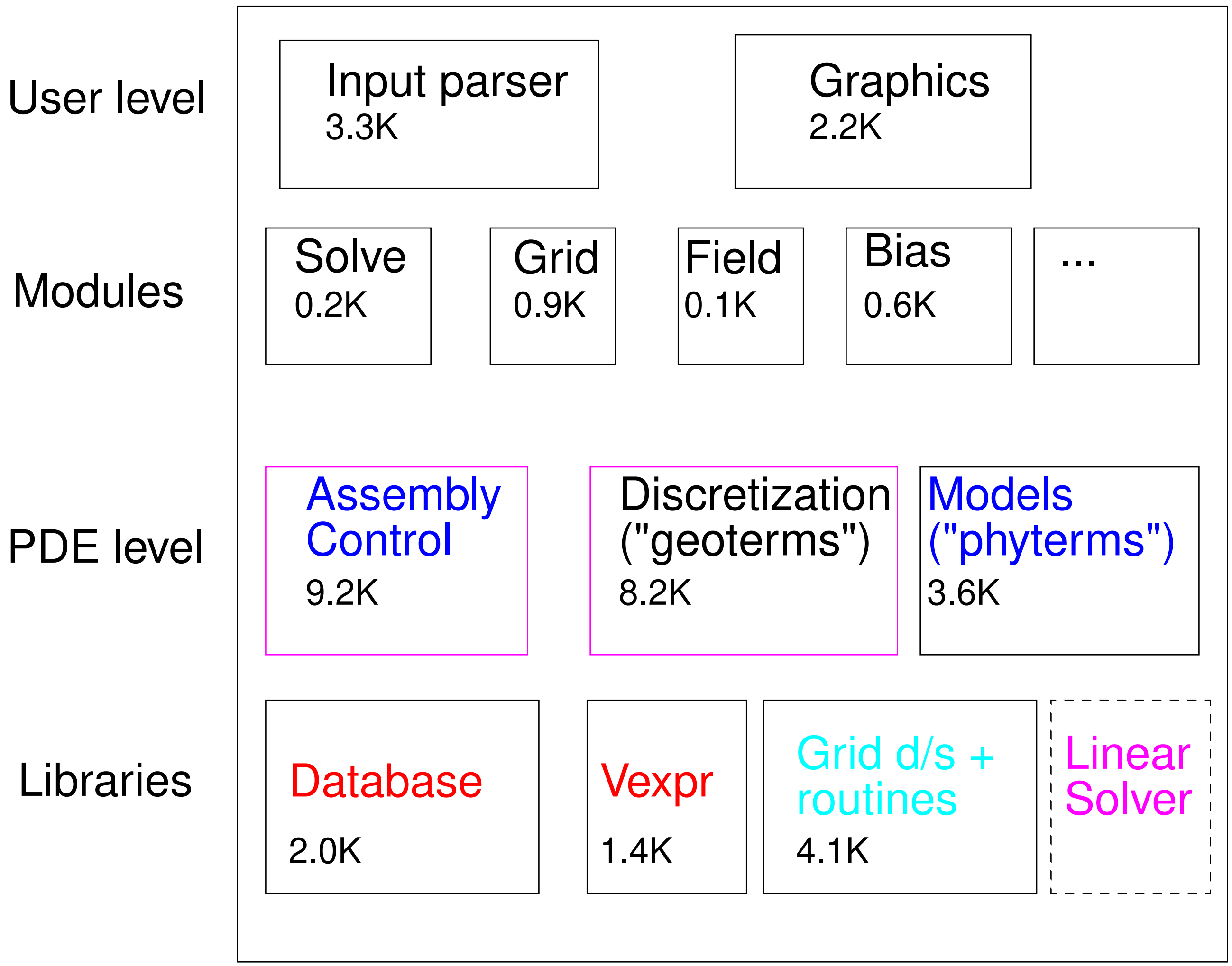
