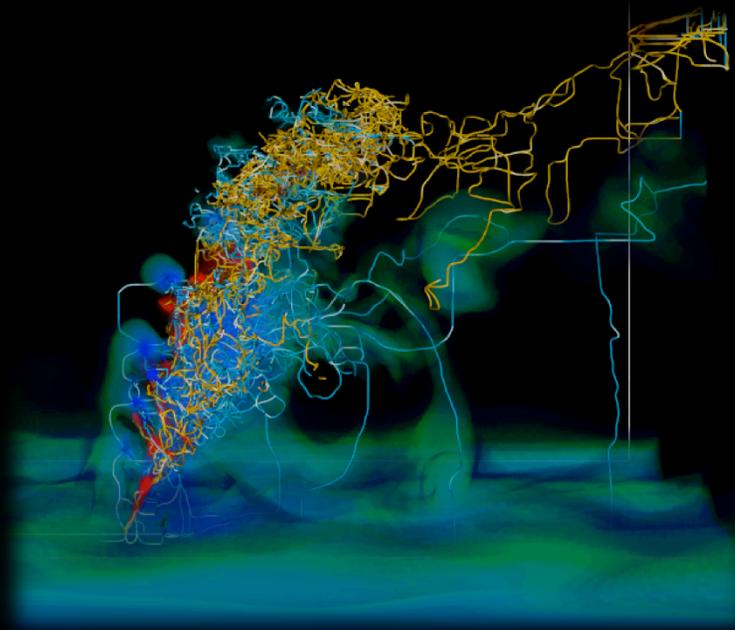


# Do-It-Yourself Data Analysis: Selected Topics and Recent Adventures (or, ten ways to win friends and parallelize data analysis)

Morse-Smale  
Complex of  
combustion in  
the presence of  
a cross flow



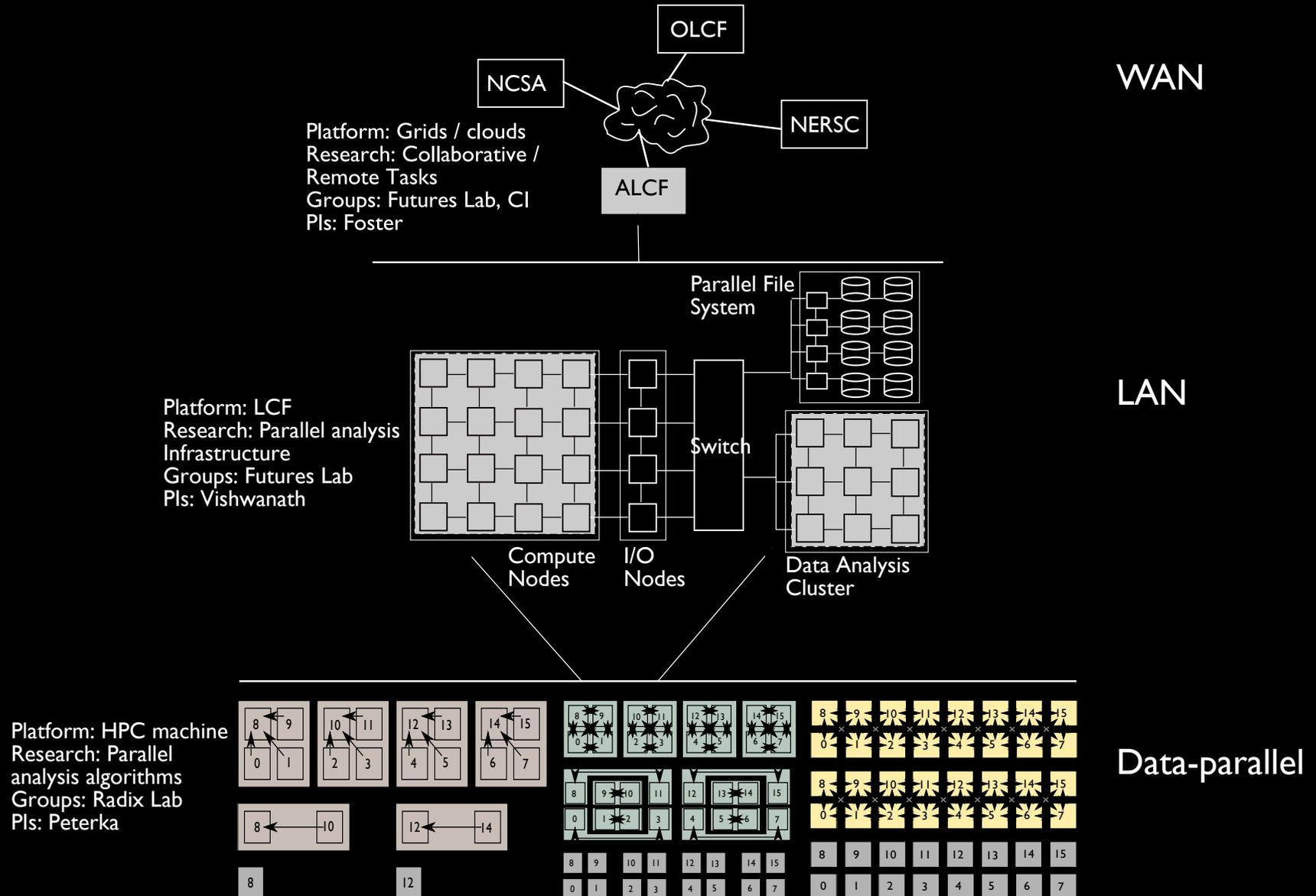
Tom Peterka

[tpeterka@mcs.anl.gov](mailto:tpeterka@mcs.anl.gov)

Mathematics and Computer Science Division

# Argonne Data Science

Several levels of data movement for data-intensive science applications



# Executive Summary

DIY helps the user write data-parallel analysis algorithms.

