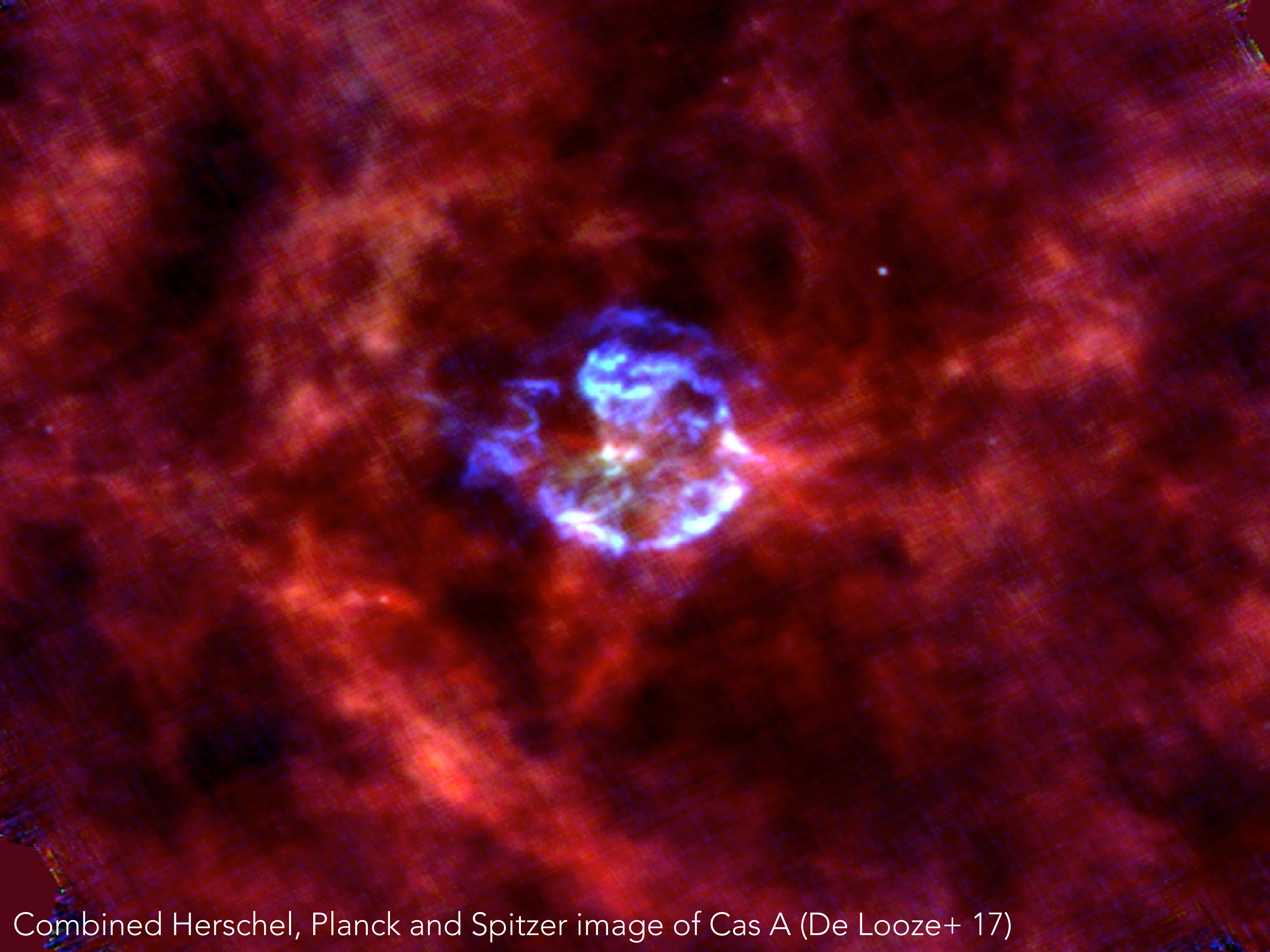
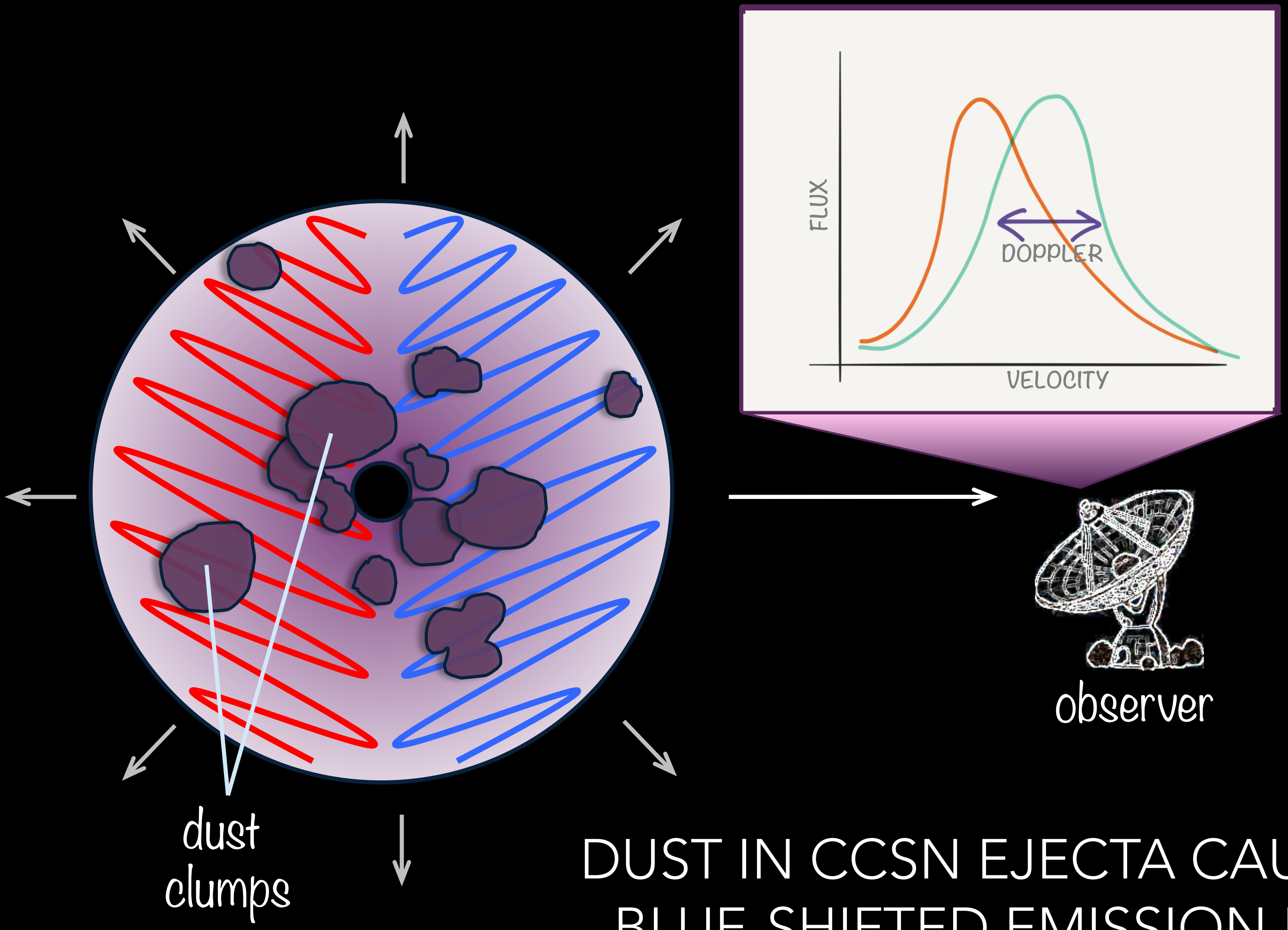
PUTTING THEORY INTO PRACTICE: HOW TO WRITE AN MCRT CODE

Antonia Bevan, UCL

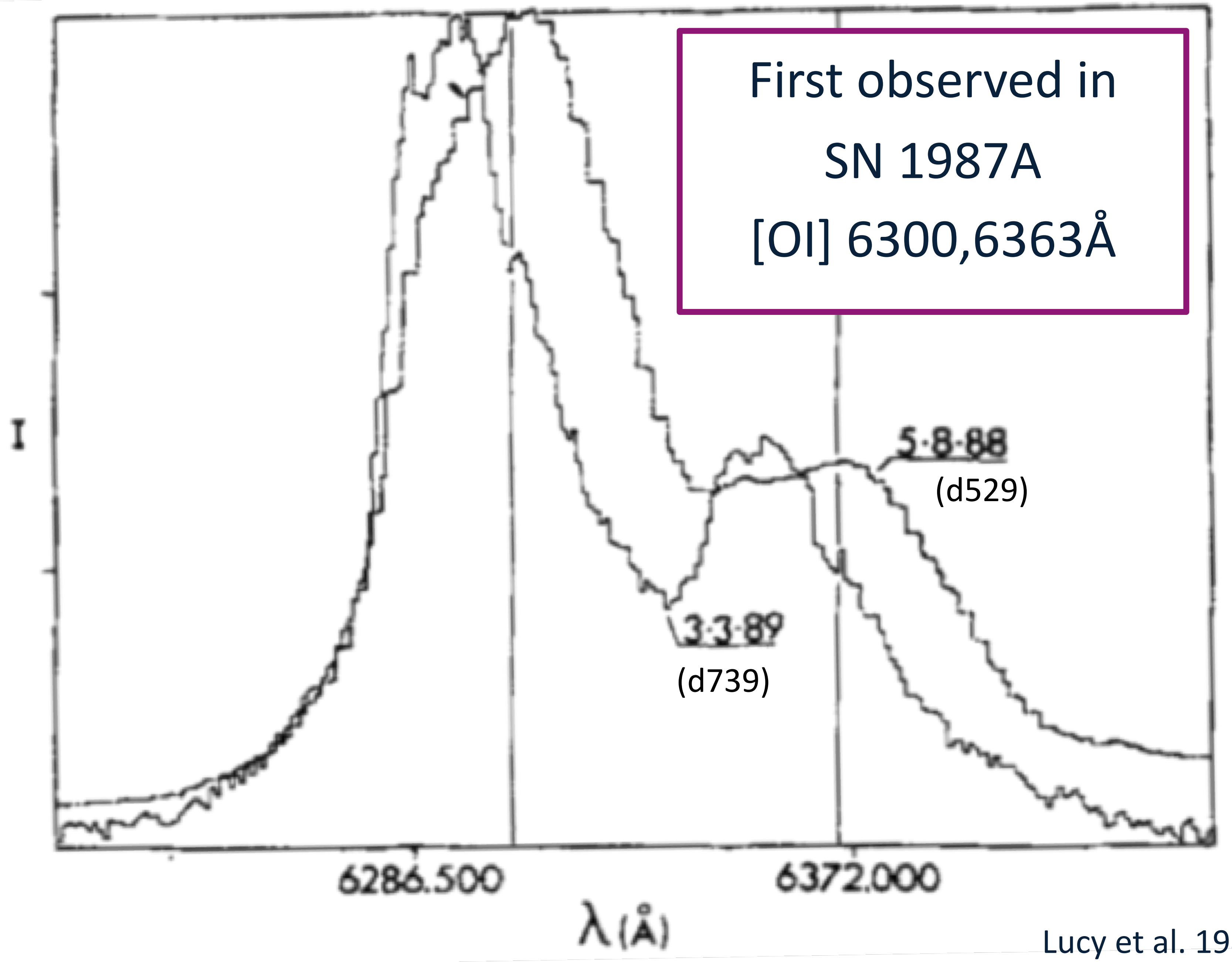
St Andrews Monte Carlo Summer School 2019

Me in October 2012 just after I started my PhD


Combined Herschel, Planck and Spitzer image of Cas A (De Looze+ 17)



DUST IN CCSN EJECTA CAUSES BLUE-SHIFTED EMISSION LINE PROFILES IN OPTICAL AND IR



First observed in

SN 1987A

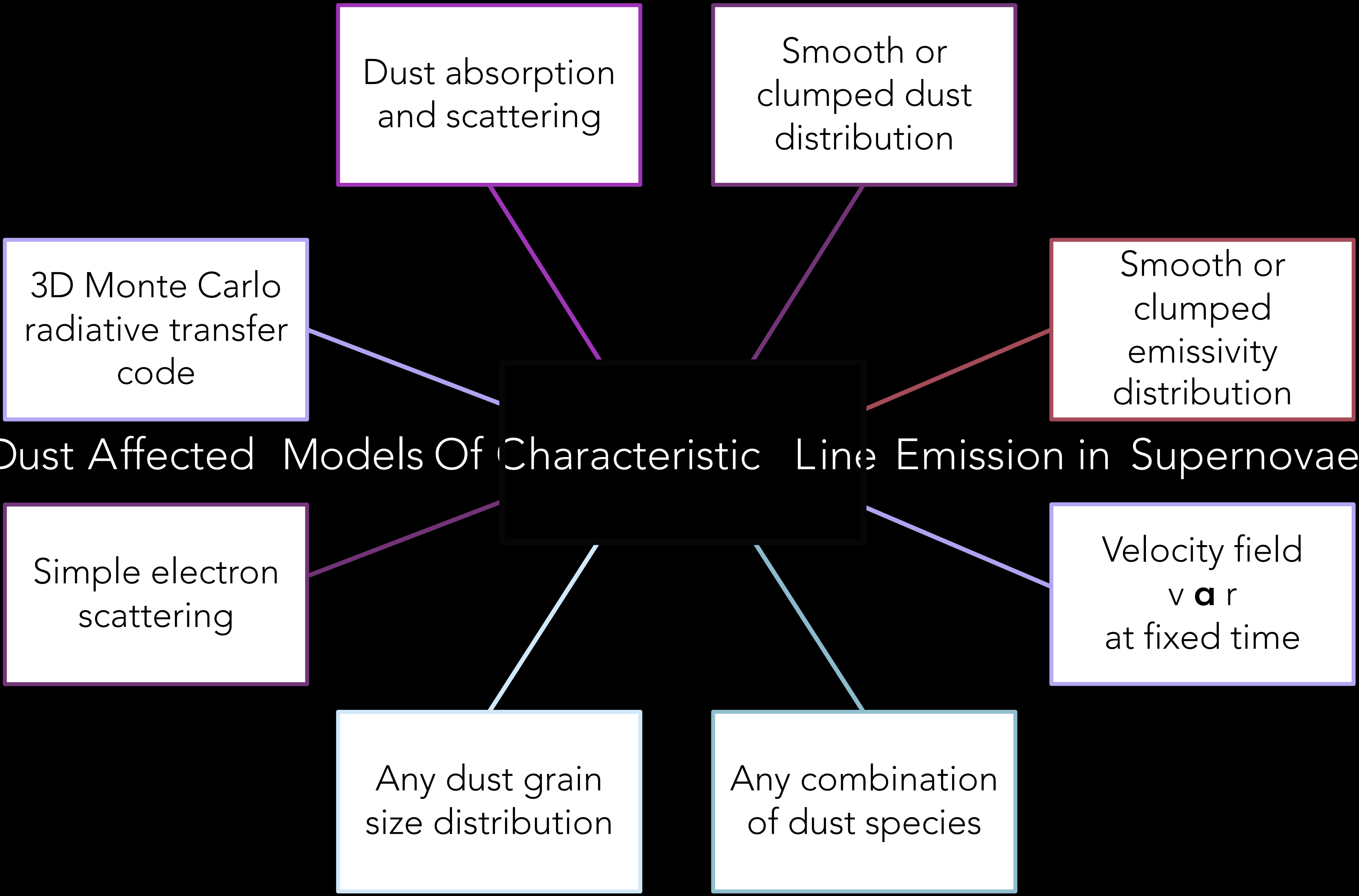
[OI] 6300,6363Å

Lucy et al. 1989

CHALLENGE:
WRITE A MONTE CARLO RADIATIVE TRANSFER CODE THAT WILL…

?

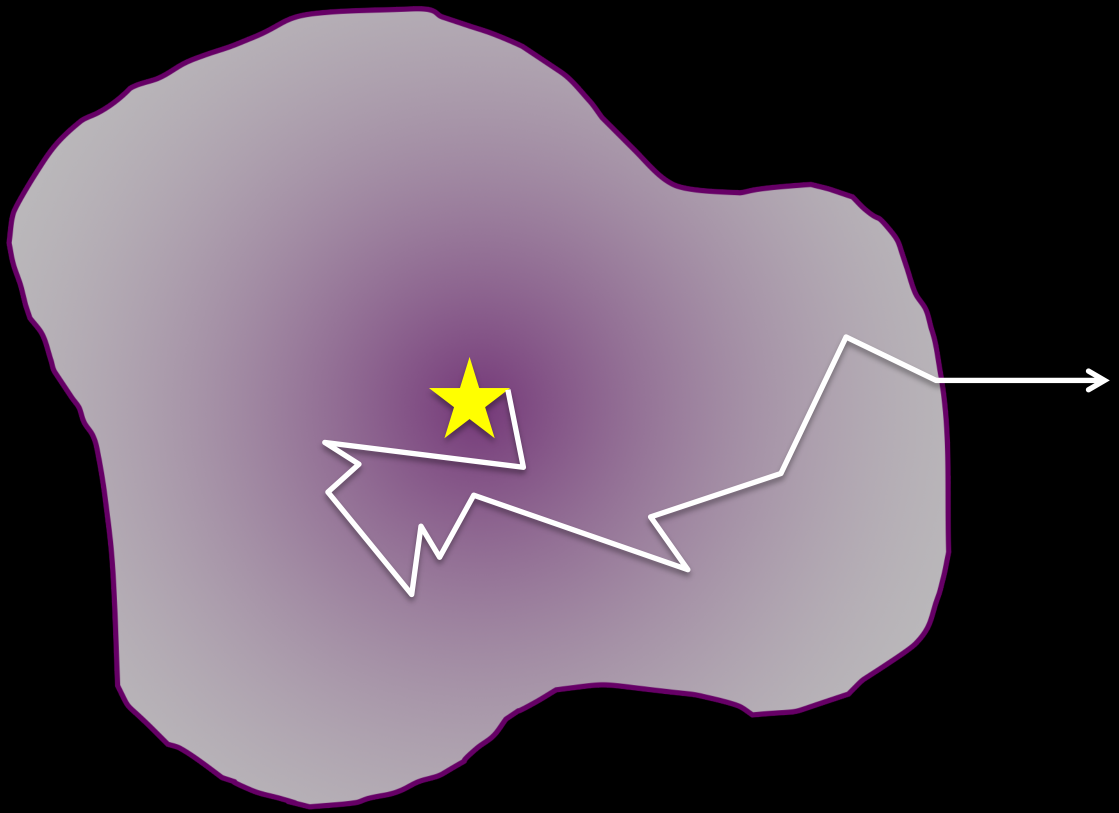
Dust Affected Models Of Characteristic Line Emission in Supernovae


Dust absorption and scattering
Smooth or clumped dust distribution
3D Monte Carlo radiative transfer code
Smooth or clumped emissivity distribution
Simple electron scattering
Velocity field v a r at fixed time
Any dust grain size distribution
Any combination of dust species

ASK YOURSELF QUESTIONS…

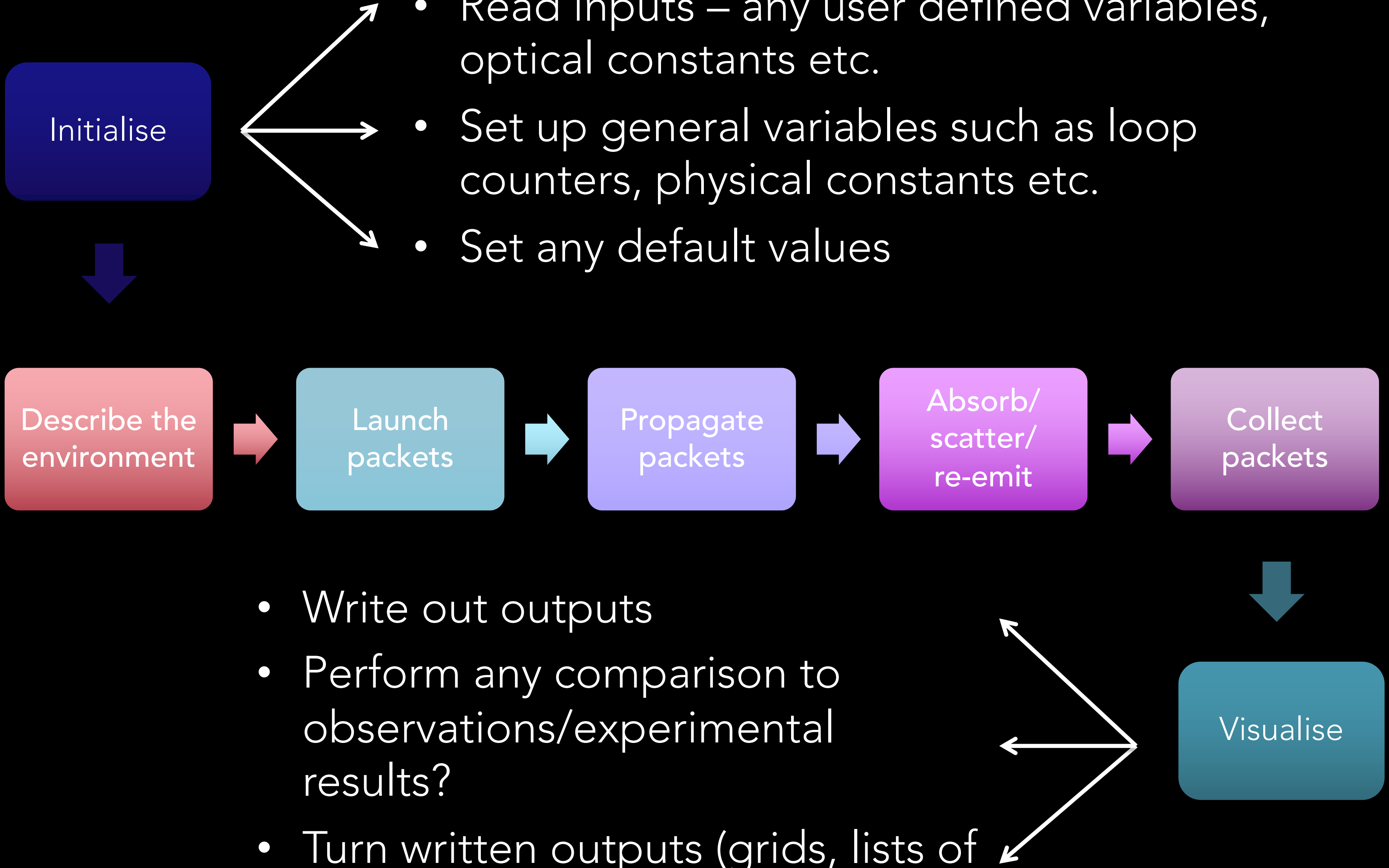
- Where will this go? Will I want to add capacity in future? Is there anything I can do now to make that easier?
- What are the inputs? What are the outputs? What will you do with your outputs?
- Which processes/physics/stats/magic will take you from your inputs to your outputs?
- What can/can't you assume?

