HWRF v3.9a Tutorial
24 January 2018 College Park, MD

Python Scripts in HWRF

Evan Kalina
Developmental Testbed Center, Boulder, CO
NOAA ESRL Global Systems Division, Boulder CO
University of Colorado CIRES, Boulder CO

Many slides contributed by
Christina Holt and Sam Trahan

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

HURRICANE WRF USERS PAGE

- Home
- Terms of Use
- Overview
- User Support ▶
- Downloads ▶
- Documentation
- Idealized
- Tutorials & Workshops ▶
- Testing and Evaluation
- HWRF Developers Info
- Additional Links

WRF FOR HURRICANES DOCUMENTS AND PUBLICATIONS

Hurricane WRF Documents

HWRF Documents

- 2017 Documents (for HWRF v3.9a release)
  - **HWRF Users' Guide v3.9a** (PDF)
  - **HWRF Scientific Documentation - October 2017** (PDF)
  - **WRF-NMM V3 User's Guide** (PDF)
- 2016 Documents (for HWRF v3.8a release)
  - **HWRF Users' Guide v3.8a** (PDF)
  - **HWRF Scientific Documentation - November 2016** (PDF)
  - **WRF-NMM V3 [illegible]e** (PDF)

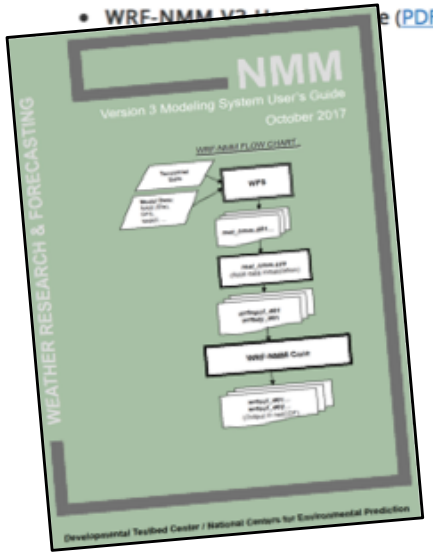
EVENTS

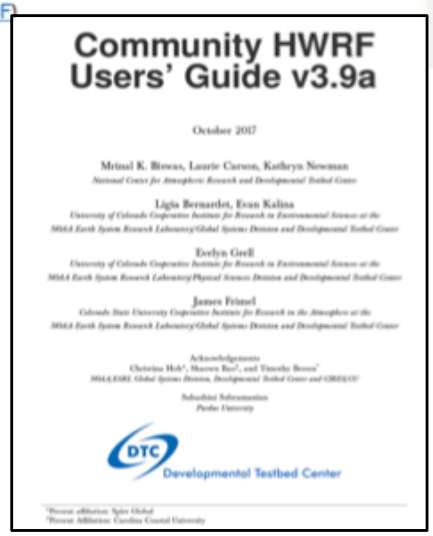
AMS 2018 NWP using Docker Containers
01.06.2018 to 01.06.2018
Location: AMS Annual Meeting in Austin, TX

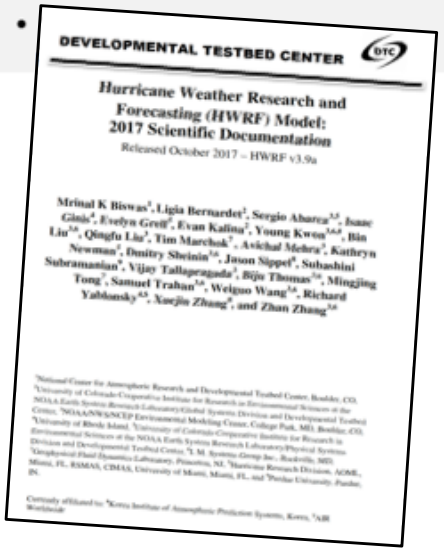
2018 Hurricane WRF Tutorial
01.23.2018 to 01.25.2018
Location: College Park, MD

ANNOUNCEMENTS

- 23-25 January 2018
  Hurricane WRF Tutorial
-

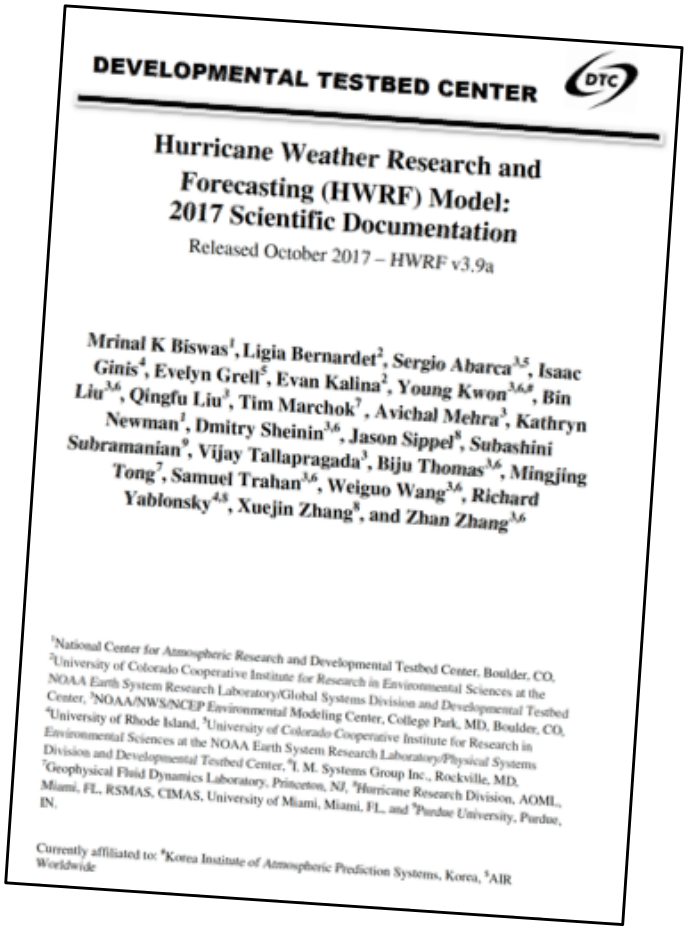






Scientific Documentation

- 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: 2017 Scientific Documentation

Released October 2017 – HWRF v3.9a

Mrinal K Biswas[1], Ligia Bernardet[2], Sergio Abarca[3,5], Isaac Ginis[4], Evelyn Grell[5], Evan Kalina[2], Young Kwon[3,6,#], Bin Liu[3,6], Qingfu Liu[3], Tim Marchok[7], Avichal Mehra[3], Kathryn Newman[1], Dmitry Sheinin[3,6], Jason Sippel[8], Subashini Subramanian[9], Vijay Tallapragada[3], Biju Thomas[3,6], Mingjing Tong[7], Samuel Trahan[3,6], Weiguo Wang[3,6], Richard Yablonsky[4,8], Xuejin Zhang[8], and Zhan Zhang[3,6]

[1]National Center for Atmospheric Research and Developmental Testbed Center, Boulder, CO, [2]University of Colorado Cooperative Institute for Research in Environmental Sciences at the NOAA Earth System Research Laboratory/Global Systems Division and Developmental Testbed Center, [3]NOAA/NWS/NCEP Environmental Modeling Center, College Park, MD, Boulder, CO, [4]University of Rhode Island, [5]University of Colorado Cooperative Institute for Research in Environmental Sciences at the NOAA Earth System Research Laboratory/Physical Systems Division and Developmental Testbed Center, [6]I. M. Systems Group Inc., Rockville, MD, [7]Geophysical Fluid Dynamics Laboratory, Princeton, NJ, [8]Hurricane Research Division, AOML, Miami, FL, RSMAS, CIMAS, University of Miami, Miami, FL, and [9]Purdue University, Purdue, IN.

