Exploring Co-Design in Chapel Using LULESH

SIAM CSE13, MS79
February 26, 2013

Greg Titus
Principal Engineer
Chapel Team, Cray Inc.

Chapel

What is Chapel?

- **An emerging parallel programming language**
  - Design and development led by Cray Inc.
    - in collaboration with academia, labs, industry
  - Initiated under the DARPA HPCS program
- **Overall goal: Improve programmer productivity**
  - Improve the programmability of parallel computers
  - Match or beat the performance of current programming models
  - Support better portability than current programming models
  - Improve the robustness of parallel codes
- **A work-in-progress**
- **http://chapel.cray.com/**

Chapel's Implementation

- **Being developed as open source at SourceForge**
- **Licensed as BSD software**
- **Target Architectures:**
  - Cray architectures
  - multicore desktops and laptops
  - commodity clusters
  - systems from other vendors
  - *in-progress:* CPU+accelerator hybrids, manycore, …

Motivating Chapel Themes

1) General Parallel Programming
2) Global-View Abstractions
3) Multiresolution Design
4) Control over Locality/Affinity
5) Reduce HPC ↔ Mainstream Language Gap

1) General Parallel Programming

With a unified set of concepts...

...express any parallelism desired in a user's program

- **Styles:** data-parallel, task-parallel, concurrency, nested, …
- **Levels:** model, function, loop, statement, expression

...target all parallelism available in the hardware

- **Types:** machines, nodes, cores, instructions

1) General Parallel Programming

With a unified set of concepts...

...express any parallelism desired in a user's program

- **Styles:** data-parallel, task-parallel, concurrency, nested, …
- **Levels:** model, function, loop, statement, expression

...target all parallelism available in the hardware

- **Types:** machines, nodes, cores, instructions

Style of HW Parallelism	Programming Model	Unit of Parallelism
Inter-node	MPI	executable/process
Intra-node/multicore	OpenMP	iteration/task
GPU/accelerator	CUDA	SIMD function/task

1) General Parallel Programming

With a unified set of concepts...

...express any parallelism desired in a user's program

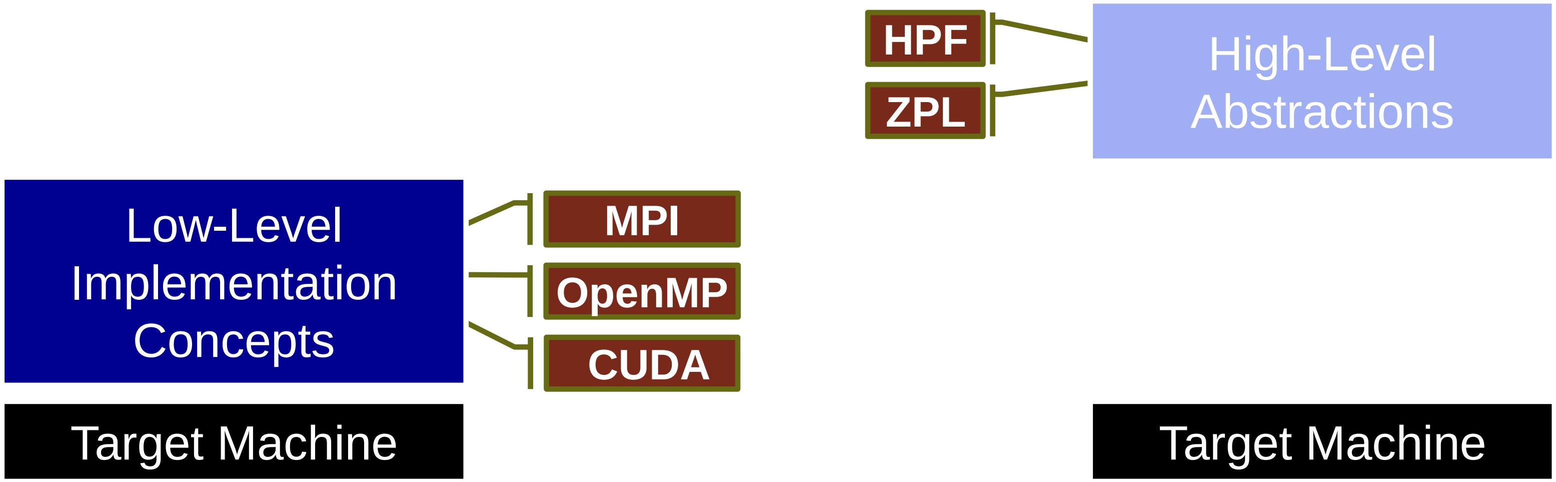
- **Styles:** data-parallel, task-parallel, concurrency, nested, …
- **Levels:** model, function, loop, statement, expression

...target all parallelism available in the hardware

- **Types:** machines, nodes, cores, instructions

Style of HW Parallelism	Programming Model	Unit of Parallelism
Inter-node	Chapel	executable/task
Intra-node/multicore	Chapel	iteration/task
GPU/accelerator	Chapel	SIMD function/task

3) Multiresolution Design: Motivation



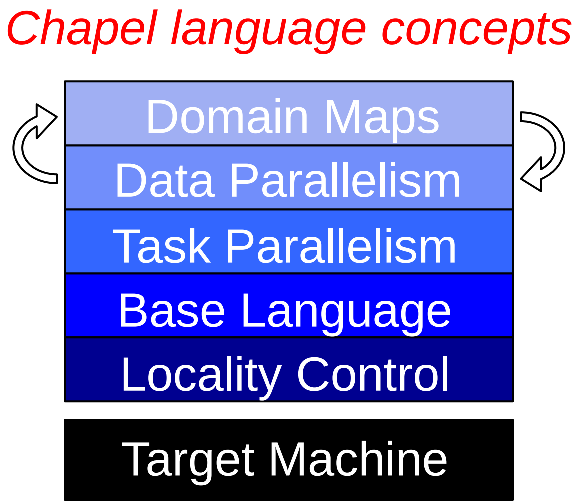
"Why is everything so tedious/difficult?"
"Why don't my programs port trivially?"

"Why don't I have more control?"

3) Multiresolution Design

Multiresolution Design: Support multiple tiers of features

- higher levels for programmability, productivity
- lower levels for greater degrees of control



- build the higher-level concepts in terms of the lower
- permit the user to intermix layers arbitrarily

LULESH (in Chapel)

What is LULESH?

