The Joint Effort for Data assimilation Integration (JEDI)

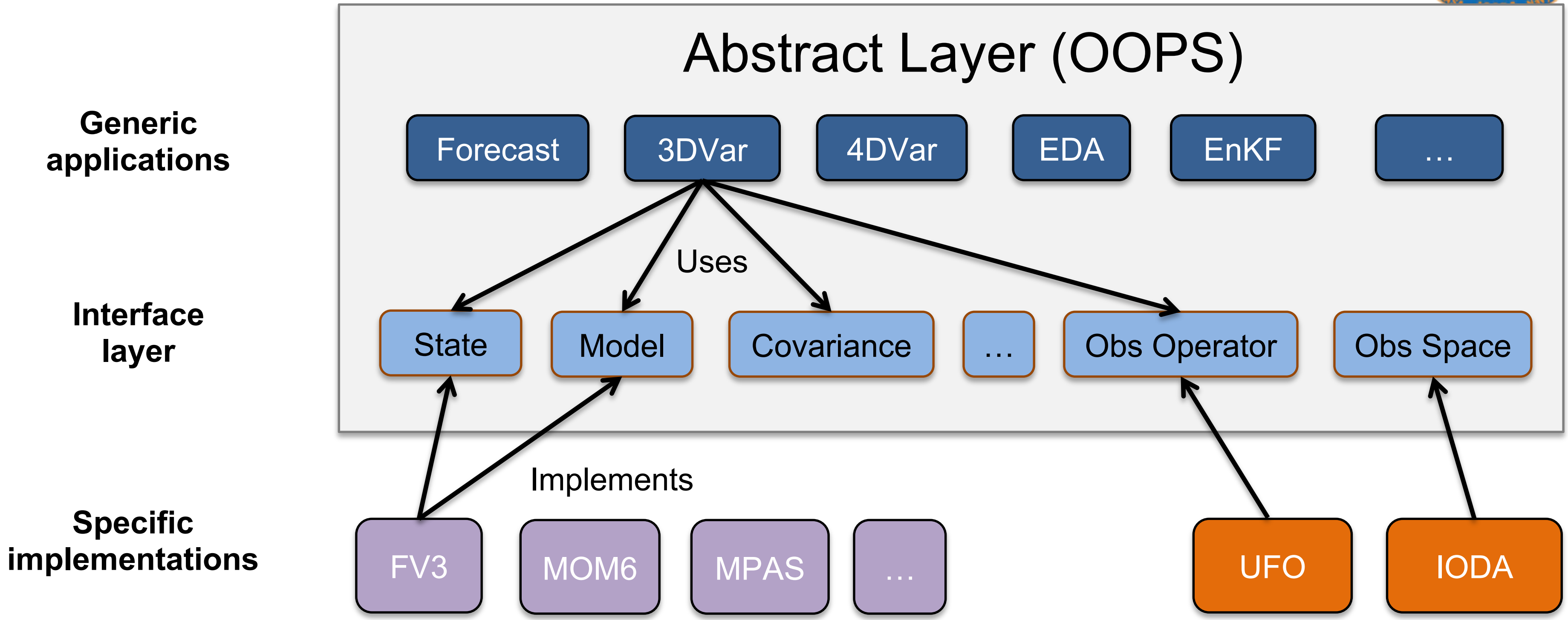
OOPS Observation Space

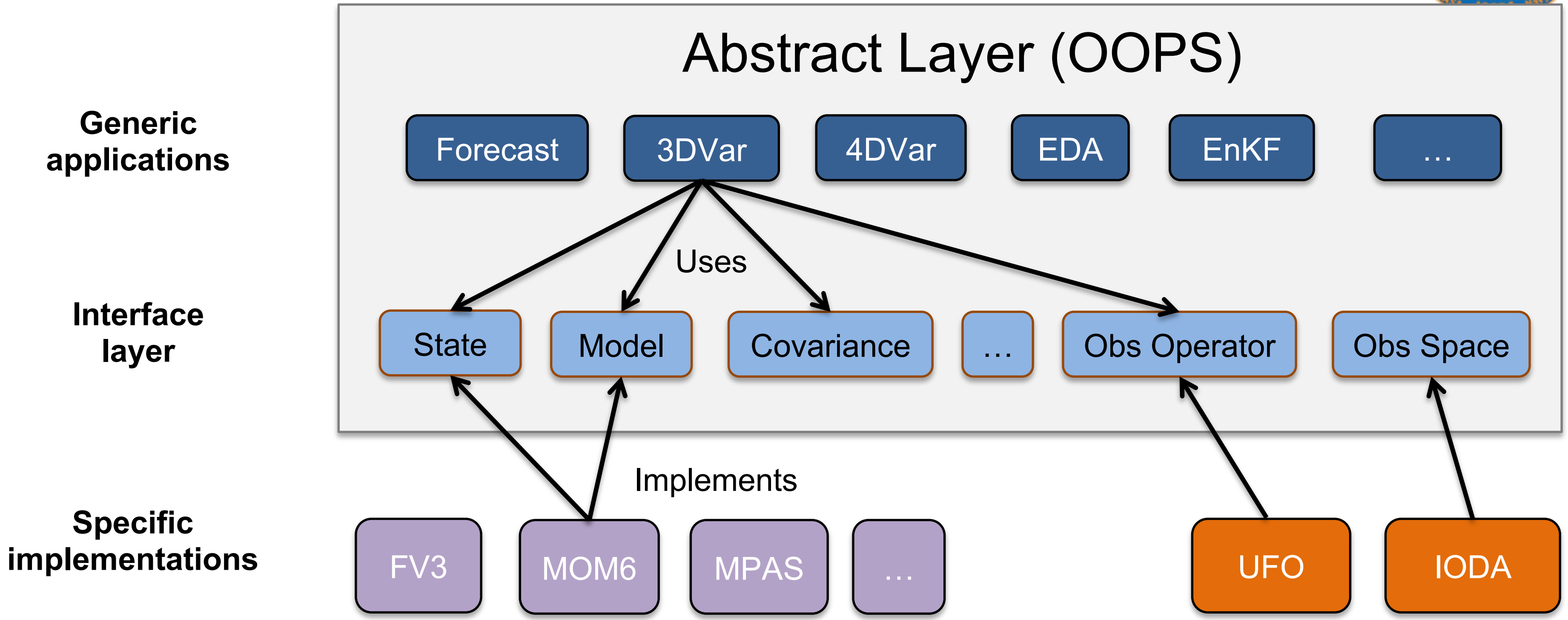
Joint Center for Satellite Data Assimilation (JCSDA)

JEDI Academy – 25-28 February 2020

OOPS Observation Space

- OOPS interfaces related to observations: what and why?
- Dataflow for the Observer postprocessor
- Using different ObsOperators for different observation types
- Configuring Observations


Abstract Layer (OOPS)
Generic applications
Forecast
3DVar
4DVar
EDA
EnKF
…
Uses
Interface layer
State
Model
Covariance
…
Obs Operator
Obs Space
Implements
Specific implementations
FV3
MOM6
MPAS
…
UFO
IODA



Abstract Layer (OOPS)
Generic applications
Forecast
3DVar
4DVar
EDA
EnKF
…
Uses
Interface layer
State
Model
Covariance
…
Obs Operator
Obs Space
Implements
Specific implementations
FV3
MOM6
MPAS
…
UFO
IODA

OOPS interfaces related to obs: Data assimilation perspective

$$J(\Delta x) = \frac{1}{2}\Delta x^T \mathbf{B}^{-1}\Delta x + \frac{1}{2}(y_o - H(x_b) - \mathbf{H}\Delta x)^T \mathbf{R}^{-1}(y_o - H(x_b) - \mathbf{H}\Delta x)$$

