GNOME:

Using Python to drive the General NOAA Operational Modeling Environment.

Christopher H. Barker
Jasmine Sandhu

NOAA Office of Response and Restoration
Emergency Response Division
Chris.Barker@noaa.gov

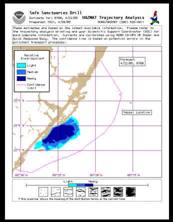
Introduction:

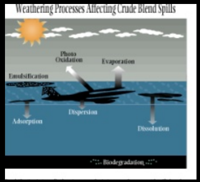
- Way too much to talk about
- Grab bag of Python API and wrapping/coding issues
- I’m not even going to mention algorithms!

NOAA Emergency Response Division

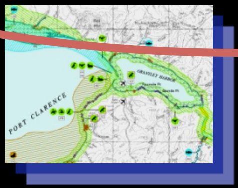
- Provide 24/7 scientific support during oil and hazardous material releases
- Identify, assess, prioritize, and mitigate injuries caused by hazardous material releases
- Accelerate restoration and recovery by integrating habitat improvement into response actions
- Provide credible, timely and appropriate advice

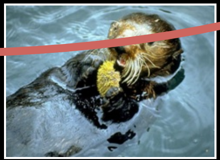
What happened?



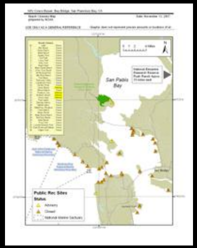


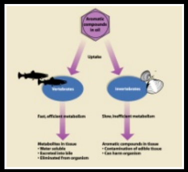
Where will it go?





Who does it hit?





How does it hurt?

So what?

Oil Spill Model

- Quick to initialize: Answers within hours
- Easy to Calibrate: What if the results do not match the field obs?
- Wide range of Scales
- Can ingest whatever is available:
  - HF Radar, other's hydro models, etc.

