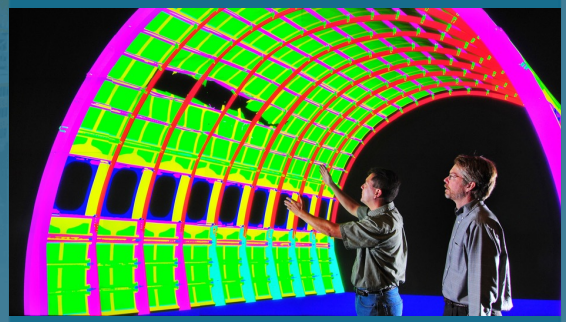


Sandia National Laboratories

kokkos

Kokkos –Performance Portability for the Exascale Era



Advanced Modeling & Simulation (AMS) Seminar Series
NASA Ames Research Center, April 4th, 2024

Christian Trott

Sandia National Laboratories, Center for Computing Research

SANDXXXX C

U.S. DEPARTMENT OF ENERGY | National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.



Kokkos & Performance Portability

Why we need Performance Portability

TOP 500 Supercomputer

10 highest ranked 11/2007



Intel Xeon 4C
3 Systems



AMD Opteron 2C
3 Systems



IBM PowerPC
4 Systems – 2C/4C

Why we need Performance Portability

TOP 500 Supercomputer

CPUs:



Threading
Building
Blocks

std::thread

Accelerators:





