



Advanced CASA

Jan-Willem Steeb and Joshua Marvil

9th VLA Data Reduction Workshop



Introduction

This presentation will provide an in-depth look at how CASA works behind the scenes

By achieving a better understanding of CASA, it is my hope that you will be able to:

- use interactive CASA more efficiently
- access advanced CASA functionality
- directly read and write CASA data products
- integrate CASA with your own Python code

Outline

Installation

Structure of CASA

Scripting

Parallel CASA

Creating your own tasks

Jupyter Notebook Example

Installation

- Prerequisite Libraries
(<https://casadocs.readthedocs.io/en/latest/notebooks/introduction.html#Prerequisite-OS-Libraries>):
 - ImageMagick
 - org-x11-server-Xvfb
 - compat-libgfortran-48
 - libnsi
 - libcanberra-gtk2
- Install if not already available using (system administrator may be required):
 - yum (RedHat/CentOS)
 - apt-get (Ubuntu)
 - brew (Mac OS)

Installation: Monolithic Tar

Downloadable tar-file distribution with Python environment included.

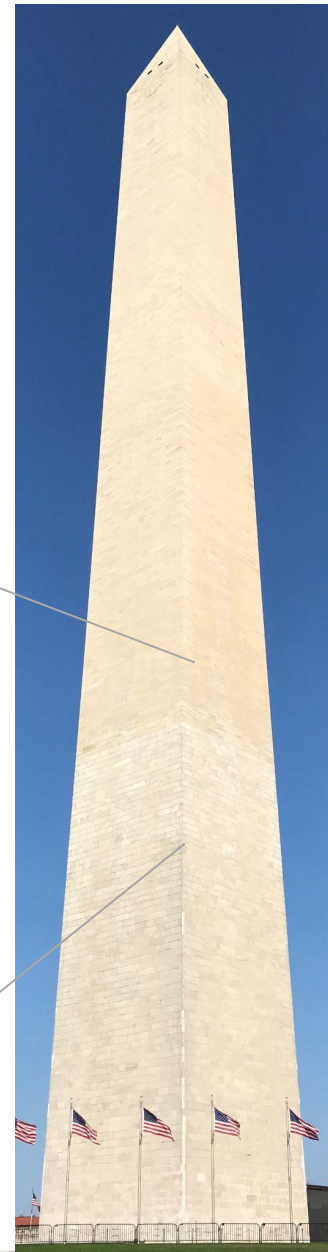
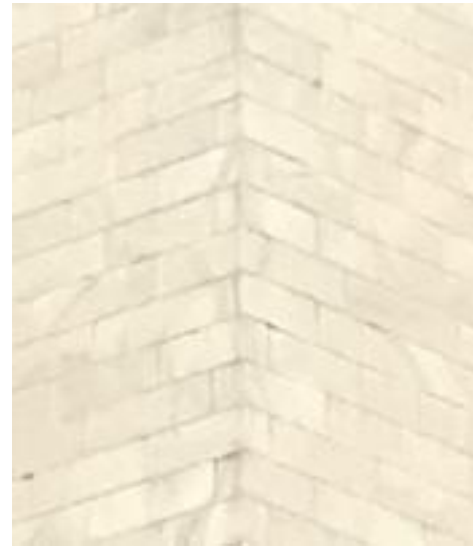
- Download Page:
 - https://casa.nrao.edu/casa_obtaining.shtml
- Compatibility Matrix:
 - <https://casadocs.readthedocs.io/en/latest/notebooks/introduction.html#Compatibility>



Installation: Modular

Modular CASA consists out of Python 3 modules that can be installed into an existing Python environment.

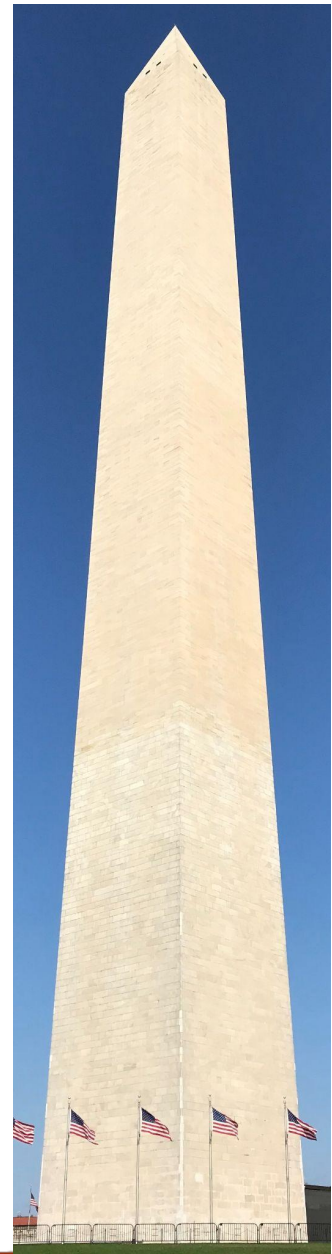
- [casatools](#)
- [casatasks](#)
- [casaplotms](#)
- [casaviewer](#)
- [casampi](#) (only Linux)
- [casashell](#)
- [casadata](#)
- [casaplotserver](#)
- [almataasks](#)
- [casatestutils](#)



<https://open-bitbucket.nrao.edu/projects/CASA>

Installation: Modular

The modular version is intended for manual data processing, and is not yet officially endorsed by ALMA or VLA. Currently, pipelines are included in and tested only for- all-inclusive monolithic CASA distributions.



Installation: Virtual Environment

Advisable to use a Python Virtual Environment.

- Python VirtualEnv

```
# In Terminal
$ python3 -m venv myvenv
$ source myvenv/bin/activate
(myvenv) $:
```



- Anaconda

```
# In Terminal
$ conda create -n myvenv python=3.8
$ conda activate myvenv
(myvenv) $:
```



Installation: Virtual Environment

Advisable to use a Python Virtual Environment.

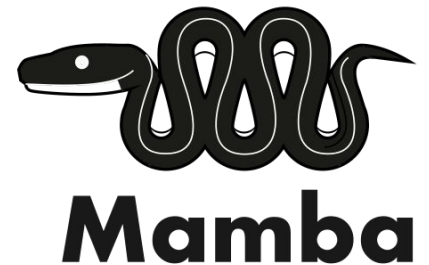
- Python VirtualEnv

```
# In Terminal
$ python3 -m venv myvenv
$ source myvenv/bin/activate
(myvenv) $:
```



- Anaconda/Miniconda/Mamba/Micromama

```
# In Terminal
$ conda create -n myvenv python=3.8
$ conda activate myvenv
(myvenv) $:
```



Installation: Virtual Environment

- Checking Virtual Environment Example:

```
# In Terminal
(base) $ which mamba
~/mambaforge/bin/mamba

(base) $ which conda
~/mambaforge/bin/conda

(base) $ which python
~/mambaforge/bin/python

(base) $ which ipython
/opt/local/bin/ipython

(base) $ conda activate myenv
(myenv) $

(myenv) $ which ipython
~/mambaforge/envs/myenv/bin/ipython
```

Installation: Virtual Environment

- Checking Virtual Environment Example:

```
# In Terminal
(base) $ which mamba
~/mambaforge/bin/mamba

(base) $ which conda
~/mambaforge/bin/conda

(base) $ which python
~/mambaforge/bin/python

(base) $ which ipython
/opt/local/bin/ipython

(base) $ conda activate myenv
(myenv) $

(myenv) $ which ipython
/users/jsteeb/mambaforge/envs/myenv/bin/ipython
```

If environments gets confused:

```
(myenv) $ conda activate base
(base) $ conda activate myenv
(myenv) $
```

or start new terminal session.

