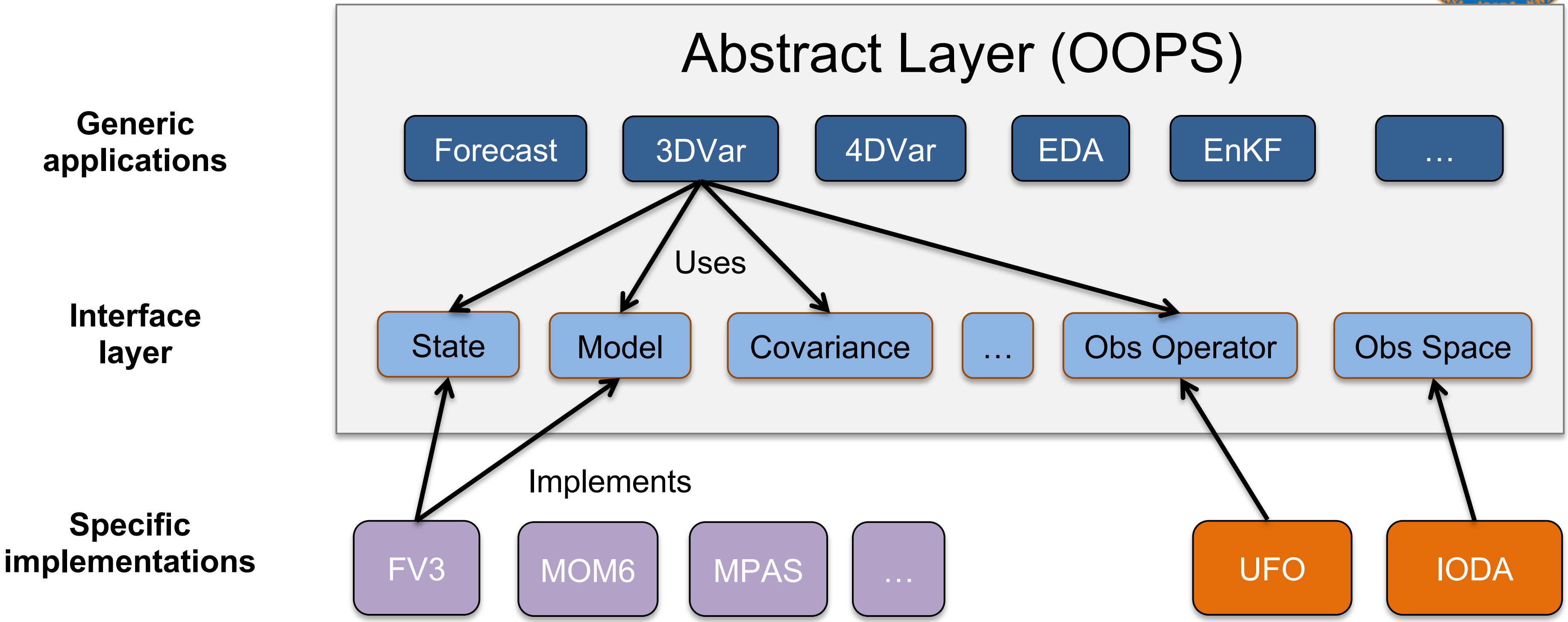
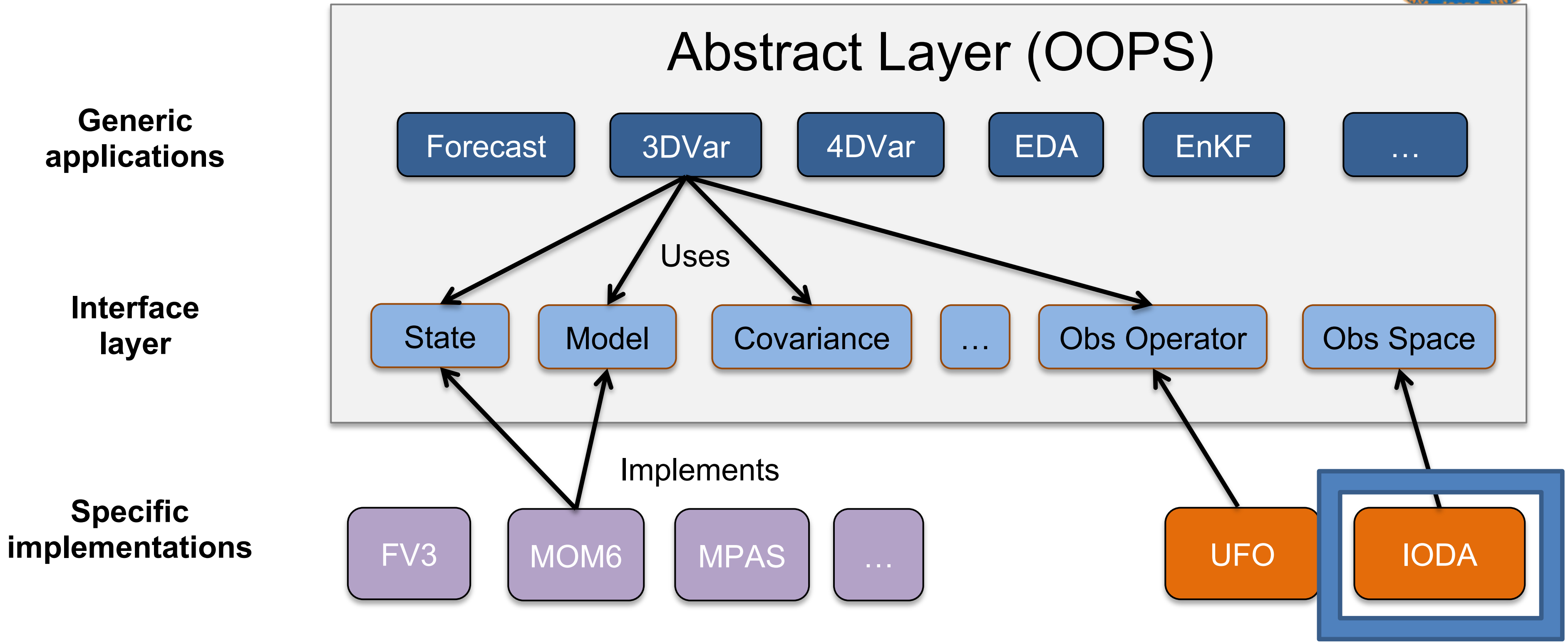
The Joint Effort for Data assimilation Integration (JEDI)

