EMIR
Data Reduction Pipeline

Sergio Pascual

sergiopr@fis.ucm.es

Why we need a DRP?

- The problem of:
- Growing complexity:
- Instruments are complex
- Data is multidimensional, difficult to visualize and manage
- Growing datasets:
- Raw data sets are large
- Processed data sets are larger: Big Data
- Handling data in the range of the PB

Why we need a DRP?

- The problem of:
- Get funding and scientific return:
- Infrastructure is expensive, the community has to maximize the return.
- Archives are very important tools: legacy and efficiency in data exploitation

Why we need a DRP?

- Processed datasets in a archive provide homogeneous data with known quality
- Directly usable

What we require of a DRP?

- Provide Quality Control
- Capable of producing scientific quality images
- Able to run in commodity hardware
- Able to run in the telescope
- Good defaults for reduction (know-how of the instrument team) but defaults can be modified
- The source code is released, it can be modified
- Good documentation

And what we have

- Open source development (GPLv3)
- Using Python as our language of choice

- Code is hosted in Github:
- https://github.com/guaix-ucm/numina
- https://github.com/guaix-ucm/pyemir

- With a CLI interface and text parameter files

- Documentation with sphinx (from the sources)

Python...

- High level, interpreted language
- Large and comprehensive library
- It can be used in scripts and interactively
- Our work is based in (C)Python 2.7/3.5-3.6
- Lots of interesting ideas and projects:
- Jupyter/IPython notebook
- Astropy.org
- Ginga
- yt
- And others

Python!

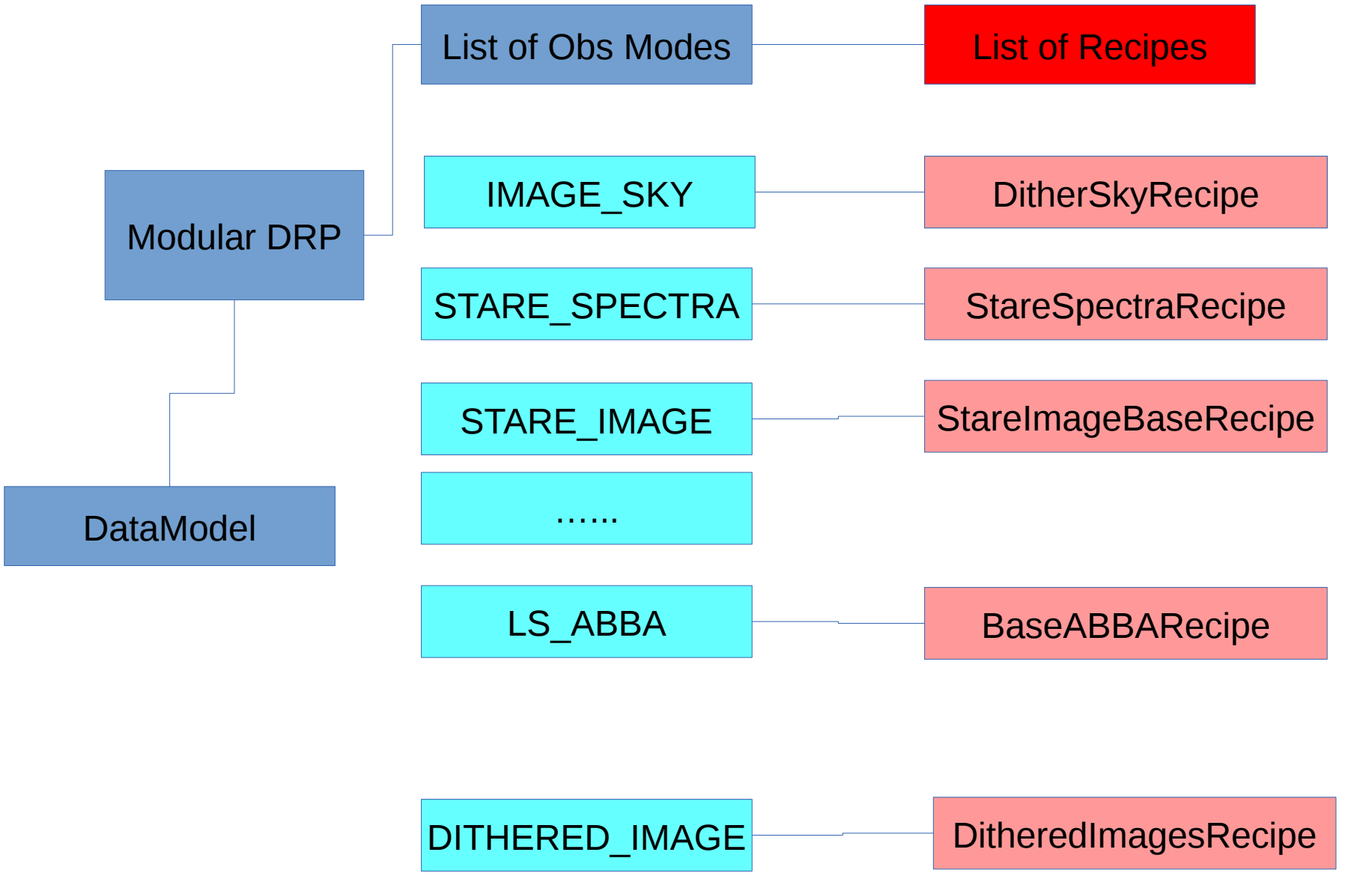
- Python provides a programming API to write *Extensions*.
- In general, C extensions perform better than pure Python code
- But are difficult to write by hand. So we use
- Cython: annotated Python code translated to C (mature)

