

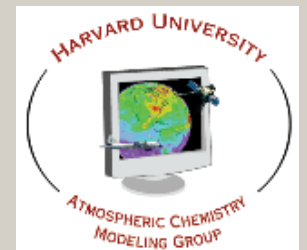
# GEOS–Chem Column Code Development

**Bob Yantosca**

Software Engineer

Atmospheric Chemistry Modeling Group  
School of Engineering & Applied Sciences  
Harvard University

Jacob Group Meeting  
Wednesday, January 13, 2010



# Table of Contents

## 1) Brief History of GEOS–Chem

- a) How existing GEOS–Chem structure came to be
- b) Limitations of existing GEOS–Chem

## 2) Motivation for Column Code

- a) From NASA
- b) From the GEOS–Chem community

## 3) “New World Order”

- a) Updated coding standards & rules for “columnizing” code

## 4) Progress Report

- a) Validation
- b) Status

# 1. Brief History of GEOS–Chem

GEOS–Chem was originally built from 2 separate models

## Harvard–GISS CTM

(Y. Wang, L. Horowitz, et al @ Harvard)  
4x5, 9-layer, driven w/ offline GISS met

Met fields: GISS 9-layer GCM

Advection: GISS SOM (M. Prather)

Cld Conv: GISS algorithm

PBL Mixing: GISS algorithm

Wet Dep: GISS algorithm

Dry Dep: DEPVEL (Harvard)

Emissions: cf. Wang et al 1998

Photolysis: SLOW–J (Harvard)

Chemistry: Nox-Ox-HC w/  
SMVGEAR I (Harvard)

## GEOS–CTM

(M. Chin @ GSFC)

This was the precursor to GOCART

Met Fields: GMAO GEOS–1

Advection: TPCORE (S-J Lin)

Cld Conv: CLDMX (S-J Lin)

PBL Mixing: TURBDAY (D. Allen/UMD)

Wet Dep: none

Dry Dep: none

Emissions: read from disk

Photolysis: none

Chemistry: offline CO or DMS

# 1. Brief History of GEOS–Chem

## Harvard–GISS CTM

Dry Dep: DEPVEL

Emissions: cf. Wang et al 1998

Photolysis: SLOW–J

Chemistry: Nox-Ox-HC; SMVGEAR I

## GEOS–CTM

Met Fields: GMAO GEOS–1

Advection: TPCORE

Cld Conv: CLDMX

PBL Mixing: TURBDAY

## GEOS–Chem

Met Fields: GMAO GEOS–1

Advection: TPCORE

Cld Conv: CLDMX

PBL Mixing: TURBDAY

Dry Dep: DEPVEL

Emissions: cf. Wang et al 1998

Photolysis: SLOW–J

Chemistry: Nox-Ox-HC; SMVGEAR I

Dynamics  
routines  
came from  
the GEOS-  
CTM

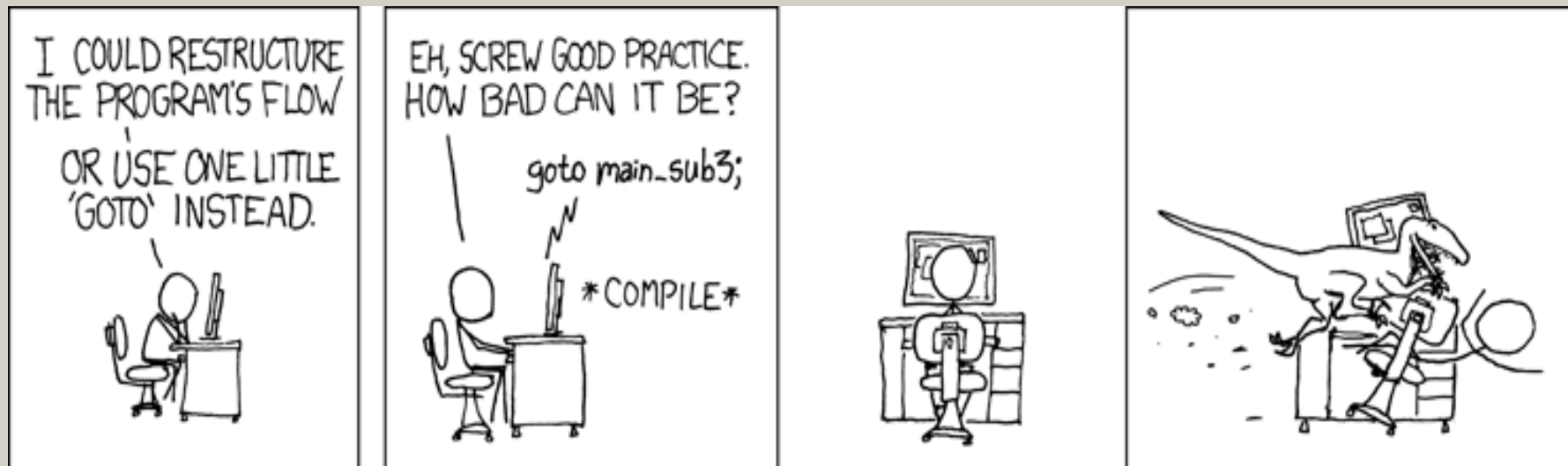
Emissions, drydep,  
chemistry routines  
came from the  
Harvard-GISS CTM

# 1. Brief History of GEOS–Chem

- Over time, various updates were made to GEOS–Chem
  - Advection: updated TPCORE versions for global & nested grids
  - Aerosols: originally from GOCART (M. Chin)
  - ATE: ISORROPIA and RPMARES
  - Met fields: GEOS–1 thru GEOS–5
  - Chemistry: SMVGEAR II, KPP, new mechanisms
  - Photolysis: SLOW–J replaced by FAST–J
  - SOA: originally from Caltech group
  - Many offline simulations (CO, Hg, Ox, CH<sub>4</sub>, etc...)
- Many of these updates were 3rd-party codes
- ***The structure of GEOS–Chem is in many places dictated by the structure of the 3rd-party codes***

# 1. Brief History of GEOS–Chem

- GEOS–Chem still contains a lot of “legacy code”
  - Many routines use obsolete F77-style coding constructs
    - Common blocks
    - GOTO's etc.
  - Why old coding constructs are not advisable anymore
    - GOTO's prevent compilers from optimizing efficiently
    - Common blocks w/ global data arrays impede MPI parallelization



# 1. Brief History of GEOS–Chem

## *OpenMP Parallelization*

- This is what is used in GEOS–Chem now
- Pros:
  - Simple to use,
  - Built into many compilers
- Cons:
  - Limited to shared-memory machines
  - Does not scale as well past about 16 CPU's

