Data Management

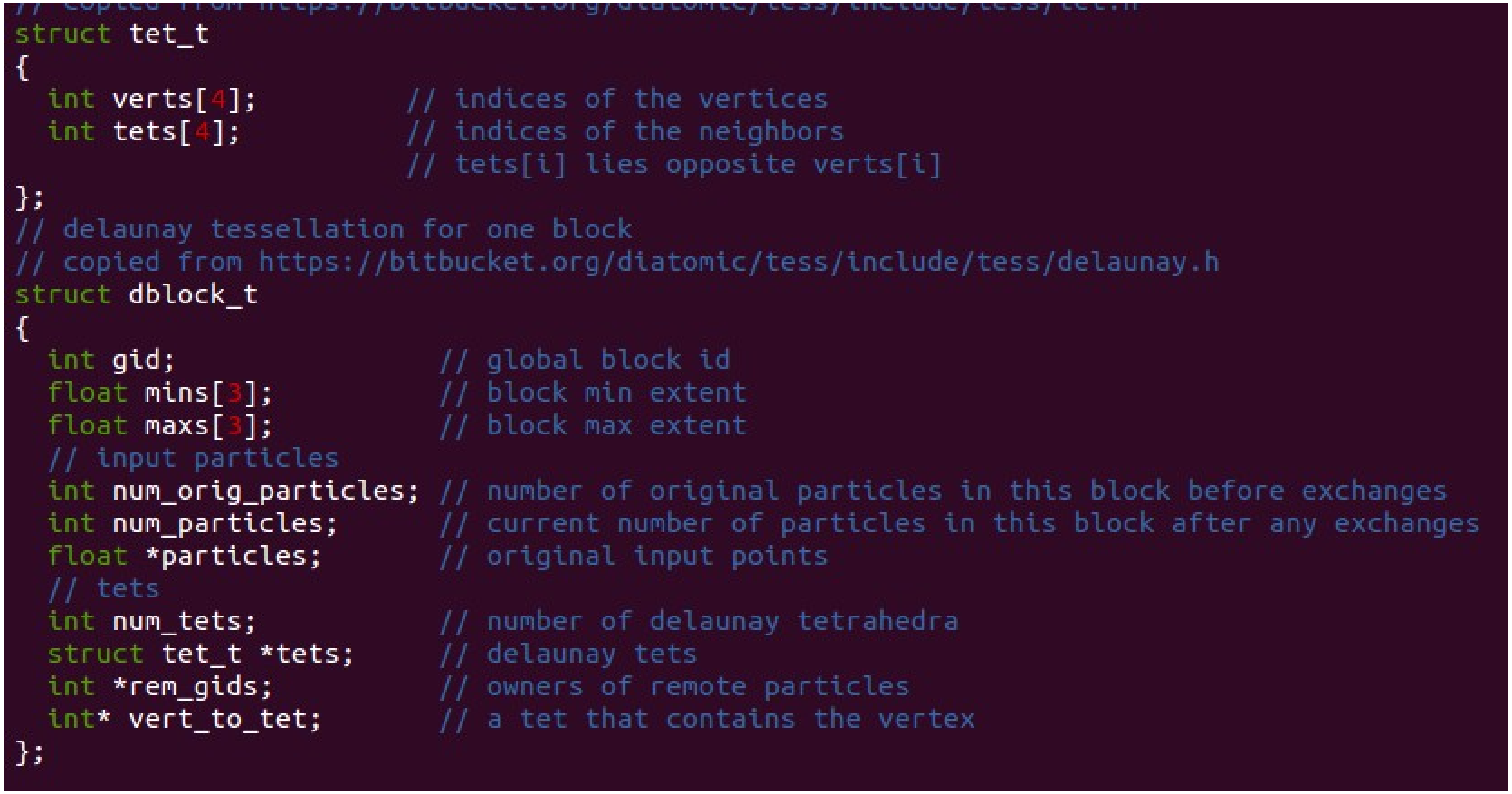
Goals

- Getting the data/Putting the data
  - User-friendly interface
  - Safe (or the least error-prone)
  - Zero-copy (?)
- Exposing the data for (extra) analysis plugins
  - In situ & In transit plugins
  - Provide synchronization primitives
- Splitting the data for implementing “Deep workflow” operations
  - Piplining, selection, redistribution (N-to-M)
  - Needs a chunking semantics
- I/O

