

# The Center for Astrophysical Thermonuclear Flashes

---

## Architecture

Flash Tutorial  
June 22, 2009  
Dr. Lynn B. Reid



An Advanced Simulation & Computing (ASC)  
Academic Strategic Alliances Program (ASAP) Center  
at The University of Chicago





# Architecture Outline

---

- Units
  - UnitMain
  - Subunits
  - Alternate implementations
- Naming Conventions
  - Files
  - Variables
- Inheritance
  - Stubs
- Setup script
  - Config files



# What's a FLASH Unit?

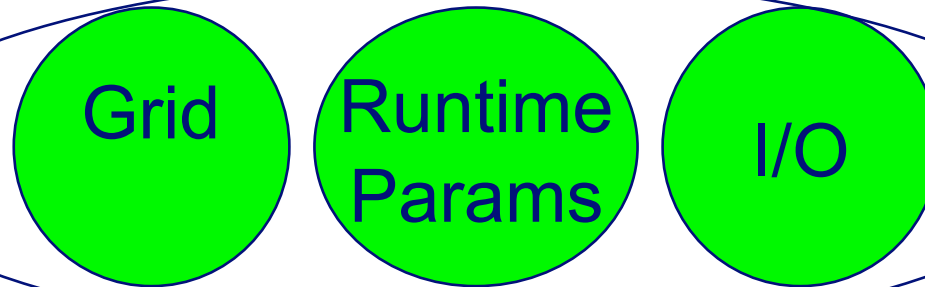
---

- ❑ FLASH basic architecture unit
  - ❑ Component of the FLASH code providing a particular functionality
  - ❑ Different combinations of units are used for particular problem setups
  - ❑ Publishes a public interface (API) for other units' use.
  - ❑ Ex: Driver, Grid, Hydro, IO etc
- ❑ Fake inheritance by use of directory structure
- ❑ Interaction between units governed by the Driver
- ❑ Not all units are included in all applications

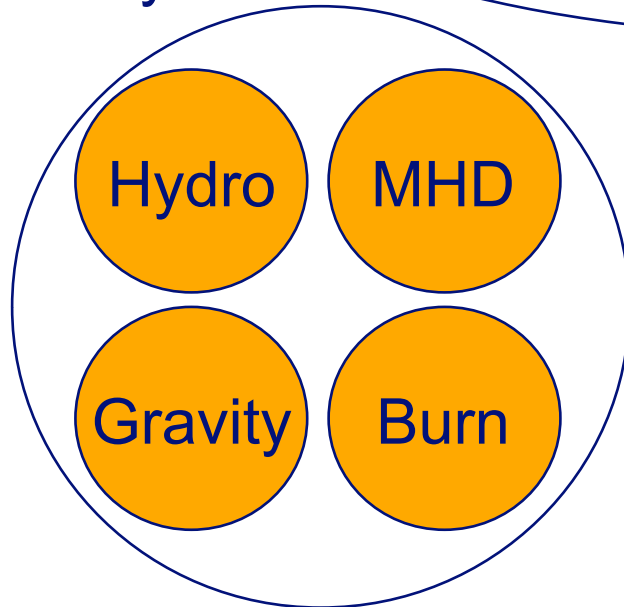


# FLASH Units: Examples

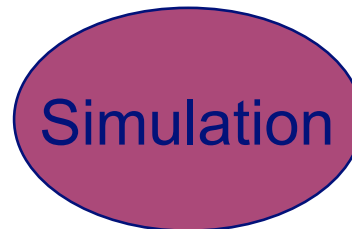
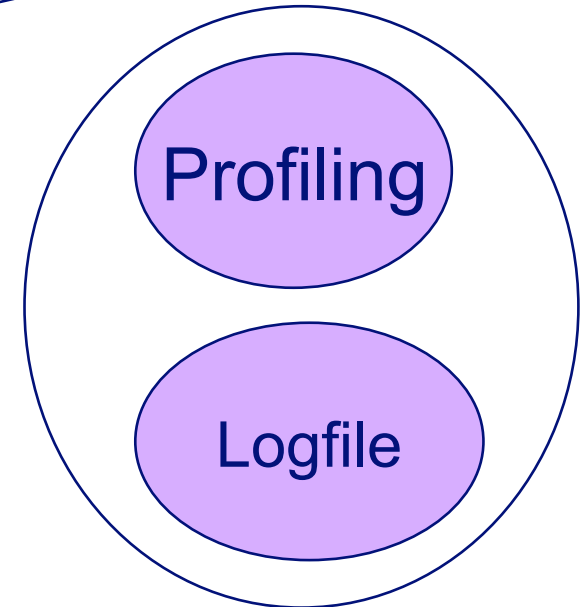
## Infrastructure



## Physics



## monitoring





## Inside a Unit: The Top Level

---

- ❑ First **capitalized** directory in a branch of the source tree is a unit
- ❑ Contains **stubs** for every public function (**API**) in the unit
  - ❑ Does not contain the data module (unit scope data)
  - ❑ Individual API functions may be implemented in different subunits
  - ❑ A unit has a minimum three functions in its API, no limit on the maximum
    - ❑ Unit\_init, Unit\_finalize and the “do-er” function for the unit
- ❑ If necessary, contains a directory for the **local API**
- ❑ May contain the **unit test**
  - ❑ Different Unit tests can reside at different levels in the unit hierarchy
- ❑ The Config file contains minimal information, no runtime parameters except “useUnit” defined
- ❑ **Makefile** includes all the API functions.



