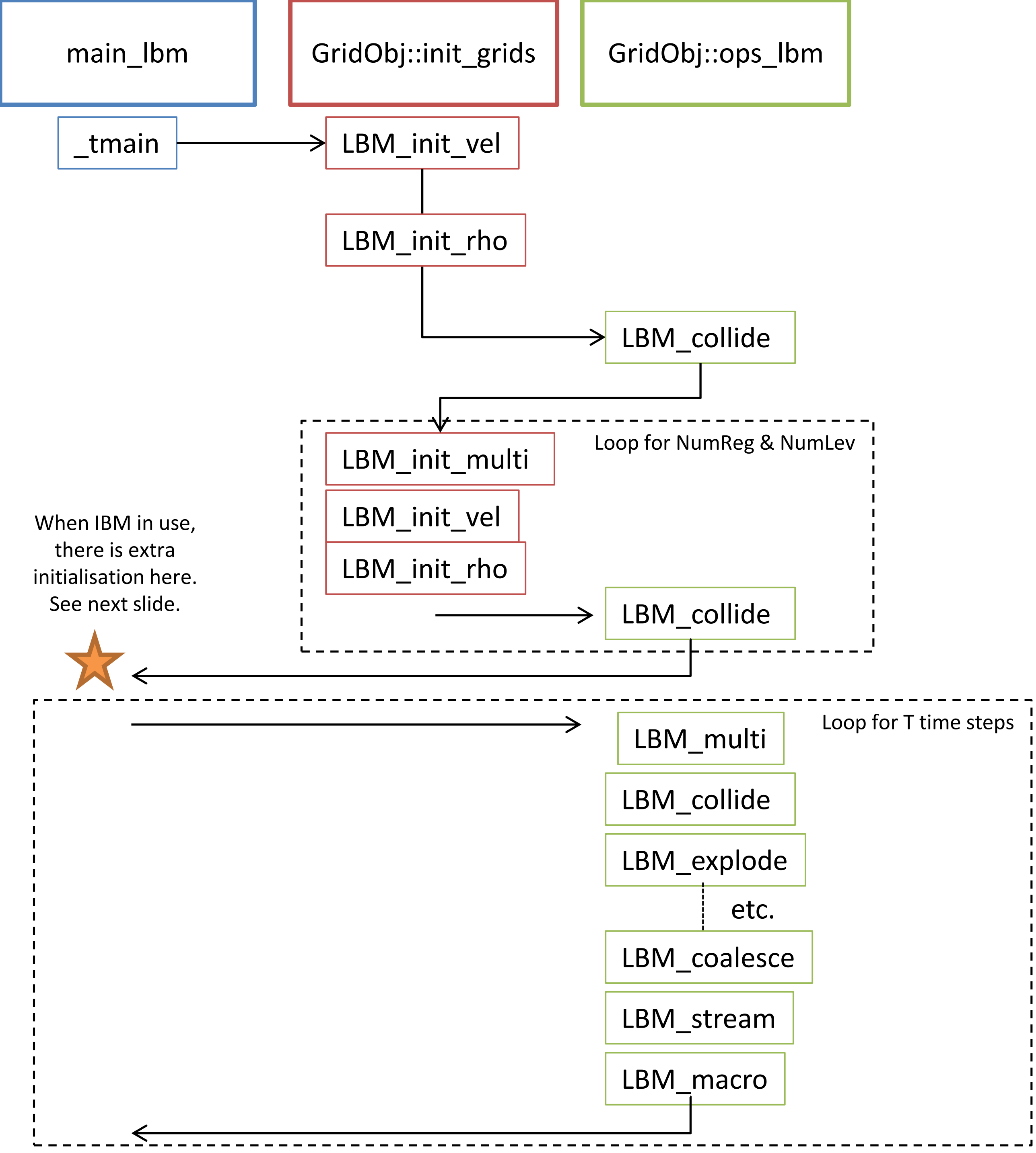


Description (IBM disabled)

