Introduction to LArSoft

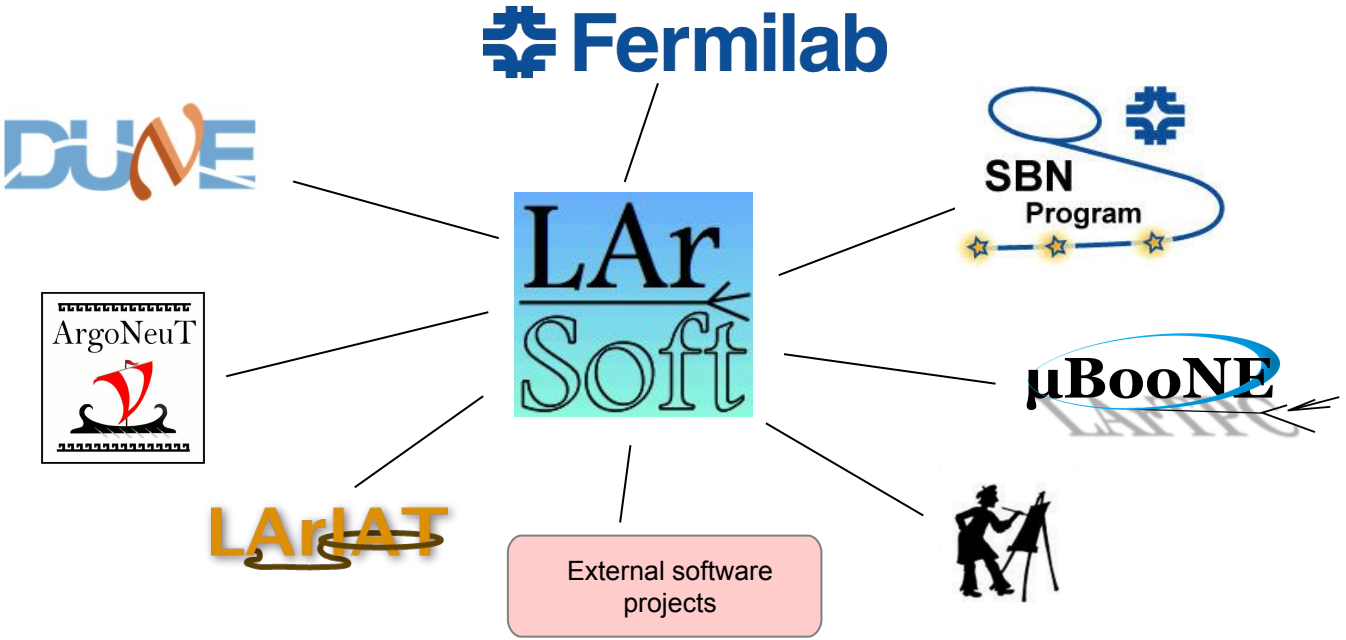
Erica Snider, *Fermilab*
on behalf of SciSoft Team
LArSoft 2019 Summer Workshop

Outline

- Overview of LArSoft
- LArSoft design
- Design principles and coding practices
- Contents of LArSoft
- Code releases and distribution
- End-user / developer resources

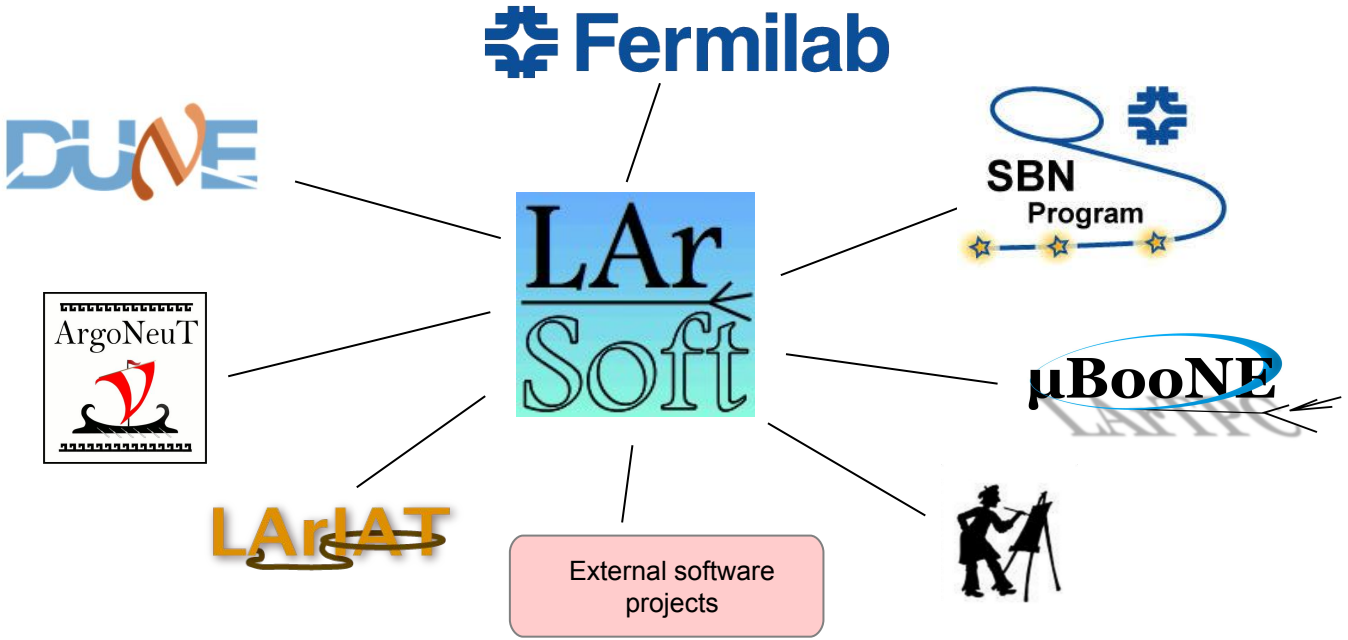
The LArSoft Collaboration

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



The LArSoft Collaboration

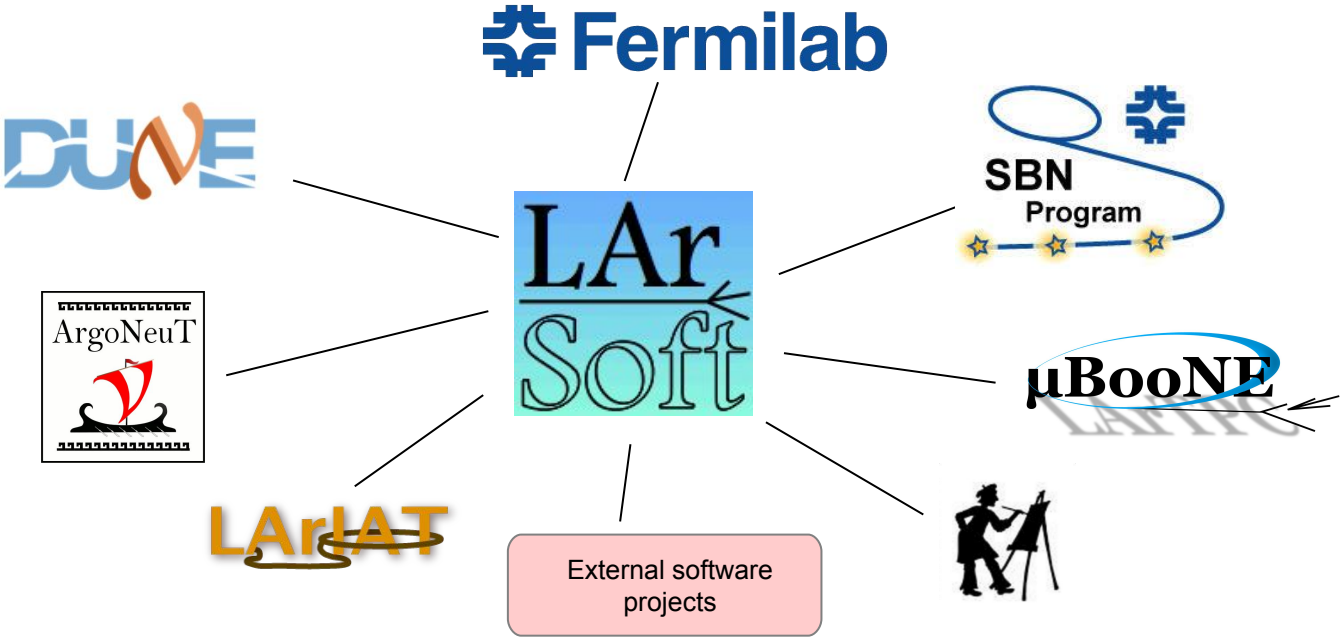
The body of shared software is also referred to as “LArSoft”



The LArSoft Collaboration

Each experiment

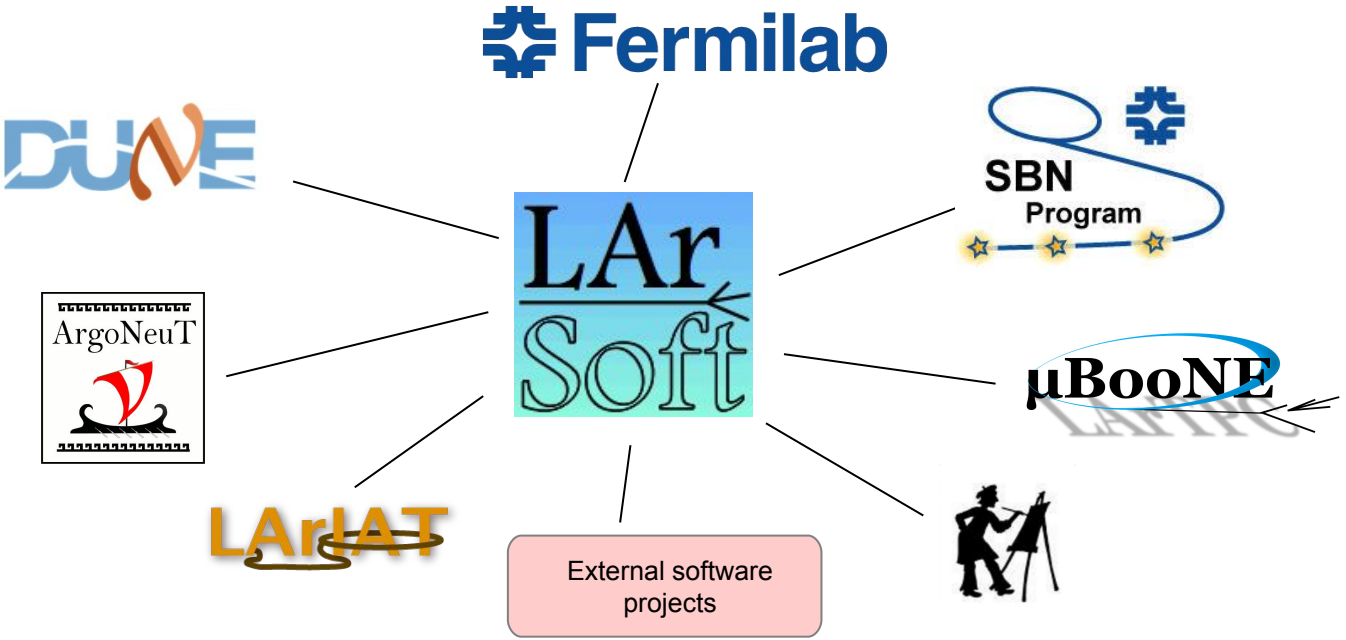
- Contributes to the shared, core LArSoft code. (All members have write access.)
- Maintains detector-specific software, configuration that builds on the core code



The LArSoft Collaboration

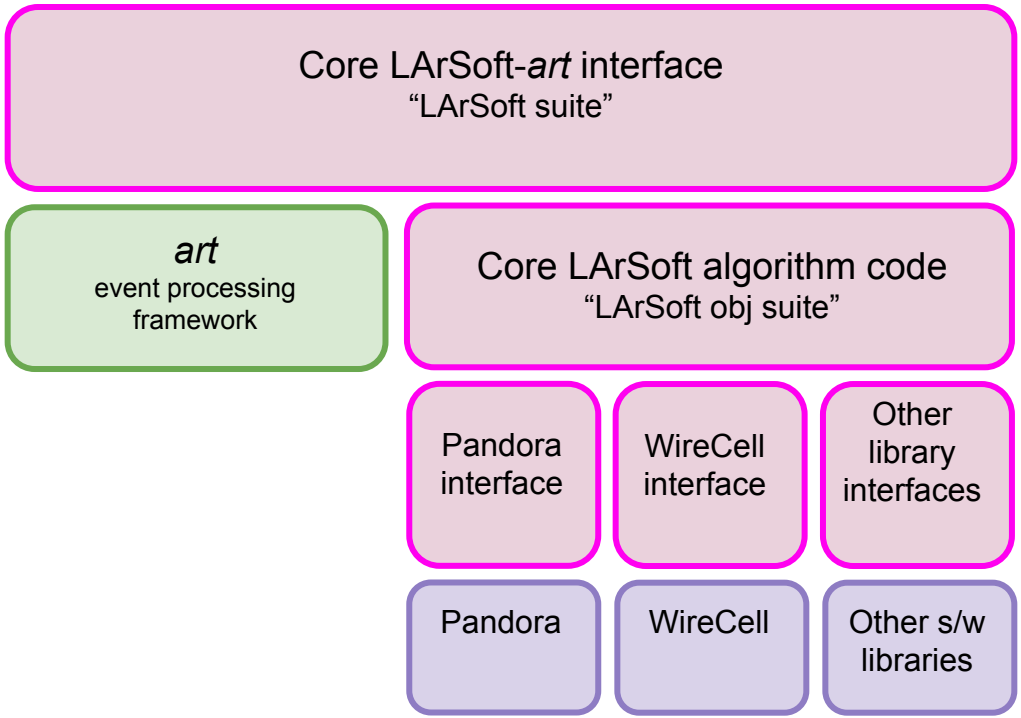
The LArSoft "project": a Fermilab-based group that

