Interoperability and Interchangeability in the Overture Framework

Kyle Chand
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California
www.llnl.gov/CASC/Overture

UCRL-PRES-209519

In the next 20 minutes:

Overview of Overture

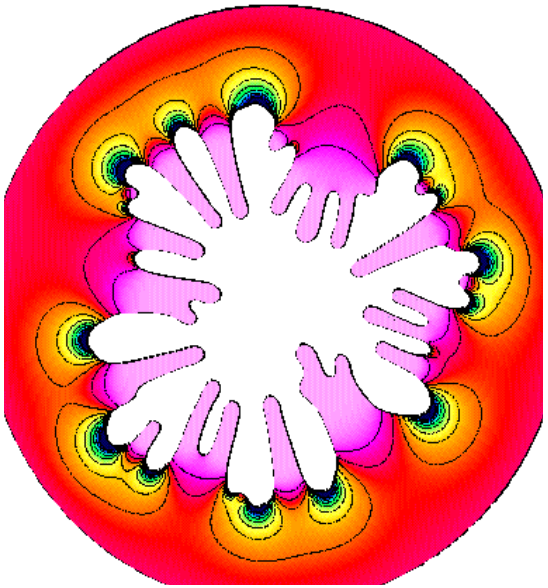
Overture's composite grid framework

Interoperable grids and discretizations

From interoperable to interchangeable components

Questions for TSTT

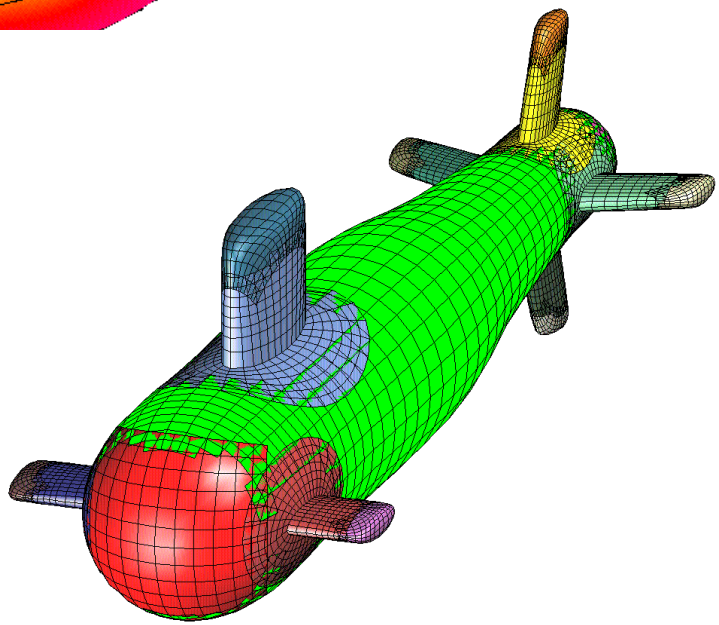
