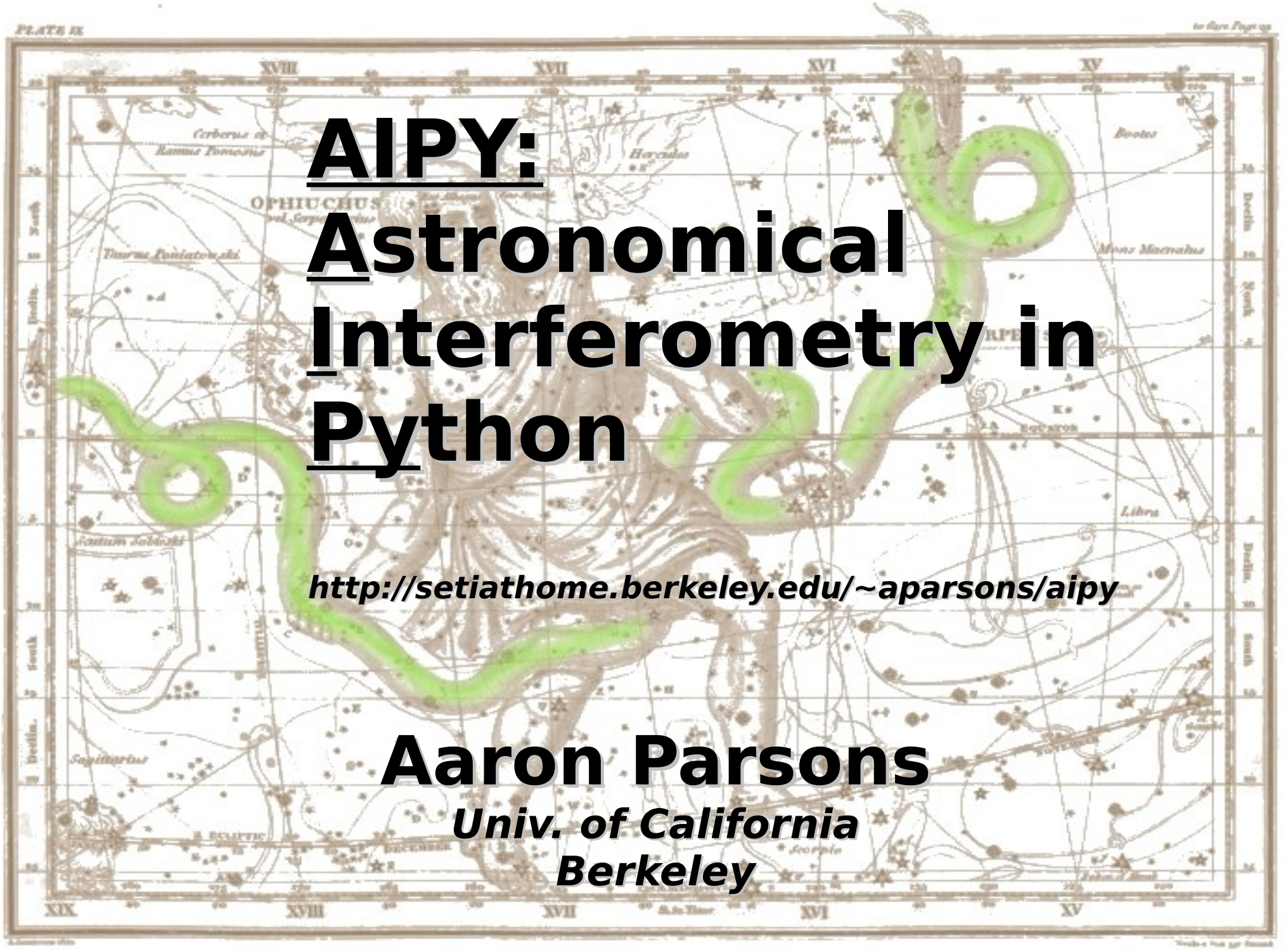


AIPY: Astronomical Interferometry in Python

http://setiathome.berkeley.edu/~aparsons/aipy

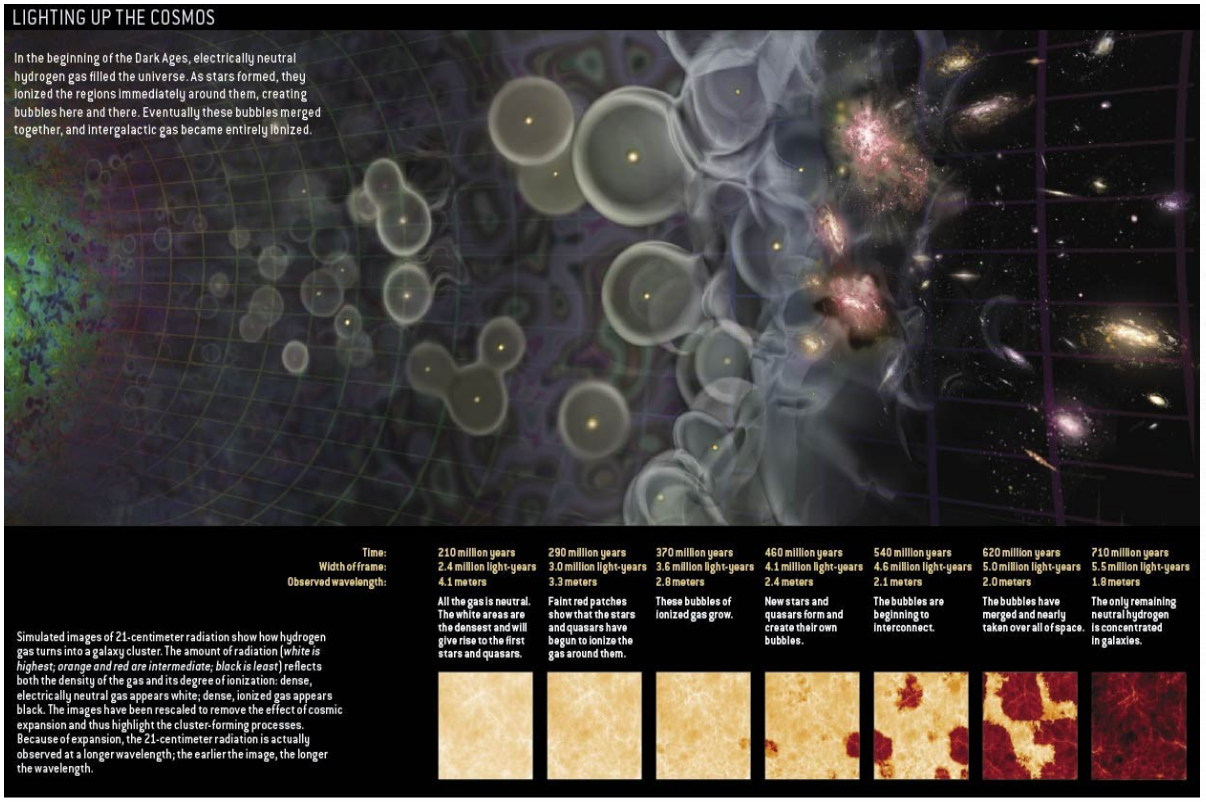
Aaron Parsons

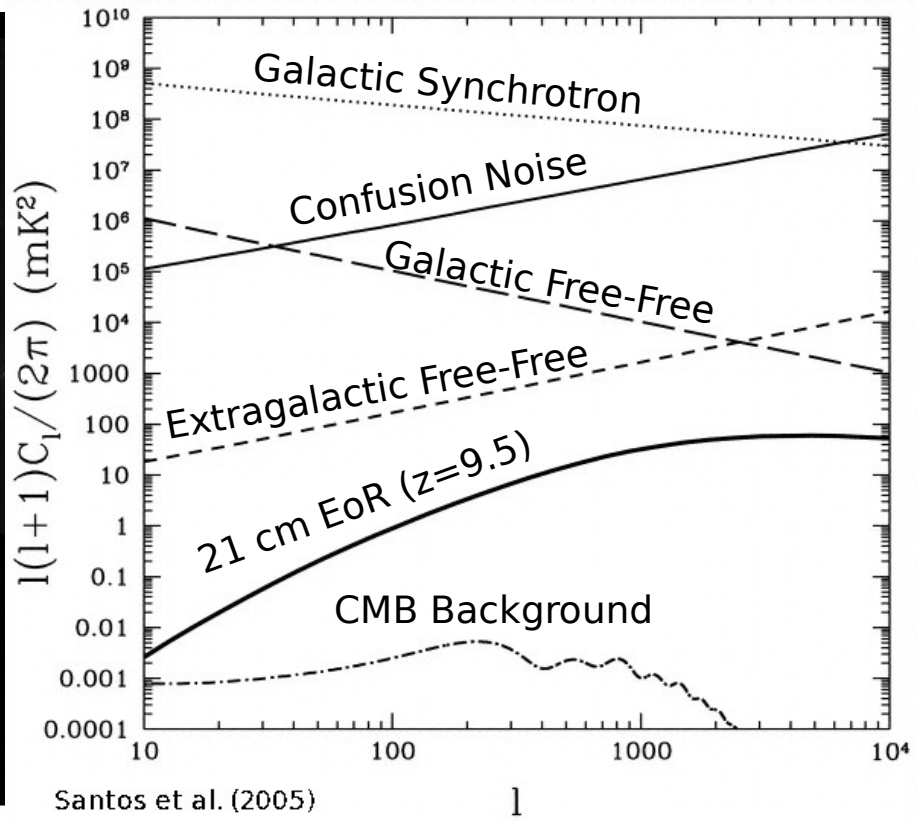
Univ. of California Berkeley

The Precision Array for Probing the Epoch of Reionization (PAPER)

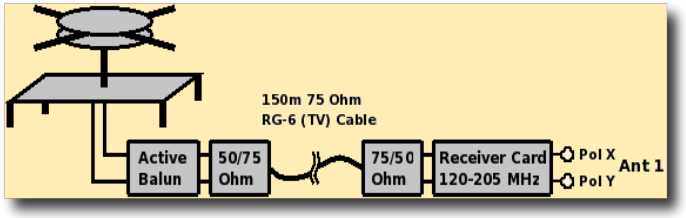
A. Parsons[1], D. Backer[1], R. Bradley[2,5], C. Parashare[2], N. Gugliucci[2], E. Mastrantonio[5], C. Carilli[6], A. Datta[6], J. Aguirre[3,4], M. Lynch[7], D. Herne[7], T. Colegate[7]

[1] U. of California, Berkeley, [2] U. of Virginia, [3] U. of Colorado, Boulder, [4] U. of Pennsylvania, [5] NRAO, Charlottesville, [6] NRAO, Socorro, [7] Curtin U., Perth, Australia



