NHSC/PACS Web Tutorials

Running PACS spectrometer pipelines

PACS-302

Level 1 to Level 2 processing:

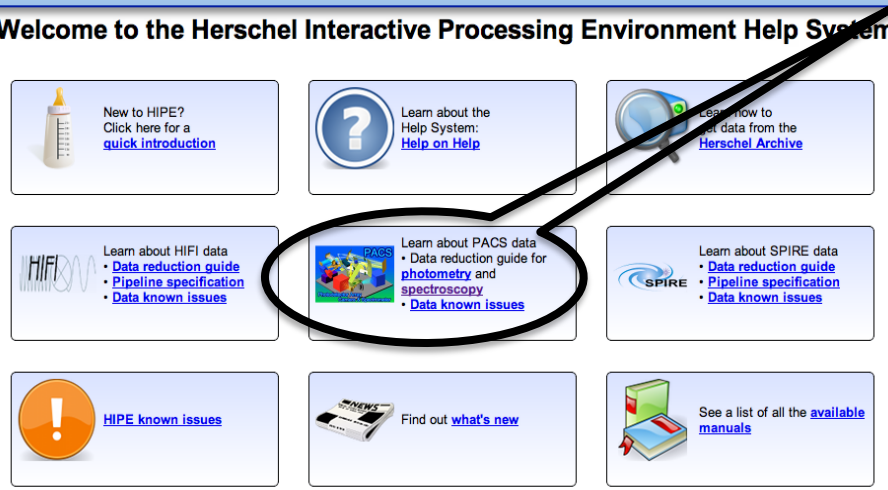
From Calibrated Frames to Rebinned and Spec-projected (WCS) Spectral Line Cubes

Original Tutorial by Philip Appleton
Updated Sept 2012 by Steve Lord
Updated Feb 2013 by Steve Lord
Updated Oct 2013 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).
It describes the processing steps needed to create a set of Pacs "Rebinned Cubes " and (especially if the observation is a raster) a Projected Cube.
Numerous break points are shown, where one can interactively examine the intermediate results of the pipeline.

The PACS Spectroscopy pipelines are documented here:
PACS Data Reduction Guide: Spectroscop, e.g., Chapter 3
These documents can be access through the HIPE help pages.



http://nhsc.ipac.caltech.edu/helpdesk/index.php

Important additional material for each release is found online
at the PACS public Twiki:
http://www.herschel.be/twiki/bin/view/Public/PacsDocumentation
http://herschel.esac.esa.int/twiki/bin/view/Public/PacsCalibrationWeb
including the latest PACS Pipeline Refs Manual and the
PACS Data Reduction Guide: Spectroscopy
Also, find out what is new for pipeline 11 here:
http://herschel.esac.esa.int/twiki/bin/view/Public/HipeWhatsNew11x#Pipeline

Tutorial Pre-requisites:

1. HIPE is running version 11.0 (or user build 11.0.2038) 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 we had just completed the last step of tutorial PACS 301. We last performed the flat field step after having created a set of sliced Cubes (step 3 below). We also did some cleanup (steps 4 & 5 below).

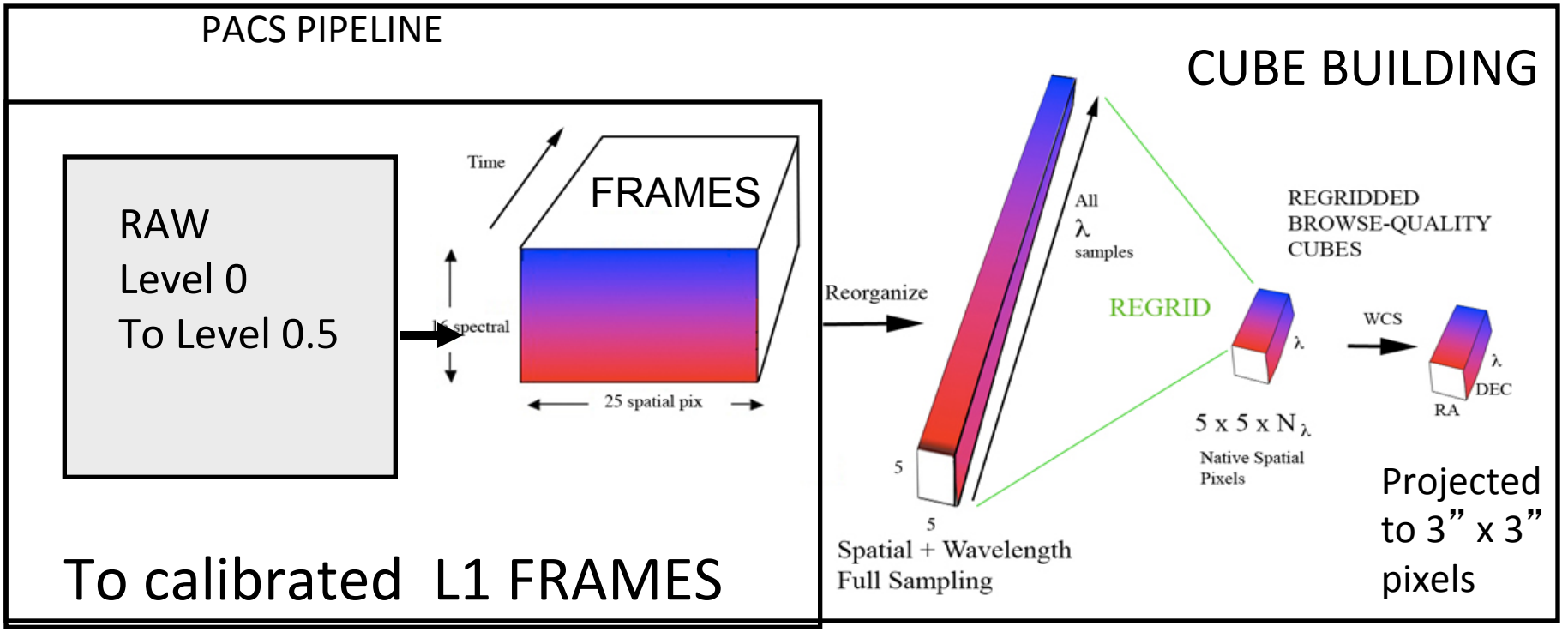
In this tutorial we continue working with obsId = 1342186799
camera = blue

Our last steps which brought us to Level 1:

```
517
518  # 3. Actual spectral flatfielding
519  # slopeInContinuum is a boolean. Set it to true for lines existing on a continuum with a significant slope
520  # scaling is a boolean. If true (highly recommended), a multiplicative correction is applied. If false,
521  # maxScaling sets a maximum to the scaling factors. At fluxes close to 0 (e.g. in outer spaxels), maxScaling
522  # to protect from abnormaly large scaling factors.
523  # Despite maxScaling, a few new outliers might arise from the flatfielding, these should be caught in the
524  # For more information, print specFlatFieldLine.__doc__ or consult the help.
525  slopeInContinuum = 1
526
527  slicedCubes = specFlatFieldLine(slicedCubesMask, scaling=1, copy=1, maxrange=[50.,230.], slopeInContinuum
528
529  # 4. Rename mask OUTLIERS to OUTLIERS_B4FF (specFlagOutliers would refuse to overwrite OUTLIERS) & deactivate
530  slicedCubes.renameMask("OUTLIERS", "OUTLIERS_B4FF")
531  slicedCubes = deactivateMasks(slicedCubes, String1d(["INLINE", "OUTLIERS_B4FF"]))
532  if verbose: maskSummary(slicedCubes, slice=0)
533
534  # 5. Remove intermediate results
535  del waveGrid, slicedRebinnedCubes, slicedCubesMask
536
537  # OPTIONAL. Compare the "cloud" of measurements for one spaxel before and after spectral flatfielding
538  # B. After FF
539  if verbose:
540          slice = 0
541          x,y = 2,2
542          offset = 0.
543          pffed = plotPixel(slicedCubes.get(slice), x=x,y=y,masks=slicedCubes.get(slice).getActiveMaskName
544
545  # --- End of Spectral Flat Fielding
```

Let's recap the structure of the data products so far, and see where we are going:

