RWTH AACHEN UNIVERSITY

Seminar: Languages for Scientific Computing

Chapel

Dominik Stengele

2013.12.05

Chapel?

- **C**ascade **H**igh-**P**roductivity **L**anguage
- Parallel programming language
- Developed by Cray Inc.
- Open source project (BSD license)
- Target architectures:
  - Architectures from Cray Inc. and other vendors
  - Multicore Desktops/Laptops
  - In progress: CPU+Accelerator hybrids

- Motivation? …

Sustained Performance Milestones

1 GF – 1988: Cray Y-MP; 8 Processors

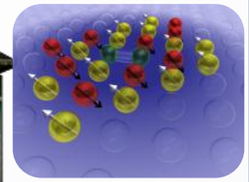
- Static finite element analysis
- Fortran77 + Cray autotasking + vectorization

1 TF – 1998: Cray T3E; 1,024 Processors

- Modeling of metallic magnet atoms
- Fortran + MPI (Message Passing Interface)

1 PF – 2008: Cray XT5; 150,000 Processors

- Superconductive materials
- C++/Fortran + MPI + vectorization



1 EF – ~2018: Cray ____; ~10,000,000 Processors

- TBD
- TBD: C/C++/Fortran + MPI + CUDA/OpenCL/OpenMP/OpenACC

Or Perhaps Something Completely Different?

Language features

- Static type inference

```
const pi = 3.14, // real
c1 = 8.2 + 3.2i, // complex
c2 = pi*c1;      // complex
```

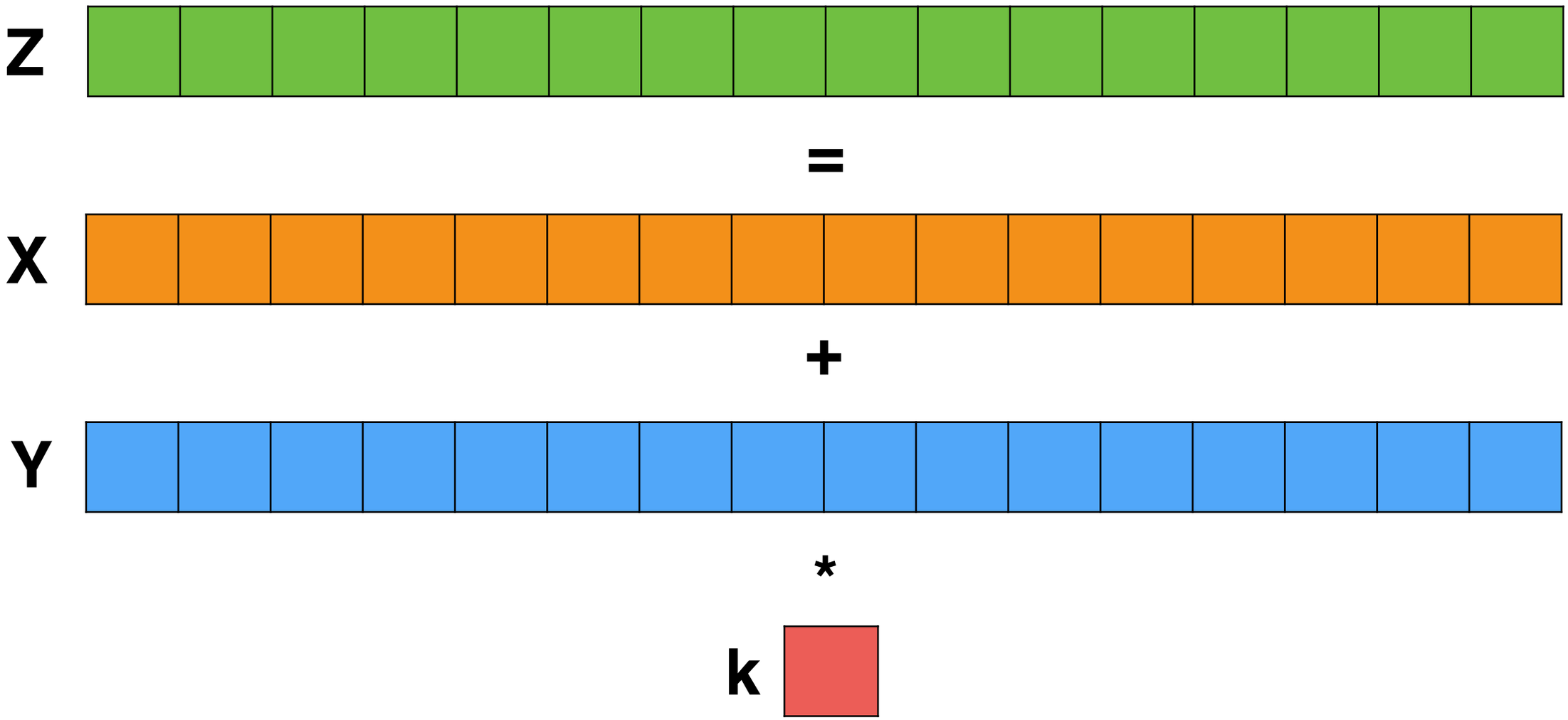
- Generic function arguments
- Range types

```
const r = 1..10 by 2;
```

