
# Energy dispersion tutorial (a.k.a. IRFs do not bite)

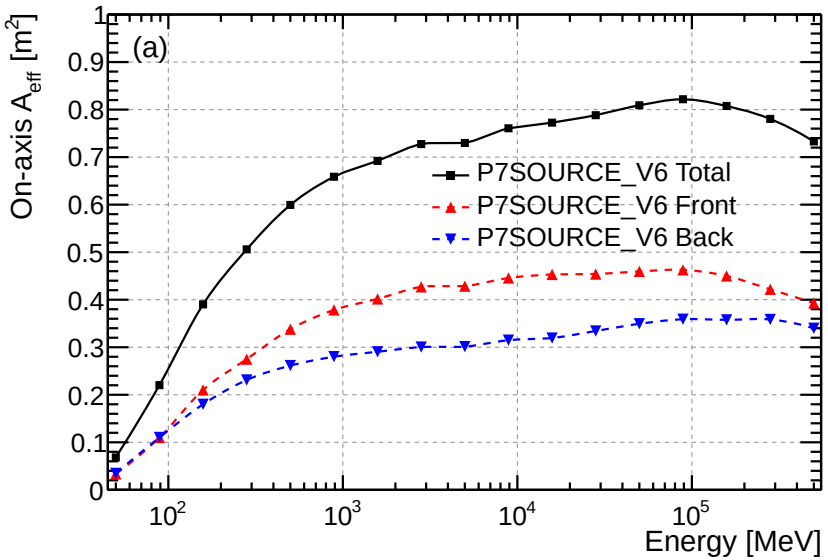
Luca Baldini
INFN–Pisa and University of Pisa
luca.baldini@pi.infn.it

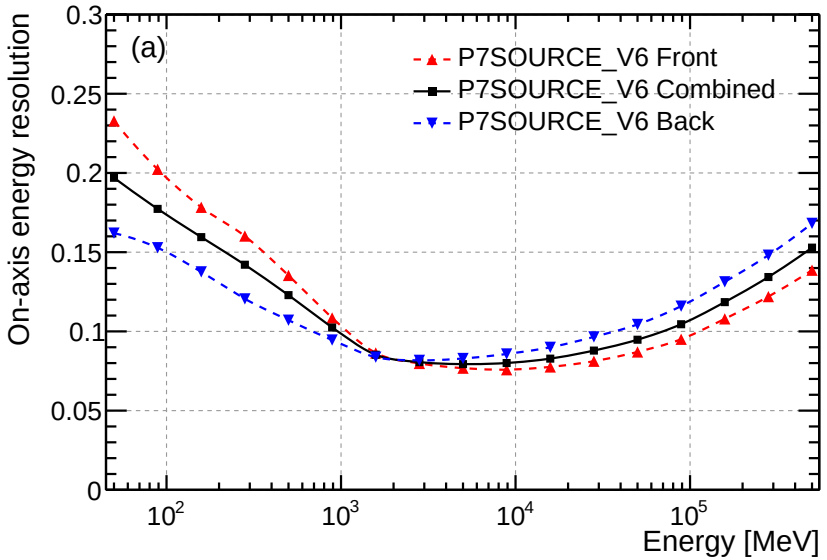
Fermi Summer School 2012
Lewes, 2012

# The name of the game

- Wouldn't it be nice if we could measure the photon energy perfectly?
- Evaluate the effect of the energy dispersion on a power-law input spectrum.
- Remember: *ScienceTools* ignore the energy dispersion by default!
- In real life this is really something you would do with *gtobbsim*:
  - all the dirty work done for you in the background;
  - you get the pointing history for free.
- This is really an excuse to dig into the IRF fits files:
  - it'll turn out to be handy if you have to create bracketing IRFs, for instance.

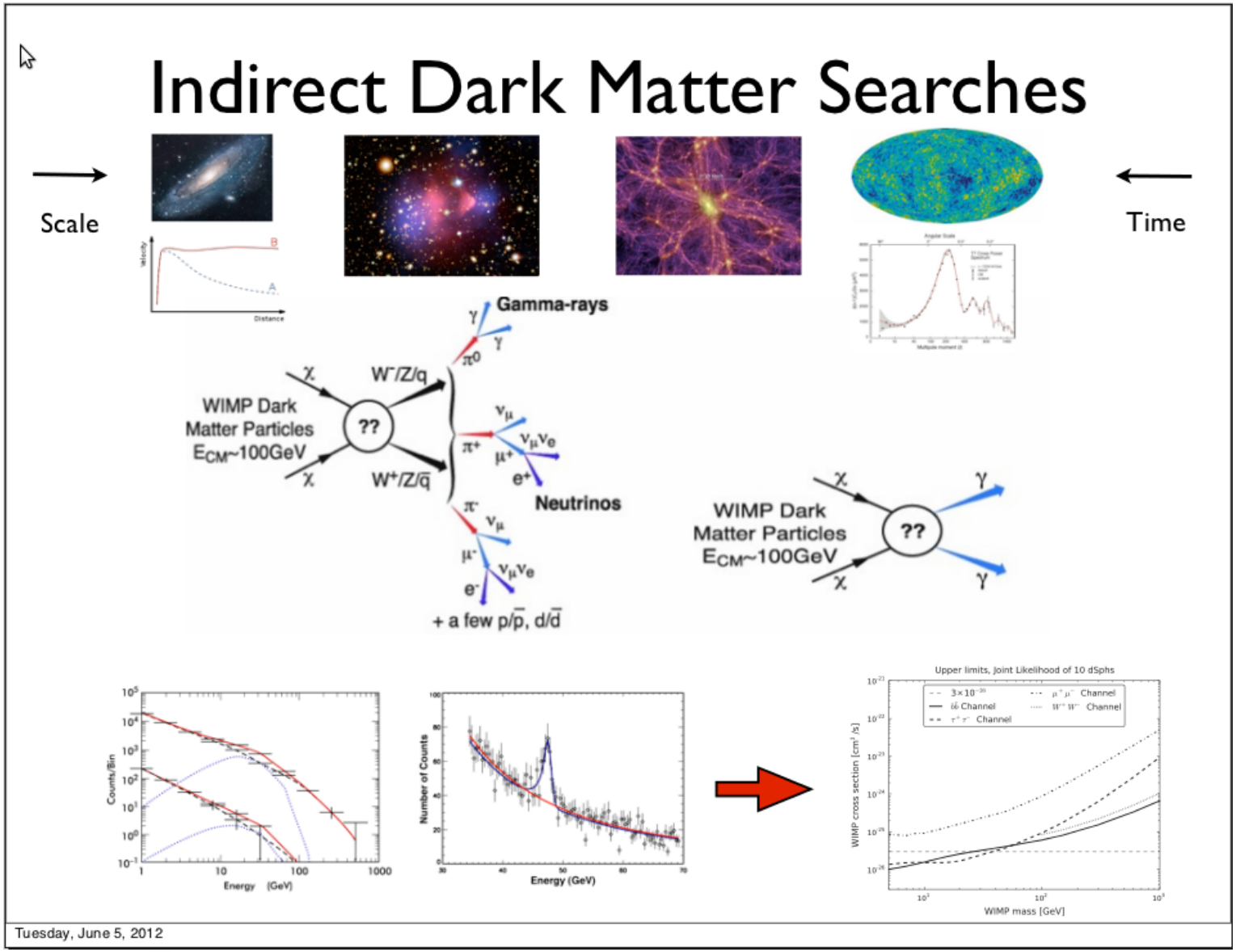
# Before we start...



