Simulation Directory I

Klaus Weide / Chris Daley / Sean Couch

The University of Chicago

The Simulation Unit

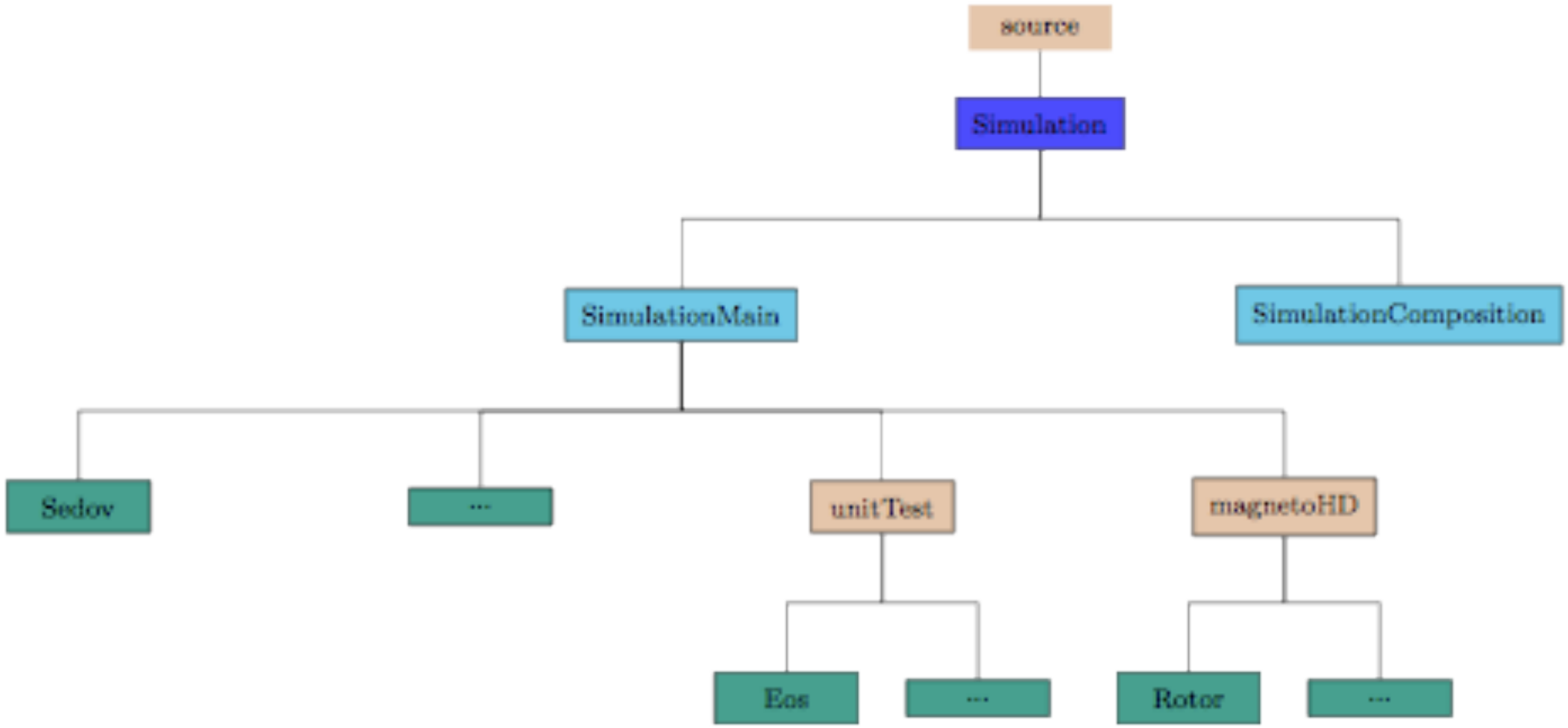

Figure 22.1: The `Simulation` unit directory tree. Only some of the provided simulation implementations are shown. Users are expected to add their own simulations to the tree.

The Simulation Unit

- Typical Unit, obeys architecture, naming conventions, inheritance, etc. rules.
- Special Unit in that it always “wins” inheritance and parameter wars.
- FLASH problems is defined by directories in source/Simulation/SimulationMain.
- The Simulation directory gives people working on a particular problem a place to put problem specific code that replaces the default functionality in the main body of the code
- It’s also a place to tell the setup script which units this problem will need from the rest of the code

What's in the Simulation Directory?

- Normal UnitMain implementation requirements
  - Simulation_data, Simulation_init, (Simulation_finalize), Simulation_initBlock
  - Makefile (with usually Simulation_data only)
  - Config file
  - Possibly other API functions: e.g. Simulation_initSpecies
- Specific to simulations:
  - Parameter files flash.par, testUG.par, etc.
  - Replacements for routines located elsewhere in directory tree
  - Routines that implement local functions e.g. sim_derivedVariables.F90

Required Code for a New Simulation

- There are certain pieces of code that all simulations must implement:
  - Simulation_data.F90: Fortran module which stores data and parameters specific to the Simulation.
  - Simulation_init.F90: Reads the runtime parameters, and performs other necessary unit initializations.
  - Simulation_initBlock.F90: Sets initial conditions in a single block.
- Optionally, a simulation could implement:
  - Simulation_initSpecies.F90: To give the properties of the species involved in a multispecies simulation

Customized Code for a new Simulation

- In a FLASH simulation directory, you can place code that overrides the functionality you would pick up from other code units
- In the custom code you can modify:
  - Boundary conditions (Grid_bcApplyToRegionSpecialized.F90, or Grid_applyBCEdge.F90)
  - Refinement criterion (Grid_markRefineDerefine.F90)
  - Diagnostic integrated quanties for output (in the flash.dat file), e.g., total mass (a default) or vorticity (IO_writeIntegralQuantities.F90)
  - Diagnostics to compute new grid scope variables (Grid_computeUserVars.F90)
- In general, this is a place to hack the code in ways specific to your problem, and you can hack basically anything