SAND2011-0814C

STK-Mesh Tutorial Minisymposium

SIAM Conference on
Computational Science and Engineering
February 28, 2011

H. Carter Edwards, Todd Coffey,
Daniel Sunderland, and Alan Williams
Sandia National Laboratory

Sandia National Laboratories

Four Parts:

- **Part 1: STK-Mesh Domain Model**
  - Comprehensive conceptual overview; no code
- **Part 2: STK-Mesh Computations**
  - How to perform computations; with code snippets
- **Part 3: STK-Mesh Modifications**
  - How to modify a mesh; with code snippets
- **Part 4: SIERRA Toolkit – beyond STK-Mesh**
  - Other modules / library components

STK-Mesh Domain Model

SIAM Conference on
Computational Science and Engineering
February 28, 2011

H. Carter Edwards
Sandia National Laboratory

Sandia National Laboratories

What is a *Domain Model*?

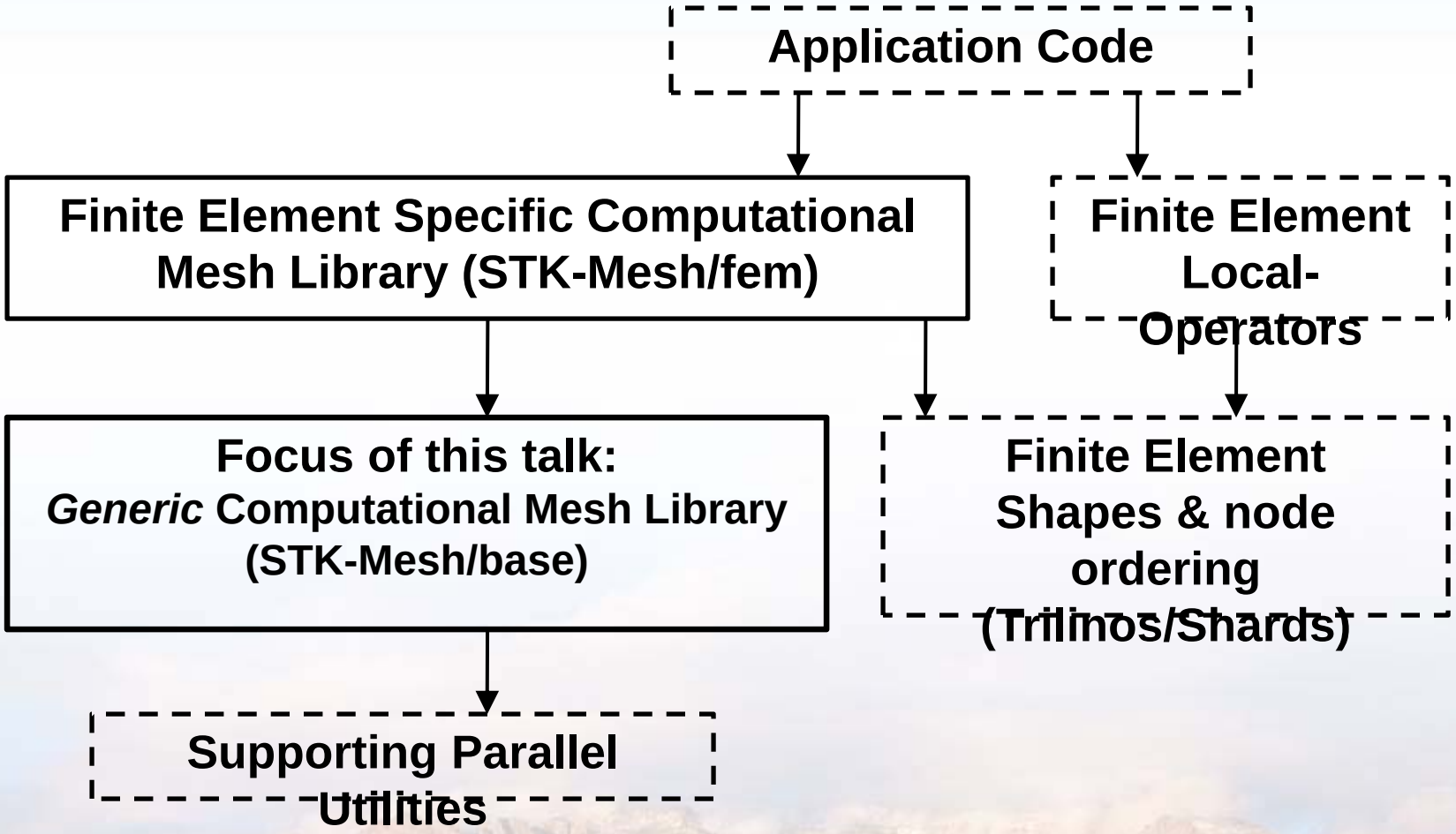
- **A formal expression of, and ubiquitous language for,**
  - Conceptual context
  - Software architecture
  - Software requirements framed within the conceptual context and software architecture context

- **"Domain-Driven Design; Tackling Complexity in the Heart of Software" by Eric Evans**
  - I highly recommend to developers of non-trivial CS&E software
  - Systematic and iterative approach to modeling a domain
  - Emphasis on managing complexity and mapping to software

The Challenging STK-Mesh *Domain*

- **An Unstructured Mesh**
  - "Weblike pattern or construction that fills a spatial domain Ω"
  - A discretization of a spatial domain Ω
  - Filled with arbitrary elemental subdomains; e.g., polyhedrons
- **Supporting Computations**
  - Simulations with multiphysics / heterogeneous phenomena
  - E.g., numerical solution of PDEs defined on the domain Ω
  - Computational data associated with entities of the discretization
- **That use Advanced Capabilities and Algorithms**
  - Massively parallel and hybrid parallel computations
  - Solution strategies that adapt the computations & discretization

Modular, Layered Architecture

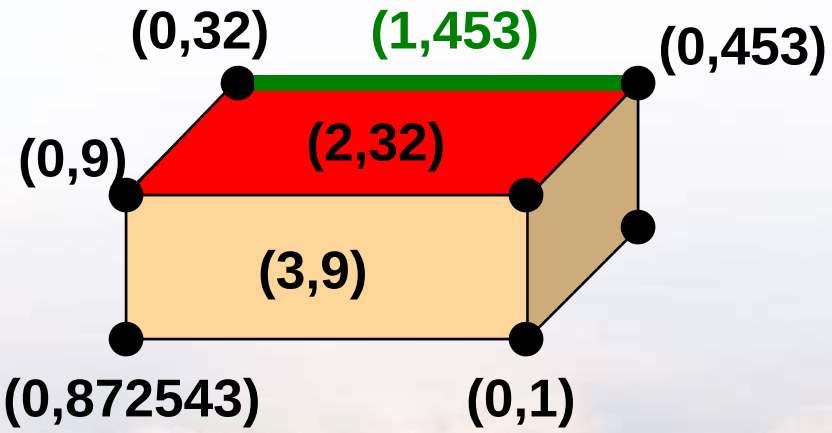


Computational Mesh Database

- **Computational Mesh**
  - Discretization data; e.g., nodes, elements, connectivity
  - Computation data; e.g., variables of the PDEs / models
- **Mesh Database Bulk-data**
  - Discretization and computation data of the problem to be solved
  - Size is proportional to the granularity of the discretization
  - Must be parallel-distributed for scalable computations
- **Mesh Database Meta-data**
  - *Description* of the bulk-data (a.k.a., the database's schema)
  - Size is proportional to the complexity of the discretization
  - Assumed to be parallel-duplicated for simplicity

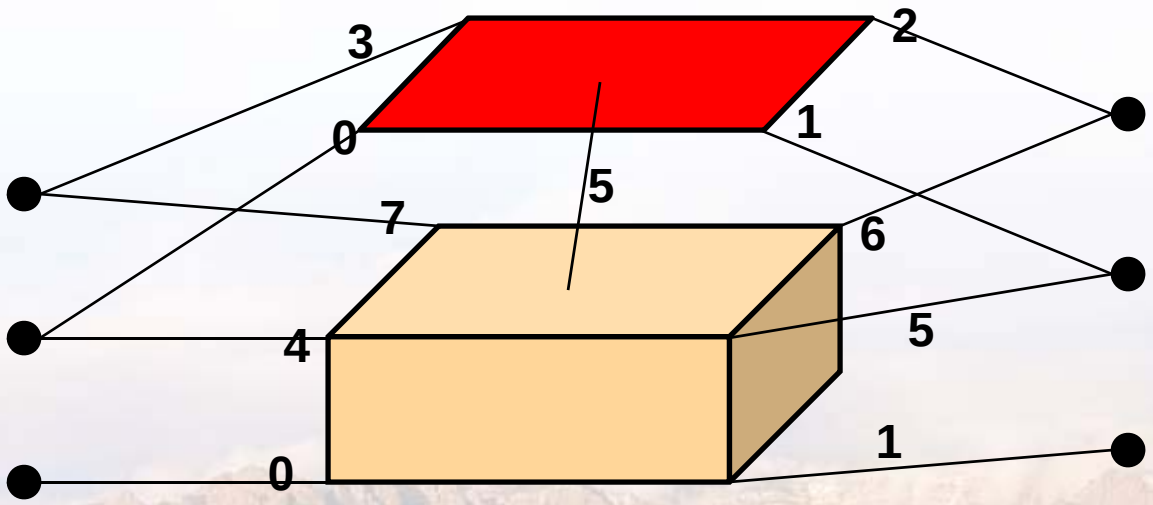
Mesh Bulk-Data: Discretization

- **Mesh Entity:** $entity_{GlobalID}^{Rank}$
  - The fundamental, atomic units of the discretization
  - E.g., finite element method's nodes, edges, faces, and elements
  - Ranking within the discretization; e.g., node < element
  - Globally unique and persistent identifier
    - Across parallel-distributed memory space
    - For the lifetime of the entity
    - Fully ordered
  - Unique: (Rank , Global-ID)



