Chapel: Overview and Features for Heterogeneity

Brad Chamberlain, Chapel Team, Cray Inc.

Heterogeneous C++ Working Group

February 5, 2018

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Copyright 2018 Cray Inc.

Plan for this morning

- Chapel context
- Overview of example core Chapel features
- Additional detail on features related to heterogeneity
- Feel free to interrupt for Questions / Discussion

Copyright 2018 Cray Inc.

Chapel and Heterogeneity in a Nutshell

- **Chapel's design supports hardware heterogeneity**
  - and in a way that is user-extensible without compiler changes
- **That said, in practice…**

  …virtually all of our work has been on homogeneous cases

  …we haven't spend much time on many of the hardest cases
    - e.g., no FPGA work, less GPU work than we'd like

Copyright 2018 Cray Inc.

What is Chapel?

Copyright 2018 Cray Inc.

What is Chapel?

Chapel: A productive parallel programming language

- portable
- open-source
- a collaborative effort

Goals:

- Support general parallel programming
  - "any parallel algorithm on any parallel hardware"
- Make parallel programming at scale far more productive

Copyright 2018 Cray Inc.

Scalable Parallel Programming Concerns

Q: What do HPC programmers need from a language?

A: ***Serial Code:*** **Software engineering and performance**

Parallelism: **What should execute simultaneously?**

Locality: **Where should those tasks execute?**

Mapping: **How to map the program to the system?**

Separation of Concerns: **Decouple these issues**

Chapel is a language designed to address these needs from first principles

Copyright 2018 Cray Inc.

Chapel and Other Languages

Chapel strives to be as…

…**programmable** as Python

…**fast** as Fortran

…**scalable** as MPI, SHMEM, or UPC

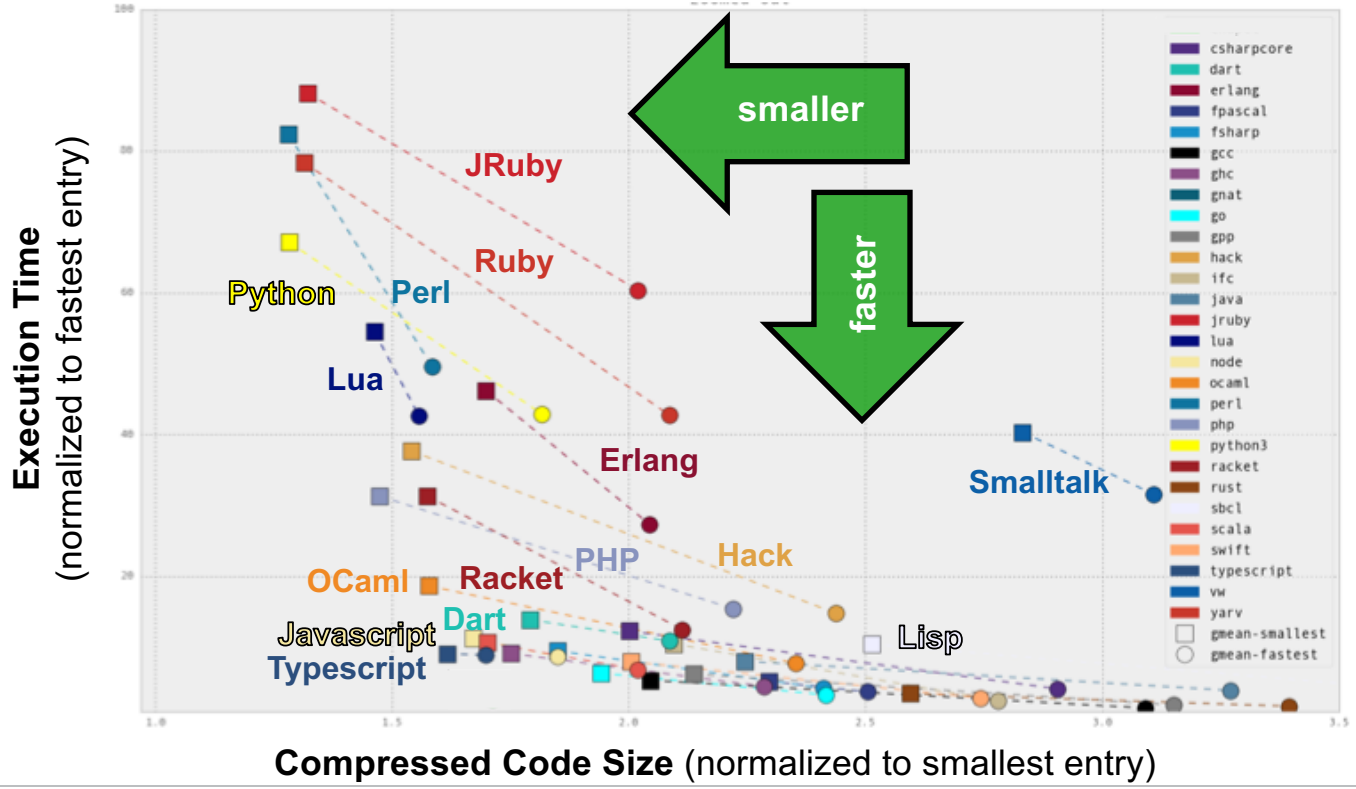
…**portable** as C

…**flexible** as C++

…**fun** as [your favorite programming language]

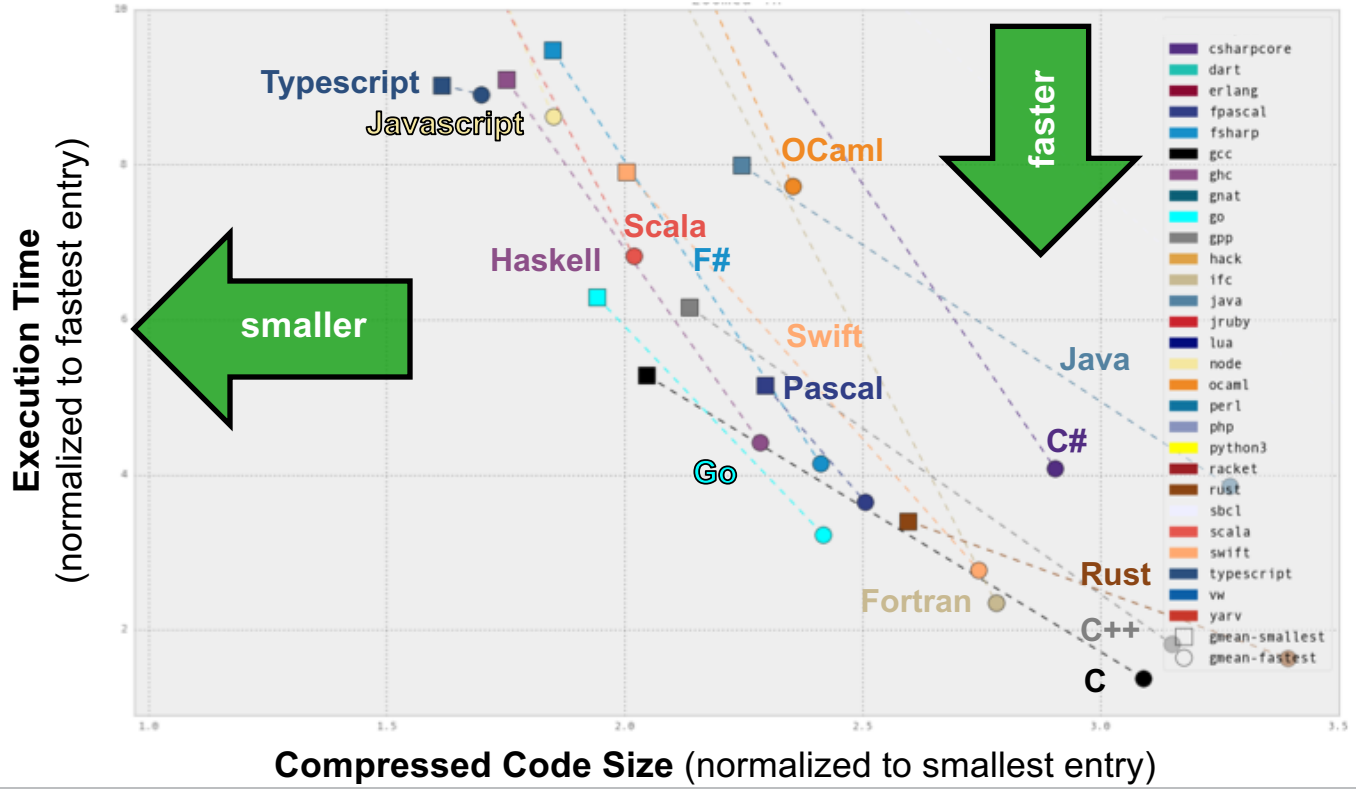
Copyright 2018 Cray Inc.

