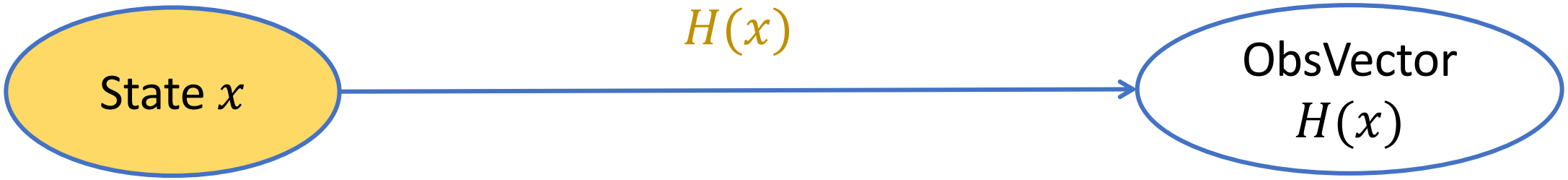
# Interface between Observations and Model

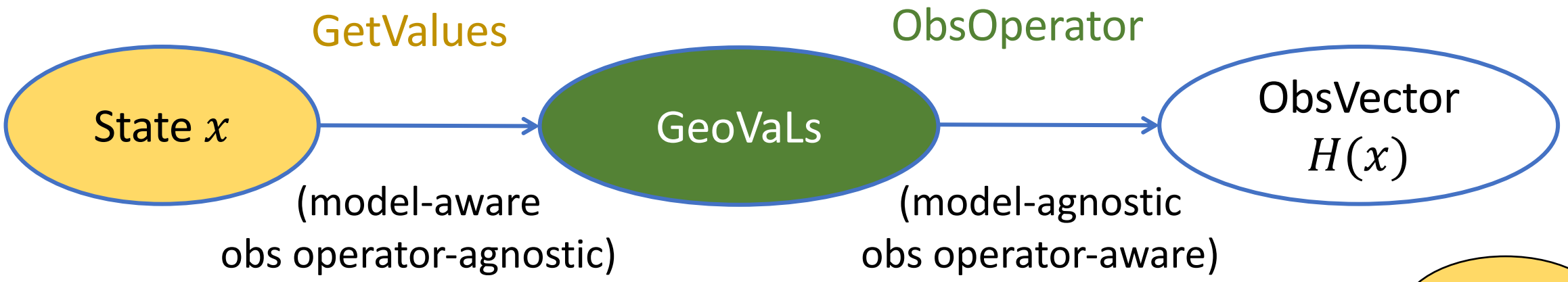



With this interface, ObsOperator would be model-specific.

MODEL

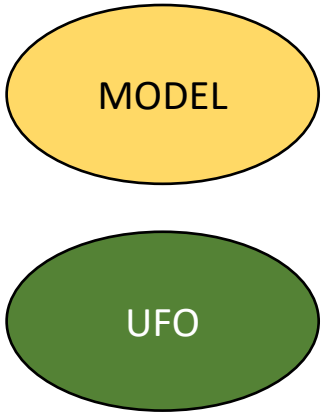
One of the JEDI goals:
Share observation operators between JCSDA partners and reduce duplication of work

# Interface between Observations and Model

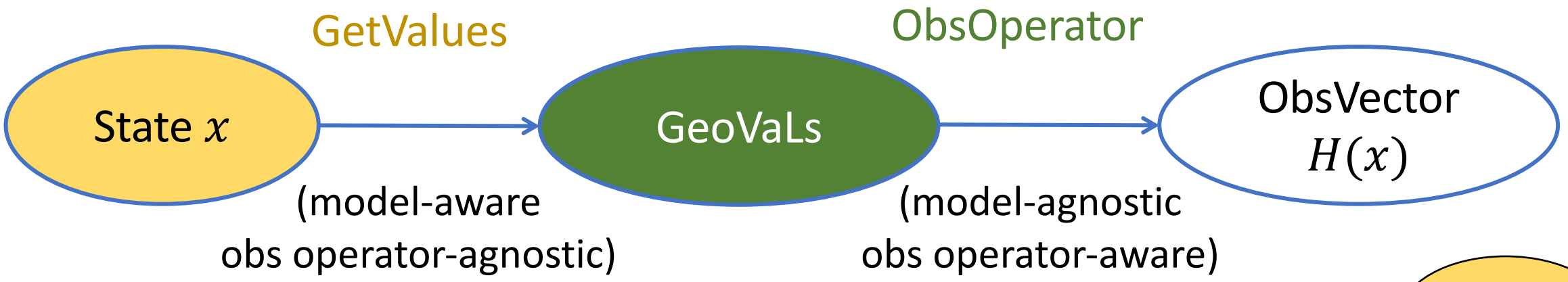



Each model implements GetValues (interpolation of requested variables).