MAP OUT YOUR PROBLEM



MCRT codes have the same basic premise but different physics, process and products

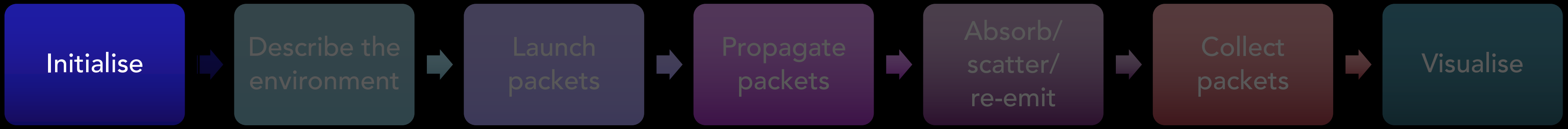
Describe the environment → Launch packets → Propagate packets → Absorb/ scatter/ re-emit → Collect packets



- Read inputs – any user defined variables, optical constants etc.
- Set up general variables such as loop counters, physical constants etc.
- Set any default values

- Write out outputs
- Perform any comparison to observations/experimental results?
- Turn written outputs (grids, lists of numbers etc.) into graphics, images, plots etc.

INITIALISE

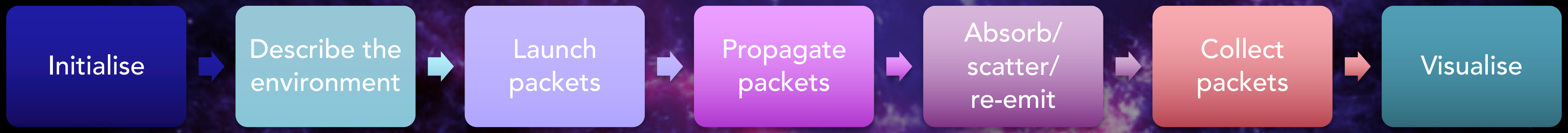

Initialise
Describe the environment
Launch packets
Propagate packets
Absorb/ scatter/ re-emit
Collect packets
Visualise

QUESTIONS TO ASK

What now?

- You're going to need to make some decisions about your tools
  - Fortran or C or Python or… or… or…?
    [you may choose to use multiple]
  - OpenMP or MPI?
  - IDE? text editor?
  - Compiler?
  - Computer…!
  - Which version control – Git, SVN etc.?

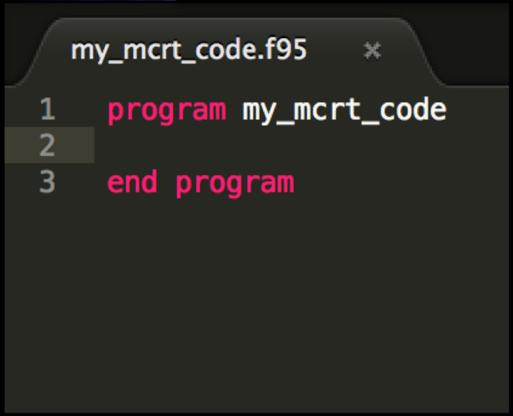
USE VERSION CONTROL



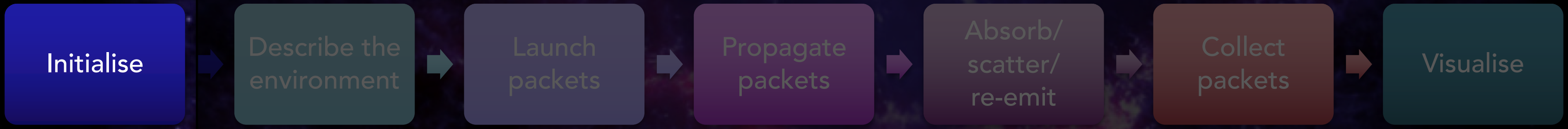
DAMOCLES development:

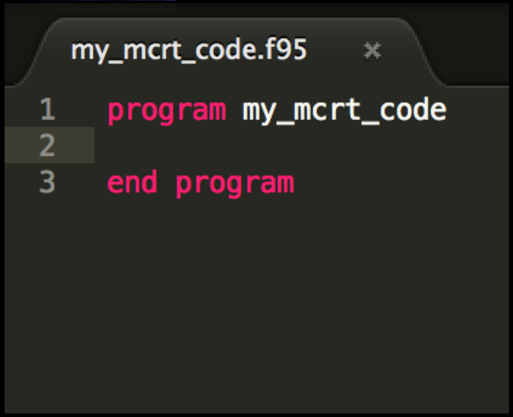
- Fortran 95 + Python
- Tools I use:
  - Eclipse IDE with Photran
  - Sublime Text & emacs for text editing
  - GitHub for version control
  - gcc compilers
  - Develop on my MacBook and run on my MacBook and clusters at UCL

Initialise → Describe the environment → Launch packets → Propagate packets → Absorb/ scatter/ re-emit → Collect packets → Visualise

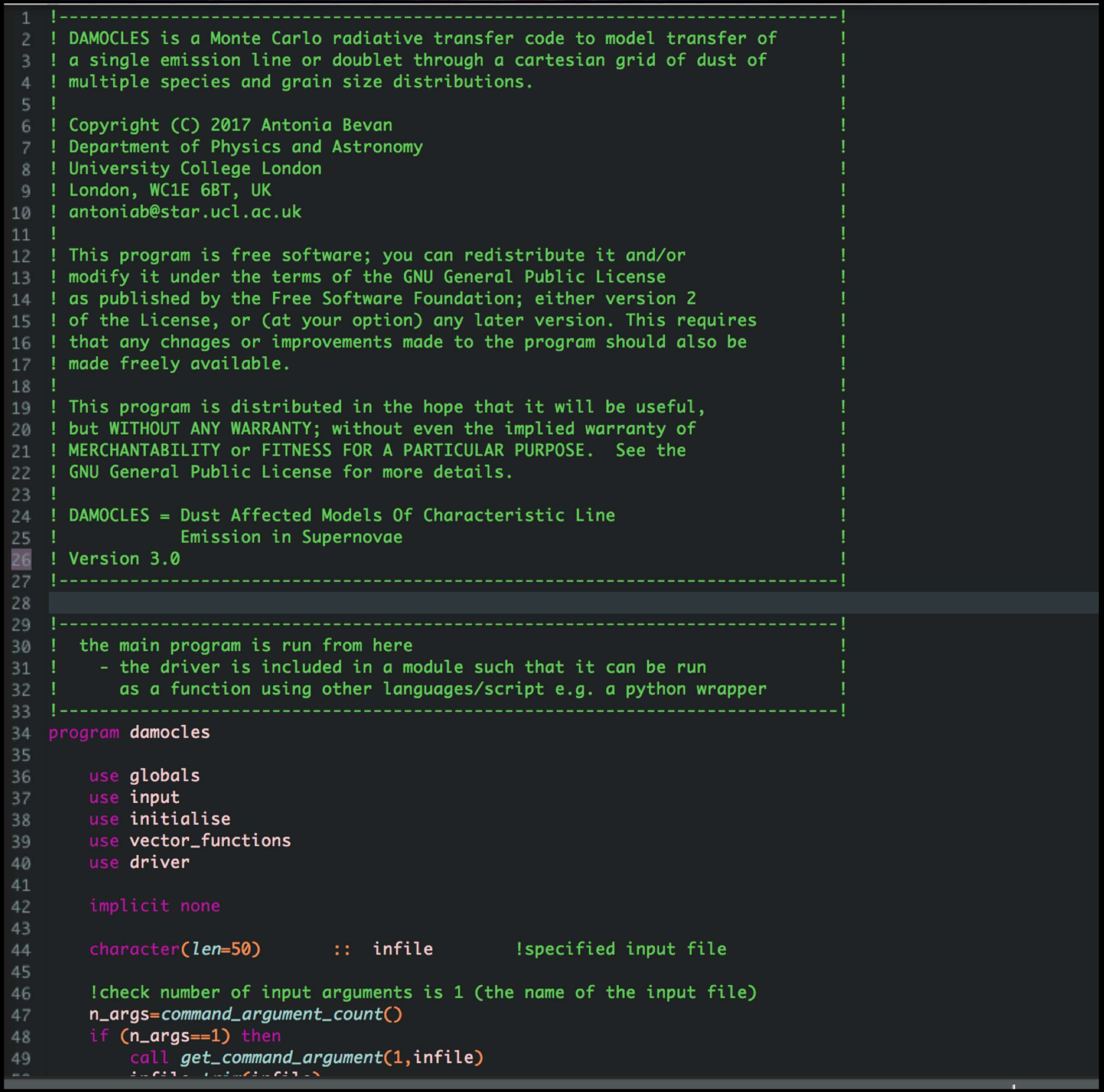

my_mcrt_code.f95

```
program my_mcrt_code

end program
```

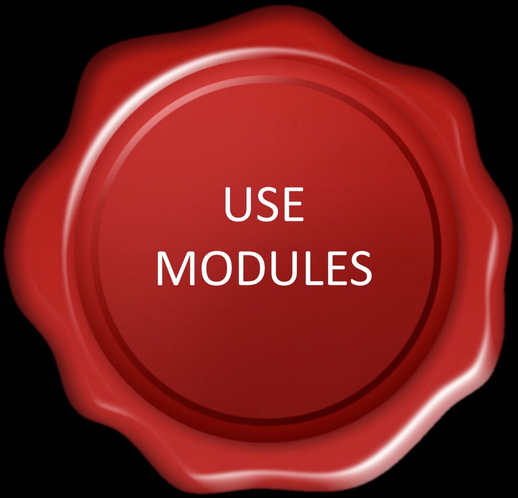


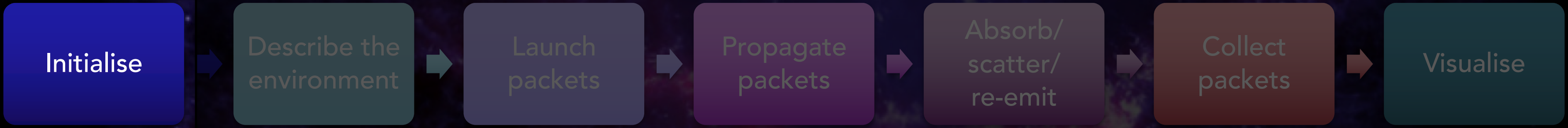


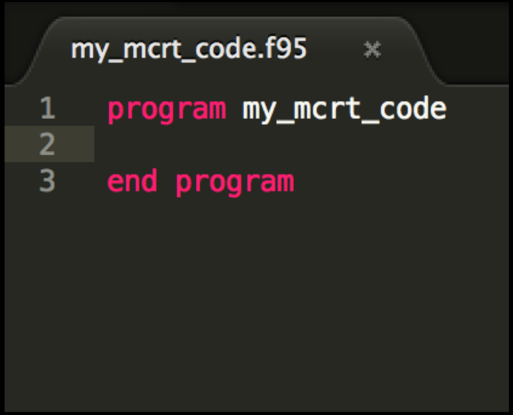
```fortran
program my_mcrt_code

end program
```



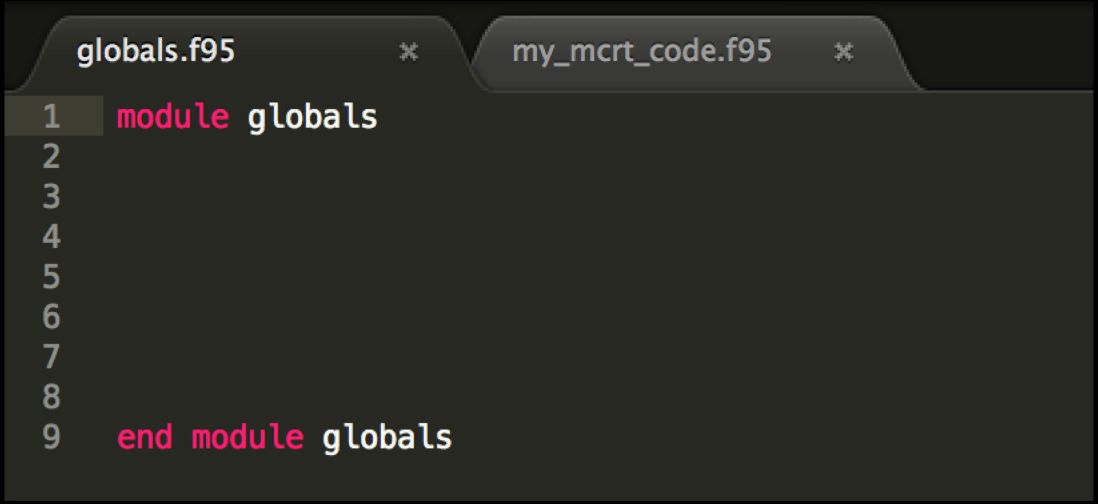
```fortran
!-----------------------------------------------------------------------------!
! DAMOCLES is a Monte Carlo radiative transfer code to model transfer of      !
! a single emission line or doublet through a cartesian grid of dust of       !
! multiple species and grain size distributions.                              !
!                                                                             !
! Copyright (C) 2017 Antonia Bevan                                            !
! Department of Physics and Astronomy                                         !
! University College London                                                   !
! London, WC1E 6BT, UK                                                        !
! antoniab@star.ucl.ac.uk                                                     !
!                                                                             !
! This program is free software; you can redistribute it and/or               !
! modify it under the terms of the GNU General Public License                 !
! as published by the Free Software Foundation; either version 2              !
! of the License, or (at your option) any later version. This requires        !
! that any chnages or improvements made to the program should also be         !
! made freely available.                                                      !
!                                                                             !
! This program is distributed in the hope that it will be useful,             !
! but WITHOUT ANY WARRANTY; without even the implied warranty of              !
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               !
! GNU General Public License for more details.                                !
!                                                                             !
! DAMOCLES = Dust Affected Models Of Characteristic Line                      !
!            Emission in Supernovae                                           !
! Version 3.0                                                                 !
!-----------------------------------------------------------------------------!

!-----------------------------------------------------------------------------!
!  the main program is run from here                                          !
!    - the driver is included in a module such that it can be run             !
!      as a function using other languages/script e.g. a python wrapper      !
!-----------------------------------------------------------------------------!
program damocles

    use globals
    use input
    use initialise
    use vector_functions
    use driver

    implicit none

    character(len=50)       ::  infile        !specified input file

    !check number of input arguments is 1 (the name of the input file)
    n_args=command_argument_count()
    if (n_args==1) then
        call get_command_argument(1,infile)
```



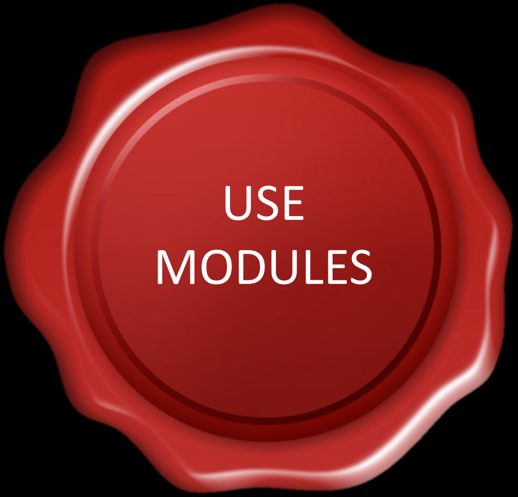



my_mcrt_code.f95

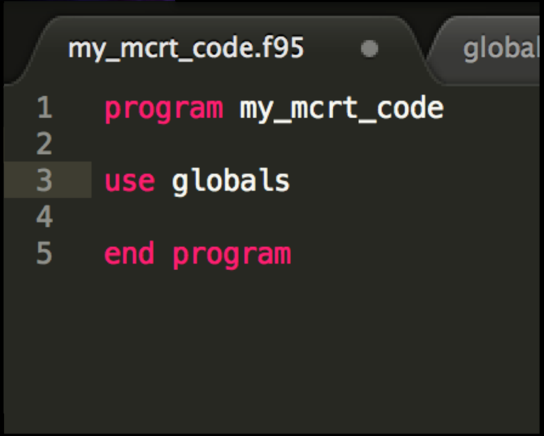
```fortran
program my_mcrt_code

end program
```


globals.f95 | my_mcrt_code.f95

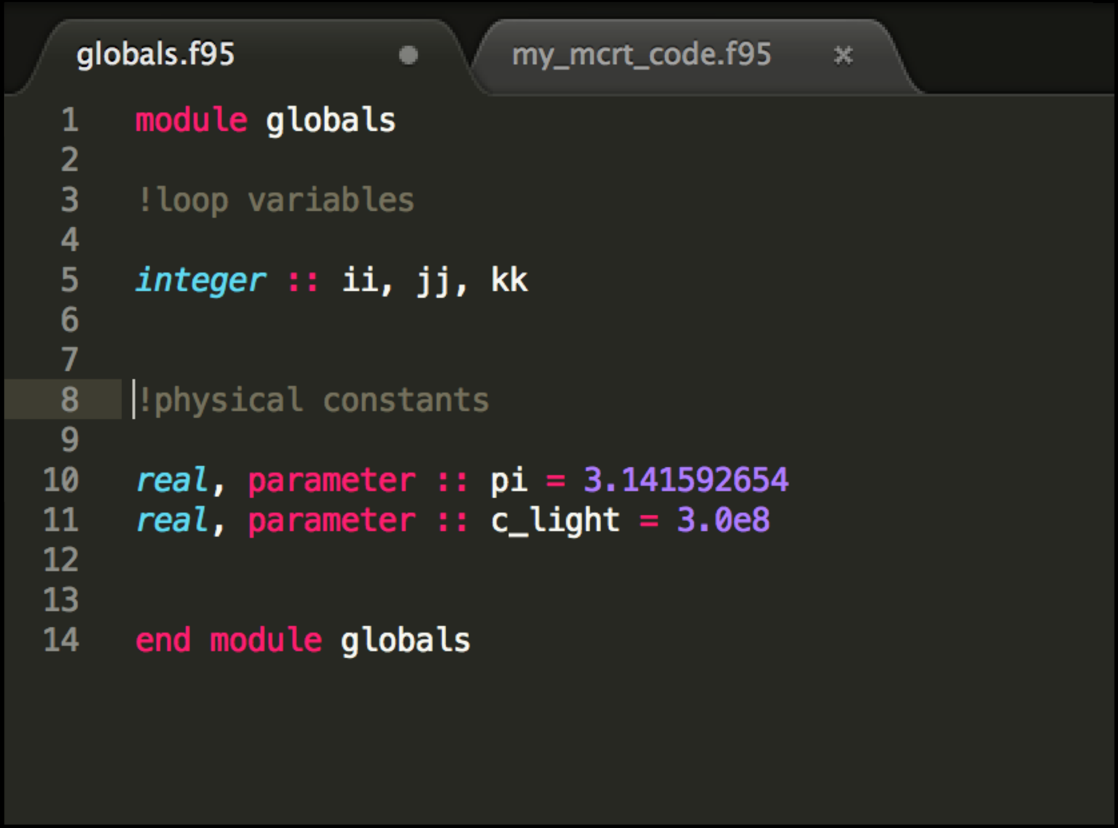
```fortran
module globals







end module globals
```



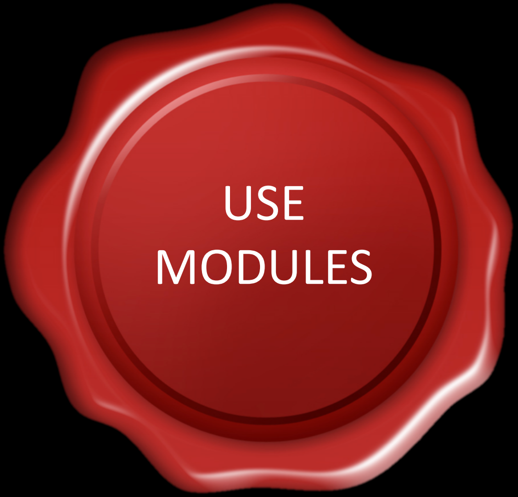
Initialise → Describe the environment → Launch packets → Propagate packets → Absorb/ scatter/ re-emit → Collect packets → Visualise

