Chapel Comes of Age: Productive Parallelism at Scale

CUG 2018

Brad Chamberlain, Chapel Team, Cray Inc.

Or: What's Chapel been up to since CUG 2013?

CUG 2018

Brad Chamberlain, Chapel Team, Cray Inc.

What is Chapel?

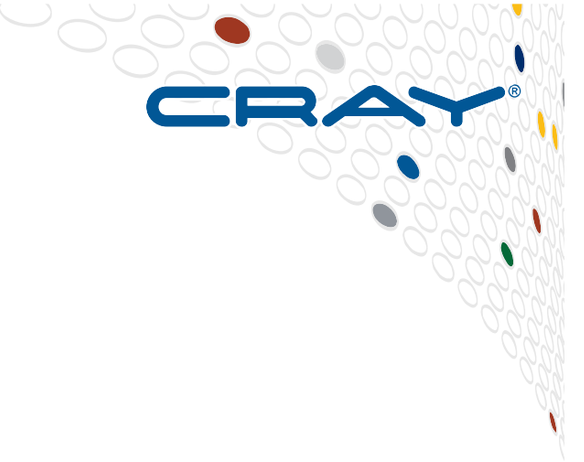
Chapel: A productive parallel programming language

- portable & scalable
- open-source & collaborative

Goals:

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

Chapel and Productivity



Chapel aims to be as…

…**programmable** as Python

…**fast** as Fortran

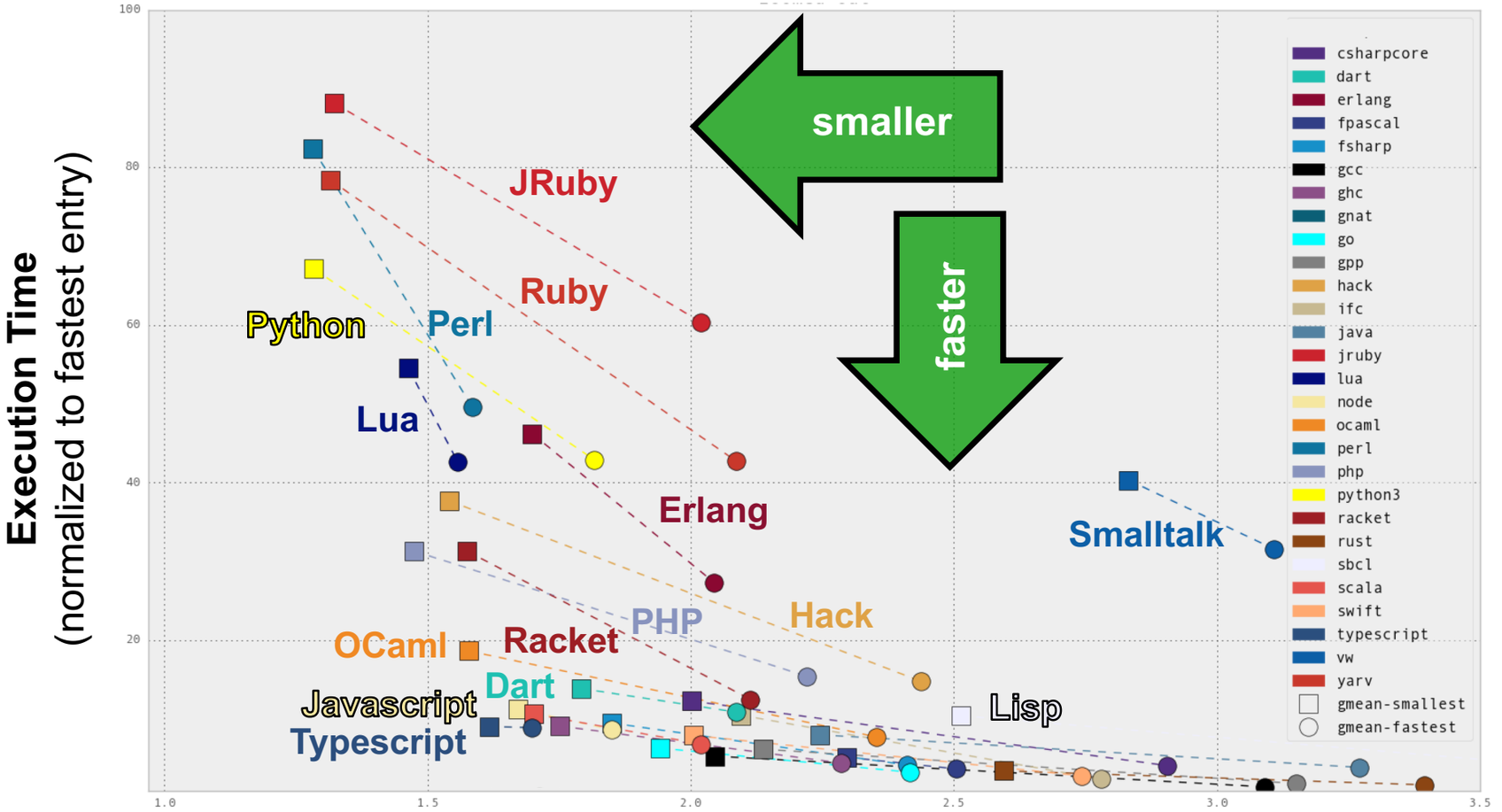
…**scalable** as MPI, SHMEM, or UPC

…**portable** as C

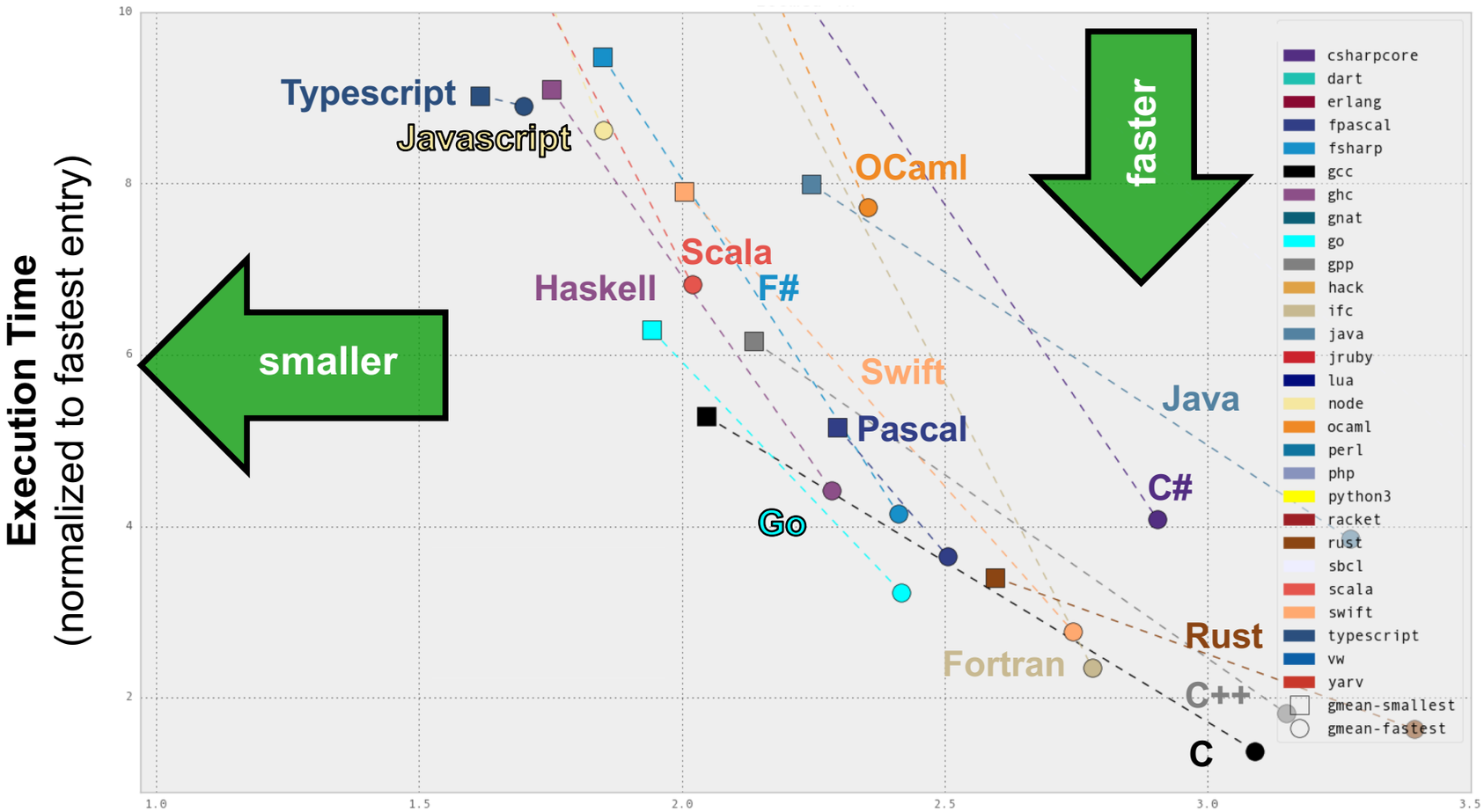
…**flexible** as C++

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

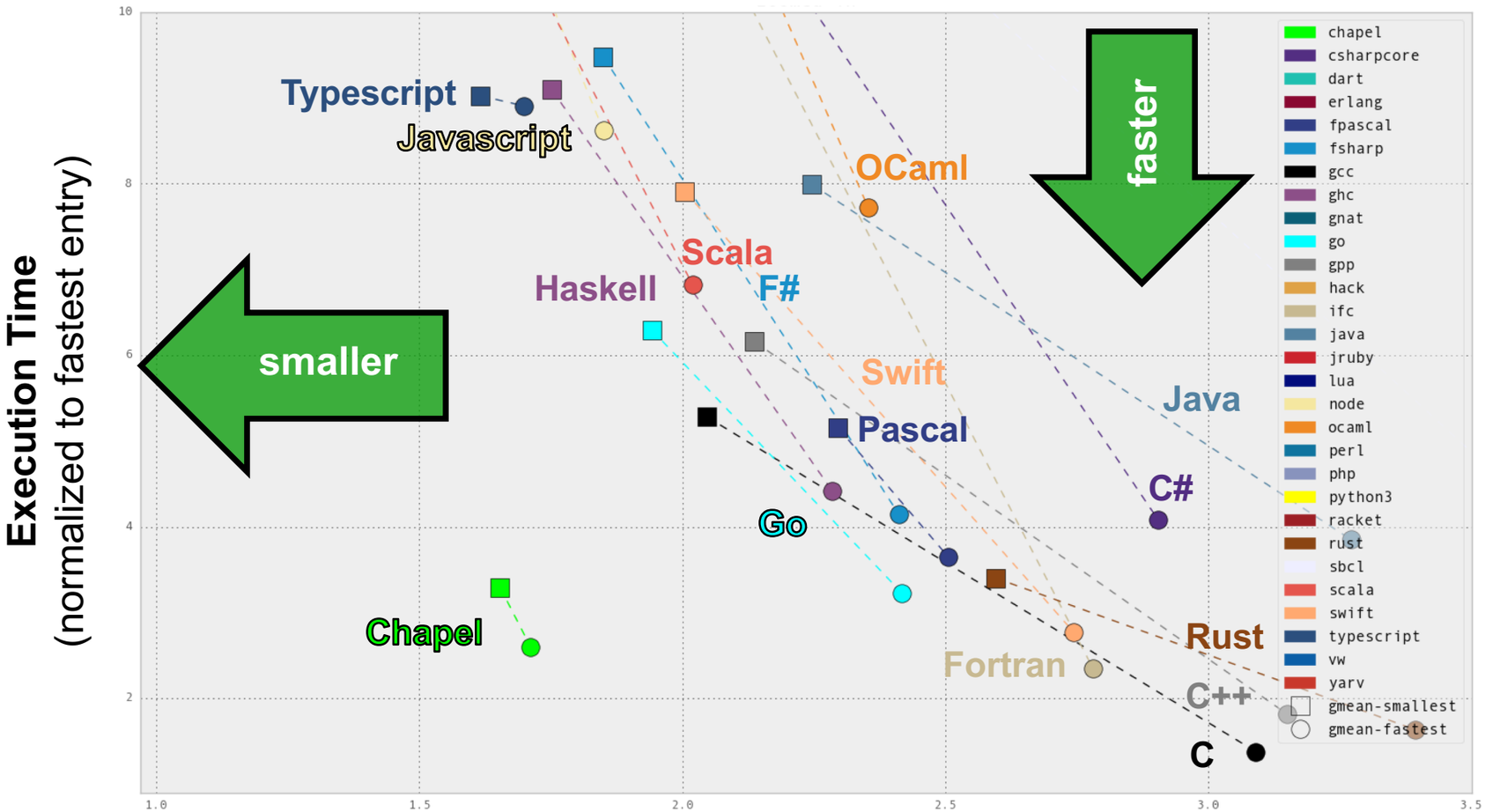
CLBG Cross-Language Summary
(Oct 2017 standings)



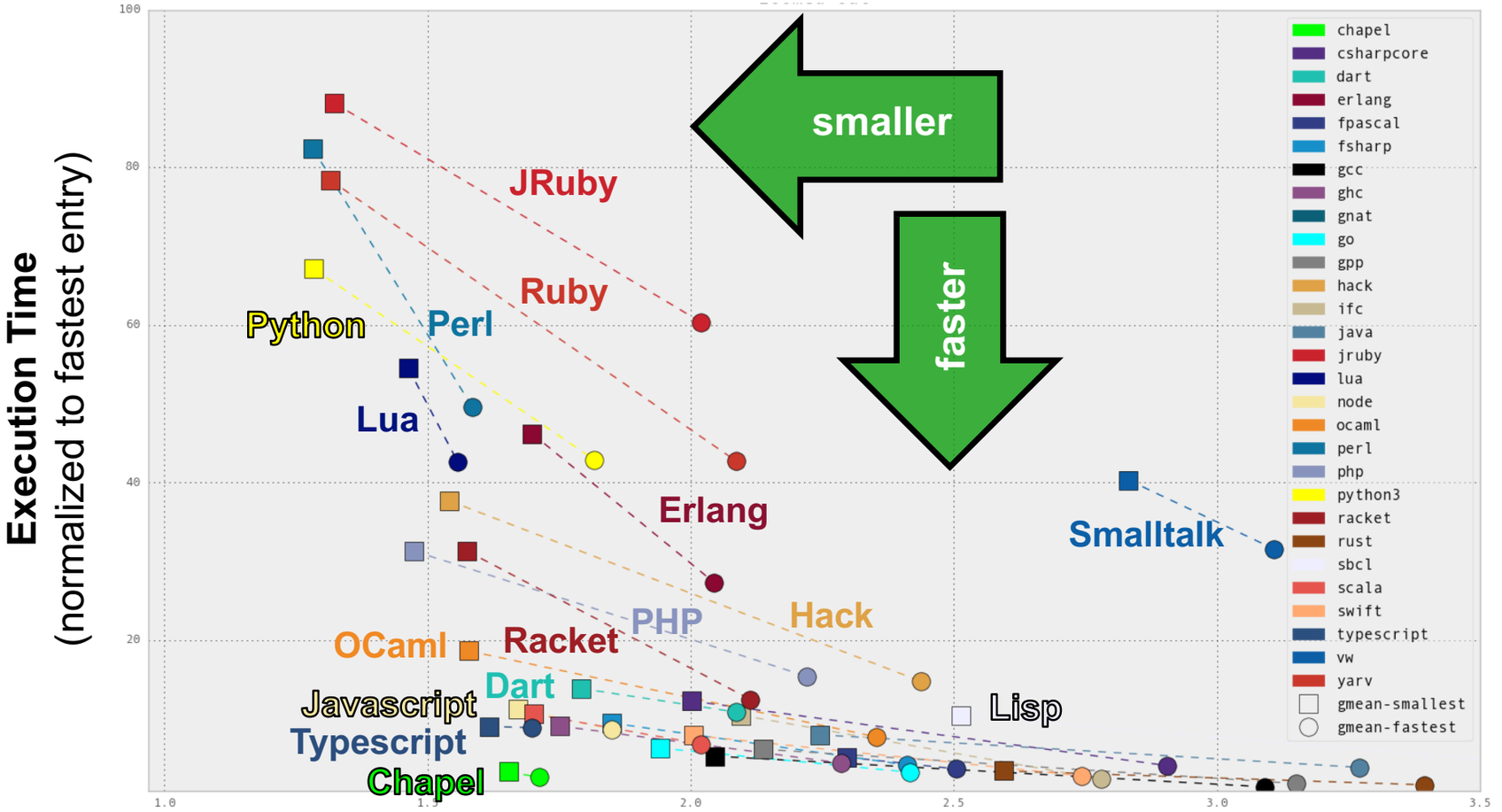
CLBG Cross-Language Summary
(Oct 2017 standings, zoomed in)



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



CLBG Cross-Language Summary
(Oct 2017 standings)



CLBG: Qualitative Code Comparisons

Can also browse program source code *(but this requires actual thought!)*:

```
proc main() {
  printColorEquations();

  const group1 = [i in 1..popSize1] new Chameneos(i, ((i-1)%3):Color);
  const group2 = [i in 1..popSize2] new Chameneos(i, colors10[i]);

  cobegin {
    holdMeetings(group1, n);
    holdMeetings(group2, n);
  }

  print(group1);
  print(group2);

  for c in group1 do delete c;
  for c in group2 do delete c;
}


//
// Print the results of getNewColor() for all color pairs.
//
proc printColorEquations() {
  for c1 in Color do
    for c2 in Color do
      writeln(c1, " + ", c2, " -> ", getNewColor(c1, c2));
  writeln();
}


//
// Hold meetings among the population by creating a shared meeting
// place, and then creating per-chameneos tasks to have meetings.
//
proc holdMeetings(population, numMeetings) {
  const place = new MeetingPlace(numMeetings);

  coforall c in population do           // create a task per chameneos
    c.haveMeetings(place, population);

  delete place;
}
```

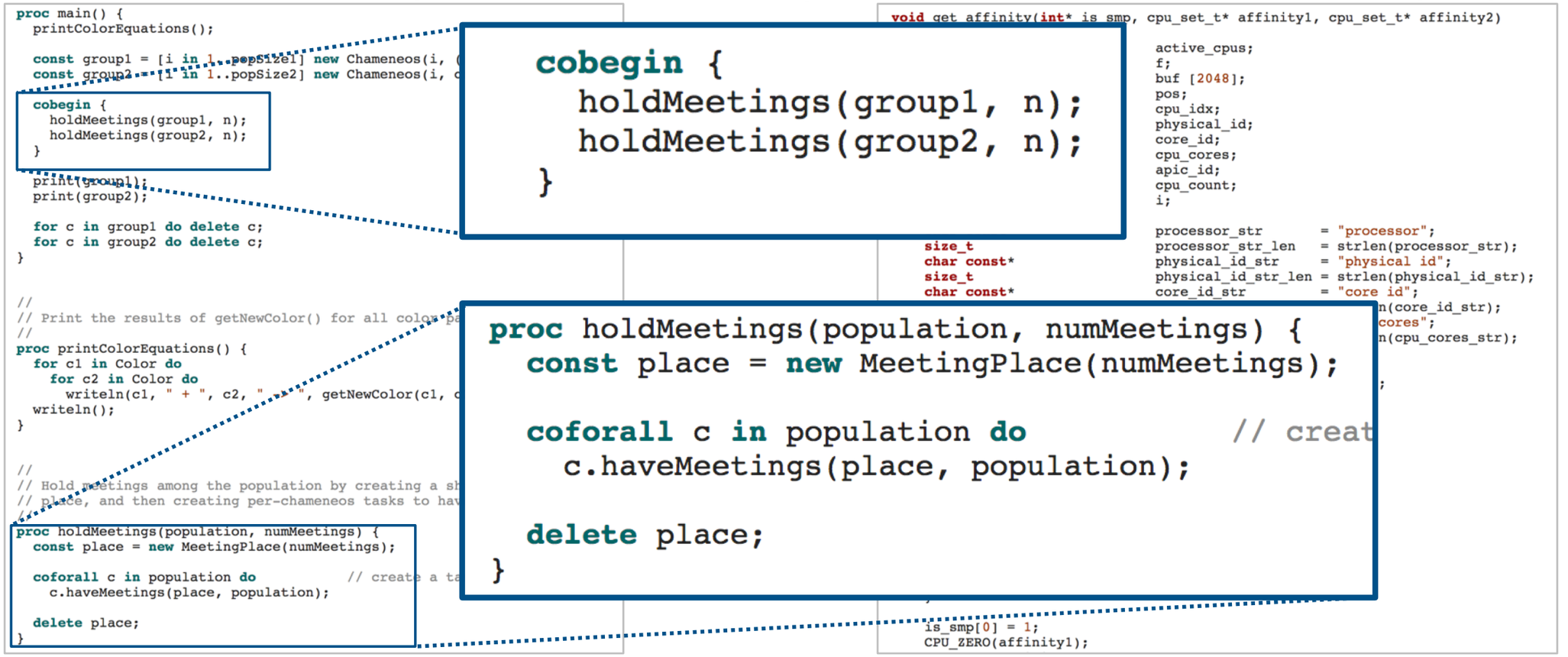
excerpt from 1210 gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t                   active_cpus;
    FILE*                       f;
    char                        buf [2048];
    char const*                 pos;
    int                         cpu_idx;
    int                         physical_id;
    int                         core_id;
    int                         cpu_cores;
    int                         apic_id;
    size_t                      cpu_count;
    size_t                      i;

    char const*                 processor_str       = "processor";
    size_t                      processor_str_len   = strlen(processor_str);
    char const*                 physical_id_str     = "physical id";
    size_t                      physical_id_str_len = strlen(physical_id_str);
    char const*                 core_id_str         = "core id";
    size_t                      core_id_str_len     = strlen(core_id_str);
    char const*                 cpu_cores_str       = "cpu cores";
    size_t                      cpu_cores_str_len   = strlen(cpu_cores_str);

    CPU_ZERO(&active_cpus);
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
    cpu_count = 0;
    for (i = 0; i != CPU_SETSIZE; i += 1)
    {
        if (CPU_ISSET(i, &active_cpus))
        {
            cpu_count += 1;
        }
    }

    if (cpu_count == 1)
    {
        is_smp[0] = 0;
        return;
    }

    is_smp[0] = 1;
    CPU_ZERO(affinity1);
```

excerpt from 2863 gz C gcc entry

CLBG: Qualitative Code Comparisons

Can also browse program source code *(but this requires actual thought!)*:



```
proc main() {
  printColorEquations();

  const group1 = [i in 1..popSize1] new Chameneos(i, (
  const group2 = [i in 1..popSize2] new Chameneos(i, c

  cobegin {
    holdMeetings(group1, n);
    holdMeetings(group2, n);
  }

  print(group1);
  print(group2);

  for c in group1 do delete c;
  for c in group2 do delete c;
}


//
// Print the results of getNewColor() for all color pa
//
proc printColorEquations() {
  for c1 in Color do
    for c2 in Color do
      writeln(c1, " + ", c2, " -> ", getNewColor(c1, c
  writeln();
}


//
// Hold meetings among the population by creating a sh
// place, and then creating per-chameneos tasks to hav
//
proc holdMeetings(population, numMeetings) {
  const place = new MeetingPlace(numMeetings);

  coforall c in population do           // create a ta
    c.haveMeetings(place, population);

  delete place;
}
```

```
cobegin {
  holdMeetings(group1, n);
  holdMeetings(group2, n);
}
```

```
proc holdMeetings(population, numMeetings) {
  const place = new MeetingPlace(numMeetings);

  coforall c in population do              // creat
    c.haveMeetings(place, population);

  delete place;
}
```

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)

  active_cpus;
  f;
  buf [2048];
  pos;
  cpu_idx;
  physical_id;
  core_id;
  cpu_cores;
  apic_id;
  cpu_count;
  i;

  size_t                processor_str_len    = strlen(processor_str);
  char const*           physical_id_str      = "physical id";
  size_t                physical_id_str_len  = strlen(physical_id_str);
  char const*           core_id_str          = "core id";
                                               n(core_id_str);
                                               cores";
                                               n(cpu_cores_str);

  is_smp[0] = 1;
  CPU_ZERO(affinity1);
```

excerpt from 1210 gz Chapel entry

excerpt from 2863 gz C gcc entry

CLBG: Qualitative Code Comparisons

Can also browse program source code *(but this requires actual thought!)*:

```
proc main() {
char const*                     core_id_str          = "core id"
size_t                          core_id_str_len      = strlen(co
char const*                     cpu_cores_str        = "cpu core
size_t                          cpu_cores_str_len    = strlen(cp

CPU_ZERO(&active_cpus);
sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
cpu_count = 0;
for (i = 0; i != CPU_SETSIZE; i += 1)
{
    if (CPU_ISSET(i, &active_cpus))
    {
        cpu_count += 1;
    }
}

if (cpu_count == 1)
{
    is_smp[0] = 0;
    return;
}
```

excerpt from 1210 gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t                   active_cpus;
    FILE*                       f;
    char                        buf [2048];
    char const*                 pos;
    int                         cpu_idx;
    int                         physical_id;
    int                         core_id;
    int                         cpu_cores;
    int                         apic_id;
    size_t                      cpu_count;
    size_t                      i;

    char const*                 processor_str       = "processor";
    size_t                      processor_str_len   = strlen(processor_str);
    char const*                 physical_id_str     = "physical id";
    size_t                      physical_id_str_len = strlen(physical_id_str);
    char const*                 core_id_str         = "core id";
    size_t                      core_id_str_len     = strlen(core_id_str);
    char const*                 cpu_cores_str       = "cpu cores";
    size_t                      cpu_cores_str_len   = strlen(cpu_cores_str);

