HWRF v3.7a Tutorial
College Park, MD, Jan 26, 2016

Python Scripts in HWRF

Christina Holt – DTC, ESRL/GSD, CIRES

Many slides contributed by
Sam Trahan

Developmental Testbed Center

Outline

- Resources for Users
- System design
- Object-oriented programming basics
- Configuring HWRF
- Data communication
- Logging

Resources for Users

User webpage

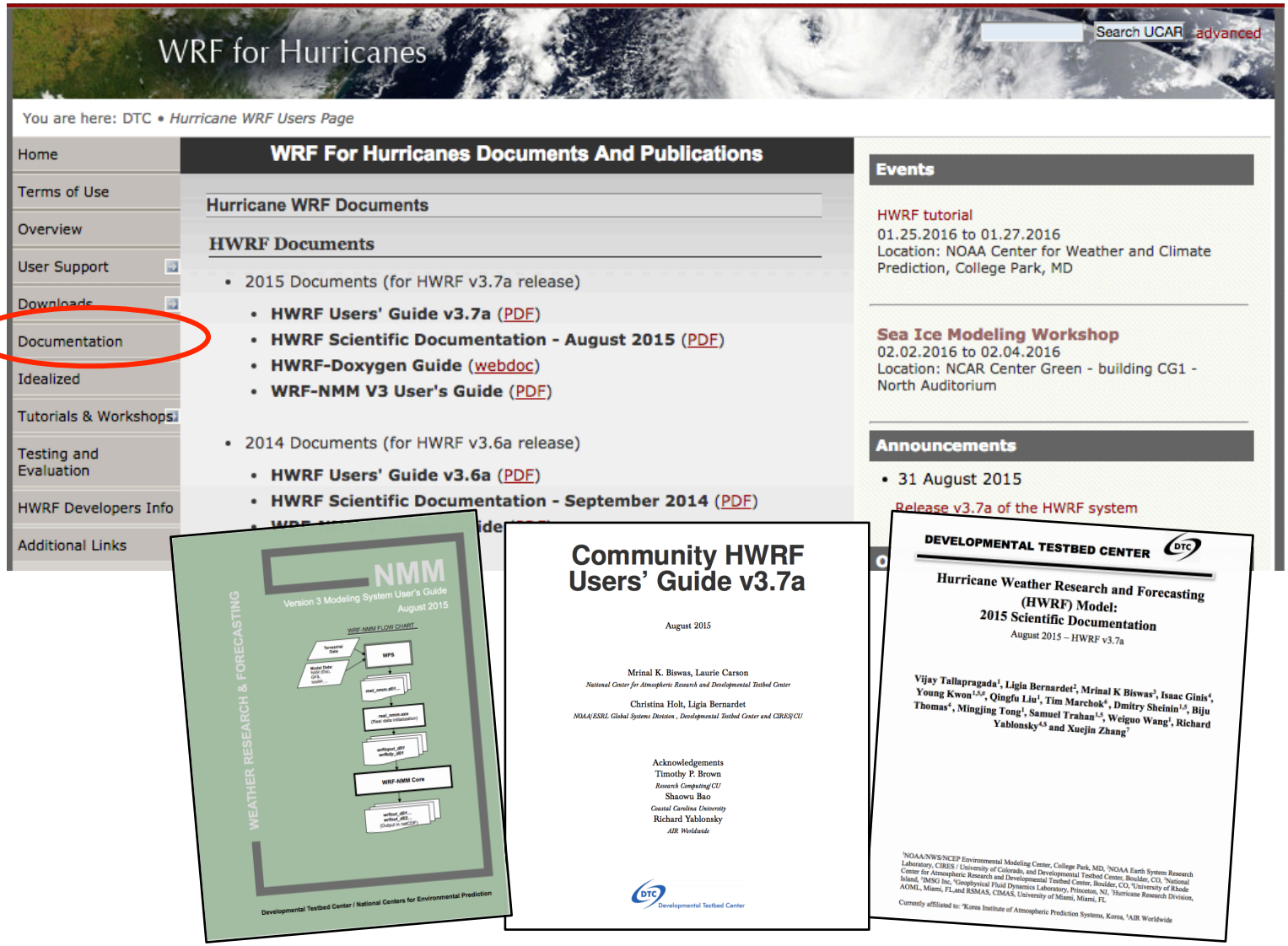
Documentation

Doxygen website

Python website

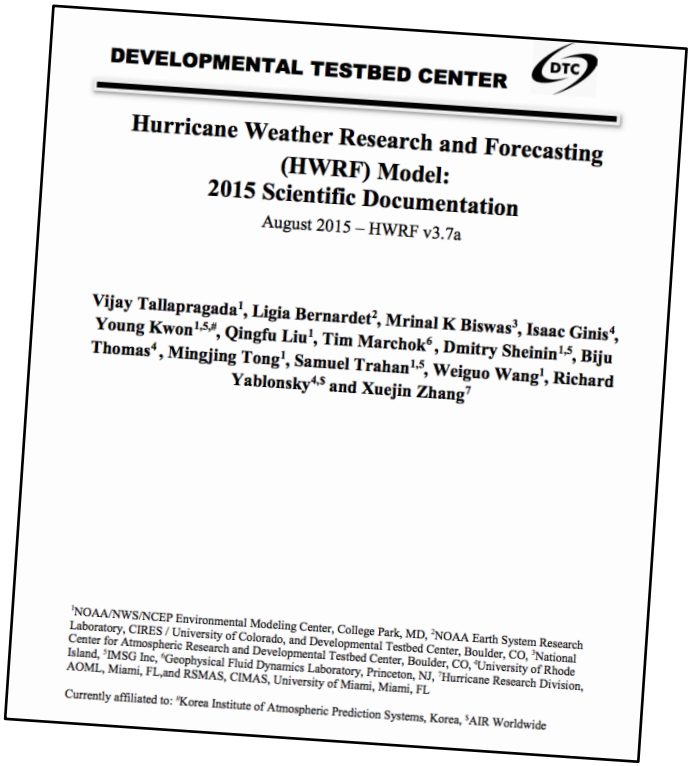
www.dtcenter.org/HurrWRF/users

User support webpage



Scientific Documentation

- Technical information covering each HWRF component
  - Authorship includes developers and experts
  - Chapters covering:
    - HWRF introduction
    - HWRF Initialization
    - MPI POM-TC
    - Physics Packages in HWRF
    - Design of moving nest
    - Use of GFDL Vortex Tracker
    - The idealized HWRF framework



DEVELOPMENTAL TESTBED CENTER

Hurricane Weather Research and Forecasting (HWRF) Model: 2015 Scientific Documentation

August 2015 – HWRF v3.7a

Vijay Tallapragada[1], Ligia Bernardet[2], Mrinal K Biswas[3], Isaac Ginis[4], Young Kwon[1,5,#], Qingfu Liu[1], Tim Marchok[6], Dmitry Sheinin[1,5], Biju Thomas[4], Mingjing Tong[1], Samuel Trahan[1,5], Weiguo Wang[1], Richard Yablonsky[4,5] and Xuejin Zhang[7]

[1]NOAA/NWS/NCEP Environmental Modeling Center, College Park, MD, [2]NOAA Earth System Research Laboratory, CIRES / University of Colorado, and Developmental Testbed Center, Boulder, CO, [3]National Center for Atmospheric Research and Developmental Testbed Center, Boulder, CO, [4]University of Rhode Island, [5]IMSG Inc, [6]Geophysical Fluid Dynamics Laboratory, Princeton, NJ, [7]Hurricane Research Division, AOML, Miami, FL,and RSMAS, CIMAS, University of Miami, Miami, FL

Currently affiliated to: [#]Korea Institute of Atmospheric Prediction Systems, Korea, [5]AIR Worldwide

