WRF Data Assimilation System

Michael Kavulich

Special thanks to:

Xin Zhang, Xiang-Yu Huang

WRFDA Tutorial, July **2013**, NCAR

Many slides are borrowed from WRF software lectures

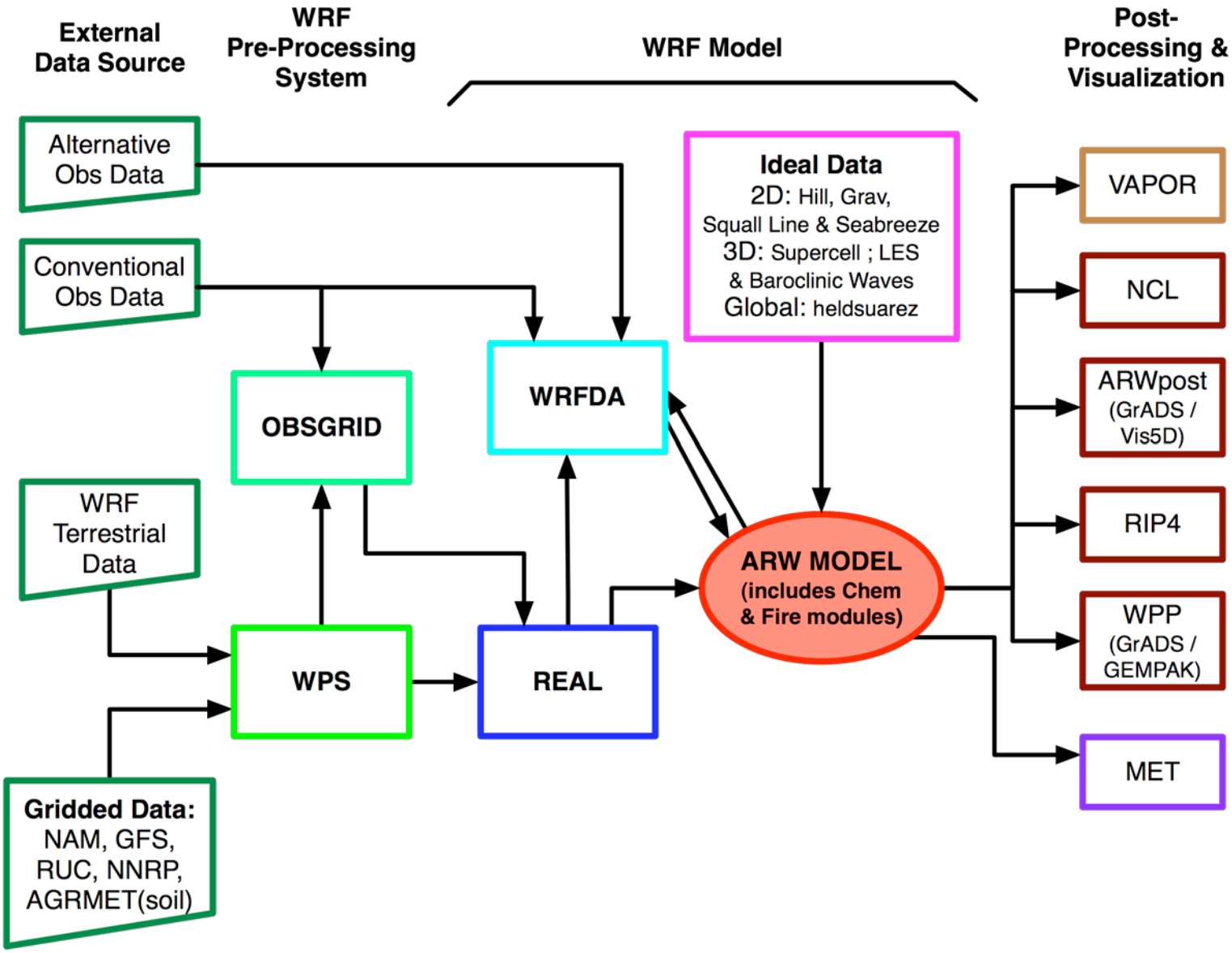
WRFDA System – Outline

- *Introduction*
- WRFDA Software Overview
- Computing Overview

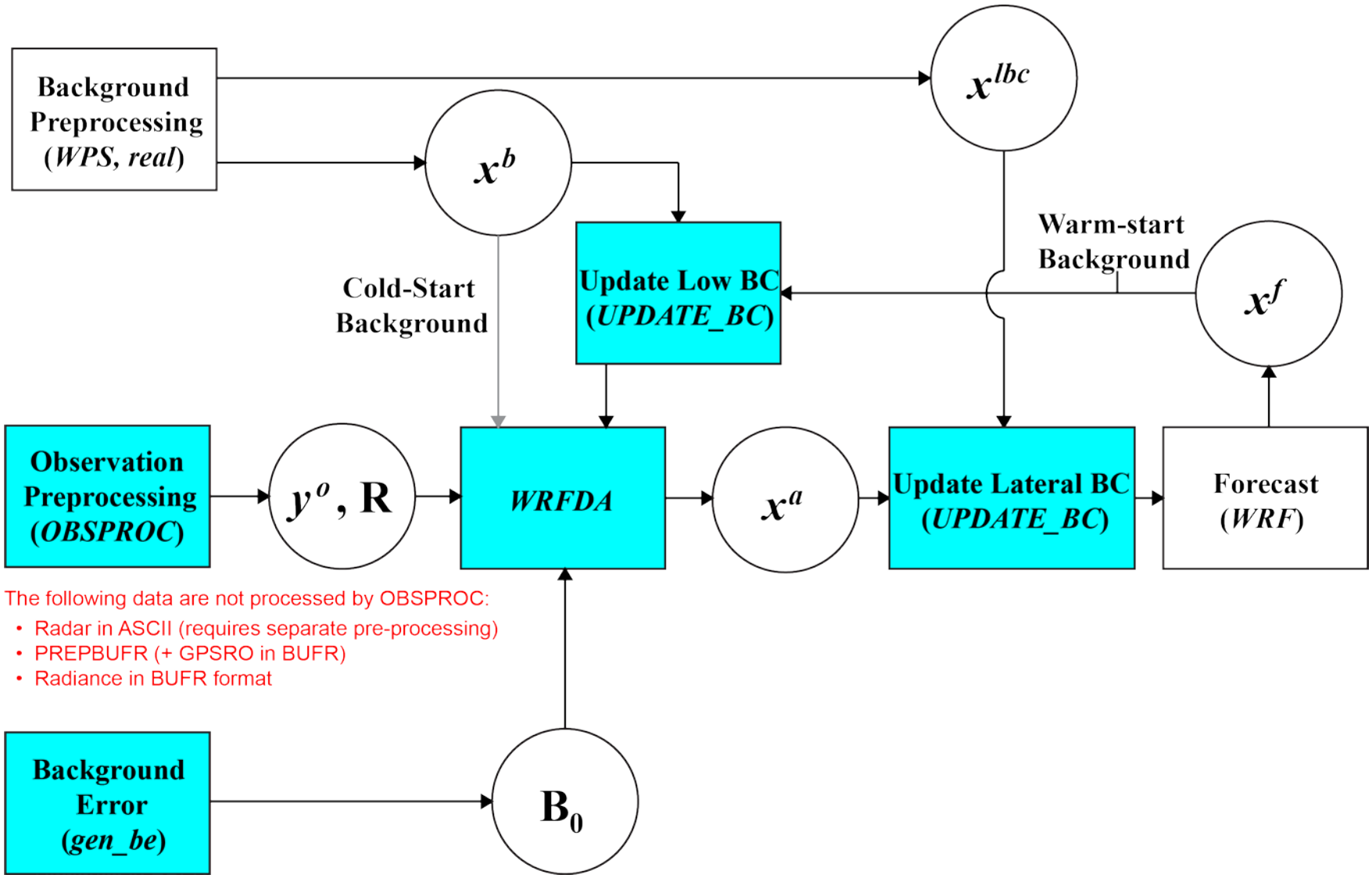
Introduction – What is WRFDA?

- A data assimilation system for the WRF Model (ARW core)
  - 3D- and 4D-VAR, 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

WRF Modeling System Flow Chart



WRFDA in the WRF Modeling System



Blue: Supported by WRFDA team

WRFDA System – Outline

- Introduction
- *WRFDA Software*
- Computing Overview

```
program da_wrfvar_main
   !-----------------------------------------------------------------------
   ! Purpose: Main program of WRF-Var.  Responsible for starting up, reading
   ! in (and broadcasting for distributed memory) configuration data, defining
   ! and initializing the top-level domain, either from initial or restart
   ! data, setting up time-keeping, and then calling the da_solve
   ! routine assimilation. After the assimilation is completed,
   ! the model is properly shut down.
   !-----------------------------------------------------------------------
   use module_symbols_util, only : wrfu_finalize
   use da_control, only : trace_use, var4d
   use da_tracing, only : da_trace_init, da_trace_report, da_trace_entry, &
      da_trace_exit
   use da_wrf_interfaces, only : wrf_shutdown, wrf_message, disable_quilting
   use da_wrfvar_top, only : da_wrfvar_init1,da_wrfvar_init2,da_wrfvar_run, &
      da_wrfvar_finalize
#ifdef VAR4D
   use da_4dvar, only : clean_4dvar, da_finalize_model
#endif
   implicit none
   ! Split initialisation into 2 parts so we can start and stop trace here
   call disable_quilting
   call da_wrfvar_init1
   if (trace_use) call da_trace_init
   if (trace_use) call da_trace_entry("da_wrfvar_main")
   call da_wrfvar_init2
   call da_wrfvar_run
   call da_wrfvar_finalize
#ifdef VAR4D
   if (var4d) then
      call clean_4dvar
      call da_finalize_model
   end if
#endif
   call wrf_message("*** WRF-Var completed successfully ***")
   if (trace_use) call da_trace_exit("da_wrfvar_main")
   if (trace_use) call da_trace_report
   call wrfu_finalize
   call wrf_shutdown
end program da_wrfvar_main
```

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

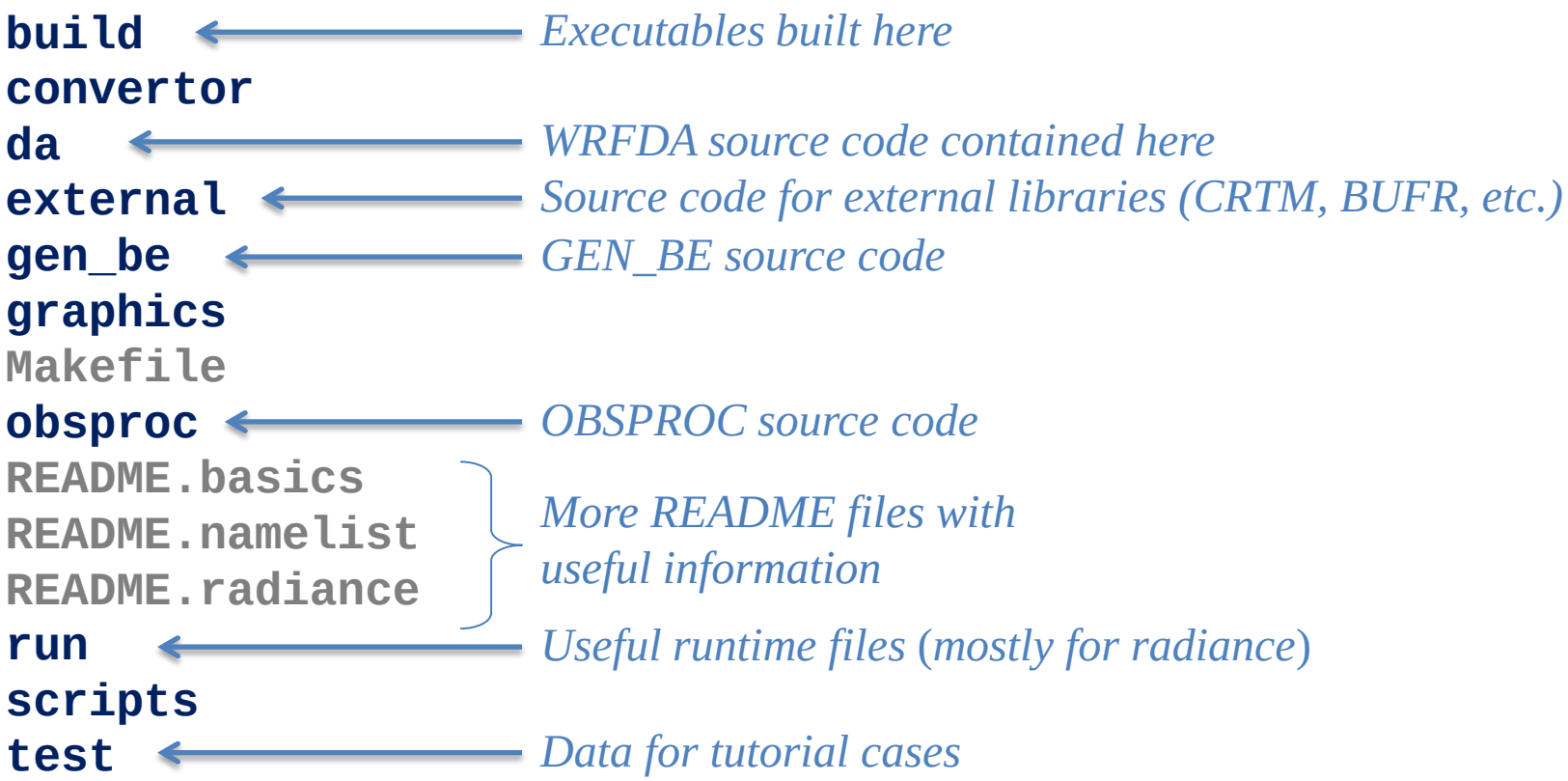
README file with information about WRFDA

Contains `registry.var`

WRFDA source code directory

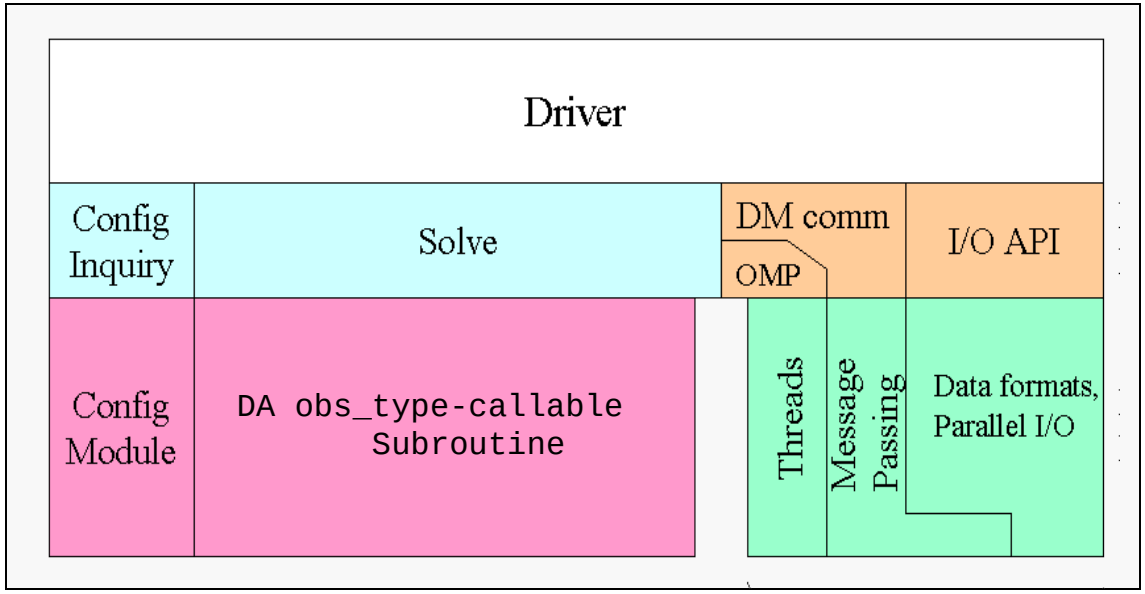
Legend:
Blue – directory
Green – script file
Gray – other text file

WRFDA/var Directory structure



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

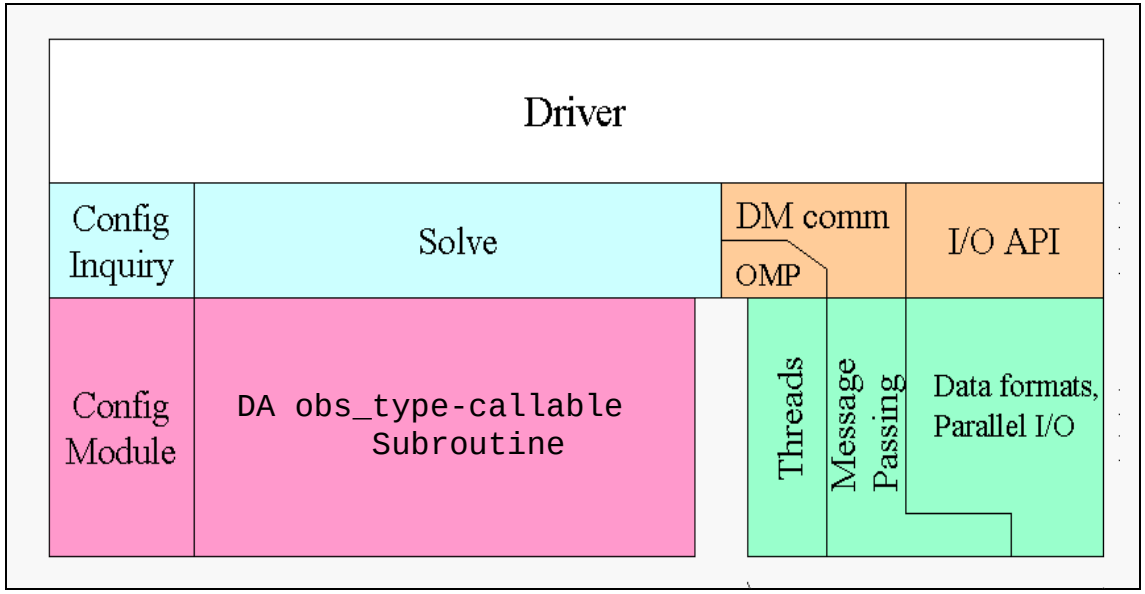
WRFDA Software – Architecture



registry.var

- 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.