Plan

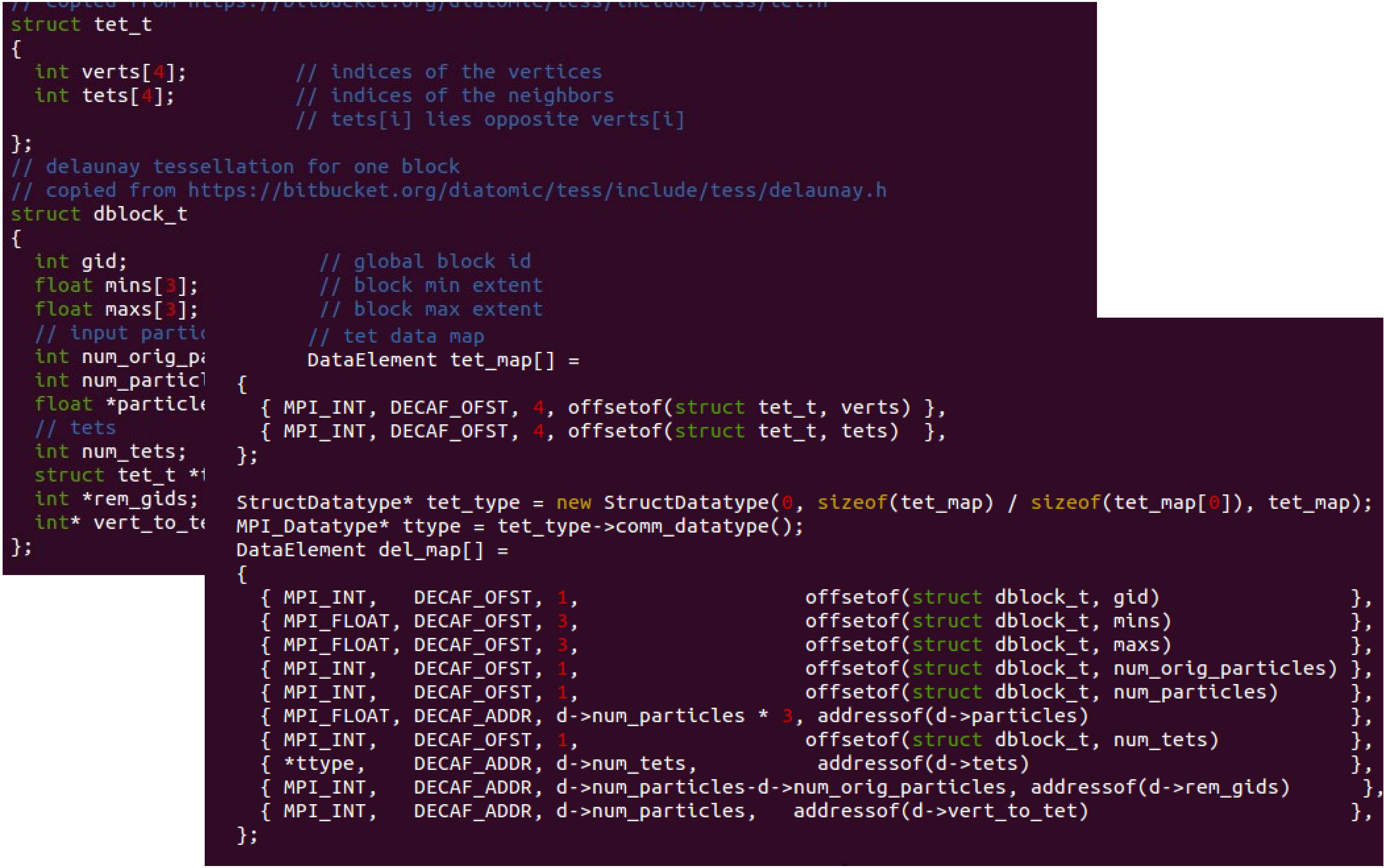
- Define internal format for data description
  - DIY format
  - Textual format (xml description, ...)
- Workflow part, we already have some primitives
  - N:M
  - Separate/overlapping sets of producers, dataflow, consumers

