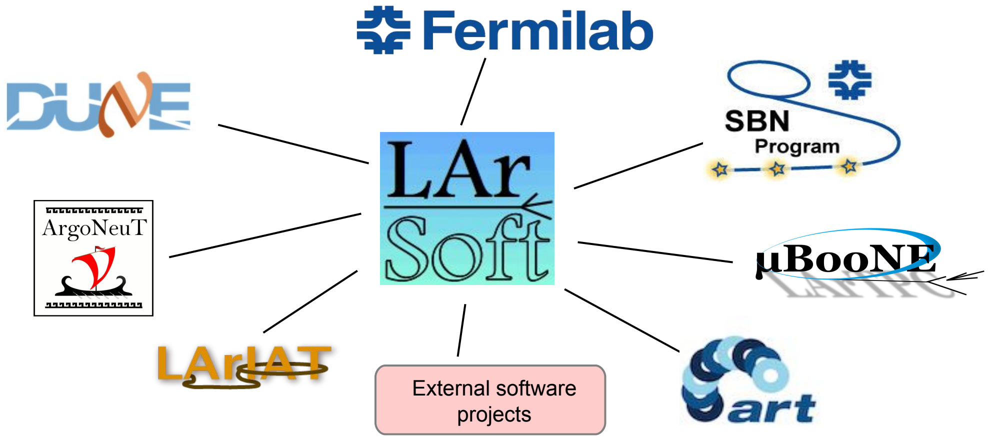
LArSoft and frameworks

Erica Snider
June 6, 2023
Fermilab Frameworks Workshop

A reminder: what is LArSoft?

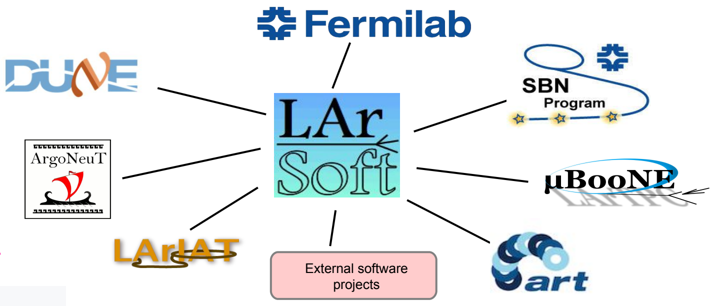
- A Collaboration

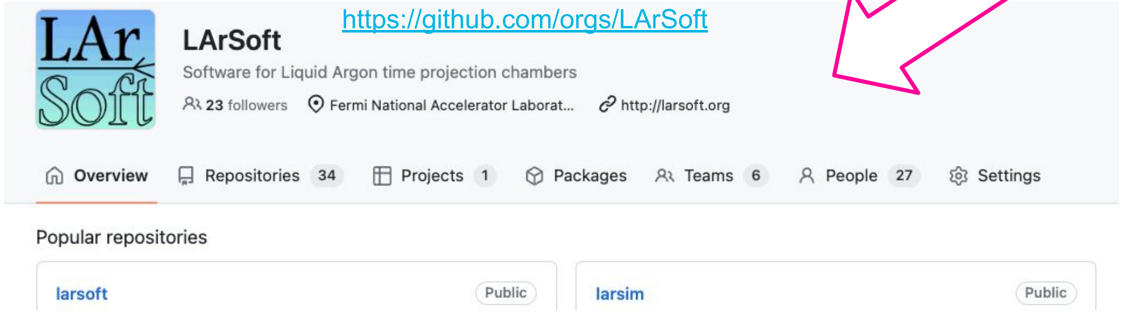


Experiments, laboratories, software projects collaborating to produce shared, detector-independent software for LArTPC simulation, reconstruction and analysis

A reminder: what is LArSoft?

- A Collaboration
- A body of shared software





https://github.com/orgs/LArSoft

LArSoft

Software for Liquid Argon time projection chambers

23 followers · Fermi National Accelerator Laborat... · http://larsoft.org

Overview · Repositories 34 · Projects 1 · Packages · Teams 6 · People 27 · Settings

Popular repositories

larsoft Public	larsim Public

Algorithms, tools and utilities for the simulation, reconstruction and analysis of LArTPC data developed, contributed, maintained by the experiments

A reminder: what is LArSoft?

