



# Overview of PACS Chop/Nod Pipeline(s)

Modified from various sources by
Phil Appleton for Hitchhiker's Guide to
Herschel Data

Largely based on
tutorials for running the pipelines is online at :

**https://nhscsci.ipac.caltech.edu/sc/index.php/Pacs/DataProcessing**

# PACS Spectrometer

## The PACS ICC recommends reprocessing of all Spectroscopy data for the following reasons:

- In the SPG bulk-reprocessing (i.e. the data in the HSA), only *one flavor* of the spectrometer pipeline per observing mode (e.g. chop nod) is run.

- However, *other flavors* of the pipeline might be more appropriate for your specific science case (depending on the brightness of your source and duration of your observation)

- Also, for a given pipeline flavor, not all tasks are run in the SPG bulk-reprocessing (e.g. drizzling)

# PACS Spectrometer

## Different pipeline flavors (for chop-nod modes)

- lineScan: calibration blocks + RSRF flux calibration (this is the standard pipeline run in SPG bulk reprocessing) → *recommended for bright sources*
- Background Normalization: uses telescope background for flux calibration → *recommended for faint sources or long duration observations or when broad band features are present.*
- Point Source Background Normalization: uses telescope background for flux calibration + correction for pointing jittering → *recommended for very bright point sources*
- *SPLIT ON/OFF Separation pipeline—If you need to separate Nods A and B. (Not to be used for regular data reduction).*

MAIN PIPELINES

# PACS Spectrometer

## Not all pipeline tasks are run in the bulk reprocessing

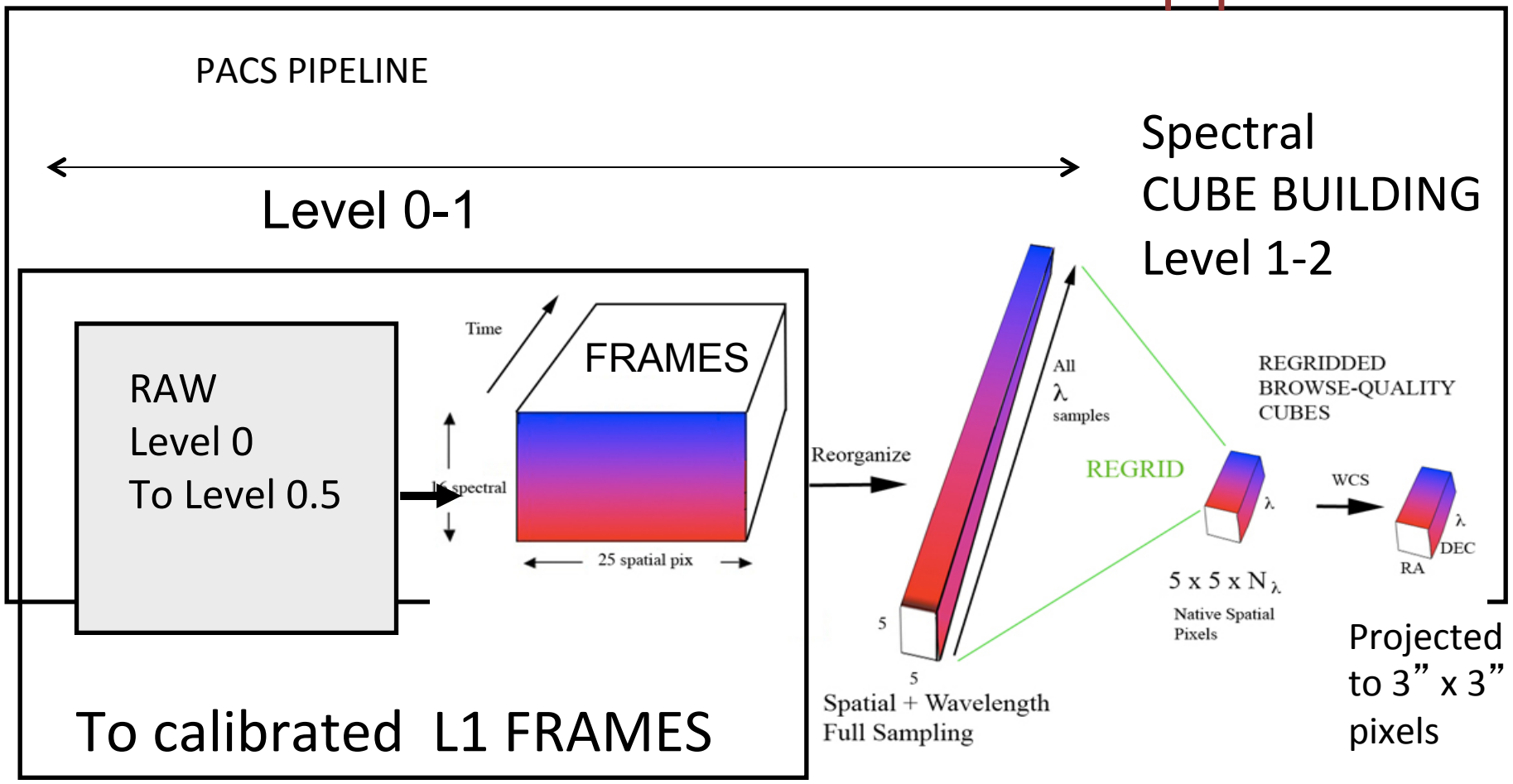
- Point sources: the re-binned cubes (HPS3DRB/R) are calibrated for extended emission by default. For point sources centered in the IFU (i.e. on spaxel (2,2)) run the task *extractCentralSpectrum.* The task works well for bright sources, i.e. continuum flux level in the range 5 – 10 Jy

- Slightly Extended Sources: the re-binned cubes (HPS3DRB/R) are calibrated for extended emission by default. For slightly extended sources (slightly bigger than the beam size at a given wavelength) centered in the IFU (i.e. on spaxel (2,2)), apply the task *pacsExtendedToPointCorrection*. You need to know the geometry of your source to successfully apply this task

- Extended sources – mapping: for well-sampled (i.e. Nyquist) line or short-range raster observations, the best projected cubes are obtained by running the *drizzling* task which is not automatically run in the SPG bulk reprocessing

NOTE: for extended sources observed without mapping (i.e. in pointed mode), use the rebinned cubes to extract your science, NOT the projected cubes.

# Overview of various entities in the pipeline




CALIBRATED FRAMES — PACSCUBE CUBE — Level 1

REBINNED CUBE — PROJECTED CUBE — Level 2

# Summary of PACS spectrometer archived products

(From Roberta's Talk yesterday)

- Level 0:

  HPENG: engineering
  HPGENHK: housekeeping
  HPSDMCB/HPSDMCR: raw DecMec (blue/red)
  HPSFITB/HPSFITR: Herschel PACS Spectrometer Fitted Blue/Red (→ "Signal" in /0)
  HPPHK: housekeeping
  HPSRAWB/HPSRAWR: raw data (blue/red)

- Level 1:

  HPS3DB/HPS3DR: Herschel PACS Spectrometer 3D Spectral Cube Blue/Red
  (→ "flux", "wave", "ra", "dec" in, e.g., /0: L2 N1 B R(0 0))
  HPSCALB/HPSCALR: calibration (response/dark)
  HPSFITB/HPSFITR: fitted slopes

- Level 2:

  HPS3DPB/HPS3DPR: Herschel PACS Spectrometer 3D Projected Cube Blue/Red
  (→ "image", "coverage" in, e.g., /0: L2 N1 A R(0 0))
  HPS3DRB/HPS3DRR: Herschel PACS Spectrometer 3D Rebinned Cube Blue/Red
  (→ "image", "ra", "dec", "stddev", "exposure", "wavegrid" in, e.g., /0: L2 N1 R(0 0)

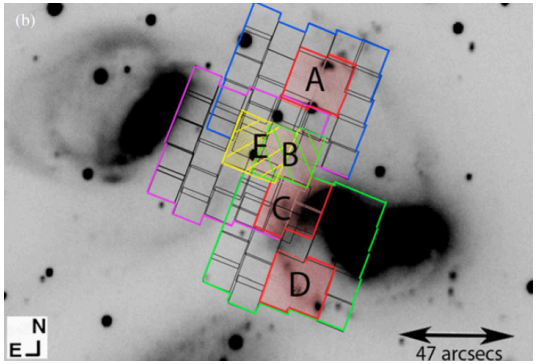
- Level 2.5: (only for range un-chopped): on and off positions are in different OBSIDs. In Level 2.5, the off is subtracted from the on

# Which final Chop/Nod product should I use?

Single pointings or sparse mapping

> Hipe 12.1 PacsRebinned Cube

Extract from rebinned
By selection of spaxels
(e.g. Appleton et al. 2013)




>HIPE 13—new projection task
called specInterpolate will make better interpolation—good for "sparse" maps with reasonable redundancy

Well-sampled mapping

> Hipe 12.1 SpecProject cubes (has wcs) wavelength channels are of variable width

> Hipe 13 Will create cube with "constant" wavelength channels (ds9 readable λ)