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

https://dtcenter.org/HurrWRF/users/docs/scientific_documents/HWRFv3.9a_ScientificDoc.pdf

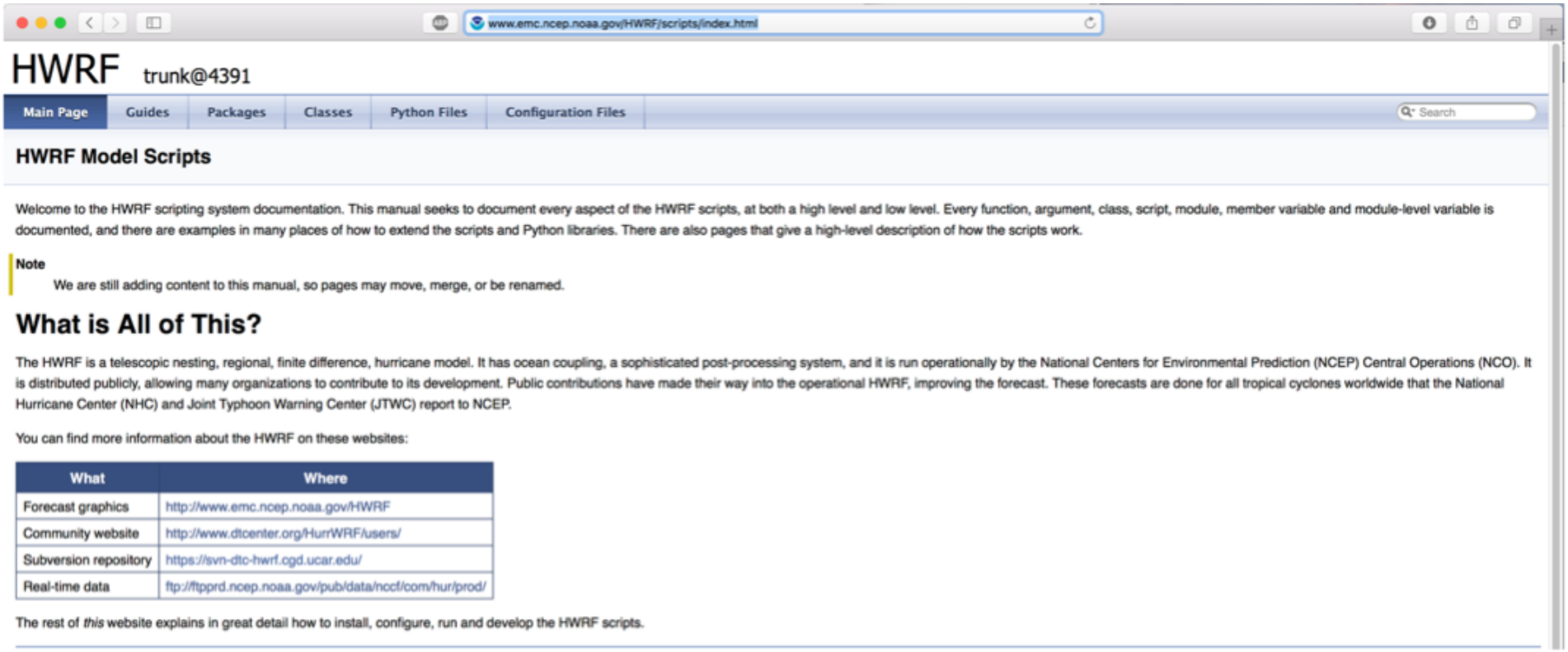
HWRF v3.9a User's Guide

- Includes detailed instructions on running each component
  - Specific to the public release
  - Explains how to run with provided wrapper scripts
- Content:
  - Introduction & software installation
  - Running HWRF
  - HWRF preprocessing system
  - Vortex Relocation
  - DA
  - Merge
  - MPIPOM-TC
  - Forecast Model
  - Post processor
  - Forecast products
  - Idealized

https://dtcenter.org/HurrWRF/users/docs/users_guide/HWRF_v3.9a_UG.pdf

Doxygen Website

- Detailed information about the HWRF configuration files and Python scripts



General Python help

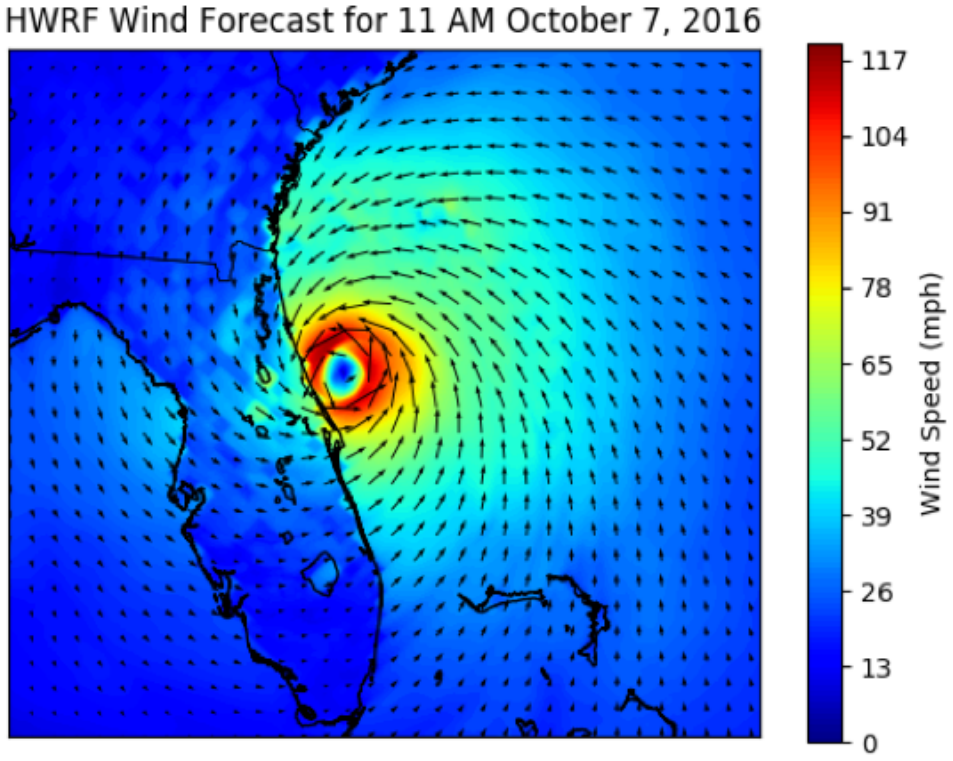
- Online (https://docs.python.org/release/2.6.6/)
- Open python in a terminal and use help() function for particular function. An example to get information with a Python list:

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

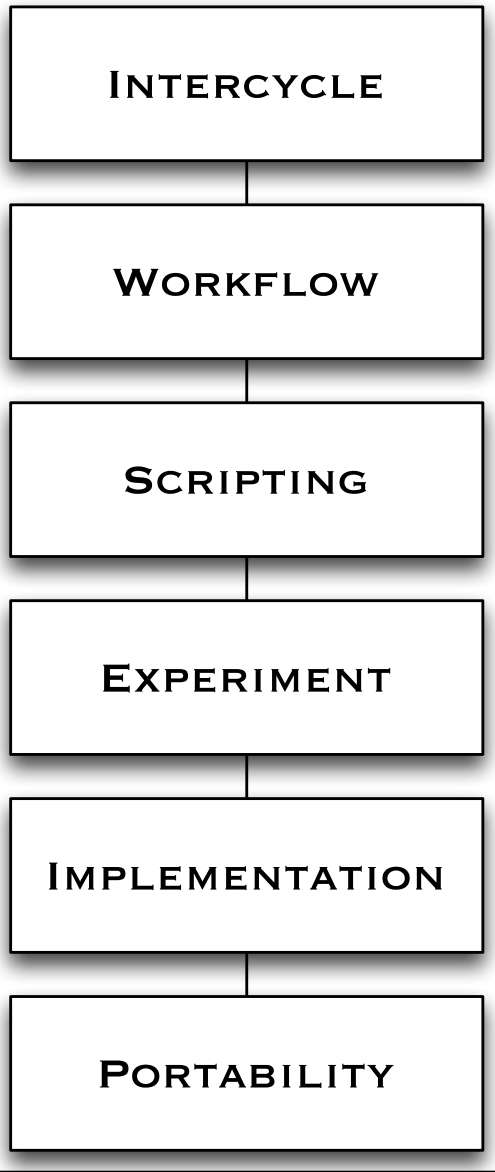
- The officially supported version for HWRF is Python v2.6.6.
  - Your version of Python should be version 2.6.6 or higher.
  - Version 3 is basically a different language.

