2018 Joint WRF/MPAS Users' Workshop, 06/13/2018

Community infrastructure for facilitating improvement and testing of physical parameterizations: the Common Community Physics Package (CCPP)

Dom Heinzeller[1,3], Ligia Bernardet[1,3], Grant Firl[2,3], Laurie Carson[2,3], Don Stark[2,3], Man Zhang[1,3], Jimy Dudhia[2], Dave Gill[2]

[1]CU/CIRES at NOAA/ESRL Global Systems Division

[2]National Center for Atmospheric Research

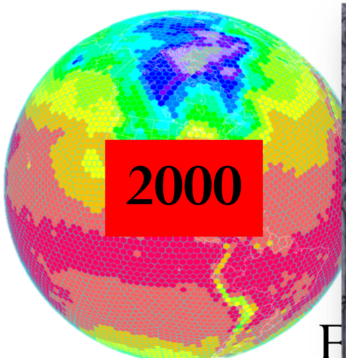
[3]Developmental Testbed Center

Representing many contributors

- GMTB (Tim Brown, Chris Harrop, Gerard Ketefian, Pedro Jimenez, Julie Schramm, Lulin Xue)
- EMC (V. Tallapragada, M. Iredell), GFDL (R. Benson)
- ESPC PI (Jim Doyle and the group)

Developmental Testbed Center

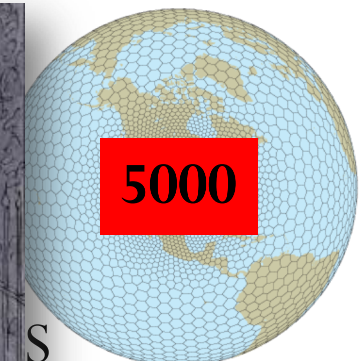
An atmospheric model zoo



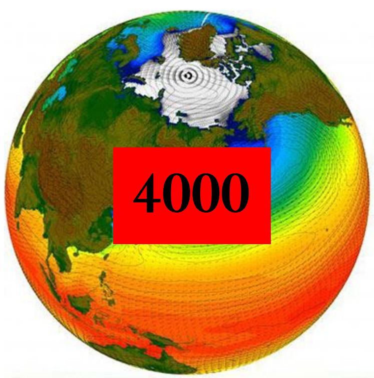
2000

F

S



5000



4000

CESM

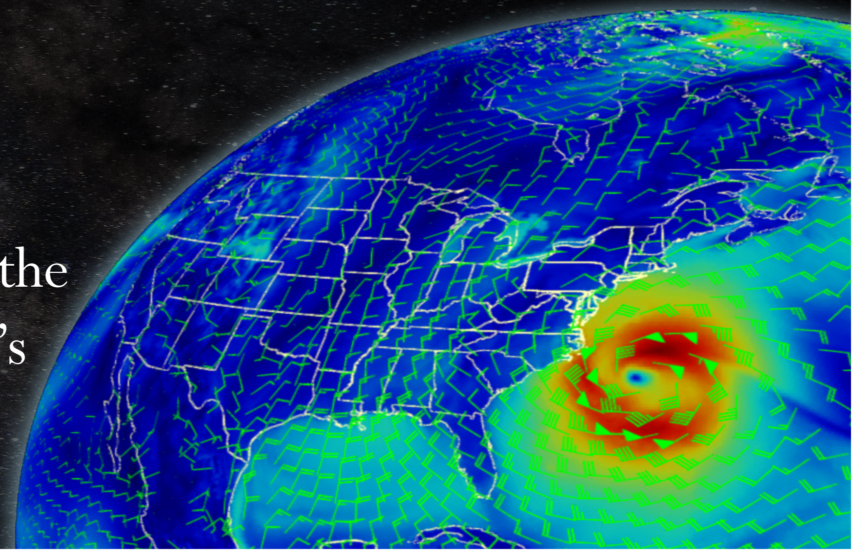
22000

WRF (ARW,NMM)

Lines of c … comments)

Global Model Test Bed (GMTB)

Area within the Developmental Testbed Center (DTC) created to accelerate transition of physics developments by the community onto NOAA's Unified Forecast System



Approach

- Infrastructure for development of parameterizations/suites
- Development of hierarchical physics testbed
- Assessment of physics innovations

See also poster P2 by Ligia Bernardet et al.

