Tutorial Notes: WRF-VAR Software 2.1

Wei Huang （黄伟）

July 28, 2005

Many are copied/modified from John Michalakes' 2004 lecture

Outline

- Introduction
- Software Overview
- Data Structures
- Registry
- Example

Introduction

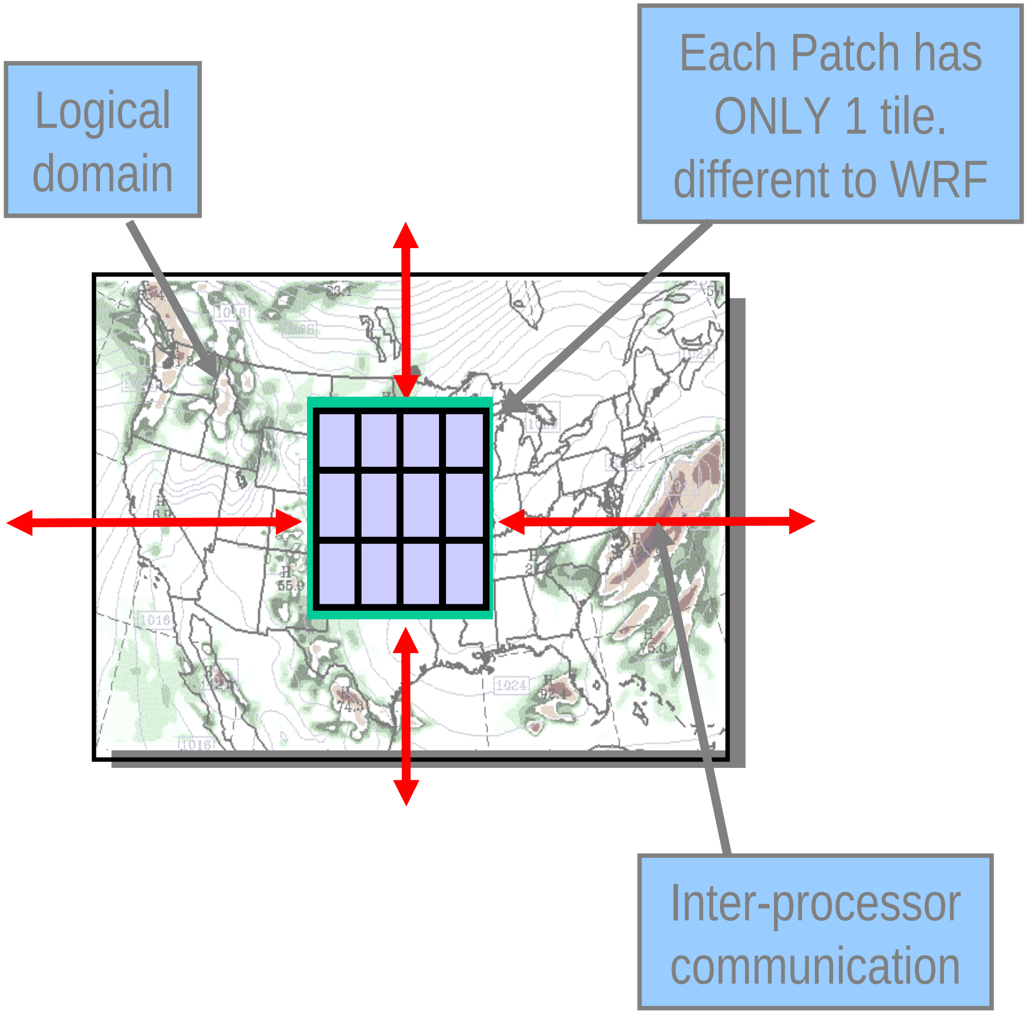
- Intended audience for this tutorial session:
  - Primarily scientific users and others who wish to:
    - Work with the code
    - Extend/modify the code to enable their work/research
    - Address problems as they arise
    - Adapt the code to take advantage of local computing resources
  - Also: developers, computer scientists and software engineers, computer vendors
    - Developing new functionality (e.g. new observations, new minimization package)
    - Porting and benchmarking new platforms

Introduction

- Supported Platforms
   - ➢ IBM (AIX)
   - ➢ HP (OSF1)
   - ➢ MAC (OS X)
   - ➢ PC (Linux)
   - ➢ SGI (IRIX)

Parallelism in WRF-VAR: MPI Decomposition

- Single version of code for efficient execution on:
  - Distributed-memory
  - Vector and microprocessors



