Hacked user finite element

Dimension finite element (0d,1d,2d,3d) and derived special entity elements, e.g. flat prism

Generic finite element

Finite element data
access by operator interface
- e.g. integration quadrature, Jacobian, …

Data structures on element sub-entities access by argument of overload doWork operator method.
- e.g. field data, DOFs indices, orientation, …

set integration rule

calculate base

…

Low level hacks (alters default element procedures by overloaded virtual methods)

DataOp operator N

…

DataOp operator 1

Sequence of DataOperators
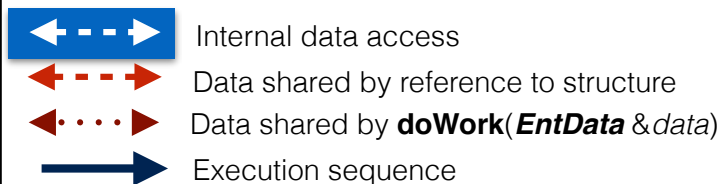
Sequence of UsersDataOperators

UserDataOp operator 1

…

UserDataOp operator N

Loop over entity on entities

UserDataStr for shared data

Legend

Internal data access

Data shared by reference to structure

Data shared by **doWork**(***EntData*** &*data*)

Execution sequence

In majority of applications user implements **UserDataOprators** and **UserDataStructure**

Type: OPROW or OPCOL (indices on rows or on columns)

Constructor:
```
UserDataOperator(const std::string &field_name, const char type)
```

UserDataOp — Loop over entity on entities
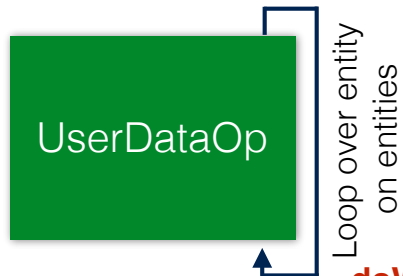
```
entities_set = { Vertices, Edge0, .., Edge5,
Face0, .., Face3, Volume }
for(o in operator_sequence)
  for(e in entities_set)
    o.doWork(side[e],type[e],ent_data[e])
```

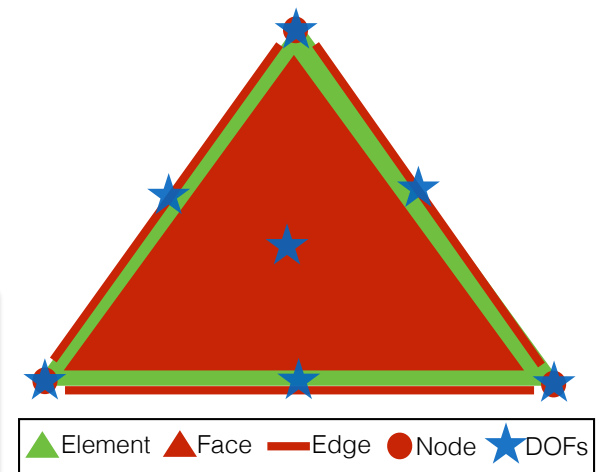**doWork** is overloaded method by user (loop is implicitly called by element)

Type: OPROWCOL (indices on rows & on columns)

Constructor:
```
UserDataOperator(const std::string &row_field_name,
const std::string &col_field_name, const char type,
const bool symm=true)
```

UserDataOp — Loop over entity on entities pairs
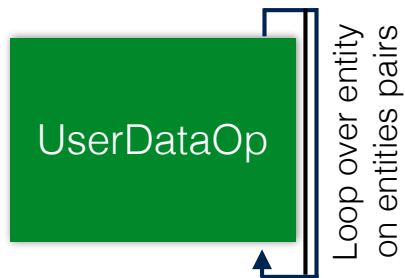
```
entities_pair_set = { {Vertices, Vertices},
{Vertices,Edge0}, .., { Volume, Volume} }
for(o in operator_sequence)
 for(e in entities_pair_set)
  o.doWork(
  row_side[e.f],row_type[e.f],row_ent_data[e.f],
  col_side[e.s],col_type[e.s],col_ent_data[e.s]
  )
```

doWork is overloaded method by user (loop is implicitly called by element)
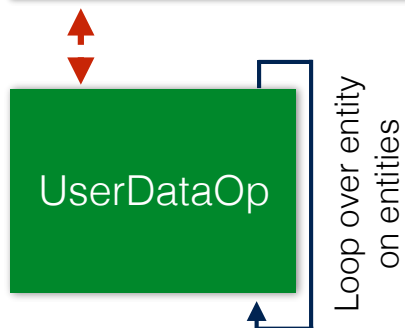
For square matrices & symmetric finite element *OPROW* & *OPCOL* are equivalent.
For *OPROWCOL*, when *symm = true*, only unique pairs are processed. It is third kind of operator, which not loop on entities of particular field, but entities of space, e.g. used to apply transformation to base functions. You can as well set *type = OPROW|OPROWCOL.*