CLBG Cross-Language Summary
(Oct 2017 standings)



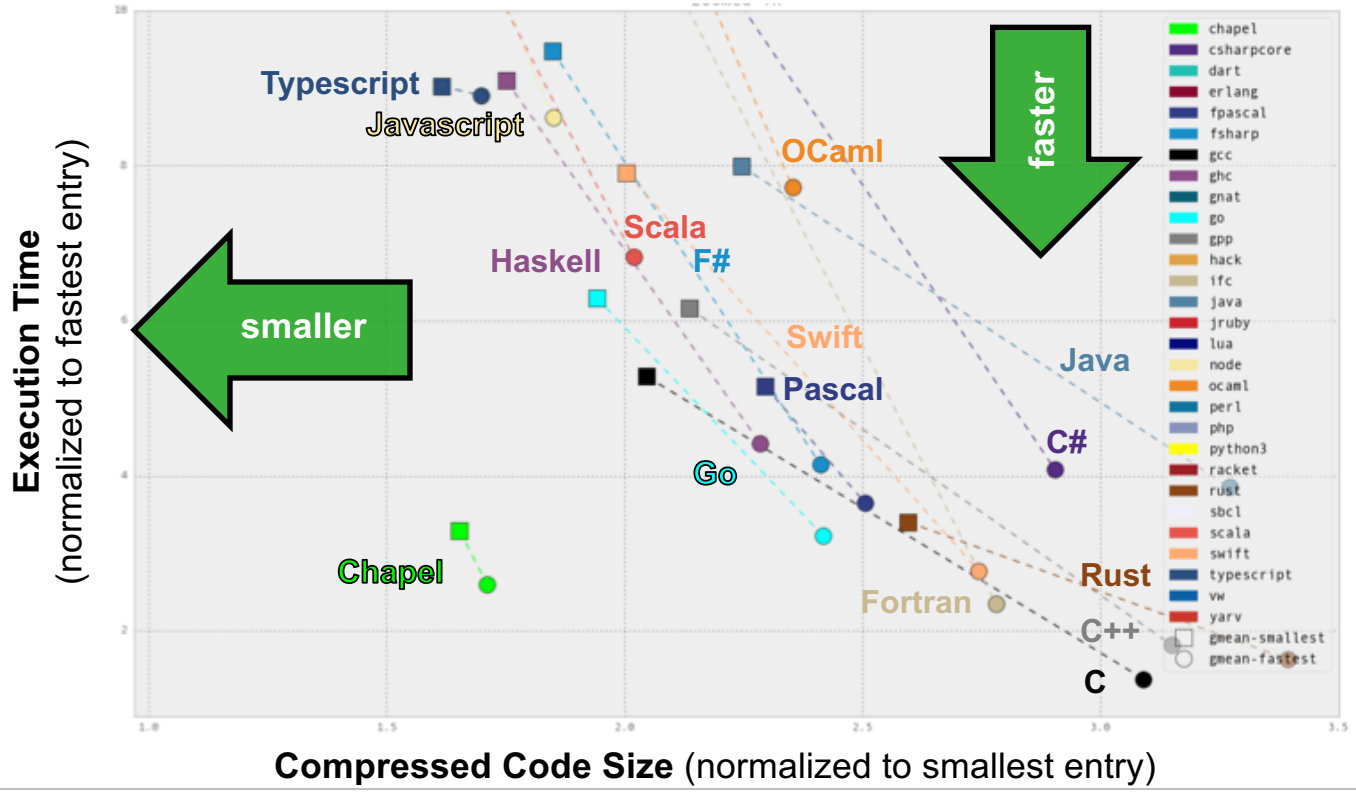
Copyright 2018 Cray Inc.

CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)



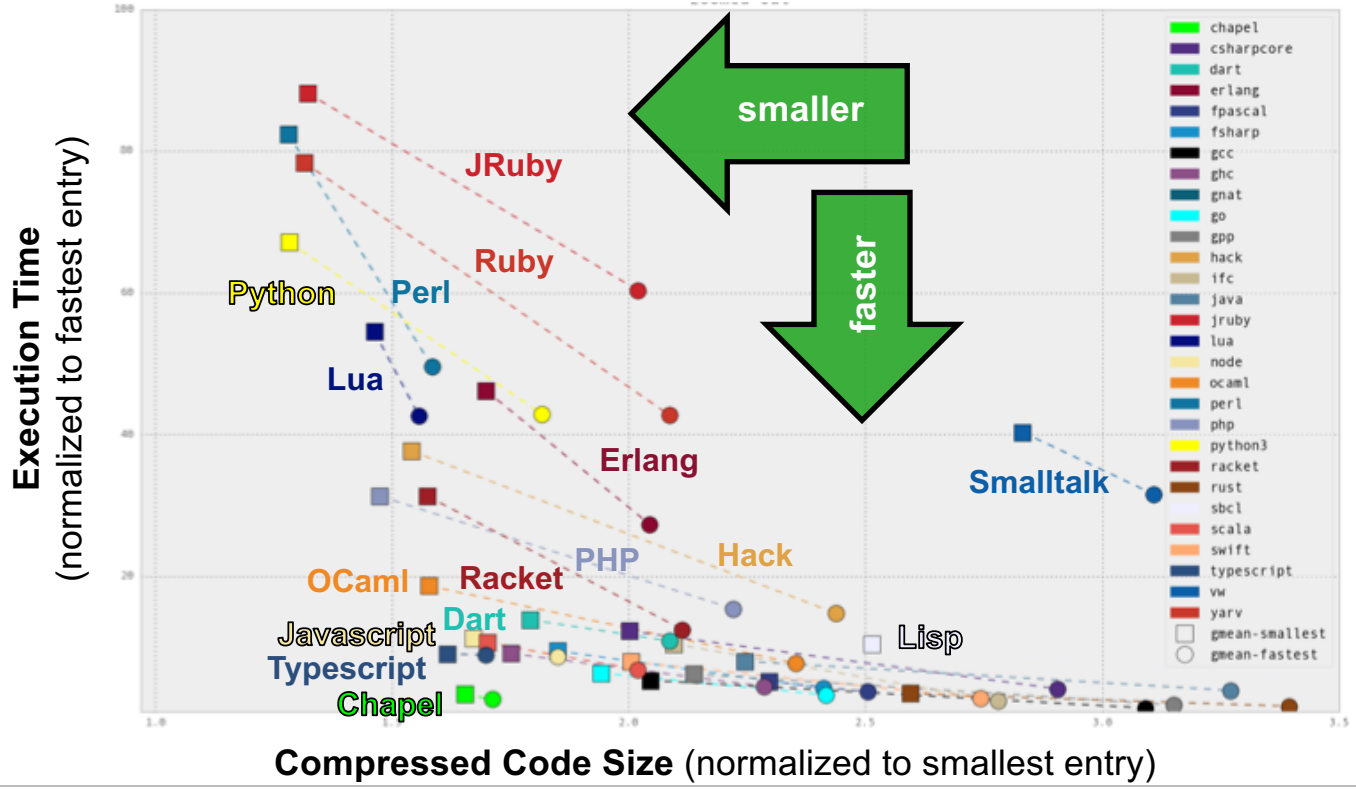
Copyright 2018 Cray Inc.

CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)



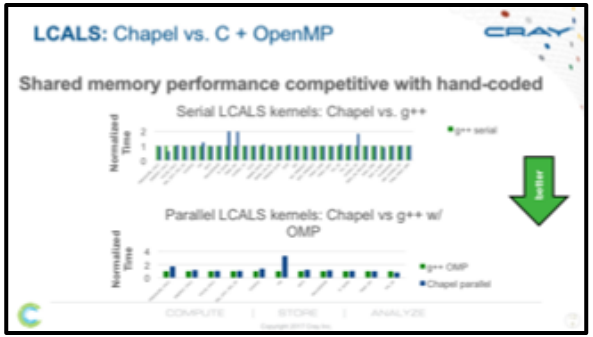
Copyright 2018 Cray Inc.

CLBG Cross-Language Summary
(Oct 2017 standings)



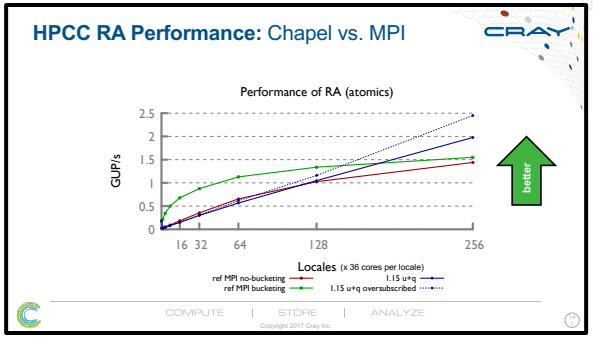
Copyright 2018 Cray Inc.

Chapel Performance: HPC Benchmarks



LCALS

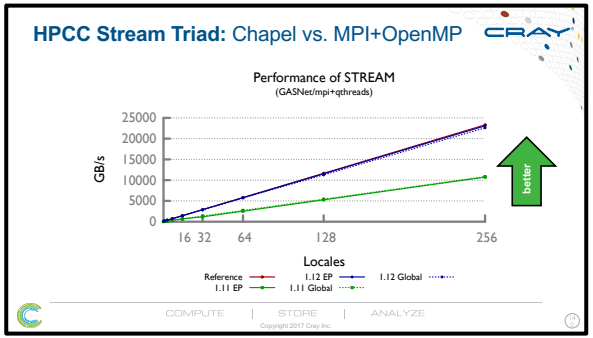
HPCC RA

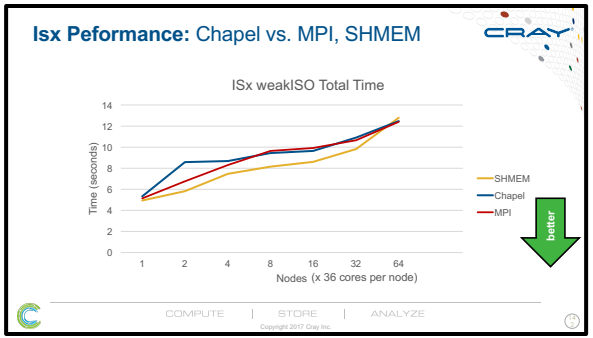


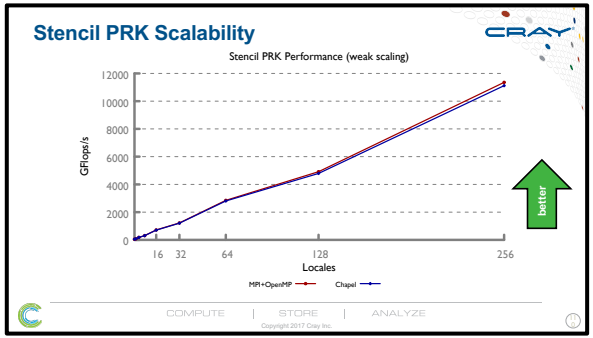
STREAM Triad

ISx

PRK Stencil







Copyright 2018 Cray Inc.

