HWRF Python Post-Processing

An Introduction for Production Users

Sam Trahan

April 22, 2014

Contents

- Overview
- Script Structure
- Package `produtil`
- Delivery
- Post-Processing Structure
- Troubleshooting
- Conclusions

Overview
Goals and Realities

- Simpler Scripts
  - 50k lines down to 30k lines despite partial rewrite
- Use less resources
  - post-processing 4 nodes down to 1.5 nodes
- Have one file that expresses full workflow
  - ush/hwrf_expt.py
  - Only python parts of workflow, and ksh bits that python monitors (WRF)
- No paths in scripts – all paths in one parm file
  - parm/hwrf.conf

Overview
Python Advantages

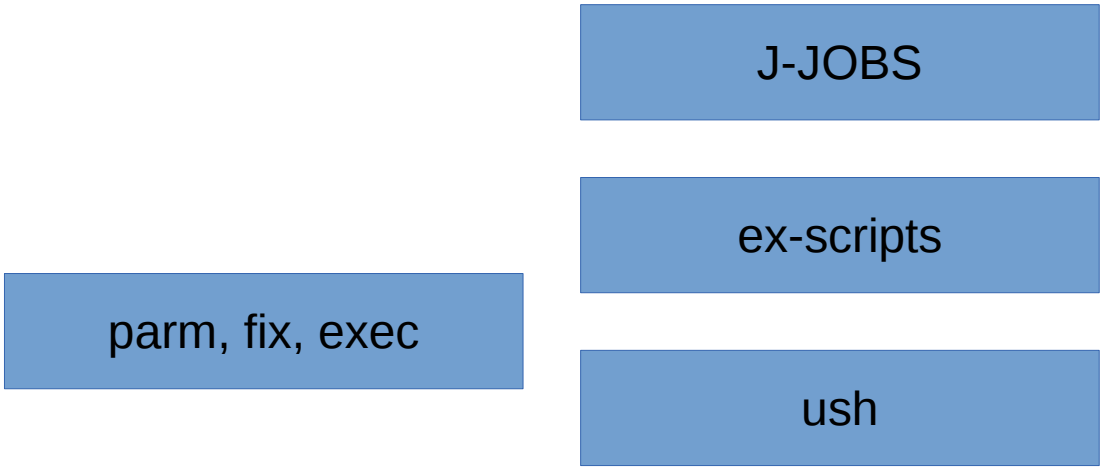
- Python over ksh:
  - better logging
  - direct access to system calls
    - don’t need cp, ln, mv, stat, …
  - better text processing and numerics
    - don’t need grep, cat, sort, awk, …
  - object-oriented
    - avoid duplicating code through inheritance
  - Result:
    - less code, fewer temporary files, no external processes

Overview
Python Problems

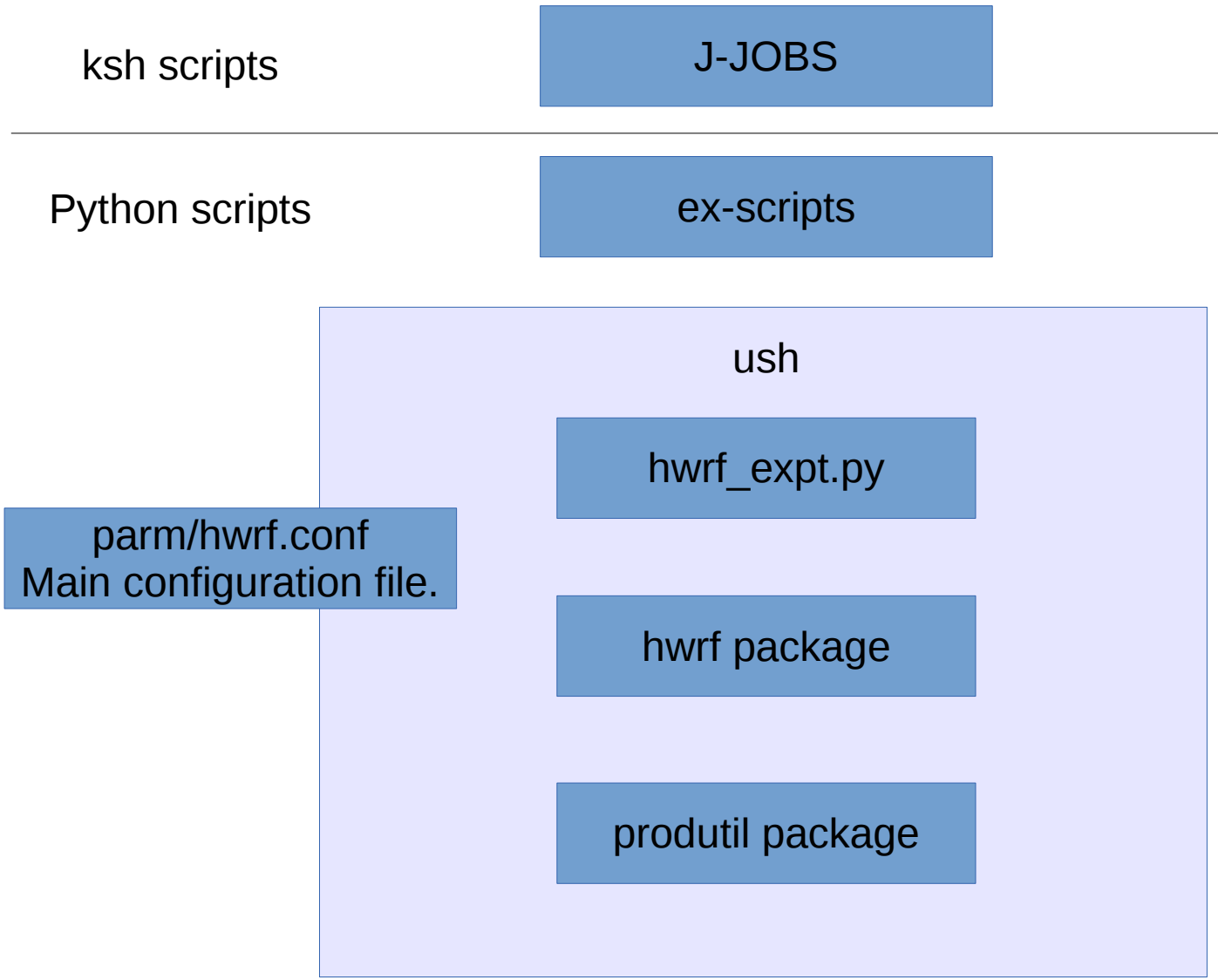
- WCOSS Python = version 2.6.6
  - old version - must work around bugs
  - no external libs: lacks critical functionality
- Others:
  - Python bad at backward compatibility
  - Flaw in RedHat libsqlite3 build scripts
- We can work around these problems