Level 0 -1, and Level 2 Products



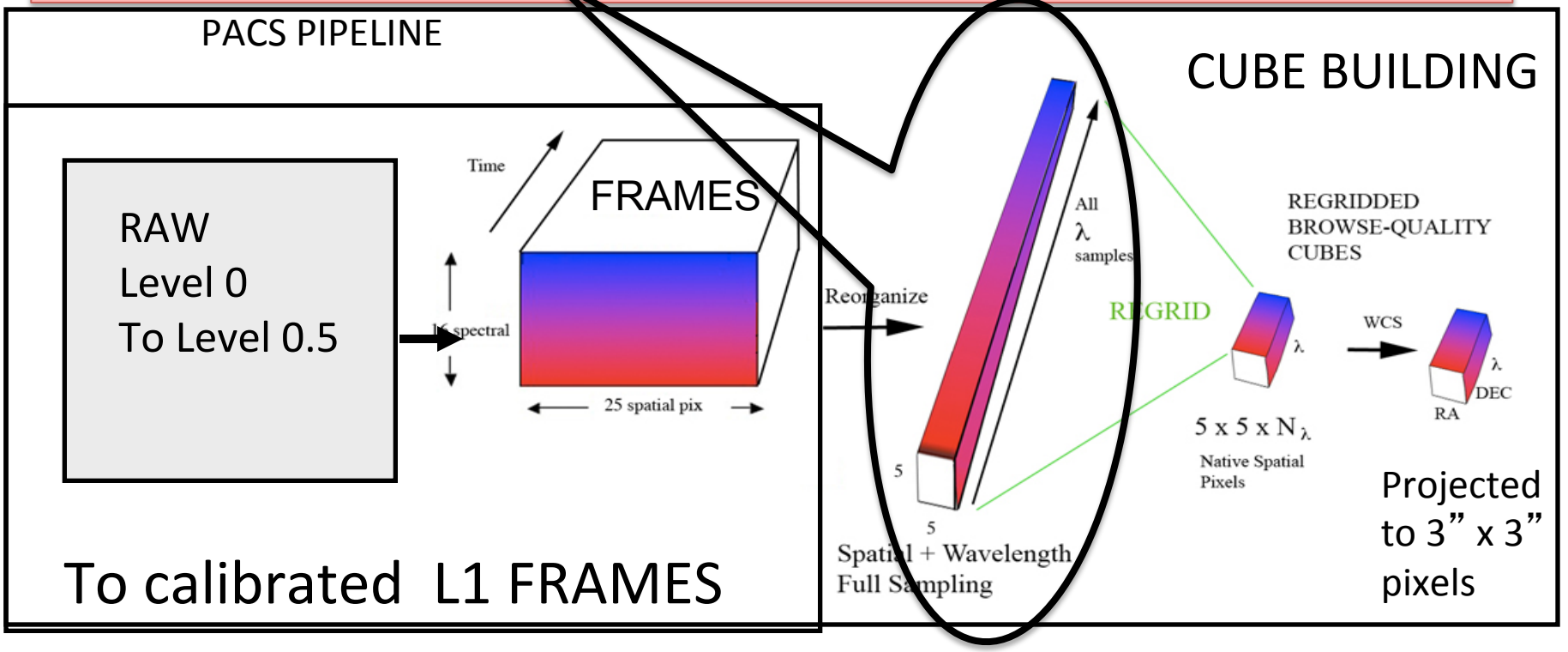
CALIBRATED FRAMES | PACSCUBE CUBE | REBINNED CUBE | PROJECTED CUBE

Level 1 | Level 2

http://nhsc.ipac.caltech.edu/helpdesk/index.php

Let's recap the structure of the data products so far, and see where we are going:

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



CALIBRATED FRAMES | PACSCUBE CUBE | REBINNED CUBE | PROJECTED CUBE

Level 1 | Level 2

http://nhsc.ipac.caltech.edu/helpdesk/index.php

Level 1 to 2 overview:

Step 1

Check number of Frame and Cube Slices from level 1

Step 2 (optional)

Select a specific line you want to reduce from those observed

Step 3

Deglitch and Inspect the central spaxel

Step 4

Create a wavelength grid for binning

Step 5

Flag data with user-controlled sigma-clipping

Step 6

Check "Outliers" mask for spectrum

Step 7

Flag data with user-controlled sigma-clipping, and average the NODS

Step 8

Inspect the Rebinned Cube Spaxel-by-Spaxel

Step 9 *(for Raster or dither pattern mapping of extended sources)*

Project Rebinned Cubes (via specProject or Drizzling) onto a WCS

Step 9 *(for point sources and single pointings)*

Correct the central pixel; extract the central pixel; and possibly projecting the Rebinned Cube

Step 10

Explore Rebinned and Projected cubes in Spectrum Explorer

loop over N lines (optional)

The Pipeline script will automatically loop over all lines if you leave the lineId blank in step 2.

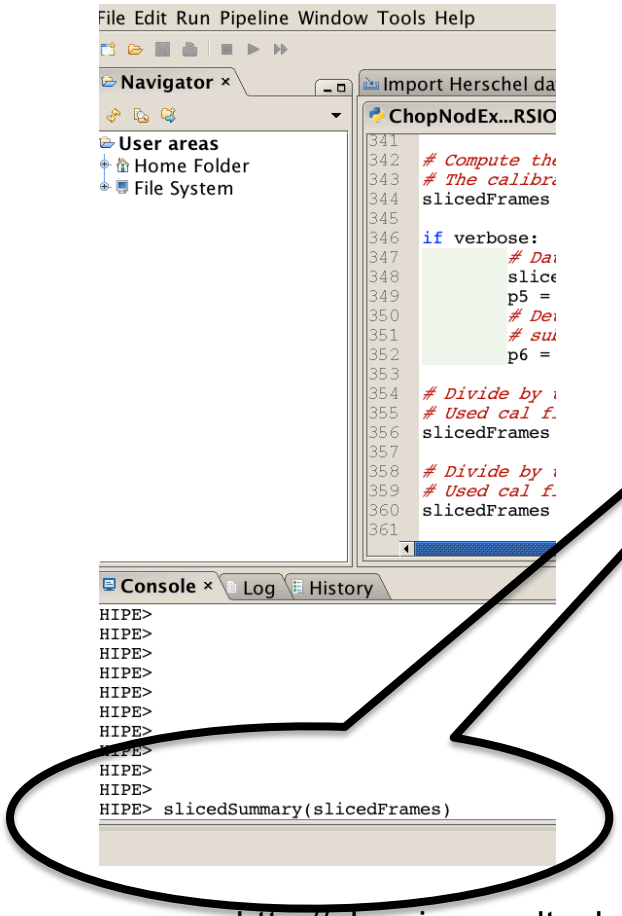
Step 1

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 x 5 x N where the 5 x 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 pacsCube as a spatial representation of sky (5 x 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 had 4 slices of Frames. 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 each with two Nod observations. Had we done not one, but "n" repetitions of the Nod cycle, we would 2n separate nods for each line. 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

Check One—Lets look to check that we have the correct number of level1 Frames and pacsCubes



Type into the Console window the following

slicedSummary(slicedFrames)
slicedSummary(slicedCubes)

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 pix 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 (optionally) select a line for processing.
At the end of the whole process you will optionally return to Step 2 to process any further lines

Next, we select a particular line to process.

The easiest way is to chose 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 is shown.

```
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. We will make rebinned cubes and projected cubes for this line. (Since this is not a raster observation, we actually need not make the projected cubes, but we do it nonetheless as an example.)

SELECT THE LINE OF CHOICE

At this point you can run the script (level 1->2) for all lines with lineId=[], or else enter the lineId of choice into the script, as shown

