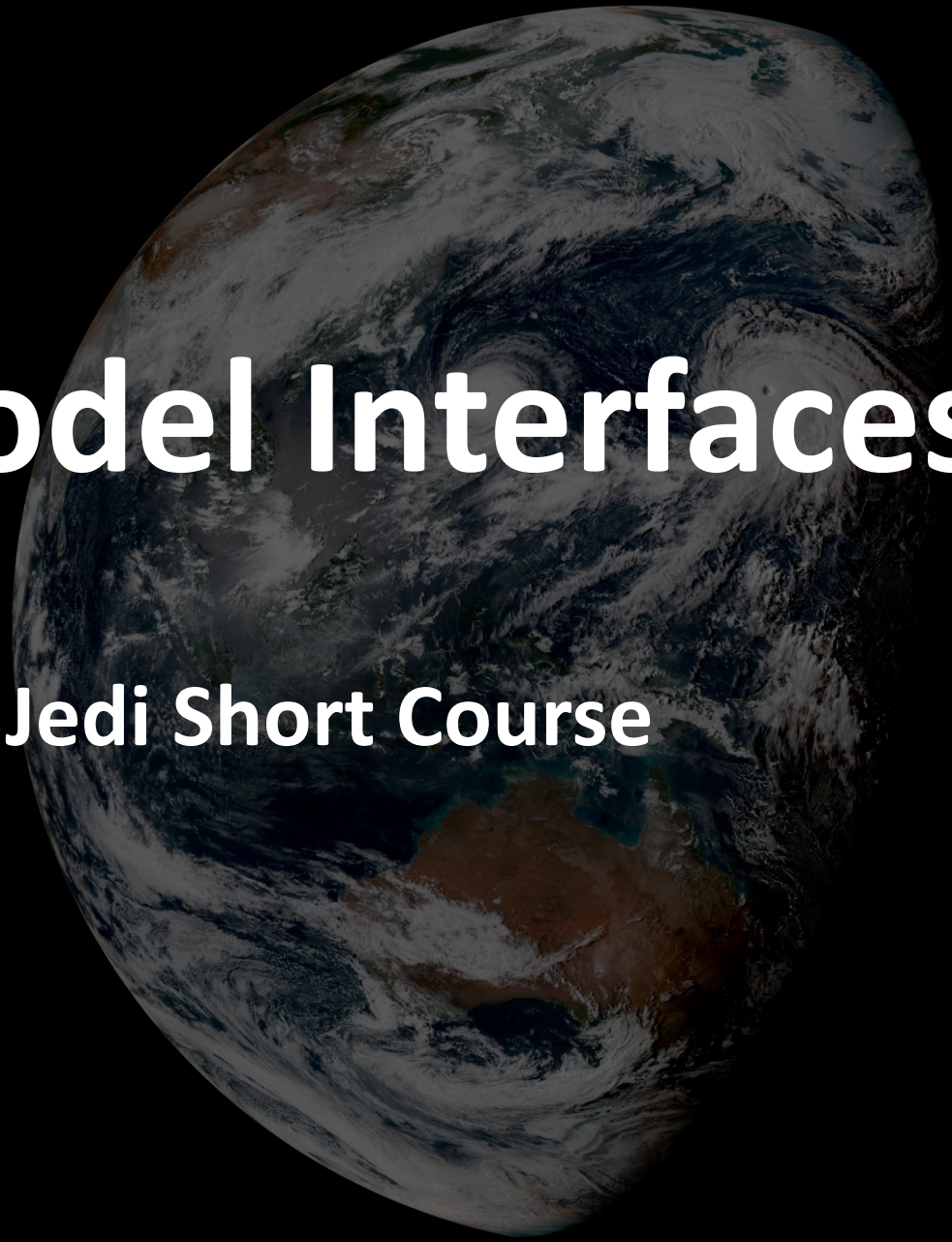


Model Interfaces

AMS Jedi Short Course

U.S. AIR FORCE

Introduction

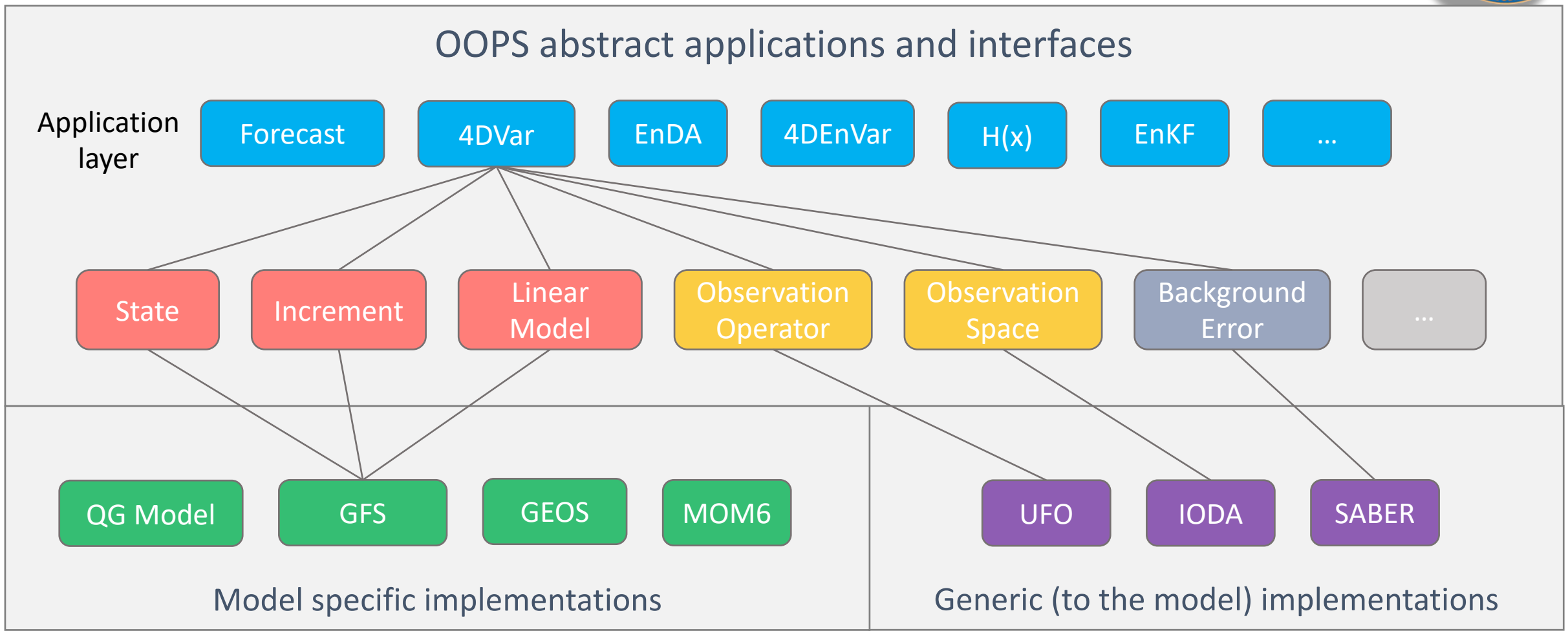
This lecture introduces the concept of the model interface in JEDI. Everything we've learnt about so far is the generic part of the code. This generic code must be implemented for a specific case. We've seen how some of it is implemented for specific observation operators. Here we learn how to implement it for specific models, whether they be atmosphere, ocean, land, space weather or otherwise.

It is not necessary to extensively understand the JEDI software to write an interface to a forecast model. Understanding needs to extend only to the interface classes associated with the forecast model and then how to implement them for some specific model.

Models being interfaced to JEDI

MODEL	TYPE	INTERFACE	OWNER
UFS / GFS	Atmosphere	fv3-jedi	NOAA-EMC
GEOS	Atmosphere	fv3-jedi	NASA-GMAO
FV3GFS GSDChem	Atmospheric chemistry	fv3-jedi	NOAA-ESRL
GEOS-AERO	Atmospheric aerosols	fv3-jedi	NASA-GMAO
FV3-SAR	Regional NWP	fv3-jedi	NOAA-EMC
FV3-SAR-CMAQ	Regional air quality	fv3-jedi	NOAA-EMC
MPAS	Atmosphere	mpas	NCAR
LFRic	Atmosphere	lfric	Met Office (UK)
Unified Model (UM)	Atmosphere	um-jedi	Met Office (UK)
MOM6	Ocean	soca	NOAA-EMC
SIS2	Sea ice	soca	NOAA-EMC
CICE6	Sea ice	soca-cice6	NOAA-EMC
NEPTUNE	Atmosphere	neptune	NRL
QG	Toy model	oops	ECMWF
Lorenz 95	Toy model	oops	ECMWF
ShallowWater	Toy model	shallow-water	NOAA-ESRL

