LArSoft and Generator Interfaces

Erica Snider
Fermilab

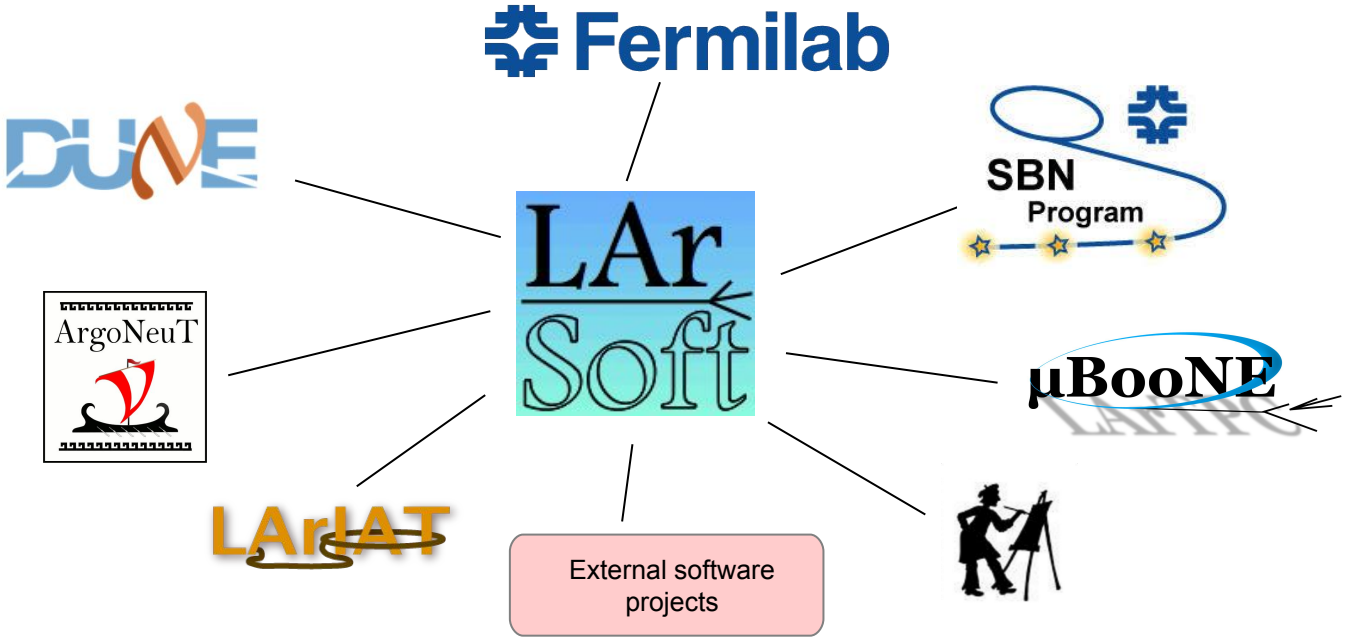
Jan 8, 2020

Outline

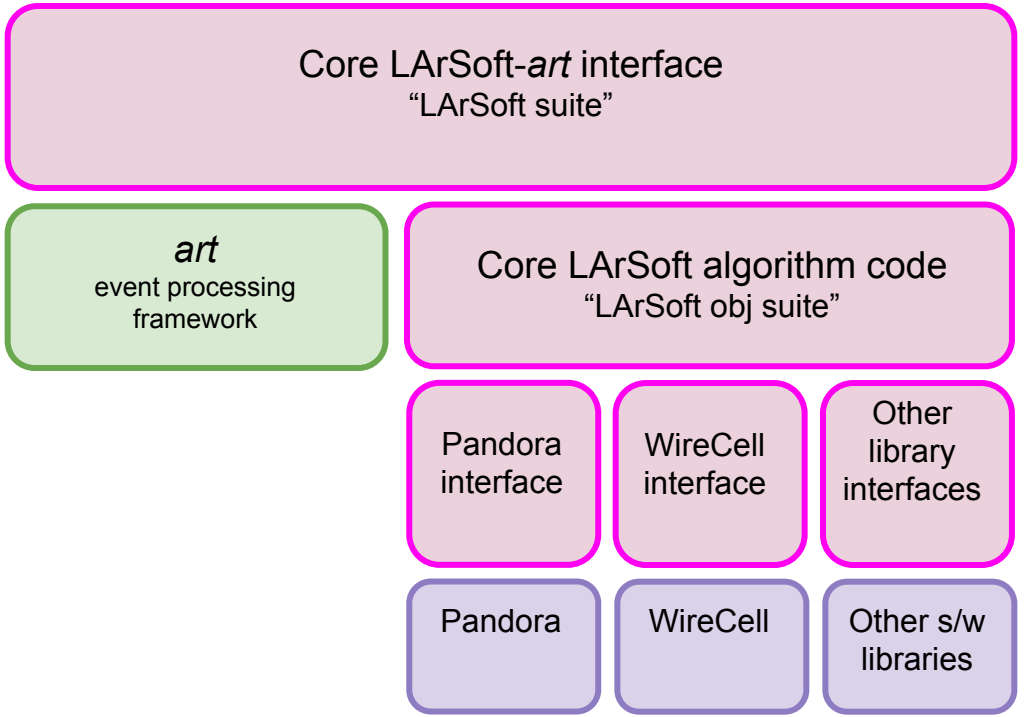
- Brief LArSoft introduction
- Current generator integration schemes
- LArSoft / generator interface design
- The data interface code

