3ML and astromodels

Abstracting multi-messenger astronomy

threeml.readthedocs.io

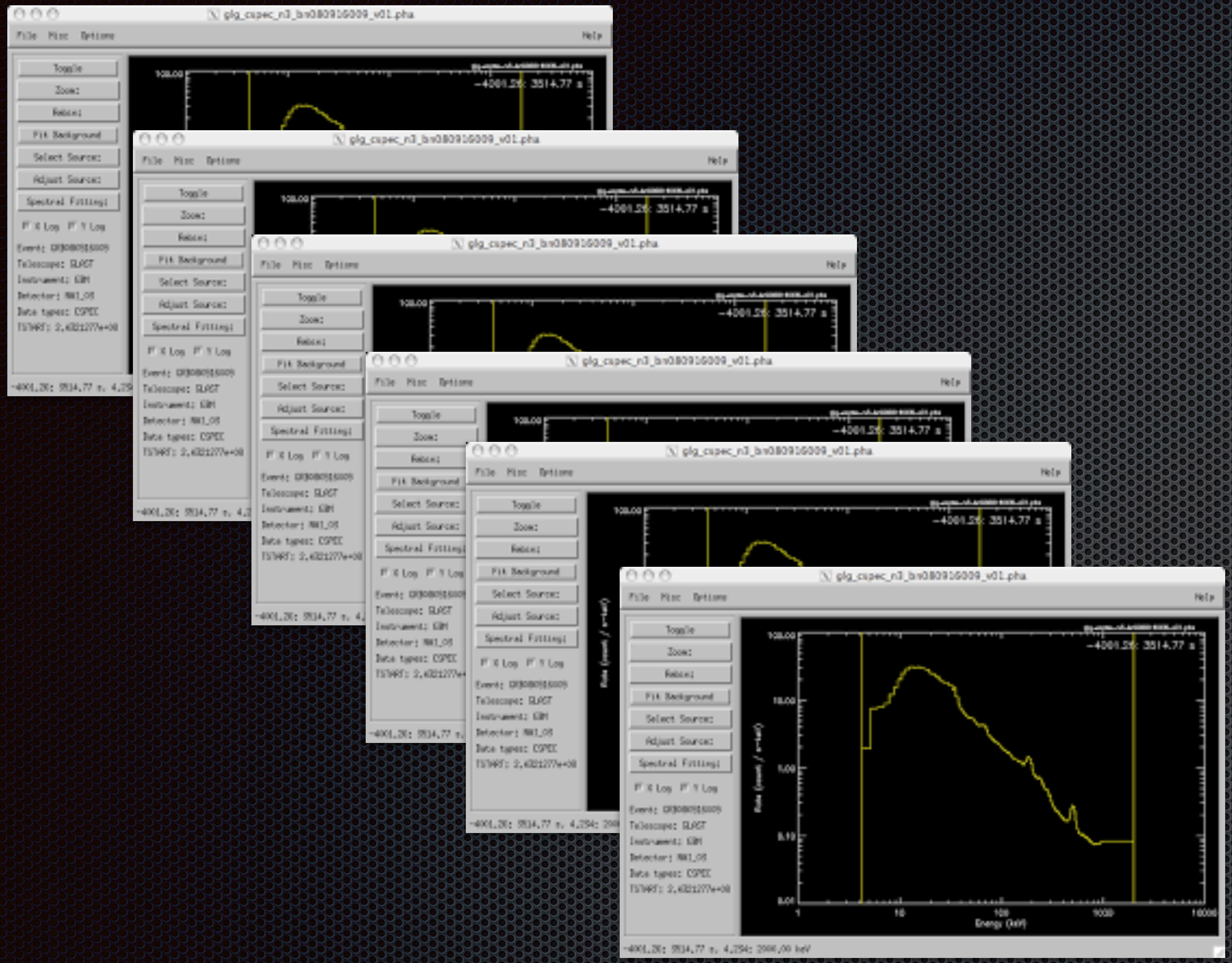
J. Michael Burgess and Giacomo Vianello

From a plotting library to an unscriptable nightmare

We were comfortable with IDL

We built analysis frameworks with it

Moreover, we couldn't analyze GBM+LAT together properly



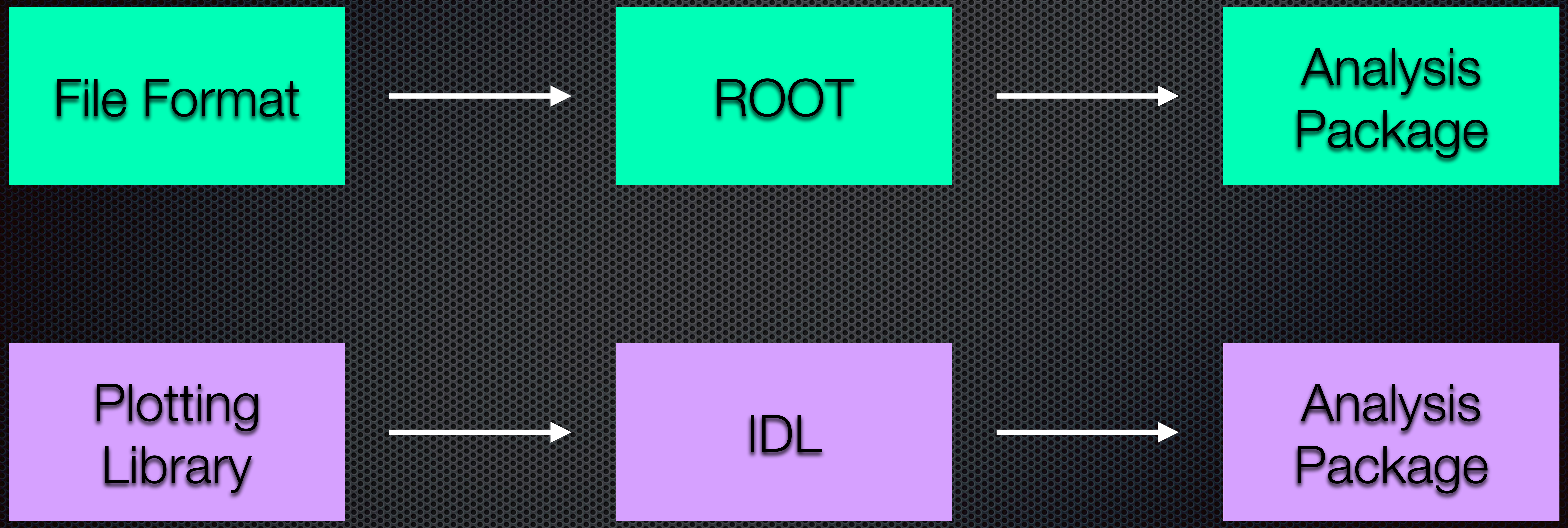
From a plotting library to an unscriptable nightmare

We were comfortable with IDL

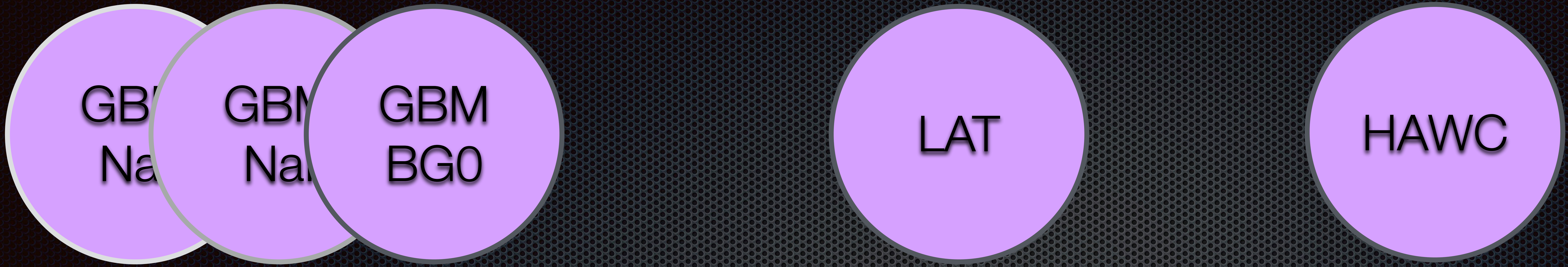
We built analysis frameworks with it

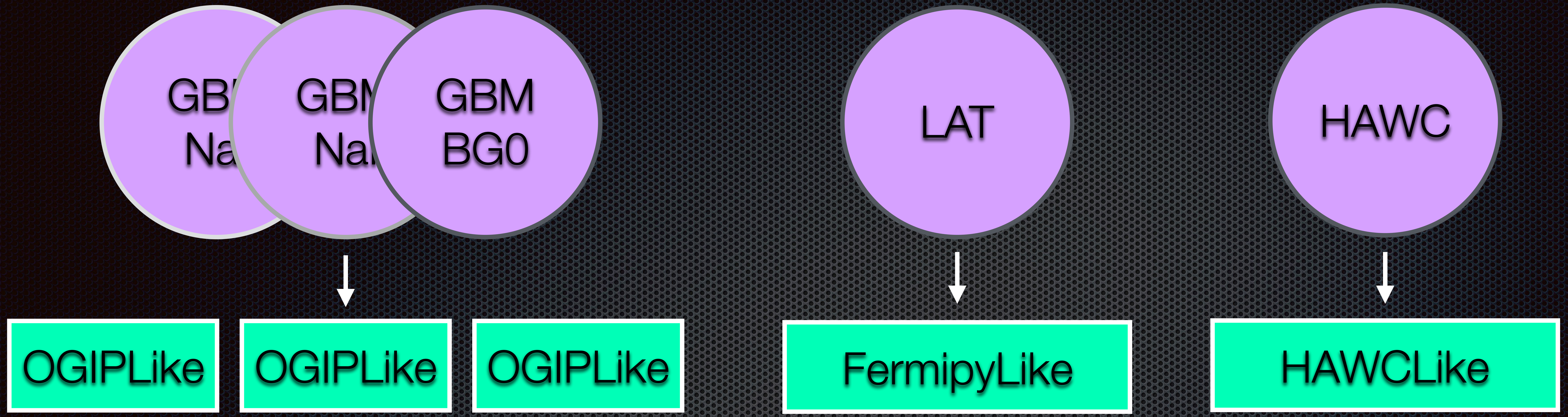
Moreover, we couldn't analyze GBM+LAT together properly

