The Joint Effort for Data assimilation Integration (JEDI)

IODA Subsystem

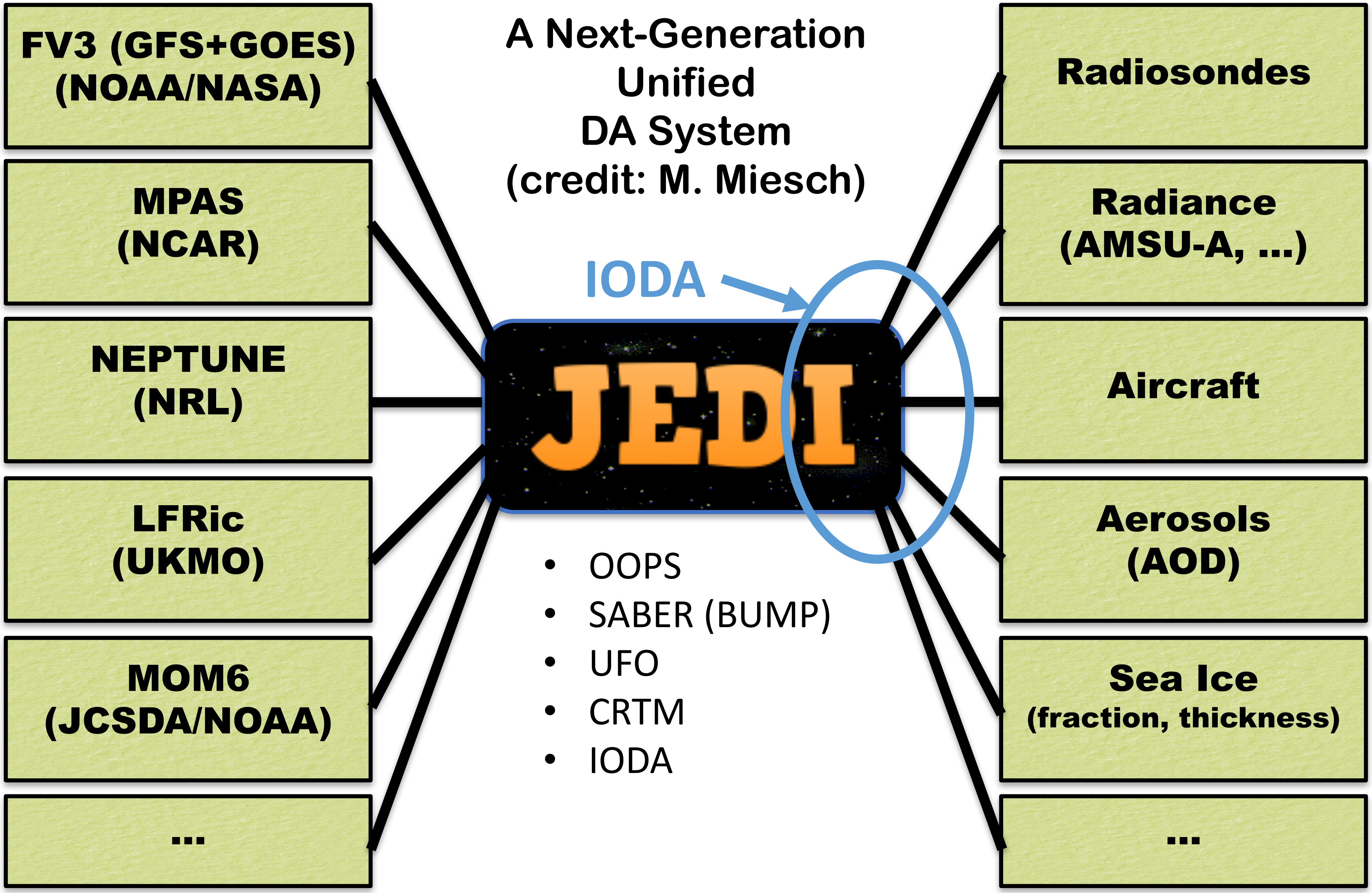
Joint Center for Satellite Data Assimilation (JCSDA)

JEDI Academy - 10-13 June 2019

What is IODA?

- IODA is the subsystem in JEDI that provides access to observation data
- **I**nterface for **O**bservation **D**ata **A**ccess
- Three levels
  - Archive: long term storage, historic database
  - File: on disk, data for one DA Cycle
  - Memory
- Two environments
  - Plotting, analyzing, verifying on workstation or laptop
  - DA and other HPC applications (MPI, threads, GPUs, …)

JEDI Overview



- Enables high leverage
- For example, add your model
- Then you have access to:
  - Obs data
  - Forward operators
  - DA flows
  - Etc.

IODA Long Term Vision

- Look and feel of a database
  - Select and filter data on various criteria
    - Select observations within a DA timing window
    - Filter on QC marks, horizontal locations, station id's, etc.
- Converge on a common file format for holding observation data
  - A common format would greatly facilitate the sharing of data and the exchange science results
- Likely that we will adopt an existing database solution
  - We will soon be evaluating ECMWF's ODB solution once the ODC software (API) becomes available

IODA Requirements

- IODA Workshop
  - February 2019 at NRL in Monterey, CA
  - Requirements gathering effort
  - First round of gathering (ala agile methodology)
- Categories of requirements include, but not limited to:
  - Access to Data and Meta-data
    - Data and meta-data are both important
    - Efficient query style access
  - Flexible
    - Wide variety of obs types
  - Reliable
    - Operational mode cannot break down
  - Portable
    - Across hardware platforms, programming languages and compilers
  - Security
    - Protected data and results

