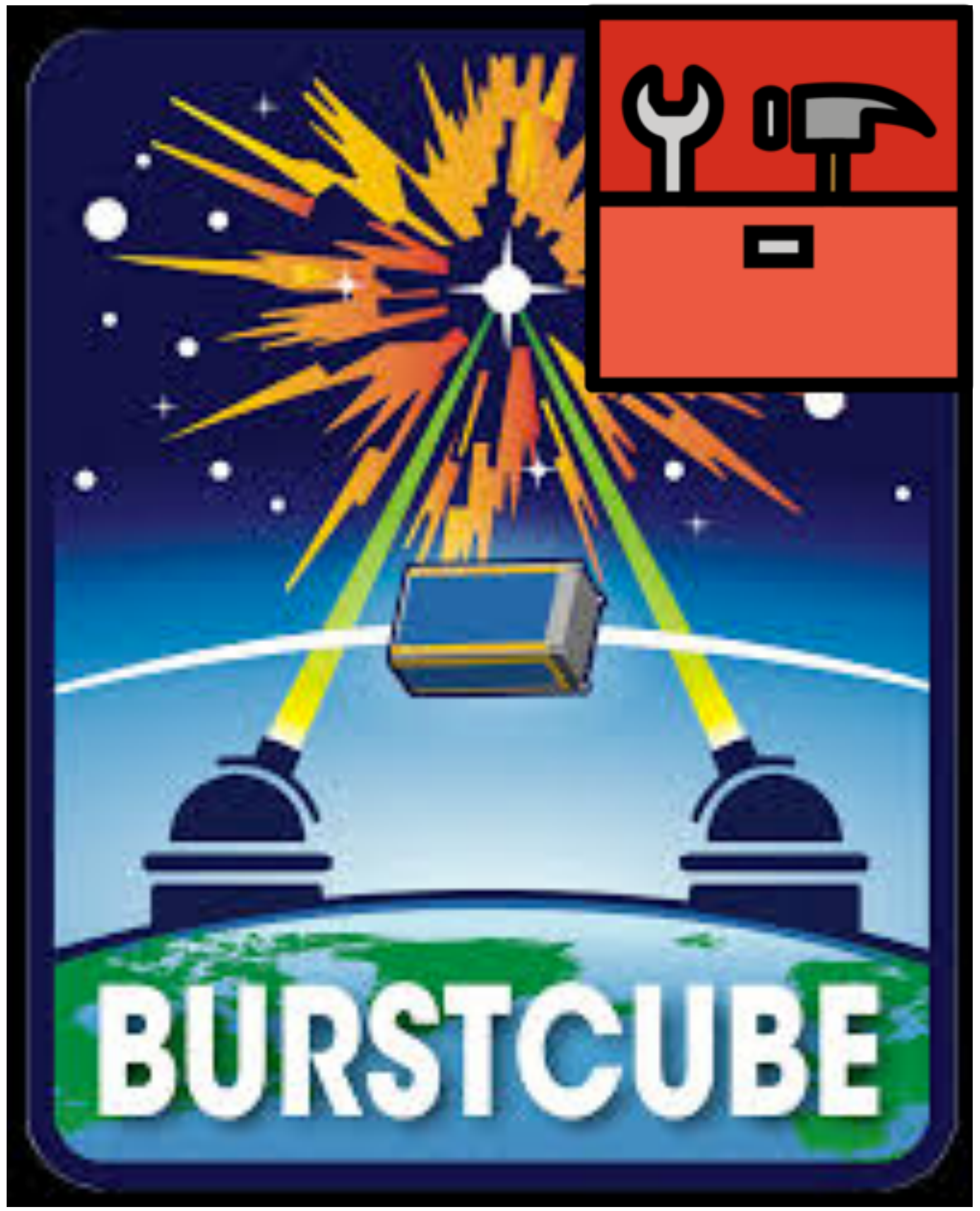
bc-tools

Israel - July 2nd, 2020 - GRB nanosats

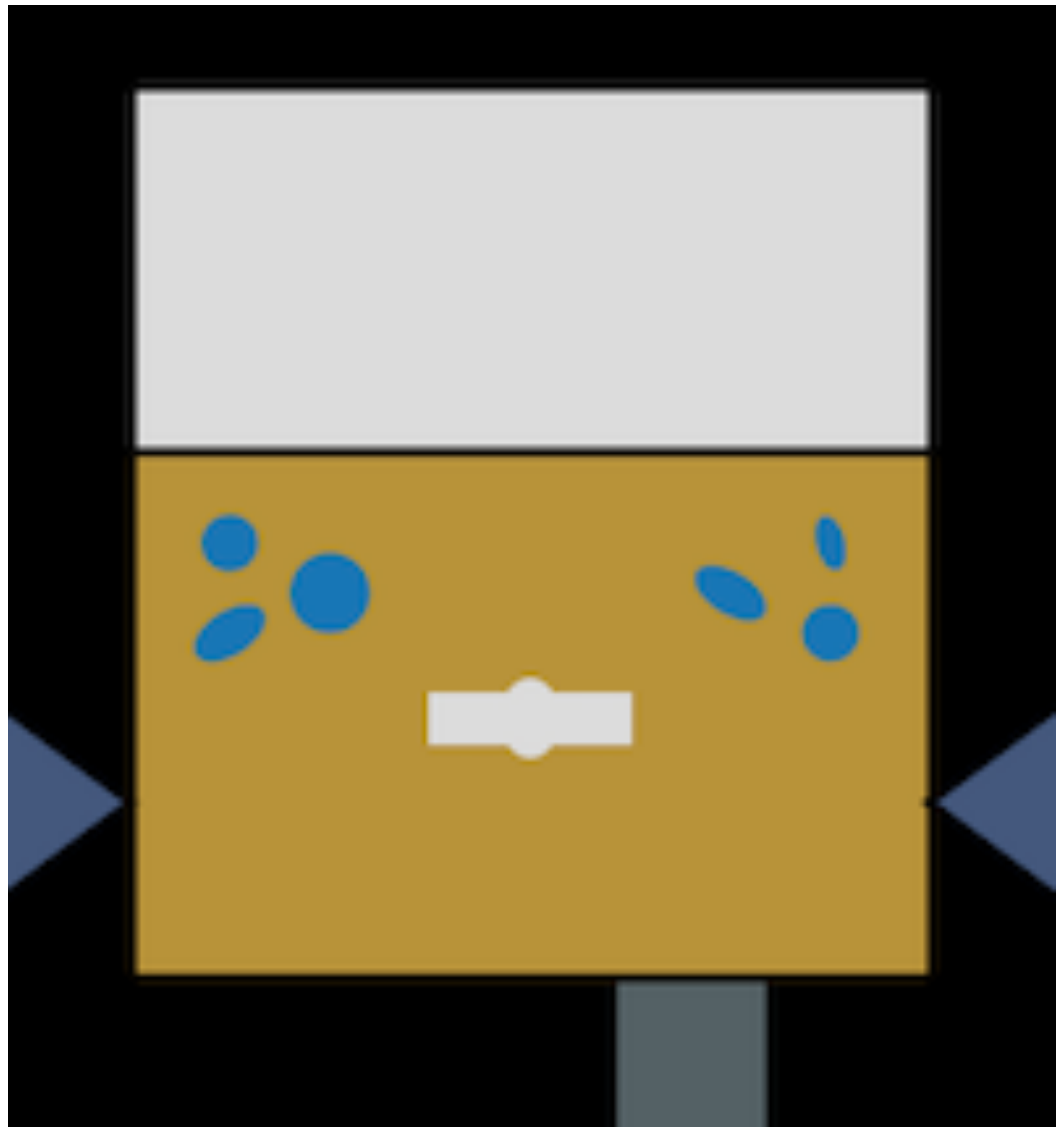
Overview

- bc-tools is BurstCube's main software package
  - Simulations
  - Analysis
- Written in Python
- Currently under development
- Built around and compatible with gbm-data-tools
- bc-tools is detector-agnostic
  - No hardcoded values
  - Easily adapted for other detectors through a configuration file