or

$$\Delta x_a = \mathbf{BH}^{\mathrm{T}}\left(\mathbf{HBH}^{\mathrm{T}} + \mathbf{R}\right)^{-1}(y_o - H(x_b))$$

Observations: vector in the observation space (for example, holding observation values)

OOPS interfaces related to obs: Data assimilation perspective

$$J(\Delta x) = \frac{1}{2}\Delta x^T \mathbf{B}^{-1}\Delta x + \frac{1}{2}(y_o - H(x_b) - \mathbf{H}\Delta x)^T \mathbf{R}^{-1}(y_o - H(x_b) - \mathbf{H}\Delta x)$$

or

$$\Delta x_a = \mathbf{B}\mathbf{H}^{\mathbf{T}}\left(\mathbf{H}\mathbf{B}\mathbf{H}^{\mathbf{T}} + \mathbf{R}\right)^{-1}(y_o - H(x_b))$$

Observations: vector in the observation space (for example, holding observation values or model simulated observation equivalents)

OOPS interfaces related to obs: Data assimilation perspective

$$J(\Delta x) = \frac{1}{2}\Delta x^T \mathbf{B}^{-1}\Delta x + \frac{1}{2}(y_o - H(x_b) - \mathbf{H}\Delta x)^T \mathbf{R}^{-1}(y_o - H(x_b) - \mathbf{H}\Delta x)$$

or

$$\Delta x_a = \mathbf{B}\mathbf{H}^{\mathbf{T}}\left(\mathbf{H}\mathbf{B}\mathbf{H}^{\mathbf{T}} + \mathbf{R}\right)^{-1}(y_o - H(x_b))$$