The LArSoft Collaboration

Experiments, laboratories, software projects collaborating to produce, shared experiment-independent software for LArTPC simulation, reconstruction and analysis



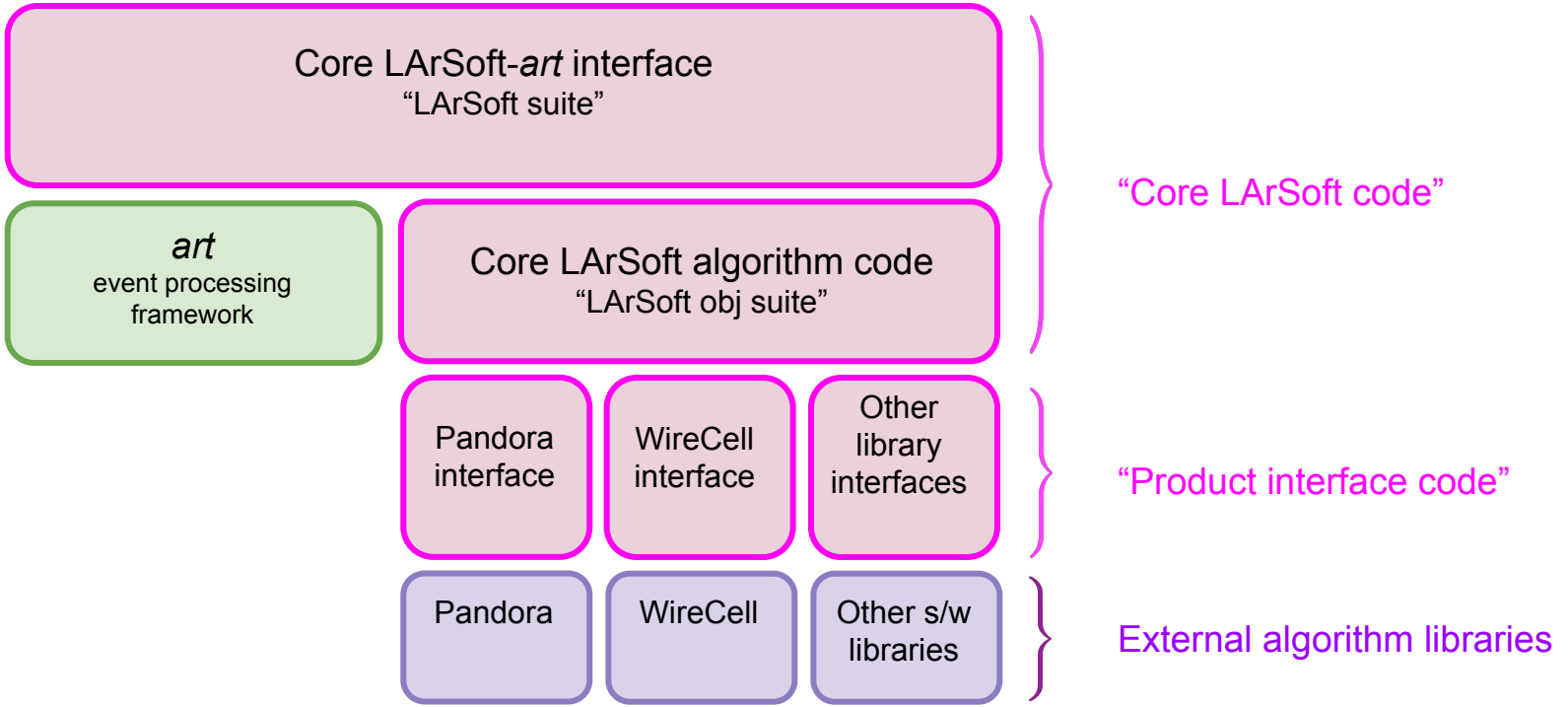
Conceptual design of LArSoft code



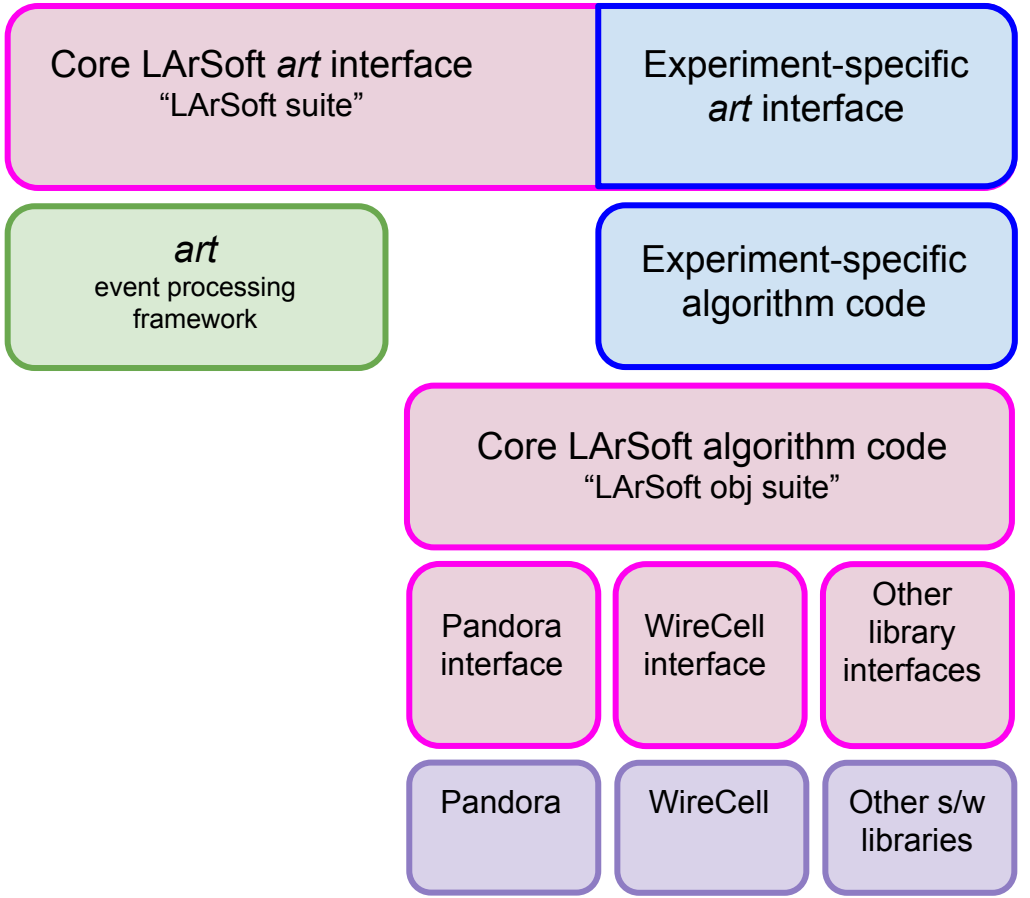
Organizing principle for LArSoft based on a layering of functionality, dependencies

Ideally, layers should only know about the **interface** to the layer **below**

Structural components of LArSoft



Experiment code



LArSoft is not stand-alone code.

Requires at least experiment / detector-specific configuration

Same basic design pertains to the experiment code

Nothing in core LArSoft code depends upon experiment code

The three generator integration schemes in use

- "Direct"
- Indirect
- Embedded

An art module runs in LArSoft
and calls external generator libraries

Examples
- Genie
  - GENIEGen_module in LArSoft
  - GENIEHelper interface in NuTools / NuGen package
- GiBUU
- MARLEY

The three generator integration schemes in use

- Direct
- “Indirect”
- Embedded

Generator is run in a stand-alone job

- LArSoft modules reads the output, fills standard generator output data structures

Examples

- NuWro
  - Reads NuWro output in root TTree file
- NDK
  - Reads NDK formatted text file
- TextFileGen