Interface classes



Implementations don't know which applications they are part of, and the applications don't know which model is being used.

The Model Interface Classes

Class	Description
ErrorCovariance	Background error covariance model (also implemented in SABER).
Geometry	The geometry of the forecast model/background grid.
GeometryIterator	Iterator over the grid points, needed only for LETKF applications.
GetValues	Interpolation from the model space state to observation locations.
Increment	Everything associated with an increment with model variables on the model grid.
LinearModel	The tangent linear and adjoint version of the forecast model.
LinearVariableChange	Transform between an increment with one set of fields and another.
Localization	Model ensemble localization (also implemented in SABER).
Model	The actual forecast model.
ModelAuxControl	Classes for dealing with model error.
ModelAuxCovariance	
ModelAuxIncrement	
State	Everything associated with the model state.
VariableChange	Transform between a state with one set of fields and another with different fields.

Interface class – method (Model Geometry.h)

C++ Model, State, Increment etc.

```
Geometry::Geometry(const eckit::Configuration & conf, const eckit::mpi::Comm & comm) {

 dx_ = conf.getInt("dx")
 dy_ = conf.getInt("dx")

}
```

Fortran Model, State, Increment etc.

```
Geometry::Geometry(const eckit::Configuration & conf, const eckit::mpi::Comm & comm) {

  interface(keyGeom_, &conf, &comm)

}
```