Logical domain

Each Patch has ONLY 1 tile. different to WRF

Inter-processor communication

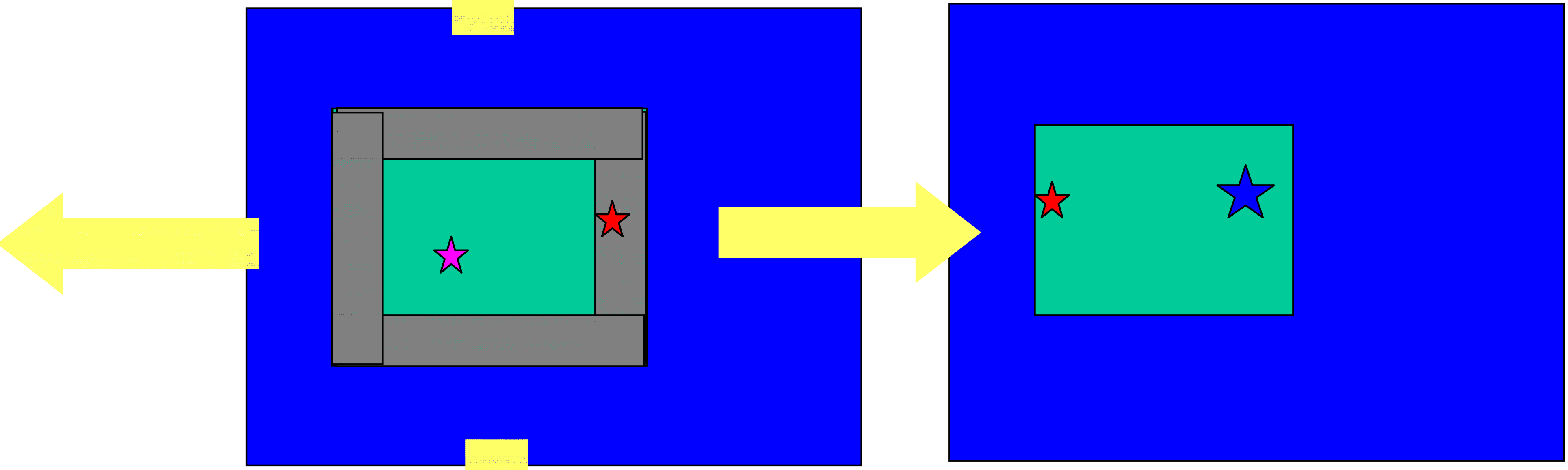
Model domain is decomposed for parallelism

Patch: section of model domain allocated to a distributed memory node

Tile: same as patch in wrf-var

Observation in Distributed Memory

- Halo Region Observation

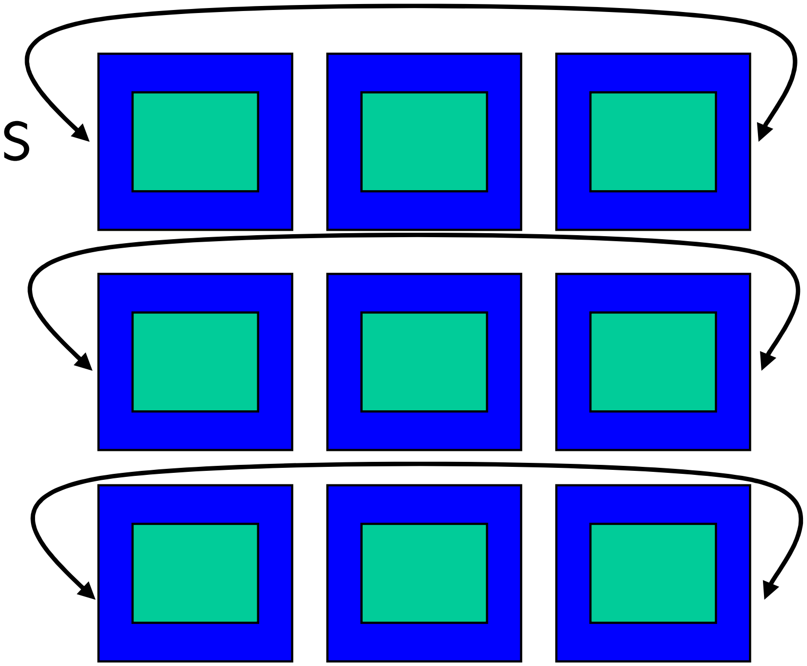


Obs. on one processor's halo

Obs. on neighboring processor

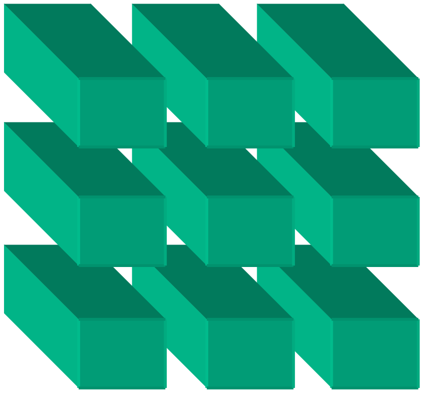
Distributed Memory Communications

- Halo updates
- Periodic boundary updates (only needed for global 3dvar)

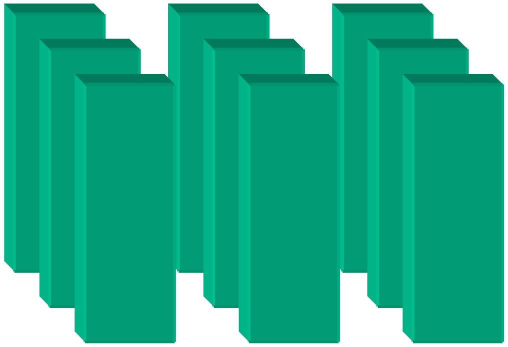


Distributed Memory Communications

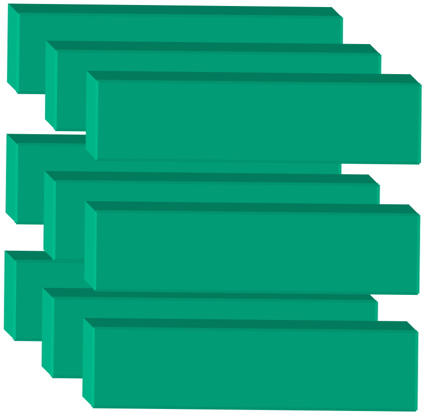
- Halo updates
- Periodic boundary updates
- Parallel transposes