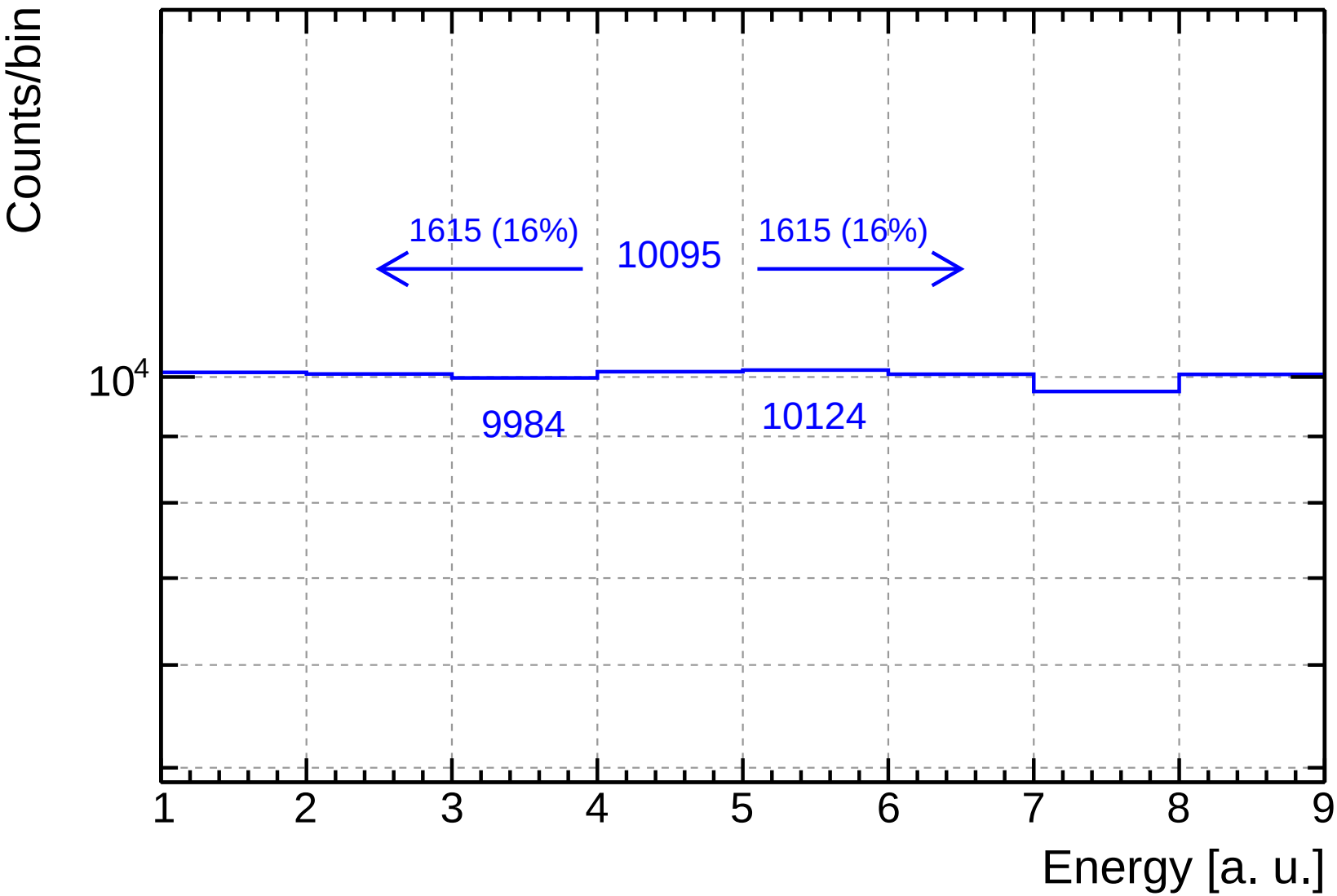

- Why the energy dispersion is (potentially) important?
  - IRFs are binned in true energy;
  - in real life you have bin-to-bin migrations in the count spectra.
- Why is the effect of the energy dispersion coupled to $A_{\rm eff}$?
- Shall we prefer low-energy or high-energy tails?
- Is the effect more pronounced for hard or steep spectra?
- Is the effect more pronounced for high or low energies?