## Main ideas and Objectives

- Large-scale parallel analysis (visual and numerical) on HPC machines
- Scientists, visualization researchers, tool builders
- In situ, coprocessing, postprocessing
- Data-parallel problem decomposition
- Scalable data movement algorithms

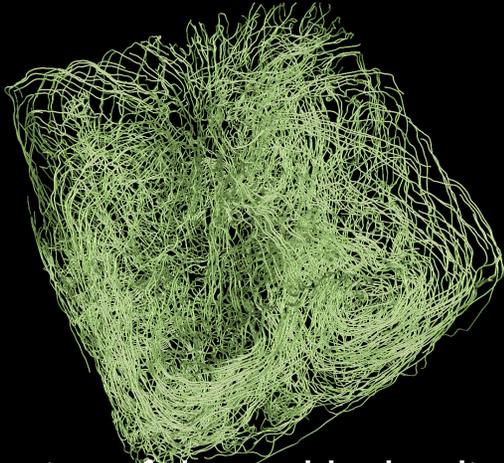
## Benefits

- Researchers can focus on their own work, not on parallel infrastructure
- Analysis applications can be custom
- Reuse core components and algorithms for performance and productivity

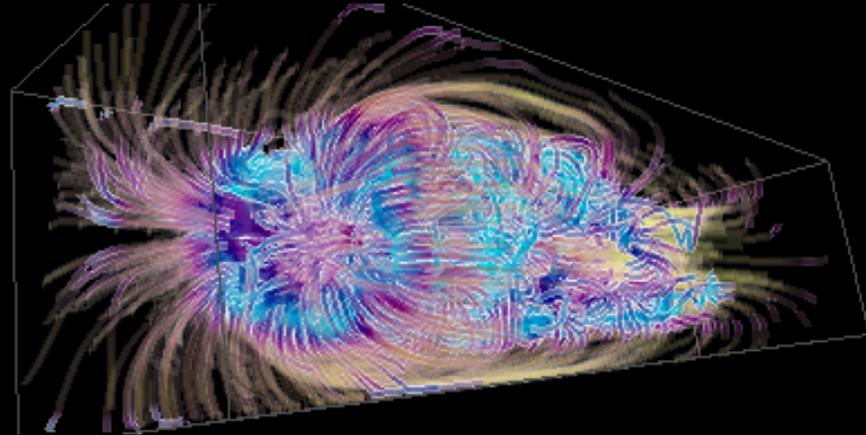
## Today's talk

- Main design concepts, that include both
  - A DIY overview, as well as
  - Recent advances and new ideas

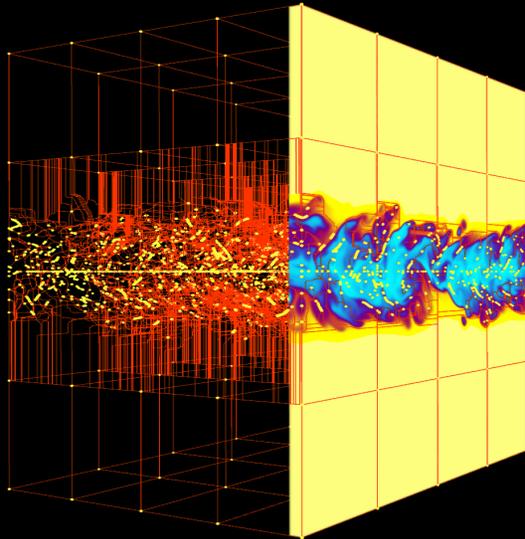
# Observation #1: Data Analysis Comes in Many Flavors



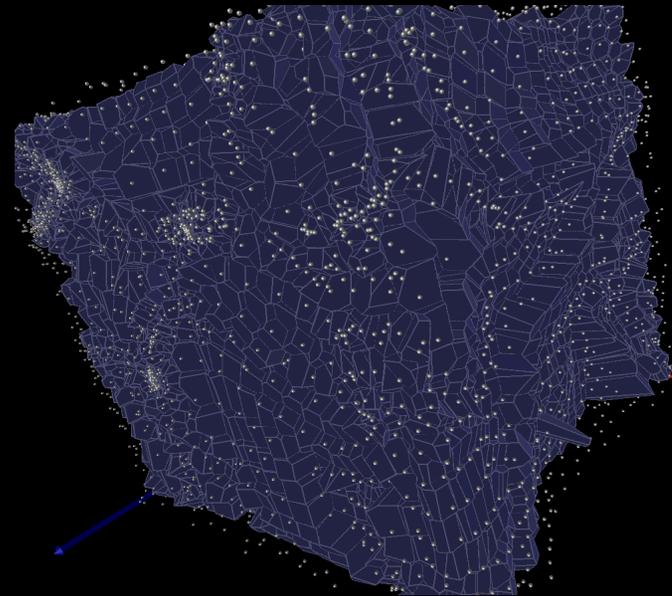
Particle tracing of thermal hydraulics flow



Information entropy analysis of astrophysics



Morse-Smale complex of combustion



Voronoi tessellation of cosmology

# #1: Separate Analysis Ops from Data Ops

Analysis	Application	Application Data Model	Analysis Data Model	Analysis Algorithm	Communication	Additional
Particle Tracing	CFD	Unstructured Mesh	Particles	Numerical Integration	Nearest neighbor	File I/O, Domain decomposition, process assignment, utilities
Information Entropy	Astrophysics	AMR	Histograms	Convolution	Global reduction, nearest neighbor	
Morse-Smale Complex	Combustion	Structured Grid	Complexes	Graph Simplification	Global reduction	
Computational Geometry	Cosmology	Particles	Tessellations	Voronoi	Nearest neighbor	

You do this yourself

Can use serial libraries such as OSUFlow, Qhull, VTK  
(don't have to start from scratch)