all y on
patch

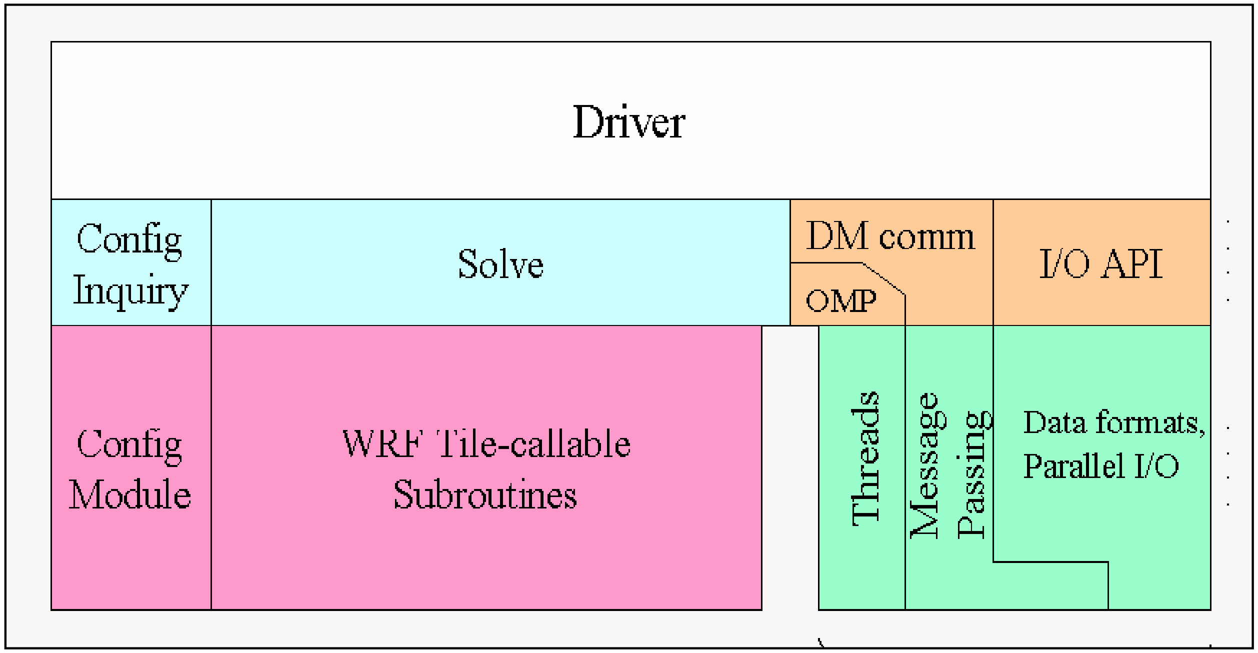


all z on
patch



all x on
patch

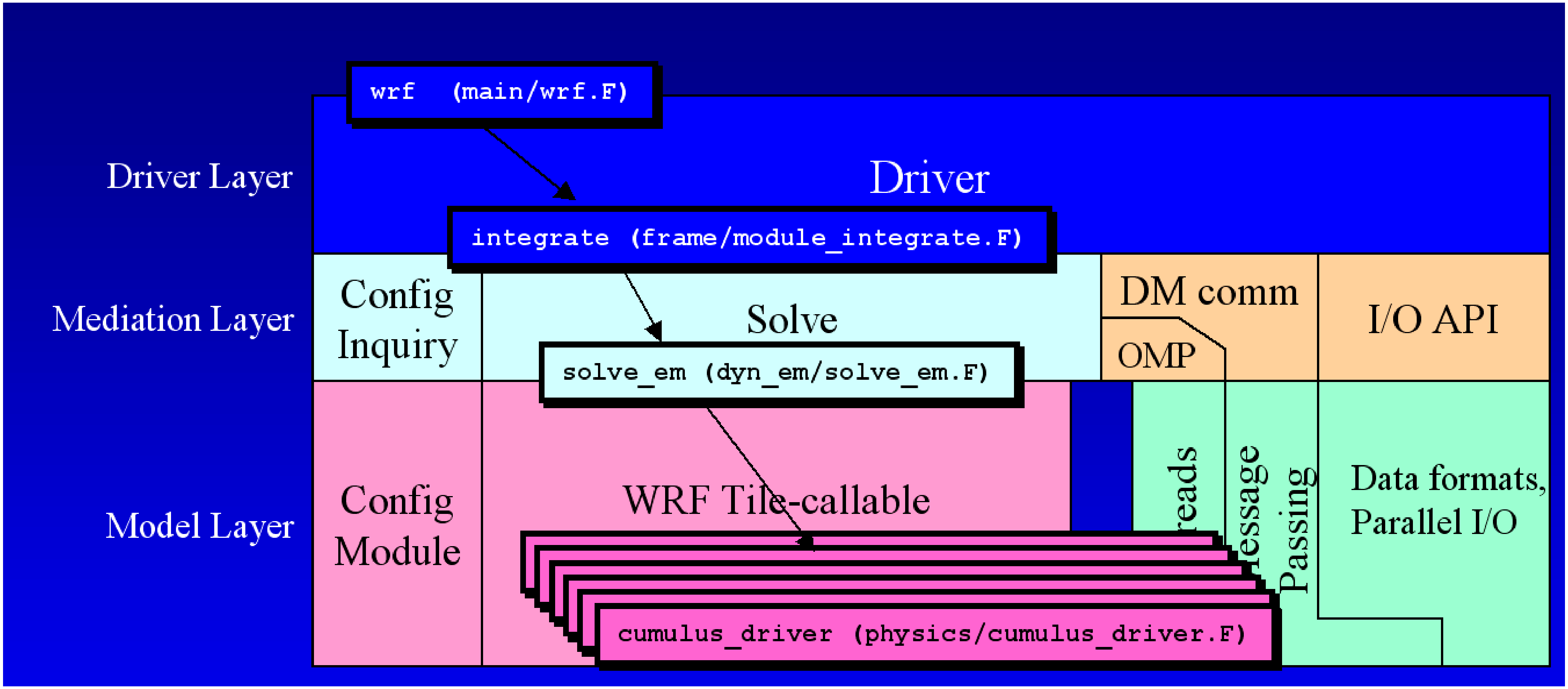
WRF-VAR Software Architecture



- 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, and model coupling