# Why is energy dispersion important?

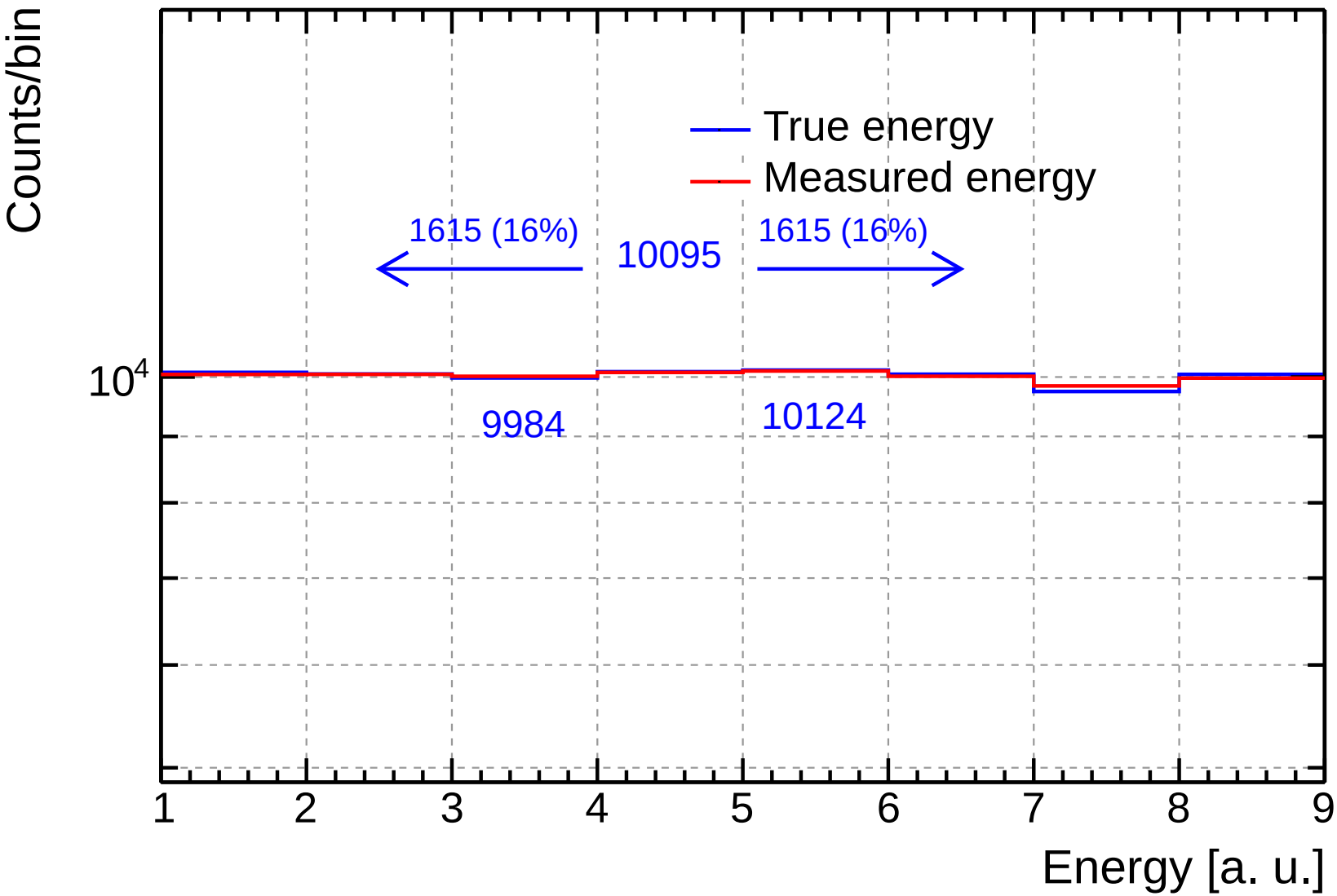



—Credits: Chris Anelli

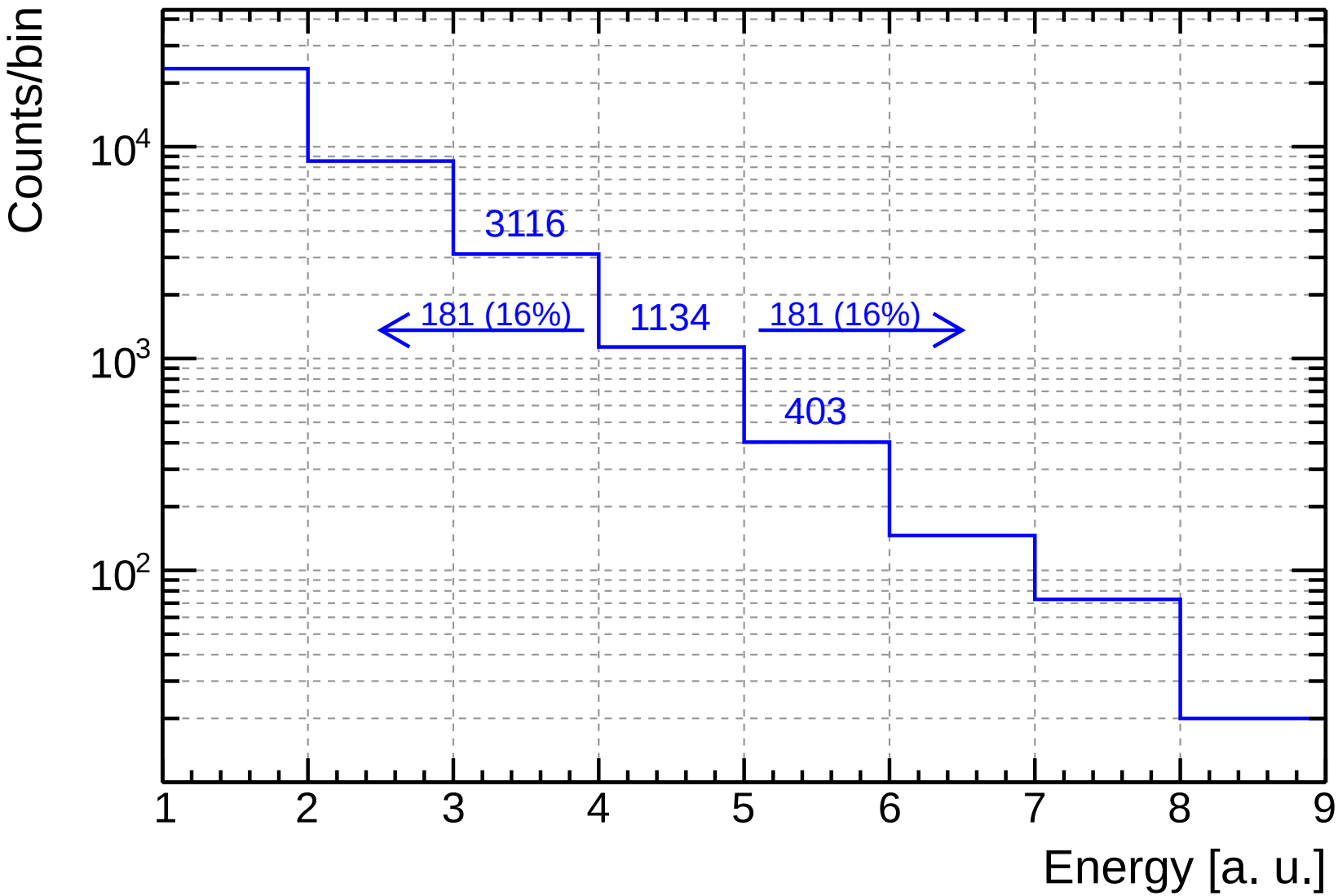
# What happens with a broadband spectrum?



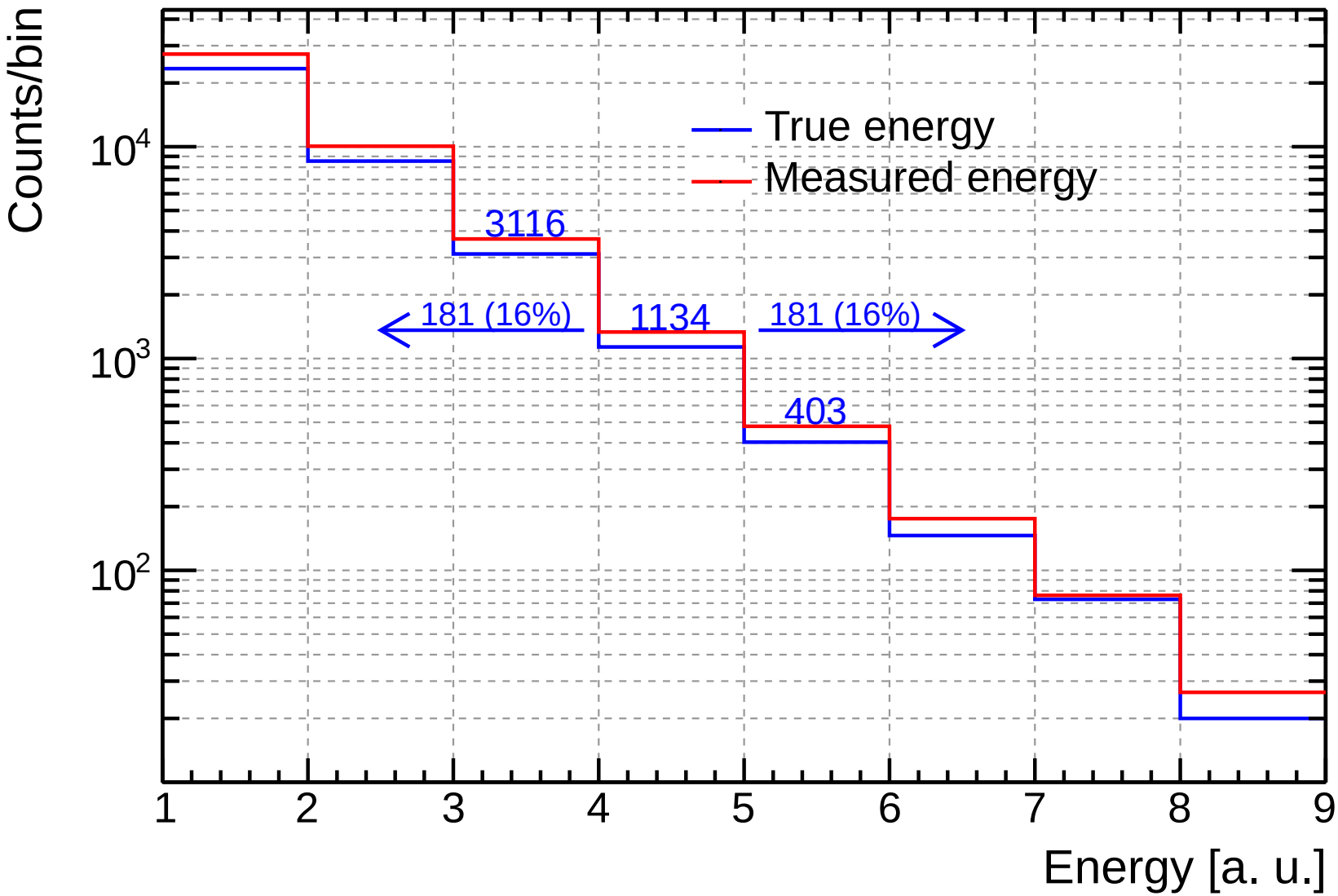
# What happens with a broadband spectrum?