Observations and observation operators in JEDI

Joint Center for Satellite Data Assimilation (JCSDA)

AMS virtual short course: Introduction to JEDI
10 March 2021

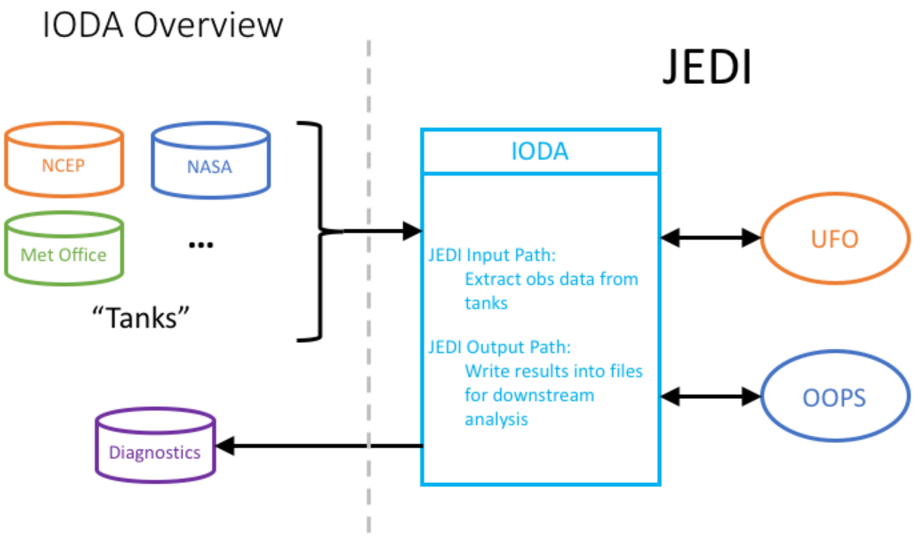

Abstract Layer (OOPS)
Generic applications
Forecast
3DVar
4DVar
EDA
EnKF
…
Uses
Interface layer
State
Model
Covariance
…
Obs Operator
Obs Space
Implements
Specific implementations
FV3
MOM6
MPAS
…
UFO
IODA



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

IODA: Interface for Observation Data Access

IODA is the component of JEDI that ingests and processes observational data (in-memory observation data structures, I/O of the observational data, parallel distribution of observations).



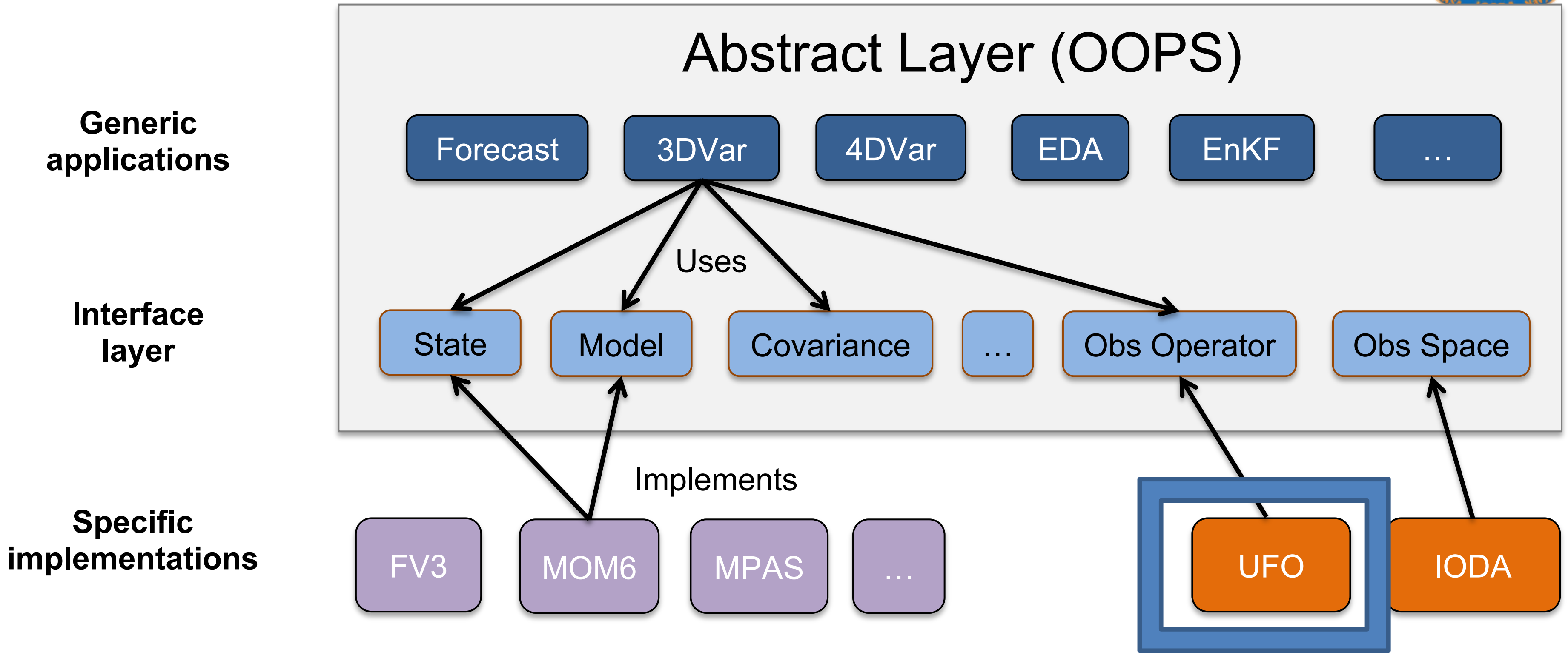
- Currently supported input file format (backend): hdf5/netcdf4. (IODA version 2.0 released soon).
- IODA converters available/worked on for multiple observation types.
- Ongoing work on IODA backends for ODC and BUFR.

