



# Chapel: Making HPC Accessible

*Michael Ferguson*  
Cray Inc.

November 10, 2015







# 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.







# Talk Outline

- Services
- How can Chapel fit in?
- What is Chapel?
- Detailed example: Processing Twitter data

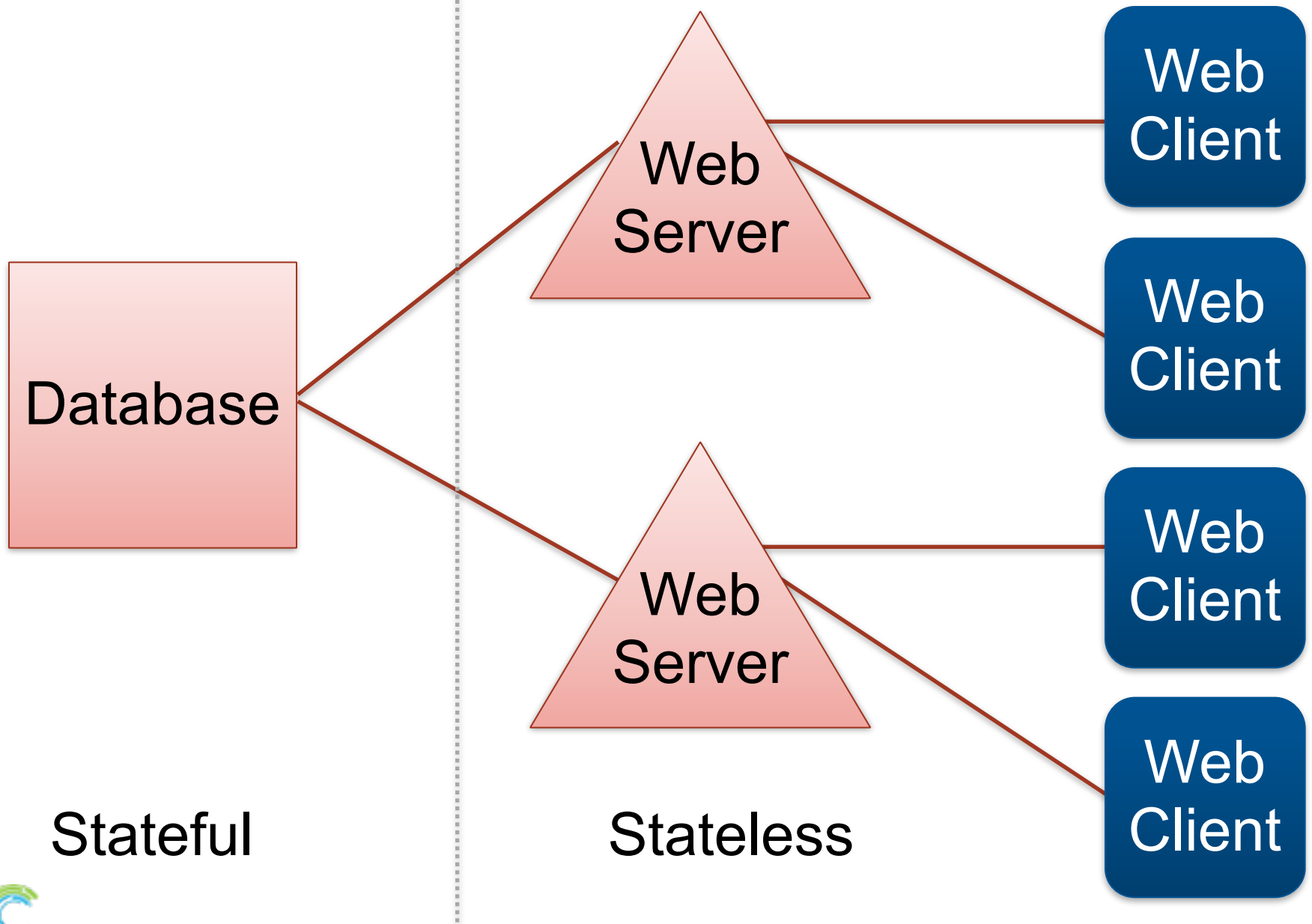




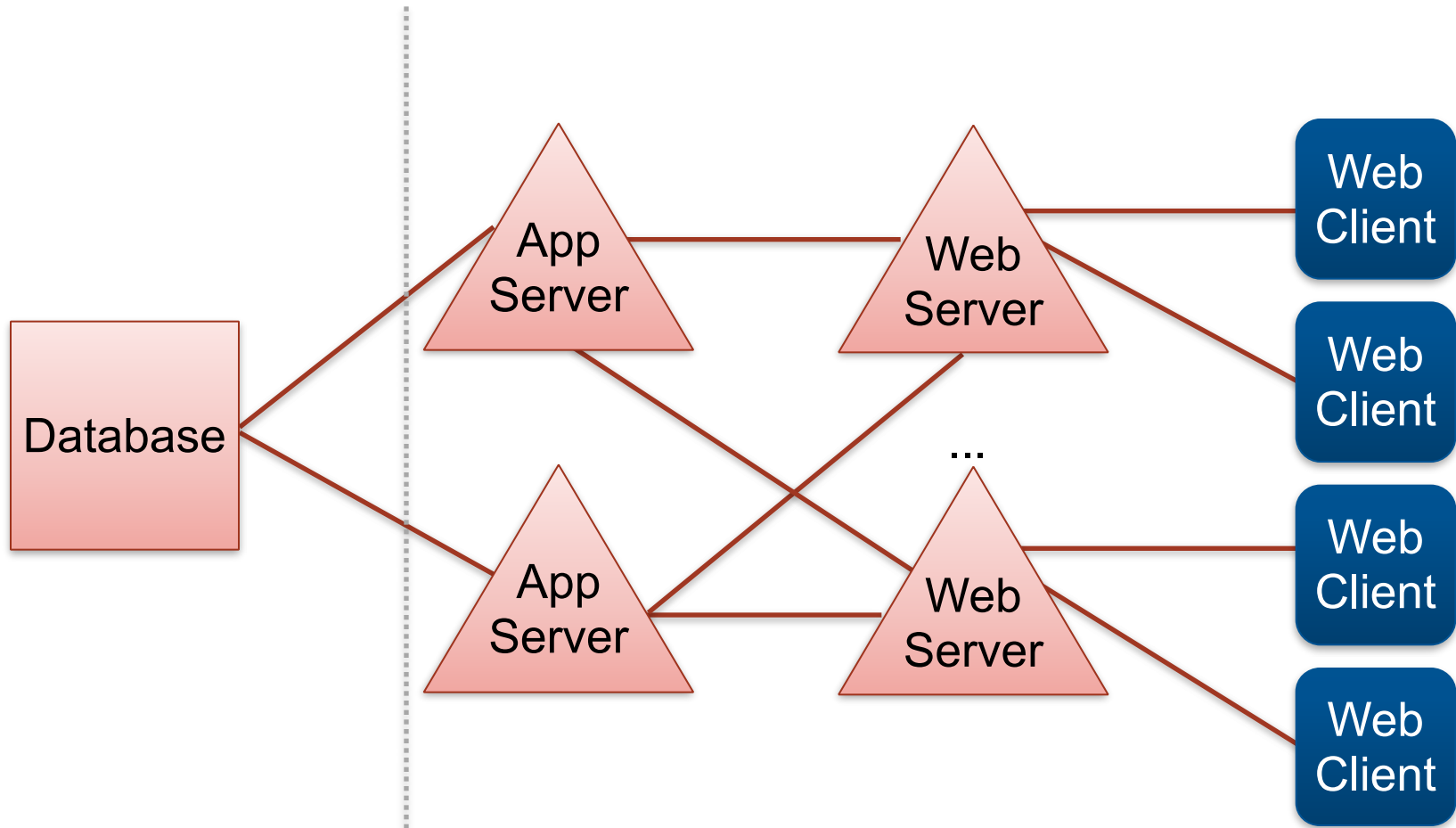
# Services









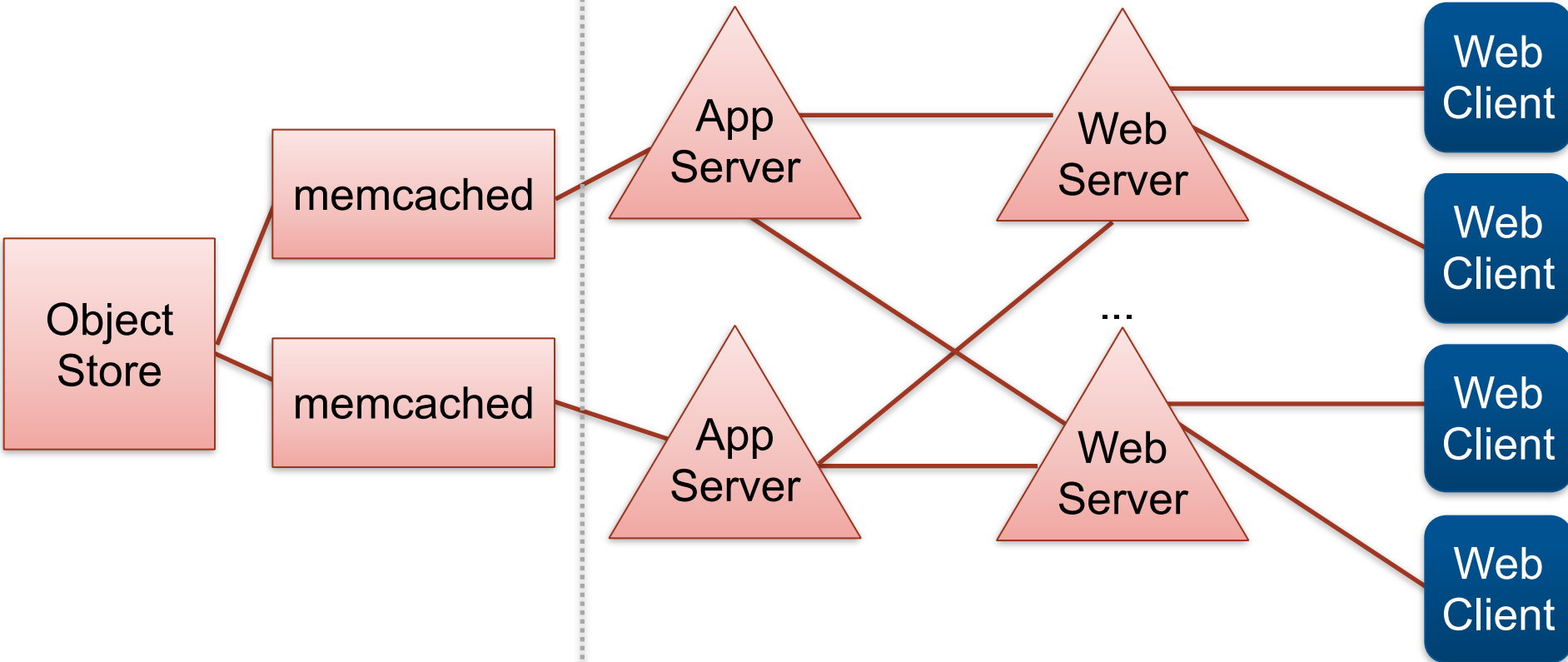


Stateful

Stateless





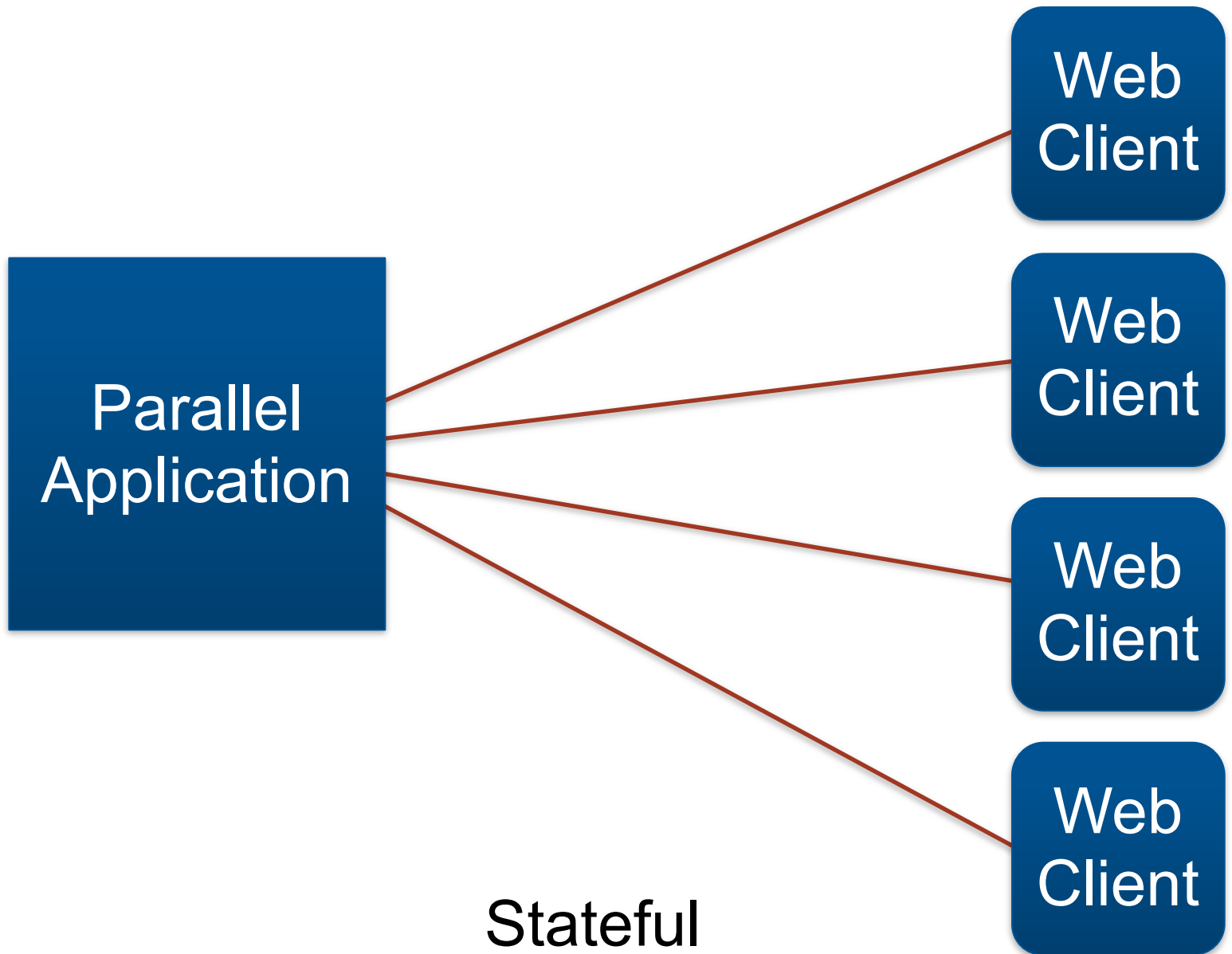


Stateful

Stateless









# Why I started working with Chapel

- Needed to speed up an application
- Thought perhaps new "supercomputer on a chip" hardware would help me... with more parallelism