DIY handles this

# DIY Overview

## Features

Parallel I/O to/from storage

- MPI-IO, BIL

Domain decomposition

- Decompose domain
- Describe existing decomposition

Network communication

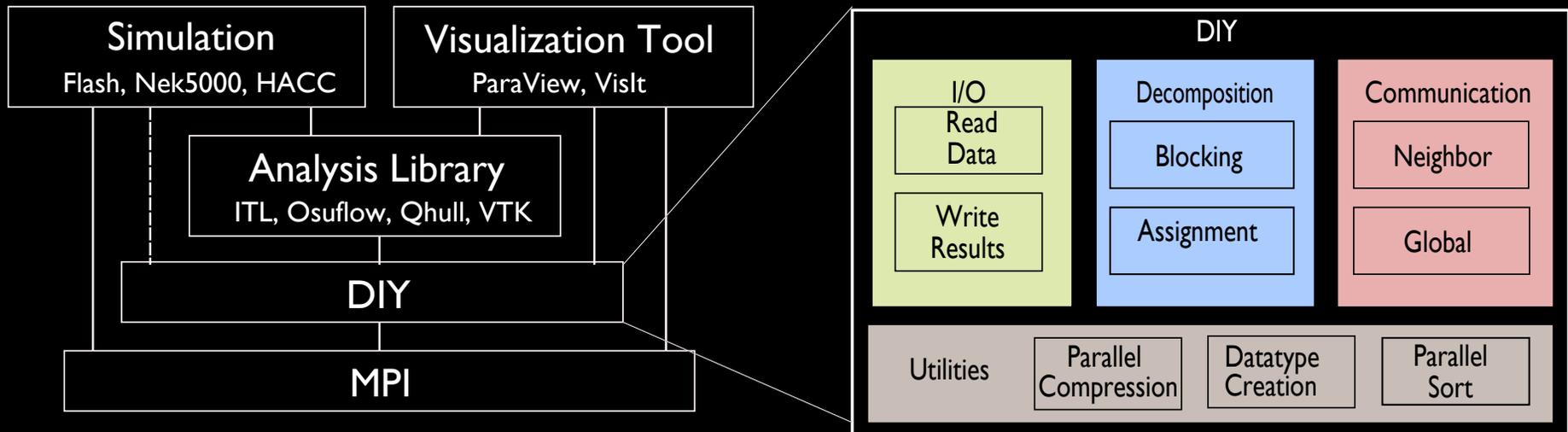
- Global reduction (2 flavors)
- Local nearest neighbor

## Library structure

Written in C++

C bindings

Future Fortran bindings



DIY usage and library organization

# Observation #2: Application Data Model $\neq$ Analysis Data Model

## HACC (cosmology) Data Model

```
int num_particles;  
float *xx, *yy, *zz;  
float *vx, *vy, *vz;  
float *phi;  
int64_t pid;  
uint16_t mask;
```

Corollary: analysis X data  
model  $\neq$  analysis Y data  
model

## Tess (voronoi tessellation) Data Model

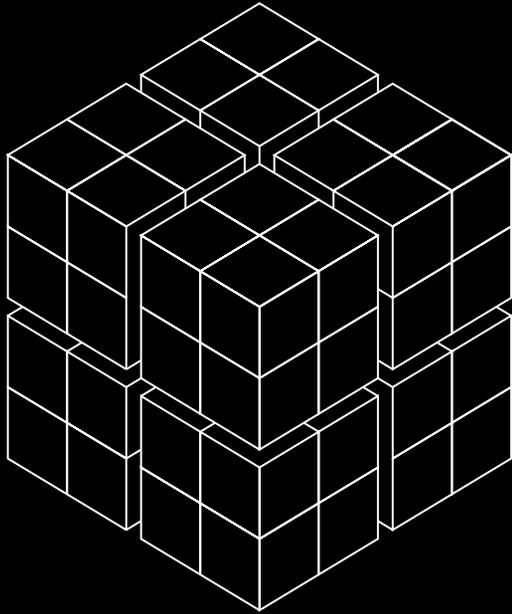
```
float mins[3];  
float maxs[3];  
int num_verts;  
int num_cells;  
double *verts;  
int *num_cell_verts;  
int tot_num_cell_verts;  
int *cells  
double *sites;  
int num_complete_cells;  
int *complete_cells;  
double *areas;  
double *vols;  
int tot_num_cell_faces;  
int *num_cell_faces;  
int *num_face_verts;  
int tot_num_face_verts;  
int *face_verts;
```

## #2: Allow User to Define Data Model

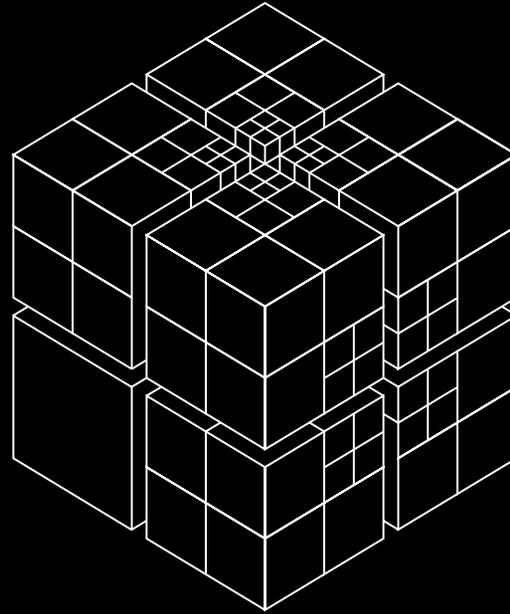
C data structure	DIY Datatype type;	DIY data type
	struct map_block_t map[] = {	
float mins[3];	{ DIY_FLOAT, OFST, 3, offsetof(struct vblock_t, mins)	},
float maxs[3];	{ DIY_DOUBLE, ADDR, v->num_verts * 3, DIY_Addr(v->verts)	},
double *verts;	{ DIY_DOUBLE, ADDR, v->num_cells * 3, DIY_Addr(v->sites)	},
double *sites;	{ DIY_INT, ADDR, v->num_complete_cells, DIY_Addr(v->complete_cells)	},
int *complete_cells;	{ DIY_DOUBLE, ADDR, v->num_complete_cells, DIY_Addr(v->areas)	},
double *areas;	{ DIY_DOUBLE, ADDR, v->num_complete_cells, DIY_Addr(v->vols)	},
double *vols;	{ DIY_INT, ADDR, v->num_complete_cells, DIY_Addr(v->num_cell_faces)	},
int *num_cell_faces;	{ DIY_INT, ADDR, v->tot_num_cell_faces, DIY_Addr(v->num_face_verts)	},
int *num_face_verts;	{ DIY_INT, ADDR, v->tot_num_face_verts, DIY_Addr(v->face_verts)	},
int *face_verts;	{ DIY_FLOAT, OFST, 3, offsetof(struct vblock_t, maxs)	},
	};	
	DIY_Create_struct_datatype(DIY_Addr(vblock), 10, map, dtype);	

- Any C/C++/Fortran data structure can be represented as an DIY (MPI) data type
- DIY uses data type to fetch data directly from memory or storage
- User does not pack / unpack (serialize / deserialize) data
- Zero copy at application level saves time and space
- DIY helps make data type creation easier