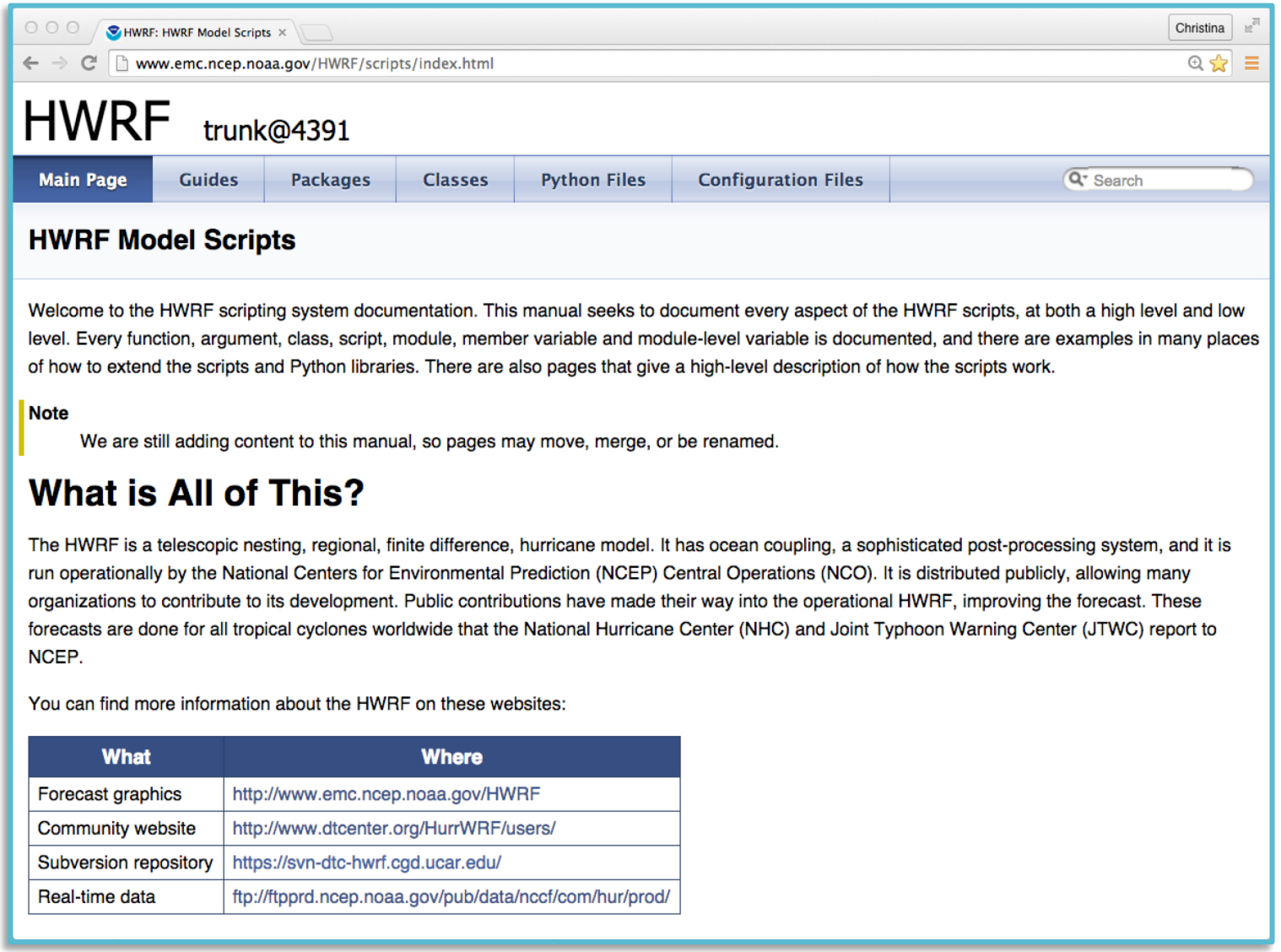
http://www.dtcenter.org/HurrWRF/users/docs/scientific_documents/HWRF_v3.7a_SD.pdf

HWRF v3.7a User's Guide

- Includes detailed instructions on running each component
  - Geared towards public release, so some aspects will be missing
  - Running with wrappers, no Rocoto information
- Content:
  - Introduction & software installation
  - Running HWRF
  - HWRF preprocessing system
  - Vortex Relocation
  - DA
  - Merge
  - MPIPOM-TC
  - Forecast Model
  - Post processor
  - Forecast products
  - Idealized

http://www.dtcenter.org/HurrWRF/users/docs/users_guide/HWRF_v3.7a_UG.pdf

Doxygen Website



General Python help

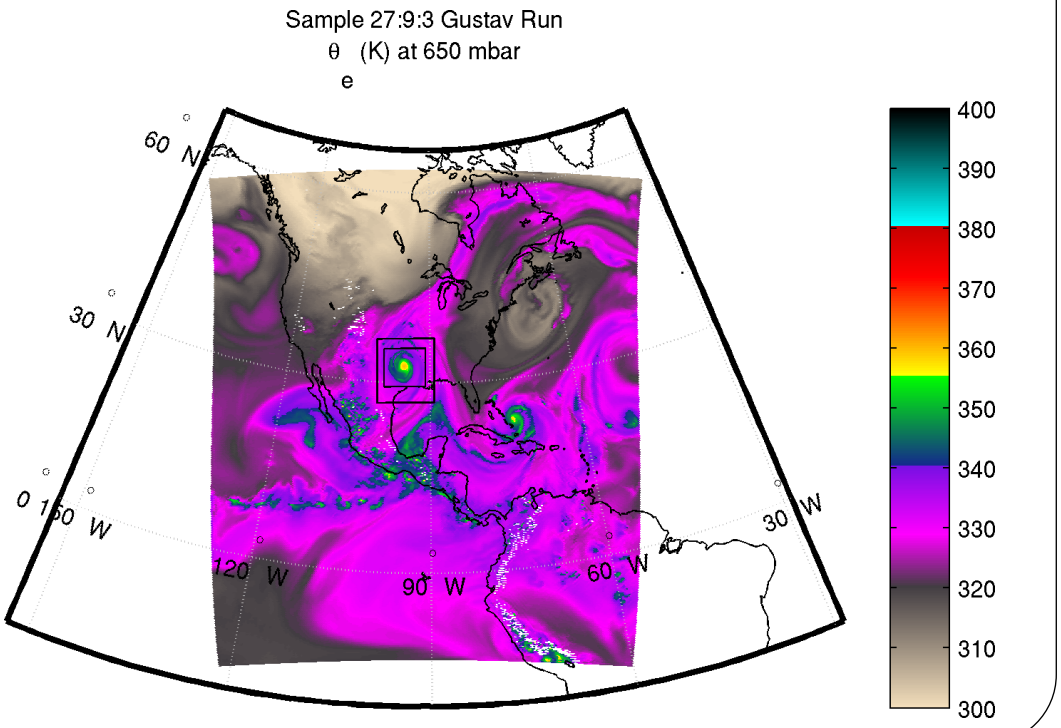
- Online (https://docs.python.org/release/2.6.6/)
- Open python in a terminal and use help() function for particular function. An example to get information with a Python list:

  ```
  $ python
  $ help(list)
  ```

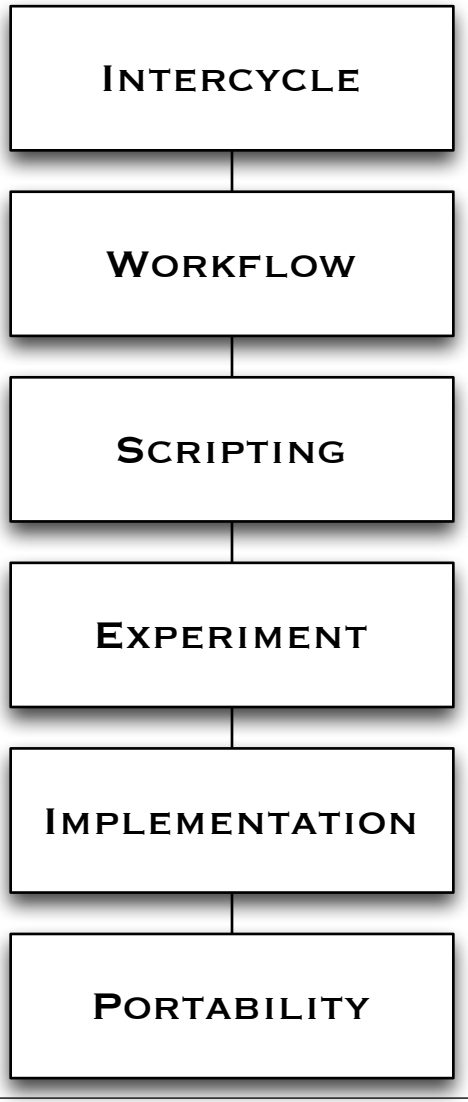
- Must use Python v2.6.6.
  - Only version available on NOAA machines.
  - 2.7 may be used in future because it's expected to have long-term support.
  - Version 3 is basically a different language.