Our SW dependencies

- Numpy for arrays
- Scipy for generic scientific methods
- Matplotlib for plotting
- Astropy for FITS, WCS, units
- PyYAML for YAML (serialization format, input files)
- Scikit-image for some image processing algorithms
- py.test for unit testing
- Sphinx for documentation

Architecture of the DRP

- Design driven by the Observing Modes
- Logic of the reduction in **Recipes**
-
-
-
-


List of Obs Modes
List of Recipes
Modular DRP
IMAGE_SKY
DitherSkyRecipe
STARE_SPECTRA
StareSpectraRecipe
STARE_IMAGE
StareImageBaseRecipe
DataModel
.....
LS_ABBA
BaseABBARecipe
DITHERED_IMAGE
DitheredImagesRecipe

Architecture of the DRP

- Design driven by the Observing Modes
- Logic of the reduction in **Recipes**
- Each **Recipe** produces some Data Products and requires the Data Products of other Recipes. Numeric parameters also.
- Processing is **non interactive**
-

EMIR

StareImageRecipe

Requirements

Results

- ObservationResult
- MasterBias (opt)
- MasterDark
- MasterBPM (opt)
- MasterFlat
- Sky (opt)

Processing

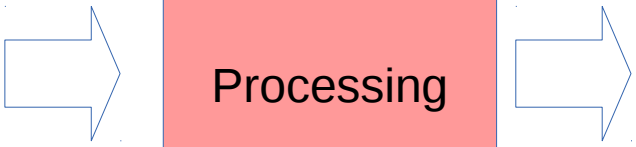
- frame (2D)

EMIR

BarDetectionRecipe

Requirements

- ObservationResult
- MasterBias (opt)
- MasterDark
- MasterBPM (opt)
- MasterFlat
- Sky
- bar_nominal
- median_filter_size
- fit_peak_npoints
- ...