- The High Performance Computing (HPC) community
  - Aims for maximal possible performance
  - Long history with parallel programming
- Maybe the rest of us can learn something from HPC





# Why I started working with Chapel

- Needed to speed up an application
- Thought perhaps new "supercomputer on a chip" hardware would help me... with more parallelism
- The High Performance Computing (HPC) community
  - Aims for maximal possible performance
  - Long history with parallel programming
- Maybe the rest of us can learn something from HPC
- ... But I needed
  - something other than Fortran + MPI
  - something portable across parallel architectures







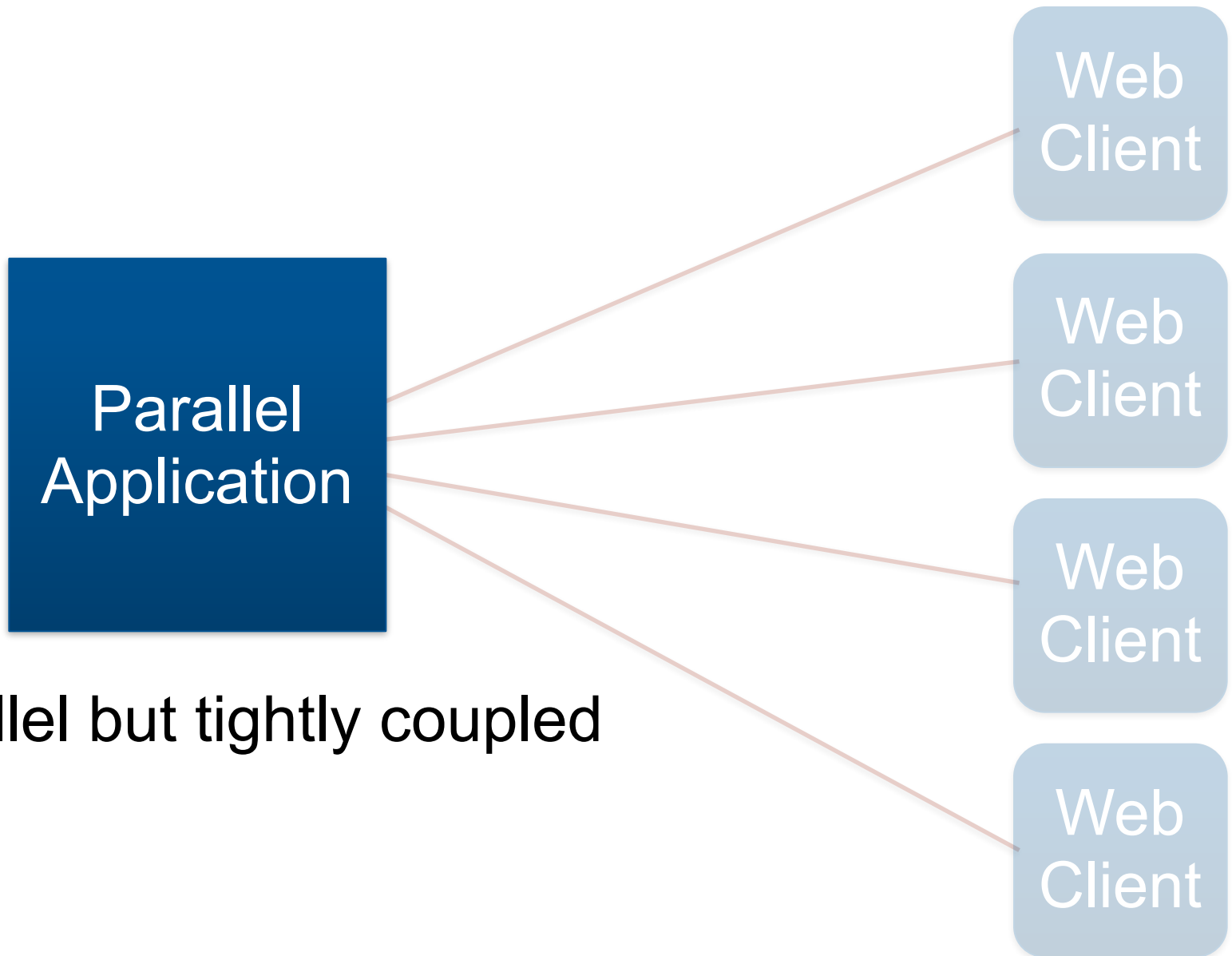
# How does Chapel fit in?