HWRF System Overview

Overview of the system design

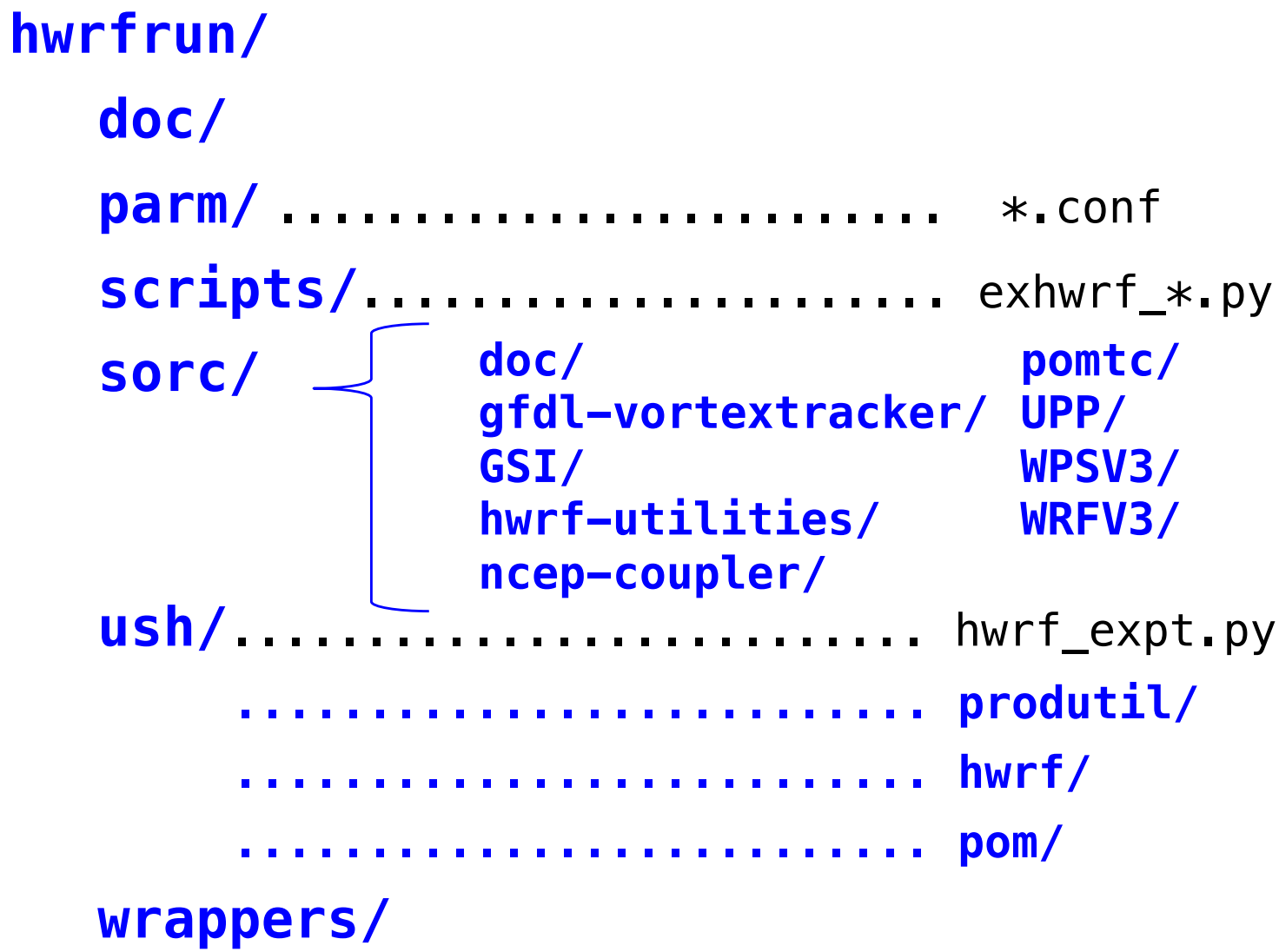


HWRF System: Overview

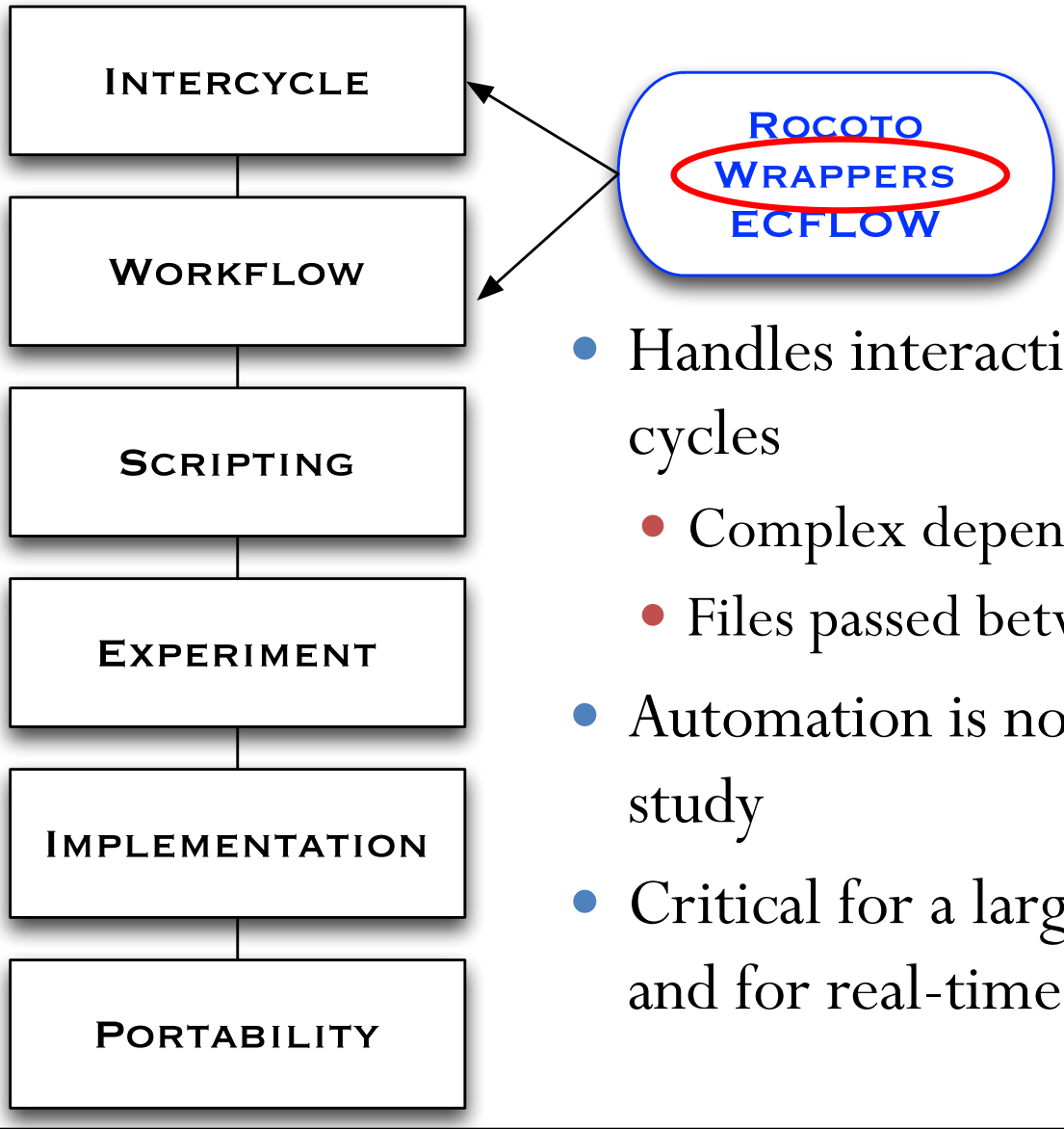


- 6 layers of scripts that are responsible for preparing the environment and data for and running the ~80 HWRF executables of the end-to-end system
- Most of these layers are written using an object-oriented (O-O) Python design
- O-O design makes the system highly configurable and reduces the footprint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

