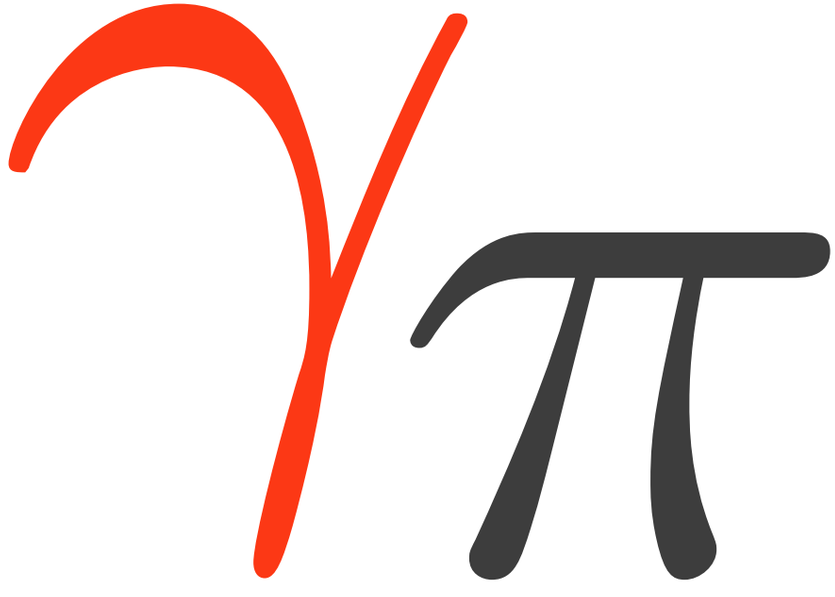
DL3 IN GAMMAPY

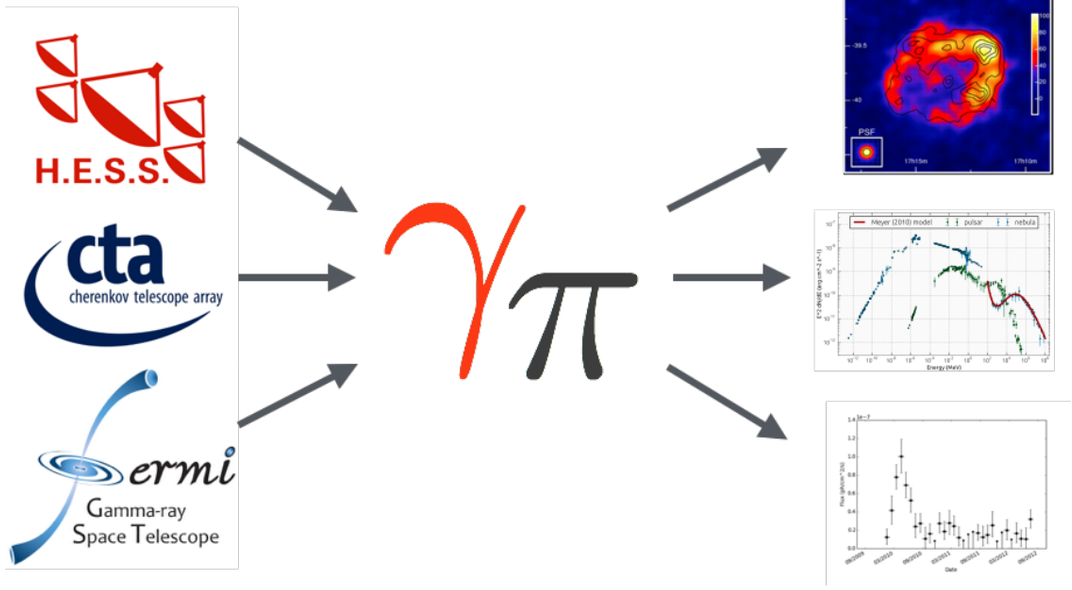
Christoph Deil, MPIK Heidelberg
IACT DL3 meeting, Meudon, April 6, 2016

WHAT IS GAMMAPY?



