

Pools in HIPE

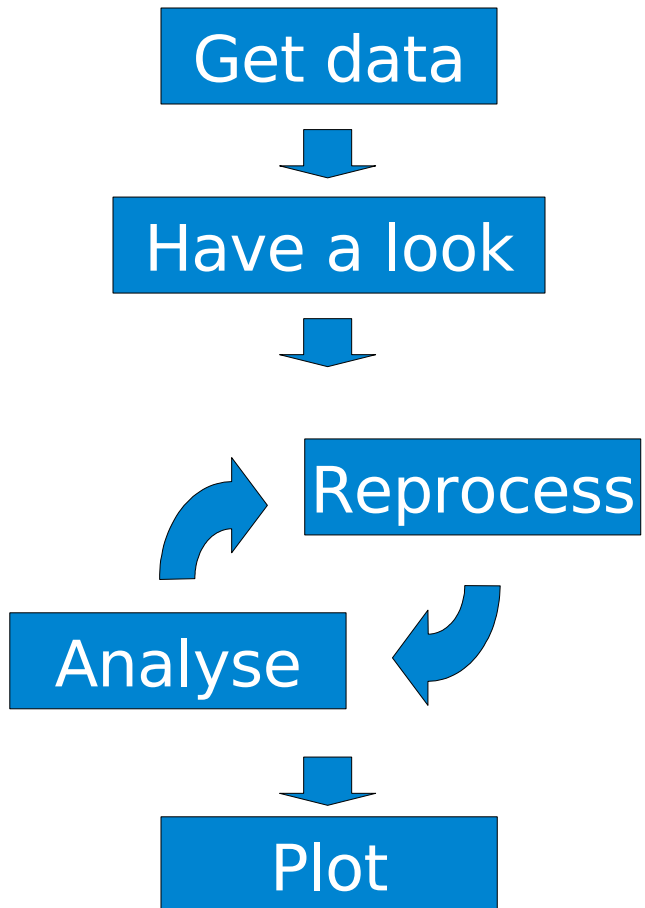
Why would I use *that*

Paul Balm

DP Workshop 2012

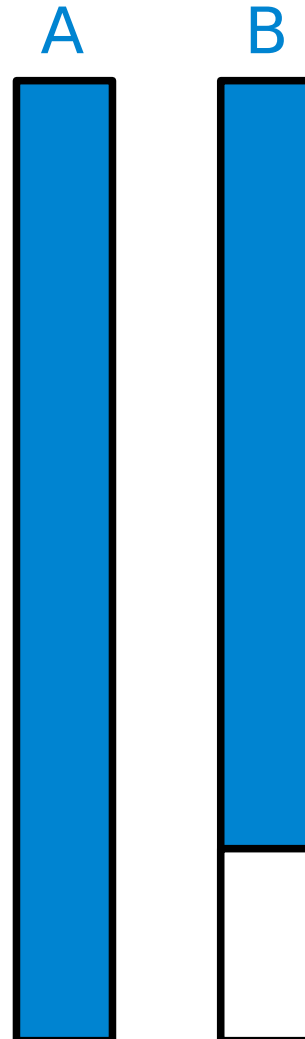
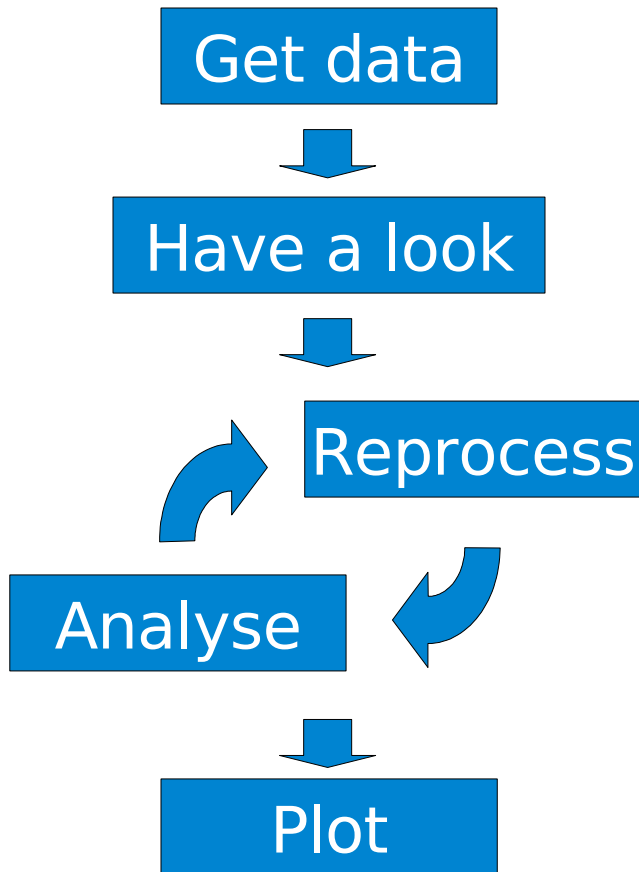
February 2012