  - Reads a text file in hepevt format
- Corsika
  - Reads SQLite DB of cosmic showers

The three generator integration schemes in use

- Direct
- Indirect
- “Embedded”

Generator source is included in a LArSoft package

Examples

- SNNueAr40CCGen
  - CC interactions from SN nu_e’s
- CRY
  - Cosmic ray generator
- RadioGen
  - Radiological decay event generator

The three generator integration schemes in use

- Direct
- Indirect
- Embedded

LArSoft experience:

- “Direct” integration offers the lowest barriers to using community supported generators

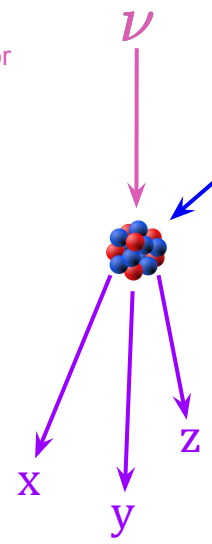
So will focus on this

The goal is to establish and maintain an architecture that allows for easy direct integration

LArSoft and elements of neutrino event generation

Stream of incident neutrinos
- Flux model from beam
- Atmospheric / solar / reactor / SN / KDAR...

Detector model
- Target material
- Geometry / flux window



Interaction model
- Nuclear modeling
- Final state interactions
- ...

Final state particles
- Input to detector simulation

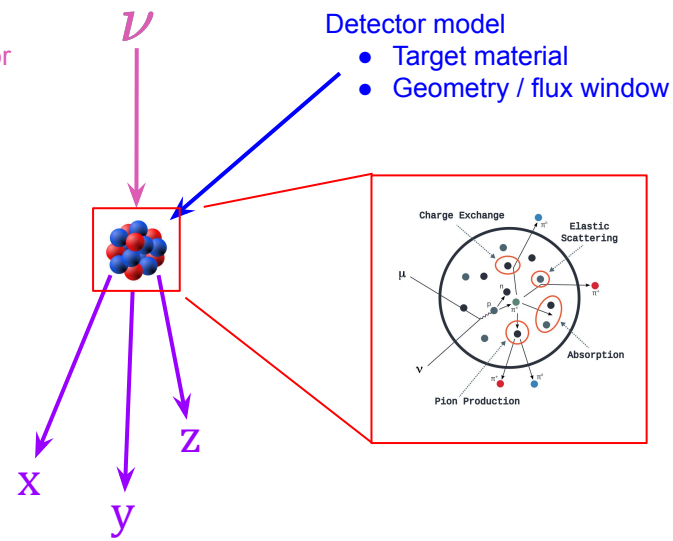
Event metadata
- Details of neutrino interaction
- Other information from generator

LArSoft and elements of neutrino event generation

Stream of incident neutrinos
- Flux model from beam
- Atmospheric / solar / reactor / SN / KDAR...