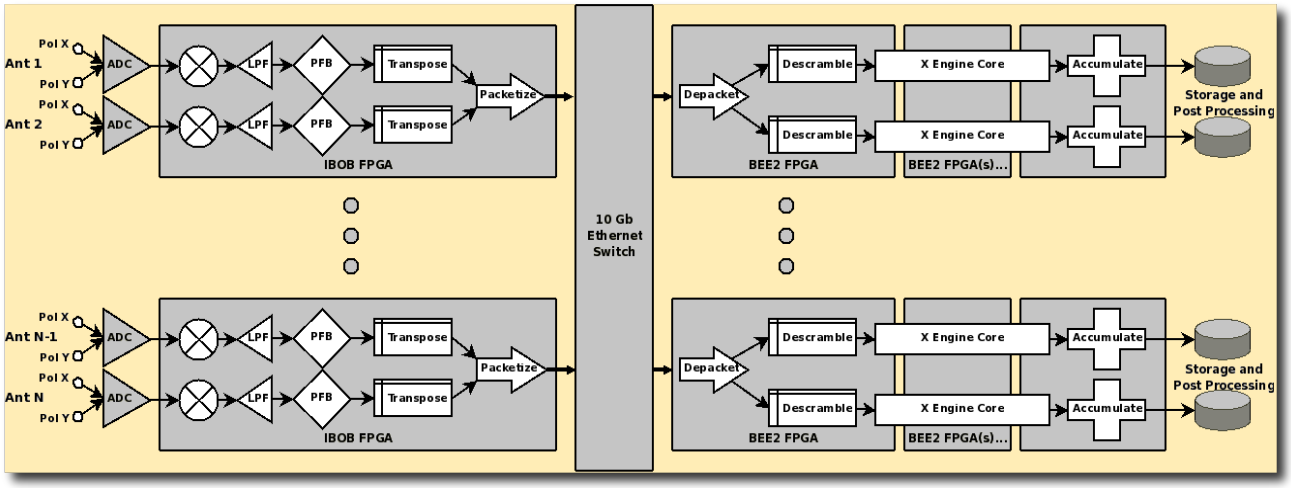


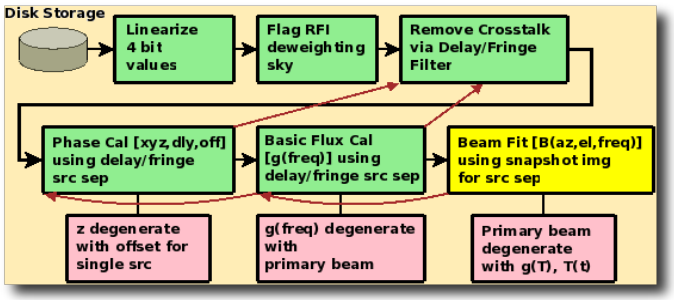
The PAPER Architecture



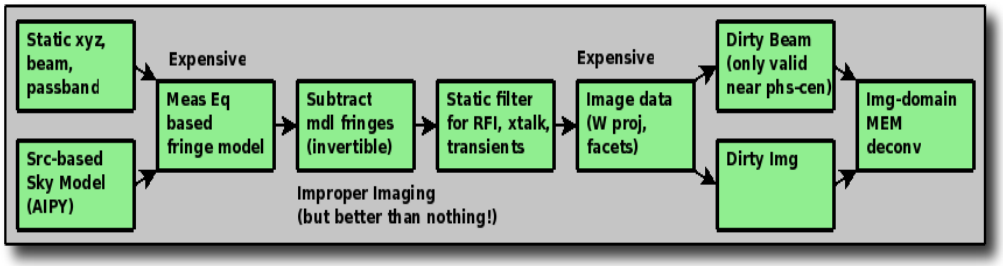
- Non-tracking Crossed Dipoles
- Wide Bandwidth (125-205 MHz)
- Movable (unburied TV cable)
- Smooth Beam

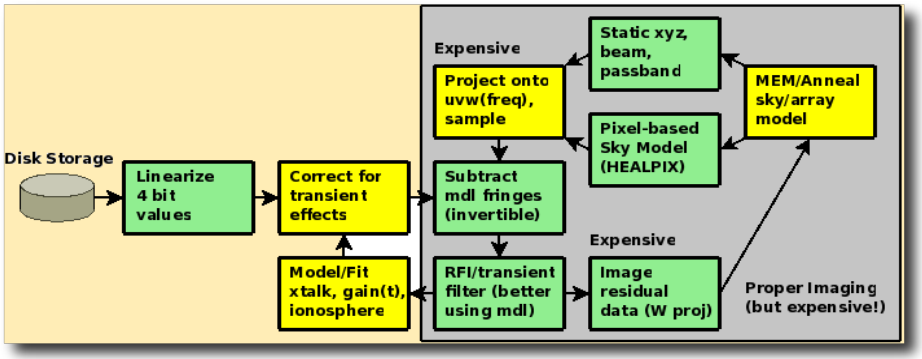
- Flexible FPGA-based
- Packetized Correlator
- Full-Stokes
- Large # Ants (scalable)
- Wide Band (up to 200 MHz)
- 2048 Channel Polyphase Filter Banks
- 4-bit Cross-Multipliers





- Snapshot Model-based Imaging/Calibration
- Bootstrap Imaging/Calibration
- W Projection, Multi-frequency Synthesis
- Python-Based, Ties in MIRIAD, FITS, HEALPIX
- Measurement Equation-based Parameter Fit

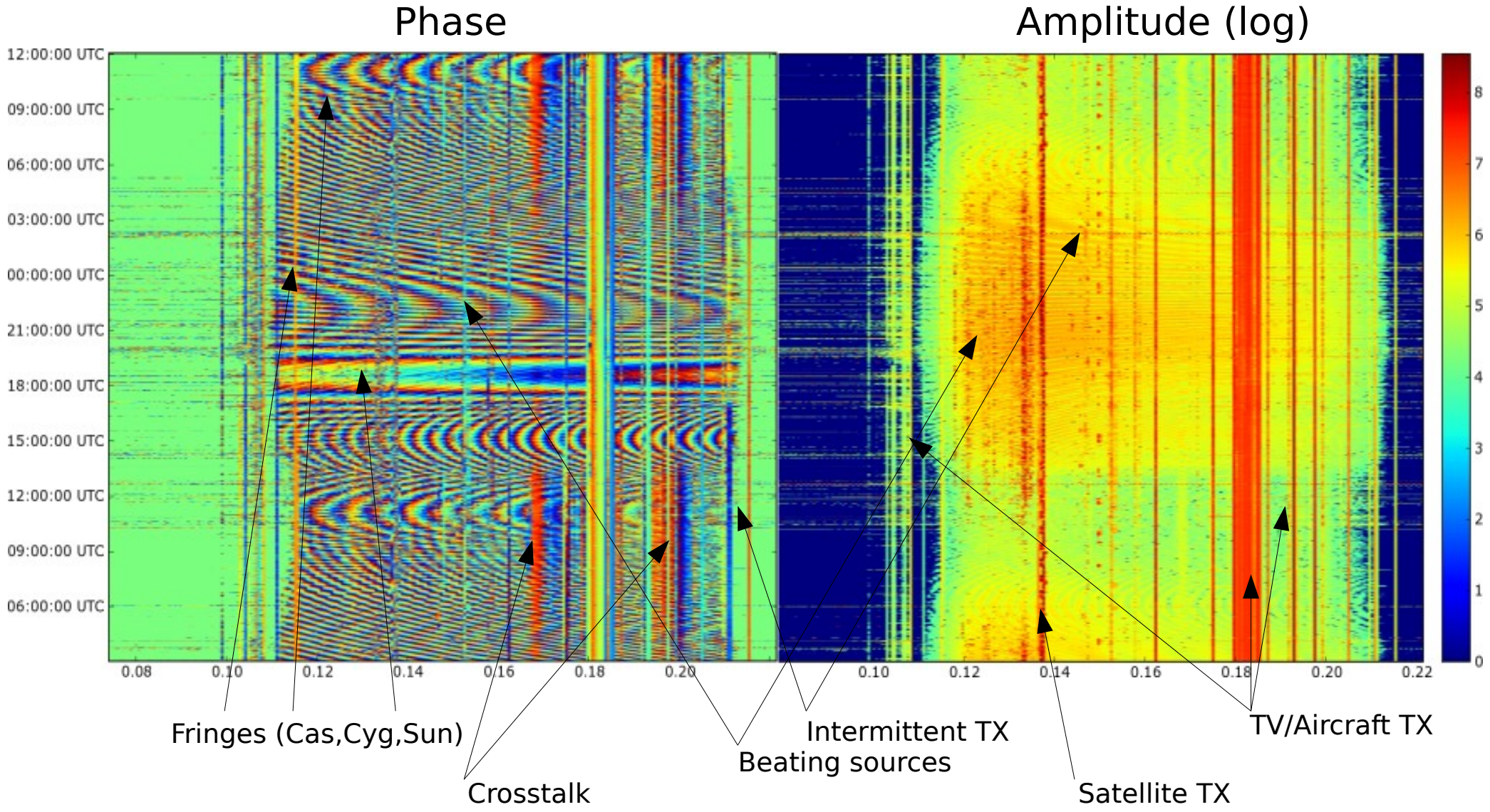




Sample Data:

1 Day, 1 Baseline

PAPER Green Bank



Another Imaging Package... Why?

New problems to solve:

- Wide fields of view
- Large relative bandwidths
- Huge numbers of antennas
- Non-tracking primary beams
- Real-time processing
- Source separation
- Ionospheric distortion

New tools available:

- W projection
- Delay imaging
- Parallel processing/clusters
- Open-source software
- More sophisticated programming models (object-oriented, run-time compiled)

Python:

- Interpreted (up to 5x more productive than compiled languages, according to Burton Group study)
- Object-Oriented
- Readable
- Fast, high-level data types
- Large community of programmers
- Fast-growing community of numerical/scientific/astronomy programmers
  - PyFITS
  - PyRAF
  - SciPy
  - NumPy
  - PyEphem
  - Pylab
  - OBIT
  - CASA
  - MeqTrees

Performance:

- Generally, only a small fraction of code needs to run fast
- Python's profiler can tell you where the bottlenecks are
- Bottlenecks can be recoded in C/C++/Fortran and wrapped into Python
- NumPy, the foundation of numerical/vectorized processing in Python, is coded in C and runs on average only 1.5 times slower than pure C
- You should only be allowed to worry about speed while your code is actually running

What AIPY Is (and Isn't)

AIPY is:

- A module adding tools for interferometry to Python (not visa versa)
- An amalgam of pure-Python and wrappers around C++ and Fortran
- Object-Oriented
- A collection of low- to mid-level operations (you write the program)
- A toolkit (i.e. a grocery store, not a restaurant)
- In beta release (frequent updates/bug-fixes/changes)

AIPY isn't:

- A solution (it just helps you find it)
- A one-stop shop (it makes use of other open-source projects)
- A replacement for other interferometry packages
- Wedded to a file format (but only MIRIAD-UV, FITS currently supported)

Philosophical Pedantry:

- Follow the (abbreviated) Zen of Python:
  - Explicit is better than implicit
  - Simple is better than complex
  - Complex is better than complicated
  - Special cases aren't special enough to break the rules
  - Practicality beats purity
  - There should be one (and preferably only one) obvious way to do it, although that way may not be obvious at first unless you're Dutch
  - Now is better than never
  - In the face of ambiguity, refuse the temptation to guess
- Don't handicap the programmer; allow them all the rope they want
- Don't hide data; return it (or at least grant access) at every turn

The Toolbox

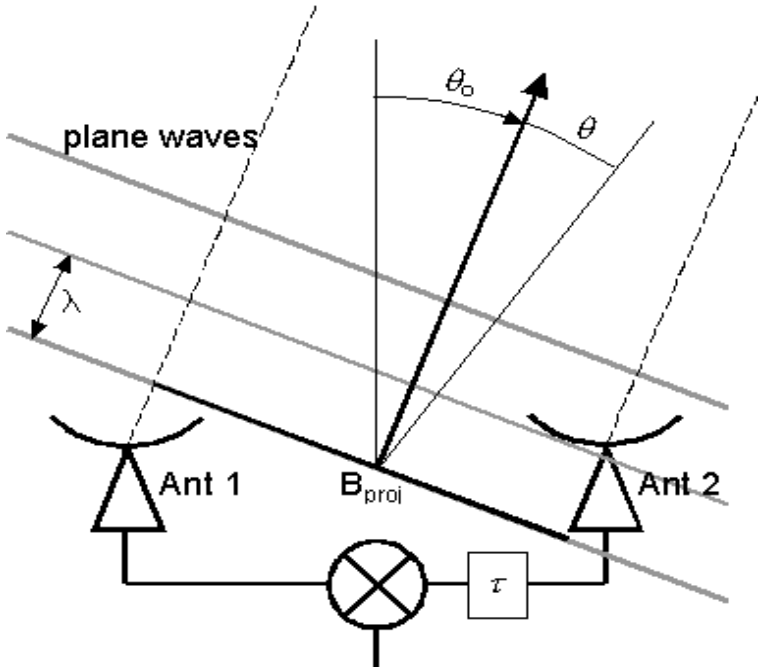
MODULES:	
ant/sim/fit	geometry, phase / gain, amplitude / parameter fitting
const	physical constants
coord	coordinate transforms (eq, ec, ga, precession, etc.)
deconv	deconvolution (clean, lsq, mem, anneal)
healpix	wrapper around healpix C++ (Max-Planck-Institut)
img	imaging (gridding/weighting, W projection, beam calc)
loc	machinery for importing user-defined AntennaArrays
map	generation of all-sky maps, faceting
miriad	interface to MIRIAD files
optimize	parameter fitting, optimization code
src	lists of known sources and parameters
----3rd party----	----------
ephem	astrometry (relied upon heavily)
pyfits	reading/writing FITS files
numpy	numerical python (fundamental to everything)
pylab/matplotlib	plotting, map projection

