


Legion Bootcamp: Partitioning

Sean Treichler

Legion Partitioning

- Partitioning Overview
- Index Spaces vs. Logical Regions
- Partitioning Operations
- Performance
- Current Status
- Related Features

Partitioning Overview

- Partitioning is essential for exposing parallelism and limiting data movement
- Want to be able to describe data used by a task as precisely as possible
  - i.e. maximize *expressivity*
- Also want to be able to compute and work with partitions as efficiently as possible
  - Optimization requires the ability to *analyze* the operations
- Unavoidable tradeoff between expressivity and tractability of analysis
  - Have to choose some point on the spectrum

One Extreme: Simplicity

- PGAS languages (e.g. X10, UPC, Chapel) generally provide only simple array-based distribution methods
  - e.g. block, cyclic, blockcyclic

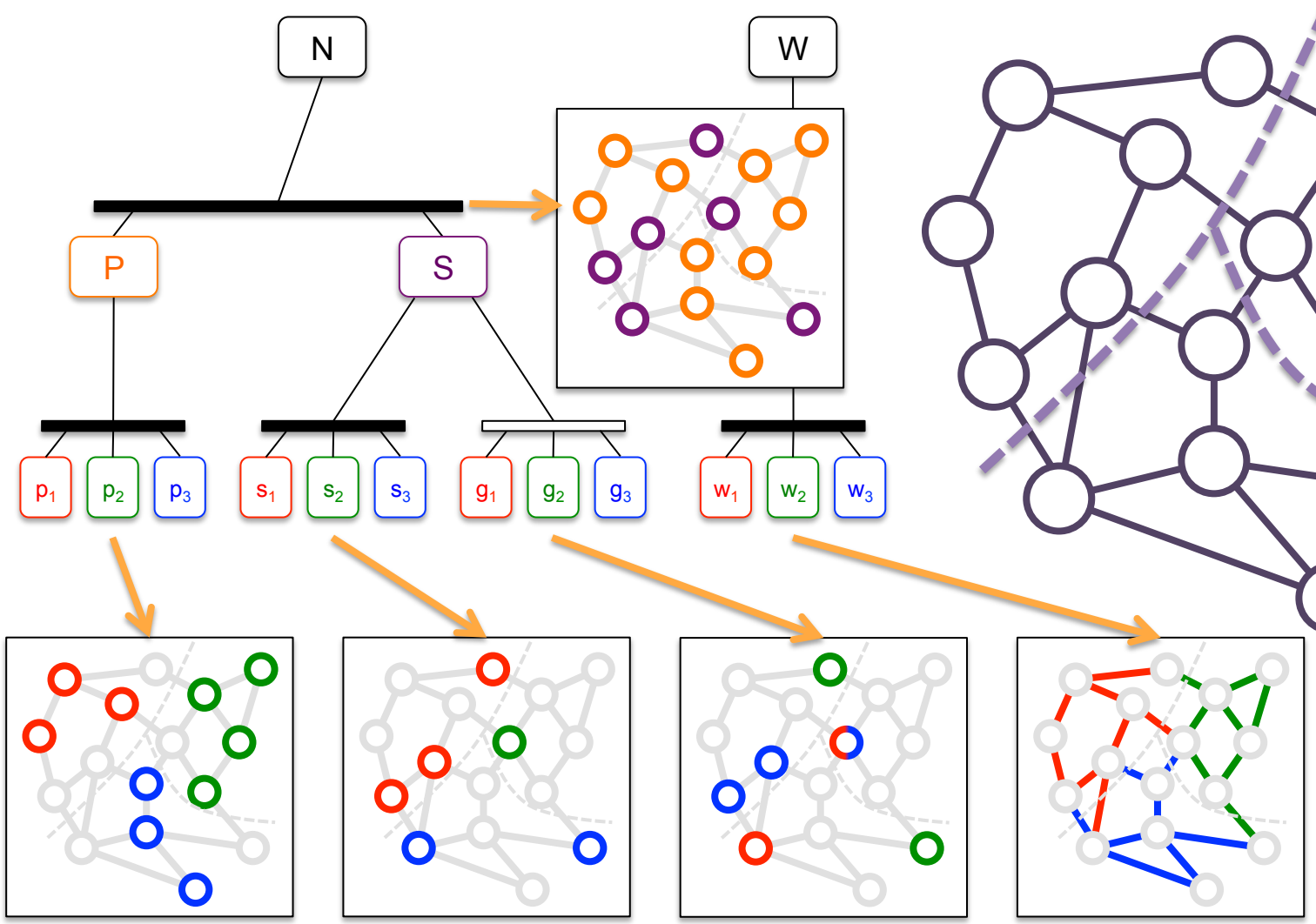
- Pros:
  - simple for programmer to describe
  - simple for compiler to verify consistency
  - simple for runtime to implement

- Cons:
  - no support for irregular (or even semi-regular) data structures
  - no support for irregular partitions of structured data
  - no support for aliased or multiple partitions

