Users

SPIRE IA will support both expert users in the ICC and consortium, and the general astronomer in the community. It is anticipated that all users will run the same IA system, with differences for different types of user being different configurations and privileges.

Users are not required to buy commercial software licences in order to run IA.

Access to Data

The normal access to data is expected to be by downloading it over the network from an HCSS node. It should be transparent to the user exactly where the data is located. All related data (e.g. calibration objects) have to be transferred as well. A request is expected to immediately initiate the data transfer.

Security considerations imply that the transfer for normal users will probably have to use the HTTP/HTTPS protocol or its successor. The architecture should allow different protocols to be used, so that, for example, an expert ICC user could run IA as a database client.

Data products produced by a pipeline at the HSC could also be transferred. Calibration objects retrieved with the data have to be the correct ones pertaining both to the version of IA being used and the date of the data being processed. It is possible that the processing time range might span more than one version of an object.

After downloading data, it will be possible for the user to save it to his or her personal file store. It will also be possible to copy this data onto other devices (e.g. laptop, mobile phone etc.) and continue processing from there. The user is not required to maintain a continuous network connection.

History

Data processed by IA will have a “memory” of what has been done to it. The IA will inform the user if a processing step is impossible or inappropriate given the previous processing. There are cases when the IA must refuse to perform the processing step. If it is *possible* to allow the processing step, then this should take the form of a warning that the user may choose to ignore.

This history mechanism will contain all the information required to allow the same processing to be applied to the same input data. In this way, any errors can be reproduced in a controlled fashion. In some cases (e.g. manual deglitching) the history could contain a significant amount of data. The user should not therefore (normally) be presented with all the data maintained by the history, but be able to view an easily readable log of the processing steps that have been applied.

It is also possible that the history of one product could be applied to a different dataset in order to process it *in the same way*. This does not make sense for certain types of processing step (e.g. manual deglitching again), so the history would have to maintain a state of whether this operation remained valid or not.

The history could also be used to implement an “undo” mechanism. For certain operations this implies that data must be copied and a process applied to the copy. In some cases this could result in a severe performance penalty. Therefore it should be optional whether a process implements “undo” or not. If it does, it should be possible to turn it off to gain performance benefits, both at system-wide and single process levels. In particular, a pipeline should run with “undo” turned off.

When data is saved to a product, the history must be saved with it. This is essential to allow saved data to be restored in the same state, and also to enable products to be transferred between users.

Similarly, if data is exported to an external data format, the history has to go with it. If data is imported from an external format, whether it originated in the IA or not, it has to be tagged as such. This is because it is impossible for the IA to know for sure what has been done to it.

Language

It is envisaged that the IA software will be developed entirely in Java. A scripting language is also needed. Currently Jython appears to be the best candidate for this. The architecture should not prevent the scripting language from being changed at a later date.

Interfaces to other Software

Data can be transferred to and from external packages by exporting to and importing from a common file format (e.g. FITS).

The IA architecture should allow additional processing modules, either written by ICC members or external users, to be “plugged in”, provided they adhere to the interface specification. It is currently planned that only the plugging in of Java modules will be supported by the ICC. However, the architecture should allow the *possibility* of plugging in modules written in different languages (principally IDL and C). It is possible that these modules would either run directly on the client or on a remote server. In the latter case, this would require the IA session to be continuously connected to the network.

These additional modules can become a part of a future version of the IA system. In this case, they must be written in Java. If they were originally developed using a different language, then they have to be ported.

User Interfaces

IA functionality will be available both from a command line and through a GUI. GUI displays could have their own command lines built in (e.g. like WinCVS). There will not be a single large GUI – each function will have its own display brought up by a master display.

A user-friendly possibility is to present a main display as a flowchart (representing a pipeline) indicating which processes have been executed. It will be possible to omit or repeat steps. Steps could be run by clicking on the relevant process. It will be possible to run all the steps together i.e. as a pipeline. There will be a number of different pipeline chains depending on instrument mode. It will also be possible for a user to create his or her own tailored pipelines.

