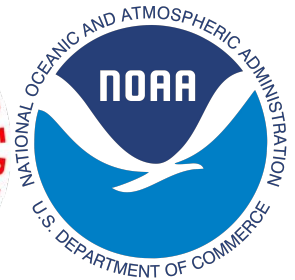


CROW: Python-based Configuration Toolbox for Operational and Development Workflows

CROW Review April 28th 2020

Kate Friedman - NOAA/EMC/EIB

Jian Kuang - I.M. Systems Group, NOAA/EMC





Outline

- Purpose of CROW – What problem are we solving?
- High-level overview of CROW design and use example with global-workflow/FV3GFS
- Development of CROW
- Implementations of CROW
- What will community support for CROW look like?

Problem statement

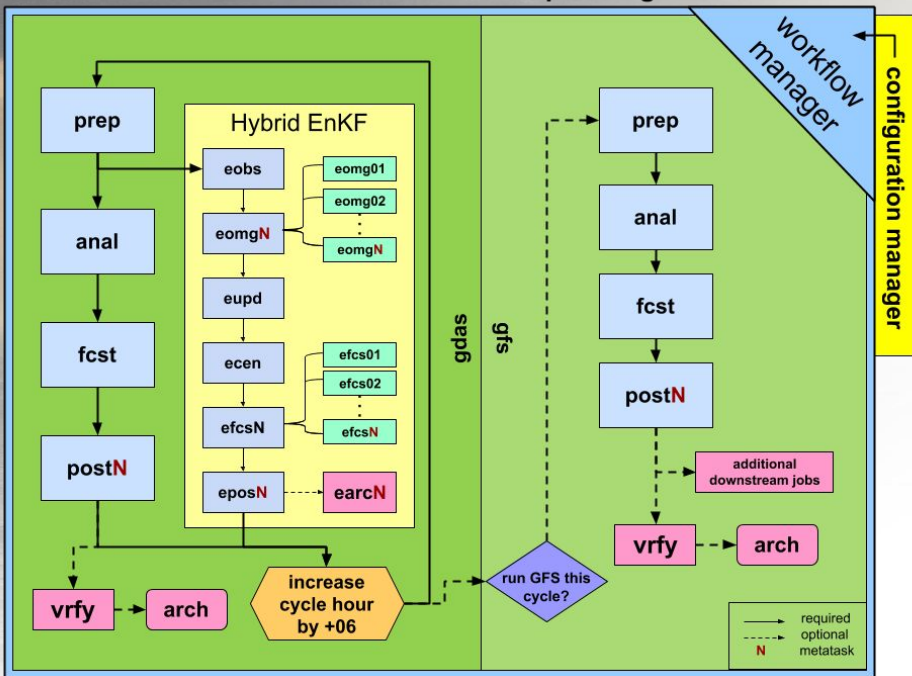
Increasing complexity of operational and developmental workflows poses significant challenges for configuration.

Has become difficult to:

- Ensure consistency and quality in configuration
- Document and archive configuration settings (e.g. provenance)
- Easily switch between workflow managers (ROCOTO, ecFlow, etc.)
- Be extendable and flexible to keep up with evolving modeling system(s)
- Handle R2O transition for multiple modeling systems

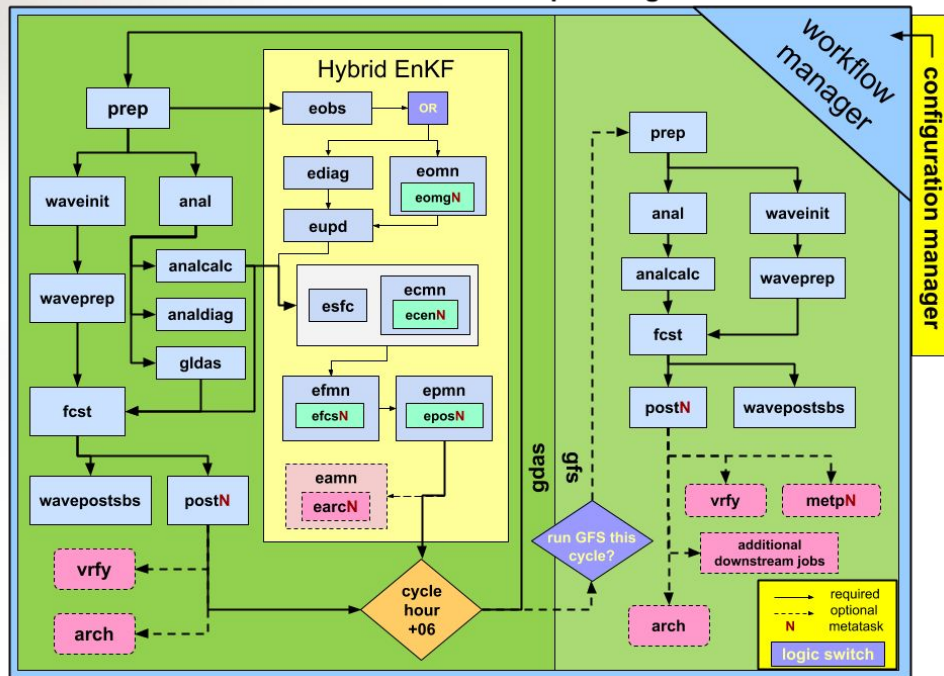
Global Workflow keeps growing!