Courtesy Ligia Bernardet

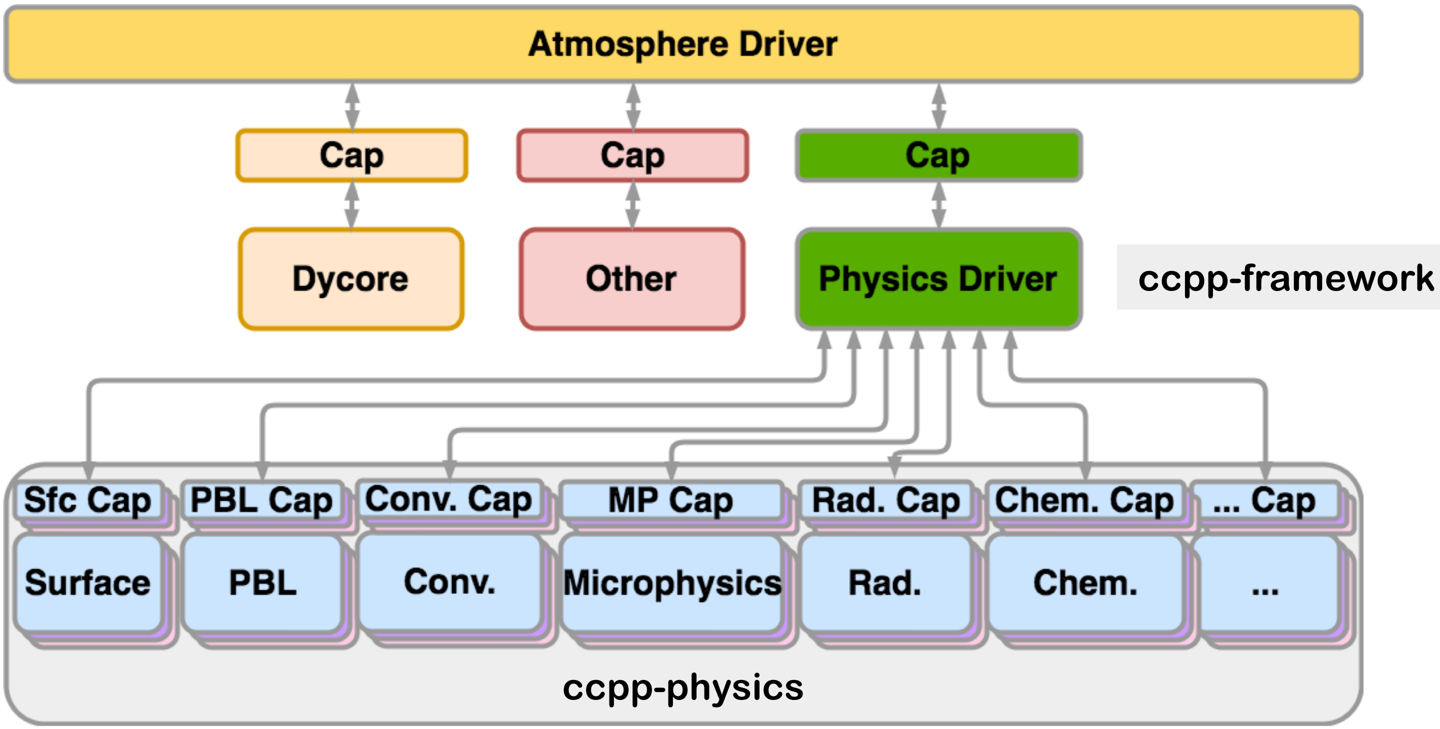
Common Community Physics Package

The Common Community Physics Package (CCPP) consists of an infrastructure component **ccpp-framework** and a collection of compliant physics suites **ccpp-physics**.

Driving principles:

- Readily available and well supported: open source, on Github, accepting external contributions (review/approval process)
- Model-agnostic to enable collaboration and accelerate innovations
- Documented interfaces (metadata) facilitate using/enhancing existing schemes, adding new schemes or transfer them between models
- Physics suite construct is important, but the CCPP must enable easy interchange of schemes within a suite (need for interstitial code)

CCPP within the model system



- Physics schemes caps: auto-generated from metadata
- Host model cap: "handcrafted", include auto-generated code (CPP)