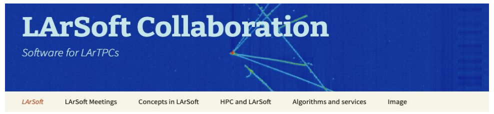
- A Collaboration
- A body of shared software
- Fermilab "project" team (SciSoft)

Supports sharing of software by

- owning architecture, infrastructure, coding guidelines
- providing user and developer support, software expertise, release management, change management, (shared) documentation, etc.
- hosting user / developer workshops

Oversight by the experiment spokes via "Steering Group"

LArSoft.org


LArSoft Collaboration
Software for LArTPCs
LArSoft · LArSoft Meetings · Concepts in LArSoft · HPC and LArSoft · Algorithms and services · Image

LArSoft wiki

LArSoft Documentation

- LArSoft wiki
- LArSoft.org: The LArSoft Collaboration
- The LArSoft github organization
- Notes about updating these pages

Issue tracker

LArSoft
+ · Overview · Activity · Roadmap · Issues
Issues

A reminder: what is LArSoft?

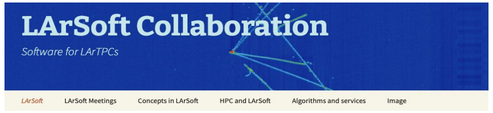
- A Collaboration
- A body of shared software
- Fermilab "project" team (SciSoft)

Supports sharing of software by

+ owning architecture, infrastructure, coding guidelines
+ providing user and developer support, software expertise, release management, change management, (shared) documentation, etc.
+ hosting user / developer workshops

Oversight by the experiment spokes via "Steering Group"

LArSoft.org



LArSoft wiki

LArSoft Documentation

- LArSoft wiki
- LArSoft.org: The LArSoft Collaboration
- The LArSoft github organization
- Notes about updating these pages

Issue tracker

Start with LArSoft design principles and practices

The philosophies and rules that underlie code sharing in core LArSoft code

1. Detector interoperability
2. Use of standardized algorithm / service interfaces
3. Separation of framework and algorithm code
4. Write code that is thread safe
5. Modularity
6. Design / write testable units of code
7. Document code in the source
8. Continuous integration

Start with LArSoft design principles and practices

The philosophies and rules that underlie code sharing in core LArSoft code

1. Detector interoperability
2. Use of standardized algorithm / service interfaces
3. Separation of framework and algorithm code
4. Write code that is thread safe
5. Modularity
6. Design / write testable units of code
7. Document code in the source
8. Continuous integration

Most relevant to the discussion today

LArSoft design principles and practices

The philosophies and rules that underlie code sharing in core LArSoft code

1. Detector interoperability
2. Use of standardized algorithm / service interfaces
3. Separation of framework and algorithm code
4. Write code that is thread safe
5. Modularity
6. Design / write testable units of code
7. Document code in the source
8. Continuous integration

The foundation of the code sharing paradigm

The nature of LArTPCs allows for the use of many common algorithms with differences expressed via configuration or hidden behind common interfaces

Common data products are central element of this

LArSoft design principles and practices

The philosophies and rules that underlie code sharing in core LArSoft code

1. Detector interoperability
2. Use of standardized algorithm / service
3. **Separation of framework and algorithm**
4. Write code that is thread safe
5. Modularity
6. Design / write testable units of code
7. Document code in the source
8. Continuous integration

- Central to the discussion today
- Generally a good practice
- Allows use of LArSoft algorithm code outside of art, such as:
  - Lightweight analysis frameworks
  - Specialized development / debugging environments
- In principle, reduces cost of migration to new framework, should that be needed

