CROW in HAFS

Sam Trahan
NOAA, GSL, CU, CIRES, DTC, etc.

(Last Updated April 21, 2020)

Executive Summary

Whole Project in One Slide

- HAFS is almost unchanged
  - Conf files now YAML with the same structure
  - Rocoto XML is entirely generated from YAML
  - Update to Python 3.6
  - Operational ecFlow suite is unchanged.
  - ConfigParser is replaced with CROW in operations
- Benefits over old system:
  - Direct connection between configuration files and workflow generation
  - Can embed calculations into configuration files
  - No longer using a retired Python version
- Disadvantage: change the system

Outline

Presentation Details These Topics

- HAFS is almost unchanged
  - **Conf files now YAML with the same structure**
  - **Rocoto XML is entirely generated from YAML**
  - Update to Python 3.6
  - **Operational ecFlow suite is unchanged.**
  - **ConfigParser is replaced with CROW in operations**
- Benefits over old system:
  - Direct connection between configuration files and workflow generation
  - Can embed calculations into configuration files
  - No longer using a retired Python version
- Disadvantage: change the system

Part 1
ConfigParser vs. YAML

- HAFS is almost unchanged
  - **Conf files now YAML with the same structure**
  - Rocoto XML is entirely generated from YAML
  - Update to Python 3.6
  - Operational ecFlow suite is unchanged.
  - ConfigParser is replaced with CROW in operations
- Benefits over old system:
  - Direct connection between configuration files and workflow generation
  - Can embed calculations into configuration files
  - No longer using a retired Python version
- Disadvantage: change the system

HAFS Configuration

ConfigParser Configuration Language

Data from later files overrides earlier ones

file1.conf

```
[section]
var1=3
var2=4
```

file2.conf

```
[section]
var1=7
var3=6
```

In-memory result

```
[section]
var1=7
var2=4
var3=6
```

HAFS Configuration

ConfigParser Configuration Language

Multiple sections

file1.conf

```
[section1]
var1=3
var2=4

[section2]
var1=8
var4=7
```

file2.conf

```
[section1]
var1=7
var3=6

[section2]
var2=11
var3=25
var4=16
```

In-memory result

```
[section1]
var1=7
var2=4
var3=6

[section2]
var1=8
var2=11
var3=25
var4=16
```

HAFS Configuration

ConfigParser Configuration Language

Variables are scoped by section.

Section 1 and 2 have their own var1

file1.conf

```
[section1]
var1=3
var2=4

[section2]
var1=8
var4=7
```

file2.conf

```
[section1]
var1=7
var3=6

[section2]
var2=11
var3=25
var4=16
```

In-memory result

```
[section1]
var1=7
var2=4
var3=6

[section2]
var1=8
var2=11
var3=25
var4=16
```

HAFS Configuration

ConfigParser Configuration Language

References between sections using {} braces

References within a section using {} braces

file1.conf

```
[section1]
var1=3
var2=4

[section2]
var1={section1/var1}
var4=7
```

file2.conf

```
[section1]
var1=7
var3=6

[section2]
var2=11
var3={var2}
var4=16
```

In-memory result

```
[section1]
var1=7
var2=4
var3=6

[section2]
var1=7
var2=11
var3=11
var4=16
```

HAFS Configuration

ConfigParser Configuration Language

Special sections: config, dir, exe

References to missing variables in a section look at the special sections

file1.conf

```
[config]
dog=roscoe
cat=apollo
```

file2.conf

```
[action]
story={cat} chases {dog}
reason={dog} annoyed {cat}
```

In-memory result

```
[config]
dog=roscoe
cat=apollo

[action]
story=apollo chases roscoe
reason=roscoe annoyed apollo
```

HAFS Configuration

CROW Configuration Language

Special sections: config, dir, exe

Use all.varname to access special sections

file1.yaml

```
config:
  dog: roscoe
  cat: apollo

math:
  two: !calc 1+1
```

file2.yaml

```
action:
  two: !iexpand "{math.two!s}"
  story: !expand "{all.cat} chases {all.dog}"
  reason: !uexpand "{all.dog} annoyed {all.cat}"
```

HAFS Configuration

CROW Configuration Language

file1.yaml

```
config:
  dog: roscoe
  cat: apollo

math:
  two: “two”
```

file2.yaml

```
math:
  two: !calc “1+1”

action:
  two: !iexpand “{math.two!s}”
  story: !expand “{all.cat} chases {all.dog}”
  reason: !uexpand “{all.dog} annoyed {all.cat}”
```

In memory. Simplified version.

```
doc = dict_eval({
  “config”: dict_eval({
    “dog”: “roscoe”,
    “cat”: “apollo”
    }),
  “math”: dict_eval({ “two”: calc(“1+1”) }),
  “action”: dict_eval({
    “two”: iexpand(“{math.two!s}”),
    “story”: expand(“{all.cat} chases {all.dog}”),
    “reason”: uexpand(“{all.dog} annoyed {all.cat}”)
    }),
  })
```

HAFS Configuration

CROW calculations: before calculating

```
  “action”: dict_eval({
    “two”: iexpand(“{math.two!s}”),
    “story”: expand(“{all.cat} chases {all.dog}”),
    “reason”: uexpand(“{all.dog} annoyed {all.cat}”)
    }),

Calculations = {
    “two”: iexpand(“{math.two!s}”),
    “story”: expand(“{all.cat} chases {all.dog}”),
    “reason”: uexpand(“{all.dog} annoyed {all.cat}”)
    }

Cache = {} # empty dict
```

```
> print(doc.action.two)
2

> print(doc.action.story)
apollo chases roscoe

> print(doc.action.reason)
roscoe annoyed apollo
```

What happens in memory?

HAFS Configuration

CROW calculations: after calculating

```
 “action”: dict_eval({
   “two”: iexpand(“{math.two!s}”),
   “story”: expand(“{all.cat} chases {all.dog}”),
   “reason”: uexpand(“{all.dog} annoyed {all.cat}”)
   }),

Calculations = {
   “two”: 2,
   “story”: expand(“{all.cat} chases {all.dog}”),
   “reason”: uexpand(“{all.dog} annoyed {all.cat}”)
   }

Cache = {
   “story”: “apollo chases roscoe”
   }
```

```
> print(doc.action.two)
2

> print(doc.action.story)
apollo chases roscoe

> print(doc.action.reason)
roscoe annoyed apollo
```

What happens in memory?

HAFS Configuration

CROW calculations: after calculating

```
  “action”: dict_eval({
    “two”: iexpand(“{math.two!s}”),
    “story”: expand(“{all.cat} chases {all.dog}”),
    “reason”: uexpand(“{all.dog} annoyed {all.cat}”)
    }),

Calculations = {
    “two”: 2,
    “story”: expand(“{all.cat} chases {all.dog}”),
    “reason”: uexpand(“{all.dog} annoyed {all.cat}”)
    }

Cache = {
    “story”: “apollo chases roscoe”
    }
```

When writing back to disk as CROW YAML, the calculations are written.