Key features of the CCPP

- **Runtime configuration**: suite definition file (XML)
- **Ordering**: user-defined order of execution of schemes
- **Subcycling**: schemes can be called at higher frequency than others or than dynamics
- **Grouping**: schemes can be called in groups with other computations in between (e.g. dycore, coupling)

```
<suite name="GFS_2017">
...
  <group name="radiation">
  <scheme>GFS_rrtmg_pre</scheme>
  <scheme>rrtmg_sw_pre</scheme>
  <scheme>rrtmg_sw</scheme>
  <scheme>rrtmg_sw_post</scheme>
  <scheme>rrtmg_lw_pre</scheme>
  <scheme>rrtmg_lw</scheme>
  <scheme>rrtmg_lw_post</scheme>
  <scheme>GFS_rrtmg_post</scheme>
  </group>
...
</suite>
```

suite interstitial   scheme interstitial   scheme

A CCPP-compliant physics scheme

```
module scheme_template
      contains

      subroutine scheme_template_init()
      end subroutine scheme_template_init

      subroutine scheme_template_finalize()
      end subroutine scheme_template_finalize

!>\section arg_table_scheme_template_run Argument Table
!!| local_name | standard_name | long_name | units | rank | type      | kind  | intent | optional |
!!|------------|---------------|-----------|-------|------|-----------|-------|--------|----------|
!!| errmsg     | error_message | error msg | none  |    0 | character | len=* | out    | F        |
!!| errflg     | error_flag    | error flg | flag  |    0 | integer   |       | out    | F        |
!!| prs        | air_pressure  | air pres. | Pa    |    2 | real      | phys  | inout  | F        |
!!
      subroutine scheme_template_run(errmsg,errflg,prs)
         implicit none
         character(len=*), intent(  out) :: errmsg
         integer,          intent(  out) :: errflg
         real(kind=phys),  intent(inout) :: prs(:,:)
         ...
      end subroutine scheme_template_run
end module scheme_template
```

Adding a parameterization is easy!

1. Add new scheme to CCPP prebuild configuration (Python)

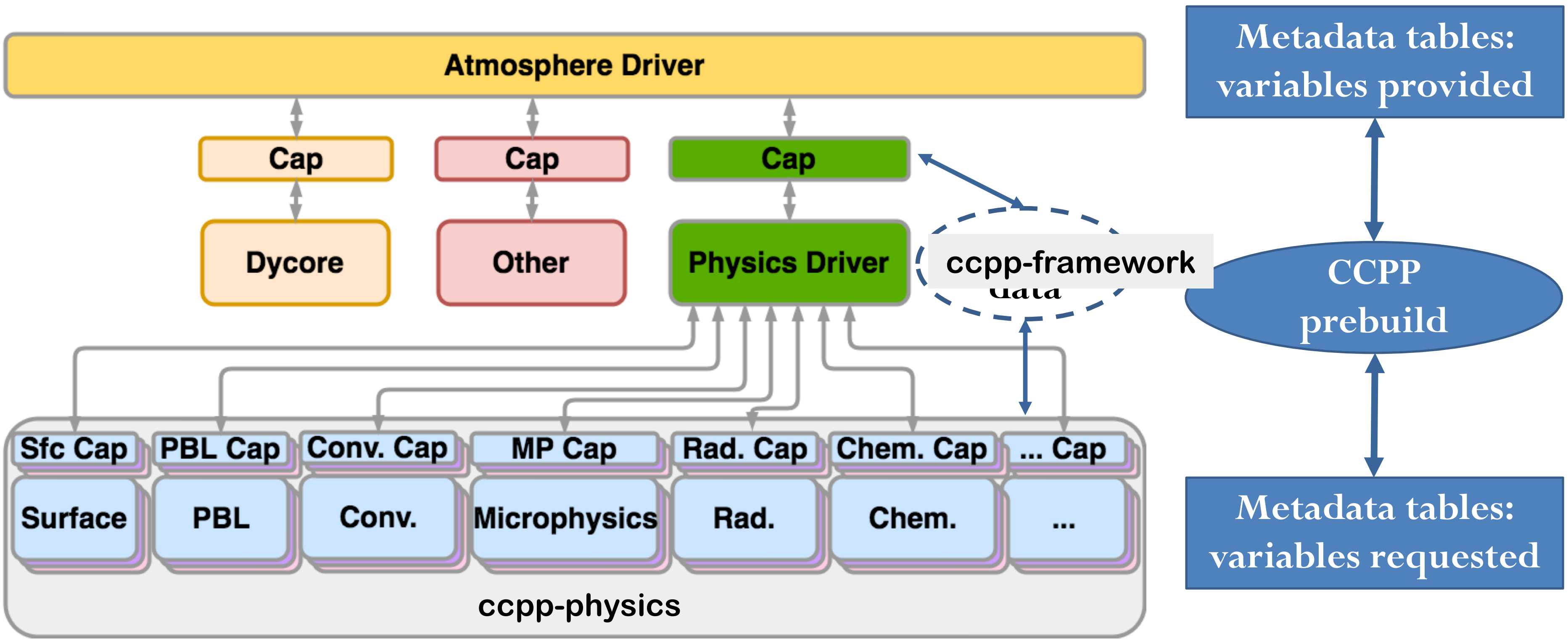
```
scheme_files = [
      "existingscheme.F90",
      "mynewscheme.F90",
      "otherexistingscheme.F90",
      ]
```