Example workflow



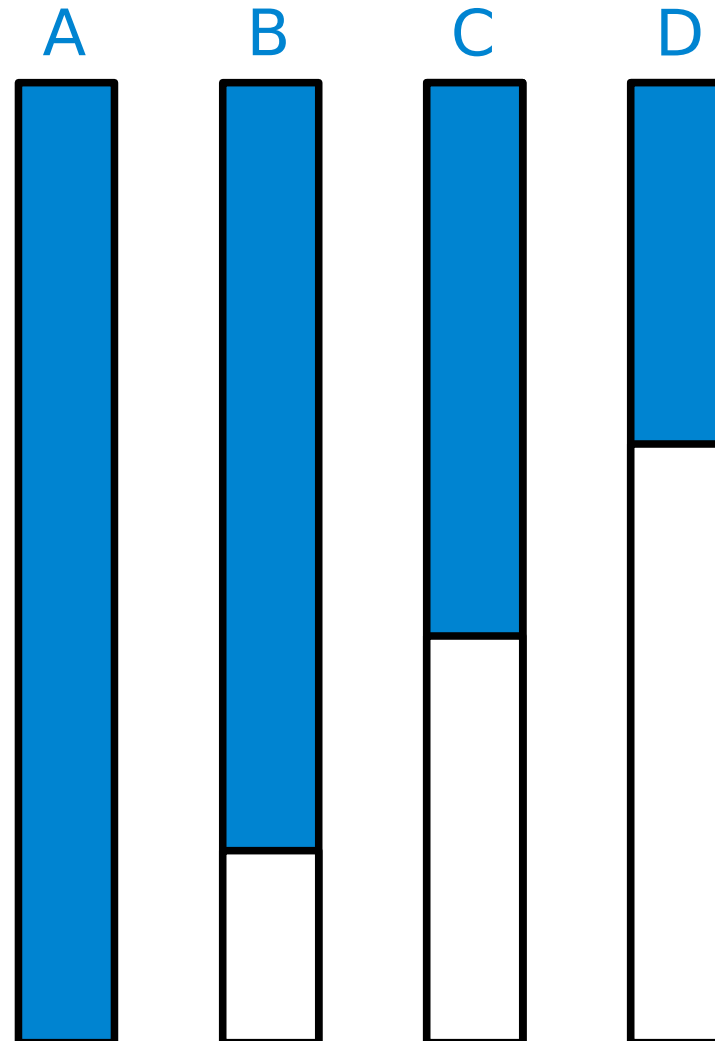
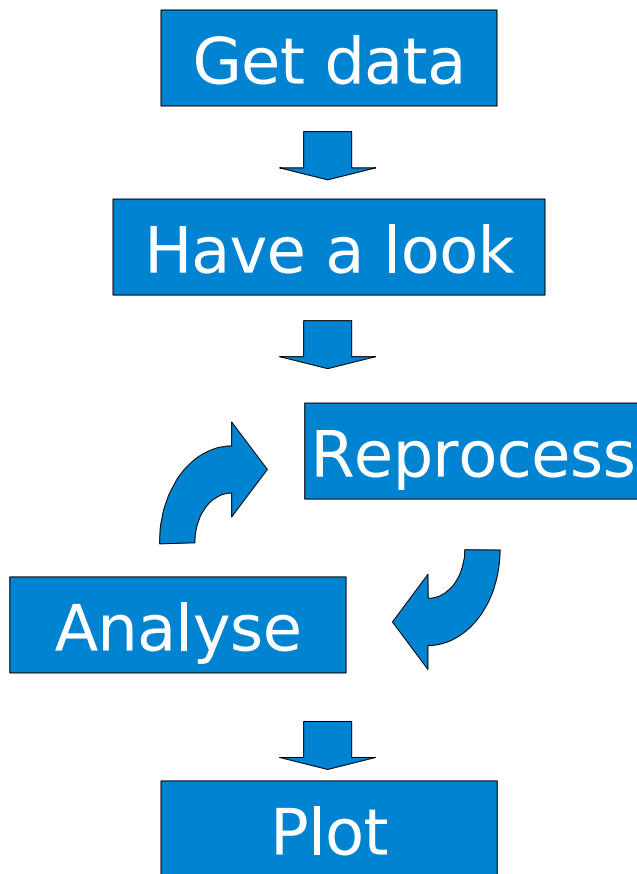
Introduction

Example workflow



Introduction

Example workflow



Scenario D: Only use HIPE “to have a look”
Export your data as FITS files.
(See “Basics on data access: ...” by Eva Verdugo)

Scenario A, B, C: You will want to store your data.

