Radhyd
Outline

- **Initialization**

  1. Call **read_pack_array_dimensions(n_dim_data)**
     (a) **n_dim_data** (output) array dimension data
     (b) Open file array_dimensions.d
     (c) Call **read_array_dimensions(nread,nprint,iskipp,nx,ny,nz,nez,nnu,nnc,n_proc)**
     Reads in array dimenisons and number of processors assigned to the run.
        i. **nread**: unit number to read from.
        ii. **nprint**: unit number to print diagnostics.
        iii. **iskipp**: read in echo flag.
        iv. **nx**: x-array (radial) dimension. Must be at least 2 + number of active radial quantities.
        v. **ny**: y-array (angular) dimension.
        vi. **nz**: z-array (azimuthal) dimension
        vii. **nez** Neutrino energy array dimension. Must be $\geq$ number of active neutrino energy zones.
        viii. **nnu**: Neutrino flavor array dimension. For the time being, set to 3.
        ix. **nnc**: Number of nuclear species not in NSE.
        x. **n_proc**: Number of processors assigned to the run.
     (d) Close file array_dimensions.d
     (e) Check array dimensions and processor number for compatibility.
     (f) Pack array dimensions in integer array **n_dim_data**
  2. Call **unpack_array_dimenisons(n_dim_data,nx,ny,nz,nez,nnu,nnc,n_proc,n_ray)**
     Unpacks array dimensions and makes them available to each processor.
     (a) **nx**: x-array (radial) dimension. Must be at least 2 + number of active radial quantities.
     (b) **ny**: y-array (angular) dimension.
     (c) **nz**: z-array (azimuthal) dimension
     (d) **nez** Neutrino energy array dimension. Must be $\geq$ number of active neutrino energy zones.
     (e) **nnu**: Neutrino flavor array dimension. For the time being, set to 3.
     (f) **nnc**: Number of nuclear species not in NSE.
     (g) **n_proc**: Number of processors assigned to the run.
     (h) **n_ray**: Number of radial rays per processor.
  3. Call **load_array_module(nx,ny,nz,nez,nnu,nnc,n_proc,n_ray)**
     Loads array dimensions into **array_module**.
  4. Call **initialize**
     (a) Call **dimension_arrays(nx,ny,nz,nez,nnu,nnc,n_proc,n_ray)**
     Dimensions and initializes the module arrays.
        i. **nx**: x-array (radial) dimension. Must be at least 2 + number of active radial quantities.

ii. **ny**: y-array (angular) dimension.

iii. **nz**: z-array (azimuthal) dimension

iv. **nez** Neutrino energy array dimension. Must be $\geq$ number of active neutrino energy zones.

v. **nnu**: Neutrino flavor array dimension. For the time being, set to 3.

vi. **nnc**: Number of nuclear species not in NSE.

vii. **n_proc**: Number of processors assigned to the run.

viii. **n_ray**: Number of radial rays per processor.

ix. Call **dimension_radhyd_arrays(nx,ny,nz,nez,nnu,nnc)**
Allocates the dimensions and initializes the master radhyd arrays

x. Call **dimension_radhyd_ray_arrays(nx,ny,nz,n_ray,nez,nnu,nnc)**
Allocates the dimensions and initializes the primary arrays on a processor

xi. Call **dimension_prb_cntl_ray_arrays(nnu)**

xii. Call **dimension_hydro_arrays(nx,ny,nz,nez,nnu,nnc)**

  A. Call **dimension_boundary_arrays(nx)**
  B. Call **dimension_convect_arrays(nx)**
  C. Call **dimension_hydro_arrays(nx,ny,nz,nez,nnu,nnc)**
  D. Call **dimension_mgfld_remap_arrays(nx,nez,nnu,nnc)**
  E. Call **dimension_psi0p_arrays(nx,nez,nnu)**
  F. Call **dimension_shock_arrays(nx)**