Nightly performance graphs online at: https://chapel-lang.org/perf

The Chapel Team at Cray (May 2017)

Copyright 2018 Cray Inc.

Chapel Community Partners

Lawrence Berkeley
National Laboratory

(and several others…)

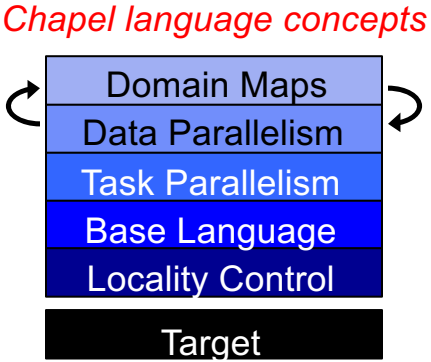
https://chapel-lang.org/collaborations.html

Copyright 2018 Cray Inc.

Introduction to Chapel, by Example

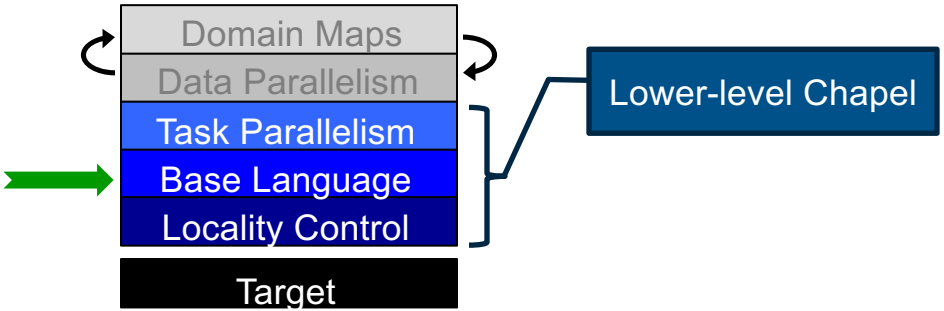
Copyright 2018 Cray Inc.

Chapel language feature areas



Copyright 2018 Cray Inc.

Base Language



Copyright 2018 Cray Inc.

Base Language Features, by example

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
...
```


Copyright 2018 Cray Inc.

Base Language Features, by example

Configuration declarations
(to avoid command-line argument parsing)
`./a.out --n=1000000`

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
...
```


Copyright 2018 Cray Inc.

Base Language Features, by example

Modern iterators

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
...
```


Copyright 2018 Cray Inc.

Base Language Features, by example

Static type inference for:
- arguments
- return types
- variables

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
...
```


Copyright 2018 Cray Inc.

Base Language Features, by example

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

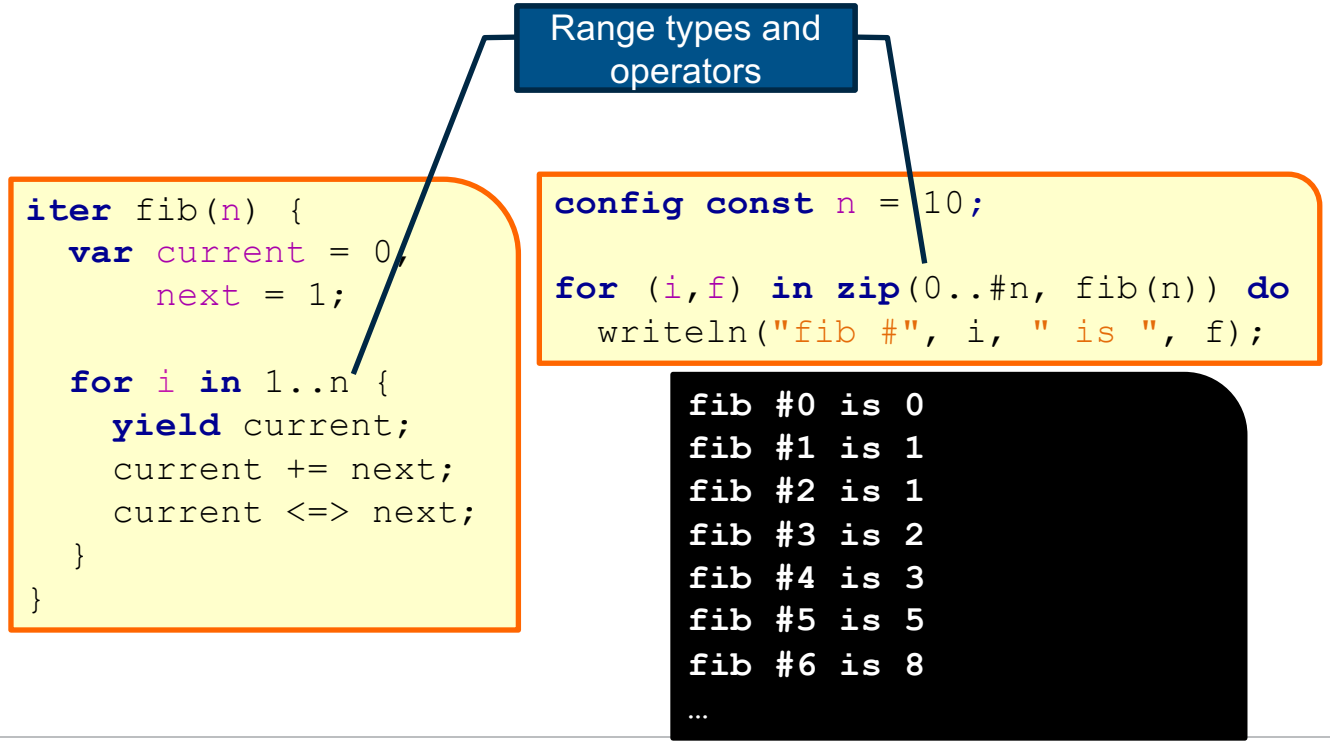
Zippered iteration

```
config const n = 10;

for (i,f) in zip(0..#n, fib(n)) do
  writeln("fib #", i, " is ", f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
…
```


Copyright 2018 Cray Inc.

Base Language Features, by example



Copyright 2018 Cray Inc.

Base Language Features, by example

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

tuples

```
config const n = 10;

for (i,f) in zip(0..#n, fib(n)) do
  writeln("fib #", i, " is ", f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
…
```


Copyright 2018 Cray Inc.

Base Language Features, by example

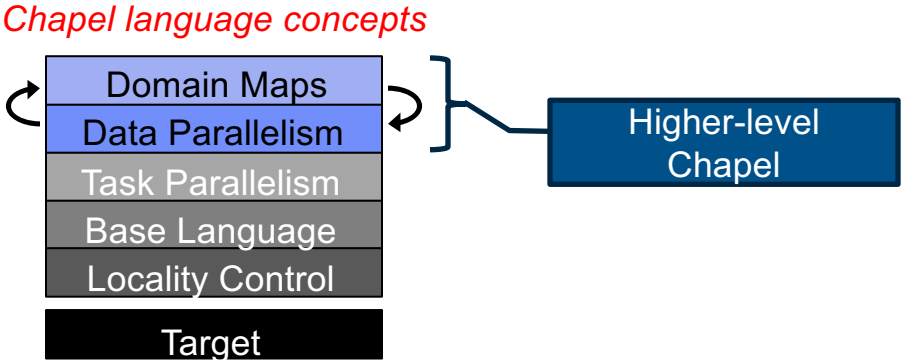
```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for (i,f) in zip(0..#n, fib(n)) do
  writeln("fib #", i, " is ", f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
…
```


Copyright 2018 Cray Inc.

Data Parallelism in Chapel



Copyright 2018 Cray Inc.

Data Parallelism, by example

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2018 Cray Inc.

Data Parallelism, by example

Domains (Index Sets)

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2018 Cray Inc.

Data Parallelism, by example

Arrays

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2018 Cray Inc.

Data Parallelism, by example

Data-Parallel Forall Loops

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2018 Cray Inc.

Data Parallelism, by example

This is a shared memory program

Nothing has referred to remote locales, explicitly or implicitly

dataParallel.chpl

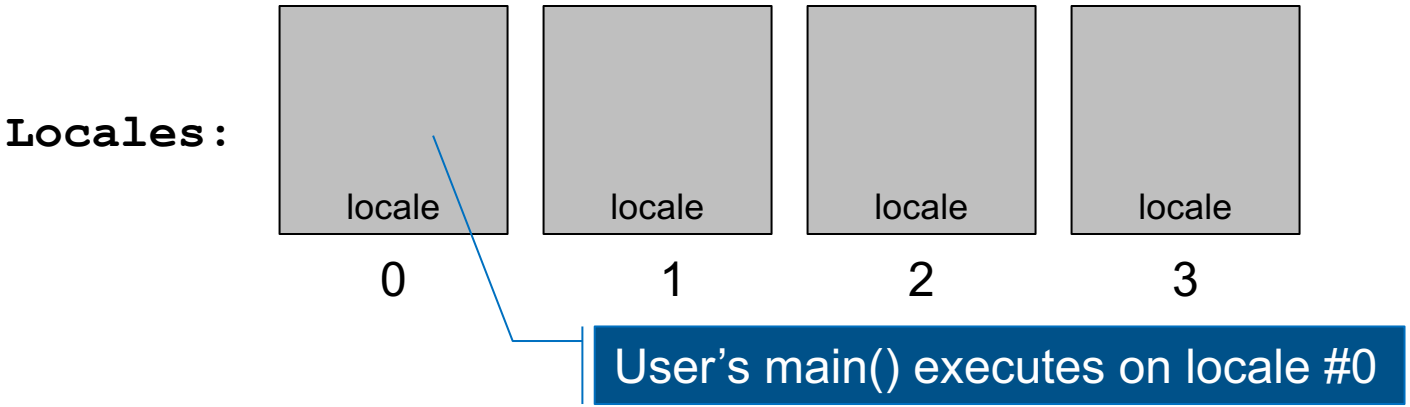
```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2018 Cray Inc.

Locales

- Unit of the target system useful for reasoning about locality
  - Each locale can run tasks and store variables
    - Has processors and memory (or can defer to something that does)
  - For most HPC systems, locale == compute node



Copyright 2018 Cray Inc.

Data Parallelism, by example

This is a shared memory program

Nothing has referred to remote locales, explicitly or implicitly

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2018 Cray Inc.

Distributed Data Parallelism, by example

Domain Maps
(Map Data Parallelism to the System)

dataParallel.chpl

```
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
        dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2018 Cray Inc.