HWRF System Overview

Overview of the system design

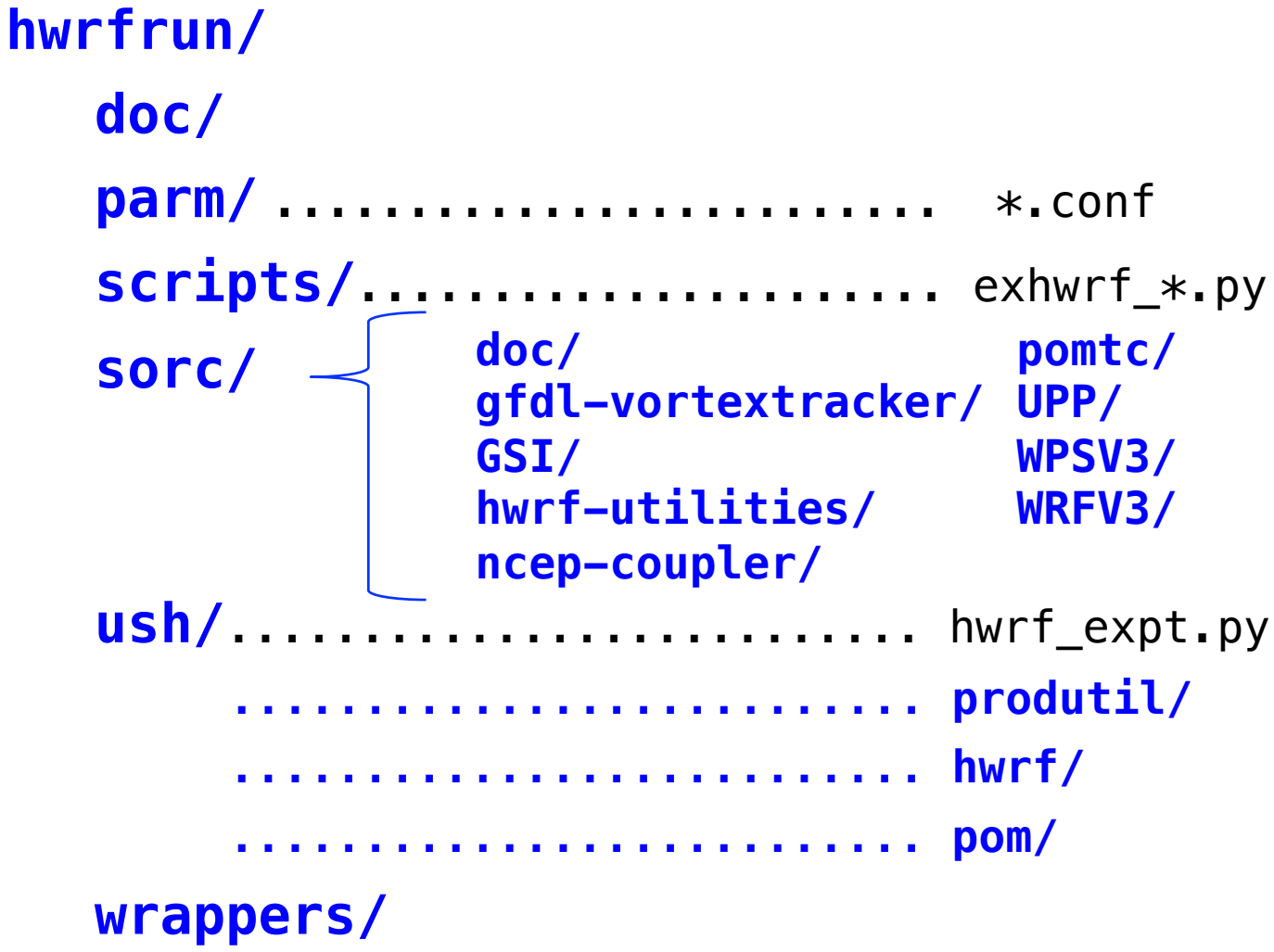


HWRF System: Overview

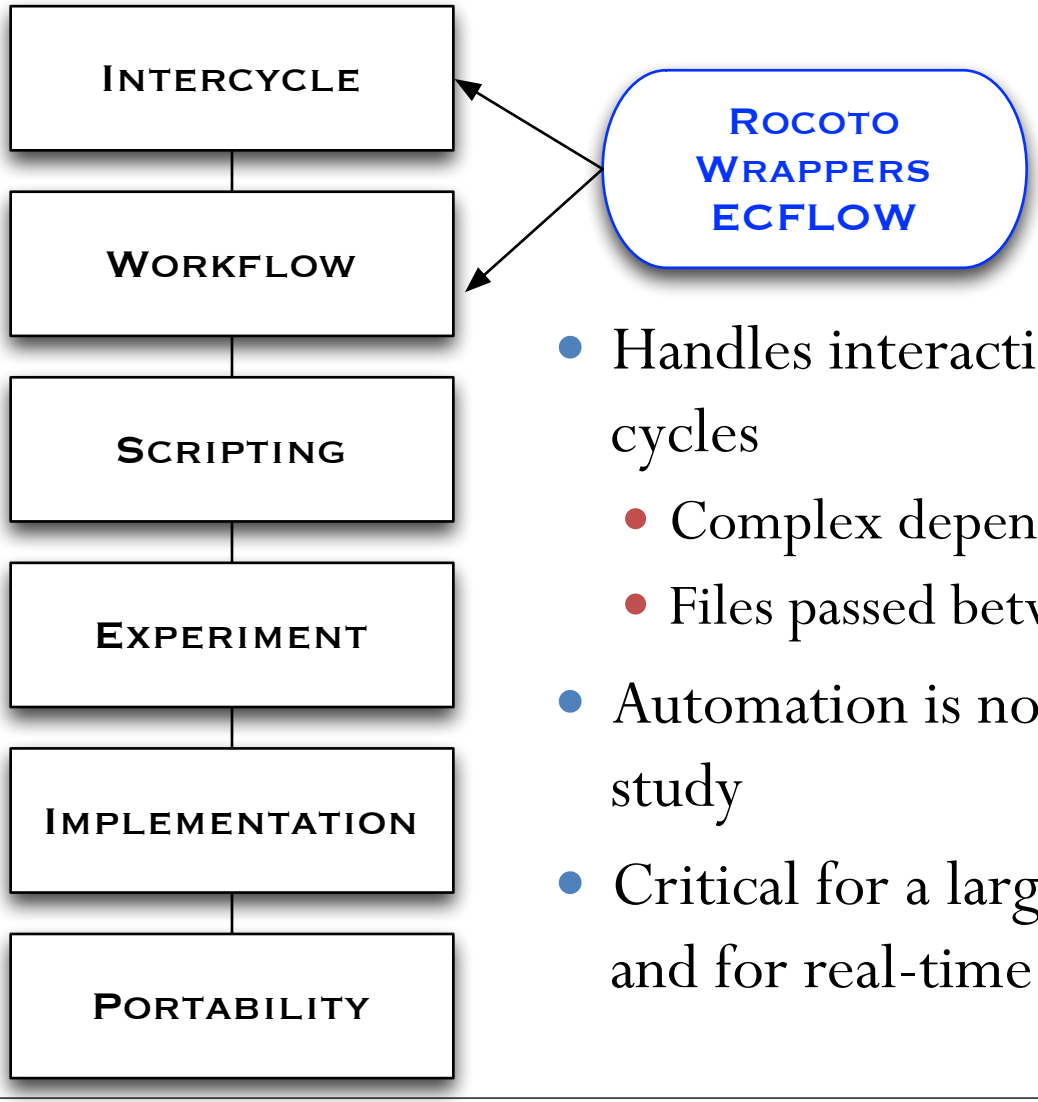


- 6 layers of scripts that are responsible for preparing the environment and data for and running the ~80 HWRF executables of the end-to-end system
- Most of these layers are written using an object-oriented (O-O) Python design
- O-O design makes the system highly configurable and reduces the footprint of the system drastically