Global Model Parallel Sequencing



FV3GFS v15 (current ops)

Global Model Parallel Sequencing



FV3GFS v16 (pre-implementation)

Workflow configuration system

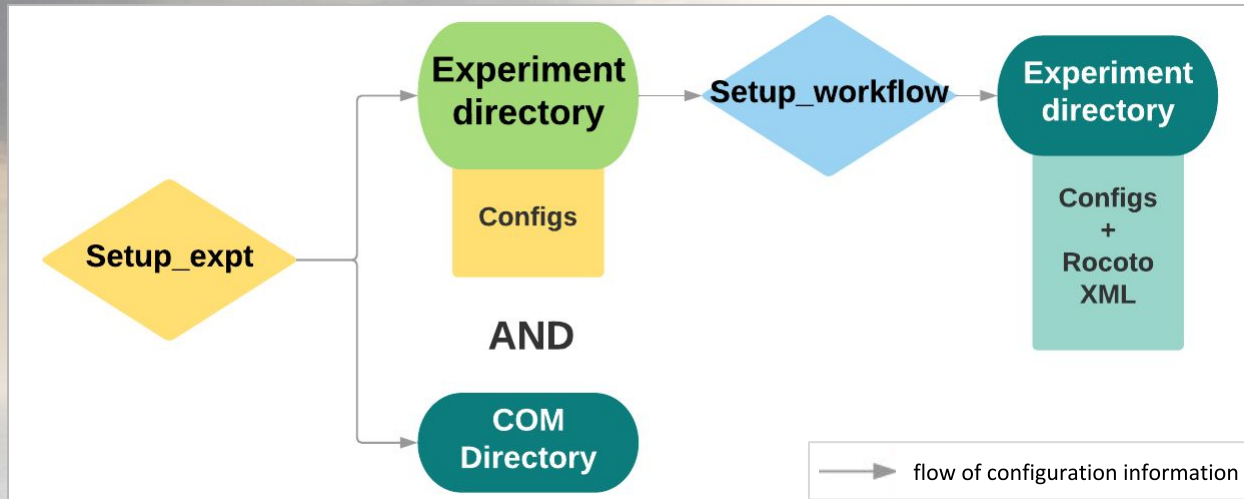
The major purpose of workflow configuration system is to enable the users to:

- Design and select the workflow structure.
- Pass through certain configurable variables into applications that are included in the workflow.

Successful workflow configuration systems include:

- User friendliness - Easier for new users to set up a end-to-end workflow.
- Consistency - Ensuring all components of workflow stay consistent with each other, eliminating a large portion of human errors.
- Accountability - Easier documentation of completed experiment. Records would be archived in a more organized way.
- Extendability - Easy to handle evolving modeling system enhancements.

Current Global Workflow Configuration System



Steps:

- Run setup_expt python script with **numerous (and growing)** command-line options
- Manually edit config files to settings for **each** program involved. These files are essentially pieces of shell scripts that will be “sourced” by the execution scripts at runtime.
- Run setup_workflow python script to generate ROCOTO XML

User friendliness?	Could be friendlier
Consistent?	Yes but limited to one model
Accountable?	Barely
Extendable?	Yes but not easy

What is CROW?

The “**C**onfiguration manager for **R**esearch and **O**perational **W**orkflows” (**CROW**) is a python-based workflow toolbox that automates and streamlines the generation and configuration of NCEP workflows and connects system jobs to a workflow manager.

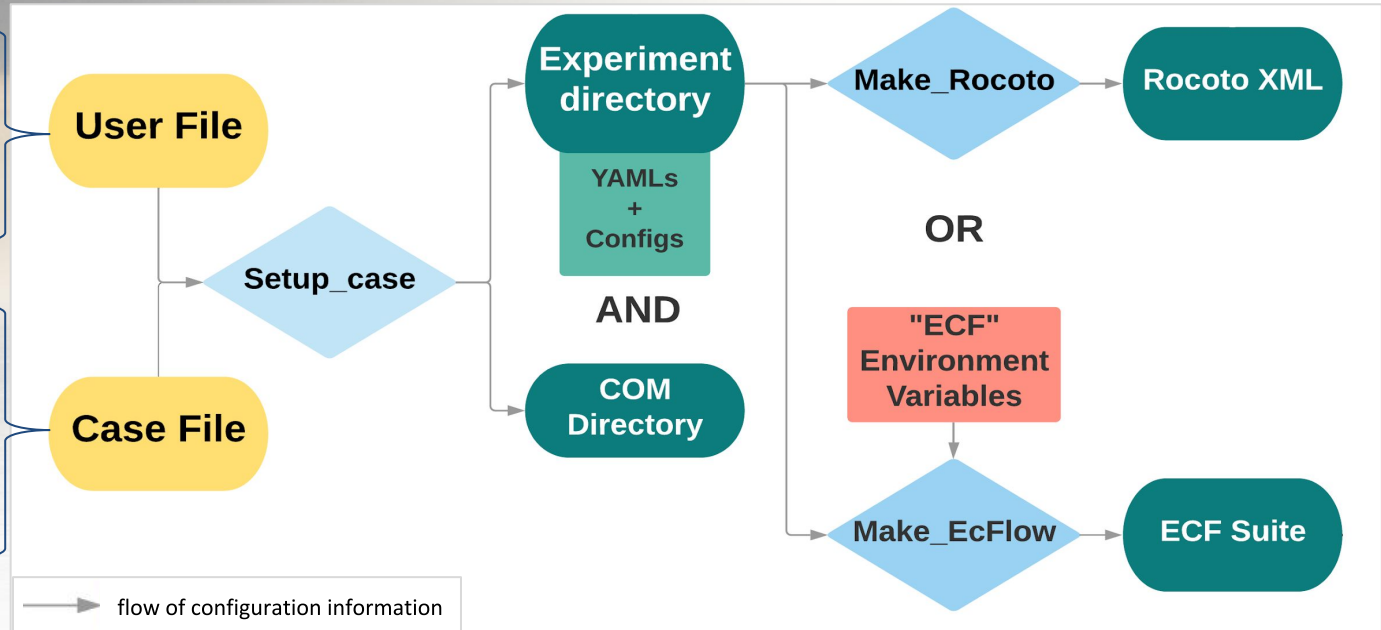
<https://github.com/NOAA-EMC/CROW>

CROW Configuration System

global-workflow implementation

User file* = where a user defines their machine specific project codes and disk space to use (more info in later slide)

