FLASH Code Tutorial

a user's perspective

Robi Banerjee
Hamburger Sternwarte
banerjee@hs.uni-hamburg.de

FLASH Code

Some background information:

- **3D AMR MHD multi-physics code**
- main code development at

  FLASH Center at the University of Chicago

  DOE NNSA-ASC OASCR Flash Center

  web: `http://flash.uchicago.edu`

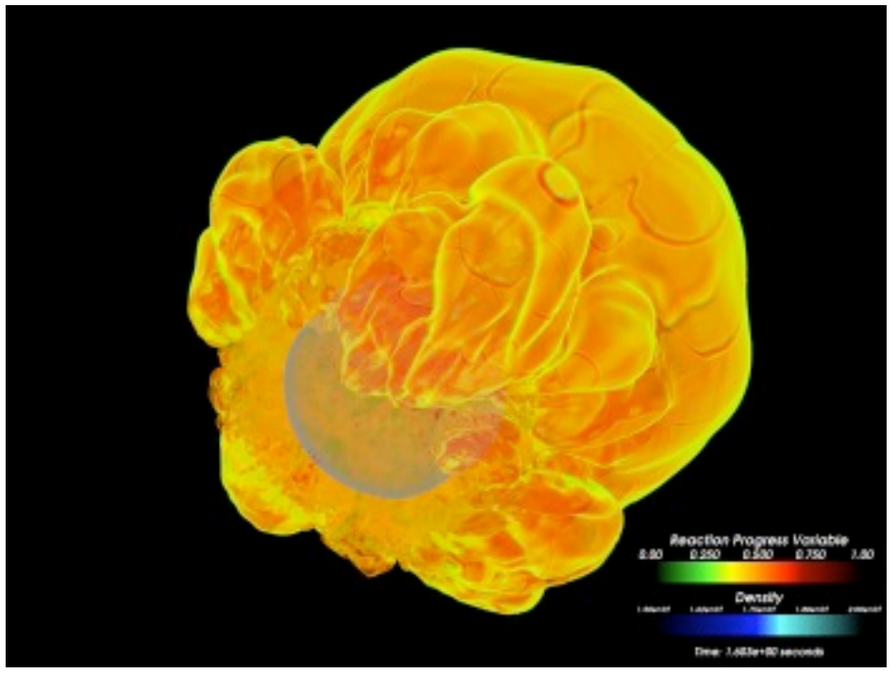
  currently about 20 people at the Flash Center
- contribution from many individual groups, e.g.
  - hybrid-characteristic ray-trace (Rijkhorst et al., Peters et al.)
  - sink particles (Federrath et al.)
  - Barns & Hut tree algorithm (Wunsch)
  - chemistry

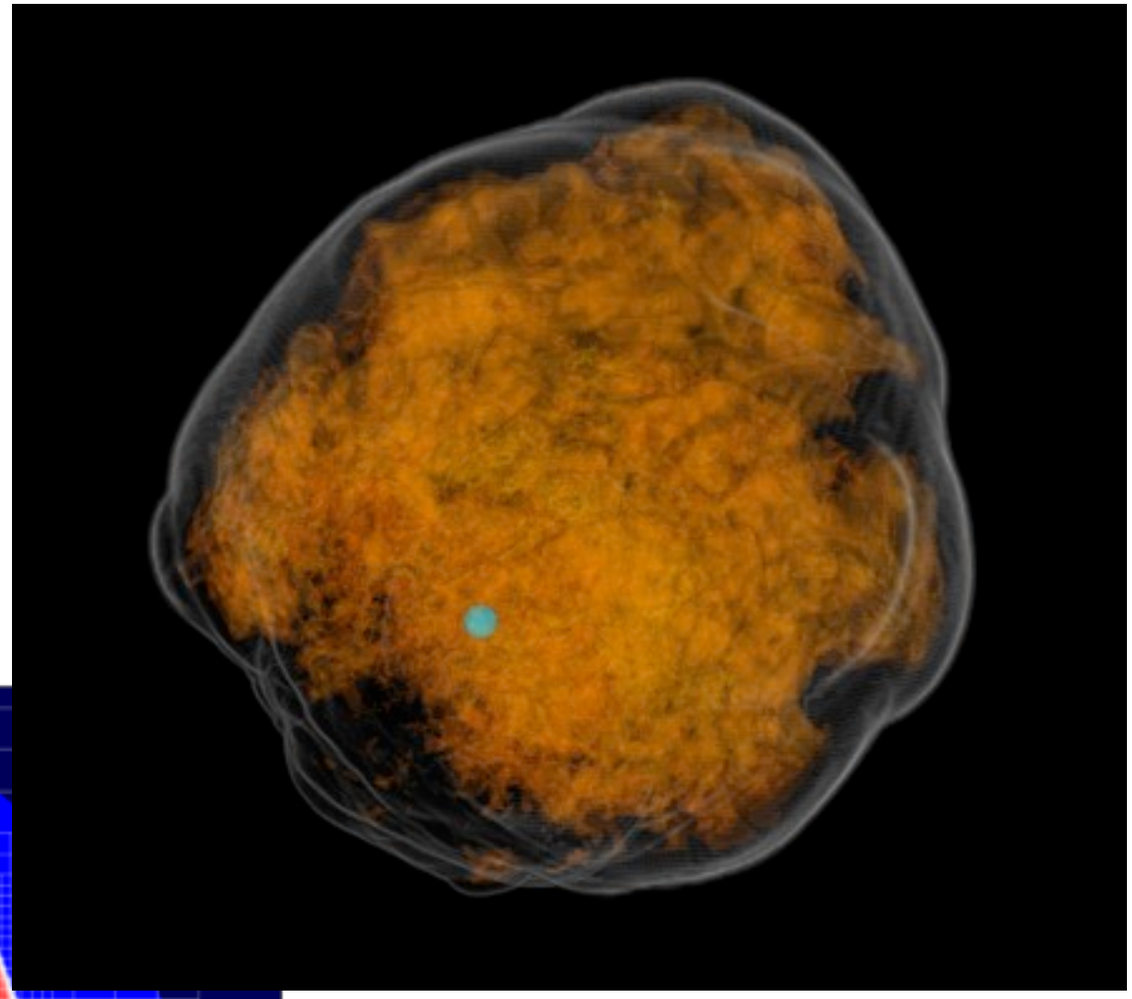
FLASH Code

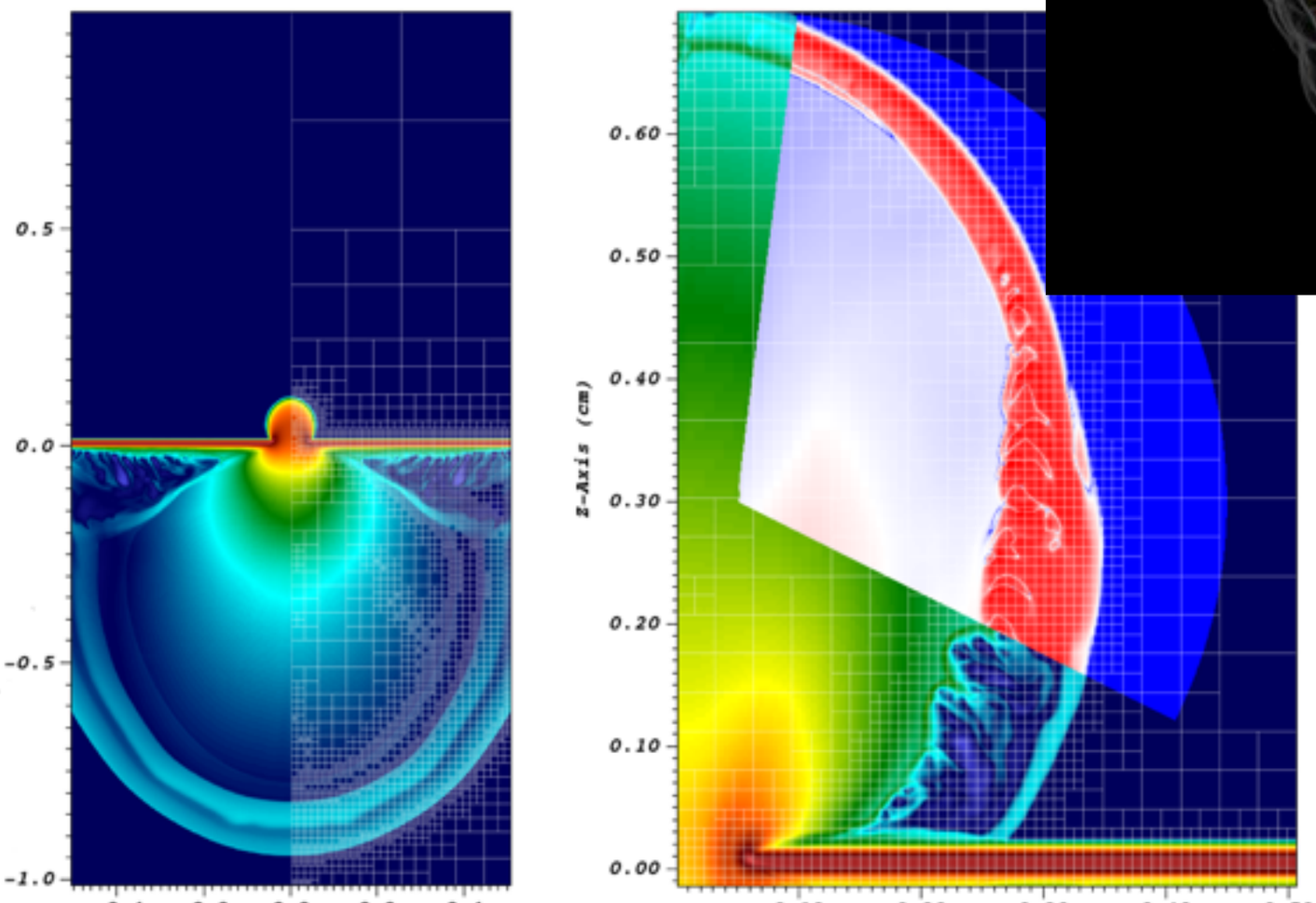
Some background information:

- Research applications
  - thermonuclear flashes
  - high energy density physics (HEDP)
  - fluid-structure interaction
  - star formation
  - star-star & star-planets interactions (CE phase)
  - cosmology
  - galaxy & galaxy cluster simulations
  - magnetic field amplification
  - turbulence
  - ...

