

---

# feelfem

A repository system  
for the finite element method

Hidehiro Fujio

NEC Corporation Japan

# Outline

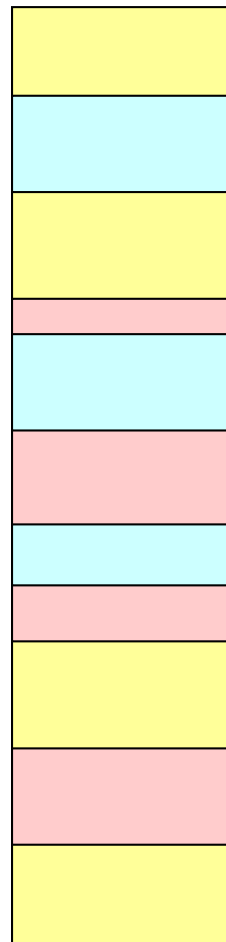
---

- Problems in making a rapid prototype parallel software
- Software reuse by code generator approach
- FEM discretizer component
- Pseudo-code representation of numerical algorithms
- Instance of pseudo code by layered C++ virtual function
- Conclusion

# 3 factors in one scientific HPC program

---

A scientific HPC program is composed of 3 factors



Mathematical  
discretization

- Fluid flow problem
- Heat transfer problem
- Chemical reaction problem

Numerical  
algorithms

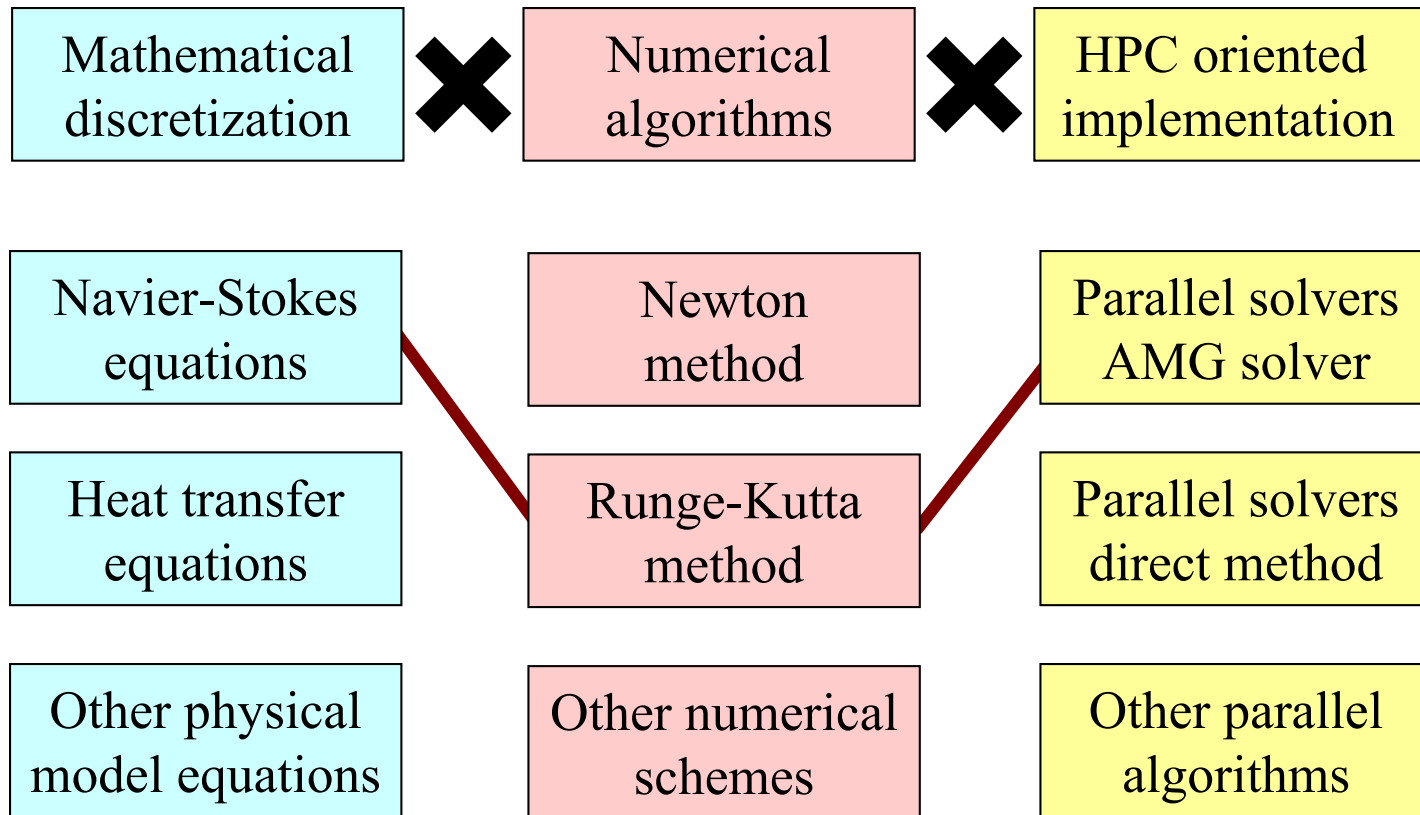
- Newton method
- Runge-Kutta algorithm
- Penalty method

HPC oriented  
implementation

- MPI programming
- Parallel solvers
- Parallel pre/post processing

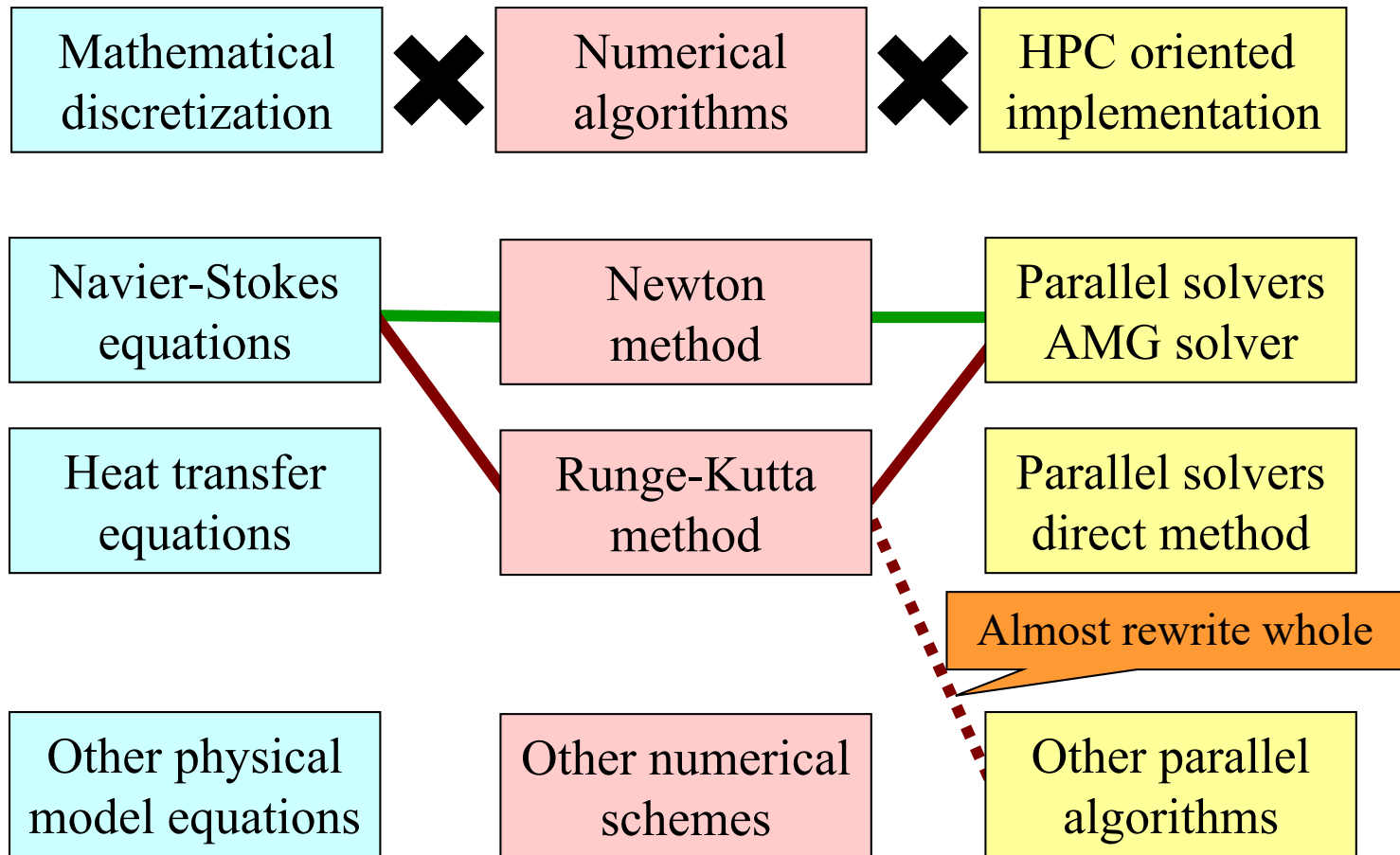
# 3 factors in scientific computing program

Before implementation, every components are independent.