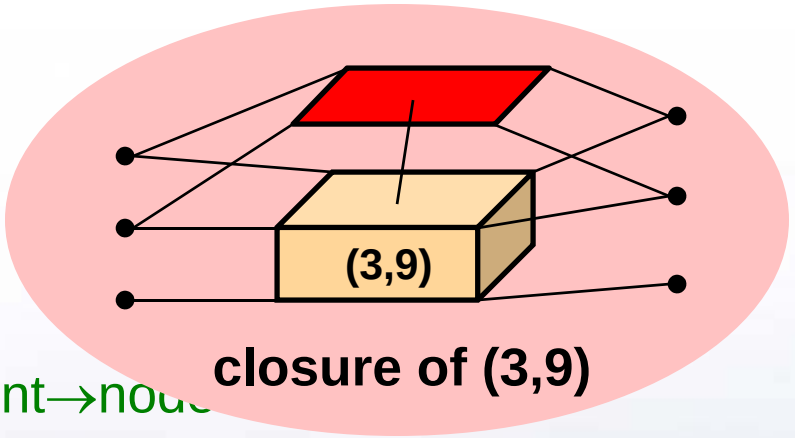
Mesh Bulk-Data: Discretization

- **Mesh Entity Relation:** $\left(entity_a^J, entity_b^K, relationID\right)$
  - Directed from higher to lower ranking entity
    - Rank J > Rank K : $entity_a^J \rightarrow entity_b^K$; required J ≠ K
  - E.g., an element is defined to be higher ranking than a node
  - Uniqueness when J > K: $\left(entity_a^J, K, relationID\right) \rightarrow entity_b^K$
    - … or does not exist



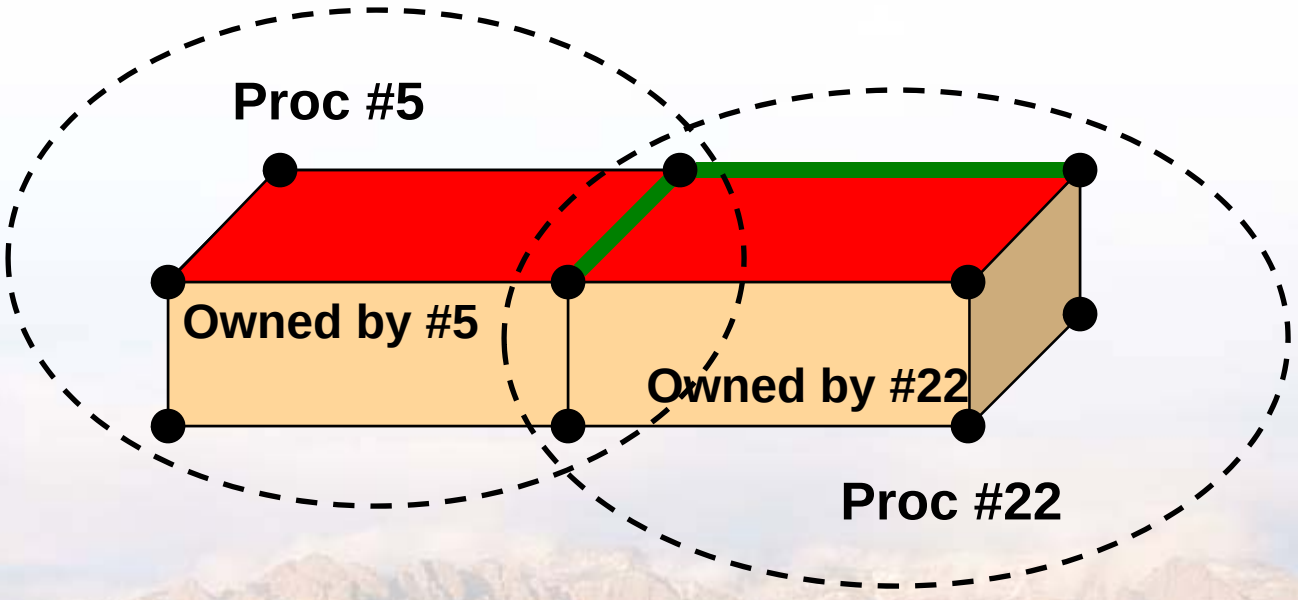
Mesh Bulk-Data: *Closure*

- **Directed Acyclic Graph (DAG)**
  - Entities are nodes of the graph
  - Relations are directed edges of the graph
  - Acyclic: directed relations higher→lower rank
  - Concept of “Closure”
- **Closure of a Mesh Entity:** $\overline{entity_a^J}$
  - Collection of entities and relations
    - Reachable from that entity
    - Following directed relations
  - E.g., an element, its nodes, and element→node
  - Assumption: The totality of computations performed on a given mesh entity require access to the closure of that mesh entity



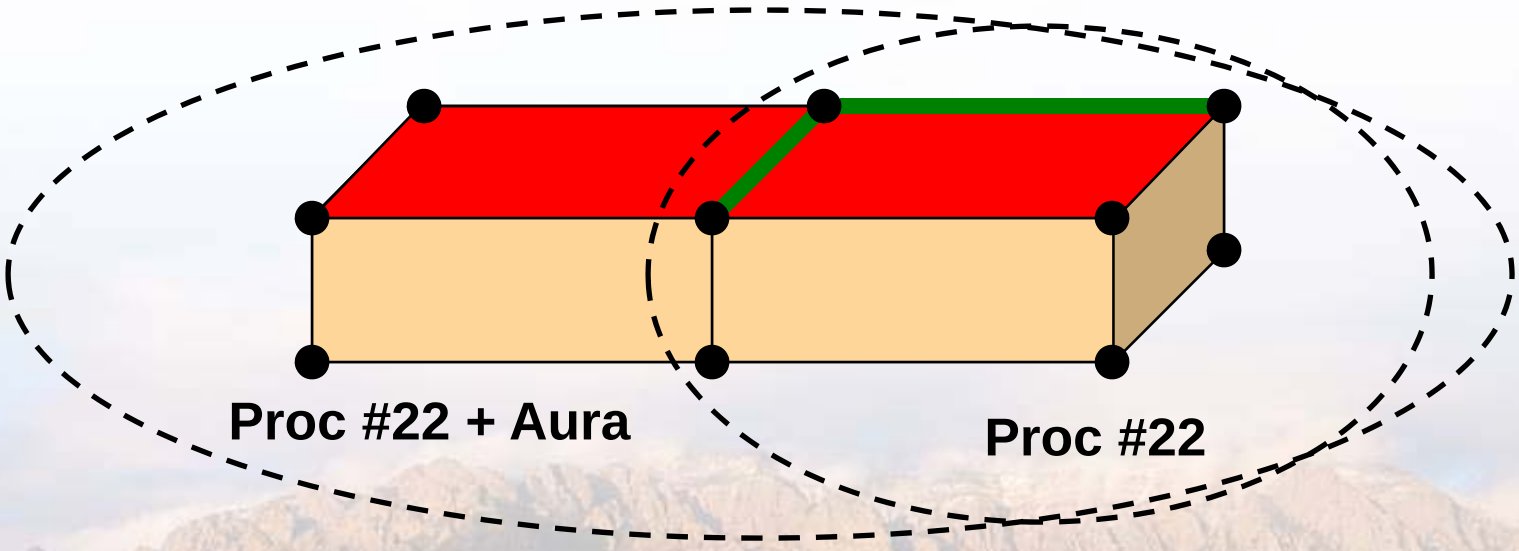
Mesh Bulk-Data: Parallel Distribution

- **Parallel distribution of mesh entities and relations**
  - Every mesh entity is uniquely owned by a parallel process
  - *Closure* of an owned mesh entity must reside on that process
  - This requires duplication (sharing) of mesh entities
  - $\overline{entity_a^J} \bigcap \overline{entity_b^J}$ is shared between the owning processes



Mesh Bulk-Data: Parallel Distribution

- **One Layer Ghosting (a.k.a., Ghosting Aura)**
  - If a mesh entity is shared by a process
  - Is in the closure of another mesh entity: $entity_b^K \in \overline{entity_a^J}$
  - Then that closure is also duplicated on the process
  - E.g., the closure of the elements connected to a shared node are ghosted on all processes on which the node is shared
  - Invaluable for neighborhood / patch accessing algorithms



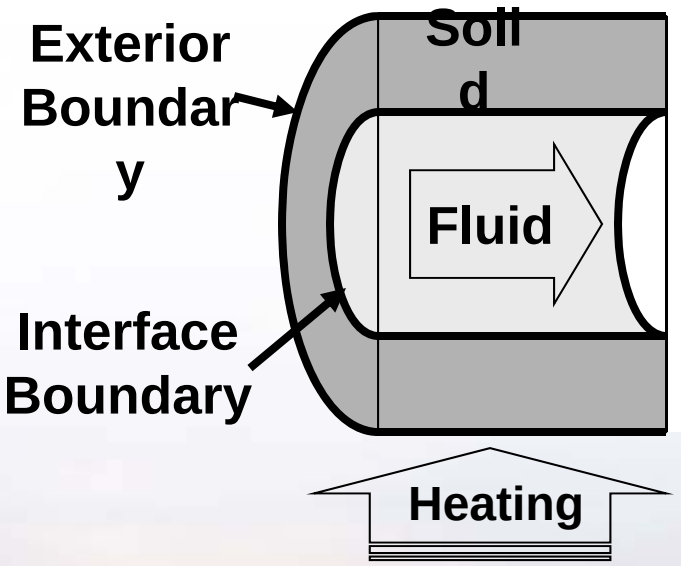
Mesh Meta Data: Defining Subsets

- **Multiphysics and Heterogeneity**
  - Different models and computations in different subdomains
  - Heterogeneous discretizations: shapes, polynomial degree, …

- **Mesh Part:** $Part_A$
  - **Define subdomains for different computations**
  - **Define collections of mesh entities with the same discretization**
  - **Define supersets of parts**

