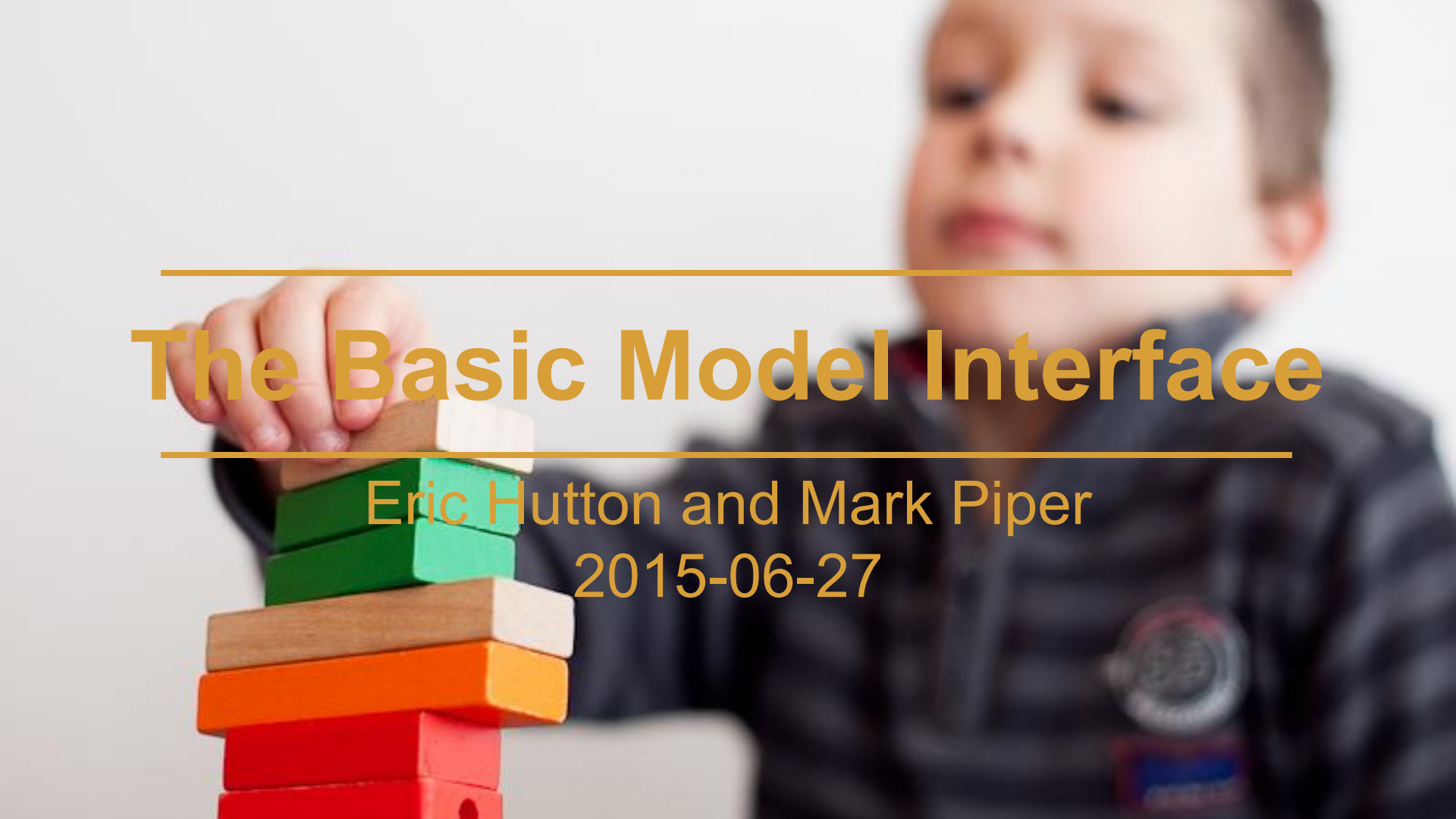


The Basic Model Interface

Eric Hutton and Mark Piper

2015-06-27

Agenda

- Factorizing your model
- Defining an interface
- The BMI walkthrough
- Hands on with a BMI (run CEM and Waves in an iPython notebook)
- Implementing a BMI (Python and then, maybe, C)

Notes are at: https://etherpad.mozilla.org/OGz6DMVtla

Some other links:

Today's clinic:

https://github.com/mcflugen/bmi-tutorial

The bmi-forum:

