WRF Data Assimilation System

Michael Kavulich

Special thanks to:

Xin Zhang, Xiang-Yu Huang

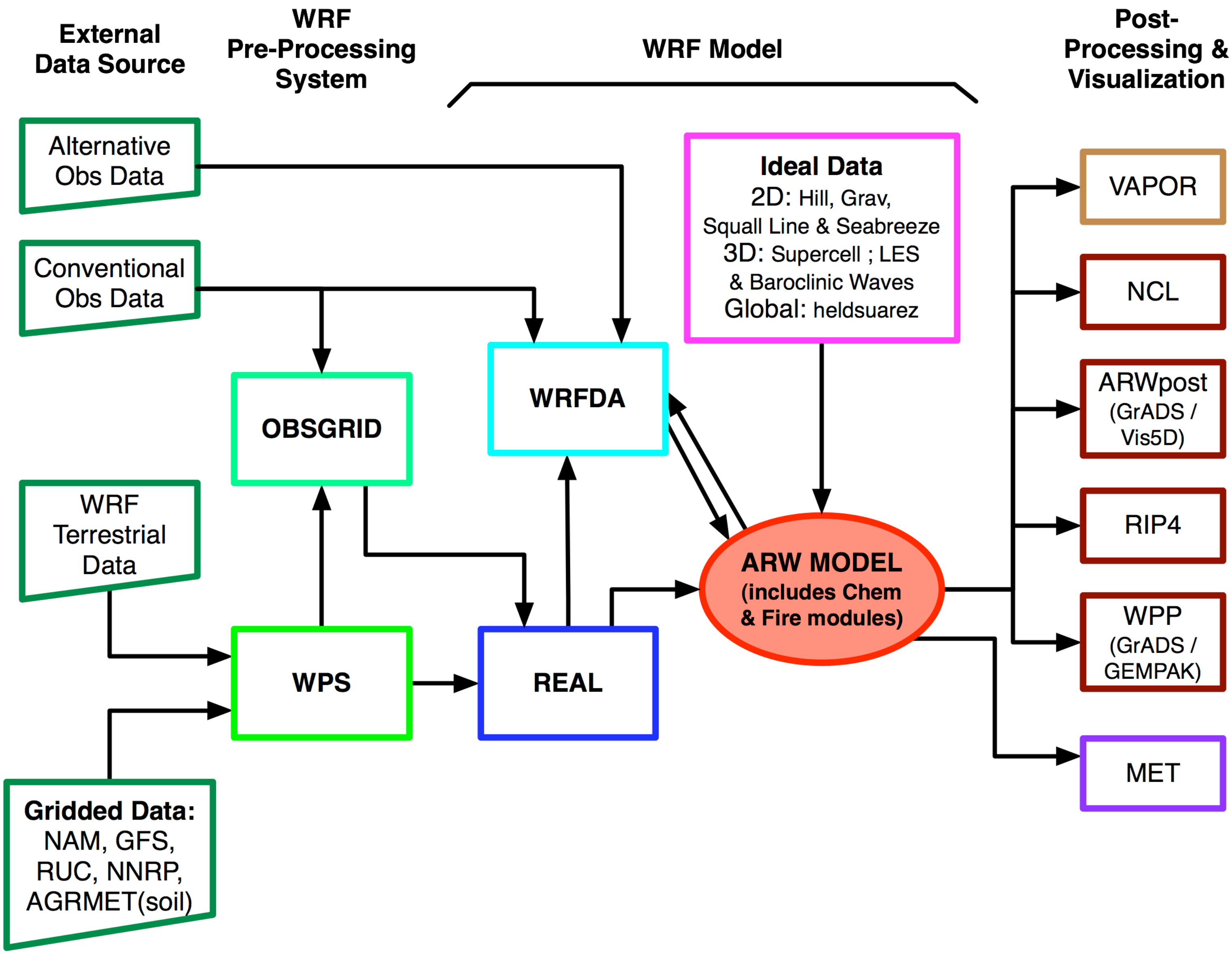
WRFDA Tutorial, July 2012, NCAR

Many slides are borrowed from WRF software lectures

Outline

- *Introduction*
- WRFDA Overview
- Computing Overview
- WRFDA Software Overview

WRF Modeling System Flow Chart



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

Introduction – 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 v3.4 (April 2012)
- Shares the WRF Software Framework

Outline

- Introduction
- *WRFDA Overview*
- Computing Overview
- WRFDA Software Overview

WRFDA Overview

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

- 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

WRFDA Software Overview

- Introduction
- WRFDA Overview
  - *Model background*
  - Background error
  - Observations
  - Run WRFDA
  - Cycling mode
- Computing Overview
- WRFDA Software Overview

Model background

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

- 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

WRFDA Software Overview

- Introduction
- WRFDA Overview
  - Model background
  - *Background error*
  - Observations
  - Run WRFDA
  - Cycling mode
- Computing Overview
- WRFDA Software Overview

Background error statistics

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

- 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

WRFDA Software Overview

- Introduction
- WRFDA Overview
  - Model background
  - Background error
  - *Observations*
  - Run WRFDA
  - Cycling mode
- Computing Overview
- WRFDA Software Overview

Observations

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

- 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

WRFDA Software Overview

- Introduction
- WRFDA Overview
  - Model background
  - Background error
  - Observations
  - *Run WRFDA*
  - Cycling mode
- Computing Overview
- WRFDA Software Overview

Run WRFDA

- Once the background, background error, and observations are prepared, WRFDA is ready to run
- Detailed installation/run instructions will take place in the next talk
- The output is the analysis, which can be used for research or to initialize a WRF forecast
- To initialize a WRF forecast, the boundary conditions must be updated, using da_update_bc.exe

