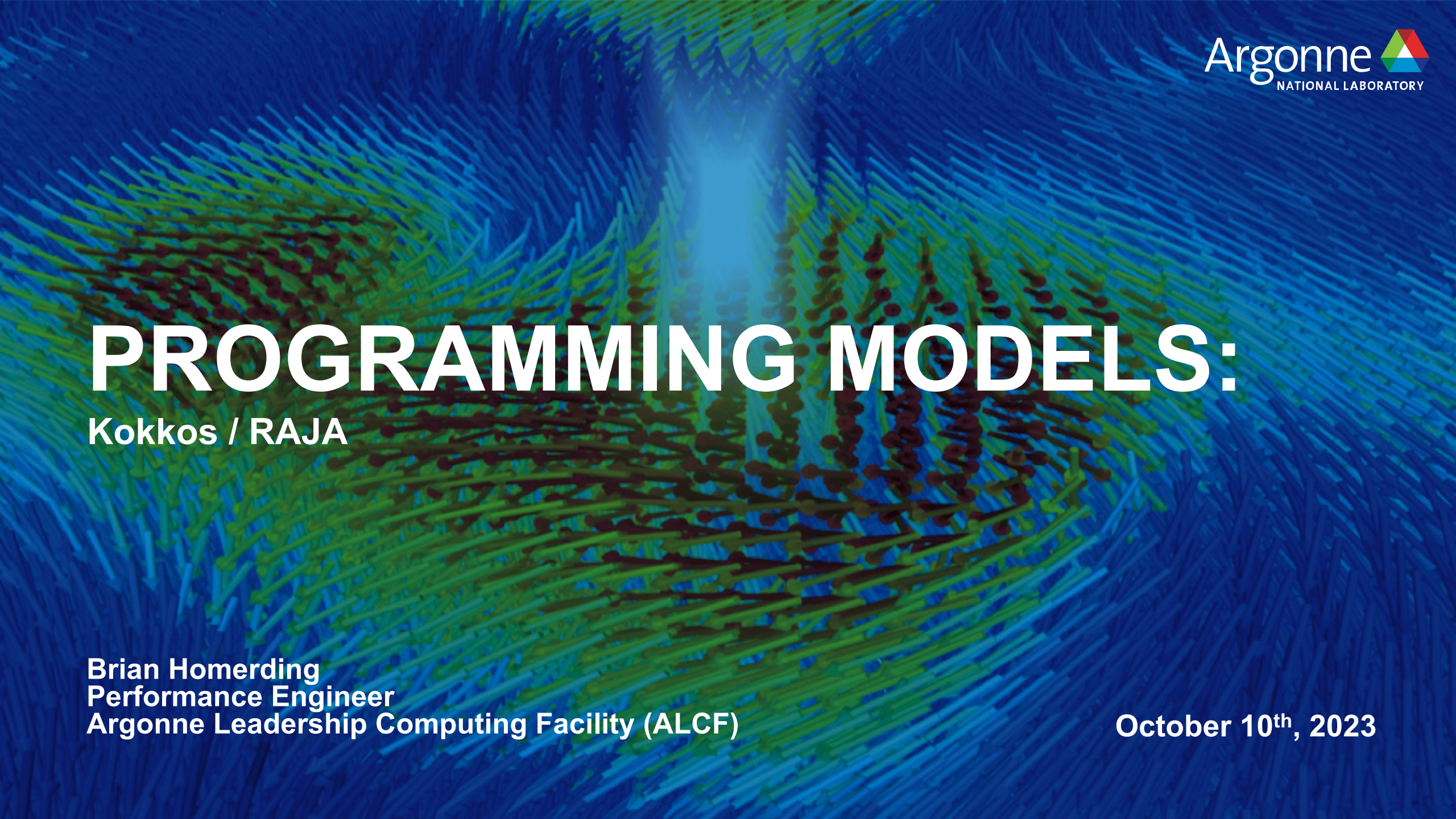

October 10-12, 2023
ALCF Hands-on HPC Workshop



Argonne NATIONAL LABORATORY

PROGRAMMING MODELS:

Kokkos / RAJA

Brian Homerding
Performance Engineer
Argonne Leadership Computing Facility (ALCF)