HWRF Directory Structure

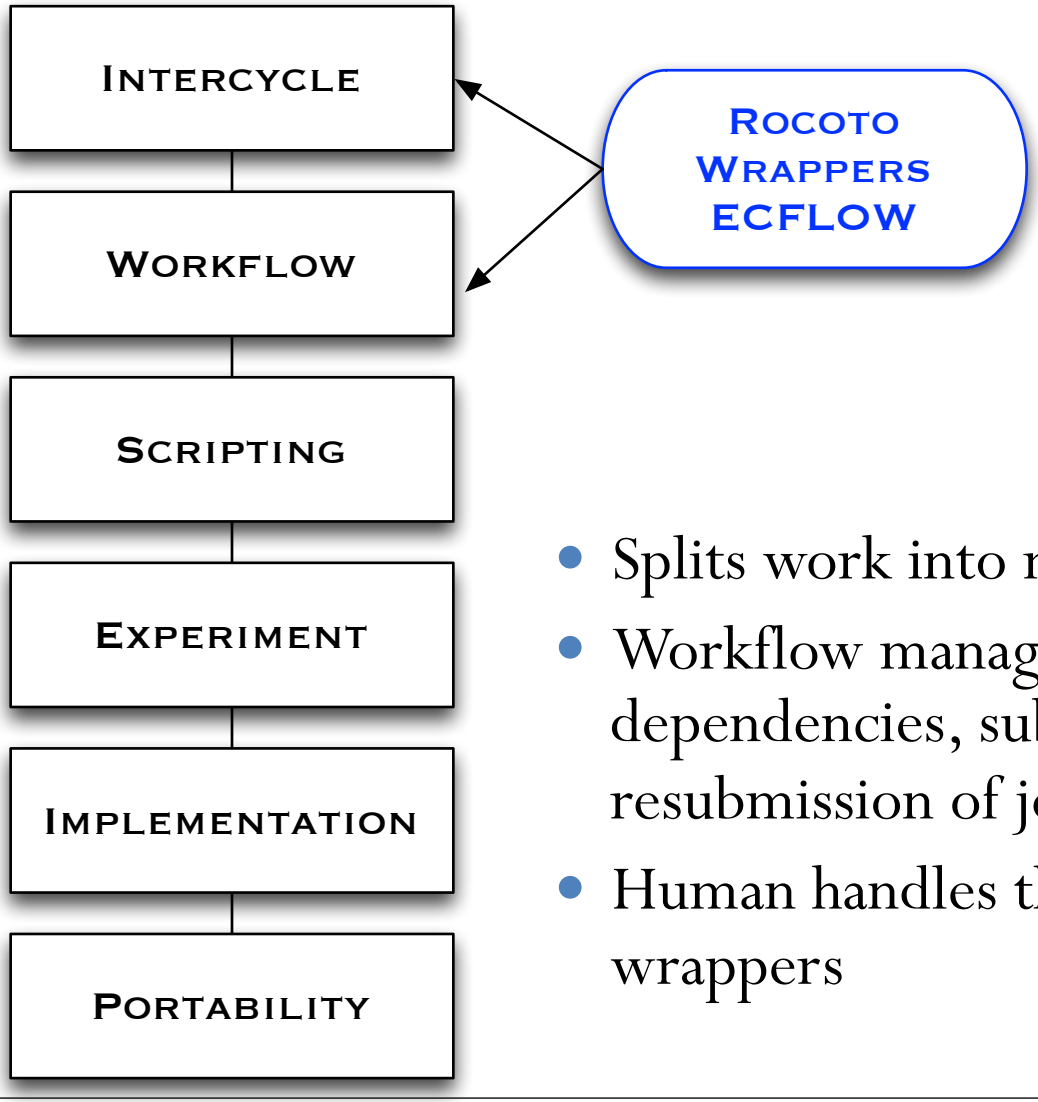


HWRF System: Intercycle Layer



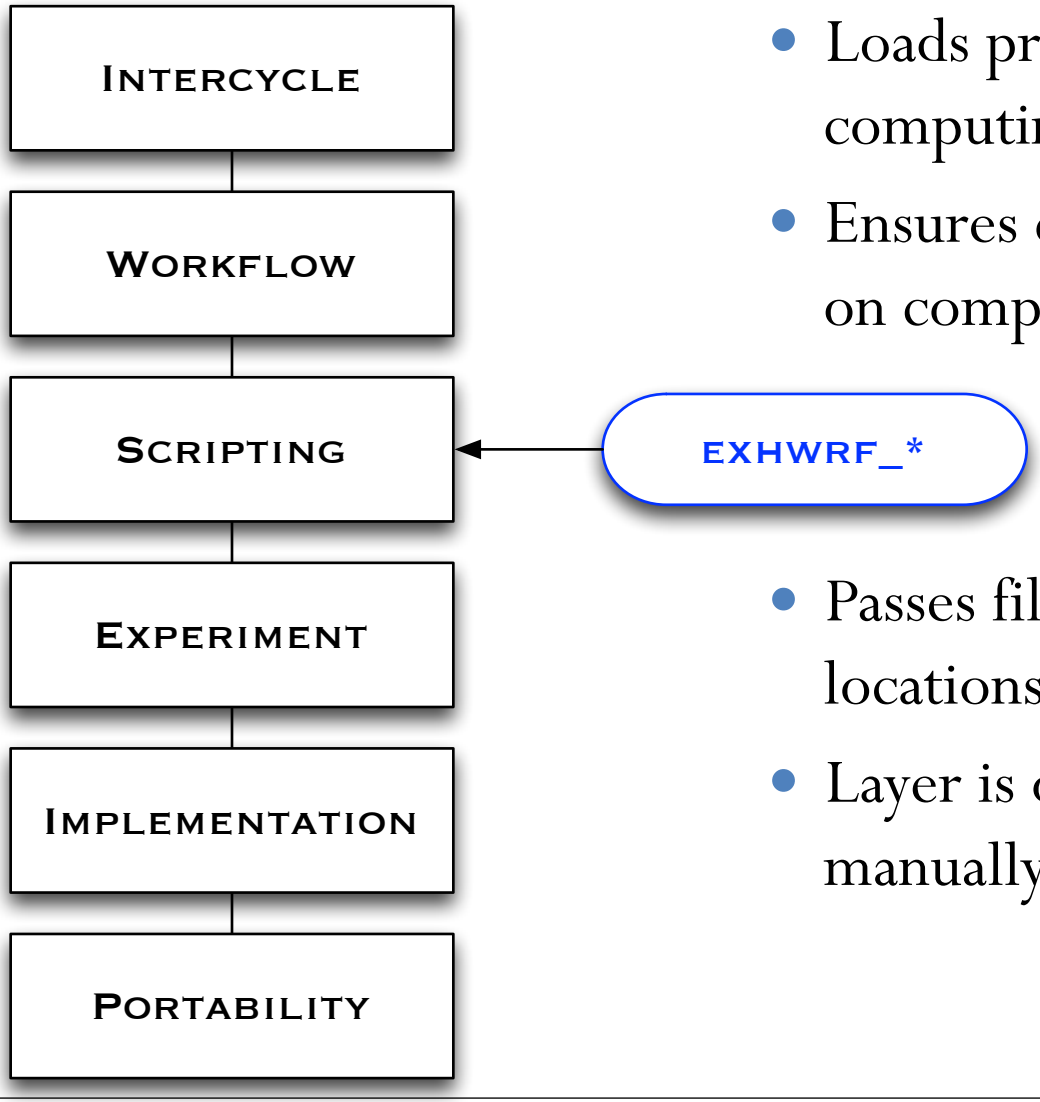
- Handles interactions between several cycles
  - Complex dependencies
  - Files passed between them
- Automation is not needed for a case study
- Critical for a large retrospective study, and for real-time automation

HWRF System: Workflow Layer



- Splits work into multiple batch jobs
- Workflow managers handle dependencies, submission, failures, and resubmission of jobs
- Human handles this process when using wrappers

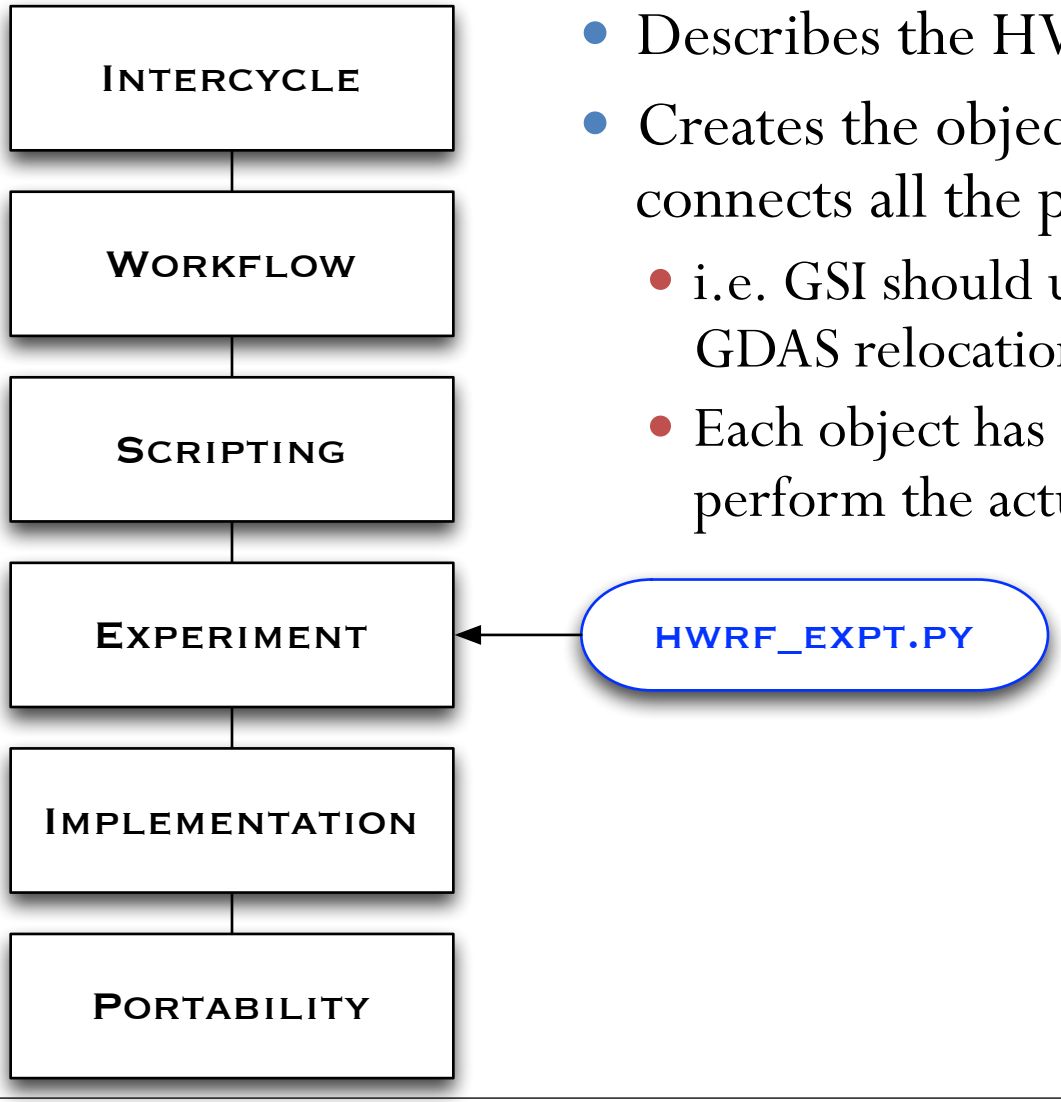
HWRF System: Scripting Layer



- Loads programs and libraries into computing environment
- Ensures connection to file system on compute node