Overture: A Toolkit for Solving PDEs

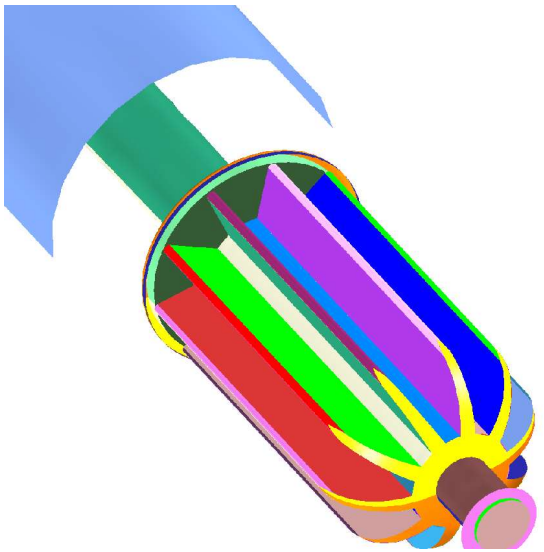


PDE Solver Development

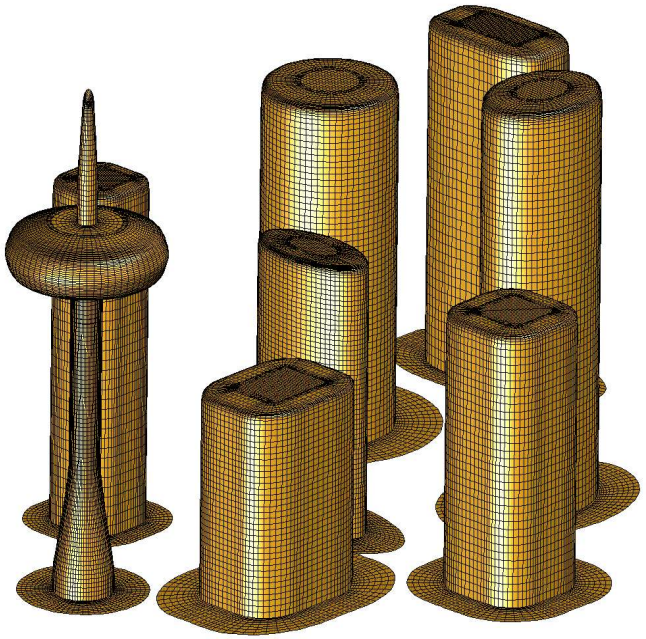
Grid Generation

Geometry





Overture's Composite Grid Framework

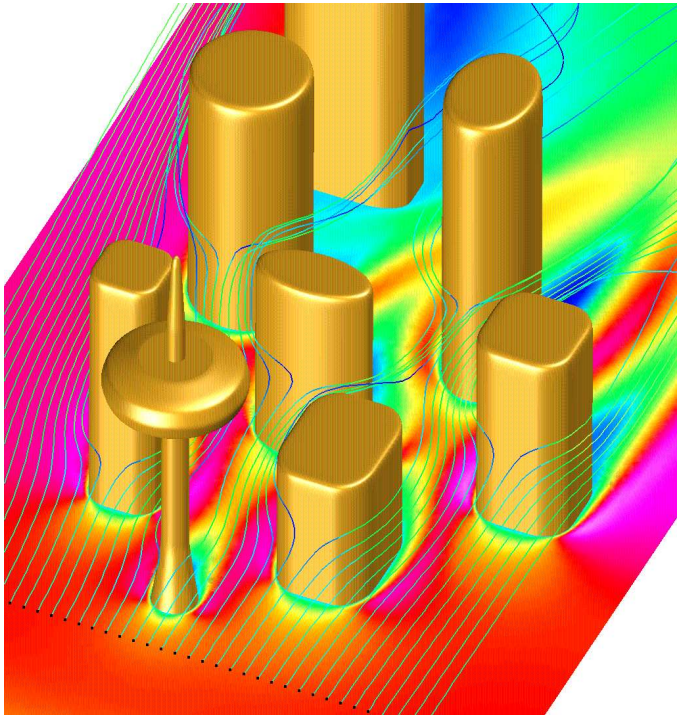


Multiple component grids are assembled
into a single representation
Optimized discretizations used on each component
Composite Grid organization can be used
to orchestrate different models on each grid