    CPU_ZERO(&active_cpus);
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
    cpu_count = 0;
    for (i = 0; i != CPU_SETSIZE; i += 1)
    {
        if (CPU_ISSET(i, &active_cpus))
        {
            cpu_count += 1;
        }
    }

    if (cpu_count == 1)
    {
        is_smp[0] = 0;
        return;
    }

    is_smp[0] = 1;
    CPU_ZERO(affinity1);
```

excerpt from 2863 gz C gcc entry

The Chapel Team at Cray (May 2018)

Chapel Community Partners

Lawrence Berkeley
National Laboratory

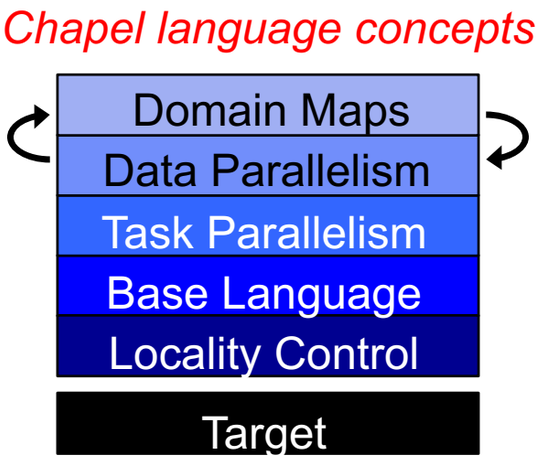
(and several others…)

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

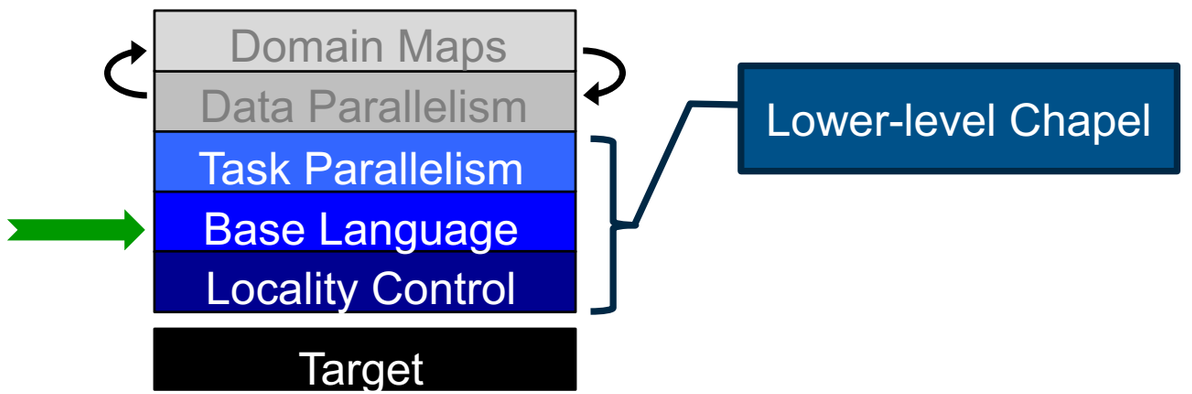
Outline

- ✓ What is Chapel?
- ➢ **Chapel Overview**
- Chapel: Then vs. Now
- Chapel User Profiles
- What's Next?

Chapel language feature areas



Base Language



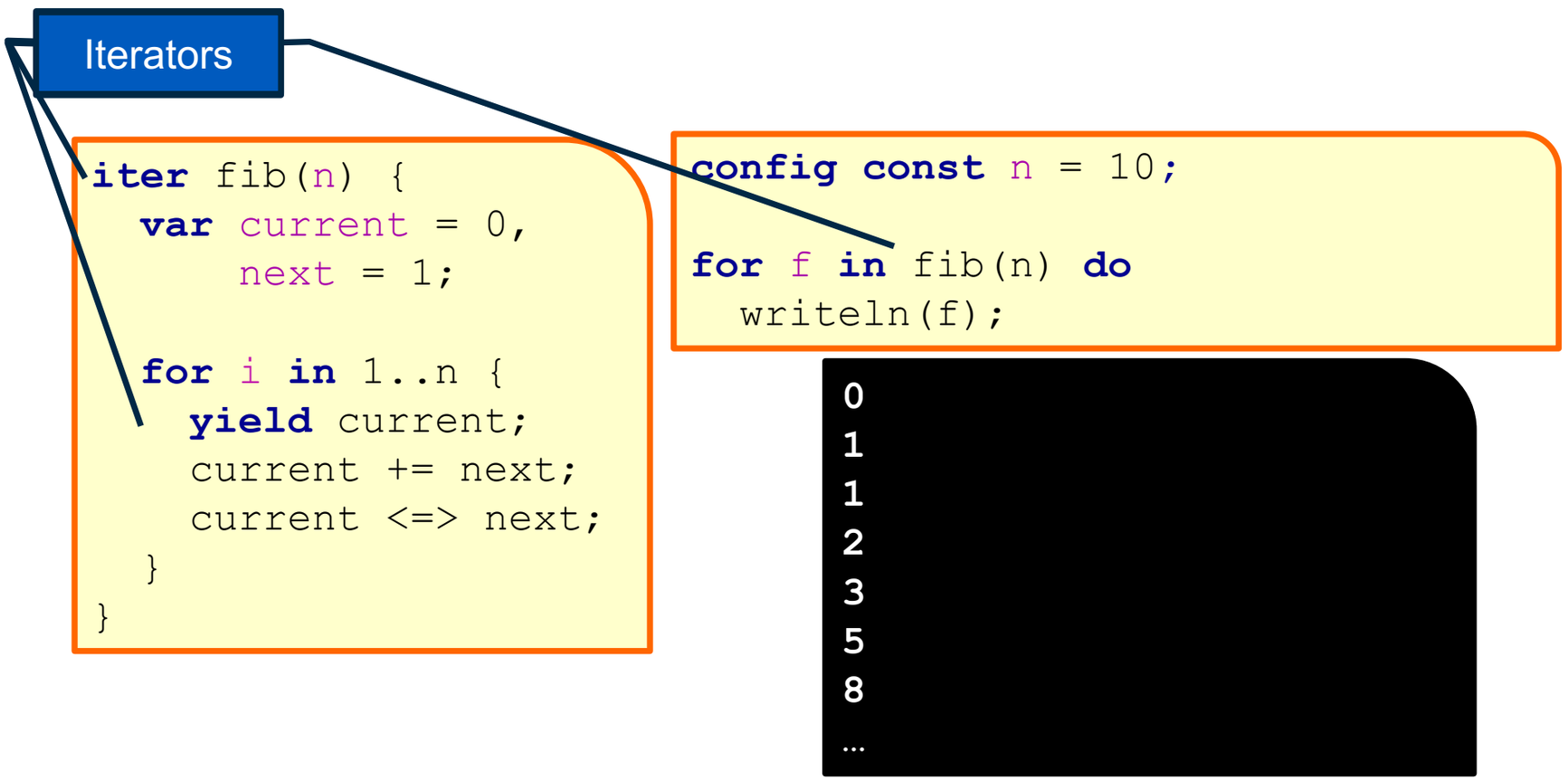
Base Language Features, by example

```
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
...
```

Base Language Features, by example



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
...
```

Base Language Features, by example


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

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

```
config const n: int = 10;

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

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

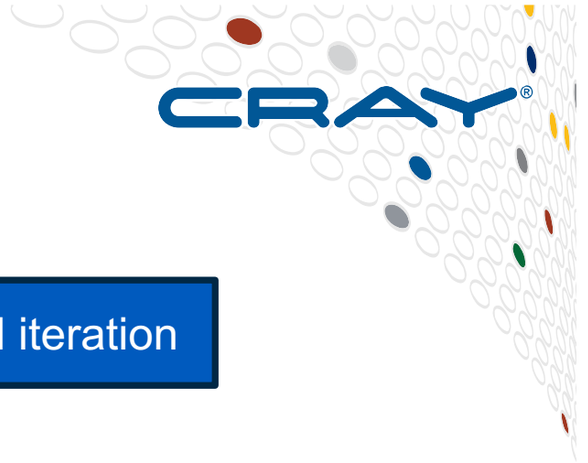
Base Language Features, by example

```
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
...
```

Base Language Features, by example



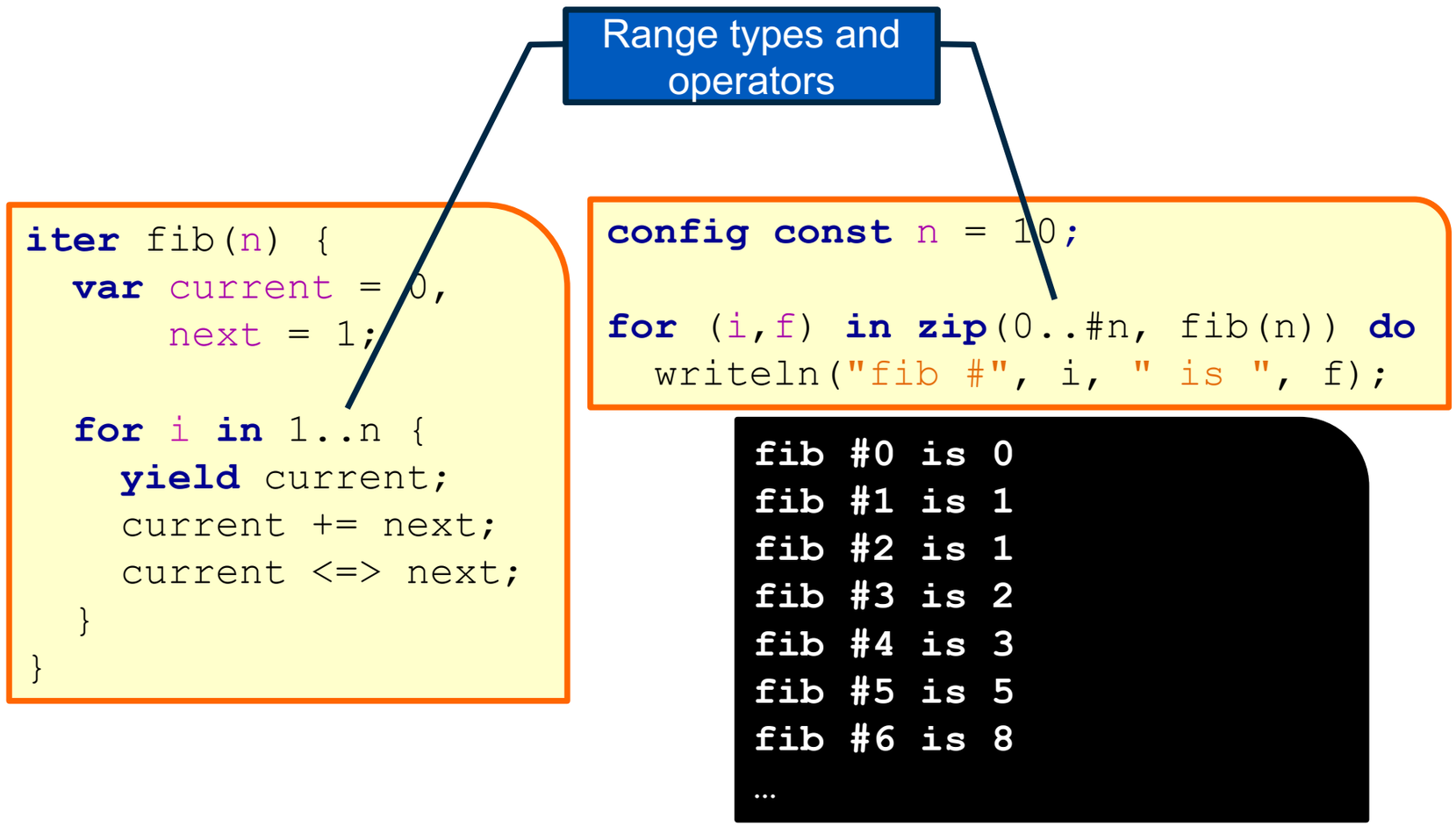
Zippered iteration

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

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

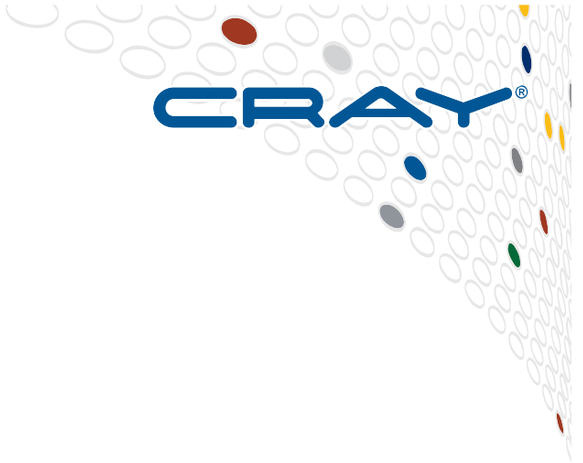
```
config const n = 10;

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
…
```

Base Language Features, by example



Base Language Features, by example



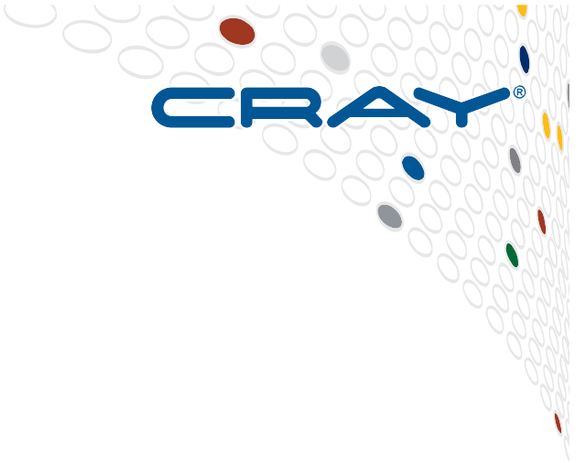
Tuples

```
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
…
```

Base Language Features, by example

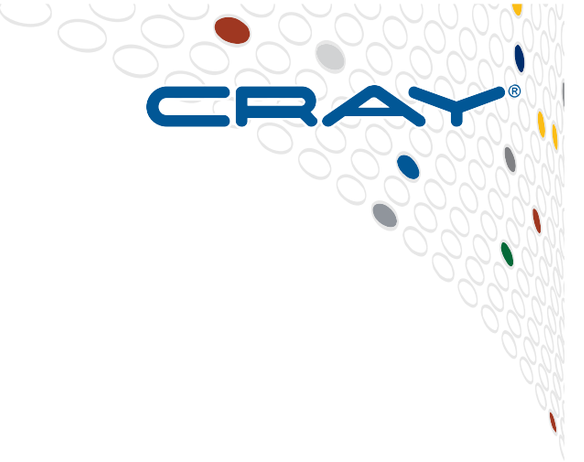


```
iter fib(n) {
  var current = 0,
      next = 1;

  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);
```

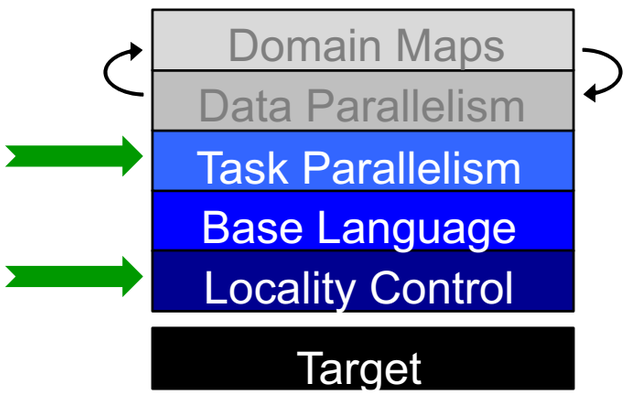
```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
…
```

Other Base Language Features



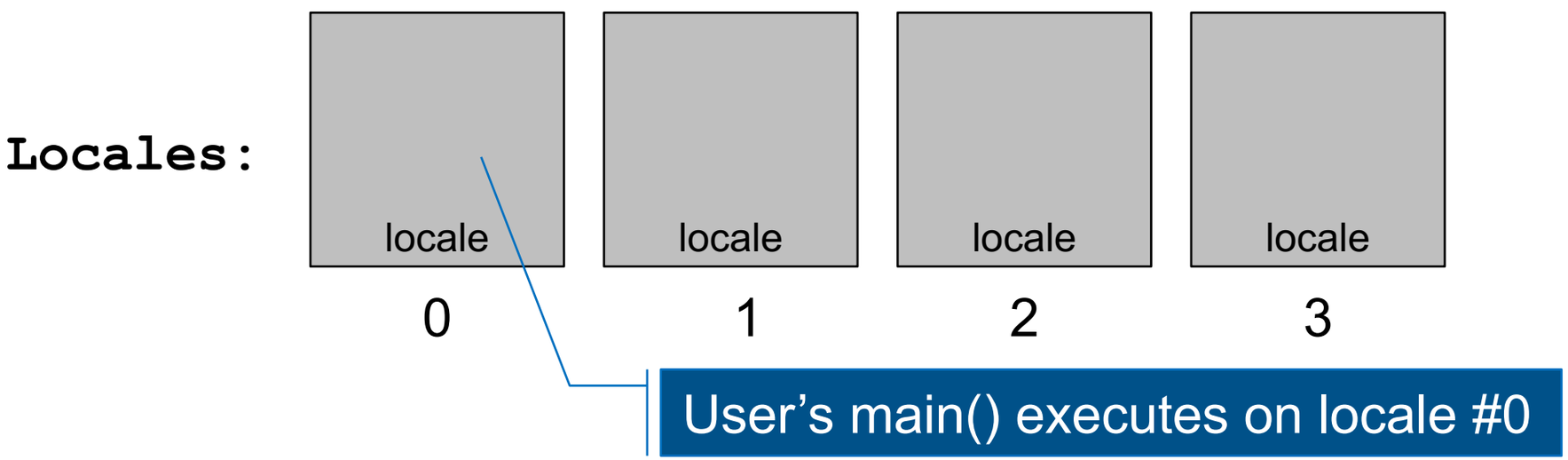
- Object-oriented features
- Generic programming / polymorphism
- Procedure overloading / filtering
- Default args, arg intents, keyword-based arg passing
- Argument type queries / pattern-matching
- Compile-time meta-programming
- Modules (namespaces)
- Error-handling
- and more…

Task Parallelism and Locality Control

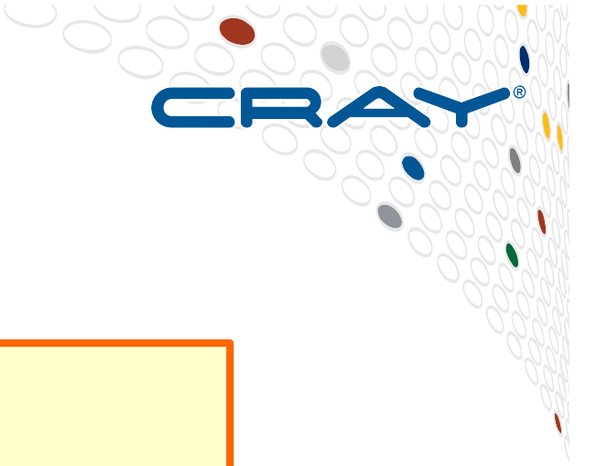


Locales, briefly

- Locales can run tasks and store variables
  - Think “compute node”



Task Parallelism and Locality, by example

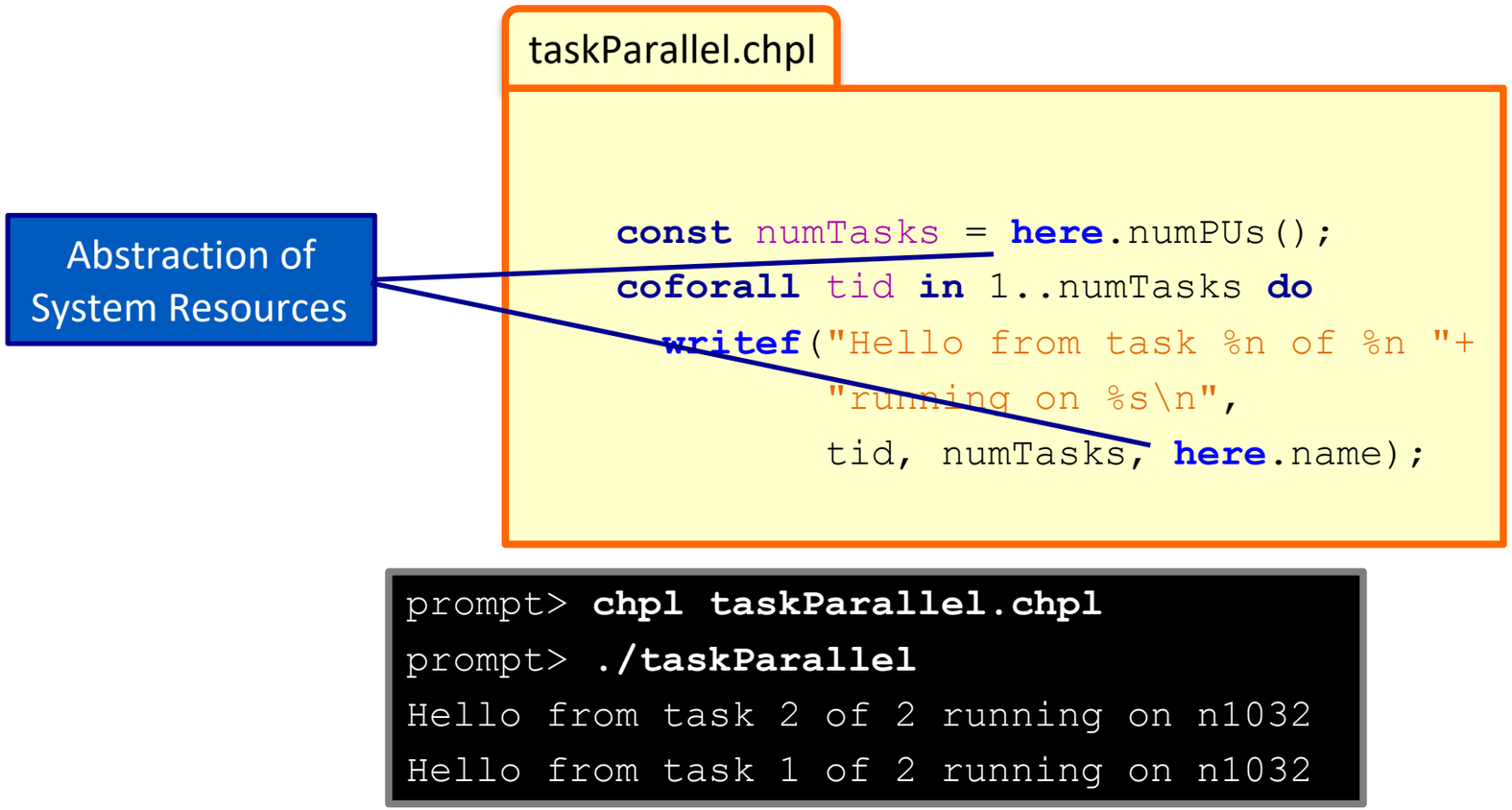


taskParallel.chpl

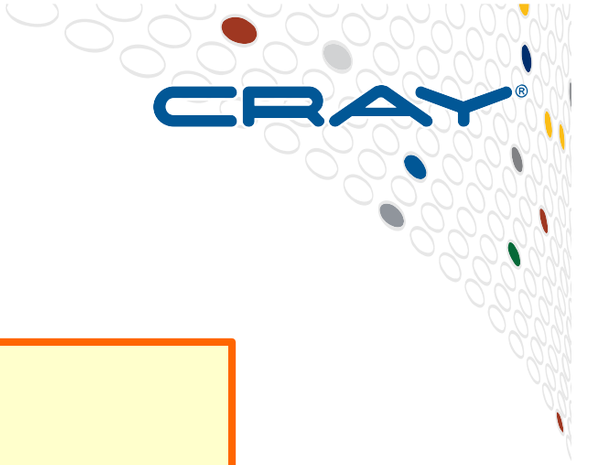
```
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  writef("Hello from task %n of %n "+
         "running on %s\n",
         tid, numTasks, here.name);
```

```
prompt> chpl taskParallel.chpl
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```

Task Parallelism and Locality, by example



Task Parallelism and Locality, by example



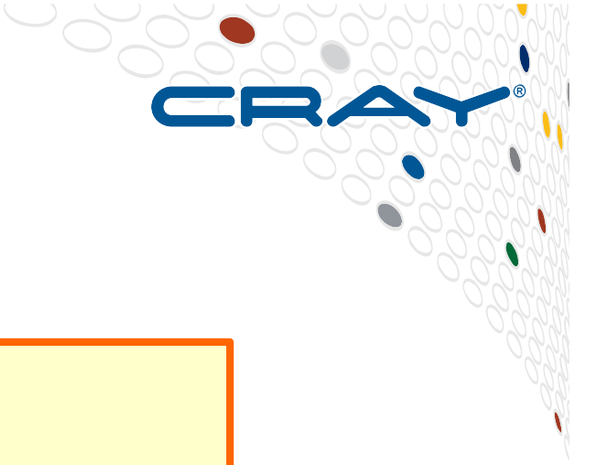
High-Level
Task Parallelism

taskParallel.chpl

```
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
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```

Task Parallelism and Locality, by example



This is a shared memory program

Nothing has referred to remote locales, explicitly or implicitly

taskParallel.chpl

```
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
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```

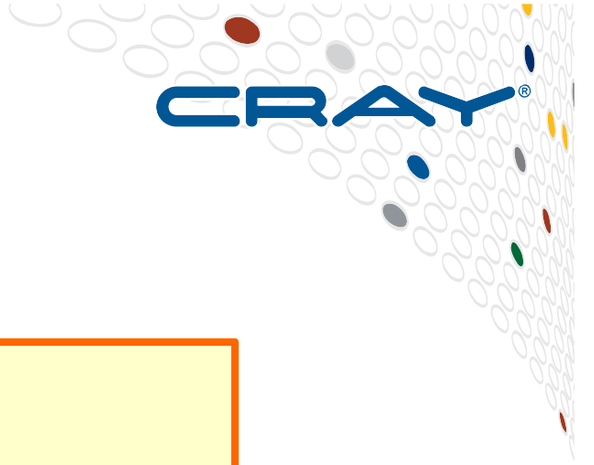
Task Parallelism and Locality, by example

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
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
```

Task Parallelism and Locality, by example



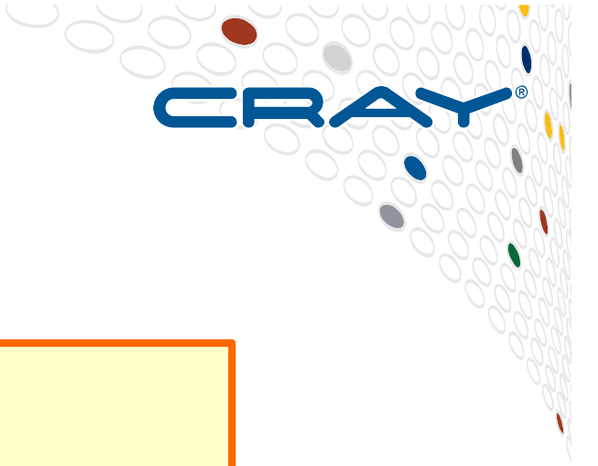
Abstraction of System Resources

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
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
```

Task Parallelism and Locality, by example



Control of Locality/Affinity

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
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
```

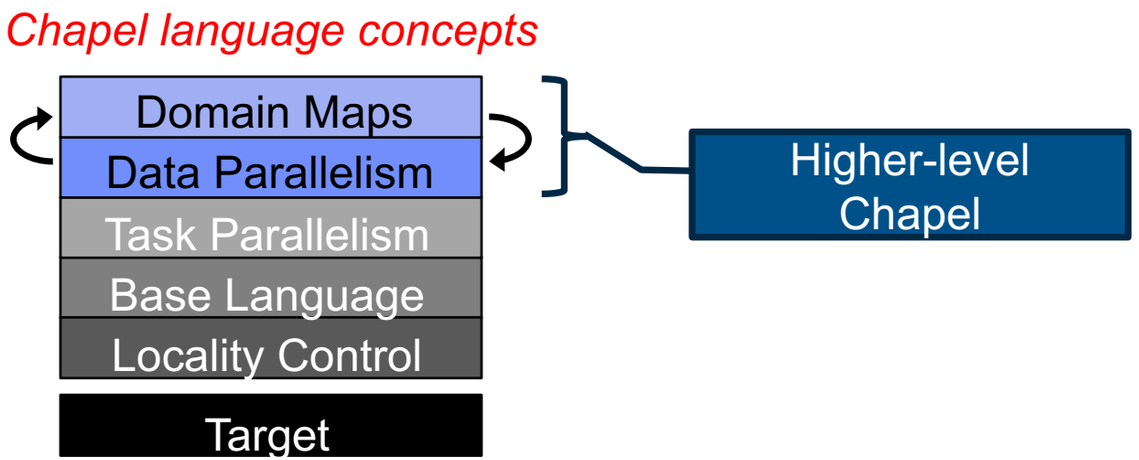
Task Parallelism and Locality, by example

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
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
```

Data Parallelism in Chapel



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
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
```

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
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
```

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
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
```

Data Parallelism, by example

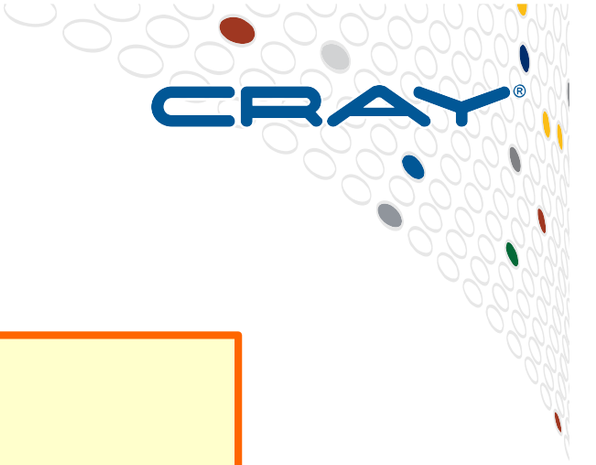
Data-Parallel Forall Loops

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
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
```