About gbm-data-tools

- Python library developed to analyzed GBM data
- Written by Adam Goldstein, William H. Cleveland and Daniel Kocevski
- Can perform most of the tasks we want in a scintillator-based gamma-ray detector:
  - Data binning and light curve generation
  - Background estimation
  - Spectral fitting
  - Source injection
- It has a great high-level API, but also well-documented access to low-level classes
- Built around existing GBM's workflow and data files.
  For each burst you have:
  - Data files with counts, such as Time-Tagged Events (TTE)
  - Detector response matrix specific for this event.
    One RSP file per detector, which you have to pair manually.

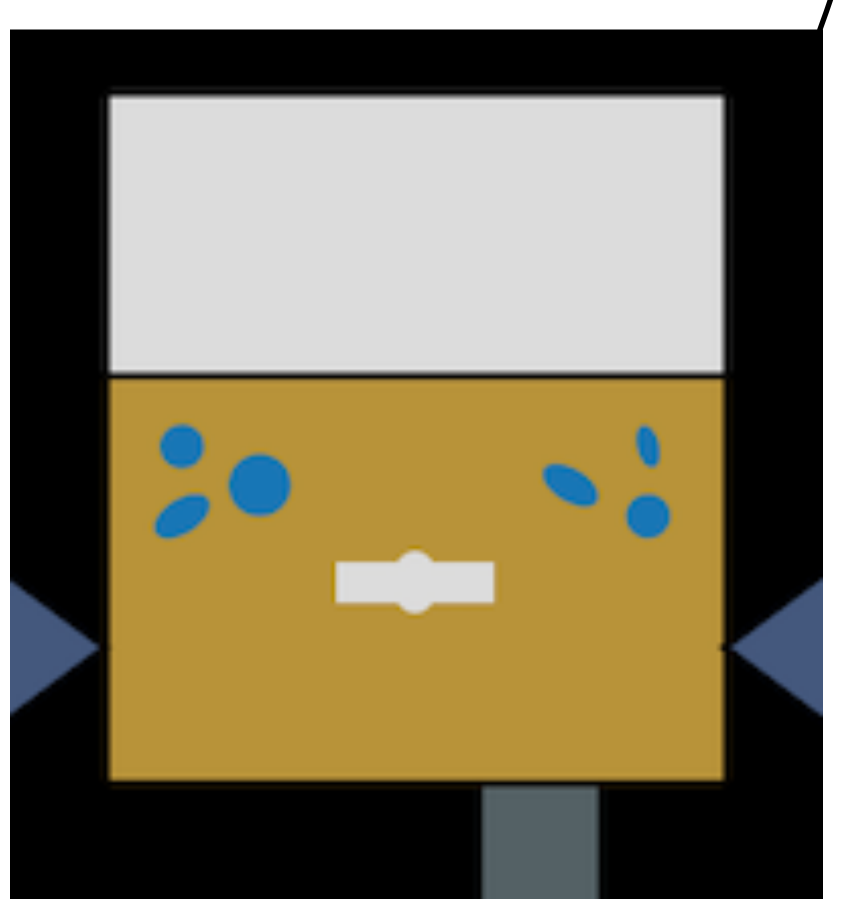


bc-tools as a detector response generator

- gbm-data-tools already does most of what we what, we plan to use it and not duplicate efforts
- BurstCube's data files are going to have the same format as GBM (FITS files)
- At first order, all we need is to generate a detector response file
  - The RSP file (also a FITS file) contains the effective area and migration matrix: real energy vs energy channels
  - Corresponds to a specific direction and to a [single] detector
  - Can contain responses for multiple time intervals (e.g. long GRB, spacecraft rocking)
- The app bc-rsp does precisely this:
  - Uses MEGALib for a particle-by-particle MC simulation
  - Build the detector response matrix (a simple 2D array)
  - This is uses to construct a GBM's RSP instantiation using the method `from_arrays()`
  - Uses `RSP.write()` to generate a file ready to be used by GBM