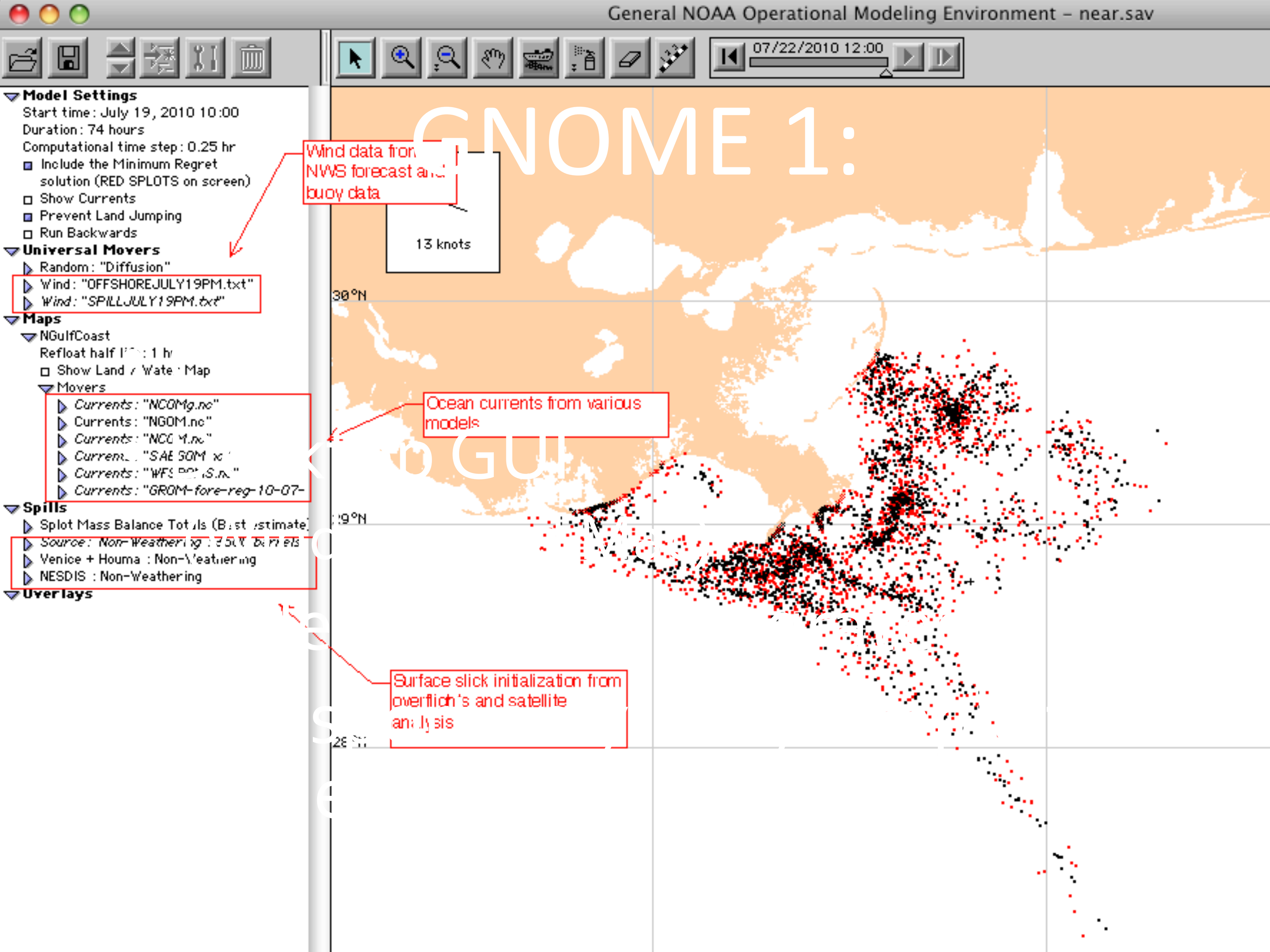
Particle Tracking

+

Flexible Framework

GNOME 1:

- C++
- Desktop GUI
  (Windows and Mac Classic)
- Code is tightly integrated
- Primarily Transport
  (very simple weathering)
- Limited Batch Mode



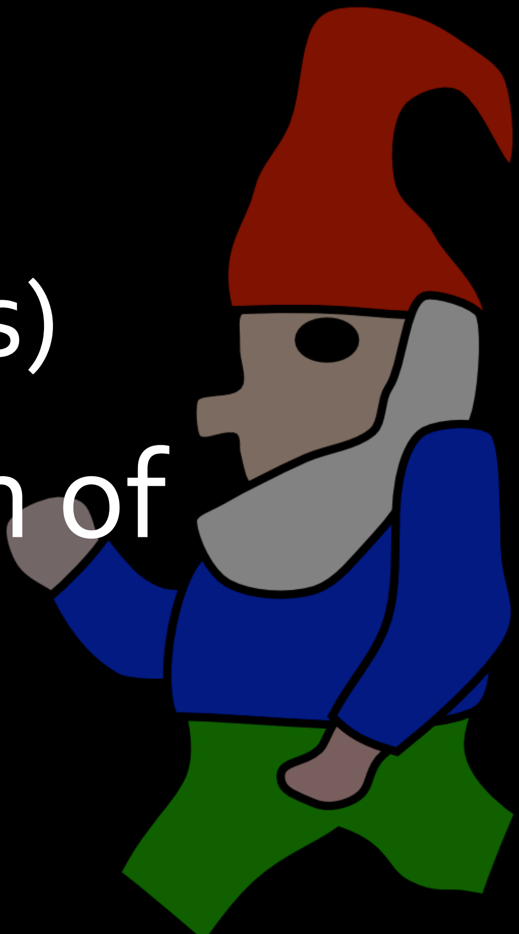
GNOME 1:
General NOAA Operational Modeling Environment – near.sav
07/22/2010 12:00
Model Settings
Start time: July 19, 2010 10:00
Duration: 74 hours
Computational time step: 0.25 hr
Include the Minimum Regret solution (RED SPLOTS on screen)
Show Currents
Prevent Land Jumping
Run Backwards
Universal Movers
Random: "Diffusion"
Wind: "OFFSHOREJULY19PM.txt"
Wind: "SPILLJULY19PM.txt"
Maps
NGulfCoast
Movers
Currents: "NCOMg.nc"
Currents: "NGOM.nc"
Spills
Overlays
Wind data from NWS forecast and buoy data
13 knots
30°N
Ocean currents from various models
Surface slick initialization from overflight's and satellite analysis

GNOME 2 Goals

- Scripting Interface
- Easier to add new features
  - Plug in your own Movers, Weatherers, Maps, Element Types
- Easier to test/maintain/improve
- Open Source Development model.