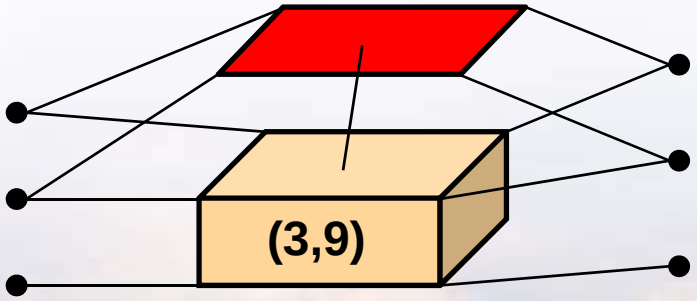
$$Part_X = Part_A \bigcup Part_B \bigcup Part_C$$

  - **# Mesh Parts depends upon heterogeneity (complexity) of the domain and computations**



Mesh Bulk-Data: Part Membership

- **A Mesh Entity is a Member of one or more Mesh Parts**
  - $entity_a^J \in Part_A \bigcap Part_C \bigcap Part_F$
  - Always a member of the universal part: $Part_\Omega$
- **Mesh Part Membership may be Induced**
  - If $entity_a^J \in Part_A$ and $entity_a^J \rightarrow entity_b^K$
  - Then $entity_b^K \in Part_A$ may be induced
  - Decision to induce membership is an attribute of the mesh part
- **Example:**
  - Entity (3,9) $\in$ Part "Block-2"
  - Nodes and face are *induced* members of Part "Block-2"

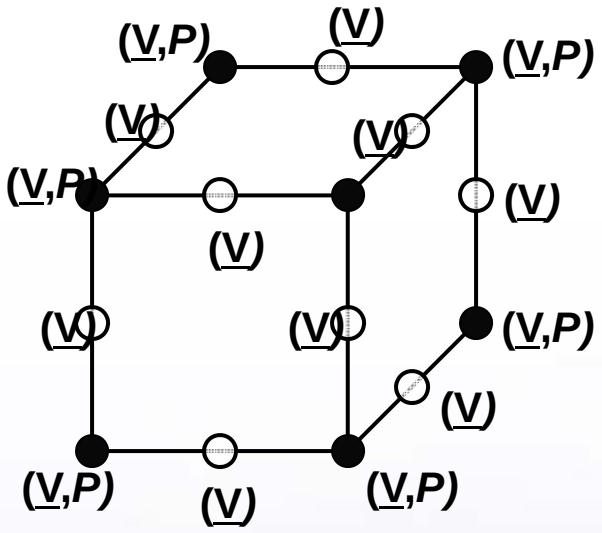


Field Data $\subset$ Computation Data

- **A *Computational* Mesh Supports Computations**
  - Computations require field data:
  - Data associated with mesh entities: $entity_a^J \rightarrow \{FieldData_X\}$
  - Heterogeneity: existence of field data varies with mesh entity
- **Field Declaration** $Field_X$
  - The *type* of the field; analogous to a 'C' or 'C++' *typedef*
  - A multidimensional array of a simple mathematical type
- **Field Restriction** $(Field_X, J, Part_A) \rightarrow [n_0, n_1, \cdots]$
  - A field exists for an $entity_a^J$
    - Which is of a specified rank J and
    - A member of a specified mesh part
  - Defines the dimensions of the multidimensional array value $entity_a^J \in Part_A$

Field Data Heterogeneity Example

- **Velocity field data on all nodes**
  - Velocity field declaration
  - Restriction $(velocity, 0, Part_{\Omega}) \rightarrow [3]$
- **Pressure field data only on vertices**
  - Pressure field declaration
  - Vertex node mesh part $Part_{VTX}$
  - Restriction $(pressure, 0, Part_{VTX}) \rightarrow 1$
  - Vertex nodes declared to be members of vertex node mesh part

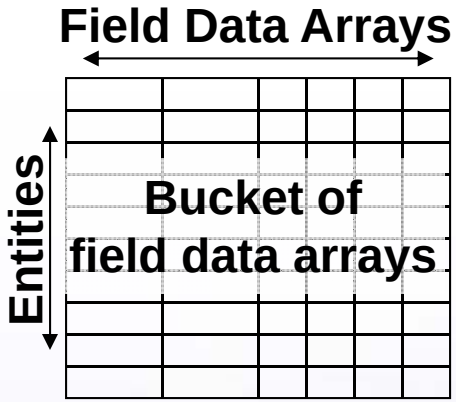


Heterogeneity Impacts Performance

- **Heterogeneous computations, heterogeneous field data**
  - Computations operate on subsets of Ω
    - Problem defined: e.g., fluid region, solid region, boundary
    - Discretization defined: e.g., hex, tet, linear, quadratic, shell
    - Parallel defined: e.g., owned by local process
  - Field data existence and dimensions can vary across Ω
- **Impact on Performance**
  - Want: Computations on nice contiguous arrays of field data
  - Have: Irregular computations and irregular field data
    - Selection logic in inner loops hurts performance, esp. GPGPU
    - Dense arrays with “ignore this entry” flags wastes memory
  - Solution: …

Field Data Arrays in *Buckets*

- **We have homogeneous subsets of mesh entities**
  - Same mesh entity rank and members of same mesh parts
  - ⇒ Have same field data of the same array dimensions
- **Buckets of homogeneous field data**
  - Contiguous arrays of field data
  - Bundled into a block of memory



- **Computations on buckets**
  - Outer loop to select buckets
  - Inner loop to computes on arrays in the selected bucket
  - Active R&D for portable thread-parallelism
    - Including GPGPU – buckets residing in device memory

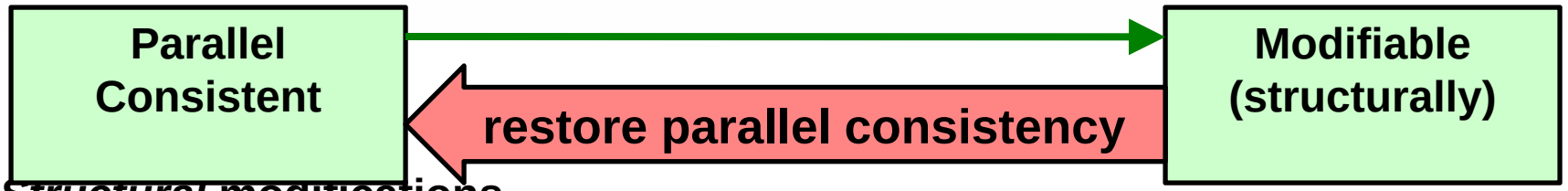
Constructing Mesh Meta-Data
(mesh database schema)

- **Declare Mesh Parts and Mesh Fields**
  - And mesh part superset-subset relationships $Part_A \subseteq Part_B$
  - And mesh field restrictions $(Field_X, J, Part_A) \rightarrow [n_0, n_1, \cdots]$
  - Detect and prohibit inconsistencies
    - Cyclic superset-subset relationships
    - Conflicting field array dimensions

- **Complete (Finalize) Mesh Meta-Data Construction**
  - Analogous to a database schema for the mesh bulk-data
  - Prohibit changes after mesh bulk-data is created …
  - Because it can be expensive and complex to edit a populated database's schema

Modifying Mesh Bulk-Data
(structural changes)

- **Two Mesh Bulk-Data Modification *States***



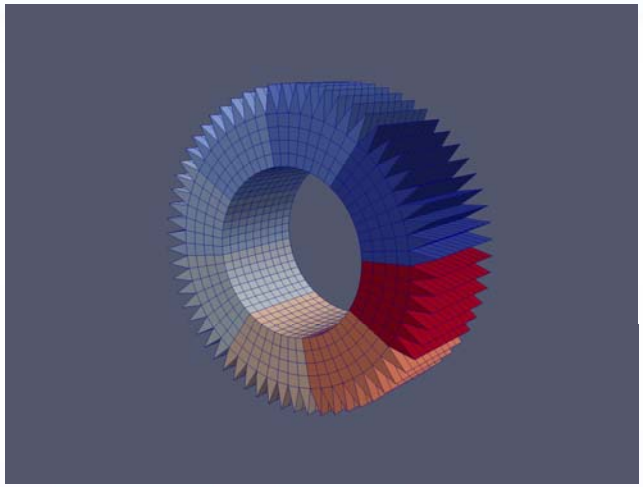
- ***Structural* modifications**
  - Declare and destroy mesh entities and relations
  - Change mesh entities' mesh part memberships
  - Local and parallel-inconsistent for shared or ghosted entities
- **Restore parallel consistency**
  - A single parallel collective operation
  - Very complex; good performance is especially hard
  - Incremental – only resolve what has been modified

STK-Mesh Finite Element Layer

- **Layer element concept onto "generic" mesh**
  - Element shapes and node ordering (a.k.a., cell topology)
    - Including element-sides and element-edges
  - Trilinos / Shards API and library of standard cell-topologies
- **Optionally Associate a Cell Topology with a Mesh Part**
  - $Part_{Tet\langle 4\rangle} \rightarrow Tetrahedron\langle 4\rangle$
  - All elements that are members of this part are tetrahedrons
- **Includes concept of element boundaries**
  - Sides & side neighbors, edges & edge neighbors
- **Foundation for finite element computations**
  - Basis functions, numerical integration, …

Conclusion

- **STK-Mesh is an active R&D effort**
  - Within the DOE ASC SIERRA Toolkit project at Sandia
  - Related R&D for field data arrays on multicore and GPGPU
  - Open source though Trilinos: http://trilinos.sandia.gov
- **Very complex domain and needs**
  - Parallel, heterogeneous, dynamically modifiable unstructured mesh
  - Computational performance requirements: field data buckets
- **Domain Modeling is Key to Managing Complexity**
  - Modular and layered architecture
  - “Lean and clean” modules, dependencies, and APIs
  - Minimize coupling between modules