Installation: Virtual Environment

- Which Virtual Environments Are Available:

```
(base) % conda info --envs
# conda environments:
#
base      *  ~/anaconda3
myenv     ~/anaconda3/envs/myenv
casa6     ~/anaconda3/envs/casa6
zinc      ~/anaconda3/envs/zinc
```

Conda Cheat Sheet:

https://docs.conda.io/projects/conda/en/latest/_downloads/69c8c83c892884d9bc3700172a949a5b/conda-4.6.pdf

Installation: Virtual Environment

Package Installation

- PIP (Pip Installs Packages):

```
# In Terminal  
(myenv)$ pip install casatools==6.2.1
```

- Conda also includes a package manager:

```
# In Terminal  
(myenv)$ conda install numpy==1.22.4
```

Installation: Virtual Environment

Package Installation

- PIP (Pip Installs Packages):

```
# In Terminal  
(myenv)$ pip install casatools==6.2.1
```

- Conda also includes a package manager:

```
# In Terminal  
(myenv)$ conda install numpy==1.22.4
```

- CASA wheels are only available via PIP.
- Advice for Conda Virtual Environments: Check if package is available first with Conda and if not then use PIP.

Installation: Virtual Environment

List all available versions of a package.

- PIP (a hack):

```
# In Terminal
(myenv)$ pip install casatools==999
ERROR: Could not find a version that satisfies the
requirement casatools==999 (from versions: 6.4.0.16,
6.4.3.27, 6.4.4.31, 6.5.0.15, 6.5.1.23)
(myenv)$ pip install casatools==6.4.0.16
```

- Conda:

```
# In Terminal
(myenv)$ conda search -f numpy
Loading channels: done
# Name      Version      Build                Channel
numpy      1.8.2        py27_blas_openblas_200  conda-forge
numpy      1.8.2        py27_blas_openblas_201  conda-forge
numpy      1.8.2        py34_blas_openblas_200  conda-forge
numpy      1.8.2        py34_blas_openblas_201  conda-forge
...
```

Installation: Virtual Environment

List Installed Packages

- PIP

```
# In Terminal
(myenv)$ pip list
Package                                Version      Editable project location
-----
requests                               2.28.1
rise                                    5.7.1
scikit-learn                           1.1.1
scipy                                    1.7.3
Send2Trash                              1.8.0
sirius                                  0.0.32      /Users/jsteeb/sirius
```

- Conda

```
# In Terminal
(myenv)$ conda list
# packages in environment at /users/jsteeb/mambaforge/envs/myenv:
#
# Name                                Version      Build             Channel
requests                             2.28.1      pypi_0           pypi
rise                                   5.7.1      py38h50d1736_1   conda-forge
scikit-learn                          1.1.1      pypi_0           pypi
send2trash                             1.8.0      pyhd8ed1ab_0     conda-forge
sirius                                 0.0.32      dev_0            <develop>
```


Installation: Virtual Environment

● Conda

```
# In Terminal
(myenv)$ conda list
# packages in environment at /users/jsteeb/mambaforge/envs/myenv:
#
# Name                Version                Build                Channel
requests             2.28.1                pypi_0              pypi
rise                  5.7.1                 py38h50d1736_1     conda-forge
scikit-learn         1.1.1                 pypi_0              pypi
send2trash           1.8.0                 pyhd8ed1ab_0       conda-forge
sirius                0.0.32                dev_0               <develop>
```

● Available Channels

```
(myenv)$ conda config --show channels
channels:
- conda-forge
- pkgw-forge
- defaults
```

● Specify Channel During Installation

```
(myenv)$ conda install -c conda-forge wcslib
```

Installation: CASA Data

Package dependencies are handled automatically by pip, with the exception of casadata which must be explicitly installed and updated by the user:

- Monolithic CASA

```
CASA <1>: !update-data
CASA <1>: pip list | grep casadata
casadata          2022.3.28
```

- Modular CASA

```
(myenv)$ pip install --upgrade --extra-index-url
https://go.nrao.edu/pypi casadata
(myenv)$ pip list | grep casadata
casadata          2022.3.28
```

Installation: CASA Data

- Note that currently the casadata package is only updated weekly. Users needing daily updates should use the custom location, repository copy, or rsync methods.
- When using a shared site-wide installation of CASA, a system administrator may be needed
- For further details see <https://casadocs.readthedocs.io/en/latest/notebooks/external-data.html#Updating-the-Data-Tables>