VERY Well-sampled maps > Drizzle maps

**Lets set up so basic things before we starts:**

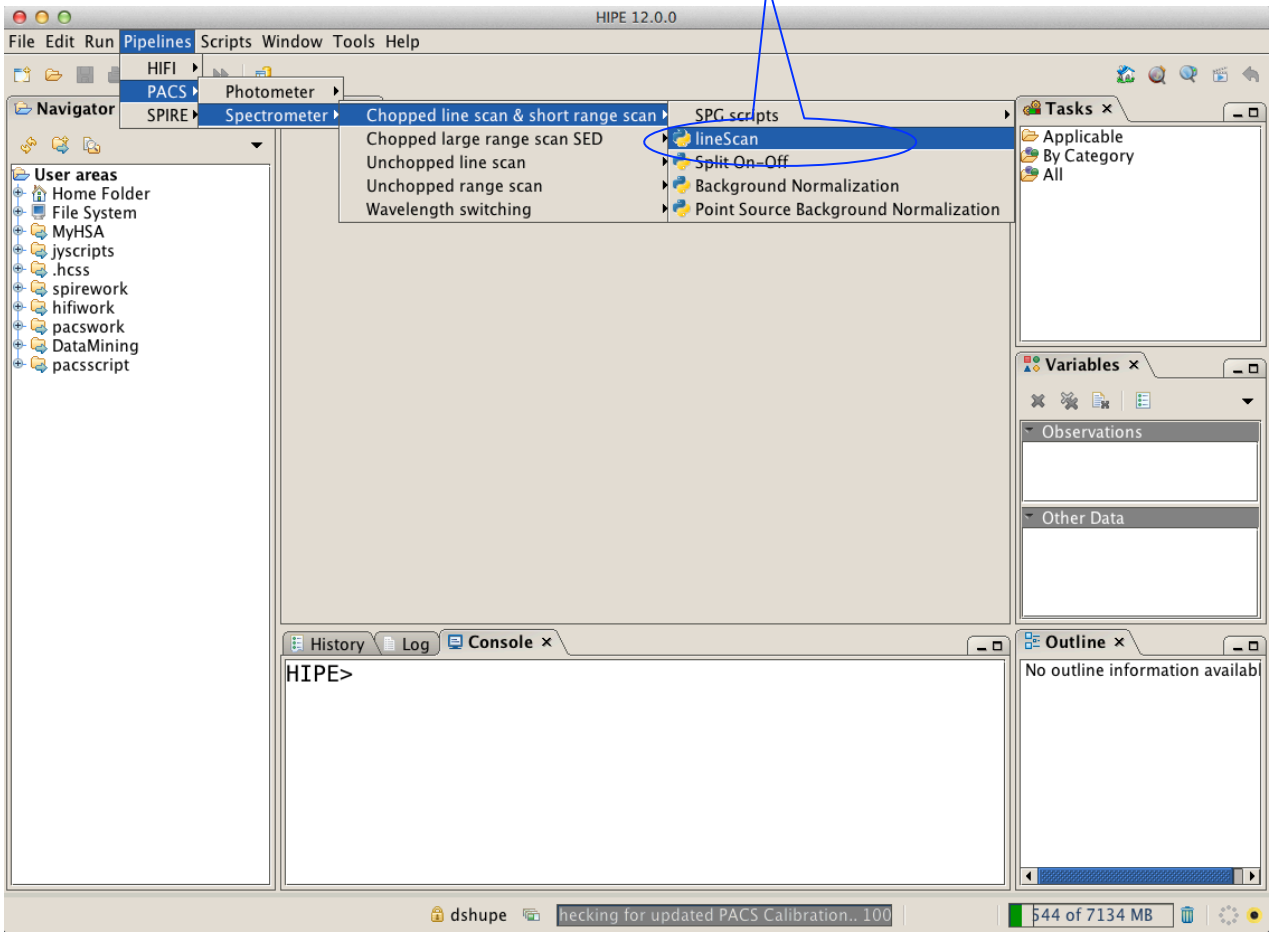
0) Provide obsid for your observation (use observation log)
1) Choose the correct pipeline
2) Read in the (obs)ervation context from correct place (either HSA or local pool)
3) Choose "camera". camera = "red" is all red lines (order 1) "blue" = "order 2 or 3"

NOTE—you need to run the pipeline twice if you have lines in both red and "blue"

4) Set up output directory
5) Have a look at an observation "summary" to make sure you have found the correct data

# Loading the script

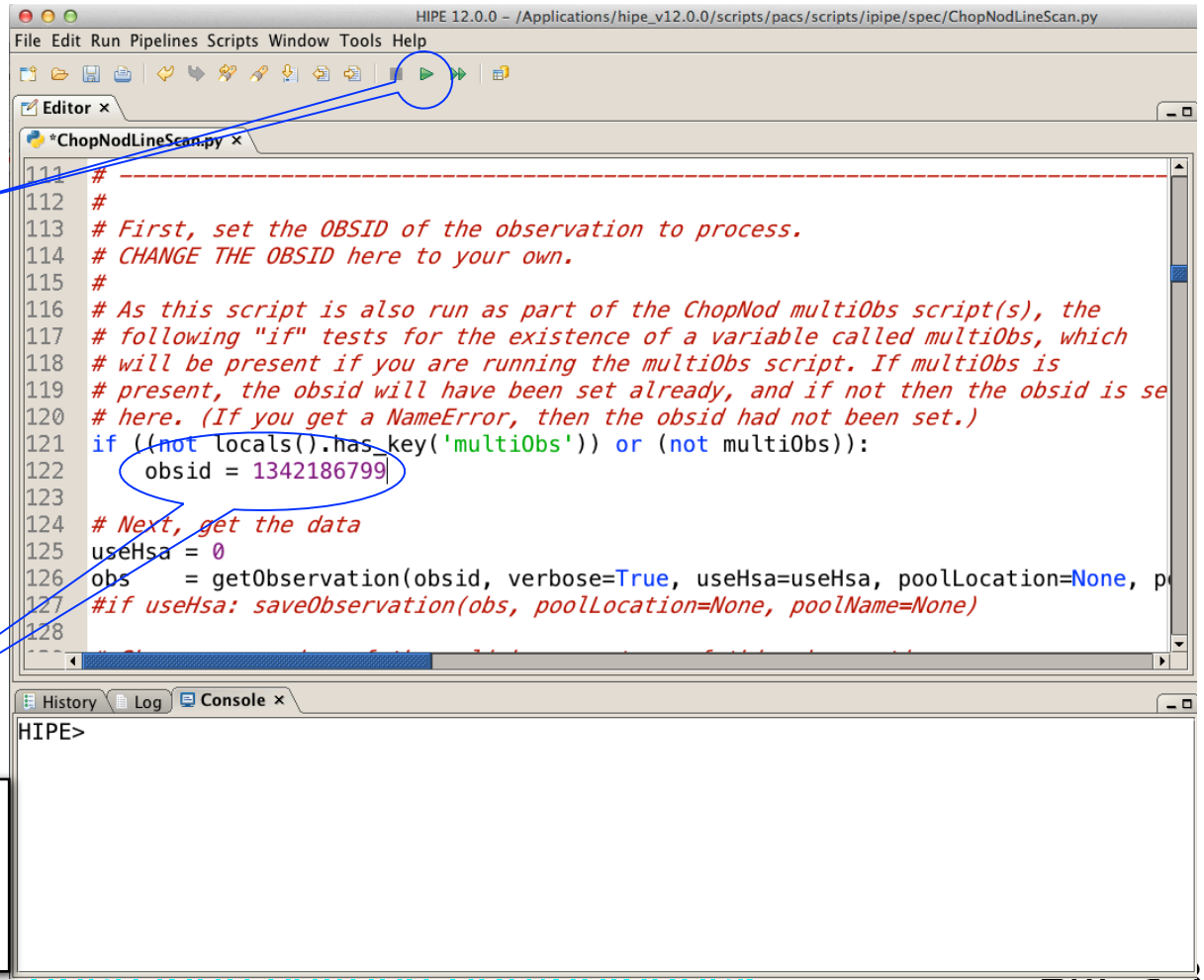
The “linescan” script used in this tutorial corresponds to the script available directly from the distribution.



# Loading the observation

Once the script is loaded, one simply steps through the lines to execute it. But first modify it for OBSID of the observation desired. Modify the obsid in the script and click through using the green arrow….

Hit the green arrow to step through the entire script




Modify this line. Here we set obsid to 1342186799.

