BLSS architecture

ARES

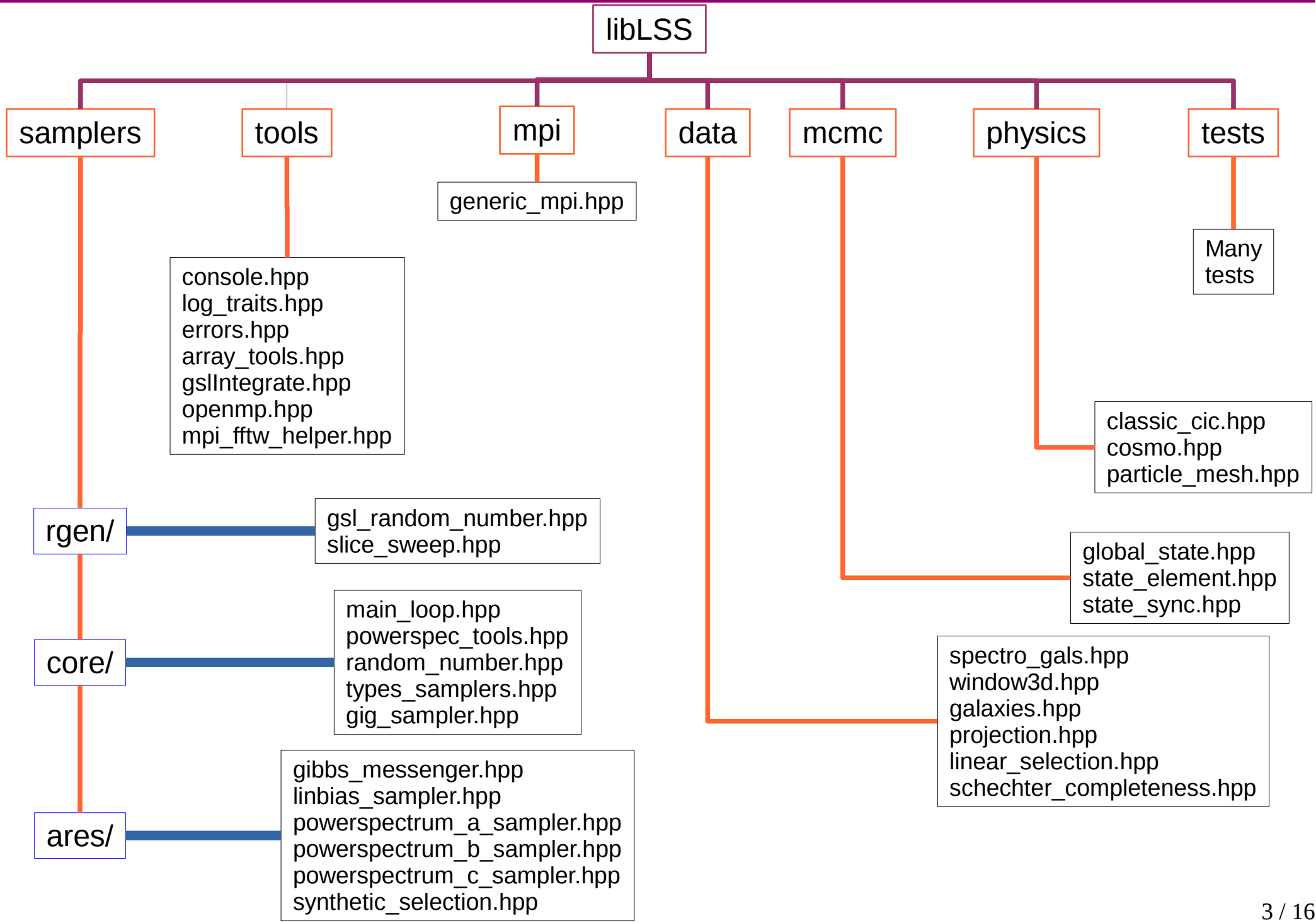
HADES

BORG

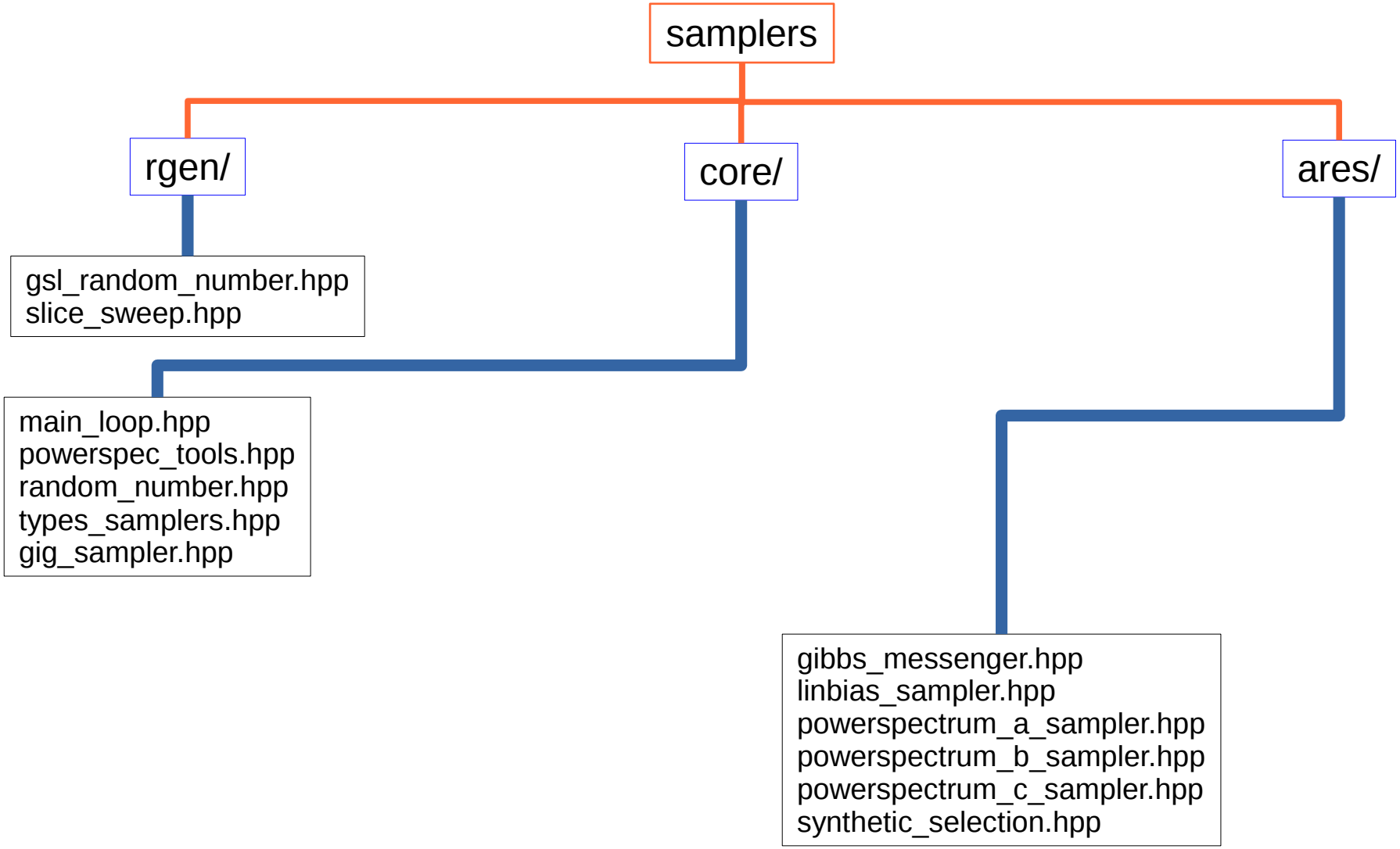
Structure

- ARES3 is actually the instrastructure package. Maybe will be renamed later
- On top of it there is an "extra/" subdirectory that contains extra modules (like foreground handling, HADES, BORG)
- Here is the list of top directories:
  - src/ → source of the main (orchestration) part of ARES3
  - libLSS/ → source code of the libLSS library, most of the interesting action happens here
  - external/ → external dependencies management
  - extra/ → additional modules (e.g. BORG, ARES, …)
  - scripts/ → some useful analysis scripts written in python
  - doc/ → a first attempt at documenting the code, needs care
  - examples/ → a directory with some example data to feed the code