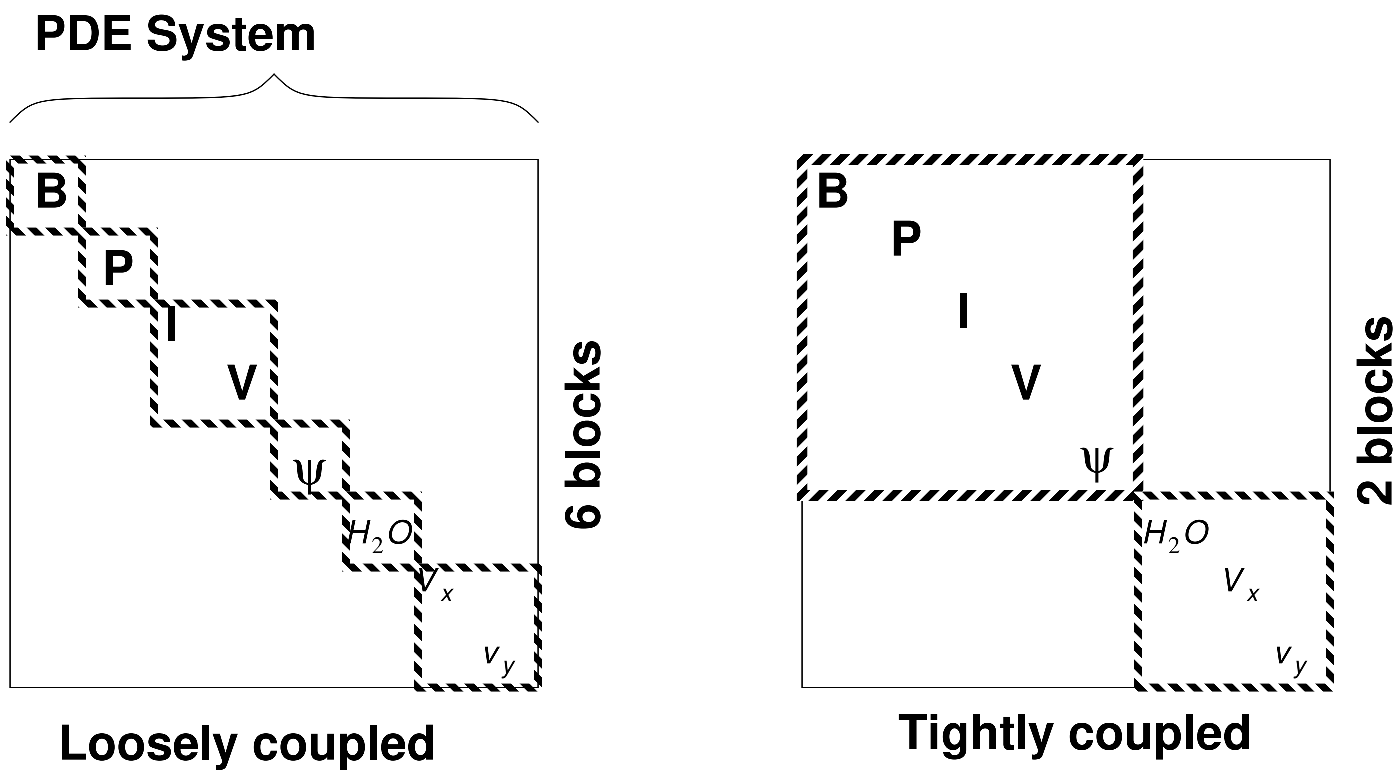
PROPHET Internals