## *MPI Parallelization*

- This is what we want to use in the near future
- Pros:
  - Can run on clusters w/ 100's or 1000's of CPU's
  - GEOS–5 GCM uses MPI
- Cons:
  - Harder to implement
  - Legacy code may prohibit effective use of MPI

# 1. Brief History of GEOS–Chem

- GEOS–Chem routines work on global-sized arrays
  - 2-D, 3-D, and 4-D arrays are passed between routines
    - Emissions: (lon, lat) or (lon, lat, level)
    - Met fields: (lon, lat, level)
    - Tracers, chemical species, rxn rates: (lon, lat, level, number)

| Grid              | # of Lons | # of Lats | # of Levels | # of Tracers | Size of Tracer Array |
|-------------------|-----------|-----------|-------------|--------------|----------------------|
| GEOS-5<br>4 x 5   | 72        | 46        | 47          | 54           | 67 MB                |
| GEOS-5<br>2 x 2.5 | 144       | 91        | 47          | 54           | 266 MB               |
| GEOS-5<br>1 x 1   | 360       | 180       | 47          | 54           | 1.3 GB               |
| GEOS-5<br>½ x 2/3 | 540       | 360       | 47          | 54           | ~ 4 GB               |
| GEOS-5<br>¼ x ¼   | 1440      | 720       | 47          | 54           | ~ 21 GB              |

The table at right shows the size of the tracer array for various GEOS–5 global grids.

Assuming 47 layers (collapsed down from 72 layers) and a REAL\*8 array.

# 1. Brief History of GEOS–Chem

Advection [  $\mathbf{A}(:, :, :)$ ,  $\mathbf{Q}(:, :, :, :)$  ]

PBL Mixing [  $\mathbf{A}(:, :, :)$ ,  $\mathbf{Q}(:, :, :, :)$  ]

Cloud Conv. [  $\mathbf{A}(:, :, :)$ ,  $\mathbf{Q}(:, :, :, :)$  ]

Dry Dep [  $\mathbf{A}(:, :, :)$ ,  $\mathbf{Q}(:, :, :, :)$  ]

Emissions [  $\mathbf{A}(:, :, :)$ ,  $\mathbf{Q}(:, :, :, :)$  ]

Chemistry [  $\mathbf{A}(:, :, :)$ ,  $\mathbf{Q}(:, :, :, :)$  ]

Wet Dep [  $\mathbf{A}(:, :, :)$ ,  $\mathbf{Q}(:, :, :, :)$  ]

## Schematic of current “standard” GEOS–Chem

Each of the GEOS–Chem operators work on global-sized arrays.

In this example:

$\mathbf{A}(:, :, :)$  represent 3-D global arrays such as met fields, emissions, These arrays have dimensions of (lon, lat, level).

$\mathbf{Q}(:, :, :, :)$  represent 4-D global arrays, such as tracers, chemical species, reaction rates, etc. These arrays have dimensions of (lon, lat, level, number).

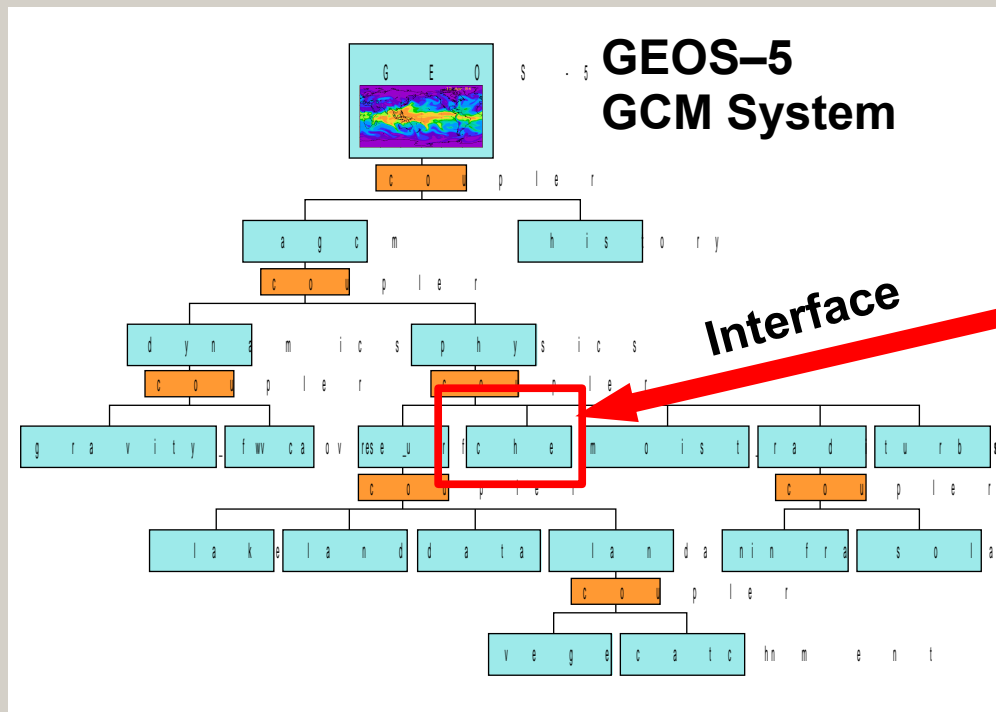
The arrow represents the program flow, managed by the main program.

# 1. Brief History of GEOS–Chem

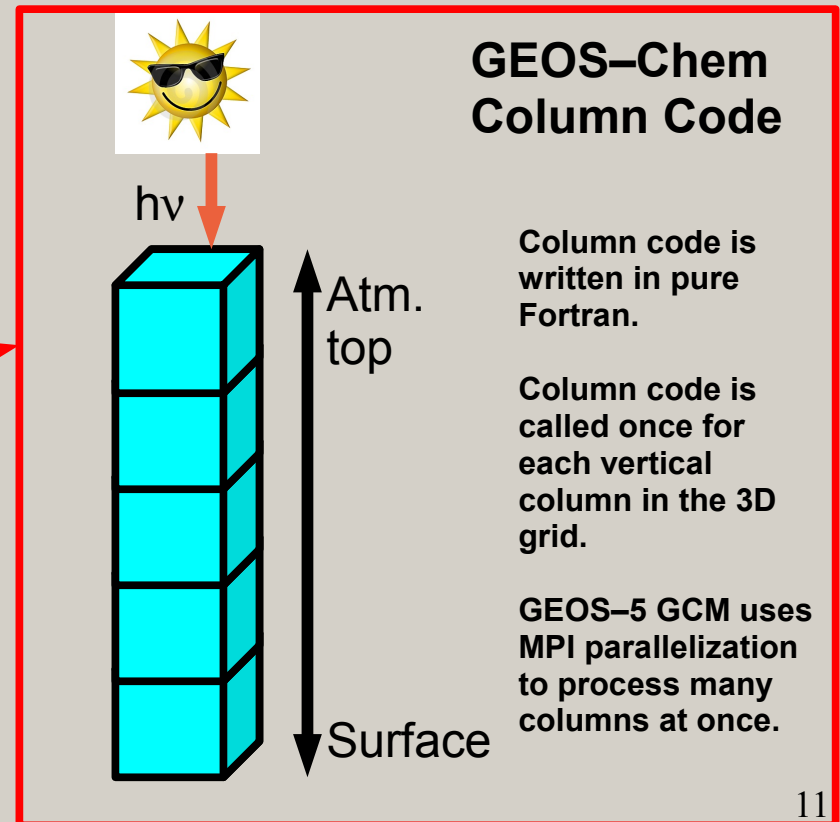
- Summary, Part 1
  - GEOS–Chem was built from 2 pre-existing code bases
    - Existing code structures were kept, then modified later
    - Significant amount of 3rd-party, legacy code still remains
  - Global arrays are passed between routines
    - This was not a problem when we were using low resolution (e.g. 4x5)
    - Array sizes get prohibitively large at high resolution
    - The same loops over (lon, lat, level, number) have to be done over and over again in various GEOS–Chem routines
- ***Some rewriting of code is required for us to use GEOS–Chem with global high-resolution grids!***

