Basic structure of Monte Python

Benjamin Audren

École Polytechnique Fédérale de Lausanne

01/04/2014

1. Code Structure
2. Important Modules in (some) detail
3. Usage
4. Practice with Monte Python

Outline

classy - wrapper around CLASS

Cython

- language to **interface** Python with C
- can be used to **speed up** a bottleneck in Python or...
- to wrap an **existing C code** and call it from Python as a normal function

classy - wrapper around CLASS

Cython

- language to **interface** Python with C
- can be used to **speed up** a bottleneck in Python or...
- to wrap an **existing C code** and call it from Python as a normal function

Classy

- classy.pyx **wraps** CLASS modules and some functions (only the ones needed from within Monte Python).
- uses a **dictionary** in place of the **.ini file** to give parameters (file content)
- if no arguments are specified, CLASS **default values** will be used.

classy - wrapper around CLASS

Cython

- language to **interface** Python with C
- can be used to **speed up** a bottleneck in Python or...
- to wrap an **existing C code** and call it from Python as a normal function

Classy

- classy.pyx **wraps** CLASS modules and some functions (only the ones needed from within Monte Python).
- uses a **dictionary** in place of the **.ini file** to give parameters (file content)
- if no arguments are specified, CLASS **default values** will be used.

Last word

You (most probably) don't need to know how `classy.pyx` is written. The only important thing is **its interface**.

General structure of the modules

Files

Input Files

- configuration file: **path to codes on machine**
- parameter file: **parameters, prior range, proposal, experiments**

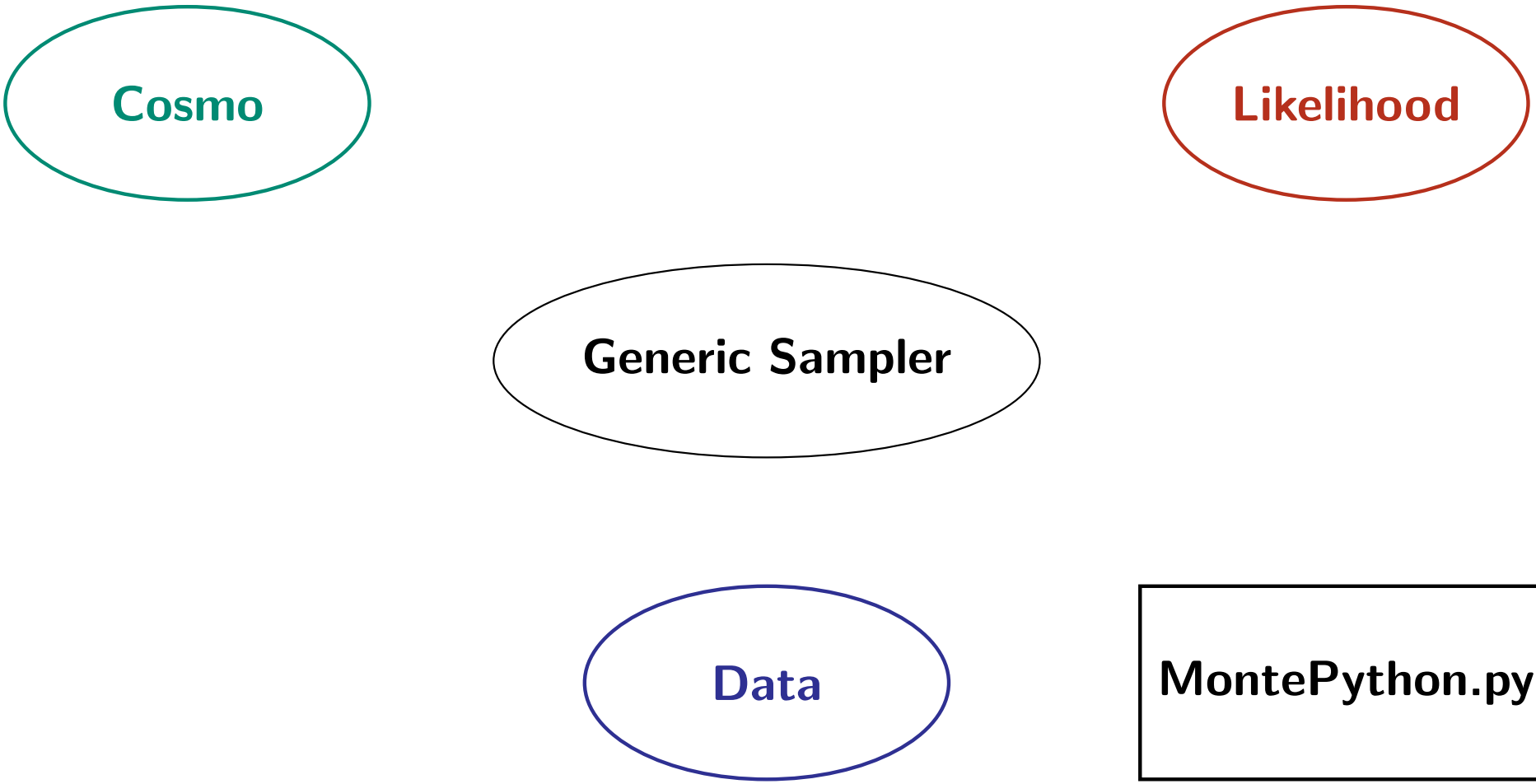
opt covariance matrix, bestfit file

Ouptut

- a folder: **stores every information concerning the run**
- a chain per run: **Markov Chain of a given length**