- PDE description
  - externals
- Database
  - externals
  - internals
- Grid datastructures
- PDE description
  - internals
- Solution flowchart
- Matrix structure



System decomposition

- Problem may be composed of several blocks of PDE/ODEs



- Blocks may be sparse

Differential Equation Specification

- 2 level decomposition
  - PDE is a sum of terms
  - Each term is a combination of a geometrical and a physical operator

$$\mathrm{PDE} = G_1 P_1 + G_2 P_2 + G_3 P_3 + \ldots$$

- Geometrical operator

$$\nabla\times$$

$$\nabla\bullet$$

$$\frac{\partial}{\partial t}$$

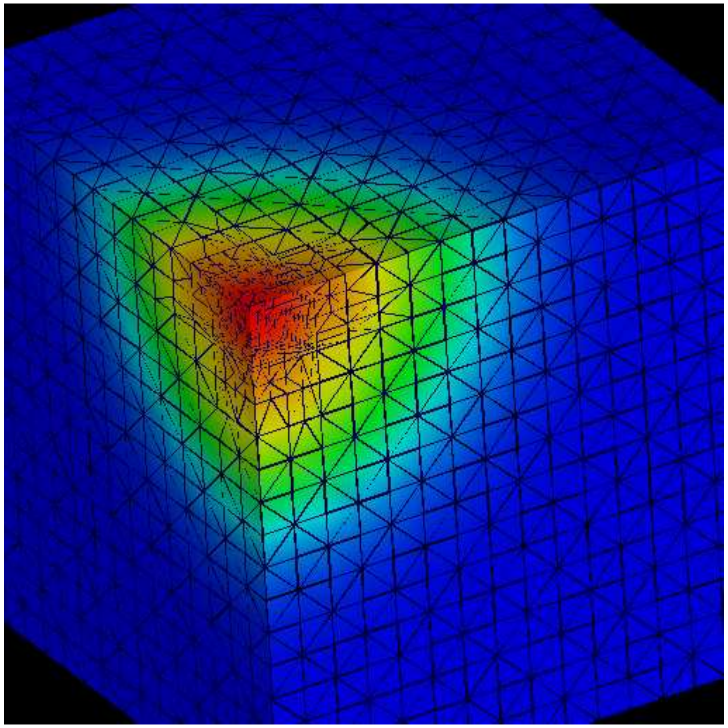
- Physical operator

$$F_A = f(A, \nabla A, X, \nabla X)$$

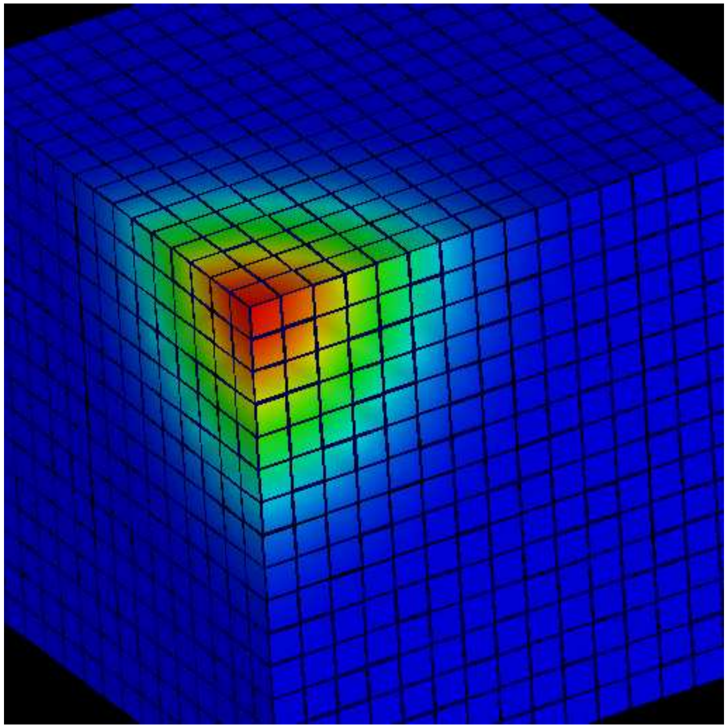
Same equations, different meshes

- Goal is to separate geometry information from physical information
  $\Rightarrow$ Model developer should not need to know about grid or discretization

Irregular mesh



Regular mesh



Decomposition of Defect Diffusion

$$\boxed{\nabla \bullet (D_I \nabla I)} \; \boxed{-k(IV - I^* V^*)} \; \boxed{\frac{\partial I}{\partial t}} = 0$$

$$\boxed{\nabla \bullet (D_V \nabla V)} \; \boxed{-k(IV - I^* V^*)} \; \boxed{\frac{\partial V}{\partial t}} = 0$$

2 Laplacians

1 binary recombination

2 transients

Description in database

```
math.defects = (list) {
    transient = (list) {
        order = (string) "int,vac";
        nterm = (int) 3;
        term0 = (list) {
            geoterm = (string) "box-laplacian";
            phyterm = (string) "lapflux";
            sol = (string) "int";
            dep = (string) "int";
            deptype = (string) "DIAG:DIAG=grad";
        };
        term1 = (list) {
            geoterm = (string) "box-laplacian";
            phyterm = (string) "lapflux";
            sol = (string) "vac";
            dep = (string) "vac";
            deptype = (string) "DIAG:DIAG=grad";
        };
        term2 = (list) {
            geoterm = (string) "diagonalweight";
            phyterm = (string) "bulkrecombination";
            sol = (string) "int,vac";
            dep = (string) "int,vac";
            deptype = (string) "ALL:ALL=conc";
        };
        term3 = (list) {
            geoterm = (string) "interface";
            phyterm = (string) "surfacerecombination";
            sol = (string) "int,vac";
            dep = (string) "int,vac";
            deptype = (string) "DIAG:DIAG=conc";
        };
        maxNewton = (int) 15;
        ...
};
```

Discretized operators

- Divergence operator
  - finite element
  - finite difference

- Upwinding operator

- Nodal weighting operator
  - lumped mass matrix
  - consistent mass matrix

- Interface weighting operator
  - lumped mass matrix

Elements

- C_0^k finite elements

- Shape functions for
  - intervals 1D

  - triangles 2D
  - quads 2D

  - tetrahedra 3D
  - bricks 3D
  - pyramids 3D
  - prisms 3D

Prefabricated physical operators

The following physical operators, among others, are predefined:

Associated with divergence operators	
lapflux	$D\nabla C_n$
equilflux	$D(\psi)(\nabla C_n + \xi_n C_n \nabla\psi)$
drift	$\mu A \nabla B$ - using central differences
updrift	$\mu A \nabla B$ - using upwinding
Associated with nodal operators	
two2one	A + B <-> C
poissonflux	$Q = q(N_d - N_a + n_i(e^{\Psi} - e^{-\Psi}))$
set_active	electrically active concentration of n chemical species
elim_carrier	$n = e^{\Psi}, p = e^{-\Psi}$
cluster	C + I <-> C
odefunc	general purpose function of n variables
expdecay	$-C/\tau$
integrate	S ($\partial C/\partial t = S$ means $C = C_0 + \int S dt$)
Associated with the interface operator	
segregation	$k(C_n[\mathrm{mat1}] - C_n[\mathrm{mat2}]/m)$
radiation	$k(C - C^*)$
odesurf	general purpose function of n variables
Associated with the dirichlet operator	
default.dirichlet	Defines dirichlet boundary conditions
device.dirichlet	Charge neutral values for ψ, n, p
Arithmetic	
prod,divide,scale	$A \cdot B, A/B, A/A^*$

Writing new modules

- Creating new operators, particularly reactions, is easy
- All operators get a standard list of arguments
- Subroutine must calculate the reaction and its derivatives
- Example: clustering

$$\frac{\partial I}{\partial t} = (k_{cr}C - k_{cf}IC) + \textit{other stuff}$$

$$\frac{\partial C}{\partial t} = -(k_{cr}C - k_{cf}IC)$$

- Operator asks for C,I as input and looks up k_{cr}, k_{cf} in the database.
- It then computes a reaction rate $F = (k_{cr}C - k_{cf}IC)$ and adds it with a +sign to the C equation and a -sign to the I equation.
- It then computes $\partial F / \partial C = k_{cr} - k_{cf}I$ and stores it, and similarly $\partial F / \partial I = k_{cf}C$

Example phyterm

$$f(c_1, c_2, c_3) = k_f c_1 c_3 - k_r c_2$$

```
      integer function odef( path, ireg, nn, dim, ifunc, ideriv
     +              nfun, mfun, nsol, msol,
     +              sol, gsol, f, df, dgf)
      integer path, ireg, dim, nfun, nsol, ifunc, ideriv, nn
      integer mfun(nfun), msol(nsol)
      double sol(nn,nsol), gsol(nn,dim,nsol)
      double f(nn,nfun), df(nn,nfun,nsol)
      double dgf(nn,dim,dim,nfun,nsol)

      name1 = idmat(ireg) //idvar(msol(1)) // idvar(msol(3)) // 'kf'
      name2 = idmat(ireg) //msol(2) // 'kr'
      libeval('library/physics/' // name1, kf)
      libeval('library/physics/' // name2, kr)

      do 300 in=1,nn
         if( ifunc .ne. 0) then
            bfunc =  kf*sol(in,1)*sol(in,3) - kr*sol(in,2)
            f(in,1) = -bfunc
            f(in,2) =  bfunc
            f(in,3) = -bfunc
         endif
         if( ideriv .ne. 0) then
            dbfdb=  kf*sol(in,3)
            dbfdi=  kf*sol(in,1)
            dbfdbi= -kr
            df(in,1,1)= -dbfdb
            df(in,1,2)= -dbfdbi
            df(in,1,3)= -dbfdi

            df(in,2,1)=  dbfdb
            df(in,2,2)=  dbfdbi
            df(in,2,3)=  dbfdi

            df(in,3,1)= -dbfdb
            df(in,3,2)= -dbfdbi
            df(in,3,3)= -dbfdi
         endif
300   continue
      odef=0
      return
      end
```

- Can be in F77, C, C++, ...

Phyterm arguments

- Every phyterm gets a standard list of arguments **`arglist`** and their descriptions **`argdescrip`**.

