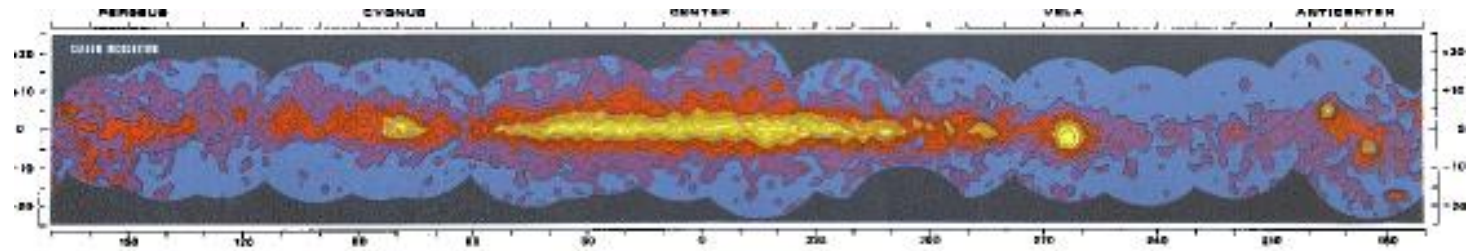


Fermi analysis with pointlike

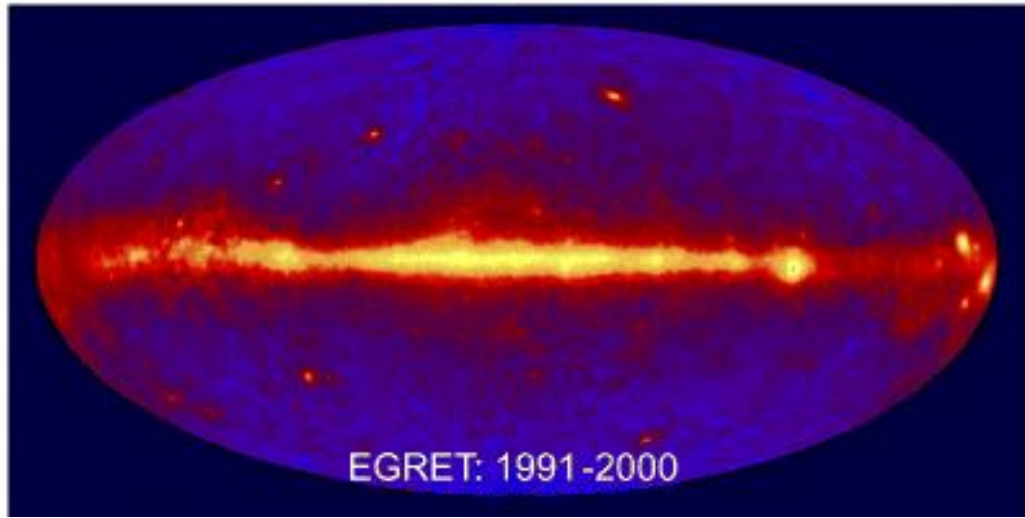
Toby Burnett

University of Washington

Gamma-ray astronomy before Fermi

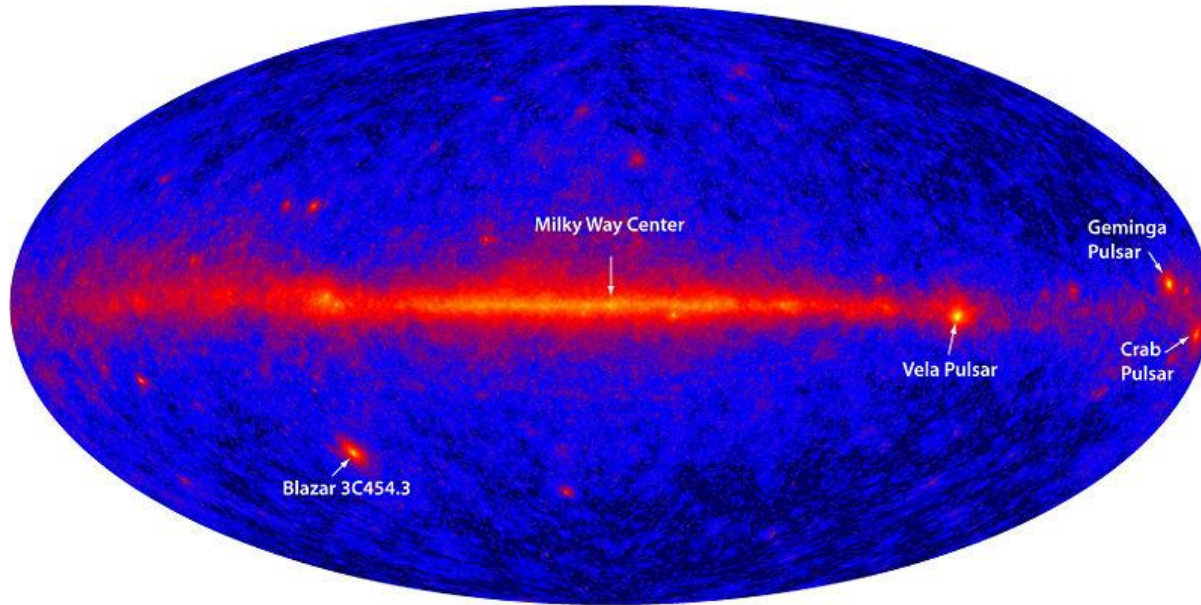


COS-B: 1975-82



Small FOV,
pointed
observations

The new era, starting 10 years ago



Large FOV, 20% of sky;
Dominantly scanning

This image is the *GLAST* first light, based on 4 days scanning in July after 11 June 2008 launch

We have been taking science data since Aug 4, 98.6% uptime

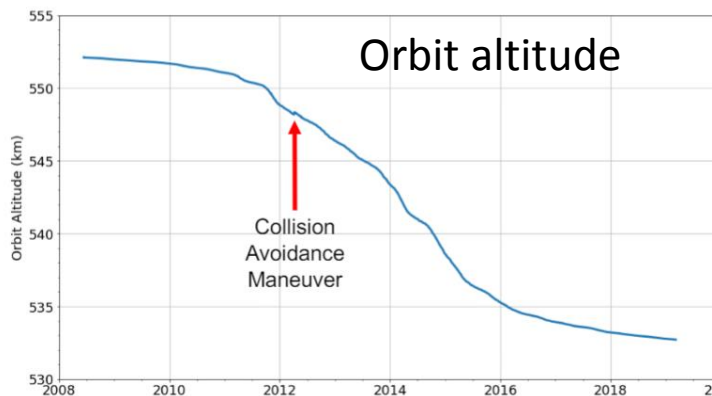
How were we so ready?

- Delayed launch, from initial 2005 expectation
- Had a full simulation used for the 2001 proposal