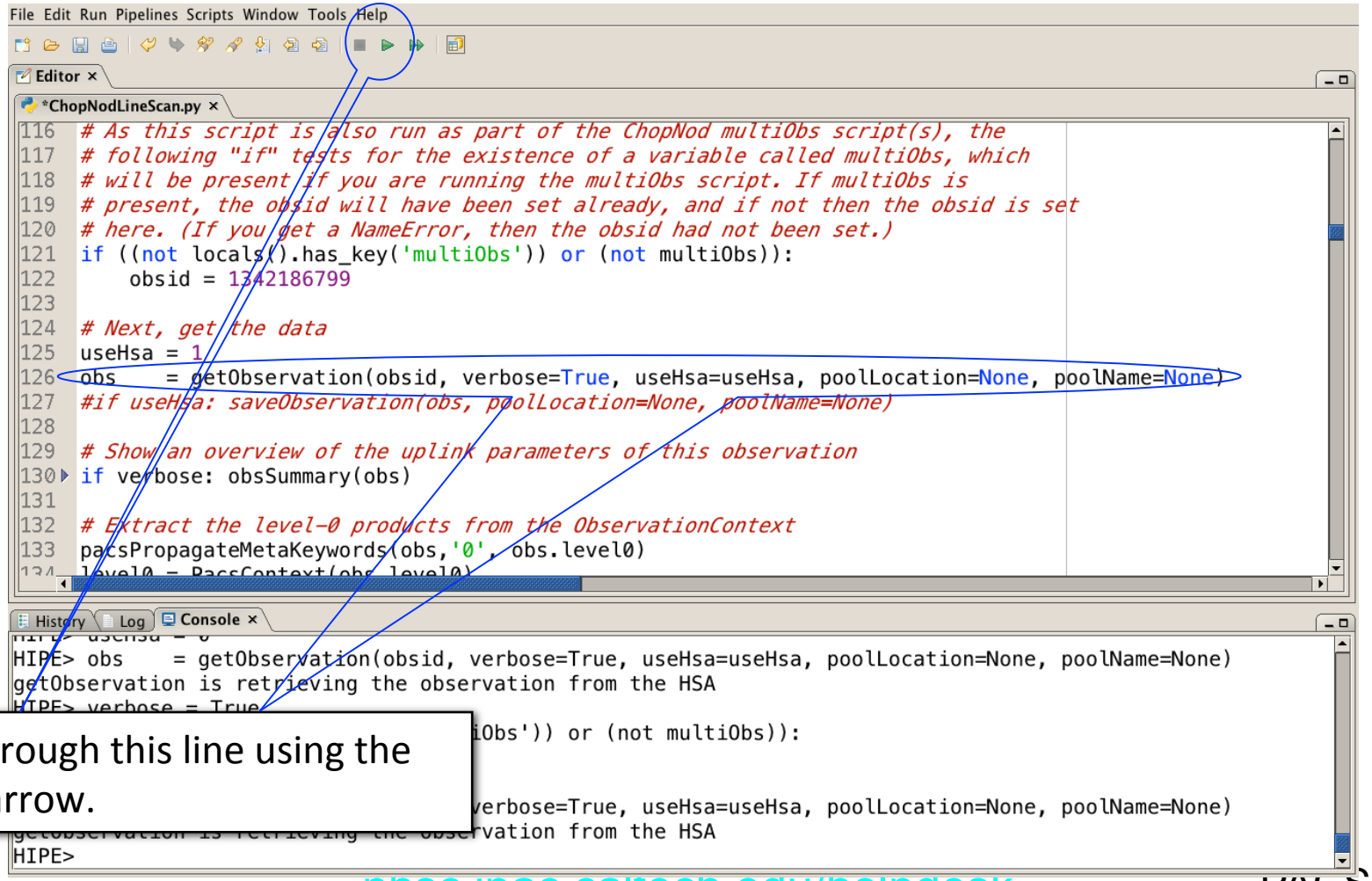
# Loading the observation

If the data is not stored as a local pool, you may need to tell getObservation to acquire the data from HSA. In this case edit the line to useHsa=1

```
116 # As this script is also run as part of the ChopNod multiObs script(s), the
117 # following "if" tests for the existence of a variable called multiObs, which
118 # will be present if you are running the multiObs script. If multiObs is
119 # present, the obsid will have been set already, and if not then the obsid is set
120 # here. (If you get a NameError, then the obsid had not been set.)
121 if ((not locals().has_key('multiObs')) or (not multiObs)):
122     obsid = 1342186799
123
124 # Next, get the data
125 useHsa = 1
126 obs    = getObservation(obsid, verbose=True, useHsa=useHsa, poolLocation=None, poolName=None)
127 #if useHsa: saveObservation(obs, poolLocation=None, poolName=None)
128
129 # Show an overview of the uplink parameters of this observation
130 if verbose: obsSummary(obs)
131
```

# Loading the observation

Next step, we load the observational context ( a structure containing all the observational data, information about them and calibration data).



# The observation summary

```
HIPE> if verbose: obsSummary(obs)
Observation Summary:
    OBSID:          1342186799
    Instrument: PACS
    AOR label:  NearGalPACS-SB-01-blue
    Proposal:   SDP_esturm_3
    Target:     M82
    Redshift:   6.77E-4 (z)
    Purpose:    ---
    Concat.:    ---
    OD:         178
    Start:      2009-11-08T15:32:00.000000 TAI (1636385520000000)
    Duration:   582.0 seconds (incl. spacecraft on-target slew time)

AOT and instrument configuration:
    AOT:        PacsLineSpec
    Mode:       Pointed, Chop/Nod
    Bands:      B3A R1 (prime diffraction orders selected)
    Is bright:  YES (shortened range mode)
    Chopper:    large throw
    Nod cycles: 1

Observation block summary:
 | Name(*) | Camera |   ID | Band(*) | Wave(*) | WaveMin | WaveMax | Repetitions(*) | LineFlux(*) | ContFlux(*) | Width(*) |
Capacitance | OutOfBand |  Channel |
 |         |        |      |         |         |         |         |                |          Jy |          Jy |          |
pF |         |          |
 |   OI 63 |   blue |    2 |     B3A |  63.223 |  63.093 |  63.379 |              3 |    2412.000 |      25.593 |   70.000 |
0.140 |        No |    prime |
 |       - |    red |  102 |      R1 | 189.686 | 189.248 | 190.123 |              3 |           - |           - |        - |
0.140 |        No | parallel |
 |  NIII 57 |   blue |    3 |     B3A |  57.359 |  57.213 |  57.548 |              3 |     485.000 |      23.970 |    0.000 |
0.140 |        No |    prime |
 |       - |    red |  103 |      R1 | 172.112 | 171.594 | 172.629 |              3 |           - |           - |        - |
0.140 |        No | parallel |
(*) = requested in HSPOT

System configuration summary:
    SPG pipeline version:                SPG v11.1.0
    Calibration tree version:            56
    SPG pipeline products creation date: 2014-01-15T20:42:31.725000 TAI (1768509751725000)
```

You may see a warning from obsSummary – it's not a concern but you can rerun with obsSummary(obs, forceUpdate=True)

Prime lines targeted, with line fluxes, continuum levels, etc estimated when the observation was planned

Pipeline and calibration versions used in making the HSA products

# Setting the camera

```
147  # ------------------------------------------------------------------------
148  # SELECT DATA FROM ONE CAMERA
149  # ------------------------------------------------------------------------
150
151  # Red or blue camera ?
152  if ((not locals().has_key('multiObs')) or (not multiObs)):
153      camera    = 'blue'
154
```

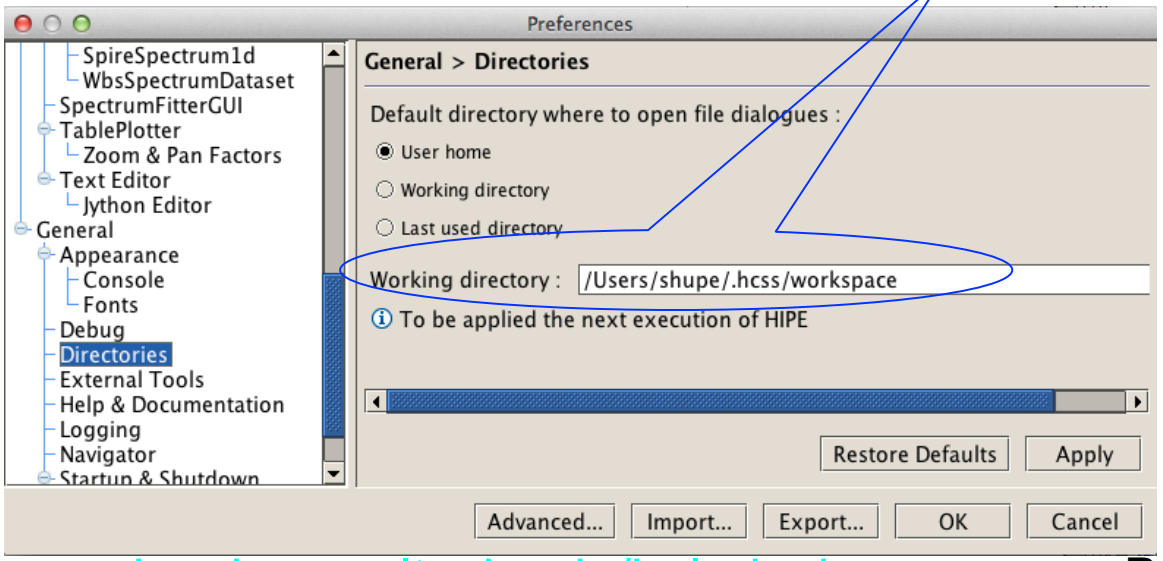
We select camera = 'blue'

After selecting the camera, we can check what camera we selected by simply printing: "print camera"

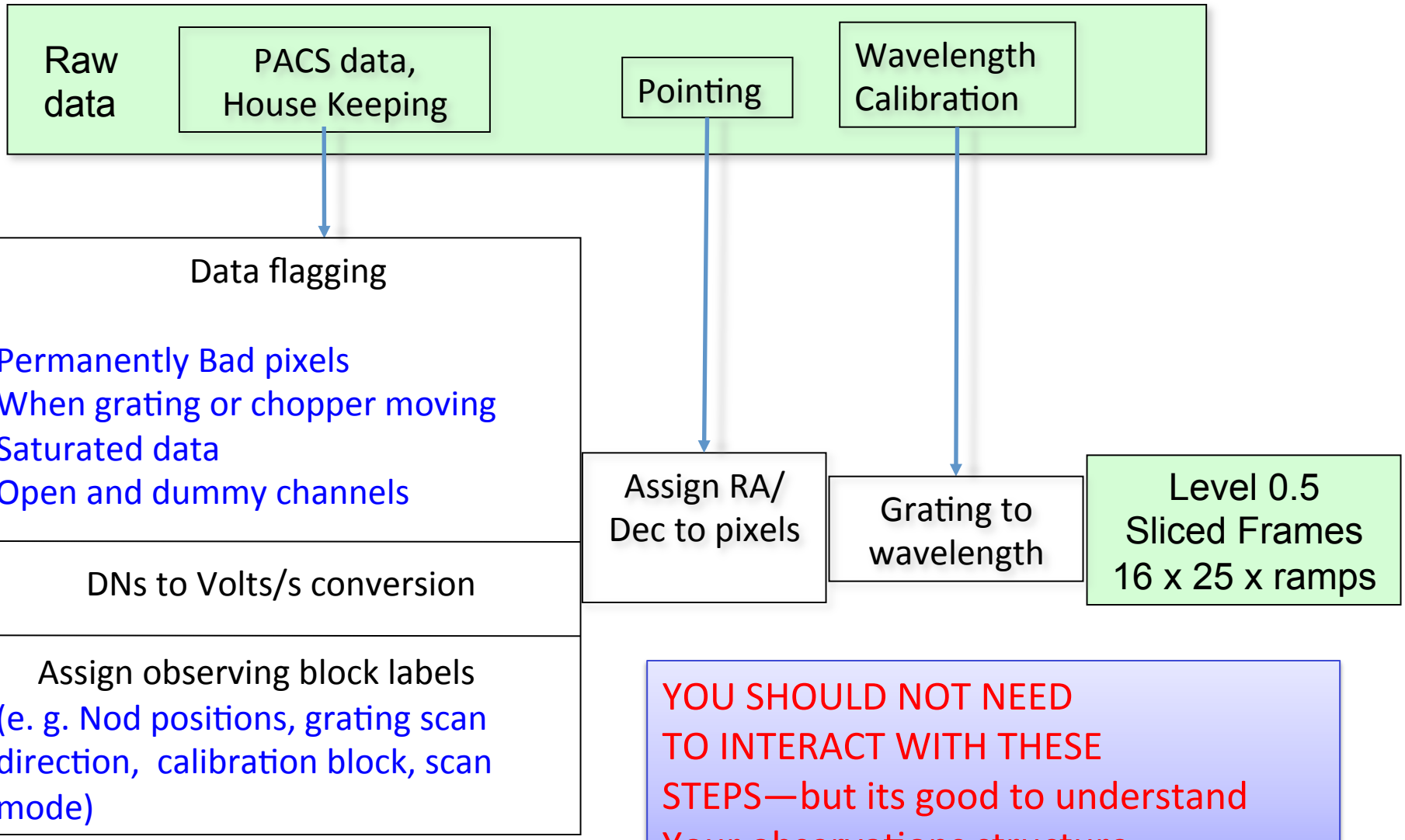
# Setting the output directory

```
178 # saveOutput: False - nothing is saved
179 #              True  - the output directory 'outputDir' will be used to store the
180 #                      products of this pipeline (intermediate and final)
181 # Example: outputDir = "/home/me/Herschel/PacsSpectro/pipelineOutput/"
182 # When saveOutput is True, nameBasis will be used as basis for the filenames of all outputs
183 saveOutput = True
184 outputDir  = str(Configuration.getWorkDir())+"/pacsSpecOut/"
185 if (not os.path.exists(outputDir)): os.mkdir(outputDir)
186 if verbose: print "The products of this pipeline will be saved in ",outputDir
187
188 nameBasis  = str(obsid)+"_"+target+"_"+od+"_Hipe_"+hipeVersion+"_calSet_"+calSet+"_"+camera
```

By default, the script will save intermediate and final products in your HIPE working directory.
You can change the HIPE working directory using Edit -> Preferences -> Directories.



