Programming Languages

- High-Performance Computing as Paradigm

- Lots of design choices in Chapel to discuss:
  - Task Creation (instead of Threads) with 'begin'.
  - Task Synchronicity with 'sync' and cobegin
  - Parallel loops: forall and coforall
  - Thread safety using variable 'sync’
  - reduce overcomes bottleneck

PL: Task Generation

```
var total = 0;
for i in 1..100 do total += i;

writeln(''Sum is '', total, ''.'');
```

We can add a Timer to measure running time!

PL: Task Generation

```
var total = 0;
for i in 1..100 do total += i;

writeln(''Sum is '', total, ''.'');
```

We can add a Timer to measure running time!

```
use Time;
var timer: Timer;
var total = 0;
timer.start();
for i in 1..100 do total += i;
timer.stop();

writeln(''Sum is '', total, ''.'');
writeln(''That took '', timer.elapsed(), '' seconds.'');
```

PL: Task Generation

Now let's use another thread!

```
use Time;
var timer: Timer;
var total = 0;
var highTotal = 0;
var lowTotal = 0;
timer.start();
begin ref(highTotal) {
    for i in 51..100 do highTotal += i;
}
for i in 1..50 do lowTotal += i;
total = lowTotal + highTotal;
timer.stop();

writeln(''Sum is '', total, ''.'');
writeln(''That took '', timer.elapsed(), '' seconds.'');
```

Note: ref(highTotal) at begin

PL: Task Generation

Now let's use another thread!

```
use Time;
var timer: Timer;
var total = 0;
var highTotal = 0;
var lowTotal = 0;
timer.start();
begin ref(highTotal) {
    for i in 51..100 do highTotal += i;
}
for i in 1..50 do lowTotal += i;
total = lowTotal + highTotal;
timer.stop();

writeln(''Sum is '', total, ''.'');
writeln(''That took '', timer.elapsed(), '' seconds.'');
```

Result: faster, but sometimes incorrect.

PL: Synchronization

Incorrect: top thread may not finish.

Chapel provides a solution: sync

```
sync {
   begin {
      …
   }
   begin {
      …
   }
   …
}
```

PL: Synchronization

Use sync:

```
...
timer.start();
sync {
   begin ref(highTotal) {
      for i in 51..100 do highTotal += i;
   }
   begin ref(lowTotal) {
      for i in 1..50 do lowTotal += i;
   }
}
total = lowTotal + highTotal;
...
```

PL: Syntactic Sugar

Ask students: How common is this?

```
sync {
   begin {
      //single line of code
   }
   begin {
      //another single line
   }
   . . .
   begin {
      //even yet another single line
   }
}
```

So, what did language designers do?

PL: Syntactic Sugar

```
cobegin {
   //single line of code
   //another single line
   . . .
   //even yet another single line
}
```

PL: forall

forall: data-parallel loop

```
var sum = 0;
forall i in 1..100 {
  sum += i;
}
writeln("Sum is: ", sum, ".");
```

PL: forall

forall: data-parallel loop

```
var sum = 0;
forall i in 1..100 {
   sum += i;
}
writeln("Sum is: ", sum, ".");
```

Ask: Why doesn't this work?

PL: HPC Concepts

- Why doesn't it work?
  - Race conditions
  - Atomicity
  - Synchronization solutions

PL: forall

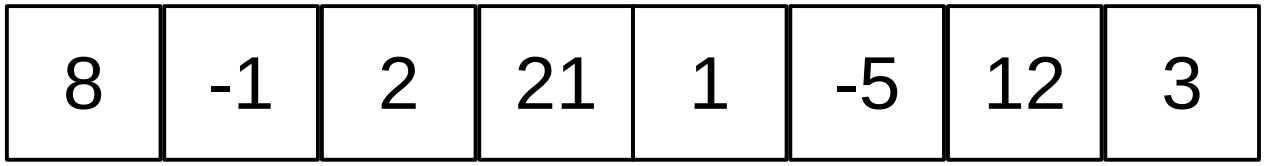
One solution: synchronized variables

```
var sum : sync int;
sum = 0;
forall i in 1..100 {
   sum += i;
}
writeln("Sum is: ", sum, ".");
```