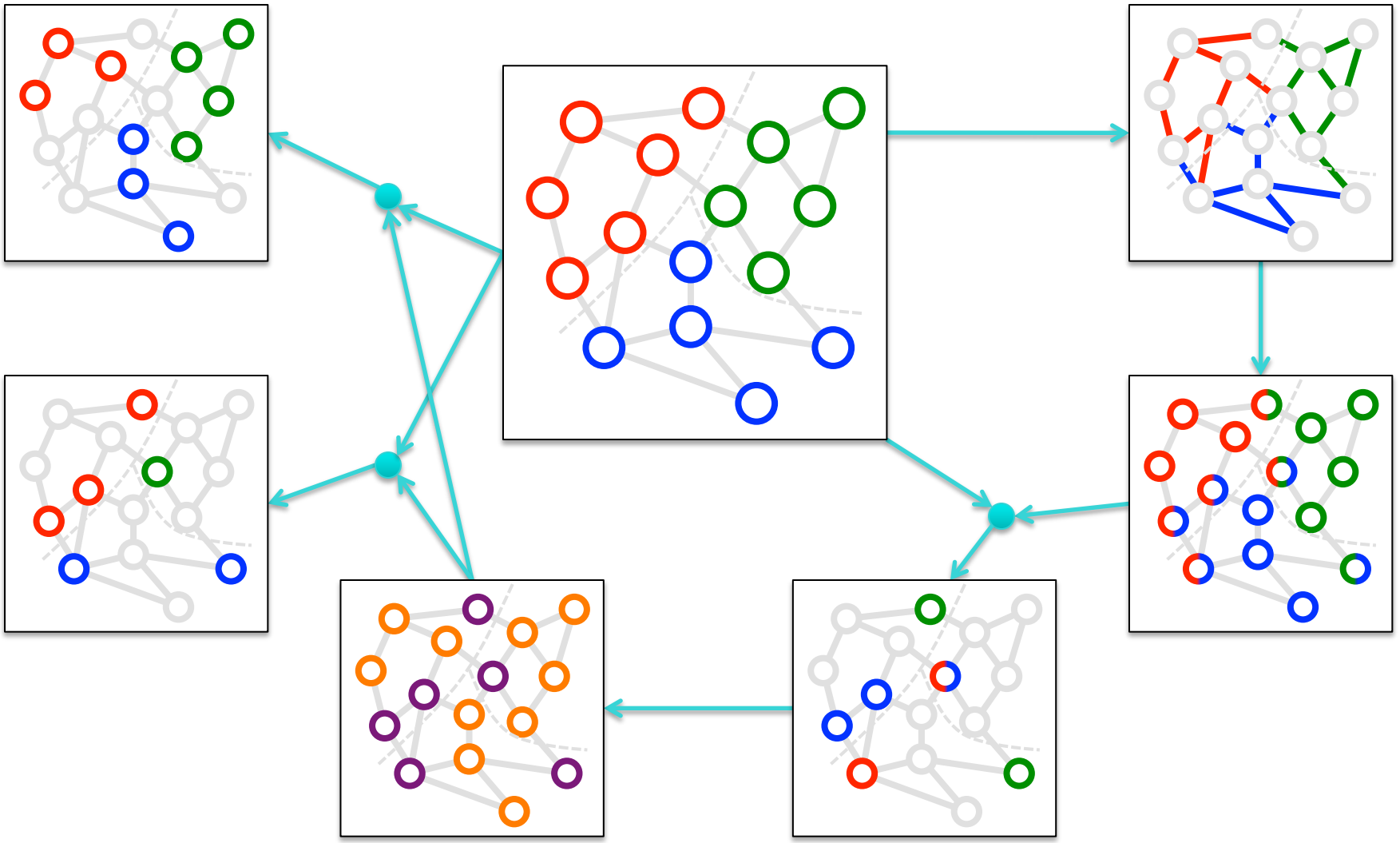
Other Extreme: Expressivity

- Old Legion partitioning used general-purpose coloring object for ALL partitioning operations
  - Application able to color each element any way it wants
- Pros:
  - support for arbitrary irregularity in data and/or partitioning
  - support for aliased partitions, multiple partitions
- Cons:
  - significant programmer effort to describe even simple partitions
  - no ability for compiler to check that related regions are partitioned consistently
  - high runtime overhead for computing and querying partitions
  - manipulation of coloring was serial, limited to single node