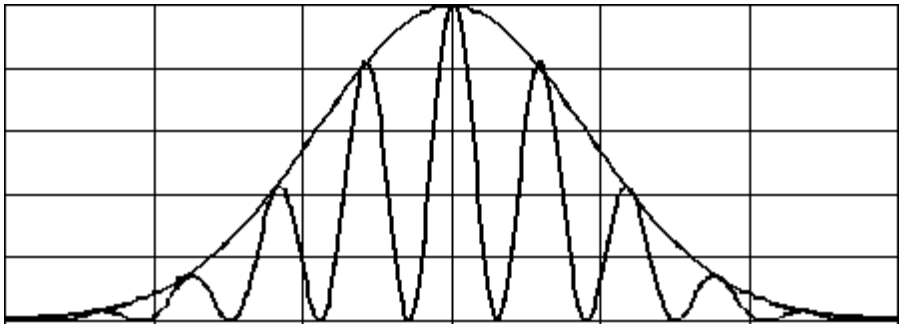
and example scripts for:
manipulating/visualizing MIRIAD files, flagging RFI, filtering out crosstalk, fitting parameters, simulating arrays, making/viewing all-sky maps, phasing to/filtering out sources, imaging, etc.

Interferometry Fundamentals

Our Parameterized Measurement Equation:

$$V_{ij}(\nu,t) = \sum_{s=srcs} g_i(\nu) g_j^*(\nu) I_{s,\nu_0} \left(\frac{\nu}{\nu_0}\right)^{\alpha_s} e^{2\pi i(\vec{b}_{ij}(\nu,t)\cdot\hat{s}_s + \nu\tau_{ij} + \phi)}$$





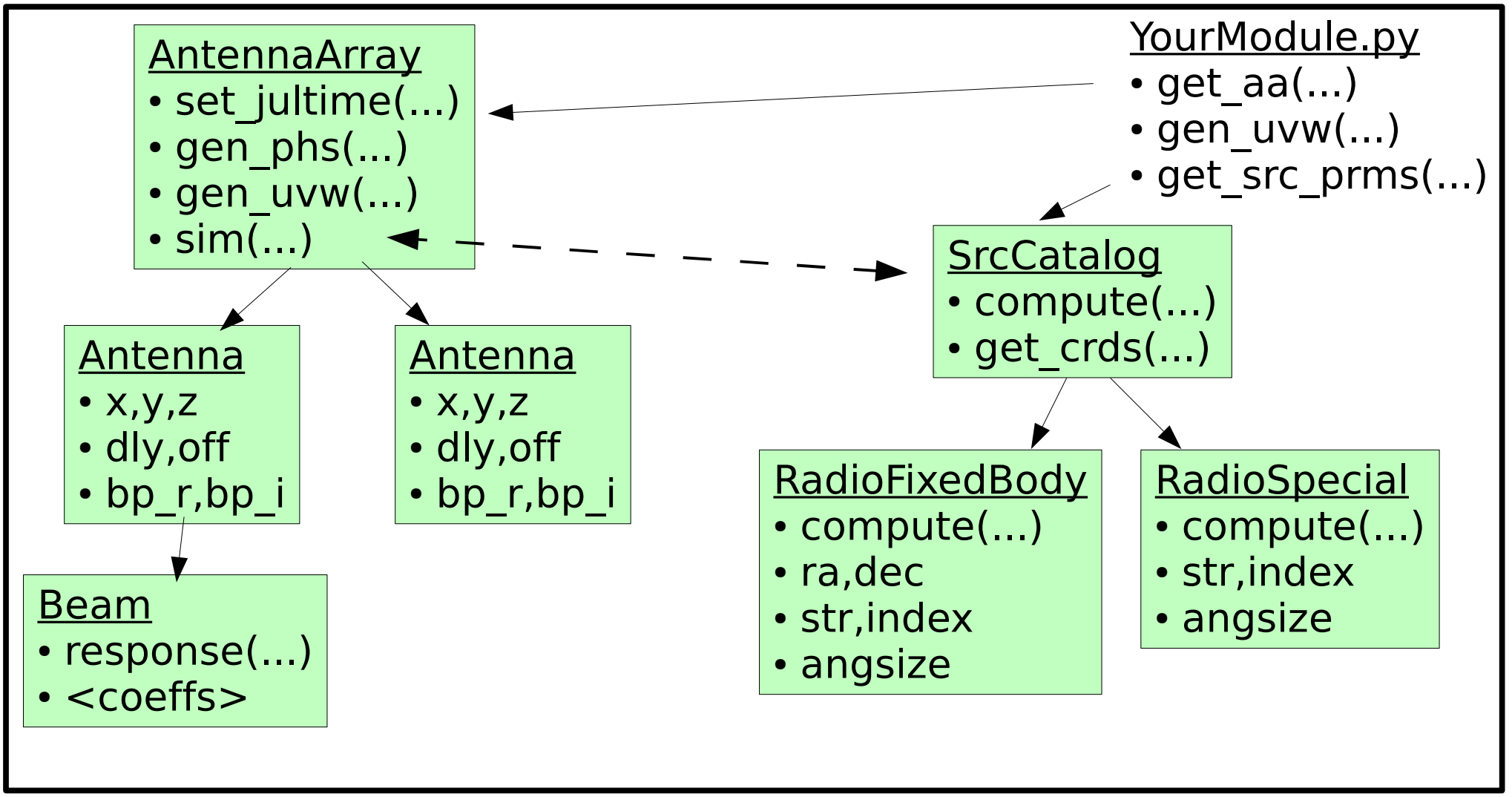
Flat Field (Small Angle) Approximation:

$$\vec{b}_{ij}\cdot\hat{s}_s = u\ell + vm + w(\sqrt{1-\ell^2-m^2}-1) \approx u\ell + vm$$

- (u,v) coordinates represent the Fourier transform of the angular sky coordinates (l,m)
- Angular power spectrum directly measured in UV (visibility) domain
- Inverse 2D FFT of gridded UV visibilities yields dirty image (convolved with synthesized PSF)
- Synthesized PSF is inverse 2D FFT of visibility weights

Building a Simulator

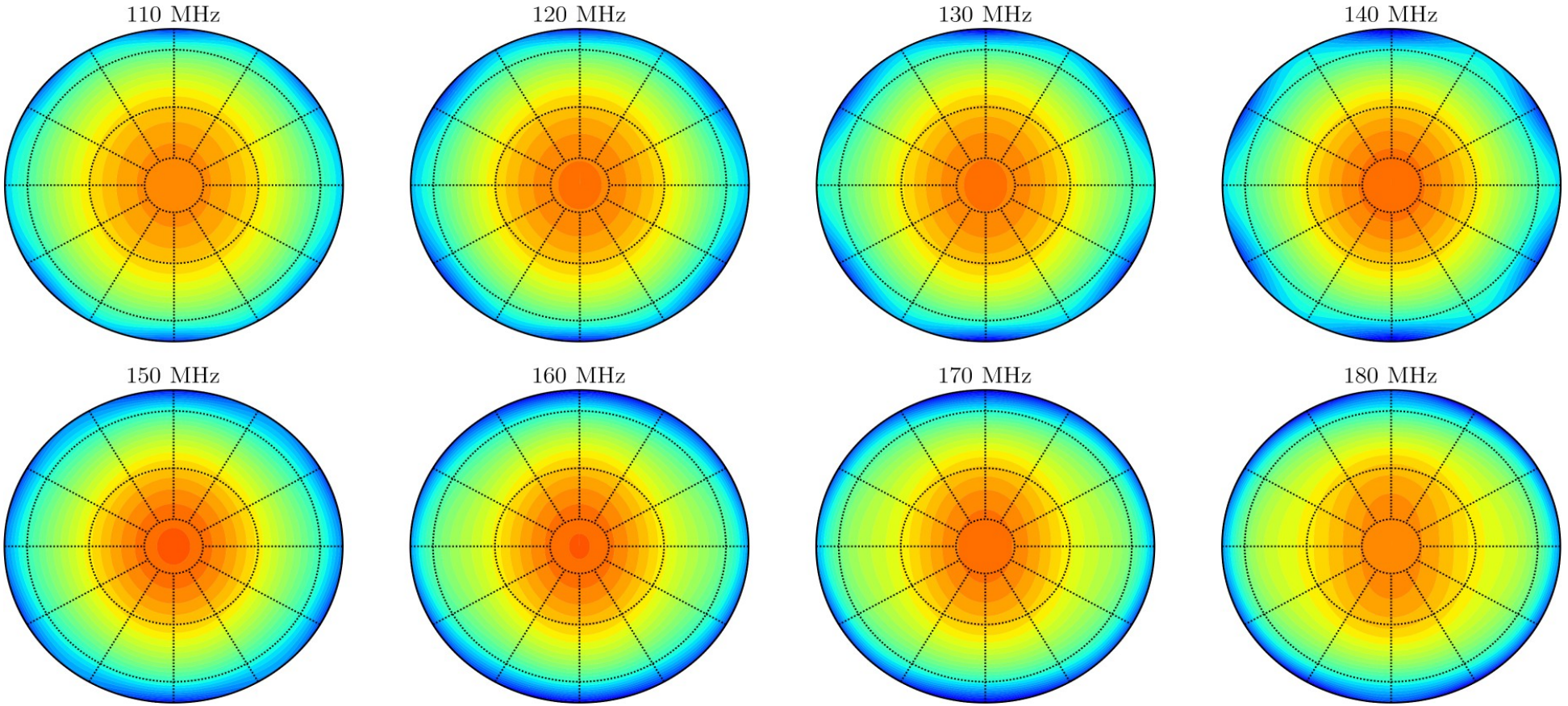
$$V_{ij}(\nu,t) = \sum_{s=srcs} g_i(\nu) g_j^*(\nu) I_{s,\nu_0} \left(\frac{\nu}{\nu_0}\right)^{\alpha_s} e^{2\pi i(\vec{b}_{ij}(\nu,t)\cdot\hat{s}_s + \nu\tau_{ij} + \phi)}$$



Modeling Beam of Dipole + Flaps

$$G_\nu(\theta,\phi) \approx \sum_{n=0}^{7} \nu^n \left(\sum_{\ell=0}^{8} A_{\ell m,n} Y^{\ell m}(\theta,\phi) \right)$$

40dB zenith to horizon, 60 degree FWHM
Model accurate to 0.1% in main lobe
Smooth spatially and vs. frequency