!iexpand replaced the calculation with the result, so the !iexpand is not written out

!uexpand, !expand are written back as they were in the input file

HAFS Configuration

CROW: f-strings and expand

CROW is based on Python f-strings

```
x=1
y=1
print(f“x+y={x}+{y}={x+y}”)
# prints x+y=1+1=2

# Use !s to pass through str()
# Use !r to pass through repr()

adjective=”tasty”
fruits=”oranges”

print(f’{fruits} are {adjective}’)
# prints oranges are tasty

print(f’{fruits} are {adjective!s}’)
# prints oranges are tasty

print(f’{fruits} are {adjective!r}’)
# prints oranges are ‘tasty’
```

HAFS Configuration
CROW: f-strings and expand

CROW is based on Python f-strings

```
x=1
y=1
print(f“x+y={x}+{y}={x+y}”)
# prints x+y=1+1=2

# Use !s to pass through str()
# Use !r to pass through repr()

adjective=”tasty”
fruits=”oranges”

print(f’{fruits} are {adjective}’)
# prints oranges are tasty

print(f’{fruits} are {adjective!s}’)
# prints oranges are tasty

print(f’{fruits} are {adjective!r}’)
# prints oranges are ‘tasty’
```

```
parm:
  x: 1
  y: 1
  adjective: tasty
  fruits: oranges

demo:
  math: !calc doc.parm.x+doc.parm.y
  use_repr: !expand ‘{parm.fruits} are {parm.adjective!r}
  use_str: !expand ‘{parm.fruits} are {parm.adjective!s}
  neither:  !expand ‘{parm.fruits} are {parm.adjective}
```

```
… python code ...

conf=… read the yaml files …
print(conf.demo.use_repr) # => oranges are ‘tasty’
print(conf.demo.use_str) # => oranges are tasty
print(conf.demo.neither) # => oranges are tasty
```

HAFS Configuration

CROW: eval and calc

CROW is based on Python eval

```
x=1
y=1
print(eval(“x+y”)
# prints 2
```

```
parm:
  x: 1
  y: 1

demo:
  math: !calc doc.parm.x+doc.parm.y


… python code ...

conf=… read the yaml files …
print(conf.demo.math) # => 2
```

HAFS Configuration

CROW: eval and calc

CROW is based on Python eval

```
x=1
y=1
print(eval("x+y"))
# prints 2
```

```
parm:
  x: 1
  y: 1

demo:
  math: !calc doc.parm.x+doc.parm.y
```

"**doc**" is top of document

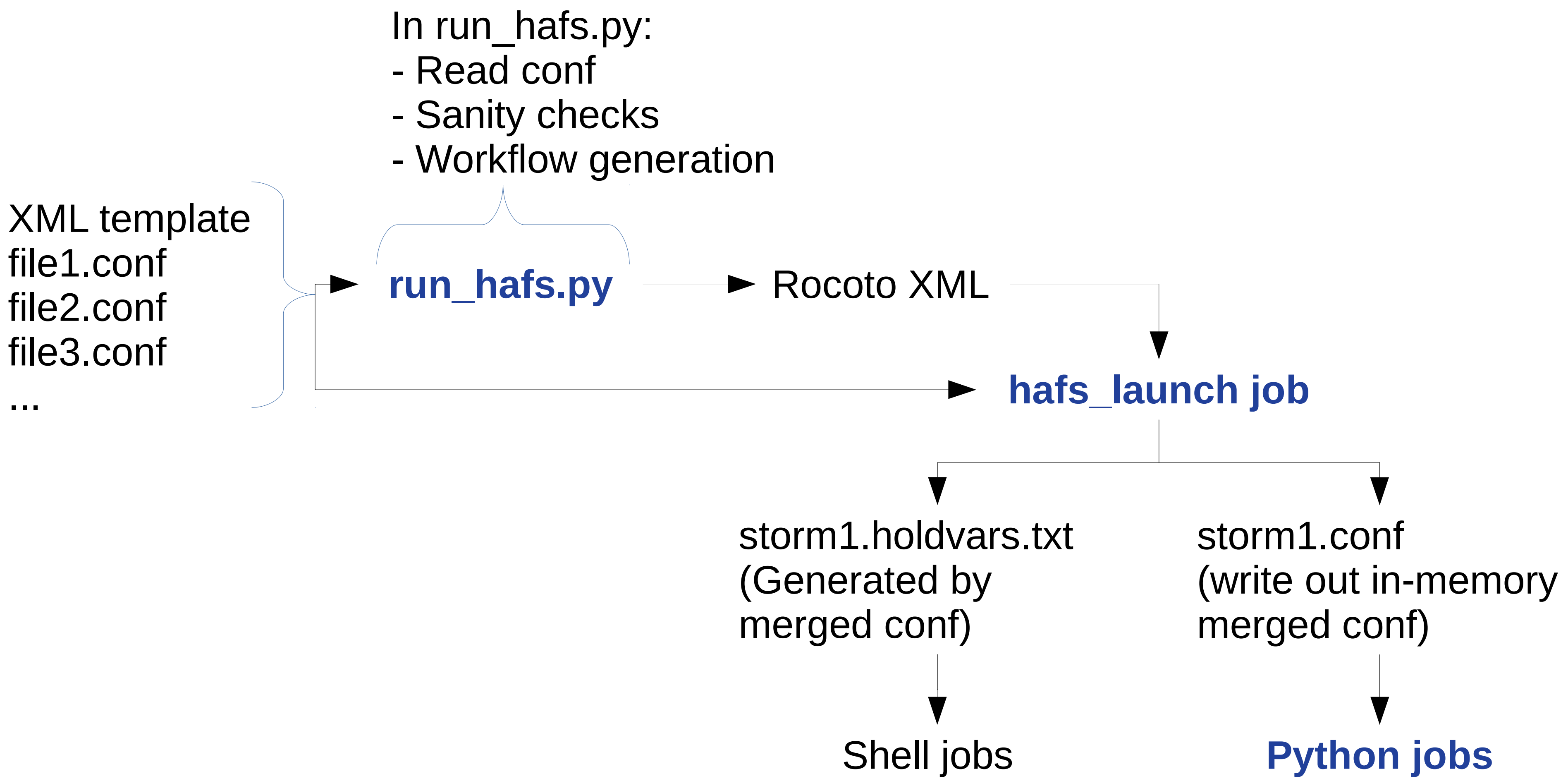
```
… python code ...

conf=… read the yaml files …
print(conf.demo.math) # => 2
```

Part 2
Research Workflow: Rocoto XML Generation

- HAFS is almost unchanged
  - Conf files now YAML with the same structure
  - **Rocoto XML is entirely generated from YAML**
  - Update to Python 3.6
  - Operational ecFlow suite is unchanged.
  - ConfigParser is replaced with CROW in operations
- Benefits over old system:
  - Direct connection between configuration files and workflow generation
  - Can embed calculations into configuration files
  - No longer using a retired Python version
- Disadvantage: change the system