After implemented, every components are tightly coupled.

# Problems of scientific software development



# The basic idea of the feelfem system

---

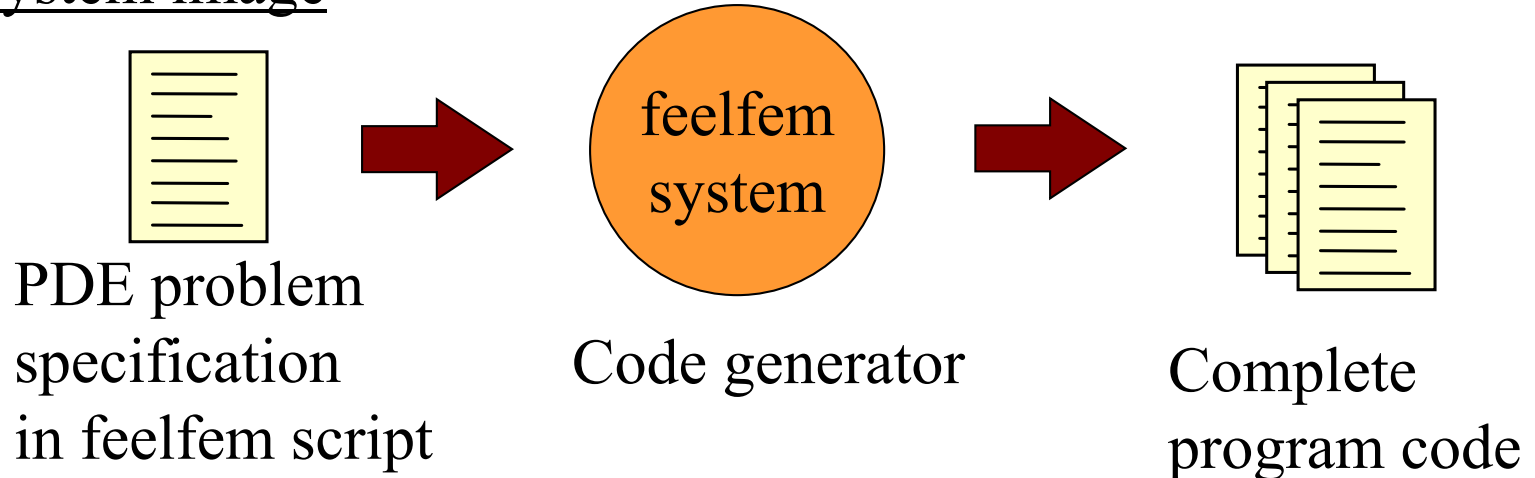
Mathematical  
discretization

Numerical  
algorithms

HPC oriented  
implementation

- Build a repository for each 3 components
- Reuse these software repositories by code generation
- Provide a very high level user input interface

## System image



# Repository mechanism

---

Mathematical  
discretization

Numerical  
algorithms

HPC oriented  
implementation

The FEM  
is adopted

Flexibility  
Applicability  
Boundary condition

Problem expressed  
in variational form

Pseudo code  
representation  
is adopted

All numerical  
schemes are  
expressed by  
pseudo code

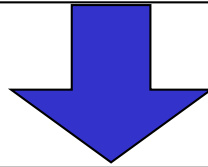
Pseudo code -  
real instance

Implemented  
C++ virtual  
code generator  
functions

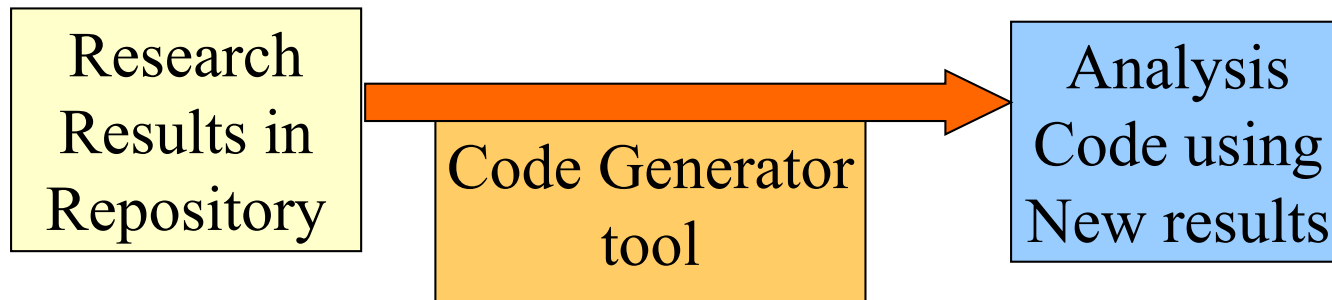
# Why code generator?

---

New Parallel/Distributed computing results are generally **provided as source codes**



To reuse these research results easily, one realistic solution is to make a software repository and generate codes.



# Repository mechanism 1 (Discretization)

---

Mathematical discretization	Numerical algorithms	HPC oriented implementation
The FEM is adopted  Flexibility Applicability Boundary condition  Problem expressed in variational form	Pseudo code representation is adopted  All numerical schemes are expressed by pseudo code	Pseudo code - real instance  Implemented C++ virtual code generator functions

# FEM problem specification

- Problems are expressed in variational formulation (both partial differential equation and boundary condition)

$$a(u, v) \rightarrow \int \left( \frac{\partial u}{\partial x} \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \frac{\partial v}{\partial y} \right) d\Omega \rightarrow$$

$$\text{integral}(\text{dx}(u) * \text{dx}(v) + \text{dy}(u) * \text{dy}(v))$$

$$b(g, v) \rightarrow \oint (l \cdot n_x + m \cdot n_y) v d\gamma \rightarrow$$

$$\text{bintegral}(g * v), g = l * n_x + m * n_y;$$

- Mathematical program generator is mostly consists of generation of programs for numerical integration from various **parameters**. (element, quadrature, continuity type, etc.)

# FEM calculation parameters

```
var {
```

```
    fem u[T2];
```

```
}
```

Element to use

```
quadrature tet5[tetra] {
```

```
    double m= 1.0/4.0;double mw=8/405;
```

```
    ...
```

```
    (m,m,m) : mw;
```

```
    ...
```

```
}
```

Numerical  
integration

```
element T2[tetra] {
```

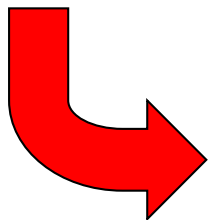
```
    (0,0,0):(1-r-t-s)*(2*(1-r-t-s)-1);
```

```
}
```

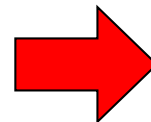
Interpolation  
functions (element)

```
p = integral[tet5] (u*u+dx(u)*dx(u));
```

Integrands



Mathematical  
Discretization  
Program generator  
Engine



FEM  
Mathematical  
Code  
(Language independent)