- Spent those years productively, especially with two “data challenges”
- Software categories
 - Onboard filter
 - Simulation, including preliminary reconstruction
 - Event Reconstruction
 - Science Tools (based on EGRET experience)-> fermipy
 - Pointlike
 - Fast, originally for localization, likelihood-based source finding
 - All-sky binning with energy-dependent pixel sizes, using sparse HEALPix (then new)

Some details

- Approx rates:
 - Trigger: 2 kHz (almost all cosmic rays)
 - Downlink: 400 Hz
 - “Photons”: 4 Hz
 - >100 MeV source analysis: 0.7 Hz

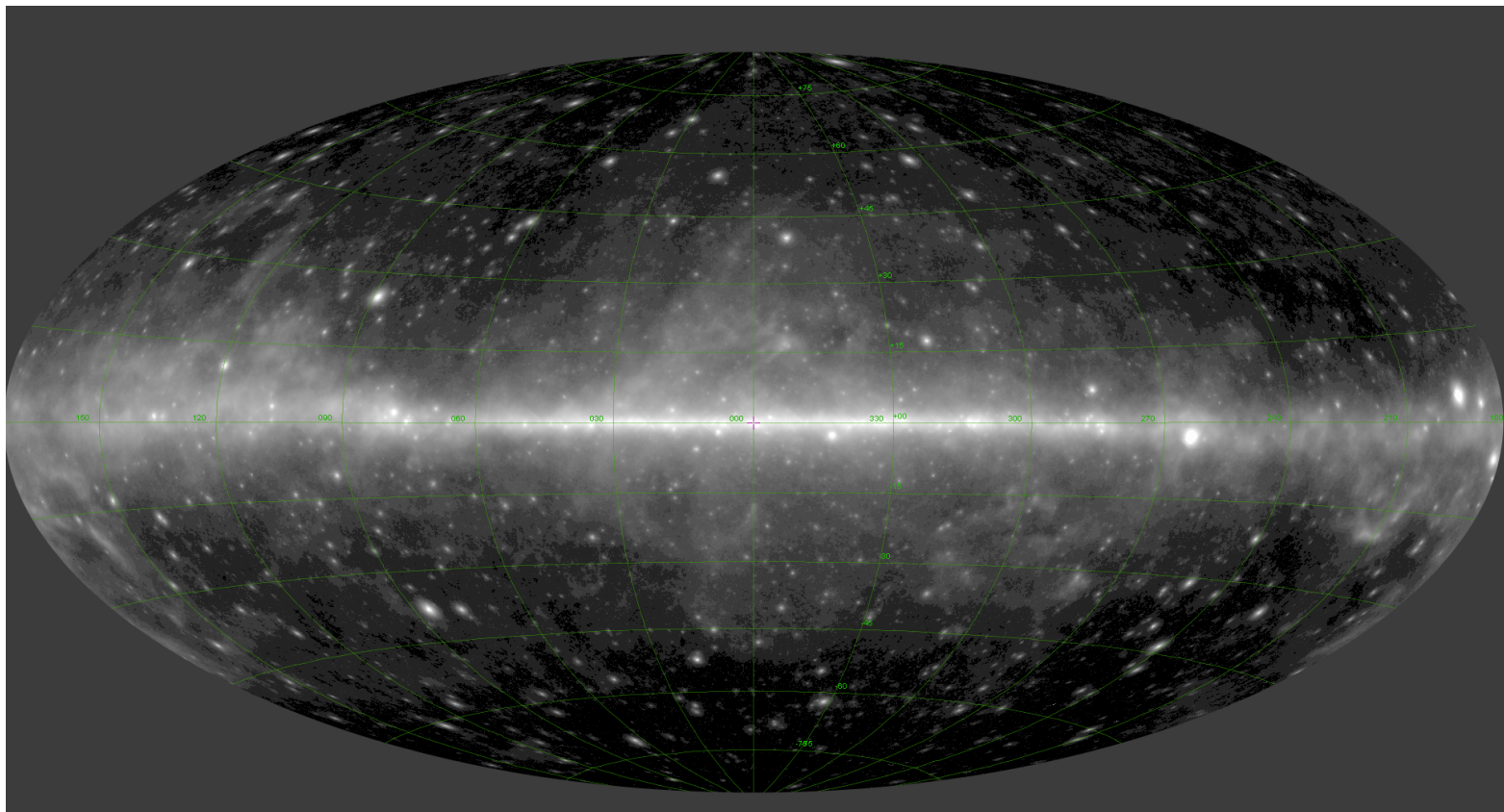
↓ A **lot** of filtering
Changes to improve efficiency
and purity were applied slowly
during mission