It will be possible to save and restore processing state at any stage that a processing step is not running. Note that a processing step could be a composite of a number of processes.

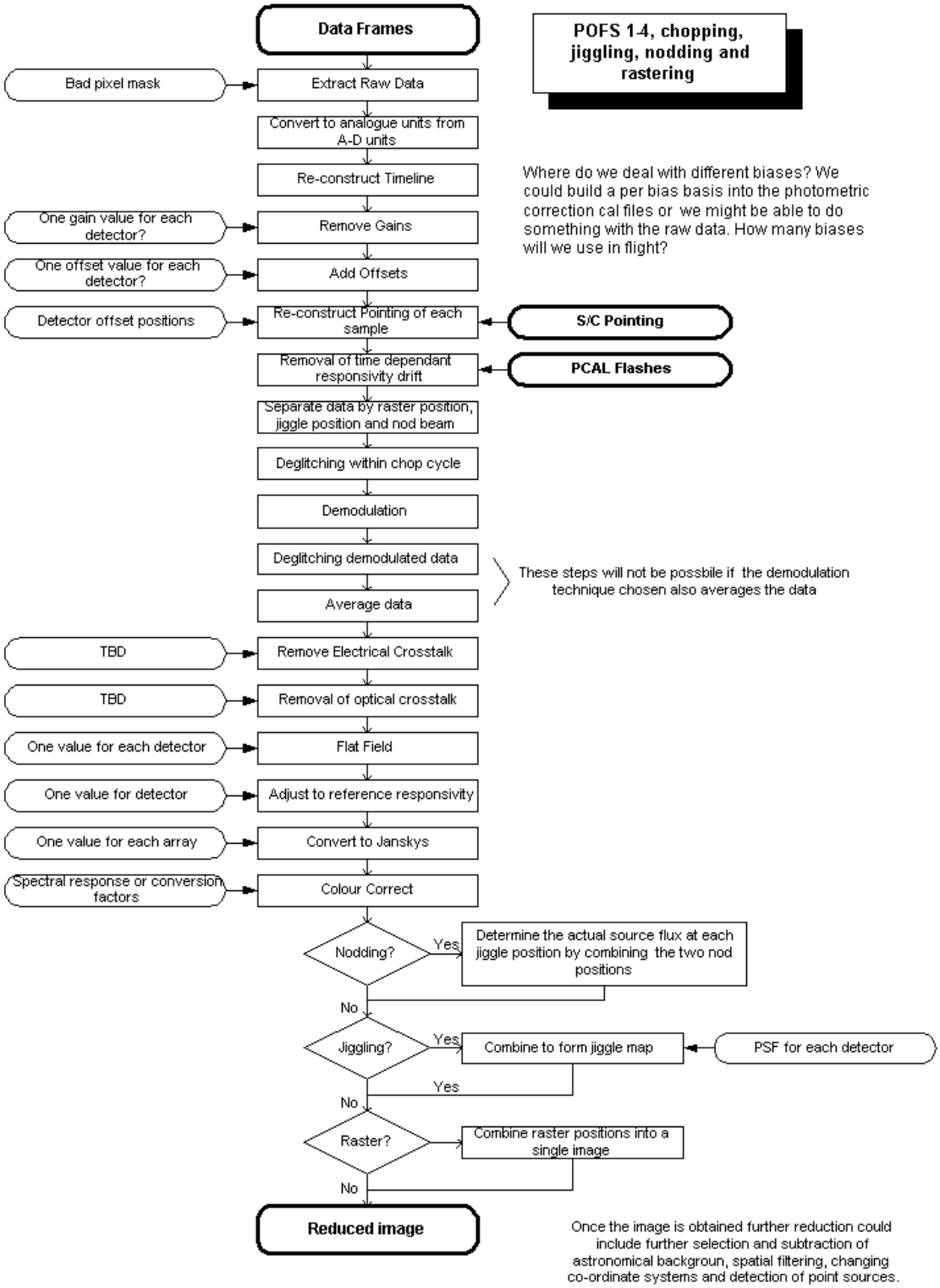
It will also be possible to apply different processing steps of the same type to the same product. For example, a user may wish to apply two different flat-fielding algorithms to an image and compare the results. Similarly, a user may wish to apply two different flat fields (with the same algorithm) and compare the results.