```
574 # ------------------------------------------------------------------------
575 #         Processing      Level 1 -> Level 2
576 # ------------------------------------------------------------------------
577 #
578
579 """
580 # From this point on, you are free to deal with a subselection of your data
581 # This can be one or several line(s) or range(s), raster position(s), one nod, etc.
582 # If you wish to do so, use slicedSummary(slicedCubes) to get an overview of the
583 # content and structure of your data, and selectSlices for the actual selection, e.g.:
584 selectedCubes = selectSlices(slicedCubes, lineId=[2], wavelength=[], rasterLine=[],|
585          rasterCol=[], nodPosition="", nodCycle=[], band="", scical="",|
586          sliceNumber=[], verbose=verbose)
587 """
588
589 if verbose:
590         slicedSummary(slicedCubes)
591         # Plot the data points that were rejected for some masks, e.g. GLITCH
```

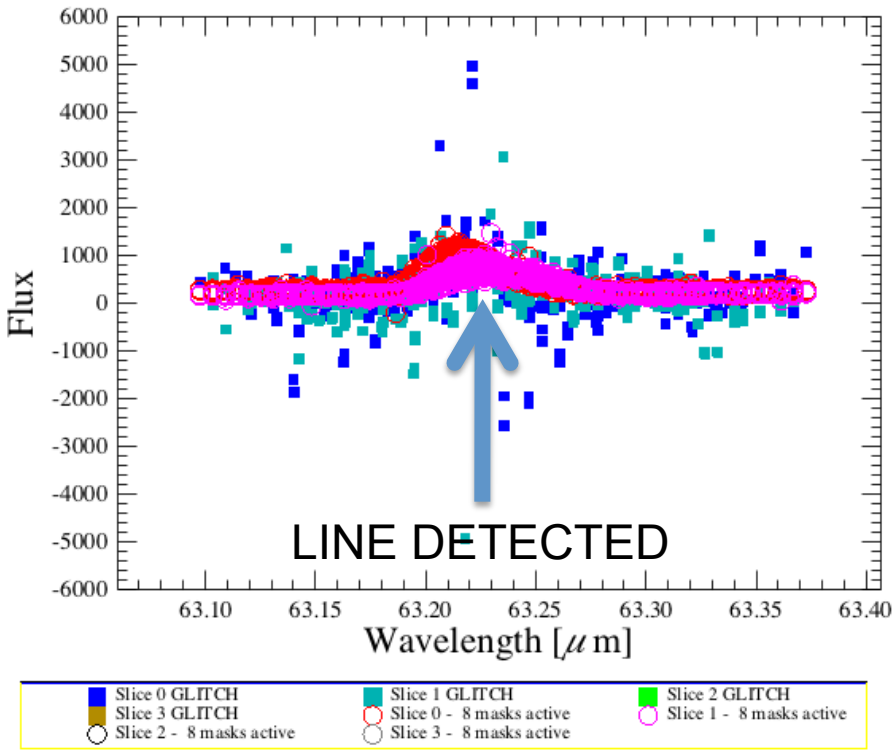
Edit the script to set lineId = [2] and set all other selections to []. Then execute down to the creation of a new set of sliced cubes called slicedCubes. These should now only contain pacsCubes for lineId=2

Step 3

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

The verbose diagnostics plot the result for each slice. We zoom-in ***by hand on the plot*** on the lineId = [2], the [OI] 63 microns found in slices 0 and 1.

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



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 that too many points are being flagged, you can employ a workaround that will ignore the glitch mask here and rely instead on sigma-clipping at a later point in the Pipeline. (We explain later how to do this, and you can try both ways.)

For the moment we will accept the glitch mask results as correct.

IF YOUR LINE IS WEAK, YOU MAY NOT SEE IT AT THIS STAGE. It may appear after further clipping is performed.

Step 4

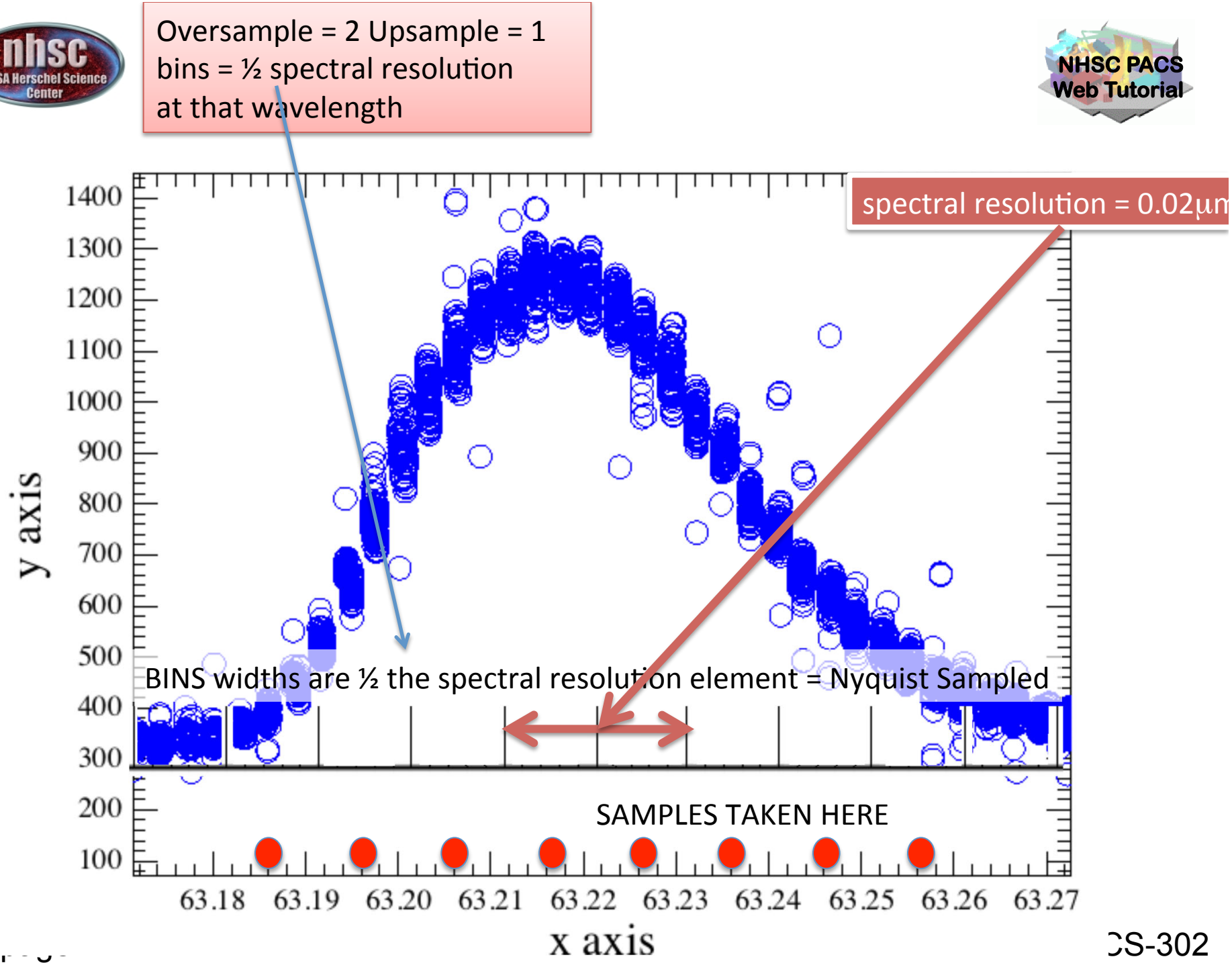
Create a wavelength grid. This will be used to grid the spectrum in "wavelength-space" and is the first step in constructing a re-binned cube. In the default case the grid is Nyquist-sampled. However, the user can select a finer grid or, alternatively, a courser grid to effectively provide additional smoothing.