Distributed Data Parallelism, by example

dataParallel.chpl

```
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
        dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

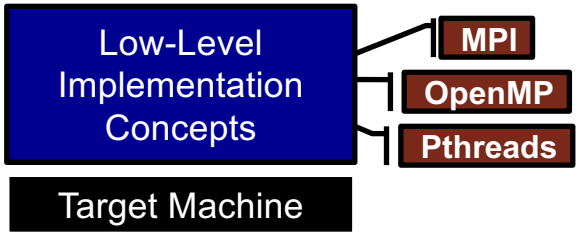
Not in the slightest…

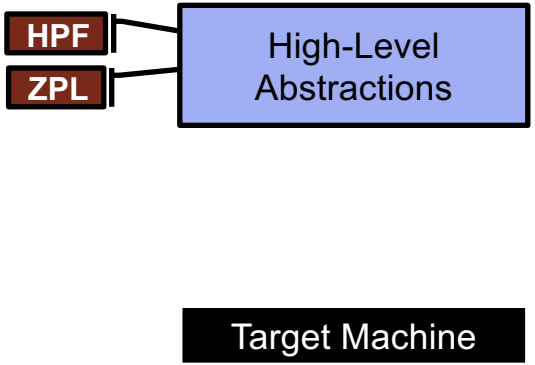
- Lowering of code is well-defined
- User can control details
- Part of Chapel's *multiresolution philosophy*…

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2018 Cray Inc.

Chapel's Multiresolution Design: Motivation





"Why is everything so tedious/difficult?"

"Why don't my programs trivially port to new systems?"

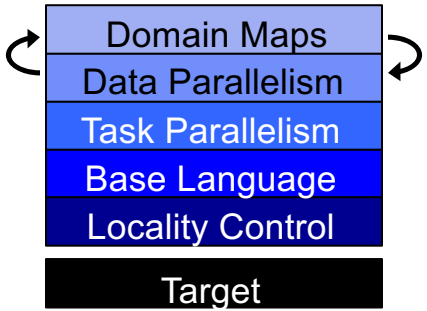
"Why don't I have more control?"

Copyright 2018 Cray Inc.

Chapel's Multiresolution Philosophy

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 users to intermix layers arbitrarily

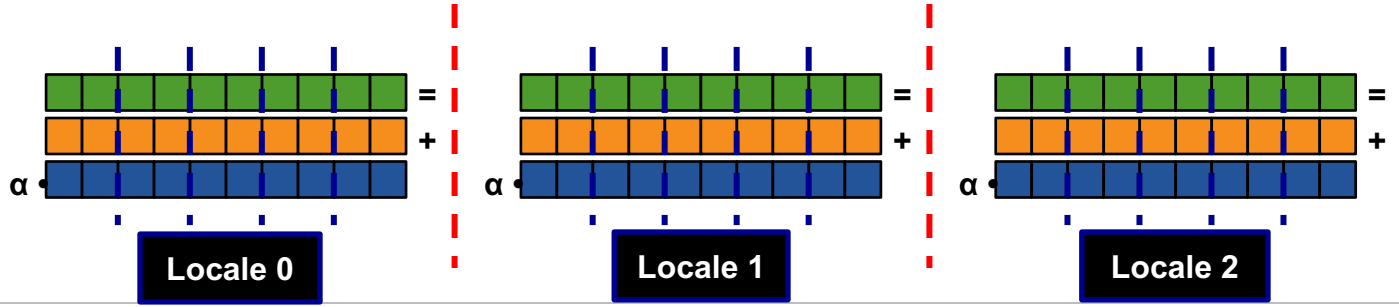
Copyright 2018 Cray Inc.

Domain Maps: A Multiresolution Feature

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


```
A = B + alpha * C;
```

…to the target locales’ memory and processors:



Copyright 2018 Cray Inc.

Authoring Domain Maps

- **Users can write their own domain maps**
  - Implemented within Chapel itself
  - Create an object type per concept:
    - The domain map itself
    - A domain
    - An array
  - Make them satisfy a standard interface
    - e.g., arrays must support iteration, random access, etc.
  - Compiler targets this interface in implementing the language
  - Goal: make the language flexible, future-proof

Note: *all* Chapel arrays are implemented this way

Copyright 2018 Cray Inc.

Distributed Data Parallelism, by example

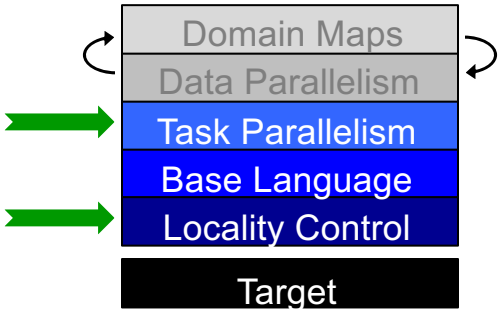
dataParallel.chpl

```
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
        dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2018 Cray Inc.

Task Parallelism and Locality Control



Copyright 2018 Cray Inc.

Task Parallelism and Locality, by example

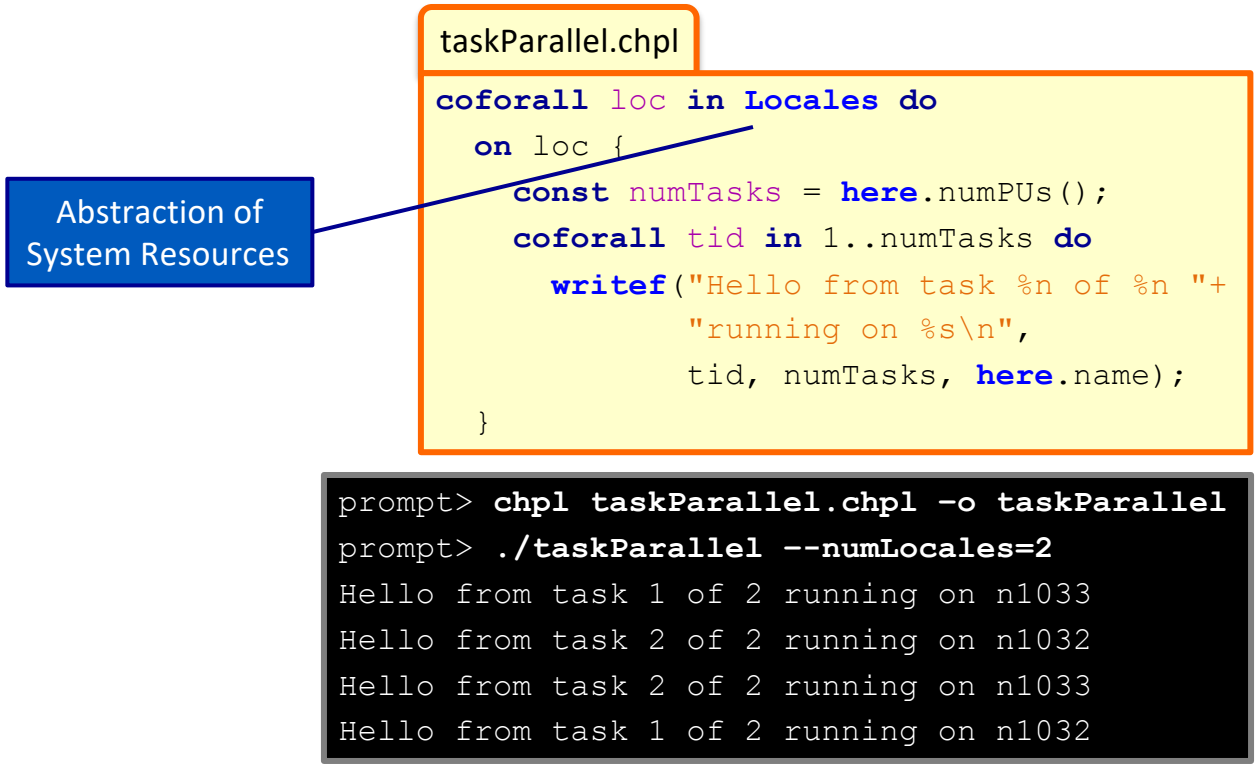
taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```


Copyright 2018 Cray Inc.

Task Parallelism and Locality, by example



Copyright 2018 Cray Inc.

Task Parallelism and Locality, by example

High-Level Task Parallelism

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```


Copyright 2018 Cray Inc.