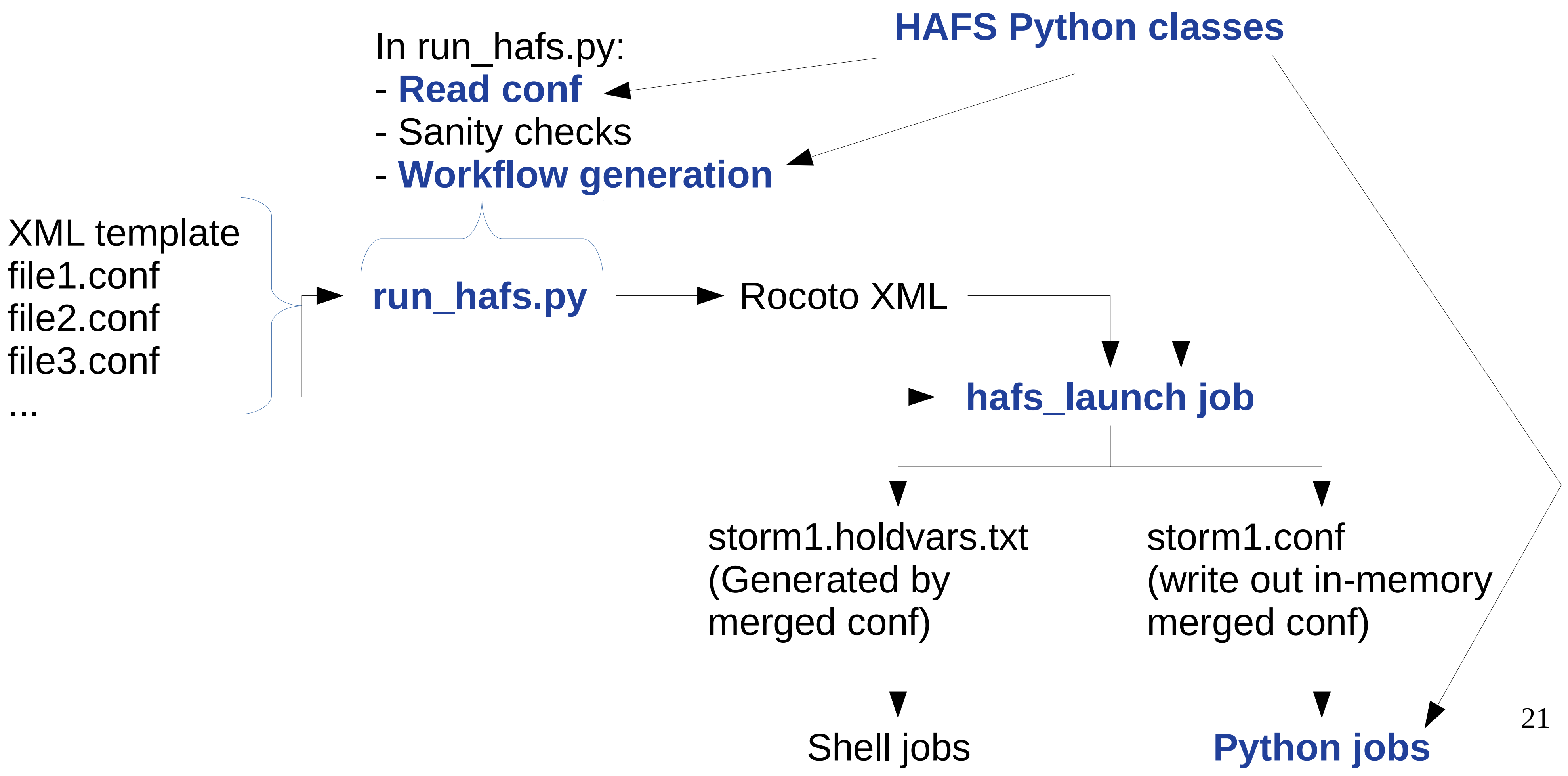
HAFS Configuration

Research Workflow (Simplified)



HAFS Configuration

Research Workflow (Simplified)



HAFS Configuration
Research Workflow XML Generation

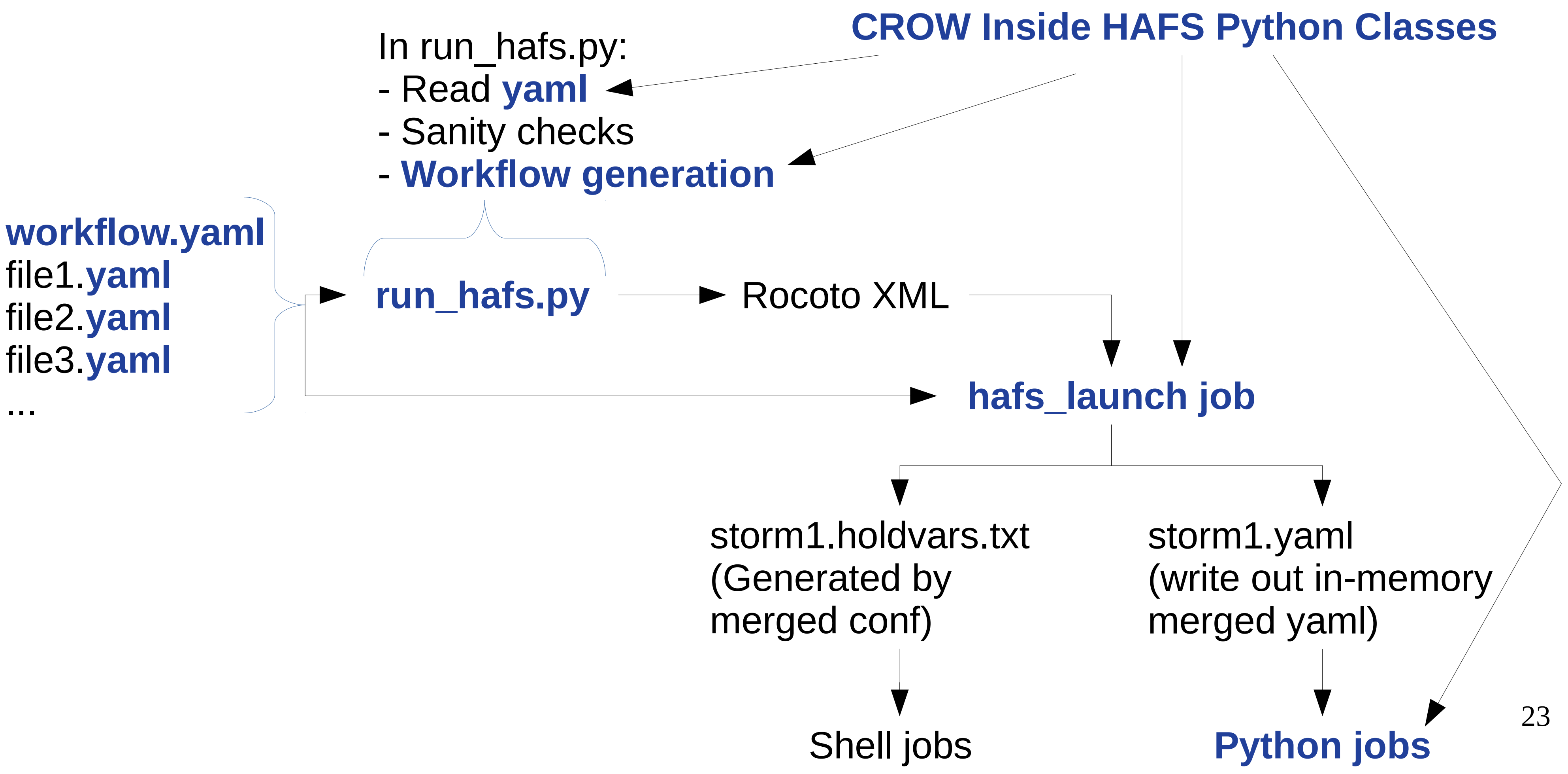
Conf-based HAFS makes the Rocoto XML from simple text replacement:

@[VARNAME]

Refers back to a dict created in run_hafs.py

```
<?xml version="1.0"?>

<!DOCTYPE workflow [
…
  <!ENTITY SCRUB_WORK "@[SCRUB_WORK]">
  <!ENTITY SCRUB_COM "@[SCRUB_COM]">
…
  <task name="launch" maxtries="99">
...
```

HAFS Configuration
Regions Impacted by Change in Config System



HAFS Configuration

Research Workflow CROW to XML

CROW YAML includes workflow definitions

Trade information between sections

Automatic dependency generation

```
workflow:
  …
    launch: !Task
      Inherit: !Inherit [ [ doc.launch_task, '.*', {recurse: inherit} ] ]
      Validate: inherit
      Trigger: !Depend ( launch.at(-6*3600) | ~ suite.has_cycle(-6*3600) )
      Time: !timedelta '+3:20:00'
```

hafs_tasks.yaml

```
launch_task:
  Inherit: !Inherit [ [ doc.task_defaults, '.*', { recurse: inherit } ] ]
  accounting: !calc doc.accounting.serial
  resources: !calc doc.resources.small_serial
  TOTAL_TASKS: 1
  ...
```

HAFS Configuration

Research Workflow CROW to XML

hafs_tasks.yaml

```
launch_task:
  Inherit: !Inherit [ [ doc.task_defaults, '.*',
                             { recurse: inherit } ] ]
  accounting: !calc doc.accounting.serial
  resources: !calc doc.resources.small_serial
  TOTAL_TASKS: 1
  ...
```

CROW YAML includes workflow definitions

Trade information between sections

Automatic dependency generation

Generate resource specifications

sites/hera.yaml

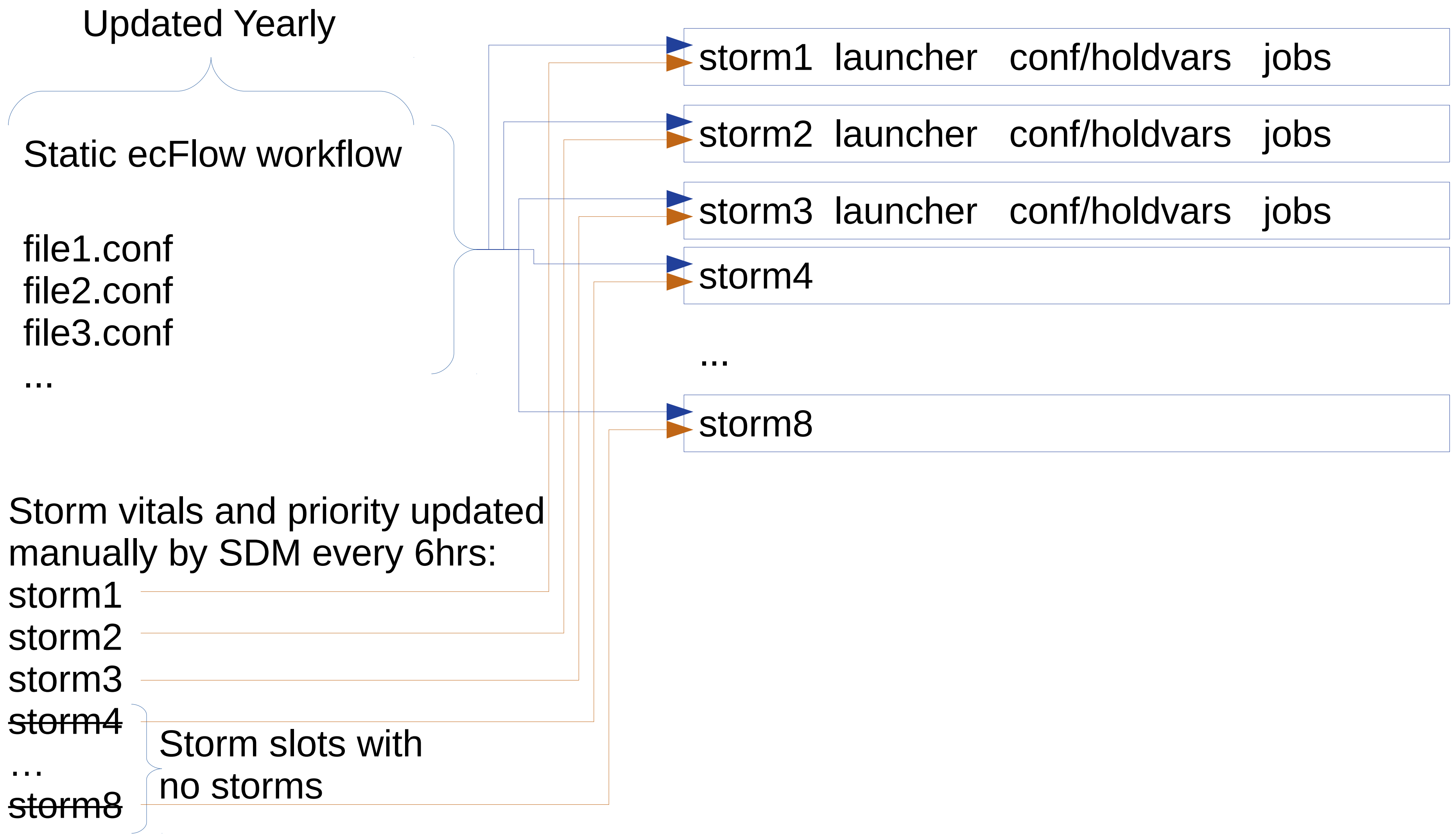
```
accounting:
  scheduler_settings:
    name: Slurm
    physical_cores_per_node: 40
    logical_cpus_per_core: 2
  serial:
    queue: batch # queue for serial jobs
    account: !ref doc.accounting.account
resources:
…
  small_serial: !JobRequest
    - &small_serial
      <<: *resources_small_serial
      walltime: !timedelta '00:15:00'
…
  chgres_ic: !JobRequest
    - OMP_NUM_THREADS: 1
      mpi_ranks: 120
      max_ppn: 40
      exclusive: true
      walltime: !timedelta '00:30:00'
```

Part 3
Operational Workflow

- HAFS is almost unchanged
  - Conf files now YAML with the same structure
  - Rocoto XML is entirely generated from YAML
  - Update to Python 3.6
  - **Operational ecFlow suite is unchanged.**
  - **ConfigParser is replaced with CROW in operations**
- Benefits over old system:
  - Direct connection between configuration files and workflow generation
  - Can embed calculations into configuration files
  - No longer using a retired Python version
- Disadvantage: change the system

HAFS Configuration

Operational Workflow (Simplified)


Updated Yearly

Static ecFlow workflow

file1.conf
file2.conf
file3.conf
...

storm1 launcher conf/holdvars jobs
storm2 launcher conf/holdvars jobs
storm3 launcher conf/holdvars jobs
storm4
...
storm8