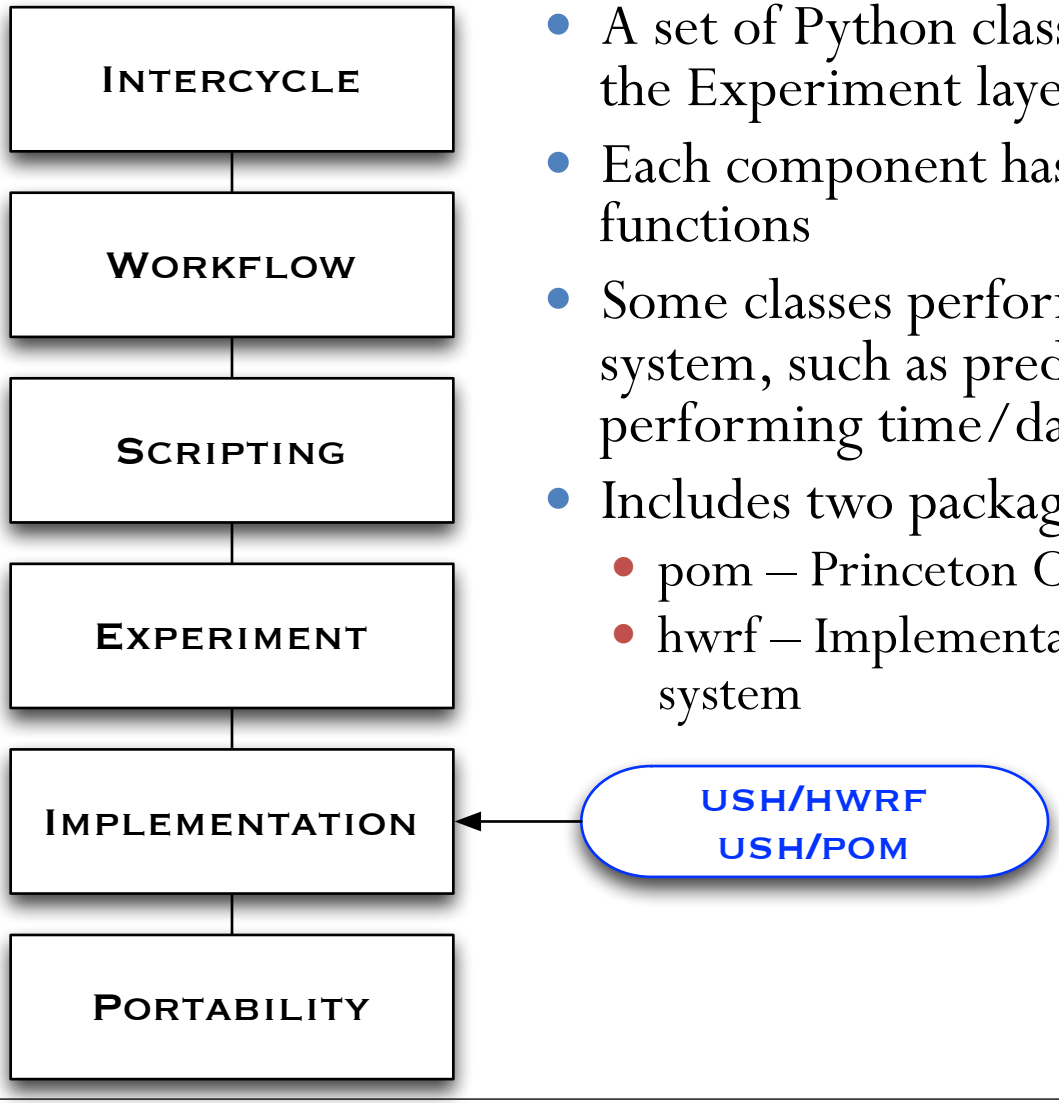
- Passes file and executable locations to the next lower layer
- Layer is optional – can be done manually by user

HWRF System: Experiment Layer



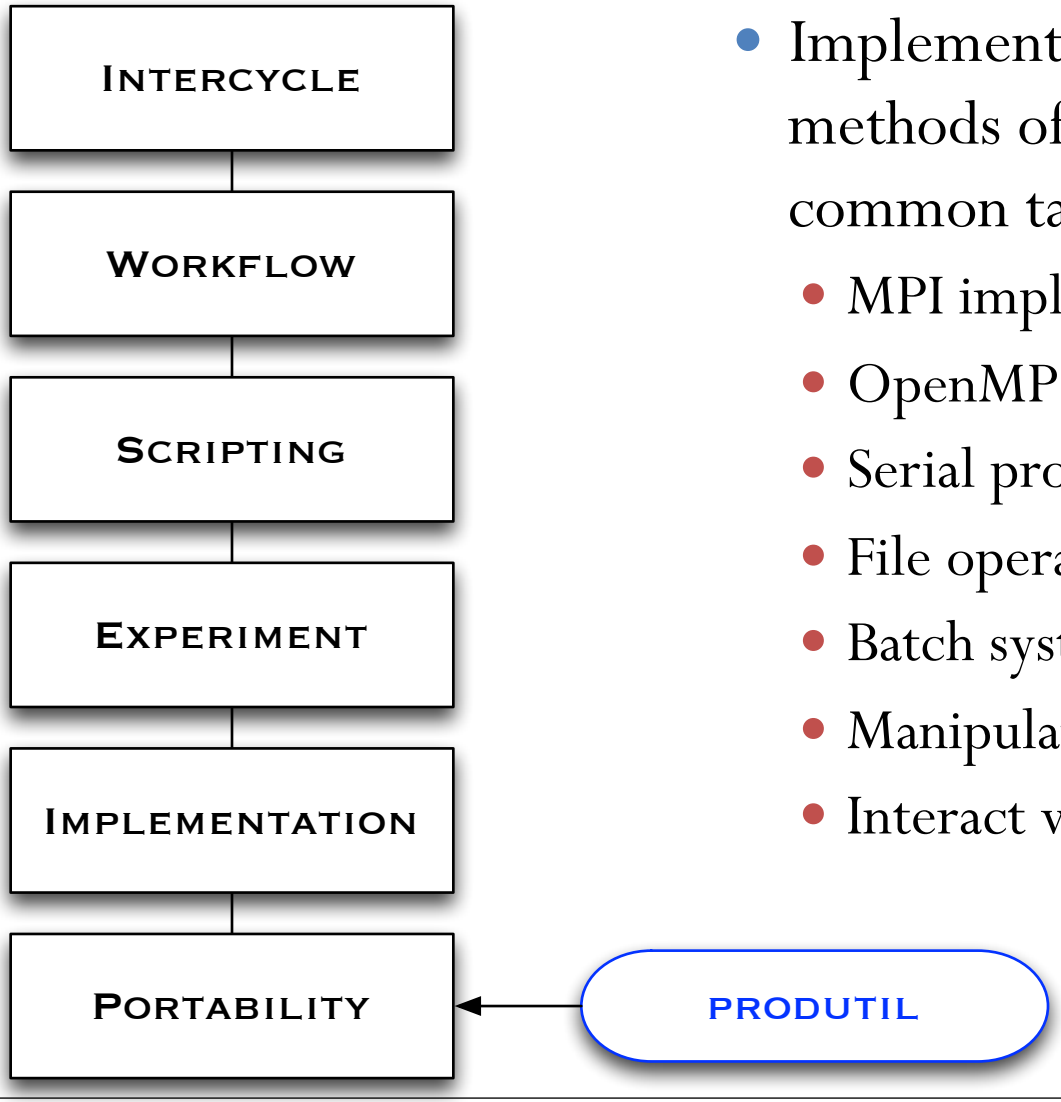
- Describes the HWRF workflow
- Creates the object structure that connects all the pieces
  - i.e. GSI should use input from the GDAS relocation output
  - Each object has a run() function to perform the actual task

