# Pysynphot: A Python Re-Implementation of a Legacy App in Astronomy

Vicki Laidler[1], Perry Greenfield, Ivo Busko,
Robert Jedrzejewski
Science Software Branch

Space Telescope Science Institute
Baltimore, MD
[1]Computer Sciences Corporation, STScI

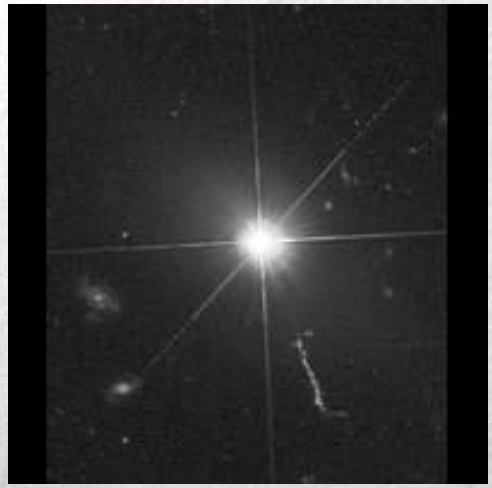
# What is synthetic photometry?

- An attempt to simulate the expected detected count rate of photons given a model for the:
  - Light from the source (star or galaxy) as a function of wavelength (a *spectrum*)
  - Telescope, instrument, and detector
- A telescope collects light and magnifies it, but each time the light passes through a lens or bounces off a mirror, a little bit of light is lost.
- Filters deliberately exclude some wavelengths of light to look at scientifically interesting domains

Spectrum * (Optics) = Different spectrum

# Why?

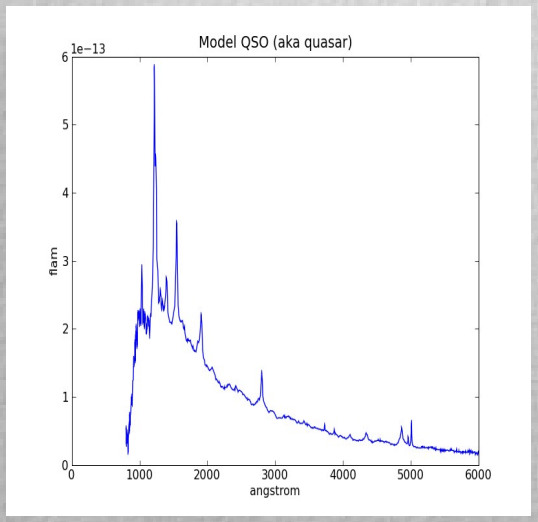
- To help plan observations
  - How long do I need to observe to find what I expect?
  - Or what's the faintest source I can see in the time I have?
- For calibration
  - Comparing the models vs observations of calibration sources
  - Used to update the models
    - In turn used to calibrate science data



