



NHSC/PACS Web Tutorials

Running PACS spectrometer pipelines

PACS-302

*Level 1 to Level 2 processing:
From Calibrated Frames to Rebinned and
Spec-projected (WCS) Spectral Line Cubes*

Original Tutorial by Philip Appleton

Last updated Sept 2012 by Steve Lord

Introduction

This tutorial presents the main steps of the standard pipeline, starting from the calibrated Frames (Level 1) and PacsCube (also Level 1) and describes the processing steps needed to create a set of Pacs “Rebinned Cubes “ and (if the observation is a raster map), to a SpecProject cube.

It also provides numerous break points to interactively check the intermediary results of the pipeline.

Documentation on the PACS Spectroscopy pipelines is here:
PACS Data Reduction Guide: Spectroscopy, e.g., Chapter 3
These documents can be access through the HIPE help pages.

Welcome to the Herschel Interactive Processing Environment Help System



The screenshot shows a grid of nine help topics. A callout bubble points to the 'Learn about PACS data' box, which contains links for 'Data reduction guide for photometry and spectroscopy' and 'Data known issues'.

 <p>New to HIPE? Click here for a quick introduction</p>	 <p>Learn about the Help System: Help on Help</p>	 <p>Learn how to get data from the Herschel Archive</p>
 <p>Learn about HIPE data</p> <ul style="list-style-type: none">Data reduction guidePipeline specificationData known issues	 <p>Learn about PACS data</p> <ul style="list-style-type: none">Data reduction guide for photometry and spectroscopyData known issues	 <p>Learn about SPIRE data</p> <ul style="list-style-type: none">Data reduction guidePipeline specificationData known issues
 <p>HIPE known issues</p>	 <p>Find out what's new</p>	 <p>See a list of all the available manuals</p>



Important new material is now online
at the PACS public Twiki:

<http://www.herschel.be/twiki/bin/view/Public/PacsDocumentation>

including the latest PACS Pipeline Refs Manual and the PACS
Data Reduction Guide: Spectroscopy

Also find out what is new for pipeline 9.1 here:

<http://herschel.esac.esa.int/twiki/bin/view/Public/HipeWhatsNew9x#Pipeline>



Pre-requisites:

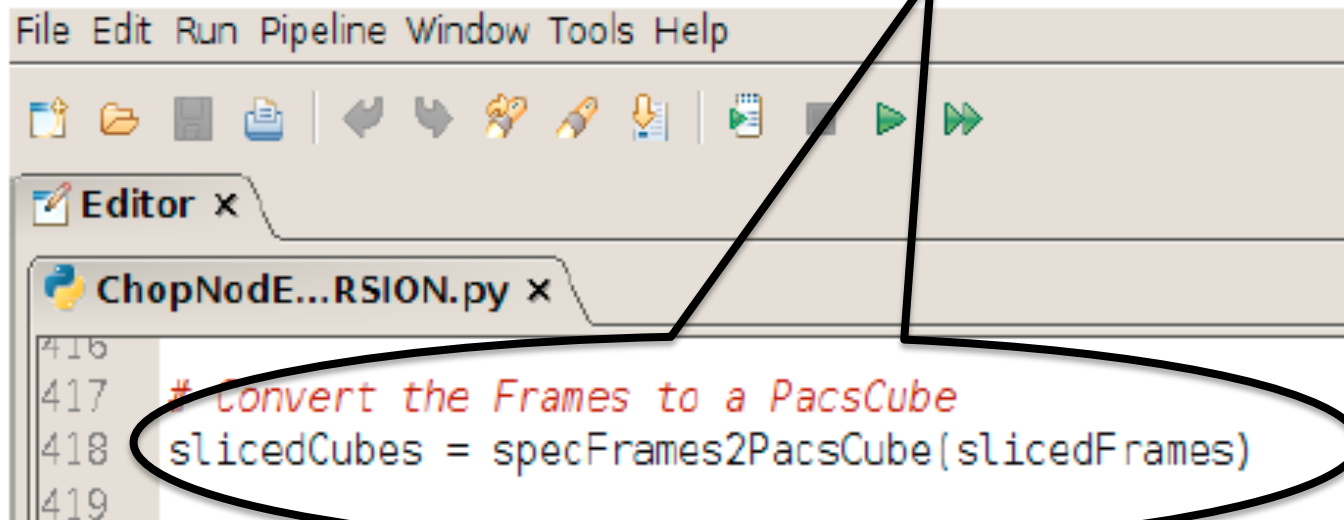
1. HIPE is running version 9.1 or later
2. You have completed the following tutorials:
 - ***PACS-101 to -104: How to use these tutorials, load data and scripts in your HIPE session.***
 - ***PACS-301 Pacs Spectroscopy Level 0 to 1 Processing***

At this point in the processing you have just completed the last step of tutorial PACS301. We finally performed the flat field step after having created a set of sliced Cubes.

In this example we work with an obsId = 1342186799
camera = blue

The cubes are made in the penultimate pipeline step...

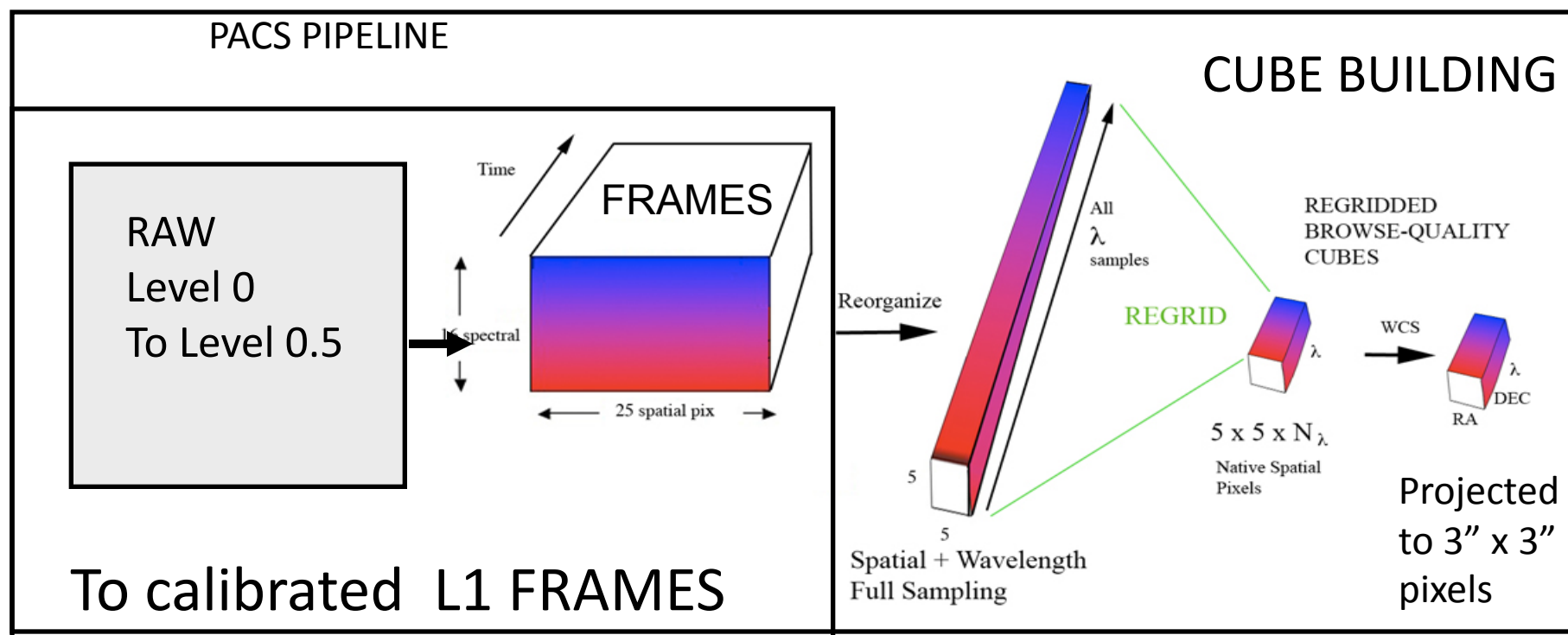
At this point, the frames are converted in cubes and we have reached level 1 !



```
File Edit Run Pipeline Window Tools Help
[Icons]
Editor x
ChopNode...RSION.py x
416
417 # Convert the Frames to a PacsCube
418 slicedCubes = specFrames2PacsCube(slicedFrames)
419
```