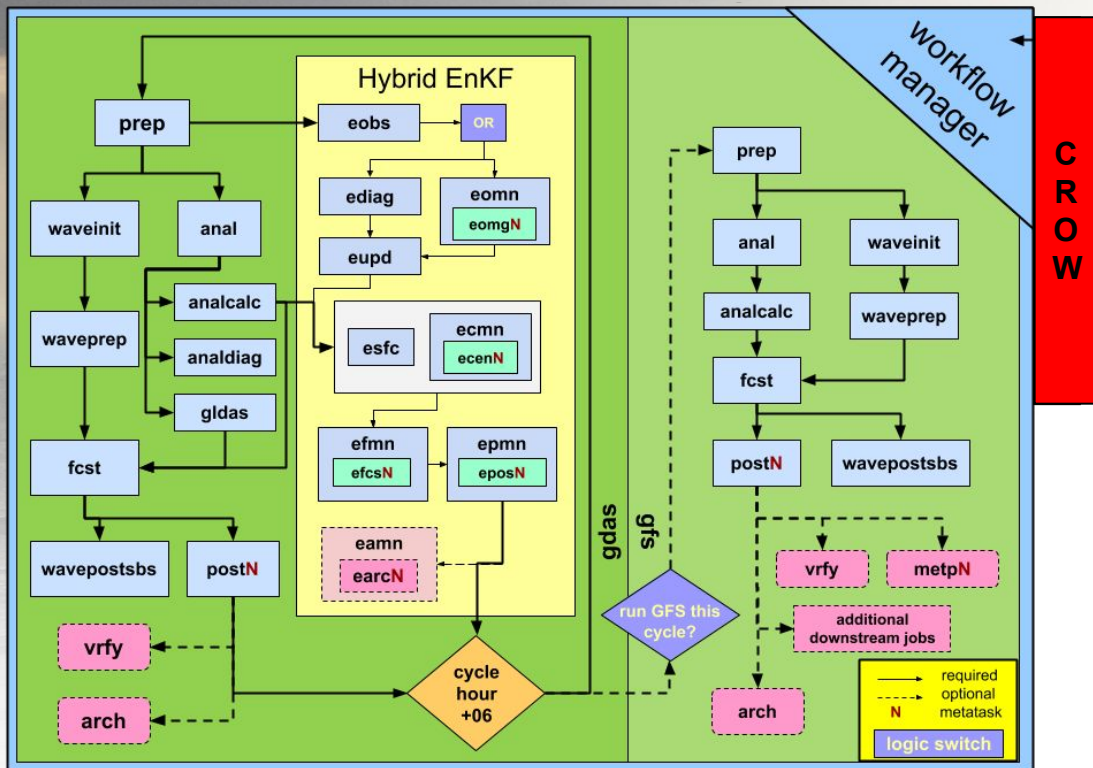
Case file* = the high-level configuration switches for an experiment; stock ones available for specific modes (more info in later slide)



*user and case files can/will be implemented differently between apps

Round-cornered boxes are ASCII files at various stages; yellow color indicates places for user input, while dark green boxes are the final products. Diamonds are scripts run by the user.

How CROW fits with global-workflow



Repository structure:

(top)global-workflow

- **docs** : related documentations
- **fix**: Directory for fix files
- **jobs**: Top level job scripts (J_* job)
- **modulefiles**: modules to be loaded
- **parm**: templates for config
- **scripts**: driving scripts
- **src**: application source code
- **ush**: lower level scripts
- **util**: various utilities
- **workflow**: configuration system
 - **CROW repository with setup and make scripts**
 - **background files**
 - **user file**
 - **case files**

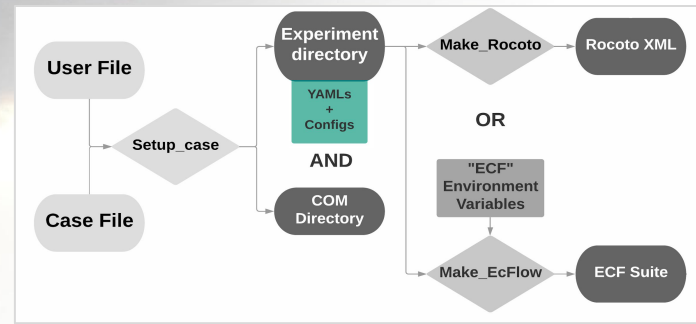
Benefits of CROW

Many benefits are behind the scenes at first:

- Self-contained -- unit test and regression test enabled
- Smart -- validation of configuration settings
- Modularized -- easier troubleshooting (never easy, but easier)
- Generalized -- easier adding/removing jobs
 - easier collaboration
 - smoother R2O transition (now supports ecFlow)
 - turn-key conversion of workflow manager
- Python based -- cross-platform portable, even on a laptop
- “Yamlized” -- shorter learning curve for new users

More to come in the future...

Background Files



To enable CROW-style configuration and provide an interface for CROW, a series of “background files” needs to be provided for a target modeling system (e.g. global-workflow). They can be located under a top-level directory named “workflow” within the target repository.