- Iterators
- Complex numbers
- Tuple types
- OOP
- …

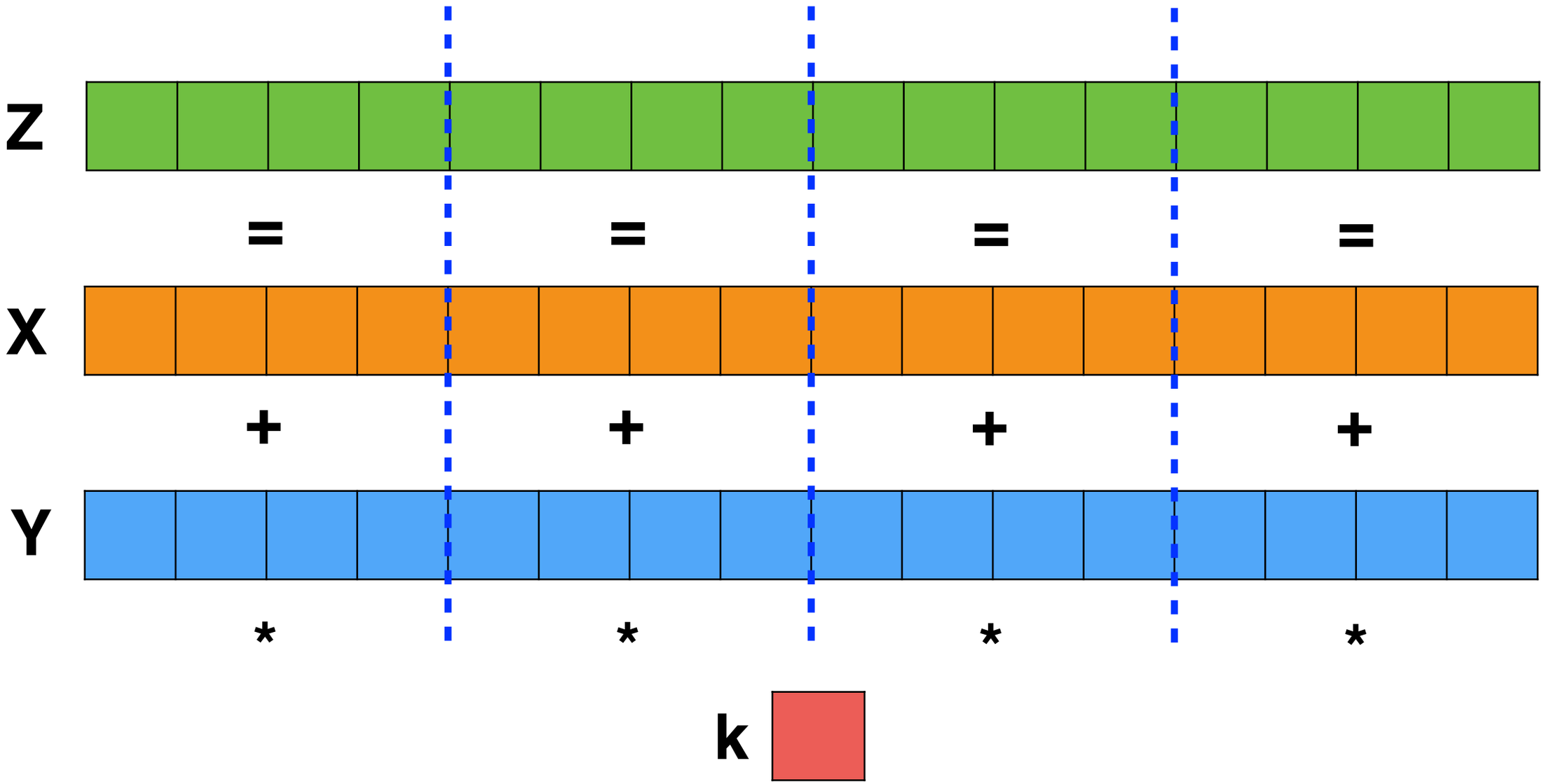
