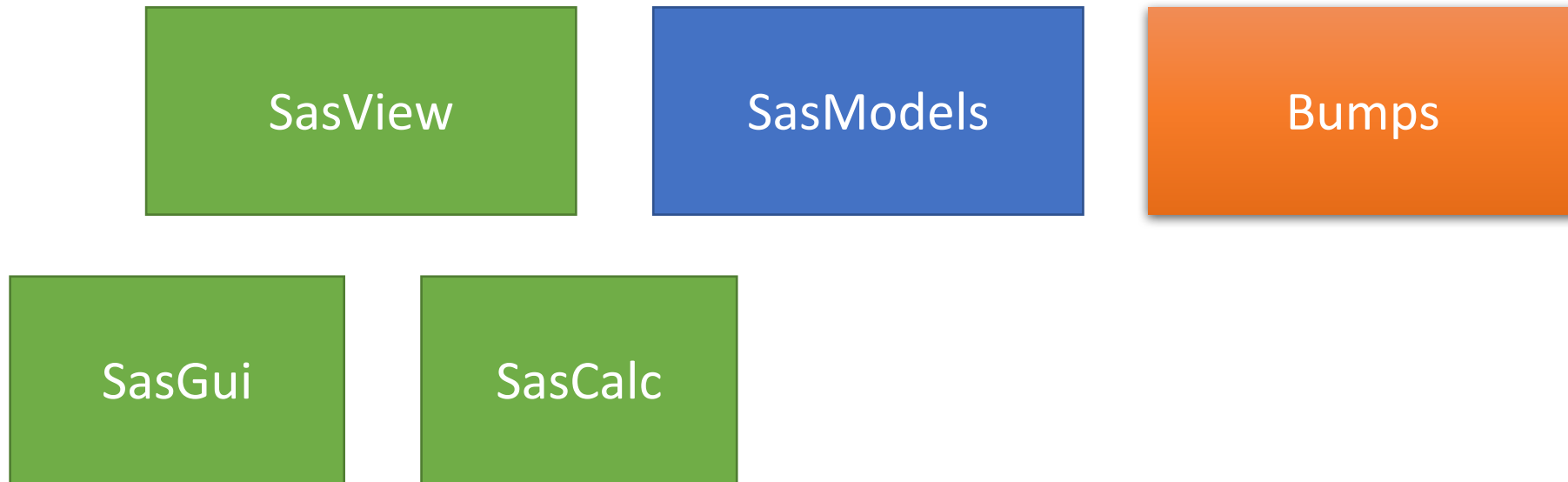


# SasView

## Code structure

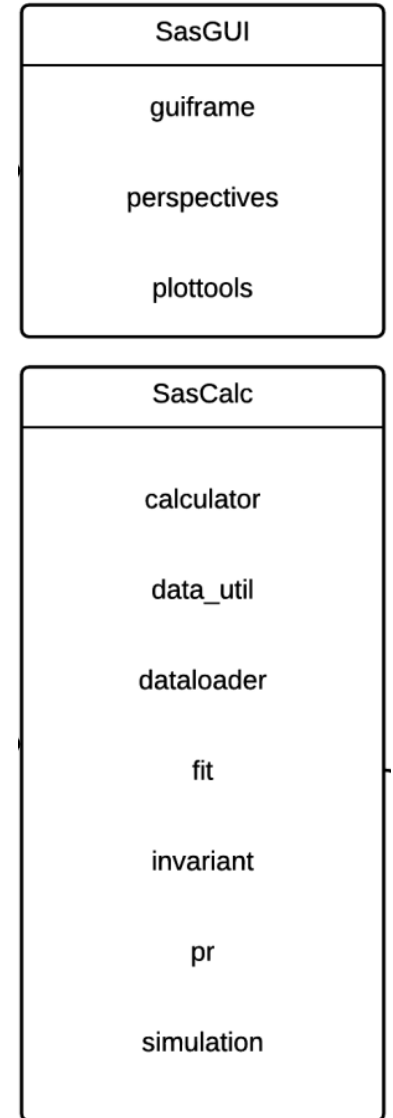
# SasView components



In ideal SasView world they are fully independent

# SasGUI

- Currently developed in wx python
- Transiting to Qt: src/sas/sasgui/qtgui in (ESS\_GUI branch)
- SasGui consists of:
  - Perspectives: src/sas/sasgui/perspectives (fitting, invariant, calculators, corfunc)
  - GuiFrame: src/sas/sasgui/guiframe (non-perspective gui elements)
  - PlotTools: src/sas/sasgui/plottols (ploting tools)