Lets Recap the structure of the data products so far and see where we are going

Level 0 -1 and Level 2 Products



CALIBRATED FRAME
PACSCUBE CUBE

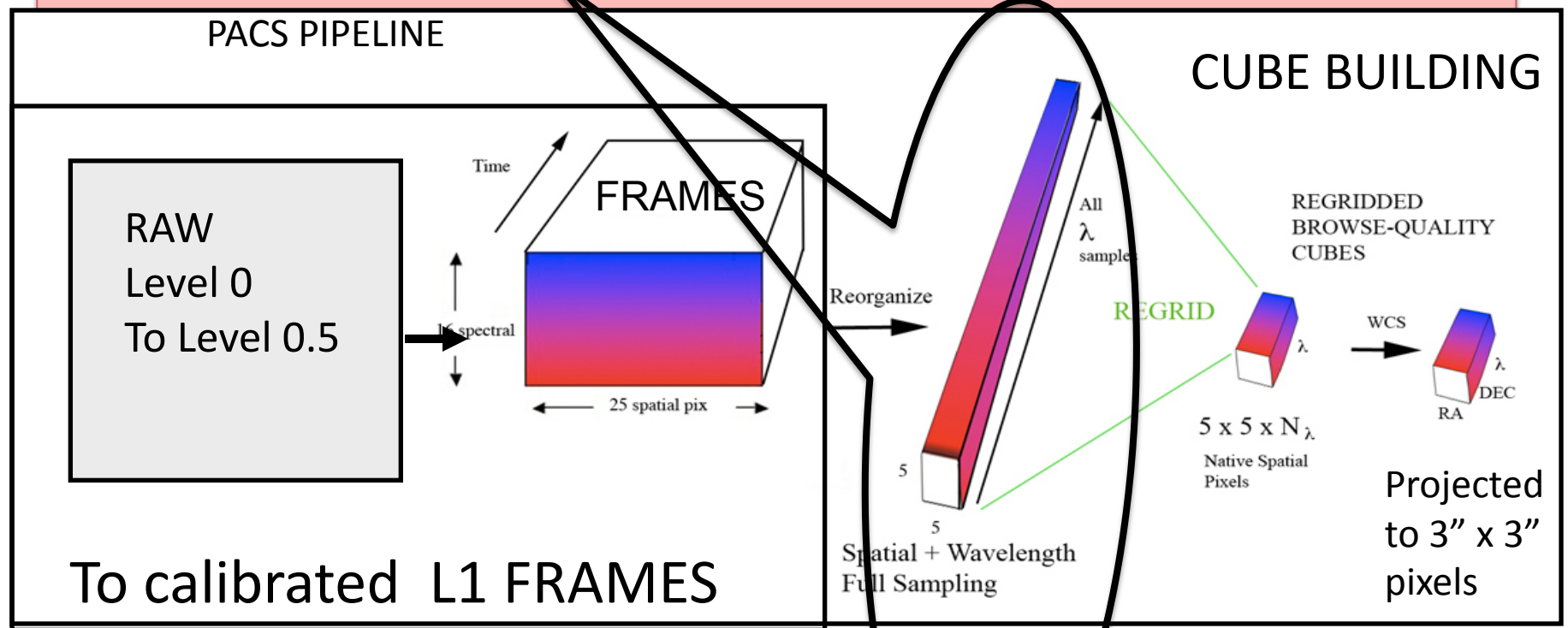
Level 1

REBINNED CUBE
PROJECTED CUBE

Level 2

Lets Recap the structure of the data products so far so that we can try to understand what we have done so far

Near the end of the Level 1 pipeline we created a set of Pacscubes. There are more than one in our example because this blue AOR contains 4 slices (NodA and Nod B for each of two different lines)



CALIBRATED
FRAME

PACSCUBE
CUBE

REBINNED
CUBE

PROJECTED
CUBE

Level 1

Level 2 PACS-302



Level 1 to 2: Overview



Step 1

Check number of Frame and Cube Slices from level 1

Step 2

Select specific line you want to reduce from those observed

Step 3

Inspect the central spaxel after deglitching

Step 4

Create a wavelength grid for binning

Step 5

Flag data with user-controlled sigma clipping

Step 6

Check "Outliers" mask on Spectrum

Step 7

Flag data with user-controlled sigma clipping

Step 8

Inspect the Rebinned Cube Spaxel by Spaxel

Step 9 and 10

Create and explore Projected Cubes in case of Rasters

loop over
N lines

**NOTE THAT THE PIPELINE WILL AUTOMATICALLY LOOP OVER ALL LINES
IF YOU LEAVE THE lineID blank in step 2**



Step 1

Check that you have converted your frames to PacsCubes (5 x 5 x N) product, and that you have the correct number of both

A pacscube is simply a reorganized Frame with dimensions which are $5 \times 5 \times N$ where the 5×5 refers to the IFU spaxels of PACS, and N is the number of time samples which translate into grating position and wavelength. In a sense you can think of the PACS cube as a spatial representation of sky (5×5) and the third dimension can be translated into wavelength—forming the basic dataset or “cloud of wavelength samples” as a function of position.

In our example we have now 4 slices of Frames and now, having executed the last step we have 4 slices of Cubes. These slicedCube and slicedFrame entities (objects) are simply collections of Frames and Cubes sliced by raster position (if relevant), Nod position (Nod A and B) and Line ID. In this case we have two lines and two Nod observations. Had we done more than one repetition of the Nod cycle we would have more than 2 separate nods. Since this was a simple pointed observation with one repetition, we have only 4 slices of both frames and now, as of the last step, 4 cubes



Summary of Level 1 Products



```
HIPE> slicedSummary(slicedFrames)
```

```
noSlices: 4
```

```
noCalSlices: 0
```

```
noScienceSlices: 4
```

slice#	isScience	nodPosition	nodCycle	rasterId	lineId	band	dimensions	wavelengths
0	true	["B"]	1	0 0	[2]	["B3A"]	[18,25,960]	63.093 - 63.379
1	true	["A"]	1	0 0	[2]	["B3A"]	[18,25,960]	63.093 - 63.379
2	true	["B"]	1	0 0	[3]	["B3A"]	[18,25,960]	57.213 - 57.548
3	true	["A"]	1	0 0	[3]	["B3A"]	[18,25,960]	57.213 - 57.548

Above are 4 Frames (18x25x960) representing 1+16+1 (=18) spectral pixels, 25 spatial pixels (the 5 x 5 array) and 960 time samples. Note the 16 spectral pixels plus 2 extra (a "open" channel and an "overscan") making 18 spectral pixels total

```
HIPE> slicedSummary(slicedCubes)
```

```
noSlices: 4
```

```
noCalSlices: 0
```

```
noScienceSlices: 4
```

slice#	isScience	nodPosition	nodCycle	rasterId	lineId	band	dimensions	wavelengths
0	true	["B"]	1	0 0	[2]	["B3A"]	[15360,5,5]	63.093 - 63.379
1	true	["A"]	1	0 0	[2]	["B3A"]	[15360,5,5]	63.093 - 63.379
2	true	["B"]	1	0 0	[3]	["B3A"]	[15360,5,5]	57.213 - 57.548
3	true	["A"]	1	0 0	[3]	["B3A"]	[15360,5,5]	57.213 - 57.548

After conversion to slicedCube we now have 4 Pacscubes, organized as time sample*16 (=15360), 5 x 5 in this case. Note that the open and overscan spectral pixels have been stripped off in the Pacscube format.



Step 2 LINE SELECTION

Determine the association between line observed and lineId by inspecting the slicingSummary.

Then select this line for processing.
At the end of this process you will return to Step 2 to process any further lines

Next, we have to select a particular line to process.