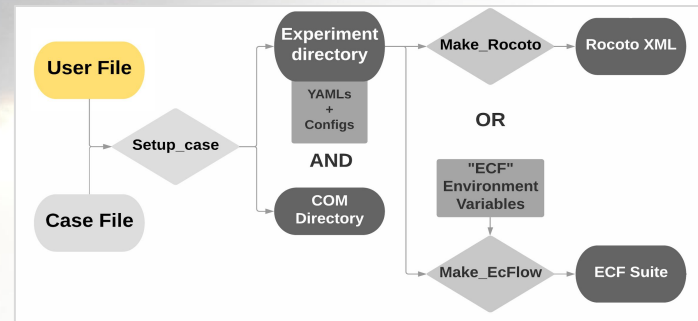
/schema	task template, information of configurable variables (name, type, values)
/defaults	default values for each group of variables
/layout	workflow layouts associated with a certain modeling system
/config	templates of all config files (a mirror of parm/config files in global-workflow)
/runtime	one-step implemented task templates comparing to /schema/task.yaml
/platforms	platform-related information. One file for each supported platform.

These files are ASCII text files formatted in YAML. They collectively serve as the interface between the modeling system and the CROW toolbox.

User File

The user file is where a user enters their user-specific information:

- **user_places**: user-specific locations and paths
 - **EXPROOT**: Designated place for experiment directory (EXPDIR)
- **accounting**: information needed to submit supercomputer job cards
 - **cpu_project**: cpu allocation group
 - **hpss_project**: hpss project code

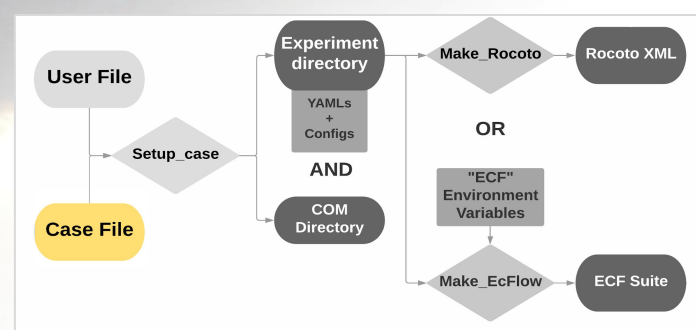


```
user_places: &user_places
EXPROOT: /mnt/lfs3/projects/hfv3gfs/
accounting: &accounting
cpu_project: hfv3gfs
hpss_project: emc-global
```

Complete list of configurable variables can be found in the CROW documentation in GitHub.io page:
<https://noaa-emc.github.io/CROW>

Case File

- **fv3_settings**: resolution and # of levels of deterministic forecast
- **places**: workflow configuration yaml file to use
- **settings**:
 - **SDATE** - start date of experiment (YYYY-MM-DDtCC:mm:ss)
 - **EDATE** - end date of experiment (YYYY-MM-DDtCC:mm:ss)
 - **run_gsi** - whether to run data assimilation ('Yes' if cycling, 'No' if free-forecast mode)
 - **chgres_and_convert_ics** - use when doing free-forecasts to run extra IC-making jobs
 - **gfs_cyc** - frequency to run gfs long forecast



case:

fv3_settings:

CASE: C192

LEVS: 65

places:

workflow_file: workflow/free_forecast_gfs.yaml

settings:

SDATE: 2019-05-01t00:00:00

EDATE: 2019-05-01t06:00:00

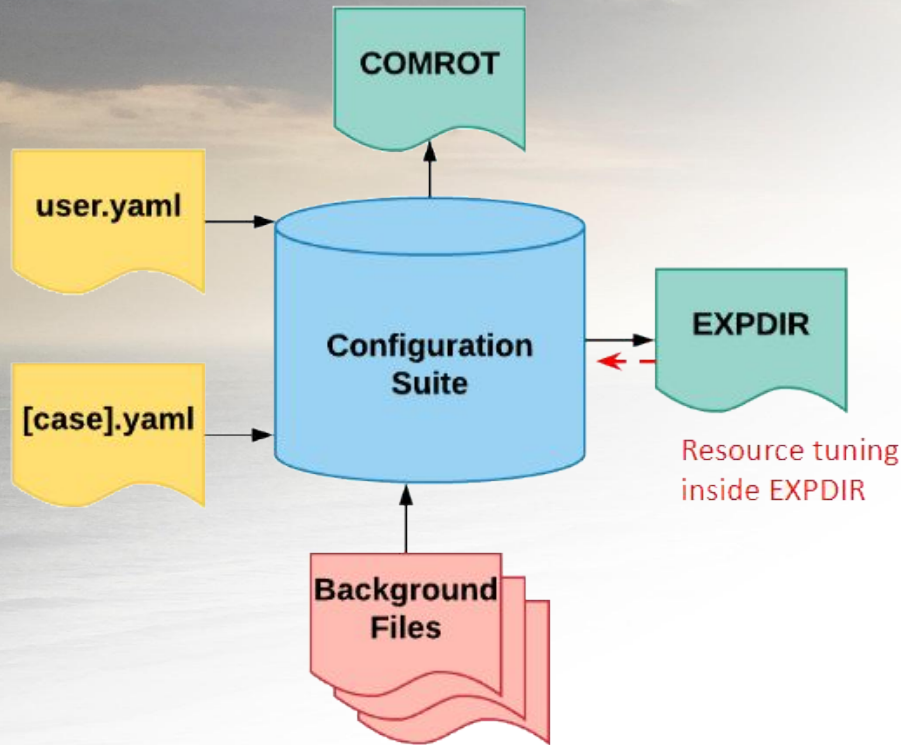
run_gsi: No

chgres_and_convert_ics: Yes

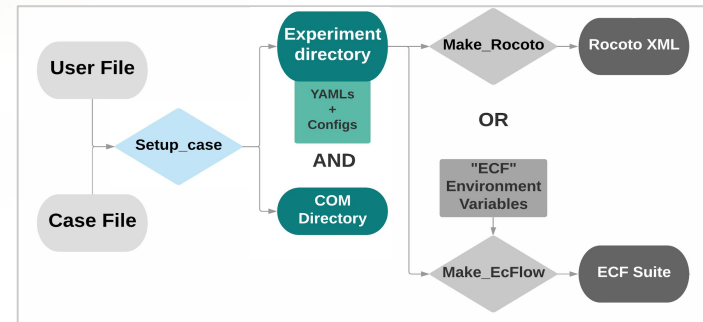
gfs_cyc: 4

Complete list of configurable variables can be found in the CROW documentation in GitHub.io page:

<https://noaa-emc.github.io/CROW>

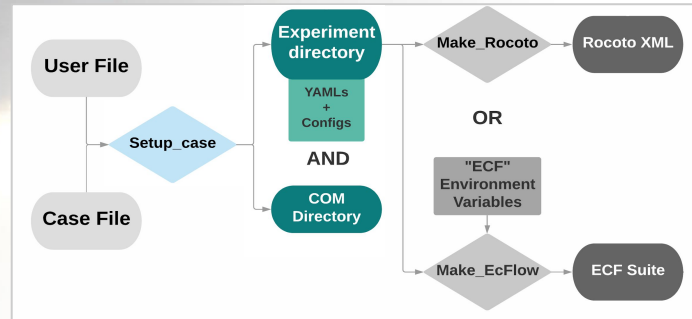


CROW YAML Step 1: Configuration



These steps happen under the hood in the CROW-HAFS implementation. These slides show an approximate implementation.

Step 1: Configuration



Step 1 establishes an experiment directory (EXPDIR) and a rotating output directory (COM a.k.a. ROTDIR) which contain the following:

1. EXPDIR: configuration files for each job in the workflow; these config files are built by parsing the system database based on the case file and populating templated configs with that information. The resulting configs are set for the request experiment design.
2. COM (a.k.a. ROTDIR): this is your rotating directory for model inputs and outputs, this naming convention comes from operations

Depending on your implementation of CROW this step may utilize a setup script called "setup_case.sh". More on this in the next slide.

Step 1: Configuration

USAGE: `setup_case.sh [options] $CASE_NAME $EXPERIMENT_NAME`

CASE_NAME: a case name from the cases/ directory; can include “.yaml” part

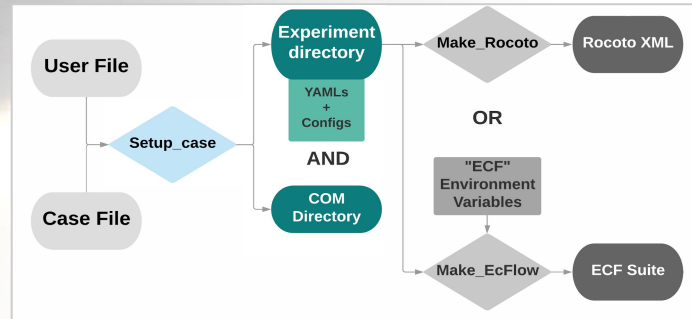
EXPERIMENT_NAME: your name for this execution. This string is used to decide where to put temporary and result files from the simulation. This must be alphanumeric and begin with a letter.

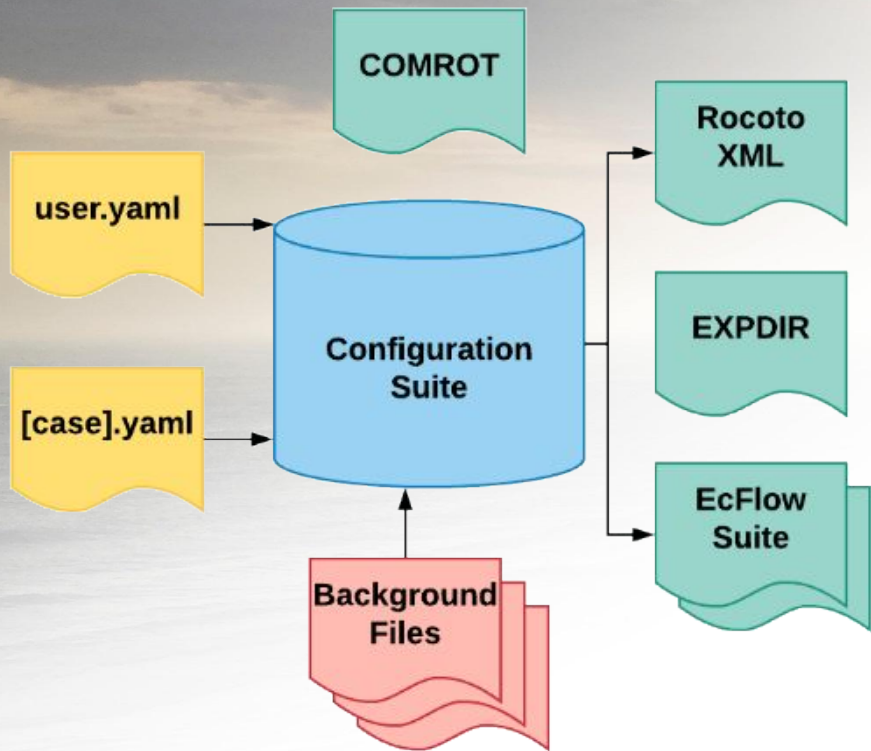
Additional options/flags:

- p \$HPC specify platform name \$HPC, **required if** multiple platforms available
- c skip COM directory creation (recommend to use when making your own initial conditions)
- f/-F force COM directory re-creation and overwrite experimental directory files (-F overwrites platform.yaml)
- v verbose mode
- d / -D debug mode / super debug mode
- m using manual mode
- v -d -D, **more and more screen output during workflow generation. -D will slow down the process quite significantly. Only recommended for developers.**
- s sandbox mode.