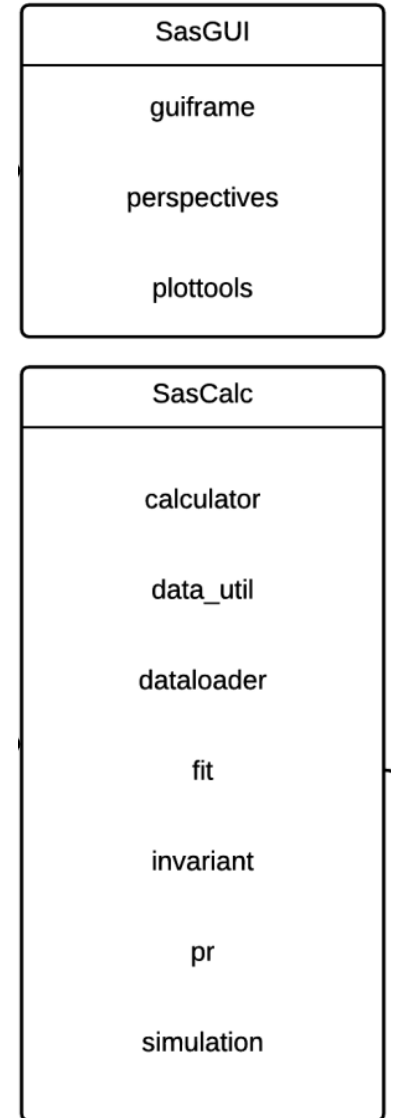


# SasGUI – less obvious cases

- Model editor:  
src/sas/sasgui/perspectives/calculator/model\_editor.py
- Category manager:  
src/sas/sasgui/guiframe/CategoryManager.py
- Settings for OpenCL:  
src/sas/sasgui/perspectives/fitting/gpu\_options.py
- Fitting options (interface to bumps):  
bumps/gui/fit\_dialog

# SasCalc

- Back-end calculations for sasgui perspective
- Data fitting:
  - `src/sas/sascalc/fit/`
- Pr inversion
  - `src/sas/sascalc/pr/`
- Data loader: `src/sas/sascalc/dataloader/loader.py`