Creating a wavegrid: the grid resolution and number of phased samplings are governed by the parameters "oversample" and "upsample", respectively. The default values of these are oversample = 2, upsample = 1. The default values provide a grid which oversamples the intrinsic spectral resolution element by a factor of 2 (i.e., Nyquist sampling). An upsample value of 1 means that one contiguous set of wavelength bins are laid-out. When oversample is "m", the binwidth used is the intrinsic resolution element/m. When upsample is "n", n sets of wavlength bins are laid-out, each phased by the binwidth/n.

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

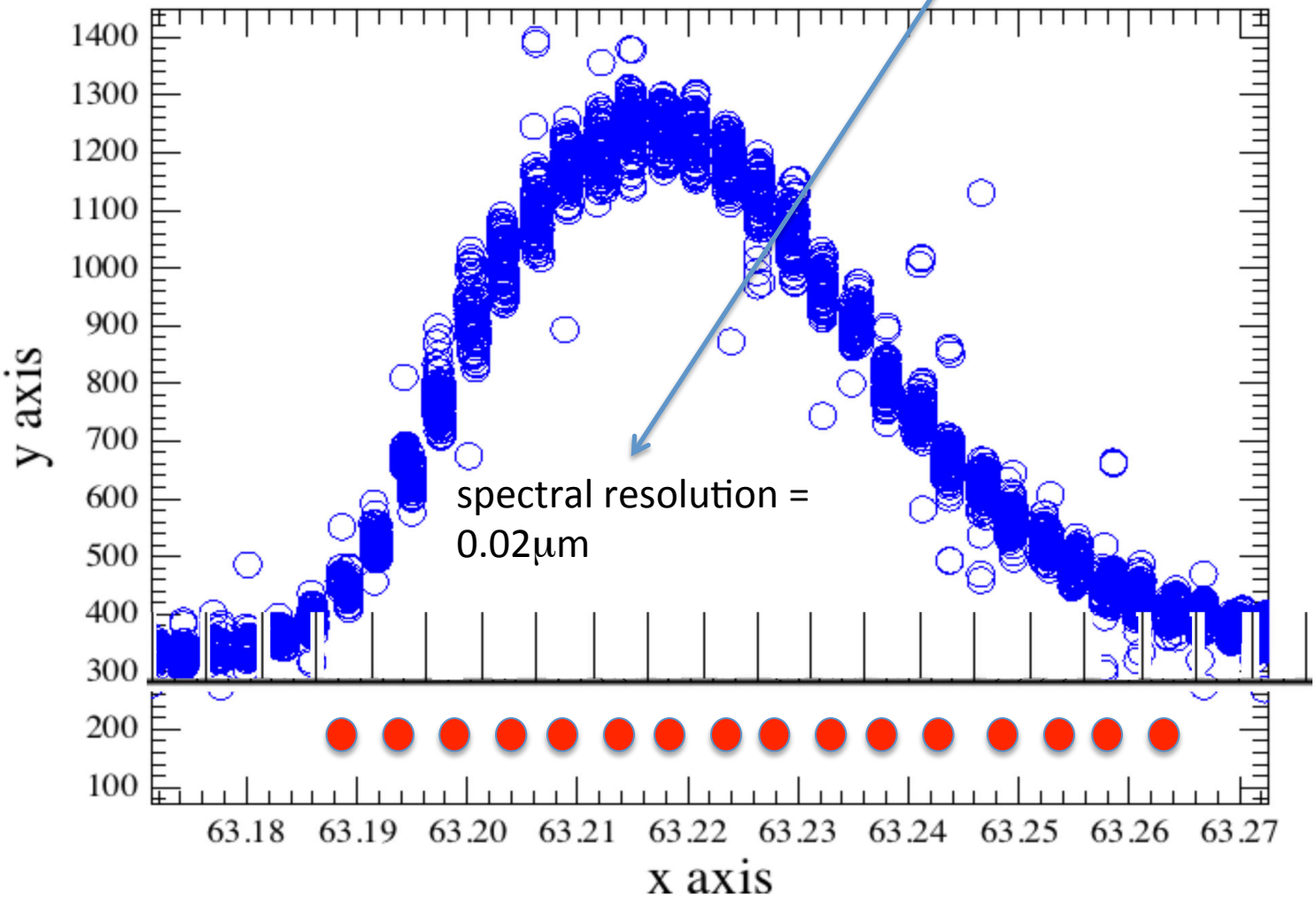
Illustrative examples follow….

Examples of Oversample and Upsample Parameter Pairs

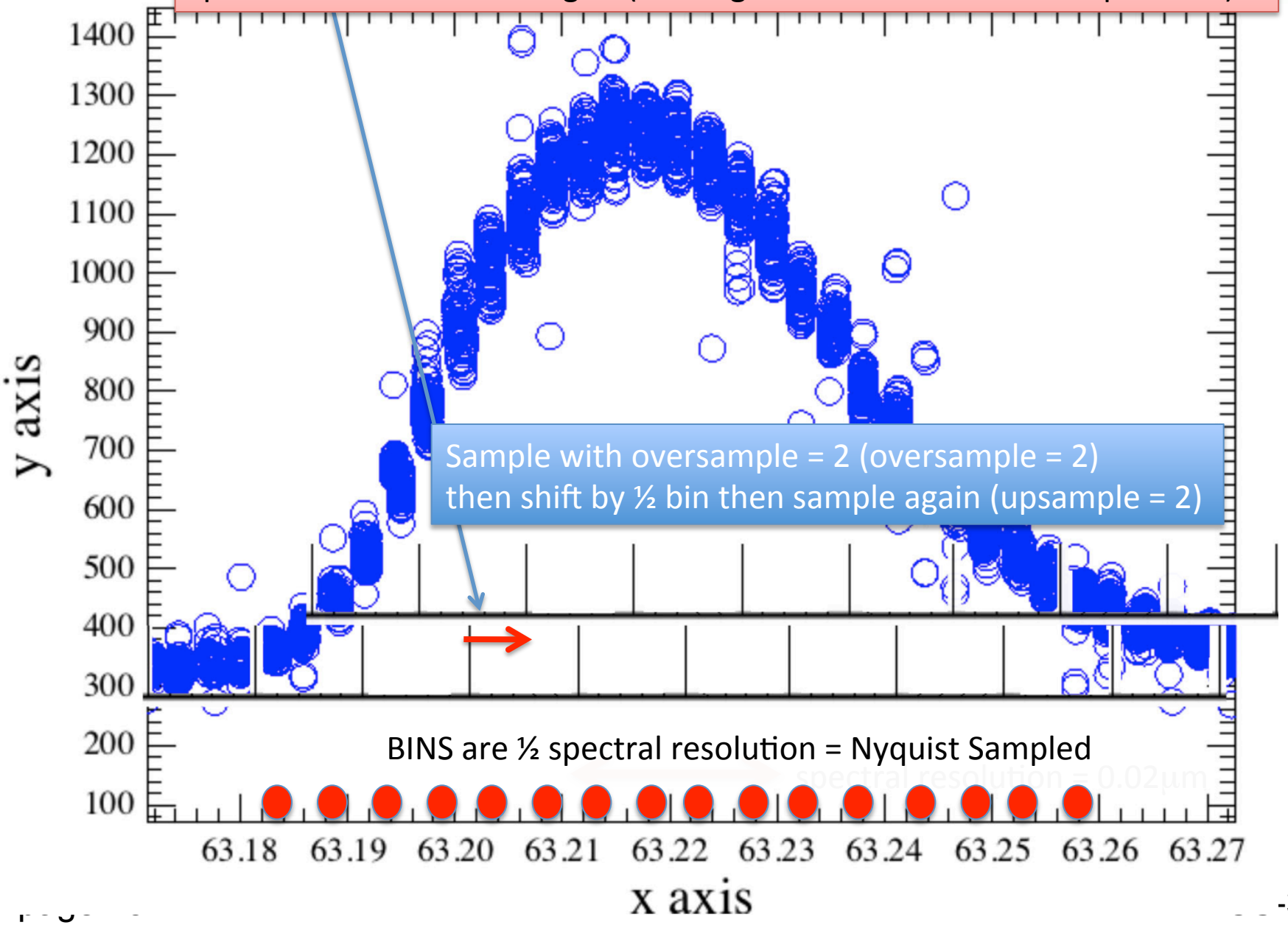

Oversample = 2 Upsample = 1
bins = ½ spectral resolution
at that wavelength
spectral resolution = 0.02µm
BINS widths are ½ the spectral resolution element = Nyquist Sampled
SAMPLES TAKEN HERE
y axis
x axis
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
63.18
63.19
63.20
63.21
63.22
63.23
63.24
63.25
63.26
63.27

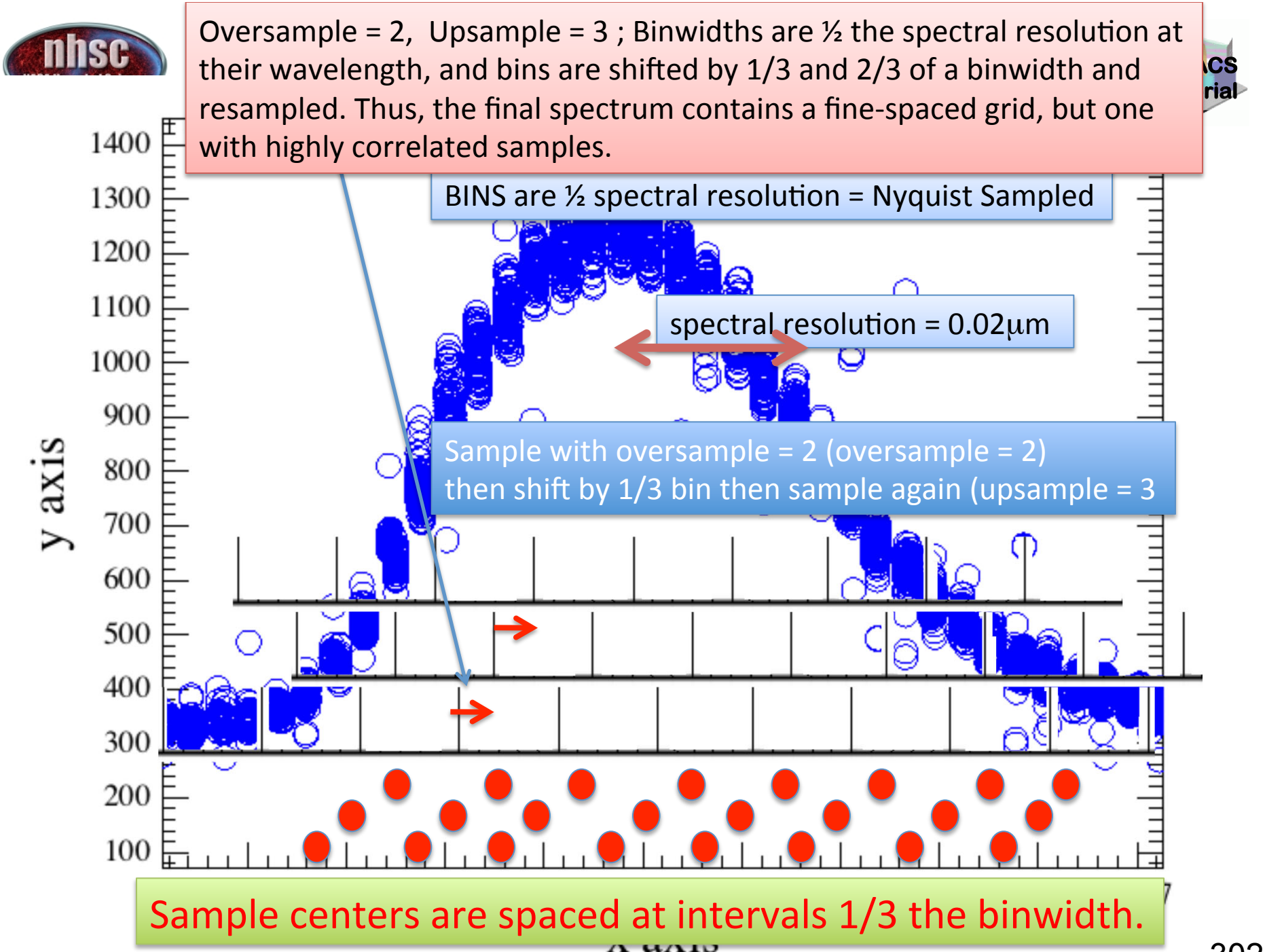
Oversample = 4, Upsample = 1
bins = 1/4 spectral resolution
at that wavelength

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



Oversample = 2 Upsample = 2 bins = ½ spectral resolution at that wavelength, but bins are shifted by ½ a bin and resampled, thus the final spectrum contains a finer grid (although the values are not independent).




Oversample = 2, Upsample = 3 ; Binwidths are ½ the spectral resolution at their wavelength, and bins are shifted by 1/3 and 2/3 of a binwidth and resampled. Thus, the final spectrum contains a fine-spaced grid, but one with highly correlated samples.
BINS are ½ spectral resolution = Nyquist Sampled
spectral resolution = 0.02μm
Sample with oversample = 2 (oversample = 2)
then shift by 1/3 bin then sample again (upsample = 3
y axis
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
Sample centers are spaced at intervals 1/3 the binwidth.

Step 5

Before regridding, we apply OUTLIERS rejection flagging to the spectra. This process is quite robust and is found to work well for PACS spectra. If the user suspects that the Deglitcher is over-flagging with GLITCH flags, the effect of these GLITCH flags can be turned-off and the de-glitching can instead be done with the OUTLIERS rejection flag method

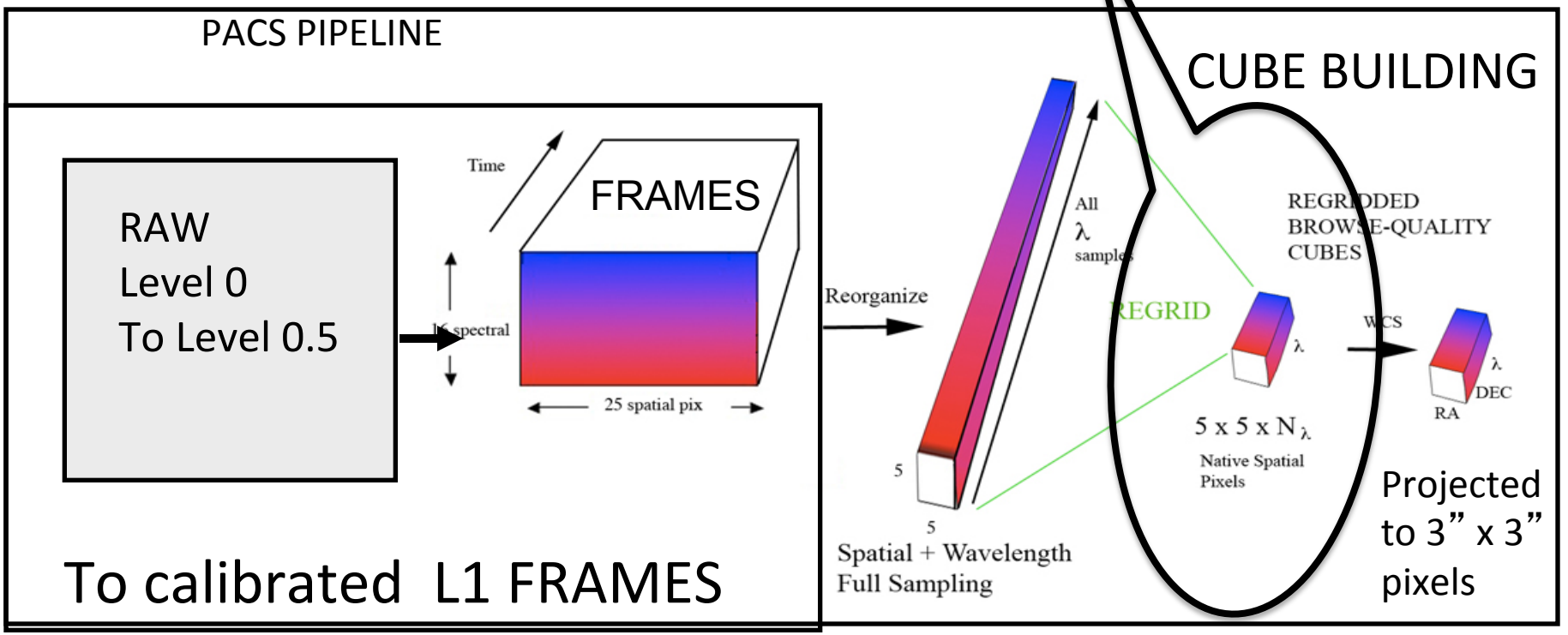
Next we active the masks and apply a sigma-clipping routine to the spectra (See PACS Data Reduction Guide: Spectroscopy, 3.3.8. for details)