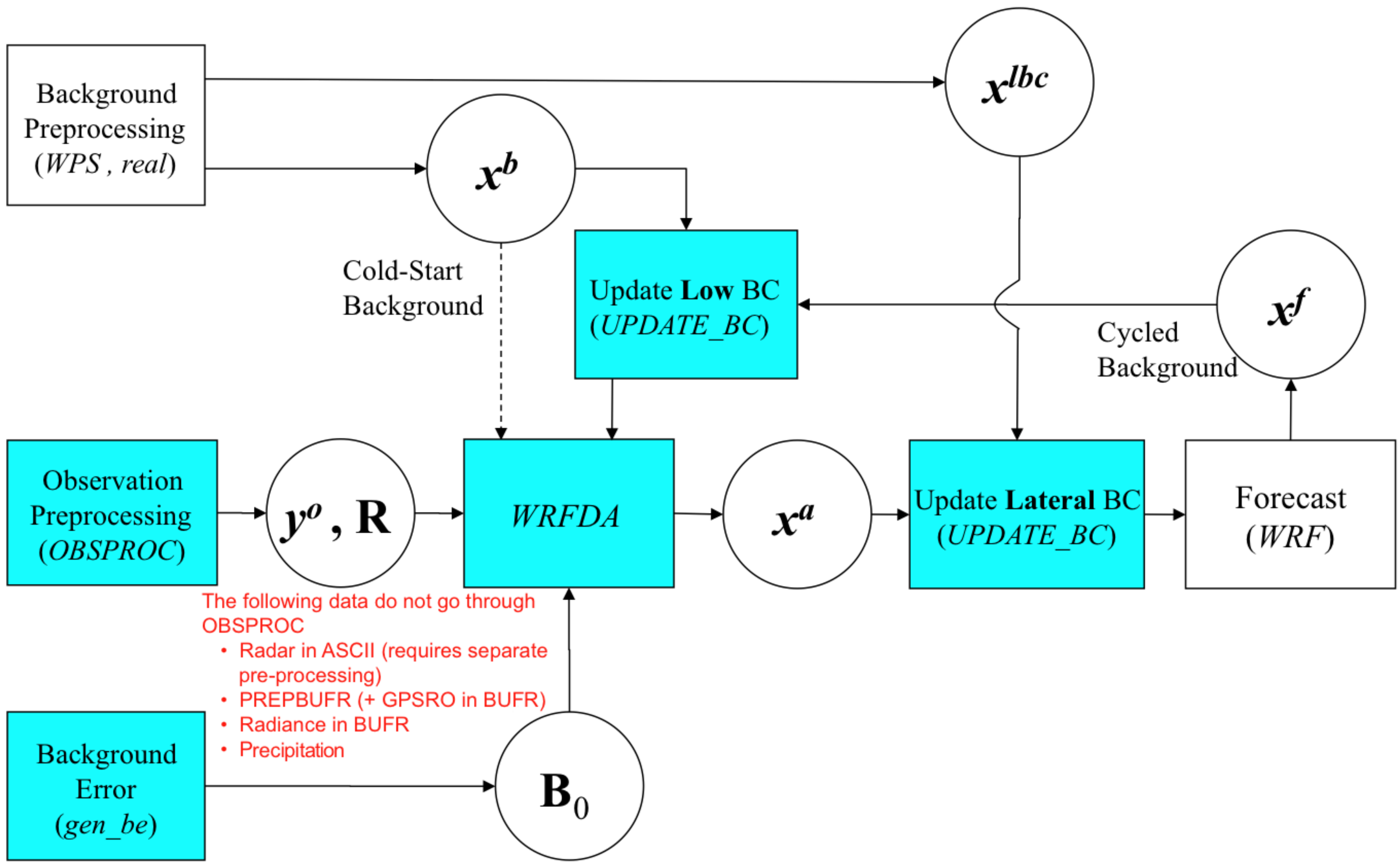
WRFDA Software Overview

- Introduction
- WRFDA Overview
  - Model background
  - Background error
  - Observations
  - Run WRFDA
  - *Cycling mode*
- Computing Overview
- WRFDA Software Overview

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 some specialized boundary condition updating

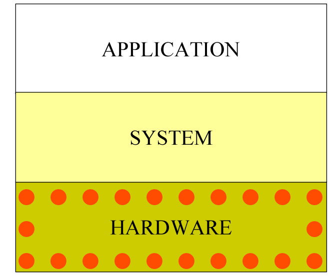
WRFDA in the WRF Modeling System



Outline

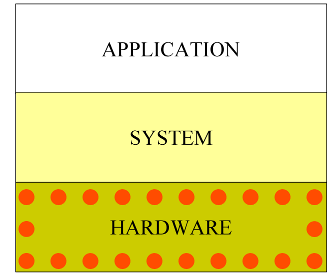
- Introduction
- WRFDA Overview
- *Computing Overview*
- WRFDA Software Overview

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

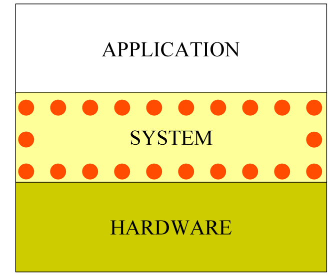
Parallel Computing Terms -- Hardware



- **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

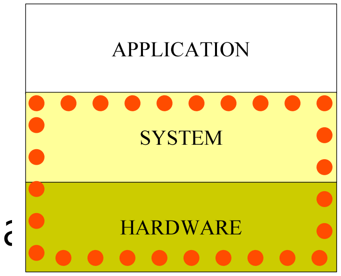
Parallel Computing Terms – System Software

"The only thing one does directly with hardware is pay for it."
John's Zeroth Law of Computing



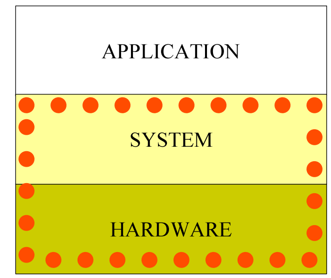
- **Process:**
  - A set of instructions to be executed on a processor
  - Enough state information to allow process execution to stop on a processor and be picked up again later, possibly by another processor
- Processes may be lightweight or heavyweight
  - **Lightweight processes**, e.g. shared-memory 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)

Jobs, Processes, and Hardware



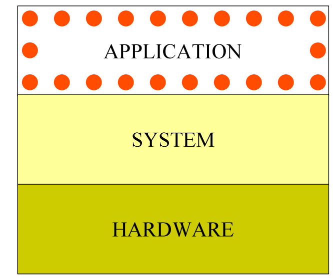
- Message Passing Interface – MPI, referred to as the communication la
- MPI is used to start up and pass messages between multiple heavyweight processes
  - The **mpirun** command controls the number of processes and how they are mapped onto nodes of the parallel machine
  - Calls to MPI routines send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

Jobs, Processes, and Hardware



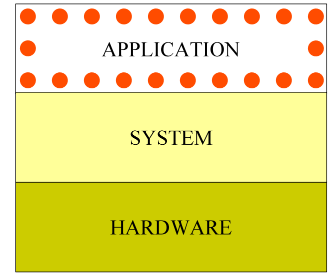
- OpenMP is used to start up and control threads within each process
  - Directives specify which parts of the program are multi-threaded
  - **OpenMP** environment variables determine the number of threads in each process
  - http://www.openmp.org
- OpenMP is usually activated via a compiler option
- 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!**

Application: WRFDA



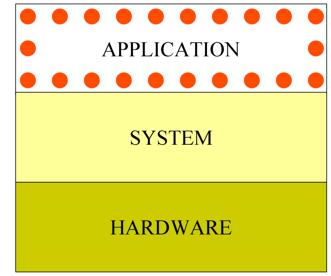
- WRFDA can be run serially or as a parallel job
- WRFDA uses ***domain decomposition*** to divide total amount of work over parallel processes