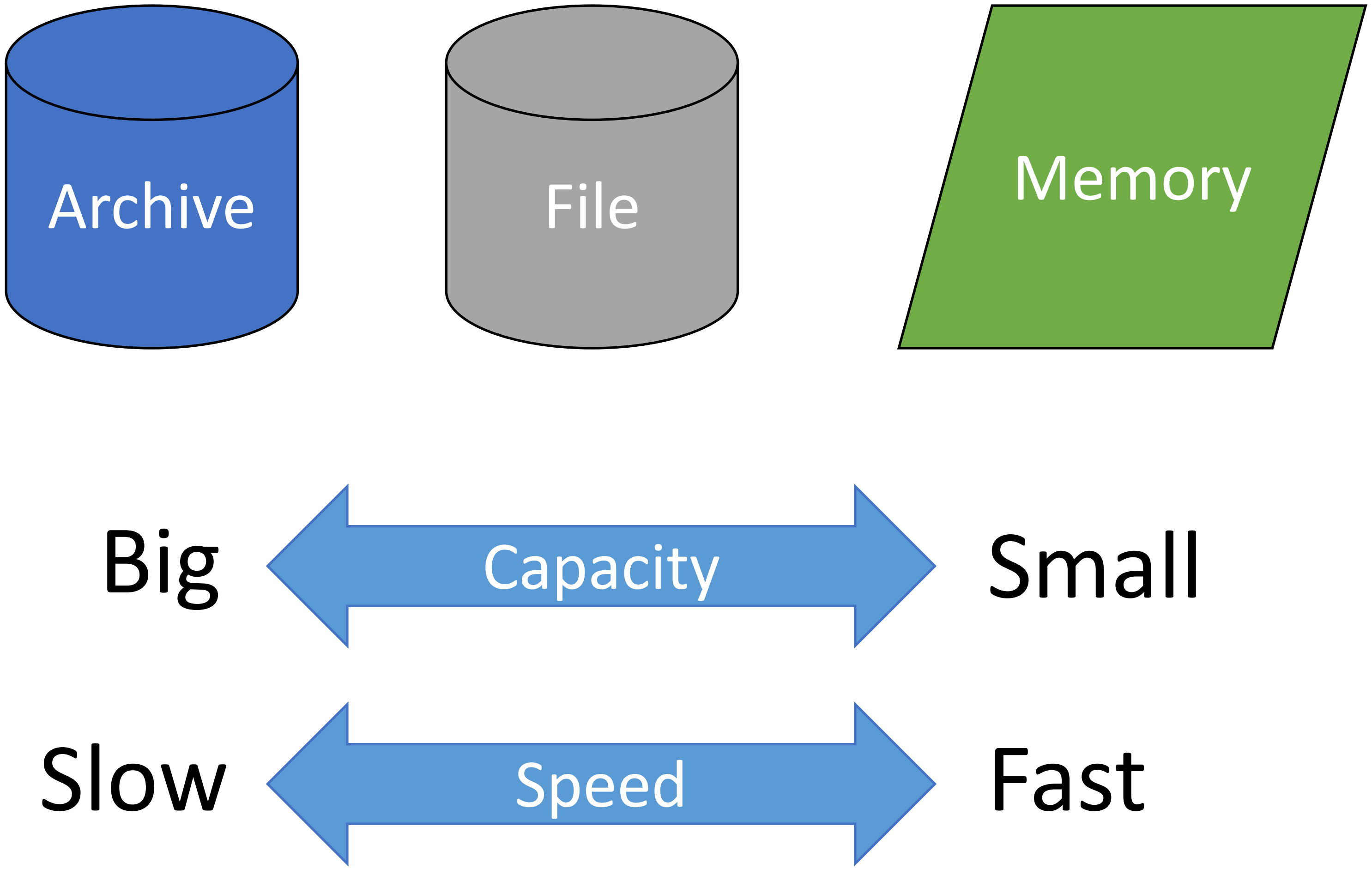
IODA Status

Observation Type (Instrument)	IODA obs file	H(x)	Notes
Aircraft	✔ (Completed)	✔ (Completed)	
Radiosonde	✔ (Completed)	✔ (Completed)	
Satwinds	✔ (Completed)	✔ (Completed)	
Additional conventional	✔ (Completed)	✔ (In Progress)	Sfc obs, ship obs, wind profiler, etc.
AMSU-A	✔ (Completed)	✔ (Completed)	n15, n18, n19, metop-a, metop-b, aqua
AIRS	✔ (Completed)	✔ (Completed)	aqua
CRIS	✔ (Completed)	✔ (Completed)	npp
HIRS-4	✔ (Completed)	✔ (Completed)	metop-a, metop-b
IASI	✔ (Completed)	✔ (Completed)	metop-a, metop-b
MHS	✔ (Completed)	✔ (Completed)	n18, n19, metop-a, metop-b
VIIRS AOD	✔ (Completed)	✔ (Completed)	
GNSSRO	✔ (Completed)	✔ (Completed)	
Marine (retrievals)	✔ (Completed)	✔ (Completed)	SST, SSS, SSH, Insitu Temp, Seaice (frac, thick)
Marine (radiances)	✔ (In Progress)	✔ (In Progress)	