# Level 0 → 0.5

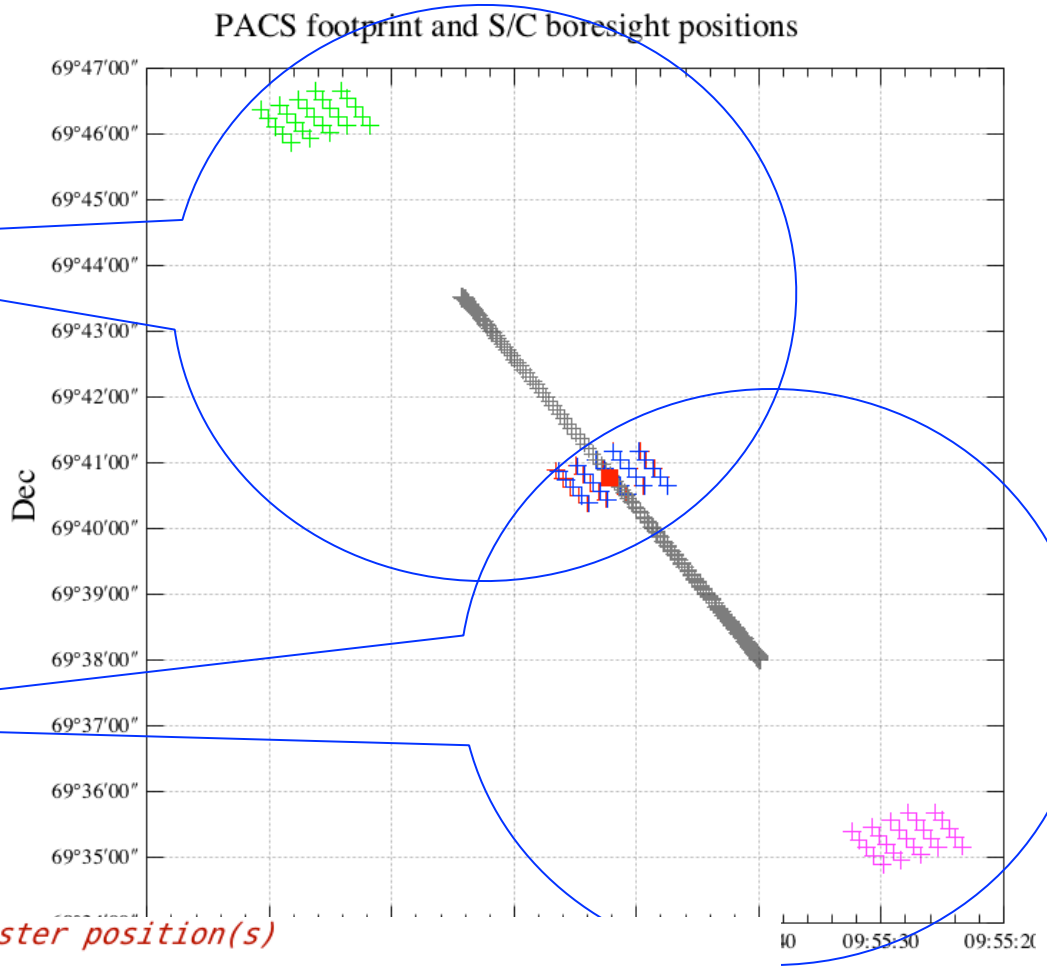



YOU SHOULD NOT NEED TO INTERACT WITH THESE STEPS—but its good to understand Your observations structure

# Plot: footprint

Nod A

Nod B




```
229 # show footprints for selected raster position(s)
230 if verbose:
231     ppoint = slicedPlotPointing(slicedFrames)
232     ppoint.setFrameTitle("slicedPlotPointing - "+str(obsid)+" "+str(camera))
```

# Slicing into nods

SLICING: Done for computational reasons/Memory/Efficiency

The slicing of the data is performed according to rules made explicit in the pipeline. In our example, two lines are observed in two nodding positions. So, we expect 4 slices plus an initial slice containing the calibration block.

```
259  # The internal structure of your data has changed
260  if verbose: slicedSummary(slicedFrames)
261
```

History | Log | Console

```
HIPE> if verbose: slicedSummary(slicedFrames)
noSlices: 5
noCalSlices: 1
noScienceSlices: 4
slice#  isScience  nodPosition   nodCycle  rasterId  lineId      band                    dimensions
 wavelengths      onSource  offSource
0       false       ["B"]          0         0 0       [1]         ["B3A"]                 [18,25,679]
 59.816 - 60.067  no         no
1       true        ["B"]          1         0 0       [2]         ["B3A"]                 [18,25,1019]
 63.093 - 63.379  both       both
2       true        ["A"]          1         0 0       [2]         ["B3A"]                 [18,25,1019]
 63.093 - 63.379  both       both
3       true        ["B"]          1         0 0       [3]         ["B3A"]                 [18,25,1019]
 57.213 - 57.548  both       both
4       true        ["A"]          1         0 0       [3]         ["B3A"]                 [18,25,1019]
 57.213 - 57.548  both       both
```

**Data is sliced early in pipeline: If you have a long observation: many slices—It's a convenient way of making sure all your data is there—compare it with what you expected from HSPOT (Uplink software used to set up the observations)**

5 slices !

Line 1 – B & A nods

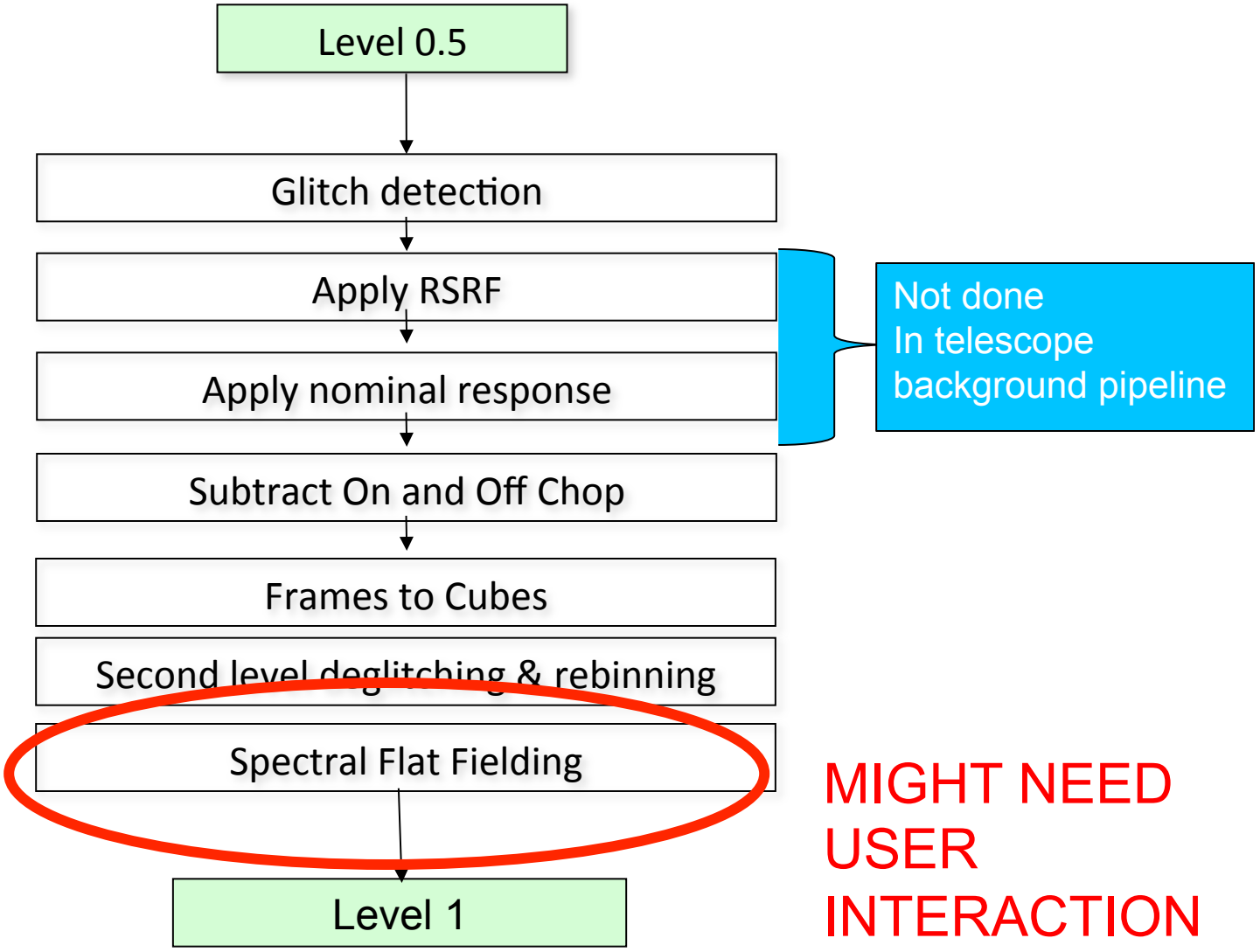
Line 2 – B & A nods

```
259  # The internal structure of your data has changed
260  if verbose: slicedSummary(slicedFrames)
261
```

