Radiation: common code and diagnostics?

Gustavo Hime

MPI-M

{gustavo.hime@mpimet.mpg.de}

Paris, January 16, 2017

Introduction

✔ Why radiation?

Introduction

- ✔ Why radiation?
  - ✘ Relatively stable/reliable.

Introduction

- ✔ **Why radiation?**
  - ✘ **Relatively stable/reliable.**
  - ✘ **Little current development.**

Introduction

- ✔ Why radiation?
  - ✘ Relatively stable/reliable.
  - ✘ Little current development.
  - ✘ Renewed/accumulated interest.

Introduction

- ✔ **Why radiation? Lowest cost/benefit ratio.**

Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs.

Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs.
  - ✘ RRTM: a F77 code from the 80s.

Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs.
  - ✘ RRTM: a F77 code from the 80s.
  - ✘ Two F90 codes *based* on it.

Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs.
  - ✘ RRTM: a F77 code from the 80s.
  - ✘ Two F90 codes *based* on it.
  - ✘ Neither is a module.

Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs: keep calm and carry on.

Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs: keep calm and carry on.
- ✔ The goal.

Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs: keep calm and carry on.
- ✔ The goal.
- ✘ Use it outside of ICON.

Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs: keep calm and carry on.
- ✔ The goal.
  - ✘ Use it outside of ICON.
  - ✘ Have a single radiation model.

Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs: keep calm and carry on.
- ✔ The goal.
  - ✘ Use it outside of ICON.
  - ✘ Have a single radiation model.
  - ✘ Maintain a single code base.

Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs: keep calm and carry on.
- ✔ The goal.
  - ✘ Use it outside of ICON.
  - ✘ Have a single radiation model.
  - ✘ Maintain a single code base.
  - ✘ Improve the quality of code base.

Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs: keep calm and carry on.
- ✔ The goal.
  - ✘ Use it outside of ICON.
  - ✘ Have a single radiation model.
  - ✘ Maintain a single code base.
  - ✘ Improve the quality of code base.
  - ✘ ... and more to come.

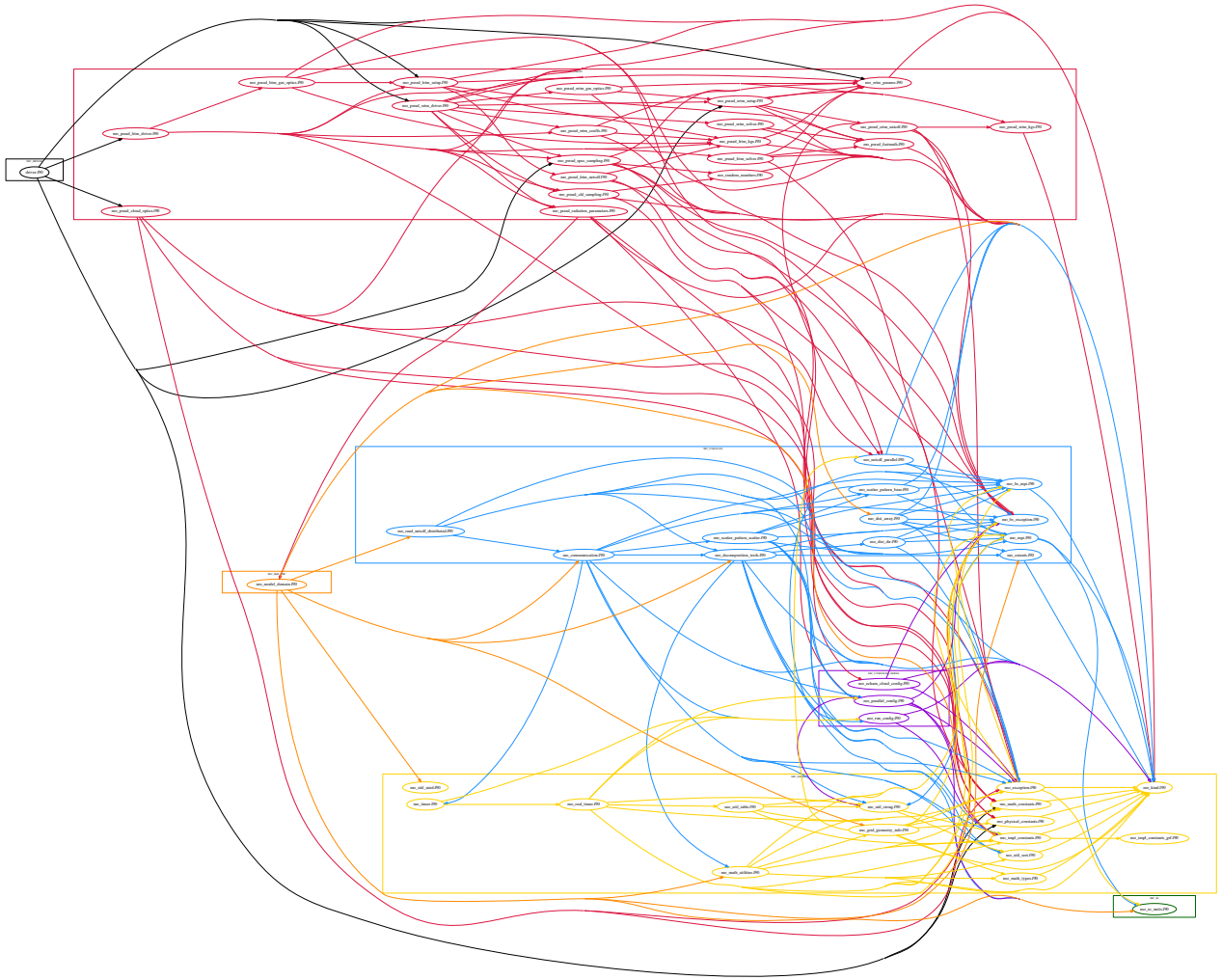
Introduction

- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs: shouldn't be too difficult.
- ✔ The goal: make a software library out of it.

Introduction

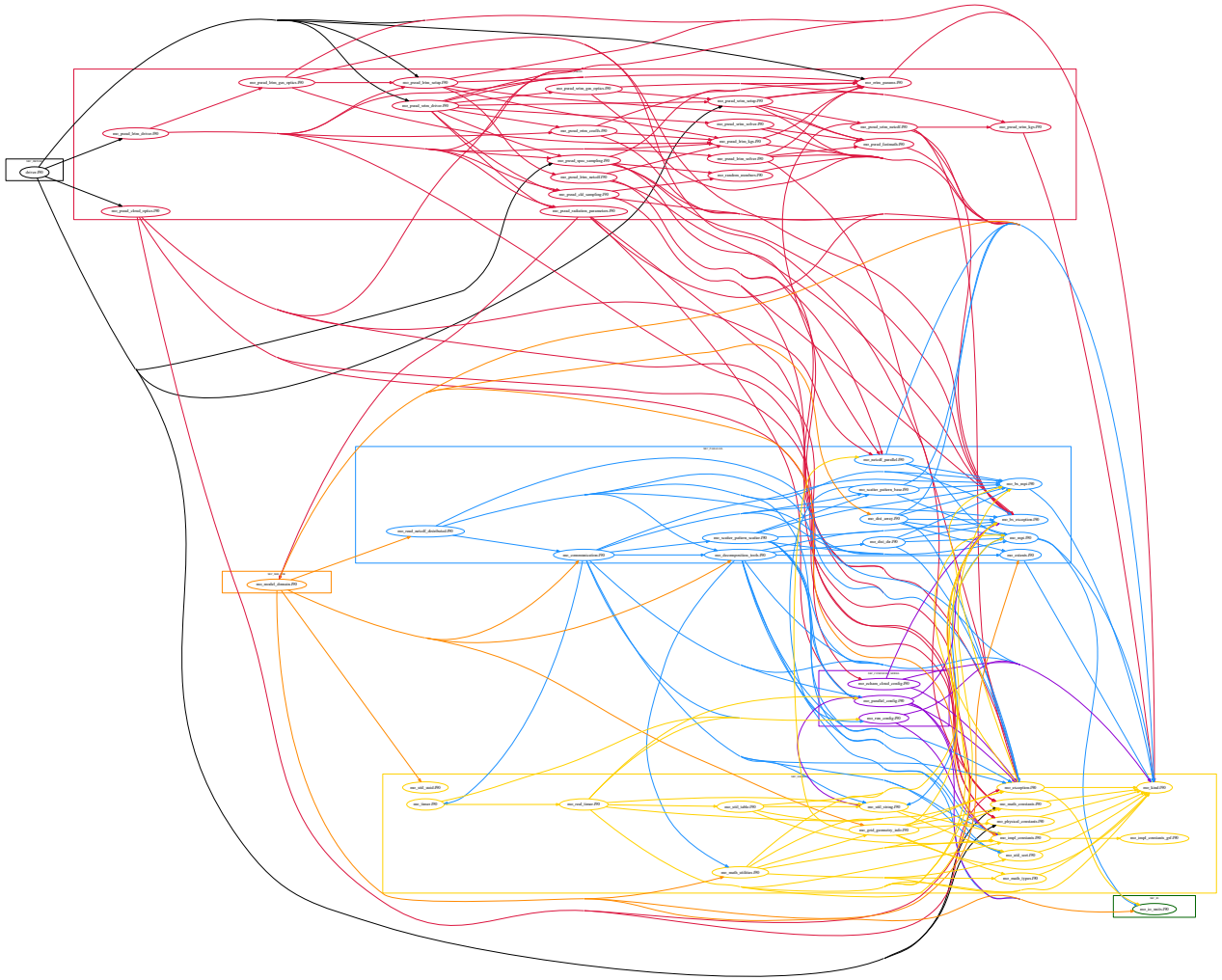
- ✔ Why radiation? Lowest cost/benefit ratio.
- ✔ The state of affairs: shouldn't be too difficult.
- ✔ The goal: make a software library out of it.
- ✔ The road...

Starting point (April 2016)



Once upon a time...

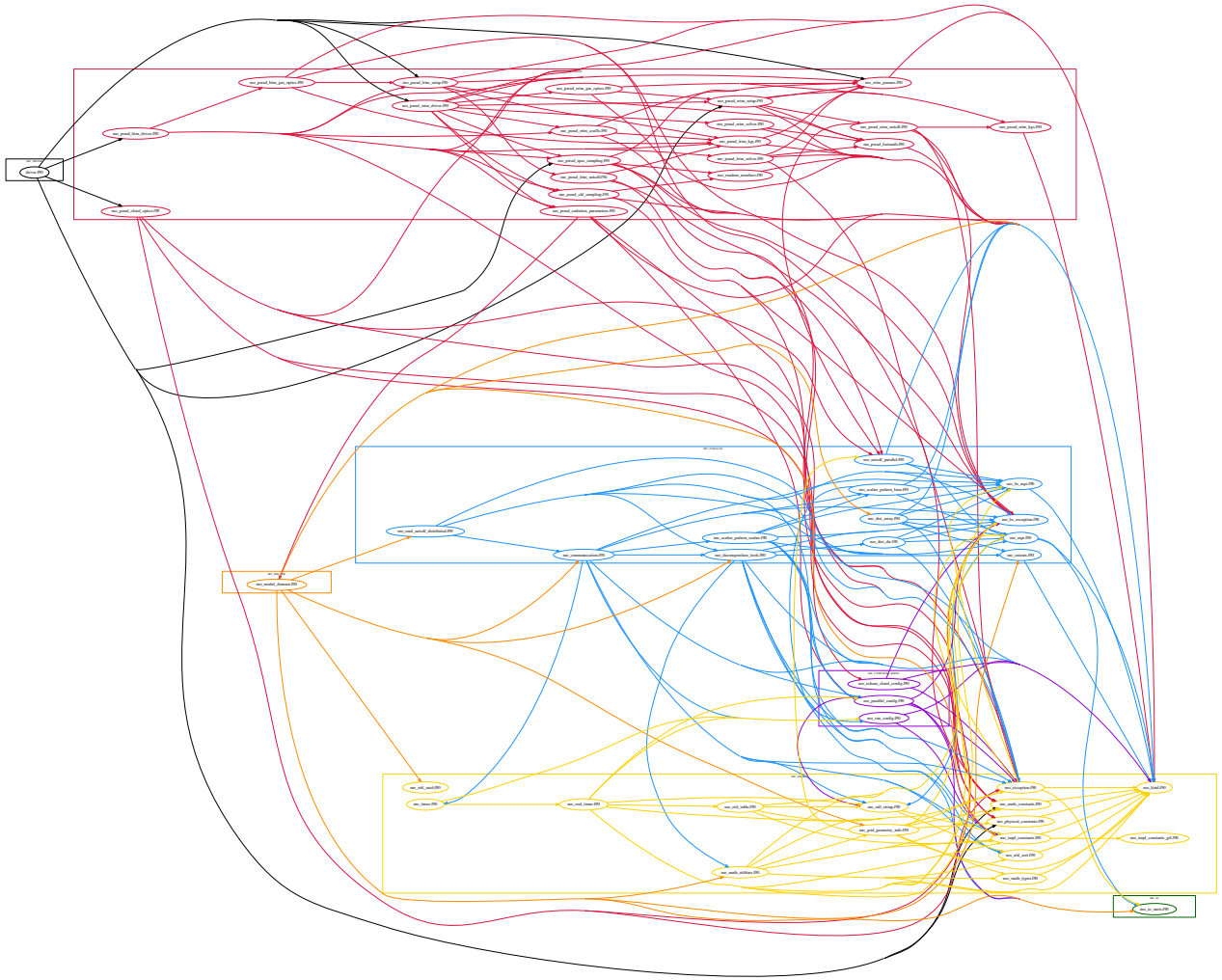
Starting point (April 2016)



Once upon a time...