# Sample feelfem script (3D linear structural problem)

```

geom {          /* interface for GiD mesh generator */
  mesher gid;
  problem stress3d;
  dimension 3;
  surface (fix,u_fix,v_fix,w_fix); surface (press);
  region (mat);
}

/* definition of quadrature */
quadrature tet3[tetra] { /* Cubic order */
  double p2,p4,p6;
  double w,wm;
  p2 = 1.0 / 2.0; p4 = 1.0 / 4.0;
  p6 = 1.0 / 6.0;
  w = 9.0 / 20.0 / 6.0; wm = -4.0 / 5.0 / 6.0;
  (p4,p4,p4) : wm;
  (p2,p6,p6) : w;
  (p6,p2,p6) : w;
  (p6,p6,p2) : w;
  (p6,p6,p6) : w;
}
element T2[tetra] {
  (0,0,0): (1.0-r-t-s)*(2.0*(1.0-r-t-s) - 1.0);
  (1,0,0): r*(2.0*r - 1.0);
  (0,1,0): s*(2.0*s - 1.0);
  (0,0,1): t*(2.0*t - 1.0);
  (0.5,0,0) : 4.0*r*(1.0-r-t-s);
  (0.5,0.5,0) : 4.0 * r * s ;
  (0,0.5,0) : 4.0 * s * (1.0-r-t-s);
  (0,0,0.5) : 4.0 * t * (1.0-r-t-s);
  (0.5,0,0.5): 4.0 * r * t;
  (0,0.5,0.5): 4.0 * s * t;
}

var {
  fem u[T2],v[T2],w[T2];
  fem fx[T2],fy[T2],fz[T2];
  double c11,c12,c13;
  double c21,c22,c23;
  double c31,c32,c33;
  double c44,c55,c66;
  ewise e[tet3],nyu[tet3];
  double p; /* pressure */
  double c;
}

scheme {

  ProgramModel feelfem90mpi;

  fx = 0.0; fy = 0.0; fz = 0.0;
  e = 200000.0;
  p = 2000.0;
  nyu = 0.3;
  c = e*(1.0-nyu)/(1.0+nyu)/(1.0-2.0*nyu);
  c11 = c * 1.0;
  c12 = c * nyu / (1.0-nyu); c13 = c * nyu / (1.0-nyu);
  c21 = c * nyu / (1.0-nyu); c22 = c * 1.0;
  c23 = c * nyu / (1.0-nyu); c31 = c * nyu / (1.0-nyu);
  c32 = c * nyu / (1.0-nyu); c33 = c * 1.0;
  c44 = c * (1.0 - 2.0*nyu)/2.0 / (1.0 - nyu);
  c55 = c * (1.0 - 2.0*nyu)/2.0 / (1.0 - nyu);
  c66 = c * (1.0 - 2.0*nyu)/2.0 / (1.0 - nyu);

  solve[u,v,w; tu,tv,tw] {
    quadrature tet3;
    solver MUMPS;

    weq: integral( (c11*dx(u)+c12*dy(v)+c13*dz(w))*dx(tu)) +
      integral( c44*(dy(u)+dx(v)) *dy(tu)) +
      integral( c66*(dz(u)+dx(w)) *dz(tu)) =
      integral( fx * tu) + bintegral( Sx * tu);

    weq: integral( (c21*dx(u)+c22*dy(v)+c23*dz(w))*dy(tv)) +
      integral( c44*(dy(u)+dx(v)) *dx(tv)) +
      integral( c55*(dz(v)+dy(w)) *dz(tv)) =
      integral( fy * tv) + bintegral( Sy * tv);

    weq: integral( (c31*dx(u)+c32*dy(v)+c33*dz(w))*dz(tw)) +
      integral( c55*(dz(v)+dy(w)) *dy(tw)) +
      integral( c66*(dz(u)+dx(w)) *dx(tw)) =
      integral( fz * tw) + bintegral( Sz * tw);

    dbc: u = 0, on fix,u_fix,uv_fix,wu_fix;
    dbc: v = 0, on fix,v_fix,vw_fix,uv_fix;
    dbc: w = 0, on fix,w_fix,wu_fix,vw_fix;

    nbc: Sx = nx*p,Sy = ny*p, Sz = nz* p, on press;
  }
  output[u,v,w](label="displacement");
}

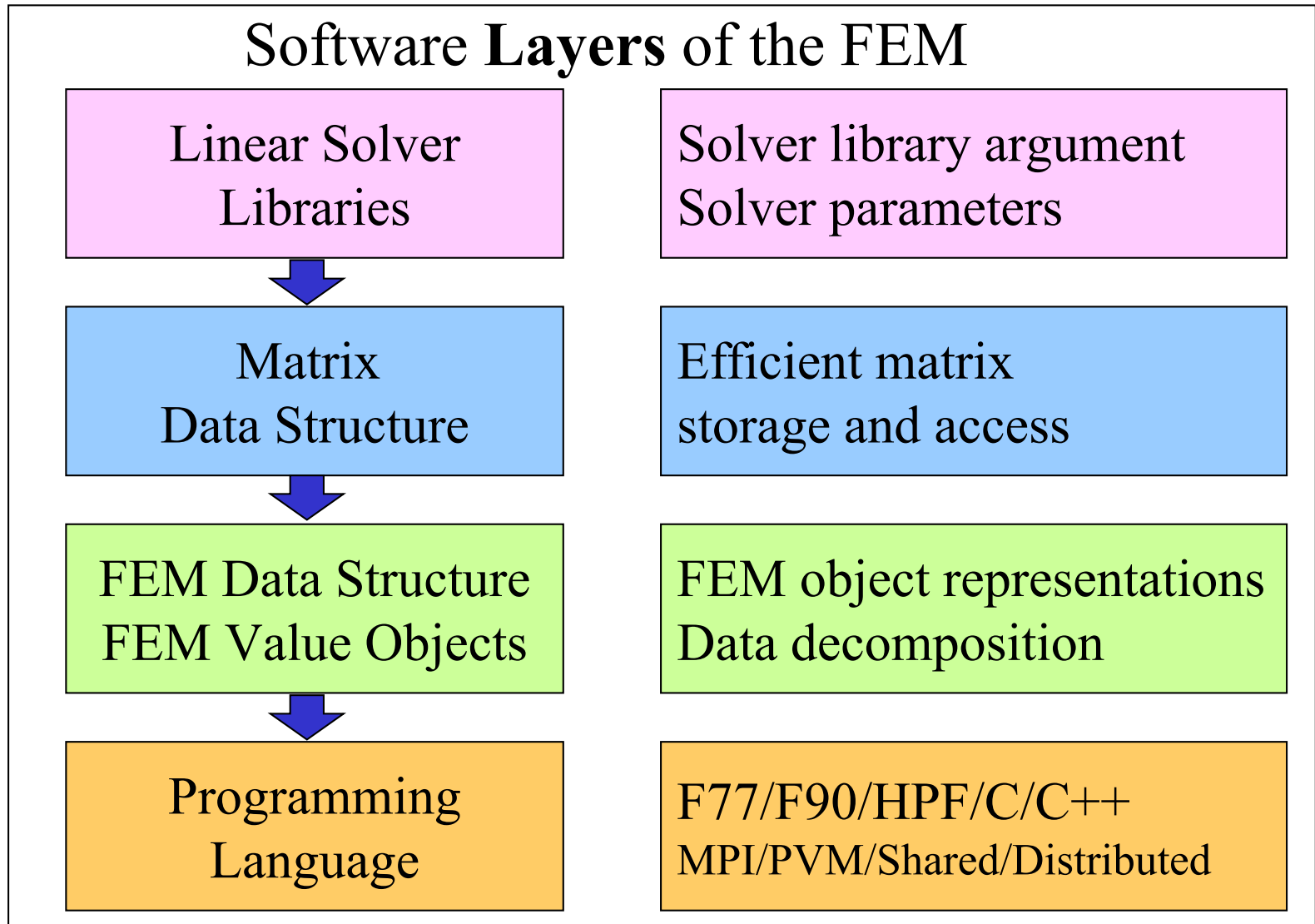
```

# Repository mechanism 2 (Algorithms)

---

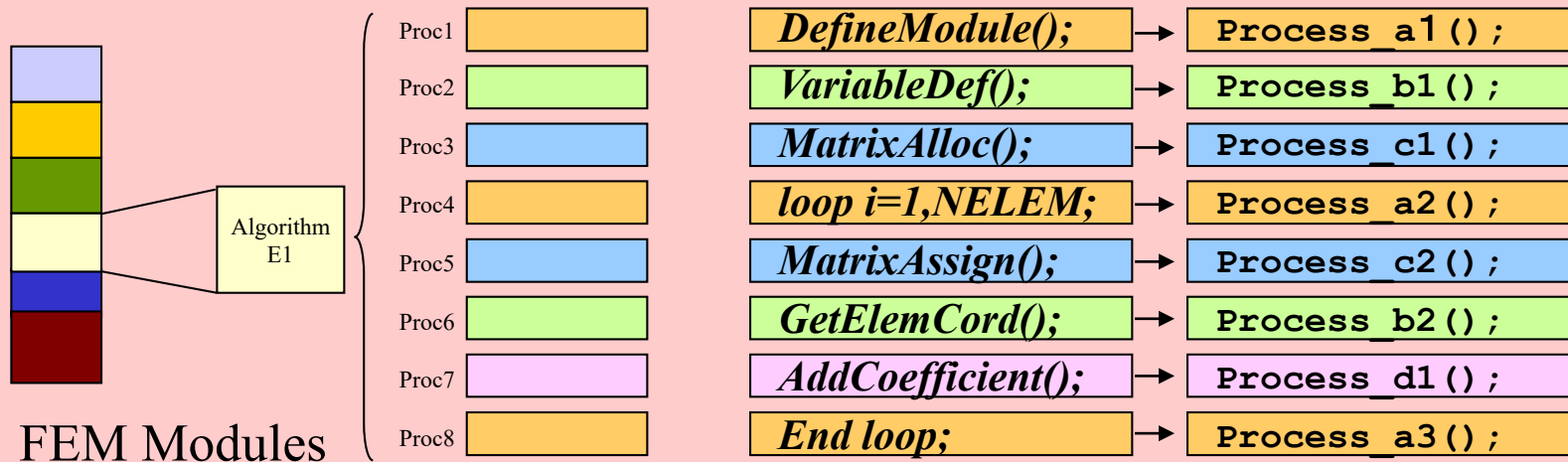
Mathematical discretization	Numerical algorithms	HPC oriented implementation
The FEM is adopted  Flexibility Applicability Boundary condition  Problem expressed in variational form	Pseudo code representation is adopted  All numerical schemes are expressed by pseudo code	Pseudo code - real instance  Implemented C++ virtual code generator functions

