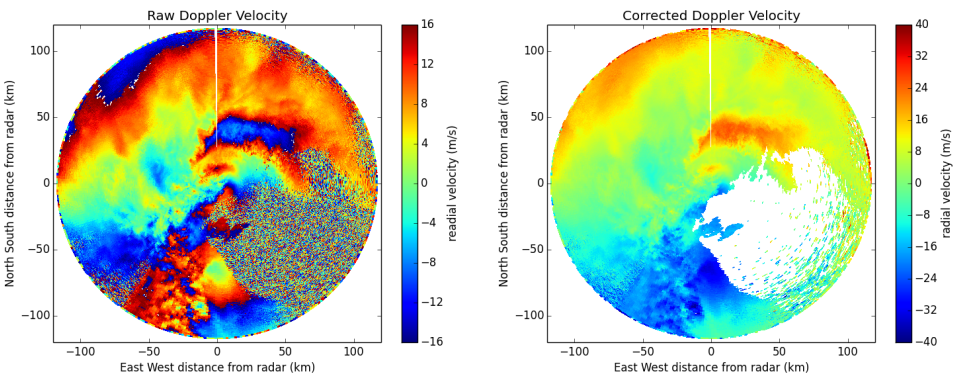
# Tools and Techniques for Developing Atmospheric Python Software: Insight from the Python ARM Radar Toolkit

Jonathan Helmus[1], Scott Giangrande[2], Kirk North[3], and Scott Collis[1]

[1] Argonne National Laboratory
[2] Brookhaven National Laboratory
[3] McGill University



# Introduction to Py-ART

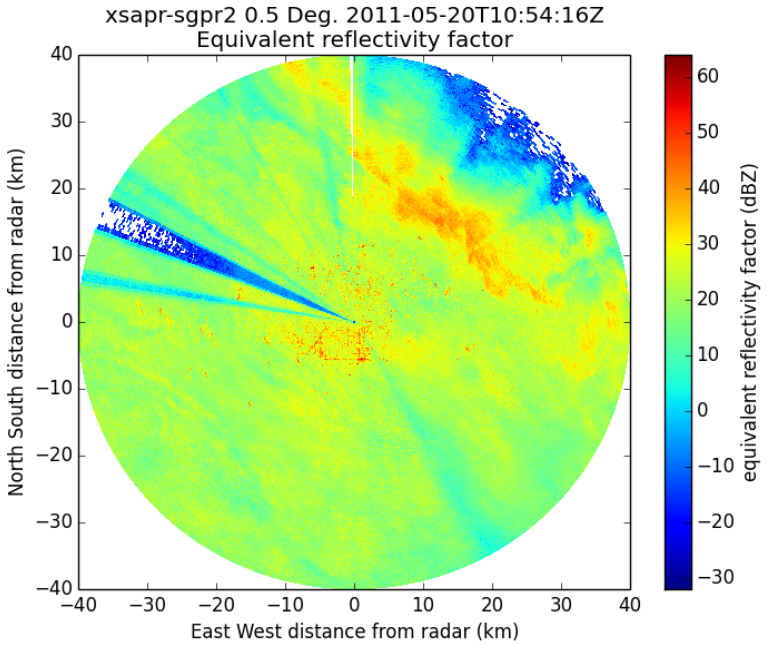
- Py-ART is a **Python module** for plotting, correcting and analyzing weather radar data.
- Development began to address the needs of the Atmospheric Radiation Measurement (**ARM**) Climate Research Facilities with acquisition of a number of new **scanning cloud and precipitation radars** as part of the American Recovery Act.
- The project was expanded to work with a **variety of weather radars** and a **wider user base** including radar researchers and climate modelers.
- Has been released on **GitHub** as **open source** software with a **BSD license**.
- Details of the project can be found online: **http://arm-doe.github.io/pyart/**
- **Contributions** from others are welcomed and encouraged!

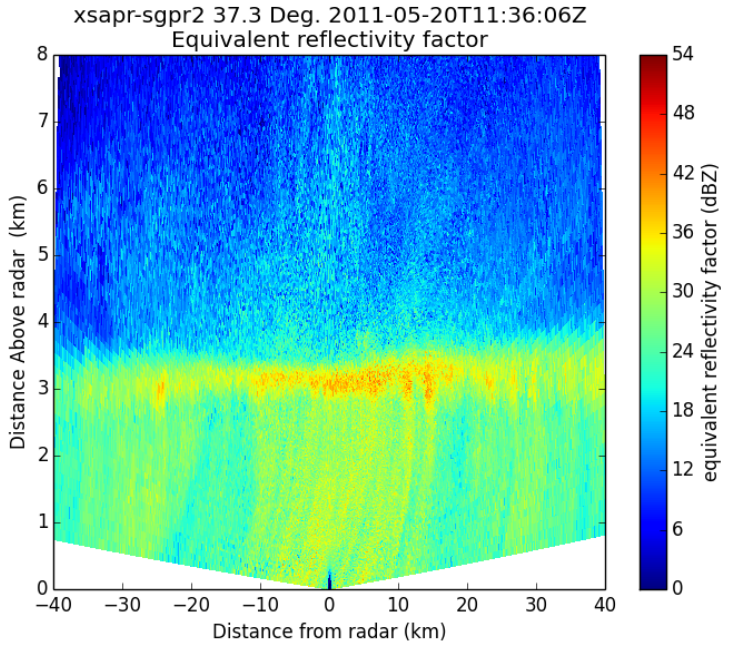
ARM CLIMATE RESEARCH FACILITY

# Py-ART: Plotting

```
import matplotlib.pyplot as plt
import pyart
radar = pyart.io.read('XSW110520105408.RAW7HHF')
display = pyart.graph.RadarDisplay(radar)
display.plot('reflectivity', 0, vmin=-32, vmax=64.)
plt.savefig('ppi_plot.png')
```

