OWP | OFFICE OF WATER PREDICTION

Lynker

NOAA

BMI Basics for NextGen

Keith Jennings (Lynker, NOAA Affiliate)

I’m not a software engineer, but I play one on TV

The Community Surface Dynamics Modeling Systems (CSDMS) group at CU Boulder develops the Basic Model Interface (BMI)

- https://csdms.colorado.edu/wiki/BMI
- https://github.com/csdms/bmi

CSDMS
community surface dynamics modeling system

Main references

- Peckham, S.D., Hutton, E.W., and Norris, B., 2013. A component-based approach to integrated modeling in the geosciences: The design of CSDMS. Computers & Geosciences, 53, pp.3-12, http://dx.doi.org/10.1016/j.cageo.2012.04.002.
- Hutton, E.W., Piper, M.D., and Tucker, G.E., 2020. The Basic Model Interface 2.0: A standard interface for coupling numerical models in the geosciences. Journal of Open Source Software, 5(51), 2317, https://doi.org/10.21105/joss.02317.


YOU'RE GOING TO LOVE BMI
I GUARANTEE IT
imgflip.com
FROM: YOUTUBE.COM/USER/MENSWEARHOUSE

TOPICS

1. Why BMI
2. What is BMI
3. BMI functions
4. BMI demonstration
5. Implementing BMI in your own model
6. BMI in NextGen

1. Why BMI?

```
type, abstract :: bmi
  contains

    ! Initialize, run, finalize (IRF)
    procedure(bmif_initialize), deferred :: initialize
    procedure(bmif_update), deferred :: update
    procedure(bmif_update_until), deferred :: update_until
    procedure(bmif_finalize), deferred :: finalize

    ! Exchange items
    procedure(bmif_get_component_name), deferred :: get_component_name
    procedure(bmif_get_input_item_count), deferred :: get_input_item_count
    procedure(bmif_get_output_item_count), deferred :: get_output_item_count
    procedure(bmif_get_input_var_names), deferred :: get_input_var_names
    procedure(bmif_get_output_var_names), deferred :: get_output_var_names

    ! Variable information
    procedure(bmif_get_var_grid), deferred :: get_var_grid
    procedure(bmif_get_var_type), deferred :: get_var_type
    procedure(bmif_get_var_units), deferred :: get_var_units
    procedure(bmif_get_var_itemsize), deferred :: get_var_itemsize
    procedure(bmif_get_var_nbytes), deferred :: get_var_nbytes
    procedure(bmif_get_var_location), deferred :: get_var_location
```

Spoiler alert

Every new model is a (sometimes painful) time-intensive learning experience

BMI makes it less so

A model without BMI

update yourself for one timestep

what's your name

what input data do you need

give me a variable value

what are your output vars

change this variable's value

what's your time step

A model with BMI

what's your name
i'm the SuperGoodHydro model

update yourself for one timestep
sure thing

what input data do you need
air temperature
precipitation

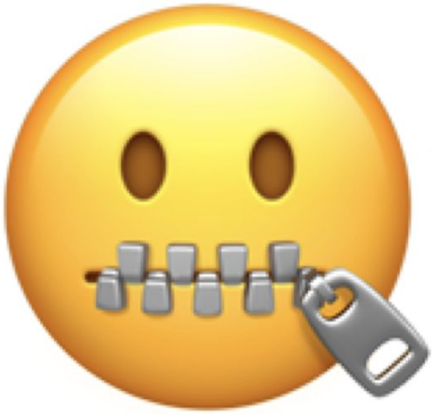
give me a variable value
q_surface = 5.3 mm h^{-1}

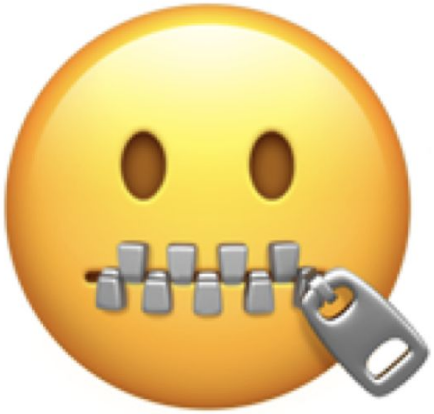
change this variable's value
bucket_state = 27.3 mm

what's your time step
3600 s

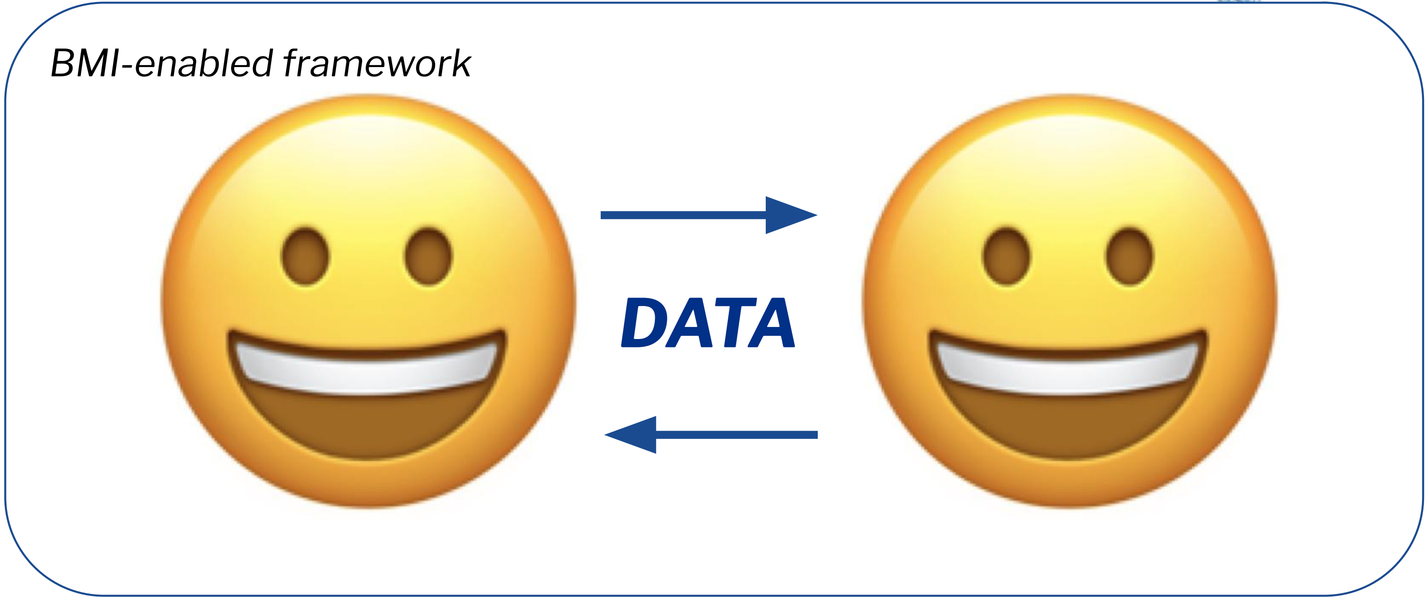
what are your output vars
streamflow
evapotranspiration