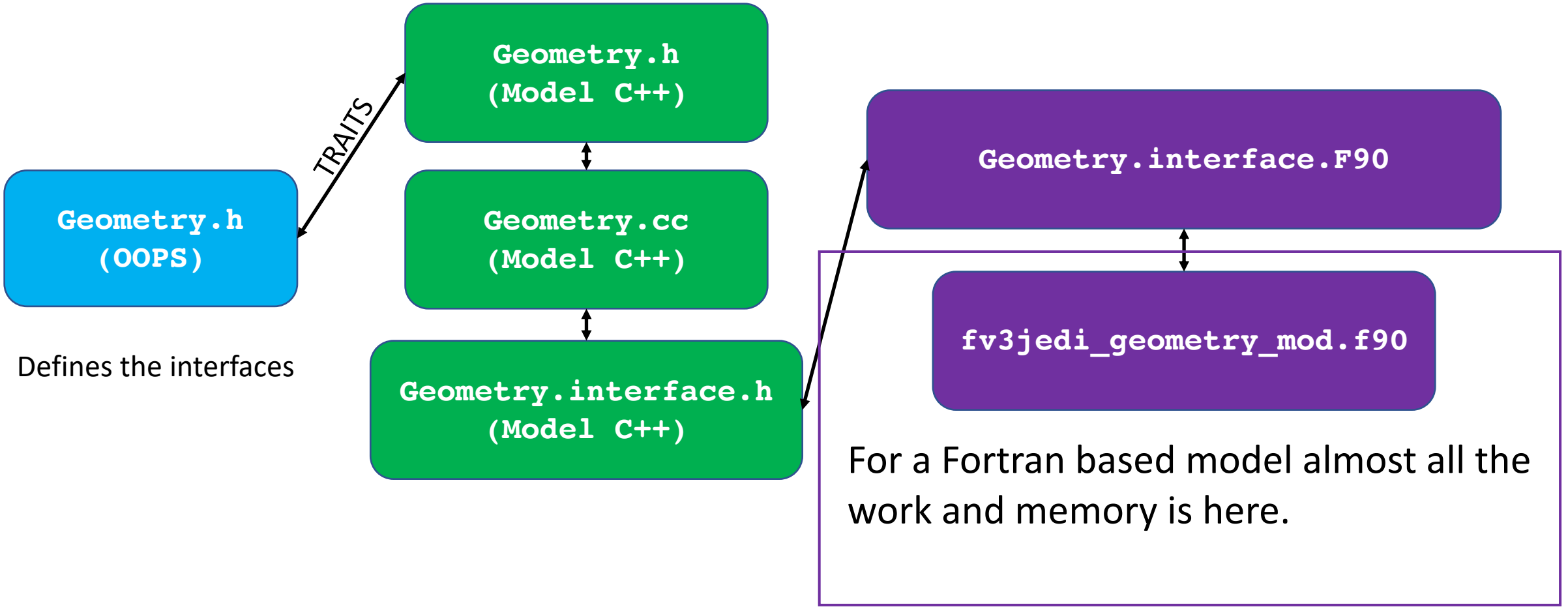
Fortran tells C++ what to pass for this object

Configuration object reference

Communicator object reference

C++ to Fortran Binding Files

Most models are written in Fortran so this is where the work is done and data stored.



Defines the interfaces

For a Fortran based model almost all the work and memory is here.

Fortran Geometry

```
type :: fv3jedi_geom
  integer :: isc, iec, jsc, jec
  integer :: npx, npy, npz, ngrid
  integer :: layout(2)
  real(kind=kind_real), allocatable, dimension(:,:) :: grid_lon, grid_lat
  contains
    procedure, public :: create
    procedure, public :: clone
    procedure, public :: delete
end type fv3jedi_geom
```

```
subroutine create(self, conf, comm)

!Arguments
class(fv3jedi_geom), target, intent(inout) :: self
type(fckit_configuration),   intent(in)    :: conf
type(fckit_mpi_comm),        intent(in)    :: comm

end subroutine create
```

```
subroutine clone(self, other)

class(fv3jedi_geom),        intent(inout) :: self
type(fv3jedi_geom), target, intent(in)    :: other

end subroutine clone
```

Fortran version of the class contains the actual data structures

Fortran code mimics the interfaces defined in the Interface class but developer working in this part of the code sees no C++.

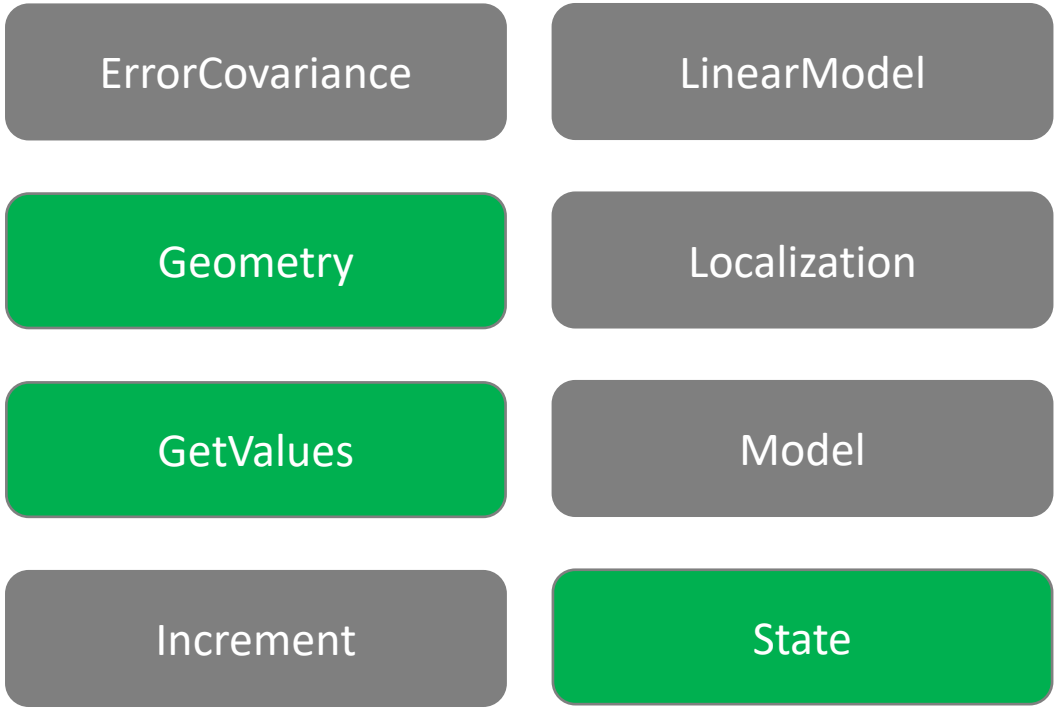
The create phase likely involves a call to the internal model routines in order to generate the grid.