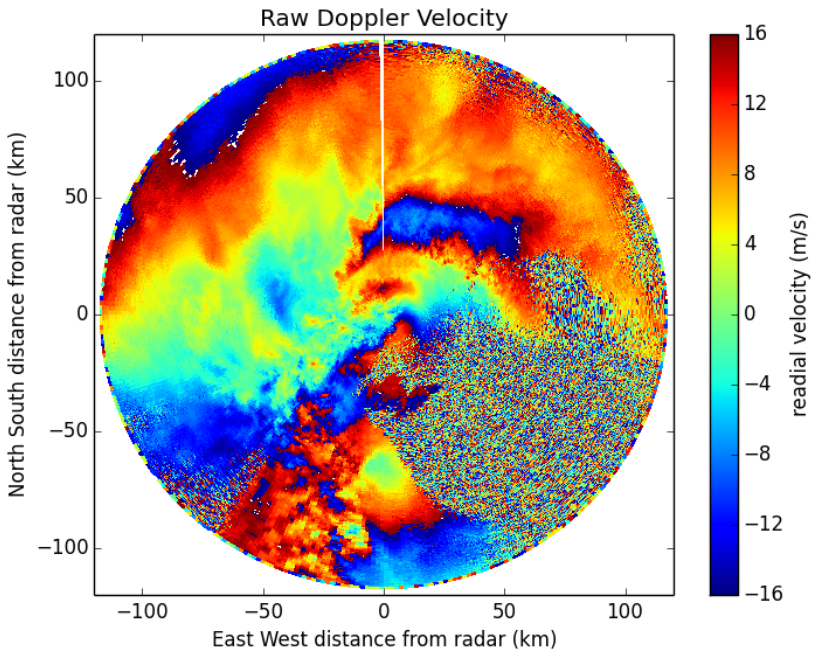
```
import matplotlib.pyplot as plt
import pyart
radar = pyart.io.read('XSW110520113537.RAW7HHL')
display = pyart.graph.RadarDisplay(radar)
display.plot('reflectivity', 0, vmin=0, vmax=54.)
display.set_limits(ylim=(0, 8))
plt.savefig('rhi_plot.png')
```

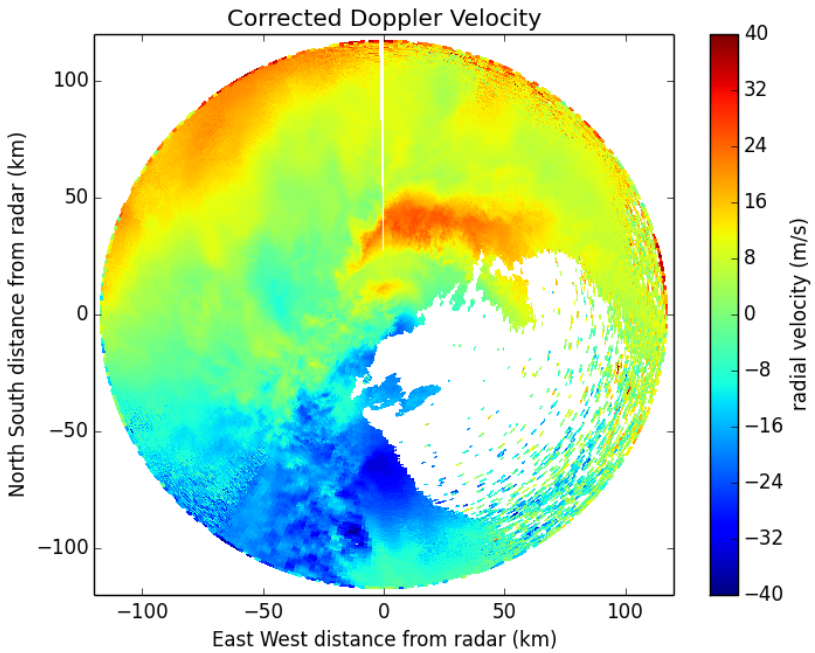


# Py-ART: Correct - Doppler Velocity Dealiasing

```
...
from pyart.correct import dealias_fourdd
dealias_data = dealias_fourdd(radar, height * 1000., speed, direction, target)
radar.add_field('corrected_velocity', dealias_data)
...
```

