



# **NHSC/PACS Web Tutorials**

## **Running PACS photometer pipelines**

### **PACS-201 (for Hipe 13.0)**

*Level 1\* to Level 2 processing:  
The High-Pass Filter pipeline*

Prepared by Kevin Xu  
May 2015



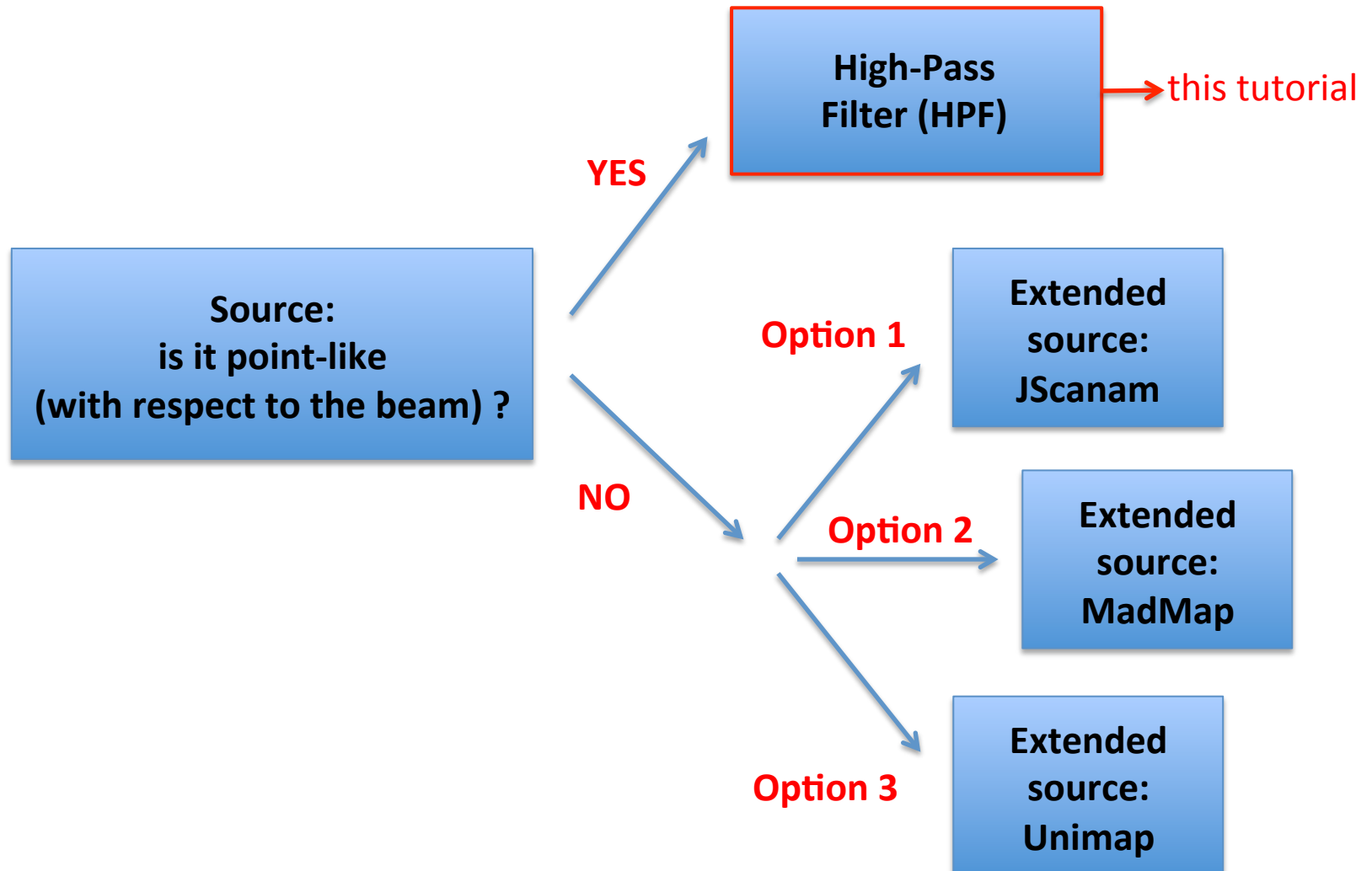
**NOTE:** between HIPE 12 and HIPE 13, there is no change in *scanmap\_Pointsources\_Photproject* ipipe script. Therefore, this tutorial differs from that for the same script in HIPE 12 only in a few updates (in slide 4 & slide 7).



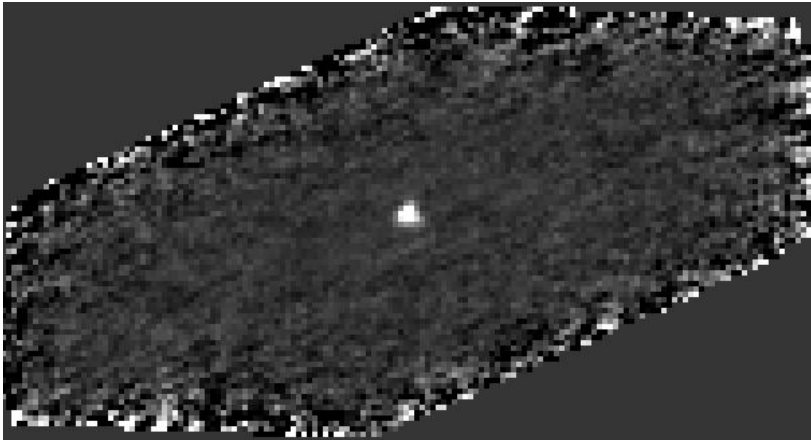
# Structure of this tutorial

- Slide 4 to 37: philosophy and step-by-step walk-through for the PACS photometer High-Pass Filter pipeline
- Slide 38 and 39: Deglitching
- Slide 40: High-Pass Filter radius
- Slide 41: outpix & pixfrac
- Slide 42: turnaround removals