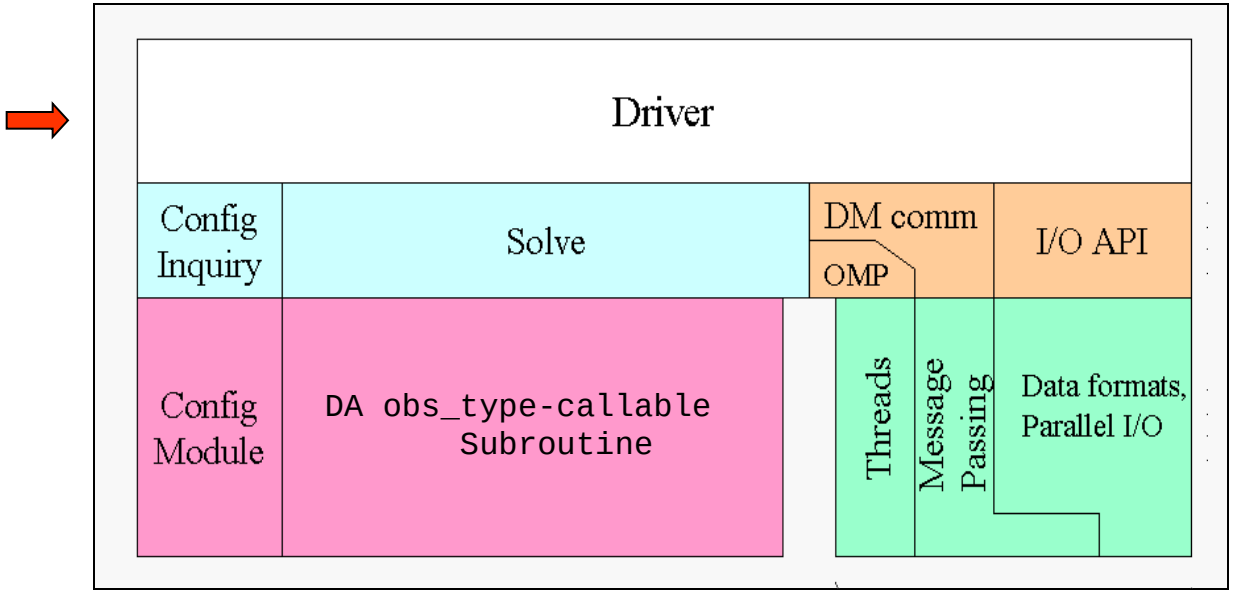
WRFDA Software – Architecture



registry.var

- 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 dictionary for WRFDA

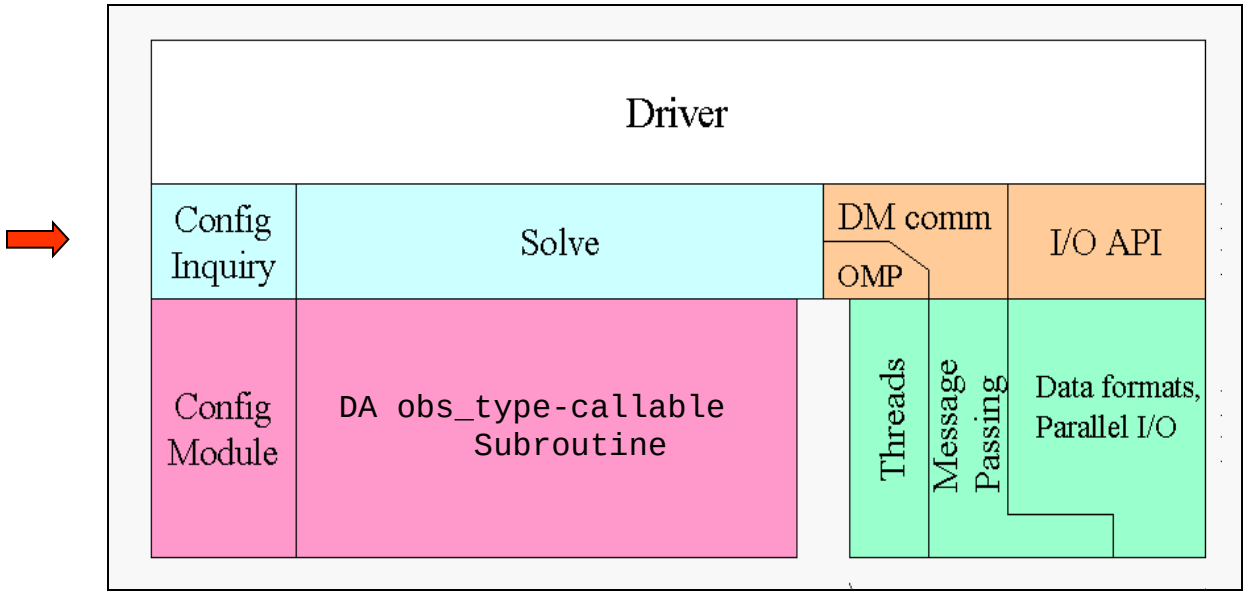
WRFDA Software – Architecture



Registry

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

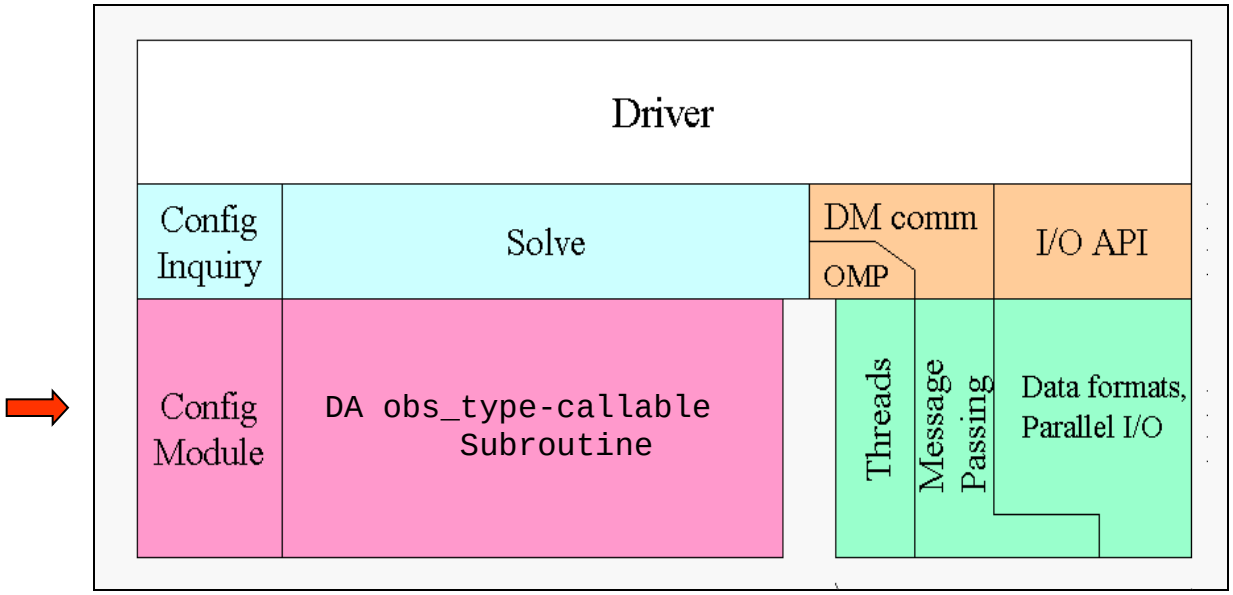
WRFDA Software – Architecture



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

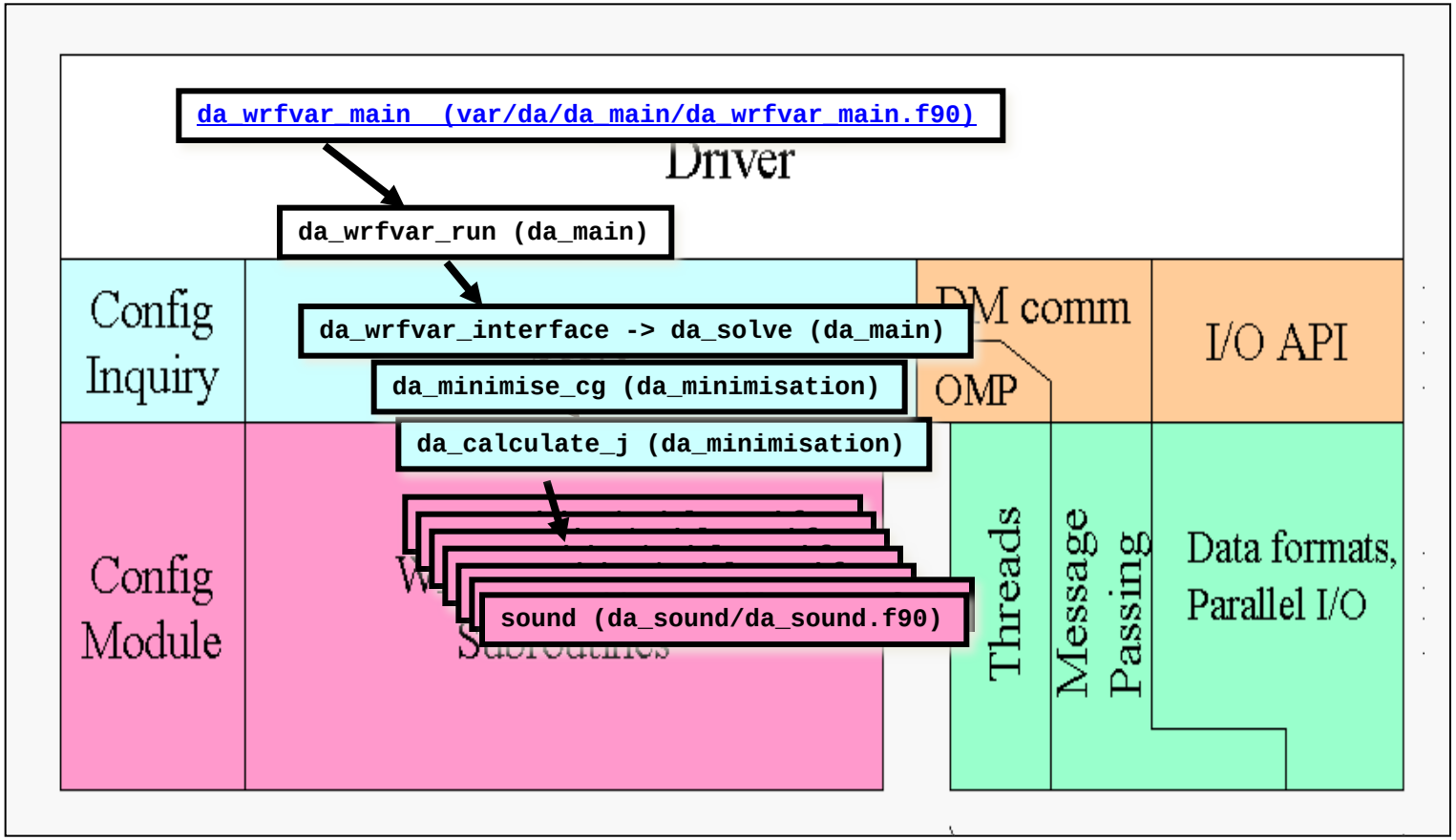


Registry

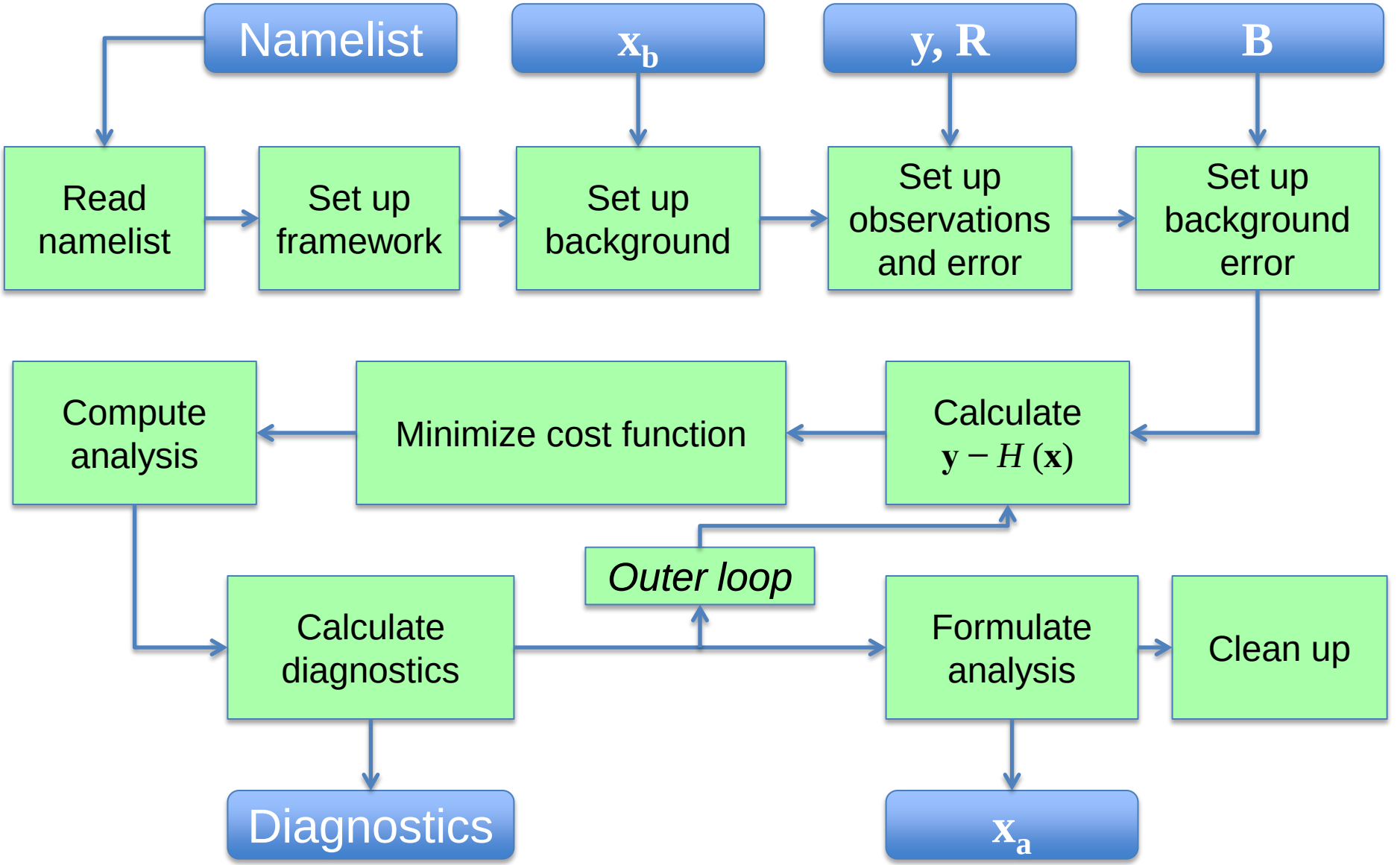
- Observation Layer
  - **Observation interfaces**: contains the gradient and cost function calculation subroutines for each type of observations.