```
# Activate all masks
slicedCubes = activateMasks(slicedCubes, String1d(["SATURATION","RAWSATURATION","NOISYPIXELS",
                    "BADPIXELS","UNCLEANCHOP","GRATMOVE","GLITCH"]), exclusive = True)
# Flag the remaining outliers, (sigma-clipping in wavelength domain)
slicedCubes = specFlagOutliers(slicedCubes, 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 these two parameters. The output of this routine is to create a mask. The mask is later applied when the actual rebinning of the spectra onto the waveGrid takes place. The mask is called “OUTLIERS” and is 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 lineId=2 case, we have two—one for Nod A and one for Nod B)



CALIBRATED FRAME → PACSCUBE CUBE (Level 1)

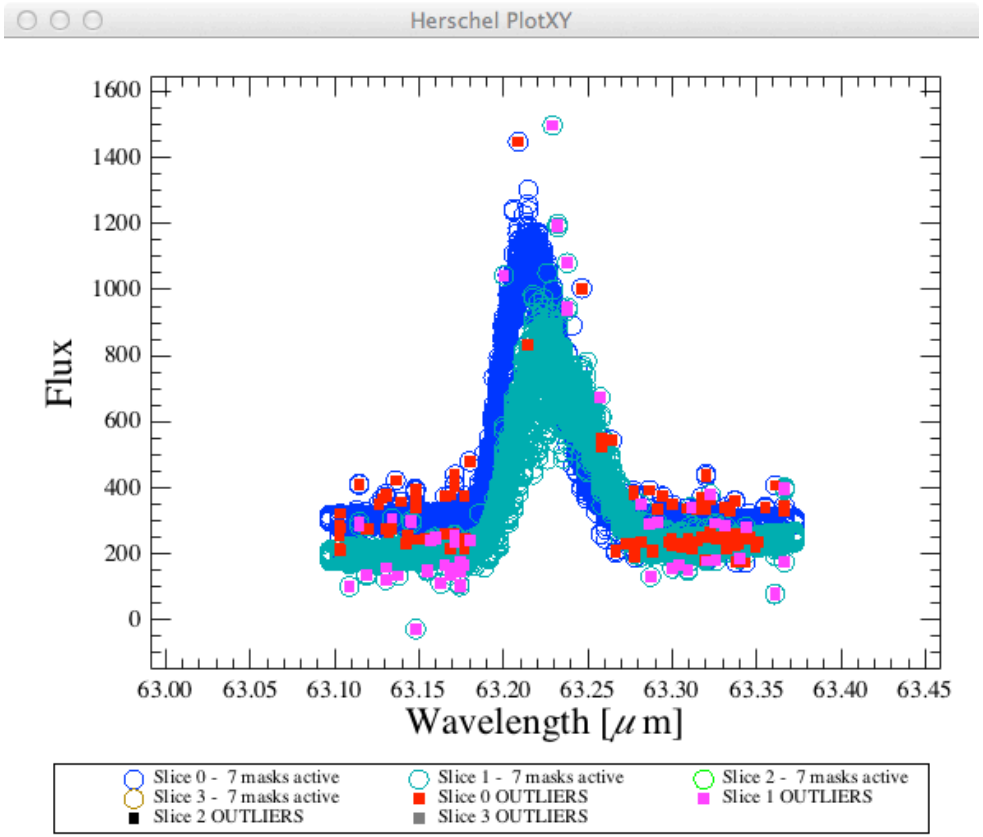
REBINNED CUBE → PROJECTED CUBE (Level 2)

http://nhsc.ipac.caltech.edu/helpdesk/index.php

Step 6

Inspect the spectra again this time looking at the "OUTLIERS" flag.
Check that it appears to be flagging points sensibly.