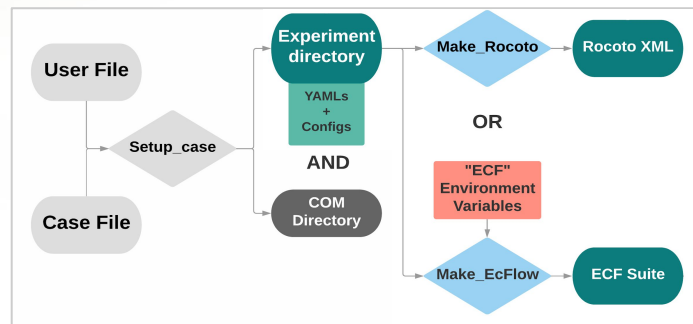
Enables workflow generation without supercomputer access. This option is developed for pure debugging purpose for CROW and workflow developers. When activated, CROW will skip platform validation. Ptmp, stmp and expdir will all be defined under PROJECT_DIR. User need to make sure writing access is granted for PROJECT_DIR.

Example with multiple flags together: `./setup_case.sh -fc free_forecast_case test`



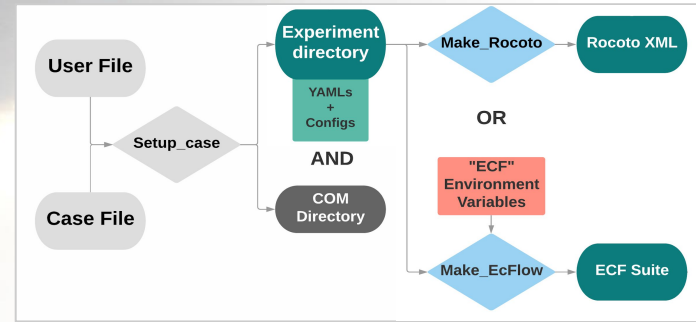


CROW YAML Step 2: Generation



These steps happen under the hood in the CROW-HAFS implementation. These slides show an approximate implementation.

Step 2: Generation



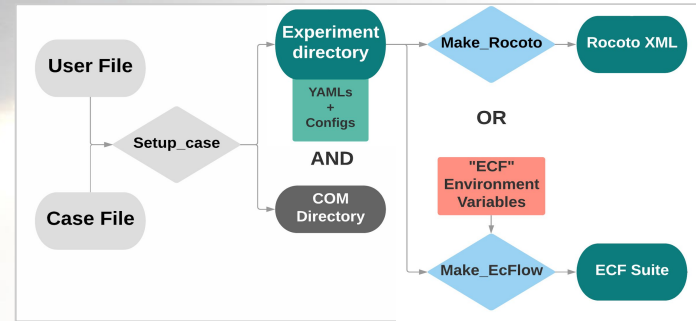
Step 2 generates the workflow manager files for your experiment. The files produced in step 1 are parsed to create the needed input files for the workflow manager you are utilizing. Depending on your implementation there will be separate scripts for each workflow manager type. More on those in the next slide.

- ROCOTO - an xml file for interacting with the scheduler and a sample crontab file for setting up unattended forward progress with cron
- ecFlow - suite definition files for interacting with an ecFlow server which interacts with the scheduler

Additional workflow managers can be supported by CROW in the future.

Step 2: Generation

> `make_rocoto_xml_for.sh $EXPERIMENT_DIRECTORY`



This will write a Rocoto xml file and a crontab file for recurring job. The usage and outcome is almost identical with the “`setup_workflow.py`” script of the legacy configuration system. However, instead of reading the config files, this file gets the configuration information totally by reading YAML configuration files under `$EXPERIMENT_DIRECTORY`. So, modifying the config files in the experiment directory won’t affect the outcome. Experiment directory config files are used by workflow jobs at run-time.

> `make_ecflow_files_for.sh $EXPERIMENT_DIRECTORY`

Note: When running with ecFlow, the four “ECF” environment variables (`$ECF_HOME`, `$ECF_ROOT`, `$ECF_PORT` and `$ECF_HOST`) need to be properly set in your environment. (More details see [ecFlow Training](#))