Call Structure Superimposed on Architecture

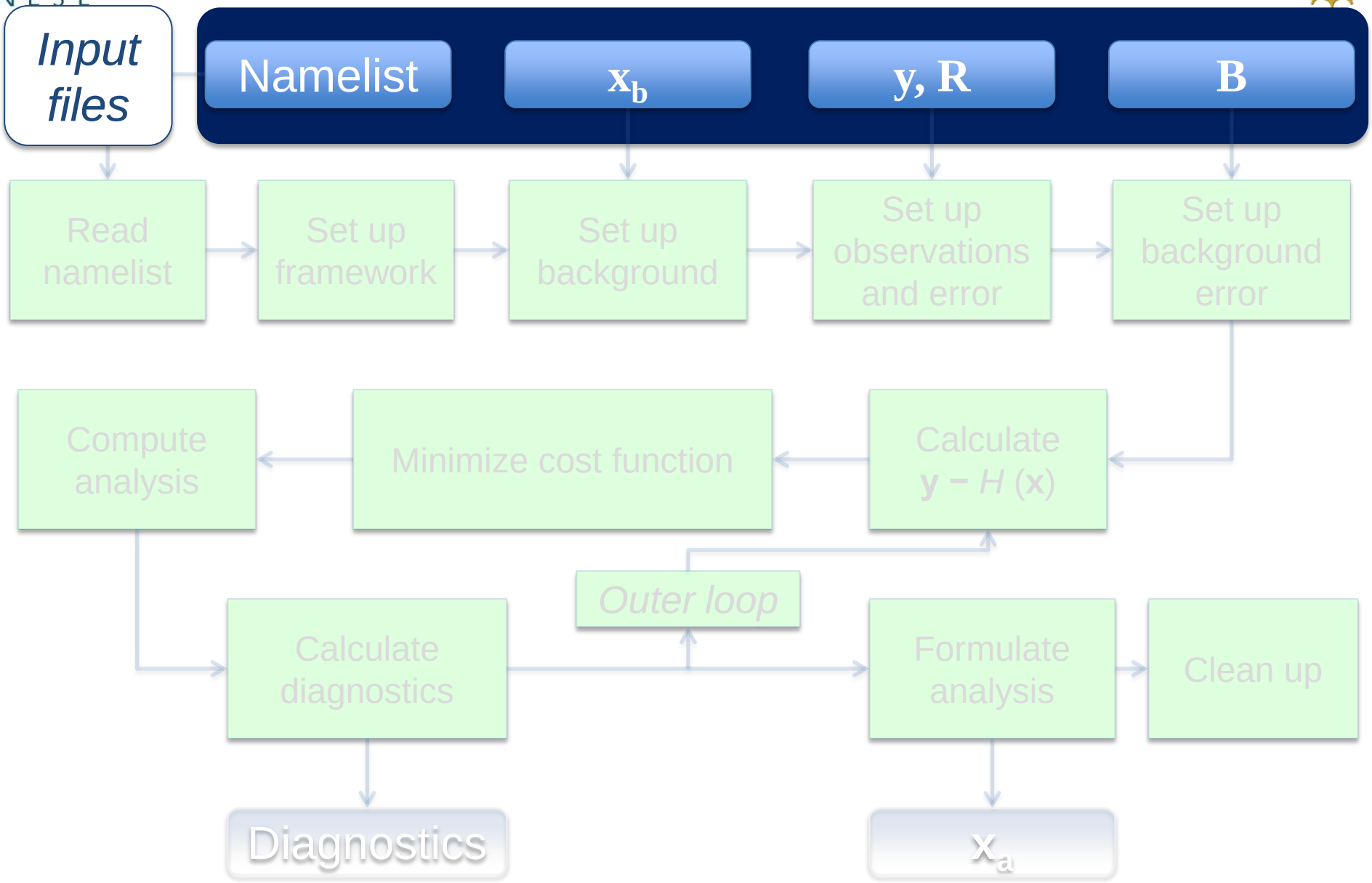
da_sound.f90(da_sound)



WRFDA broken down by process



WRFDA broken down by process



Input files: Namelist

- File name: `namelist.input`
- Specifies Input/Output options, domain details, types of observations to assimilate and how to assimilate them
- Allows user great flexibility to change the usage of WRFDA without having to recompile
- A large number (>1000) of namelist options govern the running of WRFDA; however, users will typically only be concerned with setting a few dozen of these
- More details can be found in the User's Guide

Input files: $\mathbf{x_b}$ (background)

- File name: `fg`
- Can be either a WRF input file created by WPS and real.exe, or a WRF output file from a forecast.

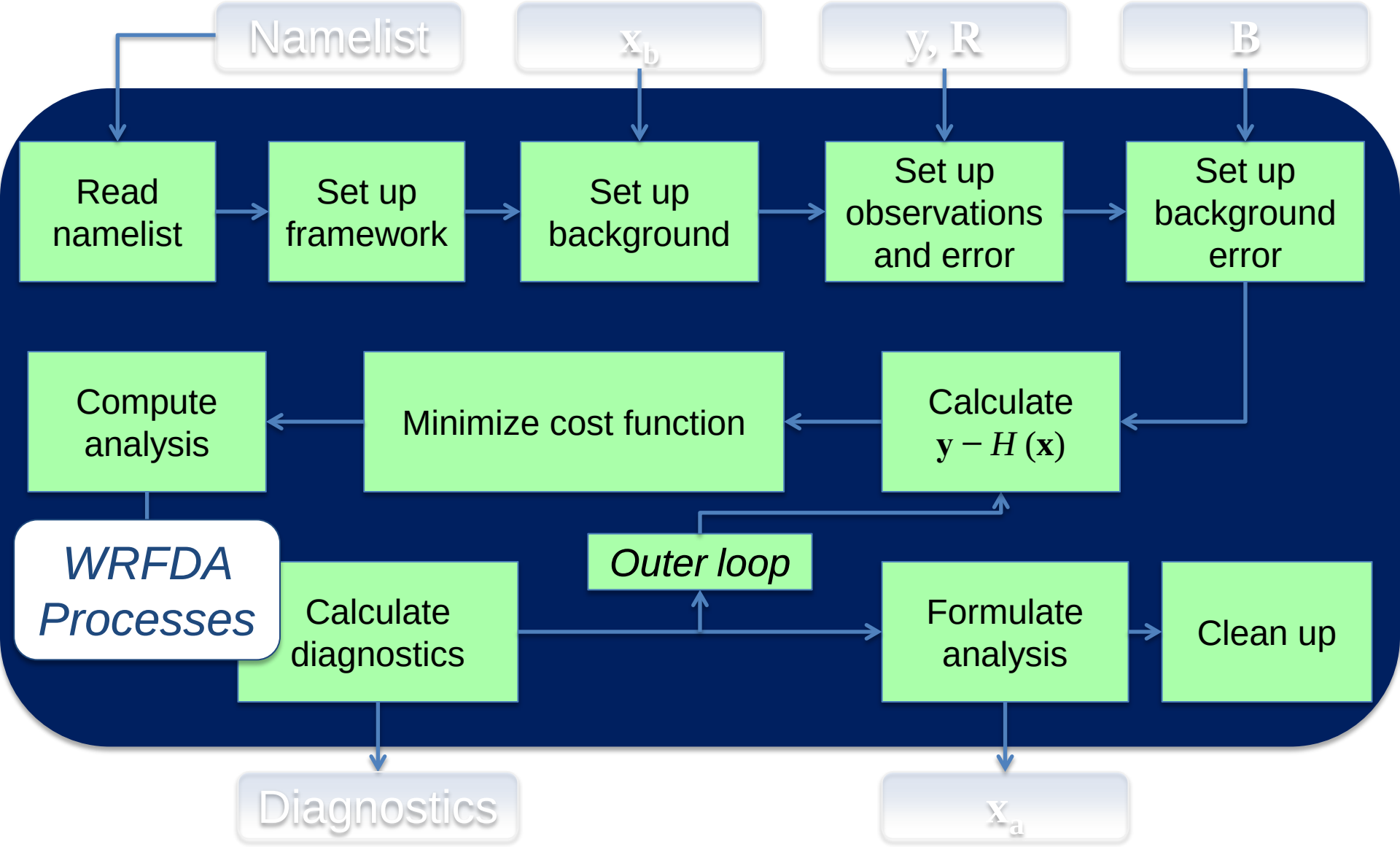
Input files: $\mathbf{y}$ (observations) and $\mathbf{R}$ (observation errors)

- File name: `ob.ascii`, `amsua.bufr`, `ob01.rain`, etc.
- WRFDA accepts a wide variety of observations in several different formats
  - OBSPROC ASCII format (surface, sounding, GPS, etc.)
  - PREPBUFR format (surface, sounding, etc.)
  - BUFR format (radiance)
  - Other ASCII format (radar, precipitation)
- Observation errors are either provided in the observation file, or standard errors (file name: obserr.txt) are used.

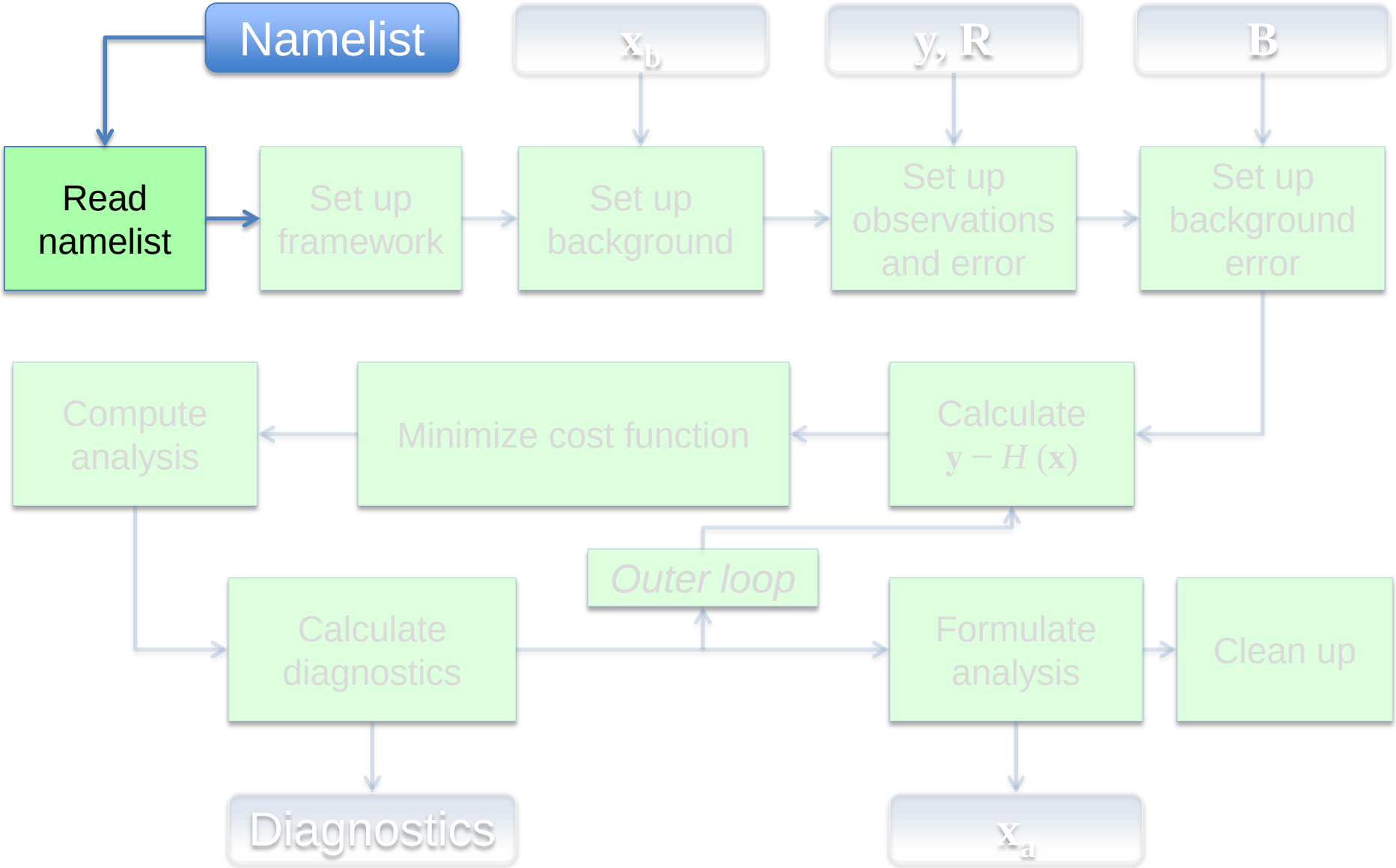
Input files: $\mathbf{B}$ (background error)

- File name: `be.dat`
- This is a binary file containing background error information
  - cv_options=3 NCEP background error formulation
    - File provided with WRFDA code
    - Not recommended: should be used with caution
  - cv_options=5 NCAR background error formulation
    - File created using gen_be utility
    - Recommended option
  - cv_options=6; Multivariate Background Error (MBE) statistics
    - Not officially supported