Obs Space and its configuration

Obs Space: responsible for I/O and in-memory access to observation metadata, observation values, etc.

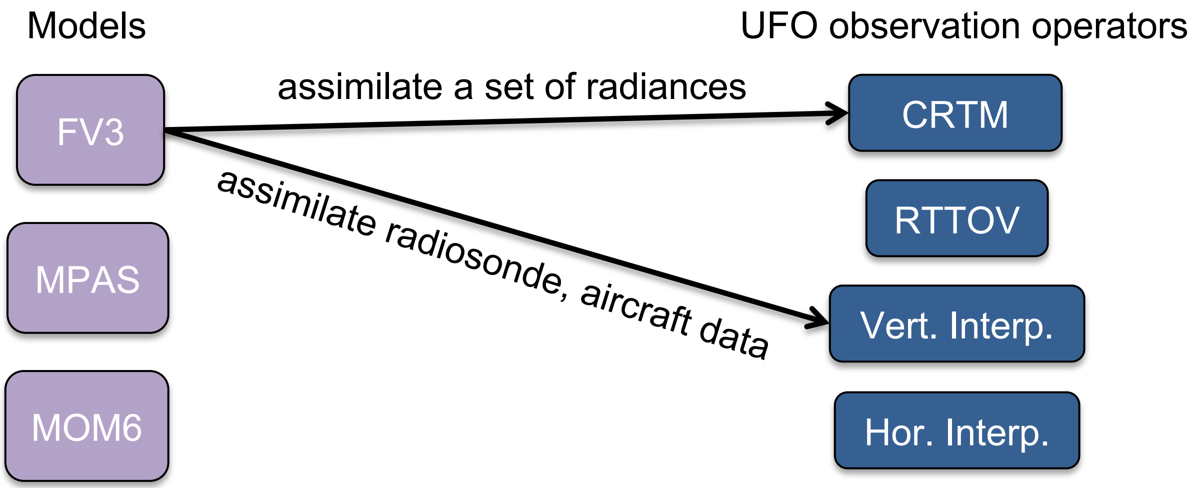
One Obs Space typically corresponds with one observation type/ observation file (e.g. aircraft observations; radiances from specific instrument/satellite)

```
observations:
- obs space:
    name: amsua_n19
    obsdatain:
      obsfile: filename
    simulated variables: [brightness_temperature]
    channels: 4-6, 9-14
- obs space:
    name: aircraft
    obsdatain: ...
    simulated variables: [air_temperature,
                          eastward_wind]
```