GNOME Key Features:

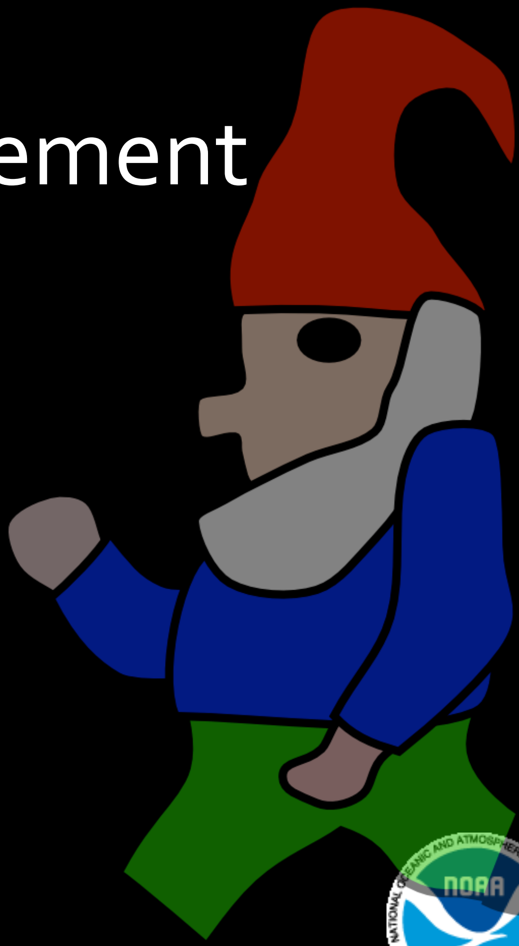
- Particle Tracking
  (Lagrangian Elements)
- Linear Superposition of Physical processes



GNOME Key Components

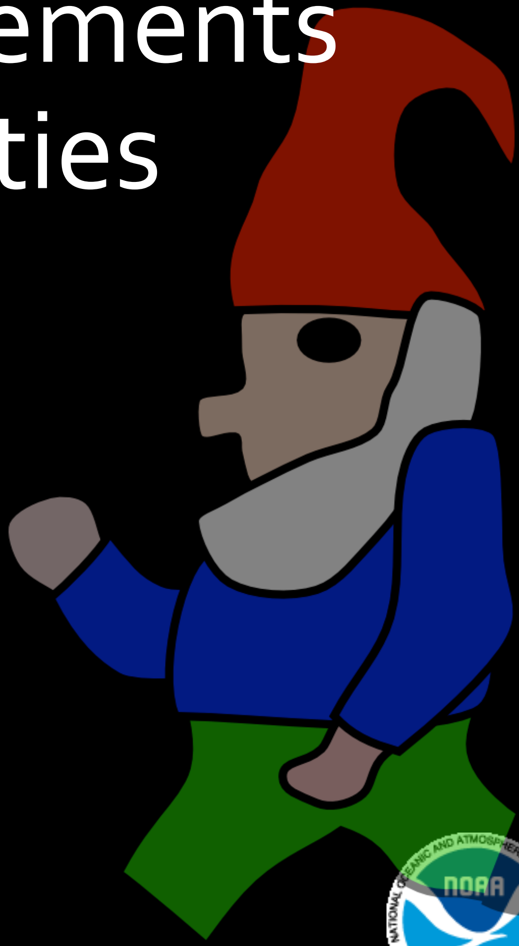
- "Movers":

  Anything that moves an element is a "mover":
  - Wind
  - Currents
  - Random Diffusion
  - Droplet Buoyancy
  - Larva Behavior
  - ???



GNOME Key Components

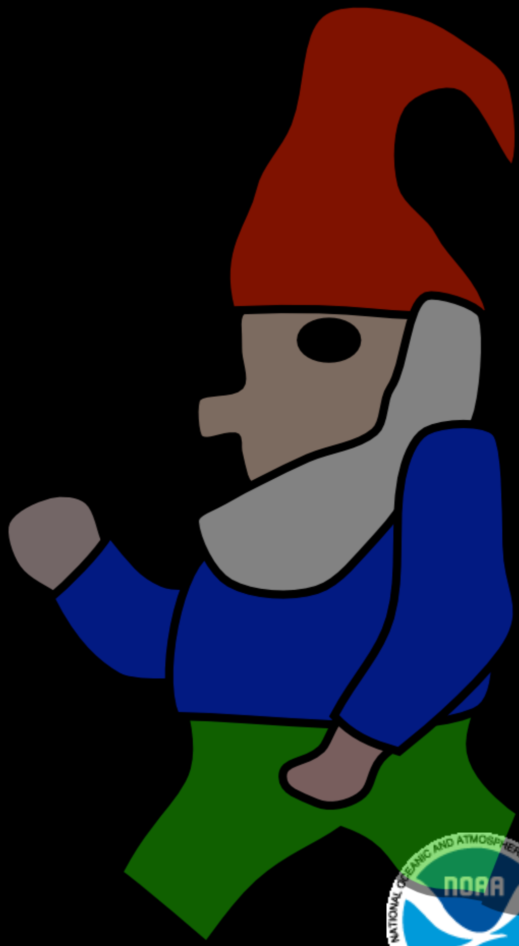
- “Spills”--Sources of elements with prescribed properties
    - Point Source
    - “spray can”
    - Known positions
    - Plume model
    - ???