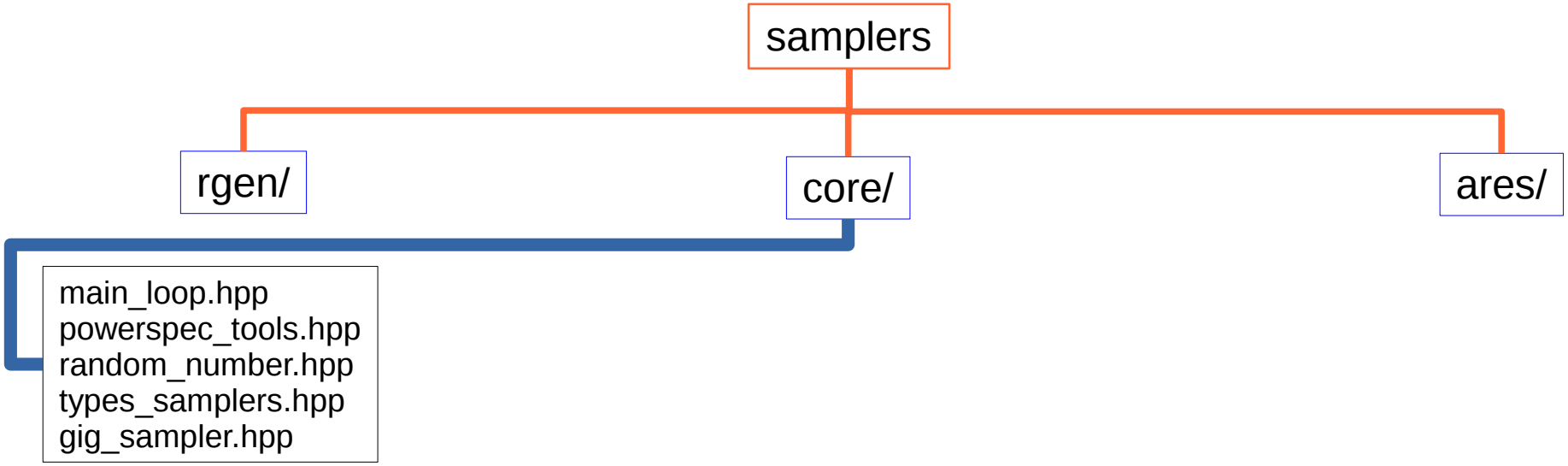
The libLSS tree



The libLSS/samplers subtree



The libLSS/samplers subtree

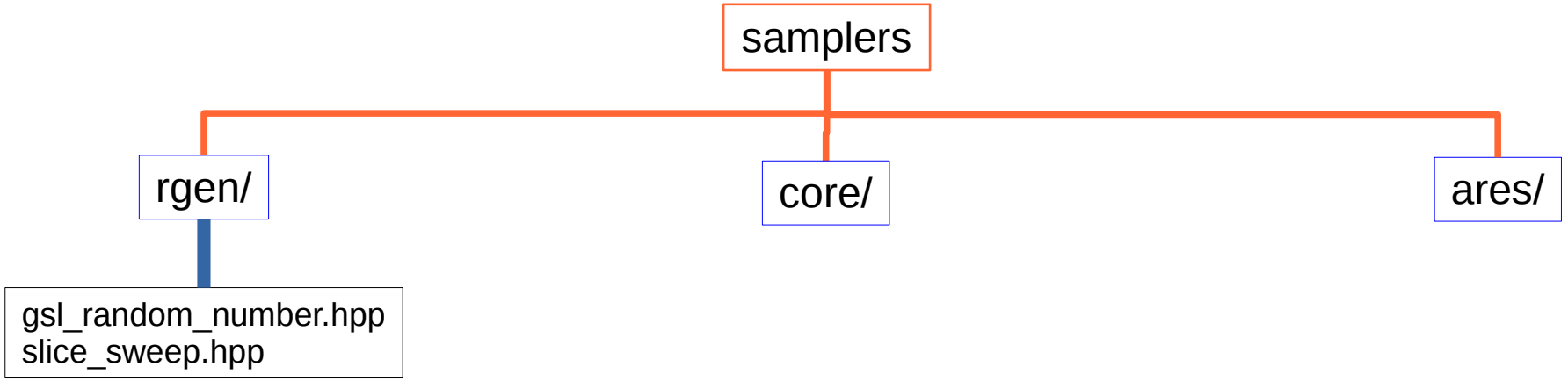


main_loop.hpp → describe a Markov Chain sampling main loop. The main user is src/ares_bundle.hpp. Class implementing some abstraction to chain samplers and optionally loop over them.
powerspec_tools.hpp → base classes for samplers handling the powerspectrum. e.g. al the powerspec_*_sampler in samplers/ares/.
random_number.hpp → base class for random number generators used everywhere in the code
types_samplers.hpp → define most common types from abstract types, like 3D arrays, memory allocators, …
gig_sampler.hpp → Generalized Inverse Gamma distribution efficient sampler

$$f(x) \propto x^{p-1} \mathrm{e}^{-(ax+b/x)/2}$$

(inverse Gamma distribution has a=0)

The libLSS/samplers subtree



Random number generator based on GSL,
support OpenMP and MPI through automatic reseeding of the Mersenne twister algorithm