October 10th, 2023

SOME C++ CONCEPTS

TEMPLATES

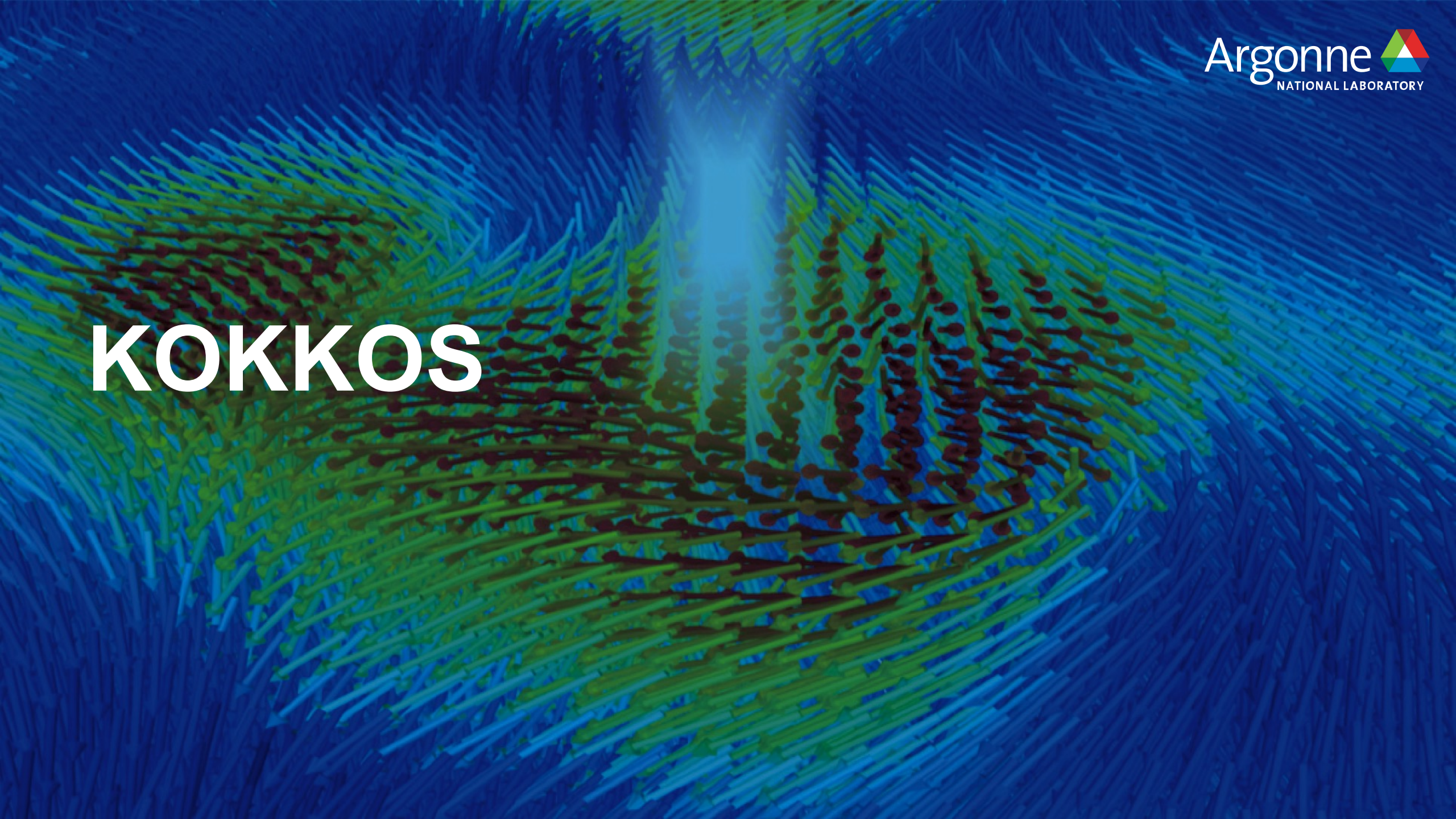
- Allows for parameterization based on template parameters.

```
template <typename T>
T templatedMax(T a, T b) {
  return (a > b) ? a : b;
}
```

LAMBDAS

- Constructs a closure. An unnamed function object.

```
auto lmbda = [&] (int b) {
  return (a > b) ? a : b;
}
a = {...}
newMax = lbmda(b);
```