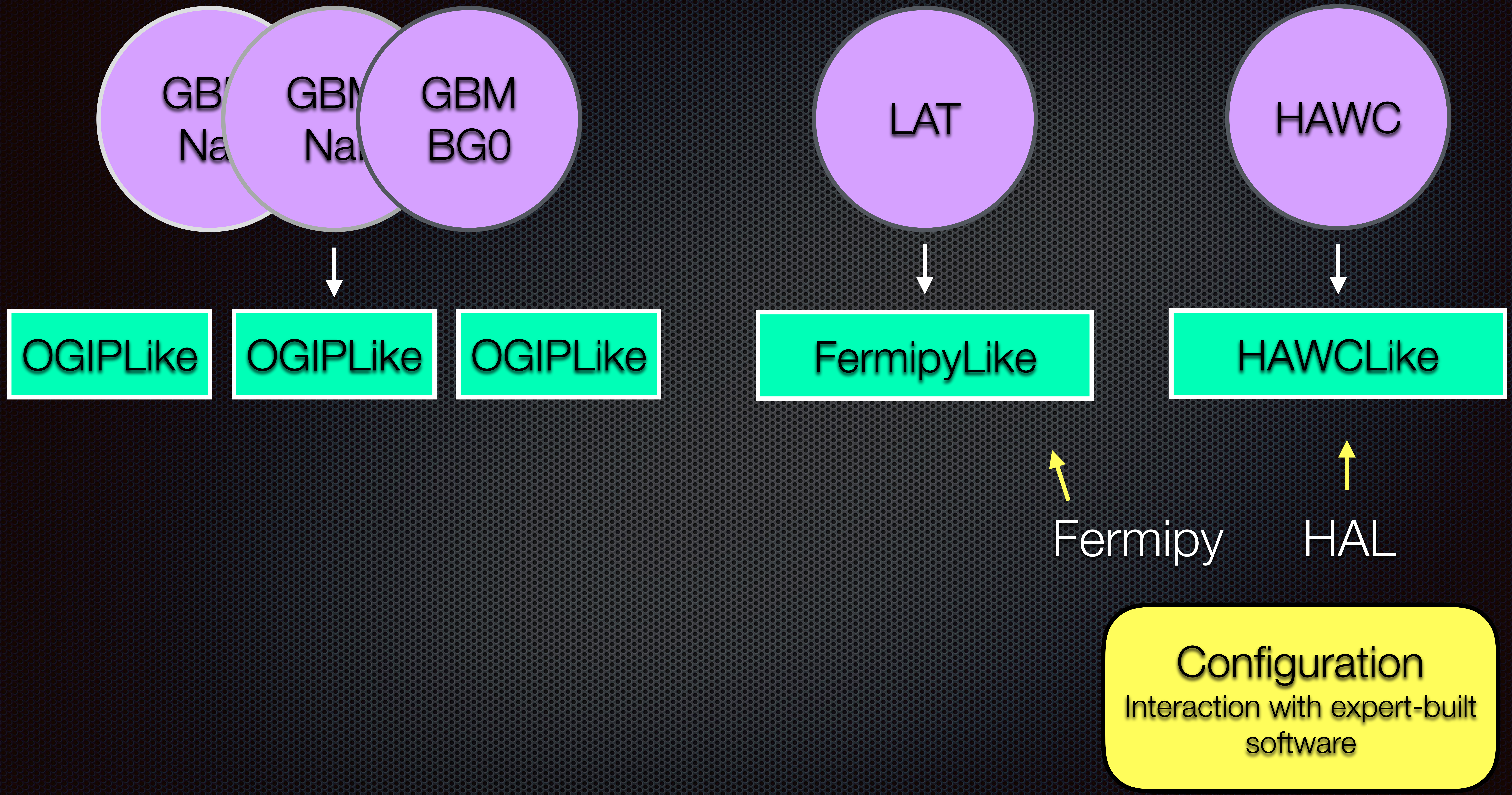
Extending a tool past its purpose

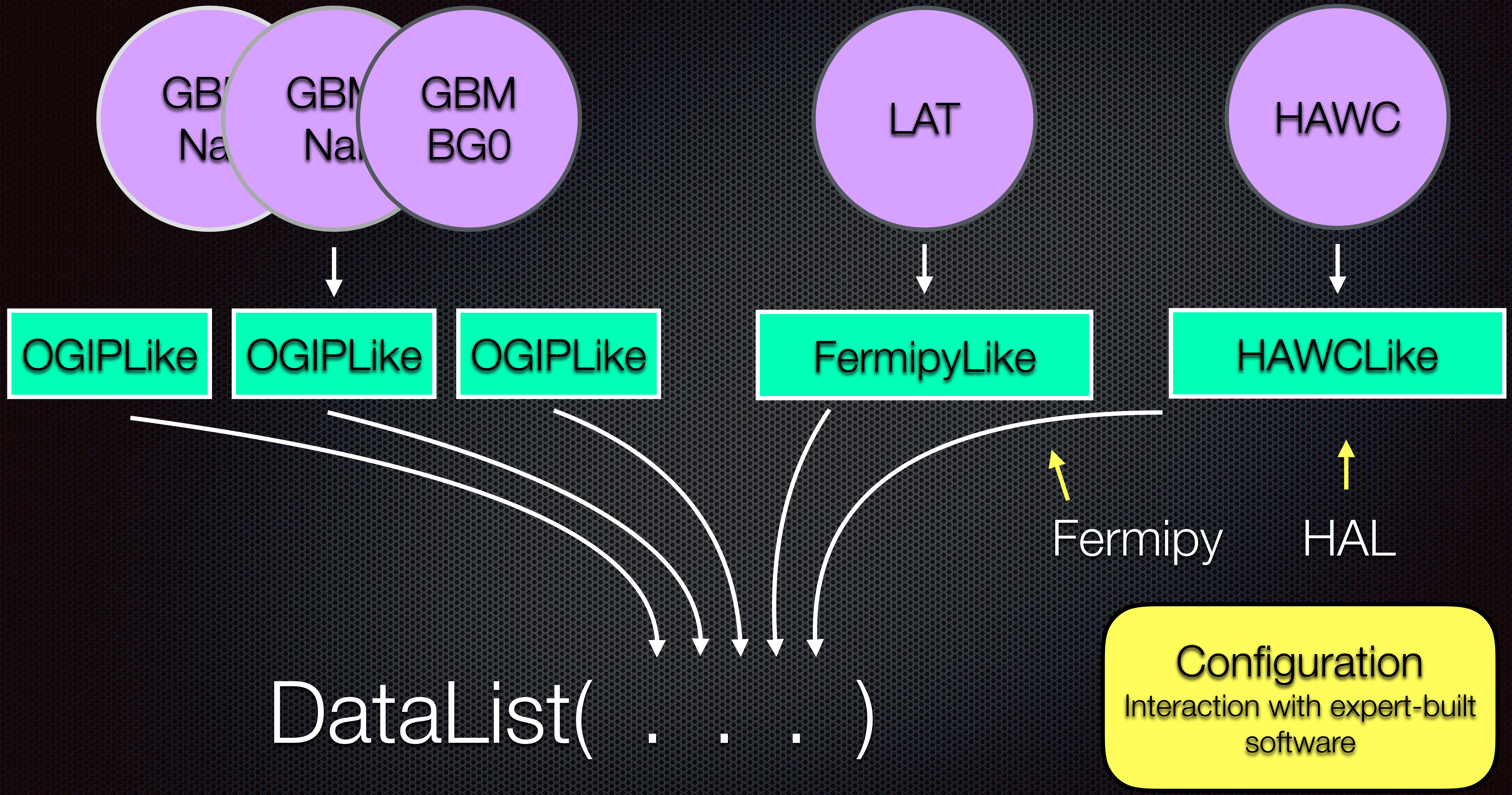


point source
data I
instrument I
plugin I
likelihood I
neutrino source
data II
instrument II
plugin II
likelihood II
data III
instrument III
plugin III
likelihood III
diffuse source
extended source
plot
model check
propagate
parameter exploration
share
compare
save


GBM
BG0
LAT
HAWC


GBM
BG0
LAT
HAWC
OGIPLike
OGIPLike
OGIPLike
FermipyLike
HAWCLike


GBM BG0
LAT
HAWC
OGIPLike
OGIPLike
OGIPLike
FermipyLike
HAWCLike
Fermipy
HAL
Configuration
Interaction with expert-built software


GBM
BG0
LAT
HAWC
OGIPLike
OGIPLike
OGIPLike
FermipyLike
HAWCLike
Fermipy
HAL
DataList(. . .)
Configuration
Interaction with expert-built software

[1]:

```python
from threeML import *
```

```
Configuration read from /home/giacomov/.threeML/threeML_config
Plotter is MatPlotlib
```

[2]:

```python
# Get some example data
from threeML.io.package_data import get_path_of_data_file

data_path = get_path_of_data_file("datasets/xy_powerlaw.txt")

# Create an instance of the XYLike plugin, which allows to ana
# with error bars
xyl = XYLike.from_text_file("xyl", data_path)

# Let's plot it just to see what we have loaded
xyl.plot(x_scale='log', y_scale='log')
```

```
Using Gaussian statistic (equivalent to chi^2) with the provid
```

[3]:
```
data = DataList(xyl)
```

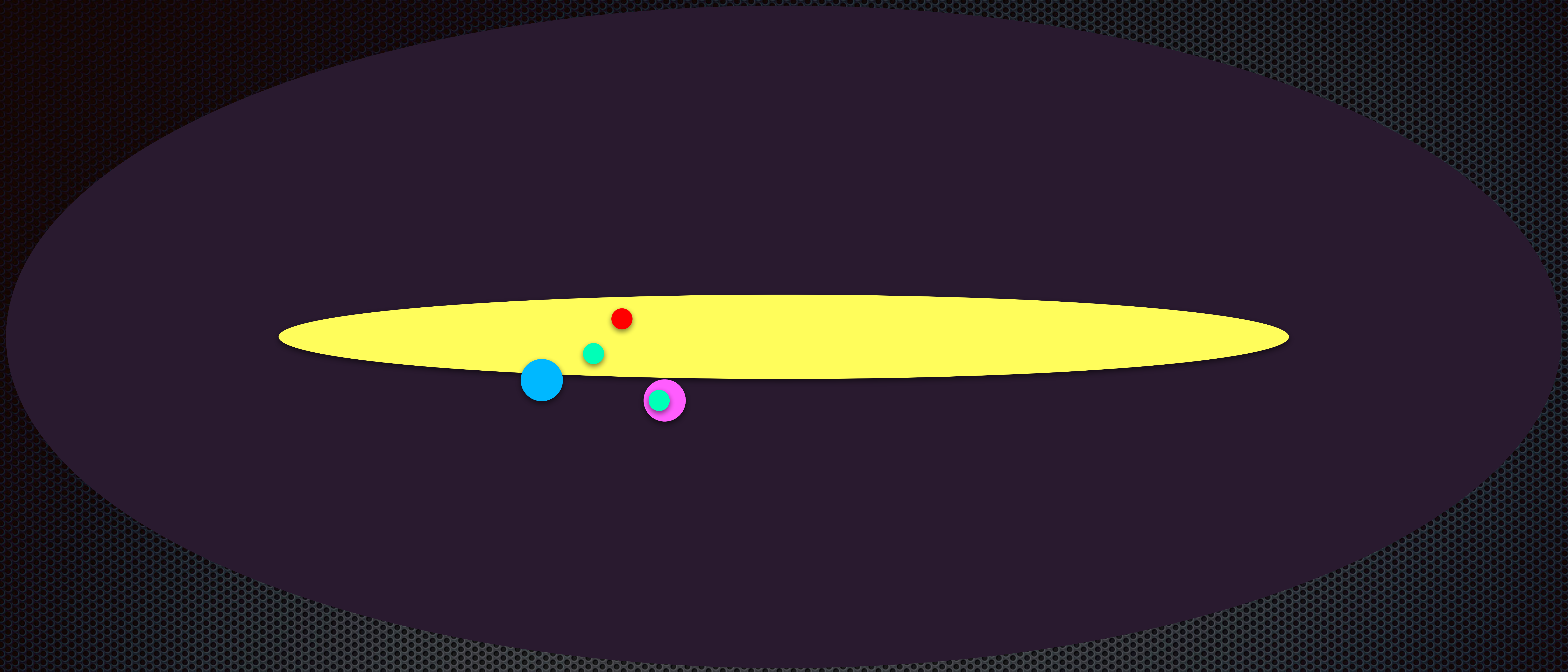
[4]:
```
# Create the second instance, this time of a different type

pha = get_path_of_data_file("datasets/ogip_powerlaw.pha")
bak = get_path_of_data_file("datasets/ogip_powerlaw.bak")
rsp = get_path_of_data_file("datasets/ogip_powerlaw.rsp")

ogip = OGIPLike("ogip", pha, bak, rsp)

# Now use both plugins
data = DataList(xyl, ogip)
```

```
Auto-probed noise models:
- observation: poisson
- background: poisson
```

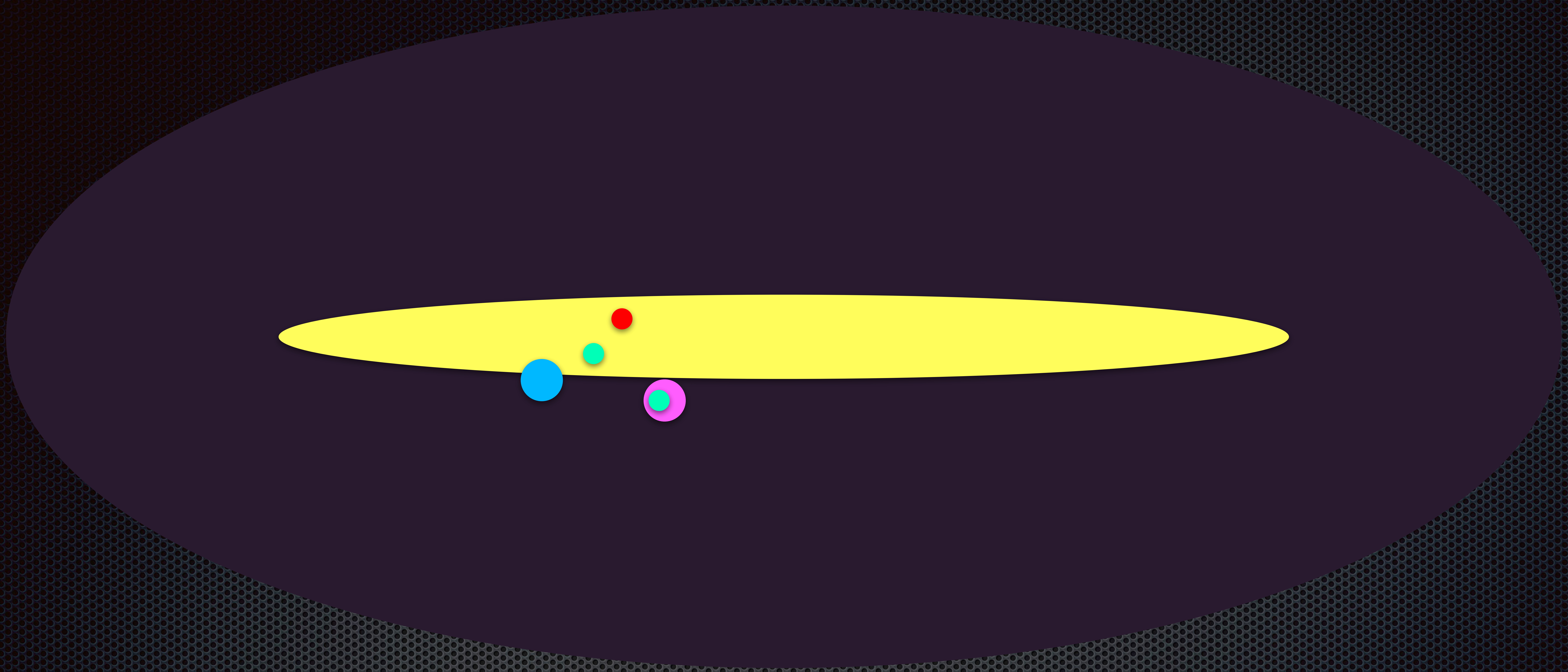
[5]:
```
# This is equivalent to write data = DataList(xyl, ogip)

my_plugins = [xyl, ogip]
data = DataList(*my_plugins)
```

Astromodels (https://astromodels.readthedocs.io/en/latest/)

- A node-tree based modeling framework
- Implemented in cPython for serialization
- 1,2,3D modeling
- Linking of parameters with arbitrary functions
- Point-, Extended-, Particle-, Polarization-source
- Table models, time-varying models.



Model(PointSource(name, ra, dec, spectrum , polarization) PointSource(name, ra, dec, spectrum)
ExtendedSource(name, ra, dec, spectrum, **shape**) ParticleSource(name, ra, dec, spectrum)
PointSource(name, ra, dec, spectrum) NeutrinoSource(name, ra, dec, spectrum))



Model(PointSource(name, ra, dec, spectrum , polarization) PointSource(name, ra, dec, spectrum)
ExtendedSource(name, ra, dec, spectrum, **shape**) ParticleSource(name, ra, dec, spectrum)
PointSource(name, ra, dec, spectrum) NeutrinoSource(name, ra, dec, spectrum))

Does the CR spectrum depend on the neutrino spectrum?

Fitting an extended source with an embedded point source?

astromodels supports arbitrary linking between parameters . This also allows for the specification of time-varying models.