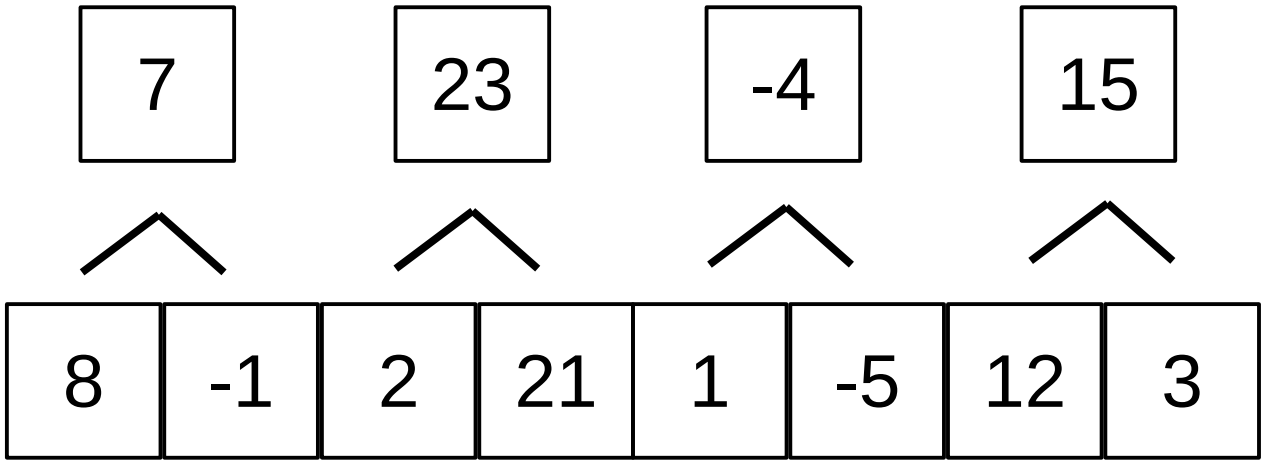
PL: sync bottleneck and reduce

- sync causes a bottleneck:
  - Running time still technically linear.
- Reductions:
  - Divide-and-conquer solution

