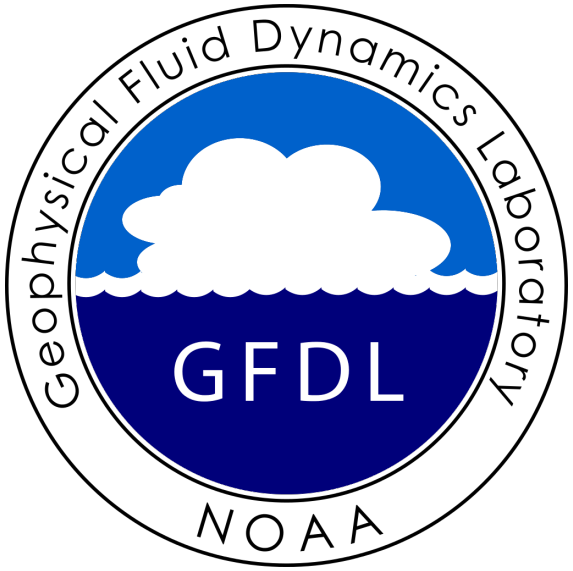
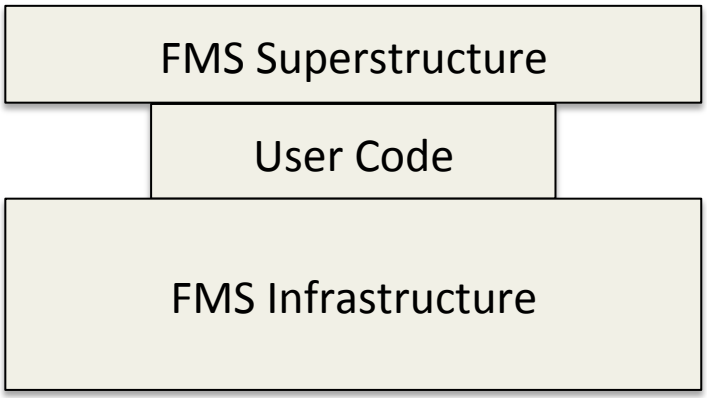
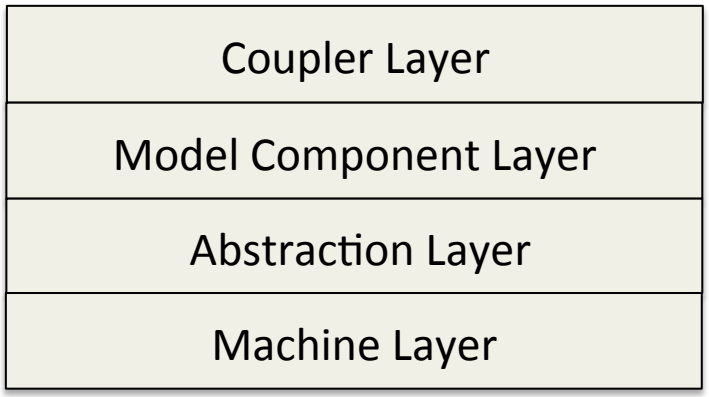
A More Detailed Description of GFDL's FMS Library



Weekly FV3GFS Technical Meeting
19 June, 2017

What is FMS





Flexible Modeling System effort began in 1998, with GFDL's first distributed memory machine

Superstructure design allows the building of multiple models

Supports multiple dynamical cores, components (ocean, ice, land, etc) and atmospheric physics packages

Infrastructure comprised of common utilities needed by model components

Superstructure Layer

Consists of:

- system driver
- flux exchange
- ensemble manager
- component drivers

System driver performs *init*, *integrate*, and *end* phases

By default, all models run as ensembles

Abstracted types hold fluxes and properties to be exchanged between model components

Component drivers provide unified interfaces between system driver and user code

Infrastructure Layer

Two logical layers: machine & abstraction

Machine layer basic interface to parallelism and I/O: *MPP*

FMS is a high-level functional layer built upon *MPP*

Abstraction layer contains multiple *"managers"*

- time manager
- diagnostic manager
- block manager
- field manager
- tracer manager
- fms_io

and other utilities/APIs:

- data override
- constants
- interpolators
- exchange grid
- astronomy functions
- math functions (ffts, tri-di solver, etc)

MPP

Communication Layer - *provides fundamental point-to-point and asynchronous, non-blocking communication*

Domains Layer – *handles decompositions and tracks which ranks are "connected" for halo updates, I/O, and nesting*

I/O Layer - *handles all file types including direct interface to NetCDF/HDF*

Clocks - *allow high-resolution timing of code segments*

Miscellaneous - *pelists, error handling, reductions, unit tests (functionality and performance), etc*

MPP: pelists

Group of MPI-ranks all working on a specific task

Stored as a sorted, one-dimensional array

Lowest order MPI-rank is considered the “root-pe”

All pelists are a subset of the global pelist

Used to create an MPI group and unique communicator

An MPI-rank can belong to multiple pelists

MPP: Communication

Default is asynchronous and non-blocking: *isend, irecv*

Synchronization functions based on mpi_wait to ensure no outstanding messages on an MPI-rank

MPP functions to: *send, recv, transmit, bcast, gather, scatter*

Uses current active pelist or optional pelist argument

MPI tags to ensure proper message delivery

Functions based on collectives exist, for performance reasons their use is discouraged (exception *allreduce*)

All datatypes supported: *real, integer, character, complex*

MPP: Domains

Responsible for domain decomposition of:

- tiled grids
- unstructured grids
- rectilinear grids
- tri-polar grids

Creates an internal data structure containing:

- halo update communication mapping
- directional data orientation and rotations
- tile information

Domain data structure is unique to a halo width

Manages nested grid <-> coarse grid boundary updates

MPP: Domains

Domain updates via:
- pre-defined groups
- message aggregation
- asynchronous (start/complete)

Gather decomposed data into a global field array:
- memory intensive
- not recommended

MPP: I/O

Abstracts the calls to NetCDF

Provides basic functionality used to build higher level abstraction layer known as *FMS_io*

Manages open/close of files and file units

Uses *domains* logic to create a special I/O domain that can be configured at runtime (see I/O Subset examples)

I/O domain can read from a single file

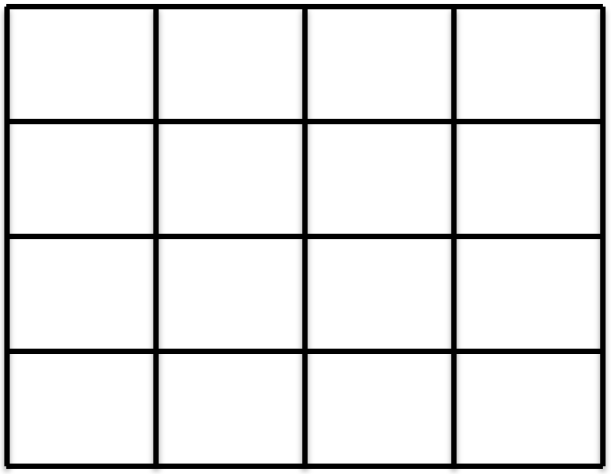
Automatically includes extra metadata buffer space

Can read “regions” of data to reduce memory bloat

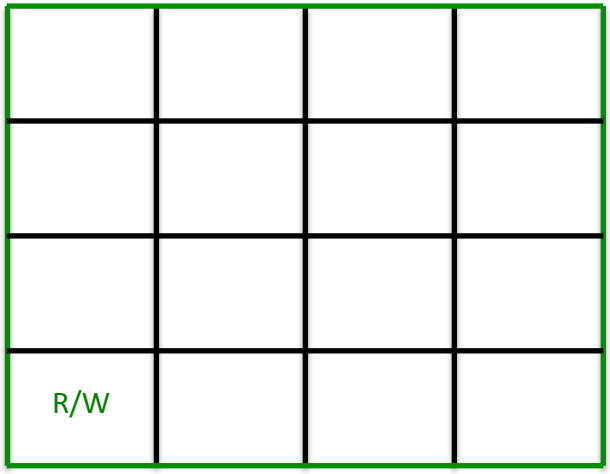
ASCII files can be read by root-pe and broadcast

