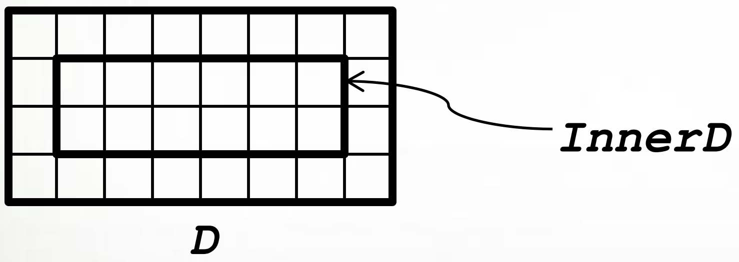
THE SUPERCOMPUTER COMPANY

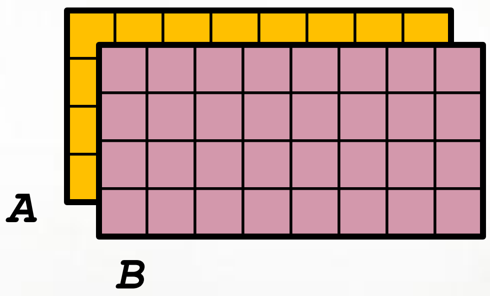
Chapel: Domain Maps

(Layouts and Distributions)

Flashback: Data Parallelism

- Domains are first-class index sets
  - Specify the size and shape of arrays
  - Support iteration, array operations, etc.





Data Parallelism: Implementation Qs

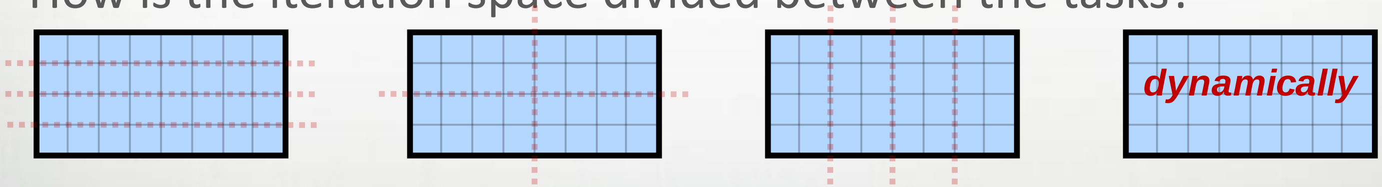
Q1: How are arrays laid out in memory?

- Are regular arrays laid out in row- or column-major order? Or…?

- What data structure is used to store sparse arrays? (COO, CSR, …?)

Q2: How are data parallel operators implemented?

- How many tasks?
- How is the iteration space divided between the tasks?



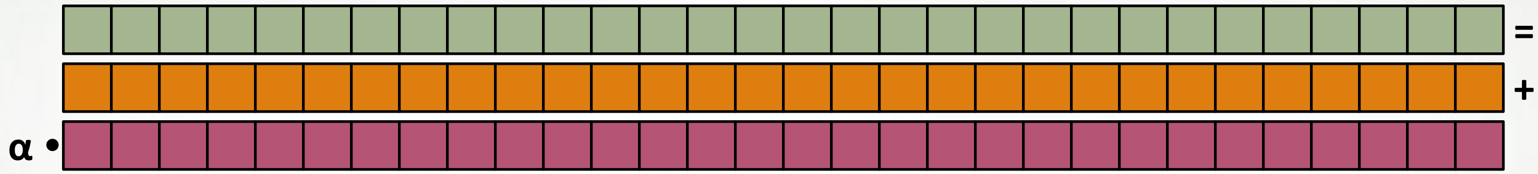
A: Chapel's *domain maps* are designed to give the user full control over such decisions