## 2. Motivation for Column Code

- GMAO requested a 1-D version of GEOS–Chem
  - Goal: to interface a “column” G–C code into GEOS–5 GCM
  - Meteorology is generated online, then passed to G–C code
  - Tracers modified by G–C, then passed back to GEOS–5



GEOS–5 passes vertical columns of grid boxes to the GEOS–Chem column code via an interface



## 2. Motivation for Column Code

- Column code also has several advantages for us (#1)
  - It's a good reason to do some “spring cleaning” of legacy code
    - Goal: to remove things that impede MPI parallelization
      - Getting rid of common blocks, GOTO's, etc.
      - Separating file I/O from places where computations are done
      - Separating out G–C code that will not be used by GEOS–5 GCM
  - G–C column code will become the “standard” code
    - Same code base can be used:
      - For use within GEOS–5 GCM
      - For our own standalone purposes
    - For our own standalone purposes, we can write our own “driver”
      - Our own “driver” would read offline met fields, etc.
      - “Driver” code can use ESMF software framework
      - This will enable us to use MPI parallelization (finally!)
      - We will be able to run at higher resolution grids

## 2. Motivation for Column Code

- Column code also has several advantages for us (#2)
  - Column code reduces memory requirements
    - You remove 2 dimensions (lon, lat) from all arrays
    - Better enables us to do high-resolution simulations

| Grid                | # of Levels | # of Tracers | Size of Tracer Array | # of Iterations over Lon & Lat |
|---------------------|-------------|--------------|----------------------|--------------------------------|
| GEOS-5<br>4 x 5     | 47          | 54           | 20.3 kB              | 72 x 46 = 3,312                |
| GEOS-5<br>2 x 2.5   | 47          | 54           | 20.3 kB              | 144 x 91 = 13,104              |
| GEOS-5<br>1 x 1     | 47          | 54           | 20.3 kB              | 360 x 180 = 64,800             |
| GEOS-5<br>1/2 x 2/3 | 47          | 54           | 20.3 kB              | 540 x 360 = 194,400            |
| GEOS-5<br>1/4 x 1/4 | 47          | 54           | 20.3 kB              | 1440 x 720 =<br>1,036,800      |

The table at right shows the size of the tracer array in the GEOS-Chem column code for various GEOS-5 global grids.

Assume 47 layers (collapsed down from 72) and a REAL\*8 array.

NOTE: We trade off memory for iterations. We no longer carry (lon,lat) dimensions in the arrays, but must now call the column code once for each (lon,lat) location.

## 2. Motivation for Column Code

Advection [  $\mathbf{A}(:, :, :), \mathbf{Q}(:, :, :, :)$  ]

```
DO J = 1, N_Latitudes
DO I = 1, N_Longitudes
  Ac(:) = A(I, J, :)
  Qc(:, :) = Q(I, J, :, :)
```

PBL Mixing [  $\mathbf{Ac}(:), \mathbf{Qc}(:, :)$  ]

Cloud Conv. [  $\mathbf{Ac}(:), \mathbf{Qc}(:, :)$  ]

Dry Dep [  $\mathbf{Ac}(:), \mathbf{Qc}(:, :)$  ]

Emissions [  $\mathbf{Ac}(:), \mathbf{Qc}(:, :)$  ]

Chemistry [  $\mathbf{Ac}(:), \mathbf{Qc}(:, :)$  ]

Wet Dep [  $\mathbf{Ac}(:), \mathbf{Qc}(:, :)$  ]

```
ENDDO
ENDDO
```

### Schematic of “columnized” GEOS–Chem

Advection is not a column process.

The GEOS–5 GCM dynamical core handles advection, PBL mixing, and cloud convection (denoted by gray boxes). Therefore these are not needed for inclusion into GEOS–5. (They will be needed for a standalone GEOS–Chem.)

By looping over longitude and latitude at the interface level, we remove 2 dimensions from all arrays.

Data pertaining to a single column at surface lon/lat location (I,J) is sent down to the individual processes.

Some 3-D global storage is needed, this is called the “internal state”.

## 2. Motivation for Column Code

- Summary, Part 2
  - To interface GEOS–Chem w/ GEOS–5 GCM
    - To ensure a steady stream of innovation from G–C to NASA
  - Column code becomes “new” GEOS–Chem for us
    - One common code base for GEOS–5 and standalone G–C
  - Several potential applications on hi-res grids, including:
    - Aircraft mission support (i.e. NRT simulations @ hi-res)
    - Studies of climate-chemistry interactions
    - Regional air quality studies
    - Aerosol microphysics
    - etc.

# 3. “New World Order”: Updated Coding Standards

- Overarching principle when “columnizing” code:
  - ***Do not change algorithms, just change interfaces!***
    - We are just changing how the data flows in & out of routines
    - We do NOT change the science content of the routines (yet)
- Ramifications:
  - We need to adhere to a set of new coding rules
    - Some are dictated by the “Interface” to GEOS–5
    - Others are dictated by various technical concerns
  - The science content in the column code will necessarily be a little bit “behind the times” during the columnization process
    - However, after the code has been columnized we can always go back and rapidly add the various scientific updates

# 3. “New World Order”: Updated Coding Standards

## ***Rule #1:***

- Data shall be passed between routines via the argument list, rather than via USE statements or common blocks.

