WRF Data Assimilation System

Michael Kavulich, Jr.

WRFDA Tutorial, August 2016, NCAR

WRFDA System – Outline

- *Introduction*
- Compiling the code
- WRFDA software structure
- Computing overview

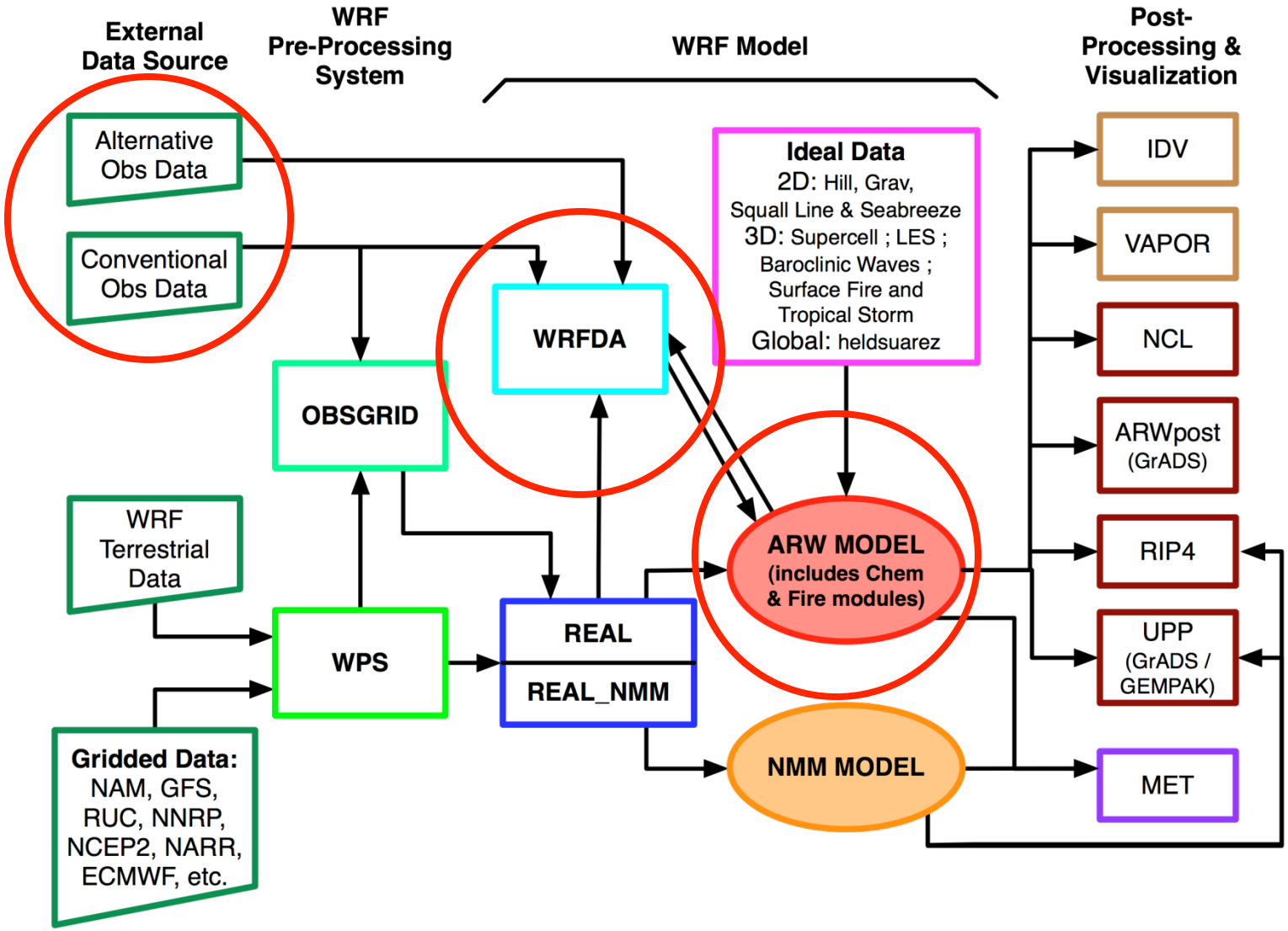
Introduction – What is WRFDA?

- A data assimilation system for the WRF Model (ARW core)
  - 3D- and 4D-VAR, FGAT, Ensemble, and Hybrid methods
- Designed to be flexible, portable and easily installed and modified
  - Open-source and public domain
  - Can be compiled on a variety of platforms
  - Part of the WRF Software Framework
- Designed to handle a wide variety of data
  - Conventional observations
  - Radar velocity and reflectivity
  - Satellite (radiance and derived data)
  - Accumulated precipitation

Introduction – What does WRFDA do?

- WRFDA takes a first guess of the atmospheric state, and combines that information with model error and observation information through one of several assimilation methods and background error options to produce a best guess of the atmospheric state at the given time

WRFDA in WRF Modeling System



WRF Model review

- real.exe creates wrfinput_d* and wrfbdy_d01
  - wrfinput_d01 file contains the 3d-initial condition state for the parent domain
  - wrfbdy_d01 contains the lateral boundary conditions for the parent domain
  - For multiple domains, you will have wrfinput_d02, wrfinput_d03, etc., which are the initial conditions for domain 2, domain 3, etc., respectively. Boundary conditions for these files are taken from the parent domains
- wrf.exe creates wrfout_d* files
  - wrfout_d##_YYYY_MM_DD:mm:ss contains one or more 3d forecast states for domain ## starting at the indicated date/time