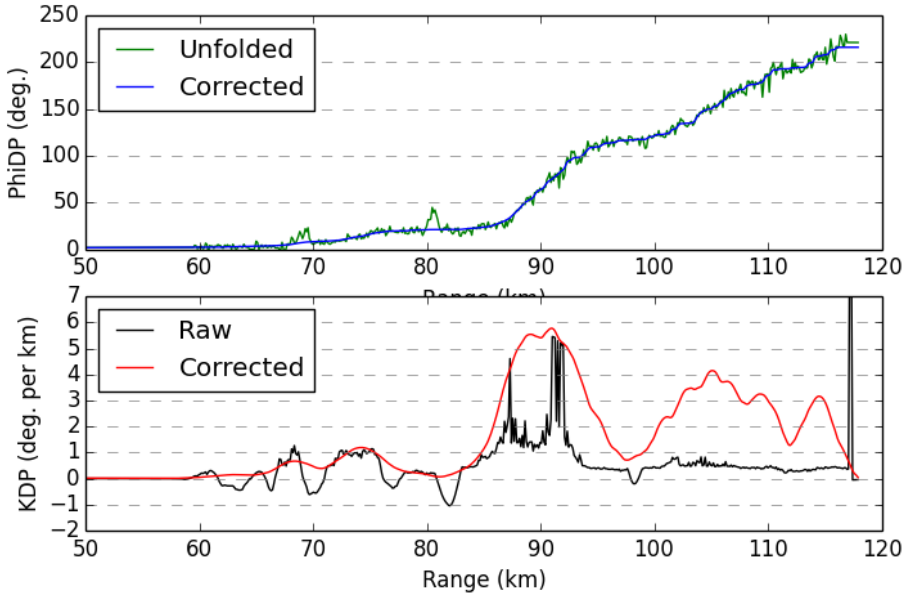


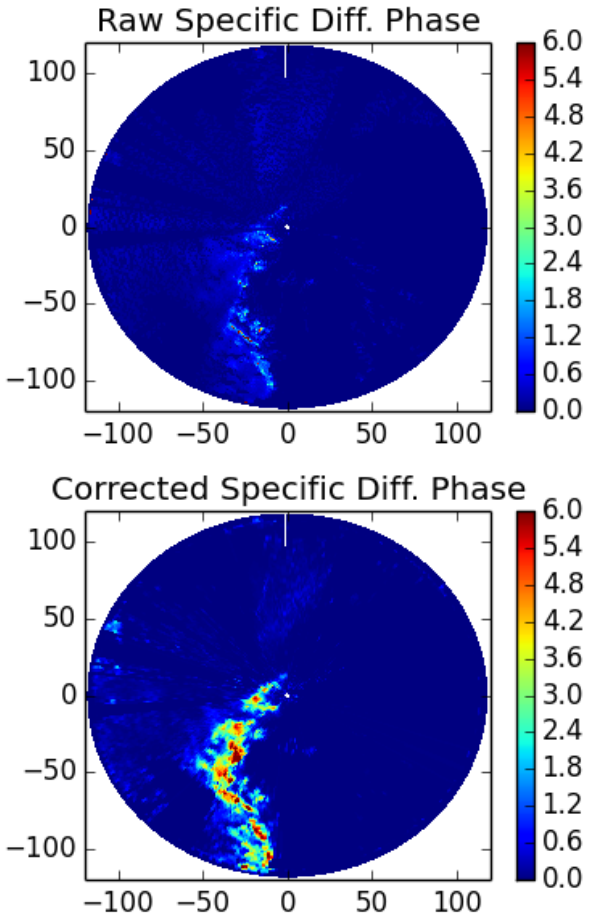




Dealiasing performed using U. Washington FourDD algorithm (James and Houze, JTech, 2001)

# Py-ART: Correct - LP phase processing

```
import pyart
radar = pyart.io.read('095636.mdv')
phidp, kdp = pyart.correct.phase_proc_lp(radar, 0.0)
radar.add_field('corrected_differential_phase', phidp)
radar.add_field('corrected_specific_diff_phase', kdp)
...
```