△ Element ▲ Face ━ Edge ● Node ★ DOFs

Tetrahedral has base & DOFs on entities.
By space:
- **Space H1:** Vertices, 6 Edges, 4 Faces (Tri, Quad), 1 Volume
- **Space H-Curl:** 6 Edges,4 Faces, 1 Volume
- **Space H-Div:** 4 Faces, 1 Volume
- **Space L2:** 1 Volume (Tet, Prism, Hex, Wedge, …)

By order:
- **H1 order 1:** Only on Vertices
- **H1 order 2:** Vertices and Edges
- **H1 order 3:** Vertices, Edges & Faces
- **H1 order 4 and more:** Verices, Egdes, Faces and Volume

In similar way for other approximation spaces.

EntData:
- Values at DOFs
- Global/Local indices of DOFs
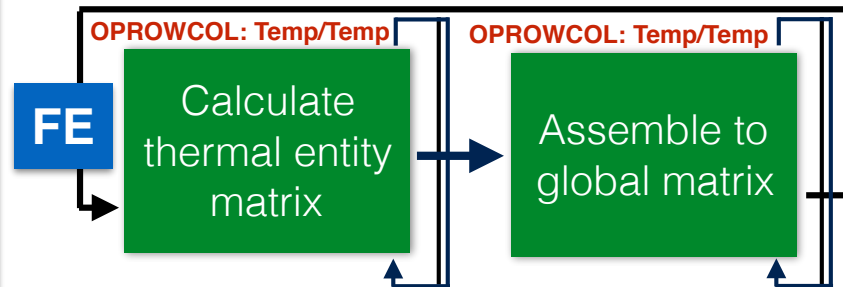- Base/Space/Order/Sense
- Base functions & more

UserDataStr
- Temperature at Gauss pts.
- Stresses at Gauss pts.
- Strains at Gauss pts.
- Local entity matrix (Lhs)
- Local entity vector (Rhs)
- Global thermal matrix
- Global elastic matrix
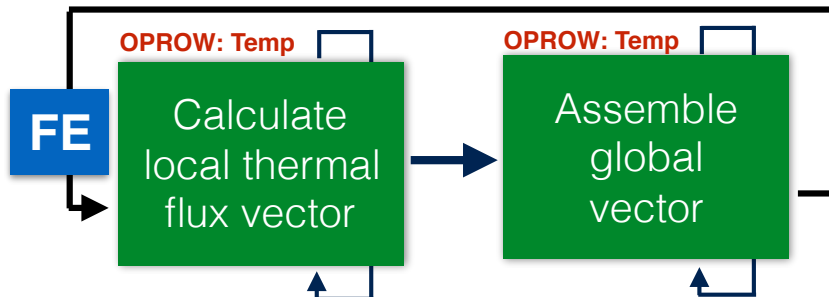- Global thermal vector
- Global elastic vector