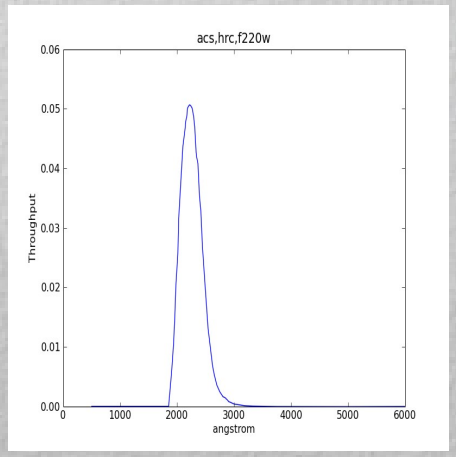
*

=

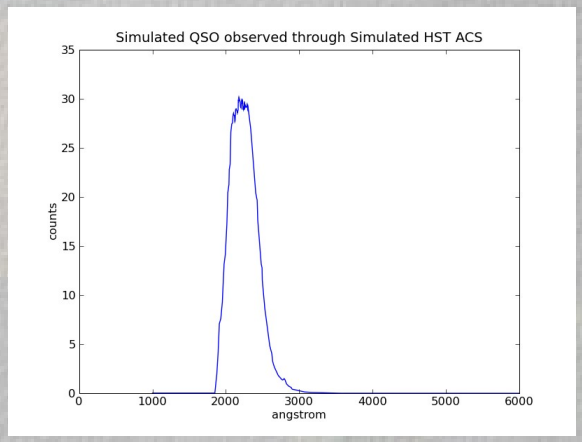

The
ASTROPHYSICAL
JOURNAL


Model QSO (aka quasar)
1e−13
flam
angstrom

*


acs,hrc,f220w
Throughput
angstrom

=


Simulated QSO observed through Simulated HST ACS
counts
angstrom

# The Legacy Application: SYNPHOT

- Written in early 90's within the IRAF system.
  - Written in the SPP language
    - Never heard of SPP?
- Task-oriented system
- SYNPHOT is a package of tasks that generally have many parameters, use files as inputs, and print their results or generate new files
  - Hard to use lower-level functionality
  - Has its own mini language for spectrum expressions
- IRAF is dying
- Need to rewrite SYNPHOT
  - Chance to fix algorithmic problems

# Problems? What problems?

Some intrinsic to original architecture, some later identified:

- Single-precision arithmetic
- Everything is an array
  - Regardless of wavelength sampling choices some spectral features are poorly sampled
- No memory caching
- Steep learning curve for mini-language
- Poor handling of redshifts

