Modifying Code in the CLM

CLM Tutorial 2019
Bill Sacks, Jackie Shuman, Danica Lombardozzi

NCAR is sponsored by the National Science Foundation

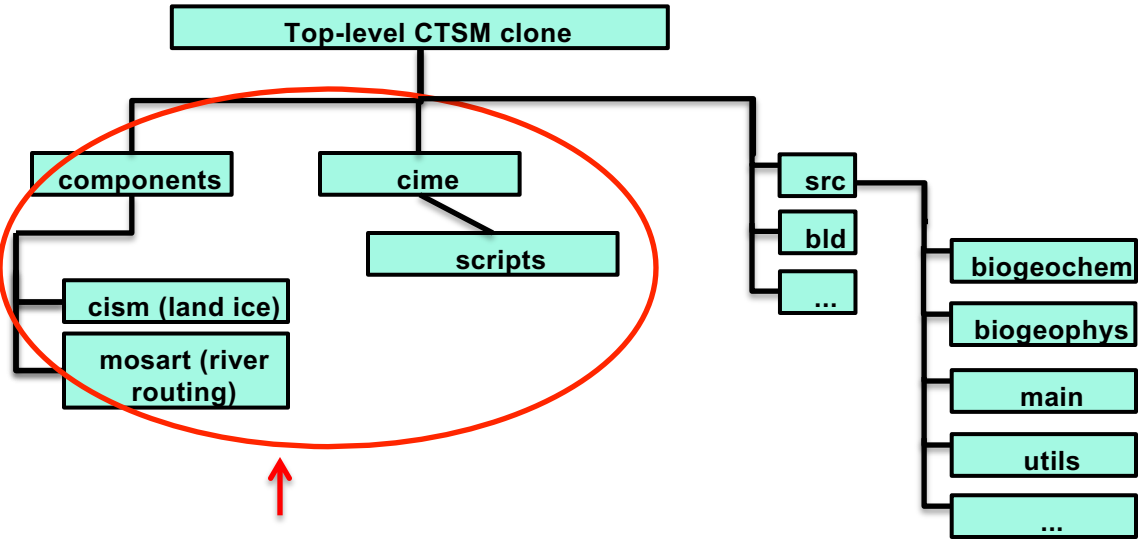
Why might you modify the code?

- Improve process representation based on new scientific findings
- Introduce a new concept
- Test the sensitivity of an existing representation
- And more…

Outline

- GitHub walkthrough
- Overview of modifying the code
- #1 best practice: don’t repeat yourself
- CLM arrays, loops and filters
- Practical exercises

GitHub walkthrough



Obtained via manage_externals

Source code changes anywhere in this directory tree affect all cases created from here.

If you have any existing cases created from a clone which are still running, or which you might want to continue: Then create a new git clone for any new work (i.e., a separate git clone for each git branch).

Use grep to find specific variables / key words of interest.

Fast search, only covers CTSM git repository (not externals):

```
git grep -i ozone
```

To search everything from this directory down, including externals (note dot at end):

```
grep -r -i ozone .
```

Review: The 4 commands to run CLM

(1) create a new case

(2) invoke case.setup

Make modifications any time before compiling.

(3) build the executable (./case.build)

Also okay to make modifications after compiling. Then need to rerun ./case.build.

(4) submit your run to the batch queue

For more elaborate mods…

Can follow examples from existing code

- Keep in mind that some examples are better than others, or at least more appropriate for the changes you want to make
- So: best to check in with experienced CLM developers initially

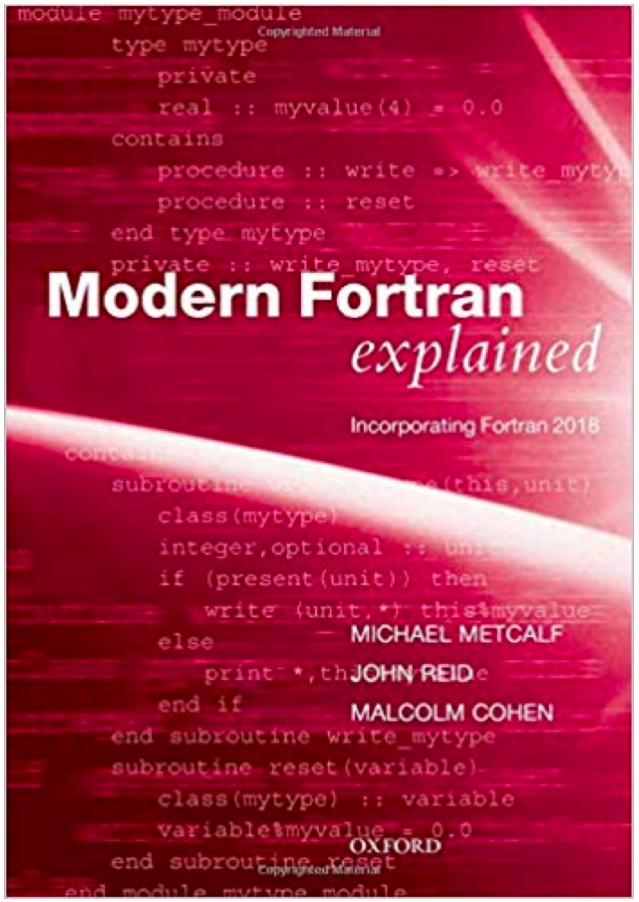
Also look through development guides on the GitHub wiki – though these are still a work-in-progress.

Code language

CLM – like most of CESM – is written in modern Fortran

(Fortran 90/95/2003)

We allow – and encourage! – the use of Fortran 2003 object orientation.