https://github.com/bmi-forum

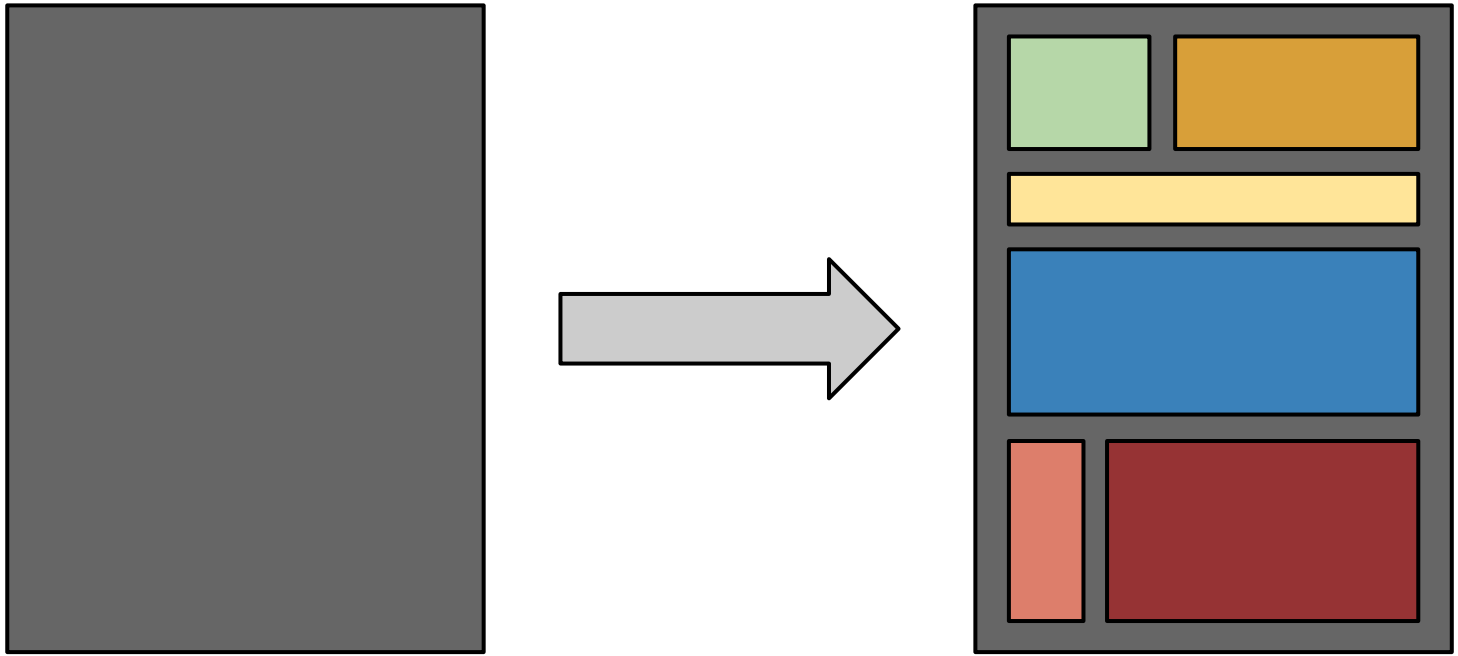
BMI documentation:

https://bmi-forum.readthedocs.org

The BMI spec:

https://github.com/csdms/bmi

Let’s forget interfaces for a moment… let’s talk about factorization.



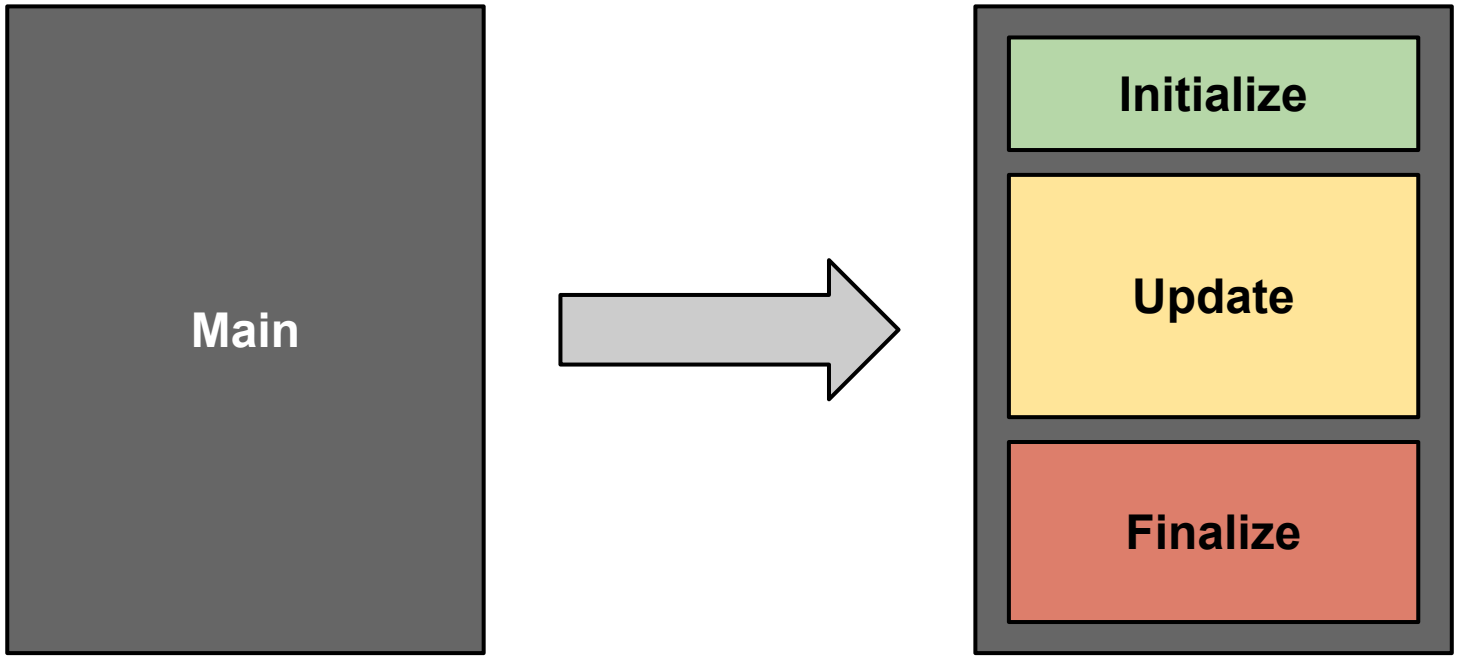
Factorization, or *Decomposition*, breaks your code into pieces but your code looks the same on the outside.

So what's the point?!