High-level interface : prototyping

Intermediate interface : development

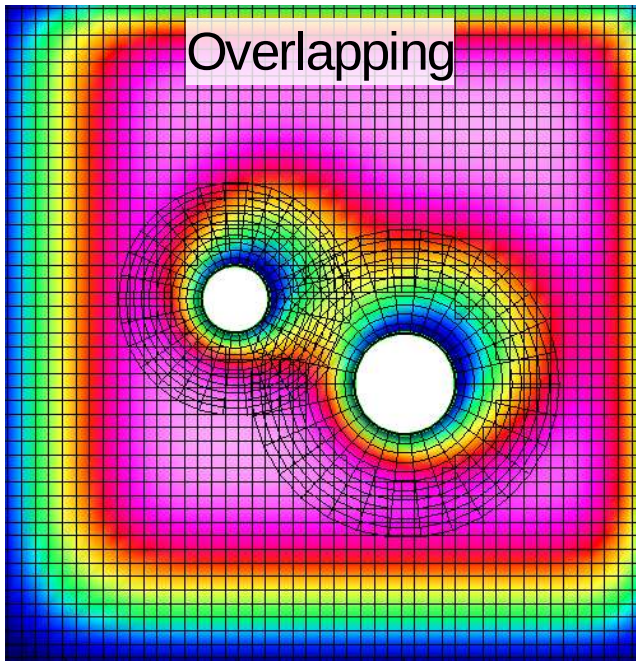
Low-level interface : performance

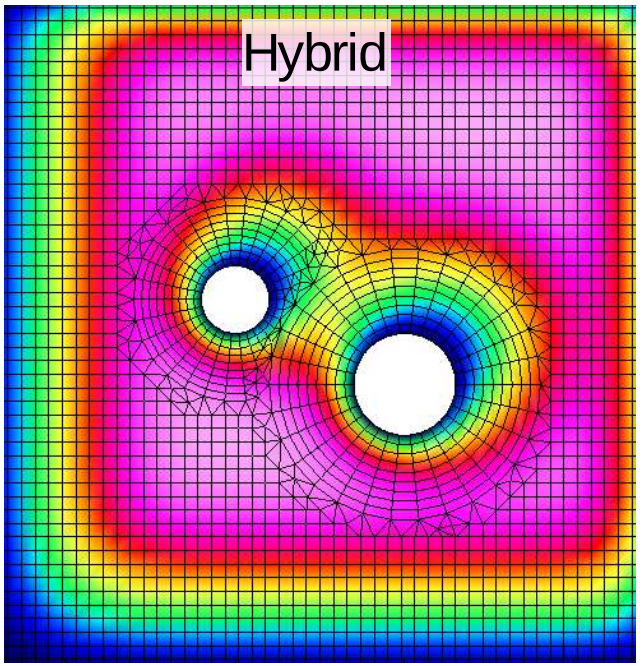


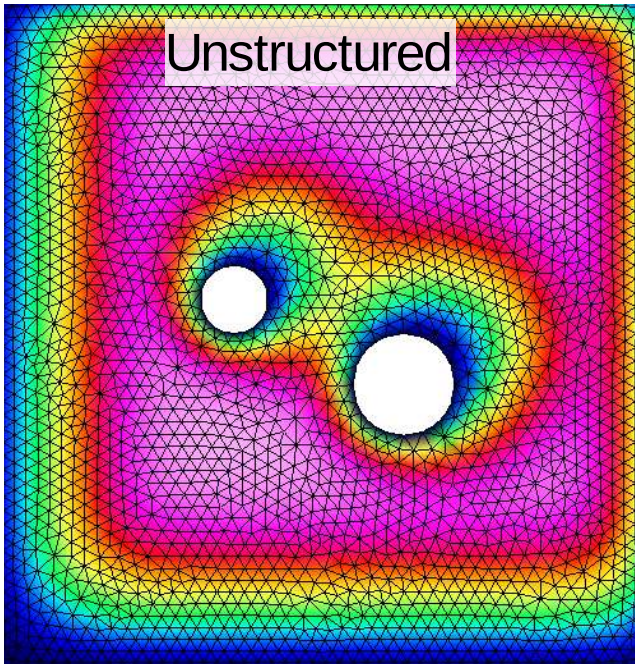
Interoperable grids and discretizations: high level interfaces

$$u_t = -au_x - bu_y + \nu(u_{xx} + u_{yy})$$

```
u += dt*( -a*u.x() -b*u.y() + nu*(u.xx() + u.yy()) );
```







Interoperable grids and discretizations: high level interfaces

Forward operators provide explicit evaluation of the operator:

```
u += dt*( -a*u.x() -b*u.y() + nu*(u.xx() + u.yy()) );
```

Backward operators are used to build coefficient matrices:

```
coeff =  -a*u.xCoefficients()
         -b*u.yCoefficients()
         + nu*(u.xxCoefficients() + u.yyCoefficients
```

Boundary condition operators:

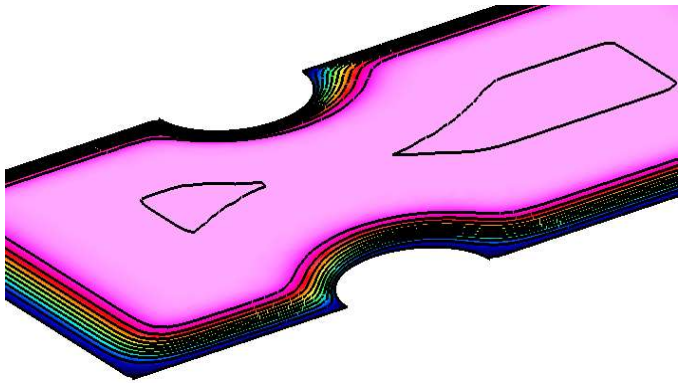
```
u.applyBoundaryCondition(0,dirichlet,allBoundaries,bcValue);
coeff.applyBoundaryConditionCoefficients(...);
```

Interpolation:

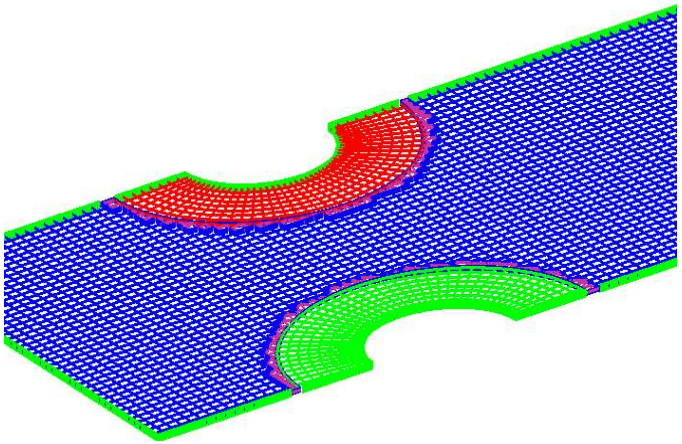
```
u.interpolate();
```

Composite Grid Components

```
CompositeGrid cg;                          cg.get("grid.hdf");
CompositeGridOperators op(cg);
realCompositeGridFunction u(cg);           u.setOperators(op);
```



`u.laplacian()`



CompositeGridFunction

- organizes individual MappedGridFunctions
- can represent a coefficient matrix
- MappedGridFunctions manage DOFs

CompositeGridOperators

- orchestrates operator evaluation for different grids and discretizations

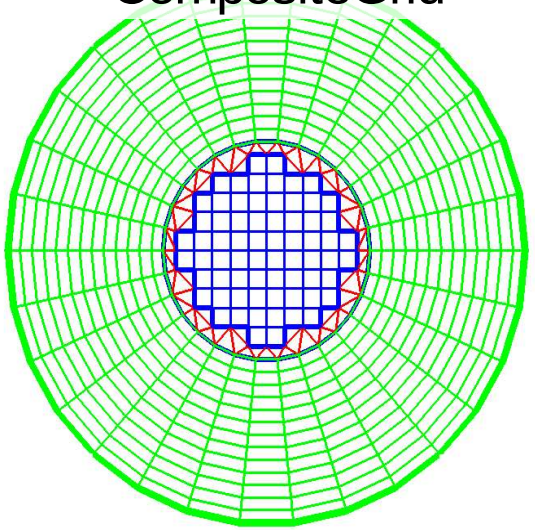
CompositeGrid

- organizes individual MappedGrids
- maintains inter-grid connectivity (expressed as interpolation locations)
- understands grid hierarchies

CompositeGrids :

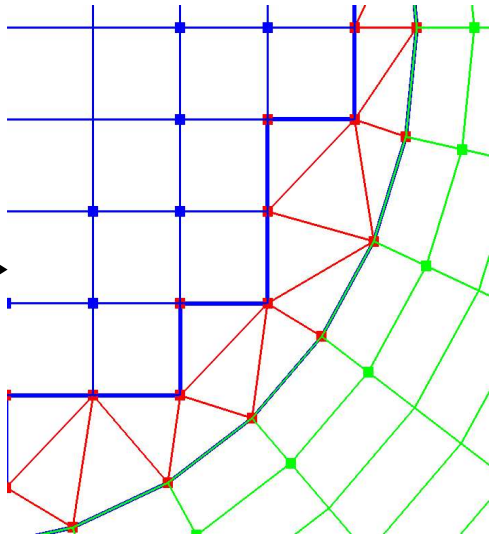
individuals organized into a coherent system

CompositeGrid

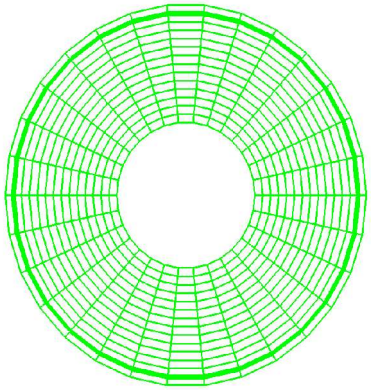


Connectivity

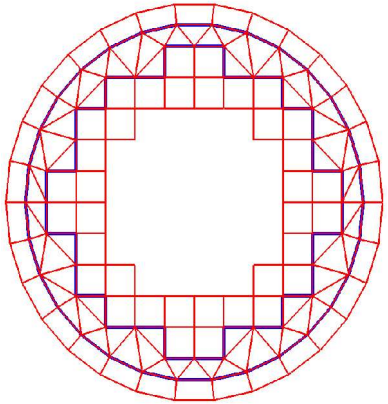
(interpolation coordinates)