Detector model
- Target material
- Geometry / flux window

Interaction model
- Nuclear modeling
- Final state interactions
- ...



Final state particles
- Input to detector simulation

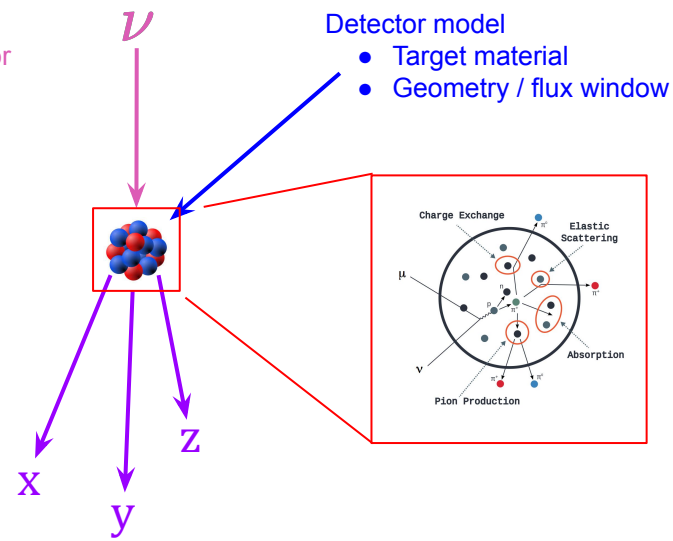
Event metadata
- Details of neutrino interaction
- Other information from generator

LArSoft and elements of neutrino event generation

Stream of incident neutrinos
- Flux model from beam
- Atmospheric / solar / reactor / SN / KDAR...

Detector model
- Target material
- Geometry / flux window

Interaction model
- Nuclear modeling
- Final state interactions
- ...



Final state particles
- Input to detector simulation

Event metadata
- Details of neutrino interaction
- Other information from generator

LArSoft / *art* event processing framework

art::Event
Stores data products for inputs / outputs / metadata

art::ServiceHandle<...>
LArSoft Geometry service
NuTools Random number service

FHiCL
Configuration data via fhicl::ParameterSet objects

art::EDProducer module
Ultimately responsible for calling generator code / storing generated output

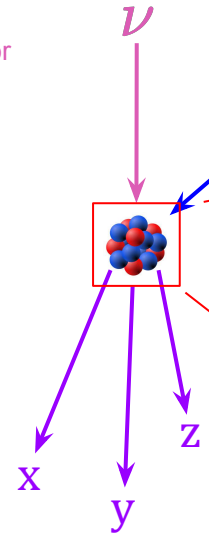
LArSoft and elements of neutrino event generation

Stream of incident neutrinos
- Flux model from beam
- Atmospheric / solar / reactor / SN / KDAR...

Interaction model
- Nuclear modeling
- Final state interactions
- ...

Final state particles
- Input to detector simulation

Event metadata
- Details of neutrino interaction
- Other information from generator



Tie these together via an interface that:

- Abstracts away specifics of generators, LArSoft, art framework, minimizing dependencies
- Provides vehicles for input configuration and neutrino data
- Provides output data needed by simulation, analyzers, allows tracing particle parentage back to earliest neutrino progenitor

Can accomplish this with common data structures, procedures to get configuration

LArSoft / *art* event processing framework

Event
Stores data products for inputs / outputs / metadata

ServiceHandle<...>
LArSoft Geometry service
NuTools Random number service

CL
Configuration data via fhicl::ParameterSet objects

EDProducer module
Ultimately responsible for calling generator code / storing generated output

LArSoft / generator interface design

External to LArSoft

Flux source model

Stream of incident neutrinos
- Flux model from beam
- Atmospheric / solar / reactor / SN / KDAR...

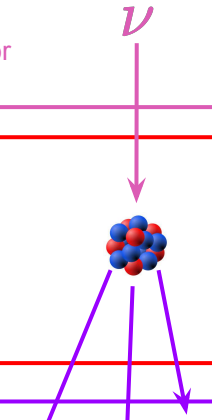
Generator

Interaction model
- Nuclear modeling
- Final state interactions
- ...

Output data

Final state particles
- Input to detector simulation

Event metadata
- Details of neutrino interaction
- Other information from generator



Defined independently of each other

LArSoft / generator interface design

External to LArSoft

Flux source model

Stream of incident neutrinos

- Flux model from beam
- Atmospheric / solar / reactor / SN / KDAR...

Generator

Interaction model

- Nuclear modeling
- Final state interactions
- ...

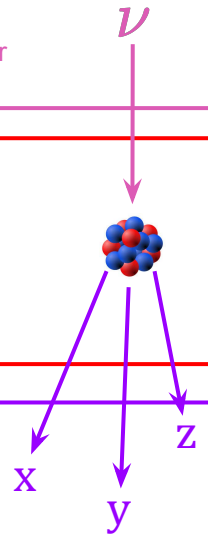
Output data

Final state particles

- Input to detector simulation

Event metadata

- Details of neutrino interaction
- Other information from generator



Interface package

(also external to LArSoft)

Generator-specific *art* module

Performs all necessary *art* interactions to configure, gather inputs

Calls generator "helper" class

Writes output in art::Event using the common output data structures

Generator-specific "helper" class

Interacts directly with the generator

LArSoft / generator interface design

External to LArSoft

Flux source model

Stream of incident neutrinos
- Flux model from beam
- Atmospheric / solar / reactor / SN / KDAR...