Model(
PointSource(name, ra, dec, spectrum , polarization)   PointSource(name, ra, dec, spectrum)
ExtendedSource(name, ra, dec, spectrum, shape)   ParticleSource(name, ra, dec, spectrum)
PointSource(name, ra, dec, spectrum)   NeutrinoSource(name, ra, dec, spectrum)
)

[6]:

```
# A point source with a power law spectrum

source1_sp = Powerlaw()
source1 = PointSource("source1", ra=23.5, dec=-22.7, spectral_

# Another source with a log-parabolic spectrum plus a power la

source2_sp = Log_parabola() + Powerlaw()
source2 = PointSource("source2", ra=30.5, dec=-27.1, spectral_

# A third source defined in terms of its Galactic latitude and
source3_sp = Cutoff_powerlaw()
source3 = PointSource("source3", l=216.1, b=-74.56, spectral_s
```

[7]:

```
# An extended source with a Gaussian shape centered on R.A., D
# and a sigma of 3.0 degrees
ext1_spatial = Gaussian_on_sphere(lon0=30.5, lat0=-27.1, sigma
ext1_spectral = Powerlaw()

ext1 = ExtendedSource("ext1", ext1_spatial, ext1_spectral)

# An extended source with a 3D function
# (i.e., the function defines both the spatial and the spectra
ext2_spatial = Continuous_injection_diffusion()
ext2 = ExtendedSource("ext2", ext2_spatial)
```

[8]:

```
model = Model(source1, source2, source3, ext1, ext2)

# We can see a summary of the model like this:
model.display()
```

Model summary:

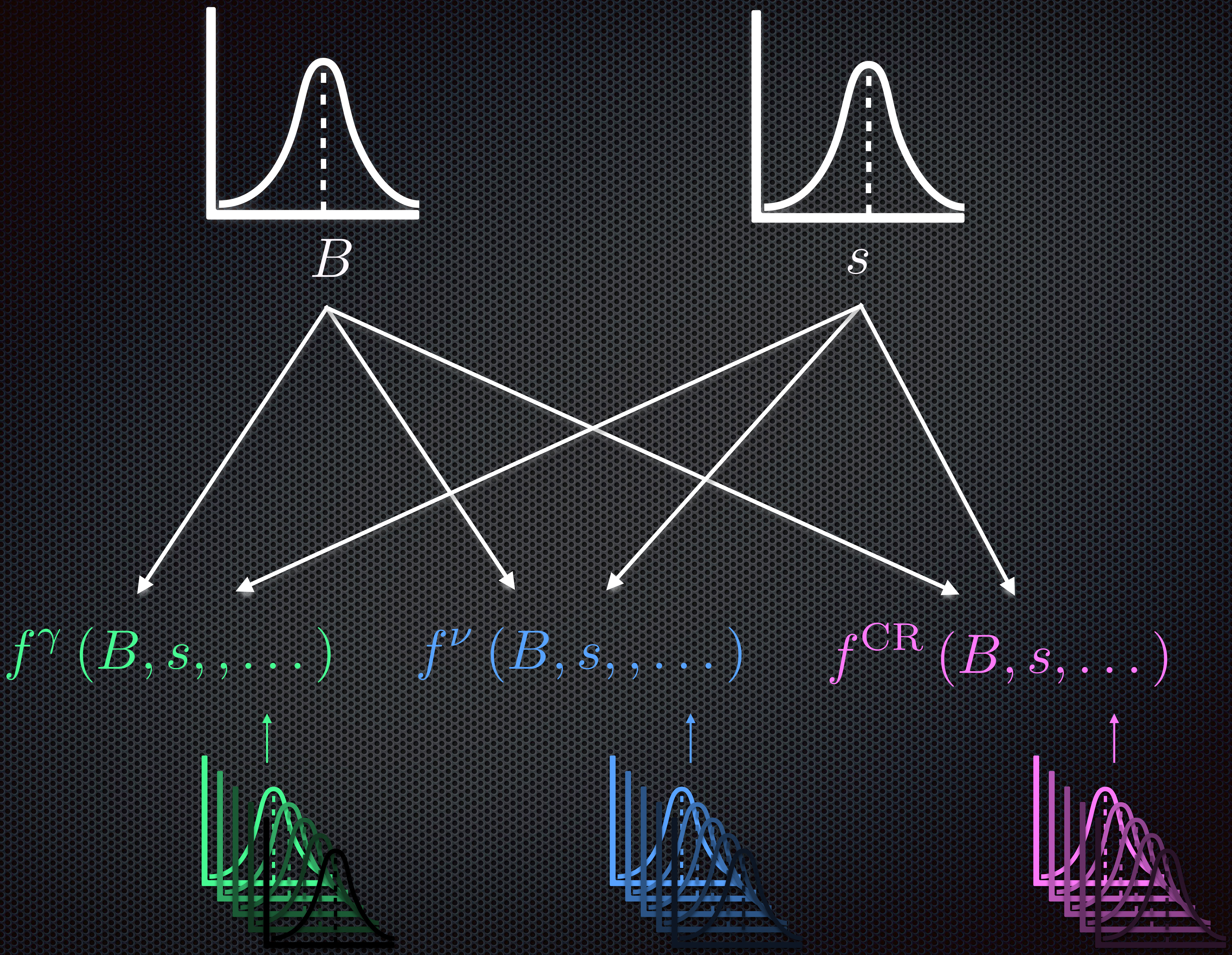
	N
Point sources	3
Extended sources	2
Particle sources	0

Free parameters (19):

	value	min_value	max_value	unit
source1.spectrum.main.Powerlaw.K	1	1e-30	1000	cm-2 keV-1 s-1
source1.spectrum.main.Powerlaw.index	-2	-10	10	
source2.spectrum.main.composite.K_1	1	1e-30	100000	cm-2 keV-1 s-1
source2.spectrum.main.composite.alpha_1	-2	None	None	
source2.spectrum.main.composite.beta_1	1	None	None	
source2.spectrum.main.composite.K_2	1	1e-30	1000	cm-2 keV-1 s-1
source2.spectrum.main.composite.index_2	-2	-10	10	
source3.spectrum.main.Cutoff_powerlaw.K	1	1e-30	1000	cm-2 keV-1 s-1
source3.spectrum.main.Cutoff_powerlaw.index	-2	-10	10	
source3.spectrum.main.Cutoff_powerlaw.xc	10	None	None	keV
ext1.Gaussian_on_sphere.lon0	30.5	0	360	deg
ext1.Gaussian_on_sphere.lat0	-27.1	-90	90	deg
ext1.Gaussian_on_sphere.sigma	3	0	20	deg
ext1.spectrum.main.Powerlaw.K	1	1e-30	1000	cm-2 keV-1 s-1
ext1.spectrum.main.Powerlaw.index	-2	-10	10	
ext2.Continuous_injection_diffusion.lon0	0	0	360	deg
ext2.Continuous_injection_diffusion.lat0	0	-90	90	deg
ext2.Continuous_injection_diffusion.rdiff0	1	0	20	deg
ext2.spectrum.main.Constant.k	0	None	None	

Fixed parameters (15):
(abridged. Use complete=True to see all fixed parameters)

Linked parameters (0):



B
s
$f^{\gamma}(B, s, , \ldots)$
$f^{\nu}(B, s, , \ldots)$
$f^{\mathrm{CR}}(B, s, \ldots)$

Model(PointSource(name, ra, dec, spectrum , polarization) ExtendedSource(name, ra, dec, spectrum, **shape**) PointSource(name, ra, dec, spectrum) PointSource(name, ra, dec, spectrum) ParticleSource(name, ra, dec, spectrum) NeutrinoSource(name, ra, dec, spectrum))

DataList(plugin1, plugin2,…)

JointLikelihood(Model(PointSource(name, ra, dec, spectrum , polarization) PointSource(name, ra, dec, spectrum)
ExtendedSource(name, ra, dec, spectrum, **shape**) ParticleSource(name, ra, dec, spectrum)
PointSource(name, ra, dec, spectrum) NeutrinoSource(name, ra, dec, spectrum))
DataList(plugin1, plugin2,…))

JointLikelihood(Model(PointSource(name, ra, dec, spectrum , polarization) ExtendedSource(name, ra, dec, spectrum, **shape**) PointSource(name, ra, dec, spectrum) PointSource(name, ra, dec, spectrum) ParticleSource(name, ra, dec, spectrum) NeutrinoSource(name, ra, dec, spectrum))
DataList(plugin1, plugin2,…))

Model(PointSource(name, ra, dec, spectrum , polarization) ExtendedSource(name, ra, dec, spectrum, **shape**) PointSource(name, ra, dec, spectrum) PointSource(name, ra, dec, spectrum) ParticleSource(name, ra, dec, spectrum) NeutrinoSource(name, ra, dec, spectrum))
DataList(plugin1, plugin2,…)

JointLikelihood(Model(PointSource(name, ra, dec, spectrum , polarization) ExtendedSource(name, ra, dec, spectrum, **shape**) PointSource(name, ra, dec, spectrum) PointSource(name, ra, dec, spectrum) ParticleSource(name, ra, dec, spectrum) NeutrinoSource(name, ra, dec, spectrum)) DataList(plugin1, plugin2,…))

BayesianAnalysis(Model(PointSource(name, ra, dec, spectrum , polarization) ExtendedSource(name, ra, dec, spectrum, **shape**) PointSource(name, ra, dec, spectrum) PointSource(name, ra, dec, spectrum) ParticleSource(name, ra, dec, spectrum) NeutrinoSource(name, ra, dec, spectrum)) DataList(plugin1, plugin2,…))

Analysis Results

- An object and a FITS file.
- Stores all the information about your model and fit
- The same interface for MLE or Bayes
- Transportable

parameter	result	unit
fake.spectrum.main.composite.a_1	(-4.000 +/- 4) x 10^-3	1 / (cm2 keV2 s)
fake.spectrum.main.composite.b_1	2.060 +/- 0.11	1 / (cm2 keV s)
fake.spectrum.main.composite.F_2	(2.900 +/- 0.4) x 10	1 / (cm2 s)
fake.spectrum.main.composite.mu_2	(2.483 +/- 0.013) x 10	keV
fake.spectrum.main.composite.sigma_2	1.140 +/- 0.10	keV

```
Correlation matrix:
```

1.00	-0.85	-0.00	-0.03	0.00
-0.85	1.00	-0.05	0.01	-0.09
-0.00	-0.05	1.00	0.17	-0.20
-0.03	0.01	0.17	1.00	0.18
0.00	-0.09	-0.20	0.18	1.00

```
Values of -log(likelihood) at the minimum:
```

	-log(likelihood)
sim_data	26.967829
total	26.967829

Error propagation

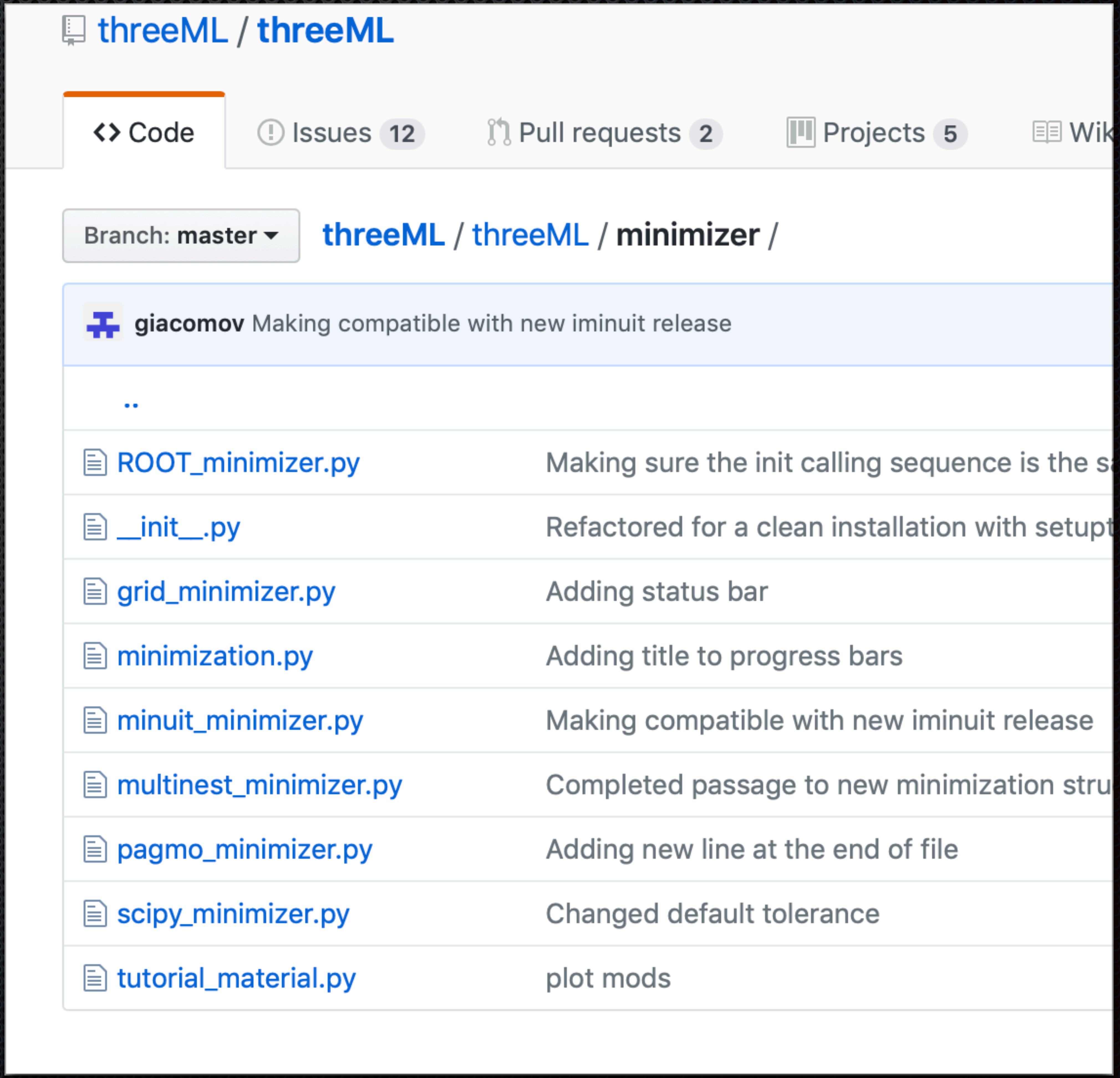
- Parameters are RandomVariates
- Full support for non-linear error propagation for any analysis

```
[53]: p1 = ar.get_variates("fake.spectrum.main.composite.a_1")
      p2 = ar.get_variates("fake.spectrum.main.composite.b_1")

      print("Propagating a+b, with a and b respectively:")
      print(p1)
      print(p2)

      print("\nThis is the result (with errors):")
      res = p1 + p2
      print(res)

      print(res.equal_tail_interval())
```

```
Propagating a+b, with a and b respectively:
equal-tail: (-4.000 +/- 4) x 10^-3, hpd: (-4.000 +/- 4) x 10^-3
equal-tail: 2.060 +/- 0.11, hpd: 2.06 -0.12 +0.11

This is the result (with errors):
equal-tail: 2.050 +/- 0.11, hpd: 2.050 +/- 0.11
(1.9430220946678243, 2.1626737366884683)
```