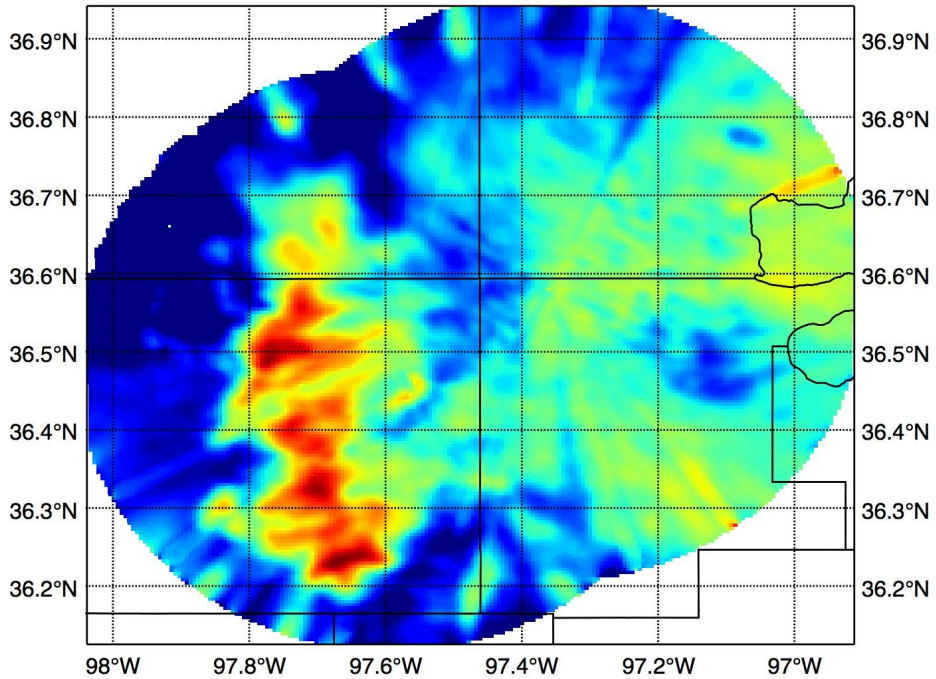



Phase processing performed using a LP based algorithm (Giangrande *et al*, JTech, 2013)

# Py-ART: Mapping to Cartesian Grids

```
import pyart
radar_sw = pyart.io.read(XSAPR_SW_FILE)
radar_se = pyart.io.read(XSAPR_SE_FILE)
grid = pyart.map.grid_from_radars((radar_se, radar_sw), grid_shape=(1, 201, 201),
        grid_limits=((1000, 1000), (-50000, 40000), (-60000, 40000)),
        grid_origin = (36.57861, -97.363611), max_refl=100.)
```



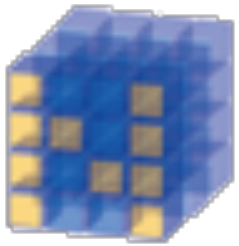

Gridding using KD-Tree nearest neighbor lookup and Cressman interpolation.

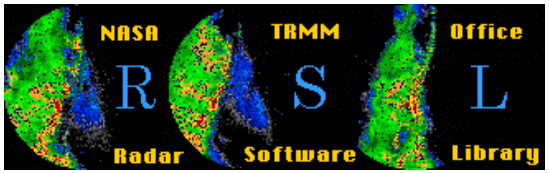
# Building Py-ART: SciPy Stack and other libraries

Py-ART utilized a number of open source Python modules and other libraries.

- Python modules from the Scientific Python stack:
  - NumPy
  - matplotlib
  - SciPy
- Specialized Python modules
  - netcdf4-python
  - basemap
- Radar specific libraries.
  - TRMM RSL
  - U. Wash. FourDD







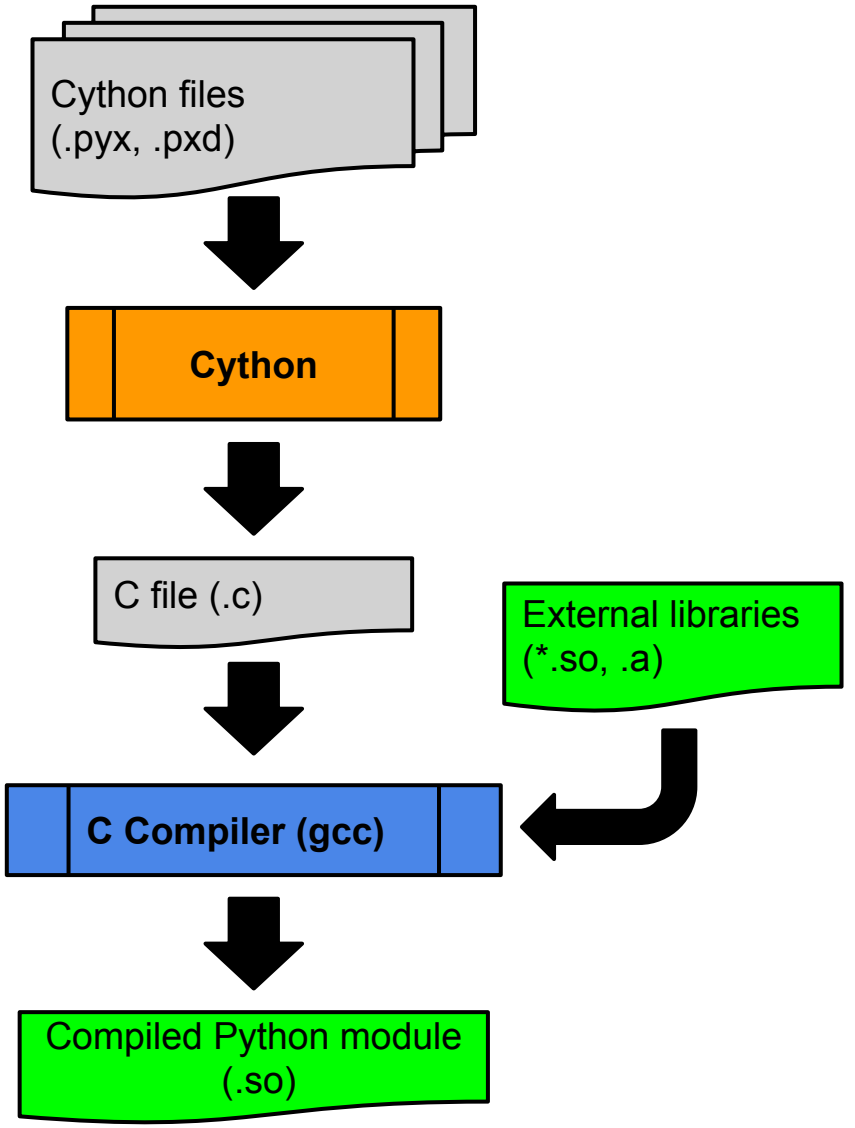
# Building Py-ART: Interfacing with legacy code