FLASH Code


Reaction Progress Variable
0.00
0.250
0.500
0.750
1.00
Density




0.5
0.0
-0.5
-1.0
-0.4
-0.2
0.0
0.2
0.4

0.60
0.50
0.40
0.30
0.20
0.10
0.00
z-Axis (cm)
0.10
0.20
0.30
0.40
0.50

FLASH Code

Some background information:

- current release 4.0

  $\Longrightarrow$ user manual ~ 500 pages!

- a bit of history:

  - 1997: founding of the ASCI (Accelerated Strategy Computing Initiative*) FLASH Center
    $\Longrightarrow$ main goal: thermonuclear flashes
    at neutron star and white dwarf surfaces

*now ASC: Advanced Simulation and Computing

FLASH Code

- a bit of history:

  - 2000: first FLASH Code release

    $\Longrightarrow$ Fryxell, Olson, Ricker, Timmes, Zingale, Lamb, MacNeice, Rosner, Truran, Tufo, *FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes*, ApJS

    $\Longrightarrow$ HD based on PROMETHEUS Code (PPM)

    (Fryxell, Müller & Arnett, 1989, MPI for Astrophysics)

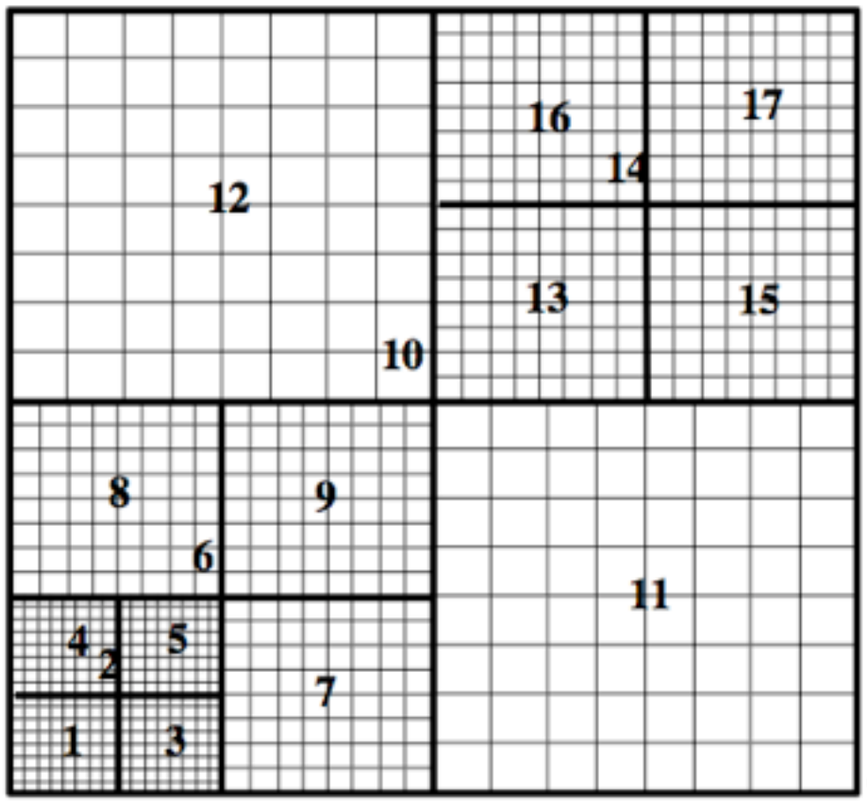
    $\Longrightarrow$ no MHD

    $\Longrightarrow$ no self-gravity

    $\Longrightarrow$ based on PARAMESH AMR library (block structured)

    Berger 1982
    Berger & Oliger 1984
    Berger & Colella 1989

    $\Longrightarrow$ supported grid geometry: cartesian, cylindrical, spherical



FLASH Code

- a bit of history:

  - 2002: major revised version 2.3

    $\Longrightarrow$ MHD solver (*Powell, Roe, Linde, Gombosi, De Zeeuw,* 1999)

    directional split, finite volume scheme

    +non-ideal terms

    reduction of $\nabla \mathbf{B}$ errors by

    - truncation error method (*Powell et al.* 1999)

      similar to *Dedner et al.* 2002 hyperbolic cleaning

    - elliptic projection method (*Brackbill & Barnes* 1980)

  $\Longrightarrow$ self-gravity / Poisson solver (*P. Ricker*)

  $\Longrightarrow$ tracer particle

  $\Longrightarrow$ "dark matter" particle

  $\Longrightarrow$ cosmology

  $\Longrightarrow$ HDF5 output format

FLASH Code

- a bit of history:

  - 2005: improved, long-lived version 2.5
    new external developments:
    $\Longrightarrow$ ray-trace based radiation-transfer for point sources
    (*Rijkhorst, Plewa, Dubey, Mellema, A&A* 2006, *Peters, RB, Klessen, Mac Low* 2010)
    hybrid-characteristics

$\Longrightarrow$ sink particles (*Federrath, RB, Clark, Klessen*, ApJ 2010)
- on the fly creation
- mass accretion
- interface for sub-grid feedback models (e.g. jets, SN, ...)

$\Longrightarrow$ new MHD solvers (*Waagan*, *Federrath, Klingenberg*, JCP 2011)
- "Bouchut 3" / "Bouchut 5" (*Bouchut* 2004, *Bouchut et al.* 2007/2010)
- ensures positivity of density/pressure

FLASH Code

- a bit of history:

  - 2008: completely restructured version 3
    $\Longrightarrow$ unsplit HD & MHD solver (*Donkwook Lee*)
    $\Longrightarrow$ support of more flexible grid structures
    e.g. uniform grid, PARAMESH 3(staggered grid),
    $\Longrightarrow$ different "database" calls
    $\Longrightarrow$ different directory structure
    $\Longrightarrow$ different structure of HDF5 output

  - 2011:Version 4 (*Fryxell et al.* 2010, *Lee* JCP 2013)
    $\Longrightarrow$ constraint transport (CT) MHD on a staggered mesh
    $\Longrightarrow$ many new physical modules:
    - diffusion solver
    - energy deposition via LASER beam

FLASH code: Overview

- extensive user manual: **`flash4_ug.pdf`**

with hyperlinks to
Flash center web site

3.4 Creating a `Simulation_init.F90`

The routine `Simulation_init` is called by the routine `Driver_initFlash` at the beginning of the simulation. `Driver_initFlash` calls Unit_init.F90 routines of every unit to initialize them. In this particular case, the `Simulation_init` routine will get the necessary runtime parameters and store them in the `Simulation_data` Fortran module, and also initialize other variables in the module. More generally, all one-time initialization required by the simulation are implemented in the `Simulation_init` routine.

FLASH Transition

In FLASH2, the contents of the `if (.firstcall.)` clause are now in the Simulation_init routine in FLASH4.

The basic structure of the routine `Simulation_init` should consist of

FLASH code: Overview

- many example setups

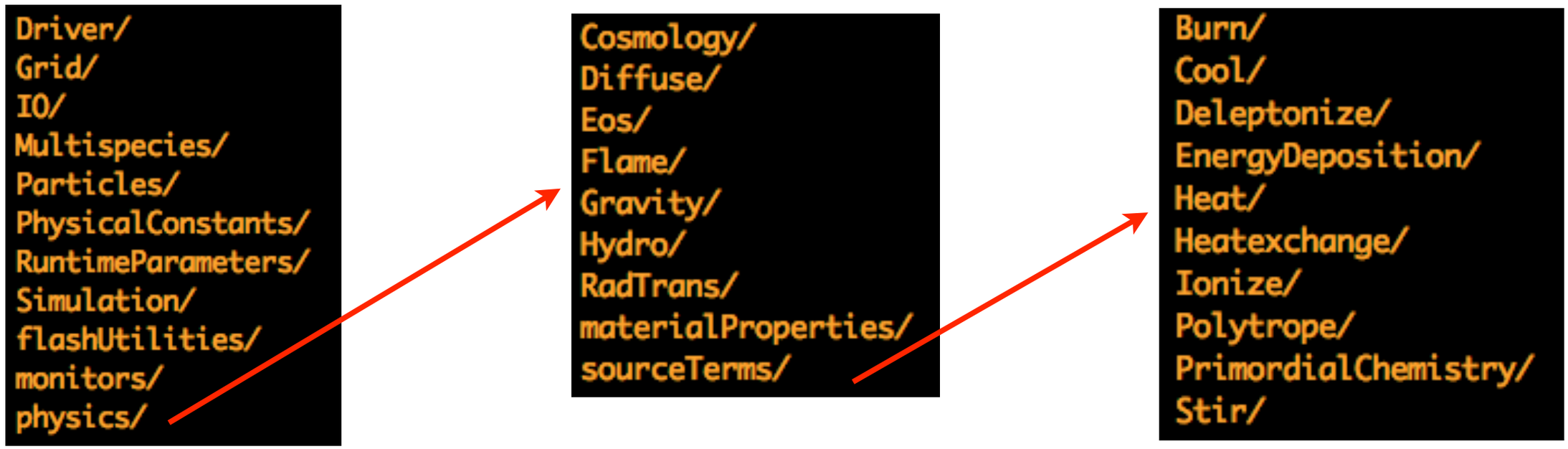
FLASH code: Overview

- many example setups

....

FLASH code: Overview

- most of the code: FORTRAN 90
- highly modular
  $\Longrightarrow$ fairly well organized source directory structure



- pre-compilation via **`setup`** python script
  $\Longrightarrow$ typical number of Fortran files : 900 (out of 3400)
  $\Longrightarrow$ typical number of code lines : 150,000

FLASH code: Overview

- hardware / computer specific makefiles in `sites` directory:

```
Aliases
Prototypes/
SEAS10927.gwu.edu/
alc.llnl.gov/
animal5/
archimedes.uchicago.edu/
bassi.nersc.gov/
bgl.llnl.gov/
bgl.mcs.anl.gov/
bgl.sdsc.edu/
bonsai.cfa.harvard.edu/
brassica.asci.uchicago.edu/
```

computer alias names

prototype makefiles:
linux, Mac OS, ...

...

```
hyades.ucsc.edu/
hydra.si.edu/
icc-9.0_fornax.uchicago.edu/
ignition/
```

FLASH code: Overview

- requirements
  - fortran compiler
  - MPI
  - HDF5
  - PARAMESH AMR library (delivered with FLASH source)
  - HYPRE (for implicit solvers, e.g. diffusion)

- output format
  - HDF5
  - NetCDF
  - some ASCII files
    log files, dat-files with integrated variables, ...