History | Log | Console ×

```
HIPE> if verbose: slicedSummary(slicedFrames)
noSlices: 5
noCalSlices: 1
noScienceSlices: 4
slice#  isScience  nodPosition   nodCycle  rasterId  lineId      band
 wavelengths      onSource  offSource
0       false       ["B"]         0         0 0       [1]         ["B3A"]
 59.816 - 60.067  no         no
1       true        ["B"]         1         0 0       [2]         ["B3A"]
 63.093 - 63.379  both       both
2       true        ["A"]         1         0 0       [2]         ["B3A"]
 63.093 - 63.379  both       both
3       true        ["B"]         1         0 0       [3]         ["B3A"]
 57.213 - 57.548  both       both
4       true        ["A"]         1         0 0       [3]         ["B3A"]
 57.213 - 57.548  both       both
```

# Level 0.5 → 1




MIGHT NEED
USER
INTERACTION



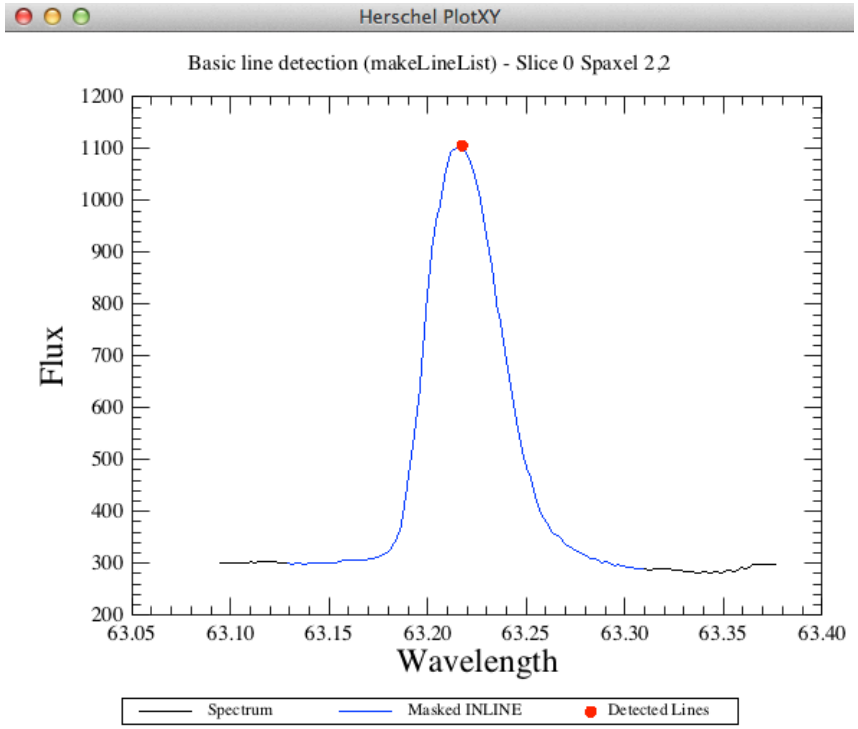
nhsc.ipac.caltech.edu/helpdesk

Spectral Flat field is one area where reprocessing may help IF LINE IS FAINT OR HAS ABSORPTION…

Why? Because flat field assumes that there is a line in the spectrum, and it tries to mask out the line before finding the level of the continuum. If there is no line, it can accidentally find a noise spike as the line….

So its better to tell the software WHERE the line is expected to be. You should input the expected OBSERVED wavelength

# Check: Spectral FlatField




For faint lines or known absorp. lines add OBSERVED wavelengths

slicedCubesMask = maskLines(slicedCubes,slicedRebinnedCubes, lineList=[], wid…….

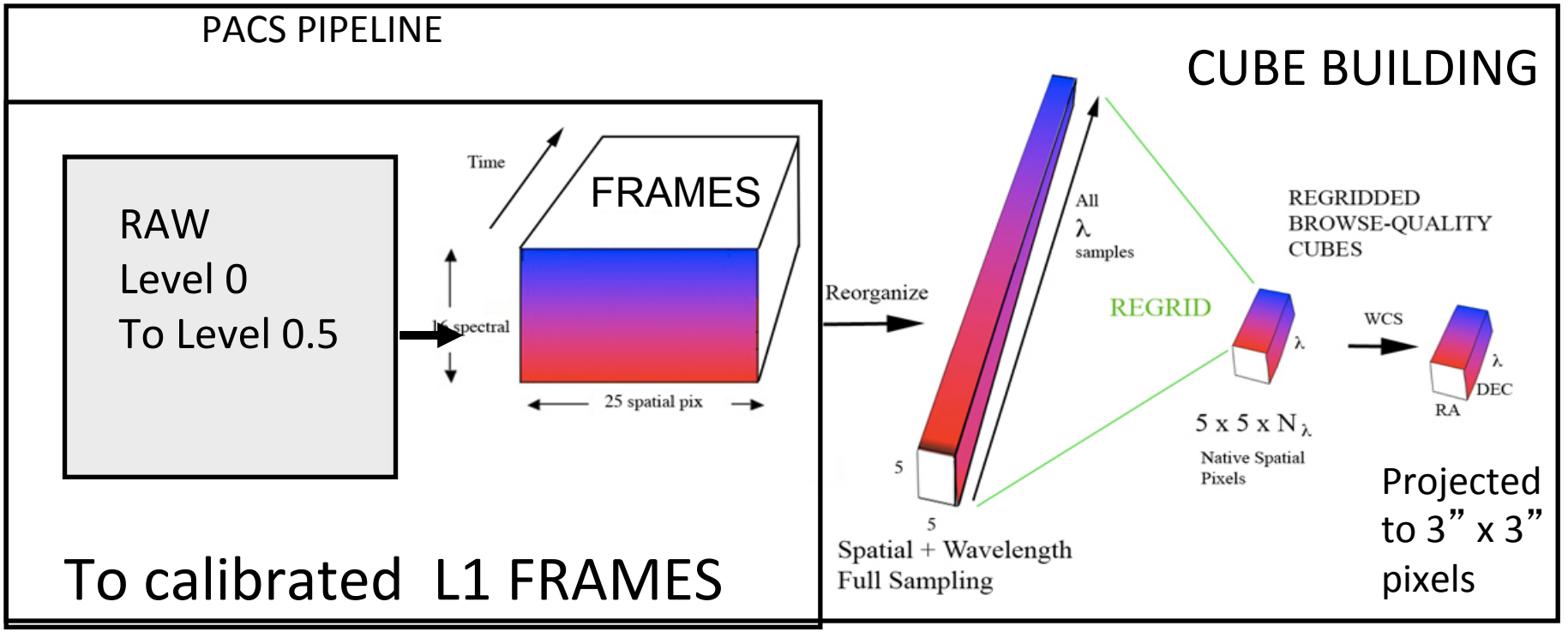
As a default, the code will search for lines in all the pixels and then mask them before computing the spectral flat field.
It is possible to give directly the list of lines to be masked via the parameter lineList = [63.227], for instance.
This user-specified lineList is usually needed *only* for absorption lines OR VERY FAINT LINES

# This concludes processing to level 1

```
351
352 # 3. Actual spectral flatfielding
353 # slopeInContinuum is a boolean. Set it to true for lines existing on a continuum with a significant
354 slopeInContinuum = 1
355
356 slicedCubes = specFlatFieldLine(slicedCubesMask, scaling=1, copy=1, maxrange=[50.,230.], slopeInConti
357
358 # 4. Rename mask OUTLIERS to OUTLIERS_B4FF (specFlagOutliers would refuse to overwrite OUTLIERS) & de
359 slicedCubes.renameMask("OUTLIERS", "OUTLIERS_B4FF")
360 slicedCubes = deactivateMasks(slicedCubes, String1d(["INLINE", "OUTLIERS_B4FF"]))
361
362 # 5. Remove intermediate results
363 del waveGrid, slicedRebinnedCubes, slicedCubesMask
364
365 # --- End of Spectral Flat Fielding
366
367 # -----------------------------------------------------------------------------
368 #         Processing      Level 1 -> Level 2
369 # -----------------------------------------------------------------------------
370 #
```

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 — Level 1

REBINNED CUBE — PROJECTED CUBE — Level 2

# Summary of Level 1 Products

HIPE> slicedSummary(slicedFrames)
noSlices: 4
noCalSlices: 0
noScienceSlices: 4

THIS EXAMPLE CLEARLY "CAMERA = BLUE"

| 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

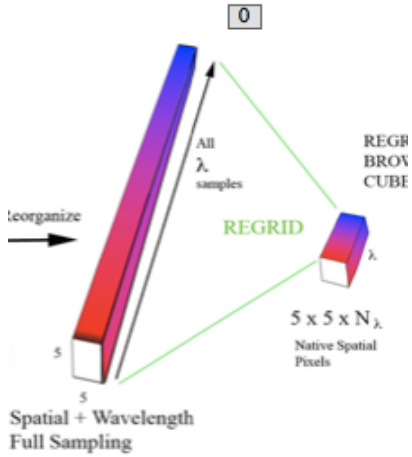
HIPE> slicedSummary(slicedCubes)
noSlices: 4
noCalSlices: 0
noScienceSlices: 4

| 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.