Application: WRFDA

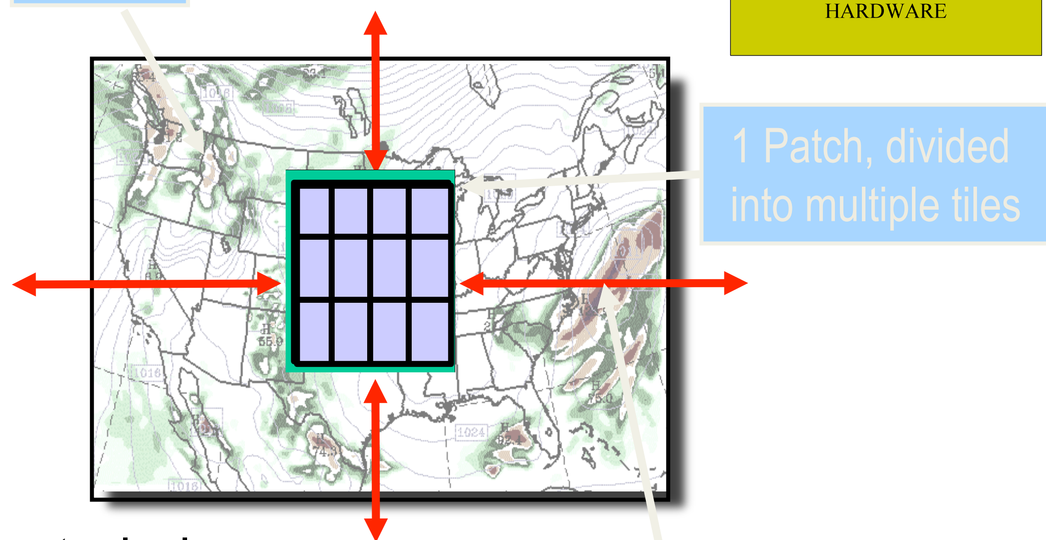


- The decomposition of the application over processes has two levels:
  - The *domain* is first broken up into rectangular pieces that are assigned to MPI (distributed memory) processes. These pieces are called *patches*
  - The *patches* may be further subdivided into smaller rectangular pieces that 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

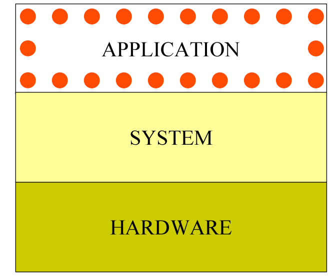
```
                      (da_transfer_xatowrf.inc)

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

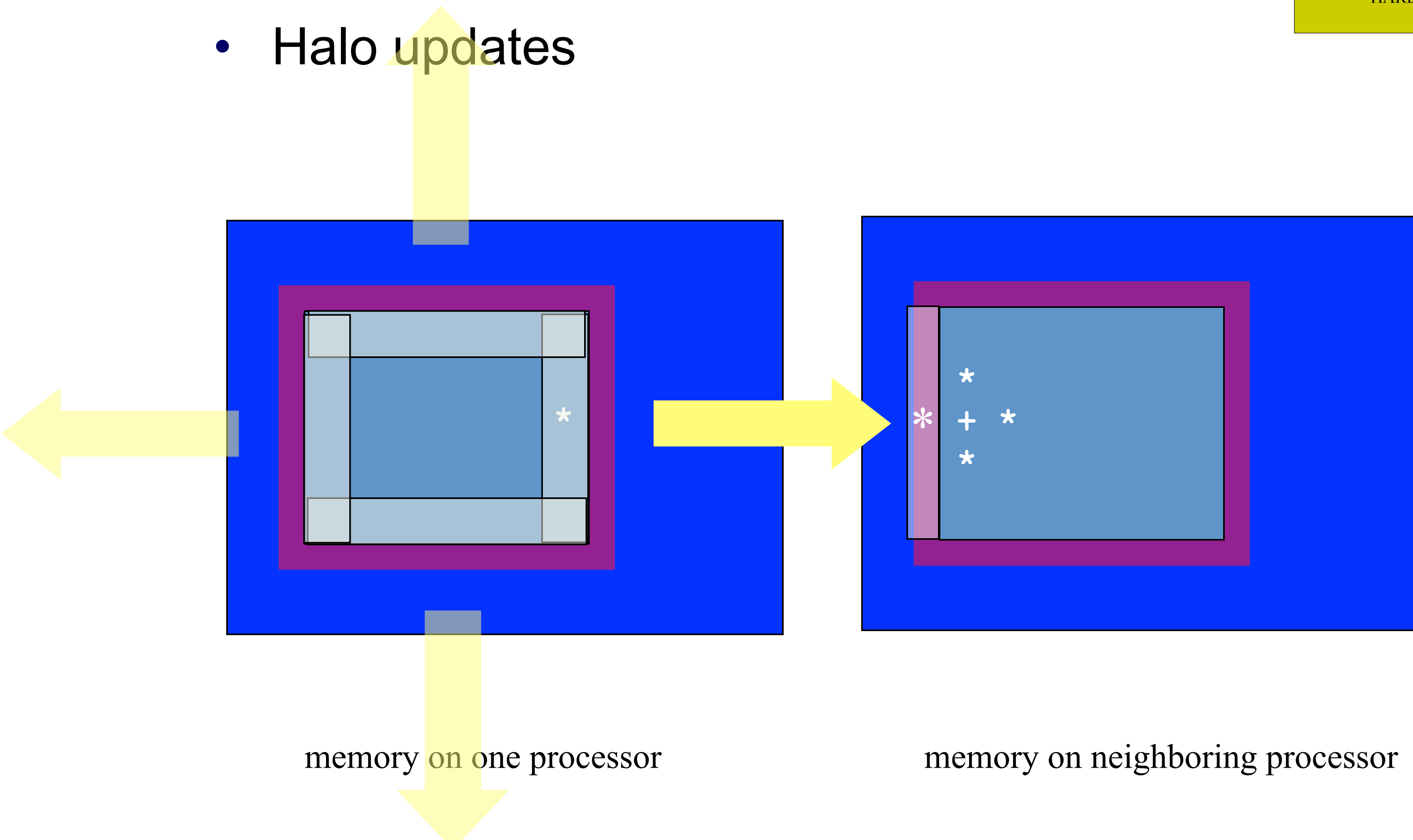
(da_transfer_xatowrf.inc)

