Making forecasts better – community infrastructure for facilitating improvement and testing of physical parameterizations

D. Heinzeller[1,3,4], L. Bernardet[1,3,4], G. Firl[2,3], L. Carson[2,3], C. Harrop[1,3,4], P. Jimenez[2],
G. Ketefian[1,3,4], J. Schramm[1,5], D. Stark[2,3], L. Xue[2,3], M. Zhang[1,3,4], D. Gill[2,3]

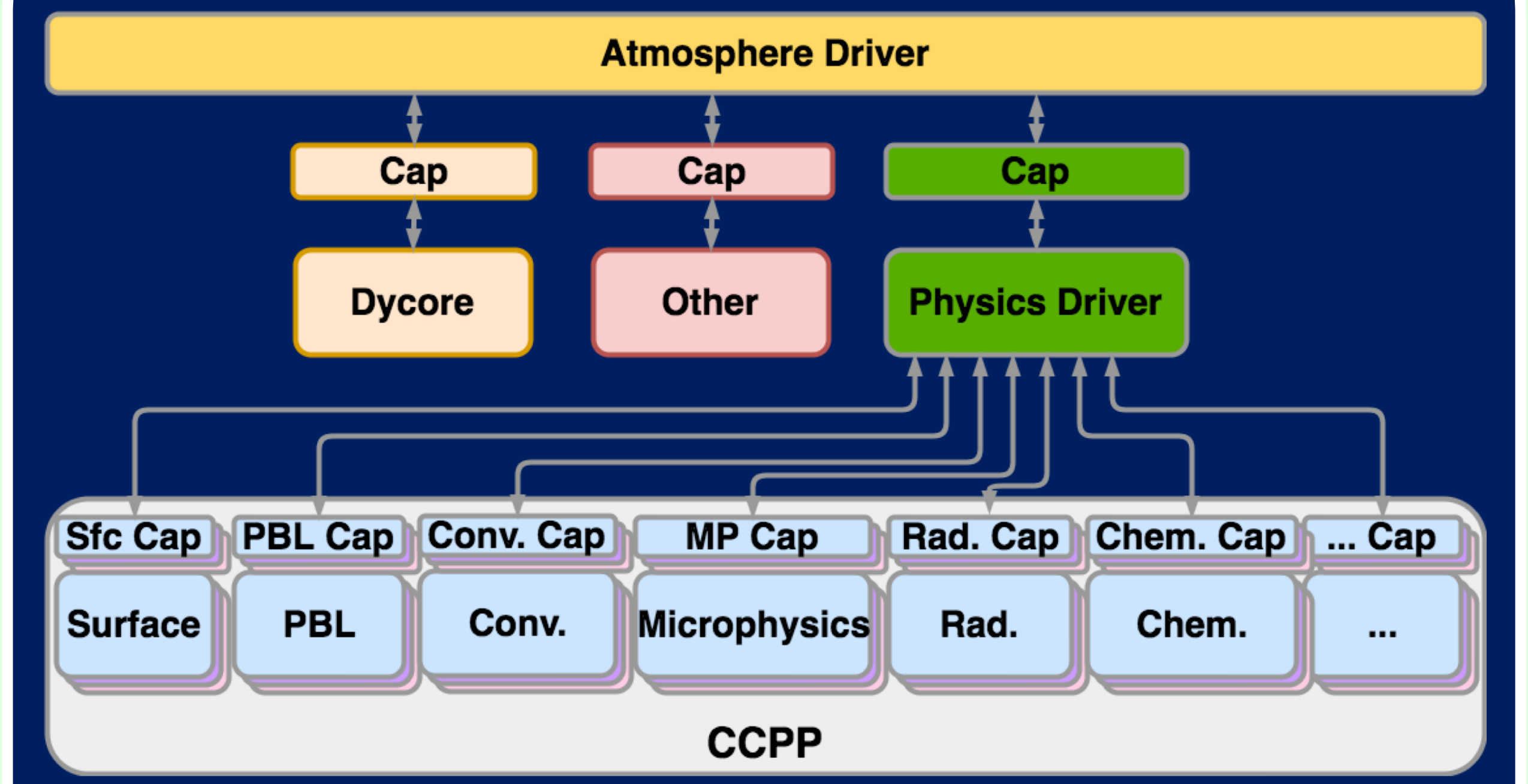
[1] NOAA, Earth System Research Laboratory, Global Systems Division; [2] National Center for Atmospheric Research;
[3] Developmental Testbed Center; [4] University of Colorado Cooperative Institute for Research in Environmental Sciences;
[5] Colorado State University Cooperative Institute for Research in the Atmosphere