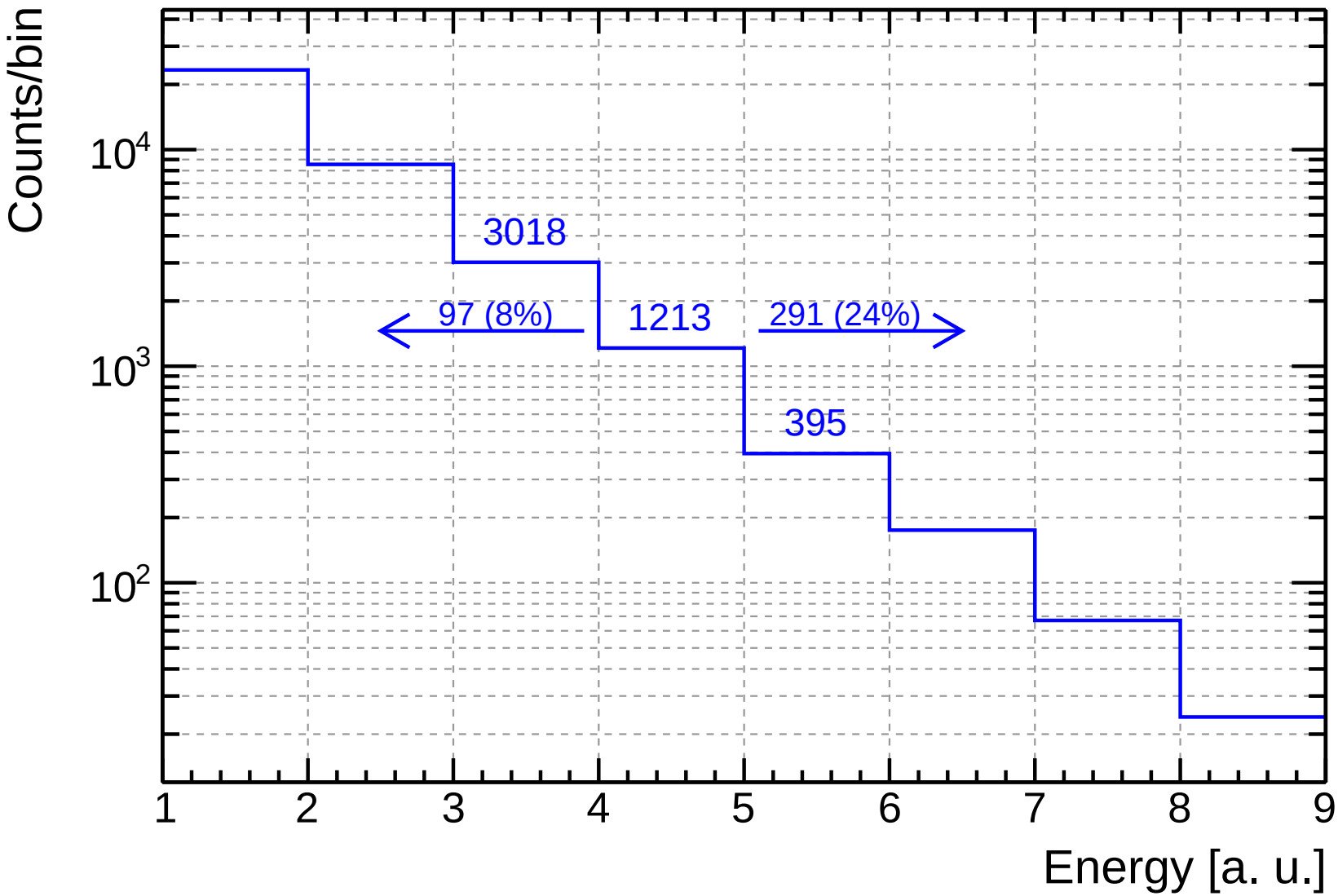
# What if the spectrum is steep?



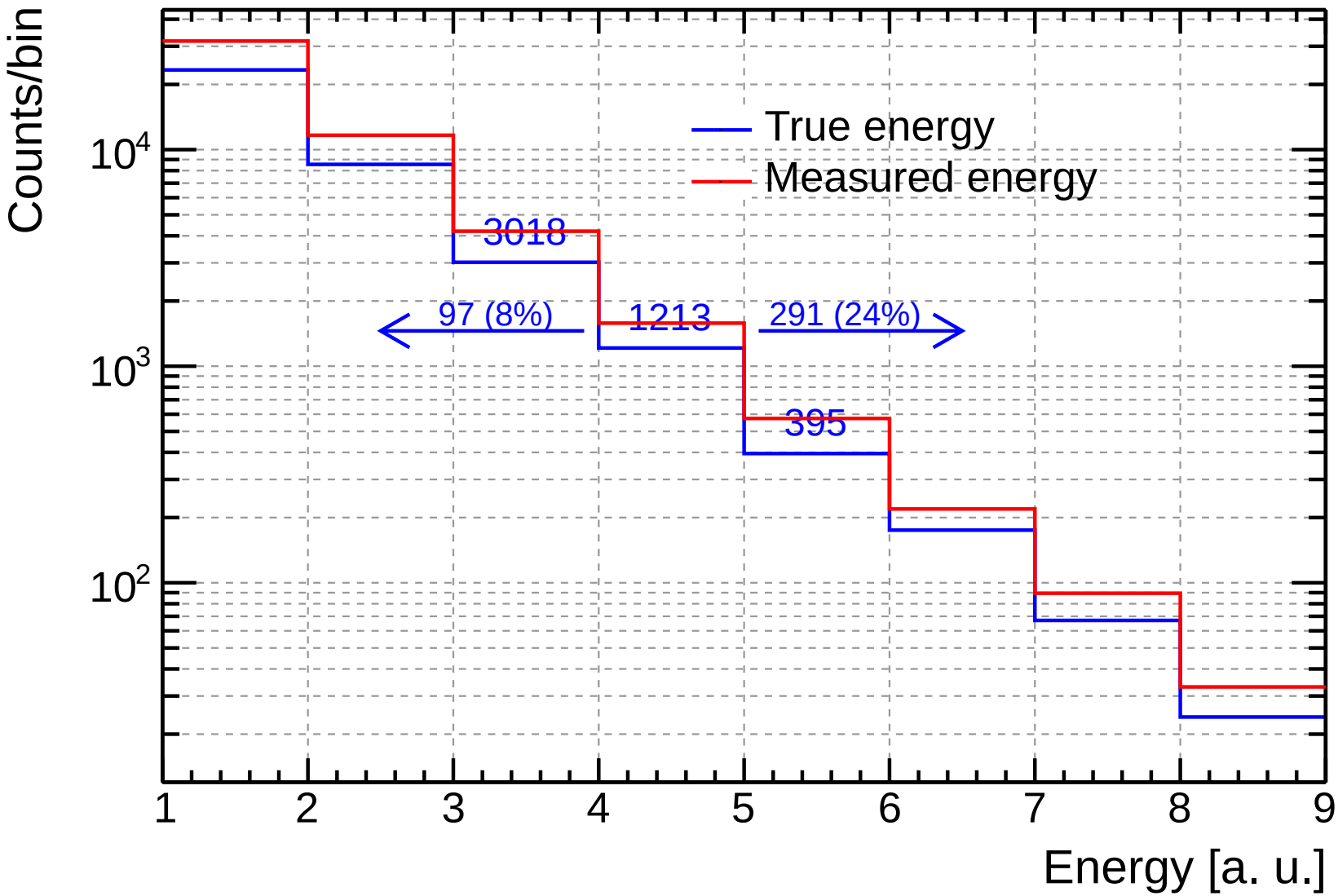
# What if the spectrum is steep?