```
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
 . . .
```

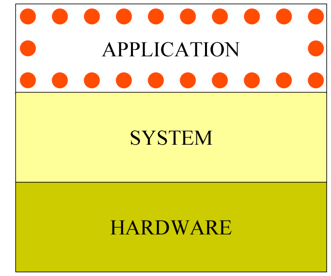
Distributed Memory MPI Communications



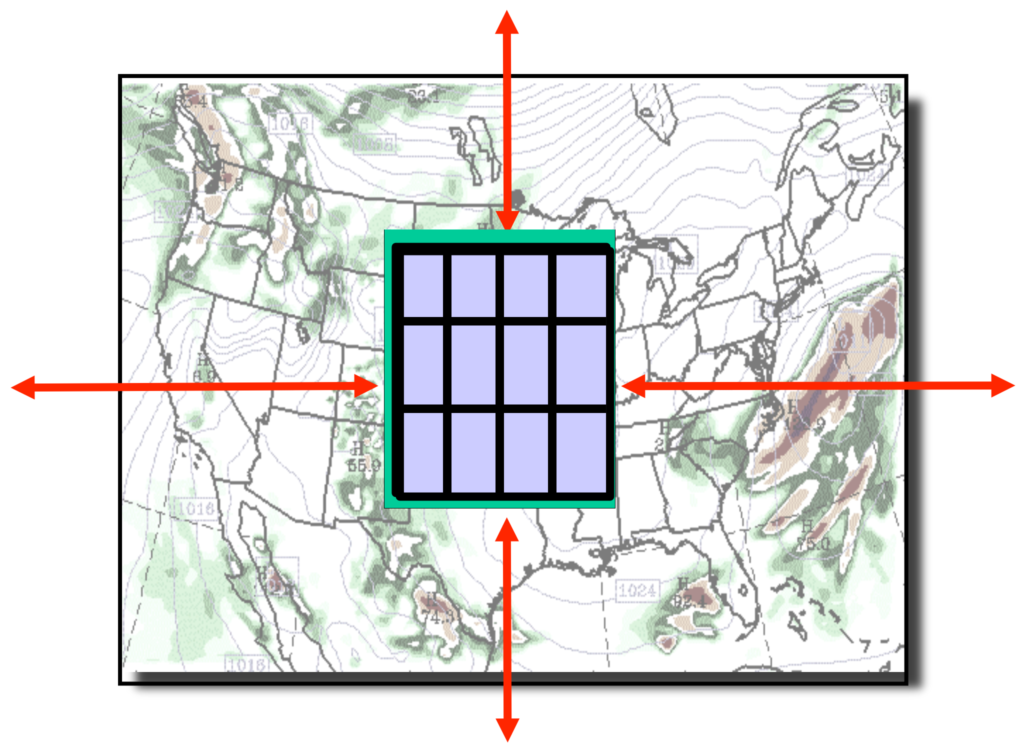
- Halo updates



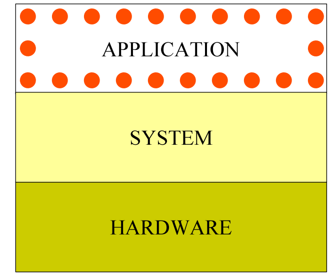
Distributed Memory (MPI) Communications



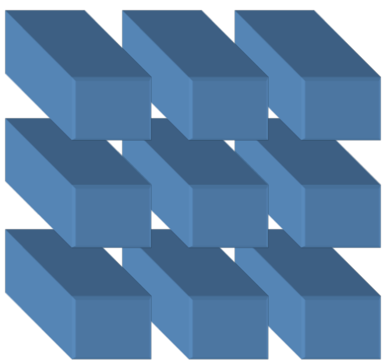
- Halo updates
- Parallel transposes



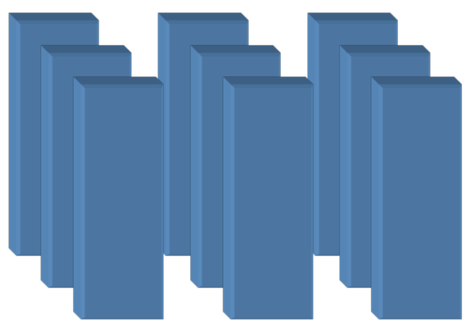
Distributed Memory (MPI) Communications



- 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
APPLICATION (WRF)	Domain	*contains*	Patches	*contain*	Tiles
SYSTEM (UNIX, MPI, OpenMP)	Job	*contains*	Processes	*contain*	Threads
HARDWARE (Processors, Memories, Wires)	Cluster	*contains*	Nodes	*contain*	Processors

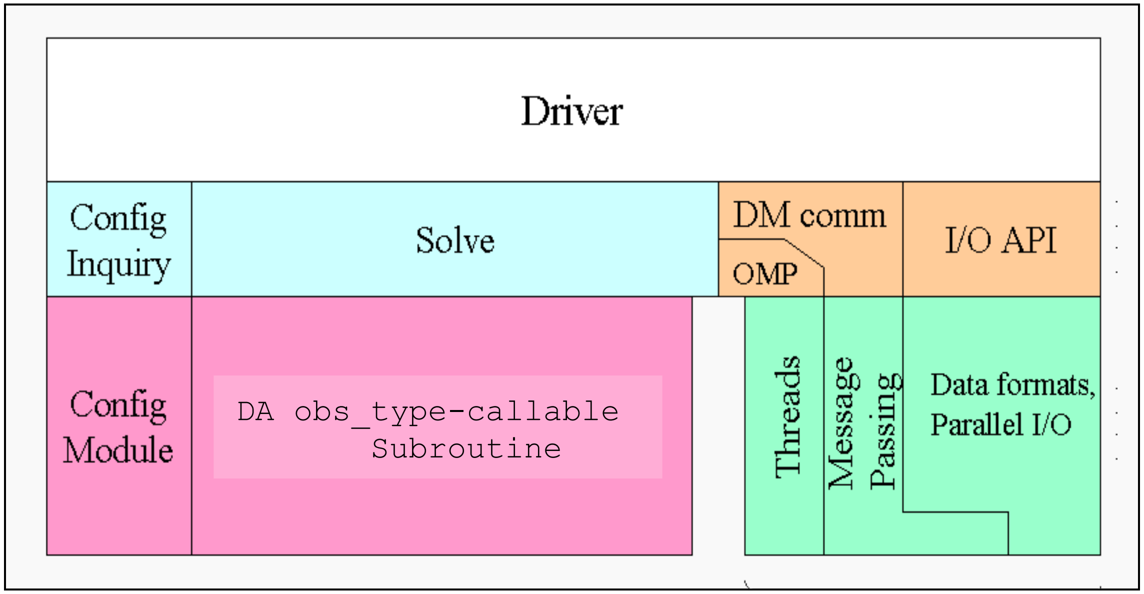
Outline

- Introduction
- WRFDA Overview
- Computing Overview
- *WRFDA Software Overview*

WRFDA Software Overview

- Introduction
- WRFDA Overview
- Computing Overview
- WRFDA Software Overview
  - *Architecture*
