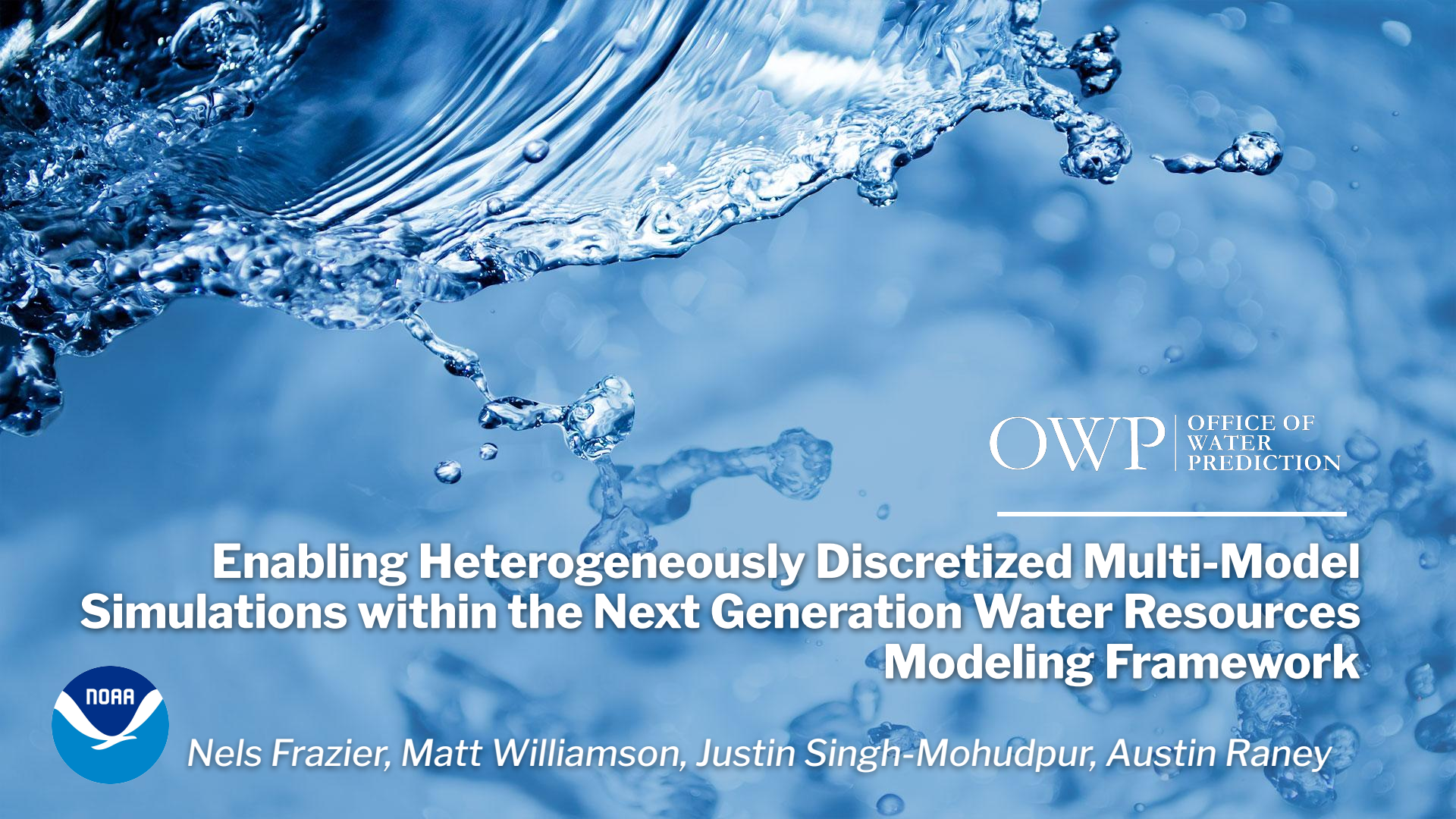


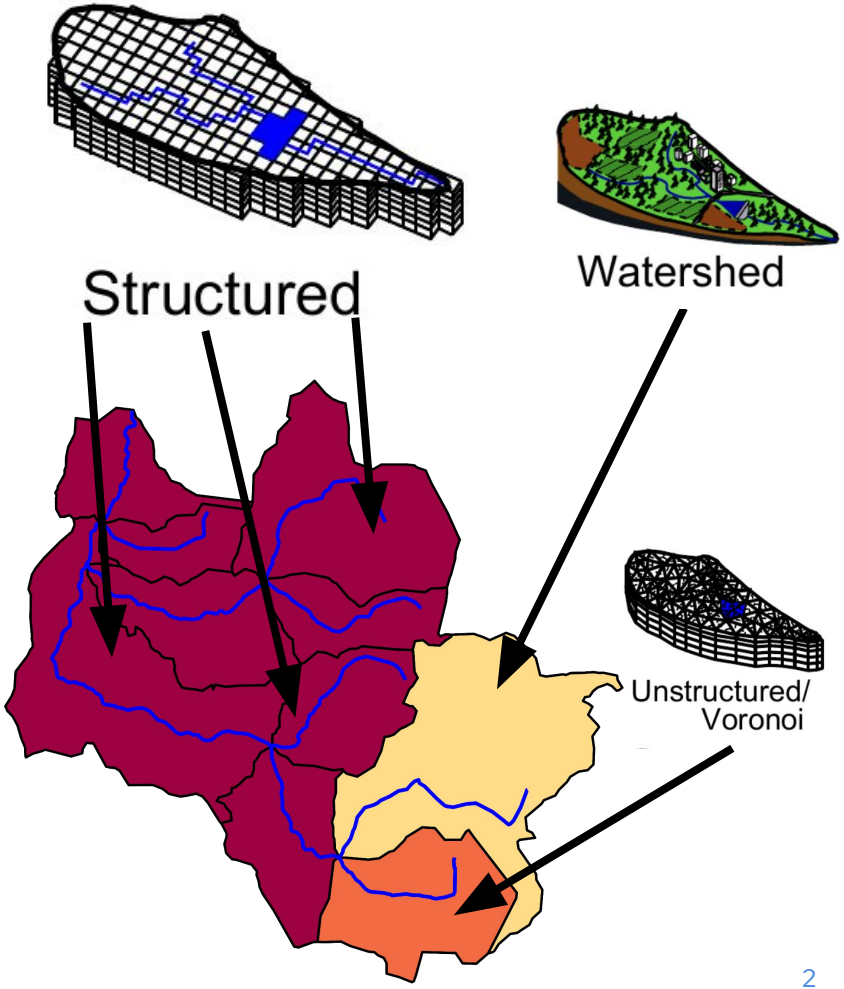
OWP | OFFICE OF WATER PREDICTION

Enabling Heterogeneously Discretized Multi-Model Simulations within the Next Generation Water Resources Modeling Framework

Nels Frazier, Matt Williamson, Justin Singh-Mohudpur, Austin Raney

Background

- NextGen needs to support multiple distrizations in a single model runtime
- Gridded Models in NextGen with BMI
  - NextGen *may* initialize a gridded model for each catchment
  - Currently ~**800,000** catchments!
- We developed a set of BMI 2.0 compliant conventions and protocols to enable **dynamic** creation of gridded domains!
- Developed a software development kit to aid in the development of NextGen Dynamic Grid formulations!



Background

For NextGen to support **heterogeneously discretized, gridded catchment formulations** three things are needed:

Background

For NextGen to support **heterogeneously discretized, gridded catchment formulations** three things are needed:

- **Extend the capabilities of NextGen with BMI**

Background

For NextGen to support **heterogeneously discretized, gridded catchment formulations** three things are needed:

- Extend the capabilities of NextGen with BMI
- **Implement gridded models “abstractly”**

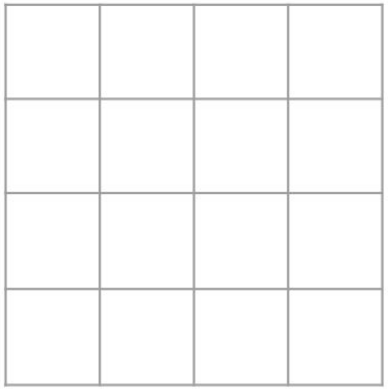
Background

For NextGen to support **heterogeneously discretized, gridded catchment formulations** three things are needed:

- Extend the capabilities of NextGen with BMI
- Implement gridded models “abstractly”
- **Framework assumes data transformation responsibilities**