- maintains / develops the architecture
- provides user support, software expertise, release management



LArSoft design

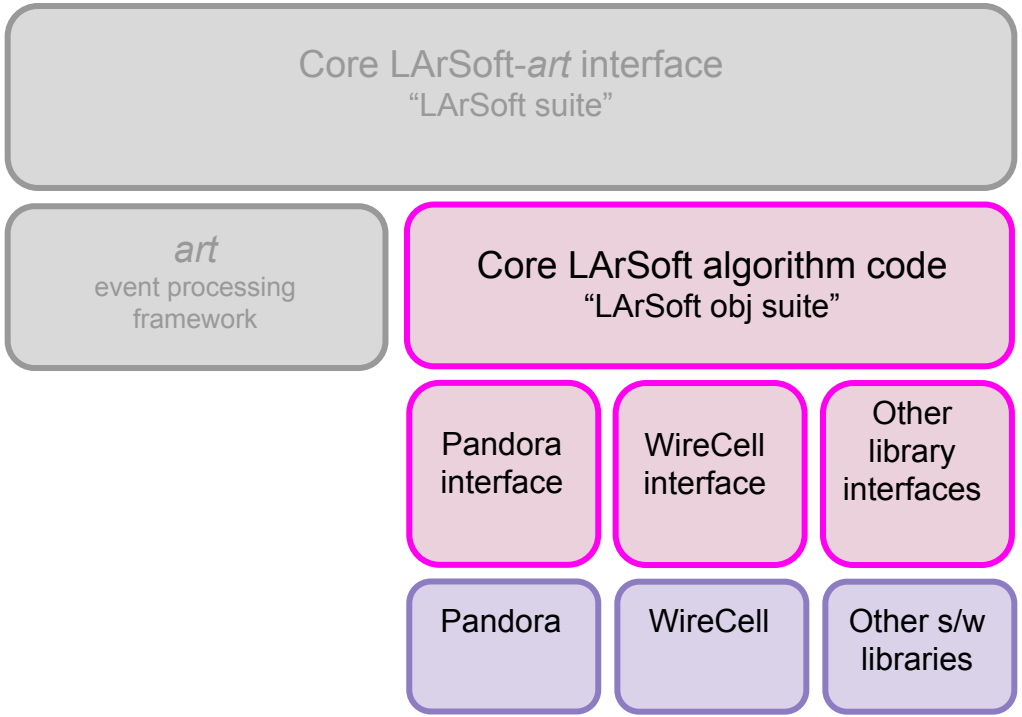
Conceptual design of LArSoft code



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

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

Conceptual design of LArSoft code



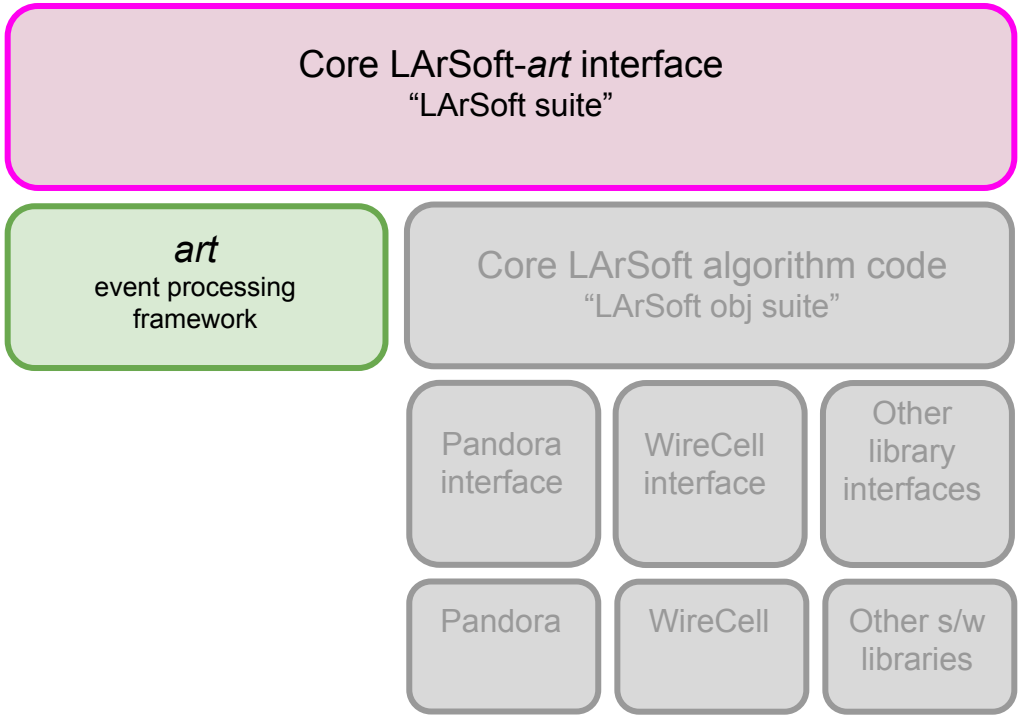
Organizing principle for LArSoft based on a layering of functionality, dependencies

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

E.g.: Neither LArSoft obj suite nor anything below it knows about or depends on *art*

- This has interesting implications, which will be discussed later

Conceptual design of LArSoft code



LArSoft built on top of *art* event processing framework

The *art* event processing framework

art
event processing framework

Quick art tutorial

- **Reads events from user-specified input source**
- **Executes workflow of tasks as configured via input FHiCL file**
  - Operate on “data products” stored in event records
- **Tasks (algorithms, event filtering, ...) carried out via user-specified “modules” and other “plug-ins”**
  - Dynamically-loaded
  - Can be user-written
  - Configurable via FHiCL files
- **Output data products may be written to output file(s)**

The *art* event processing framework

art
event processing framework

Quick art tutorial

Three types of plug-ins

1. **Modules**
   - The basic, scheduled elements within task workflows.
     - *art* calls pre-defined methods at specific times in the event loop
   - Three types
     - Producer: may modify the event
     - Filter: can alter trigger path execution
     - Analyzer: may not modify the event
2. **Services**
   - Classes with global scope that can be accessed within modules.
     - *art* calls registered methods at specific times in the event loop
3. **Tools**
   - Functions or classes with module scope that have user-specified interface to perform tasks

The *art* event processing framework

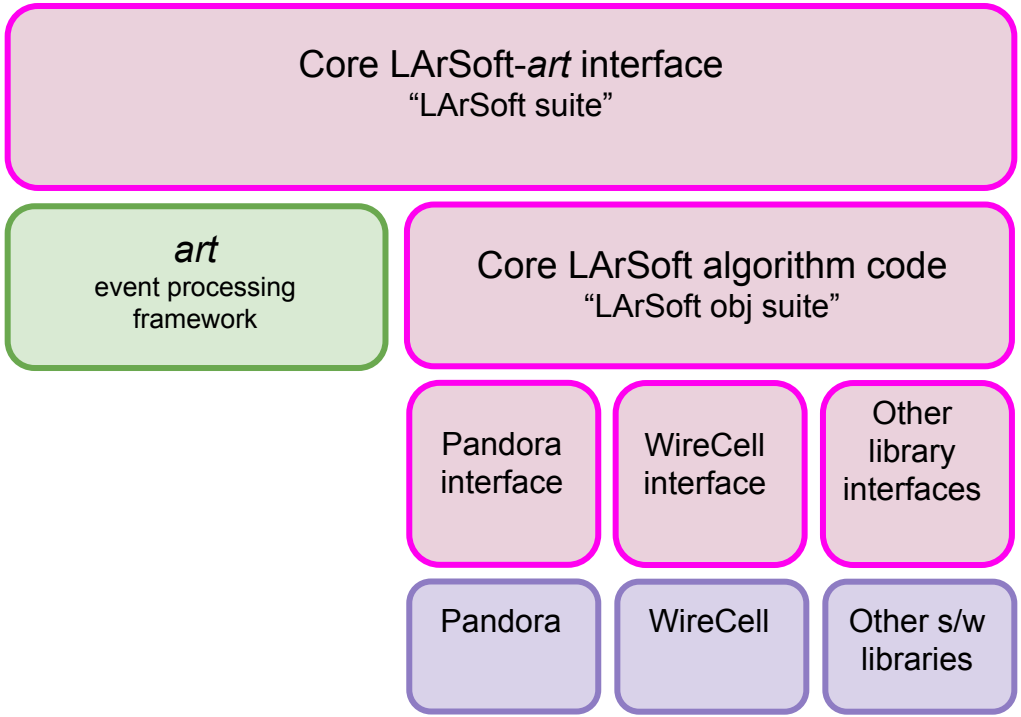
art
event processing framework

Quick art tutorial

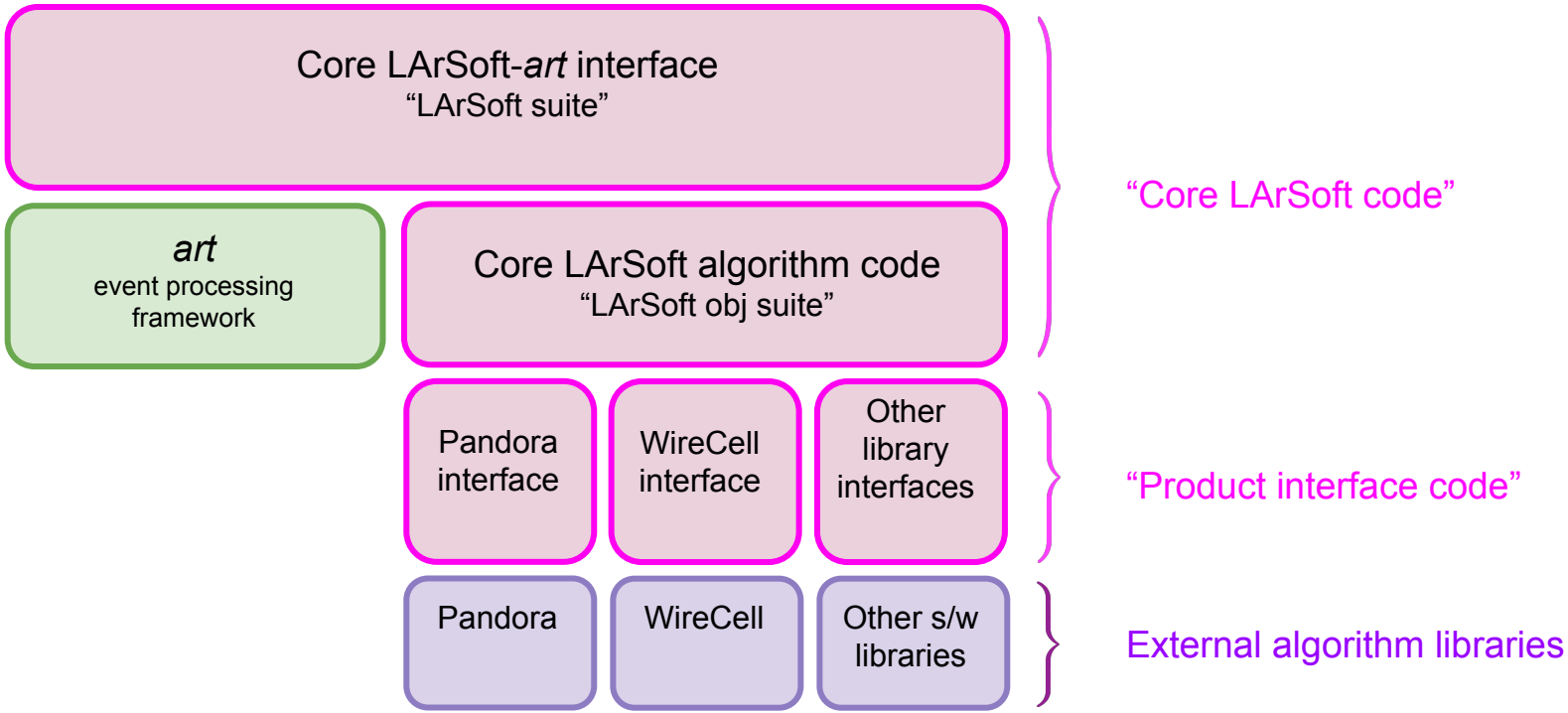
More information:

- The art documentation site: resources, detailed tutorials
  - https://art.fnal.gov/
- The art wiki: reference information, coding guidelines, issue tracker
  - https://cdcvs.fnal.gov/redmine/projects/art/wiki
- The FHiCL quick start guide
  - https://cdcvs.fnal.gov/redmine/documents/327
- The FHiCL-cpp wiki: C++ bindings
  - https://cdcvs.fnal.gov/redmine/projects/fhicl-cpp/wiki