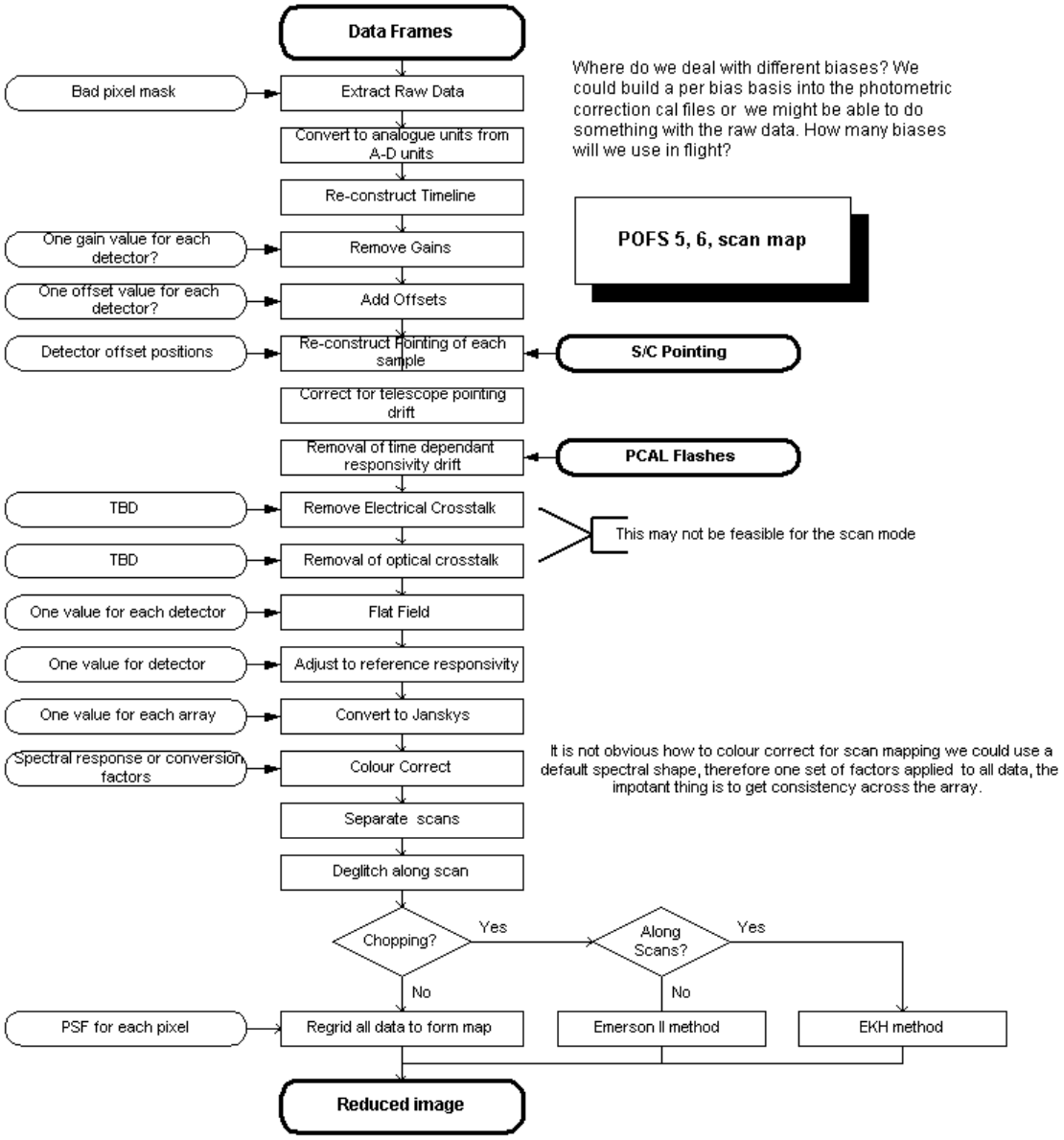
Output Products

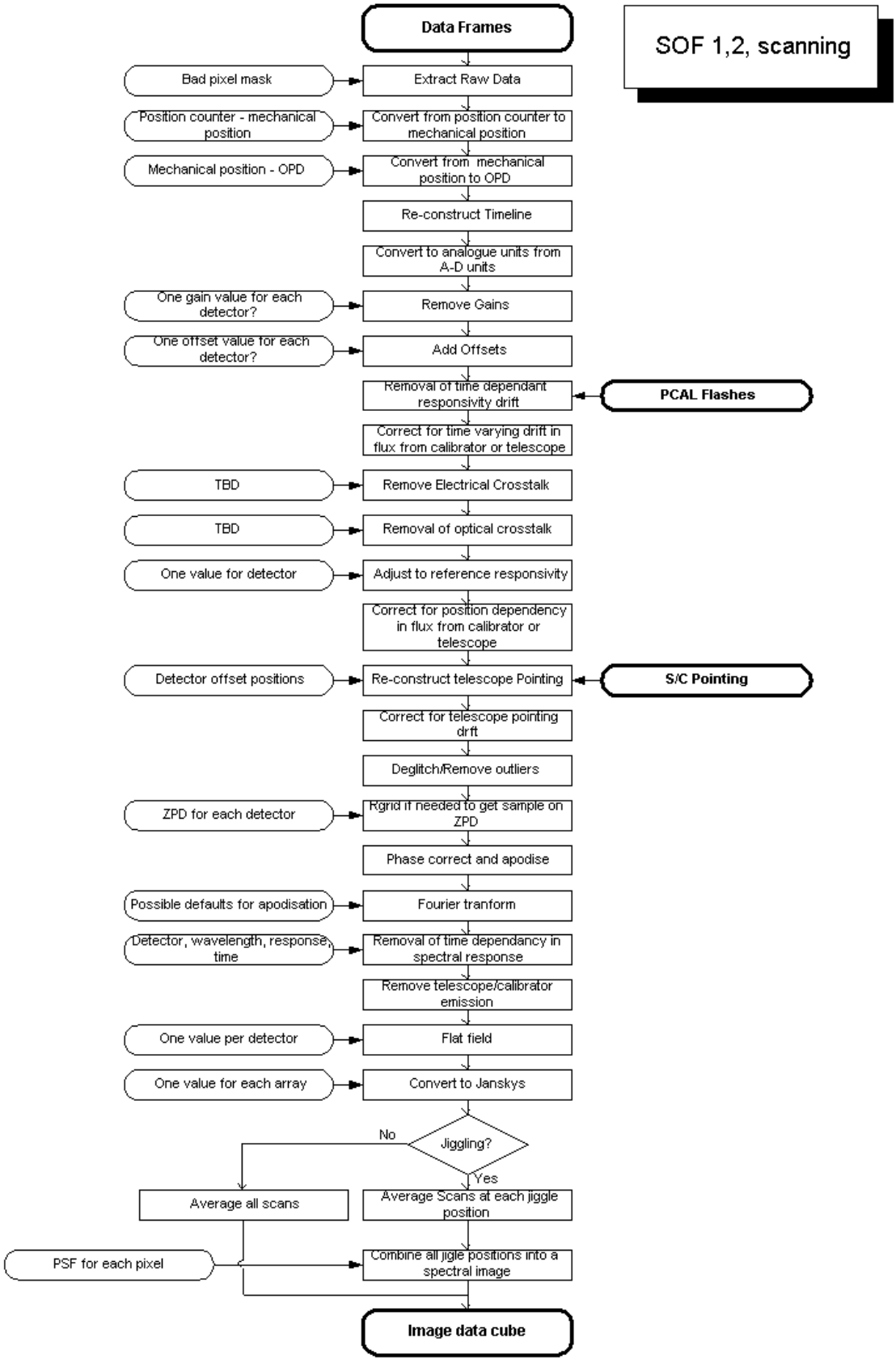
IA will be able to produce “products” as an output. These are products in the same sense as those produced by the HSC pipeline and therefore can be treated in an identical way be the IA, i.e. the output can become an input.