# NOW WE CONVERT THE LARGE "PACSCUBE" INTO A MORE COMPACT "REBINNED" CUBE

**THERE WILL BE MANY OF THESE –ONE FOR EACH POINTING, REPETITION AND NOD**




At this point the data volume goes down dramatically
IMPORTANT—WAVELENGTH GRID NOT UNIFORM LIKE HETRODYNE DATA

# Creating a wavegrid:

SOMEHOW WE HAVE TO RE-BIN THE POINTS IN THE WAVELENGTH DIMENSION INTO SMALLER "BINS"

THIS IS GOVERNED BY TWO PARAMETERS

OVERSAMPLE UPSAMPLE

WHICH IN PRINCIPAL COULD BE CHANGED
HOWEVER—DEFAULTS ARE FINE--

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=4,
                        calTree = calTree)
```

DEFAULT VALUES

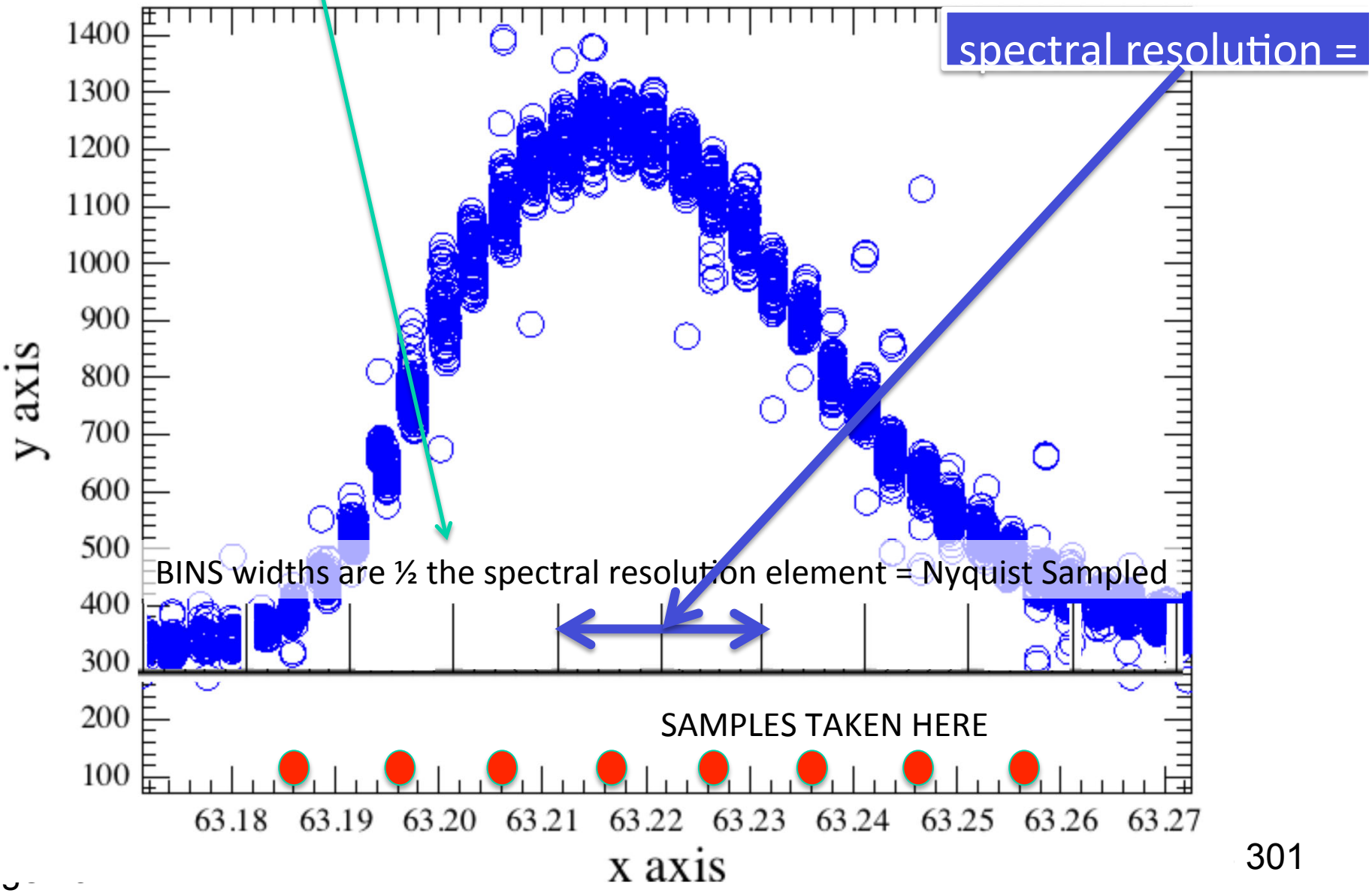
Illustrative examples follow….

# Examples of Oversample and Upsample Parameter Pairs

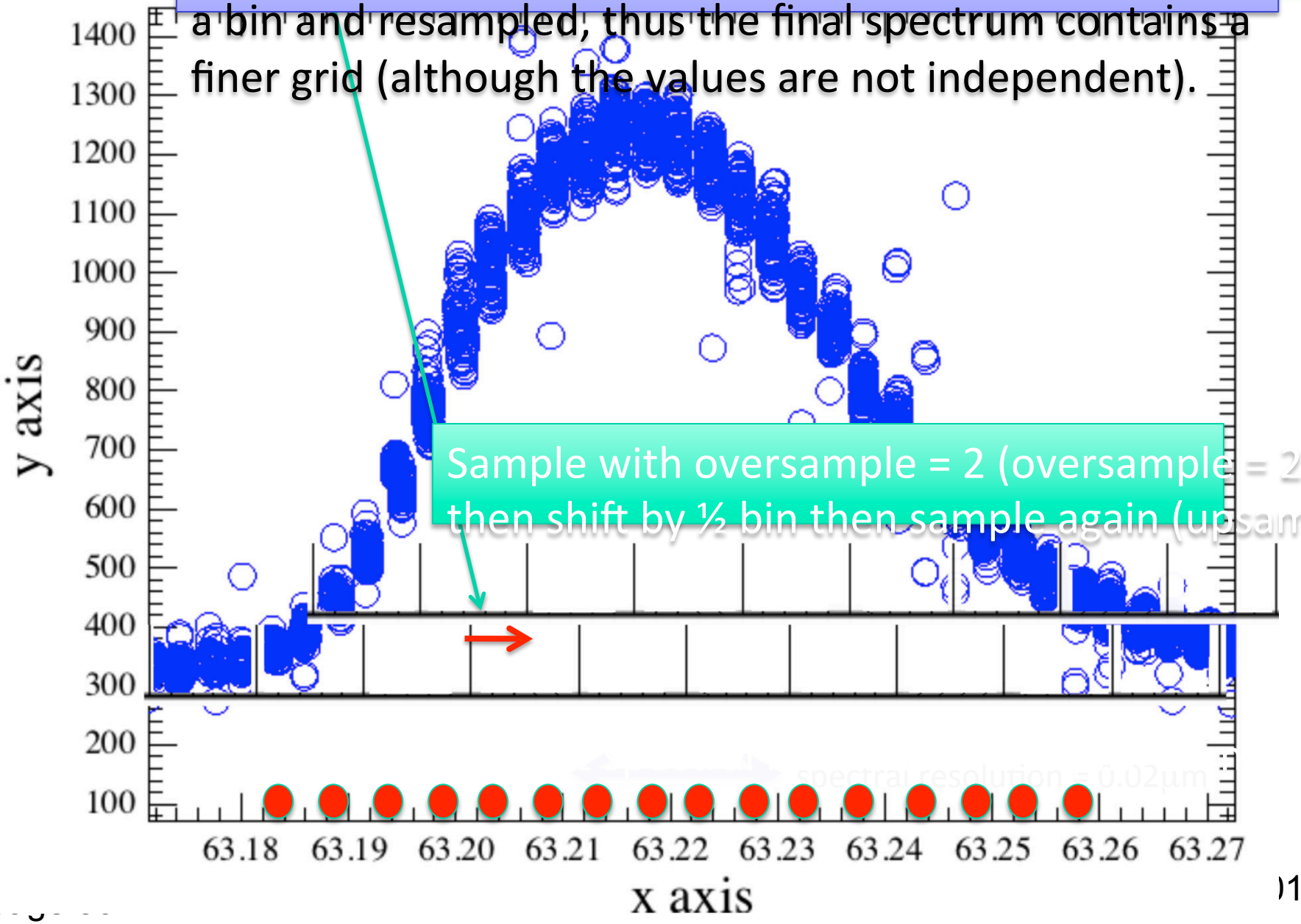
EXAMPLE: Oversample = 2
Upsample = 1
bins = ½ spectral resolution
at that wavelength

# OVERSAMPLE = 2
# = NyQUIST SAMPLING

NHSC PACS Web Tutorial



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).




Sample with oversample = 2 (oversample = 2) then shift by ½ bin then sample again (upsample

Default values for these parameters
in the pipeline are
Oversample = 2, Upsample = 4

So the individual samples will not be completely independent

# Before Regridding the Wavelength Dimension—

We apply OUTLIERS rejection flagging to the spectra. This process is quite robust and is found to work well for PACS spectra.

It does not remove data—but creates FLAGS which are then used to filter out bad points

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([str(i) for i in slicedCubes.get(0).maskTypes \
                if i not in ["INLINE", "OUTLIERS_B4FF"]]), exclusive = True)
# Flag the remaining outliers, (sigma-clipping in wavelength domain)
slicedCubes = specFlagOutliers(slicedCubes, waveGrid, nSigma=5, nIter=1)
```

specFlagOutliers CREATES MASK CALLED “OUTLIERS”