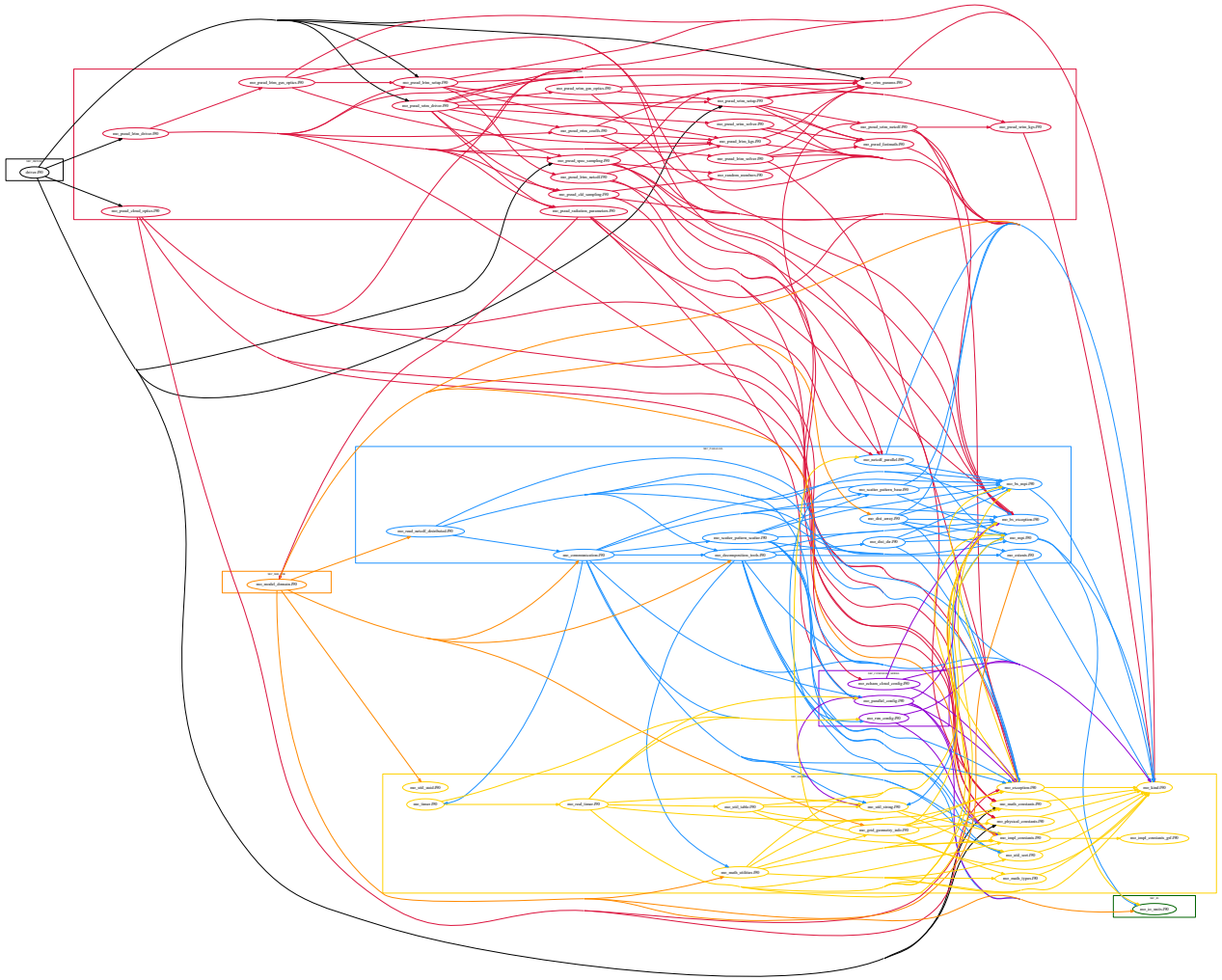
there was a hack.

Starting point (April 2016)



Once upon a time...
there was a hack.
Fortran requires compilation of dependencies.

Starting point (April 2016)



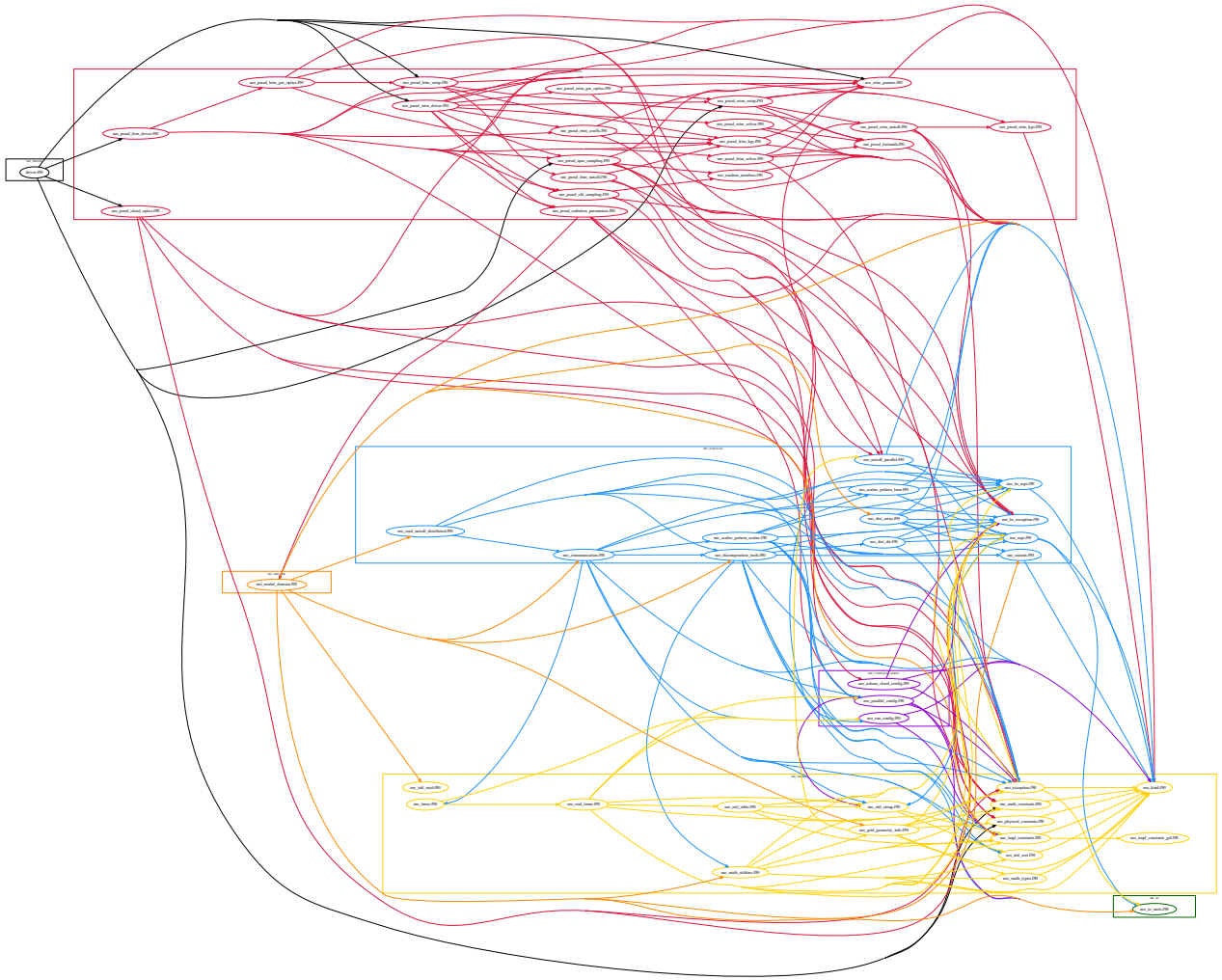
Once upon a time...

there was a hack.

Fortran requires compilation of dependencies.

Dependencies have their own dependencies too.

Starting point (April 2016)



Once upon a time...

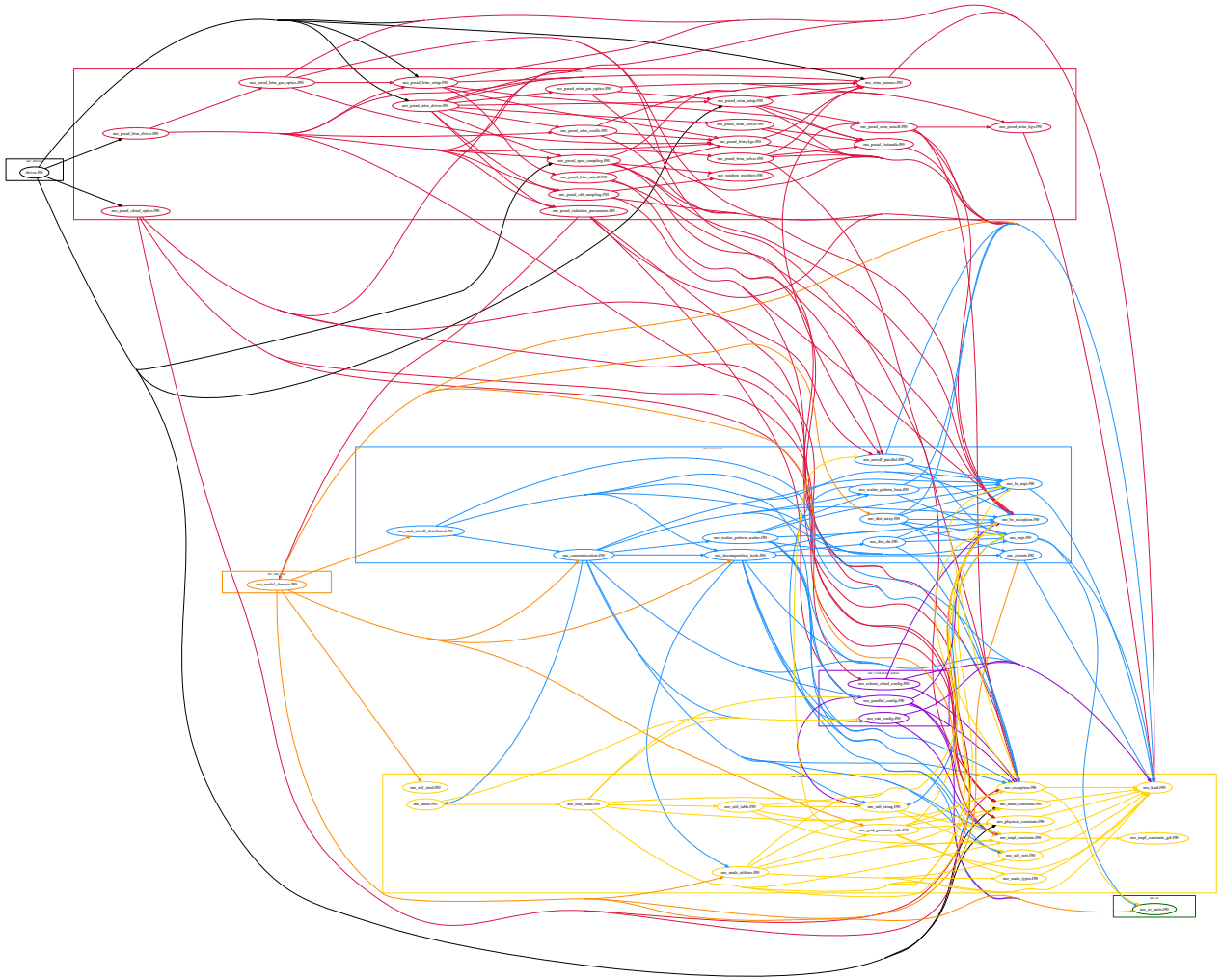
there was a hack.

Fortran requires compilation of dependencies.

Dependencies have their own dependencies too.

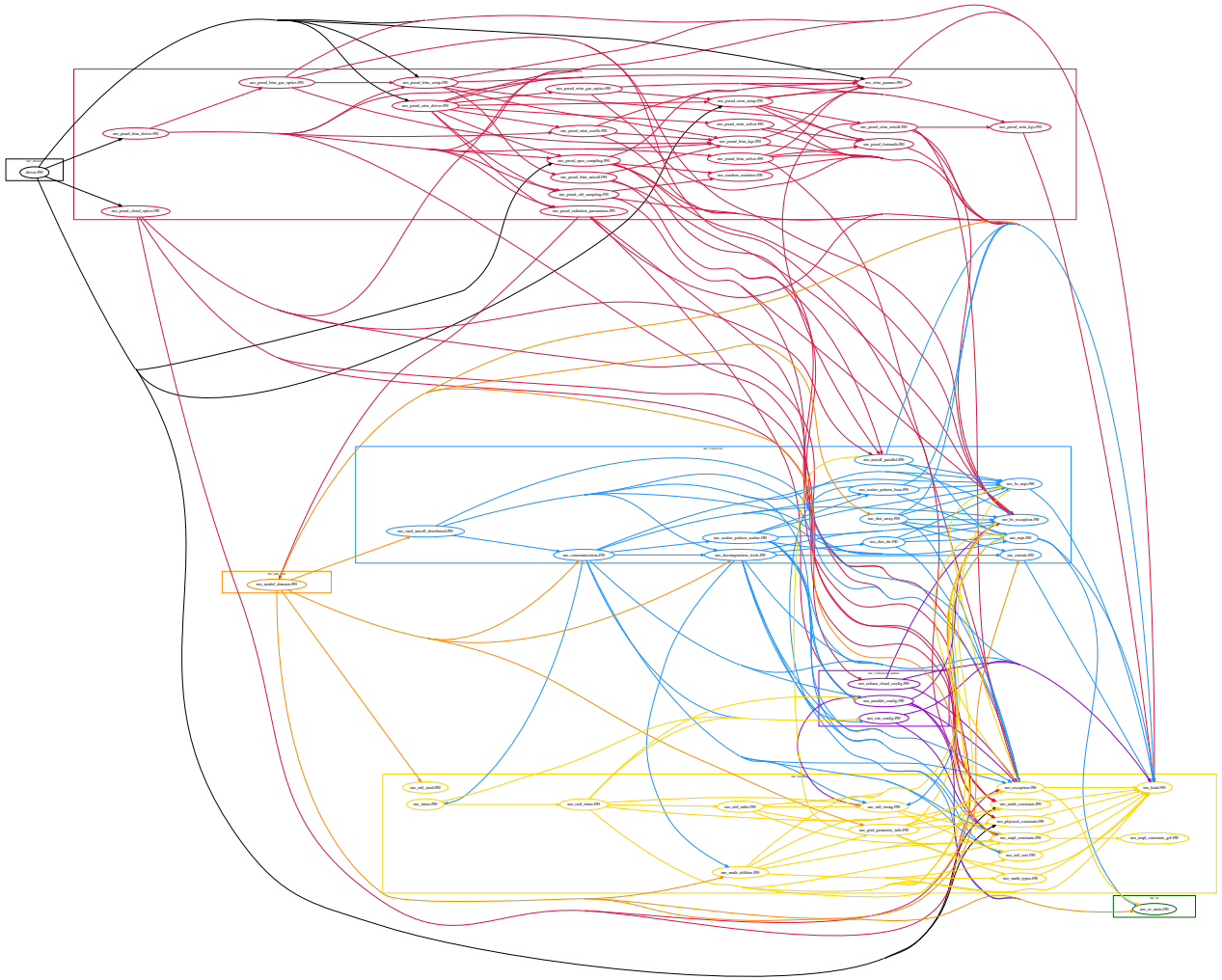
The higher the level of a source file, the larger the dependency tree.