GNOME Key Components

"Maps":

- Define Shoreline and/or bathymetry
  - Beaching/Refloating
  - Interaction with Bottom



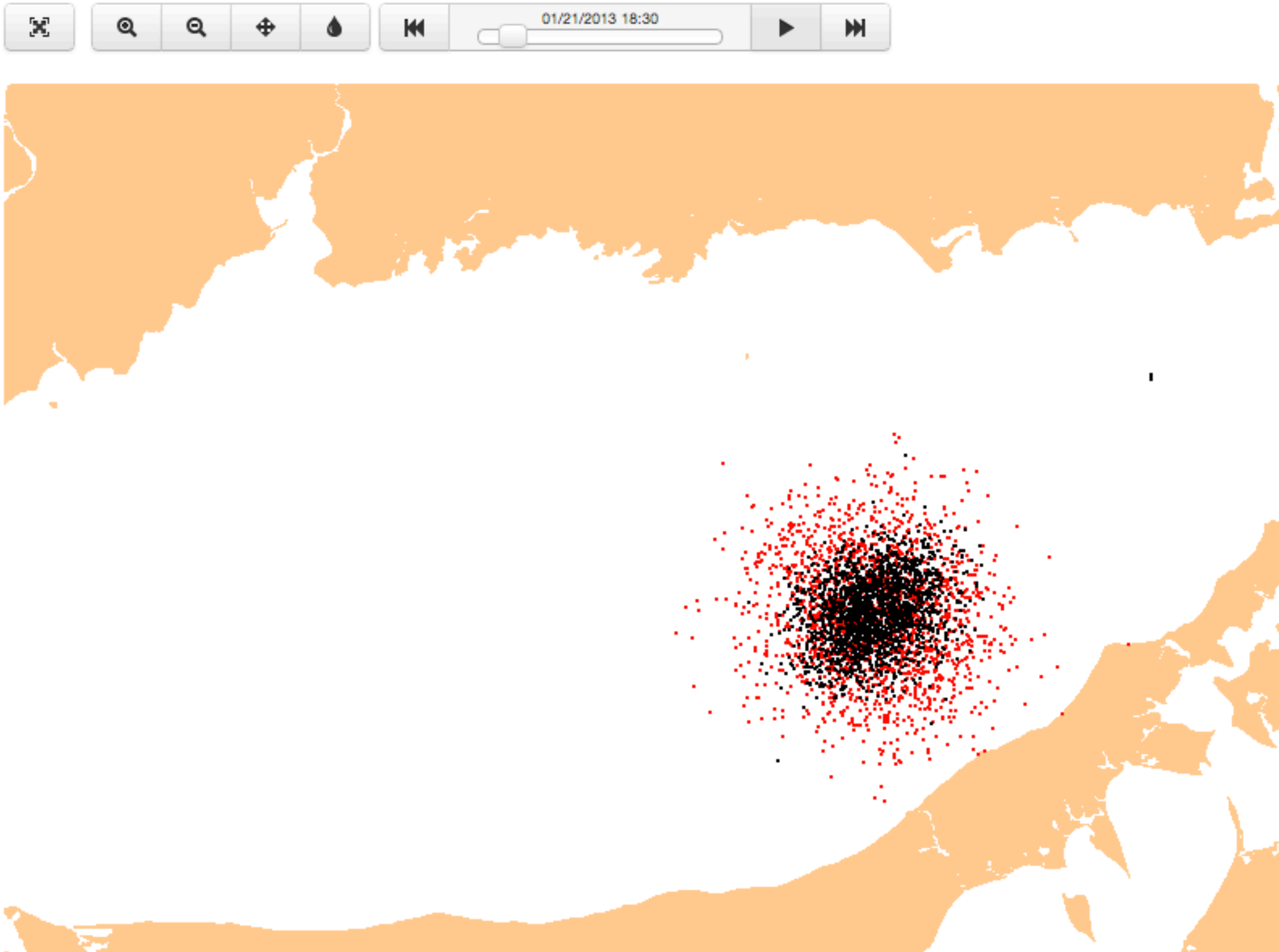
GNOME 2 Main Loop

- Initialize model--call initializer for:
  - Each spill
  - Each mover
- For each time step:
  1. Loop through the Movers
  2. Beach (refloat) the elements
  3. Write output

GNOME 2 Web Interface:

- Running on NOAA server
  - Simple use (location files)
  - Intuition building
  - Educational
  - Download/save your setup.
- Run your own server:
  - Custom locations, etc.

- Model Settings
  - Map: Long Island Sound
  - Start Time: 2013-01-21 12:00:
  - Is Uncertain: True
  - Time Step: 0.25
  - Duration Hours: 0
  - Duration Days: 2
  - Id: 0D5F159E-6404-11E2-Ab2I
- Movers
  - Wind Mover
    - On: True
    - Uncertain Angle Scale: 0.4
    - Uncertain Duration: 10800.(
    - Active Start: 1970-01-01 00
    - Active Stop: 9999-12-31 23:
    - Uncertain Angle Scale Units
    - Uncertain Time Delay: 0.0
    - Id: 0F713F30-6404-11E2-9F
    - Wind: {'Units': 'Mps', 'Time:
    - Uncertain Speed Scale: 2.0
  - Random Mover
    - Diffusion Coef: 500000.0
    - On: True
    - Active Start: 1970-01-01 00
    - Active Stop: 9999-12-31 23:
    - Id: 0F712282-6404-11E2-93



PyGnome Goals:

- Fully Python Scriptable
- Add your own movers, etc. in Python (or any code called from Python)
- Use Legacy C++ code

pyGnome Structure

- Model written in pure Python
- Clean(?) API for:
  - Movers
  - Environment
  - Maps
  - Spills
  - Outputters

The SpillContainer:

- Former structure:
  - Big C struct: all the data associated with each element -- each new application required new data, new struct
- New approach:
  - Memory Efficient:
    - Movers extract only the data they need to work with
  - Dynamic: Add new array types on the fly, at run time.
  - Adapts itself to the movers, element types being modeled.

The SpillContainer:

- Dictionary of numpy arrays:
  - all of length: number of elements
- One array for each property of the elements
- Movers only need to work with the data they need.
- Manages the addition and removal of elements
- SpillContainer gets passed to Movers, Maps, Outputters, etc.

SpillContainer API:

```
class SpillContainer(object):
    def __getitem__(self, data_name):
        return self._data_arrays[data_name]

    @property
    def num_elements(self):
        return len(self['positions'])

    def rewind(self):
        “”” reset eveything to initial conditions”””

    def release_elements(self, current_time, time_step):
        """
        This calls release_elements on all of the
        contained spills, and adds the elements to the
        data arrays
        """
```

Mover API:

```
class Mover(object):
  def __init__(self, on=True,
                     active_start=InfDateTime('-inf'),
                     active_stop=InfDateTime('inf'),
                  ):
      """
      :param on: boolean as to whether the object is on or
                 not. Default is on
      :param active_start: datetime when the mover should
                           be active
      :param active_stop: datetime after which the mover
                          should be inactive
      """
```

Mover API:

```
def prepare_for_model_run(self):
    pass


def prepare_for_model_step(self,
                           sc, #spill_container
                           time_step,
                           model_time):
    pass

def model_step_is_done(self,
                       sc=None):
    pass
```

Mover API: `get_move`

```
def get_move(self,
             sc, # spill_container
             time_step
             model_time_datetime):
```

- Returns the “delta”:
  - change in position of the elements
  - In floating point lat-lon coordinates
- Deltas for each mover independently computed: order doesn’t matter.