DIY-like format



```
// copied from https://bitbucket.org/diatomic/tess/include/tess/tet.h
struct tet_t
{
  int verts[4];         // indices of the vertices
  int tets[4];          // indices of the neighbors
                        // tets[i] lies opposite verts[i]
};
// delaunay tessellation for one block
// copied from https://bitbucket.org/diatomic/tess/include/tess/delaunay.h
struct dblock_t
{
  int gid;                // global block id
  float mins[3];          // block min extent
  float maxs[3];          // block max extent
  // input particles
  int num_orig_particles; // number of original particles in this block before exchanges
  int num_particles;      // current number of particles in this block after any exchanges
  float *particles;       // original input points
  // tets
  int num_tets;           // number of delaunay tetrahedra
  struct tet_t *tets;     // delaunay tets
  int *rem_gids;          // owners of remote particles
  int* vert_to_tet;       // a tet that contains the vertex
};
```

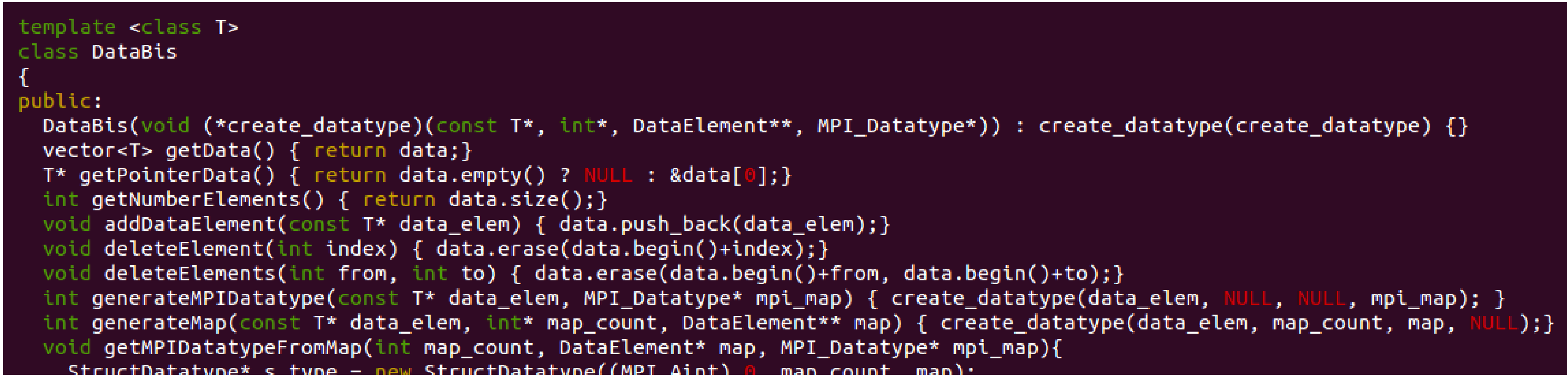
DIY-like format



```
// copied from https://bitbucket.org/diatomic/tess/include/tess/tet.h
struct tet_t
{
  int verts[4];          // indices of the vertices
  int tets[4];           // indices of the neighbors
                         // tets[i] lies opposite verts[i]
};
// delaunay tessellation for one block
// copied from https://bitbucket.org/diatomic/tess/include/tess/delaunay.h
struct dblock_t
{
  int gid;                 // global block id
  float mins[3];           // block min extent
  float maxs[3];           // block max extent
  // input partic
  int num_orig_pa
  int num_particl
  float *particle
  // tets
  int num_tets;
  struct tet_t *t
  int *rem_gids;
  int* vert_to_te
};
```