# Subunits

---

- ❑ Every unit has a **UnitMain** subunit, which must be included in the simulation if the unit is included.
  - ❑ Has implementations for the init, finalize and the main “do-er” function
  - ❑ Also contains the unit scope data module
- ❑ The API functions and private functions implemented in different subunits are **mutually exclusive**
- ❑ Subunits other than UnitMain may have private Unit scope functions that **can be called by other subunits**.
  - ❑ un\_sulnit and un\_suFinalize are the most common ones
  - ❑ (naming convention explained later)
- ❑ Subunits can also have **private data modules**, strictly within the scope limited to the specific subunit
- ❑ Subunits can have their own **unit tests**



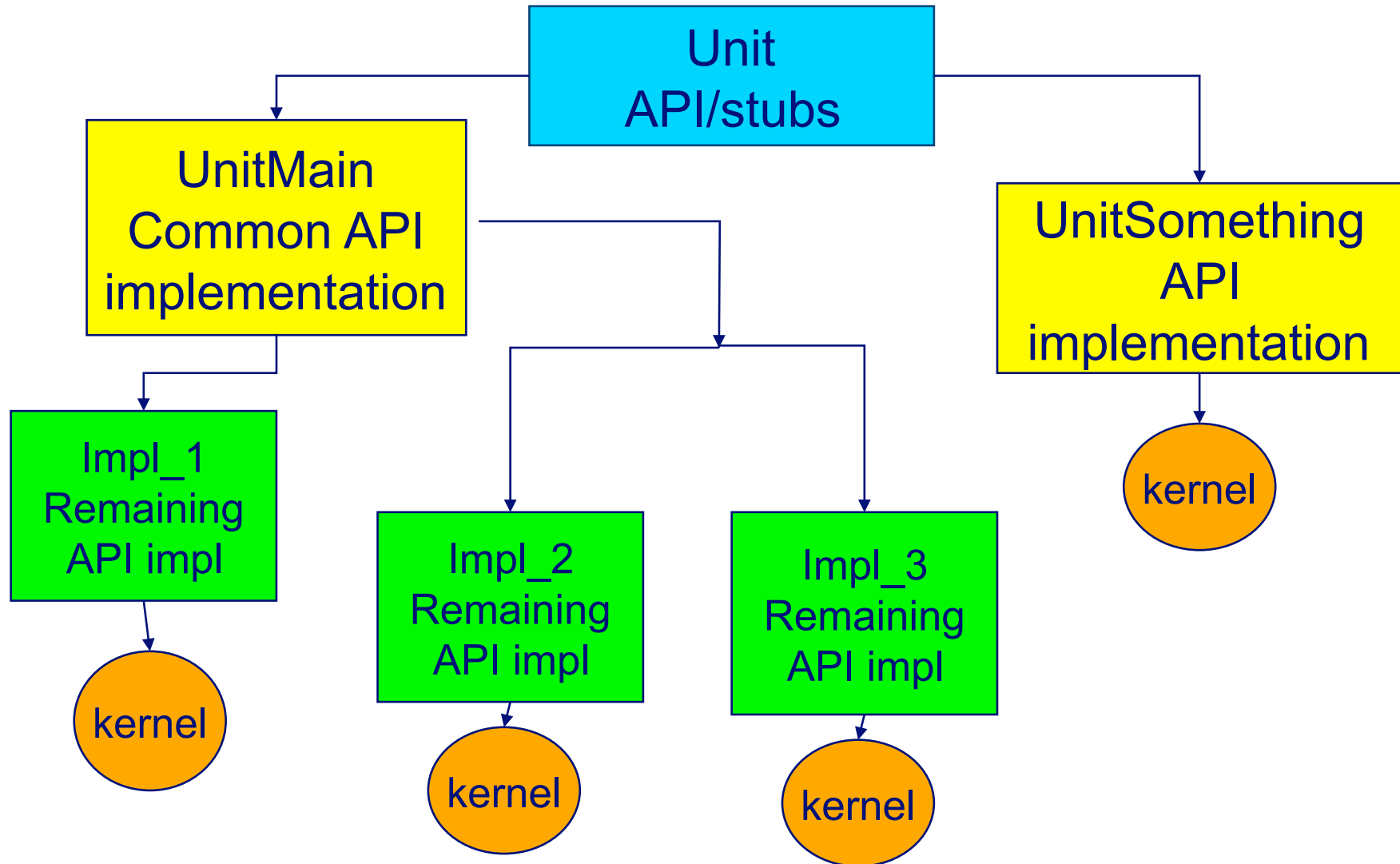
# More on Subunits

---

- ❑ A subunit may have multiple alternative implementations
- ❑ Alternative implementations of UnitMain also act as alternative implementations of the Unit.
- ❑ Some subunits have multiple implementations that could be included in the same simulation
  - ❑ GridSolver is one possible example.
  - ❑ Alternative implementations are specified using the “EXCLUSIVE” directive
- ❑ The “KERNEL” keyword indicates that subdirectories below that level need not follow FLASH architecture, and the entire subtree will be included in the simulation



