CCPP Training
College Park, MD, March 12-13, 2019

Physical Schemes and Interstitials

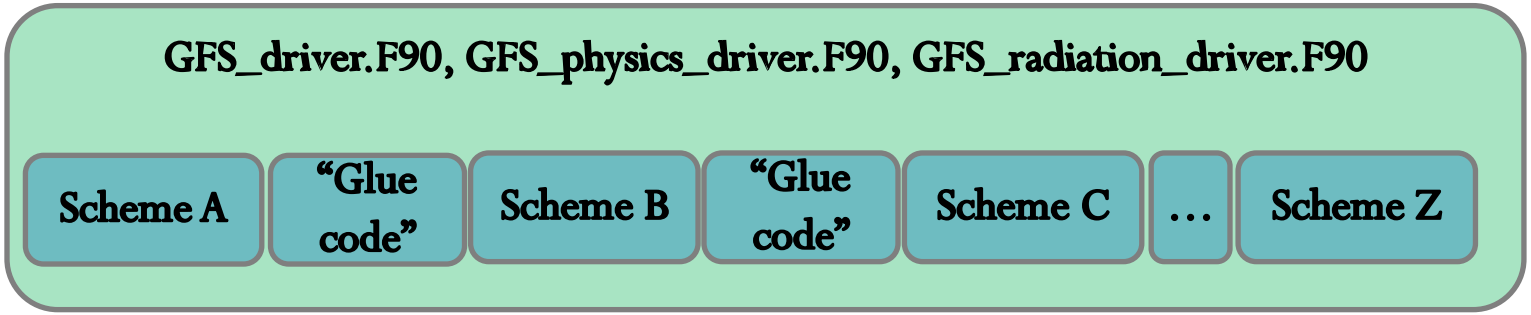
Grant Firl
Global Model Test Bed

Outline of Talk

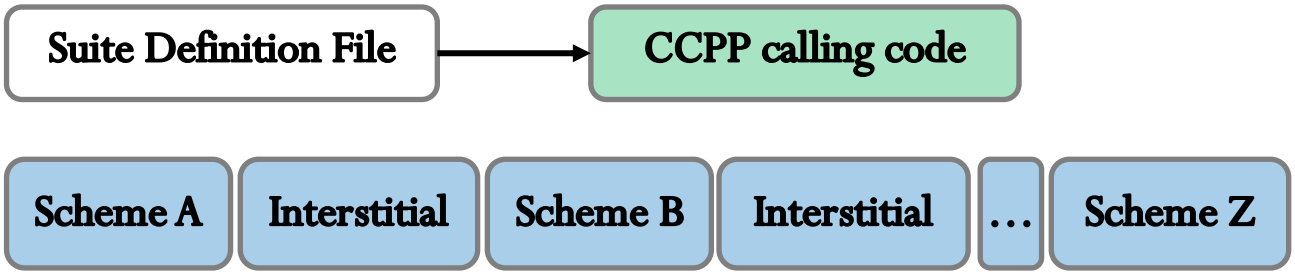
- Big Picture
- Where did all the drivers go?
- What constitutes a scheme?
- Physical parameterizations vs interstitial schemes
- Overview of changes to physics

Big Picture

- Hard-coded "suite-drivers" are called in current infrastructure:



- Suite Definition File used by CCPP infrastructure to autogenerate the equivalent of suite drivers



Disappearing Drivers

- IPDv4 within FV3 makes use of 3 drivers: GFS_driver, GFS_radiation_driver, and GFS_physics_driver
- How does this structure change with the CCPP that aims to eliminate hard-coded suite-based physics drivers?
- In the CCPP world, all physics-related code must be callable using a CCPP-compliant interface
  - CCPP requires that all of the function provided by these drivers is placed into CCPP-compliant physics "schemes"

What Constitutes a CCPP "Scheme"?