Starting point (April 2016)



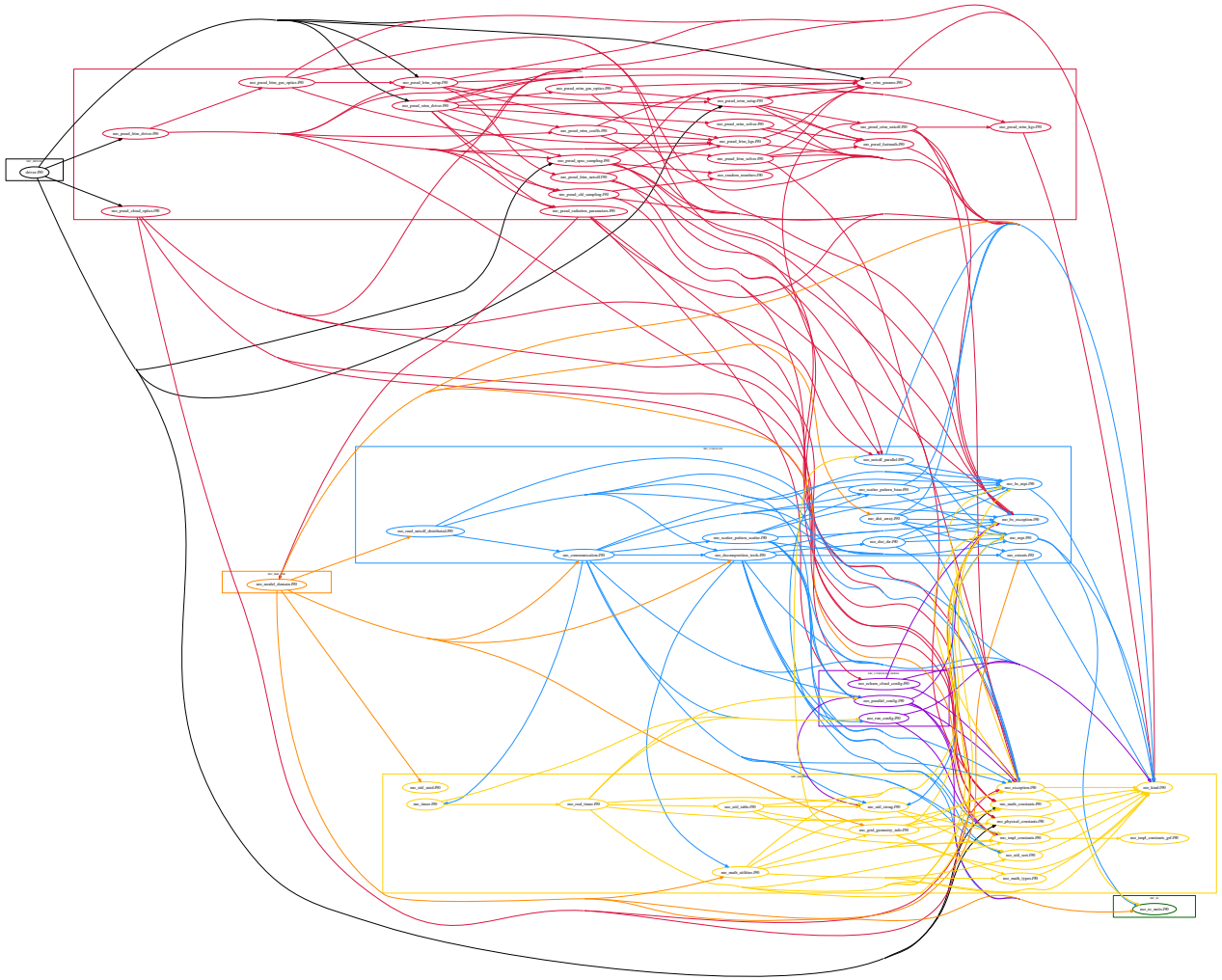
Once upon a time...
there was a hack.
Fortran requires compilation of dependencies.
Dependencies have their own dependencies too.
The higher the level of a source file, the larger the dependency tree.
To compile `driver.f90`...

Starting point (April 2016)



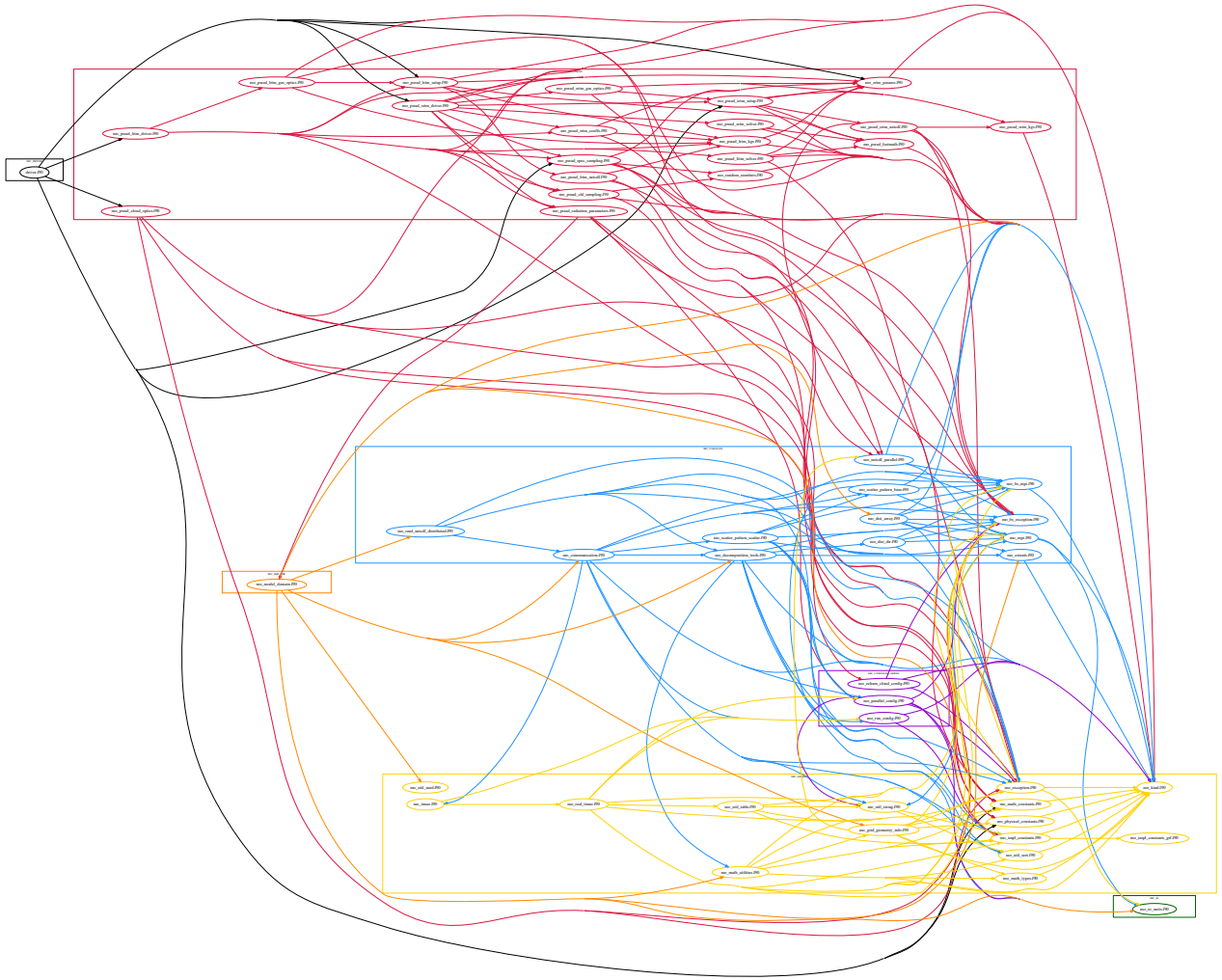
Once upon a time...
there was a hack.
Fortran requires compilation of dependencies.
Dependencies have their own dependencies too.
The higher the level of a source file, the larger the dependency tree.
To compile `driver.f90`...
more than 100 sources must be compiled first.

Starting point (April 2016)



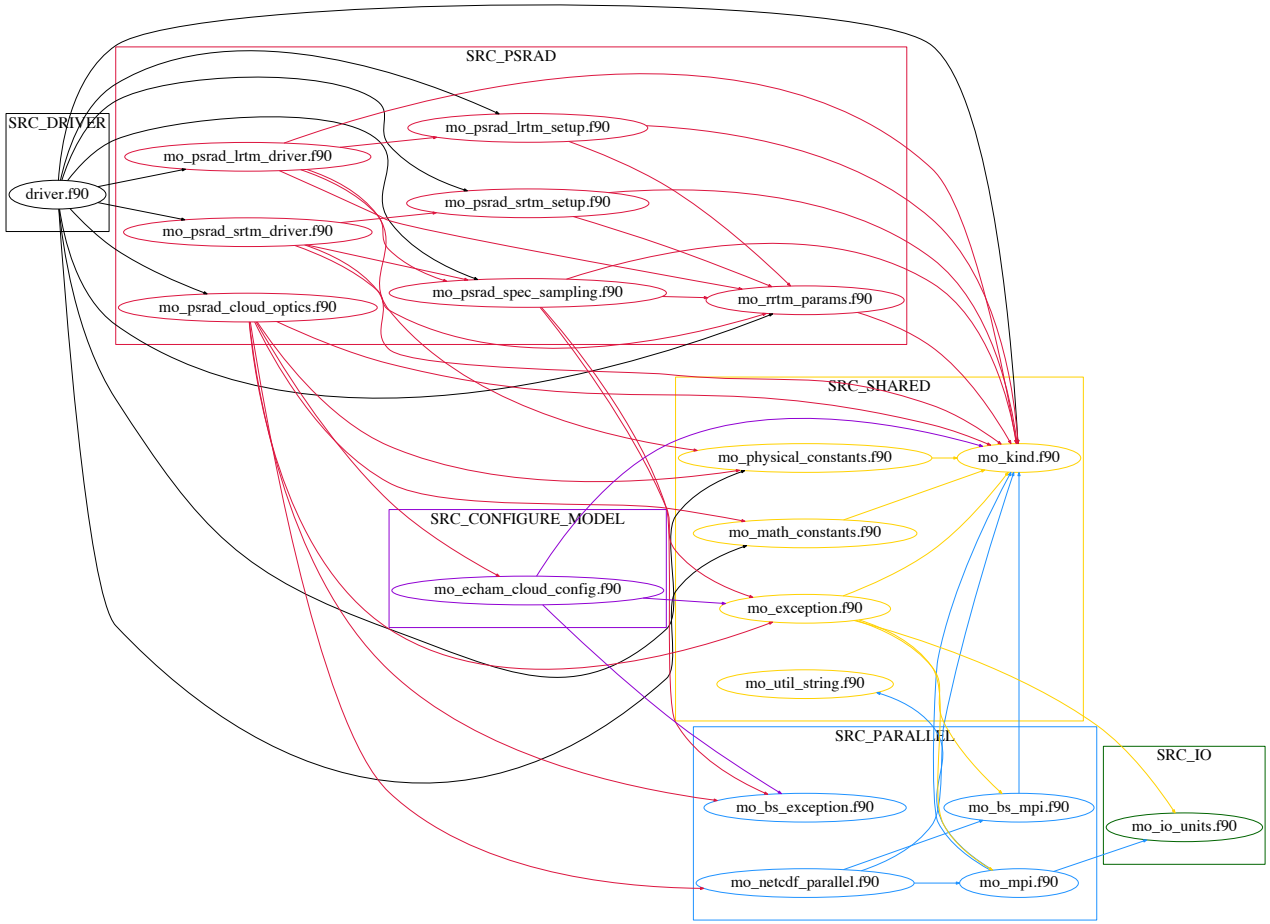
Once upon a time...
there was a hack.
Fortran requires compilation of dependencies.
Dependencies have their own dependencies too.
The higher the level of a source file, the larger the dependency tree.
To compile `driver.f90`...
more than 100 sources must be compiled first.
More exactly, 104 sources, 130k lines...

Starting point (April 2016)



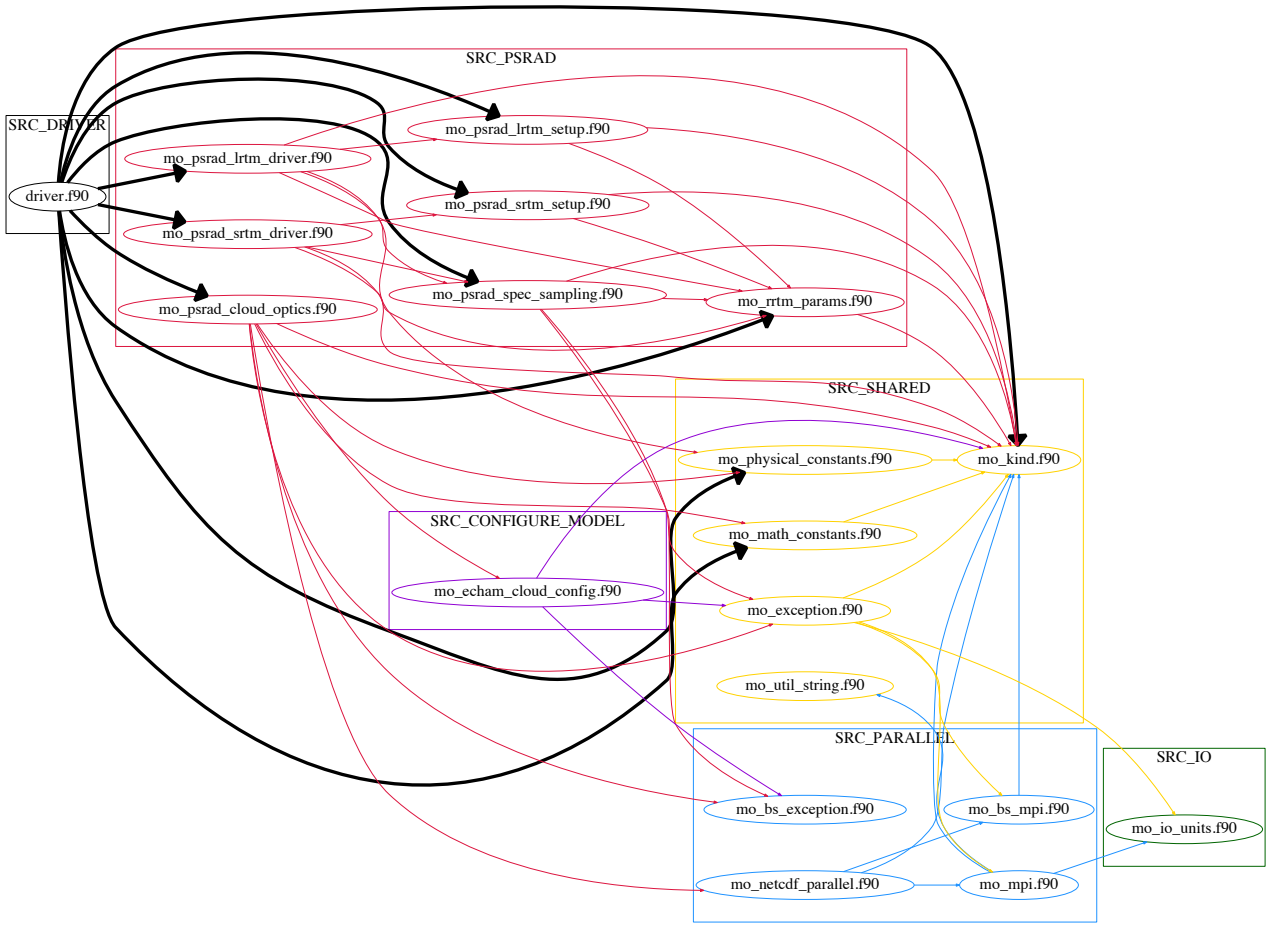
Once upon a time...
there was a hack.
Fortran requires compilation of dependencies.
Dependencies have their own dependencies too.
The higher the level of a source file, the larger the dependency tree.
To compile `driver.f90`...
more than 100 sources must be compiled first.
More exactly, 104 sources, 130k lines...
of which 107k containing code.