# PACS Photometer Pipeline: 2 main branches



The HPF branch is optimal for reducing mini scan maps or large scan maps where the focus of the science is on point sources

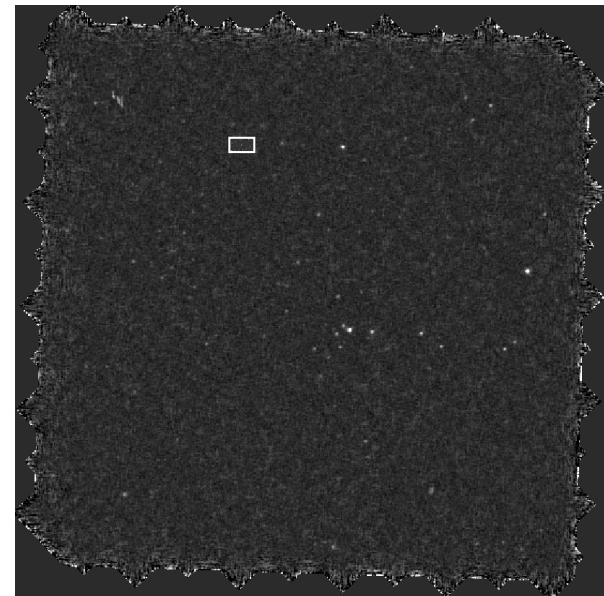


← **Mini scan map**

a single source in the center of the field

**Large scan map** →

multiple sources distributed in the field

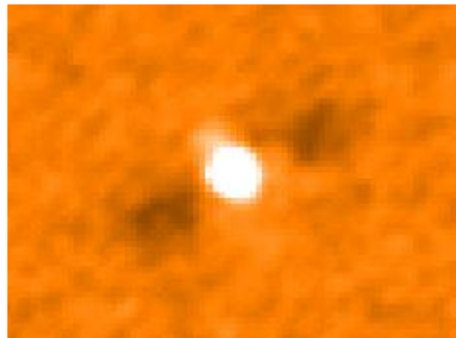
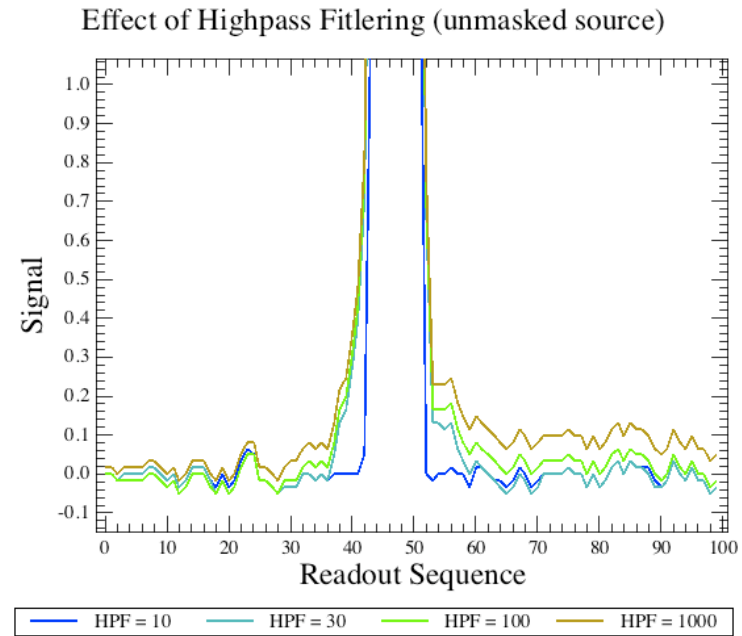


# The High-Pass Filter concept



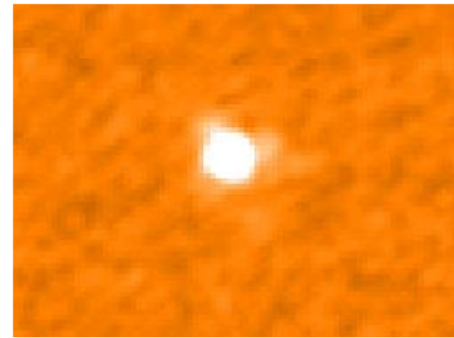
**Main Idea:**  
sliding median-filter on individual pixel timelines to remove large scale drifts

**Note:** When a bright source enters the filter box, it alters the estimate of the median and thus the drift removal



Unmasked Highpass Filtering

Sources  
have to be  
masked !