WRFDA broken down by process



Read namelist



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

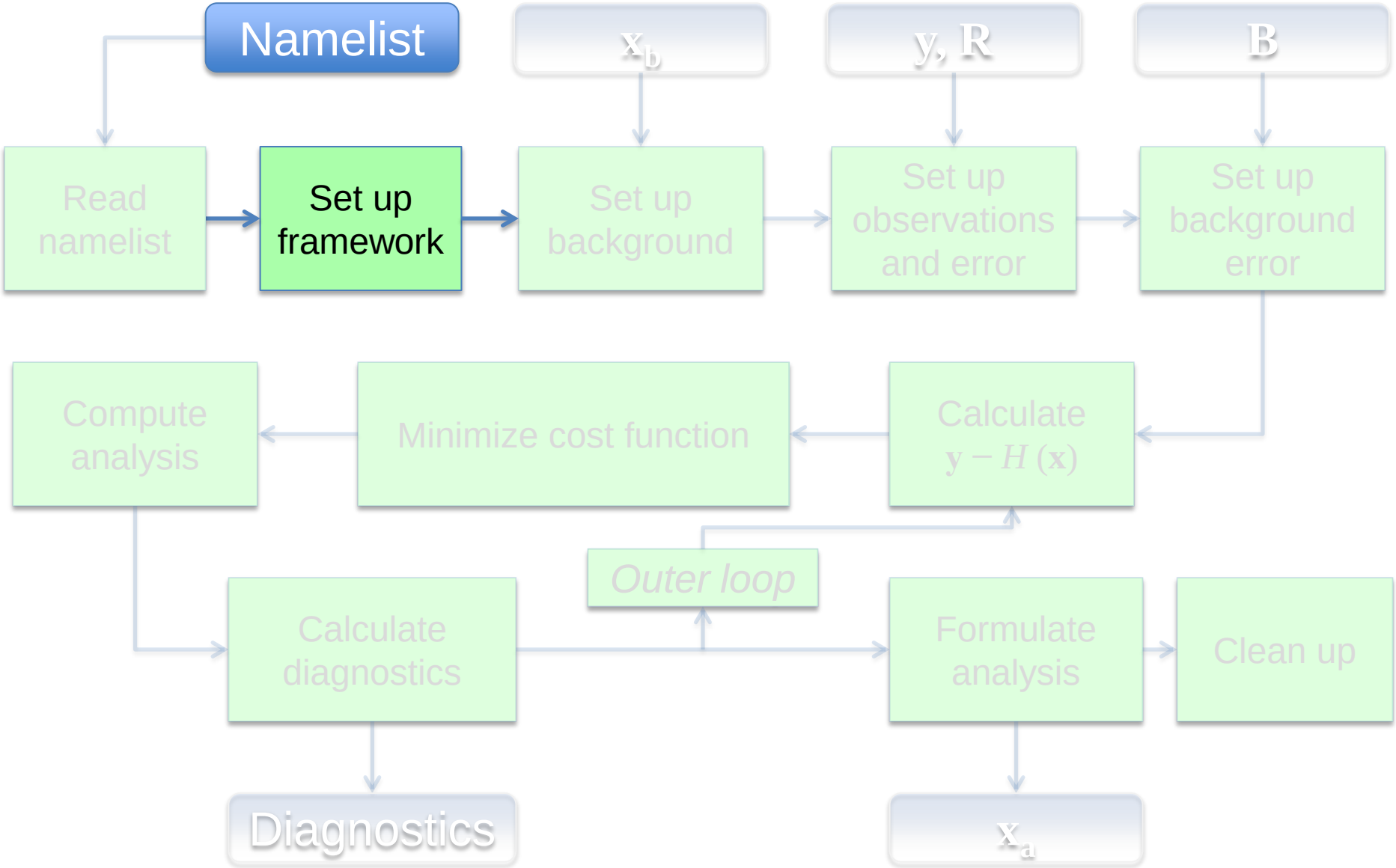
Calling order:

```
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



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

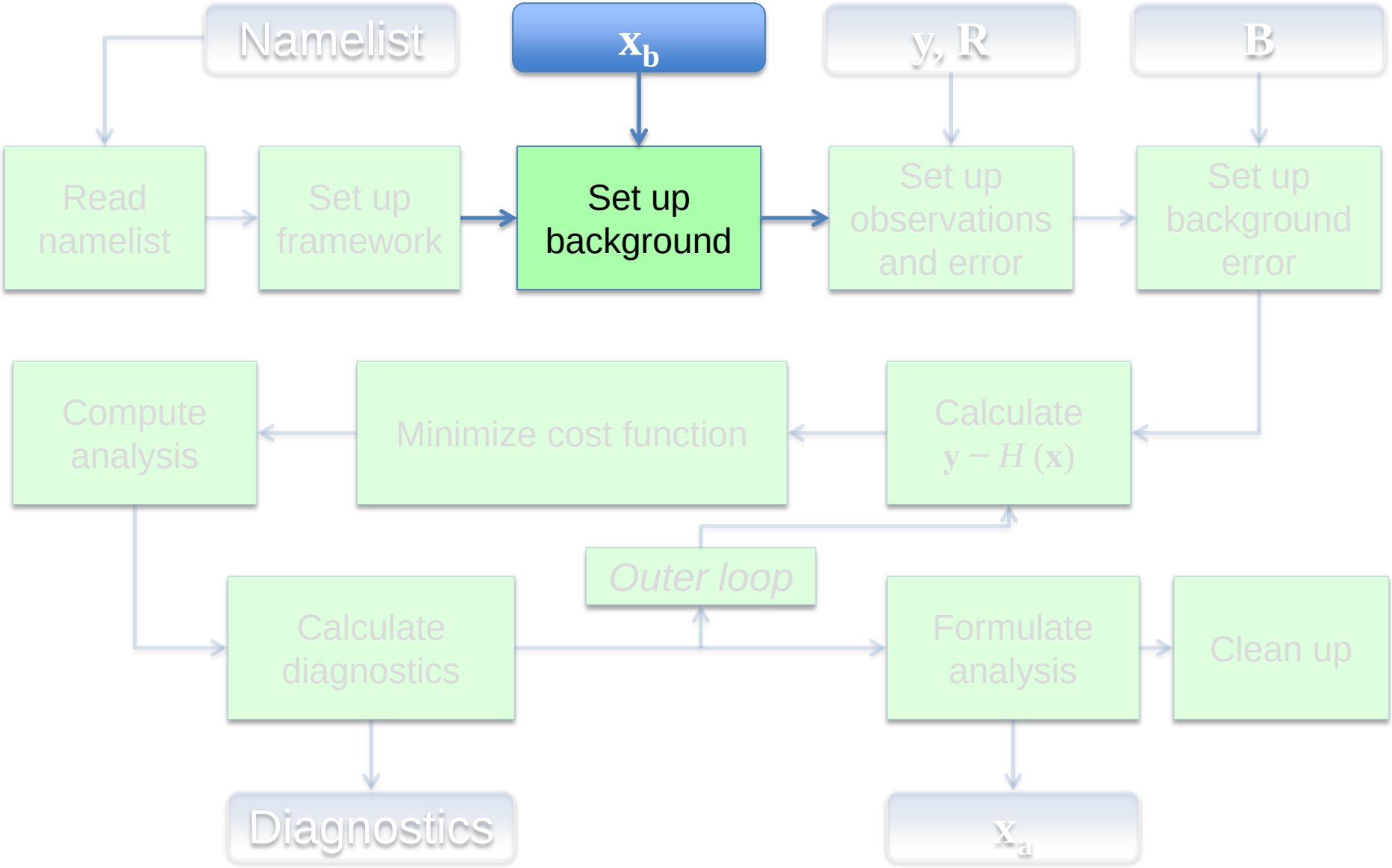
Calling order:

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



Set up background

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

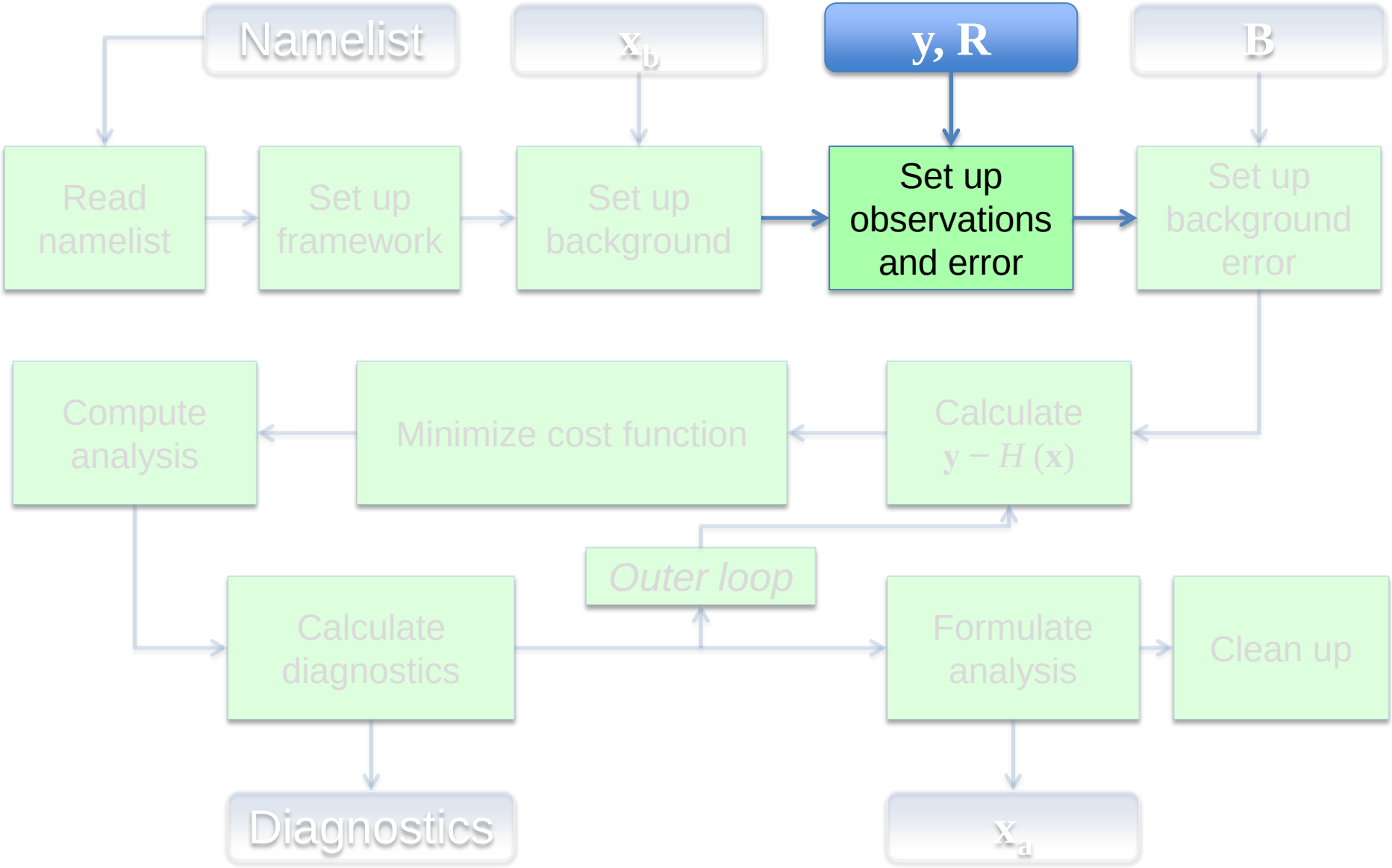
Calling order:

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



Set up observations and error

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

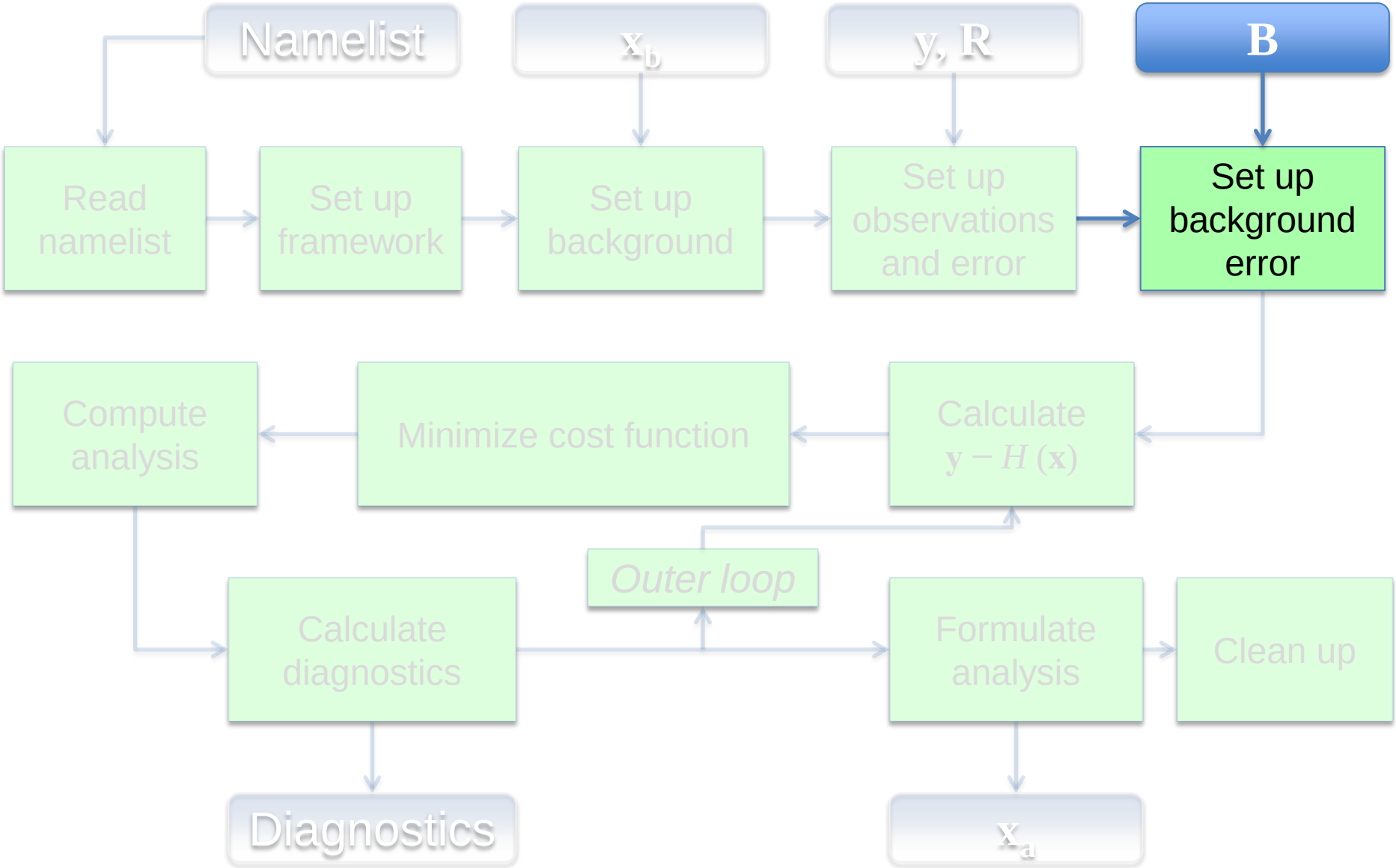
Calling order:

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



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***
- Reference :Online BE Documents

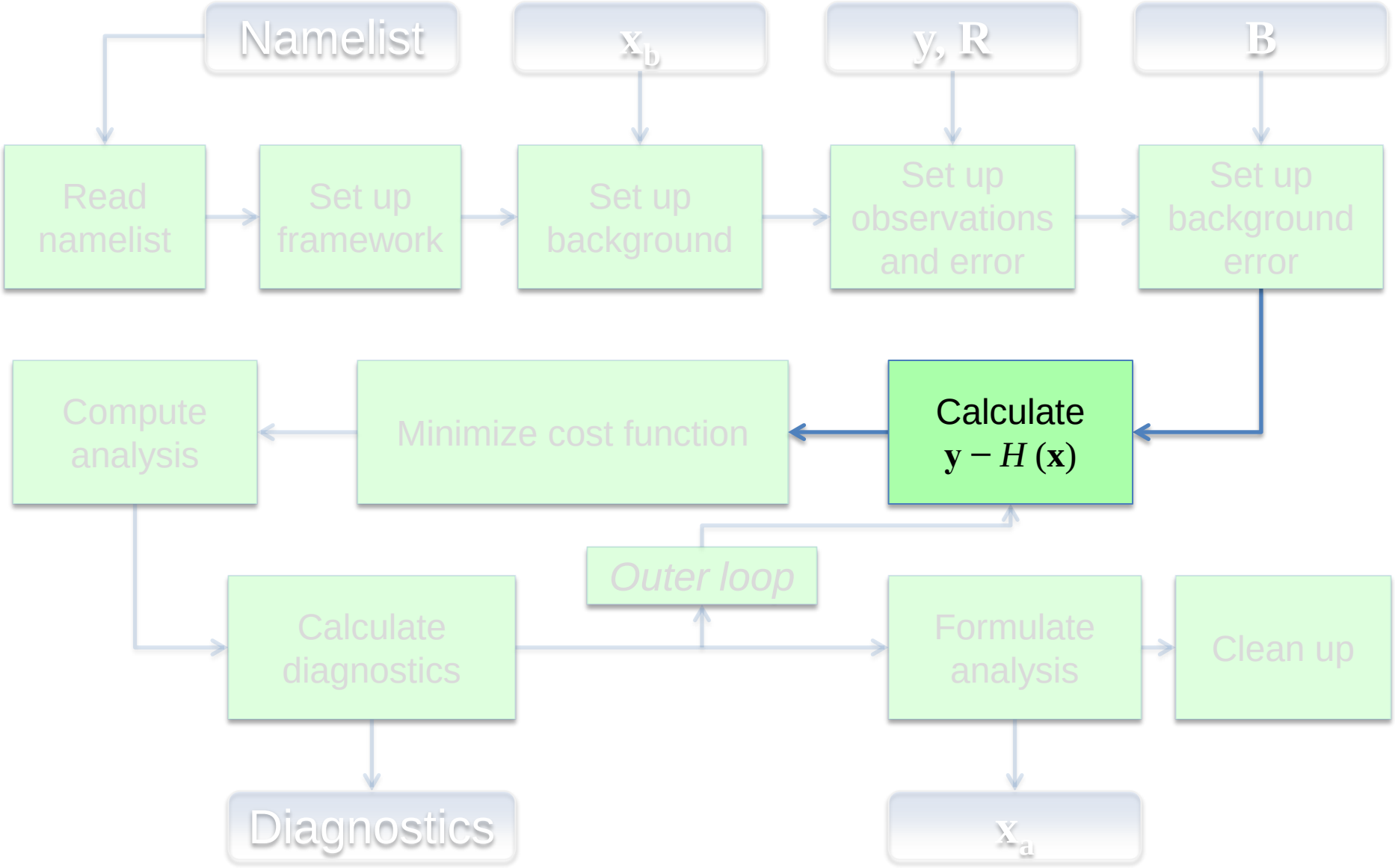
Calling order:

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**

Calculate y − H (x)



Calculate $\mathbf{y} - H(\mathbf{x})$ (Innovation)

- Calculate model equivalent of observations through interpolation and variable transformations
- Compute observation minus first guess ($\mathbf{y} - H(\mathbf{x})$) value
- Create innovation vector FORTRAN 90 derived data type ***iv***

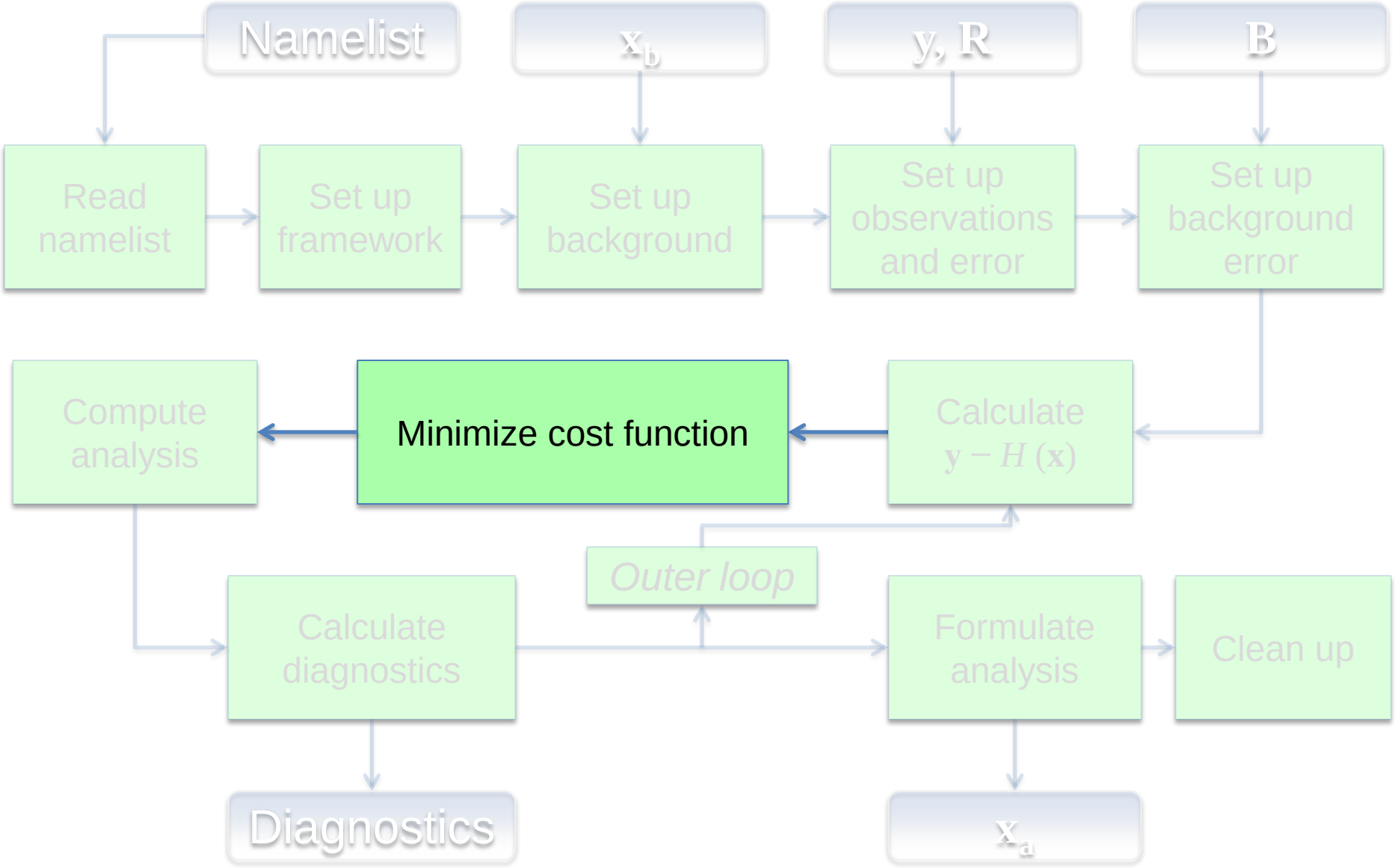
Calling order:

da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==>
call da_solve ==>call da_get_innov_vector, da_allocate_y

Calling subroutines:

da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==>
da_solve.inc ==>da_get_innov_vector.inc, da_allocate_y.inc

Minimize cost function



Minimize cost function