ObsErrorCovariance: matrix representing observation error covariances

OOPS interfaces related to obs: Data assimilation perspective

$$J(\Delta x) = \frac{1}{2}\Delta x^T \mathbf{B}^{-1}\Delta x + \frac{1}{2}(y_o - H(x_b) - \mathbf{H}\Delta x)^T \mathbf{R}^{-1}(y_o - H(x_b) - \mathbf{H}\Delta x)$$

or

$$\Delta x_a = \mathbf{BH}^\mathbf{T}\left(\mathbf{HBH}^\mathbf{T} + \mathbf{R}\right)^{-1}(y_o - H(x_b))$$

ObsOperator: observation operator for simulating observation given state

OOPS interfaces related to obs: Data assimilation perspective

$$J(\Delta x) = \frac{1}{2}\Delta x^T \mathbf{B}^{-1}\Delta x + \frac{1}{2}(y_o - H(x_b) - \mathbf{H}\Delta x)^T \mathbf{R}^{-1}(y_o - H(x_b) - \mathbf{H}\Delta x)$$

or

$$\Delta x_a = \mathbf{B}\mathbf{H}^{\mathrm{T}}\left(\mathbf{H}\mathbf{B}\mathbf{H}^{\mathrm{T}} + \mathbf{R}\right)^{-1}(y_o - H(x_b))$$

ObsOperator: observation operator for simulating observation given state

LinearObsOperator: tangent-linear and adjoint of the observation operator

OOPS interfaces related to obs

ObsVector (Observations) — Observations related classes

ObsOperator
LinearObsOperator — ObsOperator related classes

ObsErrorCovariance — ObsError

OOPS interfaces related to obs: Observations processing perspective

- Need to have access to observation-related data (observation values and metadata), efficient I/O, distribution across processors, etc: ObsSpace
- Quality control is an important aspect for real-world data assimilation: ObsFilters
- Bias correction is also important: ObsAuxControl, ObsAuxIncrement, ObsAuxCovariance

OOPS interfaces related to obs

```
ObservationSpace
ObsVector (Observations)
```

Observations related classes (IODA)

```
ObsOperator
LinearObsOperator
```

ObsOperator related classes (UFO)

```
ObsFilter
```

QC related classes (UFO)

```
ObsAuxControl
ObsAuxIncrement
ObsAuxCovariance
```

Bias correction related classes (UFO)

```
ObsErrorCovariance
```

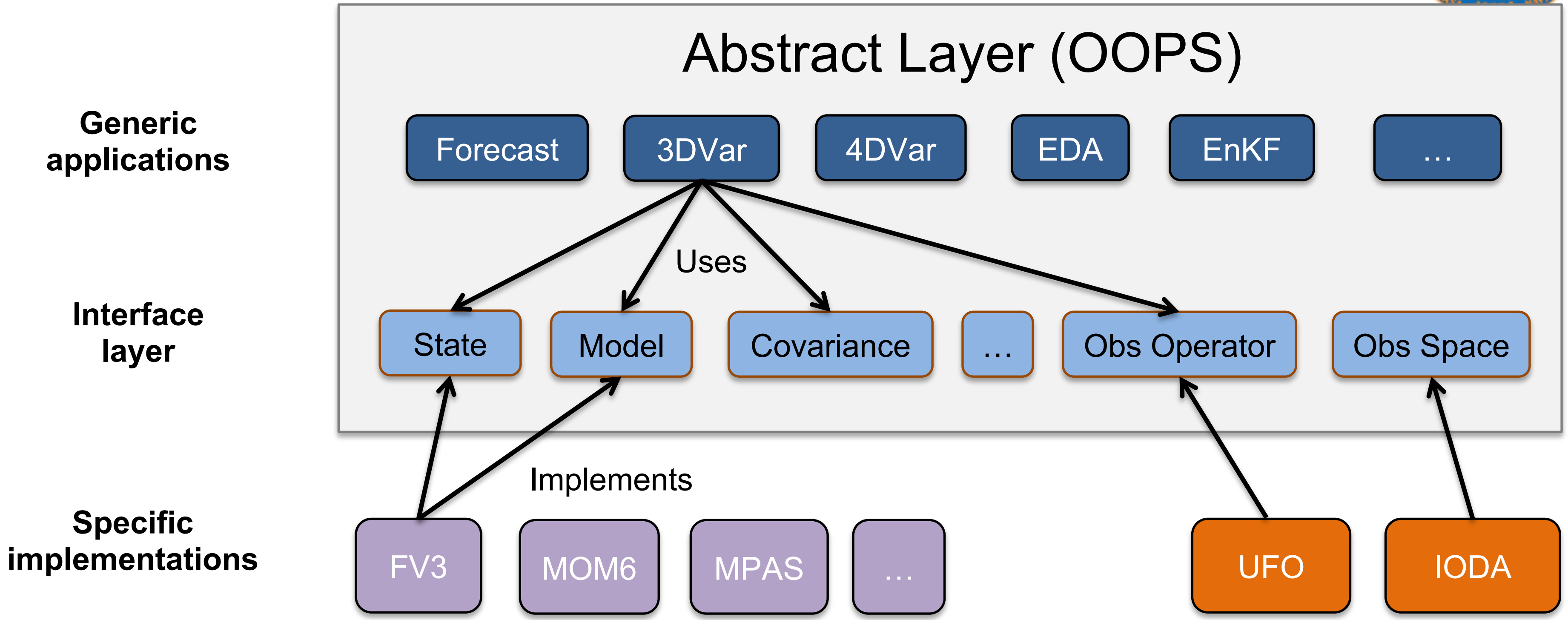
ObsError, for now using diagonal R (OOPS)