I/O Subset Example 1

4 x 4 decomposition

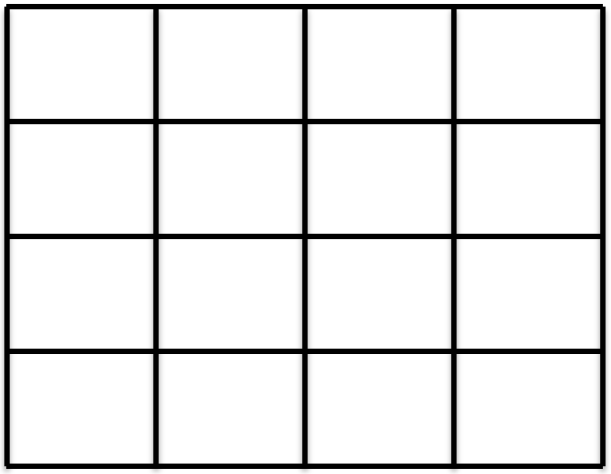


1 x 1 io_subset

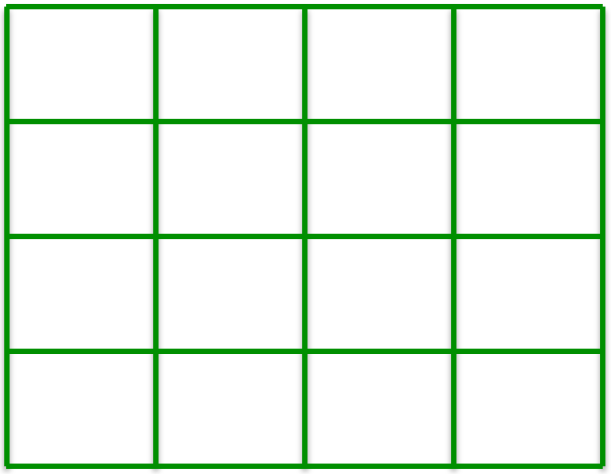


I/O Subset Example 2

4 x 4 decomposition



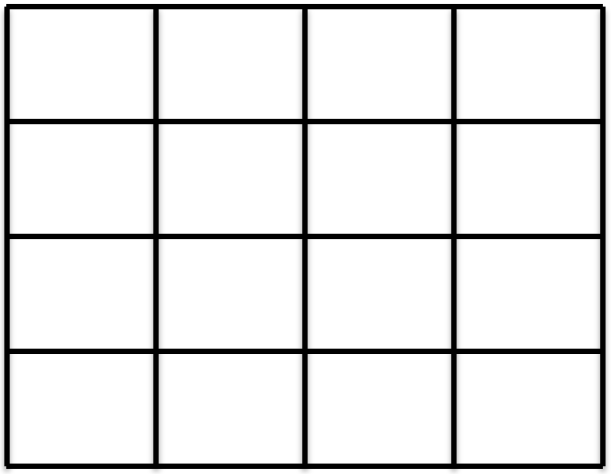
4 x 4 io_subset



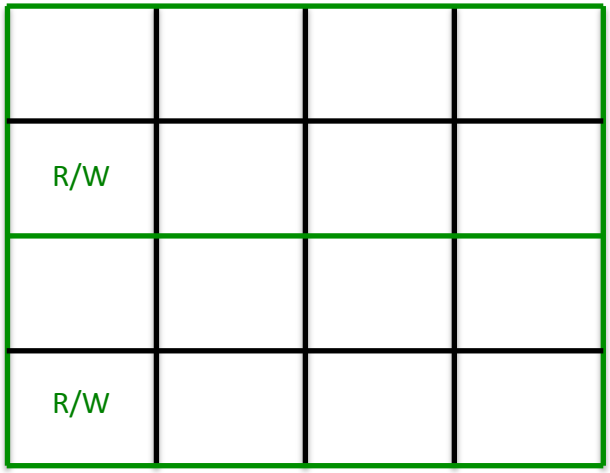
R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W

I/O Subset Example 3

4 x 4 decomposition



1 x 2 io_subset



MPP: Clocks

Uses *wtime (default)* or *system_clock*

Error checks to ensure begin/end pairs are matched

Defined with differing granularities: *component, subcomponent, driver, module, routine, loop, infrastructure*