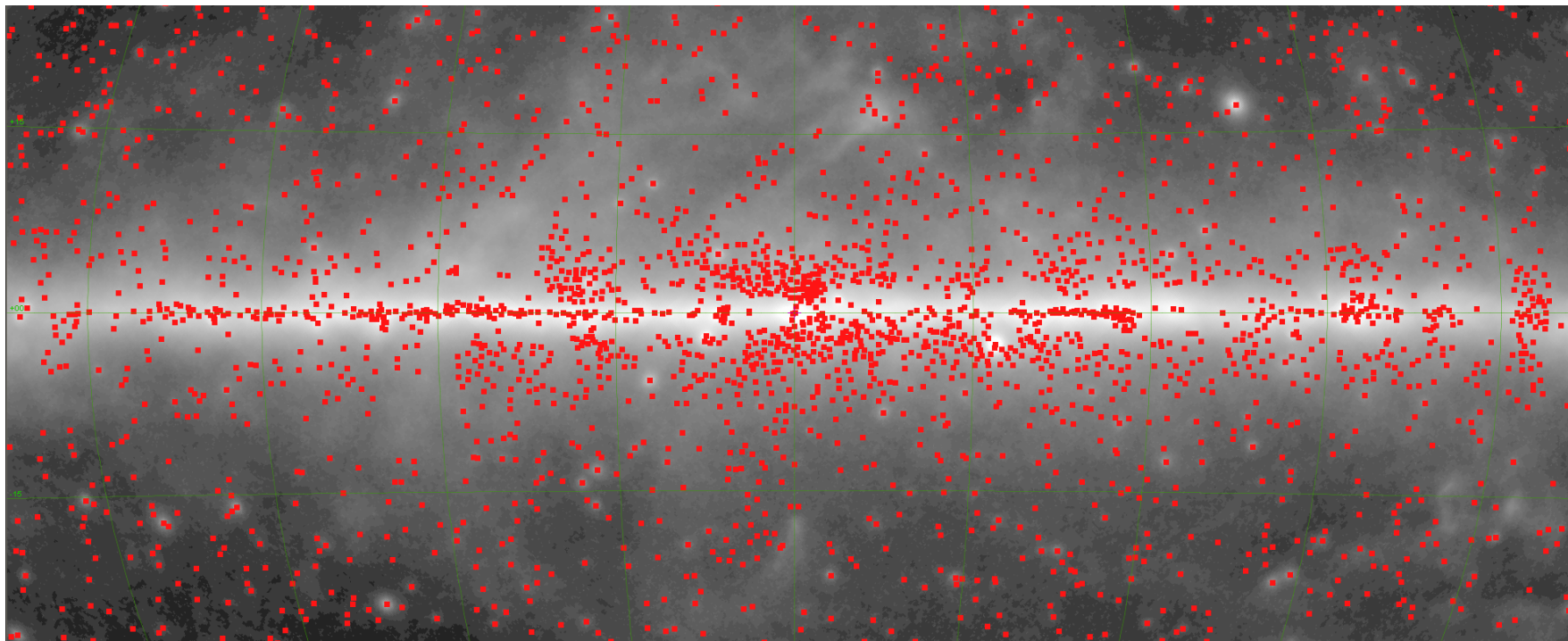
Showing two types of collisions in orbit:

- Junk (fortunately avoided)
- Atmosphere

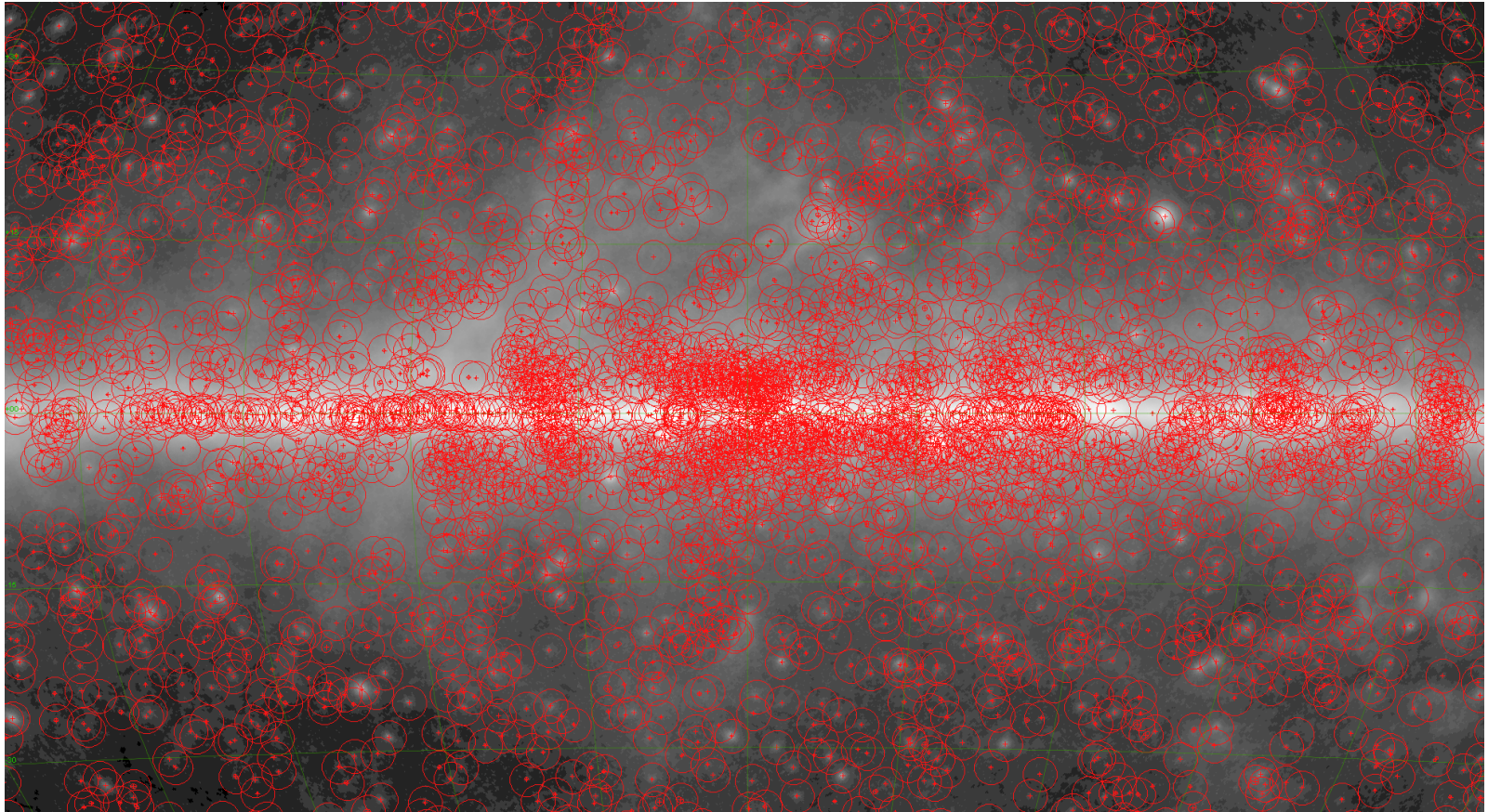
Map of the photon data



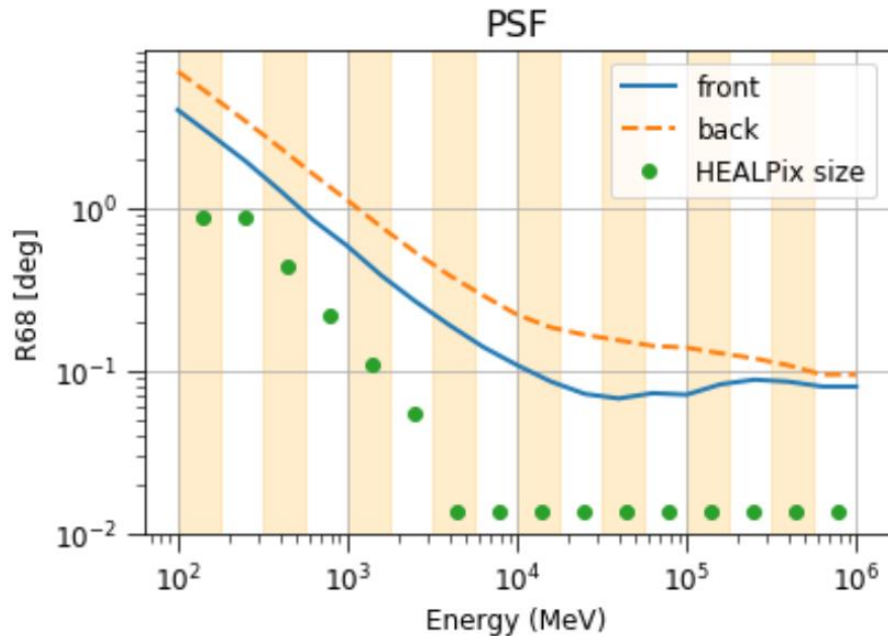
Apparent point sources near ridge



Now with 3 deg circles



PSF, Pointlike binning in position and energy

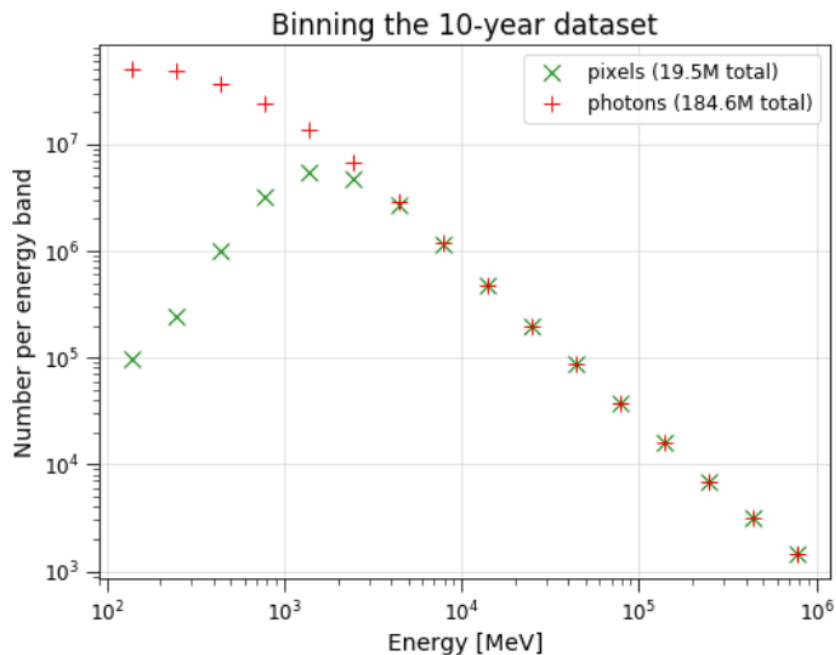


From 3 deg at low energies to 0.1 above 10 GeV