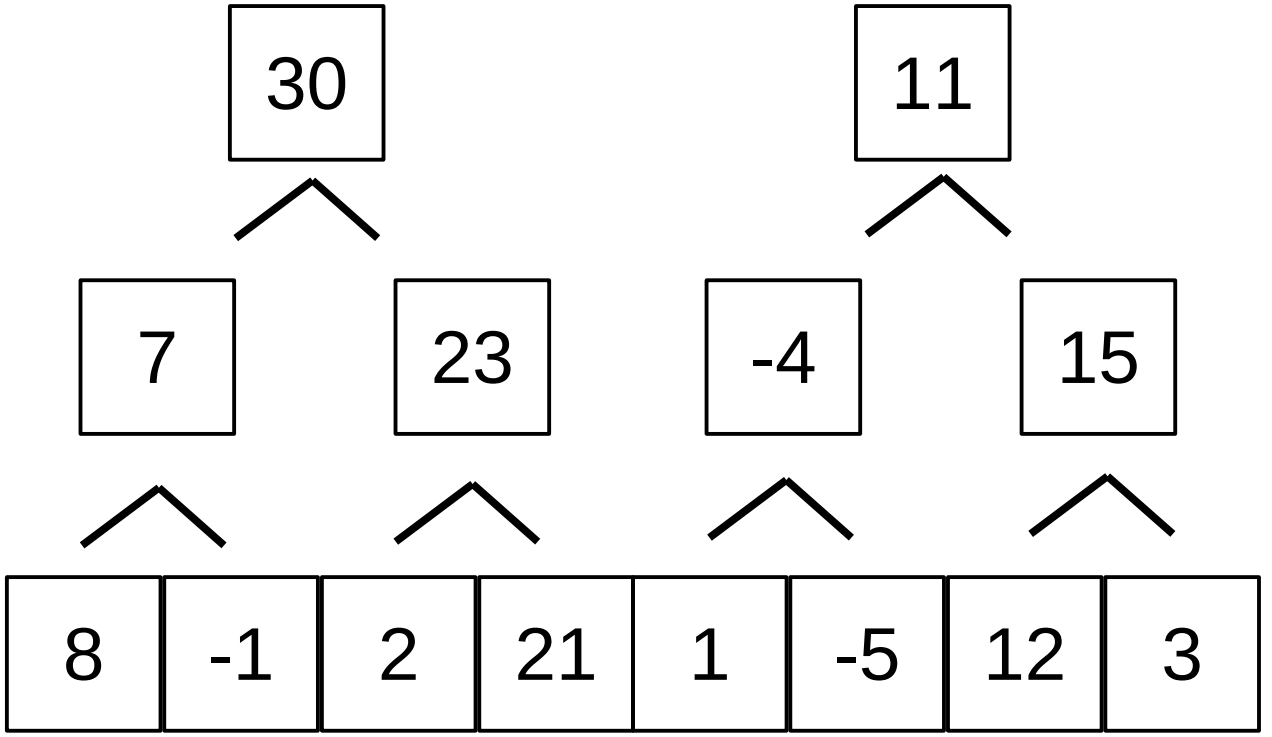
PL: Reductions



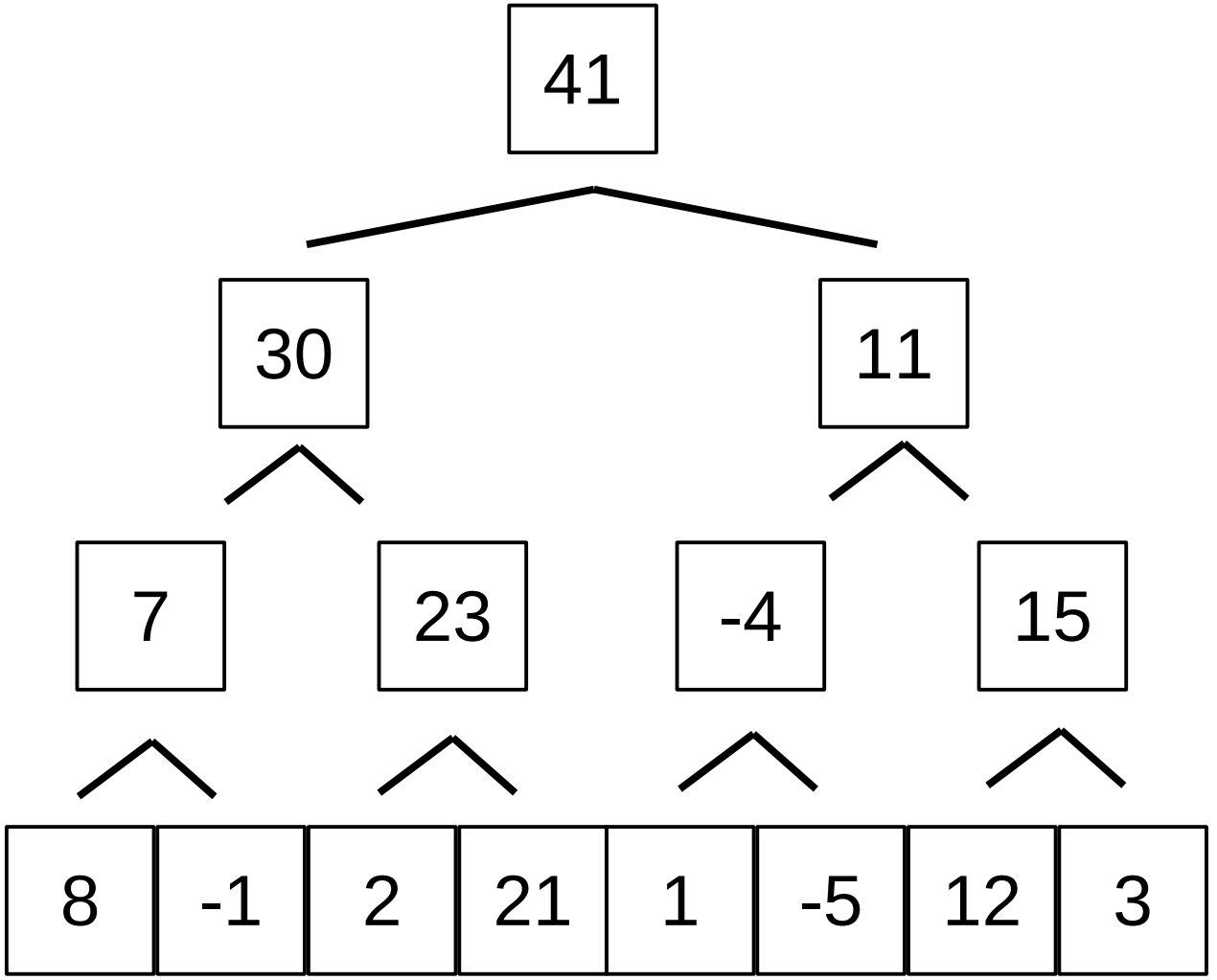
PL: Reductions



PL: Reductions



PL: Reductions



PL: sync bottleneck and reduce

- sync causes a bottleneck:
  - Running time still technically linear.
- Reductions:
  - Divide-and-conquer solution

PL: sync bottleneck and reduce

- sync causes a bottleneck:
  - Running time still technically linear.
- Reductions:
  - Divide-and-conquer solution
  - Simplify with 'reduce' keyword!

PL: sync bottleneck and reduce

```
var integers : [1..100] int;
forall i in integers.domain {
    integers[i] = i;
}
var sum = + reduce integers;
```

PL: sync bottleneck and reduce

```
var integers : [1..100] int;
forall i in integers.domain {
    integers[i] = i;
}
var sum = + reduce integers;
```

One line solution?

PL: sync bottleneck and reduce

```
var sum = + reduce (1..100);
```

PL: sync bottleneck and reduce

All intermediate values?

```
var sum = + scan array;
```

8	-1	2	21	1	-5	12	3

PL: sync bottleneck and reduce

All intermediate values?

```
var sum = + scan array;
```

8	-1	2	21	1	-5	12	3

8	7	9	30	31	26	38	41

PL: sync bottleneck and reduce

Fun Uses!

```
var factorials = * scan (1..10);

var threes : [1..10] int;
forall i in threes.domain do