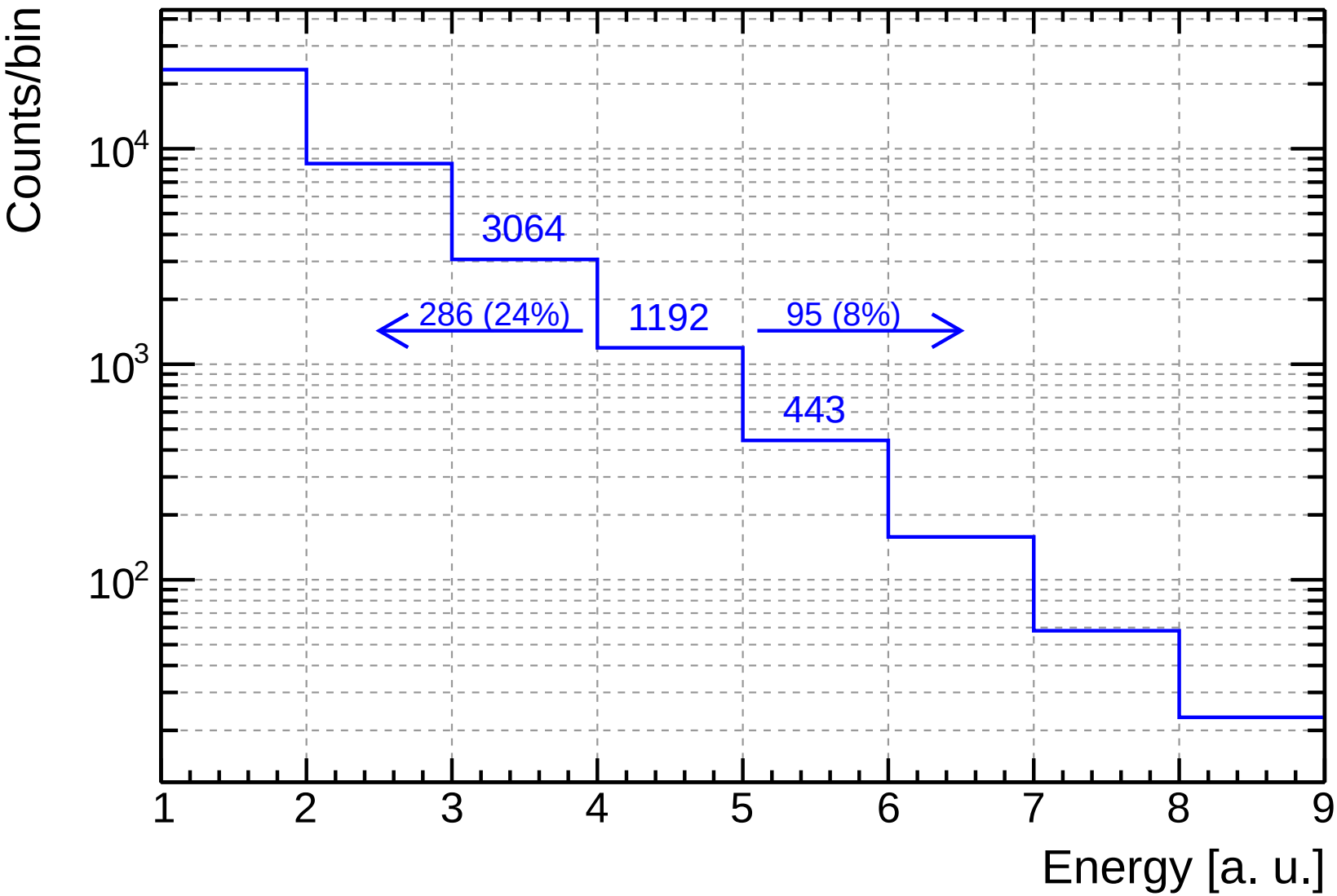
# What if the energy dispersion is asymmetric?



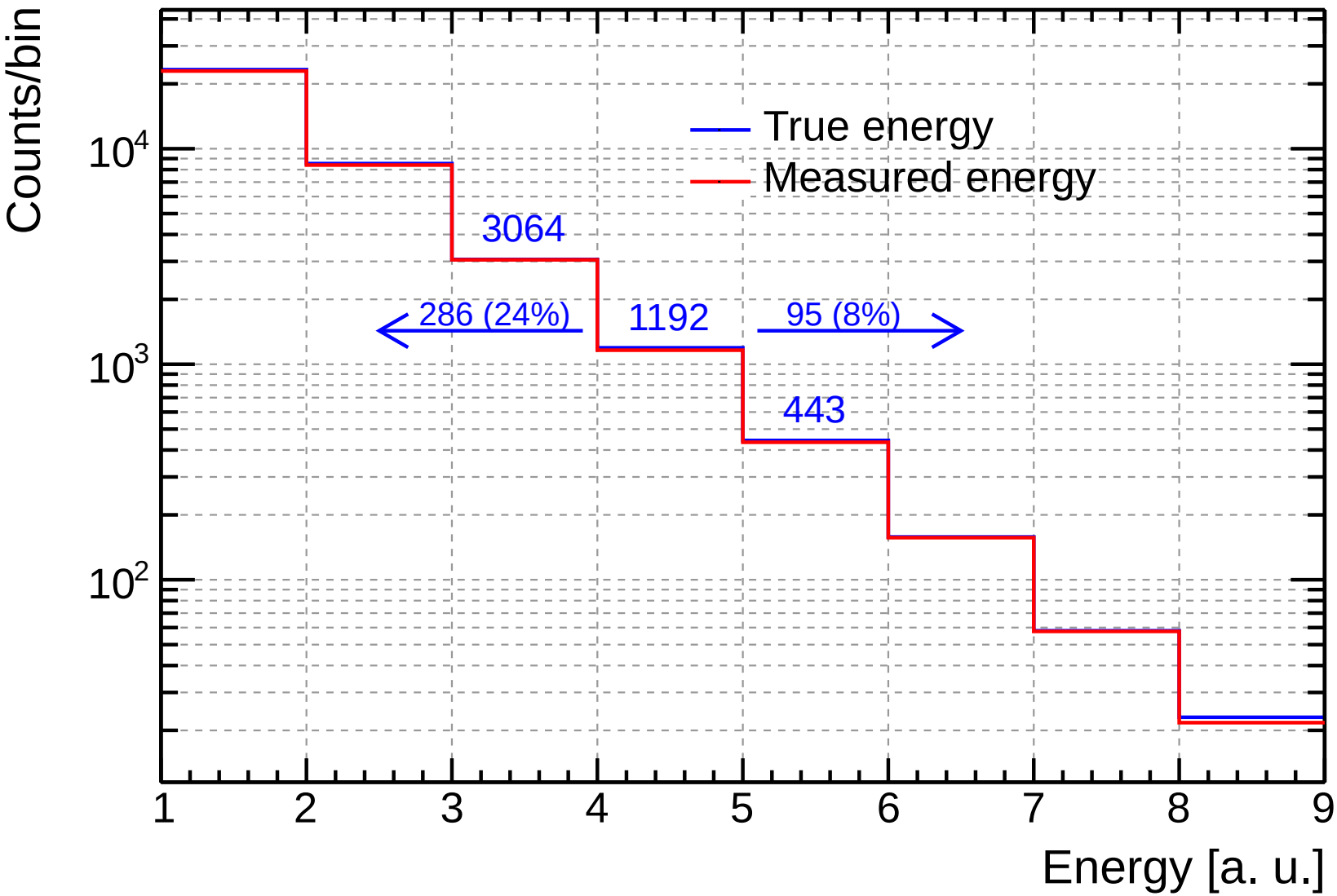
# What if the energy dispersion is asymmetric?



# What if the energy dispersion is asymmetric?



# What if the energy dispersion is asymmetric?



# Back on track

- Basic steps:
  1. generate the input spectrum (energies follow a power-law distribution with $\Gamma = 2$, directions are isotropic in the upper hemisphere in the LAT frame);
  2. fold the input spectrum with the effective area (use `P7SOURCE::FRONT` only);
  3. smear the input spectrum with the LAT energy resolution;
  4. compare the count spectra in true and measured energy.
- I'll use python as a pseudo-code, but you're welcome to use whatever you're more comfortable with!
  - Some snippets of code on

    `http://www-glast.stanford.edu/cgi-bin/viewcvs/users/lbaldini/fermiSummerSchool2012/macro/`

    (but you shouldn't use them).