Storm vitals and priority updated manually by SDM every 6hrs:
storm1
storm2
storm3
~~storm4~~
…
~~storm8~~

Storm slots with no storms

HAFS Configuration

Operational Workflow (Storm 3)

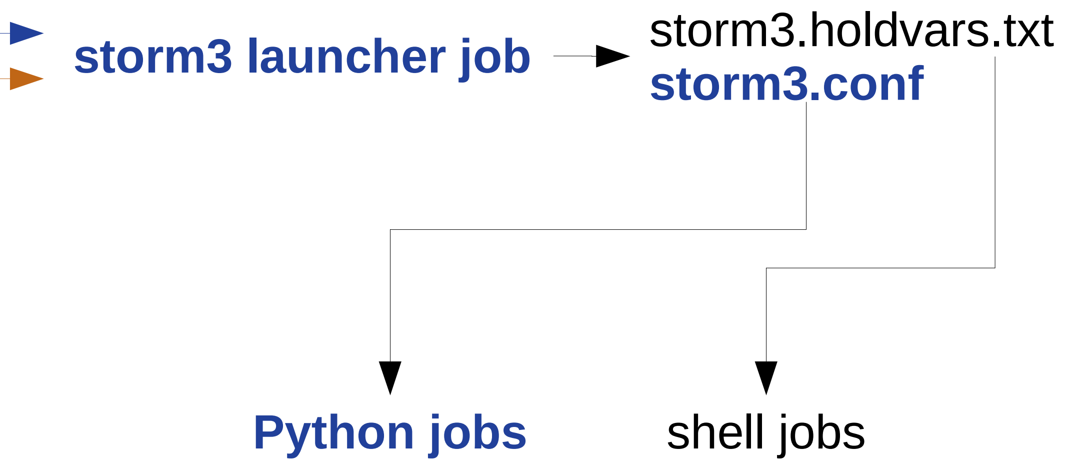
Updated Yearly

Static ecFlow workflow

file1.conf
file2.conf
file3.conf
...

Zoom in on one storm's workflow.

Regions of the workflow connected to the configuration system are in **blue**.

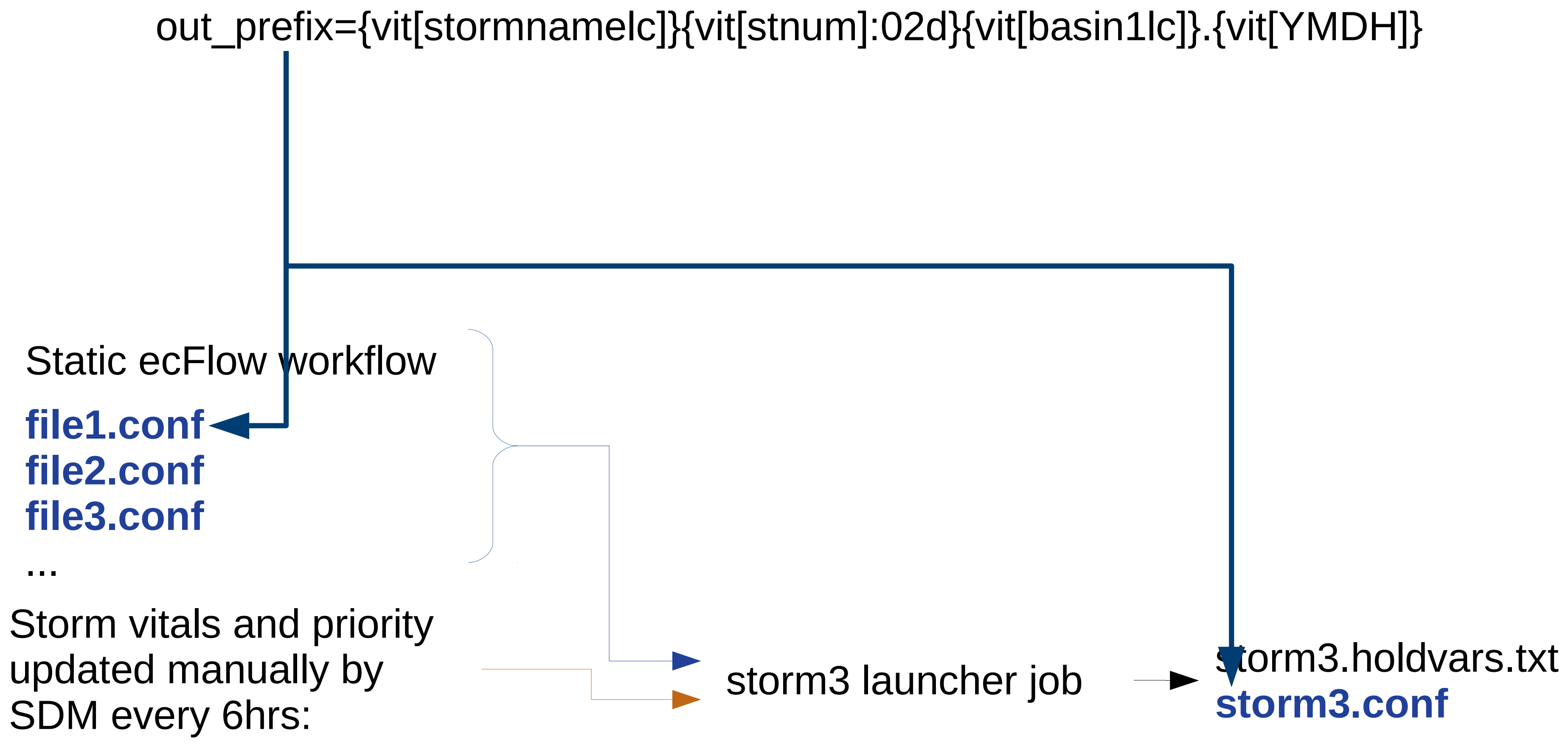


Storm vitals and priority updated manually by SDM every 6hrs:
storm1
storm2
storm3
~~storm4~~
…
~~storm8~~