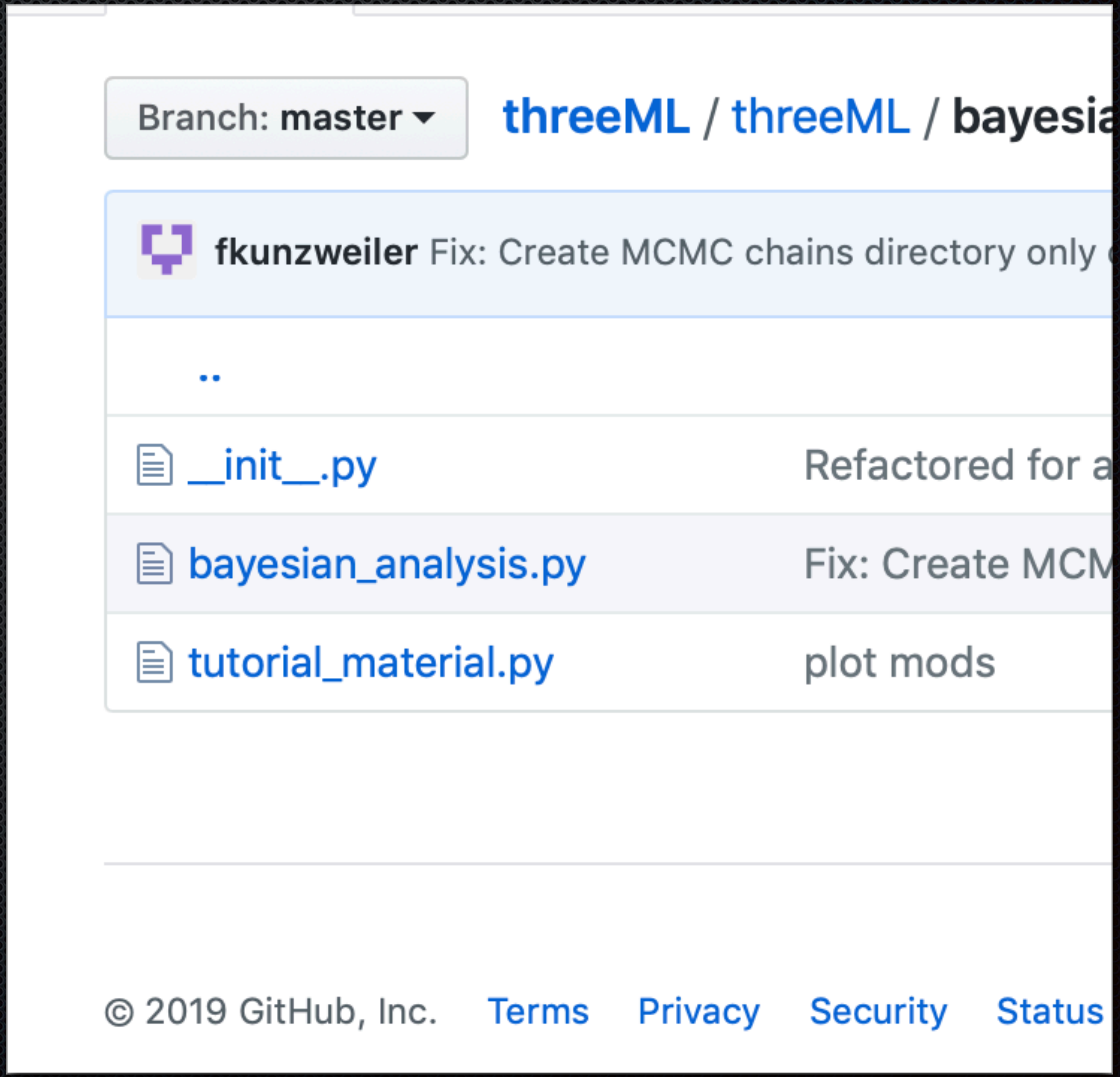
Long story short

- Optimizers
  - ROOT
  - iminuit
  - PAGMO (10^6 optimizers)



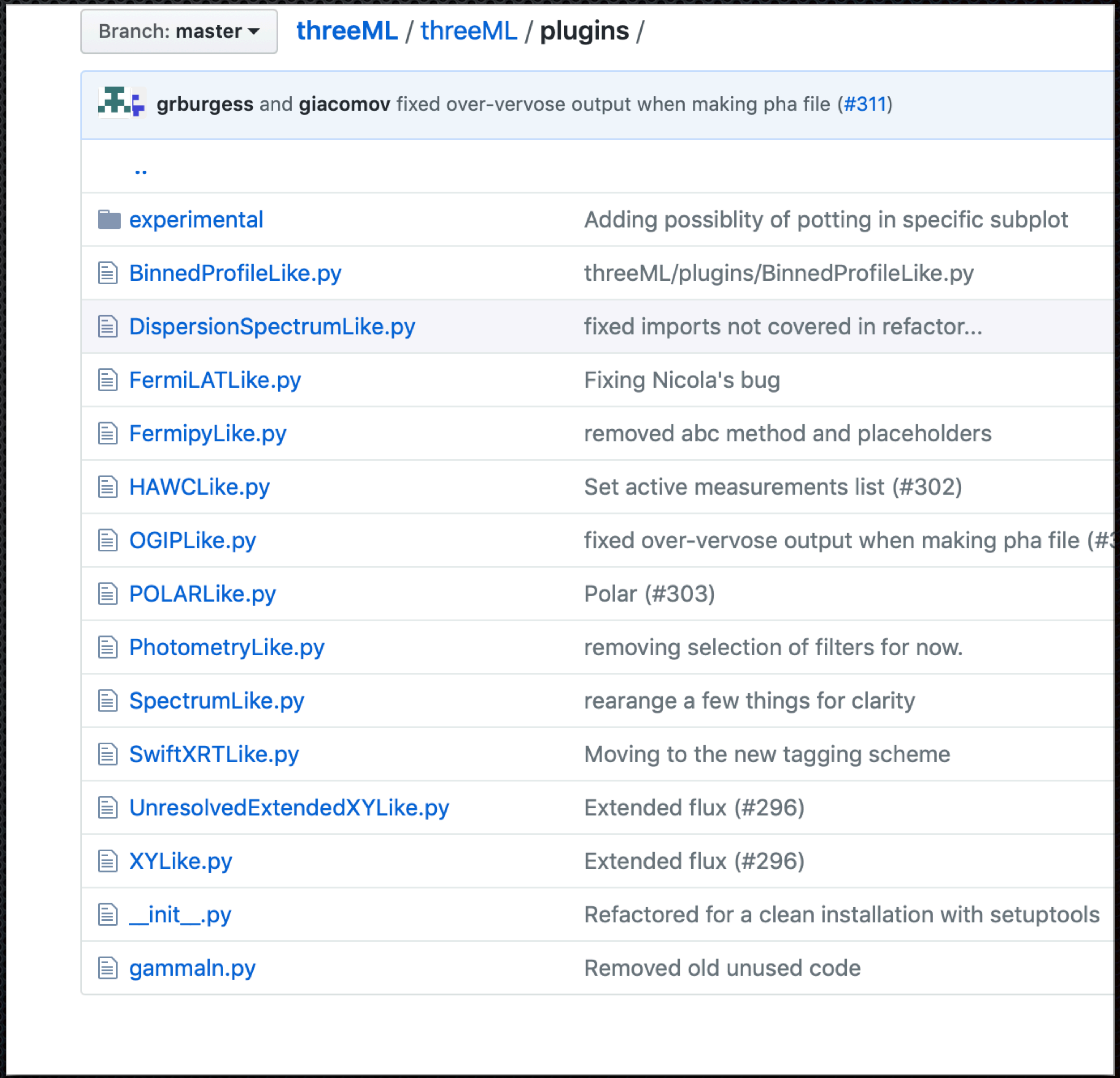
Long story short

- Posterior samplers
  - emcee
  - MULTINEST
  - PolyChord



Long story short

- Plugins
  - XYLike
  - SpectrumLike -> DispersionSpectrumLike -> OGIPLike
  - FermiLATLike / FermiPyLike
  - HAWCLike
  - CastroLike
  - VeritasLike
  - PhotometryLike
  - More and more.



Long story short

- Some low-level data builders
  - Reduce time series (energy/polarization/etc) to plugins
  - Build Fermipy configs
- Catalogs via VO
  - Fermi LAT/GBM
  - Swift
  - More coming (please help!)
- But we do not want to replace your team's software… you are the experts!

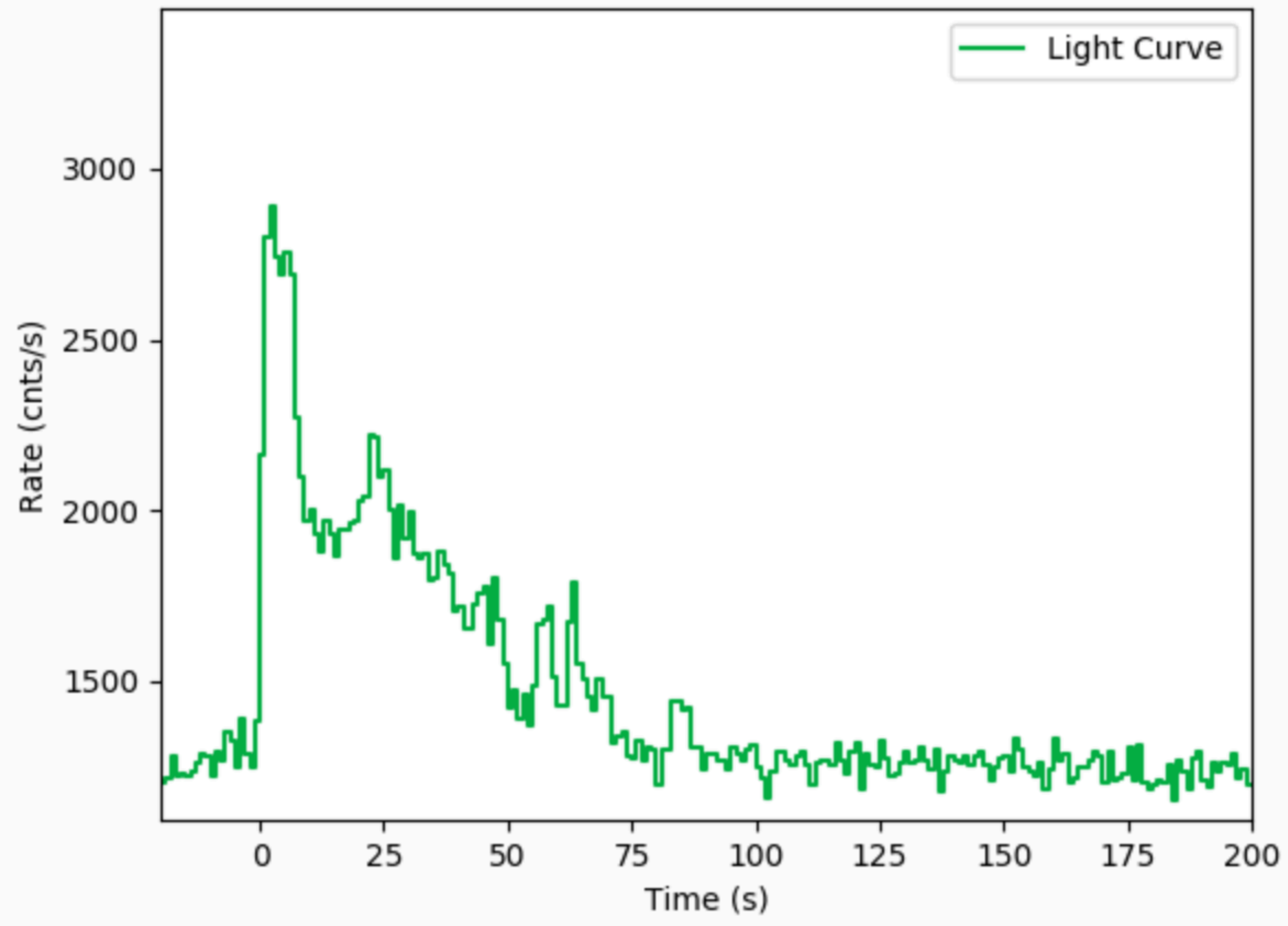
LAT LLE data is constructed in a similar fashion

[4]:
```
lle_file = get_path_of_data_file('datasets/gll_lle_bn080916009_v10.fit')
ft2_file = get_path_of_data_file('datasets/gll_pt_bn080916009_v10.fit')
lle_rsp = get_path_of_data_file('datasets/gll_cspec_bn080916009_v10.rsp')

lat_lle = TimeSeriesBuilder.from_lat_lle('lat_lle',
                                          lle_file=lle_file,
                                          ft2_file=ft2_file,
                                          rsp_file=lle_rsp)
```

Viewing Lightcurves and selecting source intervals

All time series objects share the same commands to get you to a plugin. Let's have a look at the GBM TTE lightcurve.

[5]:
```
threeML_config['lightcurve']['lightcurve color'] = '#07AE44'

gbm_tte.view_lightcurve(start=-20,stop=200);
```



Build your own!