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

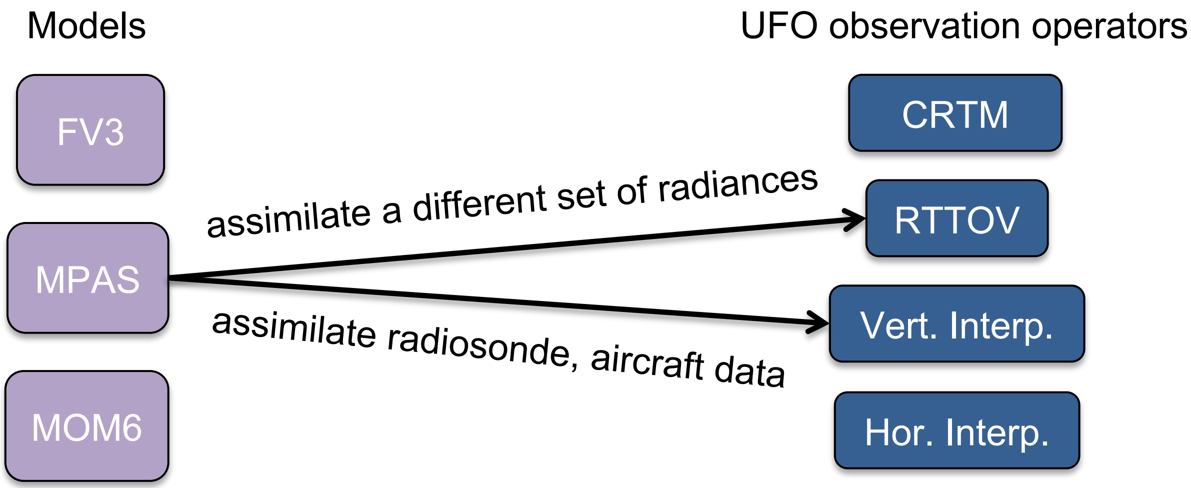
UFO: Unified Forward Operators

- “a community app-store for observation operators” (share observation operators between JCSDA partners and reduce duplication of work)
- Most of the UFO observation operators could be used by any of the models connected to JEDI. Configurable through the YAML files.



UFO: Unified Forward Operators

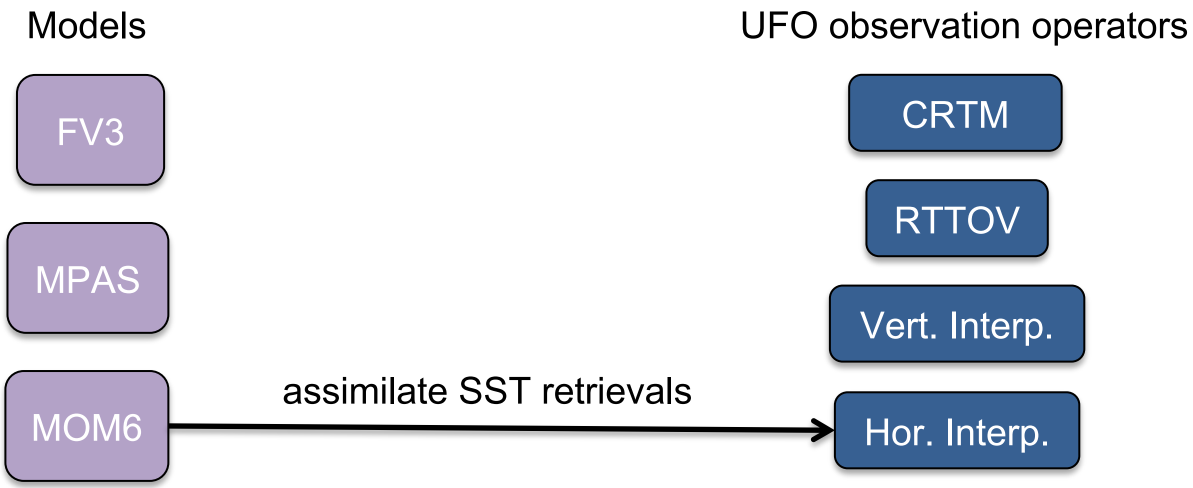
- “a community app-store for observation operators” (share observation operators between JCSDA partners and reduce duplication of work)
- Most of the UFO observation operators could be used by any of the models connected to JEDI. Configurable through the YAML files.



UFO: Unified Forward Operators

- “a community app-store for observation operators” (share observation operators between JCSDA partners and reduce duplication of work)