HEALPix sizes (for front) from nside 64 to 4096

The 4FGL data set, with pointlike sparse binning

Lowest energies: many more photon, but poor resolution, confusion with diffuse and other sources

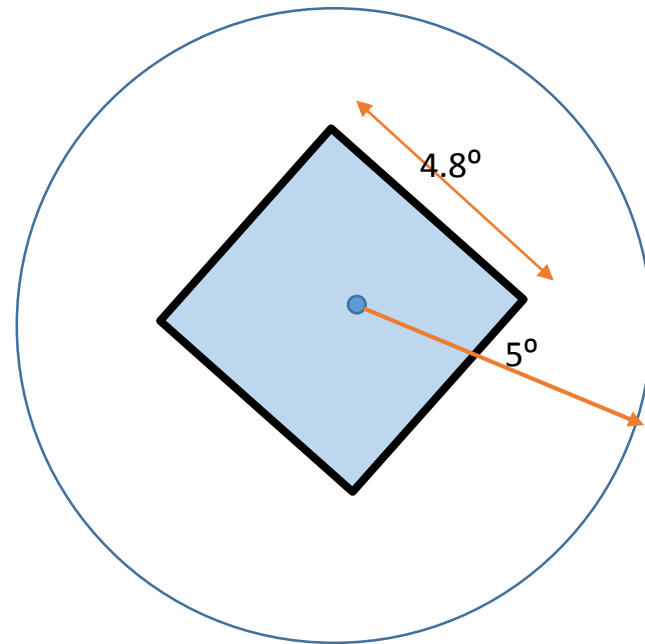
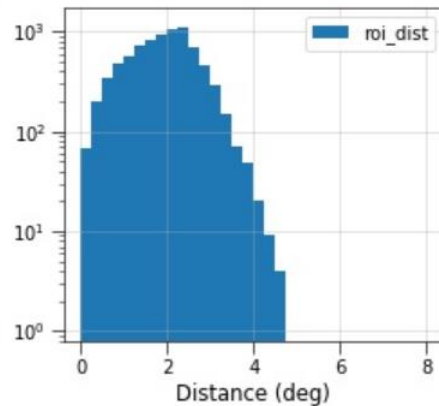
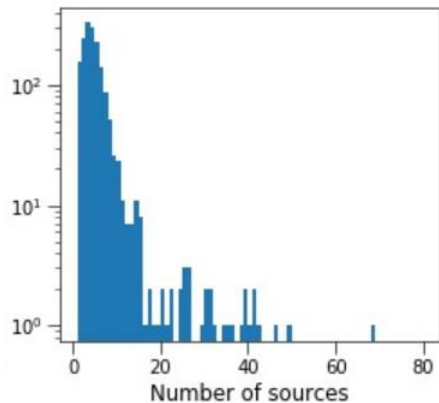