Let ROOT do the dirty work behind the scenes.

Generate a power law with pyroot

```
import ROOT
E_MIN = 10.
E_MAX = 1000000.
GAMMA = 2

POWER_LAW = ROOT.TF1('fpl', 'x**(-[0])', E_MIN, E_MAX)
POWER_LAW.SetParameter(0, GAMMA)

def plRoot():
    return POWER_LAW.GetRandom()
```

# Generating the input spectrum

## If you're brave—or you don't happen to have ROOT installed

Use the inverse transform method:

$$u = \texttt{random(0, 1)} \quad x = F^{-1}(u) \tag{1}$$

The cumulative distribution for a truncated power law is

$$F(x) = \frac{E_{\rm min}^{(1-\Gamma)} - x^{(1-\Gamma)}}{E_{\rm min}^{(1-\Gamma)} - E_{\rm max}^{(1-\Gamma)}}, \tag{2}$$

(verify it!) therefore you can use:

$$x = \left[E_{\rm min}^{(1-\Gamma)} - u\left(E_{\rm min}^{(1-\Gamma)} - E_{\rm max}^{(1-\Gamma)}\right)\right]^{\frac{1}{(1-\Gamma)}} \tag{3}$$

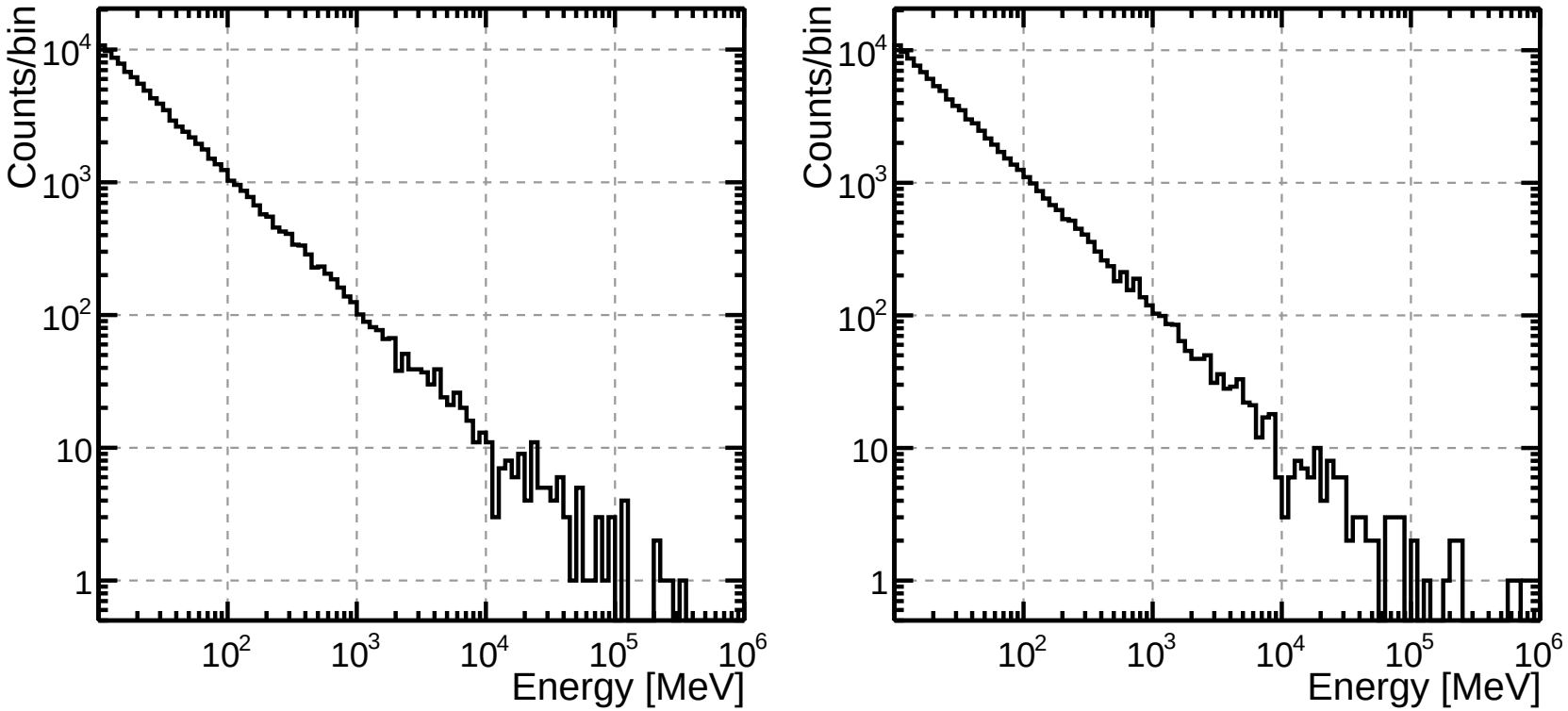
Generate a power law by the inverse transform method

```
import random
E_MIN = 10.
E_MAX = 1000000.
GAMMA = 2

def plPlain():
    u = random.random()
    x = (E_MIN**(1-GAMMA)-u*(E_MIN**(1-GAMMA)-E_MAX**(1-GAMMA)))**(1./(1-GAMMA))
    return x
```

# Testing the input spectrum

You should get something along these lines:



# Generating random directions

Standard recipe for a random point on the sphere:

$$u = \texttt{random(0, 1)}$$
$$v = \texttt{random(0, 1)}$$
$$\begin{cases} \phi = & 2\pi u \\ \theta = & \cos^{-1}(2v - 1) \end{cases} \tag{4}$$

Here we don't bother about $\phi$; also, the IRFs are tabulated in $\cos\theta$; finally we are only interested in the upper hemisphere, therefore all you need is:

$$\cos\theta = \texttt{random(0.2, 1)} \tag{5}$$

(why 0.2?)

# IRFs: the $A_{eff}$ tables

A quick look at an aeff fits file...

```
import pyfits

hdulist = pyfits.open(AEFF_FILE_PATH)
print hdulist.info()
```

...and the output should be:

```
Filename: /glast/builds/redhat5-i686-32bit-gcc41-Optimized/ScienceTools/...
No.    Name          Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU      10   ()           uint8
1    EFFECTIVE AREA  BinTableHDU     58   1R x 5C  [60E, 60E, 32E, 32E, 1920E]
2    PHI_DEPENDENCE  BinTableHDU     61   1R x 6C  [18E, 18E, 8E, 8E, 144E, 144E]
3    EFFICIENCY_PARAMS  BinTableHDU     38   4R x 1C      [6E]
```

- Three main pieces:
  1. effective area table;
  2. $\phi$ dependence correction;
  3. livetime correction.
- Effectively we only care about 1.
- And you should have your file in

  `$INST_DIR/irfs/caldb_release/CALDB/data/glast/lat/bcf/ea/aeff_P7SOURCE_V6_front.fits`

# IRFs: the $A_{\mathrm{eff}}$ tables

A quick look at the effective area table...

```
import pyfits

hdulist = pyfits.open(AEFF_FILE_PATH)
hdu = hdulist['EFFECTIVE AREA']
print 'Header data unit %s' % hdu.name
for (i, column) in enumerate(hdu.columns):
    print '%3d\t%20s\t[%6s]\t%s\n' %\
          (i, column.name, column.unit, column.format)
```

...and the output should be:

```
Header data unit EFFECTIVE AREA
  0              ENERG_LO [   MeV] 60E
  1              ENERG_HI [   MeV] 60E
  2             CTHETA_LO [      ] 32E
  3             CTHETA_HI [      ] 32E
  4               EFFAREA [    m2] 1920E
```

- So we get:
  - the energy binning (60 bins);
  - the $\cos\theta$ binning (32 bins);
  - the actual effective area values ($60 \times 32 = 1920$ values).