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. This happens before rebinning

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

# Rebin all selected cubes on the same wavelength (mandatory for specAddNod)
slicedRebinnedCubes = specWaveRebin(slicedCubes, 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 last 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 (con't)

Average the Nods

Now we average together 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

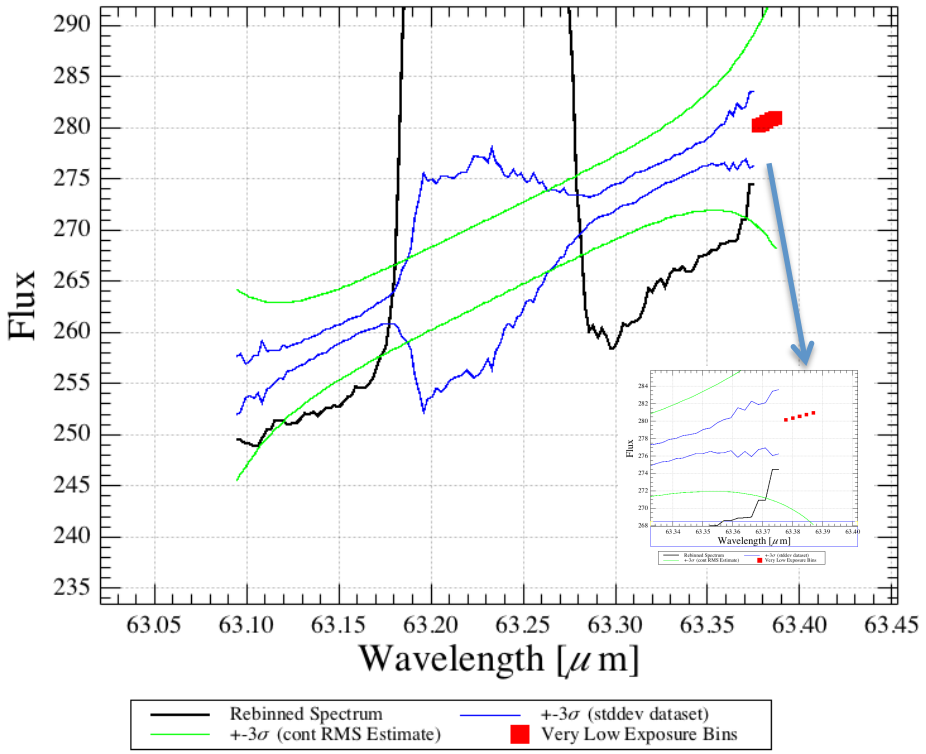
```
# Average the nod-A & nod-B rebinned cubes.
# All cubes at the same raster position are averaged.
# This is the final science-grade product currently possible, for all observations except
# the spatially oversampled raster observations
slicedFinalCubes = specAddNodCubes(slicedRebinnedCubes)
```

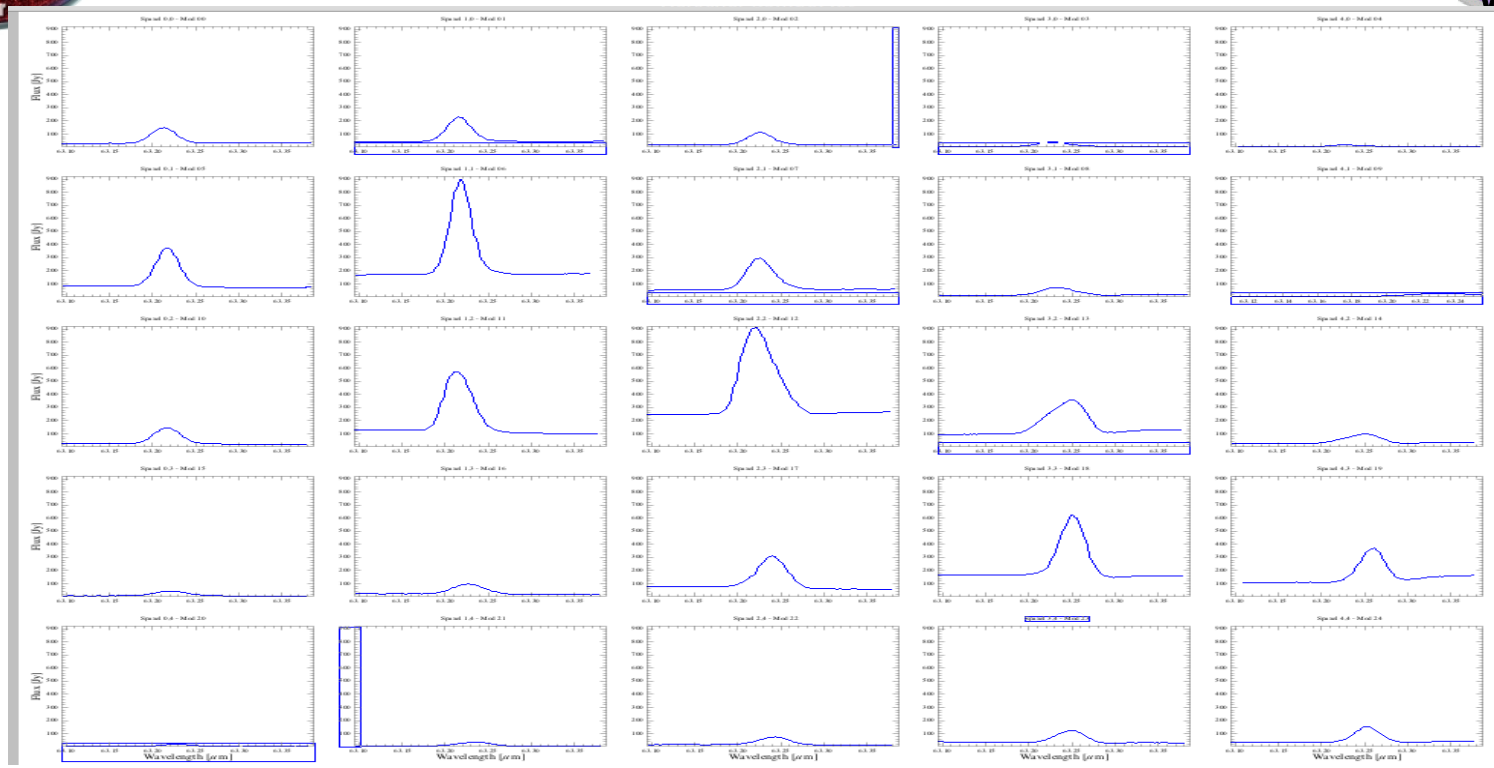
Step 8

Inspect the rebinned spectra—spaxel by spaxel using various diagnostic tools

```
if verbose:
   slicedSummary(slicedFinalCubes)
   # Central Spaxel for all raster positions (at this stage,
   # normally only 1 / range and / raster position)
   x,y = 2,2
   p10 = plotCubes(slicedFinalCubes, x=x, y=y,stroke=1)
   p10.xtitle,p10.ytitle,p10.titleText,p10.subtitleText=
        "Wavelength [$\mu$m]","Flux [Jy]",str(obsid)+"
        "+camera,"Spaxel ["+str(x)+","+str(y)+"]"
   #
   # Same as above, now overplotting an estimate of
   # the continuum RMS (biased  'inside' the spectral lines).
   # For line scans, the continuum level is estimated from
   # a polynomial of order 1 fitted outside the spectral line.
   p11 = plotCubesStddev(slicedFinalCubes, plotLowExp=1,
           plotDev=0, nsigma=3, isLineScan=1, spaxelX=x,
           spaxelY=y, verbose=verbose, calTree=calTree)
   #
   # 5x5 plot of one of the rebinned cubes
   # (one line/range at one raster position)
   slice = 0
   p12 = plotCube5x5(slicedFinalCubes.get(slice))
   #
   # Interactive inspection of your final cube
   oneFinalCube = slicedFinalCubes.get(slice)
   openVariable("oneFinalCube","Spectrum Explorer")
```

With the verbose=1 diagnostics, we can view the central pixel (e.g.) profile along with the its estimated rms of line and continuum flux.





Here we plot only the first slice. In the case we have set the lineId=2, so we have only the [OI] line.

In the case of a raster observation, you can plot the profiles for each slice separately.

if verbose:

.