#1 Best Practice: Don't Repeat Yourself

Please don't
do this

```
subroutine SaturatedExcessRunoffTopmodel (...)
  ! Compute fsat
  do fc = 1, num_hydrologyc
     c = filter_hydrologyc(fc)
     fff = 0.5_r8
     if (frost_table(c) > zwt_perched(c) .and. frost_table(c) <= zwt(c)) then
        fsat(c) = wtfact(c) * exp(-0.5_r8*fff*zwt_perched(c))
     else
        fsat(c) = wtfact(c) * exp(-0.5_r8*fff*zwt(c))
     end if
  end do

  ! Set fsat to zero for crop columns
  if (crop_fsat_equals_zero) then
     do fc = 1, num_hydrologyc
        c = filter_hydrologyc(fc)
        l = col%landunit(c)
        if(lun%itype(l) == istcrop) fsat(c) = 0._r8
     end do
  endif

  ! Compute qflx_sat_excess_surf
  do fc = 1, num_hydrologyc
     c = filter_hydrologyc(fc)
     qflx_sat_excess_surf(c) = fsat(c) * qflx_rain_plus_snomelt(c)
     if (col%urbpoi(c)) then
        qflx_sat_excess_surf(c) = qflx_sat_excess_surf(c) + qflx_floodc(c)
     end if
  end do
end subroutine SaturatedExcessRunoffTopmodel
```

```
subroutine SaturatedExcessRunoffVic (...)
  ! Compute fsat
  do fc = 1, num_hydrologyc
     c = filter_hydrologyc(fc)
     ex(c) = b_infil(c) / (1._r8 + b_infil(c))
     ! fsat is equivalent to A in VIC papers
     fsat(c) = 1._r8 - (1._r8 - top_moist_limited(c) / top_max_moist(c))**ex(c)
  end do




  ! Set fsat to zero for crop columns
  if (crop_fsat_equals_zero) then
     do fc = 1, num_hydrologyc
        c = filter_hydrologyc(fc)
        l = col%landunit(c)
        if(lun%itype(l) == istcrop) fsat(c) = 0._r8
     end do
  endif

  ! Compute qflx_sat_excess_surf
  do fc = 1, num_hydrologyc
     c = filter_hydrologyc(fc)
     qflx_sat_excess_surf(c) = fsat(c) * qflx_rain_plus_snomelt(c)
     if (col%urbpoi(c)) then
        qflx_sat_excess_surf(c) = qflx_sat_excess_surf(c) + qflx_floodc(c)
     end if
  end do
end subroutine SaturatedExcessRunoffVic
```

#1 Best Practice: Don't Repeat Yourself

- Why not to copy & paste existing code
  - For readers of the code, it's hard to tell how the two versions differ
  - If the shared piece changes, it's hard to realize that both routines need to change
  - And once they diverge, it's very hard to tell if the divergence is intentional or accidental
- Why not to copy & paste your own code
  - It will be harder to make changes that apply to each instance
  - It's harder to have confidence: need to separately test each instance of the duplicated code
  - If the instances are subtly different, it's hard to see that, and introducing a new instance is error-prone