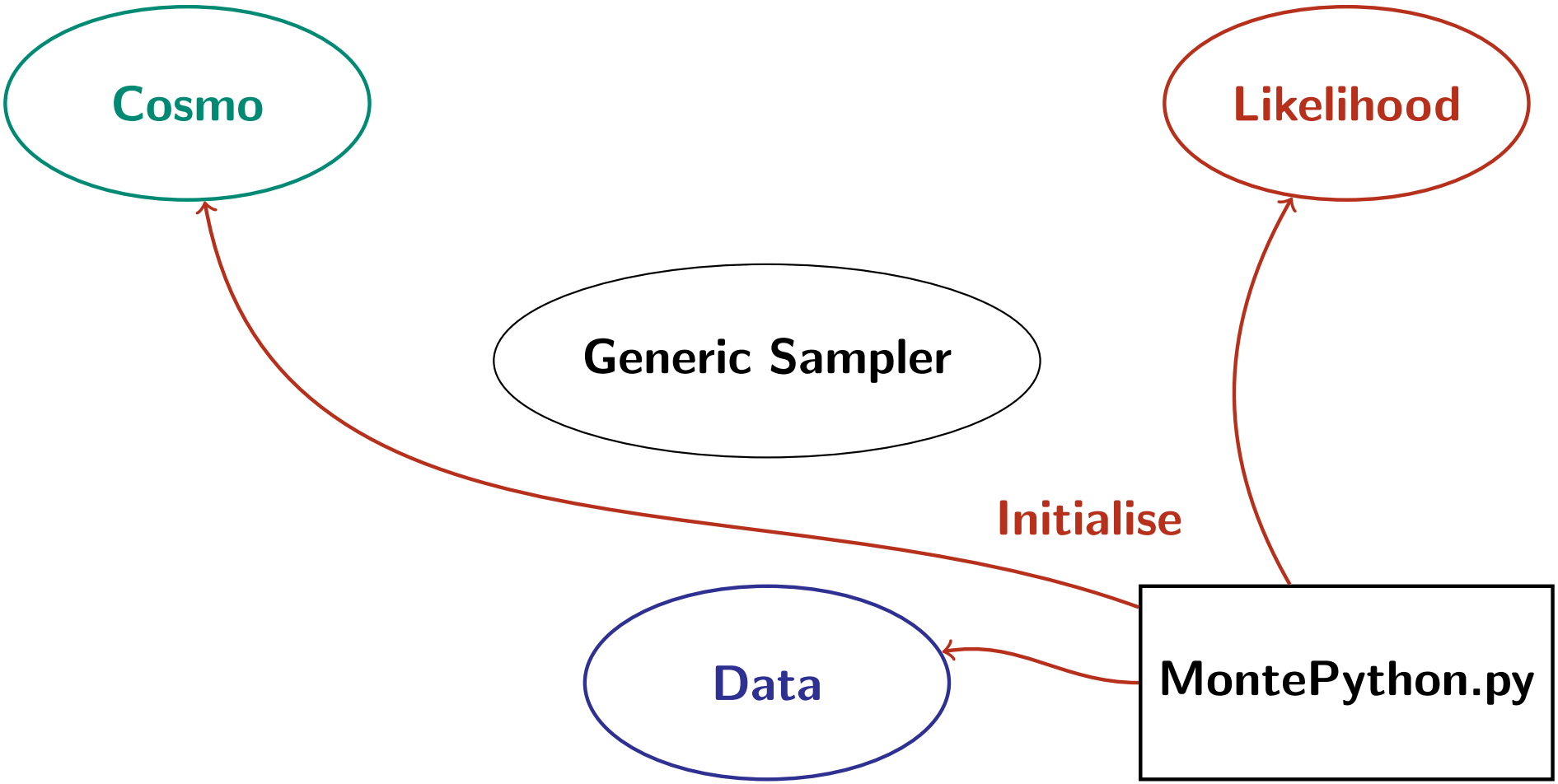
General structure of the modules

A drawing



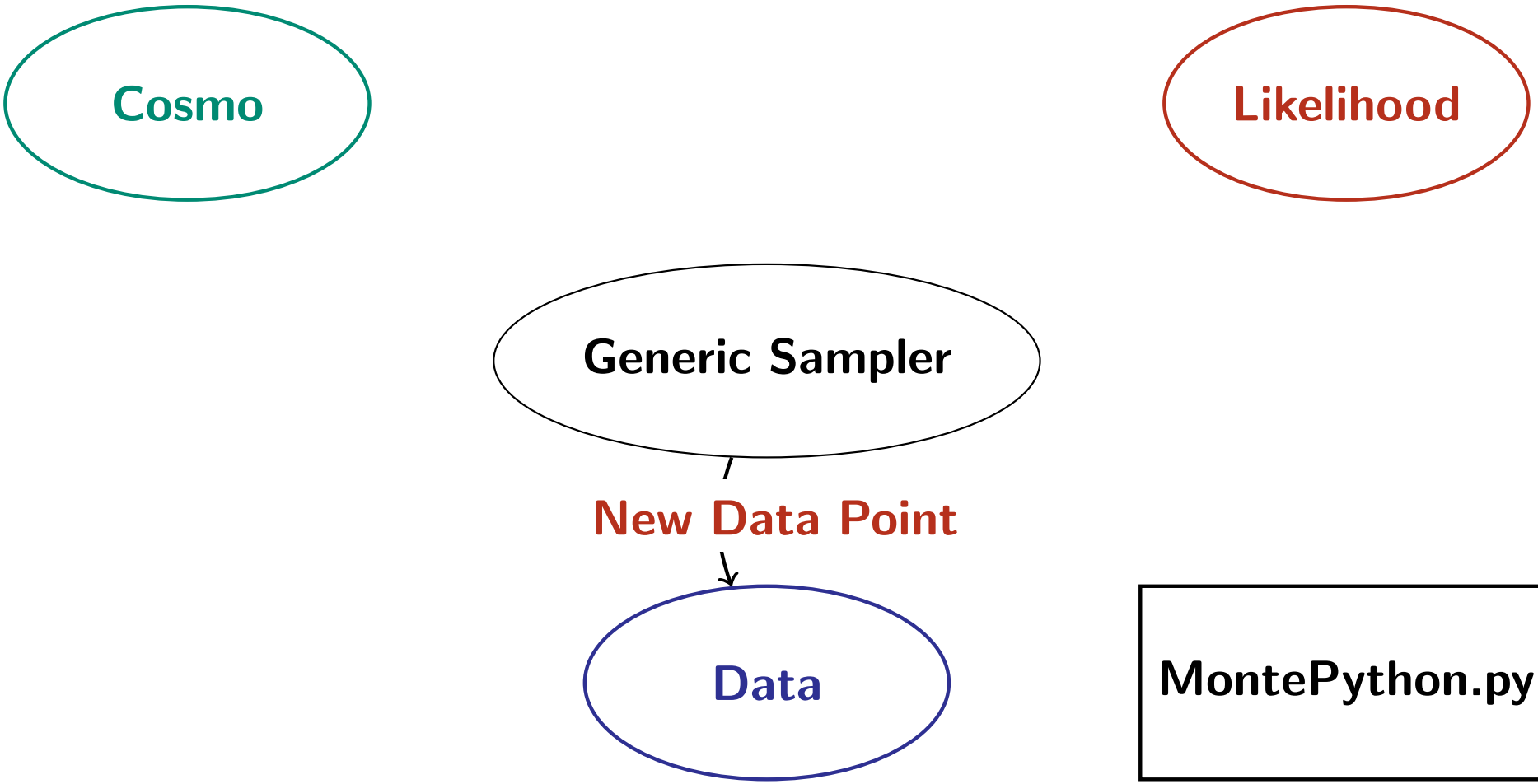
General structure of the modules

A drawing



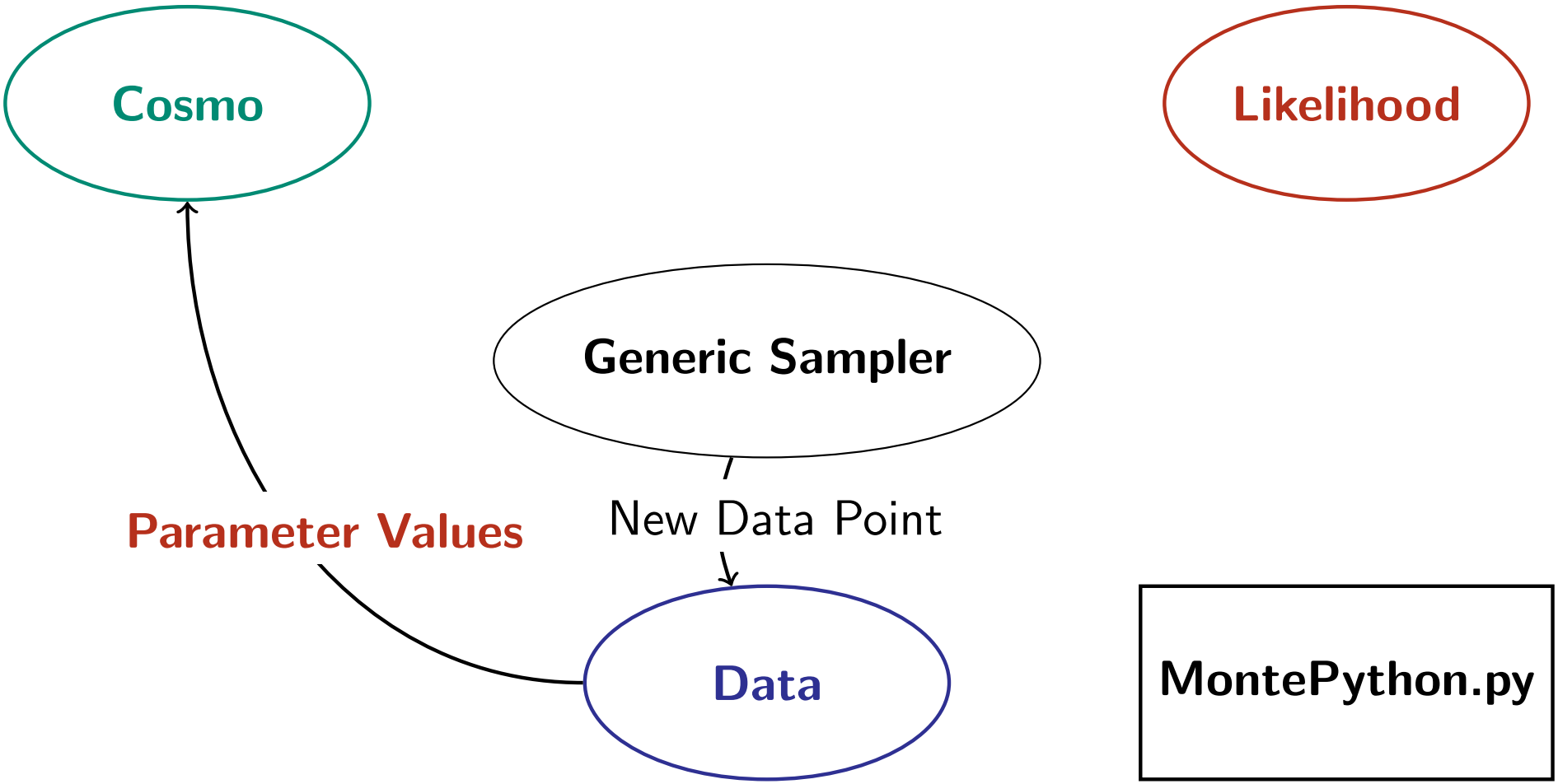
General structure of the modules

A drawing



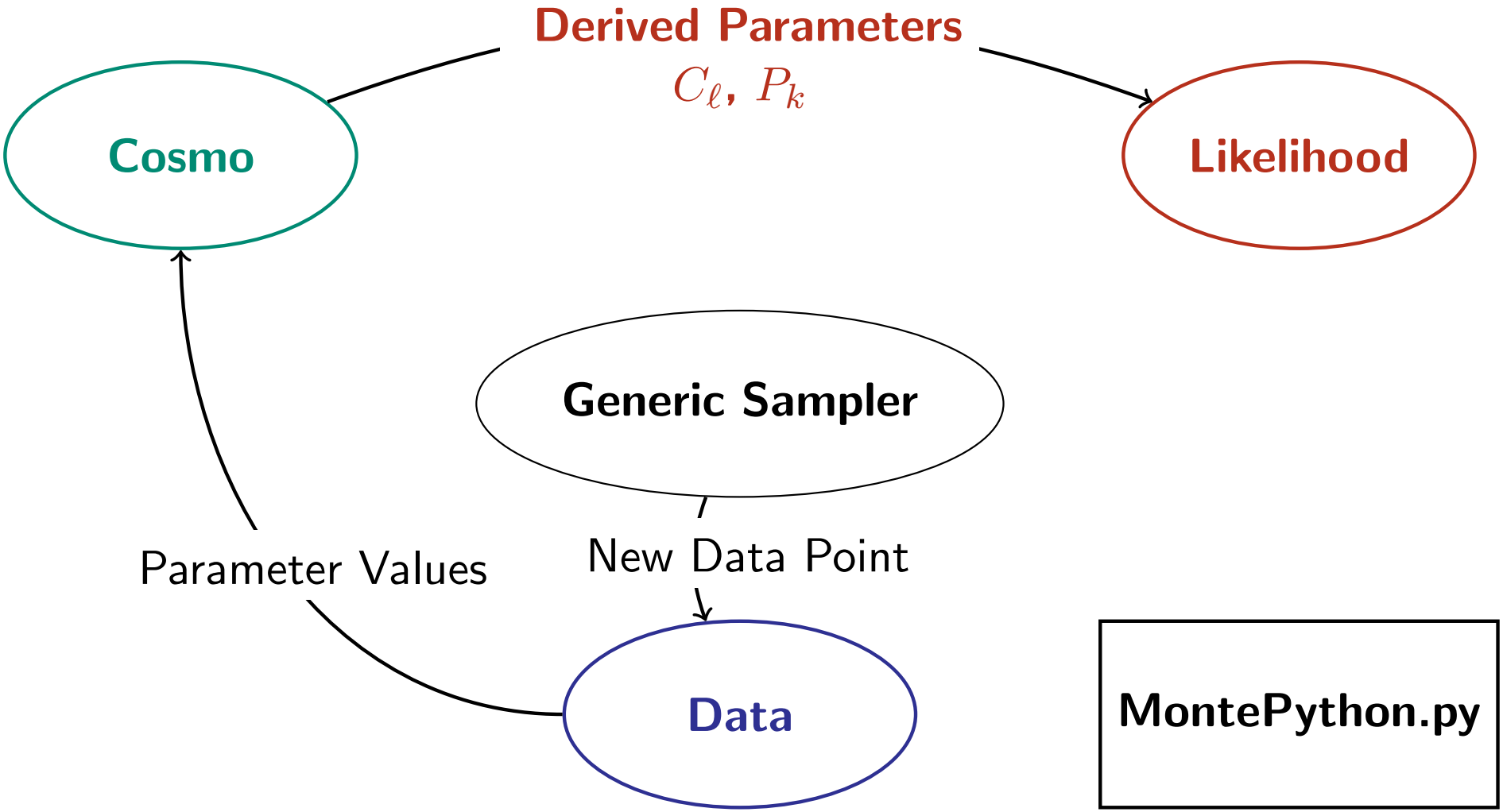
General structure of the modules

A drawing



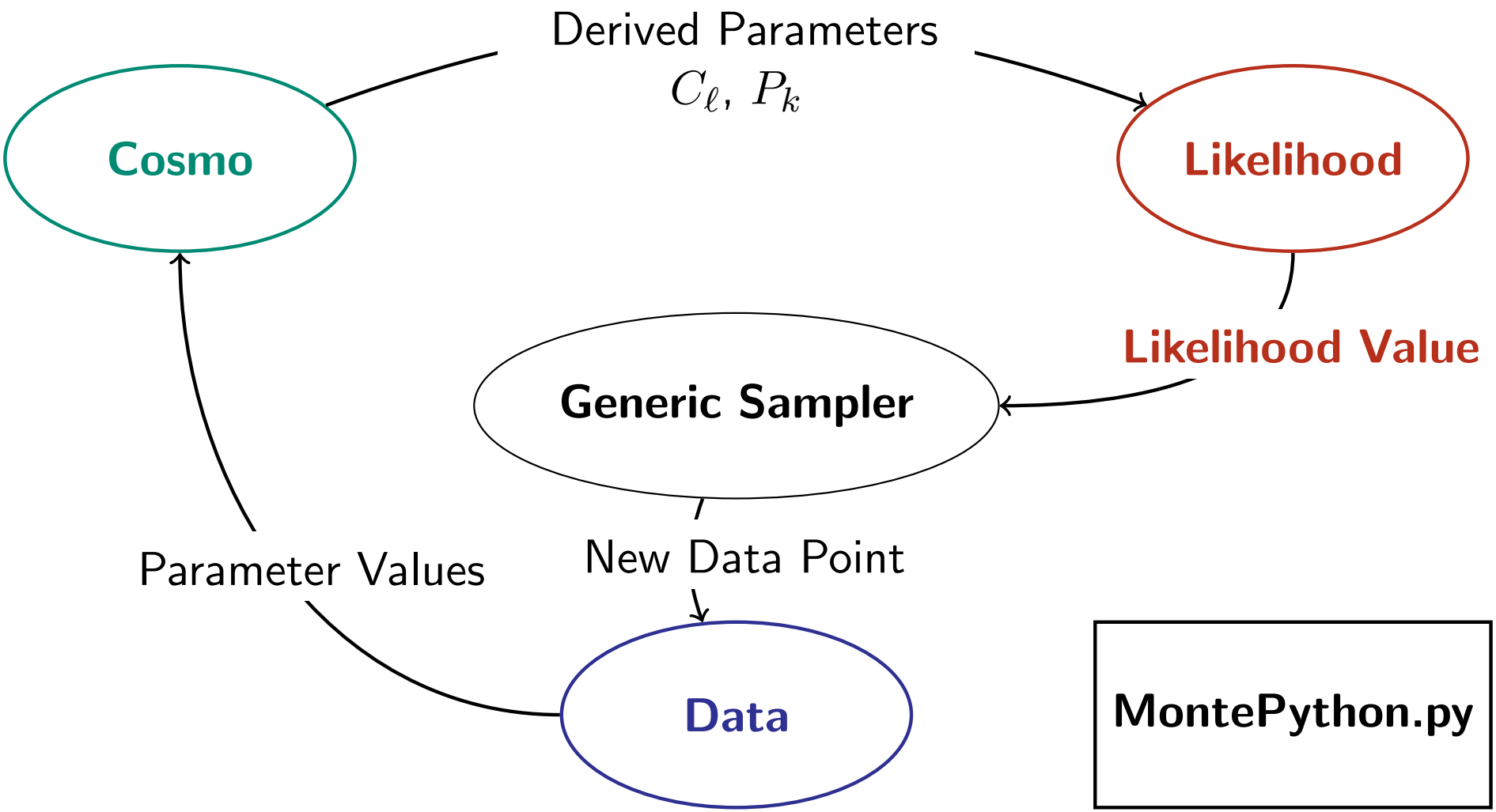
General structure of the modules

A drawing



General structure of the modules

A drawing



Information containers

Foreword: class definitions in capital letters, instances in small.

instances

- **data** initialized in `initialise.py`
- **cosmo** initialized in `initialise.py`
- **hst, bicep2, ...** initialized in `initialise.py`

Information containers

Foreword: class definitions in capital letters, instances in small.

Classes

- **Data** defined in `data.py`
- **Class** defined in `classy.pyx`
- **Likelihood** defined in `likelihood_class.py`

instances

- **data** initialized in `initialise.py`
- **cosmo** initialized in `initialise.py`
- **hst, bicep2, ...** initialized in `initialise.py`

General structure of the modules

Main Modules

- `MontePython` **Simple script launching the code**

General structure of the modules

Main Modules

- `MontePython` Simple script launching the code
- `run` **call initialise, and launch a sampler session**

General structure of the modules

Main Modules

- `MontePython` Simple script launching the code
- `run` call initialise, and launch a sampler session
- `parser_mp` **reads the command line arguments**

General structure of the modules

Main Modules

- `MontePython` Simple script launching the code
- `run` call initialise, and launch a sampler session
- `parser_mp` reads the command line arguments
- `initialise` **creates a cosmological code, Data and likelihoods instances**

General structure of the modules

Main Modules

- `MontePython` Simple script launching the code
- `run` call initialise, and launch a sampler session
- `parser_mp` reads the command line arguments
- `initialise` creates a cosmological code, Data and likelihoods instances
- `data` **defines the Data class, where Parameters are initialized**

General structure of the modules

Main Modules

- `MontePython` Simple script launching the code
- `run` call initialise, and launch a sampler session
- `parser_mp` reads the command line arguments
- `initialise` creates a cosmological code, Data and likelihoods instances