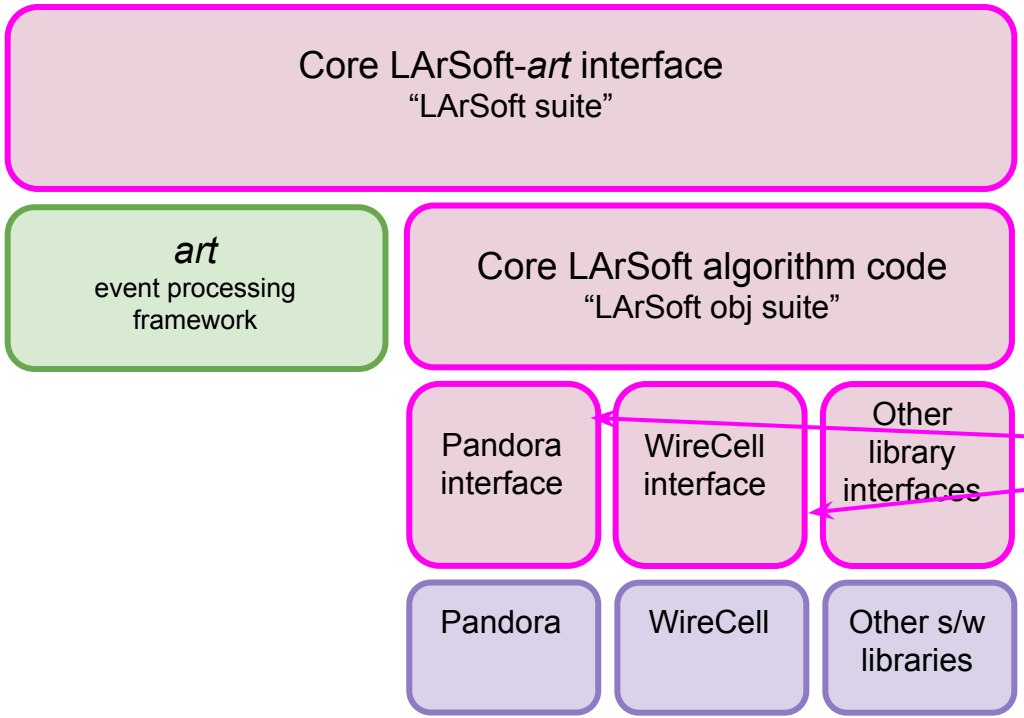
Structural components of LArSoft



Structural components of LArSoft



Structural components of LArSoft



Repositories

larcore larevt lareventdisplay
lardata larsim larpandora
larreco larana ...

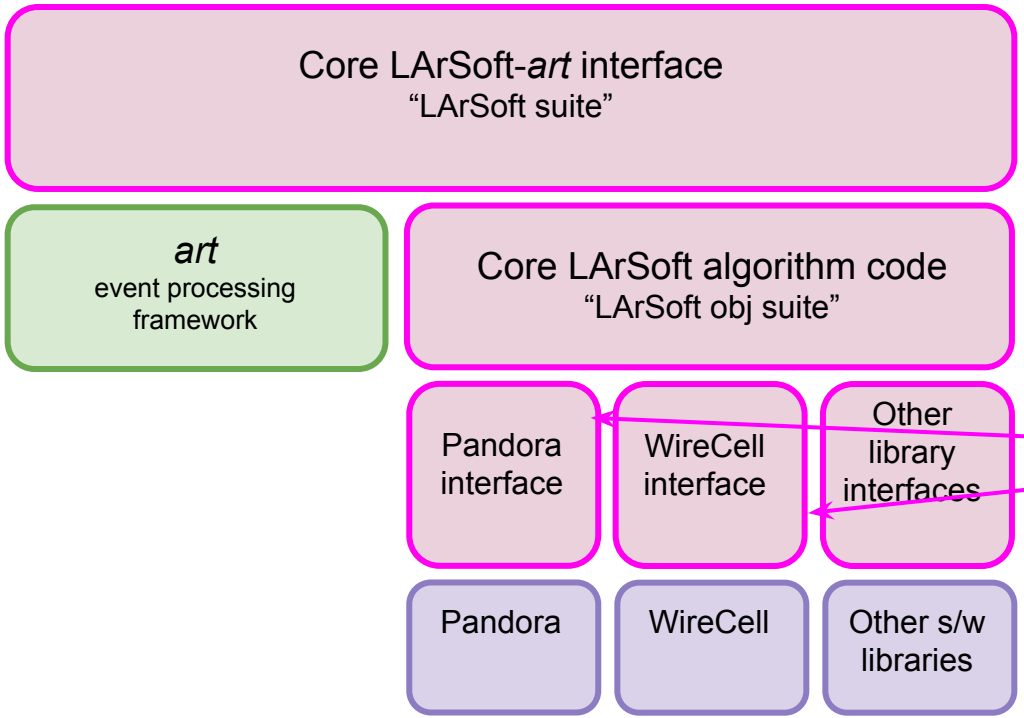
larcore**alg** lardata**obj**
larcore**obj** ...
lardata**alg**

LArSoft obj naming convention:
ends in "obj" or "alg"

larpandoracontent
larwirecell
...

Fermilab

Structural components of LArSoft



Installed ups products

larcore larevt lareventdisplay
lardata larsim larpandora
larreco larana ...

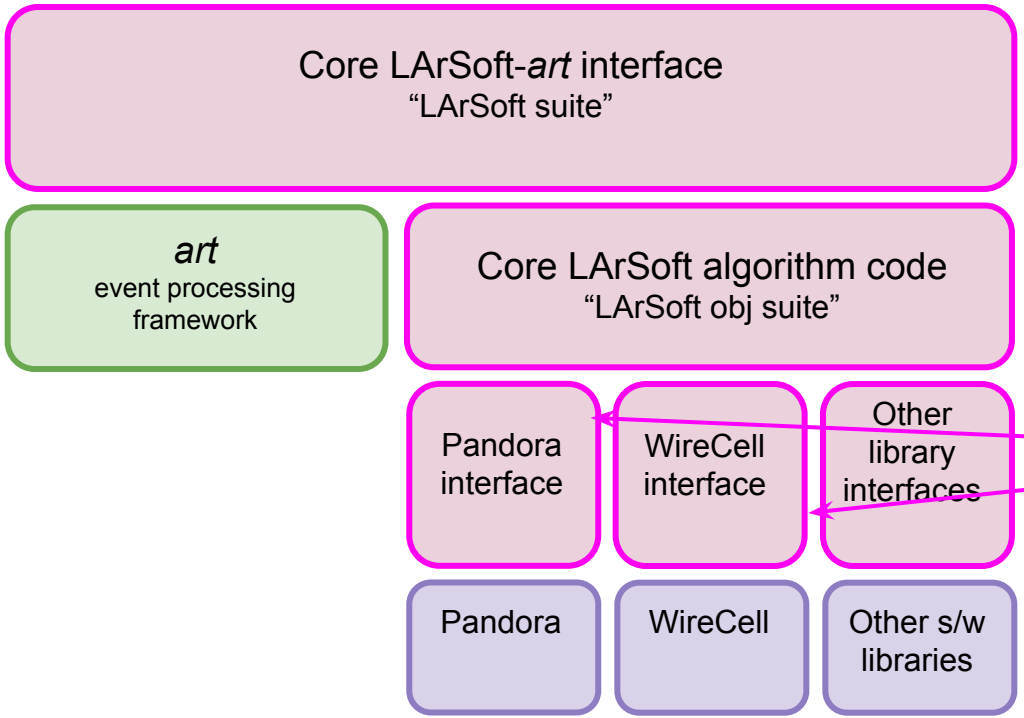
larcore**alg** lardata**obj**
larcore**obj** ...
lardata**alg**

larpandoracontent
larwirecell
...

The smallest build unit is the repository.

One installed ups product instance per repository

Structural components of LArSoft



Installed ups products

larcore larevt lareventdisplay
lardata larsim larpandora
larreco larana ...

larcore**alg** lardata**obj**
larcore**obj** ...
lardata**alg**

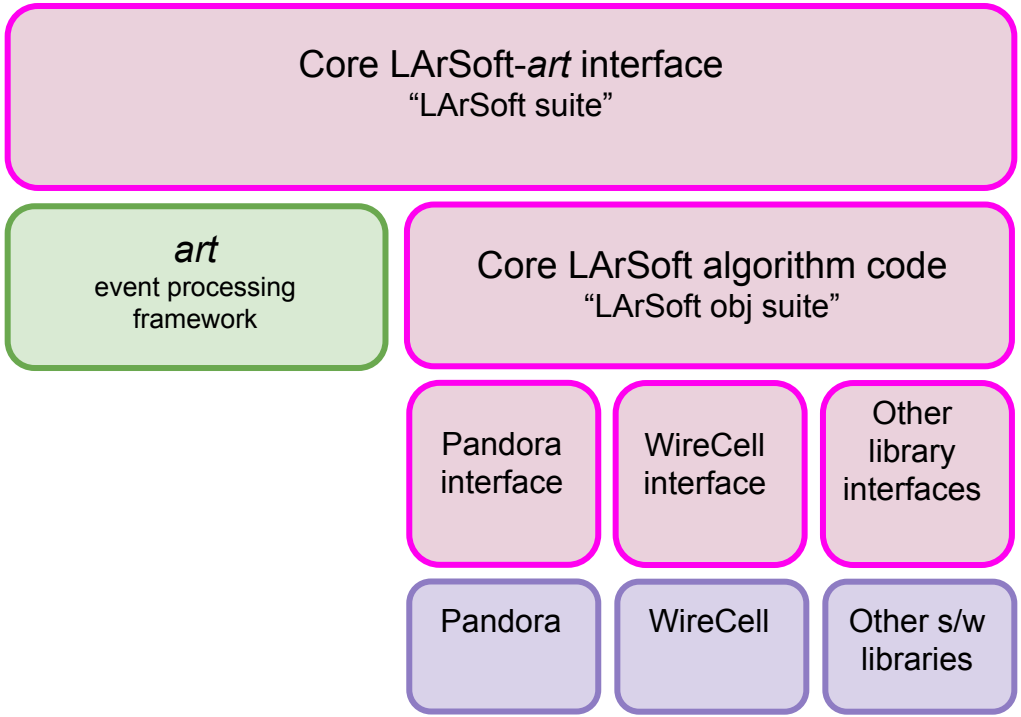
larpandoracontent
larwirecell
...

A special ups product reserved for larger configuration files (up to a few MB):

larsoft_data

Managed by LArSoft release managers

Structural components of LArSoft



Umbrella ups products

larsoft

larsoftobj

Allows single setup commands for groups of ups products

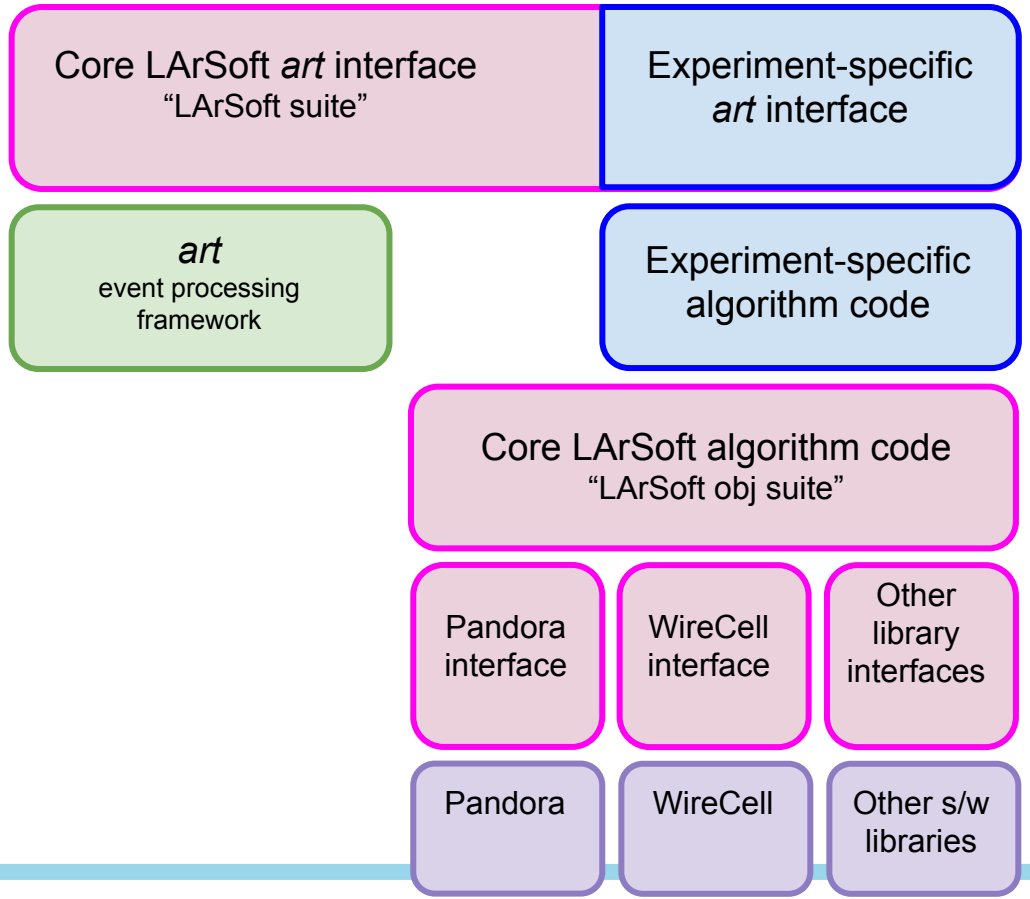
larsoft effectively depends on everything, so "setup larsoft …" sets up everything

Details for external libraries depends upon the library in question.