facilitates code reuse and exploiting of community infrastructure, e.g. ESMF.

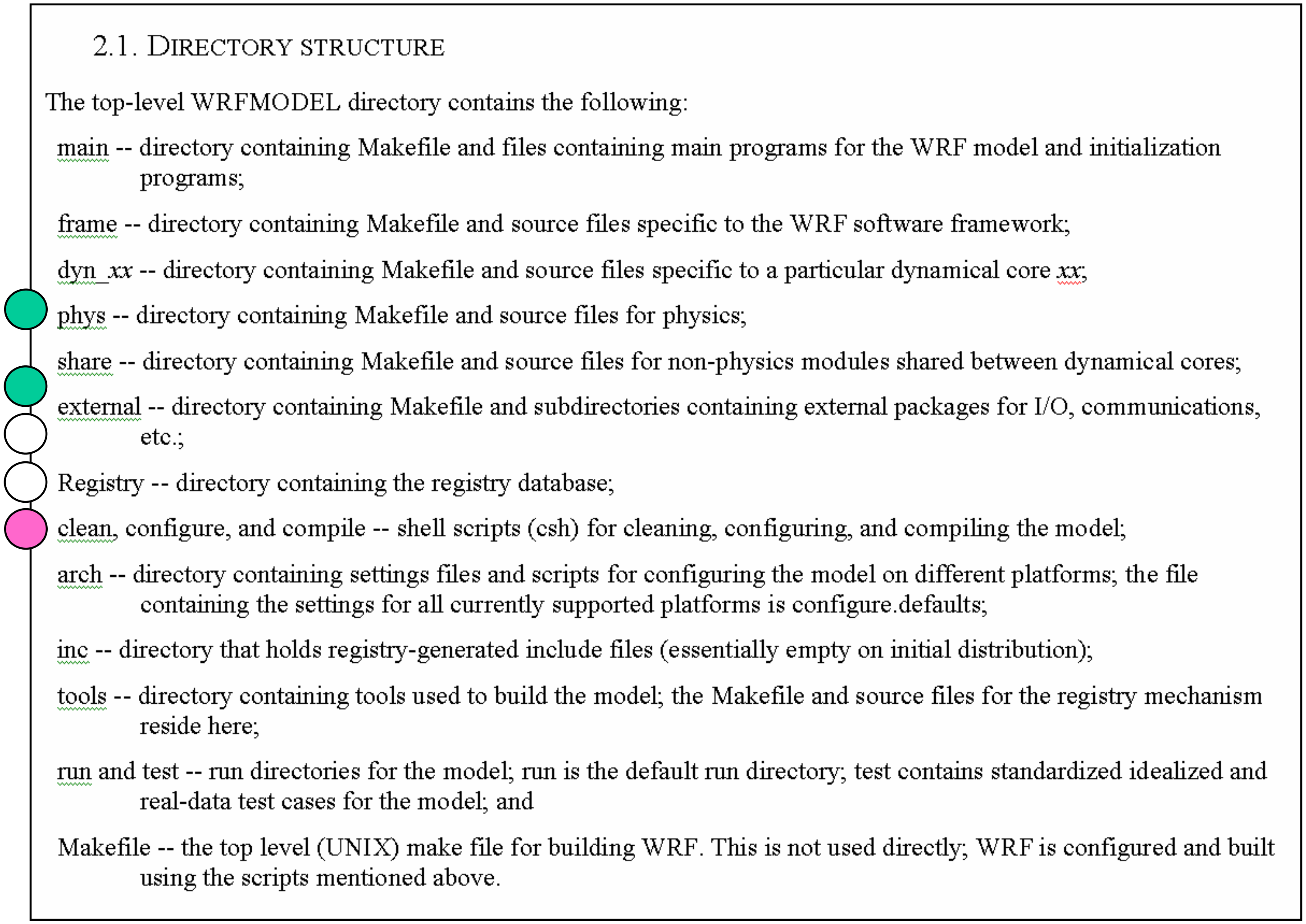
- Borrowed from John
  - Replace WRF with WRF/WRF-VAR
  - Replace solve with solve_v3d
  - There is NO OpenMP in wrfvar