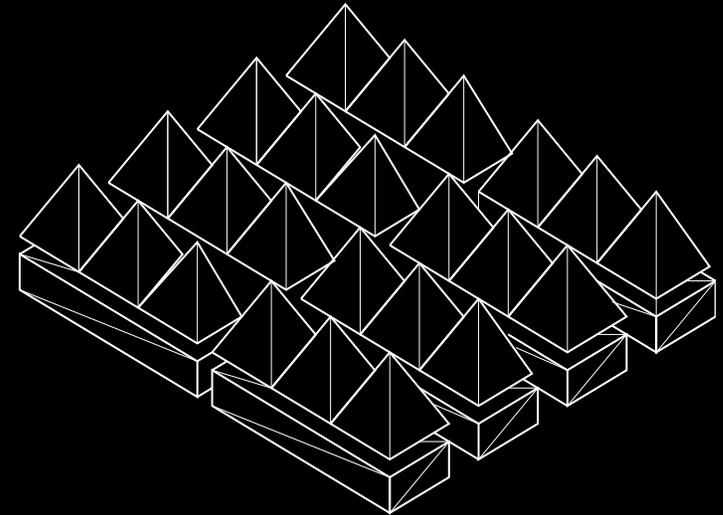
### #3: Group Data Items Into Blocks



Structured Grid



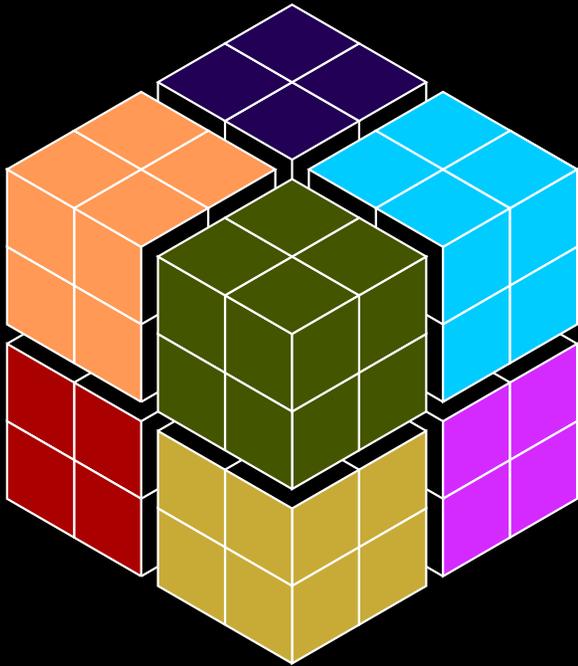
AMR Grid



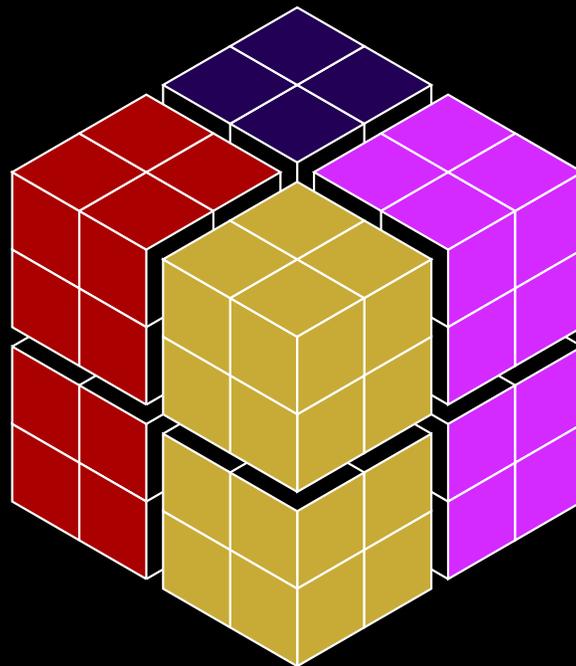
Unstructured Mesh

The block is DIY's basic unit of data. Original dataset is decomposed into generic subsets called blocks, and associated analysis items live in the same blocks. Blocks contain one or more instances of the data type described earlier.

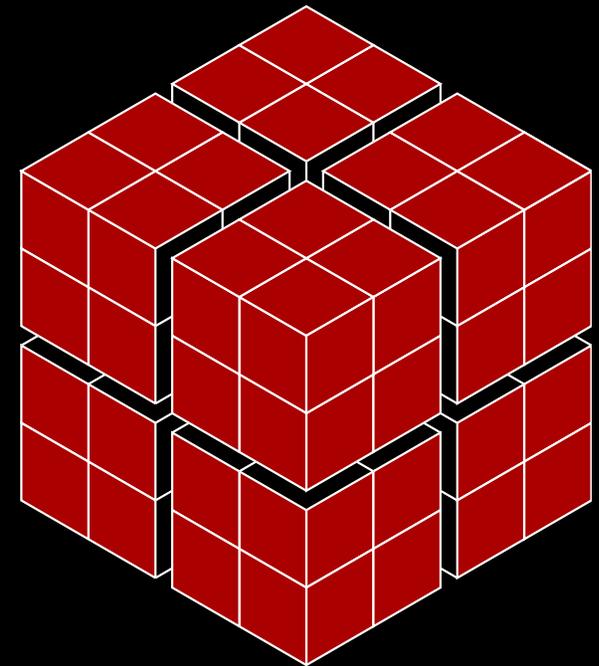
## #4: Blocking $\neq$ Process Assignment