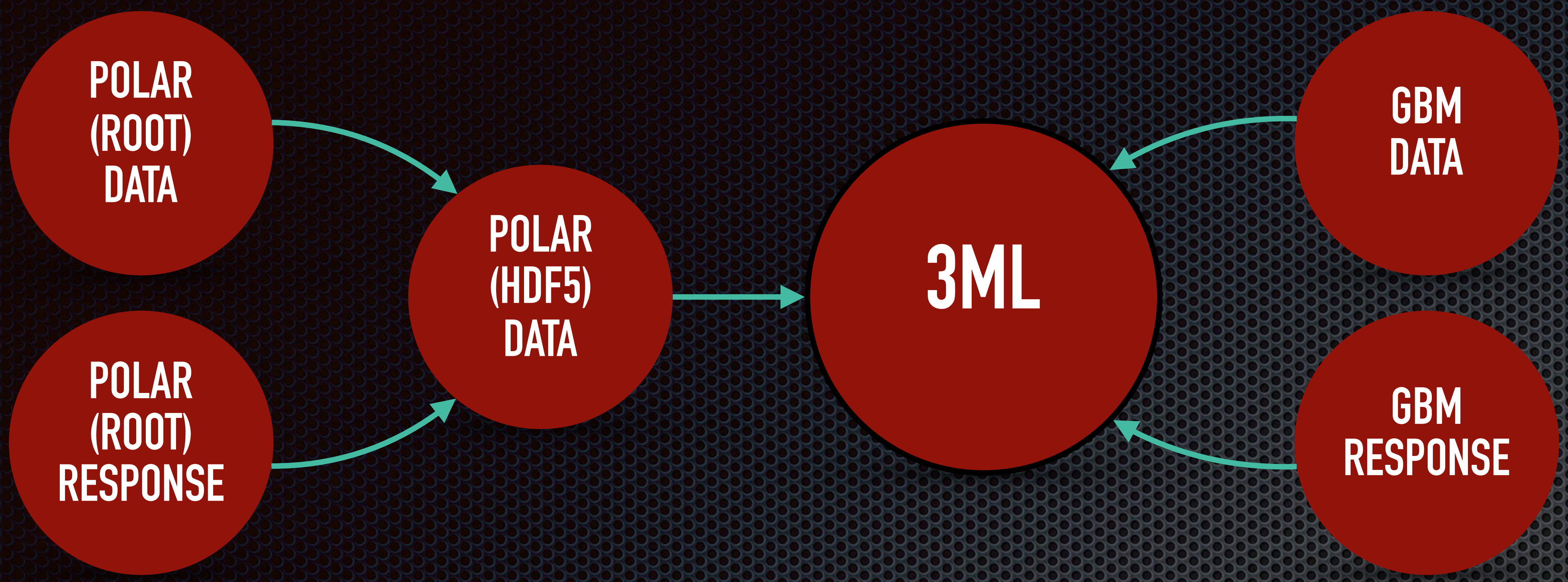
The PluginPrototype class

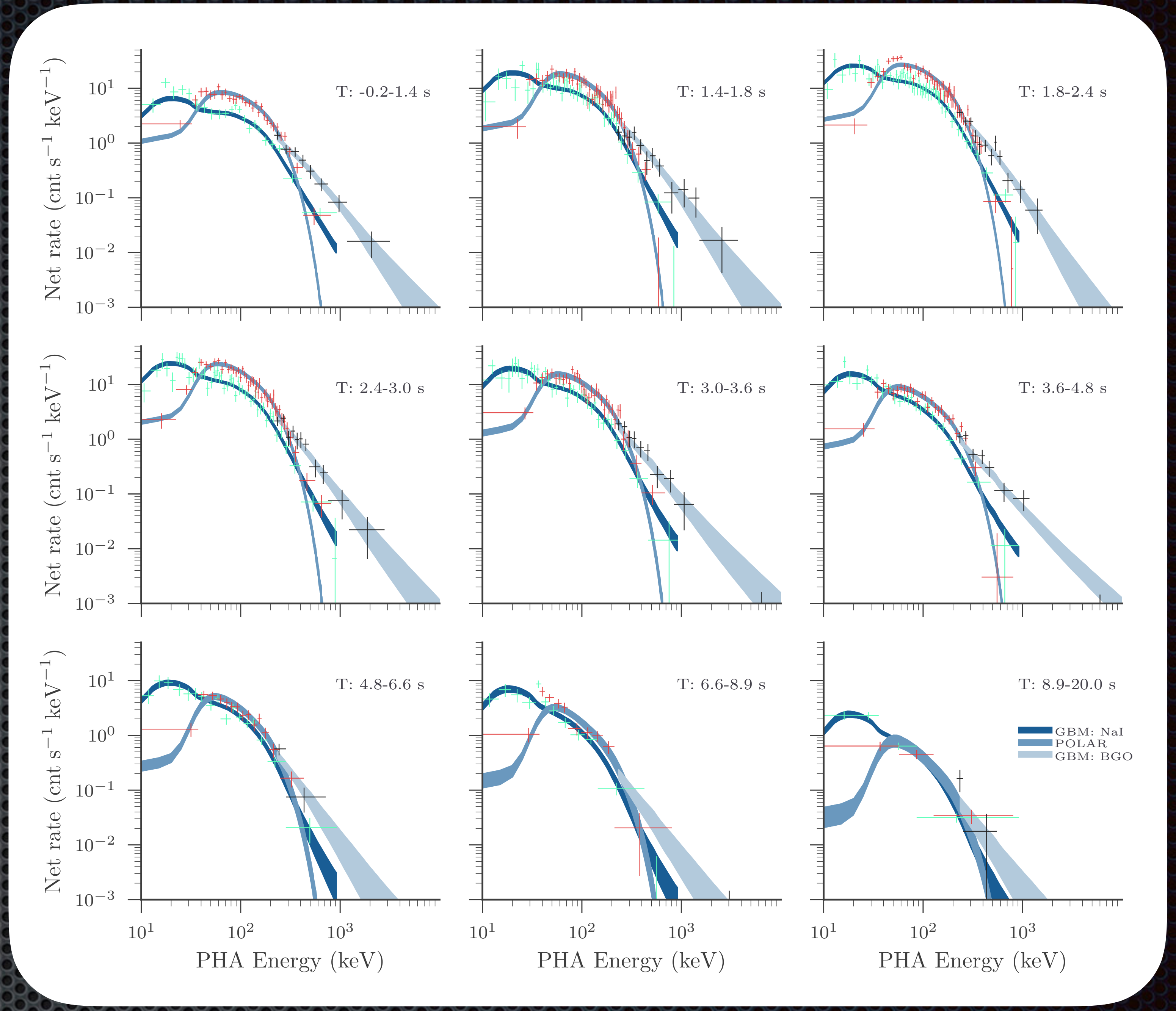
The basic functionality of any plugin is prototyped in the PluginPrototype class. This is under the main directory in the 3ML source code, but let's examine it here:

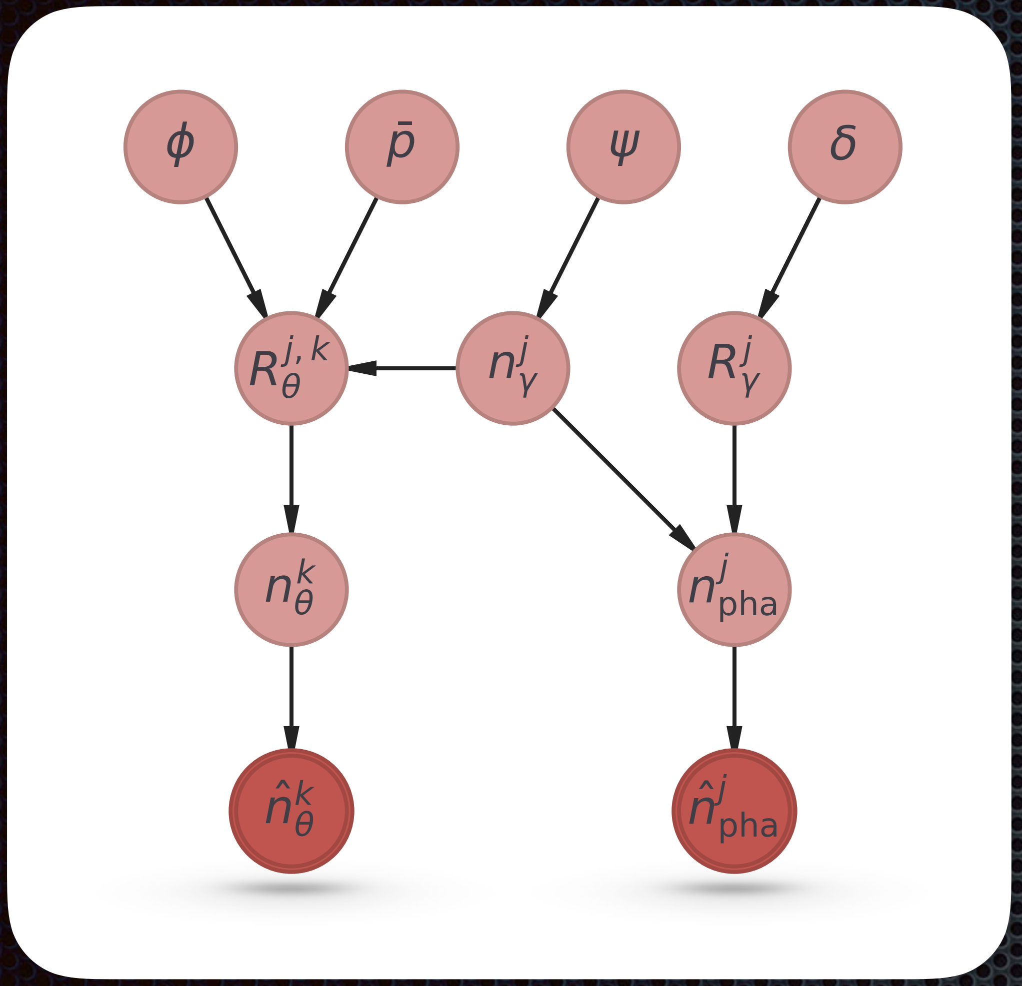
```
[ ]: class PluginPrototype(object):
         __metaclass__ = abc.ABCMeta

         def __init__(self, name, nuisance_parameters):
             assert is_valid_variable_name(name), "The name %s cannot be used as a name. You n
                                                  "python identifier: no spaces, cannot start
                                                  "operators symbols such as -, +, *, /" % nam

             # Make sure total is not used as a name (need to use it for other things, like th
             assert name.lower() != "total", "Sorry, you cannot use 'total' as name for a plug

             self._name = name

             # This is just to make sure that the plugin is legal

             assert isinstance(nuisance_parameters, dict)

             self._nuisance_parameters = nuisance_parameters

             # These are the external properties (time, polarization, etc.)
             # self._external_properties = []

             self._tag = None

         def get_name(self):
             warnings.warn("Do not use get_name() for plugins, use the .name property", Deprec

             return self.name

         @property
         def name(self):
             """
             Returns the name of this instance
             :return: a string (this is enforced to be a valid python identifier)
             """
             return self._name

         @property
         def nuisance_parameters(self):
             """
             Returns a dictionary containing the nuisance parameters for this dataset
             :return: a dictionary
             """

             return self._nuisance_parameters
```

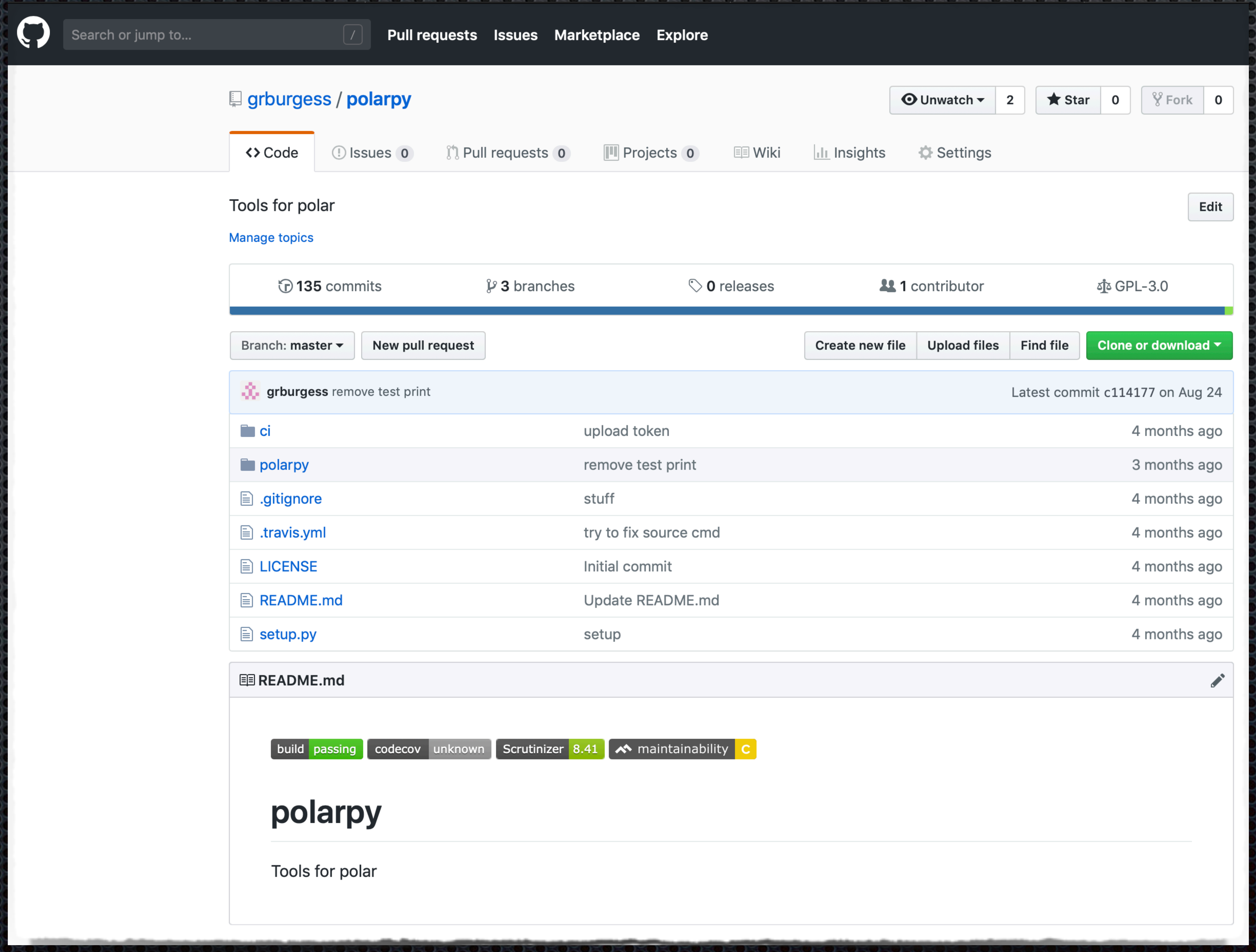
```
    def update_nuisance_parameters(self, new_nuisance_parameters):
        assert isinstance(new_nuisance_parameters, dict)

        self._nuisance_parameters = new_nuisance_parameters


    def get_number_of_data_points(self):
        """
        This returns the number of data points that are used to evaluate the likelihood.
        For binned measurements, this is the number of active bins used in the fit. For
        unbinned measurements, this would be the number of photons/particles that are
        evaluated on the likelihood
        """

        warnings.warn(
            "get_number_of_data_points not implemented, values for statistical measuremen
            "unreliable", )

        return 1.

    def _get_tag(self):

        return self._tag

    def _set_tag(self, spec):
        """
        Tag this plugin with the provided independent variable and a start and end value.
        This can be used for example to fit a time-varying model. In this case the indepe
        time and the start and end will be the start and stop time of the exposure for th
        be used to average the model over the provided time interval when fitting.
        :param independent_variable: an IndependentVariable instance
        :param start: start value for this plugin
        :param end: end value for this plugin. If this is not provided, instead of integr
        start and end, the model will be evaluate at start. Default: None (i.e., not prov
        :return: none
        """

        if len(spec) == 2:

            independent_variable, start = spec
            end = None

        elif len(spec) == 3:

            independent_variable, start, end = spec

        else:

            raise ValueError("Tag specification should be (independent_variable, start[,

        # Let's do a lazy check

        if not isinstance(independent_variable, IndependentVariable):

            warnings.warn("When tagging a plugin, you should use an IndependentVariable i
                          "an instance of a %s object. This might lead to crashes or "
                          "other problems." % type(independent_variable))
```