The easiest way is to filter on a line is to look at the output of the sliced summary from the previous slide: The line ID is associated with a particular line.

```
HIPE> slicedSummary(slicedCubes)
```

```
noSlices: 4
```

```
noCalSlices: 0
```

```
noScienceSlices: 4
```

slice#	isScience	nodPosition	nodCycle	rasterId	lineId	band	dimensions	wavelengths
0	true	["B"]	1	0 0	[2] ["B3A"]	[15360,5,5]	63.093 - 63.379	
1	true	["A"]	1	0 0	[2] ["B3A"]	[15360,5,5]	63.093 - 63.379	
2	true	["B"]	1	0 0	[3] ["B3A"]	[15360,5,5]	57.213 - 57.548	
3	true	["A"]	1	0 0	[3] ["B3A"]	[15360,5,5]	57.213 - 57.548	

```
HIPE>
```

Let's select lineId = 2 to make rebinned (and projected cubes of this line, if it is a raster observation—but we will do it anyway even though this observation does not contain a raster—just for an example)

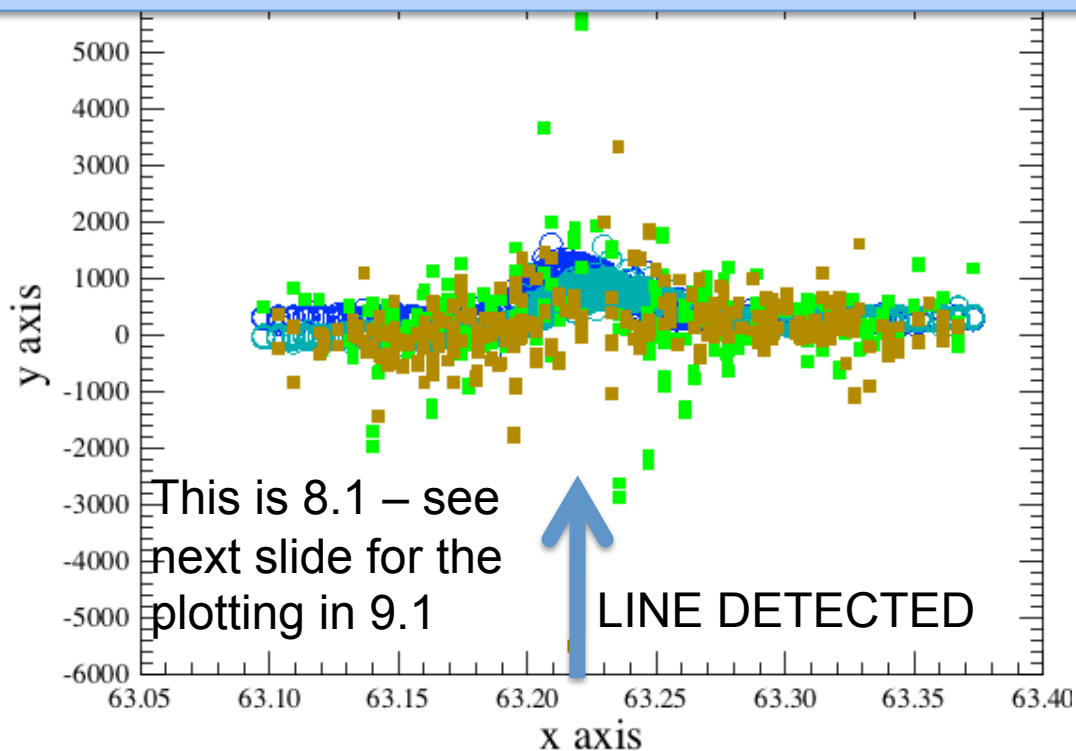


Step 3

Check the spectrum and quickly inspect what has been flagged by the deglitching algorithm

The next set of commands checks that you have only two slices and confirms the lineid = [2] as [OI]63 microns

PIPELINE PLOTS CENTRAL SPAXEL [2,2] only. You can re-run with different pixel selected



This is 8.1 – see next slide for the plotting in 9.1

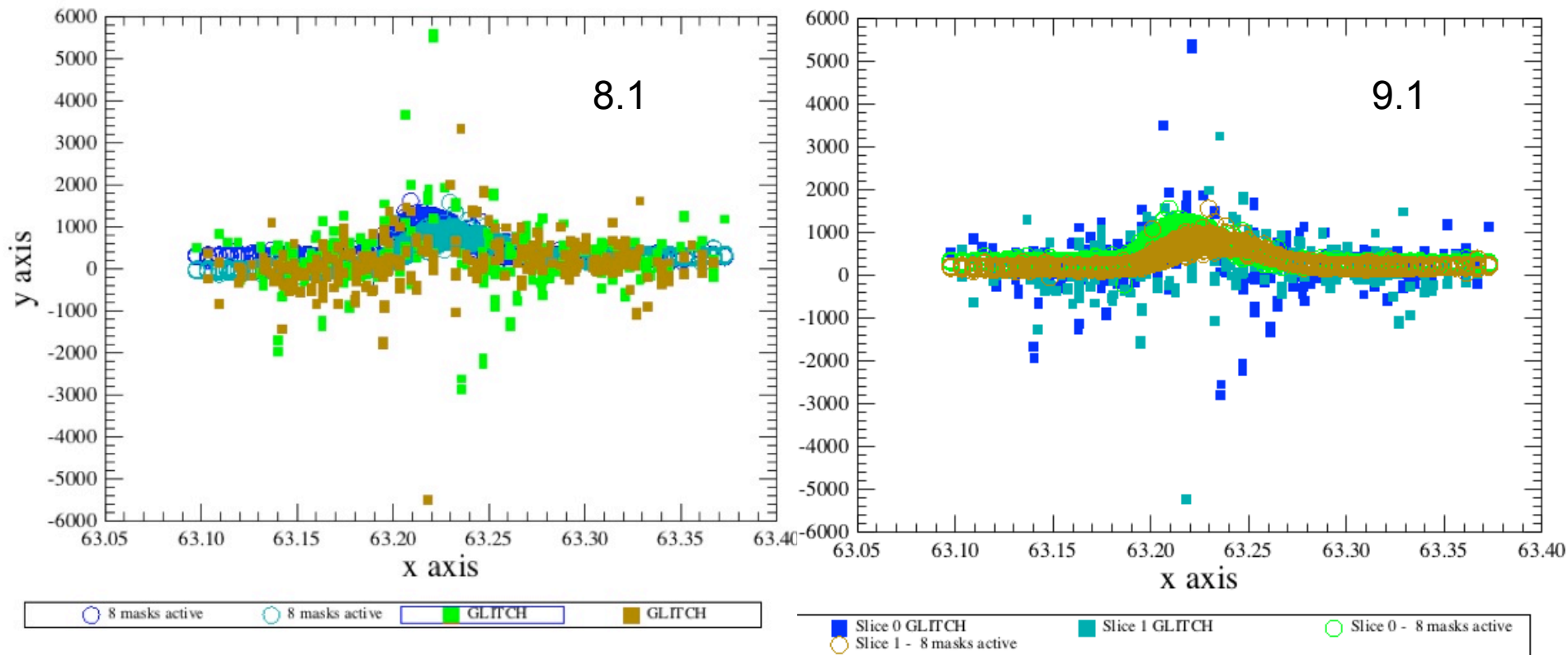
LINE DETECTED

The pipeline will plot a first-look at the spectrum including some (in this case many) samples which are flagged as glitches.