At present, for instance, most generatora and Geant4 are set up via "nutools" product

Fermilab

Experiment code



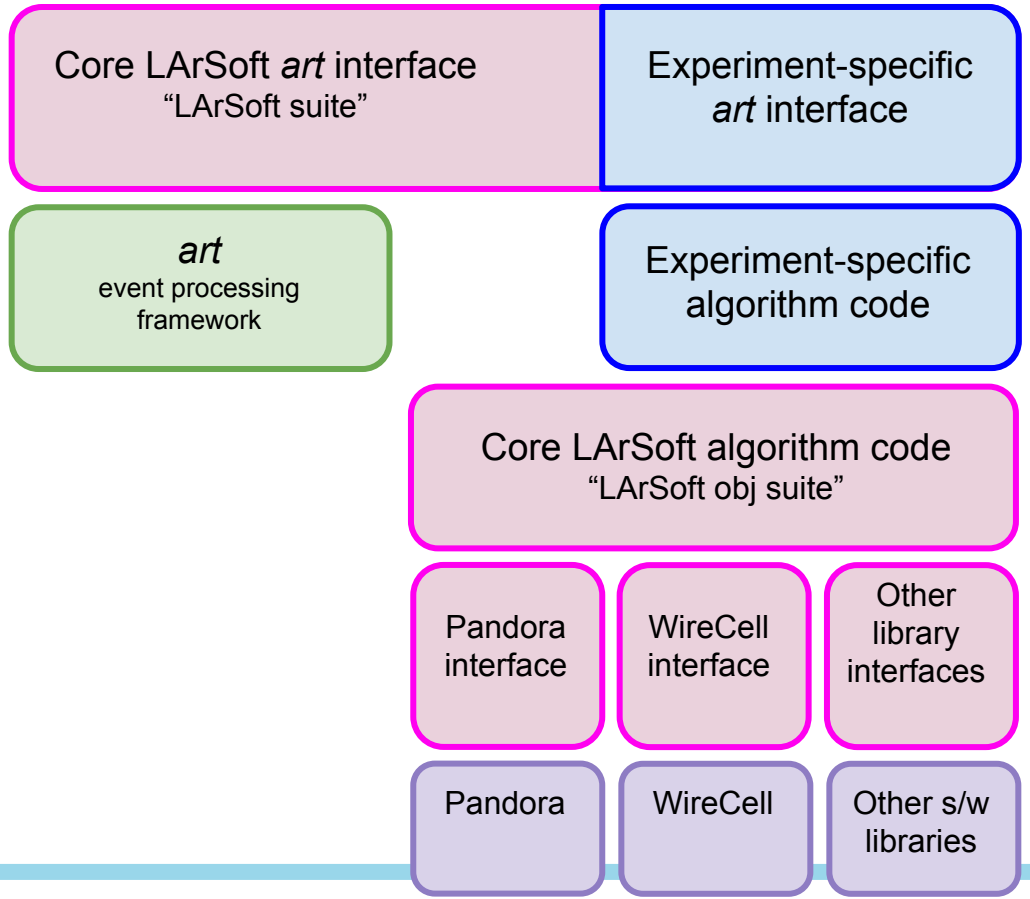
LArSoft is not stand-alone code.

Requires at least experiment / detector-specific configuration

Same basic design pertains to the experiment code

Nothing in core LArSoft code depends upon experiment code

Experiment code



Experiment repositories

MicroBooNE

uBCore
uBEvt
uBReco
...
uBObj

DUNE
dunetpc

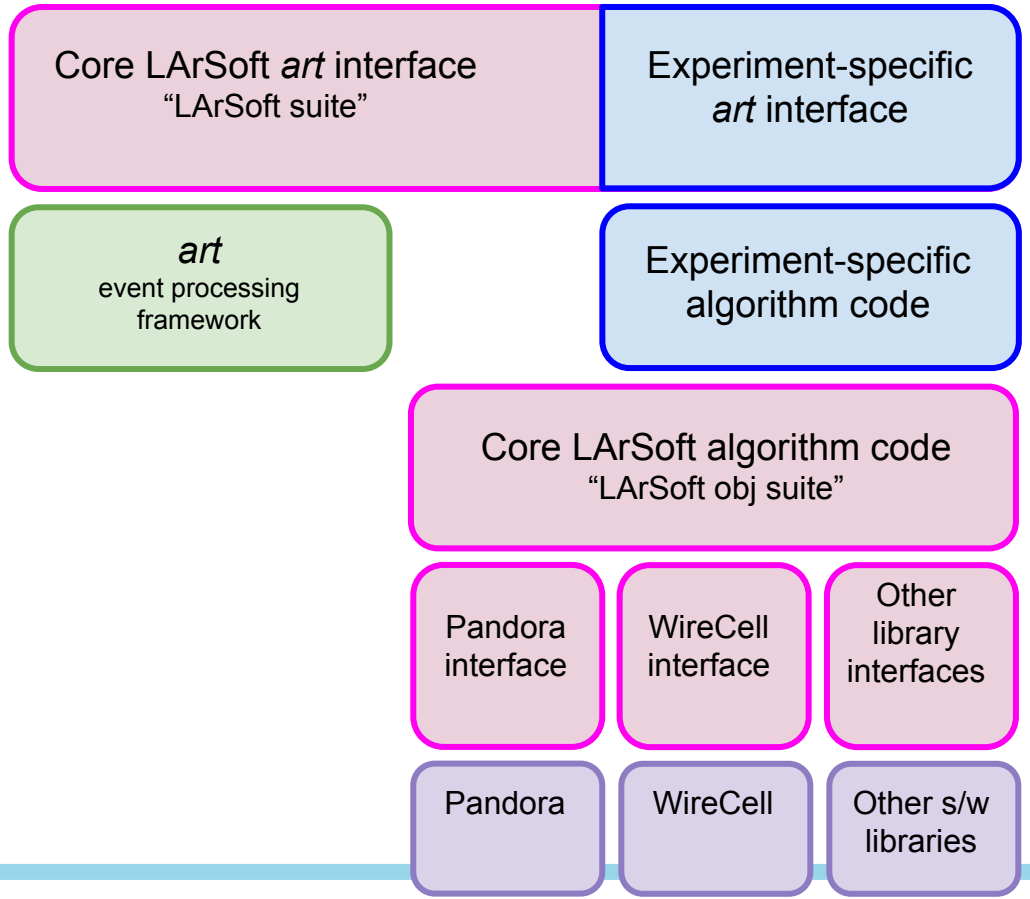
SBND
sbndcode

CARUS
icaruscode

Some experiment code may, strictly speaking, be *art* independent.

Most (all but MicroBooNE) lack required repository structure to build independently of art.

Experiment code



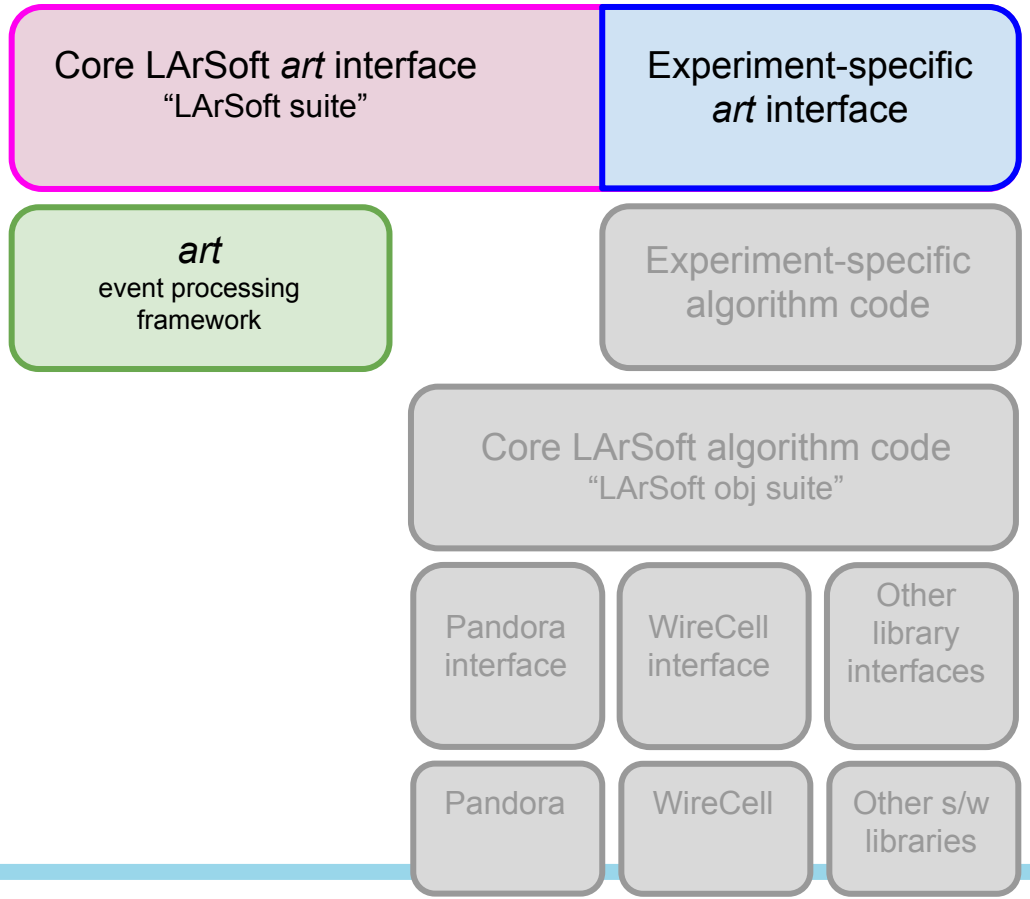
Experiment ups products

MicroBooNE:
- uboonecode (umbrella product)
- uBCore
- uBEvt
- uBReco
- ...
- uBObj

DUNE: dunetpc
SBND: sbndcode
CARUS: icaruscode

Except for MicroBooNE, umbrella products have the same name as the repositories

Conceptual design of LArSoft interfaces



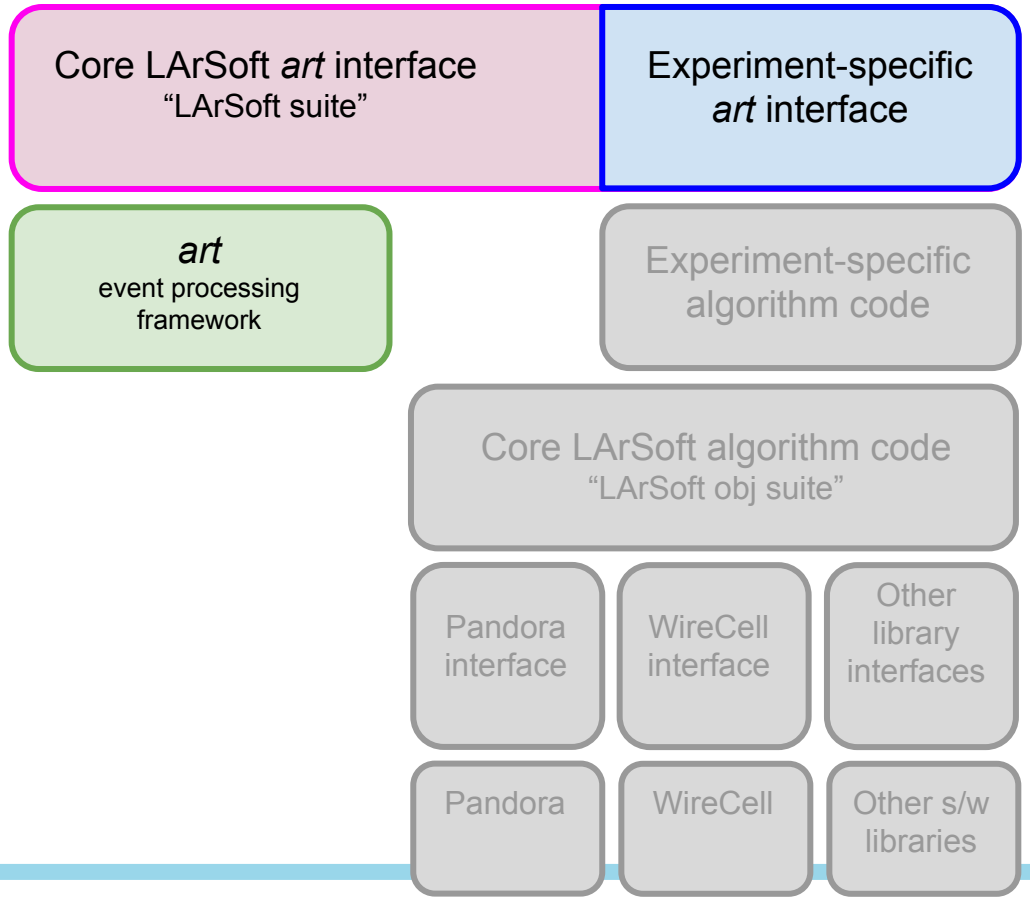
The "*art* interface" code

art module

art::Event
art::ServiceHandle<*service*>
art::Handle<*data product*>
art::make_tool<*tool type*>
…

The **event record**, **modules**, **services / service registry**, **handles** (all types), and **associated pre-processor directives**, etc., are all part of *art* interface