HIPE offers:

- Store as (FITS) files & work with them in HIPE
- Store in pools & work with data from pools

Goal of this talk



To allow you to make an informed decision
how to store your data:

- as FITS files
- in HIPE pools*

* I mean “local pools” in this talk.

What's a pool?



A pool is a database:

- store data, load it again later
 - searching (“querying”)
 - etc.
-
- Pool data is stored in files
 - Data should be accessed using HIPE,
not using the file system

- **Simplicity**
Easy to understand what you're doing.
- **Trust**
Not worried about losing your data.
- **Familiar tools** to work with files:
 - Command-line (cp, rm, ls...)
 - GUIs (file explorers)

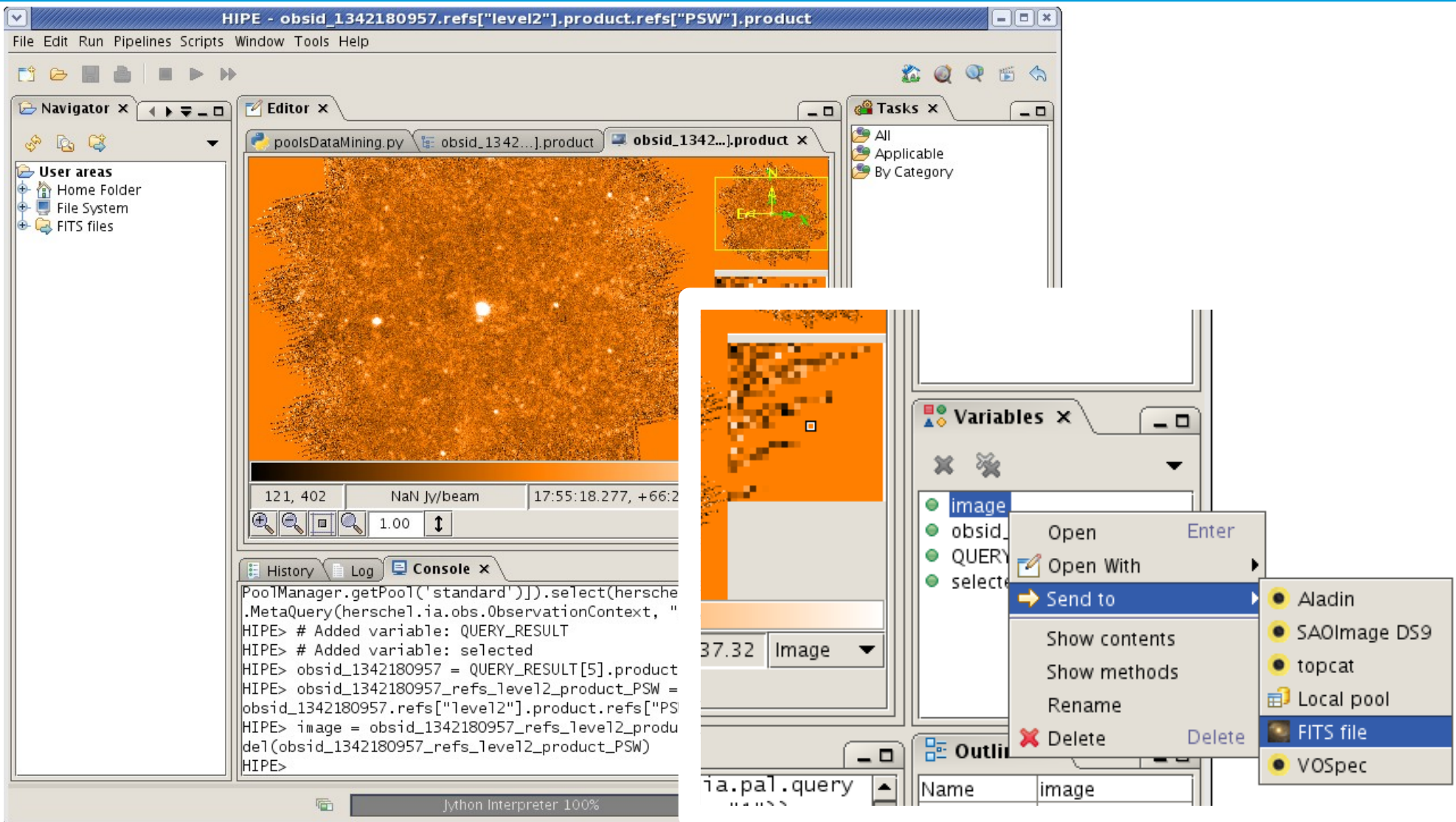
Pools offer:

- **Automatic versioning**
Keep track of different versions of your data.
- **Efficient use of disk space**
Save only the changes
- **Searching your data/Data mining**
E.g. find all images covering (ra, dec)...
- **Efficient memory usage**
Using references, load only what you need
- **Tagging**
Assign labels to your data

Disadvantages of one are lack of the advantages of the other...

- Files: Lack of features offered by pools
- Pools: Learning curve.

File I/O



The screenshot displays the HIPE software interface. The main window shows a data visualization of a star field in orange. A context menu is open over the visualization, with the 'Send to' option selected. The 'Send to' submenu is also open, showing various file formats and destinations. The console window at the bottom shows the following code and output:

```
PoolManager.getPool('standard')).select(herschel.  
.MetaQuery(herschel.ia.obs.ObservationContext, "  
HIPE> # Added variable: QUERY_RESULT  
HIPE> # Added variable: selected  
HIPE> obsid_1342180957 = QUERY_RESULT[5].product  
HIPE> obsid_1342180957_refs_level2_product_PSW =  
obsid_1342180957_refs_level2_product_PSW  
HIPE> image = obsid_1342180957_refs_level2_produ  
del(obsid_1342180957_refs_level2_product_PSW)  
HIPE>
```

The context menu options are:

- Open (Enter)
- Open With
- Send to
 - Aladin
 - SAOImage DS9
 - topcat
 - Local pool
 - FITS file
 - VOSpec
- Show contents
- Show methods
- Rename
- Delete (Delete)

Pool GUI: Product Browser



GUI for working with files: File explorer

- Find files
- Remove, open, ...

GUI for working with pools: “Product Browser”

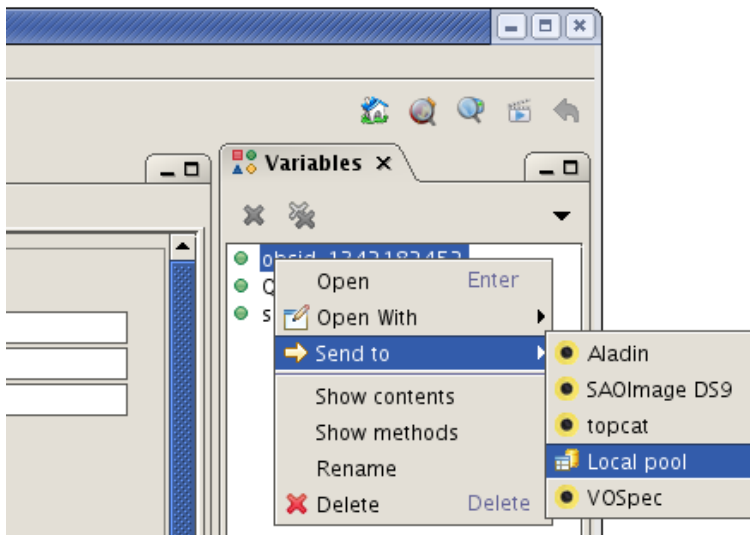
The screenshot shows the HIPE GUI with the Product Browser window open. The window title is "HIPE" and it has a menu bar with "File", "Edit", "Run", "Pipelines", "Scripts", "Window", "Tools", and "Help". The Product Browser window has tabs for "Observations", "Products", "Metadata", and "Free Metadata". The "Products" tab is active, showing search parameters and a table of results. A blue circle highlights the top-right corner of the Product Browser window, and a blue arrow points to it from the left. The console window at the bottom shows the following output:

```
HIPE> # Added variable: selected
HIPE> # Added variable: selected
HIPE> obsid_1342182452 = QUERY_RESULT[0].product
HIPE> # Added variable: selected
HIPE> # Added variable: selected
HIPE>
```