Script Structure

Three Layer NCEP Structure

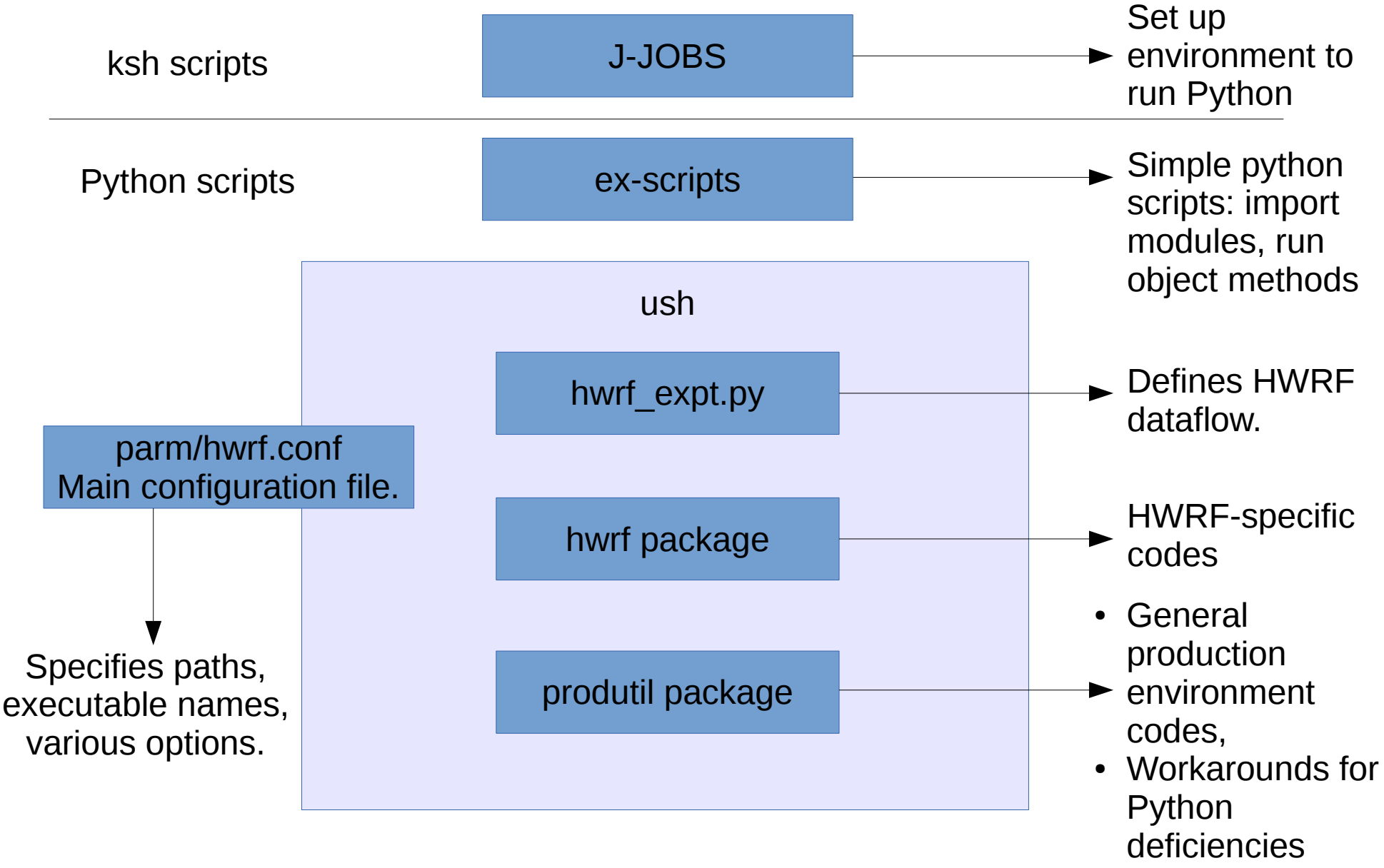


Script Structure
Python J-JOBS: Layered USH