HWRF Directory Structure

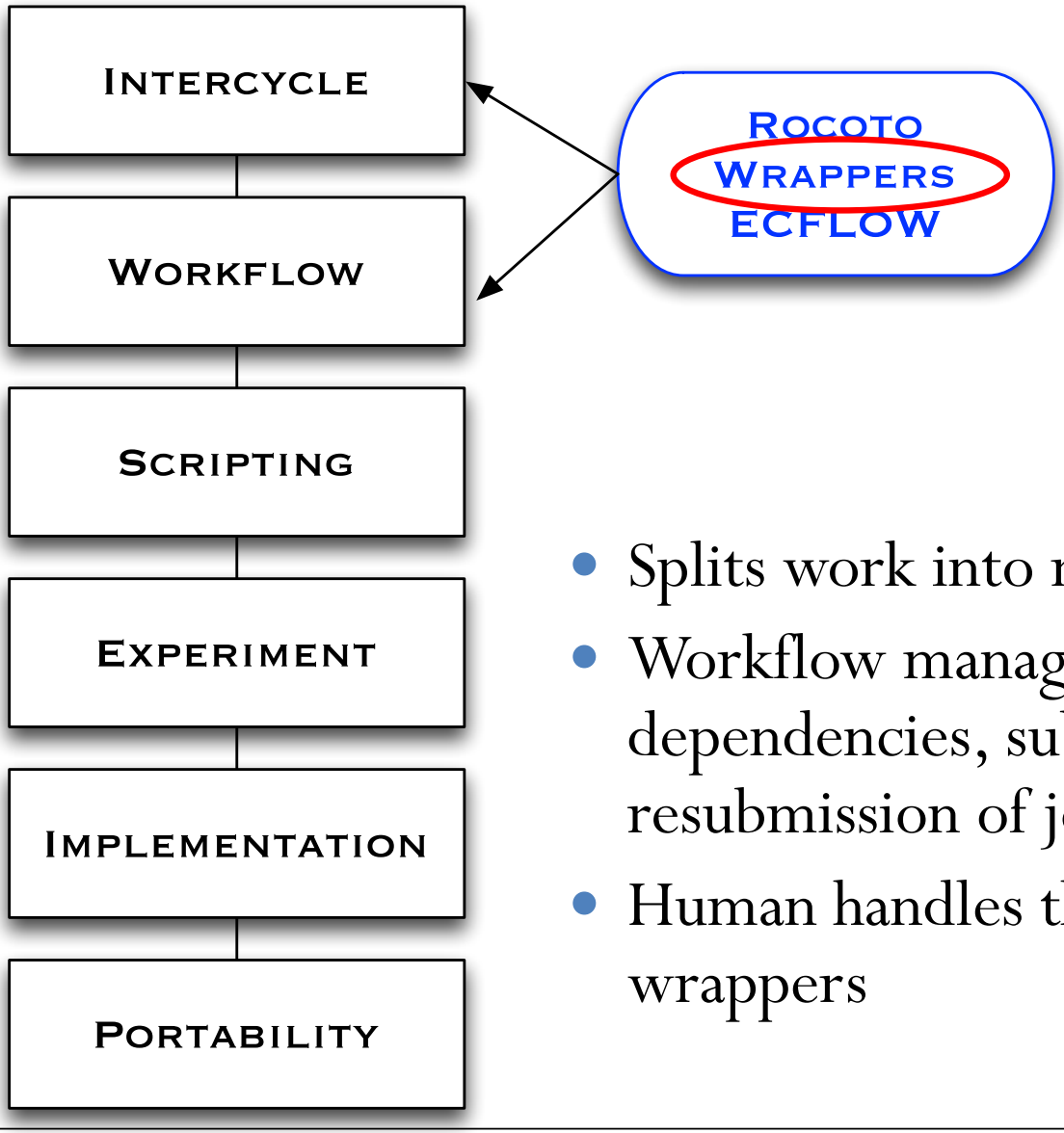


HWRF System: Intercycle Layer



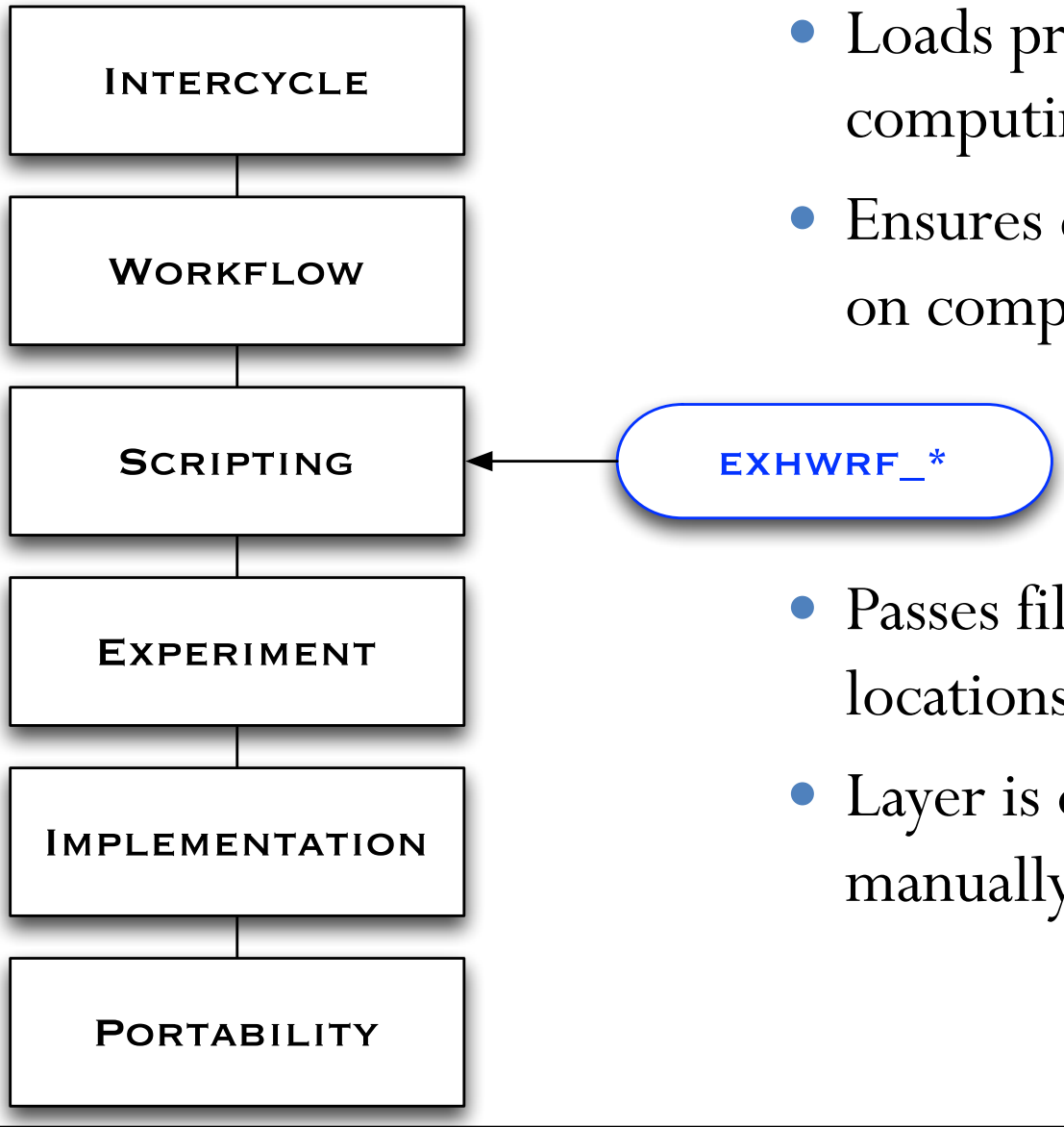
- Handles interactions between several cycles
  - Complex dependencies
  - Files passed between them
- Automation is not needed for a case study
- Critical for a large retrospective study, and for real-time automation