Directory Structure



Replace wrf with wrfvar
Replace integrate with da_solve_v3d_interface
Replace solve_em with da_solve_v3d
Replace cumulus_driver with obs. (ships)

WRF-VAR Directory Structure



2.1. Directory structure

The top-level WRFMODEL directory contains the following:

main -- directory containing Makefile and files containing main programs for the WRF model and initialization programs;

frame -- directory containing Makefile and source files specific to the WRF software framework;

dyn_*xx* -- directory containing Makefile and source files specific to a particular dynamical core *xx*;

🟢 phys -- directory containing Makefile and source files for physics;

share -- directory containing Makefile and source files for non-physics modules shared between dynamical cores;

🟢 external -- directory containing Makefile and subdirectories containing external packages for I/O, communications,
○ etc.;

○ Registry -- directory containing the registry database;

🟣 clean, configure, and compile -- shell scripts (csh) for cleaning, configuring, and compiling the model;

arch -- directory containing settings files and scripts for configuring the model on different platforms; the file containing the settings for all currently supported platforms is configure.defaults;

inc -- directory that holds registry-generated include files (essentially empty on initial distribution);

tools -- directory containing tools used to build the model; the Makefile and source files for the registry mechanism reside here;

run and test -- run directories for the model; run is the default run directory; test contains standardized idealized and real-data test cases for the model; and

Makefile -- the top level (UNIX) make file for building WRF. This is not used directly; WRF is configured and built using the scripts mentioned above.

🟢 driver
🟣 mediation
○ model

Add da_3dvar directory
No physics package needed in wrf-var (yet)

Data Structures

- Data Taxonomy
- How data appears at different levels of architecture
- Grid representation in WRF-VAR arrays
- Observations

Data Structures

- WRF-VAR Data Taxonomy
  - State data
  - Intermediate data type 1 (I1)
  - Intermediate data type 2 (I2)
  - Heap storage (COMMON or Module data)

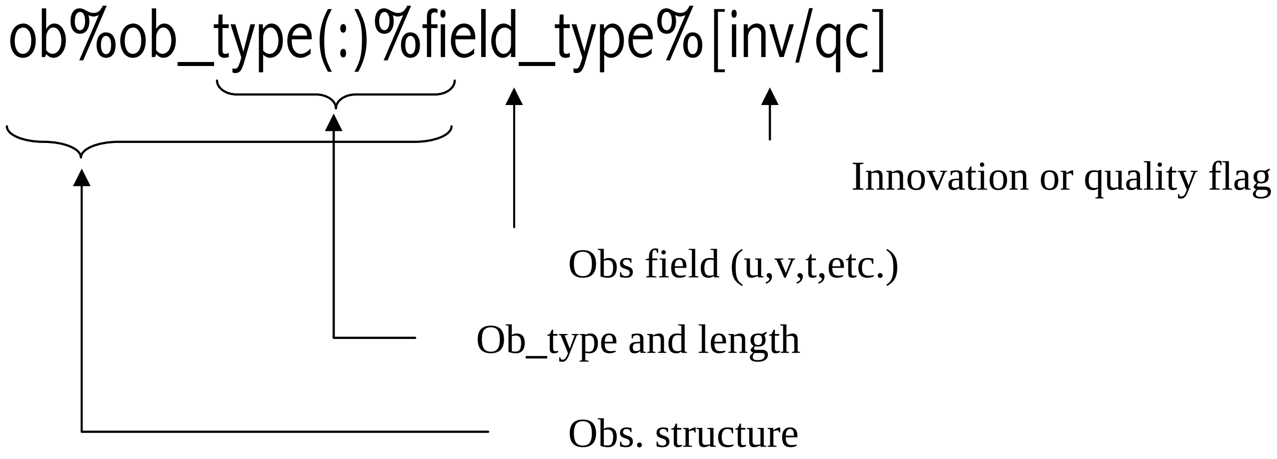
  - All WRF data are used in WRF-VAR (Because of Frame, I/O API)

State Data

- Persist for the duration of a domain
- Represented as fields in domain data structure
- Arrays are represented as dynamically allocated pointer arrays in the domain data structure
- Declared in Registry using **state** keyword
- Always **memory** dimensioned; always **thread shared**
- Only state arrays can be subject to I/O and Interprocessor communication

WRF-VAR Observations

- May be single level or multiple levels
- Have defined type of: ob, iv, re, and y. Ob looks like in code:



ob%ob_type(:)%field_type%[inv/qc]

Innovation or quality flag

Obs field (u,v,t,etc.)

Ob_type and length

Obs. structure

Example

Radiosonde observation appears as:

```
ob%sound(n)%u(lvl)%inc
ob%sound(n)%v(lvl)%qc
```

Radiosonde residual appears as:

```
re%sound(n)%u(lvl)
re%sound(n)%v(lvl)
```

Observation Storage

- Observation is stored in heap
  - Completely self-contained and private
  - Set once (Read in from disk file)
  - No exchange between processors/processes