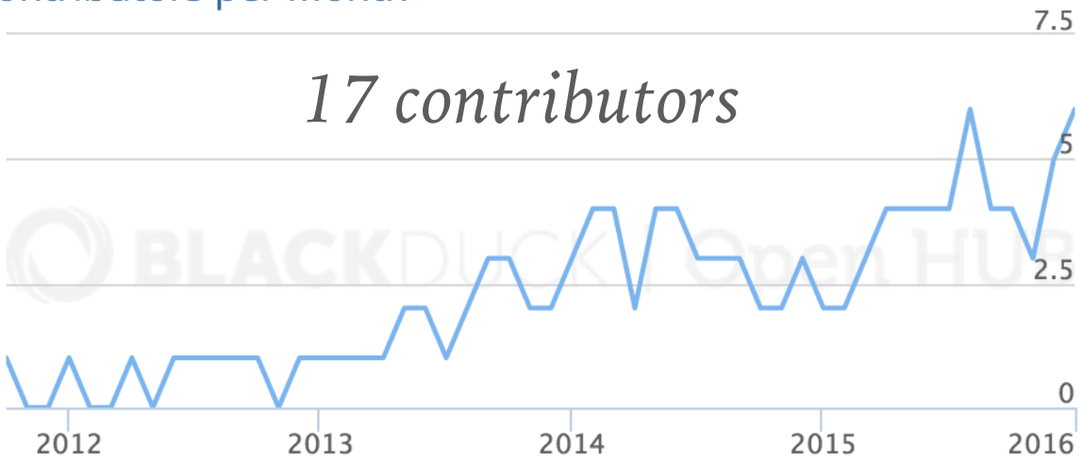
GAMMAPY INTRO

A **Python** package for **gamma-ray** astronomy

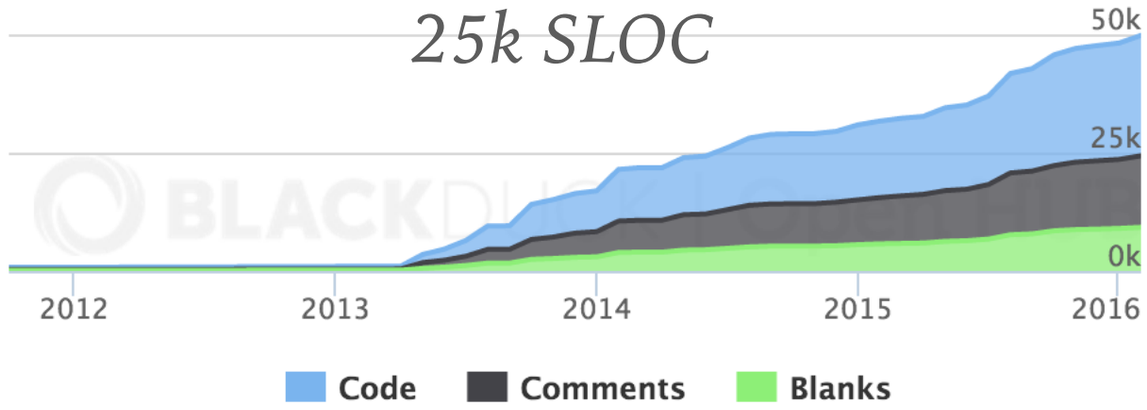


- A Python package for gamma-ray science tool data analysis.
- Open-source. Open-development. Github. github.com/gammapy/gammapy
- To-be Astropy-affiliated package.
- Scope for IACTs: DL3 — DL 5 (gammapy.shower was moved to ctapipe)
- Development pretty active.
  Plan first coding sprint at MPIK in June,
  1.0 release this summer, paper in the fall.