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

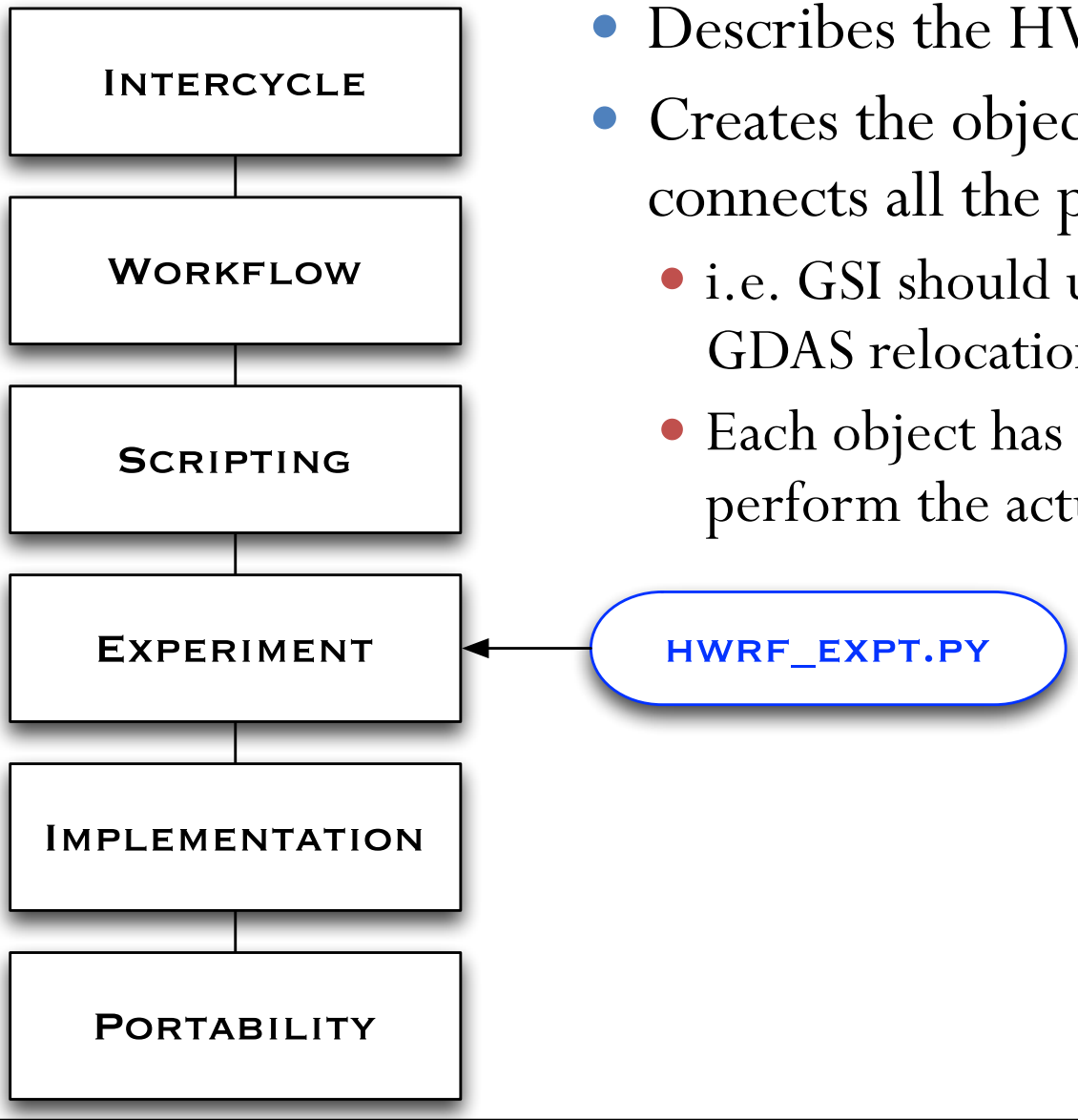
HWRF System: Scripting Layer



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

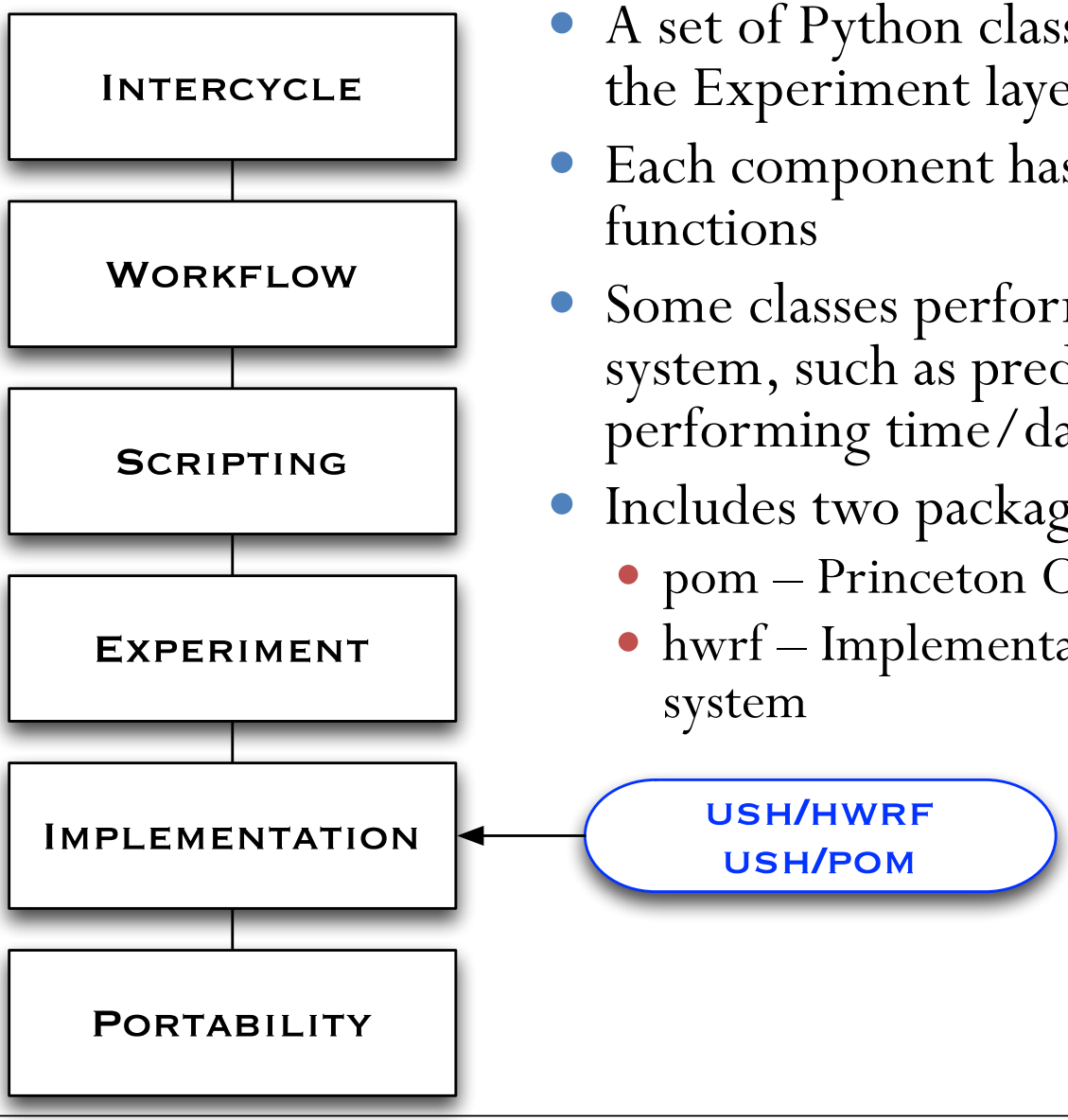
- Passes file and executable locations to the next lower layer
- Layer is optional – can be done manually by user