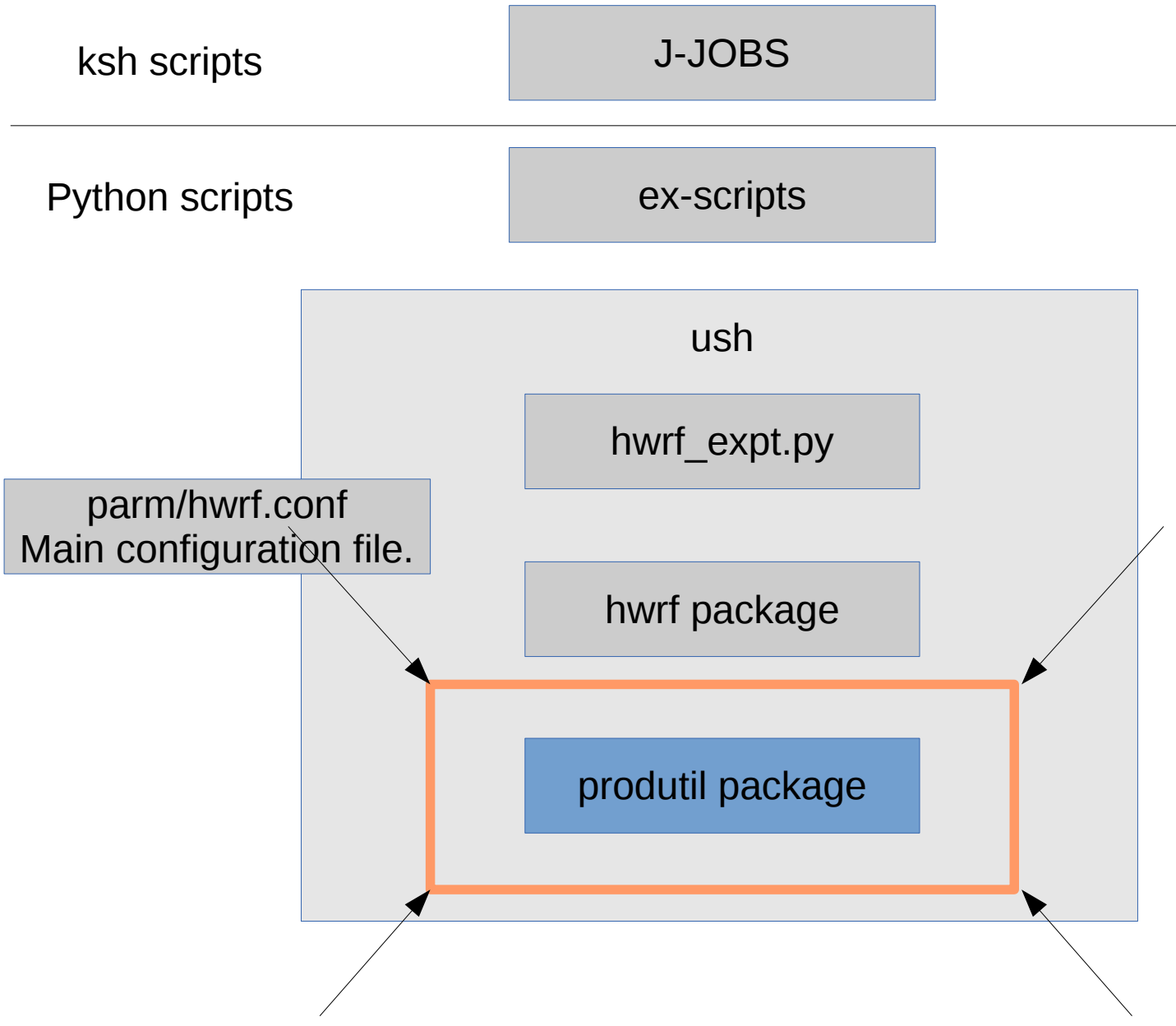
Script Structure

Python J-JOBS: Layered USH



Package `produtil`

Package “produtil”

