WhiskyTHC

Technical Overview

David Radice — May 2020

Motivation

Design decisions

Many problems in physics (including GRHD) can be written in the form of conservation laws:

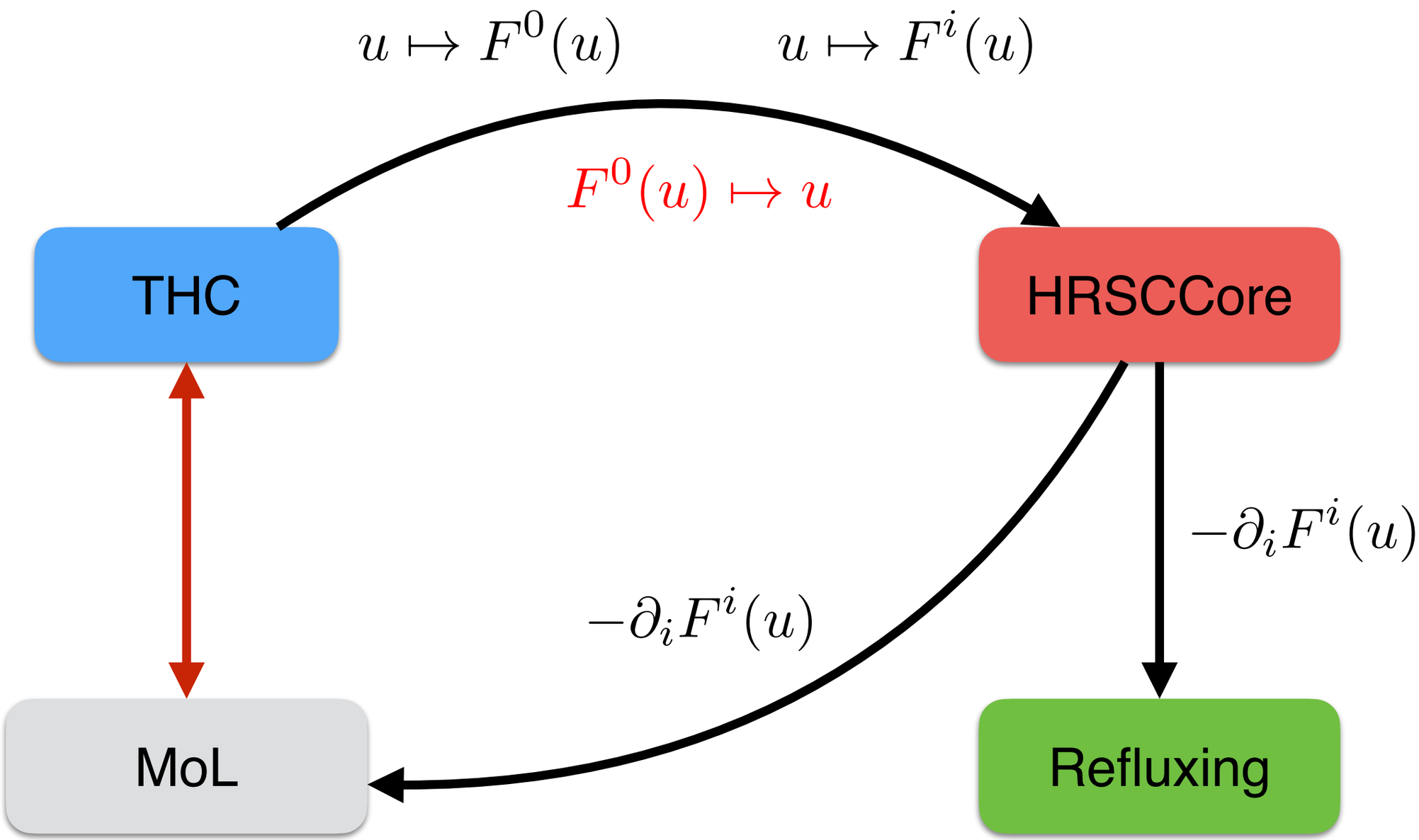
$$\partial_t F^0(u) + \partial_i F^i(u) = S(u)$$

Numerical solution to conservation laws require specialized numerical methods that need to be implemented carefully.