Task Parallelism and Locality, by example

taskParallel.chpl

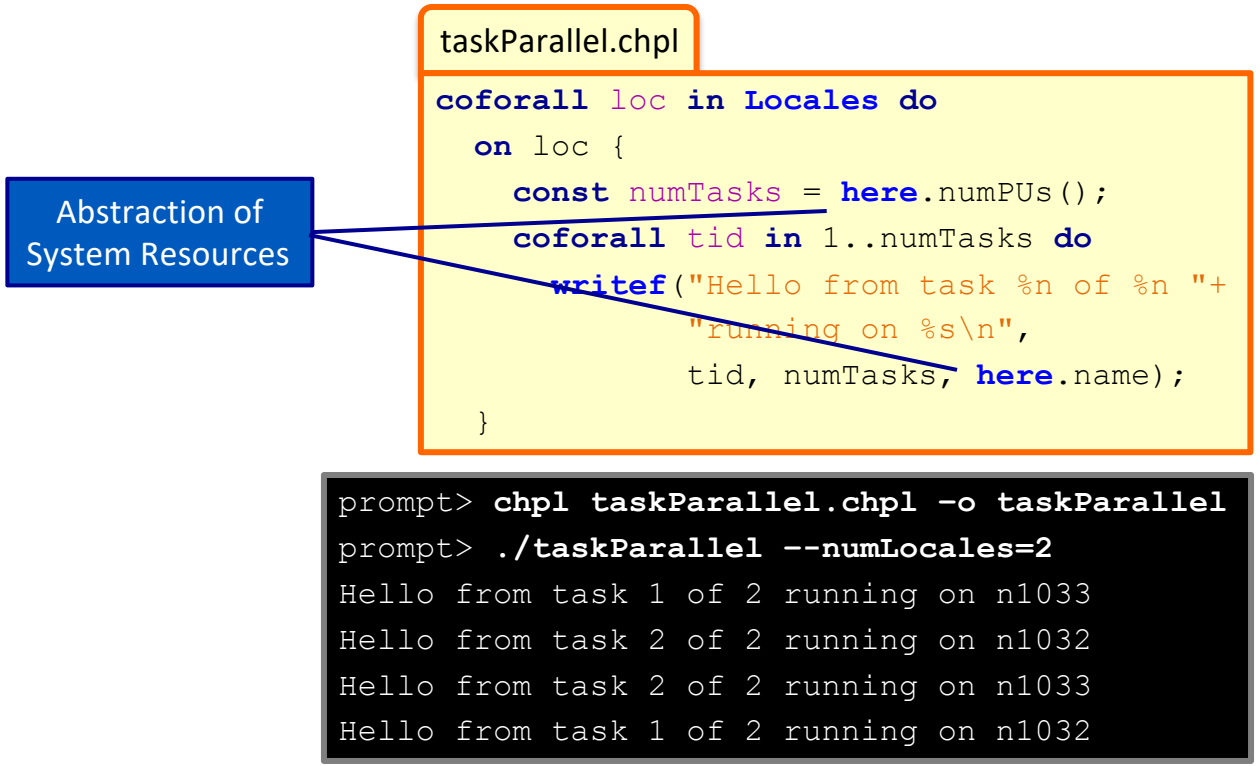
```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

Control of Locality/Affinity

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```


Copyright 2018 Cray Inc.

Task Parallelism and Locality, by example



Copyright 2018 Cray Inc.

Task Parallelism and Locality, by example

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```


Copyright 2018 Cray Inc.

Task Parallelism and Locality, by example

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

Not seen here:

Data-centric task coordination via atomic and full/empty vars

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```

Copyright 2018 Cray Inc.

Task Parallelism and Locality, by example

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```


Copyright 2018 Cray Inc.

Parallelism and Locality: Distinct in Chapel

- This is a **parallel**, but local program:

```
coforall i in 1..msgs do
  writeln("Hello from task ", i);
```

- This is a **distributed**, but serial program:

```
writeln("Hello from locale 0!");
on Locales[1] do writeln("Hello from locale 1!");
on Locales[2] do writeln("Hello from locale 2!");
```

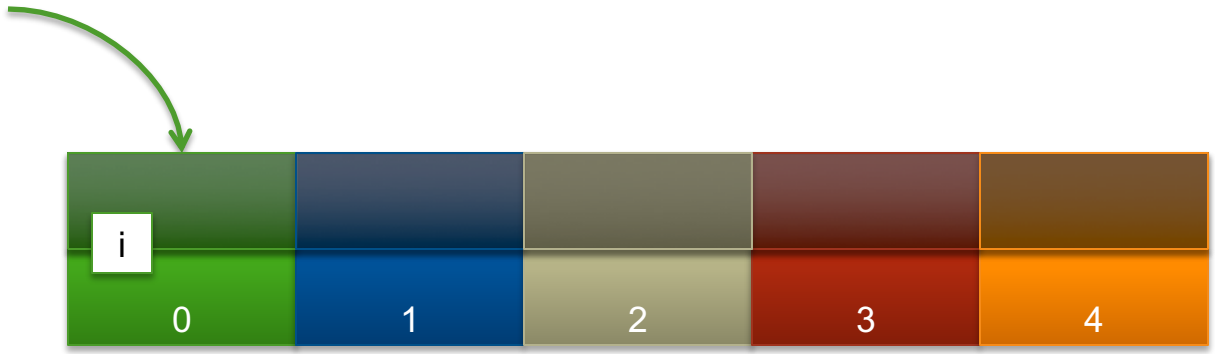
- This is a **distributed parallel** program:

```
coforall i in 1..msgs do
  on Locales[i%numLocales] do
    writeln("Hello from task ", i,
            " running on locale ", here.id);
```


Copyright 2018 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
```

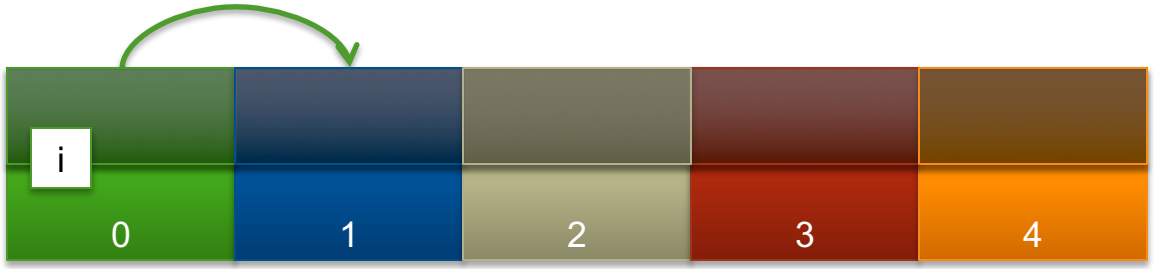


Locales (think: “compute nodes”)

Copyright 2018 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
```

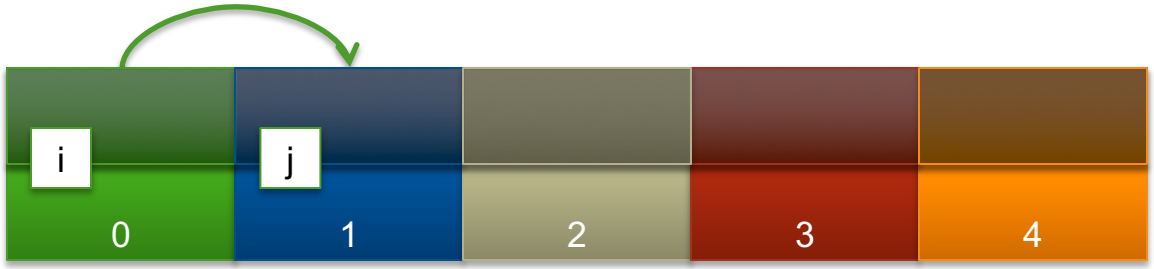


Locales (think: “compute nodes”)

Copyright 2018 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
  var j: int;
```

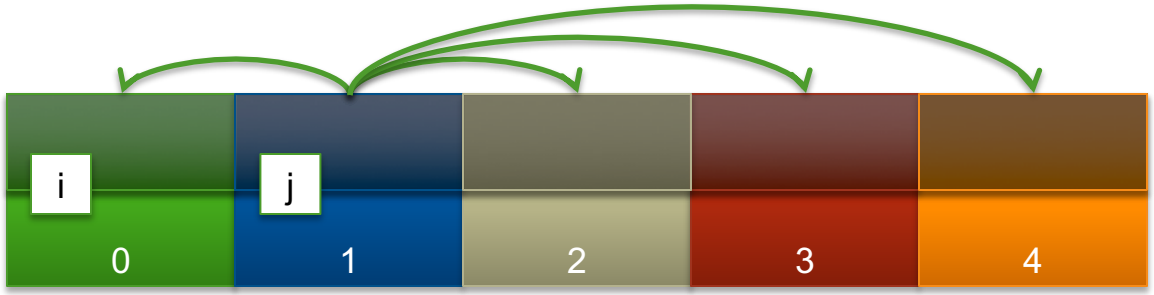


Locales (think: “compute nodes”)

Copyright 2018 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
  var j: int;
  coforall loc in Locales {
    on loc {
```

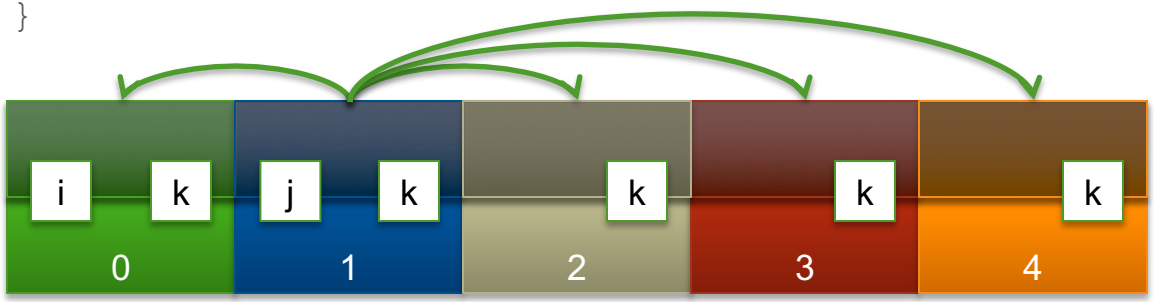


Locales (think: “compute nodes”)

Copyright 2018 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
  var j: int;
  coforall loc in Locales {
    on loc {
      var k: int;
      …
    }
  }
}
```



Locales (think: “compute nodes”)