✔ Completed

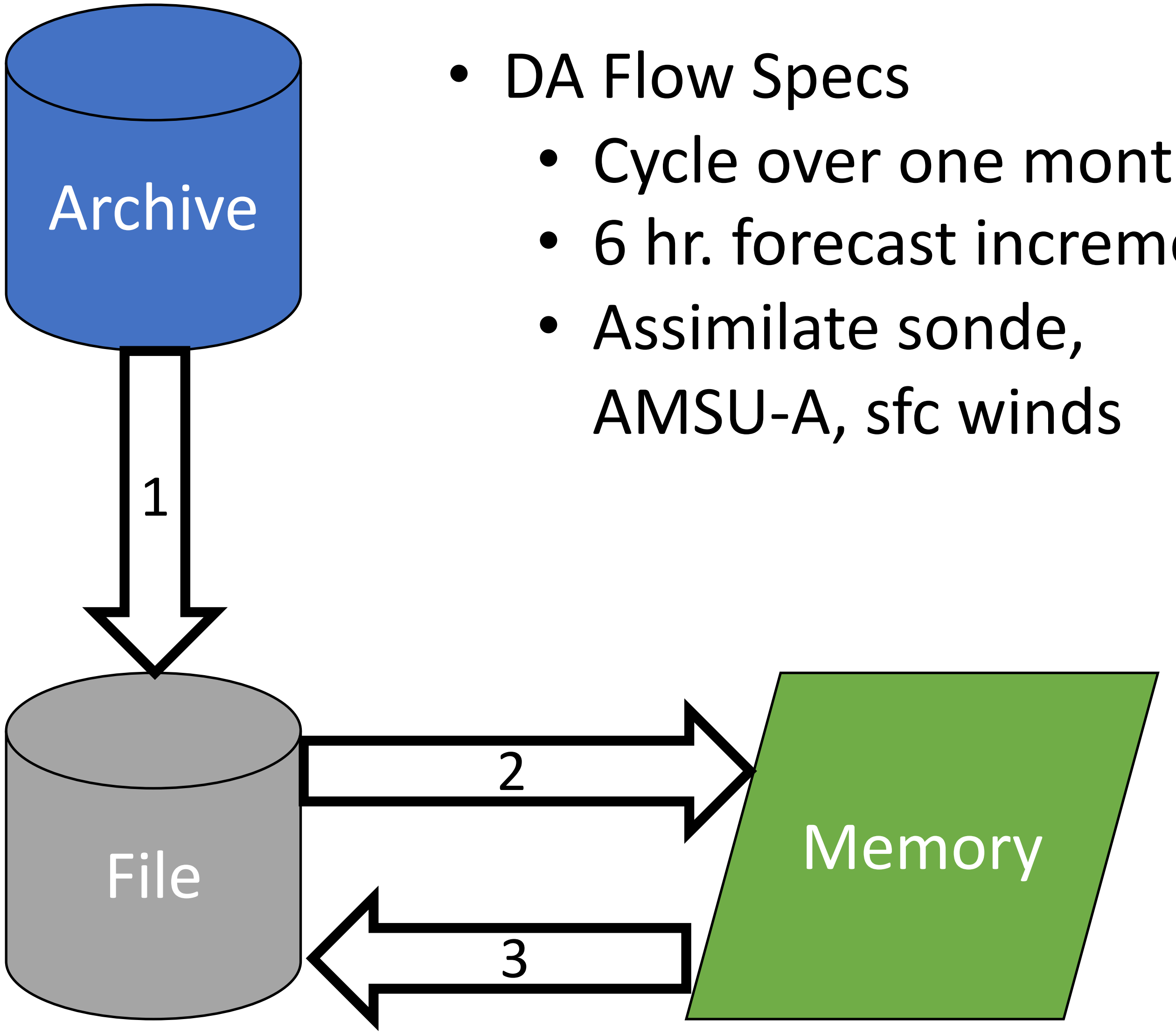
✔ In Progress

IODA Levels: Capacity-Speed Tradeoff



- **Archive**
  - All obs types
  - All dates (decades)
- **File**
  - Specific obs types
  - DA cycle begin - end
- **Memory**
  - Specific obs types
  - Forecast begin - end

DA Flow



- DA Flow Specs
  - Cycle over one month
  - 6 hr. forecast increments
  - Assimilate sonde, AMSU-A, sfc winds

1. Retrieve all sonde, AMSU-A and sfc winds within the one month period

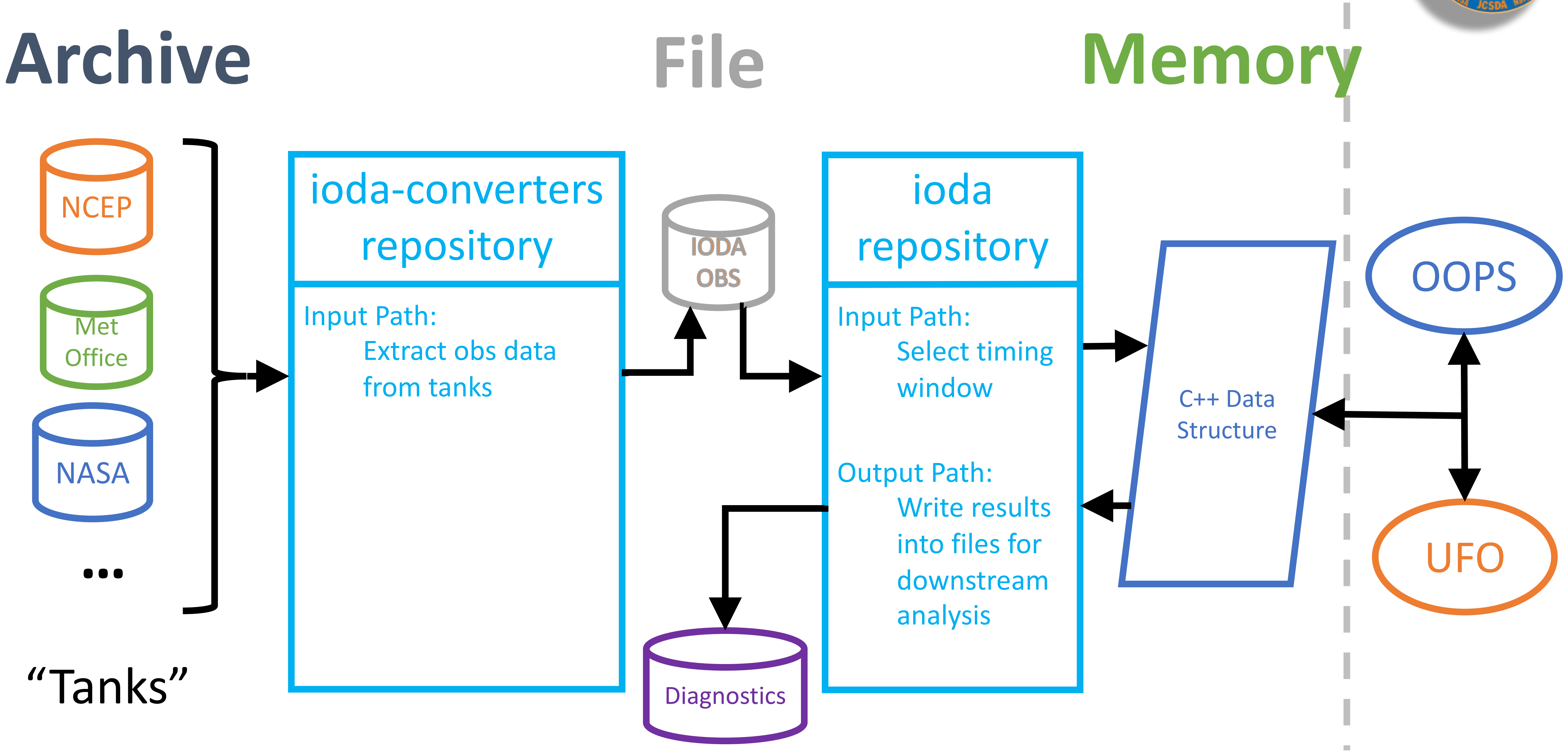
2. Loop over each 6-hr forecast window retrieving appropriate sonde, AMSU-A and sfc winds as needed