- Any piece of code with a CCPP-compliant interface:
  - Code must be wrapped within a Fortran module
  - Must contain init, run, and finalize subroutines
  - Must contain CCPP-readable metadata describing argument variables for all subroutines (init/run/finalize)
  - Use CCPP error-tracking variables rather than printing/stopping
  - Have formatted scientific/technical documentation
  - Conform to modern coding standards
- Scheme independence
  - schemes should comprise the smallest functional unit possible
    - if scheme functions will always be called together, OK to keep as one
    - if scheme functions will operate independently, separate the schemes

Types of Schemes

- Physical parameterization entry/exit point
  - A subroutine that is exposed to the CCPP framework and serves as a public interface for the underlying physical parameterization code for exchanging information with a calling application
- "Interstitial" schemes - modularized data preparation, diagnostics, and "glue code" that allows schemes to function together as a suite
  - Can specifically augment an physical parameterization (scheme-specific)
    - additional functionality beyond what scheme code provides, but tied to one specific physical parameterization
  - Can be suite-level
    - contains functionality that is not tied to one specific scheme but may:
      - provide functionality on top of a class of schemes
      - connects two or more schemes together ("glue code")
      - conversions, initializing sums, applying tendencies

Code Changes

- CCPP Physics Code Changes Documentation
  - Describes changes to contents of NEMSfv3gfs/FV3/gfsphysics for transition to CCPP
  - What happened to:
    - Existing physics schemes and their dependencies
    - New physics schemes (coming from outside EMC)
    - Existing "interstitial" code from the GFS suite drivers
    - New interstitial code (associate with new schemes)
    - Stochastic physics code
  - Also contains lists of
    - files that have not been moved over (schemes/dependencies that are not yet CCPP-compliant)
    - new testing/helper routines
    - non-FV3 files (related to SCM)

Code Changes

- Existing Physics Scheme Example
  - Hybrid EDMF PBL scheme
    - NEMSfv3gfs/FV3/gfsphysics/physics → NEMSfv3gfs/ccpp/physics/physics
    - wrap subroutine in module
    - create _init, _run, and _finalize routines
    - create CCPP-readable metadata table for all arguments of CCPP-exposed subroutines (scheme entry/exit points)
    - add errmsg and errflg arguments for tracking errors with CCPP infrastructure and initialize them in the subroutines
    - mfpbl.F (dependency) copied over to new directory with no changes
  - For details run diff between moninedmf.f versions