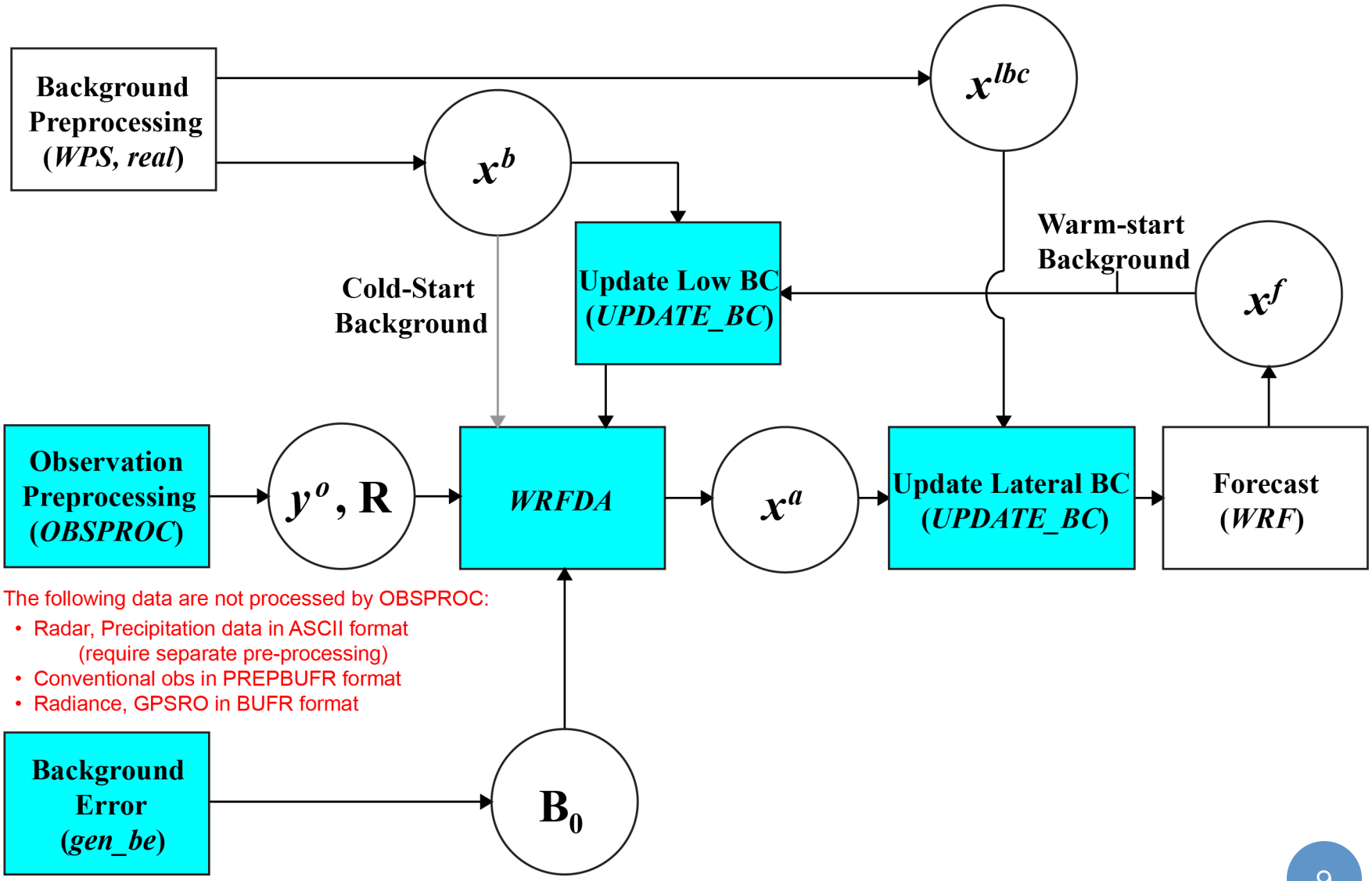
WRFDA in WRF Modeling System

- WRFDA takes a single WRF file (either wrfinput* or wrfout*) and creates a single output file (wrfvar_output)
  - This wrfvar_output file is the updated "best guess" of the atmospheric state after data assimilation
  - wrfvar_output is in the same format as wrfinput files, so can be used to initialize a WRF forecast
  - WRFDA can only use wrfout files which have a single time dimension (In WRF namelist: `frames_per_outfile=1`)
- To perform data assimilation on multiple domains or multiple times, you must run WRFDA multiple times with the appropriate input files

Cycling mode

- Because WRFDA can take WRF forecast files as input, the system can naturally be run in cycling mode
- Output from WRFDA is used to initialize a WRF forecast, the output of which is fed back into WRFDA to initialize another WRF forecast
- Requires boundary condition updating

WRFDA in the WRF Modeling System



The following data are not processed by OBSPROC:

- Radar, Precipitation data in ASCII format (require separate pre-processing)
- Conventional obs in PREPBUFR format
- Radiance, GPSRO in BUFR format

Blue: Supported by WRFDA team

WRFDA System – Outline

- Introduction
- *Compiling the code*
- WRFDA software structure
- Computing overview

Compiling – What is needed?

- WRFDA has similar system requirements to WRF
  - Can be run on a wide variety of UNIX and Linux-based systems
  - Linux/Mac, desktops/laptops, clusters with UNIX-based OS
- WRFDA computational requirements depend on your task
  - Running a small 3DVAR case may take less than 1GB of RAM
  - Large 4DVAR cases may require hundreds of GB
- A supported C and Fortran compiler
  - ifort/icc
  - gfortran/gcc
  - pgf90/pgcc
- Some have known problems; see http://www2.mmm.ucar.edu/wrf/users/wrfda/known-problems.html#compilers

Compiling – What is needed?

- Similar to WRF, there are required and optional libraries
  - netCDF C/fortran libraries are required, and must be downloaded and built by the user
    - http://www.unidata.ucar.edu/downloads/netcdf/index.jsp
  - MPI libraries (e.g. MPICH) are required for running WRFDA in parallel
  - BUFR libraries are required for reading PREPBUFR or radiance BUFR files, but they are included in WRFDA and built automatically

Compiling – What is needed?

- Similar to WRF, there are required and optional libraries
  - For radiance assimilation, a radiative transfer model is needed:
    - CRTM, the Community Radiative Transfer Model, is included with the WRFDA source code
    - RTTOV is provided by EUMETSAT/NWP SAF, and must be downloaded and built separately
      - https://nwpsaf.eu/deliverables/rtm/rtm_rttov11.html
  - New in version 3.8: AMSR2 radiance files in HDF5 format
    - HDF5 libraries are maintained by The HDF5 Group, and must be downloaded and built separately
      - https://www.hdfgroup.org/HDF5/

Compiling – Getting the source code

- Visit the WRFDA download website:
  - http://www2.mmm.ucar.edu/wrf/users/wrfda/download/get_source.html
- Click "New Users" and fill out the registration form, (registration is free), or
- Click "Returning users" and enter your email if you have previously registered to download a WRF product
- Download the latest tar file (Version 3.8)
- Unzip (`gunzip WRFDA_V3.8.tar.gz`) and un-tar (`tar -xvf WRFDA_V3.8.tar`) the code package
- You should see a directory named "WRFDA"; this is the WRFDA source code

WRFDA Directory structure

```
arch
clean
compile
configure
dyn_em
dyn_exp
external
frame
inc
main
Makefile
phys
README.DA
README.io_config
Registry
run
share
test
tools
var
```

build scripts

README file with information about WRFDA

Contains `registry.var`

WRFDA source code directory

Legend:
Blue – directory
Green – script file
Gray – other text file

WRFDA/var Directory structure

build	*Executables built here*
convertor	
da	*WRFDA main source code contained here*
external	*Source code for external libraries (CRTM, BUFR, etc.)*
gen_be	*GEN_BE source code*
graphics	
Makefile	
obsproc	*OBSPROC source code*
README.basics	*More README files with useful information*
README.namelist	
README.radiance	
run	*Useful runtime files (mostly for radiance)*
scripts	
test	*Data for tutorial cases*