Cutting it out



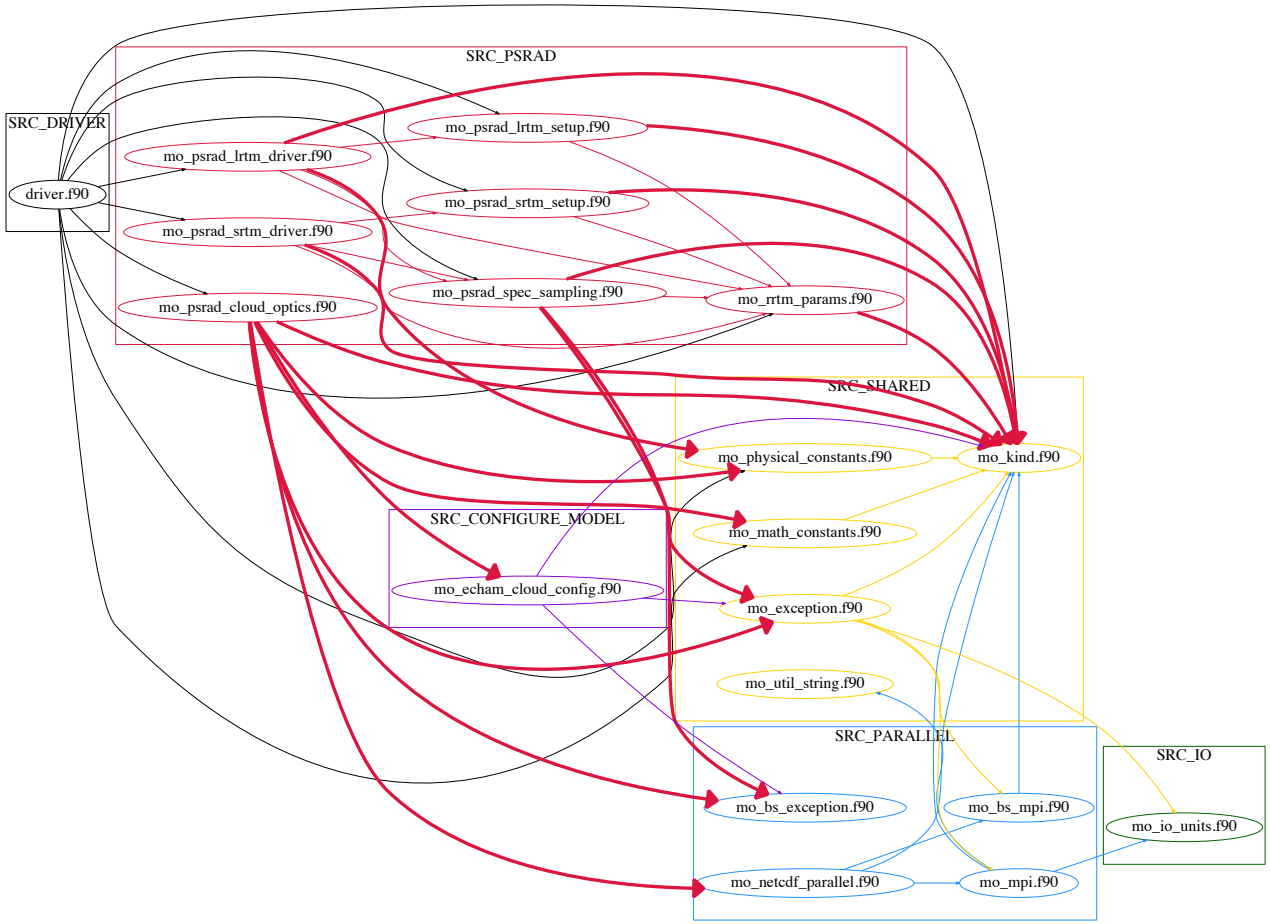
Dependency tree of `driver.f90`.

Cutting it out



Dependency tree of `driver.f90`.

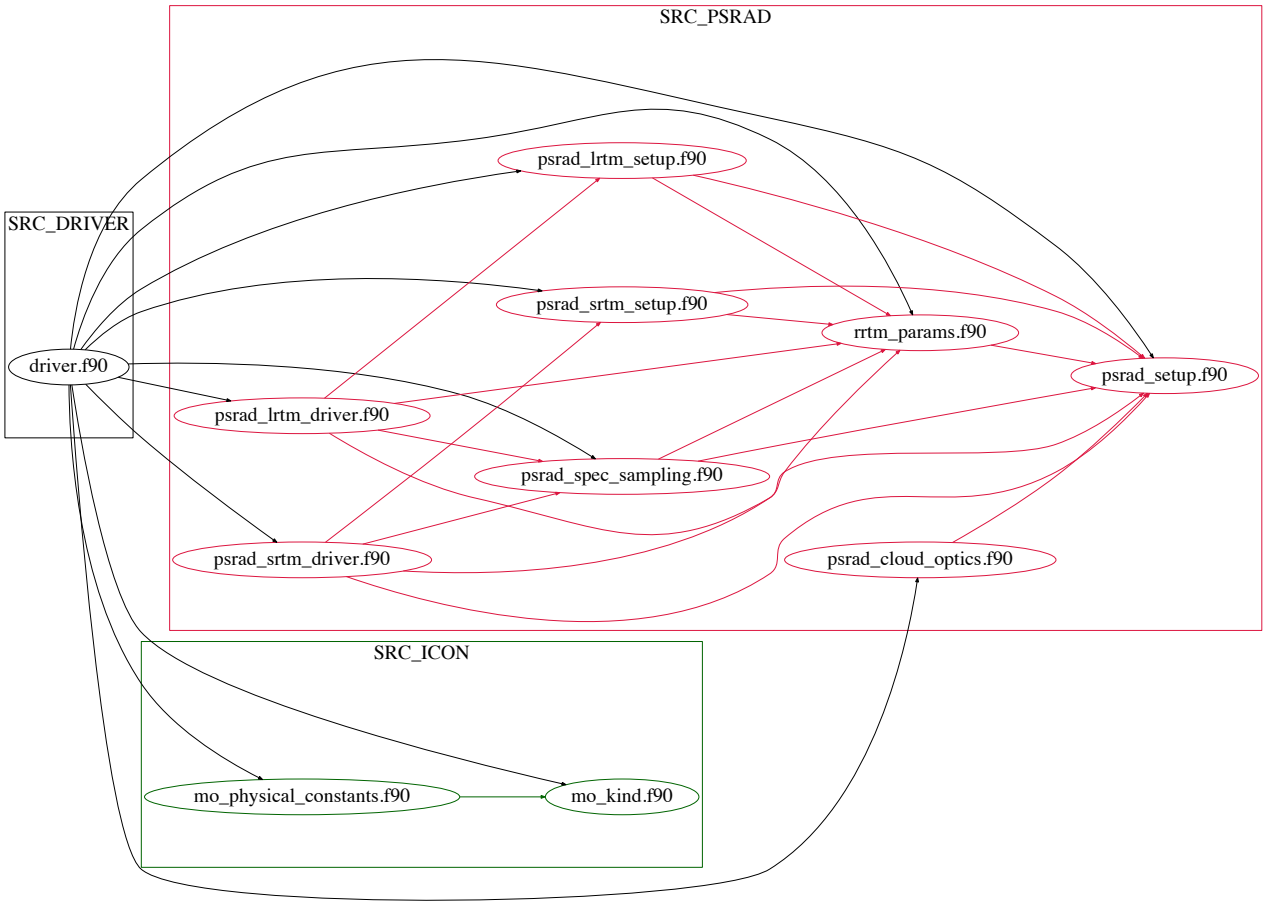
Cutting it out



Dependency tree of `driver.f90`.

Dependencies of the `PSRad` modules on other parts of ICON.

Cutting it out

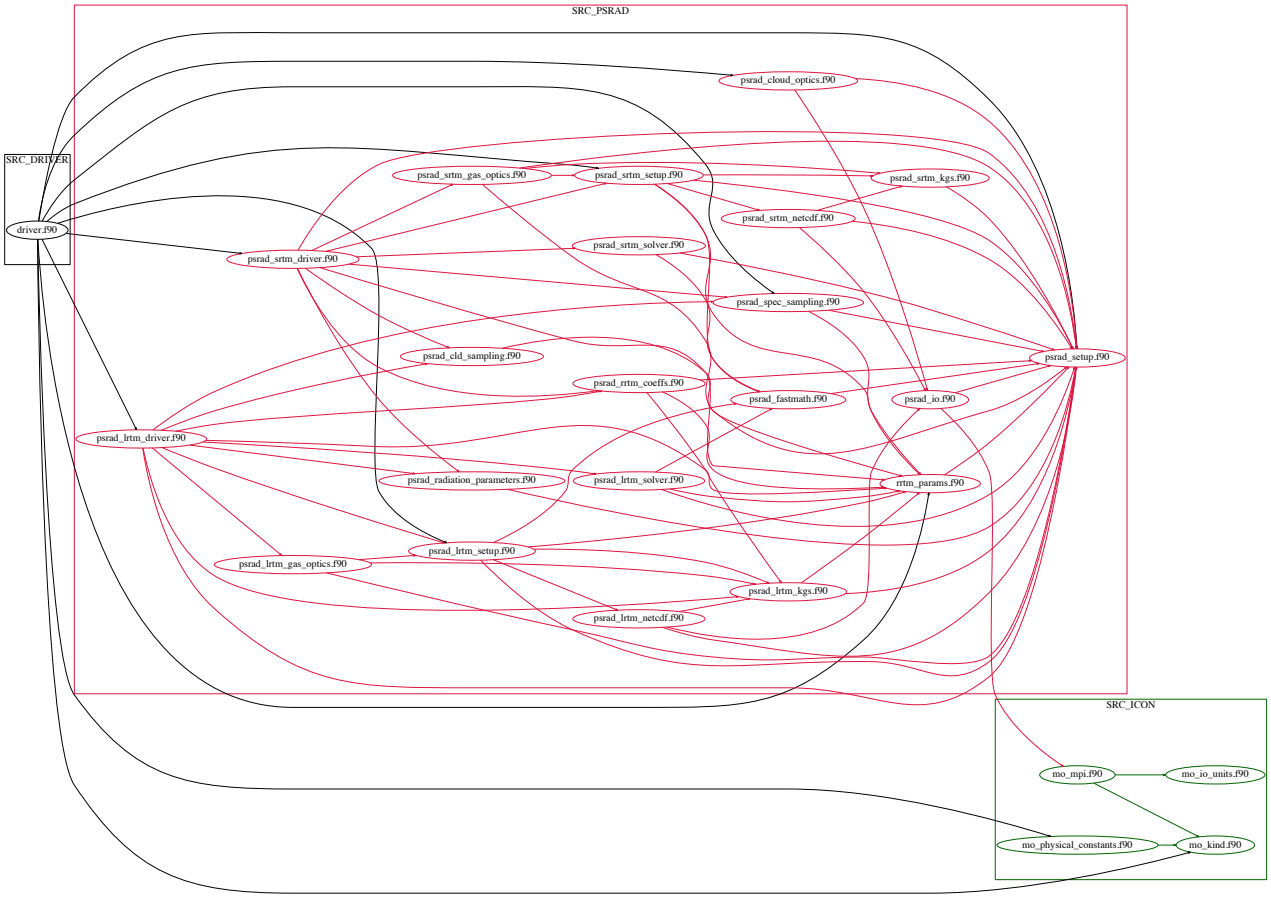


Dependency tree of `driver.f90`.

Dependencies of the `PSRad` modules on other parts of ICON.

Dependencies that resemble the usage of a library.

Cutting it out

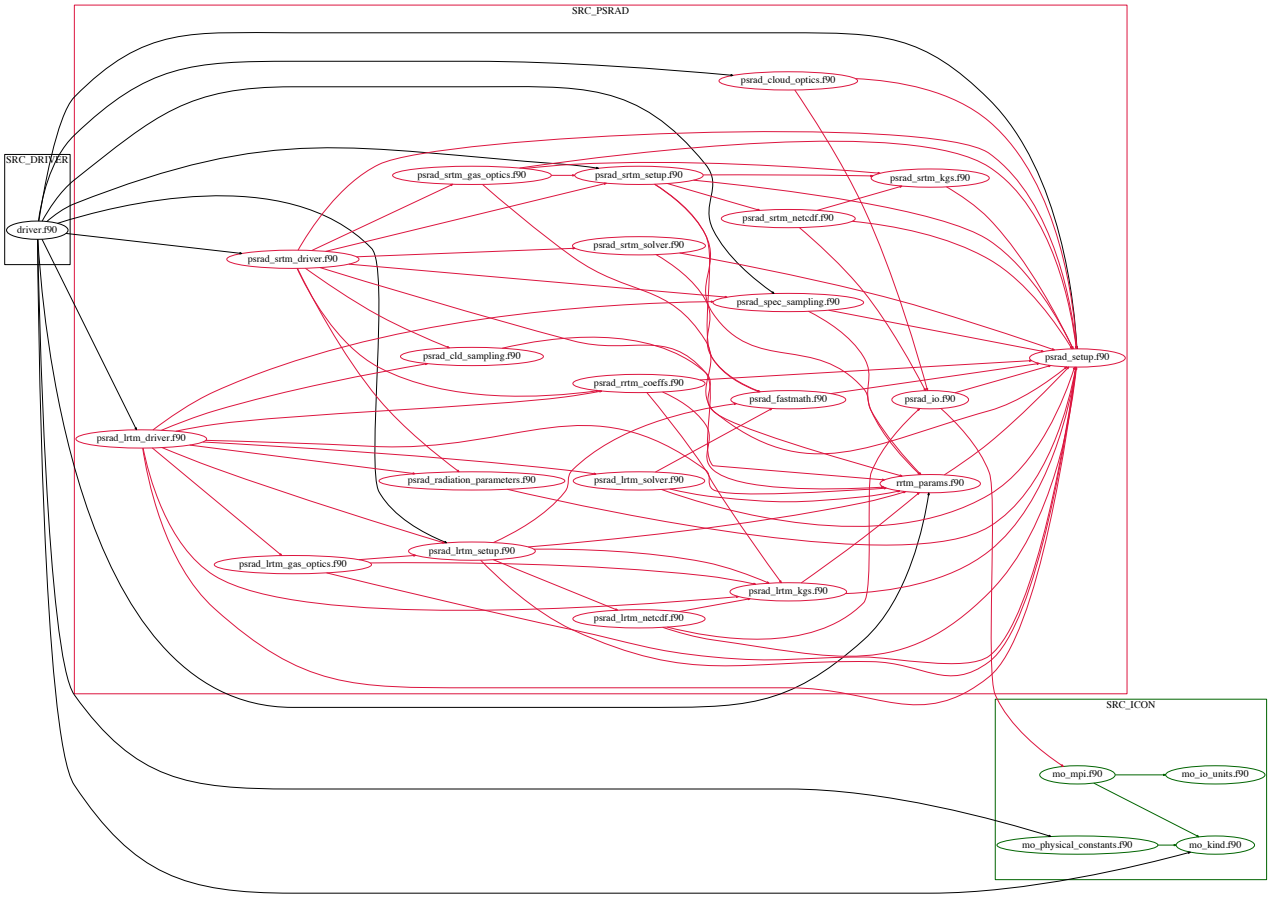


Dependency tree of `driver.f90`.