STK-Mesh Example Computation Setup and Parallel Execution

Carter Edwards, **Todd Coffey**,
Dan Sunderland, Alan Williams

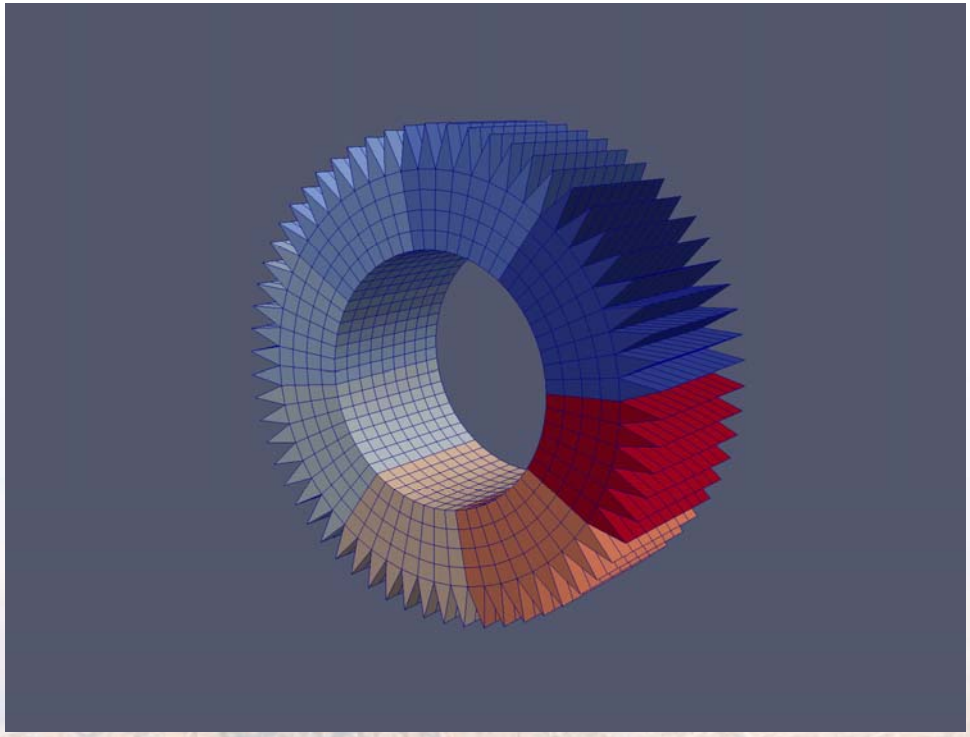
Sandia National Laboratories

Outline

- Overview of Gear Fixture & features
- Setup of Meta Data
- Setup of Bulk Data
- Computational Kernel
- Movie

Description of Gears Demo

- Basis for movie at Super-Computing 2010
- Example is in STK sources and is used for testing



Overview of Gear Fixture

- **Located in source:**
  - http://trilinos.sandia.gov/
    - packages/stk/stk_mesh/stk_mesh/fixtures/GearsFixture
    - packages/stk/stk_mesh/stk_mesh/fixtures/Gear
    - packages/stk/stk_performance_tests/stk_mesh/GearsSkinning
- **Cylindrical gear body made up of Hexahedron<8> elements**
- **Gear Teeth made up of Wedge<6> elements**
- **Parallel distribution of elements and nodes**
- **Rotation of gear**

Gear Meta Data

- **Declare the following parts of the mesh**
  - Cylindrical coordinates part
  - Hex part
  - Wedge part
- **Declare the following fields:**
  - Cartesian coordinates field (x,y,z) on nodes
  - Displacement field (Δx,Δy,Δz) on nodes
  - Translation field on nodes
  - Cylindrical coordinates field (r,θ,z) on nodes
- **Restrict the fields to parts**

Fixture Declaration of Meta Data

```
typedef Field< double , Cylindrical>  CylindricalField;
typedef Field< double , Cartesian>    CartesianField;

class GearsFixture {
public:
  MetaData   meta_data;
  BulkData   bulk_data;
…
  Part & cylindrical_coord_part;
  Part & hex_part;
  Part & wedge_part;
  CartesianField    & cartesian_coord_field;
  CartesianField    & displacement_field;
  CartesianField    & translation_field;
  CylindricalField  & cylindrical_coord_field;
…
};
```

Fixture Construction: [MetaData]

- **Inside Gear Fixture constructor:**

```
const unsigned TWO_STATE = 2;
GearsFixture::GearsFixture( … ) :
  meta_data( fem::entity_rank_names(SpatialDimension) ),
  bulk_data( meta_data , parallel_machine ),
  …,
  cylindrical_coord_part(
     meta_data.declare_part("cylindrical_coord_part”,element_rank)
     ),
  displacement_field(
     meta_data.declare_field<CartesianField>("displacement", TWO_STATE)
     ), …
{
…
}
```

Restrict Fields to Parts: [MetaData]

- Field Restrictions: Entities in the part will have the field

- Example for cylindrical coordinate field and part:

```
GearsFixture::GearsFixture(…): [prev slide] {
  put_field(
        cylindrical_coord_field,
        node_rank,
        cylindrical_coord_part,
        SpatialDimension
        );
  …
}
```

Induced Part Membership

Simplified Mesh Maintenance

- **How to apply a field to nodes that are connected to elements of a particular type?**

- **Option A: Manually**
- **Option B: Automagically**
  1. Put elements in a Part of that type
  2. Restrict the nodal field to that part
  3. All nodes in closure of elements in this part will get this field

- **Cylindrical Part limited to Elements**
- **Cylindrical coordinates Field limited to Nodes**
- **Cylindrical coordinates field restricted to cylindrical part**

Gear Bulk Data Operations

- **Create Hex entities for gear body**
  - Place Hex entities in Hex part and Cylindrical part
  - Declare relations to nodes
- **Create Wedge entities for gear teeth**
  - Place Wedge entities in Wedge part and Cylindrical part
  - Declare relations to nodes
- **Initialize field data**
- **Skin mesh with faces**
  - Skinning function in fem

Initialize Field Data (entity-wise)

- Loop over nodes
- Assign Cartesian coordinates from cylindrical coordinates

```
loop over nodes {
    const double rad = … ;
    const double angle = … ;
    const double height = … ;

    double * const cartesian_data = field_data( cartesian_coord_field , node );

    cartesian_data[0] = rad * std::cos(angle);
    cartesian_data[1] = rad * std::sin(angle);
    cartesian_data[2] = height;
}
```

Gear Rotation Computation (bucket-wise)

- **Recall: Select buckets, iterate over buckets, compute**

```
Selector select = cylindrical_coord_part &
                  ( locally_owned_part | globally_shared_part );

BucketVector selected_node_buckets;

get_buckets(
    select,
    bulk_data.buckets(Node),
    selected_node_buckets
    );
```

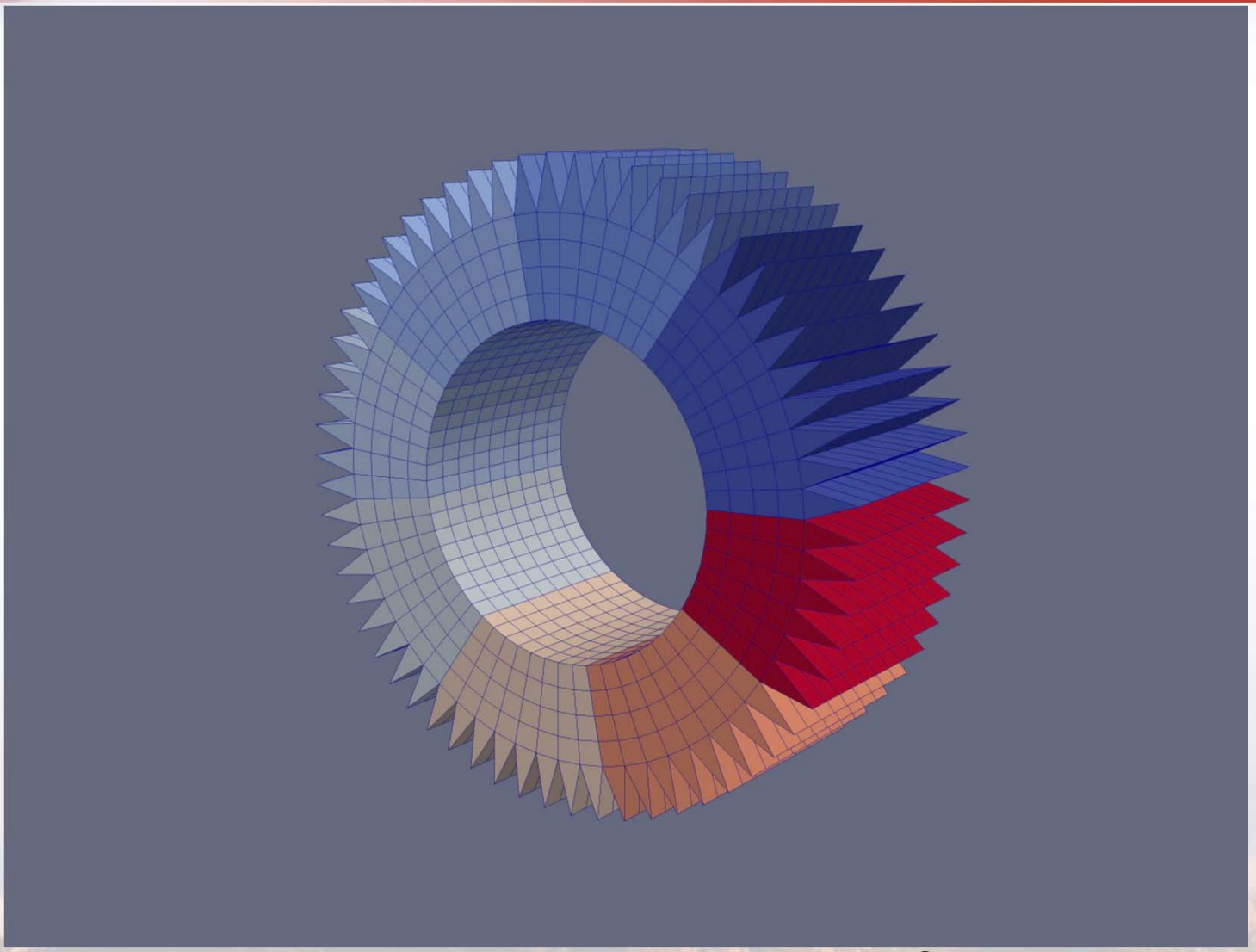
Gear Rotation Computation (bucket-wise)