FLASH Code

The grid structure:

- possible grid structures

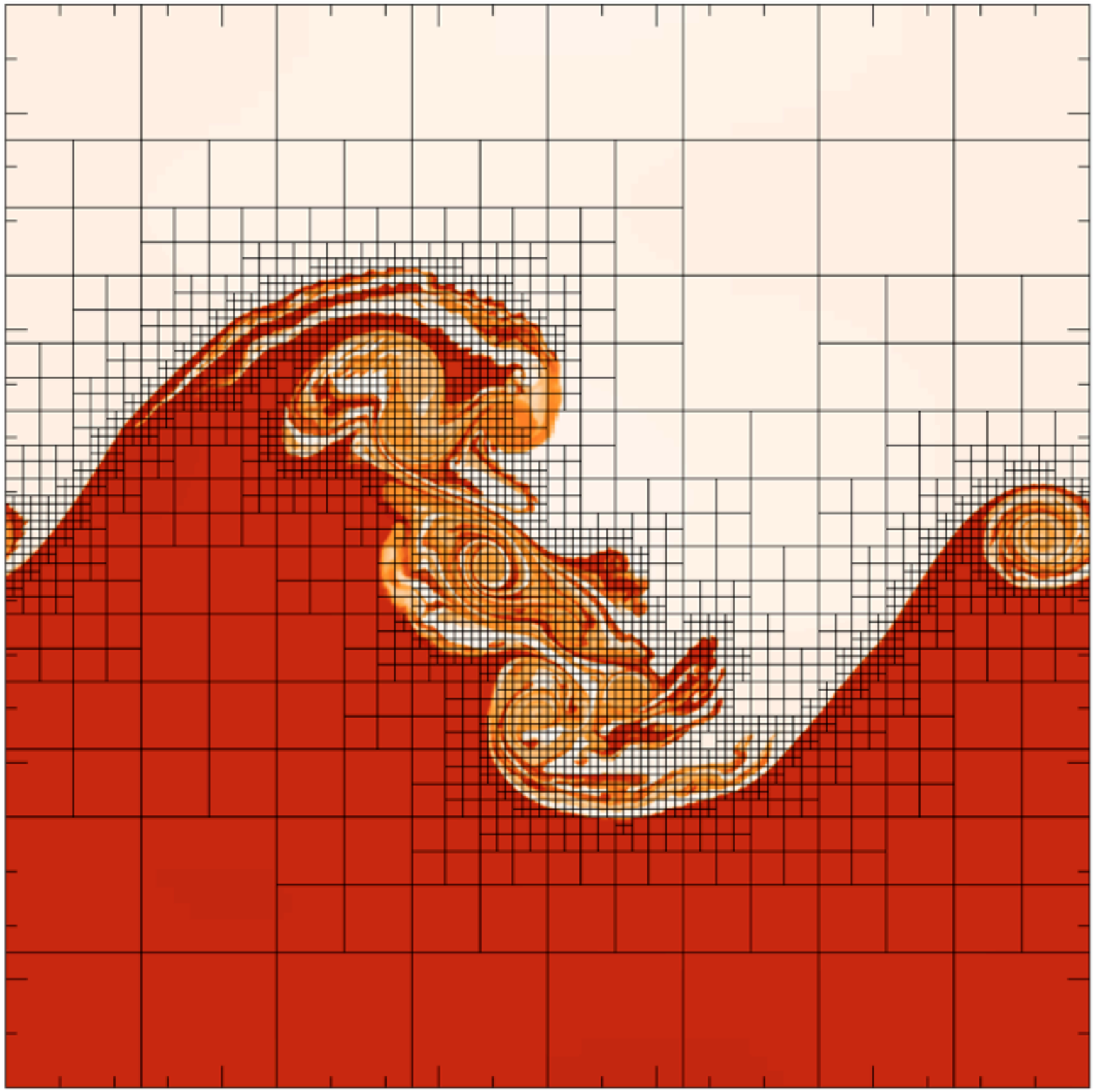
  - PARAMESH AMR (*MacNeice et al.* CPC 2000; *Olson* 2006)
    - block structured AMR
    - current version 4.
      `http://www.physics.drexel.edu/~olson/paramesh-doc/Users_manual/amr.html`

  - CHOMBO
    patch based AMR, still under development

- uniform grid
  no AMR overhead

FLASH Code

Block structured PARAMESH AMR:

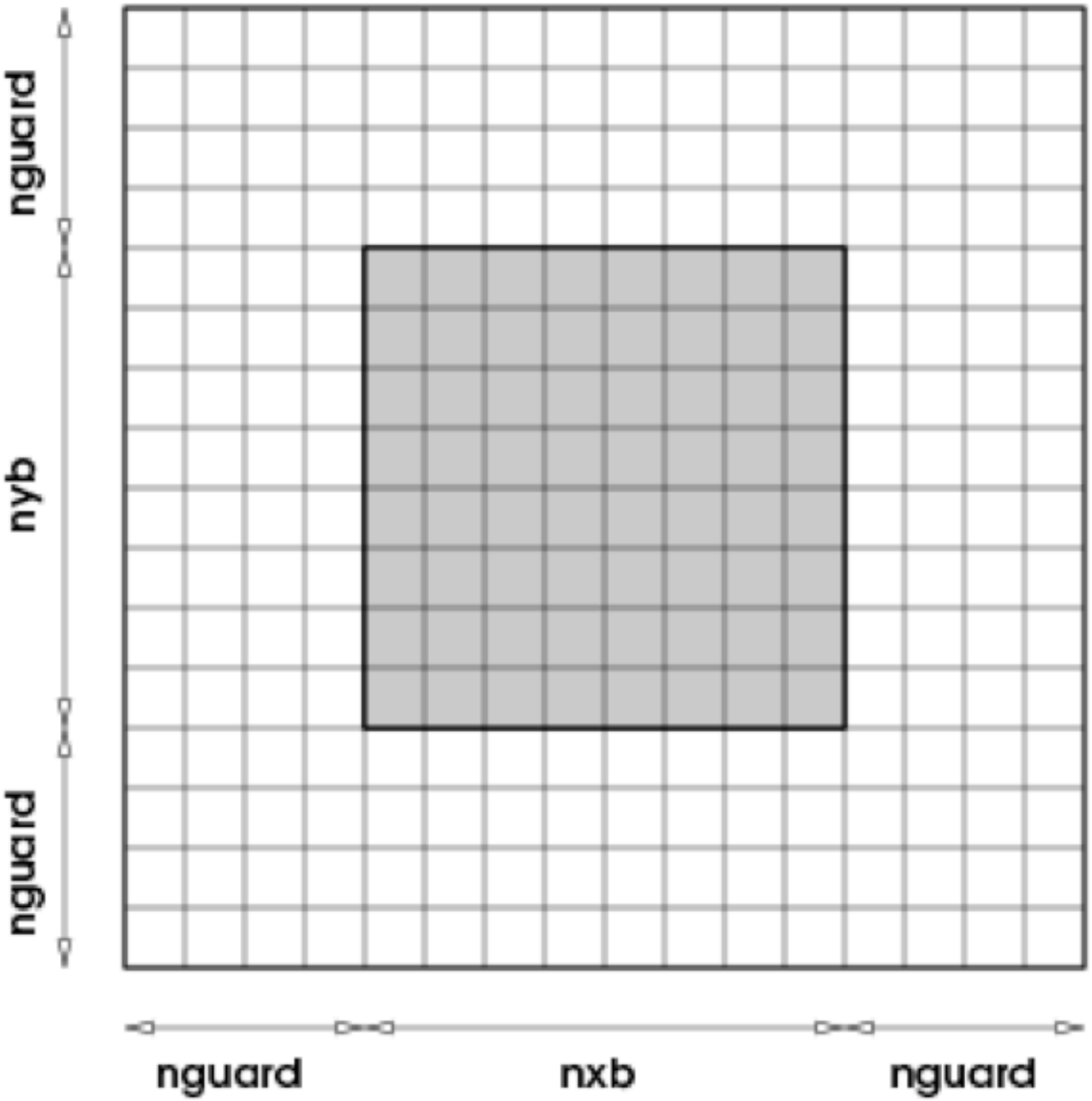


- neighbouring blocks: max relative size: 2
- typically blocks of 8^D active grid cells
- effective resolution: $2^{l\max+2}$

$(l_{\min} = 1)$

FLASH Code

Block structure:



- typically blocks of 8^D active grid cells

  `NBX, NBY, NBZ`

  $\Longrightarrow$ setup parameter

- + 2 to 4 ghost / guard cells:

  `nguard`

  dependent on order of the integration scheme:
  PPM: `nguard = 4`
  8W MHD: `nguard = 2`

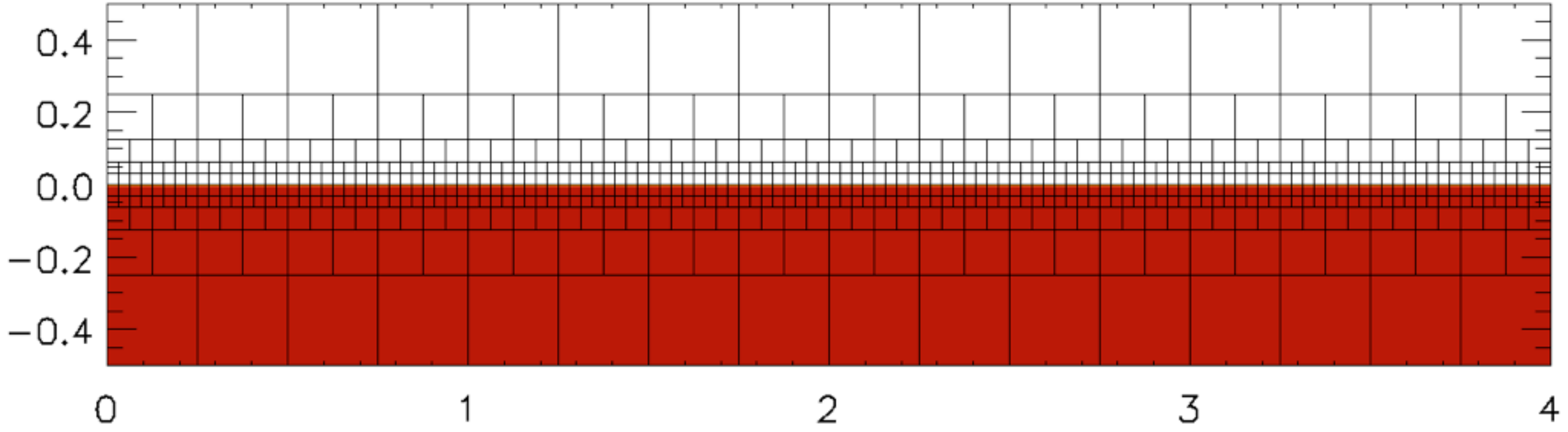
FLASH Code

Block structured PARAMESH AMR:

- setup up of non-cubic simulation box with cubic blocks:

  $\Longrightarrow$ `nblockx`, `nblocky`, `nblockz`

  = number of level-1-blocks in x,y,z direction



Example: `nblockx = 4, nblocky = 1`

`xmin = 0, xmax = 4, ymin = −0.5, ymax = 0.5`

FLASH Code

AMR: refinement / de-refinement

- standard criterion: second derivative criterion
  $\Longrightarrow$ "shock" capturing criterion
  $\Longrightarrow$ test with dimensionless error: $|\mathrm{d}^2 f/\mathrm{d}x^2| \;/\; |f\,|/(\mathrm{d}x)^2$

- refine on user defined variables (4 possible)

```
refine_var_1 = "dens"
…
```

set in `flash.par`

- parameter:

```
refine_cutoff_1     = 0.8
derefine_cutoff_1   = 0.2
```

FLASH Code

AMR: refinement / de-refinement

- further criterions
  - Jeans criterion
  - refine on particle position

  in

  `Particles/ParticlesMain/active/Sink/Particles_sinkMarkRefineDerefine.F90`

- implementing own criterion:

  copy `gr_markRefineDerefine.F90`
  in own setup directory

  $\Longrightarrow$ overrides standard routine

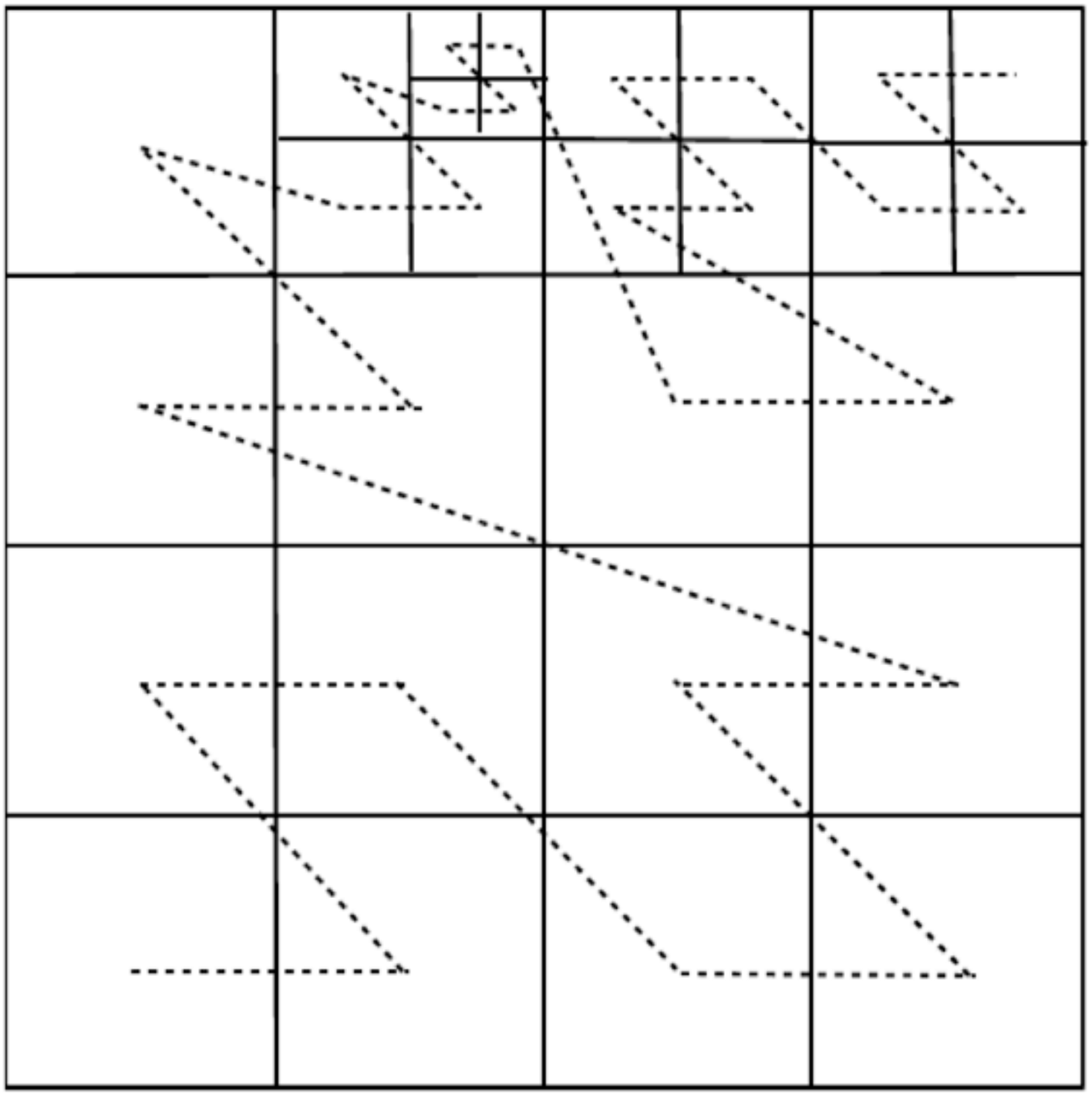
FLASH Code

simulation box boundary types

*ab*_boundary_type	Description
`periodic`	Periodic ('wrap-around')
`reflect`	Non-penetrating boundaries; plane symmetry, the normal vector components change sign
`outflow`	Zero-gradient boundary conditions; allows shocks to leave the domain
`diode`	like outflow, but fluid velocities are never allowed to let matter flow into the domain: normal velocity components are forced to zero in guard cells if necessary
`axisymmetric`	like reflect, but both normal and toroidal vector components change sign. Typically used with cylindrical geometry (R-Z) for the Z symmetry axis.
`eqtsymmetric`	like reflect for velocities but the magnetic field components, poloidal and toroidal, change sign. The sign of the normal magnetic field component remains the same. Typically used with cylindrical geometry (R-Z) for the R axis to emulate equatorial symmetry.
`hydrostatic-f2`	Hydrostatic boundary handling as in FLASH2. See remark in text.
`hydrostatic-f2+nvrefl`, `hydrostatic-f2+nvout`, `hydrostatic-f2+nvdiode`	Variants of **`hydrostatic-f2`**, where the **n**ormal **v**elocity is handled specially in various ways, analogous to **`reflect`**, **`outflow`**, and **`diode`** boundary conditions, respectively. See remark in text.
`user-defined` or `user`	The user must implement the desired boundary behavior; see text.

FLASH Code

Parallelisation / Grid-decompositon



- Morton space-filling Z-order decomposition

$\Longrightarrow$ results in good load balance

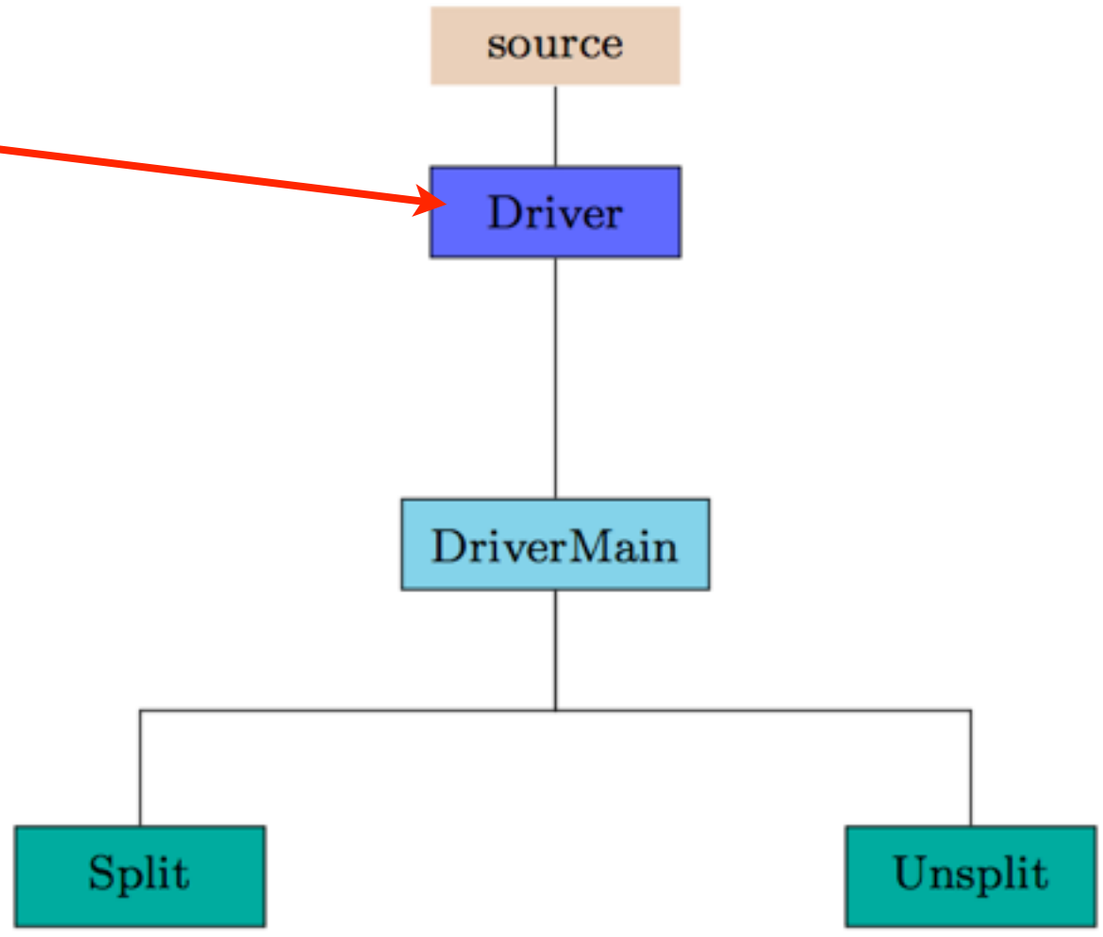
- Note:
  use of **global** timestep for **all** blocks/ refinement levels
  $\Longrightarrow$ but super-time-stepping (STS) for parabolic equations possible

FLASH Code

The directory structure:

`FLASH4.0/source`

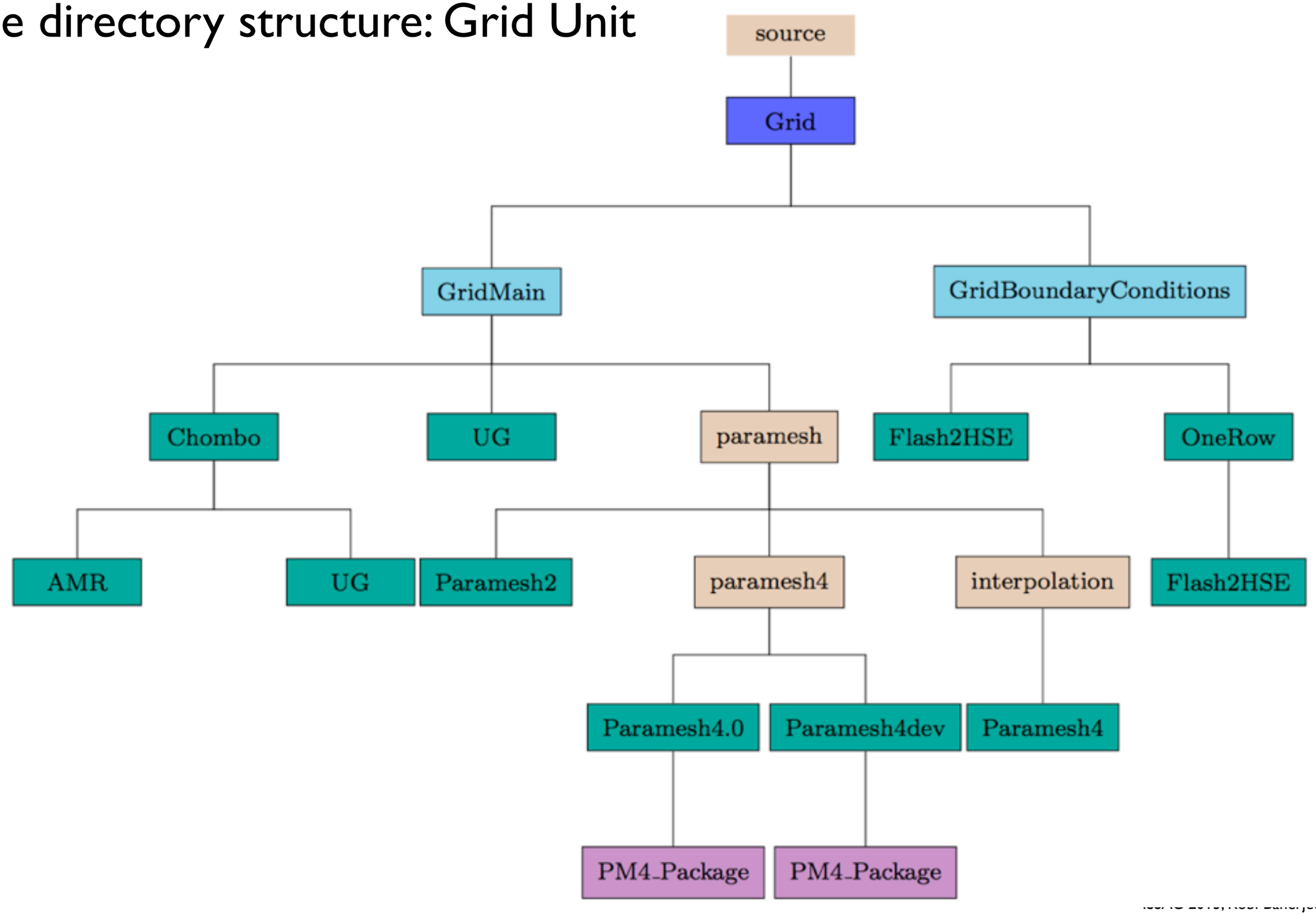
```
Driver/
Grid/
IO/
Multispecies/
Particles/
PhysicalConstants/
RuntimeParameters/
Simulation/
flashUtilities/
monitors/
physics/
```



e.g. `Driver_evolFlash.F90` : contains main evolution loop

FLASH Code

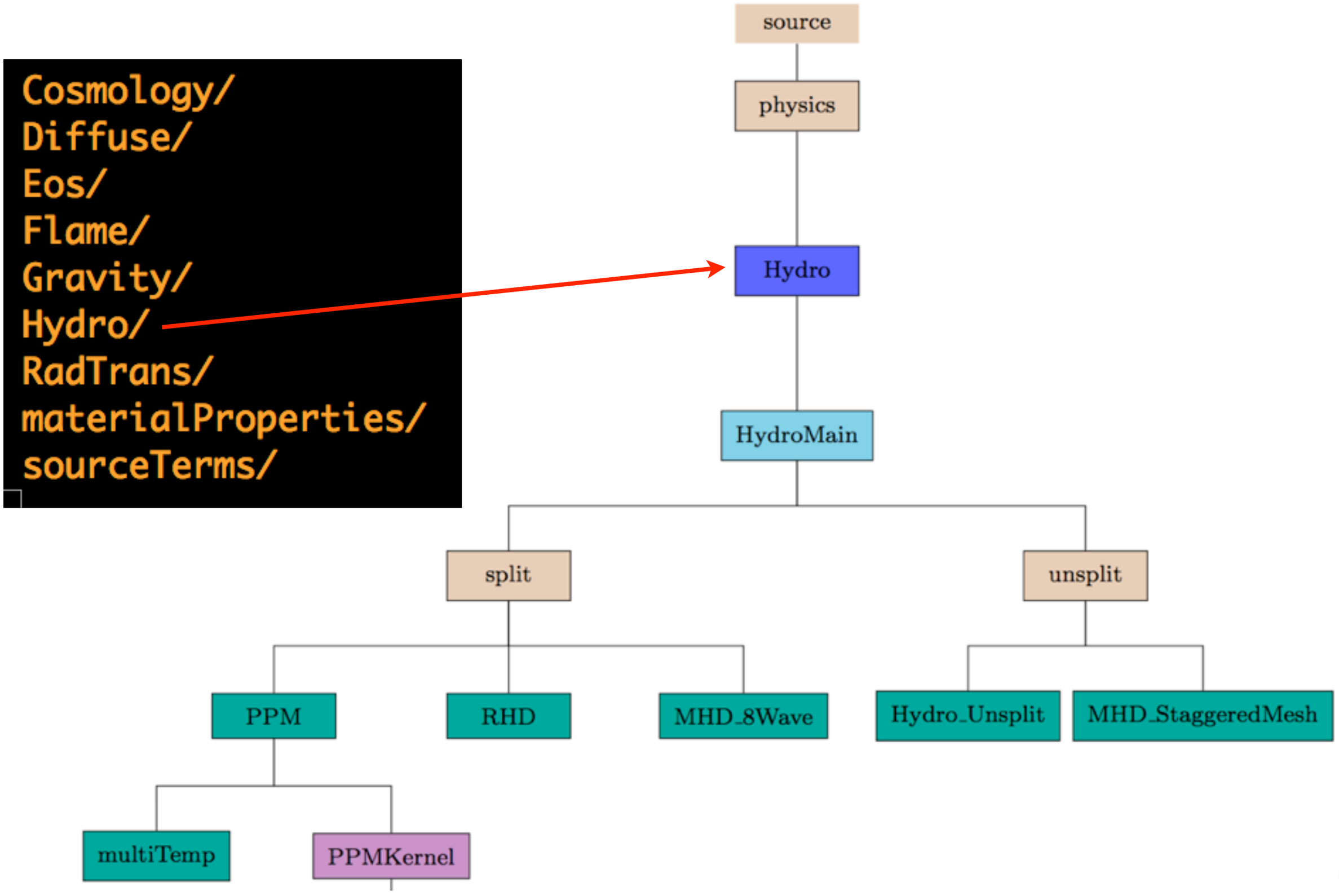
The directory structure: Grid Unit



FLASH Code

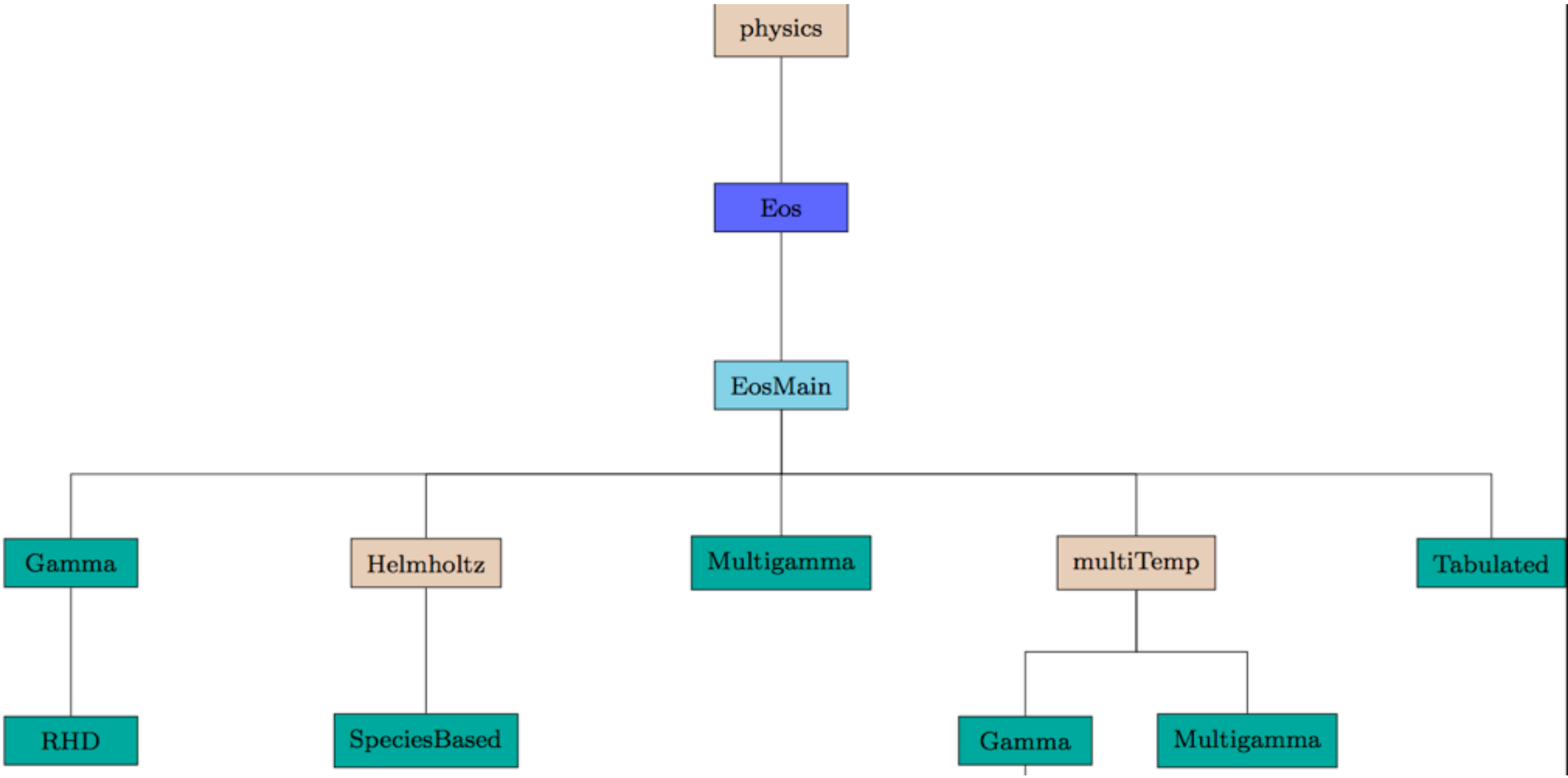
The directory structure: Physics Unit

```
Cosmology/
Diffuse/
Eos/
Flame/
Gravity/
Hydro/
RadTrans/
materialProperties/
sourceTerms/
```