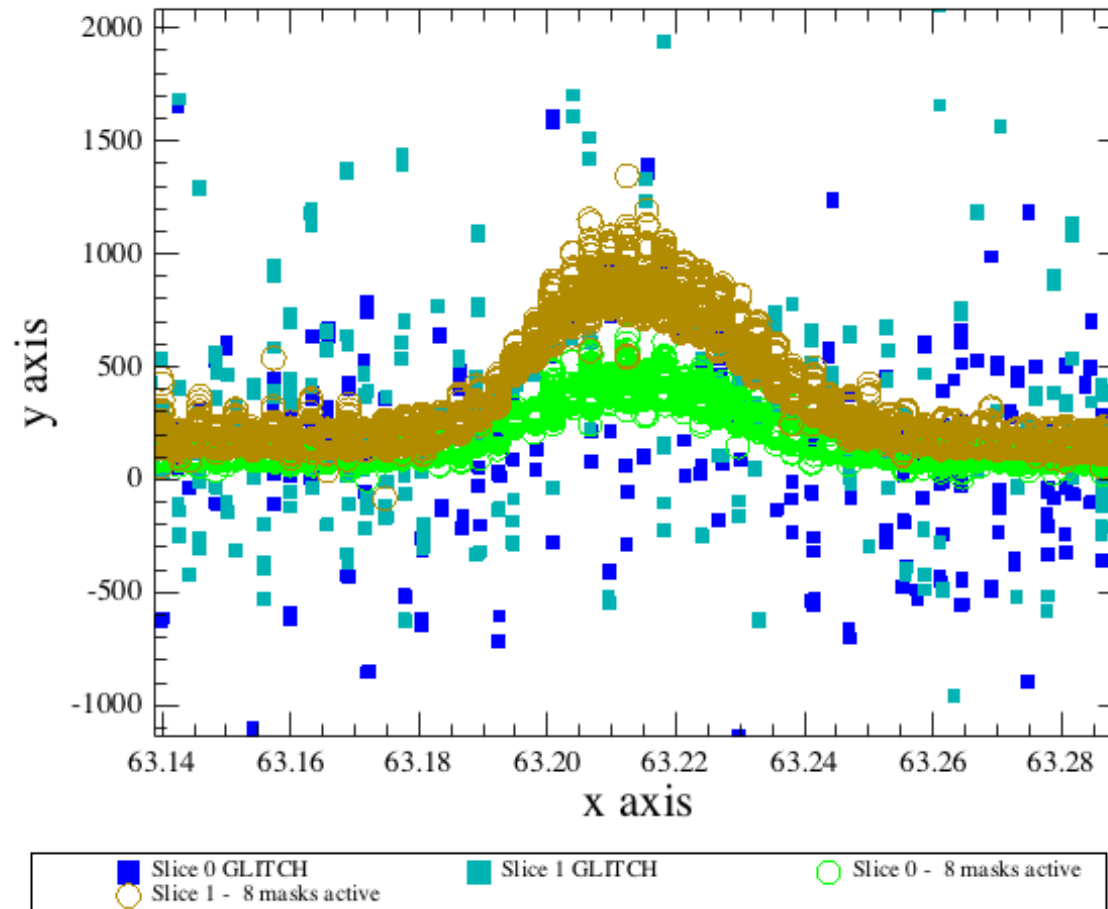
If you feel too many points are being flagged we can offer a workaround that will ignore the glitch mask and rely on sigma-clipping at a later point in the pipeline (we will explain later how to do this and you can try both ways) For the moment we will accept the glitch mask as correct.

IF YOUR LINE IS WEAK YOU MAY NOT SEE IT AT THIS STAGE until further clipping is performed

Plotting order of line and glitches changed in HIPE 9.1



Offset pixel....





Step 4

Create a wavelength grid that will be used to grid in “wavelength-space” the spectra. This step is necessary before constructing a re-binned cube. By default the grid will be Nyquist sampled but the user can made changes to the gridding, allowing for a finer grid or the addition of some smoothing



Now create a wavegrid which will be used to bin the spectra into bins in wavelength. The bin widths and distribution are governed by the parameters “oversample” and “upsample”. The default values of these are oversample = 2, upsample = 1. This provides a grid which oversamples a spectral resolution element by a factor of 2 (i.e. Nyquist sampling) and the upsample of 1 means that each wavelength bin is independent of the next and so-on.

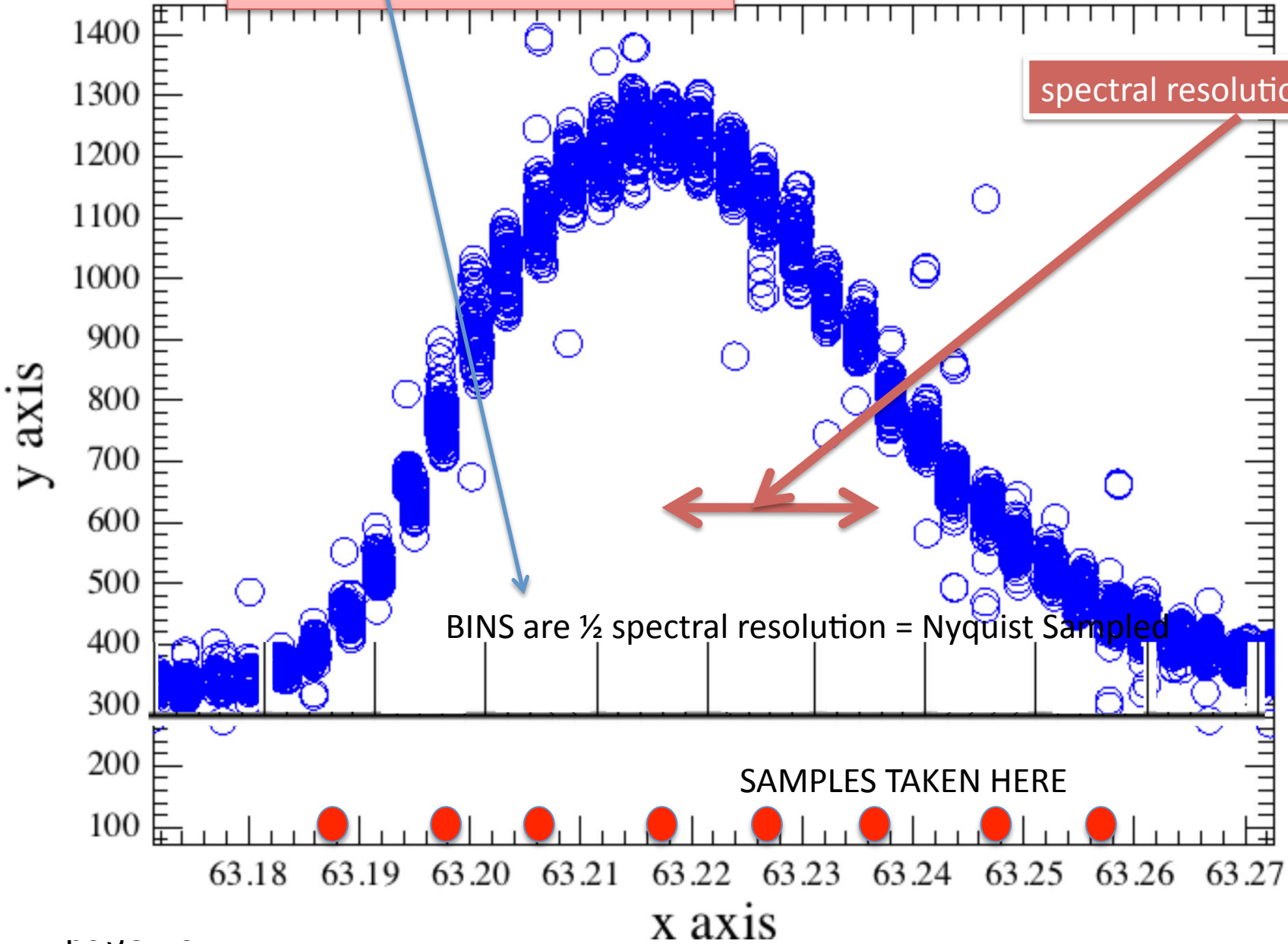
```
waveGrid=wavelengthGrid(sCubes.refs[0].product, oversample=2, upsample=1,  
                        calTree = calTree)
```