# Pseudo code layer structure



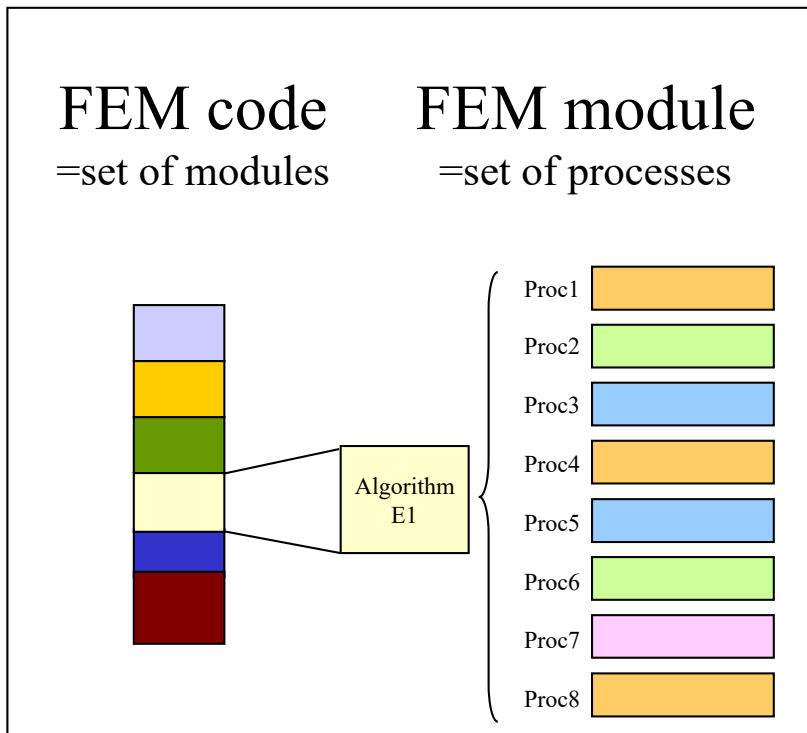
# Algorithm repository using pseudo code

- The FEM algorithm has modular structure in itself.
- Algorithms of each FEM modules are represented by pseudo code, and mathematical components are imbedded.



# Object oriented design of code generator 1

**Separation** of algorithm from implementation  
by **virtual code generator function**.



*Algorithm expressed  
in pseudocode*

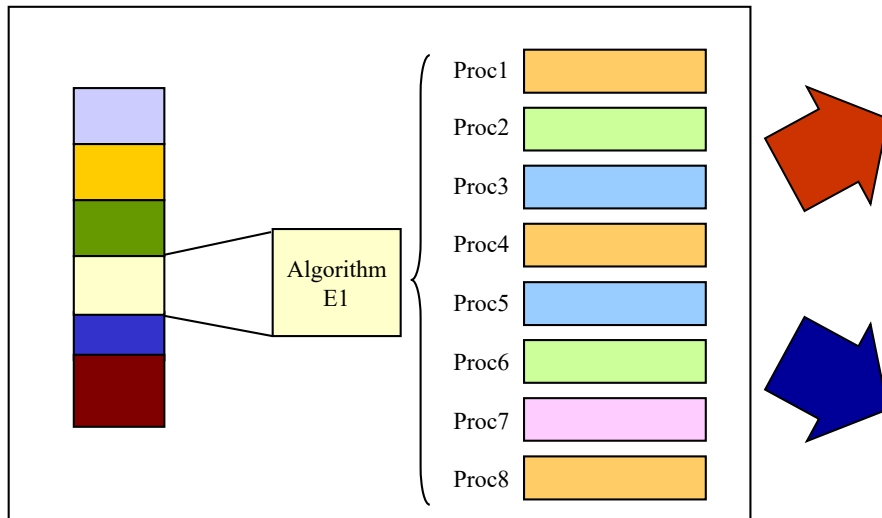
*Virtual code  
generator function*

<b>DefineModule();</b>	→	<b>Process_a1 ();</b>
<b>VariableDef();</b>	→	<b>Process_b1 ();</b>
<b>MatrixAlloc();</b>	→	<b>Process_c1 ();</b>
<b>loop i=1,NELEM;</b>	→	<b>Process_a2 ();</b>
<b>MatrixAssign();</b>	→	<b>Process_c2 ();</b>
<b>GetElemCord();</b>	→	<b>Process_b2 ();</b>
<b>AddCoefficient();</b>	→	<b>Process_d1 ();</b>
<b>End loop;</b>	→	<b>Process_a3 ();</b>

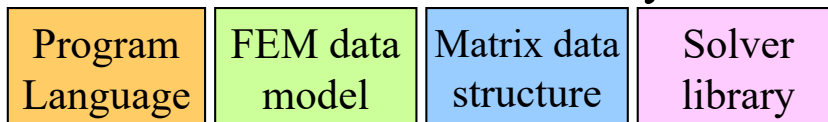
# Object oriented design of code generator 2

**Separation** of algorithm and implementation.

FEM algorithms are expressed by succession of virtual code generator function (pseudocode).



Virtual code generator functions are classified into the 4 layers.

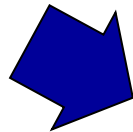
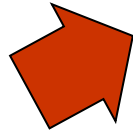
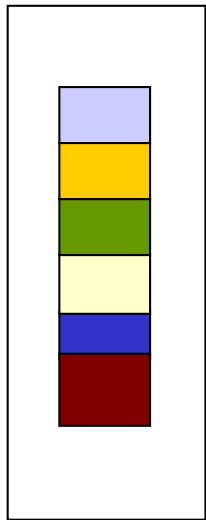


Algorithm Expression
<pre>Module_E1_generator() { Process_a1();   Process_b1();   Process_c1();   Process_a2();   Process_c2();   Process_b2();   Process_d1();   Process_a3(); }</pre>

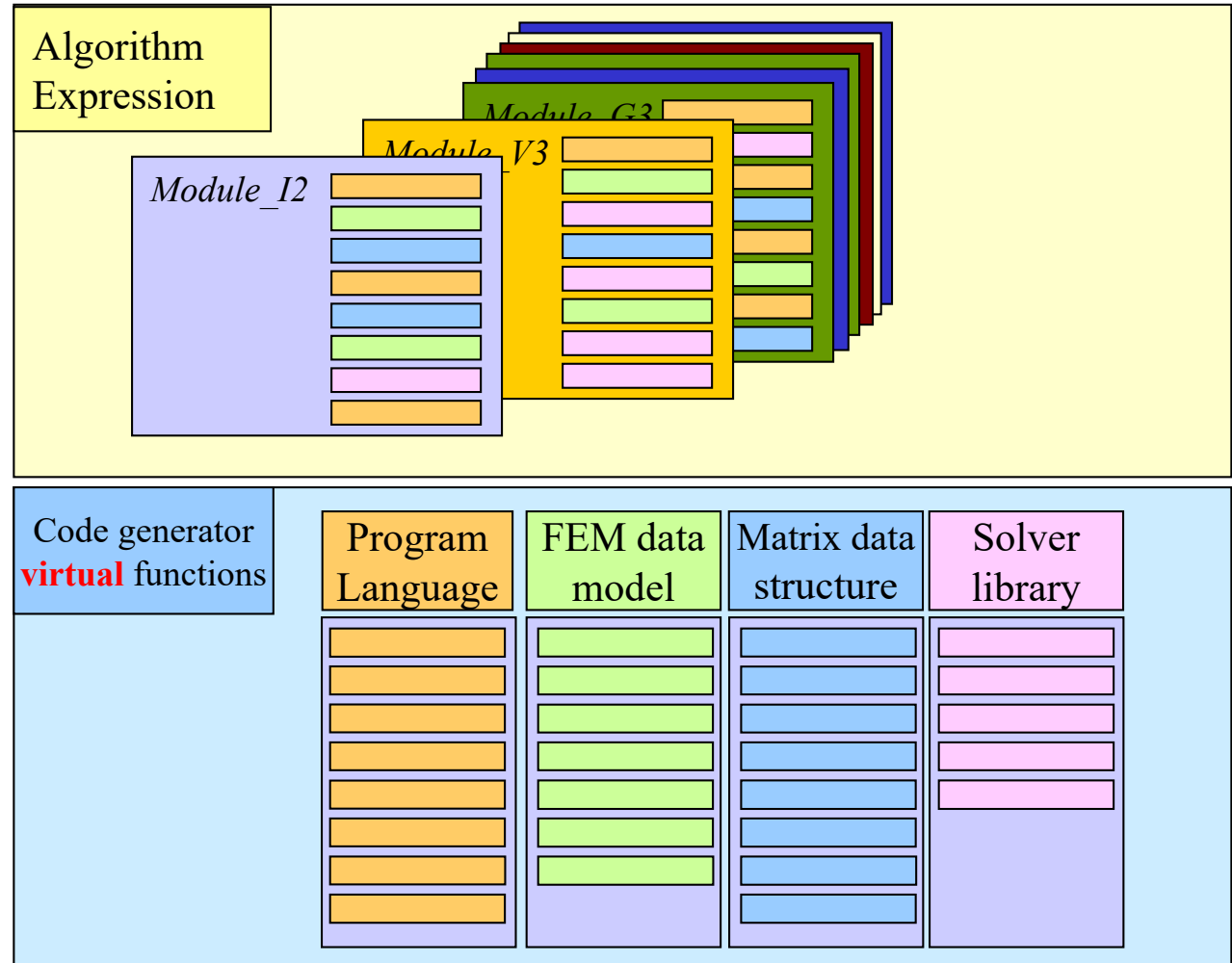
Code generator virtual functions
<pre>Process_a1(); Process_b1(); Process_c1(); Process_a2(); Process_c2(); Process_b2(); Process_d1(); Process_a3();</pre>