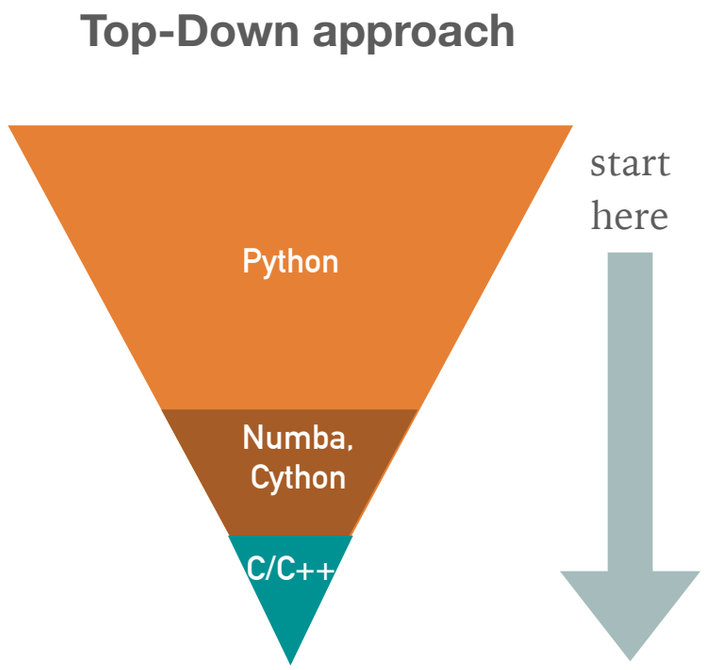
Contributors per Month



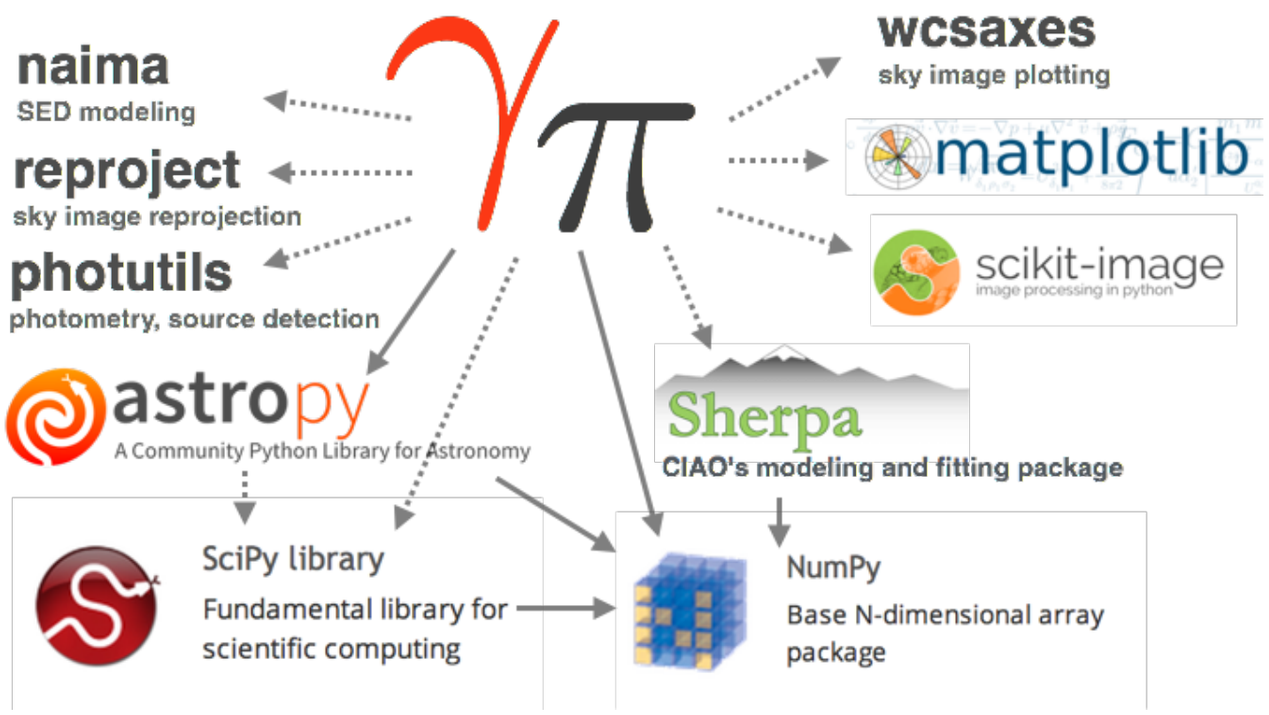
Lines of Code



GAMMAPY IMPLEMENTATION



- Python and dependencies
  - Scientific Python stack (Numpy, Scipy, matplotlib)
  - Astropy and affiliated packages (reproject, photutils, naima).
  - Sherpa for modeling and fitting



- Collaborate with Astropy community
  - E.g. we are contributing to astropy.regions, will remove gammapy.regions soon.
- Gammapy design compared to
  - ctapipe — similar
  - Gammalib / ctools — different

A **Python** package for **gamma-ray** astronomy

General documentation

- About Gammapy
- Installation
- Getting Started
- Tutorials and Examples
- Data Formats
- References
- Developer documentation
- Changelog