---

COMPUTE | STORE | ANALYZE





Parallel but tightly coupled





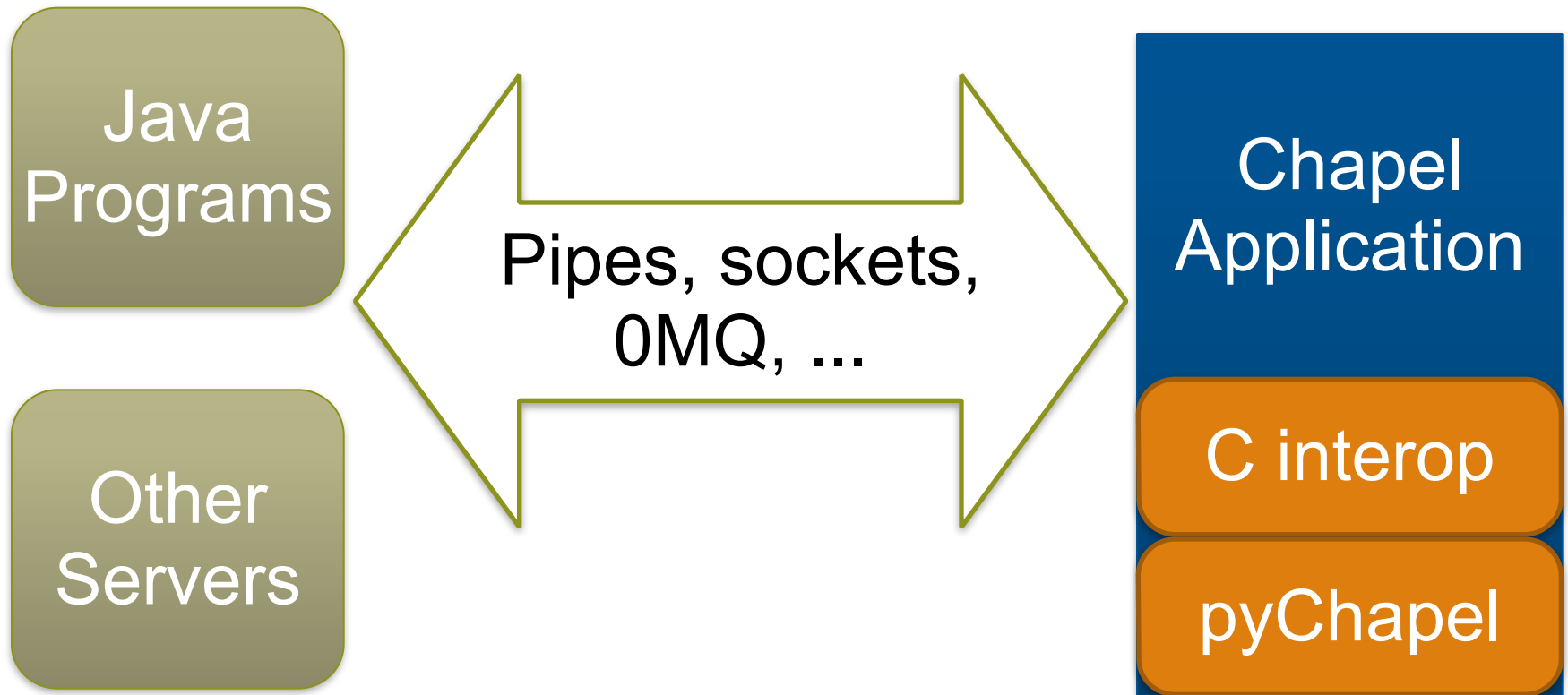
# Parallel Application



Cray Urika-XA™









# PRIMORDIAL

## MACHINE VISION SYSTEMS

[Chapel by Example](#) [Applications](#) [Development](#) [About/Contact](#)

Welcome to the PMVS home page! You'll find examples of our programs, apps, and notes here.  
Feel free to poke around.

## NEW

## CHAPEL BY EXAMPLE – IMAGE PROCESSING



```

} else {
    shuffle_sortind(lsort, cbase, midpt);

    tree(pos) = data(sortind(cbase){midpt});
    tree(pos).flags = cbase : uint(0) | HAS_L | HAS_R;
    cobegin {
        place_node(lsort[..midpt-1], cnext, left);
        place_node(lsort[midpt+1..], cnext, right);
    }
}

```

We've collected the notes we took while learning the Chapel language and developing several image processing programs in it and packaged them up into a set of [tutorials](#). You can find them on the Chapel By Example page. The examples include color converters, Gabor filters, k-means clustering, FAST corner detectors, and RANSAC feature matching.



```

private int
longitudeToTileX(final Zoom zoom, final double longitude) {
    final double xtmp;
    final int n;
    final int x;

    /* intermediate calculations */
    /* number of tiles at room level */
    /* tile coordinate */

    n = 2 << (zoom.zoom - 1);
    xtmp = longitude * Math.PI / 180.0;
    x = (int) Math.floor(n * (1 + (xtmp / Math.PI)) / 2);
    return x;
}

```

The [Development](#) page contains code samples and notes from the experience.





👁 Watch 1 ⭐ Star 3 🍴 Fork 1

3114f2d on Jun 6

1 contributor

149 lines (109 sloc) | 4.22 KB

Raw Blame History

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱ ⑲ ⑳ ㉑ ㉒ ㉓ ㉔ ㉕ ㉖ ㉗ ㉘ ㉙ ㉚ ㉛ ㉜ ㉝ ㉞ ㉟ ㊱ ㊲ ㊳ ㊴ ㊵ ㊶ ㊷ ㊸ ㊹ ㊺ ㊻ ㊼ ㊽ ㊾ ㊿

This application demonstrates how to use various important features of Chapel, such as locales, and how to minimize RPC traffic through features such as local. It also shows how to build a simple, efficient, inverted index using only integer representations.

Project link to this page: <http://chearch.pw> (church pew)

- lock-free, using atomic operations for all appropriate operations
- string-free, the entire engine is integer-based. This minimizes memory footprint while improves processing speed
- boolean queries, using an integer-based (no strings, remember?) query language called CHASM (Chearch Assembly)





# What is Chapel?



---

COMPUTE | STORE | ANALYZE



# Chapel's Origins: HPCS

## DARPA HPCS: High Productivity Computing Systems

- **Goal:** improve productivity by a factor of 10x
- **Timeframe:** summer 2002 – fall 2012
- Cray developed a new system architecture, network, software, ...
  - this became the very successful Cray XC30™ Supercomputer Series



...and a new programming language: Chapel







# Chapel Motivation

**Q:** Why doesn't parallel programming have an equivalent to Python / Matlab / Java / C++ / (your favorite programming language here) ?

- one that makes it easy to quickly get codes up and running
- one that is portable across system architectures and scales
- one that bridges the HPC, data analysis, and mainstream communities

**A:** We believe this is due not to any particular technical challenge, but rather a lack of sufficient...

...long-term efforts

...resources

...community will

...co-design between developers and users

...patience

***Chapel is our attempt to change this***



COMPUTE | STORE | ANALYZE





# Chapel's Implementation

- **Being developed as open source at GitHub**
  - Licensed as Apache v2.0 software
- **Portable design and implementation, targeting:**
  - multicore desktops and laptops
  - commodity clusters and the cloud
  - HPC systems from Cray and other vendors
  - *in-progress*: manycore processors, CPU+accelerator hybrids, ...





# Chapel is a Collaborative, Community Effort



Lawrence Berkeley  
National Laboratory



(and many others as well...)

<http://chapel.cray.com/collaborations.html>



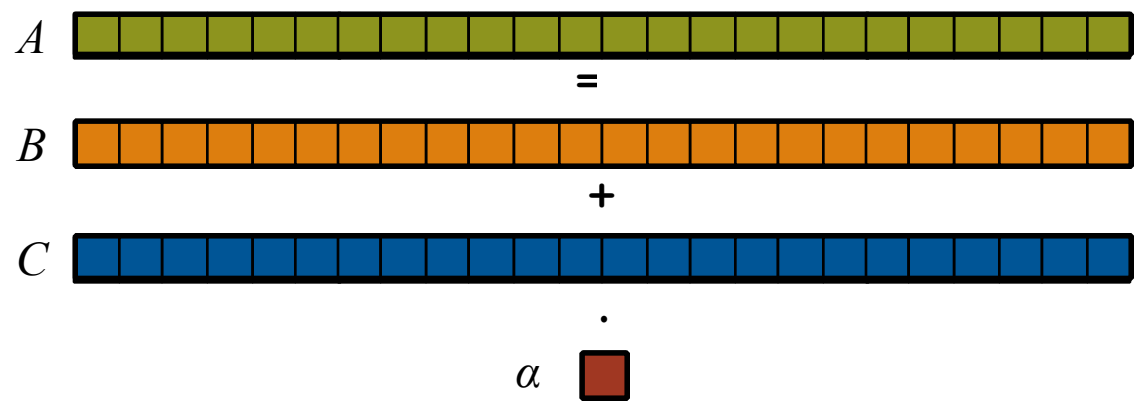


# STREAM Triad: a trivial parallel computation

**Given:**  $m$ -element vectors  $A, B, C$

**Compute:**  $\forall i \in 1..m, A_i = B_i + \alpha \cdot C_i$

In pictures:



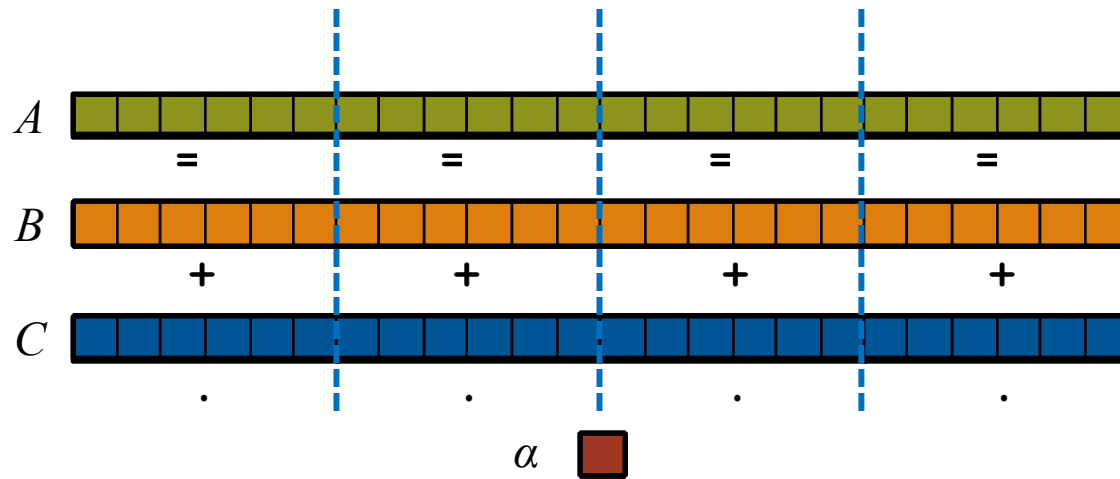


# STREAM Triad: a trivial parallel computation

**Given:**  $m$ -element vectors  $A, B, C$

**Compute:**  $\forall i \in 1..m, A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel:



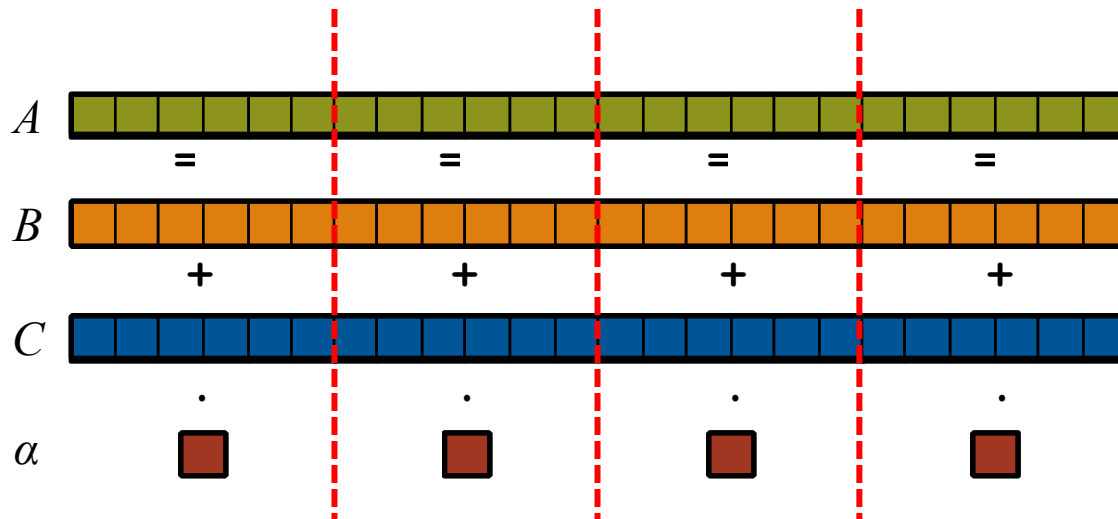


# STREAM Triad: a trivial parallel computation

**Given:**  $m$ -element vectors  $A, B, C$

**Compute:**  $\forall i \in 1..m, A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (distributed memory):



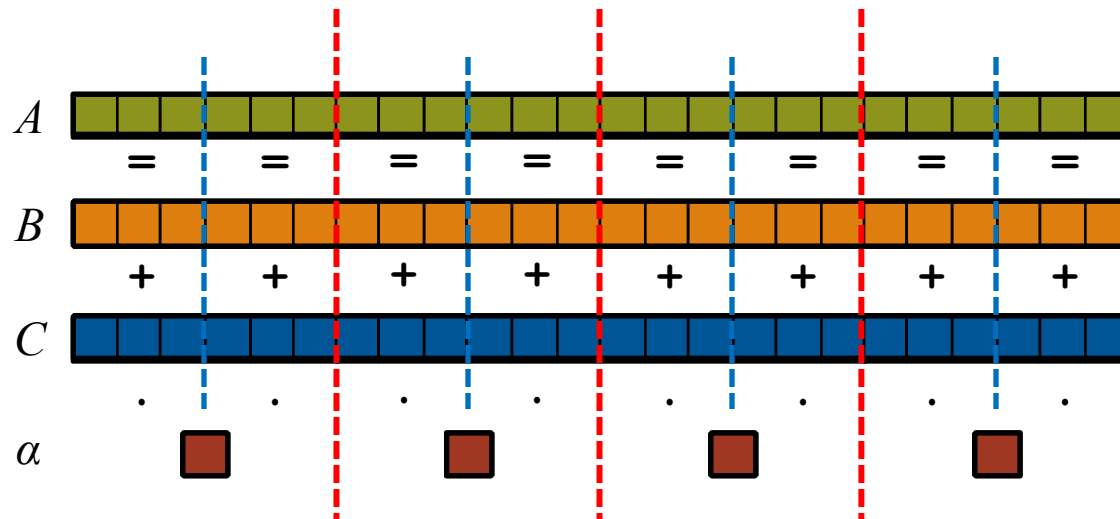


# STREAM Triad: a trivial parallel computation

**Given:**  $m$ -element vectors  $A, B, C$

**Compute:**  $\forall i \in 1..m, A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (distributed memory multicore):





# STREAM Triad: MPI



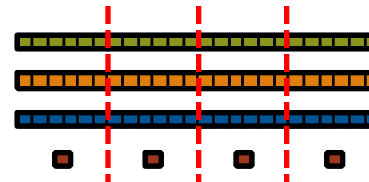
MPI

```
#include <hpcc.h>
```

```
static int VectorSize;  
static double *a, *b, *c;
```

```
int HPCC_StarStream(HPCC_Params *params) {  
    int myRank, commSize;  
    int rv, errCount;  
    MPI_Comm comm = MPI_COMM_WORLD;  
  
    MPI_Comm_size( comm, &commSize );  
    MPI_Comm_rank( comm, &myRank );  
  
    rv = HPCC_Stream( params, 0 == myRank );  
    MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0,  
               comm );  
  
    return errCount;  
}
```

```
int HPCC_Stream(HPCC_Params *params, int doIO) {  
    register int j;  
    double scalar;  
  
    VectorSize = HPCC_LocalVectorSize( params, 3,  
        sizeof(double), 0 );  
  
    a = HPCC_XMALLOC( double, VectorSize );  
    b = HPCC_XMALLOC( double, VectorSize );  
    c = HPCC_XMALLOC( double, VectorSize );
```



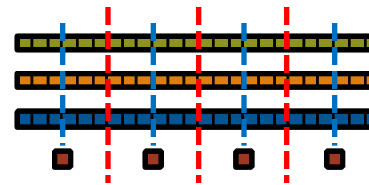
```
if (!a || !b || !c) {  
    if (c) HPCC_free(c);  
    if (b) HPCC_free(b);  
    if (a) HPCC_free(a);  
    if (doIO) {  
        fprintf( outFile, "Failed to allocate memory (%d).\n",  
            VectorSize );  
        fclose( outFile );  
    }  
    return 1;  
}
```

```
for (j=0; j<VectorSize; j++) {  
    b[j] = 2.0;  
    c[j] = 1.0;  
}  
  
scalar = 3.0;
```

```
for (j=0; j<VectorSize; j++)  
    a[j] = b[j]+scalar*c[j];  
  
HPCC_free(c);  
HPCC_free(b);  
HPCC_free(a);
```



# STREAM Triad: MPI+OpenMP



## MPI + OpenMP

```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
    int myRank, commSize;
    int rv, errCount;
    MPI_Comm comm = MPI_COMM_WORLD;

    MPI_Comm_size( comm, &commSize );
    MPI_Comm_rank( comm, &myRank );

    rv = HPCC_Stream( params, 0 == myRank );
    MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0,
        comm );

    return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
    register int j;
    double scalar;

    VectorSize = HPCC_LocalVectorSize( params, 3,
        sizeof(double), 0 );

    a = HPCC_XMALLOC( double, VectorSize );
    b = HPCC_XMALLOC( double, VectorSize );
    c = HPCC_XMALLOC( double, VectorSize );
```

```
    if (!a || !b || !c) {
        if (c) HPCC_free(c);
        if (b) HPCC_free(b);
        if (a) HPCC_free(a);
        if (doIO) {
            fprintf( outFile, "Failed to allocate memory (%d).\n",
                VectorSize );
            fclose( outFile );
        }
        return 1;
    }

#ifdef _OPENMP
#pragma omp parallel for
#endif
    for (j=0; j<VectorSize; j++) {
        b[j] = 2.0;
        c[j] = 1.0;
    }

    scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
    for (j=0; j<VectorSize; j++)
        a[j] = b[j]+scalar*c[j];

    HPCC_free(c);
    HPCC_free(b);
    HPCC_free(a);
```



# STREAM Triad: MPI+OpenMP vs. CUDA

## MPI + OpenMP

```
#ifndef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
    int myRank, commSize;
    int rv, errCount;
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_size( comm, &commSize );
    MPI_Comm_rank( comm, &myRank );

    rv = HPCC_Stream( params, 0 == myRank );
    MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );
    return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
    VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );

    a = HPCC_XMALLOC( double, VectorSize );
    b = HPCC_XMALLOC( double, VectorSize );
    c = HPCC_XMALLOC( double, VectorSize );

    if (!a || !b || !c) {
        if (c) HPCC_free(c);
        if (b) HPCC_free(b);
        if (a) HPCC_free(a);
        if (doIO) {
            fprintf( outFile, "Failed to allocate memory (%d).\n", VectorSize );
            fclose( outFile );
        }
        return 1;
    }

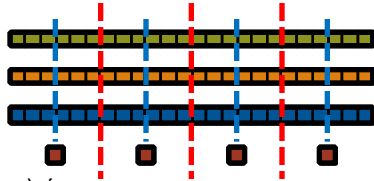
    #ifndef _OPENMP
    #pragma omp parallel for
    #endif
    for (j=0; j<VectorSize; j++) {
        b[j] = 2.0;
        c[j] = 1.0;
    }

    scalar = 3.0;

    #ifndef _OPENMP
    #pragma omp parallel for
    #endif
    for (j=0; j<VectorSize; j++)
        a[j] = b[j]+scalar*c[j];

    HPCC_free(c);
    HPCC_free(b);
    HPCC_free(a);

    return 0;
}
```



## CUDA

```
#define N 2000000

int main() {
    float *d_a, *d_b, *d_c;
    float scalar;

    cudaMalloc( (void**) &d_a, sizeof(float)*N );
    cudaMalloc( (void**) &d_b, sizeof(float)*N );
    cudaMalloc( (void**) &d_c, sizeof(float)*N );

    dim3 dimBlock(128);
    dim3 dimGrid(N/dimBlock.x);

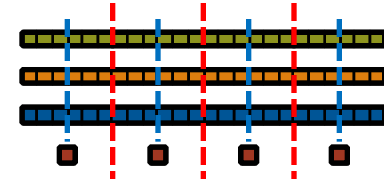
    set_array<<<dimGrid,dimBlock>>>(d_b, .5f, N);
    set_array<<<dimGrid,dimBlock>>>(d_c, .5f, N);

    scalar=3.0f;
    STREAM_Triad<<<dimGrid,dimBlock>>>(d_b, d_c, d_a, scalar, N);
    cudaThreadSynchronize();

    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    __global__ void set_array(float *a, float value, int len) {
        int idx = threadIdx.x + blockIdx.x * blockDim.x;
        if (idx < len) a[idx] = value;
    }

    __global__ void STREAM_Triad( float *a, float *b, float *c,
                                float scalar, int len) {
        int idx = threadIdx.x + blockIdx.x * blockDim.x;
        if (idx < len) c[idx] = a[idx]+scalar*b[idx];
    }
}
```



*HPC suffers from too many distinct notations for expressing parallelism and locality*





# Why so many programming models?

HPC has traditionally given users...