# Object oriented design of code generator 3

Knowledge-base  
of FEM software



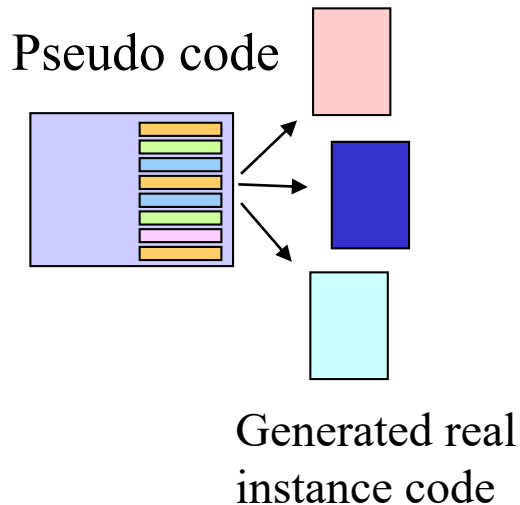
Algorithms expressed by pseudocode.



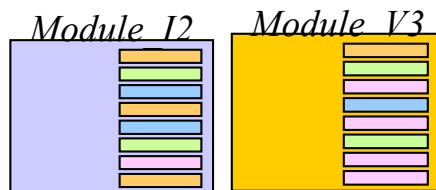
Code generator classified into 4 layers

# Merits of pseudo code representation of algorithms

---



- The implementation of algorithm into code generator is clearly visible, and independent of real implementation model.



- New algorithms can be expressed by using already defined pseudocodes.

(combination, small modification)

ex. Linear problem to nonlinear problem

# Repository mechanism 3 (Instances)

---

Mathematical  
discretization

Numerical  
algorithms

HPC oriented  
implementation

The FEM  
is adopted

Flexibility  
Applicability  
Boundary condition

Problem expressed  
in variational form

Pseudo code  
representation  
is adopted

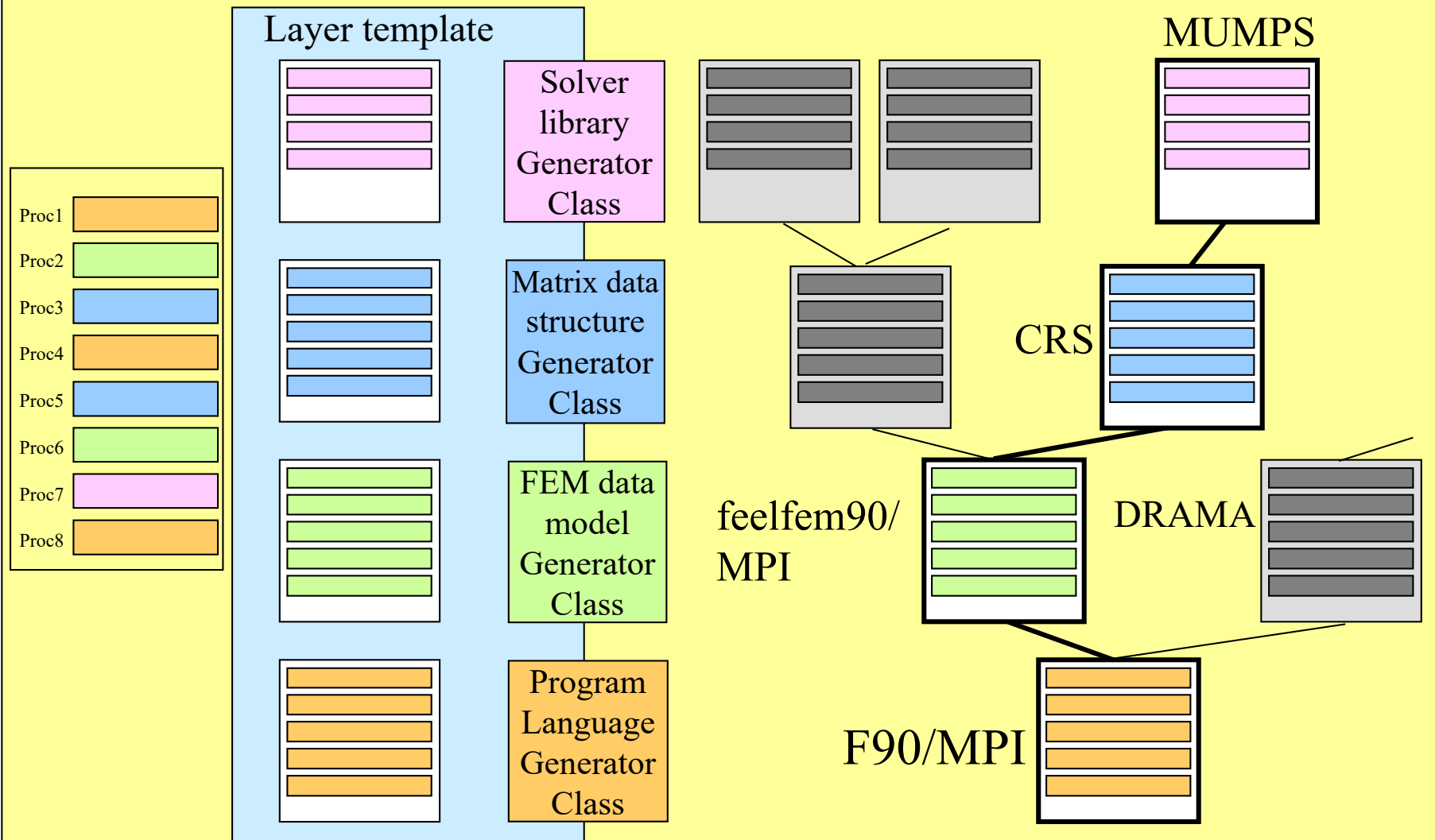
All numerical  
schemes are  
expressed by  
pseudo code

Pseudo code -  
real instance

Implemented  
C++ virtual  
code generator  
functions

# Adoption of individual ProgramModel (HPC model)

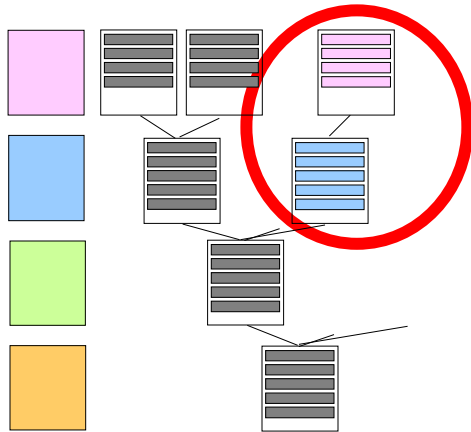
Code generator functions are implemented by the technique of Object-Oriented hierarchy



# Merit of inheritance of code generator function class

---

Code generator Classes are implemented with **template** and **inheritance** mechanism

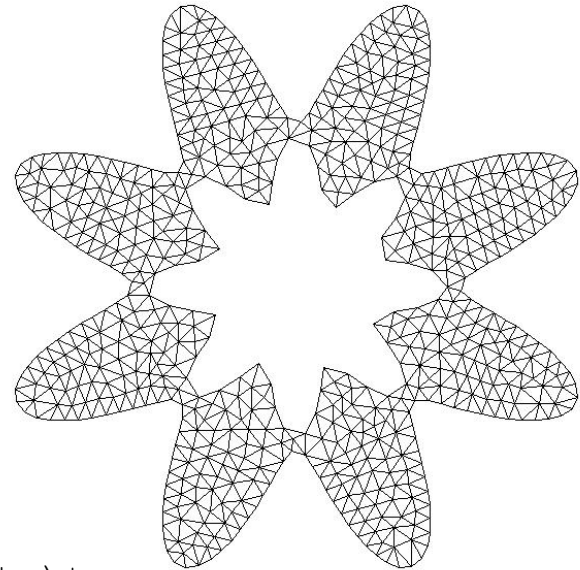


- Clear structure of the code generator
- Easy to add new library using existing generator functions
- Adding new library has no influence on old generator classes