Outline

Installation

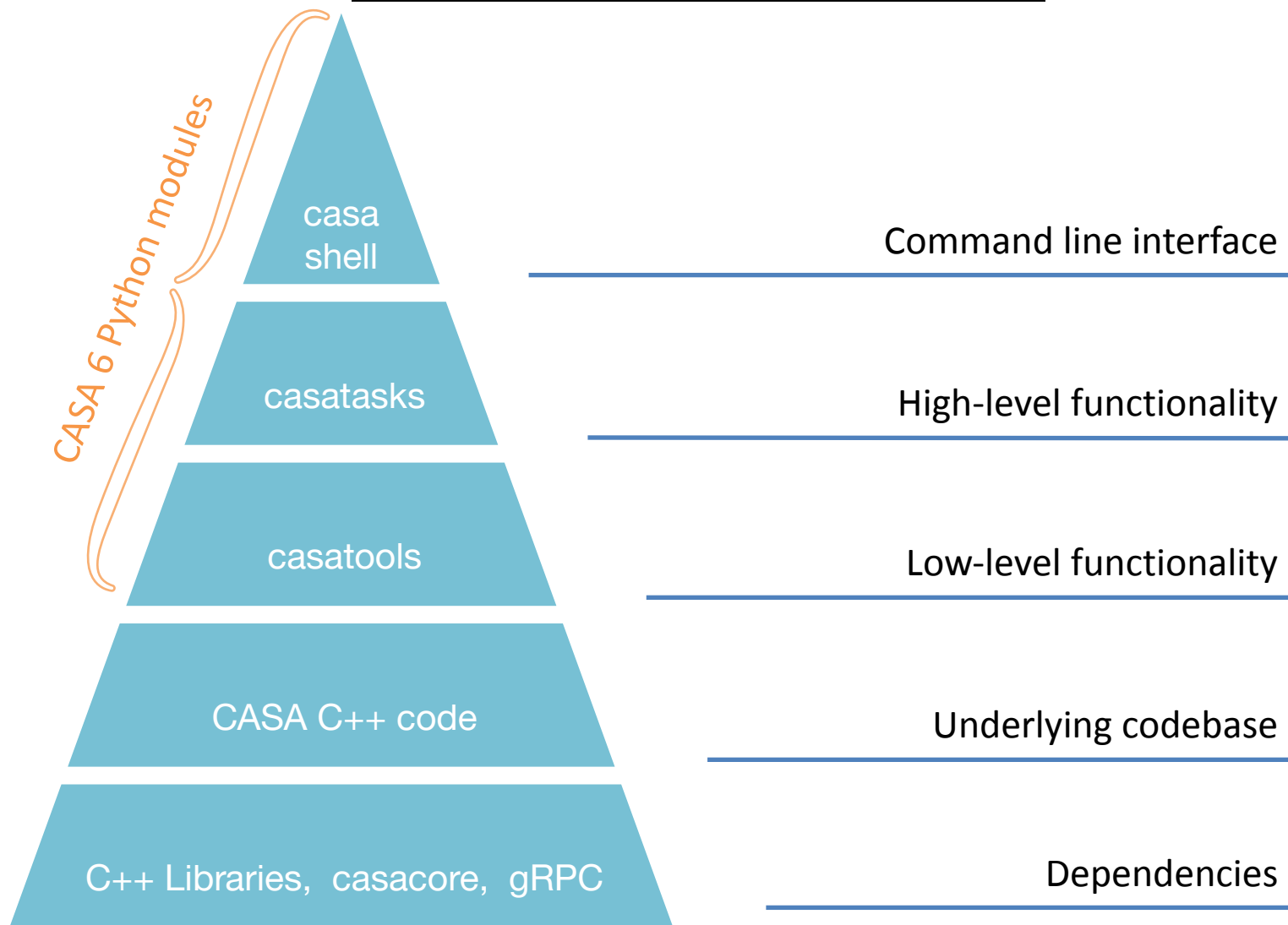
Structure of CASA

Scripting

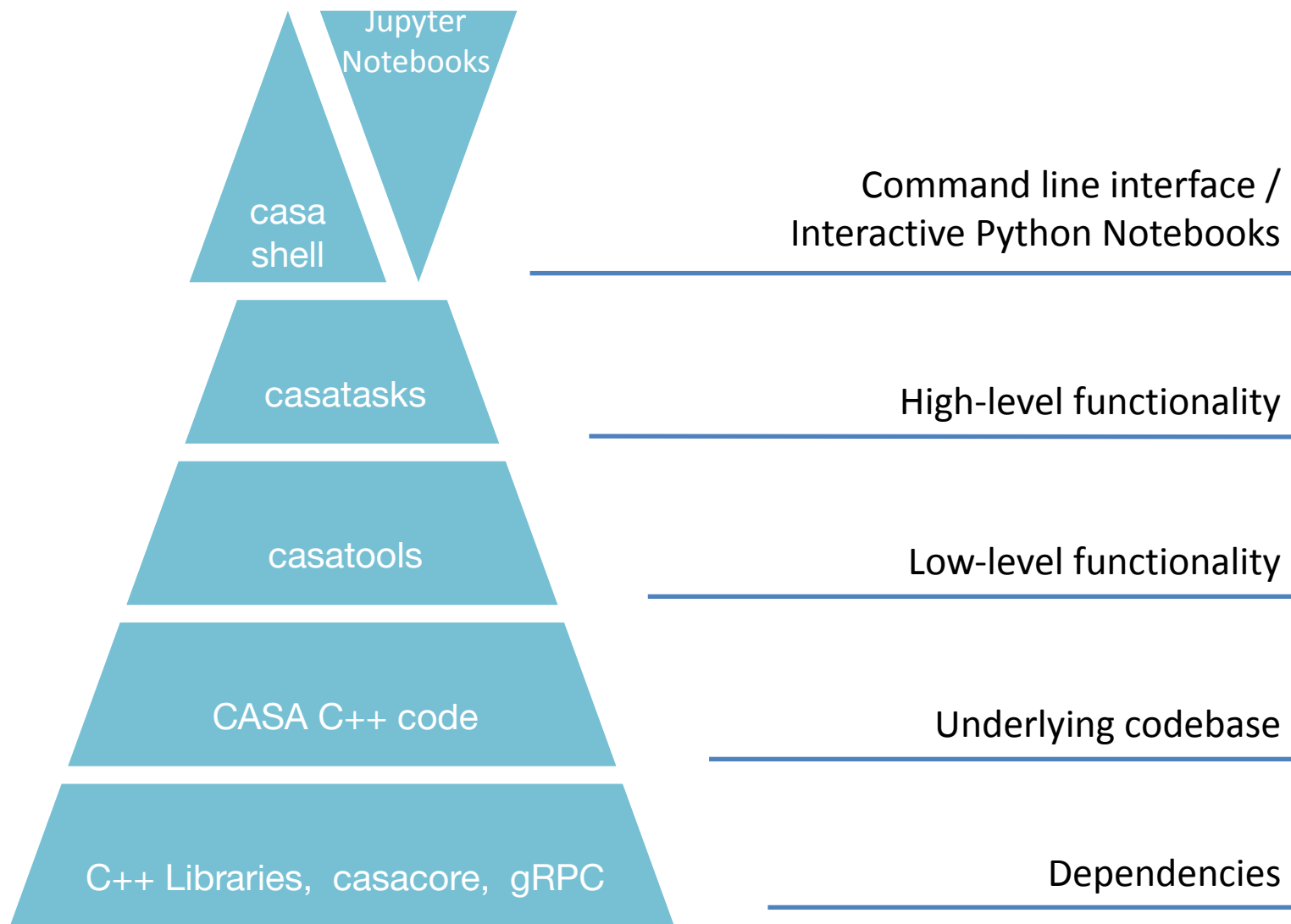
Parallel CASA

Jupyter Notebook Example

How CASA is Structured



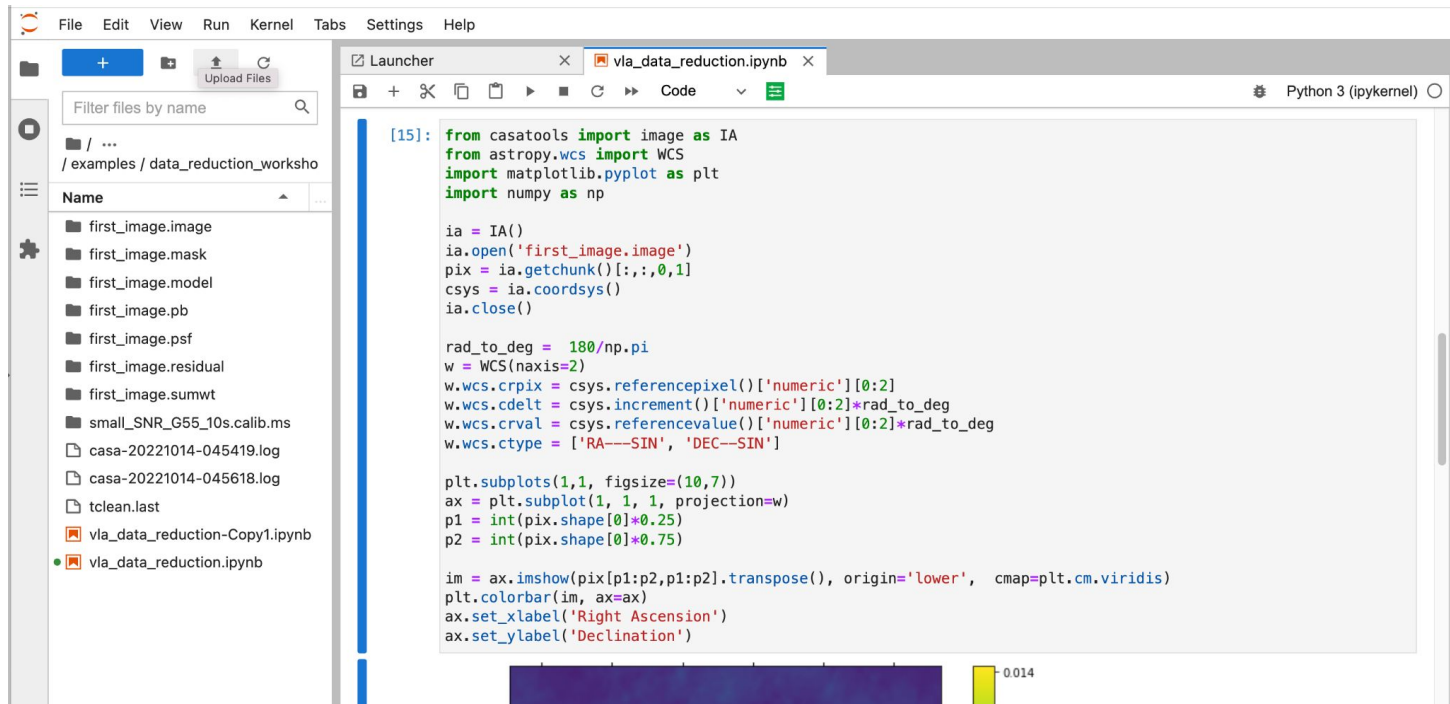
How CASA is Structured



Jupyter Notebooks

In addition to the relevant CASA packages the following packages are required

```
(myenv) $ conda install -c conda-forge jupyterhub
(myenv) $ conda install -c conda-forge jupyterlab
(myenv) $ conda install -c conda-forge ipywidgets
(myenv) $ jupyter lab
```