Circuit Partitioning: Old



Circuit: Behind the Scenes

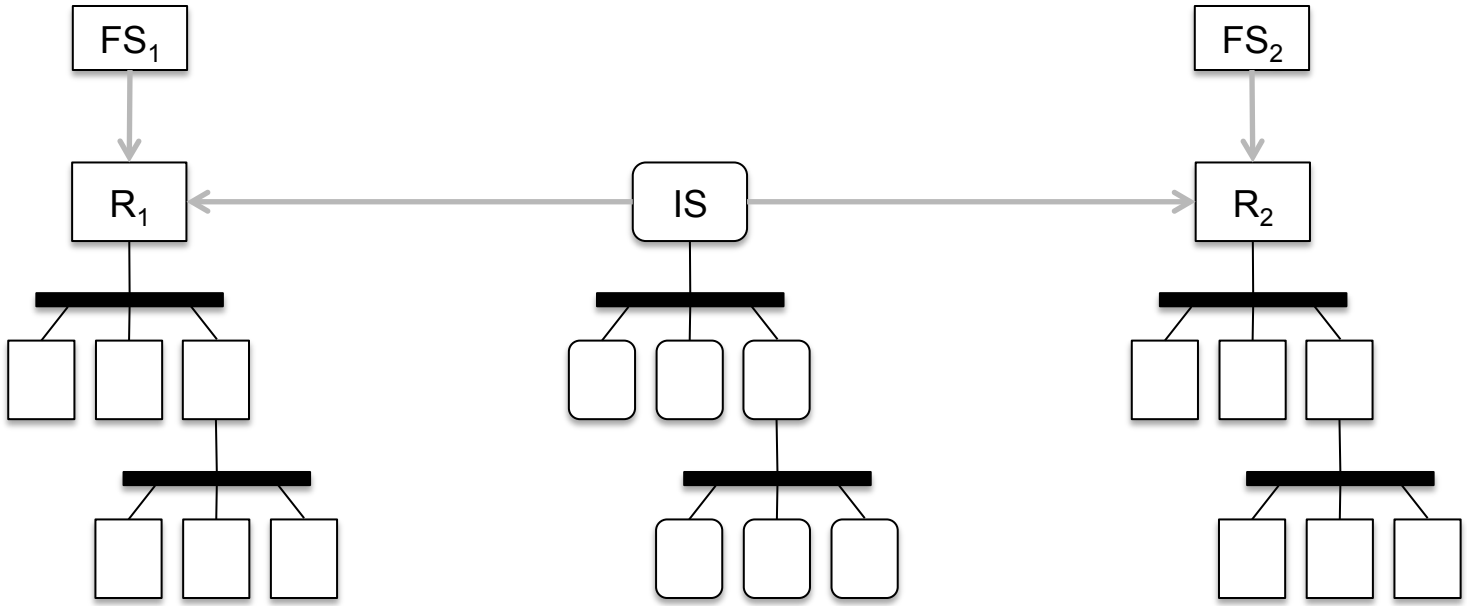


Dependent Partitioning

- A carefully chosen middle ground between these two extremes
- Allows arbitrary *independent partitions* to be computed by the application
  - But uses field data to capture intent rather than a coloring
  - Index-based partitions cover PGAS-like simple cases
- Provides an analyzable set of operations to compute *dependent partitions* from other partitions
  - Based on reachability and/or set operations
  - Consistency of dependent partitions can be verified at compile time
- Incorporated into Legion's data and execution model
  - Support for distributed partitioning, deferred execution

Index Spaces vs. Logical Regions

- A logical region is constructed from an index space and a field space
- Partitioning a logical region is actually partitioning the index space
- Partitions are usable in other regions using the same index space
- C++ API has calls to move between corresponding nodes in region, index space trees
- Regent lets you use regions any place an index space is expected

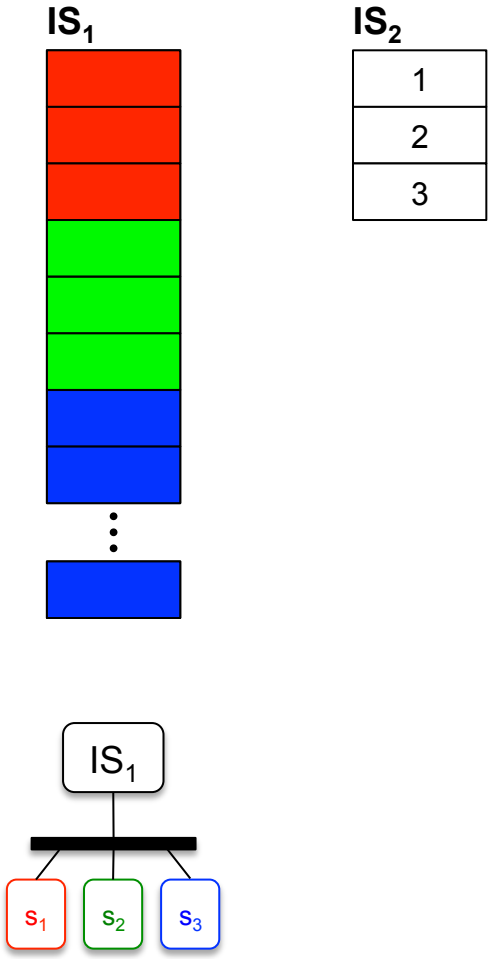


Partitioning Operations

	Independent Partitions	Dependent Partitions
Index-Based	equal weighted	restriction
Field-Based	filter	image preimage
Set Operations		union intersection difference

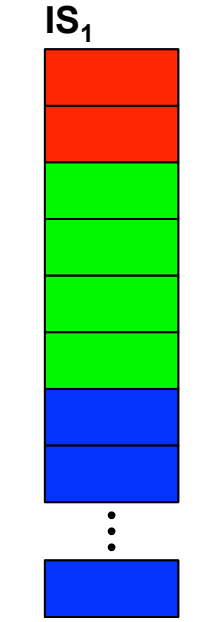
Index-Based: Equal Partition

- Splits an index space into roughly equal pieces
- Number of pieces specified with a second index space
  - One subspace for each point in that space
- Useful for structured cases
  - Or as an initial distribution when computing an unstructured partition



Index-Based: Weighted Partition

- Splits an index space into uneven pieces
- Number of pieces specified with a second index space
  - Weight for each piece specified in a field
  - Can be result of an arbitrary computation

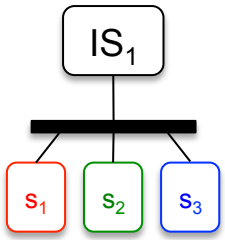


IS_2

1
2
3

LR_2

. . .	w	. . .
. . .	2	. . .
. . .	4	. . .
. . .	3	. . .