## ***Rationale for Rule #1:***

- To ensure a clean interface between routines
- To separate out code that the GEOS–5 GCM doesn't need:
  - Routines that read met fields from disk
  - GEOS–Chem diagnostic output routines, etc.
- Code is cleaner and more easily documented with all data coming in thru the argument list.

### 3. “New World Order”: Updated Coding Standards

Here is an example of a subroutine in the standard mainline GEOS–Chem code. (Comments omitted for brevity.)

Note that some of the **variables** are passed to the routine via the argument list, and others are passed via direct USE statements. Also some **routines** that are passed by direct USE statements refer to code that the GEOS–5 GCM would not use.

```
SUBROUTINE FAST_J( SUNCOS, OD, ALBD )

USE DAO_MOD,      ONLY : T, CLDF
USE ERROR_MOD,    ONLY : ERROR_STOP
USE GRID_MOD,     ONLY : GET_YMID
USE PRESSURE_MOD, ONLY : GET_PEDGE
USE TIME_MOD,     ONLY : GET_MONTH, GET_DAY, GET_DAY_OF_YEAR
USE TIME_MOD,     ONLY : GET_TAU,   GET_YEAR
USE TOMS_MOD,     ONLY : READ_TOMS

IMPLICIT NONE

#   include "cmn_fj.h"
#   include "jv_cmn.h"

REAL*8, INTENT(IN)      :: SUNCOS(MAXIJ)
REAL*8, INTENT(IN)      :: OD(LLPAR, IIPAR, JJPAR)
REAL*8, INTENT(IN)      :: ALBD(IIPAR, JJPAR)
```

# 3. “New World Order”: Updated Coding Standards

- Here is the equivalent column-code routine (all **variables** via the argument list)

```
SUBROUTINE FAST_J( L_COLUMN, N_AER, N_DUST, N_RH,  
& N_JV, MONTH, DAY, DAY_OF_YR,  
& CSZA, SFCA, YLAT, PEDGE,  
& TEMP, CLDF, OD_VIS, OD_DUST,  
& OD_AER, TO3, J_VALUE, RC
```

Note that all variables to column code routine FAST\_J are now passed as arguments.

```
IMPLICIT NONE
```

```
. . .
```

```
INTEGER, INTENT(IN) :: L_COLUMN  
INTEGER, INTENT(IN) :: N_AER  
INTEGER, INTENT(IN) :: N_DUST  
INTEGER, INTENT(IN) :: N_RH  
INTEGER, INTENT(IN) :: N_JV  
INTEGER, INTENT(IN) :: MONTH  
INTEGER, INTENT(IN) :: DAY  
INTEGER, INTENT(IN) :: DAY_OF_YR  
REAL*8, INTENT(IN) :: CSZA  
REAL*8, INTENT(IN) :: SFCA  
REAL*8, INTENT(IN) :: YLAT  
REAL*8, INTENT(IN) :: PEDGE (L_COLUMN+1 )  
REAL*8, INTENT(IN) :: TEMP (L_COLUMN )  
REAL*8, INTENT(IN) :: CLDF (L_COLUMN )  
REAL*8, INTENT(IN) :: OD_VIS (L_COLUMN )  
REAL*8, INTENT(IN) :: OD_DUST(L_COLUMN,N_DUST )  
REAL*8, INTENT(IN) :: OD_AER (L_COLUMN,N_AER*N_RH)  
REAL*8, INTENT(IN) :: TO3  
  
REAL*8, INTENT(OUT) :: J_VALUE(L_COLUMN,N_JV )  
INTEGER, INTENT(OUT) :: RC
```

Also, to the greatest extent possible, we compute quantities (e.g. month, day, day-of-year, year, total ozone column TO3, etc. ) at a higher level in the code and then simply pass them into FAST\_J as arguments.

This is necessary because when we interface with the GEOS-5 GCM, all quantities will come from the GCM as arguments and will get passed down to the underlying Fortran routines.

# 3. "New World Order": Updated Coding Standards

- One can use new coding features of F90 to reduce the number of arguments that need to be passed to routines. In particular, derived type variables:

```
! This is a derived type, analogous to a structure in IDL
! In Java or C++ we would call this a "CLASS"
```

```
TYPE :: GC_TIME
  INTEGER :: YEAR      ! Current year (YYYY)
  INTEGER :: MONTH     ! Current month (1-12)
  INTEGER :: DAY       ! Current day (1-31)
  INTEGER :: DOY       ! Day of year (0-365/366)
  INTEGER :: HOUR      ! Current hour (0-23)
  INTEGER :: MINUTE    ! Current minute (0-59)
END TYPE GC_TIME
```

```
! This is a variable based on the derived type.
! In Java or C++ we would call this an "OBJECT"
```

```
TYPE(GC_TIME) :: TIMING
```

```
! We use the "%" character to refer to fields in the TIMING variable.
```

```
PRINT*, TIMING%YEAR
```

```
! We can pass TIMING to a subroutine to reduce the # of arguments
```

```
CALL MY_SUB( TIMING, . . . )
```

# 3. “New World Order”: Updated Coding Standards

## ***Rule #2:***

- NO MORE COMMON BLOCKS!!!!!!

## ***Exceptions:***

- Common blocks for 3rd-party legacy code:
  - FAST-J, SMVGEAR, ISORROPIA, etc.
  - Arrays MUST be sized for a single vertical column (not 3-D global grid)

## ***Rationale for Rule #2:***

- See Rule #1

# 3. “New World Order”: Updated Coding Standards

## *This is bad:*

```
COMMON /BLK1/ ARRAY (MAX_LON, MAX_LAT, MAX_LEVELS)
```

- Due to technical reasons, the global 3-D ARRAY within this common block cannot be “split up” to multiple CPU's via MPI parallelization
- Memory for ARRAY is allocated statically.
  - i.e. memory gets set for ARRAY whether ARRAY is actually used in the code or not.

## *This is tolerable:*

```
COMMON /BLK1/ ARRAY (MAX_LEVELS)
```

- Now that ARRAY is 1-dimensional, it is typically small enough such that the entire ARRAY can fit onto a CPU w/o having to be split up
  - This allows us to use MPI parallelization
- Memory is still allocated statically.
  - MAX\_LEVELS typically = 47 or 72 so the array is not very large.

# 3. “New World Order”: Updated Coding Standards

## ***Rule #3:***

- Include files may be used, provided that they only contain PARAMETERS (i.e. constants).