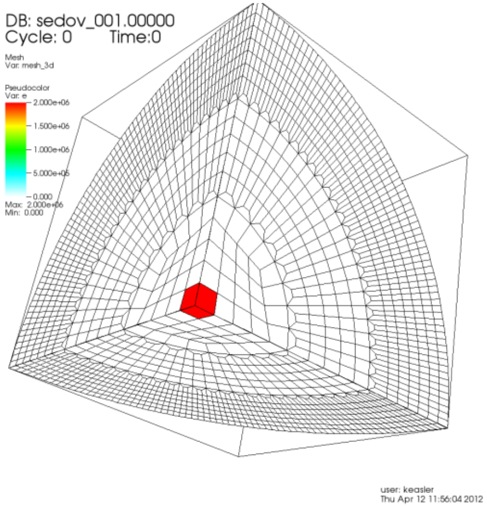
- ***Livermore Unstructured Lagrange Explicit Shock Hydrodynamics* challenge problem**
  - Developed as a *proxy application* at LLNL under DARPA UHPC
  - Includes computations and algorithms used in production codes

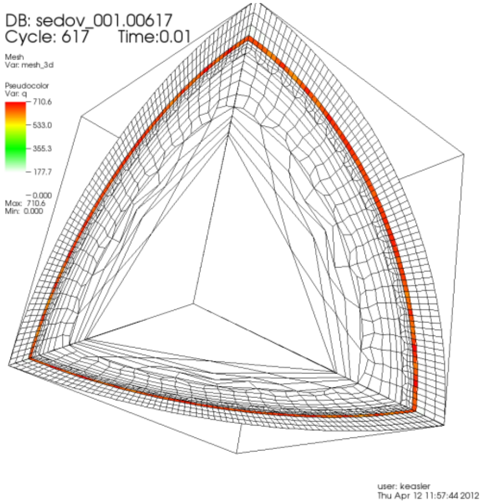
- **https://computation.llnl.gov/casc/ShockHydro/**

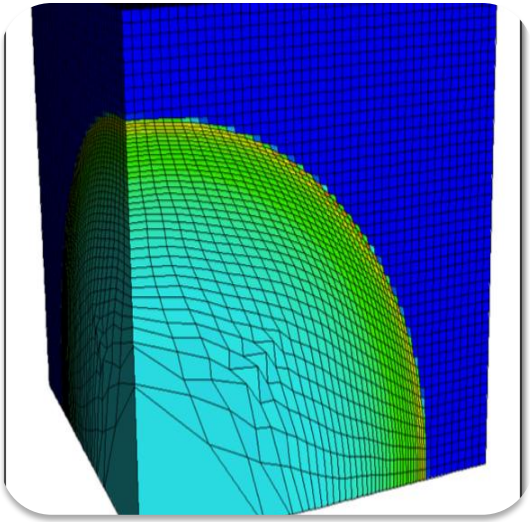
- **There are reference implementations in many languages**
  - Serial C
  - C + OMP
  - C + OMP + MPI (not publically available yet)
  - CUDA
  - Loci (logic programming)
  - A++ (C++ class library)
  - Chapel

What Does LULESH Do?

- Solve one octant of the spherical Sedov problem (blast wave) using Lagrangian hydrodynamics for a single material

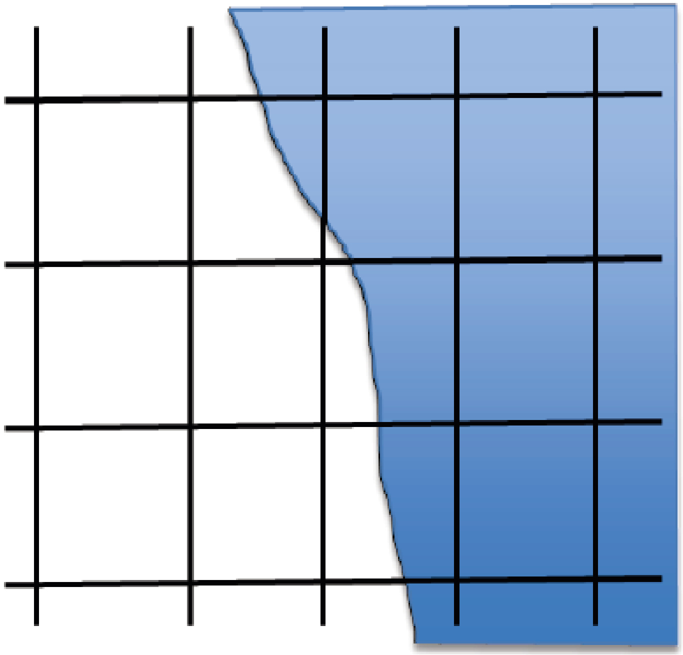




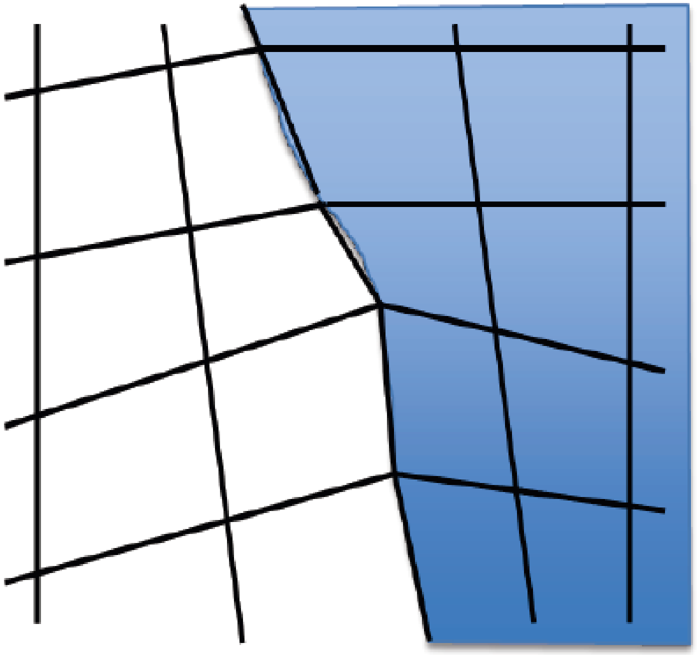


pictures courtesy of Rob Neely, Bert Still, Jeff Keasler, LLNL

Eulerian vs. Lagrangian Meshes



Eulerian mesh
(grid stays fixed)