Legend:
Blue – directory
Green – script file
Gray – other text file

WRFDA/var/da Directory structure

Main WRFDA Program (driver): `da_main`

WRFDA Subroutines (mediation layer)

- `da_4dvar`
- `da_control`
- `da_etkf`
- `da_define_structures`
- `da_dynamics`
- `da_grid_definitions`
- `da_interpolation`
- `da_minimisation`
- `da_physics`
- `da_setup_structures`
- `da_varbc`
- `da_vtox_transforms`

Observation Types

- `da_airep`
- `da_airsr`
- `da_bogus`
- `da_buoy`
- `da_geoamv`
- `da_gpspw`
- `da_gpsref`
- `da_metar`
- `da_mtgirs`
- `da_pilot`
- `da_polaramv`
- `da_profiler`
- `da_pseudo`
- `da_qscat`
- `da_radar`
- `da_radiance`
- `da_rain`
- `da_satem`
- `da_ships`
- `da_sound`
- `da_ssmi`
- `da_synop`
- `da_tamdar`

Compiling – Preparing the environment

- As mentioned before, some libraries are required for WRFDA, and some are optional depending what you are using WRFDA for
  - netCDF is required; you should set an environment variable to specify where the netCDF libraries are built on your system:
  - `setenv NETCDF full_path_for_NETCDF`
- If you plan on doing radiance assimilation, you will need CRTM or RTTOV. WRFDA can be built with either or both
  - The CRTM source code is included in the WRFDA package, use `setenv CRTM 1` to build it
  - To use RTTOV, set an environment variable specifying where RTTOV is built on your system:
  - `setenv RTTOV full_path_for_RTTOV`

Compiling – Preparing the environment

- If you plan on assimilating AMSR2 data, you will need to link to the HDF5 libraries
  - Set an environment variable specifying where HDF5 is built on your system:
  - `setenv HDF5 full_path_for_HDF5`
- To build the code faster, if your computer has the gnu make utility, you can set the environment variable J to build the code in parallel
  - `setenv J "-j 4"` (will build on 4 processors)
  - Note that this is different from compiling the code to *run* in parallel

Compiling – Building the WRFDA code

- Two scripts must be run to build the code:
- `configure` asks for some information about your machine and how you want to build the code, and generates a `configure.wrf` file
- `./configure wrfda`

```
> ./configure wrfda
checking for perl5... no
checking for perl... found /usr/bin/perl (perl)
Will use NETCDF in dir: /usr/local/netcdf-3.6.3-gfortran
Will use HDF5 in dir: /usr/local/hdf5-1.8.15-gcc
PHDF5 not set in environment. Will configure WRF for use without.
Will use 'time' to report timing information
$JASPERLIB or $JASPERINC not found in environment, configuring to build without grib2 I/O...
------------------------------------------------------------------------
Please select from among the following Linux x86_64 options:

  1. (serial)   2. (smpar)   3. (dmpar)   4. (dm+sm)   PGI (pgf90/gcc)
  5. (serial)   6. (smpar)   7. (dmpar)   8. (dm+sm)   PGI (pgf90/pgcc): SGI MPT
  9. (serial)  10. (smpar)  11. (dmpar)  12. (dm+sm)   PGI (pgf90/gcc): PGI accelerator
 13. (serial)  14. (smpar)  15. (dmpar)  16. (dm+sm)   INTEL (ifort/icc)
 ... ... ...
```

- Select the option that is best for your purposes

Compiling – Building the WRFDA code

- Two scripts must be run to build the code:
- `compile` compiles all the code for the settings you specified

```
./compile all_wrfvar >& compile.wrfda.log
```

- Depending on your machine and what options you have selected, compilation can take less than 5 minutes up to an hour. For example, gfortran compiles WRFDA quite quickly, while intel compilers take longer to build (but the executables may run faster)

Compiling – review compiled code

- When the compilation script is completed, you should see the message "build completed:" followed by the date and time.
- The script does not automatically check to make sure all executables were successfully built; you will need to check manually
- There should be 44 executables built all together: 43 in the WRFDA/var/build directory, and WRFDA/var/obsproc/obsproc.exe
- In all likelihood, you will not use most of these directly: the majority of them are called by scripts for various diagnostic packages

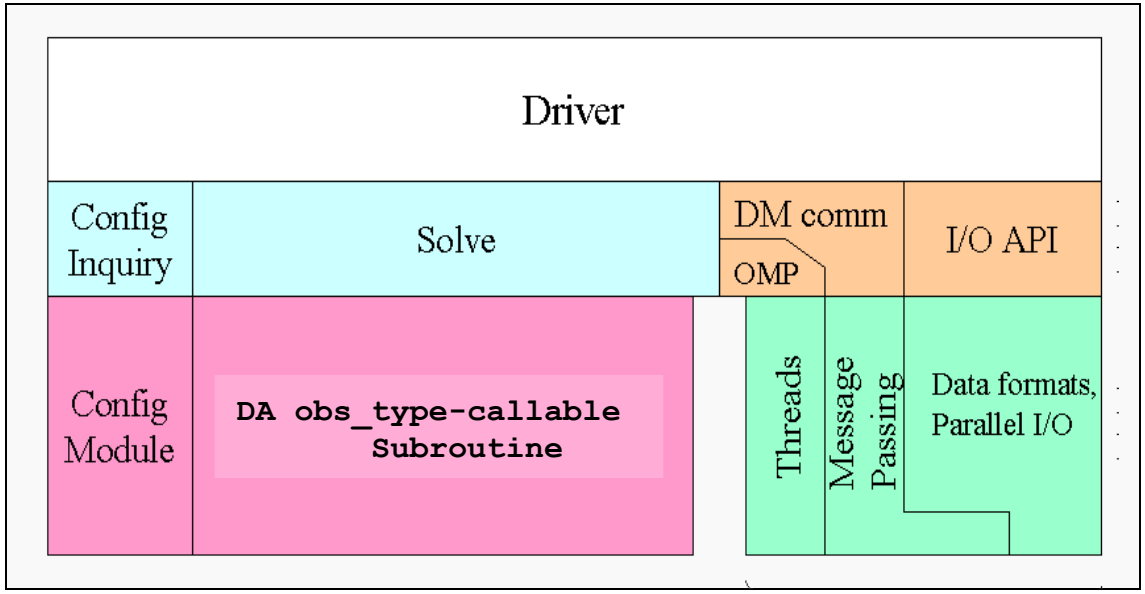
Compiling – review executables

- These are the executables you will most likely be using:
- da_wrfvar.exe
  - The main WRFDA executable: this program will perform the actual data assimilation and output a WRF-formatted wrfvar_output file
- obsproc.exe
  - The executable for OBSPROC, the observation pre-processor for text-based observation formats
- da_update_bc.exe
  - The executable for UPDATE_BC; used for updating boundary conditions after assimilation and during cycling runs