Dependencies of the `PSRad` modules on other parts of ICON.

Dependencies that resemble the usage of a library.

Cutting it out



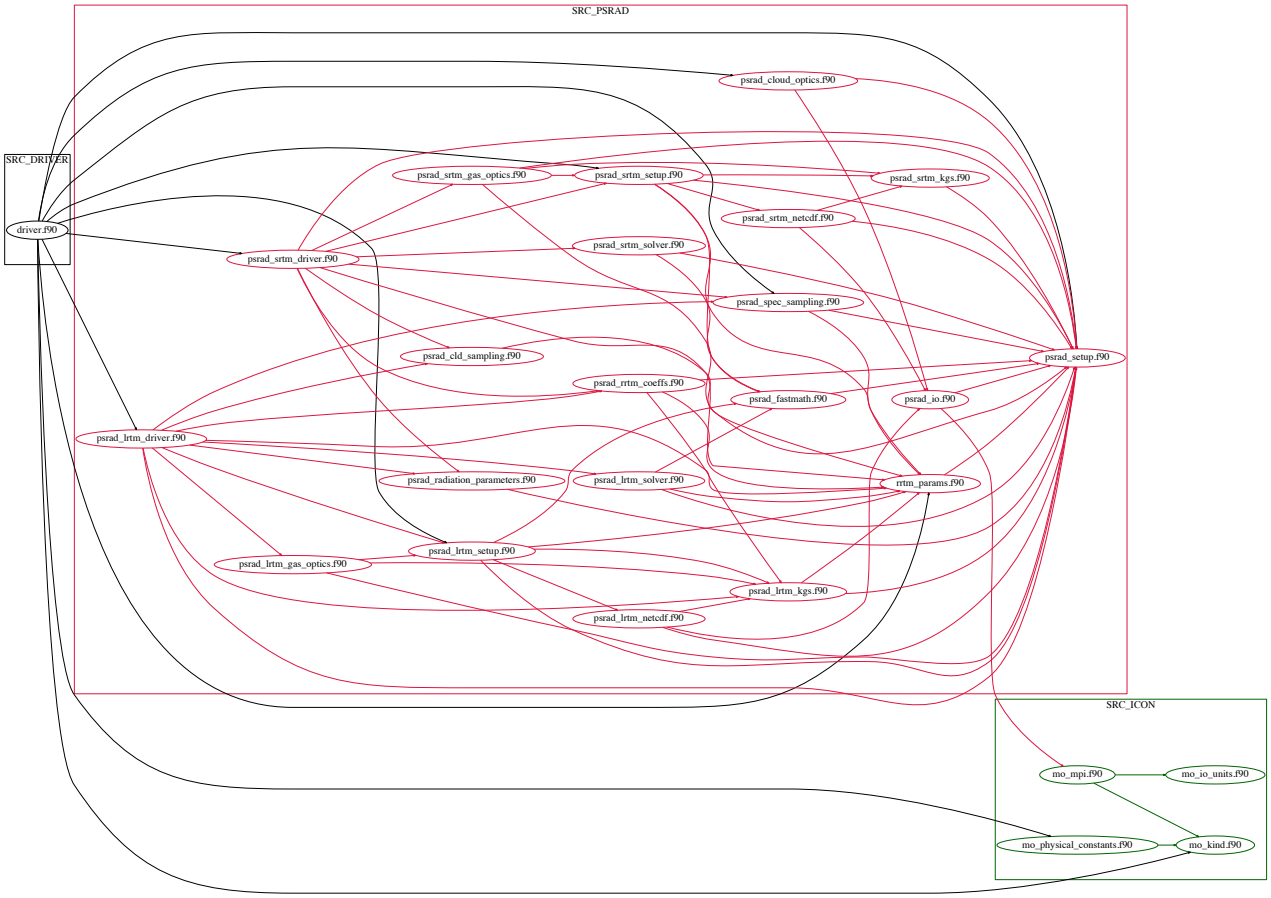
Dependency tree of `driver.f90`.

Dependencies of the `PSRad` modules on other parts of ICON.

Dependencies that resemble the usage of a library.

Now 29 sources with 22k lines, 15k of which are code (vs 104 files with 107k lines).

Cutting it out



Dependency tree of `driver.f90`.

Dependencies of the `PSRad` modules on other parts of ICON.

Dependencies that resemble the usage of a library.

Now 29 sources with 22k lines, 15k of which are code (vs 104 files with 107k lines).

Moving on in June 2016.

Stitching it in

```
SUBROUTINE lrtm(kproma, kbdim, klev, play, psfc, tlay, tlev, tsfc, &
  wkl, wx, coldry, emis, icld, cldfr, taucld, tauaer, rnseeds, uflx, dflx, &
  uflxc, dflxc)

  USE mo_psrad_general, ONLY: ngptlw, ngas, ih2o, ico2, ico, nreact

  INTEGER, INTENT(IN) :: &
    kbdim, & !< Maximum block length
    kproma, & !< Number of horizontal columns
    klev !< Number of model layers

  REAL(wp), INTENT(IN) :: &
    play(KBDIM,klev), & !< Layer pressures [hPa, mb] (KBDIM,klev)
    psfc(KBDIM), & !< Surface pressure [hPa, mb] (KBDIM)
    tlay(KBDIM,klev), & !< Layer temperatures [K] (KBDIM,klev)
    tlev(KBDIM,klev+1), & !< Interface temperatures [K] (KBDIM,klev+1)
    tsfc(KBDIM), & !< Surface temperature [K] (KBDIM)
    wkl(KBDIM,klev,ngas), & !< Gas volume mixing ratios
    wx(KBDIM,ncfc,klev), & !< CFC type gas volume mixing ratios
    coldry(KBDIM,klev), & !< Column dry amount
    emis(KBDIM,nbndlw), & !< Surface emissivity  (KBDIM,nbndlw)
    cldfr(KBDIM,klev), & !< Cloud fraction  (KBDIM,klev)
    taucld(KBDIM,klev,nbndlw), & !< Coud optical depth (KBDIM,klev,nbndlw)
    tauaer(KBDIM,klev,nbndlw)    !< Aerosol optical depth (KBDIM,klev,nbndlw)

  INTEGER, INTENT(IN) :: icld(KBDIM,klev)
  ! Seeds for random number generator (KBDIM,:)
  INTEGER, INTENT(INOUT) :: rnseeds(:,:)

  ! Longwave fluxes in [W/m2]
  REAL(wp), DIMENSION(KBDIM,klev+1), INTENT(OUT) :: &
    uflx, & ! Total sky upward
    dflx, & ! Total sky downward
    uflxc, & ! Clear sky upward
    dflxc ! Clearr sky downward
```

Fortran interface for long-wave solver.

Stitching it in

```
SUBROUTINE lrtm(kproma, kbdim, klev, play, psfc, tlay, tlev, tsfc, &
  wkl, wx, coldry, emis, icld, cldfr, taucld, tauaer, rnseeds, uflx, dflx, &
  uflxc, dflxc)

  USE mo_psrad_general, ONLY: ngptlw, ngas, ih2o, ico2, ico, nreact

  INTEGER, INTENT(IN) :: &
    kbdim, & !< Maximum block length
    kproma, & !< Number of horizontal columns
    klev !< Number of model layers

  REAL(wp), INTENT(IN) :: &
    play(KBDIM,klev), & !< Layer pressures [hPa, mb] (KBDIM,klev)
    psfc(KBDIM), & !< Surface pressure [hPa, mb] (KBDIM)
    tlay(KBDIM,klev), & !< Layer temperatures [K] (KBDIM,klev)
    tlev(KBDIM,klev+1), & !< Interface temperatures [K] (KBDIM,klev+1)
    tsfc(KBDIM), & !< Surface temperature [K] (KBDIM)
    wkl(KBDIM,klev,ngas), & !< Gas volume mixing ratios
    wx(KBDIM,ncfc,klev), & !< CFC type gas volume mixing ratios
    coldry(KBDIM,klev), & !< Column dry amount
    emis(KBDIM,nbndlw), & !< Surface emissivity  (KBDIM,nbndlw)
    cldfr(KBDIM,klev), & !< Cloud fraction  (KBDIM,klev)
    taucld(KBDIM,klev,nbndlw), & !< Coud optical depth (KBDIM,klev,nbndlw)
    tauaer(KBDIM,klev,nbndlw)    !< Aerosol optical depth (KBDIM,klev,nbndlw)

  INTEGER, INTENT(IN) :: icld(KBDIM,klev)
  ! Seeds for random number generator (KBDIM,:)
  INTEGER, INTENT(INOUT) :: rnseeds(:,:)

  ! Longwave fluxes in [W/m2]
  REAL(wp), DIMENSION(KBDIM,klev+1), INTENT(OUT) :: &
    uflx, & ! Total sky upward
    dflx, & ! Total sky downward
    uflxc, & ! Clear sky upward
    dflxc ! Clearr sky downward
```

Fortran interface for long-wave solver.

Fortran-exclusive data types.

Stitching it in

```
SUBROUTINE psrad_lrtm(kproma, kbdim, klev, play, psfc, tlay, tlev, tsfc, &
  wkl, wx, coldry, emis, icld, cldfr, taucld, tauaer, rnseeds, seed_size, &
  uflx, dflx, uflxc, dflxc) BIND(c)

  USE mo_psrad_lrtm_driver,   ONLY : lrtm
  USE mo_psrad_general, ONLY : wp, ncfc, nbndlw, ngas

  INTEGER(c_int), VALUE, INTENT(IN) :: kbdim, kproma, klev, seed_size

  INTEGER(c_int), INTENT(IN) :: icld(KBDIM, klev)
  INTEGER(c_int), INTENT(INOUT) :: rnseeds(KBDIM, seed_size)
  REAL(c_double), INTENT(IN) :: play(KBDIM,klev), psfc(KBDIM), &
    tlay(KBDIM,klev), tlev(KBDIM,klev+1), tsfc(KBDIM), &
    wkl(KBDIM,klev,ngas), wx(KBDIM,ncfc,klev), &
    coldry(KBDIM,klev), emis(KBDIM,nbndlw), cldfr(KBDIM,klev), &
    taucld(KBDIM,klev,nbndlw), tauaer(KBDIM,klev,nbndlw)
  REAL(c_double), INTENT(OUT) :: uflx(KBDIM,klev+1), dflx(KBDIM,klev+1), &
    uflxc(KBDIM,klev+1), dflxc(KBDIM,klev+1)

  CALL lrtm(kproma, KBDIM, klev, play, psfc, tlay, tlev, tsfc, &
    wkl, wx, coldry, emis, icld, cldfr, taucld, tauaer, rnseeds, &
    uflx, dflx, uflxc, dflxc)

END SUBROUTINE
```

Fortran interface for long-wave solver.

Fortran-exclusive data types.

F77 C-compatible types only.

Stitching it in

```
SUBROUTINE psrad_lrtm(kproma, kbdim, klev, play, psfc, tlay, tlev, tsfc, &
  wkl, wx, coldry, emis, icld, cldfr, taucld, tauaer, rnseeds, seed_size, &
  uflx, dflx, uflxc, dflxc) BIND(c)

  USE mo_psrad_lrtm_driver,   ONLY : lrtm
  USE mo_psrad_general, ONLY : wp, ncfc, nbndlw, ngas

  INTEGER(c_int), VALUE, INTENT(IN) :: kbdim, kproma, klev, seed_size

  INTEGER(c_int), INTENT(IN) :: icld(KBDIM, klev)
  INTEGER(c_int), INTENT(INOUT) :: rnseeds(KBDIM, seed_size)
  REAL(c_double), INTENT(IN) :: play(KBDIM,klev), psfc(KBDIM), &
    tlay(KBDIM,klev), tlev(KBDIM,klev+1), tsfc(KBDIM), &
    wkl(KBDIM,klev,ngas), wx(KBDIM,ncfc,klev), &
    coldry(KBDIM,klev), emis(KBDIM,nbndlw), cldfr(KBDIM,klev), &
    taucld(KBDIM,klev,nbndlw), tauaer(KBDIM,klev,nbndlw)
  REAL(c_double), INTENT(OUT) :: uflx(KBDIM,klev+1), dflx(KBDIM,klev+1), &
    uflxc(KBDIM,klev+1), dflxc(KBDIM,klev+1)

  CALL lrtm(kproma, KBDIM, klev, play, psfc, tlay, tlev, tsfc, &
    wkl, wx, coldry, emis, icld, cldfr, taucld, tauaer, rnseeds, &
    uflx, dflx, uflxc, dflxc)

END SUBROUTINE
```

```
void psrad_lrtm(int kproma, int kbdim, int klev, double* play,
  double* psfc, double* tlay, double* tlev, double* tsfc, double* wkl,
  double* wx, double* coldry, double* emis, int* icldlyr,
  double* cldfr, double* taucld,
  double* tauaer, int* rnseeds, int seed_size,
  double* uflx, double* dflx, double* uflxc, double* dflxc);
```

Fortran interface for long-wave solver.

Fortran-exclusive data types.

F77 C-compatible types only.

Compatible calling convention.

Stitching it in

```
void psrad_lrtm(int kproma, int kbdim, int klev, double* play,
  double* psfc, double* tlay, double* tlev, double* tsfc, double* wkl,
  double* wx, double* coldry, double* emis, int* icldlyr,
  double* cldfr, double* taucld,
  double* tauaer, int* rnseeds, int seed_size,
  double* uflx, double* dflx, double* uflxc, double* dflxc);
```

```
psrad.setup_multi(kbdim, klev, C_cloud_layer, C_cloud_lwc, C_cloud_iwc,
  C_cloud_fraction, zenith, albedo, C_pp_sfc, C_tk_sfc, C_hgt_fl_vr, \
  C_pp_fl_vr, C_tk_fl_vr, C_xm_h2o_vr, C_xm_o3_vr, C_xm_n2o_vr, \
  C_xm_co_vr, C_xm_ch4_vr)

psrad.advance_lrtm()

htngrt_lw, htngrt_clr_lw, flxd_lw_vr, flxd_lw_clr_vr, \
flxu_lw_vr, flxu_lw_clr_vr = psrad.get_lw_fluxes()
```

- Fortran interface for long-wave solver.
- Fortran-exclusive data types.
- F77 C-compatible types only.
- Compatible calling convention.
- Python wrapper.