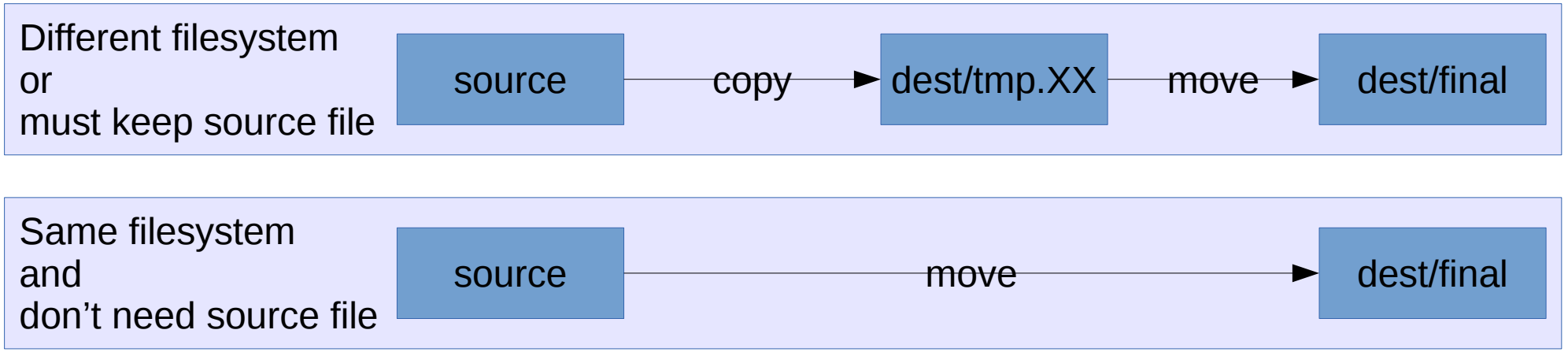


Package “produtil”
Overview

- Abstraction layer written around Python
  - Common shell tasks, with no external processes.
  - Implement NCEP-specific tasks:
    - dbn_alert wrapper
    - jlogfile
  - Better logging
  - Better error handling
  - Workarounds for deficiencies in Python
- Completely general
  - Could be adopted by other models.

Package “produtil”
Common I/O Operations: produtil.fileop

- `produtil.fileop` – file operations
  - wrappers for many common file operations
  - abstraction allows protection against Python upgrades
  - detailed logging
- `produtil.fileop.deliver_file`



Package “produtil”
Modules Related to fileop

- `produtil.retry` – for retrying operations
- `produtil.locking` – for file locking
- `produtil.tempdir` – for creating temporary directories, or cding to directories:

```
with produtil.tempdir.TempDir(keep=False) as t:
    # do things in temporary directory
#now cded out, and temp directory is gone
# (unless something failed)
# also have NamedDir for non-temporary dirs
```

Package “produtil”
Logging: produtil.log

- `produtil.log` – logging
  - wrapper around Python core `logging` package
  - custom output formats and destinations
  - jlogfile:
    - exactly like postmsg, but no external processes, no extra I/O
    - omit strack traces

jlogfile: ERROR and CRITICAL
(or anything sent to `produtil.log.jlogger`)

```
04/21 06:27:12Z JHWRF_POST-hwrf.post.2010091415: CRITICAL: post completed
```

stderr: WARNING, ERROR and CRITICAL

```
04/21 06:27:12.157 hwrf.post.2010091415 (post.py:542) CRITICAL: post completed
```

stdout: INFO, WARNING, ERROR and CRITICAL

```
04/21 06:27:12.157 hwrf.post.2010091415 (post.py:542) CRITICAL: post completed
```

Package “produtil”
Signal Safety: produtil.sigsafety

- By default, Python does not properly handle signals other than SIGINT.
  - SIGTERM, SIGQUIT, SIGHUP, etc.? Immediate exit.
  - No chance to clean up.
  - Causes problems
- `produtil.sigsafety`
  - installs signal handlers (raise `FatalSignal`)
    - cannot catch that exception through except(Exception)
  - ensures quick exit while allowing cleanup
  - provides stack trace to point code was at during signal
    - helps find deadlocks, infinite loops, infinite waits for data, etc.
  - tells `produtil.locking` to ban any new file locking

Package “produtil”
Package Initialization: produtil.setup

- Initializes produtil modules that require initialization
  - `produtil.log`, `produtil.sigsafety`, `produtil.dbnalert`
- Installs handlers, sets up default logging
  - Defaults mimic operational setup
- Configurable

```
import produtil.setup
produtil.setup.setup()
```

Package “produtil”
Command Execution: produtil.run

- `produtil.run` – run programs, shell-like syntax
  - `status=run( (exe(‘myprog’) < ‘my.stdin’)`
    `>= ‘my.log’)`
  - `checkrun(mpirun(mpi(‘hwrf_wm3c’)`
    `+ mpi(‘hwrf_ocean’)*9`
    `+ mpi(‘hwrf_wrf’)*200))`
  - `grbindex = runstr(exe(‘wgrib’)[‘-s’,’my.grib.file’])`
- Same syntax on all NOAA machines
  - implemented in `produtil.prog`, `produtil.mpiprog`, `produtil.mpi_impl` subpackage