xiii. Call **dimension_mgfld_arrays(nx,nez,nnu,n_ray)**

  A. Call **dimension_abem_arrays(nx,nez,nnu,n_ray)**
  B. Call **dimension_brem_arrays(nx,nez,nnu,n_ray)**
  C. Call **dimension_incrmnt_arrays(nx,nez,nnu,n_ray)**
  D. Call **dimension_mdl_cnfg_arrays(nx)**
  E. Call **dimension_nu_dist_arrays(nx,nez,nnu)**
  F. Call **dimension_nu_energy_grid_arrays(nez,nnu)**
  G. Call **dimension_pair_arrays(nx,nez,nnu,n_ray)**
  H. Call **dimension_scat_a_arrays(nx,nez,nnu,n_ray)**
  I. Call **dimension_scat_e_arrays(nx,nez,nnu,n_ray)**
  J. Call **dimension_scat_i_arrays(nx,nez,n_ray)**
  K. Call **dimension_scat_n_arrays(nx,nez,nnu,n_ray)**
  L. Call **dimension_scat_nn_arrays(nx,nez,nnu,n_ray)**
  M. Call **dimension_t_cntrl_arrays(nx,nnu)**

xiv. Call **dimension_edit_arrays(nx,nez,nnu)**

xv. Call **dimension_eos_bck_arrays(nx)**

xvi. Call **dimension_eos_snc_arrays(nx)**

xvii. Call **dimension_eos_ls_arrays(nx)**

xviii. Call **dimension_nucbrn_arrays(nx,nnc)**

xix. Call **dimension_e_advct_arrays(nx,nez,nnu)**

xx. Call **dimension_evh1_sweep_arrays(nx,ny,nz)**

xxi. Call **dimension_evh1_zone_arrays(nx,ny,nz)**

    xxii. Call **dimension_evh1_bound_arrays(nnc)**

(b) Call **initialize_variables(nx,ny,nz,nez,nnu,nnc)**

  i. Initializes variables not initialized in the dimension variable calls
  ii. **nx**: x-array (radial) dimension. Must be at least 2 + number of active radial quantities.
  iii. **ny**: y-array (angular) dimension.
  iv. **nz**: z-array (azimuthal) dimension
  v. **nez** Neutrino energy array dimension. Must be $\geq$ number of active neutrino energy zones.
  vi. **nnu**: Neutrino flavor array dimension. For the time being, set to 3.
  vii. **nnc**: Number of nuclear species not in NSE.
  viii. Call **initialize_global_var**
  ix. Call **initialize_cycle_arrays**
  x. Call **initialize_it_tol_arrays**
  xi. Call **initialize_bomb_arrays**
  xii. Call **initialize_rezone_arrays**

(c) Call **problem_read(c_init_data,c_radhyd_data,c_eos_data,i_radhyd_data, i_trans_data,i_e_advct_data,i_edit_data, i_hydro_data, i_nuc_data, i_model_data, d_radhyd_data, d_eos_data, d_trans_data,d_e_advct_data, d_edit_data, d_hydro_data, d_nuc_data, d_model_data)**

  i. Open **Data3/reset.d**
  ii. Open **Data3/superdump.d**
  iii. Open **Data3/rstdmp1.d**
  iv. Open **Data3/rstdmp2.d**
  v. Call **read_pack(c_init_data,c_radhyd_data,c_eos_data,i_radhyd_data, i_trans_data,i_e_advct_data,i_edit_data, i_hydro_data, i_nuc_data, i_model_data, d_radhyd_data, d_eos_data, d_trans_data,d_e_advct_data, d_edit_data, d_hydro_data, d_nuc_data, d_model_data)** Read and broadcast initial data and run keys

    A. Call **read_pack_init(nrrstp,c_init_data)**
    Reads 'head" and "nrst"
    B. If nrst = 0 ....
    C. Call **radhyd_read(c_radhyd_data,i_radhyd_data,d_radhyd_data)**
    .....Call **read_radhyd_keys(nreadp,nprint,iskip,c_radhyd_data,i_radhyd_data,d_radh**
    D. Call **eos_read(c_eos_data,d_eos_data)**
    .....Call **read_pack_eos_keys(nreadp,nprint,iskip,c_eos_data,d_eos_data)**
    E. Call **transport_read)(i_trans_data,d_trans_data)**
    .....Call **read_pack_transport_keys(nreadp,nprint,iskip,nez,nezp1,nnu, i_trans_data,d_trans_data)**
    F. Call **e_advct_read(i_e_advct_data,d_e_advct_data)**
    .....Call **read_pack_e_advct_keys(nreadp,nprint,iskip,nnu,i_e_advct_data, d_e_advct_data)**
    G. Call **edit_read(i_edit_data,d_edit_data)**
    .....Call **read_pack_edit_keys(nreadp,nprintp,iskip,nez,nnu,i_edit_data, d_edit_data)**