Numerical kernels do not need to know the details of the physics and physics codes do not need to worry about implementation details.

WhiskyTHC enforces this separation of responsibilities

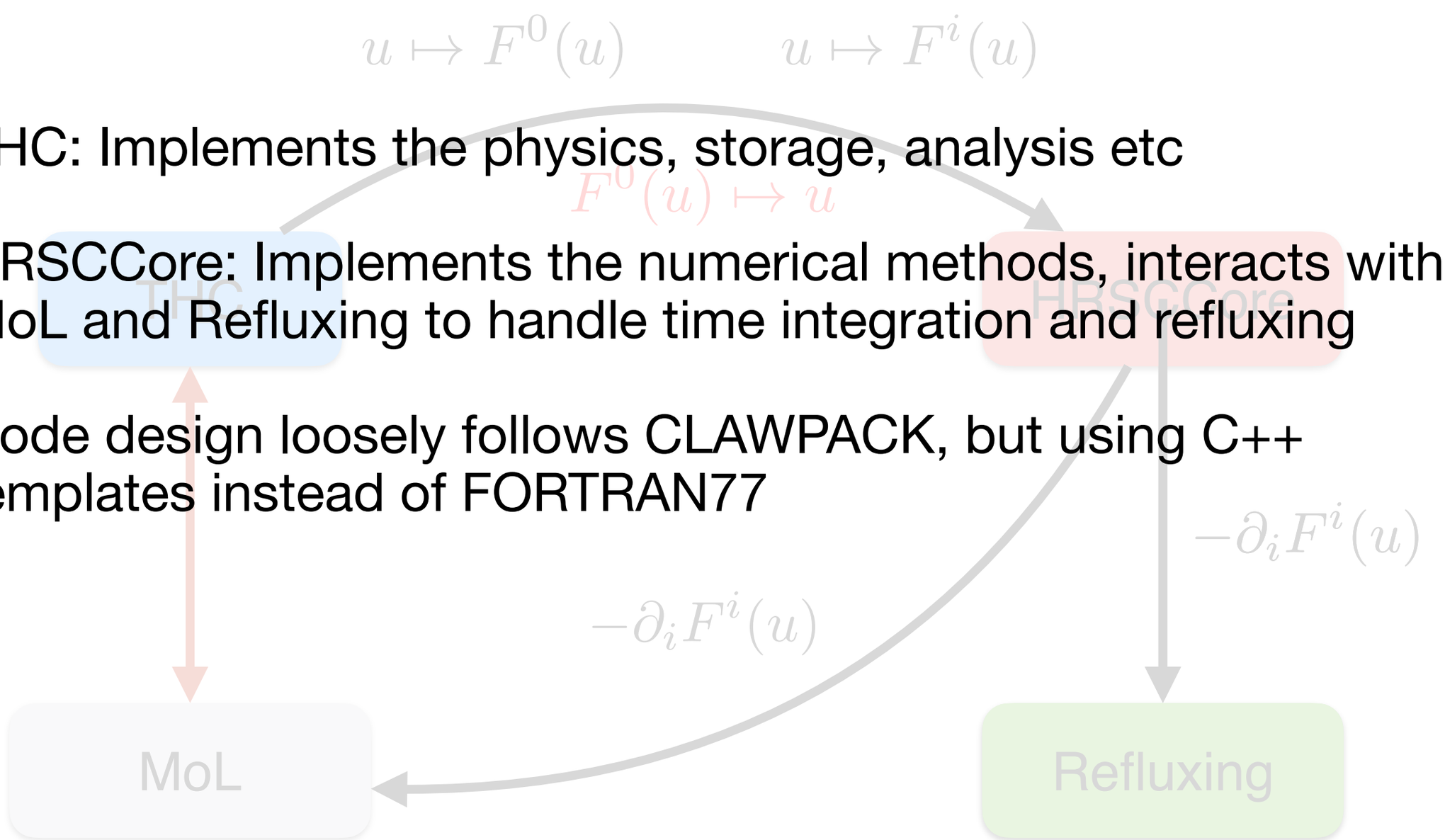
High-Level Overview



$$\partial_t F^0(u) = -\partial_i F^i(u) + S(u)$$

High-Level Overview

- THC: Implements the physics, storage, analysis etc
- HRSCCore: Implements the numerical methods, interacts with MoL and Refluxing to handle time integration and refluxing
- Code design loosely follows CLAWPACK, but using C++ templates instead of FORTRAN77



$$\partial_t F^0(u) = -\partial_i F^i(u) + S(u)$$

Code Organization

Three arrangements:

- THCBase: core utilities, numerical methods, and examples
- THCCore: core physics routines and multi-physics schemes (such as neutrino leakage etc.)
- THCExtra: thorns that are not tightly coupled with THC (many come from Wolfgang Kastaun's WhiskyThermal). Examples are ID thorns, EOS tables, analysis routines, NS trackers, etc… These are the thorns that give the "Whisky" part of the name.

Code Organization

THCBase

- CPPUtils: basic C++ classes such as tensor classes, macros for bitmasks, and loops
- FDCore: stencils for linear FD schemes, primarily used to compute metric derivative in the RHS and analysis quantities.
- HRSCCore: thorn implementing the numerical schemes

Code Organization

THCCore

- THC_Boundary: custom boundary conditions for test problems
- THC_Core: main physics driver
- THC_HydroExcision
- THC_InitialData: mostly test initial data
- THC_LeakageBase: neutrino leakage scheme
- THC_LeakageM0: heating sources used by THC_LeakageBase
- THC_Refluxing: registers fluxes with the Refluxing thorn
- THC_Test: unit tests (only few implemented)
- THC_Tracer: passive scalars

Code Organization

THCExtra

THCExtra has many thorns, they can be divided into few groups

- Infrastructure from the Pizza code: PizzaBase, PizzaIDBase, PizzaNumUtils
- EOS_Thermal framework: EOS_Barotropic, EOS_Thermal, EOS_Thermal_Extable, EOS_Thermal_Hybrid, EOS_Thermal_Idealgas, EOS_Thermal_Table3d, ID_Switch_EOS
- Neutrino rates: WeakRates and FakeRates
- Initial data: AddPuncture, LoreneID, PizzaTOV
- Trackers: BHTrackerMany, BNSTrackerGen, NSTracker
- Analysis: BNSAnalysis