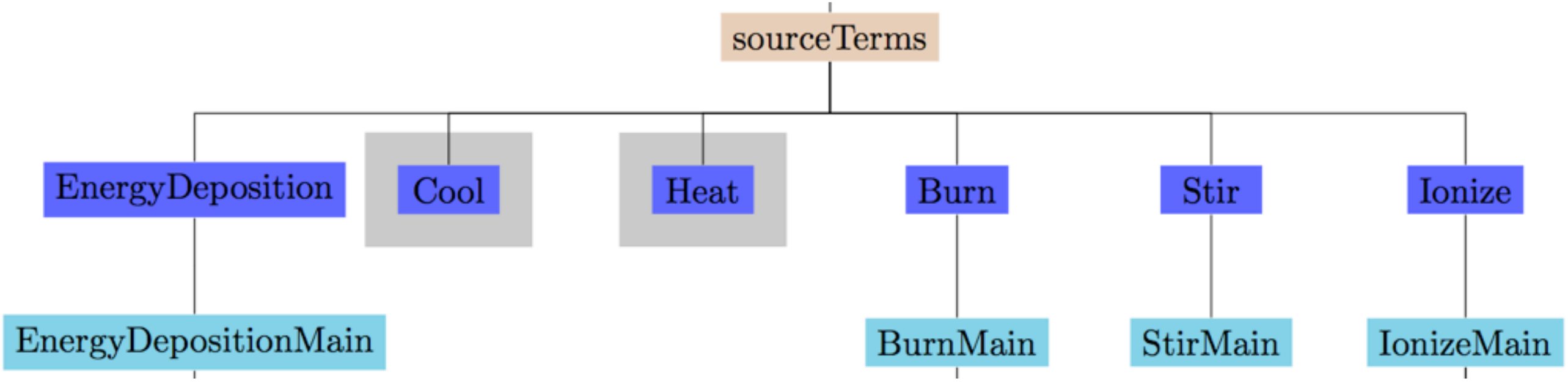
FLASH Code

The directory structure: EOS Unit

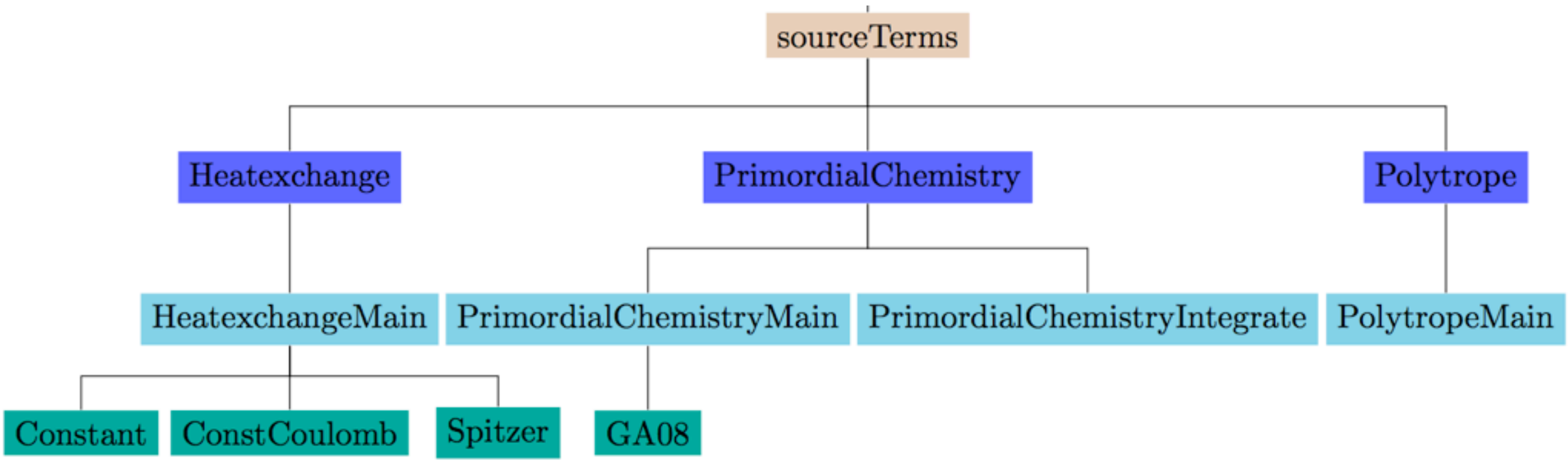


FLASH Code

The directory structure: source terms

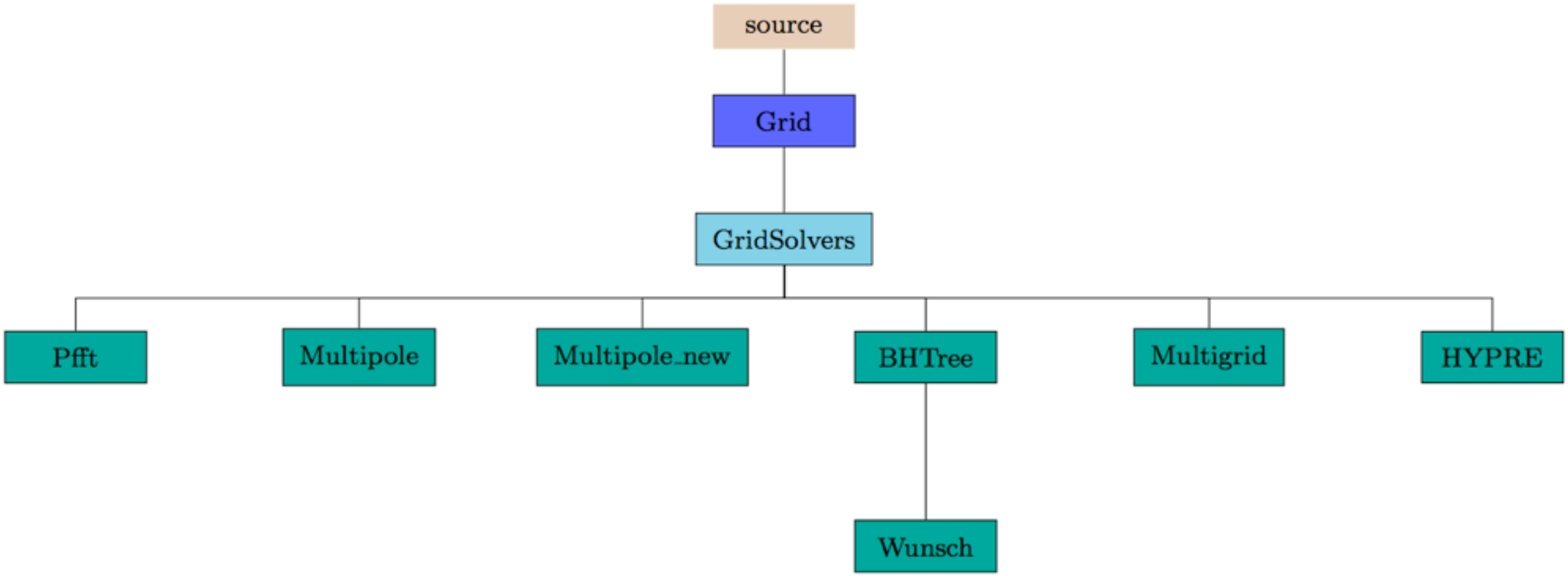


The directory structure: chemistry



FLASH Code

The directory structure: Solvers



FLASH Code

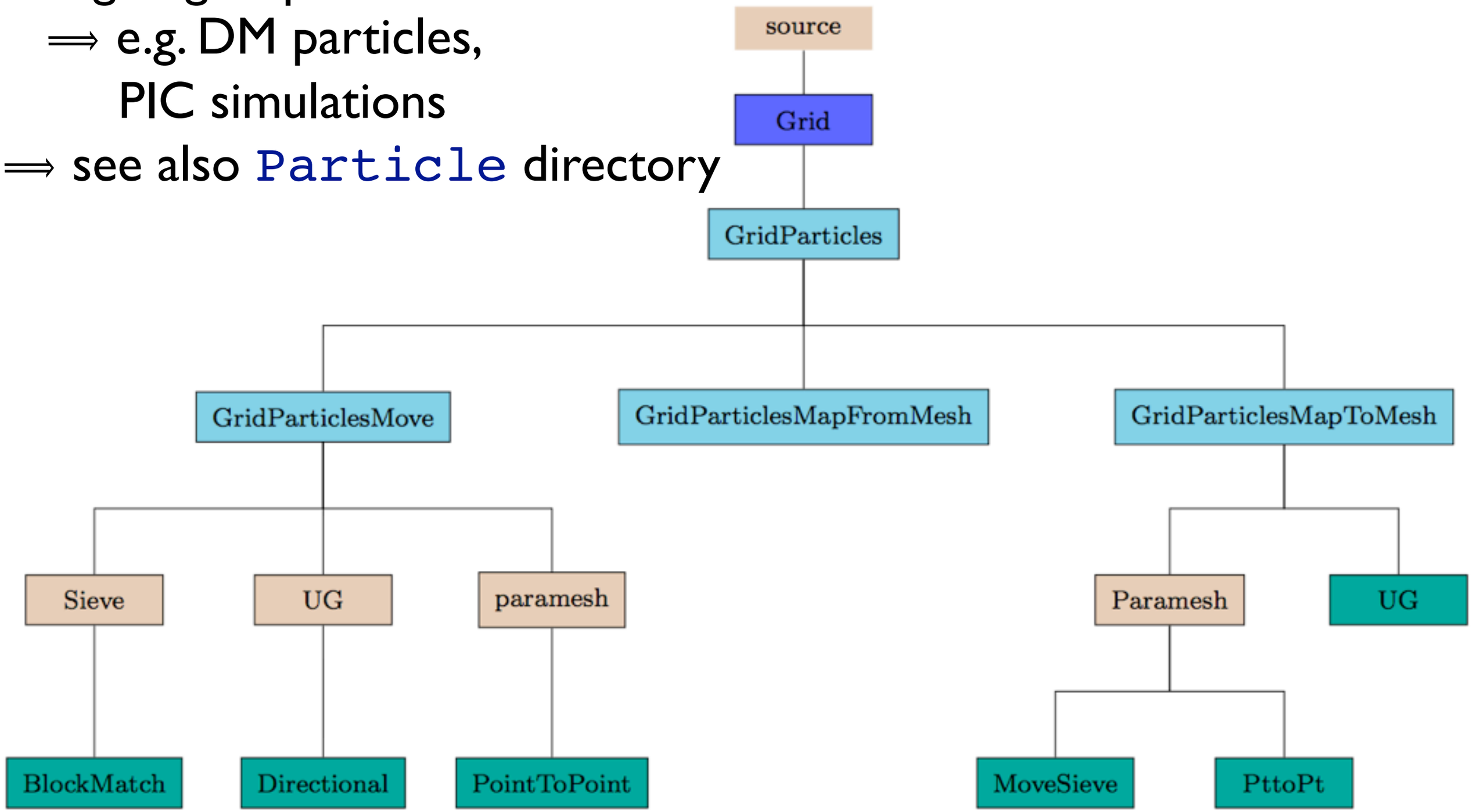
The directory structure:

Lagrangian particles

$\Longrightarrow$ e.g. DM particles,

PIC simulations

$\Longrightarrow$ see also `Particle` directory



FLASH Code

```
usage:  setup <problem-name> [options] [VAR=VALUE]...

   problem-name: see source/Simulation/SimulationMain directory
   options:

   (Science Options)
            -auto -[123]d
            -maxblocks=<#> -nxb=<#> -nyb=<#> -nzb=<#>
             -with-unit=<unit> -with-library=<libname>[,args]
             -without-unit=<unit> -without-library=<libname>

  (Setup and Make Options)
             -verbose=[DEBUG|INFO|WARN|IMPINFO|ERROR]
             [-site=<site> | -ostype=<ostype>]
             -makefile=<extension>
             [-opt| -debug | -test ]
             -objdir=<relative obj directory>
             -defines=<defines> -unitsfile=<filename>
             -datafiles=<wildcard> -parfile=<filename>
             -fbs -nofbs -tau=<makefile>