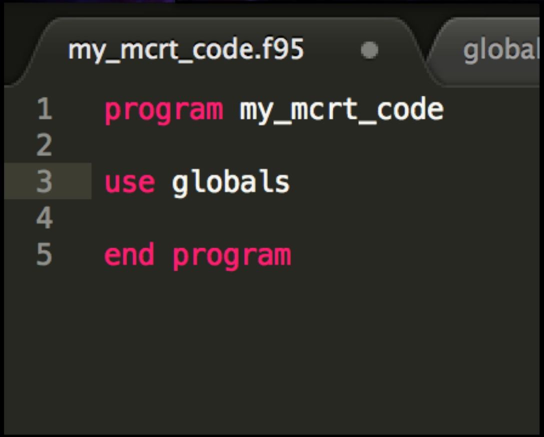


```fortran
program my_mcrt_code

use globals

end program
```

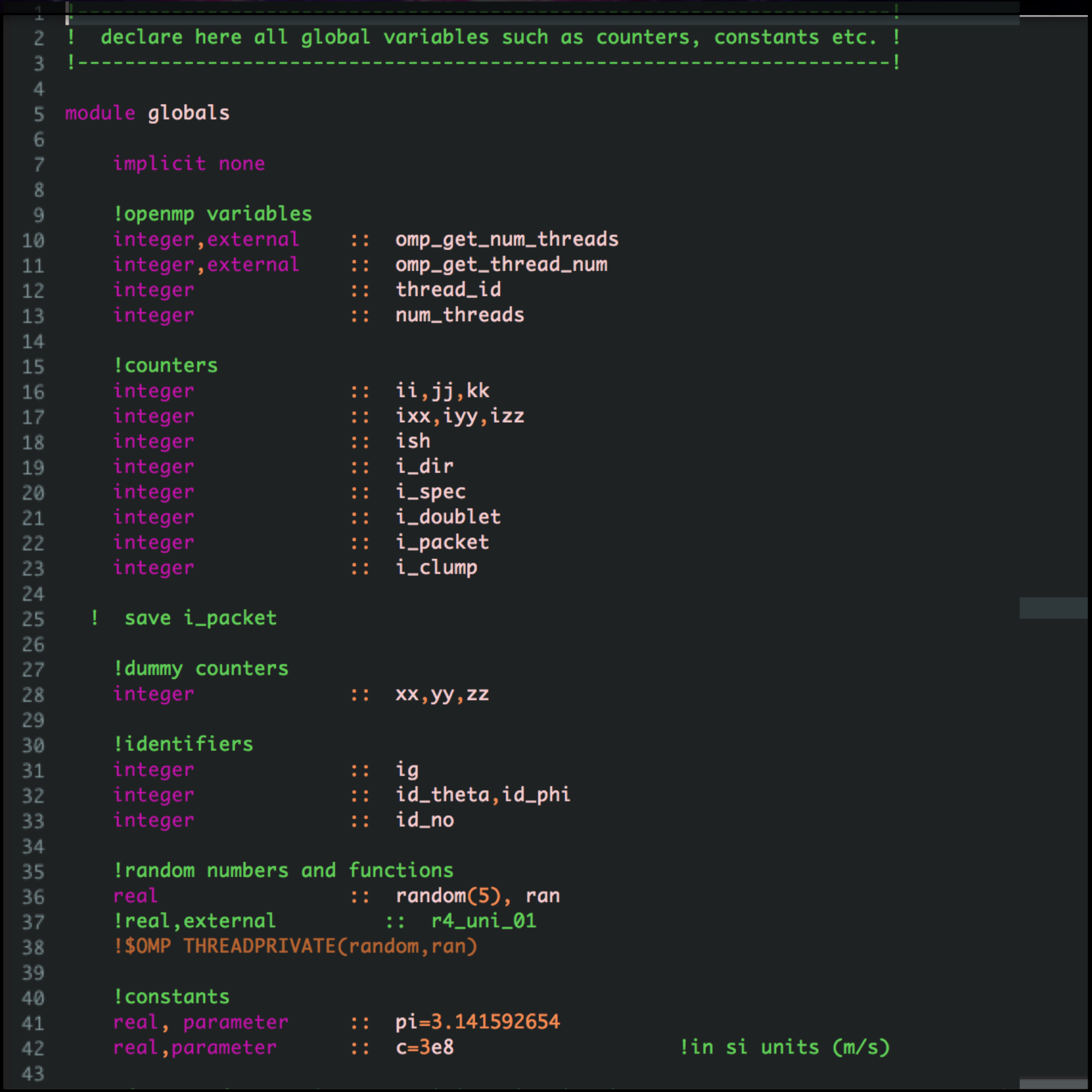


```fortran
module globals

!loop variables

integer :: ii, jj, kk


!physical constants

real, parameter :: pi = 3.141592654
real, parameter :: c_light = 3.0e8


end module globals
```

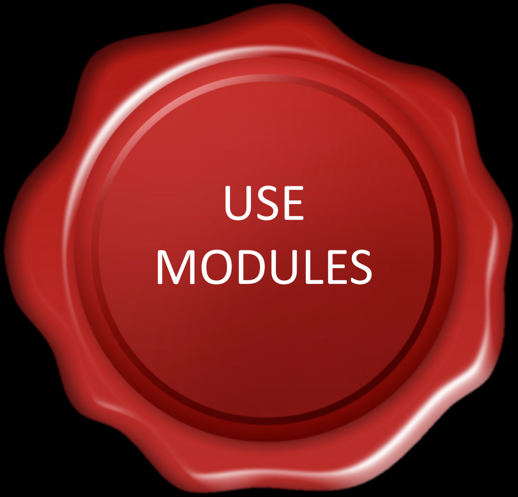


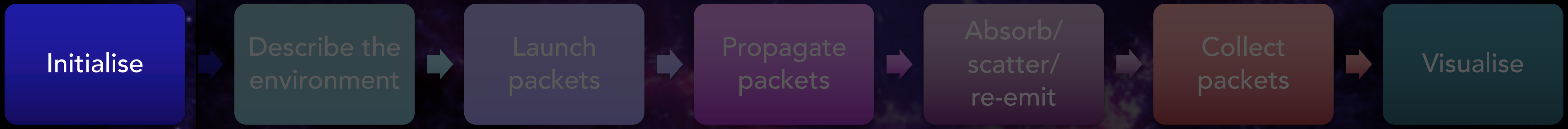


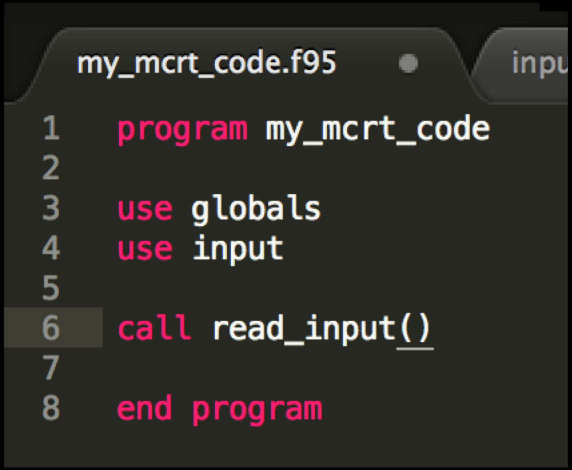
```fortran
program my_mcrt_code

use globals

end program
```



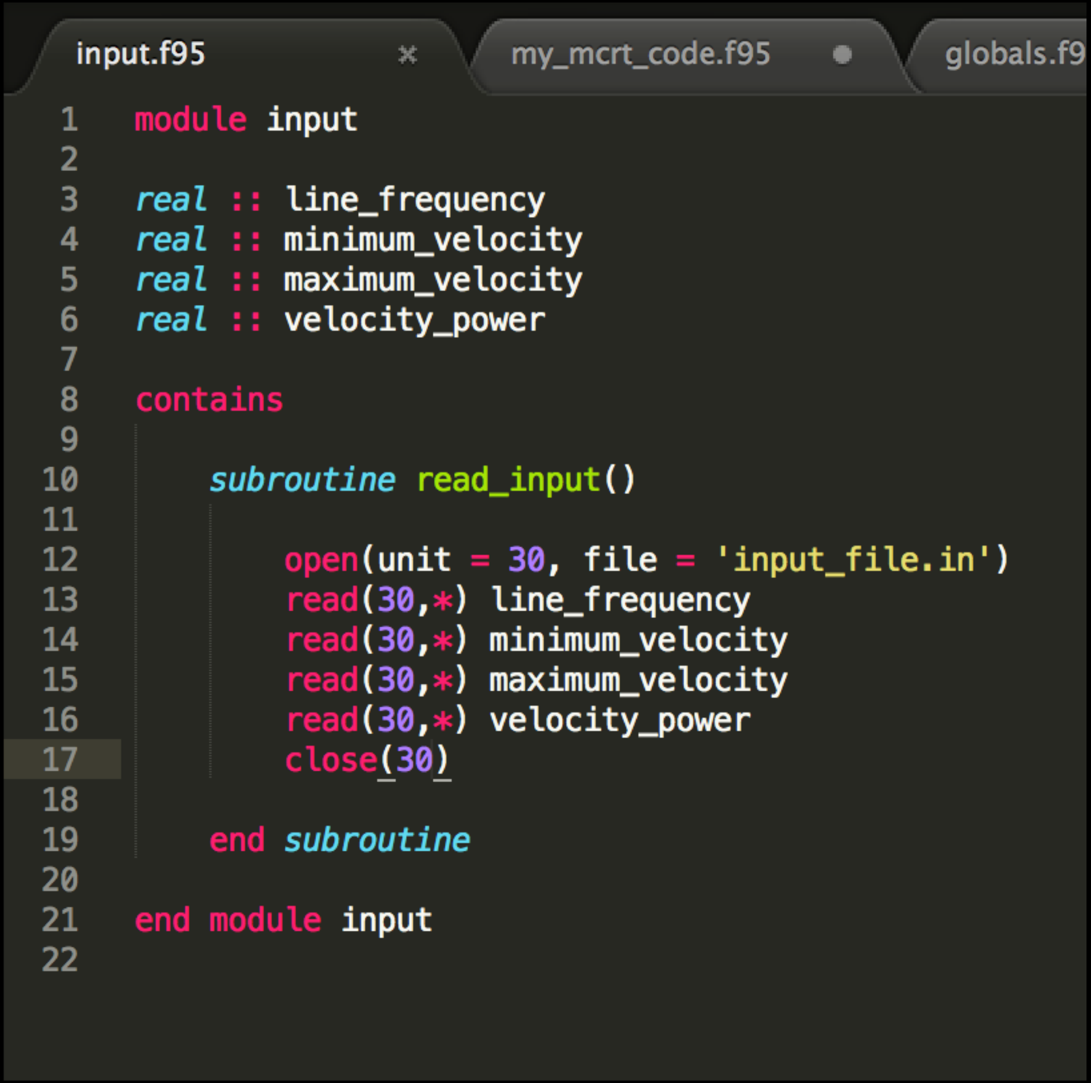
```fortran
!------------------------------------------------------------------------!
!  declare here all global variables such as counters, constants etc.  !
!------------------------------------------------------------------------!

module globals

    implicit none

    !openmp variables
    integer,external        ::  omp_get_num_threads
    integer,external        ::  omp_get_thread_num
    integer                 ::  thread_id
    integer                 ::  num_threads

    !counters
    integer                 ::  ii,jj,kk
    integer                 ::  ixx,iyy,izz
    integer                 ::  ish
    integer                 ::  i_dir
    integer                 ::  i_spec
    integer                 ::  i_doublet
    integer                 ::  i_packet
    integer                 ::  i_clump

  !  save i_packet

    !dummy counters
    integer                 ::  xx,yy,zz

    !identifiers
    integer                 ::  ig
    integer                 ::  id_theta,id_phi
    integer                 ::  id_no

    !random numbers and functions
    real                    ::  random(5), ran
    !real,external             ::  r4_uni_01
    !$OMP THREADPRIVATE(random,ran)

    !constants
    real, parameter         ::  pi=3.141592654
    real,parameter          ::  c=3e8                   !in si units (m/s)

```



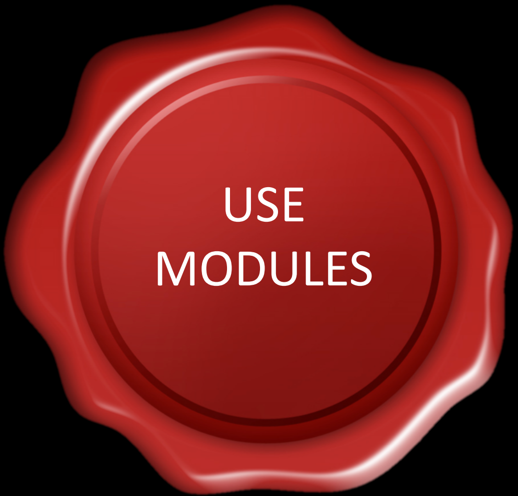




```fortran
program my_mcrt_code

use globals
use input

call read_input()

end program
```



```fortran
module input

real :: line_frequency
real :: minimum_velocity
real :: maximum_velocity
real :: velocity_power

contains

    subroutine read_input()

        open(unit = 30, file = 'input_file.in')
        read(30,*) line_frequency
        read(30,*) minimum_velocity
        read(30,*) maximum_velocity
        read(30,*) velocity_power
        close(30)

    end subroutine

end module input
```

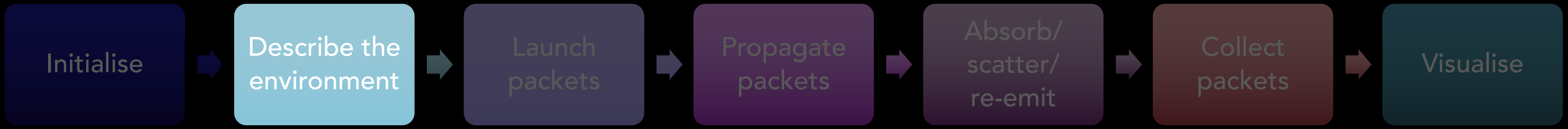


RECAP

MODULAR STRUCTURE

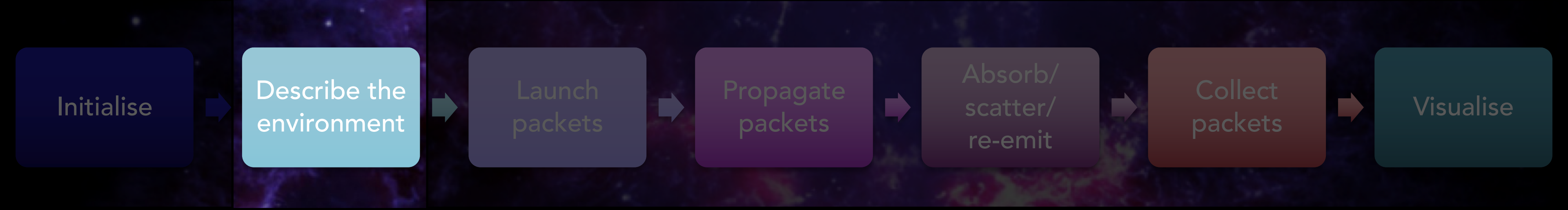
- We're building up the program block by block
- 'Program file' – runs the main body of the code calling functions and subroutines
- Break up your code into sections and save them in different files
- FORTRAN – if module A 'uses' module B, then all variables, subroutines and functions declared in module B can be seen and updated by module A
- I put all my variables, subroutines and functions in modules grouped by categories e.g. 'dust'

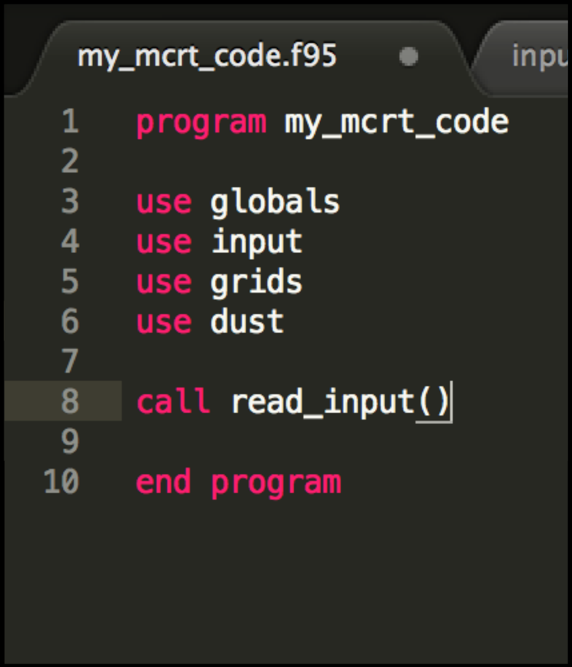
DESCRIBE THE ENVIRONMENT



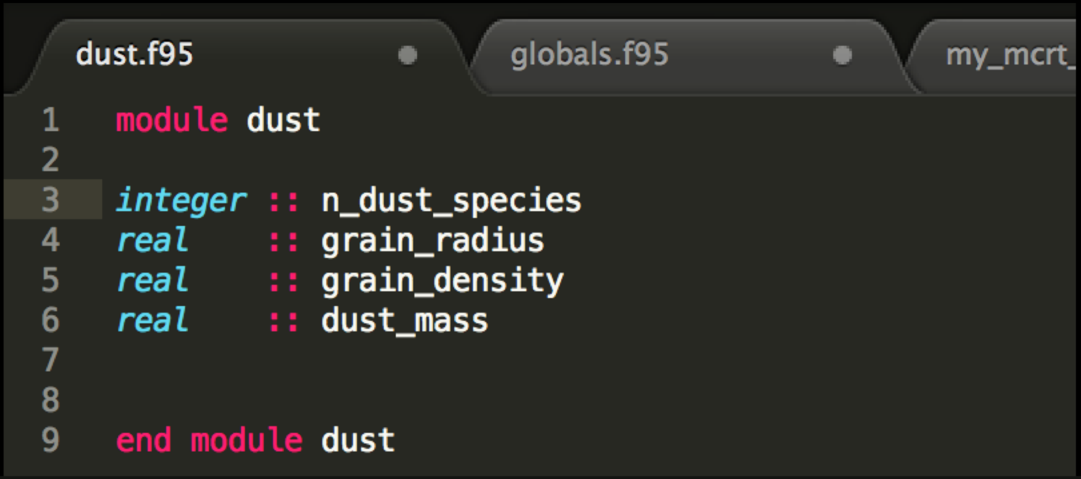
QUESTIONS TO ASK

- What is it made of?
  - Dust? Gas? Skin? Rocks?
  - Can I describe how light interacts with it?
- Where is it?
  - Density distribution – Smooth? Clumpy? Layered?
  - Physical extent – how big is it [and how do I want to describe that]?
- How should I describe it?
  - Analytic? Grid?