GENERALLY DEFAULTS ARE GOOD—
NO USER INTERACTION
NEEDED

# Now Create the Rebinned Cube

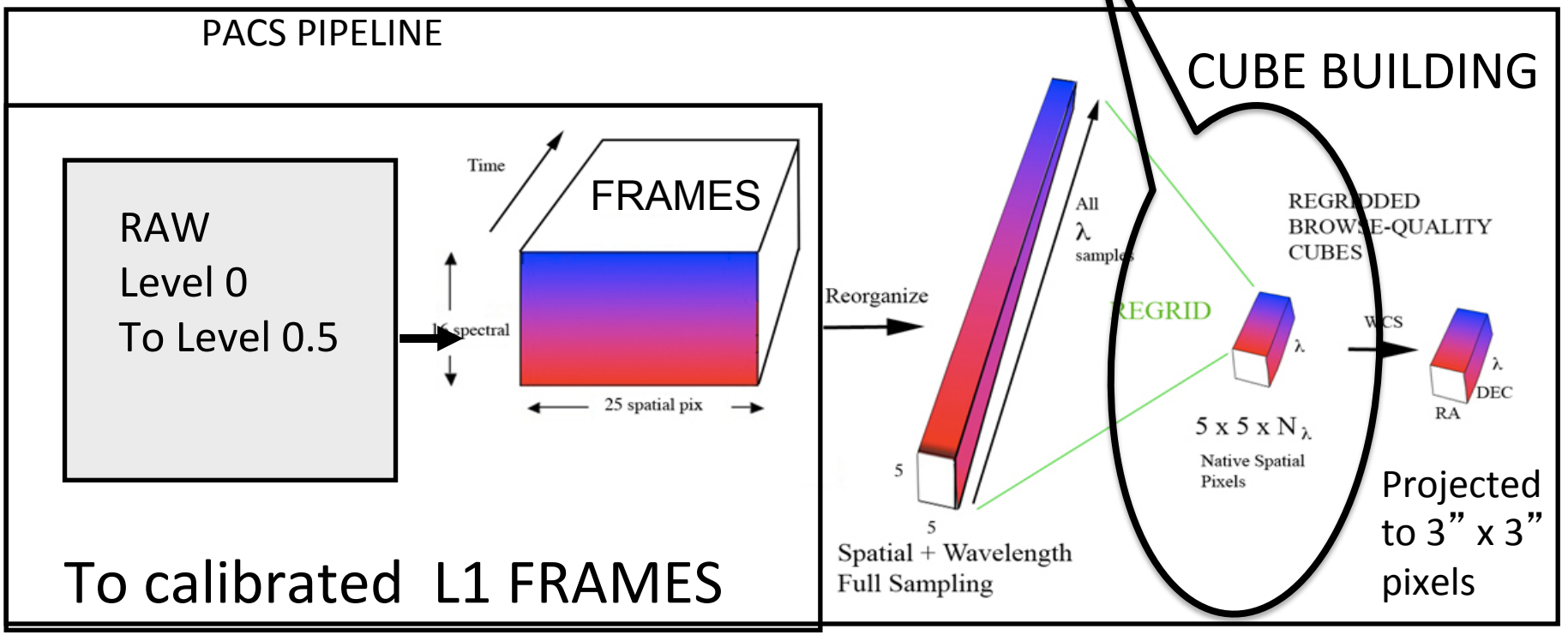
We activate the OUTLIER mask for the first time. This happens before rebinning

```
# Activate all masks
slicedCubes = activateMasks(slicedCubes, String1d([str(i) for i in slicedCubes.get(0).maskTypes \
    if i not in ["INLINE", "OUTLIERS_B4FF"]]), 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)

REMEMBER THAT ALL OUR VARIABLE ARE "SLICED"
But you don't need to be concerned with this—it's just
A computational/organizational issue.. slicedRebinnedCubes is simply a container of all the rebinned cubes…

This creates 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



# 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)
```

NOTE THAT IN LEVEL 2 PRODUCTS IN SPG (AUTOMATIC) PIPELINE THESE PRODUCTS ARE CALLED  RENINNED CUBES

# Summary of PACS spectrometer archived products

((From Roberta's Talk yesterday)

- Level 0:

  HPENG: engineering
  HPGENHK: housekeeping
  HPSDMCB/HPSDMCR: raw DecMec (blue/red)
  HPSFITB/HPSFITR: Herschel PACS Spectrometer Fitted Blue/Red (→ "Signal" in /0)
  HPPHK: housekeeping
  HPSRAWB/HPSRAWR: raw data (blue/red)

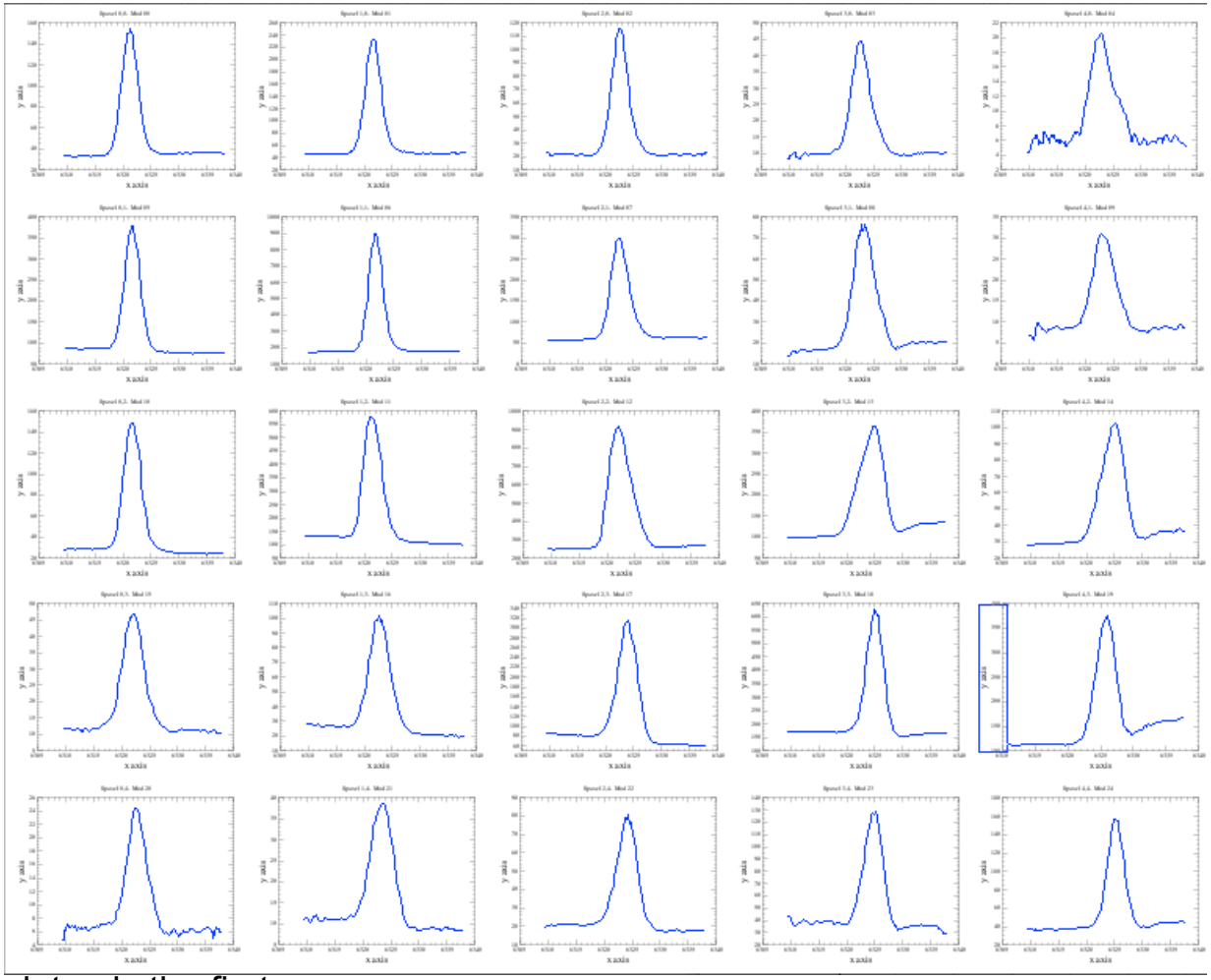
- Level 1:

  HPS3DB/HPS3DR: Herschel PACS Spectrometer 3D Spectral Cube Blue/Red
  (→ "flux", "wave", "ra", "dec" in, e.g., /0: L2 N1 B R(0 0))
  HPSCALB/HPSCALR: calibration (response/dark)
  HPSFITB/HPSFITR: fitted slopes

- Level 2:

  HPS3DPB/HPS3DPR: Herschel PACS Spectrometer 3D Projected Cube Blue/Red
  (→ "image", "coverage" in, e.g., /0: L2 N1 A R(0 0))
  HPS3DRB/HPS3DRR: Herschel PACS Spectrometer 3D Rebinned Cube Blue/Red
  (→ "image", "ra", "dec", "stddev", "exposure", "wavegrid" in, e.g., /0: L2 N1 R(0 0)

- Level 2.5: (only for range un-chopped): on and off positions are in different OBSIDs. In Level 2.5, the off is subtracted from the on

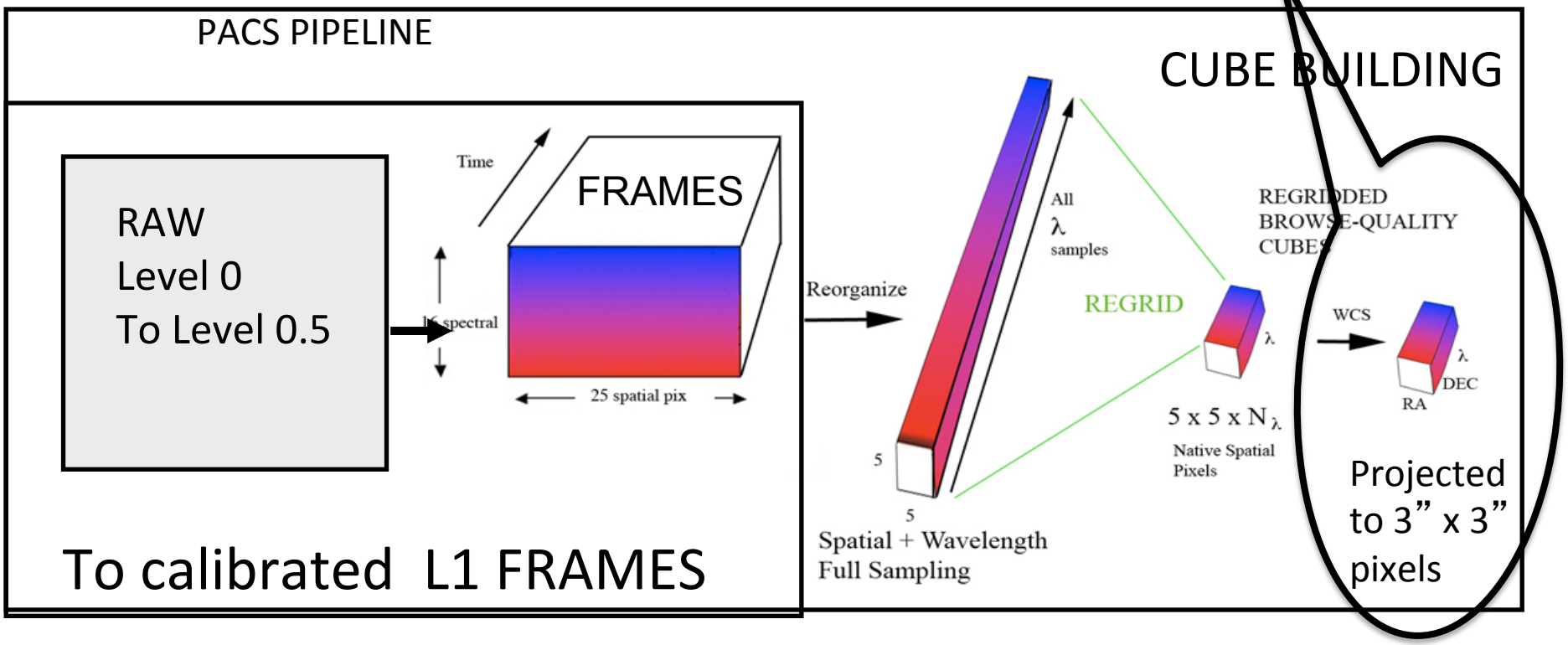


Here we plot only the first slice. In the case we have set the lineId=2, so we have only the [OI] line.