- ...low-level, *control-centric* programming models
- ...ones that are closely tied to the underlying hardware
- ...ones that support only a single type of parallelism

Type of HW Parallelism	Programming Model	Unit of Parallelism
<i>Inter-node</i>	<i>MPI</i>	<i>executable</i>
<i>Intra-node/multicore</i>	<i>OpenMP / pthreads</i>	<i>iteration/task</i>
<i>Instruction-level vectors/threads</i>	<i>pragmas</i>	<i>iteration</i>
<i>GPU/accelerator</i>	<i>Open[MP CL ACC] / CUDA</i>	<i>SIMD function/task</i>

**benefits:** lots of control; decent generality; easy to implement

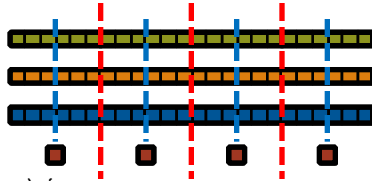
**downsides:** lots of user-managed detail; brittle to changes





# Rewinding a few slides...

## MPI + OpenMP



```
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
    int myRank, commSize;
    int rv, errCount;
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_size( comm, &commSize );
    MPI_Comm_rank( comm, &myRank );

    rv = HPCC_Stream( params, 0 == myRank );
    MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );
    return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
    VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );

    a = HPCC_XMALLOC( double, VectorSize );
    b = HPCC_XMALLOC( double, VectorSize );
    c = HPCC_XMALLOC( double, VectorSize );

    if (!a || !b || !c) {
        if (c) HPCC_free(c);
        if (b) HPCC_free(b);
        if (a) HPCC_free(a);
        if (doIO) {
            fprintf( outFile, "Failed to allocate memory (%d).\n", VectorSize );
            fclose( outFile );
        }
        return 1;
    }

#ifdef _OPENMP
#pragma omp parallel for
#endif
    for (j=0; j<VectorSize; j++) {
        b[j] = 2.0;
        c[j] = 1.0;
    }

    scalar = 3.0;

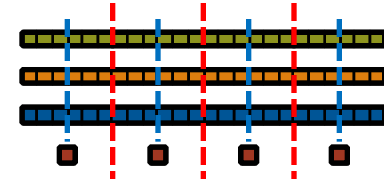
#ifdef _OPENMP
#pragma omp parallel for
#endif
    for (j=0; j<VectorSize; j++)
        a[j] = b[j]+scalar*c[j];

    HPCC_free(c);
    HPCC_free(b);
    HPCC_free(a);

    return 0;
}
```

*HPCC suffers from too many distinct notations for expressing parallelism and locality*

## CUDA



```
#define N 2000000

int main() {
    float *d_a, *d_b, *d_c;
    float scalar;

    cudaMalloc((void**) &d_a, sizeof(float)*N);
    cudaMalloc((void**) &d_b, sizeof(float)*N);
    cudaMalloc((void**) &d_c, sizeof(float)*N);

    dim3 dimBlock(128);
    dim3 dimGrid(N/dimBlock.x);

    set_array<<<dimGrid,dimBlock>>>(d_b, .5f, N);
    set_array<<<dimGrid,dimBlock>>>(d_c, .5f, N);

    scalar=3.0f;
    STREAM_Triad<<<dimGrid,dimBlock>>>(d_b, d_c, d_a, scalar, N);
    cudaThreadSynchronize();

    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    __global__ void set_array(float *a, float value, int len) {
        int idx = threadIdx.x + blockIdx.x * blockDim.x;
        if (idx < len) a[idx] = value;
    }

    __global__ void STREAM_Triad( float *a, float *b, float *c,
                                float scalar, int len) {
        int idx = threadIdx.x + blockIdx.x * blockDim.x;
        if (idx < len) c[idx] = a[idx]+scalar*b[idx];
    }
}
```



# STREAM Triad: Chapel

## MPI + OpenMP

```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params,
    int myRank, commSize;
    int rv, errCount;
    MPI_Comm comm = MPI_COMM_WORLD;

    MPI_Comm_size( comm, &commSize );
    MPI_Comm_rank( comm, &myRank );

    rv = HPCC_Stream( params, 0 == myRank
    MPI_Reduce( &rv, &errCount, 1, MPI_
    return errCount;
}

int HPCC_Stream(HPCC_Params *params,
    register int j;
    double scalar;

    VectorSize = HPCC_LocalVectorSize(
    a = HPCC_XMALLOC( double, VectorSize
    b = HPCC_XMALLOC( double, VectorSize
    c = HPCC_XMALLOC( double, VectorSize

    if (!a || !b || !c) {
        if (c) HPCC_free(c);
        if (b) HPCC_free(b);
        if (a) HPCC_free(a);
        if (doIO) {
            fprintf( outFile, "Failed to al
            fclose( outFile );
        }
    }
    return 1;
}
```

## Chapel

```
config const m = 1000,
              alpha = 3.0;

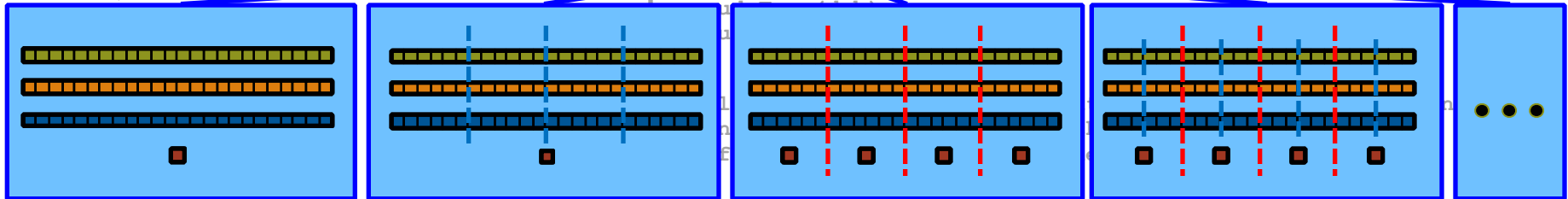
const ProblemSpace = {1..m} dmapped ...;

var A, B, C: [ProblemSpace] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

the special sauce



**Philosophy:** Good language design can tease details of locality and parallelism away from an algorithm, permitting the compiler, runtime, applied scientist, and HPC expert to each focus on their strengths.



# Motivating Chapel Themes

- 1) General Parallel Programming**
- 2) Global-View Abstractions**
- 3) Multiresolution Design**
- 4) Control over Locality/Affinity**
- 5) Reduce HPC  $\leftrightarrow$  Mainstream Language Gap**





# 1) General Parallel Programming

With a unified set of concepts...

...express any parallelism desired in a user's program

- **Styles:** data-parallel, task-parallel, concurrency, nested, ...
- **Levels:** model, function, loop, statement, expression

...target any parallelism available in the hardware

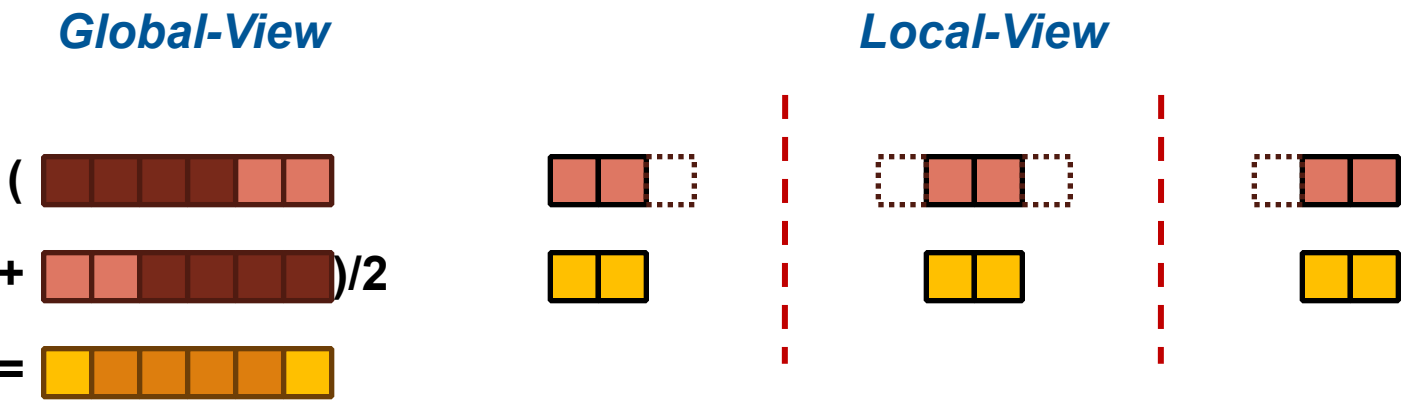
- **Types:** machines, nodes, cores, instruction

Type of HW Parallelism	Programming Model	Unit of Parallelism
<i>Inter-node</i>	<i>Chapel</i>	<i>task(or executable)</i>
<i>Intra-node/multicore</i>	<i>Chapel</i>	<i>iteration/task</i>
<i>Instruction-level vectors/threads</i>	<i>Chapel</i>	<i>iteration</i>
<i>GPU/accelerator</i>	<i>Chapel</i>	<i>SIMD function/task</i>



# 2) Global-View Abstractions

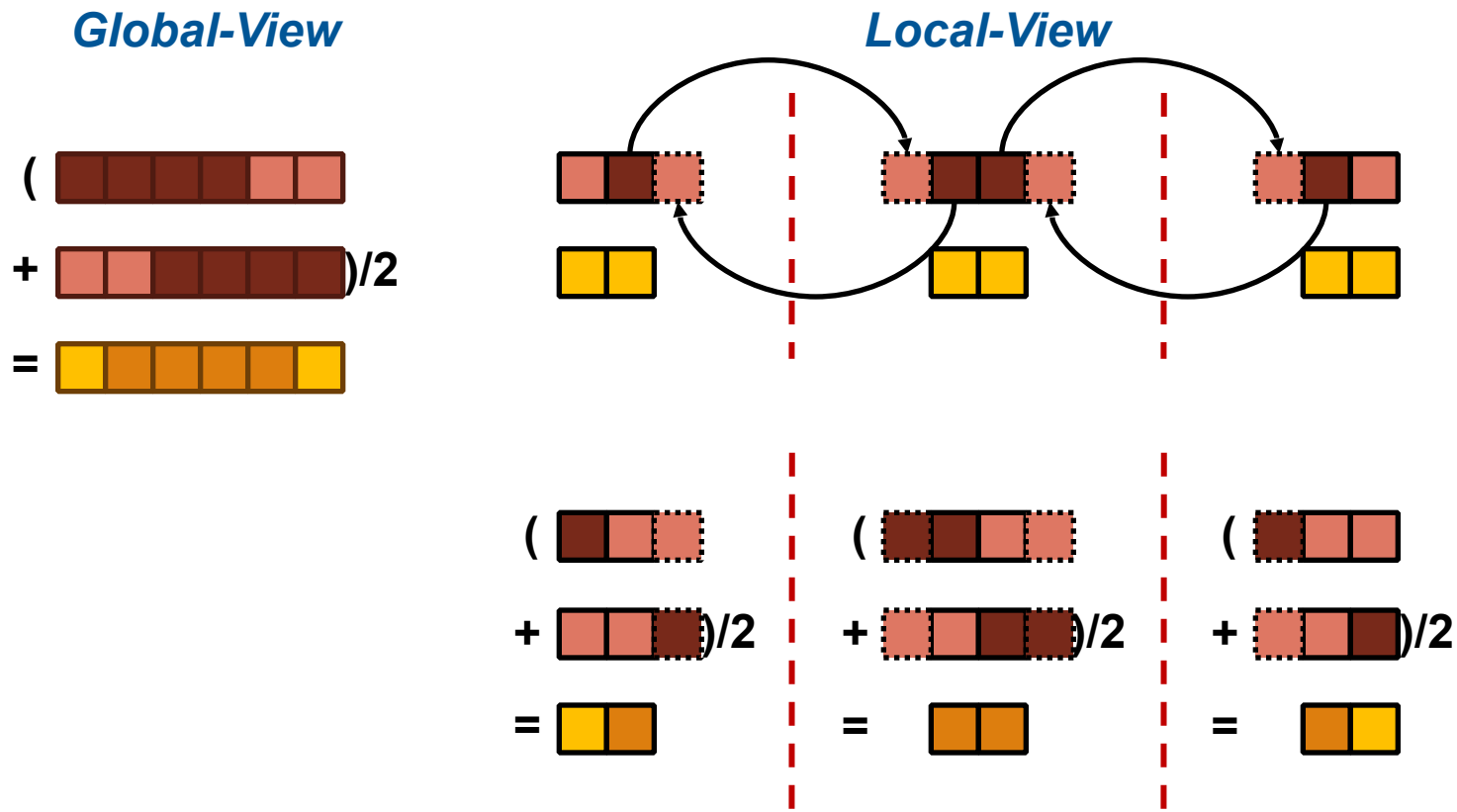
In pictures: “Apply a 3-Point Stencil to a vector”





## 2) Global-View Abstractions

In pictures: “Apply a 3-Point Stencil to a vector”

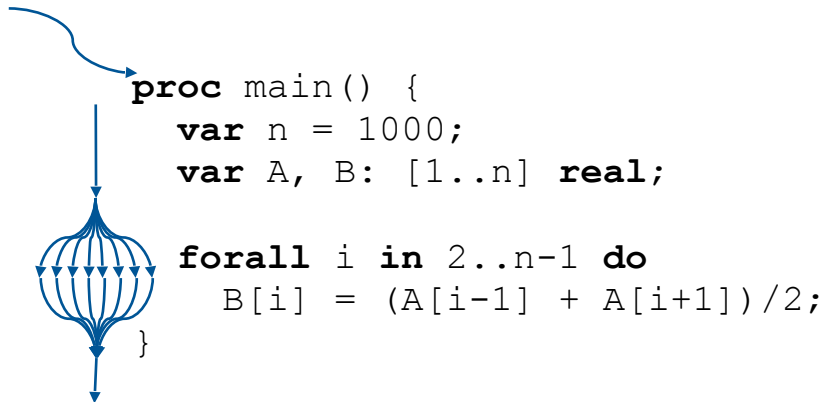




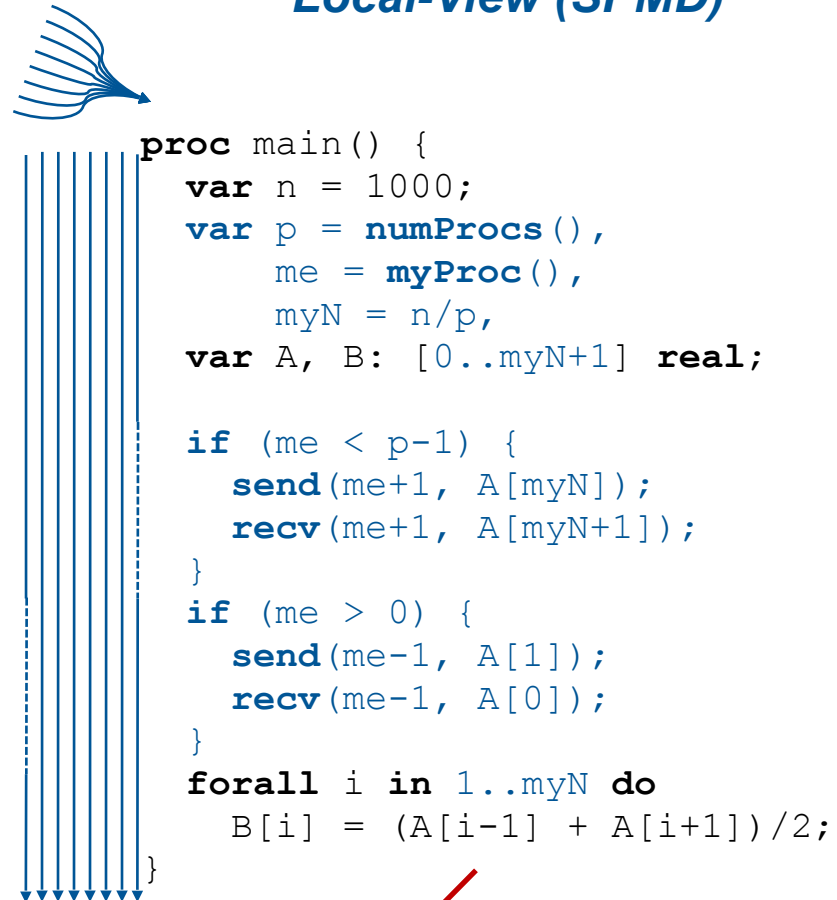
## 2) Global-View Abstractions

In code: “Apply a 3-Point Stencil to a vector”

### Global-View



### Local-View (SPMD)




Bug: Refers to uninitialized values at ends of A



## 2) Global-View Abstractions

In code: “Apply a 3-Point Stencil to a vector”

### Global-View