`int *path`	whether to compute the flux (FT_RUN) or do set-up or clean-up
`int *imtx`	whether to compute the flux or its derivatives: 1=flux 10=derivative 11=both
`int *ireg`	index of region
`int *nn`	number of nodes to work on
`int *dim`	space dimension of operator
`int *nsol`	number of output variables
`int *msol`	indices of each output variables in the global list
`int *ndep`	number of input variables
`int *mdep`	indices of each input variable in the global list
`real *coord`	coordinates of points - for models which have an explicit spatial dependence (ick)
`real *sol`	the input variables, ordered with node index fast and variable index slow (ndep,nn)
`real *gradsol`	gradients of the inputs, for computing fluxes (nsol,dim,nn)
`real *f`	the output fluxes (nsol,dim,nn)
`real *df`	derivative of output fluxes with respect to inputs (ndep,nsol,dim,nn)
`real *dgf`	derivative of output fluxes with respect to input gradients (ndep,nsol,dim,dim,nn)

Database Features

- Uniform access to
  - coefficients
  - tables
  - user input
  - control options
- Easy to define new parameters in a module without reference to other sections of simulator
- Inheritance allows easy extensions
- Database can be centralized in one file or distributed over several files as desired
- In-memory modified database can be dumped and used for subsequent simulations

Inheritance & Shadowing

Before execution

```
library/physics {
   silicon {
     boron {
       Dix = 4.2
     }
   poly {
     SeeAlso "../silicon"
   }
}
```

After execution

```
library/physics {
   silicon {
     boron {
       Dix = 4.2
     }
   poly {
     SeeAlso "../silicon"
     boron {
       SeeAlso "../silicon/boron"
     }
   }
}
```

- `find_property("library/physics/poly/boron/Dix")` returns a property with value 4.2
- `find_list("library/physics/poly/boron/Dix")` returns a modified poly list on which a new Dix property can be defined, while still inheriting other properties from silicon

Top levels of hierarchy

```
database = {
  userinput {                          % user's parsed input
    substrate {                        % first command
      thick = 2.0                      % first parameter
      orientation = 0.1                % second parameter
    }
    graph {                            % second parameter
      ...
    }
  }
  options {                            % options for run
    timestep = 1
    movie = "boron,silicon"
  }
  library {
    physics {                          % physical parameters
      silicon {
        ni = 1.0e10
        boron {
          Dix = 42.0
          }
      }
    }
    math {                             % math parameters
      ...
    }
    cards {                            % list of possible
      ...                              % cards and their
    }                                  % parameters and
    cards.defaults {                   % defaults
      ...
    }
  }
}
```

Database subroutines

Library access:

findDB(pathname, noComplaint)
    look up pathname and return property

e.g.
```
findDB("library/physics/silicon/boron/Dix", 1)
findDB("solve/temperature", 1)
findDB("options/movie", 0)
```

matco(coeff, variable n., region n., dom)
    build string `library/physics/region/variable/coeff`
    and call findDB to return property

Building/accessing properties lists:

get_property(name, list)
get_local_property(name, list)
get_next_local_property(name, list, start)
put_local_property(name, list)
put_local_list(name, list)

count_properties(list)
get_property_name(list, index)
get_property_by_index(list, index)

delete_local_property(name, list)
delete_plist(name, list)

See solvecmd() for example of use.

Grid datastructure design

- Core representation is minimal and unstructured
  - Facilitates adding and removing elements, nodes
  - Each node, element, list of vertices in an element, etc, is malloc'ed separately, in any order

- Automatic routines to build inferred information from core representation
  - Internal neighbors
  - List of unique edges
  - Interface structures
  - Reorganization into similarity groups for vector access

R0,i0,i1,i2
R1,i0,i1,i2,i3
R0,i0,i1,i2,i3
R0,i0,i1,i2,i3
R1,i0,i1,i2
R1,i0,i1,i2
R0,i0,i1,i2,i3
R1,i0,i1,i2,i3

⇒

color0
R0,i0,i1,i2
R0,i0,i1,i2

color1
R0,i0,i1,i2,i3
R0,i0,i1,i2,i3

color2
R1,i0,i1,i2,i3
R1,i0,i1,i2,i3

color3
R1,i0,i1,i2
R1,i0,i1,i2

Unstructured	lock_domain ⇒	Structured	unlock_domain ⇒	Unstructured
Mesh gen'n		PDE solving		Remeshing

+Within a color, all elements have same region and type
+All indices taken from single malloc block
+Similar for coordinates, neighbors, etc

Grid datastructures (core)

- Finite element grid representation, with special handling of material boundaries

Domain

List of elements
region number, node n's, nbr n's, edge n's,

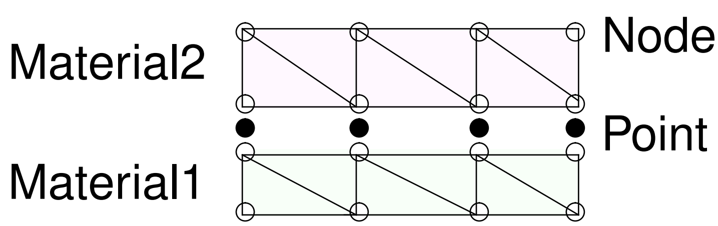
List of nodes
point number

List of points
coordinates

List of edges
node n's [2]