- Use conjugate gradient method
  - Initializes analysis increments to zero
  - Computes cost function (if desired)
  - 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:

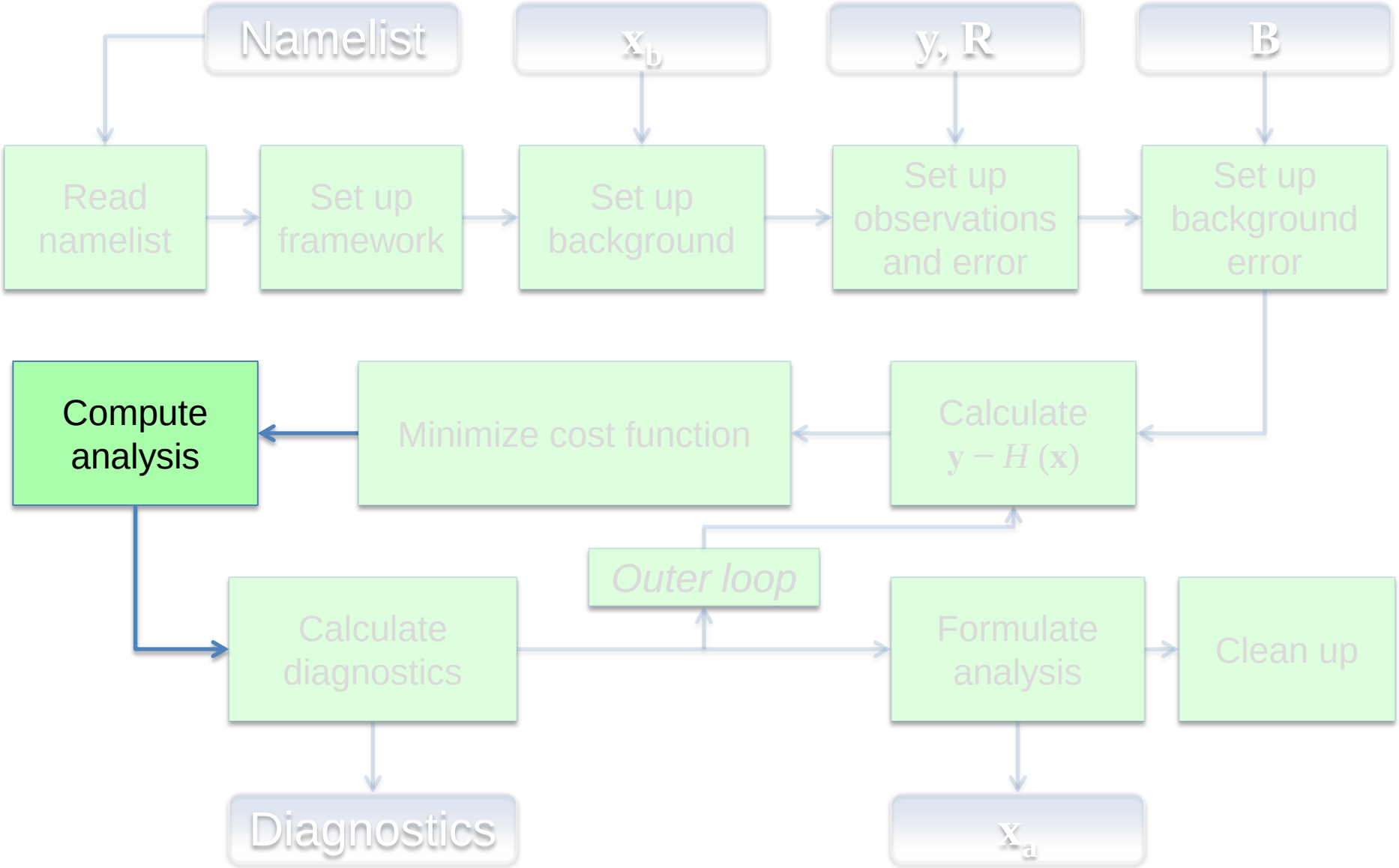
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve ==>call da_minimise_cg

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



Compute analysis

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

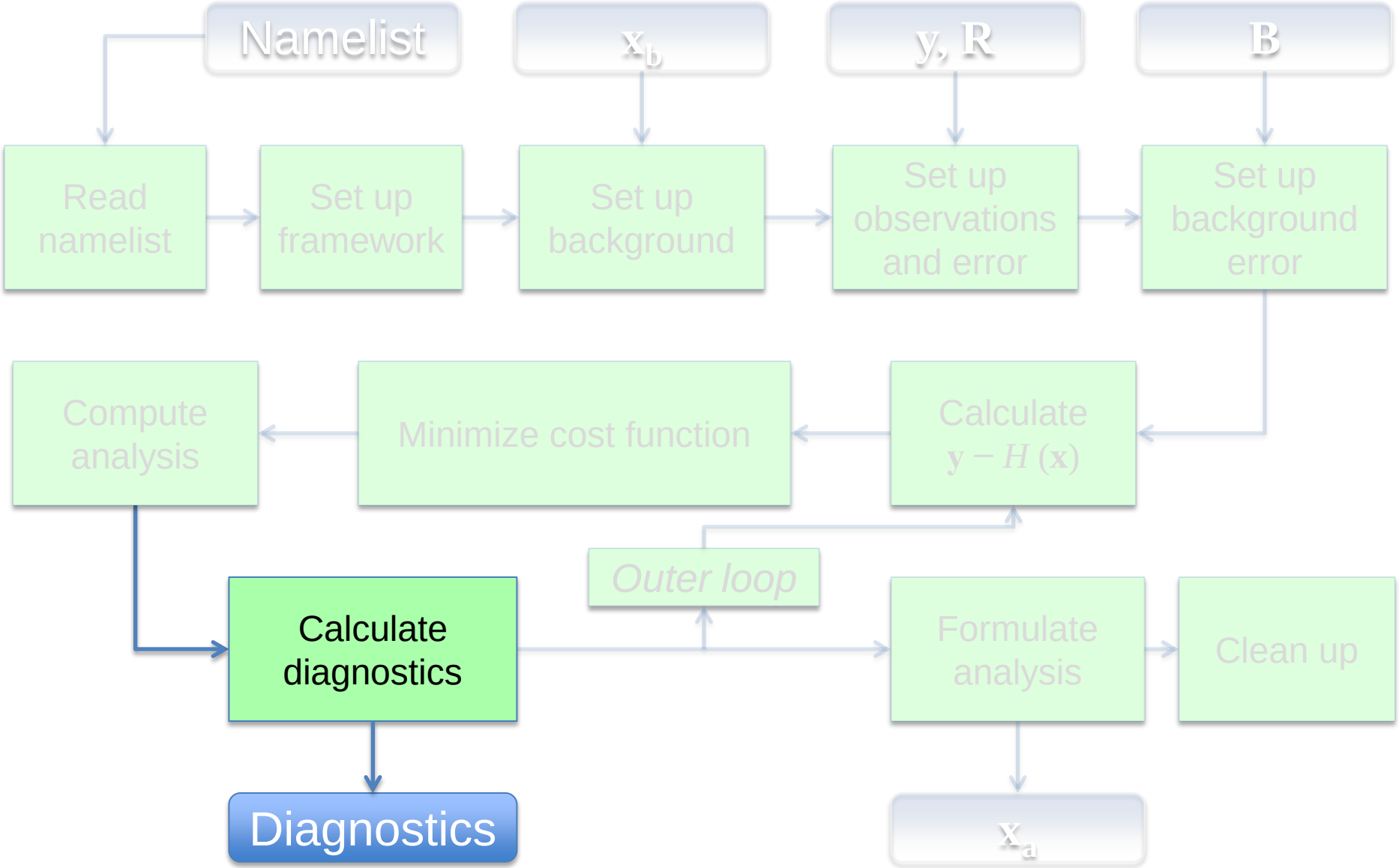
Calling order:

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



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.

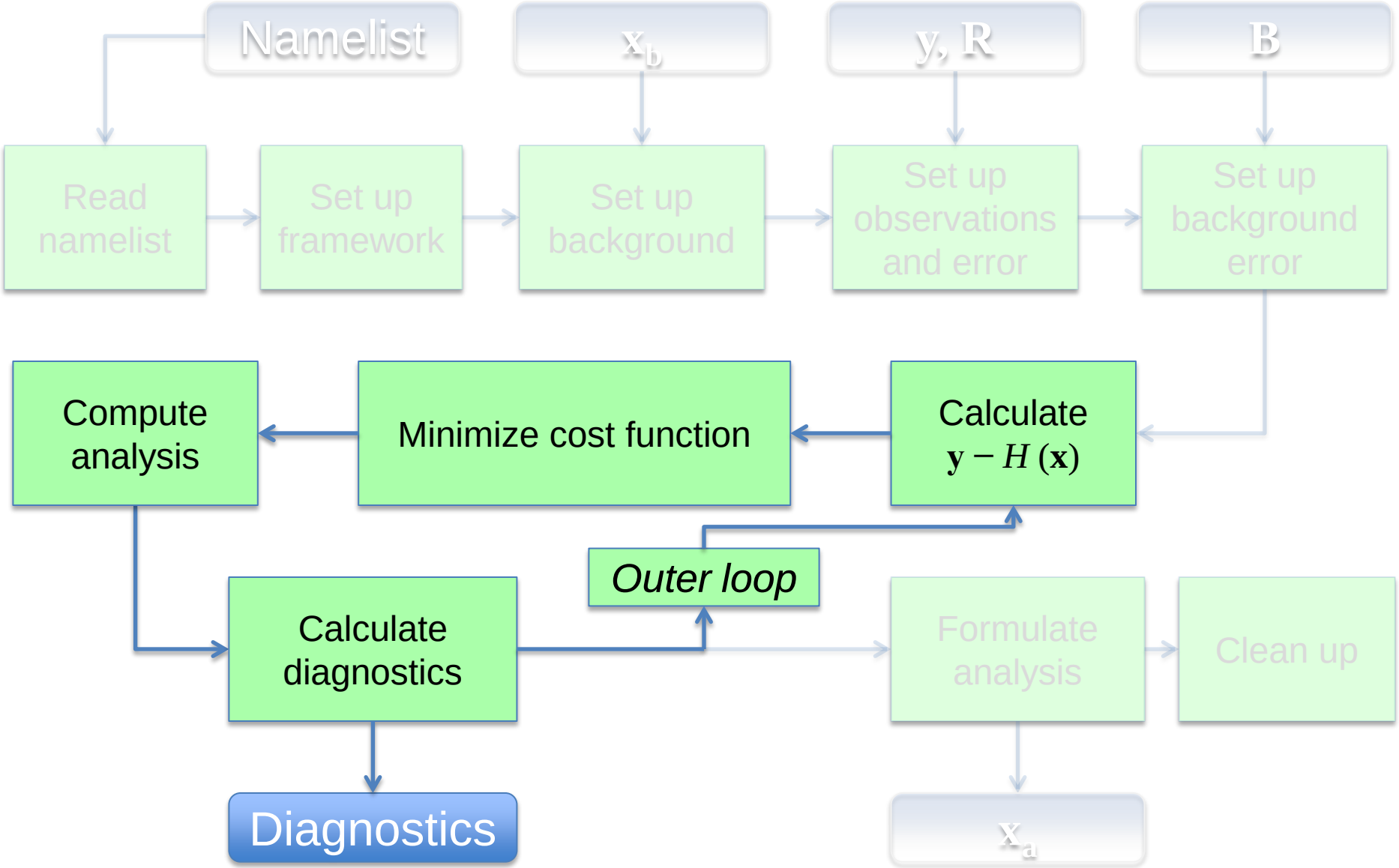
Calling order:

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


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 $\mathbf{y} - H(\mathbf{x})$
Outer loop
Calculate diagnostics
Formulate analysis
Clean up
Diagnostics
x_a

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 ten times

Calling order:

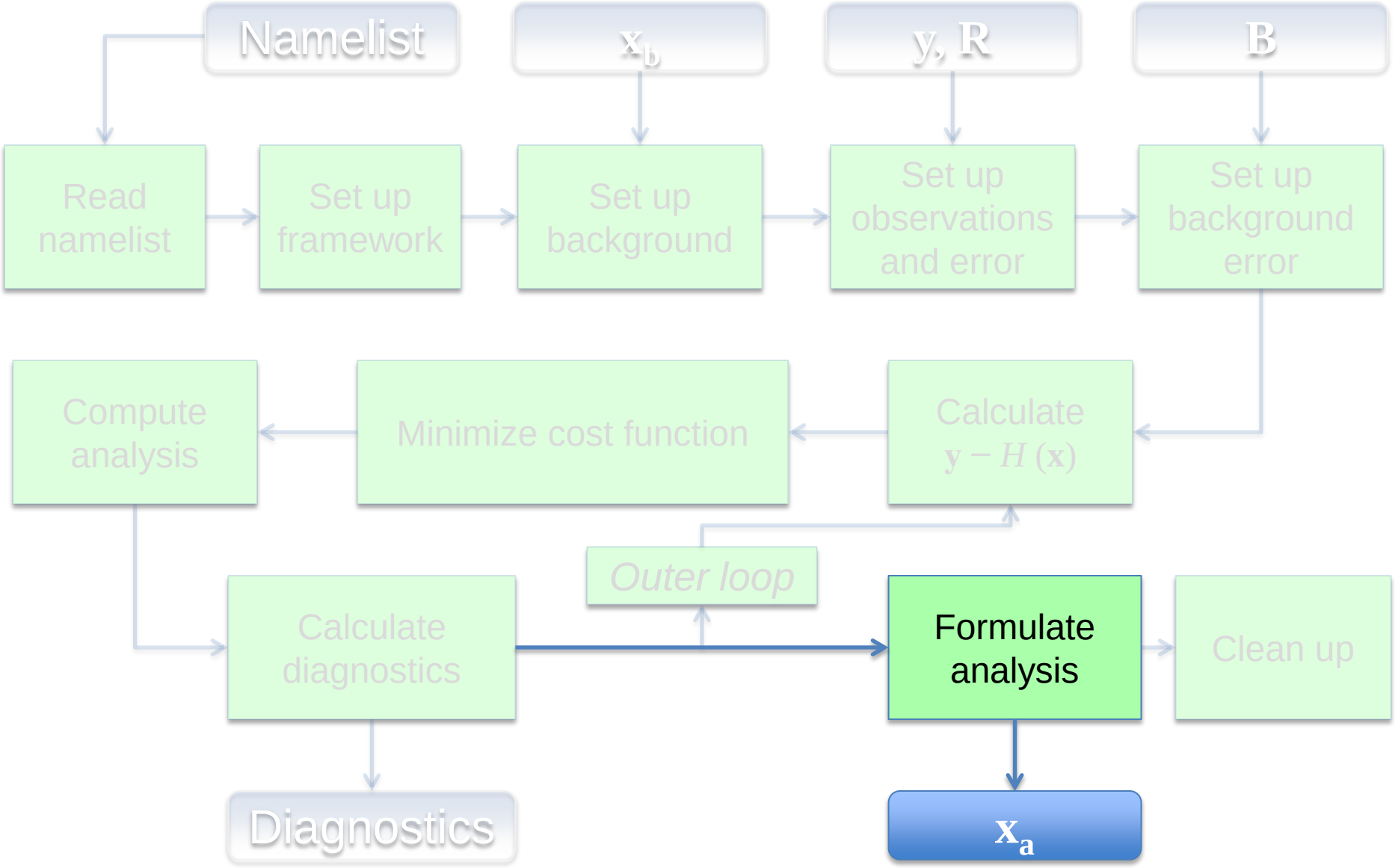
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve

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

- Write analysis file in native WRF format.

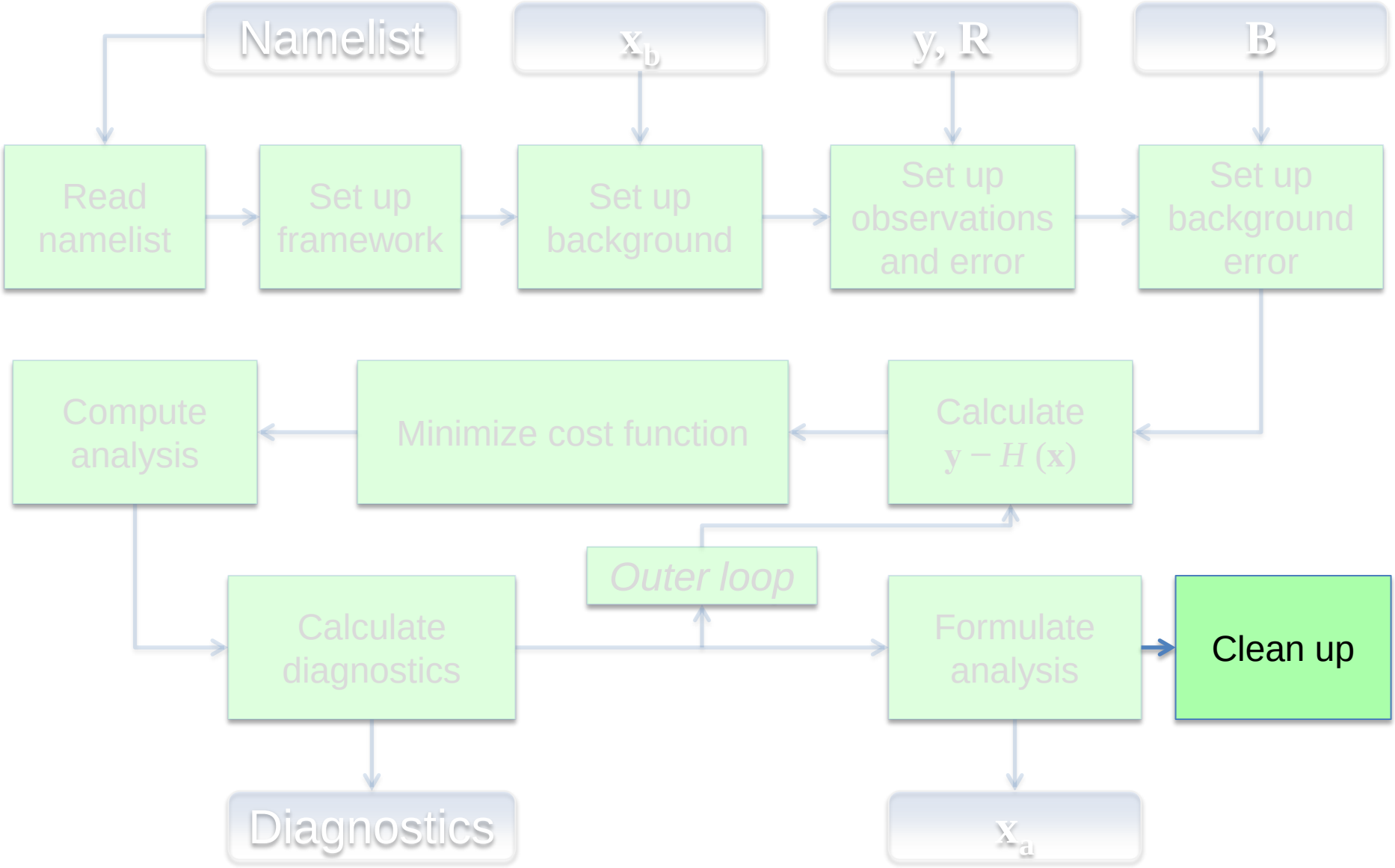
Calling order:

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


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 $\mathbf{y} - H(\mathbf{x})$
Outer loop
Calculate diagnostics
Formulate analysis
Clean up
Diagnostics
x_a

Clean up

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

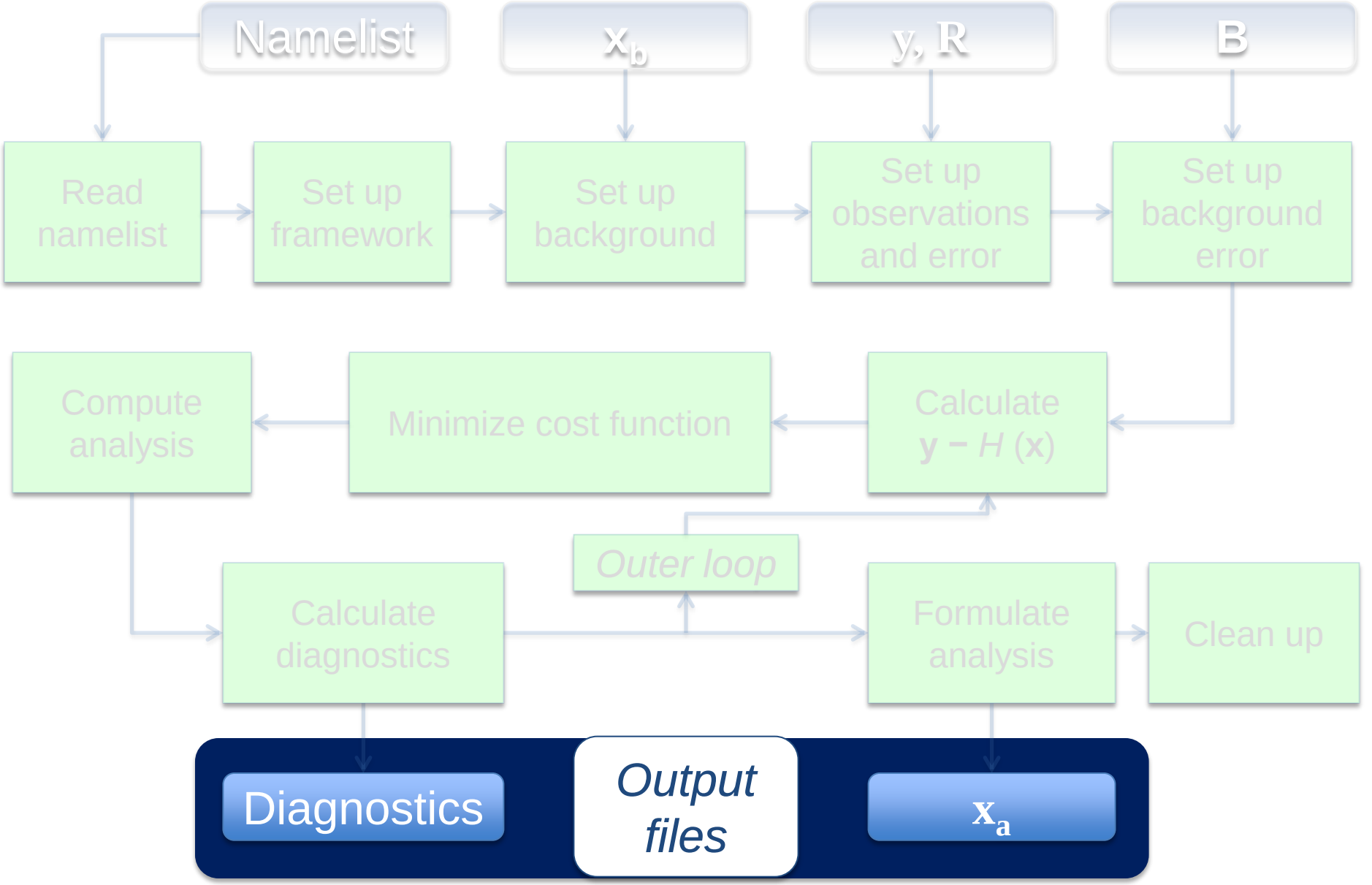
Calling order:

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



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