Implementation leads to applications incrementally

You can also think of these classes as generic building blocks for applications:

Available applications

- 3D or ‘nomodel’ H(x)
- 3DVar / 4DEnVar
- Hybrid 3DVar / 4DEnVar
- H(x), forecast and 3DVar-FGAT
- Hybrid-4DVar

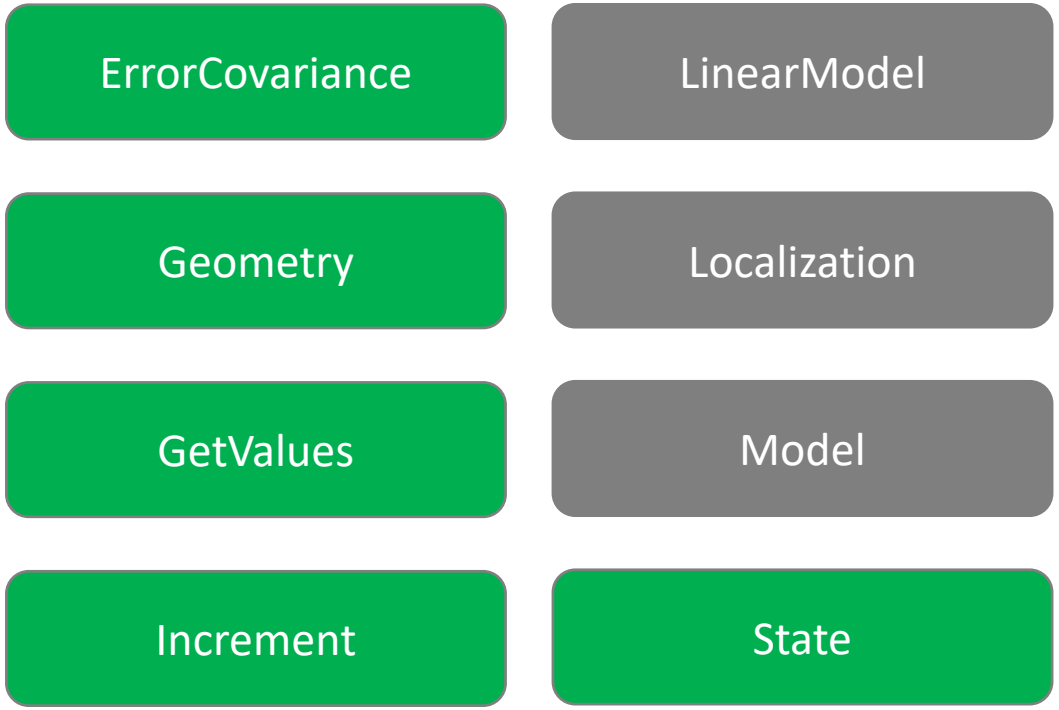


Implementation leads to applications incrementally

You can also think of these classes as generic building blocks for applications:

Available applications

- 3D or ‘nomodel’ H(x)
- 3DVar / 4DEnVar
- Hybrid 3DVar / 4DEnVar
- H(x), forecast and 3DVar-FGAT
- Hybrid-4DVar

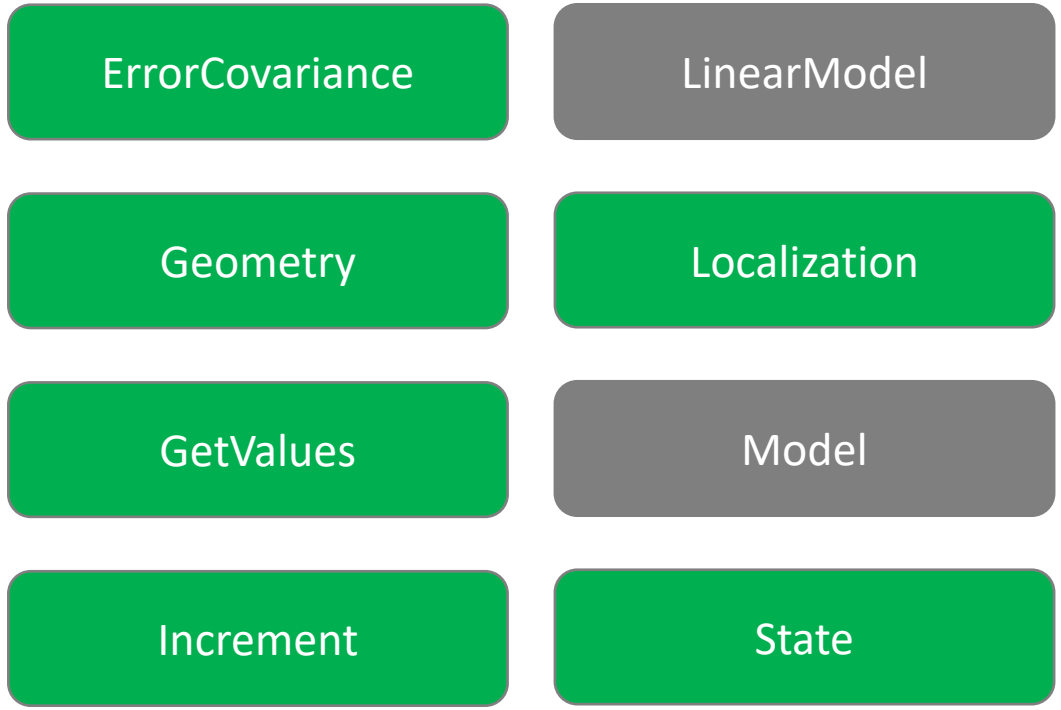


Implementation leads to applications incrementally

You can also think of these classes as generic building blocks for applications:

Available applications

- 3D or ‘nomodel’ H(x)
- 3DVar / 4DEnVar
- Hybrid 3DVar / 4DEnVar
- H(x), forecast and 3DVar-FGAT
- Hybrid-4DVar

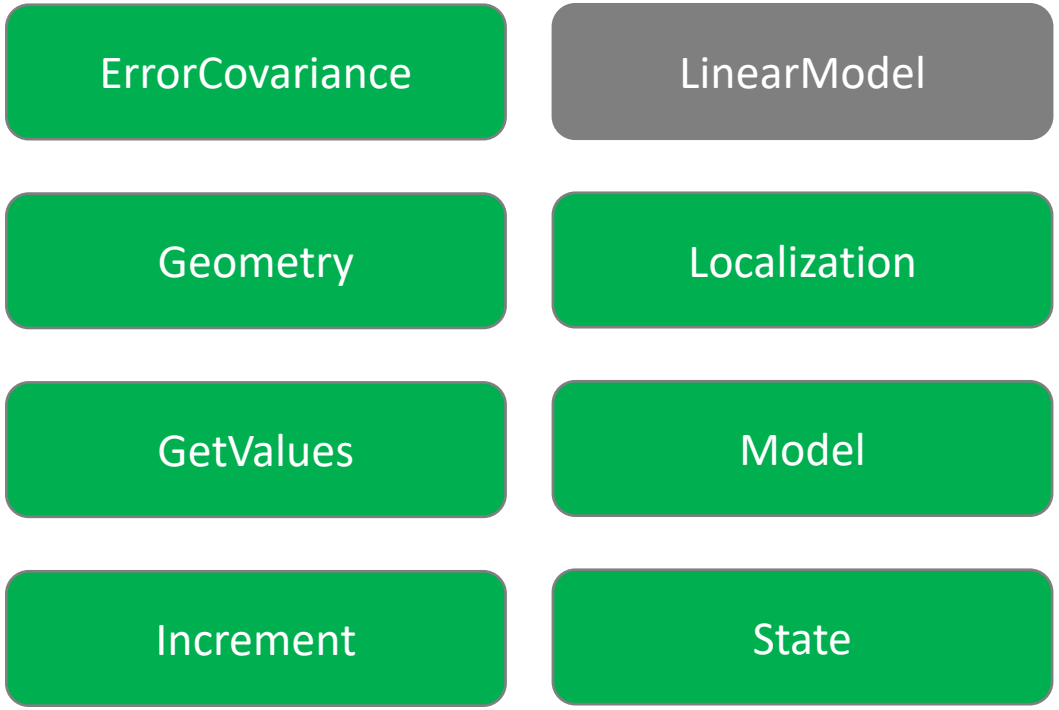


Implementation leads to applications incrementally

You can also think of these classes as generic building blocks for applications:

Available applications

- 3D or ‘nomodel’ H(x)
- 3DVar / 4DEnVar
- Hybrid 3DVar / 4DEnVar
- H(x), forecast and 3DVar-FGAT
- Hybrid-4DVar