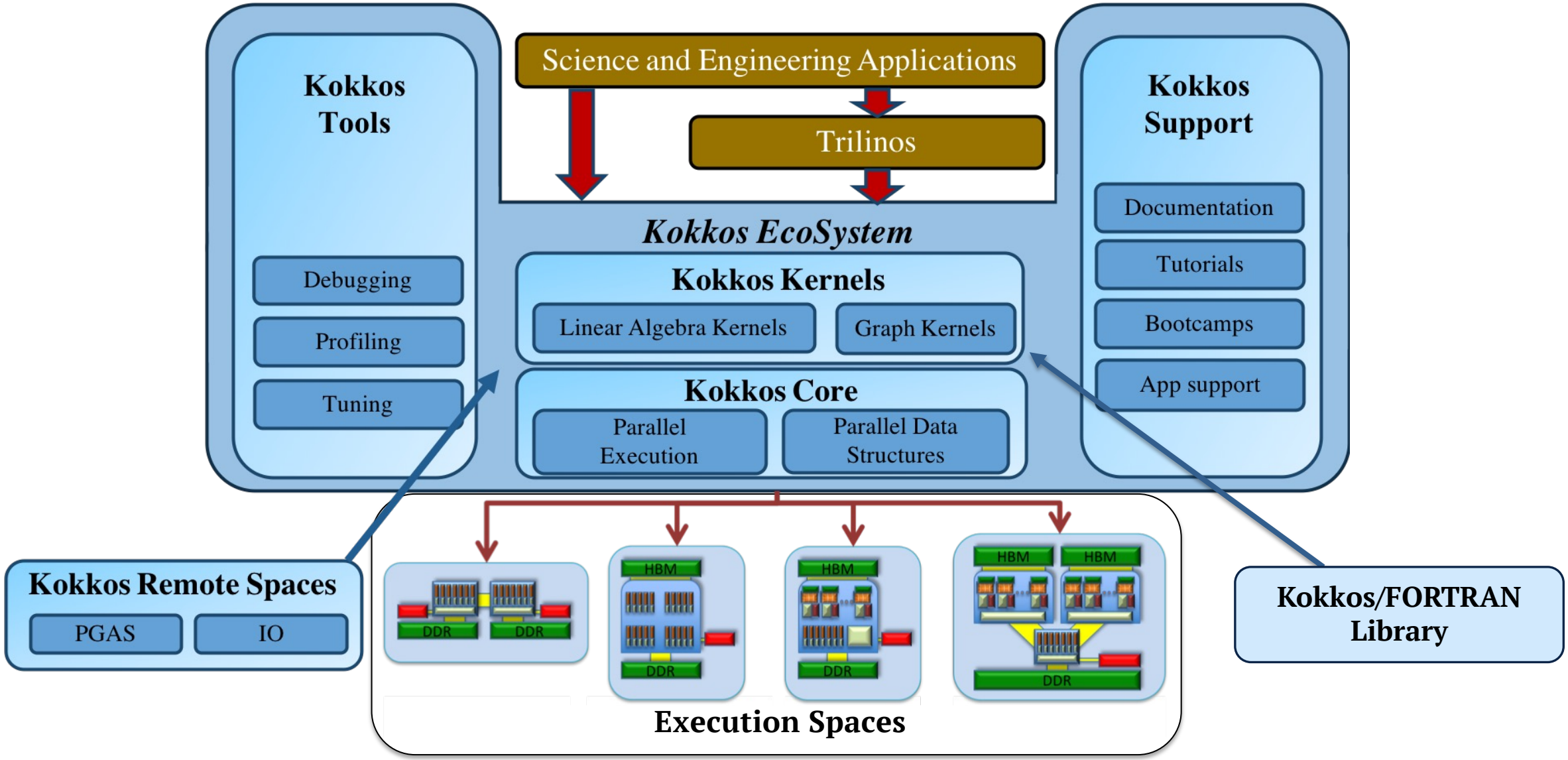

Argonne
NATIONAL LABORATORY
KOKKOS

What is Kokkos?

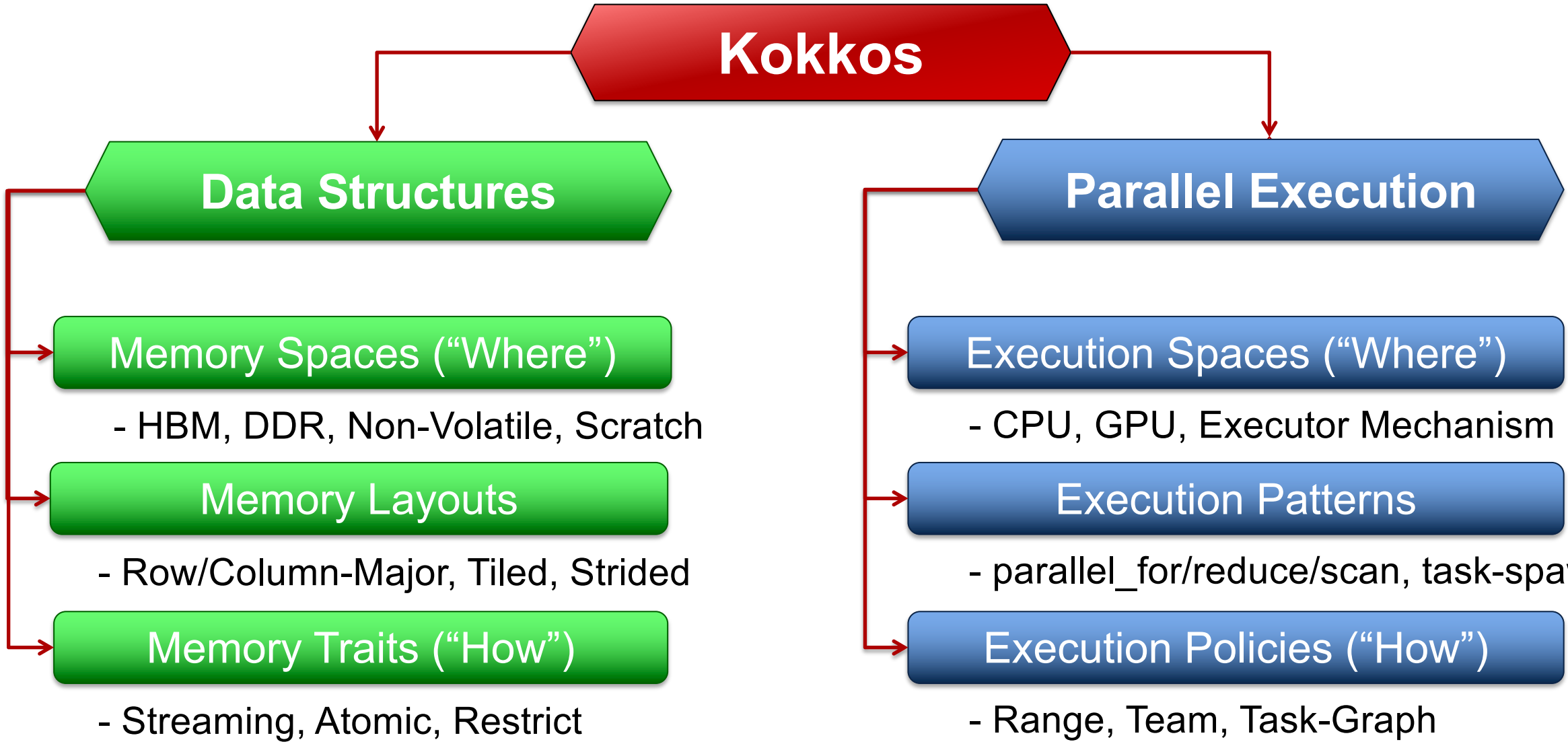
Sandia National Laboratories

- A C++ Programming Model for Performance Portability
  - Implemented as a template library on top of CUDA, OpenMP, HPX, …
  - Aims to be descriptive not prescriptive
  - Aligns with developments in the C++ standard
- Expanding solution for common needs of modern science/engineering codes
  - Math libraries based on Kokkos
  - Tools which allow inside into Kokkos
- It is Open Source
  - Maintained and developed at https://github.com/kokkos
- It has many users at wide range of institutions.