Masked Highpass Filtering

# The ipipe script for HPF processing is: scanmap\_Pointsources\_PhotProject



The screenshot shows the HIPE software interface. The main window is titled "HIPE - /Volumes/Seagate/Herschel/hcss/hcss.dp.pacs-13.0.4852/scripts/pacs/scripts/ipipe/phot/scanmap\_pointsources\_PhotProject.py". The "Pipelines" menu is open, showing a tree structure: PACS > Photometer > Scan map and minimap > Point sources PhotProject > scanmap\_PhotProject. The "Editor" window displays the script content, which includes copyright information and a detailed description of the script's purpose and usage. The script text is as follows:

```
3  # Copyright 2001-2013 Herschel Science Ground Segment
4  #
5  # HCSS is free software: you can redistribute it and/or modify
6  # it under the terms of the GNU Lesser General Public License as
7  # published by the Free Software Foundation, either version 3 of
8  # the License, or (at your option) any later version.
9  #
10 # HCSS is distributed in the hope that it will be useful,
11 # but WITHOUT ANY WARRANTY; without even the implied warranty of
12 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13 # GNU Lesser General Public License for more details.
14 #
15 # You should have received a copy of the GNU Lesser General
16 # Public License along with HCSS.
17 # If not, see <http://www.gnu.org/licenses/>.
18 #
19 #
20 # This ipipe script processes scan map and mini-scan map observations
21 # containing mostly point-like or relatively small extended sources.
22 # starting from Level 1 (calibrated data cubes in Jy/detector pixel)
23 # and combining several (crossed) observations.
24 # This script makes use of an iterative high-pass filtering of the
25 # timelines to remove the 1/f noise by masking the source(s)
26 # This processing is not adequate for extended sources
27 # as it removes large-scale structures, which cannot be
28 # properly protected (masked) from the filtering.
29 #
30 # nifty add-ons:
31 # - for faint sources, there is the possibility to inject the ra, dec of the source
32 # for the mask and the photometry such that the correct source is properly masked
33 # and the aperture is centered at the required position
34 #
35 # The source fitting function is fully described in the simple aperture photometry script
36 # that can be found in the useful scripts
37 #
38 # Bruno Altieri, Vanessa Doublier, Javier Gracia
39 # September 2013, for HIPE 12
40 ..
```

The interface also shows a "Tasks" panel on the right with options: Applicable, By Category, and All. Below that is a "Variables" panel with a "No filtered" dropdown and sections for "Observations" and "Other Data". At the bottom, there are tabs for "History", "Log", and "Console".



## *Level 1\* to Level 2 processing:*

**NOTE:** Starting with HIPE 11, the PACS photometer *ipipe* script for the High-Pass Filter branch starts from Level 1 (calibrated cubes) instead of Level 0 (*raw data*).

This is because the pipeline is now very stable between Level 0 and Level 1 and the user does not need to tweak the processing at the level of raw calibrated data.





# HPF Pipeline: Main Concept

The main idea of the HPF pipeline revolves around generating a mask as accurate as possible to “protect” the source/s of interest when the high-pass filtering (which allows the removal of the noise) is applied.

This operation is done in multiple (3) steps.



# Structure of the ipipe script:



The script is organized in 3 + 1 major blocks:

➤ **Part 1**

→ **1<sup>st</sup> pass:** for each OBSID, a mask is generated by statistically identifying sources as “peaks” above the (median) background of the scan. HPF is then applied with this mask

→ **2<sup>nd</sup> pass:** all the individual maps are combined together. A mask is built from these combined maps

➤ **Part 2**

HPF is re-applied to each OBSID (e.g. Level 1) with this improved mask. Deglitching is also applied

➤ **Part 3**

the mask generated in Part 1 and 2 is updated by using direct information on the coordinates of the source/s of interest. The mask is added to the ones previously generated, and HPF is re-applied (e.g. to Level 1 data) using this final, global mask

➤ **Photometry:** at the end of the script, aperture photometry is performed on the final map (→ THIS PART OF THE SCRIPT IS NOT TREATED IN THIS TUTORIAL)



# Part 1 – 1<sup>st</sup> pass

(from line 111 to 244)

Generate a mask for **each OBSID** and use this for 1<sup>st</sup> pass with HPF

- ✓ the Level 1 data of each OBSID are loaded into HIPE
- ✓ the calTree, which contains all the calibration files, is loaded into HIPE
- ✓ HPF is run on the Level 1 data without masking the sources
- ✓ a map is generated from these 1<sup>st</sup> pass HPF data
- ✓ the high-coverage pixels of this map are used to identify outliers. These outliers are the “sources” which allow the generation of a mask
- ✓ this mask is applied to the data before re-running HPF
- ✓ a map from the 2<sup>nd</sup> pass HPF data is created



## At the beginning of Part 1, the user has to set:

- target name, e.g. M31: **object** (line # 52)
- OBSID numbers: **obsids** (line # 54)
- band name, i.e. blue (70  $\mu\text{m}$ ), green (100  $\mu\text{m}$ ) or red (160  $\mu\text{m}$ ): **camera** (line # 57)
- working directory, i.e. directory where to store generated maps: **direc** (line # 70)
- prefix (including target name and camera) of files to generate: **fileroot** (line # 71)
- turnaround removals: **lowScanSpeed/highScanSpeed** or **limits** (line # 90, 91 and 92)