Conceptual design of LArSoft interfaces



The "*art* interface" code

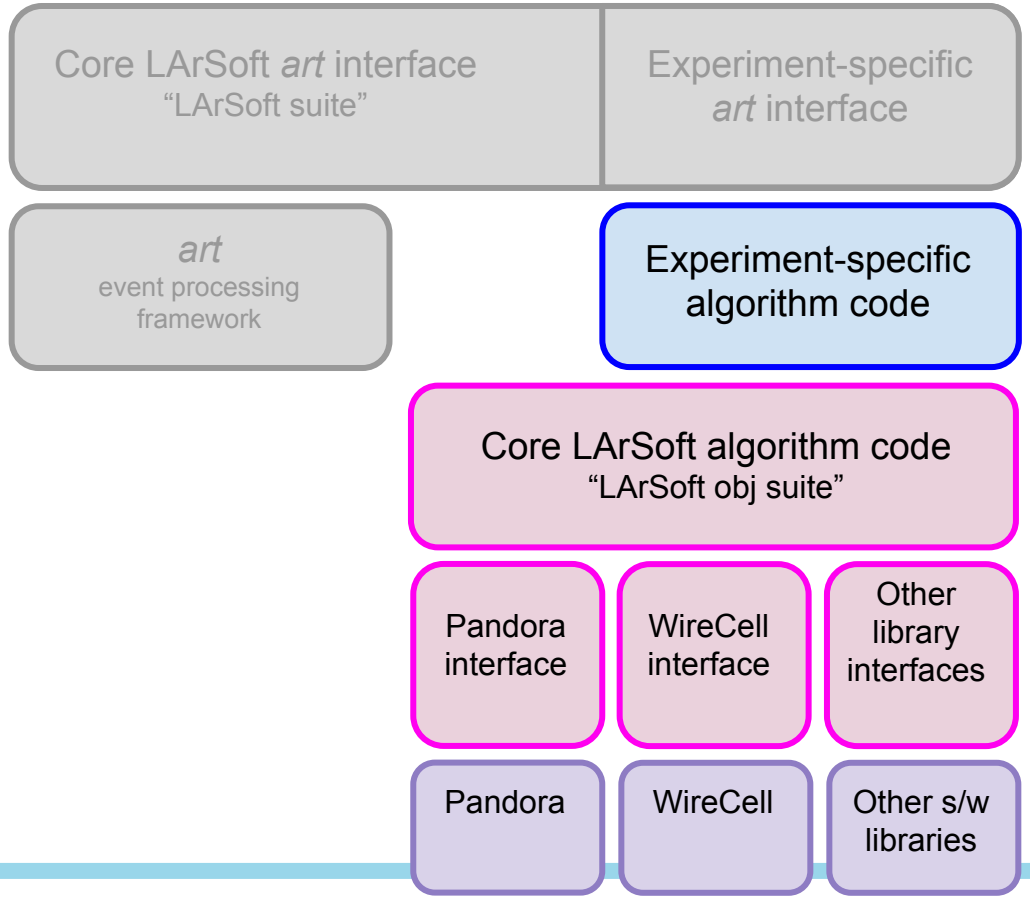
art module

art::Event
art::ServiceHandle<*service*>
art::Handle<*data product*>
art::make_tool<*tool type*>
…

The **event record**, **modules**, **services / service registry**, **handles** (all types), and **associated pre-processor directives**, etc., are all part of *art* interface

Modules should be used to get services, service-providers, parameter sets and data products, and to create tools, which should then be **passed** to algorithm code

Conceptual design of LArSoft interfaces

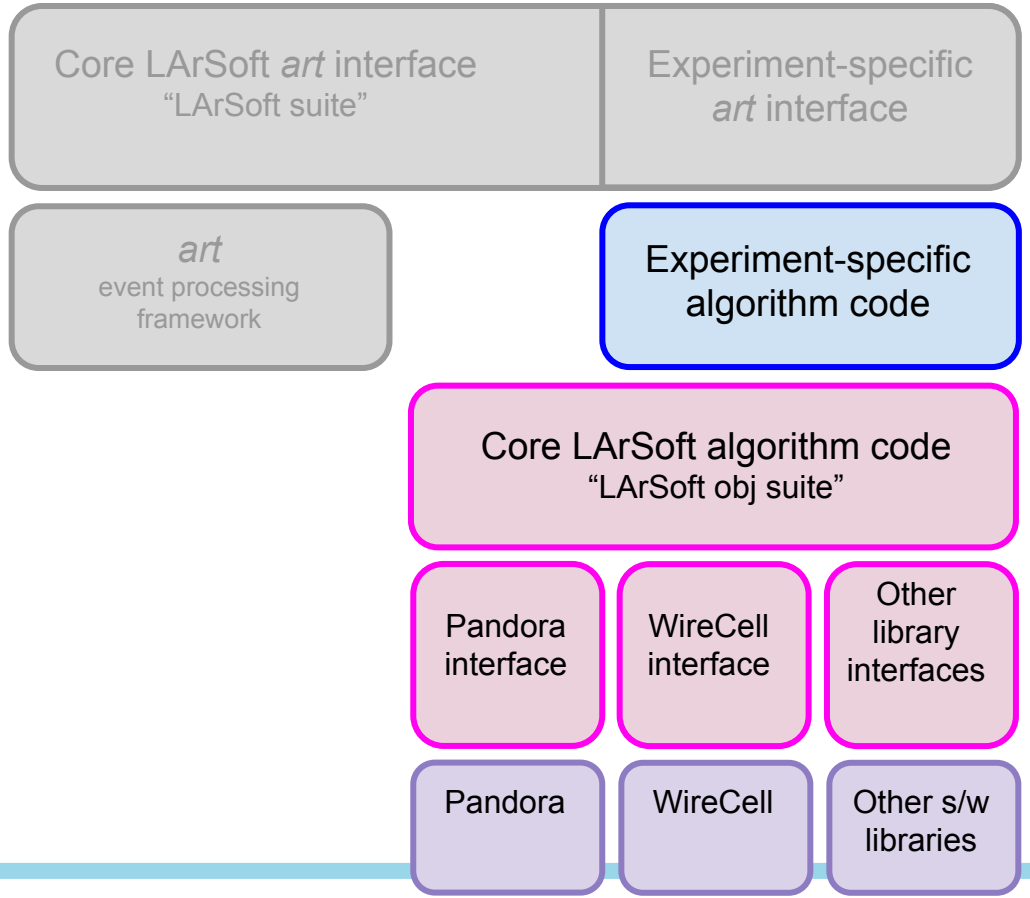


***art* independent code**

Algorithms, service-providers, data products, **should never depend on any** elements of *art* interface

Data and configuration should be **passed** into and out of algorithms, service-providers, other art-independent functions and classes.

Conceptual design of LArSoft interfaces



***art* independent code**

Algorithms, service-providers, data products, **should never depend on any** elements of *art* interface

Data and configuration should be **passed** into and out of algorithms, service-providers, other art-independent functions and classes.

Note: fhicl-cpp and message_facility are independent of *art*

- "*art* independent code" **may** include FHiCL parameter sets, message_facility calls, but need not

Why framework independence matters

Code that does not depend on *art* and all the attendant dependencies can:

- Be developed, built in a lightweight stand-alone environment
- Have easily constructed unit tests to check proper functioning
- Be used in alternate event processing / analysis frameworks and contexts
- Be used with *art* gallery
  - Provides lightweight access to art/ROOT files outside of art
  - Widely used both as analysis and development environment
  - The entire LArSoft Obj suite can be used in gallery

  More information at https://art.fnal.gov/gallery/

Design principles and coding practices

LArSoft design principles and coding practices

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

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

LArSoft design principles and coding practices

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

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

The foundation of the code sharing regime

Possible because the nature of LArTPCs allows for the use of many common interfaces, with differences expressed as differences in configuration

LArSoft design principles and coding practices

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

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

Already discussed...

LArSoft design principles and coding practices

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

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

Provides a means to hide detector-specific details behind common interfaces

Also allows layering of algorithms to build sophistication

LArSoft design principles and coding practices

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

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

Just good coding practice...

LArSoft design principles and coding practices

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

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

Ensures that code operates as intended
Simplifies code integration

LArSoft design principles and coding practices

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

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

So that other people understand what your code is supposed to do, and how to use it

So that you know what your code is supposed to do and how to use six months after you wrote it…

Use Doxygen markup in source code comments!!

LArSoft design principles and coding practices

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

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

New! (relatively)

Expect multi-threading to play an increasingly important role

- To help control scaling of memory usage
- To adapt to the evolving computing landscape

An entire session devoted to this topic tomorrow!

LArSoft design principles and coding practices

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

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

Ensures stability of the development environment

Allows rapid development cycles

Simplifies release management

LArSoft design principles and coding practices

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

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

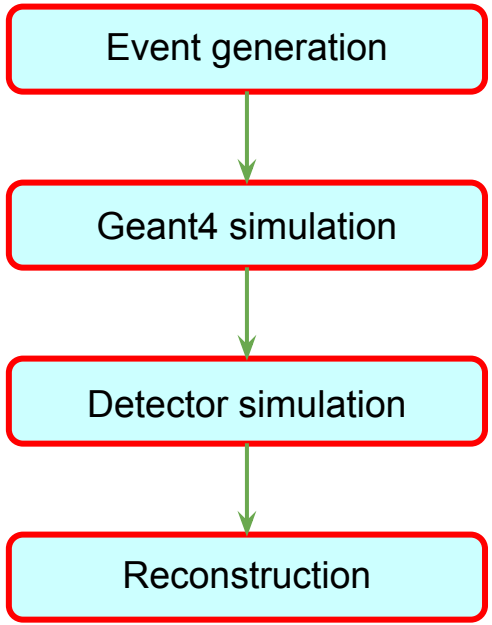
Additions and changes will be made as needed to adapt to changes in the computing landscape, or to better support code sharing

Contents of LArSoft

What does LArSoft do? And what is in it?

Provides tools to carry out simulation, reconstruction and analysis of LArTPC data

- Consider for instance, **an event generation, detector simulation, reconstruction workflow**



A general generation – simulation – reconstruction workflow

General simulation workflow

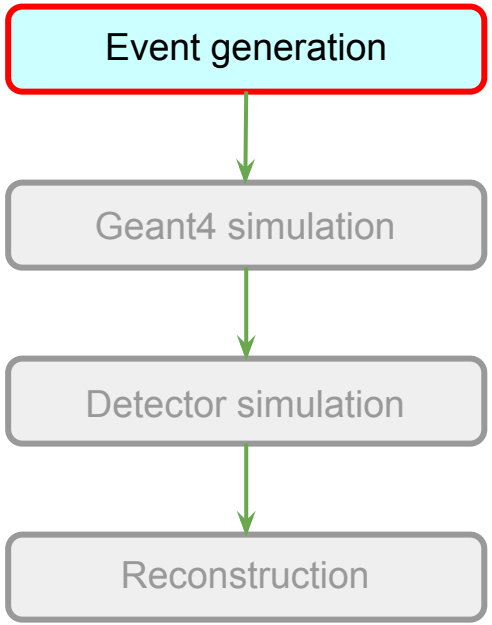
Event generators

- Genie: GENIEGen module
  - Direct interface to Genie neutrino event generator
  - larsim/larsim/EventGenerator/GENIE/
  - See genie.fcl in that directory
  - More documentation on the NuTools wiki page,
  - https://cdcvs.fnal.gov/redmine/projects/nutools/wiki

  Note: this is soon moving to NuGen product

- Single particles: SingleGen module
  - larsim/larsim/EventGenerator

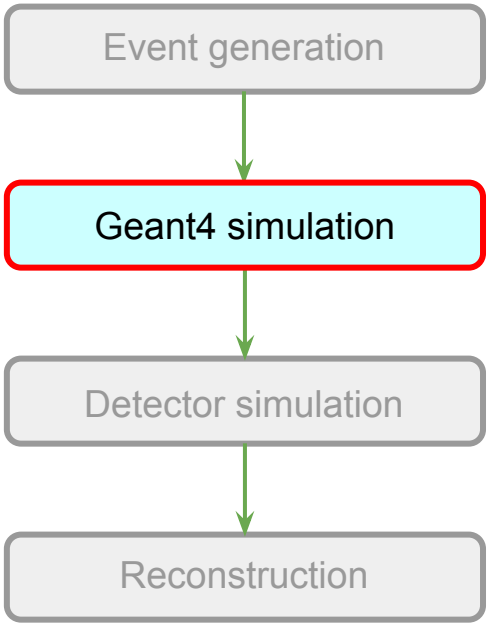
Others available via indirect common data exchange format



General simulation workflow

Geant4 detector simulation

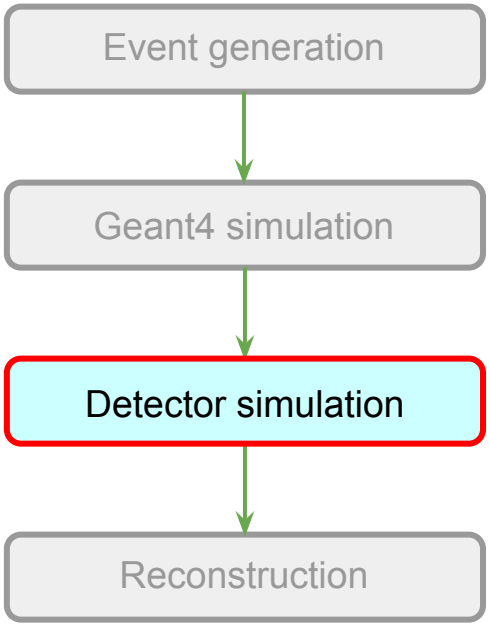
- Particle propagation simulation
- Models energy depositions in the detector
  - Rich, configurable models of particle interactions, optical properties (including detailed index of refraction, reflectivity, etc.)
  - Can perform optical simulation at single photon level
- The only simulation currently integrated with LArSoft