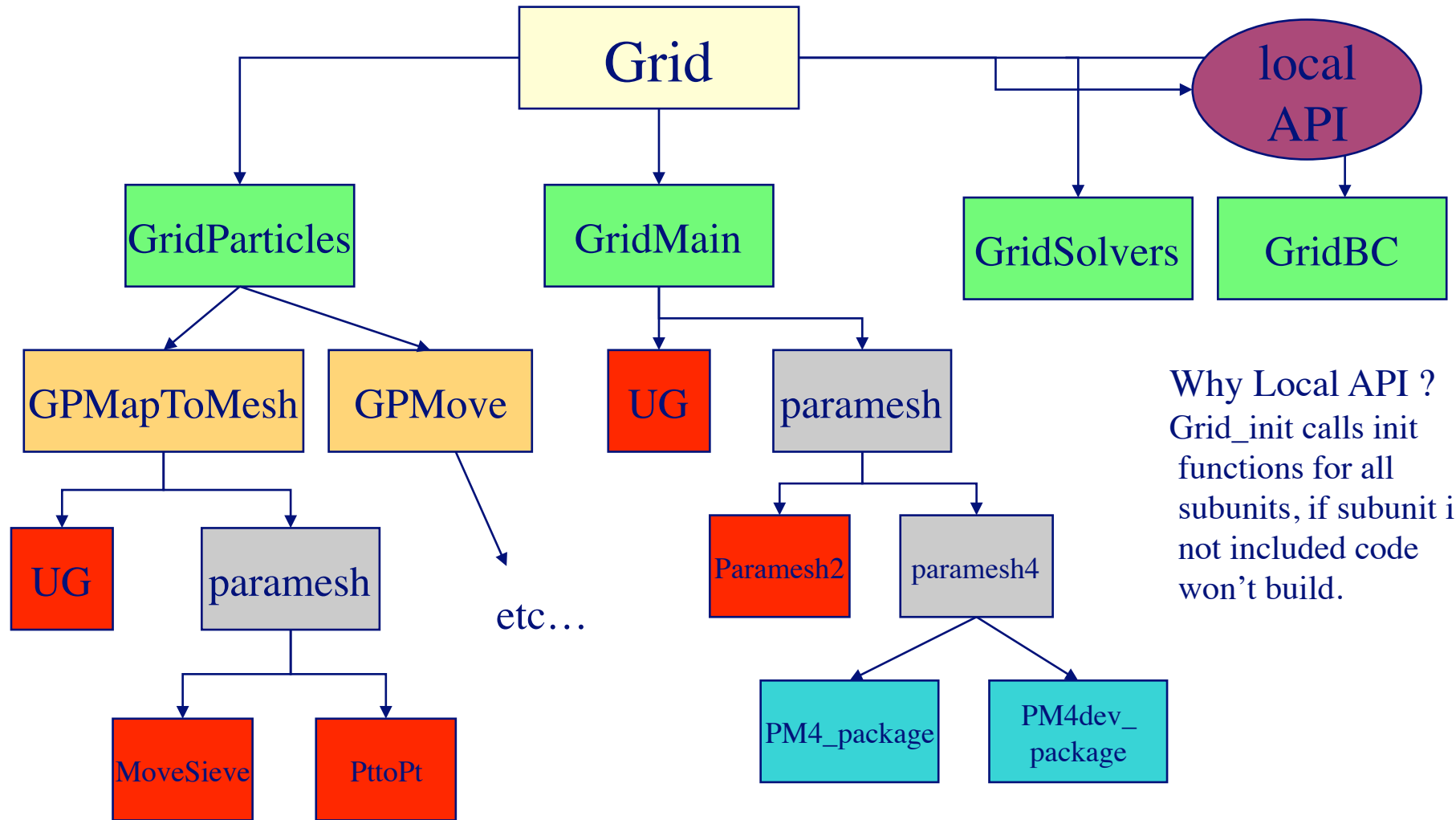
# Unit Hierarchy







# Example of a Unit – Grid (simplified)

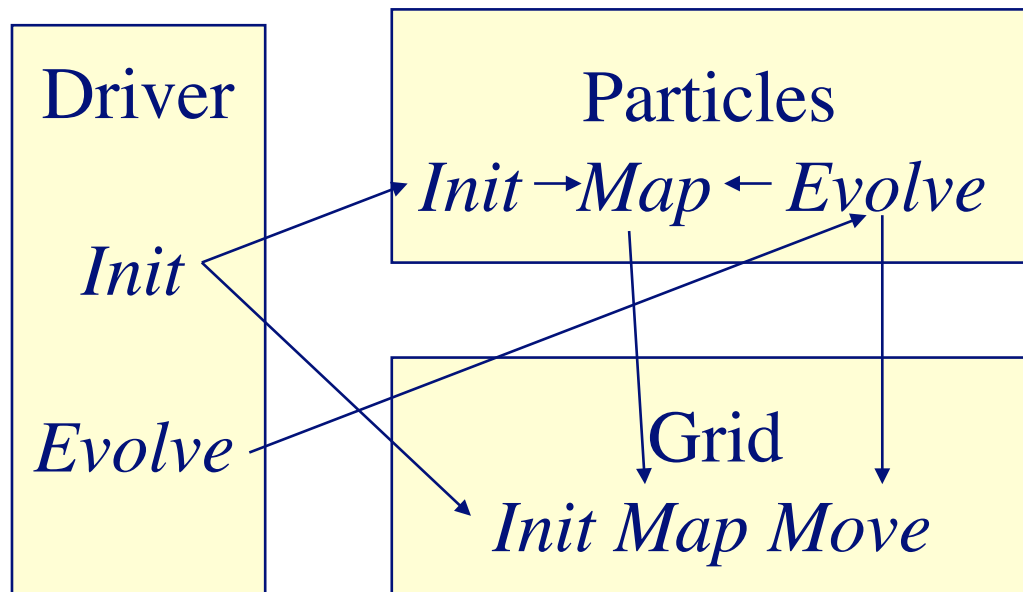


Why Local API ?  
Grid\_init calls init functions for all subunits, if subunit is not included code won't build.



# Functional Component in Multiple Units

- ❑ Example Particles
  - ❑ Position initialization and time integration in Particles unit
  - ❑ Data movement in Grid unit
  - ❑ Mapping divided between Grid and Particles
- ❑ Solve the problem by moving control back and forth between units