Py-ART uses existing libraries which were not designed to work with Python. Luckily tools exist to interact with many languages:

- F2PY : Fortran from Python
- Cython : C/C++ from Python

**Cython**

- Python to C code translator.
- Generates a Python extension module.
- Language additions make it easy to interact with C/C++ functions and classes
- Can also be used to speed up Python code by adding static types



# Building Py-ART: Cython Example - Cython side

```
cdef extern from "rsl.h":

    ctypedef struct Radar:
        Radar_header h
        Volume **v

    ctypedef struct Radar_header:
        int month, day, year
        int hour, minute
        float sec
        ...

    ctypedef struct Volume:
        Volume_header h
        Sweep **sweep
        ...
    ...

    Radar * RSL_anyformat_to_radar(char *infile)
    ...
    void RSL_free_volume(Volume *v)
    void RSL_free_radar(Radar *r)
```

_rsl_h.pxd

```
cimport _rsl_h

cdef class RslFile:
    cdef _rsl_h.Radar * _Radar
    cdef _rsl_h.Volume * _Volume

    def __cinit__(self, filename):
        self._Radar = _rsl_h.RSL_anyformat_to_radar(filename)
        if self._Radar is NULL:
            raise IOError('file cannot be read.')

    def __dealloc__(self):
        _rsl_h.RSL_free_radar(self._Radar)

    def get_volume(self, int volume_number):
        rslvolume = _RslVolume()
        rslvolume.load(self._Radar.v[volume_number])
        return rslvolume
    ...
    property month:
        def __get__(self):
            return self._Radar.h.month
        def __set__(self, int month):
            self._Radar.h.month = month
```

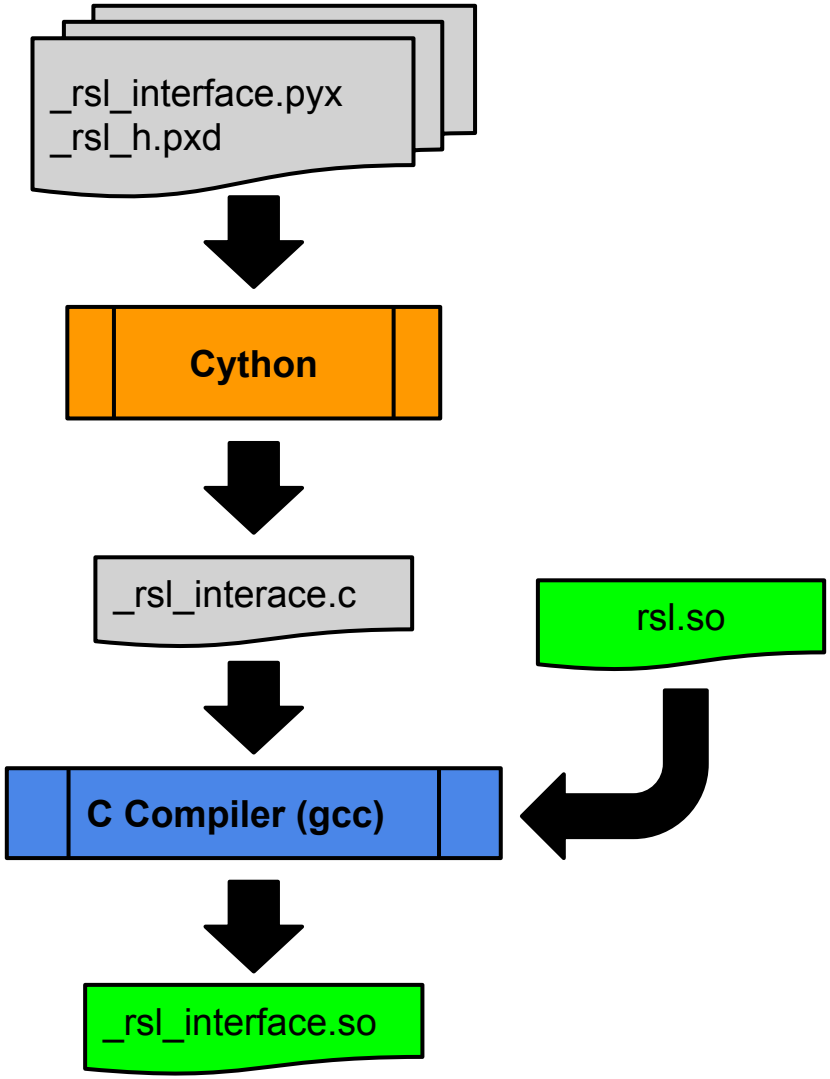
_rsl_interface.pyx



# Building Py-ART: Cython Example - Python side

```
>>> from pyart.io import _rsl_interface
>>> rslfile = _rsl_interface.RslFile('XSW110520105408.RAW')

>>> print rslfile
<pyart.io._rsl_interface.RslFile object at 0x107112d20>
>>> print rslfile.month
5
>>> rslfile.month = 12
>>> print rslfile.month
12
>>>
>>> volume = rslfile.get_volume(1)
>>> print volume
<pyart.io._rsl_interface._RslVolume object at 0x100493760>
```



# Software Engineering: Version Control

**Version control** is the method of recording changes over time to the source code and content of a project. Changes can then be examined or undone.