There are also scripts for importing EOS tables and a TOV solver

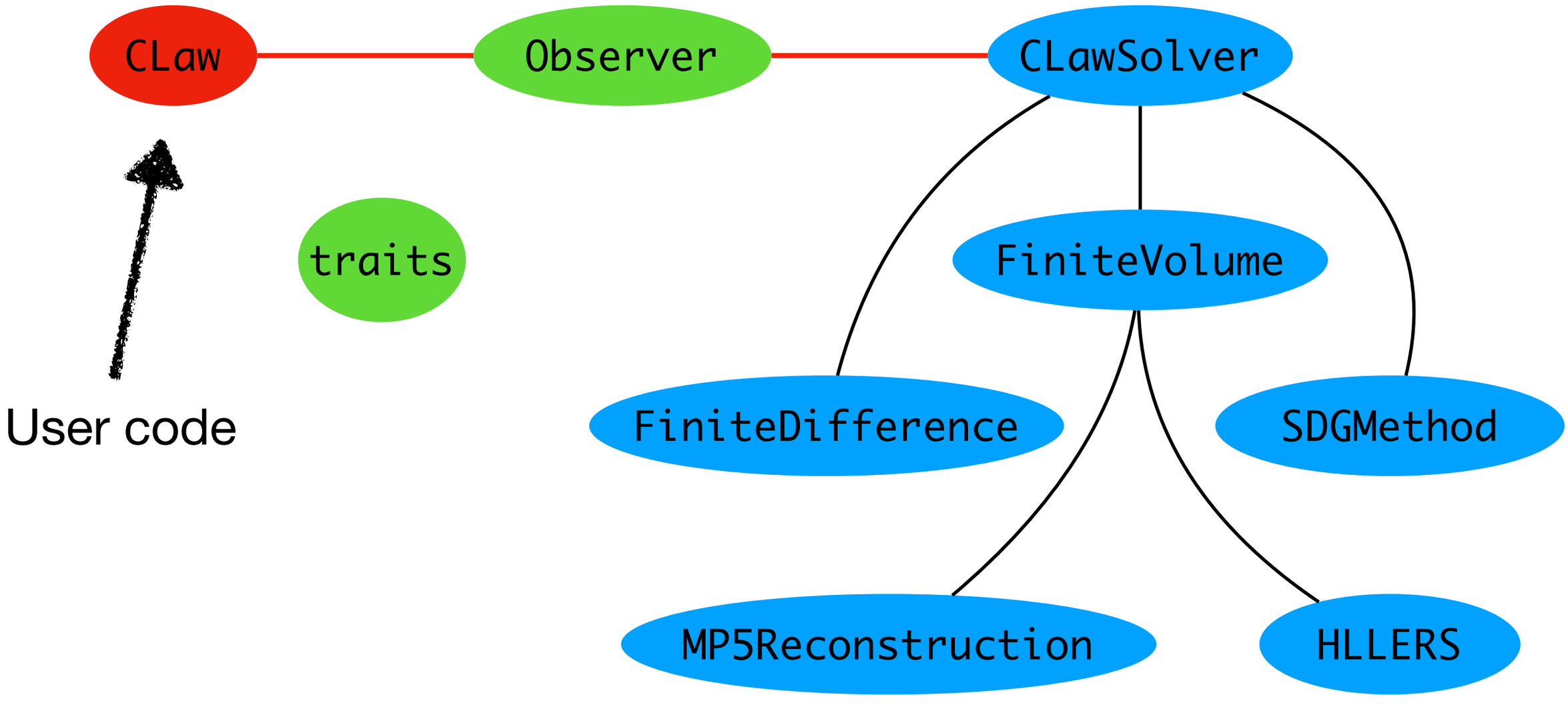
HRSCCore

Functionality

- Provides numerical methods to solve any balance law
- Currently available are: Kurganov-Tadmor central schemes, finite volume, HRSC finite difference (WENO, MP5), and spectral discontinuous Galerkin (SDG).
- All these schemes are tested and known to work, but shock capturing is not yet implemented for SDG.
- Uses template metaprogramming to inline calls to user-provided routines (`prim_to_all`, `cons_to_all`, etc).