H. Call **hydro_read(i_hydro_data,d_hydro_data)**
   .....Call **read_pack_hydro_keys(nreadp,nprint,iskip,nx,i_hydro_data, d_hydro_data)**

I. Call **nuc_read(i_nuc_data,d_nuc_data)**
   .....Call **read_pack_nuclear_keys(nreadp,nprint,iskip,nx,nnc,i_nuc_data, d_nuc_data)**

J. Call **model_read(i_model_data,d_model_data)**
   .....Call **read_initial_model(nread,nprint,iskipp)**

K. If nrst $\neq 0$ ....

L. Call **readst(nwrstp,iskip)**

vi. Close **Data3/reset.d**

vii. Close **Data3/rstdmp1.d**

viii. Close **Data3/rstdmp2.d**

(d) Call **data_check**
Check consistency of data.

(e) Call **rezone(c_radhyd_data,i_radhyd_data,d_radhyd_data,i_model_data, d_model_data,l_rezone_data,d_rezone_data,i_nuc_data, d_nuc_data)**
Unpack lagr, rezn, ngeomy, ngeomz, jm, and r. Set courant, xmin, and xmax. Initialize ymin, ymax, zmin, zmax.

i. set **courant** and **lagrangian**

ii. If rezn = 'ye'

   A. if lagrangian = true, Call **lagregrid**

   B. if lagrangian = false, Call **eulregrid**

iii. If rezn = 'no'

   A. imax = jm - 1

   B. load nse(j) in eos_snc_module

   C. Load quantities from MGFLD to RadHyd variables.

   D. Load EVH1 boundary conditions from MGFLD.

iv. if ndim ≥ 2, build a j grid. Set ymin and ymax,

v. if ndim = 3, build a k grid. Set zmin and zmax,

vi. Pack **lagrangian, imax, xmin, xmax, ymin, ymax, zmin, zmax, courant, y-coordinates, z-coordinates**

(f) Call **Call unpack_arrays(c_init_data,c_radhyd_data,c_eos_data,i_radhyd_data, i_trans_data,i_e_advct_data,i_edit_data, i_hydro_data, i_nuc_data, i_model_data, l_rezone_data, d_radhyd_data, d_eos_data,d_trans_data, d_e_advct_data, d_edit_data, d_hydro_data, d_nuc_data, d_model_data, d_rezone_data)**

i. Call **unpack_init(c_init_data)**

ii. Call **unpack_radhyd_keys(c_radhyd_data,i_radhyd_data,d_radhyd_data)**

iii. Call **unpack_radhyd_ray_keys(c_radhyd_data,i_radhyd_data,d_radhyd_data)**

iv. Call **unpack_rezone_arrays(l_rezone_data,d_rezone_data)**

v. Call **unpack_eos_keys(c_eos_data,d_eos_data)**

vi. Call **unpack_transport_keys(nez,nezp1,nnu,i_trans_data,d_trans_data)**

vii. Call **unpack_e_advct_keys(nnu,i_e_advct_data,d_e_advct_data)**

viii. Call **upack_edit_keys(nez,nnu,i_edit_data,d_edit_data)**

ix. Call **unpack_hydro_keys(nx,i_hydro_data,d_hydro_data)**

x. Call **unpack_nuclear_keys(nx,nnc,n_ray,i_nuc_data,d_nuc_data)**

xi. Call **unpack_initial_model(nx,n_ray,i_model_data,d_model_data)**

(g) Call **problem_setup(nx,nnu,n_ray)**

i. If (nrst = 0) **<u>Loop over j_ray from 1 to n_ray</u>** Call **mgfld_setup(imin,imax,nx,j_ray,n_r**