Data Parallelism, by example



This is a shared memory program

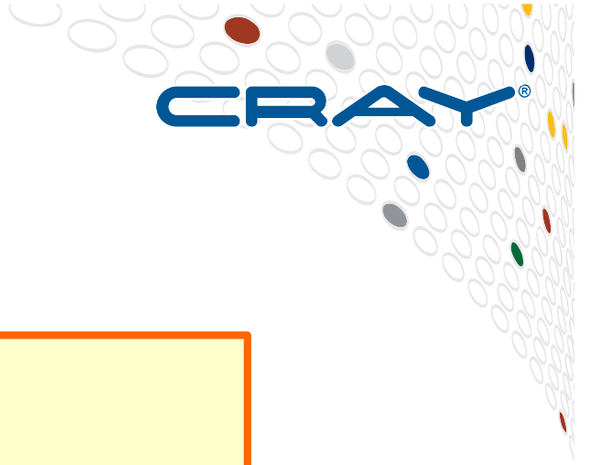
Nothing has referred to remote locales, explicitly or implicitly

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
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
```

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);
```

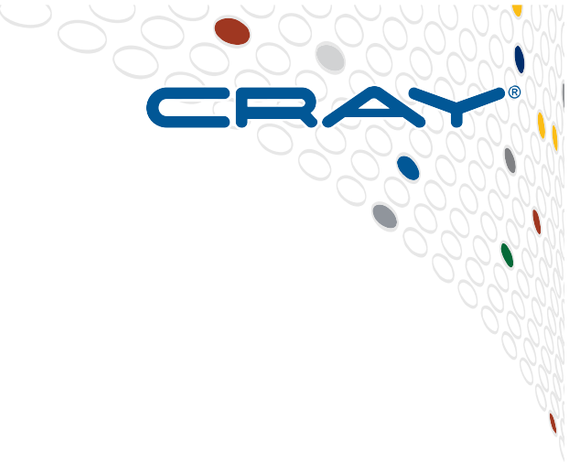
```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

Distributed Data Parallelism, by example

dataParallel.chpl

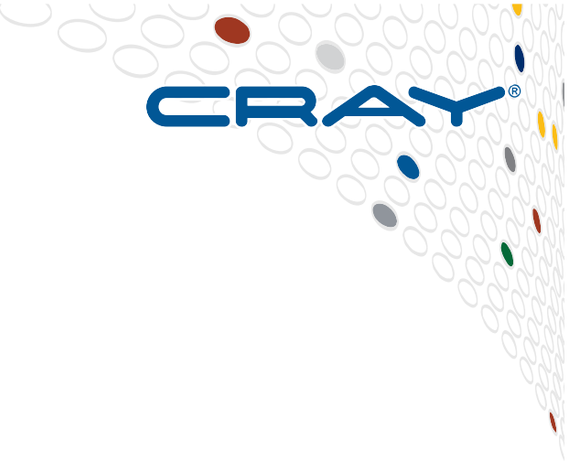
```
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
        dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl
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
```



A Brief History of Chapel

A Brief History of Chapel: Infancy



Chapel's Infancy: DARPA HPCS (2003–2012)

- ~6–7 FTEs
- Research focus:
  - distinguish locality from parallelism
  - seamlessly mix data- and task-parallelism
  - support user-defined distributed arrays, parallel iterators
- CUG 2013 paper captured post-HPCS project status:

  The State of the Chapel Union

  Chamberlain, Choi, Dumler, Hildebrandt, Iten, Litvinov, Titus

Crossing the Stream of Adoption

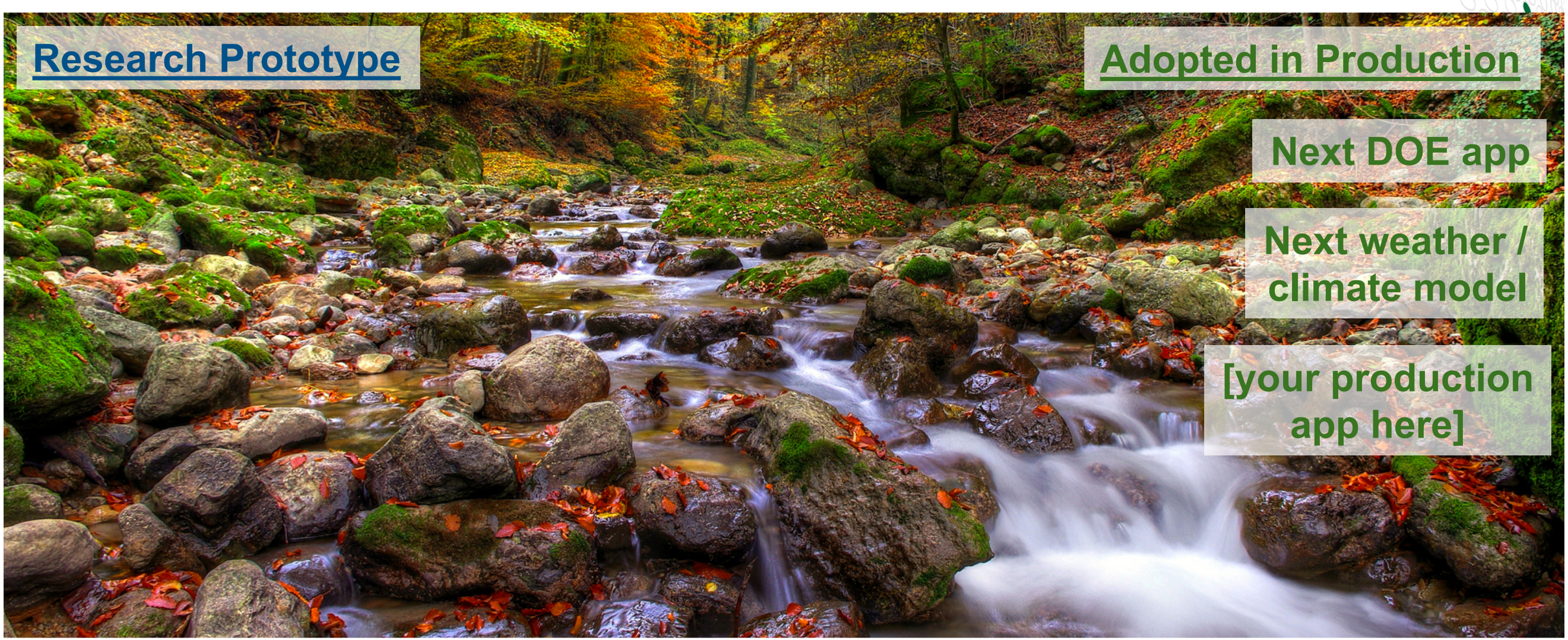


image credit: http://feelgrafix.com/813578-free-stream-wallpaper.html

Crossing the Stream of Adoption: Post-HPCS Barriers

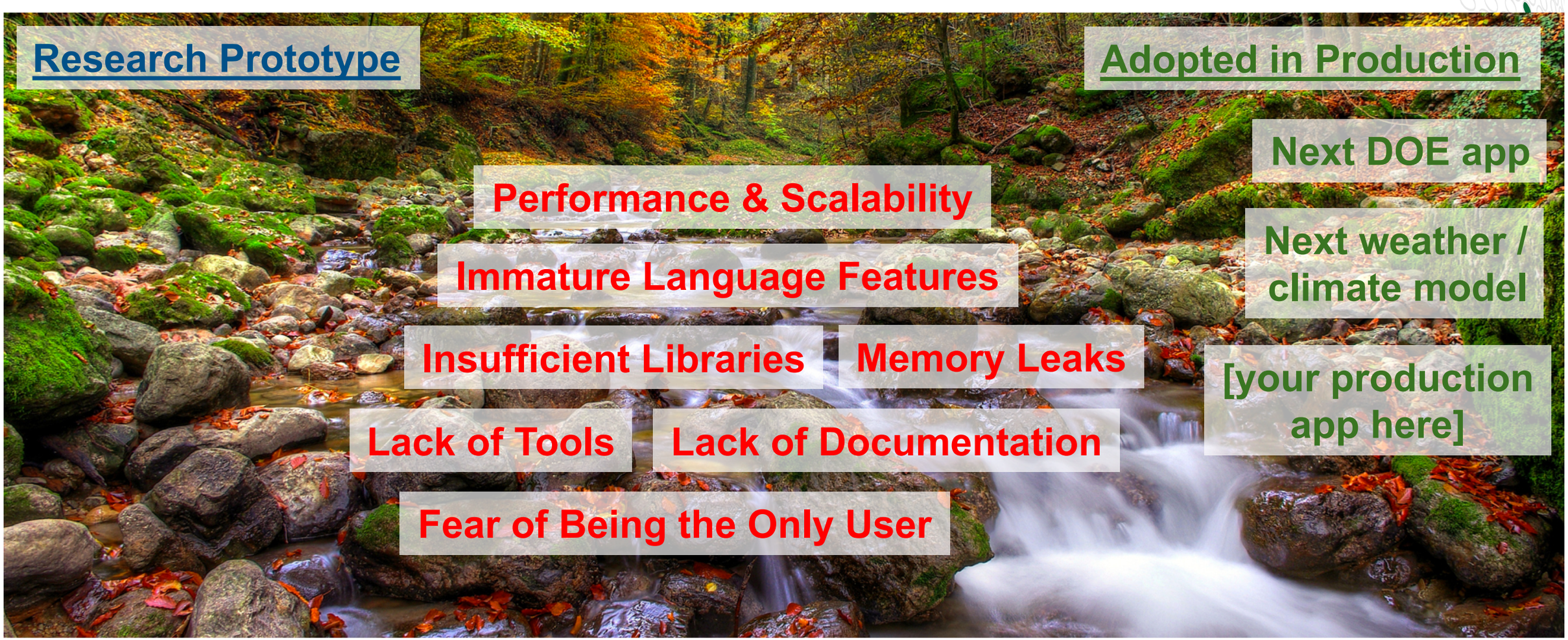
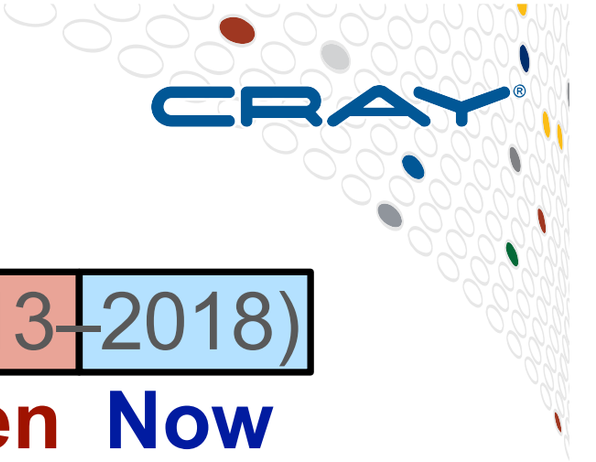


image credit: http://feelgrafix.com/813578-free-stream-wallpaper.html

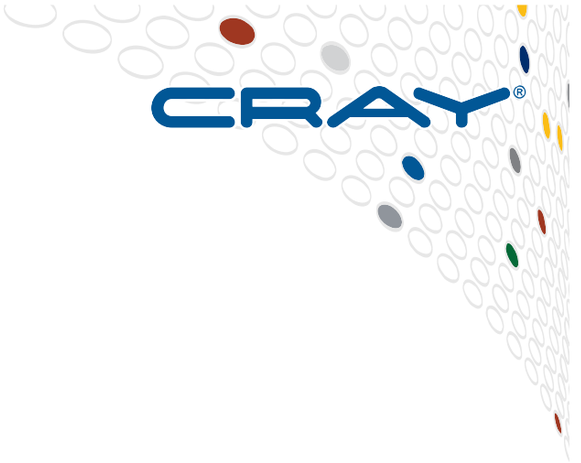
A Brief History of Chapel: Adolescence



Chapel's Adolescence: "the five-year push" (2013–2018)

Then **Now**

- Motivated by user enthusiasm for Chapel
- Development focus:
  - address weak points in HPCS prototype
  - support and grow the Chapel community
- ~13–14 FTEs
- This CUG 2018 talk & paper reports on progress during this time



Chapel Performance: Then vs. Now vs. Reference

Performance Focus Areas (during 5-year push)

Array Optimizations:

- shifted data optimization (eliminates arbitrary indexing overhead)
- loop-invariant code motion (eliminates meta-data overhead)
- eliminated multiply in indexing for 1D (and innermost dim of 2D+) arrays

Runtime Library Improvements:

- scalable parallel memory allocator
- tasks mapped to affinity aware user-level threads
- native/optimized comm with RDMA and limited software overhead

Optimized Communication:

- compiler locality analysis improvements
- bulk array assignments
- remote-value-forwarding, new distributions, fast-ons, …

Experimental Methodology

Methodology for the next several slides:

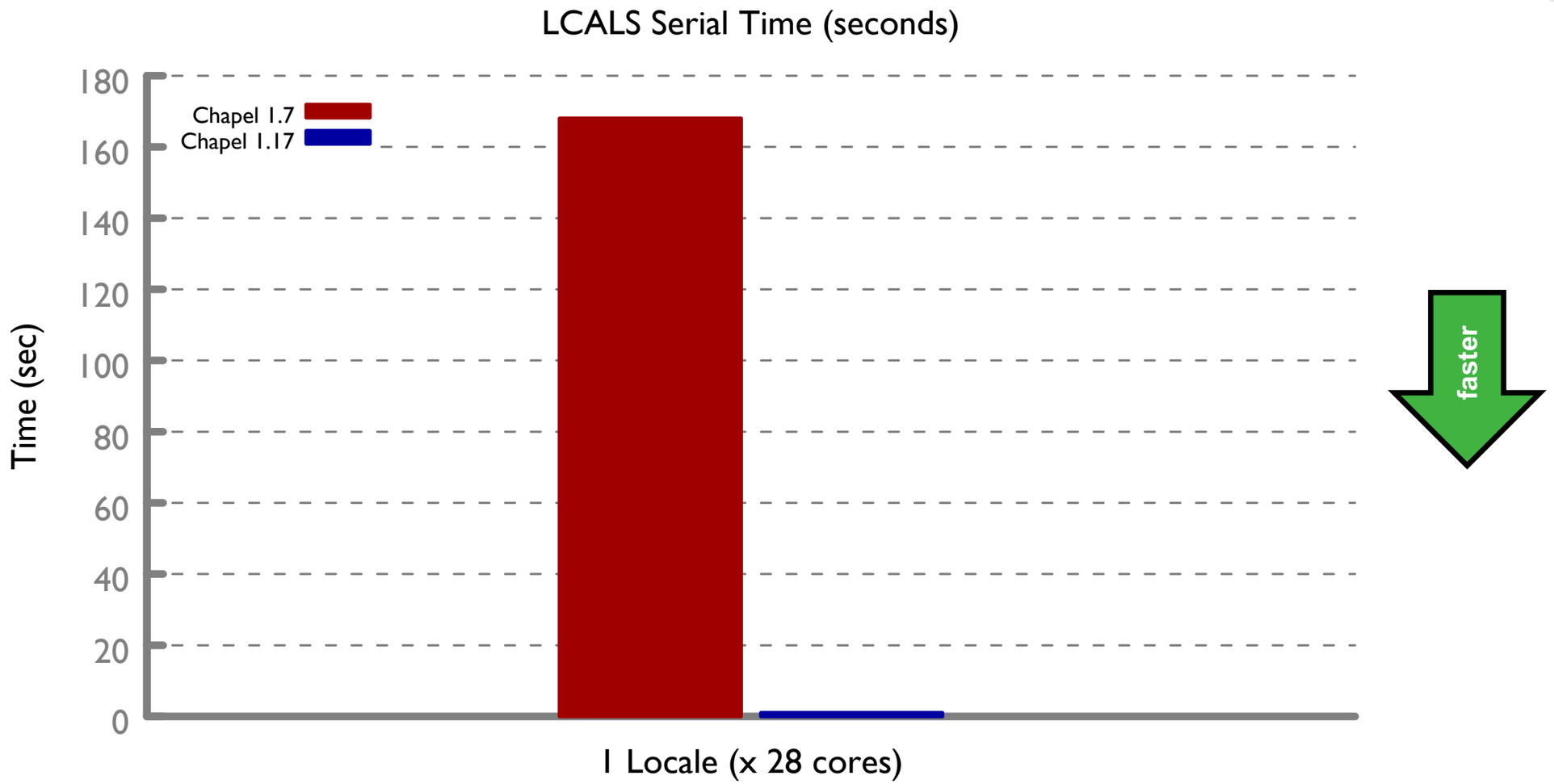
- Resurrected a copy of Chapel 1.7
  - updated it to build with current versions of gcc/g++
- Compared it to Chapel 1.17, released April 2018
- Used today's Cray systems
- Used today's benchmark codes
  - with modest edits for 1.7 in response to language changes

LCALS Serial Kernel

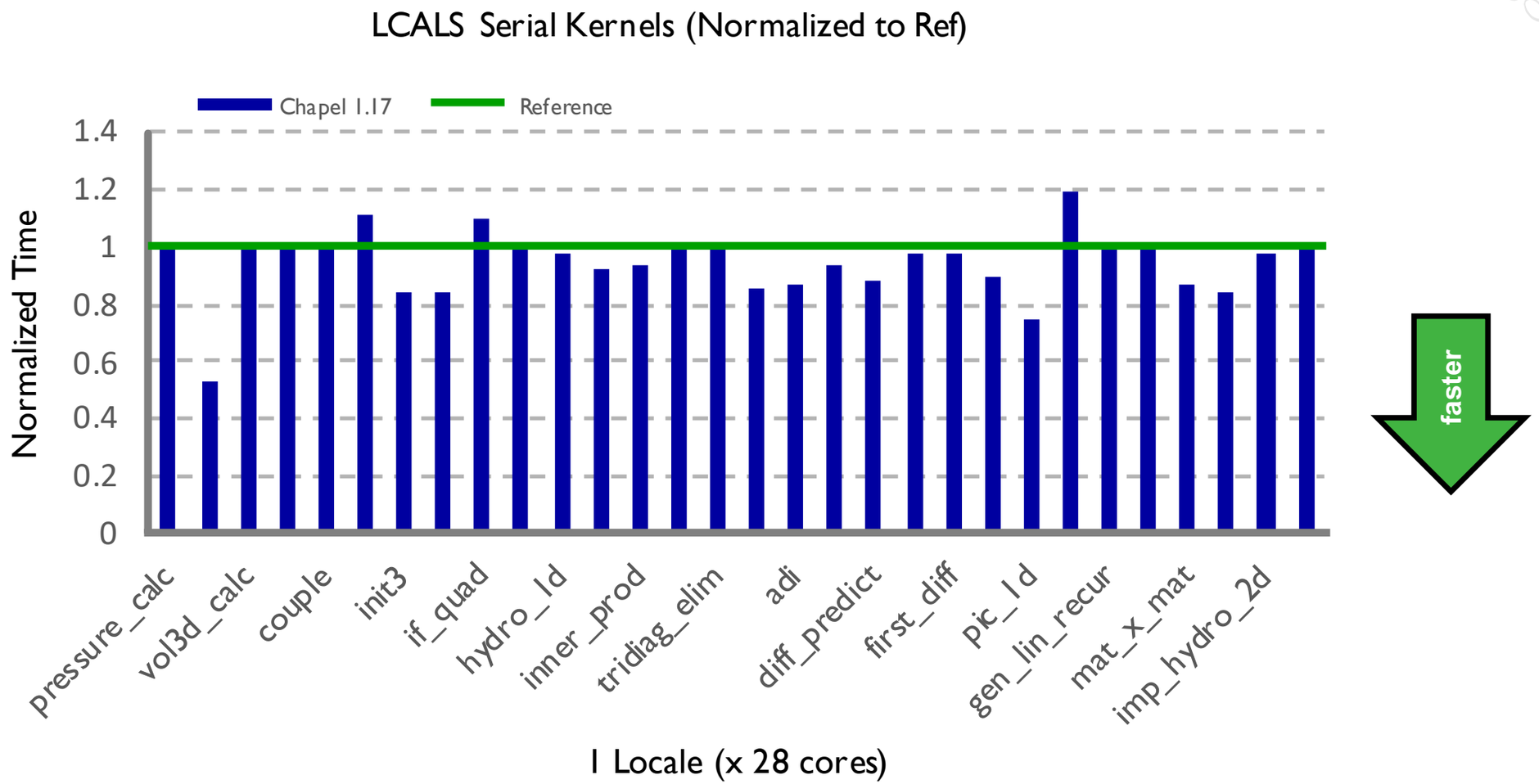
- **Chapel source:**