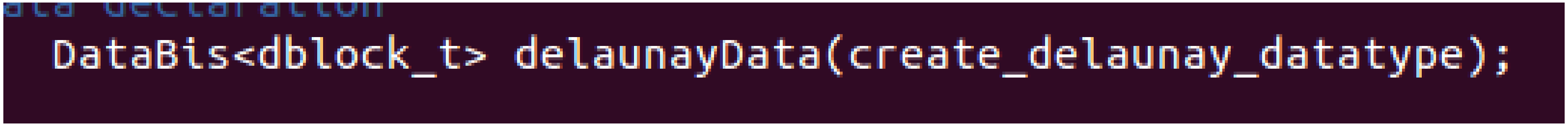
```
                        // tet data map
                        DataElement tet_map[] =
{
  { MPI_INT, DECAF_OFST, 4, offsetof(struct tet_t, verts) },
  { MPI_INT, DECAF_OFST, 4, offsetof(struct tet_t, tets)  },
};

StructDatatype* tet_type = new StructDatatype(0, sizeof(tet_map) / sizeof(tet_map[0]), tet_map);
MPI_Datatype* ttype = tet_type->comm_datatype();
DataElement del_map[] =
{
  { MPI_INT,   DECAF_OFST, 1,                    offsetof(struct dblock_t, gid)           },
  { MPI_FLOAT, DECAF_OFST, 3,                    offsetof(struct dblock_t, mins)          },
  { MPI_FLOAT, DECAF_OFST, 3,                    offsetof(struct dblock_t, maxs)          },
  { MPI_INT,   DECAF_OFST, 1,                    offsetof(struct dblock_t, num_orig_particles) },
  { MPI_INT,   DECAF_OFST, 1,                    offsetof(struct dblock_t, num_particles)     },
  { MPI_FLOAT, DECAF_ADDR, d->num_particles * 3, addressof(d->particles)                },
  { MPI_INT,   DECAF_OFST, 1,                    offsetof(struct dblock_t, num_tets)      },
  { *ttype,    DECAF_ADDR, d->num_tets,           addressof(d->tets)                     },
  { MPI_INT,   DECAF_ADDR, d->num_particles-d->num_orig_particles, addressof(d->rem_gids)      },
  { MPI_INT,   DECAF_ADDR, d->num_particles,    addressof(d->vert_to_tet)               },
};
```

DIY-like format - Data Class

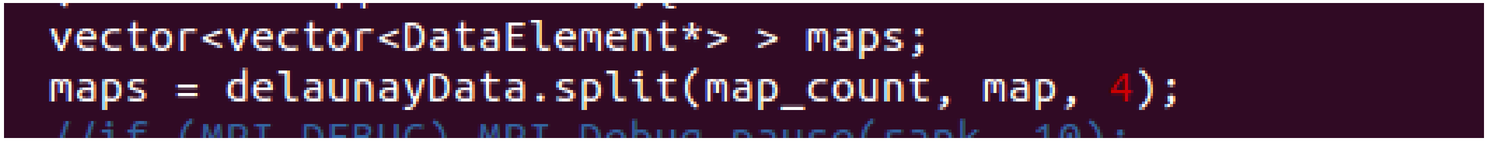


```
template <class T>
class DataBis
{
public:
  DataBis(void (*create_datatype)(const T*, int*, DataElement**, MPI_Datatype*)) : create_datatype(create_datatype) {}
  vector<T> getData() { return data;}
  T* getPointerData() { return data.empty() ? NULL : &data[0];}
  int getNumberElements() { return data.size();}
  void addDataElement(const T* data_elem) { data.push_back(data_elem);}
  void deleteElement(int index) { data.erase(data.begin()+index);}
  void deleteElements(int from, int to) { data.erase(data.begin()+from, data.begin()+to);}
  int generateMPIDatatype(const T* data_elem, MPI_Datatype* mpi_map) { create_datatype(data_elem, NULL, NULL, mpi_map); }
  int generateMap(const T* data_elem, int* map_count, DataElement** map) { create_datatype(data_elem, map_count, map, NULL);}
  void getMPIDatatypeFromMap(int map_count, DataElement* map, MPI_Datatype* mpi_map){
    StructDatatype* s_type = new StructDatatype((MPI_Aint) 0, map_count, map);
```

Using the Data Class



```
ata declaration
  DataBis<dblock_t> delaunayData(create_delaunay_datatype);
```



```
  vector<vector<DataElement*> > maps;
  maps = delaunayData.split(map_count, map, 4);
  //if (MPI_DEBUG) MPI_Debug_pause(rank, 10);
```