**Benefits:**

- Backup
- Reproducibility
- Collaboration


**Tools:**

- git
- Mercurial
- SVN


```
$ git log
commit dca8e348ba7db19f675747e8b98acf987d634915
Author: Jonathan Helmus <jjhelmus@gmail.com>
Date:   Fri Jan 24 11:46:29 2014 -0600

       ENH: pyproj no longer an optional dependency

       pyproj is included in basemap, use the version ...

commit 26c7a36dadfbdcc1c4e1dc1d8bfe188a073e08a4
Author: Jonathan Helmus <jjhelmus@gmail.com>
Date:   Mon Jan 20 15:49:15 2014 -0600

       TST: do not install cylp or cvxopt in python 2.6

commit 1eeed39faf89d4aa9eda6014c4dbc1a2458bbb9a
Author: Jonathan Helmus <jjhelmus@gmail.com>
Date:   Mon Jan 20 15:42:24 2014 -0600

       TST: do not all basemap in python 2.6

commit d1ce060537df08aad39b8ad3a75db13a9728aa1a
Author: Jonathan Helmus <jjhelmus@gmail.com>
Date:   Mon Jan 20 15:35:14 2014 -0600

       BUG: fixed bugs when running in Python 2.6
```

# Software Engineering: Testing

**Testing** is the practice of writing code that verifies the functionality of one or more components by executing the software with fixed inputs and checking that the results are correct.

**Benefits:**

- Fewer mistakes and bugs.
- Bugs are not reintroduced.
- Help avoid "it works, don't touch it."

**Tools:**

- nose
- unittest/doctest
- pytest

```python
def test_radar_creation():
    radar = pyart.testing.make_target_radar()
    assert isinstance(radar, pyart.io.Radar)

def test_add_field():
    radar = pyart.testing.make_target_radar()
    dic = {'data': np.zeros((360, 50)),
           'standard_name': 'test'}
    radar.add_field('test', dic)
    assert 'test' in radar.fields
    assert 'data' in radar.fields['test']
    assert radar.fields['test']['standard_name'] == 'test'
```

```
~/python/pyart$ nosetests -v
test_attenuation.test_attenuation ... ok
test_dealias.test_find_time_in_interp_sounde ... ok
...
test_config.test_filemetadata_custom ... ok
test_config.test_init_load ... ok
test_config.test_intergration ... ok
----------------------------------------------------------
Ran 386 tests in 20.666s

OK (SKIP=2)
```



# Software Engineering: Continuous Integration

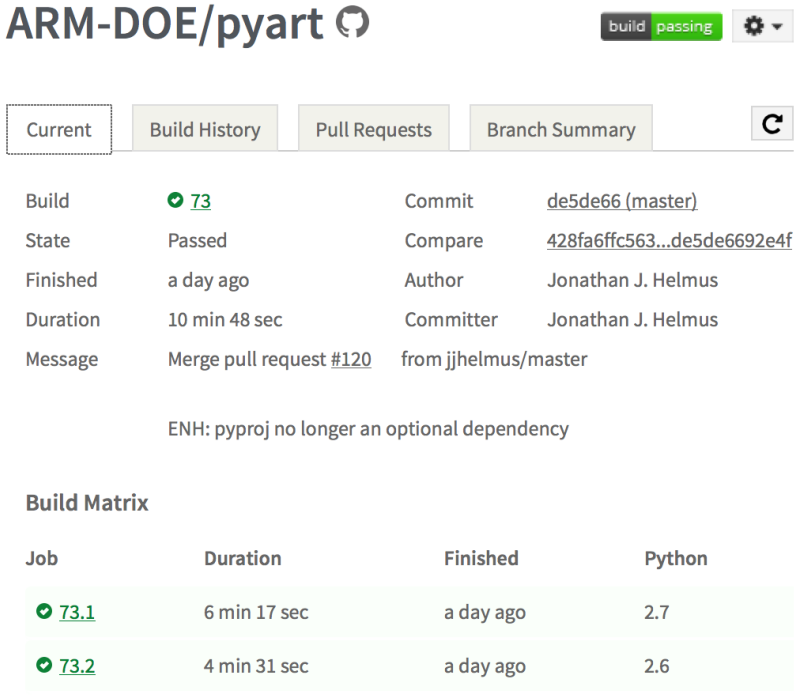
**Continuous Integration** refers to the automated process that builds a working copy of software from source and run a set of tests. This process is run "continuously", typically once a day or after every change to the source code.

**Benefits:**

- Quick testing and verification of all changes
- Bugs are caught early.
- A "current" build is always available.