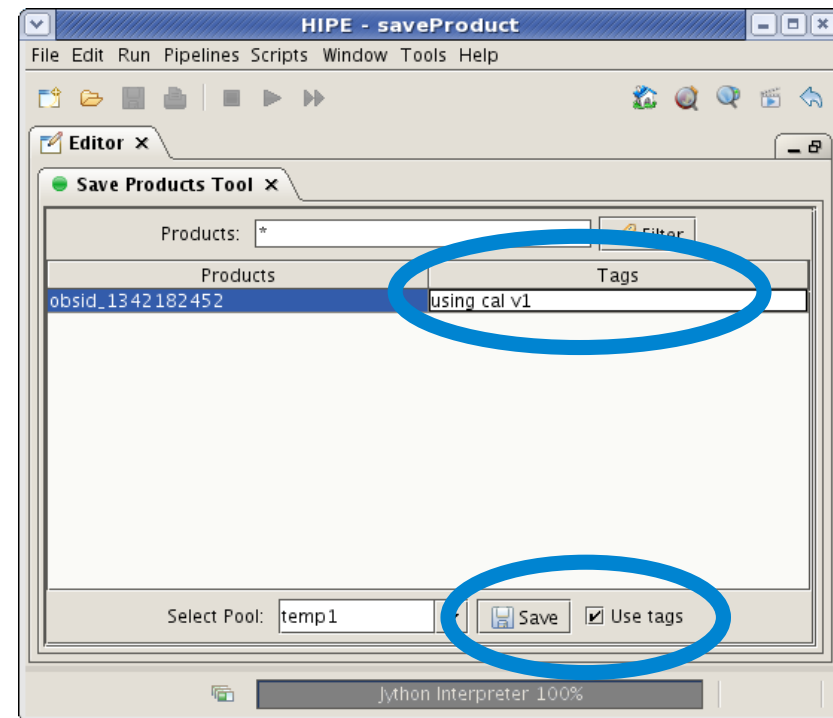
obsid	odNumber	startDate	aot	instrument	obsMode	object	proposal
1342180930	77	2009-07-29T23:00:...	Photometer	SPIRE	Large Map	19 Fortuna	Calibration_pvspire_4
1342180952	77	2009-07-30T02:57:...	Photometer	SPIRE	Point Source	ngc 6543	Calibration_pvspire_4
1342180953	77	2009-07-30T03:10:...	Photometer	SPIRE	Small Map	ngc 6543	Calibration_pvspire_4
1342180954	77	2009-07-30T03:35:...	Photometer	SPIRE	Large Map	ngc 6543	Calibration_pvspire_4
1342180955	77	2009-07-30T03:51:...	Photometer	SPIRE	Point Source	ngc 6543	Calibration_pvspire_4
1342180956	77	2009-07-30T04:04:...	Photometer	SPIRE	Small Map	ngc 6543	Calibration_pvspire_4
1342180957	77	2009-07-30T04:22:...	Photometer	SPIRE	Large Map	ngc 6543	Calibration_pvspire_4
1342182452	98	2009-08-19T19:44:...	Photometer	SPIRE	Large Map	m83ul-1	Calibration_pvspire_14