Writing a simple Mover:

Steady, uniform current

```
class SimpleMover(Mover):
  def __init__(self, velocity, **kwargs):
    """
    :param velocity: a (u, v, w) triple –
                     in meters per second
     """
      self.velocity = np.asarray( velocity,
                                  dtype=
                                  basic_types.mover_type
                                  ).reshape((3,))

    super(SimpleMover,self).__init__( **kwargs)
```

Writing a simple Mover (cont):

```
def get_move(self,spill, time_step, model_time):
    try:
        positions       = spill['positions']
        status_codes    = spill['status_codes']
    except KeyError, err:
        raise ValueError("The spill does not have the
                   required data arrays\n"+err.message)

    # which ones should we move?
    in_water_mask =  (status_codes ==
                     basic_types.oil_status.in_water)

    # compute the move
    delta = np.zeros_like(positions)

        if self.active and self.on:
```

Writing a simple Mover (cont):

```
if self.active and self.on:
    delta[in_water_mask] = self.velocity *
                           time_step

    # scale to lat-lon
    delta = proj.meters_to_lonlat(delta, positions)

    return delta
```

Spill API

Adds elements to the model:

```
class Spill(object):
    def __init__(self, on=True, id=None):

    def rewind(self):

    def release_elements(self,
                         current_time,
                         time_step,
                         array_types=None):
```

Map API

Checks for element's interaction with shoreline, bottom, boundaries of map, water surface:

```
class GnomeMap():
  def allowable_spill_position(self, coord):

  def beach_elements(self, spill_container):
      """
      Determines which LEs were or weren't beached
      or moved off map
      """
  def refloat_elements(self, spill_container,
                                 time_step):
```

PyGnome Serialization

Saving a model set-up

PyGnome Serialization

- Goal:
  - Facilitate web client/server data exchange
  - Check validity of objects prior to deserialization
    - Custom objects: Numpy arrays, inf_datetime
  - Persist model state in human readable form
    - Simply write out as text files to persist
    - Read and restore model from these files
  - Use same process for web data exchange and persistence

- Solution: serialize to JSON format!
- Use (monkey patched) colander to validate data (http://docs.pylonsproject.org/projects/colander)

PyGnome Serialization - example

Serializable mixin for serialized objects

```
class WindMover(CyMover, serializable.Serializable):
    _update = ['uncertain_duration',
               'uncertain_time_delay',
               'uncertain_speed_scale',
               'uncertain_angle_scale']
    _create = ['wind_id']
    _read = ['wind_id']
    _create.extend(_update)

    state = copy.deepcopy(CyMover.state)
    state.add(read=_read, update=_update, create=_create)
```

PyGnome Serialization - example

new_from_dict used to restore object after persistence

```
    @classmethod
    def new_from_dict(cls, dict_):
        """
        define in WindMover and check wind_id matches wind

        invokes: super(WindMover,cls).new_from_dict(dict\_)
        """
        wind_id = dict_.pop('wind_id')
        if dict_.get('wind').id != wind_id:
            raise ValueError("id of wind object does not match the wind_id
parameter")

        return super(WindMover,cls).new_from_dict(dict_)
```

Base implementation (Serializable mixin) of new_from_dict

```
    @classmethod
    def new_from_dict(cls, dict_):
        """

        creates a new object from dictionary
        This is base implementation and can be over-ridden by classes
using mixin
        """
        return cls(**dict_)
```

PyGnome Serialization - example

Examples of _to_dict and overridden from_dict

```
    def wind_id_to_dict(self):
        """
        used only for storing state so no wind_id_from_dict is defined.
        This is not a read/write attribute.
        """
        return self.wind.id

    def from_dict(self, dict_):
        """
        For updating the object from dictionary

        'wind' object is not part of the state since it is not serialized
        however, user can still update the wind attribute with new Wind
        object. It must be poped out of the dict() here, then call super
        to process the standard dict\_
        """
        self.wind = dict_.pop('wind',self.wind)

        super(WindMover, self).from_dict(dict_)
```

PyGnome Serialization for persistance

Model contains

- map
- renderer
- wind mover
- random mover
- cats shio mover
- cats ossm mover
- plain cats mover
- single spill

Output files when model is persisted

Model_a1ce3e42-db61-11e2-8899-3c075404123e.txt
MapFromBNA_a1ce1b1c-db61-11e2-b63c-3c075404123e.txt
Renderer_a1dbdd2e-db61-11e2-b63c-3c075404123e.txt
SurfaceReleaseSpill_a1dbe1ca-db61-11e2-b0e1-3c075404123e.txt