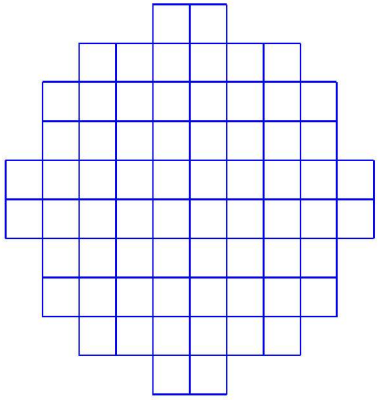
MappedGrid(Annulus)



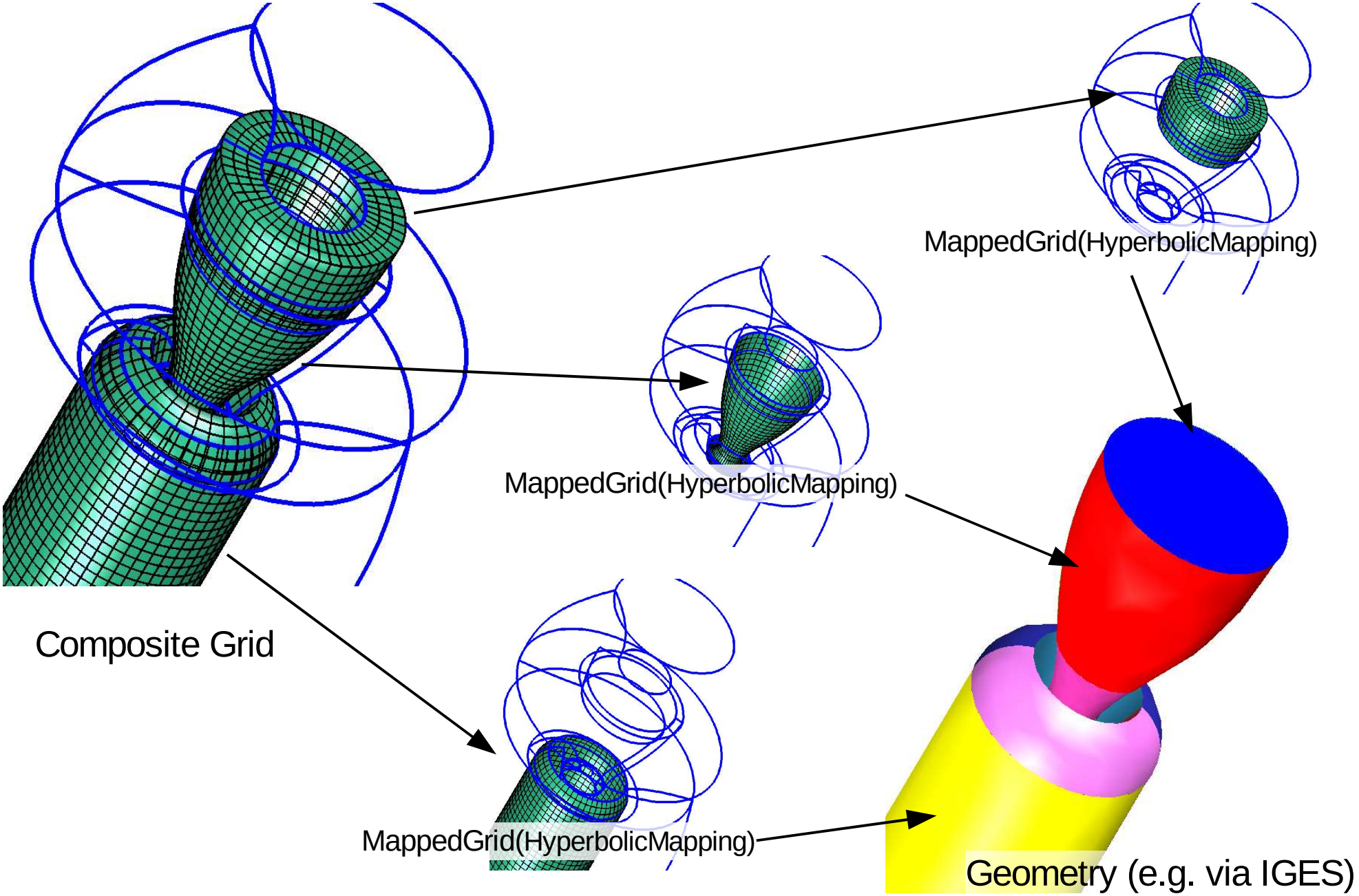
MappedGrid(Unstructured)