Multiple models without BMI





Multiple models without BMI



It’s all about standardization

- BMI works because the functions are the same across models and languages
- BMI removes (or masks) model idiosyncrasies
- BMI underpins the Next Generation Water Prediction Capability
    - We use it in the NextGen framework and in the various models and modules
    - Lets us control and couple independently developed models from different programming languages

2. What is BMI?

```
type, abstract :: bmi
  contains

    ! Initialize, run, finalize (IRF)
    procedure(bmif_initialize), deferred :: initialize
    procedure(bmif_update), deferred :: update
    procedure(bmif_update_until), deferred :: update_until
    procedure(bmif_finalize), deferred :: finalize

    ! Exchange items
    procedure(bmif_get_component_name), deferred :: get_component_name
    procedure(bmif_get_input_item_count), deferred :: get_input_item_count
    procedure(bmif_get_output_item_count), deferred :: get_output_item_count
    procedure(bmif_get_input_var_names), deferred :: get_input_var_names
    procedure(bmif_get_output_var_names), deferred :: get_output_var_names

    ! Variable information
    procedure(bmif_get_var_grid), deferred :: get_var_grid
    procedure(bmif_get_var_type), deferred :: get_var_type
    procedure(bmif_get_var_units), deferred :: get_var_units
    procedure(bmif_get_var_itemsize), deferred :: get_var_itemsize
    procedure(bmif_get_var_nbytes), deferred :: get_var_nbytes
    procedure(bmif_get_var_location), deferred :: get_var_location
```

BMI is middleware…

*"We believe that numerical models, and the sub-components that make up those models, should offer…standardization. To this end, the Community Surface Dynamics Modeling System (CSDMS) has developed the Basic Model Interface (BMI): a set of **standard control and query functions** that, when added to a model code, make that model both easier to learn and easier to couple with other software elements."*

CSDMS BMI Wiki

BMI does…

- Control model runtime using standardized functions
- Pass data in/out of models using standardized functions
- Provide model and variable information
- Work across multiple languages and models

BMI does not…

- Affect hydrologic model code
- Perform unit conversions
- Perform spatial transforms
- Provide information it’s not asked for
- Optimize model output or runtime performance

3. BMI functions

```
type, abstract :: bmi
  contains

    ! Initialize, run, finalize (IRF)
    procedure(bmif_initialize), deferred :: initialize
    procedure(bmif_update), deferred :: update
    procedure(bmif_update_until), deferred :: update_until
    procedure(bmif_finalize), deferred :: finalize

    ! Exchange items
    procedure(bmif_get_component_name), deferred :: get_component_name
    procedure(bmif_get_input_item_count), deferred :: get_input_item_count
    procedure(bmif_get_output_item_count), deferred :: get_output_item_count
    procedure(bmif_get_input_var_names), deferred :: get_input_var_names
    procedure(bmif_get_output_var_names), deferred :: get_output_var_names

    ! Variable information
    procedure(bmif_get_var_grid), deferred :: get_var_grid
    procedure(bmif_get_var_type), deferred :: get_var_type
    procedure(bmif_get_var_units), deferred :: get_var_units
    procedure(bmif_get_var_itemsize), deferred :: get_var_itemsize
    procedure(bmif_get_var_nbytes), deferred :: get_var_nbytes
    procedure(bmif_get_var_location), deferred :: get_var_location
```

CSDMS defines six categories of functions

1. Model information functions
2. Variable information functions
3. Time functions
4. Model control functions
5. Variable getter and setter functions
6. Model grid functions

Model information functions provide important details

```
get_component_name()     => model name
get_input_item_count()   => # of input vars
get_input_var_names()    => names* of input vars
get_output_item_count()  => # of output vars
get_output_var_names()   => names* of output vars
```

*Can use CSDMS standard names for variables

A quick minute on variable names

CSDMS has put together a names standard:

- https://csdms.colorado.edu/wiki/CSDMS_Standard_Names
- Each name comprises an *object* part and a *quantity* part
- Example:

$$\underbrace{\text{atmosphere\_water}}_{\textit{object}}\text{\_\_}\underbrace{\text{precipitation\_leq-volume\_flux}}_{\textit{quantity}}$$

- BUT, you don't have to use the standard names in your model code (more to come later on mapping)
- You can use the standard name in BMI for easier communication when coupling or querying

Variable info functions provide *more* important details

```
get_var_type()     => data type of var
get_var_units()    => standard units* for var
get_var_itemsize() => bytes of one var element
get_var_nbytes()   => bytes of all var elements
get_var_grid()     => integer grid value
get_var_location() => loc of var on grid
```

*Var units should be UDUNITS-compliant text or abbreviations

A quick minute on variable units

CSDMS recommends using UDUNITS-compatible unit names or abbreviations

- https://www.unidata.ucar.edu/software/udunits/
- For both state and flux vars as well as time
- Examples:
  - ‘meters’, ‘kg m-2’, ‘mm h-1’, ‘hours’, etc.
- This lets central frameworks query model info and perform unit conversion (if coded that way)

Time functions provide time step and extent info

```
get_time_step()     => numeric time step
get_time_units()    => std. units* of model time
get_current_time()=> current model time
get_start_time()    => model start time
get_end_time()      => model end time
```

*Time units should be UDUNITS-compliant text or abbreviations

Model control functions standardize execution

`initialize()` `=> start, read configuration file, allocate memory, initialize values`

`update()` `=> run one model time step`

`update_until()` `=> run until a given time`

`finalize()` `=> stop model, close files, deallocate memory`

Get and set values with getters and setters

get_value()   => get var value

get_value_ptr()=> get reference to var value
set_value()   => set var value
get_value_at_indices()=> get var value at given location(s)

set_value_at_indices()=> set var value at given location(s)

Get spatial discretization info with grid functions

get_grid_type() => model spatial discretization (scalar, vector, uniform_rectilinear, etc.)