The screenshot displays a Jupyter Notebook window titled 'vla_data_reduction.ipynb'. The left sidebar shows a file explorer with a directory structure including 'first_image.*', 'small_SNR_G55_10s.calib.ms', and 'casa-20221014-045419.log'. The main area contains Python code for image reduction using Casatools, Astropy WCS, and Matplotlib. The code defines an image object, opens it, and extracts a chunk. It then sets up a WCS object with specific parameters and creates a subplot to display the image. A colorbar is added to the subplot, and the axes are labeled 'Right Ascension' and 'Declination'. The plot shows a dark image with a colorbar on the right side, ranging from 0 to 0.014.

```
[15]: from casatools import image as IA
from astropy.wcs import WCS
import matplotlib.pyplot as plt
import numpy as np

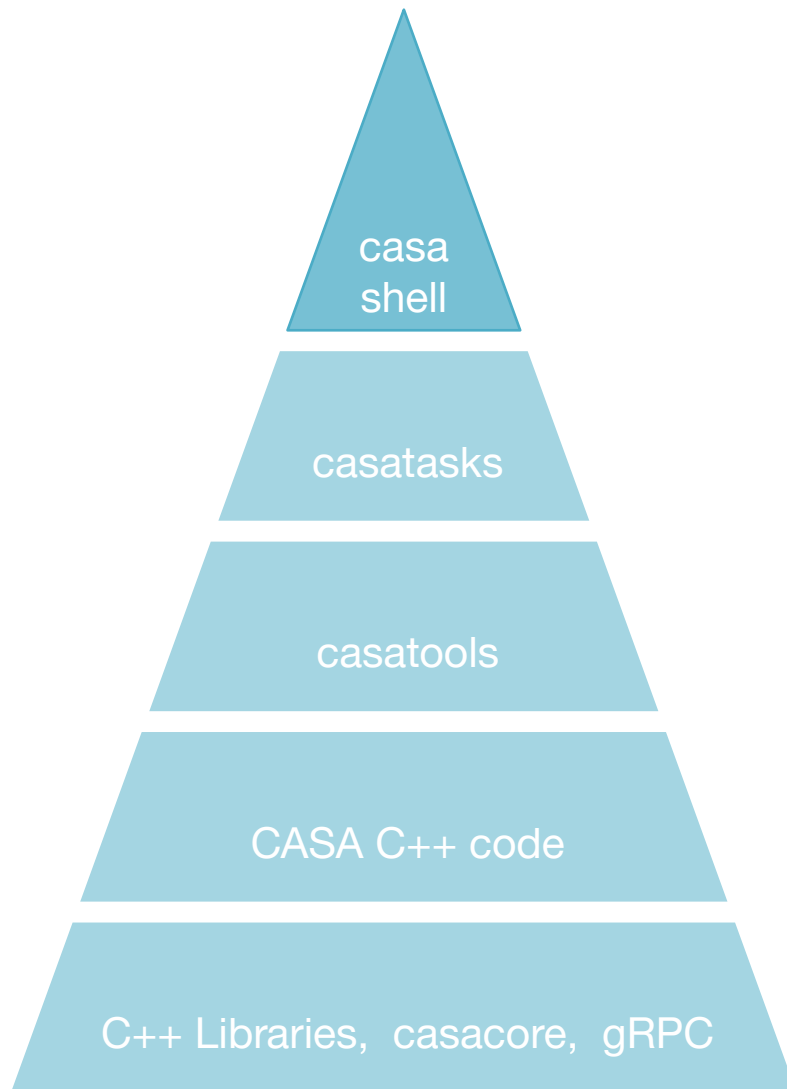
ia = IA()
ia.open('first_image.image')
pix = ia.getchunk()[::,0,1]
csys = ia.coordsys()
ia.close()

rad_to_deg = 180/np.pi
w = WCS(naxis=2)
w.wcs.crpix = csys.referencepixel()['numeric'][0:2]
w.wcs.cdelt = csys.increment()['numeric'][0:2]*rad_to_deg
w.wcs.crval = csys.referencevalue()['numeric'][0:2]*rad_to_deg
w.wcs.ctype = ['RA---SIN', 'DEC--SIN']

plt.subplots(1,1, figsize=(10,7))
ax = plt.subplot(1, 1, 1, projection=w)
p1 = int(pix.shape[0]*0.25)
p2 = int(pix.shape[0]*0.75)

im = ax.imshow(pix[p1:p2,p1:p2].transpose(), origin='lower', cmap=plt.cm.viridis)
plt.colorbar(im, ax=ax)
ax.set_xlabel('Right Ascension')
ax.set_ylabel('Declination')
```

casashell



casashell is what starts when you run 'casa'

CASA's interactive shell is a **customized Ipython environment**

Additional python functions create the task interface

CASA Shell: the Command Line Interface (CLI)

Interactive Shell

- Python interpreter
- standard library
- Python modules
- iPython extensions
- CASA modules
- startup scripts

Task system (what CASA Shell Adds)

- default
- inp
- go
- tget
- tput
- tasklist
- taskhelp

CASA Shell: the Command Line Interface (CLI)

Interactive Shell

- Python interpreter
- standard library
- Python modules
- iPython extensions
- CASA modules
- startup scripts

Task system (what CASA Shell Adds)

- default
- inp
- go
- tget
- tput
- tasklist
- taskhelp

As of CASA 6.5.2 the task system functionality is not available in a Jupyter Notebook.

Accessing CASA Shell Functionality in Modular CASA

CASA Shell that requires starting an interactive Python session:

```
#In Python
>> from casashell import start_casa
>> start_casa([ ])
Using configuration file ~/.casa/config.py
IPython 8.4.0 -- An enhanced Interactive
Python.
Using matplotlib backend: MacOSX
CASA 6.5.1.23 -- Common Astronomy Software
Applications
CASA <1>:
```

Python

Python objects, (e.g., int, float, bool, str, list, tuple, set, dict), and their methods

Other programming elements, e.g., operators, expressions, statements, syntax

Built-in functions, e.g., help, open, print, format, eval, exec, type, input

Additional Python modules

Several Python modules are available in CASA.

To find out available packages use:

```
pip list
```

Example:

```
import matplotlib.pyplot as plt  
import numpy as np  
plt.plot( np.random.randn(42) )
```

In **Monolithic** and Modular CASA non CASA modules must be imported.

IPython magic commands

IPython has many enhancements over the standard interpreter

Designed for interactive use— may not work in scripts

Magic commands, preceded by %, are one such enhancement

Others include completion <tab>, introspection ?, shell access !

To list all available magic commands use:

```
%lsmagic
```

IPython magic commands work in Monolithic CASA, Modular CASA
IPython and Modular CASA Jupyter Notebook.

IPython magic commands

Notable magic commands:

- automagic: % not required -- on by default
- autocal: () not required -- on by default
- pprint: pretty printing -- on by default
- cpaste: paste an entire cell (block of commands)
- history: view command line history

IPython system shell access

Shell commands can be executed with !