Storm slots with no storms

HAFS Configuration

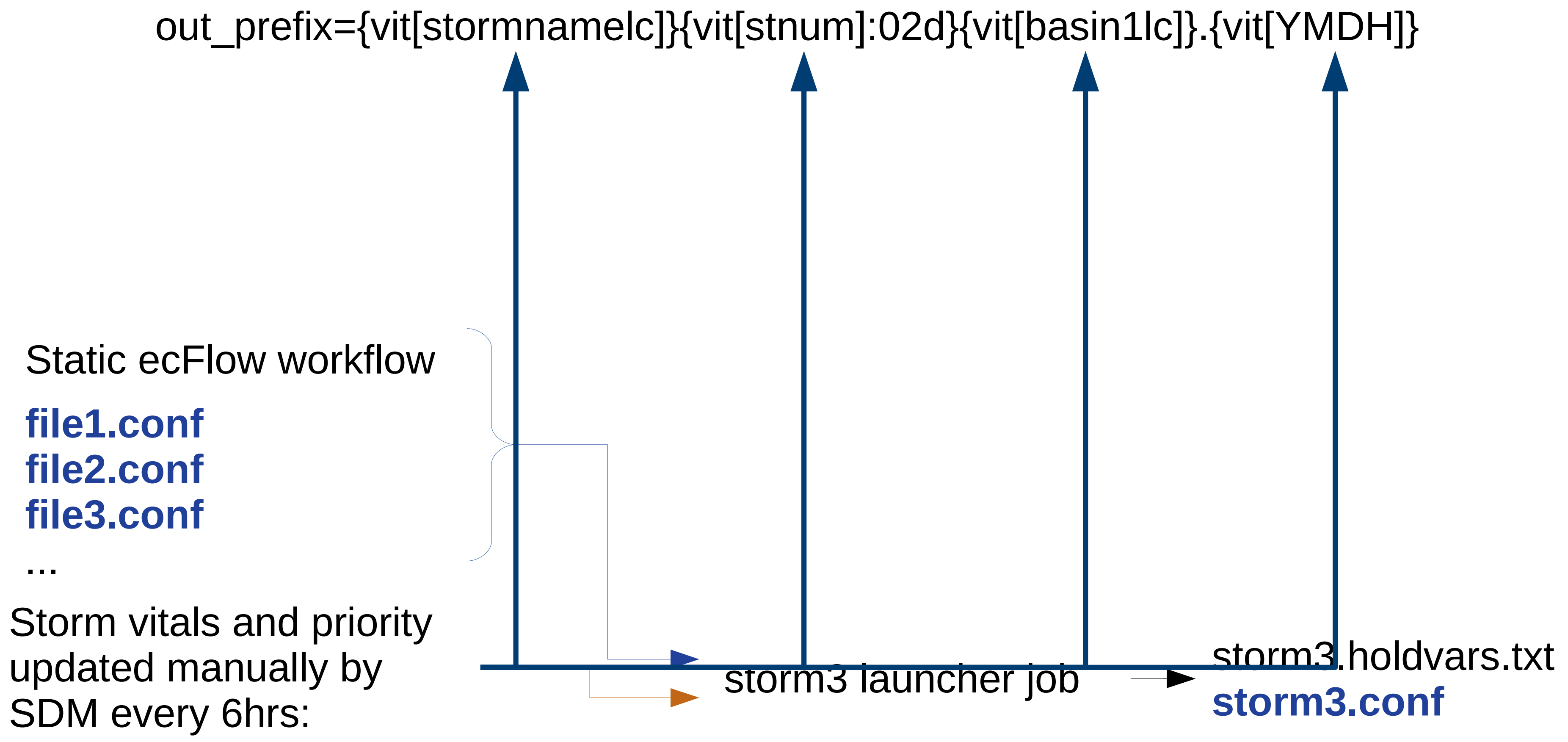
Operational Configuration at Runtime



out_prefix={vit[stormnamelc]}{vit[stnum]:02d}{vit[basin1lc]}.{vit[YMDH]}
Static ecFlow workflow
file1.conf
file2.conf
file3.conf
...
Storm vitals and priority updated manually by SDM every 6hrs:
storm3 launcher job
storm3.holdvars.txt
storm3.conf

HAFS Configuration

Operational Configuration at Runtime


out_prefix={vit[stormnamelc]}{vit[stnum]:02d}{vit[basin1lc]}.{vit[YMDH]}
Static ecFlow workflow
file1.conf
file2.conf
file3.conf
...
Storm vitals and priority updated manually by SDM every 6hrs:
storm3 launcher job
storm3.holdvars.txt
storm3.conf

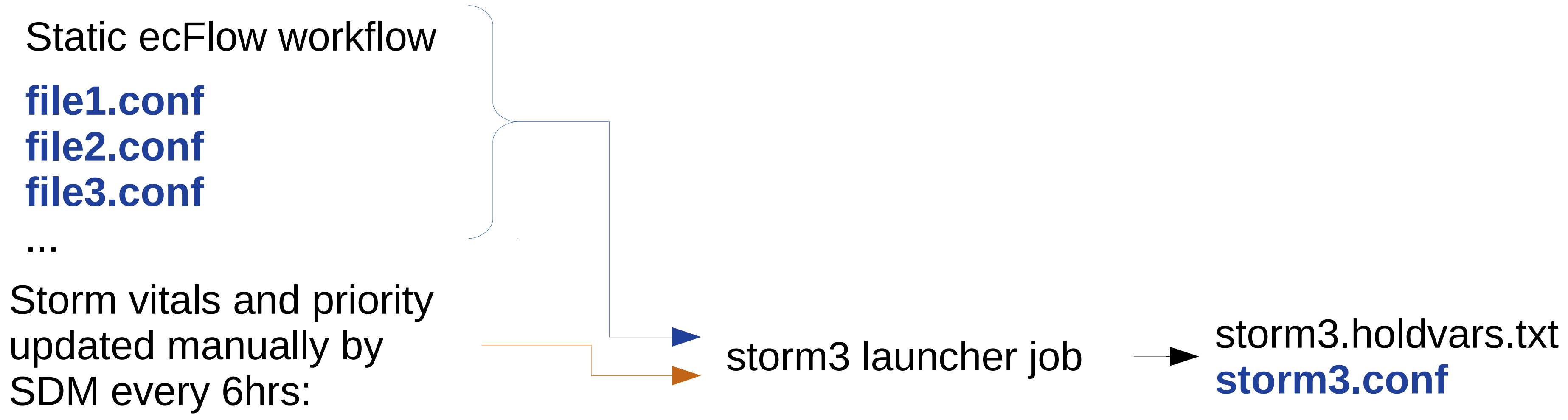
HAFS Configuration

Operational Configuration at Runtime

Some variables change with every storm and cycle in a way that cannot be predicted until runtime.

```
out_prefix={vit[stormnamelc]}{vit[stnum]:02d}{vit[basin1lc]}.{vit[YMDH]}
```

This is in hafs.conf and storm3.conf
Parsed at runtime by storm3 launcher job and several later jobs.



HAFS Configuration

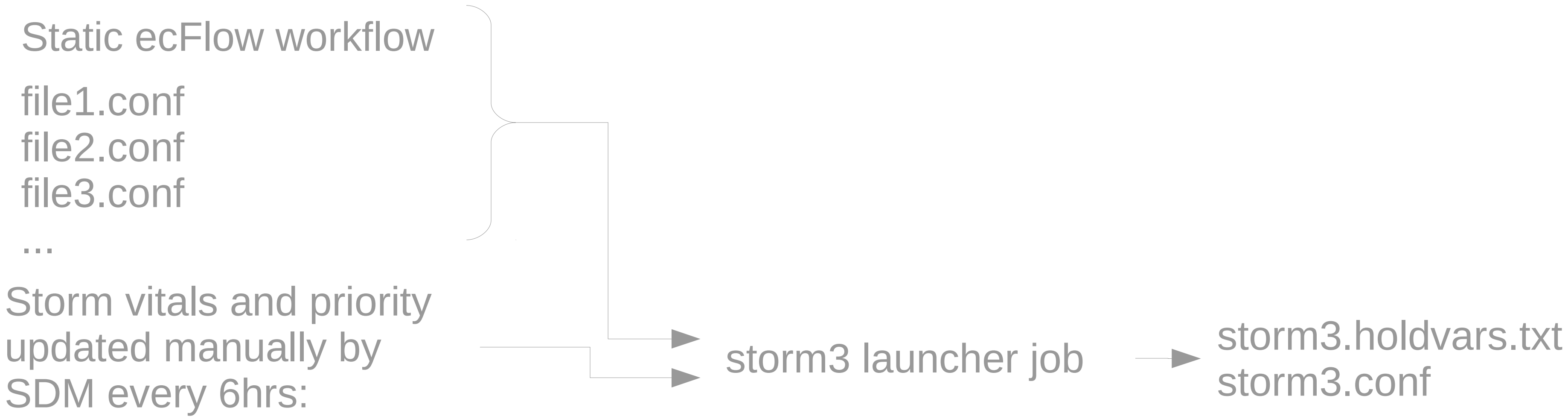
Operational Configuration at Runtime

Some variables change with every storm and cycle in a way that cannot be predicted until runtime.

```
out_prefix={vit[stormnamelc]}{vit[stnum]:02d}{vit[basin1lc]}.{vit[YMDH]}
```

This is in hafs.conf and storm3.conf
Parsed at runtime by storm3 launcher job and several later jobs.