```
for i in 0..#len do
  bvc[i] = cls * (compression[i] + 1.0);
```

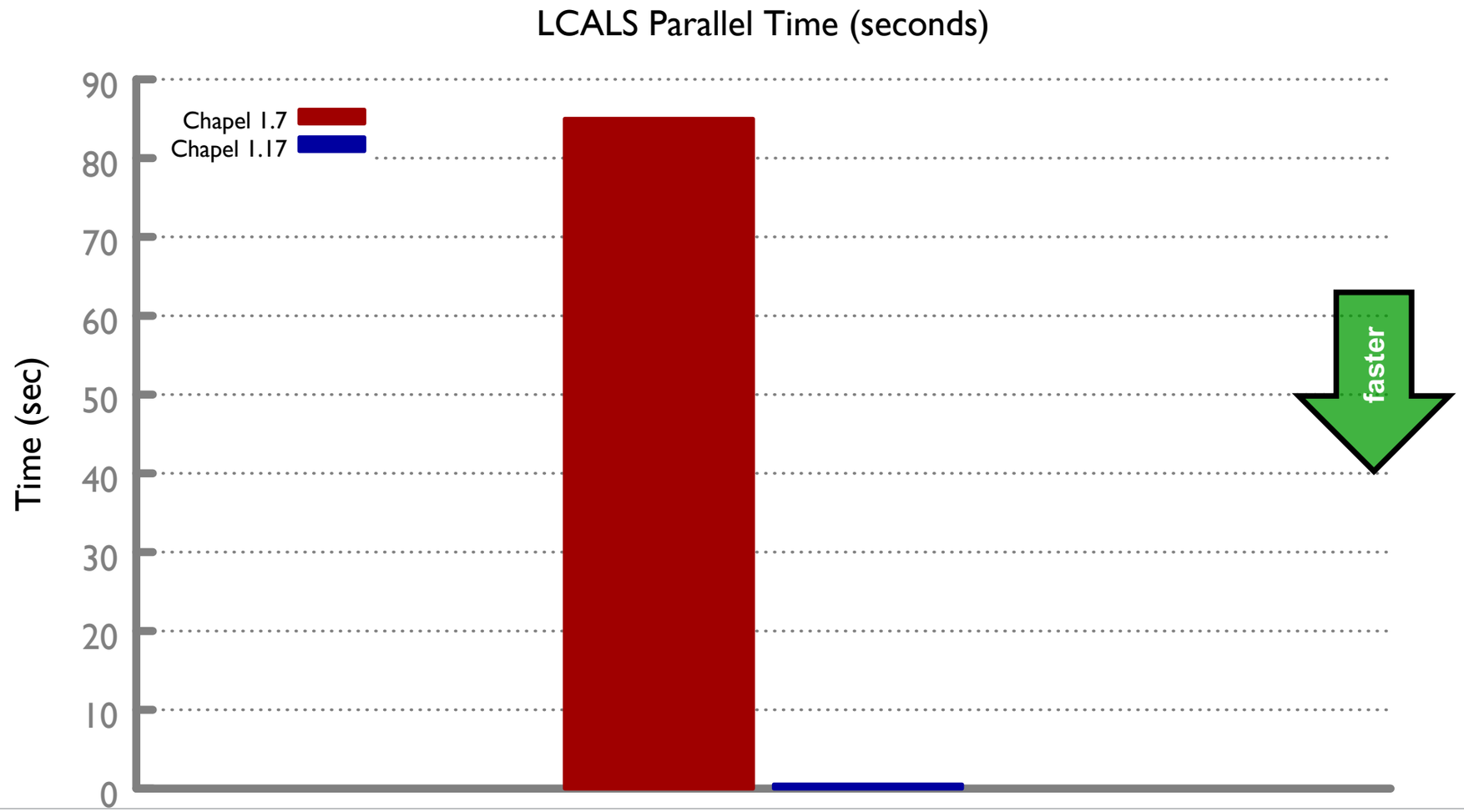
LCALS Serial Kernel: Chapel Then vs. Now



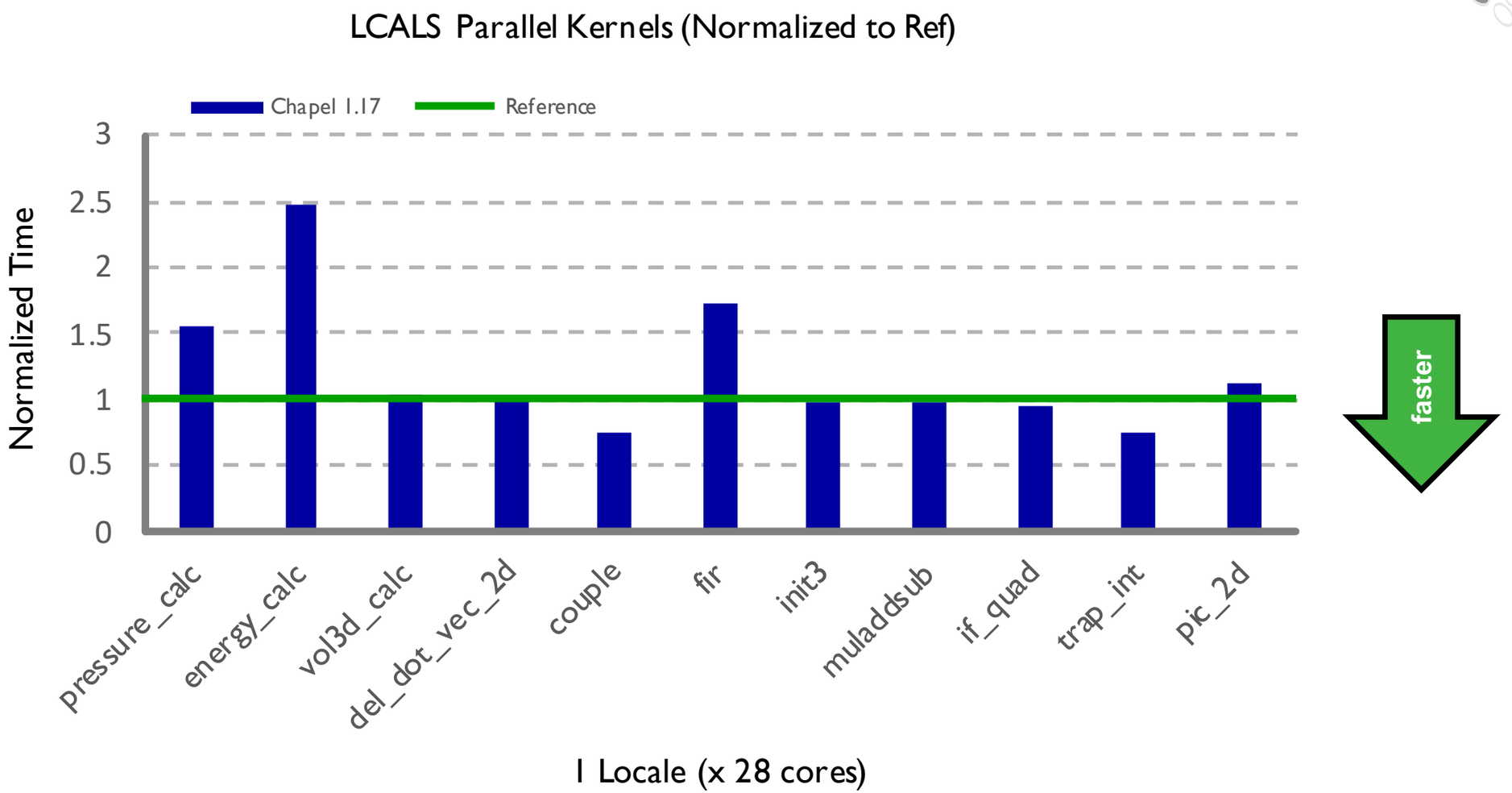
LCALS Serial Kernels: Chapel Now vs. Ref



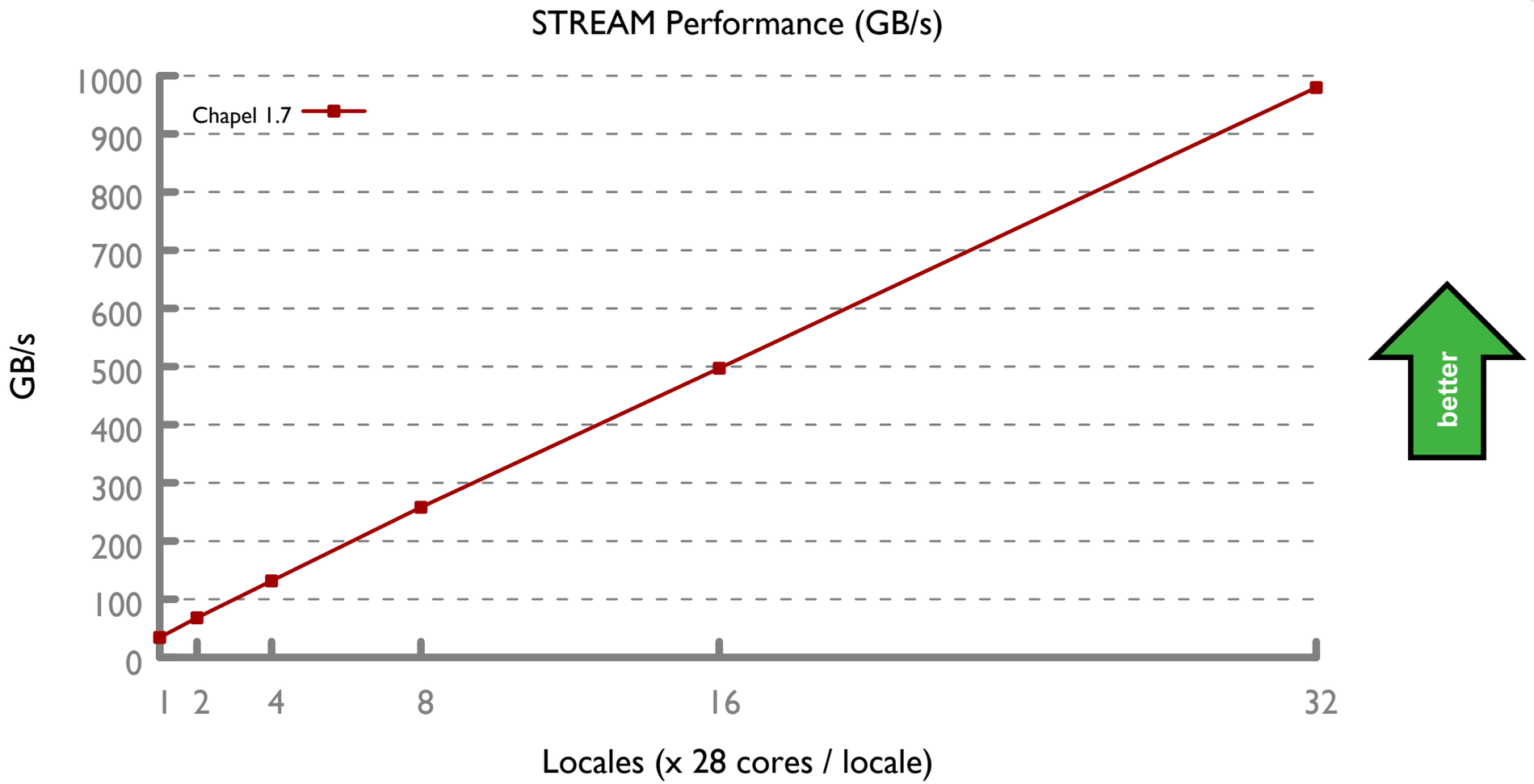
LCALS Parallel Kernel: Chapel Then vs. Now



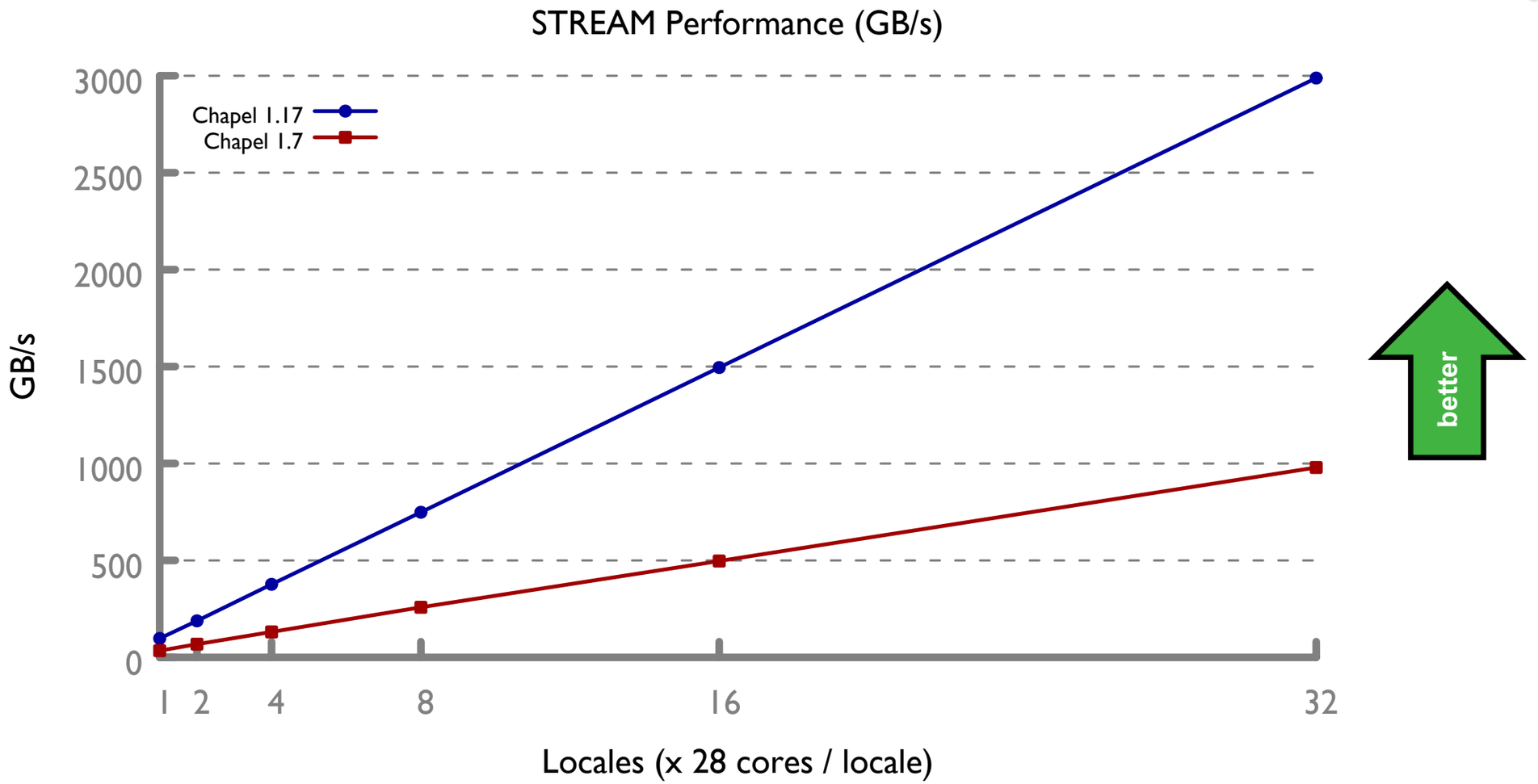
LCALS Parallel Kernels: Chapel Now vs. Ref



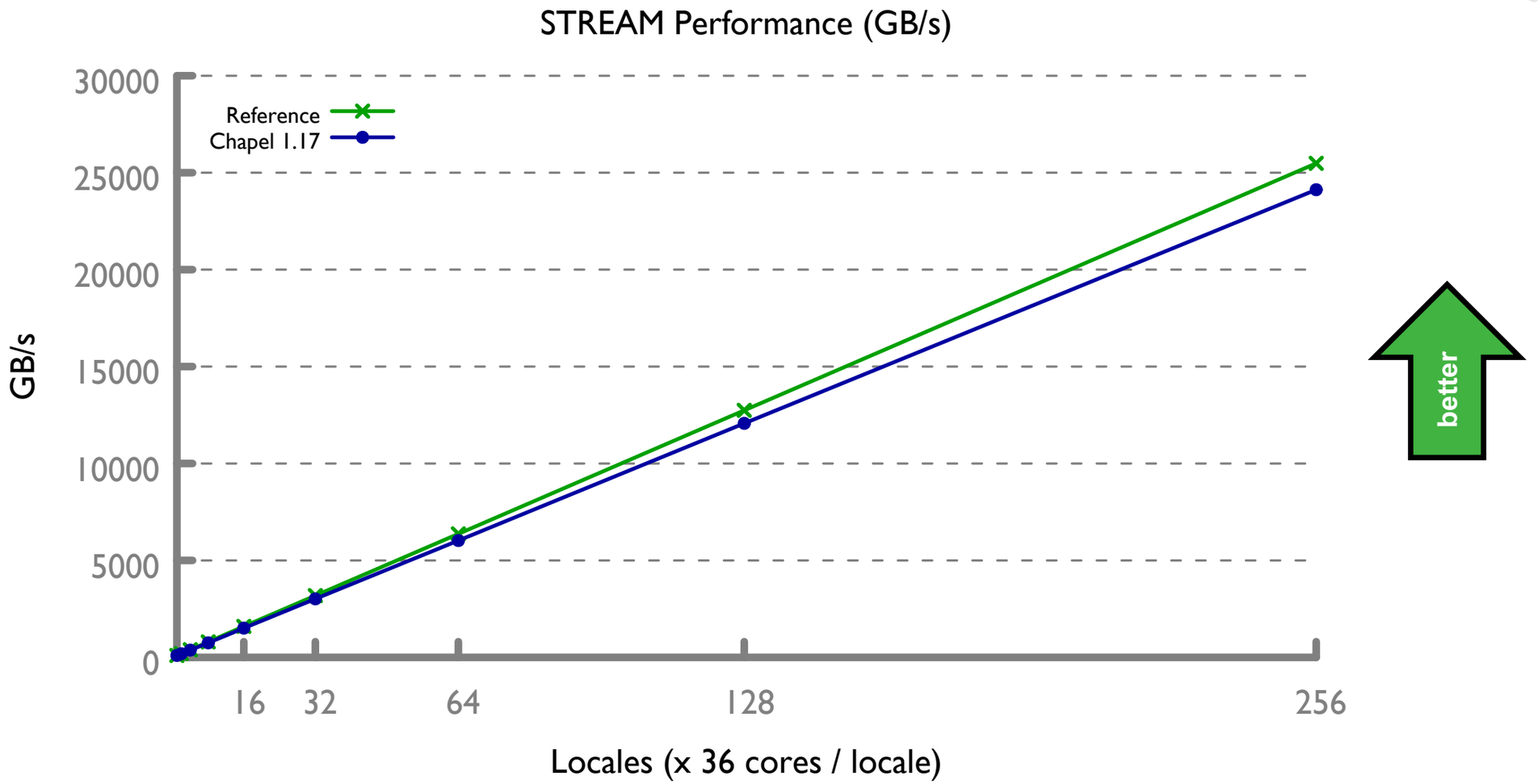
HPCC STREAM Triad: Chapel Then



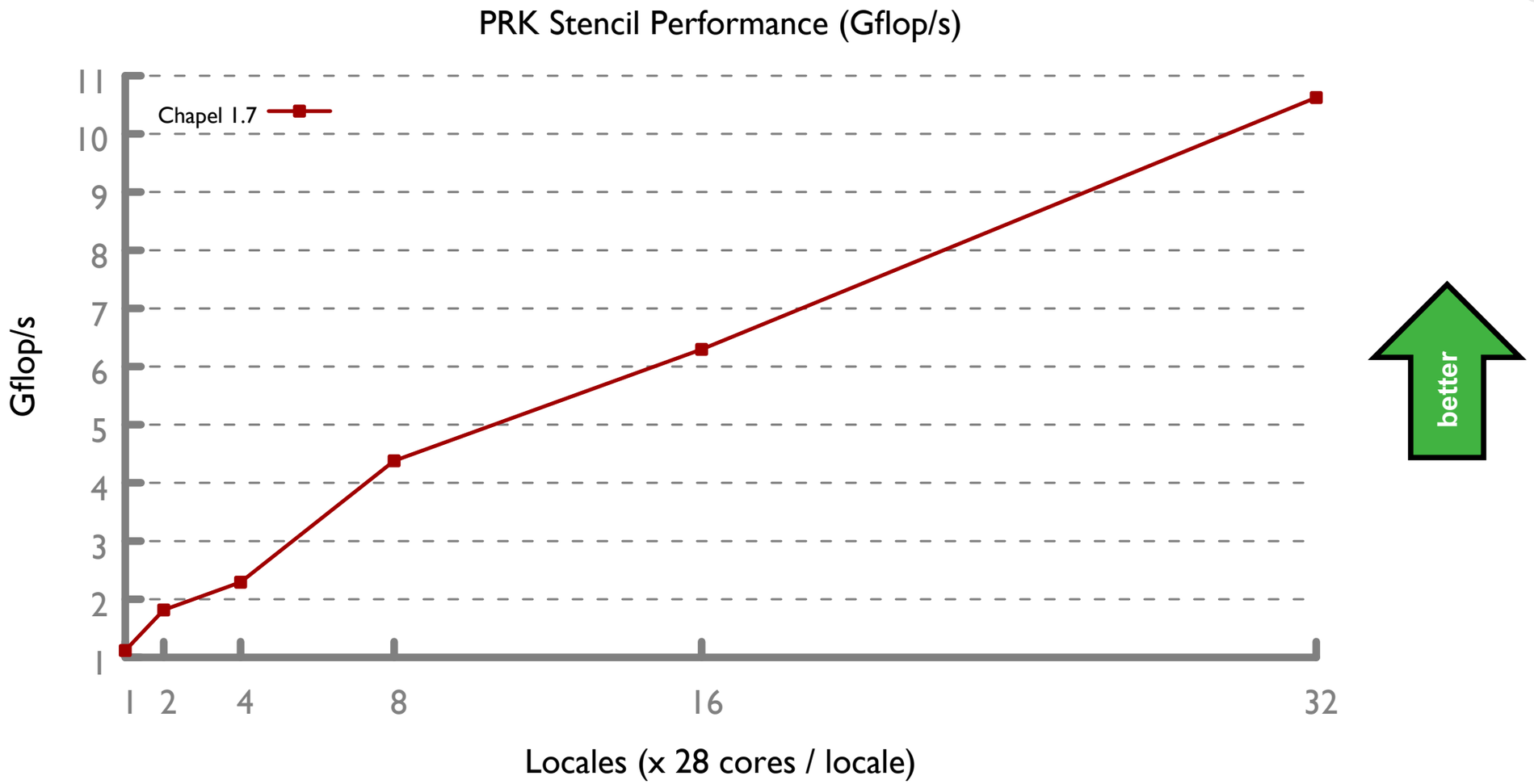
HPCC STREAM Triad: Chapel Then vs. Now



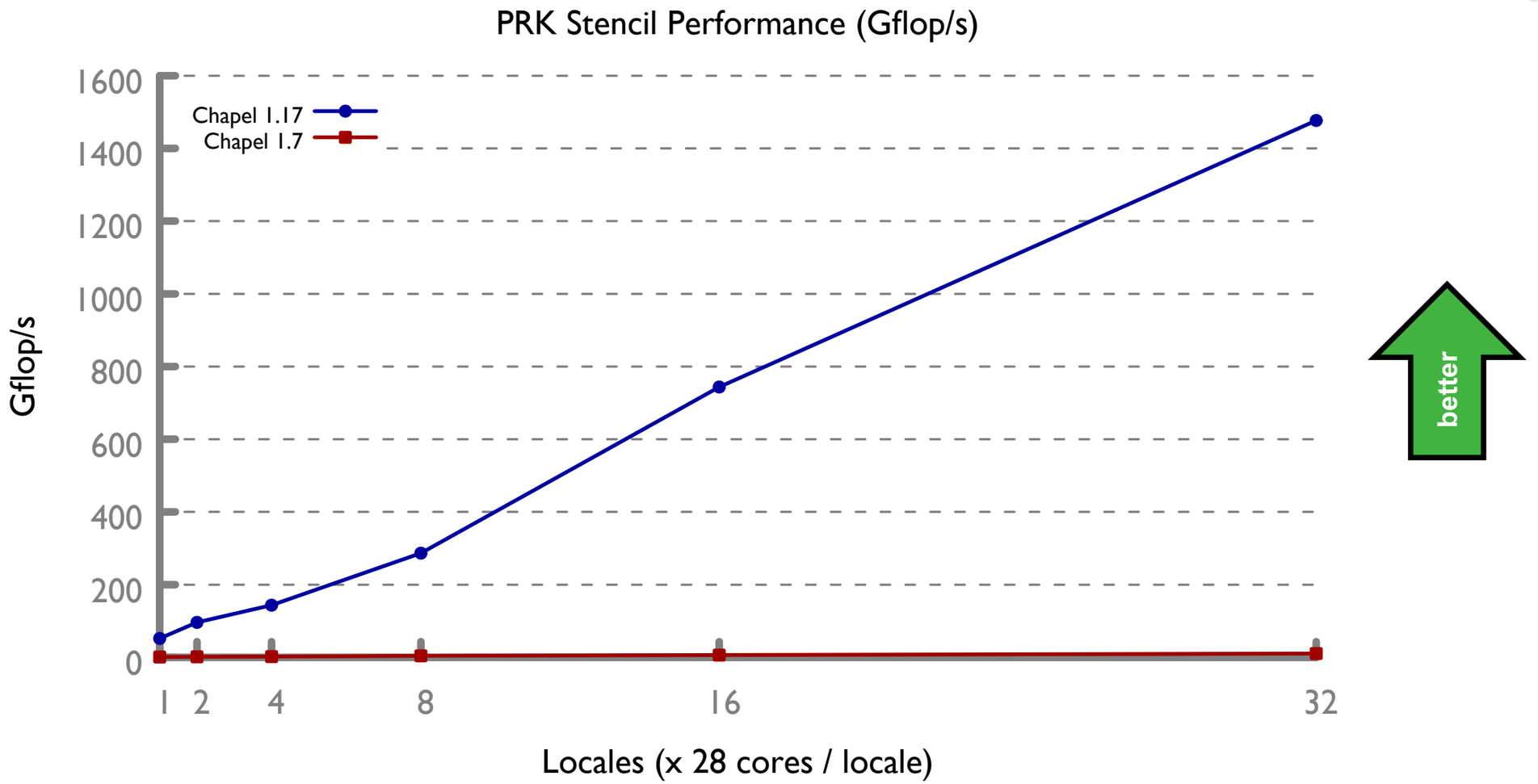
HPCC STREAM Triad: Chapel Now vs. Ref



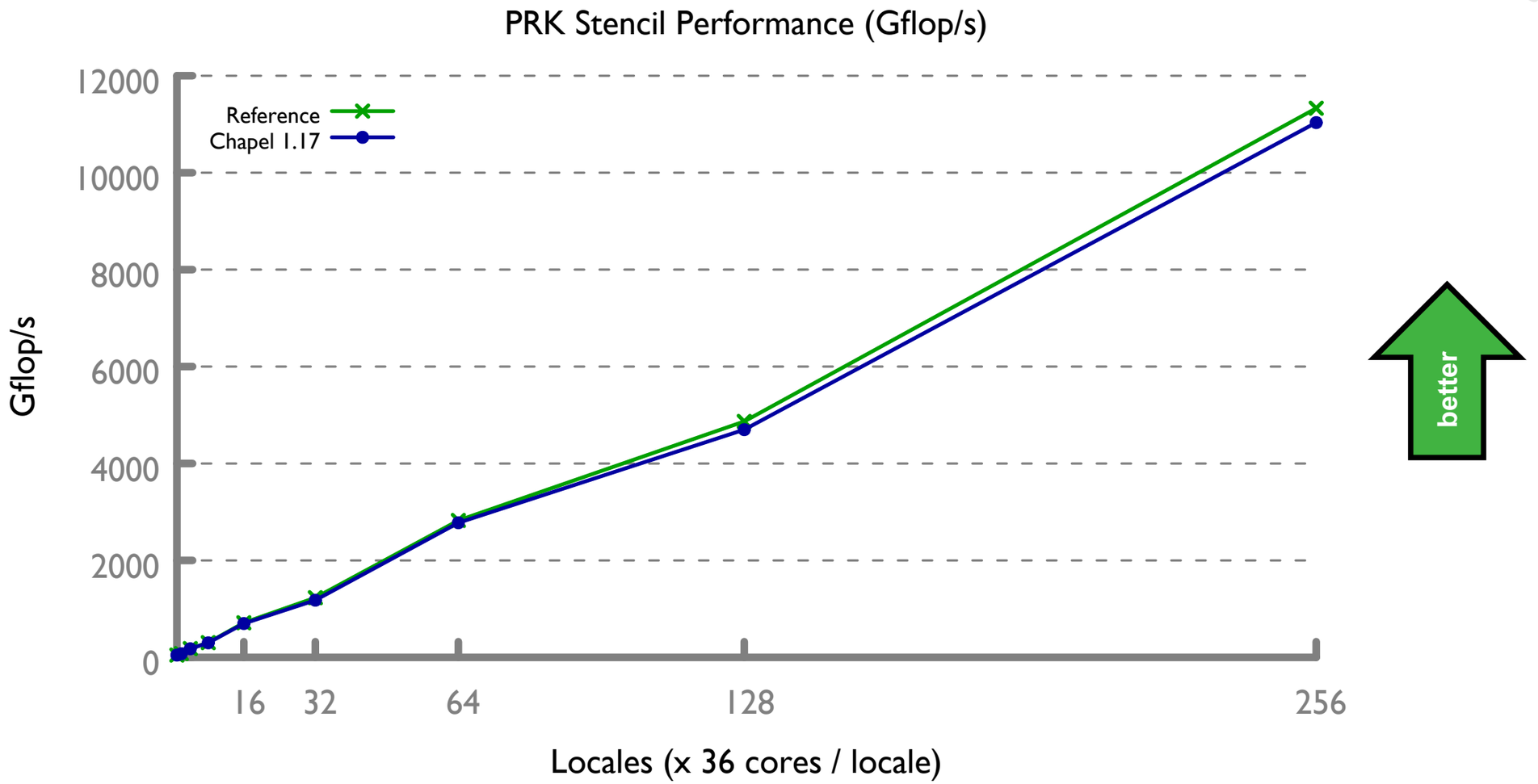
PRK Stencil: Chapel Then



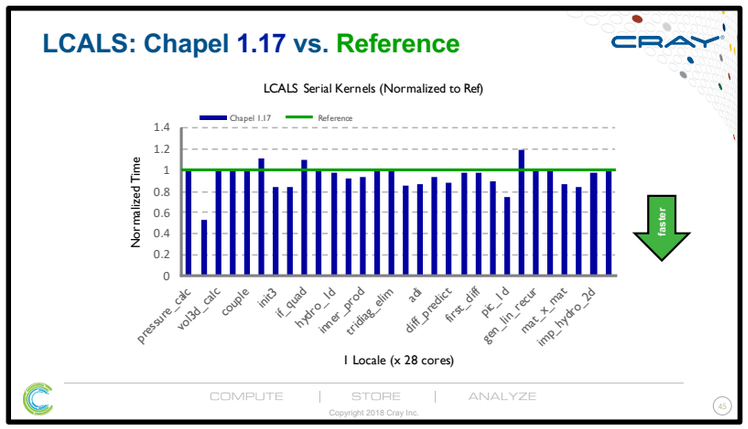
PRK Stencil: Chapel Then vs. Now



PRK Stencil: Chapel Now vs. Ref

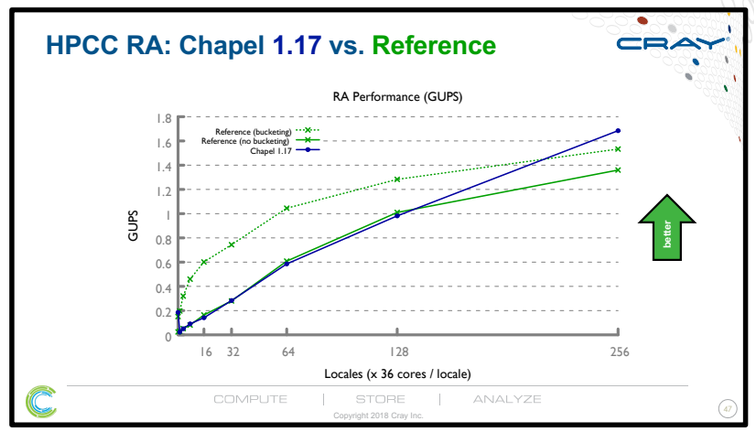


HPC Patterns: Chapel Now vs. reference



LCALS

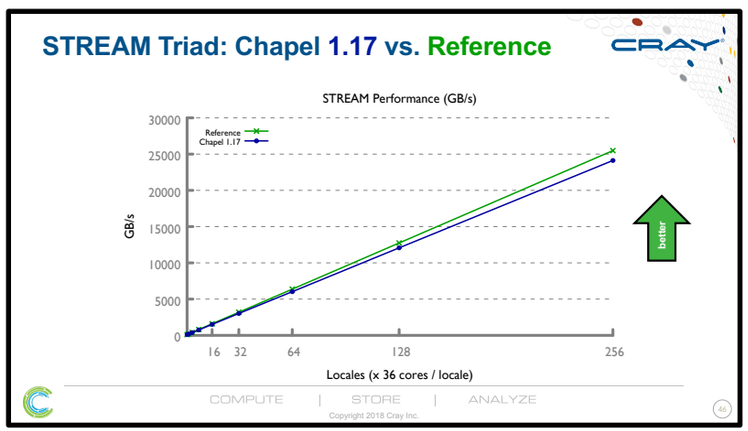
HPCC RA

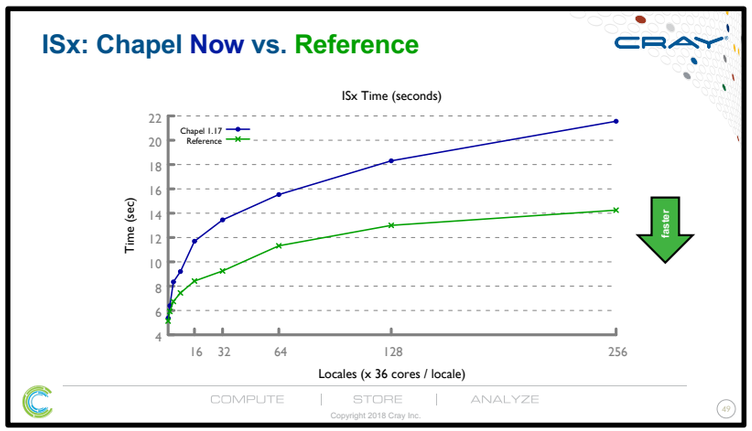


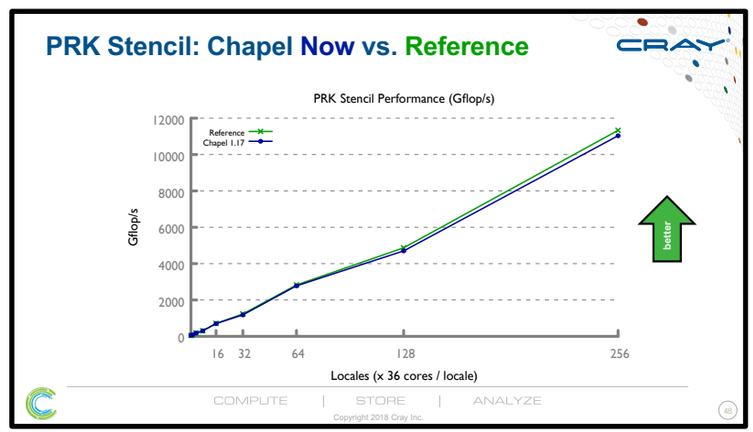
STREAM Triad

ISx

PRK Stencil

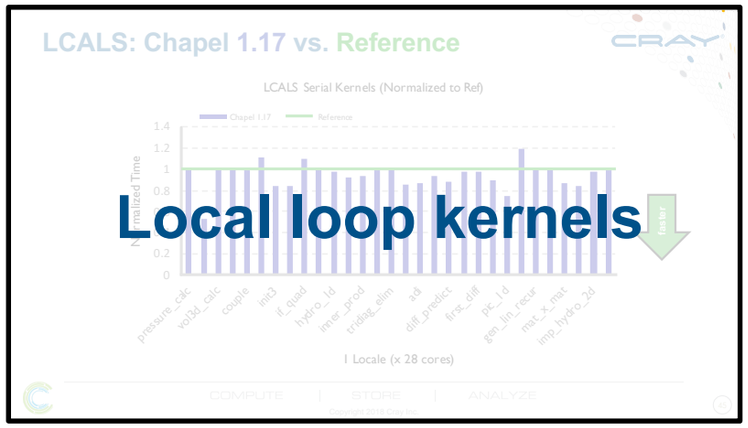






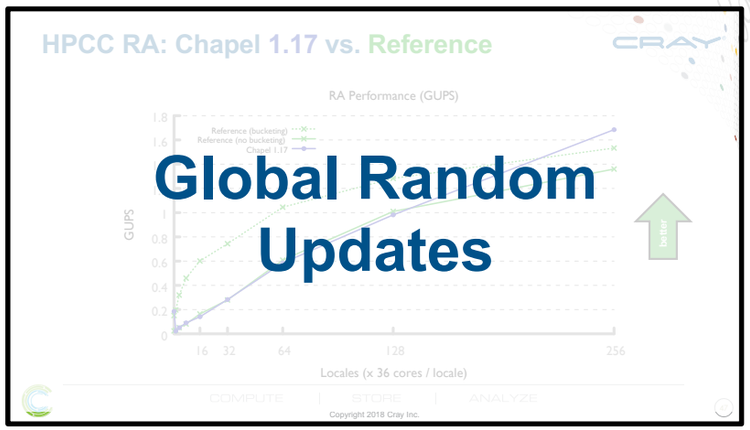
Nightly performance tickers online at:
https://chapel-lang.org/perf-nightly.html

HPC Patterns: Chapel Now vs. reference



LCALS

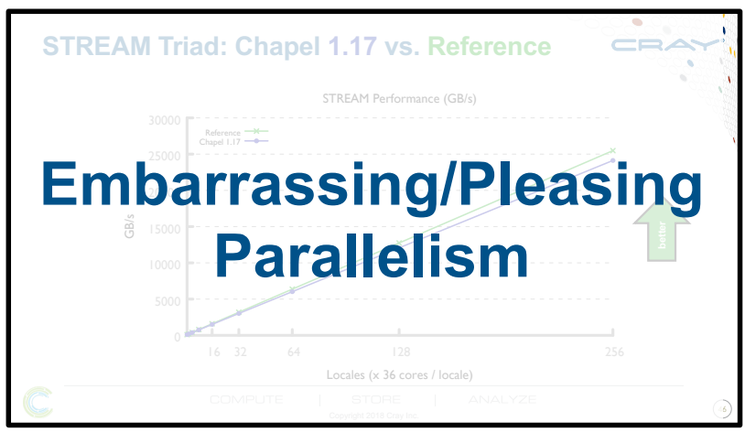
HPCC RA

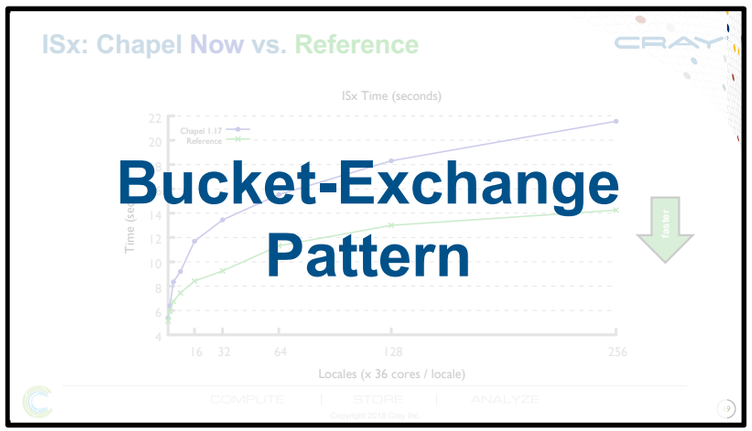


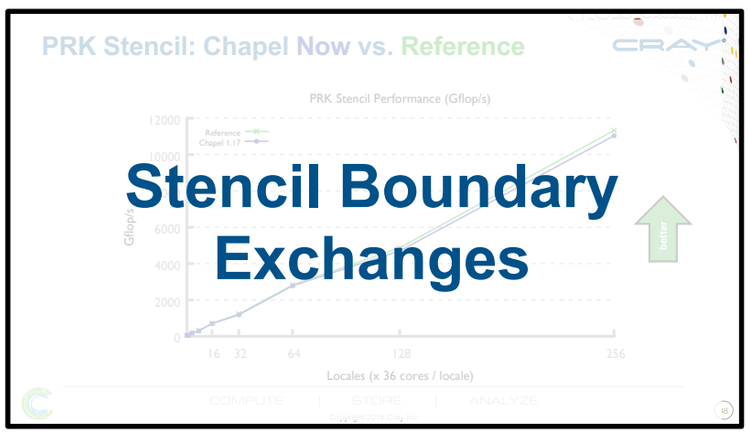
STREAM Triad

ISx

PRK Stencil

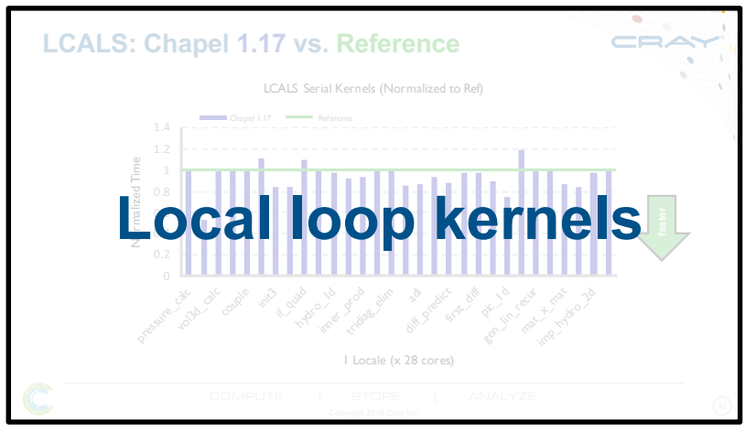






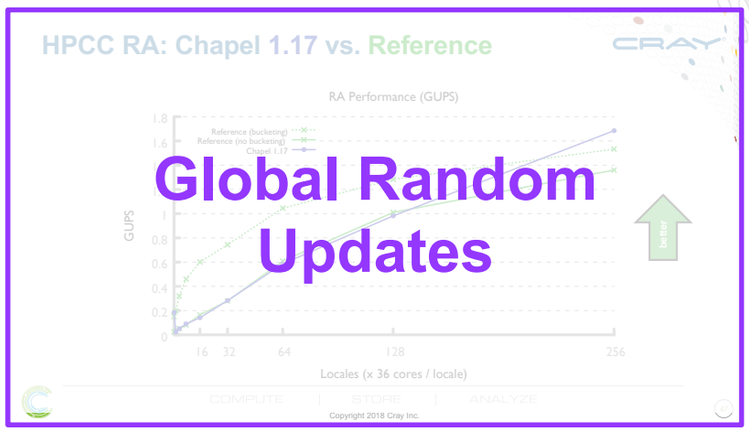
Nightly performance tickers online at:
https://chapel-lang.org/perf-nightly.html

HPC Patterns: Chapel Now vs. reference



LCALS

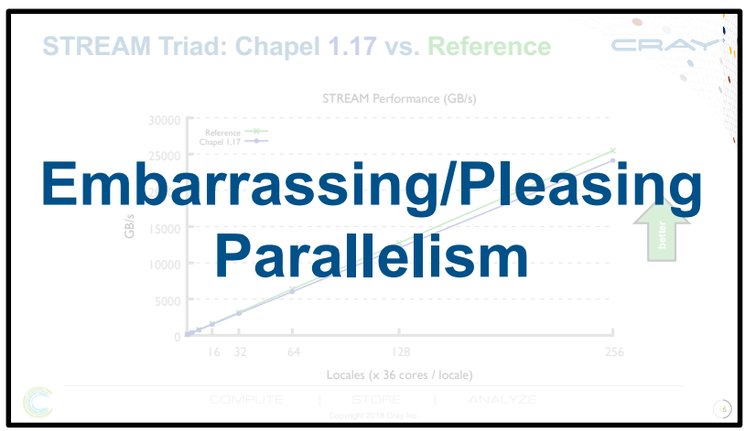
HPCC RA

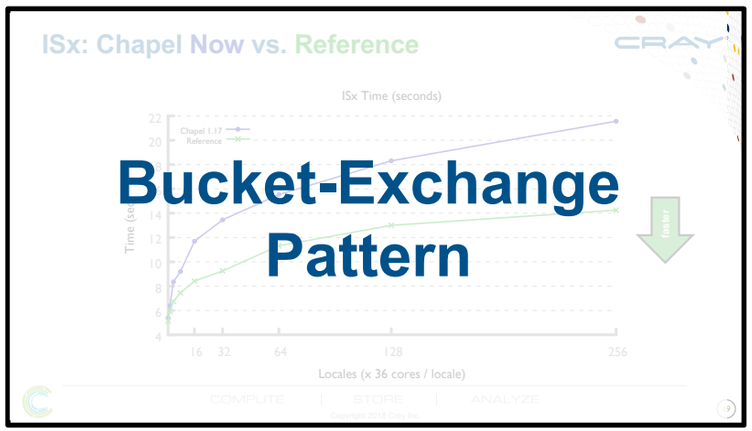


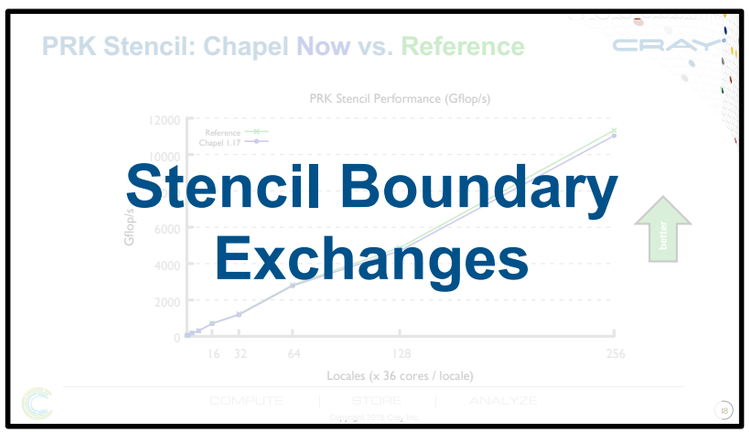
STREAM Triad

ISx

PRK Stencil





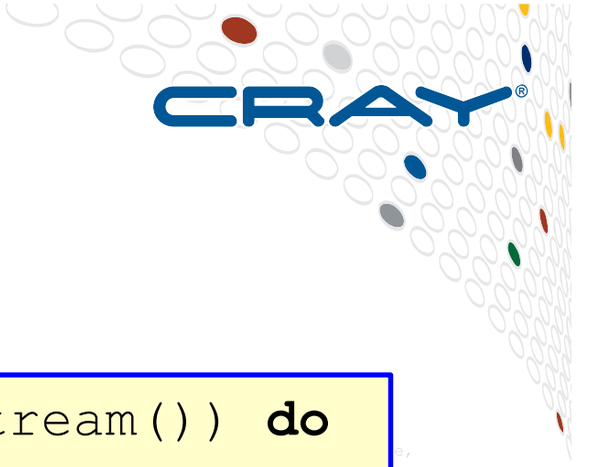


Nightly performance tickers online at:
https://chapel-lang.org/perf-nightly.html

HPCC Random Access Kernel: MPI