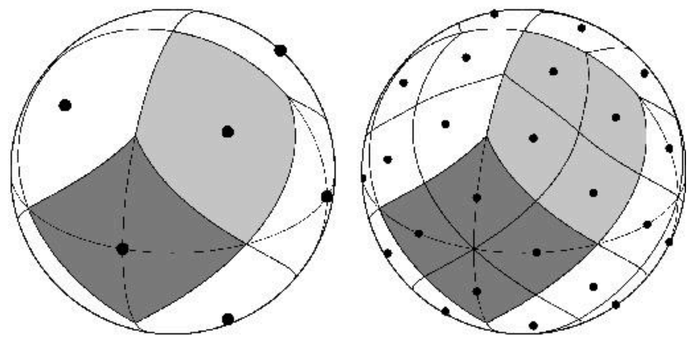
```
from gbm.data import RSP
```

.rsp



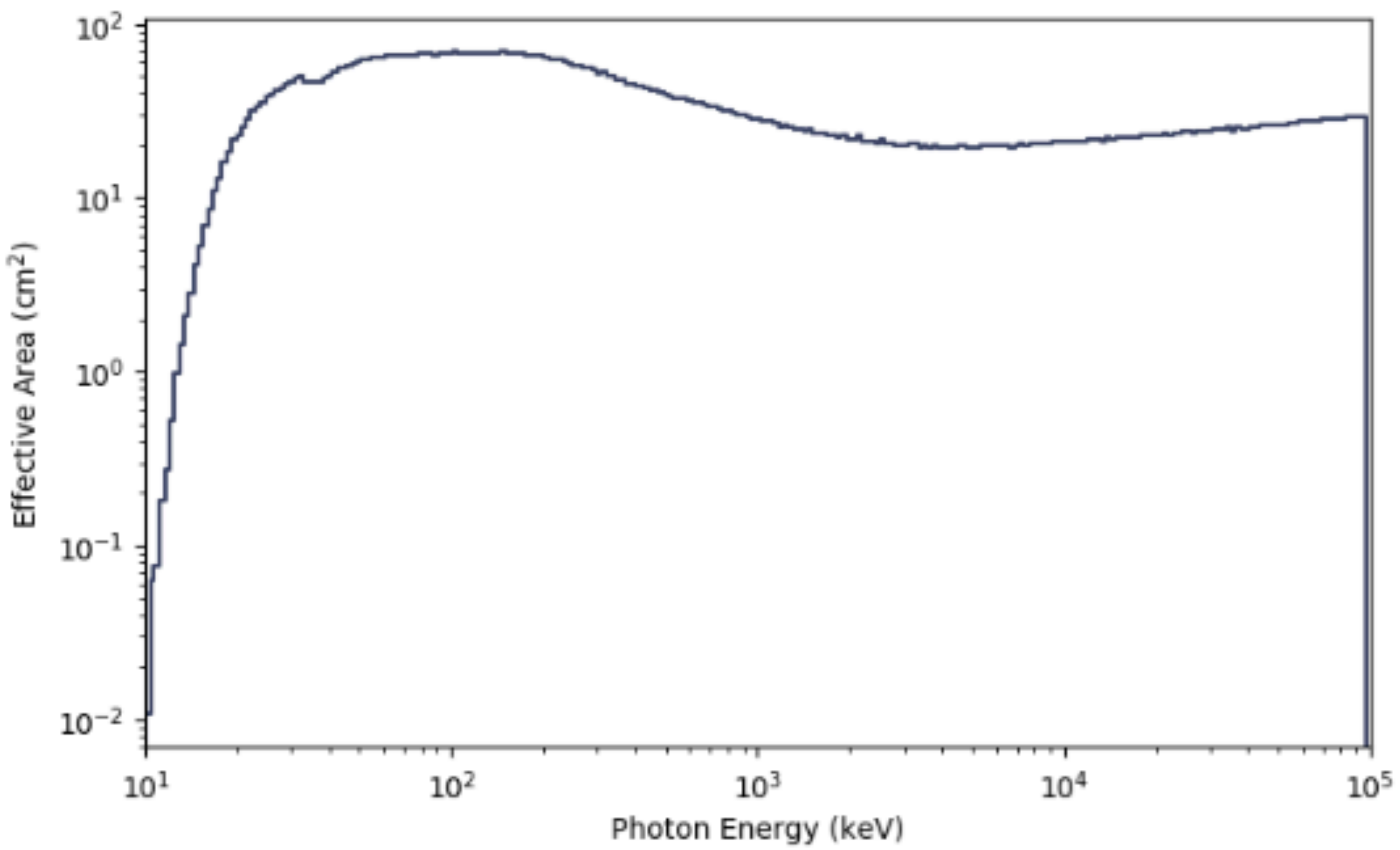
The bc-tools full detector response file

