WRF Data Assimilation System: Software and Compilation

Michael Kavulich, Jr.

WRFDA Tutorial, August 2015, NCAR

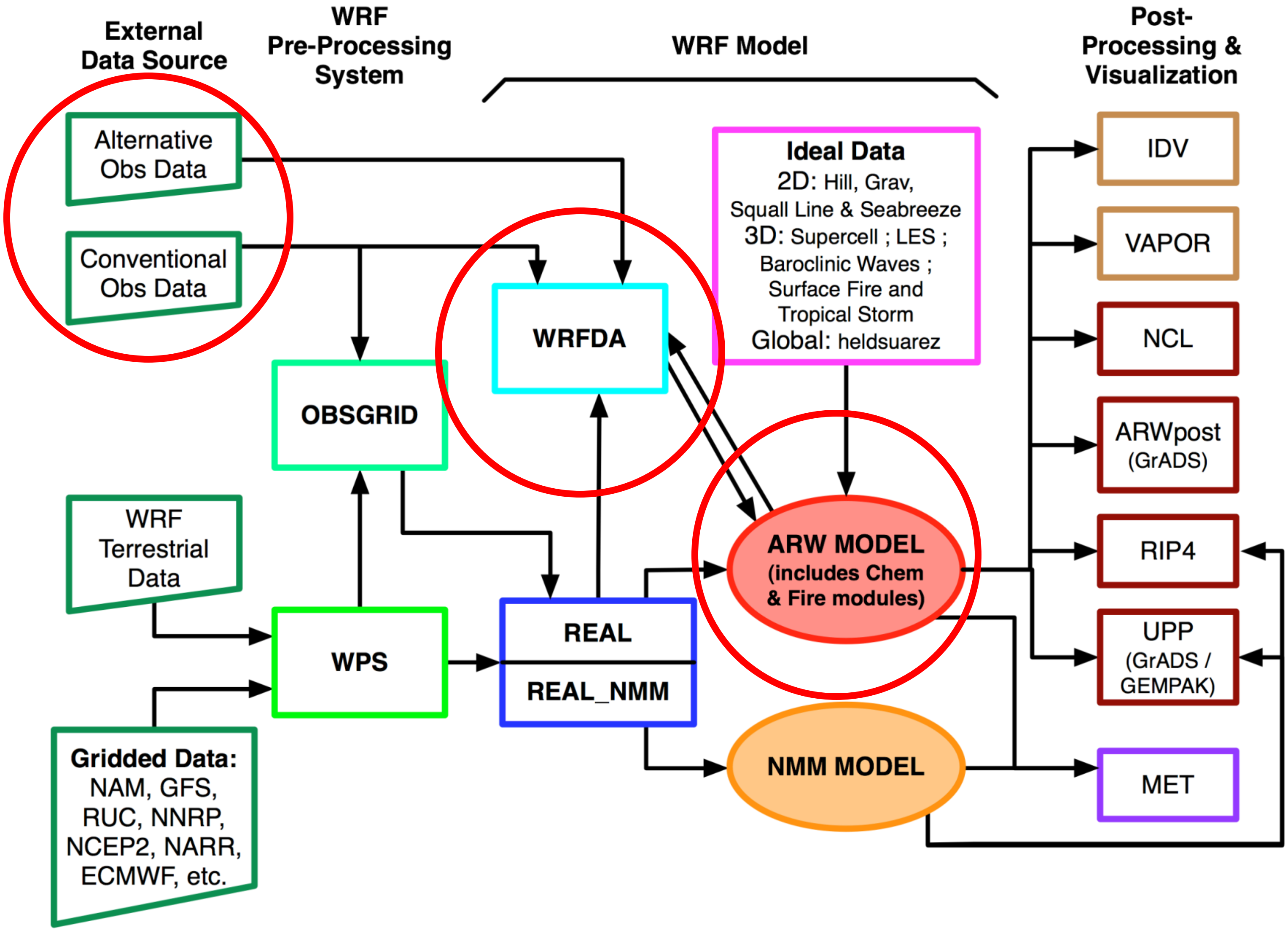
WRFDA System – Outline

- *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

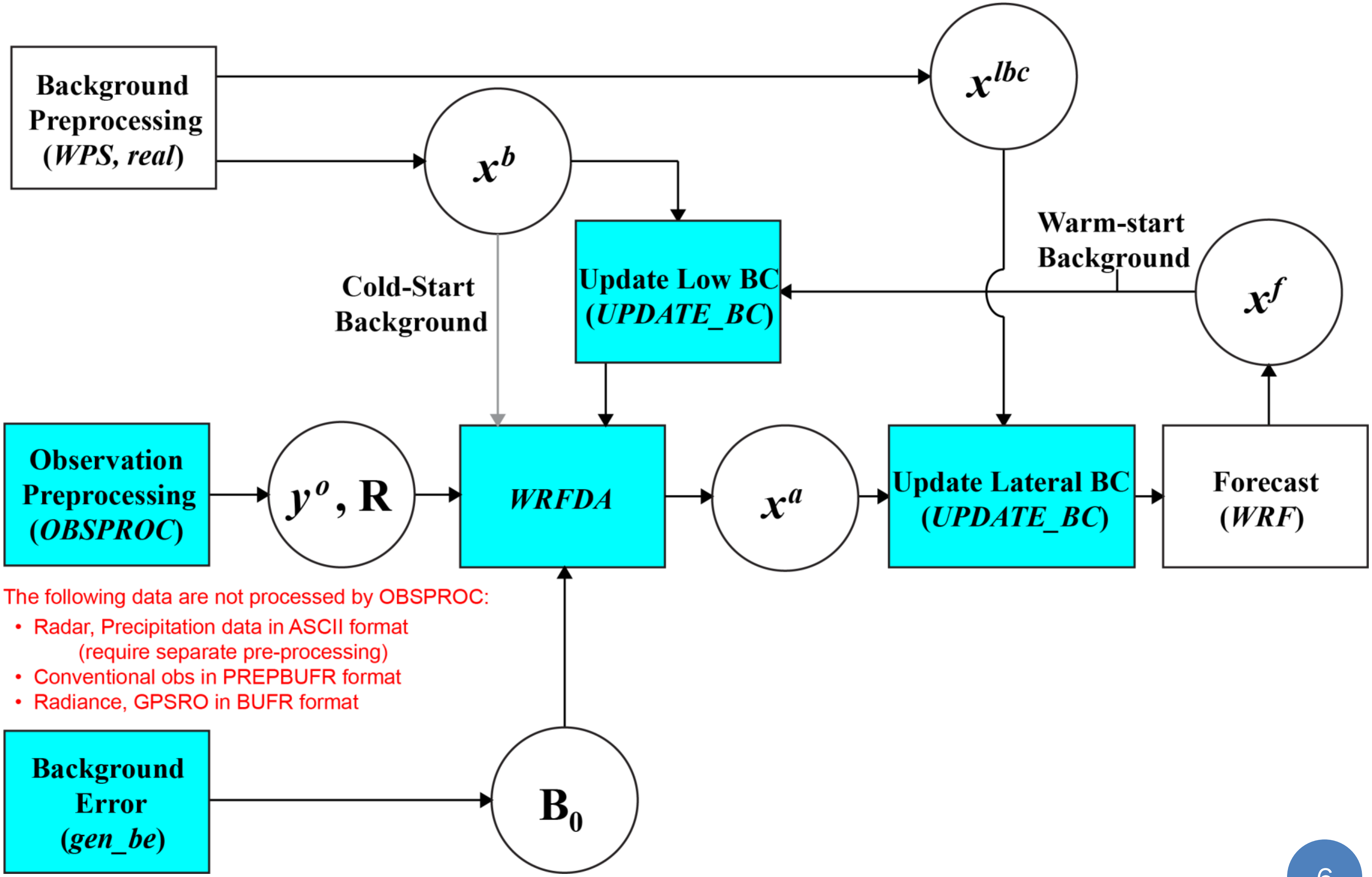
WRFDA in WRF Modeling System


External Data Source
WRF Pre-Processing System
WRF Model
Post-Processing & Visualization
Alternative Obs Data
Conventional Obs Data
WRFDA
Ideal Data
2D: Hill, Grav, Squall Line & Seabreeze
3D: Supercell ; LES ; Baroclinic Waves ; Surface Fire and Tropical Storm
Global: heldsuarez
IDV
VAPOR
NCL
ARWpost (GrADS)
OBSGRID
WRF Terrestrial Data
ARW MODEL (includes Chem & Fire modules)
RIP4
WPS
REAL
REAL_NMM
UPP (GrADS / GEMPAK)
Gridded Data: NAM, GFS, RUC, NNRP, NCEP2, NARR, ECMWF, etc.
NMM MODEL
MET