HRSCCore

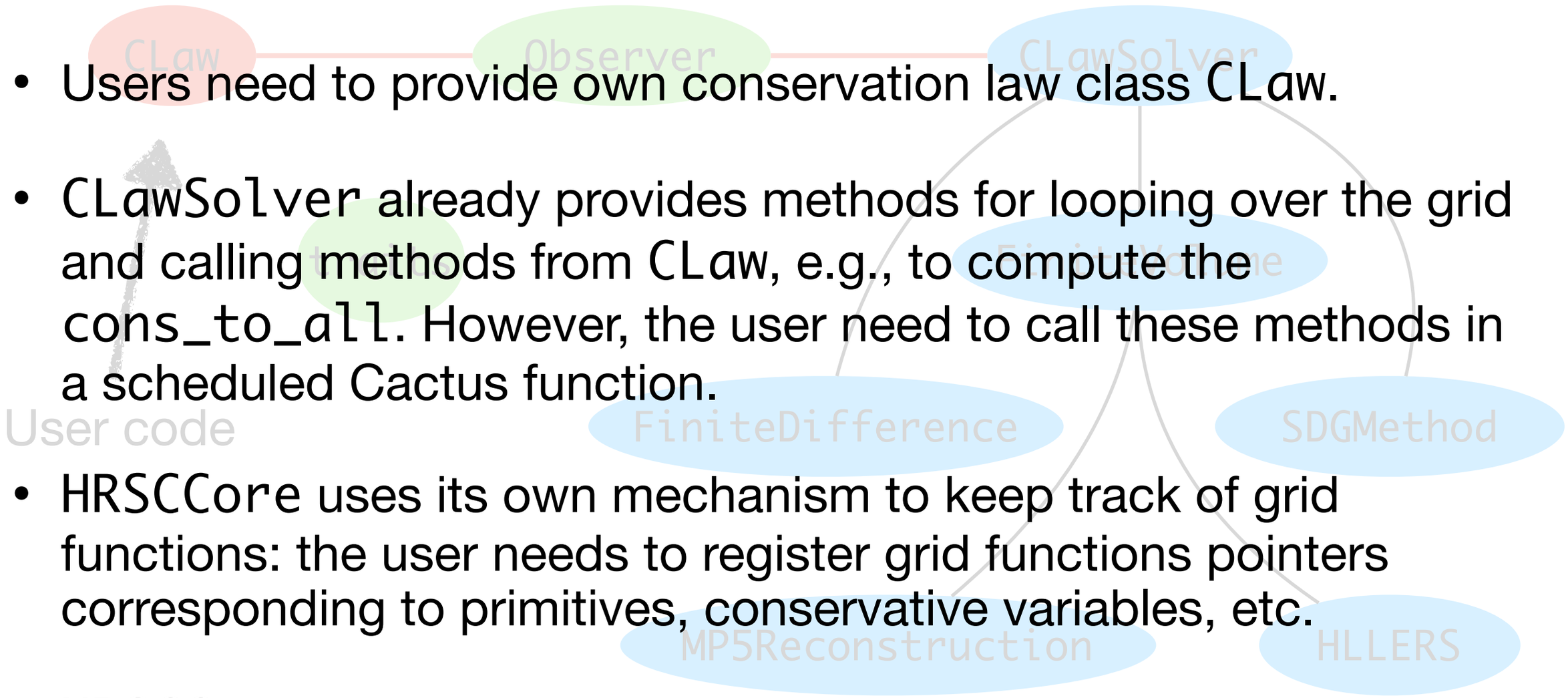
Structure



HRSCCore

Structure

- Users need to provide own conservation law class `CLaw`.
- `CLawSolver` already provides methods for looping over the grid and calling methods from `CLaw`, e.g., to compute the `cons_to_all`. However, the user need to call these methods in a scheduled Cactus function.
- `HRSCCore` uses its own mechanism to keep track of grid functions: the user needs to register grid functions pointers corresponding to primitives, conservative variables, etc.
- `HRSCCore` can handle any number of conservative variables and does not distinguish between them



HRSCCore

Example: cons_to_all

```
namespace hrscc {

template<typename law_t, typename method_t>
class CLawSolver {
    public:
        typedef law_t law;
        typedef CLaw<law> claw;
        typedef method_t method;

        […]

        //! compute all the variables from the conservatives on the grid
        void cons_to_all() {
#pragma omp parallel
            {
                Observer<claw> observer(_M_cctkGH, _M_coordinates,
                        _M_grid_variable, _M_bitmask);

                UTILS_LOOP3(cons_to_all,
                        k, 0, _M_cctkGH->cctk_lsh[2],
                        j, 0, _M_cctkGH->cctk_lsh[1],
                        i, 0, _M_cctkGH->cctk_lsh[0]) {
                    observer.jump_to_location(i, j, k);
                    _M_claw->cons_to_all(observer);
                    observer.record();
                } UTILS_ENDLOOP3(cons_to_all);
            }
        }

        […]
```

Observer is used to pass data to/from the user provided CLaw specialization

Call into user provided `cons_to_all`

Update grid functions with new primitive variables

hrscc_claw_solver.hh

CLaw

```
namespace hrscc {

//! conservation law prototype
/*!
 *  This class provides the specifications for the physics driver. The final
 *  user of the HRSCCore thorn is supposed to create a class \e MyLaw
 *  \code
 *      class MyLaw: public hrscc::CLaw<MyLaw>;
 *  \endcode
 *  which overloads all the virtual methods present here.
 *
 *  The user should also provide the required traits of his/her class by
 *  specialising hrscc::traits
 *
 *  \tparam derived_t derived class (for static polymorphism)
 */
template<typename derived_t>
class CLaw {
    public:
        typedef derived_t derived;

        //! number of equations in our conservation law
        enum {nequations = traits<derived>::nequations};
        //! number of external fields directly needed during the evolution
        enum {nexternal  = traits<derived>::nexternal};
        //! total number of CCTK_REAL variables used for the evolution
        enum {nvariables = 3*nequations + nexternal};
        //! number of bitmask fields used by the physics driver
        enum {nbitmasks  = traits<derived>::nbitmasks};

        //! \brief if true this means that we have an exact conservation law,
        //! ie with no sources
        static bool const pure = traits<derived>::pure;

        […]
```