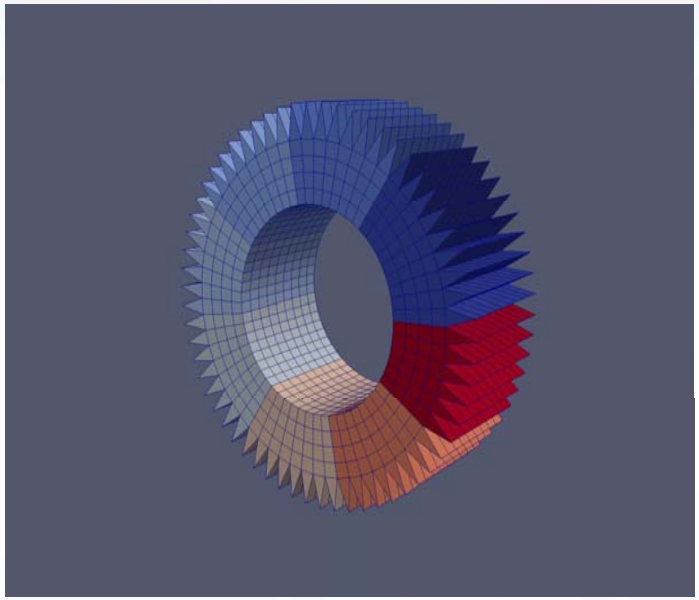
```
for (BucketVector::iterator b_itr = selected_node_buckets.begin();
      b_itr != selected_node_buckets.end();
      ++b_itr) {
  Bucket & b = **b_itr;

  const BucketArray<CartesianField>
      old_coordinate_data( cartesian_coord_field, b);
  BucketArray<CartesianField>
      displacement_data( displacement_field.field_of_state(StateNew), b);
  …

  for (size_t node_index = 0; node_index < b.size(); ++index) {
    // Compute new and old coordinate data and assign to displacements:
   displacement_data(0,index) = new_coordinate_data[0] – old_coordinate_data(0,index);
   displacement_data(1,index) = new_coordinate_data[1] – old_coordinate_data(1,index);
   displacement_data(2,index) = new_coordinate_data[2] – old_coordinate_data(2,index);
   …
  }
}
```

Hybrid Parallelism

- Alan's talk will cover hybrid parallelization of this example
- Basic Idea: Parallel For applied to bucket loops
- Also see the following example:
  stk_usecases/app/UseCase_blas_algs.hpp
  stk_usecases/app/UseCase_blas.cpp

Gears Demo (stk_mesh, epu, conjoin, paraview)





STK-Mesh Example Dynamic Mesh Modification

Carter Edwards, Todd Coffey,
Dan Sunderland, Alan Williams

Sandia National Laboratories

Outline

- **The Meta-Data**
- **Modifying the Bulk-Data**
  - Maintaining Consistency
    - Bulk-Data State
    - Parallel Consistency
    - Local Consistency
    - Field Data
    - Atomic Modifications
- **Gear Demo**
  - Overview
  - Code snippets necessary to strip teeth off of the gear

Mesh Modification

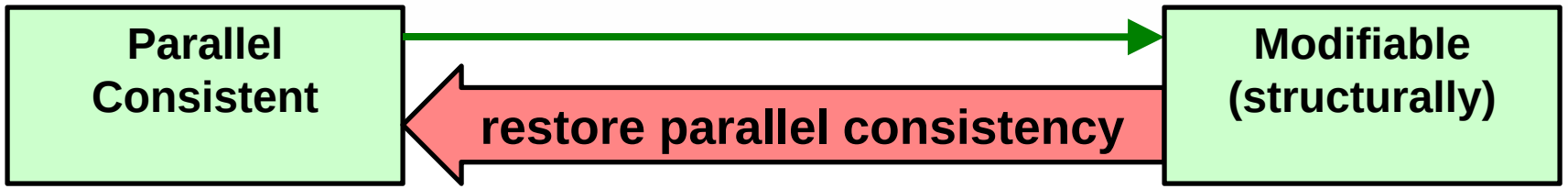
- **Meta-Data**
  - Equivalent to the schema of a database, the meta-data describes the problem domain
  - Freely modifiable before being committed
  - ***No*** modifications allowed after commit

Mesh Modification

Bulk-Data

- Contains the discretization of the problem
- Its meta-data ***must be*** committed before any function beside the bulk-data constructor can be called
  - *Meta-data check for global consistency will throw if the meta-data is not globally consistent*
- Bulk-data modifications are only allowed when the bulk-data is in a modifiable state.

Modification Cycle



- **Parallel Consistent**
  - **NO** Mesh modification allowed
- **Modifiable**
  - Guaranteed to be locally consistent
  - Atomic mesh modifications are allowed

*Parallel consistency is enforced when switching from **Modifiable** to **Parallel Consistent***

Maintaining Consistency

- The bulk-data state is transitioned by calling

```
bulk_data.modification_begin()
bulk_data.modification_end()
```

  - A new modification cycle begins whenever modification begin is called
  - Atomic mesh modifications mark affected entities and all their upward relations as *modified*
  - Parallel consistency is enforced at modification end by
    - Deleting all ghosts of modified entities
    - Resolving parallel ownership and sharing of created and destroyed entities
    - Resolving shared entities mesh part membership, entity relations, field data memory allocation, and bucket membership
    - Updating the one layer ghosted aura

Maintaining Consistency

- **The following operations are available through Atomic Mesh Modifications**
  - Creating and/or deleting entities
  - Changing entities' mesh part memberships
  - Changing entities' relations
  - Moving entities' ownership to another process

- **Atomic modifications are guaranteed to be *locally consistent***
  - Induced mesh part membership will change as necessary
  - Memory for field data will be created, resized, or deleted
  - Existing field data will move to the correct bucket

Maintaining Consistency

Field Data

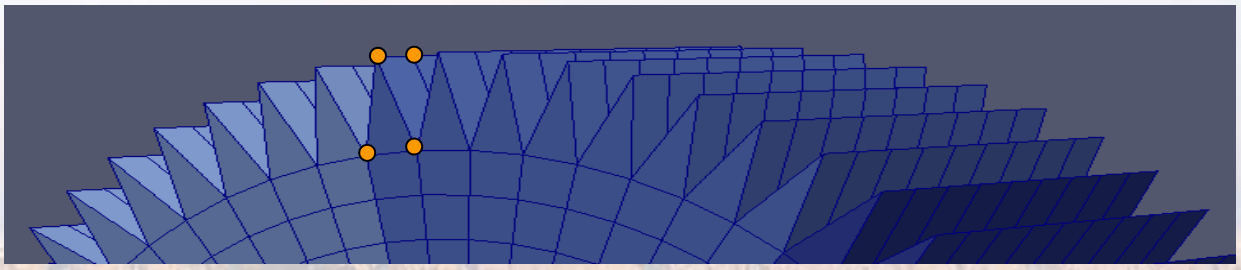
- Communicating field data **does not** modify the topology of the mesh and can happen at any time (i.e. not restricted to a modifiable bulk-data**)**
- Atomic mesh modifications will move existing field data to the correct bucket on the *local* process

  *When **changing entity owner** the field data is moved to the correct bucket on the remote process*
- After modification end space for field data of ghosted and shared entities has be allocated but the data *has not been copied* from the owner
- Field data values can be copied from owned to shared/ghosted at any time by calling

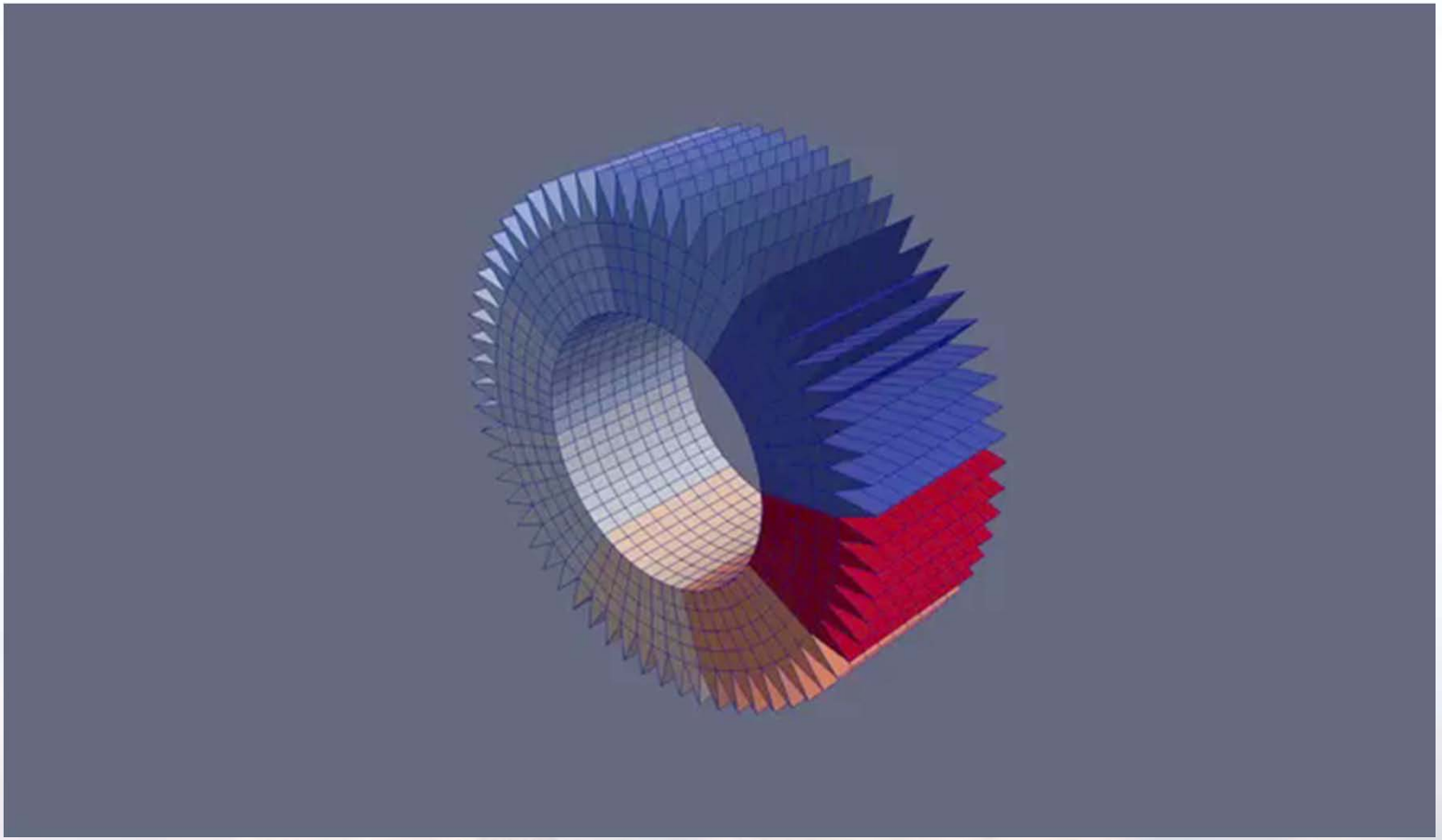
```
bulk_data.communicate_field_data(…);
```