Challenge 1: Starting from Scratch

"Sometimes you can't get started because you can't get started" – Don Backer

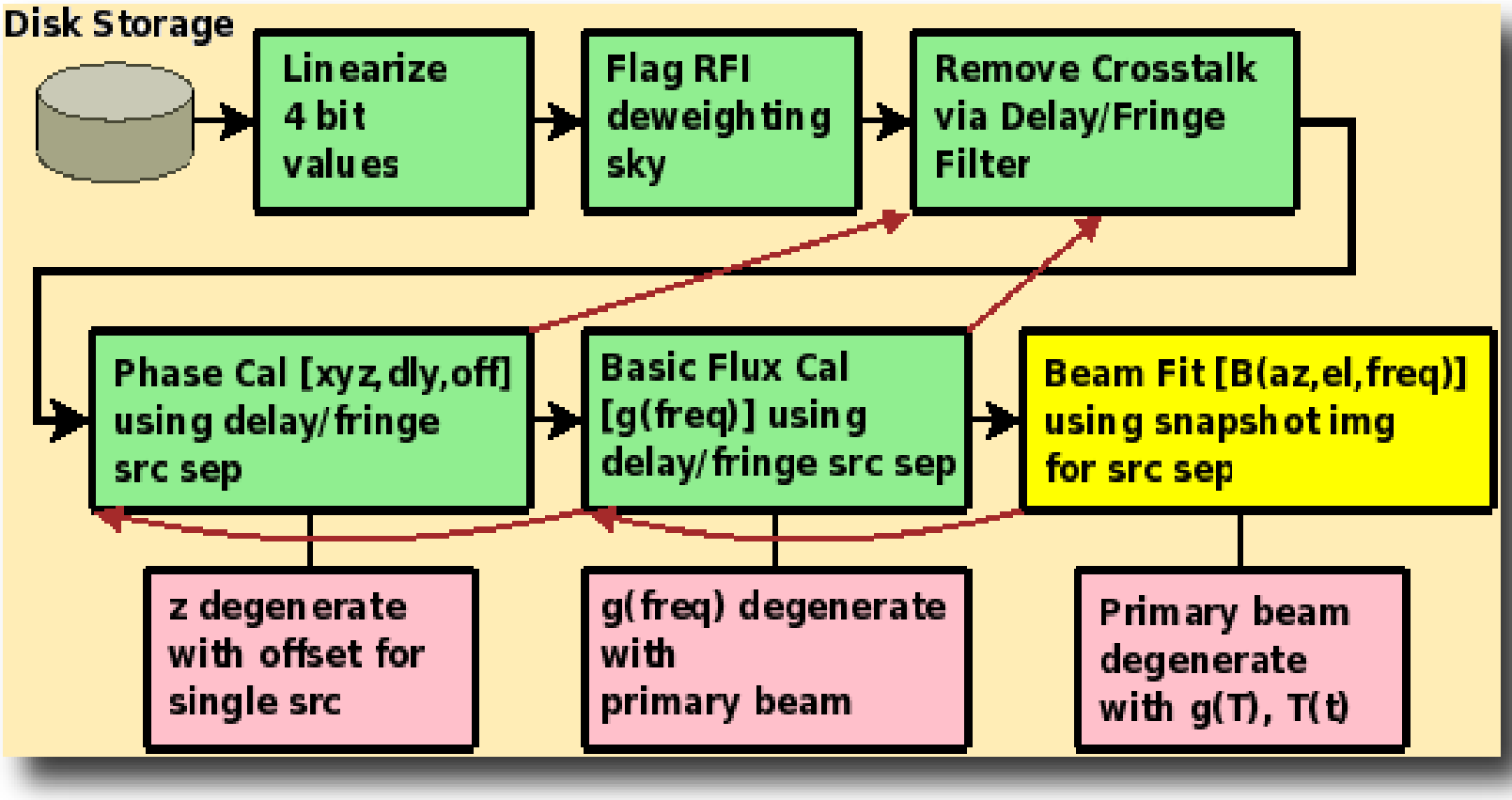
$$V_{ij}(\nu,t) = \sum_{s=srcs} g_i(\nu) g_j^*(\nu) I_{s,\nu_0} \left(\frac{\nu}{\nu_0}\right)^{\alpha_s} e^{2\pi i(\vec{b}_{ij}(\nu,t)\cdot\hat{s}_s + \nu\tau_{ij} + \phi)}$$

The Problem:

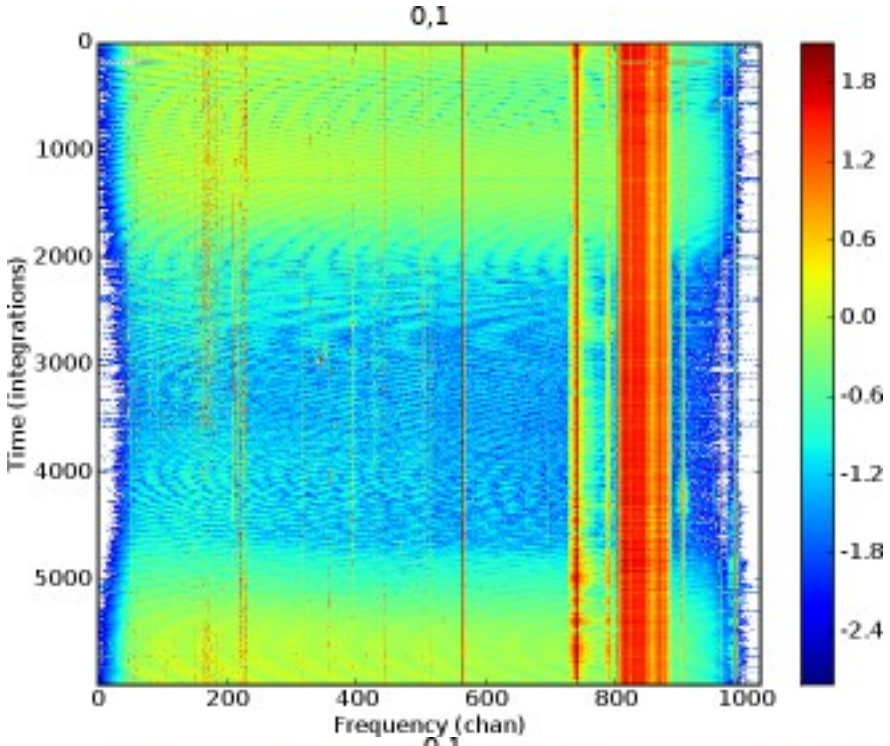
- Self-calibration needs single-source data
- Source isolation requires calibration
- Parameter fitting needs a good starting guess
- How do we get to first base?

Bootstrapping:

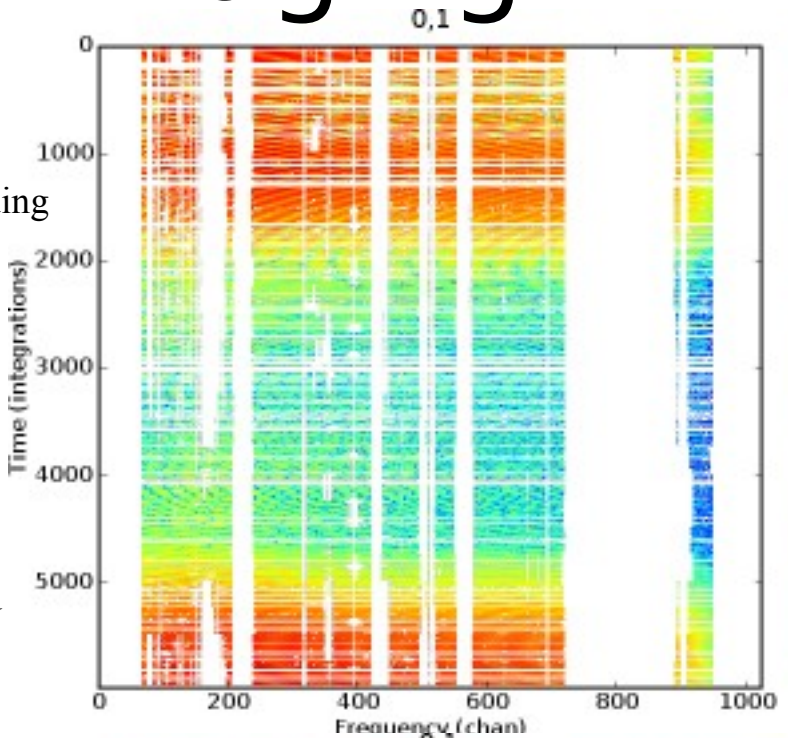
- Direct, imperfect, iterative
- Do not rely (excessively) on priors
- Take advantage of wide bandwidth
- Address degeneracies one at a time



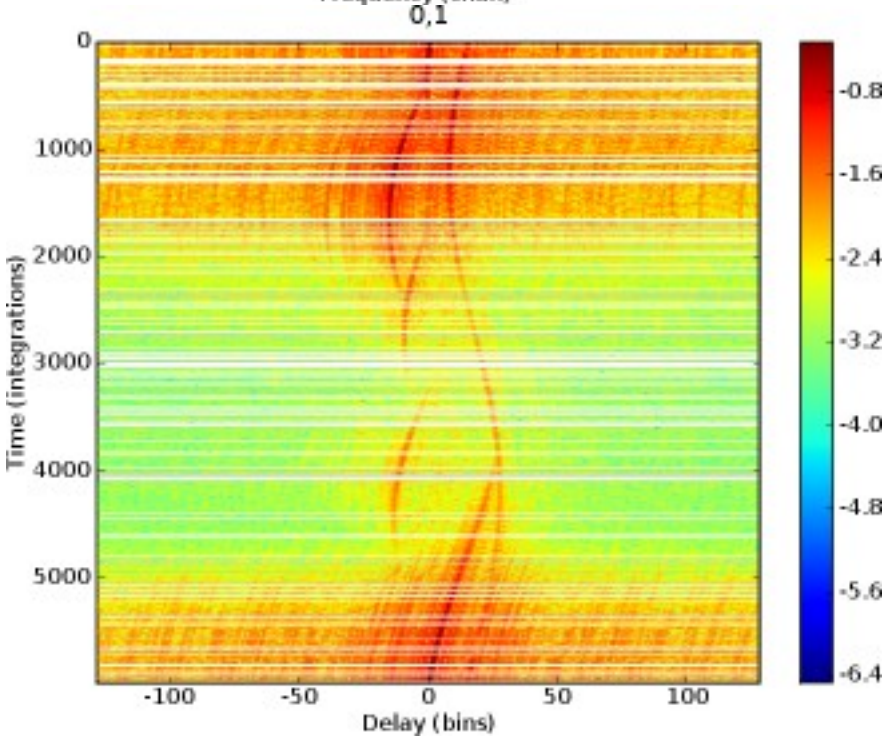
1-Dimensional Imaging



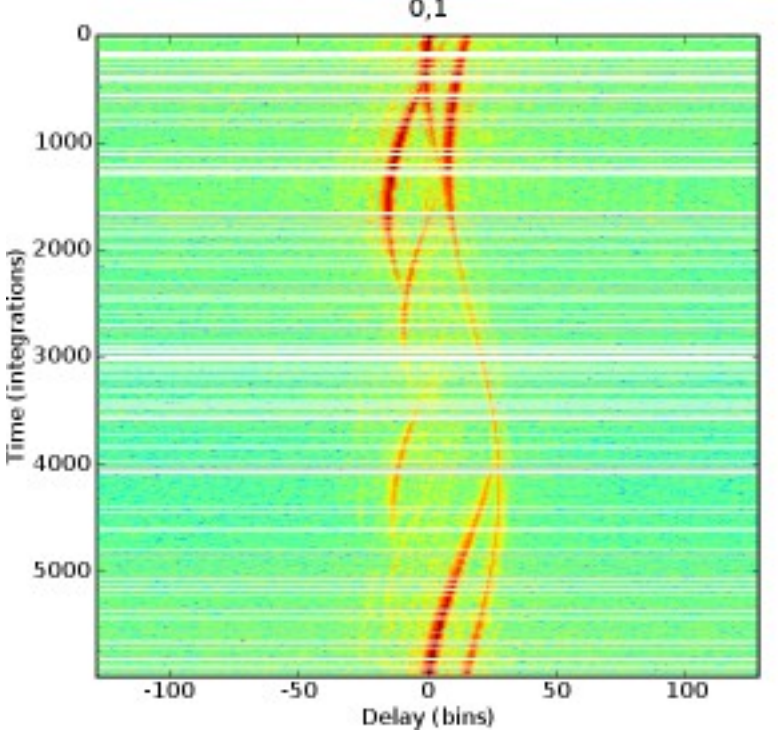
XRFI:
- Remove model sky
- Statistical thresholding
- Manual flagging