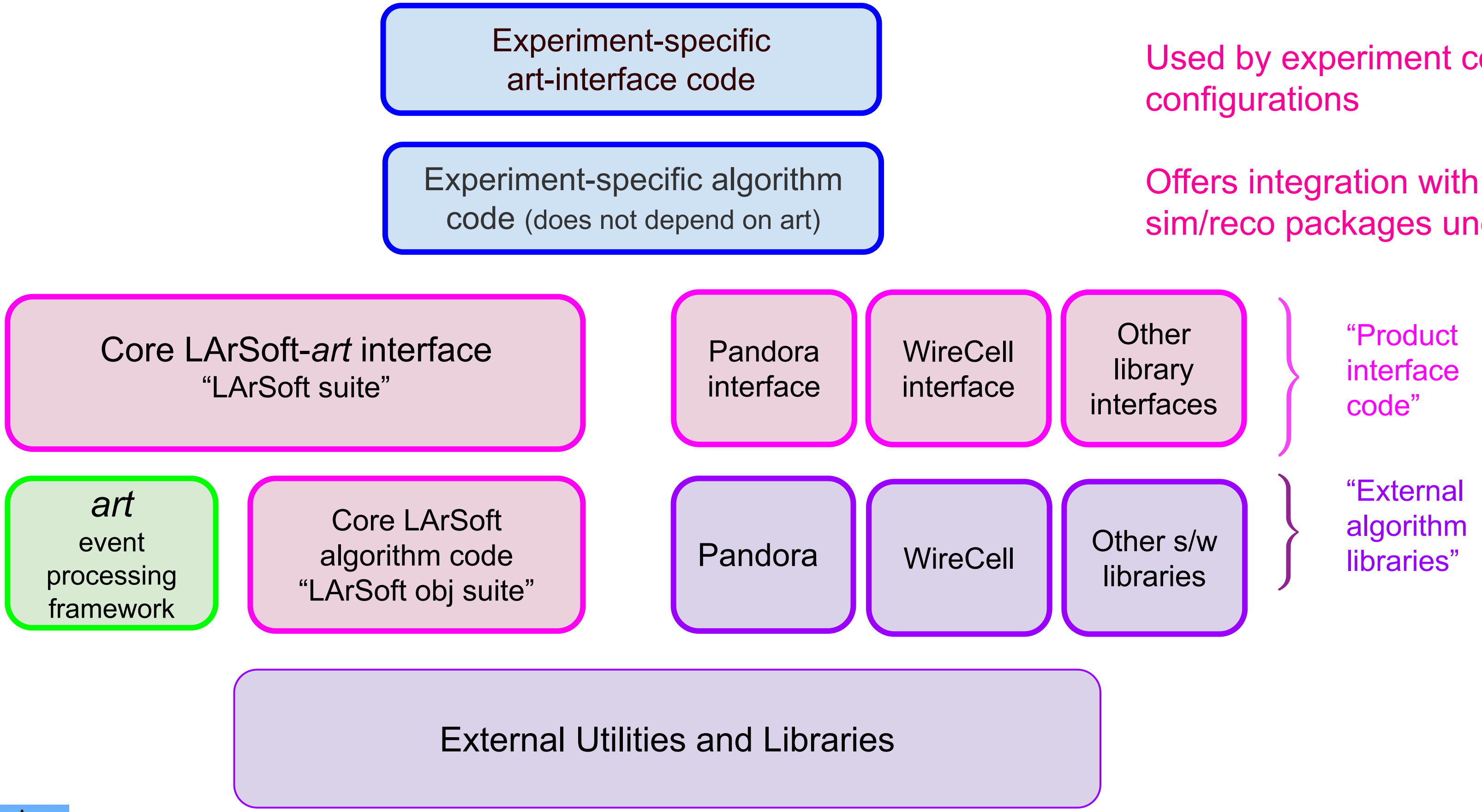
LArSoft design principles and practices

The philosophies and rules that underlie code sharing in core LArSoft code

1. Detector interoperability
2. Use of standardized algorithm / service interfaces
3. Separation of framework and algorithm code
4. Write code that is thread safe
5. Modularity
6. Design / write testable units of coc
7. Document code in the source
8. Continuous integration

Connects to the framework through coordination of thread pools and scheduling across multi-threading units, when that is needed.