Package “produtil”
NCEP-specific bits

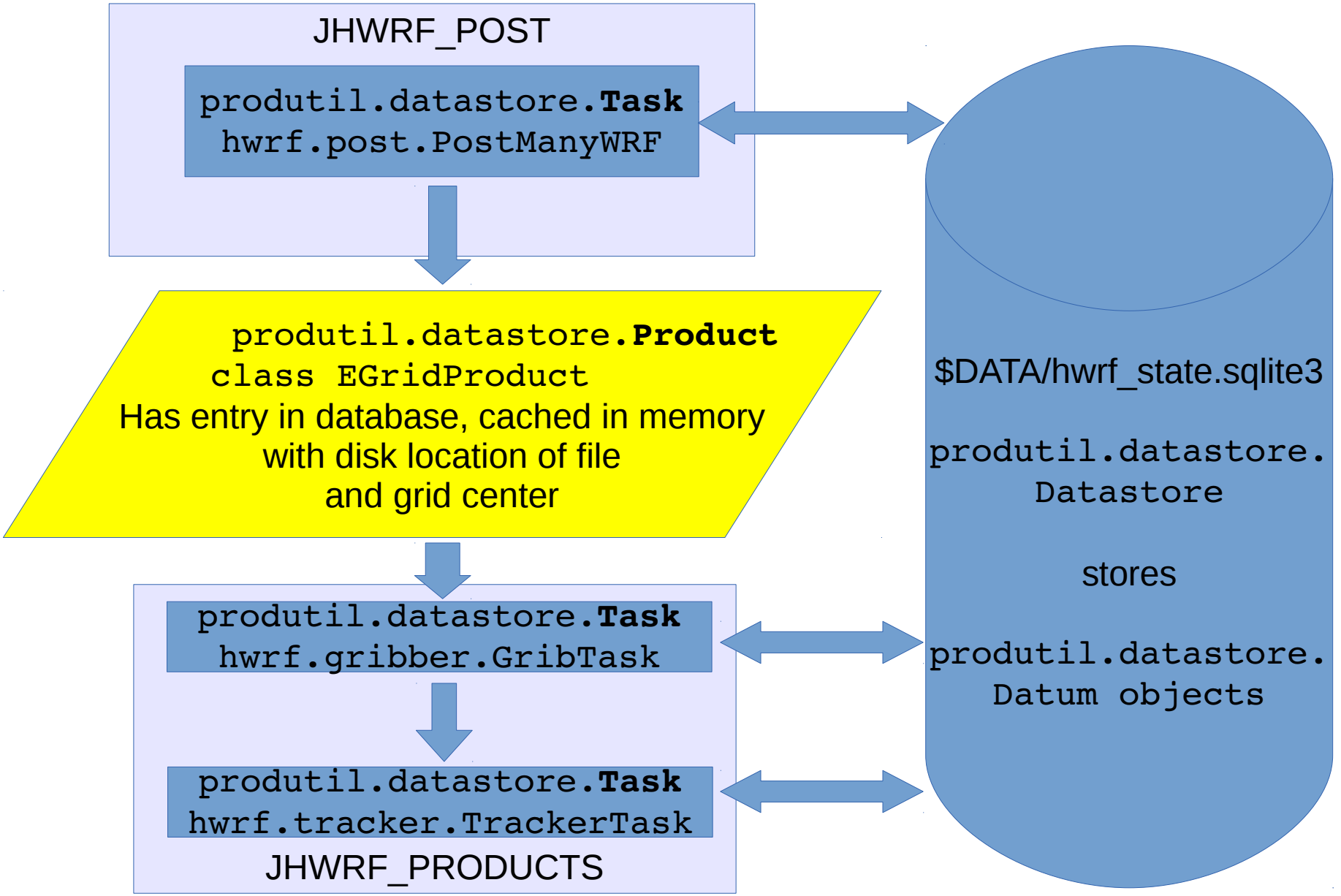
- `produtil.dbnalert` – dbn_alert wrapper
  - If $SENDDBN=NO, reports what would be alerted
  - (same with $PARAFLAG=YES)
- `produtil.log` – has jlogfile hooks with special output syntax
  - `jlogger` - object that logs to jlogfile for logging level of “INFO” or higher

```
jlogger.info('delivered track to COM')
```

```
04/21 05:55:12Z JHWRF_INIT-jlog: INFO: delivered track to COM
```

Package “produtil”

Inter-Job Communication: produtil.datastore



Package “produtil”
Inter-Job Communication: produtil.datastore

- `produtil.datastore` – dataflow between jobs
- `Datastore` class – wrapper around sqlite3 database for communication
- `Datum` class – anything that can be stored in a `Datastore`
  - `Product` class – represents a deliverable or upstream product
    - know availability, location, optional metadata
  - `Task` class – represents something that makes products.
    - has its own logger
    - has its own section in parm/hwrf.conf file
    - has complete/incomplete/failed info
    - optional metadata

Package “produtil”
Inter-Job Communication: produtil.datastore

- Critical Product methods and properties
  - Product.deliver – delivers a file
  - **Product.add_callback – adds a callable object (like a produtil.dbnalert.DBNAlert) to be called after delivery**
  - **Product.call_callbacks – calls all callbacks** DBN alerts!!
  - Product.available – True iff delivered
  - Product.location – where is it? (if available==True.)
  - Product.undeliver – undoes a delivery if possible
  - Product.check – update the “available” status
- Subclasses:
  - **UpstreamFile – an upstream file to be used by this workflow**
    - call UpstreamFile.check() to check availability
  - **FileProduct – a file produced by this workflow** Inter-job dataflow!!