Using different ObsOperators

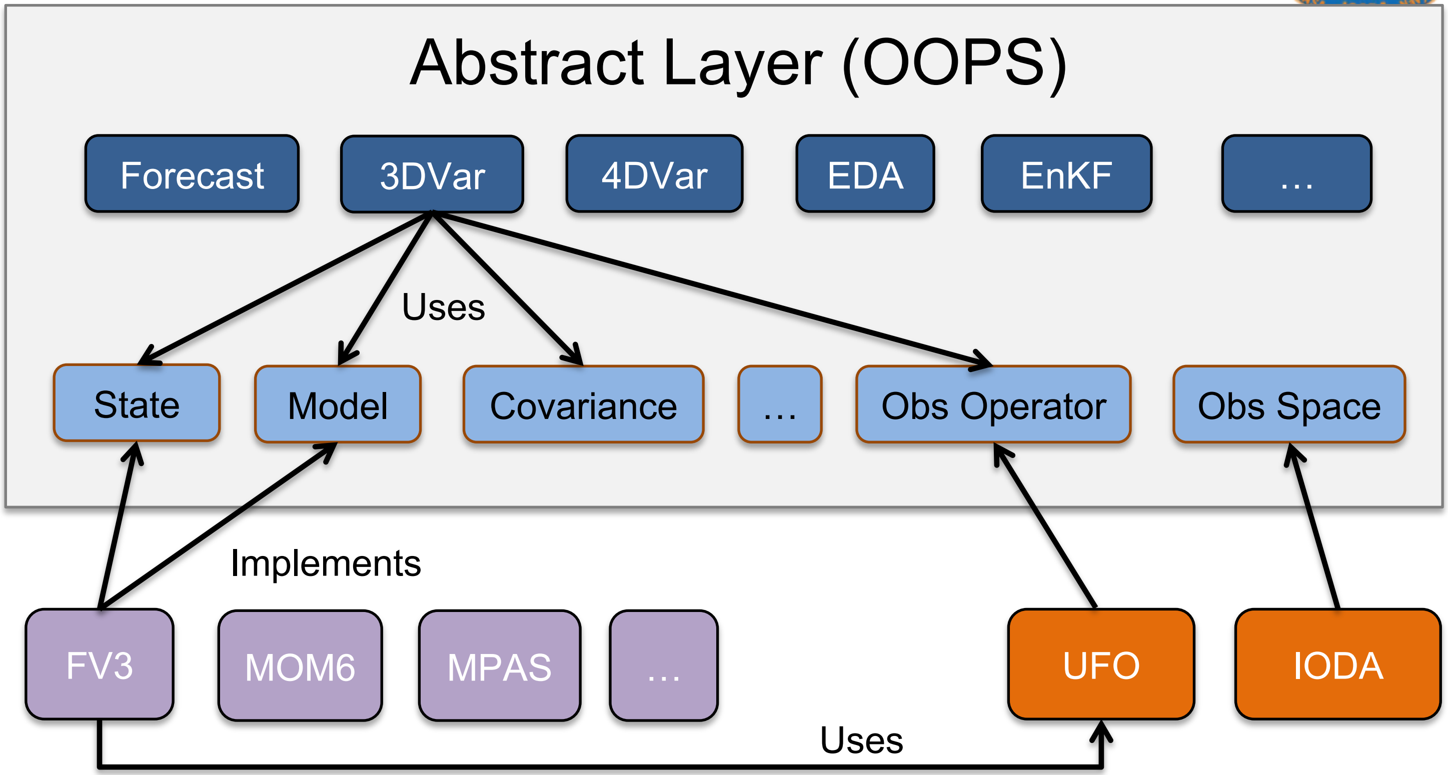
- One ObsOperator only processes one ”observation type” (e.g., there are separate ObsOperators for radiance and radiosonde)
- To assimilate different observation types, we use multiple ObsOperator’s and ObsSpace’s.
- This is handled in oops (base):
  ObsSpaces class is a vector (array) of ObsSpace