Wind_a1dc0a73-db61-11e2-9cc4-3c075404123e.txt
WindMover_a1dc0d3a-db61-11e2-a9b3-3c075404123e.txt
Tide_a1df53e6-db61-11e2-b27a-3c075404123e.txt
Tide_a1f39bcf-db61-11e2-81d0-3c075404123e.txt
CatsMover_a2100323-db61-11e2-9bb2-3c075404123e.txt
CatsMover_a1efaee3-db61-11e2-8601-3c075404123e.txt
CatsMover_a203a097-db61-11e2-b57d-3c075404123e.txt
RandomMover_a1dbf16b-db61-11e2-9fb5-3c075404123e.txt

EbbTides.cur
EbbTidesShio.txt
MassBayMap.bna
MassBaySewage.cur
MerrimackMassCoast.cur
MerrimackMassCoastOSSM.txt

PyGnome Serialization - JSON

JSON from Model_a1ce3*.txt

```
….
"environment": {
  "dtype": "<class
'gnome.environment.Environment'>",
  "id_list": [
   [
    "gnome.environment.Wind",
    "a1dc0a73-db61-11e2-9cc4-3c075404123e"
   ],
   [
    "gnome.environment.Tide",
    "a1df53e6-db61-11e2-b27a-3c075404123e"
   ],
   [
    "gnome.environment.Tide",
    "a1f39bcf-db61-11e2-81d0-3c075404123e"
   ]
  ]
 },
…
```

JSON in Wind_a1dc0a*.txt

```
{
 "obj_type": "gnome.environment.Wind"
 "name": "Wind Object",
 "updated_at":
"2013-06-22T10:31:54.839908",
 "source_type": "undefined",
 "source_id": "undefined",
 "timeseries": [
  [
   "2013-02-13T09:00:00",
   5.0,
   180.0
  ],
  [
   "2013-02-14T03:00:00",
   5.0,
   180.0
  ]
 ],
 "units": "m/s",
 "id": "a1dc0a73-
db61-11e2-9cc4-3c075404123e"
 "description": "Wind Object"
}
```

Heavy use of Cython

- "Cython is an optimizing static compiler for both the Python programming language and the extended Cython programming language":
  www.cython.org

- Calling legacy C++ code
- Optimizing bits of Python

Mapping to old C++ Mover API

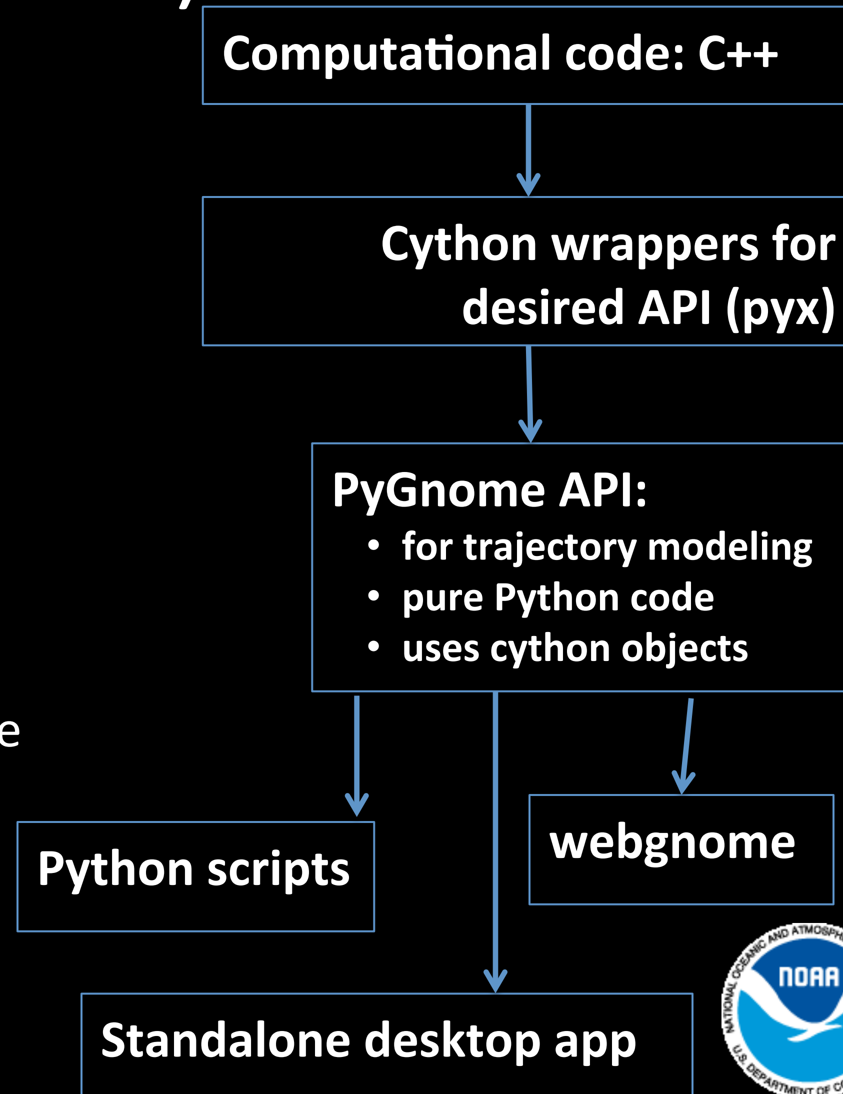
```
GetMove(const Seconds& model_time,
         Seconds timeStep
         long setIndex
         long leIndex
         LERec *theLE,
         LETYPE leType);
```