Delivery

Delivery
COM and INTERCOM

- COM
  - Only for final results.
  - Track, GRIB2 files, HTCF, swath, etc.
- INTERCOM
  - Tasks run in temporary directories which are immediately deleted (not in intercom).
  - Want to send a file to another job?
  - Copy to intercom.
  - GRIB1 files are here

Delivery
Steps

- Make file in local temp dir.
- product.deliver(frominfo=”/local/path”)
  - deliver_file(from,to,...)
  - calls callbacks
    - DBNAlert
    - nhc track delivery

Delivery
DBN Alerts

- Product.deliver calls callbacks
  - DBNAlert objects: wrapper around dbn_alert
  - ex-scripts add callbacks to them before passing control to ush/hwrf/*py level
- ush/hwrf_alerts.py
  - Contains all DBN alerts in entire workflow.
- Example DBN Alert:

```
def add_wave_alerts():
    from hwrf_expt import nhcp
    # Loop over all wrfdiag files:
    for prod in nhcp.wrfdiag_products():
        # Add a dbn_alert to send as "HWRF_WAVE_INPUT"
        prod.add_callback(DBNAlert(['MODEL',
            'HWRF_WAVE_INPUT','{job}','{location}']))
```

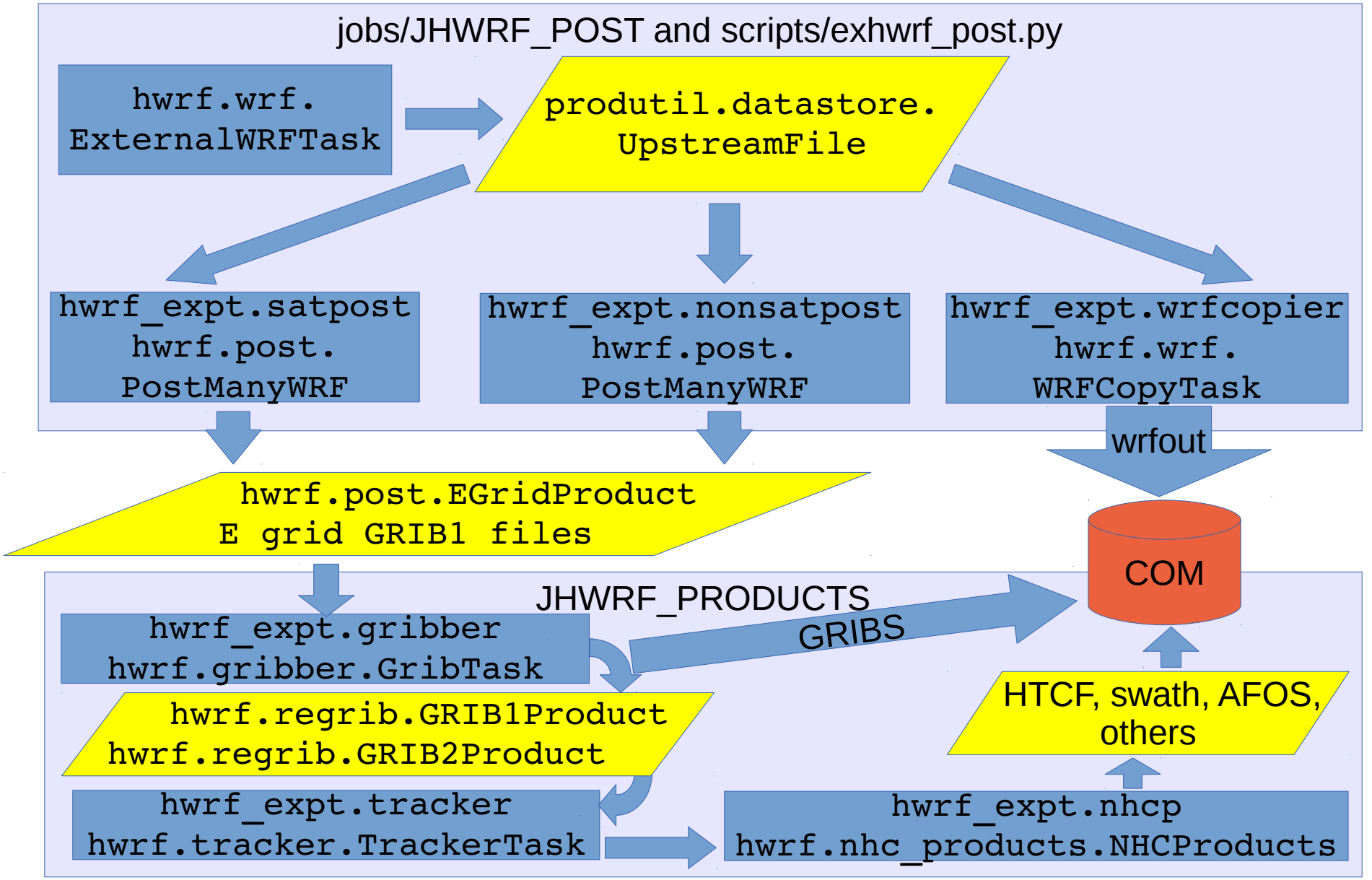
- {job} – replaced with job name
  - Default is $job – see produtil.dbnalert.init_job_string
- {location} – replaced with product disk location (Product.location)