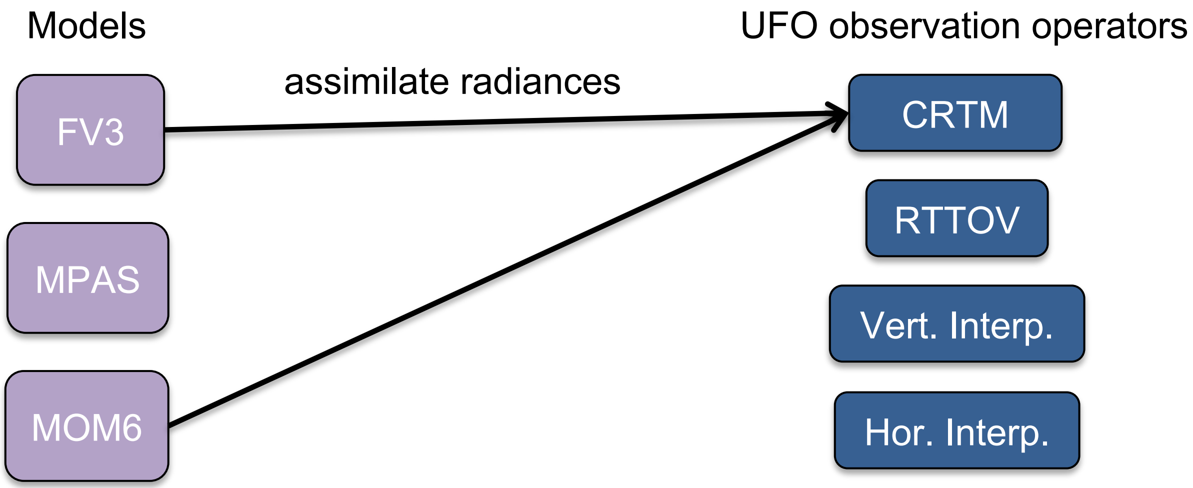
- Most of the UFO observation operators could be used by any of the models connected to JEDI. Configurable through the YAML files.



UFO: Unified Forward Operators

- “a community app-store for observation operators” (share observation operators between JCSDA partners and reduce duplication of work)

- Most of the UFO observation operators could be used by any of the models connected to JEDI. Configurable through the YAML files.



How does this work?
Ex: satellite radiance calculations

The forward operator: take model state (surface emissivity, profiles of temperature, water vapor, liquid water, graupel, etc.) and get observables (radiances or brightness temperatures).

There are four key parts:

- **GetValues:** horizontal interpolation of model field to observation location and time.
- **ObsOperators:** Apply an observation operator (e.g. CRTM) to the interpolated field and simulate measurements: $h_{crtm}(x)$.
- **Bias Correction:** Add bias correction to simulated measurement: $h(x) = h_{crtm}(x) + \sum_{i=1}^{n_p} x_{n+1} P_i$.
- **ObsFilters:** At each of these stages (input, interpolation, et cetera), apply QC filters that can set QC flags and/or change observation error variances.

What are ObsOperators?

ObsOperators translate model state into observables.
These are all written generically and can be adapted to different instruments.

[1]

```
observations:
- obs space: …
  obs operator:
    name: CRTM
    Absorbers:
    - H2O
    - O3
    - CO2
    Clouds:
    - Water
    - Ice
    obs options: …
```

[2]

```
observations:
- obs space: …
  obs operator:
    name: GnssroBndNBAM
    obs options:
      sr_steps: 2
      vertlayer: mass
      use_compress: 1
      super_ref_qc: NBAM
```

[3]

```
observations:
- obs space: …
  obs operator:
    name: VertInterp
```

Examples of ObsOperators

Short description	Where in ufo	Name in the yaml
Interface to CRTM	ufo/src/ufo/crtm	CRTM
Interface to CRTM AOD	ufo/src/ufo/crtm	Aod
Interface to RTTOV	ufo/src/ufo/rttov	RTTOV
Vertical interpolation in log-pressure (can be used for radiosondes, aircrafts, satwinds)	ufo/src/ufo/atmvertinterp	Radiosonde Aircraft Satwind
GNSSRO bending angle following GSI	ufo/src/ufo/gnssro/BndNBAM	GnssroBndGSI
GNSSRO refractivity following GSI	ufo/src/ufo/gnssro/Ref	GnssroRef
GNSSRO bending angle ROPP1D	ufo/src/ufo/gnssro/BndROPP1D	GnssroBndROPP1D
GNSSRO bending angle ROPP2D	ufo/src/ufo/gnssro/BndROPP2D	GnssroBndROPP2D
“Identity” ObsOperator (horizontal interpolation)	ufo/src/ufo/identity	Surface SeaSurfaceTemp