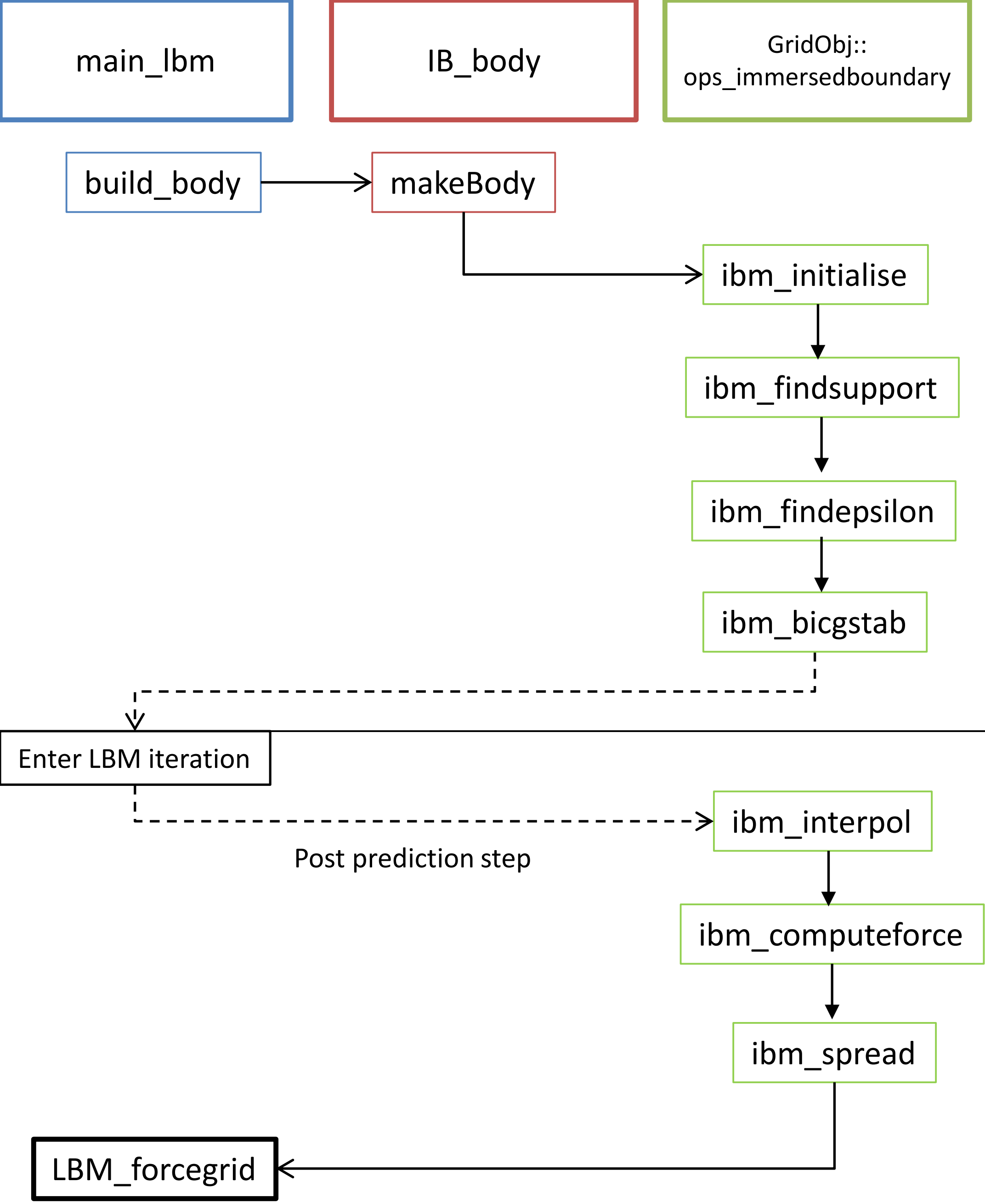
- Initialise Level 0 grid
- Initialise L0 velocity
- Initialise L0 density

- Initialise L0 f to feq using LBM_collide()

- Initialise refined grids (r = level)
- Initialise Lr velocity
- Initialise Lr density
- Initialise Lr f to feq

- Enter main kernel
  - Collide
  - Explode
  - etc.

  - Coalesce
  - Stream
  - Update Macroscopic



Description (IBM enabled)

- Makes a body by sequentially adding markers until body is completely built.
- Initialise markers by calling the following:
    - Find support of each marker and populate the marker properties appropriately
    - Find epsilon for each marker by solving ensuing partition of unity linear system.
    - System solved using biconjugate gradient method

- IBM requires saving of time step information to perform predictor step after which the following is executed:
    - Interpolate velocity
    - Compute force
    - Spread force

- As part of corrector step, LBM_forcegrid() is used to convert a Cartesian force to forces on each of the lattice directions.

```
class GridObj

std::vector<int> XInd;
std::vector<int> YInd;
std::vector<int> ZInd;

std::vector<double> XPos;
std::vector<double> YPos;
std::vector<double> ZPos;

std::vector<double> f;
std::vector<double> feq;
std::vector<double> u;
std::vector<double> rho;

std::vector<int> LatTyp;

double omega;
double dx;
double dy;
double dz;
double dt;

size_t CoarseLimsX[2];
size_t CoarseLimsY[2];
size_t CoarseLimsZ[2];
```

This class is instantiated once and nests multiple levels of refined instances within itself. Code stores each refined region of nested sub-grids in an array of `GridObj` objects called `subGrids[ ]`. The `private` members are:

Vectors of indices of the lattice sites on the grid. On L0 these vectors identify which of the sites are refined.