Global summary output at end of run: *min/max/avg*

Per MPI-rank summary available

Verbosity of summary granularity is configurable

Clocks can be "sync'ed" - *timer doesn't begin until all members of the pelist have checked in*

MPP: Miscellaneous

Error handling

NOTE	message to stdout by root-pe
WARNING	NOTE by every member of pelist
FATAL	WARNING plus program termination

Global sums

fast sum

- sum of local sums -> allreduce
- not reproducible across decompositions

bitwise exact

- Standard - gather to root-pe -> in order sum -> bcast to pelist
- EFP sum - convert to int -> mpp_fast_sum -> convert to real

Checksums

restart integrity checks
global, pelist, or local
aid in debugging

FMS_IO

Most of the IO is mediated through this layer

Wraps the mpp_io layer to extend basic functionality

Provides simplified data access

- queries, open/close, reads/writes
- reduces need to fully express metadata for all variables
- IO subsets and mosaic (tile) transparent to user

Complete restart functionality

```
id_restart = register_restart_field(Tra_restart, fname_nd, 'u', Atm(n)%u, &
                                    domain=fv_domain, mandatory=.true.)

call save_restart(Tra_restart, timestamp)

call restore_state(Tra_restart)
```

FMS_IO

Files expressed in basic format, "tile*n*" added by *MPP* layer based on domain information

Query, Set, and Nullify:

- active domain
- filename appendix (used for ensembles, nests, etc)

Query:

- Global and variable attributes
- field size and name
- mosaic (tile) number and filename

read/write data using only filename, fieldname, data

- optionally specify time-level and/or region

Time Manager

Supports multiple calendars

- julian
- gregorian
- 30-day months
- no-leap
- no-calendar

YY.MM.DD.hh.mm.ss.ticks stored as an abstract type

ticks is a configurable fraction of a second *(1000 => milliseconds)*

Operators to perform “math” on the type

Query and Set the calendar, date, and time

Alarm to alert you when an interval has been reached

Function that returns seconds for a time interval – or optionally days & seconds

Block Manager

Distributed computing can be abstracted away from scientists and developers - OpenMP requires tight interaction

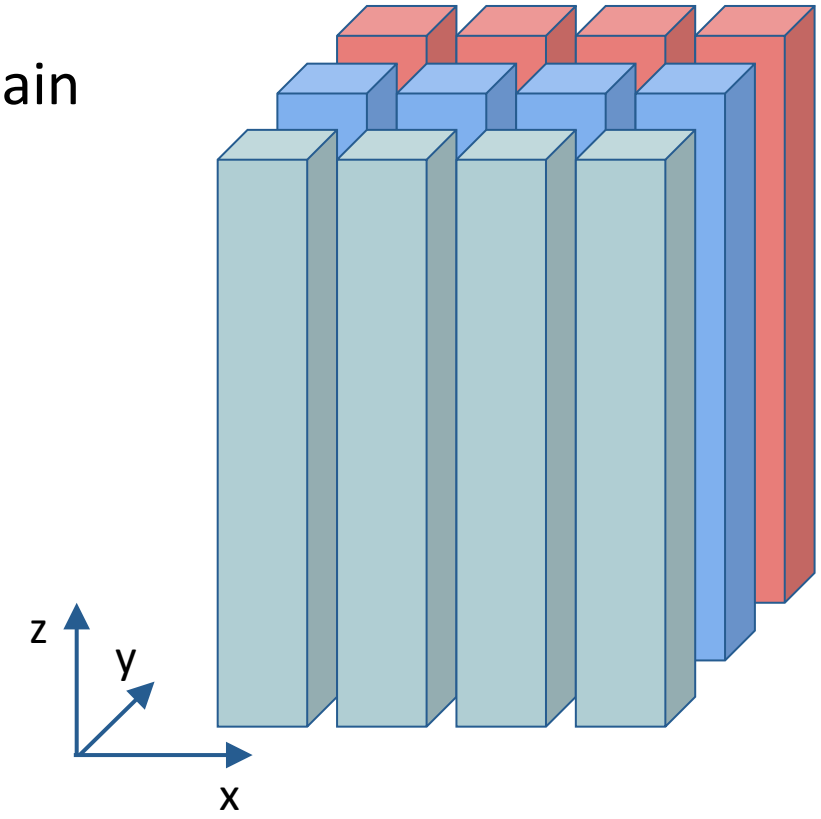
MPI-rank domains are further decomposed to aid with OpenMP threading and populates a type