The Gammapy toolbox

- Command line tools (`gammapy.scripts`)
- Astrophysical source and population models (`gammapy.astro`)
- Background estimation and modeling (`gammapy.background`)
- Catalog (`gammapy.catalog`)
- Cube Style Analysis (`gammapy.cube`)
- Data and observation handling (`gammapy.data`)
- Access datasets (`gammapy.datasets`)
- Source detection tools (`gammapy.detect`)
- Image processing and analysis tools (`gammapy.image`)
- Instrument response function (IRF) functionality (`gammapy.irf`)
- Morphology and PSF methods (`gammapy.morphology`)
- Regions (`gammapy.region`)
- Spectrum estimation and modeling (`gammapy.spectrum`)
- Statistics tools (`gammapy.stats`)
- Time handling and analysis (`gammapy.time`)
- Utility functions and classes (`gammapy.utils`)

GAMMAPY FEATURES

- Classical analysis implemented (2D images and 1D spectra)
- Cube analysis (3D lon-lat-energy) work in progress.
- So far focus on IACTs. Joint analysis with Fermi-LAT not available yet.
- Caveat: Gammapy is work in progress. If you use it you will find missing features and bugs. Let us know!

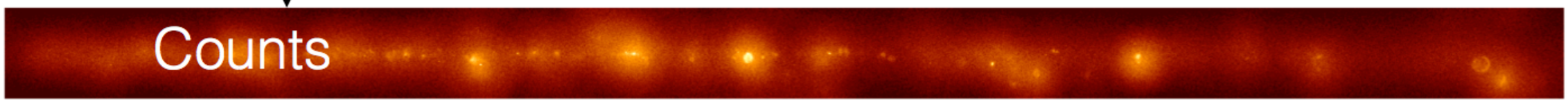
GAMMAPY APPLICATION EXAMPLE

With very little Python code, go from an event list to a source catalog for the H.E.S.S. Galactic plane survey.

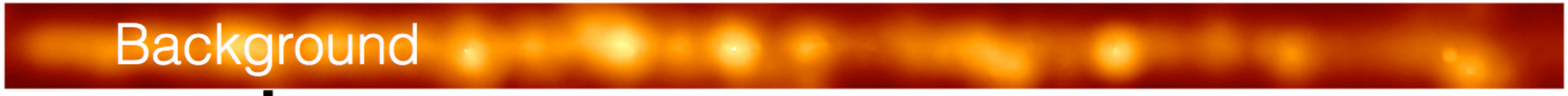
DL3

Input: Event list: | Energy | RA | DEC |

Coordinate transformations: ~ 1 minute

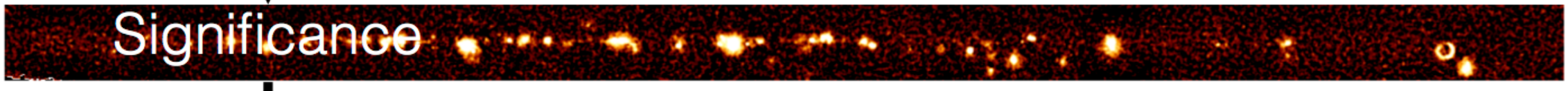


Iterative method: 2 minutes



DL4

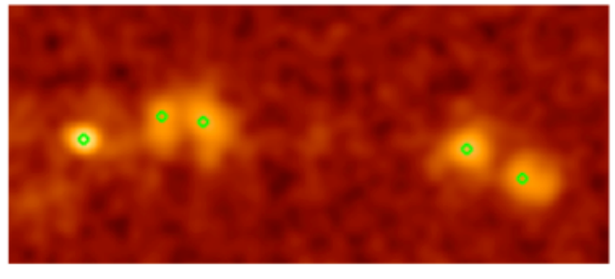
Very fast test statistic TS image computation: 5 minutes



Peak finder: ~ 1 minute

DL5

Source catalog: | Source Number | RA | DEC | ... |



DL3 IN GAMMAPY

OVERVIEW

- Can read and process DL3 if it's in the open-gamma-astro-data spec format (the one produced by HESS and VERITAS exporters, and consumed also by Gammalib)
- EVENTS, IRFs, Background
- Slice out OGIP responses (PHA, ARF, RMF) for a given source position
- DataStore to work with observation and HDU index tables