List of fields
name {`boron`,`potential`, . . .}
type {node,edge,element}
dimension {1,2,3}

List of region and interface names {`silicon`,`periodic`,...}



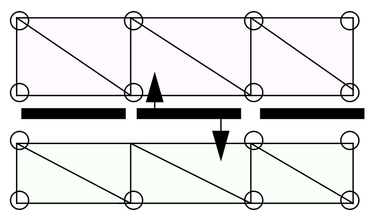
- Boundary conditions represented by external element neighbors being negative, indices into list of interface names

Grid datastructures (derived)

- Interface structures can be generated on demand

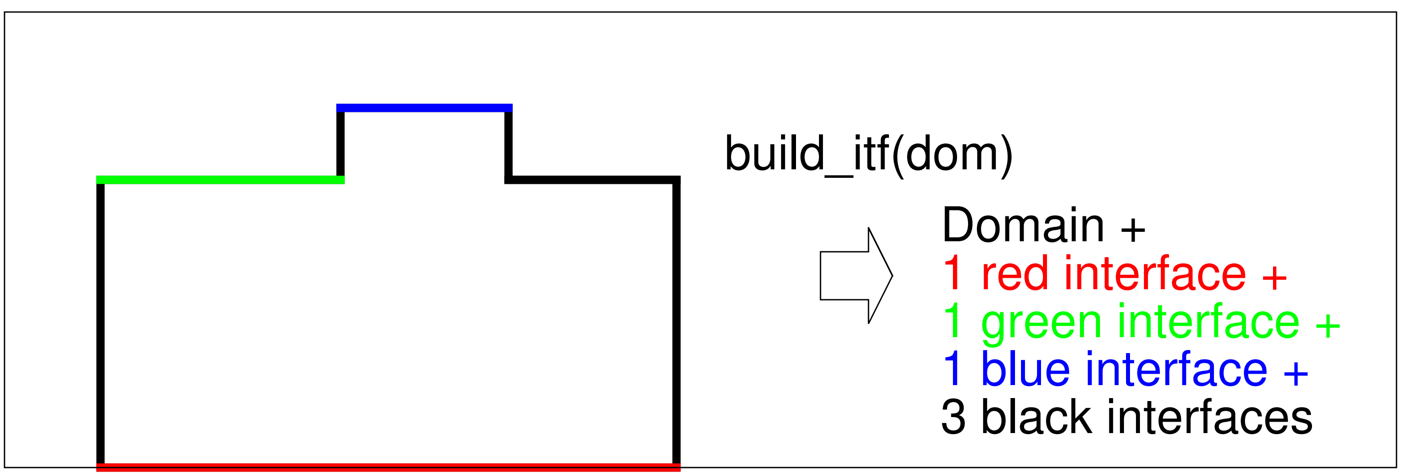
Domain

...
List of interfaces
- Reg n's [2]
- Element count nel
- Node count nno
- List of elements [nel*2] and faces[nel*2]
- List of nodes [nno*2]



- Each "element" of interface identifies which bulk element and which face of that element are on the interface

- Each "node" of interface lists the two corresponding nodes



Grid subroutines

Alloc_domain()
Alloc_domain_region(dom, name)
Alloc_domain_surface (dom, name)
ixmaterial(dom, name) - index of material in region (first if multiple)
ixsurface(dom, name) - index of surface in region (first if multiple)

alloc_node (dom, n)
alloc_point (dom, n)
alloc_edge (dom, n)
alloc_element (dom, n, type, ireg)

alloc_field(dom, type, name, dim, initvalue)
free_field(dom, index)
ixfield(dom, name)

build_nds(dom) - if given elements in terms of points, convert to node description (convenient for reading from conventional finite element grid)
build_edge(dom) - given elements and nodes, build list of unique edges
build_nbrs(dom) - given elements, build internal neighbors
build_itf(dom) - build list of unique interfaces
lock_domain(dom, andColor) - reorder for vector access
unlock_domain(dom)

domain_sanity_check(dom)

Solution flowchart

```
solvecmd
        make_diff_list
        steptime
                solcontrol
                diff_setup
                        lock_domain
                        build_itf
                        shadowDom
                        variable-specific initializa-
tion
                                usually matrix_init()
                        boxGeom
                        boundary_cond
                trbdf2_step
                        OuterGS
                                setFlux(FT_DT)
                                innerNewton
                                        assemble
                                        mtxscl
                                        c2bls
                                        update_vars
                        OuterGS
                        OuterGS
                        milne
```