2. Compile (CCPP)

3. Add new scheme to suite definition file (also runs init/finalize)

```
      <scheme>existingscheme</scheme>
      <scheme>mynewscheme</scheme>
      <scheme>otherexistingscheme</scheme>
```

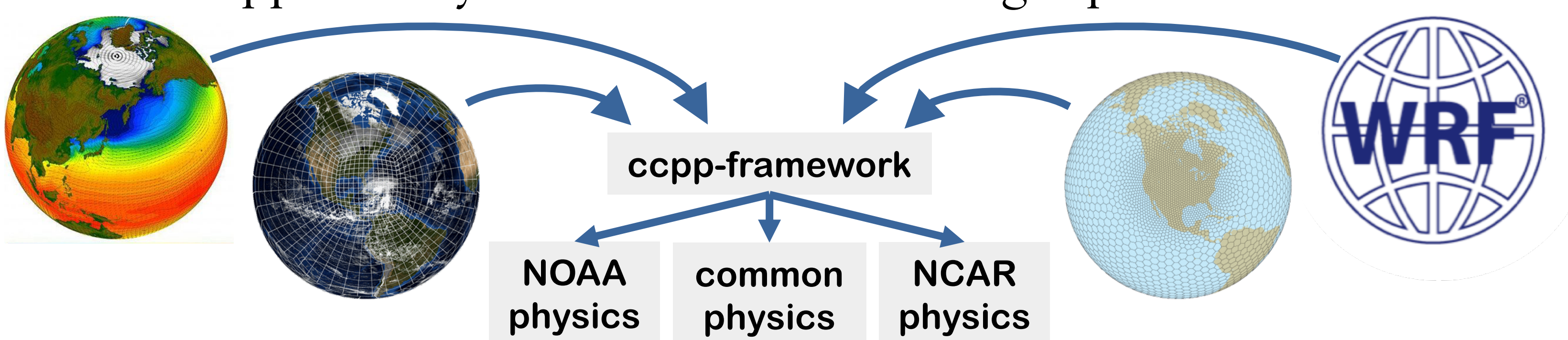
Metadata tables on host model side



ccpp-data: lookup table standard_name → address of variable in memory

CCPP's short past and long future

- First release of CCPP with GMTB Single Column Model in April 2018 (GFS physics), next release in July 2018 (with GFDL microphysics)
- Release with FV3 2018/2019 with 2020 physics candidates
- NOAA and NCAR agreed to collaborate on **ccpp-framework**: enables interoperability of physics between NOAA/NCAR models
  - Metadata updates: vertical direction, index ordering, …
  - Automatic transforms, unit conversions, performance optimization
  - Great opportunity to advance US modeling capabilities!



Bonus material

Side-effect: debugging made easy

Suppose one wants to diagnose a loss in conservation of a specific variable that gets used and modified in many places.