UserDataOp

Loop over entity on entities

Thermal element

1) Loop body domain volume entities

OPROWCOL: Temp/Temp

FE Calculate thermal entity matrix

OPROWCOL: Temp/Temp

Assemble to global matrix

2) Loop Neumann (boundary) surface entities

OPROW: Temp

FE Calculate local thermal flux vector
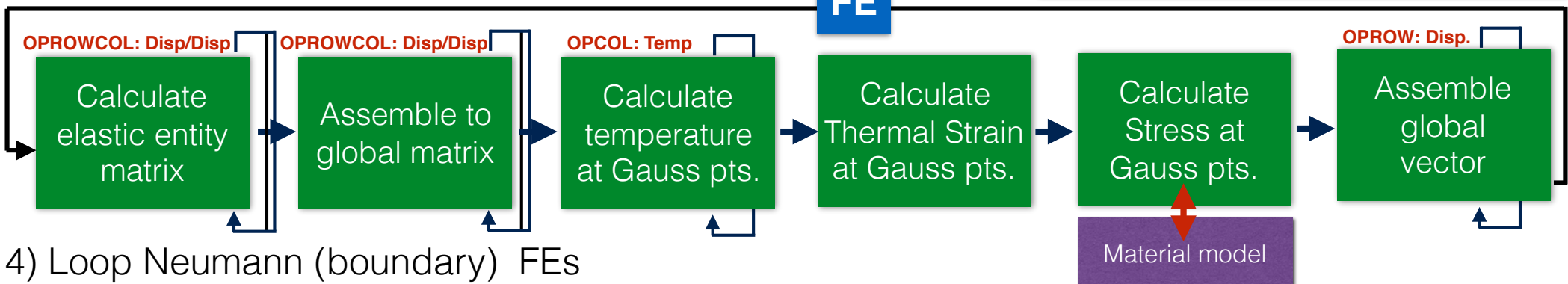
OPROW: Temp
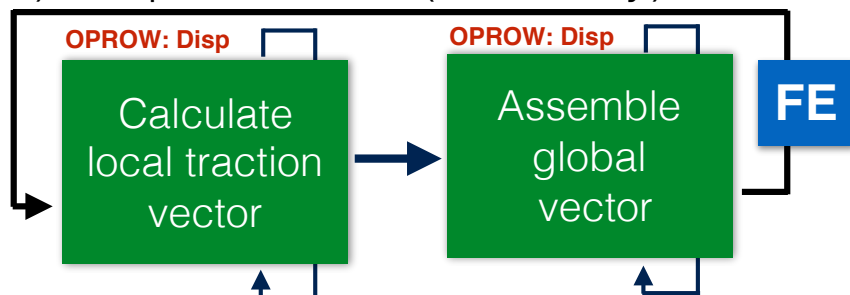
Assemble global vector

Notes:
- Operators can be implemented such that works for 2d/3d problems
- The same operator can be used in several different contexts, e.g. assembly of vector
- Unit (atom) tests can be written for each operator independently
- Each problem, thermal, elastic & thermo-elastic can be tested&run, by adding/removing operators
- Developer focus attention on operator implementation
- Loops, indexing, etc. are managed by MoFEM
- Each field is declared independently by space (H1, Hcurl,Hdiv, L2) & base (Legendre, Jacobi, …), on heterogenous and arbitrary approximation order
- Local and global indices on entity are passed by **EntData** as reference to overloaded **doWork** method in operator.

Elastic element

3) Loop body domain volume entities

FE

OPROWCOL: Disp/Disp

Calculate elastic entity matrix

OPROWCOL: Disp/Disp

Assemble to global matrix

OPCOL: Temp

Calculate temperature at Gauss pts.

Calculate Thermal Strain at Gauss pts.

Calculate Stress at Gauss pts.

Material model

OPROW: Disp.