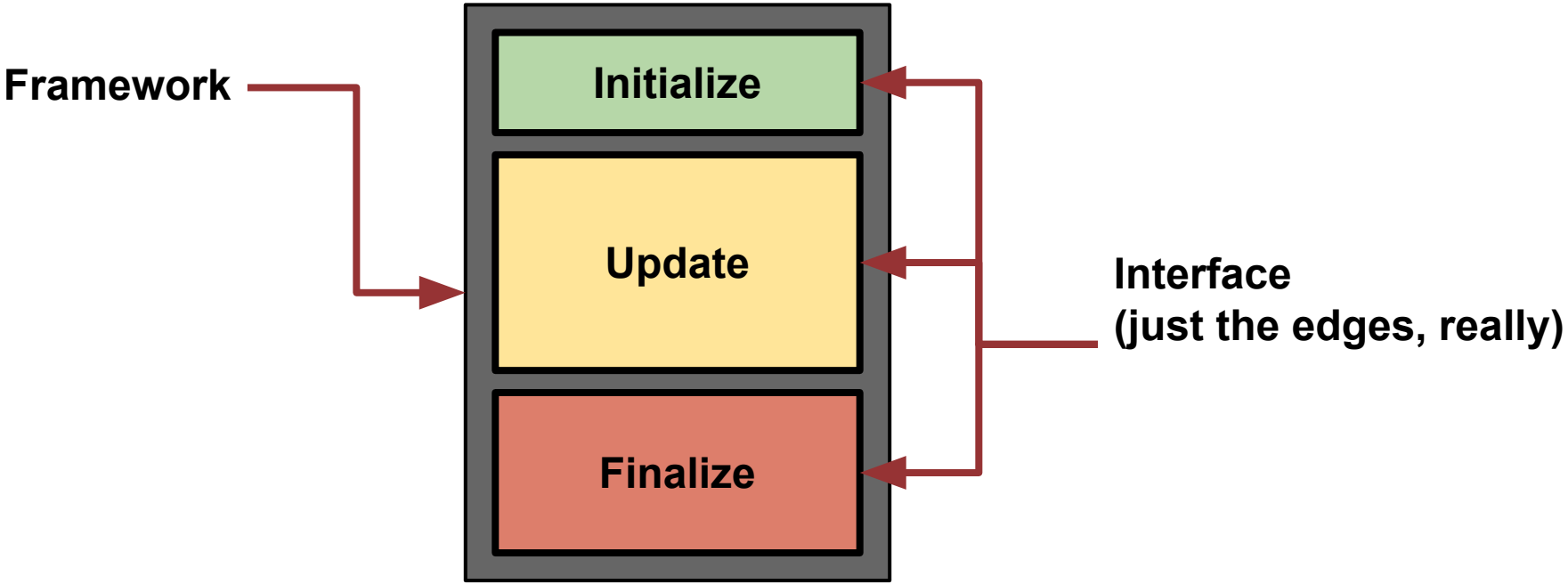
Breaks a complex problem into parts that are easier to:

- **conceive**
- **understand**
- **program**
- **maintain**
- **fine-grained entry points**
- **test!**

A Useful Decomposition for Modeling is IRF.



A framework is not the same as an interface. A framework uses and interface.