Output can be returned as a Python variable

If your desired system command is not found, check the path that CASA is using (e.g., %env PATH or os.environ['PATH'])

```
!du -hs *  
  
hostname = !echo $HOSTNAME  
  
print( hostname )
```


The IPython alias command

`%alias` is an IPython magic command that defines an alias to a system shell command

An alias is treated like a new magic command

Aliases have lower precedence than magic functions and normal Python variables

```
alias
```

```
alias mydu du -hs
```

```
mydu *
```

Further customization

CASA can be further customized by creating a file ***startup.py*** in the \$HOME/.casa directory

You can put commands here that will run every time CASA starts up, e.g., import modules, set variables, modify the PATH

You can also customize many aspects of IPython using ***startup.py***, e.g., the color scheme, magic commands, aliases, behavior of extensions

```
$ cat ~/.casa/startup.py

import sys, os
sys.path.append("/home/casa/contrib/AIV/science/analysis_scripts/")
import analysisUtils as aUes
```

CASA Tasking System

Tasking system

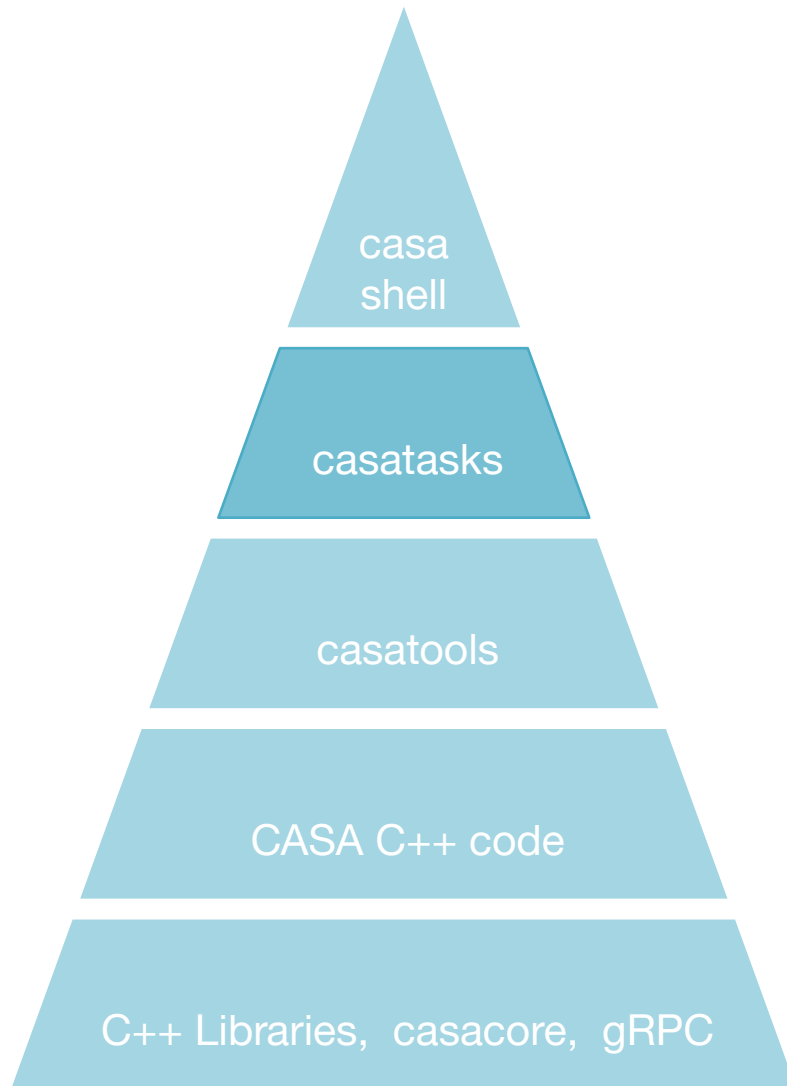
- default
- inp
- go
- tget
- tput
- tasklist
- taskhelp

Python functions that facilitate setting up and running tasks

Task parameters are variables in the global namespace

For an in depth explanation see:

<https://casadocs.readthedocs.io/en/latest/api/casashell.html>



CASA Tasks

Task (functional)
interface (XML)

- parameters
- description
- defaults
- expansions
- validation

Task script (Python)

- CASA toolkit
- logging
- history
- exceptions
- return values

CASA task XML

Each CASA task has an associated XML file

These are used by **casashell** when running the task interactively

Files are named '**taskname.xml**' and are located in a folder inside the CASA distribution

```
xml_path = casatasks.__path__[0]+'/__XML__'  
print( xml_path )  
ls $xml_path
```

CASA task 'scripts'

Each CASA task executes a Python script

It can be informative to read some of these scripts

Scripts are named '**task_***taskname*.py' and are located in a folder inside the CASA distribution

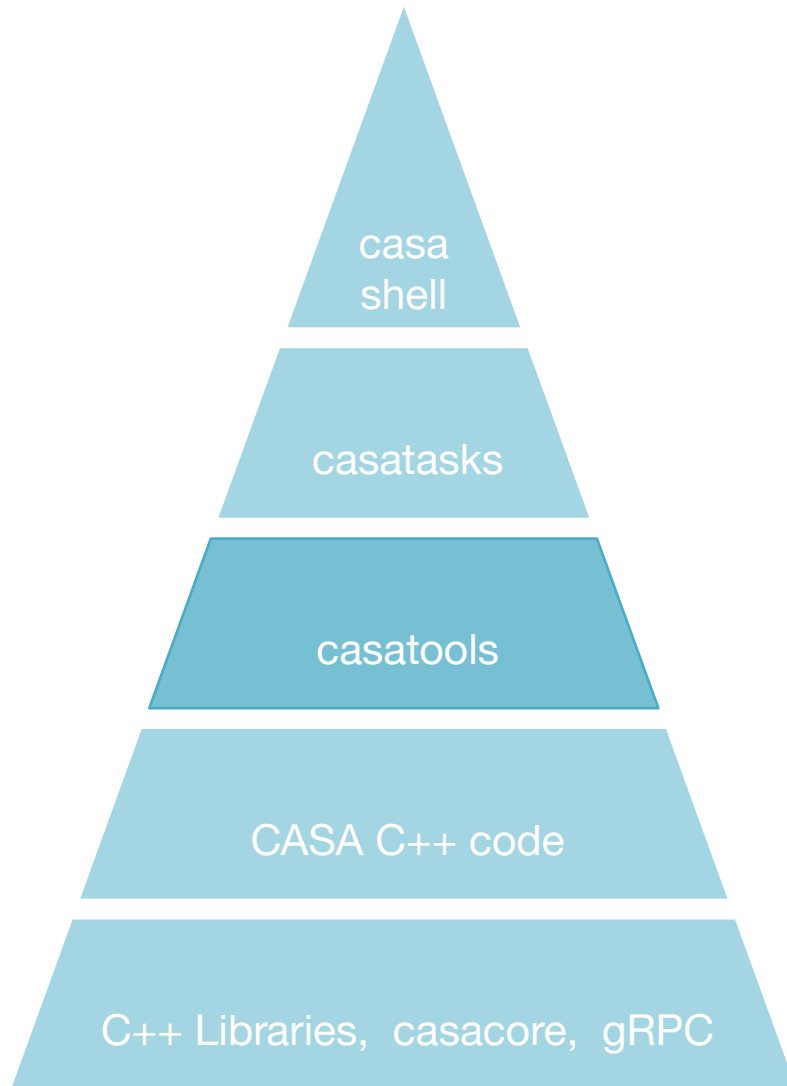
```
task_path = casatasks.__path__[0]+'private'  
print(task_path)  
ls $task_path
```

Initializing CASA tasks

The full CASA distribution will initialize tasks during startup.

If using the casatasks python module (**without CASA Shell**), you need to import them

```
from casatasks import tclean  
tclean( vis=...
```



CASA Tools

Object oriented interface.