3. As DA flow progresses, store diagnostics into output files

IODA Status

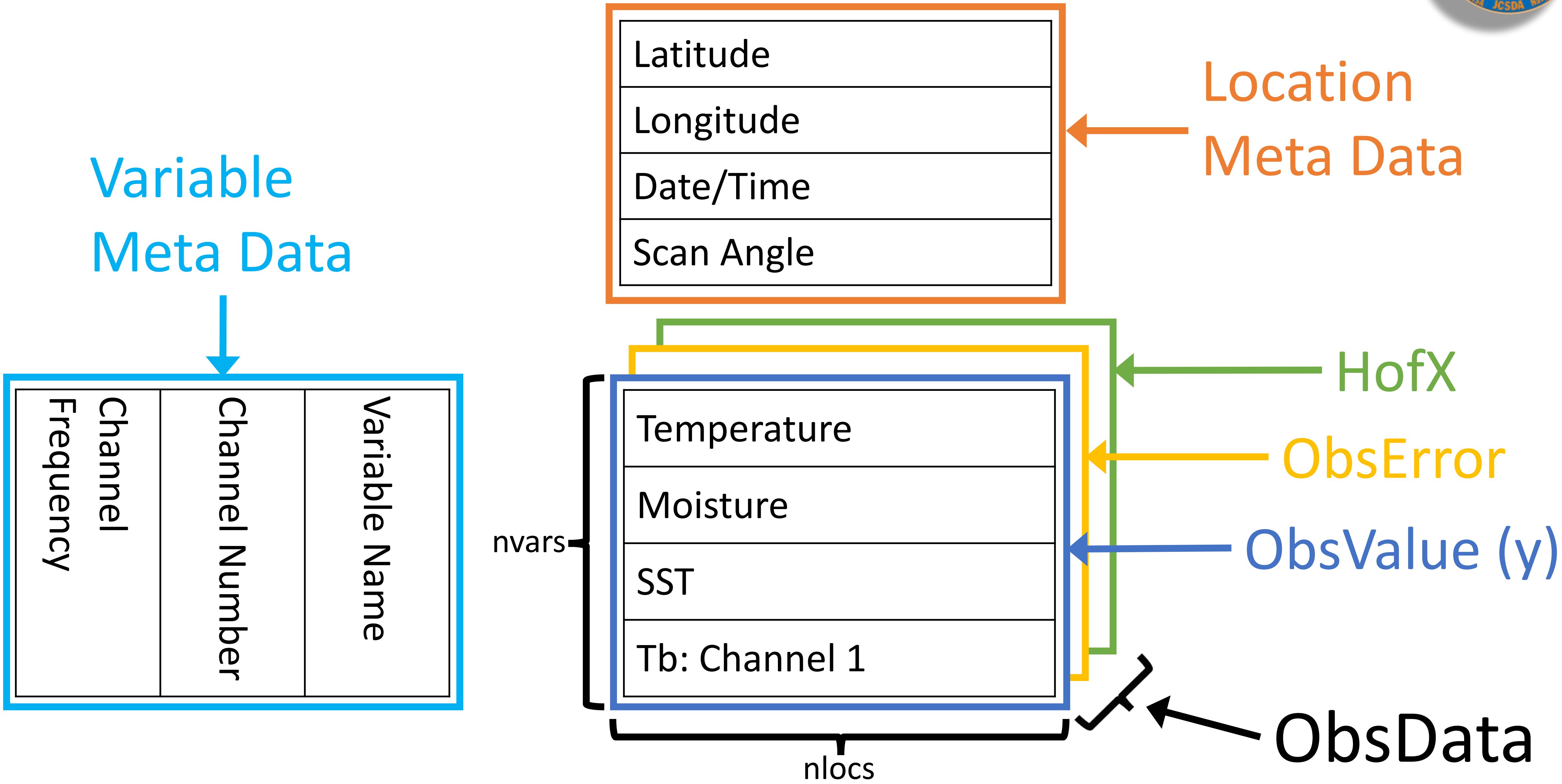
- IODA started as a simple prototype and is evolving toward the long term vision
- We are currently using pieces of existing systems to mimic the database style access to the three IODA levels
  - Archive
    - Data tanks from various data centers
    - Different file types (BUFR, netcdf, specialized binary)
    - Different methods of organizing data within the file
      - QC code semantics, internal table structure and layout, etc.
  - File
    - Netcdf
    - Unified organization within the file
  - Memory
    - C++ Standard Data Structures

IODA Today



Archive
File
Memory
NCEP
Met Office
NASA
...
"Tanks"
ioda-converters repository
Input Path:
Extract obs data from tanks
IODA OBS
ioda repository
Input Path:
Select timing window
Output Path:
Write results into files for downstream analysis
Diagnostics
C++ Data Structure
OOPS
UFO

IODA Current Observation Data Organization



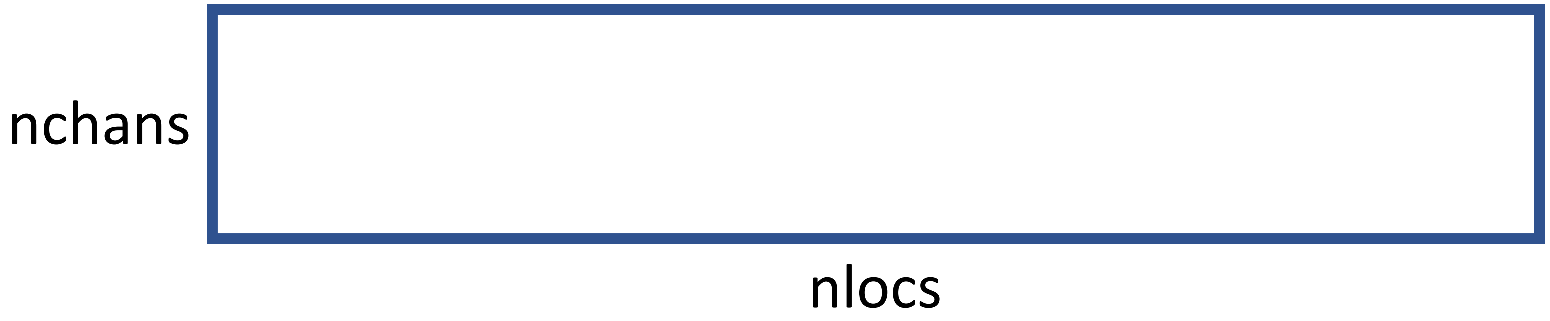
Proposed Change to Obs Data Organization