The configuration system is, out of necessity, in operations.



HAFS Configuration

Operational Workflow no CROW (Storm 3)

Updated Yearly

$\overbrace{\text{Static ecFlow workflow}}$

file1.conf
file2.conf
file3.conf
...

Zoom in on one storm’s workflow.

Regions of the workflow connected to the configuration system are in **blue**.

$$\begin{tikzcd}
\text{Static ecFlow workflow, file1.conf, file2.conf, file3.conf, ...} \arrow[r] & \textbf{storm3 launcher job} \arrow[r] & \text{storm3.holdvars.txt} \arrow[d] \\
\text{storm3} \arrow[ru] & \textbf{Python jobs} & \textbf{storm3.conf} \arrow[l] \\
 & & \text{shell jobs}
\end{tikzcd}$$

Storm vitals and priority updated manually by SDM every 6hrs:
storm1
storm2
storm3
~~storm4~~
…
~~storm8~~

Storm slots with no storms

HAFS Configuration

Operational Workflow with CROW (Storm 3)

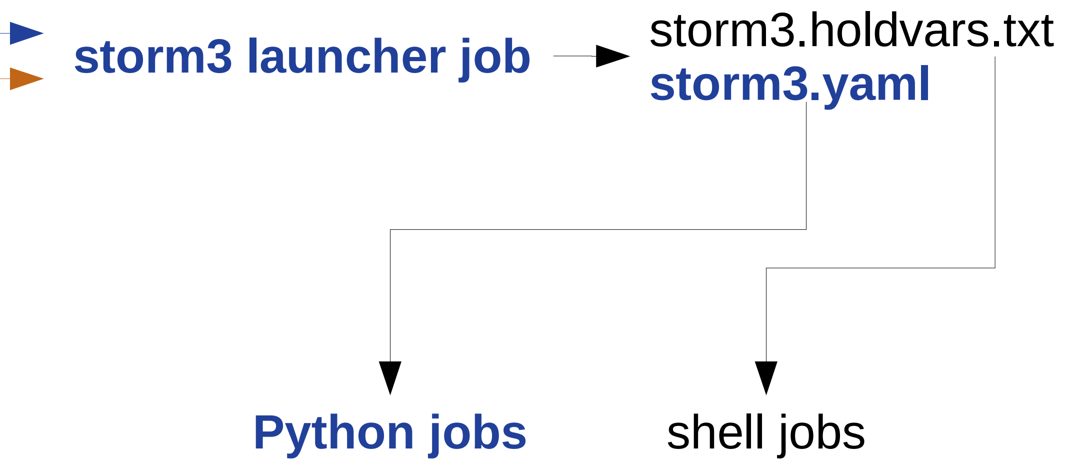
Updated Yearly

Static ecFlow workflow

file1.yaml
file2.yaml
file3.yaml
...

Zoom in on one storm's workflow.

Regions of the workflow connected to the CROW configuration system are in **blue**.



Storm vitals and priority updated manually by SDM every 6hrs:
storm1
storm2
storm3
~~storm4~~
…
~~storm8~~

Storm slots with no storms

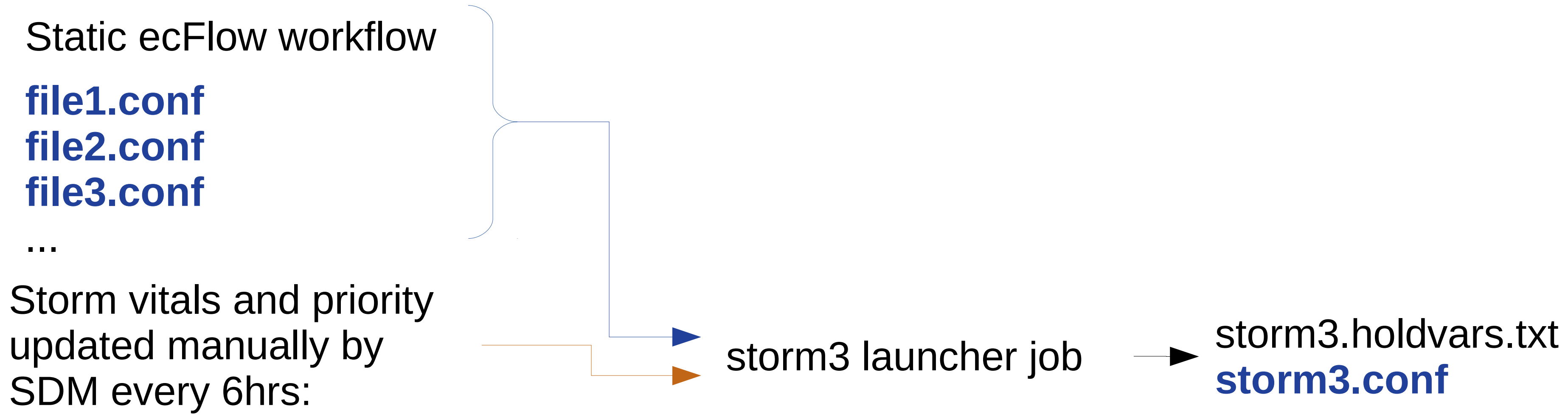
HAFS Configuration

Operational Configuration no CROW

Some variables change with every storm and cycle in a way that cannot be predicted until runtime.

out_prefix={vit[stormnamelc]}{vit[stnum]:02d}{vit[basin1lc]}.{vit[YMDH]}

This is in hafs.conf and storm3.conf
Parsed at runtime by storm3 launcher job and several later jobs.