DELAY TF:
- iFFT of passband
- Convolved by "delay beam"



1D CLEAN:
- Compensate for holes in passband
- Retain passband shape for self-cal



Ex: 1-D Imaging in AIPY

```
import aipy, numpy, sys
freqs = numpy.arange(.075, .225, (.150 / 1024))
aa = aipy.loc.get_aa('loc_key', freqs)
src = aipy.src.get_src('cyg')

def isolate_src(mir_file, preamble, data, flags):
        uvw, t, (i,j) = preamble
        if i == j: return preamble, data, flags
        aa.set_jultime(t)
        src.compute(aa)
        try:
                flags = numpy.logical_not(flags).astype(numpy.float)
                gain = numpy.sqrt(numpy.average(flags**2))
                if gain == 0: return preamble, data, flags
                kernel = numpy.fft.ifft(flags)
                data = numpy.where(flags, 0, data)
                data = aa.phs2src(data, src, i, j)
                dly_img = numpy.fft.ifft(data)
                dly_img, info = aipy.deconv.clean1d(dly_img, kernel)
                dly_img += info['res'] / gain
        except(aipy.ant.PointingError): return preamble, data, flags
        dly_img[2:-2] = 0
        data = numpy.fft.fft(dly_img)
        return preamble, data, flags

infile = aipy.miriad.UV(sys.argv[-1])
outfile = aipy.miriad.UV(sys.argv[-1] + '.cyg', status='new')
outfile.init_from_uv(infile)
outfile.pipe(infile, mfunc=isolate_src, raw=True)
```

Bring in AIPY functionality

Use an AntennaArray that you've built in loc_key.py

Phase to Cygnus A (defined in src.py or loc_key.py)

Define a function to map data from an input file to output

Set the current time (inefficient)

Compute current source position (inefficient)

Calculate gain of “delay beam”

Calculate shape of “dirty delay beam”

Blank out RFI

Phase data to Cygnus A

Calculate “dirty delay image”

Deconvolve dirty image by dirty beam

Add back in residuals with appropriate gain

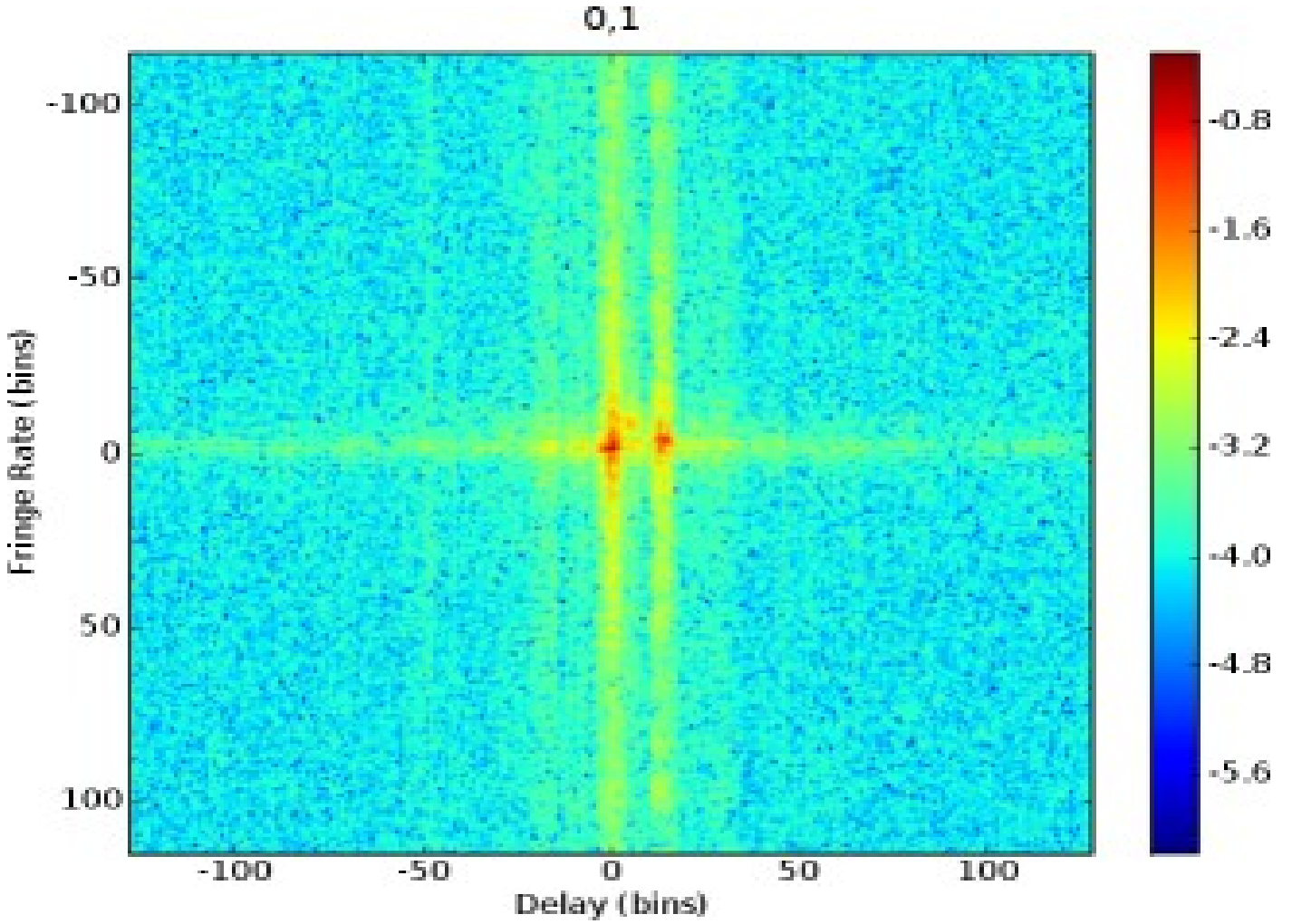
Blank out delays that aren't close to zero

Convert back to frequency domain

Pipe data into new file through the function we made

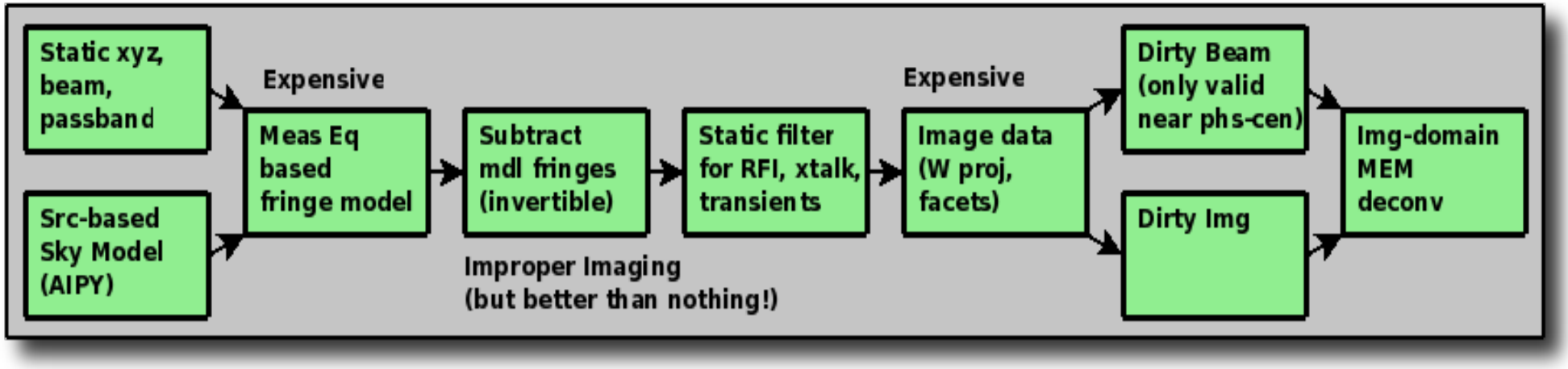
Delay/Fringe Rate Transform

- Two sources can show up at same delay for a given baseline
- Over a time interval (say, 1 hr), can be differentiated by fringe rates (see below)
- To prevent excessive blurring (changing fringe rate w/ time) best to phase to source before filtering
- Clean delay axis before iFFT of time axis
- Clean fringe rate axis to account for flagging of entire integrations



Challenge 2: Imaging Fidelity

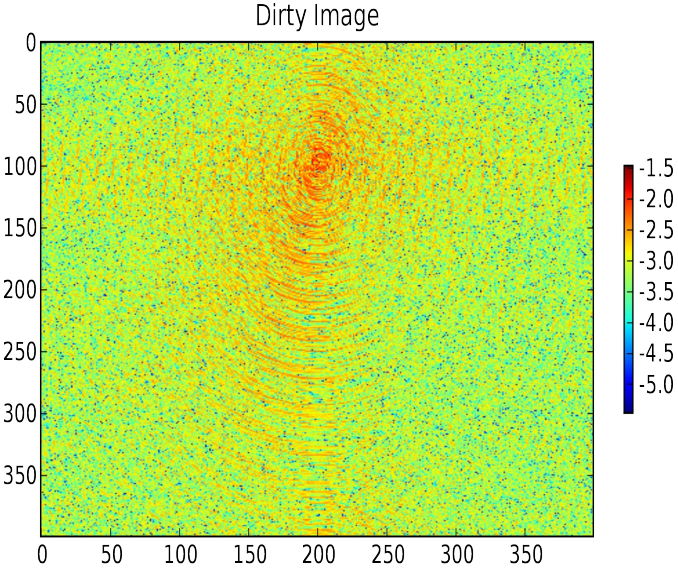
- Correct for wide-field effects (W projection)
- Correct for sampling effects (CLEAN, MEM)
- Correct for non-tracking primary beam
- Fundamentally incorrect, but can do increasingly better job with good sky & antenna models