#1 Best Practice: Don't Repeat Yourself

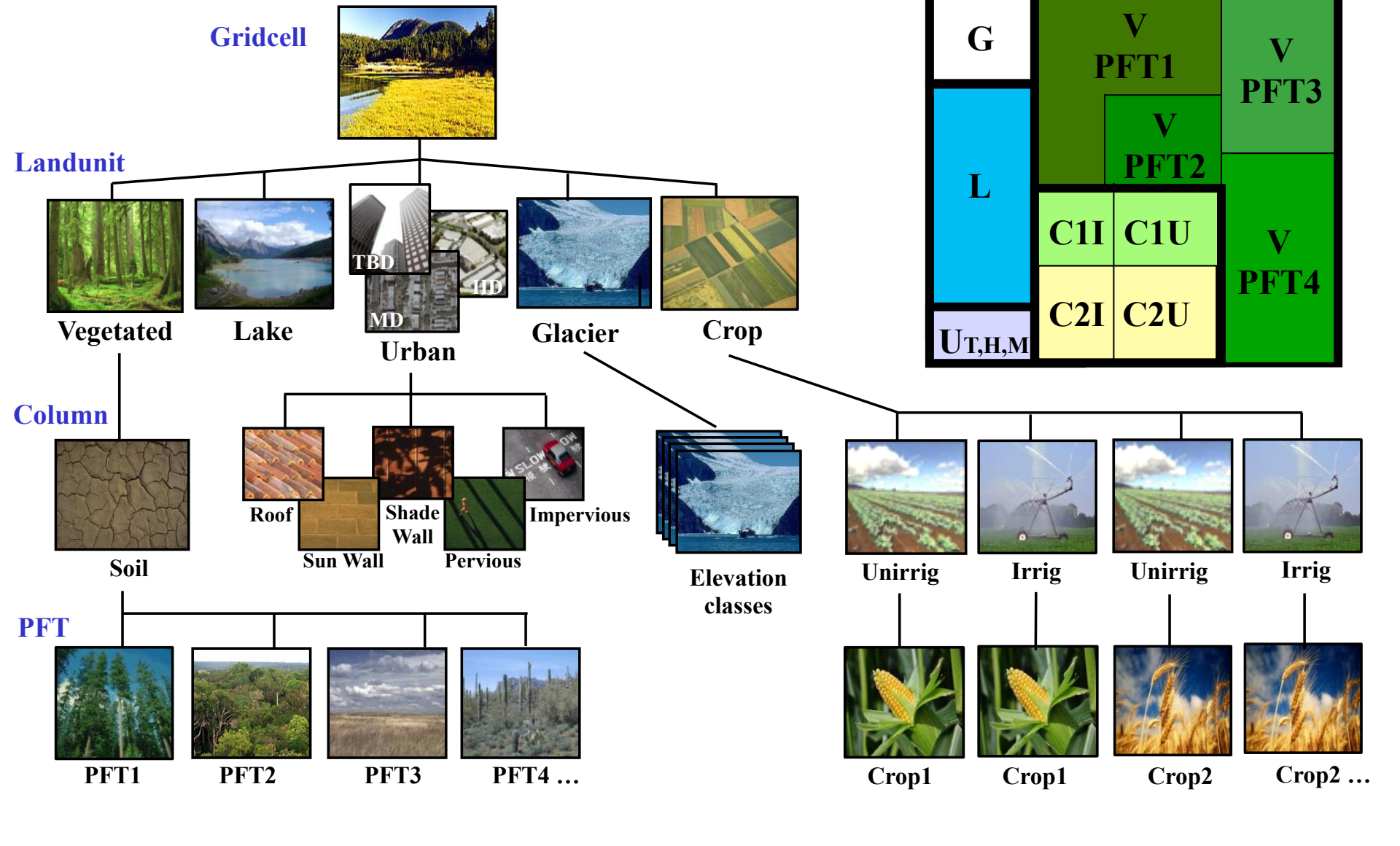
Instead

do this

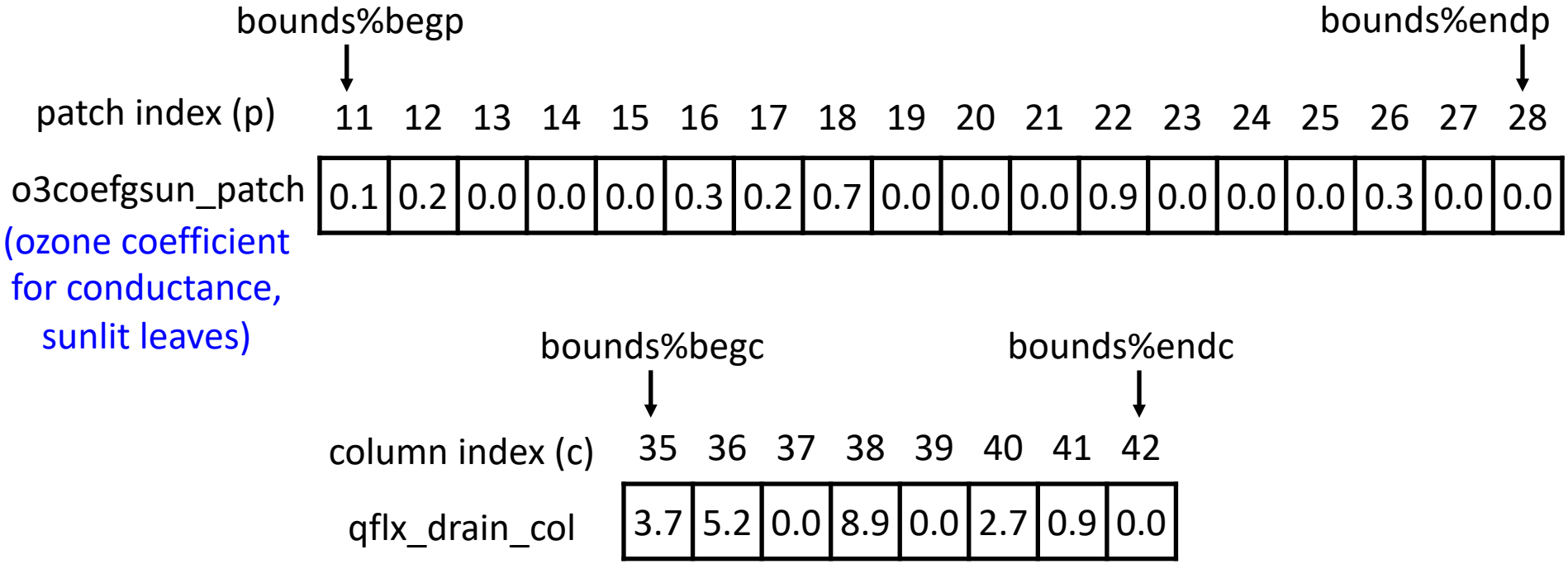
```
subroutine SaturatedExcessRunoff (...)
  ! Compute fsat
  select case (this%fsat_method)
  case (FSAT_METHOD_TOPMODEL)
     call this%ComputeFsatTopmodel(...)
  case (FSAT_METHOD_VIC)
     call this%ComputeFsatVic(...)
  case default
     call endrun(subname//' ERROR: Unrecognized fsat_method')
  end select

  ! Set fsat to zero for crop columns
  if (crop_fsat_equals_zero) then
     do fc = 1, num_hydrologyc
        c = filter_hydrologyc(fc)
        l = col%landunit(c)
        if(lun%itype(l) == istcrop) fsat(c) = 0._r8
     end do
  endif

  ! Compute qflx_sat_excess_surf
  do fc = 1, num_hydrologyc
     c = filter_hydrologyc(fc)
     qflx_sat_excess_surf(c) = fsat(c) * qflx_rain_plus_snomelt(c)
     if (col%urbpoi(c)) then
        qflx_sat_excess_surf(c) = qflx_sat_excess_surf(c) + qflx_floodc(c)
     end if
  end do
end subroutine SaturatedExcessRunoff
```

```
subroutine ComputeFsatTopmodel(...)
  do fc = 1, num_hydrologyc
     c = filter_hydrologyc(fc)
     fff = 0.5_r8
     if (frost_table(c) > zwt_perched(c) .and. frost_table(c) <= zwt(c)) then
        fsat(c) = wtfact(c) * exp(-0.5_r8*fff*zwt_perched(c))
     else
        fsat(c) = wtfact(c) * exp(-0.5_r8*fff*zwt(c))
     end if
  end do
end subroutine ComputeFsatTopmodel


subroutine ComputeFsatVic(...)
  do fc = 1, num_hydrologyc
     c = filter_hydrologyc(fc)
     ex(c) = b_infil(c) / (1._r8 + b_infil(c))
     fsat(c) = 1._r8 - (1._r8 - top_moist_limited(c) / top_max_moist(c))**ex(c)
  end do
end subroutine ComputeFsatVic
```