Most tools are Python interfaces to the C++ code using Simplified Wrapper and Interface Generator (SWIG)

Tool methods are more atomic than tasks; many tasks consist of calls to several tool methods

Tasks are meant to capture common use cases and be simple to use

Tools offer additional flexibility and functionality over tasks

Comparison: task vs. tool

imstat task

```
from casatasks import imstat #If no CASA Shell
image = '3C75_initial.image.tt0'
imstat( image )
```

equivalent toolkit (image tool) method (inside *task_imstat.py*)

```
from casatools import image #If no CASA Shell
ia = image() #If no CASA Shell

ia = ia.open( image )

ia.statistics( robust=True )

ia.close()
```

CASA Tools (partial list)

af : Agent flagger

at : Atmospheric library

cb : Calibration

cl : Component list

cs : Coordinate system

ia : Image analysis

im : Imaging

me : Measures

ms : MeasurementSet

msmd : MS metadata

mt : MS transformer

mt : MS transformer

qa : Quanta

po : Imagepol

rg : Region manipulation

sdms : Single-Dish MS

sl : Spectral line catalog

sm : Simulation

tb : Table

vp : Voltage pattern

quanta tool

Values with units, unit conversion, string formatting

```
angle = qa.quantity( '1rad' )  
print( angle )  
{'unit': 'rad', 'value': 1.0}
```

quanta tool

Values with units, unit conversion, string formatting

```
qa.convert( angle, 'arcsec' )  
  
{'unit': 'arcsec', 'value': 206264.80624709636}  
  
qa.time( angle, prec=10 )  
  
['03:49:10.9871']
```

measures tool

Reference frames, directions, conversions, measurements

```
dir1 = me.direction('J2000', '06:18:00', '+41.00.00')
dir1
{'m0': {'value': 1.6493361431346414, 'unit':
'rad'},
 'm1': {'value': 0.7155849933176751, 'unit':
'rad'},
 'refer': 'J2000',
 'type': 'direction'}
```

measures tool

Reference frames, directions, conversions, measurements

```
me.doframe( me.observatory('VLA') )  
me.doframe( me.epoch('utc','today') )  
me.measure( dir1, 'azel' )
```

```
{'m0': {'unit': 'rad', 'value':  
0.7715118613823739},  
 'm1': {'unit': 'rad', 'value':  
0.11677474713900277},  
 'refer': 'AZEL',  
 'type': 'direction'}
```

MS metadata tool

Helper functions to retrieve measurement set properties

```
fields = msmd.fieldnames()  
  
print( fields )  
  
msmd.close()  
  
['3C75']
```

measurement set tool

Measurement set access and manipulation

```
ms.open( vis )  
  
results = ms.getdata( ['time','uvw'] )  
  
ms.close()
```

Note: this will try to retrieve data from all rows of the MS

Additional selection and / or iteration may be required for large data sets to avoid memory issues

measurement set tool

Measurement set access and manipulation

```
u,v = results['uvw'][:2]
```

```
plt.scatter( u, v )
```

```
plt.scatter( -u, -v )
```

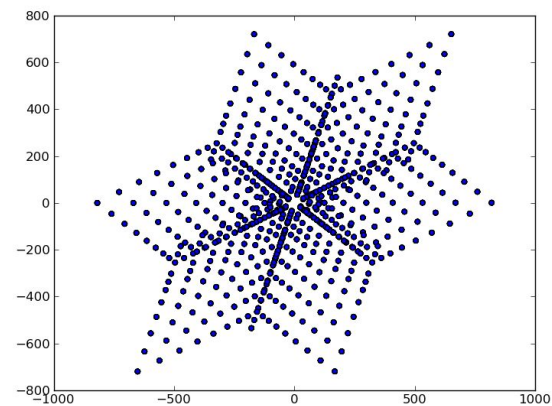
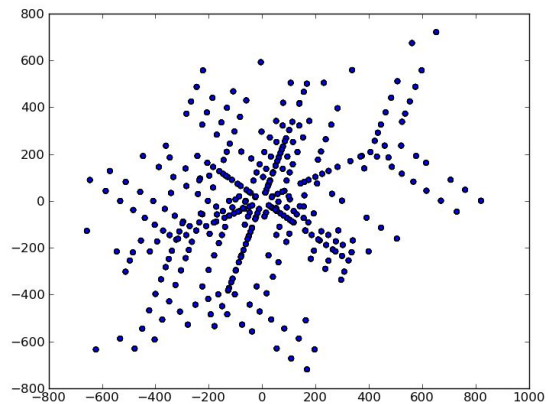


image analysis tool

Image inspection and manipulation

```
ia.open( image )  
image_data = ia.getchunk()  
ia.close()  
image_data.shape  
np.max( image_data )
```

table tool

Read, write and filter CASA tables. Works on measurement sets, ancillary tables of the MS, calibration tables, images, component lists, etc.

```
# In CASA

tb.open( vis+'/ANTENNA' )

tb.colnames()

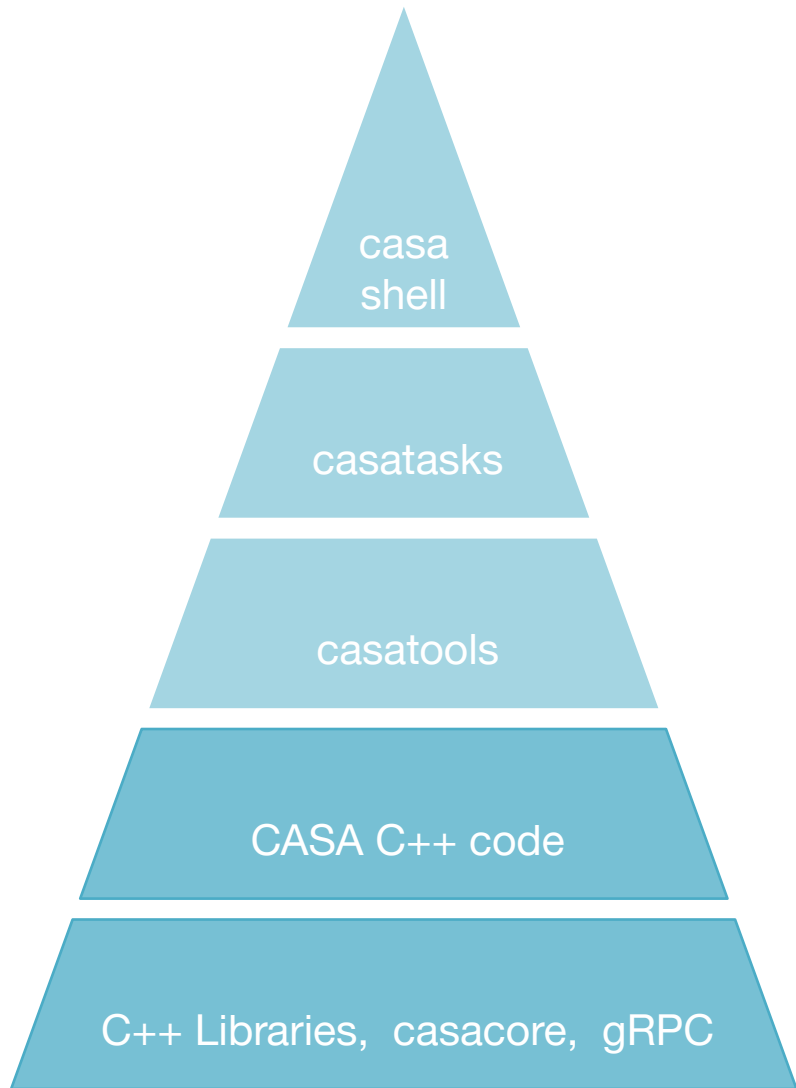
tb.getcol( 'NAME' )

stb = tb.query( "NAME == 'ea12'" )

stb.getcol( 'POSITION' )

stb.done()

tb.close()
```



CASA C++

Most of CASA is written in C++ for performance reasons

CASA is built on top of the casacore libraries (an independent project)

Other CASA C++ dependencies include: gsl, blas, lapack, atlas, cfitsio, wcslib, fftw

gRPC is used to facilitate communication with GUIs

Outline

Installation

Structure of CASA

Scripting

Parallel CASA

Jupyter Notebook Example

Getting started with CASA scripts

A CASA script is just a file that contains a sequence of commands, e.g., tasks, tools, general Python

Name your script almost anything you want, e.g. myScript.py

Run your script in CASA (several options):

```
execfile( 'myScript.py' )  
  
run -i 'myScript.py'  
  
exec(open('myfile.py').read())
```

Run your script from the terminal:

```
bash$ casa -c myScript.py
```

Why write a script?