EVENT LISTS (THE GOOD PARTS)

- **gammapy.data.EventList,** sub-class of **astropy.table.Table**
- Main data members are exposed as properties:
  - **time** — astropy.time.Time
    - Uses SOFA / ERFA internally (high precision, many scales)
  - **radec** — astropy.coordinates.SkyCoord (ICRS system)
    galactic — astropy.coordinates.SkyCoord
    altaz — astropy.coordinates.SkyCoord
    field of view coordinates not yet (open pull request to Astropy)
    - Uses SOFA / EFRA internally, exposed via the nice astropy.coordinates.SkyCoord API.
  - **energy** — Energy (astropy.units.Quantity sub-class)
    - Gammapy: quantities (= numpy array + unit) everywhere.

EVENT LIST CODE EXAMPLE

```
1  from gammapy.data import EventList
2  events = EventList.read('events.fits')
3  events.show_in_browser(jsviewer=True)
4  events.info('stats')
5  events.peek()
6
7  # do whatever analysis you want
```

EVENT LISTS (THE BAD PARTS)

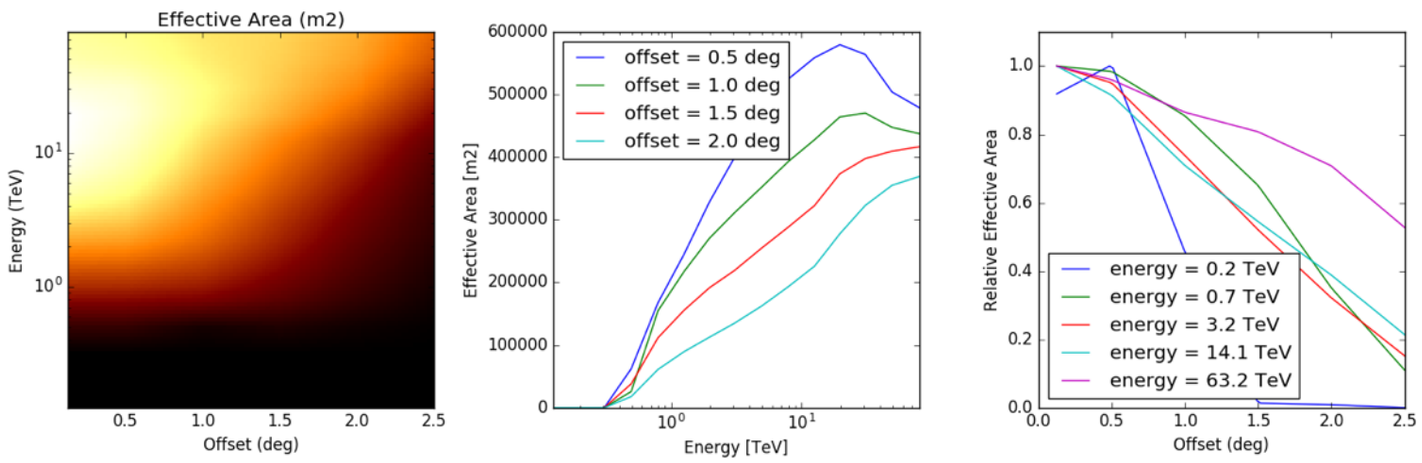
- Observation and GTI information currently a mess!
  - We're not sure where to store observation (e.g. pointing) and GTI information related to a given event list table
  - At the moment duplicated on the EventList, EventListDataset and DataStoreObservation class.
- This reflects the state that the DL3 data model is unclear.
  - How and where is observation information (e.g. pointing) stored?
    - In the event list header or a separate HDU?
  - How do event tables connect to GTIs and IRFs?