Code Changes

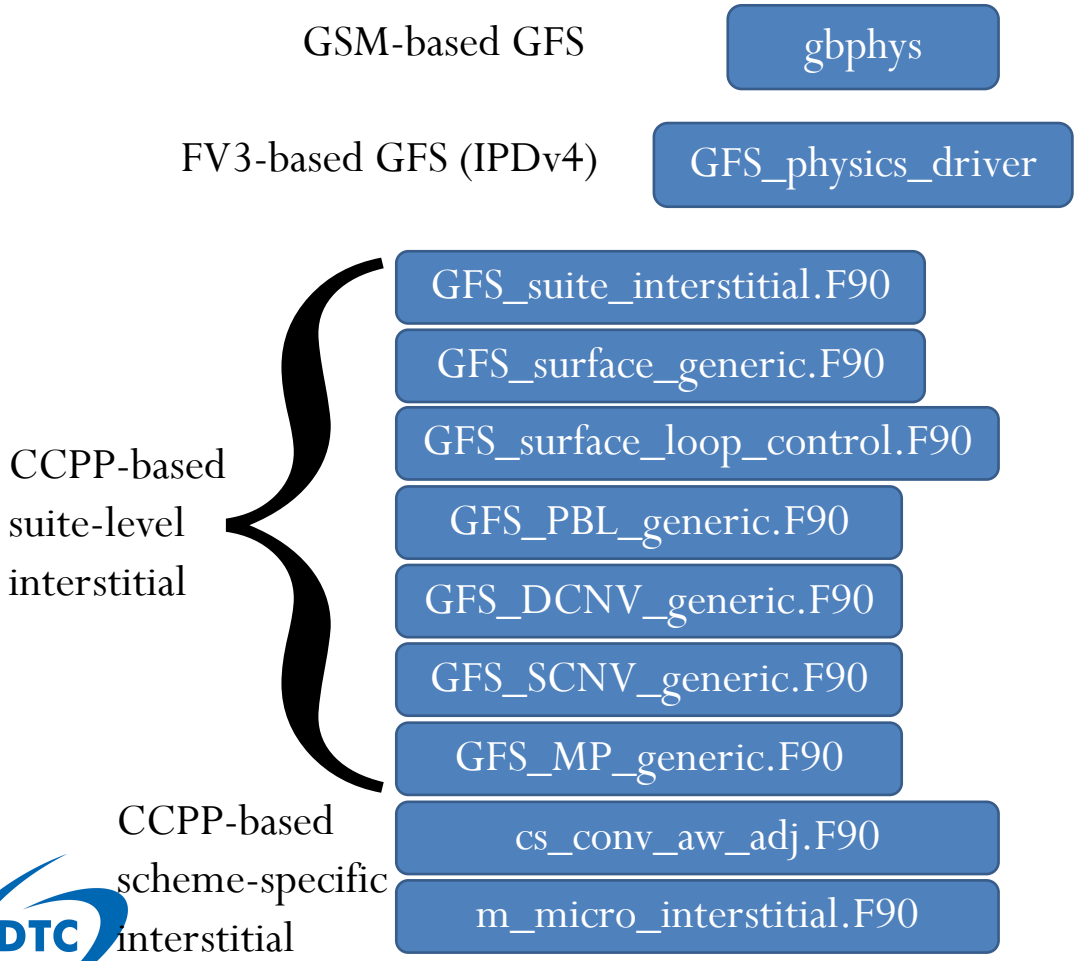
- New Physics Scheme Example
  - MYNN PBL scheme
    - module_MYNNPBL_wrapper.F90
      - conforms to same CCPP-compliancy rules as Hybrid EDMF
      - uses wrapper around existing scheme code rather than put CCPP-compliant interface within scheme algorithm code
    - depends on module_bl_mynn.F90 (main algorithm)
    - scheme also requires additional functionality to hand subgrid-scale cloudiness to radiation scheme (interstitial code – does not belong with either PBL or radiation schemes, but connects the two within the suite)
      - module_MYNNrad_pre.F90
        - contains code to save original resolved-scale clouds before the call to radiation and adds subgrid-scale clouds to the cloud field that the radiation scheme "sees"
      - module_MYNNrad_post.F90
        - restore the original resolved-scale clouds after the call to radiation

Code Changes

- Existing Interstitial Scheme Example
  - GFS_PBL_generic.F90
    - contains code for calculating tendencies, preparing/interacting with vertically-diffused tracer arrays, and calculating other diagnostics related to PBL
    - code used to be immediately before/after calls to PBL schemes in GFS_physics_driver
    - this code does not "belong" within the scheme itself because:
      - not necessary for scheme function, but plays a vital role for the GFS suite
      - may not be necessary as part of another suite if the same diagnostics are not required or if vertically diffused tracer array is prepared elsewhere in the host model
    - replacing/modifying a physics scheme requires checking for any necessary changes in these schemes too!