About the Global Model Test Bed

GMTB fosters and facilitates community engagement in atmospheric physics by

- providing a physics library and driver that allow distributed development in a model-agnostic setting
- supporting users with porting code to this library
- maintaining and supporting a comprehensive testing platform for the emerging NOAA Unified Forecast System (UFS)
- conducting testing and evaluation of innovations
- bringing together research and operational groups

Common Community Physics Package



- The Common Community Physics Package (CCPP) is made up of two components (repositories), the CCPP physics library **ccpp-physics** and the CCPP physics driver **ccpp-framework**.
- **ccpp-physics** is a collection of **vetted, dycore-agnostic**, physical parameterizations. There can be multiple of each type (PBL, cumulus etc.) to support various applications (high-res, seasonal etc.) and maturity level (operational, developmental).
- **Vetted** means that there is a governance process to determine what is included in CCPP.
- **Dycore agnostic** means that the parameterizations can be used with any dycore through the CCPP driver **ccpp-framework** with caps on both sides.
- **Runtime selection** of parameterizations/suites
- **Configurable order/frequency** of physics calls.
- **User-specified grouping** of schemes, subcycling.

CCPP Status

- Under active developed at NOAA GSD and NCAR RAL
- V1 release March/April 2018 with GMTB SCM v2
- V2 release June/July 2018 with NOAA FV3-GFS v1

Adding a scheme to CCPP

- Write CCPP-compliant scheme (see below)
- Add scheme to list of schemes in CCPP prebuild config, handle optional arguments
- Add scheme to runtime suite definition file
- Done (really!)

Adding CCPP to host model

- Is nearly as easy as adding a new scheme
- Add config for CCPP prebuild (see below)
- Write host model cap to abstract away CCPP calls from dycore (init, run, finalize)
- Add prebuild script & CCPP to build system

A CCPP-compliant physics scheme

```
module myscheme

  contains

  subroutine myscheme_init ()
  end subroutine myscheme_init

  subroutine myscheme_finalize()
  end subroutine myscheme_finalize

!> \section arg_table_myscheme_run Argument Table
!!| local_name  | standard_name  | long_name     | units  | rank | type      | kind  | intent | optional |
!!|-------------|----------------|---------------|--------|------|-----------|-------|--------|----------|
!!| prs         | air_pressure   | air pressure| Pa     |    3 | real      | kind=8| inout  | F        |
!!| rnd         | random_number  | random no     | none   |    3 | integer   | kind=4| in     | F
!!| errmsg      | error_message  | CCPP errmsg   | none   |    0 | character | len=* | out    | F        |
!!| errflg      | error_flag     | CCPP errflg   | flag   |    0 | integer   |       | out    | F        |
!!
      subroutine myscheme_run (prs, geo, errmsg, errflg)

         implicit none
         real(kind=8),      intent(inout) :: prs(:,:,:)
         integer(kind=4),   intent(in)    :: rnd(:,:,:)
         character(len=*),  intent(out)   :: errmsg
         integer,           intent(out)   :: errflg

         !--- initialize intent(out) variables
         errmsg = ''
         errflg = 0

         !--- add your code here

      end subroutine myscheme_run

    end module myscheme
```

module with scheme name

three entry points scheme_{init,run,finalize}

standard metadata table for subroutines in use; parsed by CCPP prebuild script

all information through argument list, no “use external_module”