8 processes



4 processes

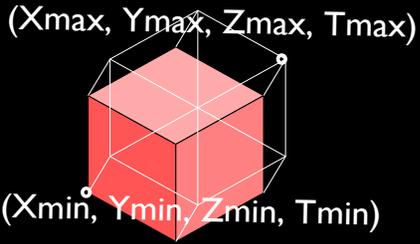
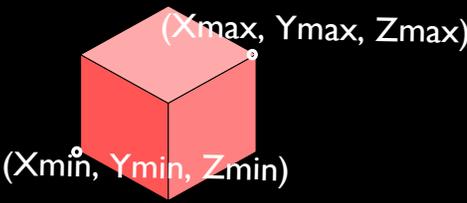
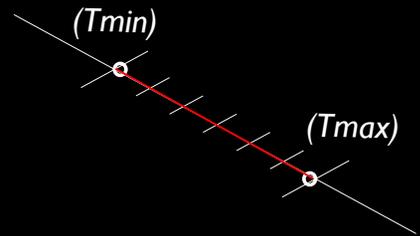
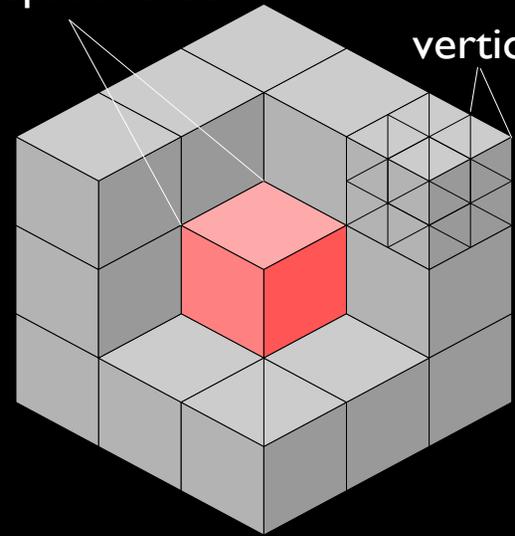
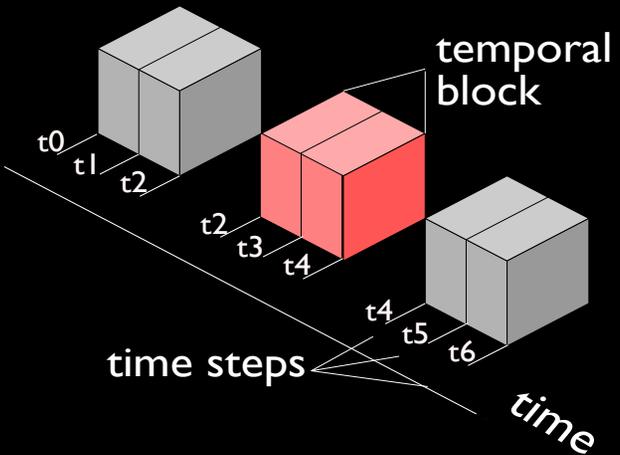


1 process

All data movement operations are per **block**; blocks exchange information with each other using DIY's communication algorithms. DIY manages and optimizes exchange between processes based on the process assignment. This allows for flexible process assignment as well as easy debugging.

# #5: Time is Like Space, but Special

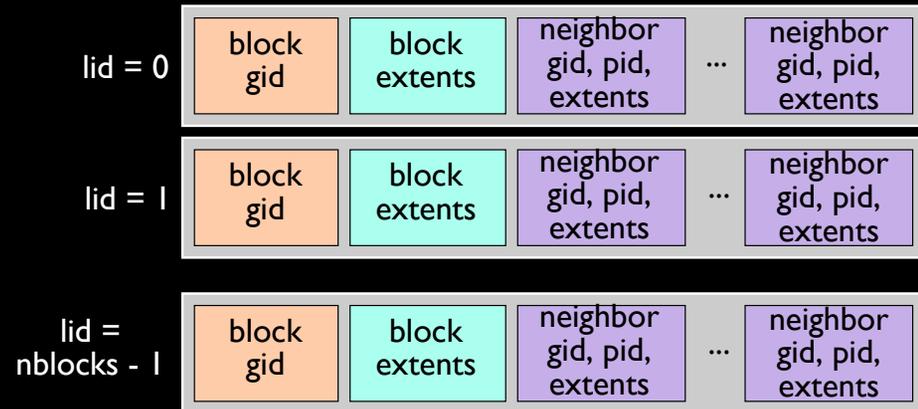
- Time often goes forward only
- Usually do not need all time steps at once

4D	=	3D	×	1D
<p>(<math>X_{max}, Y_{max}, Z_{max}, T_{max}</math>)</p>  <p>(<math>X_{min}, Y_{min}, Z_{min}, T_{min}</math>)</p> <p>4D Block</p>		<p>(<math>X_{max}, Y_{max}, Z_{max}</math>)</p>  <p>(<math>X_{min}, Y_{min}, Z_{min}</math>)</p> <p>3D Spatial Extent</p>		 <p>1D Temporal Extent</p>
<p>4D Neighborhood (not drawn)</p>		<p>spatial block</p>  <p>vertices</p> <p>3D Spatial Neighborhood</p>		 <p>temporal block</p> <p>time steps</p> <p>time</p> <p>1D Temporal Neighborhood</p>

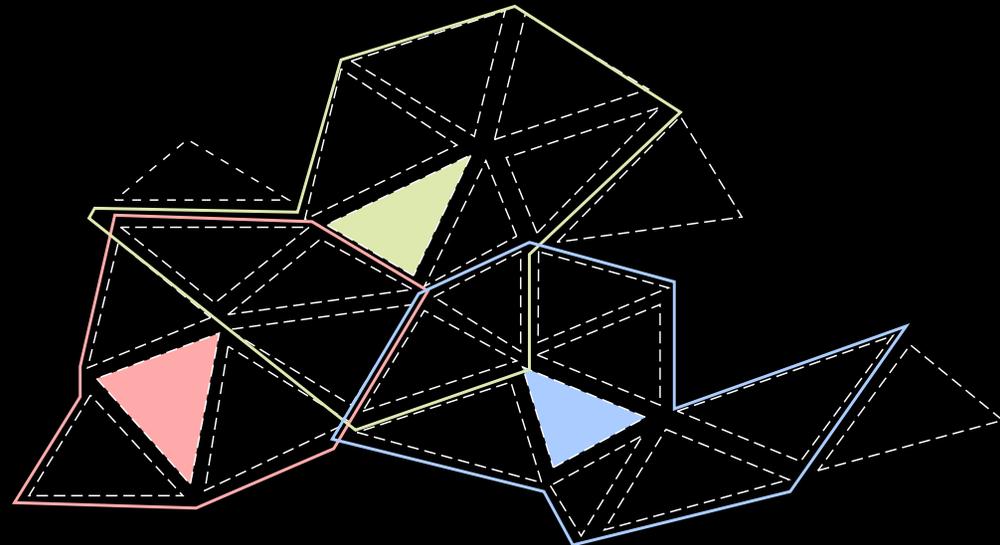
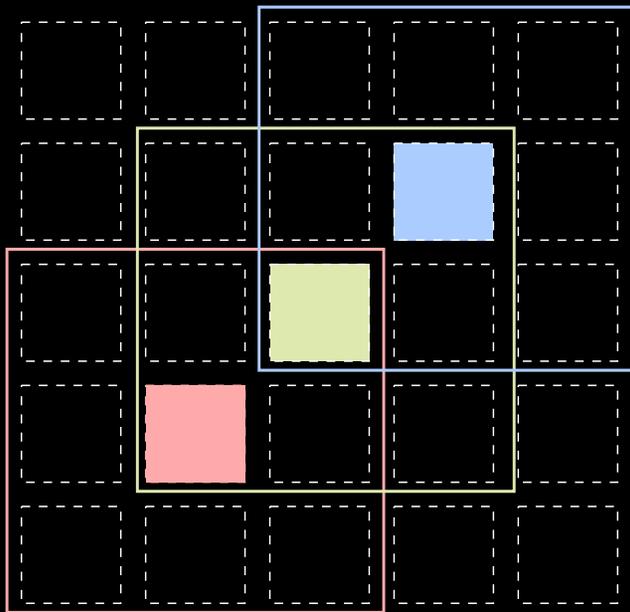
Hybrid 3D/4D time-space decomposition. Time-space is represented by 4D blocks that can also be decomposed such that time blocking is handled separately. <sub>11</sub>