Outline

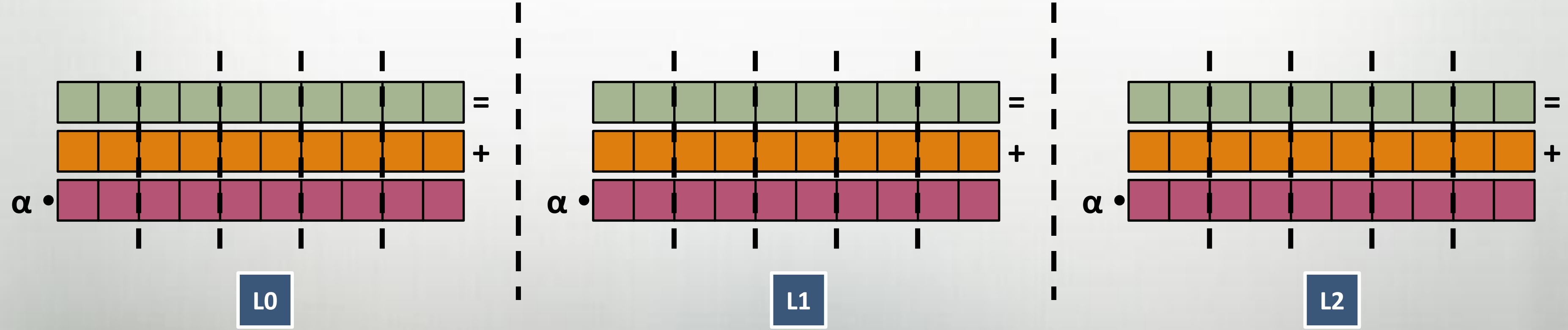
- Data Parallelism Revisited
- Domain Maps
  - Layouts
  - Distributions
- Chapel Standard Layouts and Distributions
- User-defined Domain Maps

Domain Maps

Domain maps are "recipes" that instruct the compiler how to map the global view of a computation…



…to a locale's memory and processors:



Domain Map Definitions

Domain maps define:

- Ownership of domain indices and array elements
- Underlying representation of indices and elements
- Standard operations on domains and arrays
  - E.g, iteration, slicing, access, reindexing, rank change
- How to farm out work
  - E.g., forall loops over distributed domains/arrays

Domain maps are built using Chapel concepts

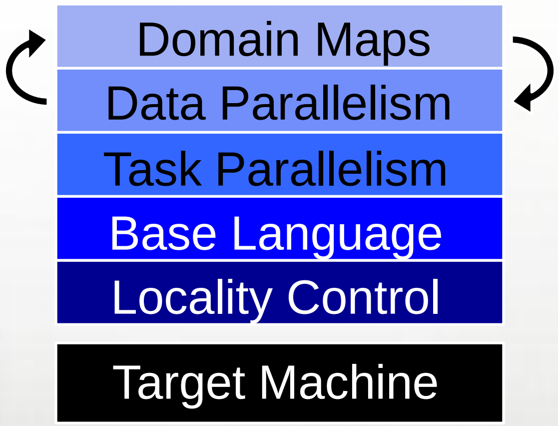
- classes, iterators, type inference, generic types
- task parallelism
- locales and on-clauses
- domains and arrays

Multiresolution Language Design, Revisited

Multiresolution Design: Support multiple tiers of features

- higher levels for programmability, productivity
- lower levels for performance, control
- build the higher-level concepts in terms of the lower-

Chapel language concepts



- separate concerns appropriately for clean design

Domain Maps: Layouts and Distributions

Domain Maps fall into two major categories:

layouts: target a single shared memory segment

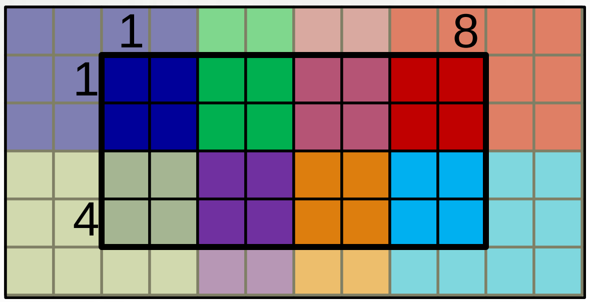
- (that is, a desktop machine or multicore node)
- **examples:** row- and column-major order, tilings, compressed sparse row

distributions: target distinct memory segments

- (that is a distributed memory cluster or supercomputer)
- **examples:** Block, Cyclic, Block-Cyclic, Recursive Bisection, …