```

FLASH Code

shortcuts for often used setup options

Shortcut	Description
+cartesian	use cartesian geometry
+cylindrical	use cylindrical geometry
+noio	omit IO
+nolog	omit logging
+pm2	use the PARAMESH2 grid
+pm40	use the PARAMESH4.0 grid
+pm4dev	use the PARAMESH4DEV grid
+polar	use polar geometry
+spherical	use spherical geometry
+ug	use the uniform grid in a fixed block size mode
+nofbs	use the uniform grid in a non-fixed block size mode
+usm	use the Unsplit Staggered Mesh MHD solver
+8wave	use the 8-wave MHD solver
+unsplitHydro	use the Unsplit Hydro solver

FLASH Code

Getting started: `run flash4`

- copy `flash4` and `flash.par` to work directory
- single core run:

```
./flash4 [-par_file parameter-filename]
```

- parallel run:
  use queuing system: `qsub`, `qstat`, `qdel`, ...

  with q-script

  ....

```
mpirun -n 32 -env I_MPI_FABRICS shm:ofa ./flash4
```

FLASH Code

Visualisation of simulation results:

- IDL tools:
  `tools/fidlr3.0` ⟹ `xflash3` GUI
  ⟹ `setenv XFLASH3_DIR flash4.0.1/tools/fidlr3.0`
  ⟹ IDL-path:

```
!path = !path + ':' + getenv('XFLASH3_DIR')
```

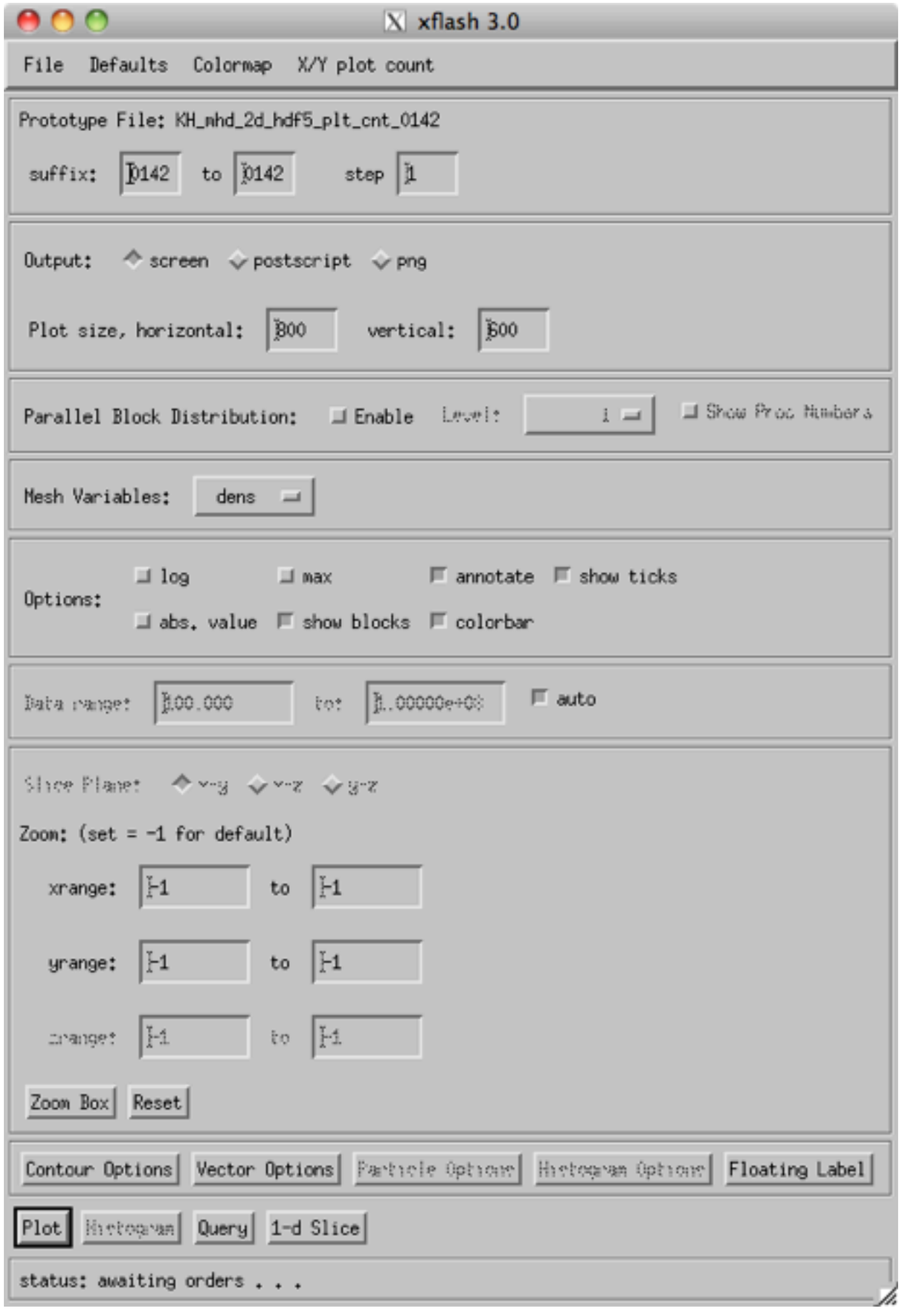
- VisIt
  `https://wci.llnl.gov/codes/visit`

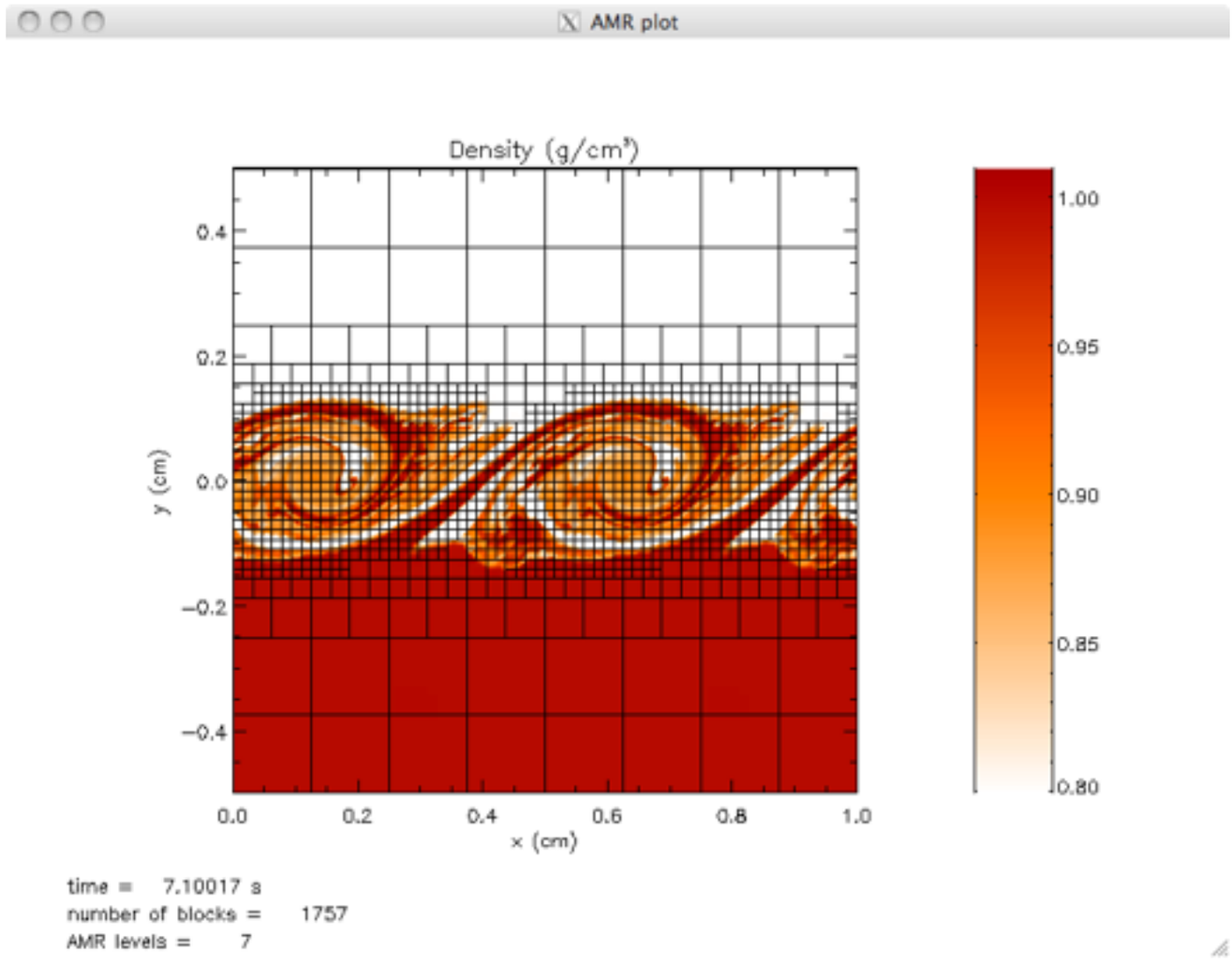
- yt: python based scripts
  http://yt-project.org