```
/* Perform updates to main table.  The scalar equivalent is:
 *
 *    for (i=0; i<NUPDATE; i++) {
 *      Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *      Table[Ran & (TABSIZE-1)] ^= Ran;
 *    }
 */

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
 while (i < SendCnt) {
   /* receive messages */
   do {
     MPI_Test(&inreq, &have_done, &status);
     if (have_done) {
       if (status.MPI_TAG == UPDATE_TAG) {
         MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
         bufferBase = 0;
         for (j=0; j < recvUpdates; j ++) {
           inmsg = LocalRecvBuffer[bufferBase+j];
           LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                         tparams.GlobalStartMyProc;
           HPCC_Table[LocalOffset] ^= inmsg;
         }
       } else if (status.MPI_TAG == FINISHED_TAG) {
         NumberReceiving--;
       } else
         MPI_Abort( MPI_COMM_WORLD, -1 );
       MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                 MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
     }
   } while (have_done && NumberReceiving > 0);
   if (pendingUpdates < maxPendingUpdates) {
     Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B ? POLY : ZERO64B);
     GlobalOffset = Ran & (tparams.TableSize-1);
     if ( GlobalOffset < tparams.Top)
       WhichPe = ( GlobalOffset / (tparams.MinLocalTableSize + 1) );
     else
       WhichPe = ( (GlobalOffset - tparams.Remainder) /
                 tparams.MinLocalTableSize );
     if (WhichPe == tparams.MyProc) {
       LocalOffset = (Ran & (tparams.TableSize - 1)) -
                     tparams.GlobalStartMyProc;
       HPCC_Table[LocalOffset] ^= Ran;
   }  else {
        HPCC_InsertUpdate(Ran, WhichPe, Buckets);
        pendingUpdates++;
      }
      i++;
    }
    else {
      MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
      if (have_done) {
        outreq = MPI_REQUEST_NULL;
        pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                             &peUpdates);
        MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
                  UPDATE_TAG, MPI_COMM_WORLD, &outreq);
        pendingUpdates -= peUpdates;
      }
    }
  }
  /* send remaining updates in buckets */
  while (pendingUpdates > 0) {
    /* receive messages */
    do {
      MPI_Test(&inreq, &have_done, &status);
      if (have_done) {
        if (status.MPI_TAG == UPDATE_TAG) {
          MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
          bufferBase = 0;
          for (j=0; j < recvUpdates; j ++) {
            inmsg = LocalRecvBuffer[bufferBase+j];
            LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                          tparams.GlobalStartMyProc;
            HPCC_Table[LocalOffset] ^= inmsg;
          }
        } else if (status.MPI_TAG == FINISHED_TAG) {
          /* we got a done message.  Thanks for playing... */
          NumberReceiving--;
        } else {
          MPI_Abort( MPI_COMM_WORLD, -1 );
        }
        MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                  MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
      }
    } while (have_done && NumberReceiving > 0);
    MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
    if (have_done) {
      outreq = MPI_REQUEST_NULL;
      pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                           &peUpdates);
      MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
                UPDATE_TAG, MPI_COMM_WORLD, &outreq);
      pendingUpdates -= peUpdates;
    }
  }
  /* send our done messages */
  for (proc_count = 0 ; proc_count < tparams.NumProcs ; ++proc_count) {
    if (proc_count == tparams.MyProc) { tparams.finish_req[tparams.MyProc] =
                                        MPI_REQUEST_NULL; continue; }
    /* send garbage - who cares, no one will look at it */
    MPI_Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,
              MPI_COMM_WORLD, tparams.finish_req + proc_count);
  }
  /* Finish everyone else up... */
  while (NumberReceiving > 0) {
    MPI_Wait(&inreq, &status);
    if (status.MPI_TAG == UPDATE_TAG) {
      MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
      bufferBase = 0;
      for (j=0; j < recvUpdates; j ++) {
        inmsg = LocalRecvBuffer[bufferBase+j];
        LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                      tparams.GlobalStartMyProc;
        HPCC_Table[LocalOffset] ^= inmsg;
      }
    } else if (status.MPI_TAG == FINISHED_TAG) {
      /* we got a done message.  Thanks for playing... */
      NumberReceiving--;
    } else {
      MPI_Abort( MPI_COMM_WORLD, -1 );
    }
    MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
              MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
  }

  MPI_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
```

HPCC Random Access Kernel: MPI



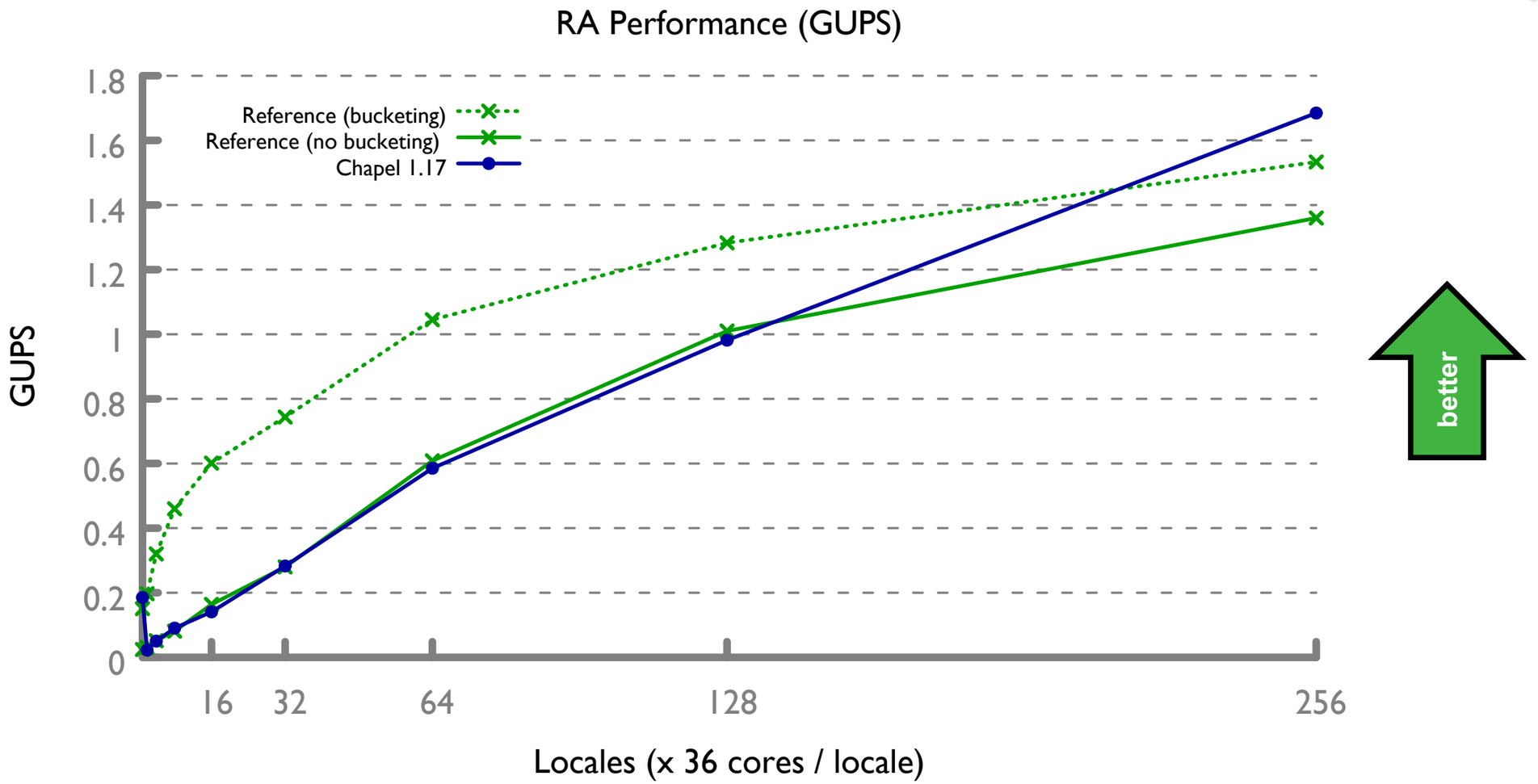
Chapel Kernel

```
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask] ^= r;
```

MPI Comment

