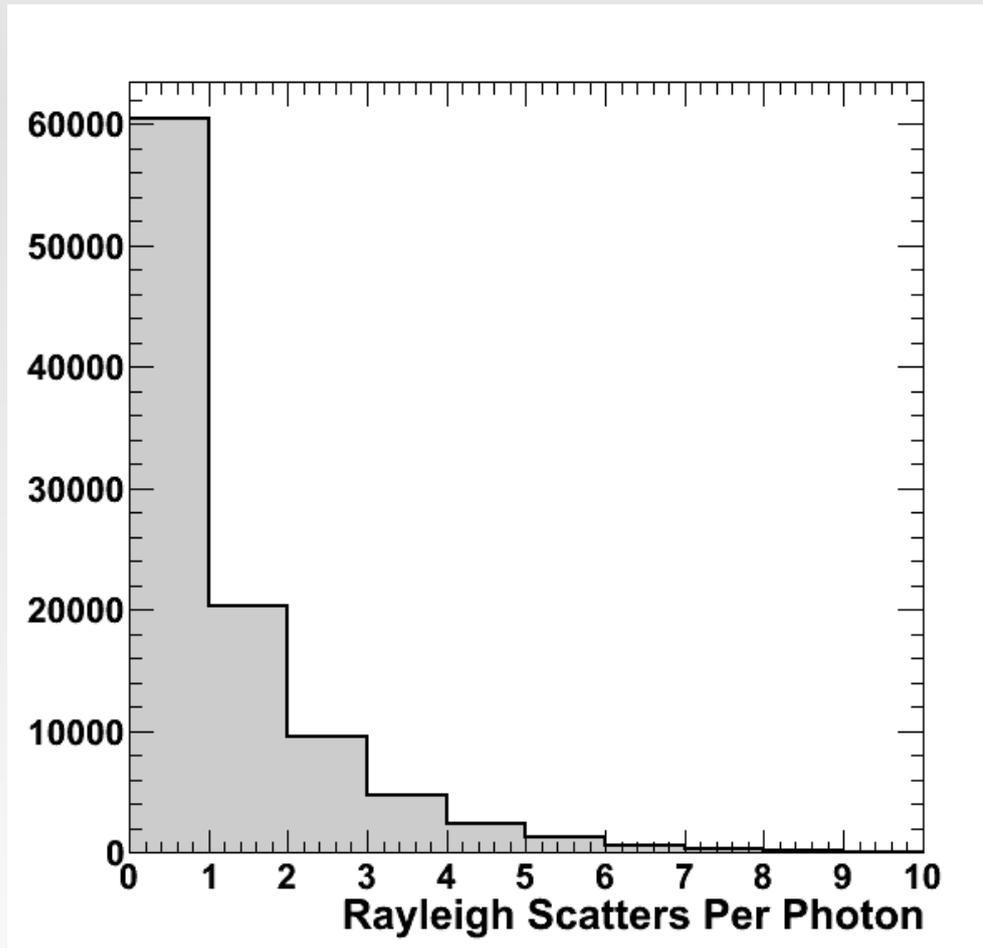
    <property type = "const" name = "FASTTIMECONSTANT" unit = "ns">
      Fast time constant for scintillation in liquid argon
      <value>    6    </value>
    </property>

  </material>
</materialdoc>
```

# G4OpBoundarySimple

- The G4OpBoundaryProcess model is too complicated given our knowledge of various reflectivities, and the number of photons we want to simulate
- Inserted a new process, G4OpBoundarySimple
- Throw a random number to decide if it reflects or is absorbed
- If it reflects, throw another random number to decide if it is a specular or diffuse reflection
- Probabilities for each type of boundary loaded by MaterialPropertyLoader
- So far, only steel / LAr boundaries are parameterized, with constant reflectance 25% and diffuse fraction 50%
- From one sample event, 95161 photons were generated of which 58996 were eventually absorbed at a steel surface and 20932 were absorbed into a "black area".