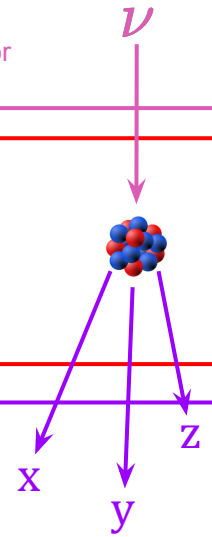
Generator

Interaction model
- Nuclear modeling
- Final state interactions
- ...

Output data

Final state particles
- Input to detector simulation

Event metadata
- Details of neutrino interaction
- Other information from generator



Interface package

(also external to LArSoft)

Generator-specific *art* module

Performs all necessary *art* interactions to configure, gather inputs

Calls generator "helper" class

Writes output in art::Event using the common output data structures

Generator-specific "helper" class

Interacts directly with the generator

LArSoft

LArSoft services
- Geometry (detector model)
- Random number
- …

LArSoft / generator interface design

External to LArSoft

Flux source model

Stream of incident neutrinos
- Flux model from beam
- Atmospheric / solar / reactor / SN / KDAR...

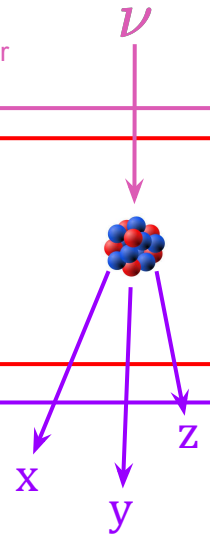
Generator

Interaction model
- Nuclear modeling
- Final state interactions
- ...

Output data

Final state particles
- Input to detector simulation

Event metadata
- Details of neutrino interaction
- Other information from generator



Interface package

(also external to LArSoft)

Generator-specific *art* module

Performs all necessary *art* interactions to configure, gather inputs

Calls generator "helper" class

Writes output in art::Event using the common output data structures

Generator-specific "helper" class

Interacts directly with the generator

LArSoft

LArSoft services
- Geometry (detector model)
- Random number
- …

A feature of LArSoft:

Functionality provided by services is separable from *art* and LArSoft

So all functional pieces can in principle operate in separate context

LArSoft / generator interface design

Flux source model

Stream of incident neutrinos
- Flux model from beam
- Atmospheric / solar / reactor / SN / KDAR...

ν

Generator

Interaction model
- Nuclear modeling
- Final state interactions
- ...

x y z

Output data

Final state particles
- Input to detector simulation

Event metadata
- Details of neutrino interaction
- Other information from generator