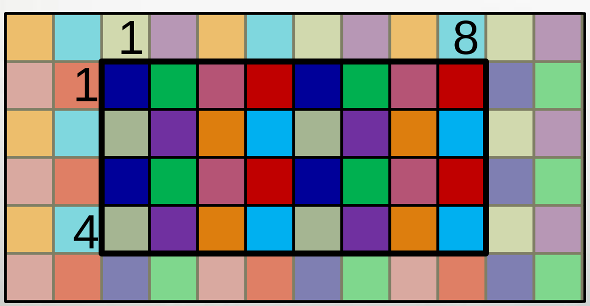
Sample Distributions: Block and Cyclic

```
var Dom: domain(2) dmapped Block(boundingBox=[1..4, 1..8])
       = [1..4, 1..8];
```



distributed to


```
var Dom: domain(2) dmapped Cyclic(startIdx=(1,1))
       = [1..4, 1..8];
```



distributed to

Chapel's Domain Map Strategy

1. Chapel provides a library of standard domain maps
   - to support common array implementations effortlessly
2. Advanced users can write their own domain maps in Chapel
   - to cope with shortcomings in our standard library
3. Chapel's standard layouts and distributions will be written using the same user-defined domain map framework
   - to avoid a performance cliff between "built-in" and user-defined domain maps
4. Domain maps should only affect implementation and performance, not semantics
   - to support switching between domain maps effortlessly

Using Domain Maps

- Syntax

```
dmap-type:
  dmap(dmap-class(…))
dmap-value:
  new dmap(new dmap-class(…))
```

- Semantics
  - Domain maps specify how a domain and its arrays are implemented
- Examples

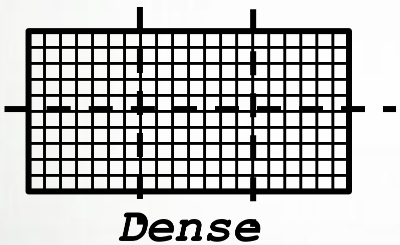
```
use myDMapMod;
var DMap: dmap(myDMap(…)) = new dmap(new myDMap(…));

var Dom: domain(…) dmapped DMap;
var A: [Dom] real;
```

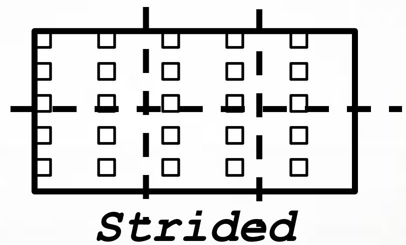
Domain Map Types

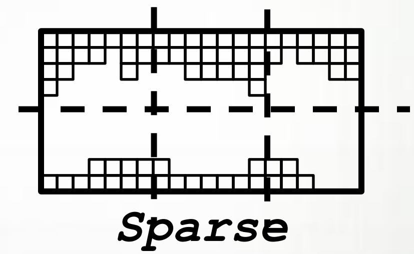
All domain types can be dmapped.