**Tools:**

- Travis-CI
- Jenkins




# Software Engineering: Embedded Documentation

Reference **software documentation** can be embedded in the source code and then extracted and formatted into human-friendly documents by means of a documentation generator.

**Benefits**

- Documentation and code are changed at the same time.
- Multiple forms of documentation can be created from the same source.

**Tools:**

- Sphinx
- numpydoc
- Readthedocs

```
def grid_from_radars(radars, grid_shape, grid_limits, **kwargs):
    """
    Map one or more radars to a Cartesian grid returning a Grid object.

    Additional arguments are passed to :py:func:`map_to_grid`

    Parameters
    ----------
    radars : tuple of Radar objects.
        Radar objects which will be mapped to the Cartesian grid.
    grid_shape : 3-tuple of floats
        Number of points in the grid (z, y, x).
```

pyart.map.**grid_from_radars**(*radars*, *grid_shape*, *grid_limits*, ***kwargs*) ¶ [source]

Map one or more radars to a Cartesian grid returning a Grid object.

Additional arguments are passed to `map_to_grid`

**Parameters :**

**radars** : tuple of Radar objects.
Radar objects which will be mapped to the Cartesian grid.

**grid_shape** : 3-tuple of floats
Number of points in the grid (z, y, x).

**grid_limits** : 3-tuple of 2-tuples
Minimum and maximum grid location (inclusive) in meters for the z, x, y coordinates.

**Returns :**

**grid** : Grid
A `pyart.io.Grid` object containing the gridded radar data.

**See also:**

`map_to_grid`
Map to grid and return a dictionary of radar fields

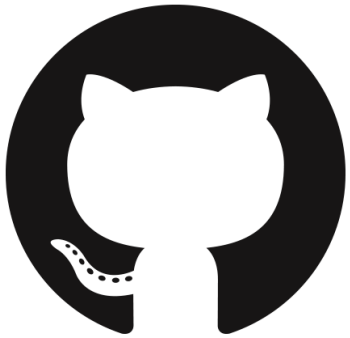


# GitHub



Py-ART uses GitHub for project hosting:

- Git repository for source code.
- Issue tracker.
- Documentation/webpage hosting.
- Wiki
- Collaboration (Pull Requests)
- Excellent integration with Travis-CI
- Free (for open source projects)



**https://github.com/ARM-DOE/pyart**

Similar features can be found at BitBucket and other open source project hosting services.

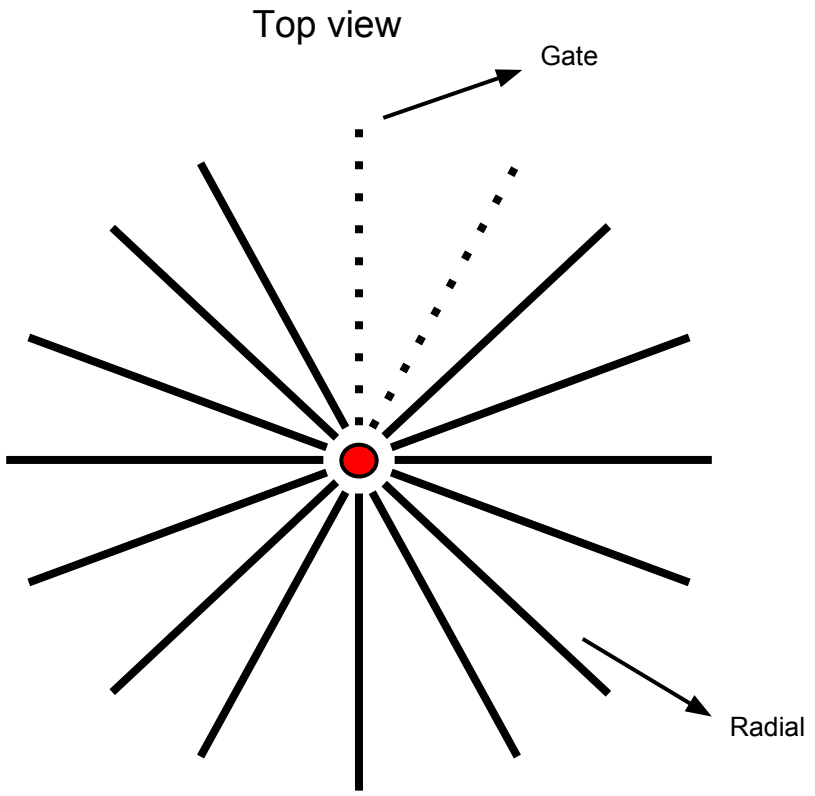
# Py-ART: File I/O

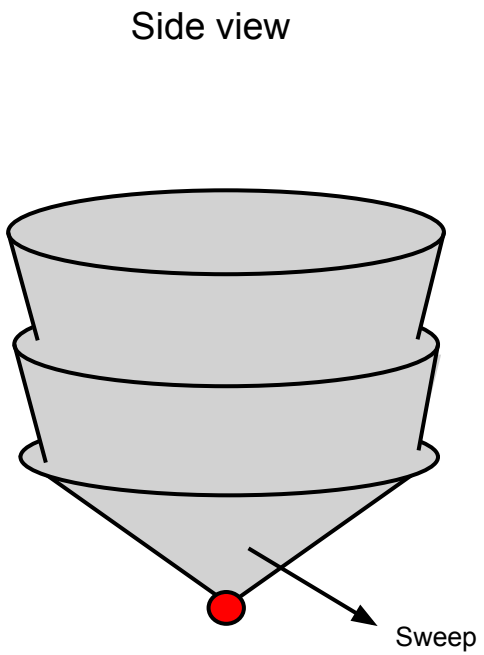
- Py-ART can read radar data in a number of formats in a Radar object. A number of formats are supportive natively and additional formats can be read if the TRMM RSL library is installed.