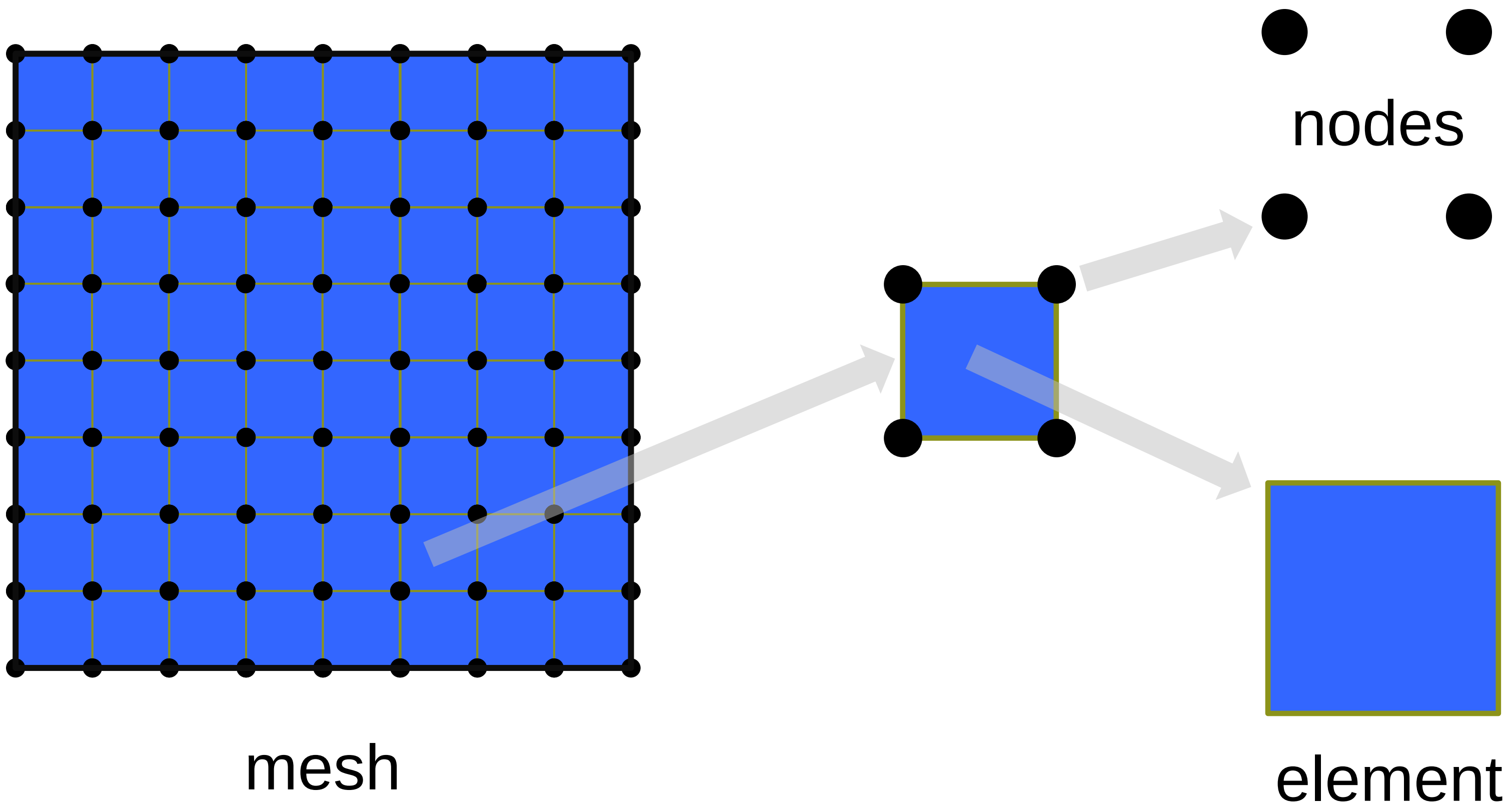
Lagrangian mesh
(grid adapts to materials)

Image Source: LULESH specification, LLNL-TR-490254
https://computation.llnl.gov/casc/ShockHydro/

LULESH Compared to a Real Hydrocode

- **LULESH**
  - Structured input provided (3D regular)
  - Single material per cell
- **Real Hydrocodes**
  - Unstructured input (compact, irregular)
  - Could have multiple materials in a cell
- **But: LULESH uses code idioms similar to those in a real code, so as to stress compilation and execution similarly**

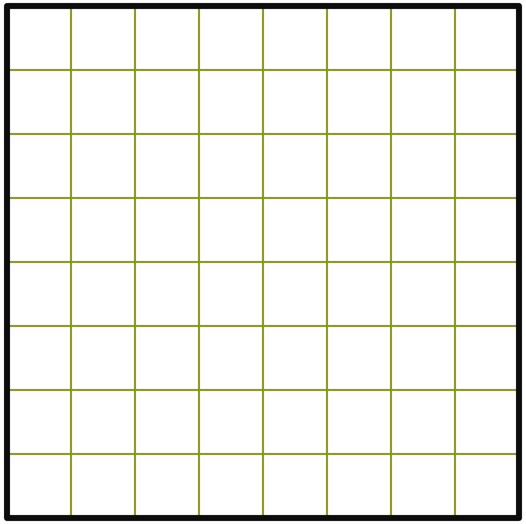
Fundamental LULESH Concepts/Terminology



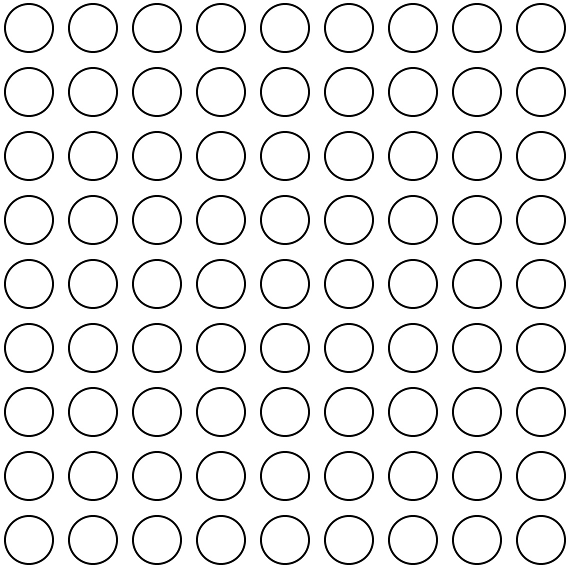
Chapel Representation (Structured)