    threes[i] = 3;
var powersOfThree = * scan threes;
```

Chapel Ranges

- What is a range?
- How are ranges used?
- Range operations

Chapel Ranges

- What is a range?
  - A range of values
  - Ex: var someNaturals : range = 0..50;
- How are they used?
    - Indexes for Arrays
    - Iteration space in loops
- Are there cool operations?

Chapel Ranges

- What is a range?
  - A range of values
  - Ex: var someNaturals : range = 0..50;
- How are they used?
    - Indexes for Arrays
    - Iteration space in loops
- Are there cool operations?

  Yes!

Range Operation Examples

```
var someNaturals: range = 0..50;
var someEvens = someNaturals by 2;
                    (someEvens: 0, 2, 4, ..., 48, 50)
var someOdds = someEvens align 1;
                    (someOdds: 1, 3, 5, 7, ..., 47, 49)
var fewerOdds = someOdds # 6;
                    (fewerOdds: 1, 3, 5, 7, 9, 11)
```

Other Cool Range Things

- Can create “infinite” ranges:

```
var naturals: range = 0..;
```

- Ranges in the “wrong order” are auto-empty:

```
var nothing: range = 2..-2;
```

- Otherwise, negatives are just fine

Chapel Domains

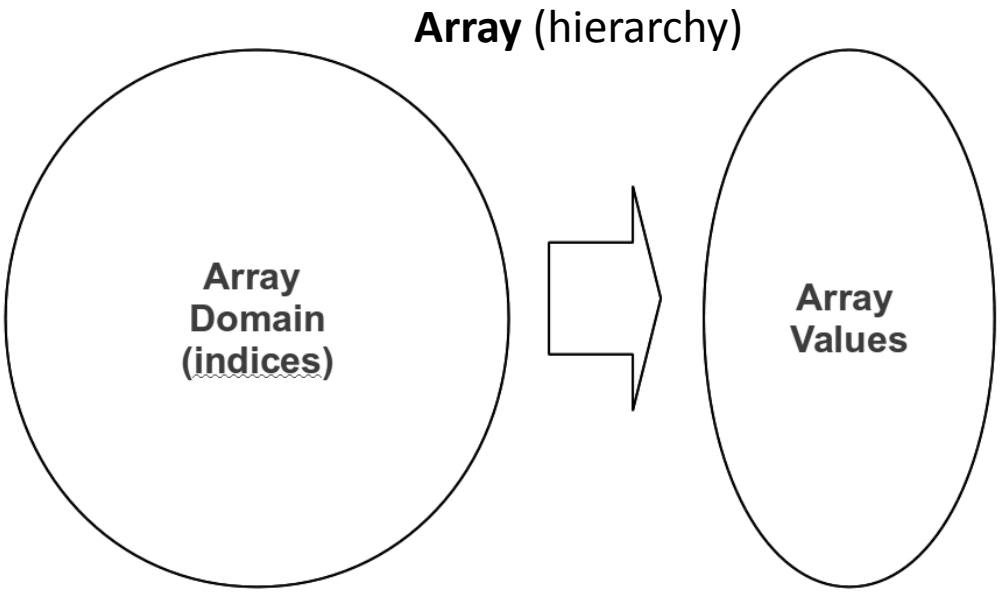
- What is a domain?
- How are domains used?
- Operations on domains
- Example: Game of Life