File size: 190 MB

- We have been doing this since before launch: HEALPix was very new then
- Now, to be consistent with gammapy, we adopted its HEALPix sparse data format, increasing the filesize by ~50%
- Another accommodation: only use power-of-2 nside values

Geometry of an RoI

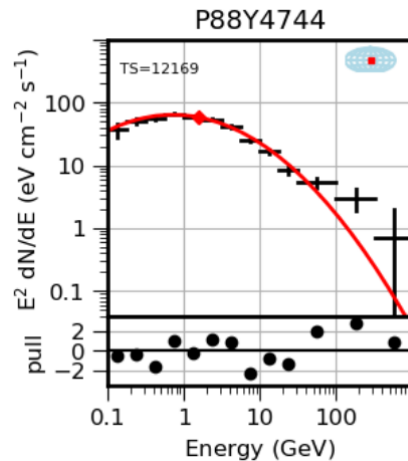
- nside=12, 1728 locations
- Data: include pixels within 5 deg.circle
- All sources, out to 10 deg or so, which contribute to counts
- Sources within diamond are variable in fit



Interactive Analysis: fast by design

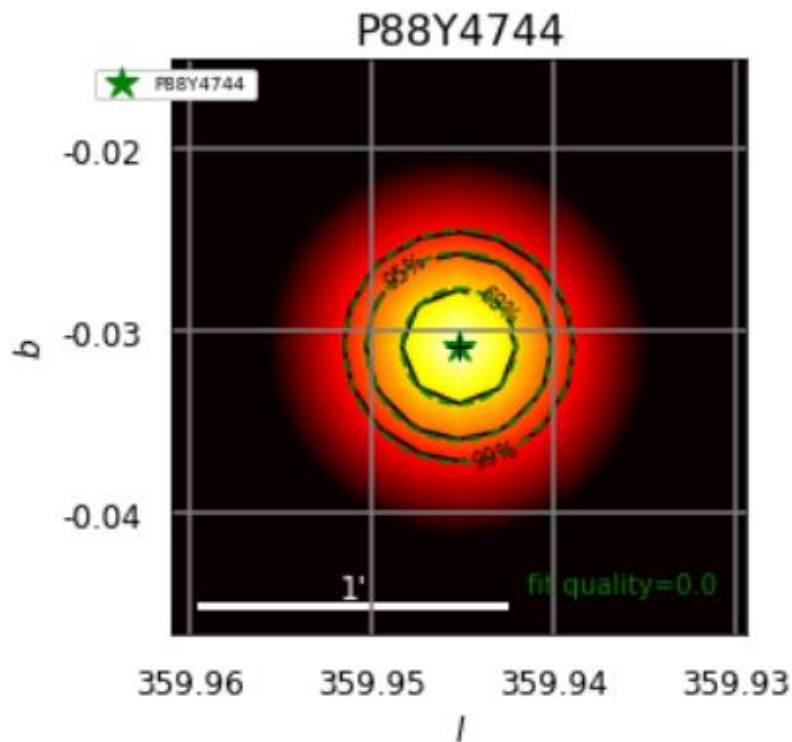
- Load the full 8-year dataset: 5 s.
- Setup (almost) worst case ROI, at GC: 4 min.
 - 567 sources (48 free)
- Perform full fit (96 parameters): 1 min.
 - (I use `scipy.optimize.fmin_l_bfgs_b`, calculate hessian myself.)
- Fit a single source (use selection mechanism: one column): 3.2 s
 - Evaluate each band for SED: 8.0 s.
 - Generate SED plot: 7.4 s.

Strong source near GC:
maybe SGR A*?