# Inheritance

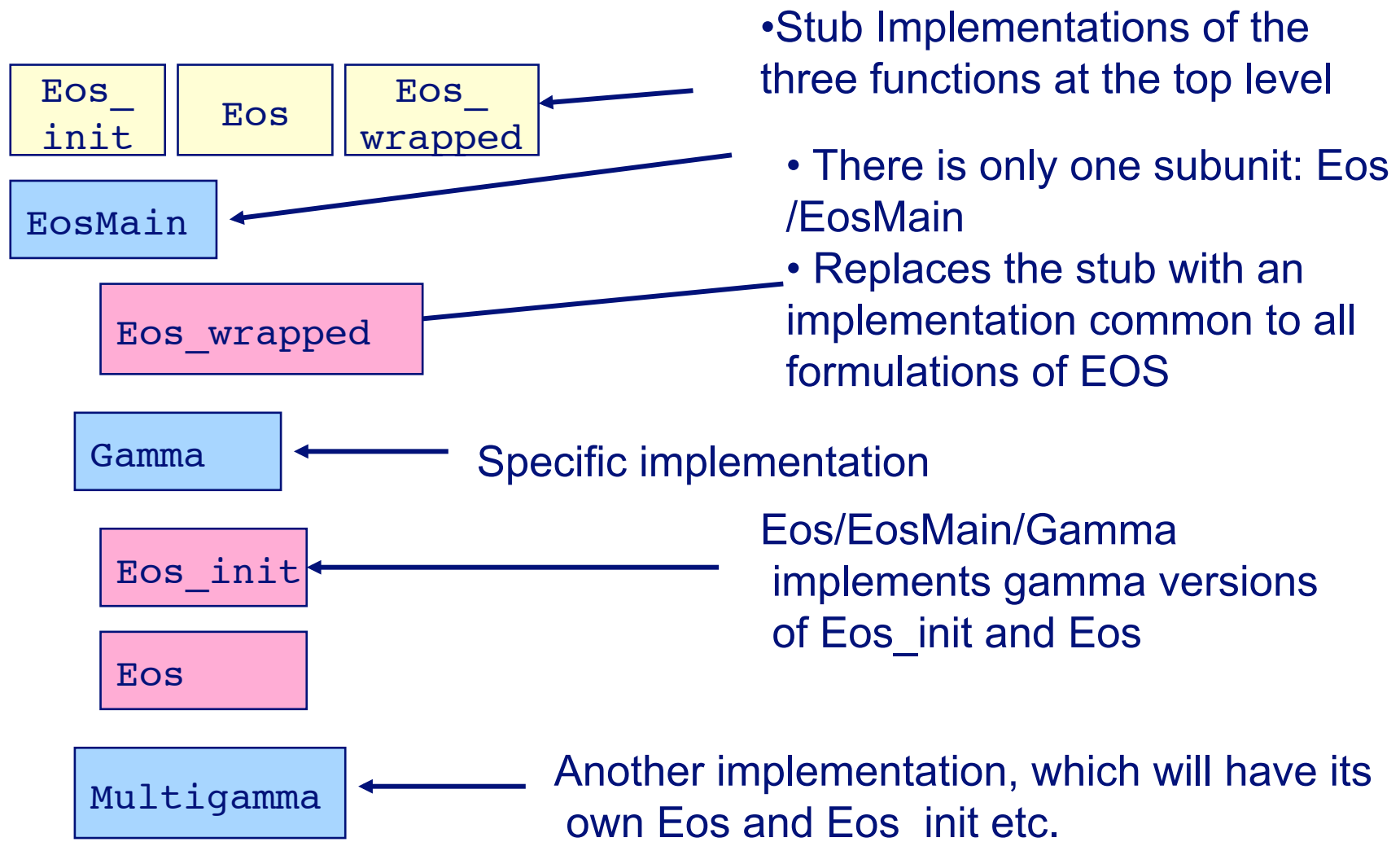
---

- ❑ Inheritance implemented through directory structure and Config file directives understood by the setup script
- ❑ A child directory inherits all functions from the parent directory
  - ❑ If the child directory has its own implementation of a function, it replaces the inherited one.
  - ❑ The implementation in the lowest level offspring replaces all implementations in higher level directories.
  - ❑ An implementation in the “Simulation/MyProblem” directory overrides all implementations when running MyProblem
- ❑ Config files arbitrate on multiple implementations through “Default” keyword
- ❑ Runtime environment is created by taking a union of all variables, fluxes, and runtime parameters in Config files of included directories.
  - ❑ Value given to a runtime parameter in the “Simulation/MyProblem/Config” overrides any value given to it in other Config files
  - ❑ Value in “flash.par” overrides any value given in any Config file

Multiple Config file initial values of a runtime parameter in units other than the simulation unit can lead to non-deterministic behavior since there are no other precedence rules.



# Inheritance Through Directories: Eos





## Naming Convention – Basic Files Rules

---

- ❑ Namespace directories are capitalized, organizational directories are not
- ❑ All API functions of unit start with Unit\_ (i.e. Grid\_getBlkPtr, Driver\_initFlash etc)