- Instead of 2D tables in the data organization, use n-dimension arrays
  - Accommodate more complex obs type, such as ocean wave spectra
- The number of dimensions is variable
  - Each dimension has an associated meta data table
- Examples
  - Radiosonde
    - 2D array, dimensions (nvars, nlocs)
    - Metadata: variables (nvars), locations (nlocs)
  - Radiance
    - 3D array, dimensions (nvars, nlocs, nchans)
    - Metadata: variables (nvars), locations (nlocs) and channels (nchans)
  - Wave spectra
    - 3D array, dimensions (nvars, nlocs, nfreqs, ndirs)
    - MetaData: variables (nvars), locations (nlocs), frequencies (nfreqs) and directions (ndirs)

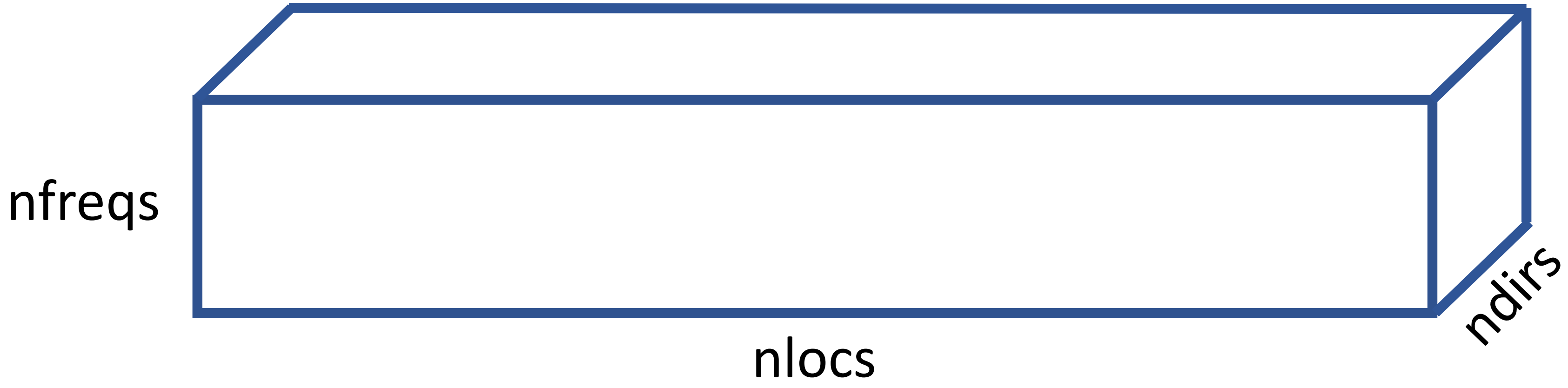
Multi-Dimensioned Observation Data

T(nlocs)

NOTE: nvars dimension not shown for simplicity



Tb(nlocs, nchans)