Stores extent for the MPI-rank domain and for each block

Includes per-block translation from MPI-rank domain to block (indexing)

Blocks expressed in packed *(ix,k)* or rectilinear *(ib,jb,k)* format

Blocks are not required to be uniform in size



Field/Tracer Manager

Reads in field descriptions from a flat file

```
"TRACER", "atmos_mod", "o3mr"
        "longname",    "ozone mixing ratio"
        "units",       "kg/kg"
        "profile_type", "fixed", "surface_value=1.e30" /
```

Tracks fields by component and classification

APIs for

- initialization of the field from metadata information
- querying array location mapping from tracer "name"
- querying generic attributes for specific purposes

Tracers can be added mid-run

- diagnostic tracers no impact on solution (e.g. particle traces)
- prognostic tracers may require equilibration or relaxation

Diagnostic Manager

Set of simple calls for parallel diagnostics on distributed systems

Built on the parallel I/O interfaces from *FMS_IO*

Capable of multiple sampling and/or averaging intervals specified at run-time

Ouput scalars up to 3D fields

Support for limited-area extents (regional)

Run-time specification of diagnostics controlled through the *diag_table* file

Diagnostic Manager

```
fvGFS_20150801.18Z
2015 8 1 18 0 0

# Output files
"grid_spec",                -1,  "months",   1, "days",  "time"
"atmos_4xdaily",             6,  "hours",    1, "days",  "time"
"atmos_daily",              24,  "hours",    1, "days",  "time"
"atmos_static",             -1,  "hours",    1, "hours", "time"
# files needed for NGGPS evaluation
"nggps3d_4xdaily",          6,  "hours",    1, "days",  "time"
"nggps2d",                  0,  "hours",    1, "hours", "time"

# Output fields
"dynamics", "grid_lon",   "grid_lon",   "grid_spec",          "all", .false., "none", 2
"dynamics", "grid_lat",   "grid_lat",   "grid_spec",          "all", .false., "none", 2
"dynamics", "zsurf",      "zsurf",      "atmos_static",       "all", .false., "none", 2
"dynamics", "ps",         "SLP",        "atmos_daily",        "all", .true., "none", 2
"dynamics", "ps",         "SLP",        "atmos_4xdaily",      "all", .true., "none", 2
"dynamics", "vort850",    "vort850",    "atmos_4xdaily",      "all", .false., "none", 2
"dynamics", "pt",         "temp",       "nggps3d_4xdaily", "all", .true.,"236.0, 295.0, -25.0, 49.5,-1,-1",2
"gfs_phys", "totprcp",    "PRATEsfc",   "nggps2d",            "all",  .false., "none",  2
```

Diagnostic Manager

Similar format to that used by restarts

```
id_swdn(1,i) = register_diag_field (mod_name, 'swdn_200hPa_clr', &
                    axes(1:2), Time, 'clear sky SW flux down at 200 hPa', &
                    'watts/m2', missing_value=missing_value)

used = send_data (id_swdn, swdn, Time)

used = send_data (id_swdb, swdn, Time, rmask=Grd%tmask, &
                    is_in=isc, js_in=jsc, ks_in=1, ie_in=iec, je_in=jec, ke_in=nk)
```

Multi-entry variables in the *diag_table* require a single send_data call, diag_manager will manage all variations

send_data:

- called every timestep (for time averaging and accumulating)
- alarms used to trigger output[1]
- optional index arguments for OpenMP/blocked data

(1) data sent to NetCDF-managed buffers - actual flush of data controlled separately