# SasCalc

```
from sas.sascal.dataloader.loader import Loader
from sas.sascal.pr.invertor import Invertor
```

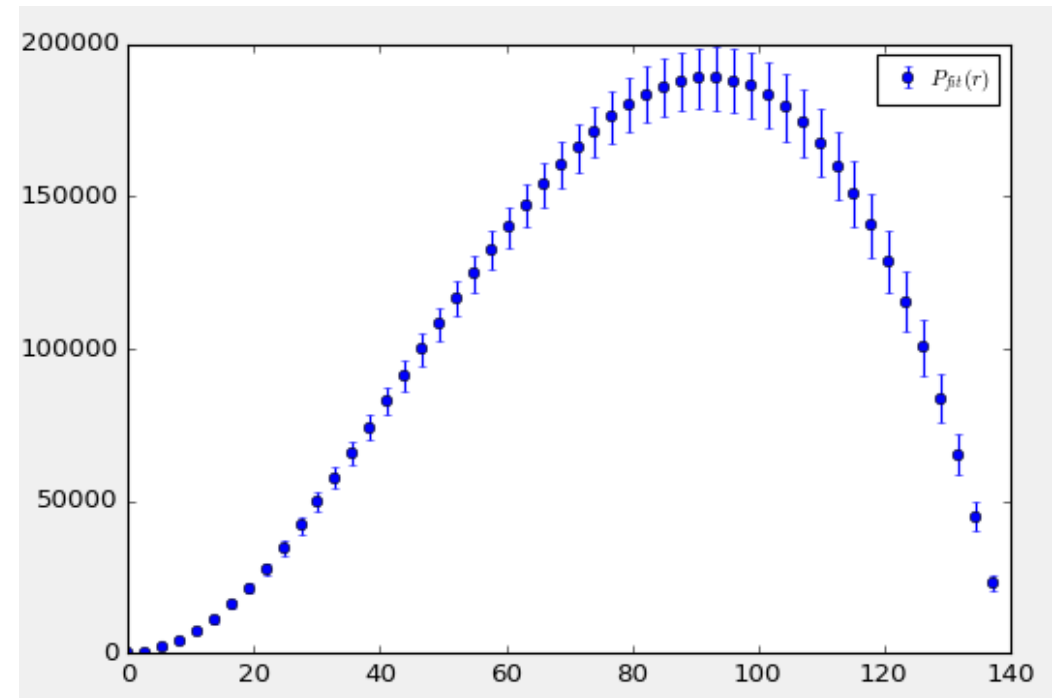
```
loader = Loader()
test_data = loader.load("sphere_80.txt")
```

```
pr = Invertor()
```

```
# Set data
pr.x = test_data.x
pr.y = test_data.y
```

```
# Perform inversion and show graph
x, y = pr.invert()
```

```
import matplotlib.pyplot as plt
plt.plot(x, y)
plt.show()
```



# SasModels

- Models located in: sasmodels/models
- Three different types of models:
  - Python only: broad\_peak.py
  - Python with embedded C: stickyhardsphere.py
  - Python and separate C: barbell.py and barbell.c

# SasModels essential components

- Model Description: Docstring
- Name, title, short description
- Parameters table
- C files to be included during compilation
- Demo parameters values
- Unit tests

# SasModels functions

- `lq` - 1D scattering intensity functions for the case where the scatterer is randomly oriented
- `lqxy` - the 2D scattering intensity functions provide  $I(Q, \phi)$  for an oriented system as a function of a  $Q$
- `Form_volume` - calculates the volume of particle  $V$ , which is used to normalize form factor.
- `ER` – calculates effective radius
- `VR` - calculate particle volume/total volume for shape models
- `INVALID(v)` returns False if `v.parameter` is invalid for some parameter or other (e.g., `v.bell_radius < v.radius`).

# SasModels core modules

- Model computation:
  - sasmodels/sasview\_model.py – interface to sasview
  - sasmodels/kernel.py -interface to all kernel models
  - sasmodels/kerneldll.py –dll driver for c kernels
  - sasmodels/kernelpy.py – python driver
  - sasmodels/kernelcl.py – opencl driver
  - sasmodels/core.py - prepares the model for the execution
  - sasmodels/generate.py – generates model file based on the template

# Sasmodels compare

- Computes model and compares between different platforms (opencl, dll)
- Allows for comparison with old sasview version
- `./compare.sh modelname -1d`
- `./compare.sh modelname -2d`
- Multiple precision, q ranges options

# Unit tests

- SasView tests are located: sasview/test
  - calculatorview
  - corfunc
  - fileconverter
  - pr\_inversion
  - sascalculator
  - sasdataloader
  - sasguiframe
  - sasinvariant
  - sasrealspace
- Sasmodels unit tests defined in models

# Running bumps with sasmodels

- Sasmodels example contains data set for oriented rod-like shape
- `sasmodels/examples/fit.py` (sasmodels interface to bumps)
- Running
  - `bumps fit.py cylinder --preview`
  - `Fit.py` accepts two arguments: type of model and view (radial and tangential)

# Custom settings and models

- Local configuration
  - `~/.sasview/config/custom_config.py`
- Plugin models:
  - `~/.sasview/plugin_models/`
- Model categories:
  - `~/.sasview/categories.json`