CROW YAML Reader

**CROW Config
YAML**

**CROW YAML Config
parser**

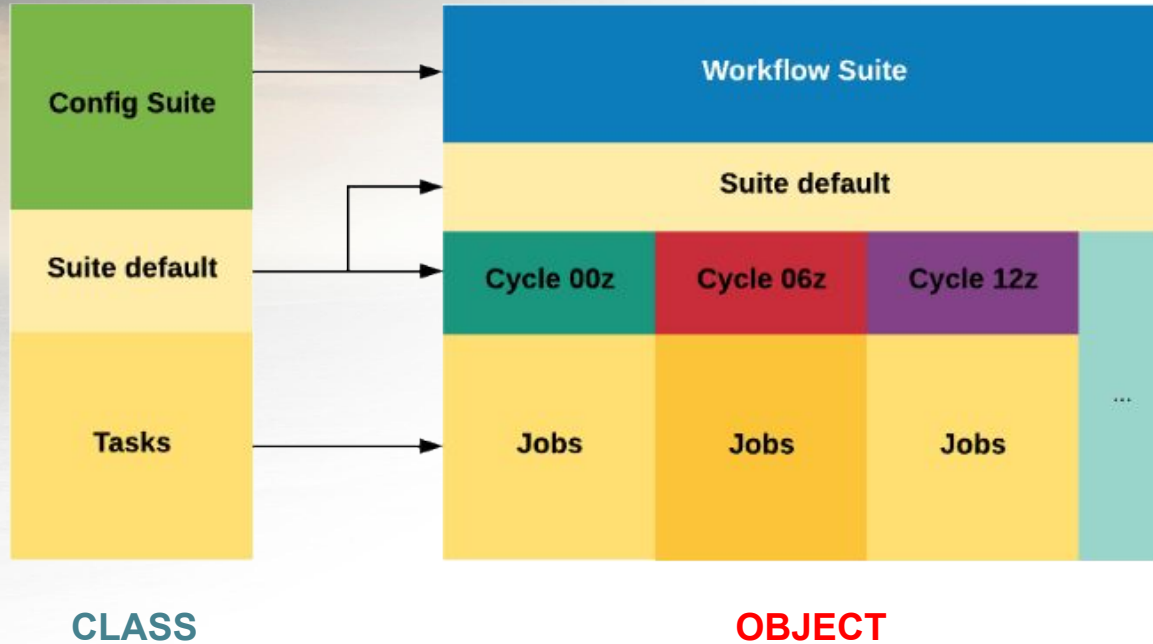
Python data structure

```
my_crow_yaml:  
  five: 5  
  six: 6  
  thirty: !calc five*six  
  sixty: !calc thirty*2  
  true: !calc thirty<sixty  
  tomorrow: !timedelta +24:00:00  
  htxt: hello  
  wtxt: world  
  hello_world: !expand "{htxt}  
  {wtxt}"
```

**crow.
config**

```
my_crow_yaml=eval_dict({  
  "five": 5,  
  "six": 6,  
  "thirty": strcalc("five*six"),  
  "sixty": strcalc("thirty*2"),  
  "true": strcalc("thirty<sixty"),  
  "tomorrow": TimedeltaMaker("+24:00:00"),  
  "htxt": "hello",  
  "wtxt": "world",  
  "hello_world": strexand("{htxt} {wtxt}")  
})
```

Workflow “task” and “job”



Development of CROW - How/Where

CROW is open for development. Users can open issues and create pull requests here:

- Publicly available on GitHub:

<https://github.com/NOAA-EMC/CROW>

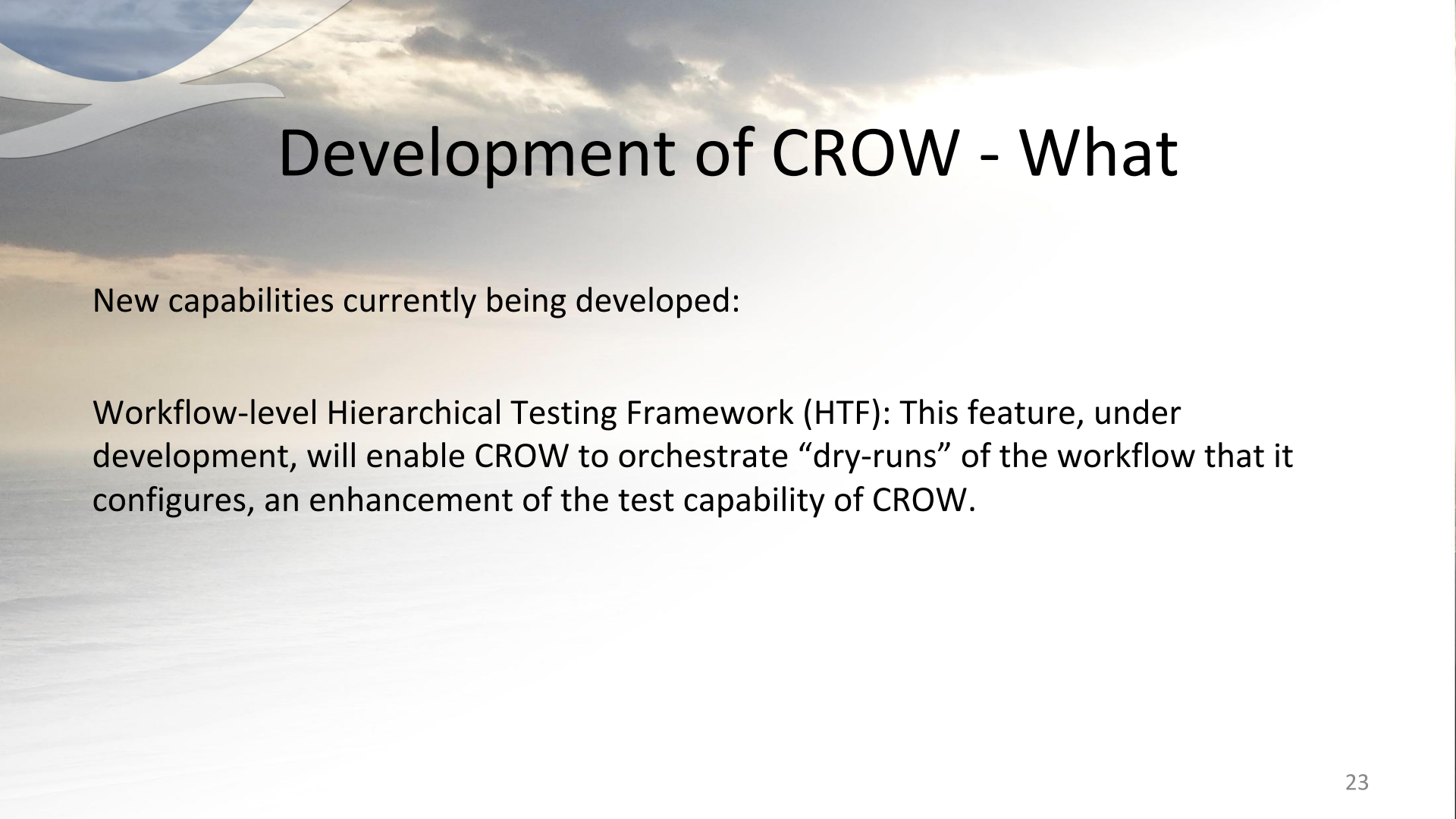
- Documentation being built:

<https://noaa-emc.github.io/CROW/>

Code managers and points of contact:

Jian Kuang (jian.kuang@noaa.gov)

Kate Friedman (kate.friedman@noaa.gov)



Development of CROW - What

New capabilities currently being developed:

Workflow-level Hierarchical Testing Framework (HTF): This feature, under development, will enable CROW to orchestrate “dry-runs” of the workflow that it configures, an enhancement of the test capability of CROW.

CROW Implementations

The following are ongoing implementations of CROW at various stages:

global-workflow/FV3GFS: Introducing alongside current configuration system.

Ocean-DA: Data assimilation for global ocean simulation.

MOM6-CICE Coupling: Significant upgrade of the workflow and configuration systems need for upcoming operational coupled forecast system.

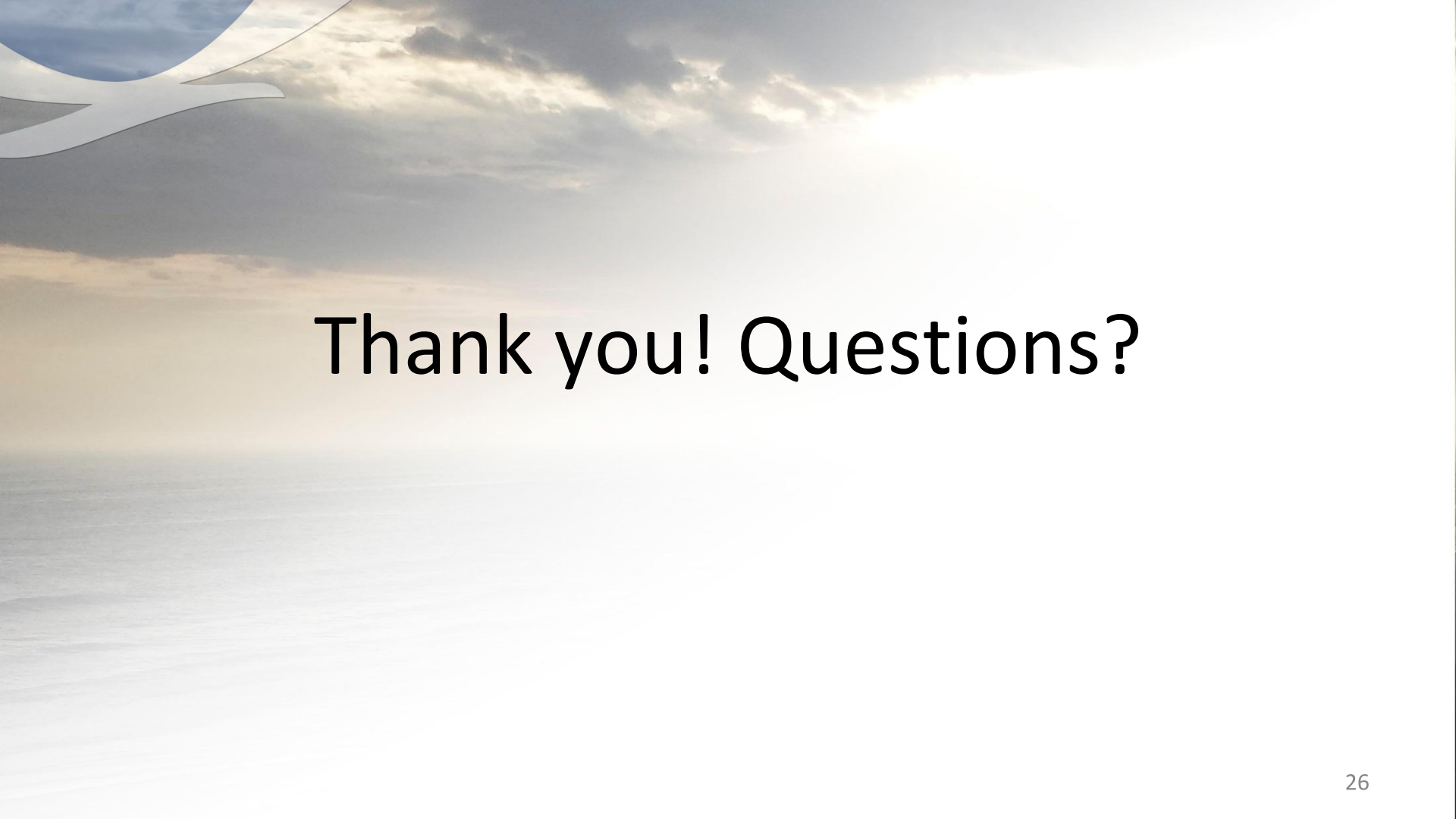
HAFS: Next generation hurricane forecast system for NCEP.



Community Support

Beyond collaborative efforts through the GitHub repository this is still mostly TBD.

Need community input on this!



Thank you! Questions?