CLM Arrays, Loops and Filters



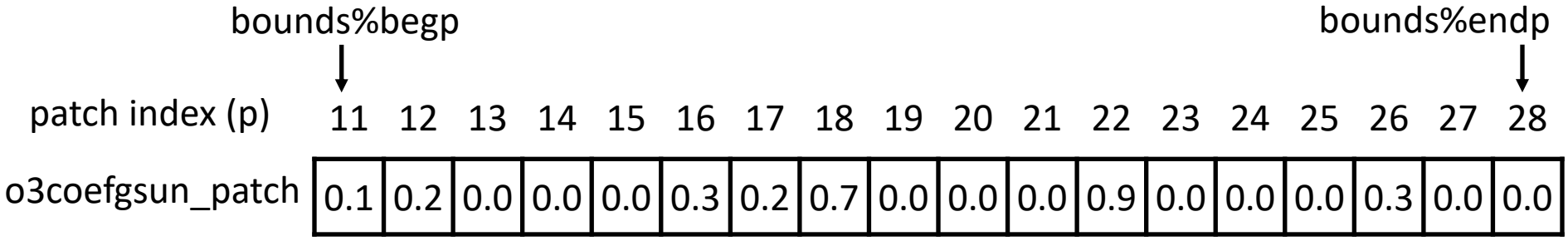
CLM Arrays, Loops and Filters



The _patch or _col often doesn't appear in the body of the code, but you can find it by looking at the 'associate' statement for a subroutine, which defines aliases:

```
associate( &
     o3coefvsun => this%o3coefvsun_patch, &
     )
```

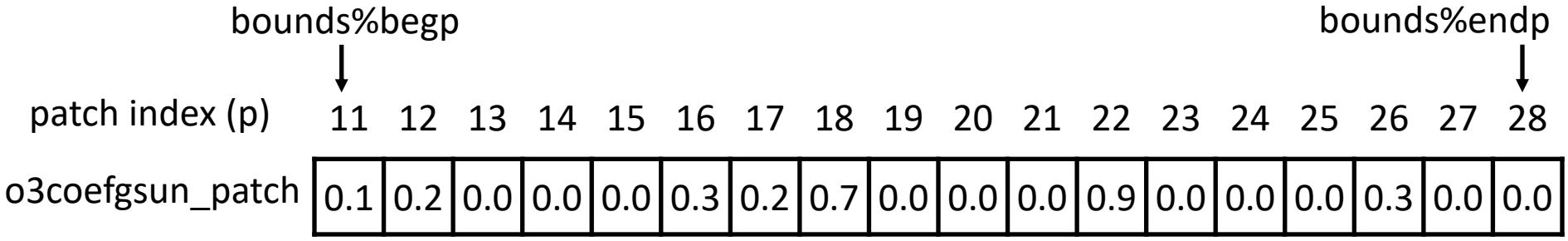
CLM Arrays, Loops and Filters



You could loop through a patch-level array like this:

```
do p = bounds%begp, bounds%endp

   o3coefgsun(p) = [some expression]
```

CLM Arrays, Loops and Filters



You could loop through a patch-level array like this:

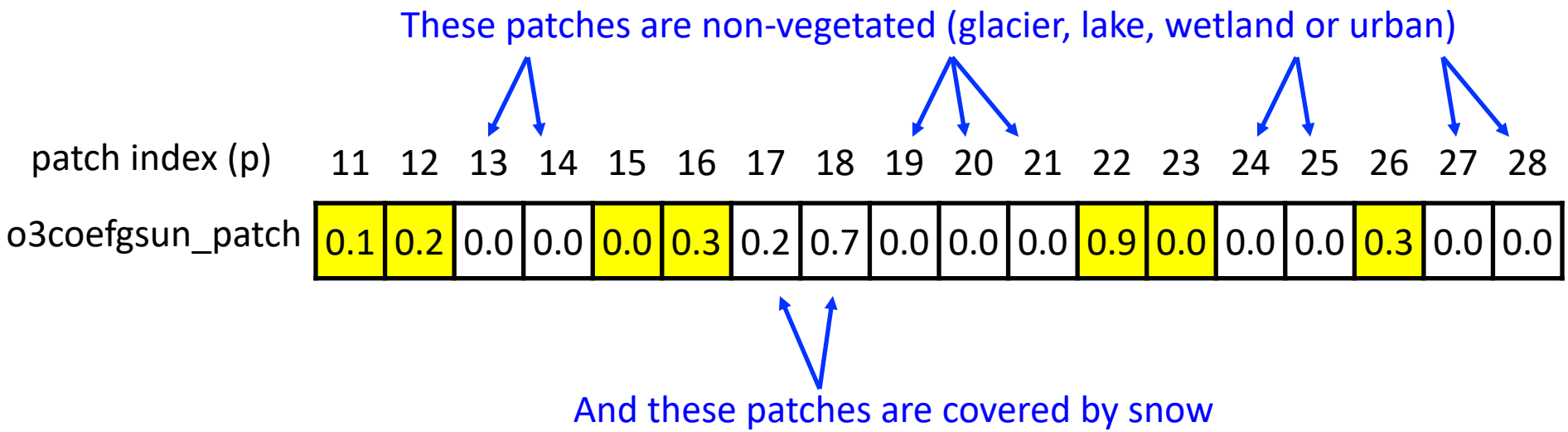
```
do p = bounds%begp, bounds%endp
   c = patch%column(p)

   o3coefgsun(p) = [some expression]
```

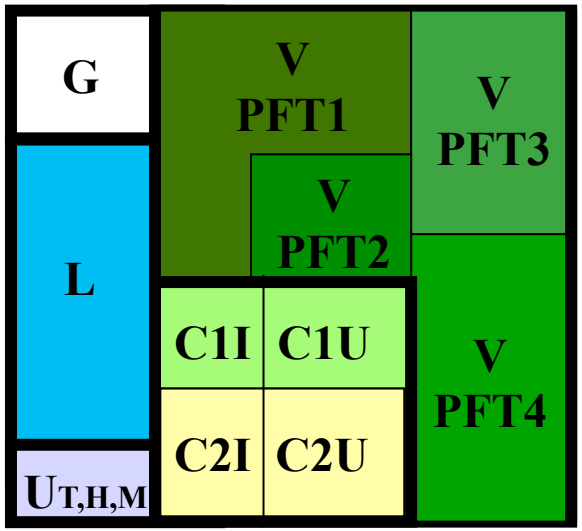
This line is only needed if you need to access column-level arrays in the same loop

But typically in CLM we use “filters” for efficiency (computing time is money)...