General simulation workflow

A separate workflow in itself

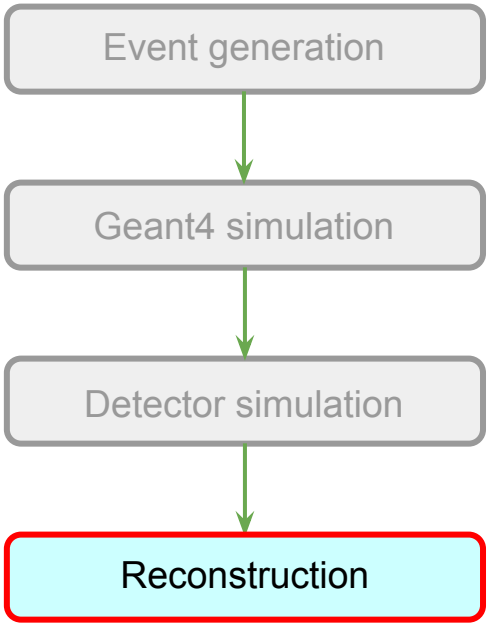
- Factorized into the following steps (implemented as separate modules / partly combined in WireCell)
  - Ionization and scintillation light modeling from energy depositions
  - Drift electron simulation
  - Anode region simulation, signal induction and noise modeling, digitization
  - Photon transport and detection model, including “S2 light” simulation for dual-phase detectors
  - Optical signal induction, noise modeling and digitization



General simulation workflow

Three major paradigms, each with its own variants, modules, workflows

- 2D clustering and view matching
  - Pandora multi-algorithm approach
  - TrajCluster 2D
- Image processing / deep learning techniques
  - Pixel-level track/shower tagging from 2D images (code not yet fully available)
  - Hit-based track/shower discrimination
- 3D imaging
  - Wire-cell: charge matching across wire planes in time slices
  - TrajCluster3D / Cluster3D: time / charge matching across wire planes using hits.



Code releases and distribution

LArSoft releases

A release contains all LArSoft code, ups products in a frozen state for distribution

Several types of releases

- Production
- Integration
- Test release
- Release candidate

LArSoft releases

A release contains all LArSoft code, ups products in a frozen state for distribution

Several types of releases

- Production
- Integration
- Test release
- Release candidate

- Any release designated as “production” by an experiment
  - Contents approved by the experiment
- Typically used for large-scale processing campaigns
- Created on demand
- Retained indefinitely on disk
- Numbering: vxx_yy_zz, e.g., v08_22_00

LArSoft releases

A release contains all LArSoft code, ups products in a frozen state for distribution

Several types of releases

- Production
- Integration
- Test release
- Release candidate

- Any release designated as “production” by an experiment
  - Contents approved by the experiment
- Typically used for large-scale processing campaigns
- Created on demand
- Retained indefinitely on disk
- Numbering: vxx_yy_zz, e.g., v08_22_00

Major version Minor version Patch version

Details on “LArSoft release naming and retention policy” wiki page

LArSoft releases

A release contains all LArSoft code, ups products in a frozen state for distribution

Several types of releases

- Production
- Integration
- Test release
- Release candidate

- Created weekly, or on demand for special purposes
- Provides a stable code base for development that is close to the head of repositories
- Contents approved at LArSoft Coordination Meetings
  - Head of develop + additional branches approved at LCM or via email
- May be removed without notice after about a month (though has never happened…)
- Numbering: vxx_yy_zz (same sequence as production releases)

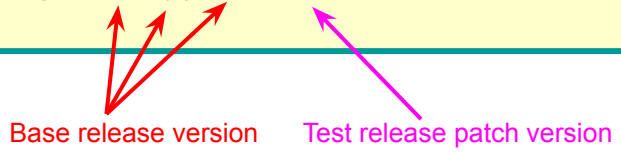
LArSoft releases

A release contains all LArSoft code, ups products in a frozen state for distribution

Several types of releases

- Production
- Integration
- Test release
- Release candidate

- Created to allow experiments to test a new product or new produce version (e.g., Genie, Geant4, art (sometimes)) on top of a known release
- Identical to some base integration or production release except for that product version + any adaptations needed for integration
- Retained on disk until testing is completed
- Numbering: vxx_yy_zz_kk



LArSoft releases

A release contains all LArSoft code, ups products in a frozen state for distribution

Several types of releases

- Production
- Integration
- Test release
- Release candidate

- Created to allow experiments to test a new major version of LArSoft.
  - Sometimes (rarely), a major change to a critical underlying product will trigger this condition
- Retained on disk until testing is completed
- Numbering: vxx_yy_zz_rcn

Target release version

Release candidate version

LArSoft releases

A release contains all LArSoft code, ups products in a frozen state for distribution

Several types of releases

- Production
- Integration
- Test release
- Release candidate

The list of all LArSoft releases, the purpose, significant changes listed on the "LArSoft release list" wiki page (https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki/LArSoft_release_list)

Each entry has a link to release notes for that release

LArSoft code distribution

LArSoft releases are distributed via two mechanisms

- cvmfs
  - CERN virtual file system
  - Appears as locally mounted disk area
    - /cvmfs/larsoft.opensciencegrid.org/products/larsoft
- Binary and source tarballs
  - Downloadable from scisoft.fnal.gov
    - https://scisoft.fnal.gov/
  - Instructions for installing, building (when needed) are linked from the release notes

LArSoft code distribution

Every release is distributed in several build variants

- Operating system
- Combination of compiler version + other build flags
- Optimized versus debug versions

Distinguished during setup by

- The current operating system (or as specified in the setup command)
- Qualifiers specified in the setup command

More on this later in Saba Sehrish's talk

Supported platforms

- “Supported platforms”
  - Builds actively supported
  - Code runs and works as intended (as reported by CI system)
  - Source and binary distributions available on cvmfs and scisoft.fnal.gov

    Currently includes:

    - SLF6 and SLF7

Supported platforms

- "Known to work"
  - We know of someone (usually us!) who has succeeded in building and running
  - LArSoft does not officially support builds or distribution

A special "best effort" category exists in this space

- Includes operating systems considered as important to LArSoft developer community
- Support on-demand builds, or regular builds after release of "supported platform" distributions
- May or may not include CI system support

Currently includes:

- MacOS: regular builds (usually), CI system support
- Ubuntu LTS 16, 18: on-demand, no CI system support

End-user / developer resources

Documentation

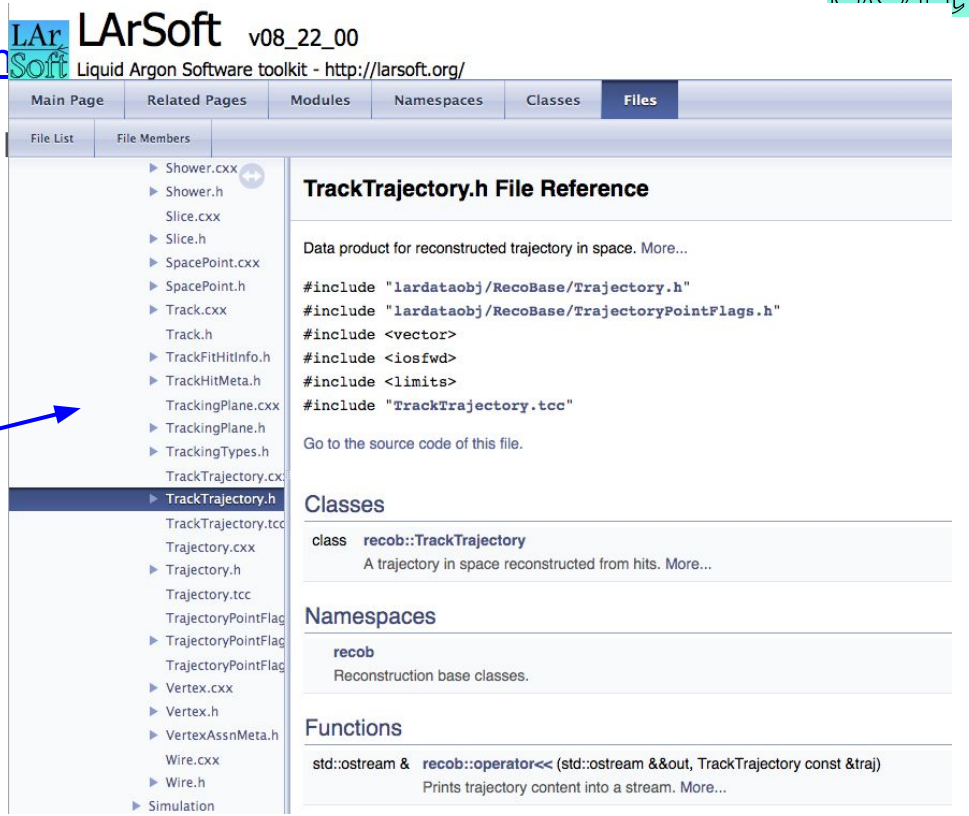
Doxygen: http://nusoft.fnal.gov/larsoft/doxsvn/html/