User provided class

Number of variables

traits are used for reflection

CLaw

```
namespace hrscc {

template<typename derived_t>
class CLaw {
    public:
        […]

        //! Cactus variable indices of the conserved quantities
        static int conserved_idx[nequations];
        //! Cactus variable indices of the primitive quantities
        static int primitive_idx[nequations];
        //! Cactus variable indices of the RHS variables
        static int rhs_idx[nequations];
        //! Cactus variable indices for the extra fields
        static int field_idx[nexternal];
        //! Cactus variable indices of the bitmasks used by the physics driver
        static int bitmask_idx[nbitmasks];
        //! Cactus variable indices for the numerical fluxes in each direction
        /*!
         *  Each point stores the flux flowing to its left
         */
        static int num_flux_idx[3*nequations];

        //! Lower bound for the conserved variables (for the positivity limiter)
        static CCTK_REAL conserved_lbound[nequations];

        //! compute all the variables from the primitive ones in a point
        inline void prim_to_all(
                //! [in, out] local value of all the variables
                Observer<CLaw<derived> > & observer
                ) const {
            static_cast<derived const*>(this)->prim_to_all(observer);
        }

        […]
```

Cactus var indices

This is compile time polymorphism

Observer

Used to exchange data between HRSCCore, Cactus, and the physics thorns

Debugging information

True if we are on a cell face

```
namespace hrscc {

template<typename claw_t>
class Observer {
    public:
        typedef claw_t claw;

        //! cactus grid hierarchy
        cGH const * const cctkGH;

        //! conserved variables at the point
        CCTK_REAL conserved[claw::nequations];
        //! primitive variables at the point
        CCTK_REAL primitive[claw::nequations];
        //! rhs at the point
        CCTK_REAL rhs[claw::nequations];
        //! external fields at the point
        CCTK_REAL field[claw::nexternal];

        //! bitmasks at the point
        CCTK_INT bitmask[claw::nbitmasks];

        //! array for the fluxes at the point
        CCTK_REAL flux[3][claw::nequations];
        //! array of eigenvalue at the point
        CCTK_REAL eigenvalue[claw::nequations];
        //! \brief array of left eigenvectors at the point stored in a
        //! row-major order
        CCTK_REAL left_eigenvector[claw::nequations][claw::nequations];
        //! \brief array of right eigenvectors at the point stored in a
        //! row-major order
        CCTK_REAL right_eigenvector[claw::nequations][claw::nequations];

        //! current location on the grid
        CCTK_REAL x, y, z;

        //! current position in the grid (local index)
        int i, j, k;
        int ijk;

        //! signals if the current position is shifted from the grid point
        bool shift[3];
```

Observer

Public Member Functions

	Observer (cGH const *const **cctkGH**, CCTK_REAL const *const coordinates[3], CCTK_REAL *const grid_variable[claw::nvariables], CCTK_INT *const **bitmask**[claw::nbitmasks]) The constructor. More...
void	**jump_to_location** (int i_, int j_, int k_) Moves the observer to a given grid point. More...
void	**linterp** (int i_, int j_, int k_, bool shift_x, bool shift_y, bool shift_z) jump to the given cell interface and reconstruct the fields with a linear interpolation More...
void	**record** () writes values of the variables into the grid functions More...
CCTK_REAL	**check_eigenvectors** () const consistency check on the given eigenvectors More...
void	**reset_eigenvalues** () reset the eigenvalues More...
void	**reset_eigenvectors** () reset the eigenvectors More...

(Yes all of HRSCCore is documented using Doxygen!)