Vectors of positions in a global reference frame of the lattice sites on the grid

Arrays of populations, and macroscopic quantities for every lattice site on the grid

Array of labels assigned to each lattice site upon initialisation which identifies whether it is to be operated on or passed over by operating subroutines.

Relaxation time (note this is `public`) and lattice site spacing for grid.

For sub-grids these are unsigned integers containing the limits of the coarse grid patch that the region refines.

class GridObj

```
// Initialisation functions
void LBM_init_vel();                  // Initialise the velocity field
void LBM_init_rho();                  // Initialise the density field
void LBM_init_grid();                 // Initialise top level grid with a velocity and denstiy field
void LBM_init_subgrid(double offsetX, double offsetY, double offsetZ, double dx0, double omega_coarse);
                                      // Initialise subgrid with all quantities

// LBM operations
void LBM_multi();                                 // Launch the multi-grid kernel
void LBM_collide(bool core_flag);     // Apply collision + 1 overload for just computing feq
double LBM_collide(int i, int j, int k, int v);
void LBM_stream();                                // Stream populations
void LBM_macro();                                 // Compute macroscopic quantities
void LBM_boundary(int bc_type_flag);              // Apply boundary conditions

// Multi-grid operations
void LBM_explode(int RegionNumber);   // Explode populations from coarse to fine
void LBM_coalesce(int RegionNumber);              // Coalesce populations from fine to coarse

// Potentially deprecated methods -- remove in future release
bool isEdge(size_t i, size_t j, size_t k, int RegionNumber);        // Check whether point is edge of subGrid[]
bool isWithin(size_t i, size_t j, size_t k, int RegionNumber);      // Check whether point lies within subGrid[]

// Add subgrid
void LBM_addSubGrid(int RegionNumber);            // Add and initialise subgrid structure for a given region number

// IO methods
void lbm_write3(int t);               // Writes out the contents of the class as well as any subgrids
// EnsightGold methods
void genCase(int nsteps, int saveEvery);          // Generate case file
void genGeo();                                    // Generate geometry file
void genVec(int fileNum);                         // Generate vectors file
void genScal(int fileNum);                        // Generate scalars file
```

```
class GridObj

// IBM objects
IB_body iBody                              // An immersed boundary object such as a cylinder

// IBM methods
void build_body(int type);                 // Build a new pre-fab body
void ibm_initialise();                     // Initialise a built immersed body with support
double ibm_deltakernel(double rad, double dilation);     // Evaluate kernel (delta function approximation)
void ibm_interpol();                       // Interpolation of velocity field
void ibm_spread();                         // Spreading of restoring force
void ibm_findsupport(unsigned int m);      // Populates support information for the m-th marker of a body
void ibm_computeforce();                   // Compute restorative force at each marker in a body
void ibm_findepsilon();                    // Method to find epsilon weighting parameter

// Biconjugate gradient stablised method for solving asymmetric linear system required by finding epsilon
double ibm_bicgstab(std::vector< std::vector<double> >& Amatrix, std::vector<double>& bVector, std::vector<double>& epsilon,
   double tolerance, unsigned int maxiterations);
```

```
class IB_body

// Objects
Std::vector<IB_marker> markers             // An array of marker objects each representing an individual Lagrange
                                           marker which makes up an immersed body
// IBM methods
void addMarker(double x, double y, double z, bool flex_rigid);       // Add Lagrange marker to the body

// Prefab body building methods

// Method to construct sphere/circle
void makeBody(double radius, std::vector<double> centre, bool flex_rigid);

// Method to construct cuboid/rectangle
void makeBody(std::vector<double> width_length_depth, std::vector<double> centre, bool flex_rigid);
```

```
class IB_marker

// General vectors
std::vector<double> position;     // Position vector of Largange marker location
std::vector<double> fluid_vel;    // Fluid velocity interpolated from lattice nodes
std::vector<double> desired_vel;  // Desired velocity
std::vector<double> force_xyz;    // Restorative force vector on marker

// Vector of indices for the Eulerian (fluid) nodes considered to be in support of marker
std::vector<unsigned int> supp_i;
std::vector<unsigned int> supp_j;
std::vector<unsigned int> supp_k;

std::vector<double> deltaval;     // Value of delta function for a given support node

// Scalars
bool flex_rigid;        // false == rigid/fixed; true == flexible/moving
double epsilon;         // Scaling parameter
double local_area;      // Area associated with support node (same for all points if from same grid and regularly spaced
                        like LBM)
double dilation;        // Dilation parameter (same in all directions for uniform Eulerian grid)
```