Stitching it in

```
void psrad_lrtm(int kproma, int kbdim, int klev, double* play,
  double* psfc, double* tlay, double* tlev, double* tsfc, double* wkl,
  double* wx, double* coldry, double* emis, int* icldlyr,
  double* cldfr, double* taucld,
  double* tauaer, int* rnseeds, int seed_size,
  double* uflx, double* dflx, double* uflxc, double* dflxc);
```

```
psrad.setup_multi(kbdim, klev, C_cloud_layer, C_cloud_lwc, C_cloud_iwc,
  C_cloud_fraction, zenith, albedo, C_pp_sfc, C_tk_sfc, C_hgt_fl_vr, \
  C_pp_fl_vr, C_tk_fl_vr, C_xm_h2o_vr, C_xm_o3_vr, C_xm_n2o_vr, \
  C_xm_co_vr, C_xm_ch4_vr)

psrad.advance_lrtm()

htngrt_lw, htngrt_clr_lw, flxd_lw_vr, flxd_lw_clr_vr, \
flxu_lw_vr, flxu_lw_clr_vr = psrad.get_lw_fluxes()
```

- Fortran interface for long-wave solver.
- Fortran-exclusive data types.
- F77 C-compatible types only.
- Compatible calling convention.
- Python wrapper.
- Universally bindable as of September 2016.

As of October 2016, refactoring the code.

Results so far:

As of October 2016, refactoring the code.

Results so far:

- ✔ **Fully vectorized long-wave code.**

As of October 2016, refactoring the code.

Results so far:

- ✔ Fully vectorized long-wave code.
- ✔ Long-wave computations are fully data-driven.

As of October 2016, refactoring the code.

Results so far:

- ✔ **Fully vectorized long-wave code.**
- ✔ **Long-wave computations are fully data-driven.**
- ✔ **Performance increase of x2, probably more.**

As of October 2016, refactoring the code.

Results so far:

- ✔ **Fully vectorized long-wave code.**
- ✔ **Long-wave computations are fully data-driven.**
- ✔ **Performance increase of x2, probably more.**
- ✔ **Code redundancy dramatically reduced.**

As of October 2016, refactoring the code.

Results so far:

- ✔ **Fully vectorized long-wave code.**
- ✔ **Long-wave computations are fully data-driven.**
- ✔ **Performance increase of x2, probably more.**
- ✔ **Code redundancy dramatically reduced.**
- ✔ **LRTM line count dropped by 68% (from 5124 to 1609).**

As of October 2016, refactoring the code.

Results so far:

- ✔ **Fully vectorized long-wave code.**
- ✔ **Long-wave computations are fully data-driven.**
- ✔ **Performance increase of x2, probably more.**
- ✔ **Code redundancy dramatically reduced.**
- ✔ **LRTM line count dropped by 68% (from 5124 to 1609).**

Coming up next:

As of October 2016, refactoring the code.

Results so far:

- ✔ Fully vectorized long-wave code.
- ✔ Long-wave computations are fully data-driven.
- ✔ Performance increase of x2, probably more.
- ✔ Code redundancy dramatically reduced.
- ✔ LRTM line count dropped by 68% (from 5124 to 1609).

Coming up next:

- ✔ Retiring old code.

As of October 2016, refactoring the code.

Results so far:

- ✔ Fully vectorized long-wave code.
- ✔ Long-wave computations are fully data-driven.
- ✔ Performance increase of x2, probably more.
- ✔ Code redundancy dramatically reduced.
- ✔ LRTM line count dropped by 68% (from 5124 to 1609).

Coming up next:

- ✔ Retiring old code.
- ✔ SRTM: two for the price of one and a half.

As of October 2016, refactoring the code.

Results so far:

- ✔ Fully vectorized long-wave code.
- ✔ Long-wave computations are fully data-driven.
- ✔ Performance increase of x2, probably more.
- ✔ Code redundancy dramatically reduced.
- ✔ LRTM line count dropped by 68% (from 5124 to 1609).

Coming up next:

- ✔ Retiring old code.
- ✔ SRTM: two for the price of one and a half.
- ✔ Cross-validation with other radiation models.

As of October 2016, refactoring the code.

Results so far:

- ✔ Fully vectorized long-wave code.
- ✔ Long-wave computations are fully data-driven.
- ✔ Performance increase of x2, probably more.
- ✔ Code redundancy dramatically reduced.
- ✔ LRTM line count dropped by 68% (from 5124 to 1609).

Coming up next:

- ✔ Retiring old code.
- ✔ SRTM: two for the price of one and a half.
- ✔ Cross-validation with other radiation models.
- ✔ Replacing underlying data.

As of October 2016, refactoring the code.

Results so far:

- ✔ Fully vectorized long-wave code.
- ✔ Long-wave computations are fully data-driven.
- ✔ Performance increase of x2, probably more.
- ✔ Code redundancy dramatically reduced.
- ✔ LRTM line count dropped by 68% (from 5124 to 1609).

Coming up next:

- ✔ Retiring old code.
- ✔ SRTM: two for the price of one and a half.
- ✔ Cross-validation with other radiation models.
- ✔ Replacing underlying data.
- ✔ Performance portability.

As of October 2016, refactoring the code.

Results so far:

- ✔ Fully vectorized long-wave code.
- ✔ Long-wave computations are fully data-driven.
- ✔ Performance increase of x2, probably more.
- ✔ Code redundancy dramatically reduced.
- ✔ LRTM line count dropped by 68% (from 5124 to 1609).

Coming up next:

- ✔ Retiring old code.
- ✔ SRTM: two for the price of one and a half.
- ✔ Cross-validation with other radiation models.
- ✔ Replacing underlying data.
- ✔ Performance portability.

ETOC?

As of October 2016, refactoring the code.

Results so far:

- ✔ Fully vectorized long-wave code.
- ✔ Long-wave computations are fully data-driven.
- ✔ Performance increase of x2, probably more.
- ✔ Code redundancy dramatically reduced.
- ✔ LRTM line count dropped by 68% (from 5124 to 1609).

Coming up next:

- ✔ Retiring old code.
- ✔ SRTM: two for the price of one and a half.
- ✔ Cross-validation with other radiation models.
- ✔ Replacing underlying data.
- ✔ Performance portability.

ETOC? TBD

Thank you for your time and attention.

gustavo.hime@mpimet.mpg.de

A kind of a problem

```
   !-----------------------------------------------
   !
   ! Floating point section
   ! ----------------------
   !
   INTEGER, PARAMETER :: ps =   6
   INTEGER, PARAMETER :: rs =  37
   !
   INTEGER, PARAMETER :: pd =  12 !!! SHOULD BE 15!!!
   INTEGER, PARAMETER :: rd = 307
   !!< single precision
   INTEGER, PARAMETER :: sp = SELECTED_REAL_KIND(ps,rs)
   !!< double precision
   INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND(pd,rd)
   !!< selected working precision
   INTEGER, PARAMETER :: wp = dp
   !
#ifdef __MIXED_PRECISION
   INTEGER, PARAMETER :: vp = sp
#else
   INTEGER, PARAMETER :: vp = wp
#endif
```

A roundabout way to arrive at the IEEE standard.

A kind of a problem

```
   !-----------------------------------------------
   !
   ! Floating point section
   ! ----------------------
   !
   INTEGER, PARAMETER :: ps =   6
   INTEGER, PARAMETER :: rs =  37
   !
   INTEGER, PARAMETER :: pd =  12 !!! SHOULD BE 15!!!
   INTEGER, PARAMETER :: rd = 307
   !!< single precision
   INTEGER, PARAMETER :: sp = SELECTED_REAL_KIND(ps,rs)
   !!< double precision
   INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND(pd,rd)
   !!< selected working precision
   INTEGER, PARAMETER :: wp = dp
   !
#ifdef __MIXED_PRECISION
   INTEGER, PARAMETER :: vp = sp
#else
   INTEGER, PARAMETER :: vp = wp
#endif
```

A roundabout way to arrive at the IEEE standard.

- ✔ Cluttered source (3 or 4 more characters per variable / argument / constant!)

A kind of a problem

```
   !-----------------------------------------------
   !
   ! Floating point section
   ! ----------------------
   !
   INTEGER, PARAMETER :: ps =   6
   INTEGER, PARAMETER :: rs =  37
   !
   INTEGER, PARAMETER :: pd =  12 !!! SHOULD BE 15!!!
   INTEGER, PARAMETER :: rd = 307
   !!< single precision
   INTEGER, PARAMETER :: sp = SELECTED_REAL_KIND(ps,rs)
   !!< double precision
   INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND(pd,rd)
   !!< selected working precision
   INTEGER, PARAMETER :: wp = dp
   !
#ifdef __MIXED_PRECISION
   INTEGER, PARAMETER :: vp = sp
#else
   INTEGER, PARAMETER :: vp = wp
#endif
```

A roundabout way to arrive at the IEEE standard.

- ✔ **Cluttered source (3 or 4 more characters per variable / argument / constant!)**
- ✔ **Default kind can and should be selected and used - as the *default*.**

A kind of a problem

```
   !-----------------------------------------------
   !
   ! Floating point section
   ! ----------------------
   !
   INTEGER, PARAMETER :: ps =   6
   INTEGER, PARAMETER :: rs =  37
   !
   INTEGER, PARAMETER :: pd =  12 !!! SHOULD BE 15!!!
   INTEGER, PARAMETER :: rd = 307
   !!< single precision
   INTEGER, PARAMETER :: sp = SELECTED_REAL_KIND(ps,rs)
   !!< double precision
   INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND(pd,rd)
   !!< selected working precision
   INTEGER, PARAMETER :: wp = dp
   !
#ifdef __MIXED_PRECISION
   INTEGER, PARAMETER :: vp = sp
#else
   INTEGER, PARAMETER :: vp = wp
#endif
```

A roundabout way to arrive at the IEEE standard.

- ✔ **Cluttered source (3 or 4 more characters per variable / argument / constant!)**
- ✔ **Default kind can and should be selected and used - as the *default*.**
- ✔ **Roundabout $\neq$ Simple.**

A kind of a problem

```
  !----------------------------
  !
  ! Floating point section
  ! ---------------------
  !
  INTEGER, PARAMETER :: ps =   6
  INTEGER, PARAMETER :: rs =  37
  !
  INTEGER, PARAMETER :: pd =  15
  INTEGER, PARAMETER :: rd = 307
  !!< single precision
  INTEGER, PARAMETER :: sp = &
    SELECTED_REAL_KIND(ps,rs)
  !!< double precision
  INTEGER, PARAMETER :: dp = &
    SELECTED_REAL_KIND(pd,rd)
  !!< selected working precision
  INTEGER, PARAMETER :: wp = dp
  !
#ifdef __MIXED_PRECISION
  INTEGER, PARAMETER :: vp = sp
#else
  INTEGER, PARAMETER :: vp = wp
#endif
```

A roundabout way to arrive at the IEEE standard.

- ✔ Cluttered source (3 or 4 more characters per variable / argument / constant!)
- ✔ Default kind can and should be selected and used - as the *default*.
- ✔ Roundabout $\neq$ Simple.

All compatibility problems within and outside the codebase remain:

A kind of a problem

```
  !-----------------------------
  !
  ! Floating point section
  ! ---------------------
  !
  INTEGER, PARAMETER :: ps =   6
  INTEGER, PARAMETER :: rs =  37
  !
  INTEGER, PARAMETER :: pd =  15
  INTEGER, PARAMETER :: rd = 307
  !!< single precision
  INTEGER, PARAMETER :: sp = &
    SELECTED_REAL_KIND(ps,rs)
  !!< double precision
  INTEGER, PARAMETER :: dp = &
    SELECTED_REAL_KIND(pd,rd)
  !!< selected working precision
  INTEGER, PARAMETER :: wp = dp
  !
#ifdef __MIXED_PRECISION
  INTEGER, PARAMETER :: vp = sp
#else
  INTEGER, PARAMETER :: vp = wp
#endif
```

A roundabout way to arrive at the IEEE standard.

- ✔ **Cluttered source (3 or 4 more characters per variable / argument / constant!)**
- ✔ **Default kind can and should be selected and used - as the *default*.**
- ✔ **Roundabout $\neq$ Simple.**

All compatibility problems within and outside the codebase remain:

- ✔ **Routine interfaces with mismatching array arguments/parameters.**

A kind of a problem

```
  !-----------------------------
  !
  ! Floating point section
  ! ---------------------
  !
  INTEGER, PARAMETER :: ps =   6
  INTEGER, PARAMETER :: rs =  37
  !
  INTEGER, PARAMETER :: pd =  15
  INTEGER, PARAMETER :: rd = 307
  !!< single precision
  INTEGER, PARAMETER :: sp = &
    SELECTED_REAL_KIND(ps,rs)
  !!< double precision
  INTEGER, PARAMETER :: dp = &
    SELECTED_REAL_KIND(pd,rd)
  !!< selected working precision
  INTEGER, PARAMETER :: wp = dp
  !
#ifdef __MIXED_PRECISION
  INTEGER, PARAMETER :: vp = sp
#else
  INTEGER, PARAMETER :: vp = wp
#endif
```

A roundabout way to arrive at the IEEE standard.

- ✔ **Cluttered source (3 or 4 more characters per variable / argument / constant!)**
- ✔ **Default kind can and should be selected and used - as the *default*.**
- ✔ **Roundabout $\neq$ Simple.**

All compatibility problems within and outside the codebase remain:

- ✔ **Routine interfaces with mismatching array arguments/parameters.**
- ✔ **Duplication of code for different precisions.**

A kind of a problem

```
  ! Floating point section
  INTEGER, PARAMETER :: &
    !!< single precision
    sp = SELECTED_REAL_KIND(6, 37), &
    !!< double precision
    dp = SELECTED_REAL_KIND(15, 307),&
    !!< working precision
    wp = dp, &
#ifdef __MIXED_PRECISION
    vp = sp
#else
    vp = wp
#endif
```

Attain the Zen.

A roundabout way to arrive at the IEEE standard.

- ✔ Cluttered source (3 or 4 more characters per variable / argument / constant!)
- ✔ Default kind can and should be selected and used - as the *default*.
- ✔ Roundabout $\neq$ Simple.

All compatibility problems within and outside the codebase remain:

- ✔ Routine interfaces with mismatching array arguments/parameters.
- ✔ Duplication of code for different precisions.