  - Data Structures
  - I/O

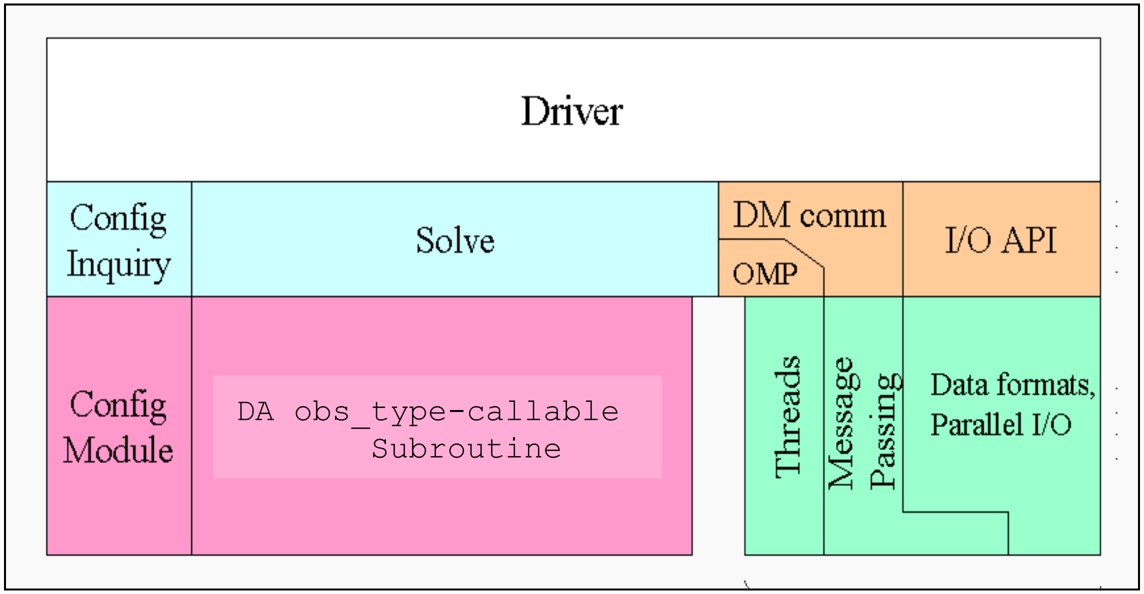
WRFDA Software – Architecture

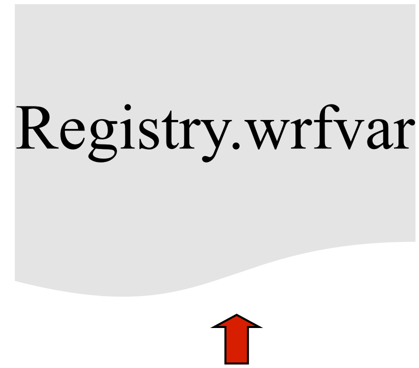


Registry.wrfvar

- 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.

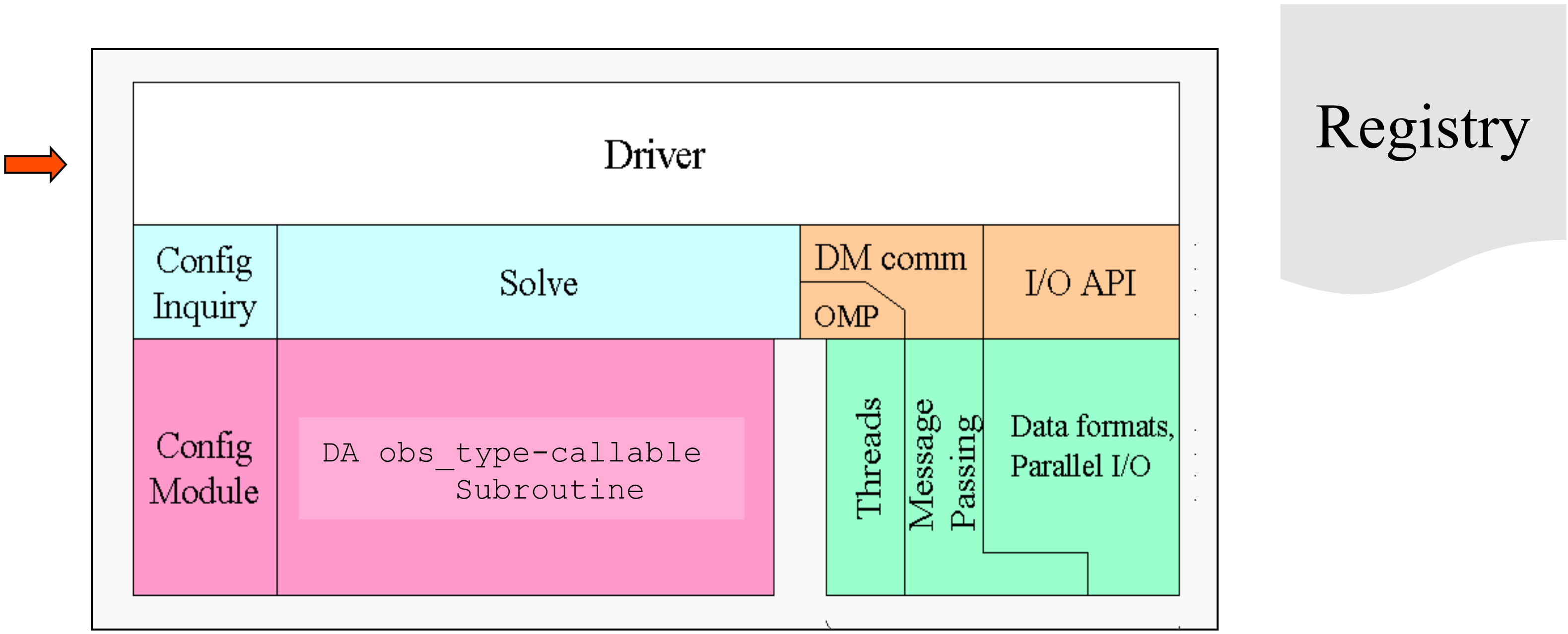
WRFDA Software – Architecture





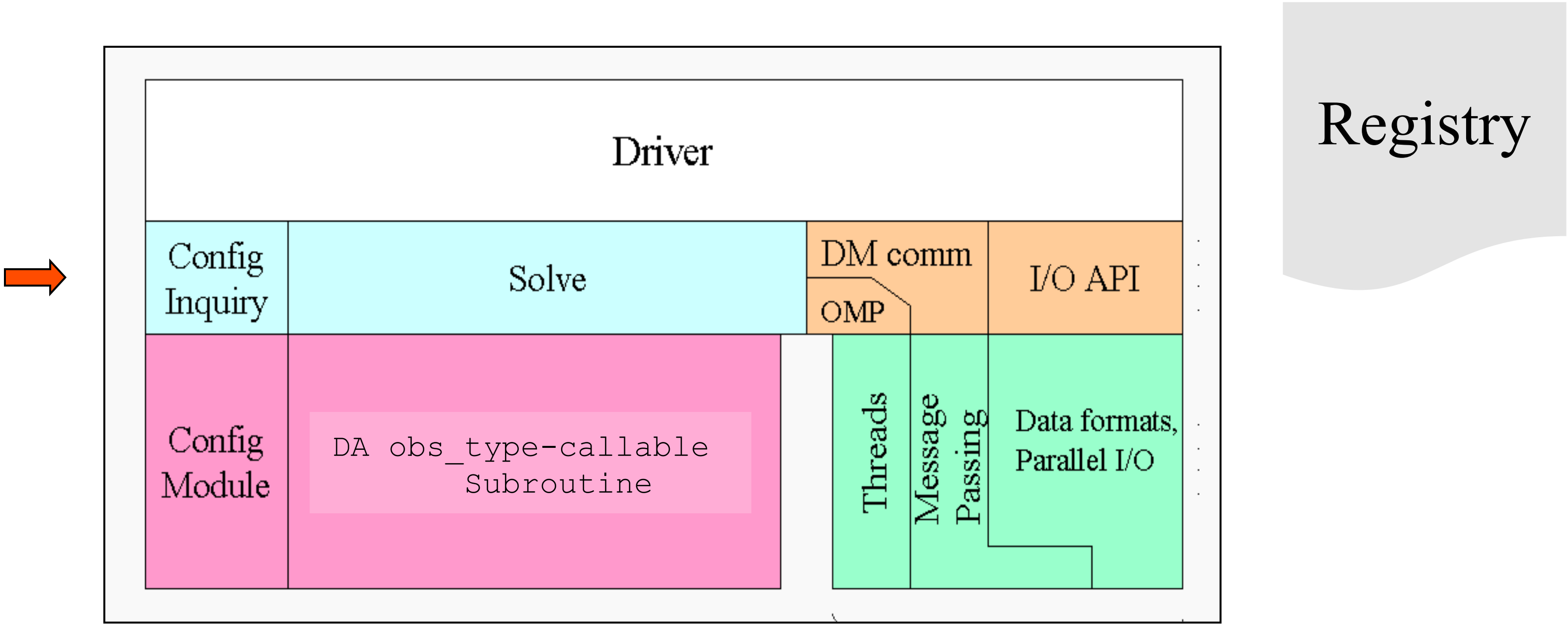
- Registry: an "Active" data dictionary
  - Tabular listing of model state and attributes
  - Large sections of interface code generated automatically
  - Scientists manipulate model state simply by modifying Registry, without further knowledge of code mechanics
  - Registry.wrfvar is the dictionary for WRFDA

WRFDA Software – Architecture



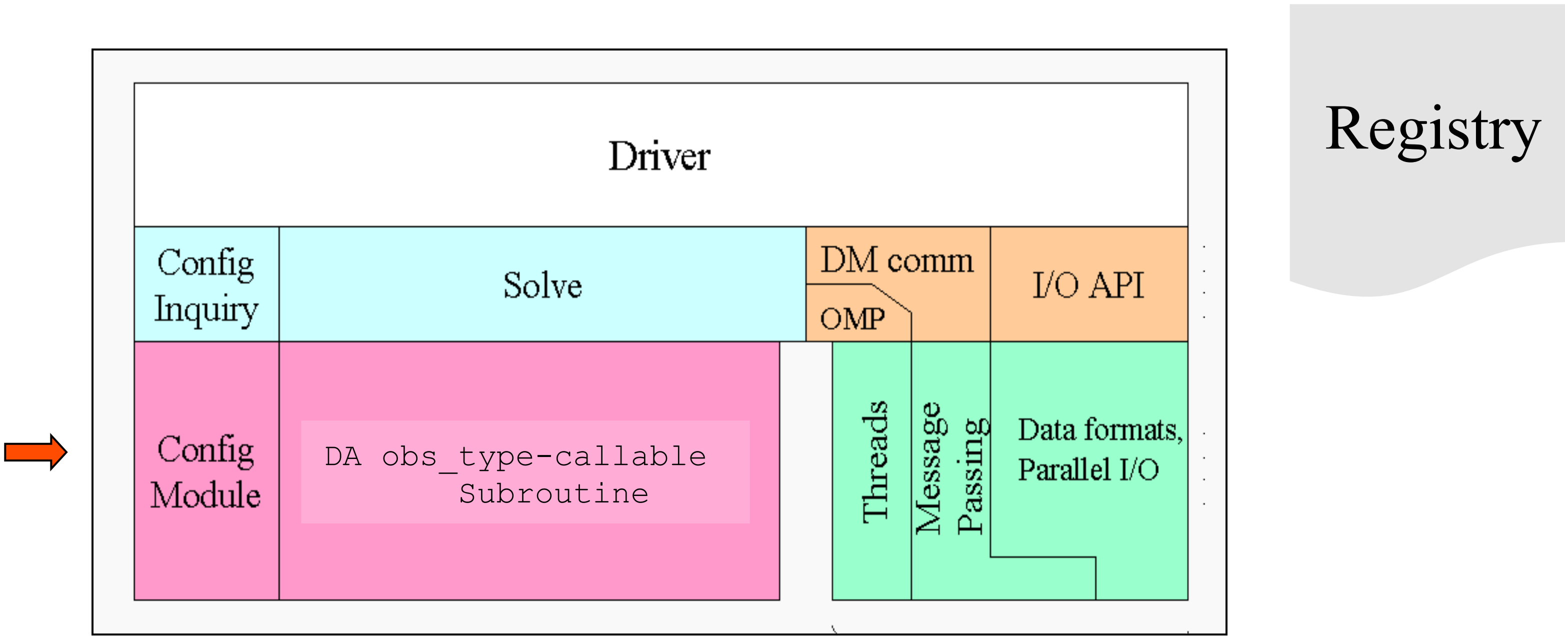
- Driver Layer
  - **Domains**: Allocates, stores, decomposes, represents abstractly as single data objects

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.

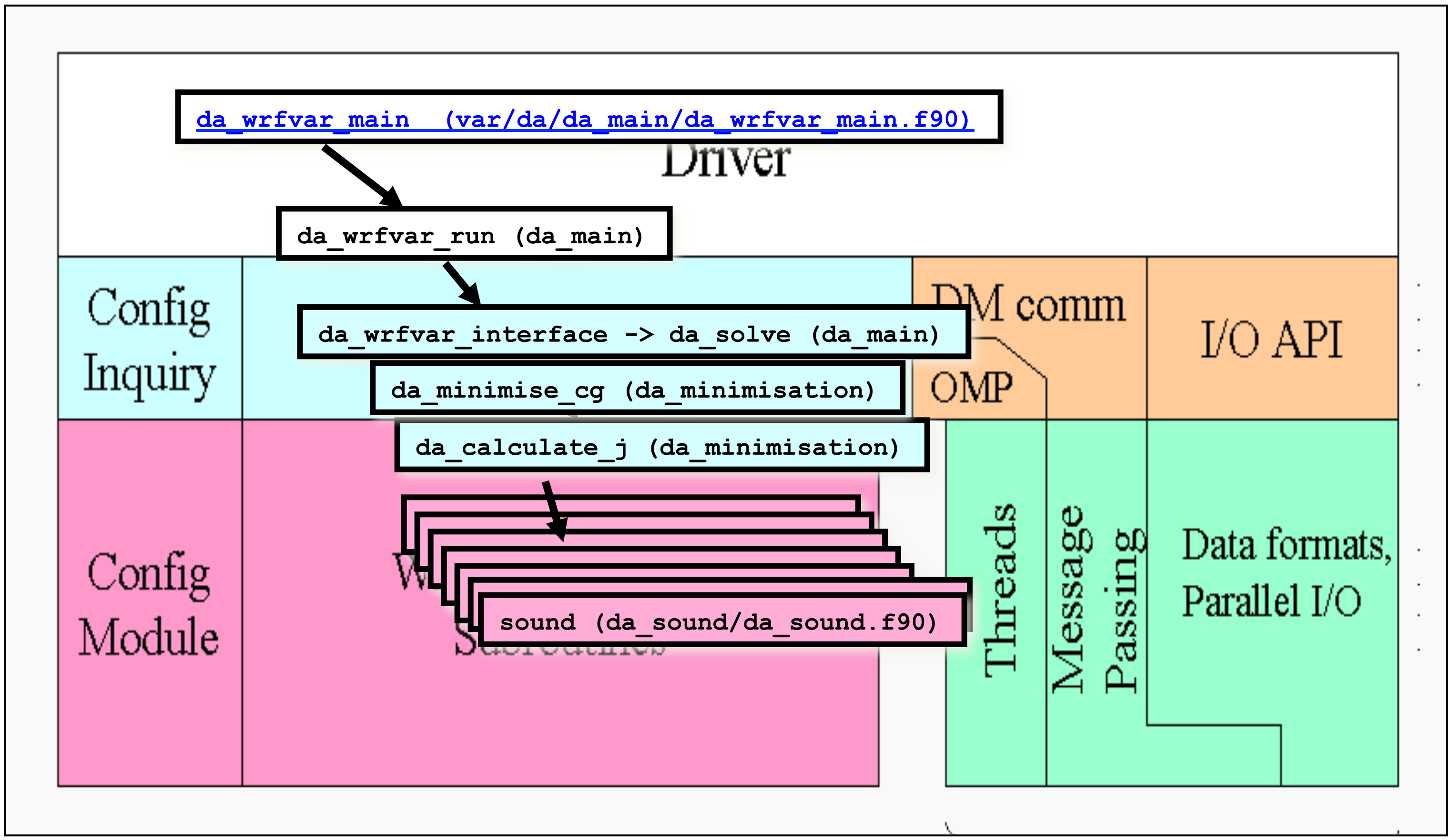
WRFDA Software – Architecture



- 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)