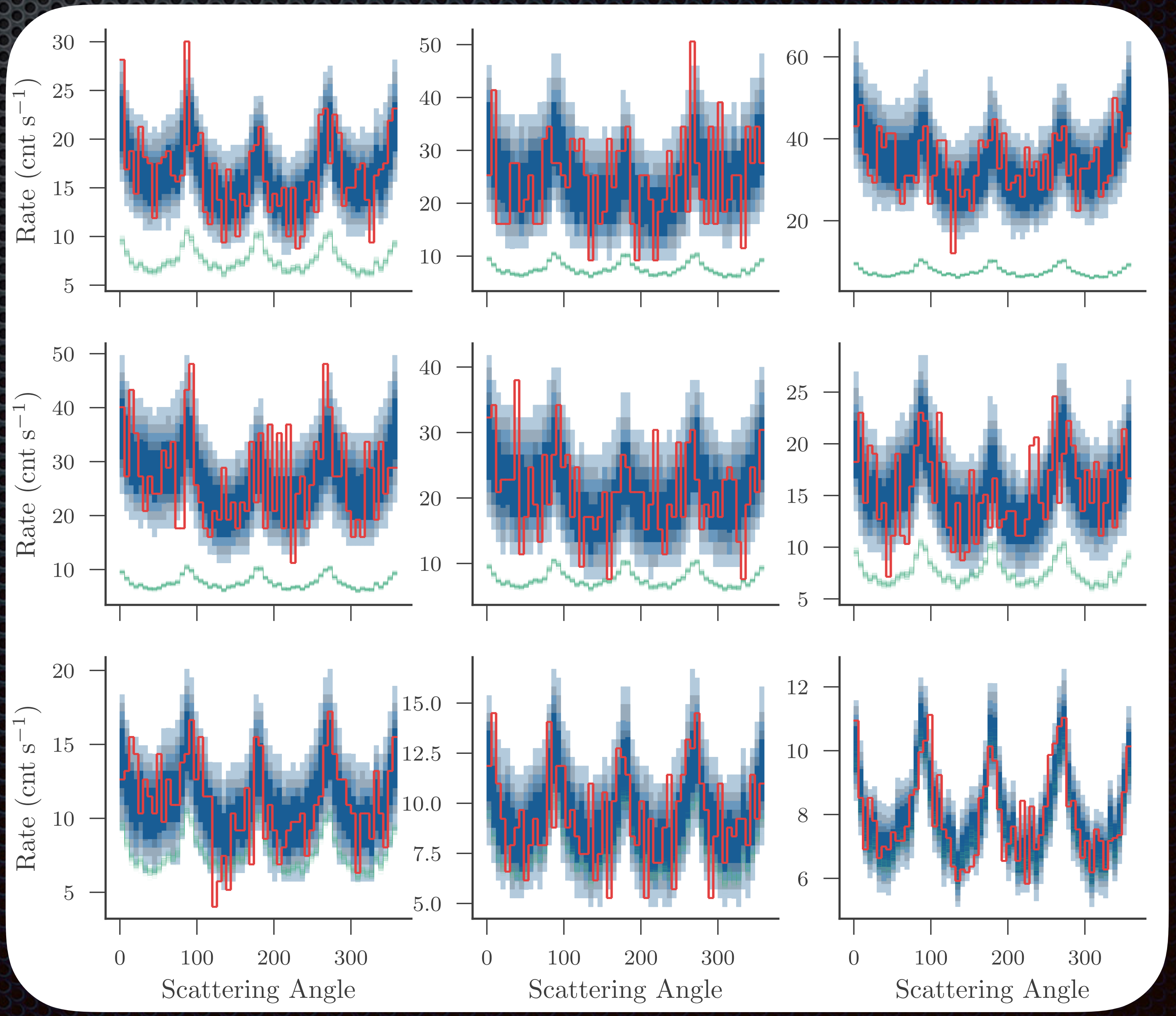
```
    ######################################################################
    # The following methods must be implemented by each plugin
    ######################################################################

    @abc.abstractmethod
    def set_model(self, likelihood_model_instance):
        """
        Set the model to be used in the joint minimization. Must be a LikelihoodModel ins
        """
        pass

    @abc.abstractmethod
    def get_log_like(self):
        """
        Return the value of the log-likelihood with the current values for the
        parameters
        """
        pass

    @abc.abstractmethod
    def inner_fit(self):
        """
        This is used for the profile likelihood. Keeping fixed all parameters in the
        LikelihoodModel, this method minimize the logLike over the remaining nuisance
        parameters, i.e., the parameters belonging only to the model for this
        particular detector. If there are no nuisance parameters, simply return the
        logLike value.
        """
        pass
```


POLAR (ROOT) DATA
POLAR (ROOT) RESPONSE
POLAR (HDF5) DATA
3ML
GBM DATA
GBM RESPONSE


T: -0.2-1.4 s
T: 1.4-1.8 s
T: 1.8-2.4 s
T: 2.4-3.0 s
T: 3.0-3.6 s
T: 3.6-4.8 s
T: 4.8-6.6 s
T: 6.6-8.9 s
T: 8.9-20.0 s
GBM: NaI
POLAR
GBM: BGO
Net rate (cnt s^{-1} keV^{-1})
PHA Energy (keV)


ϕ
$\bar{p}$
ψ
δ
$R_\theta^{j,k}$
n_γ^j
R_γ^j
n_θ^k
n_{pha}^j
$\hat{n}_\theta^k$
$\hat{n}_{\mathrm{pha}}^j$

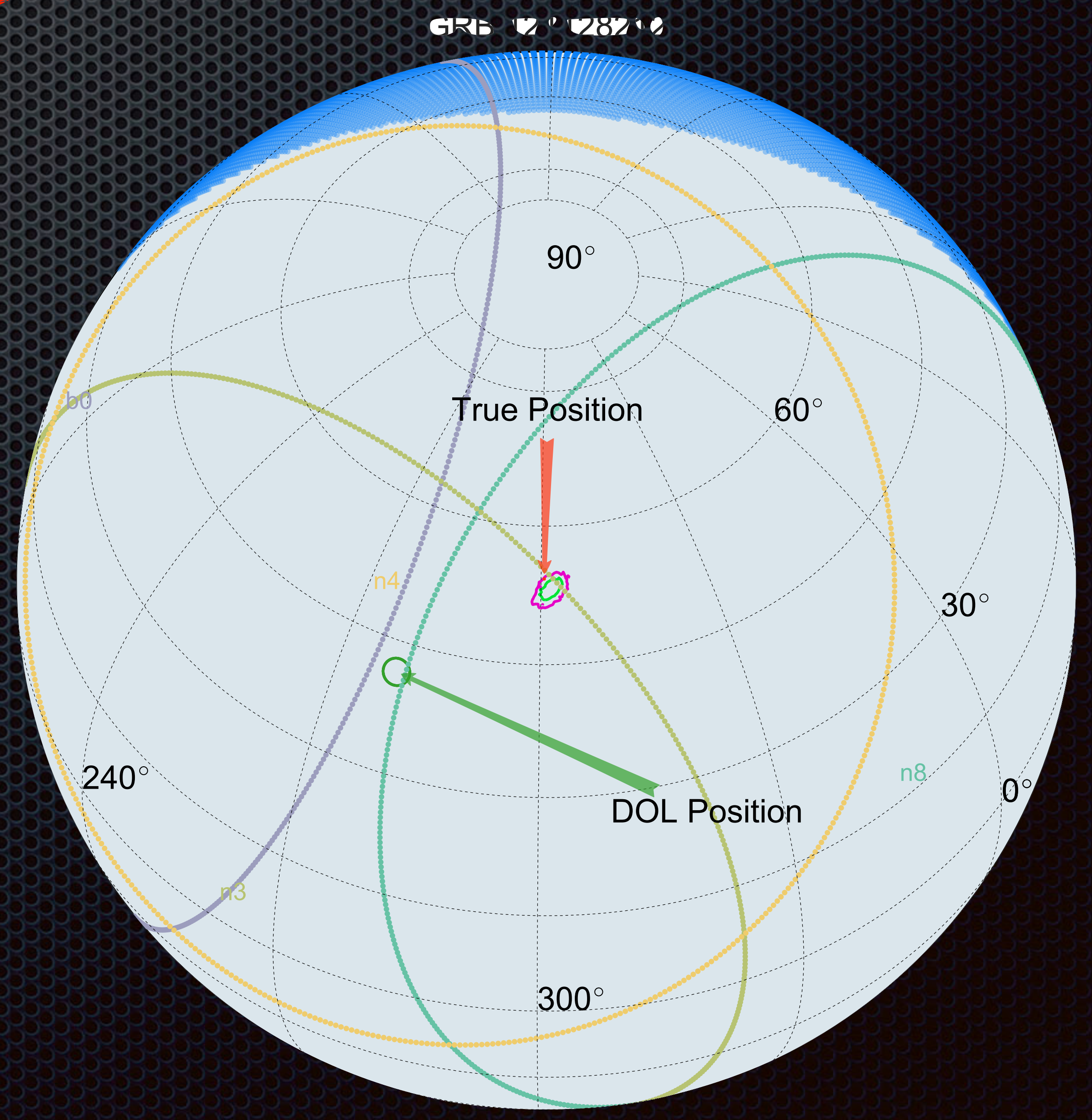

Search or jump to...
Pull requests
Issues
Marketplace
Explore
grburgess / polarpy
Unwatch 2
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
Tools for polar
Edit
Manage topics
135 commits
3 branches
0 releases
1 contributor
GPL-3.0
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
grburgess remove test print
Latest commit c114177 on Aug 24
ci
upload token
4 months ago
polarpy
remove test print
3 months ago
.gitignore
stuff
4 months ago
.travis.yml
try to fix source cmd
4 months ago
LICENSE
Initial commit
4 months ago
README.md
Update README.md
4 months ago
setup.py
setup
4 months ago
README.md
build passing
codecov unknown
Scrutinizer 8.41
maintainability C
polarpy
Tools for polar