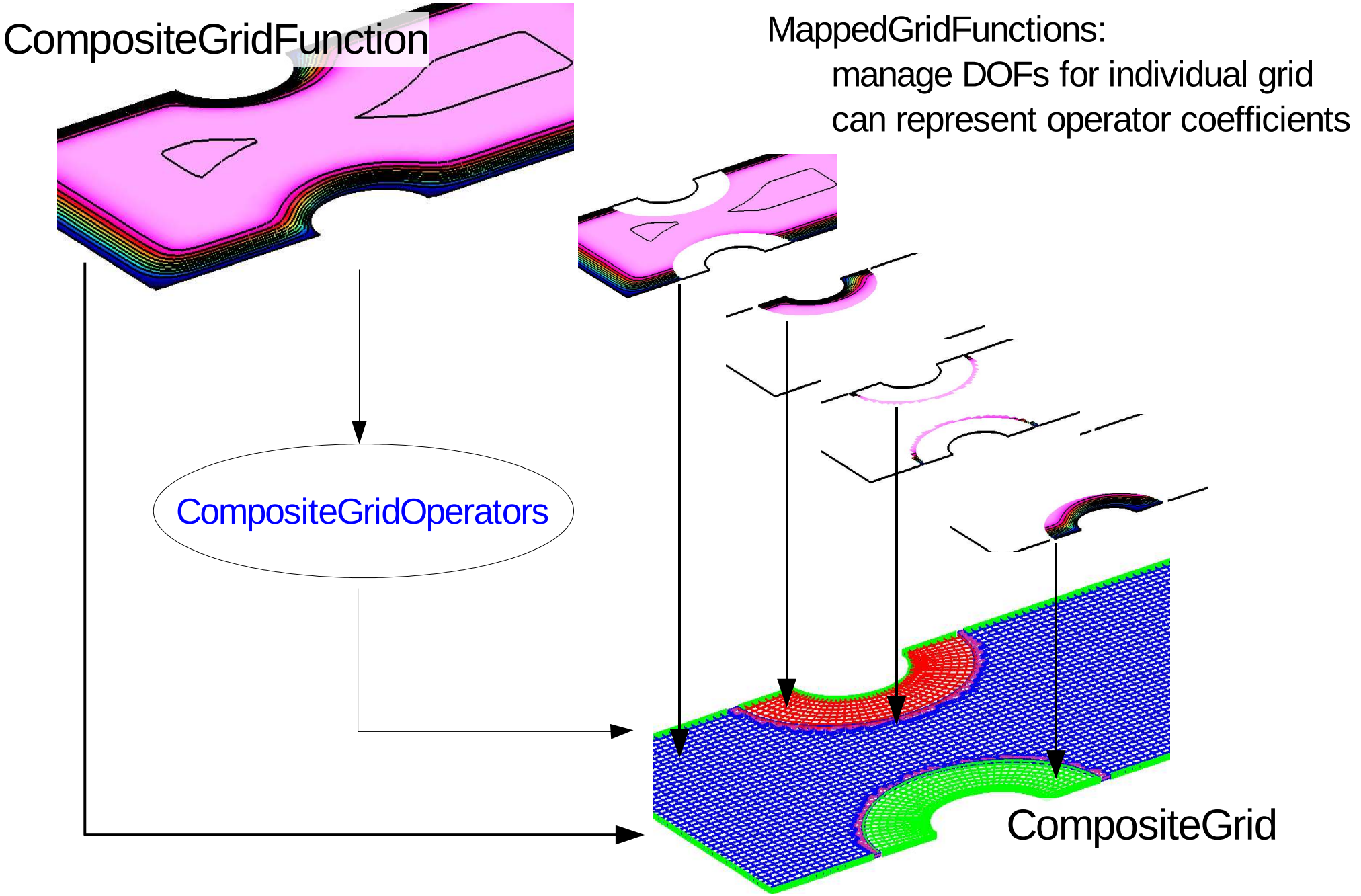
MappedGrid(Square)