Kokkos EcoSystem



Kokkos Core Abstractions

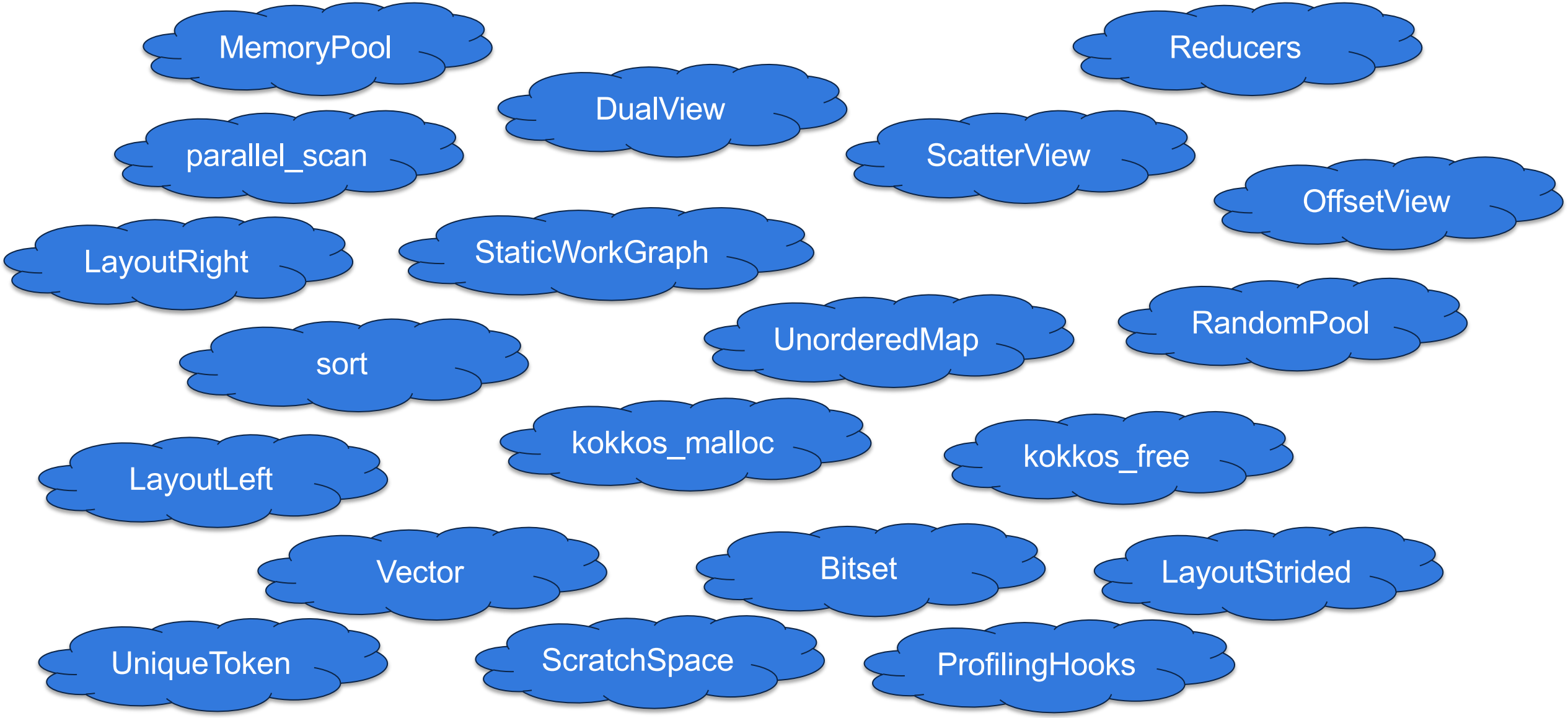


Kokkos Core Capabilities

Sandia National Laboratories

Concept	Example
Parallel Loops	**parallel_for**(N, **KOKKOS_LAMBDA** (int i) { ...BODY… });
Parallel Reduction	**parallel_reduce**(**RangePolicy**<ExecSpace>(0,N), **KOKKOS_LAMBDA** (int i, double& upd) { …BODY... upd += ... }, Sum<>(result));
Tightly Nested Loops	**parallel_for**(**MDRangePolicy**<**Rank**<3> > ({0,0,0},{N1,N2,N3},{T1,T2,T3}, **KOKKOS_LAMBDA** (int i, int j, int k) {…BODY...});
Non-Tightly Nested Loops	**parallel_for**(**TeamPolicy**<**Schedule**<**Dynamic**>>(N, TS), **KOKKOS_LAMBDA** (Team team) { … COMMON CODE 1 ... **parallel_for**(**TeamThreadRange**(team, M(N)), [&] (int j) { ... INNER BODY... }); … COMMON CODE 2 ... });
Task Dag	**task_spawn**(**TaskTeam**(scheduler , priority), **KOKKOS_LAMBDA** (Team team) { … BODY });
Data Allocation	**View**<double**, Layout, MemSpace> a(“A”,N,M);
Data Transfer	**deep_copy**(a,b);
Atomics	atomic_add(&a[i],5.0); View<double*,MemoryTraits<AtomicAccess>> a(); a(i)+=5.0;
Exec Spaces	Serial, Threads, OpenMP, Cuda, HPX (experimental), HIP (experimental), OpenMPTarget (experimental)

More Kokkos Capabilities



CG Solve: The AXPBY

- Simple data parallel loop: Kokkos::parallel_for
- Easy to express in most programming models
- Bandwidth bound
- Serial Implementation:

```
void axpby(int n, double* z, double alpha, const double* x,
                             double beta,  const double* y) {
  for(int i=0; i<n; i++)
    z[i] = alpha*x[i] + beta*y[i];
}
```