get_grid_size() => # of spatial elements

get_grid_shape() => array of grid dimensions

get_grid_spacing() => cell size

Plus other BMI functions to specify grid parameters

4. BMI demonstration

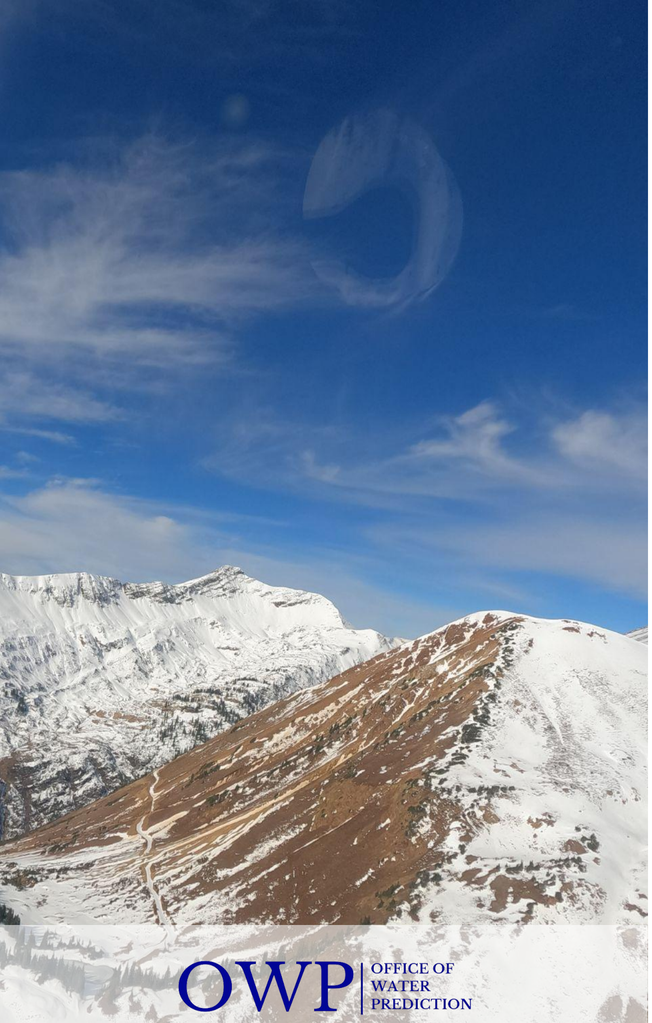
```
type, abstract :: bmi
  contains

    ! Initialize, run, finalize (IRF)
    procedure(bmif_initialize), deferred :: initialize
    procedure(bmif_update), deferred :: update
    procedure(bmif_update_until), deferred :: update_until
    procedure(bmif_finalize), deferred :: finalize

    ! Exchange items
    procedure(bmif_get_component_name), deferred :: get_component_name
    procedure(bmif_get_input_item_count), deferred :: get_input_item_count
    procedure(bmif_get_output_item_count), deferred :: get_output_item_count
    procedure(bmif_get_input_var_names), deferred :: get_input_var_names
    procedure(bmif_get_output_var_names), deferred :: get_output_var_names

    ! Variable information
    procedure(bmif_get_var_grid), deferred :: get_var_grid
    procedure(bmif_get_var_type), deferred :: get_var_type
    procedure(bmif_get_var_units), deferred :: get_var_units
    procedure(bmif_get_var_itemsize), deferred :: get_var_itemsize
    procedure(bmif_get_var_nbytes), deferred :: get_var_nbytes
    procedure(bmif_get_var_location), deferred :: get_var_location
```

Let’s go to the Jupyter notebook



Colorado SPLASH Soil Moisture Survey, October 2021

OWP | OFFICE OF WATER PREDICTION

https://github.com/SnowHydrology/ciroh_workshop_2023/blob/main/examples/ciroh_dev_workshop.ipynb

5. Implementing BMI in your own model

```fortran
type, abstract :: bmi
  contains

    ! Initialize, run, finalize (IRF)
    procedure(bmif_initialize), deferred :: initialize
    procedure(bmif_update), deferred :: update
    procedure(bmif_update_until), deferred :: update_until
    procedure(bmif_finalize), deferred :: finalize

    ! Exchange items
    procedure(bmif_get_component_name), deferred :: get_component_name
    procedure(bmif_get_input_item_count), deferred :: get_input_item_count
    procedure(bmif_get_output_item_count), deferred :: get_output_item_count
    procedure(bmif_get_input_var_names), deferred :: get_input_var_names
    procedure(bmif_get_output_var_names), deferred :: get_output_var_names

    ! Variable information
    procedure(bmif_get_var_grid), deferred :: get_var_grid
    procedure(bmif_get_var_type), deferred :: get_var_type
    procedure(bmif_get_var_units), deferred :: get_var_units
    procedure(bmif_get_var_itemsize), deferred :: get_var_itemsize
    procedure(bmif_get_var_nbytes), deferred :: get_var_nbytes
    procedure(bmif_get_var_location), deferred :: get_var_location
```

So you've got a model, now what?

The model needs to:

- Be coded in a supported language
  - C, C++, Fortran, Python, Java
- Follow the initialize-update-finalize paradigm
  - Separation of concerns
- Implement time loops in a BMI-compliant way
- Have state variables accessible via BMI functions
- Have all BMI functions implemented
  - Even if they just return BMI_FAILURE

Initialize-update-finalize paradigm

- BMI has three model control functions
    - Meaning: model execution must fit into this paradigm
- Initialize
    - Start the model, read in a configuration file, allocate memory, set initial parameter and state values
- Update
    - Execute one (or more with update_until) time step
- Finalize
    - Close files, deallocate memory
- Model functions go inside one of these BMI functions
    - Otherwise, running the model with BMI won't work (BMI can't access what it doesn't see)
        - There are a few exceptions (keeping the model clock)

Using this paradigm makes code cleaner

- Model update without BMI

```
do itime = 1, ntime

    forcing%P_ML      = forcing%SFCPRS                 ! surf press estimated at model level [Pa], can avg multi-level nwp
    forcing%O2PP      = parameters%O2 * forcing%P_ML          ! atmospheric co2 concentration partial pressure (Pa)
    forcing%CO2PP     = parameters%CO2 * forcing%P_ML         ! atmospheric o2 concentration partial pressure (Pa)

    call UtilitiesMain (itime, domain, forcing, energy)
    call InterceptionMain (domain, levels, options, parameters, forcing, energy, water)
    call EnergyMain (domain, levels, options, parameters, forcing, energy, water)
    call WaterMain (domain, levels, options, parameters, forcing, energy, water)

end do ! time loop
```

- Model update with BMI

```
do while (current_time < end_time)
    status = m%update()                      ! run the model one time step
    status = m%get_current_time(current_time) ! update current_time
end do
```

Separation of concerns

Non-compliant

```
!------------------------------
!   initialize
!------------------------------
call namelist%ReadNamelist()
call options%Init()
call parameters%Init(namelist)
call energy%Init(namelist)
call water%Init(namelist)

! Initializations
! for soil water
water%qinsur     = 0.0
water%snoflow    = 0.0
water%qseva      = 0.0
water%etrani     = 0.0
water%btrani     = 0.0
water%btran      = 0.0
```