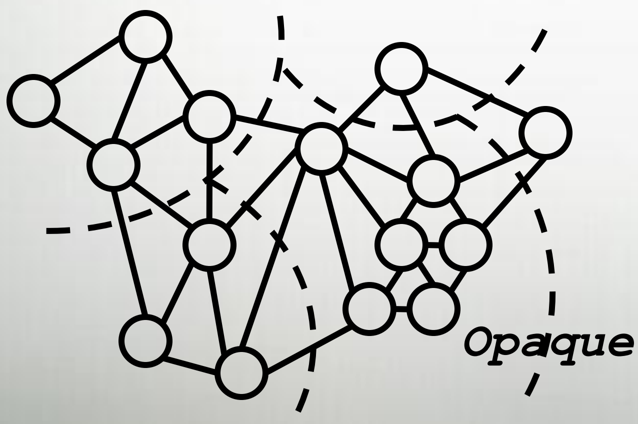
Semantics are independent of domain map.

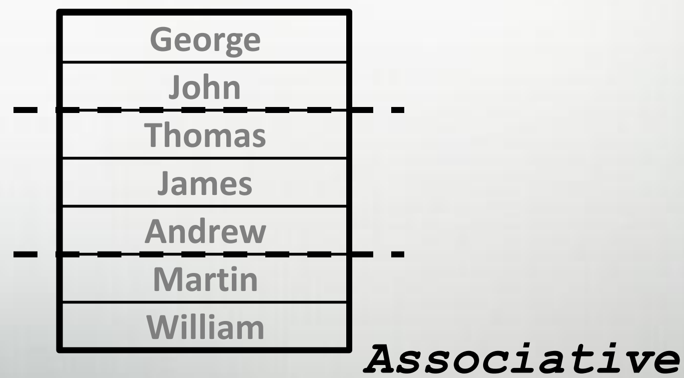
(Though performance and parallelism will vary...)









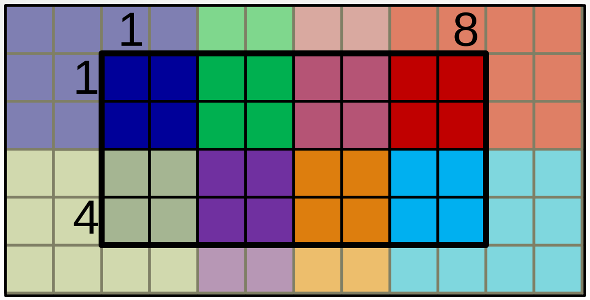


Outline

- Data Parallelism Revisited
- Domain Maps
- Chapel Standard Layouts and Distributions
  - Block
  - Cyclic
- User-defined Domain Maps

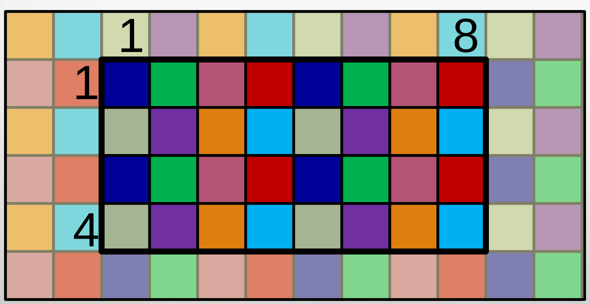
Sample Distributions: Block and Cyclic

```
var Dom: domain(2) dmapped Block(boundingBox=[1..4, 1..8])
        = [1..4, 1..8];
```



distributed to

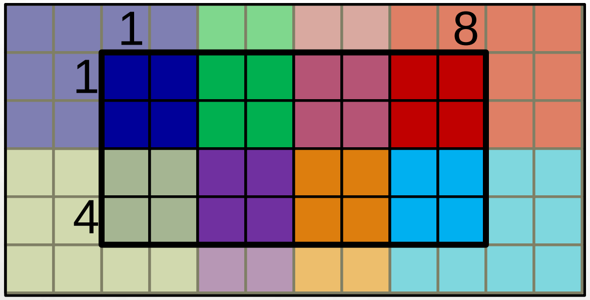

```
var Dom: domain(2) dmapped Cyclic(startIdx=(1,1))
        = [1..4, 1..8];
```



