The Snake In The MeqTree: Using Python For Calibration & Simulation Of Radio Astronomical Data

Oleg Smirnov

smirnov@astron.nl

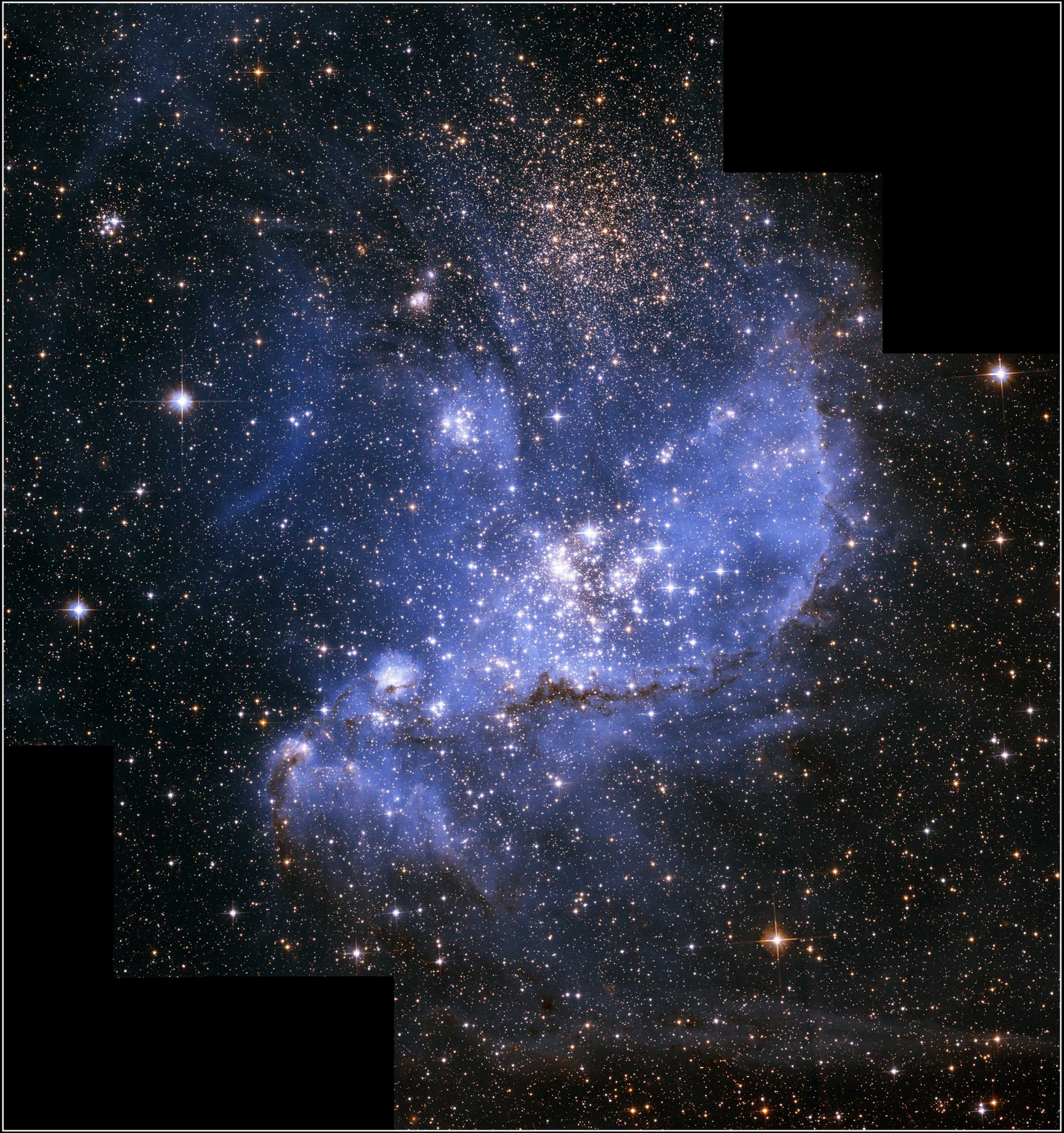
Netherlands Foundation For Research In Astronomy (ASTRON)

Outline

- The lonely, masochistic world of radio astronomical observations
- Calibration: putting the toothpaste back in the tube
- Donald Rumsfeld's strange but lucrative modelling career
- MeqTrees: models gone wild
- A large snake saves the world

Optical Telescopes: Point & Click?

=

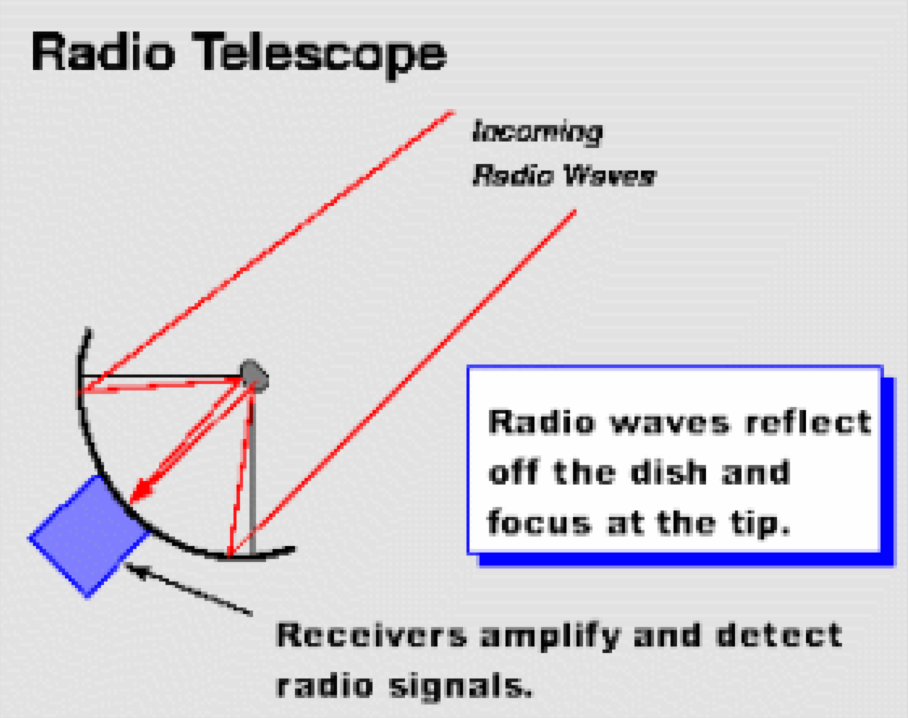


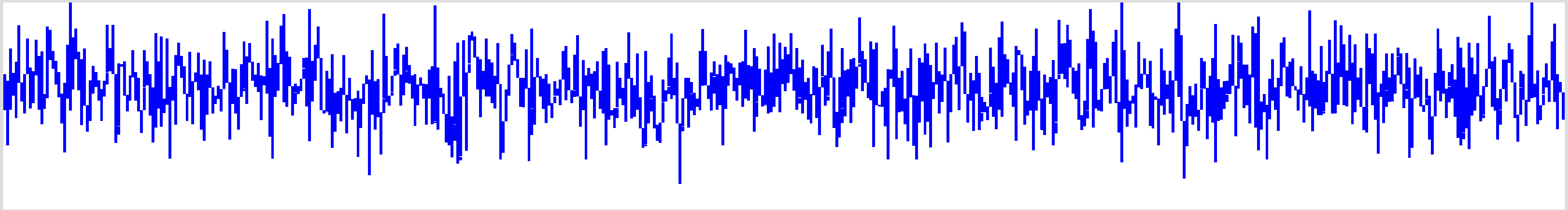
NGC 346 in the Small Magellanic Cloud
Hubble Space Telescope • ACS

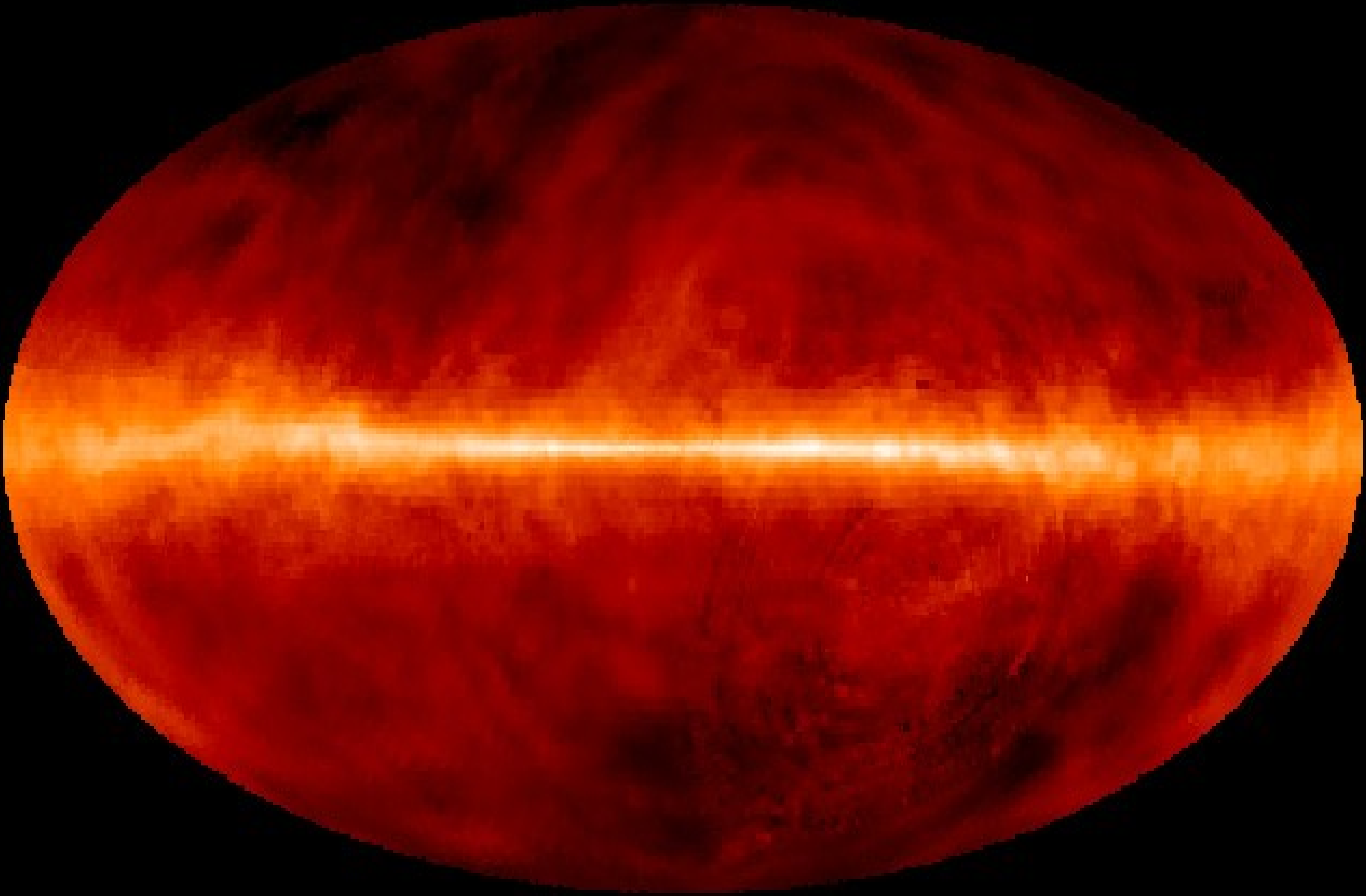
NASA, ESA and A. Nota (STScI)

STScI-PRC05-04

Radio Telescopes:
Point & what's the point?