- ❑ Subunits have composite names that include unit name followed by a capitalized word describing the subunit (i.e. ParticlesMain, ParticlesMapping, GridParticles etc)
- ❑ Private unit functions and unit scope variables are named un\_routineName (i.e. gr\_createDomain, pt\_numLocal etc)
- ❑ Private functions in subunits other than UnitMain are encouraged to have names like un\_suRoutineName, as are the variables in subunit scope data module



## Naming Conventions: Within files

---

- ❑ Constants are all uppercase, usually have preprocessor definition, multiple words are separated by an underscore.
  - ❑ Permanent constants in “constants.h” or “Unit.h”
    - ❑ #define MASTER\_PE 0
    - ❑ #define CYLINDRICAL 3
  - ❑ Generated by setup script in “Flash.h”
    - ❑ #define DENS\_VAR 1
    - ❑ #define NFACE\_VARS 6
- ❑ Style within routines
  - ❑ Variables from Unit\_data start with unit\_variable: “eos\_eintSwitch”
  - ❑ Variables begin lowercase, additional words begin with uppercase: “massFraction”



## Naming Conventions – How they help

---

- ❑ The significance of capitalizing unit names:
  - ❑ A new unit can be added without the need to modify the setup script.
  - ❑ If the setup script encounters a top level capitalized directory without an API function to initialize the unit, it issues a warning.
- ❑ Variable Style:
  - ❑ Immediately clear if variable is CONSTANT, local (massFraction) or global (eos\_eintSwitch) in scope