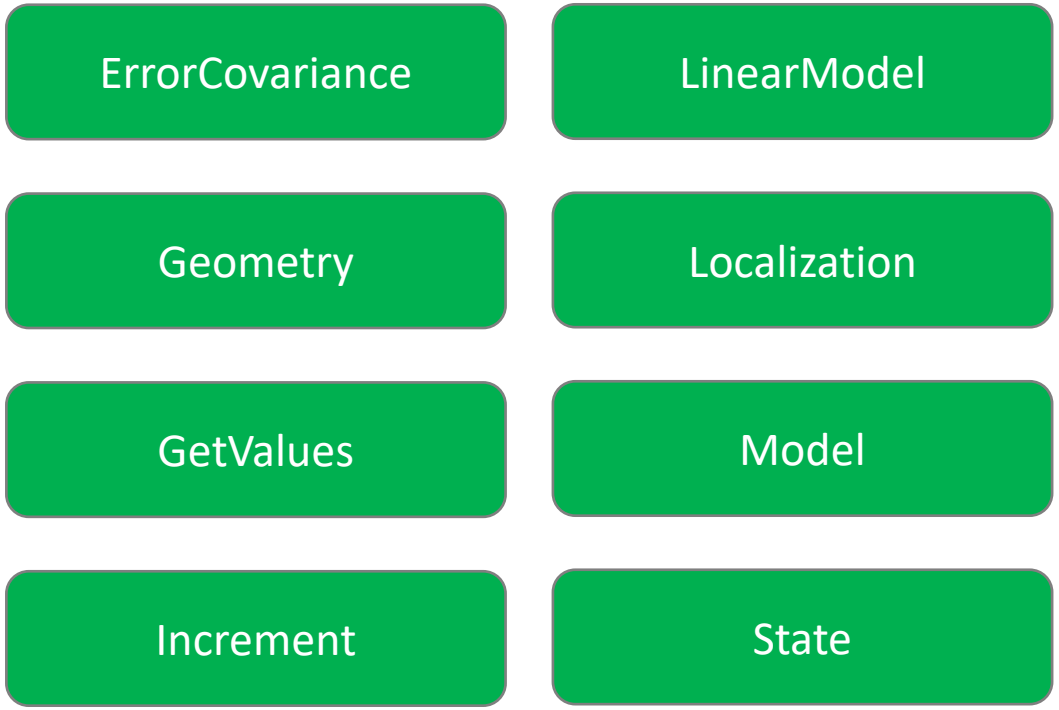


Implementation leads to applications incrementally

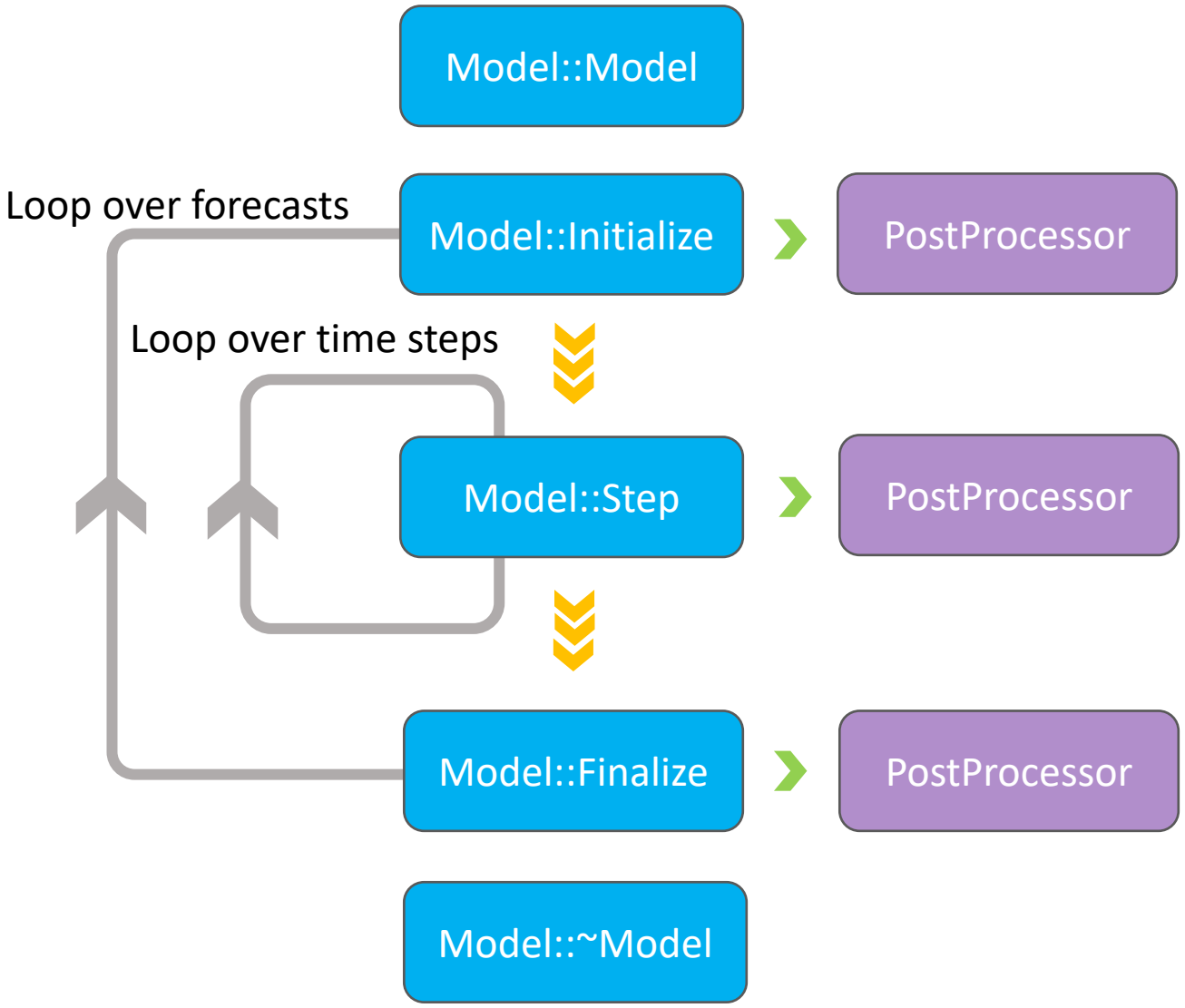
You can also think of these classes as generic building blocks for applications:

Available applications

- 3D or ‘nomodel’ H(x)
- 3DVar / 4DEnVar
- Hybrid 3DVar / 4DEnVar
- H(x), forecast and 3DVar-FGAT
- Hybrid-4DVar



Forecasts in JEDI



Model constructor and destructor are called once per applications.

Model::Initialize and Model::Finalize are called once per forecast.

Model::Step is called once per timestep per forecast.

Variable changes

Incremental hybrid-4DVar involves a number of linear and nonlinear variable transforms:

$$\frac{\partial J}{\partial \delta \mathbf{x}_0} = \mathbf{B}^{-1}\left(\delta \mathbf{x}_0 - \delta \mathbf{x}_b\right) - \sum_{k=0}^{K} K_m^{\top} \mathbf{M}_k^{\top} K_h^{\top} \mathbf{H}^{\top} \mathbf{R}_k^{-1}\left(\mathbf{d}_k - \mathbf{H} K_h \delta \mathbf{x}_k\right)$$

$$\delta \mathbf{x}_k = \mathbf{M}_{t_{k-1} \rightarrow t_k} \mathbf{M}_{t_{k-1} \rightarrow t_{k-2}} \ldots \mathbf{M}_{t_0 \rightarrow t_1} K_m \delta \mathbf{x}_0$$

$$\mathbf{d}_k = \mathbf{y}_k^o - h\left(k_h\left\{m_{t_0 \rightarrow t_k}\left[k_m\left(\mathbf{x}_0\right)\right]\right\}\right)$$

$$B = K_b D C D K_b^{\top}$$

Variable changes

Sets of variables:

Background: the variables read from the background and used to create the increment.