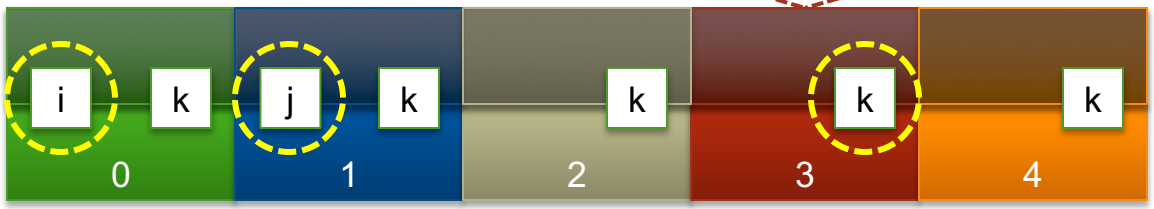
Copyright 2018 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
  var j: int;
  coforall loc in Locales {
    on loc {
      var k: int;
      k = 2*i + j;
    }
  }
}
```

OK to access *i*, *j*, and *k* wherever they live

`k = 2*i + j;`



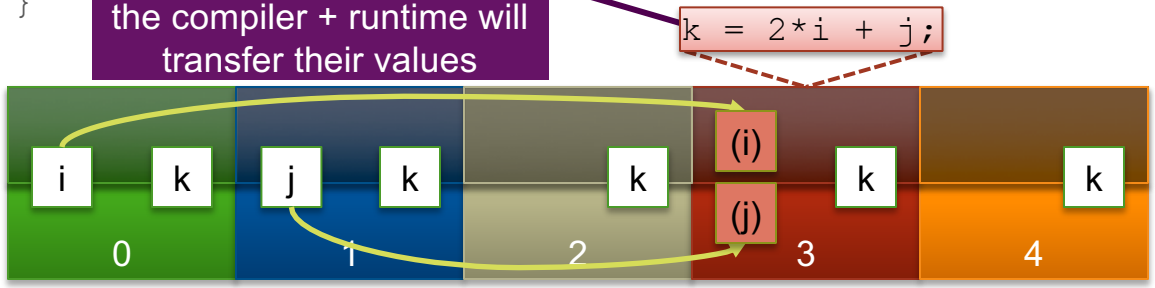
Locales (think: "compute nodes")

Copyright 2018 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
  var j: int;
  coforall loc in Locales {
    on loc {
      var k: int;
      k = 2*i + j;
    }
  }
}
```

here, *i* and *j* are remote, so the compiler + runtime will transfer their values

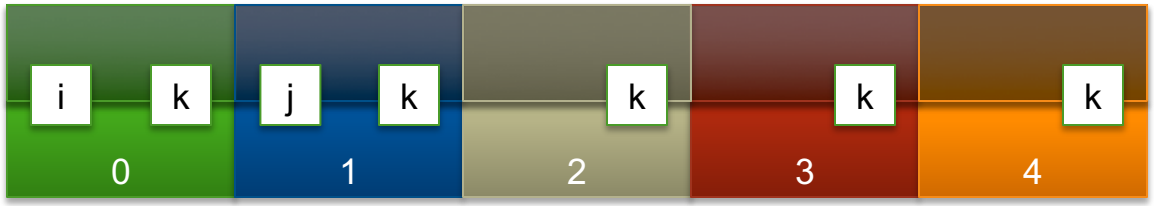


Locales (think: "compute nodes")

Copyright 2018 Cray Inc.

Chapel: Locality queries

```
var i: int;
on Locales[1] {
  var j: int;
  coforall loc in Locales {
    on loc {
      …here…          // query the locale on which this task is running
      …j.locale…      // query the locale on which j is stored
      …here.physicalMemory(…)…  // query system characteristics
      …here.runningTasks()…     // query runtime characteristics
    }
}
```

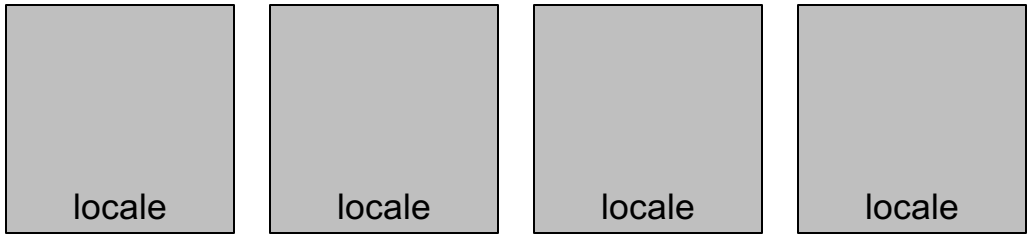


Locales (think: "compute nodes")

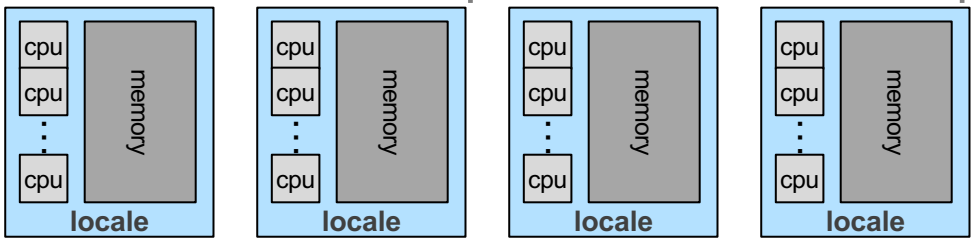
Copyright 2018 Cray Inc.

Classic Locales

- **Historically, Chapel's locales were black boxes**
  - Intra-node concerns handled by compiler, runtime, OS

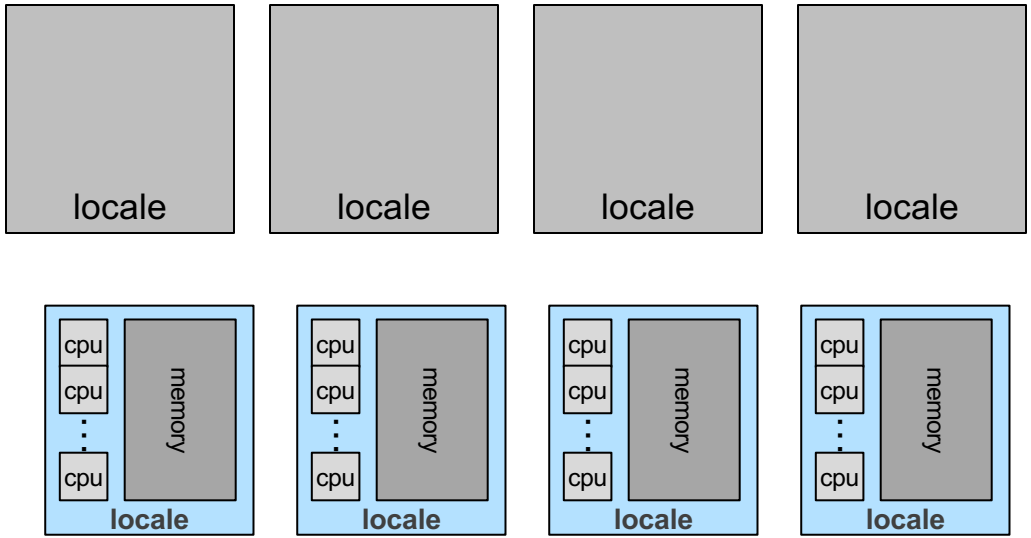


  - This was sufficient when compute nodes were simple



Copyright 2018 Cray Inc.

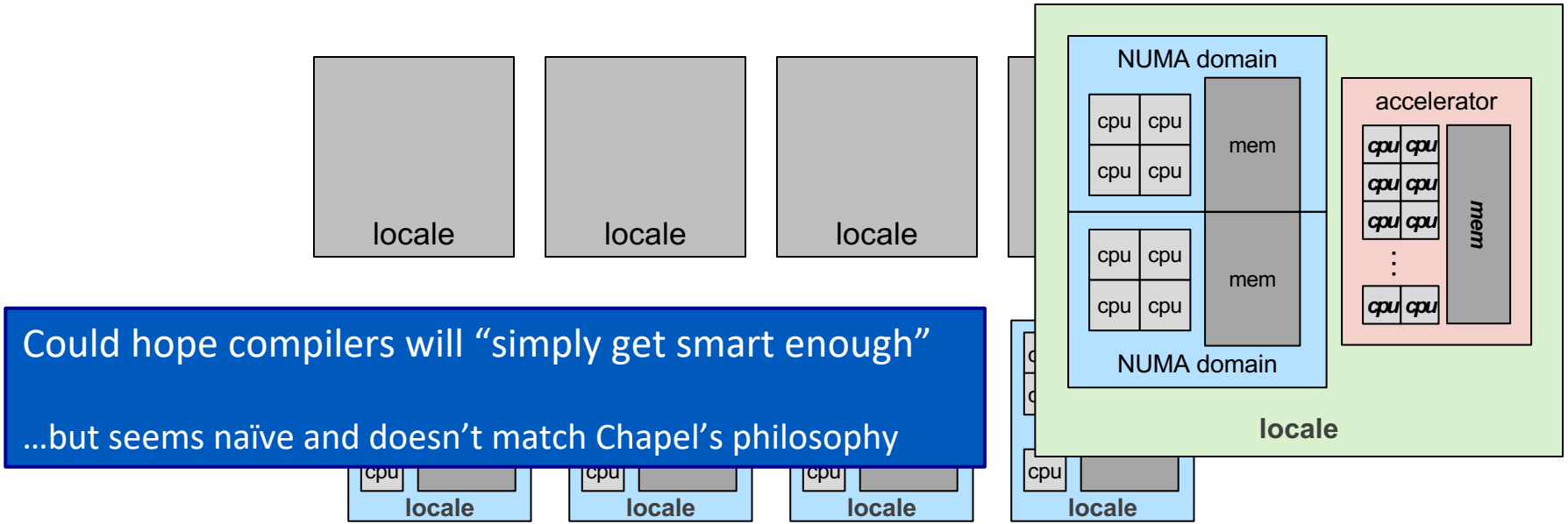
Classic Locales



Copyright 2018 Cray Inc.