- **Abstract Element and Node *Domains*:**

```
const nodesPerEdge = elemsPerEdge+1;
const ElemSpace = {0..#elemsPerEdge, 0..#elemsPerEdge},
      NodeSpace = {0..#nodesPerEdge, 0..#nodesPerEdge};
```



ElemSpace

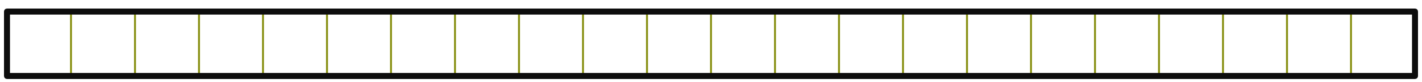


NodeSpace

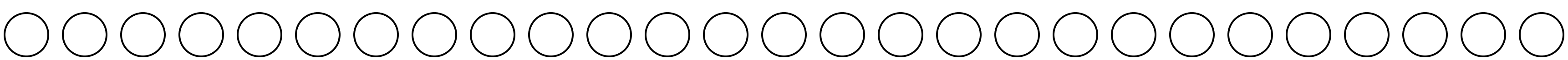
Chapel Representation (Unstructured)

- **Abstract Element and Node *Domains*:**

```
const ElemSpace = {0..#numElems},
      NodeSpace = {0..#numNodes};
```



ElemSpace

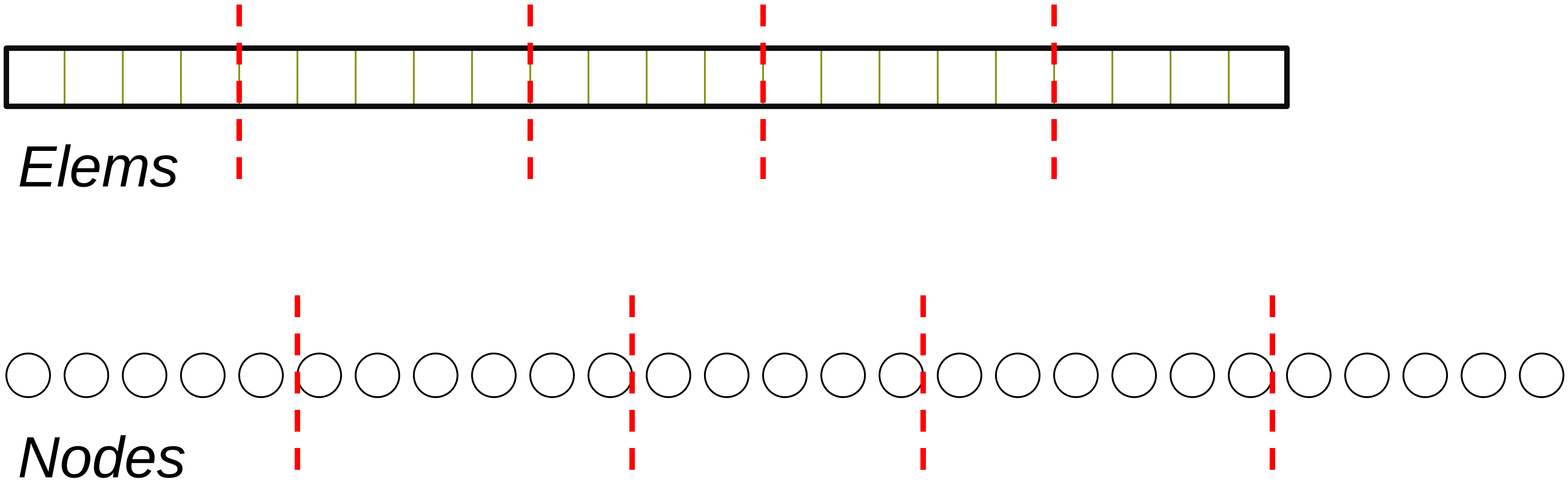


NodeSpace

Chapel Representation (Multi-locale)

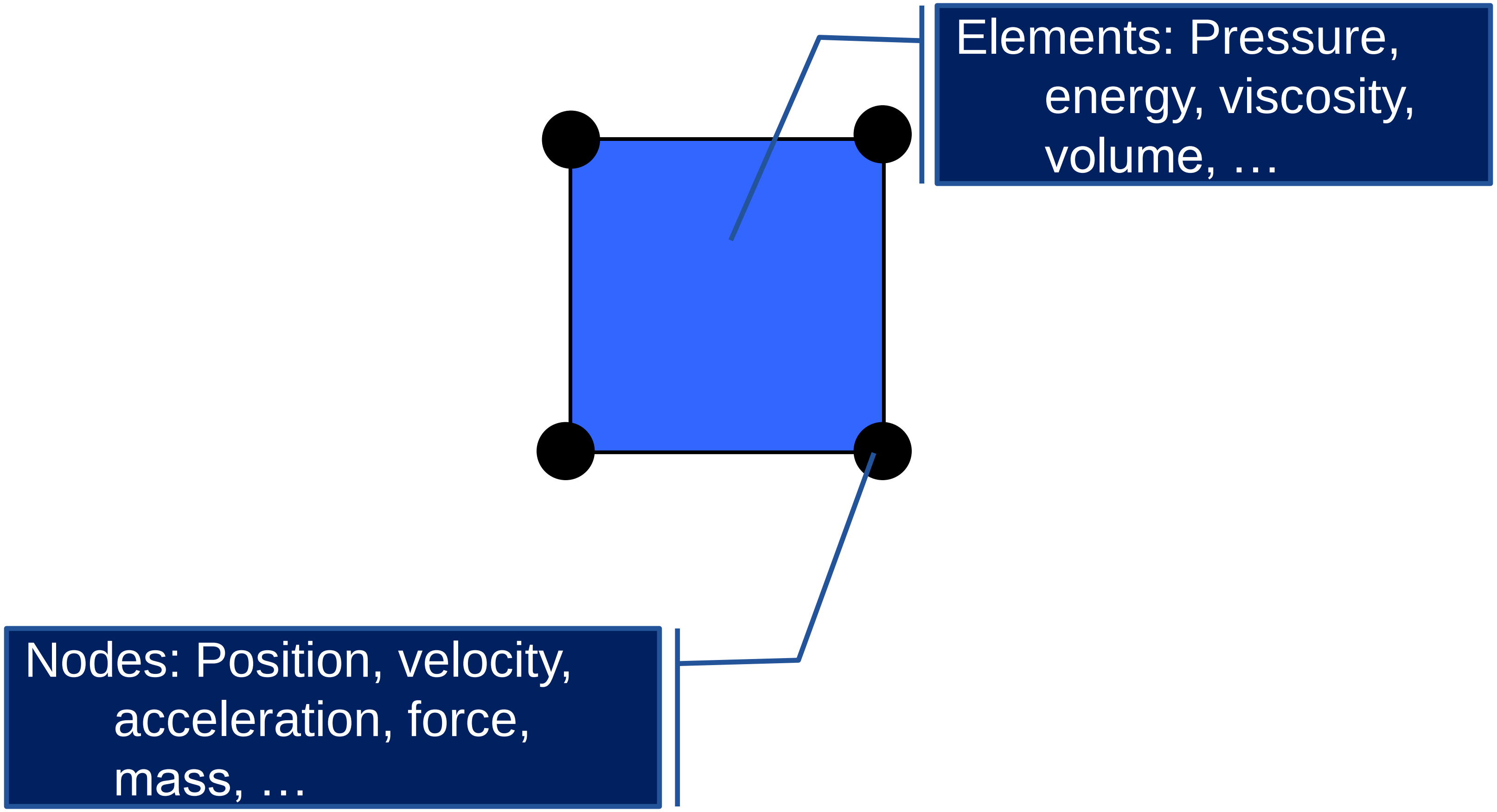
- **Distributed Element/Node Domains:**

```
const Elems = ElemSpace dmapped Block(ElemSpace),
      Nodes = NodeSpace dmapped Block(NodeSpace);
```