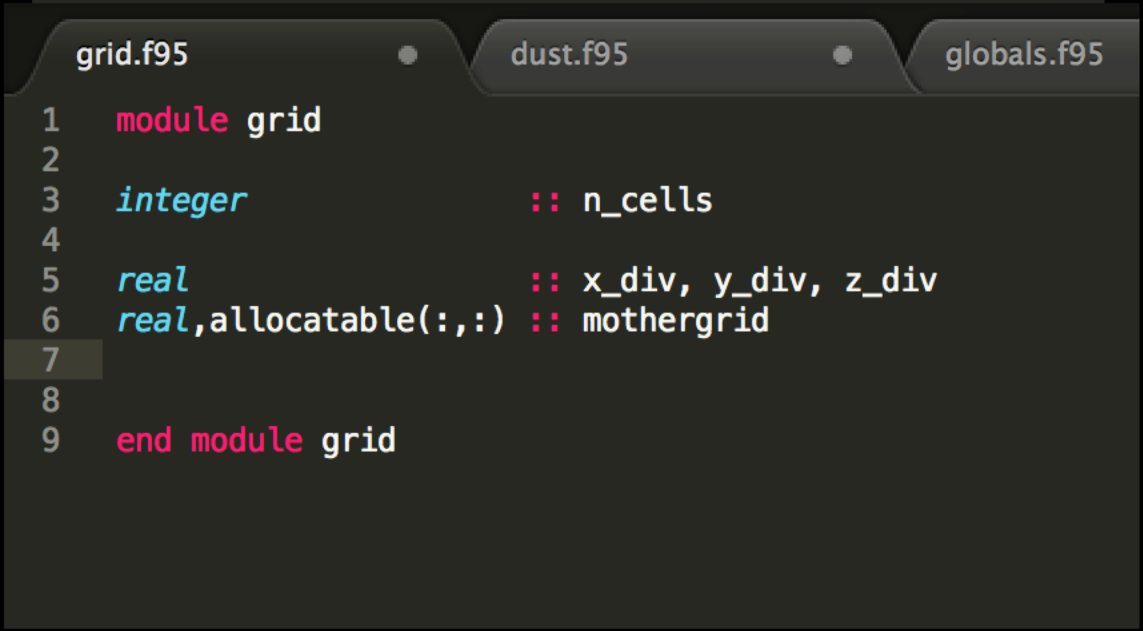




```fortran
program my_mcrt_code

use globals
use input
use grids
use dust

call read_input()

end program
```

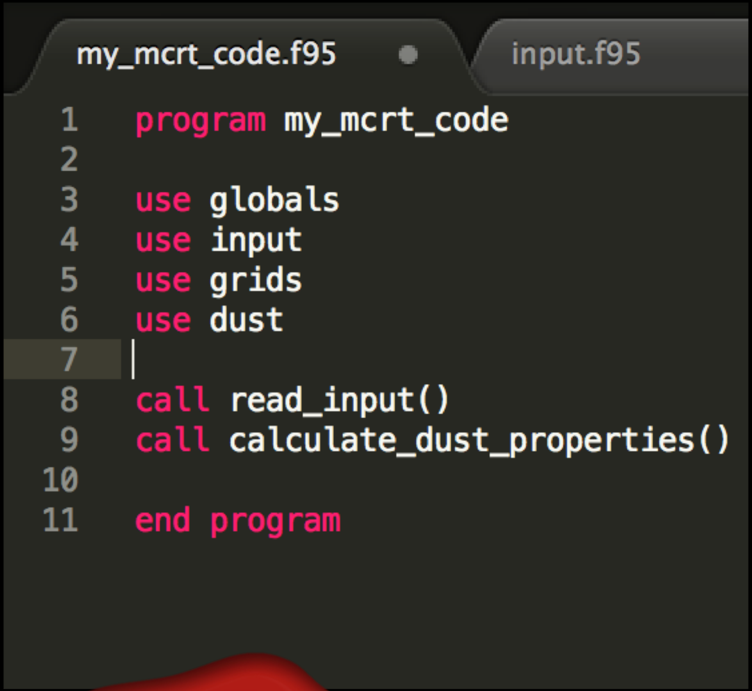


```fortran
module dust

integer :: n_dust_species
real    :: grain_radius
real    :: grain_density
real    :: dust_mass


end module dust
```

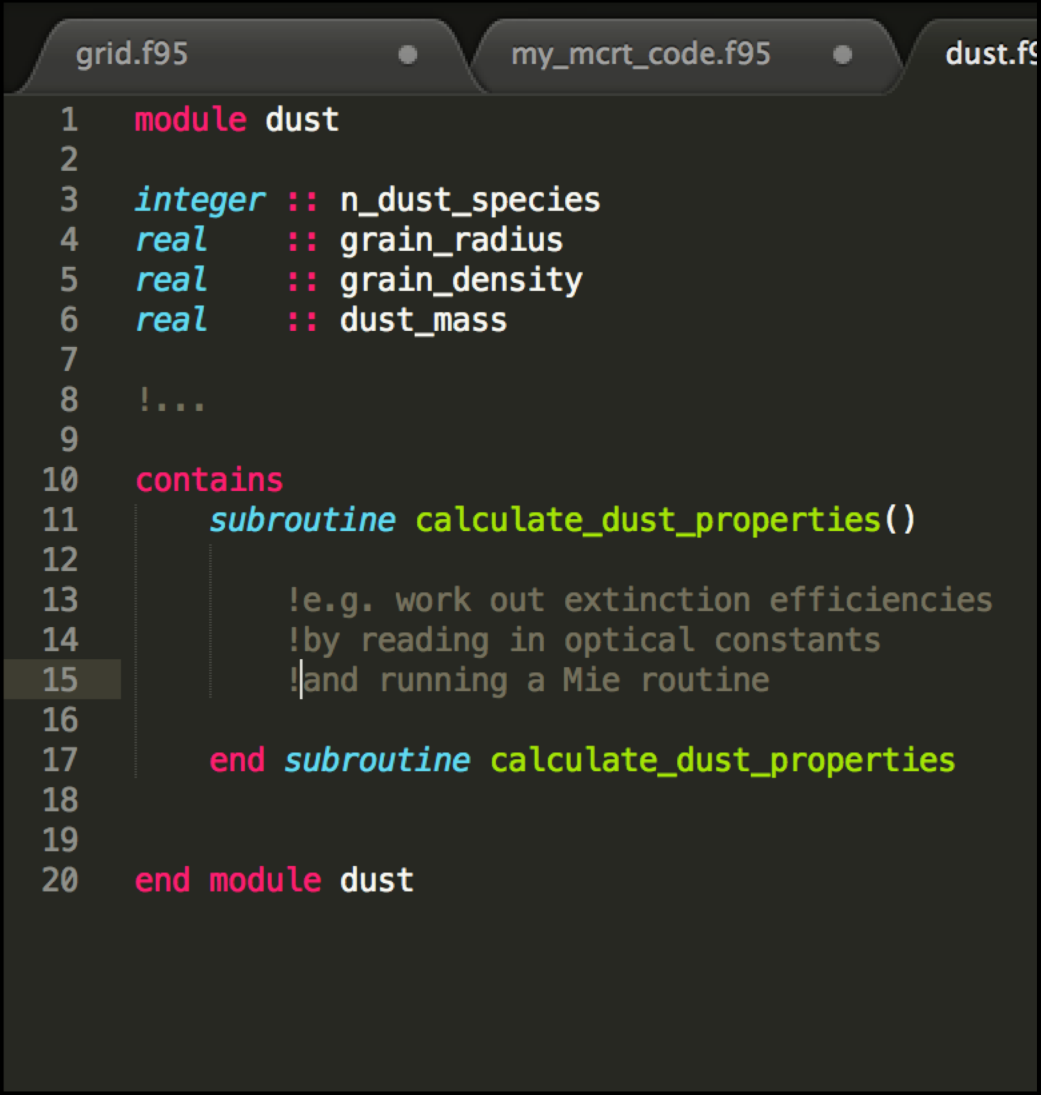


```fortran
module grid

integer                   :: n_cells

real                      :: x_div, y_div, z_div
real,allocatable(:,:)     :: mothergrid


end module grid
```

Initialise → **Describe the environment** → Launch packets → Propagate packets → Absorb/ scatter/ re-emit → Collect packets → Visualise

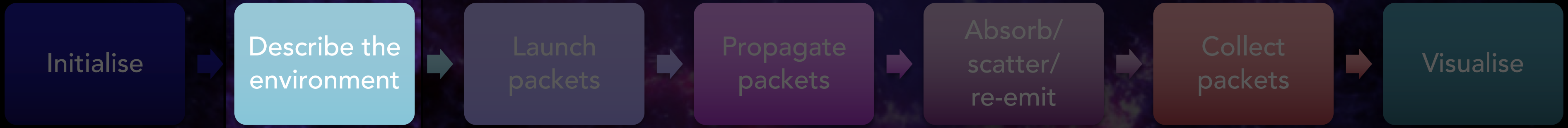


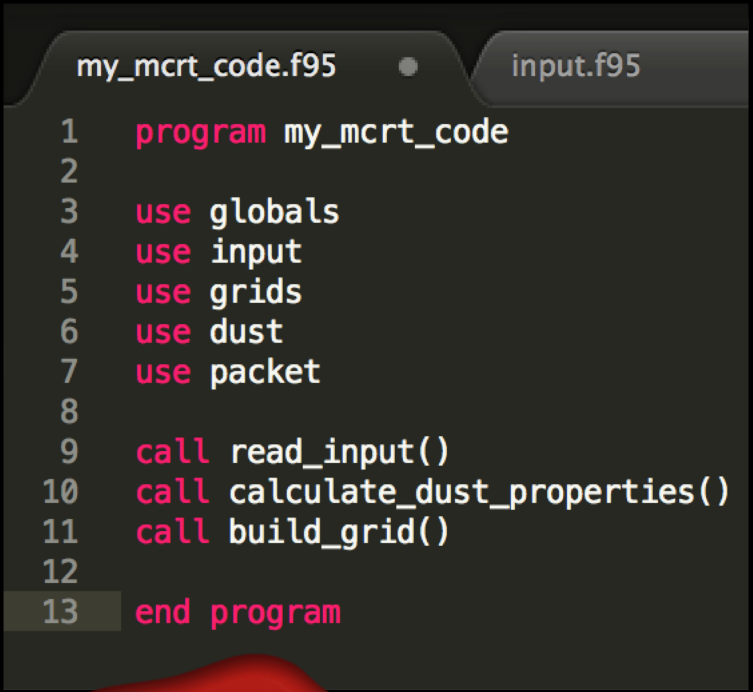
```fortran
program my_mcrt_code

use globals
use input
use grids
use dust

call read_input()
call calculate_dust_properties()

end program
```



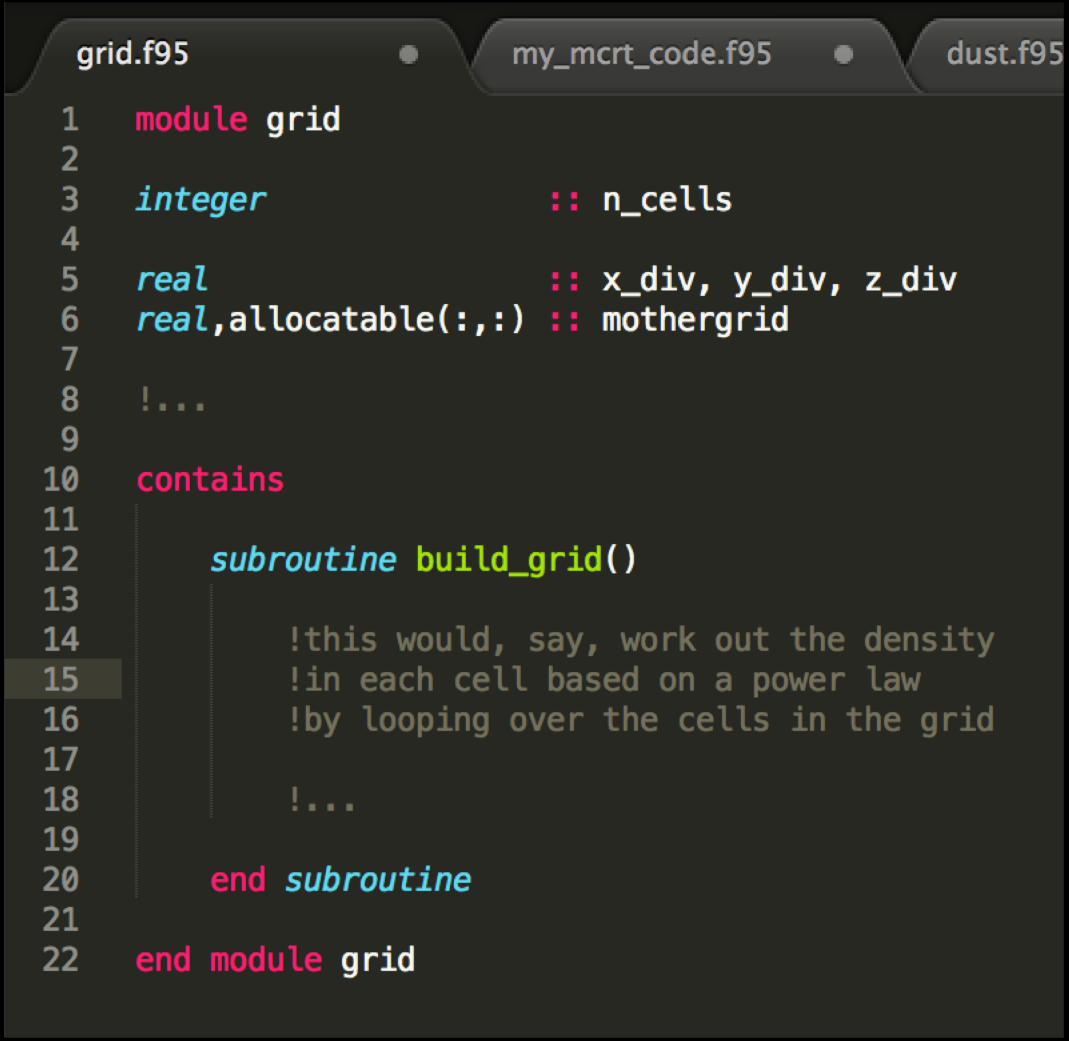
```fortran
module dust

integer :: n_dust_species
real    :: grain_radius
real    :: grain_density
real    :: dust_mass

!...

contains
    subroutine calculate_dust_properties()

        !e.g. work out extinction efficiencies
        !by reading in optical constants
        !and running a Mie routine

    end subroutine calculate_dust_properties


end module dust
```

USE LIBRARIES AND PACKAGES



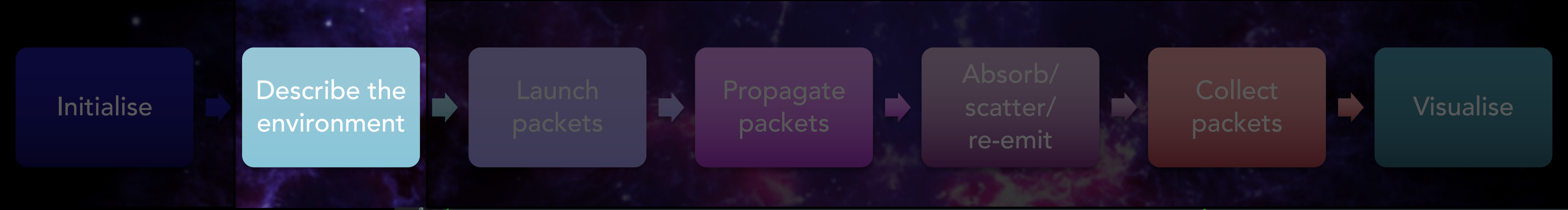


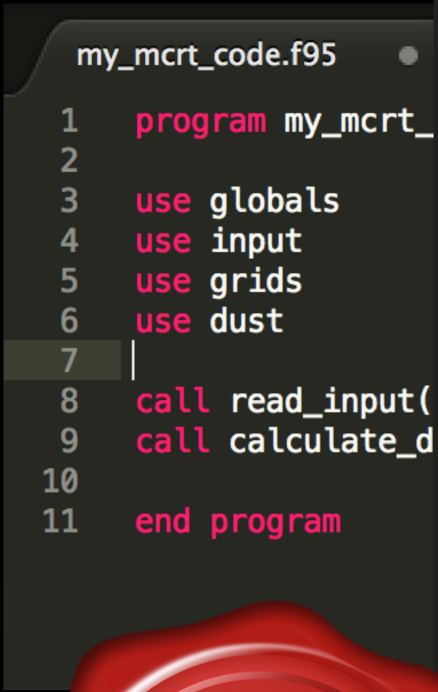
```fortran
program my_mcrt_code

use globals
use input
use grids
use dust
use packet

call read_input()
call calculate_dust_properties()
call build_grid()

end program
```



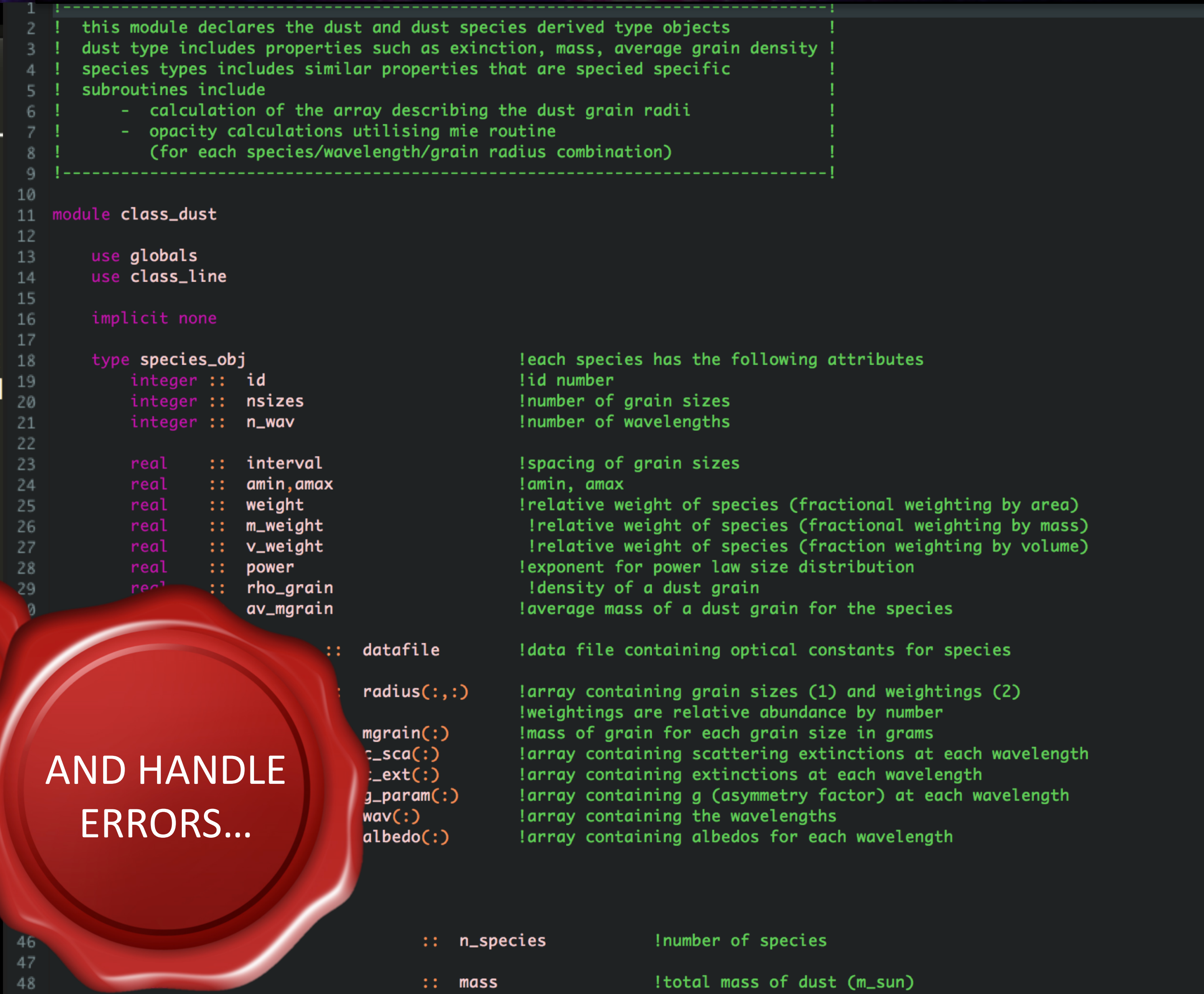
```fortran
module grid

integer                      :: n_cells

real                         :: x_div, y_div, z_div
real,allocatable(:,:) :: mothergrid

!...

contains

    subroutine build_grid()

        !this would, say, work out the density
        !in each cell based on a power law
        !by looping over the cells in the grid

        !...

    end subroutine

end module grid
```