A. **imin**: (input) inner physical x-zone index

B. **imax**: (input) outer physical x-zone index

C. **nx**: (input) logical x-array dimension

D. **j_ray**: (input) index denoting a specific radial ray

E. **n_ray**: (input) number of rays assigned to a processor

F. **nnu**: (input) neutrino flavor extent

G. **ka**: (input) value of z-zone index

H. **rho_c(:,:,:)**: (input) density (g/cm^3)

I. **t_c(:,:,:)**: (input) temperature (MeV)

J. **ye_c(:,:,:)**: (input) electron fraction

K. **x_e(:)**: (input) radial coordinate (face) (cm)

L. **dx_c(:)**: (input) radial coordinate thickness (cm)

M. **u_c(:,:,:)**: (input) radial velocity (face) (cm/s)

N. **xn_c(:,:,:)**: (input) abundance mass fractions

O. **be_nuc_c(:,:)**: (input) binding energies

P. **a_nuc_c(:,:)**: (input) nuclear mass numbers

Q. **z_nuc_c(:,:)**: (input) nuclear charge numbers

ii. ....

A. Initialize radial array index boundaries (set jm, jmin, jmax, jmaxp, jnumax, jnumaxp)

B. Set quatities at inner edge of configuration

C. Transfer zone-centered independent variables to mgfld arrays

D. Transfer zone-edgeed independent variables to mgfld arrays

E. Compute Newtonian rest masses

F. Call **pblmst1** Modify problem before eos table setup (if appropriate)

G. Call **esrgnz_x** Load equation of state

H. Call **eqstz_x**

I. Call **gammaz_x**

J. Set m-1 and m+1 values of independent variables

K. Call **genst_rel** Compute GR quantities if irelhy = 1

L. Call **agr_cal** Time dilation factors

M. Call **gamgr_nu_cal** Put GR gammas in neutrino variables

N. Call **gamgra_nu_cal** Put updated GR gammas in neutrino variables

O. Call **agr_nu_cal** Put time dilation factors in neutrino variables

P. Call **agra_nu_cal** Put updated time dilation factors in neutrino variables

Q. Call **e_zone** Compute neutrino group energies at infinity

R. Call **enu_cal** Compute GR neutrino energy arrays

S. Call **pre_trans** Compute quantities needed for neutrino transport
T. Call **pblmst2** Modify problem given the neutrino energies (if appropriate)
U. Call **gennur** Read in and regrid Wick's neutrino interaction rates
V. Call **abemset** Compute absorption and emission opacities on table corners
W. Call **scataset** Compute Wick's scattering opacities on table corners
X. Call **scateset** Compute neutrino-electron scattering opacities on table corners
Y. Call **scatiset** Compute isoenergetic scattering opacities on table corners
Z. Call **pairset** Compute pair annihilation opacities on table corners
A. Call **bremset** Compute nucleon-nucleon bremsstrahlung opacities on table corners
B. Call **scatnset** Compute neutrino-nucleon elastic scattering opacities on table corners
C. Call **scatnnset** Compute neutrino-nucleon inelastic scattering opacities on table corners
D. Call **abemrate** Interpolate absorption and emission opacities
E. Call **sctarate** Interpolate Wick's opacities
F. Call **scterate** Interpolate neutrino-electron scattering opacities
G. Call **sctirate** Interpolate isoenergetic scattering opacities
H. Call **pairrate** Interpolate pair annihilation opacities
I. Call **bremrate** Interpolate nucleon-nucleon bremsstrahlung opacities
J. Call **sctnrate** Interpolate neutrino-nucleon elastic scattering opacities
K. Call **sctnnrate** Interpolate neutrino-nucleon inelastic scattering opacities
L. Call **nu_number** Compute the neutrino number and energy
M. Call **mfp_cal** Compute neutrino inverse mean free paths
N. Call **nu_sphere** Compute location of neutrinospheres
O. Call **diffc** Compute neutrino diffusion coefficients
P. Call **nu_stress** Compute neutrino stresses
Q. Call **eddington** Compute neutrino flux and eddington
R. Call **nu_U** Compute neutrino energy density

iii. If (nrst $\neq 0$) Call **genrst** Reinitialize problem from restart data

(h) Call **load_radhyd_ray_arrays(nx,nnu,nnc,n_ray)**
(i) Call **load_evh1_arrays**
(j) Call **time_step_check(n_ray)**
Checks that the given time step is not larger than the minimum time step given by the Courant condition.