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

```
434     # 5x5 plot of one of the rebinned cubes (one li
435     slice = 0
436     p55 = plotCube5x5(slicedFinalCubes.get(slice))
437     p55.setFrameTitle(str(obsid)+" "+camera+": slic
```

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

IF YOU DO NOT HAVE A RASTER OBSERVATION:

IN HIPE 12—YOUR FINAL PRODUCTS ARE THE REBINNED CUBES..

For “mapped” observations, you have two choices:

a) specProject > Projects data onto WCS in simple way
b) Drizzle> For VERY WELL SAMPLED DATA THIS CAN IMPROVE THE FIANL MAP

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

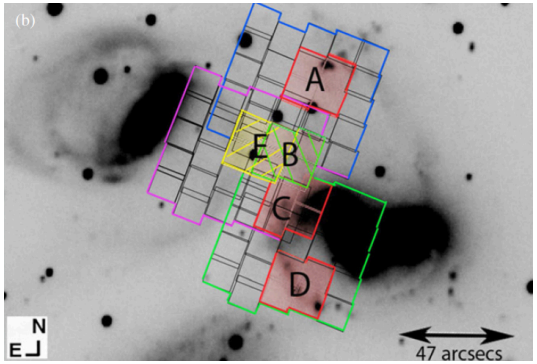
# Invoking specProject

## Use only when you have well sampled maps

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

# Which final Chop/Nod product should I use?

Single pointings or sparse mapping
> Hipe 12.1 PacsRebinned Cube




Extract from rebinned
By selection of spaxels

>HIPE 13—new projection task
called specInterpolate will make better interpolation—good for "sparse" maps with reasonable redundancy

Well-sampled mapping
> Hipe 12.1 SpecProject cubes (has wcs) wavelength channels are of variable width
> Hipe 13 Will create cube with "constant" wavelength channels (ds9 readable λ)

VERY Well-sampled maps > Drizzle maps (Careful….)

# SPECIAL CASE OF KNOWN BRIGHT POINT SOURCE

HERE WE DON"T CARE ABOUT THE MAP BUT
WE CAN USE THE SPAXEL DISTRIBUTION TO CORRECT
THE CENTRAL SPAXEL FOR POINTING
OFFSETS and APPLY APERTURE CORRECTIONS

**RUN LAST PART OF THE iPIPE TO**
**CREATE 3-flavors of "central spectrum"**

# Correcting and Extracting Point Sources

Caveats: The instructions following 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])**.

extractCentralSpectrum will return three spectra:

a) Spectrum of the central pixel corrected for the part of the beam that the central spaxel misses. This is known as the **c1** spectrum.
b) Spectrum of the sum of the central 9 pixels, with the (much smaller) correction for the part of the beam that this "superspaxel" misses: the **c9** spectrum.
c) Spectrum of the central pixel, with the "3x3" correction that compares the ratio of **c1** to **c9** to the ratio expected from the beam profile; this is intended to correct for small mis-pointings and is the **c129** spectrum ('c one-to-nine', get it?). To trust this correction the source continuum must be of order 10 Jy.

Discussions of these corrections are found in the PACS Data Reduction Guide: Spectroscopy section 3.7.3.

# Extracting the central spaxel and returning three spectra with various corrections

```
for slice in range(len(slicedFinalCubes.refs)):
    if verbose: print
    # a. Extract central spectrum, incl. point source correction (c1)
    # b. Extract SUM(central_3x3 spaxels), incl. point source correction (c9)
    # c. Scale central spaxel to the level of the central_3x3 (c129 -> See PDRG & print
                extractCentralSpectrum.__doc__)
    c1,c9,c129 = extractCentralSpectrum(slicedFinalCubes, slice=slice, smoothing='median', isLineScan=-1,
                 calTree=calTree, verbose=verbose)
    #
    # Save to Fits
    if saveOutput:
        name = nameBasis + "_centralSpaxel_PointSourceCorrected_Corrected3x3NO_slice_"
        simpleFitsWriter(product=c1,file = outputDir + name + str(slice).zfill(2)+".fits")
        name = nameBasis + "_central9Spaxels_PointSourceCorrected_slice_"
        simpleFitsWriter(product=c9,file = outputDir + name + str(slice).zfill(2)+".fits")
        name = nameBasis + "_centralSpaxel_PointSourceCorrected_Corrected3x3YES_slice_"
        simpleFitsWriter(product=c129,file = outputDir + name + str(slice).zfill(2)+".fits")
```

Some plots are produced but are not shown here. Since the example in this tutorial is not a point source, the plots are not helpful to discuss at this point.



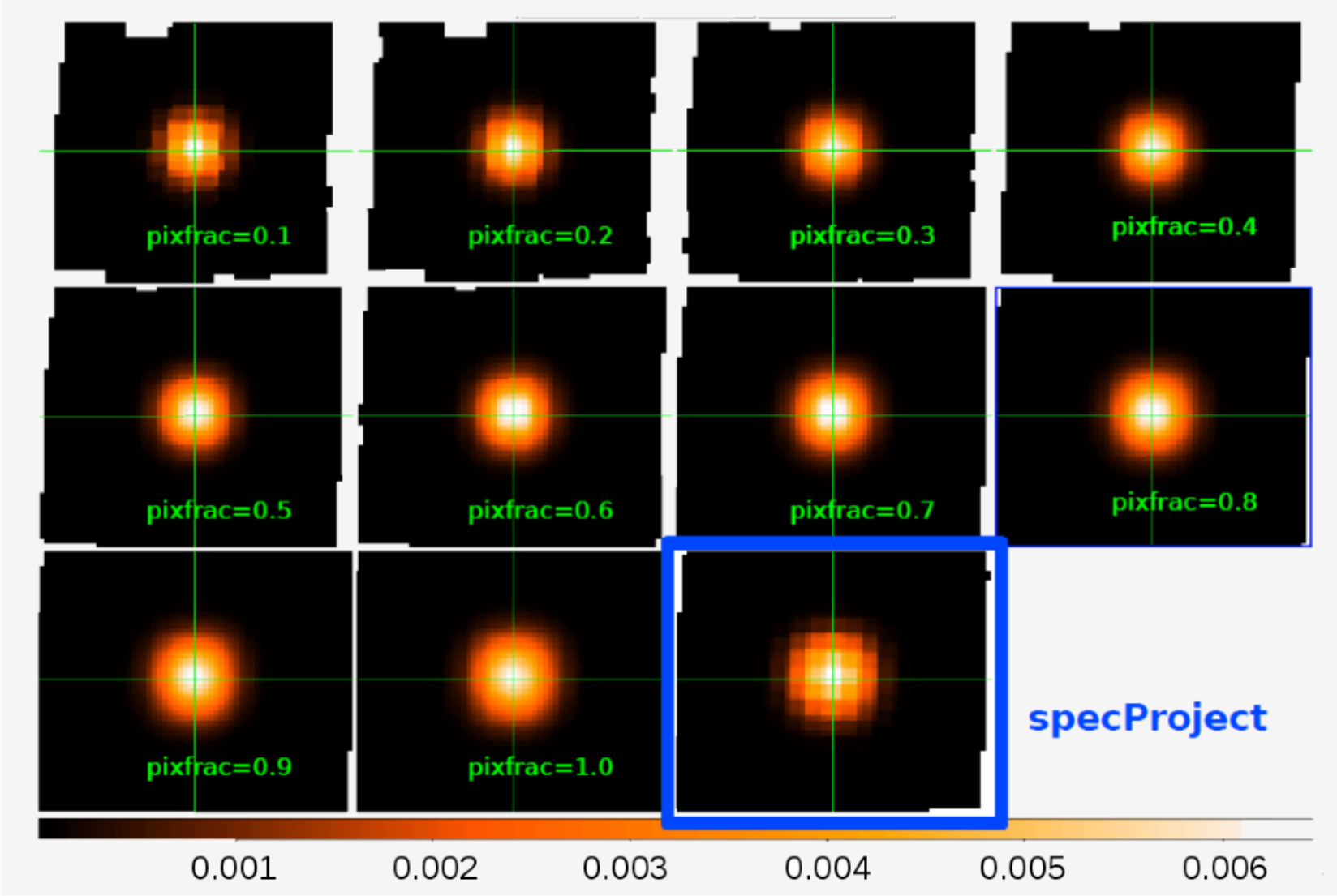
# Drizzling=VERY EXPENSIVE IN COMPUTER RESOURCES

**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.***

(Primary drizzle document: http://www.stsci.edu/ftp/science/hdf/combination/drizzle.html)

For super-sampled data—will produce excellent results



# Invoking Drizzling (slicedDrizzled Cubes)

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

```
  Drizzling works from the not-rebinned cubes
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)[0]
```

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

# Invoking Drizzling (WARNING)
# ONLY USE THIS IF YOU HAVE VERY WELL SAMPLED DATA: MOST IS NOT!!

Warning: In Pipeline 12 ipipe – 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.