HWRF System: Experiment Layer



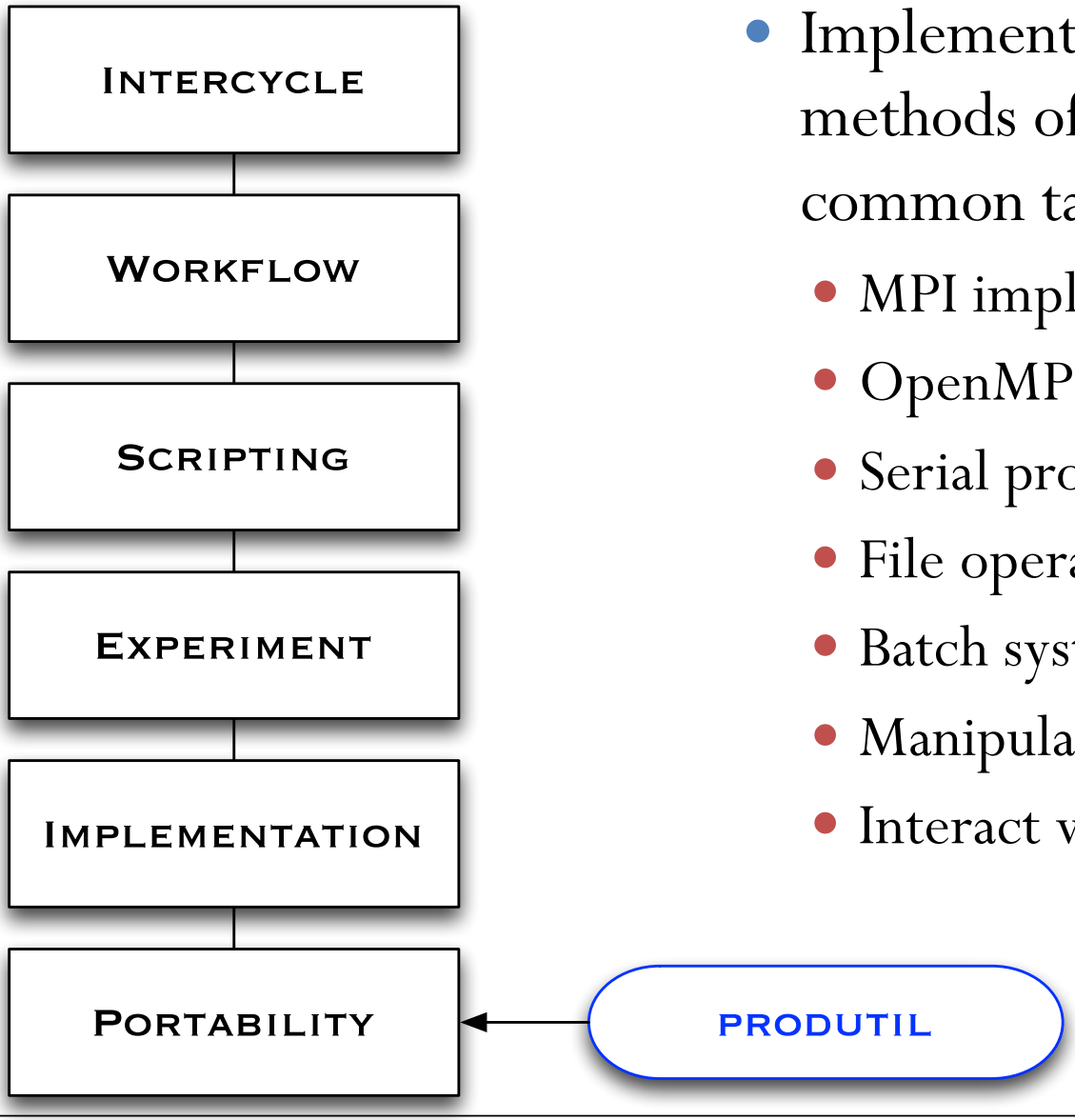
- Describes the HWRF workflow
- Creates the object structure that connects all the pieces
  - i.e. GSI should use input from the GDAS relocation output
  - Each object has a run() function to perform the actual task

HWRF System: Implementation Layer



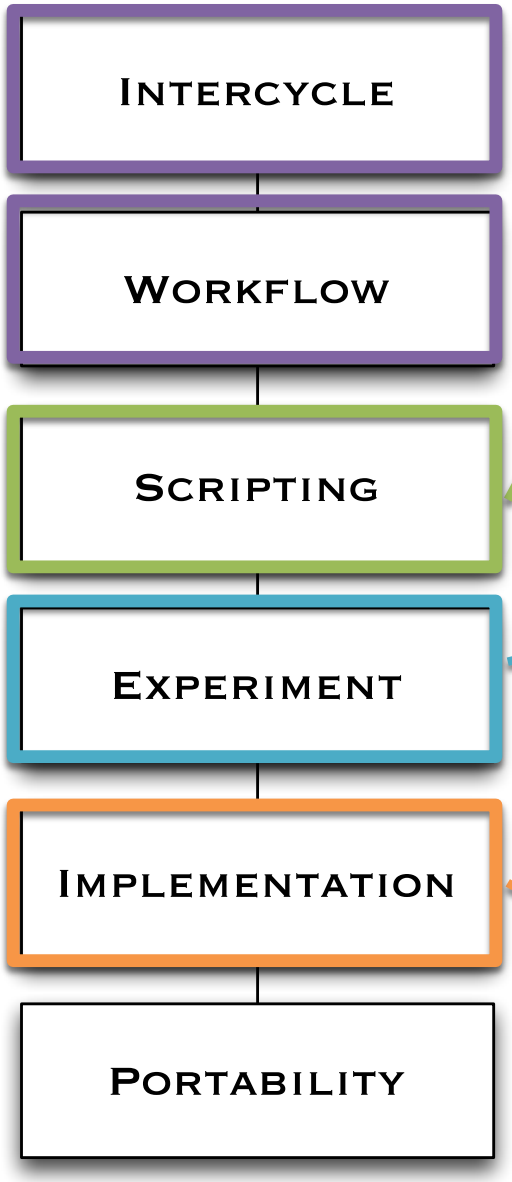
- A set of Python classes and functions used by the Experiment layer to run HWRF
- Each component has its own class and set of functions
- Some classes perform utilities to support the system, such as predicting filenames and performing time/date arithmetic
- Includes two packages
  - pom – Princeton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

A real-world example

Object-oriented programming: Some definitions

- A class is a blueprint (e.g., for building/riding a bike). It defines all of the characteristics (attributes – e.g., color) and behaviors (methods – e.g., coasting) of that object (e.g., a bike).

- An object is created from a class. From the blueprint provided by the class, you can build many different kinds of bikes (fast red bike, slow blue bike, etc.). Each object (bike) is a specific instance of the bike class.

- Helpful hints for distinguishing between objects, methods, and attributes:
  - Objects: think nouns: person, place, or thing
  - Methods: think action verbs: behaviors that can be performed
  - Attributes: think adjectives: characteristics that describe the object

Object-oriented Python

- Instantiation
  - sam = bicycle()

Object-oriented Python

- Attributes
  - sam.make = “raleigh”
  - sam.model = “revenio”
  - sam.wheel_diameter = 700 # in mm

Object-oriented Python

- Methods
  - sam.ride("College Park, MD", "Boulder, CO")
  - sam.tuneup(3000)

Putting it all together

```
class bicycle: # this is a class

  def __init__(self,make,model,wheel_diameter):
    # this is a special method called by Python to create the class
    self.make=make
    self.model=model
    self.wheel_diameter=wheel_diameter

  def ride(self,start,finish): # this is another method
    self.lats=[]
    self.lons=[]
    # some logic here to calculate the optimal path
    self.lats=[38.99, 39.10, 40.02]
    self.lons=[-76.94, -94.58, -105.27]
    return self

  def tuneup(self,num_km): # this is another method
    self.toDo=[]
    if (num_km >= 3000. and num_km < 5000.):
      self.toDo=['replace tires', 'replace chain', 'clean frame']
    # insert additional logic for other maintenance milestones
    return self
```