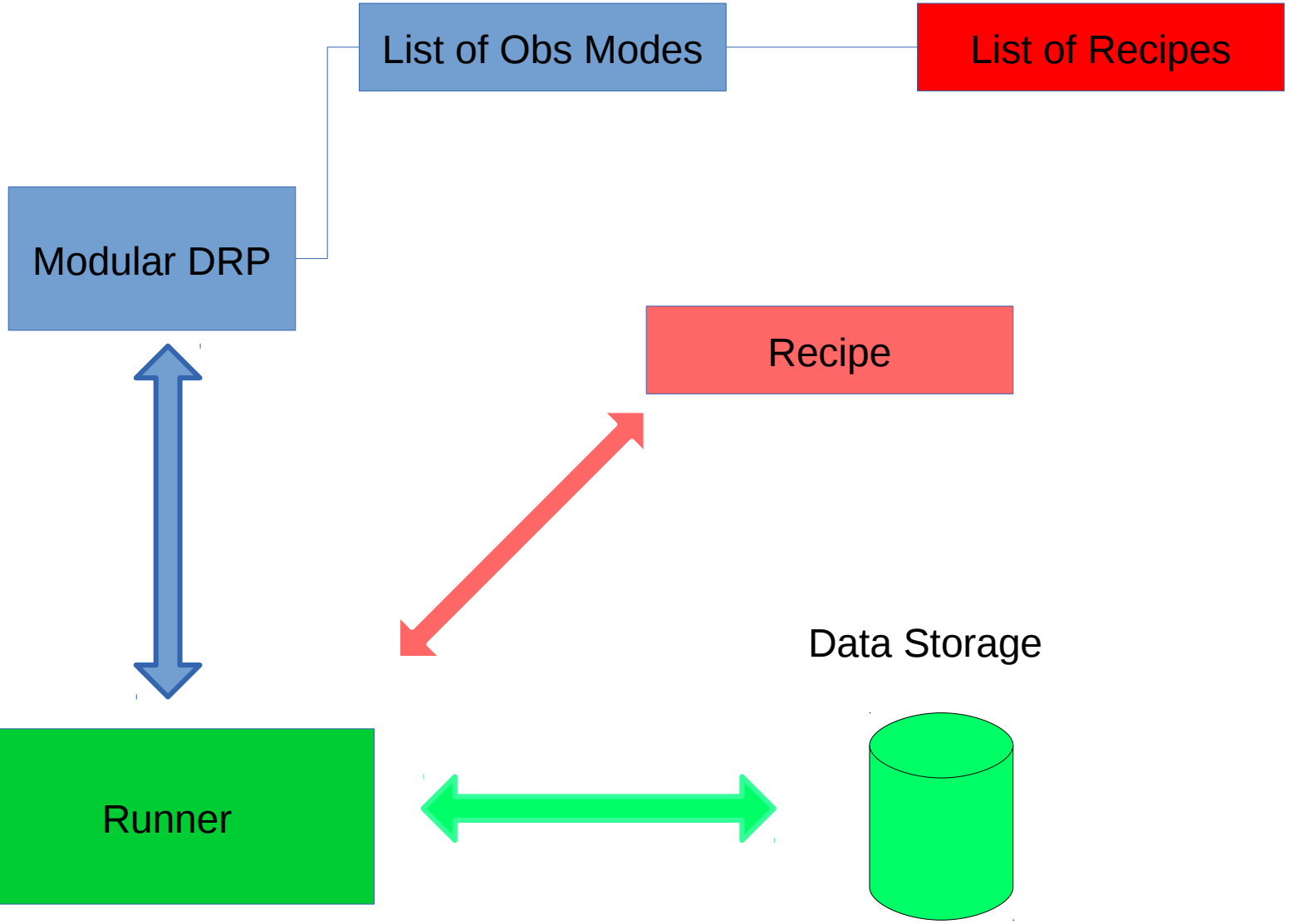
Processing

Results

- frame (2D)
- slits (Table)
- positions (Table)
- csupos (Table)

Architecture of the DRP

- Design driven by the Observing Modes
- Logic of the reduction in **Recipes**
- Each **Recipe** produces some Data Products and requires the Data Products of other Recipes. Numeric parameters also.
- Processing is **non interactive**
- Recipes do not implement the logic needed to acquire its inputs. It is implemented in a different layer, the **Runner**
- Data processing is **isolated from the bookkeeping**


List of Obs Modes
List of Recipes
Modular DRP
Recipe
Data Storage
Runner

A Recipe Runner for the GCS

- Integrated in Data Factory Branch
- It:
- keeps track of the raw images
- initializes the recipe
- and handles the outputs
- Recipes run unmodified
- Images are processed as soon as the OB is finished
- OBs can be nested, Recipes can require the results of previous runs under the same parent OB

A Recipe Runner for your laptop

- The default Runner is a CLI tool called **numina**
- It is designed to be run standalone
- It
- reads configuration files in the disk
- verifies that the requirements are in the disk
- initializes the recipe
- allows intermediate results
- and writes the outputs to disk

Capabilities

- The pipeline processes image mode, both online and offline
  - (There are still bugs, both online and offline)
  - The algorithms can (and will) be improved, with the ~~complains~~ feedback of the users.