WRFDA System – Outline

- Introduction
- Compiling the code
- *WRFDA software structure*
- Computing overview

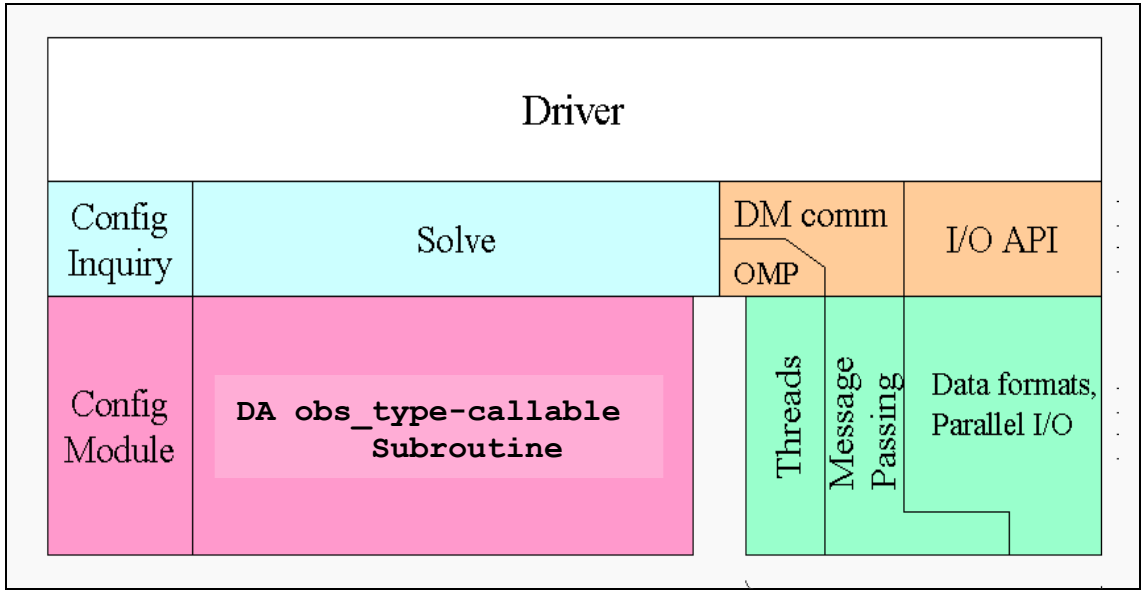
WRFDA Software – Architecture



Registry.wrfvar

- Hierarchical software architecture
  - Insulate scientists' code from parallelism and other architecture/implementation-specific details
  - Well-defined interfaces between layers, and external packages for communications, I/O.

WRFDA Software – Architecture



Registry.wrfvar

- Registry: an "Active" data dictionary
  - Tabular listing of model state and attributes
  - Large sections of interface code generated automatically
  - Scientists manipulate model state simply by modifying Registry, without further knowledge of code mechanics
  - **registry.var** is the main dictionary for WRFDA
  - registry.var is combined at compile time with Registry.EM_COMMON.var and others to produce Registry.wrfvar, which contains all of the registry definitions used by WRFDA