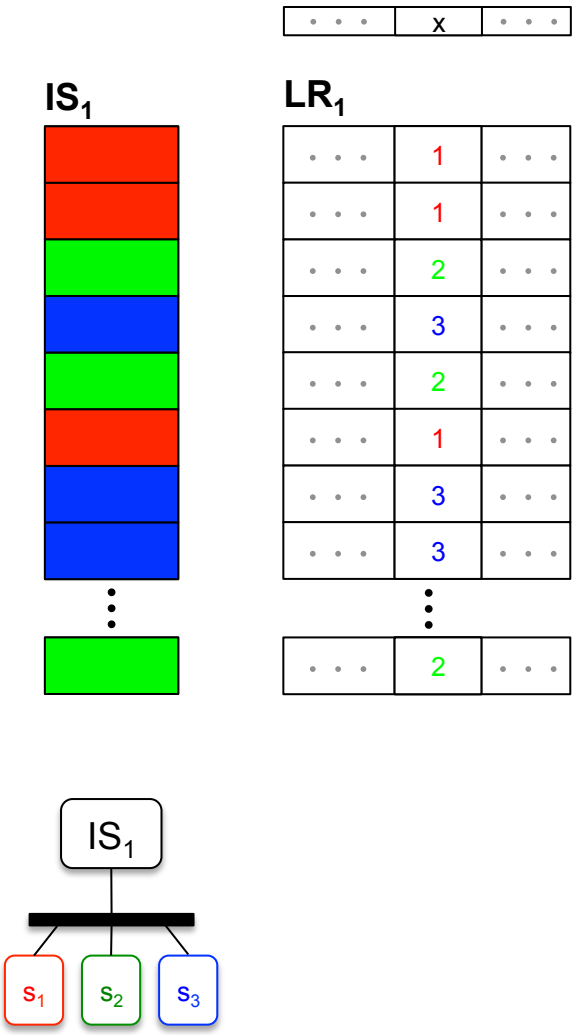


Field-Based: Filtering

- Use a field's content as the "color" of an element
  - "Color" is now a point in some index space
  - Allows desired partitioning to be computed in parallel/ distributed fashion
  - Like a "GROUP BY" in SQL
- Only raw field value for now
  - Soon: function composition

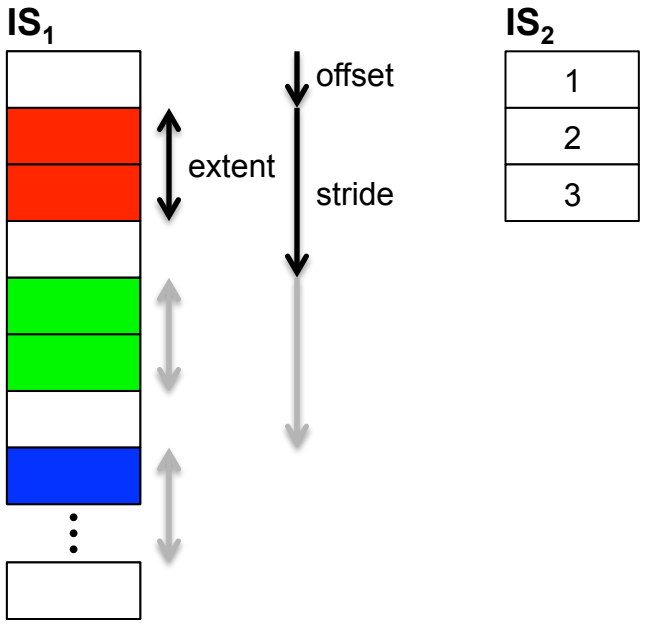
    e.g. x > 5

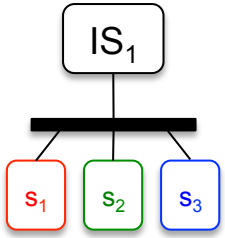
    floor(x / grid_step)