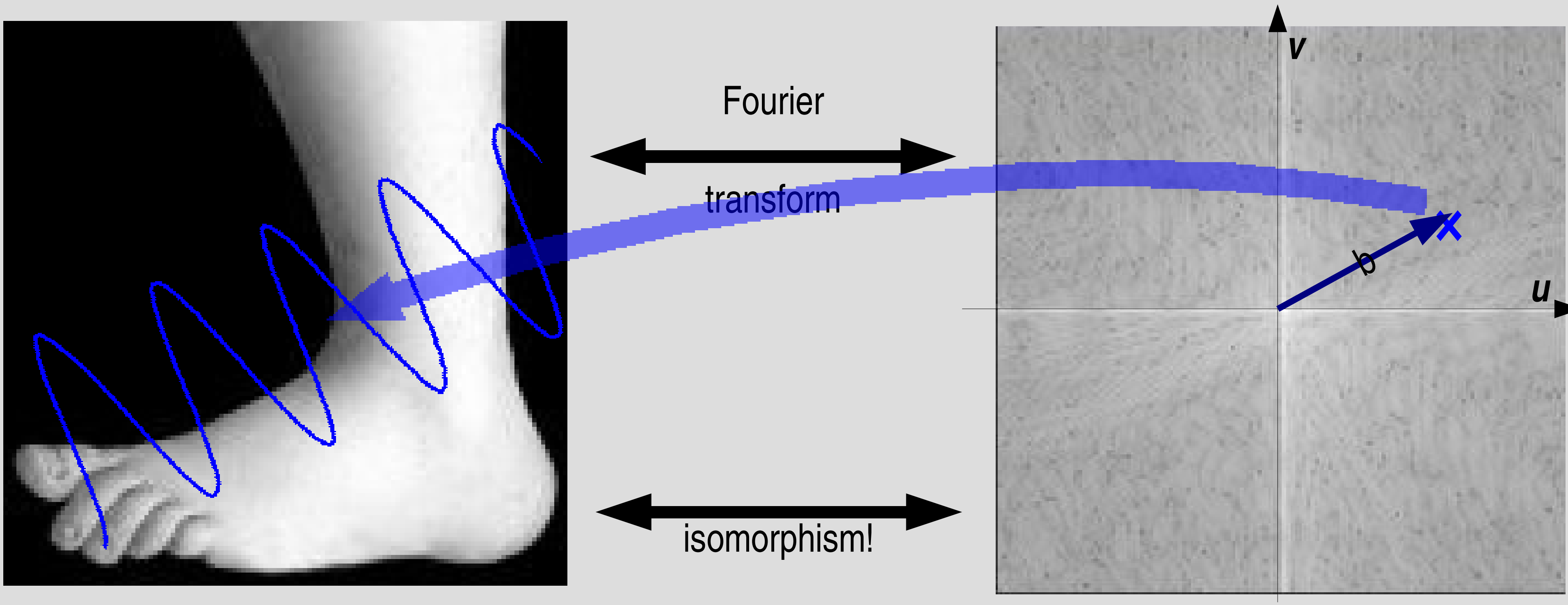






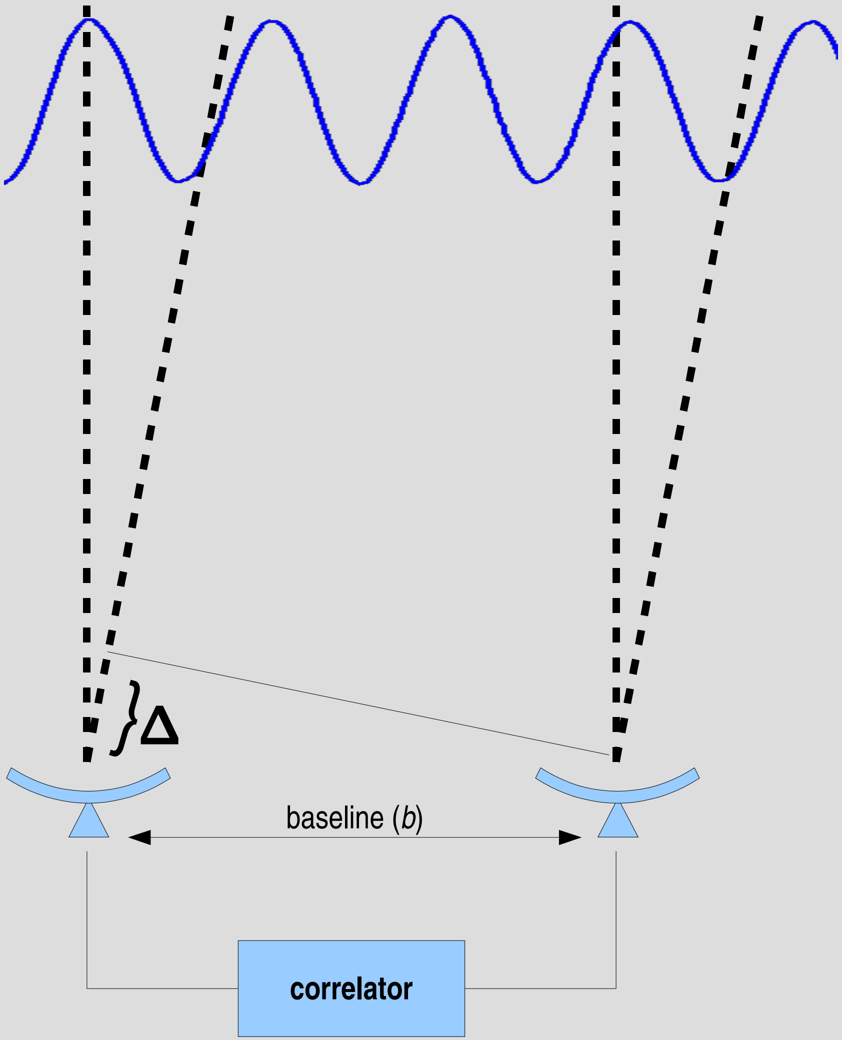
all-sky map of neutral hydrogen (λ=21cm)

Observing In The Fourier Plane

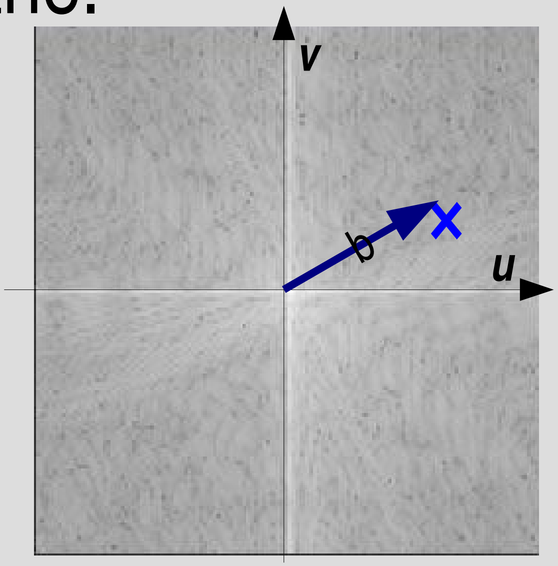


$$F(u,v)=\iint_{-\infty}^{\infty} f(x,y)\exp(-2\pi i(ux+vy))\,dx\,dy$$

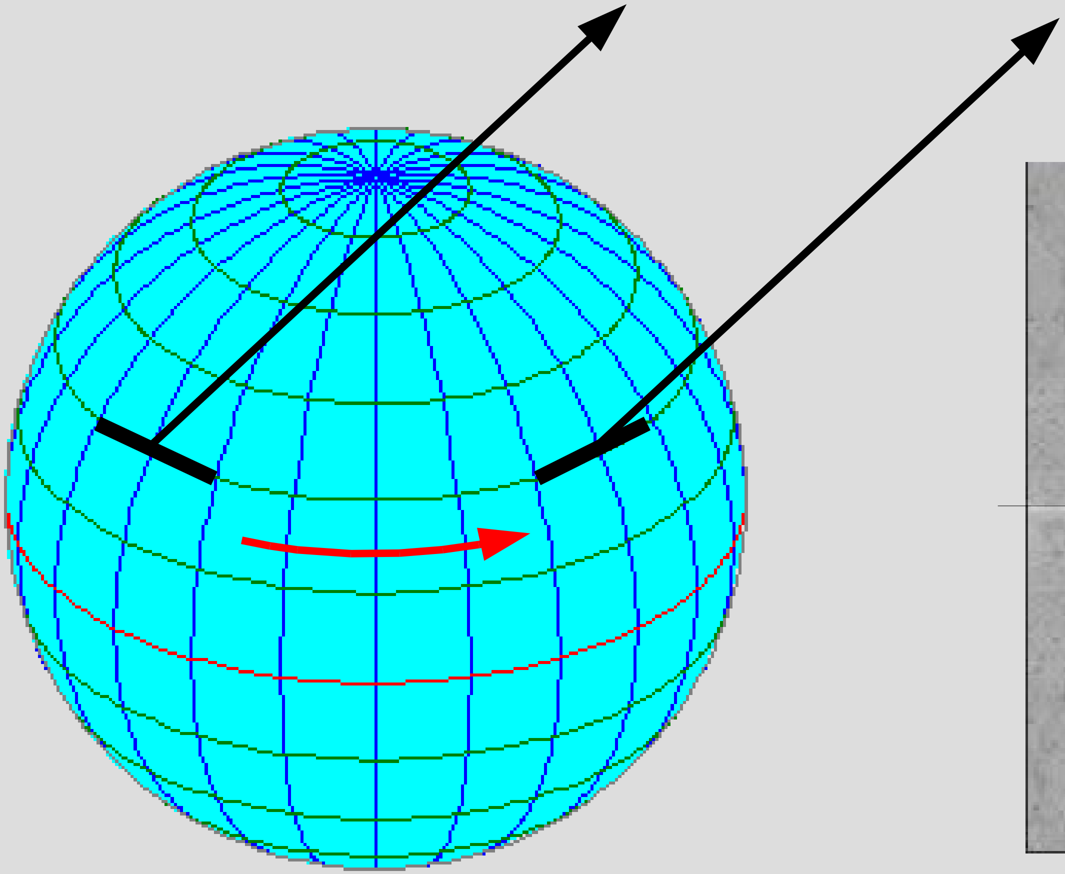
A Simple Radio Interferometer

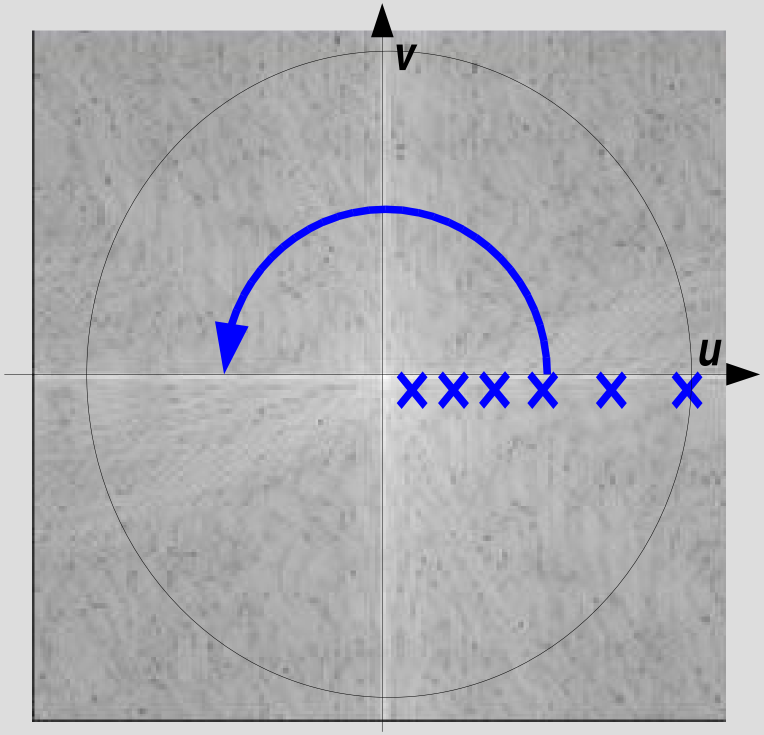


- A correlator multiplies and integrates signals from two dishes
- Samples one point in the *uv* plane.



Covering The UV Plane





- Let the Earth do the hard work!