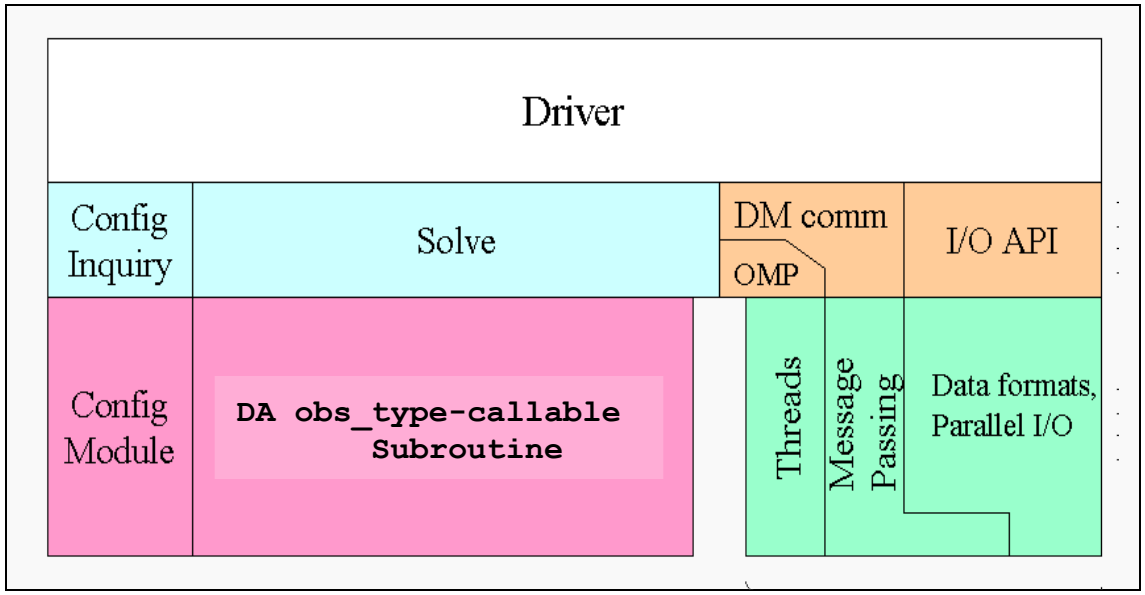
WRFDA Software – Architecture

Variable type | Variable name | Namelist name | Variable size | Default value

registry.var

```
rconfig   integer   rttov_emis_atlas_ir     namelist,wrfvar14  1   0        - "rttov_emis_atlas_ir"      ""   "\"
rconfig   integer   rttov_emis_atlas_mw     namelist,wrfvar14  1   0        - "rttov_emis_atlas_mw"      ""   "\"
rconfig   integer   rtminit_print           namelist,wrfvar14  1   1        - "rtminit_print"            ""   "\"
rconfig   integer   rtminit_nsensor         namelist,wrfvar14  1   1        - "rtminit_nsensor"          ""   "\"
rconfig   integer   rtminit_platform        namelist,wrfvar14  max_instruments -1       - "rtminit_platform"          ""
rconfig   integer   rtminit_satid           namelist,wrfvar14  max_instruments -1.0     - "rtminit_satid"             ""
rconfig   integer   rtminit_sensor          namelist,wrfvar14  max_instruments -1.0     - "rtminit_sensor"            ""
rconfig   integer   rad_monitoring          namelist,wrfvar14  max_instruments 0        - "rad_monitoring"            ""
rconfig   real      thinning_mesh           namelist,wrfvar14  max_instruments 60.0     - "thinning_mesh"             ""
rconfig   logical   thinning                namelist,wrfvar14  1   .true.   - "thinning "                 ""   "\"
rconfig   logical   read_biascoef           namelist,wrfvar14  1   .false.  - "read_biascoef"             ""   "\"
rconfig   logical   biascorr                namelist,wrfvar14  1   .false.  - "biascorr"                  ""   "\"
rconfig   logical   biasprep                namelist,wrfvar14  1   .false.  - "biasprep"                  ""   "\"
rconfig   logical   rttov_scatt             namelist,wrfvar14  1   .false.  - "rttov_scatt"               ""   "\"
rconfig   logical   write_profile           namelist,wrfvar14  1   .false.  - "write_profile"             ""   "\"
rconfig   logical   write_jacobian          namelist,wrfvar14  1   .false.  - "write_jacobian"            ""   "\"
rconfig   logical   qc_rad                  namelist,wrfvar14  1   .true.   - "qc_rad"                    ""   "\"
rconfig   logical   write_iv_rad_ascii      namelist,wrfvar14  1   .false.  - "write_iv_rad_ascii"        ""   "\"
rconfig   logical   write_oa_rad_ascii      namelist,wrfvar14  1   .false.  - "write_oa_rad_ascii"        ""   "\"
rconfig   logical   write_filtered_rad      namelist,wrfvar14  1   .false.  - "write_filtered_rad"        ""   "\"
rconfig   logical   use_error_factor_rad    namelist,wrfvar14  1   .false.  - "use_error_factor_rad"      ""   "\"
rconfig   logical   use_landem              namelist,wrfvar14  1   .false.  - "use_landem"                ""   "\"
rconfig   logical   use_antcorr             namelist,wrfvar14  max_instruments .false.  - "use_antcorr"           ""   "
rconfig   logical   use_mspps_emis          namelist,wrfvar14  max_instruments .false.  - "use_mspps_emis"        ""   "
rconfig   logical   use_mspps_ts            namelist,wrfvar14  max_instruments .false.  - "use_mspps_ts"          ""   "
```

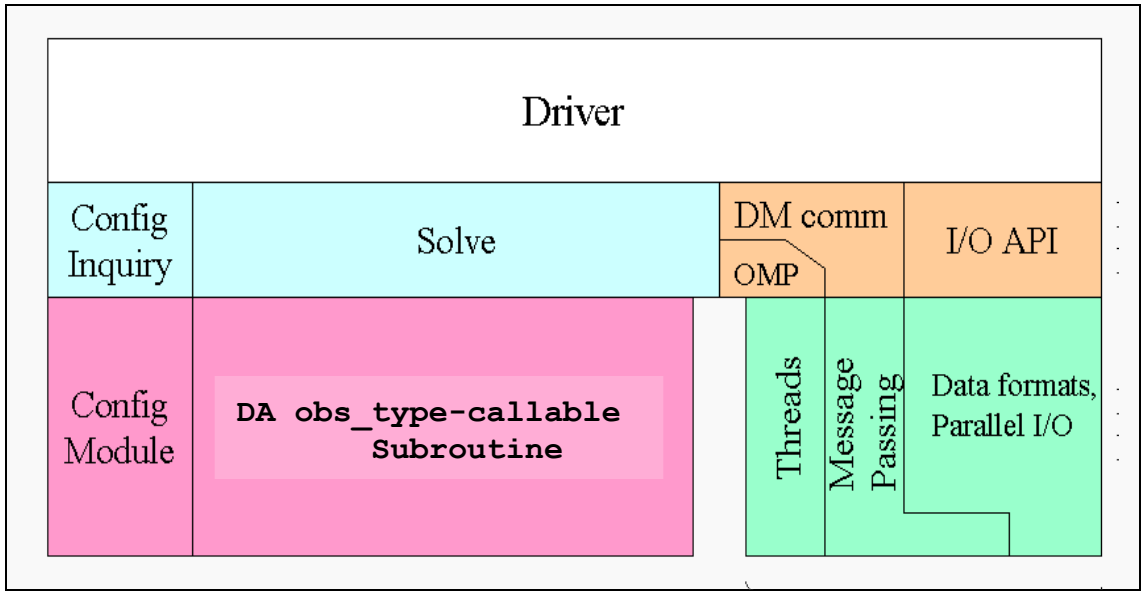
WRFDA Software – Architecture



Registry

- Driver Layer
  - **Domains**: Allocates, stores, decomposes, represents abstractly as single data objects

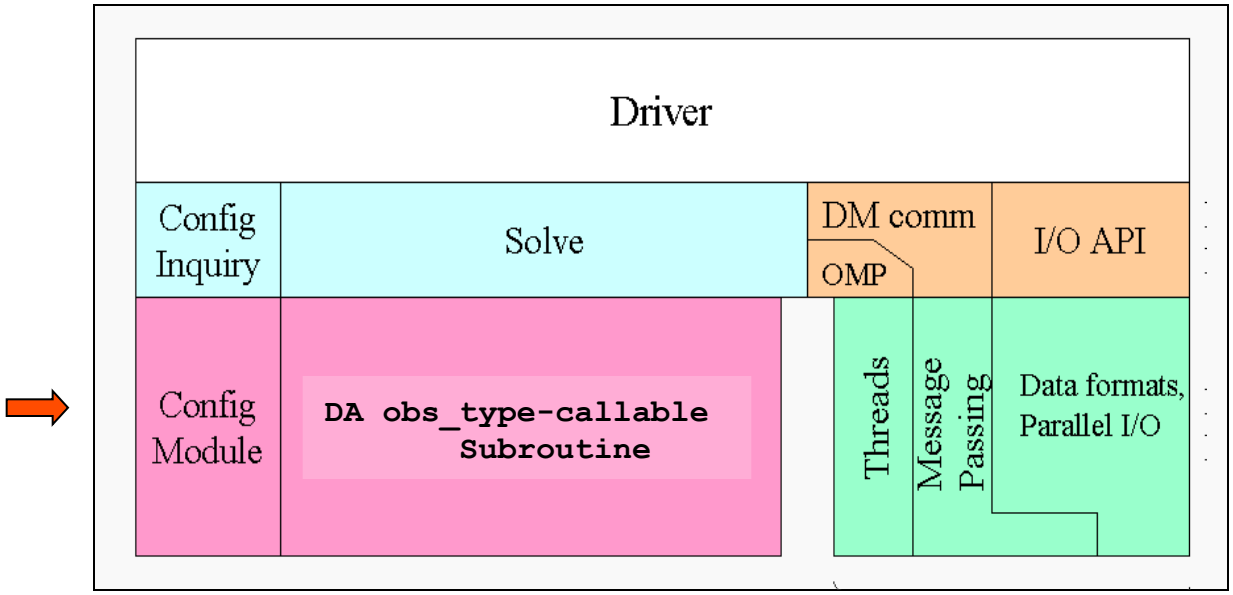
WRFDA Software – Architecture



Registry

- Minimization/Solver Layer
  - Minimization/Solver routine, choose the function based on the namelist variable, 3DVAR, 4DVAR, FSO or Verification, and choose the minimization algorithm.