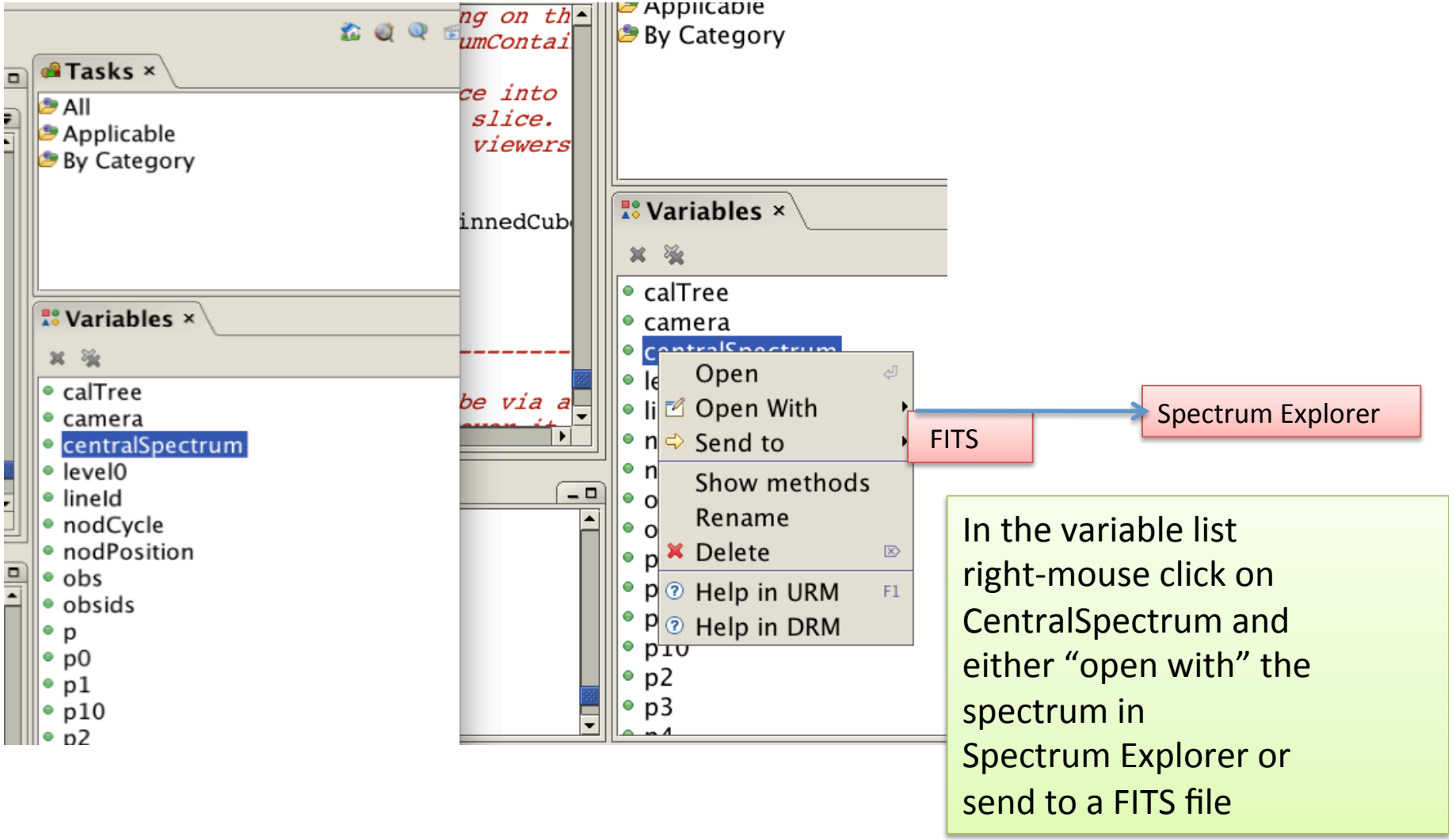
```
#
# 5x5 plot of one of the rebinned cubes
# (one line/range at one raster position)
slice = 0
p12 = plotCube5x5(slicedFinalCubes.get(slice))
#
# Interactive inspection of your final cube
oneFinalCube = slicedFinalCubes.get(slice)
openVariable("oneFinalCube","Spectrum Explorer")
```

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 spaxel X 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



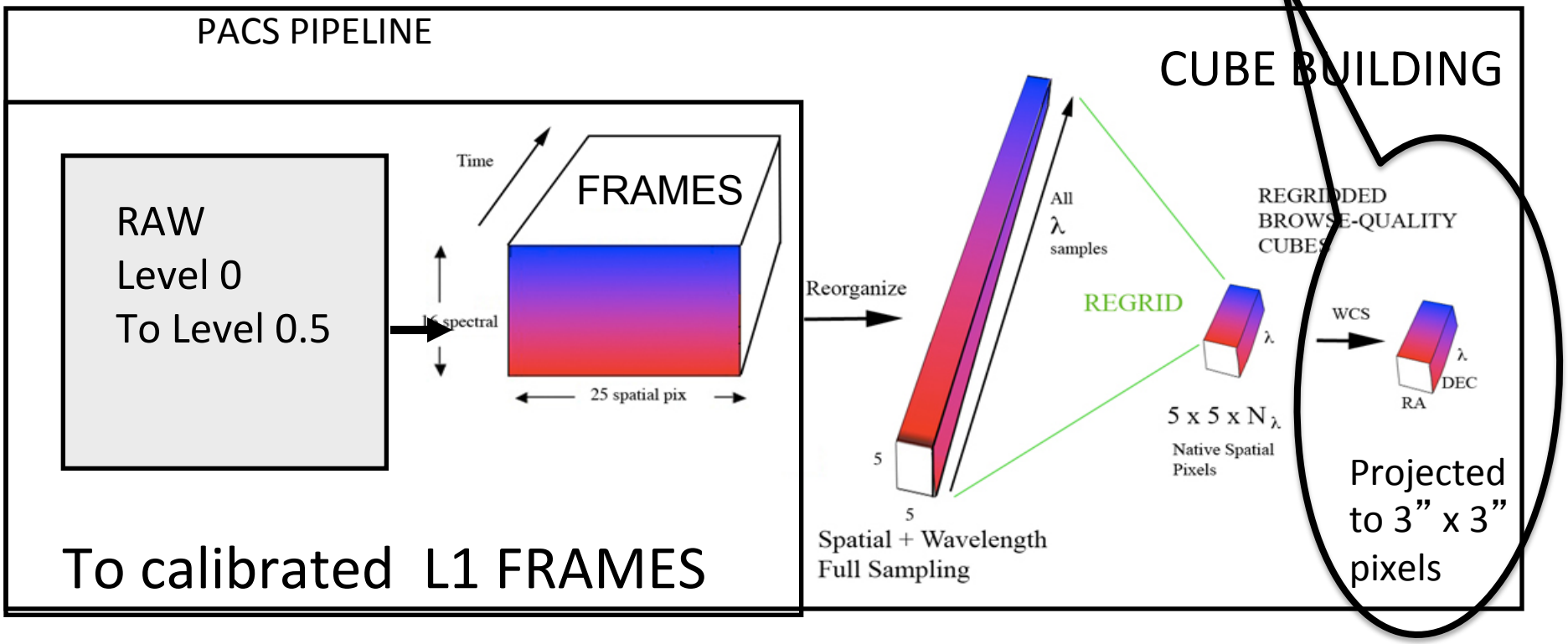
In the variable list right-mouse click on CentralSpectrum and either “open with” the spectrum in Spectrum Explorer or send to a FITS file

Step 9 *(for Raster Observations of Extended Sources)*

In order to mosaic your cubes onto a uniform World Coordinate System (WCS) grid, users with raster observations* should select between the two methods of mosaicking: you may using the superior method called "Drizzling", which is also by far the more computer intensive method, or the alternative method, called "specProject" The methods project onto a 3" grid by default.

*As opposed to the single-pointing example data set of this tutorial)

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

http://nhsc.ipac.caltech.edu/helpdesk/index.php

specProject

(Refer to section 3.7 of the "PACS Data Reduction Guide for Spectroscopy") For it relative speed, specProject is recommended for large wavelength range observations and SEDS. Each output spaxel is formed from a weighted combination of overlapping and/or undersampled spaxels from the slicedFinalCubes (which have been rebinned in wavelength) for each spatial and wavelength point onto a (default) 3" grid.

Invoking specProject

```
# specProject works on the final rebinned cubes.
slicedProjectedCubes = specProject(slicedFinalCubes,outputPixelsize=3.0)
```

```
# Display the projected cube in the Spectrum Explorer
if verbose:
   slice = 0
   oneProjectedSlice = slicedProjectedCubes.refs[slice].product
   openVariable("oneProjectedSlice", "Spectrum Explorer")
```

The present script makes a slicedProjectedCube and even opens up a slice in Soectrum Explorer. Also available are other viewers (via right-mouse clicking on the variable “slicedProjectedCube”) as well as the same FITS writing capability as before.