UFO observation "filters"

- Abstract "observation filters", most can be fairly generic.
- Observation filters have access to:
  - Observation values and metadata
  - Model values at observations locations
  - Simulated observation value and diagnostics
- Generic filters can be used with many observation types (options controled through YAML configuration files)
- Examples of generic filters:
  - gross error checks, background checks, blacklisting, thinning…

Bounds Check filter

```yaml
- filter: Bounds Check
  filter variables:
  - name: brightness_temperature
    channels: 4-6
  minvalue: 240.0
  maxvalue: 300.0
```

The Bounds Check filters out-of-range data.

- *Bounds check applies to channels 4 through 6*
- *QC flags data where*

  $T_{b,k} < 240$ *or* $T_{b,k} > 300$ *for* $k = 4,5,6$

Domain Check filter

```yaml
- filter: Domain Check
  where:
  - variable:
      name: sea_surface_temperature@GeoVaLs
    minvalue: 200
    maxvalue: 300
  - variable:
      name: height@MetaData
    is_defined:
  - variable:
      name: station_id@MetaData
    is_in: 3, 6, 11-120
  - variable:
      name: air_pressure@MetaData
    is_not_defined:
```

Example: Sanity checks on sea surface temperatures, height, station ids and air pressure.

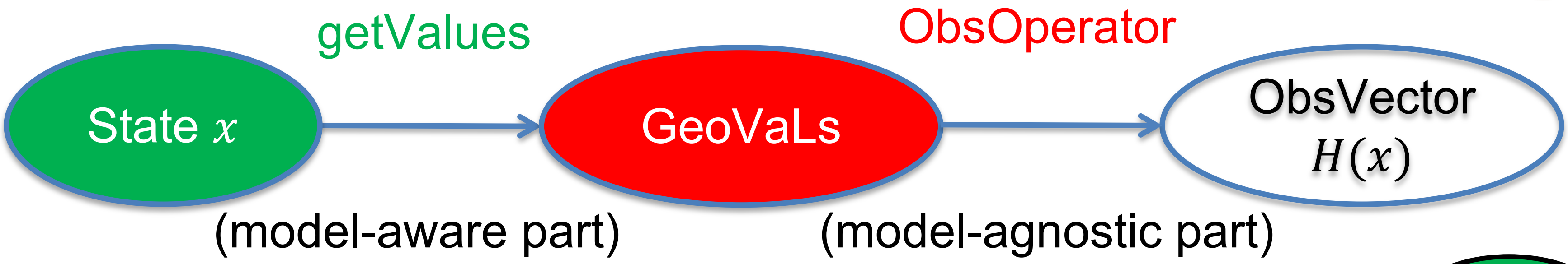
Retain only observations:

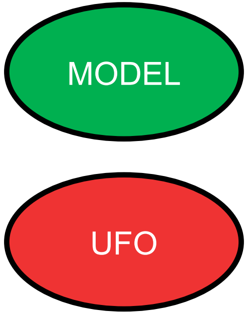
- *taken at locations where the sea surface temperature retrieved from the model is between 200 and 300 K (inclusive),*
- *with valid height metadata (not set to "missing value"),*
- *taken by stations with IDs 3, 6 or belonging to the range 11-120, and*
- *without valid air_pressure metadata.*

Thank You

Questions?

Interpolated model state (GeoVaLs)





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.

Interpolated model state (GeoVaLs)

- GeoVaLs are vertical profiles of requested model variables at observation x-y-t location. Forward operator defines which variables it needs from the model to compute H(x)).
- Examples:
  - radiances: vertical profiles of t, q, ozone, pressure; surface variables: wind, SST, land properties, etc.
  - radiosondes/aircrafts: vertical profiles of pressure (to do vertical interpolation), t, u, v, q
  - sea ice concentration retrieval: sea ice concentrations for different ice thickness categories
  - SST retrieval: SST (observation operator becomes an identity in this case)