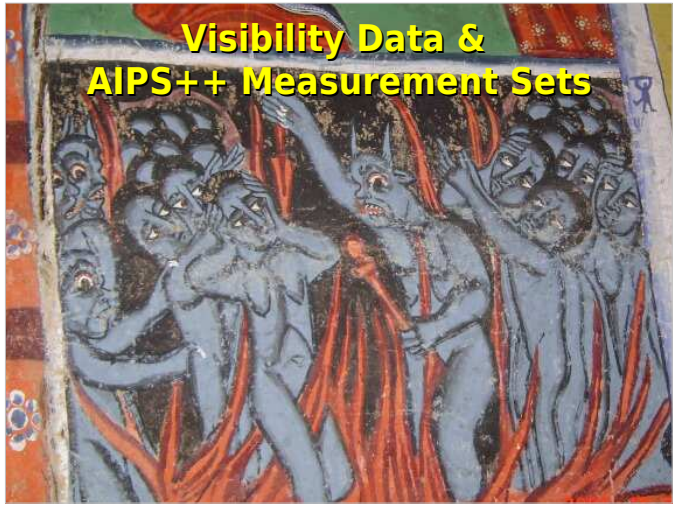

AIPS++...

2. AIPS++...
   - [5] heard of it
   - [2] tried to run it once
   - [9] succeeded in running it once
   - [5] have used it in anger
   - [0] invented it

- AIPS++ is making great progress: at the previous workshop we had “tried” ≈ “succeeded”
  - (and one inventor that owned up to it all)

On a Related Note...

2a. Reduction package of choice:
- [7] Classic AIPS
- [3] AIPS++
- [2] Miriad
- [1] MeqTrees!!!
- [1] NEWSTAR
- [1] MabCal

Working With Visibility Data

- MeqTrees interface with AIPS++ Measurement Sets
  - other formats can be supported as necessary
- An “empty” MS has to be pre-fabricated using external tools:
  - you can use the AIPS++ “simulator”, see `Workshop2007/demo_sim.g` (and ask Tony)
  - there's also a “makems” tool floating around (ask Ronald/Marcel/Joris)

Meet Our Guinea Pig Skeleton

- I have prepared **Workshop2007/demo.MS**; this will serve most of our whims this week.
- There is a pristine backup copy available, so if you screw up, restore it with:

```
$ cd ~/Workshop2007
$ rm -fr demo.MS
$ cp -a (/net/birch)/data/oms/Workshop2007/demo.MS .
```

(**/apps/Timba/data/oms/Workshop2007**, if on jop01)

VLA In Space (About demo.MS)

- This contains 27 antennas in VLA-C configuration...
  - ...but blown up by a factor of 10
- So the max baseline is ~30km
- 8 hours observation, 5 minute sampling, 96 timeslots
- 32 frequency channels of 16MHz each, from 800MHz to 1.31GHz
- Four polarizations: XX XY YX YY
- One pointing

A Simple MS Tree

- Load `Intro2/demo1-sink.py`
- Under “TDL Exec”, select Tile size: 10
- Load up the “MS Grids” bookmark
- Run “test forest”
- ...note the “history” slider in the visualizer

Why “Skeleton”?

- An MS provides a time/frequency grid (e.g., for use in simulations)
  - thus, “skeleton”: we ignore the data in the MS (and write our own)
- **Sink** nodes turn this grid into a request and send it up the tree.
  - one Sink per interferometer
- When a result comes back, this can be written out to a visibility column in the MS.
- MSs are processed in chunks of time called “tiles”.

The VisDataMux

- A **VisDataMux** node was created for us automatically.
- The VDM is responsible for interfacing with the MS, reading data, and activating its child Sinks as appropriate.
- To start the process, we give a specially-formed request to the VDM, containing **input** and **output** records telling it what and how to read (or write).

We Can Read, Too!

- Load `Intro2/demo2-spigot.py`
- Under “TDL Exec”, select Tile size: 10
- Load up the “Spigots” bookmark, and the “Inspector” bookmark
- Run “test forest”
- ...note the “history” slider in the visualizer

Sinks And Spigots

- A **Spigot** node reads the visibility data from an MS, and returns it as a visibility matrix
- Check visibilities using the history slider.
  - You're looking at XX data, use the “Change selected Vells” option to look at the other correlations
- You can probably guess what kind of observation **demo.MS** contains...

Inspector (Collections) Plot

- The last script introduced a “Collections” plotter (the ns.inspector node)
- A Meq.Composer node collects results from all its children into a single huge Result, which is plotted as a function of time.
- This plotter expects one data point per timeslot, so we use a Meq.Mean() node to take the mean in frequency.
- The inspector is attached as a special child to the VisDataMux node, labelled “post”. This makes it execute *after* (i.e., post) all the Sinks have fired. The result is published to the viewer (if active), then discarded.

Matrices And Tensors

- Visibility data comes out as a 2x2 matrix $\begin{pmatrix} XX & XY \\ YX & YY \end{pmatrix}$
- In MeqTrees, this is represented by a **Result** with 4 **VellSets**, and a **dims=**[2,2] field: $\begin{pmatrix} V_0 & V_1 \\ V_2 & V_3 \end{pmatrix}$
  - no dims implies a 4-vector
  - and remember that each element can be its own function of frequency/time/etc.
- This can be generalized to tensors of arbitrary rank
  - e.g., the “inspector” node collects its children into a 351x2x2 tensor

On MS Columns

- An AIPS++ MS has three standard “columns” for visibility data: DATA, MODEL_DATA, CORRECTED_DATA.
- MeqTrees can “attach” to any column, or even create new columns.
- Tools like the AIPS++ imager assign specific meanings to these columns though, and do not support other names...
- Speaking of the imager, run this script:

```
$ glish -l make_image.g DATA ms=demo.MS
```


Let's Modify Some Data

- Load `Intro3/demo3-mod-vis.py`
- Here we apply a gain term:

$$g_{pq}=(1+.1p)e^{2\pi iq/3}$$

- Under “TDL Exec”, select Tile size: 30 (the “go faster” option)
- Load up the “Inspector” bookmark
- Run “test forest”
- Switch inspectors to display complex phases (via right-click)
- Make an image:

```
$ glish -l make_image.g MODEL_DATA ms=demo.MS
```

More Inspectors

- Here we have created two inspector nodes
- A Meq.ReqMux() node is used to feed a request to multiple children, we need it since we can only have one “post” child on a VisDataMux.

Exercise 1: Freq-Dependent Gains

- Start with Intro2/demo3-mod-vis.py
- Apply a frequency-dependent gain to the data:

$$g_{pq}=(1+.1p\frac{(\nu-\nu_0)}{\Delta\nu})e^{2\pi i\frac{q}{3}\frac{(\nu-\nu_0)}{\Delta\nu}},\quad \nu_0=8\cdot10^8,\Delta\nu=5\cdot10^8$$

tip: use Meq.Polar(x,y) to compose xe^{iy}

- Make a per-channel image using:

```
$ glish -l make_image.g MODEL_DATA ms=demo.MS
mode=channel
```