# #6: Group Blocks into Neighborhoods

- Limited-range communication
- Allow arbitrary groupings
- Distributed, local data structure and knowledge of other blocks (not master-slave global knowledge)

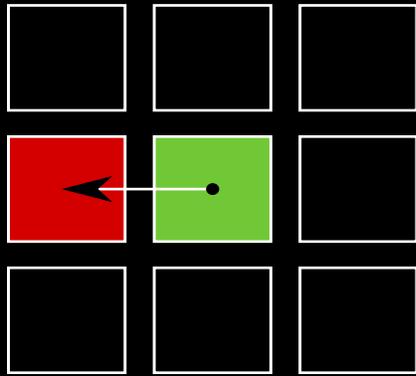


gid = global block identification  
lid = local block identification  
pid = process identification

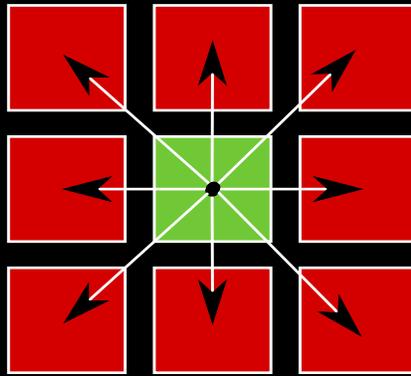


Two examples of 3 out of a total of 25 neighborhoods

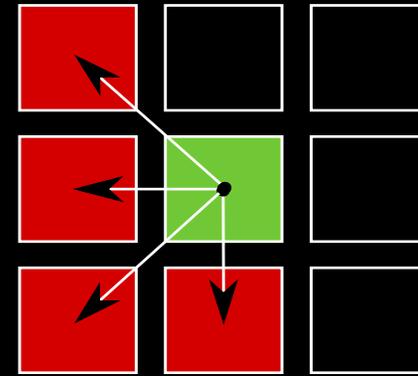
# #7: Provide Different Neighborhood Communication Patterns



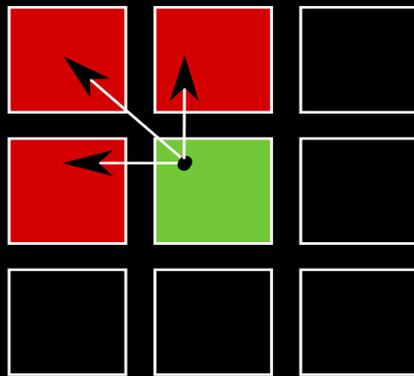
DIY\_Enqueue\_item\_pt()  
DIY\_Enqueue\_item\_mask()



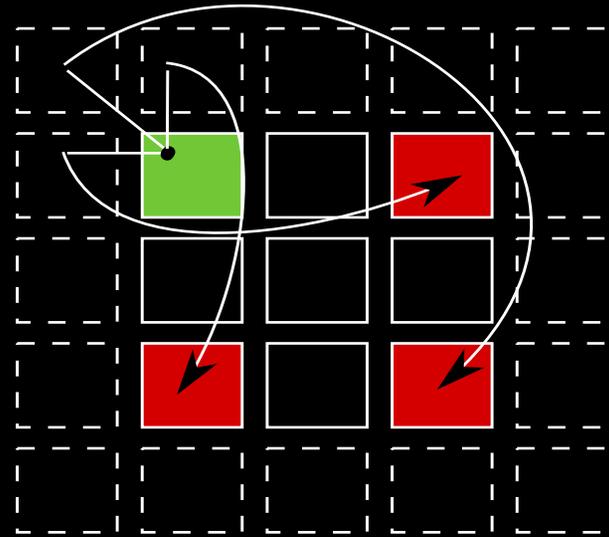
DIY\_Enqueue\_item\_all()



DIY\_Enqueue\_item\_half()



DIY\_Enqueue\_item\_all\_near()  
DIY\_Enqueue\_item\_half\_near()



Support for wraparound neighbors  
(repeating boundary conditions)

DIY provides point to point and different varieties of collectives within a neighborhood via its enqueue\_item mechanism. Items are enqueued and subsequently exchanged (2 steps).