Simple Mesh Modification Example

- **Breaking teeth off of the gear**
  - Distribute mesh over available processes
  - Create 6 new nodes to attach to the wedge
  - Destroy the relationship between the current nodes and the wedge
  - Attach the new nodes to the wedge
  - Copy the field data from the current nodes to the new nodes
    - The current nodes stay with the body gear



Putting it all together



Distribute mesh across available processes

- **Change entity owner (moving the entity to another process)**
  - The bulk data function change_entity_owner is a parallel collective call that ***gives away*** ownership of entities to other processes

```
//EntityProc is the pair
//(Entity *owned_entity, unsigned to_proc_id)
std::vector< stk::mesh::EntityProc> > move_entities;

// move the wedge to proc 2
if ( bulk_data.parallel_rank() ==  0) {
  move_entites.push_back( std::pair(&wedge,2));
}

// Parallel collective call
bulk_data.change_entity_owner( move_entities );

// The field data will also be moved with the wedge
```

Creating a new wedge

- **Creating new entities**
  - When new entities are created the creating process is the *owner* of the entity
  - If two or more processes create an entity with the same rank and identifier, the entity will be ***shared***

```
//Declare a new wedge element
stk::mesh::PartVector add_parts;
add_parts.push_back( wedge_part );
stk::mesh::Entity & wedge = bulk_data.declare_entity(
                                                      element_rank,
                                                      element_id,
                                                      add_parts
                                                    );
```

Creating new nodes

- **Generating new entities**
  - The bulk data function generate_new_entities is a parallel collective call that will create new entities of the requested ranks with *globally unique ids*

```
//create 6 nodes on process 0
std::vector<size_t> num_requested_entities(num_entities_rank,0);

if ( bulk_data.parallel_rank() == 0) {
        num_requested_entities[node_rank]=6;
}

stk::mesh::EntityVector requested_nodes;

//parallel collective call
bulk_data.generate_new_entities(
                                num_requested_entities,
                                request_nodes
                               );
```

Attaching the nodes to the wedge

- **Creating/Destroying entity relations**
  - Relations are directed from the higher ranking entity to the lower ranking entity, the converse relation is automatically inserted/deleted
  - Processes can only create/destroy relations when they own or share the higher ranking entity
  - Creating or deleting relations may change entity's induced mesh part membership

```
// declare relations from the wedge to the nodes
for ( size_t i=0; i<6; ++i) {
  bulk_data.declare_relation (
                        wedge, // from entity
         requested_nodes[i], // to entity
                            i  // relation identifer
                               );
}
```

Destroying nodes

- **Destroying entities**
  - A process may destroy its reference to an entity if the entity does not have a relation to a higher ranking entity in its owned closure.
  - If the owning process destroys an entity and another process *shares* the entity, ownership automatically transfers to a *sharing* process

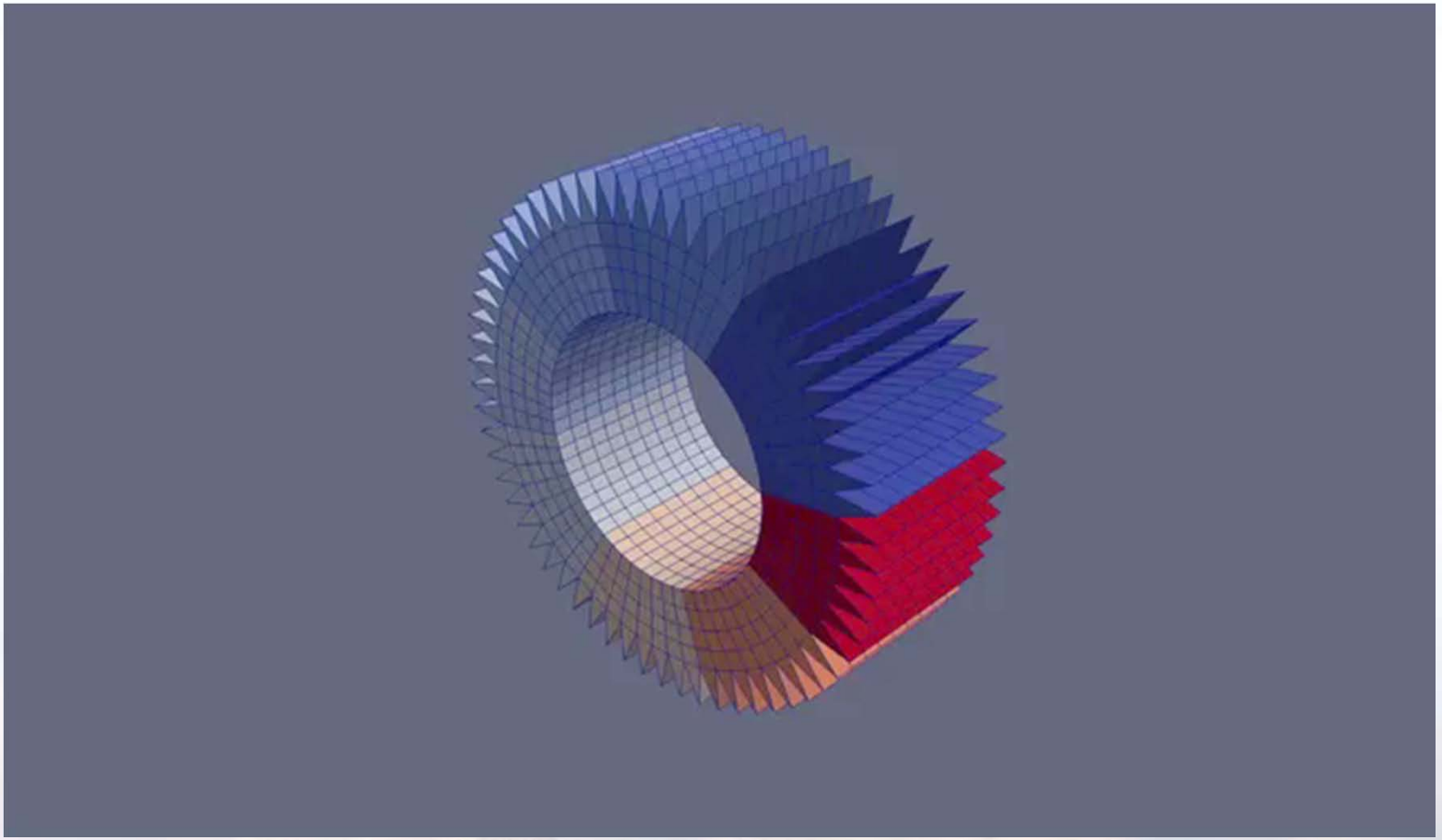
```
// destroy the old nodes attached to the wedge
stk::mesh::PairIterRelations
    node_relations = wedge.relations(node_rank);

for ( size_t i=0; i<node_relations.size(); ++i) {
  stk::mesh::Entity * node = node_relations[i].entity();
  //destroy relation to wedge
  bulk_data.destroy_relation( wedge, *node );
  //destroy the entity
  bulk_data.destroy_entity( node );
}
```

Changing mesh part membership

- Processes can only change part membership when they **own** or **share** the entity
- The entity will be moved to a different bucket
- The fields available to the entity will change to match the fields restrictions

```
// add the wedge to the cylindrical coordinate part
stk::mesh::PartVector add_parts, remove_parts;
add_parts.push_back( & cylindrical_coord_part );

bulk_data.change_entity_parts(
                                wedge,
                                add_parts,
                                remove_parts
                               );
// If cylindrical_coord_part was declare to be of rank element,
then the nodes will be induced into this part and the
cylindrical coordinate field will be available to the nodes
```

Gear Demo: Putting it all together



STK: Beyond stk-mesh
(what else is in STK?)