WRFDA Software Overview

- Introduction
- WRFDA Overview
- Computing Overview
- WRFDA Software Overview
  - Architecture
  - *Data Structures*
  - I/O

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 Software Overview

- Introduction
- WRFDA Overview
- Computing Overview
- WRFDA Software Overview
  - Architecture
  - Data Structures
  - *I/O*

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

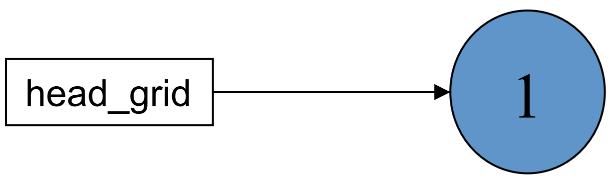
- WRFDA is designed to be an easy-to-use data assimilation system for use with the WRF model
- WRFDA is designed within the WRF Software Framework for rapid development and ease of modification
- WRFDA can be run in parallel for quick assimilation of large amounts of data

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, l1, 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.

Appendix – Derived Data Structures

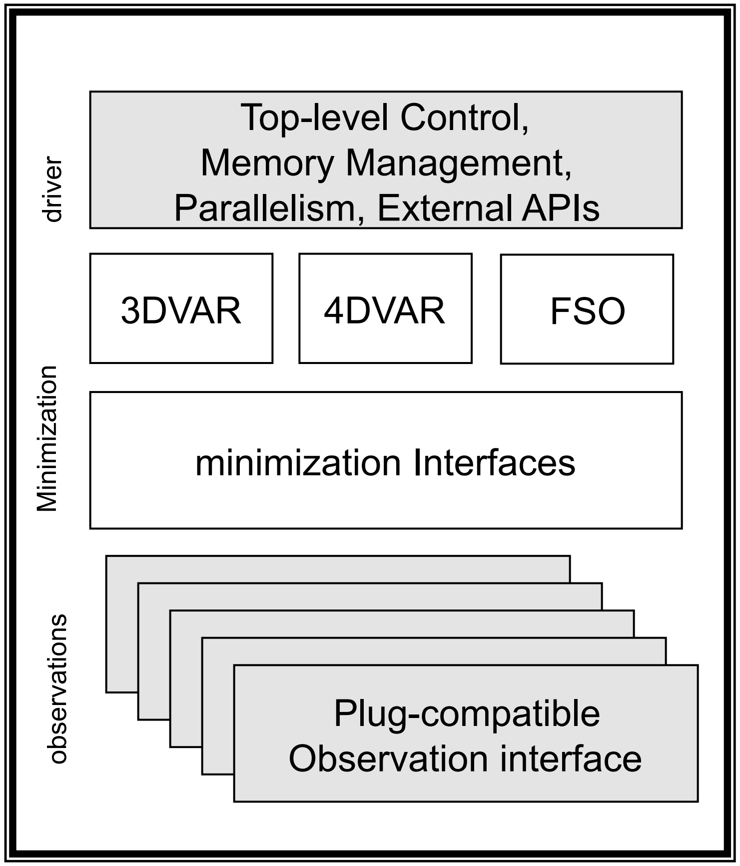
- be
  - It is used to store the background error covariance.