Your program that calls the bicycle class

```
# Instantiate the bicycle class
sam = bicycle('rayleigh','revenio',700.)
# sam is an object. A specific instance of the bicycle class.

# Call the ride method
sam = sam.ride('College Park, MD', 'Boulder, CO')
print(sam.lats, sam.lons)

# Call the tuneup method
sam = sam.tuneup(3000)
print(sam.toDo)
```

```
[Evan.Kalina@fe2 python]$ python
Python 2.7.13 |Anaconda 4.3.1 (64-bit)|

>>> execfile("bike.py")
([38.99, 39.1, 40.02], [-76.94, -94.58, -105.27])
['replace tires', 'replace chain', 'clean frame']
```

Object-oriented Programming

An example for HWRF

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

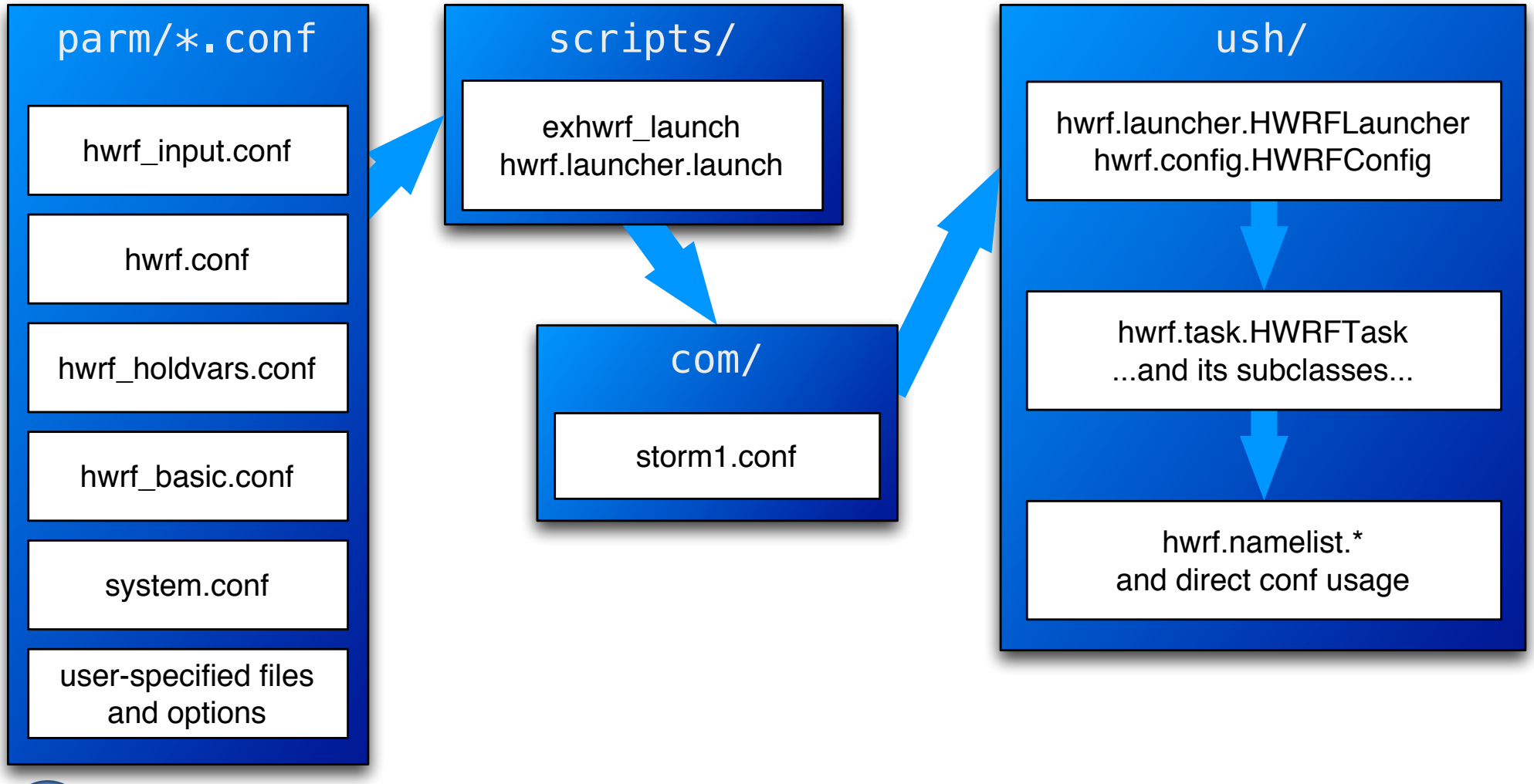
```
class HWRFPost(UnifiedPost):
  def make_itag (self):
    with open("itag","wt") as f:
      itagdata=self.when.strftime(
       "INFILE\nnetcdf\n%Y-%m-%d_%H:%M:%S" "\nNMM NEST\n")
      f.write(itagdata)

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

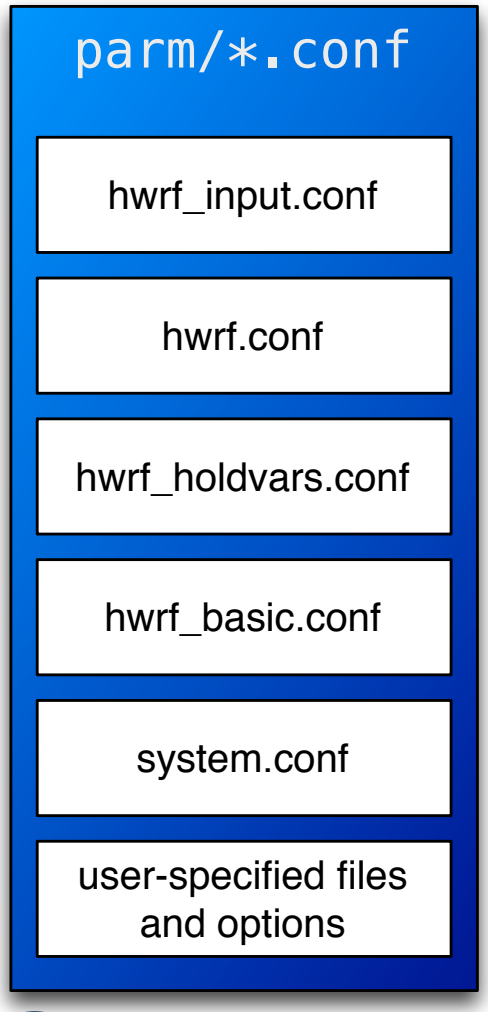
Conf files

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

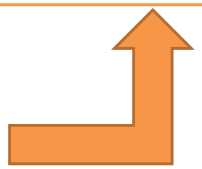
```
## Short description of section
#
# Long description of section
# @note Doxygen+markdown syntax
[section]
key=value ;; short description of key
## Short description of key2
#
# long description of key2
key2=value2
```

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}
```