Observations, Departures, ObsVector

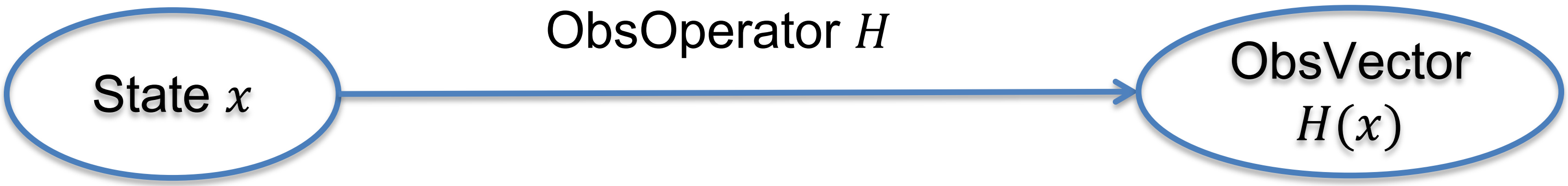
- Observations and Departures are OOPS classes that contain vector (array) of ObsVectors for all ObsSpaces (making it a long vector size of all observations).
- The algorithms in OOPS use Observations and Departures.
- ObsOperator in UFO use ObsVector, and know nothing about Observations/Departures or algorithms (separation of concerns).


Abstract Layer (OOPS)
Generic applications
Forecast
3DVar
4DVar
EDA
EnKF
…
Uses
Interface layer
State
Model
Covariance
…
Obs Operator
Obs Space
Implements
Specific implementations
FV3
MOM6
MPAS
…
UFO
IODA

Generic applications
Interface layer
Specific implementations


Abstract Layer (OOPS)
Forecast
3DVar
4DVar
EDA
EnKF
…
Uses
State
Model
Covariance
…
Obs Operator
Obs Space
Implements
FV3
MOM6
MPAS
…
UFO
IODA
Uses

Interface between Observations and Model

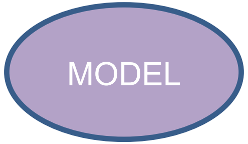


Observation operator computes model equivalent in the observation space.

Possible (obvious) interface:

```
ObsOperator::simulateObs(const State &,
                         ObsVector &)
```