Classic Locales

- **Classic model breaks down for more complex cases**
  - E.g. multiple flavors of memory or processors



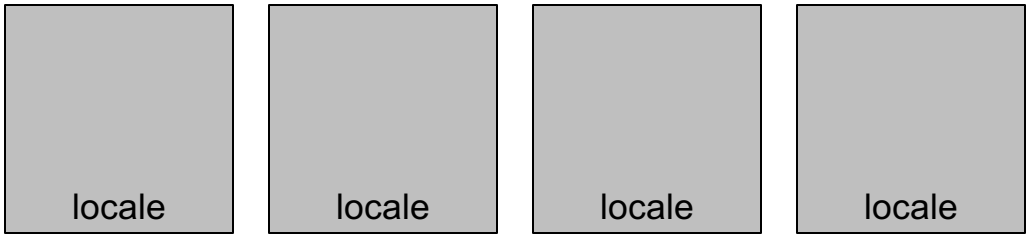
Could hope compilers will "simply get smart enough"

…but seems naïve and doesn't match Chapel's philosophy

Copyright 2018 Cray Inc.

Hierarchical Locales

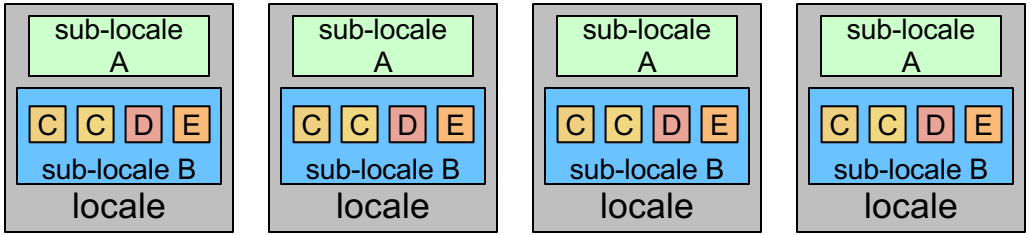
- So, we made locales hierarchical



Copyright 2018 Cray Inc.

Hierarchical Locales

- **So, we made locales hierarchical**
  - Locales can now themselves contain locales
    - E.g., an accelerator sub-locale, a scratchpad memory sub-locale



  - Target sub-locales with on-clauses, as before

```
on Locales[0].GPU do computationThatLikesGPUs();
```

    - Ideally, hide such logic in abstractions: domain maps, parallel iterators
  - Introduced a new multiresolution type: *locale models*

Copyright 2018 Cray Inc.

Chapel's Locale Models

- **User-specified type representing locales**
- **Similar goals to domain maps:**
  - Support user implementation of key high-level abstractions
  - Make language future-proof (w.r.t. emerging architectures)

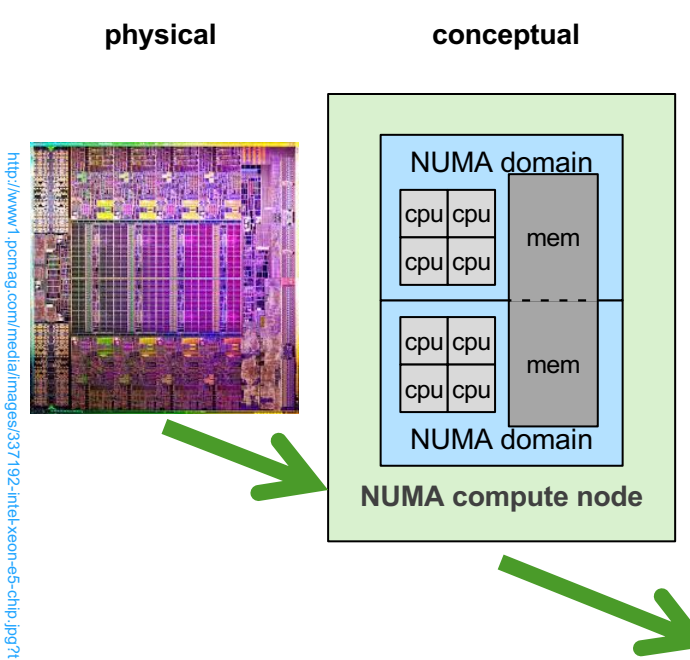
Copyright 2018 Cray Inc.

Authoring a Locale Model

- **Creating a locale model:**
  - Create a top-level locale object type
    - In turn, it can contain fields representing sub-locales
  - Each locale / sub-locale type must meet a required interface:
    - **Memory:** How is it managed? (malloc, realloc, free)
    - **Tasking:** How do I launch and synchronize tasks?
    - **Communication:** How are data & control transferred between locales?
      - gets, puts, active messages
      - widening of pointers

Copyright 2018 Cray Inc.

An Example: The numa Locale Model



http://www1.pcmag.com/media/images/337192-intel-xeon-e5-chip.jpg?thumb=y

$CHPL_HOME/modules/…/numa/LocaleModel.chpl

```
class NumaDomain : AbstractLocaleModel {
  const sid: chpl_sublocID_t;
}

// The node model
class LocaleModel : AbstractLocaleModel {
  const numSublocales: int;
  var childSpace: domain(1);
  var childLocales: [childSpace] NumaDomain;
}

// support for memory management
proc chpl_here_alloc(size:int, md:int(16)) { … }

// support for "on" statements
proc chpl_executeOn
     (loc: chpl_localeID_t, // target locale
      fn: int,              // on-body func idx
      args: c_void_ptr,     // func args
      args_size: int(32)    // args size
     ) { … }

// support for tasking stmts: begin, cobegin, coforall
proc chpl_taskListAddCoStmt
     (subloc_id: int,       // target subloc
      fn: int,              // body func idx
      args: c_void_ptr,     // func args
      ref tlist: _task_list, // task list
      tlist_node_id: int    // task list owner
     ) { … }
```


Copyright 2018 Cray Inc.

Locale Models: Status

- **All Chapel compilations use a locale model**
  - Set via environment variable or compiler flag
- **Current locale models:**
  - **flat**: the default, has no sublocales (as in the classic model)
  - **numa:** supports a sub-locale per NUMA domain within the node
  - **knl:** for Intel® Xeon Phi™: numa w/ sublocale for HBM/MCDRAM
- **In practice…**
  - we use the 'flat' locale model almost exclusively

Copyright 2018 Cray Inc.

Performance: ‘numa’ vs. ‘flat’

- **Using ‘numa’ leads to performance overheads**
  - Local arrays must be “chunked” between the numa sublocales
  - Indexing must do extra work to pick the right chunk
- **Though ‘flat’ has no sub-locales, it’s also NUMA-aware**
  - First-touch heuristics used to map sub-arrays to NUMA domains
  - Yet array remains contiguous in memory ⇒ simple indexing
- **As a result, ‘numa’ rarely outperforms ‘flat’**

Copyright 2018 Cray Inc.

Challenges: Static analysis & locality

- **Local vs. Remote Locales** (distributed or shared memory?)
  - In general, given:

    ```
    on x.locale …
    ```

    can't statically tell whether locale shares memory with `here` or not
  - May result in overheads due to conservatism
    - "Assuming it's remote, I'll introduce wide pointers & communication, …"

Copyright 2018 Cray Inc.

Challenges: Memory-only locales

- **What if a locale only represents memory? (say)**
  - Have interface run tasks on other locales with processors
    - Choice of policy is up to locale model author
    - Round-robin, dynamic load-balance, nearest, …

Copyright 2018 Cray Inc.

Wrapping Up

Copyright 2018 Cray Inc.

Summary

- **Chapel's design uses a multiresolution philosophy**
  - High-level for productivity
  - Low-level for control
  - User-extensible for flexibility, future-proof design
- **Locale models support mapping to new architectures**
  - Provide bridge from compiler to system resources
  - Enables targeting of heterogeneous resources
- **Chapel performance can match C+MPI+OpenMP**
  - With improvements in readability, writability, code size

Copyright 2018 Cray Inc.

Possible Discussion Topics

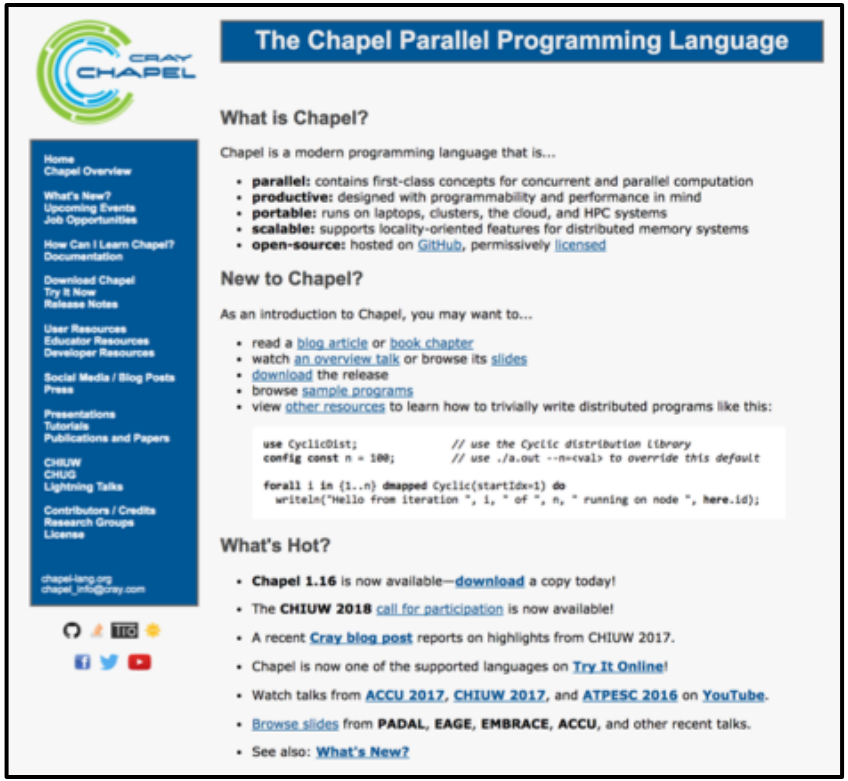
- Error-handling / Exceptions

Copyright 2018 Cray Inc.