CLM Arrays, Loops and Filters

Filter of vegetated patches not covered by snow:





filter index (fp)	1	2	3	4	5	6	7
filter_exposedvegp	11	12	15	16	22	23	26

CLM Arrays, Loops and Filters

patch index (p)	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
o3coefgsun_patch	0.1	0.2	0.0	0.0	0.0	0.3	0.2	0.7	0.0	0.0	0.0	0.9	0.0	0.0	0.0	0.3	0.0	0.0

filter index (fp)	1	2	3	4	5	6	7
filter_exposedvegp	11	12	15	16	22	23	26

A loop using this filter looks like this:

```
do fp = 1, num_exposedvegp
   p = filter_exposedvegp(fp)

   o3coefgsun(p) = [some expression]
```

7 in this case

Overview of today's exercises

1) Run a control case for 5 days
   - Create and setup a case
   - Change namelist to enable ozone damage
   - Deal with a typo in user_nl_clm
   - Build and submit case

2) Run another case where we change the ozone plant stress coefficient
   - Create a new git clone and branch for this work
   - Change OzoneMod.F90
   - Create and setup another case
   - Deal with a compilation error
   - Build and submit case
   - Run will (probably) crash

3) Run another case that is the same as (2) but built in DEBUG mode
   - Create and setup another case
   - Change a setting in env_build.xml to build in DEBUG mode rather than optimized mode
   - Build and submit case
   - Run will crash; examine log files to determine the cause of the crash
   - Fix the problem, rebuild and resubmit

Bonus exercises

4) Rebuild and resubmit the non-debug experimental case

- Rebuild and resubmit
- Compare with the control case

5) Add a history (diagnostic) field

- Make a source code change to add a history field
- Rebuild and resubmit the earlier case
- Examine output to confirm the field has been added correctly

Notes about today's exercises

Today's exercises are largely about how to track down problems – typos, bugs, etc. So the exercises will often ask you to do something that isn't quite right. If you notice this immediately, give yourself a pat on the back – but to get the most benefit from these exercises, you should type things in exactly as written, bugs and all. We'll work through these problems together in the following slides.

As in previous exercises: `green text in a fixed-width font` indicates things you will type, either at the command line or in an editor.

Start Practical Here

Exercise 3.1:
Namelist changes
to set up control simulation

Create a new git clone

We'll start with a fresh git clone for today's work to avoid problems in case you changed anything yesterday.

(1) Create a directory for today's tutorial

```
cd ~
mkdir practical3
cd practical3
```

(2) Create a fresh clone

```
git clone -b release-clm5.0 https://github.com/ESCOMP/ctsm.git clm5.0_control
cd clm5.0_control
./manage_externals/checkout_externals
```

Create a new case

(3) Create a new case

```
cd cime/scripts
./create_newcase --case ~/clm_tutorial_cases/ozone_control --compset
I2000Clm50SpGs --res f45_g37 --run-unsupported --project UCGD0004
```

`Gs` in the compset name indicates a stub glacier model (rather than CISM); we're using that to speed up the build

`--run-unsupported` is needed because this compset/resolution combination isn't in our test suite

(4) Do the initial case setup

```
cd ~/clm_tutorial_cases/ozone_control
./xmlchange JOB_QUEUE=R4231261 --subgroup case.run --force
./xmlchange JOB_WALLCLOCK_TIME=00:05:00
./case.setup
```

Change some namelist options

(5) Output daily diagnostics, since we're only running for the default 5 days: Add the following to user_nl_clm:

```
hist_nhtfrq = -24.0
hist_mfilt = 6
```

`--hist_mfilt` puts all output from this 5-day run in a single file

(6) Change two physics options: turn on ozone (which is central to this exercise), and turn off plant hydraulic stress (because the interaction between ozone and plant hydraulic stress is counter-intuitive). Add the following to user_nl_clm:

```
use_ozone = .true.
use_hydrstress = .false.
```

(7) Run the script to generate namelists to make sure your namelist settings don't have any typos:

```
./preview_namelists
```

What happened?

(See next slide for solution)

Change some namelist options

(8) Fix this setting in user_nl_clm:

```
hist_nhtfrq = -24
```

(9) Run the script to generate namelists to make sure your namelist settings don't have any typos:

```
./preview_namelists
```

This time you should see:

```
Finished creating component namelists
```

Build and submit the run

(10) Build the model:

```
qcmd -q R4231261 -- ./case.build
```

***If you had an account on cheyenne before the tutorial**, make sure the PBS_ACCOUNT is set to UCGD0004 before you build.*

You should see the following, indicating that the build completed successfully:

```
MODEL BUILD HAS FINISHED SUCCESSFULLY
```

Since this will take a few minutes, you can go on to Exercise 3.2 in a new terminal window while waiting for the build to finish.

(11) Submit the run:

```
./case.submit
```

Check for successful completion

(12) When the job finishes, confirm that it completed successfully:

```
tail CaseStatus
```

You should see something like the following:

```
2019-02-03 15:22:27: model execution starting
 ----------------------------------------------------
2019-02-03 15:22:47: model execution success
 ----------------------------------------------------
2019-02-03 15:22:47: case.run success
 ----------------------------------------------------
2019-02-03 15:22:52: st_archive starting
 ----------------------------------------------------
2019-02-03 15:22:53: st_archive success
 ----------------------------------------------------
```

Exercise 3.2:
Source code changes
to set up experimental simulation

Create a new git clone

We recommend creating a new git clone before making source code changes to avoid interfering with any ongoing cases from your previous clone.

(1) Create a fresh clone

```
cd ~/practical3/
git clone -b release-clm5.0 https://github.com/ESCOMP/ctsm.git clm5.0_ozone_mods
cd clm5.0_ozone_mods
./manage_externals/checkout_externals
```

Before moving on to the code changes on the next slide, feel free to browse through the contents of the src/ subdirectory to get a general feel for its contents. You can also come back and do this later, such as while waiting for the model to build.