Drizzling

The Drizzling projection is so-named because the total "drops" of emission collected within input pixels are broken into smaller "droplets" to be collected and weighted into the output pixels. It differs from spacProject in that it operates on the Level 1 slicedPacsCubes prior to wavelength rebinning (not Level 2 cubes) and also differs in the inputs it takes when using the unchopped or chopped AORS.

The ChopNodLineScan.py script includes both the Drizzle and specProject methods.

(Ref. to a primary drizzle document: http://www.stsci.edu/ftp/science/hdf/combination/drizzle.html

Invoking Drizzling (slicedDrizzled Cubes)

For parameter details, refer to section 3.7 of the PACS Data Reduction Guide (Spectroscopy)

```
# Drizzling is new from hipe 10.0. It works from the not-rebinned PacsCubes
oversampleWave  = 2
upsampleWave    = 3
waveGrid        = wavelengthGrid(slicedCubes, oversample=oversampleWave,
  upsample=upsampleWave, calTree = calTree)
oversampleSpace = 3
upsampleSpace   = 2
pixFrac         = 0.6
spaceGrid       = spatialGrid(slicedCubes, wavelengthGrid=waveGrid,
  oversample=oversampleSpace, upsample=upsampleSpace,
  pixfrac=pixFrac, calTree=calTree)
slicedDrizzledCubes = drizzle(slicedCubes, wavelengthGrid=waveGrid,
  spatialGrid=spaceGrid)
```

These Drizzled Cubes may be viewed in Spectrum Explorer as well.

Invoking Drizzling (WARNING)

Warning: In Pipeline 11 – DRIZZLING is run by default – that is – the drizzling script lines shown in the previous slide are **not commented-out**.

The user should be sure that he/she wants to run drizzle before invoking it.

The drizzle routine is an excellent optimized and powerful projection method (when sufficient oversampling has been obtained within the observation) but it can require many hours of execution time and large amounts of computer memory, especially for large maps or line ranges. It is best run on a large and perhaps remote virtual machine (via a VNC session) and utilizing multi-threading.

The user's alternative, specProject will, in general, yield a fairly good projected map in much shorter processing times.

Step 9 *(for point sources and single pointings)*

Correct the central pixel; extract the central pixel; and possibly projecting the Rebinned Cube

Caveat: Making a projected cube from a single pointing

Making a projected cube (using specProject or Drizzling) from a single pointing is not recommended, but it is also not prohibited. However, the formal final product from a single pointing is considered to be a rebinned cube, not any projected cube. One may attempt to project a single pointing for visualization purposes (such as is done in this tutorial), but, because the PACS IFU footprints are distorted, flux conservation is typically not maintained in the output map.

Only in the case of a fully sampled raster (spatial dithered mapping) can one expect to recover proper fluxes from a projected cube.

Correcting and Extracting Point Sources

Caveats: The instructions below assume that the user has employed a single pointing for a target that is a point source*, and the target's photocenter is somewhere within the central spaxel [2,2] (i.e., the flux peaks at spaxel [2,2])**.

There are three actions or corrections that can be applied to point source observations, at least two of which the user will likely wish to do: a) extract the spectrum of the central pixel b) correct the flux in the central pixel for the undersampling (due to the beam profile) of the point source flux that the central spaxel sees, c) If there is sufficient signal-to-noise in the 8 pixels surrounding the central spaxel, correct for pointing jitter that may have left less flux in the central spaxel (this is the so-called 3x3 correction).

Discussions of these steps are found in the PACS Data Reduction Guide: Spectroscopy section 3.7.3 and for drizzle section 6.5 and in this summary: https://nhscsci.ipac.caltech.edu/workshop/PACS_Webinars_2012/Dec2012/pointSources.pdf

*If there are multiple pointings, these should be handled separately as per the instructions below using the individual rebinned cubes. (The PACS point source corrections operate only on individual rebinned cubes.)
**If the source is in a spaxel other than the central spaxel, use "PACS Point Source Loss Correction" script found in HIPE->Scripts->PACS useful scripts.

Correcting and Extracting Point Sources

a) Extracting the central pixel. In general the central spaxel will have far better signal to noise than the surrounding spaxel. While flux is loss from the central spaxel it is typically better to correct the flux than to add the measurements of the adjacent pixels. The task extractCentralSpectrum will produce a 2^d "simple spectrum" from the central spaxel.

b) When extractCentralSpectrum is used, and applyPointSourceCorrection is set to 1, the spatial undersampling correction for the beam at a given wavelength is applied during the extraction. The effect can be large: at the longest wavelength over 65% of the flux can be lost. For individual lines, "width" should be set to 0 to apply one constant correction across the line wavelengths

c) When extractSpectrum pixel is used, and *if* there is good signal to noise in the surrounding 8 pixels, the 3x3 correction be applied. This routine estimates the pointing offset by the relative level of flux found in each spaxel. The pipeline script runs with this parameter correct3x3 set to 0 (off) and 1 (on) for

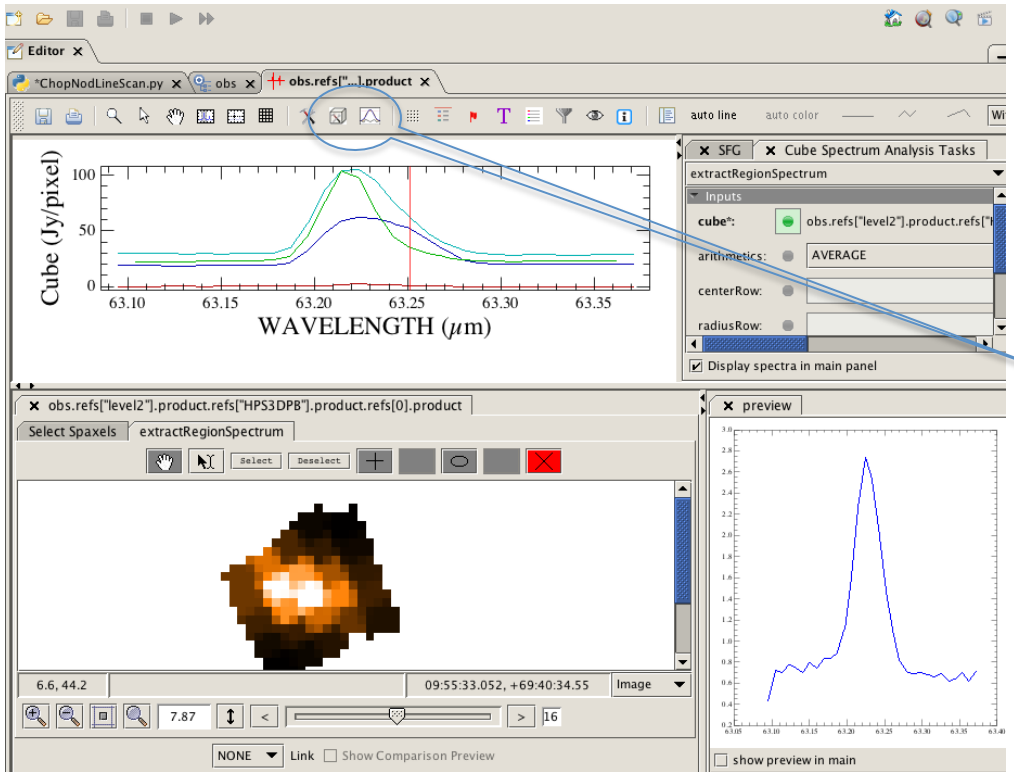
Extracting the central spaxel with both point source corrections set to "On"

```
# a. Extract central spectrum
# b. Apply a simple flux correction by comparison between the central spaxel and its 8 neighbours
# c. Apply the point source correction
# the 3x3 correction can be set to 0 or 1 depending on the signal to noise of the data
correct3x3 = 1
name = "OBSID_"+str(obsid)+"_"+target+"_"+camera
            +"_centralSpectrum_PointSourceCorrected_Corrected3x3YES_slice_"
            centralSpectrumCorr3x3 = extractCentralSpectrum(slicedFinalCubes,
            slice=slice,noNaNs=1,correct3x3=correct3x3, width=0,
            applyPointSourceCorrection=applyPointSourceCorrection, verbose=verbose,
            calTree=calTree)
```

Step 10

Explore your final WCS projected cube with the Spectrum Explorer or your favorite viewer and/or export to FITS.

It is suggested that you now explore your final cubes using the various tools offered!



You may display and examine the final spectral cubes with the Spectrum Explorer.
Further processing can be conducted there using, e.g., the Cube Tool Box and the Spectrum Fitter GUI. These allow for area extractions, baseline fitting and the formulation 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!