BMI Grids

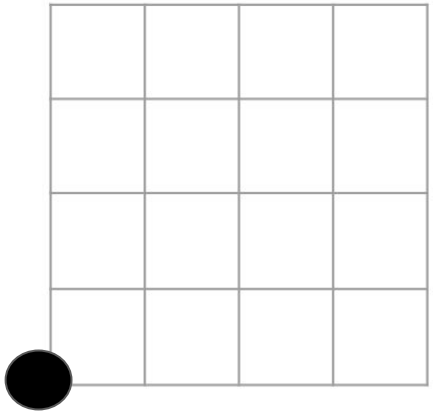
- The BMI supports several grid variants and provides methods for querying a model's domain
- Grid Variants
  - Scalar
  - Points
  - Vector
  - Unstructured
  - Structured quadrilateral
  - **Rectilinear**
  - **Uniform Rectilinear**



BMI Grid Metadata

Regular grids – BMI supports *querying* Origin

- **get_grid_origin()**



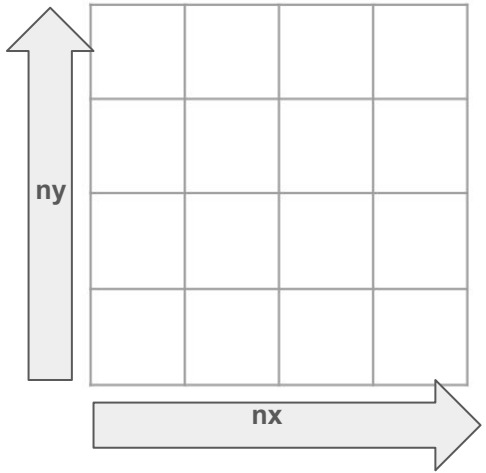
BMI Grid Metadata

Regular grids – BMI supports *querying*
Origin

- **get_grid_origin()**

Shape

- **get_grid_shape()**



BMI Grid Metadata

Regular grids – BMI supports *querying*

Origin

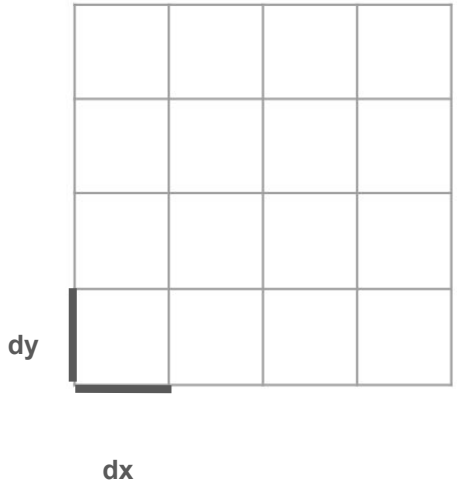
- **get_grid_origin()**

Shape

- **get_grid_shape()**

Spacing

- **get_grid_spacing()**



BMI Grid Metadata

Regular grids – BMI supports *querying*

Origin

- **get_grid_origin()**

Shape

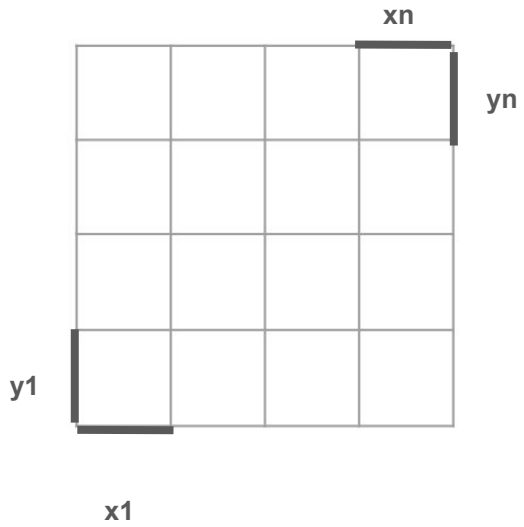
- **get_grid_shape()**

Spacing

- **get_grid_spacing()**

Grid Coordinates

- **get_grid_x/y/z()**



BMI Grid Metadata