1d slice sampler algorithm

```
template<typename Random, typename LogLikelihood>
double slice_sweep(
        MPI_Communication *comm,
        Random& rng,
        LogLikelihood lh,
        double a0,
        double step,
        int ROOT = 0)
```

MPI communicator for parallel sampling

Generic random number generator

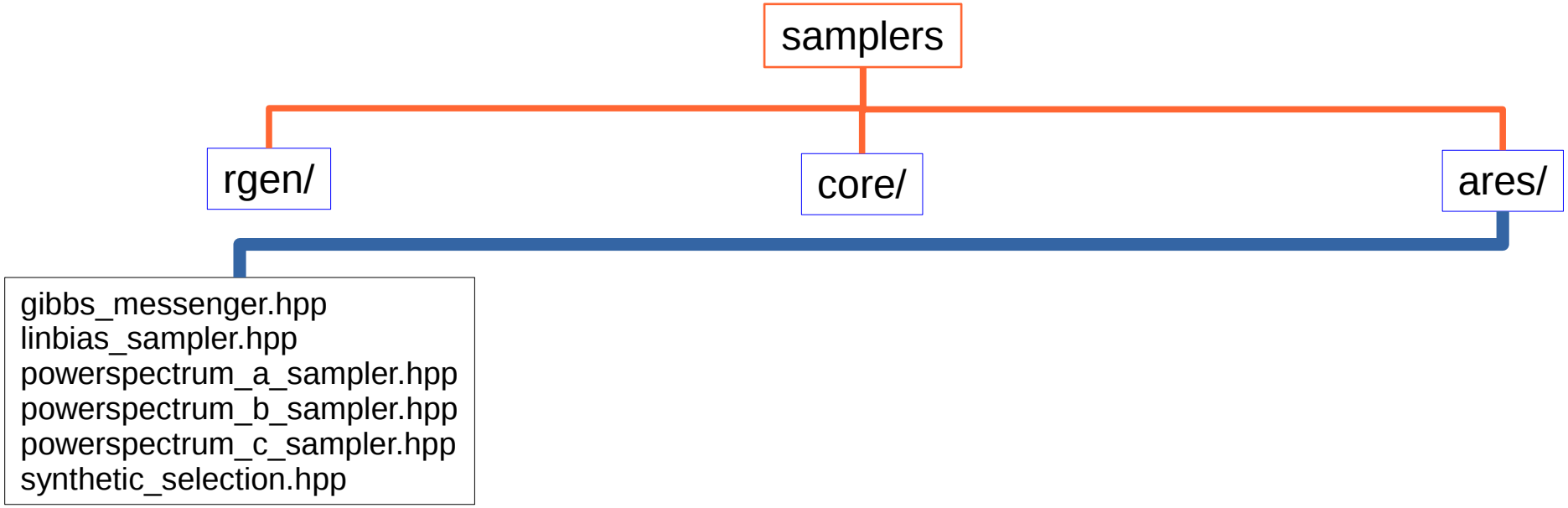
Functor of returning the value of the log likelihood

Initial guess position (must be previous step in the chain)

Initial step to use to explore the log likelihood (must be constant)

Set the master in communication (optional)

The libLSS/samplers subtree



gibbs_messenger.hpp → implements the Gibbs Messenger of Jasche & Lavaux 2015. It corresponds actually to two samplers that must be linked together in the main loop (one for messenger and for signal)

linbias_sampler.hpp → implements the mean galaxy number / voxel and bias sampler

powerspectrum_*_sampler.hpp → different variants of powerspectrum samplers.