- `data` defines the Data class, where Parameters are initialized
- `sampler` **Generic Sampler calling MCMC, or MultiNest, or CosmoHammer**

General structure of the modules

Main Modules

- `MontePython` Simple script launching the code
- `run` call initialise, and launch a sampler session
- `parser_mp` reads the command line arguments
- `initialise` creates a cosmological code, Data and likelihoods instances
- `data` defines the Data class, where Parameters are initialized
- `sampler` Generic Sampler calling MCMC, or MultiNest, or CosmoHammer
- `likelihood_class` **Likelihood computation for generic ones**

General structure of the modules

Main Modules

- `MontePython` Simple script launching the code
- `run` call initialise, and launch a sampler session
- `parser_mp` reads the command line arguments
- `initialise` creates a cosmological code, Data and likelihoods instances
- `data` defines the Data class, where Parameters are initialized
- `sampler` Generic Sampler calling MCMC, or MultiNest, or CosmoHammer
- `likelihood_class` Likelihood computation for generic ones

Helper Modules

- `analyze` **Computes convergence, posterior from chains**

General structure of the modules

Main Modules

- `MontePython` Simple script launching the code
- `run` call initialise, and launch a sampler session
- `parser_mp` reads the command line arguments
- `initialise` creates a cosmological code, Data and likelihoods instances
- `data` defines the Data class, where Parameters are initialized
- `sampler` Generic Sampler calling MCMC, or MultiNest, or CosmoHammer
- `likelihood_class` Likelihood computation for generic ones

Helper Modules

- `analyze` Computes convergence, posterior from chains
- `io_mp` **Handles I/O, error message**

Outline

Note on the documentation

Note on the documentation

Use it!

Monte Python uses an automatic documentation tool, *sphinx*, that generates a website automatically. `http://baudren.web.cern.ch/baudren/documentation/index.html`

Note on the documentation

Use it!

Monte Python uses an automatic documentation tool, *sphinx*, that generates a website automatically. http://baudren.web.cern.ch/baudren/documentation/index.html

Use the forum!

Go to the issues page https://github.com/baudren/montepython_public/issues, to require a feature, report a bug.

Note on the documentation

Use it!

Monte Python uses an automatic documentation tool, *sphinx*, that generates a website automatically. http://baudren.web.cern.ch/baudren/documentation/index.html