Reproducible - an executable record of your processing

Efficient - launch a sequence of several commands

Inspectable - easy to edit, expand, debug

Shareable - distribute to colleagues / helpdesk / etc.

Transportable - run on different resources

Generalizable - compatible with other similar datasets

Tips for CASA scripts

It is generally advised to run CASA tasks as functions

Use Python fundamentals where appropriate, e.g., variables, loops, conditionals, exception handling

Learn from the examples in the documentation, e.g., casadocs, casaguides, the toolkit reference manual

Avoid Ipython magic commands, e.g., autocall

Use a persistent session when running remotely, e.g., screen, screen, nohup

Work with a Python-aware text editor

-block indent, block comment, syntax highlighting

Example CASA scripts

Example Script: G55.7 CASAguides tutorial

```
vis = 'G55.7+3.4_10s.ms'  
field = '0542*'  
spw = '2~3,5~6'  
  
setjy(vis=vis, field=field, spw=spw,  
      modimage= '3C147_L.im')  
  
gaincal(vis=vis, field=field, spw=spw,  
        caltable=vis.replace('.ms', '.G0'),  
        solint='int', calmode='p')
```

Example CASA scripts

For loops:

```
spws = ['2', '3', '4', '5']  
  
for spw in spws:  
    tclean( spw=spw, imagename='image_spw'+spw ... )
```

Flow control:

```
do_polcal = True  
  
if do_polcal:  
    polcal( ... )
```

3rd Party tasks and tools

There are many CASA tasks and tools written by members of the user community that can be imported into CASA

These are available from various locations, e.g., github, observatory websites, personal websites

Many of these are listed on the following CASAGuide page:

casaguides.nrao.edu/index.php/Contributed_CASA_Tasks_and_Scripts

Writing your own CASA tasks

You can turn your own code into a CASA task in 3 easy steps:

- Create your own ***task_taskname.py***
- Create a xml file ***taskname.xml***
- Run ***!buildmytasks*** to create ***taskname.py***

Then you can import and use your new task!

More info here:

[casadocs.readthedocs.io/en/latest/api/casa
shell/buildmytasks.html](https://casadocs.readthedocs.io/en/latest/api/casa_shell/buildmytasks.html)

Outline

Installation

Structure of CASA

Scripting

Parallel CASA

Jupyter Notebook Example

Parallel Processing

- **OpenMP** (Open Multi-Processing) is a shared-memory parallel programming library which enables non-trivial parallelization and is presently only implemented in a few areas of the CASA codebase (for example imaging). This automatically runs in the background.
- **MPI** (Message Passing Interface) is a standardized and portable message-passing standard designed to function on parallel computing architectures.
 - Some tasks in imaging and calibration can run in parallel using a “Multi-MS” or “MMS”.
 - tclean (imaging code) can run in parallel on an unpartitioned ms.

Parallel Processing

- Parallel imaging on an MS file (rather than Multi-MS files) in tclean is an official mode of operations in the ALMA pipeline since Cycle-6, and officially endorsed by CASA as per CASA 5.4. We recommend users interested in parallel processing to use this mode of operation.
- For large data products, the imaging step often dominates the overall runtime, as well as the advantages that can be achieved with parallelization.
- Processing Multi-MS files, either for imaging or calibration, remains at the discretion of the user.
- Documentation:
<https://casadocs.readthedocs.io/en/stable/notebooks/parallel-processing.html#>

Parallel Processing

The collection of CASA processes which will run the jobs from parallelized tasks, is set up via *mpicasa*. The simplest example is to run CASA in parallel on the *localhost* using the available cores in the machine. A typical example would be to run CASA on a desktop with 16 cores such as the following example:

```
path_to_casa/mpicasa -n 16 path_to_casa/casa <casa_options>
```

1. *mpicasa*: Wrapper around *mpirun*
2. *-n* : MPI option to get the number of processes to run.
3. *16*: The number of cores to be used in the localhost machine.
4. *casa*: Full path to the CASA executable, *casa*.
5. *casa_options*: CASA options such as: *-c*, *-nogui*, *-log2term*, etc.

Parallel processing using *mpicasa* is not support for Mac OS/OSX.

Parallel Processing

It is also possible to use other nodes, which can form a “cluster”. Following the requirements given above, replace the “-n” option of *mpicasa* with a “-hostfile *host_file*”, as shown below:

```
mpicasa -hostfile <host_file> path_to_casa/casa <casa_options>
```

1. *host_file*: It is a text file containing the name of the nodes forming the cluster and the number of cores to use in each one of the nodes.

Running on a cluster is not an official mode of operations for the ALMA or VLA pipeline.