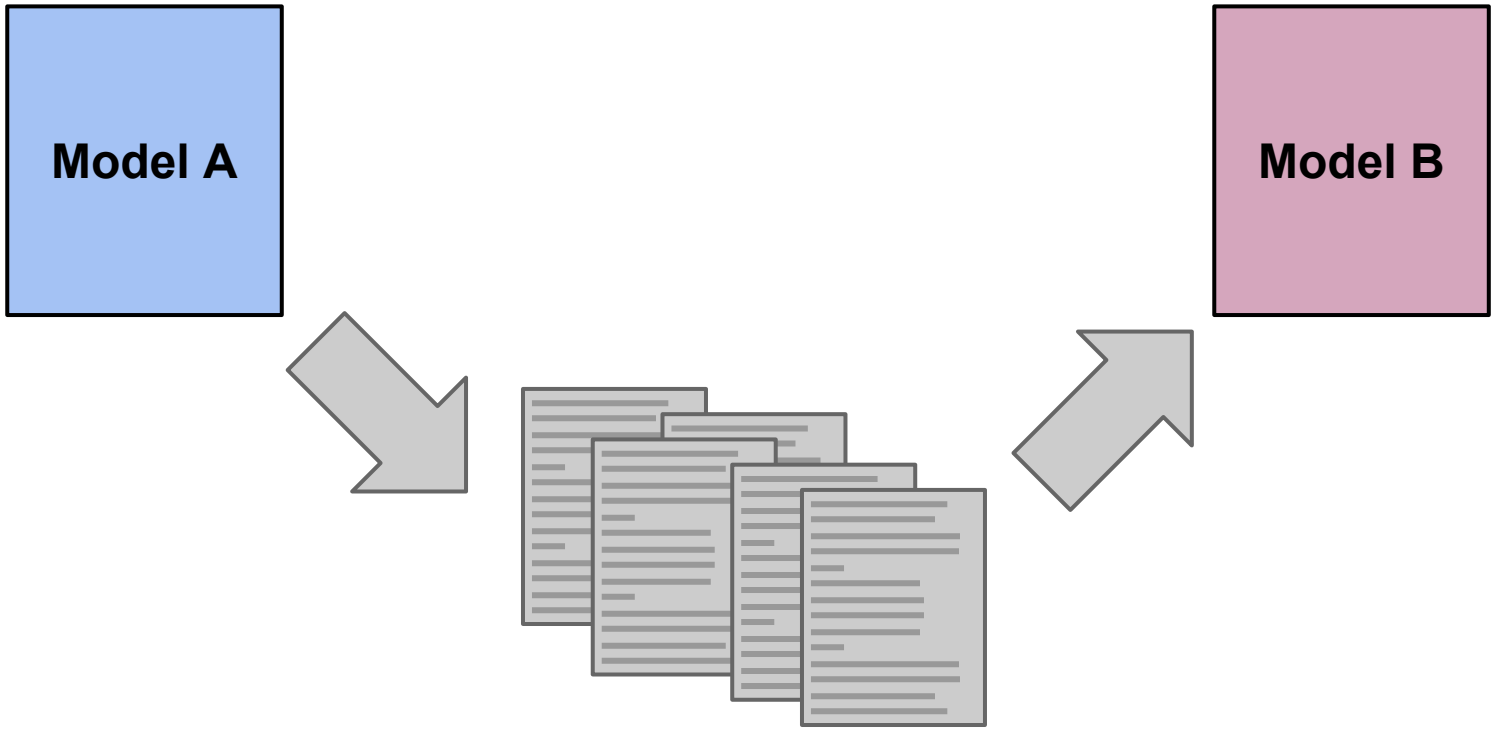



Defining an Interface

Why the Basic Model Interface?

- **Make models easy for others to use and extend - if you know one, you know them all**
- **“Others” could be people or computers or web services…**
- **Software reuse**
- **Entry points allow models to become more interactive**
- **Provides a design pattern for organizing code**
- **Model coupling**

Models interact through files, look to file format standards for guidelines.



The interface should be as easy as possible to implement as possible.

- No derived data types - just primitives
- Never inject a framework in the code
- Self describing (think netCDF)
- Don’t worry about memory management
- No user interfaces
- Auto-generated code is better than manually writing code

Remember, the interface is *not* a framework.

Some things the interface does not do:

- **Provide language interoperability**
- **Unit conversion**
- **Grid mapping**
- **Memory management**

In fact, the interface doesn't *do* anything.

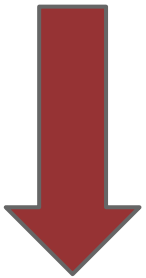
Another thing the BMI is *not*...

Finished!

The BMI Walkthrough

There are several levels to the BMI.

Basic



Advanced

Control Functions

Variables

Grids

When building libraries, we have to be very specific about the interface.

```
initialize("input.txt") != initialise()
```

The Initialize, Run, and Finalize are the most basic.

Initialize

```
for (time = 0; time < end_time; time ++) {
```

Update

```
}
```

Finalize

The Initialize, Run, and Finalize are the most basic.

```
int initialize(char* filename);

for (time = 0; time < end_time; time ++) {

    int update(void);

}

int finalize(void);
```

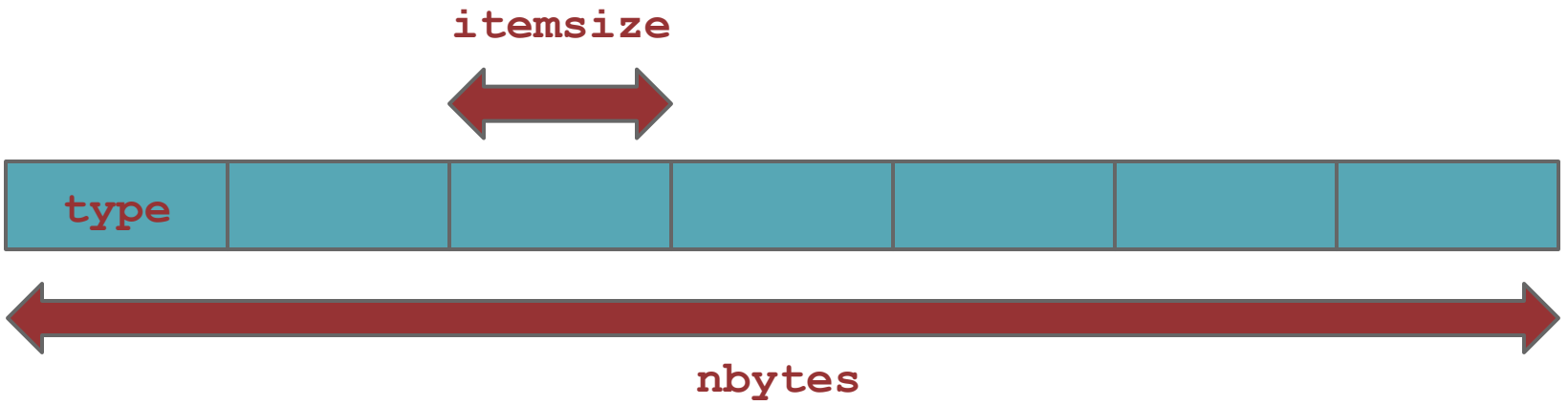
Typically, models advance through time.


```
int get_current_time(double* now);

int get_start_time(double* start);

int get_end_time(double* stop);

int get_time_step(double* dt);

int get_time_units(char* units);
```

Inquire about model variables by name through the `get_var_*` functions.