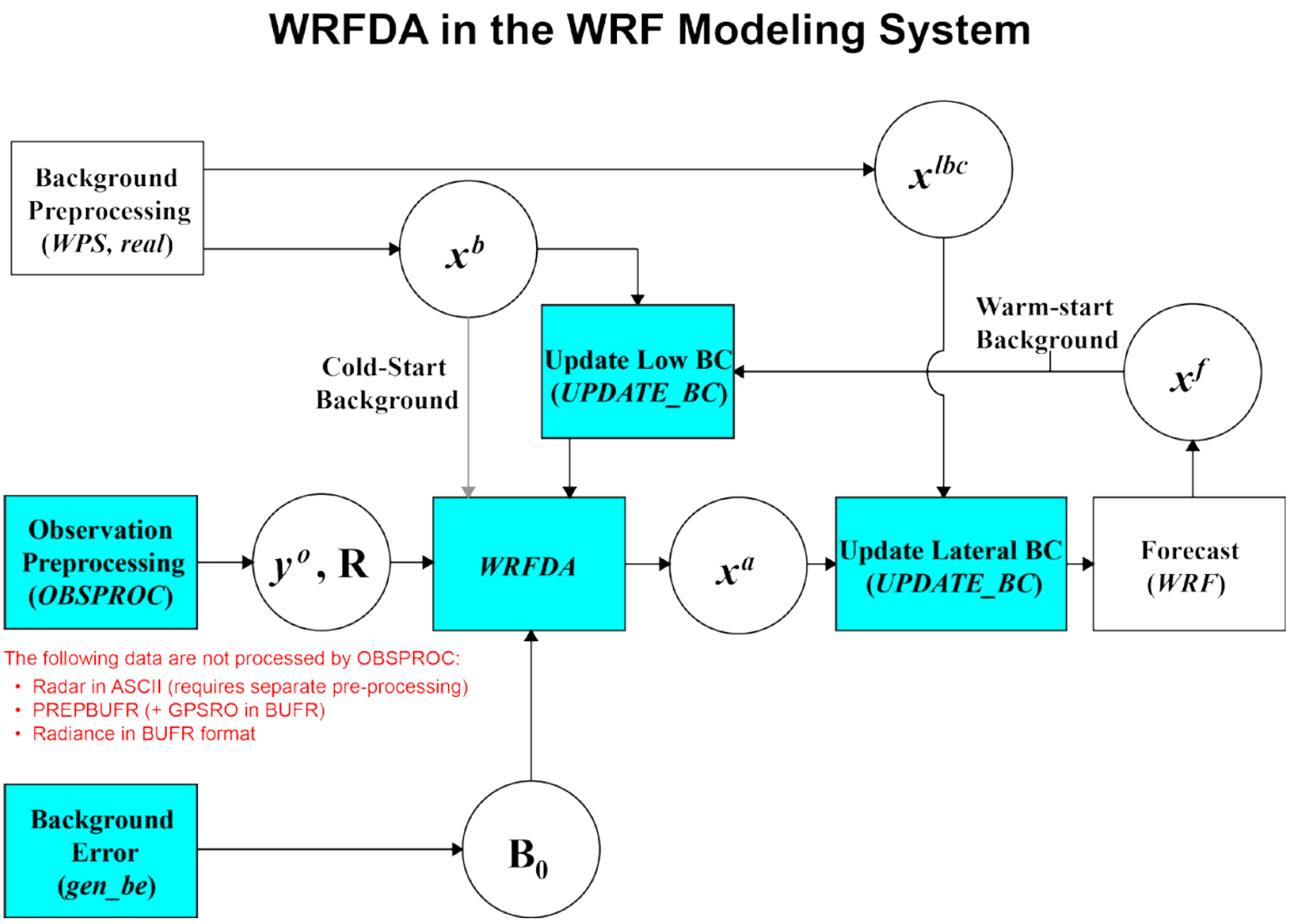
Output files: $\mathbf{x}_a$ (analysis)

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

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

Cycling mode

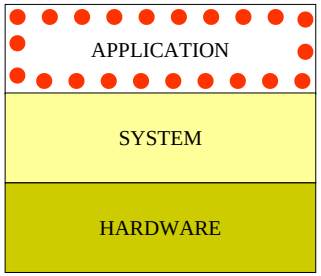


Further reading: User's Guide, Chapter 6, section "Updating WRF Boundary Conditions"

WRFDA System – Outline

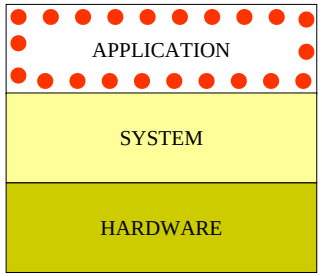
- Introduction
- WRFDA Software Overview
- *Computing Overview*

WRFDA Parallelism

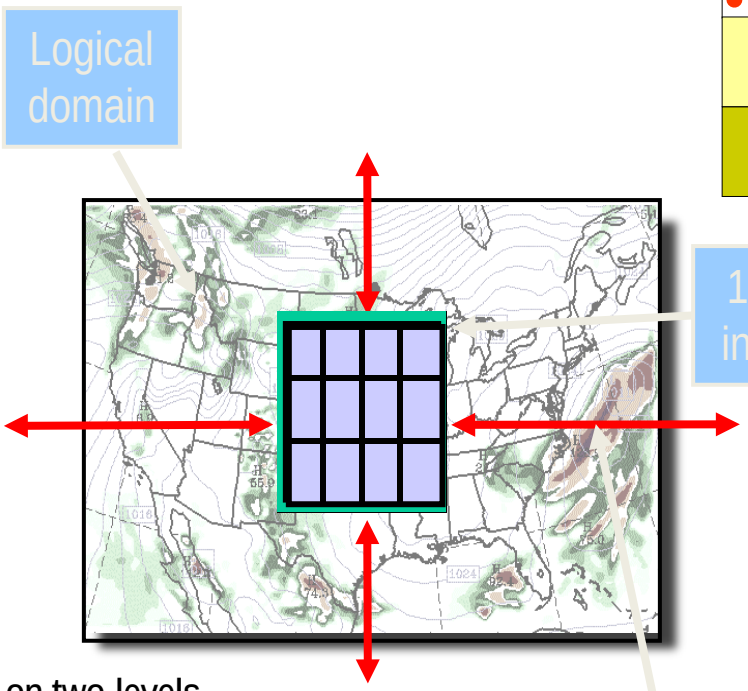


- WRFDA can be run serially or as a parallel job
- WRFDA uses ***domain decomposition*** to divide total amount of work over parallel processes
- The decomposition of the application over processes has two levels:
  - The ***domain*** is first broken up into rectangular pieces that are assigned to MPI (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.

Parallelism in WRFDA: Multi-level Decomposition



- Single version of code for efficient execution on:
  - Distributed-memory
  - Shared-memory (SMP)
  - Clusters of SMPs
  - Vector and microprocessors



Model domains are decomposed for parallelism on two-levels

Patch: section of model domain allocated to a distributed memory node, this is the scope of a minimization layer solver.

Tile: section of a patch allocated to a shared-memory processor within a node; this is also the scope of a observation layer subroutine.

Distributed memory parallelism is over patches; shared memory parallelism is over tiles within patches

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

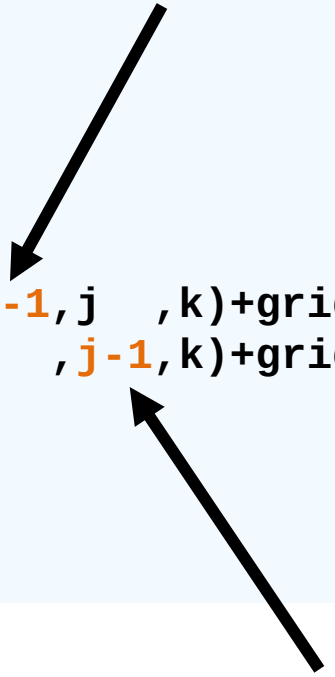
(da_transfer_xatowrf.inc)