Parallel Pattern: for loop

String Label: Profiling/Debugging

Execution Policy: do n iterations

Loop Body

Iteration handle: integer index

- Kokkos Implementation:

```
void axpby(int n, View<double*> z, double alpha, View<const double*> x,
                                   double beta,  View<const double*> y) {
  parallel_for("AXpBY", n, KOKKOS_LAMBDA ( const int i) {
    z(i) = alpha*x(i) + beta*y(i);
  });
}
```

Kokkos Kernels

Sandia National Laboratories

- BLAS, Sparse and Graph Kernels on top of Kokkos and its View abstraction
  - Scalar type agnostic, e.g. works for any types with math operators
  - Layout and Memory Space aware
- Can call vendor libraries when available
- Views contain size and stride information => Interface is simpler

```
// BLAS
int M,N,K,LDA,LDB; double alpha, beta; double *A, *B, *C;
dgemm('N','N',M,N,K,alpha,A,LDA,B,LDB,beta,C,LDC);
```

```
// Kokkos Kernels
double alpha, beta; View<double**> A,B,C;
gemm('N','N',alpha,A,B,beta,C);
```

- Interface to call Kokkos Kernels at the teams level (e.g. in each CUDA-Block)

```
parallel_for("NestedBLAS", TeamPolicy<>(N,AUTO), KOKKOS_LAMBDA (const team_handle_t& team_handle) {
  // Allocate A, x and y in scratch memory (e.g. CUDA shared memory)
  // Call BLAS using parallelism in this team (e.g. CUDA block)
  gemv(team_handle,'N',alpha,A,x,beta,y)
});
```

Kokkos Tools

Sandia National Laboratories

- Profiling
  - New tools are coming out
  - Worked with NVIDIA to get naming info into their system
- Auto Tuning (Under Development)
  - Internal variables such as CUDA block sizes etc.
  - User provided variables
  - Same as profiling: will use dlopen to load external tools
- Debugging (Under Development)
  - Extensions to enable clang debugger to use Kokkos naming information
- Static Analysis (Under Development)
  - Discover Kokkos anti patterns via clang-tidy

Kokkos Tools Static Analysis

Sandia National Laboratories

- clang-tidy passes for Kokkos semantics
- Under active development, requests welcome
- IDE integration

```
// Base case
Kokkos::parallel_for(
    TPolicy, KOKKOS_LAMBDA(TeamMember const& t) {
      int a = 0;

      Kokkos::parallel_for(TTR(t, 1), [&](int i) { Lambda capture modifies reference capture variable 'a' that is a local
        a += 1;
        cv() += 1;
      });
    });

// One with variable Lambda
Kokkos::parallel_for(
    TPolicy, KOKKOS_LAMBDA(TeamMember const& t) {
      int b      = 0;
      auto lambda = [&](int i) { Lambda capture modifies reference capture variable 'b' that is a local
        b += 1;
        cv() += 1;
      };
      Kokkos::parallel_for(TTR(t, 1), lambda);
    });
```