Analysis (increment): the variables making up the increment and the variables added to the model variables to create the analysis at the end.

Model: the variables that the model needs and what you will have in the analysis file.

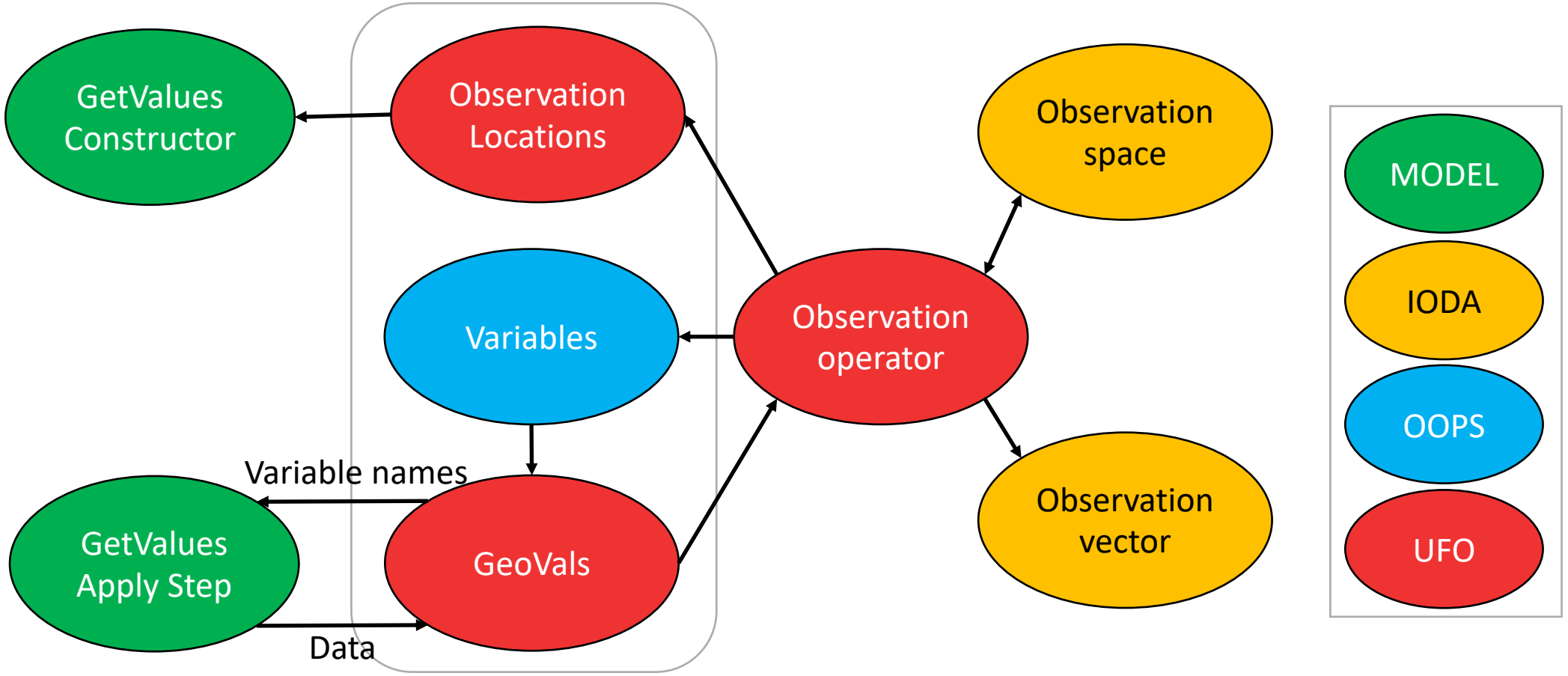
LinearModel: variables needed to drive the linearized version of the forecast model.

B matrix: the variables used in the B matrix, e.g. unbalanced stream function and velocity potential.

UFO: list of variables (delivered via the GeoVaLs) that the observation operators need.