## #8: Make Global and Neighborhood Communication Fast and Easy

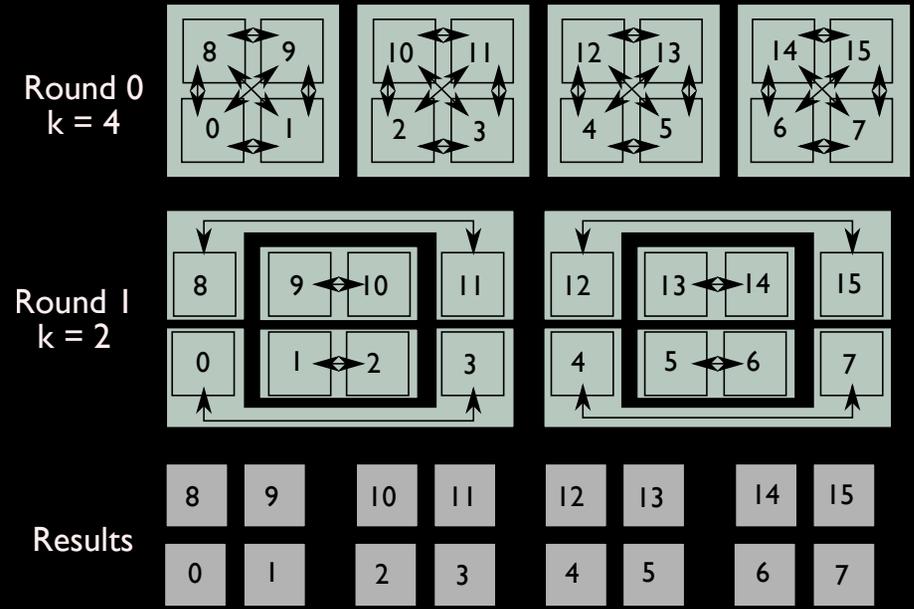
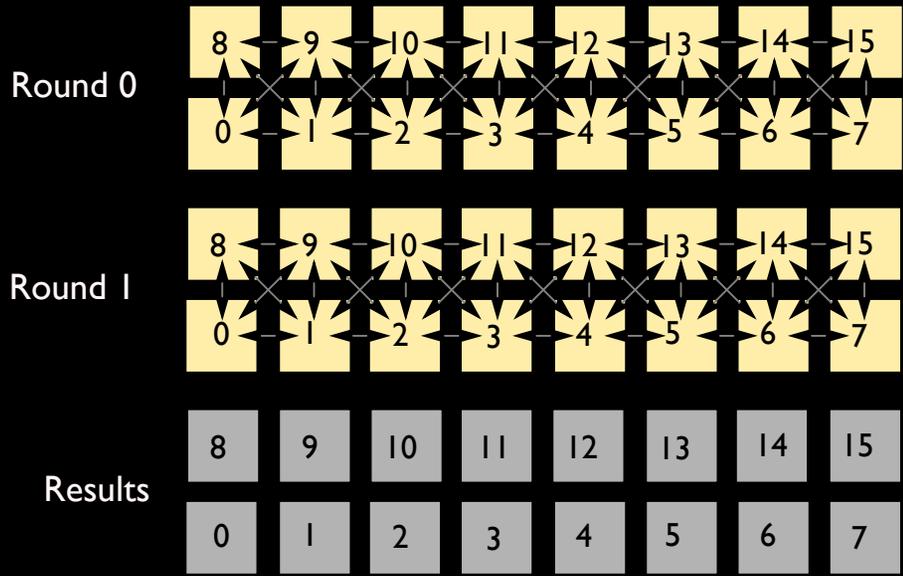
Analysis	Communication
Particle Tracing	Nearest neighbor
Global Information Entropy	Merge-based reduction
Point-wise Information Entropy	Nearest neighbor
Morse-Smale Complex	Merge-based reduction
Computational Geometry	Nearest neighbor
Region growing	Nearest neighbor
Sort-last rendering	Swap-based reduction

Factors to consider when selecting communication algorithm:

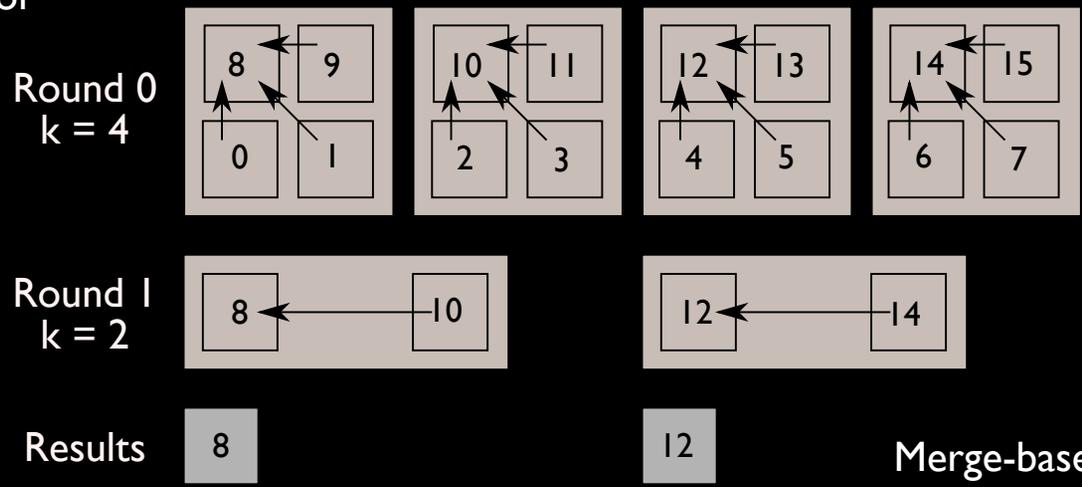
- associativity
- number of iterations
- data size vs. memory size
- homogeneity of data

DIY provides 3 efficient scalable communication algorithms on top of MPI. May be used in any combination.