## ... and then decide:

- Improve on 2<sup>nd</sup> level deglitching: **doIndLevelDeg** (line # 78, see slide 38/39)
- whether to do (aperture) photometry as well as processing: **doPhotometry** (line # 79)
- do 2-d gaussian fit to determine source centroid and improve mask: **doSourceFit** (line # 80)
- get coordinates of the source/s from external file to improve mask: **fromAFile** (line # 82)
- If previous line set to 'True', enter filename: **tfile** (line # 83)



## At this stage, the script starts looping over the list of OBSIDs:



1. The Level 1 data of each OBSID, *i*, in the “*obsids*” list, is loaded into HIPE. First load the observation context:

```
Console x
HIPE> obs.append(getObservation(obsids[i], useHsa=True, instrument="PACS"))
```

2. Load the Calibration Tree (*calTree*):

```
Console x
HIPE> calTree.append(getCalTree(obs=obs[i]))
```

The Calibration Tree (*calTree*) contains all the files necessary to process your data

### 3. then extract the Level1:

Syntax for Blue/Red array

```
Console x  
HIPE> frames=obs[i].level1.refs["HPPAVGB"].product.refs[0].product
```

**HPPAVGB/R: Herschel PACS Photometer AVGerage Blue/Red**  
This is the signal downloaded from the spacecraft after on-board averaging

Now that we have the basic pieces, we can get to the core of Part 1, i.e.:

the generation of the mask for each OBSID



At this stage the mask is created **blindly**. This means that the assumption is that one does not know the location (coordinates) of the sources, and these have to be identified as “peaks” above the median background.

## Let's see how this works:

4. First apply high-pass filtering without a mask
5. Then create a preliminary map
6. ....next slide

For more information on "hpfradius" see slide # 40

```
Console x
HIPE> frames = highpassFilter(frames, hpfradius, interpolatedMaskedValues=True)
HIPE> ....
HIPE> map, mi = photProject(frames, pixfrac=pixfrac, outputPixelSize=outpixsz, calTree=calTree[i])
```

**NOTE:** This map is NOT good for photometry:  
It must be used *\*only\** for identifying sources

For more information on "pixfrac" and  
"outputPixelSize" see slide # 41



6. Identify the region of the map with high coverage (i.e. coverage > “med”)
7. Use this region to estimate the signal standard deviation of the map (stdev)
8. Set a threshold (i.e. 3.0) above which map outliers are identified
9. ....next slide

```
Console x
HIPE > med=MEDIAN(map.coverage[map.coverage.where(map.coverage > 0.)])
HIPE > index = map.image.where((map1.coverage > med) & (ABS(map.image) < 1e-2))
HIPE > signal_stdev=STDDEV(map.image[index])
HIPE > threshold=3.0*signal_stdev
```

9. Mask everything above the threshold → **these are the “sources”**
10. Save the mask in a .fits file
11. Mask in the timeline all readouts at the same coordinates of the map pixels with signal above the threshold

The *frames* are saved in standard fits format. The saved file can be read back into HIPE:  
HIPE> frames = simpleFitsReader(maskfile)

```
Console x
HIPE > mask=map.copy()
HIPE > mask.image[mask.image.where(map.image > threshold)] = 1.0
HIPE > mask.image[mask.image.where(map.image < threshold)] = 0.0
HIPE > mask.image[mask.image.where(map.coverage < 0.5*med)] = 0.0
HIPE > simpleFitsWriter(mask, fileRoot + "_" + str(obsids[i]) + "_mask_firstStep.fits" )
HIPE > frames, outMask = photReadMaskFromImage(frames, si=mask,maskname="HighpassMask",
extendedMasking=True, calTree=calTree[i])
```

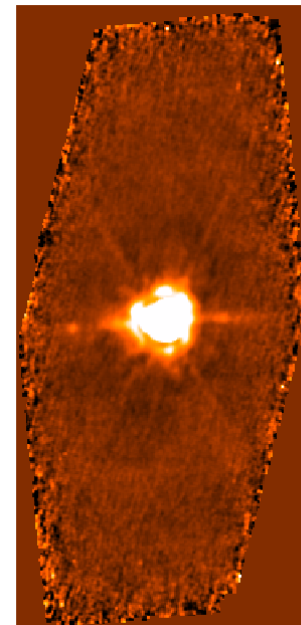
**NOTE:** since HIPE 12.0, every task having more than one output has this syntax. In this case, the two outputs of the task are the updated frames and the mask

This is how a mask is “attached” to the Level 1 frames !

9. now that you have a mask, re-run high-pass filtering with it
10. create map from high-pass filtered data
11. save map in a .fits file

```
Console X
HIPE > frames = highpassFilter(frames,hpfradius,maskname="HighpassMask", interpolateMaskedValues=True)
HIPE > .....
HIPE > map, mi = photProject(frames,pixfrac=pixfrac,outputPixelSize=outpixsz,calTree=calTree[i])
HIPE > simpleFitsWriter(map, fileRoot + "_" + str(obsids[i]) + "_map_firstStep.fits")
```

Map from an individual OBSID – 1<sup>st</sup> pass





# Part 1 – 2<sup>nd</sup> pass

(from line 213 to 244)