Index-Based: Restriction

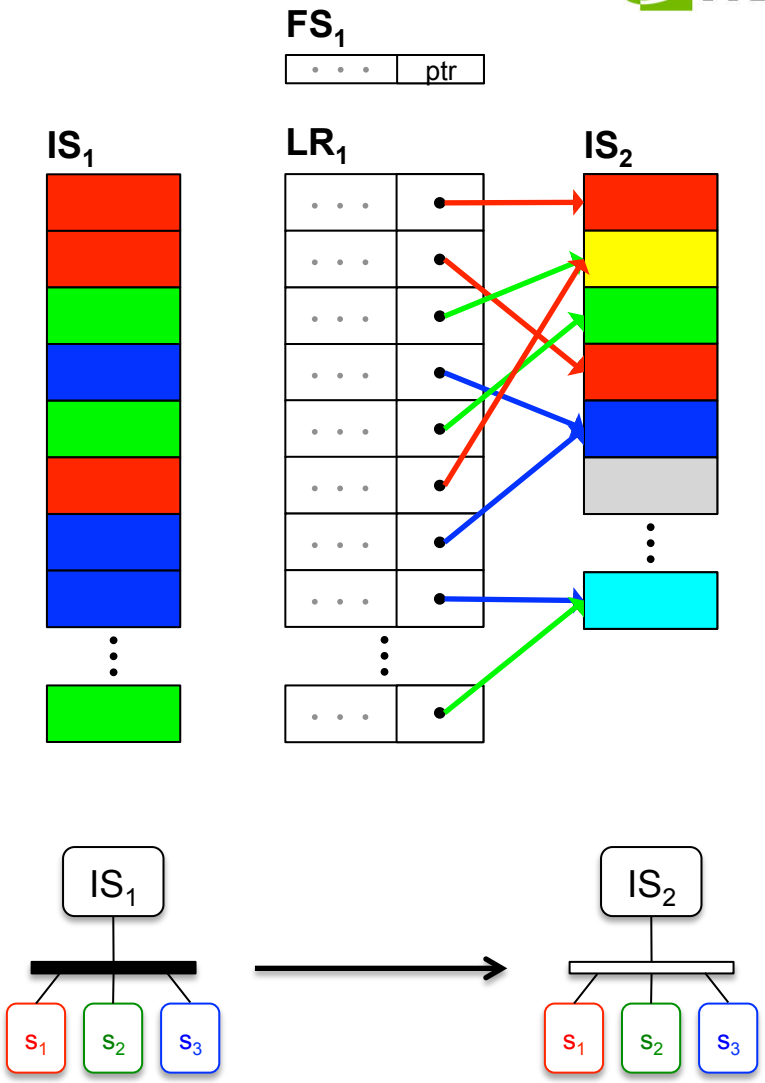
- Creates subspaces by restricting an input space to intervals in the original index type
- Number of pieces specified with a second index space
  - Each subspace has the same *extent*
  - Subspaces are separated by a *stride*, with a starting *offset*
- Again most useful in structured cases
  - Also useful for chunking unstructured work





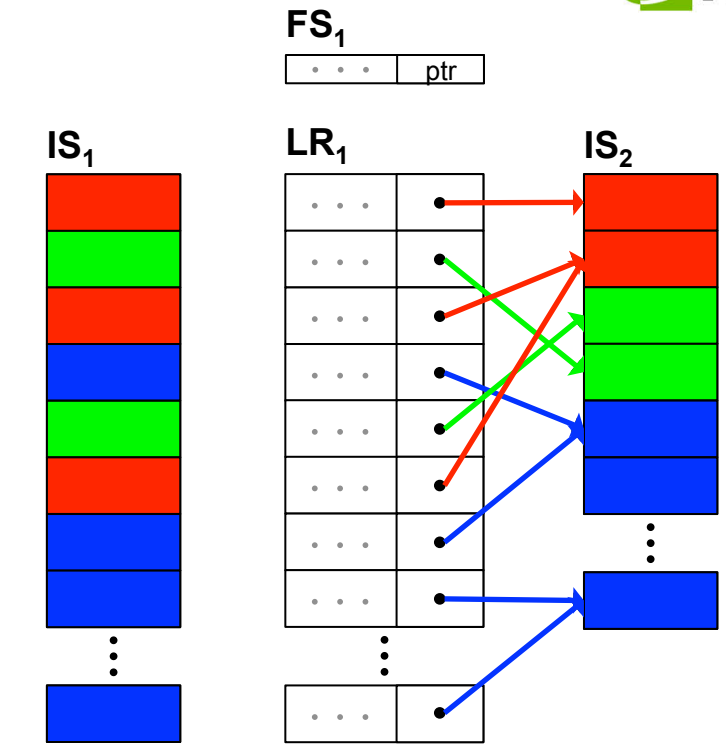
Field-Based Ops: Image

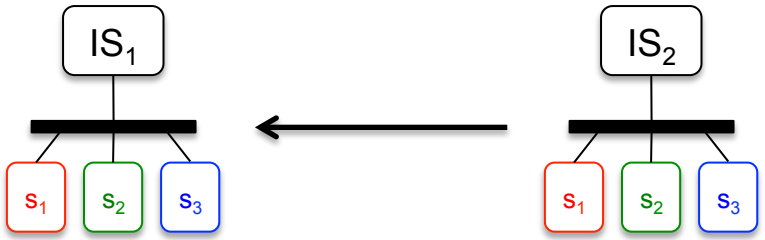
- Computes elements reachable via a field lookup
  - Equivalent to *semi-join* in relational algebra
  - Can be applied to index space or another partition
  - Computation is distributed based on location of data
- Regent understands relationship between partitions
  - Can check safety of region relation assertions at compile time



Field-Based Ops: Preimage

- Opposite of image – computes elements that reach a given subspace
  - Preserves disjointness
- Multiple images/preimages can be combined
  - can capture complex task access patterns
  - Limitation: no transitive reachability