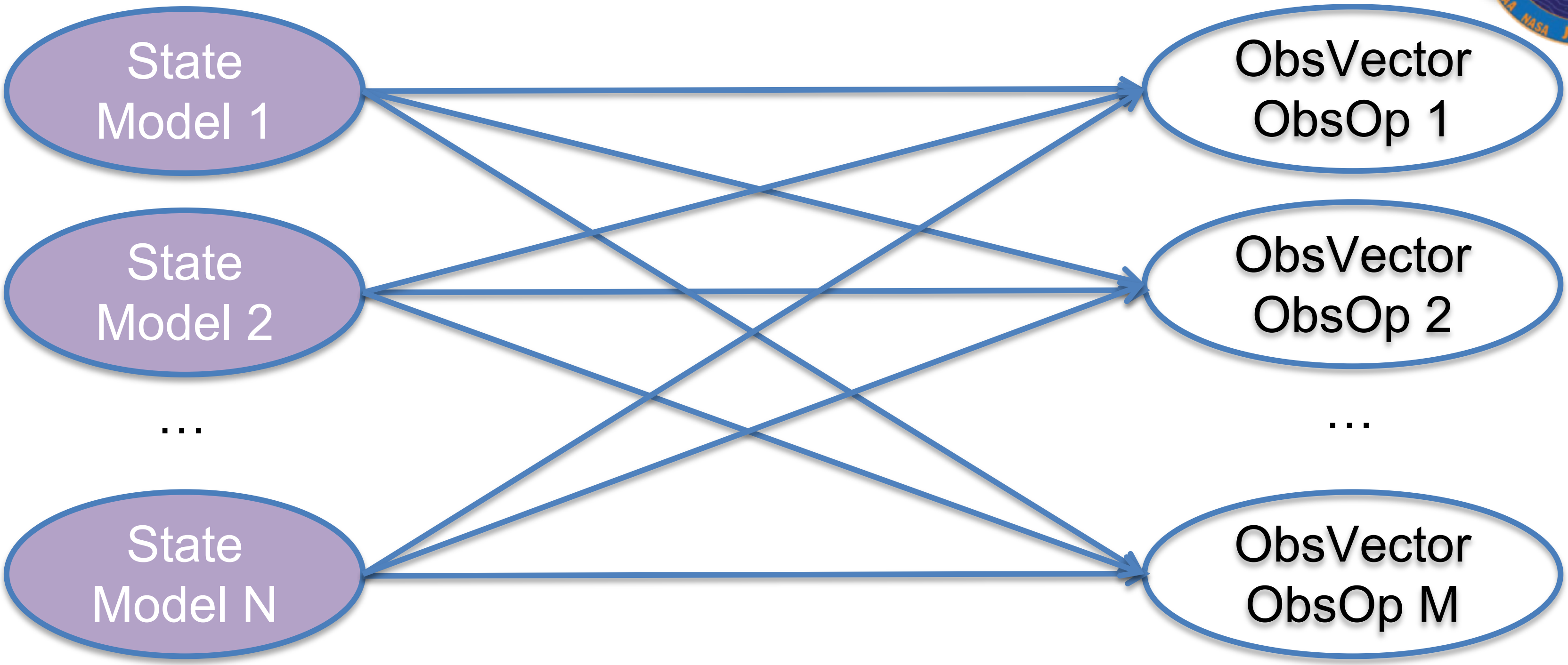
Interface between Observations and Model



With this interface, ObsOperator becomes model-specific.

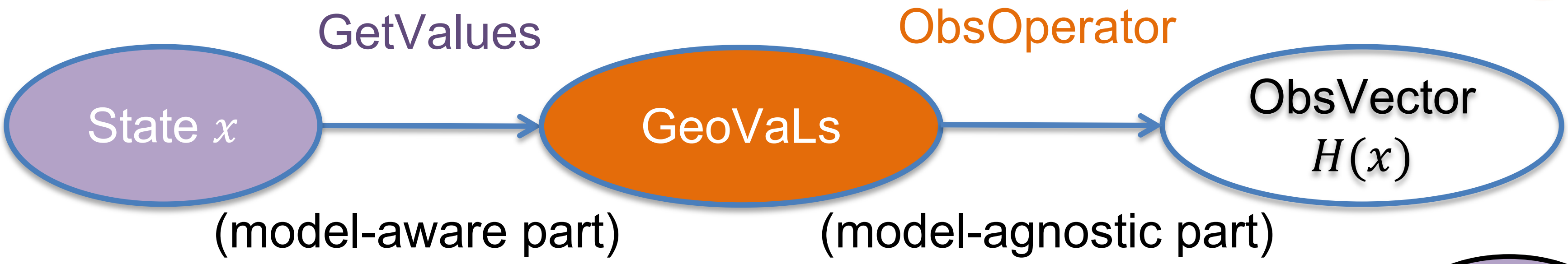
One of the JEDI goals:
Share observation operators between JCSDA partners and reduce duplication of work

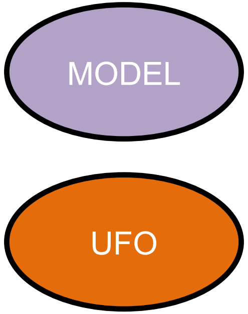
Interface between Observations and Model



With this design, each model would have to implement all observation operators it needs: duplication of work