Edit OzoneMod.F90

(2) Open src/biogeophys/OzoneMod.F90 in an editor and add a line:
On line 377, add the following:

```
o3coefgsun(p) = o3coefgsun(c)^3._r8
```

Reminder: *You might notice some issues with this new line. Play along: we'll fix them together.*

The next slide shows this change in context, with some helpful notes about the code in this loop.

Save and exit your editor, then confirm the change by typing the following git command (type `q` to exit the diff):

```
git diff
```

Edit OzoneMod.F90

We're in a loop over a patch (p) filter

This loop also sets the column (c) index associated with each patch

Code duplication removed via repeated call to a subroutine that does all the work

```
do fp = 1, num_exposedvegp
   p = filter_exposedvegp(fp)
   c = patch%column(p)

   ! Ozone stress for shaded leaves
   call CalcOzoneStressOnePoint( &
        forc_ozone=forc_ozone, forc_pbot=forc_pbot(c), forc_th=forc_th(c), &
        rs=rssha(p), rb=rb(p), ram=ram(p), &
        tlai=tlai(p), tlai_old=tlai_old(p), pft_type=patch%itype(p), &
        o3uptake=o3uptakesha(p), o3coefv=o3coefvsha(p), o3coefg=o3coefgsha(p))

   ! Ozone stress for sunlit leaves
   call CalcOzoneStressOnePoint( &
        forc_ozone=forc_ozone, forc_pbot=forc_pbot(c), forc_th=forc_th(c), &
        rs=rssun(p), rb=rb(p), ram=ram(p), &
        tlai=tlai(p), tlai_old=tlai_old(p), pft_type=patch%itype(p), &
        o3uptake=o3uptakesun(p), o3coefv=o3coefvsun(p), o3coefg=o3coefgsun(p))
   o3coefgsun(p) = o3coefgsun(c)^3._r8
   tlai_old(p) = tlai(p)

end do
```

Line to be added

Edit OzoneMod.F90

`o3coefgsun` is a multiplier of stomatal conductance. It varies from 0 to 1, with 1 meaning no effect and 0 shutting down stomatal conductance. The intent of this change is to make ozone's effect on stomatal conductance much more extreme, just for sunlit leaves. With this change, for example, an effect of 0.5 will get turned into 0.5^3 = 0.125.

Take a moment to think about what impact you expect this change to have on canopy transpiration when comparing a new case with this change to your control case (which enabled ozone stress, but at its standard level rather than this more extreme level). **See the next slide for the answer.**

Expected impact on canopy transpiration

We are making the effect of ozone on stomatal conductance more extreme, thus decreasing conductance further than in the control run. A decrease in stomatal conductance should lead to a decrease in canopy transpiration. So we expect to see lower stomatal conductance in our experimental run compared with the control run.

It's always a good idea to go through a thought exercise like this before running the code with modifications, so you are prepared to examine the results with a critical eye. (When I haven't already formed a hypothesis like this, I find it's too easy to just quickly look at the model output and convince myself that it's working correctly, when really it may not be.)

However, for a real project, the "eyeball sanity check" that we'll do here should NOT be your only means for verifying your changes. You should do additional, careful checks by comparing the results with hand calculations for a few points in one timestep, and/or adding unit tests, and/or other, similar methods.

Create a new case

(3) Create a new case

```
cd ~/practical3/clm5.0_ozone_mods/cime/scripts
./create_newcase --case ~/clm_tutorial_cases/ozone_expt --compset I2000Clm50SpGs
--res f45_g37 --run-unsupported --project UCGD0004
```

(4) Do the initial case setup

```
cd ~/clm_tutorial_cases/ozone_expt
./xmlchange JOB_QUEUE=R4231261 --subgroup case.run --force
./xmlchange JOB_WALLCLOCK_TIME=00:05:00
./case.setup
```

(5) Copy over your previous namelist settings

```
cp ~/clm_tutorial_cases/ozone_control/user_nl_clm user_nl_clm
cat user_nl_clm
./preview_namelists
```

Build the model

(6) Build the model:

```
qcmd -q R4231261 -- ./case.build
```

***If you had an account on cheyenne before the tutorial**, make sure the PBS_ACCOUNT is set to UCGD0004 before you build.*

Did the build complete successfully? See the next slide for some discussion

Build failure

You should see output like this:

```
Building lnd with output to /glade/scratch/sacks/ozone_expt/bld/lnd.bldlog.190204-154438
/gpfs/u/home/sacks/practical3/clm5.0_ozone_mods/src/biogeophys/OzoneMod.F90(377): error #5078: Unrecognized
token '^' skipped


/gpfs/u/home/sacks/practical3/clm5.0_ozone_mods/src/biogeophys/OzoneMod.F90(377): error #5082: Syntax error,
found REAL_KIND_CON '3.' when expecting one of: .EQV. .NEQV. .XOR. .OR. .AND. .LT. < .LE. <= .EQ. == .NE. /=
.GT. > ...


/gpfs/u/home/sacks/practical3/clm5.0_ozone_mods/src/biogeophys/OzoneMod.F90(377): error #6385: The highest data
type rank permitted is INTEGER(KIND=8).   [O3COEFGSUN]


/gpfs/u/home/sacks/practical3/clm5.0_ozone_mods/src/biogeophys/OzoneMod.F90(377): error #6385: The highest data
type rank permitted is INTEGER(KIND=8).   [3.]


Component lnd build complete with 2 warnings
clm built in 121.780889 seconds
ERROR: BUILD FAIL: clm.buildlib failed, cat /glade/scratch/sacks/ozone_expt/bld/lnd.bldlog.190204-154438
```

Look at the first reported error. Do you see what the problem is? (The rest of the errors are misleading: as often happens, the compiler has gotten confused from the first error.)

Sometimes the build error doesn't get printed like this, and you'll need to scroll through the lnd bldlog to find the first error (`/glade/scratch/sacks/ozone_expt/bld/lnd.bldlog.190204-154438` in this case).