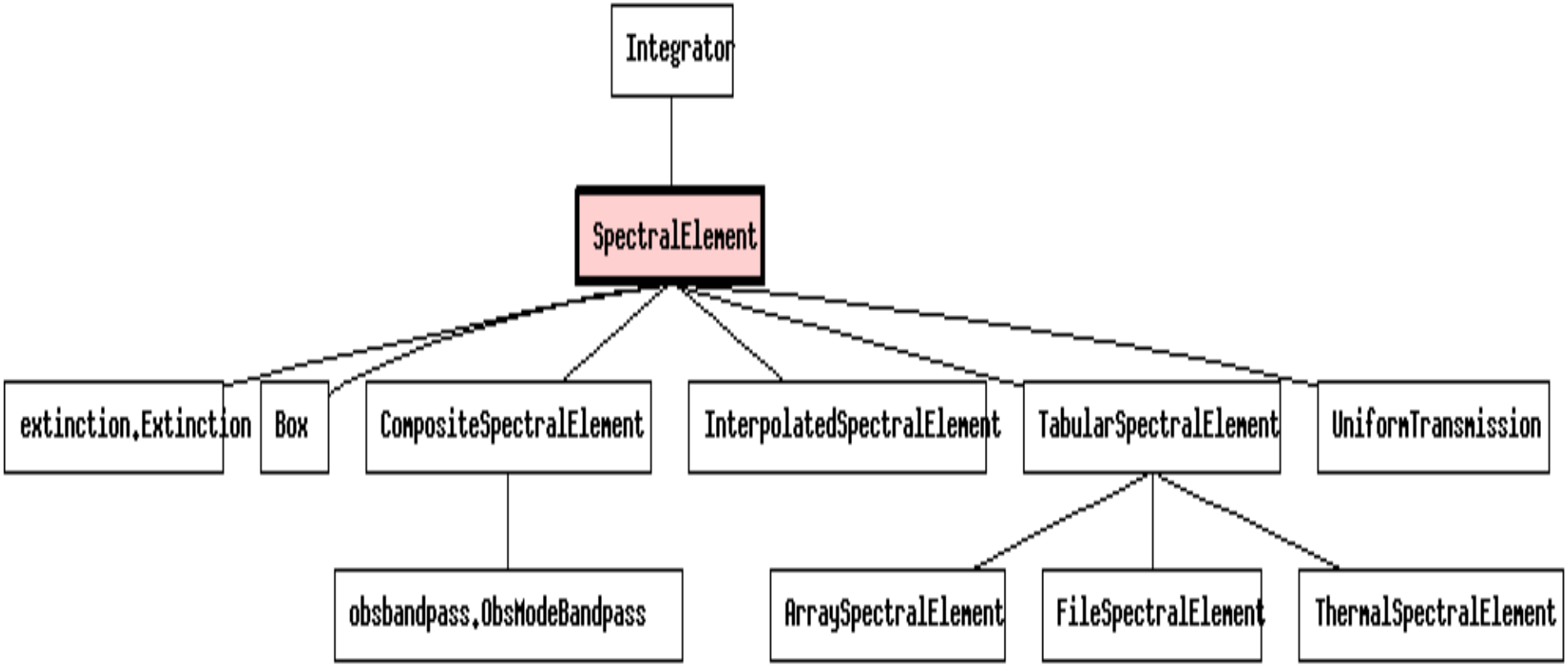
# Solutions

- Use double precision arithmetic
- Take advantage of Object-Oriented features
- Avoid lots of file-to-file operations.
- Objects as “building blocks” with easy-to-use UI
- Remain flexible on wavelength sampling
  - Defer wavelength choices as long as possible
- Leave plotting and array manipulation to other software libraries.
- Use Python, of course

# Pysynphot Objects

- **SpectralElement** (model telescope, lens or mirror)
  - Know how to multiply themselves with other optics (to produce a CompositeSpectralElement (aka Bandpass) object
  - Don't permit addition with other optics
  - Know how to sample themselves appropriately in wavelength to capture all important structure.
  - Know how to evaluate themselves at arbitrarily specified wavelengths (including arrays of wavelengths)
  - Analytic or table-driven models permitted
  - FITS files (astronomer-standard data file format) used for table models