Generator-specific *art* module

Performs all necessary *art* interactions to configure, gather inputs

Calls generator "helper" class

Writes output in art::Event using the common output data structures

Generator-specific "helper" class

Interacts directly with the generator

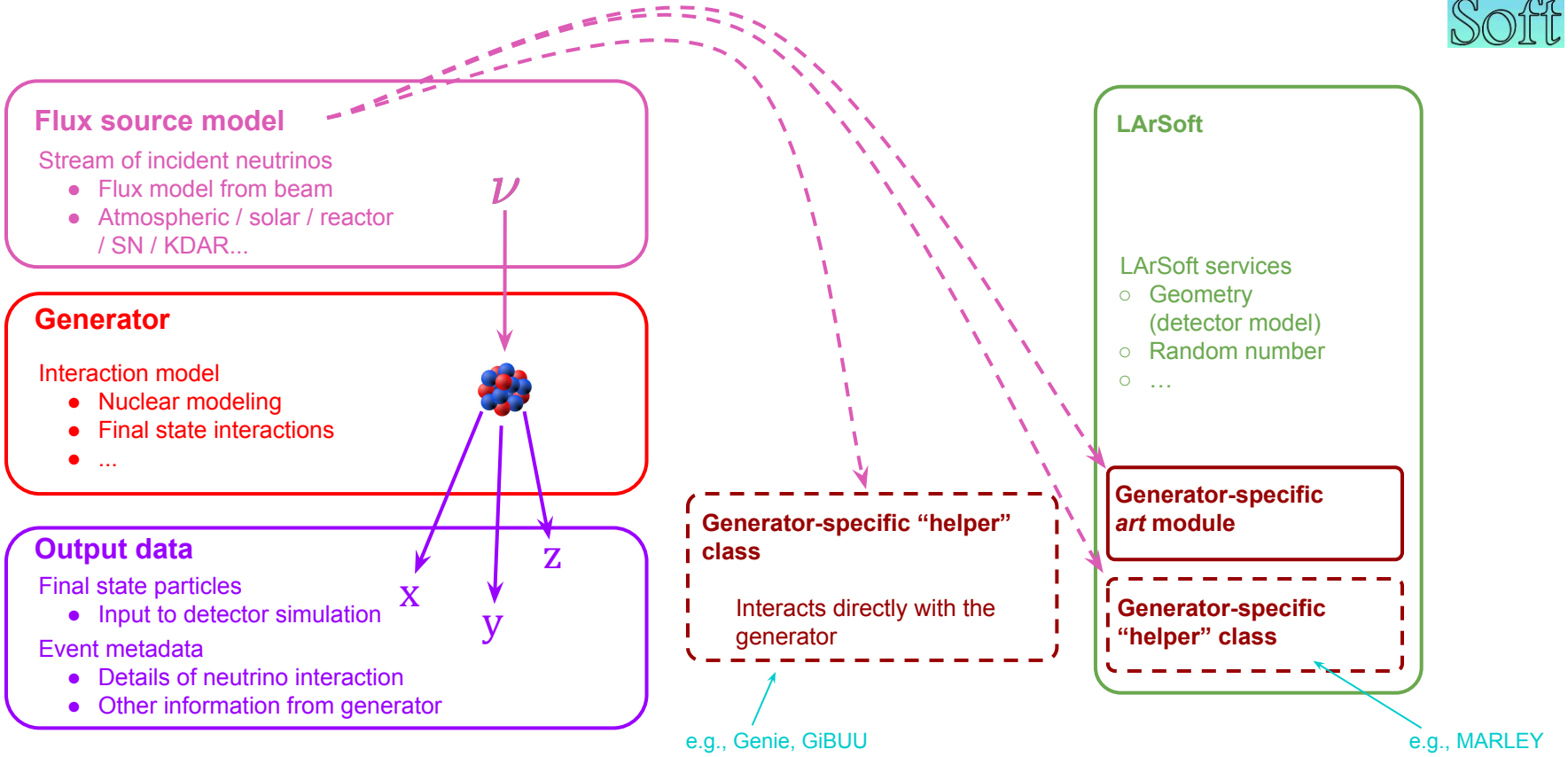
LArSoft

Flux input module
Reads input (various formats)

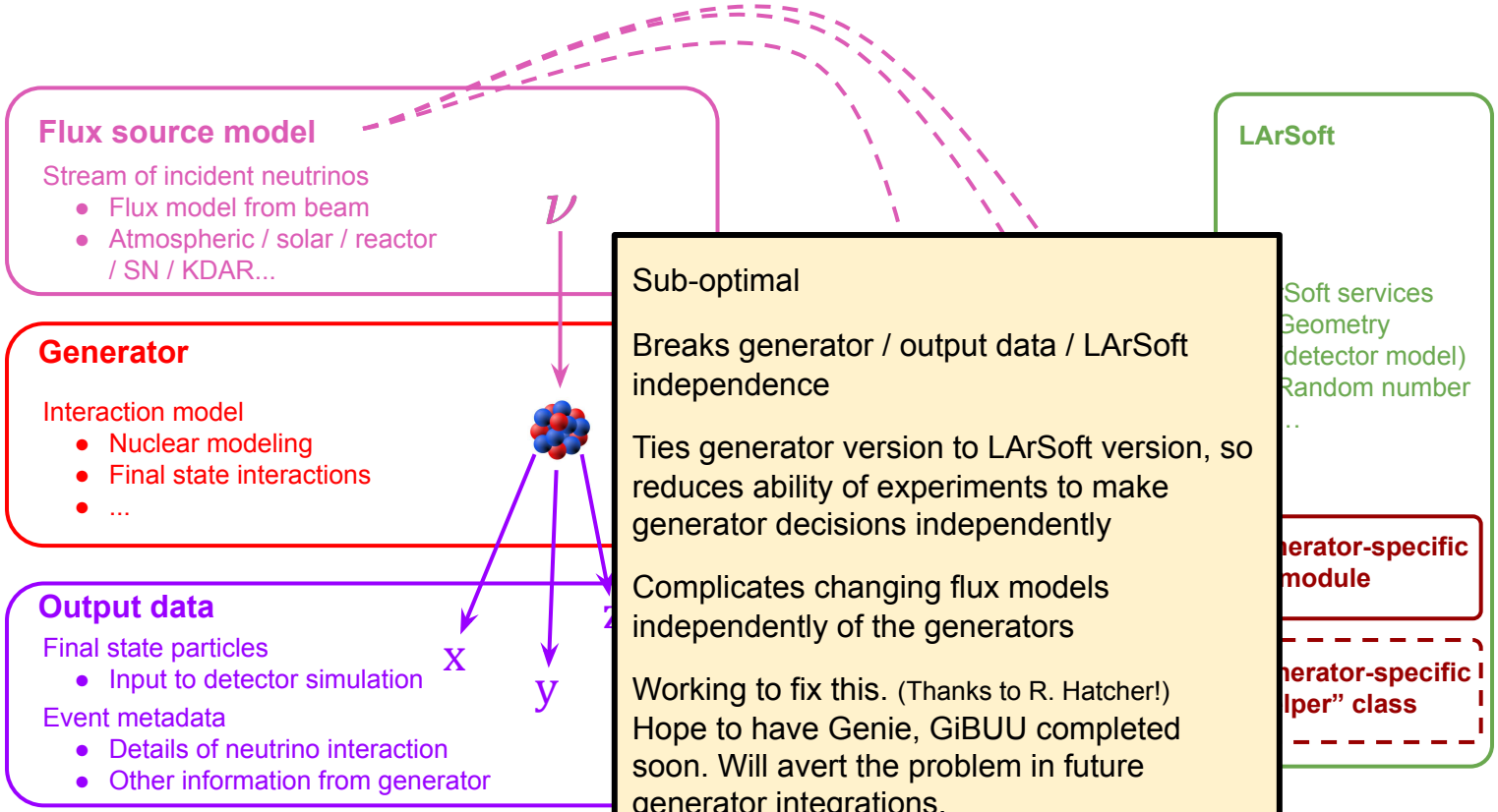
Stores incident neutrinos in art::Event

LArSoft services
- Geometry (detector model)
- Random number
- …

LArSoft / generator interface design ***in practice***



LArSoft / generator interface design *in practice*



Sub-optimal

Breaks generator / output data / LArSoft independence

Ties generator version to LArSoft version, so reduces ability of experiments to make generator decisions independently

Complicates changing flux models independently of the generators

Working to fix this. (Thanks to R. Hatcher!) Hope to have Genie, GiBUU completed soon. Will avert the problem in future generator integrations.