- Each photon underwent a mean of 0.76 Rayleigh scatters and 0.19 reflections

# Reflections / Scatters per event

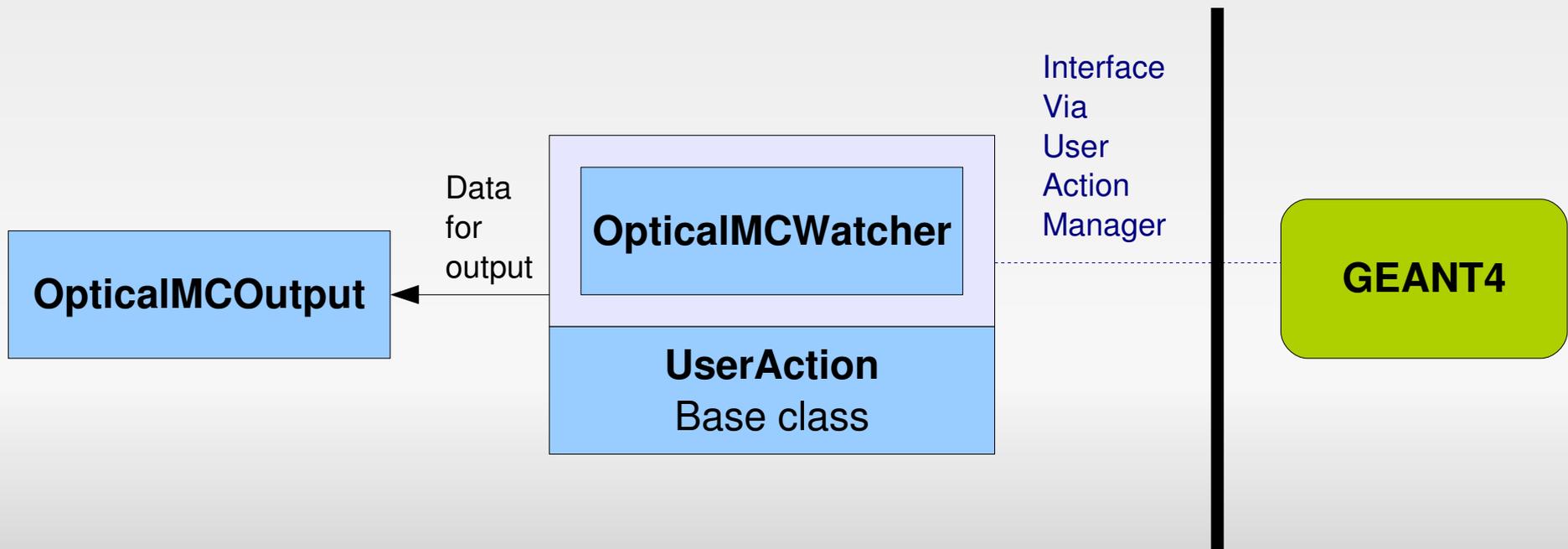


# Watchable physics processes

- Want to know where reflections / rayleigh scatters / absorptions occur, distribution of momentum changes, etc
- Can't get data directly from geant4 processes
- So instead, create new sister processes, OpRayleighWatchable, etc.
- Degree of output generated by each process controlled by parameter in LArG4.xml, OpticalSimOutputLevel. Can generate trees of information on a per photon basis, per event basis or per run basis.
- With the new configurable physics lists, watchable and original geant4 processes can be swapped in and out at runtime.
- Do we need these in the long term? Currently unclear.

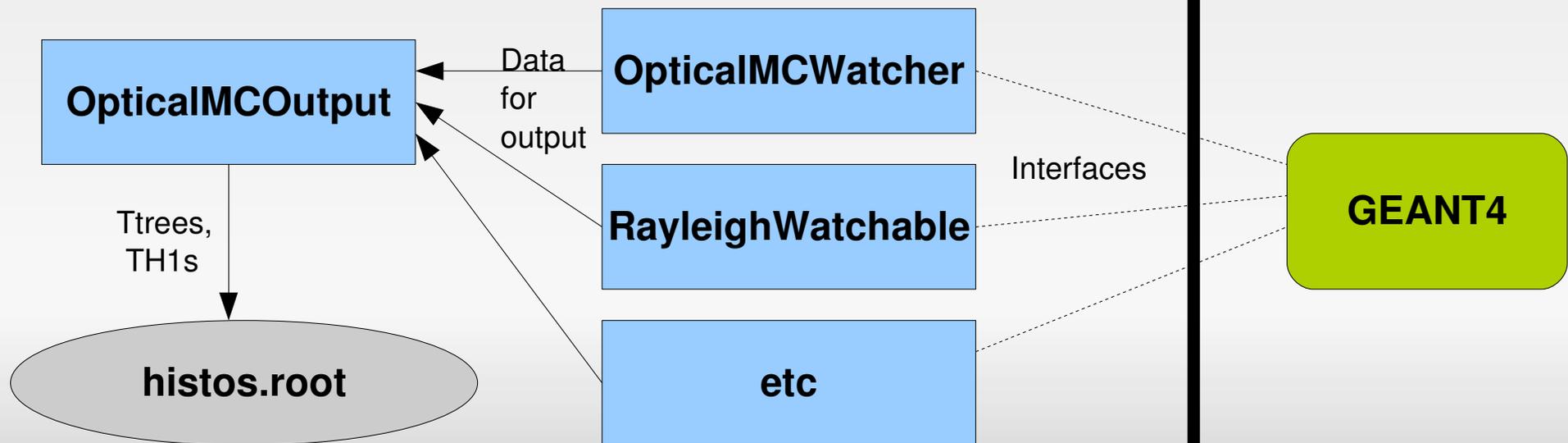
# OpticalMCWatcher

- Can also track the Geant4 simulation stepwise.
- OpticalMCWatcher inherits from UserAction and can execute routines per step, per track or per run
- OpticalMCWatcher output level also controlled by OpticalSimOutputLevel
- Currently used to extract birth and death place of each photon – but eventually this object will probably be the workhorse of optical MC analysis.



# OpticalMCOutput

- Various classes are producing output on a per photon or per event basis, for further analysis after simulation
- Want to combine all these into one set of TTree objects
- OpticalMCOutput is a class which accepts data from each class and packages it into individual Ttrees at the end of the run
- Only one instance of OpticalMCOutput is declared, at LArG4 scope, and it builds trees of information per photon and per event.
- Currently in development, not in CVS yet...



# Short term goals

- Finish XmlMaterialPropertyLoader, OpticalMCOOutput classes
- Find out which materials are absorbing most of the unaccounted for photons and find down optical data for them
- Write some analysis code to see how feasible the simulation is looking on a photon per photon basis (eg from plots produced so far, looks like too few rayleigh scatters)
- Add wavelength dependent scintillation process
- Ongoing search for better parameterised properties for liquid argon