Compliant

```
!------------------------------
!   Initialize
!------------------------------
call get_command_argument(1, arg)
status = m%initialize(arg)
```

Inaccessible code from driver moves to initialize() function

A model should update one time step at a time

- Time loops encase the update function
  - Update_until can run multiple update steps

BMI can only find what you give it access to

- Map your variable names to standard names
- Make sure their scope extends through model execution
  - I.e. temporary values within a function/module/subroutine are not accessible via BMI

```
self._values = {"atmosphere_water__precipitation_leq-volume_flux": self._model.ppt_mm,
                "land_surface_air__temperature": self._model.tair_c,
                "snowpack__liquid-equivalent_depth": self._model.swe_mm,
                "snowpack__melt_volume_flux": self._model.melt_mm}
```

Any other tips? Yes

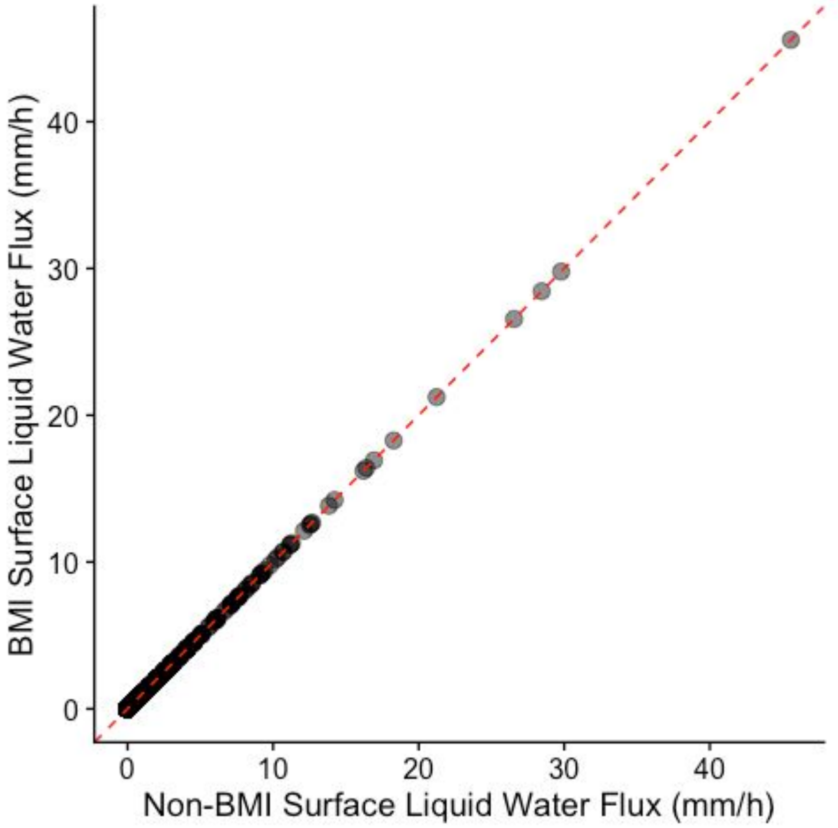
- Implement simplest functions first
  - Model and variable info, then time functions
- Perform targeted refactoring
  - Roll code chunks into larger functions
    - Can be put directly into BMI code or have a BMI function call a new model function
  - Remember separation of concerns (no initializing in update, for example)
- Use GitHub for version control
- Test, test, test
  - And test some more

Unit testing is an important part of BMI implementation

- You want to know your functions work
- And model output hasn't changed with BMI

```
TEST BMI GETTER SETTER FUNCTIONS
********************************
 updating... timesteps in test loop: 1

 current time: 3600.000000
  atmosphere_water__liquid_equivalent_precipitation_rate
   get value: 0.000000
   get value at indices: 0.000000
   get value ptr: 0.000000
   set value: 99.900000
   new get value: 99.900000
   set value at indices: 11.100000
   new get value at indices: 11.100000
```



Modifications for NextGen

- Compiler directives (or options) for forcing and output
- Remove print statements
- Models as shared libraries
  - No program or main
  - Importance of separation of concerns