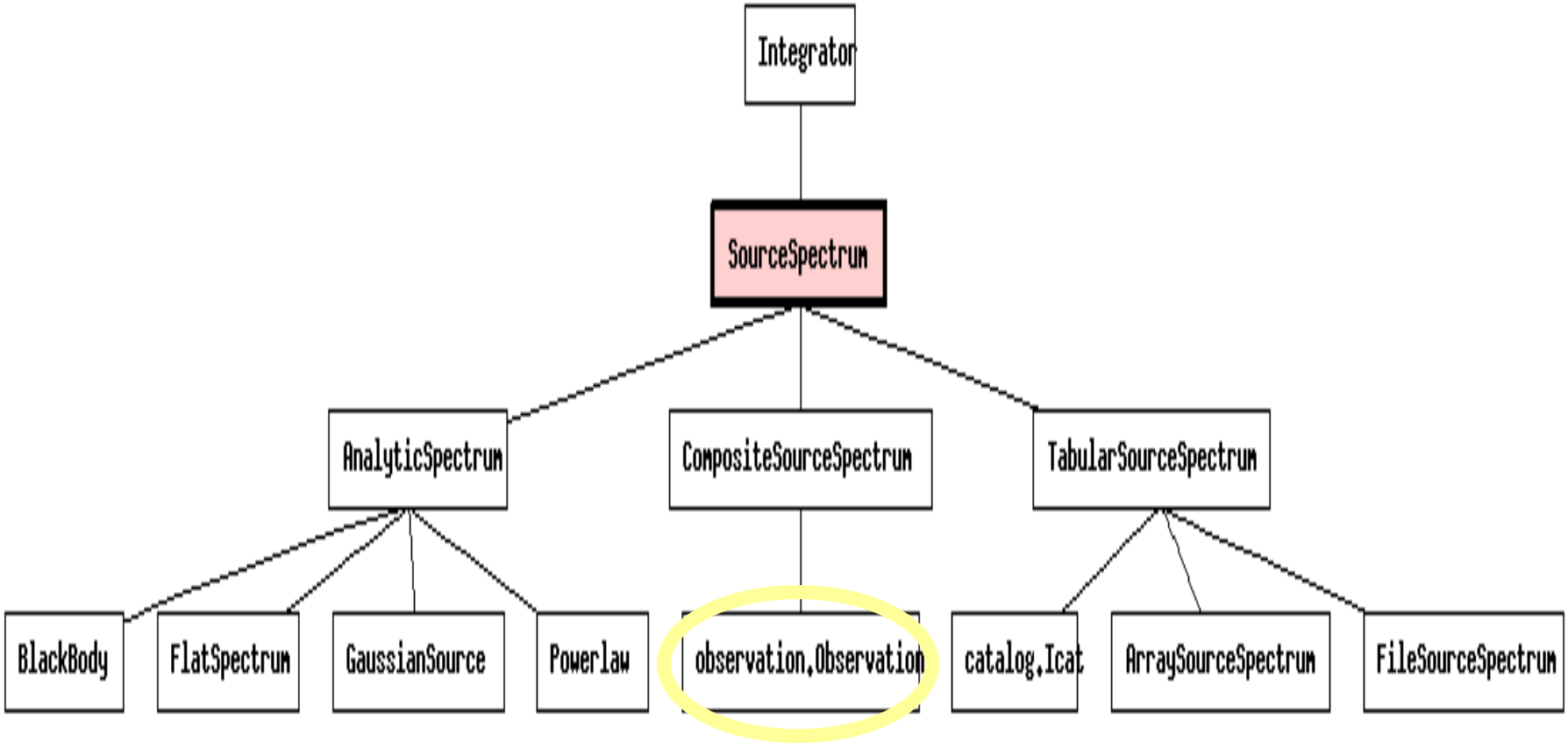
# SpectralElement Class Hierarchy



# Pysynphot Objects (cont.)

- **Spectrum** (model star or galaxy):
  - Have all characteristics of SpectralElements
  - Except:
    - Can be added to other sources (to produce a new source: CompositeSpectrum)
    - Can be multiplied with optical objects (to produce a new source)
    - Can't be multiplied with other sources
  - Plus:
    - Knows how to integrate itself (for total flux)
    - Know how to represent its data in different units for:
      - Flux
      - Wavelength

# Spectrum Class Hierarchy



# Pysynphot Objects (cont.)

- **Observation** (simulates an observed spectrum)
  - A Spectrum can collaborate with a Bandpass to produce an **Observation**
  - cannot be further manipulated
  - cannot be combined with anything else
  - can be queried for interesting things it knows about itself
  - knows how to bin its flux into a specified set of bins (integrate over each bin)
    - eg, corresponding to the actual light distribution onto detector pixels for a particular instrument configuration
    - Doing so accurately is important!
      - Astronomers do not want to lose any flux

# Use Python's OO Features

- Operator overloading to allow Python syntax to construct composite objects through Python expressions.
  - No need for special expression language!
- User interface is Python
  - Functionality of interest to the user is exposed via methods and properties
- Expressions only construct relationships, do no evaluation of fluxes or bandpasses.
  - Results are only computed when asked for through a method.
- Use of ***properties*** provides consistent .wave, .flux UI regardless of how these quantities need to be calculated

# …provides smart wavelength handling

- Composite objects know their components
- Each component (SourceSpectrum or SpectralElement) knows its own wavelength sampling
  - Integrations and samplings use ***union*** of all relevant wavelengths of combined object
    - Thus the context of a component (whether alone, or as part of a composite) changes the sampling over which it will be integrated
  - Narrow lines on wide spectrum no longer a problem