```
/* Perform updates to main table. The scalar equivalent is:
 *
 *     for (i=0; i<NUPDATE; i++) {
 *       Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *       Table[Ran & (TABSIZE-1)] ^= Ran;
 *     }
 */
```

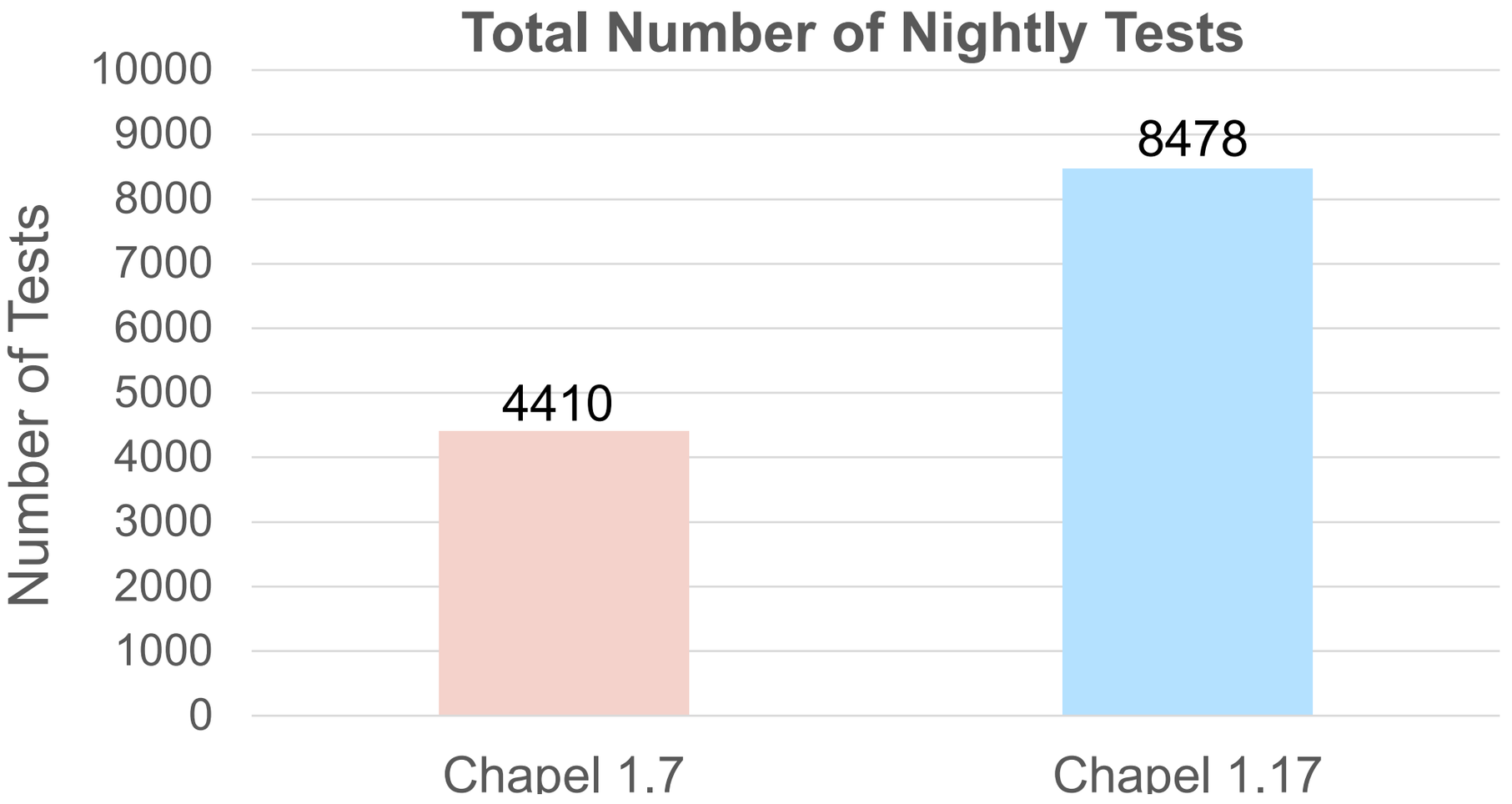
HPCC RA: Chapel Now vs. Ref



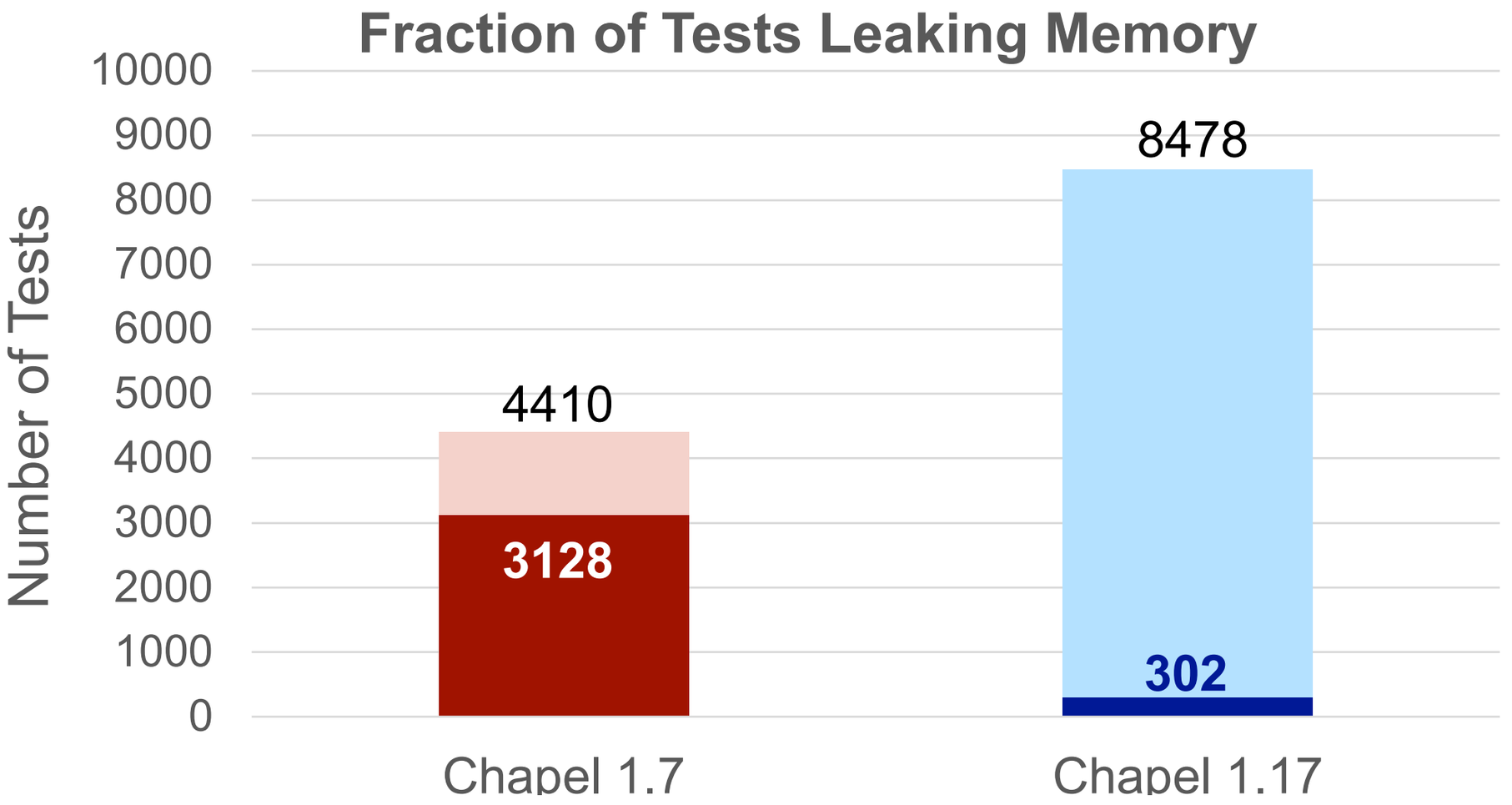
Memory Leaks: Then vs. Now

(skipped at CUG due to time constraints)

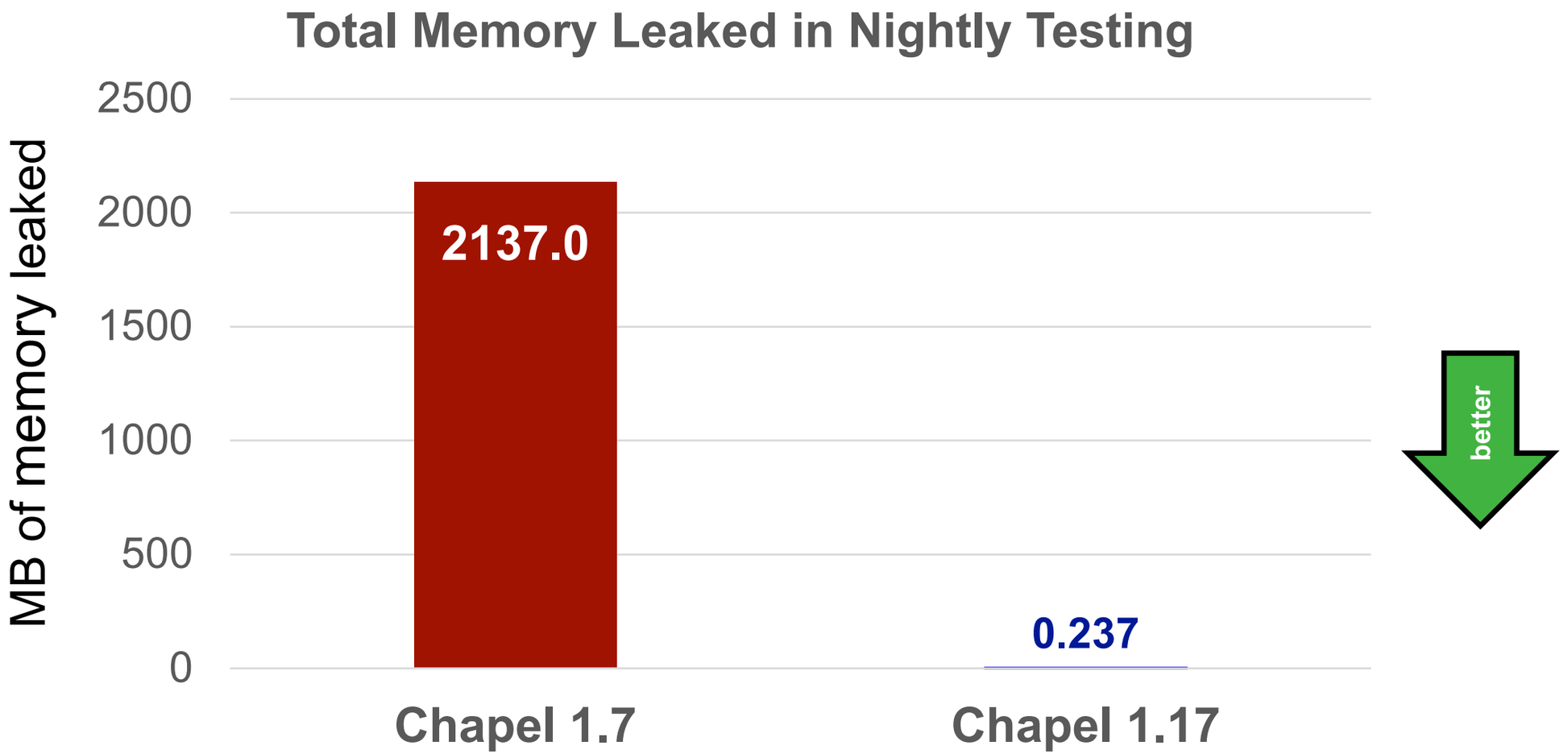
Memory Leaks: Chapel Then vs. Now



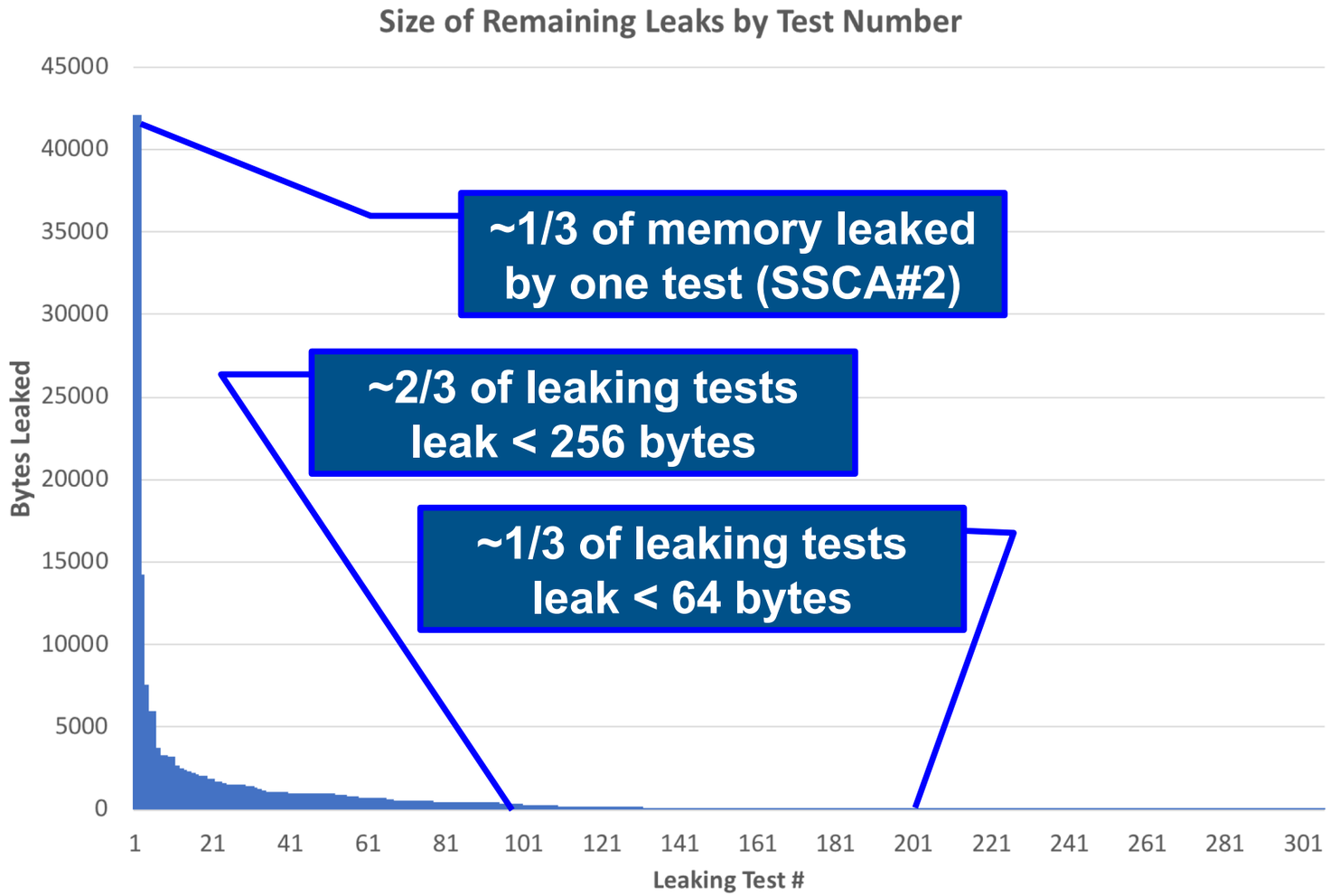
Memory Leaks: Chapel Then vs. Now

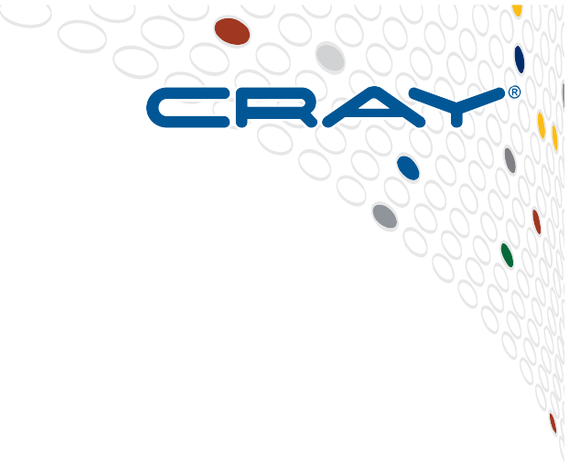


Memory Leaks: Chapel Then vs. Now



Memory Leaks: Remaining Leaks





Chapel Language: Then vs. Now

Language: Then

Parallelism and Locality: Generally in good shape

- not many changes here since HPCS

Base Language: Left much to be desired

- lots of focus here since HPCS

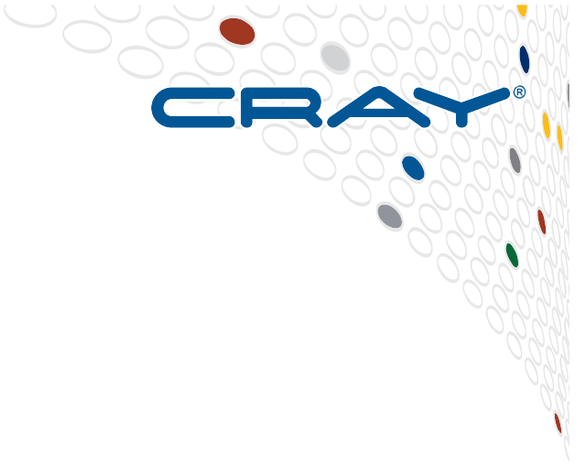
Language: Now

Parallelism and Locality

- introduced *task intents* to reduce chances of race conditions
- and *user-defined locale models* to support new node architectures

Base Language

- fixed a number of problems with object-oriented programming
  - **records:** poor memory management discipline
  - **classes:** problems with generic classes, class hierarchies
- made strings usable
- added error-handling features
- made namespace improvements (and much more…)

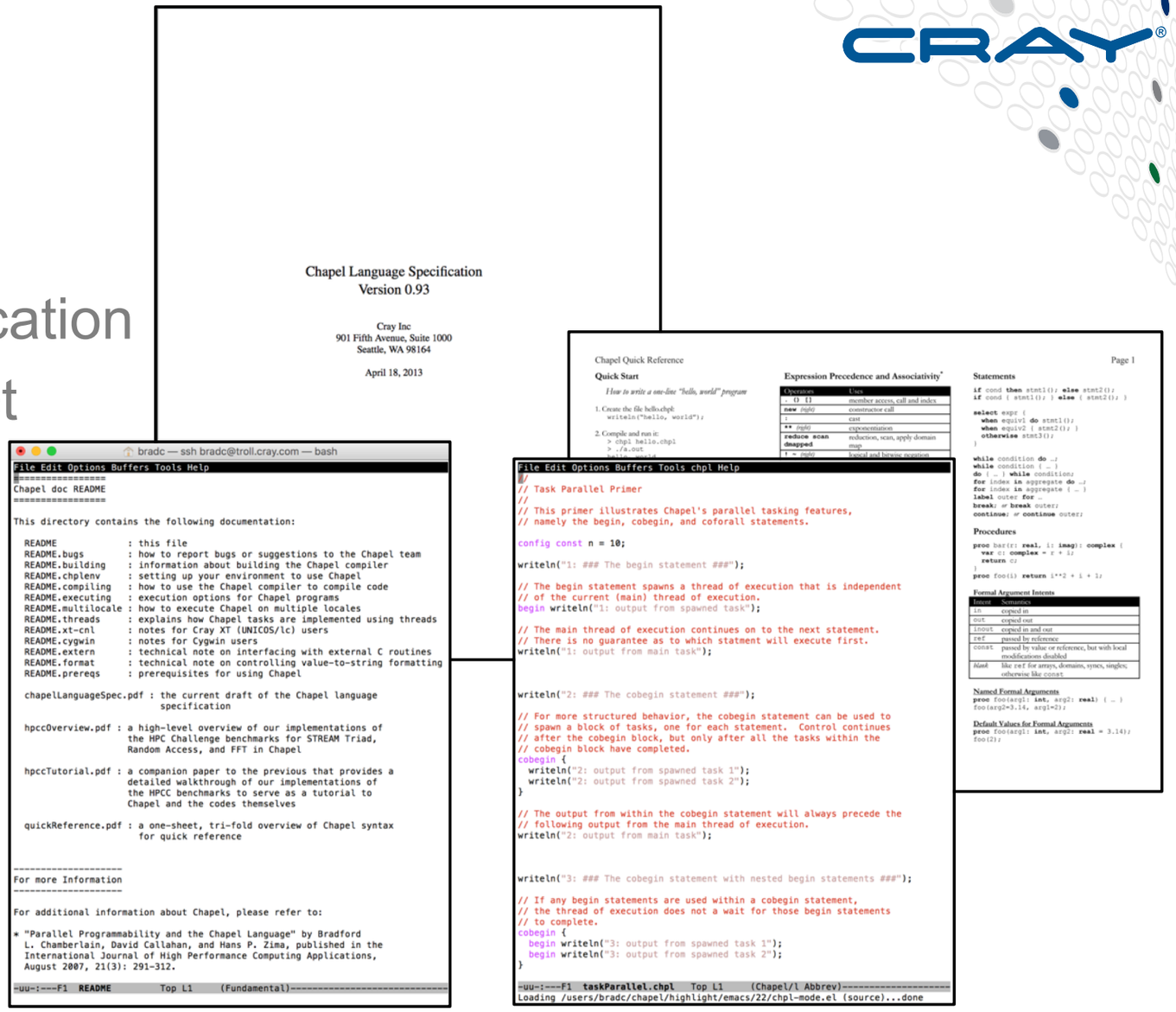


Chapel Ecosystem: Then vs. Now

Documentation: Then

After HPCS:

- a PDF language specification
- a Quick Reference sheet
- a number of READMEs
- ~22 primer examples



Chapel Language Specification
Version 0.93

Cray Inc
901 Fifth Avenue, Suite 1000
Seattle, WA 98164

April 18, 2013

bradc — ssh bradc@troll.cray.com — bash

```
File Edit Options Buffers Tools Help
================
Chapel doc README
================

This directory contains the following documentation:

  README             : this file
  README.bugs        : how to report bugs or suggestions to the Chapel team
  README.building    : information about building the Chapel compiler
  README.chplenv     : setting up your environment to use Chapel
  README.compiling   : how to use the Chapel compiler to compile code
  README.executing   : execution options for Chapel programs
  README.multilocale : how to execute Chapel on multiple locales
  README.threads     : explains how Chapel tasks are implemented using threads
  README.xt-cnl      : notes for Cray XT (UNICOS/lc) users
  README.cygwin      : notes for Cygwin users
  README.extern      : technical note on interfacing with external C routines
  README.format      : technical note on controlling value-to-string formatting
  README.prereqs     : prerequisites for using Chapel

  chapelLanguageSpec.pdf : the current draft of the Chapel language
                           specification

  hpccOverview.pdf : a high-level overview of our implementations of
                     the HPC Challenge benchmarks for STREAM Triad,
                     Random Access, and FFT in Chapel

  hpccTutorial.pdf : a companion paper to the previous that provides a
                     detailed walkthrough of our implementations of
                     the HPCC benchmarks to serve as a tutorial to
                     Chapel and the codes themselves

  quickReference.pdf : a one-sheet, tri-fold overview of Chapel syntax
                       for quick reference


--------------------
For more Information
--------------------

For additional information about Chapel, please refer to:

* "Parallel Programmability and the Chapel Language" by Bradford
  L. Chamberlain, David Callahan, and Hans P. Zima, published in the
  International Journal of High Performance Computing Applications,
  August 2007, 21(3): 291-312.

-uu-:---F1  README         Top L1     (Fundamental)----------------------------
```

Chapel Quick Reference — Page 1

Quick Start

How to write a one-line "hello, world" program

1. Create the file hello.chpl:
```
writeln("hello, world");
```
2. Compile and run it:
```
> chpl hello.chpl
> ./a.out
```

Expression Precedence and Associativity*

Operators	Uses
. () []	member access, call and index
new *(right)*	constructor call
:	cast
** *(right)*	exponentiation
reduce scan	reduction, scan, apply domain
dmapped	map
! ~ *(right)*	logical and bitwise negation

Statements

```
if cond then stmt1(); else stmt2();
if cond { stmt1(); } else { stmt2(); }

select expr {
  when equiv1 do stmt1();
  when equiv2 { stmt2(); }
  otherwise stmt3();
}

while condition do …;
while condition { … }
do { … } while condition;
for index in aggregate do …;
for index in aggregate { … }
label outer for …
break; or break outer;
continue; or continue outer;
```

Procedures

```
proc bar(r: real, i: imag): complex {
  var c: complex = r + i;
  return c;
}
proc foo(i) return i**2 + i + 1;
```

Formal Argument Intents

Intent	Semantics
in	copied in
out	copied out
inout	copied in and out
ref	passed by reference
const	passed by value or reference, but with local modifications disabled
blank	like ref for arrays, domains, syncs, singles; otherwise like const

Named Formal Arguments

```
proc foo(arg1: int, arg2: real) { … }
foo(arg2=3.14, arg1=2);
```

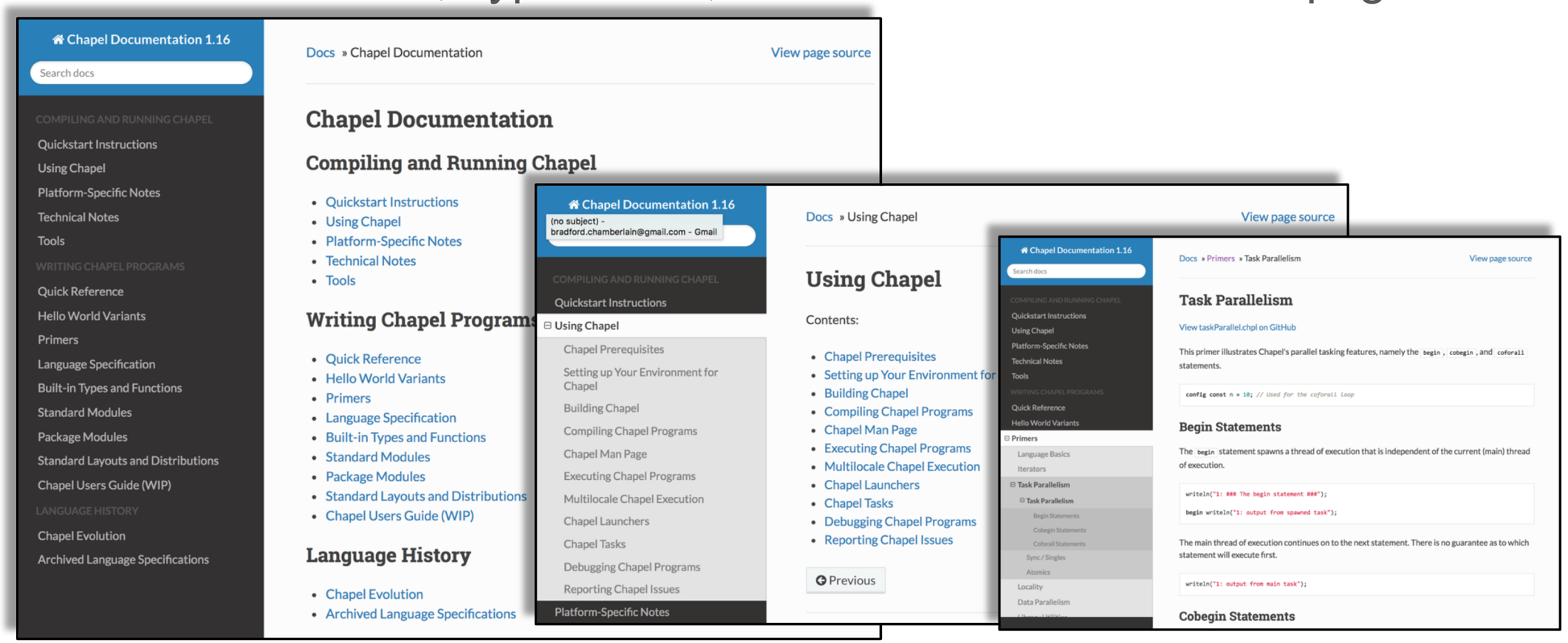
Default Values for Formal Arguments

```
proc foo(arg1: int, arg2: real = 3.14);
foo(2);
```

```
File Edit Options Buffers Tools chpl Help
//
// Task Parallel Primer
//
// This primer illustrates Chapel's parallel tasking features,
// namely the begin, cobegin, and coforall statements.

config const n = 10;

writeln("1: ### The begin statement ###");

// The begin statement spawns a thread of execution that is independent
// of the current (main) thread of execution.
begin writeln("1: output from spawned task");

// The main thread of execution continues on to the next statement.
// There is no guarantee as to which statment will execute first.
writeln("1: output from main task");


writeln("2: ### The cobegin statement ###");

// For more structured behavior, the cobegin statement can be used to
// spawn a block of tasks, one for each statement.  Control continues
// after the cobegin block, but only after all the tasks within the
// cobegin block have completed.
cobegin {
  writeln("2: output from spawned task 1");
  writeln("2: output from spawned task 2");
}

// The output from within the cobegin statement will always precede the
// following output from the main thread of execution.
writeln("2: output from main task");


writeln("3: ### The cobegin statement with nested begin statements ###");

// If any begin statements are used within a cobegin statement,
// the thread of execution does not a wait for those begin statements
// to complete.
cobegin {
  begin writeln("3: output from spawned task 1");
  begin writeln("3: output from spawned task 2");
}