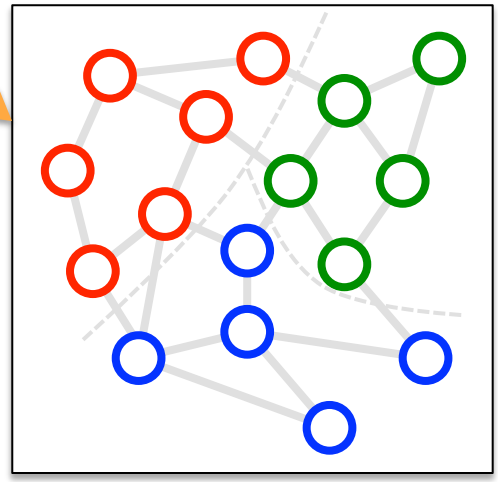


Set-Based Operations

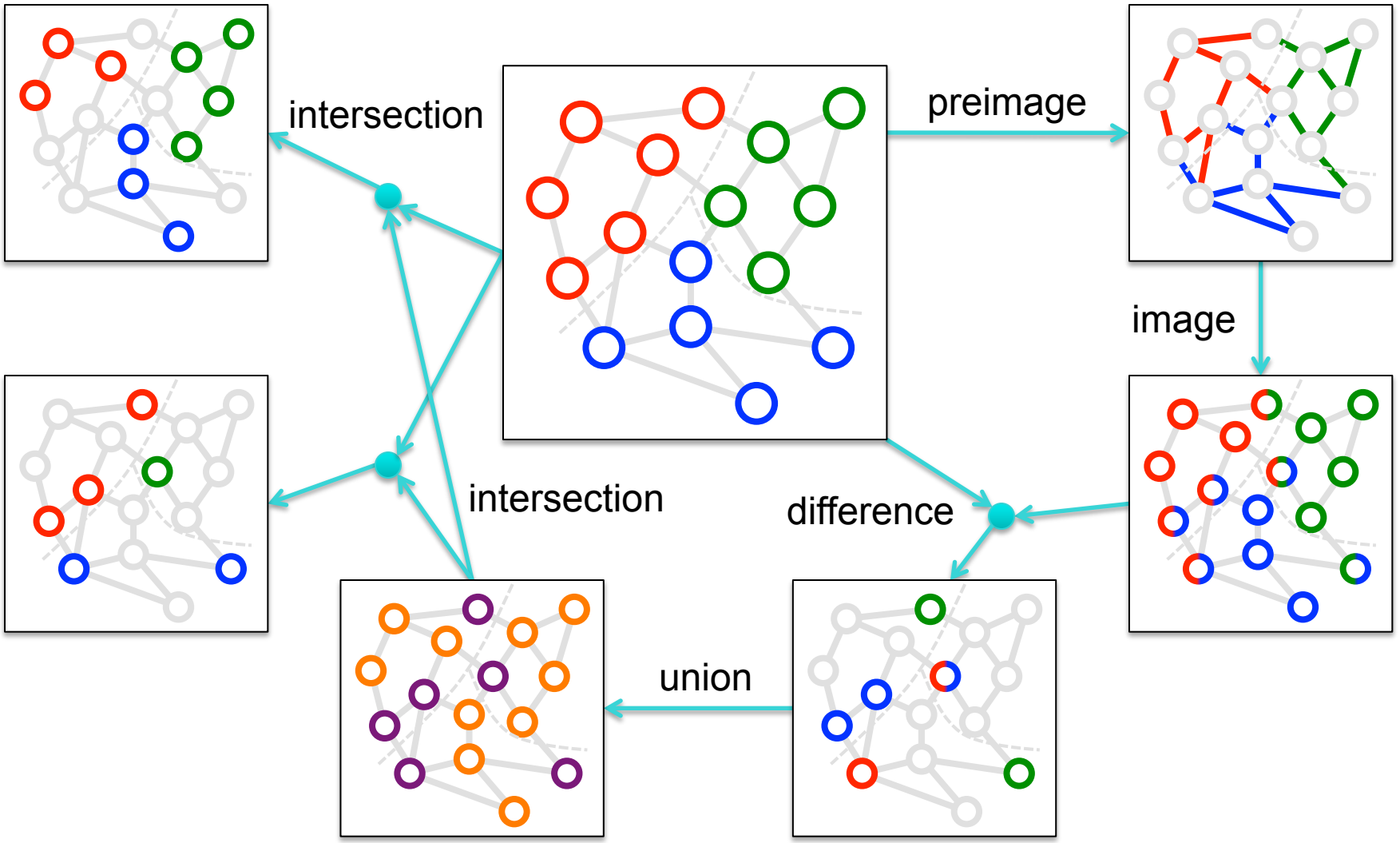
- Index spaces are sets of points, so we can compute new sets by:
  - intersection
  - union
  - difference
- Either/both operands may be a partition
  - Result is a new partition where subspace is result of operation on corresponding subspaces of inputs
- All subspaces of a partition may be reduced to a single index space by union/intersection

Circuit Partitioning: New

```
task simulate_circuit(W : region(Node), W : region(Wire))
  where reads(W), reads(N), writes(N.part_num)
{
  var part_space = ispace(int, num_subcircuits)

  parmetis(N, W, part_space) // uses index-based partition internally

  // "independent" partition from parmetis' "coloring"
  var p_nodes = partition(N, N.part_num, part_space)

  // wires partitioned by ownership of "in" node
  var p_wires = preimage(W, p_nodes, W.in_node)

  // ghost nodes are connected to our wires but not owned by us
  var p_ghost = image(N, p_wires, W.out_node) - p_nodes

  // shared nodes are those that are ghost for somebody
  var N_allshared = union_reduce(p_ghost)

  // private are the others
  var N_allprivate = N - N_allshared

  // private and shared for each circuit piece by intersection
  var p_pvt = N_allprivate & p_nodes
  var p_shr = N_allshared & p_nodes

  ...
}
```