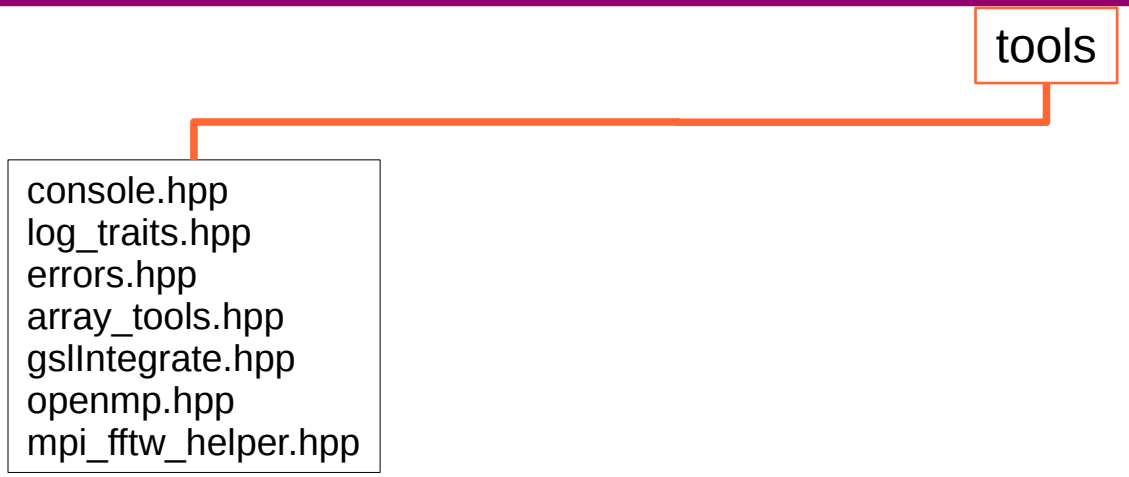
"a" → standard inverse gamma distribution, quick to evaluate but long correlations

"b" → Gaussian approximation with full small scale noise description, good to reduce correlations of the chain on small scales

"c" → slice sampling of the full log-likelihood, at fixed white phases, much better mixing of modes correlated due to the mask. **BUT** may be very slow.

synthetic_selection.hpp → not a sampler per se, useful when other parameters describe the selection function changes and thus the mask/selection must be rebuilt.

The libLSS/tools subtree



console.hpp → Most important element: console and log handling to make the output manageable.
log_traits.hpp → Policy for logs are defined here: e.g. LOG_VERBOSE, LOG_INFO, ...
errors.hpp → Pre-defined classical errors that will cause the run to stop are defined here. They can be easily triggered using "error_helper<SomeError>("My message");"
array_tools.hpp → Many array instrumentation functions (reordering, copying, scaling), parallelized if possible.
gslIntegrate.hpp → Implements a small useful wrapper to compute integrals with GSL.
openmp.hpp → A small abstraction layer to make OpenMP functions always available (remove ifdefs)
mpi_fftw_helper.hpp → if you ever wants to run FFTWs in ARES, you should now use this class. It will make your life less miserable. It is designed to handle 3d mesh grid, parallelized optionally over MPI. Supersampling is implemented there.
fused_assign.hpp → infrastructure for handling operations on arrays more generically, with automatic parallelization. Handle with care. Example in libLSS/tests/test_fused_array.cpp

The libLSS/mpi subtree

Abstract layer to handle MPI and non-MPI builds.
C++ wrapper around C MPI functions to make error handling more automatic, automatic type inference, … to reduce mistakes.