# 3 Communication Patterns



## Nearest neighbor



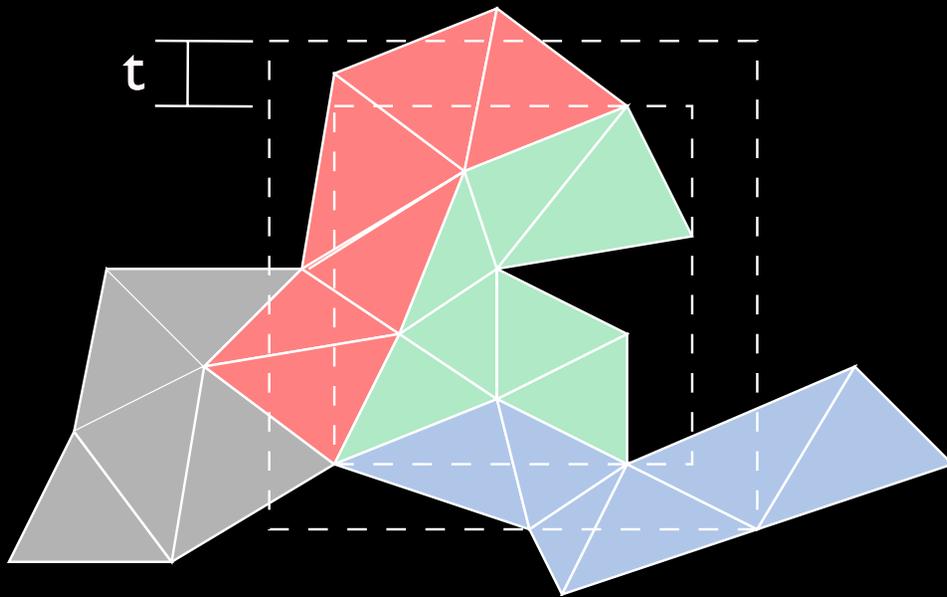
## Swap-based reduction

## Merge-based reduction

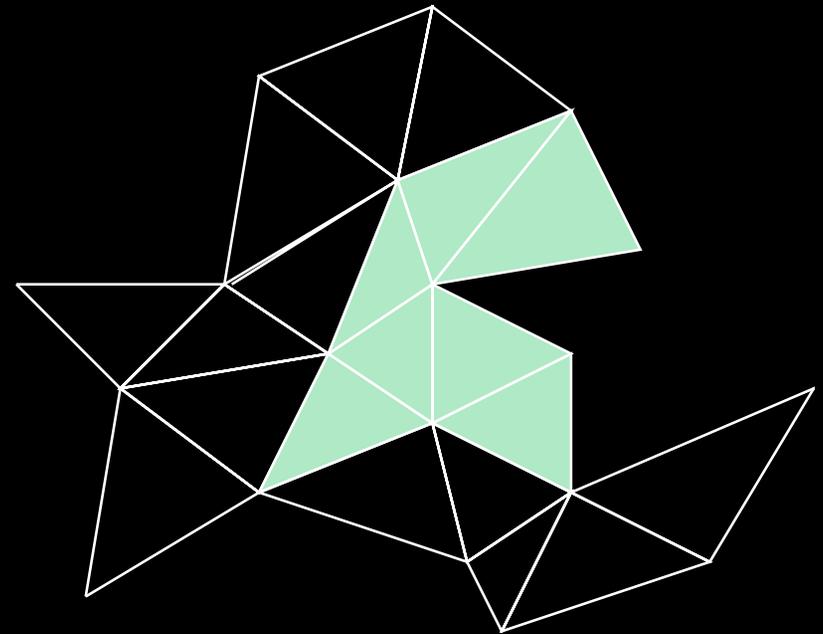
## #9: Support Applications

### In Situ Unstructured Spectral Meshes With Help from MOAB

- Decomposition assigned by the application, not DIY
- DIY needs to get the decomposition from the app
- Call on MOAB for help with connectivity



Given the above mesh, assume the green block wants ghost cells in a given ghost radius of size  $t$ .

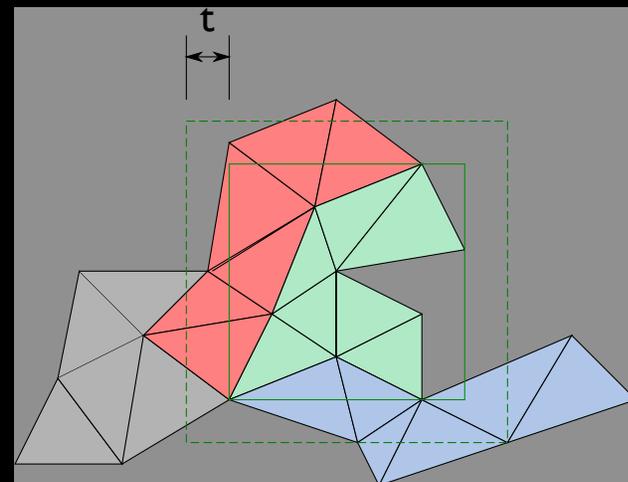


Result: the green block will have these cells (original green cells plus transparent cells)

## #9 Continued

```
void foo(imesh *mesh) { // MOAB mesh
  DIY_Init(num_blocks);
  for (num_blocks) {
    // query MOAB for verts in block
    get_adjacencies(hex, adj_verts);
    BlockBounds(bounds); // find min/max of verts
    // query MOAB for local neighbors of vertices
    get_adjacencies(adj_verts, adj_hexes);
    store adj_hexes in neighbors, num_neighbors
    // query MOAB for remote neighbors
    get_sharing_data(adj_verts, remote_handles,
      remote_procs);
    remote_data = remote_handles, remote_procs;
    // query MOAB for local vertex ids
    loc_vids[block] =
      id_from handle(shared_adj_verts);
  }
  DIY_Decomposed(blocks, bounds, remote_data,
    num_remote_data, loc_vids, neighbors,
    num_neighbors);
}
```

```
while (!done) {
  for (cells) {
    for (neighbors) {
      if (cell intersects neighbor extents + t &&
        cell was not sent already &&
        cell did not come from neighbor)
        post cell to neighbor;
    }
  }
  num_recvd = DIY_Exchange_neighbors();
  done = DIY_Check_done_all(!num_recvd);
}
```



## #10: Work With Other Libraries

### We are helped by:

**Zoltan**-partitioning for dynamic load balancing

**MOAB**-unstructured mesh management

**HDF5** and **parallel netCDF**-high level storage

**MPI** (of course)

### We can help with:

**ITL**-Information theoretic analysis

**MSC**-Morse-Smale analysis

**OSUFlow**-Particle tracing

**Qhull**-Computational geometry

**VTK**-Visualization and analysis filters

## Do-It-Yourself Data Analysis: Selected Topics and Recent Adventures

### Acknowledgments:

#### Facilities

Argonne Leadership Computing Facility (ALCF)  
Oak Ridge National Center for Computational Sciences (NCCS)

#### Funding

DOE SDMAV Exascale Initiative  
DOE Exascale Codesign Center

[http://www.mcs.anl.gov/~tpeterka/  
software.html](http://www.mcs.anl.gov/~tpeterka/software.html)

<https://svn.mcs.anl.gov/repos/diy/trunk>

Tom Peterka

[tpeterka@mcs.anl.gov](mailto:tpeterka@mcs.anl.gov)