6. BMI in NextGen

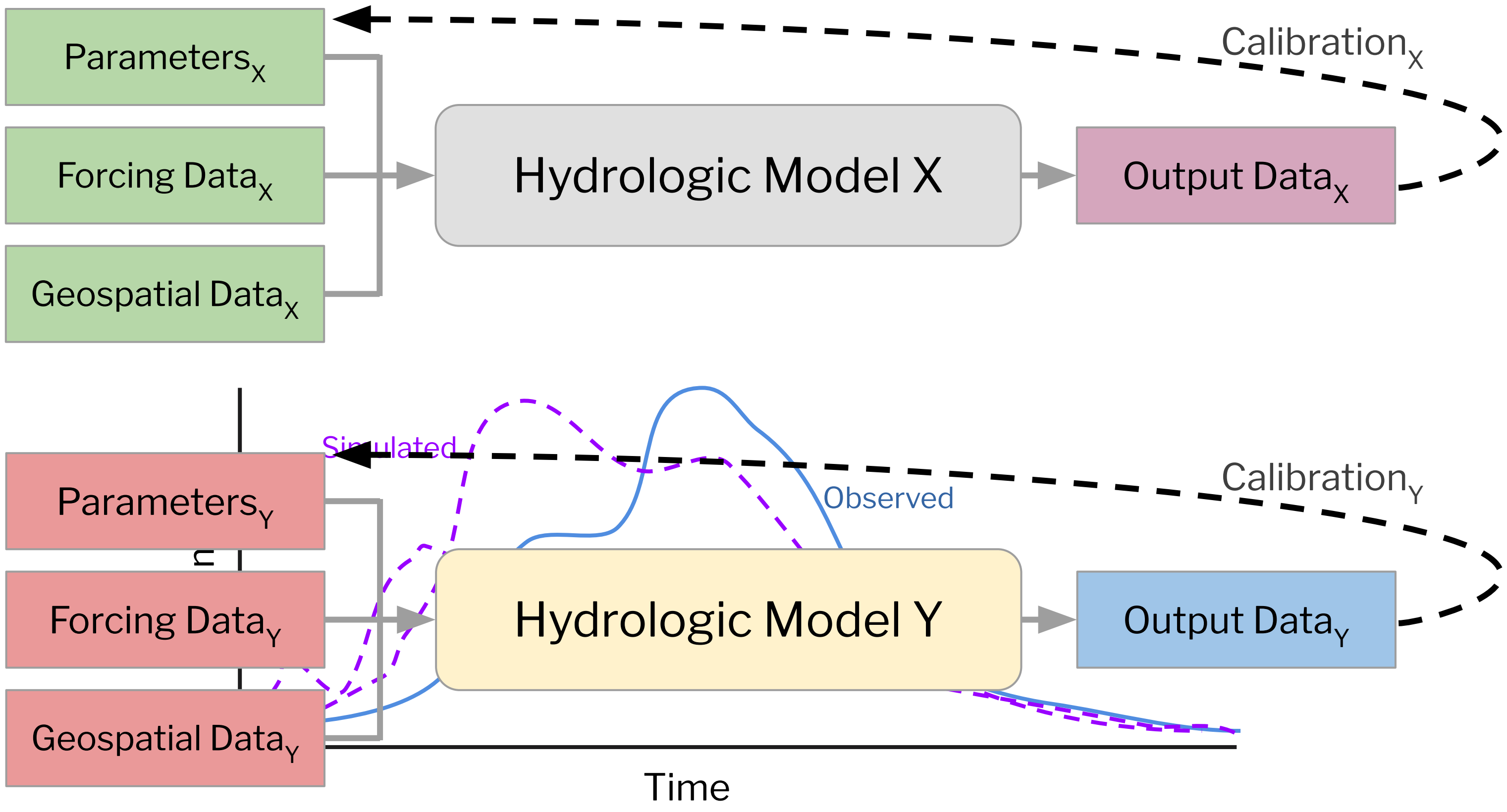
```
type, abstract :: bmi
  contains

    ! Initialize, run, finalize (IRF)
    procedure(bmif_initialize), deferred :: initialize
    procedure(bmif_update), deferred :: update
    procedure(bmif_update_until), deferred :: update_until
    procedure(bmif_finalize), deferred :: finalize

    ! Exchange items
    procedure(bmif_get_component_name), deferred :: get_component_name
    procedure(bmif_get_input_item_count), deferred :: get_input_item_count
    procedure(bmif_get_output_item_count), deferred :: get_output_item_count
    procedure(bmif_get_input_var_names), deferred :: get_input_var_names
    procedure(bmif_get_output_var_names), deferred :: get_output_var_names

    ! Variable information
    procedure(bmif_get_var_grid), deferred :: get_var_grid
    procedure(bmif_get_var_type), deferred :: get_var_type
    procedure(bmif_get_var_units), deferred :: get_var_units
    procedure(bmif_get_var_itemsize), deferred :: get_var_itemsize
    procedure(bmif_get_var_nbytes), deferred :: get_var_nbytes
    procedure(bmif_get_var_location), deferred :: get_var_location
```

Prepping Data and Models Is a Major Time Investment

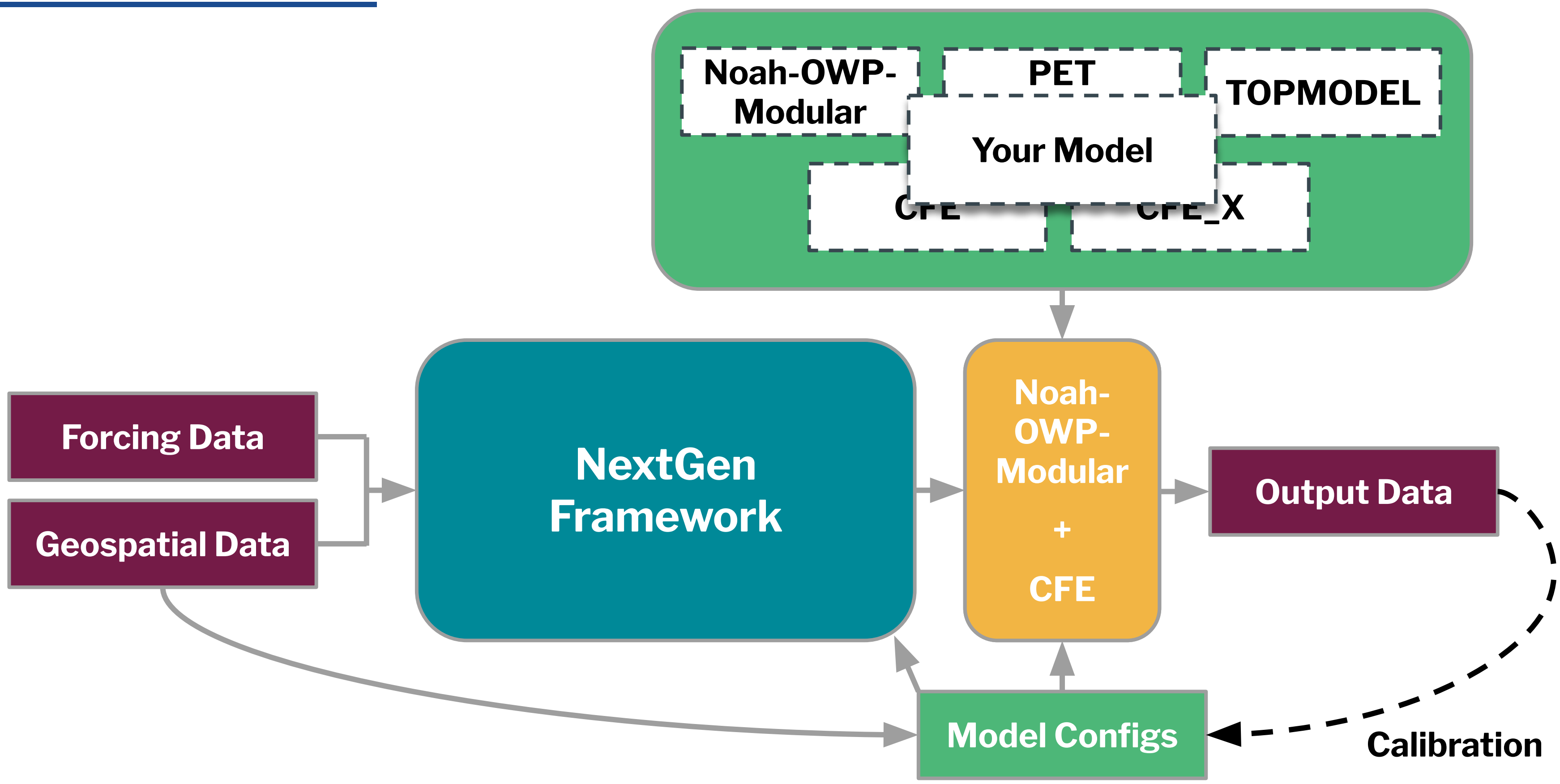


What if there is a better way?