WRFDA Software – Architecture

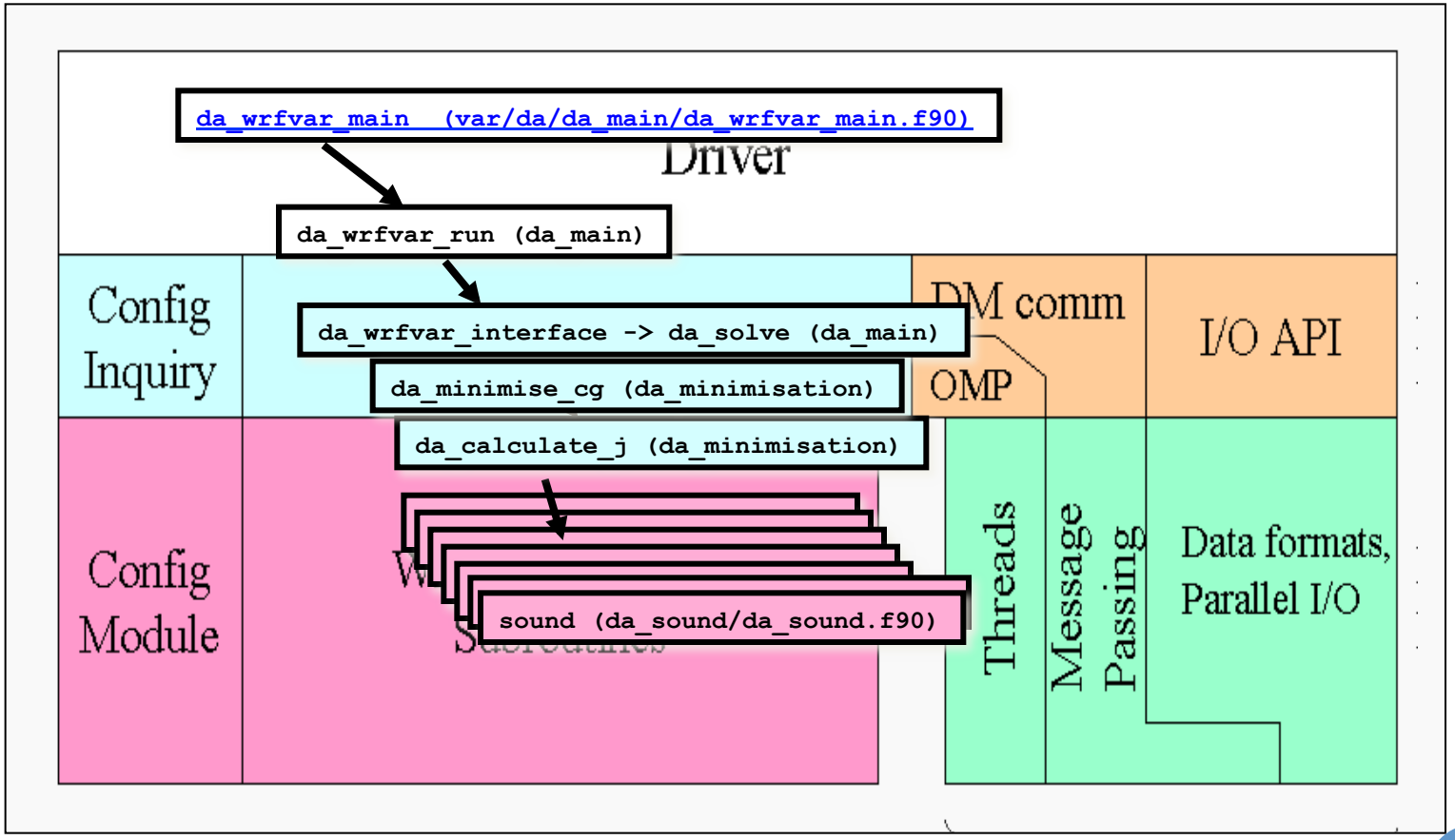


Registry

- Observation Layer
  - **Observation interfaces**: contains the gradient and cost function calculation subroutines for each type of observations.

Call Structure Superimposed on Architecture

da_sound.f90(da_sound)