Cycling mode

- Because WRFDA takes WRF forecast files as input, the system can naturally be run in cycling mode
- WRFDA initializes 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


Background Preprocessing (*WPS, real*)
x^{lbc}
x^b
Update Low BC (*UPDATE_BC*)
Warm-start Background
x^f
Cold-Start Background
Observation Preprocessing (*OBSPROC*)
y^o, R
WRFDA
x^a
Update Lateral BC (*UPDATE_BC*)
Forecast (*WRF*)
Background Error (*gen_be*)
B_0

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, such as MPICH, are required for running WRFDA in parallel
  - 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/NWC SAF, and must be downloaded and built separately
      - https://nwpsaf.eu/deliverables/rtm/rtm_rttov11.html
  - BUFR libraries are required for reading PREPBUFR or radiance BUFR files, but they are included in WRFDA and built automatically

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 3.7)
- Unzip (`gunzip WRFDA_V3.7.tar.gz`) and un-tar (`tar -xvf WRFDA_V3.7.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
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_pseudo
da_airsr        da_qscat
da_bogus        da_radar
da_buoy         da_radiance
da_geoamv       da_rain
da_gpspw        da_satem
da_gpsref       da_ships
da_metar        da_sound
da_mtgirs       da_ssmi
da_pilot        da_synop
da_polaramv     da_tamdar
da_profiler
```

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`
- To build 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)

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
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 will 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

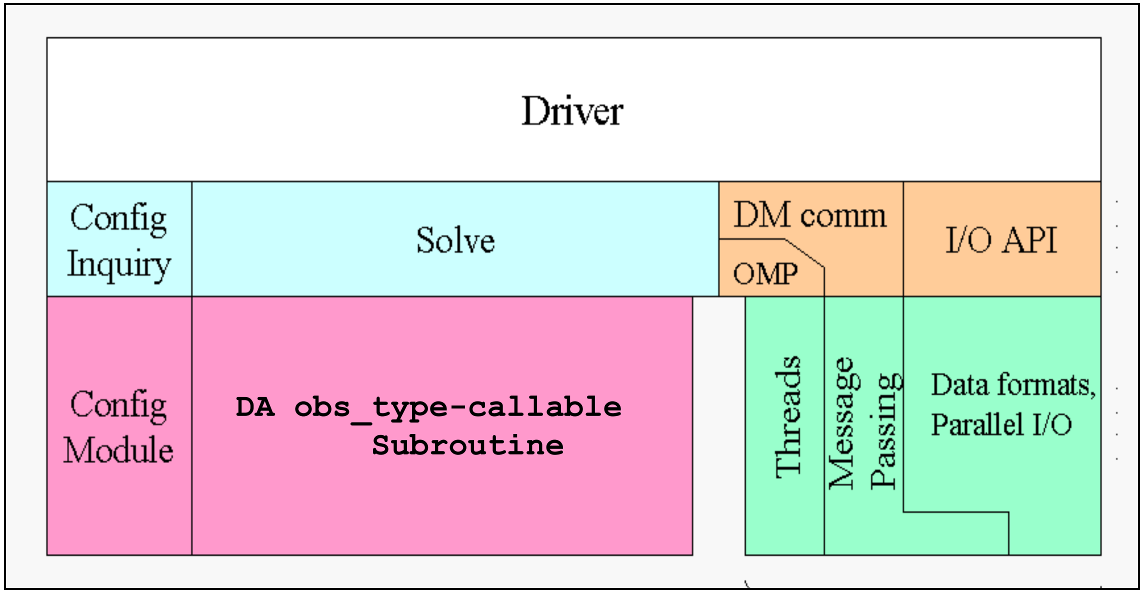
Compiling – review executables

- 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/minimization
- 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

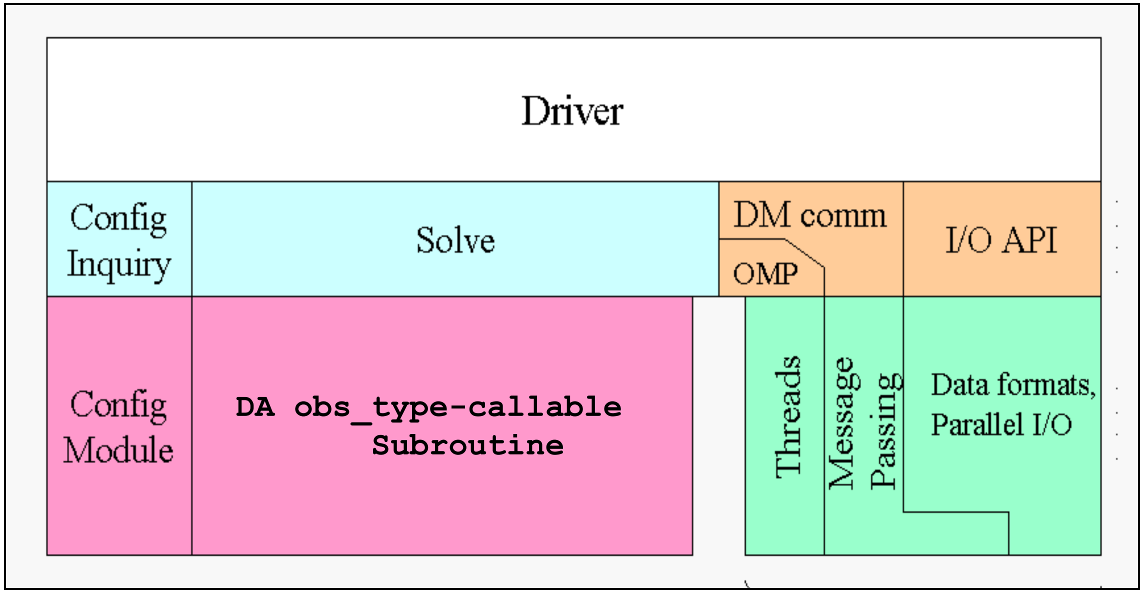
WRFDA Software – Architecture



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: 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

registry.var

Variable type

Variable name

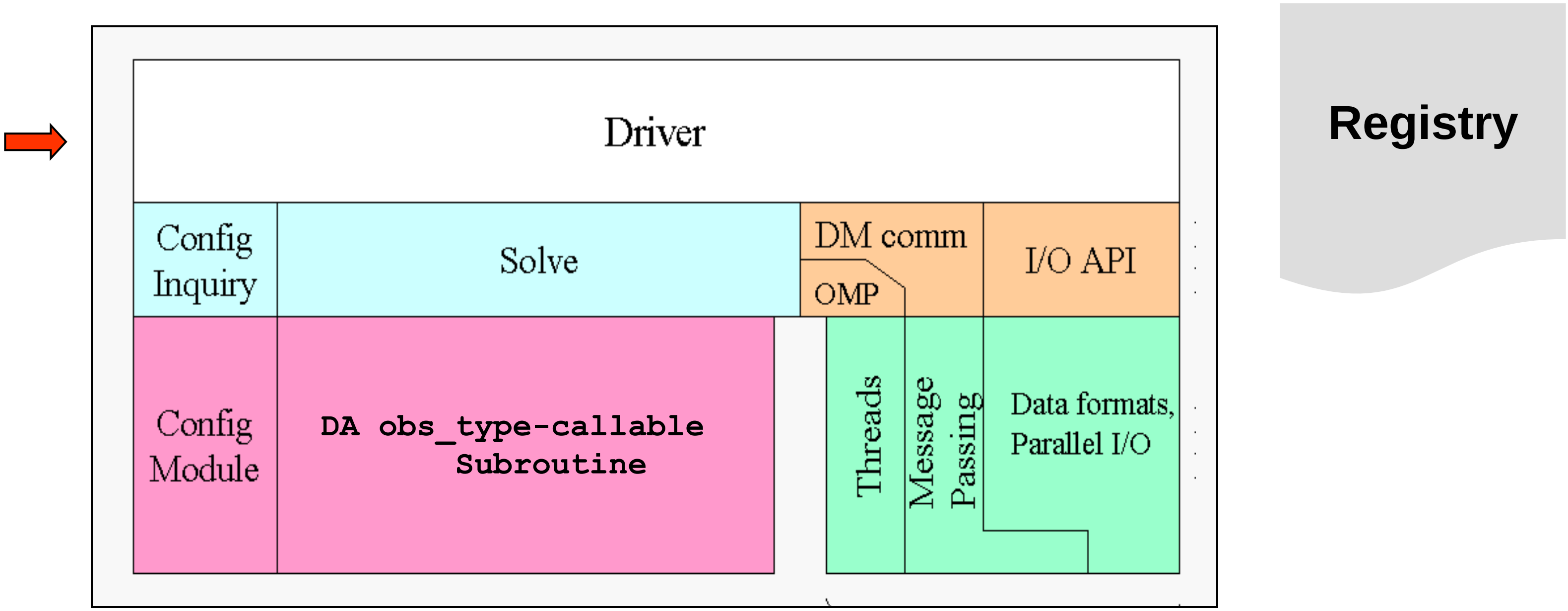
Namelist name

Variable size

Default value

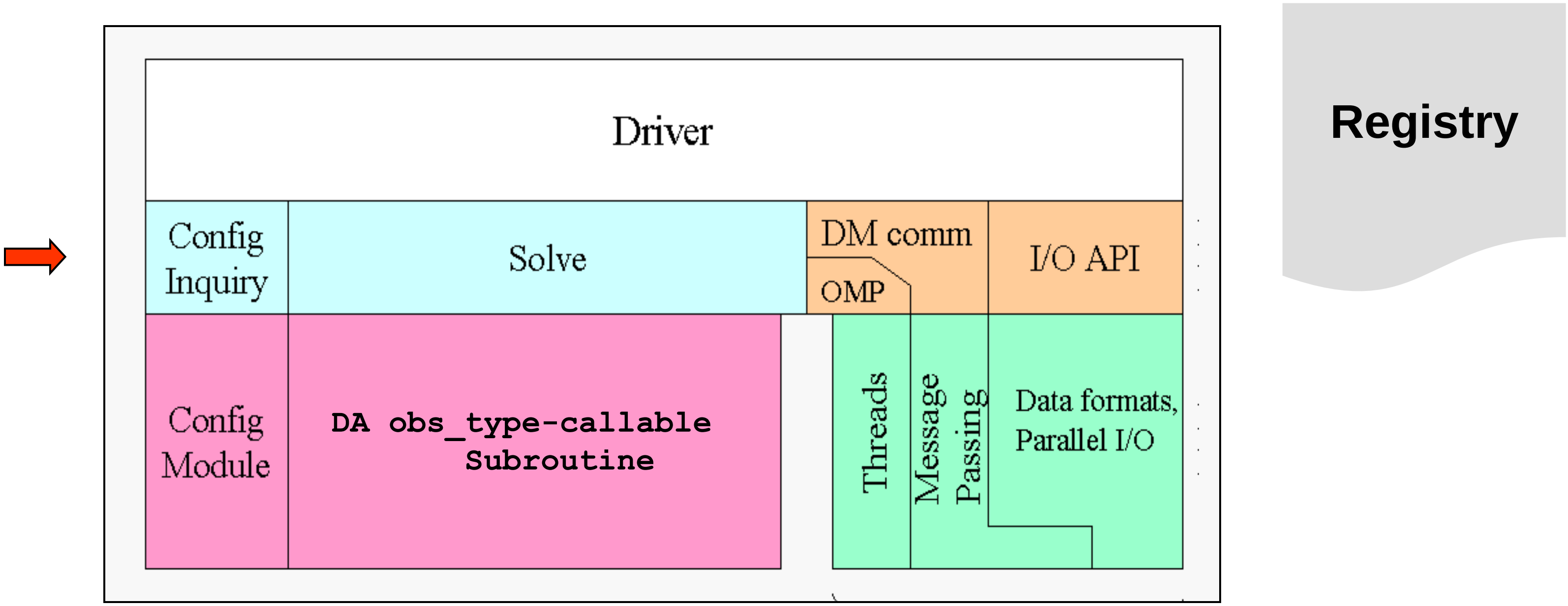
```
rconfig   integer   rttov_emis_atlas_ir      namelist,wrfvar14  1   0        - "rttov_emis_atlas_ir"      ""  ""
rconfig   integer   rttov_emis_atlas_mw      namelist,wrfvar14  1   0        - "rttov_emis_atlas_mw"      ""  ""
rconfig   integer   rtminit_print            namelist,wrfvar14  1   1        - "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