Some Kokkos Users

Sandia National Laboratories

Pacific Northwest NATIONAL LABORATORY

Atlas 3D

Los Alamos NATIONAL LABORATORY EST. 1943

Sandia National Laboratories

ARL

MICHIGAN STATE

NREL NATIONAL RENEWABLE ENERGY LABORATORY

OAK RIDGE National Laboratory

Argonne NATIONAL LABORATORY

U.S. NAVAL RESEARCH LABORATORY

CSCS

THE UNIVERSITY OF UTAH

DE LA RECHERCHE À L'INDUSTRIE cea

JÜLICH Forschungszentrum

TECHNISCHE UNIVERSITÄT MÜNCHEN

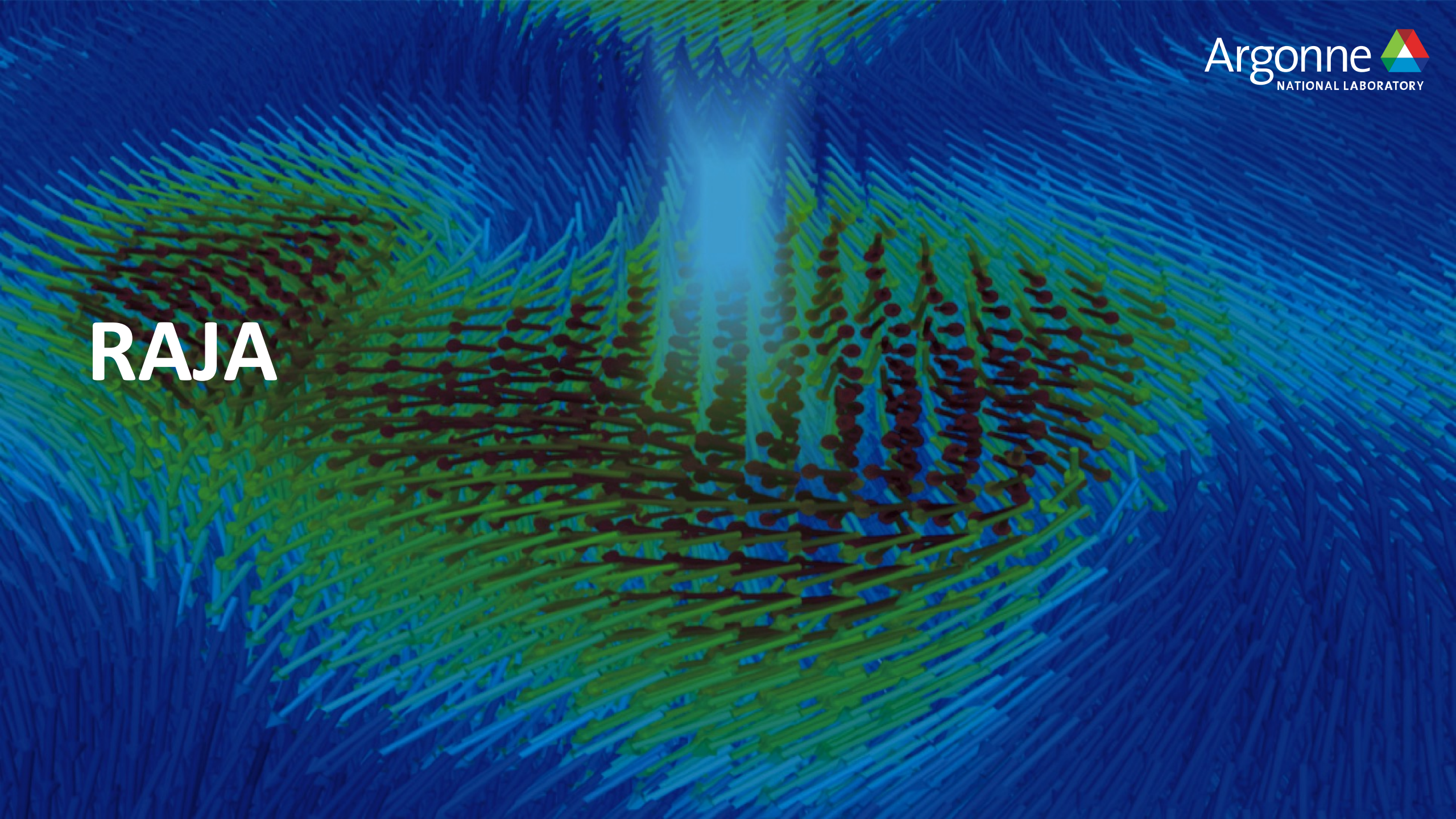
IPP Max-Planck-Institut für Plasmaphysik

BERKELEY LAB

Rensselaer

Links

- https://github.com/kokkos Kokkos Github Organization
  - **Kokkos:** *Core library, Containers, Algorithms*
  - **Kokkos-Kernels:** *Sparse and Dense BLAS, Graph, Tensor (under development)*
  - **Kokkos-Tools:** *Profiling and Debugging*
  - **Kokkos-MiniApps:** *MiniApp repository and links*
  - **Kokkos-Tutorials:** *Extensive Tutorials with Hands-On Exercises*
- https://cs.sandia.gov Publications (search for 'Kokkos')
  - Many Presentations on Kokkos and its use in libraries and apps
- http://on-demand-gtc.gputechconf.com Recorded Talks
  - Presentations with Audio and some with Video
- https://kokkosteam.slack.com Slack channel for user support


Argonne
NATIONAL LABORATORY
RAJA

RAJA and performance portability

- RAJA is a **library of C++ abstractions** that enable you to write **portable, single-source** kernels – run on different hardware by re-compiling
  - Multicore CPUs, Xeon Phi, NVIDIA GPUs, …
- RAJA **insulates application source code** from hardware and programming model-specific implementation details
  - OpenMP, CUDA, SIMD vectorization, …
- RAJA supports a variety of **parallel patterns** and **performance tuning** options
  - Simple and complex loop kernels
  - Reductions, scans, atomic operations, multi-dim data views for changing access patterns, …
  - Loop tiling, thread-local data, GPU shared memory, …

RAJA provides building blocks that extend the generally-accepted **"parallel for" idiom**.

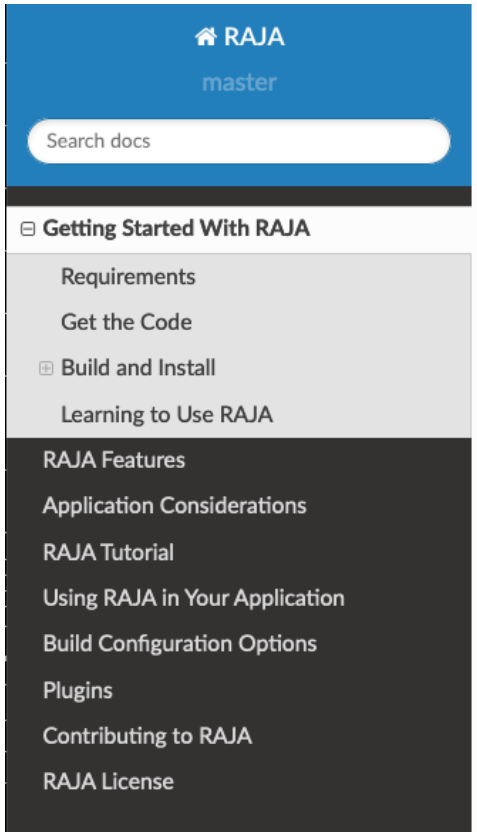
RAJA design goals target usability and developer productivity