Use the forum!

Go to the issues page https://github.com/baudren/montepython_public/issues, to require a feature, report a bug.

Use the wiki!

https://github.com/baudren/montepython_public/wiki, or https://github.com/lesgourg/class_public/wiki for classy business.

MontePython.py

Role

Convenience script that calls the Monte Python run function.

Initialise I

Main

- **Reads command line, configuration file**

```
29    # Parsing line argument
30    command_line = parser_mp.parse(custom_command)
31
32    # Recovering the local configuration
33    path = recover_local_path(command_line)
```

Initialise I

Main

- Reads command line, configuration file
- **Creates a data instance**

```
56          data = Data(command_line, path)
```

Initialise I

Main

- Reads command line, configuration file
- Creates a data instance
- **Initializes the cosmological module**

```
72    # Loading up the cosmological backbone. For the moment, only
          CLASS has been
73    # wrapped.
74    cosmo = recover_cosmological_module(data)
```

Data I

Defining a data class

- **Initialization**

```
35 class Data(object):
36     """
37     Store all relevant data to communicate between the different
           modules.
38
39     """
40
41     def __init__(self, command_line, path):

136         self.cosmo_arguments = {}

145         self.mcmc_parameters = od()
```

Data I

Defining a data class

- Initialization
- **Fill in parameter information**

```
195         # Read from the parameter file to fill properly the
                mcmc_parameters
196         # dictionary.
197         self.fill_mcmc_parameters()
```

Data I

Defining a data class

- Initialization
- Fill in parameter information
- **Log parameter file if needed**

Data II

Defining a data class

- **Initialization of likelihood (dynamical)**

```
338         for elem in self.experiments:

343             # ... import easily the likelihood.py program
344             exec "from likelihoods.%s import %s" % (
345                 elem, elem)

350             exec "self.lkl['%s'] = %s('%s/%s.data',\
351                 self,command_line)" % (
352                 elem, elem, folder, elem)
```

Data II

Defining a data class

- Initialization of likelihood (dynamical)

Why so complicated

No hard coded likelihood! The code does not know the names: **no need to modify the core code to add a new likelihood**