- Auto-generated documentation from markup embedded in source comments

Documentation

Doxygen: http://nusoft.fnal.gov/larsoft/doxsvn

- Auto-generated documentation from
  embedded in source comments

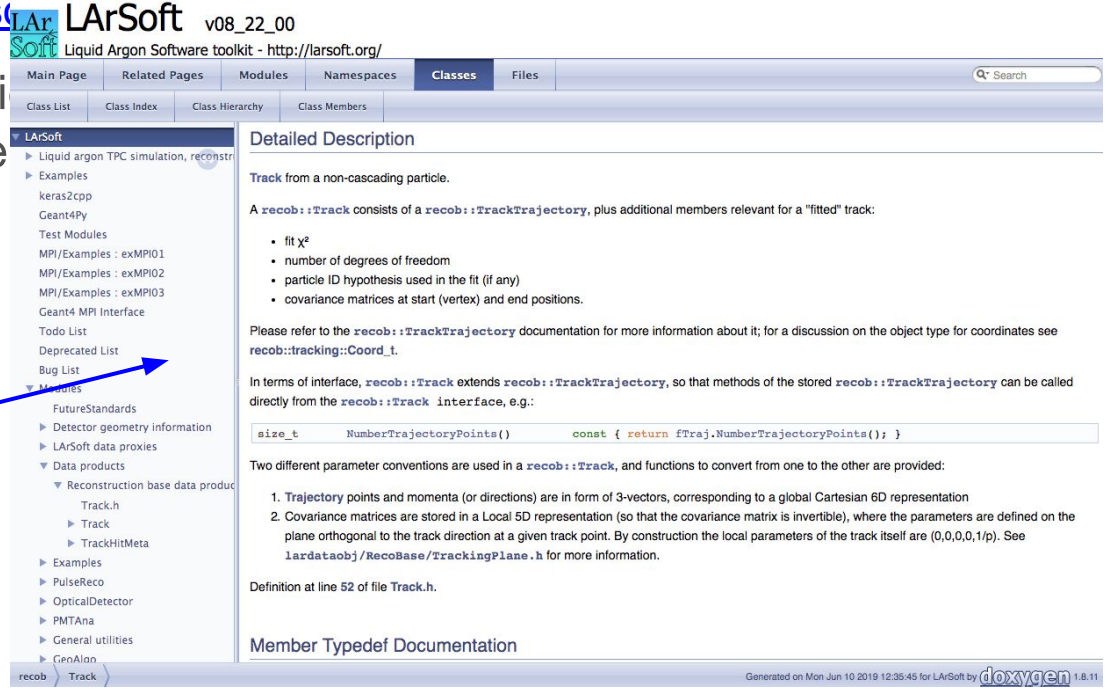


"File" view

Documentation

Doxygen: http://nusoft.fnal.gov/larso

- Auto-generated documentati
  embedded in source comme

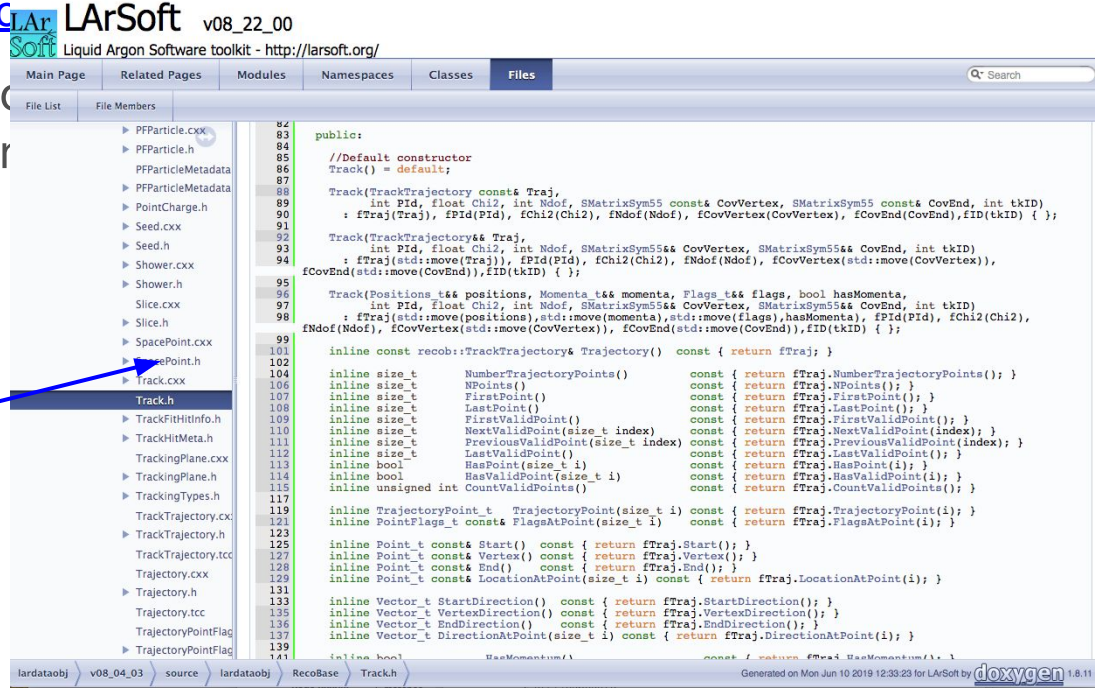


"Class" view

Documentation

Doxygen: http://nusoft.fnal.gov/larsc

- Auto-generated documentatic
  embedded in source commer



"Source" view

Documentation

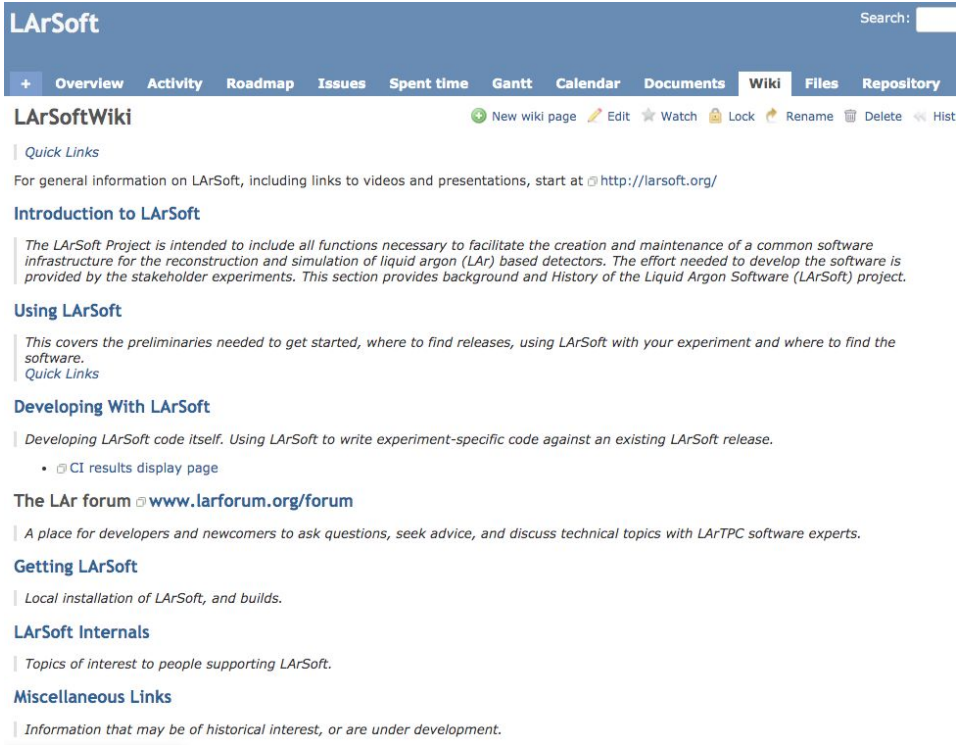
Doxygen: http://nusoft.fnal.gov/larsoft/doxsvn/html/

- Auto-generated documentation from markup embedded in source comments
- Pros:
  - A significant fraction of code includes such comments
  - Should always be up to date with the code you are viewing
- Cons:
  - Provides no high-level view or context
  - Quality varies greatly due to absence of enforceable standards or conventions

LArSoft Redmine site

https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki

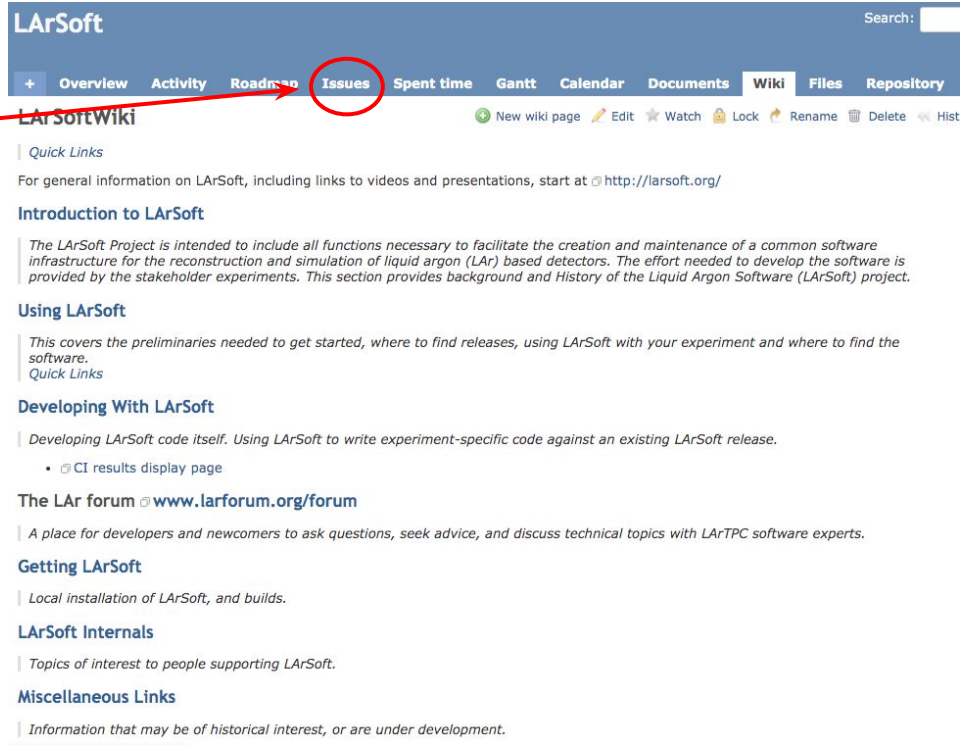
- Technical reference
- Issue tracker
- Repository browser



LArSoft Redmine site

https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki

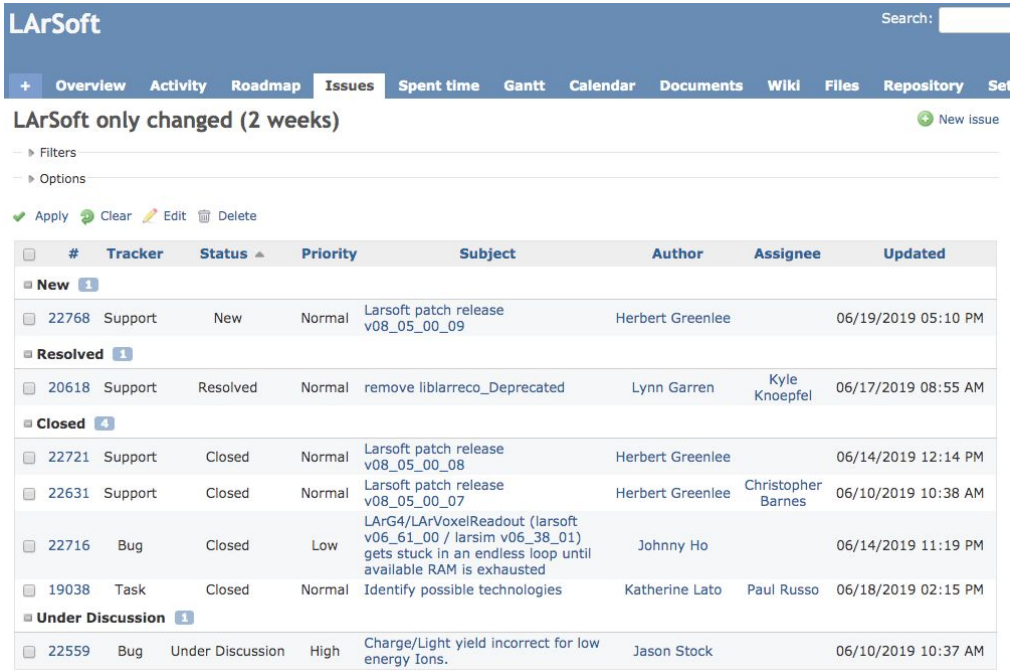
- Technical reference
- Issue tracker
- Repository browser



LArSoft Redmine site

https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki

- Technical reference
- Issue tracker
- Repository browser

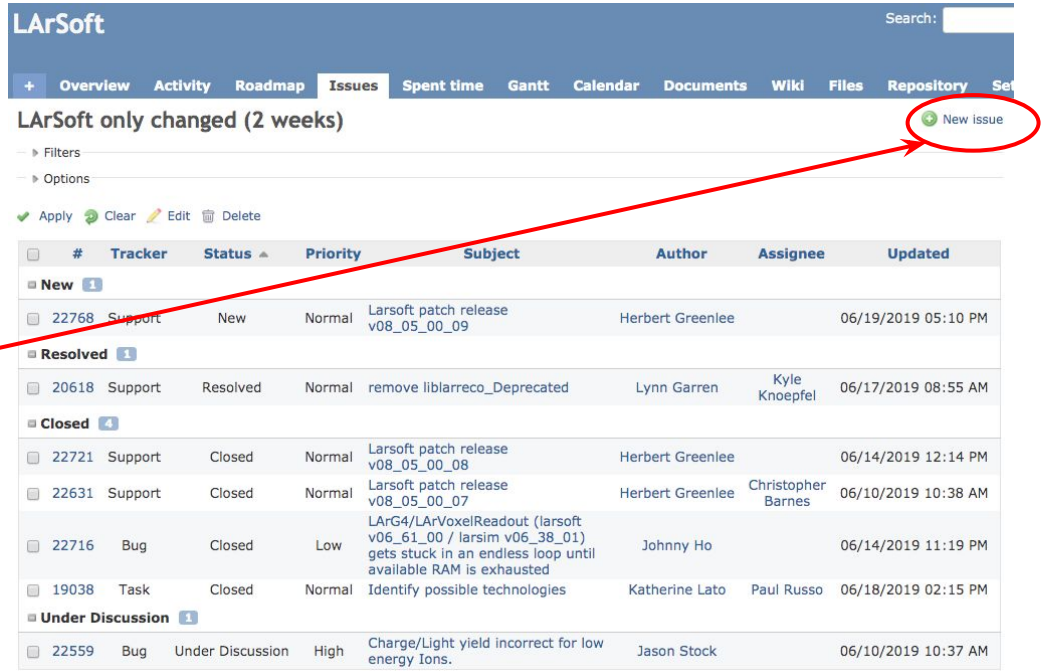


LArSoft Redmine site

https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki

- Technical reference
- Issue tracker
- Repository browser

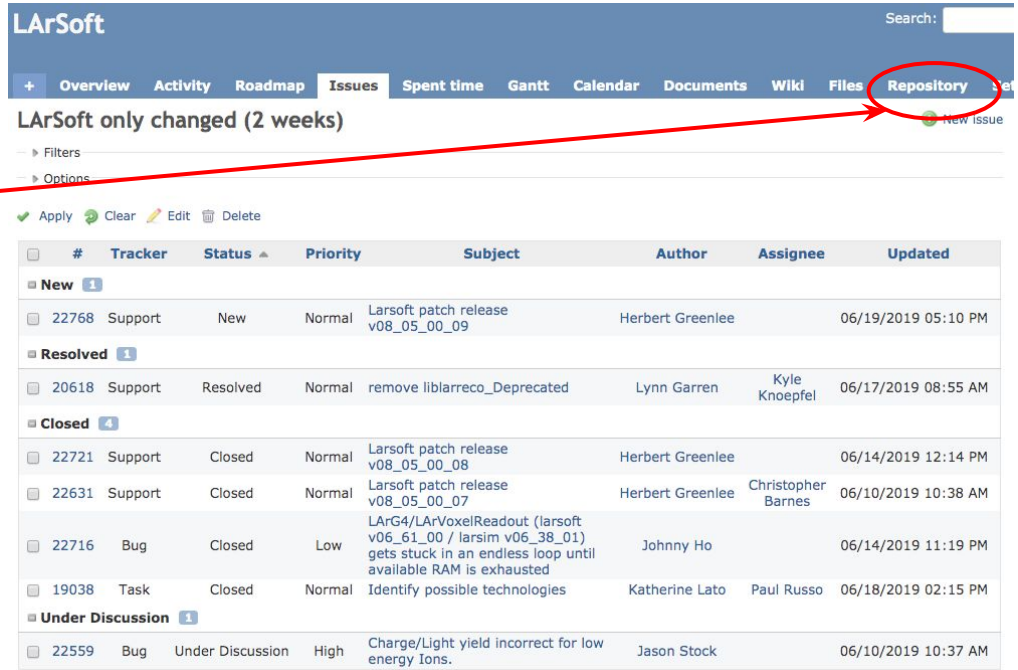
Report problems
Make requests
Ask questions
Make suggestions