Example

Z = X + Y * k



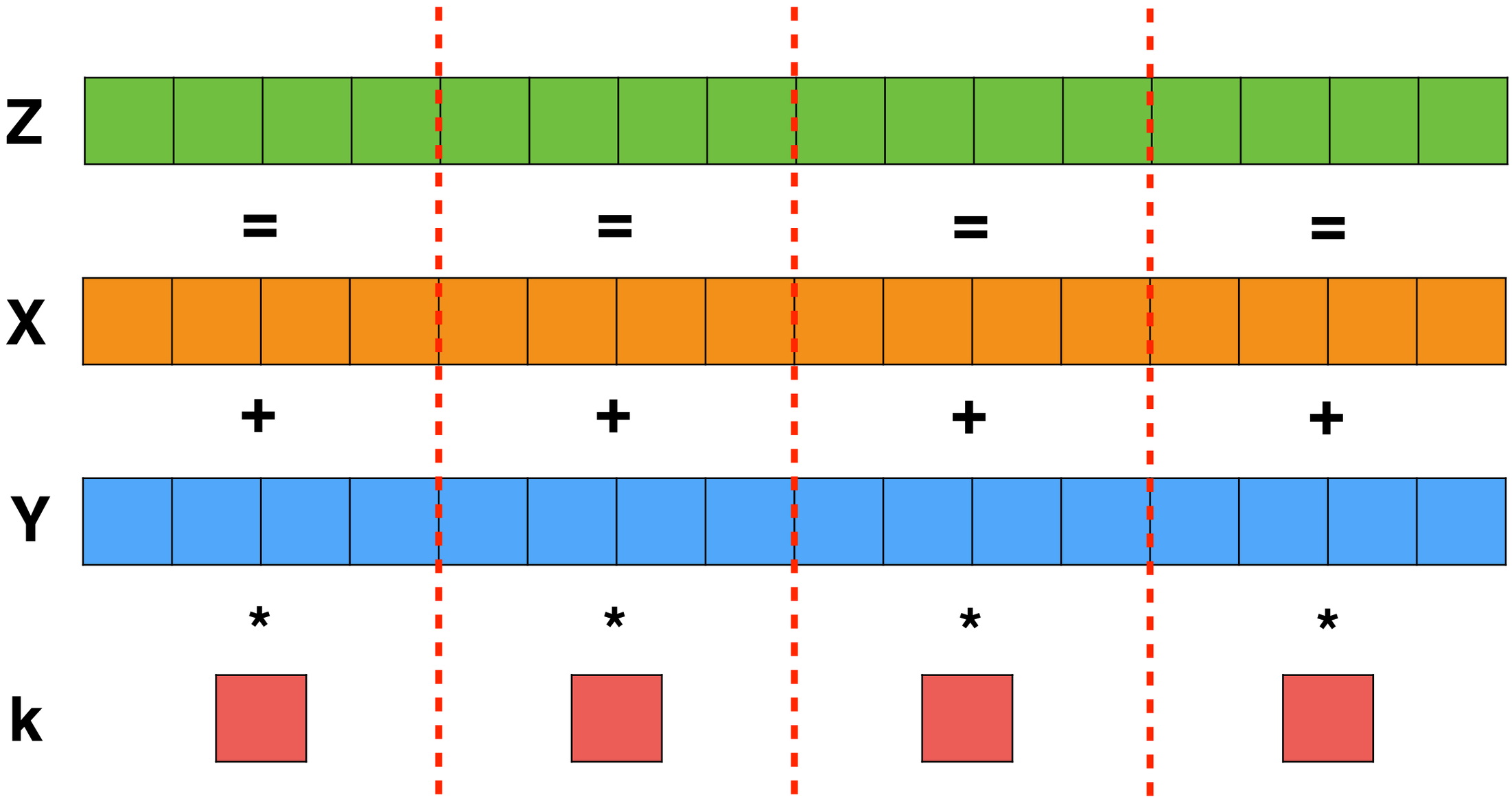
Example

- In parallel (multicore)