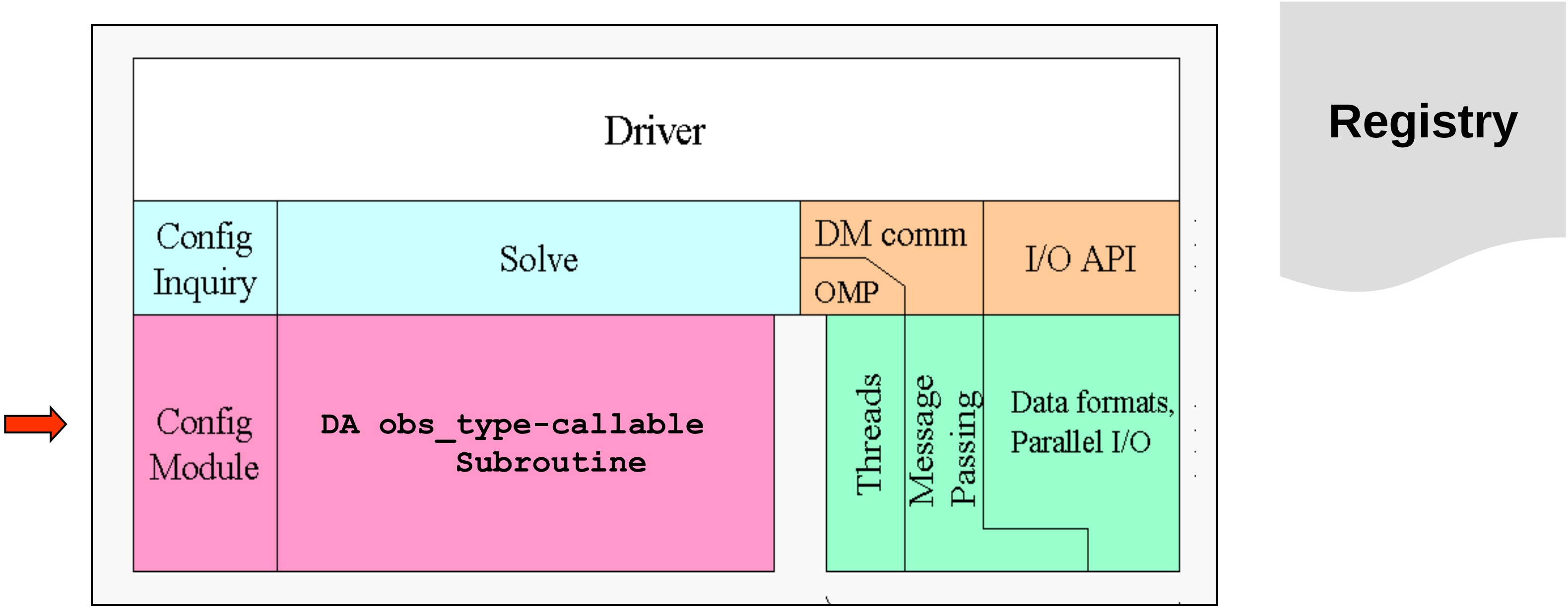
- Driver Layer
  - **Domains**: Allocates, stores, decomposes, represents abstractly as single data objects