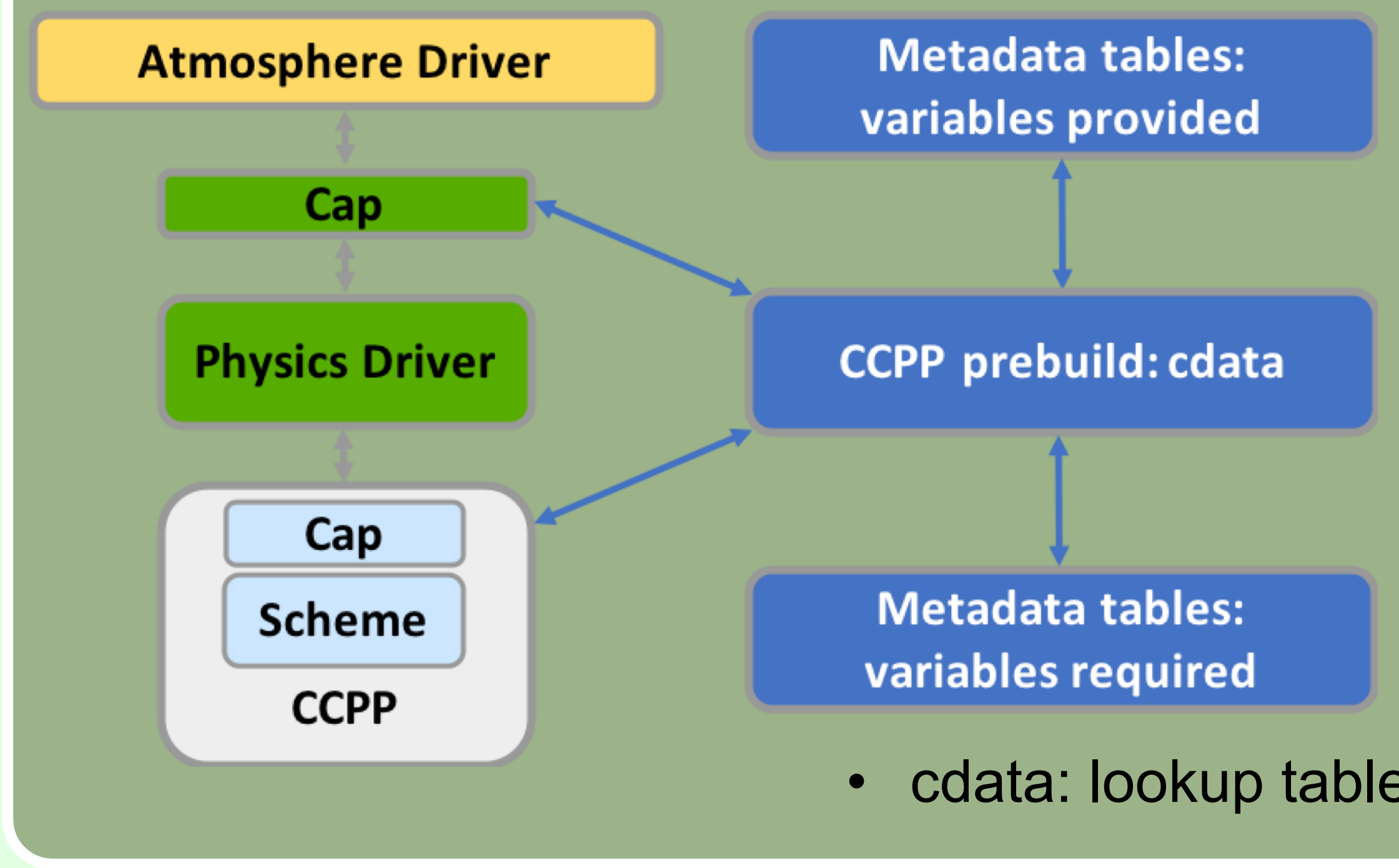
modern Fortran, specify intent and use “implicit none”

initialize all intent(out) variables

error handling by host model, set errflg = 1 and assign errmsg

performance through flexibility: can use threading inside and/or outside physics!

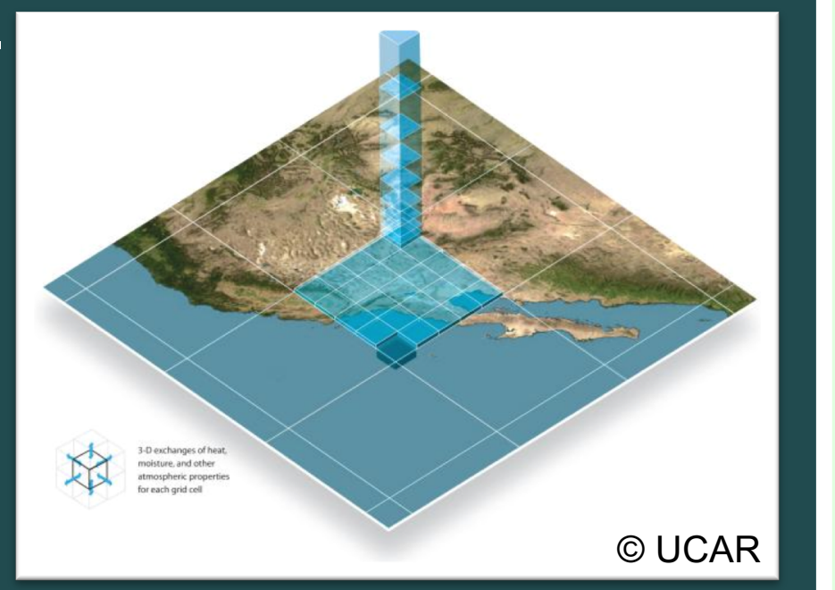
Behind the scenes: CCPP technical implementation