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

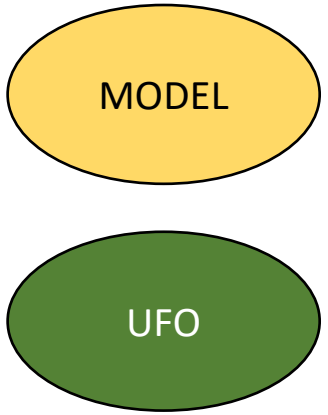


# Interface between Observations and Model

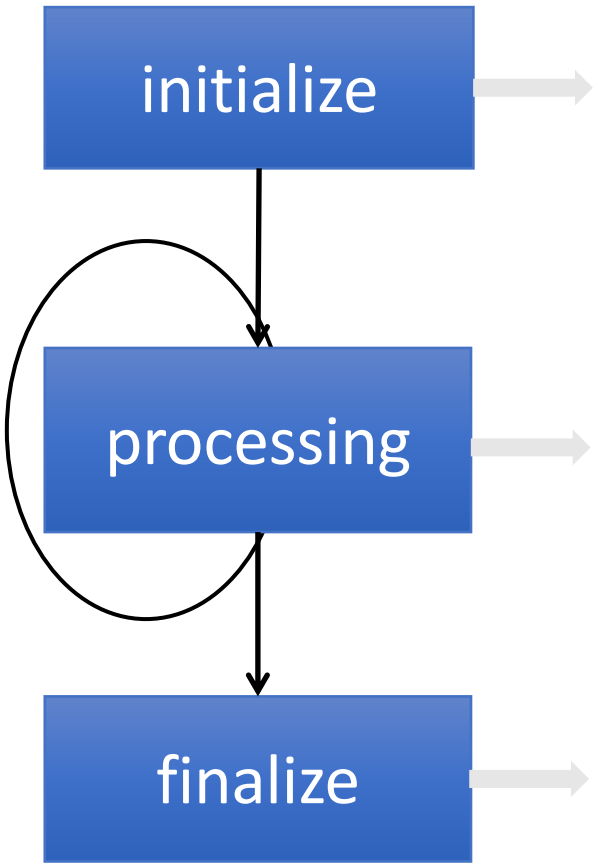



Model (or grid)-aware part: interpolation of state variables that ObsOperator needs to compute $H(x)$.

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



# Observer postprocessor

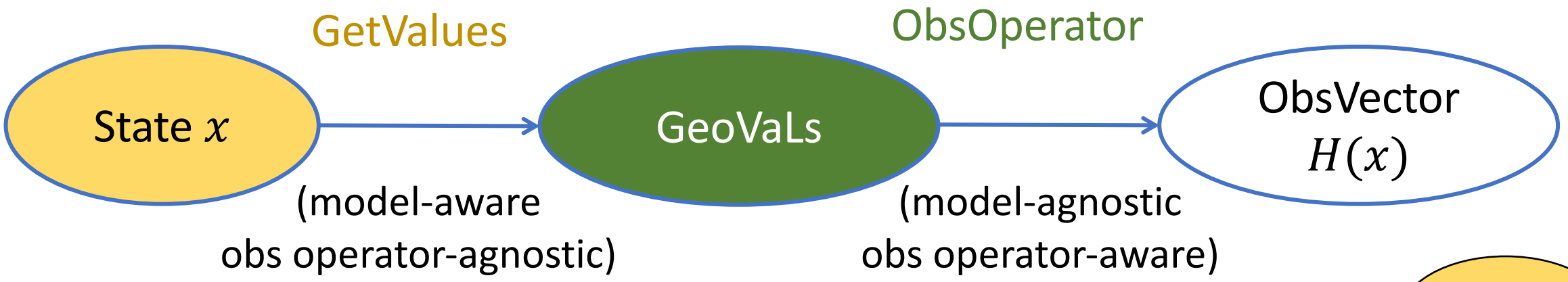



- Setup variables to be requested from the model
- Allocate GeoVaLs for the full assimilation window