USE LIBRARIES AND PACKAGES

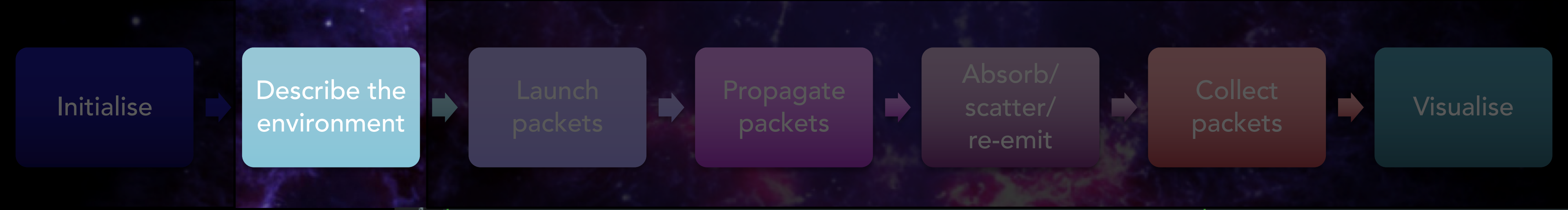


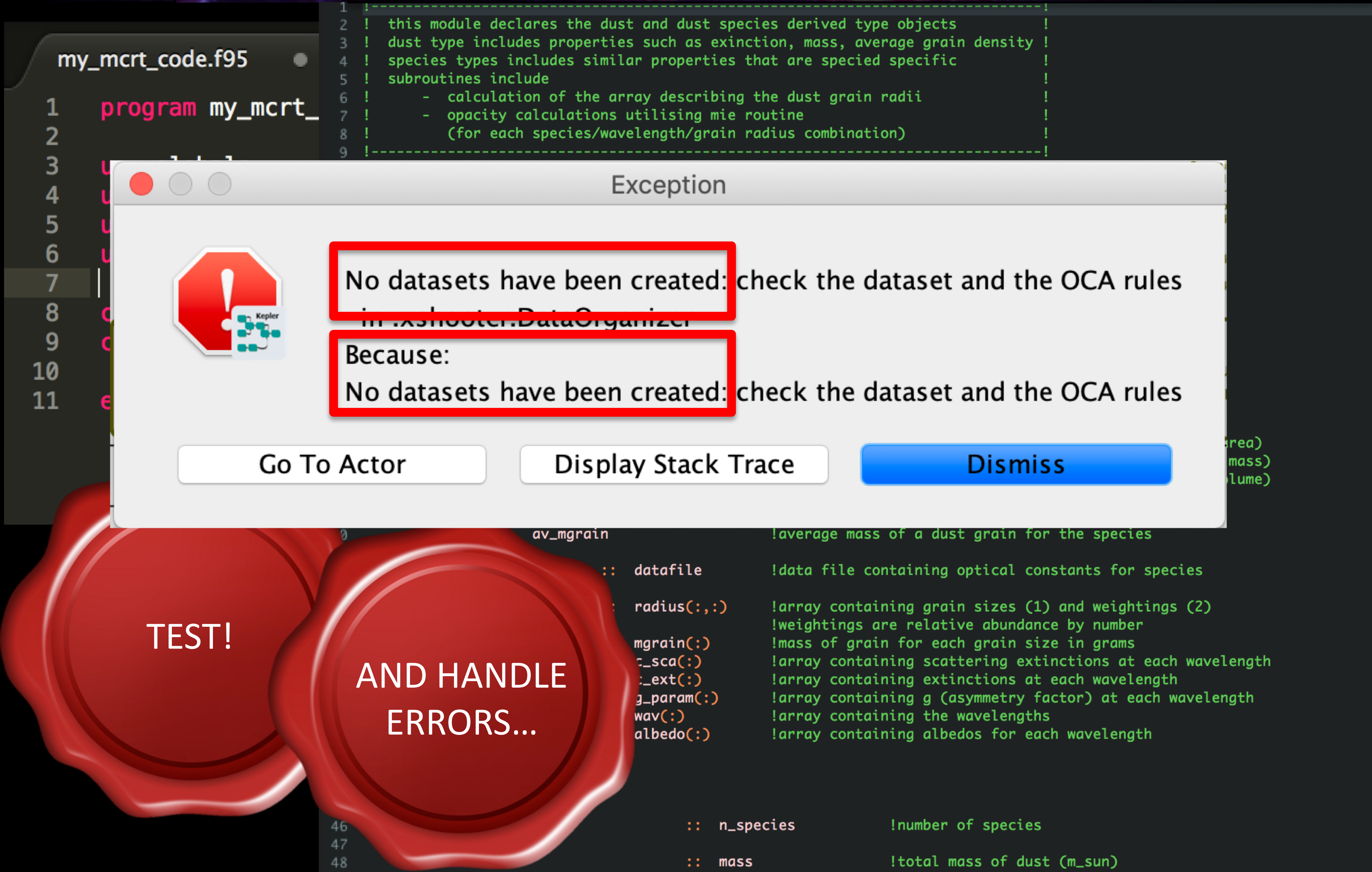


```fortran
program my_mcrt_

use globals
use input
use grids
use dust

call read_input(
call calculate_d

end program
```



```fortran
!--------------------------------------------------------------------------------!
!  this module declares the dust and dust species derived type objects           !
!  dust type includes properties such as exinction, mass, average grain density !
!  species types includes similar properties that are specied specific           !
!  subroutines include                                                           !
!       -  calculation of the array describing the dust grain radii              !
!       -  opacity calculations utilising mie routine                            !
!          (for each species/wavelength/grain radius combination)                !
!--------------------------------------------------------------------------------!

module class_dust

    use globals
    use class_line

    implicit none

    type species_obj                                  !each species has the following attributes
        integer ::  id                                !id number
        integer ::  nsizes                            !number of grain sizes
        integer ::  n_wav                             !number of wavelengths

        real    ::  interval                          !spacing of grain sizes
        real    ::  amin,amax                         !amin, amax
        real    ::  weight                            !relative weight of species (fractional weighting by area)
        real    ::  m_weight                           !relative weight of species (fractional weighting by mass)
        real    ::  v_weight                           !relative weight of species (fraction weighting by volume)
        real    ::  power                             !exponent for power law size distribution
        real    ::  rho_grain                          !density of a dust grain
                    av_mgrain                         !average mass of a dust grain for the species

                        ::  datafile                  !data file containing optical constants for species

                            radius(:,:)               !array containing grain sizes (1) and weightings (2)
                                                      !weightings are relative abundance by number
                            mgrain(:)                 !mass of grain for each grain size in grams
                            c_sca(:)                  !array containing scattering extinctions at each wavelength
                            _ext(:)                   !array containing extinctions at each wavelength
                            g_param(:)                !array containing g (asymmetry factor) at each wavelength
                            wav(:)                    !array containing the wavelengths
                            albedo(:)                 !array containing albedos for each wavelength


                              ::  n_species             !number of species

                              ::  mass                  !total mass of dust (m_sun)
```





my_mcrt_code.f95

```
program my_mcrt_
```

```
!------------------------------------------------------------------------------!
!  this module declares the dust and dust species derived type objects         !
!  dust type includes properties such as exinction, mass, average grain density !
!  species types includes similar properties that are specied specific         !
!  subroutines include                                                         !
!      -  calculation of the array describing the dust grain radii             !
!      -  opacity calculations utilising mie routine                           !
!         (for each species/wavelength/grain radius combination)               !
!------------------------------------------------------------------------------!
```

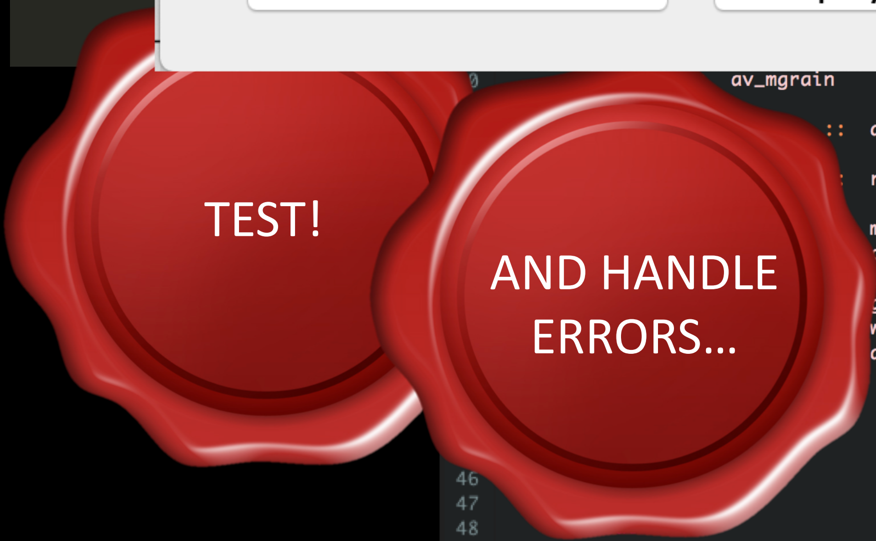
```
                                                                              rea)
                                                                              mass)
                                                                              lume)
         av_mgrain                  !average mass of a dust grain for the species

                 ::  datafile       !data file containing optical constants for species

                  :  radius(:,:)    !array containing grain sizes (1) and weightings (2)
                                    !weightings are relative abundance by number
                     mgrain(:)      !mass of grain for each grain size in grams
                     c_sca(:)       !array containing scattering extinctions at each wavelength
                     c_ext(:)       !array containing extinctions at each wavelength
                     g_param(:)     !array containing g (asymmetry factor) at each wavelength
                     wav(:)         !array containing the wavelengths
                     albedo(:)      !array containing albedos for each wavelength


                         ::  n_species           !number of species

                         ::  mass                !total mass of dust (m_sun)
```

Exception

No datasets have been created: check the dataset and the OCA rules in .xshooter.DataOrganizer

Because:
No datasets have been created: check the dataset and the OCA rules

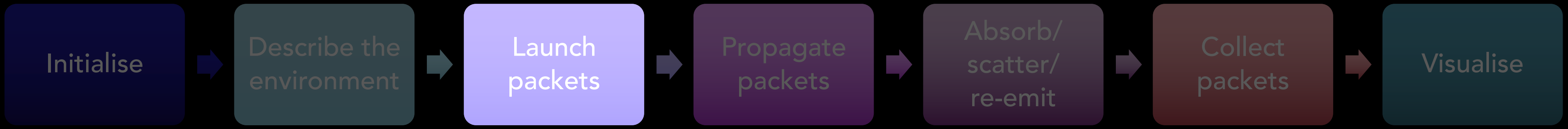
Go To Actor | Display Stack Trace | Dismiss



RECAP

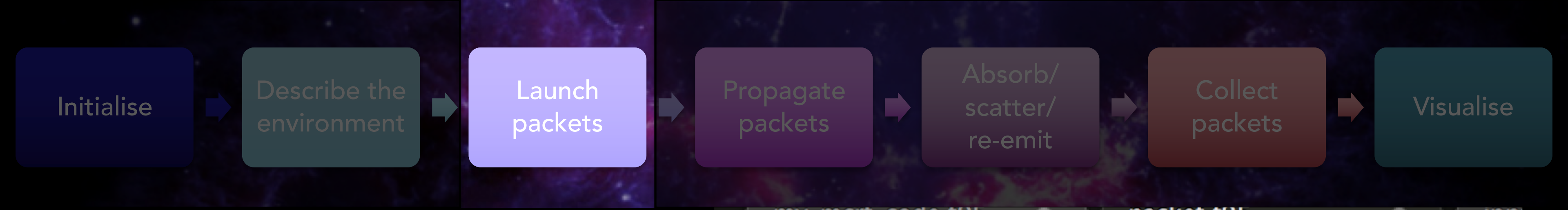
- Build up each section of code as you go
- Test each section as you go
  - Use benchmark tests and analytical results
  - Use sense checks and ‘count checks’
  - Consider writing unit tests
- Calculate the properties of your medium in advance and store
- Grids allow for flexibility – properties in a given grid cell are constant

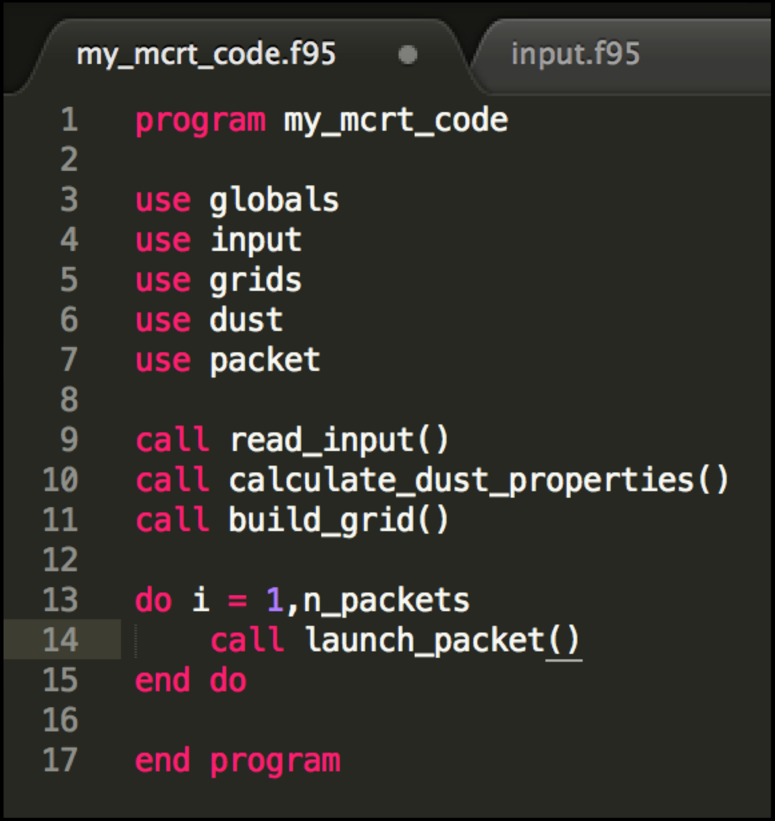
LAUNCH PACKETS



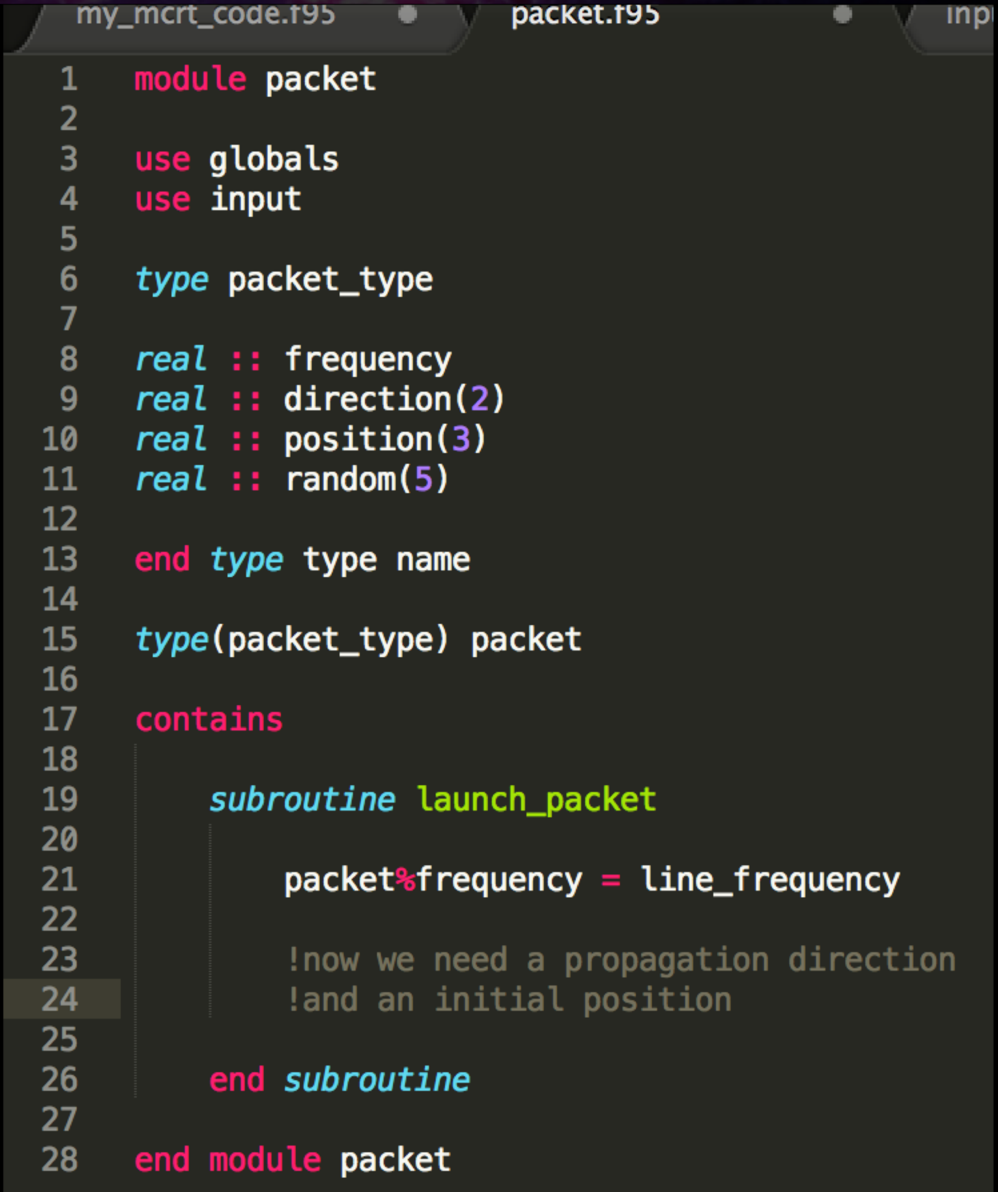
QUESTIONS TO ASK

- Frequency distribution *P(ν) α* ?
  - Blackbody? Monochromatic? Continuum?
- Spatial emissivity distribution *i(x,y,z) α* ?
  - Proportional to density? Radial distribution? Point source? Arbitrary distribution?
- Propagation direction
  - Isotropic? Non-isotropic? Plane parallel?

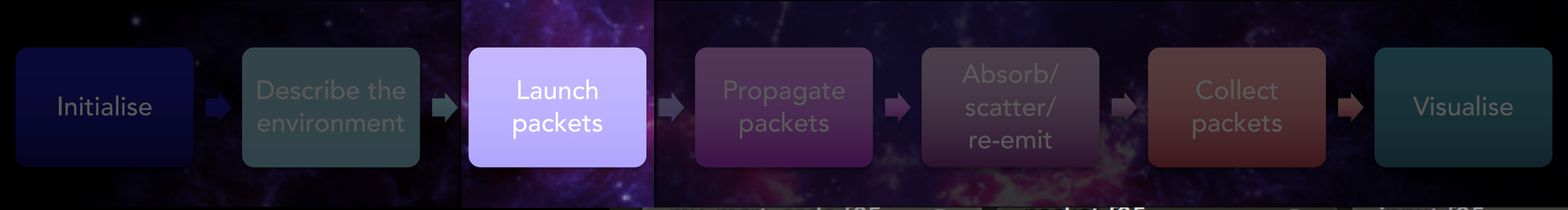




```
program my_mcrt_code

use globals
use input
use grids
use dust
use packet

call read_input()
call calculate_dust_properties()
call build_grid()

do i = 1,n_packets
    call launch_packet()
end do

end program
```



```
module packet

use globals
use input

type packet_type

real :: frequency
real :: direction(2)
real :: position(3)
real :: random(5)

end type type name

type(packet_type) packet

contains

    subroutine launch_packet

        packet%frequency = line_frequency

        !now we need a propagation direction
        !and an initial position

    end subroutine

end module packet
```



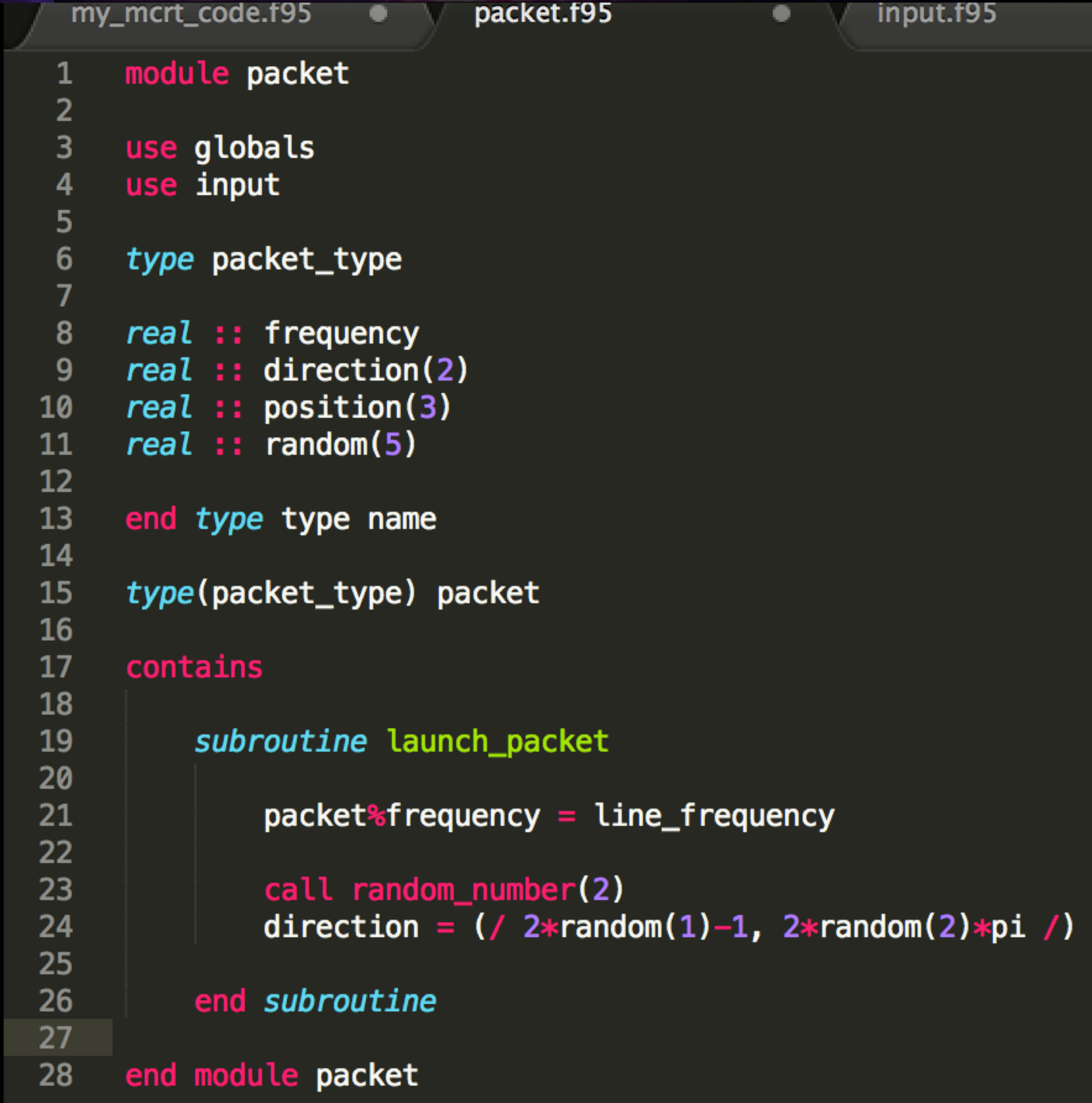
Propagation direction:

e.g.

sample from an isotropic distribution

$$\cos\theta = 2\zeta - 1$$

$$\phi = 2\pi\xi$$



```
module packet

use globals
use input

type packet_type

real :: frequency
real :: direction(2)
real :: position(3)
real :: random(5)

end type type name

type(packet_type) packet

contains

    subroutine launch_packet

        packet%frequency = line_frequency

        call random_number(2)
        direction = (/ 2*random(1)-1, 2*random(2)*pi /)

    end subroutine

end module packet
```