GAMMAPY.IRF

- Classes to read and use various formats for AEFF, EDISP, PSF and BKG that are in use.
- FITS I/O via astropy.table.Table.
  Bilinear interpolation via scipy.interpolate.RegularGridInterpolator.
  Integration (where needed) implemented just using numpy.sum.
- We assume smooth DL3 IRFs, no de-noising applied on read.
  (this is an issue, because some H.E.S.S. IRFs are noisy)
- Not well implemented at the moment!
  Lots of code duplication between the IRF classes.
- Will re-factor after this meeting, or, if flexIRF or ctapipe or someone else provides a nice package we can use, just import that or contribute there!
- Requirements that we would use it in Gammapy:
  - easy to install (conda package)
  - efficient to use from Python (support numpy arrays)

GAMMAPY.IRF CODE EXAMPLE

```
$ ipython

In [1]: from gammapy.irf import EffectiveAreaTable2D

In [2]: aeff2d = EffectiveAreaTable2D.read('aeff_2d_23523.fits.gz', hdu='EFFECTIVE AREA')

In [3]: print(aeff2d.info())

Summary EffectiveArea2D info
----------------------------
Energy         : size =    16, min =  0.099 TeV, max = 101.193 TeV
Offset         : size =     6, min =  0.125 deg, max =  2.500 deg
Effective Area : size =    90, min =  0.000 m2, max = 621144.938 m2
Safe energy threshold lo:  0.542 TeV
Safe energy threshold hi: 37.883 TeV
Effective area at 0.5 deg and 1.0 TeV : 206384.250 m2

In [4]: aeff2d.peek()
```



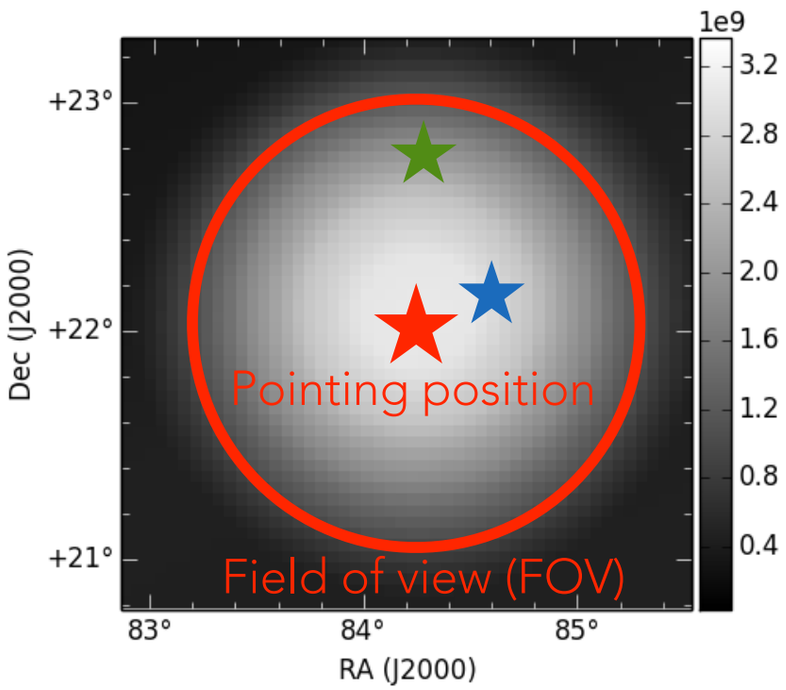
1D SPECTRAL ANALYSIS. OGIP FILES.

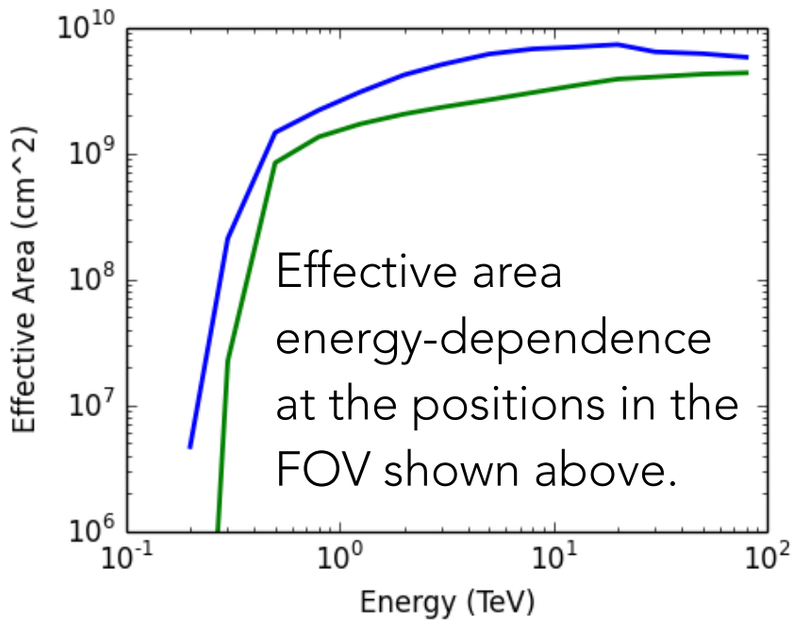
- Gammapy provides tools to slice out 1D response in OGIP format (PHA, ARF, RMF) and pass that to Sherpa for spectral analysis.
- How important is this for existing IACTs and CTA?
  - For the first CTA data challenge (CTA-1DC) OGIP responses were distributed.
- Should ARF, RMF, RPSF be part of the DL3 spec?
  - I think yes. (It's good to have a page linking to the OGIP specs, explaining what they are and how to obtain them from the other DL3 IRFs and e.g. defining header keys for on-region radius and safe energy threshold)
- Should ARF, RMF, RPSF be distributed as DL3 files?
  - I think no. (They contain a subset of the info in the IRF files we have now and can easily be computed for any source region.)