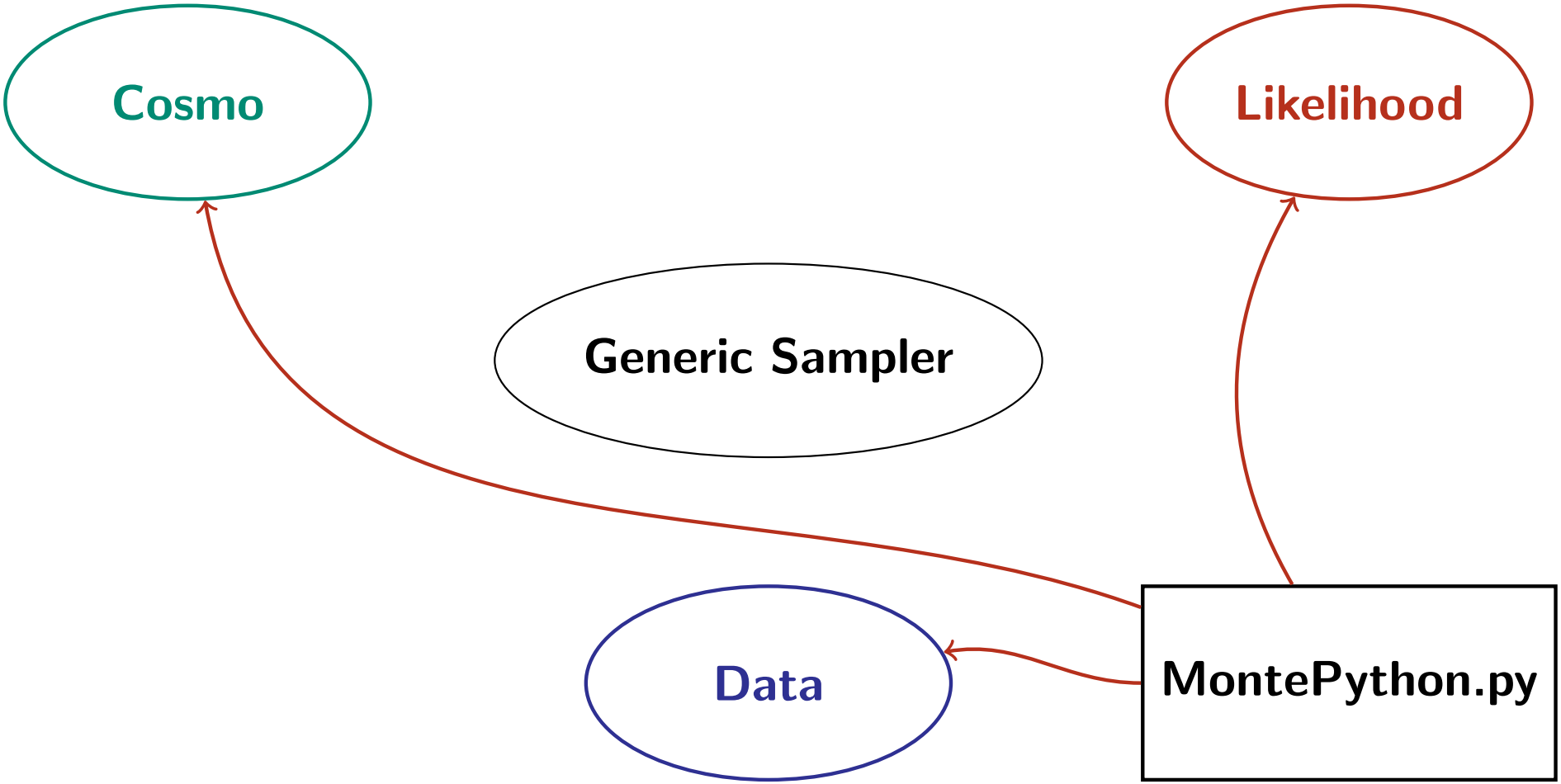
Data III

essential function

`get_mcmc_parameters` returns the list of desired parameters.

- `get_mcmc_parameters(['varying'])`
- `get_mcmc_parameters(['cosmo', 'nuisance'])`
- `get_mcmc_parameters(['cosmo', 'varying'])`