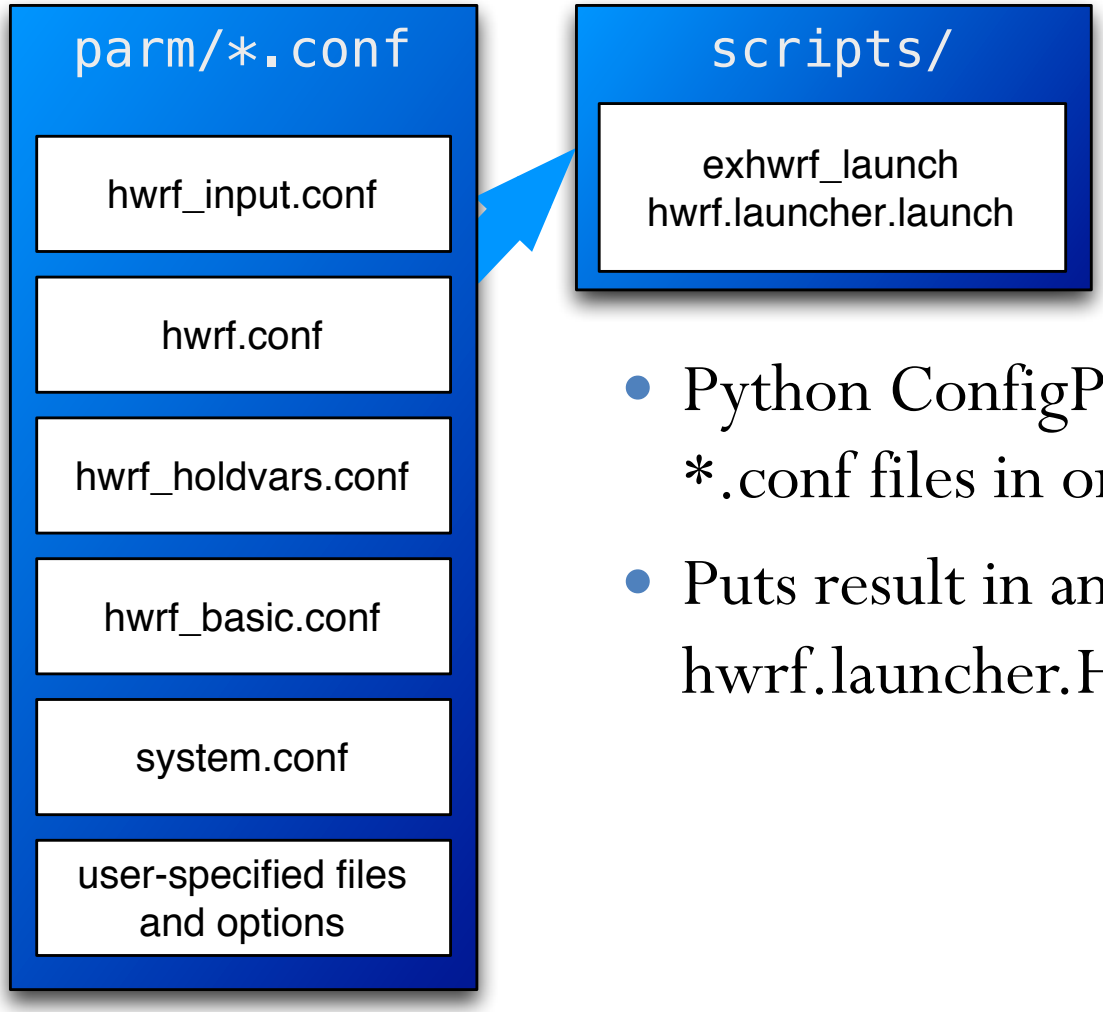
```
exename = myexe_05.x
```

- Substitute from other sections

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

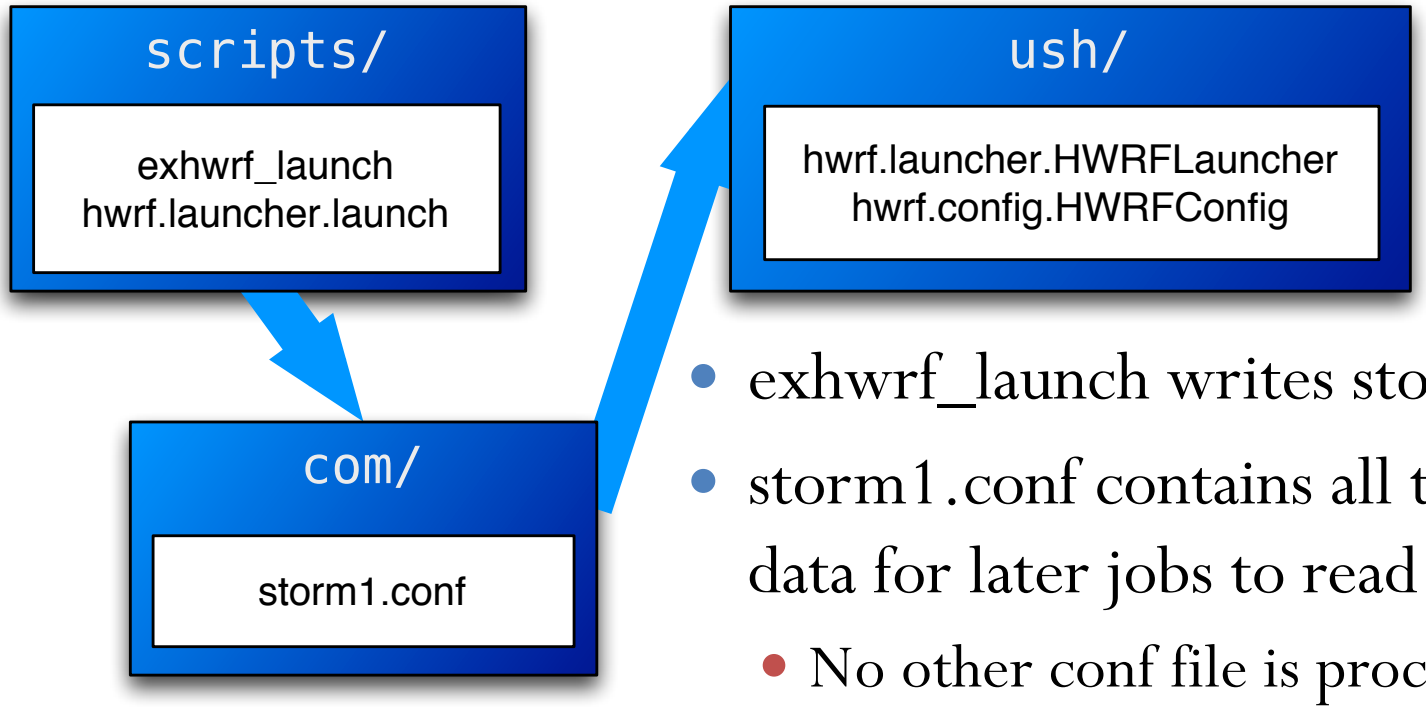
```
exename = myexe_05.x
```

Config Processing



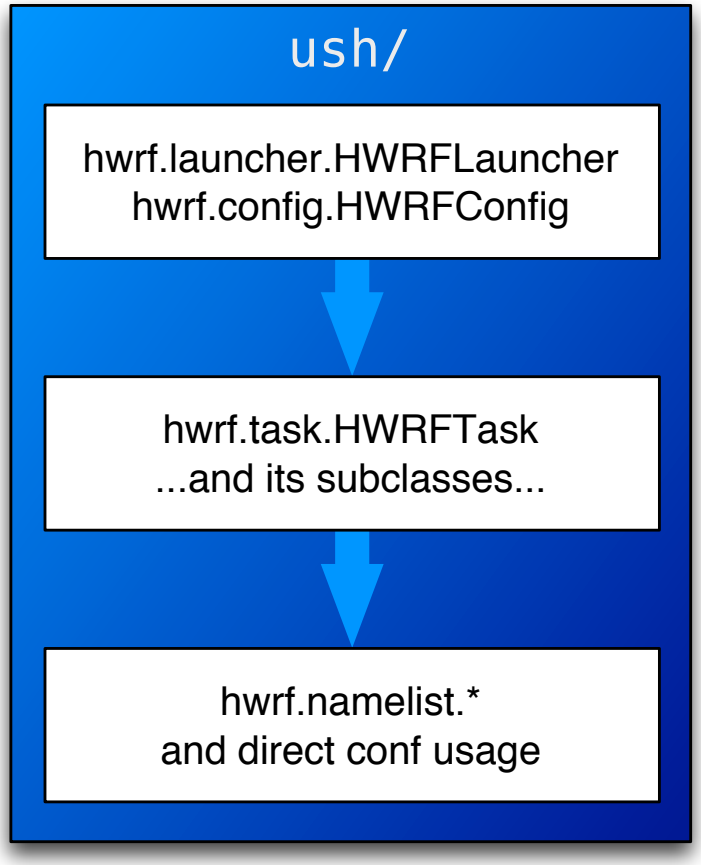
- Python ConfigParser.ConfigParser parses the *.conf files in order
- Puts result in an in-memory hwrf.launcher.HWRFLauncher object

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/en
kf.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?