Fix the build error and submit the run

(7) Open ~/practical3/clm5.0_ozone_mods/src/biogeophys/OzoneMod.F90 in an editor and edit the line you added (line 377): Change ^ to **:

```
o3coefgsun(p) = o3coefgsun(c)**3._r8
```

Save and exit your editor

(8) Rebuild the model. (This should go quickly, because the build can pick up where it left off.)

```
cd ~/clm_tutorial_cases/ozone_expt
qcmd -q R4231261 -- ./case.build
```

This time, you should see the following, indicating that the build completed successfully:

```
MODEL BUILD HAS FINISHED SUCCESSFULLY
```

(9) Submit the run:

```
./case.submit
```

Check for successful completion

(10) When the job finishes, check whether it completed successfully:

```
tail CaseStatus
```

Due to the nature of this error, your results may vary, but I got a model crash:

```
2019-02-04 16:37:31: case.run error
ERROR: RUN FAIL: Command 'mpiexec_mpt  -np 180 -p "%g:"  omplace -tm
open64  /glade/scratch/sacks/ozone_expt/bld/cesm.exe  >> cesm.log.$LID 2>&1 ' failed
See log file for details: /glade/scratch/sacks/ozone_expt/run/cesm.log.4233830.chadmin1.190204-163712
```

If yours ran to completion, you can see my results here:
`/glade/scratch/sacks/ozone_expt_run_saved`

As the above message indicates, you can check the cesm.log file for details. However, I usually start by checking the lnd.log file, and sometimes some of the other component log files: these component log files contain output from just the master processor of each component, and are easier to read than the cesm.log file, which contains output from all processors, sometimes interleaved.

(Continued on next slide.)

Look into failure

(11) Go into your run directory and look at log files

```
cd /glade/scratch/$USER/ozone_expt/run
ls -lrt
tail lnd.log....
```

Fill in this log file name with the actual names of your log files.

Nothing seems amiss for me there (again, your results may vary). So let's check the full cesm.log. Open your cesm.log file in an editor or view it with `less`, then search for the first occurrence of "ERROR".

I see the following:

```
128: ENDRUN:
128: ERROR: BandDiagonal ERROR: dgbsv returned error code
```

along with a bunch of matrix elements, some of which are NaN (not a number). The "128:" prefix is the processor ID that printed that message.

Hmmmm, this doesn't really help pin down the problem....

Exercise 3.3:
Retry the last case in DEBUG mode

Overview

When a run crashes or gives garbage results and the cause isn't obvious, it is often a good idea to rebuild and rerun in DEBUG mode. This turns on additional error-checking for issues such as division by zero and trying to access array elements outside the bounds of the array.

In fact, it is important to do a short (e.g., 5-day) run in DEBUG mode whenever you make code changes, before starting a long production run, even if it looks like things are working right. This can help catch bugs that may otherwise go unnoticed.

However, note that running in DEBUG mode is much more expensive, so you shouldn't run long simulations this way.

You can rebuild an existing case in DEBUG mode, but you first need to fully clean the existing build. In many cases it's just as easy to create a new case, and that's what we'll do here.

Create a new case from your modified code

(1) Create a new case

```
cd ~/practical3/clm5.0_ozone_mods/cime/scripts
./create_newcase --case ~/clm_tutorial_cases/ozone_expt_debug --compset
I2000Clm50SpGs --res f45_g37 --run-unsupported --project UCGD0004
```

(2) Do the initial case setup

```
cd ~/clm_tutorial_cases/ozone_expt_debug
./xmlchange JOB_QUEUE=R4231261 --subgroup case.run --force
./xmlchange JOB_WALLCLOCK_TIME=00:05:00
./case.setup
```

(3) Copy over your previous namelist settings

```
cp ~/clm_tutorial_cases/ozone_control/user_nl_clm user_nl_clm
cat user_nl_clm
./preview_namelists
```

Build the model

(4) Change a setting in *env_build.xml:*

```
./xmlchange DEBUG=TRUE
```

(5) Build the model:

```
qcmd -q R4231261 -- ./case.build
```

***If you had an account on cheyenne before the tutorial**, make sure the PBS_ACCOUNT is set to UCGD0004 before you build.*

You should see the following, indicating that the build completed successfully:

```
MODEL BUILD HAS FINISHED SUCCESSFULLY
```

(6) Submit the run:

```
./case.submit
```

Look into failure

(7) When the job finishes, check whether it completed successfully:

```
tail CaseStatus
```

You should see something like this:

```
2019-02-05 16:18:21: case.run error
ERROR: RUN FAIL: Command 'mpiexec_mpt  -np 180 -p "%g:"  omplace -tm open64
/glade/scratch/sacks/ozone_expt_debug/bld/cesm.exe  >> cesm.log.$LID 2>&1 ' failed
See log file for details: /glade/scratch/sacks/ozone_expt_debug/run/cesm.log.4240878.chadmin1.190205-161750
```

Open the referenced cesm.log file in an editor or view it with less. Then either scroll to the bottom or search for the first instance of “severe”.

Spend a minute thinking about what you’re seeing, then go on to the next slide.

Look into failure

You should see something like this in your cesm.log file (actually, many instances of this – one for each processor):

```
87:forrtl: severe (408): fort: (3): Subscript #1 of the array O3COEFGSUN has value 3683 which is less than the
lower bound of 5647
87:
87:Image              PC                Routine            Line        Source
87:cesm.exe           0000000004197046  Unknown               Unknown  Unknown
87:cesm.exe           0000000001A104FE  ozonemod_mp_calco         377  OzoneMod.F90
87:cesm.exe           00000000011E9D33  canopyfluxesmod_m        1300  CanopyFluxesMod.F90
87:cesm.exe           0000000000884909  clm_driver_mp_clm         543  clm_driver.F90
87:cesm.exe           000000000084711A  lnd_comp_mct_mp_l         456  lnd_comp_mct.F90
87:cesm.exe           0000000000461C2D  component_mod_mp_         728  component_mod.F90
87:cesm.exe           0000000000430944  cime_comp_mod_mp_        2712  cime_comp_mod.F90
87:cesm.exe           00000000004495DC  MAIN__                    125  cime_driver.F90
```

The first line tells you the problem, and the following lines give what is known as a stack trace, showing where the error occurred (near the top), then where that subroutine was called from, then where *that* subroutine was called from, and so on. Unsurprisingly, the error occurred on the new line that you added.