```
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
 . . .
```

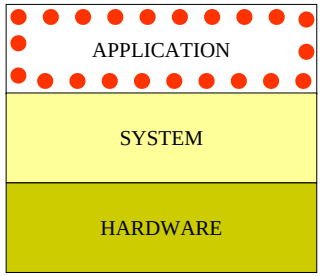
Distributed Memory Communications

(da_transfer_xatowrf.inc)

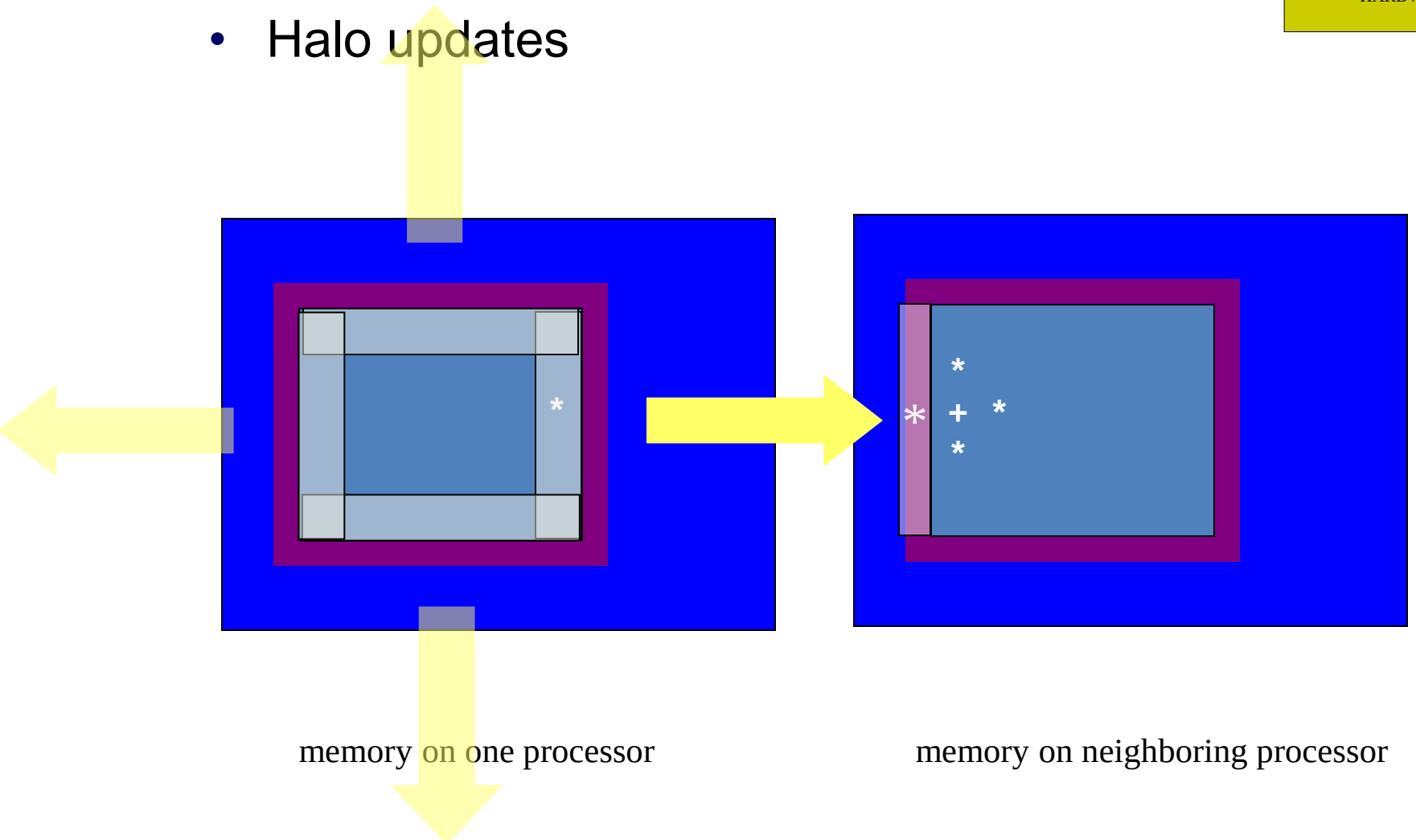
```
subroutine da_transfer_xatowrf(grid)
. . .
 do k=kts,kte
      do j=jts,jte+1
         do i=its,ite+1
           u_cgrid(i,j,k)=0.5*(grid%xa%u(i-1,j  ,k)+grid%xa%u(i,j,k))
           v_cgrid(i,j,k)=0.5*(grid%xa%v(i  ,j-1,k)+grid%xa%v(i,j,k))
         end do
      end do
   end do
 . . .
```



Distributed Memory (MPI) Communications



- Halo updates



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

WRFDA I/O

- Streams: pathways into and out of model
  - Input
    - fg is the name of the input
    - wrfvar_output is the name of output
  - Boundary
    - Only needed for 4DVAR.

Summary

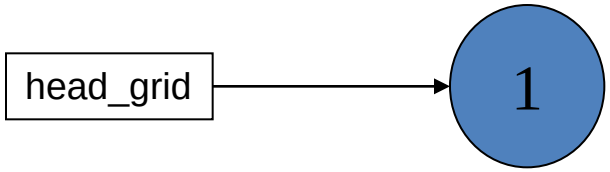
- WRFDA is designed to be an easy-to-use data assimilation system for use with the WRF model
- WRFDA is designed within the WRF Software Framework for rapid development and ease of modification
- WRFDA can be run in parallel for quick assimilation of large amounts of data

Appendix – WRFDA Resources

- WRFDA users page
  - http://www.mmm.ucar.edu/wrf/users/wrfda
  - Download WRFDA source code, test data, related packages and documentation
  - Lists WRFDA news and developments
- Online documentation
  - http://www.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 – Derived Data Structures

- Driver layer
  - All data for a domain is an object, a domain **derived data type** (DDT)
  - The domain DDT is dynamically allocated/deallocated
  - Only one DDT is allowed in WRFDA; it is **head_grid**, defined in frame/module_domain.F
  - WRFDA doesn't support nested domains.



- Every Registry defined **state**, **I1**, and **namelist** variable is contained inside the DDT (locally known as a **grid** of type **domain**), where each node in the tree represents a separate and complete 3D model domain/nest.

Appendix – Derived Data Structures

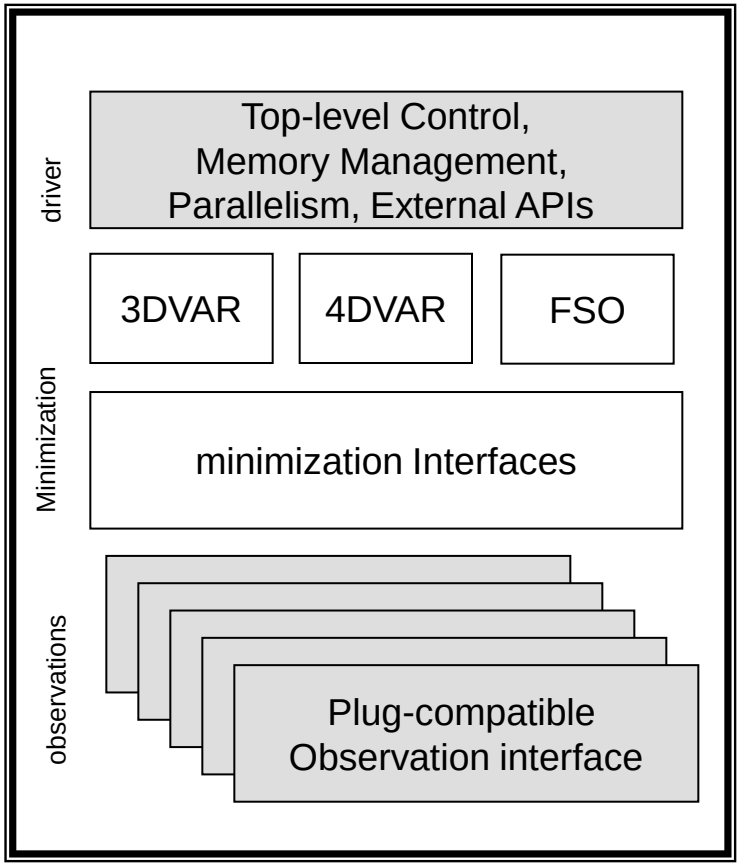
- cvt
  - Real type array to store the control variables
  - It is an all-ZERO array during the first outer loop and will be updated at the end of each outer loop
- xhat
  - Real type array to store the control variables
  - It stores the control variables for each inner loop.
- be
  - It is used to store the background error covariance.

Appendix – Derived Data Structures

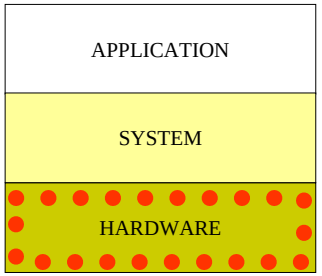
- iv
  - Stores the innovations for each observational type $\mathbf{y} - \mathbf{Hx}$
- ob
  - Stores the observations $\mathbf{y}$
- re
  - Store the residual $\mathbf{y} - \mathbf{H}(\mathbf{x} + \Delta\mathbf{x})$

Appendix – WRFDA structure

- Primarily written in Fortran and C
- Part of the WRF Software Framework
  - Hierarchical organization
  - Multiple functions
  - Plug observation type interface
  - Abstract interfaces (APIs) to external packages
  - Performance-portable

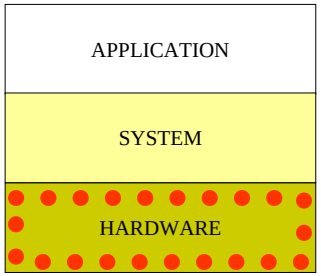


Appendix – Parallel Computing Terms (Hardware)



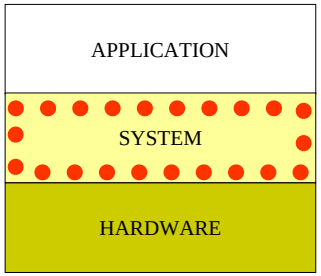
- **Processor**:
  - A device that reads and executes instructions in sequence from a memory device, producing results that are written back to a memory device
- **Node**: One memory device connected to one or more processors.
  - Multiple processors in a node are said to share-memory and this is “shared memory parallelism”
  - They can work together because they can see each other’s memory
  - The latency and bandwidth to memory affect performance

Appendix – Parallel Computing Terms (Hardware)



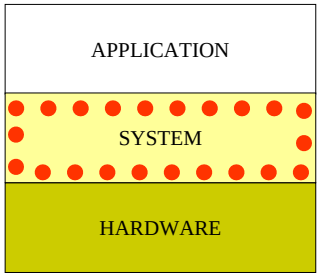
- **Cluster**: Multiple nodes connected by a network
  - The processors attached to the memory in one node can not see the memory for processors on another node
  - For processors on different nodes to work together they must send messages between the nodes. This is "distributed memory parallelism"
- **Network:**
  - Devices and wires for sending messages between nodes
  - Bandwidth – a measure of the number of bytes that can be moved in a second
  - Latency – the amount of time it takes before the first byte of a message arrives at its destination

Appendix – Parallel Computing Terms (Software)



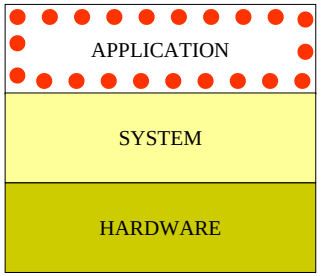
- **Process:**
  - A set of instructions to be executed on a processor
  - Enough state information to allow process execution to stop on a processor and be picked up again later, possibly by another processor
- Processes may be lightweight or heavyweight
  - **Lightweight processes**, e.g. shared-memory threads, store very little state; just enough to stop and then start the process
  - **Heavyweight processes**, e.g. UNIX processes, store a lot more (basically the memory image of the job)

Appendix – Parallel Computing Terms (Software)



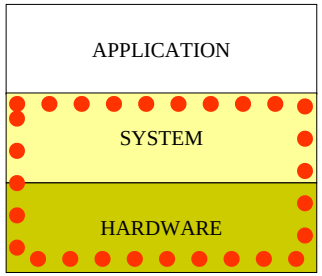
- Every job has at least one heavy-weight *process*.
  - A job with more than one heavy-weight process is a distributed-memory parallel job
  - Even on the same node, heavyweight processes do not share memory
- Within a heavyweight process you may have some number of lightweight processes, called *threads.*
  - Threads are shared-memory parallel; only threads in the same memory space can work together.
  - A thread never exists by itself; it is always inside a heavy-weight process.
- Heavy-weight processes are the vehicles for distributed memory parallelism
- Threads (light-weight processes) are the vehicles for shared-memory parallelism

Appendix – Parallel Computing in WRFDA context



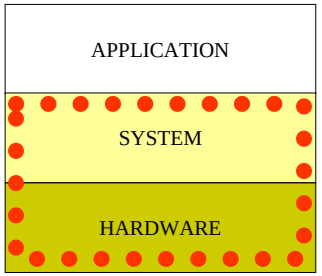
- Since the process model has two levels (heavy-weight and light-weight = MPI and OpenMP), the decomposition of the application over processes has two levels:
  - The ***domain*** is first broken up into rectangular pieces that are assigned to heavy-weight processes. These pieces are called ***patches***
  - The ***patches*** may be further subdivided into smaller rectangular pieces that are called ***tiles***, and these are assigned to ***threads*** within the process.

Appendix – Parallel Computing APIs



- Message Passing Interface – MPI, referred to as the communication layer
- MPI is used to start up and pass messages between multiple heavyweight processes
  - The **mpirun** command controls the number of processes and how they are mapped onto nodes of the parallel machine
  - Calls to MPI routines send and receive messages and control other interactions between processes
  - http://www.mcs.anl.gov/mpi

Appendix – Parallel Computing APIs

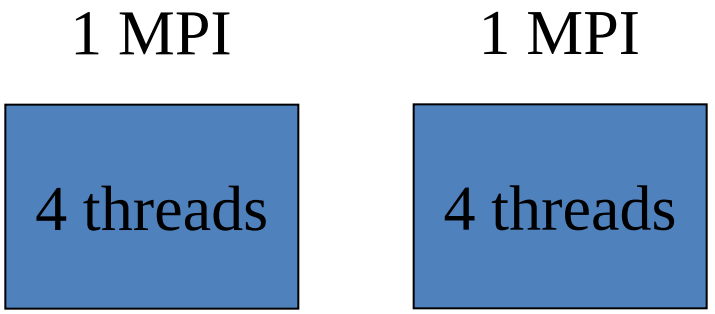


- OpenMP is used to start up and control threads within each process
  - Directives specify which parts of the program are multi-threaded
  - **OpenMP** environment variables determine the number of threads in each process
  - http://www.openmp.org
- OpenMP is usually activated via a compiler option
- MPI is usually activated via the compiler name
- The number of **processes** (number of MPI processes times the number of threads in each process) usually corresponds to the number of **processors**
- **In general, WRFDA should not be run with shared memory!**

Appendix – MPI/OpenMP

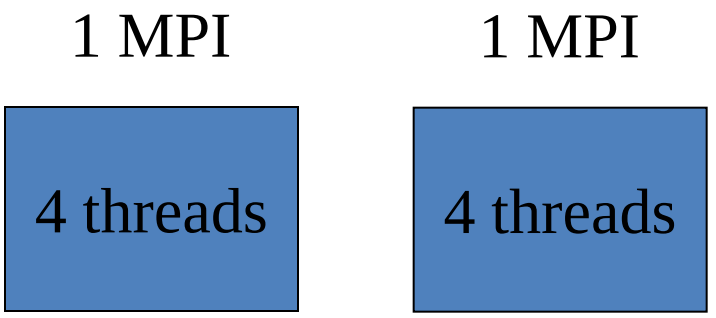
- If the machine consists of 4 nodes, each with 4 processors, how many different ways can you run a job to use all 16 processors?
  - 4 MPI processes, each with 4 threads