Assemble global vector

4) Loop Neumann (boundary) FEs

OPROW: Disp

Calculate local traction vector

OPROW: Disp

Assemble global vector

FE

- Base functions and derivatives are evaluated by element and easy accessed from users operator from **EntData**
- **Note that implementation of UserOperator is for entities on element, not element itself.**
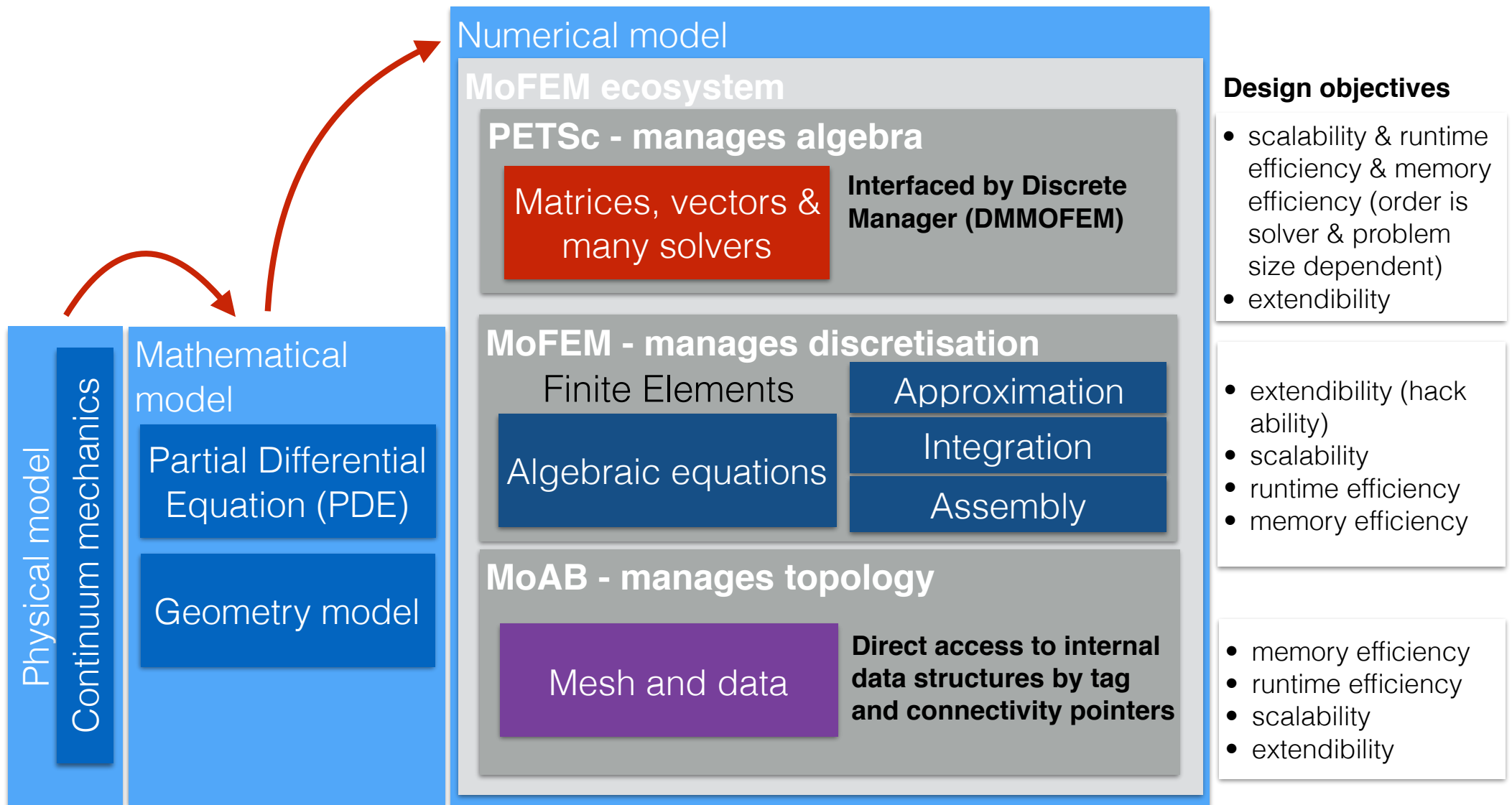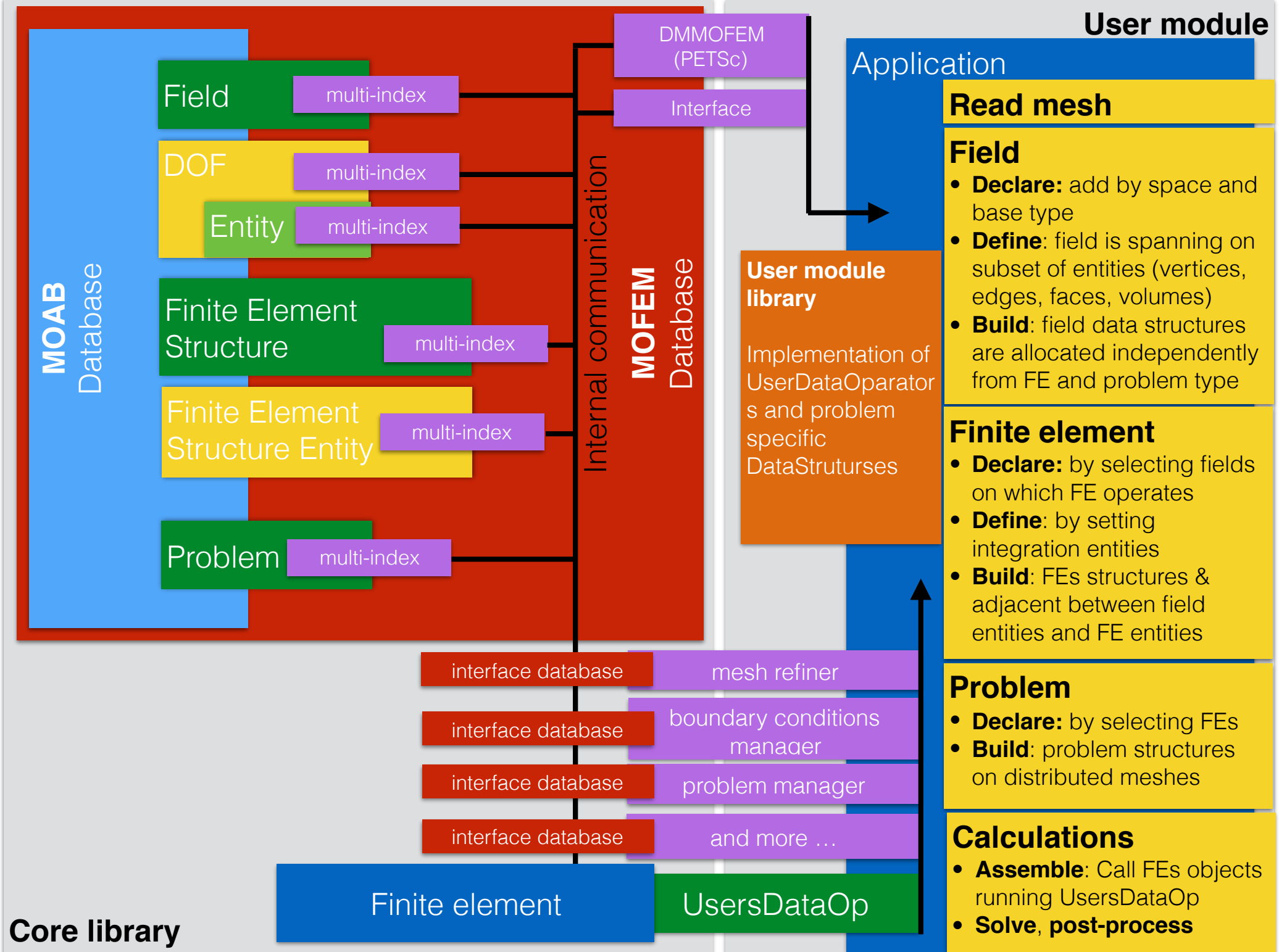
MoAB Database:
- Mesh (connectivity & adjacency)
- Tags on mesh