HWRF System: Implementation Layer



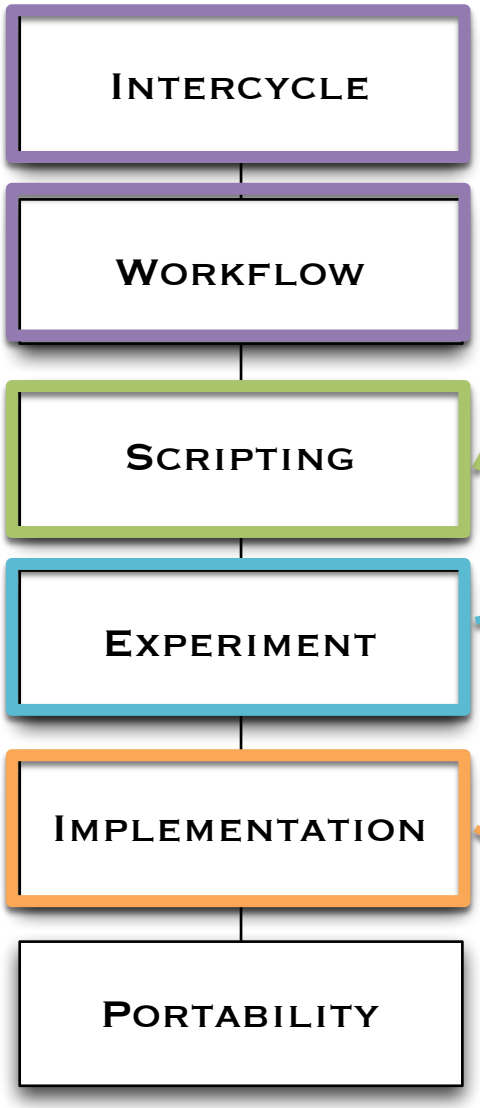
- A set of Python classes and functions used by the Experiment layer to run HWRF
- Each component has its own class and set of functions
- Some classes perform utilities to support the system, such as predicting filenames and performing time/date arithmetic
- Includes two packages
  - pom – Princeton Ocean Model initialization
  - hwrf – Implementation of most of the HWRF system

HWRF System: Portability Layer



- Implements cross-platform methods of performing common tasks
  - MPI implementation
  - OpenMP
  - Serial programs
  - File operations
  - Batch system interaction
  - Manipulate resource limitations
  - Interact with database file

Workflow Object Structure



wrappers/post_wrapper

export TOTAL_TASKS=24

$EXhwrf/exhwrf_post.py

scripts/exhwrf_post.py:

import hwrf_expt

hwrf_expt.init_module()

hwrf_expt.post.run_post()

ush/hwrf_expt.py:

post=HWRFPost('/path/to/infile',

'/path/to/fixd', '/path/to/hwrf_post',

to_datetime('2015081818'))

ush/hwrf/post.py:

class HWRFPost

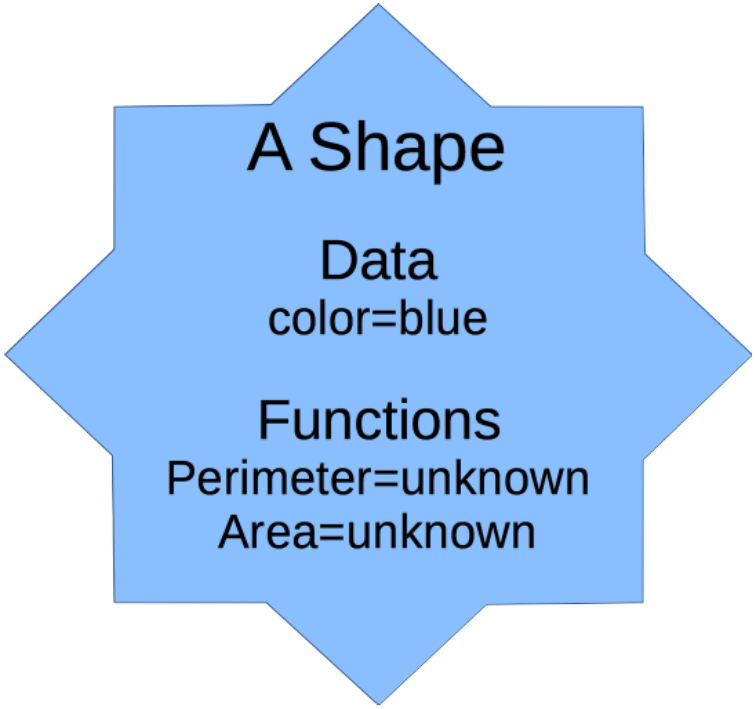
def run_post

Object-oriented Programming

An example for HWRF

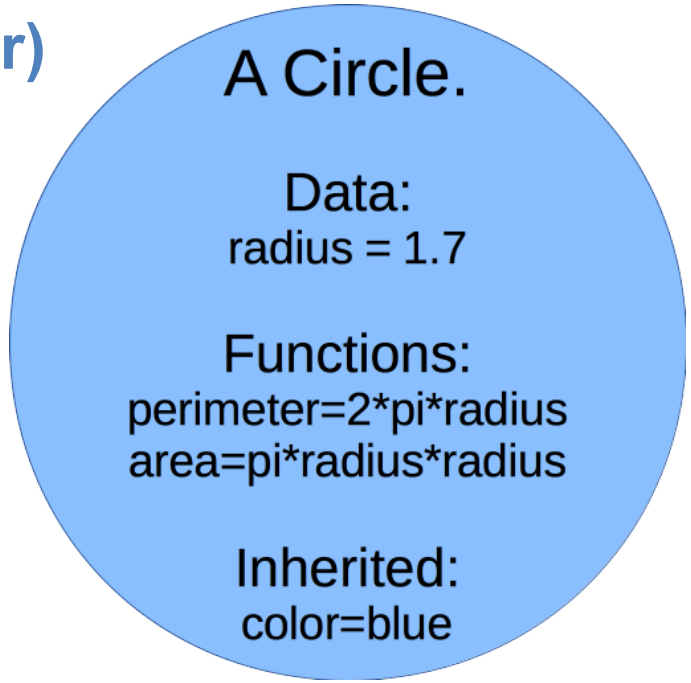
Object-oriented Python

```
class Shape:
   def __init__(self,color):
      self.__color=color
   @property
   def color(self):
      return self.__color
   @property
   def perimeter(self):
      return NotImplemented
   @property
   def area(self):
      return NotImplemented
```



Object-oriented Python

```
class Circle(Shape):
    def __init__(self,color,radius):
        super(self,Circle).__init__(color)
        self.__radius=radius
    @property
    def perimeter(self):
        return math.pi*self.__radius*2
    @property
    def area(self):
        return math.pi*self.__radius**2
```



An example for UnifiedPost

```
class UnifiedPost:
  def __init__(self,infile,fixd,postexec,when):
    (self.infile,self.fixd.self.postexec,self.when)=\
        infile,   fixd,   postexec,   when
  def run_post(self):
    self.link_fix()
    self.make_itag()
    make_symlink(self.infile,"INFILE",
                 logger=self.log(),force=True)
    cmd=mpirun(mpi(self.postexec)<"itag")
    checkrun(cmd,all_ranks=true,logger=self.log())
  def link_fix(self):
    fixes=[f for f in glob.glob(fixd+"/*")]
    make_symlinks_in(fixes,".",logger=self.log())
```