Saving products with tags

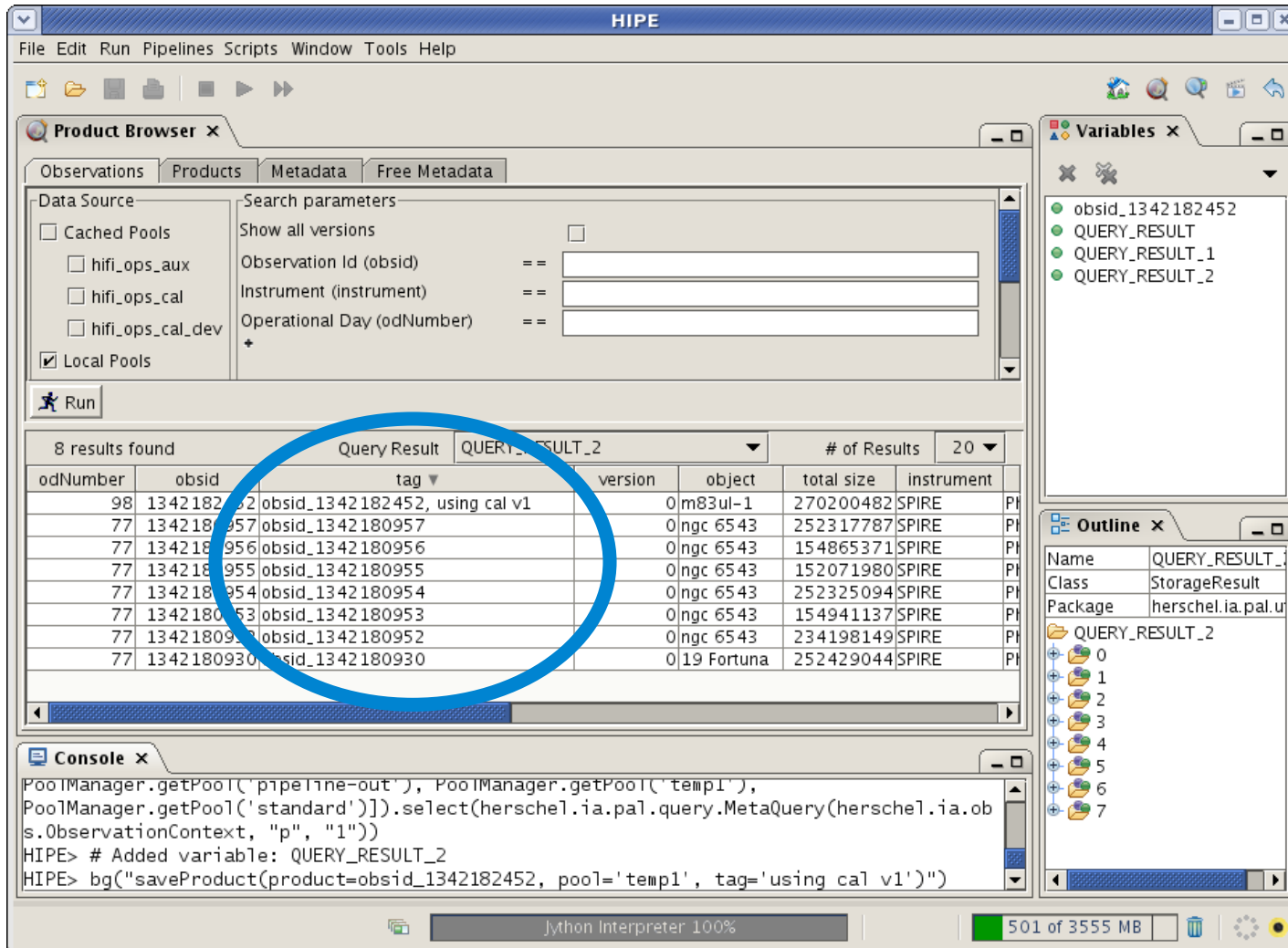
Use right-click menu:



Or: Window-->Show View-->
Data Access-->Save Products to Pool



Using tags



The screenshot shows the HIPE software interface. The main window displays a table of query results. A blue circle highlights the 'tag' column in the table. The table has 8 rows and 8 columns: odNumber, obsid, tag, version, object, total size, instrument, and # of Results. The first row is highlighted in blue.

odNumber	obsid	tag	version	object	total size	instrument	# of Results
98	1342182452	obsid_1342182452, using cal v1	0	m83ul-1	270200482	SPIRE	20
77	1342180957	obsid_1342180957	0	ngc 6543	252317787	SPIRE	
77	1342180956	obsid_1342180956	0	ngc 6543	154865371	SPIRE	
77	1342180955	obsid_1342180955	0	ngc 6543	152071980	SPIRE	
77	1342180954	obsid_1342180954	0	ngc 6543	252325094	SPIRE	
77	1342180953	obsid_1342180953	0	ngc 6543	154941137	SPIRE	
77	1342180952	obsid_1342180952	0	ngc 6543	234198149	SPIRE	
77	1342180930	obsid_1342180930	0	19 Fortuna	252429044	SPIRE	

The console window at the bottom shows the following commands and output:

```
PoolManager.getPool('pipeline-out'), PoolManager.getPool('temp1'),  
PoolManager.getPool('standard'])).select(herschel.ia.pa1.query.MetaQuery(herschel.ia.ob  
s.ObservationContext, "p", "1"))  
HIPE> # Added variable: QUERY_RESULT_2  
HIPE> bg("saveProduct(product=obsid_1342182452, pool='temp1', tag='using cal v1')")
```