Conceptual design

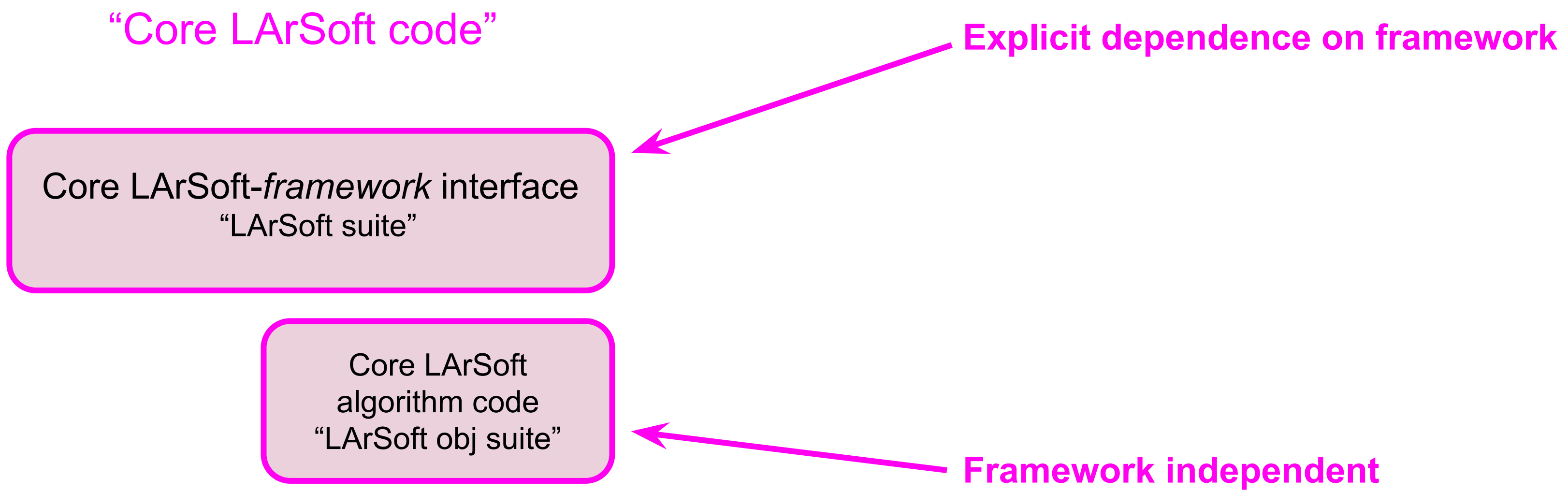

Experiment-specific art-interface code
Experiment-specific algorithm code (does not depend on art)
Core LArSoft-*art* interface "LArSoft suite"
Pandora interface
WireCell interface
Other library interfaces
"Product interface code"
art event processing framework
Core LArSoft algorithm code "LArSoft obj suite"
Pandora
WireCell
Other s/w libraries
"External algorithm libraries"
External Utilities and Libraries

LArSoft is not stand-alone

Used by experiment code, detector configurations

Offers integration with other sim/reco packages under *art*

Conceptual design


"Core LArSoft code"
Explicit dependence on framework
Core LArSoft-*framework* interface
"LArSoft suite"
Core LArSoft algorithm code
"LArSoft obj suite"
Framework independent

Conceptual design

“Core LArSoft code”

Repositories

larcore
lardata
larreco

larevt
larsim
larana

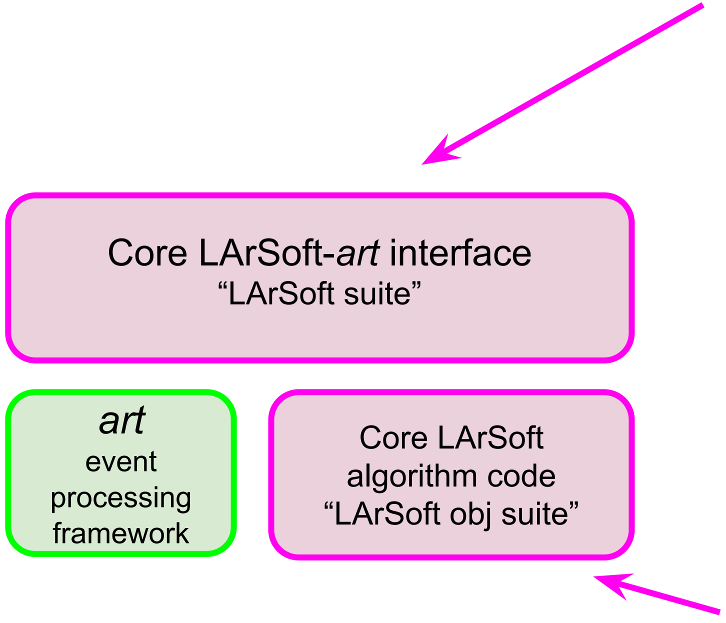
lareventdisplay
…

Core LArSoft-*framework* interface
“LArSoft suite”

Core LArSoft
algorithm code
“LArSoft obj suite”

larcore**alg**
larcore**obj**
lardata**alg**
lardata**obj**
…

Conceptual design



Definition of *art* modules

- Interactions with **art::Event** to retrieve and store **data products**
- Access to module-specific configuration data via **fhicl parameter sets**
- Event processing ***art* state transitions**
- Access to **art Service** and **art Tool** plugins