## ***Rationale for Rule #3:***

- Certain constants need to be shared between several routines
  - Maximum limits for arrays
  - Physical constants
  - Error return codes
- It is more efficient (and safer!) to declare these constants in a file (or multiple files) for inclusion into the various routines
  - Otherwise we'd have to declare these constants in many routines
  - This leads to errors (e.g. if we update in one routine but not another)

# 3. “New World Order”: Updated Coding Standards

- This include file `smv_physconst.h` defines various physical constants.
  - All variables are declared as **PARAMETERS**
  - Also note, there are no COMMON blocks in this file

```
! !INCLUDE: smv_physconst.h
!  
! !DESCRIPTION: This include file contains physical constants for the
! GEOS-Chem column chemistry code.
!\\
!\\
! !DEFINED PARAMETERS:
!  
! Molecular weight of air [28.97e-3 kg/mol]
REAL*8, PARAMETER :: MW_AIR          = 28.97d-3  
  
! Avogadro's # [# /mol]
REAL*8, PARAMETER :: AVO              = 6.022d23  
  
! g0      : Gravity at Surface of Earth [9.8 m/s^2]
REAL*8, PARAMETER :: g0               = 9.8d0  
  
! PI      : Double-Precision value of PI
REAL*8, PARAMETER :: PI               = 3.14159265358979323d0  
  
! Re      : Radius of Earth [m]
REAL*8, PARAMETER :: Re               = 6.375d6  
  
Etc ...
```

- And this is how we include it into a code:

```
# include "smv_physconst.h"
```

# 3. “New World Order”: Updated Coding Standards

## ***Rule #4:***

- When the code encounters an error, do not simply “die” with an error message.
  - Instead, immediately exit the routine (and all higher-level routines) with a descriptive error code
  - Error trapping is done in the “Interface” level.

## ***Rationale for Rule #4:***

- When the G–C column code is inserted into the GEOS–5 GCM, stopping unexpectedly can cause unforeseen side effects
  - It could hang a machine or cause some other bad result
  - Better to give GEOS–5 GCM the opportunity to shut down gracefully

# 3. “New World Order”: Updated Coding Standards

- Include file `smv_errcode.h` contains various parameters which specify success or failure in various locations in the code.

```
! Return w/ success
INTEGER, PARAMETER :: SMV_SUCCESS           = 0
. . .
! Return codes for chemistry setup routines
INTEGER, PARAMETER :: SMV_FAIL_GASCONC     = -1200
```

- And then each routine that we call (in this case, GASCONC), will return a variable **RC**.
  - **RC == 0** if the routine finishes normally. Anything else is a failure and we return.
  - Similar IF statements in the calling routines will make sure we exit out of those too, until we reach the “Interface” level, where the error is handled.

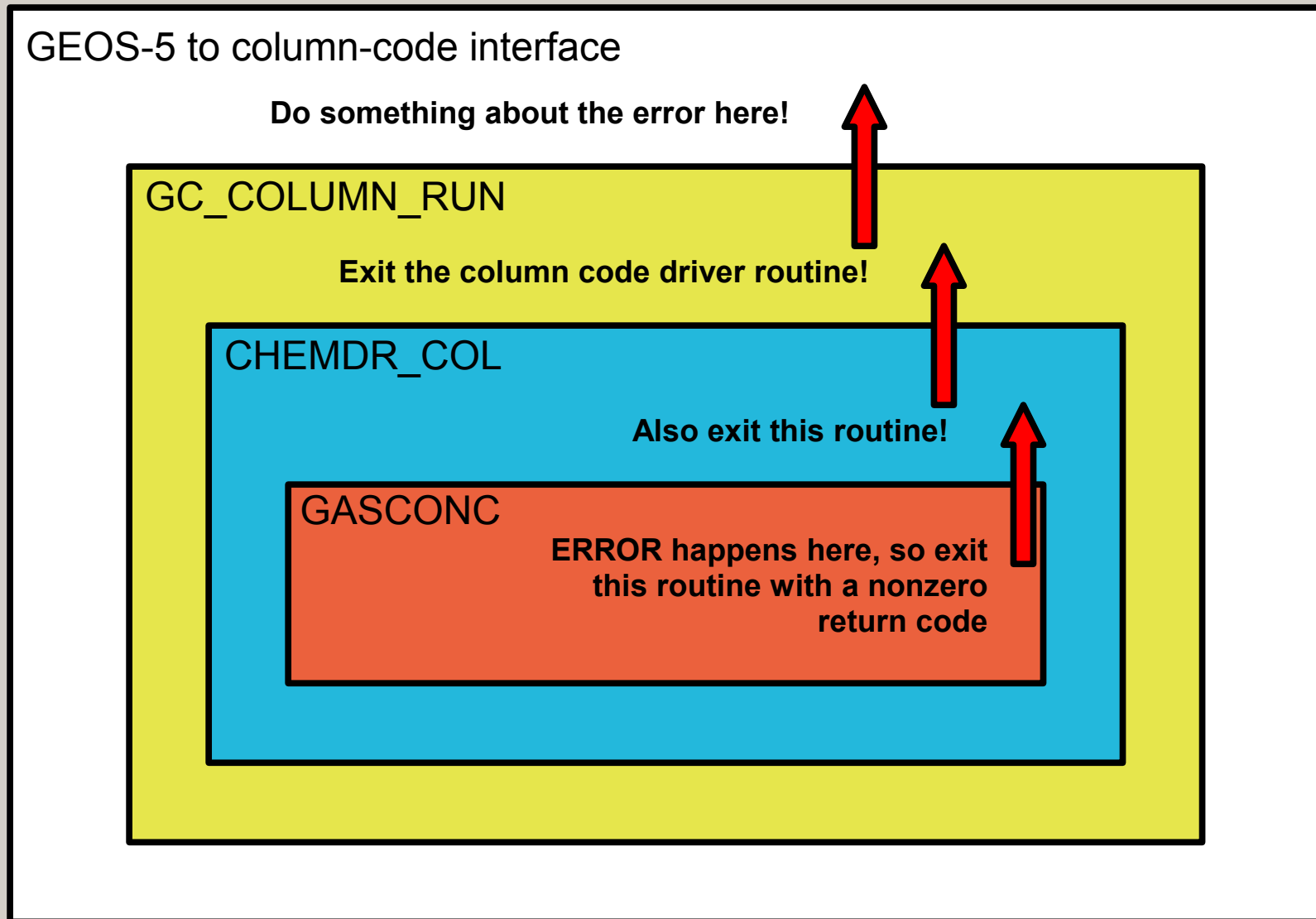
```
! Include file contains various parameters, each of which describe a
! different type of error.
# include "smv_errcode.h"

! Call GASCONC
CALL GASCONC( . . . , RC )

! Return w/ error code if failure
IF ( RC /= SMV_SUCCESS ) RETURN
```

# 3. “New World Order”: Updated Coding Standards

- “Fire Drill” error trapping!



### 3. “New World Order”: Updated Coding Standards

#### ***Rule #5:***

- All columnized routines shall use the ProTeX documentation headers.