Regular grids – BMI supports *querying*

Origin

- **get_grid_origin()**

Shape

- **get_grid_shape()**

Spacing

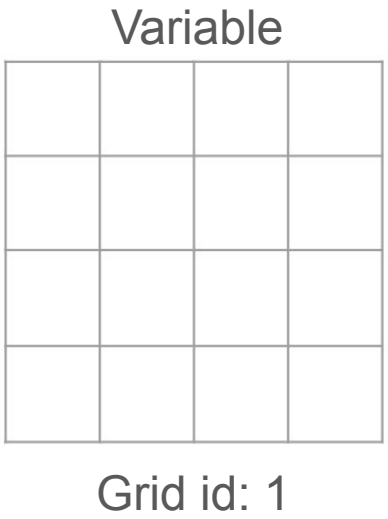
- **get_grid_spacing()**

Grid Coordinates

- **get_grid_x/y/z()**

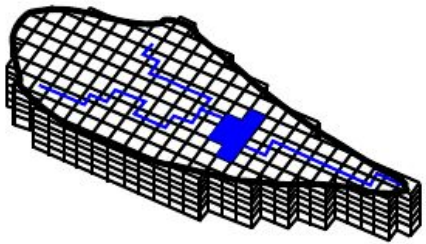
Grid Variables

- **get_var_grid()**



NextGen Dynamic BMI Grid Module Requirements

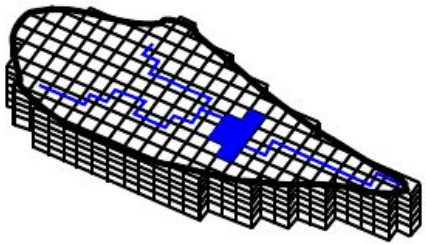
- Minimal Grid Metadata required from BMI modules
  - Grid Identifier **get_var_grid**
  - Grid Rank (dimensionality) **get_grid_rank**
  - Grid Type, e.g. **get_grid_type**
    - Uniform Rectilinear
    - Rectilinear
- These are already BMI 2.0 requirements



BMI Dynamic Grid Conventions for NextGen

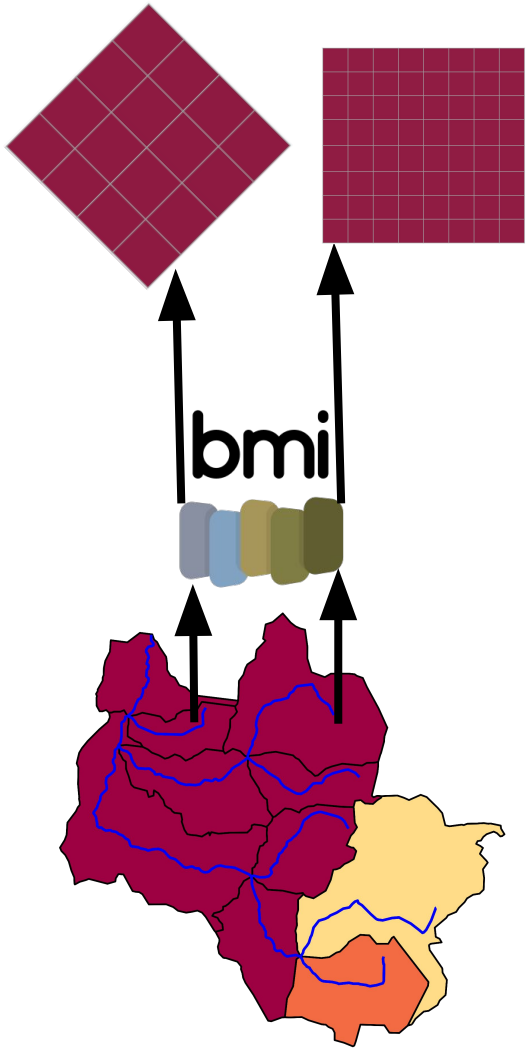
- Modules supporting dynamic grid assignment must recognize the following **grid meta-data *variables***
  - grid_<id>_shape
  - grid_<id>_origin
  - grid_<id>_spacing
  - grid_<id>_units

- This allows us to ***emulate*** functionality such as **set_grid_shape( )** which isn't part of the BMI 2.0 interface!