A multi-wavelength, multi-scale tour of NGC253

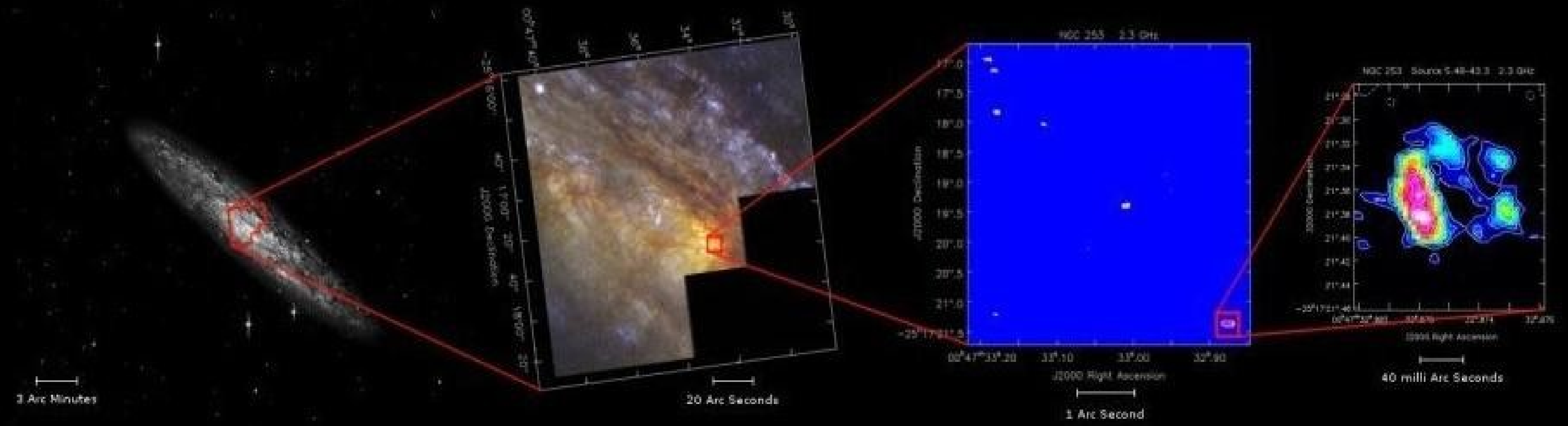
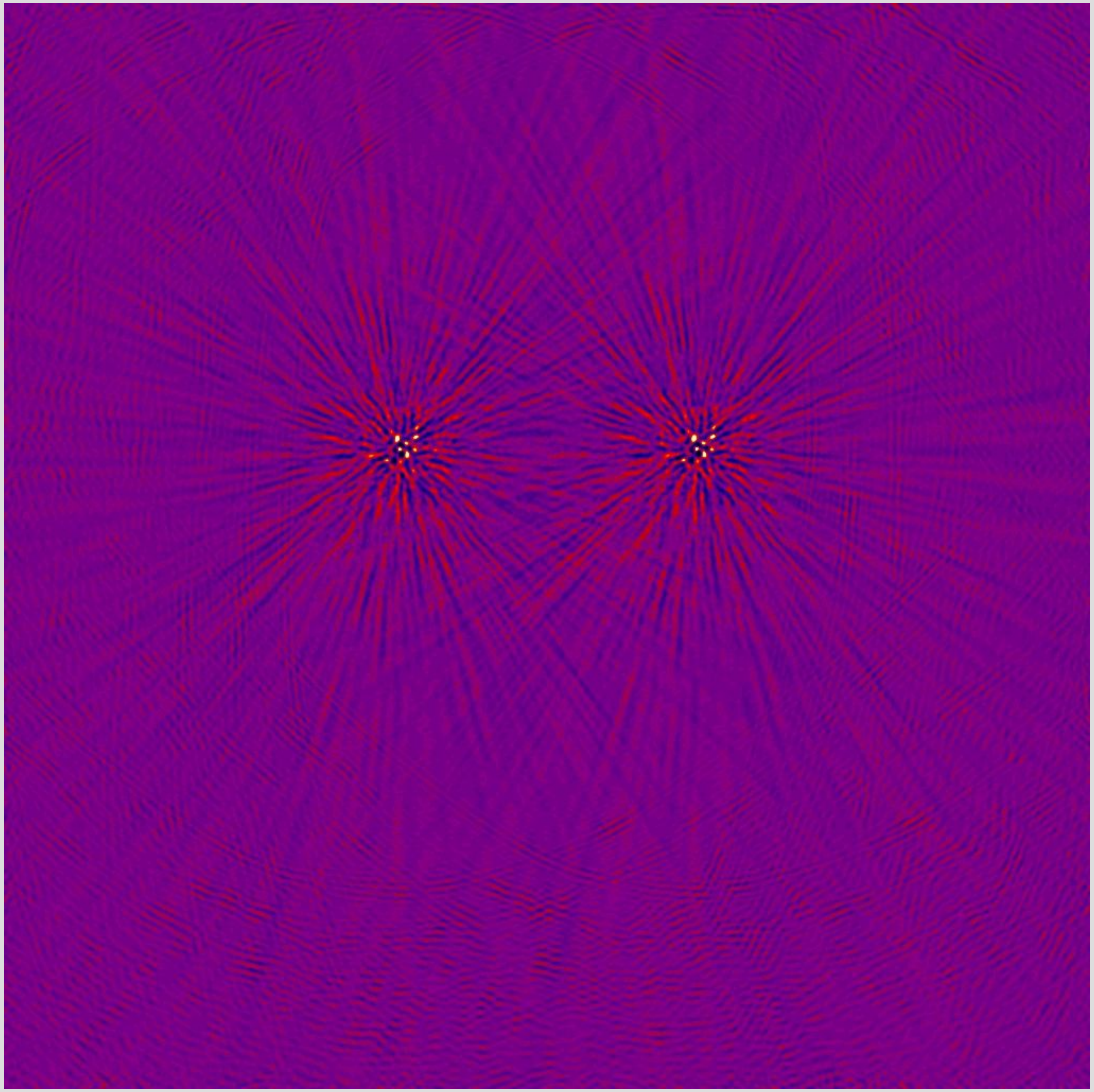


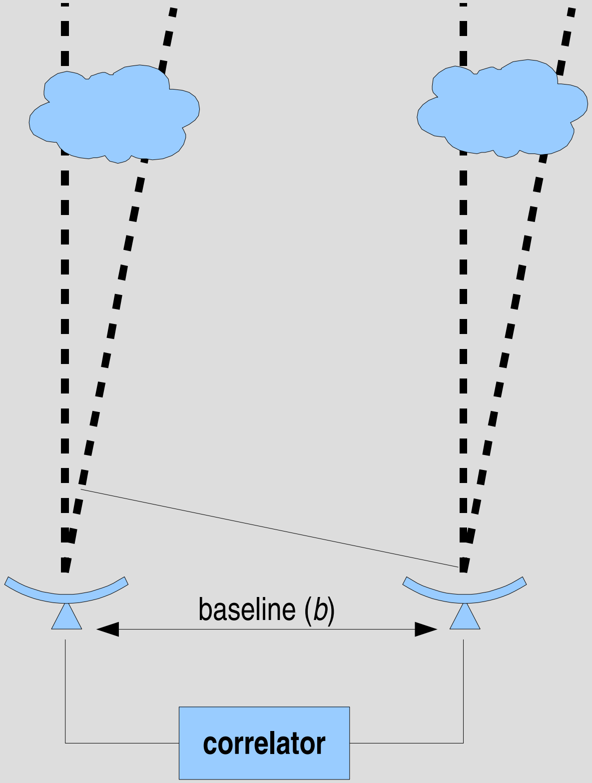
image: Emil Lenc (Swinburne U., Australia)

- resolution of a single telescope: wavelength / diameter
- HST: *d*=2.4m, λ=475nm: resolution ~ **0.05″**
  (1" = 1€ / 4km)
- resolution of interferometer: **wavelength / baseline**
- compact radio array (WSRT, VLA):
  λ=21cm, *b*=2km: resolution ~ **20″**
- VLBI:
  λ=3mm, b=10000km: resolution **~ 0.00006″ !!!**

Calibration: Living With Observational Errors

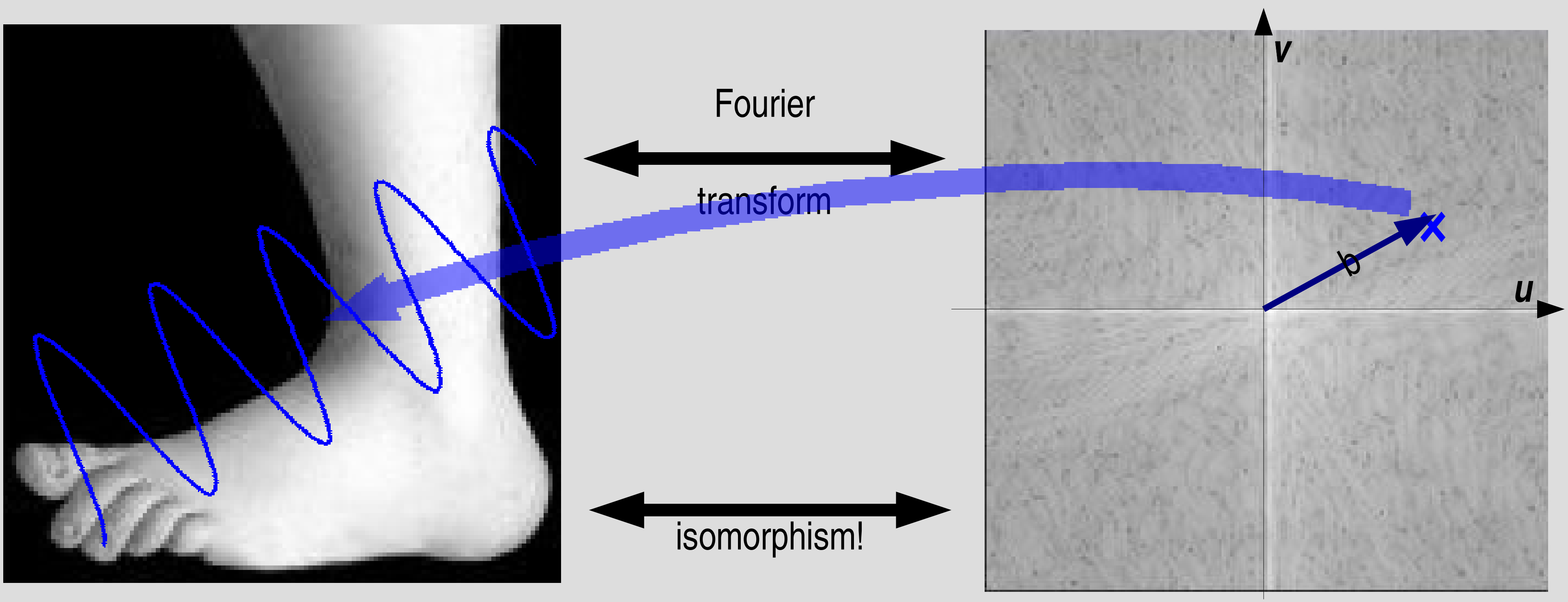
SPLAT!





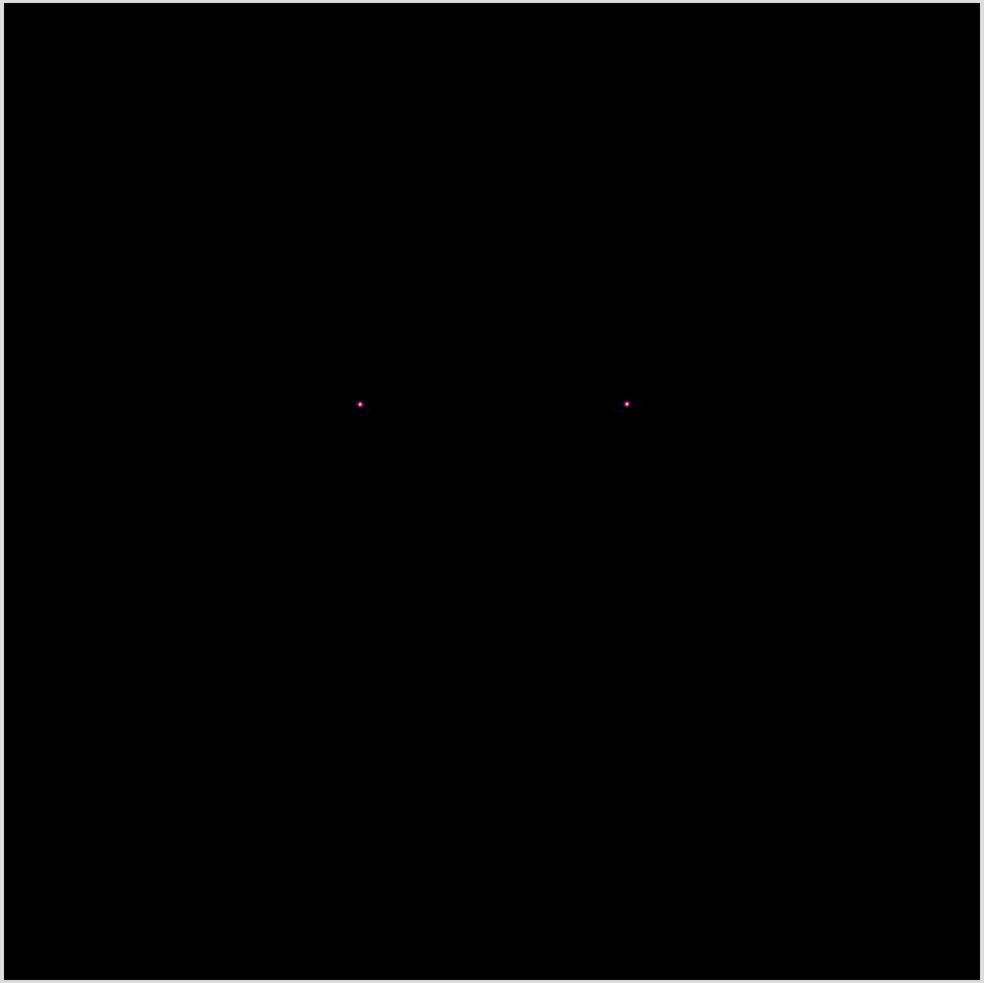
- Phase errors change the apparent “sky position”

Errors In the *UV* Plane: Tangled Up (In Blue)