Recap Initialisation



Sampler I

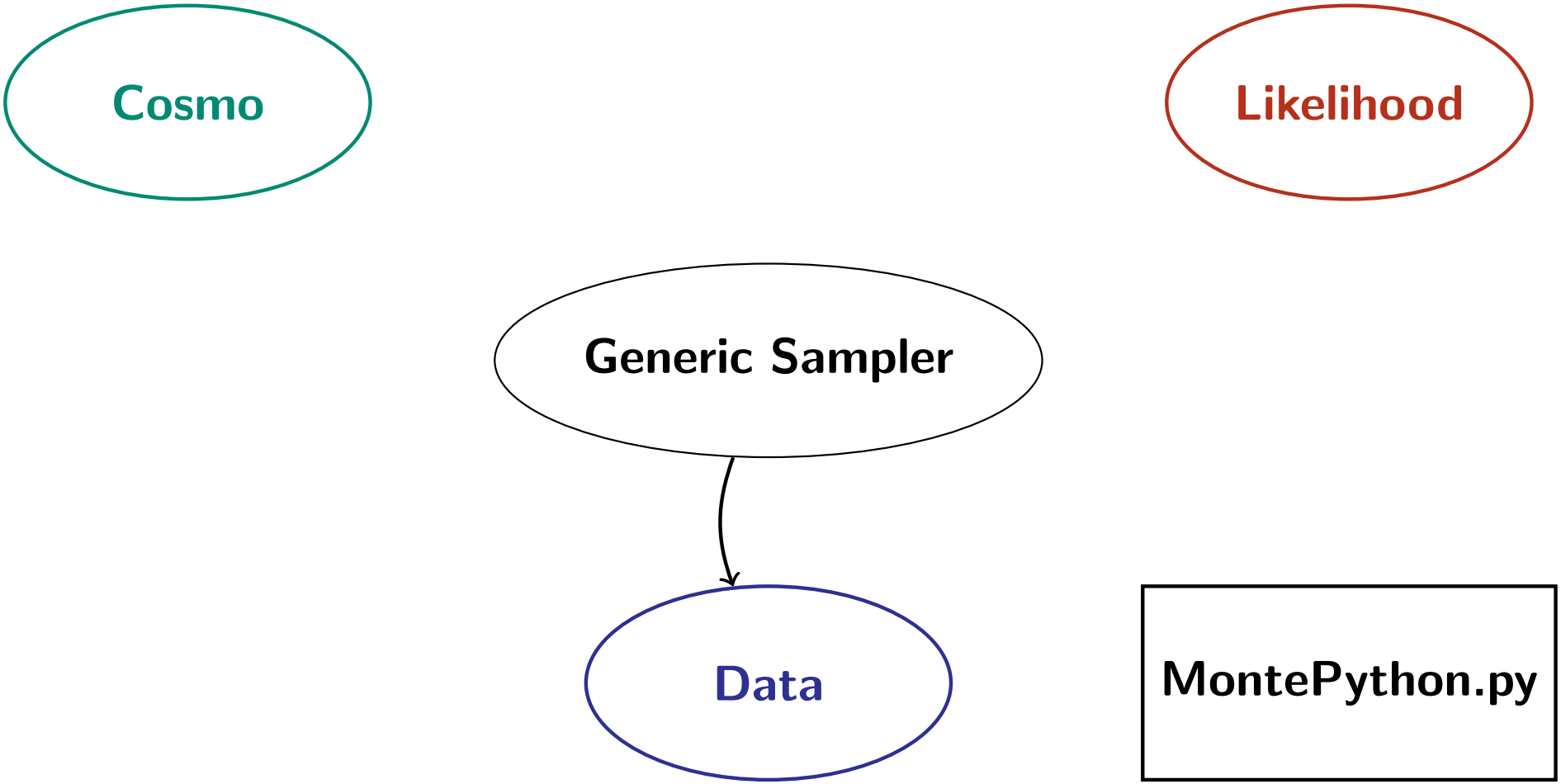
Generic helper functions

- `compute_lkl(cosmo, data)`
- `get_covariance_matrix(data, command_line)`

Role

calling the sampler specified via the **command line**

Choosing a new point



Sampler II

Get new position

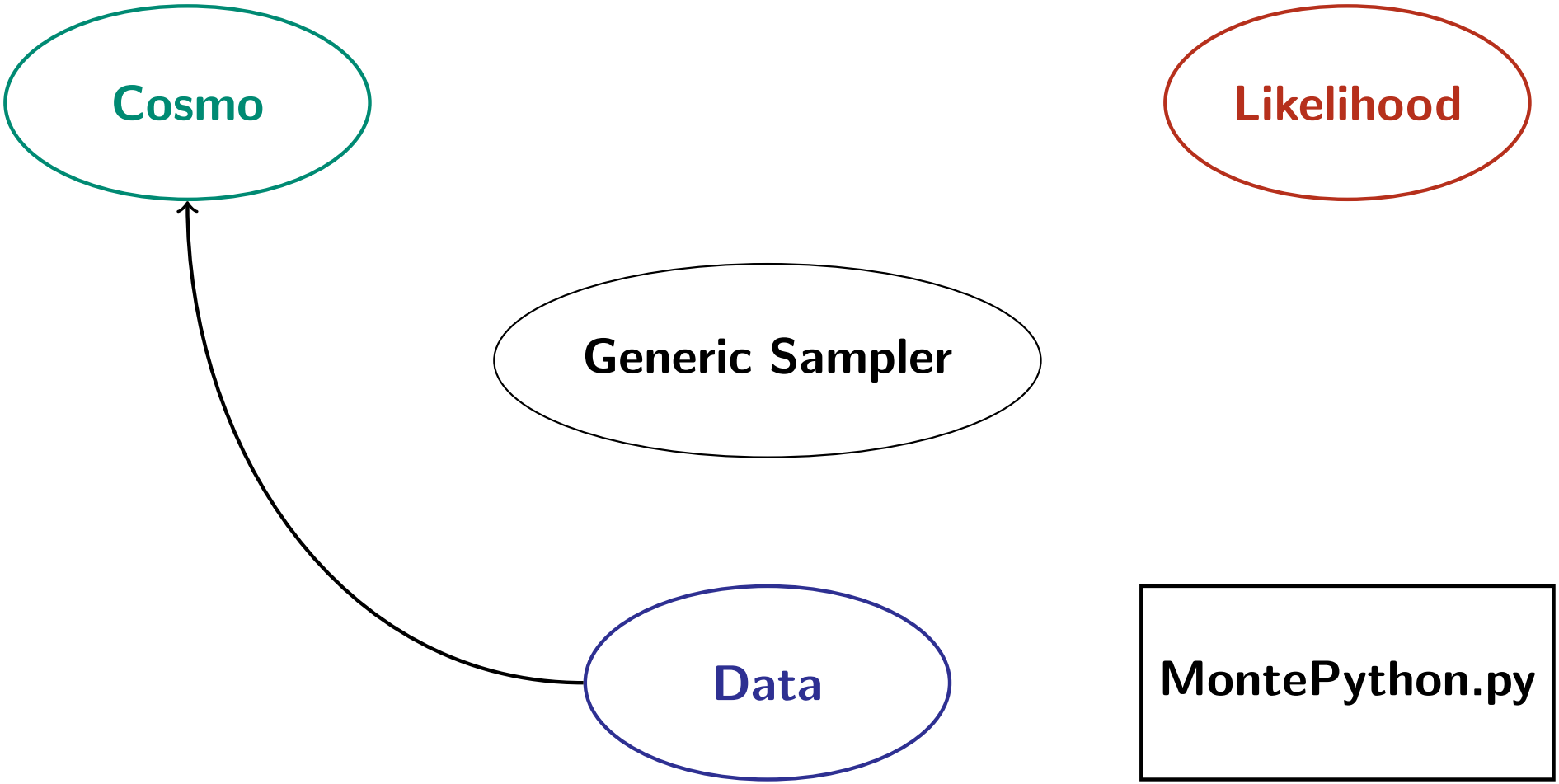
Sampler

How to choose a new point?

- basic eigen-values/vector decomposition
- Cholesky decomposition (Planck) (`-j fast`)
- Nested Sampling with MultiNest (`-m NS`)
- Emcee with Cosmo Hammer (`-m CH`)

Compute Likelihood

Set the cosmo



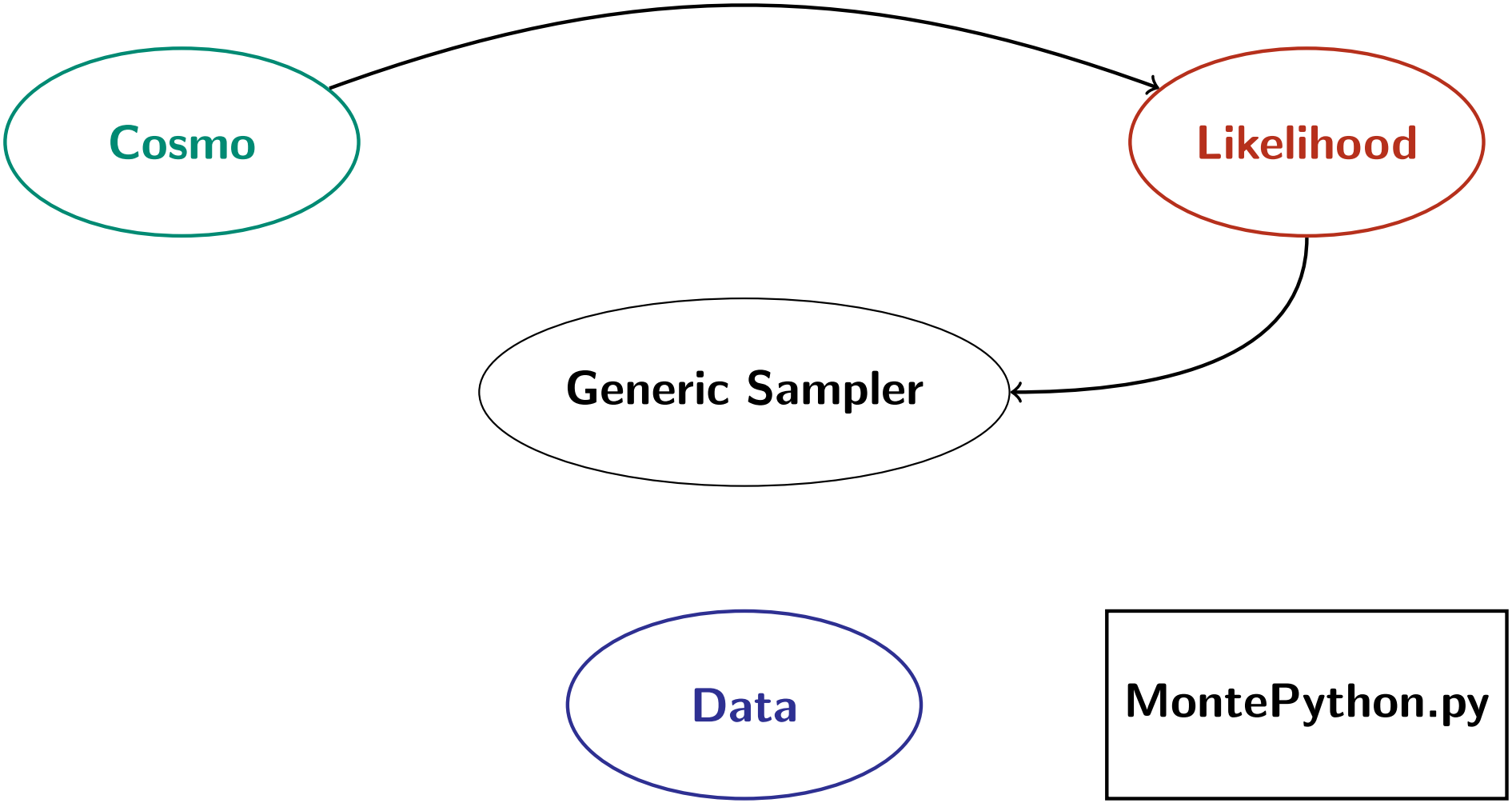
Sampler III

Compute likelihood

```
334 def compute_lkl(cosmo, data):

370     if ((data.need_cosmo_update) or
371             (not cosmo.state) or
372             (data.jumping_factor == 0)):
373
374         # Prepare the cosmological module with the new set of
                parameters
375         cosmo.set(data.cosmo_arguments)

390         try:
391             cosmo.compute(["lensing"])
392         except CosmoComputationError:
393             return data.boundary_loglike
394         except CosmoSevereError, message:
395             print str(message)
396             raise io_mp.CosmologicalModuleError(
397                 "Something went wrong when calling CLASS")
```