distributed to

The Block class constructor

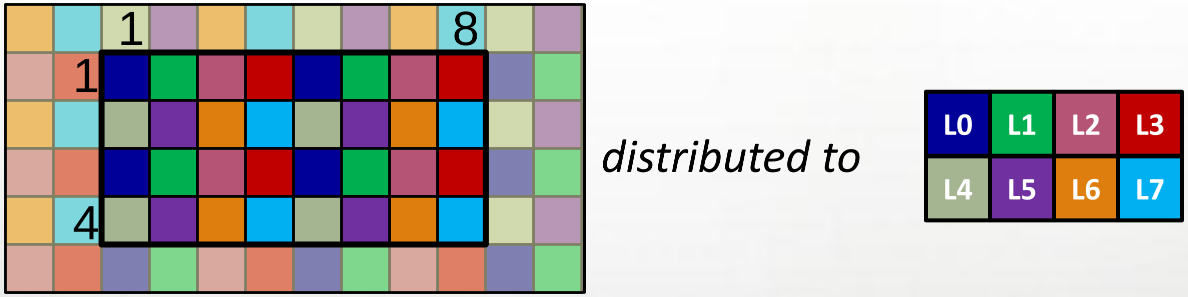
```
def Block(boundingBox: domain,
          targetLocales: [] locale = Locales,
          dataParTasksPerLocale = ...,
          dataParIgnoreRunningTasks = ...,
          dataParMinGranularity = ...,
          param rank = boundingBox.rank,
          type idxType = boundingBox.dim(1).eltType)
```



distributed to

The Cyclic class constructor

```
def Cyclic(startIdx,
           targetLocales: [] locale = Locales,
           dataParTasksPerLocale = ...,
           dataParIgnoreRunningTasks = ...,
           dataParMinGranularity = ...,
           param rank: int = infered from startIdx,
           type idxType = infered from startIdx)
```



Outline

- Data Parallelism Revisited
- Domain Maps
- Chapel Standard Layouts and Distributions
- User-defined Domain Map Descriptors

User-Defined Distribution Descriptors

	Domain Map	Domain	Array
Global one instance per object (logically)	**Role:** Similar to layout's domain map descriptor	**Role:** Similar to layout's domain descriptor, but no Θ(#indices) storage **Size:** Θ(1)	**Role:** Similar to layout's array descriptor, but data is moved to local descriptors **Size:** Θ(1)
Local one instance per node per object (typically)	**Role:** Stores node-specific domain map parameters	**Role:** Stores node's subset of domain's index set **Size:** Θ(1) → Θ(*#indices* / *#nodes*)	**Role:** Stores node's subset of array's elements **Size:** Θ(*#indices* / *#nodes*)

Sample Block Distribution Descriptors

	Domain Map	Domain	Array
Global one instance per object (logically)	boundingBox = [1..4, 1..8] targetLocales = L0 L1 L2 L3 / L4 L5 L6 L7	indexSet = [1..4, 1..8]	--
Local one instance per node per object (typically)	L4 myIndexSpace = [3..max, min..2]	L4 myIndices = [3..4, 1..2]	L4 myElems =

```
var Dom: domain(2) dmapped Block(boundingBox=[1..4, 1..8])
      = [1..4, 1..8];
```

Domain Maps: Status

- Full-featured Block- and Cyclic distributions
- Serial COO and CSR Sparse layouts supported
- Serial quadratic probing Associative layout supported
- Block-Cyclic, Associative distributions underway
- Parallel irregular layouts and distributions underway
- Need to finalize user-defined domain map interfaces

Future Directions

- More standard distributions and layouts
- Specify interface for user-defined domain maps
- Advanced uses of domain maps:
  - GPU programming
  - Dynamic load balancing
  - Resilient computation
  - *in situ* interoperability
  - Out-of-core computations

Questions?

- Data Parallelism Revisited
- Domain maps
  - Layouts
  - Distributions
- The Chapel Standard Distributions
  - Block Distribution
  - Cyclic Distribution
- User-defined Domain Maps