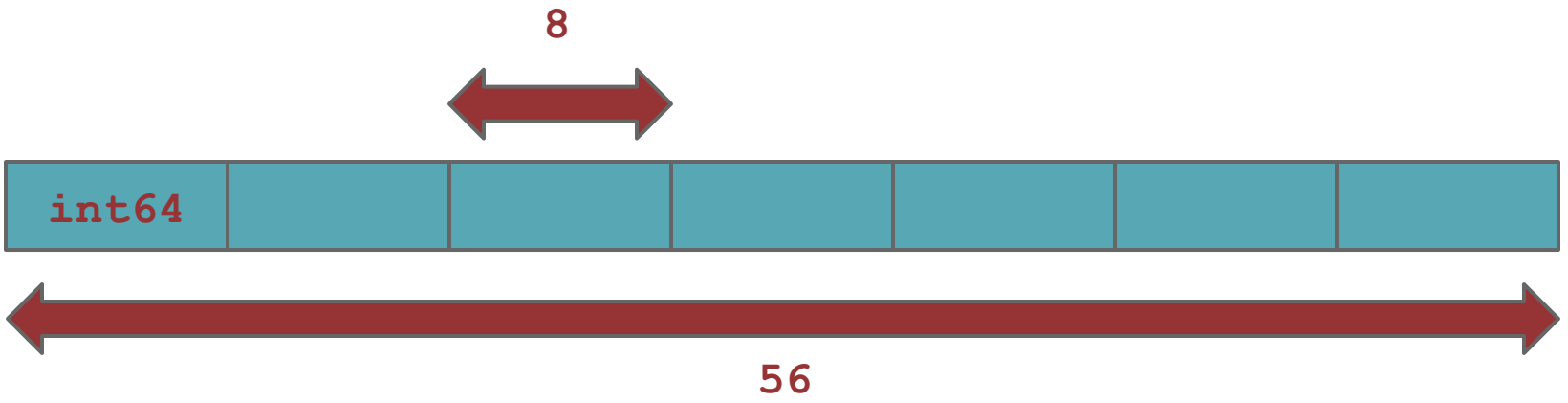

```
int get_var_units(char* name, char *units);

int get_var_type(char* name, char *type);

int get_var_nbytes(char* name, int *nbytes);

int get_var_itemsize(char* name, int *itemsize);

int get_var_grid(char* name, int *grid);
```

A model can, optionally, have input and output variables.




```
(number_of_items = nbytes / itemsize)
```

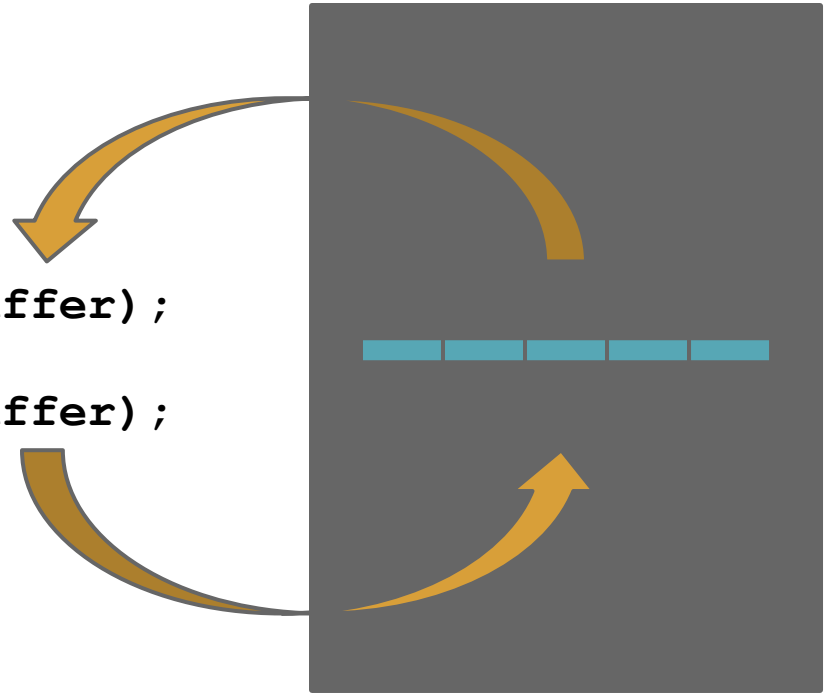
A model can, optionally, have input and output variables.



(number_of_items = 56 / 8 = 7)

Get and Set a Model's variables.

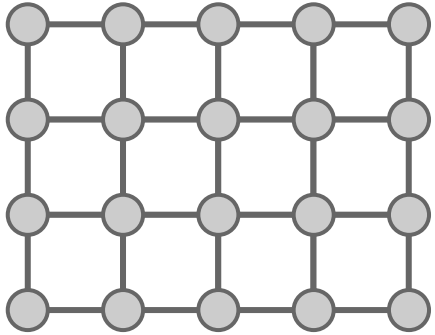
```
int get_value(char* name, void *buffer);

int set_value(char* name, void *buffer);
```



Model variables can have multiple grids.