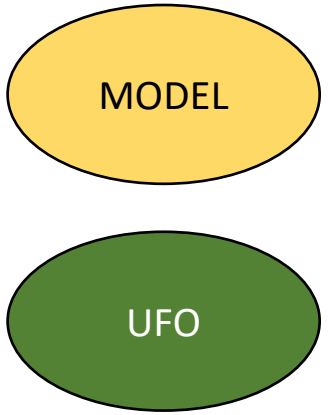
- Fill in GeoVaLs for the obs within the current time window (t1, t2]

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

# Interface between Observations and Model




Pull request oops#514 separates GetValues into a class (instead of method on State)




This will require refactoring changes in all models.

# Why change interfaces?

- Clear separation from State and Increment classes allows to have several GetValues classes in a model.
- In the [near] future different GetValues classes can be used with different ObsOperators. See branch feature/observerfactory in oops for implementation in oops and use in the toy models (no examples of using different GetValues yet).

# Why change interfaces? Some benefits:

- CRTM operator can use GetValues that work with SimpleLocations (lat/lon/time) or with SlantPathLocations. Different GetValues would be used for those different Locations.
- There could be GetValues that does vertical interpolation:
  - HorizonatlInterpGetValues for SimpleLocations + AtmVertInterp ObsOperator
  - HorVertInterpGetValues for 4DLocations +  Identity ObsOperator
- One could implement and use GetValues that do time interpolation or time accumulation.

# GetValues: new interfaces

```
class GetValues {
/// Constructor (called once per H(x) computation through window)
  GetValues(const Geometry_ &, const Locations_ & all);
/// Get state values at observation locations (called at every timestep)
/// replaces State::getValues(const Locations_ & subset,
///                           const Variables &, GeoVaLs & all)
  void compute(const State_ & current, const util::DateTime & t1,
               const util::DateTime & t2, GeoVaLs_ & all) const;
}
```

# Differences in interfaces

```
State::getValues(const Locations_ & subset_t1_t2, const Variables &,
                 GeoVaLs & all) const;

GetValues::compute(const State_ & current, const util::DateTime & t1,
                   const util::DateTime & t2, GeoVaLs_ & all) const;
```

- Variables no longer passed to GetValues (GeoVaLs has information on Variables)
- Passing t1, t2 instead of Locations(t1, t2): GetValues class has information on all Locations (from the constructor) and can find all relevant locations (t1, t2]

# LinearGetValues: new interface

```
class LinearGetValues {
/// Constructor (called once per H(x) computation through window)
  LinearGetValues(const Geometry_ &, const Locations_ & all);
/// Sets trajectory for current subwindow (called at every timestep)
/// Replaces State::getValues(const Locations_ & subset, const Variables &,
///                           GeoVaLs_ & all, InterpolatorTraj_ &);
  void setTrajectory(const State_ & current, const util::DateTime & t1,
                     const util::DateTime & t2, GeoVaLs_ & all);
/// Replaces Increment::getValuesTL & Increment::getValuesAD (called at every timestep)
  void computeTL(const Increment_ & current, const util::DateTime & t1,
                 const util::DateTime & t2, GeoVaLs_ & all) const;
  void computeAD(Increment_ & current, const util::DateTime & t1,
                 const util::DateTime & t2, const GeoVaLs_ & all) const;
}
```

# Differences in interfaces

```
State::getValues(const Locations_ & subset, const Variables &,
                 GeoVaLs_ & all, InterpolatorTraj_ &);
LinearGetValues::setTrajectory(const State_ & current,
                 const util::DateTime & t1, const util::DateTime & t2,
                 GeoVaLs_ & all);
```

- InterpolatorTraj class is removed – trajectory can now be saved in LinearGetValues instead
- Previously InterpolatorTraj was created for each (t1, t2] subwindow, now LinearGetValues is created for the whole window.