```

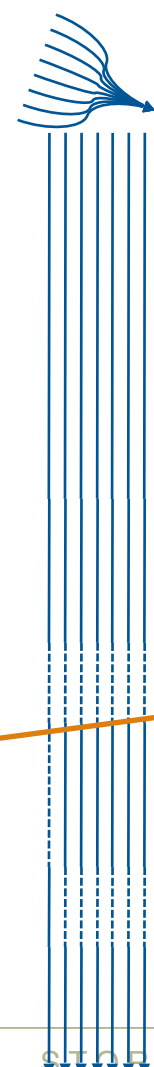
proc main() {
  var n = 1000;
  var A, B: [1..n] real;

  forall i in 2..n-1 do
    B[i] = (A[i-1] + A[i+1])/2;
  }

```

Communication becomes geometrically more complex for higher-dimensional arrays

### Local-View (SPMD)



```

proc main() {
  var n = 1000;
  var p = numProcs(),
      me = myProc(),
      myN = n/p,
      myLo = 1,
      myHi = myN;
  var A, B: [0..myN+1] real;

  if (me < p-1) {
    send(me+1, A[myN]);
    recv(me+1, A[myN+1]);
  } else
    myHi = myN-1;
  if (me > 0) {
    send(me-1, A[1]);
    recv(me-1, A[0]);
  } else
    myLo = 2;
  forall i in myLo..myHi do
    B[i] = (A[i-1] + A[i+1])/2;
  }

```

Assumes p divides n





## 2) Global-View Programming: A Final Note

- A language may support both global- and local-view programming — in particular, Chapel does

```
proc main() {  
    forall loc in Locales do  
        on loc do  
            MySPMDProgram(loc.id, Locales.numElements);  
        }  
    }  
  