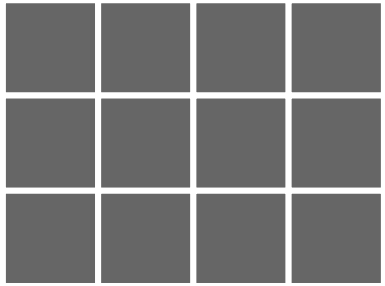
`land_surface__elevation`

grid 0



`soil__thickness`

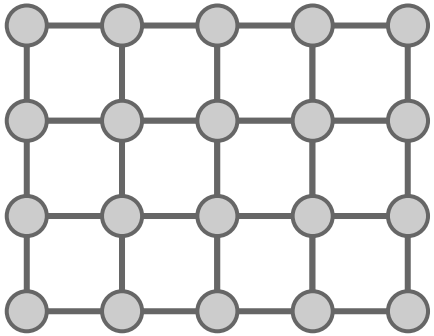
grid 1



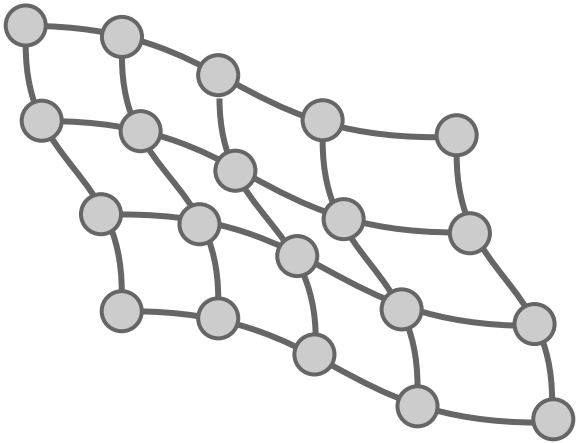
Inquire about grids with the `get_grid_*` functions.

```
int get_grid_rank(int grid_id, int *rank);
int get_grid_type(int grid_id, char *type);
```

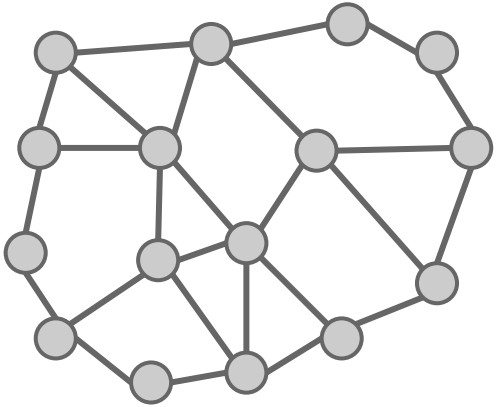
uniform_recitlinear



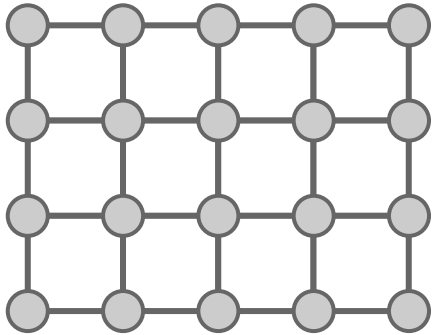
structured_quadrilaterals



unstructured



Three functions define a uniform rectilinear grid.