Definitions of Oversample and Upsample Parameters

Oversample = 2 Upsample = 1
bins = $\frac{1}{2}$ spectral resolution
at that wavelength

spectral resolution = $0.02\mu\text{m}$

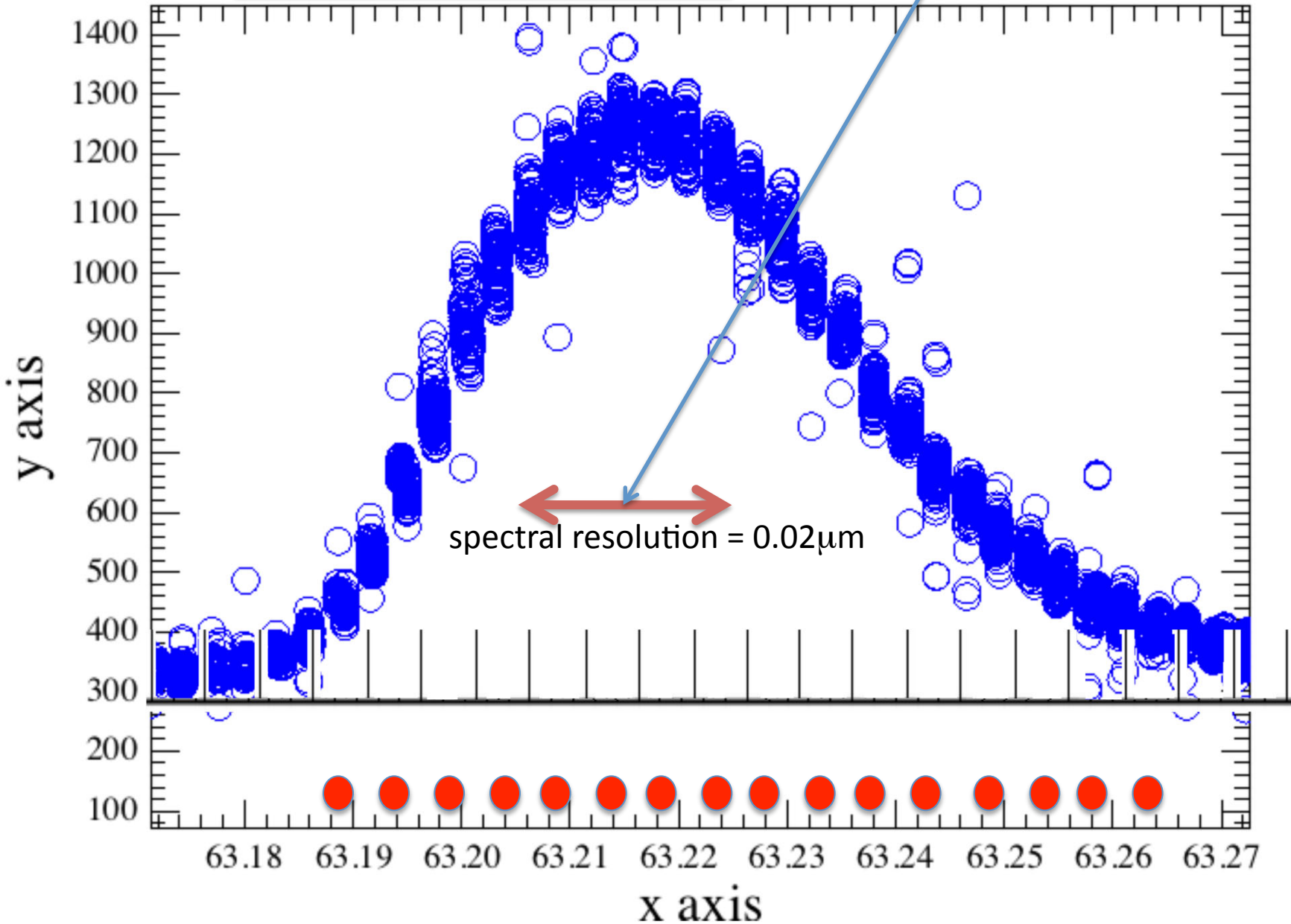


BINS are $\frac{1}{2}$ spectral resolution = Nyquist Sampled

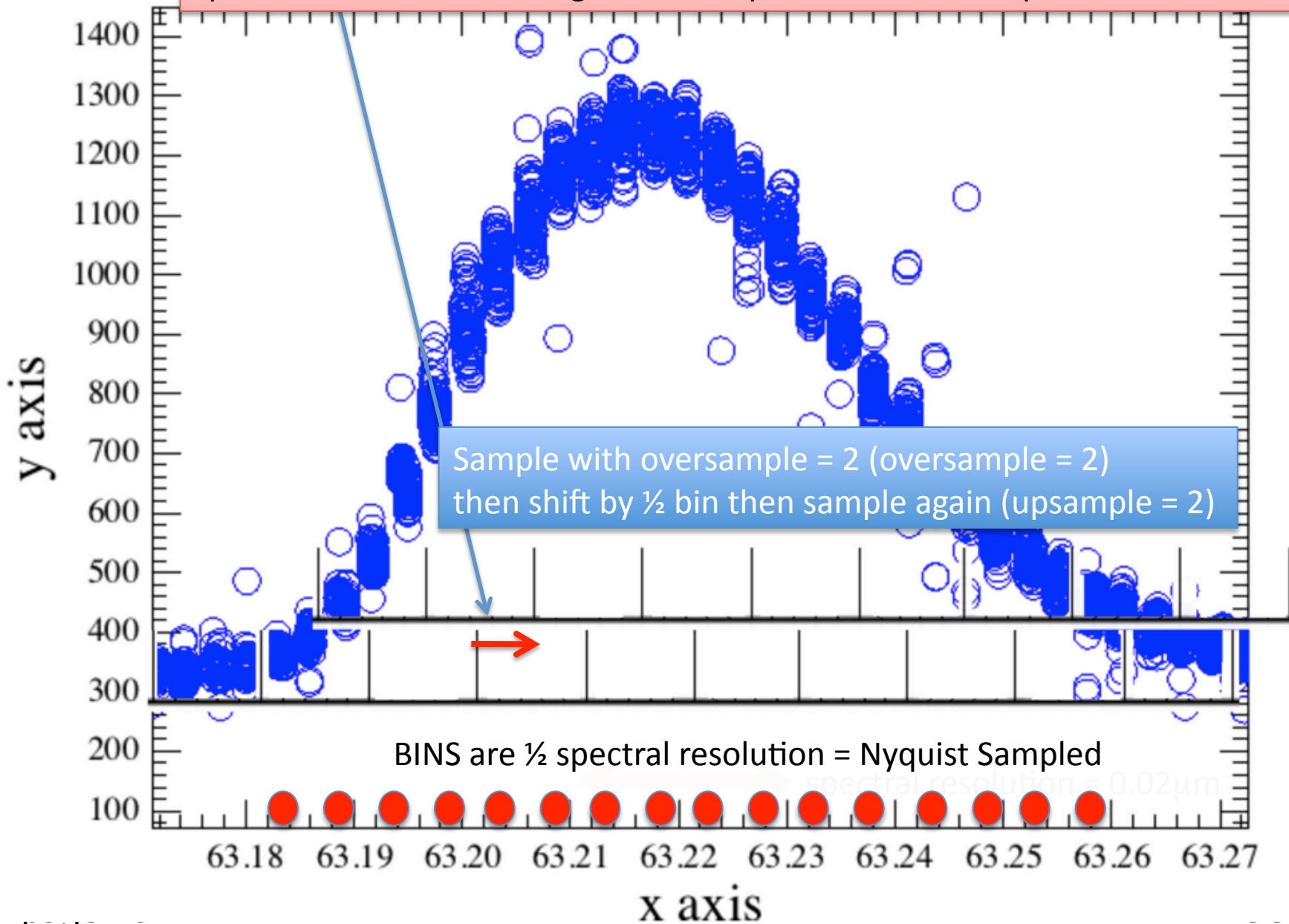
SAMPLES TAKEN HERE

Oversample = 4, Upsample = 1
bins = 1/4 spectral resolution
at that wavelength

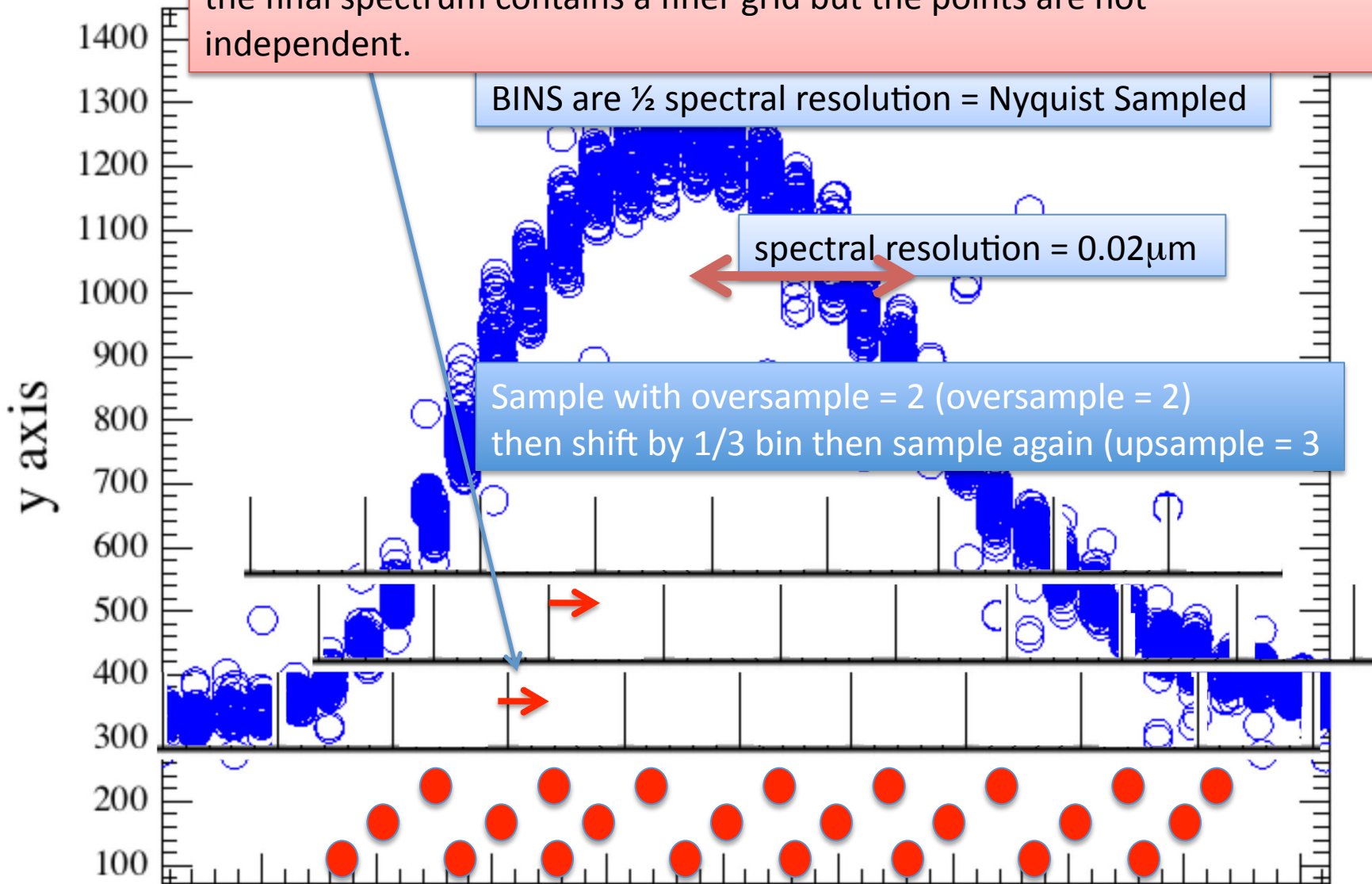
Oversample = 4 means
4 times narrower bins than the
spectral resolution (2 x better than Nyquist)



Oversample = 2 Upsample = 2 bins = $\frac{1}{2}$ spectral resolution at that wavelength, but bins are shifted by $\frac{1}{2}$ a bin and resampled, thus the final spectrum contains a finer grid but the points are not independent.



Oversample = 2, Upsample = 3 ; = $\frac{1}{2}$ spectral resolution at that wavelength, but bins are shifted by $\frac{1}{3}$ and $\frac{2}{3}$ a bin and resampled, thus the final spectrum contains a finer grid but the points are not independent.



SAMPLES ARE 3 X FINER THAN SPECTRAL RESOLUTION



Step 5