- bc-rsp not just generates a .rsp for a given direction and detector, but a full response describing the instrument as a whole
  - All detectors are included, bc-tools knows how to handle them based on their name
  - The response is computed for multiple directions in the sphere:
    - A HEALPix grid is used



    - Response for arbitrary locations are obtained through interpolation.
- This full detector response is saved into a HDF5 file (~GB size)
  - Supports partial loading, only the needed bytes are loaded into memory
- You can extract GBM's RSP objects from there and switch to using gbm-data-tools if you want:

```
from BurstCube.io import FullRSP
from BurstCube.util.coords import SpacecraftCoords
import BurstCube.util.units as u

with FullRSP(data.path/'sims/drm.h5') as frsp:

    rsp = frsp.get_rsp(detector = "SQD0",
                       coords = SpacecraftCoords(45*u.deg, 45*u.deg))


from gbm.plot import PhotonEffectiveArea

effarea_energy = PhotonEffectiveArea(data=rsp)
```

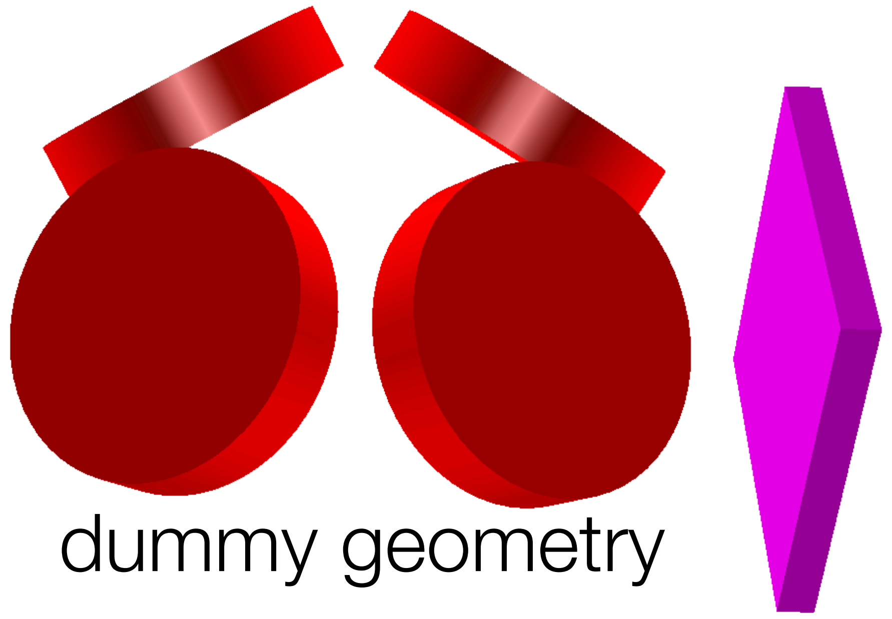


Using bc-tools on its own

- In the final design, gbm-data-tools will be under the hood
- The user will have access to the full detector response:
  - Timing and detector matching will be done automatically
  - The full response will be used for localization as well (not part of gbm-data-tools)
- Will output the results from the maximum likelihood calculations. This allows to:
  - Perform a more in-depth analysis. e.g. localization vs spectrum correlations
  - Combine the data easily with other experiments:
    - LIGO-Virgo joint analysis
    - 3ML plug-in
- The ability to get a GBM-compatible .rsp will always be there
  - Backward compatibility with software like Xspec