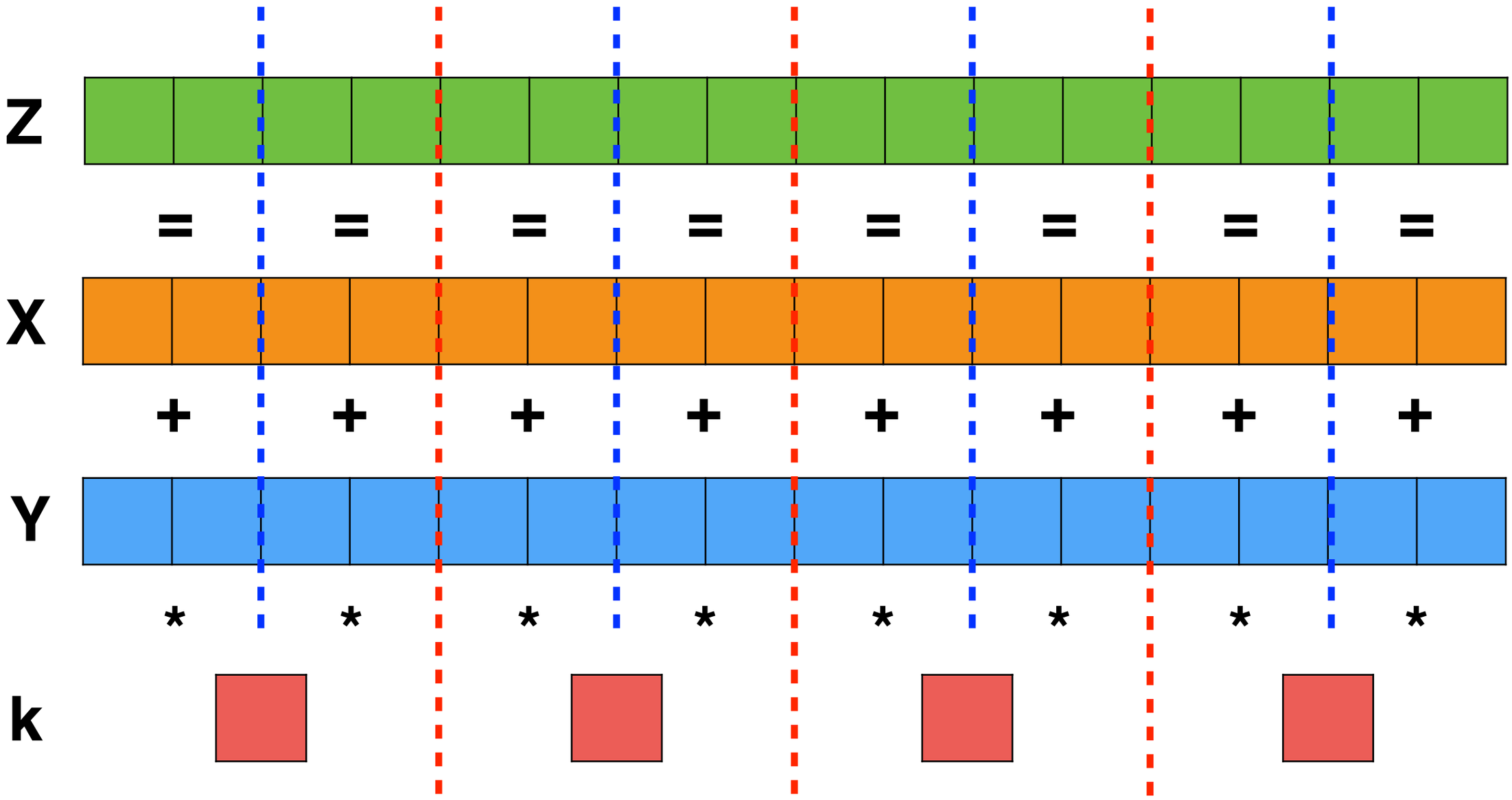
Example

- In parallel (distributed memory)



Example

- In parallel (distributed memory & multicore)



Example: Implementation

- C + MPI + OpenMP

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

static int VectorSize;
static double *Z, *X, *Y;