Before regridding we apply outlier rejection flagging to the spectra. This process is quite robust and works well for PACS spectra. If the user is suspicious that the Deglitcher is over-flagging, these flags can be turned off and the de-glitching would be done with the outlier rejection method

Next we active the masks and apply
a sigma-clipping routine to the spectra
(See PACS Data Reduction Guide: Spectroscopy, 3.3.8. for details)

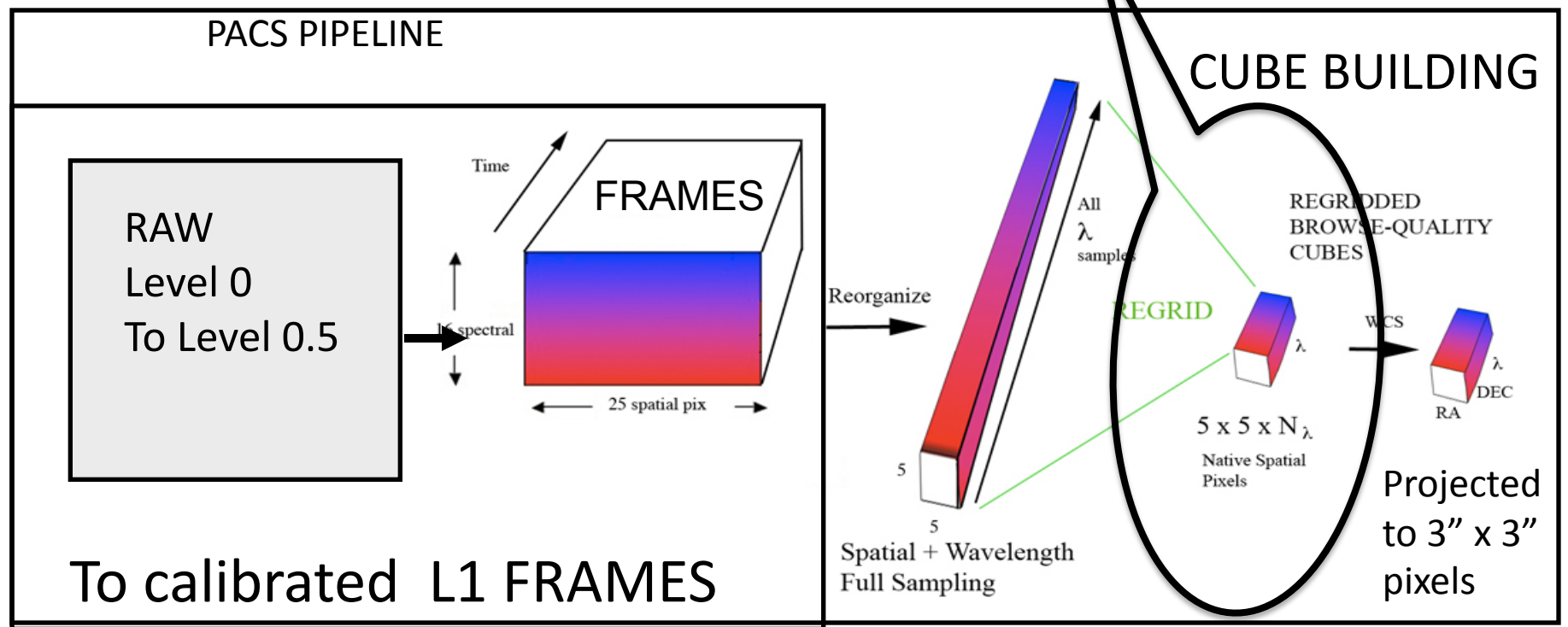
```
# Activate all masks
sCubes = activateMasks(sCubes, String1d(["GLITCH","UNCLEANHOP","SATURATION",
"GRATMOVE", "BADPIXELS"]), exclusive = True)

# Flag the remaining outliers, (sigma-clipping in wavelength domain)
sCubes = specFlagOutliers(sCubes, waveGrid, nSigma=5, nIter=1)
```

specFlagOutliers has the task of defining a new flag called "OUTLIERS" created using a simple sigma-clip algorithm on each bin. nIter=1 means that the algorithm has been applied once and then again iteratively rejecting points > nSigma. The user can experiment with this. The only outcome of running this routine is to create a mask which is applied when the actual rebinning of the spectra onto the waveGrid takes place. The mask is called "OUTLIERS" and will be activated in the next step.