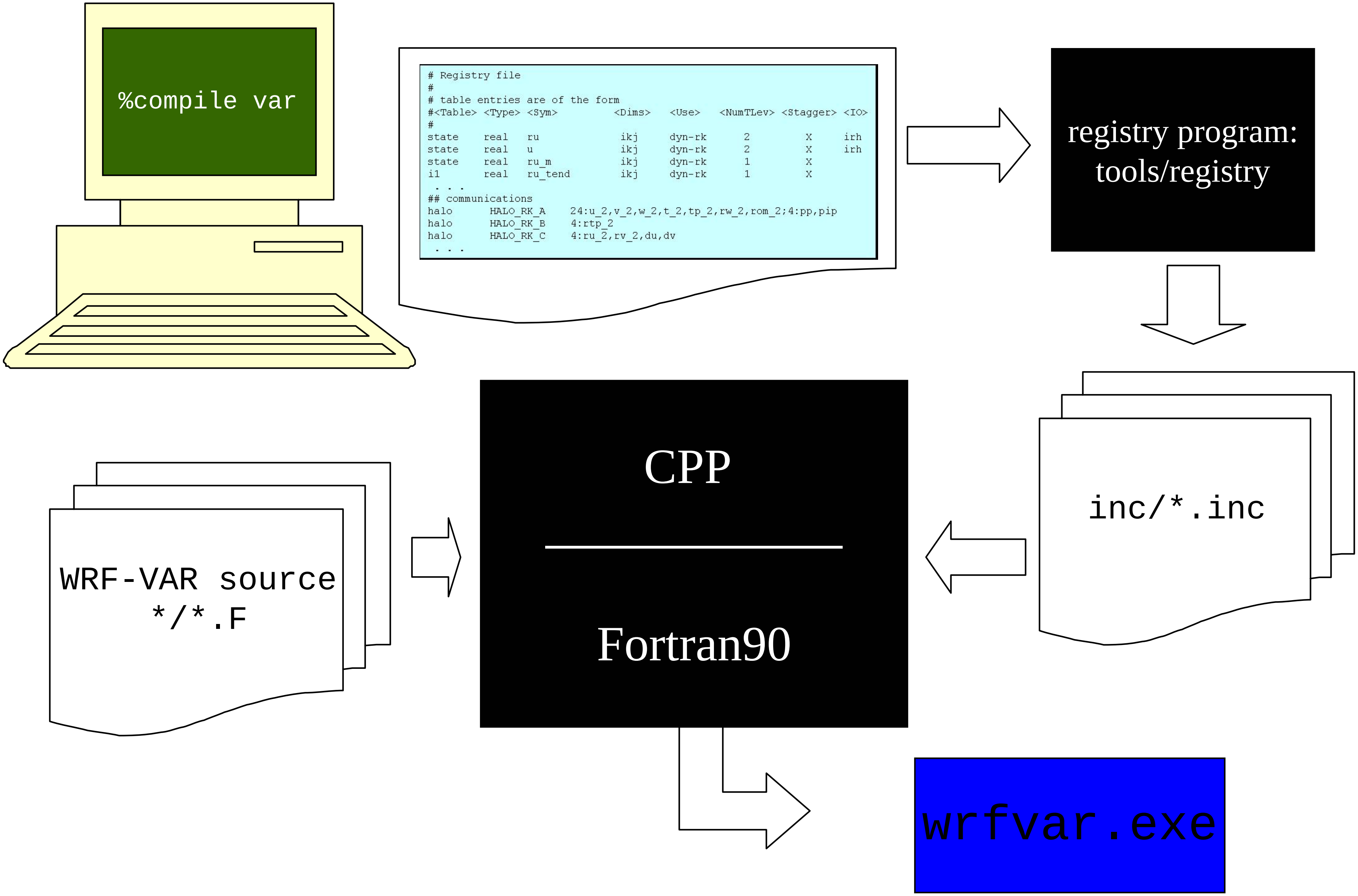
Grid Representation in Arrays

- Increasing indices in WRF-VAR 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 WRF-VAR is IJK, but this is a WRF -VAR convention, not a restriction of the WRF Software Framework
- WRF-VAR grid data are all converted to mass-grid point

WRF-VAR Registry

- "Active data-dictionary" for managing WRF-VAR data structures
  - Database describing attributes of model state, intermediate, and configuration data
    - Dimensionality, number of time levels, staggering
    - Association with physics
    - I/O classification (history, initial, restart, boundary)
    - Communication points and patterns
    - Configuration lists (e.g. namelists)
  - Program for auto-generating sections of WRF from database:
    - Allocation statements for state data, I1 data
    - Argument lists for driver layer/mediation layer interfaces
    - Interprocessor communications: Halo and periodic boundary updates, transposes
    - Code for defining and managing run-time configuration information
    - Code for forcing, feedback and interpolation of nest data
- Automates time consuming, repetitive, error-prone programming
- Insulates programmers and code from package dependencies
- Allow rapid development
- Documents the data

Registry Mechanics



%compile var

```
# Registry file
#
# table entries are of the form
#<Table> <Type> <Sym>          <Dims>   <Use>   <NumTLev> <Stagger> <IO>
#
state    real   ru              ikj     dyn-rk      2         X     irh
state    real   u               ikj     dyn-rk      2         X     irh
state    real   ru_m            ikj     dyn-rk      1         X
i1       real   ru_tend         ikj     dyn-rk      1         X
 . . .
## communications
halo      HALO_RK_A    24:u_2,v_2,w_2,t_2,tp_2,rw_2,rom_2;4:pp,pip
halo      HALO_RK_B    4:rtp_2
halo      HALO_RK_C    4:ru_2,rv_2,du,dv
 . . .
```

registry program:
tools/registry

inc/*.inc

CPP

Fortran90

WRF-VAR source
/.F

wrfvar.exe

Registry Data Base

- Currently implemented as a text file: Registry/Registry
- Types of entry:
  - *State* – Describes state variables and arrays in the domain structure
  - *Dimspec* – Describes dimensions that are used to define arrays in the model
  - *I1* – Describes local variables and arrays in solve
  - *Typedef* – Describes derived types that are subtypes of the domain structure
  - *Rconfig* – Describes a configuration (e.g. namelist) variable or array
  - *Package* – Describes attributes of a package (e.g. physics)
  - *Halo* – Describes halo update interprocessor communications
  - *Period* – Describes communications for periodic boundary updates
  - *Xpose* – Describes communications for parallel matrix transposes