- Acknowledge that there is no best model
  - Run the optimal configuration in a given location
- Reuse forcing data, geospatial data, runtime commands, etc.
- Develop open-source software (no proprietary tools)
- Collaborate openly via GitHub

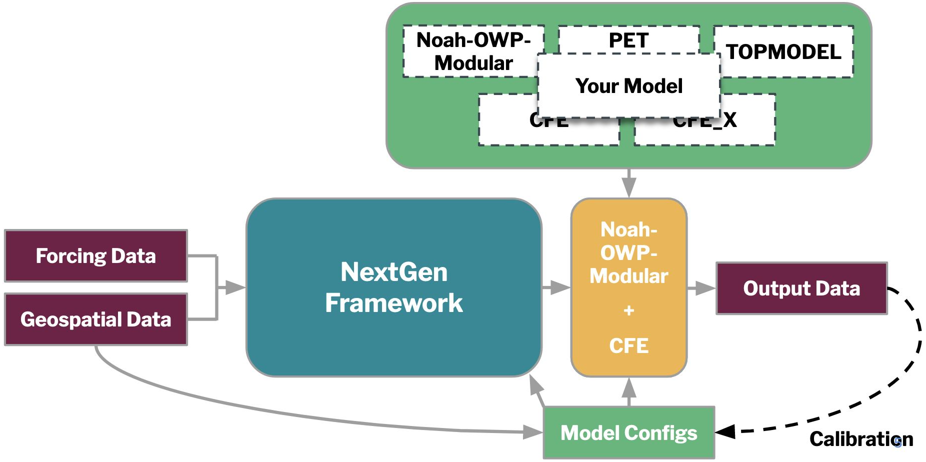
Enter the Next Generation Water Prediction Capability...

Simplify the process

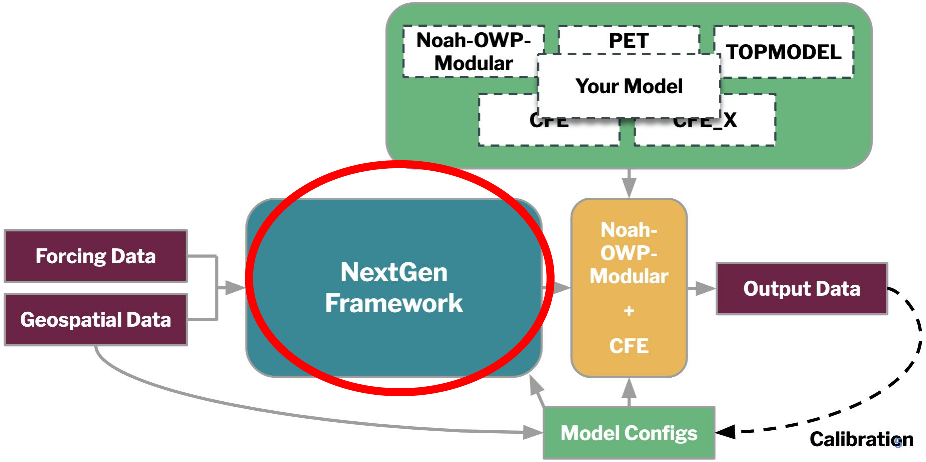


A few advantages of this approach

- Language- and model-agnostic
- Standard forcing and geospatial formats
- Easy model coupling via the Basic Model Interface
- Saves model setup time
- Lets you run different model and module sets (formulations) in a single instance



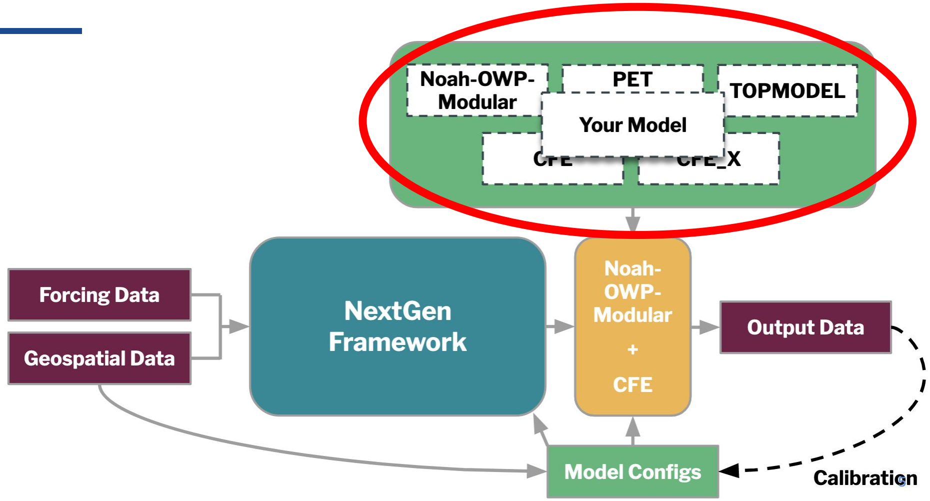
Quick component overview: framework



The **framework** is the heart of NextGen

- Controls model runtime and execution
- Reads input data and passes it to the models
- Couples models via Basic Model Interface functions
- Writes output data from models

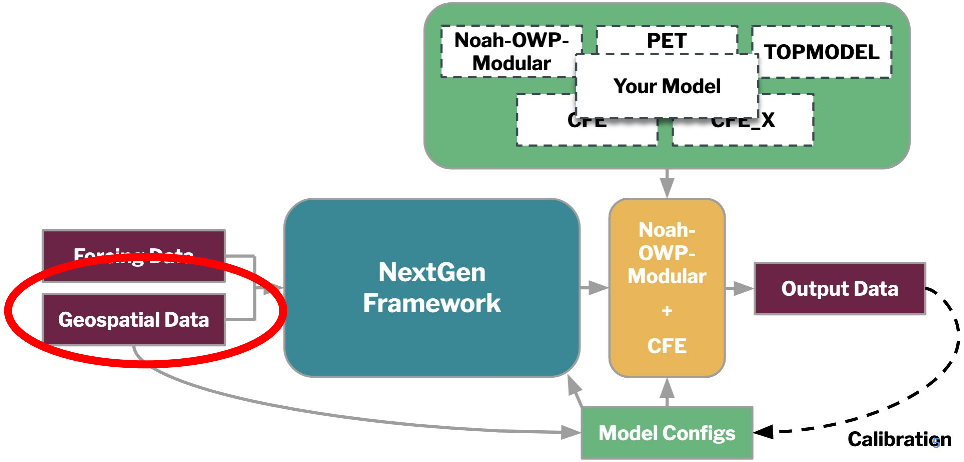
Quick component overview: formulations



Formulations are the coupled models in a given location

- Surface processes (Noah-OWP-Modular, PET, Snow-17)
- Soil processes (CFE, TOPMODEL, Sac-SMA, LGAR)
- Machine learning (LSTM)
- Initialized with config files made from the hydrofabric