5. Call **radhyd_to_edit(j_ray_min,j_ray_max,i_edit)**
   (a) **j_ray_min**: (input) minimum ray index
   (b) **j_ray_max**: (input) maximum ray index
   (c) **i_edit**: (input) edit flag
   (d) <u>**Loop over j_ray from j_ray_min to j_ray_max**</u>
      i. **CALL mgfld_edit_in(is,ie,idim,j_ray_min,j_ray_max,n_ray,rho_ci,rho_c,t_c,ye_c, x_e,u_c,psi0_c,psi1_e,dtnph,time,i_editp,ncycle,xn_c,be_nuc_c,a_nuc_c,z_nuc_c, nse_c, nedc,nedmi,nedma,nedh,nedps,nedu,nedy,nedsc,nedn,nedng)**

            A. Load edit counters into edit_module
            B. Load state variables into mdl_cnfg_module
            C. Load neutrino distribution functions into nu_dist_module
            D. Load composition variables into eos_snc_module and nucbrn_module
            E. **Call eqstz_x(jmin,jmaxp,j_ray)**
            F. **Call gammaz_x(jmin,jmaxp,j_ray)**
            G. **Call mgfld_edit(j_ray,i_editp,first)**
        ii. **Call mgfld_edit_out(nedc,nedmi,nedma,nedh,nedps,nedu,nedy,nedsc, nedn,nedng)**
            Bring back updated edit counters
    (e) **End Loop over j_ray**

Cycling MGFLD Transport

- Call **cycle**
  Updates cycle number, opens **Data3/cycle.d** prints cycle number, closes **Data3/cycle.d**
- Initialize increment arrays
- Initialize **svel**
- **Loop over j_ray from 1 to n_ray**
  1. **Call store_int_radhyd_var(j_ray)**
     Stores initial values of state variables in **radhyd_variable_module** (variables end with an "i")
  2. **Call radhyd_to_evh1_x_lagr(nx,j_ray,n_ray)**
     (a) **Call evh1_x_lagr(imin,imax,nx,j_ray,n_ray,x_e,dx_c,x_c, y_e,dy_c,y_c,z_e, dz_c,z_c,rho_c,t_c,ye_c,ei_c,u_c,v_c,w_c,nu_str_c,time,dtime)**
         i. Set **nmin, nmax, ntot**
         ii. Load padded arrays for Lagrangian update
         iii. Load initial values in mgfld_remap_module
         iv. Initialize **dt**
         v. **Call etotal(.false.)**
         vi. **Call sweepx(j_ray)**
             A. **Call tgvndeye_sweep(nmin,nmax,j_ray,r0i,r0i)**
                Update **t, p, s, gc, ge**
             B. **Call sweepbc(nleftx,nrightx,j_ray)**
                Fill ghost zones
             C. **Call volume (ngeomx)**
             D. **Call paraset(ntot,zparax,dx,xa,nmin-4,nmax+4)**
             E. **Call e_compose(xa,dx,ntot,zparax)**
             F. **Call ppm(ngeomx,ntot,zparax,j_ray)**
             G. **Call tgvndeye_sweep(nmin,nmax,j_ray,r,r0i)**
         vii. Put variables advanced by Lagrangian hydro step back in radhyd ray arrays
- **Call radhyd_to_eos_x_reset(nx, i_ray_dim, i_ray, nnc)**

- **Call nu_transport_inout(imin, imax, nx, i_ray, i_ray_dim, nez, nnu, nprintp, rhop, tp, yep, rhobarp, rp, up, psi0p, psi1p, nu_strp, dt, jdt, dtnph_trans, dtime_trans)**

  1. **Call mgfld_reset(i_ray)**
     Reset opacity tables if dictated by criteria.
  2. **Call mgfld_transport(i_ray, i_ray_dim, nx, nez, nnu, jdt, dtnph_trans))**
     Sets up for neutrino transport
     - (a) **Call extrap**
     - (b) **Call w_cal(jr_min, jr_max, i_ray, rho, t, ye, wgr, nx)**
     - (c) **Call agr_cal(jr_min, jr_max, i_ray, rho, t, ye, r, nx)**
     - (d) **Call agr_nu_cal(jr_min, jr_max)**
     - (e) **Call gamgr_nu_cal(jr_min, jr_max)**
     - (f) **Call enu_cal(jr_min, jr_max)**
     - (g) **Call nu_adv(jr_min, jr_max, i_ray, i_ray_dim, rho, t, ye, r, rstmss, nx, nez, nnu, jdt, dtnph_trans)**
       - i. **Call pre_trans(jr_min, jr_max, rho, r, nx, nnu)**
         Computes **ncoefa, ecoefa, ecoefae, rjmh, area, areajmh, vol, drjmh, drjmh_inv**
       - ii. **Call nu_sphere(jr_min, jr_max, i_ray, i_ray_dim, r, rho, t, rstmss, nx, nez, nnu, j_sphere, r_sphere, d_sphere, t_sphere, m_sphere)**