WRFDA Software – Architecture



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

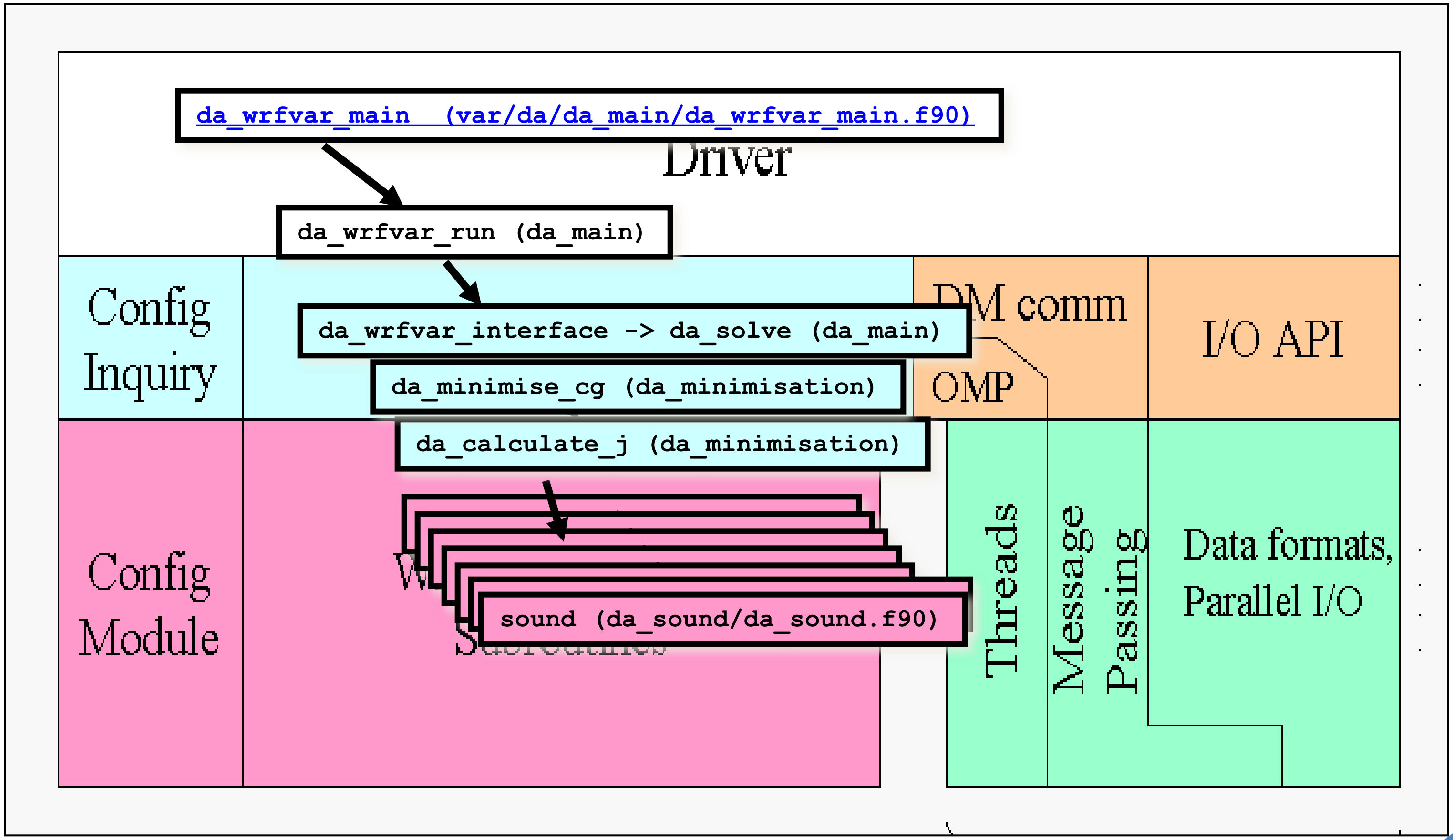
WRFDA Software – Architecture



- 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)



da_wrfvar_main (var/da/da_main/da_wrfvar_main.f90)

da_wrfvar_run (da_main)

da_wrfvar_interface -> da_solve (da_main)

da_minimise_cg (da_minimisation)

da_calculate_j (da_minimisation)

sound (da_sound/da_sound.f90)

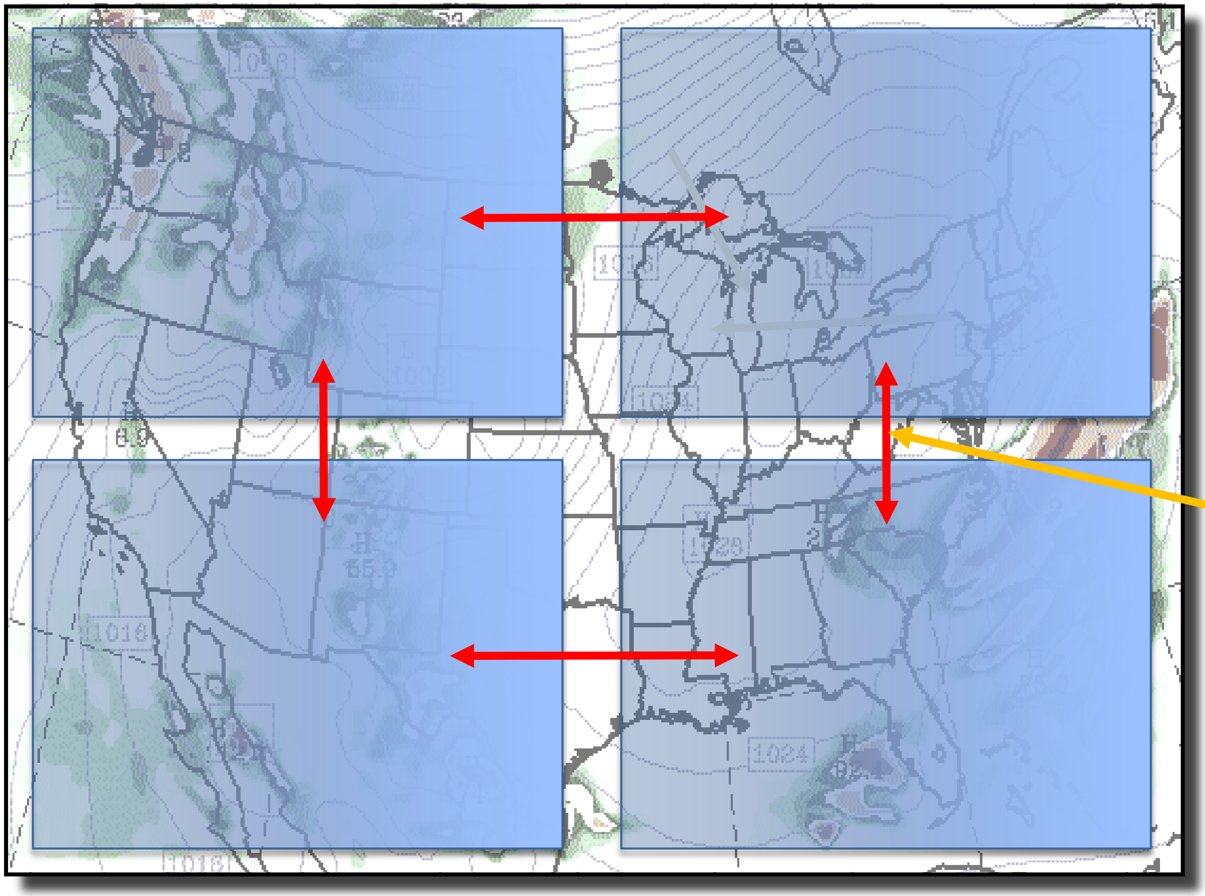
WRFDA System – Outline

- 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.