# Setup Script Implements Architecture

---

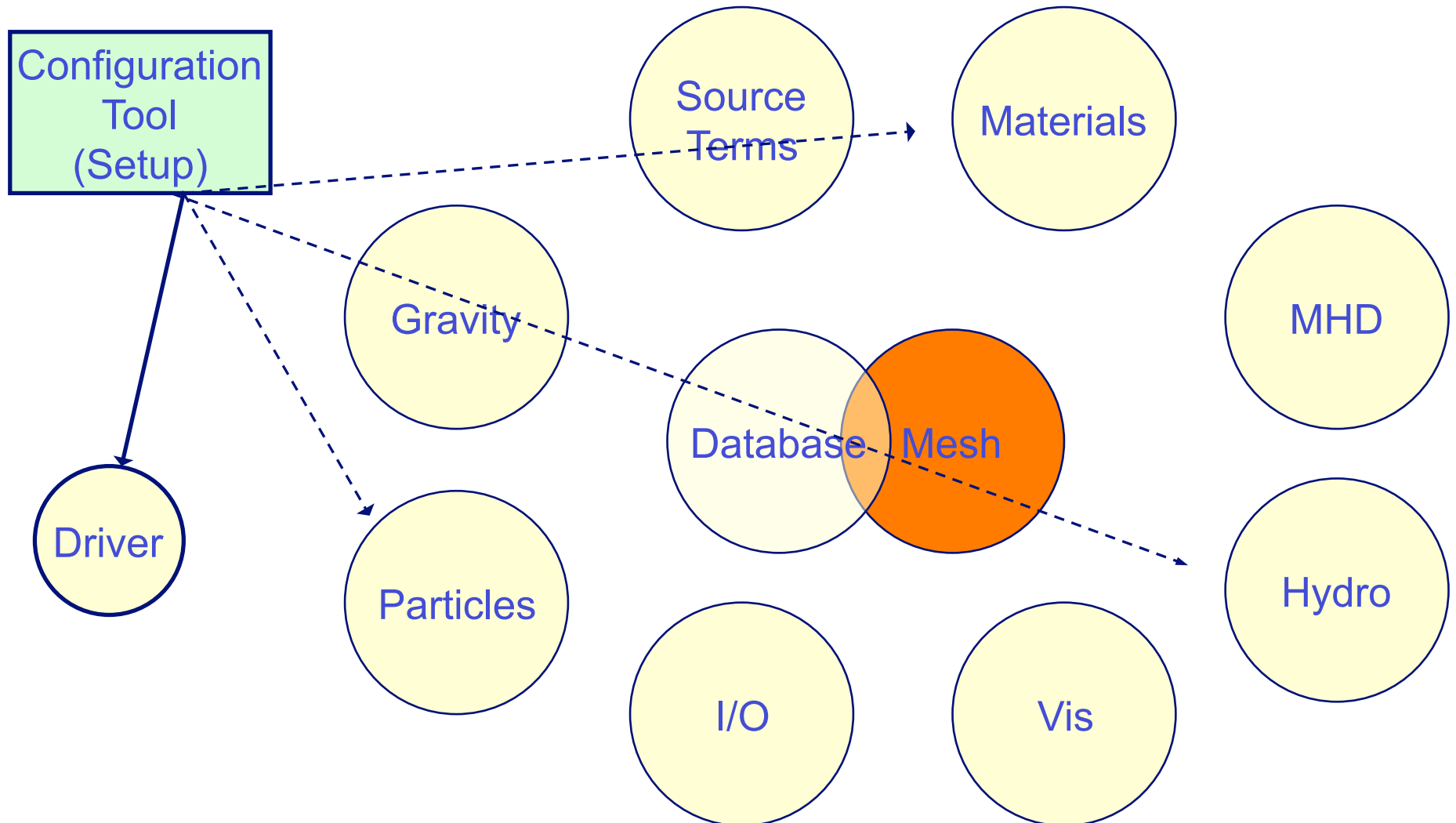
## **Python code links together needed physics and tools for a problem**

- ❑ Traverse the FLASH source tree and link necessary files for a given application to the object directory
- ❑ Creates a file defining global constants set at build time
- ❑ Builds infrastructure for mapping runtime parameters to constants as needed
- ❑ Configures Makefiles properly
- ❑ Determine solution data storage list and create Flash.h
- ❑ Generate files needed to add runtime parameters to a given simulation.
- ❑ Generate files needed to parse the runtime parameter file.





# Setup Building an Application





## Config file: Purpose

---

- ❑ Written in a FLASH-dependent syntax
- ❑ Needed in each Unit or Simulation directory
- ❑ Define dependencies at all levels in the source tree:
  - ❑ Lists required, requested, exclusive modules
- ❑ Declare solution variables, fluxes
- ❑ Declare runtime parameters
  - ❑ Sets defaults and allowable ranges – do it early!
  - ❑ Documentation – start line with “D”
- ❑ Variables, Units are additive down the directory tree
- ❑ Provides warnings to prevent dumb mistakes
  - ❑ Better than compiling and then crashing



# Config Files: Spiffy bits

---

- ❑ Programming syntax of sorts:
  - ❑ Allows some implementations choices with LINKIF
  - ❑ Gives Unit-specific pre-processor symbols with PPDEFINE
  - ❑ Comments are supported, use them!
- ❑ Hints:
  - ❑ Keep the parameter space clean & documented
  - ❑ Use the least aggressive request for an implementation
    - ❑ Consider REQUESTS instead of REQUIRES
    - ❑ Use REQUIRES with the most general level possible (i.e. Grid /GridMain rather than a specific implementation)
  - ❑ Set default values and constraints
    - ❑ The earlier an error is detected, the easier it is to fix.
    - ❑ Prevents operator misuse



# Config file example

```
# Configuration File for setup Stirring Turbulance
REQUIRES Driver
REQUIRES physics/sourceTerms/Stir/StirMain
REQUIRES physics/Eos
REQUIRES physics/Hydro
REQUIRES Grid
REQUESTS IO
```