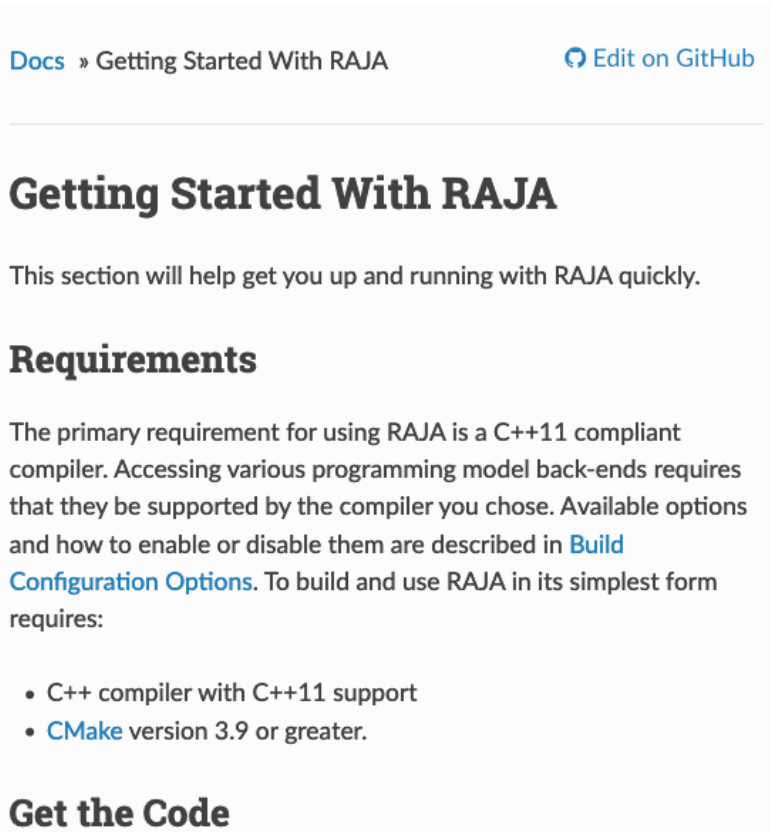
- We want applications to maintain **single-source kernels** (as much as possible)
- In addition, we want RAJA to…
  - Be **easy to understand and use** for app developers (esp. those who are not CS experts)
  - Allow **incremental and selective adoption**
  - **Not force major disruption** to application source code
  - Promote flexible algorithm implementations via **clean encapsulation**
  - Make it **easy to parameterize execution** via type aliases
  - Enable **systematic performance tuning**

These goals have been affirmed by production LLNL application teams using RAJA.

We maintain other related open source projects…

- **RAJA User Guide:** getting started info, details about features and usage, etc. (readthedocs.org/projects/raja)
- **RAJA Project Template:** shows how to use RAJA in an application that uses CMake or Make (https://github.com/LLNL/RAJA-project-template)
- **RAJA Performance Suite:** loop kernels for assessing compilers and RAJA performance. Used by us, vendors, for DOE platform procurements, etc. (https://github.com/LLNL/RAJAPerf)
- **CHAI:** array abstraction library that automatically migrates data as needed based on RAJA execution contexts (https://github.com/LLNL/CHAI)





All of these are linked on the RAJA GitHub project page.

Let's start simple…

Simple loop execution

Consider a typical C-style for-loop…

"daxpy" operation: y = a * x + y, where x, y are vectors of length N, a is a scalar

```
for (int i = 0; i < N; ++i)
{
  y[i] = a * x[i] + y[i];
}
```

Note that all aspects of execution are explicit in the source code – execution (sequential), loop iteration order, data access pattern, etc.

Converting a loop to RAJA mainly involves changing the loop header

C-style for-loop

```
for (int i = 0; i < N; ++i)
{
  y[i] = a * x[i] + y[i];
}
```

"RAJA Transformation"

RAJA-style loop

```
RAJA::forall< EXEC_POL >( it_space, [=] (int i)
{
   y[i] = a * x[i] + y[i];
} );
```

RAJA encapsulates loop execution details

C-style for-loop

```
for (int i = 0; i < N; ++i)
{
  y[i] = a * x[i] + y[i];
}
```

Typically, definitions like these go in header files.

RAJA-style loop

```
using EXEC_POL = ...;

RAJA::RangeSegment it_space(0, N);

RAJA::forall< EXEC_POL >( it_space, [=] (int i)
{
   y[i] = a * x[i] + y[i];
} );
```

By changing the “execution policy” and “iteration space”, you change the way the loop runs.

The loop header is different with RAJA, but the loop body is the same (in most cases)

C-style for-loop

```
for (int i = 0; i < N; ++i)
{
  y[i] = a * x[i] + y[i];
}
```

RAJA-style loop

```
using EXEC_POL = ...;

RAJA::RangeSegment it_space(0, N);

RAJA::forall< EXEC_POL >( it_space, [=] (int i)
{
  y[i] = a * x[i] + y[i];
} );
```

Same loop body.

RAJA loop execution has four core concepts

```
using EXEC_POLICY = ...;
RAJA::RangeSegment range(0, N);

RAJA::forall< EXEC_POLICY >( range, [=] (int i)
{
   // loop body...
} );
```

1. Loop **execution template** (e.g., ‘forall’)
2. Loop **execution policy type** (EXEC_POLICY)
3. Loop **iteration space** (e.g., ‘RangeSegment’)
4. Loop **body** (C++ lambda expression)

RAJA loop execution core concepts

```
RAJA::forall< EXEC_POLICY >( iteration_space,
   [=] (int i) {
      // loop body
   }
);
```

- RAJA::forall method runs loop based on:
  - **Execution policy type** (sequential, OpenMP, CUDA, etc.)

RAJA loop execution core concepts

```
RAJA::forall< EXEC_POLICY > ( iteration_space,
  [=] (int i) {
     // loop body
  }
);
```

- RAJA::forall template runs loop based on:
  - Execution policy type (sequential, OpenMP, CUDA, etc.)
  - **Iteration space object** (stride-1 range, list of indices, etc.)