Cython to:

- Extract arrays from SpillContainer
- Loop through elements
- Call C++ method

PyGnome Cython bindings (C++ libgnome)

- Cython wrappers
  - Serves to test C++ code
  - Easier to test C++ and Python functionality independently
  - Take advantage of C++ heritage and computational code
  - Extend C++ functionality. In some cases implement features in Python instead of using C++ methods



PyGnome: C++/Cython/Python

C++

Mover_c
Base class: C++ movers

inheritance

Random_c
WindMover_c
GridCurrentMover_c
GridWindMover_c

Cython

CyMover
Base class: cy_* movers

inheritance

CyRandomMover
CyWindMover
CyGridCurrentMover
CyGridWindMover

Python

Mover
Base class: Python movers

inheritance

RandomMover
WindMover
GridCurrentMover
GridWindMover

Composition

Polymorphism in Cython bindings

- CyMover
  - Implement methods common to all movers
- cy_mover.pxd defines`Mover_c` type
  - Similar to header file; accessible by cython objects
  - Derived classes instantiate `Mover_c` object
  - Derived classes must do a `dynamic_cast`
  - dynamic_cast operation not directly supported in Cython

Polymorphism in Cython bindings

cy_mover.pyx

```
cdef class CyMover(object):
    """
    Class serves as a base class for cython wrappers around C++
movers. This provides the
    default implementation for




    In general, the cython wrappers (cy_*) will instantiate the
correct C++ object, say
    cy_wind_mover instantiates self.mover as a WindMover_c object.

    def prepare_for_model_run(self):
        """
        default implementation. It calls the C++ objects's
PrepareForModelRun() method
        """
        if self.mover:
            self.mover.PrepareForModelRun()
```

Polymorphism in Cython bindings

cy_mover.pxd

```
"""
Class serves as a base class for cython wrappers around C++
movers. The C++ movers derive from Mover_c.cpp. CyMover
defines the mover pointer, but each class that derives from
CyMover must instantiate this object to be either a
WindMover_c, RandomMover_c, and so forth
"""
cdef class CyMover:
    cdef Mover_c * mover
```

Polymorphism in Cython bindings

cy_random_mover.pyx

```
cdef extern from *:
    Random_c* dynamic_cast_ptr "dynamic_cast<Random_c *>" (Mover_c *)
except NULL

cdef class CyRandomMover(cy_mover.CyMover):

    cdef Random_c *rand

    def __cinit__(self):
        self.mover = new Random_c()
        self.rand = dynamic_cast_ptr(self.mover)
```

This does not work!

```
self.rand = dynamic_cast<Random_c *>(self.mover)
```

Generated C++ does the dynamic_cast correctly

```
dynamic_cast<Random_c *>(__pyx_v_self->__pyx_base.mover);
```

```
__pyx_t_1 = dynamic_cast<Random_c *>(__pyx_v_self->__pyx_base.mover);
```

A Note on Standards

When we are not sharing code, let's at least sjare data and results:

- Unstructured Grid standard:
    - Py_ugrid sprint

- Particle tracking model standard for netcdf

The GNOME-dev team

Caitlin O’Connor
Jasmine Sandhu
James Makela
Amy Macfadyen
Brian Zelenke
(Andrew Brookins)

Partner With Us

Chris Barker
NOAA Office of Response and Restoration
Emergency Response Division
Chris.Barker@noaa.gov

The Source:

https://github.com/NOAA-ORR-ERD/GNOME