Element and Node Fields

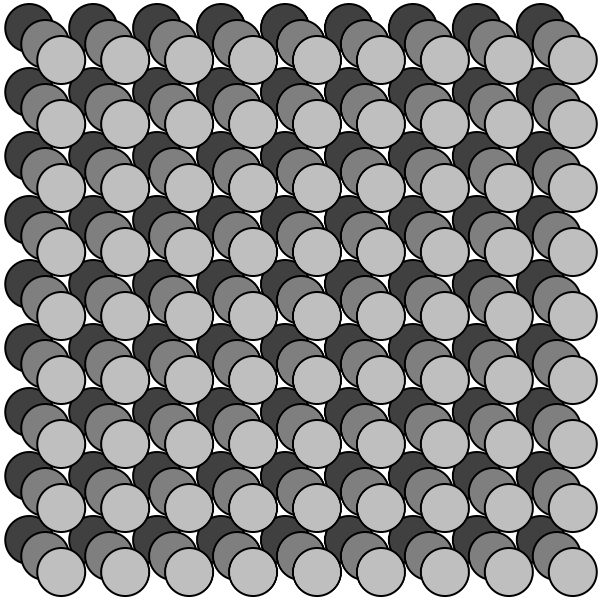
- Some variables (*fields*) are associated with elements, others with nodes.



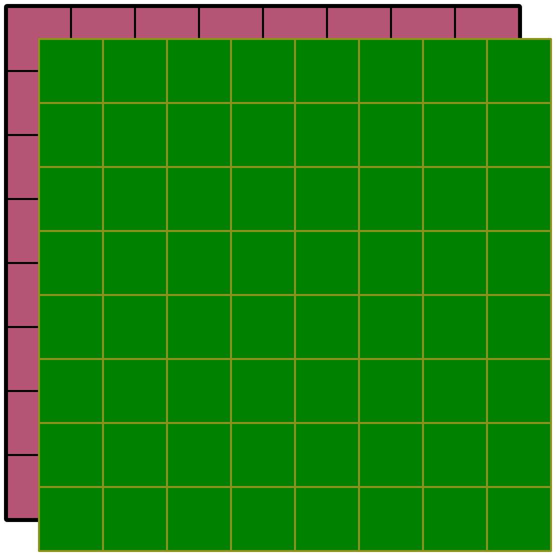
Representation of Fields in Chapel

- **Sample field declarations:**

```
var x, y, z: [Nodes] real;
var e, p: [Elems] real;
```



x, y, z

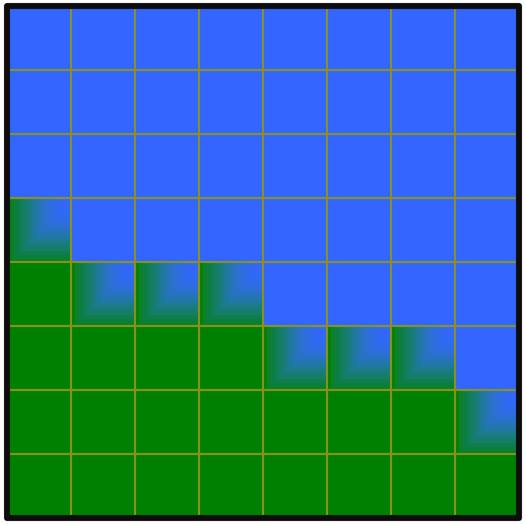


e, p

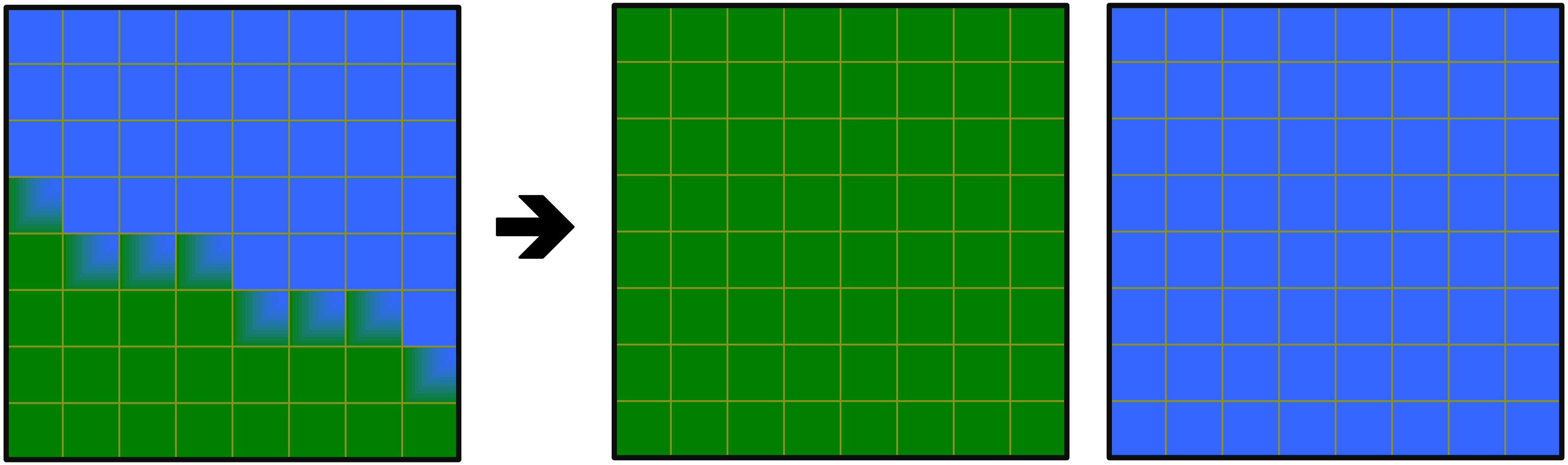
(Conceptual representation)