Definition of art Services

- Contains an *art*-independent **“service provider”** class
- Aware of *art* **state transitions**

Definition of art Tools

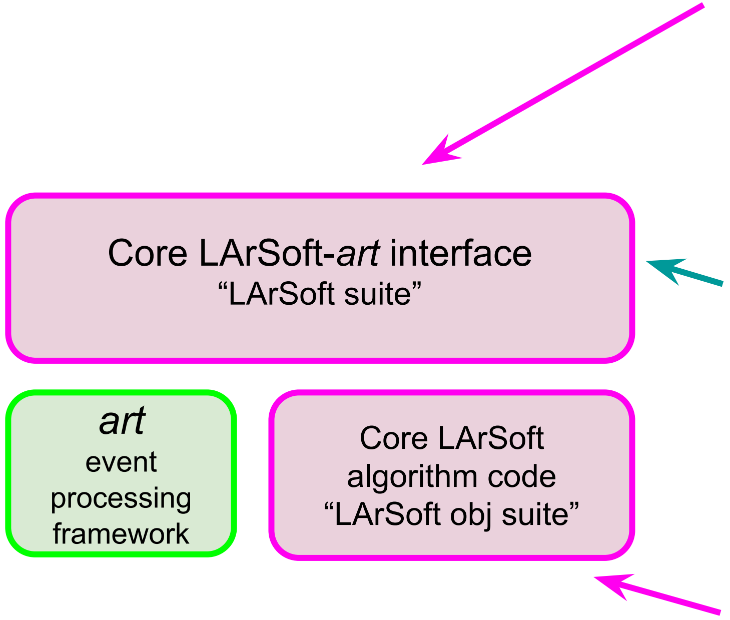
Algorithm code and utilities

- Event data, service “providers”, tool algorithms, fhicl parameter sets **passed into algorithms / utilities**

Definition of service providers

- **Do all the work** required of the service
- **Not aware of framework state transitions**

Conceptual design *in practice*



Definition of *art* modules

- Interactions with **art::Event** to retrieve and store **data products**
- Access to module-specific configuration data via **fhicl parameter sets**
- Event processing ***art* state transitions**
- Access to **art Service** and **art Tool** plugins

Definition of art Services

- Contains an *art*-independent **"service provider"** class
- Aware of *art* **state transitions**

Definition of art Tools

Algorithm code

- **Direct use of art::Event to store / retrieve data products**
- **Direct access to *art* Services and Tool plugins**

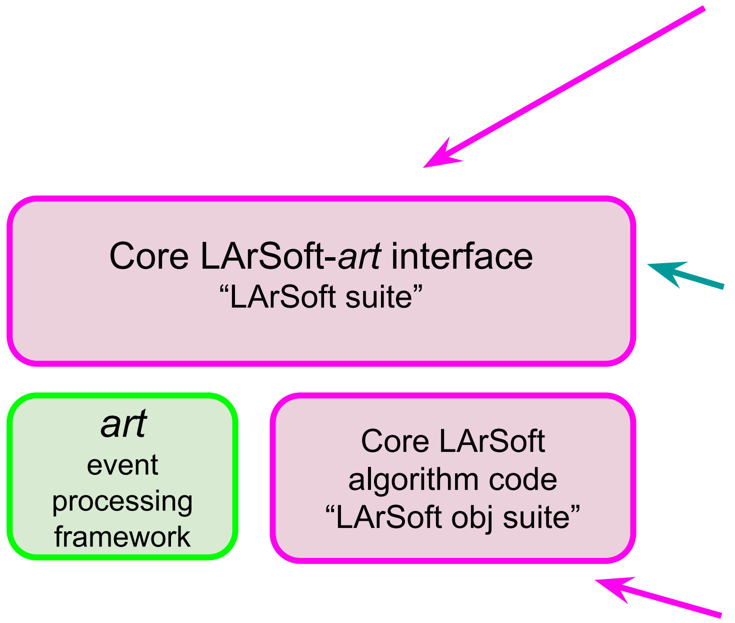
Algorithm code and utilities

- Event data, service "providers", tool algorithms, fhicl parameter sets **passed into algorithms / utilities**

Definition of service providers

- **Do all the work** required of the service
- **Not aware of framework state transitions**

Conceptual design *in practice*



Core LArSoft-*art* interface
"LArSoft suite"
art
event processing framework
Core LArSoft algorithm code
"LArSoft obj suite"