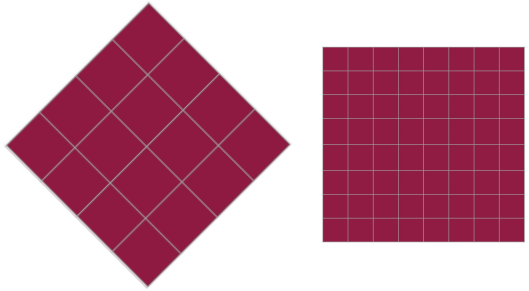


NextGen Dynamic BMI Grid Creation

- Module with rectilinear and/or uniform rectilinear grids
  - initialize( )
  - get_var_grid(variable)
  - get_grid_rank (<id>)
  - set_value( ) **for grid metadata variables**
    - grid_<id>_shape
    - grid_<id>_origin
    - grid_<id>_spacing
    - grid_<id>_units
- set_value( ) **on model input and parameter variables**

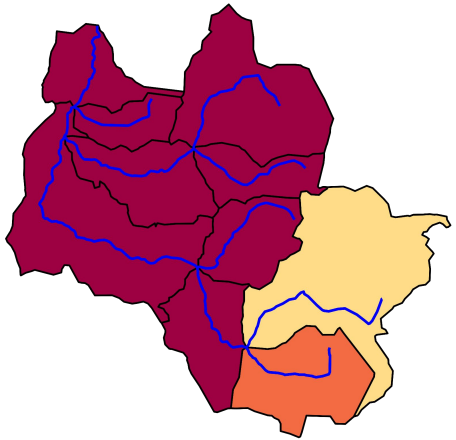


NextGen BMI Software Development Kit



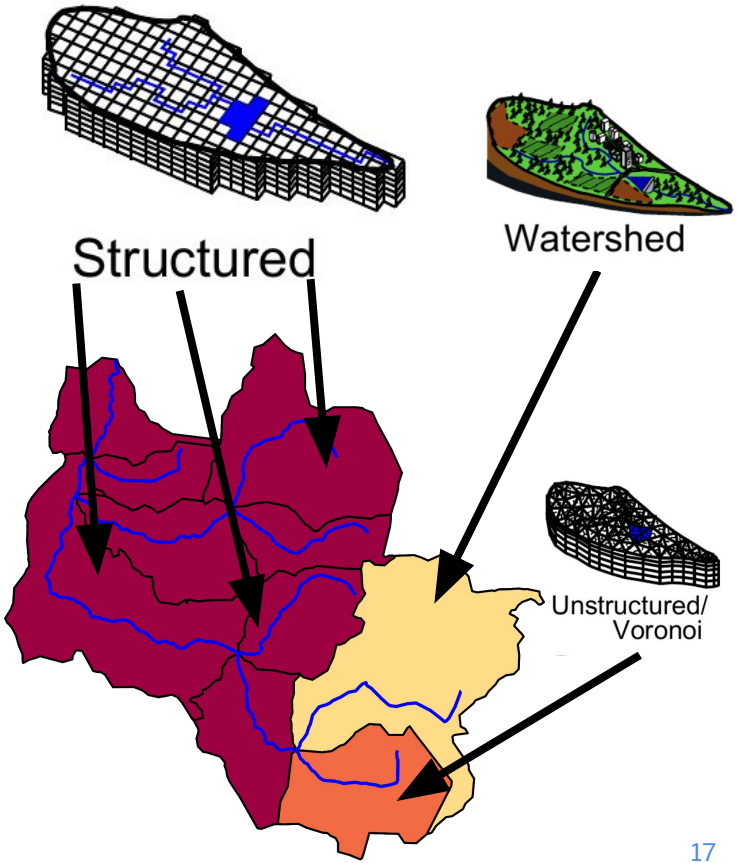
- Python and Fortran library/module developed
  - Manage BMI **grid meta data**
  - Allow **dynamic updates** of grid definition (i.e. shape)
  - 6 regular grid functions become "**boilerplate**"
  - Models can implement grid operations/algorithms using the interface

- BMI now facilitates **dynamic allocation** of model variables
  - set_value ("grid_<id>_shape", ...) now requires
    - **dynamic allocation** of internal model variables
    - May also require **re-initialization**

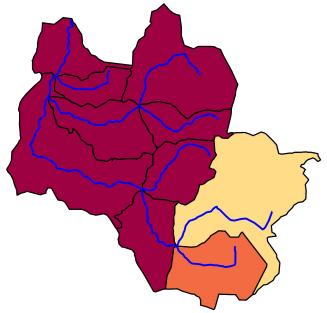


Key Takeaways

- Extend the capabilities of NextGen with BMI
  - **Conventions for grid metadata**
  - **Dynamic BMI grid creation**
- Implement gridded models “abstractly”
  - Framework code manages spatial domains
  - BMI model **development kit**
- Framework assumes data transformation
  - Integration with hydrofabric
  - Regridding
  - Re-projection
  - Aggregation and disaggregation



Future Developments



- Module Developments
  - Grid module incorporated into Noah-OWP-Modular Fortran Model
  - Dynamic reallocation of Noah-OWP-Modular state variables
- Framework Developments
  - Development of NextGen generic spatial features
  - Implementation of dynamic grid BMI protocol with Noah-OWP-Modular
  - Framework regridding capabilities

Thank You!