```
setenv OMP_NUM_THREADS 4
mpirun –np 4 da_wrfvar.exe
```



  - 8 MPI processes, each with 2 threads

```
setenv OMP_NUM_THREADS 2
mpirun –np 8 da_wrfvar.exe
```



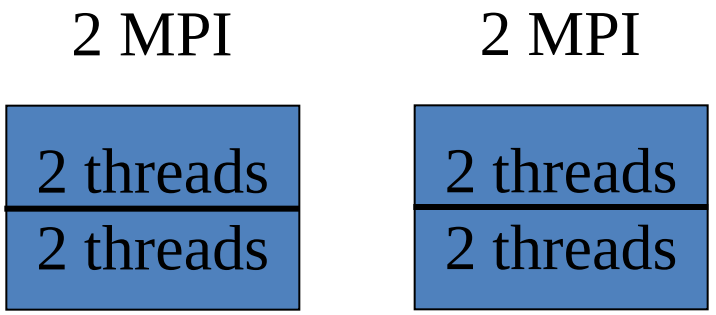
  - 16 MPI processes, each with 1 thread

```
setenv OMP_NUM_THREADS 1
mpirun –np 16 da_wrfvar.exe
```

Appendix – MPI/OpenMP

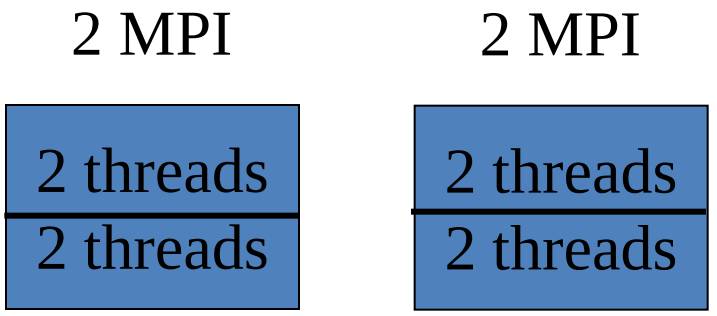
- If the machine consists of 4 nodes, each with 4 processors, how many different ways can you run a job to use all 16 processors?
  - 4 MPI processes, each with 4 threads

```
setenv OMP_NUM_THREADS 4
mpirun –np 4 da_wrfvar.exe
```



  - 8 MPI processes, each with 2 threads

```
setenv OMP_NUM_THREADS 2
mpirun –np 8 da_wrfvar.exe
```



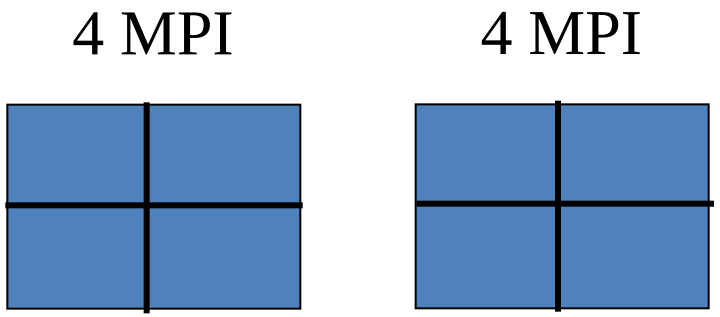
  - 16 MPI processes, each with 1 thread

```
setenv OMP_NUM_THREADS 1
mpirun –np 16 da_wrfvar.exe
```

Appendix – MPI/OpenMP

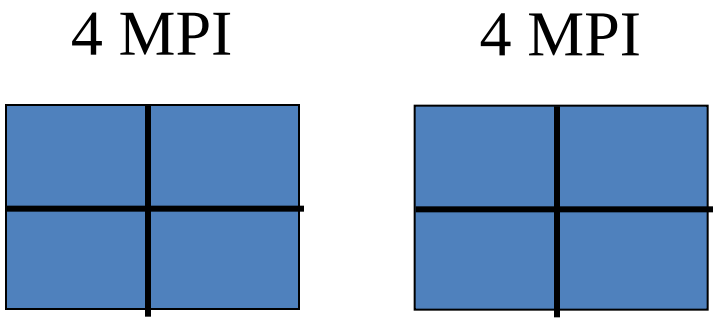
- If the machine consists of 4 nodes, each with 4 processors, how many different ways can you run a job to use all 16 processors?
  - 4 MPI processes, each with 4 threads

```
setenv OMP_NUM_THREADS 4
mpirun –np 4 da_wrfvar.exe
```



  - 8 MPI processes, each with 2 threads

```
setenv OMP_NUM_THREADS 2
mpirun –np 8 da_wrfvar.exe
```



  - 16 MPI processes, each with 1 thread

```
setenv OMP_NUM_THREADS 1
mpirun –np 16 da_wrfvar.exe
```

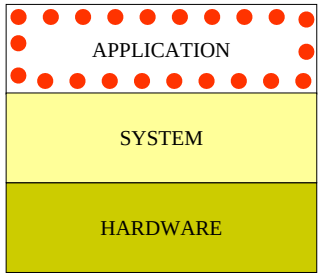
Appendix – MPI/OpenMP

- Note, since there are 4 nodes, we can never have fewer than 4 MPI processes because nodes do not share memory

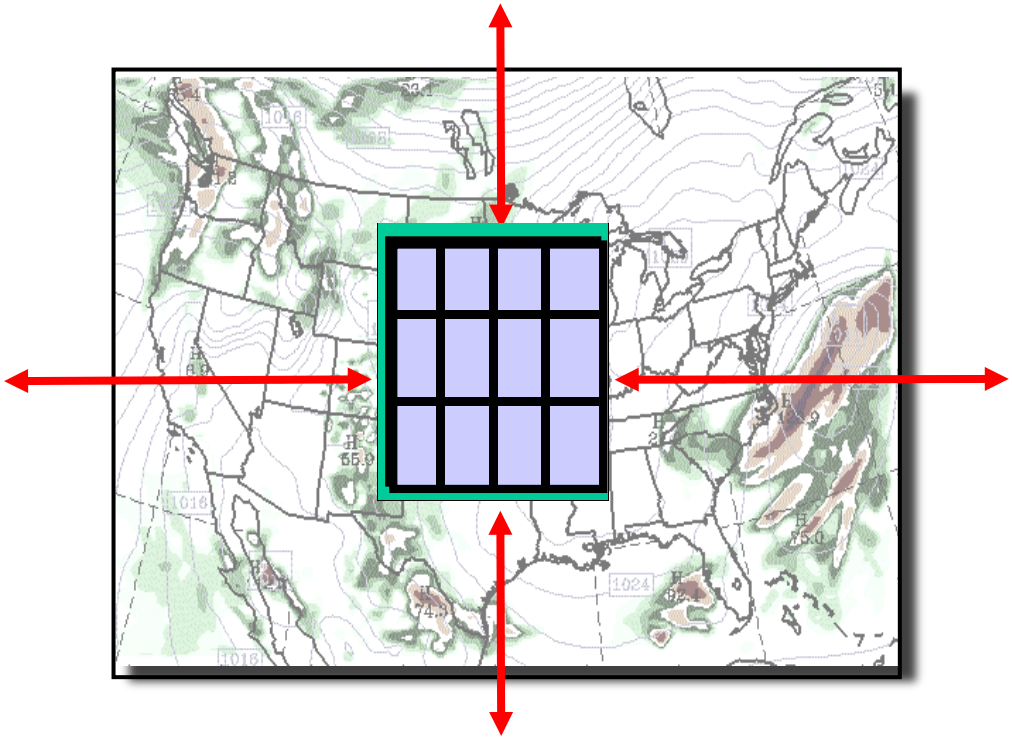
- What happens on this same machine for the following?

```
setenv OMP_NUM_THREADS 8
mpirun –np 32 da_wrfvar.exe
```

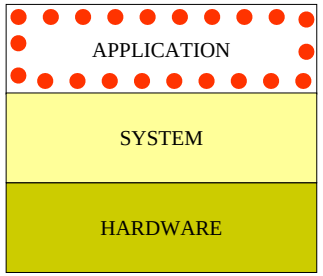
Distributed Memory (MPI) Communications



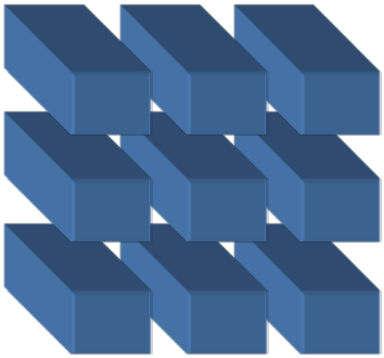
- Halo updates
- Parallel transposes



Distributed Memory (MPI) Communications



- Halo updates
- Parallel transposes

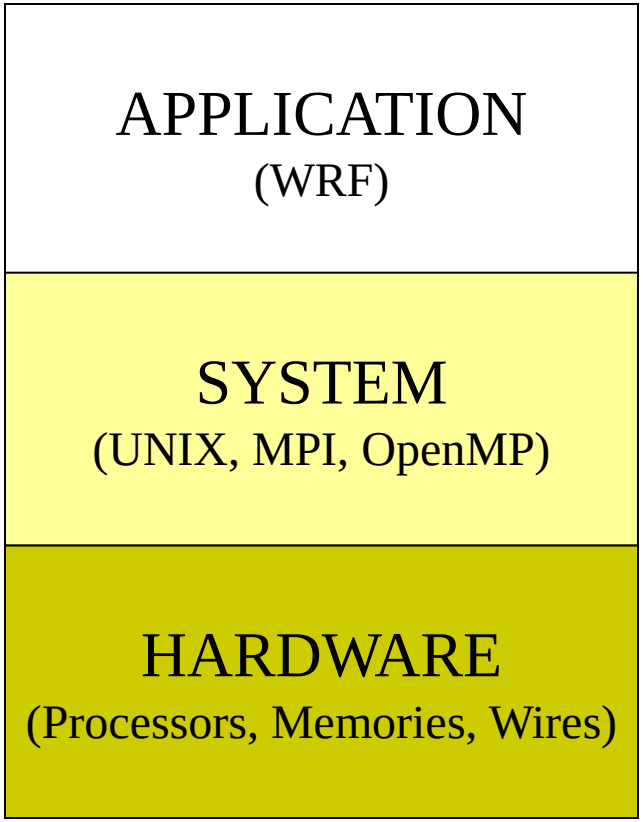


all y on patch

all z on patch

all x on patch

Review – Computing Overview



		Distributed Memory Parallel		Shared Memory Parallel
Domain	*contains*	Patches	*contain*	Tiles
Job	*contains*	Processes	*contain*	Threads
Cluster	*contains*	Nodes	*contain*	Processors

Appendix – 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

Appendix – WRFDA History

- Developed from MM5 3DVar beginning around 2002, first version (2.0) released December 2003
- Developed and supported by WRFDA group of MMM, part of NESL
- Requirements emphasize flexibility over a range of platforms, applications, users, performance
- Current release WRFDA v3.5 (April 2013)
- Shares the WRF Software Framework

Appendix – WRFDA and J

$$J(x) = \frac{1}{2}\left(x - \boldsymbol{x^b}\right)^{\mathrm{T}} \mathbf{B_0}^{-1}\left(x - \boldsymbol{x^b}\right) + \frac{1}{2}\left(y_0 - H(x)\right)^{\mathrm{T}} \mathbf{R}^{-1}\left(y_0 - H(x)\right)$$

- Model background ($\boldsymbol{x^b}$)
- Background error ($\mathbf{B_0}$)
- Observations (y_0) and their associated error statistics ($\mathbf{R}$)
- Minimize this cost function ($J(x)$) to find the analysis ($\boldsymbol{x^a}$)
- Run forecast, repeat for cycling mode

Appendix – WRFDA and J

$$J(x) = \frac{1}{2}\left(x - \boldsymbol{x^b}\right)^{\mathrm{T}} \mathbf{B_0}^{-1}\left(x - \boldsymbol{x^b}\right) + \frac{1}{2}\left(y_0 - H(x)\right)^{\mathrm{T}} \mathbf{R}^{-1}\left(y_0 - H(x)\right)$$

- Model background ($\boldsymbol{x^b}$) is the "first guess" of the atmospheric state before data assimilation
- The background can be provided by WPS and real.exe, or a full WRF forecast

Appendix – WRFDA and J

$$J(x) = \frac{1}{2}\left(x - \boldsymbol{x^b}\right)^{\mathrm{T}} \mathbf{B_0}^{-1}\left(x - \boldsymbol{x^b}\right) + \frac{1}{2}\left(y_0 - H(x)\right)^{\mathrm{T}} \mathbf{R}^{-1}\left(y_0 - H(x)\right)$$

- Background error covariance ($\mathbf{B_0}$) describes the relationship between different errors in the background
- Arguably the most important ingredient to a successful forecast
- Several options available:
  - cv_options=3; generic NCEP Background Error model
    - This is recommended for testing and debugging **only**
  - cv_options=5; the NCAR Background Error model
    - The default and recommended BE formulation
    - Requires gen_be (learned in later presentation)
  - cv_options=6; Multivariate Background Error (MBE) statistics
    - Not officially supported

Appendix – WRFDA and J

$$J(x) = \frac{1}{2}\left(x - \boldsymbol{x^b}\right)^{\mathrm{T}} \mathbf{B_0}^{-1}\left(x - \boldsymbol{x^b}\right) + \frac{1}{2}\left(y_0 - H(x)\right)^{\mathrm{T}} \mathbf{R}^{-1}\left(y_0 - H(x)\right)$$

- Observations (y_0) and their associated errors ($\mathbf{R}$) are essential to any data assimilation process
- WRFDA can assimilate a wide variety of observations
  - Conventional observations
    - Includes radiosonde, ships, surface, etc.
    - Should be in LITTLE_R format for ingest into OBSPROC
  - Satellite radiance data
    - Can assimilate data from dozens of instruments
    - Assimilated directly in BUFR format
    - Requires radiative transfer model (CRTM or RTTOV)
- Radar velocity and reflectivity, accumulated precipitation, others