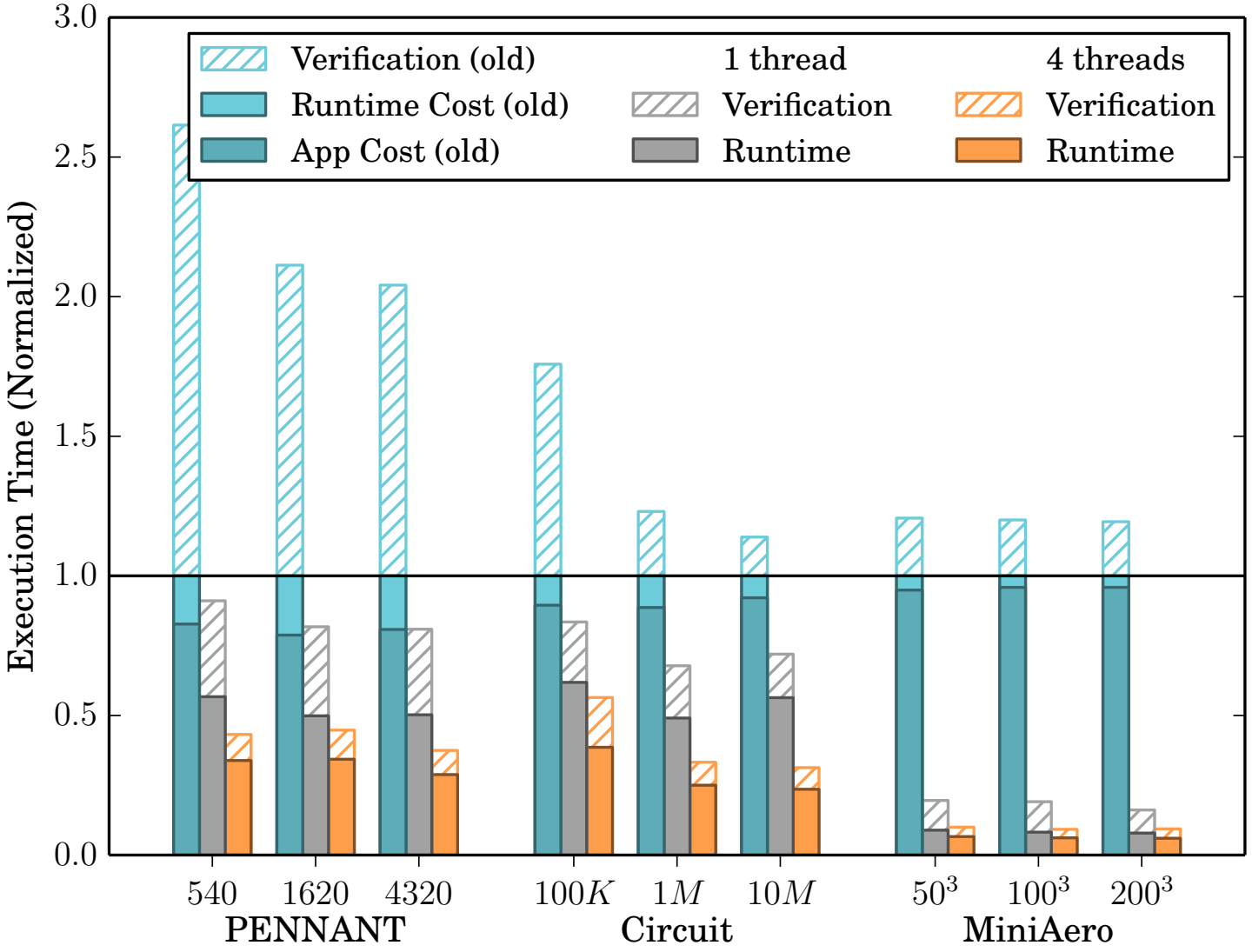
Circuit: Dependent Partitioning



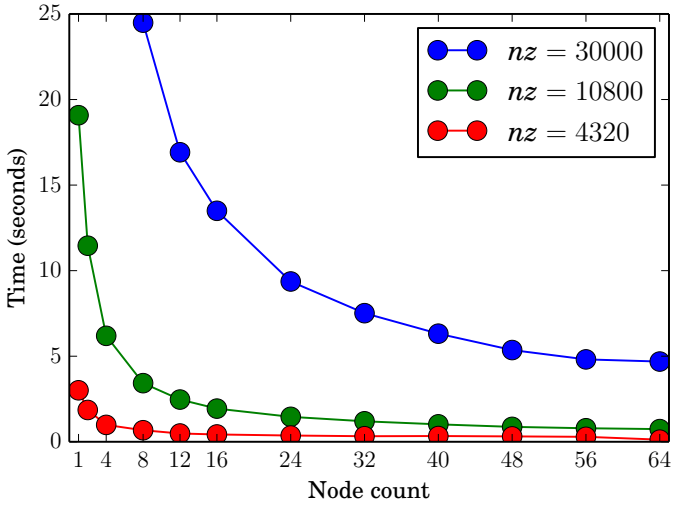
Partitioning Implementation

- Realm implements actual partitioning operations:
  - Makes operations available for all “users”
  - Lowest overhead for inter-node communication
  - Realm uses index spaces for instances, copy operations
  - Like other operations, use events and deferred execution
- Legion handles:
  - Mapping of fields in logical regions to physical instances
  - Extraction of parallelism in/around partitioning operations
  - Maintains index space tree for dynamic dependence analysis
- Regent provides:
  - more productive interface
  - compile-time checking of consistency of dependent partitions