# Pre/Post processor Interaction

## Interface to the Bamg mesh generator (for 2D)

```
mesh {  
  dimension 2;  
  
  double length = 1000;  
  double r = length*0.5*0.5;  
  double or = length / 2.0;  
  double ir = length / 8.0;  
  double cx = length/2.0, cy = length/2.0;  
  double pi = 3.14159265;  
  double rotate = pi / 11.0;  
  double pe = 6;  
  double pr = 0.3;  
  
  pedge hole [ ((r+r*cos(pe*t)*pr)*cos(t+rotate)+cx,  
               (r+r*cos(pe*t)*pr)*sin(t+rotate)+cy), t=0,6.28];  
  pedge outer[ ((or+or*cos(pe*t)*pr)*cos(t+pi/pe)+cx,  
               (or+or*cos(pe*t)*pr)*sin(t+pi/pe)+cy), t=0,6.28];  
  region reg1[tri](outer:[hole]);  
  domain dom(reg1);  
  vertices (800);  
}
```

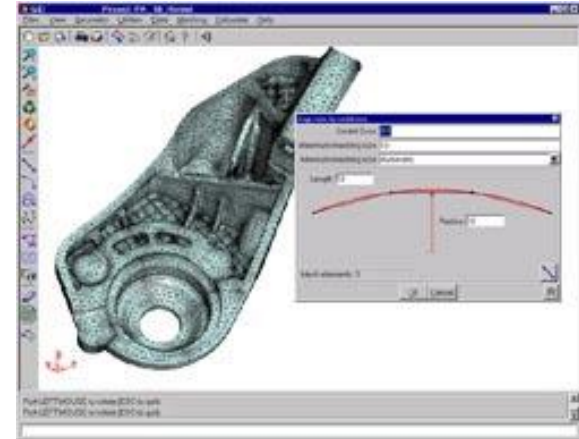


# Interaction with GiD pre/post processor

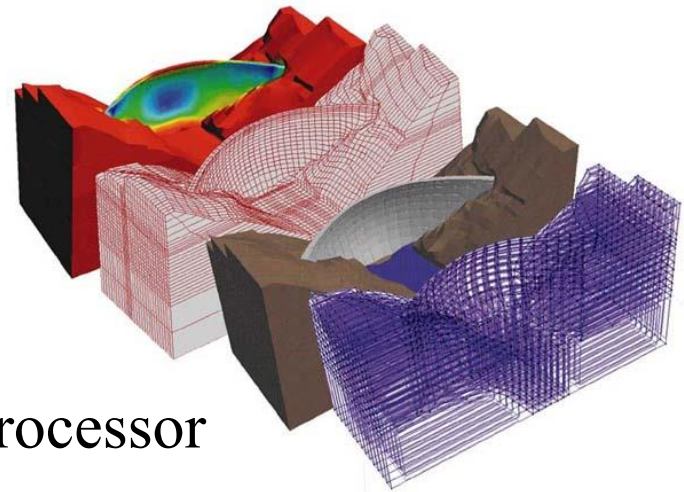
GiD is a pre/post processor developed in CIMNE/Barcelona