Position:

e.g. radial power law in 1D

$$\text{For } P(r) = Cr^{n} \ \text{ where } r \in [r_{\min}, r_{\max}]$$

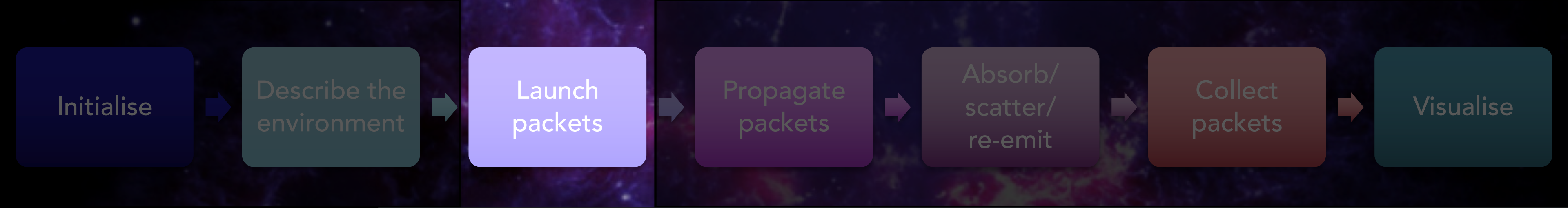
$$r = [(r_{\max}^{n+1} - r_{\min}^{n+1})\gamma + r_{\min}^{n+1}]^{\frac{1}{n+1}}$$

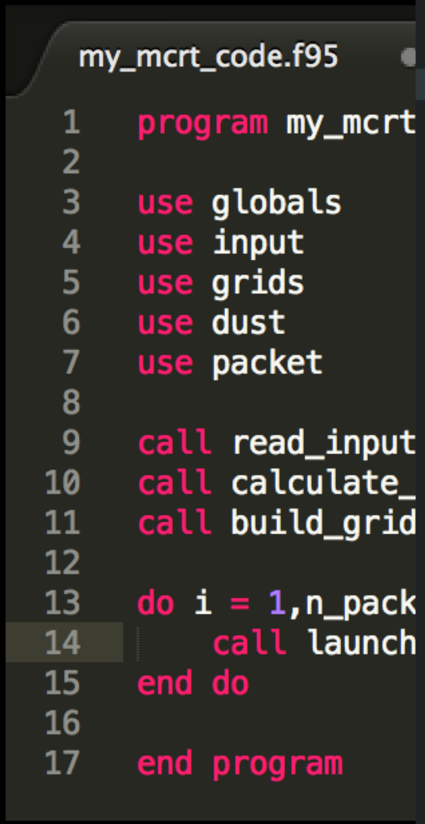
CARE WITH RANDOM NUMBERS

Wolfram MathWorld™ the we

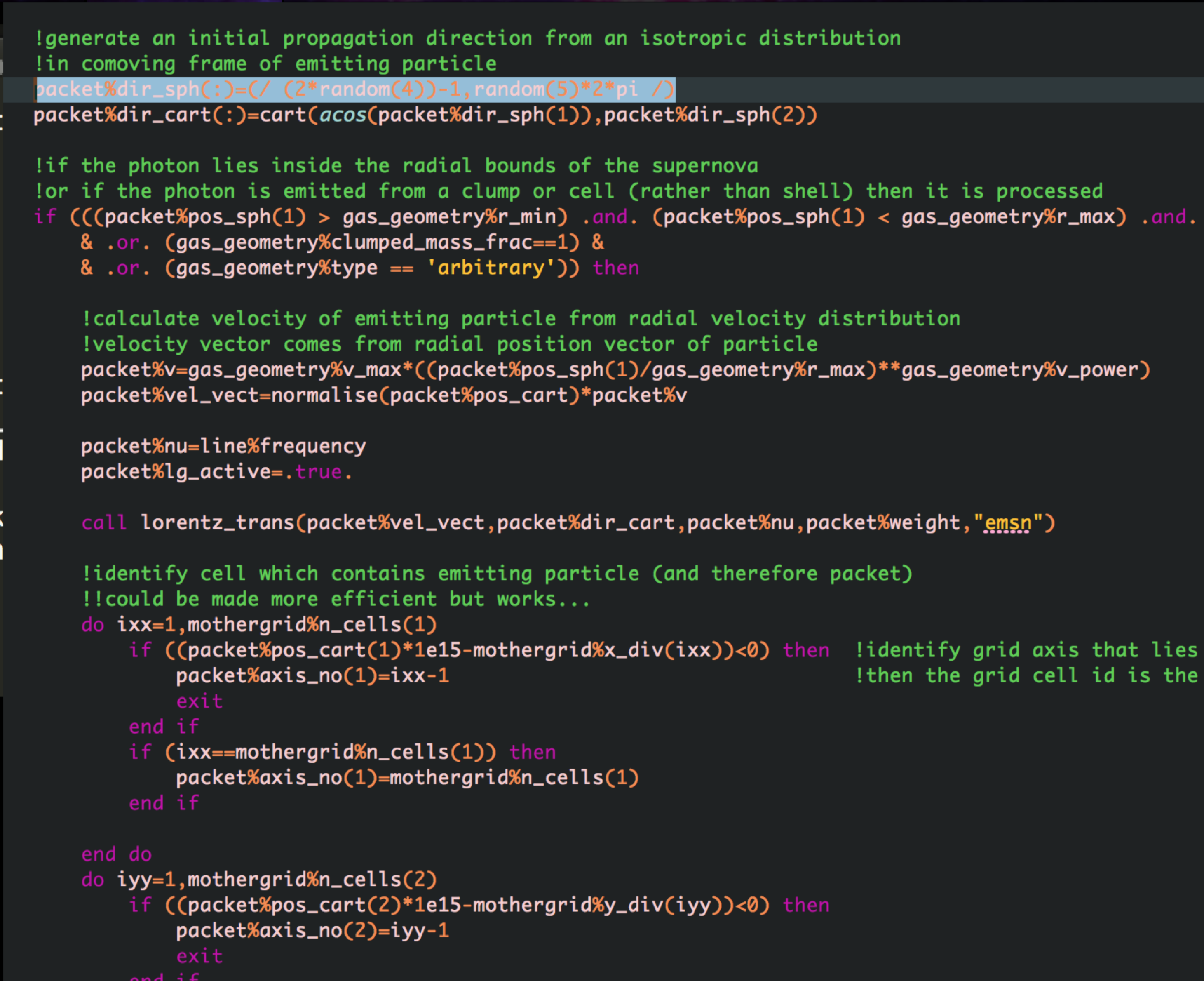
Built with Mathematica Technology

Uniform random numbers in [0,1) can be converted to other distributions



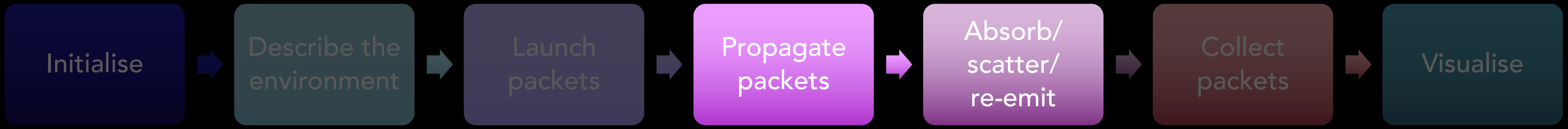

my_mcrt_code.f95

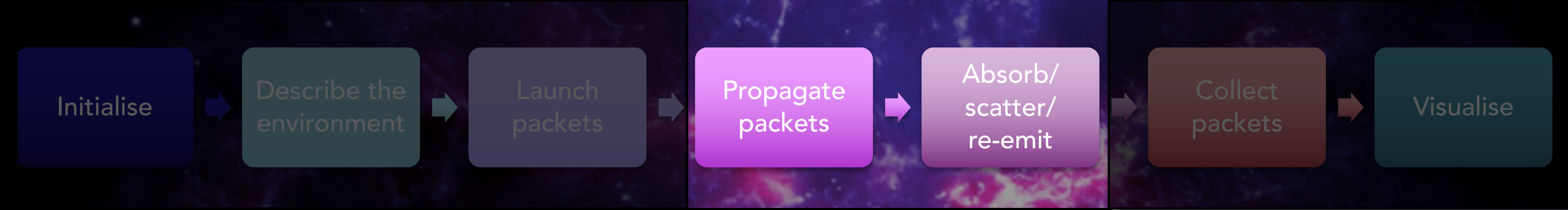
```fortran
program my_mcrt

use globals
use input
use grids
use dust
use packet

call read_input
call calculate_
call build_grid

do i = 1,n_pack
      call launch
end do

end program
```



```fortran
!generate an initial propagation direction from an isotropic distribution
!in comoving frame of emitting particle
packet%dir_sph(:)=(/ (2*random(4))-1,random(5)*2*pi /)
packet%dir_cart(:)=cart(acos(packet%dir_sph(1)),packet%dir_sph(2))

!if the photon lies inside the radial bounds of the supernova
!or if the photon is emitted from a clump or cell (rather than shell) then it is processed
if (((packet%pos_sph(1) > gas_geometry%r_min) .and. (packet%pos_sph(1) < gas_geometry%r_max) .and.
    & .or. (gas_geometry%clumped_mass_frac==1) &
    & .or. (gas_geometry%type == 'arbitrary')) then

    !calculate velocity of emitting particle from radial velocity distribution
    !velocity vector comes from radial position vector of particle
    packet%v=gas_geometry%v_max*((packet%pos_sph(1)/gas_geometry%r_max)**gas_geometry%v_power)
    packet%vel_vect=normalise(packet%pos_cart)*packet%v

    packet%nu=line%frequency
    packet%lg_active=.true.

    call lorentz_trans(packet%vel_vect,packet%dir_cart,packet%nu,packet%weight,"emsn")

    !identify cell which contains emitting particle (and therefore packet)
    !!could be made more efficient but works...
    do ixx=1,mothergrid%n_cells(1)
        if ((packet%pos_cart(1)*1e15-mothergrid%x_div(ixx))<0) then  !identify grid axis that lies
            packet%axis_no(1)=ixx-1                                  !then the grid cell id is the
            exit
        end if
        if (ixx==mothergrid%n_cells(1)) then
            packet%axis_no(1)=mothergrid%n_cells(1)
        end if

    end do
    do iyy=1,mothergrid%n_cells(2)
        if ((packet%pos_cart(2)*1e15-mothergrid%y_div(iyy))<0) then
            packet%axis_no(2)=iyy-1
            exit
        end if
```

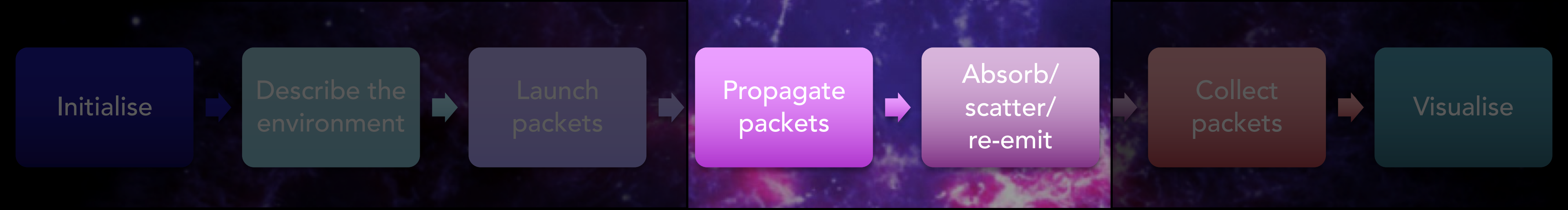
PROPAGATE PACKETS & ABSORPTION/SCATTERING/RE-EMISSION

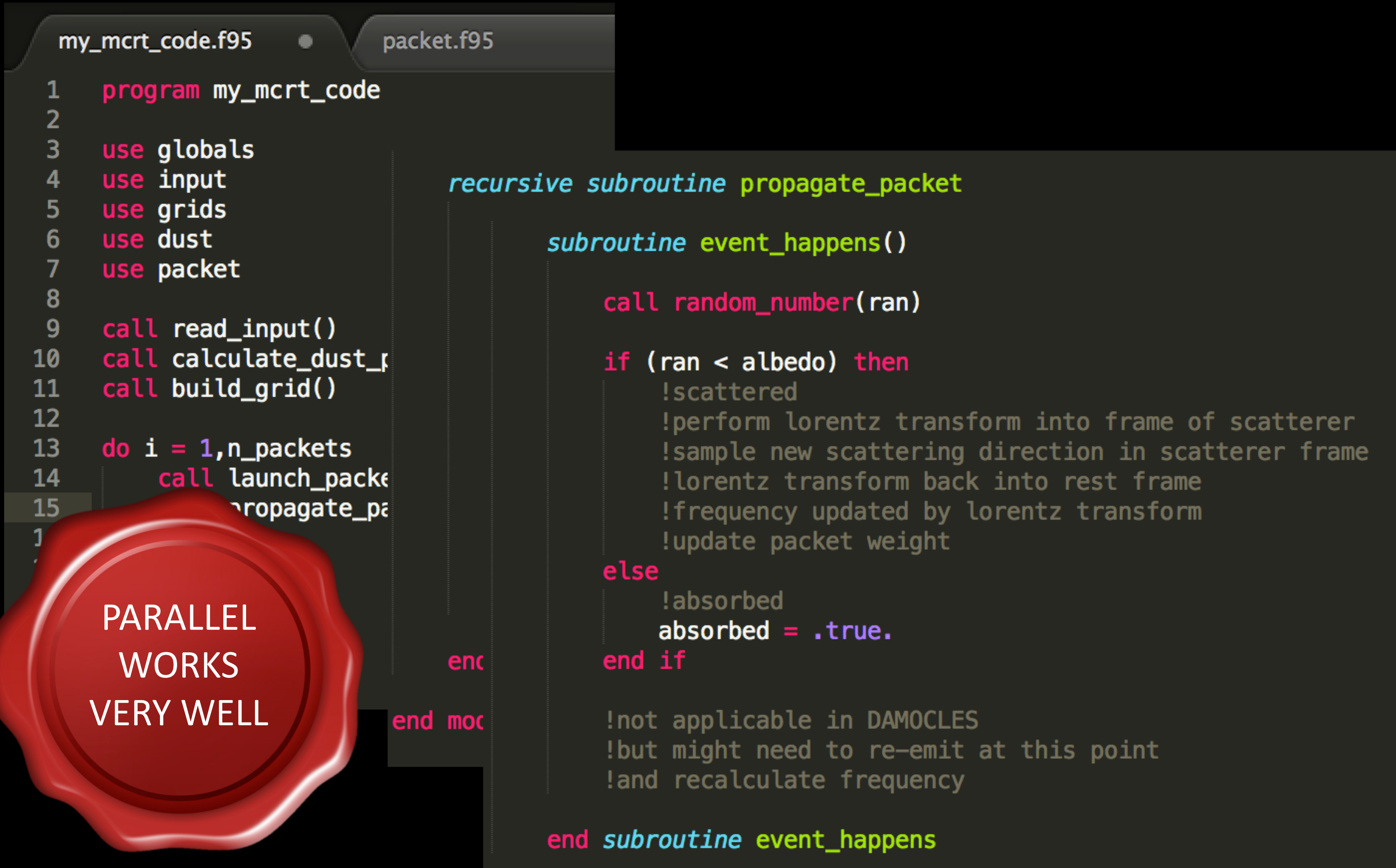




QUESTIONS TO ASK

- What happens to absorbed packets?
- Do I need to use weighted packets?
- With what direction are scattered packets re-emitted?
  - Isotropic? Phase function?
- Are packets re-emitted immediately?
- Do I need to iterate?
  - Update grid properties? Determine thermal balance?



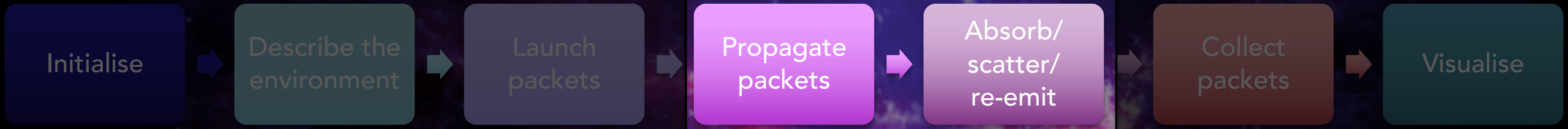


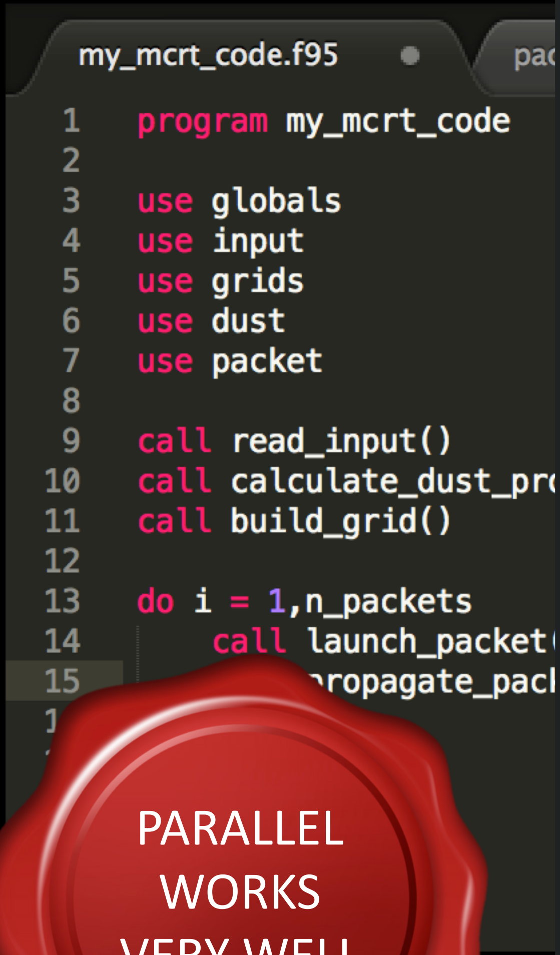
```fortran
program my_mcrt_code

use globals
use input
use grids
use dust
use packet

call read_input()
call calculate_dust_
call build_grid()

do i = 1,n_packets
    call launch_packe
         ropagate_pa
```

```fortran
enc

end moc
```

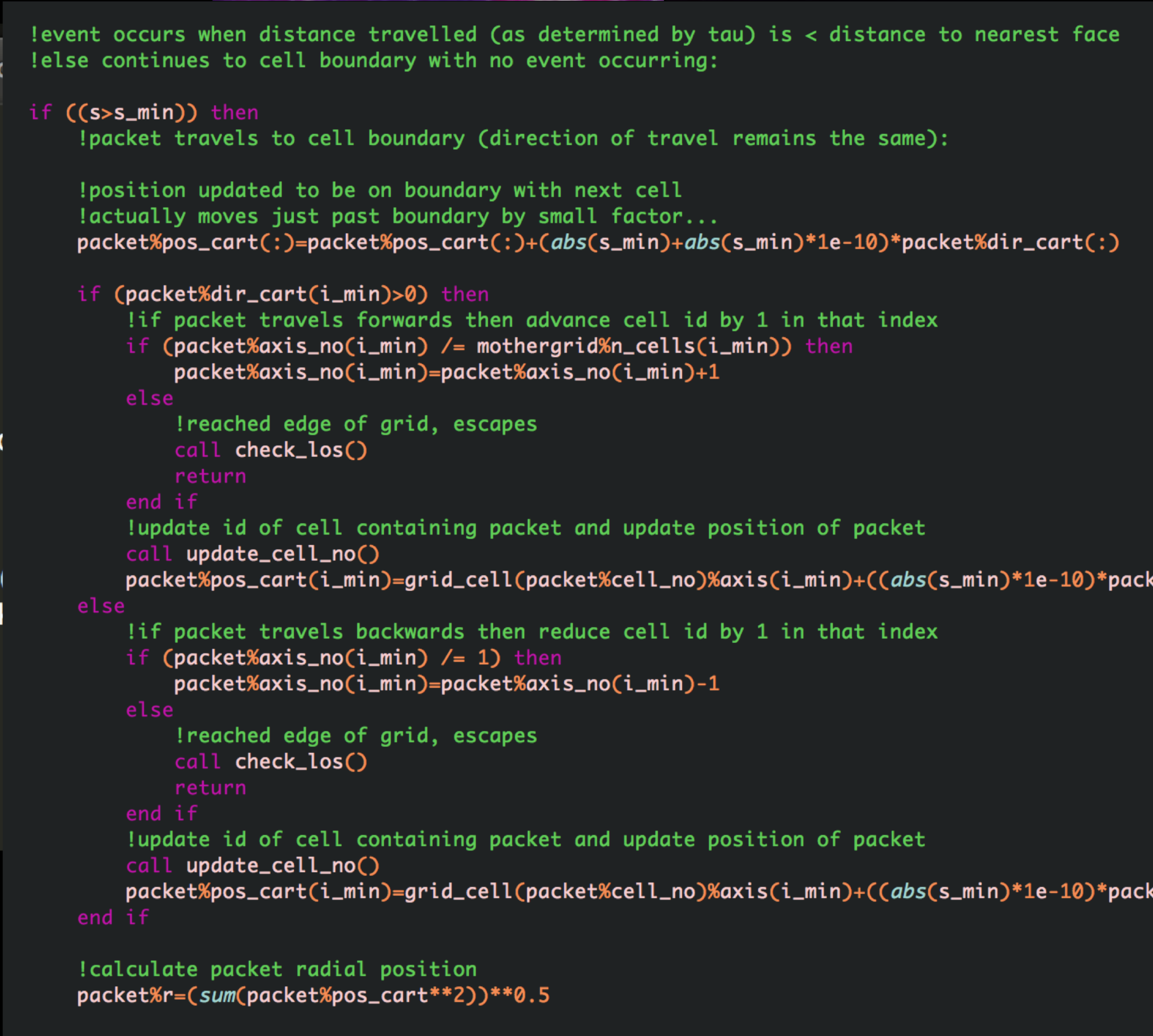
```fortran
recursive subroutine propagate_packet

    subroutine event_happens()

        call random_number(ran)

        if (ran < albedo) then
            !scattered
            !perform lorentz transform into frame of scatterer
            !sample new scattering direction in scatterer frame
            !lorentz transform back into rest frame
            !frequency updated by lorentz transform
            !update packet weight
        else
            !absorbed
            absorbed = .true.
        end if

        !not applicable in DAMOCLES
        !but might need to re-emit at this point
        !and recalculate frequency

    end subroutine event_happens
```