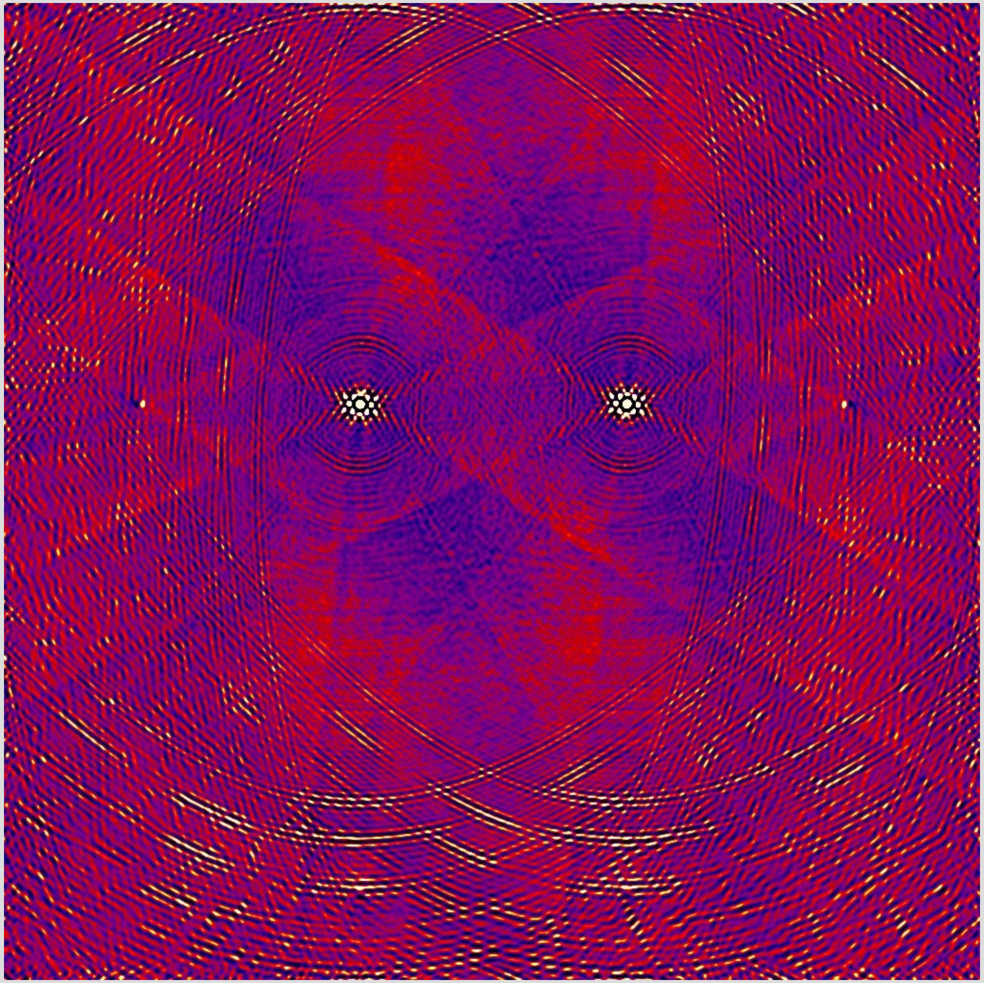


- Fourier transform (& inverse) is an *integration*
- Error at any *uv* point affects the entire image
- Signal at any image point affects the entire *uv* plane

A Hidden Splat

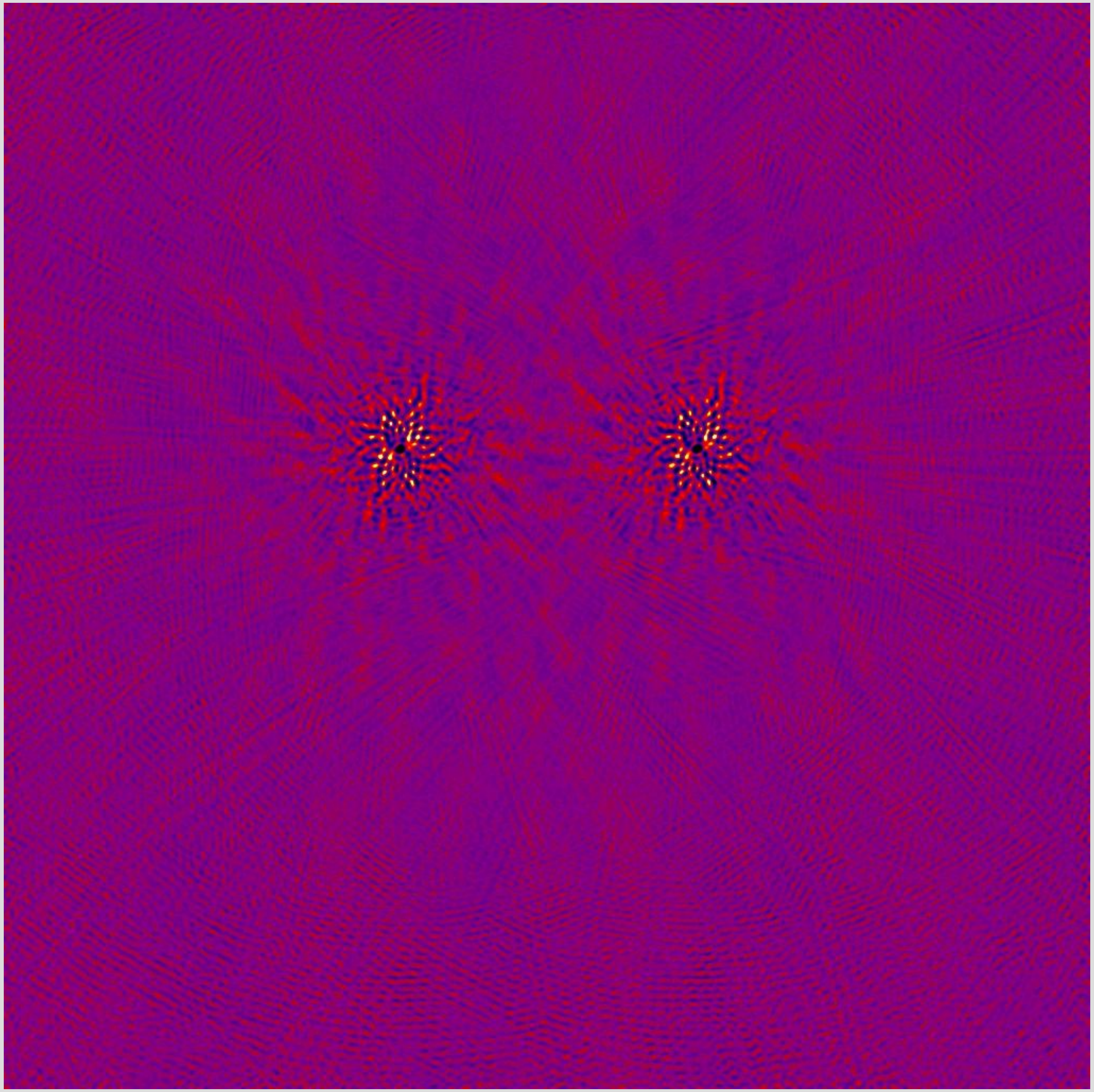


colormap range:
0 ~ 0.5 Jy



colormap range:
±1.5 mJy

Putting The Toothpaste Back In The Tube



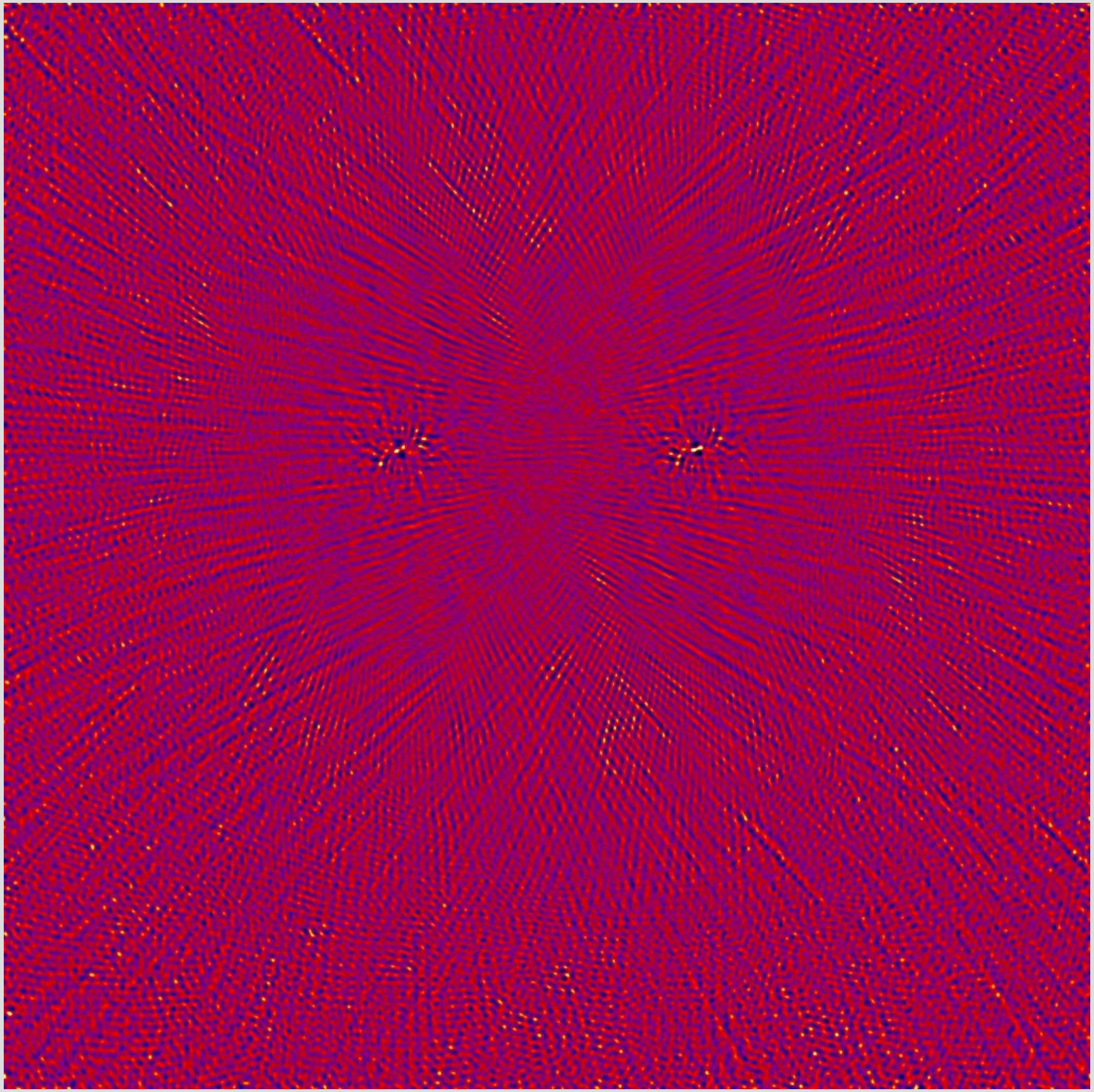
- make a *combined* model of the bright sources + instrumental errors
- fit model to data
- subtract best-fitting model from data and look for residual signal
- dynamic range = ratio between brightest and faintest detectable signal

Left: residuals after fitting a 0th-order model

colormap: ±5 mJy

dynamic range: 1:100

Putting The Toothpaste Back In The Tube

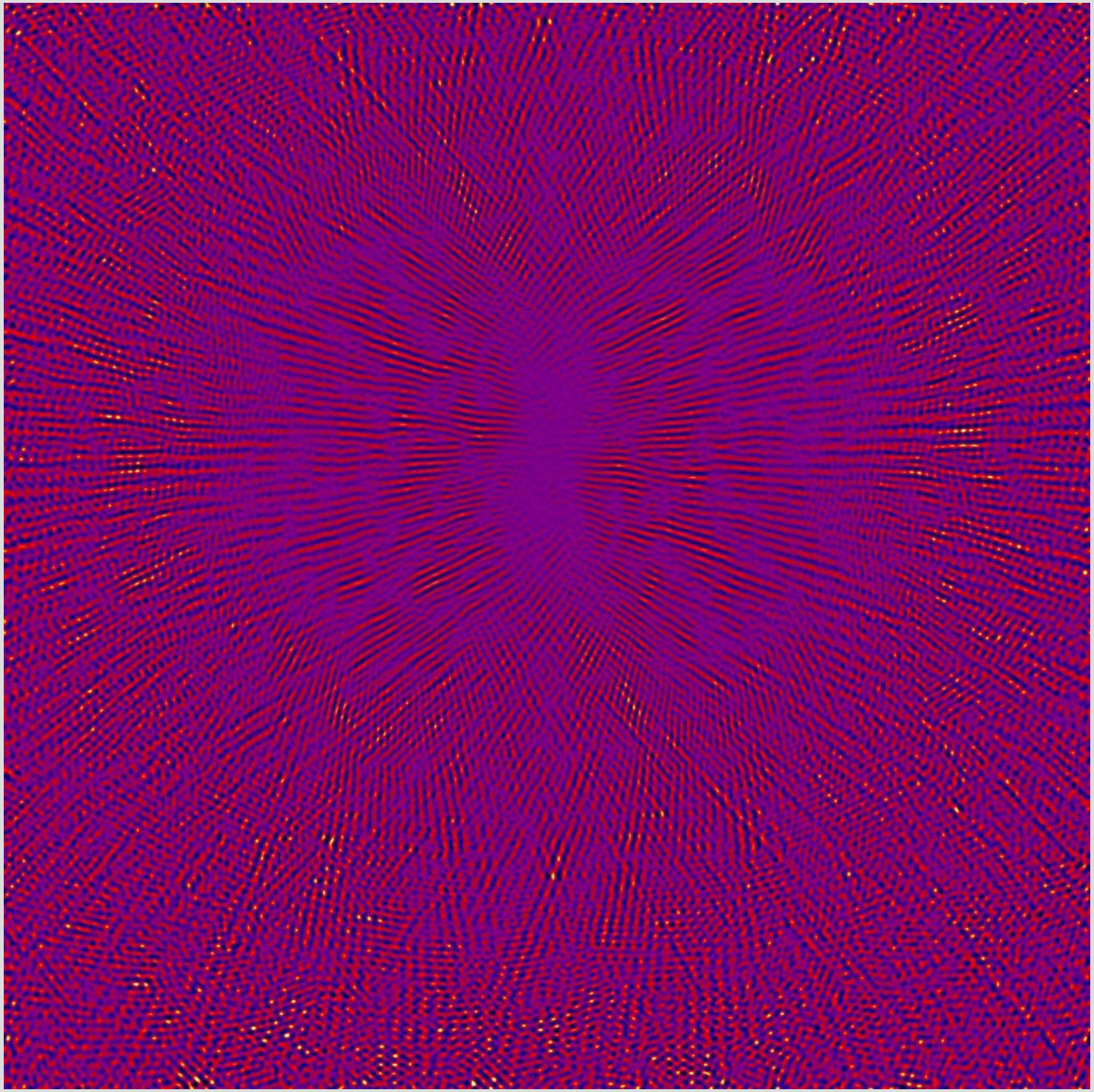


- image is dominated by "crud" (i.e. signal from the bright sources that is *unaccounted for* by the model)
- dynamic range limited by level of crud
- by improving the model we increase the D/R