STK Tutorial

Carter Edwards, Todd Coffey,
Dan Sunderland, Alan Williams

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND2009-2803P

Sandia National Laboratories

STK modules overview

Algorithm-Support (AlgSup)

- Multi-threaded execution of bucket-loop algorithms

Search

- Proximity, mesh independent

Linsys, IO, Rebalance

- Bridges from mesh-data to external capabilities
- Built optionally

Util

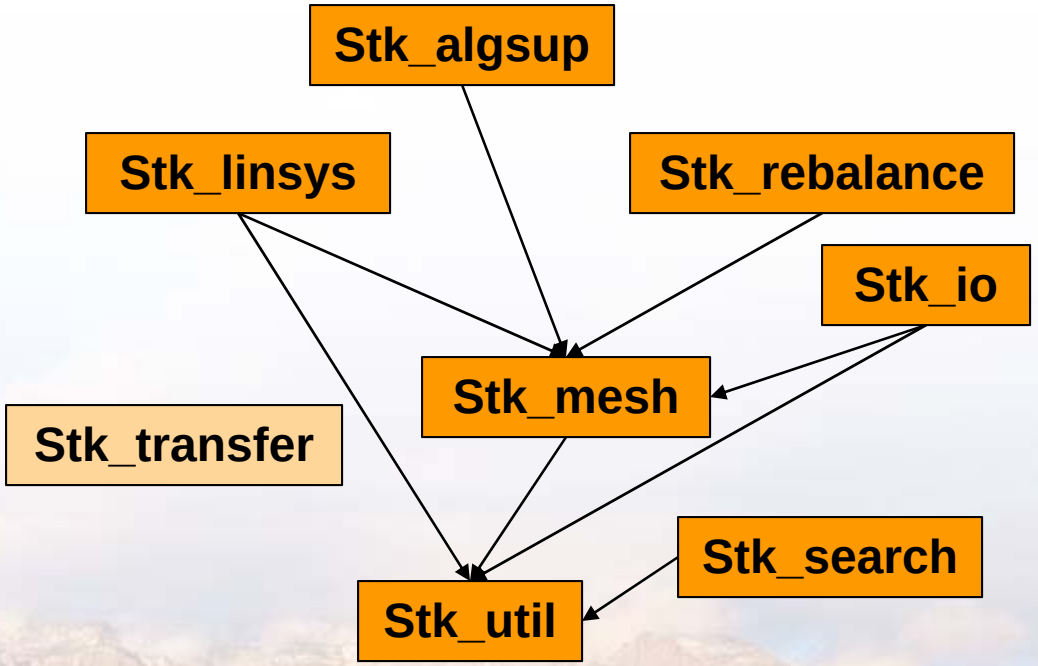
- Everything depends on util directly or indirectly

Transfer

- Not yet implemented

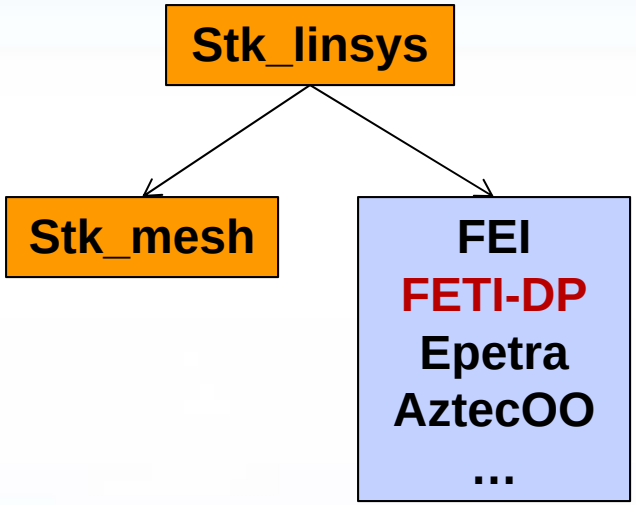
Dependency diagram:

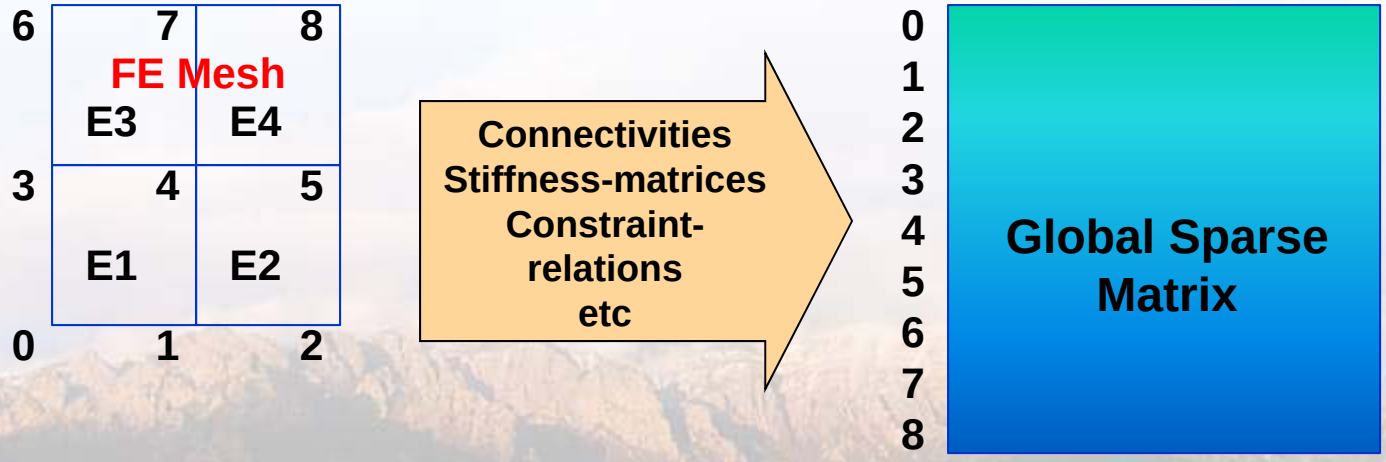
- Arrows point towards a module that is used (depended on) by another module.



STK linsys – bridge between mesh-data and linear-systems

Stk_linsys provides helpers for assembling linear-systems





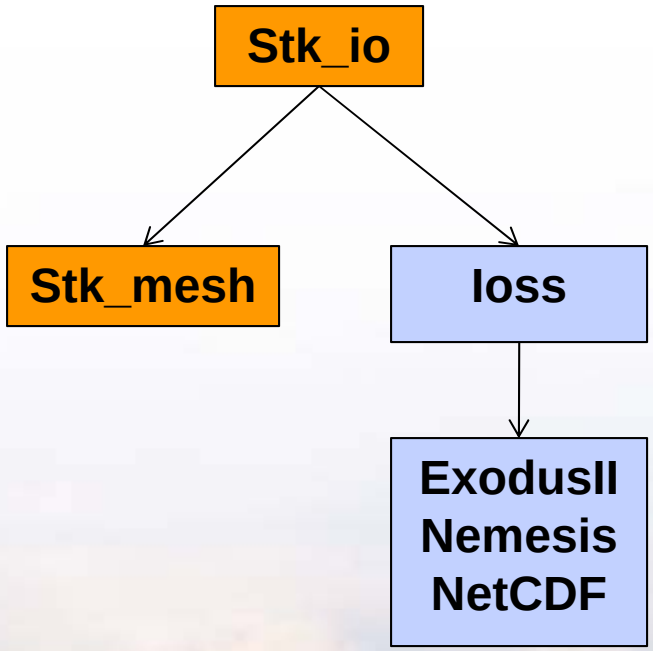
STK IO – Bridge between mesh-data and disk IO

IOSS: Abstract Finite Element mesh interface

- Object-Oriented
- Database-Independent
- ExodusII is primary output format
- Also supports XDMF, heartbeat, history, and pamgen

Stk_io is a Bridge layer that understands the interfaces of stk_mesh and Ioss.

- Provides functions for moving data to/from stk_io and Ioss.
- Provides high-level functions for creating a mesh input and/or results outputwith minimal code.
- Lower-level functions also provided for more control of the data movement.



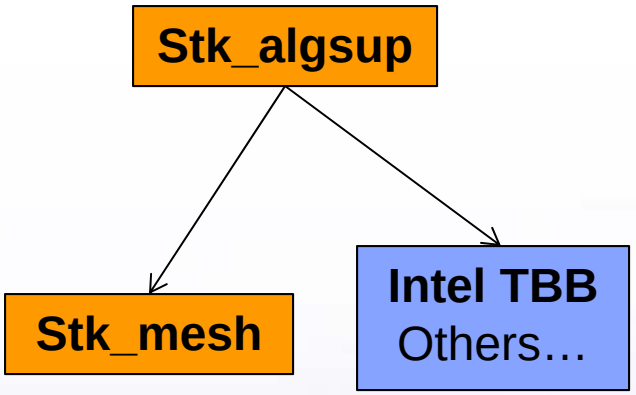
STK_AlgSup – Algorithm Support

Infrastructure for executing bucket-loop algorithms, including multi-threaded.

Recall the gears demo had loops for updating coordinates, etc.

In general, many algorithms on mesh data can be described as:

- create selector for desired parts
- get buckets
- for each bucket:
  - get field-data
  - perform computations



The bucket-loop is often compute-intensive and can benefit from thread-parallel (or GPU) execution.

Gear Rotation Computation (bucket-wise)

```
for (BucketVector::iterator b_itr = selected_node_buckets.begin();
      b_itr != selected_node_buckets.end(); ++b_itr)  {
    Bucket & b = **b_itr;

    const BucketArray<CartesianField> old_coord_data( cartesian_coord_field, b);
    BucketArray<CartesianField> displ_data(displacement_field.field_of_state(StateNew), b);
    …

    for (size_t node_index = 0; node_index < b.size(); ++index) {

      // Compute new and old coordinate data and assign to displacements:

      displ_data(0,index) = new_coord_data[0] – old_coord_data(0,index);
      …

    }
}
```

Sandia National Laboratories

STK_AlgSup – Algorithm Support (continued)

An Algorithm can be any class that has an 'apply' method…
(and the apply method contains the body of your bucket-loop**)**

```
struct My_Algorithm {

void apply(stk::mesh::Bucket::iterator begin,
           stk::mesh::Bucket::iterator end) const
{
  size_t num_nodes = std::distance(end – begin);

  const BucketArray<CartesianField> old_coord_data( cartesian_coord_field, begin, end
  BucketArray<CartesianField>
       displ_data(displacement_field.field_of_state(StateNew), begin, end);

  for (size_t node_index = 0; node_index < num_nodes; ++node_index) {
    // Compute new and old coordinate data and assign to displacements:

    displ_data(0,node_index) = new_coord_data[0] – old_coord_data(0,node_index);
    …
  }
}
 …
};
```

Sandia National Laboratories

STK_AlgSup – Algorithm Support

Example using Intel Thread Building Blocks (TBB)

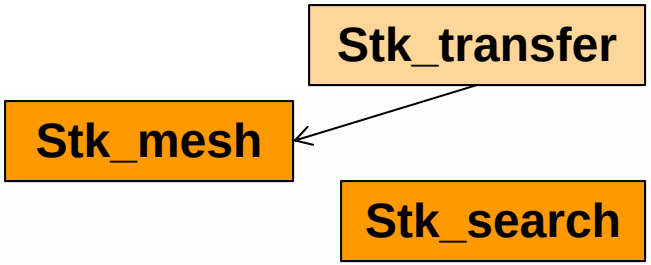
```
stk::AlgorithmRunnerInterface*
        alg_runner = stk::algorithm_runner_tbb(num_threads);

stk::mesh::Part& gear_part = …
stk::mesh::Selector select_nodes = gear_part & meta_data.locally_owned_part();

My_Algorithm alg;

alg_runner->run(select_nodes, …, bulk_data.buckets(NODE_RANK), alg);
```

The algorithm-runner implementation can launch buckets on different threads.