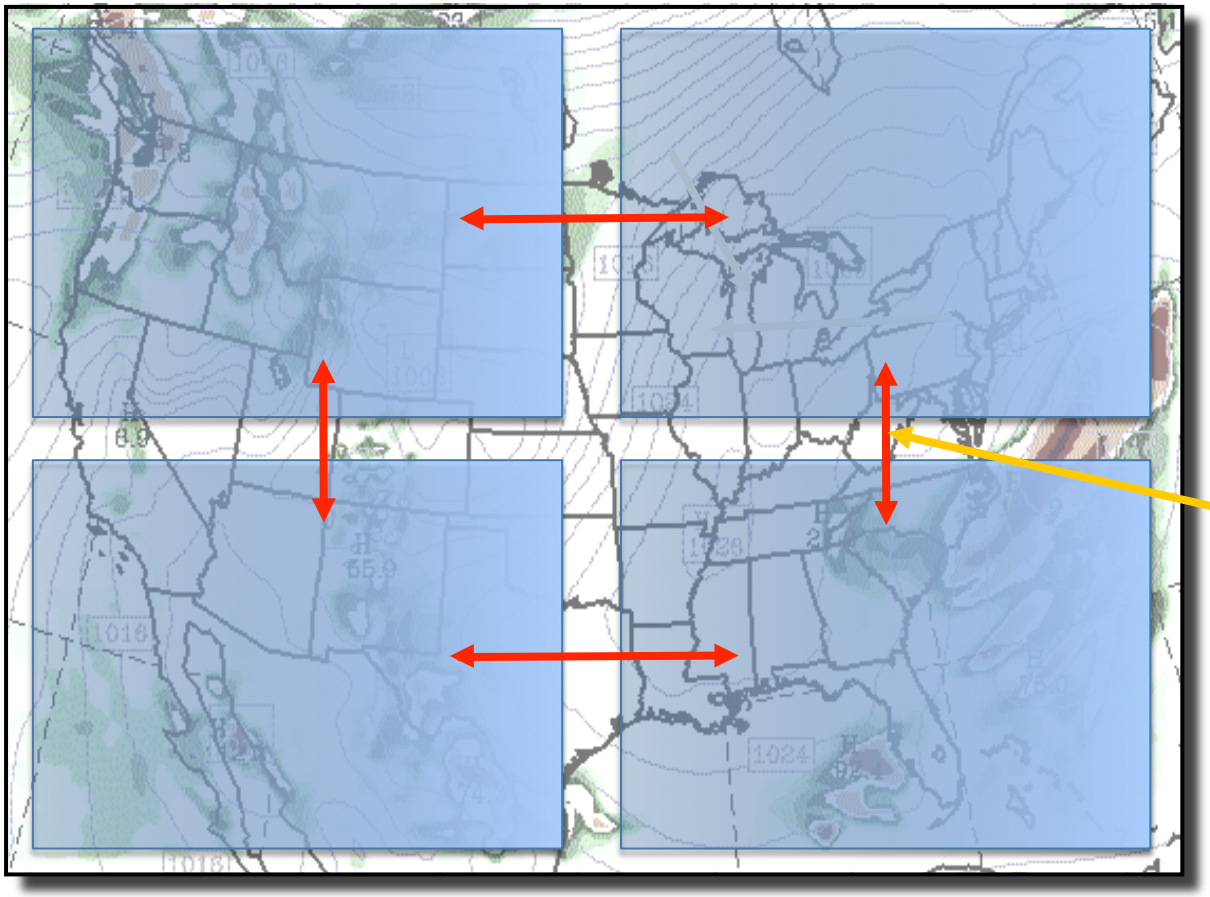
WRFDA System – Outline

- Introduction
- Compiling the code
- WRFDA software overview
- *Computing overview*

WRFDA Parallelism

- WRFDA can be run serially or as a parallel job
- WRFDA uses ***domain decomposition*** to divide total amount of work over parallel processes
- The decomposition of the application over processes has two levels:
  - The ***domain*** is broken up into rectangular pieces that are assigned to MPI (distributed memory) processes. These pieces are called ***patches***
  - The ***patches*** may be further subdivided into smaller rectangular pieces that are called ***tiles***, and these are assigned to ***shared-memory threads*** within the process.
- *However, WRFDA does not support shared memory parallelism! So distributed memory is what I will cover here.*

Parallelism in WRFDA: Multi-level Decomposition



Inter-processor communication

Distributed Memory Communications

When Needed?

Communication is required between patches when a horizontal index is incremented or decremented on the right-hand-side of an assignment.

Why?

On a patch boundary, the index may refer to a value that is on a different patch.

Following is an example code fragment that requires communication between patches

Signs in code

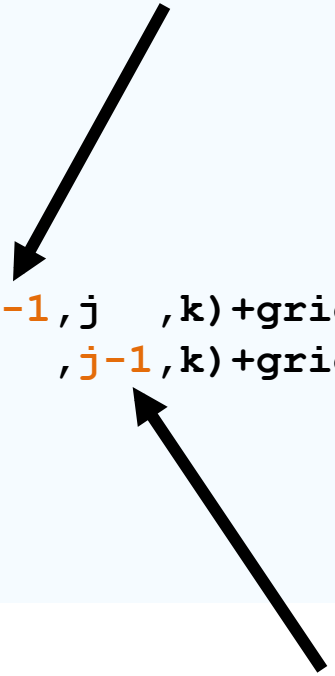
Note the tell-tale +1 and –1 expressions in indices for rr, H1, and H2 arrays on right-hand side of assignment.

These are *horizontal data dependencies* because the indexed operands may lie in the patch of a neighboring processor. That neighbor's updates to that element of the array won't be seen on this processor.

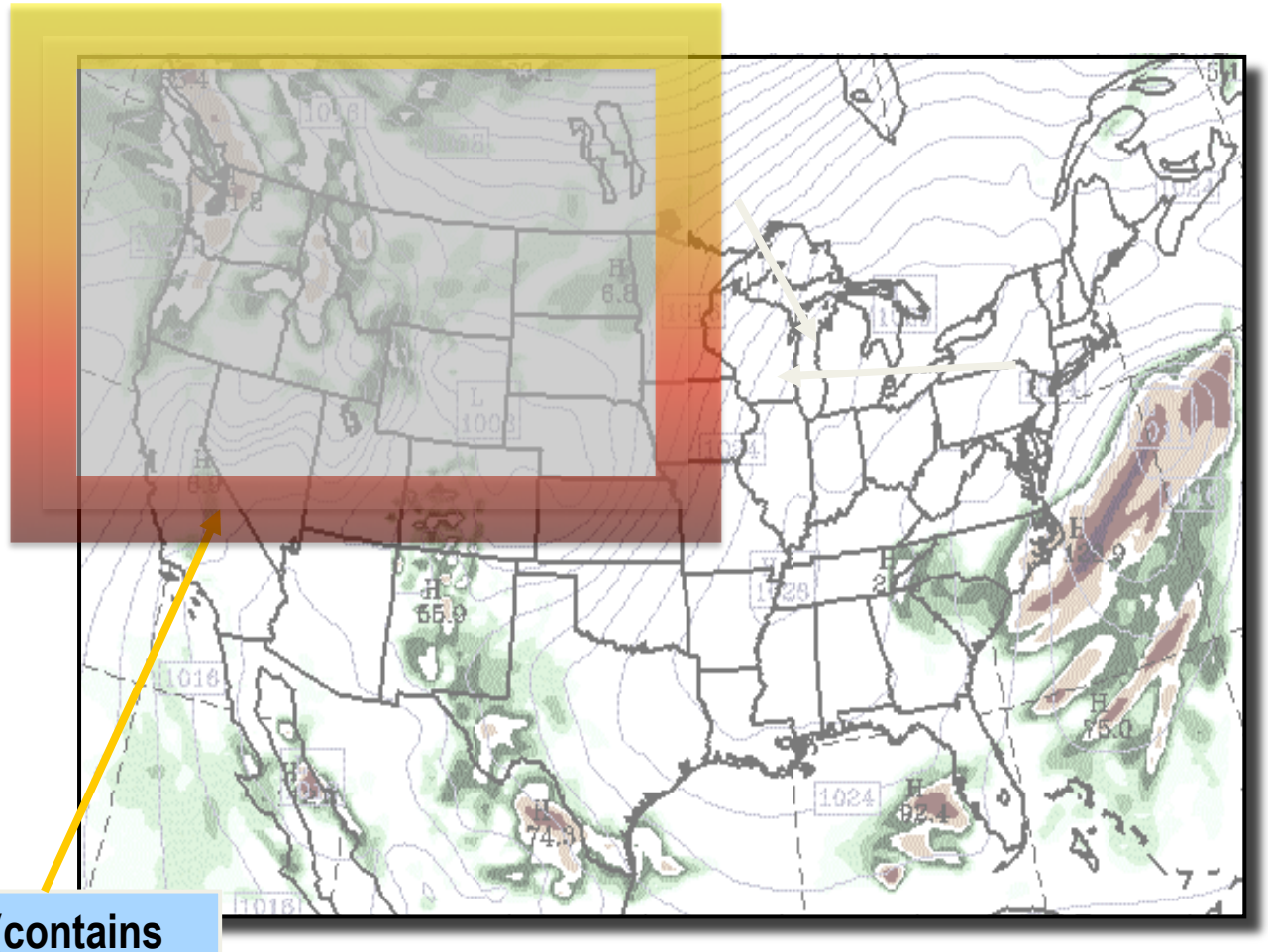
Distributed Memory Communications

(da_transfer_xatowrf.inc)

```
subroutine da_transfer_xatowrf(grid)
. . .
 do k=kts,kte
      do j=jts,jte+1
         do i=its,ite+1
            u_cgrid(i,j,k)=0.5*(grid%xa%u(i-1,j  ,k)+grid%xa%u(i,j,k))
            v_cgrid(i,j,k)=0.5*(grid%xa%v(i  ,j-1,k)+grid%xa%v(i,j,k))
         end do
      end do
   end do
 . . .
```