Materials Representation

- Not all elements will contain all materials, and some will contain combinations



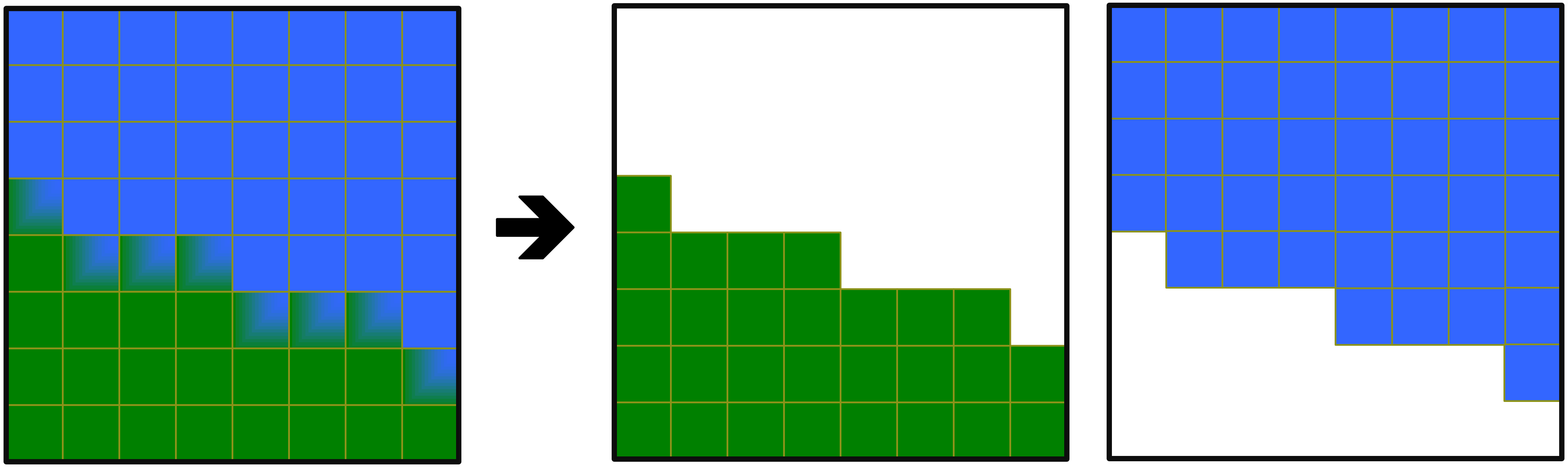
Materials Representation (Dense)



naïve approach: store all materials everywhere
(reasonable for LULESH, but not in practice)

```
const Mat1Elems = Elems,
      Mat2Elems = Elems;
```

Materials Representation (Sparse)



improved approach: use sparse subdomains to only store materials where necessary

```
var Mat1Elems: sparse subdomain(Elems) = enumerateMat1Locs(),
    Mat2Elems: sparse subdomain(Elems) = enumerateMat2Locs();
```

LULESH in Chapel

LULESH in Chapel

trunk/test/release/examples/benchmarks/lulesh/*.chpl in the SourceForge repository, as of r21020 (2/14/13)

There are: 1288 lines of code
266 lines of comments
487 blank lines

(the C+MPI+OpenMP version is nearly 4x bigger)

LULESH in Chapel

The Representation Dependent Code

```
const ElemSpace = if use3DRepresentation
                    then {0..#elemsPerEdge, 0..#elemsPerEdge, 0..#elemsPerEdge}
                    else {0..#numElems},
      NodeSpace = if use3DRepresentation
                    then {0..#nodesPerEdge, 0..#nodesPerEdge, 0..#nodesPerEdge}
                    else {0..#numNodes};

const Elems = if useBlockDist then ElemSpace dmapped Block(ElemSpace)
                              else ElemSpace,
      Nodes = if useBlockDist then NodeSpace dmapped Block(NodeSpace)
                              else NodeSpace;

var elemToNode: [Elems] nodesPerElem*index(Nodes);

var XSym, YSym, ZSym: sparse subdomain(Nodes);

const MatElems: MatElemsType = if sparseMaterials then enumerateMatElems()
                                                  else Elems;
```

domains for elements and nodes

The Representation Dependent Code

```
const ElemSpace = if use3DRepresentation
                    then {0..#elemsPerEdge, 0..#elemsPerEdge, 0..#elemsPerEdge}
                    else {0..#numElems},
      NodeSpace = if use3DRepresentation
                    then {0..#nodesPerEdge, 0..#nodesPerEdge, 0..#nodesPerEdge}
                    else {0..#numNodes};

const Elems = if useBlockDist then ElemSpace dmapped Block(ElemSpace)
                              else ElemSpace,
      Nodes = if useBlockDist then NodeSpace dmapped Block(NodeSpace)
                              else NodeSpace;

var elemToNode: [Elems] nodesPerElem*index(Nodes);

var XSym, YSym, ZSym: sparse subdomain(Nodes);

const MatElems: MatElemsType = if sparseMaterials then enumerateMatElems()
                                                  else Elems;
```

potentially distributed domains for elements and nodes

The Representation Dependent Code

```
const ElemSpace = if use3DRepresentation
                    then {0..#elemsPerEdge, 0..#elemsPerEdge, 0..#elemsPerEdge}
                    else {0..#numElems},
      NodeSpace = if use3DRepresentation
                    then {0..#nodesPerEdge, 0..#nodesPerEdge, 0..#nodesPerEdge}
                    else {0..#numNodes};

const Elems = if useBlockDist then ElemSpace dmapped Block(ElemSpace)
                              else ElemSpace,
      Nodes = if useBlockDist then NodeSpace dmapped Block(NodeSpace)
                              else NodeSpace;

var elemToNode: [Elems] nodesPerElem*index(Nodes);

var XSym, YSym, ZSym: sparse subdomain(Nodes);

const MatElems: MatElemsType = if sparseMaterials then enumerateMatElems()
                                                  else Elems;
```