The specifics

Note: all interface data structures defined in the NuTools product
https://cdcvs.fnal.gov/redmine/projects/nutools/wiki

- nusimdata/SimulationBase
  http://nusoft.fnal.gov/larsoft/doxsvn/html/dir_07cfd2ee7aa7d354d1822fe672c1b00e.html

The specifics

Candidate input data structure

simb::MCFlux
http://nusoft.fnal.gov/larsoft/doxsvn/html/MCFlux_8h_source.html

Well suited to accelerator beam sources
(taking no position on other potential sources)

```
namespace simb{

  enum flux_code_{
    kHistPlusFocus  = +1,
    kHistMinusFocus = -1,
    kGenerator      =  0,
    kNtuple         =  2,
    kSimple_Flux    =  3,
    kDk2Nu          =  4
  };

  class MCFlux {

  public:

    MCFlux();

    // maintained variable names from gnumi ntuples
    // see http://www.hep.utexas.edu/~zarko/wwwgnumi/v19/[/v19/output_gnumi.html]

    int    frun;
    int    fevtno;
    double fndxdz;
    double fndydz;
    double fnpz;
    double fnenergy;
    double fndxdznea;
    double fndydznea;
    double fnenergyn;
    double fnwtnear;
    double fndxdzfar;
    double fndydzfar;
    double fnenergyf;
    double fnwtfar;
    int    fnorig;
    int    fndecay;
    int    fntype;
    double fvx;
    double fvy;
    double fvz;
    double fpdpx;
    double fpdpy;
    double fpdpz;
    double fppdxdz;
```

The specifics

Output data structures

simb::MCTruth http://nusoft.fnal.gov/larsoft/doxsvn/html/MCTruth_8h_source.html

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

private:

  std::vector<simb::MCParticle> fPartList;
  simb::MCNeutrino              fMCNeutrino;
  simb::Origin_t                fOrigin;
  simb::MCGeneratorInfo         fGenInfo;
  bool                          fNeutrinoSet;

public:
  const simb::MCGeneratorInfo&  GeneratorInfo()     const;
  simb::Origin_t                Origin()            const;
  int                           NParticles()        const;
  const simb::MCParticle&       GetParticle(int i)  const;
  const simb::MCNeutrino&       GetNeutrino()       const;
  bool                          NeutrinoSet()       const;

  void            Add(simb::MCParticle const& part);
  void            Add(simb::MCParticle&& part);
  void            SetGeneratorInfo(simb::Generator_t generator,
                                   const std::string & genVersion,
                                   const std::unordered_map<std::string, std::string>&
genConfig);
  void            SetOrigin(simb::Origin_t origin);
  void            SetNeutrino(int CCNC,
                              int mode,
                              int interactionType,
                              int target,
                              int nucleon,
                              int quark,
                              double w,
                              double x,
                              double y,
                              double qsqr);
```

Contains:

- a collection of simb::MCParticle
- a simb::MCNeutrino
- a simb::MCGeneratorInfo
- a few options for neutrino source

The specifics

Output data structures

simb::MCParticle

http://nusoft.fnal.gov/larsoft/doxsvn/html/MCParticle_8h_source.html

Contains:

- Particle type
- Kinematic variables
- Creation vertex / process
- Mother / daughter indices
- ...

```
class MCParticle  {
public:

  // An indicator for an uninitialized variable (see MCParticle.cxx).
  static const int s_uninitialized;

  MCParticle();

protected:
  typedef std::set<int>   daughters_type;

  int                     fstatus;
  int                     ftrackId;
  int                     fpdgCode;
  int                     fmother;
  std::string             fprocess;
  std::string             fendprocess;
  simb::MCTrajectory      ftrajectory;
  double                  fmass;
  TVector3                fpolarization;
  daughters_type          fdaughters;
  double                  fWeight;
  TLorentzVector          fGvtx;
  int                     frescatter;

public:

  // Standard constructor.  If the mass is not supplied in the
  // argument, then the PDG mass is used.
  // status code = 1 means the particle is to be tracked, default it to be tracked
  // mother = -1 means that this particle has no mother
  MCParticle(const int trackId,
             const int pdg,
             const std::string process,
             const int mother  = -1,
             const double mass = s_uninitialized,
             const int status  = 1);


  // our own copy and move assignment constructors (default)
  MCParticle(MCParticle const &)            = default; // Copy constructor.
  MCParticle& operator=( const MCParticle&) = default;
  MCParticle(MCParticle&&) = default;
  MCParticle& operator= (MCParticle&&) = default;


  // constructor for copy from MCParticle, but with offset trackID
  MCParticle(MCParticle const&, int);

  // Accessors.
  //
  // The track ID number assigned by the Monte Carlo.  This will be
  // unique for each Particle in an event. - 0 for primary particles
  int TrackId() const;

  // Get at the status code returned by GENIE, Geant4, etc
  int StatusCode() const;