LArSoft Redmine site

https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki

- Technical reference
- Issue tracker
- Repository browser

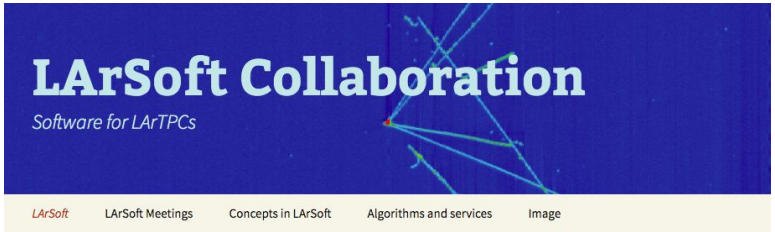


#	Tracker	Status	Priority	Subject	Author	Assignee	Updated
New 1							
22768	Support	New	Normal	Larsoft patch release v08_05_00_09	Herbert Greenlee		06/19/2019 05:10 PM
Resolved 1							
20618	Support	Resolved	Normal	remove liblarreco_Deprecated	Lynn Garren	Kyle Knoepfel	06/17/2019 08:55 AM
Closed 4							
22721	Support	Closed	Normal	Larsoft patch release v08_05_00_08	Herbert Greenlee		06/14/2019 12:14 PM
22631	Support	Closed	Normal	Larsoft patch release v08_05_00_07	Herbert Greenlee	Christopher Barnes	06/10/2019 10:38 AM
22716	Bug	Closed	Low	LArG4/LArVoxelReadout (larsoft v06_61_00 / larsim v06_38_01) gets stuck in an endless loop until available RAM is exhausted	Johnny Ho		06/14/2019 11:19 PM
19038	Task	Closed	Normal	Identify possible technologies	Katherine Lato	Paul Russo	06/18/2019 02:15 PM
Under Discussion 1							
22559	Bug	Under Discussion	High	Charge/Light yield incorrect for low energy Ions.	Jason Stock		06/10/2019 10:37 AM

LArSoft.org

https://larsoft.org/

- Organizational information about the collaboration
  - Governance structure
  - Meeting notes
- High-level documentation
- Links to training information / sessions



LArSoft

The Liquid Argon Software (LArSoft) Collaboration develops and supports a shared base of physics software across Liquid Argon (LAr) Time Projection Chamber (TPC) experiments.

A video introduction to LArSoft by Ruth Pordes and Erica Snider is available here. The pdf of the paper is available here.

The LArSoft Collaboration is driven by the needs of the participating experiments as represented by the steering group, which consists of spokespeople of the experiments as well as representatives from Fermilab's Scientific Computing and Neutrino Divisions.

More information about LArSoft is at:

- LArSoft Training – links to videos and presentations about LArSoft
- LArSoft Article – short introduction for general public
- LArSoft conference paper by Erica Snider and Gianluca Petrillo

LArSoft CI system

Documentation: https://cdcvs.fnal.gov/redmine/projects/lar_ci/wiki

Monitoring app: http://lar-ci-history.fnal.gov/LarCI/app

- Drives both rapid turn-around CI testing and more comprehensive validation workflows and testing
- Users can run tests locally prior to committing code, or launch jobs to look at specified combinations of branches

LArSoft CI system

Documentation: https://cdcvs.fnal.gov/redmine/projects/lar_ci/wiki

Monitoring app: http://lar-ci-history.fnal.gov/LarCI/app

lar_ci wiki page

Fermilab

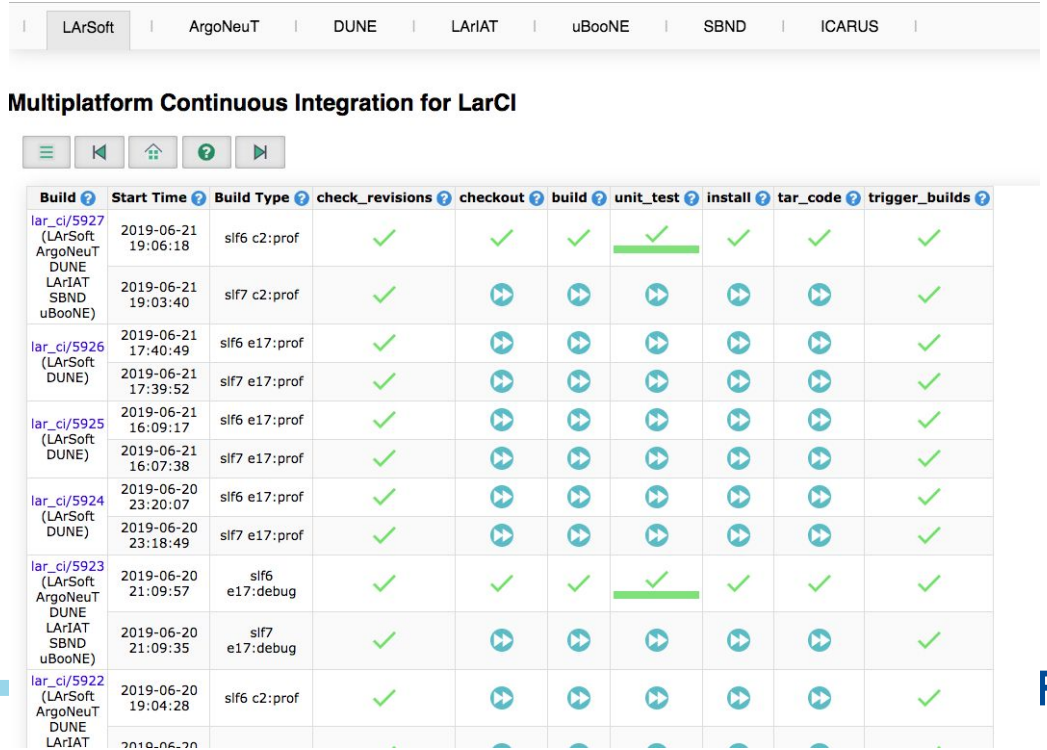
LArSoft CI system

Documentation: https://cdcvs.fnal.gov/redmine/projects/lar_ci/wiki

Monitoring app: http://lar-ci-history.fnal.gov/LarCI/app

Monitoring app

Drill-down by experiment to see test results at increasingly fine detail



Build	Start Time	Build Type	check_revisions	checkout	build	unit_test	install	tar_code	trigger_builds
lar_ci/5927 (LArSoft ArgoNeuT DUNE LArIAT SBND uBooNE)	2019-06-21 19:06:18	slf6 c2:prof	✓	✓	✓	✓	✓	✓	✓
	2019-06-21 19:03:40	slf7 c2:prof	✓	⏩	⏩	⏩	⏩	⏩	✓
lar_ci/5926 (LArSoft DUNE)	2019-06-21 17:40:49	slf6 e17:prof	✓	⏩	⏩	⏩	⏩	⏩	✓
	2019-06-21 17:39:52	slf7 e17:prof	✓	⏩	⏩	⏩	⏩	⏩	✓
lar_ci/5925 (LArSoft DUNE)	2019-06-21 16:09:17	slf6 e17:prof	✓	⏩	⏩	⏩	⏩	⏩	✓
	2019-06-21 16:07:38	slf7 e17:prof	✓	⏩	⏩	⏩	⏩	⏩	✓
lar_ci/5924 (LArSoft DUNE)	2019-06-20 23:20:07	slf6 e17:prof	✓	⏩	⏩	⏩	⏩	⏩	✓
	2019-06-20 23:18:49	slf7 e17:prof	✓	⏩	⏩	⏩	⏩	⏩	✓
lar_ci/5923 (LArSoft ArgoNeuT DUNE LArIAT SBND uBooNE)	2019-06-20 21:09:57	slf6 e17:debug	✓	✓	✓	✓	✓	✓	✓
	2019-06-20 21:09:35	slf7 e17:debug	✓	⏩	⏩	⏩	⏩	⏩	✓
lar_ci/5922 (LArSoft ArgoNeuT DUNE LArIAT	2019-06-20 19:04:28	slf6 c2:prof	✓	⏩	⏩	⏩	⏩	⏩	✓

Fermilab

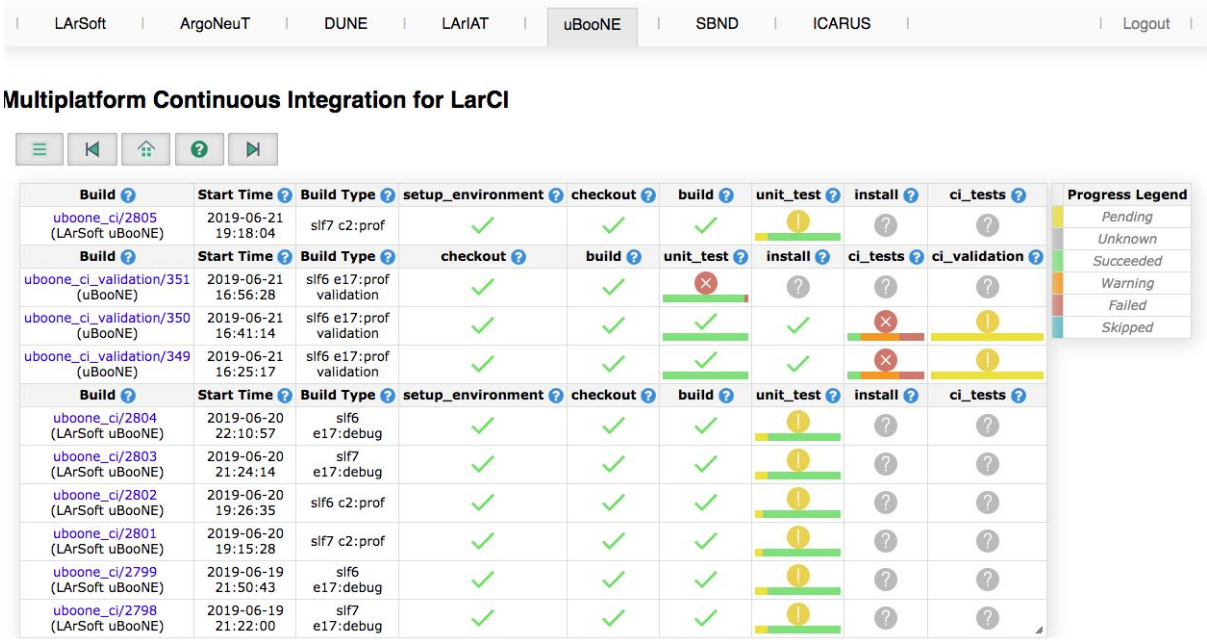
LArSoft CI system

Documentation: https://cdcvs.fnal.gov/redmine/projects/lar_ci/wiki

Monitoring app: http://lar-ci-history.fnal.gov/LarCI/app

Monitoring app

Drill-down by experiment to see test results at increasingly fine detail



Build	Start Time	Build Type	setup_environment	checkout	build	unit_test	install	ci_tests
uboone_ci/2805 (LArSoft uBooNE)	2019-06-21 19:18:04	slf7 c2:prof	✓	✓	✓	!	?	?

Build	Start Time	Build Type	checkout	build	unit_test	install	ci_tests	ci_validation
uboone_ci_validation/351 (uBooNE)	2019-06-21 16:56:28	slf6 e17:prof validation	✓	✓	✗	?	?	?
uboone_ci_validation/350 (uBooNE)	2019-06-21 16:41:14	slf6 e17:prof validation	✓	✓	✓	✓	✗	!
uboone_ci_validation/349 (uBooNE)	2019-06-21 16:25:17	slf6 e17:prof validation	✓	✓	✓	✓	✗	!

Build	Start Time	Build Type	setup_environment	checkout	build	unit_test	install	ci_tests
uboone_ci/2804 (LArSoft uBooNE)	2019-06-20 22:10:57	slf6 e17:debug	✓	✓	✓	!	?	?
uboone_ci/2803 (LArSoft uBooNE)	2019-06-20 21:24:14	slf7 e17:debug	✓	✓	✓	!	?	?
uboone_ci/2802 (LArSoft uBooNE)	2019-06-20 19:26:35	slf6 c2:prof	✓	✓	✓	!	?	?
uboone_ci/2801 (LArSoft uBooNE)	2019-06-20 19:15:28	slf7 c2:prof	✓	✓	✓	!	?	?
uboone_ci/2799 (LArSoft uBooNE)	2019-06-19 21:50:43	slf6 e17:debug	✓	✓	✓	!	?	?
uboone_ci/2798 (LArSoft uBooNE)	2019-06-19 21:22:00	slf7 e17:debug	✓	✓	✓	!	?	?

SciSoft support team

Provides support for LArSoft (among many other software projects, e.g., *art*) via:

- User support
- Technical expertise, problem solving
- Software solutions
- Architecture maintenance and development
- LArSoft work plan execution
- Release management
- Project management

SciSoft support team

Team members:

- Developers / experts / user support
  - Vito di Benedetto
  - Giuseppe Cerati
  - Patrick Gartung
  - Chris Green
  - Robert Hatcher
  - Marc Paterno
  - Paul Russo
  - Saba Sehrish
  - Mike Wang
- Project manager
  - Katherine Lato
- Leaders
  - Kyle Knoepfel
  - Erica Snider
- LArSoft project technical lead
  - Erica Snider

Email to scisoft-team@fnal.gov

The end