How to not make a library

```
  SUBROUTINE solar_parameters(decl_sun, dist_sun, time_of_day                    &
!!$   &                     ,sinlon, sinlat, coslon, coslat                    &
      &                      ,p_patch                                          &
      &                      ,flx_ratio, cos_mu0, daylght_frc)

    REAL(wp), INTENT(in)  :: &
         decl_sun,           & !< delination of the sun
         dist_sun,           & !< distance from the sun in astronomical units
         time_of_day           !< time_of_day (in radians)
    TYPE(t_patch), INTENT(in) ::      p_patch
!!$         sinlon(:,:),        & !< sines of longitudes
!!$         sinlat(:,:),        & !< and latitudes
!!$         coslon(:,:),        & !< cosines of longitudes
!!$         coslat(:,:)           !< and latitudes
    REAL(wp), INTENT(out) :: &
         flx_ratio,          & !< ratio of actual to average solar constant
         cos_mu0(:,:),       & !< cos_mu_0, cosine of the solar zenith angle
         daylght_frc(:,:)      !< daylight fraction (0 or 1) with diurnal cycle
```

An innocent argument of type `t_patch`...

So back to making a library

```
nprom=nproma
npromz=p_patch%npromz_c
nblks=p_patch%nblks_c
ALLOCATE(sinlon(nprom,nblks))
ALLOCATE(sinlat(nprom,nblks))
ALLOCATE(coslon(nprom,nblks))
ALLOCATE(coslat(nprom,nblks))
sinlon(:,:)=0._wp
sinlat(:,:)=0._wp
coslon(:,:)=0._wp
coslat(:,:)=0._wp
sinlon(1:nprom,1:nblks-1)=SIN(p_patch%cells%center(1:nprom,1:nblks-1)%lon)
sinlat(1:nprom,1:nblks-1)=SIN(p_patch%cells%center(1:nprom,1:nblks-1)%lat)
coslon(1:nprom,1:nblks-1)=COS(p_patch%cells%center(1:nprom,1:nblks-1)%lon)
coslat(1:nprom,1:nblks-1)=COS(p_patch%cells%center(1:nprom,1:nblks-1)%lat)
sinlon(1:npromz,nblks)=SIN(p_patch%cells%center(1:npromz,nblks)%lon)
sinlat(1:npromz,nblks)=SIN(p_patch%cells%center(1:npromz,nblks)%lat)
coslon(1:npromz,nblks)=COS(p_patch%cells%center(1:npromz,nblks)%lon)
coslat(1:npromz,nblks)=COS(p_patch%cells%center(1:npromz,nblks)%lat)
```

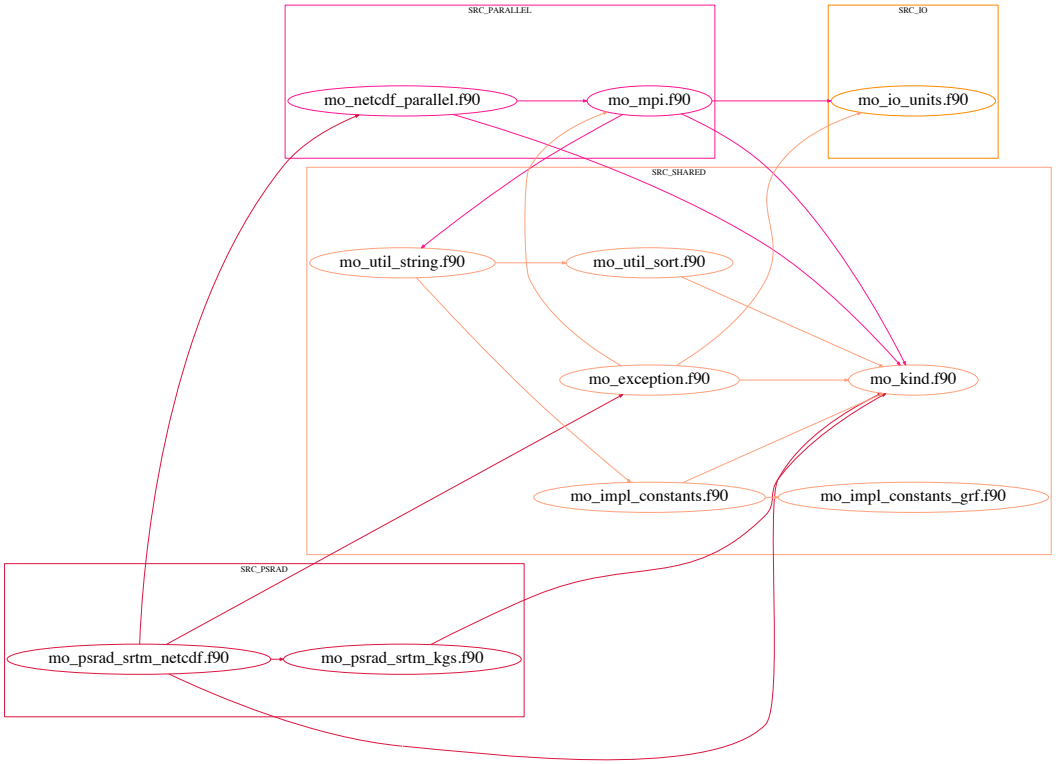
An innocent argument of type `t_patch`...

From which a very particular information is extracted.

Asking for trouble

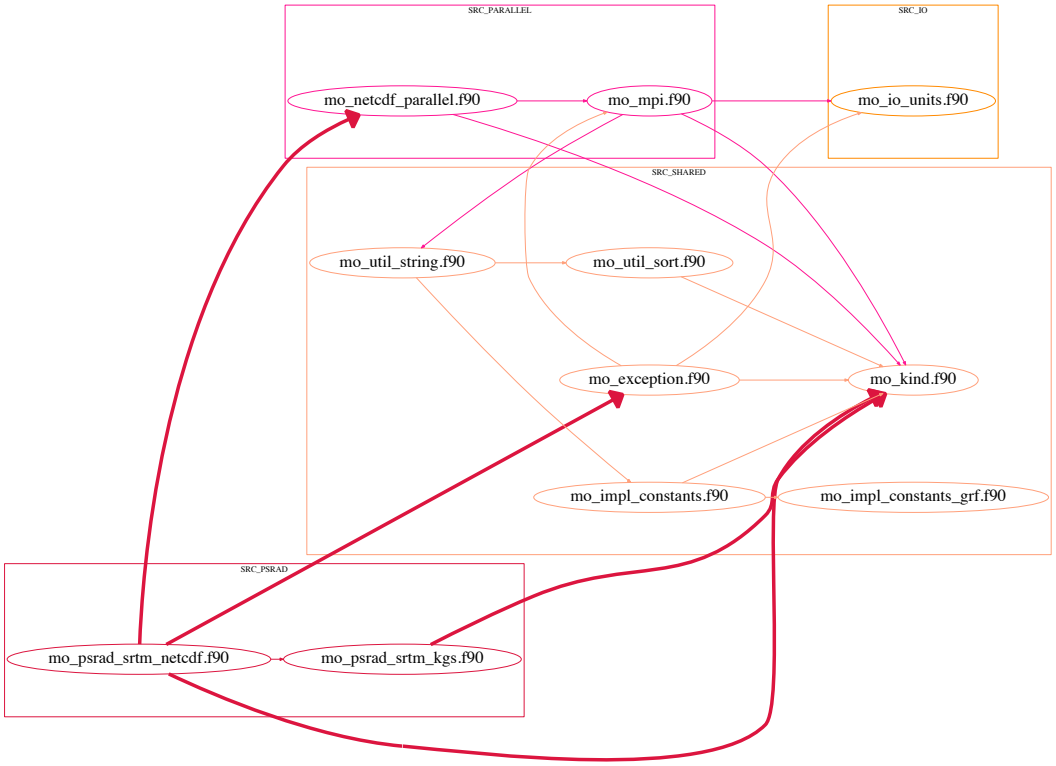
```
REAL (wp), PARAMETER ::  euler      = 2.71828182845904523536028747135266250_wp
REAL (wp), PARAMETER ::  log2e      = 1.44269504088896340735992468100189214_wp
REAL (wp), PARAMETER ::  log10e     = 0.43429448190325182765112891891660508 2_wp
REAL (wp), PARAMETER ::  ln2        = 0.69314718055994530941723212145817656 8_wp
REAL (wp), PARAMETER ::  ln10       = 2.30258509299404568401799145468436421_wp
!
REAL (wp), PARAMETER ::  pi         = 3.14159265358979323846264338327950288_wp
!
REAL (wp), PARAMETER ::  pi_2       = 1.57079632679489661923132169163975144_wp
REAL (wp), PARAMETER ::  pi_4       = 0.78539816339744830961566084581987572 1_wp
REAL (wp), PARAMETER ::  rpi        = 0.31830988618379067153776752674502872 4_wp
REAL (wp), PARAMETER ::  rpi_2      = 0.63661977236758134307553505349005744 8_wp
REAL (wp), PARAMETER ::  rsqrtpi_2  = 1.12837916709551257389615890312154517_wp
REAL (wp), PARAMETER ::  sqrt2      = 1.41421356237309504880168872420969808_wp
REAL (wp), PARAMETER ::  sqrt1_2    = 0.70710678118654752440084436210484903 9_wp
REAL (wp), PARAMETER ::  sqrt3      = 1.73205080756887729352744634150587 23_wp
REAL (wp), PARAMETER ::  sqrt1_3    = 0.57735026918962576450914878050195 75_wp
REAL (wp), PARAMETER ::  cos45      = sqrt1_2
REAL (wp), PARAMETER ::  one_third  = 1.0_wp/3.0_wp
```

Under the skin



Taking a closer look...

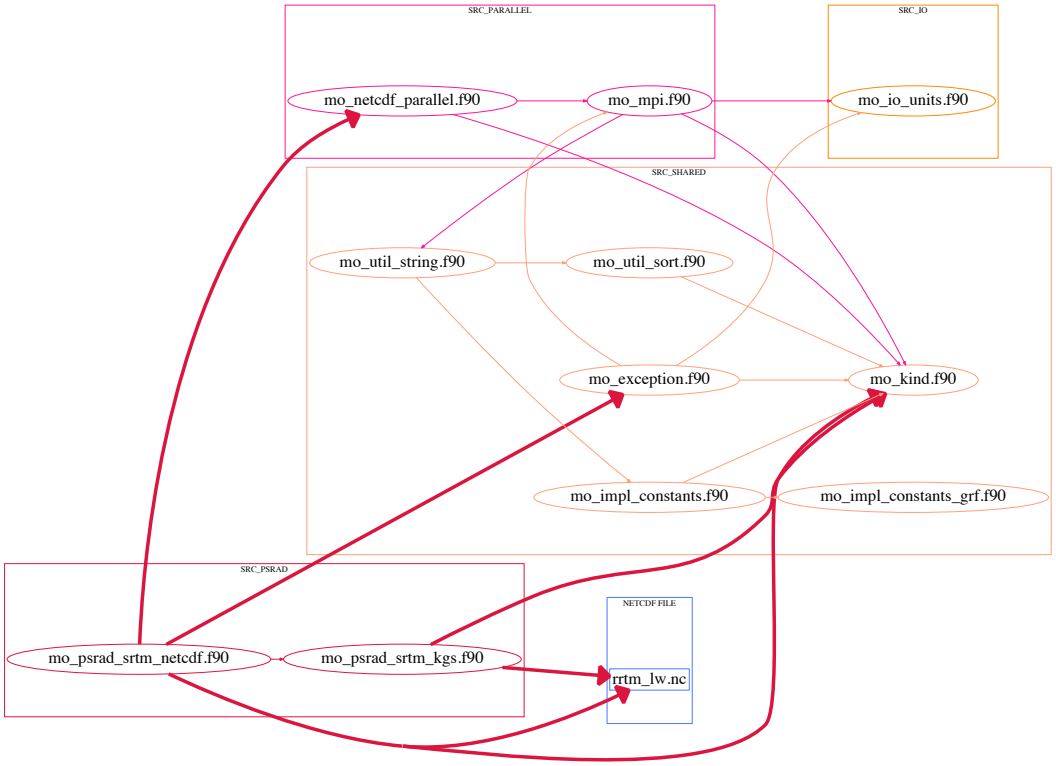
Under the skin



Taking a closer look...

...there are not so many dependencies...

Under the skin

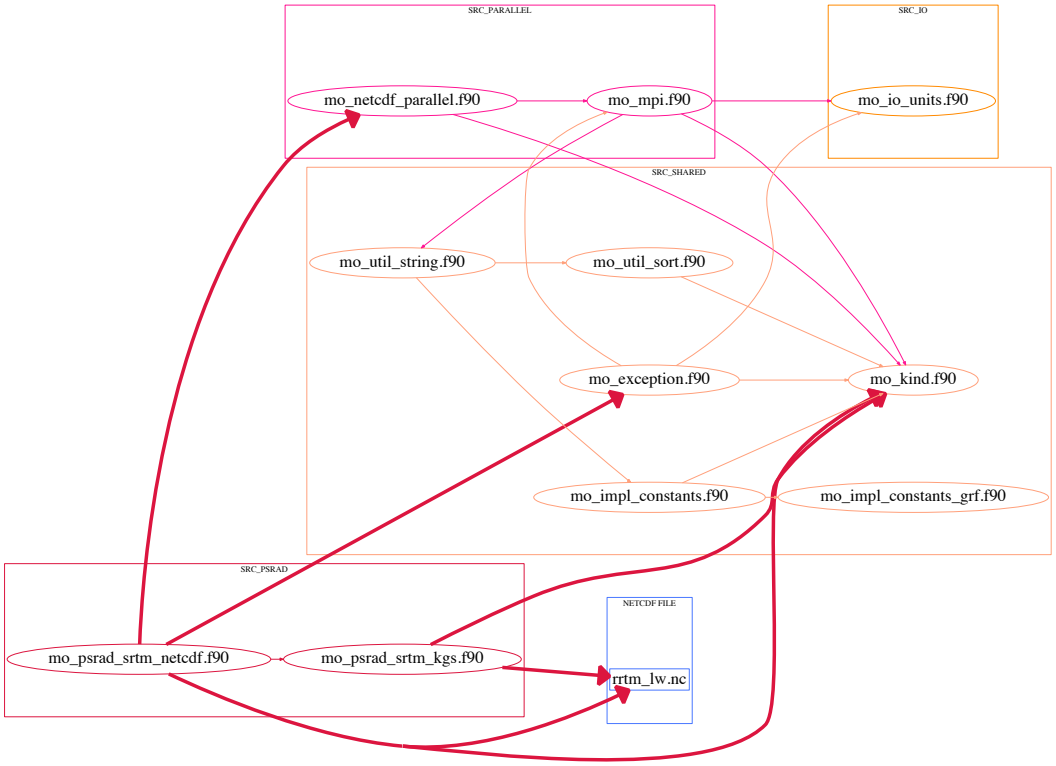


Taking a closer look...

...there are not so many dependencies...

...still more than meets the eye.