Data Override

Similar capability to *gcycle* in GFS physics

Using information in the *data_table*, perform spatial and temporal interpolation to replace a prognostic field

```
# Ice overrides (Old Format)
"ICE", "sic_obs", "SIC", "INPUT/sst_ice_clim.nc", .FALSE., 0.01
"ICE", "sit_obs", "SIC", "INPUT/sst_ice_clim.nc", .FALSE., 1.06
"ICE", "sst_obs", "SST", "INPUT/sst_ice_clim.nc", .FALSE., 1.0

# Atmosphere overrides (New Format)
"ATM", "dust1_aerosol", "dust1", "INPUT/aerosol_month.nc", "none", 1.0

# Land overrides (New Format)
"LND", phot_co2, "co2", "INPUT/co2_data.nc", "bilinear", 1.0e-6
```

Overrides done on cyclical basis

```
call data_override ('ATM', 'u_obs', obs, Time, override=done)
call data_override ('ATM', 'v_obs', obs, Time, override=done)
```

FMS: Miscellaneous

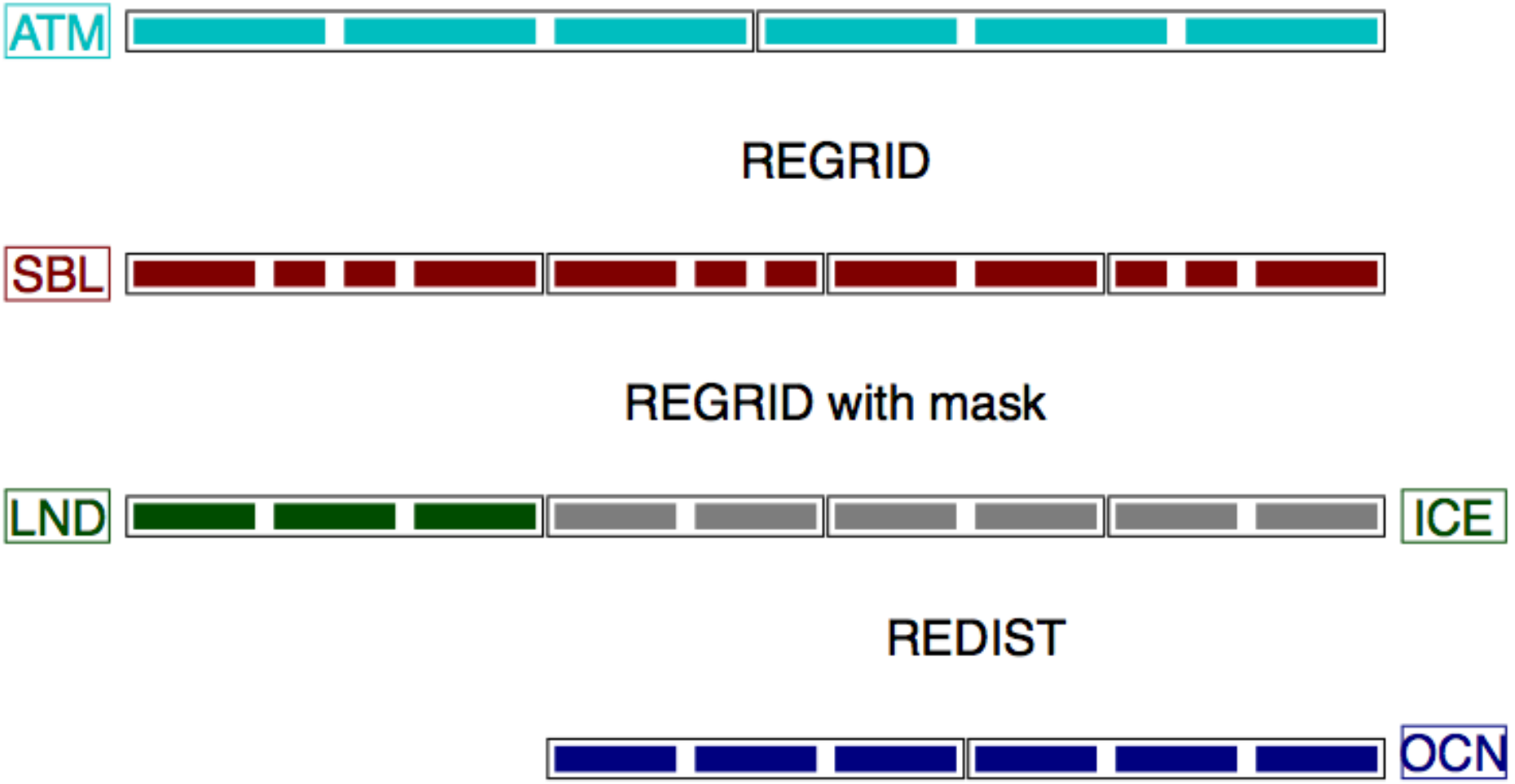
Astronomy:

- astronomical variables and properties
- used mainly by shortwave radiation

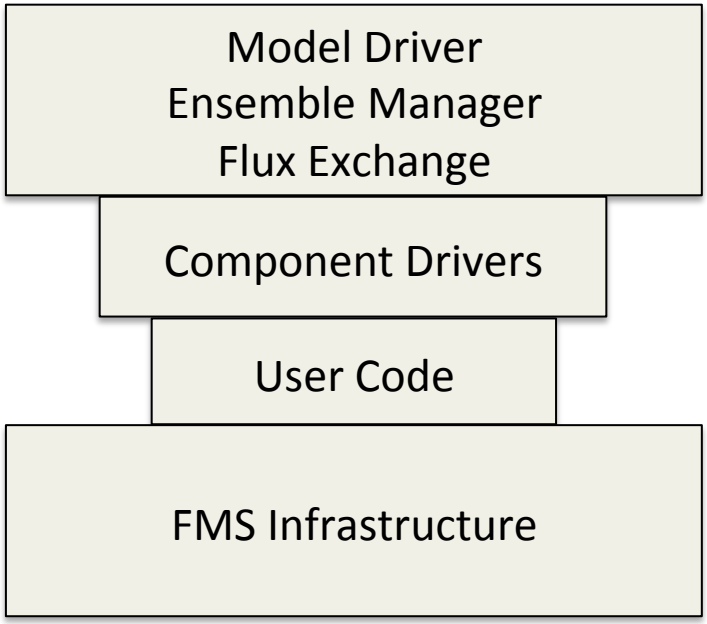
Sat_vapor_pres:

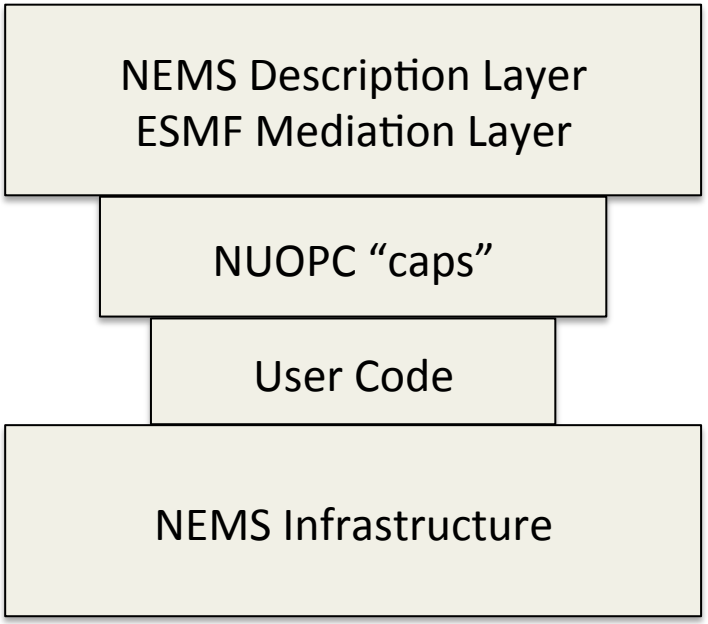
- initializes the lookup tables
- given the relative humidity, calculates:
  - saturation vapor pressure
  - specific humidity
  - vapor mixing ratio
- compute derivatives with respect to temperature of:
  - saturation vapor pressure
  - specific humidity
  - vapor mixing ratio

Exchange Grid



FMS <-> NEMS





Model Driver: template-based which evolves as components materialize

Infrastructure: a collection of enabling software in a library

NEMS layer: built at compile time based on configuration specification

Infrastructure: a collection of enabling libraries