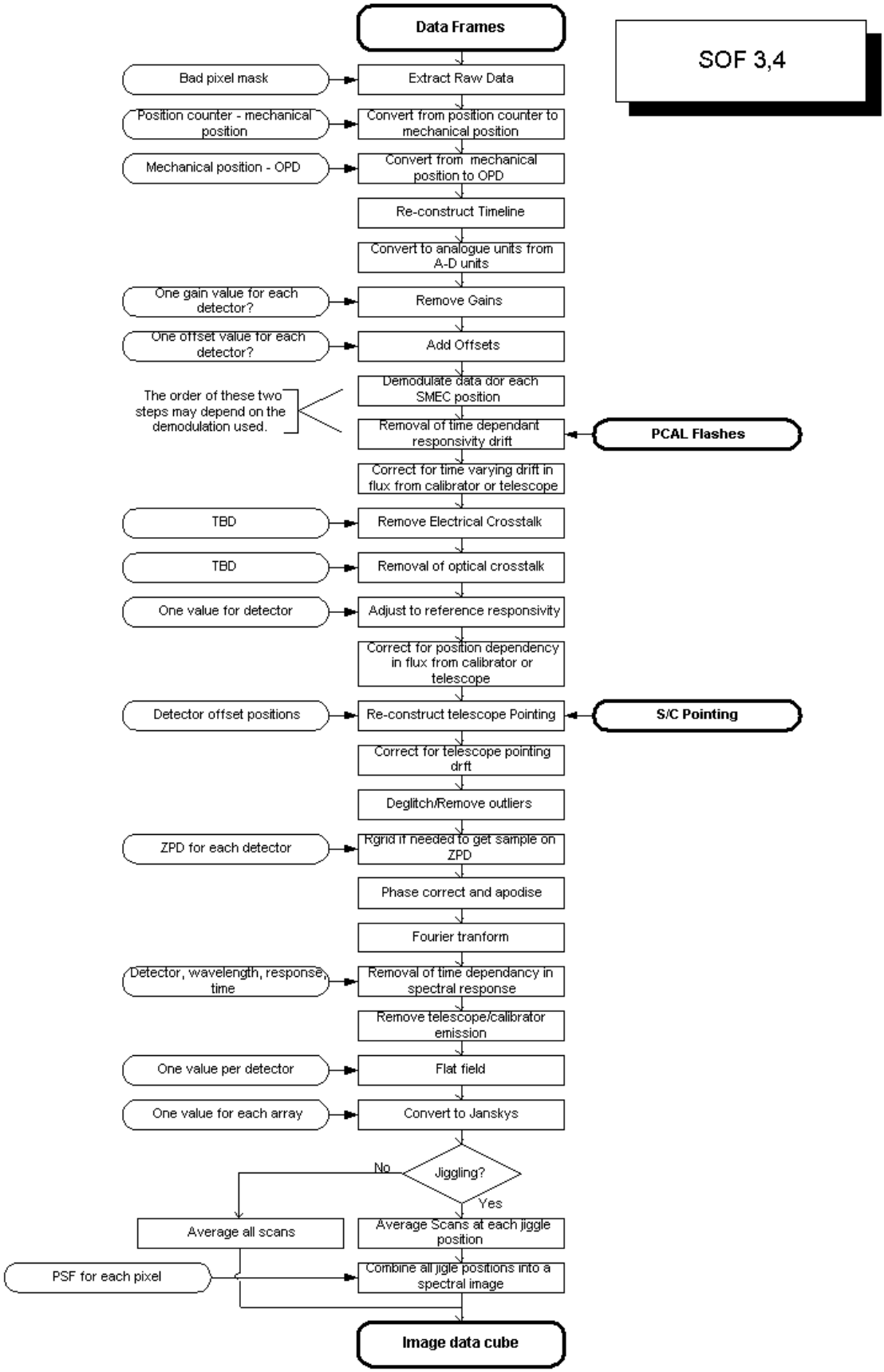
The general user will not be allowed to write products back into the database. However, in order to achieve a cohesive data archive and maximise data mining potential, a privileged user will be able to do so. Note that this does not disallow the addition to the archive of user-reduced data. Products that are added should be associated with the appropriate data in the archive. This is not necessarily an observation – it could, for example, be a survey.