# IRFs: the $A_{eff}$ tables

A quick look at the effective area table values...

```
import pyfits

hdulist = pyfits.open(AEFF_FILE_PATH)
data = hdulist['EFFECTIVE AREA'].data[0]
print data.field('ENERG_LO')
print data.field('ENERG_HI')
print data.field('CTHETA_LO')
print data.field('CTHETA_HI')
print data.field('EFFAREA')
# Retrieve the value of the i-th energy bin and the j-th cos theta bin.
# (You might or not need the ``flatten()'', here).
nebins = 60
i = 10
j = 22
print data.field('EFFAREA').flatten()[i + nebins*j]
```

The output is too long but you can imagine how it looks.

# IRFs: parsing the $A_{eff}$ table...

## ...and sticking it into a ROOT histogram

Read an effective area table and create a TH2F

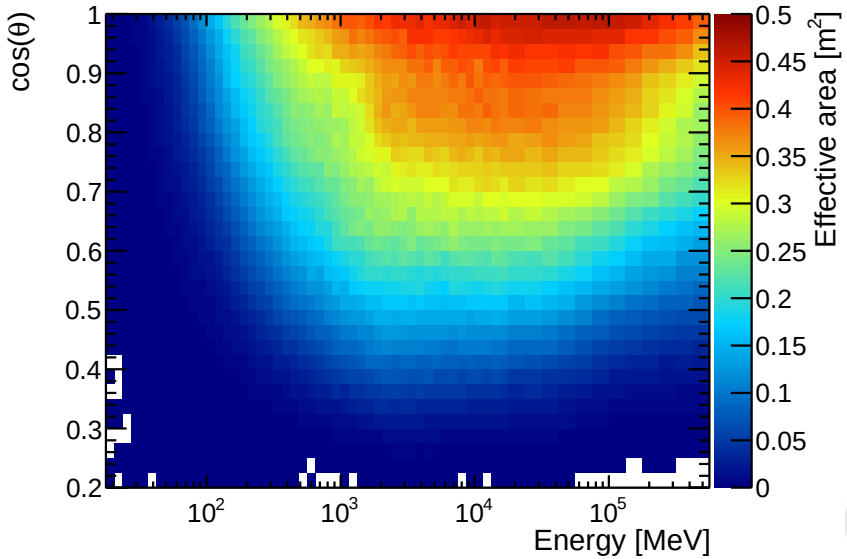
```
# Open the file and retrieve the data.
hdulist = pyfits.open(AEFF_FILE_PATH)
data = hdulist['EFFECTIVE AREA'].data[0]

# Create the arrays for the histogram binning.
xbins = numpy.hstack((data.field('ENERG_LO'), data.field('ENERG_HI')[-1]))
nx = len(xbins) - 1
ybins = numpy.hstack((data.field('CTHETA_LO'), data.field('CTHETA_HI')[-1]))
ny = len(ybins) - 1

# Create and fill the histogram.
h = ROOT.TH2F('aeff', 'Effective area', nx, xbins, ny, ybins)
for i in xrange(nx):
    for j in xrange(ny):
        aeff = data.field('EFFAREA').flatten()[i + nx*j]
        h.SetBinContent(i + 1, j + 1, aeff)
```

- You can use your favorite language/framework/data structure to store the $A_{eff}$ values (and retrieve them!).

# Testing the $A_{\mathrm{eff}}$ representation




- Your effective area table should look more or less like this.
  - Note that the bin width is not constant in $\log(E)$—in case you are tempted to find the bin corresponding to a given $E$ with one division;
  - If you use ROOT you can rely on the `ROOT::TH2F::Interpolate()` method.

# Energy dispersion recap

First we define the *scaled energy deviation*:

$$x = \frac{1}{S_D(E,\theta)} \frac{(E' - E)}{E} \tag{6}$$

where

$$S_D(E,\theta) = c_0(\log_{10} E)^2 + c_1(\cos\theta)^2 + c_2 \log_{10} E + c_3 \cos\theta + \\ + c_4 \log_{10} E \cos\theta + c_5. \tag{7}$$

Then, in each energy/angle bin we fit the distribution of the scaled deviation with four piecewise Rando functions:

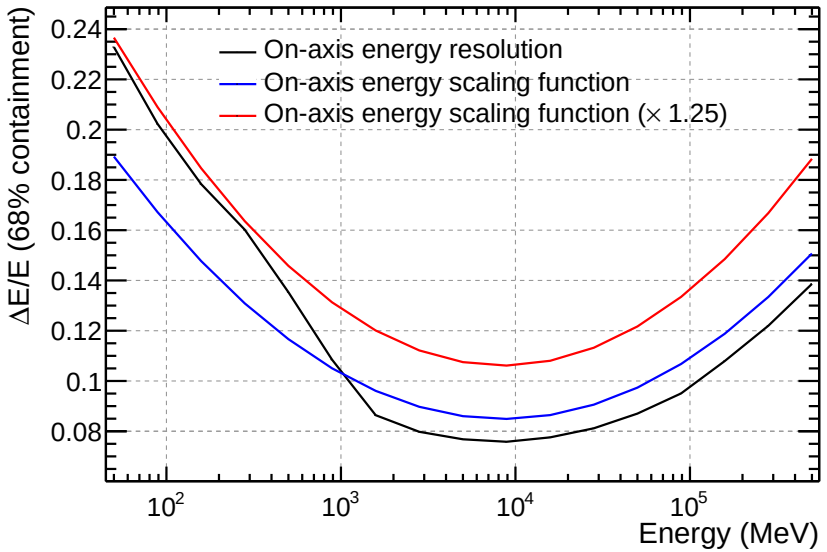
$$R(x, x_0, \sigma, \gamma) = N \exp\left(-\frac{1}{2}\left|\frac{x - x_0}{\sigma}\right|^{\gamma}\right). \tag{8}$$

All the parameters are stored in the fits files, so in principle you can reverse-engineer that (and you're very welcome to do so).

# A cheap proxy for the energy dispersion

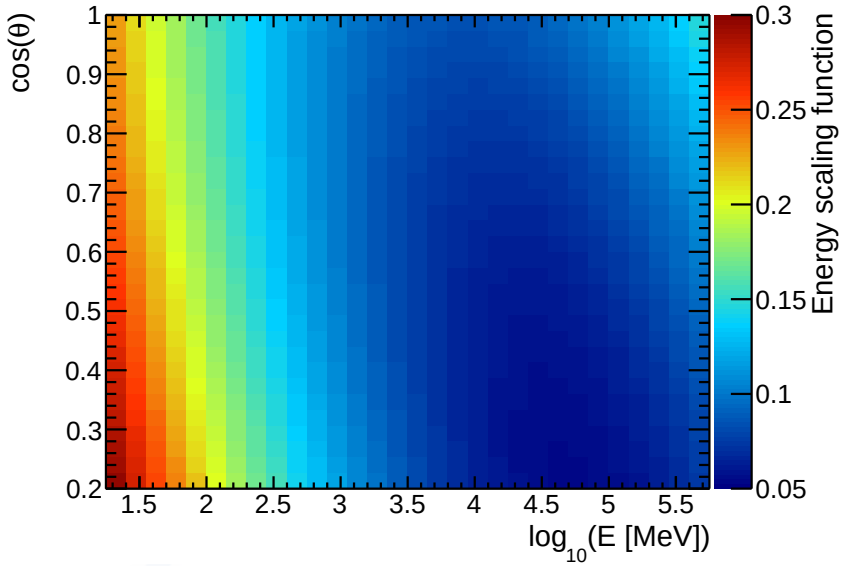
- Remember: the scaling function is defined with the goal of making the energy dispersion as independent as possible from energy and angle:
  - if the energy dispersion was Gaussian...
  - ...and the prescaling function captured all the richness of the energy dispersion itself...
  - then we wouldn't need to perform the fitting step: the scaled energy deviation would be $Norm(0, 1)$ and the value of the scaling function would be the energy resolution.
- For the purpose of this exercise (at least to start with) we'll cheat (twice):
  - take the energy scaling function as a proxy for the energy resolution;
  - assume the energy dispersion is Gaussian.
- The parameters of the energy scaling function (7) are stored in the edisp fit file as the first 6 numbers of the `'EDISP_SCALING_PARAMS'` HDU.