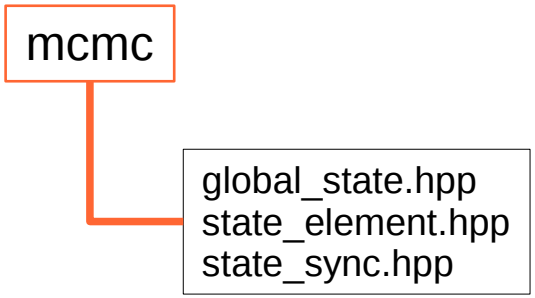
The libLSS/data subtree

data

- spectro_gals.hpp
- window3d.hpp
- galaxies.hpp
- projection.hpp
- linear_selection.hpp
- schechter_completeness.hpp

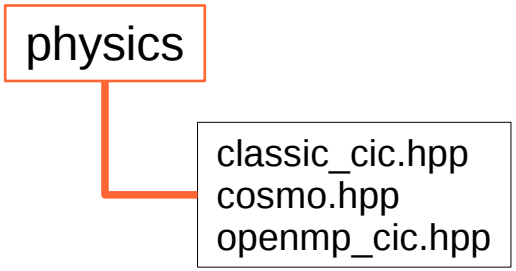
spectro_gals.hpp → Abstract definition of a galaxy survey (spectroscopic, but also photo-z is possible to be described).
window3d.hpp → Algorithm to compute the selection in 3d volume from 2d+1d information.
galaxies.hpp → Define structure that describe a galaxy in a survey. By default only spectral information is contained. Additional photo-z possible.
projection.hpp → Nearest grid point projection of galaxies from a survey to a 3d grid.
linear_selection.hpp → Implements a radial selection function defined piecewise, with linear interpolation
schechter_completeness.hpp → A toolset to compute radial completeness from Schechter function parameters.

The libLSS/mcmc subtree



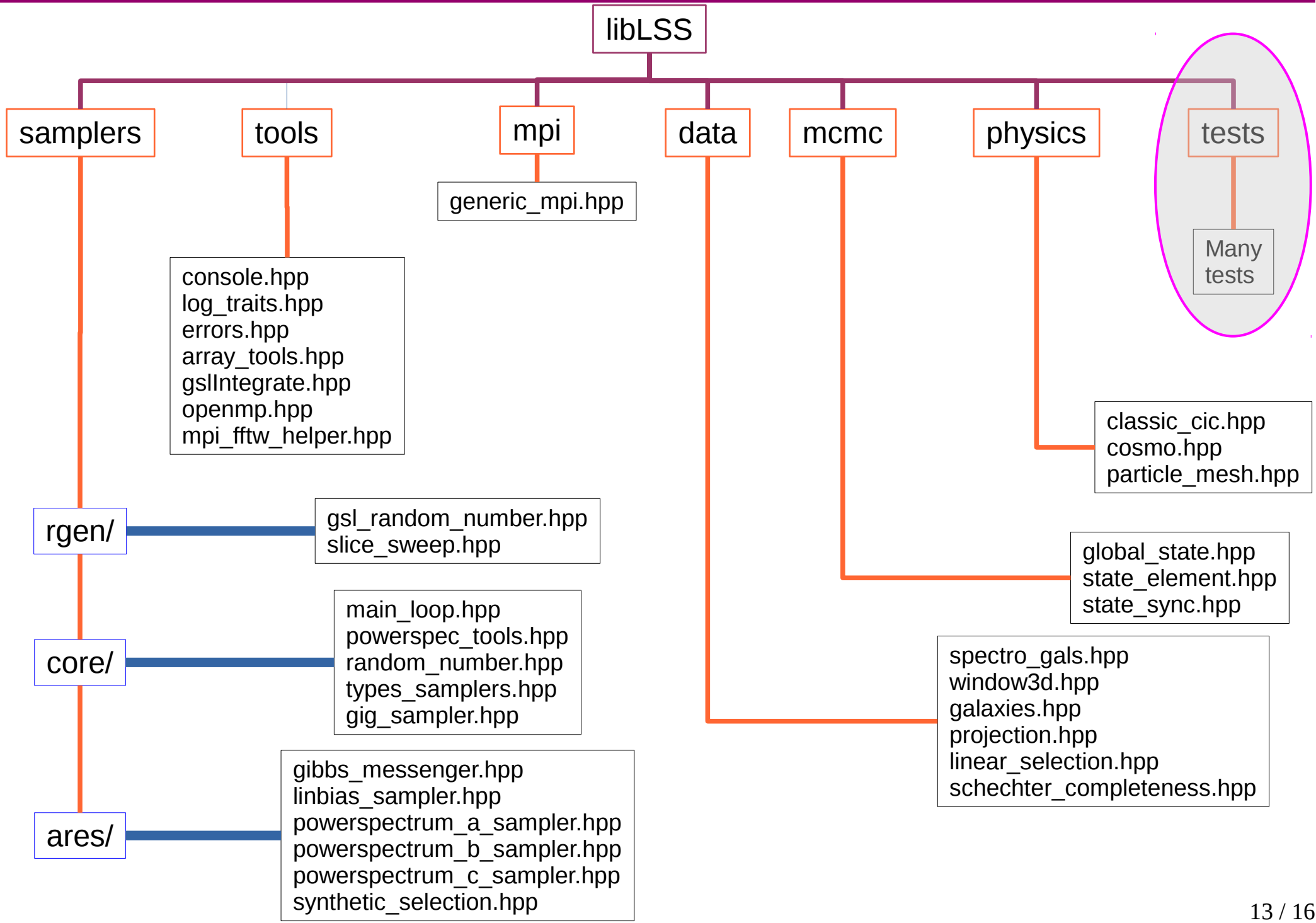
global_state.hpp → Holds many state elements together in a dictionnary ready to be serialized in a HDF5 file.
state_element.hpp → Set of classes to hold and describe how to serialize elements of a Markov Chain state. For example, you may have scalars (ScalarStateElement) or full N-dimensional arrays (GenericArrayStateElement). The arrays also support MPI-splitting. Regeneration happens for mcmc_*.h5 files (see later)
state_sync.hpp→Special class to synchronize several state elements variables over MPI. Not fully used at the moment and can be mostly ignored.