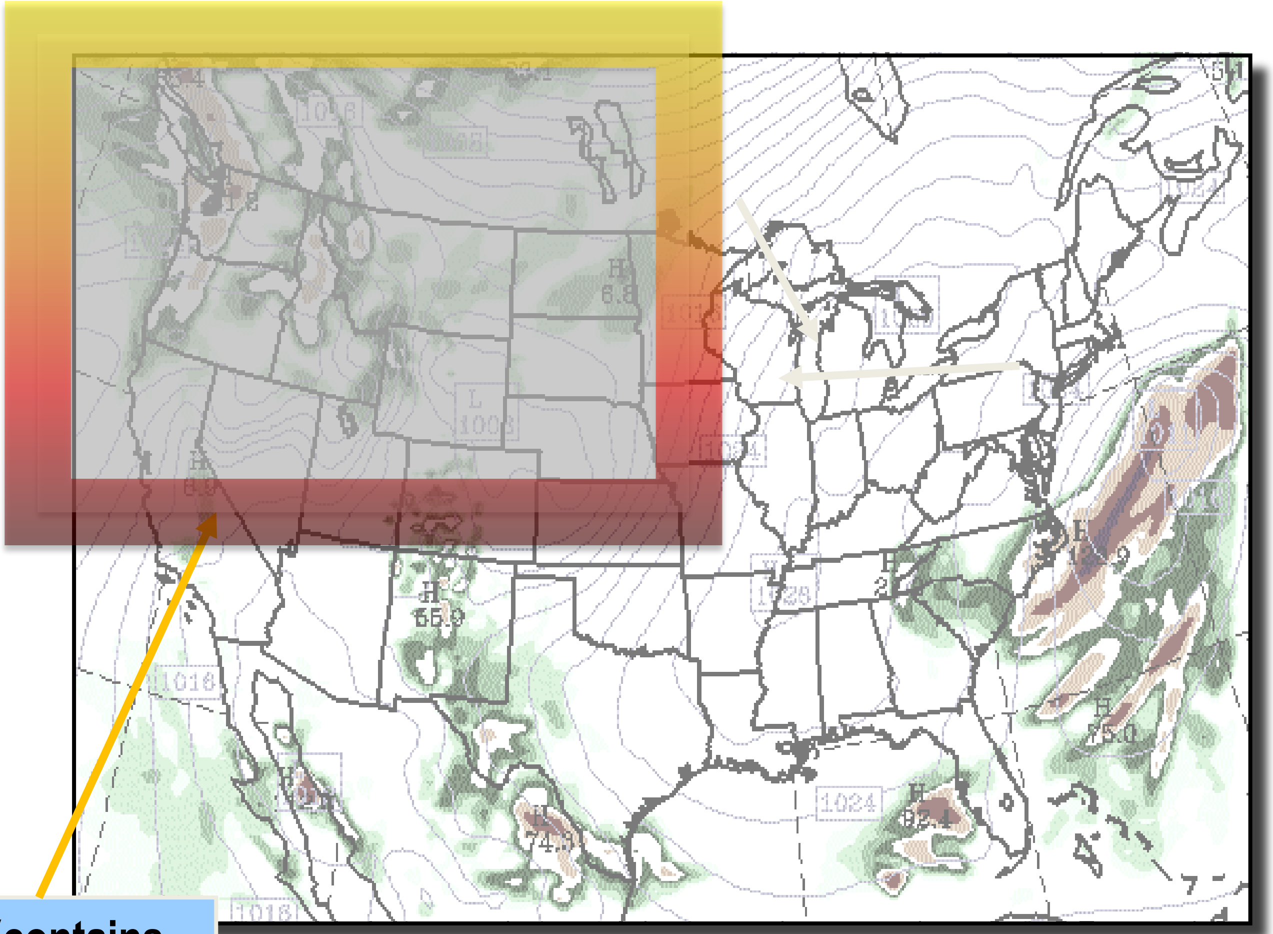
Following is an example code fragment that requires communication between patches

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.

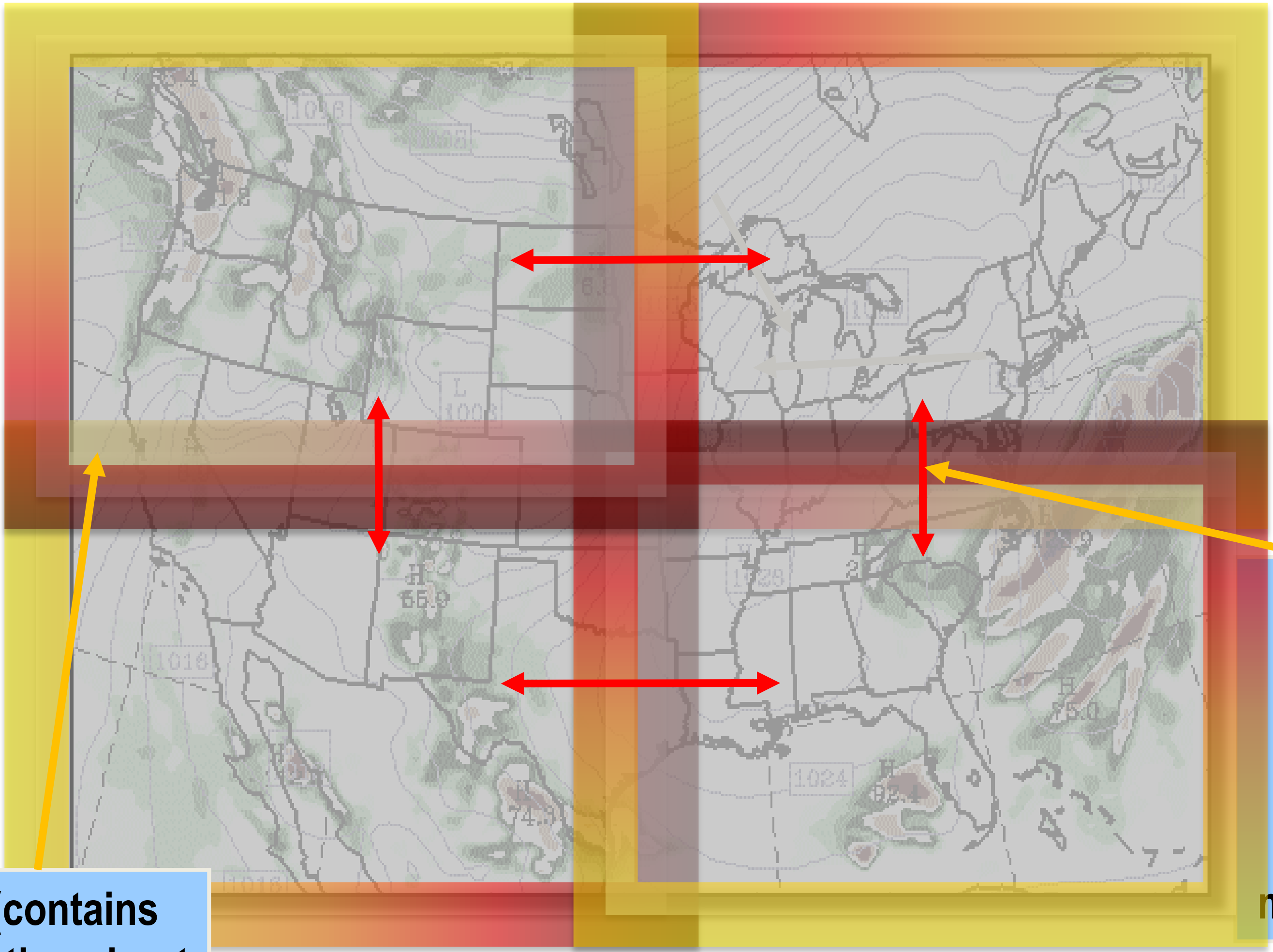
These are *horizontal data dependencies* because the indexed operands may lie in the patch of a neighboring processor. That neighbor's updates to that element of the array won't be seen on this processor.