Chapel Domains

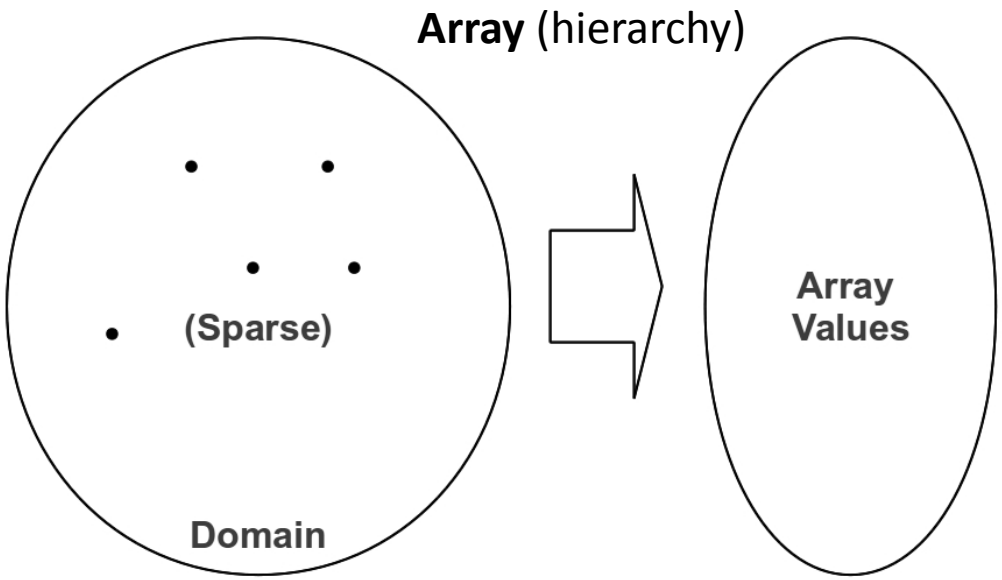
- Domain: index set
  - Used to simplify addressing
  - Every array has a domain to hold its indices
  - Can include ranges or be sparse
- Example:

```
var A: [1..10] int; //indices are 1, 2, ..., 10
...
for i in A.domain {
    //do something with A[i]
}
```