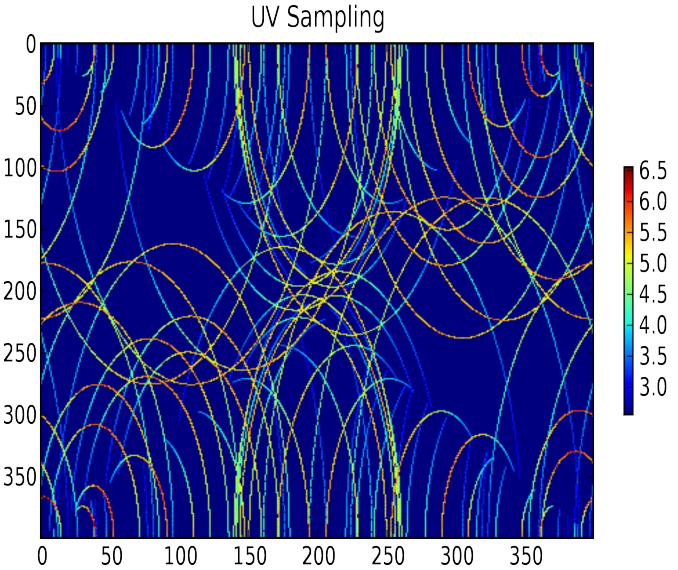


Wide Fields and W Projection

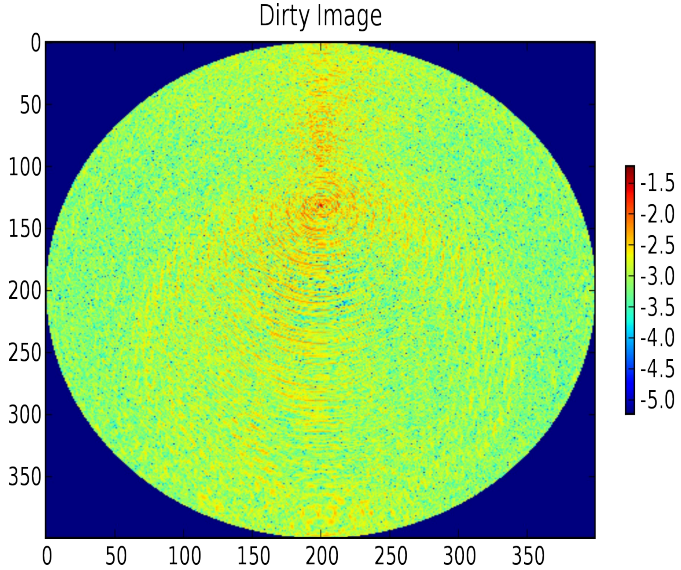
- Strong source far from phase center may have sidelobes in area of interest
- Need a point-source on which to pin a “dirty beam”: W projection (Cornwell et al.)
- Enhances fidelity of forward imaging at moderate computational cost

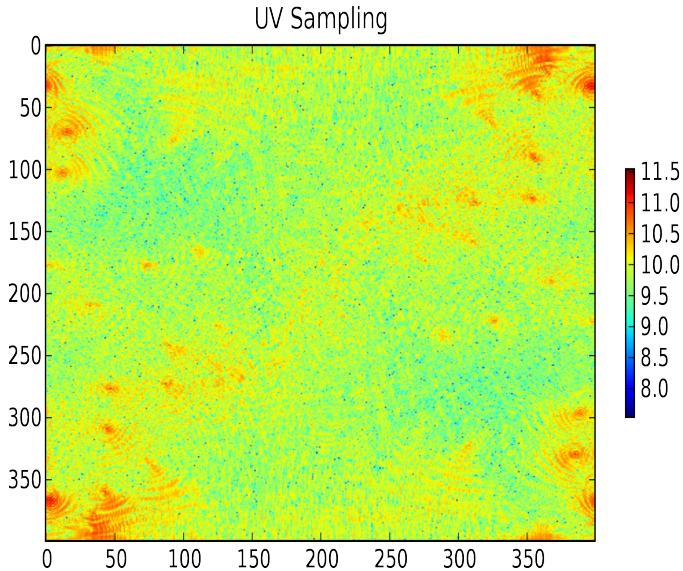
Standard Gridding

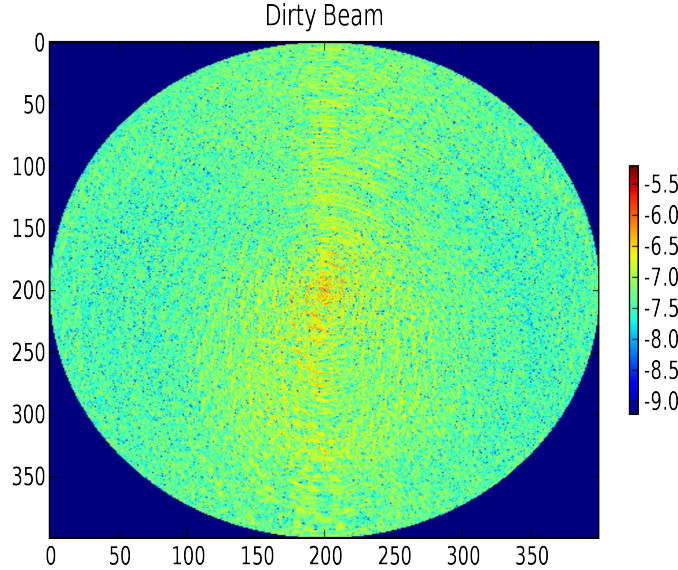


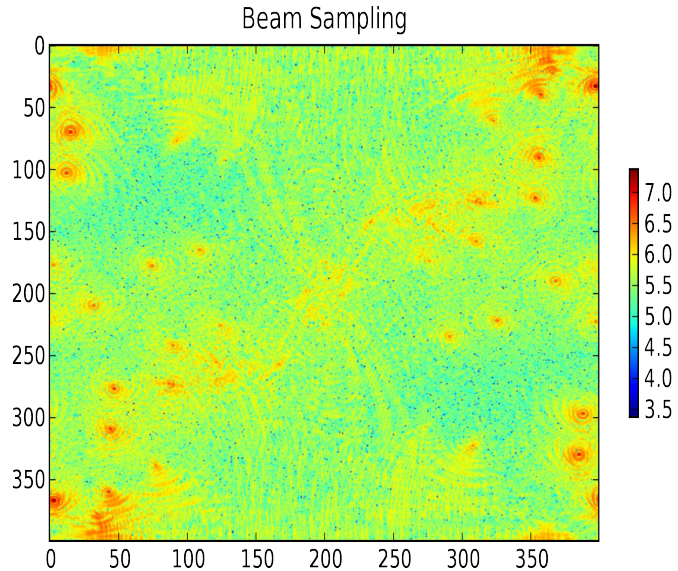


W Projected Gridding, Src 30° off center





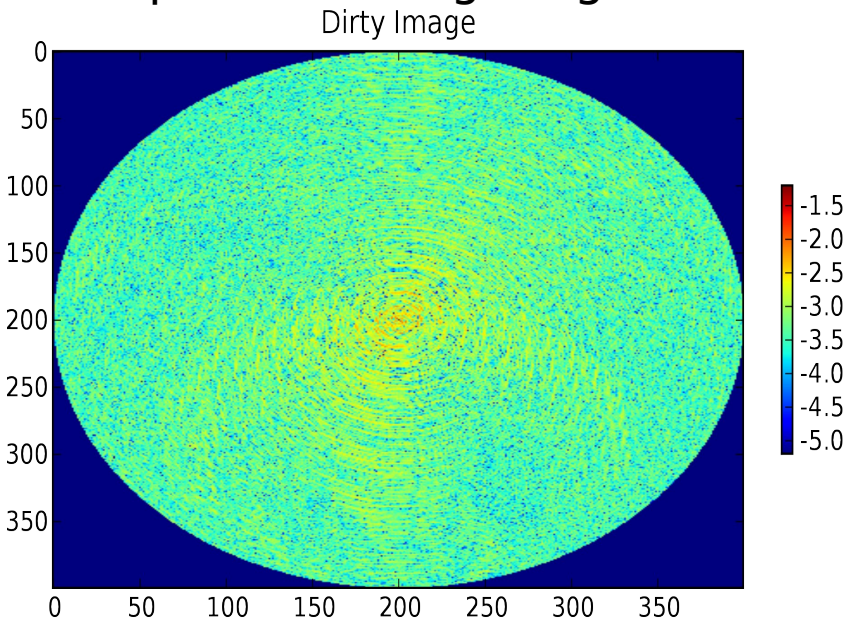


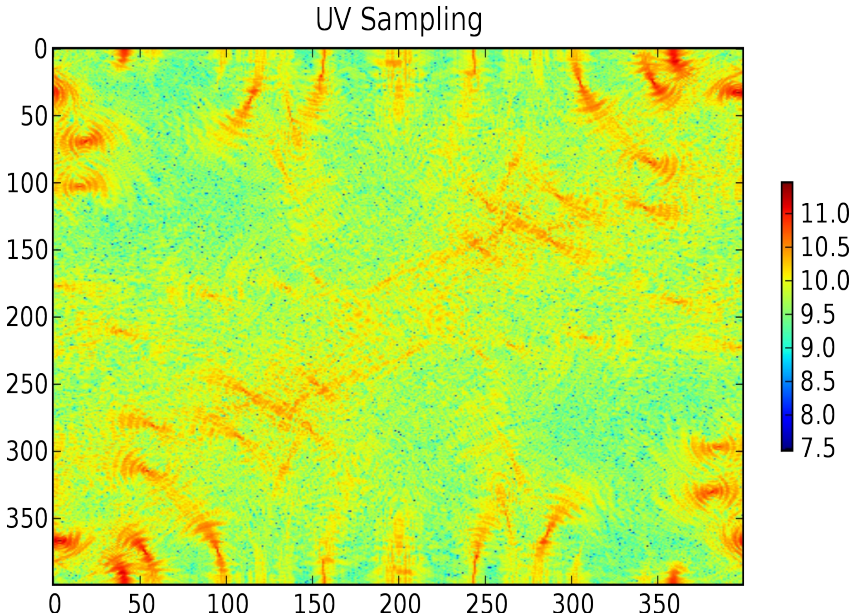


Imaging with Non-Tracking Beams

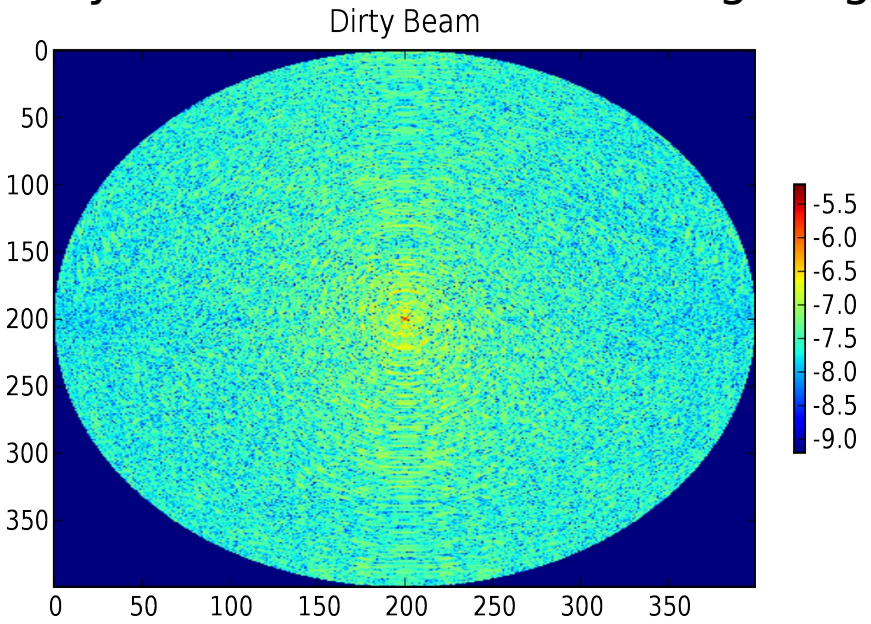
- Only possible to weight for one track through primary beam
- Dirty beam weighted for phase center imperfectly deconvolves away from center
- Can evade problem with snapshot imaging