Interface between Observations and Model

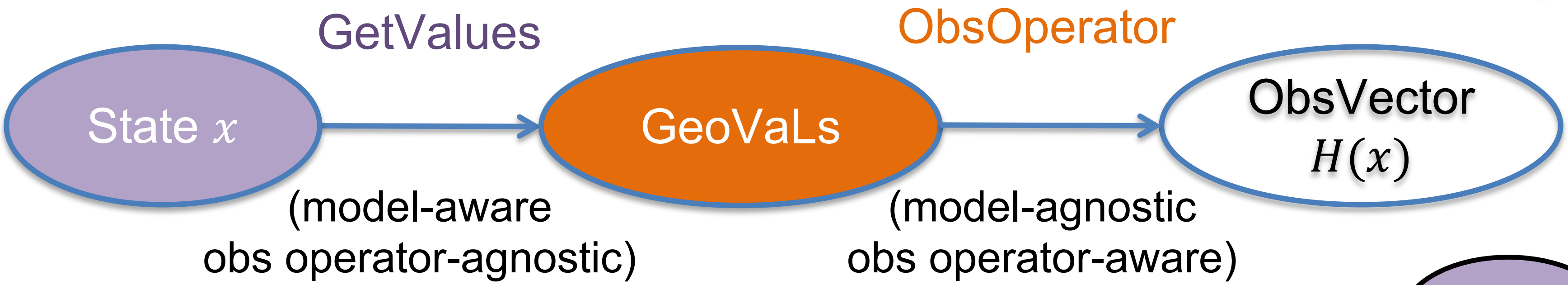


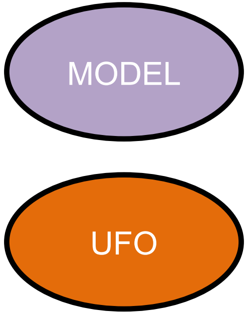


Each model implements getValues (interpolation of requested variables).

Observation operators are then independent of the model and can easily be shared, exchanged, compared

Interface between Observations and Model

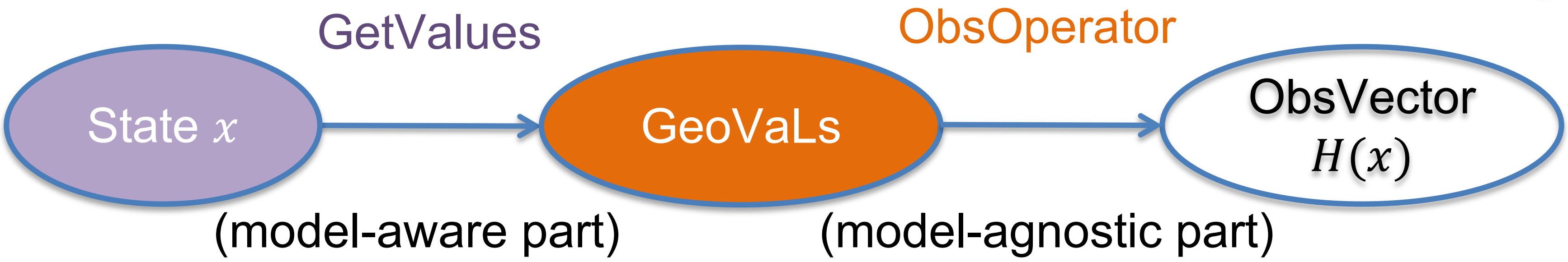




Model (or grid)-aware part: horizontal interpolation of state variables that ObsOperator needs to compute $H(x)$.

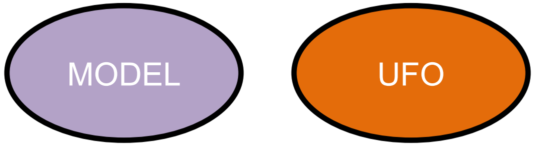
Model-agnostic part: everything that ObsOperator needs to do after getting model fields interpolated to observation location.