An example for UnifiedPost

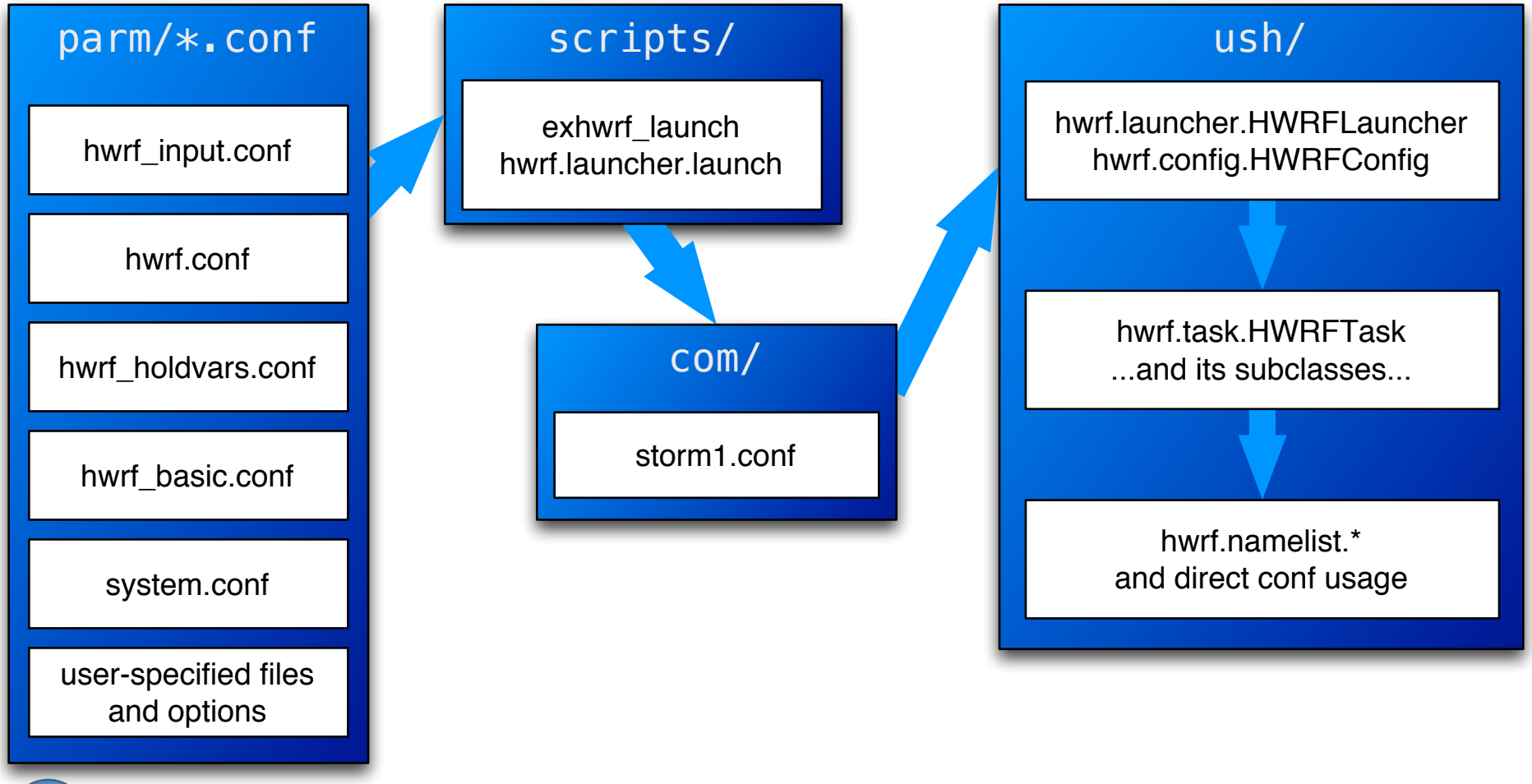
```
class HWRFPost(UnifiedPost):
  def make_itag (self):
    with open("itag","wt") as f:
      itagdata=self.when.strftime(
       "INFILE\nnetcdf\n%Y-%m-%d_%H:%M:%S" "\nNMM NEST\n")
      f.write(itagdata)

class NEMSPost(UnifiedPost):
  def make_itag (self):
    with open("itag","wt") as f:
      itagdata=self.when.strftime(
       "INFILE\nnetcdf\n%Y-%m-%d_%H:%M:%S""\nNEMS\n")
      f.write(itagdata)
```

Configuring HWRF

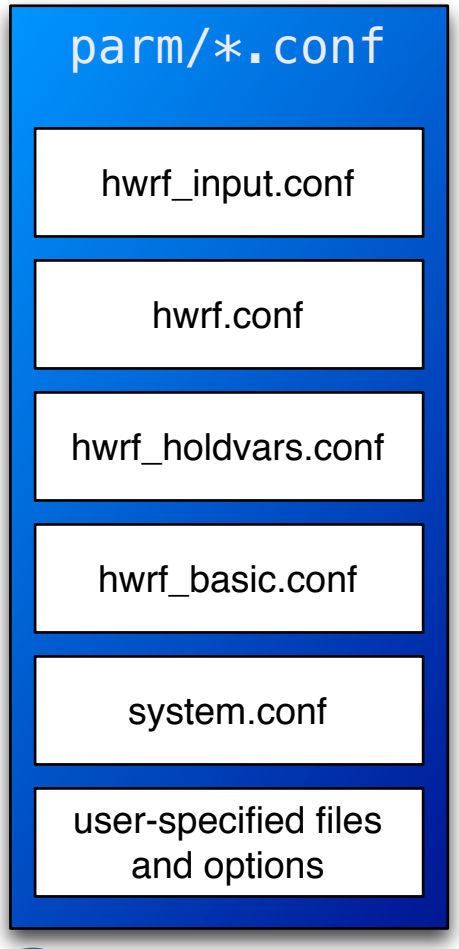
Conf files

hwrf_expt.py

Configuring HWRF Overview



Unix .conf Files



Simple format

```
# This is a comment
[section]
key=value ; This is also a comment
key2=value2
```

Doxygen format

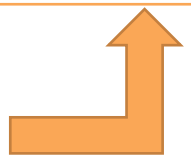
```
## Short description of section
#
# Long description of section
# @note Doxygen+markdown syntax
[section]
key=value ;; short description of key
## Short description of key2
#
# long description of key2
key2=value2
```

Unix Conf Files

- String substitution

```
[myprog]
basedir = /some/path
exename = myexe.x
exepath = {basedir}/exec/{exename}
```

```
exepath = /some/path/exec/myexe.x
```



- String substitution with formatting

```
[myprog]
gridnum = 5
exename = myexe_{gridnum:02d}.x
exepath = {basedir}/exec/{exename}
```

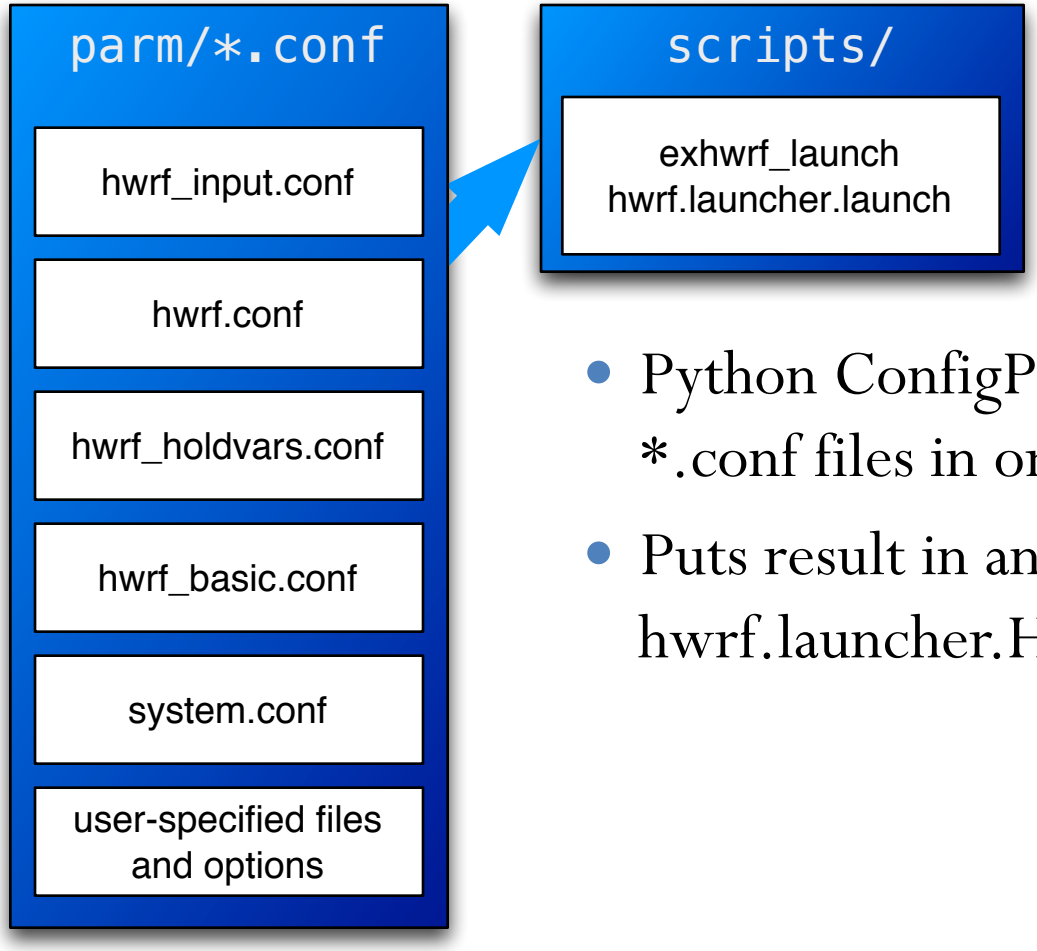
```
exename = myexe_05.x
```

- Substitute from other sections

```
[grid]
num     = 5
[myprog]
exename = myexe_{grid/num:02d}.x
exepath = {basedir}/exec/{exename}
```