#### ***Rationale for Rule #5:***

- This allows us to generate a detailed reference document in LaTeX from the comments in the source code
- LaTeX file can be converted to both PS and PDF formats
- For an example, see the file:  
[http://acmg.seas.harvard.edu/geos/wiki\\_docs/column/gc\\_column\\_chemistry.pdf](http://acmg.seas.harvard.edu/geos/wiki_docs/column/gc_column_chemistry.pdf)

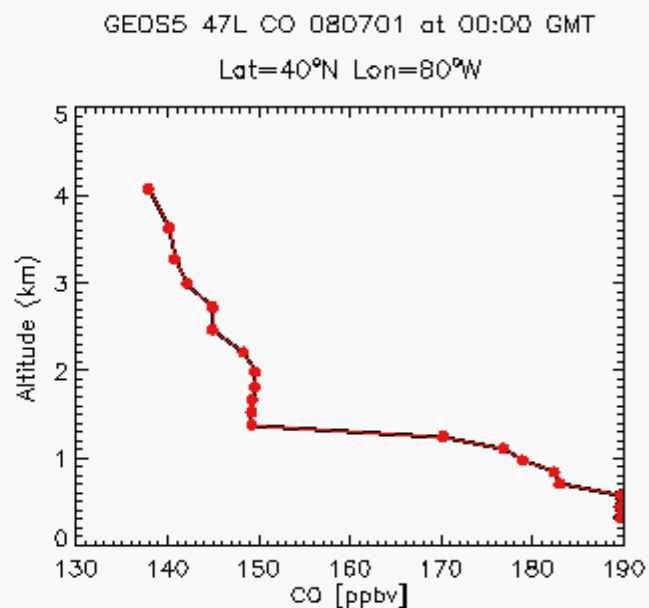
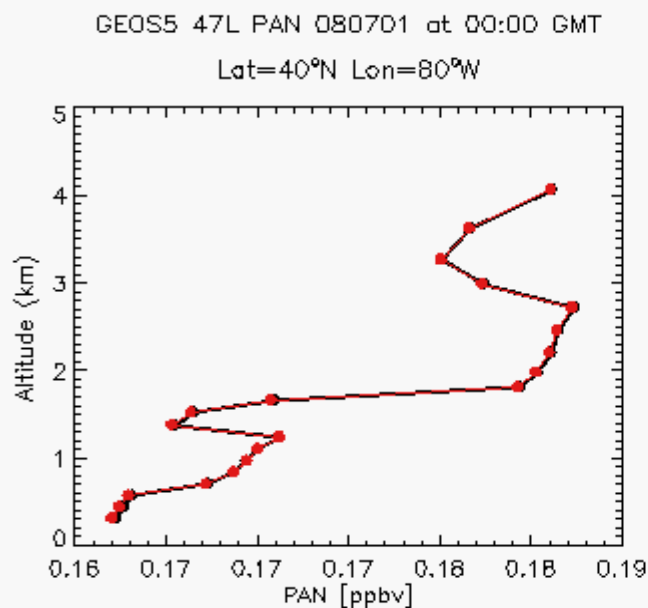
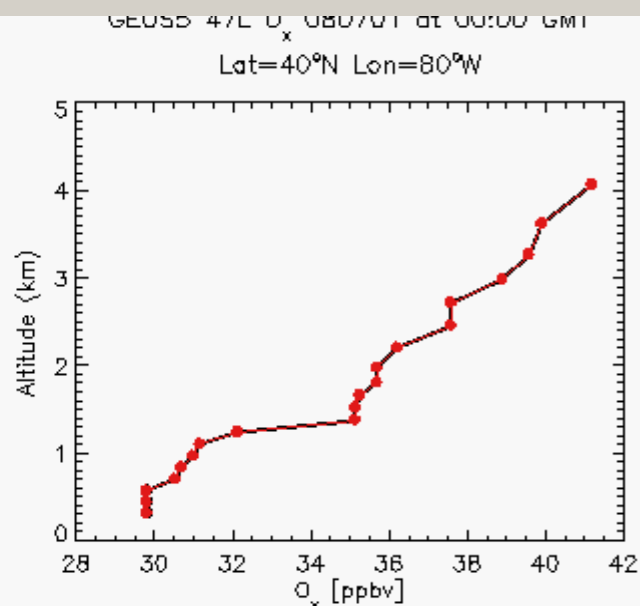
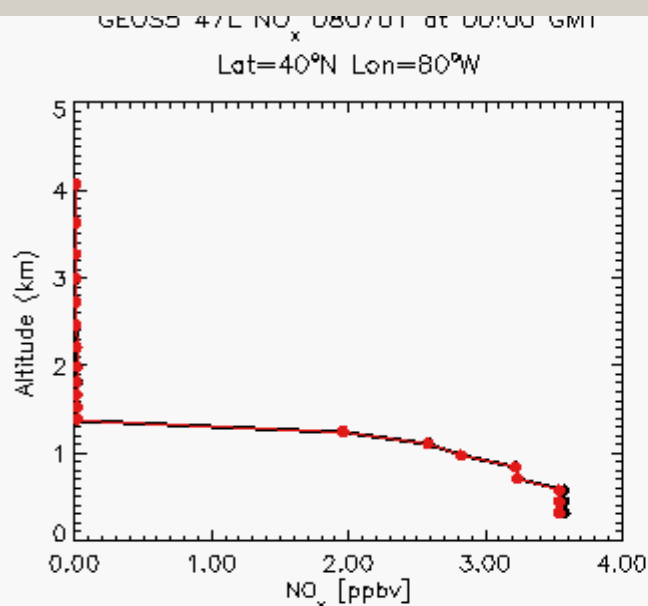
### 3. “New World Order”: Updated Coding Standards

- Summary of “New World Order”
  - 1) All variables thru the argument list
  - 2) No common blocks
    - a) Except in legacy code
    - b) And only if the common blocks contain data on one single column
  - 3) Include files may contain only PARAMETER values
  - 4) Exit all the way out of the column code upon errors
    - a) Deal with the error in the “interface” level
  - 5) Use ProTeX documentation
- We encourage all GEOS–Chem users to start using these new rules of programming now!

# 4. Progress Report

- Testing and Validation
  - The column code is being tested within a version of G–C
    - Base version: v8-01-04
    - An “interface” has been set up to extract the met fields and other relevant values from G–C for passing down to the column code.
  - Column code is tested against a “reference” code
    - At all stages of the columnization process, the column code is checked against the same “base” version of GEOS–Chem
    - Various values were compared, including:
      - Reaction rates
      - Photolysis rates
      - Concentrations
      - Mean OH
      - etc.
    - “Quick-look” maps – vertical profiles of tracer concentrations
    - NOTE: 100% strict numerical compatibility not possible.

# 4. Progress Report



Example of a  
“quick look”  
profile plot.