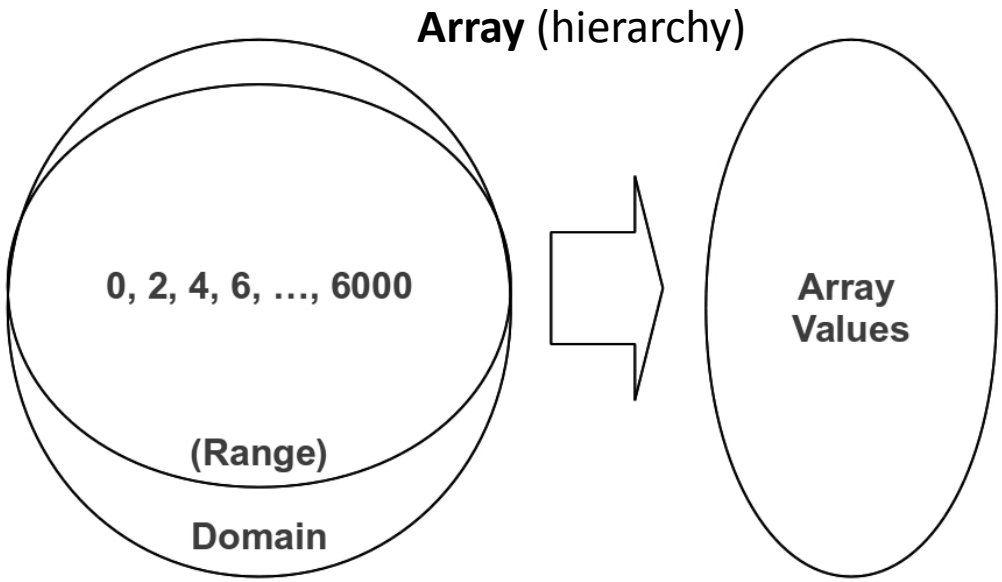
Chapel Domains



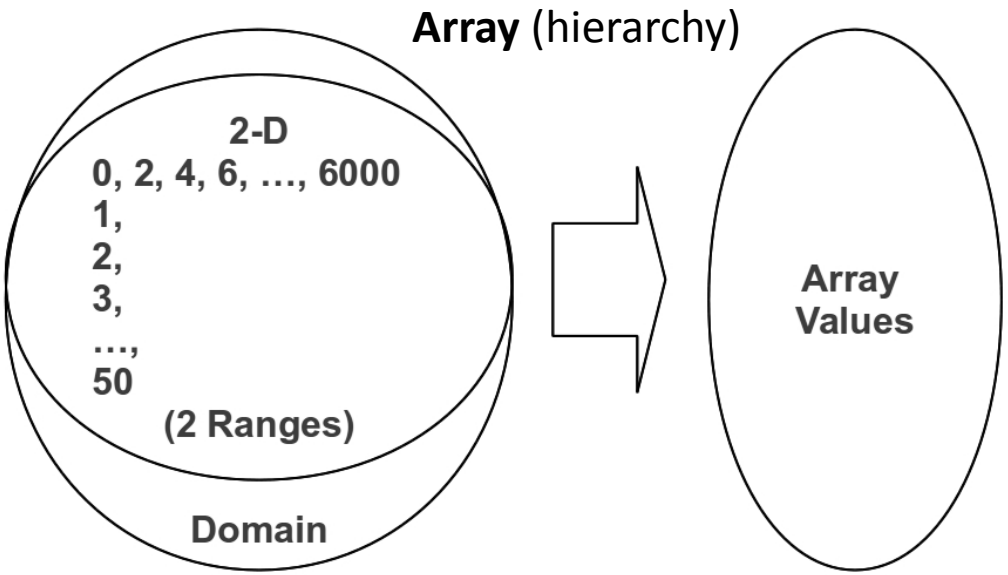
Chapel Domains



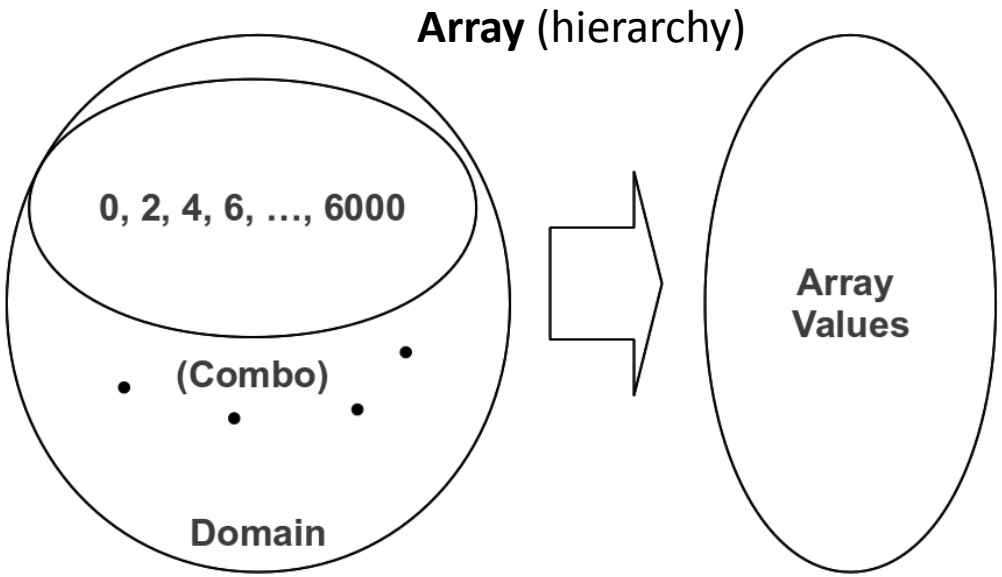
Chapel Domains



Chapel Domains



Chapel Domains



Chapel Domains

- Domain Declaration:
  - var D: domain(2) = {0..m, 0..n};
    - D is 2-D domain with (m+1) x (n+1) entries
  - var A: [D] int;
    - A is an array of integers with D as its domain

Chapel Domains

- Domain Declaration:
  - var D: domain(2) = {0..m, 0..n};
    - D is 2-D domain with (m+1) x (n+1) entries
  - var A: [D] int;
    - A is an array of integers with D as its domain

Why is this useful?

Chapel Domains

- Changing D changes A automatically!
- D = {1..m, 0..n+1}

  decrements height; increments width!

  (adds zeroes)

1	2	3
4	5	6
7	8	9

4	5	6	0
7	8	9	0

Domains vs. Ranges

- Despite how similar they seem so far, domains and ranges are different
  - Domains remain tied to arrays so that resizing the domain resizes the array:

```
var R : range = 1..10;
var A : [R] int;
R = 0..10;      //no effect on array
A[0] = 5;       //runtime error
```

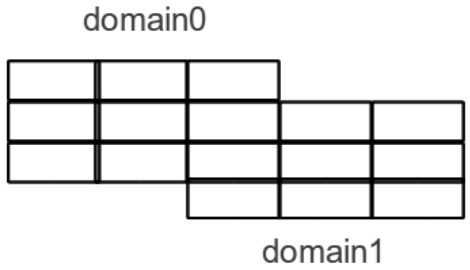