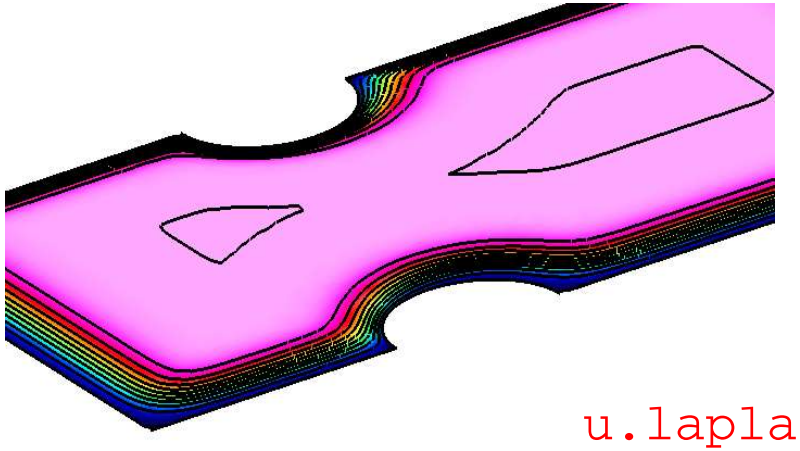
CompositeGrids access geometry via components



CompositeGridFunctions:



CompositeGridOperators:



MappedGridOperators:

- interface to discretization
- forward operator evaluation
- backward coefficient construction

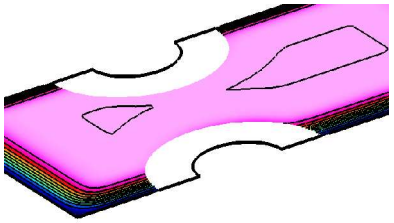
`u.laplacian()`

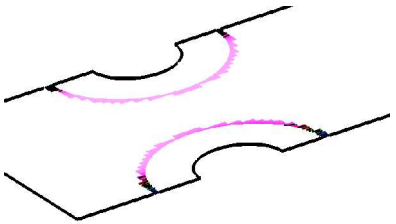
CompositeGridOperators

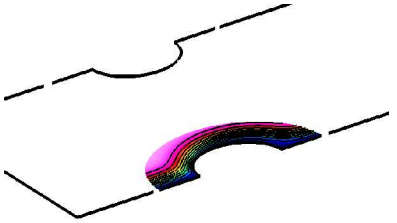
MappedGridOperators (Cartesian FD)

MappedGridOperators (Unstructured FV)

MappedGridOperators (Curvilinear FD)







Intermediate Interfaces:

better performance through lower level interfaces

```
u += dt*( -a*u.x() -b*u.y() + nu*(u.xx() + u.yy()) );
```

Individual operator evaluation

Simultaneous array temporary generation/propagation for each grid

Highest level interface :

- fast for prototyping, execution performance suffers

Overture provides an intermediate level interface to interoperable operators:

- the operators are told what to evaluate for each grid
- all the evaluations for a single grid are performed at once
- memory allocations are reduced to a minimum
- clear path from high to intermediate level

Intermediate Interfaces:

Example

```
u += dt*( -a*u.x() -b*u.y() + nu*(u.xx() + u.yy()) );
```

what to evaluate

```
for ( ...each grid... )
   {
      operators[grid].setNumberOfDerivativesToEvaluate ( 4 );
      operators[grid].setDerivativeType(0,xDerivative,uu[0]);
      operators[grid].setDerivativeType(0,xxDerivative,uu[1]);
      ...
   }
```

when to evaluate

```
for ( ...each timestep... )
   {
      for ( ...each grid... )
      {
         u[grid].getDerivatives(...);
         u[grid] += dt*( -a*uu[0] -b*uu[2] + nu*(uu[1] + uu[2]));
            ... (bc's and interp)
      }
   }
```

Interchangeable interfaces:
compile time optimization

Existing codes may want access to discretizations without adopting a large framework

Development of interoperable components from generic/interchangeable source

Overture provides a low-level, macro driven interface to many operators:

- C/C++ and Fortran code generation
- composition of operators for high-order finite difference schemes
- generation of Cartesian/curvilinear/unstructured code from single source
- pre-processing performed by “bpp”

Interchangeable interfaces:
a trivial bpp example

Getting the vertex coordinates to compute initial conditions:

Macro Definition

```
#defineMacro VERTEX_COORD(X, GRIDTYPE)

   FORLOOP(a, 0, ndim)
   #If GRIDTYPE==”Cartesian”
         X(a) = x0(a) + dx(a)*ii(a);
   #Else
         X(a) = vertex(ii(0),ii(1),ii(2),a)
   #End

#endMacro
```