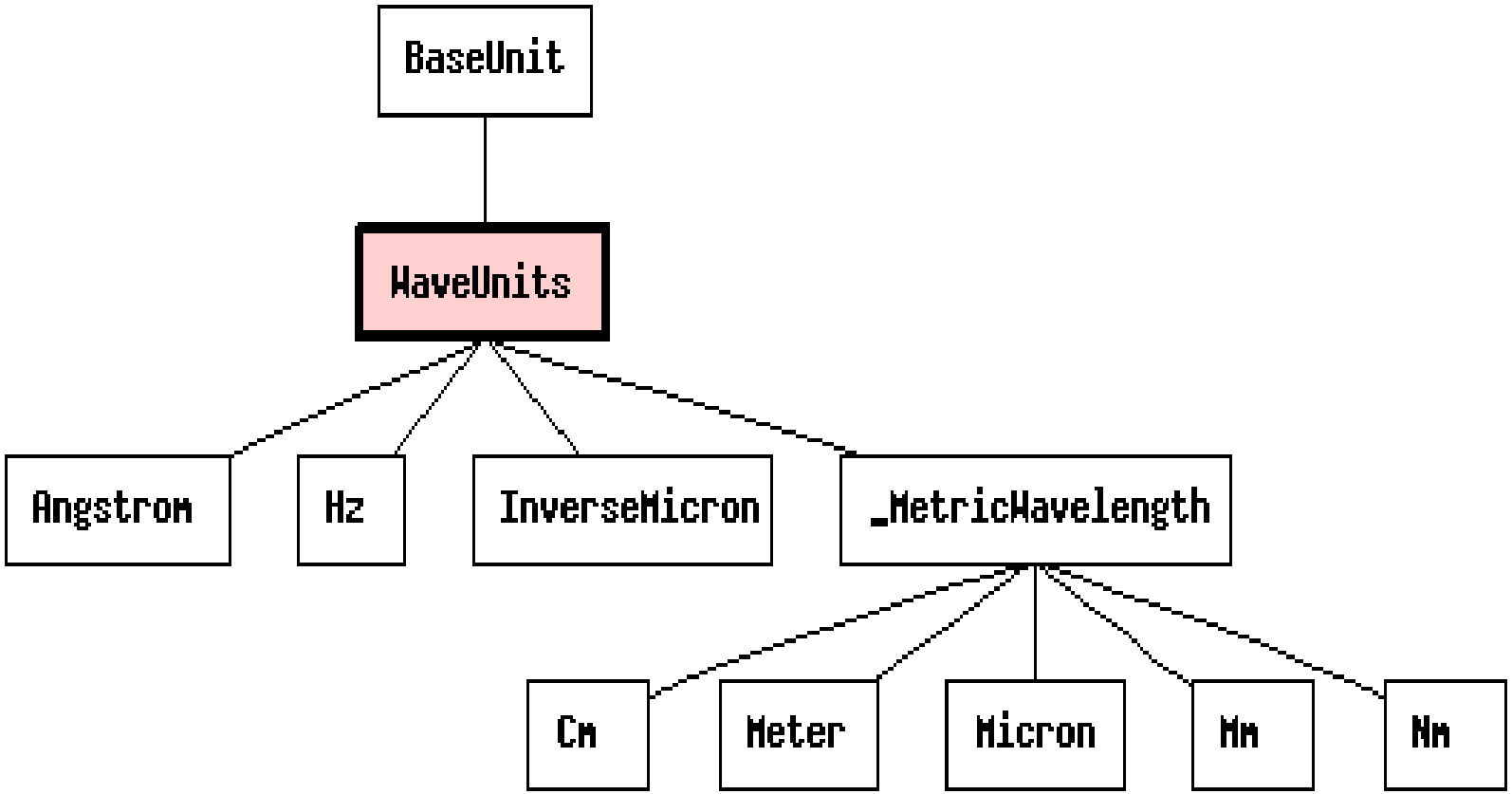
# Spectrum = “flux per wavelength”, but…

- Wavelength:
  - Angstroms (Å), microns (μ ), nm, mm, cm, m, eV, keV, meV, Hz, kHz, mHz, gHz
- Flux:
  - $F_\lambda$ (ergs/s/cm$^2$/Å), $F_\nu$ (ergs/s/cm$^2$/Hz), Janskys (Jy), milliJansky (mJy)
  - photons/s/cm$^2$/Å, photons/s/cm$^2$/Hz
  - Counts
  - Magnitudes (logarithmic; many varieties)