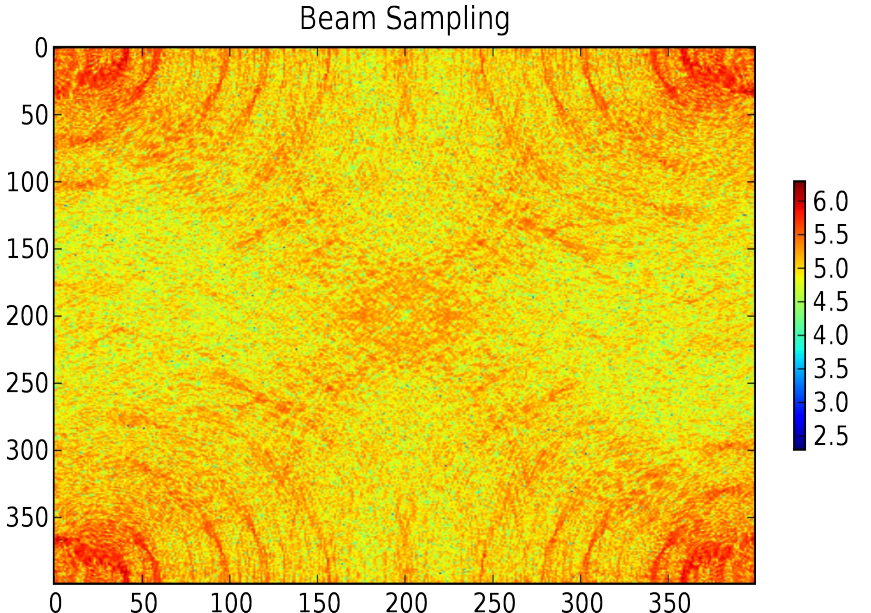
Position-Dependent Weighting in Data



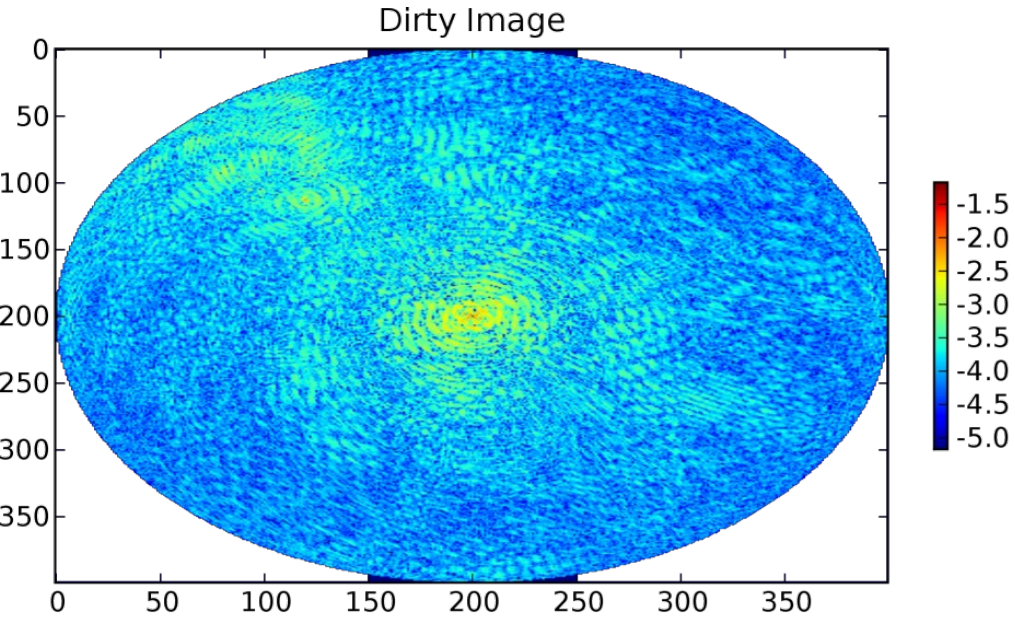


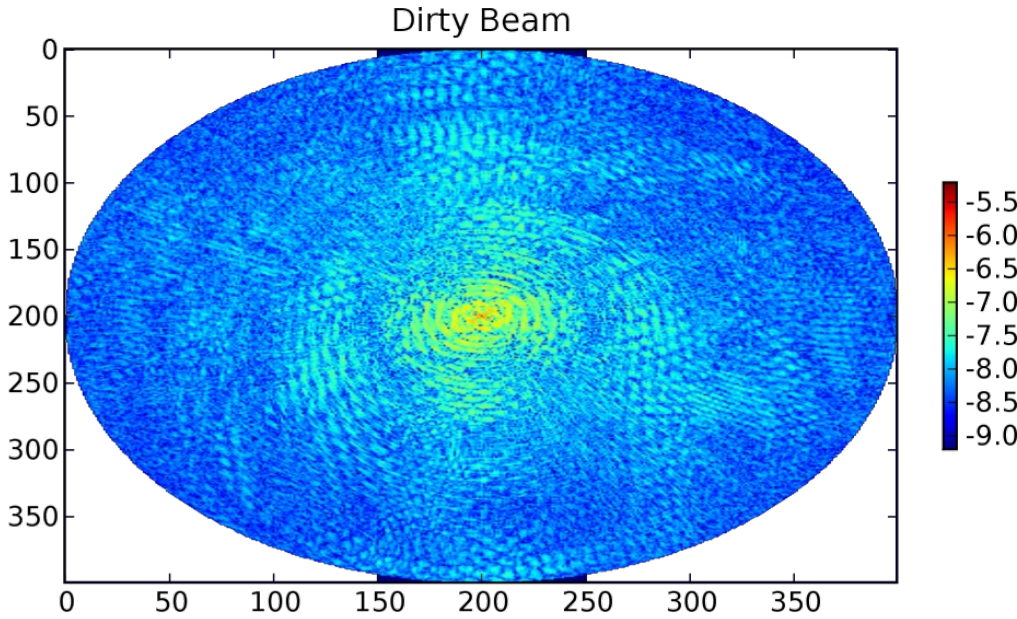
Dirty Beam with Incorrect Weighting

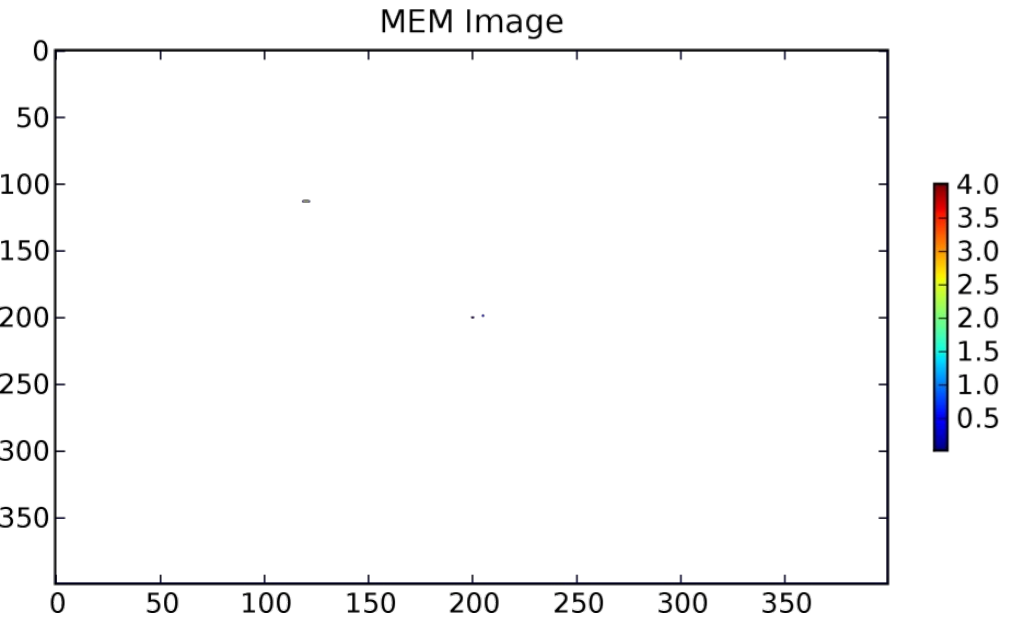


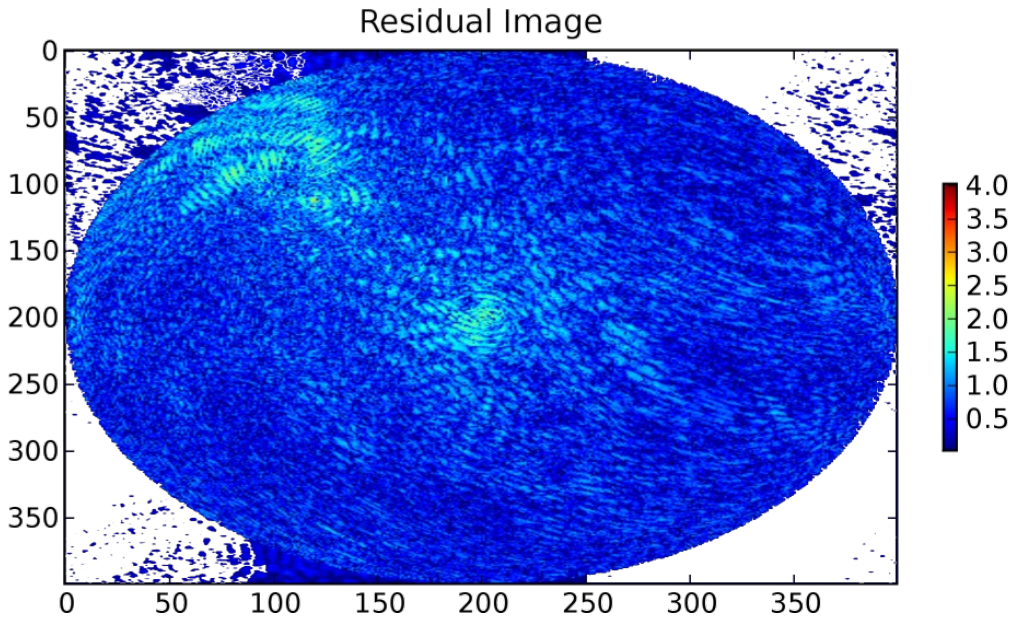


PAPER Example: Cyg A, Cas A, MEM





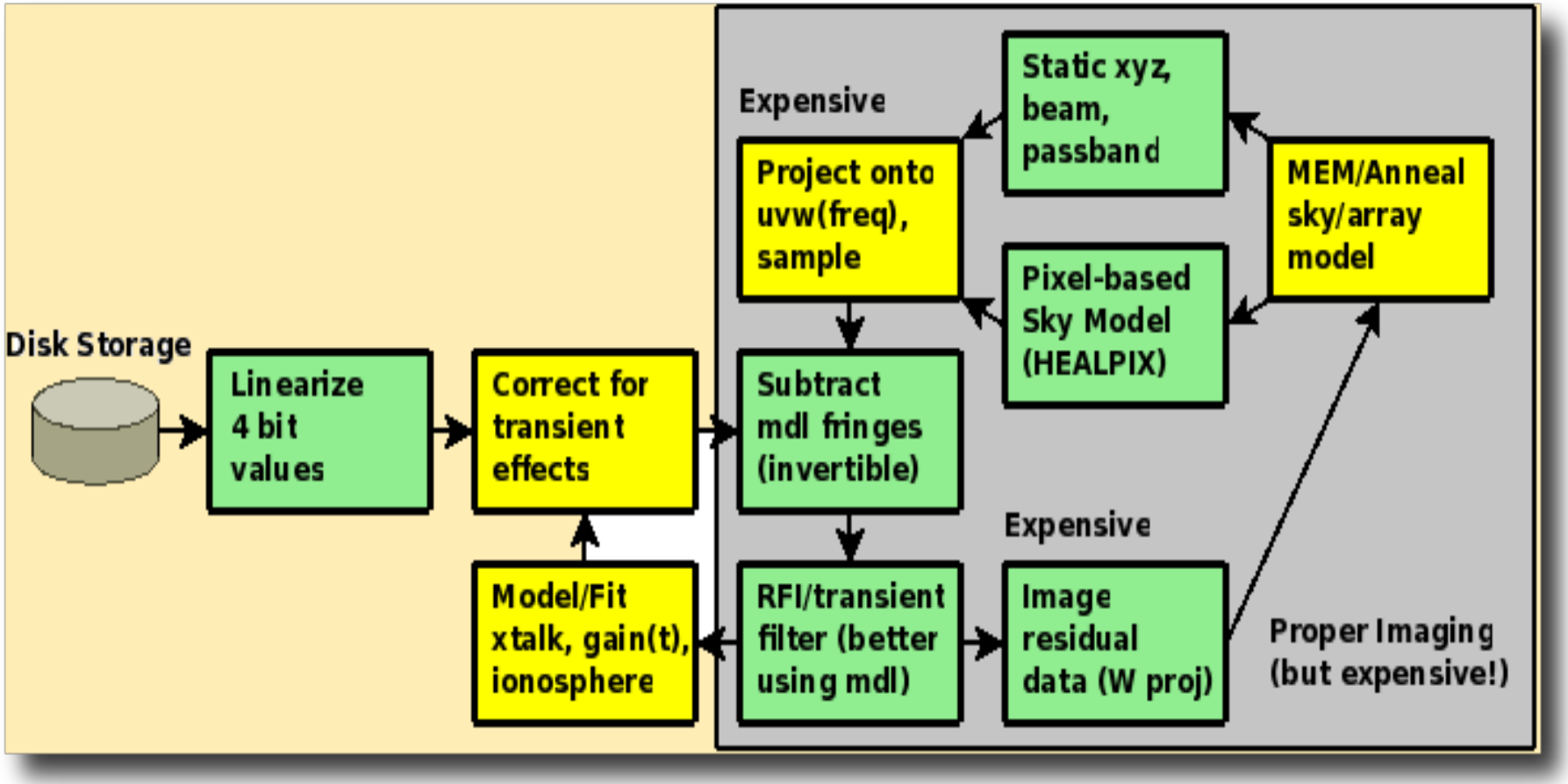




Challenge 3: Proper Imaging

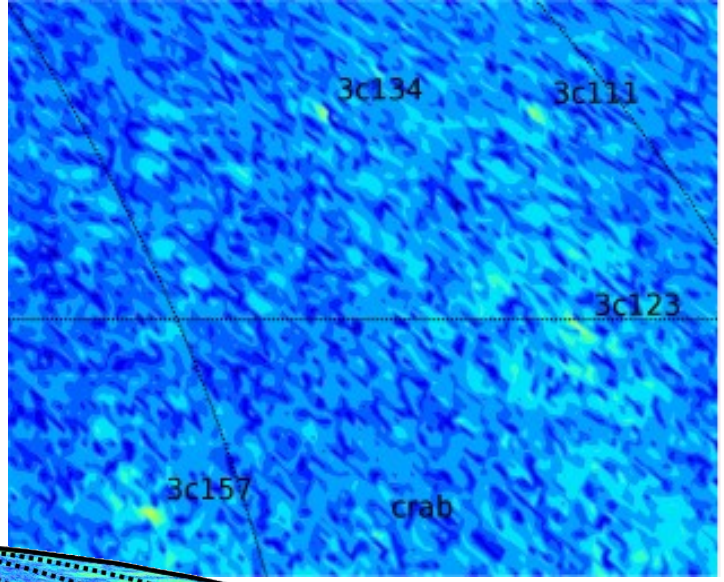
- Classic inverse problem: easier to guess the answer, check, and correct than to solve
- Simulate on reasonable time-scales
- Maximize fidelity of forward imaging (make better guesses)
- For distributed flux, avoid pixelization effects (compute using spherical harmonics)

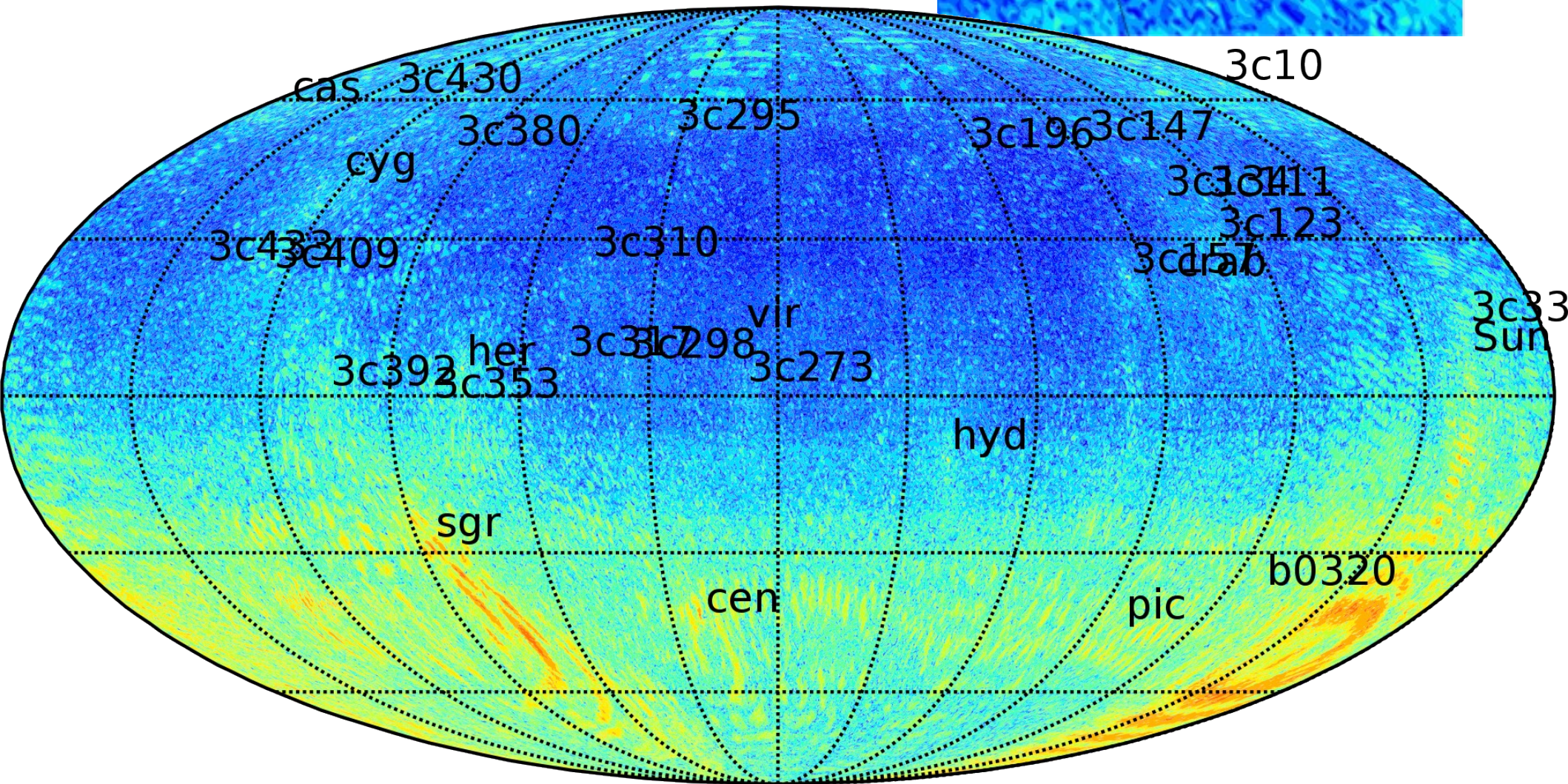
- Many parameters are strongly degenerate, requiring simultaneous fitting to tease them apart.
- Proper image deconvolution involves using the full measurement equation.