Left: residuals after fitting a linear model

colormap: ±0.2 mJy
D/R: 1:2500

Putting The Toothpaste Back In The Tube

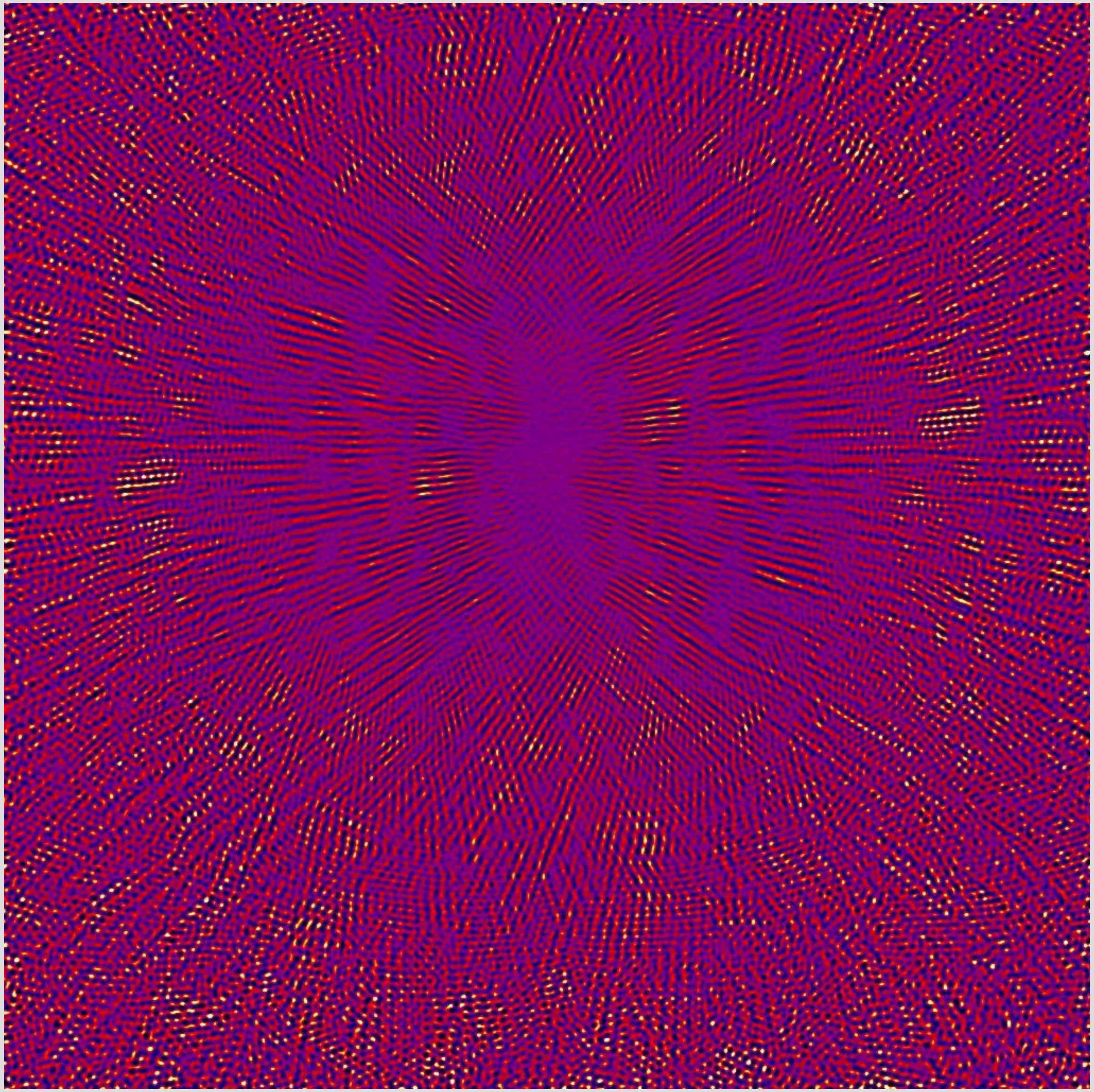


- image is dominated by "crud" (i.e. signal from the bright sources that is *unaccounted for* by the model)
- dynamic range limited by level of crud
- by improving the model we increase the D/R

Left: residuals after fitting a 2nd-order model

colormap: ±10 μJy
D/R: 1:50,000

Putting The Toothpaste Back In The Tube



- image is dominated by "crud" (i.e. signal from the bright sources that is *unaccounted for* by the model)
- dynamic range limited by level of crud
- by improving the model we increase the D/R

Left: residuals after fitting a 3rd-order model

colormap: ±0.2 μJy
D/R: 1:2,500,000

Putting The Toothpaste Back In The Tube

- image is dominated by "crud" (i.e. signal from the bright sources that is *unaccounted for* by the model)
- dynamic range limited by level of crud
- by improving the model we increase the D/R

Left: residuals after fitting a 4th-order model

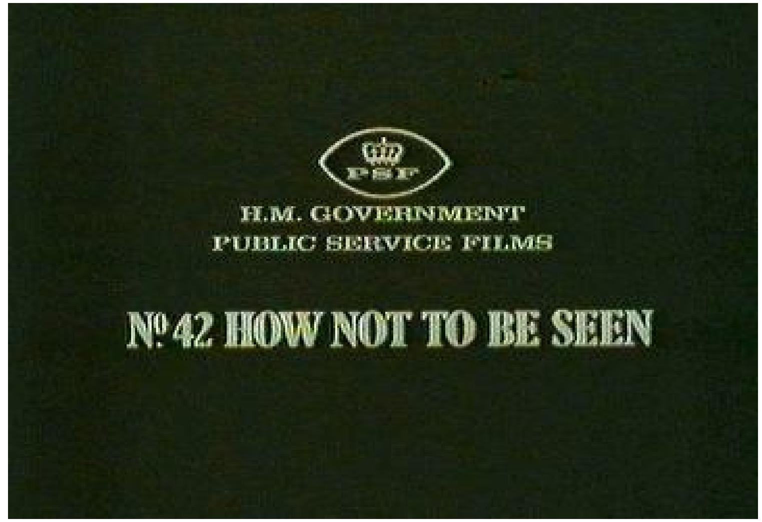
colormap: ±10 nJy
D/R: 1:50,000,000

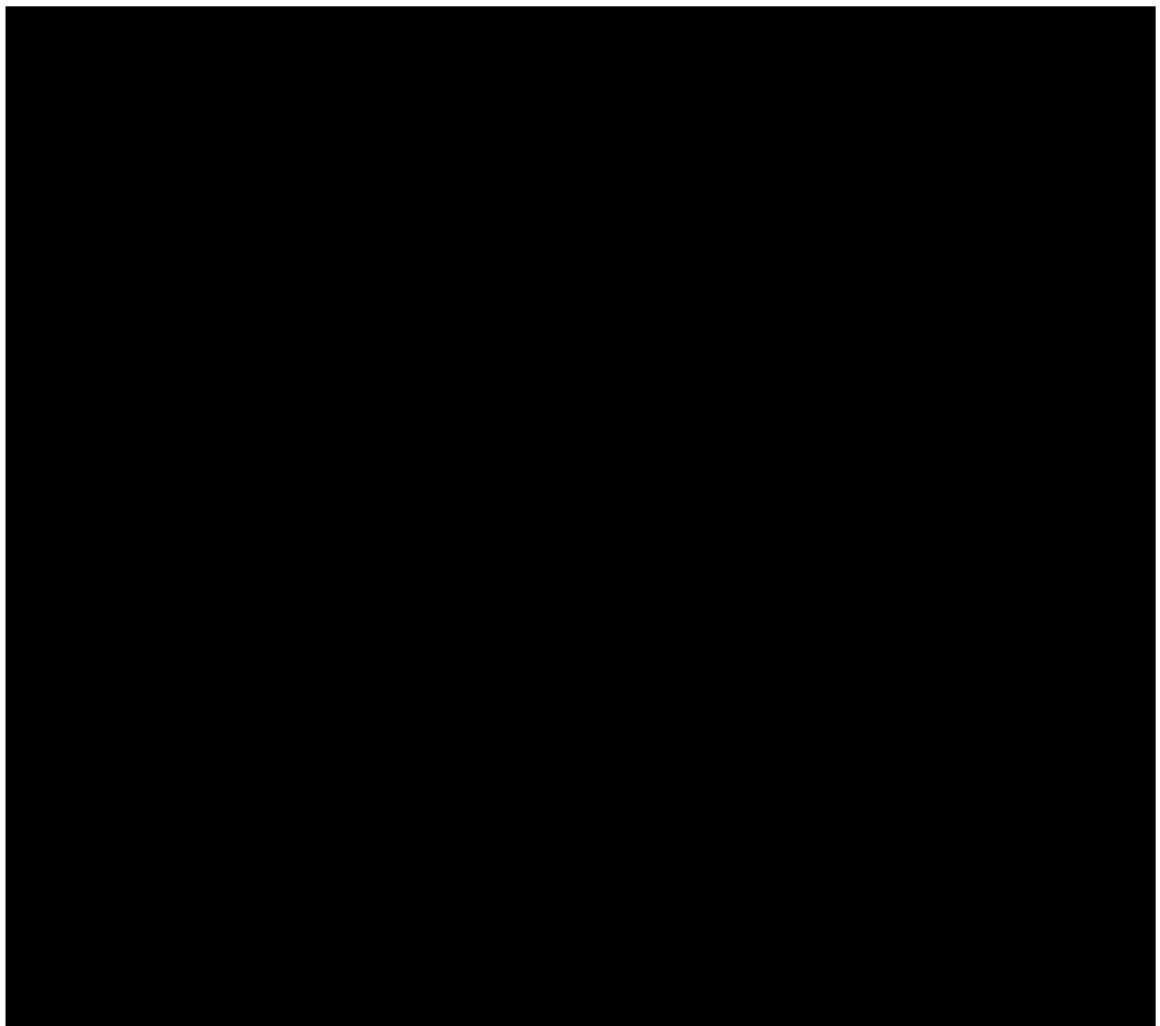
Putting The Toothpaste Back In The Tube

- image is dominated by “crud” (i.e. signal from the bright sources that is *unaccounted for* by the model)
- dynamic range limited by level of crud
- by improving the model we increase the D/R