Delivery
NHC A Decks

- Also in hwrf_alerts.py
  - hwrf_alerts.send_nhc_track
  - **Must set $ATCFdir in environment**
- Mimics old code
  - no storm sorting
    - not needed since HWRF runs one storm at a time

Post-Processing Structure

Post-Processing Structure
Workflow



Post-Processing Structure
Regribbing

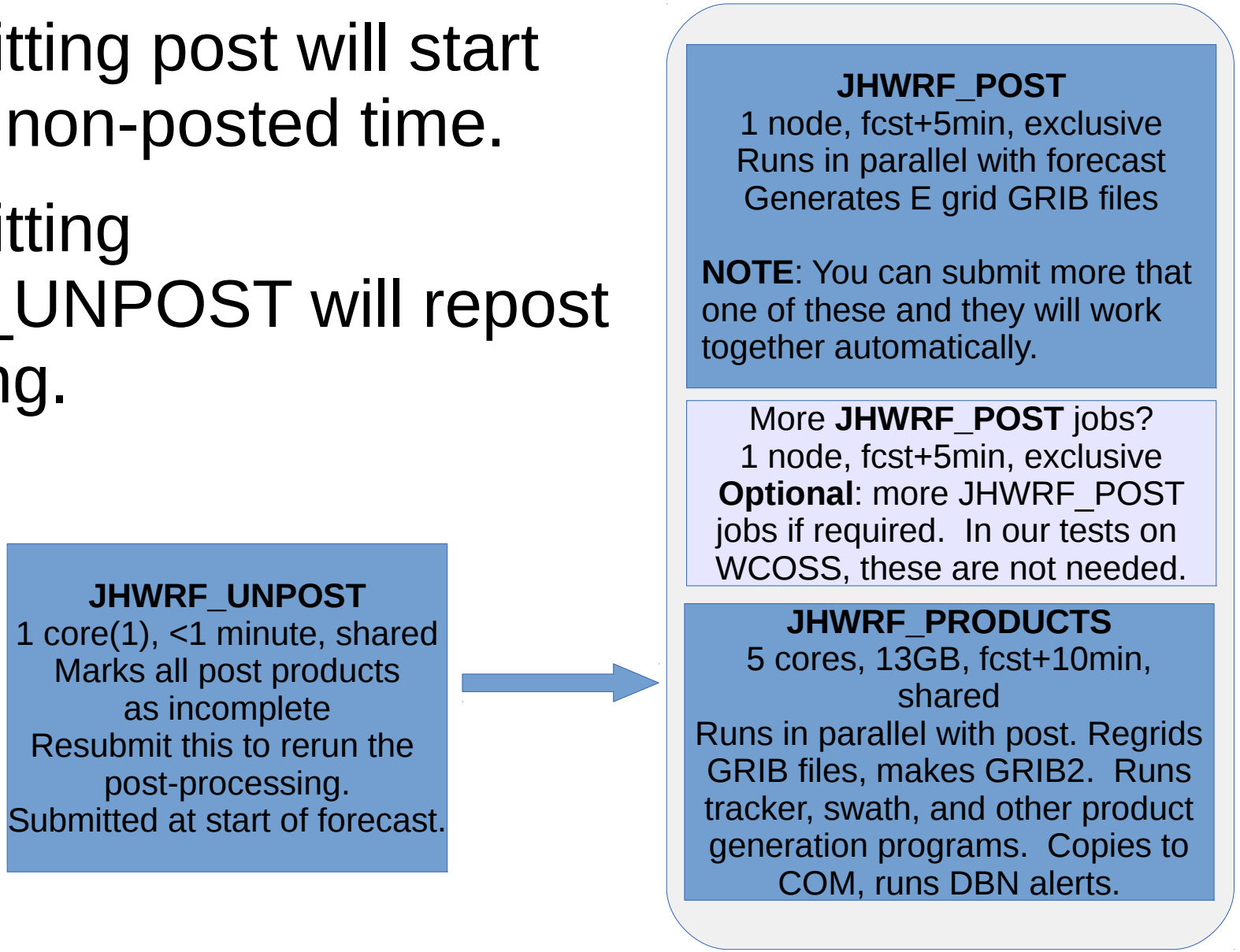
- Abstract, human-readable syntax for expressing regribbing
  - Merge domain 2 and domain 3 into combined grid and convert to GRIB2
  - Grab tracker subset (“trksub”), run vint and tave programs

    `vinttave((egrib2*d23 + egrib3*d23)/trksub*GRIB2)`
- Full example from ush/hwrf_expt.py:

```
grid3=igrb1(nonsatpost,domain=storm1inner)
grid2=igrb1(nonsatpost,domain=storm1outer)
grid1=igrb1(nonsatpost,domain=moad)
r=RegribMany(copygb=alias(exe(conf.getexe('copygb'))),
             wgrib=alias(exe(conf.getexe('wgrib'))))
r.add('d23',clatlon(grid2,res=[0.03,0.03],
      size=[12.,15.],scan=136,n=[501,401]))
r.add('hwrfprs_m',grid2*r.grid('d23')+grid3*r.grid('d23'))
r.to_intercom('{nidymdh}.hwrfprs_m.grbf{fahr:02d}','hwrfprs_m')
r.add(‘hwrf2prs_m’,r.GRIB(‘hwrfprs_m’)*GRIB2)
r.to_com(‘{nidymdh}.hwrfprs_m.grb2f{fahr:02d}’,’hwrf2prs_m’)
```