Compute Likelihood

For each likelihood



Sampler III

Compute likelihood

```
404     loglike = 0
```

```
410     for likelihood in data.lkl.itervalues():
411         if likelihood.need_update is True:
412             value = likelihood.loglkl(cosmo, data)
413             # Storing the result
414             likelihood.backup_value = value
415         # Otherwise, take the existing value
416         else:
417             value = likelihood.backup_value
418         loglike += value
```

. . . fiducial . . .

```
446     return loglike
```

Sampler IV

Get the covariance matrix

Main ideas

- **stores values without scale factors for numerical reason**

Sampler IV

Get the covariance matrix

Main ideas

- stores values without scale factors for numerical reason
- **automatic handling of parameters**

Sampler IV

Get the covariance matrix

Main ideas

- stores values without scale factors for numerical reason
- automatic handling of parameters
- **computes eigen vectors, values, and Cholesky decomposition**

Likelihood class

Heavily object oriented

in `likelihood_class.py` are defined:

- the basic `Likelihood` class (parent of all others)

Likelihood class

Heavily object oriented

in `likelihood_class.py` are defined:

- the basic `Likelihood` class (parent of all others)
- `Likelihood_newdat` (standard format)

Likelihood class

Heavily object oriented

in `likelihood_class.py` are defined:

- the basic `Likelihood` class (parent of all others)
- `Likelihood_newdat` (standard format)
- `Likelihood_clik` (Planck, WMAP)

Likelihood class

Heavily object oriented

in `likelihood_class.py` are defined:

- the basic `Likelihood` class (parent of all others)
- `Likelihood_newdat` (standard format)
- `Likelihood_clik` (Planck, WMAP)
- `Likelihood_mpk` (WiggleZ, Euclid)

Likelihoods

Implementation

in the `likelihoods` folder, always the following structure:

- `likelihoods/something/__init__.py` and

Likelihoods

Implementation

in the `likelihoods` folder, always the following structure:

- `likelihoods/something/__init__.py` and
- `likelihoods/something/something.data`

Likelihoods

Implementation

in the `likelihoods` folder, always the following structure:

- `likelihoods/something/__init__.py` and
- `likelihoods/something/something.data`
- alway **inherit** at least from: `Likelihood`

Analyze

Convergence Computation

Gelman-Rubin Diagnostic:
variance between chains $=$ variance within chains
This gives a number R, that must be < 0.01.
Beware of this number computed for a single file!

Analyze

Convergence Computation

Gelman-Rubin Diagnostic:
variance between chains $=$ variance within chains
This gives a number R, that must be < 0.01.
Beware of this number computed for a single file!

Plotting

pdf output (or png): triangle plot and 1-dimensional **marginalized posterior** and **Mean likelihood** (visual indication of convergence)

Analyze

Convergence Computation

Gelman-Rubin Diagnostic:
variance between chains = variance within chains
This gives a number R, that must be < 0.01.
Beware of this number computed for a single file!

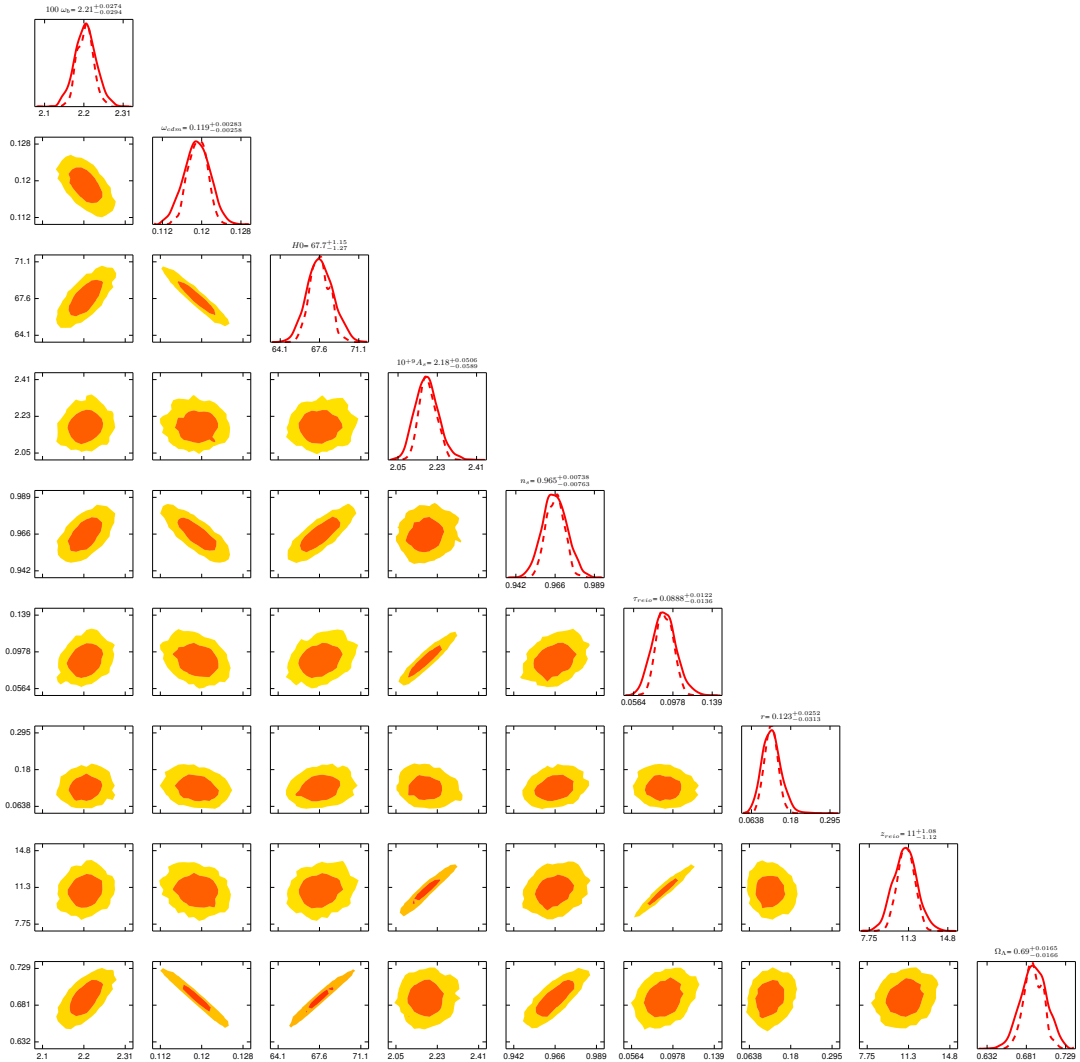
Plotting

pdf output (or png): triangle plot and 1-dimensional **marginalized posterior** and **Mean likelihood** (visual indication of convergence)