These core concepts are common threads throughout our discussion

Simple loops

```
RAJA::forall< EXEC_POLICY > ( iteration_space,
  [=] (int i) {
     // loop body
  }
);
```

- RAJA::forall template runs loop based on:
  - Execution policy type (sequential, OpenMP, CUDA, etc.)
  - Iteration space object (contiguous range, list of indices, etc.)
- **Loop body is cast as a C++ lambda expression**
  - Lambda argument is the loop iteration variable

The programmer must ensure the loop body works with the execution policy; e.g., thread safe

The execution policy determines the programming model back-end


```
RAJA::forall< EXEC_POLICY >( range, [=] (int i)
{
   x[i] = a * x[i] + y[i];
} );
```

```
RAJA::simd_exec
```

```
RAJA::omp_parallel_for_exec
```

```
RAJA::cuda_exec<BLOCK_SIZE, Async>
```

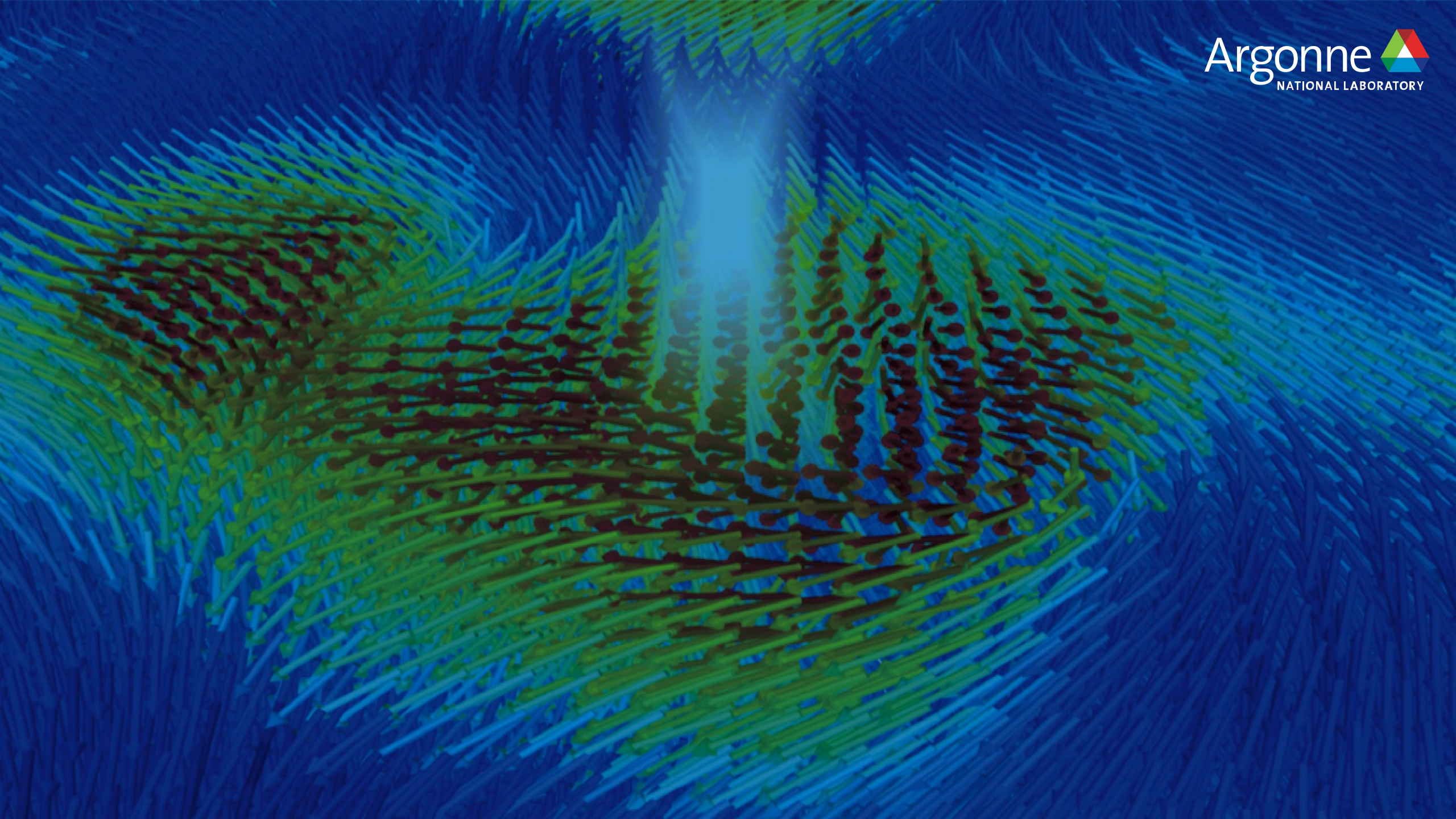
```
RAJA::omp_target_parallel_for_exec<MAX_THREADS_PER_TEAM>
```

```
RAJA::tbb_for_exec
```

A sample of RAJA loop execution policy types.

Materials that supplement this presentation are available

- Complete working example codes are available in the RAJA source repository
  - https://github.com/LLNL/RAJA
  - Many similar to examples we presented today and expands on them
  - Look in the “RAJA/examples” and “RAJA/exercises” directories
- The RAJA User Guide
  - Topics we discussed today, plus configuring & building RAJA, etc.
  - Available at http://raja.readthedocs.org/projects/raja (also linked on the RAJA GitHub project)


Argonne
NATIONAL LABORATORY