Build your own instrument

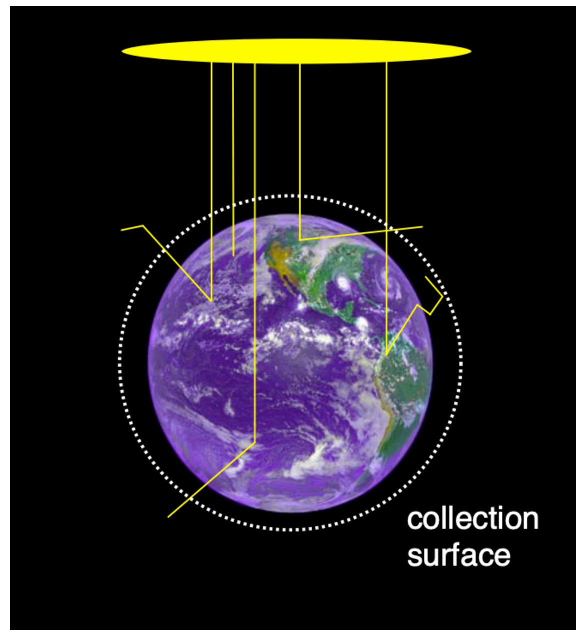
- An instrument is defined by:
  - A Geomega (MEGALib) geometry
  - Names for the single detectors
  - Calibrated instrument effects. e.g.
    - Energy resolution vs energy
    - Efficiency vs energy
- A YAML config file holds these and most other parameters
  - Job specific (e.g. random seeds) are passed through command-line options
  - Any parameter can be modified on the fly from the command-line for easy testing. e.g. `--override simulations:geometry=calib_bc.geo`


dummy geometry

```
# Example of a configuration file for BurstCube

# Note: Since some parameters are long lists, this is easier to
# read if you deactivate line wrapping e.g. less -S bc_config.yaml

# Stuff related to MEGAlib simulation
simulations:
  #Geomega file, relative to this file
  geometry: "geometry/BurstCube.geo.setup"

  # When a calorimeter is hit Cosima reports the location of the
  # center of mass. There is no ID or detector name in the .sim file.
  # This location is not trivial to compute from the geometry file, the
  # simplest way to get this locations is to run Cosima once and
  # copy-paste them here. e.g. for SDQ0 yoy get a hit entry like this:
  # HTsim 4;    4.88500;    4.88500;   -0.03536; 120.12985;0.00000e+00;2
  # This makes me sad :(
  detectors:
    location: {"4.88500, 4.88500, -0.03536": "SQD0",
               "-4.88500, 4.88500, -0.03536": "SQD1",
               "-4.88500, -4.88500, -0.03536": "SQD2",
               "4.88500, -4.88500, -0.03536": "SQD3"}

  # Spectrum of thrown particle. As long as we have narrow energy bin,
  # the result do not strongly depend on this
  # See Spectrum class for options
  # Note that the normalization is arbitrary.
  spectrum:
    name: "PowerLaw"
    args:
      norm: 1
      index: 1
```

Final remarks

- Generation of full detector response files is pretty much done
- Source localization code is the next step
- Automatic science pipelines still need to be developed
- A possible challenge: atmospheric scattering
  - The atmospheric simulation itself will not be part of bc-tools
  - We are hoping GBM's simulations are good enough for BurstCube.



- The project is public and we welcome contributions through merge requests:
  gitlab.com/burstcube/bc-tools