Data mining using pools

```
HIPE - /home/pbalm/builds/Untitled.py
File Edit Run Pipelines Scripts Window Tools Help

Editor x
Untitled.py x

1 # Demo script for searches down to data level
2 # Author: Paul Balm (HSC)
3
4 # I have some data in a pool called temp1
5 pool = PoolManager.getPool('temp1')
6 storage = ProductStorage(pool)
7
8 # Define a function that returns true for the images we're looking for
9 def testImage(i):
10     # image type: Blue detector
11     if i.type != "HPPPMAPBS":
12         return False
13     # AND: at least one pixel over 1 Jy
14     if max(i.image) < 1.0:
15         return False
16     return True
17
18 # Define the query using this function and perform the search
19 query = FullQuery(SimpleImage, 'i', 'testImage(i)')
20 refs = storage.select(query)
21 print refs.size(), ' images found'
22
23 # Print the obsids of the found images which are the only ones with fluxes above 1 jy.
24 i = 0
25 for ref in refs:
26     m = ref.meta
27     print str(i), ': Obsid', m['obsid'].value
28     i += 1

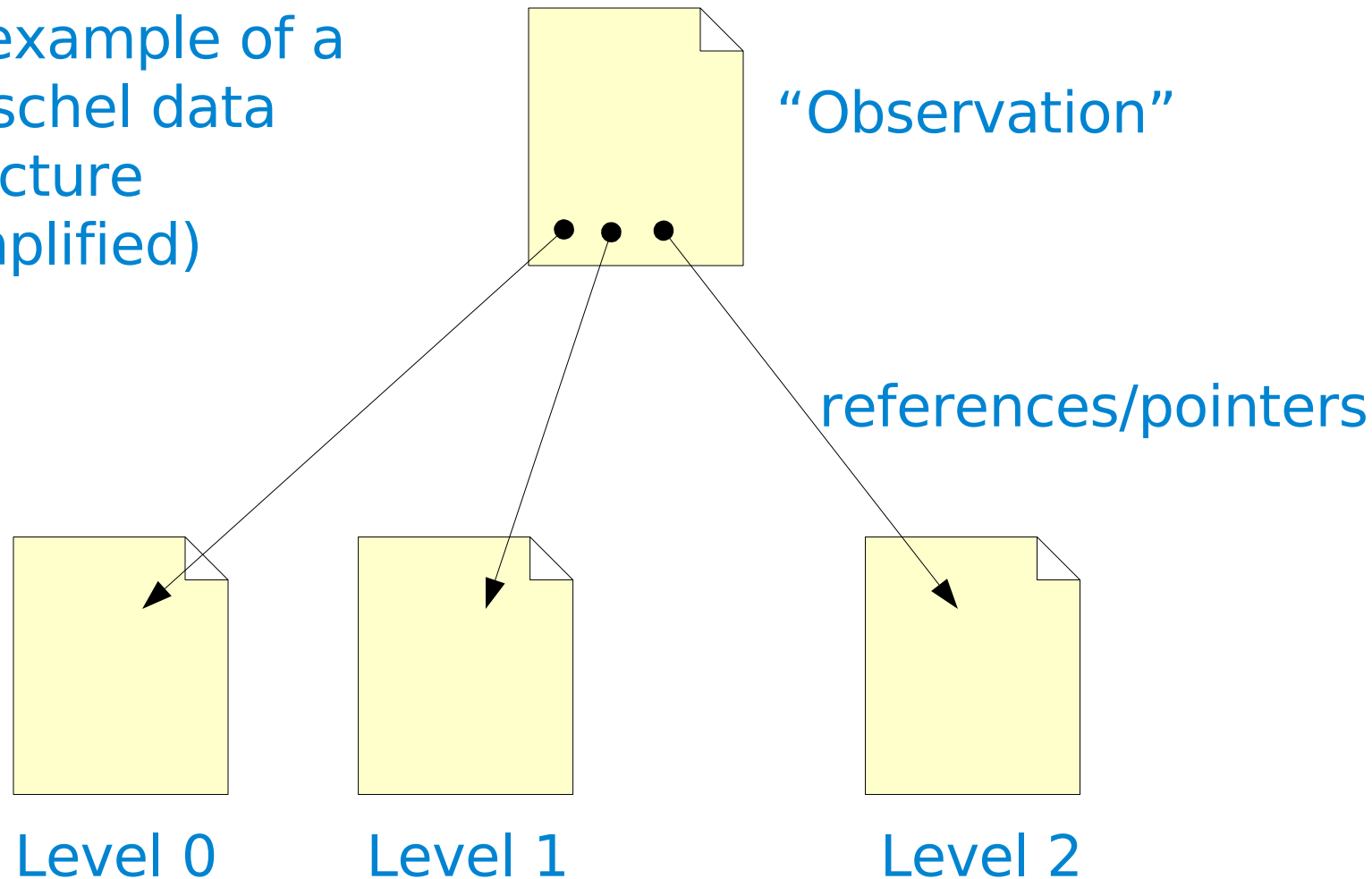
Untitled 100% | 965 of 3555 MB
```