Rate (cnt s^{-1})
Scattering Angle

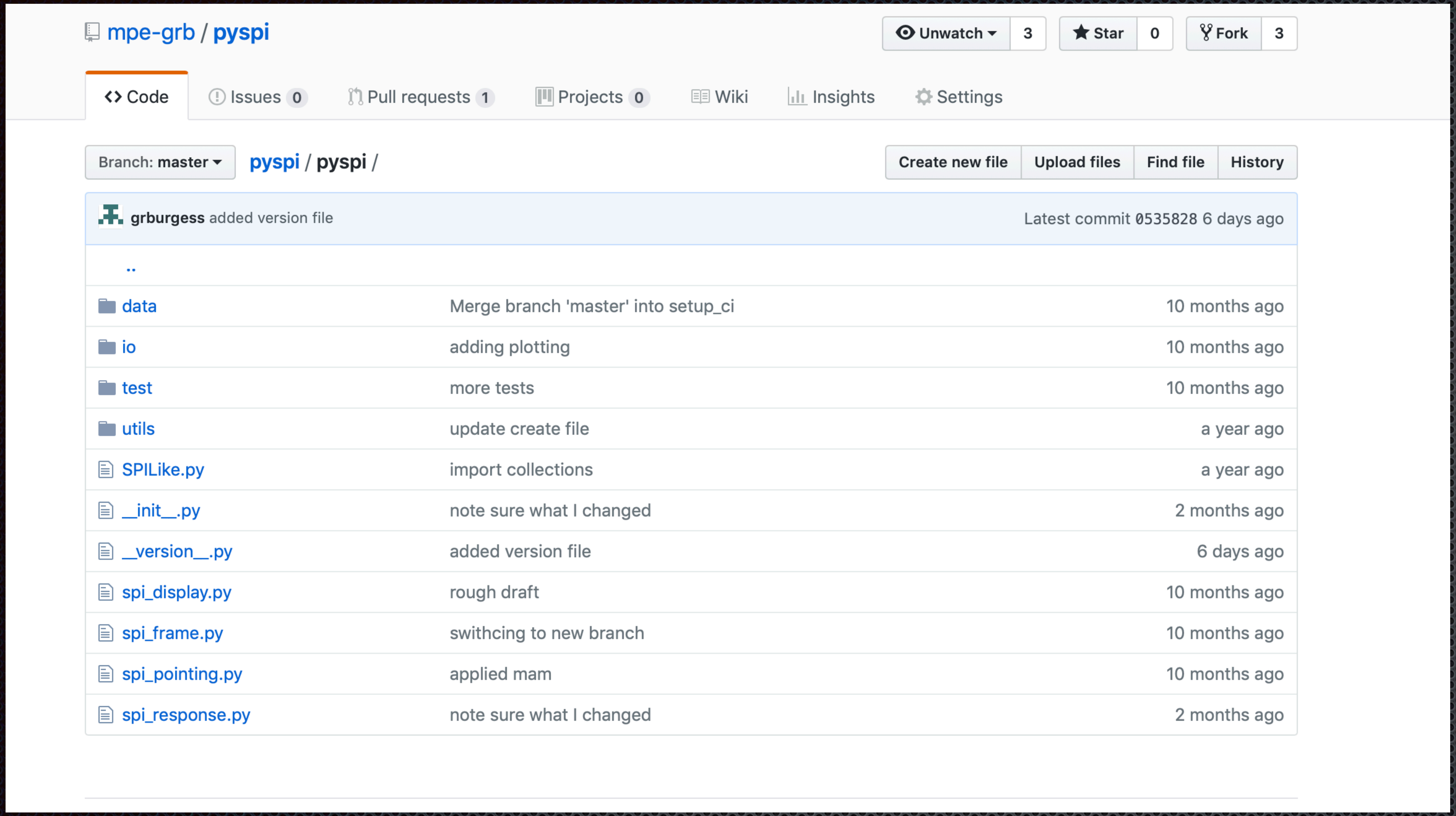
POLAR

```python
class BALROGLike(DispersionSpectrumLike):
    def __init__(self,
                 name,
                 observation,
                 drm_generator=None,
                 background=None,
                 time=0,
                 free_position=True,
                 verbose=True):
        """
        BALROGLike is a general plugin for fitting GBM spectra and locations at the same time


        :param name: plugin name
        :param observation: observed spectrum
        :param drm_generator: the drm generator for this
        :param background: background spectrum
        :param time: time of the observation
        :param free_position: keep the position free
        :param verbose: the verbosity level of the plugin
        """

        self._free_position = free_position



        if drm_generator is None:

            # If a generator is not supplied
            # then make sure that there is a
            # balrog response

            assert isinstance(observation.response, BALROG_DRM), 'The response associated with the observation is not a BALROG'


        else:

            # here we will reset the response
            # this is violating the fact that
            # the response is provate

            balrog_drm =  BALROG_DRM(drm_generator,0.,0.)

            observation._rsp = balrog_drm
```

```python
def set_model(self, likelihoodModel):
    """
    Set the model and free the location parameters


    :param likelihoodModel:
    :return: None
    """

    # set the standard likelihood model

    super(BALROGLike, self).set_model(likelihoodModel)

    # now free the position
    # if it is needed

    if self._free_position:

        if self._verbose:
            print('Freeing the position of %s and setting priors' % self.name)

        for key in self._like_model.point_sources.keys():
            self._like_model.point_sources[key].position.ra.free = True
            self._like_model.point_sources[key].position.dec.free = True

    self._like_model.point_sources[
        key].position.ra.prior = Uniform_prior(
        lower_bound=0., upper_bound=360)
    self._like_model.point_sources[
        key].position.dec.prior = Cosine_Prior(
        lower_bound=-90., upper_bound=90)

    ra = self._like_model.point_sources[key].position.ra.value
    dec = self._like_model.point_sources[key].position.dec.value

    self._rsp.set_location(ra, dec)
```



BALROG


mpe-grb / pyspi
Unwatch 3
Star 0
Fork 3
Code
Issues 0
Pull requests 1
Projects 0
Wiki
Insights
Settings
Branch: master
pyspi / pyspi /
Create new file
Upload files
Find file
History
grburgess added version file
Latest commit 0535828 6 days ago
..
data
Merge branch 'master' into setup_ci
10 months ago
io
adding plotting
10 months ago
test
more tests
10 months ago
utils
update create file
a year ago
SPILike.py
import collections
a year ago
__init__.py
note sure what I changed
2 months ago
__version__.py
added version file
6 days ago
spi_display.py
rough draft
10 months ago
spi_frame.py
swithcing to new branch
10 months ago
spi_pointing.py
applied mam
10 months ago
spi_response.py
note sure what I changed
2 months ago

PYSPI

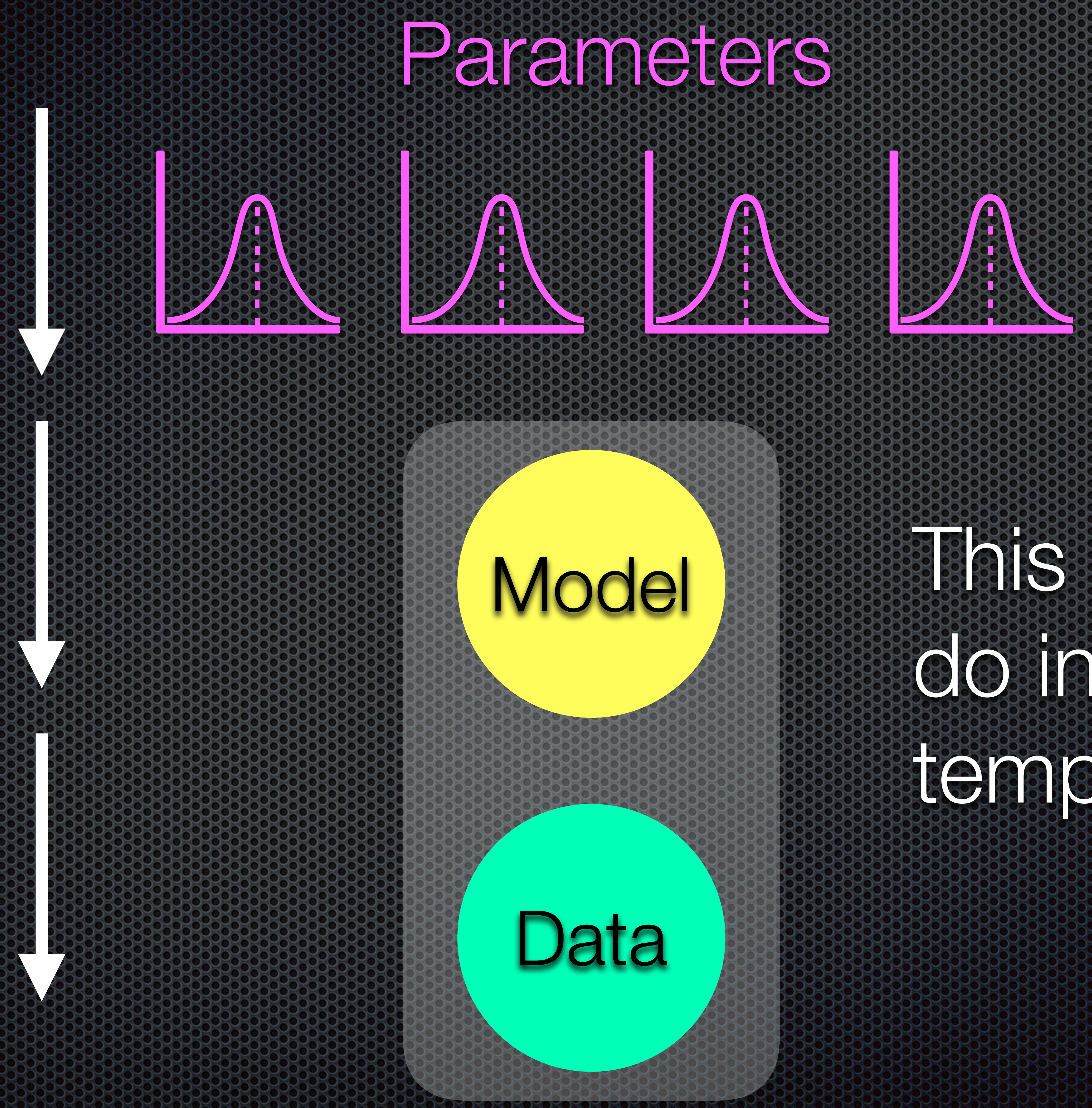
Michael, this is where you go to the Fermipy

The Bayes-ics

- Single-level metropolis-hastings schemes: emcee
- Nested sampling: MULTINEST, PolyChord, D4Nest
- Hierarchical samplers: Stan, Jags, BUGS

But…

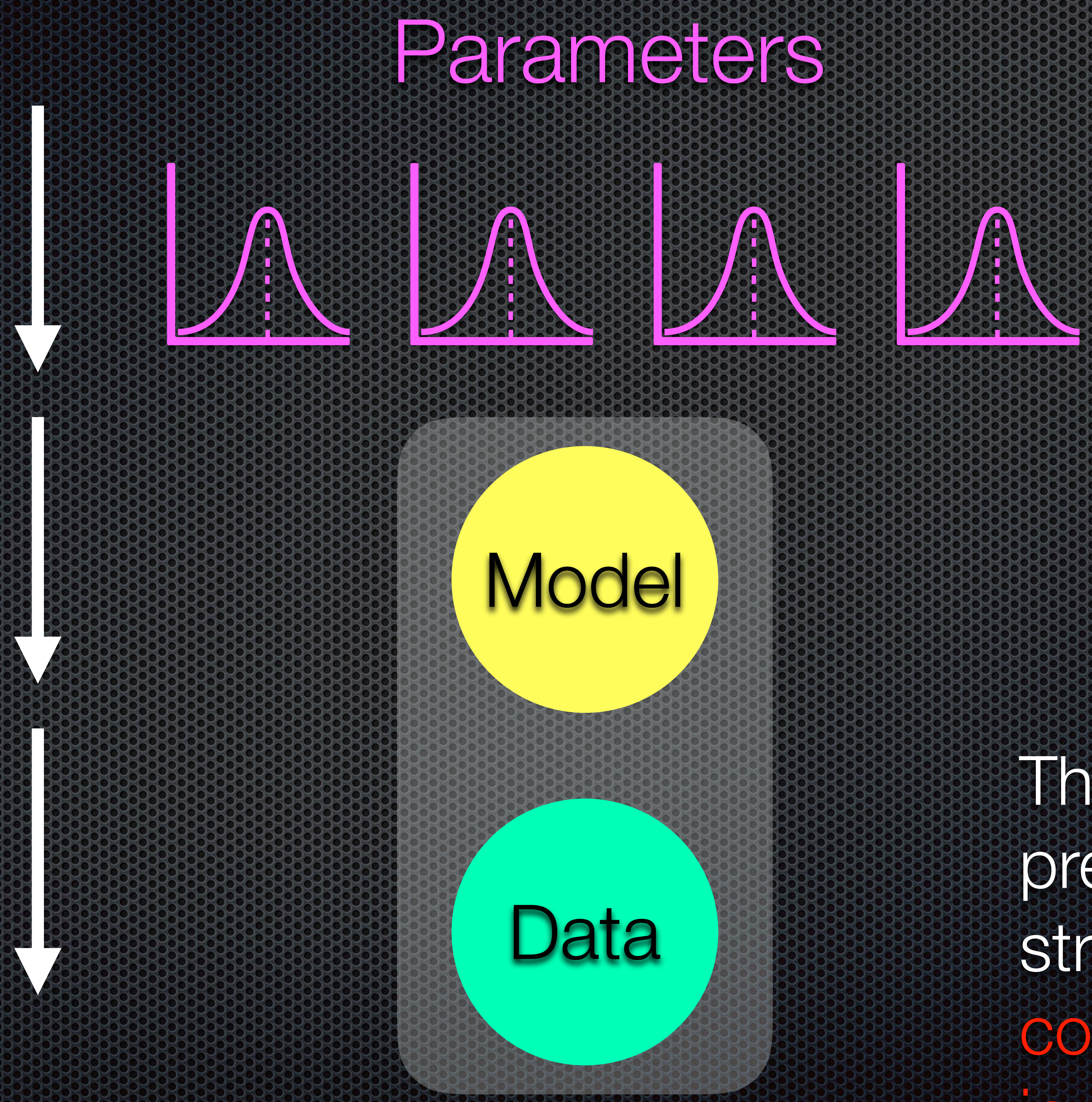
What is a single level model?



This is typically all we will do in our spectral/temporal/spatial fits.

What is a multi level model?

Multi level models are unavoidable in many cases.
Fitting data with uncertainties in both directions? You need a hierarchy.