Interactive source analysis, cont.

- Perform profile fit to energy flux: 11s
flux = 57.138 (1 + 0.011 - 0.011) eV/cm²/s, TS=13360.
- Localize a source: 0:31
- Make a TS map w/ fit info: 1:18



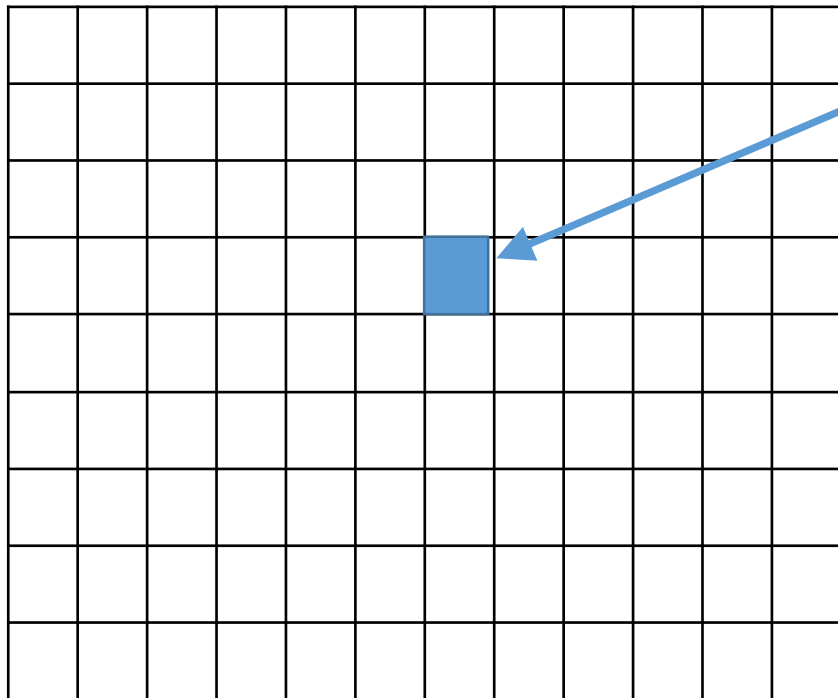
Under the hood

How the likelihood calculation is organized

The likelihood grid of Response objects

Sources: global, extended, point

Bands: each energy range, event type



- ❑ Each cell: a *Response* object for the given source: calculates photons for each pixel, and the total in the RoI
- ❑ Each row: a *BandLike* object to calculate the likelihood for (selected) sources
- ❑ The table: a *BandLikeList* object calculates total likelihood for (selected) set of bands.

The likelihood grid of Response objects

Sources: global, extended, point

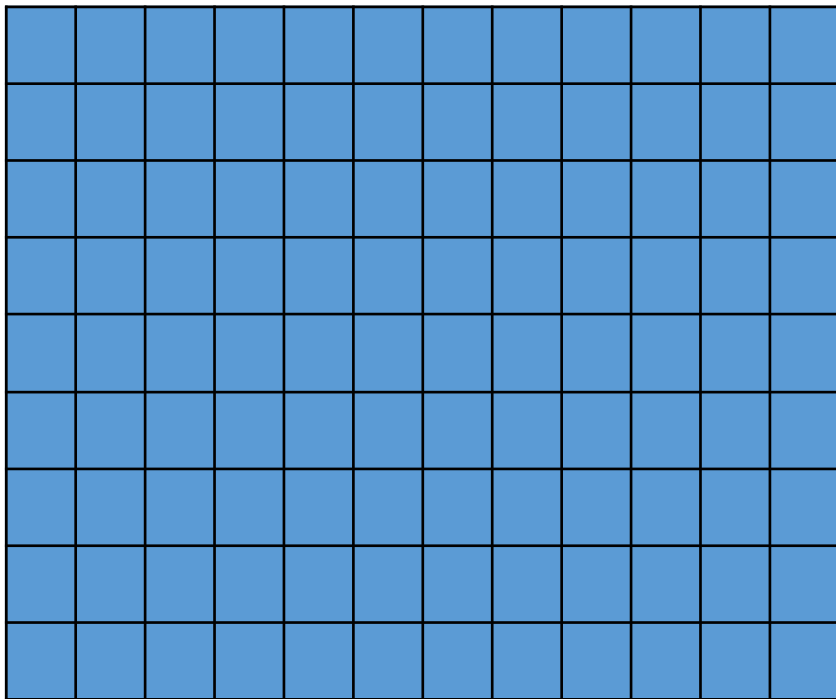
Bands: each energy range, event type

- ❑ Each cell: a *Response* object for the given source: calculates photons for each pixel, and the total in the RoI
- ❑ Each row: a *BandLike* object to calculate the likelihood for (selected) sources
- ❑ The table: a *BandLikeList* object calculates total likelihood for (selected) set of bands.