Assembly Flowchart

```
assemble
       assemble_alloc
       assemble_space
              assemble_eliminate
              assemble_elementals
                     call divtrm to discretize laplacian
                            interpolate to quad point
     each element           call a phyterm
     color                  store to window
                     call addstf to transfer window to matrix
              assemble_nodals
                     pdediag for each region
                            copy variables to contiguous storage
     each region            call a phyterm
                            store to rhs, matrix
              assemble_box
                            interpolate to midpoints
     each edge              call a phyterm
     color                  store to rhs, matrix
              assemble_itf_nodals
                            similar
       assemble_time
       condenseItf
```

PDE description - internal

- Output of solcontrol

Database description → solcontrol → Internal description

globsolstruc: for each field of grid

solmeth

- sol_type {general, pdesteady, pdetransient, formula}
- setup routine (e.g. special Tdep initialization for interstitials)
- refresh routine
- teardown routine
- block of pde's u.pb

pdeblock

- nsol number of solution variables
- sol list of solution variables
- nterm number of terms
- term[0]

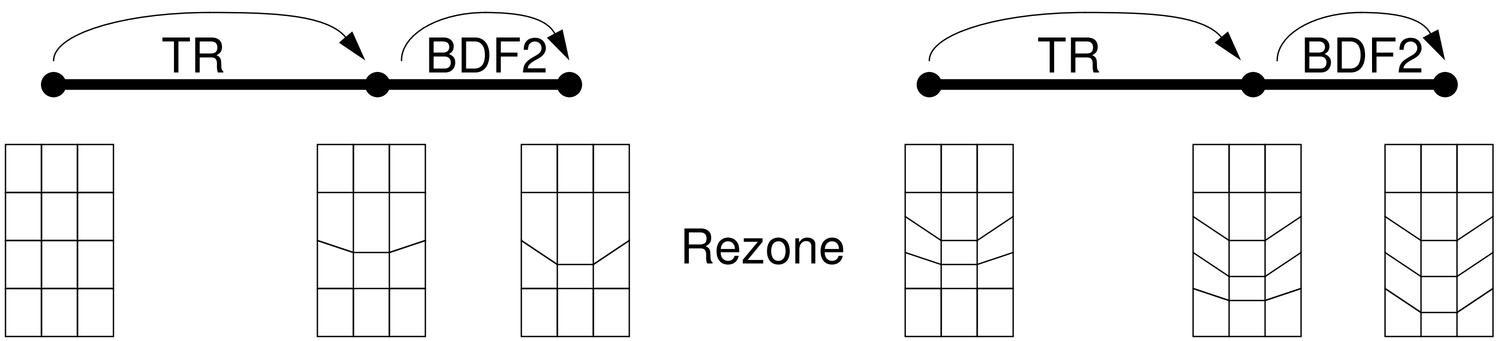
pdeterm

- geoterm
- phyterm
- number and list of input variables
- number and list of output equations
- output-input coupling flags
- between[ni] - which interfaces (if BC term)

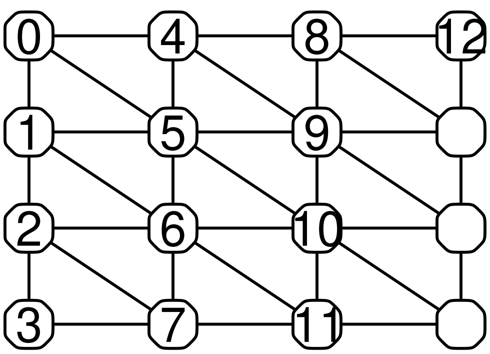
- term[1]...
- mtxtype - block sparsity array
- linear solution options

Timestep layout

- Grid structural changes happen between timesteps
- Grid motion happens during timestep
- "Timestep" is composite TRBDF2 step



Sparse Matrix Structure
(Bank/Smith)



Grid

$$\frac{\partial B}{\partial t} = D_B \nabla^2 B + f(B, \psi)$$

$$\frac{\partial C}{\partial t} = D_C \nabla^2 C + g(C, \psi)$$

$$\varepsilon \nabla^2 \psi = h(B, C)$$

Equations

- Represent graph structure only once

- Represent PDE structure by heirarchy

ia 17 20 23 14 5 2 5 6 3 6 7...