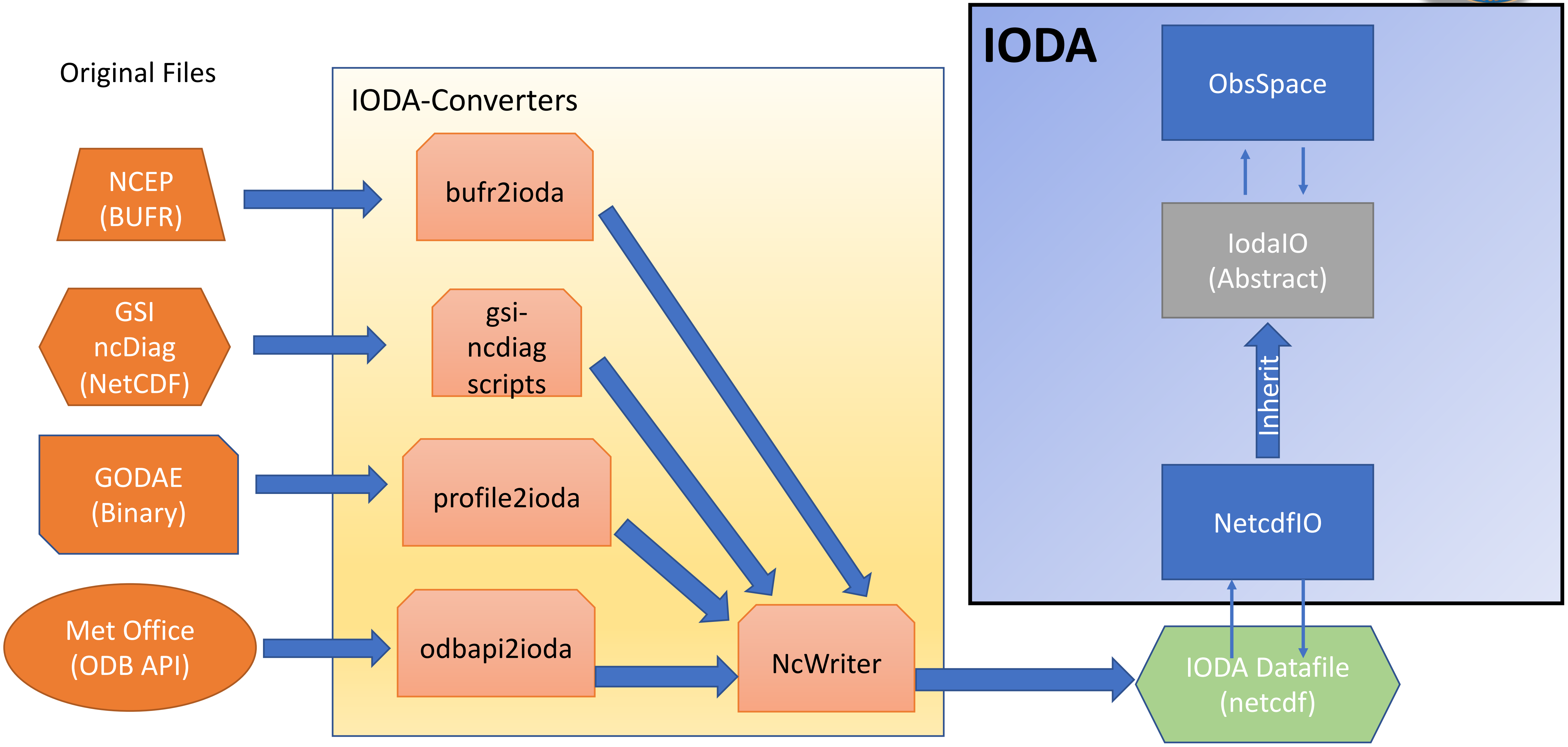


Wave Spectra
(nlocs, nfreqs, ndirs)

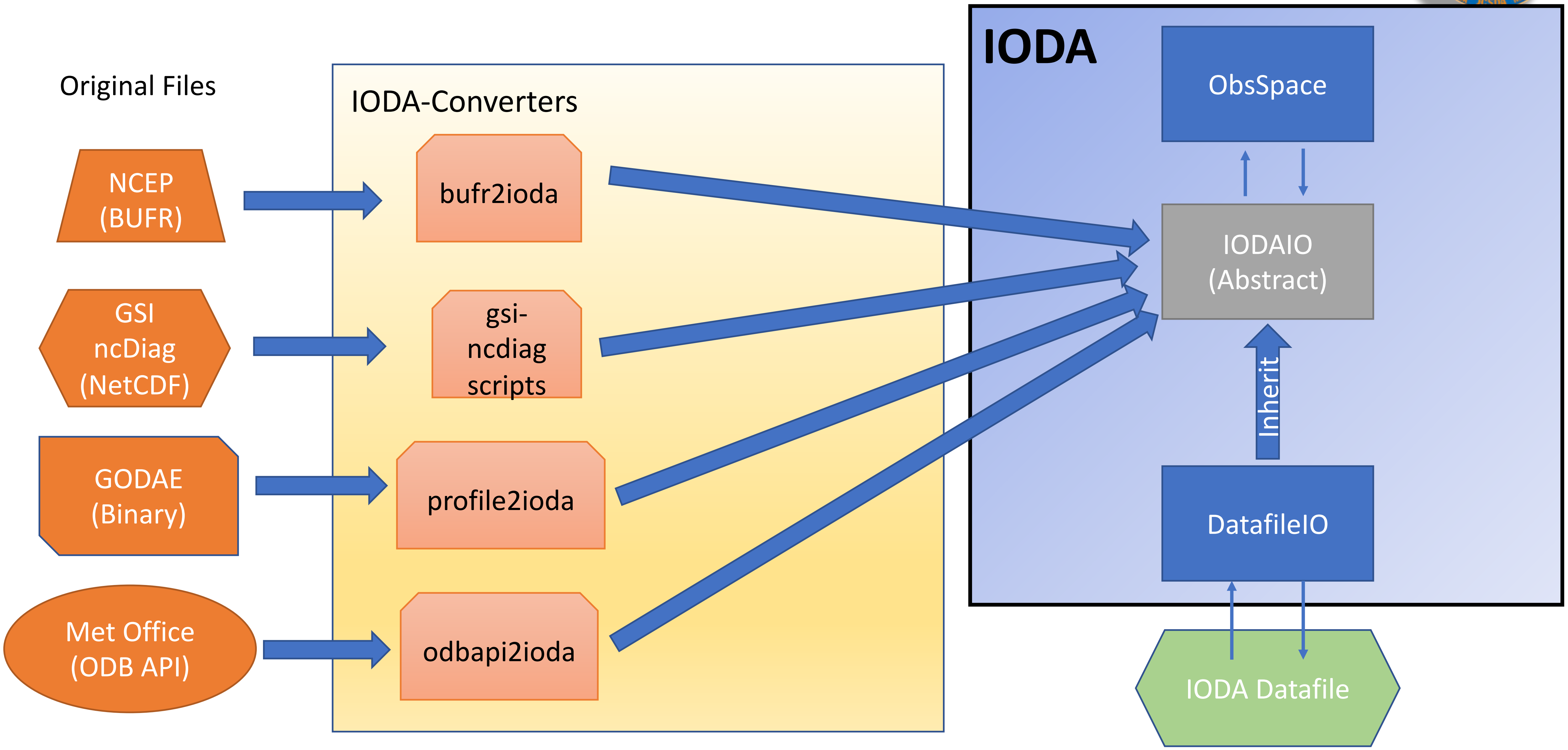
IODA Converters

- Set of scripts and programs to convert various formats into the target IODA format
  - Python (using a common python netcdf writer class)
  - Fortran
- The currently used IODA file format is:
  - Netcdf
    - ODB will be evaluated when ODC interface becomes available
  - Obs data organization from previous slides
- Currently, can convert:
  - Marine GODAS, GODAE, etc. (mix of netcdf and custom binary)
  - NCEP prepBUFR
  - GNSSRO raw BUFR
  - GSI Ncdiag (netcdf diagnostics)
  - UK Met Office ODB