OWP | OFFICE OF WATER PREDICTION

For More Information:

Nels Frazier

nels.frazier@noaa.gov

https://water.noaa.gov

Abstract – This slide will be removed

Hydrological models use a variety of discretizations (e.g. structured or unstructured grids / lumped catchment). An assortment of transformations may be required to pass information between models with differing discretizations that simulate a common domain. This may include spatial transformations, re-projection, or aggregation / disaggregation. The Next Generation Water Resources Modeling Framework (NextGen) aims to simplify the establishment of gridded model domains by eliminating the requirement of a model application code to understand and handle gridded input/output (I/O) formats, spatial transformations and projections, and other spatially related pre-and-post processing techniques.

In this work, we **propose a set of naming conventions and a protocol that can be implemented using the current version of the Basic Model Interface (BMI 2.0)** which allows 1) model applications to **advertise their supported variable dimensionality**, 2) the **framework to communicate domain origin, extent, and spacing (dy and dx) to model applications dynamically**, and 3) **dynamic allocation of memory required for model grids**. We show how this has been implemented for Python and Fortran BMI models executed within the framework for uniform and rectilinear grids, as well as discuss the software development kit (SDK) built to bring this functionality to new and existing BMI models. By reducing the responsibility of model code using this technique, the framework can assume responsibility for geospatial I/O, formats, and pre-and-post processing of gridded model data, allowing the model to focus on the mathematical representation and physical characterization of the domain.

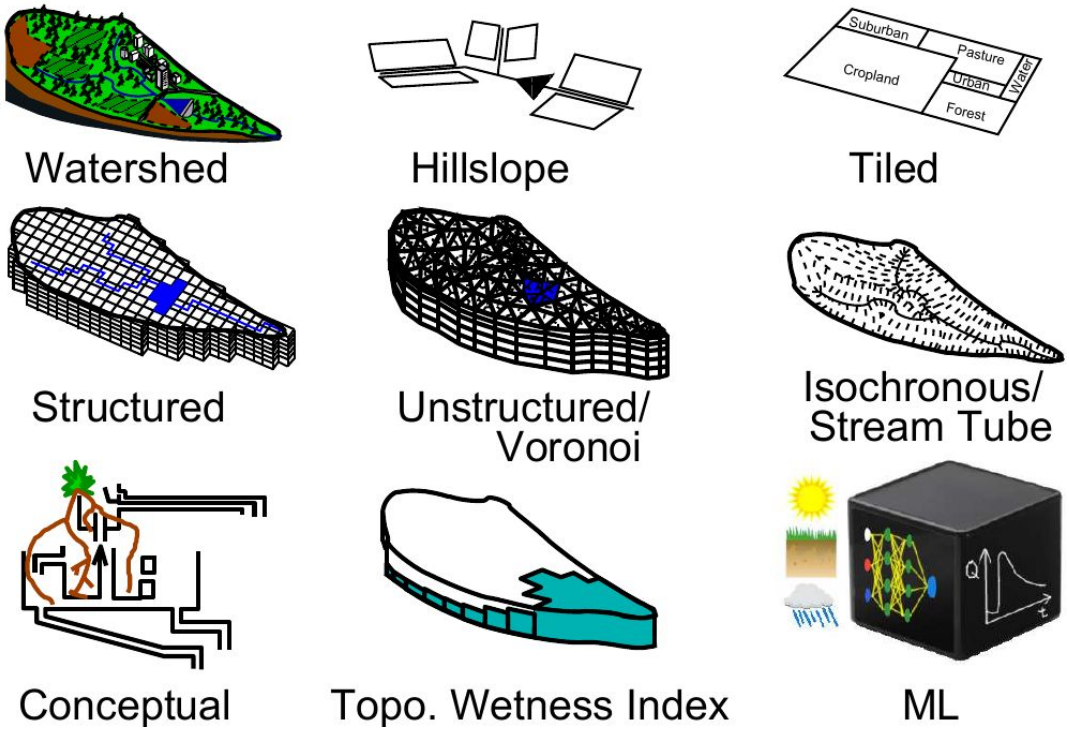
Plain Lang Summary – This slide will be removed