State entry

- Elements
  - *Entry*: The keyword “state”
  - *Type*: The type of the state variable or array (real, double, integer, logical, character, or derived)
  - *Sym*: The symbolic name of the variable or array
  - *Dims*: A string denoting the dimensionality of the array or a hyphen (-)
  - *Use*: A string denoting association with a solver or 4D scalar array, or a hyphen
  - *NumTLev*: An integer indicating the number of time levels (for arrays) or hypen (for variables)
  - *Stagger*: String indicating staggered dimensions of variable (X, Y, Z, or hyphen)
  - *IO*: String indicating whether and how the variable is subject to I/O and Nesting
  - *DName*: Metadata name for the variable
  - *Units*: Metadata units of the variable
  - *Descrip*: Metadata description of the variable
- Example

```
#       Type Sym  Dims   Use      Tlev Stag IO    Dname
Descrip
# definition of a 3D, two-time level, staggered state array

state   real u   ijk    dyn_em   2   X    irh   "U"  "X WIND
COMPONENT“
…
typedef xb_type real   u   ijk        -              1     -        -
…
state xb_type xb - -
```

Dimspec entry

- Elements
  - *Entry*: The keyword “dimspec”
  - *DimName*: The name of the dimension (single character)
  - *Order*: The order of the dimension in the WRF framework (1, 2, 3, or ‘-‘)
  - *HowDefined*: specification of how the range of the dimension is defined
  - *CoordAxis*: which axis the dimension corresponds to, if any (X, Y, Z, or C)
  - *DatName*: metadata name of dimension
- Example

```
#<Table>  <Dim>  <Order> <How defined>        <Coord-axis>  <DatName>
dimspec    i      1     standard_domain            x          west_east
dimspec    j      2     standard_domain            y          south_north
dimspec    k      3     standard_domain            z          bottom_top
dimspec    l      3     namelist=num_soil_layers   z          soil_layers
```

Comm entries: halo

- Elements
  - *Entry*: keywords “halo” or “period”
  - *Commname*: name of comm operation
  - *Description*: defines the halo or period operation
    - For halo: *npts:f1,f2,…[;npts:f1,f2,…]**
    - For period: *width:f1,f2,…[;width:f1,f2,…]**
- Example

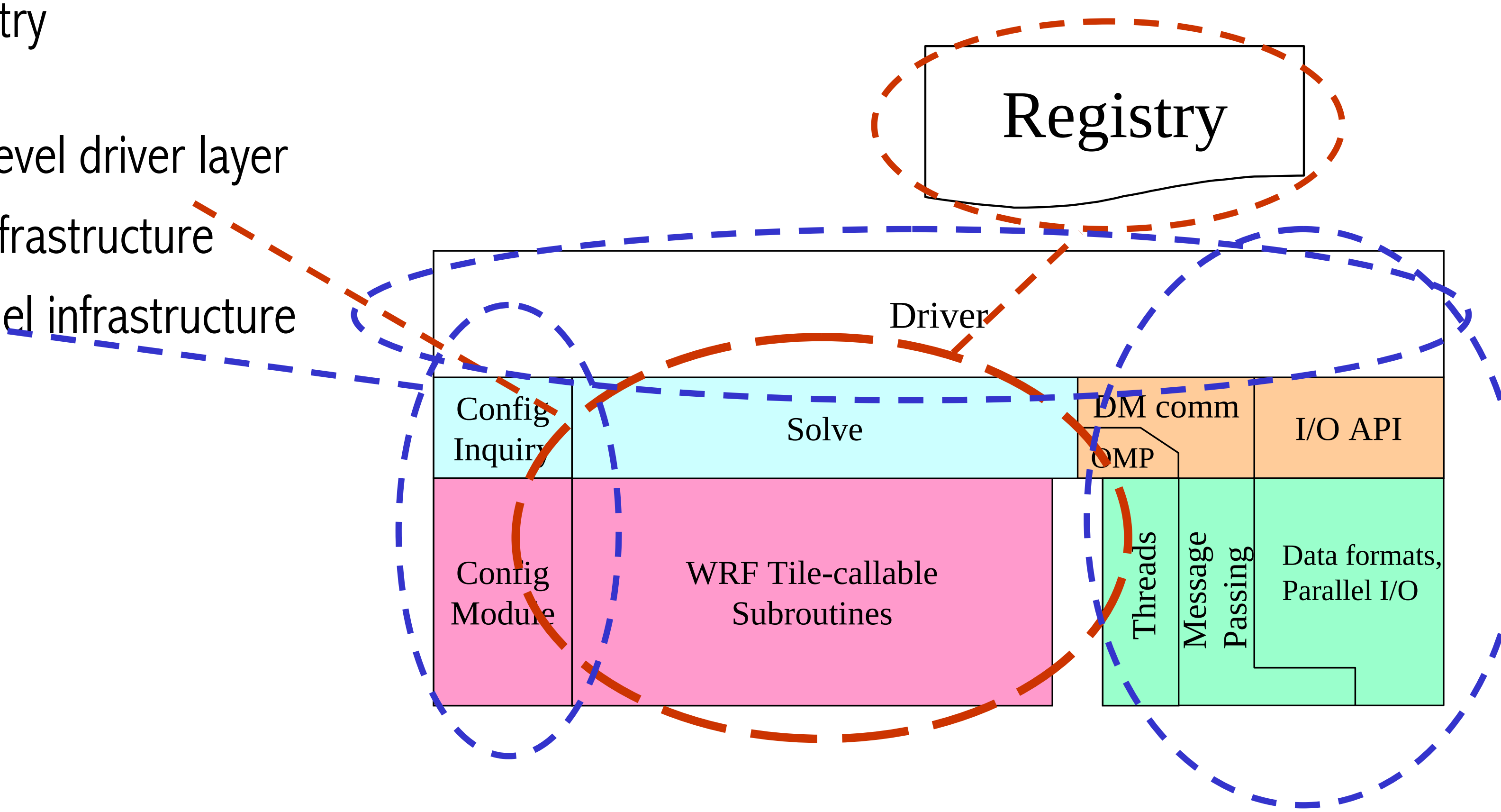
```
halo    HALO_XA    dyn_em 24:xa%u,xa%v,xa%q,xa%p,xa%t,xa%rho,xa%rh,xa%psfc,xa%qcw,xa%qrn,xa%qt
halo    HALO_XB    dyn_em 24:xb%u,xb%v,xb%w,xb%wh,xb%q,xb%p,xb%t,xb%rho,xb%rh,xb%psfc,xb%slp
```