PARALLEL WORKS VERY WELL




my_mcrt_code.f95

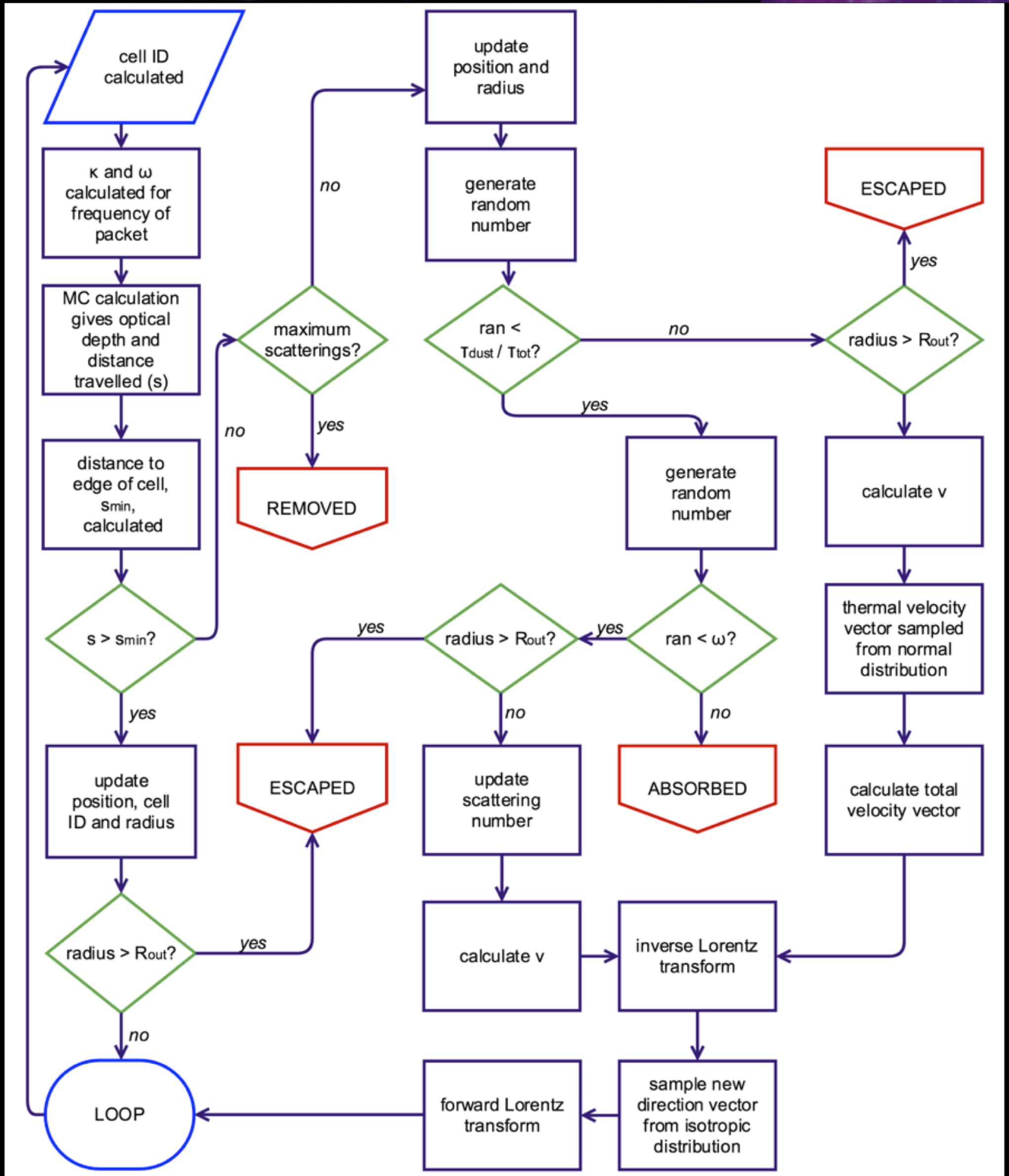
```
program my_mcrt_code

use globals
use input
use grids
use dust
use packet

call read_input()
call calculate_dust_pr
call build_grid()

do i = 1,n_packets
    call launch_packet(
       ropagate_pacl
```



```
!event occurs when distance travelled (as determined by tau) is < distance to nearest face
!else continues to cell boundary with no event occurring:

if ((s>s_min)) then
    !packet travels to cell boundary (direction of travel remains the same):

    !position updated to be on boundary with next cell
    !actually moves just past boundary by small factor...
    packet%pos_cart(:)=packet%pos_cart(:)+(abs(s_min)+abs(s_min)*1e-10)*packet%dir_cart(:)

    if (packet%dir_cart(i_min)>0) then
        !if packet travels forwards then advance cell id by 1 in that index
        if (packet%axis_no(i_min) /= mothergrid%n_cells(i_min)) then
            packet%axis_no(i_min)=packet%axis_no(i_min)+1
        else
            !reached edge of grid, escapes
            call check_los()
            return
        end if
        !update id of cell containing packet and update position of packet
        call update_cell_no()
        packet%pos_cart(i_min)=grid_cell(packet%cell_no)%axis(i_min)+((abs(s_min)*1e-10)*pack
    else
        !if packet travels backwards then reduce cell id by 1 in that index
        if (packet%axis_no(i_min) /= 1) then
            packet%axis_no(i_min)=packet%axis_no(i_min)-1
        else
            !reached edge of grid, escapes
            call check_los()
            return
        end if
        !update id of cell containing packet and update position of packet
        call update_cell_no()
        packet%pos_cart(i_min)=grid_cell(packet%cell_no)%axis(i_min)+((abs(s_min)*1e-10)*pack
    end if

    !calculate packet radial position
    packet%r=(sum(packet%pos_cart**2))**0.5
```

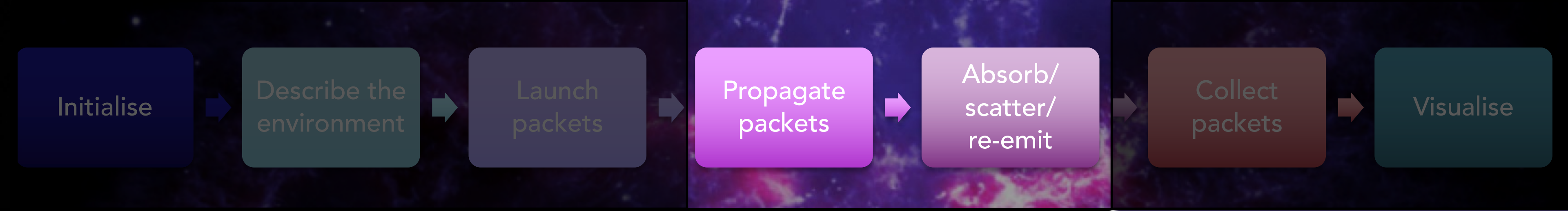
PARALLEL
WORKS
VERY WELL

Initialise → Describe the environment → Launch packets → Propagate packets → Absorb/ scatter/ re-emit → Collect packets → Visualise


cell ID calculated
κ and ω calculated for frequency of packet
MC calculation gives optical depth and distance travelled (s)
distance to edge of cell, smin, calculated
s > smin?
yes
no
update position, cell ID and radius
radius > Rout?
yes
no
LOOP
maximum scatterings?
no
yes
REMOVED
ESCAPED
update position and radius
generate random number
ran < Tdust / Ttot?
no
yes
generate random number
ran < ω?
yes
no
ABSORBED
radius > Rout?
yes
no
update scattering number
calculate v
inverse Lorentz transform
sample new direction vector from isotropic distribution
forward Lorentz transform
ESCAPED
yes
radius > Rout?
calculate v
thermal velocity vector sampled from normal distribution
calculate total velocity vector

A packet's path through the ejecta in DAMOCLES

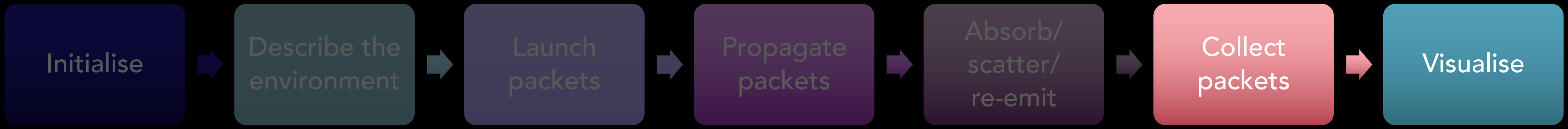
(note, no re-emission…)

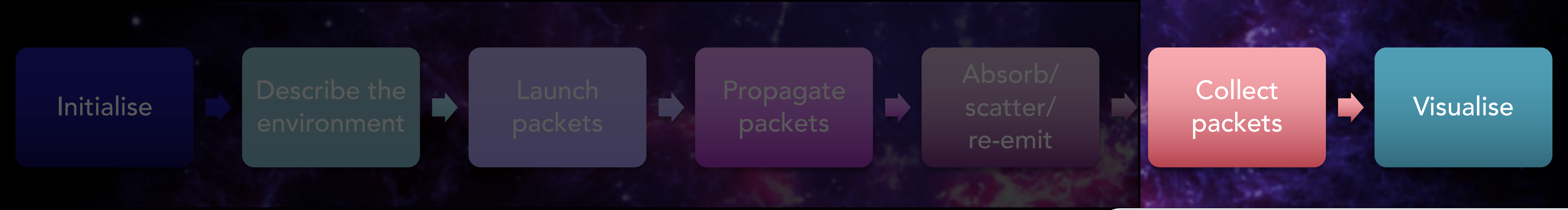


RECAP

- Packets do not interact so…
  - can run multiple at once
- Monte Carlo very, very parallel (and fairly easy to do so)
  - Lots of packets required (10^4 – 10^9 typical) means it can be slow so parallelisation helps
- Random numbers determine random walk but also events along the way

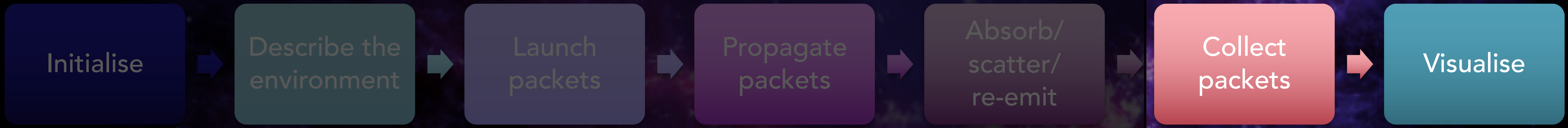
COLLECT PACKETS & VISUALISE

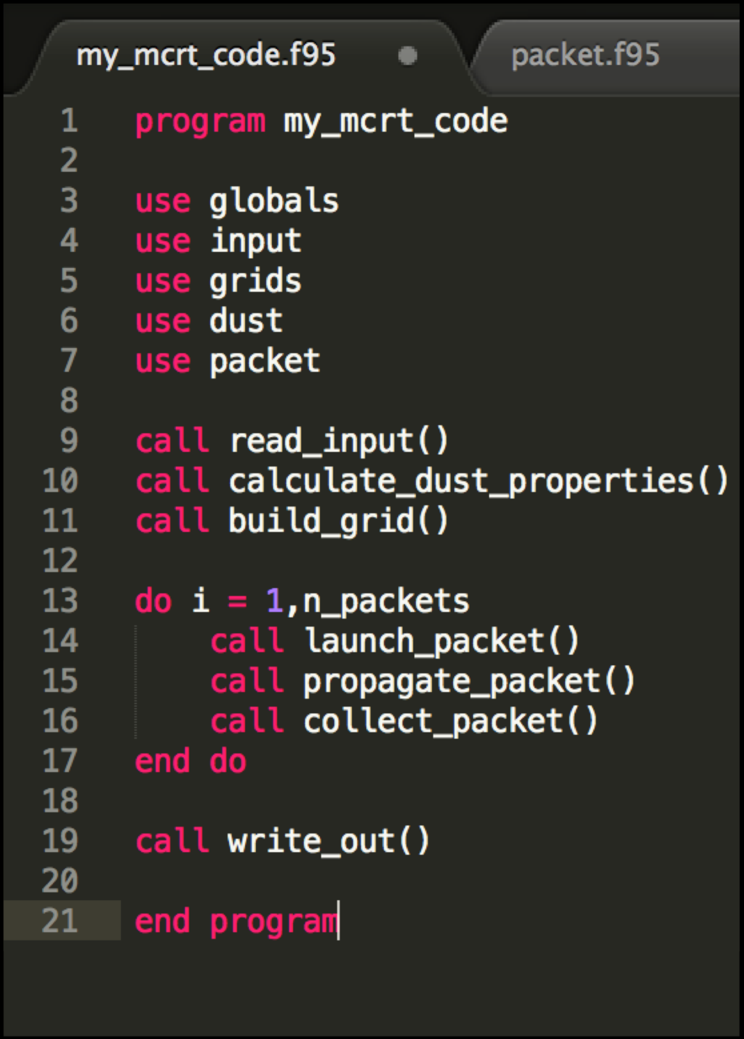




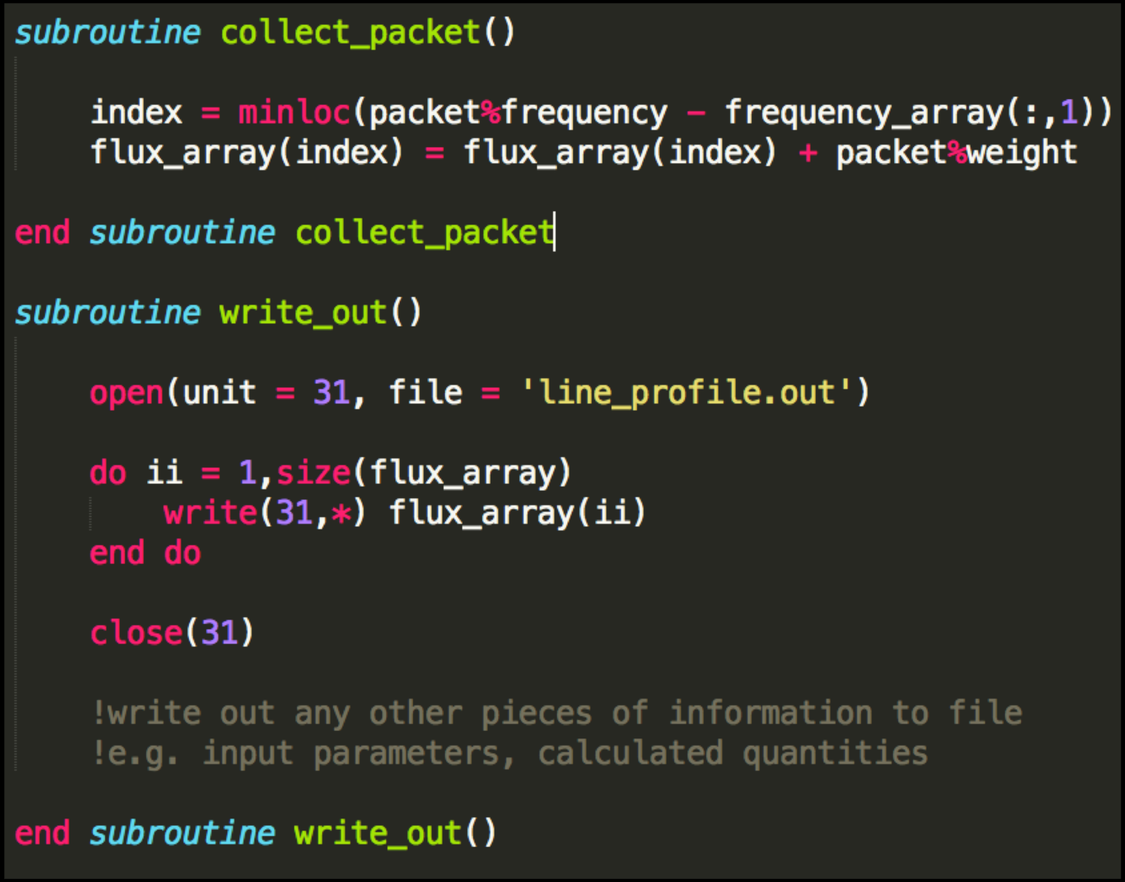
QUESTIONS TO ASK

- What information should be collected?
  - Weights, frequencies, positions, directions…
- How should the information be collated?
  - Binning – what resolution?
- Viewing angles?
- What visualisation/analysis can be used to explore the results of the simulation?
- What graphics/images best represent the model? Convolve to observation?

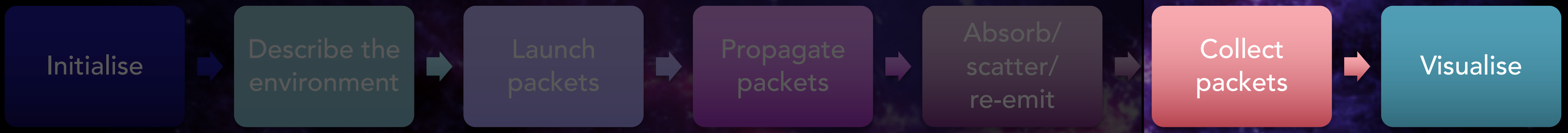


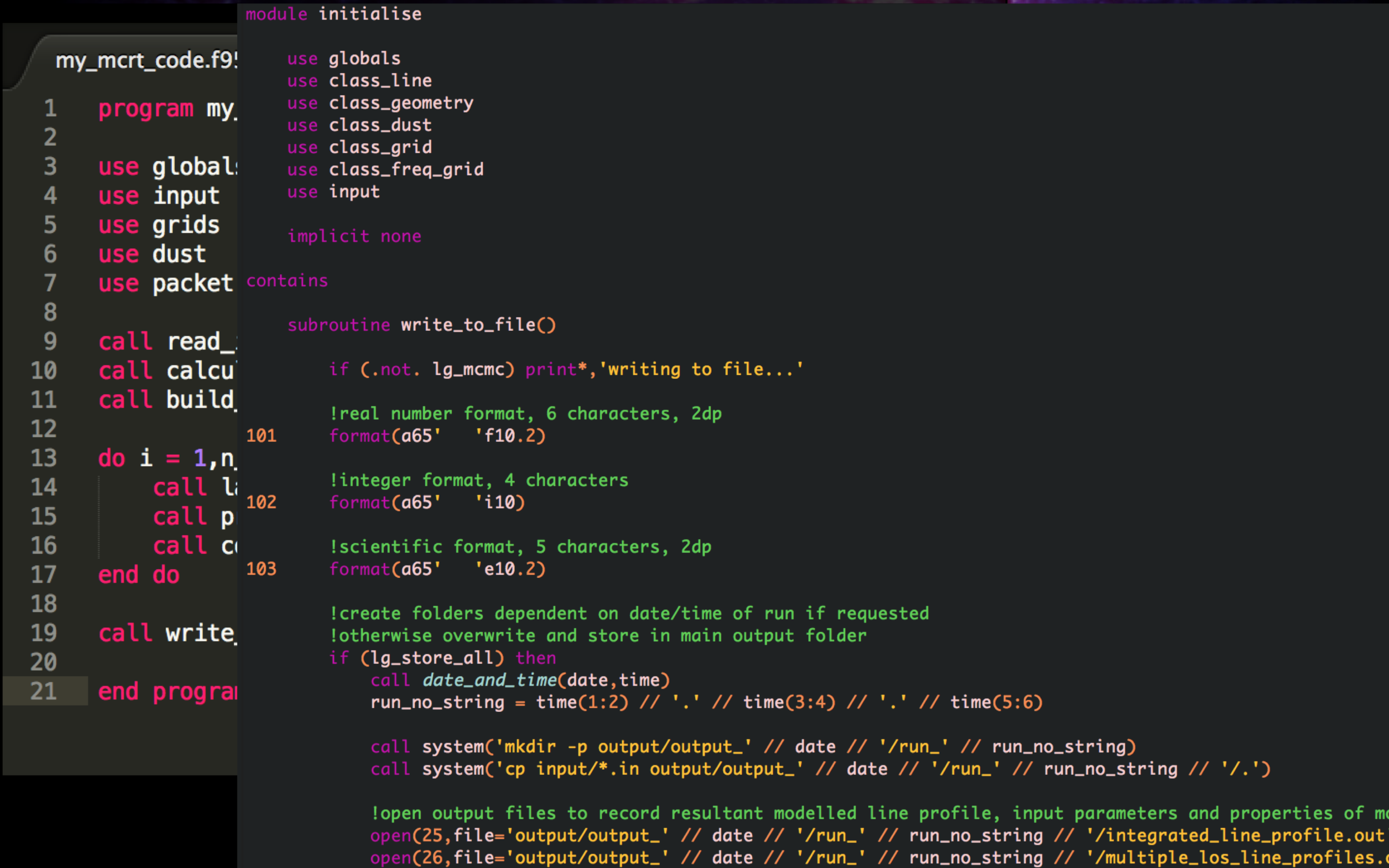


```fortran
program my_mcrt_code

use globals
use input
use grids
use dust
use packet

call read_input()
call calculate_dust_properties()
call build_grid()

do i = 1,n_packets
    call launch_packet()
    call propagate_packet()
    call collect_packet()
end do

call write_out()

end program
```



```fortran
subroutine collect_packet()

    index = minloc(packet%frequency - frequency_array(:,1))
    flux_array(index) = flux_array(index) + packet%weight

end subroutine collect_packet

subroutine write_out()

    open(unit = 31, file = 'line_profile.out')

    do ii = 1,size(flux_array)
        write(31,*) flux_array(ii)
    end do

    close(31)

    !write out any other pieces of information to file
    !e.g. input parameters, calculated quantities

end subroutine write_out()
```