# How much of a cheat is cheating?




- A little bit, but not too much;
  - the general trend is well reproduced;
  - we do underestimate the energy resolution at low energy;
  - (this is actually the most interesting part, so you might as well multiply the scaling function by $\sim 1.25$);
  - and remember we are underestimating the tails (you might even multiply by $\sim 1.5$).

# Testing the edisp representation




- Your energy scaling function should look more or less like this.

# Now we're ready to go!

poor man's attempt at putting everything together

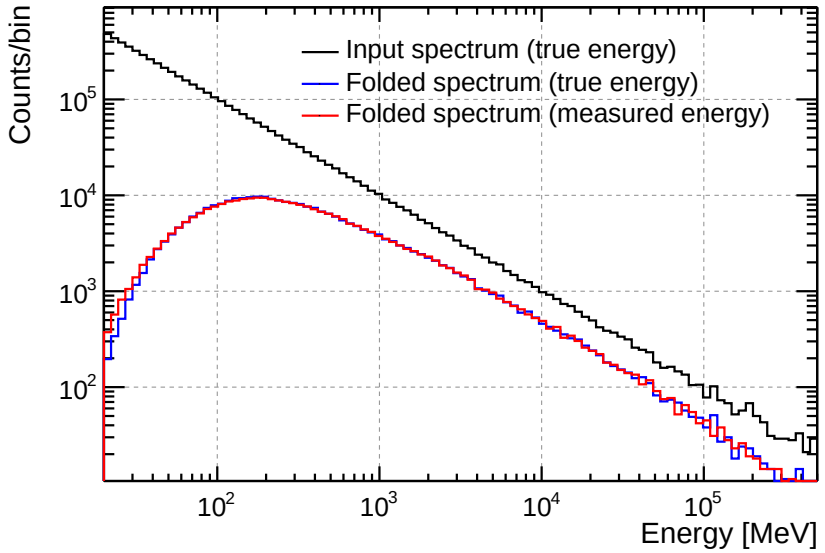
```
import random

MAX_EFF_AREA = 0.5 # put the maximum of the effective area, here.

for i in xrange(NUM_EVENTS):
    # Extract energy and direction
    energy = ... # use your function, here
    costheta = random.uniform(0.2, 1)
    # Convolve with the effective area.
    aeff = ... # retrieve your effective area at the right energy/angle.
    ... # do your book-keping if needed.
    if random.random() < aeff/MAX_EFF_AREA:
        eres = ... # get the value of the scaling function.
        measuredEnergy = energy*random.gauss(1, 1.5*eres)
        ... # more book-keeping.
```

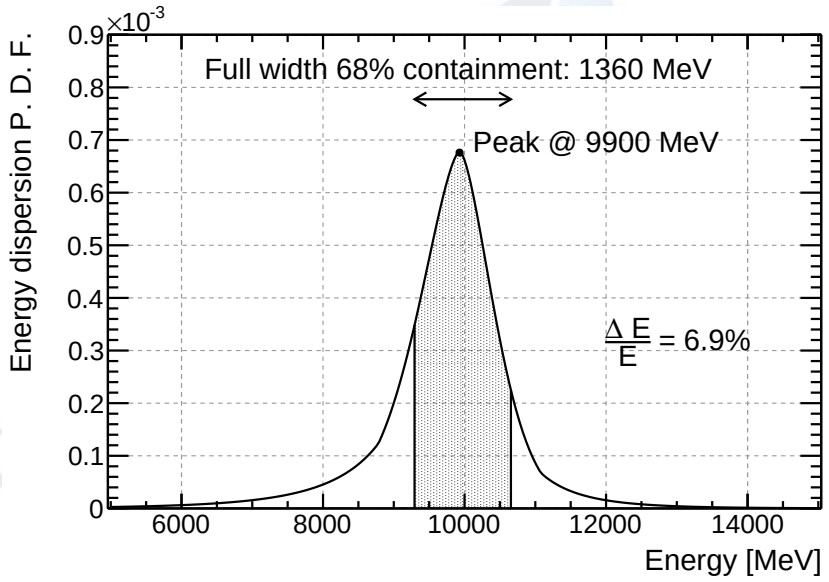
Fill in the blanks...

# And finally...




- The difference between blue and red is what you're after:
  - plot the ratio of the two histograms.
- You can change the input spectral index!
  - Things get better or worst with harder spectra? Why?
- And you can change the energy scale too!
  - Can you explain analytically what happens above a few GeV?

# One last thing




- It would be totally awesome if you plotted the actual energy dispersion, say at 10 GeV on axis;
  - (or at your favorite energy/angle)!

# Some directions...

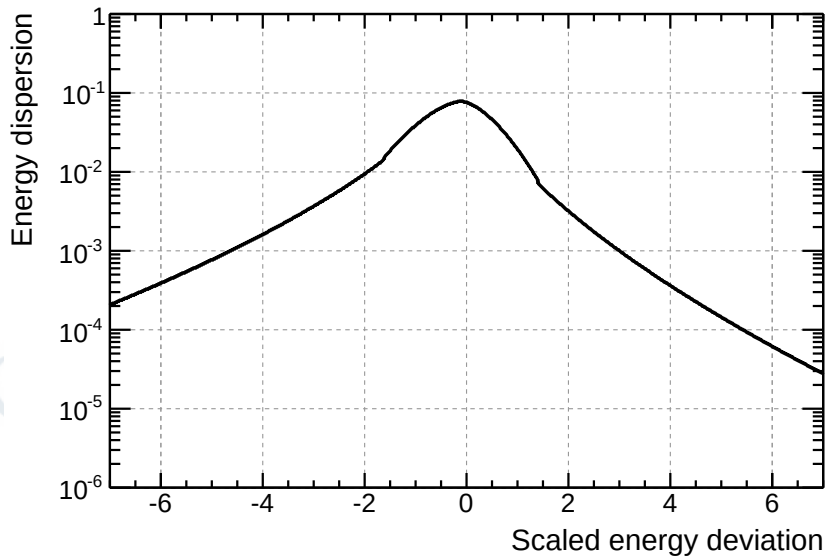
$$R(x, x_0, \sigma, \gamma) = N \exp\left(-\frac{1}{2}\left|\frac{x - x_0}{\sigma}\right|^{\gamma}\right) \tag{9}$$

$$D(x) = \begin{cases} N_L R(x, x_0, \sigma_L, \gamma_L) & \text{if } (x - x_0) < -\tilde{x} \\ N_l R(x, x_0, \sigma_l, \gamma_l) & \text{if } (x - x_0) \in [-\tilde{x}, 0] \\ N_r R(x, x_0, \sigma_r, \gamma_r) & \text{if } (x - x_0) \in [0, \tilde{x}] \\ N_R R(x, x_0, \sigma_R, \gamma_R) & \text{if } (x - x_0) > \tilde{x} \end{cases} \tag{10}$$

| $\tilde{x}$ | $\gamma_L$ | $\gamma_l$ | $\gamma_r$ | $\gamma_R$ |
|---|---|---|---|---|
| 1.5 | 0.6 | 1.6 | 1.6 | 0.6 |

Find in the FITS files: the normalization $N_r = N_l$ (`NORM`), the centroid position $x_0$ (`BIAS`), the two core scales $\sigma_r$ (`RS1`) and $\sigma_l$ (`LS1`) and the two tail scales $\sigma_R$ (`RS2`) and $\sigma_L$ (`LS2`).

# Some directions...




- And then you have to un-prescale...