       - iii. **Call nu_trans(jr_min, jr_max, i_ray, i_ray_dim, rho, t, t_new, ye, ye_new, r, rstmss, nx, nez, nnu, jdt, dtnph_trans)**

         - A. **Call psi_bd(jr_max, k, 1, radius, nx, psi_ratio)**
         - B. **Call psi1_cal(jr_min, jr_max, i_ray, i_ray_dim, rho, t_new, ye_new, radius, rstmss, u_vel, psi0, psi1, nx, nez, nnu, it)**
         - C. **Call ddc_dpsi(jr_min, jr_max, i_ray, i_ray_dim, radius, ddcpsjph, ddcpsjmh, nx, nez, nnu)**
         - D. **Call abemrate(jr_min, jr_max, i_ray, rho, t_new, ye_new, nx)**
         - E. **Call scterate(1, jr_min, jr_max, i_ray, rho, t_new, ye_new, nx)**
         - F. **Call pairrate(1, jr_min, jr_max, i_ray, rho, t_new, ye_new, radius, nx)**
         - G. **Call bremrate(1, jr_min, jr_max, i_ray, rho, t_new, ye_new, radius, nx)**
         - H. **Call sctnrate(1, jr_min, jr_max, i_ray, rho, t_new, ye_new, nx)**
         - I. **Call sctnnrate(1, jr_min, jr_max, i_ray, rho, t_new, ye_new, nx)**
         - J. **Call sctnArate(1, jr_min, jr_max, i_ray, rho, t_new, ye_new, nx)**
         - K. **Call ludcmp(O, ne_4+1, 4*nez+1, indx, d_perm)**
         - L. **Call lubksb(O, ne_4+1, 4*nez+1, indx, O_inv(1,k)**
         - M. **Call eqstt_x(2, j, i_ray, rho(j), t_new(j), ye_0(j), e, dedd, dedt, dedy)**
         - N. **Call tgvndeye_x(j, i_ray, rho(j), e - de(j), ye_new(j), t_new(j), t_new(j))**

O. **Call psi1_cal(jr_min, jr_max, i_ray, i_ray_dim, rho, t_new, ye_new, radius, rstmss, u_vel, psi0, psi1, nx, nez, nnu, 2)**

P. **Call flux(jr_min, jr_max, n)**

(h) **Call nu_number(jr_min, jr_max, n, i_ray, nx, nez, nnu, r, u, psi0, psi1)**

(i) **Call eqstz_x(jr_min, jmaxp, i_ray)**

(j) **Call gammaz_x(jr_min, jr_max, i_ray)**

(k) **Call nu_stress_x(jr_min, jr_max, i_ray, rho, r)**

Cycling Remap_x

- Call **radhyd_to_remap_x(nx, i_ray, i_ray_dim, nez, nnu, nnc)**

1. CALL **remap_x_inout(imin, imax, nx, i_ray,i_ray_dim nez, nnu, ls, le, nnc, x_el, dx_cl, x_cl, x_ef, dx_cf, x_cf, rho_c, t_c, ye_c, ei_c, u_c, v_c, w_c, psi0_c, xn_c, a_nuc_rep_c, z_nuc_rep_c, be_nuc_rep_c)**