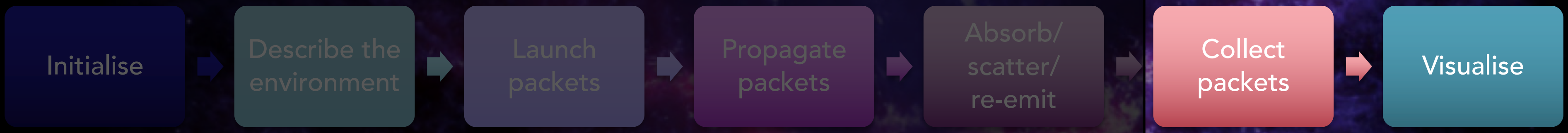
my_mcrt_code.f9

```
program my
                      
use global
use input
use grids
use dust
use packet

call read_
call calcu
call build

do i = 1,n
    call l
    call p
    call c
end do

call write

end progra
```

```
module initialise

    use globals
    use class_line
    use class_geometry
    use class_dust
    use class_grid
    use class_freq_grid
    use input

    implicit none

contains

    subroutine write_to_file()

        if (.not. lg_mcmc) print*,'writing to file...'

        !real number format, 6 characters, 2dp
101     format(a65'    'f10.2)

        !integer format, 4 characters
102     format(a65'    'i10)

        !scientific format, 5 characters, 2dp
103     format(a65'    'e10.2)

        !create folders dependent on date/time of run if requested
        !otherwise overwrite and store in main output folder
        if (lg_store_all) then
            call date_and_time(date,time)
            run_no_string = time(1:2) // '.' // time(3:4) // '.' // time(5:6)

            call system('mkdir -p output/output_' // date // '/run_' // run_no_string)
            call system('cp input/*.in output/output_' // date // '/run_' // run_no_string // '/.')

            !open output files to record resultant modelled line profile, input parameters and properties of mo
            open(25,file='output/output_' // date // '/run_' // run_no_string // '/integrated_line_profile.out
            open(26,file='output/output_' // date // '/run_' // run_no_string // '/multiple_los_line_profiles.o
```

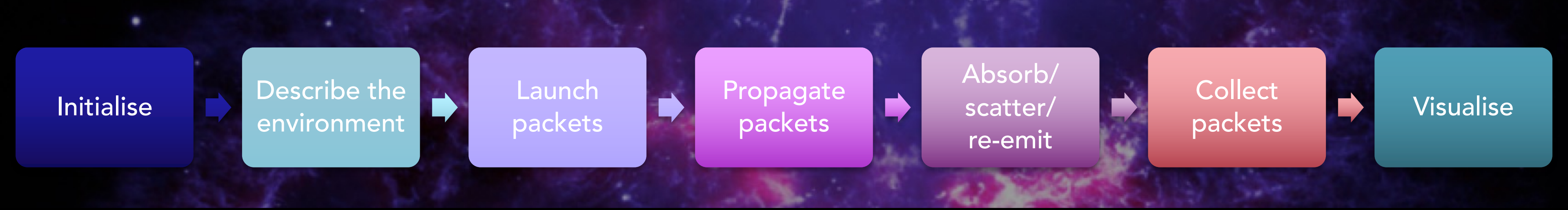
VISUALISATION IS IMPORTANT

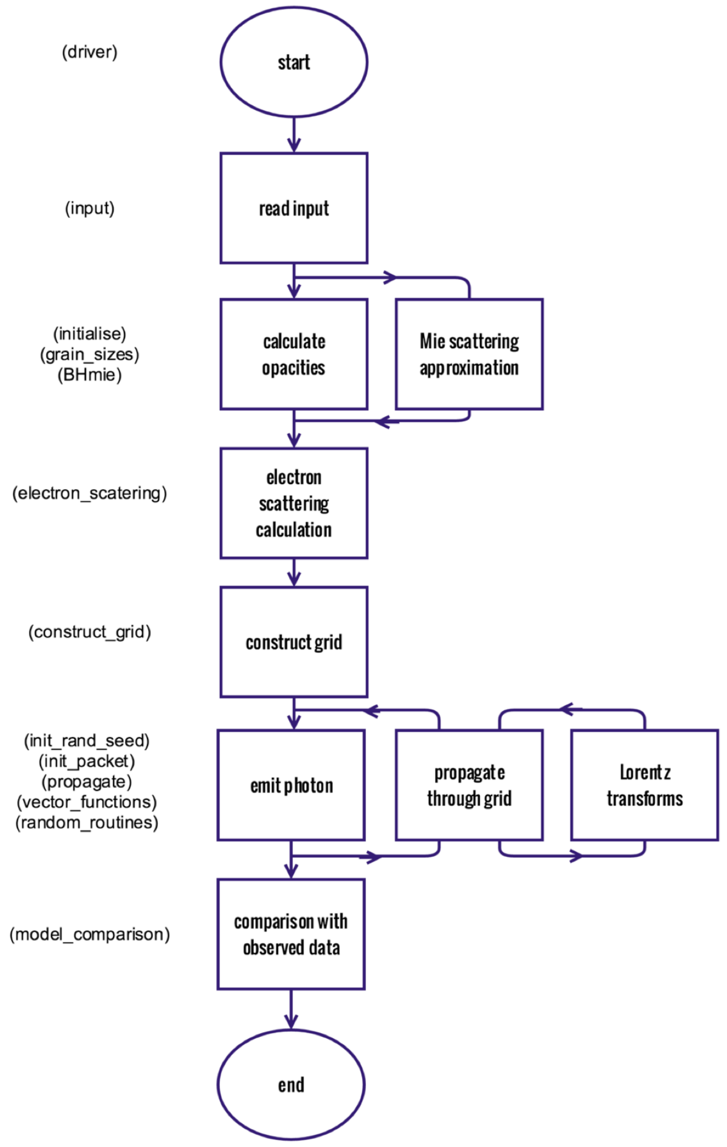
- It is how your model is seen by the outside world
- It allows you to easily assess and analyse the results
- Normally worth writing scripts for visualising then packaging with the code
- Many tools and libraries in e.g. Python, matlab



RECAP

- Important part of the code
- Normally worth investing time in writing post-processing scripts for visualising results
- Try to collect as many outputs as you might be interested in
- Make an automated output option e.g. automated file names based on date/time or input parameters etc.


Initialise
Describe the environment
Launch packets
Propagate packets
Absorb/ scatter/ re-emit
Collect packets
Visualise


(driver)
start
(input)
read input
(initialise)
(grain_sizes)
(BHmie)
calculate opacities
Mie scattering approximation
(electron_scatering)
electron scattering calculation
(construct_grid)
construct grid
(init_rand_seed)
(init_packet)
(propagate)
(vector_functions)
(random_routines)
emit photon
propagate through grid
Lorentz transforms
(model_comparison)
comparison with observed data
end

It didn't quite go like that…

- ~~Spherical grid~~ $\longrightarrow$ Cartesian grid
- Dust absorption
- Dust scattering + electron scattering
- ~~Smooth dust distribution + clumped dust distribution + clumped emissivity distribution~~

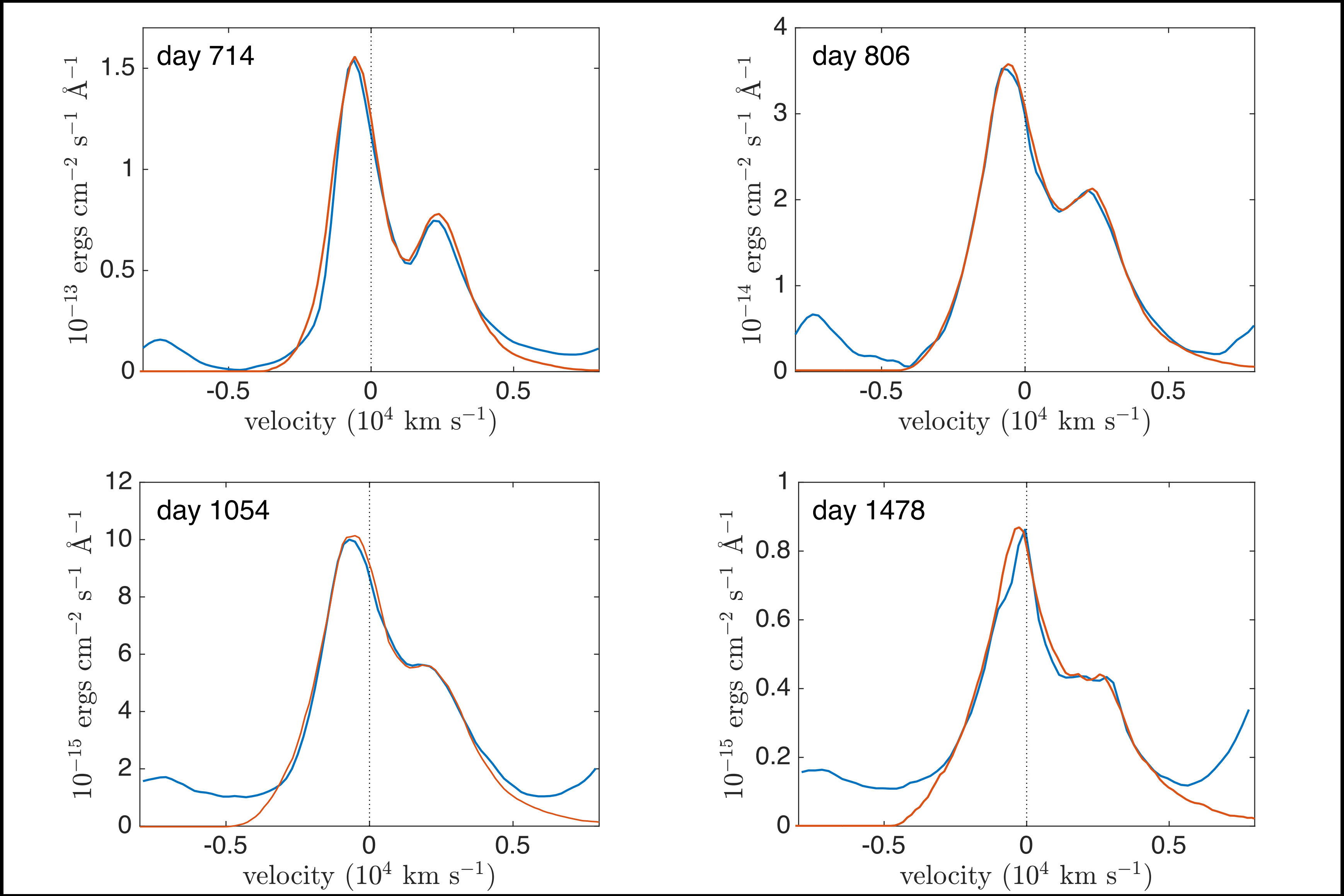
ANY DISTRIBUTION YOU LIKE

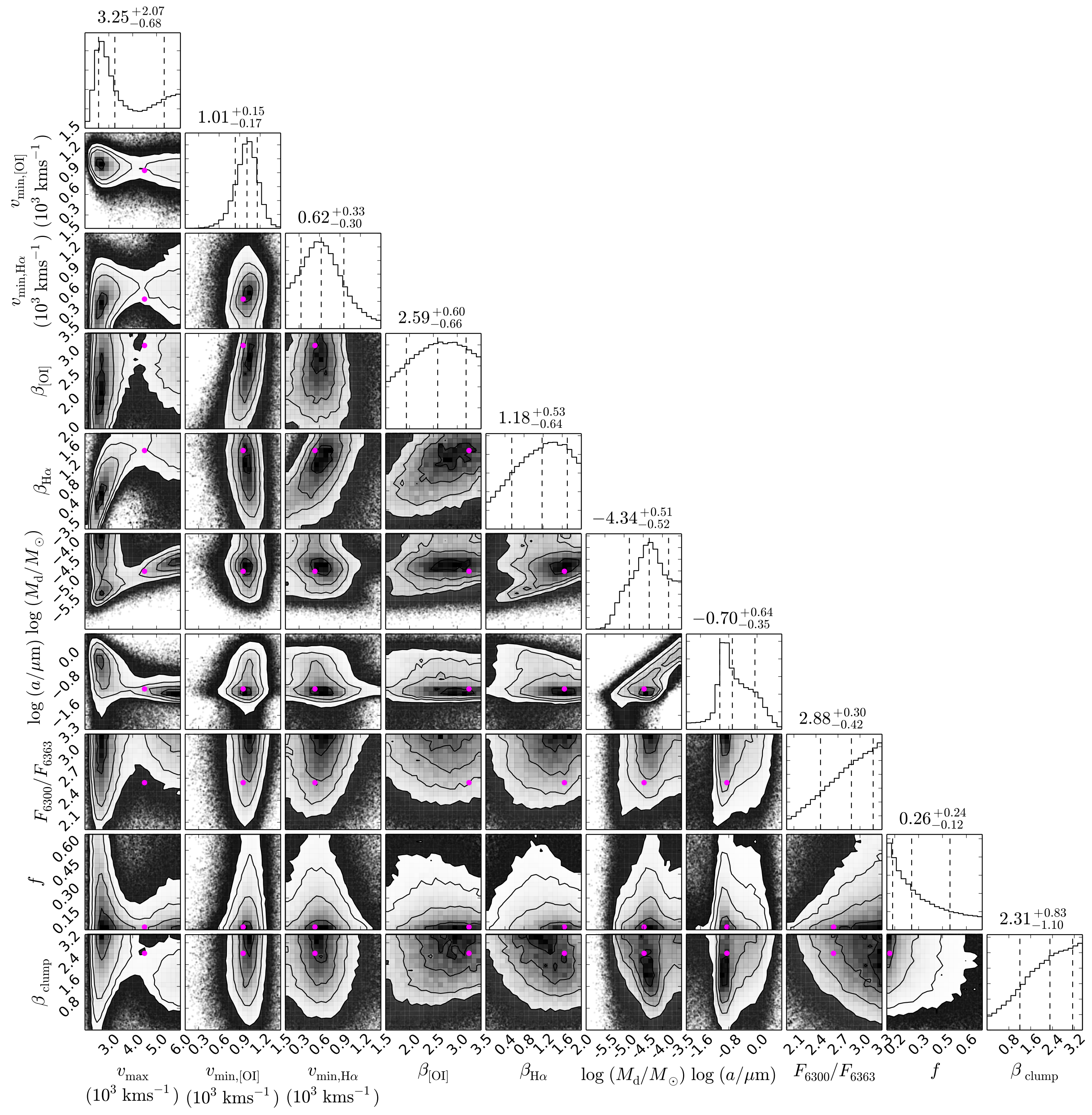
- Monochromatic line packets

BUT ALSO DOUBLETS AND TRIPLETS…

BAYESIAN MCMC WRAPPER IN PYTHON

END RESULT! FITS TO THE SN 1987A [OI]6300,6363 DOUBLET



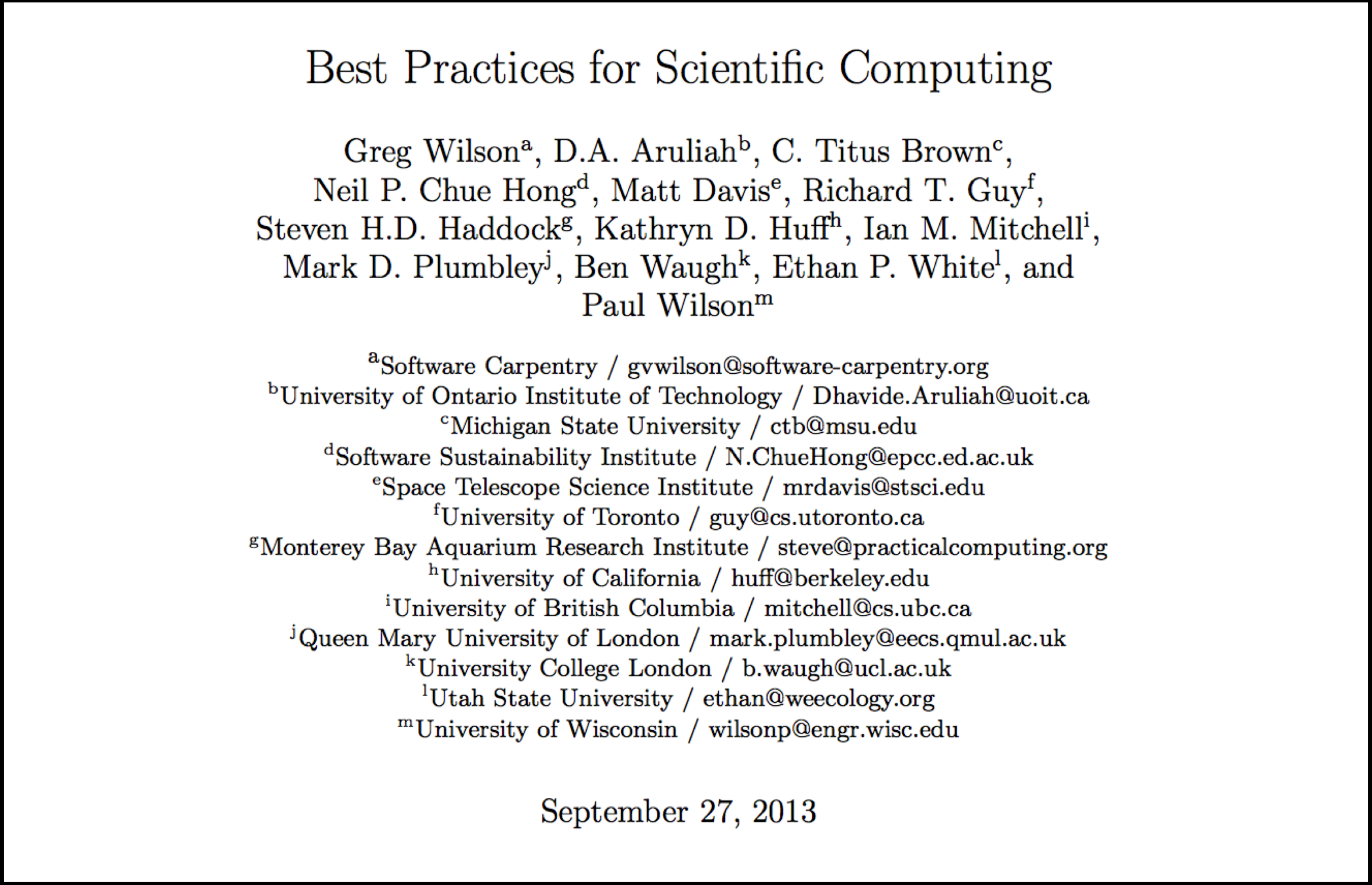


Bayesian Modelling

SN 1987A Hα & [OI] day 714

smooth

gas & dust coupled



Best Practices for Scientific Computing

Greg Wilson[a], D.A. Aruliah[b], C. Titus Brown[c], Neil P. Chue Hong[d], Matt Davis[e], Richard T. Guy[f], Steven H.D. Haddock[g], Kathryn D. Huff[h], Ian M. Mitchell[i], Mark D. Plumbley[j], Ben Waugh[k], Ethan P. White[l], and Paul Wilson[m]

[a]Software Carpentry / gvwilson@software-carpentry.org
[b]University of Ontario Institute of Technology / Dhavide.Aruliah@uoit.ca
[c]Michigan State University / ctb@msu.edu
[d]Software Sustainability Institute / N.ChueHong@epcc.ed.ac.uk
[e]Space Telescope Science Institute / mrdavis@stsci.edu
[f]University of Toronto / guy@cs.utoronto.ca
[g]Monterey Bay Aquarium Research Institute / steve@practicalcomputing.org
[h]University of California / huff@berkeley.edu
[i]University of British Columbia / mitchell@cs.ubc.ca
[j]Queen Mary University of London / mark.plumbley@eecs.qmul.ac.uk
[k]University College London / b.waugh@ucl.ac.uk
[l]Utah State University / ethan@weecology.org
[m]University of Wisconsin / wilsonp@engr.wisc.edu

September 27, 2013

https://arxiv.org/pdf/1210.0530.pdf

- Use version control
- Use modules
- Use libraries and packages
- Test twice (at least!)
- Take care with random numbers
- Parallelise

- Don't assume that only you will be using your code…
  - Comments
  - Clear variable and module names

Ask questions!