CLawSolver

```
namespace hrscc {

template<typename law_t, typename method_t>
class CLawSolver {
    public:
        typedef law_t law;
        typedef CLaw<law> claw;
        typedef method_t method;

        […]

    protected:
        //! cactus grid hierarchy
        cGH const * const _M_cctkGH;
        //! physical driver class
        claw * _M_claw;

        […]

        //! global index for all the grid variables registered by claw
        /*!
         *  this index does not include the numerical fluxes
         */
        CCTK_REAL * _M_grid_variable[claw::nvariables];
        //! alias for the conserved variables
        CCTK_REAL * _M_conserved[claw::nequations];
        //! alias for the primitive variables
        CCTK_REAL * _M_primitive[claw::nequations];
        //! alias for the grid variables containing the RHS
        CCTK_REAL * _M_RHS[claw::nequations];
        //! alias for the external fields registered by claw
        CCTK_REAL * _M_field[claw::nexternal];

        //! bitmasks requested by claw
        CCTK_INT  * _M_bitmask[claw::nbitmasks];

        //! numerical fluxes
        CCTK_REAL * _M_num_flux[3][claw::nequations];
};
```

Cactus grid functions, we use `CCTK_VarDataPtrI` to initialize these quantities using the indices stored in the `CLaw` class.

CLawSolver

Methods I

Public Member Functions

	CLawSolver (cGH const *const cctkGH) constructor More...
	~CLawSolver () destructor More...
claw const *	**get_claw** () const get a read-only reference to the conservation law More...
Observer< **claw** > *	**observer_alloc** () const make an observer More...
void	**observer_free** (**Observer**< **claw** > *observer) const de-allocate an observer More...
void	**prim_to_all** () compute all the variables from the primitives on the grid More...
void	**cons_to_all** () compute all the variables from the conservatives on the grid More...

CLawSolver

Methods II

template<policy::direction_t dir>

CCTK_REAL **diff** (CCTK_REAL const *grid_function, int i, int j, int k) const

"virtual" method to compute the derivative of a grid function in a point More...

template<policy::direction_t dir>

CCTK_REAL **wdiff** (CCTK_REAL const *grid_function, int i, int j, int k) const

"virtual" method to compute the weak derivative of a grid function in a point More...

template<policy::direction_t dir>

CCTK_REAL **jump** (CCTK_REAL const *grid_function, int i, int j, int k) const

"virtual" method to compute the weighted value of the jump of a given grid function in a point More...

void **reset_rhs** ()

set the RHS to zero More...

void **compute_rhs** ()

"virtual" method to compute the RHS of the equations More...

DG specific stuff — not used

Calls into the specific solver, computes RHS for MoL

Linear advection example

traits

$$\partial_t \phi + \nabla \cdot (\phi \mathbf{v}) = 0$$

```
class LinearAdvection;

namespace hrscc {

template<>
class traits<LinearAdvection> {
    public:
        enum {nequations = 1};
        enum {nexternal = 3};
        enum {nbitmasks = 0};
        static bool const pure = true;
};
```

ϕ

v^i

adv_traits.hh

The source term is zero

`traits<LinearAdvection>` provide CLaw metadata

Linear advection example

CLaw I

```
class LinearAdvection: public hrscc::CLaw<LinearAdvection> {
    public:
        typedef hrscc::CLaw<LinearAdvection> claw;

        LinearAdvection();

        inline void prim_to_all(
                hrscc::Observer<claw> & // observer
                ) const {}

        template<hrscc::policy::direction_t dir>
        inline void fluxes(
                hrscc::Observer<claw> & observer
                ) const {
            observer.flux[dir][0] = (*observer.field[dir])*
                    (*observer.primitive[0]);
        }

        template<hrscc::policy::direction_t dir>
        inline void eigenvalues(
                hrscc::Observer<claw> & observer
                ) const {
            observer.eigenvalue[0] = *observer.field[dir];
        }

        template<hrscc::policy::direction_t dir>
        inline void eig(
                hrscc::Observer<claw> & observer
                ) const {
            observer.eigenvalue[0] = *observer.field[dir];
            observer.left_eigenvector[0][0] = 1.0;
            observer.right_eigenvector[0][0] = 1.0;
        }
};
```

adv_claw.hh

$$\partial_t \phi + \nabla \cdot (\phi \mathbf{v}) = 0$$

- Linear advection equation toy
- THC_Core implements different CLaws for ideal-gas, tabulated
- Future physics, e.g., MHD, will be implemented as new CLaw objects
- THC_Core also defines Cactus scheduled functions that call into HRSCCore