Chapel Resources

Copyright 2018 Cray Inc.

Chapel Central: https://chapel-lang.org/



The Chapel Parallel Programming Language

Home
Chapel Overview

What's New?
Upcoming Events
Job Opportunities

How Can I Learn Chapel?
Documentation

Download Chapel
Try It Now
Release Notes

User Resources
Educator Resources
Developer Resources

Social Media / Blog Posts
Press

Presentations
Tutorials
Publications and Papers

CHIUW
CHUG
Lightning Talks

Contributors / Credits
Research Groups
License

chapel-lang.org
chapel_info@cray.com

What is Chapel?

Chapel is a modern programming language that is...

- **parallel:** contains first-class concepts for concurrent and parallel computation
- **productive:** designed with programmability and performance in mind
- **portable:** runs on laptops, clusters, the cloud, and HPC systems
- **scalable:** supports locality-oriented features for distributed memory systems
- **open-source:** hosted on GitHub, permissively licensed

New to Chapel?

As an introduction to Chapel, you may want to...

- read a blog article or book chapter
- watch an overview talk or browse its slides
- download the release
- browse sample programs
- view other resources to learn how to trivially write distributed programs like this:

```
use CyclicDist;              // use the Cyclic distribution library
config const n = 100;        // use ./a.out --n=<val> to override this default

forall i in {1..n} dmapped Cyclic(startIdx=1) do
  writeln("Hello from iteration ", i, " of ", n, " running on node ", here.id);
```

What's Hot?

- **Chapel 1.16** is now available—**download** a copy today!
- The **CHIUW 2018** call for participation is now available!
- A recent **Cray blog post** reports on highlights from CHIUW 2017.
- Chapel is now one of the supported languages on **Try It Online**!
- Watch talks from **ACCU 2017**, **CHIUW 2017**, and **ATPESC 2016** on **YouTube**.
- Browse slides from **PADAL**, **EAGE**, **EMBRACE**, **ACCU**, and other recent talks.
- See also: **What's New?**

Copyright 2018 Cray Inc.

How to Track Chapel

http://facebook.com/ChapelLanguage

http://twitter.com/ChapelLanguage

https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/

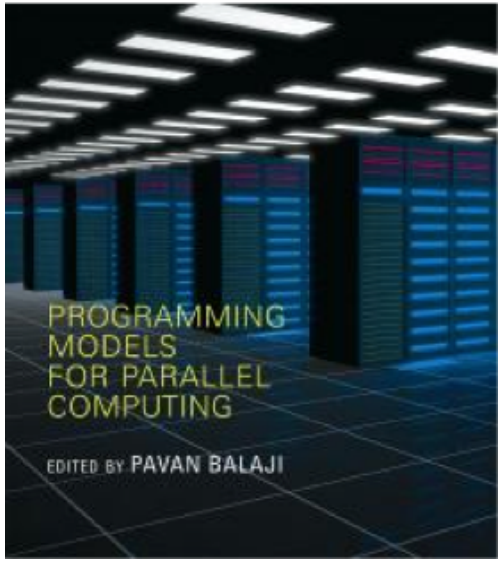
chapel-announce@lists.sourceforge.net

Copyright 2018 Cray Inc.

Suggested Reading (healthy attention spans)

Chapel chapter from ***Programming Models for Parallel Computing***

- a detailed overview of Chapel's history, motivating themes, features
- published by MIT Press, November 2015
- edited by Pavan Balaji (Argonne)
- chapter is now also available online



Other Chapel papers/publications available at **https://chapel-lang.org/papers.html**

Copyright 2018 Cray Inc.

Suggested Reading (short attention spans)

CHIUW 2017: Surveying the Chapel Landscape, Cray Blog, July 2017.

- *a run-down of recent events*

Chapel: Productive Parallel Programming, Cray Blog, May 2013.

- *a short-and-sweet introduction to Chapel*

Six Ways to Say "Hello" in Chapel **(parts 1, 2, 3)**, Cray Blog, Sep-Oct 2015.

- *a series of articles illustrating the basics of parallelism and locality in Chapel*

Why Chapel? **(parts 1, 2, 3)**, Cray Blog, Jun-Oct 2014.

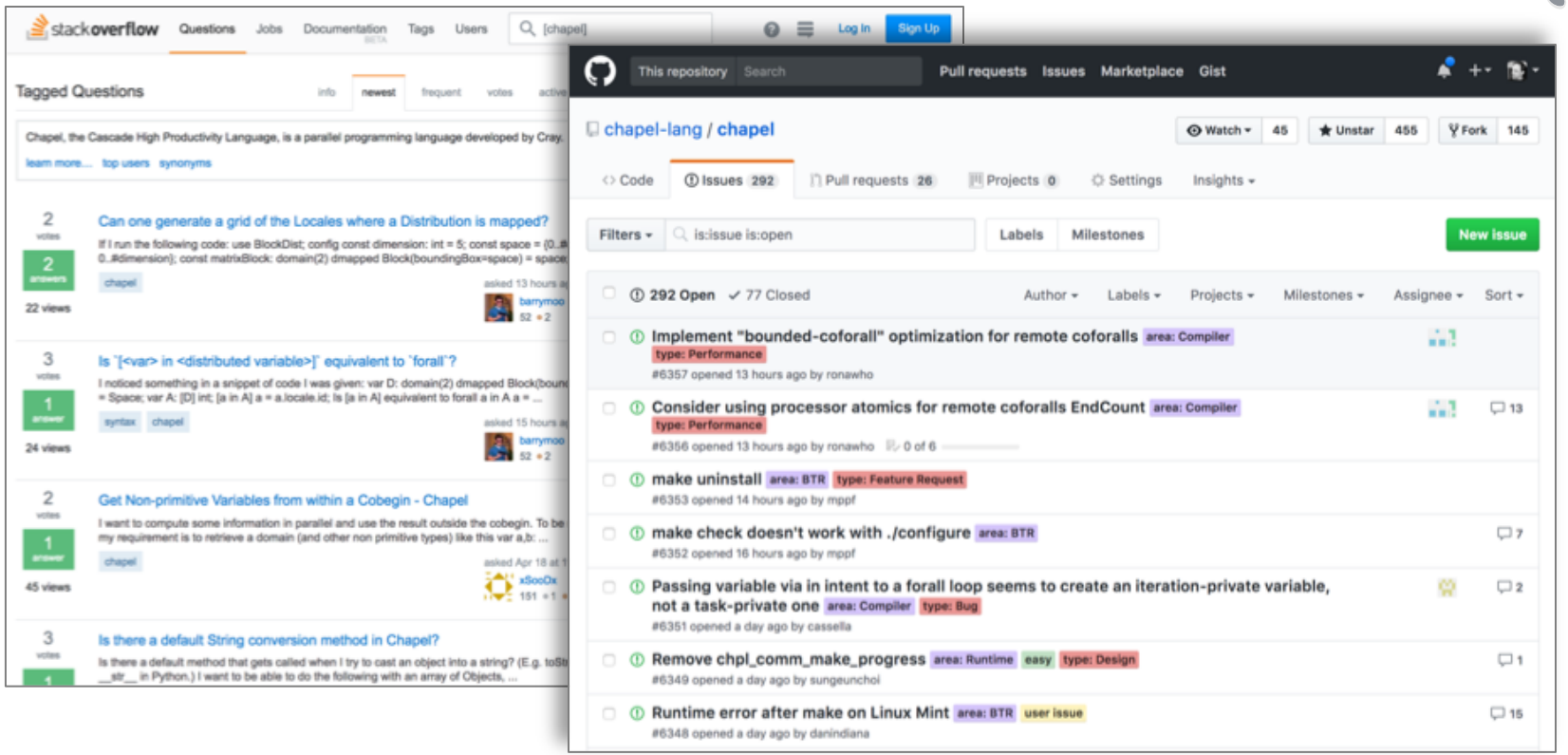
- *a series of articles answering common questions about why we are pursuing Chapel in spite of the inherent challenges*

[Ten] Myths About Scalable Programming Languages, IEEE TCSC Blog (index available on chapel-lang.org "blog posts" page), Apr-Nov 2012.

- *a series of technical opinion pieces designed to argue against standard reasons given for not developing high-level parallel languages*

Copyright 2018 Cray Inc.

Chapel StackOverflow and GitHub Issues



Copyright 2018 Cray Inc.

Where to..

Submit bug reports:

GitHub issues for chapel-lang/chapel: public bug forum

chapel_bugs@cray.com: for reporting non-public bugs

Ask User-Oriented Questions:

StackOverflow: when appropriate / other users might care

#chapel-users (irc.freenode.net): user-oriented IRC channel

chapel-users@lists.sourceforge.net: user discussions

Discuss Chapel development

chapel-developers@lists.sourceforge.net: developer discussions

#chapel-developers (irc.freenode.net): developer-oriented IRC channel

Discuss Chapel's use in education

chapel-education@lists.sourceforge.net: educator discussions

Directly contact Chapel team at Cray: chapel_info@cray.com

Copyright 2018 Cray Inc.

Questions?

Copyright 2018 Cray Inc.

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

Copyright 2018 Cray Inc.

CRAY

THE SUPERCOMPUTER COMPANY