- Spectroscopic support is very basic. Just basic reduction, no extraction, no wavelength calibration.
  - But N. Cardiel will show you tomorrow our progress in the field.

- For MOS acquisition, improvements in the online recipes will be required.

Development

- We plan to integrate N. Cardiel improvements as recipes soon (~1 month)
- Beginning next year we plan to have a better recipe runner, so that you don't have to write YAML files. (Automated data ingestion…)
- Open to other improvements (time permitting…)
- We know documentation is … sparse…

Installation

- We need to install **pyemir** and **numina** (dependencies)
- Install via source code, requires development libraries
- Install via conda
- Conda provides precompiled packages for all our dependencies
- Install via pip
- Pypi is starting to provide precompiled packages also
- We will provide wheels (currently building in Travis)

Installation

- We need to install **pyemir** and **numina** (dependencies)
- Install via source code, requires development libraries
- Install via conda
- Conda provides precompiled packages for all our dependencies
- Install via pip
- Pypi is starting to provide precompiled packages also
- We will provide wheels (currently building in Travis)

Installation with conda

Install anaconda from:

https://www.anaconda.com/download

Both versions (2 and 3) are supported.

If you have conda installed already, you don't need to do it again.

Installation with conda

Create and activate an environment (beware of $PATH)

```
$ conda create --name emir python=3
$ source activate emir
```

Installation with conda

Install dependencies

$ conda install numpy scipy astropy matplotlib six scikit-image cython pyyaml pytest

$ conda install -c astropy photutils lmfit

$ pip install sep

Installation with conda

Install source code.

Choose a top level directory for keeping pyemir source code, then:

```
$ git clone https://github.com/guaix-ucm/numina.git
$ git clone https://github.com/guaix-ucm/pyemir.git
```

Installation with conda

Build and install both packages

```
$ cd numina
$ python setup.py build
$ python setup.py install
$ cd ../pyemir
$ python setup.py build
$ python setup.py install
$ cd ..
```

Installation with conda

Check that the pipeline is installed. Run

```
$ numina show-instruments
```

The output should be:

```
DEBUG: Numina simple recipe runner version 0.15.dev5
Instrument: EMIR
 has configuration 'Default configuration' uuid=225fcaf2-7f6f-49cc-972a-70fd0aee8e96
 default is 'Default configuration'
 has datamodel 'emirdrp.datamodel.EmirDataModel'
 has pipeline 'default', version 1
```

Documentation?

- http://numina.readthedocs.io/en/latest/
- http://pyemir.readthedocs.io/en/latest/

Example input file

```
id: 211
mode: STARE_IMAGE
instrument: EMIR
frames:
 - 0000915960-20160615-EMIR-STARE_IMAGE.fits
 - 0000915963-20160615-EMIR-STARE_IMAGE.fits
 - 0000915966-20160615-EMIR-STARE_IMAGE.fits
 - 0000915969-20160615-EMIR-STARE_IMAGE.fits
 - 0000915972-20160615-EMIR-STARE_IMAGE.fits
enabled: True
---
id: 212
mode: STARE_IMAGE
instrument: EMIR
frames:
 - 0000915962-20160615-EMIR-STARE_IMAGE.fits
 - 0000915965-20160615-EMIR-STARE_IMAGE.fits
 - 0000915961-20160615-EMIR-STARE_IMAGE.fits
 - 0000915977-20160615-EMIR-STARE_IMAGE.fits
 - 0000915973-20160615-EMIR-STARE_IMAGE.fits
enabled: False
```

Example control file

```
version: 1
rootdir: "/home/spr"
products:
  EMIR:
    - {id: 301, type: 'MasterBadPixelMask', tags: {},
content: ‘mask_bpm.fits’}
    - {id: 101, type: 'MasterDark', tags: {}, content:
‘master_dark.fits’}
    - {id: 201, type: 'MasterIntensityFlat', tags: {‘filter’:
‘Ks’}, content: ‘master_flat_Ks.fits’}
    - {id: 202, type: 'MasterIntensityFlat', tags: {‘filter’:
‘J’}, content: ‘master_flat_J.fits’}
requirements:
 {}
```