Appendix – Data Processing Flow




Data Frames
Bad pixel mask
Extract Raw Data
Convert to analogue units from A-D units
Re-construct Timeline
One gain value for each detector?
Remove Gains
One offset value for each detector?
Add Offsets
Detector offset positions
Re-construct Pointing of each sample
S/C Pointing
Correct for telescope pointing drift
Removal of time dependant responsivity drift
PCAL Flashes
TBD
Remove Electrical Crosstalk
TBD
Removal of optical crosstalk
This may not be feasible for the scan mode
One value for each detector
Flat Field
One value for detector
Adjust to reference responsivity
One value for each array
Convert to Janskys
Spectral response or conversion factors
Colour Correct
Separate scans
Deglitch along scan
Chopping?
Yes
Along Scans?
Yes
No
No
PSF for each pixel
Regrid all data to form map
Emerson II method
EKH method
Reduced image
Where do we deal with different biases? We could build a per bias basis into the photometric correction cal files or we might be able to do something with the raw data. How many biases will we use in flight?
POFS 5, 6, scan map
It is not obvious how to colour correct for scan mapping we could use a default spectral shape, therefore one set of factors applied to all data, the impotant thing is to get consistency across the array.


Data Frames
SOF 1,2, scanning
Bad pixel mask
Extract Raw Data
Position counter - mechanical position
Convert from position counter to mechanical position
Mechanical position - OPD
Convert from mechanical position to OPD
Re-construct Timeline
Convert to analogue units from A-D units
One gain value for each detector?
Remove Gains
One offset value for each detector?
Add Offsets
Removal of time dependant responsivity drift
PCAL Flashes
Correct for time varying drift in flux from calibrator or telescope
TBD
Remove Electrical Crosstalk
TBD
Removal of optical crosstalk
One value for detector
Adjust to reference responsivity
Correct for position dependency in flux from calibrator or telescope
Detector offset positions
Re-construct telescope Pointing
S/C Pointing
Correct for telescope pointing drft
Deglitch/Remove outliers
ZPD for each detector
Rgrid if needed to get sample on ZPD
Phase correct and apodise
Possible defaults for apodisation
Fourier tranform
Detector, wavelength, response, time
Removal of time dependancy in spectral response
Remove telescope/calibrator emission
One value per detector
Flat field
One value for each array
Convert to Janskys
No
Jiggling?
Yes
Average all scans
Average Scans at each jiggle position
PSF for each pixel
Combine all jigle positions into a spectral image
Image data cube


Data Frames
SOF 3,4
Bad pixel mask
Extract Raw Data
Position counter - mechanical position
Convert from position counter to mechanical position
Mechanical position - OPD
Convert from mechanical position to OPD
Re-construct Timeline
Convert to analogue units from A-D units
One gain value for each detector?
Remove Gains
One offset value for each detector?
Add Offsets
The order of these two steps may depend on the demodulation used.
Demodulate data dor each SMEC position
Removal of time dependant responsivity drift
PCAL Flashes
Correct for time varying drift in flux from calibrator or telescope
TBD
Remove Electrical Crosstalk
TBD
Removal of optical crosstalk
One value for detector
Adjust to reference responsivity
Correct for position dependency in flux from calibrator or telescope
Detector offset positions
Re-construct telescope Pointing
S/C Pointing
Correct for telescope pointing drft
Deglitch/Remove outliers
ZPD for each detector
Rgrid if needed to get sample on ZPD
Phase correct and apodise
Fourier tranform
Detector, wavelength, response, time
Removal of time dependancy in spectral response
Remove telescope/calibrator emission
One value per detector
Flat field
One value for each array
Convert to Janskys
Jiggling?
No
Yes
Average all scans
Average Scans at each jiggle position
PSF for each pixel
Combine all jigle positions into a spectral image
Image data cube