Left: residuals after fitting a 5th-order model

colormap: ±10 nJy
D/R: 1:50,000,000

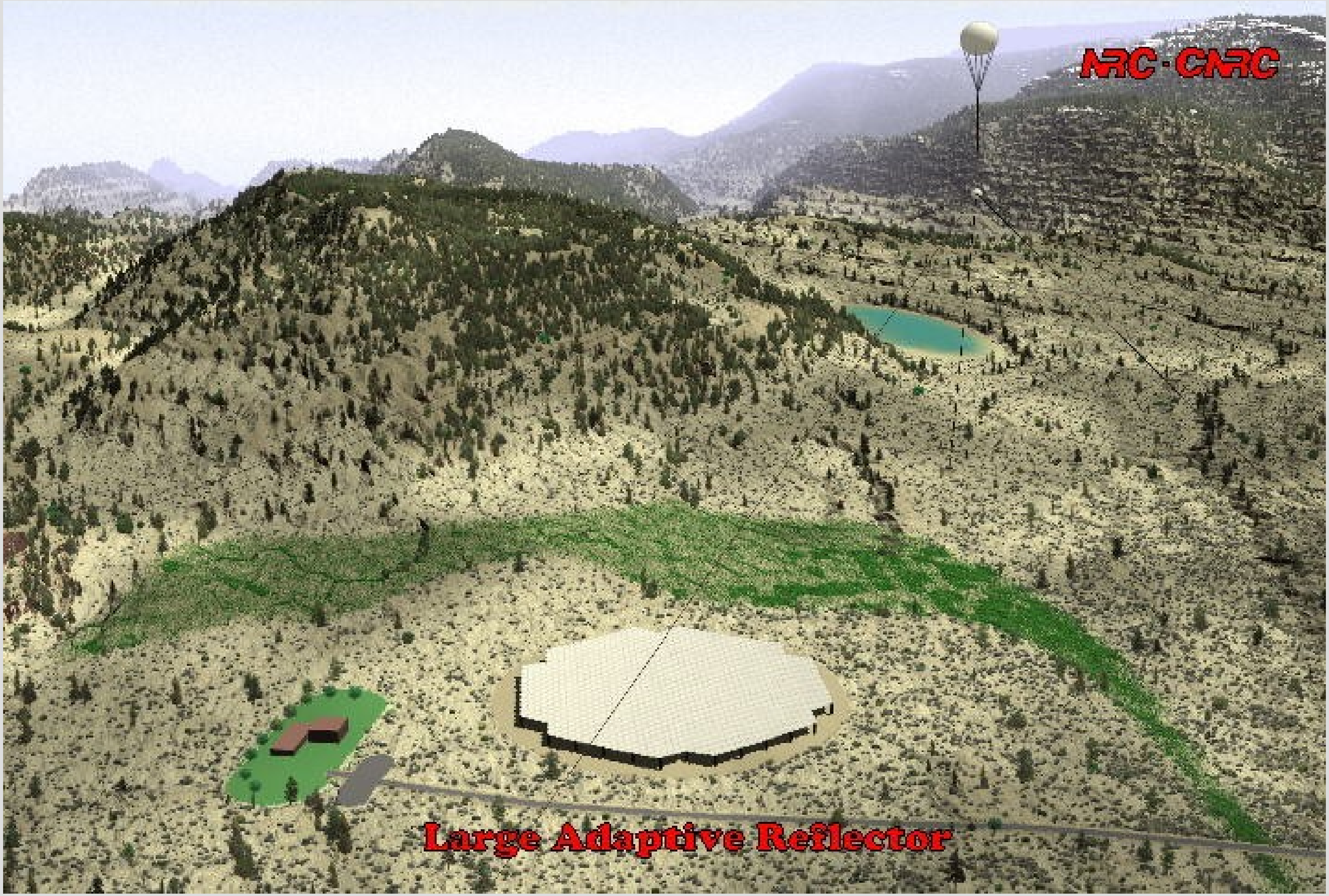

PSF
H.M. GOVERNMENT
PUBLIC SERVICE FILMS
№ 42 HOW NOT TO BE SEEN



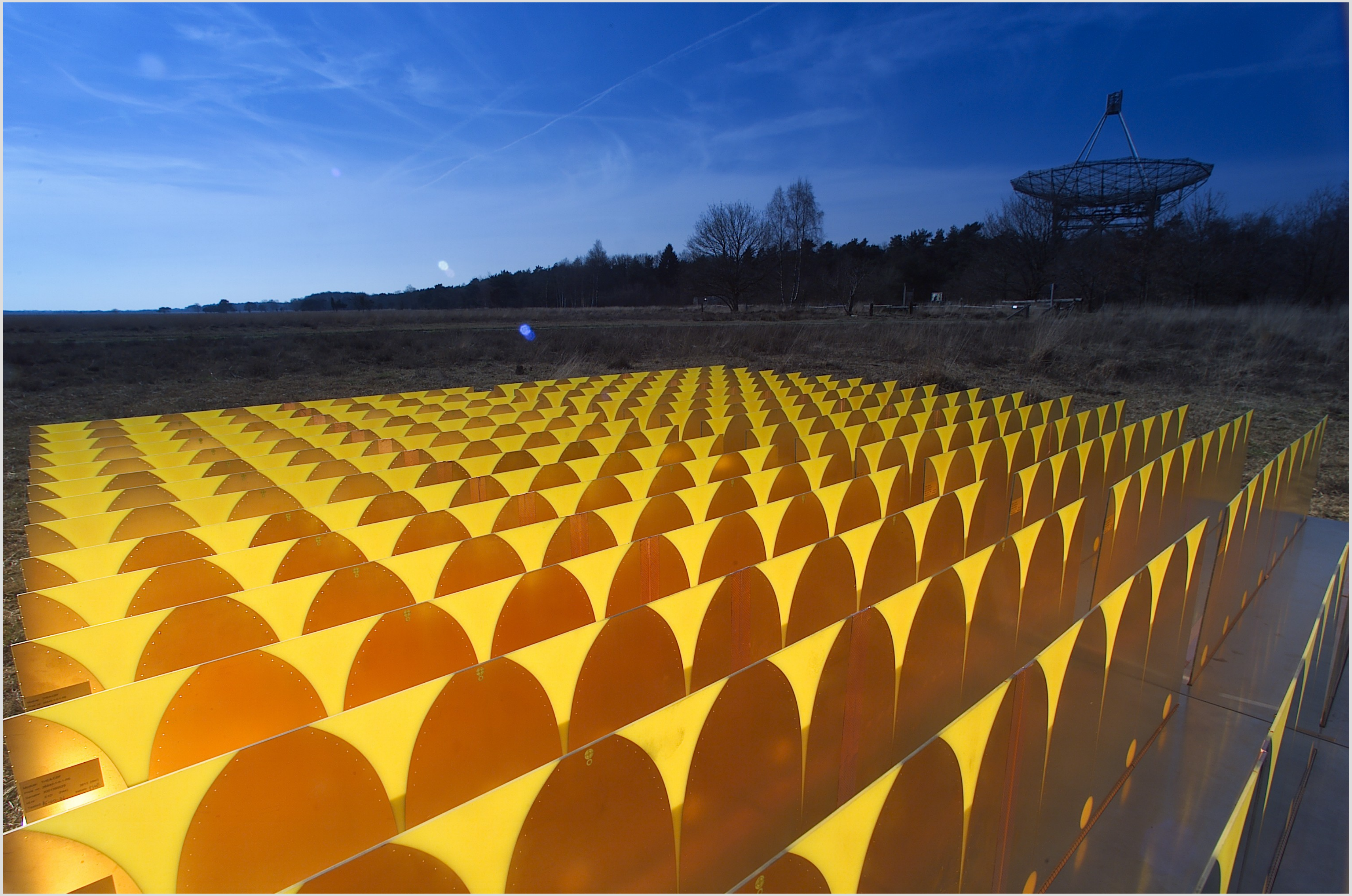
The Past:
Beautiful Engineering

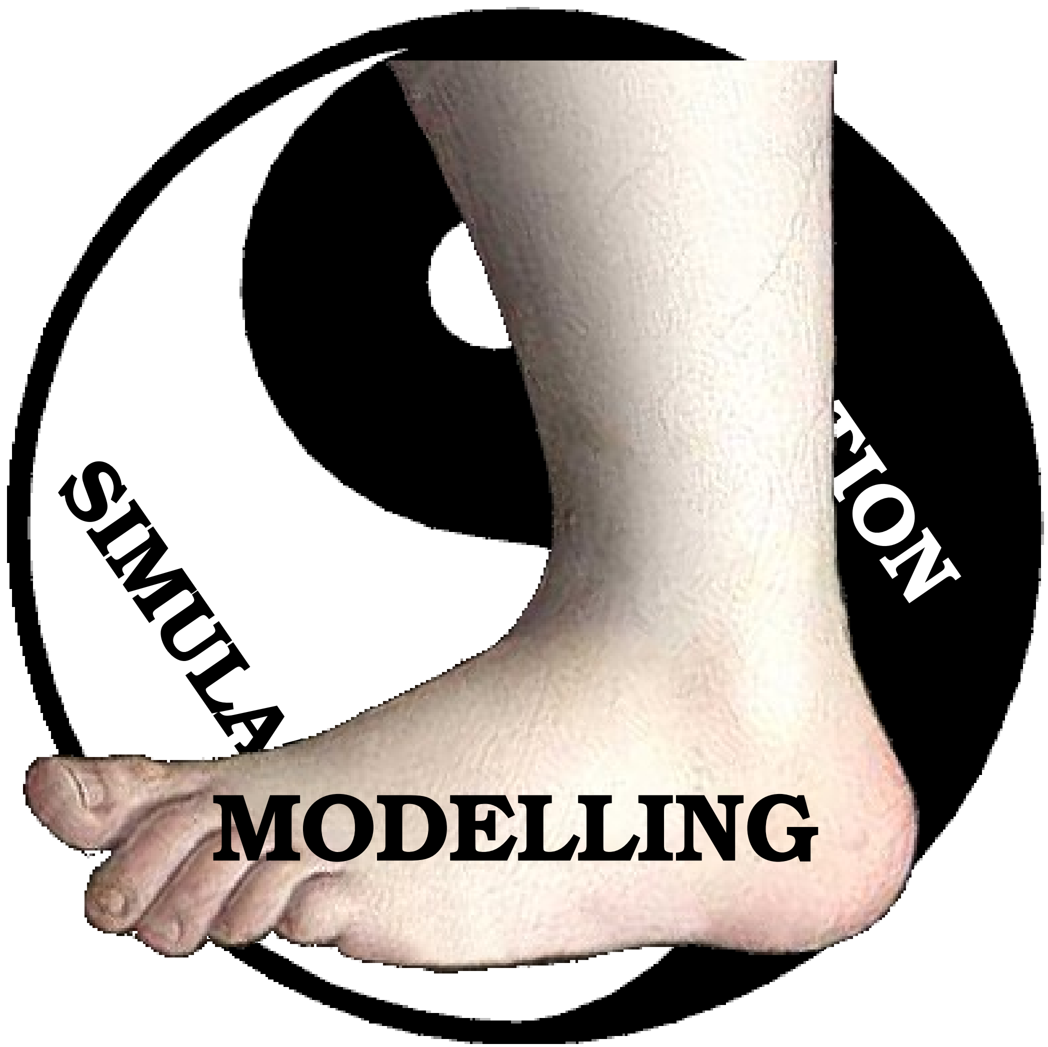
The Present: Cheap But Plentiful

The Future:
Just Plain Wierd



...And Weirder




SIMULA
TION
MODELLING

The Modelling Conundrum

“... there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns -- the ones we don't know we don't know.”

-- *Donald Rumsfeld, on the calibration of radio astronomical data*

MeqTrees aka Timba:
A Modelling Toolkit

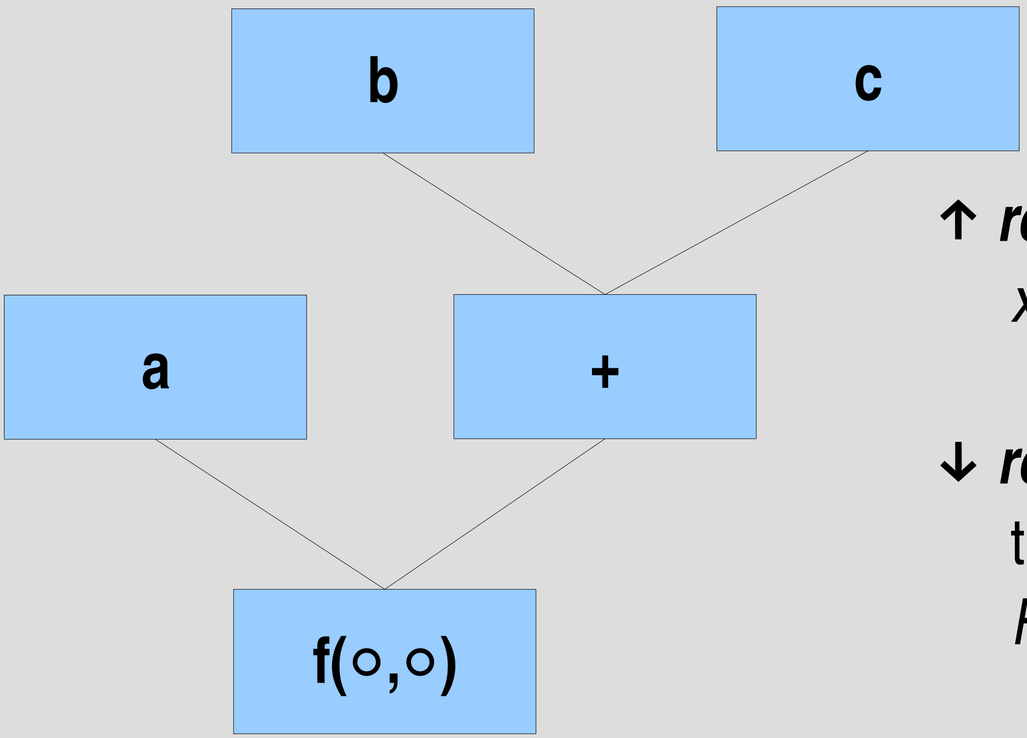
The Measurement Equation

$$R_j R_k^* = G_{Rj} G_{Rk}^* \left[E_{Rj} E_{Rk}^* e^{-i\left(\phi_j - \phi_k\right)} + D_{Rk}^* E_{Rj} E_{Lk}^* e^{-i\left(\phi_j + \phi_k\right)} + D_{Rj} E_{Lj} E_{Rk}^* e^{i\left(\phi_j + \phi_k\right)} + D_{Rj} D_{Rk}^* E_{Lj} E_{Lk}^* e^{i\left(\phi_j - \phi_k\right)} \right]$$

$$V_{ik} = \sum_n J_i X_n \bar{J}_k$$

$$R_j L_k^* = G_{Rj} G_{Lk}^* \left[E_{Rj} E_{Lk}^* e^{-i\left(\phi_j + \phi_k\right)} + D_{Lk}^* E_{Rj} E_{Rk}^* e^{-i\left(\phi_j - \phi_k\right)} + D_{Rj} E_{Lj} E_{Lk}^* e^{i\left(\phi_j - \phi_k\right)} + D_{Rj} D_{Lk}^* E_{Lj} E_{Rk}^* e^{i\left(\phi_j + \phi_k\right)} \right].$$