Interface between Observations and Model

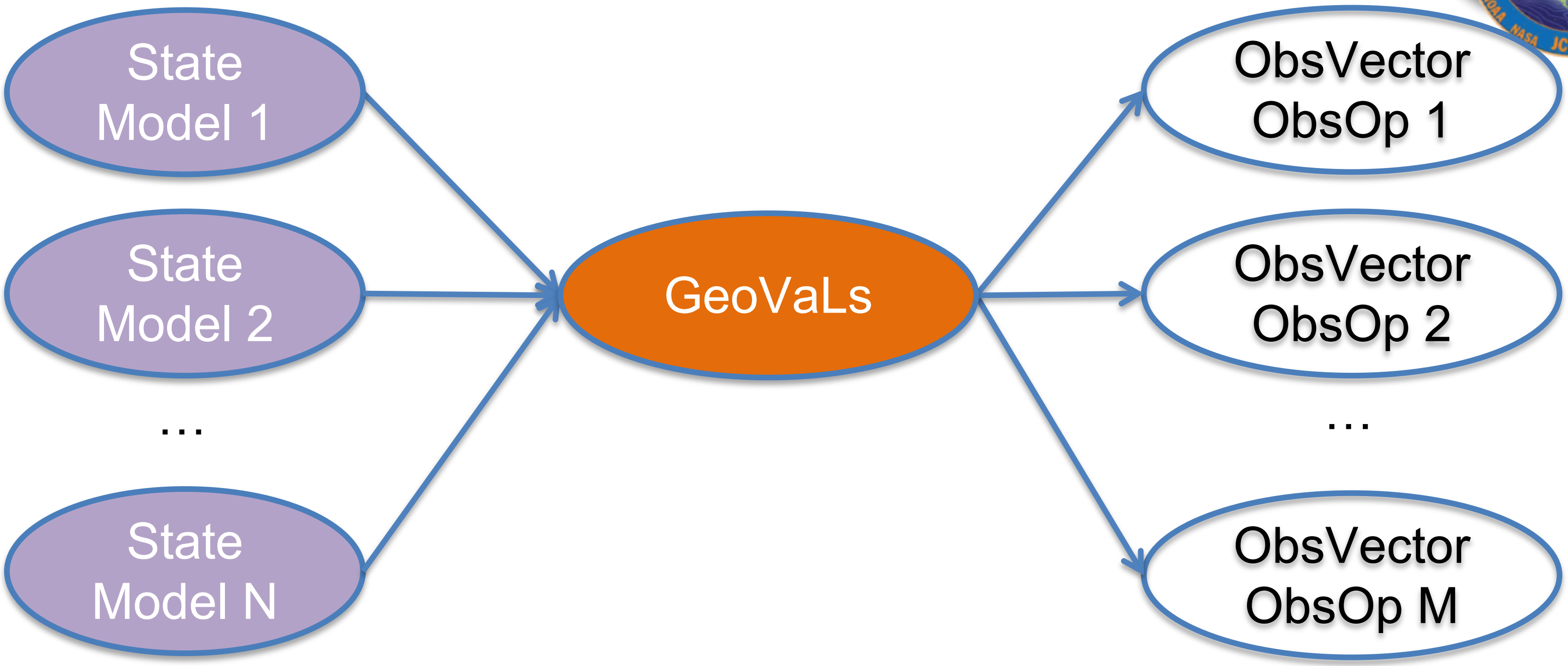


Interfaces:

```
GetValues::fillGeoVaLs(const State &, const Locations &,
                       const Variables &, GeoVaLs &)
ObsOperator::simulateObs(const GeoVaLs &,
                         ObsVector &)
```



Interface between Observations and Model



With this design, each model only has to implement GetValues, and the observation operators can be shared by many models.

OOPS interfaces related to obs

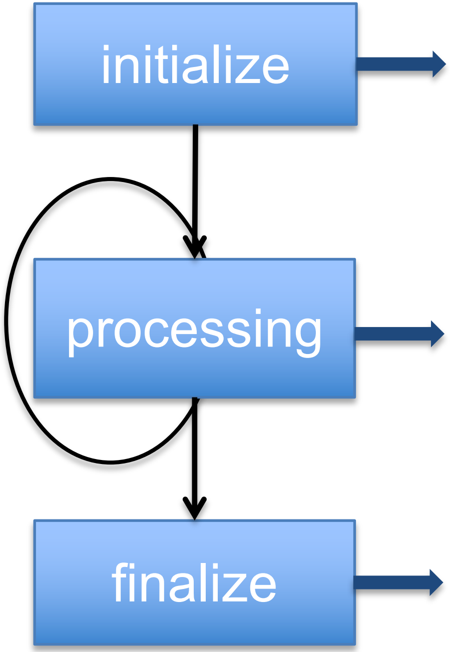
```
ObservationSpace
ObsVector

GeoVaLs
Locations
ObsOperator
LinearObsOperator
ObsFilter
ObsAuxControl
ObsAuxIncrement
ObsAuxCovariance

ObsErrorCovariance
```

- ObservationSpace, ObsVector: Observations related classes (IODA)
- GeoVaLs, Locations, ObsOperator, LinearObsOperator: ObsOperator related classes (UFO)
- ObsFilter: QC related classes (UFO)
- ObsAuxControl, ObsAuxIncrement, ObsAuxCovariance: Bias correction related classes (UFO)
- ObsErrorCovariance: ObsError, for now using diagonal R (OOPS)

Observer postprocessor



- Setup variables to be requested from the model (everything that is needed for ObsOperator, ObsBias and ObsFilters)
- Allocate GeoVaLs for the full assimilation window

- Fill in GeoVaLs for the obs within the current time window

- Run all Prior Filters
- Calculate H(x)
- Run all Posterior Filters

Observations section of yaml file

```
ObsTypes:
- ObsSpace:              # required
  ObsOperator:           # required
  Filters:
  Covariance:            # required when doing DA
  ObsBias:
  ObsBiasCovariance:
- ObsSpace:
  ObsOperator:
  Filters:
  Covariance:
```