Plain-language Summary

This work proposes a simplifying method for managing gridded model representations in the Next Generation Water Resources Modeling Framework. We show how the framework communicates with models in multiple programming languages, using the Basic Model Interface, to **establish gridded domains managed by the framework**. By **reducing the responsibility of model code** using this technique, the **framework can assume responsibility for geospatial input and out, formats, and pre-and-post processing of gridded model data**, allowing the **model to focus on the mathematical representation** and physical characterization of the domain.

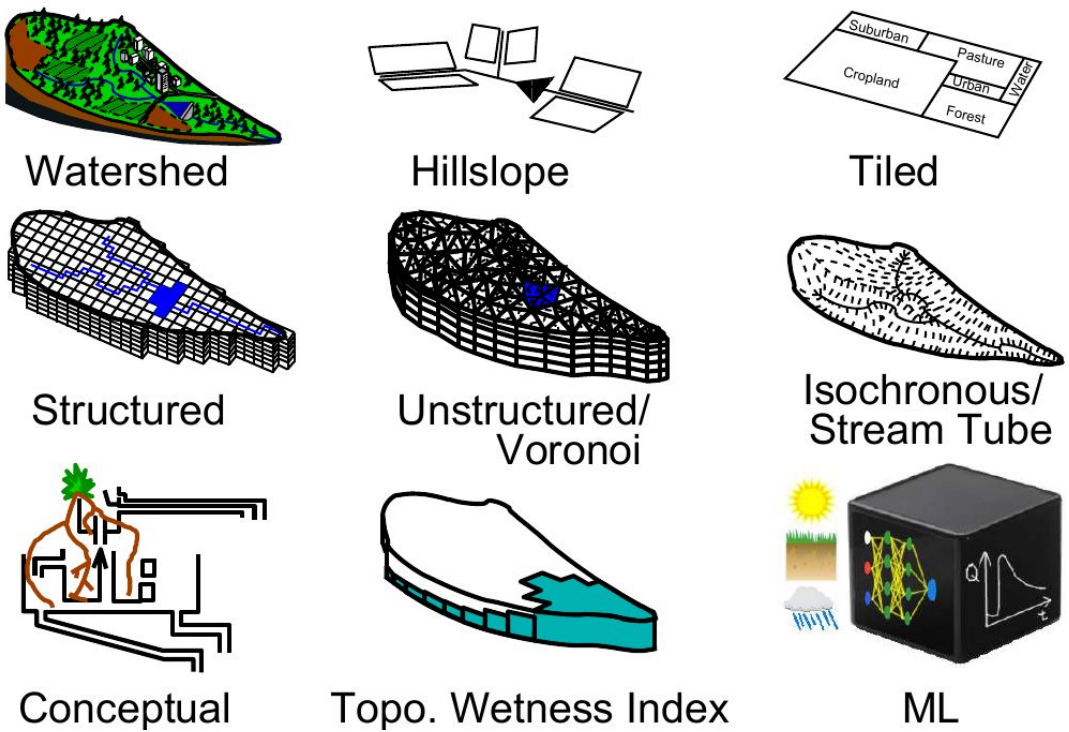
Background

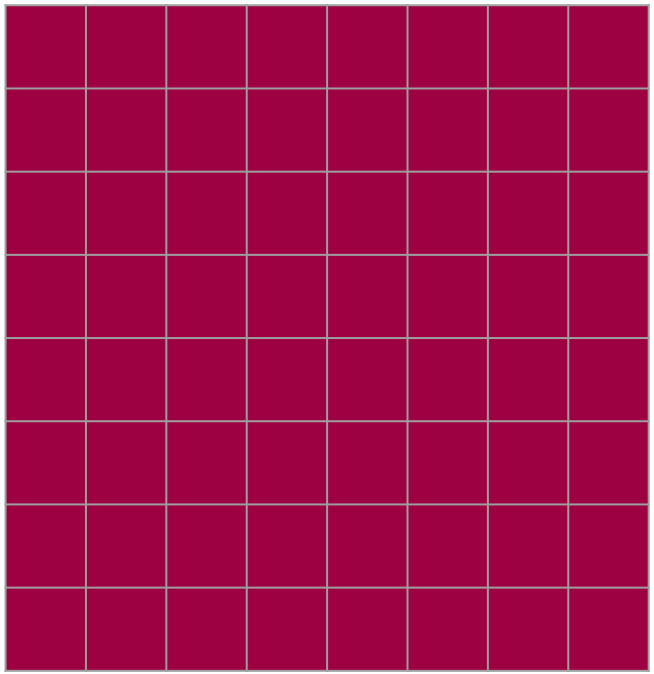
- To this point, NextGen has focused on running novel multi-module lumped catchment scale formulations
- However, hydrological models use a variety of discretizations
- This work focuses on the Structured/Regular grid formulations.

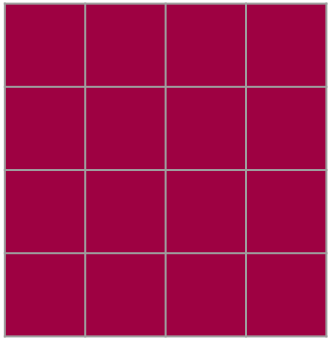


Background

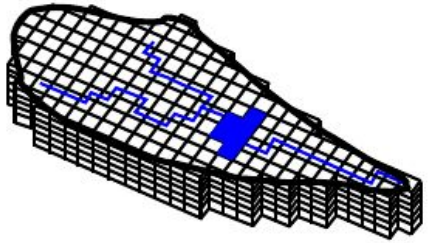
- Hydrological models use a variety of discretizations
- NextGen can run any model with a proper BMI implementation in C, C++, Python, and Fortran
- If modules are discritized differently, need to be able to transform data between them effectively







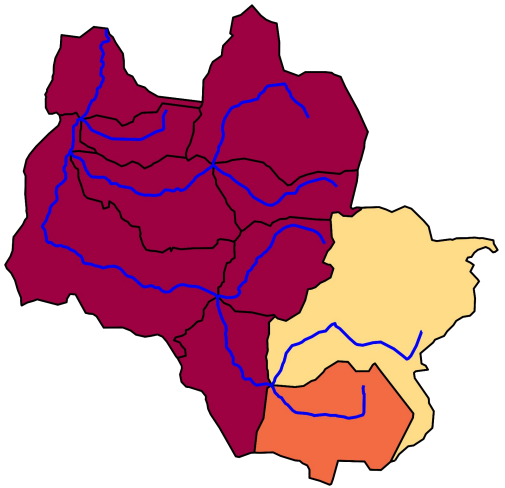
Extending capabilities with BMI