-uu-:---F1  taskParallel.chpl   Top L1     (Chapel/l Abbrev)----------------------
Loading /users/bradc/chapel/highlight/emacs/22/chpl-mode.el (source)...done
```

Documentation: Now

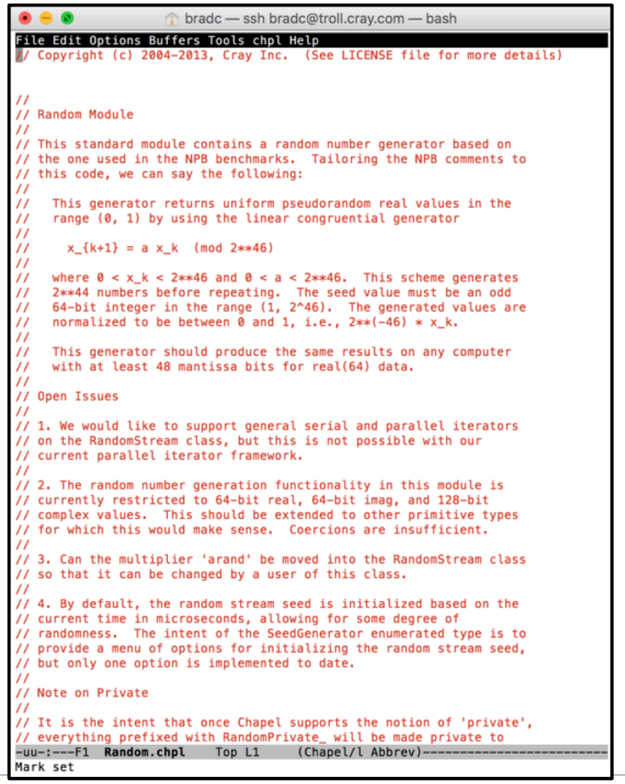
Now: 200+ modern, hyperlinked, web-based documentation pages

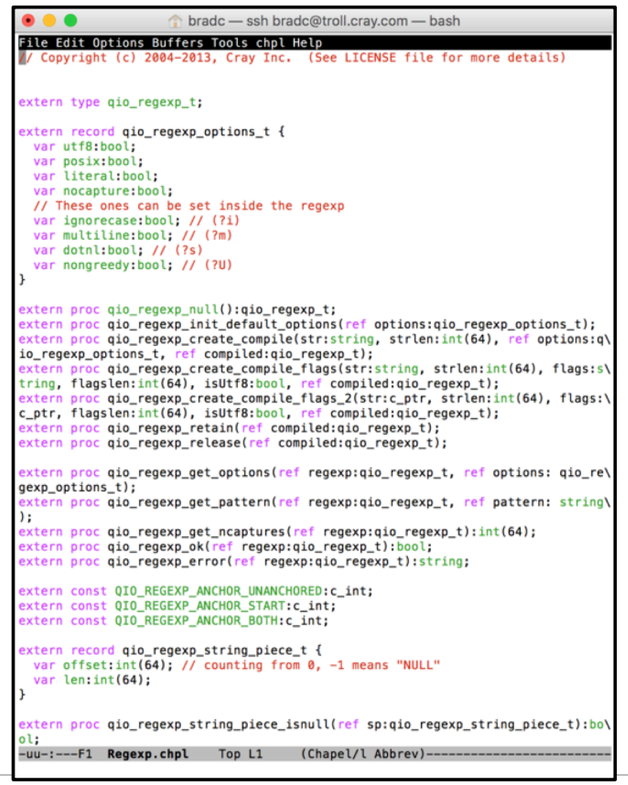


Libraries: Then

After HPCS: ~25 library modules

- documented via source comments, if at all:

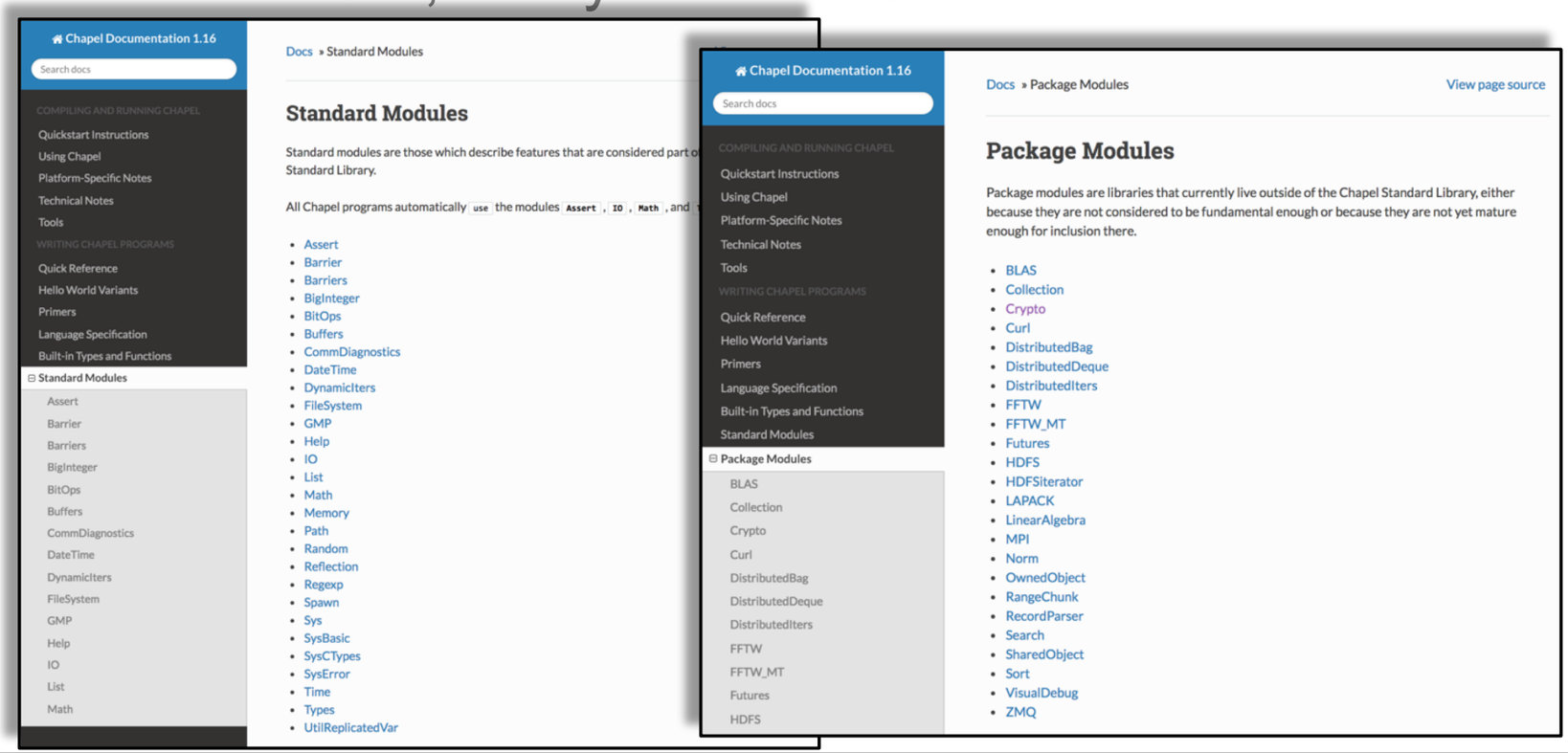




Libraries: Now

Now: ~60 library modules

- web-documented, many user-contributed



Libraries: Now

Math: FFTW, BLAS, LAPACK, LinearAlgebra, Math

Inter-Process Communication: MPI, ZMQ (ZeroMQ)

Parallelism: Futures, Barrier, DynamicIters

Distributed Computing: DistributedIters, DistributedBag, DistributedDeque, Block, Cyclic, Block-Cyclic, …

File Systems: FileSystem, Path, HDFS

Others: BigInteger, BitOps, Crypto, Curl, DateTime, Random, Reflection, Regexp, Search, Sort, Spawn, …

Tools: Then

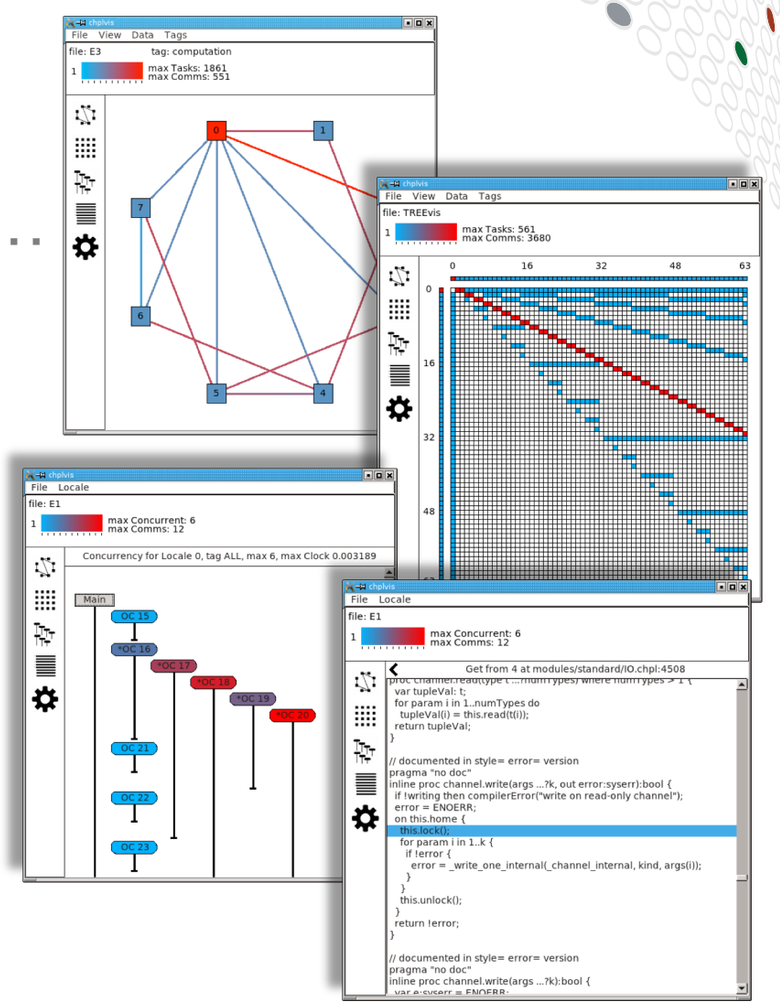
After HPCS:

- **highlighting modes** for emacs and vim
- **chpldoc:** documentation tool (early draft)

Tools: Now

Now:

- **highlighting modes** for emacs, vim, atom, …
- **chpldoc:** documentation tool
- **mason:** package manager
- **c2chapel:** interoperability aid
- **bash tab completion:** command-line help
- **chplvis:** performance visualizer / debugger



Then vs. Now: And so much more…

Interoperability:

- passing arrays & functions to C, working with C pointers, …

Development process:

- GitHub, Jenkins, Travis, interactive nightly performance graphs…

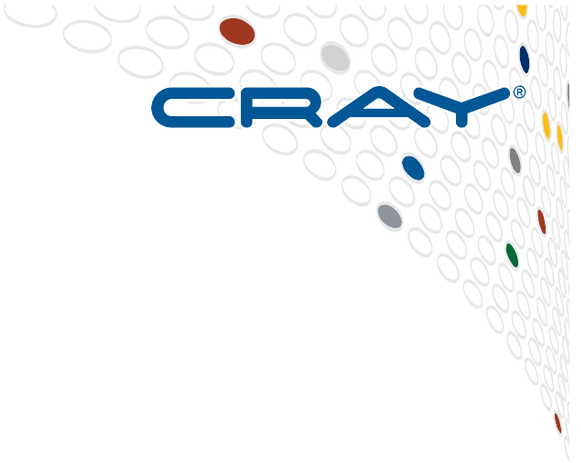
Social media: Twitter, Facebook, YouTube

User support: GitHub issues, StackOverflow, Gitter, email

Web presence: CLBG, Try It Online, CyberDojo, …

Memory Leaks: significantly reduced

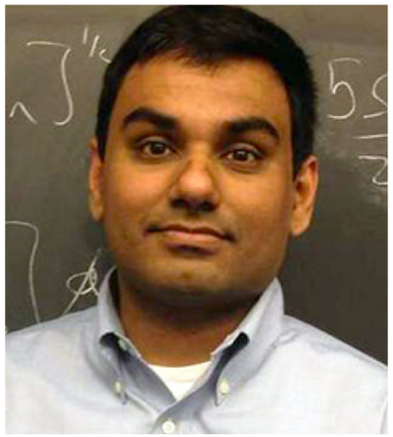
CHIUW: annual community workshop



Chapel User Profiles

Chapel User Profiles

Current Users:



Time-to-science Cosmologist

Commercial AI Scientist

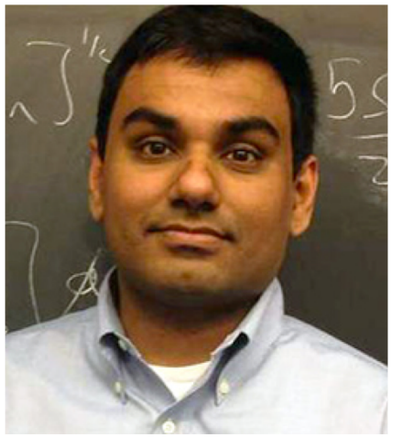
Potential Users:

Genomic Researcher

DOE Scientist

Chapel User Profiles

Current Users:

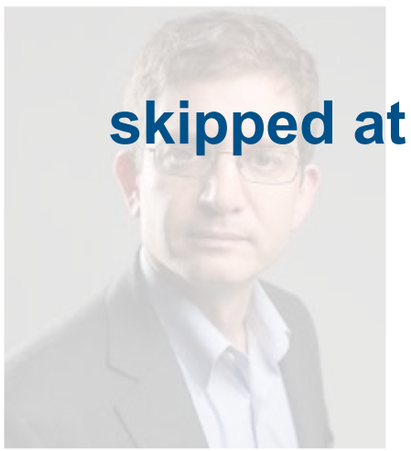


Time-to-science Cosmologist

Commercial AI Scientist

skipped at CUG due to time constraints

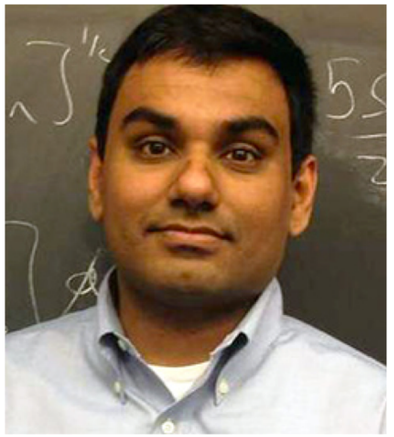
Potential Users:



Genomic Researcher

DOE Scientist

User Profile: Time-to-Science Cosmologist



Name: Nikhil Padmanabhan

Title: Associate Professor of Physics and Astronomy, Yale University

Computations: Surveys of galaxies to constrain cosmological models, n-body simulations of gravity

Why Chapel? "My interests in Chapel developed from a desire to have a lower barrier to writing parallel codes. In particular, I often find myself writing prototype codes (often serial), but then need to scale these codes to run on large numbers of simulations/datasets. **Chapel allows me to smoothly transition from serial to parallel codes with a minimal number of changes.**

"Another important issue for me is "my time to solution" (some measure of productivity vs performance). Raw performance is rarely the only consideration."

User Profile: Commercial AI Scientist

Name: Brian Dolan

Title: Co-Founder and Chief Scientist of Deep 6 AI

Computations: Natural language processing, AI and ML applications, network analysis, community detection, reinforcement learning in the form of Deep Q-Networks

Why Chapel? "I have used Fortran, R, Java and Python extensively. If I had to give up Chapel, I would probably move to C++. **I prefer Chapel due to the extreme legibility and performance.** We have abandoned Python on large problems for performance reasons.

"**We've now developed thousands of lines of Chapel code and a half dozen open source libraries** for things like database connectivity, numerical libraries, graph processing, and even a REST framework. We've done this because AI is about to face an HPC crisis, and the folks at Chapel understand the intersection of usability and scalability."

Potential User Profile: Genomic Researcher

Name: Jonathan Dursi

Title: Senior Research Associate, The Hospital for Sick Children, Toronto

Computations: Human genomics, bioinformatics, and medical informatics

Why Chapel? "My interest in Chapel lies in its potential for bioinformatics tools that are currently either written in elaborately crafted, threaded but single node, C++ code, or in Python. Either has advantages and disadvantages (performance vs rapid development cycles), but neither has a clear path to cross-node computation, for performance as well as larger memory and memory bandwidth. **Chapel has the potential to have some of the best of both worlds in terms of C++ and Python, as well as having a path to distributed memory.**"

Potential User Profile: DOE Scientist

Name: Anshu Dubey

Title: Computer Scientist, Argonne National Laboratory

Computations: Design and development of Multiphysics software that can serve multiple science domains; solvers for PDEs and ODEs

Why Chapel? "In Multiphysics applications separation of concerns and use of high level abstractions is critical for sustainable software. Chapel combines language features that would enable this for clean implementation.

"HPC Scientific software is made more complex than it needs to be because the only language designed for scientific work, Fortran, is losing ground for various reasons. Its object oriented features are clunky and make it nearly as unsuitable as other languages for scientific work. **Chapel appears to be parallel and modern Fortran done better, therefore has the potential to become a more suitable language.**"

Chapel and Productivity

Chapel aims to be as…

…**programmable** as Python

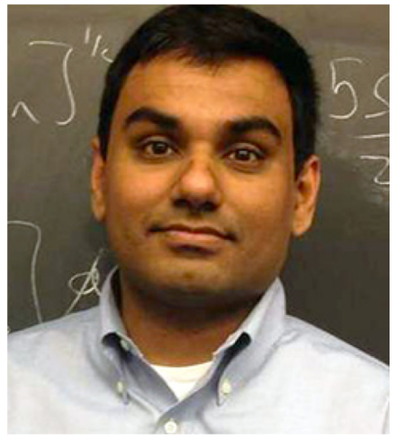
…**fast** as Fortran

…**scalable** as MPI, SHMEM, or UPC

…**portable** as C

…**flexible** as C++

…**fun** as [your favorite language]



What's Next?

Crossing the Stream of Adoption

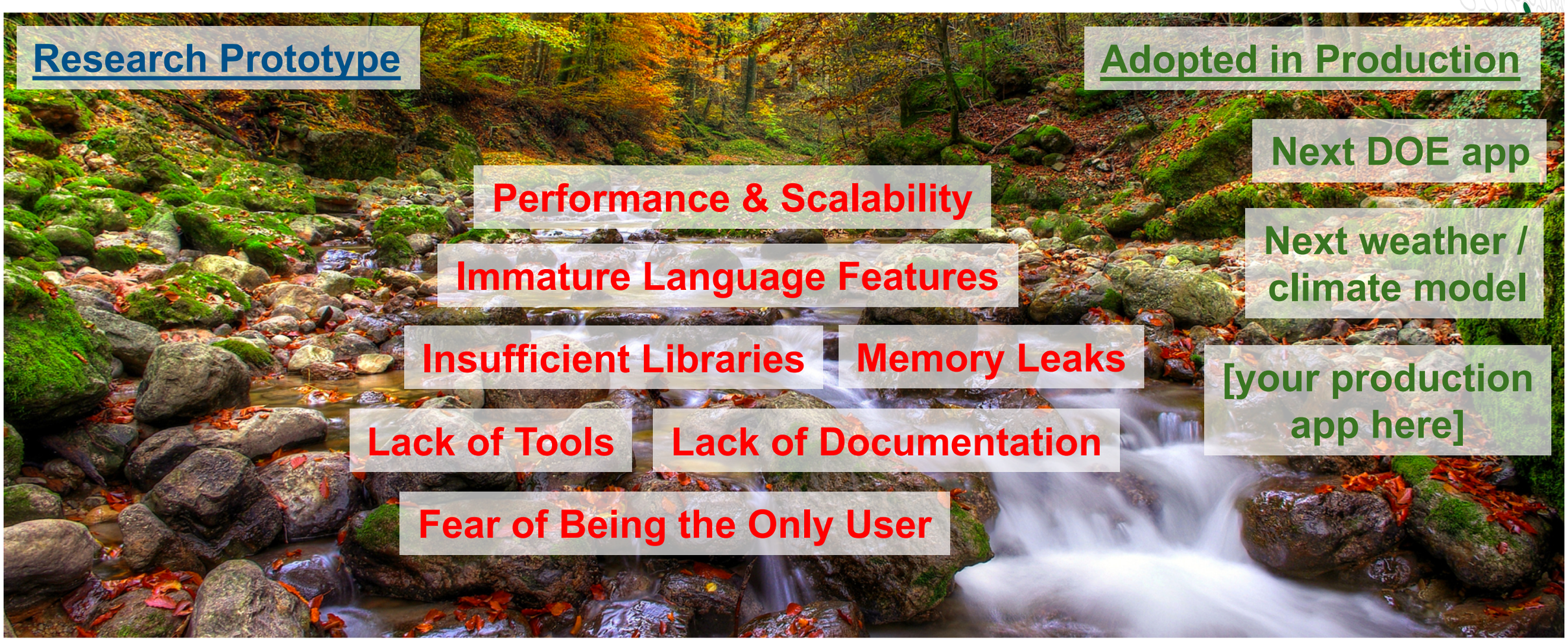


image credit: http://feelgrafix.com/813578-free-stream-wallpaper.html

Crossing the Stream of Adoption

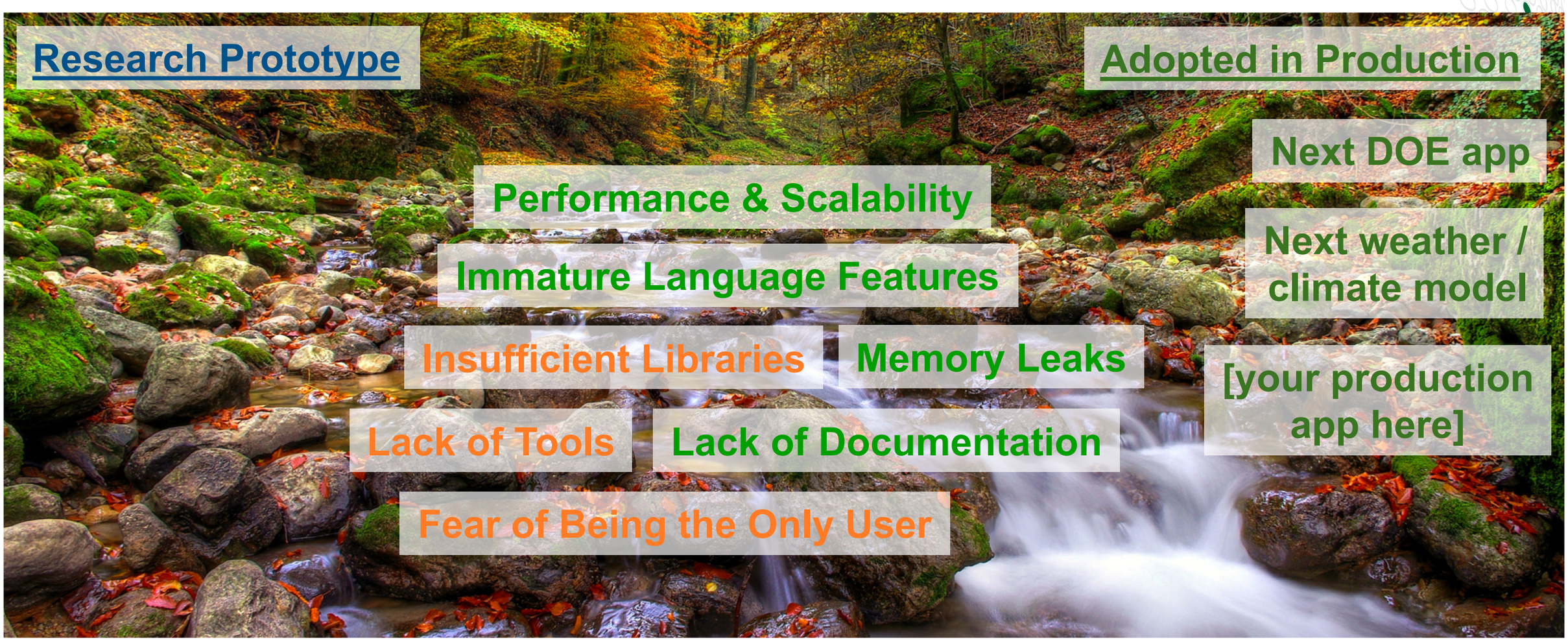


image credit: http://feelgrafix.com/813578-free-stream-wallpaper.html

Crossing the Stream of Adoption

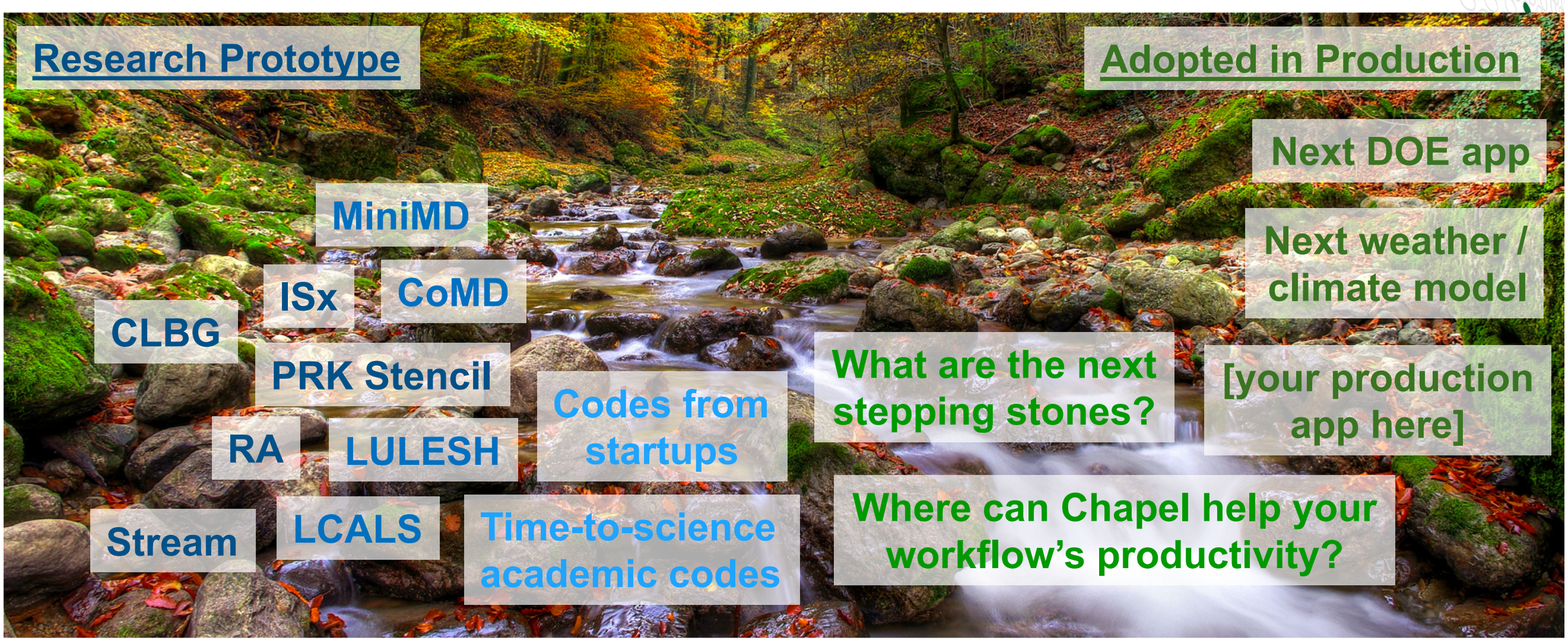
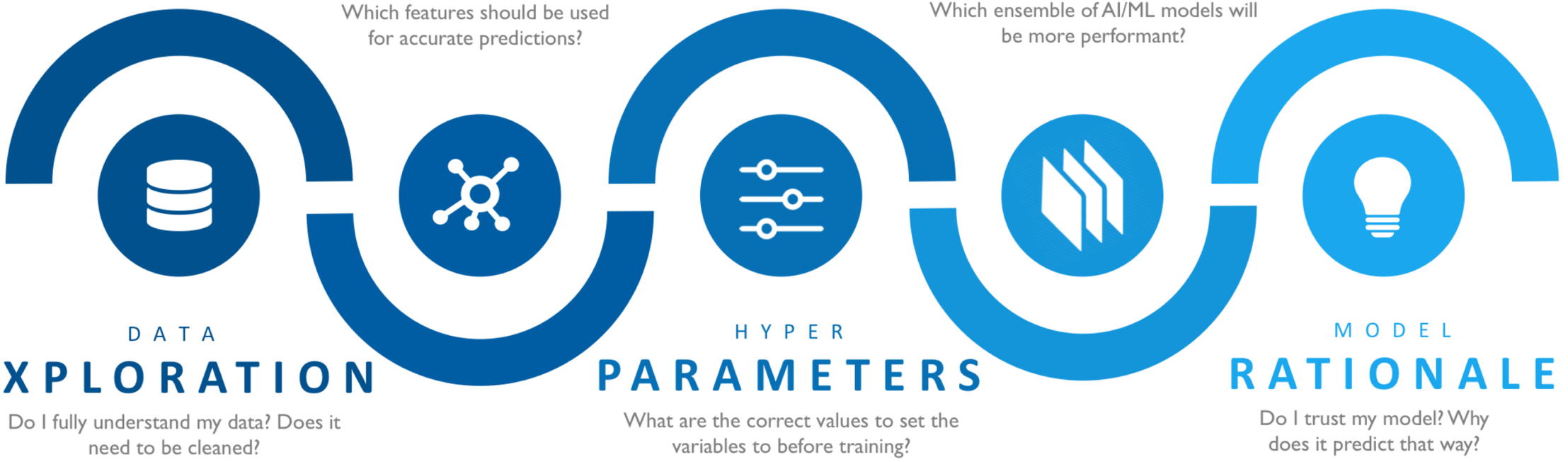