Outline

Installation

Structure of CASA

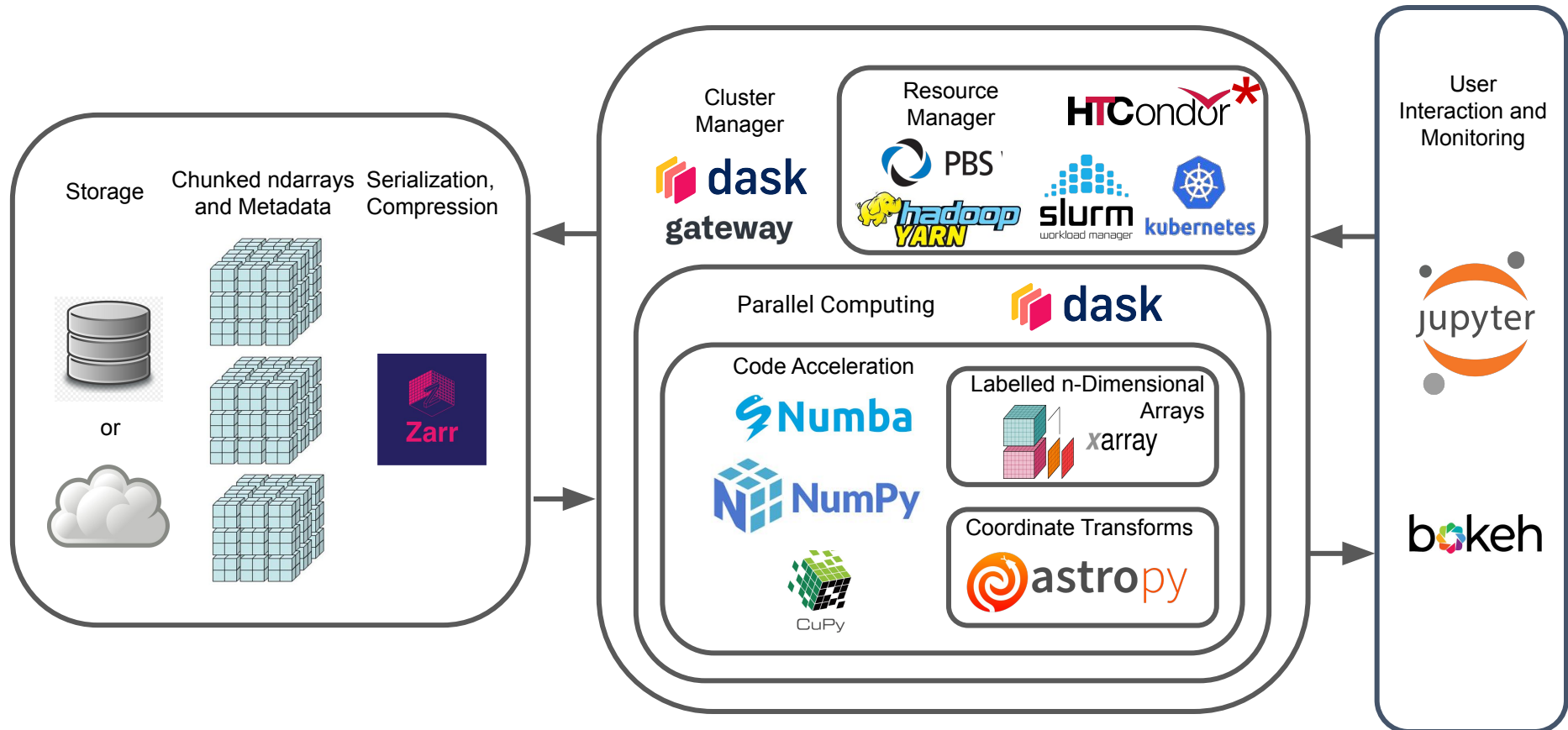
Scripting

Parallel CASA

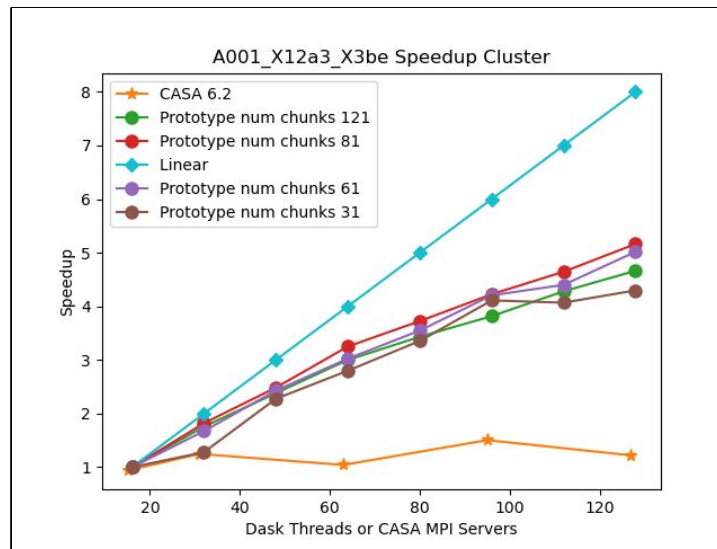
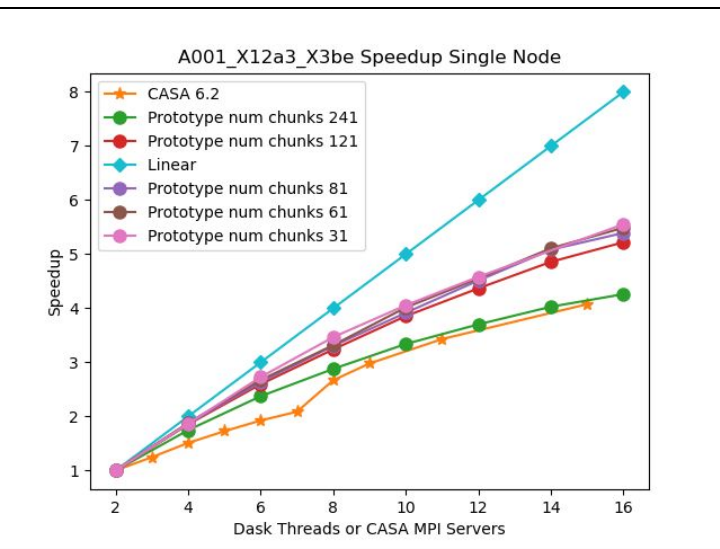
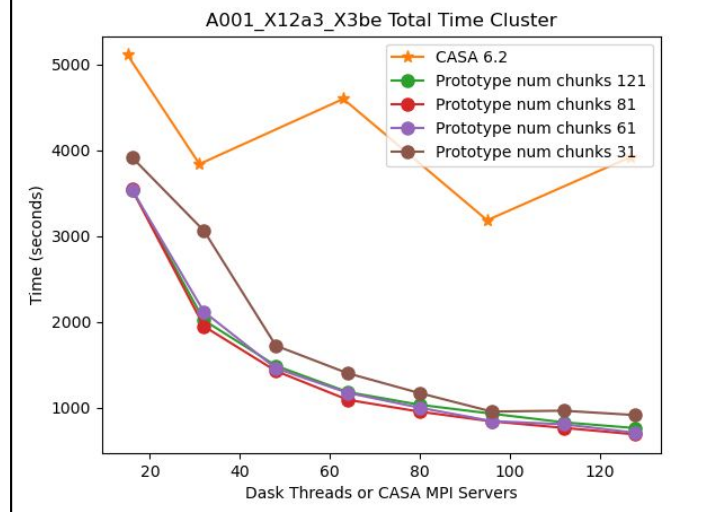
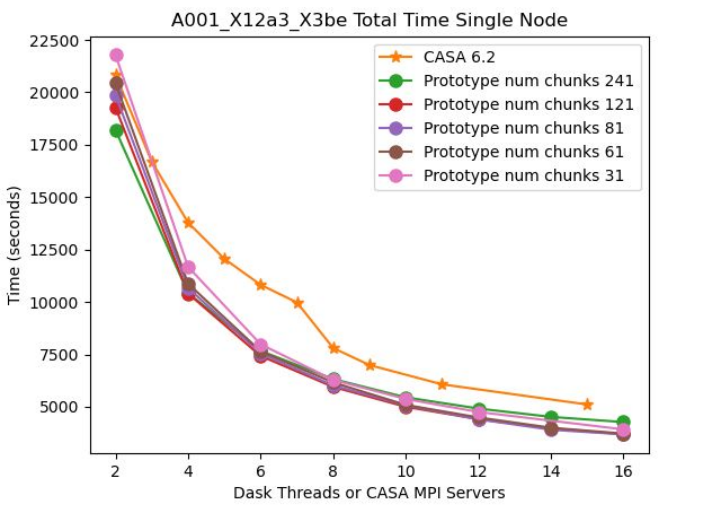
Jupyter Notebook Example

CASA for the ngVLA

Visibility and Image Parallel Execution Reduction (VIPER) Framework



uid://A001/X12a3/X3be,
Mosaic
Data: 305 GB



Documentation

Python: docs.python.org/

iPython: ipython.readthedocs.io/

CASA Guides: casaguides.nrao.edu

CASA Docs: <https://casadocs.readthedocs.io/>

Measurement Set: casa.nrao.edu/Memos/229.html

Table Query Language: casa.nrao.edu/aips2_docs/notes/199



www.nrao.edu
science.nrao.edu
public.nrao.edu

*The National Radio Astronomy Observatory is a facility of the National Science
Foundation
operated under cooperative agreement by Associated Universities, Inc.*

Parallel Processing: Modular CASA

```
$ MPICC=/path/to/mpicc pip installcasampi
```