Geometry editor

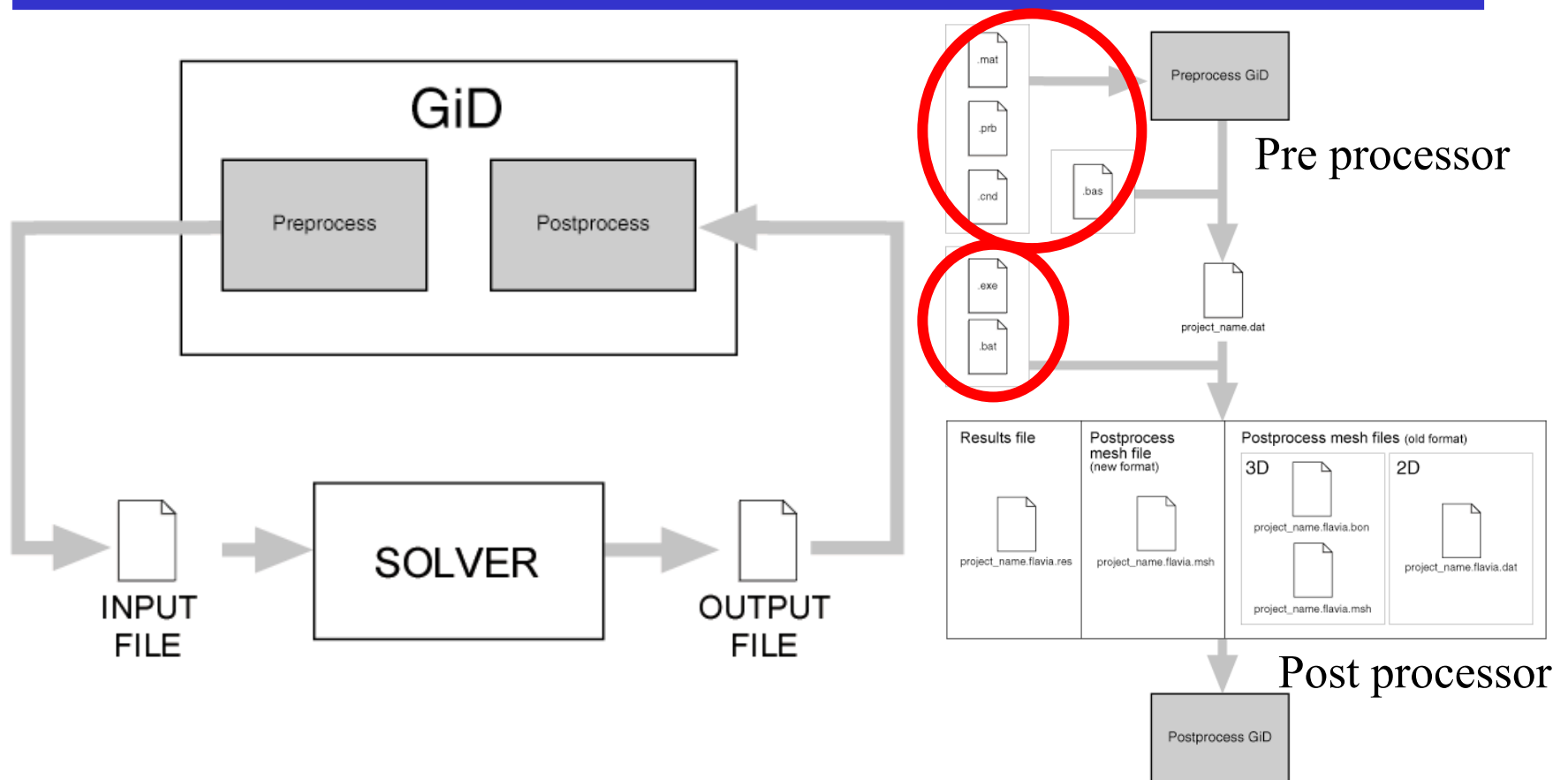


Meshing



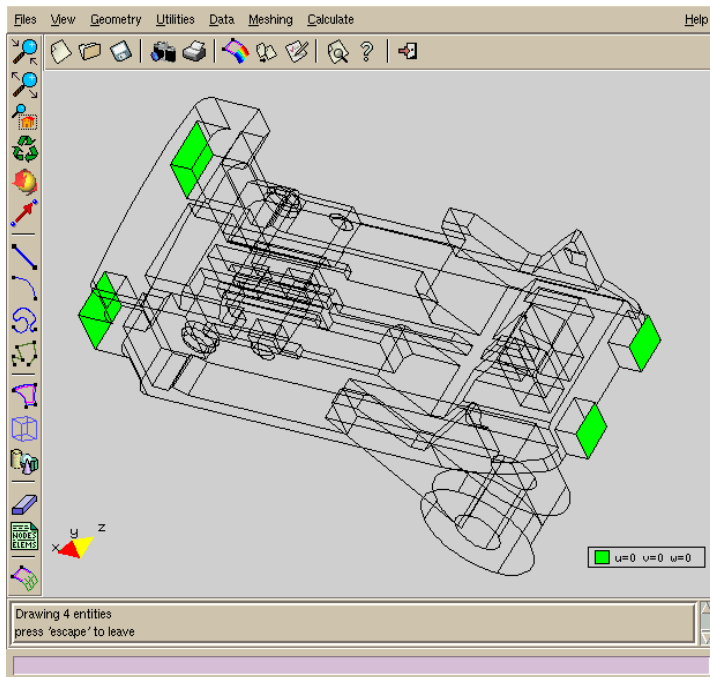
Post Processor

# Interaction with GiD

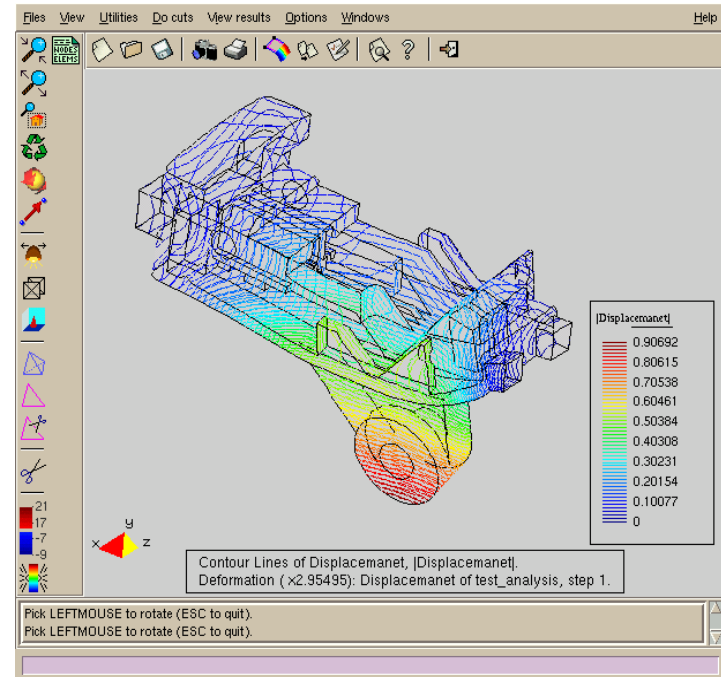


Interface files to connect GiD are automatically generated by feelfem.

# Examples of GiD interface

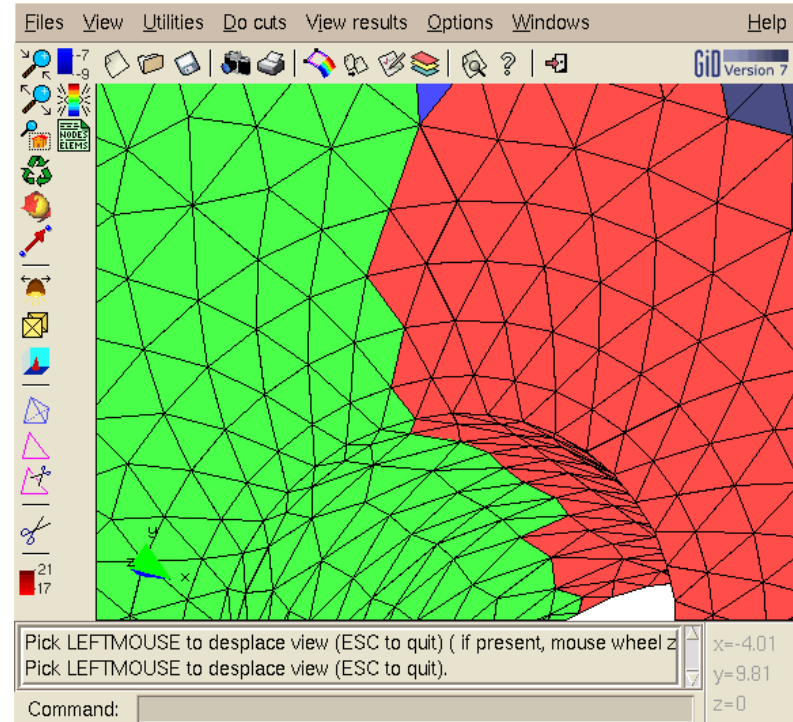
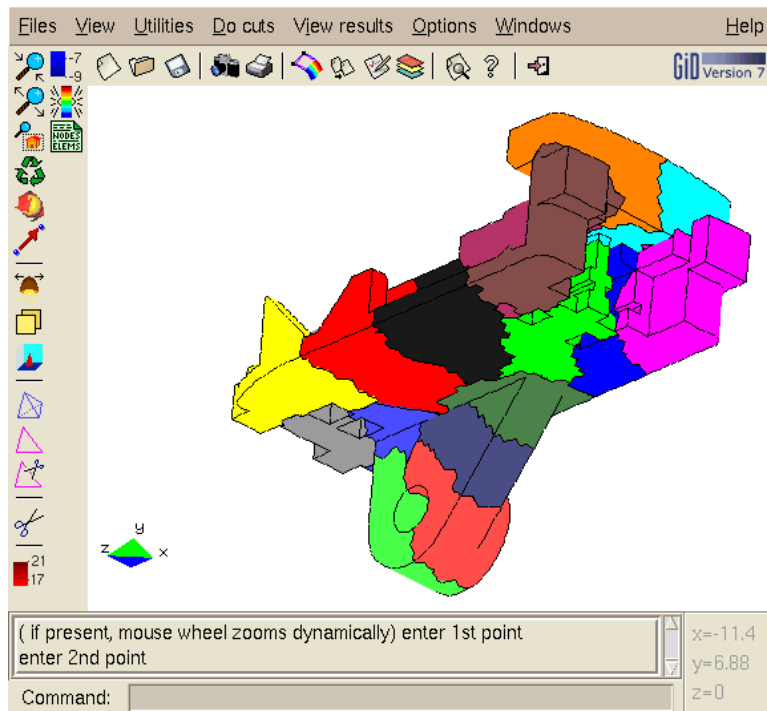


Pre processor  
to define boundary conditions



Post processor  
to see results

# Example pictures (parallel computation case)



Now MUMPS(CERFACS, France), parallel AMG(Germany), PILTUS (NEC Lab. Germany), PCP (AIST, Japan) parallel solvers are supported

# Conclusion

---

- Be able to handle various problems by variational formulation expressions (the FEM)
- Once a numerical algorithm is stored in this system, its reuse with various combinations is easy.  
(mathematical models, parallel programming paradigms, parallel solvers, etc.)
- Object-Oriented structure of this system enables easy expandability and easy maintenance.

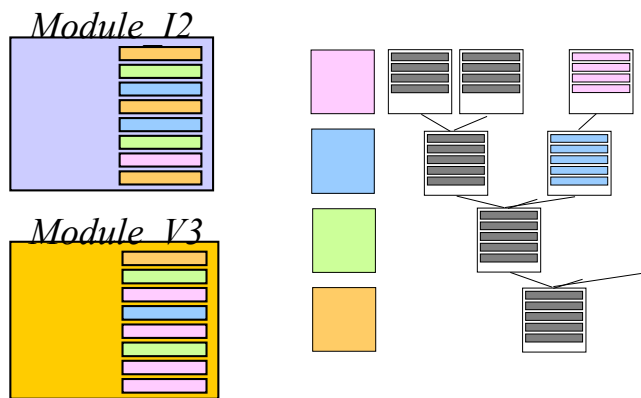
# Why Code Generator?

Similar project : Diffpack (C++ simulation code)

- + The FEM, using Object Oriented technology, Very flexible
- New results have to be translated into Diffpack by someone.
- Existing software is provided in various languages.(f77,c,f90,etc)
- C++ class design is elegant, but difficult to modify later.

FEEL code Generator : The system is not an analysis code, but a “knowledge database” software to store results from various people.

User can add new program models, new algorithms independently.

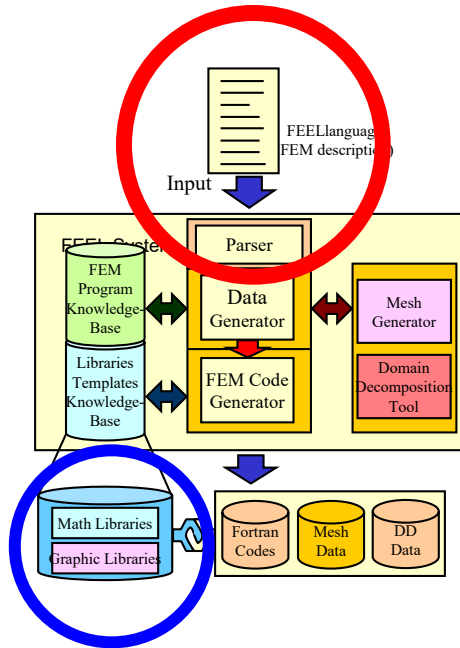


# PETSc、MATLAB

---

## MATLAB/PDE toolkit :

- + Portable rapid prototyping tool
  - + Fine mathematical style inputs
  - Basically interpreter.
  - Not for High Performance Computing
- 



## Project : PETSc

- + Sophisticated parallel library
- + Abstract level input
- Users have to know ....