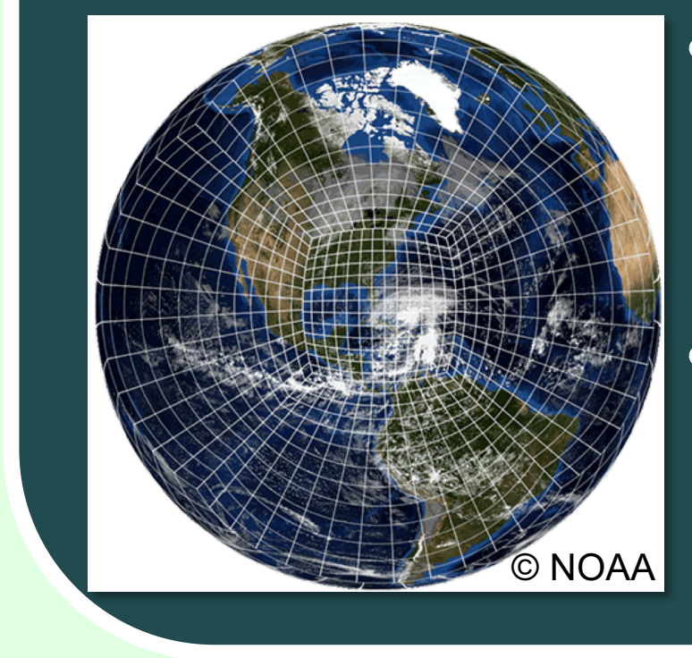

Atmosphere Driver
Cap
Physics Driver
Cap
Scheme
CCPP
Metadata tables: variables provided
CCPP prebuild: cdata
Metadata tables: variables required

- Metadata tables on host side provide information on variables available from the host model.
- Required variables must be provided by host model, incl. memory management (allocation).
- Python script **ccpp_prebuild.py** runs before build time, matches variables by standard_name.
  - Consistency checks of units, rank, type, etc.
  - Auto-generates caps for physics schemes.
  - Auto-generates code inside host model cap to populate **cdata** structure (see below)
  - Auto-generates makefiles for schemes, caps.
- cdata: lookup table standard_name → address in memory in C space.

Tested environments

- CCPP is implemented in the GMTB Single Column Model SCM and the Geophysical Fluid Dynamics Laboratory Finite Volume Cubed-Sphere FV3-based NOAA GFS
- Hierarchical model development: **SCM** allows for testing a physics suite using external forcing w/o dycore feedbacks


3-D exchanges of heat, moisture, and other atmospheric properties for each grid cell
© UCAR


© NOAA

- **FV3**: selected as dycore for Next Generation Global Prediction System (NGGPS) to replace GSM core of GFS
- Runs as a unified, fully-coupled system, in NOAA’s Environmental Modeling System infrastructure.

Host cap – runtime physics selection

```
...
subroutine physics_init(ccpp_suite_name)
  character(len=*), intent(in) :: ccpp_suite_name
  integer :: ierr

  call ccpp_init(ccpp_suite_name, cdata, ierr)
  ! Auto-generated list of calls to ccpp_field_add
  call ccpp_field_add(t2, ‘air_temperature’, ierr)
  ...

end subroutine physics_init

subroutine physics_run(part)
  integer, intent(in) :: part
  integer :: ierr

  call ccpp_run(cdata%suite%ipds(part), cdata, ierr)
  if (ierr/=0) ...

end subroutine physics_run
...
```

runtime suite definition file

add host model vars to cdata structure

error handling by host model

Runtime suite definition file

```
<suite name="GFS_oper_2017" lib="gfsphys" ver="1">
  <init>IPD_initialize</init>
  ...
  <ipd part="2">
    <subcycle loop="1">
      ...
      <scheme>GFS_rrtmg_pre_run</scheme>
      <scheme>rrtmg_sw_pre_run</scheme>
      <scheme>rrtmg_sw_run</scheme>
      <scheme>rrtmg_sw_post_run</scheme>
      ...
    </subcycle>
  </ipd>
  ...
  <finalize>IPD_finalize</finalize>
</suite>
```

init

suite-specific interstitial

run

scheme-specific interstitial

finalize