Reopen ~/practical3/clm5.0_ozone_mods/src/biogeophys/OzoneMod.F90 and look at line 377 to see if you can identify the problem, then move on to the next slide.

Hint

Hint: o3coefgsun is a patch-level array (just above the loop, you can see that it is aliased to o3coefgsun_patch). How do you see this array being indexed in other parts of this subroutine?

See the next slide for the answer.

Fix the bug

Answer: Since o3coefgsun is a patch-level array, it should be indexed by a patch-level index (typically “p”), not a column-level index (typically “c”): column-level indices are completely different from patch-level indices.

(8) Open ~/practical3/clm5.0_ozone_mods/src/biogeophys/OzoneMod.F90 in an editor and edit the line you added (line 377): Change `(c)` to `(p)`:

```
o3coefgsun(p) = o3coefgsun(p)**3._r8
```

Save and exit your editor

(8) Rebuild the model. (This should go quickly, because it just needs to rebuild the changed file and anything that depends on it.)

```
cd ~/clm_tutorial_cases/ozone_expt_debug
qcmd -q R4231261 -- ./case.build
```

(9) Resubmit the run:

```
./case.submit
```

(10) When the job finishes, confirm that it completed successfully (check the CaseStatus file)

Think about your non-DEBUG run

So, what do you think happened in your previous, non-DEBUG run?

See the next slide for the answer.

Think about your non-DEBUG run

When you try to access an array element outside the bounds of an array, if you haven't turned on DEBUG-mode checks, you'll get whatever value happens to be in memory in the location that *would have* contained that array element, *if* the array had been big enough. In most cases, this means you'll get some garbage value from some other, totally unrelated variable!

If you're "lucky", this will cause the model to crash. But in some cases, the model will continue to run and will just give incorrect results. Sometimes this will be obvious, but you can't count on that. **This is why it is important to test your code changes in DEBUG mode.**

Bonus Exercise 3.4 (if you have time):
Rebuild and rerun the non-DEBUG case, compare with the control case

Overview

We will now return to the non-DEBUG experimental case – the one that died with a cryptic error. We'd like to compare the output from this case with your original control case as a sanity check that your code change is working approximately as expected.

You shouldn't use the DEBUG case for this comparison, because the code gives different answers when run in DEBUG vs. non-DEBUG mode: compiler optimizations in non-DEBUG mode can lead to roundoff-level differences in floating point operations. Although these differences should start off very small, they can grow over time due to the nonlinearity of the model.

Rebuild and resubmit the non-DEBUG case

(1) Rebuild the non-DEBUG case. Since this case was created from the same code directory as the DEBUG case, the rebuild will pick up the changes you made to fix the DEBUG case.

```
cd ~/clm_tutorial_cases/ozone_expt
qcmd -q R4231261 -- ./case.build
```

(2) Resubmit the run:

```
./case.submit
```

(3) When the job finishes, confirm that it completed successfully (check the CaseStatus file)

Compare output with your control case

(4) Create a difference file: experiment *minus* control:

```
cd /glade/scratch/$USER/archive/ozone_expt/lnd/hist
module load nco
ncdiff ozone_expt.clm2.h0.0001-01-01-00000.nc
/glade/scratch/$USER/archive/ozone_control/lnd/hist/ozone_control.clm2.h0.0001-01-01-00000.nc diffs.nc
```

(5) View the differences:

```
module load ncview
ncview diffs.nc &
```

Click on the button labeled “3d vars” and select FCTR (canopy transpiration).

Change the range to be symmetrical about zero by clicking on the “Range” button above the color bar. Change the minimum to -10 and the maximum to 10.

Change the color bar by clicking on the leftmost (“3gauss”) button above the color bar. Continue clicking until you find a good color bar for a difference plot (e.g., “blu_red”).

Scroll through time by clicking on the button labeled “1-Jan-0001”.

You should mainly see negative values – i.e., lower transpiration in the experimental case, where we made the ozone effect more extreme. This is what we expected. Does it make sense to you that there is little difference in the middle and high latitudes of the Northern Hemisphere?

Bonus Exercise 3.5 (if you have time): Adding a history field

i.e., including a new variable in the model output

Edit OzoneMod.F90

(1) Open ~/practical3/clm5.0_ozone_mods/src/biogeophys/OzoneMod.F90 in an editor and add the following block of code in the subroutine InitHistory (around line 217):

This is needed for history fields that remain uninitialized (NaN) for some points; to be safe, we do this for all history fields. NaN values cannot be output, so we instead ensure that any unused points are set to spval ("special value") for fields that are output to history files. spval is a constant that signals to the history averager that that point should be excluded from averaging.

```
this%o3coefgsun_patch(begp:endp) = spval
call hist_addfld1d (fname='O3COEFGSUN', units='unitless', &
     avgflag='A', long_name='ozone coefficient for conductance, sunlit leaves (0 - 1)', &
     ptr_patch=this%o3coefgsun_patch)
```

ptr_patch is used for patch-level variables, ptr_col for column-level, etc.

The various strings here (fname, units and long_name) can be whatever you want – they have no special meaning.

Produce time averages. This is by far the most common. Other options are instantaneous, maximum over time, etc.

Note that, unless you specify otherwise (with the "default" optional argument to hist_addfld1d), your new history field will automatically appear on the h0 history files.

Rebuild, resubmit, and check output

(2) Rebuild the non-DEBUG case:

```
cd ~/clm_tutorial_cases/ozone_expt
qcmd -q R4231261 -- ./case.build
```

(3) Resubmit the run:

```
./case.submit
```

(4) Look for your new field in the output:

```
cd /glade/scratch/$USER/archive/ozone_expt/lnd/hist
module load ncview
ncview ozone_expt.clm2.h0.0001-01-01-00000.nc &
```

Click on the button labeled “3d vars” and select O3COEFGSUN. Scroll through times, making sure it looks like it has valid data (between 0 and 1).