Distributed Memory Communications



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

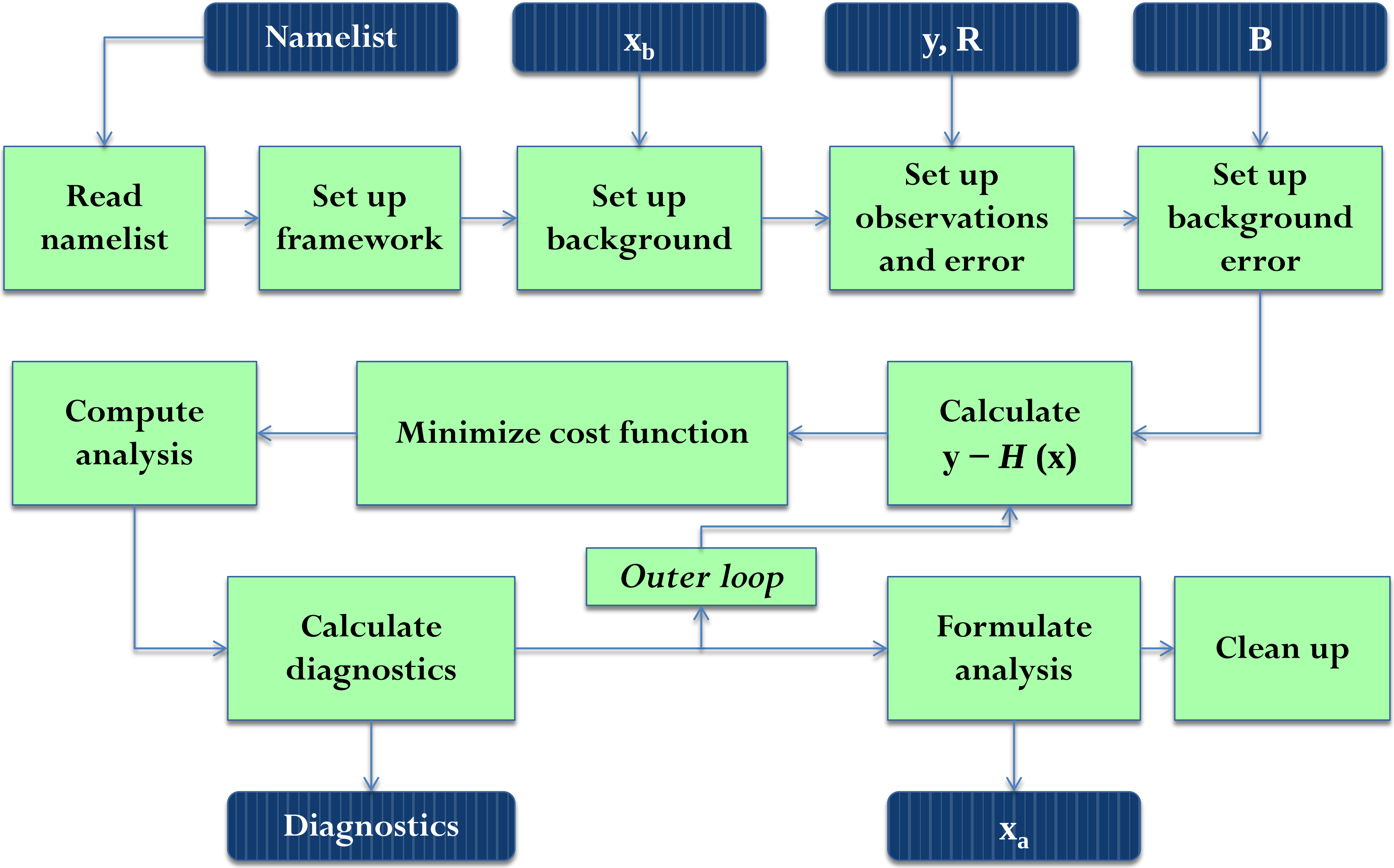
- Developed from MM5 3DVar beginning around 2002, first version (2.0) released December 2003
- 4DVAR capability added in 2008, made practical with parallelism starting with Version 3.4 (April 2012)
- Developed and supported by WRFDA group of the Mesoscale and Microscale Meteorology Lab of NCAR
- Requirements emphasize flexibility over a range of platforms, applications, users, performance
- Current release WRFDA v3.7 (April 2015)
- Shares the WRF Software Framework

WRFDA and J

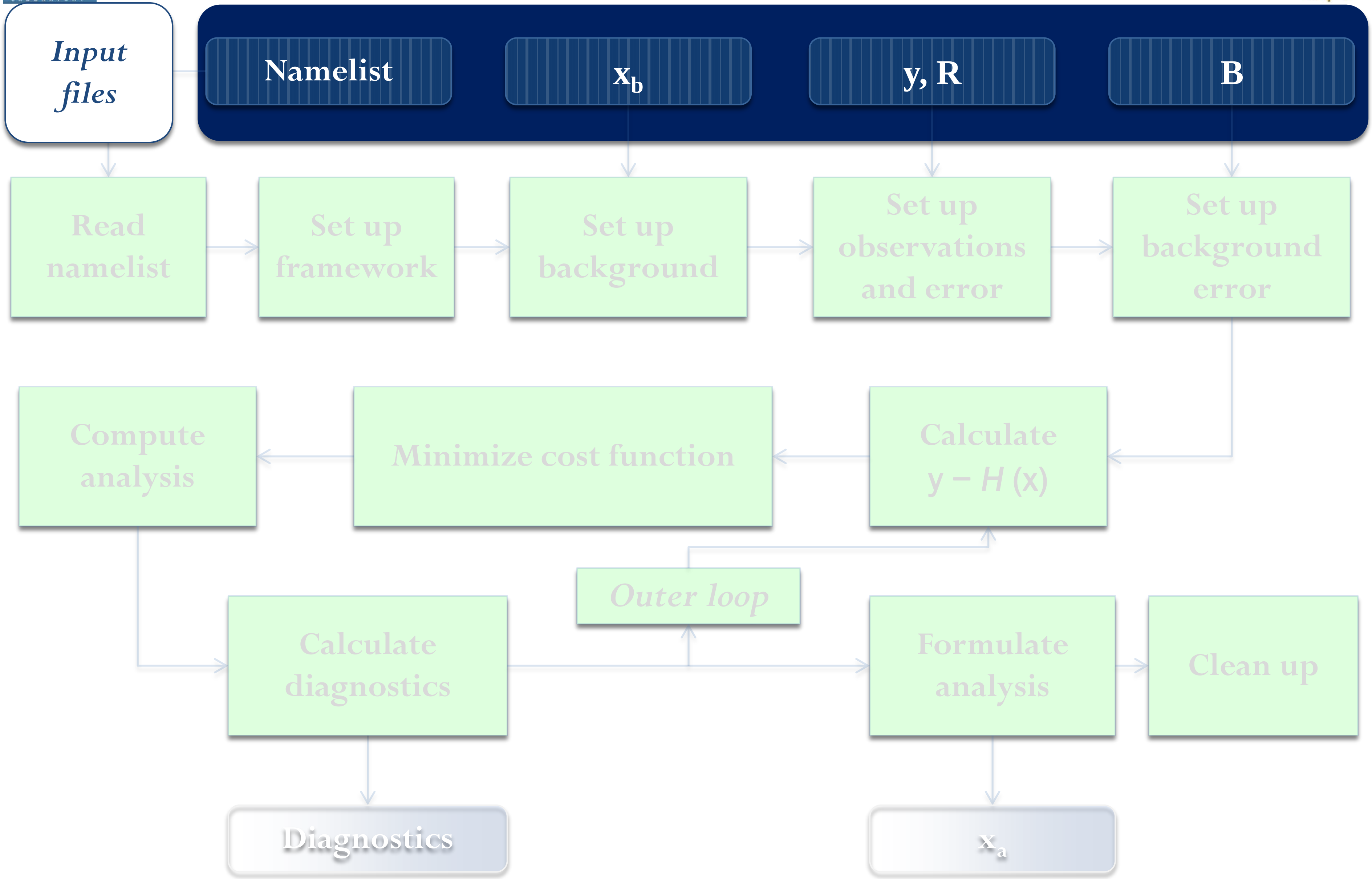
$$J(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x_b})^{\mathrm{T}}\mathbf{B}^{-1}(\mathbf{x} - \mathbf{x_b}) + \frac{1}{2}(\mathbf{y} - H(\mathbf{x}))^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{y} - H(\mathbf{x}))$$