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 k;
    VectorSize = HPCC_LocalVectorSize(params, 3, sizeof(double), 0);
    Z = HPCC_XMALLOC(double, VectorSize);
    X = HPCC_XMALLOC(double, VectorSize);
    Y = HPCC_XMALLOC(double, VectorSize);
    if(!Z || !X || !Y)
    {
        if(Z)
            HPCC_free(Z);
        if(X)
            HPCC_free(X);
        if(Y)
            HPCC_free(Y);
        if(doIO)
        {
            // Failed to allocate memory
        }
        return 1;
    }

    #ifdef _OPENMP
    #pragma omp parallel for
    #endif
    for (j = 0; j < VectorSize; ++j)
    {
        X[j] = 2.0;
        Y[j] = 3.0;
    }
    k = 5.0;

    #ifdef _OPENMP
    #pragma omp parallel for
    #endif
    for(j = 0; j < VectorSize; ++j)
        Z[j] = X[j]+Y[j]*k;

    HPCC_free(Z);
    HPCC_free(X);
    HPCC_free(Y);
    return 0;
}
```

Example: Implementation

- C + CUDA

```
#define N 2000000

int main()
{
    float *d_Z, *d_X, *d_Y;
    float k;

    cudaMalloc((void**)&d_Z, sizeof(float)*N);
    cudaMalloc((void**)&d_X, sizeof(float)*N);
    cudaMalloc((void**)&d_Y, sizeof(float)*N);

    dim3 dimBlock(128);
    dim3 dimGrid(N/dimBlock.x);
    if(N % dimBlock.x != 0)
        dimGrid.x+=1;

    set_array<<<dimGrid,dimBlock>>>(d_X, 2.0f, N);
    set_array<<<dimGrid,dimBlock>>>(d_Y, 3.0f, N);
    k = 5.0f;
    STREAM_Triad<<<dimGrid,dimBlock>>>(d_Z, d_X, d_Y, k, N);
    cudaThreadSynchronize();

    cudaFree(d_Z);
    cudaFree(d_X);
    cudaFree(d_Y);
    return 0;
}

__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* Z, float* X, float* Y, float k, int N)
{
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if(idx < N)
        Z[idx] = X[idx]+Y[idx]*k;
}
```

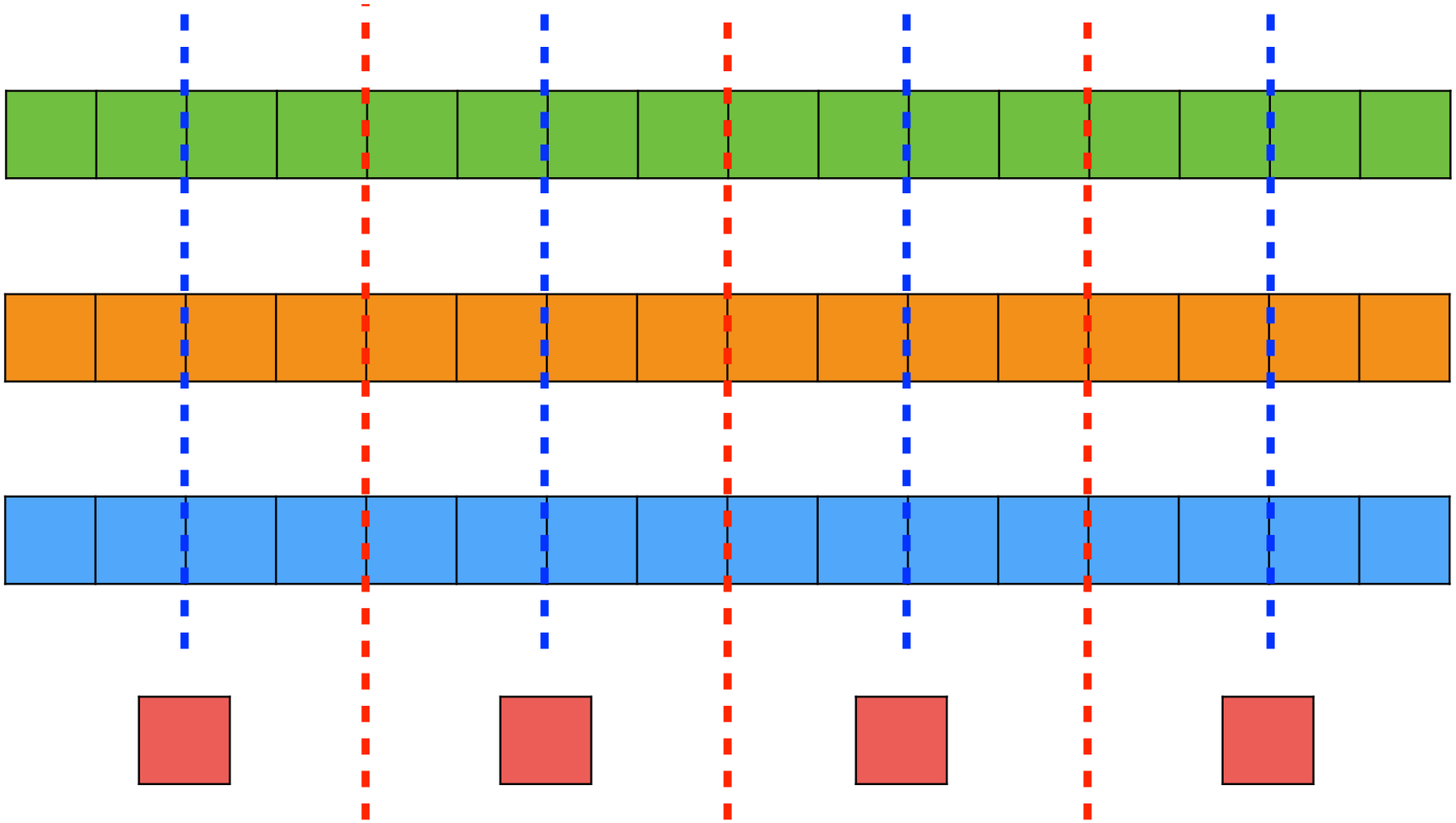
Example: Implementation

- Chapel

