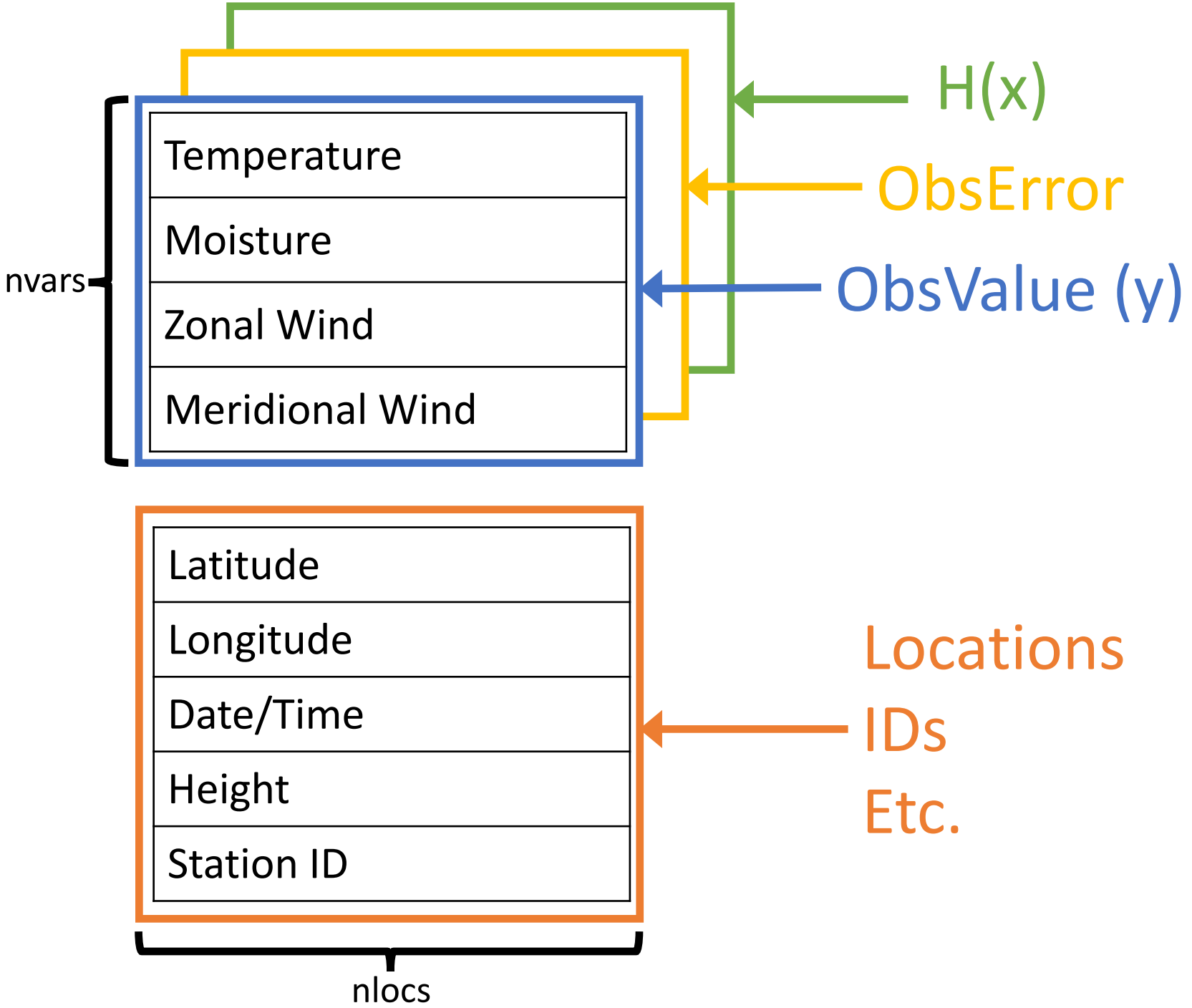
IODA Goals

- Solidify the interface
  - Allows us to modify the implementation without thrashing the clients of the interface
- Move toward an SQL-like database
  - Start with simple Fortran structures (linked list of vectors)
  - Move to C++ structures (Boost::MultiIndex)
  - Move to database (ODB, SQLite, ?)

IODA Interface

ObsData



MetaData

Locations
IDs
Etc.

IODA Interface: OOPS

- C++
- Developers shouldn’t have to worry about this
- Access to an ObsData table in the data store
  - ObsVector is an example of a client
    - read() and save() methods call ObsSpace getObsVector() and putObsVector() methods
- ObsSpace methods

```
void getObsVector(const std::string &, std::vector<double> &) const;
void putObsVector(const std::string &, const std::vector<double> &);
```

  - 1st argument is name of ObsData table (eg, ”ObsValue”, “HofX”)
  - 2nd argument is a vector holding the data

IODA Interface: UFO

- Fortran
- Developers of Obs Operators will use these
- Access to an individual row in the data store
  - Ie, a row from either of the ObsData or MetaData tables
- ObsSpace methods

```
integer function obsspace_get_nobs(obss)
integer function obsspace_get_nlocs(obss)
subroutine obsspace_get_db(obss, group, vname, length, vect)
subroutine obsspace_put_db(obss, group, vname, length, 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., “air_temperature”, “eastward_wind”
  - vect argument is a Fortran 1D array (vector) of doubles
  - length argument is the size of the 1D array given by vect

IODA Fortran interface usage

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

```
type(c_ptr), value, intent(in)                    :: obss
```

```
real(kind_real), allocatable :: pressure(:)
```

```
! observation of pressure (for vertical interpolation)
nlocs = obsspace_get_nlocs(obss)
allocate(pressure(nlocs))
call obsspace_get_db(obss, "ObsValue", "air_pressure", nlocs, pressure)
```

```
deallocate(pressure)
```

IODA Interface notes

- Bookkeeping quantities
  - nlocs
    - Number of unique locations
    - Size of a Locations object or MdataVector object
  - nvars
    - Number of variables in the obs data table
  - nobs
    - Number of unique locations
    - Size of an ObsVector object
    - Equal to nvars * nlocs
- Individual channels on satellite instruments are treated as separate variables
  - One channel per row in the ObsData table
- Missing values
  - Not all locations have all variables (t,q,u,v) for radiosonde and aircraft
  - Run-time QC can create missing values (ie, throw out some obs)

IODA Interface: Future

- Add a mechanism for selecting “records”
  - A record is an atomic unit that should not be broken down any further
    - Eg, a single radiosonde sounding
    - A record is preserved when distributing across multiple process elements
  - Bookkeeping
    - nrecs
      - Number of unique records
      - Used during MPI distribution and QC filtering
- Add a second type of MetaData table to store information on a variable-by-variable basis
  - Eg. Frequencies associated with each satellite channel