Required Units

```
# include IO routine only if IO unit included
LINKIF IO_writeIntegralQuantities.F90 IO/IOMain
LINKIF IO_writeUserArray.F90 IO/IOMain/hdf5/parallel
LINKIF IO_readUserArray.F90 IO/IOMain/hdf5/parallel
```

Alternate local IO routines

```
LINKIF IO_writeUserArray.F90.pnetcdf IO/IOMain/pnetcdf
LINKIF IO_readUserArray.F90.pnetcdf IO/IOMain/pnetcdf
```

Runtime parameters and  
documentation

```
D      c_ambient      reference sound speed
D      rho_ambient    reference density
D      mach            reference mach number
PARAMETER c_ambient  REAL    1.e0
PARAMETER rho_ambient REAL    1.e0
PARAMETER mach       REAL    0.3
```

Additional scratch grid variable

```
GRIDVAR mvrt
```

```
USESETUPVARS nDim
IF nDim <> 3
```

```
  SETUPERROR At present Stir turb works correctly only in 3D. Use ./setup StirTurb -3d blah blah
ENDIF
```

Enforce geometry or other conditions



# Simple setup

```
hostname:Flash3> ./setup MySimulation -auto
```

setup script will automatically generate the object directory based on the MySimulation problem you specify

## Sample Units File

```
INCLUDE Driver/DriverMain/TimeDep
INCLUDE Grid/GridMain/paramesh/Paramesh3/PM3_package/headers
INCLUDE Grid/GridMain/paramesh/Paramesh3/PM3_package/mpi_source
INCLUDE Grid/GridMain/paramesh/Paramesh3/PM3_package/source
INCLUDE Grid/localAPI
INCLUDE IO/IOMain/hdf5/serial/PM
INCLUDE PhysicalConstants/PhysicalConstantsMain
INCLUDE RuntimeParameters/RuntimeParametersMain
INCLUDE Simulation/SimulationMain/Sedov
INCLUDE flashUtilities/general
INCLUDE physics/Eos/EosMain/Gamma
INCLUDE physics/Hydro/HydroMain/split/PPM/PPMKernel
INCLUDE physics/Hydro/HydroMain/utilities
```

Try manually changing IO  
/IOMain/hdf5/serial/PM to IO  
/IOMain/hdf5/parallel/PM -  
Then run setup without the  
-auto flag

*If you don't use the -auto flag, you must have a valid Units file  
in the object FLASH directory (FLASH3/object/Units)*



## setup Shortcuts & help

---

- ❑ `./setup --help` shows many fascinating options
- ❑ Shortcuts allows many setup options to be included with one keyword
- ❑ To use a shortcut, add `+shortcut` to your setup line
  - ❑ The shortcut `ug` is defined as:
    - ❑ `ug:--with-unit=Grid/GridMain/:Grid=UG:`
    - ❑ `prompt> ./setup MySimulation -auto +ug`
    - ❑ this is equivalent to typing in unit options with
      - ❑ `-unit=Grid/GridMain/UG`
      - ❑ `-unit=IO/IOMain/hdf5/serial/UG` (because the appropriate IO is included by default)
- ❑ Look in `Flash3/bin/setup_shortcuts.txt` for more examples and to define your own



# Important Files Generated by setup

<b>setup_call</b>	contains the options with which setup was called and the command line resulting after shortcut expansion
<b>setup_datafiles</b>	contains the complete path of data files copied to the object directory
<b>setup_defines</b>	contains a list of all pre-process symbols passed to the compiler invocation directly
<b>setup_flags</b>	contains the exact compiler and linker flags
<b>setup_libraries</b>	contains the list of libraries and their arguments (if any) which was linked in to generate the executable
<b>setup_params</b>	contains the list of runtime parameters defined in the Config files processed by setup
<b>setup_units</b>	contains the list of all units which were included in the current setup
<b>setup_vars</b>	contains the list of variables, fluxes, species, particle properties, and mass scalars used in the current setup, together with their descriptions



## Additional Files created by setup

---

- ❑ `Flash.h` contains
  - ❑ Problem dimensionality and size e.g. `NDIM`, `MAXBLOCKS`
  - ❑ Fixed block size dimensionality e.g. `NXB`, `GRID_IJI_GC`
  - ❑ Variable, species, flux, mass scalar numbers and list e.g. e.g. `NSPECIES`, `DENS_VAR`, `EINT_FLUX`
  - ❑ Possibly grid geometry `GRID_GEOM`
  - ❑ `PPDEFINE` variables showing which units are included e.g. `FLASH_GRID_PARAMESH3`
  
- ❑ `Simulation_mapIntToStr.F90`,  
`Simulation_mapStrToInt.F90`
  - ❑ Converts text strings to equivalent index in `Flash.h` e.g. “dens” maps to `DENS_VAR=1`
  - ❑ Similar functionality to `FLASH2`'s `dbaseKeyNumber`





# Architecture

---

Questions?