The libLSS/physics subtree



classic_cic.hpp → A canonical implement of a Cloud-In-Cell
openmp_cic.hpp → An openmp version of the same algorithm. More expensive in memory but faster when lots of particles must be injected on a "few" cells.
cosmo.hpp → Cosmology class that implements the diverse quantity related to classical homogeneous Universe expansion and its perturbations (Hubble, growth factor, comoving distance, …).

The libLSS tree



More on libLSS tree

This was the first part of the tree!

Each module adds new capabilities and tools to the library.

ARES_FG

New sampler libLSS/samplers/ares_fg/negative_foreground_sampler.hpp
This handles systematic effects in the target selection.

HADES

New random number generator (also generic sampler infrastructure) based on Hamiltonian Markov chain:

libLSS/samplers/rgen/hmc/hmc_density_sampler.hpp

Likelihoods and their adjoint gradient + Meta parameters:

(Poisson + log transform + power law bias)
libLSS/samplers/hades/hades_likelihood.hpp
libLSS/samplers/hades/hades_meta.hpp

(Gaussian + linear bias / ARES model)
libLSS/samplers/hades/hades_linear_likelihood.hpp
libLSS/samplers/hades/hades_linear_meta.hpp

(Gaussian + log-transform)
libLSS/samplers/hades/hades_lognormal_.hpp
libLSS/samplers/hades/hades_linear_meta.hpp

Abstract forward model description

libLSS/physics/forward_model.hpp

Generic symplectic integrators

libLSS/tools/symplectic_integrator.hpp

More on libLSS tree

This was the first part of the tree!

Each module adds new capabilities and tools to the library.

BORG

Complex forward models:

LPT

libLSS/samplers/borg/borg_lpt.hpp

PM

libLSS/samplers/borg/borg_pm.hpp

Likelihoods:

Poisson + Truncated power law bias + generic forward model

libLSS/samplers/borg/borg_poisson_likelihood.hpp
libLSS/samplers/borg/borg_poisson_meta.hpp

Gaussian + Linear bias + generic forward model

libLSS/samplers/borg/borg_linear_likelihood.hpp
libLSS/samplers/borg/borg_linear_meta.hpp

! Other models are in this directory, but do not use them !

Data structures