```
var D : domain(1) = {1..10};
var A : [D] int;
D = 0..10;      //resizes array
A[0] = 5;       //ok
```

- Domains are more general; some are not sets of integers

Domain Slices (Intersection)

domain0: {0..2, 1..3}

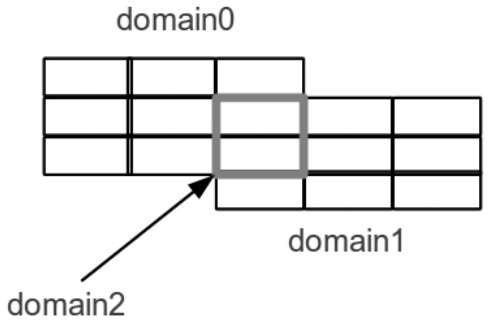
domain1: {1..3, 3..5}



Domain Slices (Intersection)

domain0: {0..2, 1..3}

domain1: {1..3, 3..5}



domain2: {1..2, 3..3}

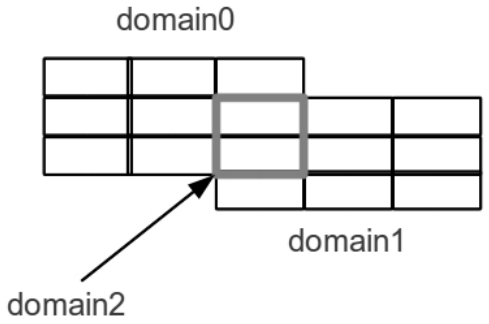
Domain Slices (Intersection)

//domain2 is the intersection of domain1 and domain0

var domain2 = domain1 [domain0];

domain0: {0..2, 1..3}

domain1: {1..3, 3..5}



domain2: {1..2, 3..3}

Domain Slices (Intersection)

```
//domain2 is the intersection of domain1 and domain0
var domain2 = domain1 [domain0];
```

PL: Projects

- Matrix Multiplication
  - Matrix-vector multiplication in class
  - Different algorithms:
    - Column-by-column
    - One entry at a time
- Collatz conjecture testing
  - Generate lots of tasks (coforall)
  - How to synchronize?

PL: Takeaways

- Lots of language features to discuss!
- Learning HPC ↔ Motivates Syntax
- Students love it!