Code Changes

- Keeping your eye on the ball…

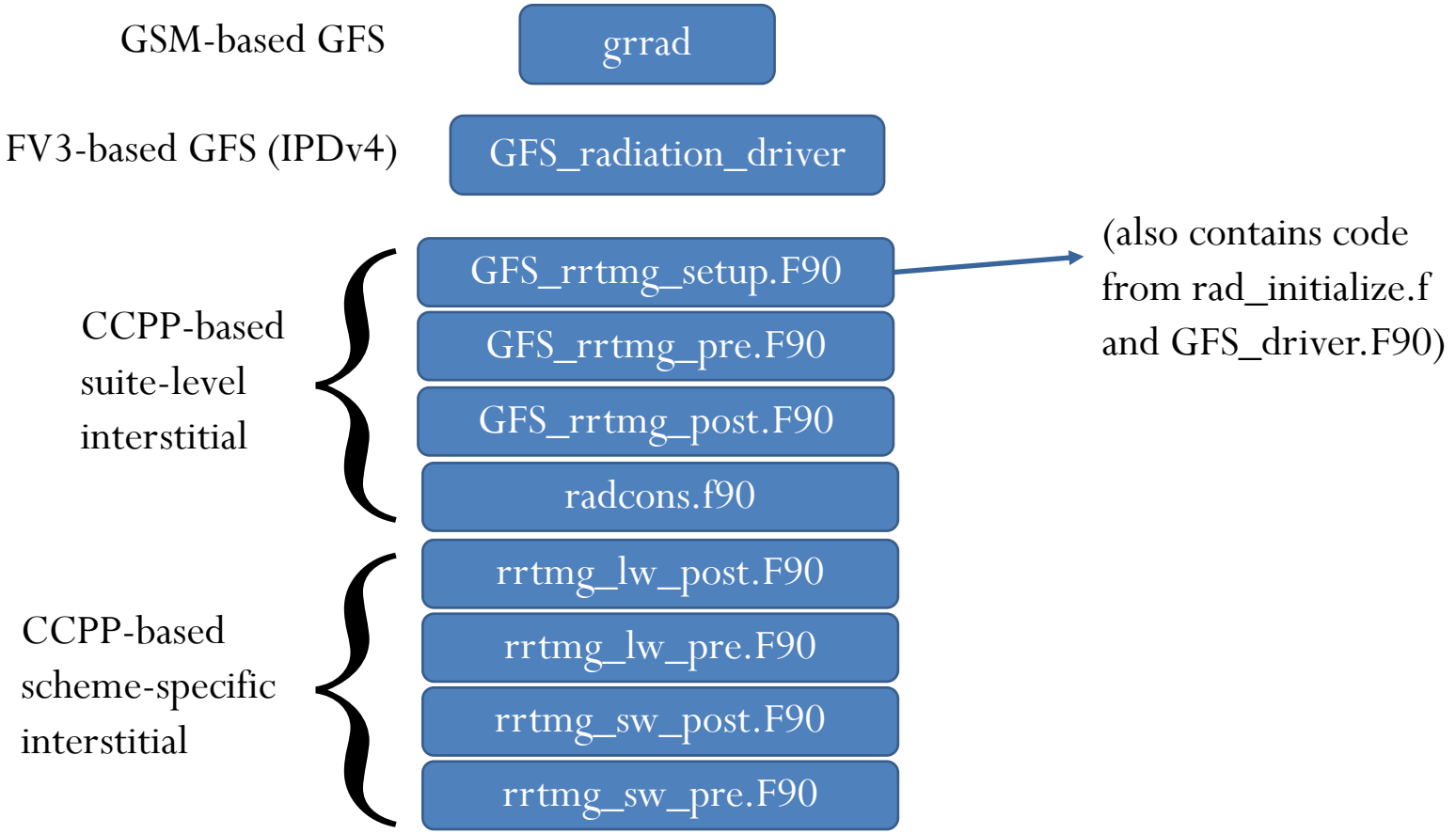


Several sections of scheme-specific interstitial code moved into scheme source code files:

- sfc_nst_pre/post
- dcyc2t3_post
- sfc_diag_post
- gwdps_pre/post
- gwdc_pre/post
- samfshalcnv_post