- Model background ($\mathbf{x_b}$)
- Background error (**B**)
- Observations (y_0) and their associated error statistics (**R**)
- Minimize this cost function ($J(\mathbf{x})$) to find the analysis (**X**)
- Run forecast, repeat for cycling mode

WRFDA broken down by process



WRFDA broken down by process



Input files

- `namelist.input`

  The input file where the user specifies the different options for a WRFDA run. This allows user great flexibility to change the usage of WRFDA without having to recompile

- `fg`

  "First guess"; can be either a WRF input file created by WPS and real.exe, or a WRF output file from a forecast.

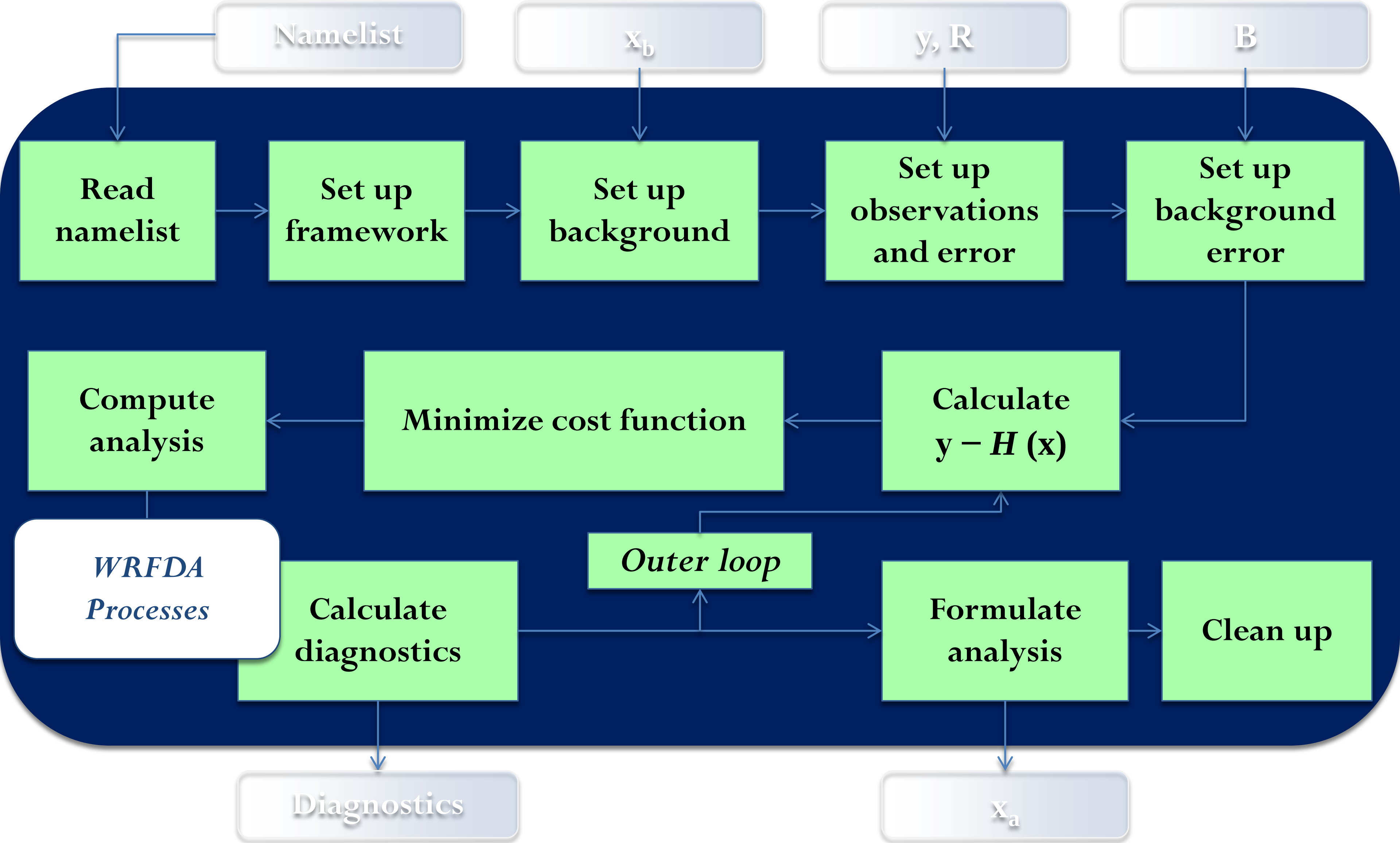
- `ob.ascii, amsua.bufr, ob01.rain, etc`

  WRFDA accepts a wide variety of observations in several different formats, which will be described in later talks

- `be.dat`

  This is a binary file containing background error information; it can be generated using the GEN_BE utility, which will be described in a later talk

WRFDA broken down by process



Read namelist

- Read user-specified options from `namelist.input`
- Set default values for options *not* specified in the namelist
- Perform consistency checks between namelist options

Calling order:

```
da_wrfvar_main ==> call da_wrfvar_init1, da_wrfvar_init2 ==> call initial_config
```

Calling subroutines:

```
da_wrfvar_main.f90 ==> da_wrfvar_init1.inc, da_wrfvar_init2.inc ==> module_configure.F
```

Set up framework

- Utilize WRF Software Framework distributed memory capability to allocate and configure the domain
- Allocate needed memory, initializes domain and tile dimensions, etc.
- Create output files

Calling order:

```
da_wrfvar_main ==> call da_wrfvar_init2 ==> call alloc_and_configure_domain
da_wrfvar_main ==> call da_wrfvar_run.inc ==> call da_wrfvar_interface ==> call
da_solve ==> call da_solve_init
```

Calling subroutines:

```
da_wrfvar_main.f90 ==> da_wrfvar_init2.inc ==> module_domain.F
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc
==> da_solve_init.inc
```

Set up background

- Read the first-guess file
- Extract fields used by WRFDA
- Create background FORTRAN 90 derived data type ***xb***, etc.