IODA Converters Current Design



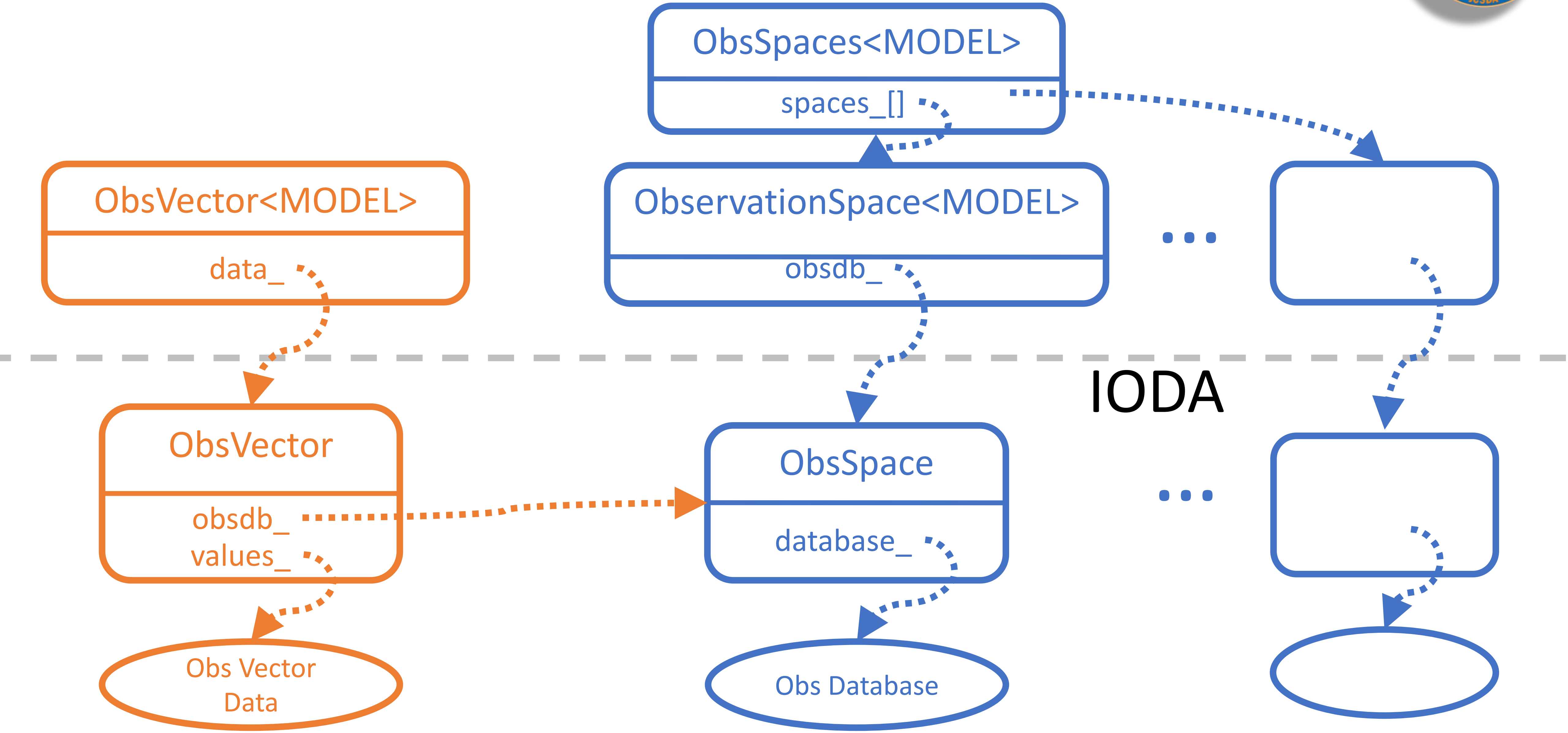
IODA Converters Target Design



IODA C++ Class Relationship with OOPS

- OOPS provides an abstract interface layer
  - Templated classes
    - Allows multiple implementations of underlying concrete classes
- IODA provides concrete classes that implement two of the OOPS interface classes
  - ObsVector
    - Holds quantities such as y and H(x)
  - ObsSpace
    - Analogous to a mathematical space that contains vectors
    - Provides an interface to observation data stored in files