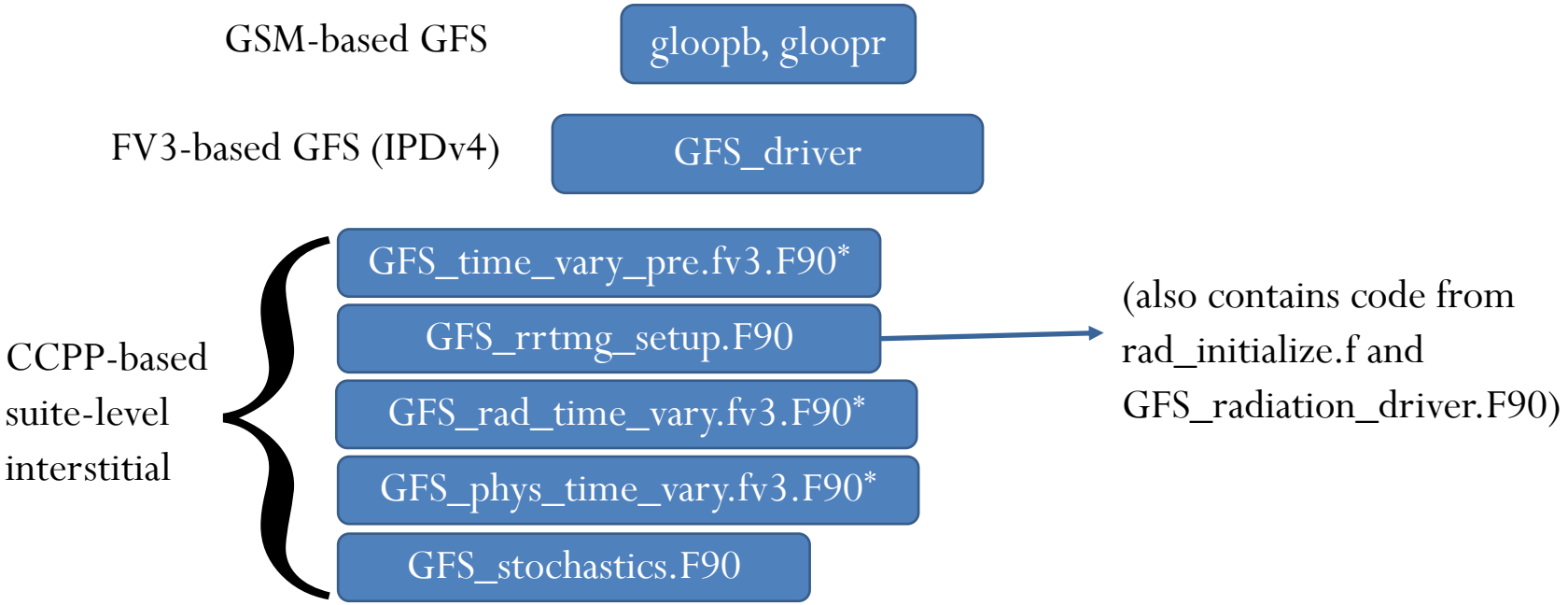
Code Changes

- Keeping your eye on the ball…



Code Changes

- Keeping your eye on the ball…

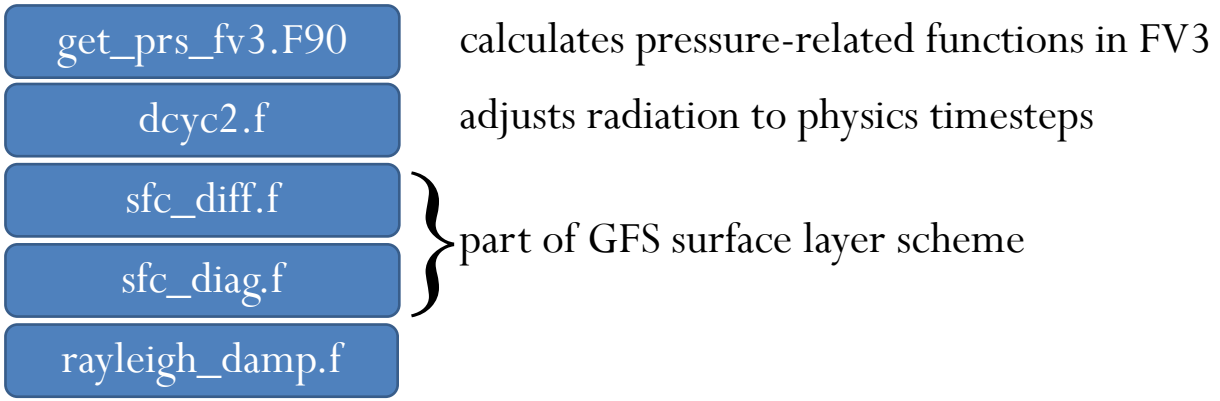


- Physics initialization code in GFS_driver.F90/GFS_initialize moved to _init subroutines within their respective schemes.

* these files are host-model dependent since they are run over the entire domain and depend on how the host model handles ozone, aerosols, etc.

Code Changes

- Keeping your eye on the ball…
  - Several subroutines previously called through GFS_physics_driver have now been made into "schemes" even though these may not typically be thought of as physics



Wrap Up

- "Suite-drivers" to modular CCPP-based schemes
- What constitutes a scheme within CCPP
  - physical parameterization vs interstitial schemes
- Where did code in previous drivers end up?