Shared Conventions

- The following existing BMI functions must support the **grid meta data variables**, i.e. **grid_<id>_***
  - get/set_value
  - get/set_value_ptr
  - get_var_item_size
  - get_var_nbytes

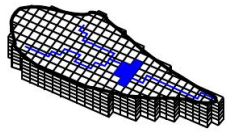
- set_value (“grid_<id>_shape”, …) now requires
  - **dynamic allocation** of internal model variables
  - May also require **re-initialization**



Dynamic Allocation Examples

```
case("grid_2_shape")
  grids(3)%shape = src
  !OUTPUT vars must be allocated.  If they have already, deallocate and allocate the correct size
  if( allocated(this%model%grid_var_3) ) deallocate( this%model%grid_var_3 )
  ! src is in z, y, x order (last dimension first)
  ! make this variable be x, y, z
  allocate( this%model%grid_var_3(src(3), src(2), src(1)) )
  if( allocated(this%model%grid_var_4) ) deallocate( this%model%grid_var_4 )
  ! src is in z, y, x order (last dimension first)
  ! make this variable be z, y, x as well
  allocate( this%model%grid_var_4(src(1), src(2), src(3)) )
  bmi_status = BMI_SUCCESS
```

Fortran



```
if( var_name == 'grid_1_shape' ):
    self.grid_1.shape = values
    for var, grid in self._grid_map.items():
        if grid.id == 1:
            #shape is set externally, need to reshape/allocate all vars associated with the grid
            self._values[var] = np.resize(self._values[var], self._grid_map[var].shape)
```

Python