Black:  
reference  
code

Red:  
column code

If all is well,  
the two  
profiles  
should  
overlap each  
other.

Some minor  
numerical  
differences  
expected.

# 4. Progress Report

- The following operations have been columnized: (#1)
  - Chemistry
    - Gas-phase mechanism w/ SMVGEAR
    - FAST–J photolysis
    - Aerosol chemistry routines:
      - Carbonaceous aerosols
      - Mineral dust aerosols
      - Sea salt aerosols
      - Sulfate aerosols
      - Aerosol thermodynamical equilibrium
  - Dry Deposition
  - Wet Deposition

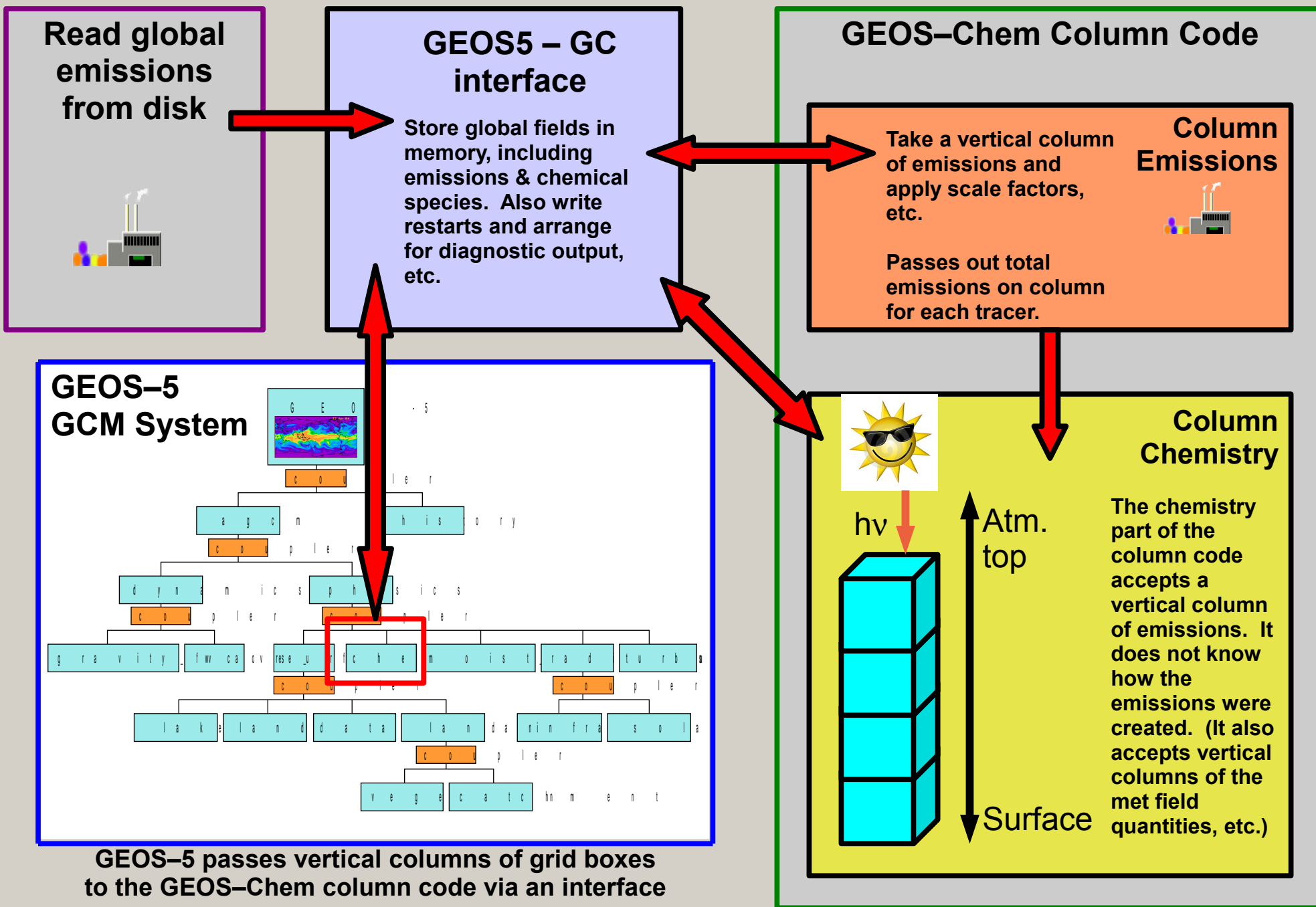
# 4. Progress Report

- The following operations have been columnized: (#2)
  - Routines not needed for GEOS–5 GCM but only for the standalone GEOS–Chem column code.
    - PBL Mixing
    - Cloud Convection
  - Land surface data
    - Daily leaf area indices
    - Olson land type information (i.e. `vegtype.global`)
  - Emissions
    - Dust mobilization
    - NO<sub>x</sub> from soils & fertilizers

# 4. Progress Report

- Much remains to be done, however
  - Continue to columnize the various emissions routines:
    - Aircraft
    - Anthropogenic (global & regional inventories)
    - Biomass burning (e.g. GFED)
    - Biogenic (e.g. MEGAN)
    - Lightning
    - Plus monthly / annual scale factors, etc.
  - “Emissions reader” component
    - To read in 2D and 3D emissions data from disk
    - All data needs to be read at the start of the simulation, instead of being continually read every month
    - Columns of emissions data will be sent to the chemistry code, where they will be further processed
    - Some code could be taken from GMI