1. Create a new "scheme" writing diagnostic output to screen/file

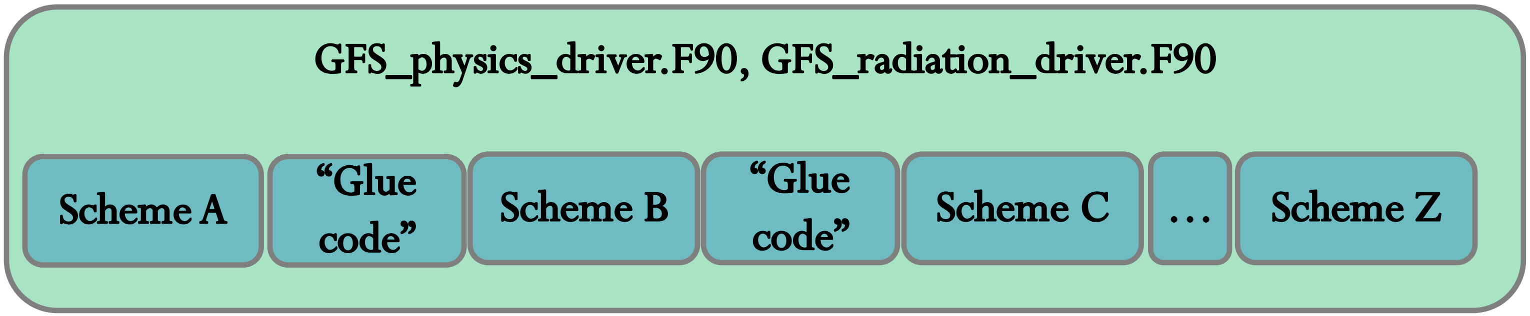
2. Add scheme to relevant places in suite definition file

```
...
<scheme>GFS_examplescheme</scheme>
<scheme>GFS_diagtoscreen</scheme>
...
<scheme>GFS_anotherexamplescheme</scheme>
<scheme>GFS_diagtoscreen</scheme>
...
```

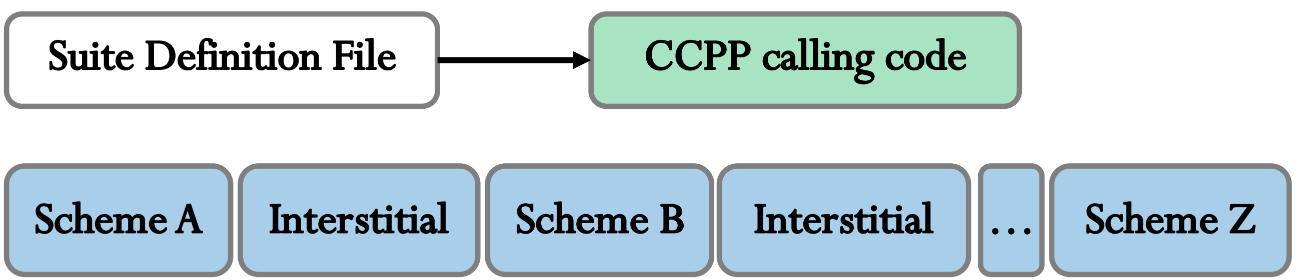
3. No tinkering with host model code (driver, …)!

Interstitital code

- “Suite-drivers” are called in current infrastructure (e.g. FV3):

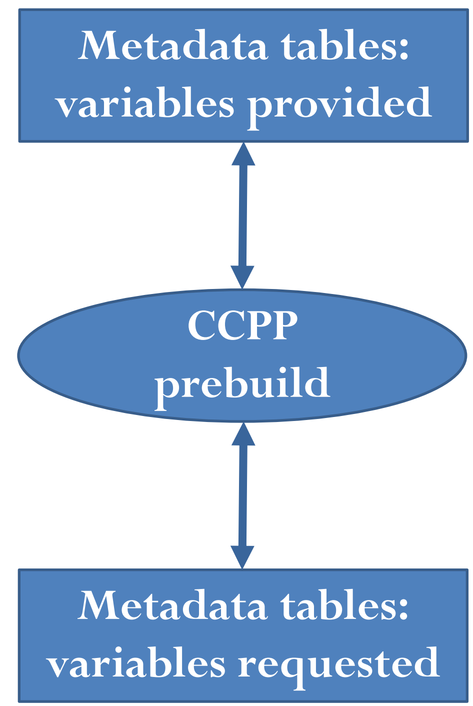


- Suite Definition File instructs CCPP infrastructure to call individual schemes; “interstitial” code within suite drivers ➔ interstitial schemes



Magic behind the scenes

- Python script ccpp_prebuild.py
  - requires metadata tables on both sides
  - checks requested vs provided variables by standard_name
  - checks units, rank, type (more to come)
  - creates Fortran code that adds pointers to the host model variables and stores them in the ccpp-data structure (ccpp_fields.inc)
  - creates caps for physics schemes
  - populates makefiles with schemes and caps



How to hook up CCPP w/ host model

- Python script ccpp_prebuild.py
  - does all the magic before/at build time
- Model developers need to
  - create ccpp_prebuild_MODEL.py config
  - include auto-generated makefiles (and ccpp_prebuild.py) in build system
  - write host model cap that contains ccpp_run calls and include statements for auto-generated code (ccpp_fields.inc)
  - manage memory for cdata structure