I/O

- Use WRF I/O API

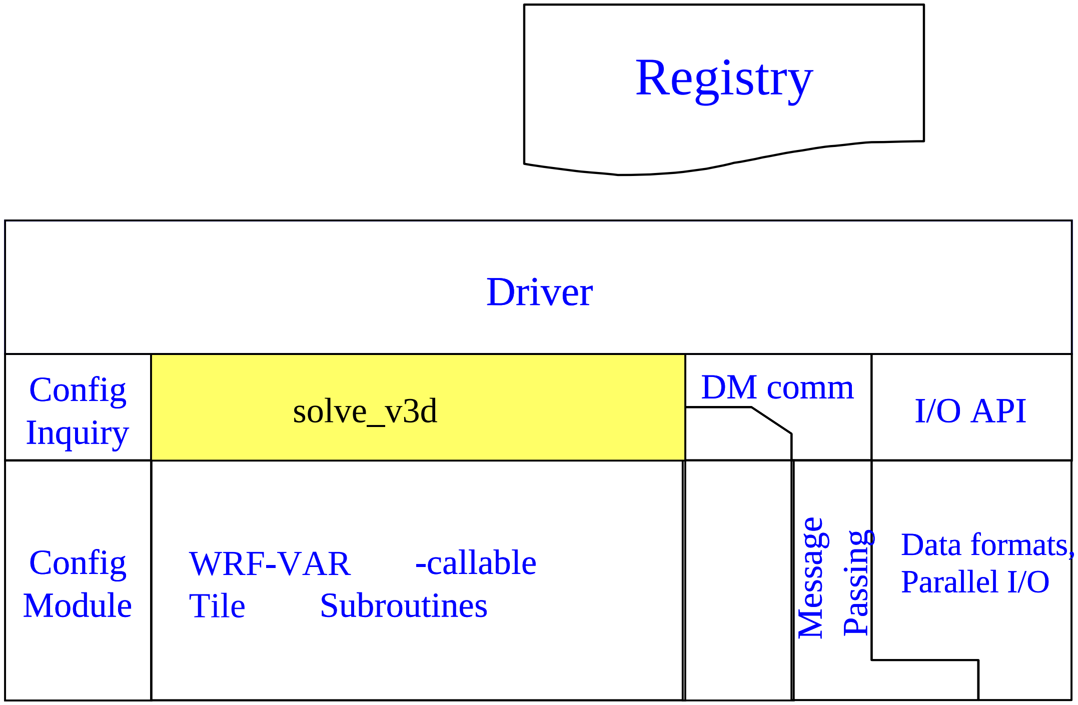
Adding WRF-VAR to WRF Frame

- Conceptual
  - WRF framework can slot in new dynamics as run-time selectable option
    - Changes to:
      - Mediation layer, model layer
      - Registry
    - Reuse:
      - Top-level driver layer
      - I/O infrastructure
      - Parallel infrastructure



Adding WRF-VAR

- Steps
  - Develop new or convert existing code:
    - Mediation layer routine: solve
    - Model layer subroutines called by solver
  - Add to WRF
    - Add code to source tree
    - Incorporate into build mechanism
    - Registry entries: data, solver options, comms
    - Some additional splicing
  - Single processor testing
  - Analyze data-dependencies, define and implement communication for parallelism
  - Multi-processor testing



Add new Observation

- Edit DA_Define_Structure.F to add new type
- May need add grid array in Registry
- Make a new obs directory
- Input observation
- Link into minimization package