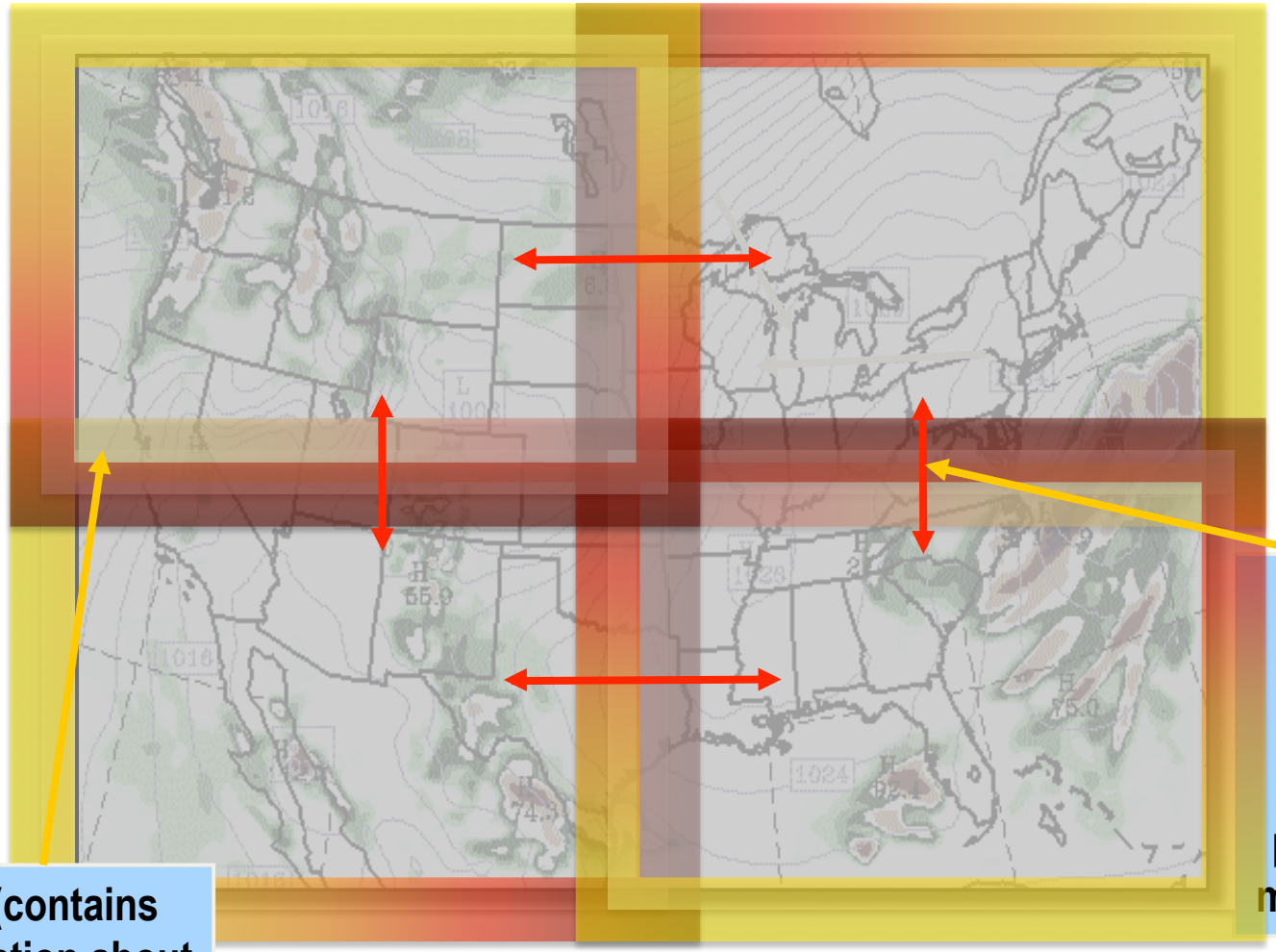


Distributed Memory Communications



Halo (contains information about adjacent patch)

Distributed Memory Communications



Halo (contains information about adjacent patch)

Inter-processor communication

(Halos update from adjacent patch after each minimization step)

Grid Representation in Arrays

- Increasing indices in WRFDA arrays run
  - West to East (X, or I-dimension)
  - South to North (Y, or J-dimension)
  - Bottom to Top (Z, or K-dimension)
- Storage order in WRFDA is IJK , but for WRF, it is IKJ (ARW) and IJK (NMM)
- Output data has grid ordering independent of the ordering inside the WRFDA model

Grid Representation in Arrays

- The extent of the logical or *domain* dimensions is always the "staggered" grid dimension. That is, from the point of view of a non-staggered dimension (also referred to as the ARW “mass points”), there is always an extra cell on the end of the domain dimension
- In WRFDA, the minimization is on A-grid (non-staggered grid). The wind components will be interpolated from A-grid to C-grid (staggered grid) before they are output, to conform with standard WRF format

Summary

- WRFDA
  - is designed to be an easy-to-use data assimilation system for use with the WRF model
  - is designed within the WRF Software Framework for rapid development and ease of modification
  - is compiled in much the same way as WRF
  - can be run in parallel for quick assimilation of large amounts of data on large domains

Appendix – WRFDA Resources

- WRFDA users page
  - http://www2.mmm.ucar.edu/wrf/users/wrfda
  - Download WRFDA source code, test data, related packages and documentation
  - Lists WRFDA news and developments
- Online documentation
  - http://www2.mmm.ucar.edu/wrf/users/docs/user_guide_V3/users_guide_chap6.htm
  - Chapter 6 of the WRF Users' Guide; documents installation of WRFDA and running of various WRFDA methods
- WRFDA user services and help desk
  - wrfhelp@ucar.edu

Appendix – WRFDA History

- Developed from MM5 3DVar beginning around 2002, first version (2.0) released December 2003
- 4DVAR capability added in 2008, made practical with parallelism starting with Version 3.4 (April 2012)
- Developed and supported by WRFDA group of the Mesoscale and Microscale Meteorology Lab of NCAR
- Requirements emphasize flexibility over a range of platforms, applications, users, performance
- Current release WRFDA v3.8 (April 2016)
- Shares the WRF Software Framework