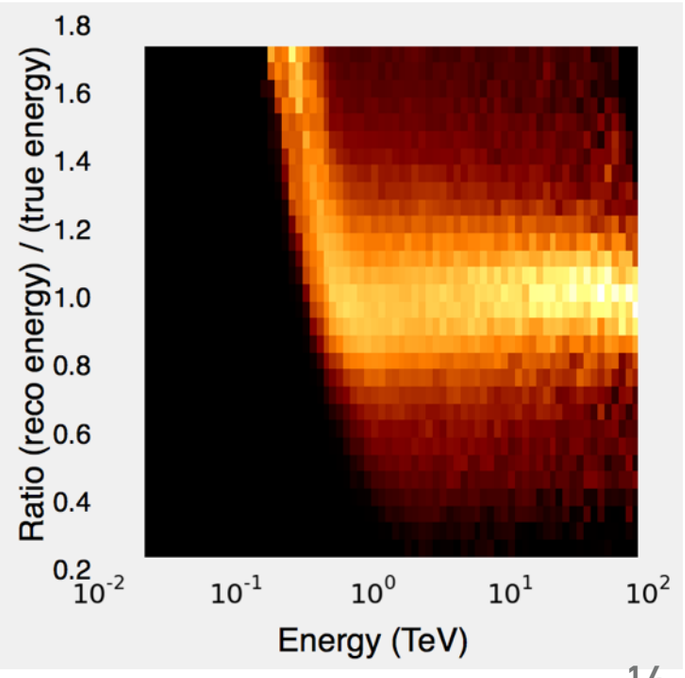
OGIP EXAMPLE

```
>>> from gammapy.irf import EffectiveAreaTable2D
>>> aeff2d = EffectiveAreaTable2D.read('aeff_2d_23523.fits.gz', hdu='EFFECTIVE AREA')
>>> arf = aeff2d.to_effective_area_table(offset='1.3 deg')
>>> print(arf.info())

Summary ARF info
----------------
Energy Bounds  : size =    16, min =  0.099 TeV, max = 101.193 TeV
Effective area : size =    15, min =  0.000 m2, max = 426802.094 m2
Safe energy threshold lo:  0.542 TeV
Safe energy threshold hi: 37.883 TeV
```

```
>>> arf.write('arf.fits')
```