| Format | Native Read | Read w/ RSL |
|---|---|---|
| **Sigmet/IRIS** | X | X |
| **MDV** | X | |
| **Cf/Radial** | X (+writing) | |
| **Universal (UF)** | | X |
| **Lassen** | | X |
| **NEXRAD Level II** | X | |
| **DORAD** | | X |



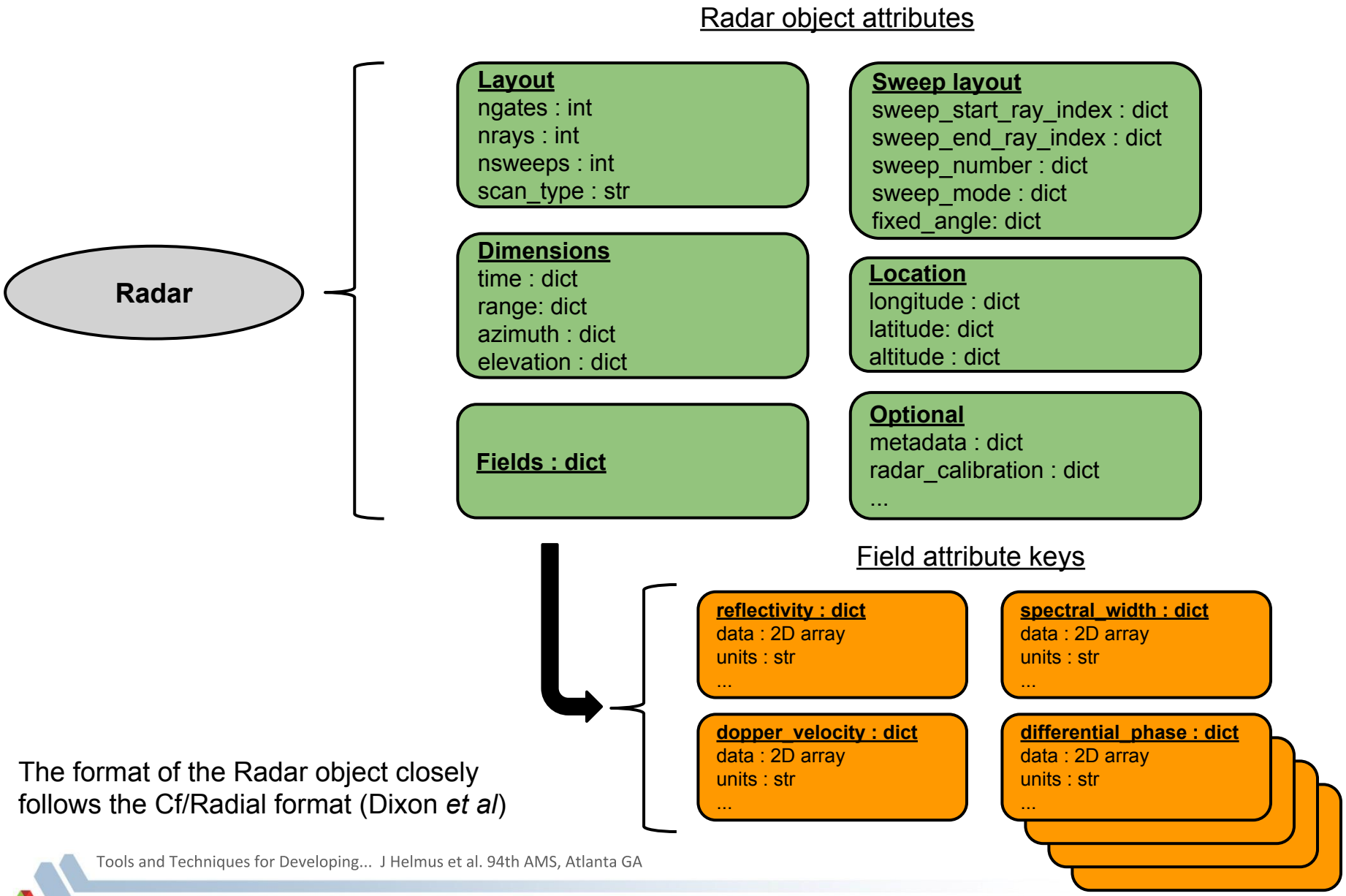
# Radar 101




**Radar Volume**
- **Sweep**
  - Radial
  - Radial
  - Radial
  - **Radial**
    - Gate
    - Gate
    - Gate
    - Gate
    - Gate
- Sweep
- Sweep

# Py-ART: Data layout




The format of the Radar object closely follows the Cf/Radial format (Dixon *et al*)