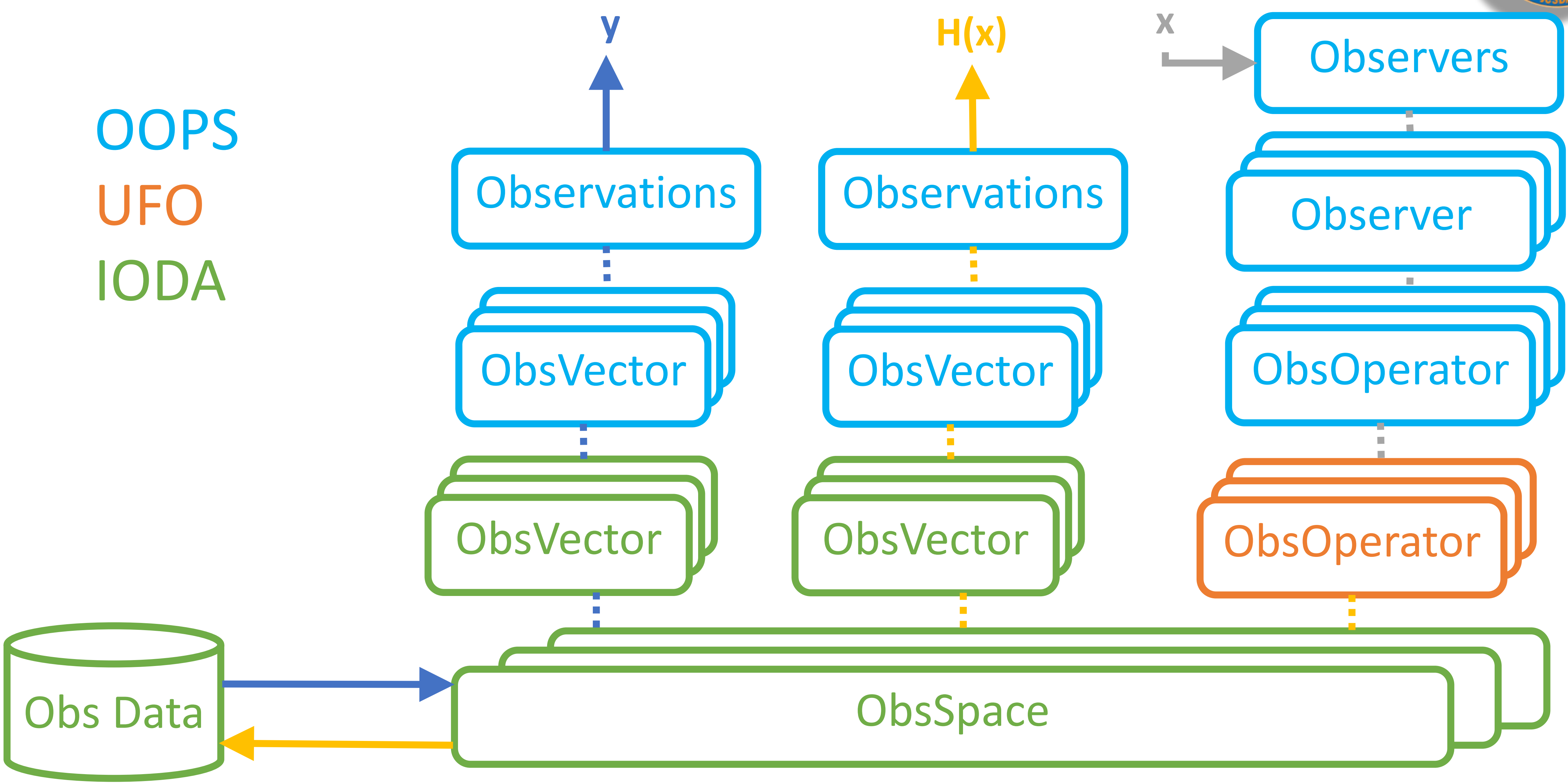
IODA Class Structure



Multiple Observation Spaces and Vectors

- The total observation vector (e.g., y) is chopped up into pieces according to observation type
  - Different observation types require different algorithms to simulate those observations
    - Radiosonde
    - Radiance
    - AOD
- UFO holds ObsOperator objects that implement the various observation simulation algorithms
- OOPS manages these pieces with Observations, ObsSpaces and Observers, ObsOperators collector classes
  - Corresponding ObsOperator and ObsVector objects are paired up and OOPS chains these pieces together for the cost function minimization step

IODA Data Flow



Interface with OOPS

- C++
- Access to ObsData array in the data store
- ObsVector methods

```
// I/O
  void read(const std::string &);
  void save(const std::string &) const;
```

IODA-OOPS Interface Usage

- Data is transferred between ObsVector and ObsSpace objects
  - The constructor of an ObsSpace defines the variables that comprise the observation vector
  - Each variable corresponds to a row in an ObsData array
  - The ObsVector may select a subset of the rows in the ObsData array
- Read: transfer data from ObsSpace to ObsVector

```
Log::trace() << "CostJo::CostJo start" << std::endl;
yobs_.read("ObsValue");
Log::trace() << "CostJo::CostJo done" << std::endl;
```

- Save: transfer data from ObsVector to ObsSpace

```
//  Save H(x)
    boost::scoped_ptr<Observations_> yobs(pobs->release());
    Log::test() << "H(x): " << *yobs << std::endl;
    yobs->save("hofx");
```

Interface with UFO

- Fortran
- Access to an individual row in the data store
  - I.e., a row from either of the ObsData or MetaData arrays
- ObsSpace methods

```
integer function obsspace_get_nlocs(obss)
subroutine obsspace_get_db(obss, group, vname, vect)
subroutine obsspace_put_db(obss, group, vname, vect)
```

  - obss argument is a C pointer to an ObsSpace object
  - group argument is a Fortran string with the table (group) name
    - Eg., “ObsValue”, “HofX”
  - vname argument is a Fortran string with the variable (row) name
    - Eg., “Temperature”, “Moisture”
  - vect argument is a Fortran 1D array (vector)

IODA-UFO Interface Usage

- It is the client's responsibility to allocate memory for the vector data
- Rows of the tables are nlocs in length

```
nlocs = obsspace_get_nlocs(obss)
allocate(TmpVar(nlocs))

call obsspace_get_db(obss, "MetaData", "Sat_Zenith_Angle", TmpVar)
geo(:)%Sensor_Zenith_Angle = TmpVar(:)

call obsspace_get_db(obss, "MetaData", "Sol_Zenith_Angle", TmpVar)
geo(:)%Source_Zenith_Angle = TmpVar(:)

call obsspace_get_db(obss, "MetaData", "Sat_Azimuth_Angle", TmpVar)
geo(:)%Sensor_Azimuth_Angle = TmpVar(:)

call obsspace_get_db(obss, "MetaData", "Sol_Azimuth_Angle", TmpVar)
geo(:)%Source_Azimuth_Angle = TmpVar(:)

call obsspace_get_db(obss, "MetaData", "Scan_Position", TmpVar)
geo(:)%Ifov = TmpVar(:)

call obsspace_get_db(obss, "MetaData", "Scan_Angle", TmpVar)
geo(:)%Sensor_Scan_Angle = TmpVar(:)

deallocate(TmpVar)
```

ObsSpace Configuration (YAML)

```yaml
Jo:
  ObsTypes:
  - ObsSpace:
      name: Aircraft
      ObsDataIn:
        obsfile: Data/obs/aircraft_obs_2018041500_m.nc4
      ObsDataOut:
        obsfile: Data/hofx/aircraft_hyb-3dvar-gfs_2018041500_m.nc4
      simulate:
        variables: [air_temperature]
```

...

```yaml
  - ObsSpace:
      name: AMSUA-NOAA19
      ObsDataIn:
        obsfile: Data/obs/amsua_n19_obs_2018041500_m.nc4
      ObsDataOut:
        obsfile: Data/hofx/amsua_n19_hyb-3dvar-gfs_2018041500_m.nc4
      simulate:
        variables: [brightness_temperature]
        channels: 1-15
```

- Required keywords
  - name
  - ObsDataIn
  - simulate
- Optional keywords
  - ObsDataOut
  - channels

IODA Next steps

- Short-term
  - Ioda interface (file writer) for converters
  - Expansion of data store to multi-dimensioned observations
    - Ocean wave-spectra, e.g.
- Longer-term (this year)
  - Complete the design of long term IODA subsystem
    - Database design
      - Select a database solution (ODC, other?)
      - Define how to organize data within the database file and memory structures
    - This task will determine the common file format for IODA
  - Create the IODA Archive Level
    - Data storage strategy (cloud)
    - Interface for archiving and retrieving data
    - Tools to convert raw observation data to the IODA common file format

Summary

- The IODA subsystem provides access to observation data for the OOPS and UFO subsystems in JEDI
  - Ioda-converters are used to get external obs data prepared for ingest into JEDI
  - Ioda is used to store obs data with associated metadata (ObsSpace), and to present y and H(x) vectors (ObsVector) to UFO and OOPS
- We have implemented a prototype interface that is able to handle observation data of a variety of observation types using a common data organization
  - Want to move this to a database solution