The goal is for application code (My_Algorithm) to be unaware of threading, as much as possible.
(Of course thread-safety issues must still be kept in mind…)

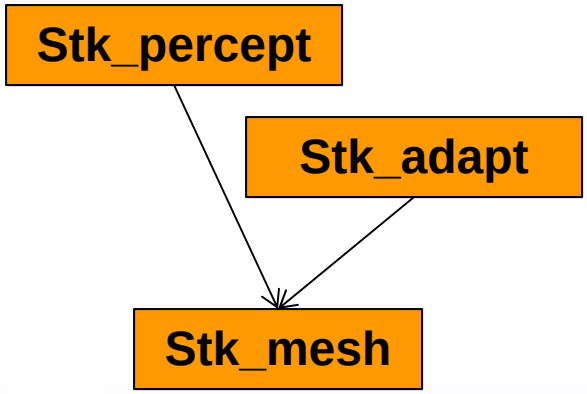
STK_Search, STK_Transfer

- Geometric proximity searches including: OctTree, BIH
- Proximity search doesn't depend on stk_mesh
  - Users provide lists of bounding boxes or bounding spheres
- Stk_transfer will use stk_search, and will depend on stk_mesh
- …but there will also be stk_mesh – to – non-stk_mesh transfers



STK_adapt, STK_percept

- Stk_percept
  - Verification tools, including:
    - Postprocessing
    - Error metrics (e.g. norms)
    - Manufactured solutions
- Stk_adapt
  - Extend/modify the discretization
    - Refine elements
    - Increase polynomial order
    - Convert topology (e.g. hex to tet)



STK_util - Utilities

STK_util doesn't depend on any other STK module

STK_util contains several subdirectories of related tools

- **Diag**
  - Timers
  - Ostream manipulators
- **Environment**
  - Error Reporting
  - Output Logging
- **Parallel**
  - Comm utils
  - Distributed-Index
- **Util**
  - String comparisons
  - Array Ops
  - etc

Future Modules

- **Stk_transfer:**
  - Mesh to mesh transfers
  - Including stk-mesh – to – non-stk-mesh transfers
- **Stk_parser:**
  - Input deck parsing
  - Sierra SDDM
- **Stk_coupling:**
  - Code coupling layer

Goals of the Toolkit Team

- **Clarity of code**
- **Use-case driven API & implementation**
- **Capability/Flexibility/Performance**
  - Growing collection of performance tests, use-cases, …
- **Modularity**
- **Highly unit-tested code**

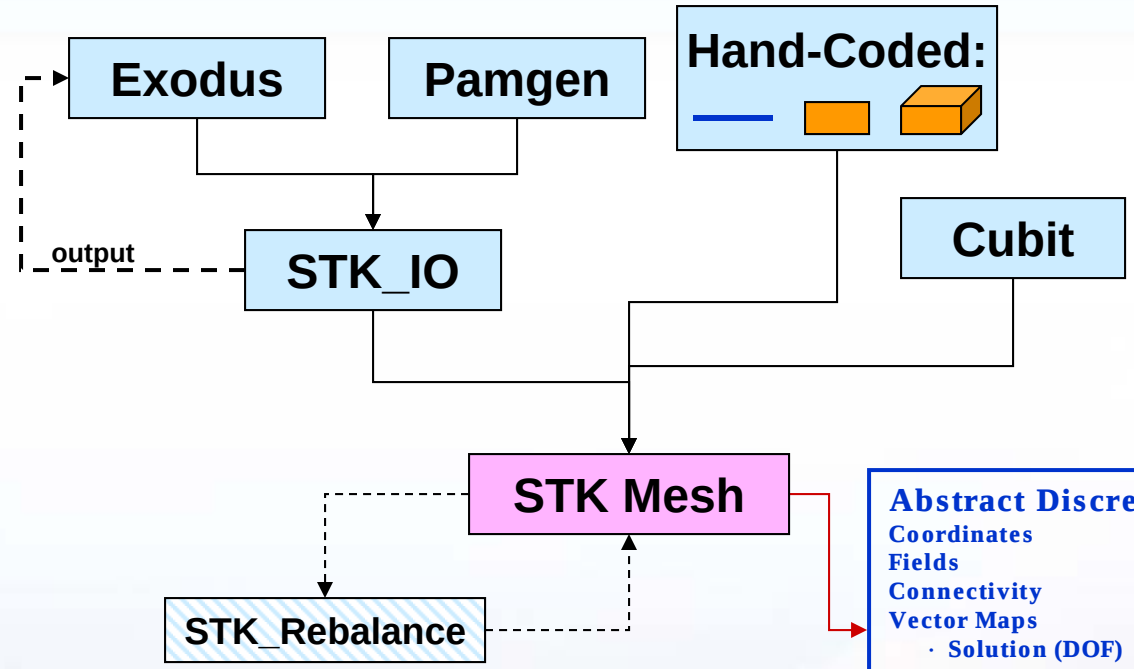
stk_mesh/stk_mesh/base		90.1 %	3272 / 3631
stk_mesh/stk_mesh/baseImpl		93.8 %	1099 / 1172

- **Threading/GPU support**
- **Facilitate external collaborations with Sierra**
- **Agile development**

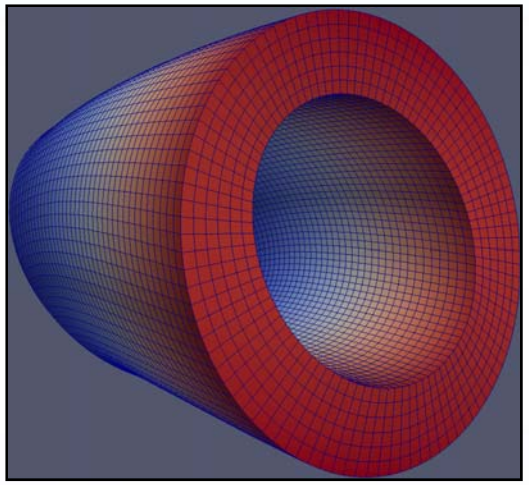
Current Users

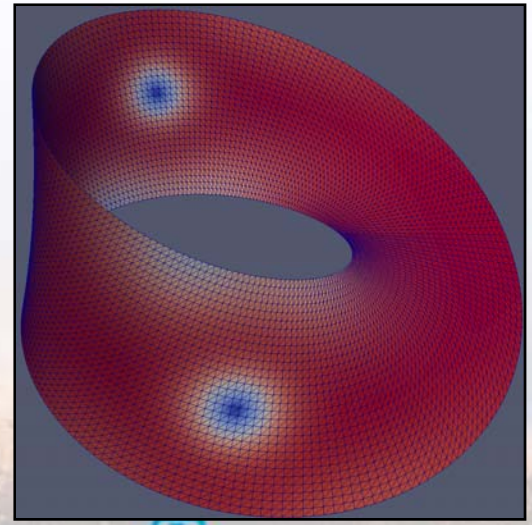
- Sierra (suite of finite-element analysis applications), SNL
- Cubit (Mesh generation)
  Steve Owen/Matt Staten, SNL
- Albany (Prototyping rapid PDE-application development)
  Andy Salinger, SNL
- Adaptive Mesh Refinement
  Ulrich Küttler, Technische Universität München
- Superconductivity simulations
  Nico Schloemer, University of Antwerp
- Fracture mechanics
  Joe Bishop, Fracture LDRD, SNL
- Charon (planned for FY11), SNL

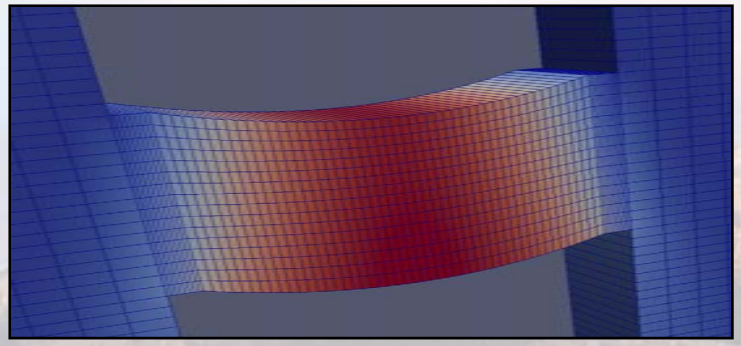
STK Capability in Albany



Abstract Discretization
- Coordinates
- Fields
- Connectivity
- Vector Maps
  - Solution (DOF)
  - Coordinate
  - Node
- Jacobian Graph
- Node Sets







Thank you!

- **For more information:**
  http://trilinos.sandia.gov/packages/stk/
  stk-users@software.sandia.gov email list

- **We want more feedback!**
  - Good clean clear APIs and code can't be achieved without user feedback!

- **Thanks for attending this tutorial!**

- **Developers:**
  Dave Baur, Todd Coffey, Carter Edwards,
  Jim Foucar, Russell Hooper, James Overfelt,
  Greg Sjaardema, Dan Sunderland, Alan Williams