Definition of *art* modules

- Interactions with **art::Event** to retrieve and store **data products**
- Access to module-specific configuration data via **fhicl parameter sets**
- Event processing ***art* state transitions**
- Access to **art Service** and **art Tool** plugins

Definition of art Services

- Contains an *art*-independent **"service provider"** class
- Aware of *art* **state transitions**

Definition of art Tools

Algorithm code

- **Direct use of art::Event to store / retrieve data products**
- **Direct access to *art* Services and Tool plugins**

Algorithm code and utilities

ter sets

D

Have demonstrated that this code runs in Gallery and LArLite, two light-weight analysis environments that do not depend on *art*

Conceptually a thin connection of LArSoft to framework

- Algorithms / service providers are atomic
  - Run on data served to them
    - How it is served is a function of framework + framework interface

- Both also thread safe at a minimum (thought not all are in practice, yet)
  - Have demonstrated algorithm-level multi-threading in production workflows
    - See Giuseppe Cerati's SciDAC-4 presentation from yesterday
  - Multi-threading currently based on TBB, which is owned by framework

- Generally speaking, however, "LArSoft" (and the project team) is not a dominant driver of framework requirements / features

A couple of observations on current framework functionality

- Difficult to manage memory in cases where "event" data is large
  - Granularity of memory-resident data is confined to the "event"
    - Based on model that assumes "event" = a small number of drift windows (e.g., three) for a small number of TPCs (e.g., one)
  - Very limited ability to manage chunks of data smaller than an "event"
  - Algorithms already don't care how large / small the data is

- No facility to deal with asynchronous data / continuous readout streams
  - Demands of continuous data streams are very different than for beam data
  - Kyle touched on this during "Future framework development" discussion yesterday

A couple of observations on current framework functionality

- Support for concurrency and HPC
  - Concurrency needed now to solve some grid utilization efficiency problems
    - Jobs allocate multiple slots for memory, but run single threaded leaving CPUs idle
    - Grid resources are currently important, and will remain important looking ahead
    - Current support seems adequate for this
  - Expect the use case for “external work” (in CMS language) to grow
    - LArSoft already has GPUaaS for ML inferencing
    - Many other algorithms well suited to GPUs or other accelerators
    - Can we do this efficiently? Do we, e.g., need to recover latency time?
    - Will need to understand efficiency issues as this evolution occurs
- Added flexibility in configuration tools can help simplify specification in cases
  - Allow for calculable configurations (e.g., loops, conditionals)
    - Echoes a Mu2e wish list item discussed by Rob Kutschke yesterday
  - Access to the environment, and environment variables in particular

The end

Backup

Separation of framework and algorithm

A LArSoft algorithm must be able to perform its task using only:

- LArSoft data products and their associations (input and output data)
- Service providers
- FHiCL parameter sets
- Calls to message_service

Write art modules that:

- Get configuration data from ParameterSet passed to module
- Get data products from, and put them into the event
- Get service instances
- Create algorithm instances (if they are classes)
- Call algorithm methods, passing data products, service providers, ParameterSet(s)

Separation of framework and algorithm

Gallery

gallery provides lightweight access to event data in *art*/ROOT files outside the *art* event processing framework.

gallery is not an alternative framework; rather, it provides a library that can be used to write programs that need to read (but not write) *art*/ROOT files. You must have access to the ROOT dictionaries for the classes in a data file to use that data file. The availability of such dictionaries is provided by the experiments.

gallery is built:

- without the use of EDProducers, EDAnalyzers, etc., thus
- without the facilities of the framework (e.g. callbacks from framework transitions, writing of *art*/ROOT files).

Algorithm code may be called within code that uses Gallery for event access

Separation of framework and algorithm

Canvas

The canvas package is the infrastructure required for providing I/O operations for the full art framework and the lightweight gallery framework. In particular, the ROOT dictionaries art provides for experiments to use are located in canvas.