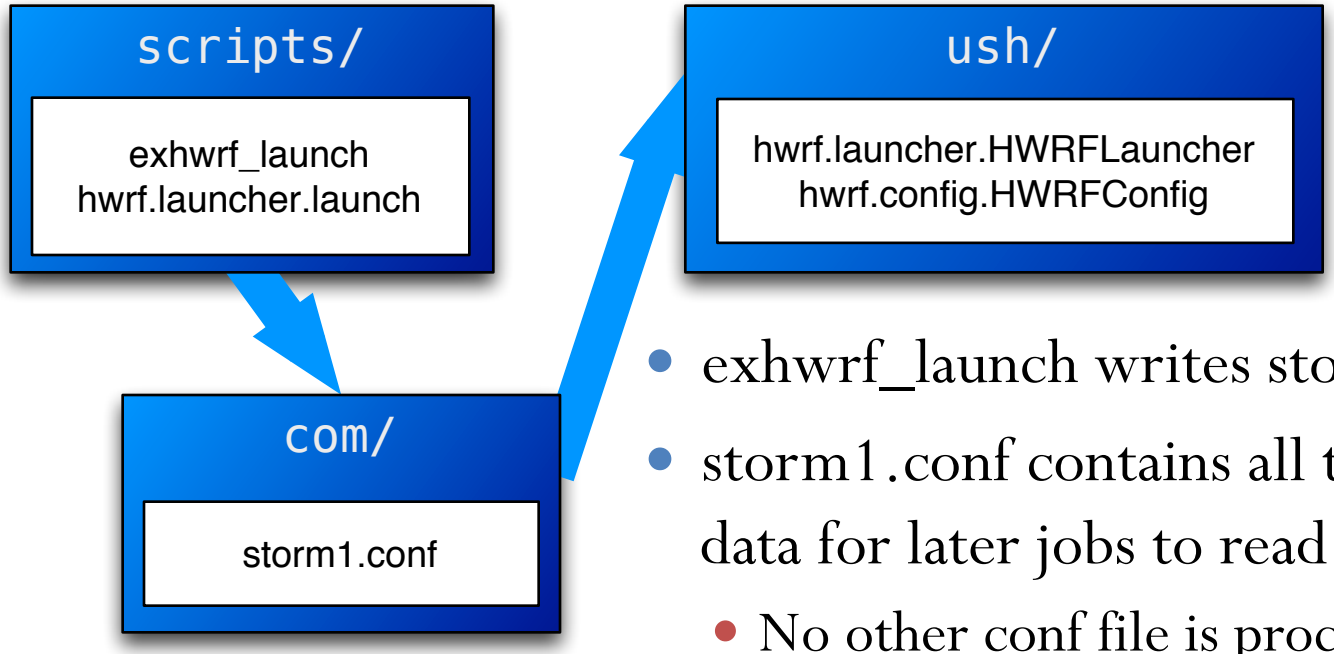
```
exename = myexe_05.x
```

Config Processing



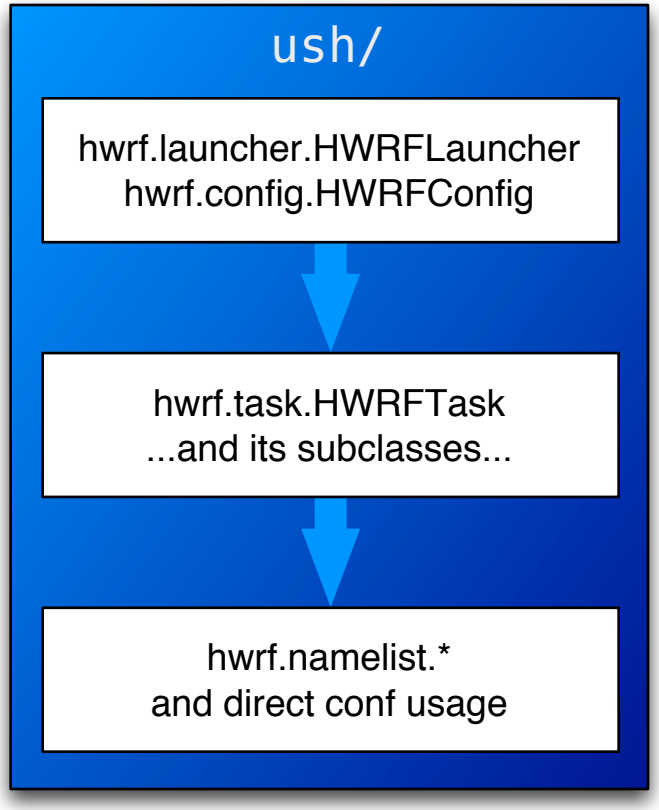
- Python ConfigParser.ConfigParser parses the *.conf files in order
- Puts result in an in-memory hwrf.launcher.HWRFLauncher object

storm1.conf



- exhwrf_launch writes storm1.conf
- storm1.conf contains all the processed config data for later jobs to read
  - No other conf file is processed
- Later jobs read storm1.conf using hwrf.launcher.load
- hwrf.launcher.HWRFLauncher contains many convenience functions for using the conf info

HWRF Python Tasks



- HWRFLauncher & HWRFConfig
  - Classes that access conf data
  - getstr(section, key, default)
    - Returns default value if none specified in storm1.conf
  - getint, getfloat, getbool, etc. (see docs for full list)
- HWRFTask is an instance of each of the tasks to be completed
  - Examples include GeogridTask, WRFAtmos, etc.
  - Has a database task name, a conf section, and an HWRFConfig
- hwrf.namelist.NamelistInserter reformats storm1.conf information into Fortran namelist files needed for various components

Data Communication

Database introduction

Passing around information

HWRF Database

- HWRF needs to know the status/availability of files millions of times per cycle
- When a file becomes available, a Python script puts its location, availability, and other metadata into an SQLite3 database

Table "products"

id	available	location	type
geogrid::geo_nmm_nest	0	/path/to/file	Product

Table "metadata"

id	key	value
geogrid::geo_nmm_nest	minsize	100000000

HWRF Database & produtil

- The produtil package contains all the HWRF utilities to write to and query the SQLite3 database
- produtil includes methods to check, deliver, and "undeliver" files
  - prod.check – Check for file of specified minimum size and age
    - Returns status as RUNNING, COMPLETED, FAILED
  - prod.undeliver – Remove file from working area
  - prod.deliver – Deliver file to specified location
- You can query the database on your own like any other SQLite3 database
- For a list of the input/output needed for HWRF, see hwrf.fcsttask.WRFTaskBase

Logging

stderr and stdout

- Located in the $HOMEhwrf/wrappers directory
- stdout files contain all the logging (info, error, critical level) messages from the Python scripts
- stderr files contain all the error and critical messages, plus the submission information for the job (PROLOGUE, EPILOGUE)
- Can be separated into *.out and *.err, or joined into one stream. Name and location depend on your job submission script.
- At least one set/file for each task.
- Multiple processor jobs have multiple sets of logs
  - post, products, tracker, etc.

Writing to the standard out

- Adding log messages can be done from the ush scripts with a few simple commands

```
logger=self.log()
logger.info('This is the value of some_variable:
                %s' %(some_variable))
logger.warning('This is a warning!')
logger.error('This is an error')
logger.critical('This is really bad!')
```

Result:

```
01/08 04:34:45.706 hwrf.gfsinit (relocate.py:353) INFO: This
is the value of some_variable: 270.0
01/08 04:34:45.902 hwrf.gfsinit (relocate.py:354) WARNING:
This is a warning!
```

Python Exception Stacks

- Several lines you get when you fail.

```
Traceback (most recent call last):
  File "/pan2/projects/dtc-hurr/dtc/HWRF_training//scripts/
exhwrf_gsi.py", line 60, in <module>
    main()
  File "/pan2/projects/dtc-hurr/dtc/HWRF_training//scripts/
exhwrf_gsi.py", line 53, in main
    hwrf_expt.gsi_d02.run()
  File "/pan2/projects/dtc-hurr/dtc/HWRF_training/ush/hwrf/gsi.py", line
982, in run
    self.grab_enkf_input()
  File "/pan2/projects/dtc-hurr/dtc/HWRF_training/ush/hwrf/gsi.py", line
285, in grab_enkf_input
    self.grab_gfs_enkf()
  File "/pan2/projects/dtc-hurr/dtc/HWRF_training/ush/hwrf/gsi.py", line
607, in grab_gfs_enkf
    %(there,))
GSIInputError: required input file is empty or non-existent: /pan2/
projects/dtc-hurr/dtc/HWRF_training/pytmp/HWRF_training/2015082000/17W/
hwrfdata/enkf.2015081918/sfg_2015081918_fhr06s_mem001
```

Output from components

- Many components have their own log files
- For example:
  - WRF: rsl.out.* and rsl.err.*
  - WPS: metgrid.log.*, geogrid.log.*, ungrib.log
  - GSI: stdout
  - Coupler: cpl.out

Questions?