Appendix – Derived Data Structures

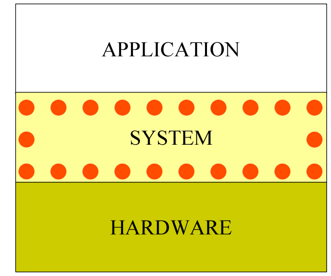
- iv
  - Stores the innovations for each observational type $\mathbf{y} - \mathbf{H}\mathbf{x}$
- ob
  - Stores the observations $\mathbf{y}$
- re
  - Store the residual $\mathbf{y} - \mathbf{H}(\mathbf{x} + \Delta\mathbf{x})$

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

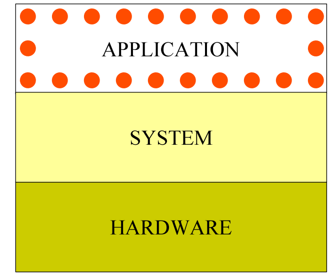


Appendix – More parallel computing terms



- 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 – More parallel computing in WRFDA



- 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 – 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

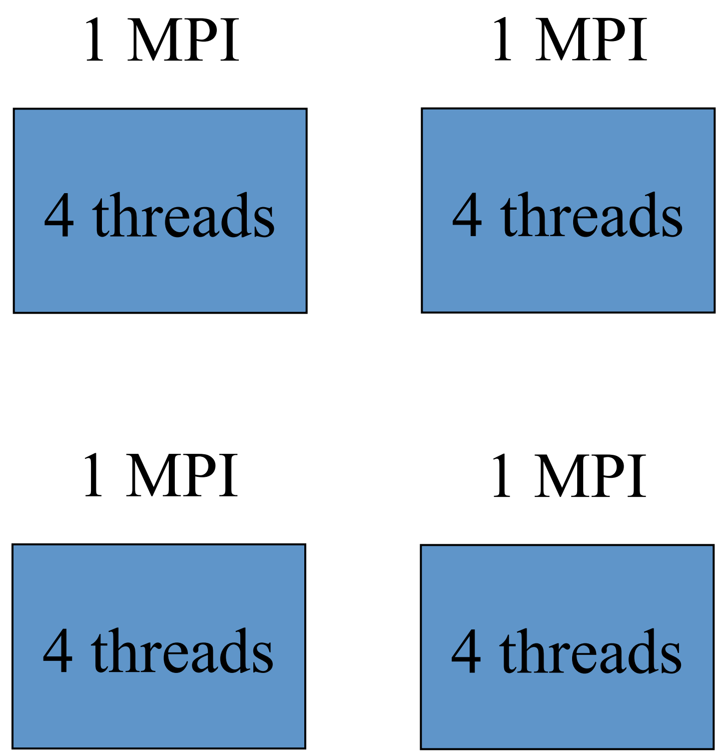
    ```
    setenv OMP_NUM_THREADS 4
    mpirun -np 4 da_wrfvar.exe
    ```

  - 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

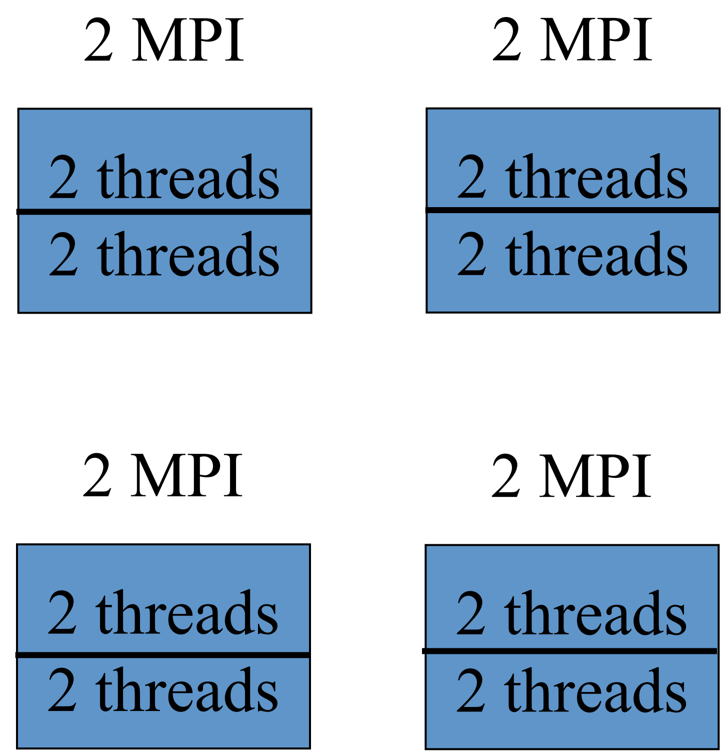
```
setenv OMP_NUM_THREADS 4
mpirun -np 4 da_wrfvar.exe
```

  - 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