Derive a mask from the **combined maps**

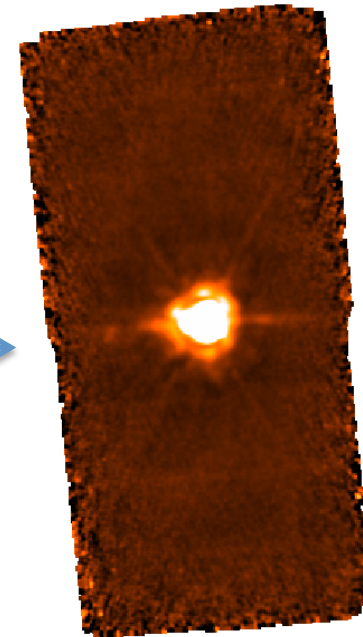
- ✓ the maps generated in Part 1 are read-in, co-added
- ✓ the new combined map is used to generate an improved mask, following the same procedure as in Part 1 for the individual OBSID maps: the high-coverage pixels of this map are used to identify outliers, and these outliers are the “sources” which allow the generation of a mask

# 1. First, loop over the OBSIDs and co-add them:

**NOTE:** in Jython the loop is denoted with an indentation

```
Console x
HIPE > for i in range(len(obsids))
HIPE >     ima=simpleFitsReader(file=fileRoot + "_" + str(obsids[0]) + "_map_firstStep.fits")
HIPE >     images.add(ima)
HIPE > mosaic=MosaicTask()(images=images, oversample=0)
```

Map from  
combined OBSIDs





**2. Then generate a mask using the combined map.**

**The procedure is the same as in Part 1 (slide # 17 to 19).**

**These steps are not repeated in the tutorial, but in summary:**

- Identify the region of the map with high coverage (i.e. coverage > “med”)
- Use this region to estimate the signal standard deviation of the map (stdev)
- Set a threshold above which map outliers are identified

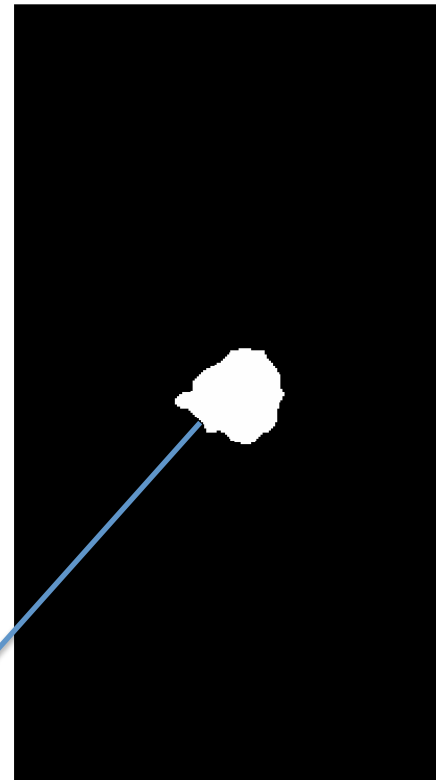
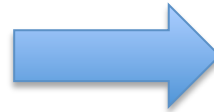
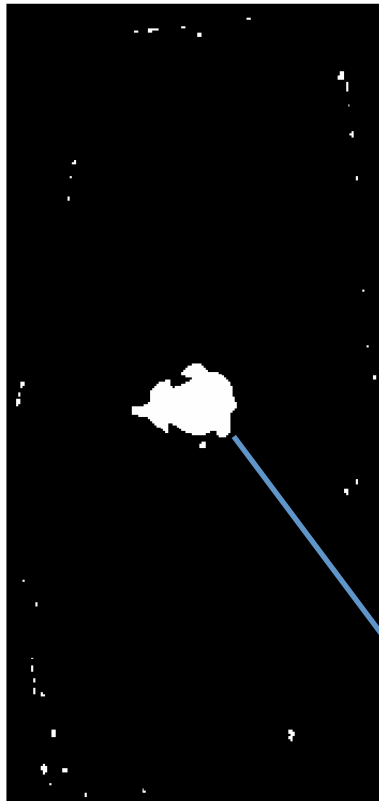
**....and then:**

- Mask everything above the threshold → **these are the “sources”**
- Save the mask in a .fits file
- Mask in the timeline all readouts at the same coordinates of the map pixels with signal above the threshold

**Note:** in Part 2, the threshold is set to **2.0** instead of **3.0** as in Part 1

Mask from Part 1:  
**One for each** OBSID

Updated Mask from Part 2:  
**combined** OBSIDs



source



## Part 2

(from line 251 to 310)

**Now step back, i.e. apply the newly improved mask to do HPF on individual OBSIDs**

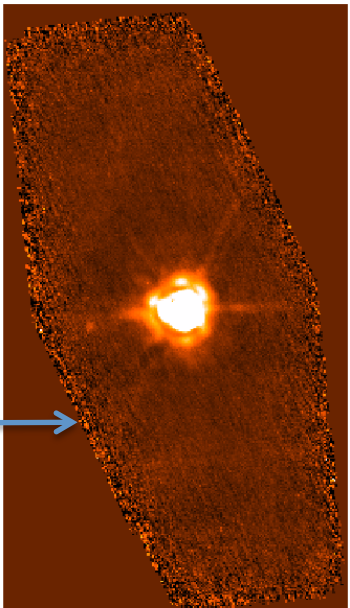
- ✓ With the improved mask derived from the combined maps, step back to Level 1 data of each OBSID: apply HPF using the new mask, and create the map
- ✓ co-add again the individual maps