- Various parameters (ionosphere,gain,xtalk) change on different timescales.
- Huge parameter space, different variance in parameters -> simulated annealing?
- If parameter space is not smooth, this is not an easy problem.



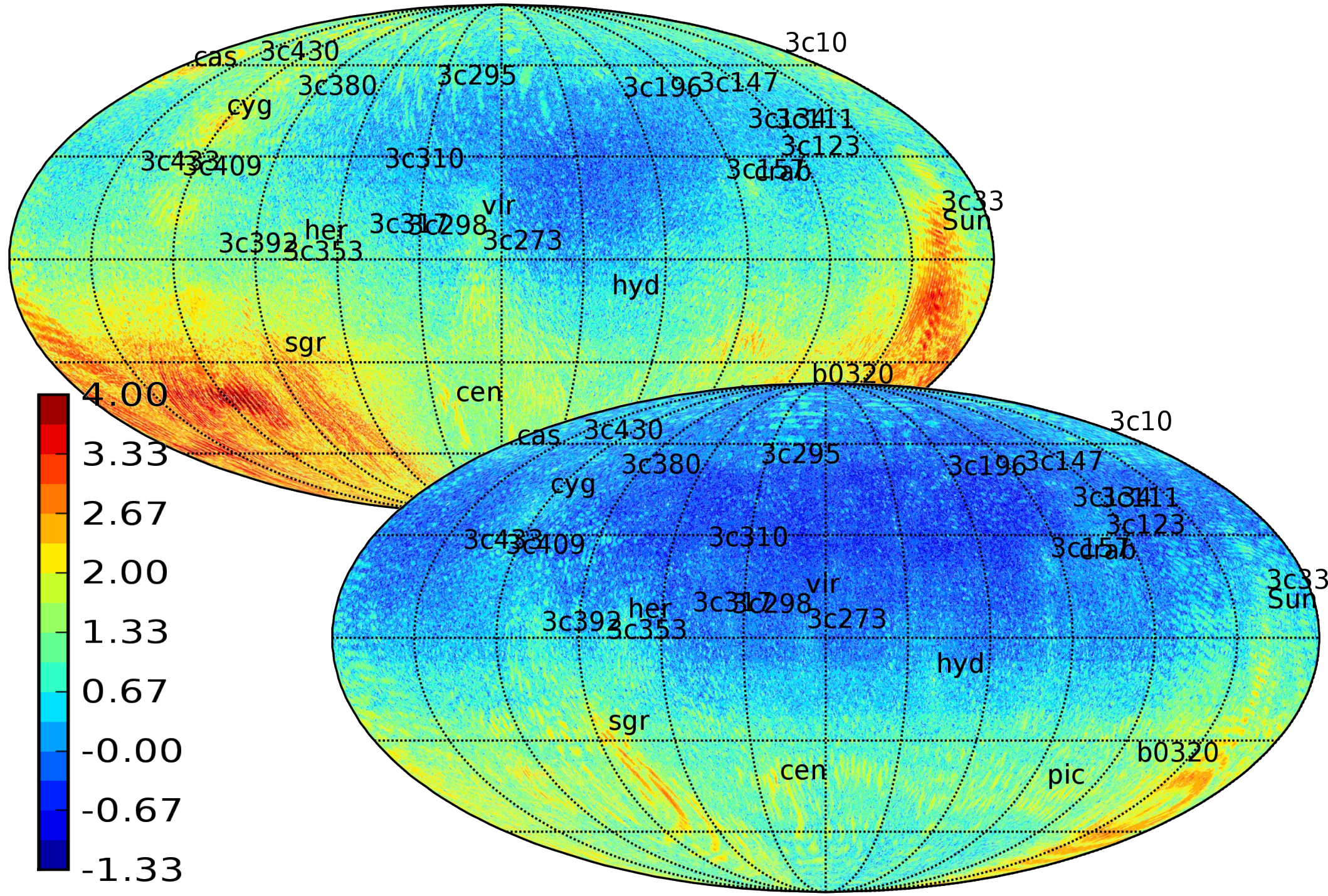
150 MHz PGB-8

- 3 days of 8-element single polarization data
- Spans 138.8 to 174.0 MHz
- Delay/Fringe subtraction of Sun, Cyg, Cas, Vir, Crab
- Map has constant flux scale (but changing noise level)
- Imaged on 15° grid





Removing Top 5 Sources



More to Come!

- Parallelization (running on clusters, probably using pp.py)
- Model distributed flux in spherical harmonics (faster)
- Close the loop on proper imaging
- Improved estimation of image variance for MEM
- Real-time system for modeling ionosphere, crosstalk, gain fluctuations
- Developing tools for communicating with other packages/file formats
- Building a community of developers/users