Macro Use

```
if ( isCartesian )
   VERTEX_COORD( xVert, “Cartesian” );
else
   VERTEX_COORD( xVert, “” );
```

The bpp pre-processor can generate for Fortran or C
Looks like cpp, acts like Perl

Interchangeable interfaces:
an interesting example

```
u += dt*( -a*u.x() -b*u.y() + nu*(u.xx() + u.yy()) );
```

array operators, individual operator evaluation, multiple array allocations

```
#Include "defineDiffOrder4f.h"

#If #GRIDTYPE== "rectangular"
    #defineMacro UXX(I,J,K) uxx42r(I,J,K,0)
    ...
#Else
    #defineMacro UXX(I,J,K) uxx42(I,J,K,0)
    ...
#End

...

u[grid](i,j,k)
    += dt*( a*UX(i,j,k)+b*UY(i,j,k) + nu*(UXX(i,j,k) + UYY(i,j,k));
```

2D 4th order curvilinear grid operator
~100 continuation lines!

scalar indexed, optimizable loop with inlined operators, no temporaries

We need better tools for code generation and optimization!

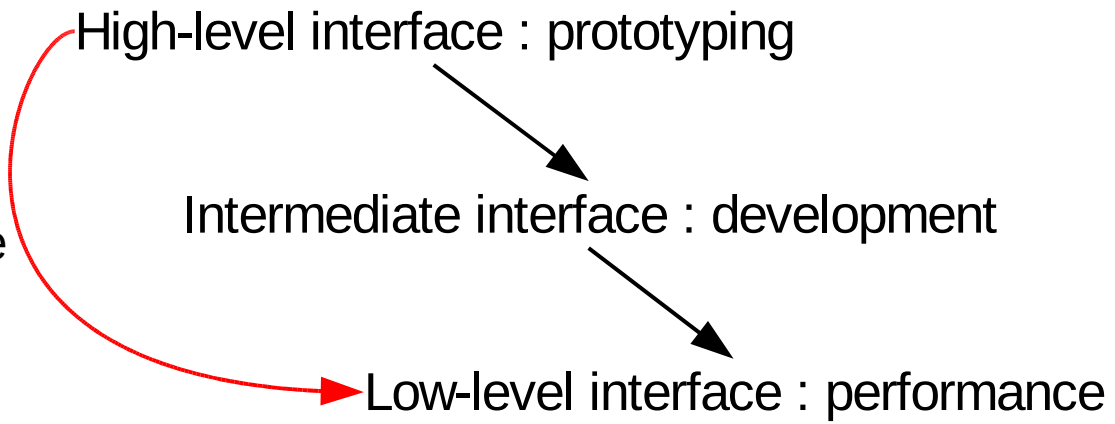
BPP:

More powerful than cpp
- easier to combine macros to any depth
- more sophisticated programming
- multiple source files from one bpp source

More convenient than C++ inline functions
- no need to pass lots of arguments
- easier on compiler optimization analysis
- works with Fortran

More usable than C++ expression templates
- readable code results in easier debugging
- easier on optimizers
- finite compile times
- works with Fortran

Has Issues
- generated code can be obscure
- yet another tool to support; no standards



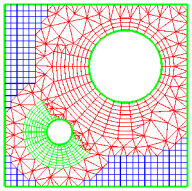
What we really want:

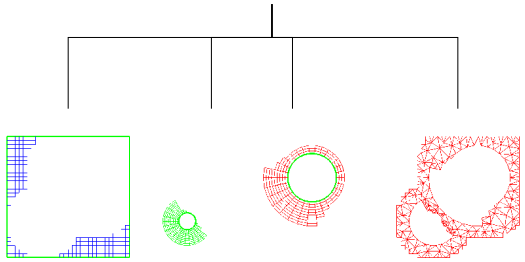
Automatic optimization of high level abstractions

Rose pre-processor aims to provide this capability

Implementation developers specify transformations from high level abstractions to optimized code

Questions for TSTT





TSTT has essentially ignored interchangeable software
TSTT has ignored structured composite grid applications

Should TSTT consider some high level interfaces to guide the current low level work?

TSTT has an “interoperable” unstructured mesh interface
- how do we get better performance?
- how can we make it clear what is interoperable and what is interchangeable?
- it should be done consistently across interfaces...

Can/Should TSTT consider a prototype->performance interface hierarchy?
- if so, does TSTT need a tool like BPP ?
- we really want a source-to-source translator built with Rose !

Obtaining Overture

Overture home page:
www.llnl.gov/CASC/Overture