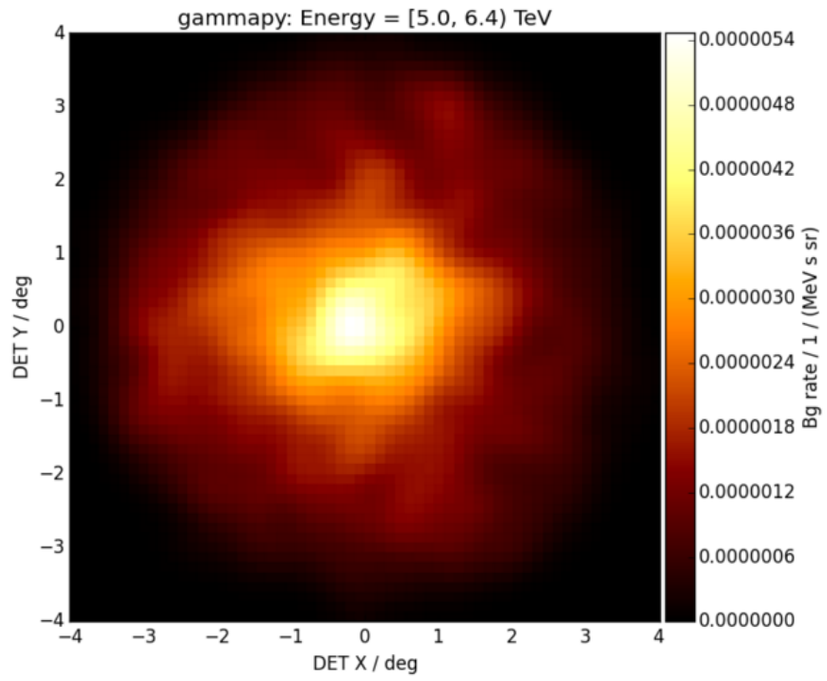




GAMMAPY.BACKGROUND

- 2D and 3D background models as written in the spec.
- Some background modeling methods implemented
- Next steps: proper implementation of field of view coordinates, reprojection into sky maps.
- Side comment: we need to talk about background models for CTA (from off and AGN observations, I don't believe that from MC is possible!)

- Who produces them? Regular re-productions?
- How are they distributed to end users?
- Probably one of the most tricky things for DL3 production.



DL3 DATA STORE

- The **gammapy.data.DataStore** class is our **butler** (other experiments like LSST have similar, but much fancier classes).
- We have classes matching the index files in the spec:
  - observation index for selection of list of OBS_ID
  - HDU index for locating EVENTS, IRF and BKG for a given OBS_ID
- I think it's good to have this as an abstraction layer to isolate science code from serialisation.

GAMMAPY DATASTORE CODE EXAMPLE

```
>>> from gammapy.data import DataStore
>>> data_store = DataStore.from_dir('$GAMMAPY_EXTRA/datasets/hess-crab4-hd-hap-prod2')
>>> data_store.info()

Data store summary info:
name: noname

HDU index table:
BASE_DIR: /Users/deil/code/gammapy-extra/datasets/hess-crab4-hd-hap-prod2
Rows: 28
OBS_ID: 23523 -- 23592
HDU_TYPE: ['aeff', 'edisp', 'events', 'gti', 'psf']
HDU_CLASS: ['aeff_2d', 'edisp_2d', 'events', 'gti', 'psf_3gauss', 'psf_king', 'psf_table']

Observation table:
Number of observations: 4

>>> obs = data_store.obs(obs_id=23592)
>>> type(obs.events)
gammapy.data.event_list.EventList
>>> type(obs.events)
gammapy.irf.effective_area_table.EffectiveAreaTable2D

# Science analysis code can loop over obs and processes them,
# to compute images, spectra, cubes, lightcurves
```

SUMMARY DL3 IN GAMMAPY

- Gammapy can consume DL3 in the most common format (open spec, HESS, VERITAS, Gammalib)
- The peek and info methods are useful for quick-look at DL3, we're using it to debug the HESS FITS exporters.
- There's methods to slice out OGIP response (ARF, RMF) that we use for XSPEC-style analysis via Sherpa.
- Analysis is currently OBS_ID based, will switch to GTI-based analysis or whatever is decided for the DL3 spec.
- So far no code-sharing with ctapipe (except for some copy & paste of utility functions) or Gammalib / ctools.

PERSONAL COMMENTS ON DL3 IN GAMMAPY

- Prototyping in Gammapy (and ctools) and open-gamma-astro-data spec writing were done in parallel in the past year.
  - I think this is a very good way to develop both the format and code … iteratively add features, re-factor when better concepts become clear.
- I do like the DataStore (might rename to Butler).
  I'm convinced ctapipe will need something similar and that it's also useful to have for science tools. Would it be useful to have a shared IRF or data access library?
- We only do binned analysis (considering adopting sparse HEALPIX from pointlike for efficient low-stats analysis). IRF interpolation efficiency is not a concern for us! Although, we use bi-linear interpolation, which is very fast.
- I don't like having to support a ton of analytical IRFs.
  We could just histogram when producing DL3 and avoid having to update the science tools every time someone wants to try a new parametrisation (for ctapipe and DL3 analytical IRFs are useful though for de-noising or memory efficiency).