NOTE ON EXCLUSION OF GLITCH MASK: If you choose to exclude the glitch detection from your results, remove the "GLITCH" string from the activateMasks command above. This will ignore the glitch mask. We have had good results by doing this.

Now we will create our first set of rebinned cubes—
 one for each Pacscube (in our case we have two—
 one for Nod A and one for Nod B)



CALIBRATED
FRAME

PACSCUBE
CUBE

REBINNED
CUBE

PROJECTED
CUBE

Level 1

Level 2

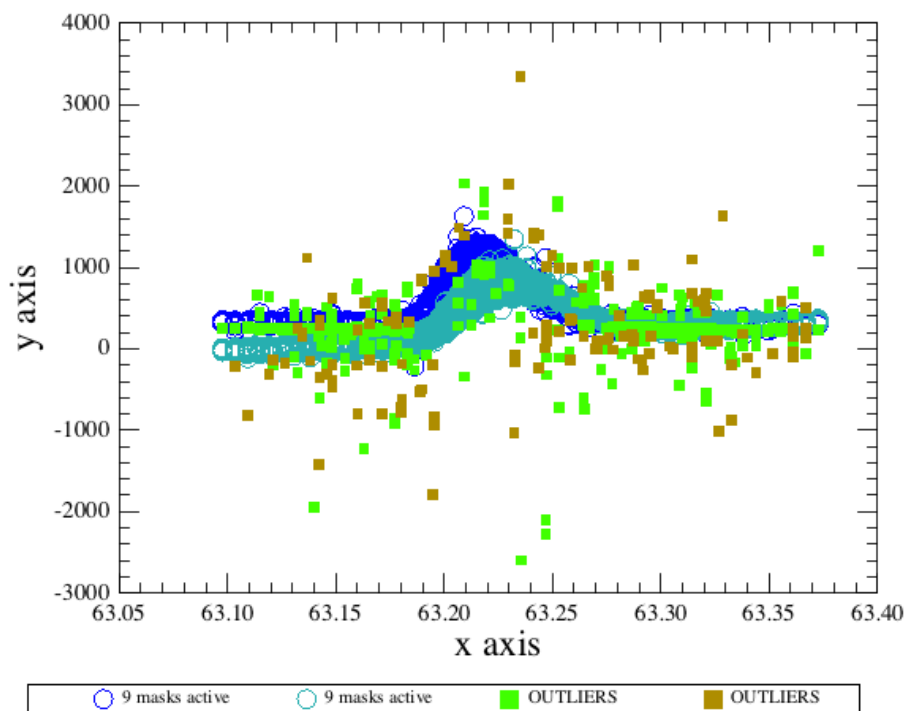


Step 6

Inspect the spectra again this time looking at the “OUTLIERS” flag.

Check that it appears to be flagging sensible points.

Now we activate the masks including the new “OUTLIERS” mask and create a Rebinned Cube



This plot shows the points for pixel [2,2] that have been flagged by the specFlagOutliers task. It can be seen that it does a good job of flagging outlier points.



Step 7

Create the Rebinned Cube

of dimensions 5 x 5 x rebinned wavelength
elements

Now Create the Rebinned Cube

We activate the OUTLIER mask for the first time before rebinning

```
# Activate all masks
sCubes = activateMasks(sCubes, String1d(["GLITCH", "UNCLEANCHOP", "SATURATION",
"GRATMOVE", "OUTLIERS", "BADPIXELS"]), exclusive = True)

# Rebin all selected cubes on the same wavelength (mandatory for specAddNod)
slicedRebinnedCubes = specWaveRebin(sCubes, waveGrid)
```

This step produces a set of rebinned cubes (one slice per cube)

NOTE ON EXCLUSION OF GLITCH MASK: If you choose to exclude the glitch detection from your results, remove the "GLITCH" string from the activateMasks command above. this will ignore the glitch mask. We have had good results by doing this. This is the second and only time you need to worry about ignoring the glitch mask. If you performed this step and the one before specFlagOutliers then you will have a rebinned cube which excludes the glitch detection and relies on the sigmaClipping for removal of outliers. In general we have found sigmaClipping to produce the better results.



Step 7

Average the Nods



Now we average together the the two Nod positions. For an observation with a set of raster positions there will still be more than one final cube (store as slices)—one for each raster position

```
# Average the nod-A & nod-B rebinned cubes.  
# All cubes at the same raster position are averaged.  
# This is the final science-grade product currently possible, for all observations except  
# the spatially oversampled rasters  
slicedRebinnedCubes = specAddNodCubes(slicedRebinnedCubes)
```



Step 8

Inspect the rebinned spectra—spaxel by spaxel using various tools

THIS IS THE END OF LEVEL 1 for Pointed Observations

if verbose:

```
slicedSummary(slicedRebinnedCubes)
```

```
#5x5 plot of one of the rebinned cubes
```

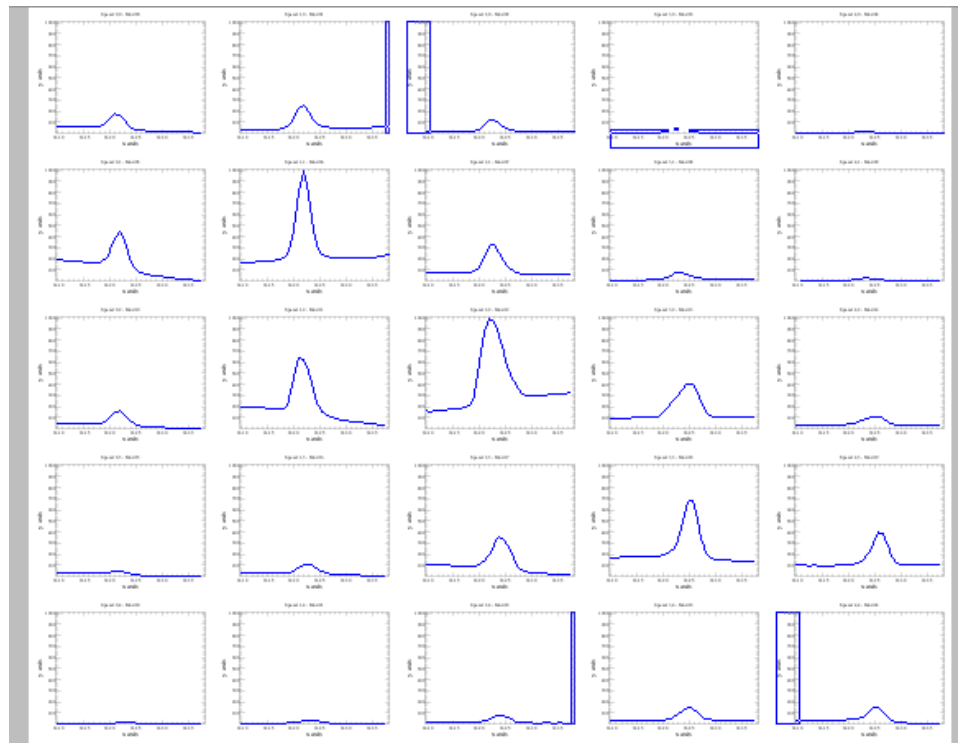
```
(one line/range at one raster position)
```

```
slice = 0
```

```
p10 = plotCube5x5
```

```
(slicedRebinnedCubes.refs[slice].product)
```

Plot only the first slice—in this case we have only one for the [OI] line since we have averaged this single pointed observation. If you have a raster you can look in turn at each one if you wish by changing this.