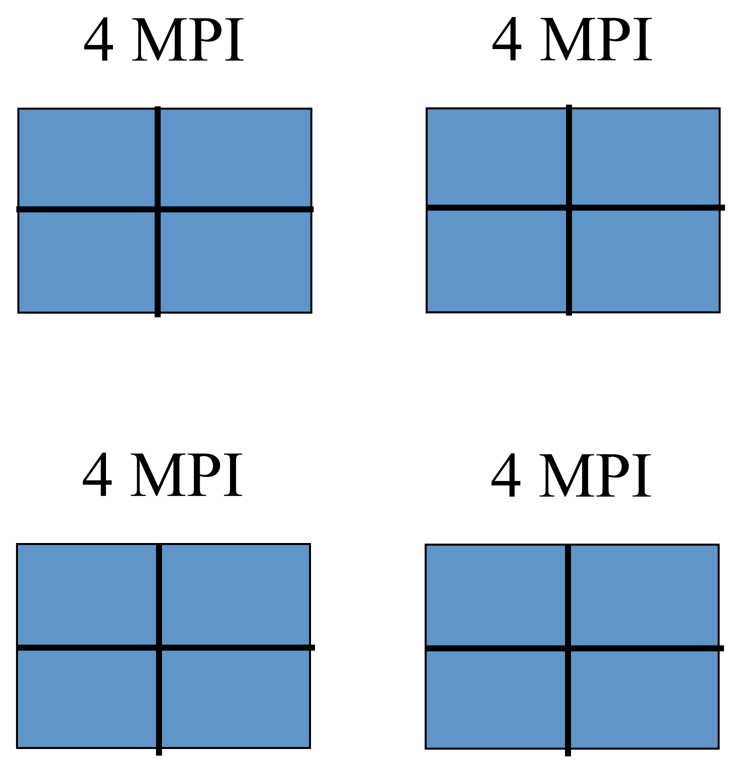
```
setenv OMP_NUM_THREADS 4
mpirun -np 4 da_wrfvar.exe
```

  – 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
```

WRFDA Top-Level Directory Structure

```
Makefile
README
README_test_cases
clean
compile
configure
Registry/
  Registry.wrfvar
arch/
dyn_em/
dyn_nnm/
external/
frame/
inc/
main/
phys/
share/
tools/
var/
run/
test/
```

build scripts (clean, compile, configure)

WRFDA Registry (Registry/, Registry.wrfvar)

DA source code directory (var/)

WRFDA Directory Structure

```
Makefile
README.basics
README.namelist
README.radiance
build/
da/
external/
gen_be/
obsproc/
run/
test/
```

build/ — building directory

da/, external/, gen_be/, obsproc/ — source code directories

run/, test/ — execution directories

WRFDA Directory Structure

- da_main/
- da_minimization/
- da_pilot/
- da_qscat/
- da_radar/
- da_radiance/
- da_metar/
- da_ships/
- da_synop/
- da_gpsref/
- da_metar/
- …

source code directories

DRIVER

MINIMIZATION

OBS