  // The PDG code of the particle.  Note that Geant4 uses the
  // "extended" system for encoding nuclei; e.g., 1000180400 is an
  // Argon nucleus.  See "Monte Carlo PArticle Numbering Scheme" in
  // any Review of Particle Physics.
  int PdgCode() const;

  // The track ID of the mother particle.  Note that it's possible
  // for a particle to have a mother that's not recorded in the
  // Particle list; e.g., an excited nucleus with low kinetic energy
  // emits a photon with high kinetic energy.
  int Mother() const;

  const TVector3& Polarization() const;
  void            SetPolarization( const TVector3& p );

  // The detector-simulation physics process that created the
  // particle. If this is a primary particle, it will have the
  // value "primary"
  std::string Process()   const;

  std::string EndProcess()   const;
  void SetEndProcess(std::string s);

  // Accessors for daughter information.  Note that it's possible
  // (even likely) for a daughter track not to be included in a
  // Particle list, if that daughter particle falls below the energy cut.
  void AddDaughter( const int trackID );
  int  NumberDaughters()                  const;
  int  Daughter(const int i)              const; //> Returns the track ID for the "i-th" daughter.

  // Accessors for trajectory information.
  unsigned int NumberTrajectoryPoints() const;

  // To avoid confusion with the X() and Y() methods of MCTruth
  // (which return Feynmann x and y), use "Vx,Vy,Vz" for the
  // vertex.
  const TLorentzVector& Position( const int i = 0 ) const;
  double                Vx(const int i = 0)         const;
  double                Vy(const int i = 0)         const;
  double                Vz(const int i = 0)         const;
  double                 T(const int i = 0)         const;

  const TLorentzVector& EndPosition() const;
  double                EndX()        const;
  double                EndY()        const;
  double                EndZ()        const;
  double                EndT()        const;

  const TLorentzVector& Momentum( const int i = 0 ) const;
  double                Px(const int i = 0)         const;
  double                Py(const int i = 0)         const;
  double                Pz(const int i = 0)         const;
  double                 E(const int i = 0)         const;
```

The specifics

Output data structures

simb::MCNeutrino
http://nusoft.fnal.gov/larsoft/doxsvn/html/MCNeutrino_8h_source.html

Contains:
- Some details of the neutrino interaction

```
class MCNeutrino {
public:

  MCNeutrino();

private:

  simb::MCParticle fNu;
  simb::MCParticle fLepton;
  int              fMode;
  int              fInteractionType;
  int              fCCNC;
  int              fTarget;
  int              fHitNuc;
  int              fHitQuark;
  double           fW;
  double           fX;
  double           fY;
  double           fQSqr;

public:

  MCNeutrino(simb::MCParticle &nu,
             simb::MCParticle &lep,
             int CCNC,
             int mode,
             int interactionType,
             int target,
             int nucleon,
             int quark,
             double w,
             double x,
             double y,
             double qsqr);

  const  simb::MCParticle& Nu()              const;
  const  simb::MCParticle& Lepton()          const;
  int                      CCNC()            const;
  int                      Mode()            const;
  int                      InteractionType() const;
  int                      Target()          const;
  int                      HitNuc()          const;
  int                      HitQuark()        const;
  double                   W()               const;
  double                   X()               const;
  double                   Y()               const;
  double                   QSqr()            const;
  double                   Pt()              const;
  double                   Theta()           const;
  friend std::ostream&  operator<< (std::ostream& output,
};
}
```

The specifics

Output data structures

- An aside
  - Though sufficient for simulation, should note that the combination of
    - simb::MCTruth with simb::MCParticle and simb::MCNeutrino

    is insufficient to re-create information tracked in Genie event (for instance), and that is needed in re-weighting. Likely a common problem.
  - Addressed by adding simb::GTruth
    http://nusoft.fnal.gov/larsoft/doxsvn/html/GTruth_8h_source.html

This may be an adequate solution in general, but should perhaps consider alternatives?

- Though the cost of changing such low-level code is high...

The specifics

Output data structures

simb::MCGeneratorInfo
http://nusoft.fnal.gov/larsoft/doxsvn/html/MCGeneratorInfo_8h_source.html

Contains:
- Which generator
- The configuration
  - Note: art preserves the FHiCL configuration in the output file, which may not be reflective of entire generator configuration

```
  typedef enum class _ev_generator
  {
    kUnknown,
    kGENIE,
    kCRY,
    kGIBUU,
    kNuWro,
    kMARLEY,
    kNEUT,
    kCORSIKA,
    kGEANT,
    kNumGenerators, //  this should always be the last entry
  } Generator_t;

  struct MCGeneratorInfo
  {
    simb::Generator_t                                generator;
    std::string                                      generatorVersion;
    std::unordered_map<std::string, std::string> generatorConfig;

    MCGeneratorInfo(Generator_t gen = Generator_t::kUnknown,
                    const std::string ver = "",
                    const std::unordered_map<std::string, std::string> config = {})
      : generator(gen), generatorVersion(ver), generatorConfig(config)
    {}
  };
}
```

Summary

- LArSoft promotes a standard architecture that simplifies independent development, running of multiple neutrino source models and event generators to drive simulation, analysis
  - Some existing generators are partially within this scheme, others not at all
  - Working to bring all generators into this model
  - Open to evolving existing scheme as needed to meet requirements
- Current data interface model is known to be incomplete
  - Would be good to address this on both input and output sides
- Hope we can facilitate making additional generators available

The end