Output Files

`.covmat`, `.v_info`, `h_info`, `.bestfit`, `.log`

Analyze



Analyze

horizontal info

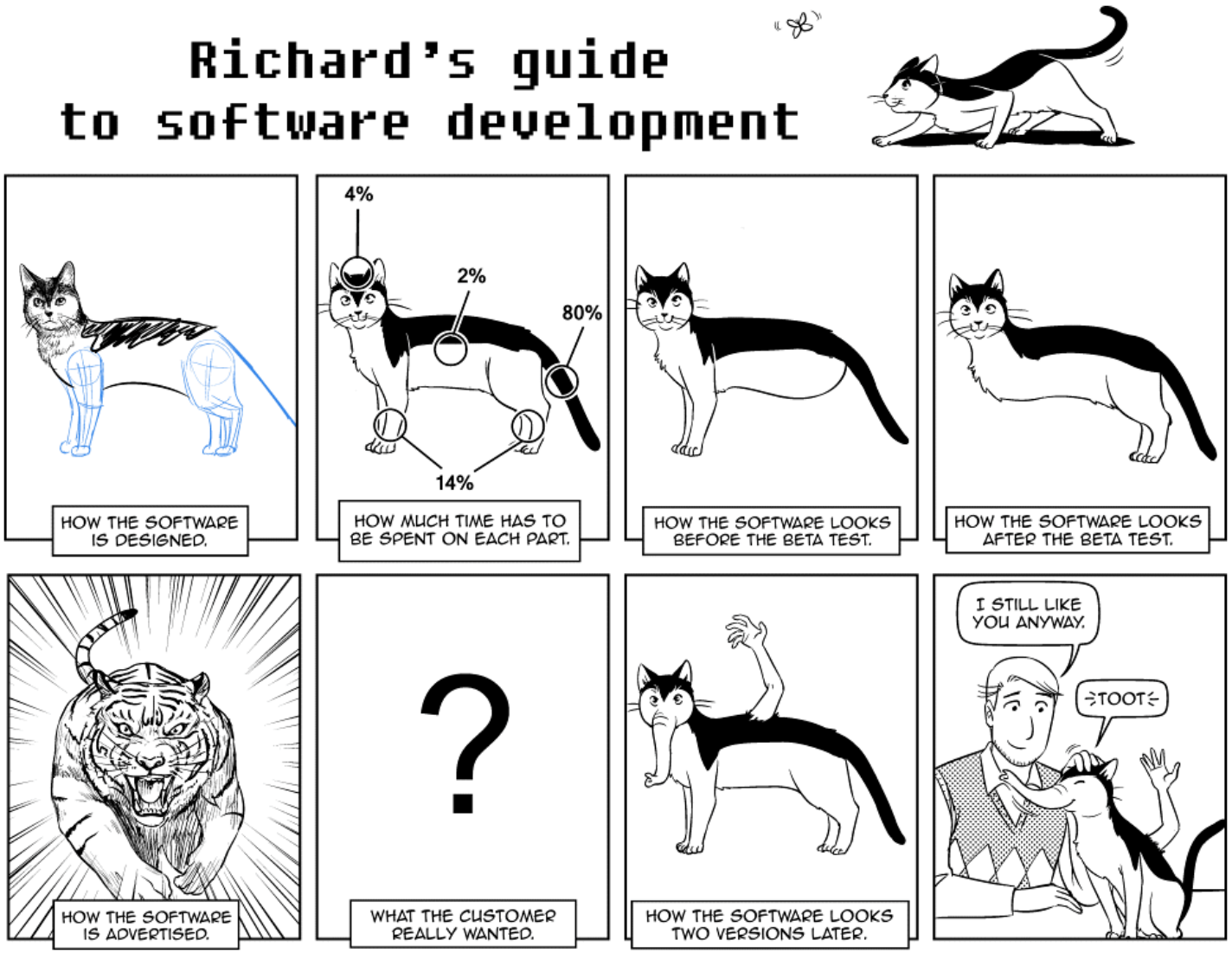
```
param names   :   Omega_L           h
R-1 values :   0.000055   0.000015
Best Fit     :   7.918934e-01   7.339652e-01
mean         :   7.781465e-01   7.308724e-01
sigma        :   6.793778e-02   2.382595e-02

1-sigma -    : -5.799105e-02  -2.310380e-02
1-sigma +    :   7.788452e-02   2.454811e-02
2-sigma -    : -1.418414e-01  -4.808685e-02
2-sigma +    :   1.333294e-01   4.764994e-02
3-sigma -    : -2.379013e-01  -7.257030e-02
3-sigma +    :   1.779086e-01   6.790438e-02

1-sigma >    :   7.201555e-01   7.077686e-01
1-sigma <    :   8.560310e-01   7.554205e-01
2-sigma >    :   6.363051e-01   6.827856e-01
2-sigma <    :   9.114759e-01   7.785224e-01
3-sigma >    :   5.402452e-01   6.583021e-01
3-sigma <    :   9.560552e-01   7.987768e-01
```

Analyze

other files

- run.log
- run.covmat
- run.tex

Conclusion on Design



Outline

Organizing the directory

Installing CLASS Python wrapper

```
]$ cd class/
class]$ make; cd python; python setup.py install --user
```

Configuration

- `cp default.conf.template default.conf` and **edit it**
- `python montepython/MontePython.py -o chains/test \ -p example.param -N 10`

This last line will create the file `chains/test/2014-03-18_10.txt`

Work session example

Test a model with running tilt with (fake) Planck data

- **Copy** `example.param` **into** `example_alpha.param`

```
data.experiments=['fake_planck_bluebook']
```

```
# Cosmological parameters list
data.parameters['omega_b']   = [2.249,  None, None, 0.016, 0.01,'cosmo']
data.parameters['omega_cdm'] = [0.1120, None, None, 0.0016,1,   'cosmo']
data.parameters['n_s']       = [0.963,  None, None, 0.004, 1,   'cosmo']
data.parameters['A_s']       = [2.42,   None, None, 0.038, 1e-9,'cosmo']
data.parameters['h']         = [0.703,  None, None, 0.0065,1,   'cosmo']
data.parameters['tau_reio']  = [0.085,  None, None, 0.0044,1,   'cosmo']
```

Work session example

Test a model with running tilt with (fake) Planck data

- Copy `example.param` into `example_alpha.param`
- **Edit** `example_alpha.param`, **add** Ω_k

```
data.experiments=['fake_planck_bluebook']
```

```
# Cosmological parameters list
data.parameters['omega_b']    = [2.249,  None, None, 0.016, 0.01,'cosmo']
data.parameters['omega_cdm']  = [0.1120, None, None, 0.0016,1,   'cosmo']
data.parameters['n_s']        = [0.963,  None, None, 0.004, 1,   'cosmo']
data.parameters['A_s']        = [2.42,   None, None, 0.038, 1e-9,'cosmo']
data.parameters['h']          = [0.703,  None, None, 0.0065,1,   'cosmo']
data.parameters['tau_reio']   = [0.085,  None, None, 0.0044,1,   'cosmo']
data.parameters['alpha_s']    = [0,      None, None, 0.01,  1,   'cosmo']
```

Work session example

Test a model with running tilt with (fake) Planck data

- Copy `example.param` into `example_alpha.param`
- Edit `example_alpha.param`, add Ω_k
- **Launch a short run to see if it works:**

```
python montepython/MontePython.py -o chains/running \
-p example_alpha.param -N 10
```

Analyzing and Plotting

After running longer chains

- do: `python montepython/MontePython.py -info chains/planck/Omega_k -bins 10`
- use the output covariance matrix as an input (new chains!)

Outline

Exercices

I) Hst, SN, BAO

Make a ΛCDM run with Hubble Space Telescope, Super Novae and BAO data.

II) Analyze the result

Using chains in the dropbox folder

III) Have fun with classy

Use the classy wrapper in a Python interpreter (notebook) to redo yesterday's exercices (see example in the dropbox folder).

IV) Installing Planck likelihood

and trying it with `base.param`

http://pla.esac.esa.int/pla/aio/planckProducts.html