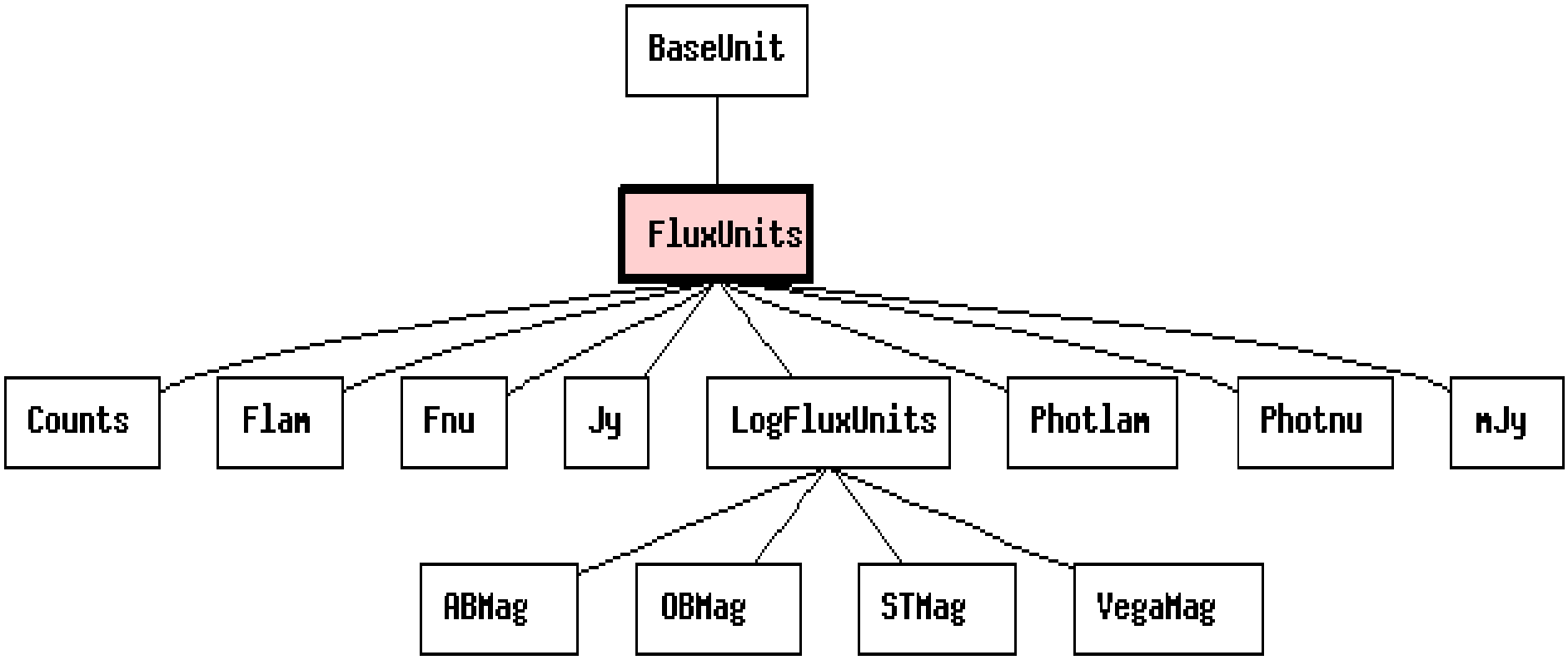
# Handling units

- Transforming flux density between wavelength and frequency is not a simple scaling factor ($d\nu = -d\lambda / \lambda^2$)
- **WaveUnit** and **FluxUnit** objects contain all necessary conversion code
- Internal representation always uses Angstroms (Å) for wavelength and photons/s/cm$^2$/Å for Flux.
  - Selection of units only changes how results are displayed, not how they are computed internally.

# WaveUnit Class Hierarchy



# FluxUnit Class Hierarchy



# Benefits from Other Libraries

- Array manipulation provided by numpy
  - No C extensions needed so far (computational load done by numpy)
- FITS table I/O provided by PyFITS
- Graphics provided by matplotlib/pylab
  - synphot package included several plotting tasks
  - we haven't written a single line of plotting code yet
  - Test users all say "Wow!"

# Conclusions

- The need to port a legacy application became an opportunity to improve it
  - thus “re-implementation” with new architecture, not simply port
- Python’s OO features and available packages made the job much easier
- Python’s ability to support functional-style as well as OO is important in lowering the adoption barrier by astronomers

# Status

- Already being used for one instrument's Exposure Time Calculator (SYNPHOT was too inaccurate)
- All Hubble ETCs will use it next year
- Commissioning process in progress
  - compare pysynphot results to SYNPHOT, to build confidence/acceptance among SYNPHOT users
- Preliminary public release Nov 08
- UI Documentation: Spring 09 release
- Scheduled v1.0 release: Aug 09