MeqTrees = Expressions



↑ ***requests:*** an *N*-dimensional gridding *x,y,...*

↓ ***results:*** the value of some (scalar or tensor) function *F* over the given grid: *F(x,y)*

○ ***tree:*** builds compound function from primitive building blocks.

$$F(x,y) = f(a(x,y), b(x,y) + c(x,y))$$

Derivatives For Free

Parm "A":

$a_0+a_1x+a_2y$

$$A(x,y;\mathbf{a})\quad \frac{\partial A}{\partial a_0}(x,y;\mathbf{a})\quad \frac{\partial A}{\partial a_1}(x,y;\mathbf{a})\quad \ldots$$

Parm "B":

$b_0+b_1x^2y^2$

$$B(x,y;\mathbf{b})\quad \frac{\partial B}{\partial b_0}(x,y;\mathbf{b})\quad \frac{\partial B}{\partial b_1}(x,y;\mathbf{b})\quad \ldots$$

Function "f":

$f(\circ,\circ)$

$$F(x,y;\mathbf{a},\mathbf{b})\quad \frac{\partial F}{\partial a_i}(x,y;\mathbf{a},\mathbf{b})\quad \frac{\partial F}{\partial b_j}(x,y;\mathbf{a},\mathbf{b})\quad \ldots$$

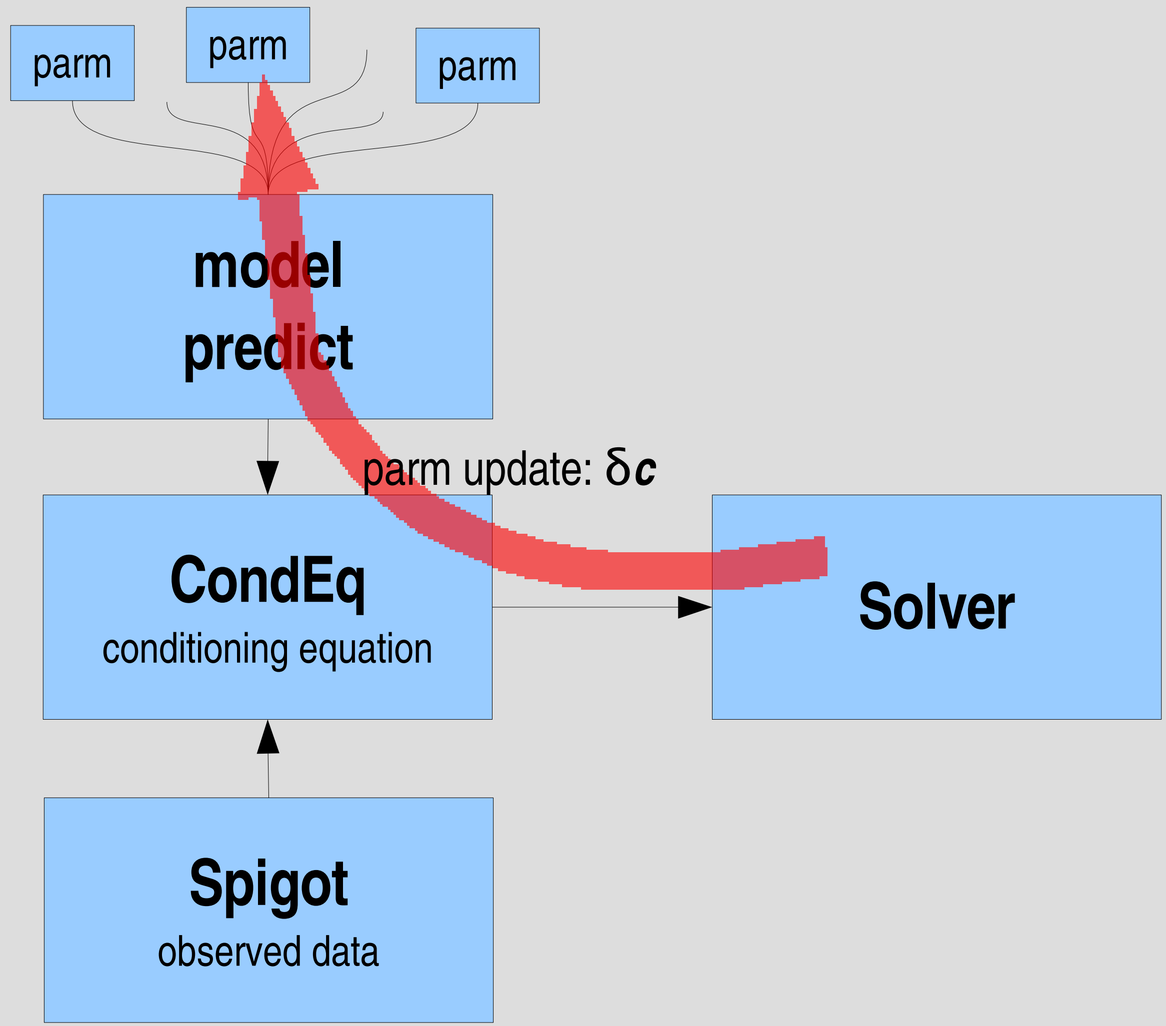
$$F(x,y;\mathbf{a},\mathbf{b})=f(A(x,y;\mathbf{a}),B(x,y;\mathbf{b}))$$

Solving For Parameters

$$F(x,y;\mathbf{c}) \quad \frac{\partial F}{\partial c_i}(x,y;\mathbf{c}) \quad \ldots$$

$$\Delta(x,y;\mathbf{c}) \quad \frac{\partial \Delta}{\partial c_i}(x,y;\mathbf{c}) \quad \ldots$$

$$D(x,y)$$



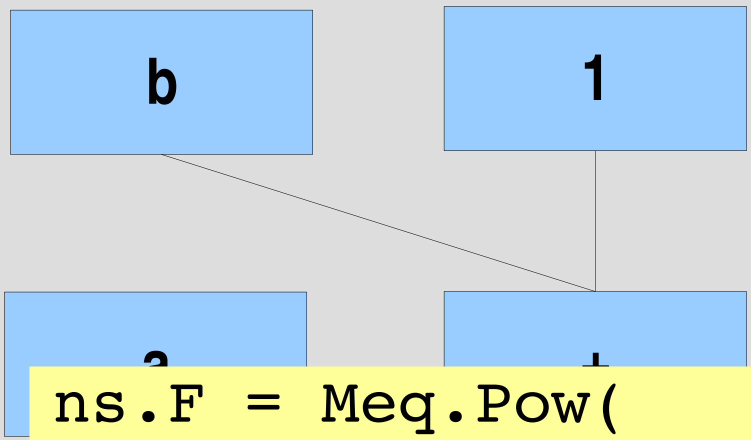
Other MeqTree Features

- Computational kernel (node classes, etc.) written in C++.
- Policy free!
  - Nodes know nothing about radio astronomy.
  - All domain-specific policy *emerges* from the tree structure.
- Local intelligence
  - Cache and reuse results, minimize computations, etc.
- Reasonably fast
  - within a factor of 2 of specifically optimized code.
- Multithreaded; in the future distributed.

Runaway Complexity

- Infinite flexibility brings infinite complexity.
- Real-life trees: 1000 ~ 20000 ~ 1 million nodes.
- Complexity-wise, same problems as very large computer programs really.
- ...but very different in nature.
- How to construct?
- How to run, examine, test and debug?

TDL: Declaring Trees



- TDL = Tree Definition Language
- Implemeted in terms of Python classes and operator methods.
- Python

```
ns.F = Meq.Pow(
  ns.a << Meq.Parm,
  (ns.b << Meq.Parm(shape=(2,))) + 1
);
```

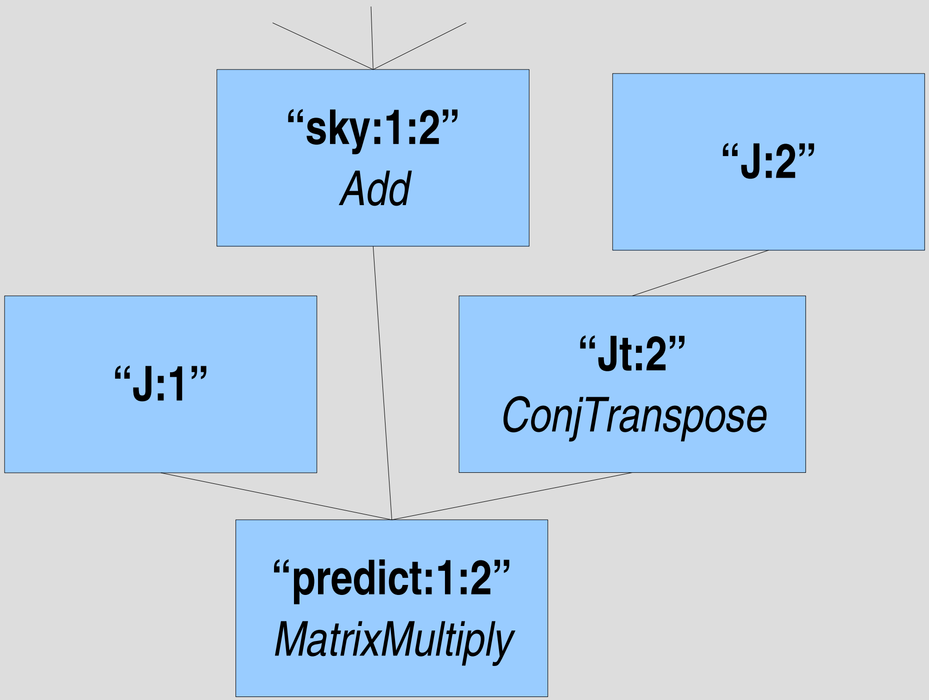
$$F(x,y)=a(x,y)^{b(x,y)+1}$$

```
ns.a = Meq.Parm
ns.b = Meq.Parm(shape=(2,))
ns.c = 1
ns.F = Meq.Pow(ns.a,ns.b+ns.c)
```

TDL: Programming Trees

- Most trees have a regular, repetitive structure.
- Usually, you want to create a bunch of subtrees that look the same, but have different node names.
- E.g. a calibration tree:
  - identical trees to implement the Measurement Equation per each baseline (*pair* of antennas)
  - similar trees to implement instrumental effects per each antenna
  - similar trees to implement models per sky source.
- We want **for** loops and **if-else** statements.