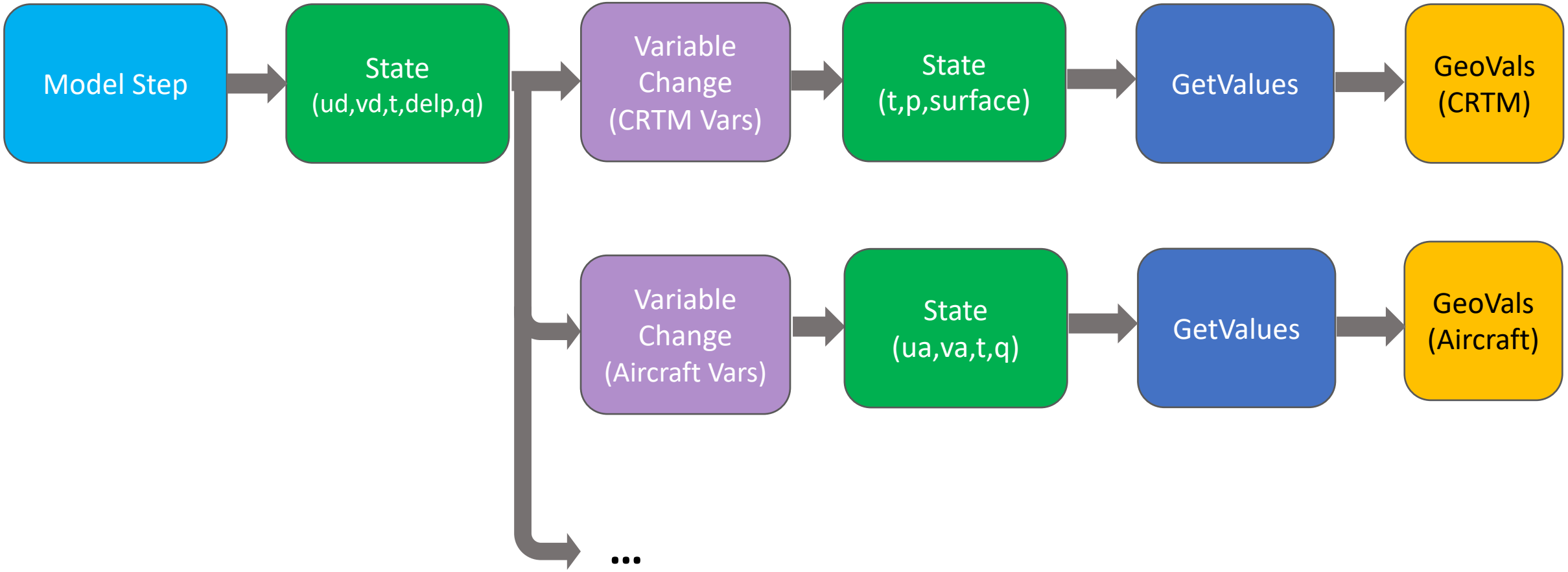
OBS–MODEL interface

- A key functionality in the model is to prepare the data for the observation operators.
- This is done through the GetValues class, which fills GeoVaLs.

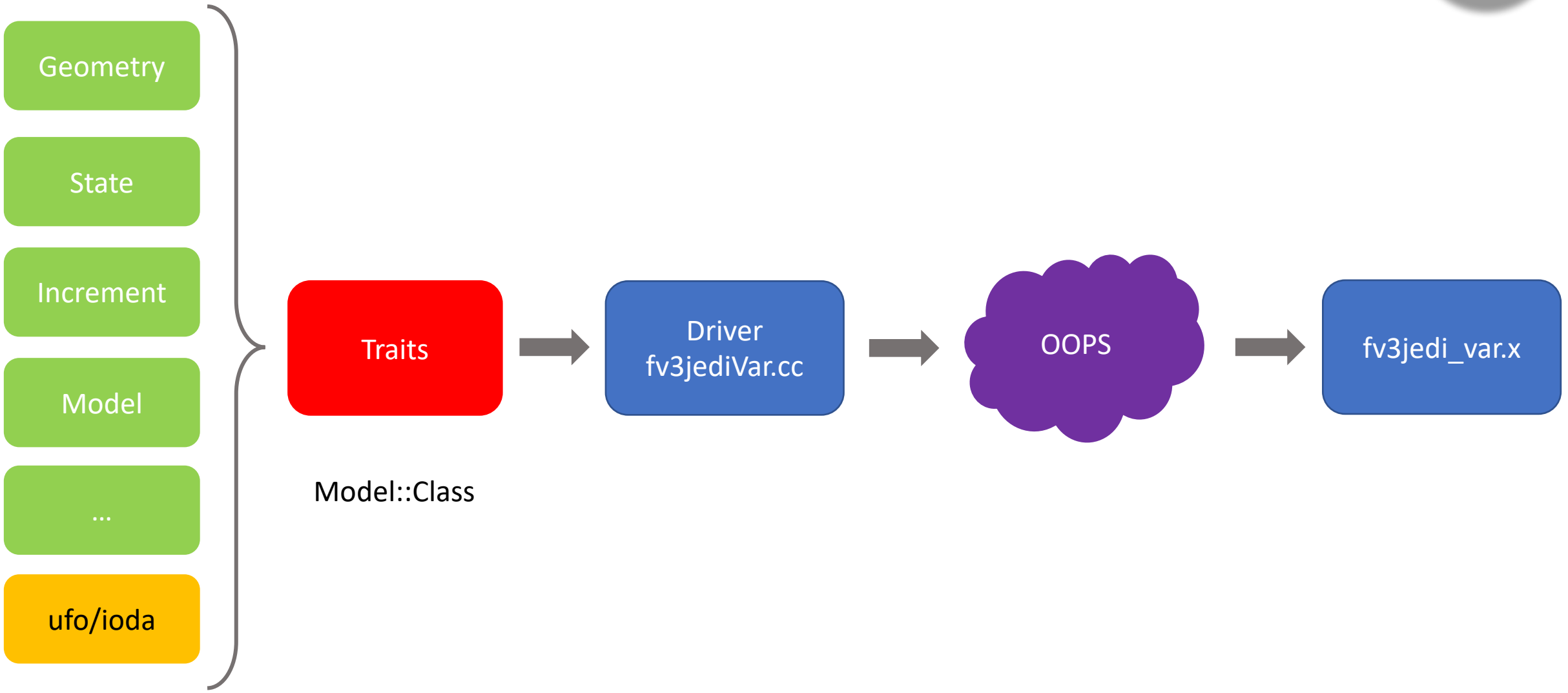


GetValues

After each model step the post processor loops over all the observation operators and fills the GeoVaLs (Geophysical Values at observation Locations). Each GeoVaLs object is stored until the model run finishes and the observation operators are called.



Building an application driver



Adding a new model to JEDI

- Follow an existing model.
- Populate the class methods and data to work for the model grid and fields that you need.
- Implement OOPS unit tests for each implemented interface class.
- Gradually build the applications, each time adding a test.

Questions?