nodes adjacent to each element

The Representation Dependent Code

```
const ElemSpace = if use3DRepresentation
                    then {0..#elemsPerEdge, 0..#elemsPerEdge, 0..#elemsPerEdge}
                    else {0..#numElems},
      NodeSpace = if use3DRepresentation
                    then {0..#nodesPerEdge, 0..#nodesPerEdge, 0..#nodesPerEdge}
                    else {0..#numNodes};

const Elems = if useBlockDist then ElemSpace dmapped Block(ElemSpace)
                              else ElemSpace,
      Nodes = if useBlockDist then NodeSpace dmapped Block(NodeSpace)
                              else NodeSpace;

var elemToNode: [Elems] nodesPerElem*index(Nodes);

var XSym, YSym, ZSym: sparse subdomain(Nodes);

const MatElems: MatElemsType = if sparseMaterials then enumerateMatElems()
                                                  else Elems;
```

symmetry planes

The Representation Dependent Code

```
const ElemSpace = if use3DRepresentation
                    then {0..#elemsPerEdge, 0..#elemsPerEdge, 0..#elemsPerEdge}
                    else {0..#numElems},
      NodeSpace = if use3DRepresentation
                    then {0..#nodesPerEdge, 0..#nodesPerEdge, 0..#nodesPerEdge}
                    else {0..#numNodes};

const Elems = if useBlockDist then ElemSpace dmapped Block(ElemSpace)
                              else ElemSpace,
      Nodes = if useBlockDist then NodeSpace dmapped Block(NodeSpace)
                              else NodeSpace;

var elemToNode: [Elems] nodesPerElem*index(Nodes);

var XSym, YSym, ZSym: sparse subdomain(Nodes);

const MatElems: MatElemsType = if sparseMaterials then enumerateMatElems()
                                                  else Elems;
```

domain describing elements that contain the material

The Representation Dependent Code

```
proc MatElemsType type {
  if sparseMaterials {
    if (printWarnings && useBlockDist && numLocales > 1) then
      writeln("WARNING: The LULESH Material Elements (MatElems) are not yet\n",
              "         distributed, so result in excessive memory use on,\n",
              "         and communication with, locale 0\n");
    return sparse subdomain(Elems);
  } else
    return Elems.type;
}

iter elemToNodes(elem) {
  for param i in 1..nodesPerElem do
    yield elemToNode[elem][i];
}

iter elemToNodesTuple(e) {{
  for i in 1..nodesPerElem do
    yield (elemToNode[e][i], i);
}
```

the type of the domain describing elements that contain the material

The Representation Dependent Code

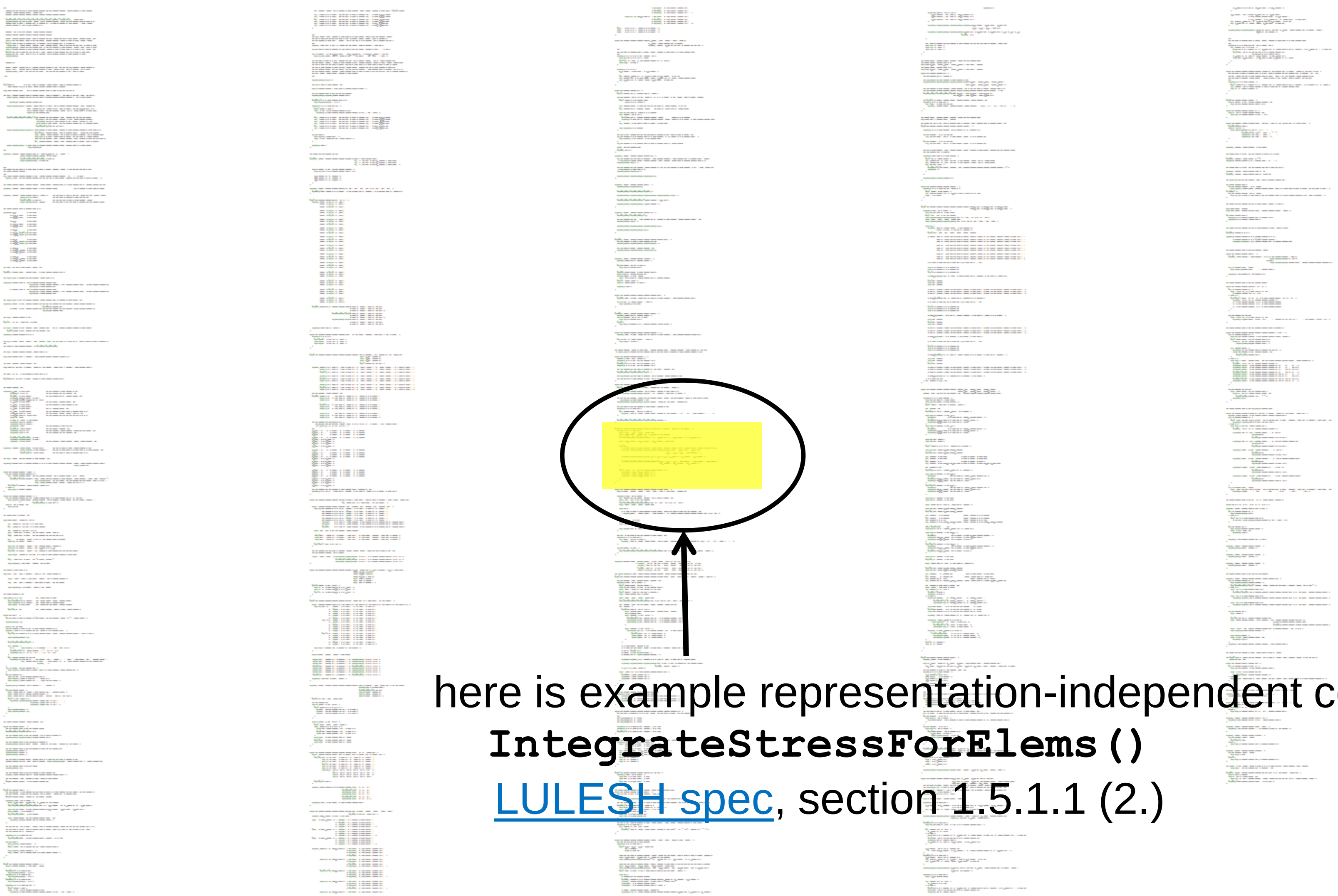
```
proc MatElemsType type {
  if sparseMaterials {
    if (printWarnings && useBlockDist && numLocales > 1) then
      writeln("WARNING: The LULESH Material Elements (MatElems) are not yet\n",
              "         distributed, so result in excessive memory use on,\n",
              "         and communication with, locale 0\n");
    return sparse subdomain(Elems);
  } else
    return Elems.type;
}
```

```
iter elemToNodes(elem) {
  for param i in 1..nodesPerElem do
    yield elemToNode[elem][i];
}

iter elemToNodesTuple(e) {{
  for i in 1..nodesPerElem do
    yield (elemToNode[e][i], i);
}
```

iterators mapping elements to their adjacent nodes

LULESH in Chapel



here is example representation-independent code:

`IntegrateStressForElems()`

LULESH spec, section 1.5.1.1 (2.)

Representation Independent Physics!

```
proc IntegrateStressForElems(sigxx, sigyy, sigzz, determ) {
  forall k in Elems {
    var b_x, b_y, b_z: 8*real;
    var x_local, y_local, z_local: 8*real;
    localizeNeighborNodes(k, x, x_local, y, y_local, z, z_local);

    var fx_local, fy_local, fz_local: 8*real;

    local {
      /* Volume calculation involves extra work for numerical consistency. */
      CalcElemShapeFunctionDerivatives(x_local, y_local, z_local,
                                       b_x, b_y, b_z, determ[k]);

      CalcElemNodeNormals(b_x, b_y, b_z, x_local, y_local, z_local);

      SumElemStressesToNodeForces(b_x, b_y, b_z, sigxx[k], sigyy[k], sigzz[k],
                                  fx_local, fy_local, fz_local);
    }

    for (noi, t) in elemToNodesTuple(k) {
      fx[noi].add(fx_local[t]);
      fy[noi].add(fy_local[t]);
      fz[noi].add(fz_local[t]);
    }
  }
}
```

loop over all elements

collect nodes neighboring this element; localize node fields

update node forces from element stresses

All of this is independent of:

- structured vs. unstructured mesh
- shared vs. distributed data
- sparse vs. dense representation

Codesign

LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

Summer 2011: Cray intern ports LULESH to Chapel

- *caveat:* used structured mesh to represent data arrays

LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

Summer 2011: Cray intern ports LULESH to Chapel

- *caveat:* used structured mesh to represent data arrays

Nov 2011: Chapel team tunes LULESH for single-node performance

Dec 2011: Chapel team visits LLNL (talk, tutorial, 1-on-1 sessions)

LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

Summer 2011: Cray intern ports LULESH to Chapel

- *caveat:* used structured mesh to represent data arrays

Nov 2011: Chapel team tunes LULESH for single-node performance

Dec 2011: Chapel team visits LLNL (talk, tutorial, 1-on-1 sessions)

Mar 2012: Jeff Keasler (LLNL) visits Cray to pair-program

- in one afternoon, converted from structured to unstructured mesh
- impact on code minimal (mostly in declarations)

LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

Summer 2011: Cray intern ports LULESH to Chapel

- *caveat:* used structured mesh to represent data arrays

Nov 2011: Chapel team tunes LULESH for single-node performance

Dec 2011: Chapel team visits LLNL (talk, tutorial, 1-on-1 sessions)

Mar 2012: Jeff Keasler (LLNL) visits Cray to pair-program

- in one afternoon, converted from structured to unstructured mesh
- impact on code minimal (mostly in declarations)

Apr 2012: LLNL reports on collaboration at Salishan

Apr 2012: Chapel 1.5.0 release includes LULESH as an example code

LULESH in Chapel, Codesign Timeline

Apr 2011: LLNL expresses interest in Chapel at Salishan Conference

- Introduced us to the LULESH benchmark

Summer 2011: Cray intern ports LULESH to Chapel

- *caveat:* used structured mesh to represent data arrays

Nov 2011: Chapel team tunes LULESH for single-node performance

Dec 2011: Chapel team visits LLNL (talk, tutorial, 1-on-1 sessions)

Mar 2012: Jeff Keasler (LLNL) visits Cray to pair-program

- in one afternoon, converted from structured to unstructured mesh
- impact on code minimal (mostly in declarations)

Apr 2012: LLNL reports on collaboration at Salishan

Apr 2012: Chapel 1.5.0 release includes LULESH as an example code

Sep-Nov 2012: performance tuning

Nov 2012: SC12

- Chapel HPC Challenge entry
- LLNL talk at the Chapel Lightning Talks BoF
- Cray talk at Proxy Apps BoF

Dec 2012: Multi-institution LULESH paper accepted to IPDPS '13

Next Steps

- **Performance Optimizations and Tuning**
  - Reductions
  - Communication optimizations
    - Aggregation
    - Overlap
    - Atomics
- **Explore array-of-structs vs. struct-of-arrays ideas**
- **Identify funding to dedicate focus on DOE proxy apps**

Codesign Takeaways for Chapel Team

- Improved comprehension of the science behind the code and data structures
- Deeper understanding of array-of-struct vs. struct-of-arrays tensions
- Awareness of performance issues based on past LLNL experience

"The opportunity to work on LULESH with computational scientists at LLNL has been incredibly valuable. In part, this is due to the level of expertise that we've had access to. In part it's due to LULESH's design: it is compact enough to be manageable for our team to understand while being realistic enough to carry weight with actual users."

Brad Chamberlain, Chapel Technical Lead, Cray Inc.

Codesign Takeaways for LULESH Team

- Impact of representation-independent features made evident firsthand
- Saw value of using global-view sparse domains to avoid local↔global index translation
- View Chapel as an opportunity for code that ports across next-gen architectures

"Chapel is a maintainable future-proof language. With additional back-end performance enhancements, we would be using it to develop science codes, with an eye towards multiphysics production codes."

Jeff Keasler, ASC code developer, LLNL

Summary of the LULESH Effort in Chapel

- **Evidence that Chapel's language design is solid**
  - Not just an HPCS technology demonstrator
- **Evidence that people are getting serious about Chapel**
  - LLNL sees Chapel as a serious contender for hydrocodes
- **Co-design success story**
  - Access to experts for a code that people actually care about
  - Feedback on the language and implementation
  - New challenges for the language and implementation

Questions?