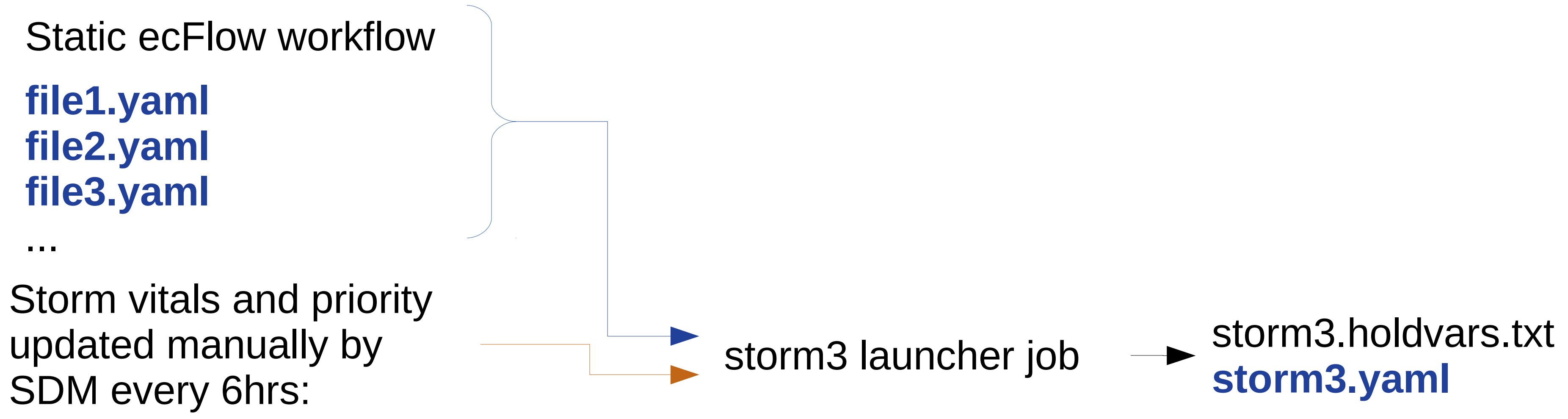
HAFS Configuration

Operational Configuration with CROW

Some variables change with every storm and cycle in a way that cannot be predicted until runtime.

out_prefix: !uexpand "{vit.stormnamelc}{vit.stnum:02d}{vit.basin1lc}.{vit.YMDH}"

This is in hafs.yaml and storm3.yaml
Parsed at runtime by storm3 launcher job and several later jobs.



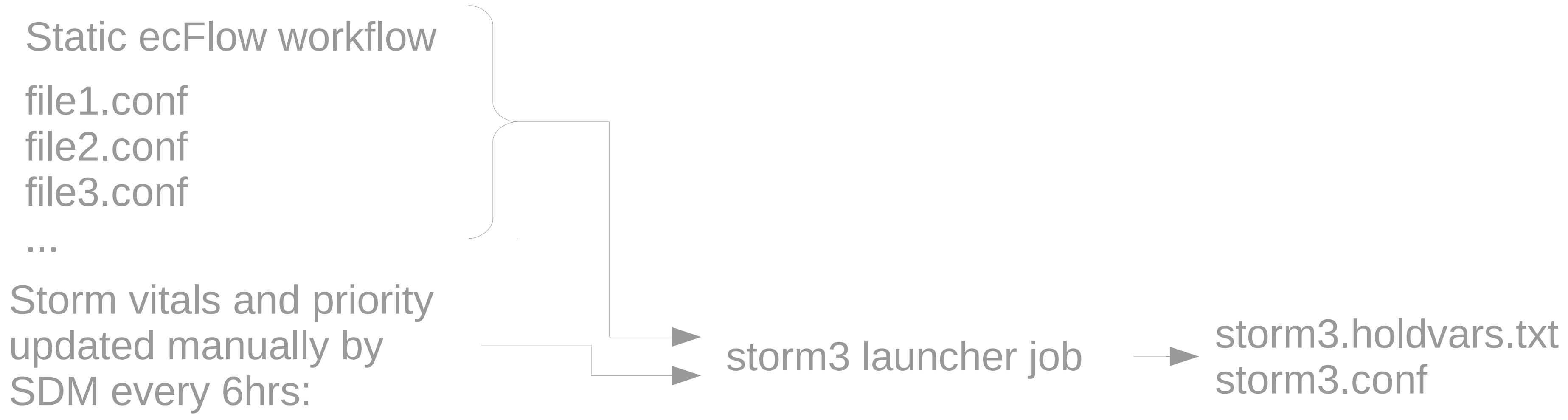
HAFS Configuration
Operational Configuration no CROW

Some variables change with every storm and cycle in a way that cannot be predicted until runtime.

out_prefix={vit[stormnamelc]}{vit[stnum]:02d}{vit[basin1lc]}.{vit[YMDH]}

This is in hafs.conf and storm3.conf
Parsed at runtime by storm3 launcher job and several later jobs.

The configuration system is, out of necessity, in operations.



HAFS Configuration

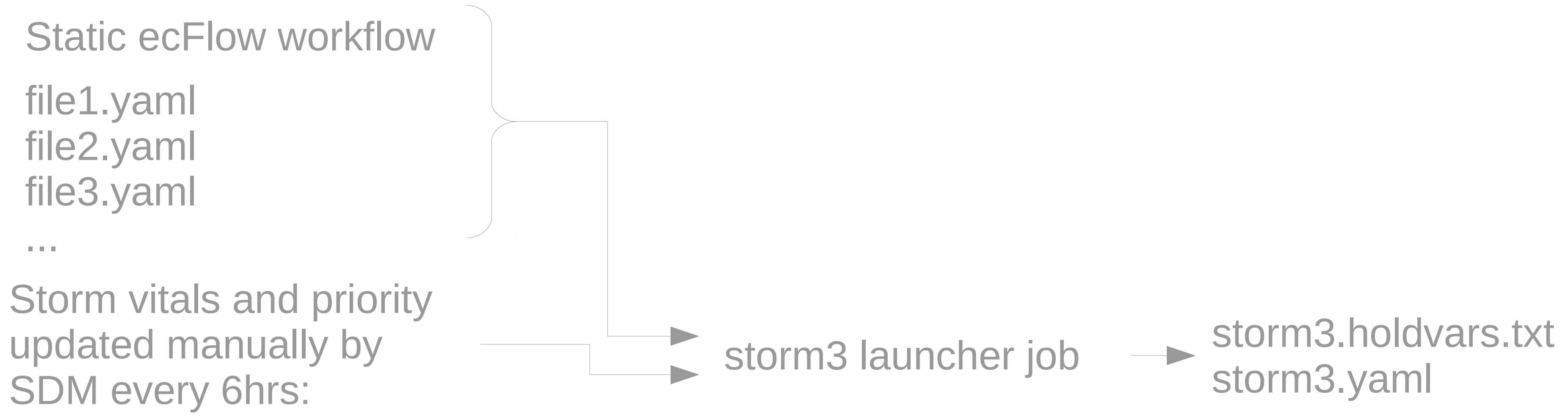
Operational Configuration with CROW

Some variables change with every storm and cycle in a way that cannot be predicted until runtime.

out_prefix: !uexpand "{vit.stormnamelc}{vit.stnum:02d}{vit.basin1lc}.{vit.YMDH}"

This is in hafs.yaml and storm3.yaml
Parsed at runtime by storm3 launcher job and several later jobs.

The CROW configuration system is, out of necessity, in operations.



Executive Summary
Whole Project in One Slide

- HAFS is almost unchanged
  - Conf files now YAML with the same structure
  - Rocoto XML is entirely generated from YAML
  - Update to Python 3.6
  - Operational ecFlow suite is unchanged.
  - ConfigParser is replaced with CROW in operations
- Benefits over old system:
  - Direct connection between configuration files and workflow generation
  - Can embed calculations into configuration files
  - No longer using a retired Python version
- Disadvantage: change the system