proc MySPMDProgram(myImageID, numImages) {  
    ...  
}
```

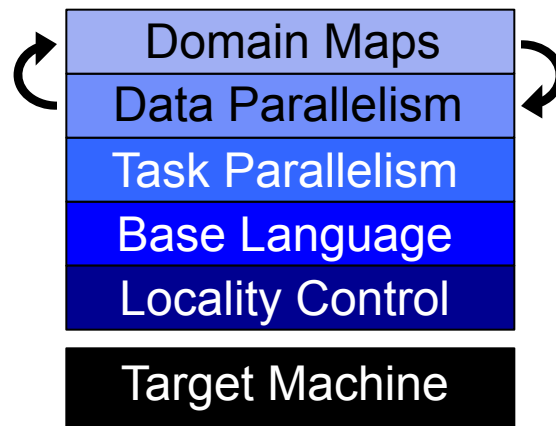


### 3) Multiresolution Design

#### ***Multiresolution Design:*** Support multiple tiers of features

- higher levels for programmability, productivity
- lower levels for greater degrees of control

#### *Chapel language concepts*



- build the higher-level concepts in terms of the lower
- permit the user to intermix layers arbitrarily





## 4) Control over Locality/Affinity

### Consider:

- Scalable architectures package memory near processors
- Remote accesses take longer than local accesses

### Therefore:

- Placement of data relative to tasks affects scalability
- Give programmers control of data and task placement

### Note:

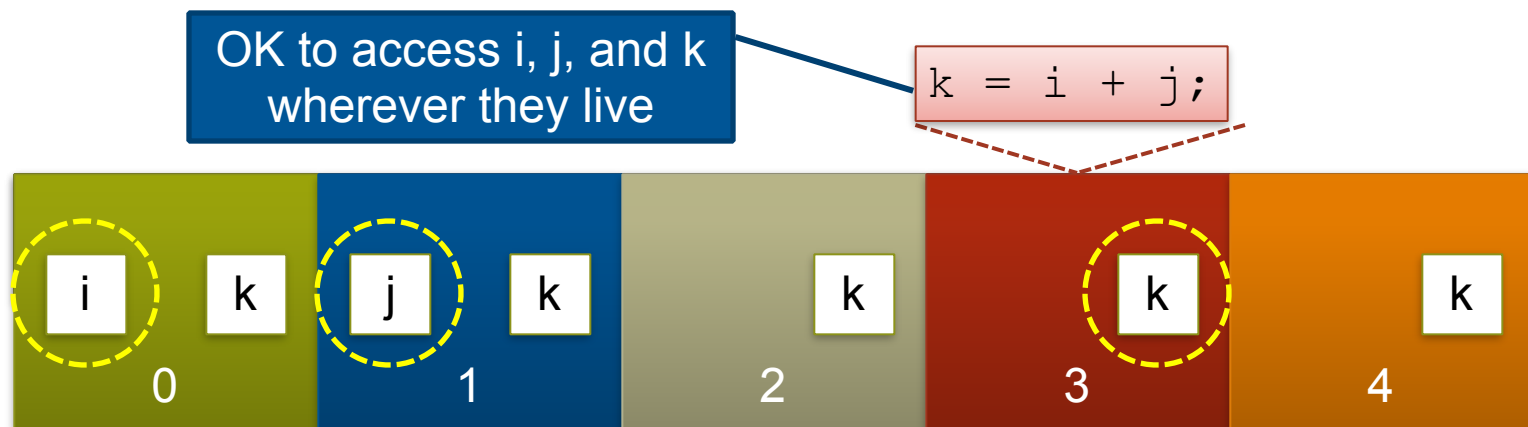
- Over time, we expect locality to matter more and more within the compute node as well



# PGAS Programming in a Nutshell

## Global Address Space:

- permit parallel tasks to access variables by naming them
  - regardless of whether they are local or remote
  - compiler / library / runtime will take care of communication



Images / Threads / Locales / Places / etc. (think: “compute nodes”)

COMPUTE | STORE | ANALYZE



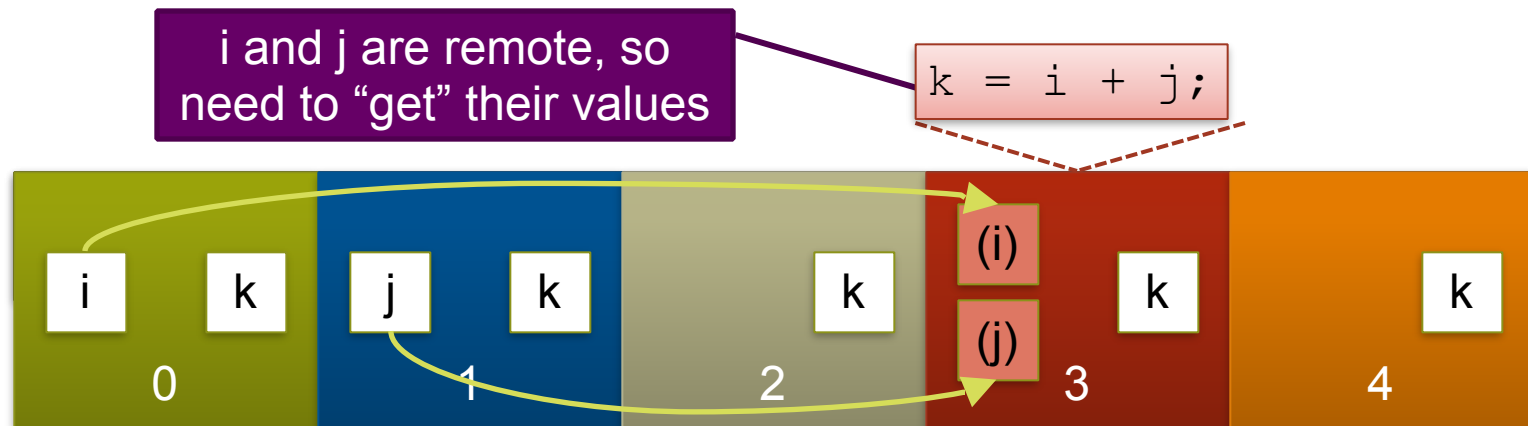
# PGAS Programming in a Nutshell

## Global Address Space:

- permit parallel tasks to access variables by naming them
  - regardless of whether they are local or remote
  - compiler / library / runtime will take care of communication

## Partitioned:

- establish a strong model for reasoning about locality
  - every variable has a well-defined location in the system
  - local variables are typically cheaper to access than remote ones



Images / Threads / Locales / Places / etc. (think: "compute nodes")



# PGAS Programming in a Nutshell

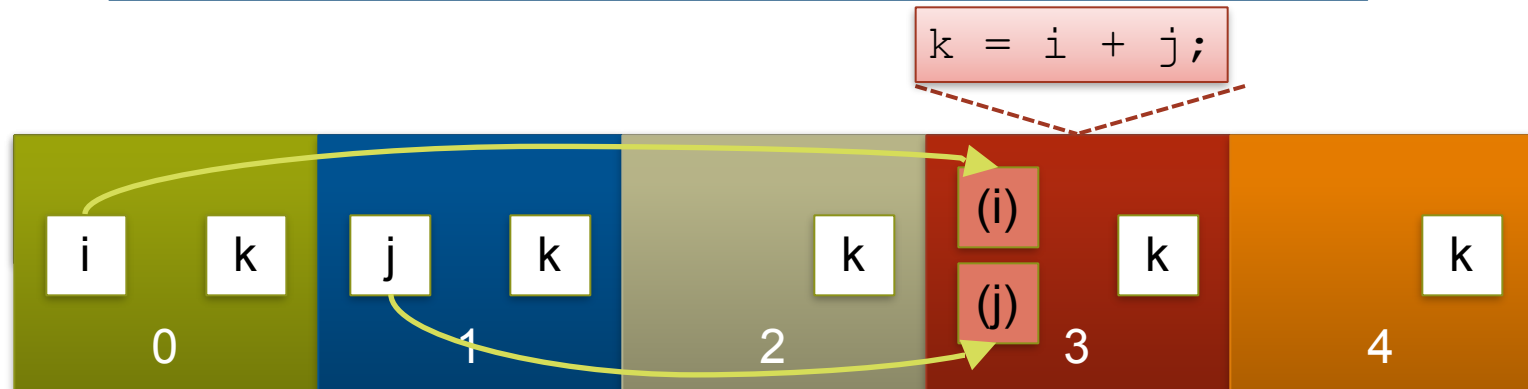
## Global Address Space:

- permit parallel tasks to access variables by naming them
  - regardless of whether they are local or remote
  - compiler / library / runtime will take care of communication

## Partitioned:

- establish
- even
- local

Communication is implicit!  
One sided GET and PUT.



Images / Threads / Locales / Places / etc. (think: “compute nodes”)

COMPUTE | STORE | ANALYZE



## 5) Reduce HPC ↔ Mainstream Language Gap



### Consider:

- Students graduate with training in Java, Matlab, Python, etc.
- Yet HPC programming is dominated by Fortran, C/C++, MPI

### We'd like to narrow this gulf with Chapel:

- to leverage advances in modern language design
- to better utilize the skills of the entry-level workforce...
- ...while not alienating the traditional HPC programmer
  - e.g., support object-oriented programming, but make it optional



# Processing Twitter @mentions Graphs in Chapel







# Processing Tweets: Motivation

## **Motivating Question:** Is Chapel useful for Data Analytics?

- What would it look like?
- What features are we missing?





# Processing Tweets: Background

**Twitter:** an online social networking service that enables users to send and read short 140-character messages called "tweets" -- Wikipedia

- tweets support referencing other users via `@username`



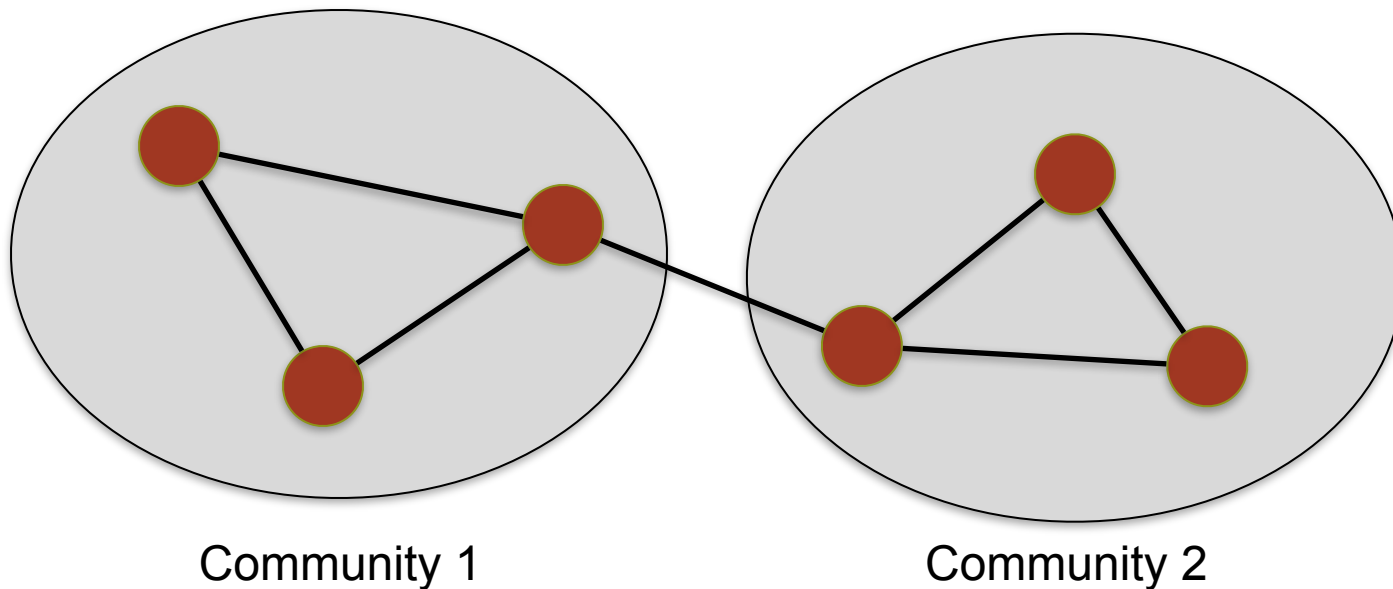
## Benchmark: Label Propagation for Community Detection

- can be considered to capture a data analytics workflow
- see CUG'15 paper: *Implementing a social-network analytics pipeline using Spark on Urika XA*
- a few implementations of this benchmark exist
  - e.g., Spark



# Processing Tweets: Computation Steps

- **Computation consists of these steps:**
  - Read in gzip files storing JSON-encoded tweets
  - Find pairs of Twitter users that @mention each other
  - Construct a graph from such users
  - Run a **label propagation algorithm** on that graph
  - Output the community structure resulting from label propagation





# Processing Tweets: Label Propagation

## Label Propagation Algorithm

(described in [Near linear time algorithm to detect community structures in large-scale networks](#))

1. Initialize the labels at all nodes in the network.
2. Set  $i = 1$ .
3. Arrange the nodes in the network in a random order and set it to  $X$ .
4. For each  $x$  in  $X$ , set node  $x$ 's label to the one that occurs most frequently among neighbors, with ties broken uniformly randomly.
5. If every node has a label that the maximum number of neighbors have, stop the algorithm. Otherwise, set  $i = i + 1$  and go to step 3.





# Processing Tweets: Implementation Overview

- **< 400 lines of Chapel code**
  - plus a Graph module ( < 300 lines, to become a standard module)
  - plus some improvements to existing Chapel modules
- **current version is single-locale**
  - ultimately, need to support multi-locale in order to run larger data sets
- **graph representation similar to other Chapel graph codes**
  - e.g., SSCA#2







# Processing Tweets: I/O

- Reading the tweets to build the graph is ~1/2 of the code
- Command line input lists files and directories to process
- `findfiles()` iterator used to enumerate files in a directory
- Reads file using `gunzip` via the new Spawn module
- **Uses new functionality for JSON I/O**
  - concept: use types and I/O that ignore irrelevant fields
    - (details in a sidebar following this section)







# Processing Tweets: Algorithm in Chapel

Algorithm closely matches the psuedocode:

```
var i = 0;
var go: atomic bool;
go.write(true);
while go.read(...) && i < maxiter {
    go.write(false);
    // for each x in the randomized order
    forall vid in reordered_vertices {
        // set the label to the most frequent among neighbors
        mylabel = labels[vid].read(memory_order_relaxed);
        maxlabel = mostCommonLabelInNeighbors(vid);
        if countNeighborsWith(vid, mylabel) <
            countNeighborsWith(vid, maxlabel) then
            go.write(true); // stop the algorithm if ...
        labels[vid].write(maxlabel, memory_order_relaxed);
    }
    i += 1;
}
```







# Processing Tweets: Caveats

The next few slides compare our Chapel version against a Spark version

## Important Notes:

- Spark includes resiliency features while Chapel currently does not
- neither implementation is necessarily optimal







# Processing Tweets: Productivity Comparison

## Spark

- **RDDs are immutable**
  - create new RDD every iteration through algorithm
- **Algorithm written in terms of mapping a fn on data**
  - difficult to visit vertices in random order
  - movement of information is described as messages contributing to a new RDD
  - breaking ties randomly might require a custom operator

## Chapel

- **Chapel arrays are mutable**
  - Algorithm can update labels in-place
- **Algorithm written in terms of parallel loops**
  - straightforward to visit vertices in random order
  - movement of information occurs through variable reads and writes
  - breaking ties randomly is an easy change

*These differences reflect Spark's declarative nature vs. Chapel's imperative design.*







# Processing Tweets: Performance Comparison

- **We performed an initial performance comparison between our Chapel version and the Spark version**
  - preliminary results are promising
- **However, there are several caveats:**
  - the results are completely apples-to-oranges:
    - different architectures
    - different system scales
    - different data set sizes(reflects Chapel code being single-locale only, early stages of study)
  - a multi-locale Chapel version will likely perform very differently
    - multi-locale execution will be necessary for larger dataset scales
- **For these reasons, we've decided not to release results until we can perform a more rigorous study**
  - specifically, multi-locale Chapel, same data set, same architecture





# Processing Tweets: Impact, Status, Next Steps

## Impact:

- A positive early indication of Chapel's applicability to data analytics

## Status:

- Have a prototype data analytics benchmark
  - reliant on pending modifications to Chapel library
- Productivity and performance are promising

## Next Steps:

- Commit library modifications to master
- Create a multi-locale version
  - primary effort: multi-locale graph data structures / domain maps
- Compare performance with other implementations, scientifically
- Describe this study in a paper to disseminate the results, get feedback



## Sidebar on I/O for Twitter Processing in Chapel







# Example Tweet in JSON format

- **Tweets have 34 top-level fields**
  - including nested structures containing much more data

```
{ "coordinates": null, "created_at": "Fri Oct 16 16:00:00 +0000 2015", "favorited": false, "truncated": false, "id_str": "28031452151", "entities":  
{ "urls": [ { "expanded_url": null, "url": "http://chapel.cray.com", "indices": [ 69, 100 ] } ], "hashtags": [ ], "user_mentions": [ { "name": "Cray  
Inc.", "id_str": "23424245", "id": 23424245, "indices": [ 25, 30 ], "screen_name": "cray" } ] }, "in_reply_to_user_id_str": null, "text": "Let's  
mention the user @cray – here is an embedded url ..... http://chapel.cray.com", "contributors": null, "id": 28039652140, "retweet_count":  
null, "in_reply_to_status_id_str": null, "geo": null, "retweeted": false, "in_reply_to_user_id": null, "user": { "profile_sidebar_border_color":  
"C0DEED", "name": "Cray Inc.", "profile_sidebar_fill_color": "DDEEF6", "profile_background_tile": false, "profile_image_url": "http://  
a3.twimg.com/profile_images/2342452/icon_normal.png", "location": "Seattle, WA", "created_at": "Fri Oct 10 23:10:00 +0000 2008", "id_str":  
"23502385", "follow_request_sent": false, "profile_link_color": "0084B4", "favourites_count": 1, "url": "http://cray.com",  
"contributors_enabled": false, "utc_offset": -25200, "id": 23548250, "profile_use_background_image": true, "listed_count": 23, "protected":  
false, "lang": "en", "profile_text_color": "333333", "followers_count": 1000, "time_zone": "Mountain Time (US & Canada)", "verified": false,  
"geo_enabled": true, "profile_background_color": "C0DEED", "notifications": false, "description": "Cray Inc.", "friends_count": 71,  
"profile_background_image_url": "http://s.twimg.com/a/2349257201/images/themes/theme1/bg.png", "statuses_count": 302,  
"screen_name": "gnip", "following": false, "show_all_inline_media": false }, "in_reply_to_screen_name": null, "source": "web", "place": null,  
"in_reply_to_status_id": null }
```







# Reading JSON Tweets

*// define Chapel records whose fields reflect only  
// the portions of the JSON data we care about*

```
record TweetUser {  
    var id: int;  
}  
record TweetEntities {  
    var user_mentions: list(TweetUser);  
}  
record User {  
    var id: int;  
}  
record Tweet {  
    var id: int,  
        user: User,  
        entities: TweetEntities;  
}
```

```
proc process_json(...) {  
    var tweet: Tweet;  
  
    while true {  
        // “%~jt” format string:  
        //   j: JSON format  
        //   t: any record  
        //   ~: skip other fields  
        got = logfile.readf("%~jt",  
                                tweet,  
                                error=err);  
  
        if got && !err then  
            handle_tweet(tweet);  
        if err == EFORMAT then ...;  
        if err == EOF then break;  
    }
```







# Open Issue: How to Read Arrays from JSON

## Current approach:

```
record TweetEntities {  
    var user_mentions: list(TweetUser);  
}
```

## Desired approach:

```
record TweetEntities {  
    var user_mentions: [1..0] TweetUser;  
    // not possible to know array's length in advance; determined by file contents  
    // would like a way to read this record that resizes the array appropriately...  
}
```







# 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, URIKA, and YARCDATA. 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 2014 Cray Inc.*





# CRAY



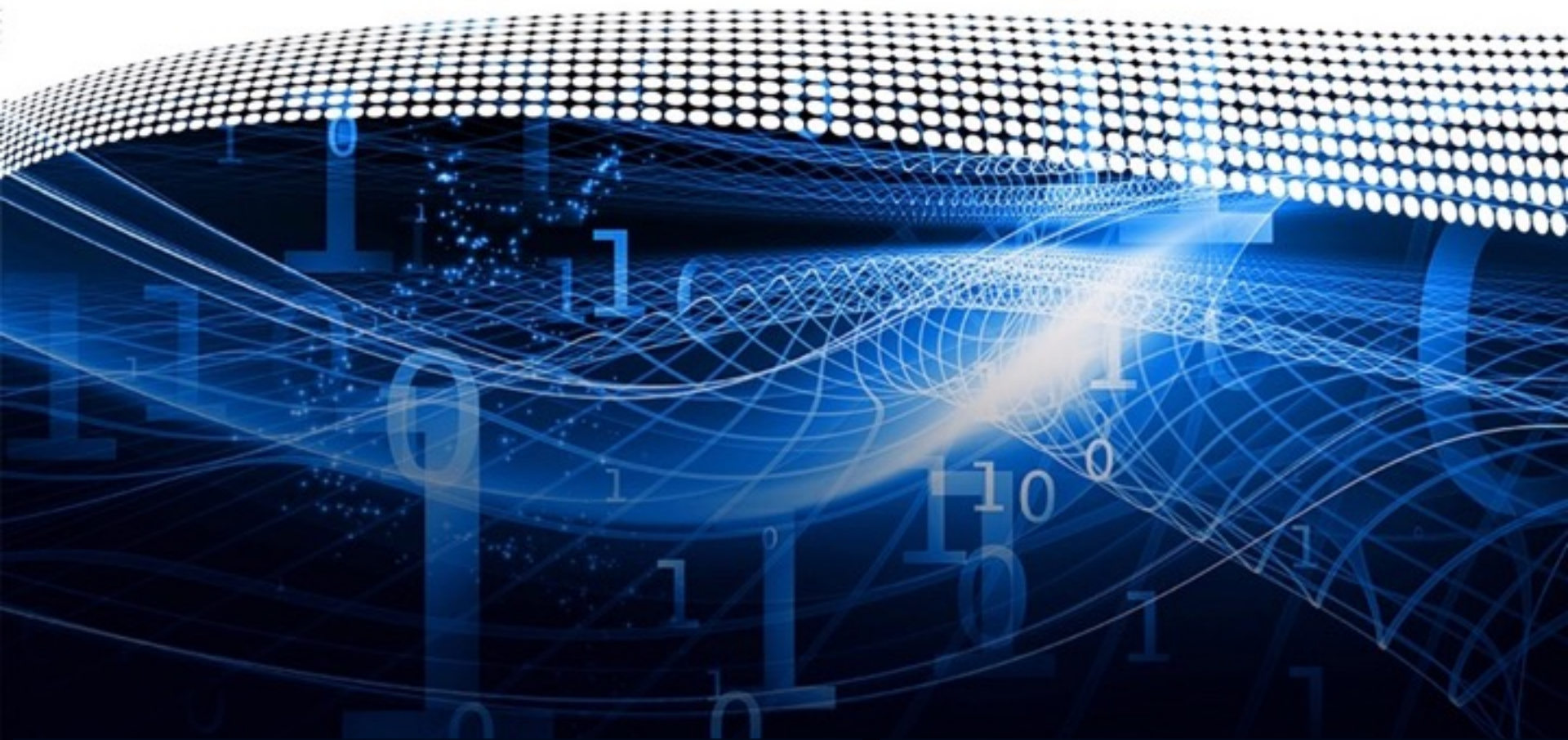
COMPUTE

|

STORE

|

ANALYZE



<http://chapel.cray.com>

[chapel\\_info@cray.com](mailto:chapel_info@cray.com)

<http://github.com/chapel-lang/chapel/>