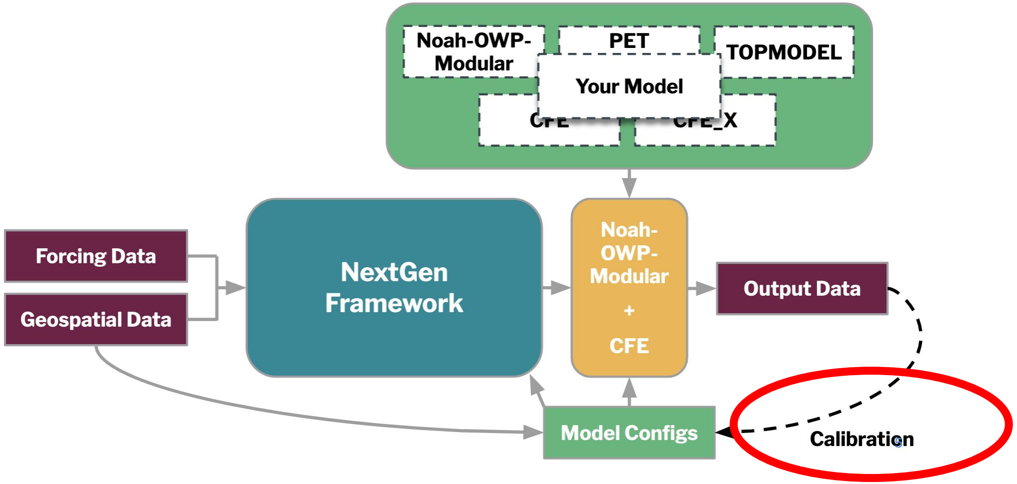
Quick component overview: hydrofabric



The **hydrofabric** defines the model domain, network connectivity, and basin attributes

- Created using fully open-source tools
- Derives data from publicly available catalog https://noaa-owp.github.io/hydrofabric/index.html

Quick component overview: calibration



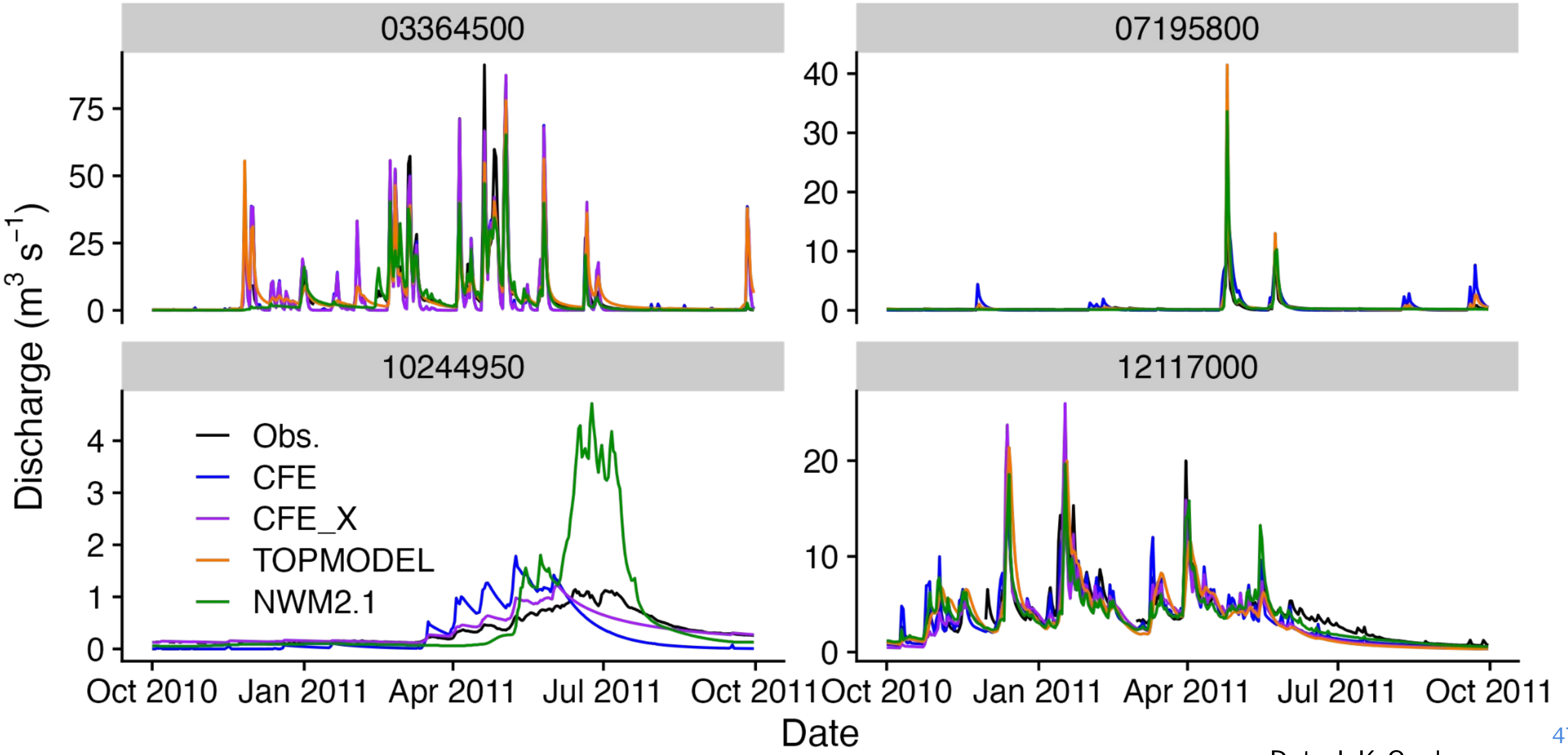
ngen_cal is a model-agnostic calibration package

- User can choose models, domains, time periods, default values, objective functions, and algorithms
- Relies on novel use of BMI functions to set parameters
- Compares output to USGS streamflow observations

BMI is the glue that holds NextGen together

- Framework uses the standardized BMI functions to control and couple models from different languages
  - It initializes, updates, and finalizes
    - It keeps track of model time, too
  - It uses get/set value to couple models and pass data
- Examples:
  - Noah-OWP-Modular coupled to CFE and TOPMODEL