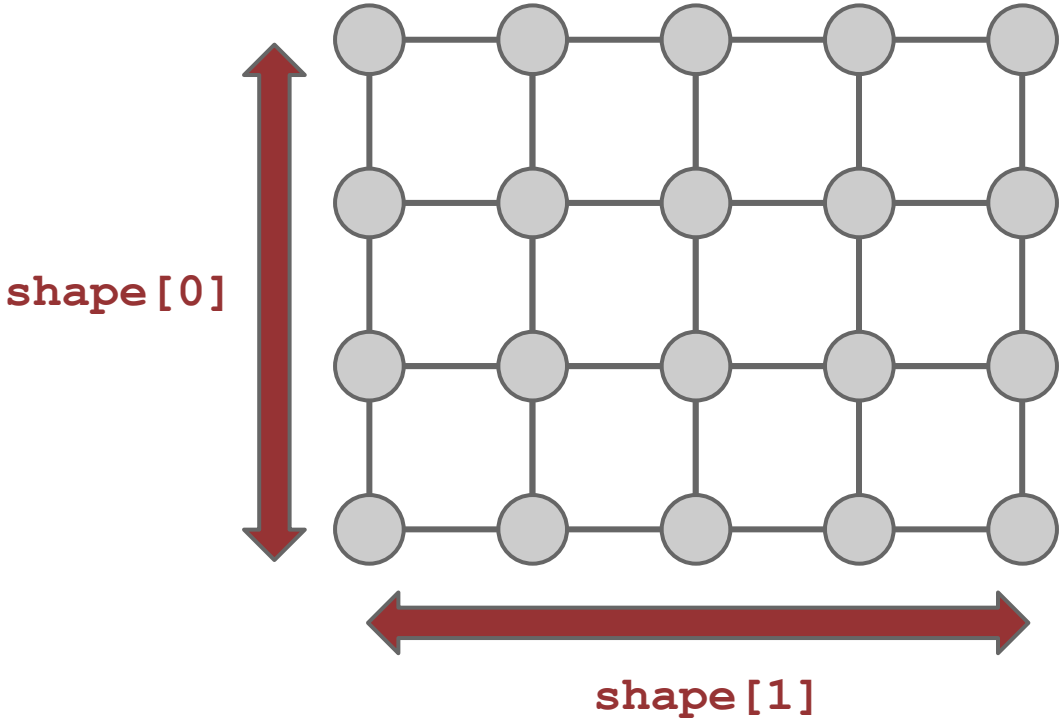


input

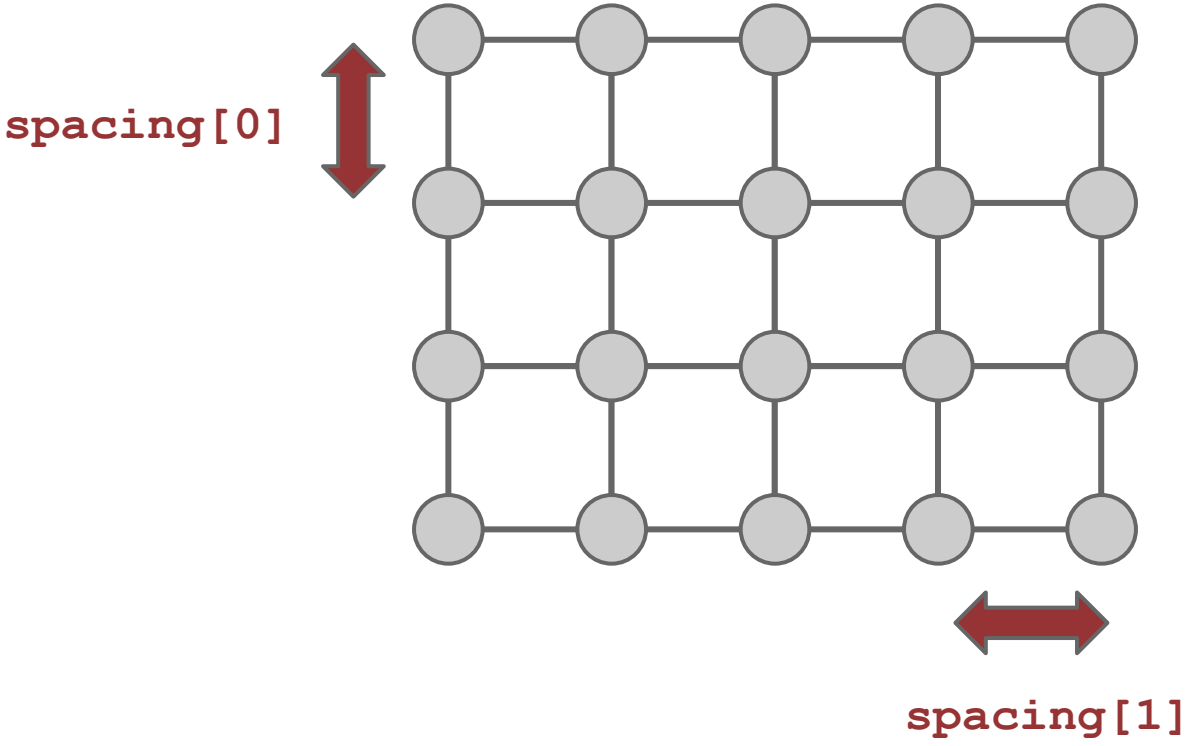
output

```
int get_grid_spacing(int grid_id, double *spacing);

int get_grid_shape(int grid_id, int *shape);

int get_grid_origin(int grid_id, double *origin);
```

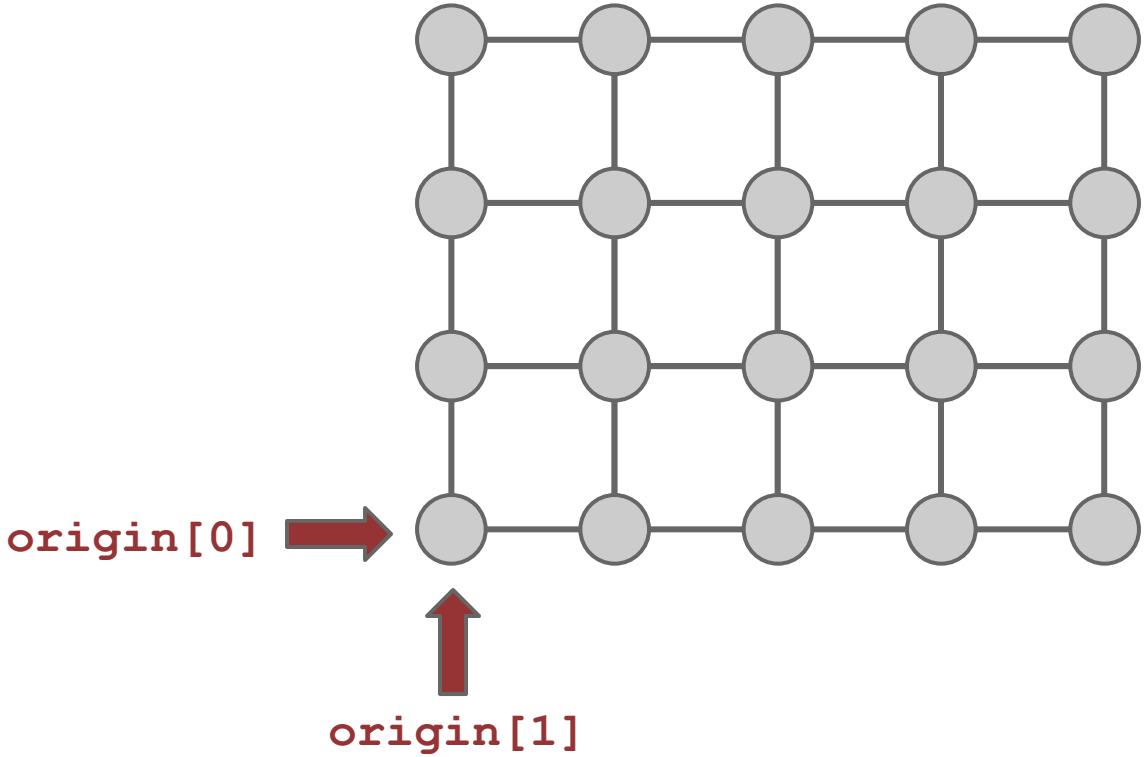
Three functions define a uniform rectilinear grid.