Calling order:

```
da_wrfvar_main ==> call da_wrfvar_init2 ==> call da_med_initialdata_input
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve
==>call da_setup_firstguess
```

Calling subroutines:

```
da_wrfvar_main.f90 ==> da_wrfvar_init2.inc ==> da_med_initialdata_input.inc
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc
==>da_setup_firstguess.inc
```

Set up observations and error

- Read in observations
- Assign observational error
- Create observation FORTRAN 90 derived data type ***ob***
- Domain and time check

Calling order:

```
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve
==> call da_setup_obs_structures
```

Calling subroutines:

```
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc
==>da_setup_obs_structures.inc
```

Set up background error

- Reads in background error statistics from be.dat
- Extracts necessary quantities: eigenvectors, eigenvalues, lengthscales, regression coefficients, etc.
- Creates background error FORTRAN 90 derived data type ***be***
- Specifics of background error in WRFDA be covered in more detail in a later talk

Calling order:

```
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve
==>call da_setup_background_errors
```

Calling subroutines:

```
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc
==>da_setup_background_errors.inc
```

Minimize cost function

- Use conjugate gradient method
  - Initializes analysis increments to zero
  - Computes cost function
  - Computes gradient of cost function
  - Uses gradient of the cost function to calculate new value of analysis control variable
- Increment this process until specified minimization is achieved

Calling order:

```
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve
==>call da_minimise_cg
```

Calling subroutines:

```
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc
==>da_minimise_cg.inc
```

Further reading: Shewchuk, Jonathan Richard, 1994. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain (http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf)

Compute analysis

- Convert control variables to model space analysis increments
- Calculate analysis = first-guess + analysis increment
- Perform consistency checks (e.g., remove negative humidity)

Calling order:

```
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve
==>call da_transfer_xatoanalysis
```

Calling subroutines:

```
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc
==>da_transfer_xatoanalysis.inc
```

Calculate diagnostics

- Output $\mathbf{y} - H(\mathbf{x}_b)$, $\mathbf{y} - H(\mathbf{x}_a)$ statistics for all observation types and variables
- Compute $\mathbf{x}_a - \mathbf{x}_b$ (analysis increment) statistics for all model variables and levels
- Statistics include minimum, maximum (and their locations), mean and standard deviation.

Calling order:

```
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve
==>call da_transfer_xatoanalysis
```

Calling subroutines:

```
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc
==>da_transfer_xatoanalysis.inc
```

Outer loop

- An outer loop is a method of iterative assimilation to maximize contributions from observations non-linearly related to the control variables (e.g., GPS refractivity, Doppler radial velocity)
    - After the previous steps, the analysis $\mathbf{x}_a$ is used as the new first guess
    - The cost function minimization and diagnostic steps are repeated
    - This can be repeated up to 100 times, though only a few should be necessary

Calling order:

```
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve
```

Calling subroutines:

```
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc
```

Further reading: Rizvi et al., 2008 (http://www.mmm.ucar.edu/wrf/users/workshops/WS2008/abstracts/P5-03.pdf)

Write analysis

- Write analysis file in native WRF format (netCDF).

Calling order:

```
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve
==>call da_transfer_xatoanalysis
```

Calling subroutines:

```
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc
==>da_transfer_xatoanalysis.inc
```

Clean up

- Deallocate dynamically-allocated arrays, structures, etc.
- Timing information
- Clean end to WRFDA

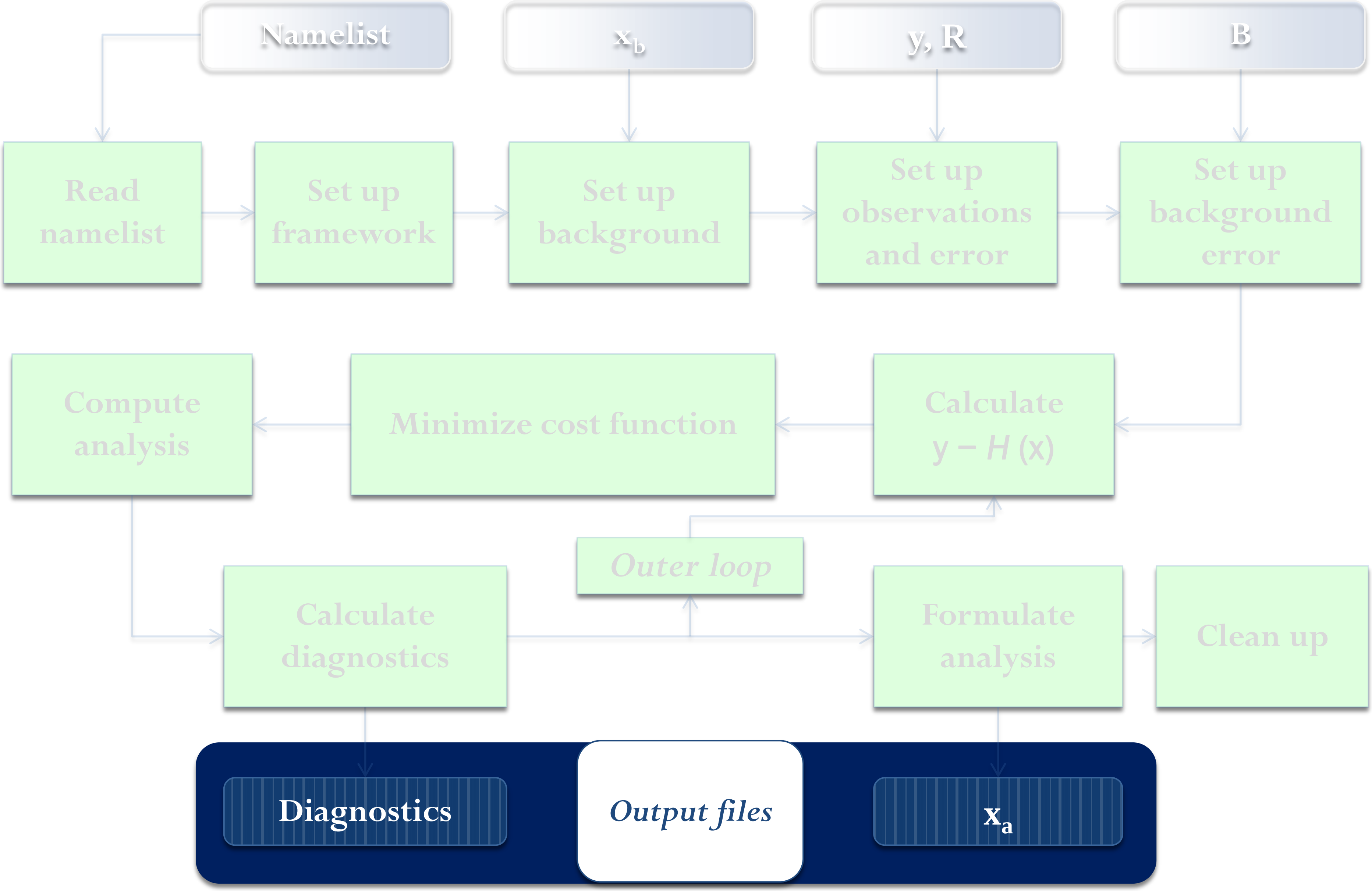
Calling order:

```
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve
da_wrfvar_main ==> call da_wrfvar_finalize
```

Calling subroutines:

```
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc
da_wrfvar_main.f90 ==> da_wrfvar_finalize.inc
```

WRFDA broken down by process



Namelist

x_b

y, R

B

Read namelist

Set up framework

Set up background

Set up observations and error

Set up background error

Compute analysis

Minimize cost function

Calculate $y - H(x)$

Outer loop

Calculate diagnostics

Formulate analysis

Clean up

Diagnostics

Output files

x_a

Output files: Diagnostics

- File names: `grad_fn`, `jo`, `qcstat_conv*`, `statistics`, etc.
- There will be a number of diagnostics files output by WRFDA
  - Many will end in .0000, .0001, etc.; these are diagnostics specific to each processor used
  - Many will also contain a _01; these files will appear for each outer loop as _02, _03, etc.
- More or fewer output files can be specified by certain namelist options

Output files: x_a (analysis)

- File name: `wrfvar_output`
- This is the model output in WRF native format (netCDF). This file can be used directly for research purposes, or used to initialize a WRF forecast