RefEntity (dense tag storage):
- **BitRefinmentLevel**

Field meshset:
- **name of the field**
- **ID of space (H1,Hdiv,etc.)**
- **ID of approximation base (Legendre,etc.)**
- **Coeoffincnts number (**rank**)**
- **Coordinate systems (**reference and current base**)**

FieldEntity (sparse or dense storage):
- **field order of approx.**
- **filed DOFs values**

ptr to moab tags

Field multi-index

Shared pointer to container of field structures (not many of those).
In structure:
- ID & pointers to internal MOAB tags storage
- sequences (vectors) of field entities/dofs structures

FieldEntity seq. 0	**FieldEntity** seq. 1	
..	**FieldEntity** Seq. N	
DOFs Seq. 0	..	
DOFs Seq. 2		

interface<PTR>

inheritance by pointer in interface

FieldEntity multi-index

Aliased shared pointer to element of sequence container of FieldEntities.
- ID (owner proc | EntityHandle | Field ID)
- sequence to dofs on entity (wihch are not in Field data structure)
- approx. order & tag ptr. to field data on mesh

DOFs sequence

interface<PTR>

inheritance by pointer in interface

Aliased share pointer to sequence

```
FieldEntity by aliased shared pointer:
shared_ptr<vector<FieldEntity> > seq0;
entity_n = shared_ptr<FieldEntity>(&seq0[n],seq0);

FieldEntity by aliased shared pointer:
shared_ptr<vector<Dof> > seq0;
dof_n = shared_ptr<Dof>(&seq0[n],seq0);
```

Dof multi-index

Aliased shared pointer to container of dofs structures (large number of those).
In structure:
- ID (dof number on entity | UId of FieldEntity)
- Shared pointer to **FieldEntity Interface**

interface<PTR>

* Vector of FieldEntities/Dofs is destroyed when all elements in sequence are destroyed. Memory is allocated in sequences (blocks) to minimalist set-up/build database time.