You can also extract a spectrum (in this case the central one [2,2] and create a Spectrum1dproduct

```
# Optional: extract the central spaxel of one slice into a simple spectrum
# You can do this for any spaxelX and Y and any slice. It creates a
# Spectrum1d class of product, on which various viewers will work
slice = 0
spaxelX, spaxelY = 2,2
centralSpectrum = extractSpaxelSpectru (slicedRebinnedCubes,
slice=slice, spaxelX=spaxelX, spaxelY=spaxelY)
```

On the next page we show how this can be studied in the variable list so that you can write to FITS or open in spectrum explorer.

You have reached LEVEL 2 for a Single Pointed Observation

In the variable list right-mouse click on CentralSpectrum and either "open with" the spectrum in Spectrum Explorer or send to a FITS file



Step 9

If you have a raster observation gather all the rebinned cubes together and project them onto the sky to make a final projected cube.



Raster versus Pointed Observations

If your observations contain rasters you may want to consider combining them into a specProject Cube which re-grids these data spatially onto a WCS sub-grid (the default pixel size is 3 x 3 arcsecs). Unless you modify the code, specProject will not create a cube from a simple pointed observation like the one in the example. Making a specProject map from a single pointing is not recommended because the PACS IFU footprints are distorted and can cause flux conservation problems in the map. Only in the case of a fully sampled raster map can one expect to recover proper fluxes.

If its a mapping observation then
all the differently pointed rebinned cubes
will be gathered together and projected onto the
sky with specProject: The last step to Level 2!

```
projectedCube = specProject(slicedRebinnedCubes)
# display the projected cube in the cube analysis toolbox
if verbose: openVariable("projectedCube", "CubeAnalysisToolBox")
```

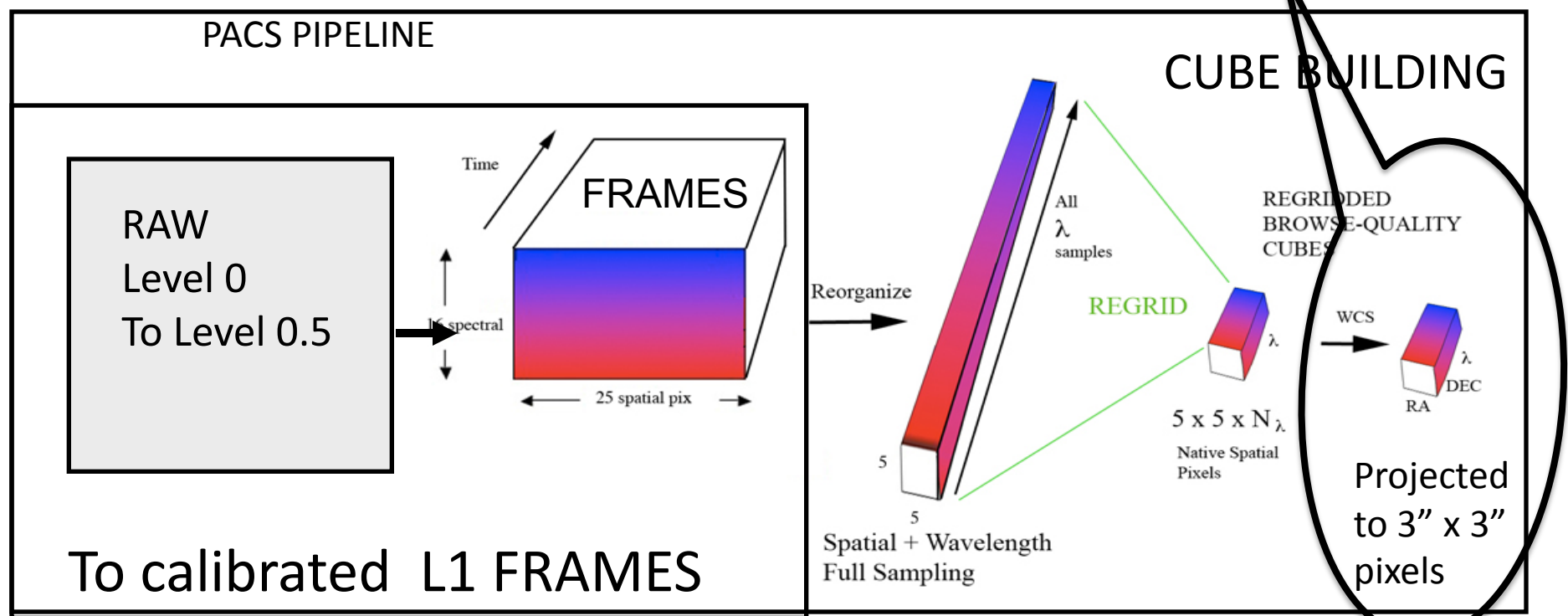
The present script makes a projectedCube and even opens up a viewer—although this and other viewers are available via right-mouse clicking on the variable “projectedCube” as well as FITS writing capability as before.



Step 10

Explore your final WCS projected cube with your favorite viewer or export to FITS.

The projected cube is only recommended for raster observation where you have many rebinned cubes and these are combined into a final projected cube



CALIBRATED
FRAME

PACSCUBE
CUBE

REBINNED
CUBE

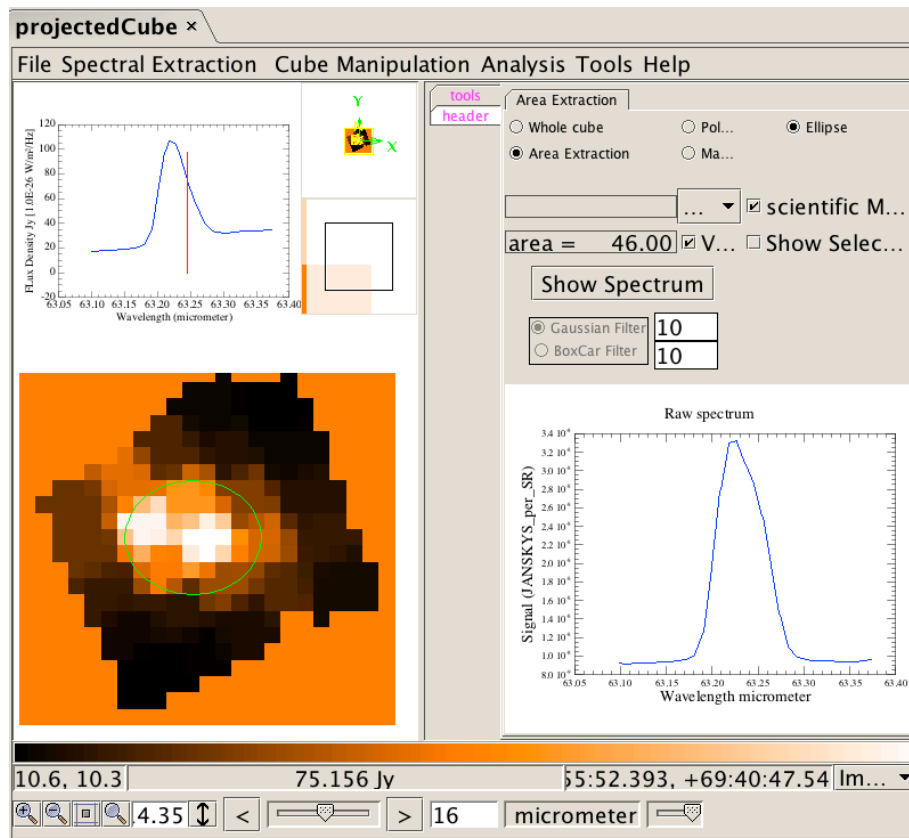
PROJECTED
CUBE

Level 1

Level 2

YOU HAVE REACHED LEVEL 2

Congratulations—now explore your cubes with the various tools offered!



Displaying a spectral cube in the cubeAnalysisToolbox, which allows for area extractions, baseline fitting and the formation of integrated line maps etc.

**IF YOU HAVE MORE THAN ONE
LINE—LOOP BACK TO STEP 2 AND PROCESS FOR
THE NEXT LINE IN THE LINE ID LIST**

Note that had you left the lineId blank, the pipeline will process all the lines observed. However, we think that for interactive analysis it is better for the user to select the ones they want themselves (at least the first time through) so that you can monitor the behavior of each one separately.

**CONGRATULATIONS YOU HAVE NOW
REACHED LEVEL 2!**