Data mining from start to finish:

- Get observations from HSA
(see “*Basics on data handling...*”)
- Save them to a pool
 - Having data in multiple pools is ok as well
- Run the data-mining script

Efficient use of memory & disk space

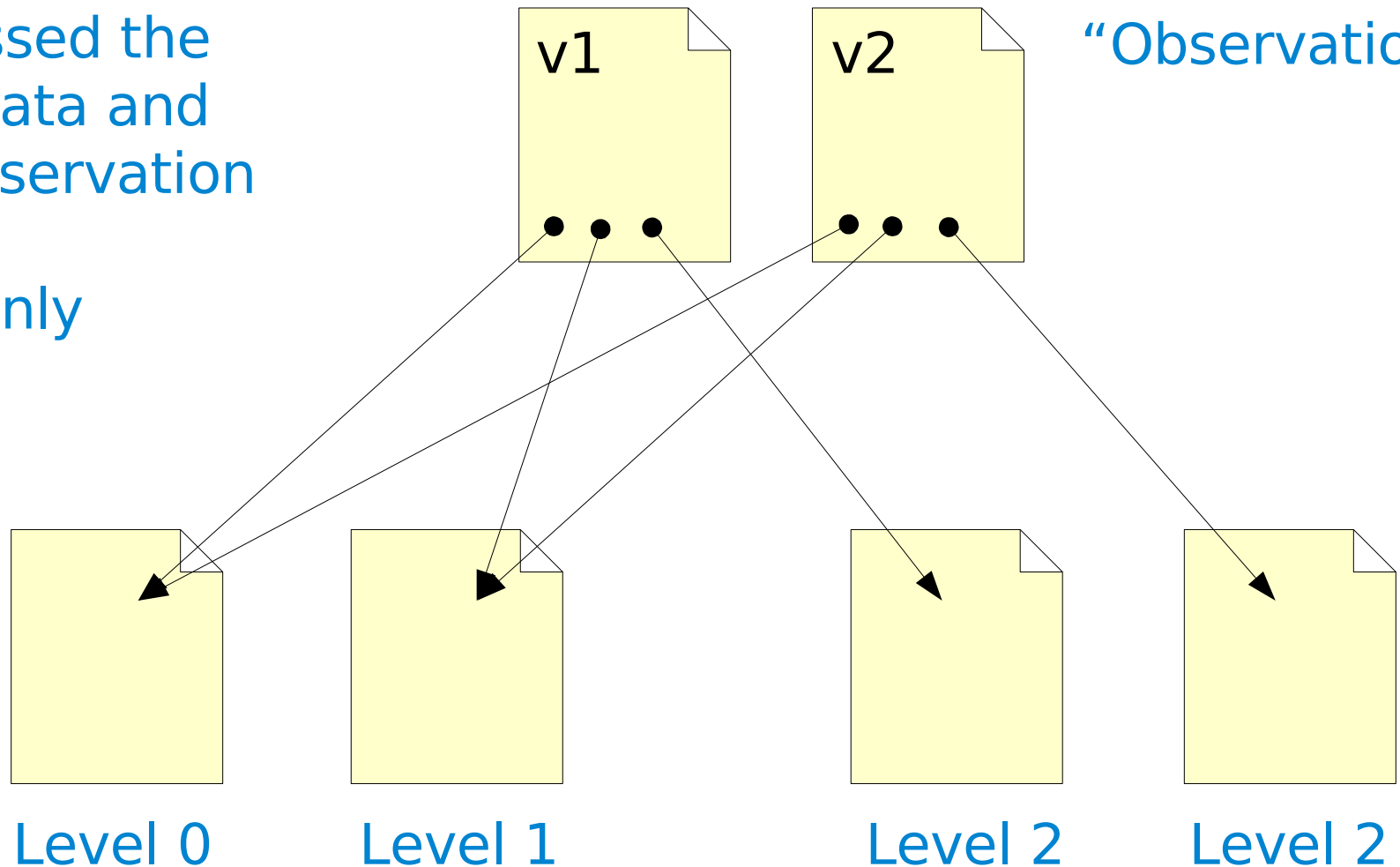
An example of a
Herschel data
structure
(simplified)



Efficient use of memory & disk space

Reprocessed the Level 2 data and saved observation again
- saved only new data

“Observation”



Level 2 reprocessed

Level 0.5
Calibration
Auxiliary
Quality