0 16 17

pointers connections

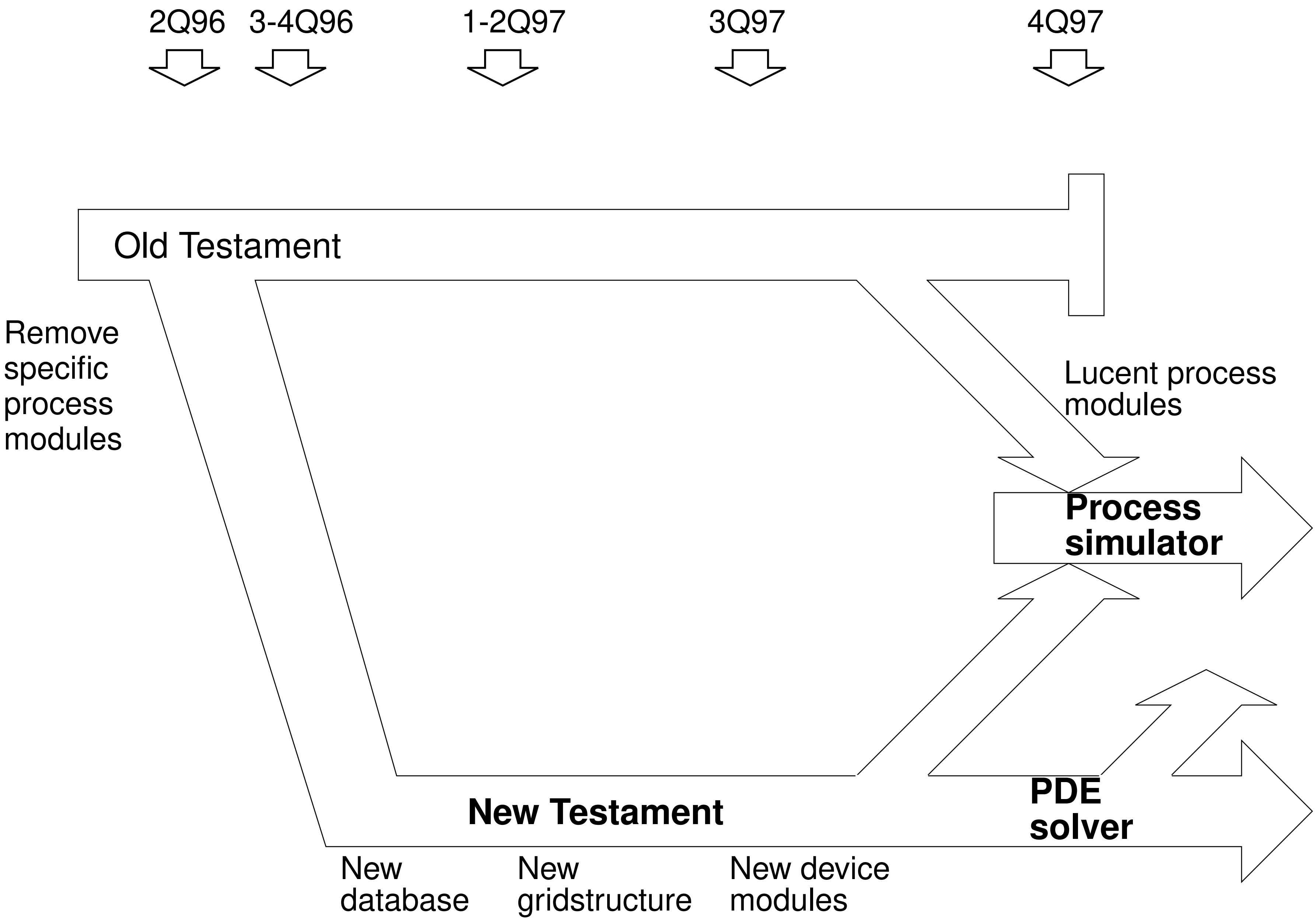
real storage for one block "aa"

diagonal z upper triangle lower triangle

locar[i+nb*j] is location of matrix for ith equation, jth variable
All storage taken from single matrix array a
e.g. diagonal #10 of C equation wrt B variable is a[locar[1]+10]

Right hand side rhs stored B0,C0,psi0,B1,C1,psi1,...

PROPHET Genealogy



Directory structure

src/			
	Dbase/		database implementation
	Grid/		grid implementation
	Guide/		
	Main/		input parser
	Misc/		common macros
	Mod/		modules
		Bias/	device simulation
		Boundary/	define BC's
		Dbase/	modify database contents
		Dump/	raw output
		Field/	define fields over grid
		Graph/	1D/2D graphics
		Grid/	define working grid
		Solve/	call solver
		xgraph/	X-based graphics tool
	PDE/		
		Casmbly/	assembly control
		Fasmbly/	discretization
		Fluxes/	phyterms
	Vexpr/		expression parser
	lib/		
		dbase.prophet	text of database
	arch/		.a and .exe - per architecture

Colored = soon to be replaced by TCL/TK