Linear advection example

CLaw II

```
#include "cctk.h"
#include "cctk_Arguments.h"
#include "cctk_Parameters.h"

#include "hrscc_config.hh"
#include "utils.hh"

#include "adv_claw.hh"

namespace {

void rhs(cGH const * const cctkGH) {
    DECLARE_CCTK_PARAMETERS

    hrscc::compute_rhs<LinearAdvection>(cctkGH);
}

}

extern "C" void AdvectHRSC_RHS(CCTK_ARGUMENTS) {
    DECLARE_CCTK_PARAMETERS

    if(verbose) {
        CCTK_INFO("AdvectHRSC_RHS");
    }

    rhs(cctkGH);
}
```

adv_rhs.hh

- `hrscc::compute_rhs<CLaw>` wraps the creation and call of the appropriate `CLawSolver` depending on the `HRSCCore` parameters in the Cactus parfile

- Compilation is very slow, because we instantiate all templates: all combination of method, reconstruction, Riemann solver etc., for about ~250 schemes!

- We should revert back to compile time selection of numerical scheme

THC_Core

scheduler

Calls into HRSCCore

Only SYNC statements

```
if (CCTK_Equals(evolution_method, "THCode")) {
    SCHEDULE GROUP THC_SetExcisionMask IN HydroBase_Con2Prim
    {
    } "Thorns providing excision should set the bitmask"

    SCHEDULE THC_ConsToAll IN HydroBase_Con2Prim AFTER THC_SetExcisionMask
    {
        LANG: C
    } "Computes the primitives from the conservatives"

    SCHEDULE THC_InitSource IN HydroBase_RHS
    {
        LANG: C
    } "Initializes the source term"

    if(CCTK_Equals(physics, "GRHD")) {
        SCHEDULE THC_GRSource IN HydroBase_RHS AFTER THC_InitSource
        {
            LANG: C
        } "Computes the geometric source terms"
    }

    SCHEDULE THC_RHS IN HydroBase_RHS AFTER (THC_InitSource THC_GRSource)
    {
        LANG: C
    } "Compute the RHS for MoL"

    if(CCTK_Equals(physics, "GRHD")) {
        SCHEDULE THC_AddToTmunu IN AddToTmunu
        {
            LANG: C
        } "Adds the contribution of the fluid to the global stress-energy tensor"
    }

    SCHEDULE THC_SelectBC IN HydroBase_Select_Boundaries
    {
        LANG: C
        # This goes in SINGLEMAP, because cctk_nghostzones
        # is not defined in LEVEL mode
        OPTIONS: SINGLEMAP
        SYNC: dens
        SYNC: densxn
        SYNC: densxp
        SYNC: scon
        SYNC: tau
        SYNC: densgain
    } "Select boundary conditions"
}
```

Next steps

[1] D. Radice, *HRSCCore thorn documentation*, 2012

[2] D. Radice, L. Rezzolla, *THC: a new high-order finite-difference high-resolution shock-capturing code for special-relativistic hydrodynamics,* Astronomy & Astrophysics, 547, A26 (2012), 1206.6502

[3] D. Radice, L. Rezzolla, F. Galeazzi, *High-Order Fully General-Relativistic Hydrodynamics: new Approaches and Tests*, Class. Quantum Grav. 31 075012 (2014), 1312.5004

[4] D. Radice, F. Galeazzi J. Lippuner, L. F. Roberts, C. D. Ott, L. Rezzolla, *Dynamical Mass Ejection from Binary Neutron Star Mergers,* Monthly Notices of the Royal Astronomical Society 2016 460 (3): 3255-3271, 1601.02426

[5] D. Radice, A. Perego, K. Hotokezaka, S. A. Fromm, S. Bernuzzi, L. F. Roberts, *Binary Neutron Star Mergers: Mass Ejection, Electromagnetic Counterparts and Nucleosynthesis*, The Astrophysical Journal 869:130 (2018), 1809.11161