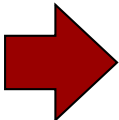
**PETSc may be implemented in FEEL  
as one of code model in coming version.**

# Typical usage for advanced users

---

## a) Try newly developed algorithm

New interpolation function (sometimes called element)  
New scheme (ex. Operator splitting schemes, etc.)  
New solvers (including parallel solvers)

 Once implemented in reusable form, user can apply to various problems without programming.

## b) Research and development of new algorithms

Development of numerical method using feel language

 Only for very advanced users, but quite effective

## c) Prototype generator

Use generated codes as a skeleton of prototype

 Effective tool for using new libraries

# feelfem = FEM Software knowledge data base with program and data generation functionality

---

1. High level programming interface for the FEM schemes
2. Code generation for new libraries, new parallel algorithms, etc.
3. Complete symbolic interface to mesh generators (for 2D)

## Who is the user, and what is the usage?

### *Advanced users -- numerical analysts*

- a) Try newly developed software results without programming
- b) Develop new algorithm using feel language
- c) Use as a skeleton generator for new program model

### *Major users -- students*

- a) Easy access to numerical analysis
- b) Easy access to parallel computation
- c) Easy access to the finite element method

# Solving PDE based problems

---

## Solving PDE based Problems

Problem Solving by solving PDE  
for daily analysis

- Crash analysis in automobile companies
- Weather forecast

Problem Solving by solving PDE  
for **NON**-daily analysis

- Rapid prototyping analysis
- Using new models, new algorithms

# Conclusion

---

- We have made an object oriented implementation of a code generator
  - Natural class hierarchy
  - Easy to expand new software components
  - Easy maintenance
- Several Program model/Matrix model/Library model are implemented and evaluated.

# Problems for building a code generator

---

Experience of developing FEEL prototype version

- The code generator itself may be a huge program system

- Reuse the old program
- Expandable to new libraries

# Environment for solving PDE

---

## Problem Solving for daily analysis

They can afford software development costs.

➡ Commercial package / Development team

## Problem Solving for **NON-daily** analysis

Engineers and researchers have to do programming to use new research results.

Programming is quite expensive, especially for High Performance Computing.

➡ Tool?

# Outline

---

- ◆ Importance of building a “knowledge-database” for PDE based Problem Solving Environment by a Code Generator
- ◆ Requirements for developing a FEM code generator
- ◆ Object Oriented Implementation of a code generator
- ◆ Examples and Current status
- ◆ Conclusion

# Demonstration

---

- Language functionality in feel language
- An upwind scheme
- New quadratic element
- Changing the programming languages and solvers

# Specification of operator in Matrix

---

$$\partial_x, \partial_y, \nabla$$

Simple operators

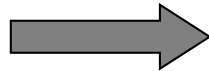


Specified by formula style

Program is automatically generated

**Special operator**

Special upwind  
technique, etc.



User subroutine which calculate  
the corresponding matrix

Description method in feel  
language will be introduced

# High-level Programming Interface

---

Discretize function  $u$  with triangular linear element

**fem u[P1] ;**

Assign 10.0 to  $u$  only in regionA

**u = 10.0, in regionA;**

Boundary condition  $u = 20 + x$  is defined on edgeB

**dbc: u = 20.0+x, on edgeB;**

display a contour graph of  $u$

**contour[u] ;**

# Components in *feel language*

---

- Region definition block
- Element definition block
- Quadrature definition block
- Variable definition block
- Scheme definition block
  - PDE definition block

# Typical FEEL program

## **/\* Region Definitions \*/**

```
mesh {  
  point a(0,0),b(0.17,0),c(0.17.085),d(0,0.085);  
  edge bcd(b,c,d);  
  region reg1[rect](a,b,c),reg2[rect](a,c,d);  
  domain dom1(reg1,reg2);  
  nodes dom1(400);  
}
```

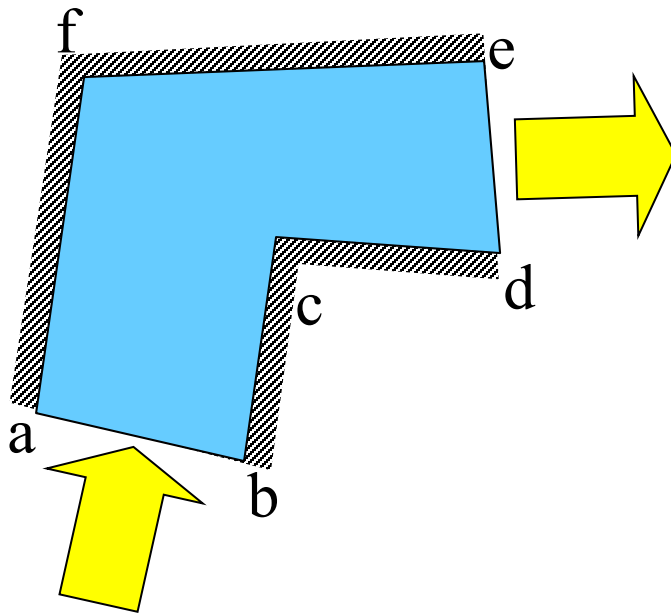
## **/\* Variable Definitions \*/**

```
var {  
  double rho,c,t0,tf,h,time,dt;  
  ewise kx[dom1]; ewise ky[dom1];  
  fem u[Q1],u0[Q1];  
}
```

## **/\* Scheme Descriptions \*/**

```
scheme {  
  kx = 10.0,in reg1;   kx=20.0,in reg2;  
  .....  
  iter:   time = time + dt;   i = i + 1;  
  
  solve[u]{  
    linear method: skyline;  
    quadrature method: gauss3x3;  
    weq:a*(u-u0)*$+dt*k*(dx(u)*dx($)+dy(u)*dy($)),<dt*g*$> ;  
    nbc: g = h * ( u - tf ), on bcd;  
  }  
  
  contour[u](umax=600,umin=50,mesg='Plot u',winsiz=300);  
  u0 = u;  
  if(i < 20) goto iter;  
}
```

# Region Definition



Mesh {

point a(0,1),b(3,0),c(4,5);  
point .....

edge inflo (a,b);  
edge outflow(d,e);  
edge fix(b,c,d);

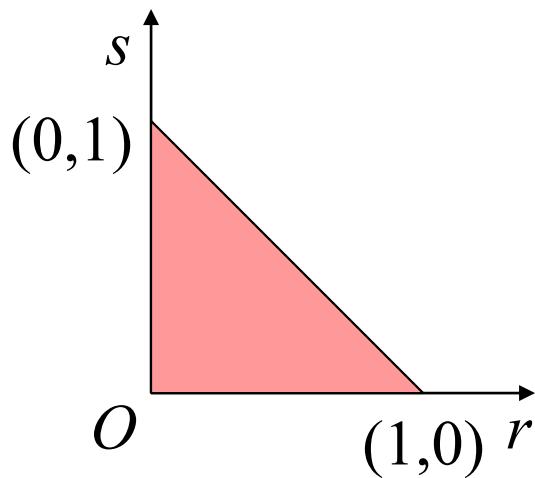
region reg1[tri](a,b,c,d,e,f);

domain dom1(reg1);  
nodes dom1(1000);

}

# Element Definition

Triangle linear element(pre-defined in FEEL)



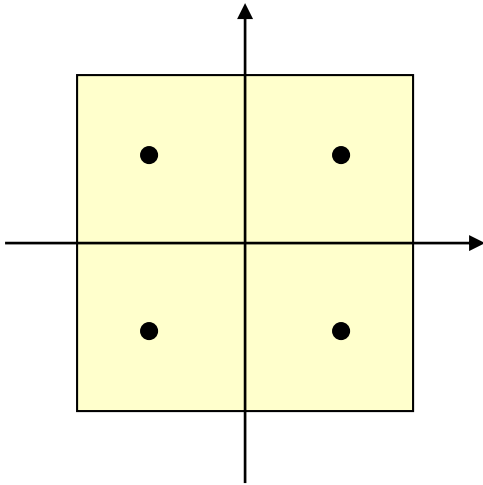
Reference element

```
element P1[tri] {  
  (0,0) : 1 - r - s;  
  (1,0) : r;  
  (0,1) : s;  
}
```

tri	triangle
rect	rectangle

# Quadrature Definition

---



```
quadrature gauss2x2[rect] {  
    const p = 0.577350269189626;  
    const pm= -0.577350269189626;  
    ( p, p): 1.0;  
    ( p,pm): 1.0;  
    (pm, p): 1.0;  
    (pm,pm): 1.0;  
}
```

# Scheme Definitions

---

```
scheme {  
  
    assign    statements (scalar, element-wise,node-wise)  
    if/goto   statements (conditional branch)  
    graphic   statements (contour,vector,perspective,... )  
    I/O       statements (read,write, file I/O)  
  
    solve blocks (define PDE problem)  
  
}
```