# 1. Step back to Level 1 data for each OBSID and apply HPF with improved mask

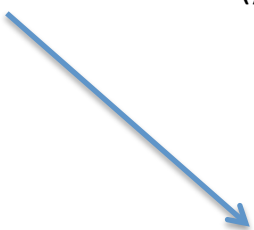
```
Console X
HIPE > for i in range(len(obsids))
HIPE > ....
HIPE > frames, outMask = photReadMaskFromImage(frames, si=mosaicMask, maskname="HighpassMask",
        extendedMasking=True, calTree =calTree[i])
HIPE > frames = highpassFilter(frames,hpfradius,maskname="HighpassMask", interpolateMaskedValues=True)
HIPE > ....
HIPE > map, mi = photProject(frames, pixfrac=pixfrac ,outputPixelsize=outpixsz,
        calTree=calTree[i])
HIPE > simpleFitWriter(map, fileRoot + "_" + str(obsids[i]) + "_map_secondStep.fits")
```

Map from individual OBSID – 2<sup>nd</sup> pass

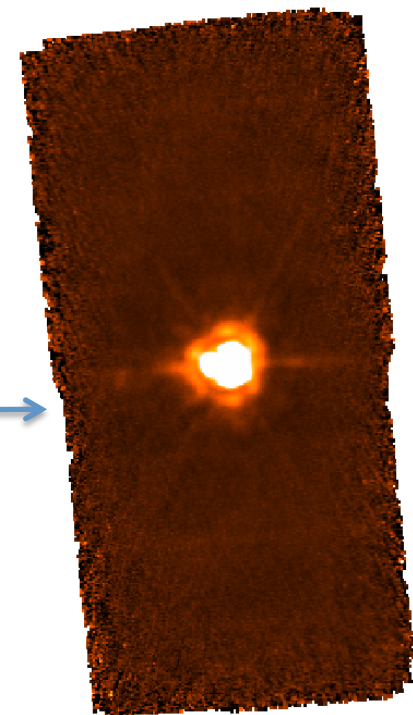


## 2. Now co-add the 2<sup>nd</sup> pass HPF maps from individual OBSIDs

```
Console X
HIPE > images=ArrayList()
HIPE > for i in range(len(obsids)):
HIPE >   ima = simpleFitsReader(file=fileRoot + "_" + str(obsids[i]) + "_map_secondStep.fits")
HIPE >   images.add(ima)
HIPE > mosaic=MosaicTask()(images=images, oversample=0)
```



Map from  
**combined OBSIDs** - from 2<sup>nd</sup> pass





# Part 3

(from line 313 to 437)

**After generating a “blind” mask in Part 1 and 2, the user can now try to improve on that mask by using the information on the location of the sources (e.g. coordinates), if this information is available.**

- ✓ To use the known coordinates of the sources, the script provides two options:
  - A. The user inputs these coordinates (e.g. from a file) and a 2-d gaussian procedure is used to “refine” the input source/s coordinates. The fitted centroids are then used to generate a mask;
  - B. The input coordinates (from a file or from the metadata) are used to directly generate a mask at these locations;



# Option A: 2-d gaussian fitting

To use this option, `doSourceFit` (line # 80) has to be set to `True`

Then:

- ✓ Read-in the file with the input coordinates
- ✓ For each input source, create a postage stamp using a given “cropsize”
- ✓ do a 2-d gaussian fit on each postage stamp
- ✓ use fitted “rasource” and “decsource” to generate a mask

**NOTE:** option A also includes the alternative case (line # 350 to 353) in which the user, with no a-priori information on the location of the sources (i.e. no input file), performs a “blind” 2-d gaussian image on the map (e.g. mosaic2, see slide # ) and uses the fitted centroids to generate a mask.

# Option A: in practice..

## 1. Input source file:

```
Console x
HIPE> tlist,ralist,declist=readTargetList(tfile)
```

## 2. For each source, convert ra/dec into pixel coordinates:

```
Console x
HIPE> pixcoor = mosaic.wcs.getPixelCoordinates(ralist[i],declist[i])
```

## 3. Define boundaries of postage stamp in pixel coordinates: r1, r2, c1, c2. The postage stamp size is defined by **“cropsize”** (default = 20 pixels), e.g:

```
Console x
HIPE> r1 = int(pixcoor[0]-cropsize/2.)
```

#### 4. Create postage stamp:

```
Console x
HIPE > cmap = crop(image=mosaic,row1=int(pixcoor[0]-cropsiz/2.), \
row2=int(pixcoor[0]+cropsiz/2.), \
column1=int(pixcoor[1]-cropsiz/2.) \
column2=int(pixcoor[1]+cropsiz/2.))
```

#### 5. For each postage stamp, do a 2-d gaussian fit:

```
Console x
HIPE> sfit = mapSourceFitter(cmap)
```

#### 6. Get the coordinates centroid, ra/dec, from the fit:

```
Console x
HIPE > rasource[i] = Double([sfit["Parameters"].data[1]])
HIPE > decsource[i] = Double([sfit["Parameters"].data[2]])
```



# Option B: Source coordinates from file or metadata

In this case, **doSourceFit** (line # 80) has to be set to **False**

Then:

- ✓ If input catalog file is provided, read-in the file with the input coordinates
- ✓ use the input coordinates to generate a mask

Or:

- ✓ get source/s coordinates from metadata
- ✓ use these coordinates to generate a mask

# Option B: in practice..

→ If input catalog file is available:

```
Console x
HIPE> tlist,ralist,declist=readTargetList(tfile)
HIPE > rsource = Double1d(ralist)
HIPE > decsource = Double1d(declist)
```

→ alternatively, if source/s coordinates are read-in from metadata:

```
Console x
HIPE > tlist = String1d(1, obs[0].meta["object"].value)
HIPE > rsource = Double1d(1, obs[0].meta["ra"].value)
HIPE > decsource = Double1d(1, obs[0].meta["dec"].value)
```



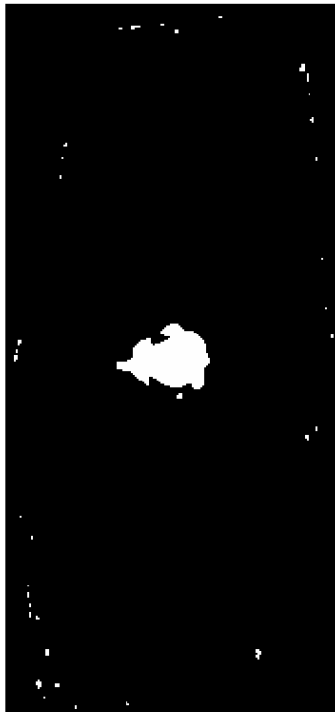
## In common to option A and B:

→ With the source/s coordinates (“rasource”, “decsource”) – either from 2-d gaussian fit/s or direct input/s from catalog/metadata – the new mask can now be generated:

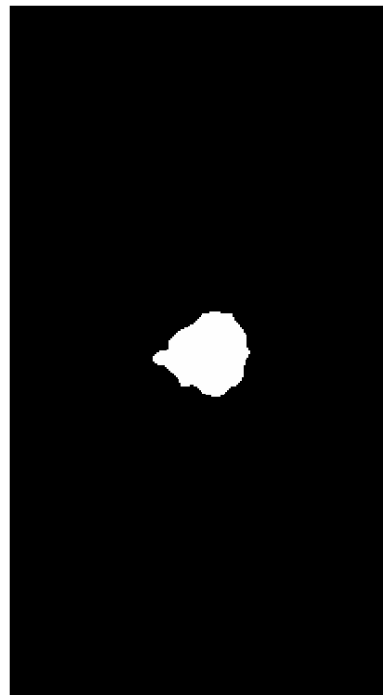
```
Console X
HIPE > radius = 20.
HIPE > combinedMask = mosaicMask.copy()
HIPE > mfc = MaskFromCatalogueTask()
HIPE > combinedMask = mfc(combinedMask,rasource,decsource,Double1d(len(rasource),radius),copy=1)
```

Centered on each source position (ra and dec), create a circular “patch” (i.e. a mask) with a 20” radius. The patch is big enough to fully cover the source in all bands.

Mask from Part 1:  
from **each** OBSID



Updated Mask from Part 2:  
from **combined** OBSIDs



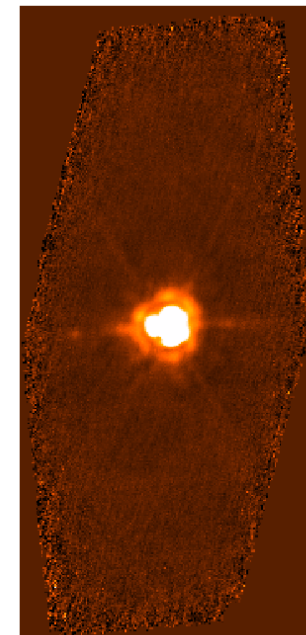
Updated Mask from Part 3:  
From **known** source/s coords.



→ Now step back and apply this new mask (improved version of the mask from Part 1, 2 and 3) to do HPF on the Level 1 data of each OBSID:

```
Console X
HIPE > for i in range(len(obsids)):
HIPE > ...
HIPE > frames, outMask = photReadMaskFromImage(frames, si=combinedmask, maskname="HighpassMask",
extendedMasking=True,calTree = calTree[i])
HIPE > frames = highpassFilter(frames,hpfradius,maskname="HighpassMask",interpolateMaskedValues=True)
HIPE > ....
HIPE > map, mi = photProject(frames, pixfrac=pixfrac,outputPixelSize=outpixsz,
calTree=calTree[i])
HIPE > simpleFitsWriter(map, fileRoot + "_" + str(obsids[i]) + "_finalMap.fits")
```

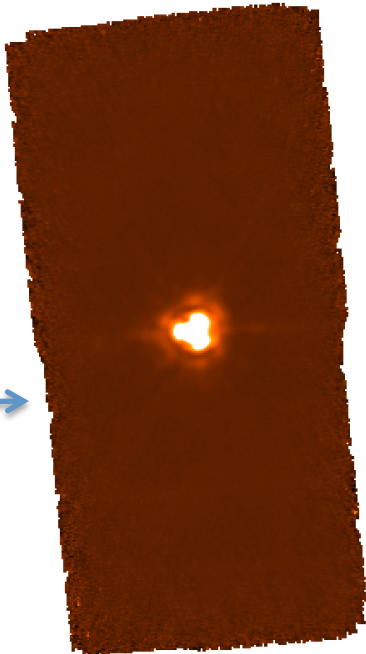
Map from  
individual OBSID – 3<sup>rd</sup> pass