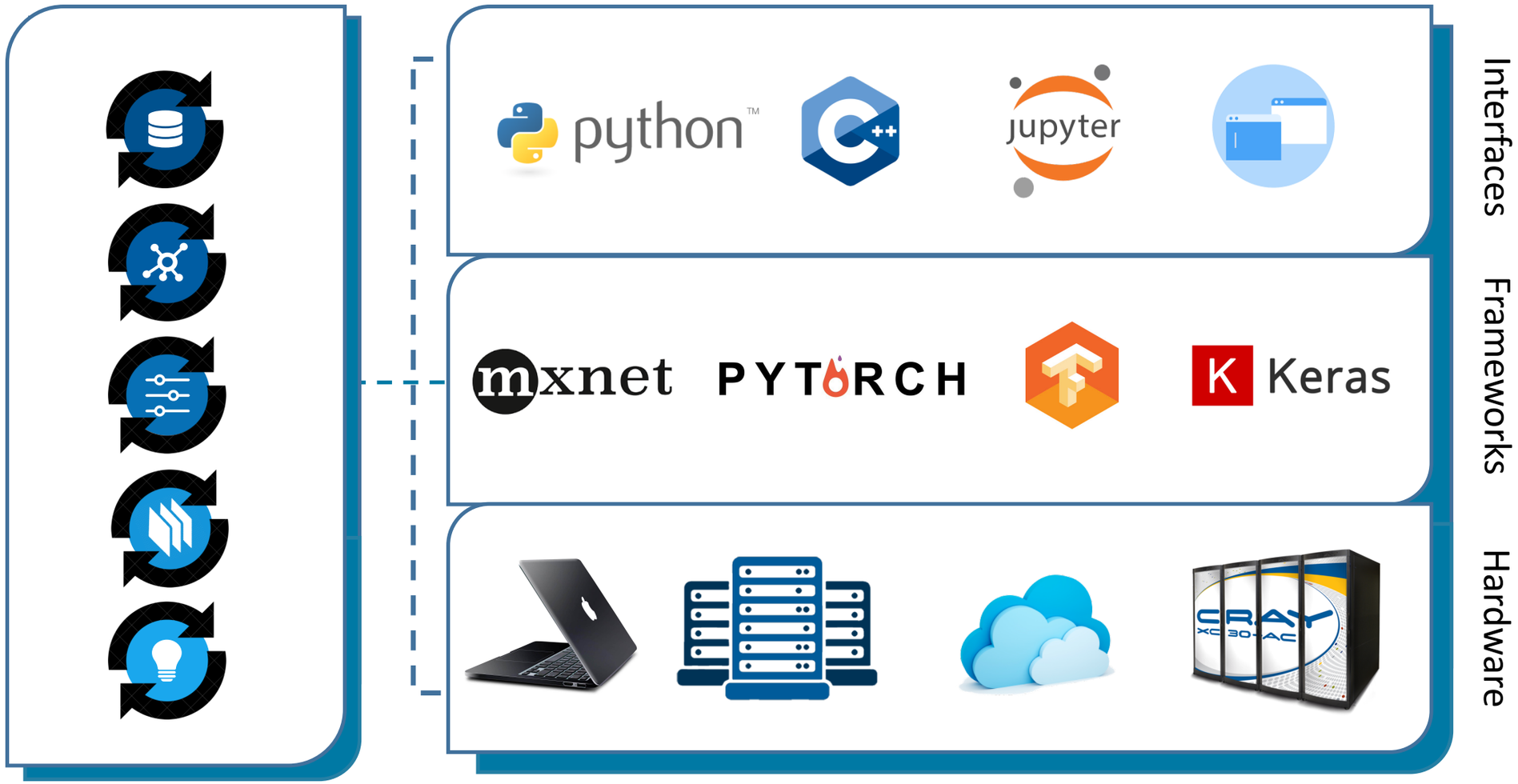
image credit: http://feelgrafix.com/813578-free-stream-wallpaper.html

Discovery Roadblocks

Data Science Pain Points



Chapel AI Ecosystem

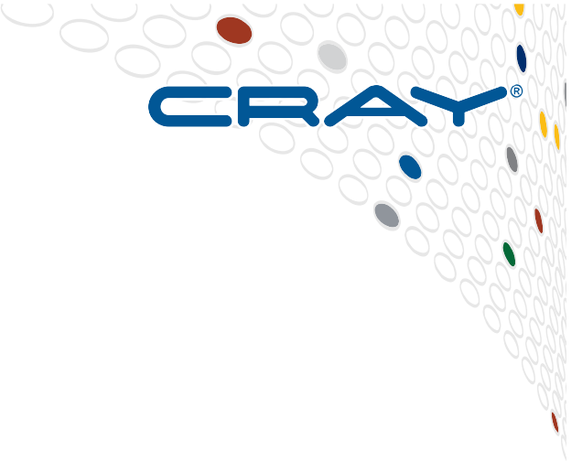


Sample Chapel AI Workflow

- **User works from within a Jupyter notebook**
- **Uses Chapel to ingest large HDF5 data files**
  - read in parallel
  - transformed / analyzed during ingestion
  - stored in a distributed Dataframe
- **Starts working on model locally on laptop**

- **As confidence in model grows, tunes it at scale**
  - feature selection
  - hyperparameter optimization

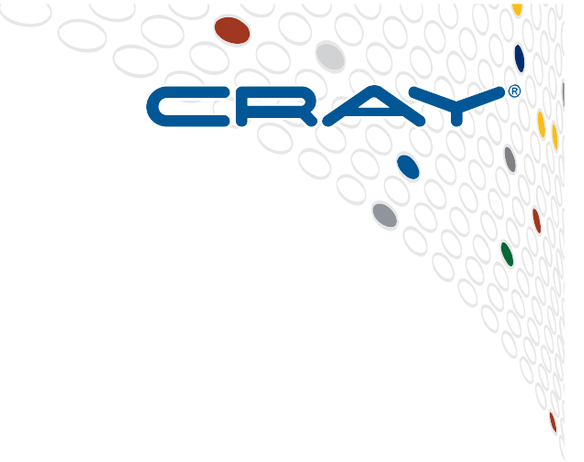
What's Next?



Chapel's college years: plans for 2018-2021

- **Language Core**
- **Interoperability / Usability**
- **Portability**
- **Data Ingestion**
- **Chapel AI**

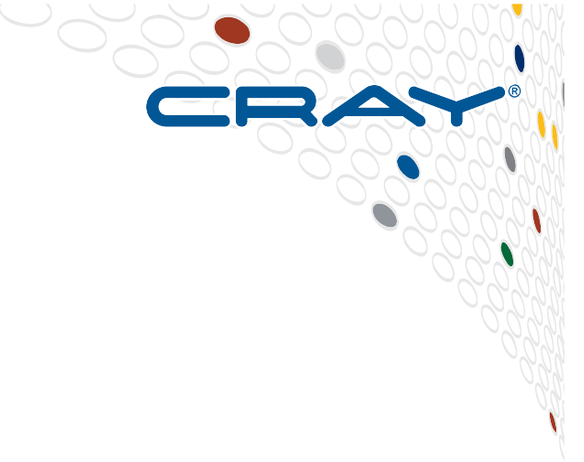
What's Next?



Chapel's college years: plans for 2018-2021

- **Language Core**
  - Language stabilization: avoid backward-breaking changes
  - Sparse array improvements, partial reductions, delete-free features, …
  - Additional performance and scalability improvements
- **Interoperability / Usability**
- **Portability**
- **Data Ingestion**
- **Chapel AI**

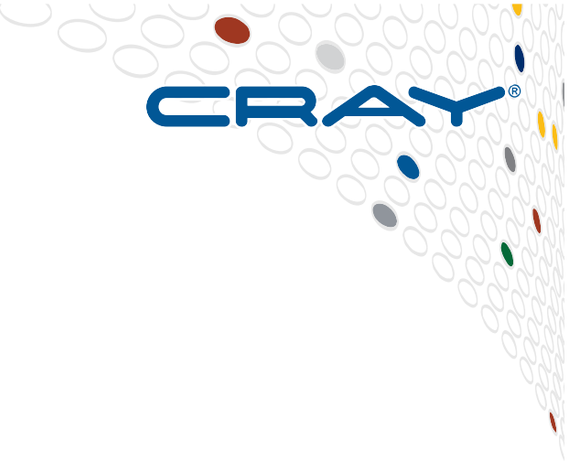
What's Next?



Chapel's college years: plans for 2018-2021

- **Language Core**
- **Interoperability / Usability**
  - Python / C++ interoperability
  - Support for Jupyter notebooks / REPL
- **Portability**
- **Data Ingestion**
- **Chapel AI**

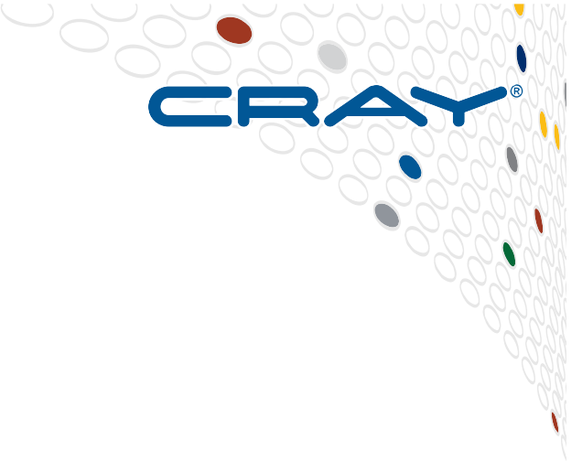
What's Next?



Chapel's college years: plans for 2018-2021

- **Language Core**
- **Interoperability / Usability**
- **Portability**
  - LLVM back-end
  - Target Libfabric/OFI
  - Target GPUs
  - Cloud computing support
- **Data Ingestion**
- **Chapel AI**

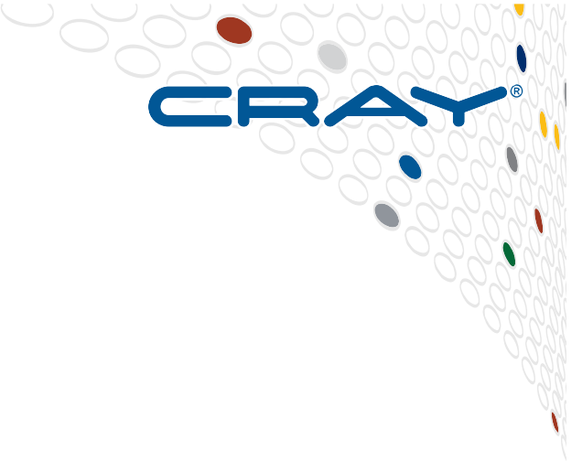
What's Next?



Chapel's college years: plans for 2018-2021

- **Language Core**
- **Interoperability / Usability**
- **Portability**
- **Data Ingestion**
  - Support HDF5, NetCDF, CSV, …
  - Transform-on-ingest
  - Distributed DataFrames support
- **Chapel AI**

What's Next?



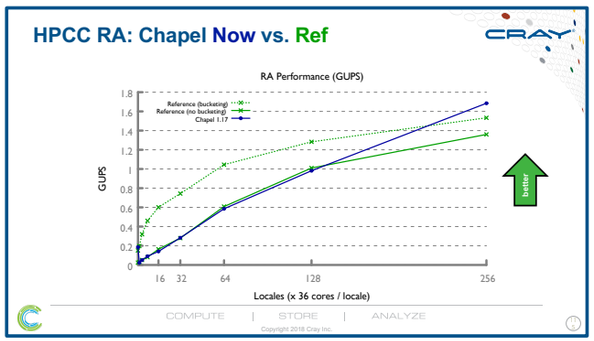
Chapel's college years: plans for 2018-2021

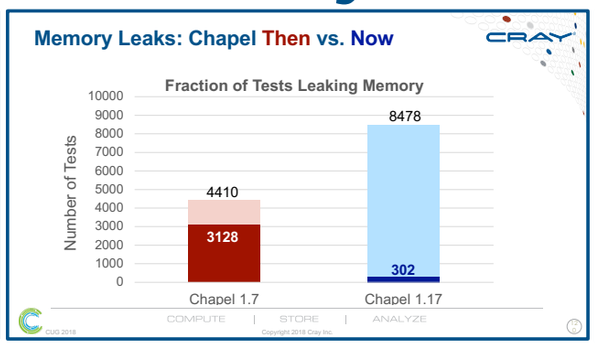
- **Language Core**
- **Interoperability / Usability**
- **Portability**
- **Data Ingestion**
- **Chapel AI**
  - Hyperparameter optimization
  - Deep Learning
  - …

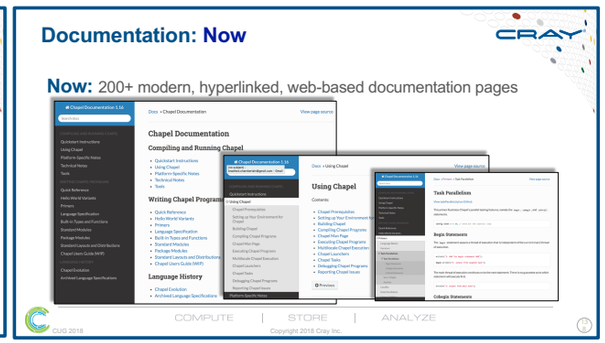
Summary

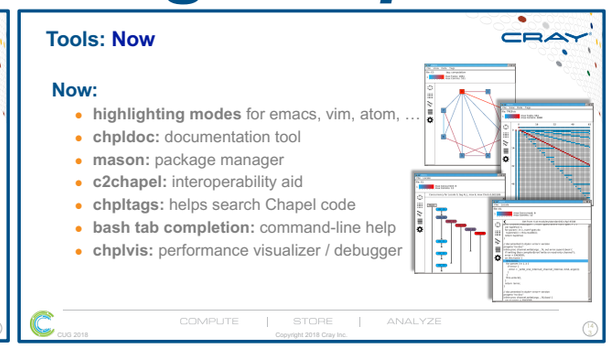
Chapel has made huge strides over the past five years

We've addressed many historical barriers to using Chapel









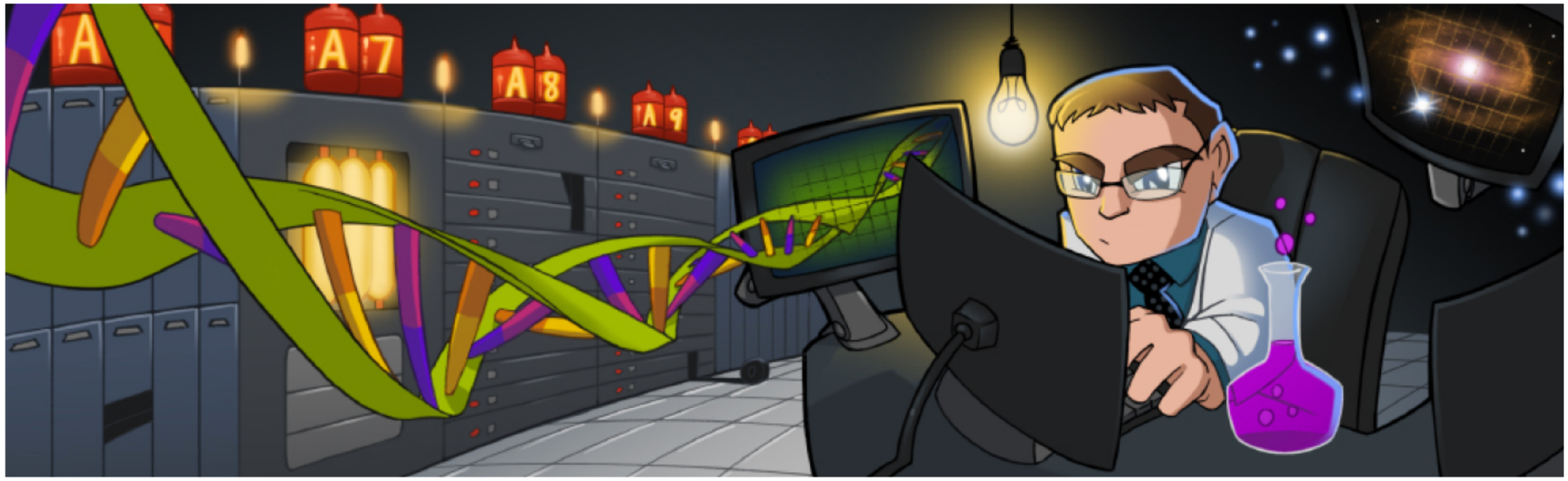
We're continuing our work to support and improve Chapel

We're looking for the next generation of Chapel users, as well as concrete use cases for AI / ML

CHIUW 2017 Keynote

Chapel's Home in the Landscape of New Scientific Computing Languages
(and what it can learn from the neighbours)

Jonathan Dursi, *The Hospital for Sick Children, Toronto*



Quote from CHIUW 2017 keynote

"My opinion as an outsider…is that Chapel is important, Chapel is mature, and Chapel is just getting started.

"If the scientific community is going to have frameworks…that are actually designed for our problems, they're going to come from a project like Chapel.

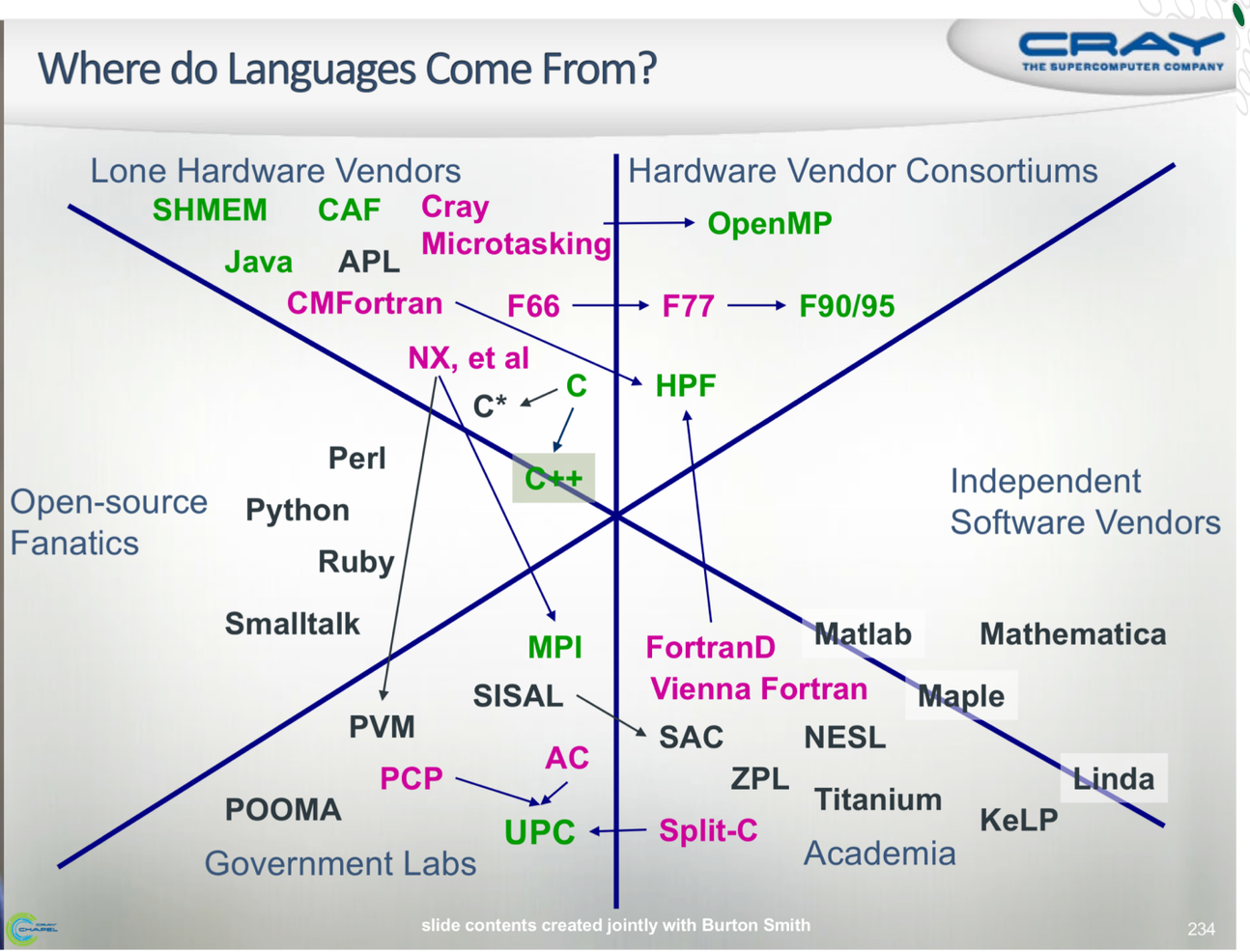
"And the thing about Chapel is that the set of all things that are 'projects like Chapel' is 'Chapel.'"

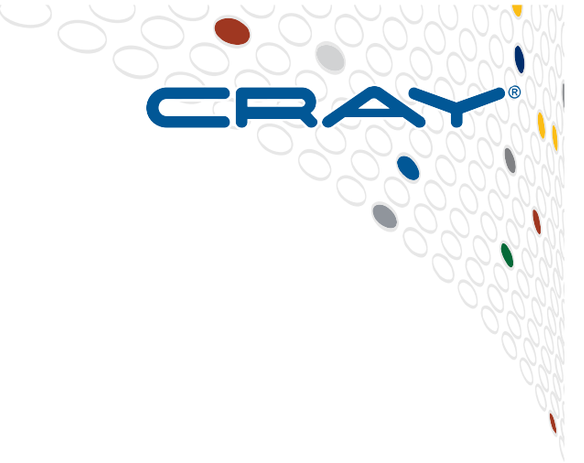
–Jonathan Dursi

Chapel's Home in the New Landscape of Scientific Frameworks
(and what it can learn from the neighbours)
CHIUW 2017 keynote

https://ljdursi.github.io/CHIUW2017 / https://www.youtube.com/watch?v=xj0rwdLOR4U

Dedicated to the Memory of Burton Smith



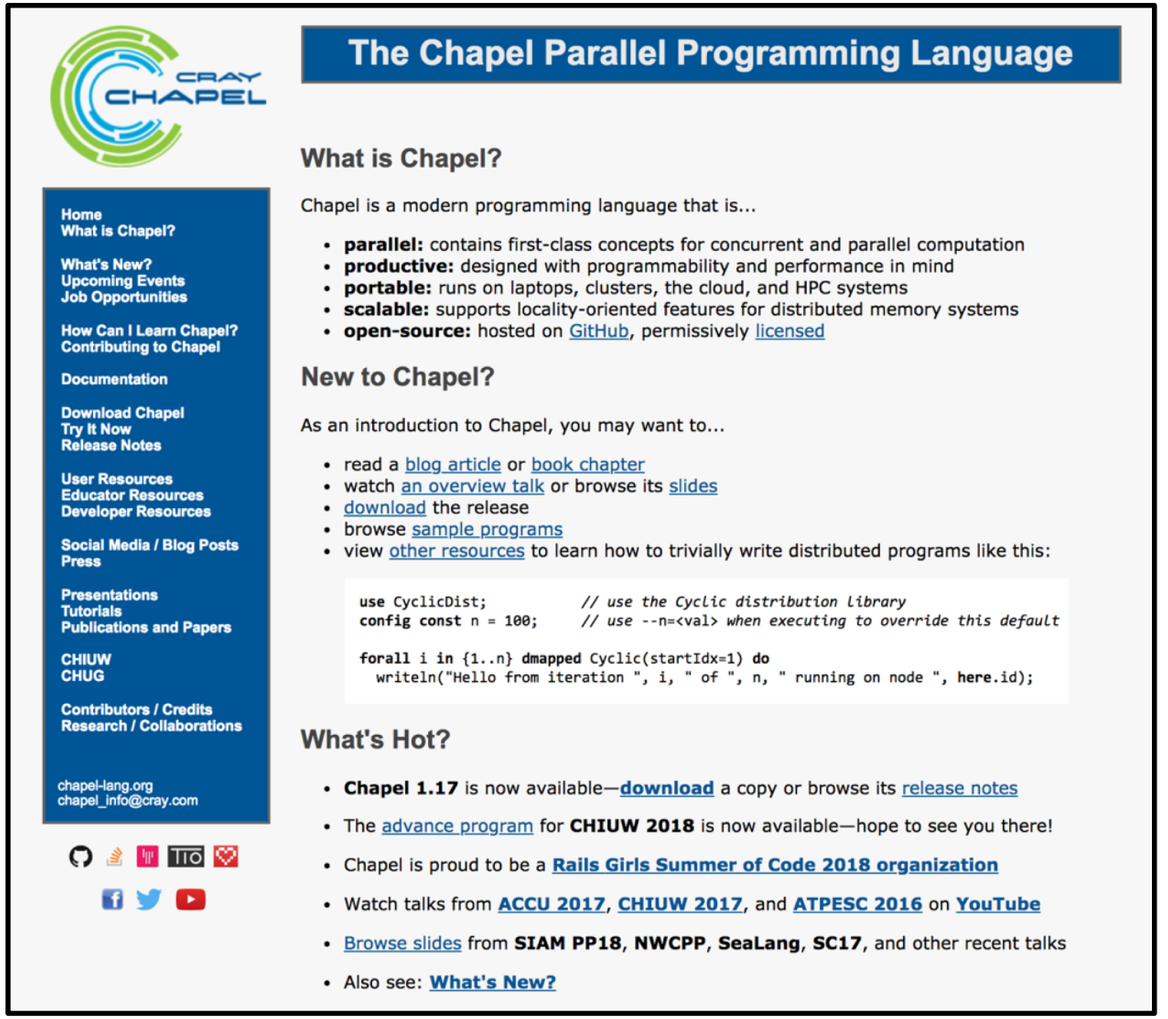


Chapel Resources

Chapel Central

https://chapel-lang.org

- downloads
- documentation
- resources
- presentations
- papers



The Chapel Parallel Programming Language

Home
What is Chapel?
What's New?
Upcoming Events
Job Opportunities
How Can I Learn Chapel?
Contributing to 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
Contributors / Credits
Research / Collaborations
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 --n=<val> when executing 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 1.17** is now available—**download** a copy or browse its release notes
- The advance program for **CHIUW 2018** is now available—hope to see you there!
- Chapel is proud to be a **Rails Girls Summer of Code 2018 organization**
- Watch talks from **ACCU 2017**, **CHIUW 2017**, and **ATPESC 2016** on **YouTube**
- Browse slides from **SIAM PP18**, **NWCPP**, **SeaLang**, **SC17**, and other recent talks
- Also see: **What's New?**

Chapel Social Media (no account required)

http://twitter.com/ChapelLanguage

http://facebook.com/ChapelLanguage

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

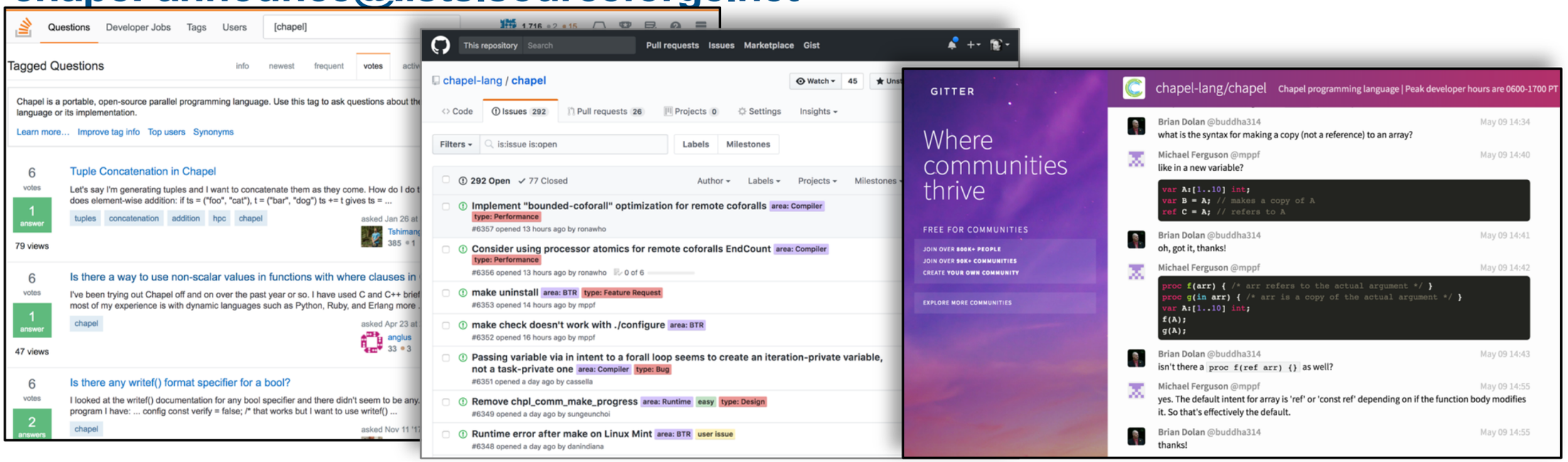
Chapel Community

https://stackoverflow.com/questions/tagged/chapel

https://github.com/chapel-lang/chapel/issues

https://gitter.im/chapel-lang/chapel

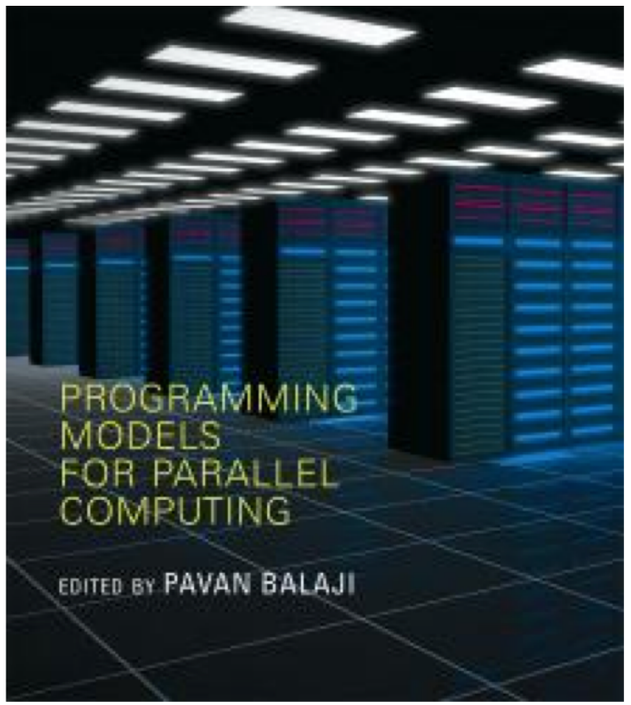
chapel-announce@lists.sourceforge.net



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 also available online



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

Suggested Reading (short attention spans)

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

- *a run-down of recent events (as of 2017)*

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*

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
- Gitter (chapel-lang/chapel): community chat with archives
- chapel-users@lists.sourceforge.net: user discussions

Discuss Chapel development

- chapel-developers@lists.sourceforge.net: developer discussions
- GitHub issues for chapel-lang/chapel: for feature requests, design discussions

Discuss Chapel's use in education

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

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

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 publicly 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, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. 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 on this website are the property of their respective owners.


Q&A
Brad Chamberlain
bradc@cray.com