TDL: Programming Trees



$$V_{ik} = \sum_{n} J_i X_n \bar{J}_k$$

- Use node *qualifiers* and loops to create identical subtrees.

```
for ant1,ant2 in baseline_list:
 ns.predict(ant1,ant2) << Meq.MatrixMultiply(
  ns.J(ant1),
  ns.sky(ant1,ant2) << Meq.Add(*[ns.source(src,ant1,ant2)
                                  for src in source_list]),
  ns.Jt(ant2) << Meq.ConjTranspose(ns.J(ant2)));
```

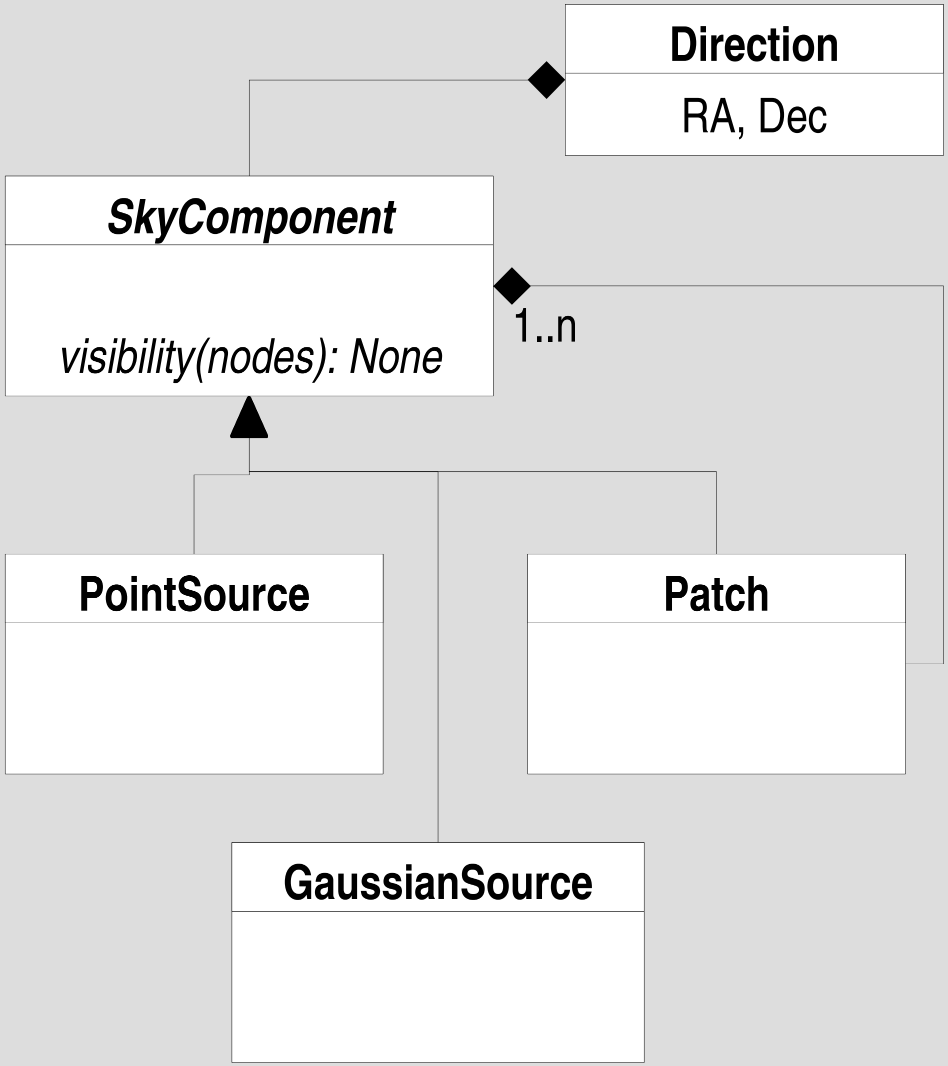
TDL: Modularity

```
def make_antenna_phase_model (Jnode):
 for ant in antenna_list:
  px = Jnode(ant,'phase','x') << Meq.Parm;
  py = Jnode(ant,'phase','y') << Meq.Parm;
  Jnode(ant) << Meq.Matrix22(Meq.Polar(1,px),0,
                              0,Meq.Polar(1,py));
 pass;
...
# create nodes for antenna J terms,
# name them "J:<ant>"
make_antenna_phase_model(ns.J);
```

$$J_k = \begin{pmatrix} i\,\phi_x^{(k)} & 0 \\ 0 & i\,\phi_y^{(k)} \end{pmatrix}$$

- An *unqualified* node can refer to a group of nodes as a whole.
- Easy to pass around collections of nodes.
- ...and of course "normal" Python structures can also be used.

TDL: Object-Orientedness

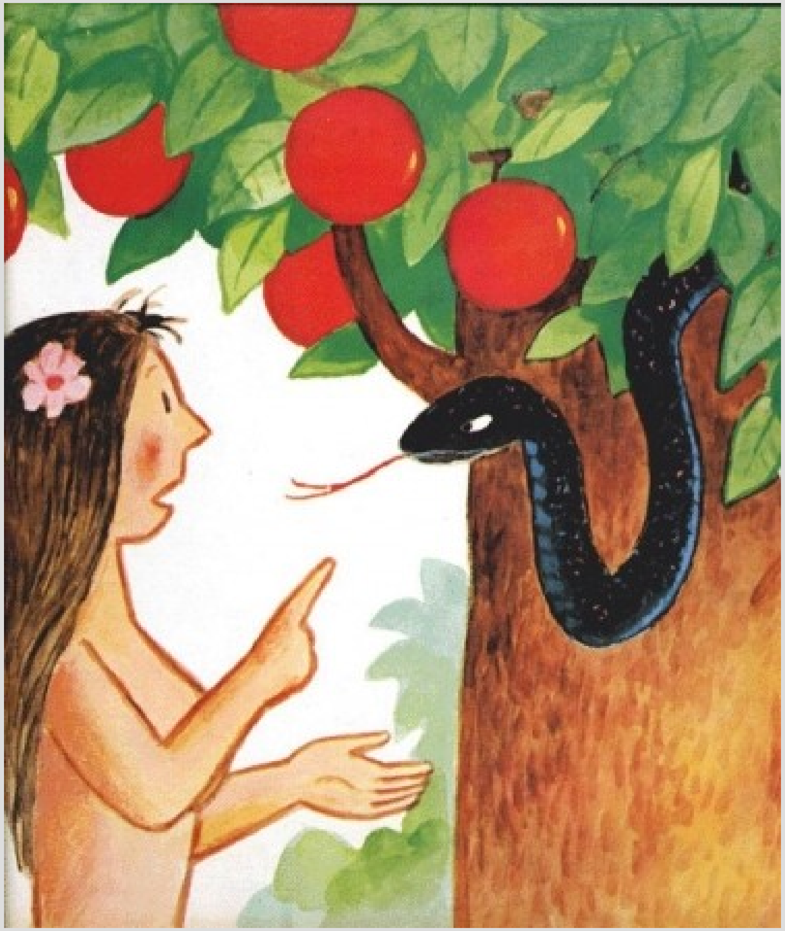


- Abstract class to model a source on the sky.
- Abstract method to create trees for visibility predictions.
- Subclasses implement specific kinds of source models.
- Hides specifics of source models.

TDL: Summary

- TDL: molding Python into a domain language.
- Provides a simple syntax for declaring trees.
- Leverages all the power of Python
  - control structures & functions
  - modules, packages → code reuse
  - object-orientedness
- Keeps the “messy stuff” (model definition and policy) in the scripting domain.
- Keeps the “heavy lifting” (computations) in the C++ domain.
- Quantum jump in our ability to build MeqTrees.

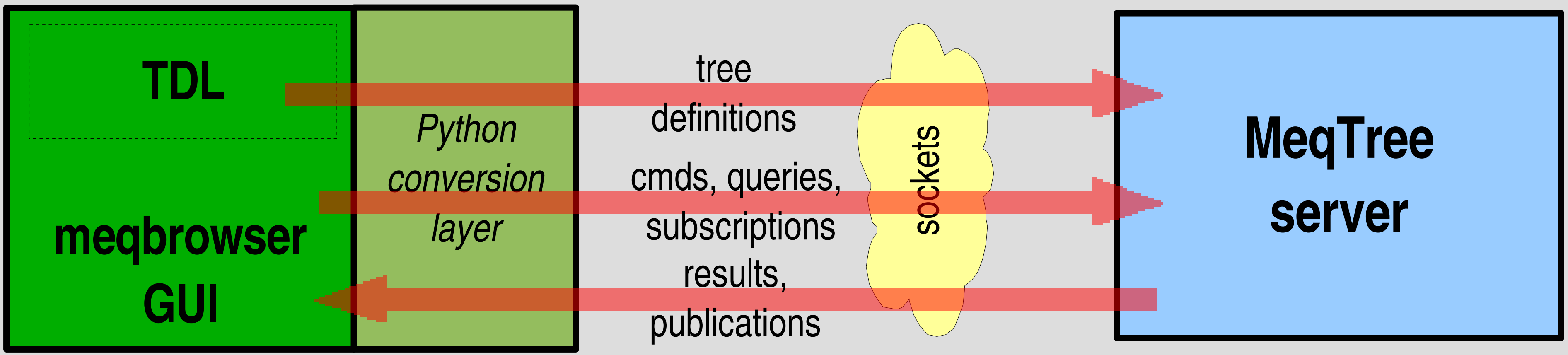
Fun (and Dangerous) Things To Do With Trees



This image is approved by Mr. Donald Rumsfeld

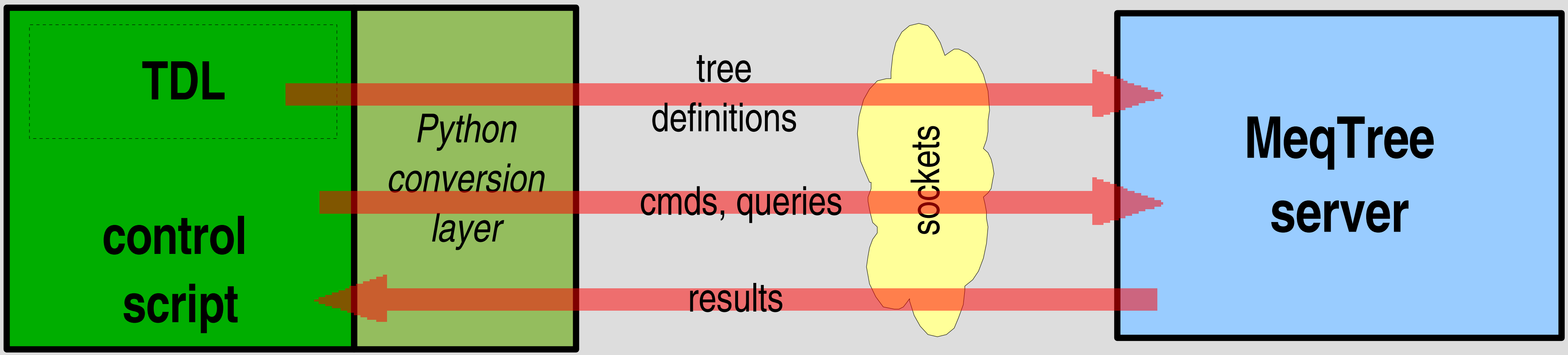
- Interfacing to datasets
- Specifying run-time parameters
- Controlling execution
- Examining status
- Visualizing results
- Debugging a tree
- Batch operation, testing

Embracing The Snake



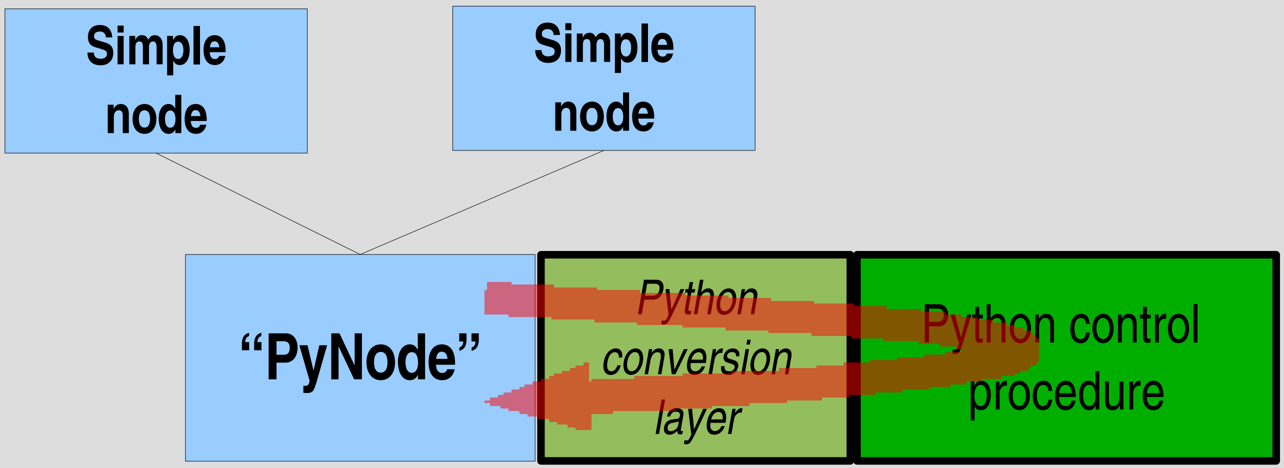
- Data transparency
  - all internal kernel structures have a Python representation, can be packed into a message
- Examine everything, keep track of anything
  - subscribe to node status -- so no overhead if not subscribed
- Python GUI (PyQt3 + PyQwt + ?)

Non-interactive Operation



- GUI-less operations also supported
- Python control script runs TDL, talks to server
- Useful for batch operations and automated testing

Embedding The Snake



- Most nodes have simple but CPU-intensive jobs
- A few specific nodes engage in "messy policy"
- Example: Solver
  - various least-squares fit strategies possible, and they can be very application specific
- Solution: embed Python in the MeqTree kernel, have "messy nodes" call a user-supplied control procedure to support complex behaviour.

THE END?