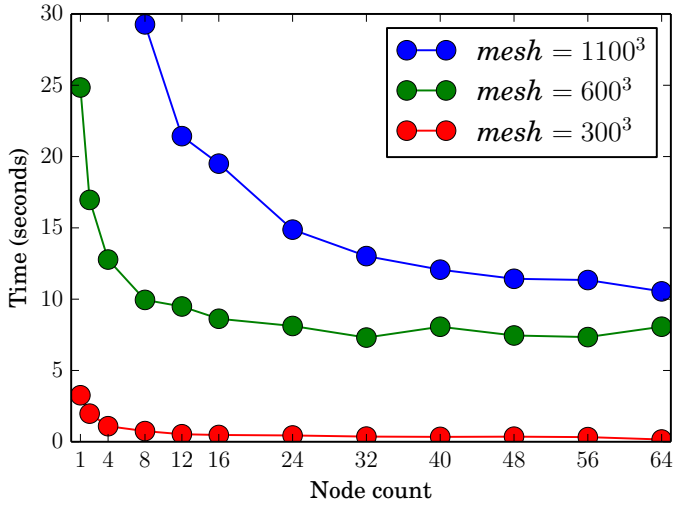
Performance: Single Node



Performance: Strong Scaling

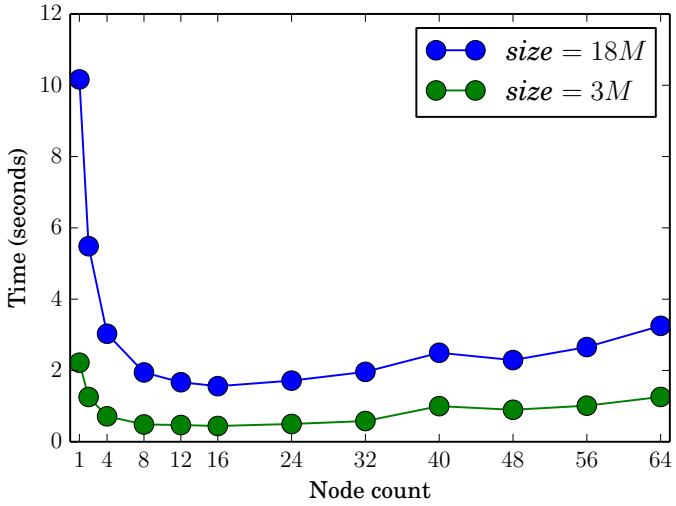


PENNANT



MiniAero

Circuit



Current Status

- Realm implementation nearly complete
  - Few optimizations related to multi-dimensional cases and bitmasks
- Legion API implemented, but needs a few tweaks
  - Possible change to mapper for field-based operations
- Regent updates in progress
  - Working out syntax for some cases
  - Need to incorporate static analysis into main Regent compiler
- ETA: end of Dec 2015

Related Features

- Not part of the core “dependent partitioning” effort
- Either part of the same rewrite or enabled by it

- Unification of structured/unstructured index spaces
- Changes to dynamic allocation
- Index space compaction
- Additional partitioning functionality/optimizations

Structured vs. Unstructured

- All index spaces will be structured
  - No more ptr_t in C++ API (Regent may keep it)
  - Index space has a base type (e.g. int32, int64) and
  - Dimensionality (e.g. 1, 2, 3, ...)
  - Implemented using templates for speed/extensibility
- Index spaces may be sparse
  - Consisting of set of points or dense subrectangles
  - Dense index spaces are an (optimized) special case
- Coming as part of partitioning changes (Dec 2015)
  - Realm ready – need to push templating into Legion API
  - May not be 100% backwards-compatible for apps

Dynamic Allocation

- Index spaces are now immutable in Realm
  - Improves performance of iterators, partitioning operations
  - Interactions between alloc/free and physical instances source of continuing problems
- Alloc/free can be implemented at “user-level” using a boolean field (i.e. is_allocated)
  - Special instance layout for boolean fields allows fast find_first_set and find_first_unset
- TBD how high up we push this (Legion?, Regent?)
  - Standard tension between programmer control and ease of use
- Working with a few test cases to weigh pros/cons
  - Iso-surface generation, particle/fluid interactions
- Hoping to settle on plan in next couple weeks
  - Implementation early next year

Index Space Compaction

- Standard region instances are direct mapped arrays
  - Allows for efficient element access, iteration
  - Wastes space for sparse regions
- Hash map saves space for sparse instances
  - Extra overhead on lookup, non-linear access patterns bad
  - Sometimes explicit compaction of data is the best answer
- Partitioning operation that accepts a sparse index space and computes:
  - New (dense) index space
  - Fields that map between spaces (either/both directions)
- Support for indirect (i.e. scatter or gather) copy operations
- Also using iso-surface as driving example (ETA early next year)

Functions, Complex Operations

- Partitioning operations can use (pure) functions in place of (or composed with) field data
  - Useful for dependent partitioning on structured grids
  - Even more powerful when JIT compilation is added
  - With source (or IR), static analysis of functions is possible, enabling further optimizations
- Many dependent partitions use multiple operations with intermediate values that are discarded
  - Opportunities to fuse these operations
  - Lots of work on this in the database community
- ETA later next year
  - Will benefit from having more examples of actual dependent partitioning usage