# Final picture: Column code w/in GEOS-5

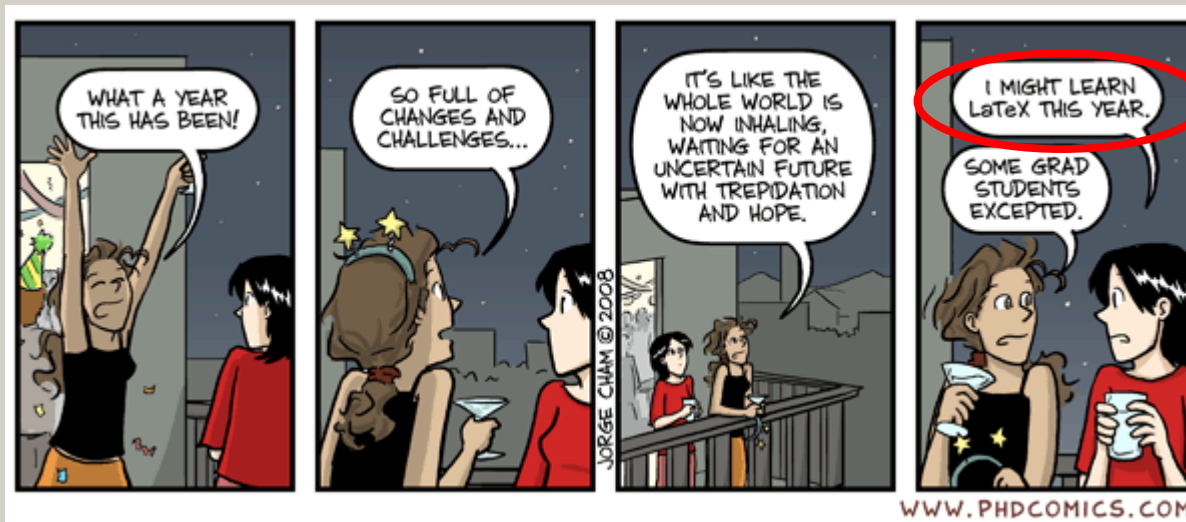


# 4. Progress Report

- Acknowledgements
  - NASA/GMAO
    - Steven Pawson
    - Tom Clune
    - Max Suarez
    - Arlindo da Silva
  - Harvard
    - Philippe Le Sager
    - Daniel Jacob
    - And all of you for your understanding! :-)

# Questions???

## *Happy New Year!!!!*



Needed for use with ProTeX!

# Extra Slides

# 3. “New World Order”: Updated Coding Standards

- Here is an example of a function. It's similar to a subroutine, only that the output comes via the variable specified with the RESULT keyword.

```
FUNCTION BIO_FIT( COEFF1, XLAI1,  
&                SUNCOS1, CFRAC1 ) RESULT( BIOFIT )  
  . . .  
  
  ! Arguments  
  REAL*8, INTENT(IN) :: COEFF1  
  REAL*8, INTENT(IN) :: XLAI1  
  REAL*8, INTENT(IN) :: SUNCOS1  
  REAL*8, INTENT(IN) :: CFRAC1  
  
  ! Return value  
  REAL*8                :: BIOFIT  
  
  BIOFIT = . . .  
  
END SUBROUTINE BIO_FIT
```

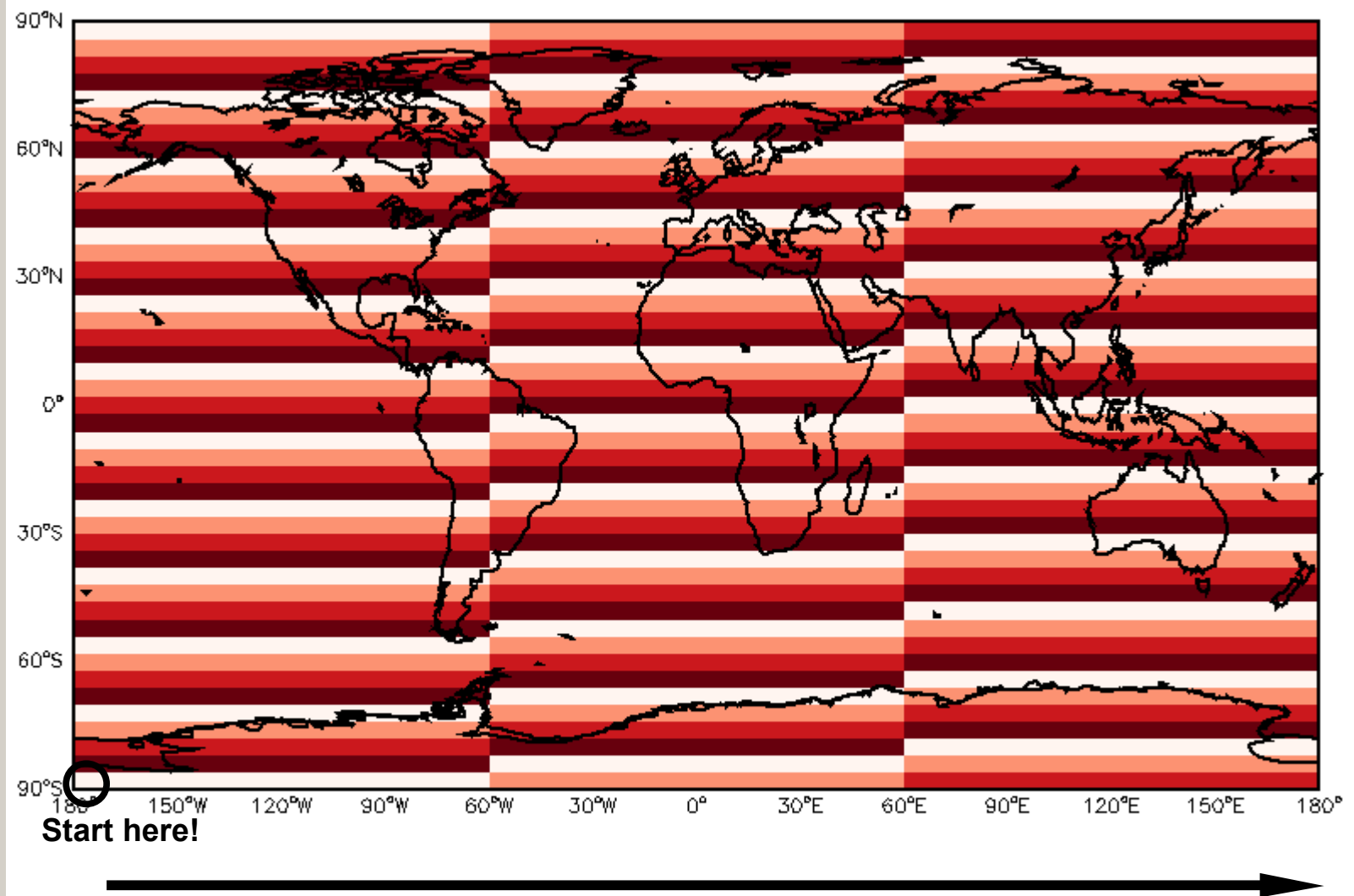
# GEOS-Chem overview

## Map of how SMVGEAR groups boxes for parallel processing @ 4x5

SMVGEAR sends “blocks” of G-C grid boxes to the CPU's on your system. This is done in the loop in routine PHYSPROC.

Each “block” contains 24 G-C grid boxes contiguous in longitude. We start at the S Pole box at the surface and then move eastward.

Here we assume a 4x5 grid and 4 CPUs (which are represented by different colors).



# GEOS-Chem overview

## Map of how SMVGEAR groups boxes for parallel processing @ 2x25

Similar example, this time for a 2 x 2.5 grid and w/ 4 CPU's. Each CPU is represented by a different color on the plot.

As you can see, This distribution of grid boxes does not suit the column nature of the chemistry processes.

It also requires several 3-D arrays to map between the "global" and "local" arrays w/in the code.