⟹ all programmes available on hyades via `module load` command

FLASH Code

`xflash3` IDL GUI





FLASH Code

A typical problem setup

- the files in
  `FLASH4.0.1/source/Simulation/SimulationMain/PROBLEM`

define parameters specific to this problem

specific compiler dependencies

declare specific runtime parameters

```
 637 Jul 24 18:33 Config
 114 May 27 06:36 Makefile
 743 Jul 24 18:34 Simulation_data.F90
1655 Jul 24 18:34 Simulation_init.F90
6442 Jul 24 18:37 Simulation_initBlock.F90
1553 May 27 06:36 flash.par
```

read in and initialise runtime parameters

set initial grid variables (e.g. density, velocity, ...)

default runtime parameter file for this setup

FLASH Code

A typical problem setup: **`Simulation_data.F90`**

```
module Simulation_data

  implicit none

#include "constants.h"

  !! *** Runtime Parameters *** !!
  real, save    :: sim_gamma,sim_smallX, sim_smallRho, sim_smallP
  real, save    :: sim_xMin, sim_xMax, sim_yMin, sim_yMax, sim_zMin, sim_zMax
  real, save    :: sim_perturbation
  logical, save :: sim_gCell, sim_killdivb

  integer, save :: sim_meshMe
end module Simulation_data
```

`module`:
variables can be
accessed within other
subroutines

FLASH Code

A typical problem setup: **`Simulation_init.F90`**

```
  call Driver_getMype(MESH_COMM, sim_meshMe)

  call RuntimeParameters_get('gamma',    sim_gamma)
  call RuntimeParameters_get('xmin',     sim_xMin)
  call RuntimeParameters_get('ymin',     sim_yMin)
  call RuntimeParameters_get('zmin',     sim_zMin)
  call RuntimeParameters_get('xmax',     sim_xMax)
  call RuntimeParameters_get('ymax',     sim_yMax)
  call RuntimeParameters_get('zmax',     sim_zMax)
  call RuntimeParameters_get('killdivb',sim_killdivb)
  call RuntimeParameters_get('smallp',   sim_smallP)
  call RuntimeParameters_get('smallx',   sim_smallX)

  !! for 3D velocity perturbation
  call RuntimeParameters_get('perturbation',  sim_perturbation)
  sim_gCell = .true.

end subroutine Simulation_init
```

FLASH Code

A typical problem setup: the **`Config`** file

```
#       Configuration file for Orszag Tang MHD vortex problem
#       (Orszag and Tang, J. Fluid Mech., 90:129--143, 1979)

REQUIRES physics/Hydro/HydroMain
REQUIRES physics/Eos/EosMain/Gamma

USESETUPVARS withParticles

IF withParticles
   PARTICLETYPE passive INITMETHOD lattice MAPMETHOD quadratic ADVMETHOD rungekutta

   REQUIRES Particles/ParticlesMain
   REQUESTS IO/IOMain
   REQUESTS IO/IOParticles
   REQUESTS Particles/ParticlesMapping/Quadratic
   REQUESTS Particles/ParticlesInitialization/Lattice
ENDIF

D   tiny            Threshold value used for numerical zero
PARAMETER tiny          REAL     1.e-16
PARAMETER perturbation REAL     0.2
```

can be used at `setup` command

required modules

specific runtime parameters with default values

FLASH Code

A typical problem setup: the **`flash.par`** file

```
#        Size of computational volume
xmin                = 0.
xmax                = 1.
ymin                = -0.5
ymax                = 0.5

#        Boundary conditions
xl_boundary_type = "periodic"
xr_boundary_type = "periodic"
yl_boundary_type = "reflect"
yr_boundary_type = "reflect"

#        Simulation (grid, time, I/O) parameters
run_comment      = "KH MHD problem"
log_file         = "KH_mhd_2d.log"
basenm           = "KH_mhd_2d_"
restart          = .false.
nend             = 10000000
tmax             = 10
cfl              = 0.8
plot_var_1       = "dens"

#perturbation
perturbation     = 0.0125
```

FLASH Code

A typical problem setup: **`Simulation_initBlock.F90`**

```
subroutine Simulation_initBlock(blockID)

  use Simulation_data, ONLY : sim_gCell, sim_gamma,   &
                              sim_smallX, sim_smallP, &
                              sim_killdivb, sim_perturbation

  use Grid_interface, ONLY : Grid_getBlkIndexLimits, &
                             Grid_getCellCoords,     &
                             Grid_getBlkPtr,         &
                             Grid_releaseBlkPtr


  implicit none

#include "constants.h"
#include "Flash.h"

  !!$ Arguments -----------------------
  integer, intent(in) :: blockID
  !!$ ---------------------------------

  integer :: i, j, k, n, istat, sizeX, sizeY, sizeZ

  real,allocatable,dimension(:) :: xCoord,yCoord,zCoord
  integer, dimension(2,MDIM) :: blkLimits, blkLimitsGC
  real :: enerZone, ekinZone, eintZone
  real, pointer, dimension(:,:,:,:) :: solnData, facexData,faceyData,facezData


  call Grid_getBlkIndexLimits(blockID,blkLimits,blkLimitsGC)

  sizeX = blkLimitsGC(HIGH,IAXIS)-blkLimitsGC(LOW,IAXIS)+1
  sizeY = blkLimitsGC(HIGH,JAXIS)-blkLimitsGC(LOW,JAXIS)+1
  sizeZ = blkLimitsGC(HIGH,KAXIS)-blkLimitsGC(LOW,KAXIS)+1

  allocate(xCoord(sizeX),stat=istat)
  allocate(yCoord(sizeY),stat=istat)
  allocate(zCoord(sizeZ),stat=istat)

  xCoord = 0.0
  yCoord = 0.0
  zCoord = 0.0
```

from
`Simulation_data.F90`
`Simulation_init.F90`

FLASH Code

A typical problem setup: set the hydro variables

`Simulation_initBlock.F90`

```
call Grid_getBlkPtr(blockID,solnData,CENTER)
! Loop over cells in the block.
do k = blkLimitsGC(LOW,KAXIS),blkLimitsGC(HIGH,KAXIS)

   do j = blkLimitsGC(LOW,JAXIS),blkLimitsGC(HIGH,JAXIS)

      do i = blkLimitsGC(LOW,IAXIS),blkLimitsGC(HIGH,IAXIS)

         ! Multiple species
!solnData(SPECIES_BEGIN,i,j,k)=1.0e0-(NSPECIES-1)*sim_smallX
do n=SPECIES_BEGIN,SPECIES_END
   solnData(n,i,j,k)=sim_smallX
enddo

solnData(PRES_VAR,i,j,k)= sim_pres

if (yCoord(j) .gt. yMax2) then

            ! Cell-centered values
            solnData(DENS_VAR,i,j,k)= sim_rhoUp
            solnData(VELX_VAR,i,j,k)= sim_Vx0

            solnData(VELY_VAR,i,j,k)= sim_epsilon*sin(sim_modes*2d0*PI*xCoord(i))*exp( -100d0*
            solnData(VELZ_VAR,i,j,k)= 0.
            solnData(TEMP_VAR,i,j,k)= sim_rhoUp*sim_cs*sim_cs/sim_gamma*sim_rhoUp
```

access to hydro variables from memory ‘database’

FLASH Code

A typical problem setup: set the hydro variables

`Simulation_initBlock.F90`

```
! Compute the gas energy and set the gamma-values needed for the EOS
ekinZone = 0.5 * dot_product(solnData(VELX_VAR:VELZ_VAR,i,j,k),&
                             solnData(VELX_VAR:VELZ_VAR,i,j,k))

! specific internal energy
eintZone = solnData(PRES_VAR,i,j,k)/(sim_gamma-1.)/solnData(DENS_VAR,i,j,k)

! total specific gas energy
enerZone = eintZone + ekinZone

! Take a limit value
enerZone = max(enerZone, sim_smallP)

solnData(ENER_VAR,i,j,k)=enerZone
solnData(EINT_VAR,i,j,k)=eintZone
solnData(GAMC_VAR,i,j,k)=sim_gamma
solnData(GAME_VAR,i,j,k)=sim_gamma
```

make sure the total energy is set properly!

FLASH Code

setup your own problem

- create a new directory (e.g. `KH`) in
  `source/Simulation/SimulationMain/`
- copy setup files from similar setup (e.g. `OrszagTang`)
- modify setup files
  - `Config`
  - `Simulation_data.F90`
  - `Simulation_init.F90`
  - `Simulation_initBlock.F90`
  - `flash.par`

- setup and compile

```
./setup Sod -auto -2d -maxblocks=5000 -objdir=obj_KH
cd obj_KH
make
```

- run simulation using new `flash4`

FLASH Code

task

try

- Sod-Shock tube problem
- Orszag-Tang MHD vortex problem

- set up a Kelvin-Helmholtz instability problem