Under the skin



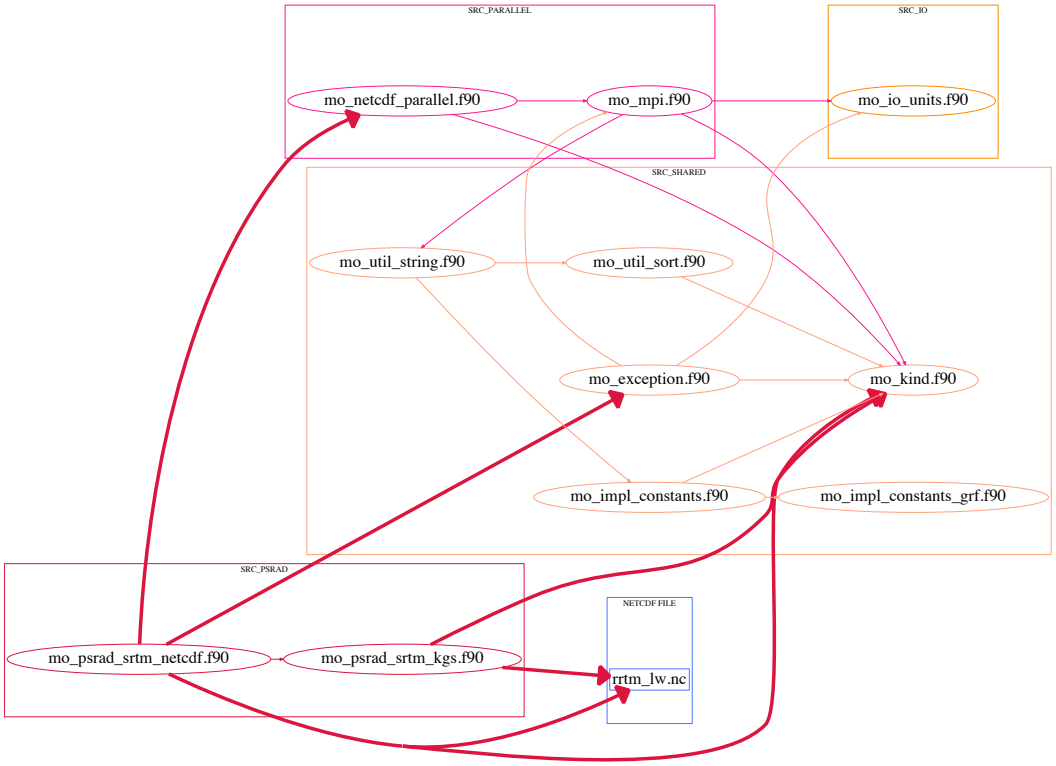
Taking a closer look...

...there are not so many dependencies...

...still more than meets the eye.

Data structures reflect input file format.

Under the skin



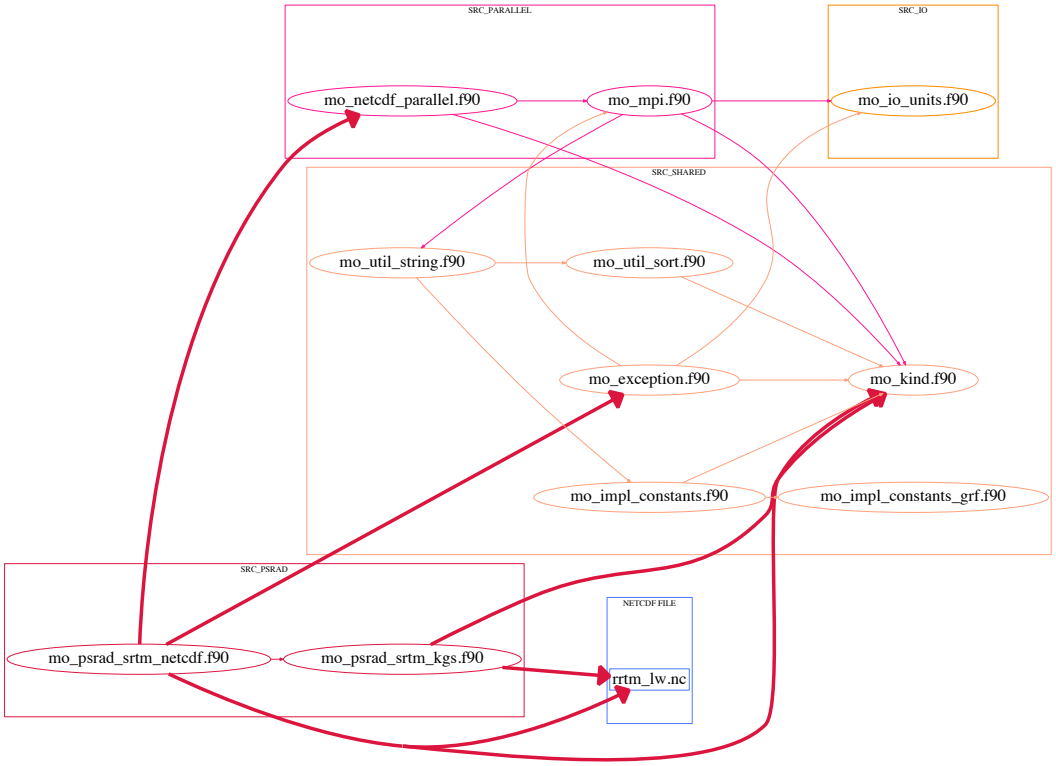
Taking a closer look...

...there are not so many dependencies...

...still more than meets the eye.

Data structures reflect input file format.
Input routines reflect data structures.

Under the skin



Taking a closer look...

...there are not so many dependencies...

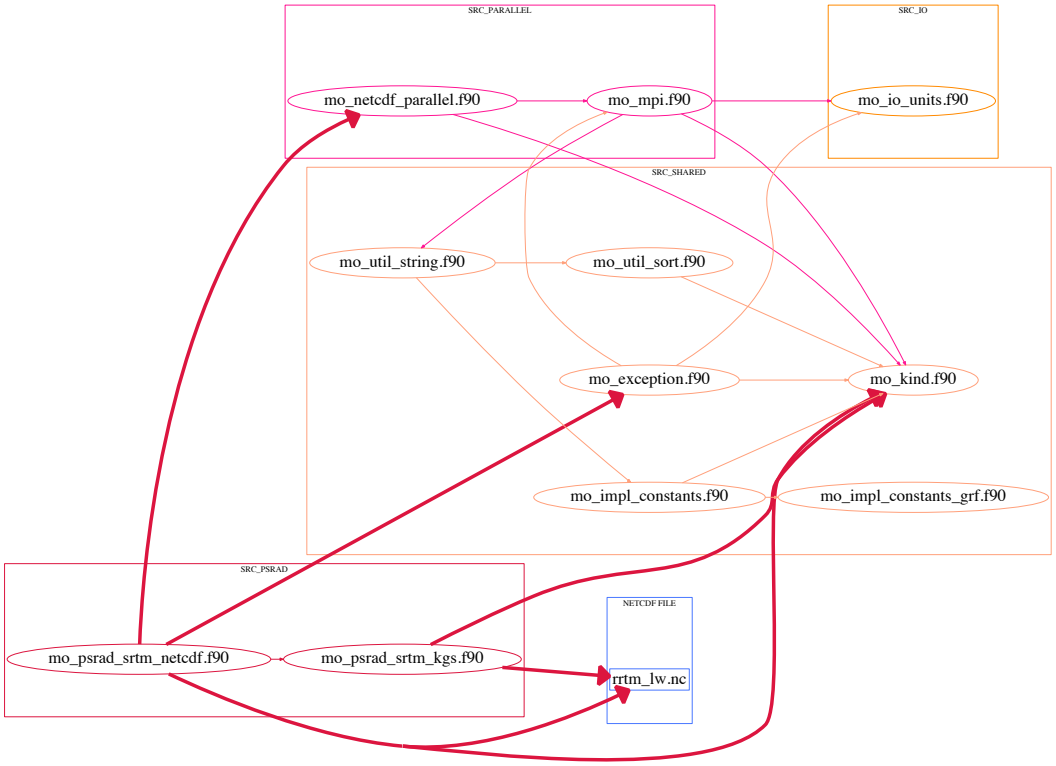
...still more than meets the eye.

Data structures reflect input file format.

~~Input routines reflect data structures.~~

ALL routines reflect data structures.

Under the skin



Taking a closer look...

...there are not so many dependencies...

...still more than meets the eye.

Data structures reflect input file format. $\Leftarrow$ Contingent.

~~Input routines reflect data structures.~~

ALL routines reflect data structures. $\Leftarrow$ Tautological.

Under the skin

Variable	Long wave							Short wave						
	i	j	k	l	m	n	Size	i	j	k	l	m	n	Size
AbsorberAmountMLS	—	—	—	12	—	—	12	—	—	—	—	—	—	—
AbsorptionCoefficientsLowerAtmos	9	19	16	12	16	2	1M	9	19	16	12	14	2	920K
AbsorptionCoefficientsUpperAtmos	5	19	16	12	16	2	580K	5	19	16	12	14	2	510K
H2OForeignAbsorptionCoefficients	4	—	16	—	2	—	64K	—	—	—	—	—	—	—
H2OSelfAbsorptionCoefficients	10	—	16	—	2	—	320K	10	—	14	—	2	—	280K
IntegratedPlanckFunction	181	—	—	—	16	—	2.8K	—	—	—	—	—	—	—
IntegratedPlanckFunctionBand16	181	—	—	—	—	—	181	—	—	—	—	—	—	—
KeySpeciesAbsorptionCoefficientsLowerAtmos	9	5	16	2	2	2	5.7K	9	5	16	2	2	2	5.7K
KeySpeciesAbsorptionCoefficientsUpperAtmos	5	5	16	2	2	2	3.2K	5	5	16	2	2	2	3.2K
PlanckFractionLowerAtmos	9	—	—	—	—	2	18	—	—	—	—	—	—	—
PlanckFractionUpperAtmos	5	—	—	—	—	2	10	—	—	—	—	—	—	—
H2OForeignAbsorptionCoefficientsLowerAtmos	—	—	—	—	—	—	—	3	—	—	—	14	2	84
H2OForeignAbsorptionCoefficientsUpperAtmos	—	—	—	—	—	—	—	2	—	—	—	14	2	56
RayleighExtinctionCoefficientsLowerAtmos	—	—	—	—	—	—	—	9	—	—	—	14	2	252
RayleighExtinctionCoefficientsUpperAtmos	—	—	—	—	—	—	—	5	—	—	—	14	2	140
SolarSourceFunctionLowerAtmos	—	—	—	—	—	—	—	9	—	—	—	14	2	252
SolarSourceFunctionUpperAtmos	—	—	—	—	—	—	—	5	—	—	—	14	2	140

Under the skin

Variable	Long wave							Short wave						
	i	j	k	l	m	n	Size	i	j	k	l	m	n	Size
AbsorberAmountMLS	—	—	—	12	—	—	12	—	—	—	—	—	—	—
AbsorptionCoefficientsLowerAtmos	9	19	16	12	16	2	1M	9	19	16	12	14	2	920K
AbsorptionCoefficientsUpperAtmos	5	19	16	12	16	2	580K	5	19	16	12	14	2	510K
H2OForeignAbsorptionCoefficients	4	—	16	—	2	—	64K	—	—	—	—	—	—	—
H2OSelfAbsorptionCoefficients	10	—	16	—	2	—	320K	10	—	14	—	2	—	280K
IntegratedPlanckFunction	181	—	—	—	16	—	2.8K	—	—	—	—	—	—	—
IntegratedPlanckFunctionBand16	181	—	—	—	—	—	181	—	—	—	—	—	—	—
KeySpeciesAbsorptionCoefficientsLowerAtmos	9	5	16	2	2	2	5.7K	9	5	16	2	2	2	5.7K
KeySpeciesAbsorptionCoefficientsUpperAtmos	5	5	16	2	2	2	3.2K	5	5	16	2	2	2	3.2K
PlanckFractionLowerAtmos	9	—	—	—	—	2	18	—	—	—	—	—	—	—
PlanckFractionUpperAtmos	5	—	—	—	—	2	10	—	—	—	—	—	—	—
H2OForeignAbsorptionCoefficientsLowerAtmos	—	—	—	—	—	—	—	3	—	—	—	14	2	84
H2OForeignAbsorptionCoefficientsUpperAtmos	—	—	—	—	—	—	—	2	—	—	—	14	2	56
RayleighExtinctionCoefficientsLowerAtmos	—	—	—	—	—	—	—	9	—	—	—	14	2	252
RayleighExtinctionCoefficientsUpperAtmos	—	—	—	—	—	—	—	5	—	—	—	14	2	140
SolarSourceFunctionLowerAtmos	—	—	—	—	—	—	—	9	—	—	—	14	2	252
SolarSourceFunctionUpperAtmos	—	—	—	—	—	—	—	5	—	—	—	14	2	140

Some 3.7M elements.

Under the skin

Variable	Long wave							Short wave						
	i	j	k	l	m	n	Size	i	j	k	l	m	n	Size
AbsorberAmountMLS	—	—	—	12	—	—	12	—	—	—	—	—	—	—
AbsorptionCoefficientsLowerAtmos	9	19	16	12	16	2	1M	9	19	16	12	14	2	920K
AbsorptionCoefficientsUpperAtmos	5	19	16	12	16	2	580K	5	19	16	12	14	2	510K
H2OForeignAbsorptionCoefficients	4	—	16	—	2	—	64K	—	—	—	—	—	—	—
H2OSelfAbsorptionCoefficients	10	—	16	—	2	—	320K	10	—	14	—	2	—	280K
IntegratedPlanckFunction	181	—	—	—	16	—	2.8K	—	—	—	—	—	—	—
IntegratedPlanckFunctionBand16	181	—	—	—	—	—	181	—	—	—	—	—	—	—
KeySpeciesAbsorptionCoefficientsLowerAtmos	9	5	16	2	2	2	5.7K	9	5	16	2	2	2	5.7K
KeySpeciesAbsorptionCoefficientsUpperAtmos	5	5	16	2	2	2	3.2K	5	5	16	2	2	2	3.2K
PlanckFractionLowerAtmos	9	—	—	—	—	2	18	—	—	—	—	—	—	—
PlanckFractionUpperAtmos	5	—	—	—	—	2	10	—	—	—	—	—	—	—
H2OForeignAbsorptionCoefficientsLowerAtmos	—	—	—	—	—	—	—	3	—	—	—	14	2	84
H2OForeignAbsorptionCoefficientsUpperAtmos	—	—	—	—	—	—	—	2	—	—	—	14	2	56
RayleighExtinctionCoefficientsLowerAtmos	—	—	—	—	—	—	—	9	—	—	—	14	2	252
RayleighExtinctionCoefficientsUpperAtmos	—	—	—	—	—	—	—	5	—	—	—	14	2	140
SolarSourceFunctionLowerAtmos	—	—	—	—	—	—	—	9	—	—	—	14	2	252
SolarSourceFunctionUpperAtmos	—	—	—	—	—	—	—	5	—	—	—	14	2	140

Some 3.7M elements. Some 28.2Mb data.