→ now co-add the 3<sup>rd</sup> pass HPF maps from individual OBSIDs: this is the FINAL map !

```
Console X
HIPE > images=ArrayList()
HIPE > for i in range(len(obsids)):
HIPE >   ima = simpleFitsReader(file=fileRoot + "_" + str(obsids[i]) + "_finalMap.fits")
HIPE >   images.add(ima)
HIPE > .....
HIPE > mosaic=MosaicTask()(images=images,oversample=0)
HIPE > simpleFitsWriter(mosaic, fileRoot + "_finalMosaic.fits")
```

Map from **combined OBSIDs** -  
3<sup>rd</sup> pass (e.g. mosaic)





**This concludes the walk through for  
the PACS photometer HPF pipeline !**



# Deglitching

## (more information in tutorial # 402)

- At the beginning of the script (line # 78), the user has to set the switch `doIndLevelDeg` to either True or False;
- If `doIndLevelDeg` is set to `True`, then deglitching is performed again with a threshold for outliers detection more aggressive (15) with respect to threshold (30) applied before Level 1 in the standard SPG pipeline. You may want to follow this approach when the final mosaic still contains many glitches
- Note that, in the script, deglitching is performed in Part 2 (line # 257 to 263)

## 2<sup>nd</sup> Level Deglitching

**Spatial (2<sup>nd</sup> Level) Deglitching** identifies glitches by exploiting spatial redundancy.  
This is how it works:

```
Console X
HIPE > s = Sigclip(10, 15)
HIPE > s.behavior = Sigclip.CLIP
HIPE > s.outliers = Sigclip.BOTH_OUTLIERS
HIPE > s.mode = Sigclip.MEDIAN
HIPE > mapDeglitch(frames, algo = s, deglitchvector="timeordered", calTree=calTree[i])
```

Outliers are detected with a **sigma-clipping** algorithm and flagged as glitches. Both **positive** and **negative** outliers are detected. By default, outliers are detected with respect to the **median**.



# High-Pass Filter Radius: hpfradius



## Default Values (line # 161 to 164):

```
HIPE > If camera == 'blue':  
HIPE >   hpfradius = 15  
HIPE > else:  
HIPE >   hpfradius = 25
```



These values (units: readouts) allow removal of  $1/f$  noise while preserving as much as possible the flux in the wings of the PSF (Point Spread Function)

**NOTE:** the values above are optimized for point-sources. If the source of interest is slightly extended (a few times the fwhm), the use of larger “hpfradius” is recommended.



# Making the map: outpixsz & pixfrac

The **photProject** task performs a simple co-addition of the images using the drizzle method (Fruchter and Hook, 2002, PASP, 114, 144). The key parameters are the output pixel size and the drop size (**pixfrac**). **A small pixfrac value can help to reduce the correlated noise due to the projection.**

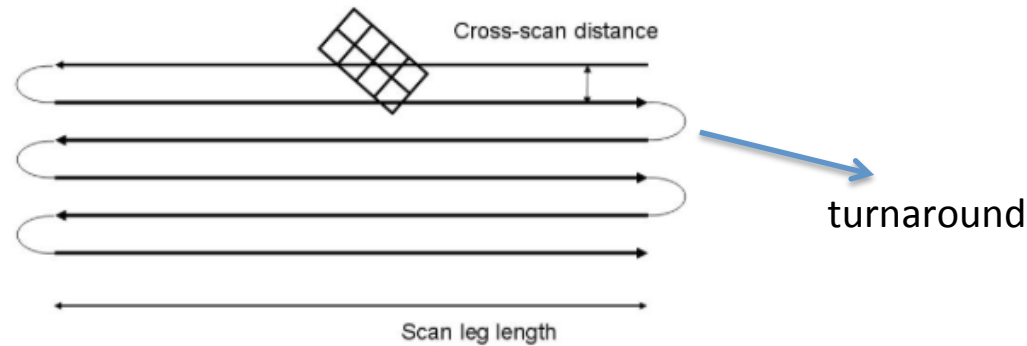
```
Console x
HIPE > map=photProject(frames, pixfrac = pixfrac, outputPixelSize=outpixsz, calTree=calTree[i])
```

## Default Values (line # 104 to 108):

```
HIPE > If camera == 'blue':
HIPE >   outpixsz = 1.0
HIPE >   pixfrac = 0.1
HIPE > else:
HIPE >   outpixsz = 2.0
HIPE >   pixfrac = 0.1
```

# Turnaround removal

**NOTE:** in the script, the frames corresponding to the telescope turnaround are removed each time before creating the map. Turnaround frames are characterized by much lower coverage (hence sensitivity) than normal science frames.



```
Console x
HIPE > frames = filterOnScanSpeed(frames, limit=limits)
```

Turnaround frames are executed at different scan speed than normal science frames

The parameter "limit" is set at the beginning of the script (line # 92).  
"Limits = 10" means: remove all the frames with a scan speed 10% higher/lower than science frames scan speed.



**Thank you !**