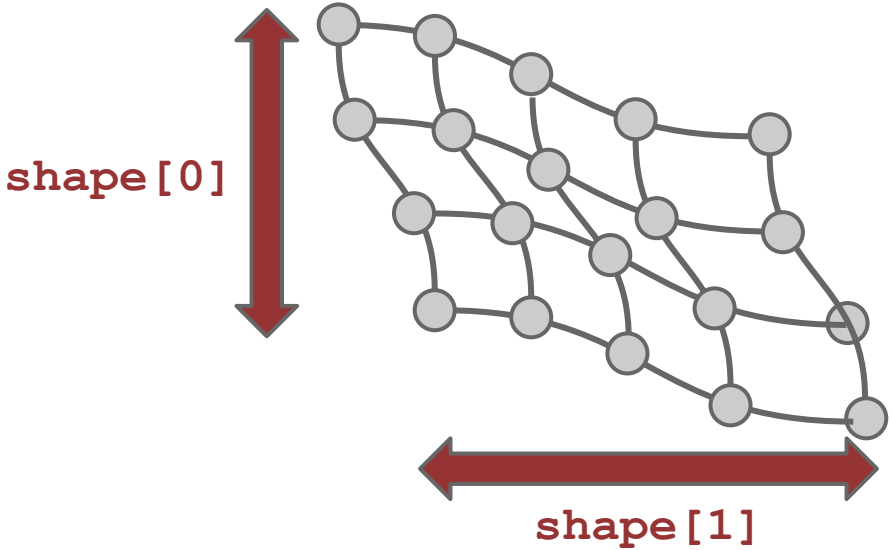
Three functions define a uniform rectilinear grid.



Three functions define a uniform rectilinear grid.

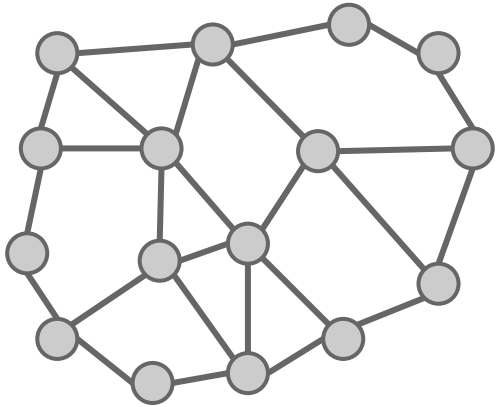


A different set of functions define a structured quadrilateral grid.



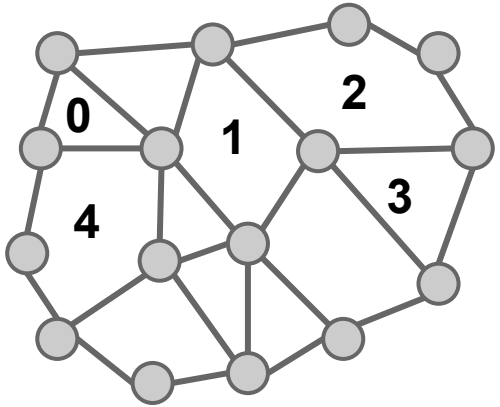

```
int get_grid_shape(int grid_id, int *shape);

int get_grid_x(int grid_id, double *x);

int get_grid_y(int grid_id, double *y);
```

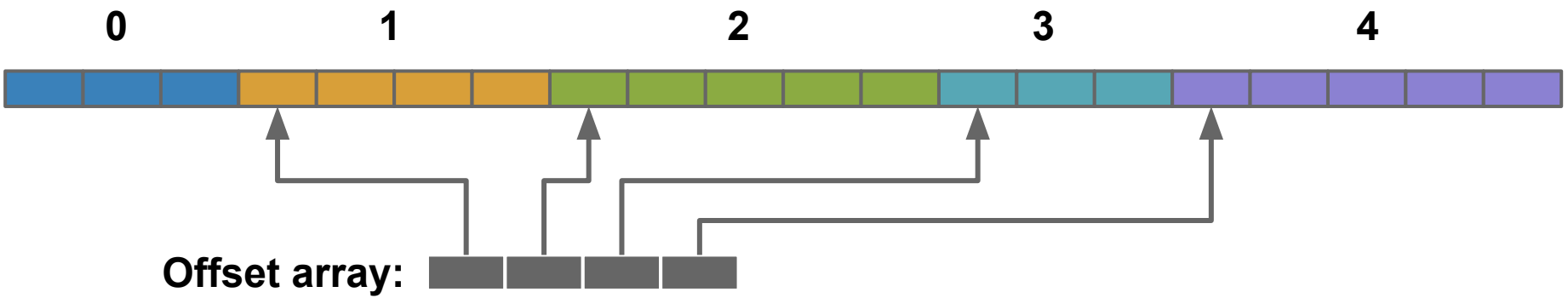
A different set of functions define an unstructured grid.



```
int get_grid_connectivity(int grid_id, int *connectivity);

int get_grid_offset(int grid_id, int *offset);

int get_grid_x(int grid_id, double *x);

int get_grid_y(int grid_id, double *y);
```

The connectivity array is one contiguous array of node IDs.





Hands on with the BMI

If you would like to follow along...

You will need:

- **internet connection**
- **beach.colorado.edu login and password**
- **ssh**
- **tutorial files**

Open an iPython notebook on beach.

Connect to beach with ssh:

```
locahost> ssh [username@]beach.colorado.edu
```

Clone clinic files, configure environment, and start ipython notebook:

```
beach> git clone https://github.com/mcflugen/bmi-tutorial
beach> cd bmi-tutorial
beach> source scripts/setup-beach.sh
beach> ipython notebook --no-browser
```

Connect to the iPython notebook

In another terminal:

```
locahost> ssh -N -L 8888:localhost:8888 beach.colorado.edu
```

Go here: http://localhost:8888


Building a BMI

We'll step through an example that solves the heat equation on a 2D grid.

Step 1: Refactor into IRF

Step 2: Wrap code as a class

Step 3: Add some more advanced BMI functions

These files are in the examples folder of the bmi-tutorial repository.

https://github.com/mcflugen/bmi-tutorial


Questions?