The likelihood grid of Response objects

Sources: global, extended, point

Bands: each energy range, event type



A 10x10 grid of blue squares, representing a likelihood grid. The grid is composed of 10 rows and 10 columns of squares, all filled with a solid blue color. The grid is used to illustrate the structure of the likelihood grid for different sources and bands.

- ❑ Each cell: a *Response* object for the given source: calculates photons for each pixel, and the total in the RoI
- ❑ Each row: a *BandLike* object to calculate the likelihood for (selected) sources
- ❑ The table: a *BandLikeList* object calculates total likelihood for (selected) set of bands.

Implementation detail: views

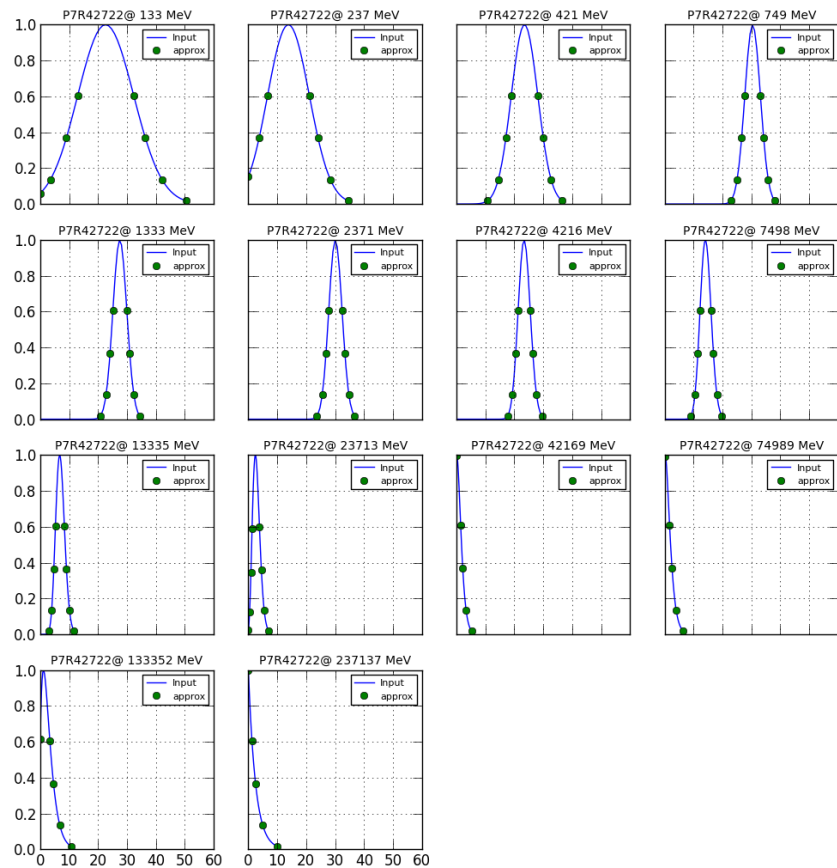
- I define functor classes that implement the “with” statement to:
 - Select all, or a subset of parameters for fitting
 - used to fit a single source
 - Select a source, evaluate likelihood as function of energy flux
 - Used for SED
 - Select a source, evaluate likelihood as function only of normalization
 - Used for profile fits
 - Select a source for optimizing position
- Encapsulate particular fitting use-cases, ensure that changes to likelihood calculating engine are restored

Parameters

- I needed an interface between the parameters managed by the source objects, and the need for a list of values, and first derivatives, to supply to the fitter
- Solution was a dedicated class containing a list of the sources
 - If it detects a change, it sets a flag in the source object, that notifies the likelihood engine to update the response – very important
 - Propagates the error matrix back to sources (losing inter-source correlations)
- A subclass manages the case where a subset of parameters are being fit

Poisson fitter

- likelihood functions are:
 - Expensive to calculate
- Generally very Poisson-like, observation from Nolan, et al., 2003, ApJ 597:615:627
- I use a fitter to take advantage of this for SED analysis, and the profile fit: evaluate the likelihood only enough to fit it, then use that function



All-sky analysis

- Optimizing sources in a single ROI is not sufficient
 - Nearby sources can contribute, but are not adjusted
 - Global sources, especially the galactic diffuse, need to be understood everywhere
- I have a pipeline using the SLAC batch system, using the Fermi interface developed for Fermi L1 processing.
 - Optimize each of the 1728 ROIs with ~300 jobs.
 - Repeat until changes are small: usually within 5 iterations
 - One important feature: do not apply full correction after first iteration: avoid loops

Summary

- I hope this was useful, but (at least) two concerns:
 - Too much detail
 - Missed something that could be important
- All code is on github, <https://github.com/tburnett/pointlike>
- I rely very heavily on a system that generates plots with documentation, web accessible. The all-sky pipeline uses this. I have a lightning talk coming up.