   (a) Load final coordinate values, pad with ghost zones

      i. load **xa0** and **dx0** in evh1_sweep

      ii. load **xa0** and **dx0** in mgfld_remap_module

   (b) Load Lagrangian coordinates, pad with ghost zones

      i. load **xa** and **dx** in evh1_sweep

      ii. load **xa** and **dx** in mgfld_remap_module

   (c) Load state variables, pad with ghost zones

      i. load **r**, **temp**, **ye**, **u**, **v**, **w**, **ei** in evh1_sweep

      ii. load **r**, **temp**, **ye** in mgfld_remap_module

   (d) Load radiation variables, pad with ghost zones

      i. load **psi0_re** = **psi0** in mgfld_remap_module

   (e) Load abundances, (no padding)

      i. load **a_nuc**, **z_nuc**, **be_nuc**, **xn** in nucbrn_module

   (f) Call **sweepbc(nleftx, nrightx, nmin, nmax, i_ray)**
   Load boundary values in ghost zones of state variables and load in evh1_sweep

   (g) Call **volume (ngeomx)**
   Compute volumes of zones with Lagrangian coordinate values and with final coordinate values

   (h) Call **paraset(imax+12, zparax, dx, xa, nmin-4, nmax+4)**
   Calculates coefficients for PPM on the Lagrangian coordinates; store zparax in evh1_zone

   (i) Call **pre_remap_psi(nleftx, nrightx, nnu)**

      i. Call **coord_bc(nleft, nright, xa, dx, xa0, dx0, imax+12)**
      Computes coordinate boundary values and returns them. They are then loaded into mgfld_remap_module

      ii. Load left (inner) ghosts for **psi0_re**, store in mgfld_remap_module

      iii. Load right (outer) ghosts for **psi0_re**, store in mgfld_remap_module

   (j) Call **remap_psi_x(ngeomx, is, ie, i_ray, nx, nez, nnu)**

    i. Call **parabola(nmin-1, nmax+1, imax+12, zparax, psi, dpsi, psi6, psil, dm, 0, 0)**
       Compute the PPM interpolation coefficients for psi0 for each k and n; use zparax from evh1_zone
    ii. Calculate the volume of the overlapping subshells (delta)
    iii. Compute psi0 to be advected
    iv. Advect psi0 by moving the subshell quantities into the appropriate Eulerian zone.
    v. Restore psi0
    vi. Book keeping

(k) CALL **pre_remap_comp(nleftx, nrightx, i_ray, i_nnse, ldim)**

    i. Initialize **comp**
    ii. Find nse - nonnse boundary
    iii. Load **comp**
    iv. CALL **coord_bc(nleft, nright, xa, dx, xa0, dx0, imax+12)**
        Computes coordinate boundary values
    v. CALL **sweepbc_r(nleft ,nright)**
        A. CALL **coord_bc(nleft, nright, xa, dx, xa0, dx0, imax+12)**
        B. Load left (inner) ghosts for **r**.
        C. Load right (outer) ghosts for **r**.
    vi. Load left (inner) ghosts for **comp** and **r**.
    vii. Load right (outer) ghosts for **comp** and **r**.

(l) Call **remap_comp(ngeomx, i_nnse, i_ray, nx, ldim)**

    i. Initialize **fluxbe** to 0 and store initial values of **xn** in **xn0**.
    ii. Calculate volumes before and after Eul remap.
    iii. Call **eos_nnse_e(j, r(n), temp(n), ye(n), xn_t, nnc, a_nuc(j), z_nuc(j), be_nuc(j), e_ph, e_elec, e_drip, e_hvy, e_bind, e_no_bind, e_total)**
         Computes the total binding energy for zones not in nse.
    iv. Set **eb(n) = eb(nminc)** for n < nminc.
    v. Load boundary values of binding energy.
    vi. Call **parabola(nmin-1, nmax+1, imax+12, zparax, r, dr, r6, rl, dm, 0, 0)**
        Computes PPM coefficients for the density.
    vii. Call **parabola(nminc-1, nmax+1, imax+12, zparax, cmp, dcmp, cmp6, cmpl, dm, 0, 0)**
         Computes PPM coefficients for the composition.
    viii. Calculate the volume of the overlapping subshells (**delta**)
    ix. Calculate the total mass, **fluxr**, and the mass, **fluxcmp**, of each specie to be advected.
    x. Compute the total binding energy, **fluxbe**, of the mass being transferred.
    xi. Compute the total electron fraction, **fluxye_comp**, being transferred.
    xii. Advect mass and composition mass fractions by moving the subshell quantities into the appropriate Eulerian zone.
    xiii. Restore composition to **xn**