# setup Options: use `./setup -help`

<b>-auto</b>	Automatically generates the Units file
<b>-unit=&lt;unit&gt;</b>	Forces a specific unit to be used
<b>-without-unit=&lt;unit&gt;</b>	Forces a specific unit to be left out
<b>-[123]d</b>	Specifies the dimension, default is 2d
<b>-nxb=&lt;#&gt; -nyb=&lt;#&gt; -nzb=&lt;#&gt;</b>	Specifies number of zones/block, default is 8
<b>-maxblocks=&lt;#&gt;</b>	Assigns maxblocks per processor. Defaults are in place but you may want to modify depending on problem and machine specs
<b>-site=&lt;site&gt;   -ostype=&lt;ostype&gt;</b>	Allows you to directly specify the host or ostype. Typically setup finds this info but on some machines it isn't directly accessible. -site=sphere.asci.uchicago.edu or -ostype=Linux
<b>-debug   -test</b>	-debug: makes compiler put debugging symbols in executable, possibly checks for array out of bounds conditions, etc -test: compiles code with no debugging or optimization options -opt: compiles code for highest performance (default)
<b>-portable</b>	Normally setup links files from the source directory to the object directory. With -portable files are copied to object dir instead
<b>-objdir=&lt;relative obj dir&gt;</b>	By default setup script links all files needed in compilation to the object directory. The -objdir flag allows you to specify a different or new directory
<b>-noclobber</b>	To setup a simulation over a previous compiled object directory, so that previously included units are not recompiled, saving compilation time
<b>-verbose</b>	More wordy explanations during setup
<b>-parfile=&lt;file&gt;</b>	This causes setup to copy the specified file in the simulation directory to the object directory as flash.par



# Basic Config File Syntax

<b>DEFAULT</b> <i>sub-unit</i>	Every unit and subunit designates one implementation to be the ``default''. If no specific implementation of the unit or its sub-units is selected by the application, the designated default implementation gets included
<b>EXCLUSIVE</b> <i>implementation...</i>	Specify a list of implementations that cannot be included together
<b>REQUIRES</b> <i>unit[/sub-unit[/implementation...]] [ OR unit[/sub-unit...]]...</i>	Specify a unit requirement. Unit requirements can be general, without asking for a specific implementation, so that unit dependencies are not tied to particular algorithms
<b>REQUESTS</b> <i>unit[/sub-unit[/implementation...]]</i>	Requests a unit to be added to the Simulation. All requests, are upgraded to a ``REQUIRES'' if they are not negated by a "-without-unit" option from the command line. If negated, the REQUEST is ignored. This can be used to turn off profilers and other ``optional'' units which are included by default.
<b>CONFLICTS</b> <i>unit1[/sub-unit[/implementation...]] ...</i>	Specifies that the current unit, subunit or specific implementation is not compatible with the list of units, subunits or other implemenations list that follows. Setup issues an error if the user attempts to set up a conflicting unit configuration.
<b>PARAMETER</b> <i>name type [constant] default [range spec]</i>	Specify a runtime parameter. Parameter names are unique up to 20 characters and may not contain spaces. Admissible types include REAL, INTEGER, STRING, and BOOLEAN
<b>DATAFILES</b> <i>wildcard</i>	Declare that all files matching the given wildcard in the unit directory should be copied over to the object directory.



# More Config File Syntax

---

<b>VARIABLE</b> <i>name</i>	Register variable with the framework with name <i>name</i> . The setup script collects variables from all the included units, and creates a comprehensive list with no duplications. It then assigns defined constants to each variables and calculates the amount of storage required in the data structures for storing the variables. The defined constants, and the calculates sizes are written to the file Flash.h.
<b>FLUX</b> <i>name</i>	Register flux variable <i>name</i> with the framework. When using adaptive mesh, flux conservation is needed at fine-coarse boundaries. Paramesh uses a data structure for this purpose, the flux variables provide index into that data structure.
<b>SPECIES</b> <i>name</i>	An application that uses multiple species uses this keyword to define them
<b>MASS_SCALAR</b> <i>name</i>	If a quantity is defined with keyword MASS_SCALAR, space is created for it in the ``unk" data structure. It is treated like any other variable by Paramesh, but the hydrodynamic unit treats it differently. It is advected, but other physical characteristics don't apply to it.
<b>GRIDVAR</b> <i>name</i>	This keyword is used in connection with the grid scope scratch space supported by FLASH3 (This feature wasn't available in FLASH2). It lets you ask for scratch space for variables specified with this keyword.
<b>PPDEFINE</b> <i>name</i>	Define a preprocessor symbol which appears in Flash.h. Useful for differentiating Fortran code which depends upon a certain Unit.
<b>USESETUPVARS</b> <i>var</i>	Provides logical action within setup.
<b>LINKIF</b> <i>file unit/directory/path</i>	Provides alternate linking routines dependent upon included Unit path.