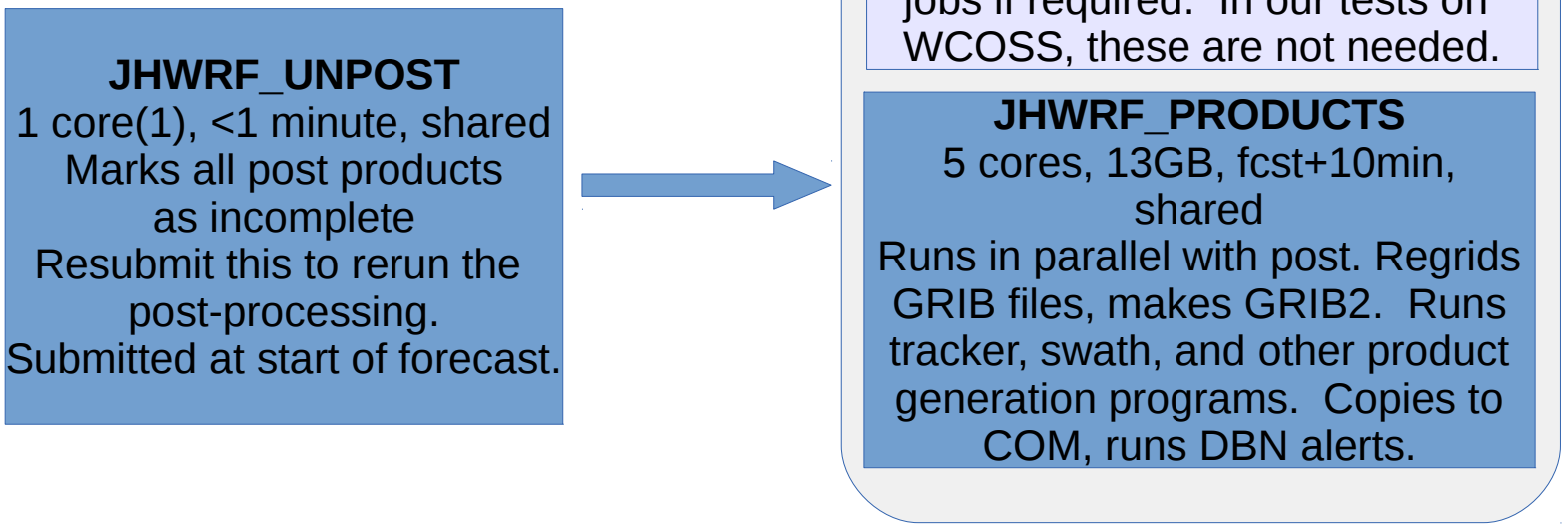
Troubleshooting
Post Resubmission

- Resubmitting post will start with first non-posted time.
- Resubmitting JHWRF_UNPOST will repost everything.



Troubleshooting
Post Resubmission

- Bug: if rerunning with coupling status changed, must delete:
  - $DATA/hwrf_state.sqlite3*
- Then resubmit JHWRF_UNPOST
- Fix is on its way.



Troubleshooting
Python Exceptions

- Example code:

```
def a(x):
    return x*3
def b(x,y):
    print y+a(x)
b(5,5)
```

- Result:

```
20
```

Troubleshooting
Python Exceptions

- Example code:

```
def a(x):
    return x*3
def b(x,y):
    print y+a(x)
b(“5”,5)
```

- Result:

```
Traceback (most recent call last):
  File "test.py", line 5, in <module>
    b("5",5)
  File "test.py", line 4, in b
    print y+a(x)
TypeError: unsupported operand type(s)
for +: 'int' and 'str'
```

- What happened?
  - “5”*3 = “555”
  - 5 is not “5”

Troubleshooting
Python Exceptions

- Example code:

```
def a(x):
    return x*3
def b(x,y):
    print y+a(x)
b(5,"5")
```

- Result:

```
Traceback (most recent call last):
  File "test.py", line 5, in <module>
    b(5,"5")
  File "test.py", line 4, in b
    print y+a(x)
TypeError: cannot concatenate 'str'
and 'int' objects
```

- Last line is where the error was detected.
- Passing bad inputs causes detection deeper in stack.

Conclusions

- Successful so far:
  - reduced from 4 nodes to 1.5 for post-processing
  - better logging
  - fewer temporary files
  - ~40% code reduction even with partial rewrite
  - configuration, delivery all in three files:
    - ush/hwrf_alerts.py, ush/hwrf_expt.py, parm/hwrf.conf
- `produtil` package fixes Python problems, takes advantage of Python strengths
  - Would have been easier with full installation of latest Python 2.7.x
  - Could remove some extraneous scripts and programs