Many traces, one modeling framework



Data: L.K. Cunha

Spatially variable outcomes in performance

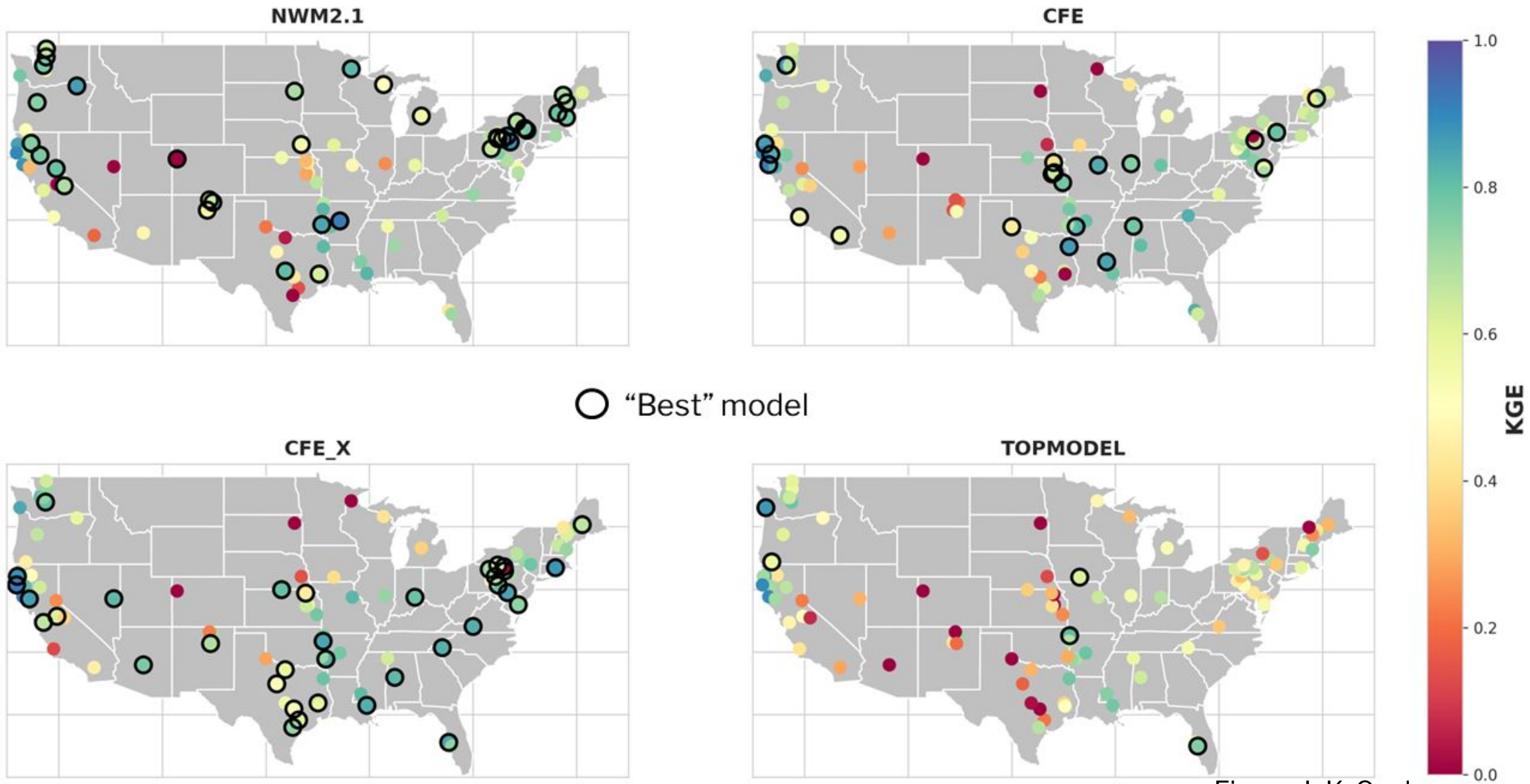


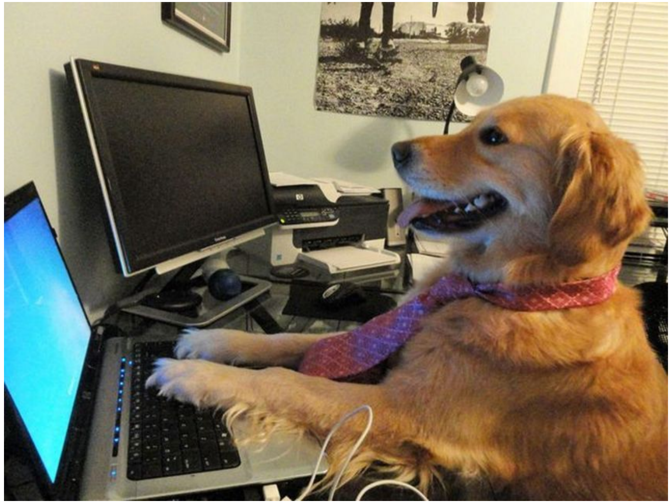
Figure: L.K. Cunha

Where we're going from here

- Updated forcing engine
- Heterogeneous routing
- Coastal modules for total water level forecasts
- More spatial discretizations (e.g. regular grids)
- Automatic config generation
- MaaS deployment
- More refined calibration

BUT THAT'S NOT ALL

- You have a favorite model??
- Implement BMI and start running it in NextGen



Be a NextGen contributor!

- https://github.com/NOAA-OWP …
  - /ngen
  - /ngen-cal
  - /hydrofabric
  - /noah-owp-modular
  - /cfe
  - /topmodel
  - /lstm
  - /evapotranspiration
  - /soilfreezethaw
  - /snow17
  - /sac-sma
  - /t-route

Acknowledgments

Computer Science

- Bobby Bartel, Shengting Cui, Nels Frazier, Donald Johnson, Cham Pham, Austin Raney, Chris Tubbs, Matt Williamson

Calibration

- Xia Feng, Yuqiong Liu

Hydrofabric and Geospatial Science

- Mike Johnson, Rich Gibbs

Hydrologic Science

- Luciana Kindl da Cunha, Grey Evenson, Jessica Garrett, Ahmad Jan, Keith Jennings, Peter La Follette, Rachel McDaniel, Naoki Mizukami, Scott Peckham, Andy Wood, Wanru Wu, Zhengtao Cui

Inland Routing and Coastal Modules

- Jason Ducker, Sean Horvath, DongHa Kim, Julio Zyserman

NOAA-NWS OWP Leadership and Project Management

- Trey Flowers, Fred Ogden, Nick Casiday, Ryan Jones, Brian Cosgrove, Ed Clark, Tom Graziano

Partners

- USACE, USGS, CIROH

Thank You!

Keith Jennings

Lynker

keith.jennings@noaa.gov | kjennings@lynker.com

https://water.noaa.gov

NOAA

OWP | OFFICE OF WATER PREDICTION