A tutorial is available at: https://github.com/marcpaterno/gallery-demo

Algorithm code may use Canvas internally to support data product associations

Project team works to annually revised work plan

- Describes the high-level plan of work for the project team.
  - Developed through process of one-on-one meetings with experiments followed by iterations on the draft until presentation to / approval by the Steering Group
  - Reflects experiment requirements and requests
  - Implements the strategic directions for the shared code of the collaboration

  LArSoft / SciSoft play a strong leadership role in defining direction, strategy
- The 2023 LArSoft Work Plan

2023 LArSoft Work Plan

Strategic directions of the 2023 work plan

1. Support multi-threading to optimize running on grid resources
2. Enable / facilitate optimized running on GPU and HPC resources
3. Facilitate / simplify integration of machine learning workflows
4. Support heterogeneous detector readouts in simulation and reconstruction
5. Provide a multi-experiment capable event display framework
6. Expand adoption of community / industry supported tools

2023 LArSoft Work Plan

Strategic directions of the 2023 work plan

1. Support multi-threading to optimize running on grid resources
2. Enable / facilitate optimized running on GPU and HPC resources
3. Facilitate / simplify integration of machine learning paradigms
4. Support heterogeneous detector readouts in simulation and reconstruction
5. Provide a multi-experiment capable event display framework
6. Expand adoption of community / industry supported tools

LArSoft held “LArTPC Multi-threading and Acceleration Workshop” Mar 2–3

(will come back to this at end…)

1. Support multi-threading

- All experiments report using multiple grid slots to accommodate memory of jobs
  - Running single-threaded programs leads to significant underutilization of CPU
  - Memory use driven by event-level data
  - Need sub-event level multi-threading or more granular data management strategies

- Project work
  - Working with experiments to ensure thread-safety in common and experiment-specific code
  - Implementing multi-threading in common services
  - Past integration of contributions from SciDAC4 efforts

2. Enable / facilitate optimized running on GPU and HPC

- Many LArTPC computing problems highly parallelizable that can benefit from hardware acceleration
  - Low-level data and signal processing
  - Simulation
  - Machine learning
- Multi-threading, GPU acceleration create paths to optimized running on HPC
  - Several experiments / projects have experience with LArSoft code on HPC
  - Demonstrated in SciDAC4 work by Giuseppe Cerati, Sophie Berkman, et al.
- Project work
  - Focus on making low-level data structures suited to GPU processing,
  - Work with experiments on specific algorithms (to be identified)
  - Current Spack migration well suited to needs of HPC-enabled builds

3. Facilitate / simplify integration of machine learning workflows

- Most ML efforts within experiments are completely external to LArSoft
  - MicroBooNE experience:
    - ML-based analysis branch all but isolated to small group of analyzers.
    - Separate data production workflows required, which slowed data availability
  - Integration into LArSoft would alleviate all these issues
- Some ML algorithms benefit from GPU acceleration at inference stage
  - Highly dependent on the problem and solution
  - Some overlap with acceleration work previously noted
- Experiment groups in ICARUS and ND-LAr working on fully ML workflows
- Project work
  - Ensure configurations, inputs and outputs are available to ML interfaces
  - Assist experiment groups with interfacing to LArSoft
  - Past integration of Sonic-derived GPUaaS into LArSoft targeted ML inferencing

4. Support heterogeneous detector readouts in sim and reco

- Primarily aimed at accommodating pixelated readouts (ND-LAr)
  - Also intended to allow future detectors to have completely different readout schemes

- Project work
  - Adapt geometry and simulation systems
    - Portions of reconstruction code must differ
      - Will be provided by experiments
  - Geometry: requires re-factoring readout from volume geometry
    - Several wire-plane readout configurations already supported
    - Readout geometry currently tightly intertwined with more generic volume geometry
  - Past work adapted simulation via similar abstraction of anode simulation
    - The "artg4tk / LArG4" re-factoring completed several years ago

5. Provide a multi-experiment capable event display framework

- A persistent and vocal ask from many experiments
- Would add value in exactly the same way that common sim/reco do.

- Project work
  - Design, develop event display framework, or adapt an existing ED to requirements
  - Experiments provide customizing code

Requires local ED / visualization expertise, which is currently lacking
  - Can view this as a request to build this expertise

6. Expand adoption of community / industry supported tools

- A good strategy wherever possible and cost effective
- Recent major examples
  - Migration to GitHub (last year)
  - Migration to Spack (continuing)
- Project work
  - Nothing beyond existing work currently in plan, but always seeking opportunities

SciSoft team

- Vito di Benedetto
- Patrick Gartung
- Chris Green
- Robert Hatcher
- Kyle Knoepfel (co-lead)
- Lynn Garren (ret.)
- Marc Paterno
- Saba Sehrish
- Erica Snider (co-lead)
- Mike Wang
- Hans Wenzel