```
// chpl -o test test.chpl

use BlockDist;

config const N = 1000;

const ProblemSpace = {1..N} dmapped Block(boundingBox={1..N});

var Z, X, Y: [ProblemSpace] real;

X = 2.0;
Y = 3.0;
const k = 5.0;

Z = X + Y * k;
```

Domain Maps

- Instruct the compiler, how to map the global view to memory and locals



Domain Maps

Layouts

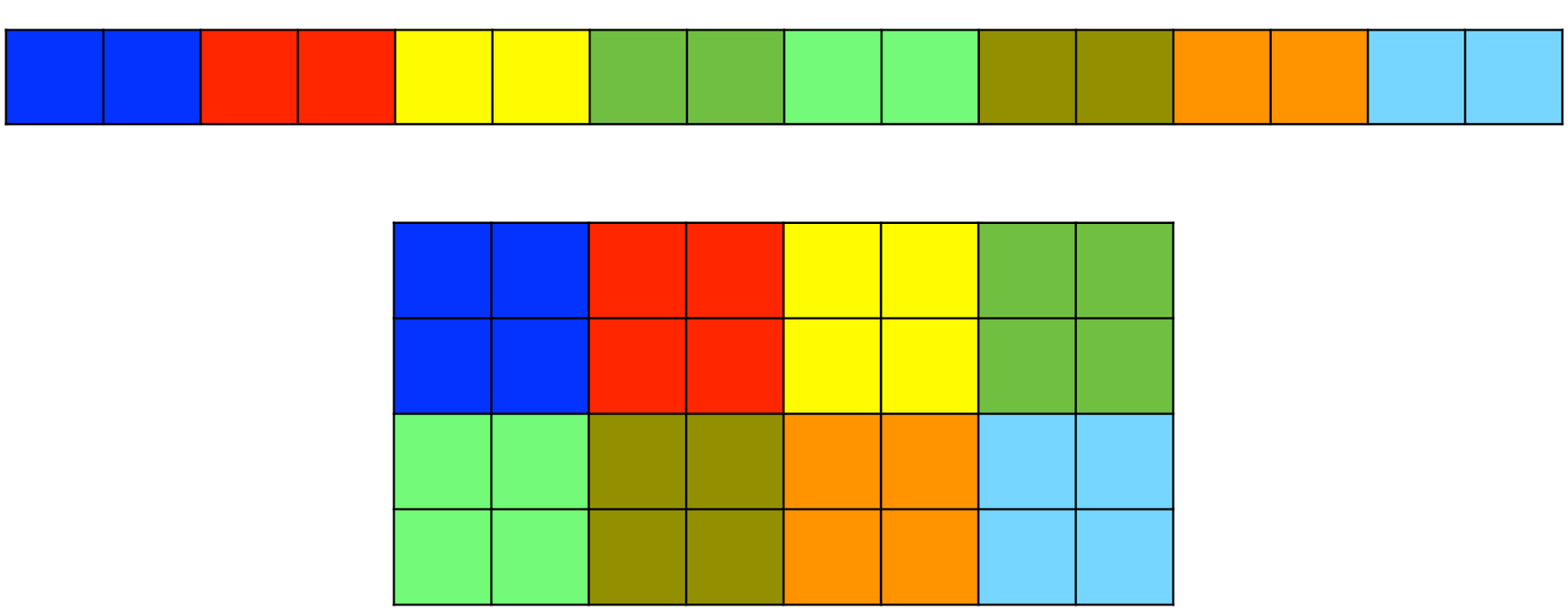
- **Single-Locale** domain maps
- Takes advantage of local resources (e.g. multiple cores)

Distributions

- **Multi-Locale** domain maps
- Controls data and tasks
- May use Layouts for local implementation
- `Block`
- `Cyclic`
- `Block-Cyclic`
- …

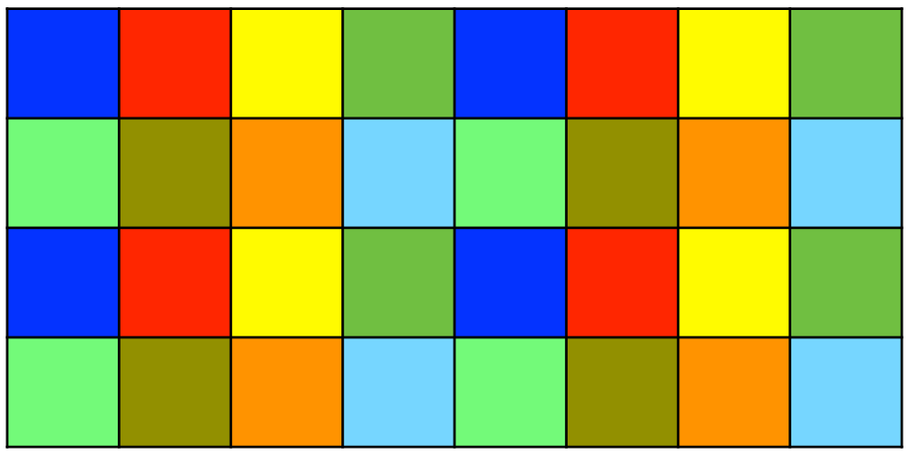
Standard Domain Maps

- Block Distribution
- Maps domain in dense fashion across the target Locales, according to `boundingBox`
- `Block(boundingBox=[1..4, 1..8])`



Standard Domain Maps

- Cyclic Distribution
- Maps the domain in a round-robin fashion across the target Locales according to `startIdx`
- `Cyclic(startIdx=(1, 1))`



begin/cobegin

- `begin` creates a new thread for a statement

```
begin writeln(“1”);
begin writeln(“2”);
writeln(“3”);          // Order of all lines is undefined
```

- `cobegin` behaves similar, but blocks mean thread

```
cobegin
{
    writeln(“1”);