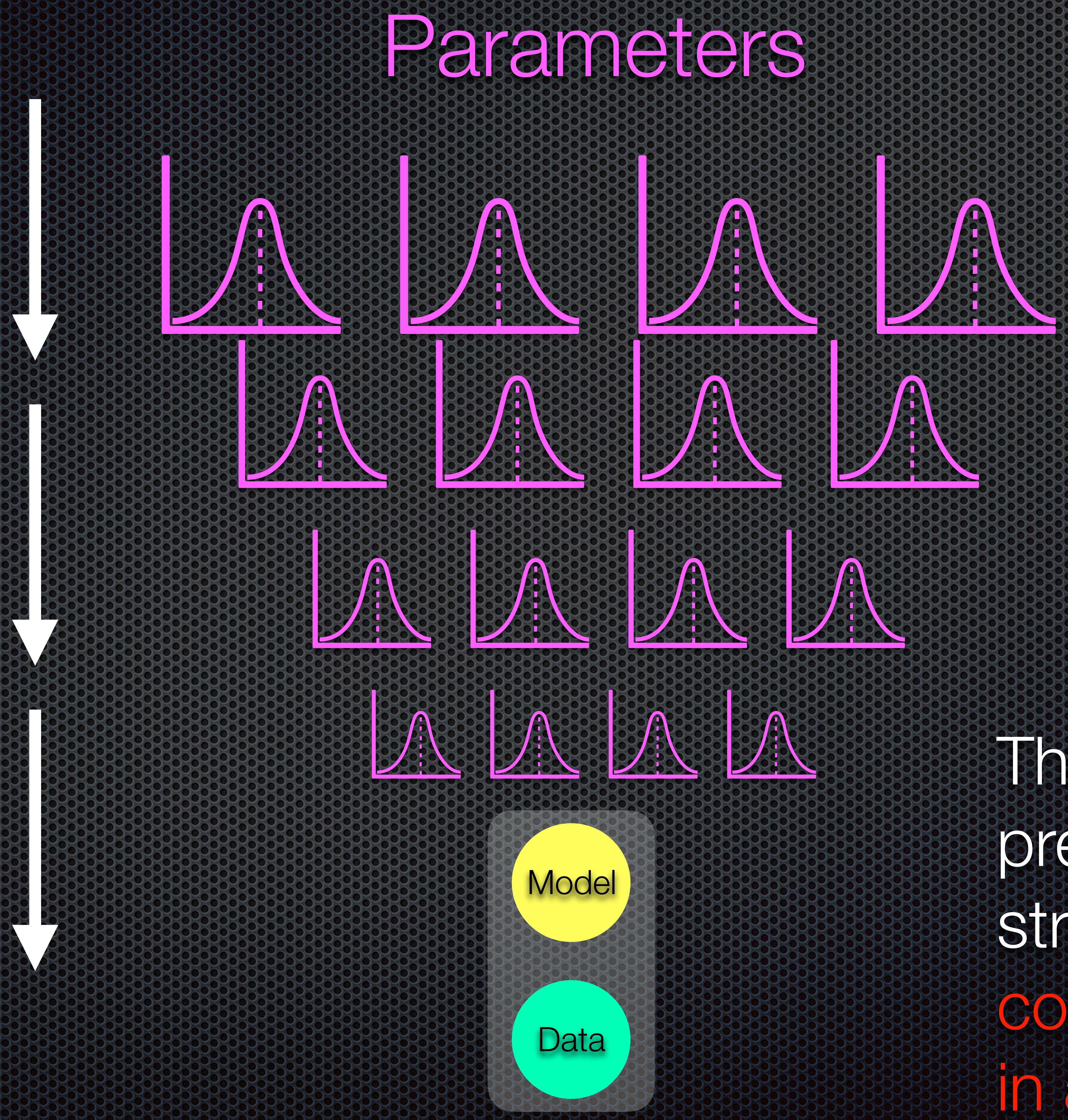


Think stacking but with a preservation of the correlation structure and without the regularity conditions that almost always break in astrophysical problems.

https://github.com/JohannesBuchner/astrostatistics-recipes/blob/master/line/line_fitting.ipynb

What is a multi level model?

Multi level models are unavoidable in many cases.
Fitting data with uncertainties in both directions? You need a hierarchy.



Think stacking but with a preservation of the correlation structure and without the regularity conditions that almost always break in astrophysical problems.

https://github.com/JohannesBuchner/astrostatistics-recipes/blob/master/line/line_fitting.ipynb

Bayes Factors

$\pi(\mathcal{M})$ $\mathcal{M}$ – closed

Bayes Factors

$\pi(\mathscr{M}) \qquad \mathscr{M} - \text{closed}$

Bayes theorem introduces a measure over a model configuration space.

Bayes Factors

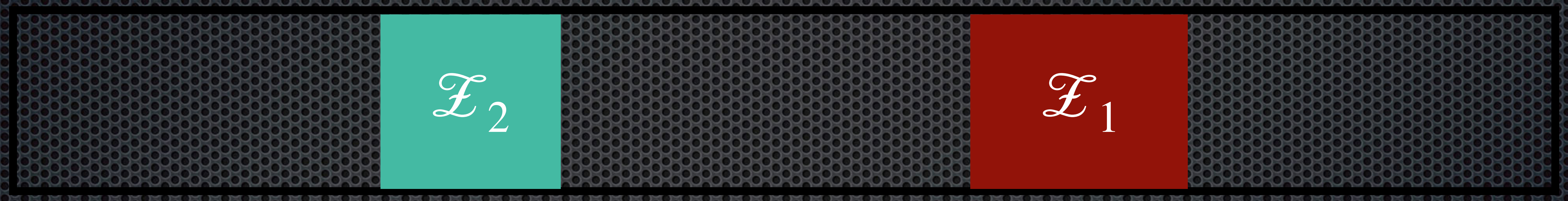
$$\pi(\mathscr{M}) \qquad \mathscr{M} - \text{closed}$$

Bayes theorem introduces a measure over a model configuration space.

Nested models have problems and are sensitive to priors.

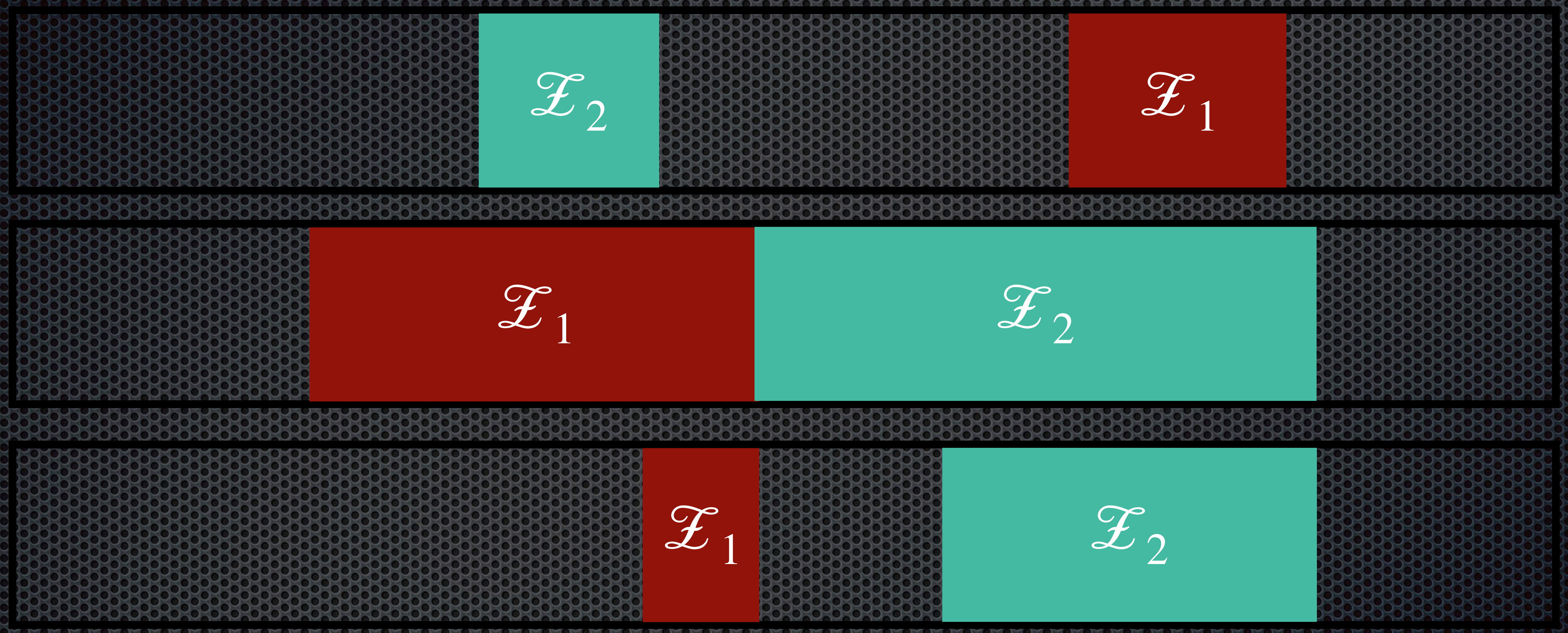
Bayes Factors

$\pi(\mathscr{M}) \qquad \mathscr{M} - \text{closed}$



Priors completely dominate the configurations of Bayes factor ratios

Bayes Factors



All Bayes Factors produce the same parameter inferences!

Information Criteria

BIC: Bayesian Information Criterion

- Model comparison for M-closed for discrete model set
- Statisticians do not know what this is
- It is not going to be a useful for your analysis

Information Criteria

AIC: Akaike Information Criterion

- Asymptotic comparison
- Linear, Gaussian, high-data regime
- Meh… Not much better than the LRT

Information Criteria

DIC: Deviance Information Criterion

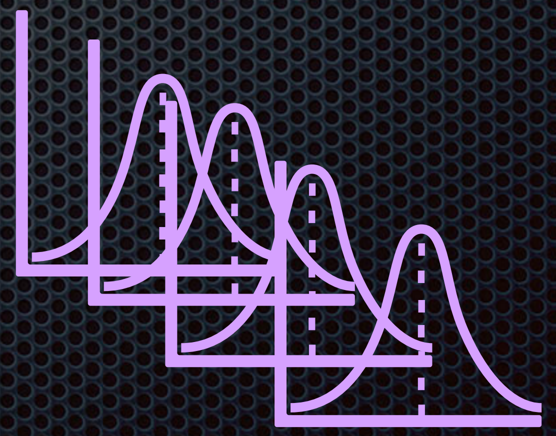
- Uses the posterior to approximate the effective complexity of the model
- Easily obtained as a byproduct of the posterior sampling
- Convergence criteria unknown (and estimated to be very slow)

Posterior Predictive Checks

$$\pi\left(\tilde{y} \,|\, y\right) = \int \mathrm{d}\theta \, \pi\left(\tilde{y} \,|\, \theta\right) \pi\left(\theta \,|\, y\right)$$

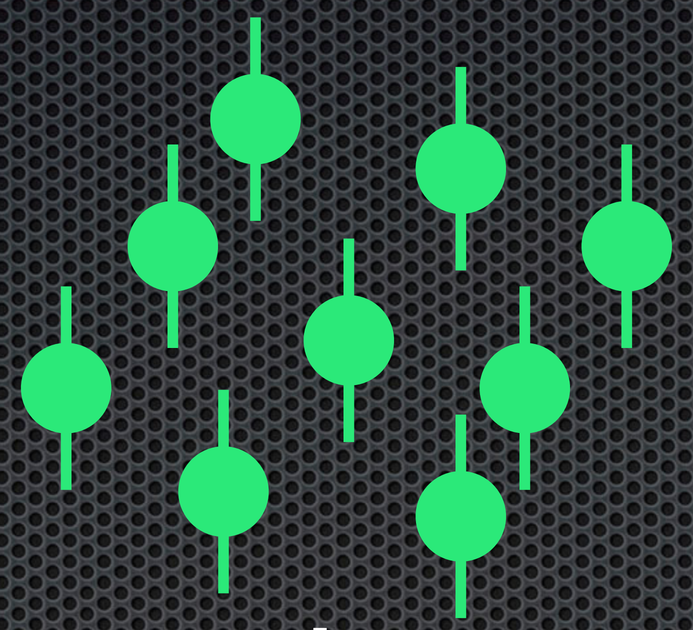
Posterior Predictive Checks

$$\pi\left(\tilde{y} \,|\, y\right) = \int \mathrm{d}\theta \; \pi\left(\tilde{y} \,|\, \theta\right) \pi\left(\theta \,|\, y\right)$$



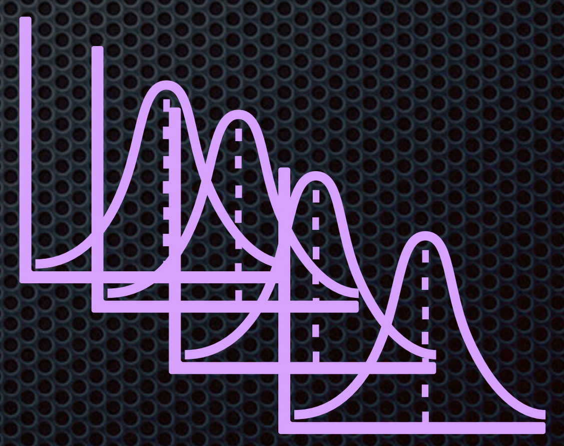
posterior

Posterior Predictive Checks



likelihood

$$\pi\left(\tilde{y}\,|\,y\right) = \int \mathrm{d}\theta \, \pi\left(\tilde{y}\,|\,\theta\right) \pi\left(\theta\,|\,y\right)$$



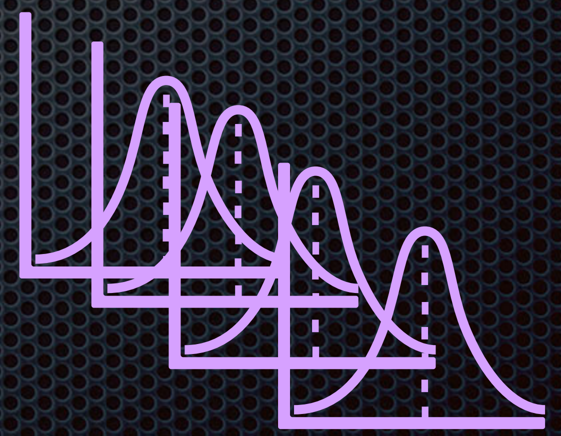
posterior

Posterior Predictive Checks

replicated data

likelihood

$$\pi\left(\tilde{y}\,|\,y\right) = \int \mathrm{d}\theta\, \pi\left(\tilde{y}\,|\,\theta\right) \pi\left(\theta\,|\,y\right)$$



posterior

Posterior Predictive Checks

likelihood

replicated data

$$\pi\left(\tilde{y}\,|\,y\right) = \int \mathrm{d}\theta\, \pi\left(\tilde{y}\,|\,\theta\right) \pi\left(\theta\,|\,y\right)$$

measured data

posterior

Posterior Predictive Checks