    writeln(“2”);   // Order undefined
}
writeln(“3”);       // Guaranteed to be executed last
                    // Possible outputs: 1 2 3 or 2 1 3
```

forall/coforall

- `forall` creates a multithreaded for-loop
  - Number of threads depends on available cores
- `coforall` creates a new thread for every iteration of the loop
  - Behaves like `cobegin`, but order of statements within the loop is guaranteed

```
coforall i in 1..2
{
    write(“i = ”);
    writeln(i);
}
writeln(“3”);
```

References & Further reading

http://chapel.cray.com
Cray Inc.
Official Webpage

“The Chapel Parallel Programming Language”
B.L. Chamberlain
Presentation

“Parallel Programmability and the Chapel Language”
B.L. Chamberlain, D. Callahan, H.P. Zima
Paper

http://faculty.knox.edu/dbunde/teaching/chapel
Knox College
Tutorial

Demo: Jacobi Method

- Compute the solution of a Laplace equation using the Jacobi method
- Used in heat flow, electrostatics, gravity, …
- Equation is discretized using an (n+1)x(n+1) grid
- `X` stores approximate solution and is updated with new solution `XNew`
- Terminates when difference is sufficiently small

RWTH AACHEN UNIVERSITY

Seminar: Languages for Scientific Computing

Chapel

Dominik Stengele

2013.12.05