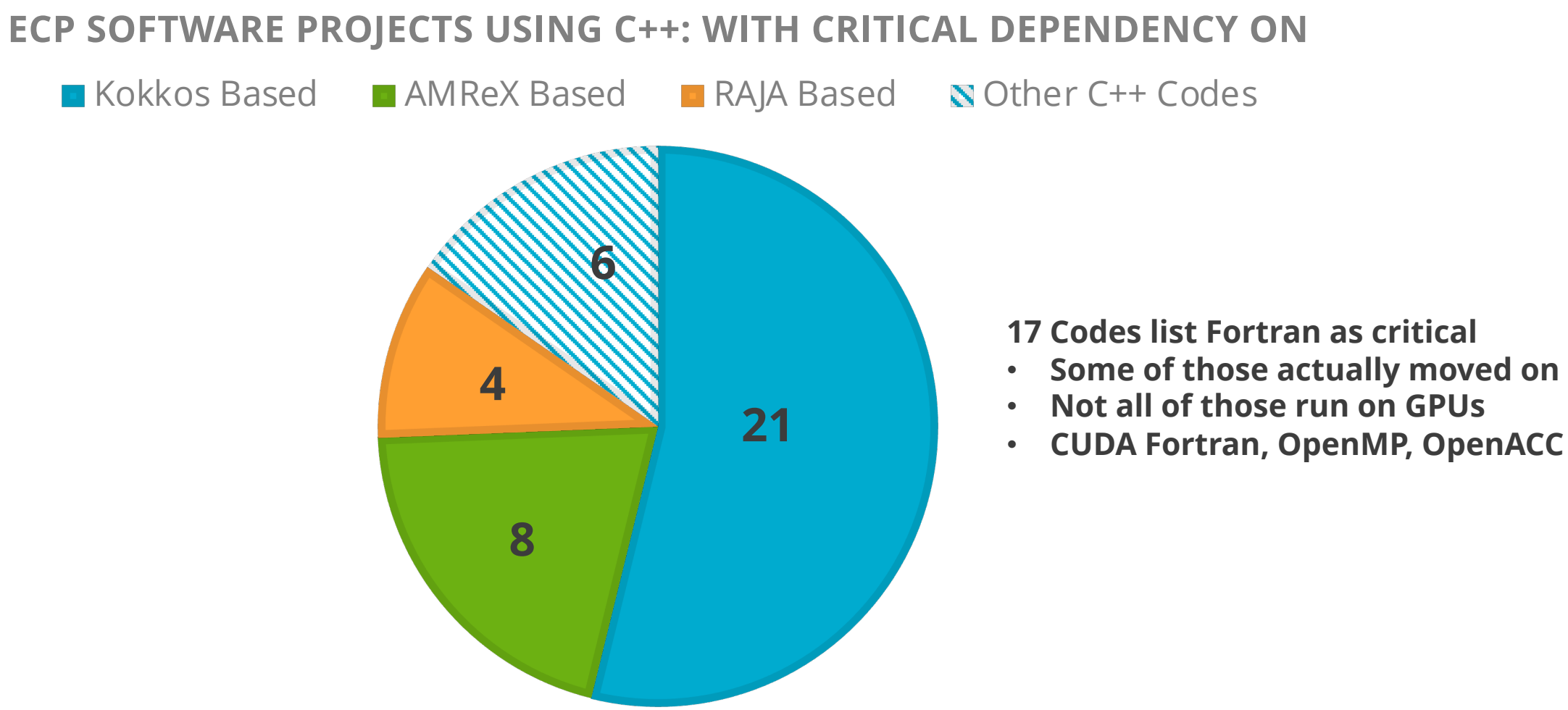






Hardware Architectures and the Means To Program Them Have Diversified

DOE Exascale Computing Project



17 Codes list Fortran as critical

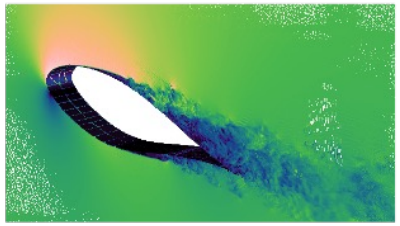
- **Some of those actually moved on**
- **Not all of those run on GPUs**
- **CUDA Fortran, OpenMP, OpenACC**

Large Majority of C++ Codes Chose Abstraction over Vendor Models!

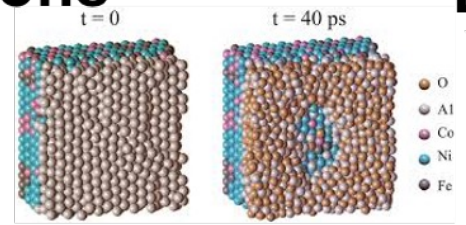
Applications

Libraries

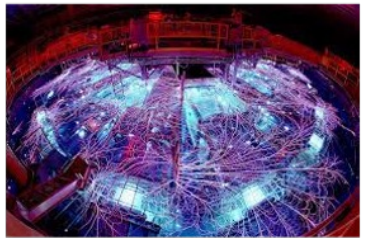
Frameworks



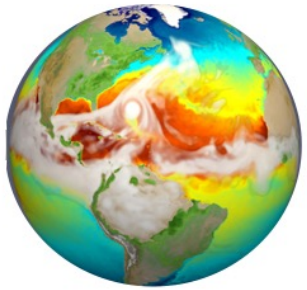
Computational Fluid Dynamics



Molecular Dynamics



Electro Magnetics

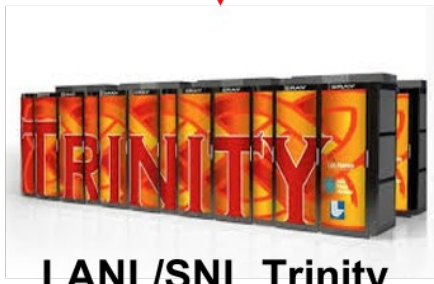


Climate Simulation

Kokkos



ORNL Frontier
AMD GPUs



LANL/SNL Trinity
Intel CPUs



ANL Aurora
Intel GPUs



Riken Fugaku
ARM CPUs



NERSC Permutter
NVIDIA GPUS

What is Kokkos?

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
- Replaces usage of CUDA, OpenMP, HIP, etc.

Expanding solution for common needs of modern science/engineering codes

- Math libraries based on Kokkos
- Tools which enable insight into Kokkos

It is Open Source

- Maintained and developed at https://github.com/kokkos

It has many users at wide range of institutions.



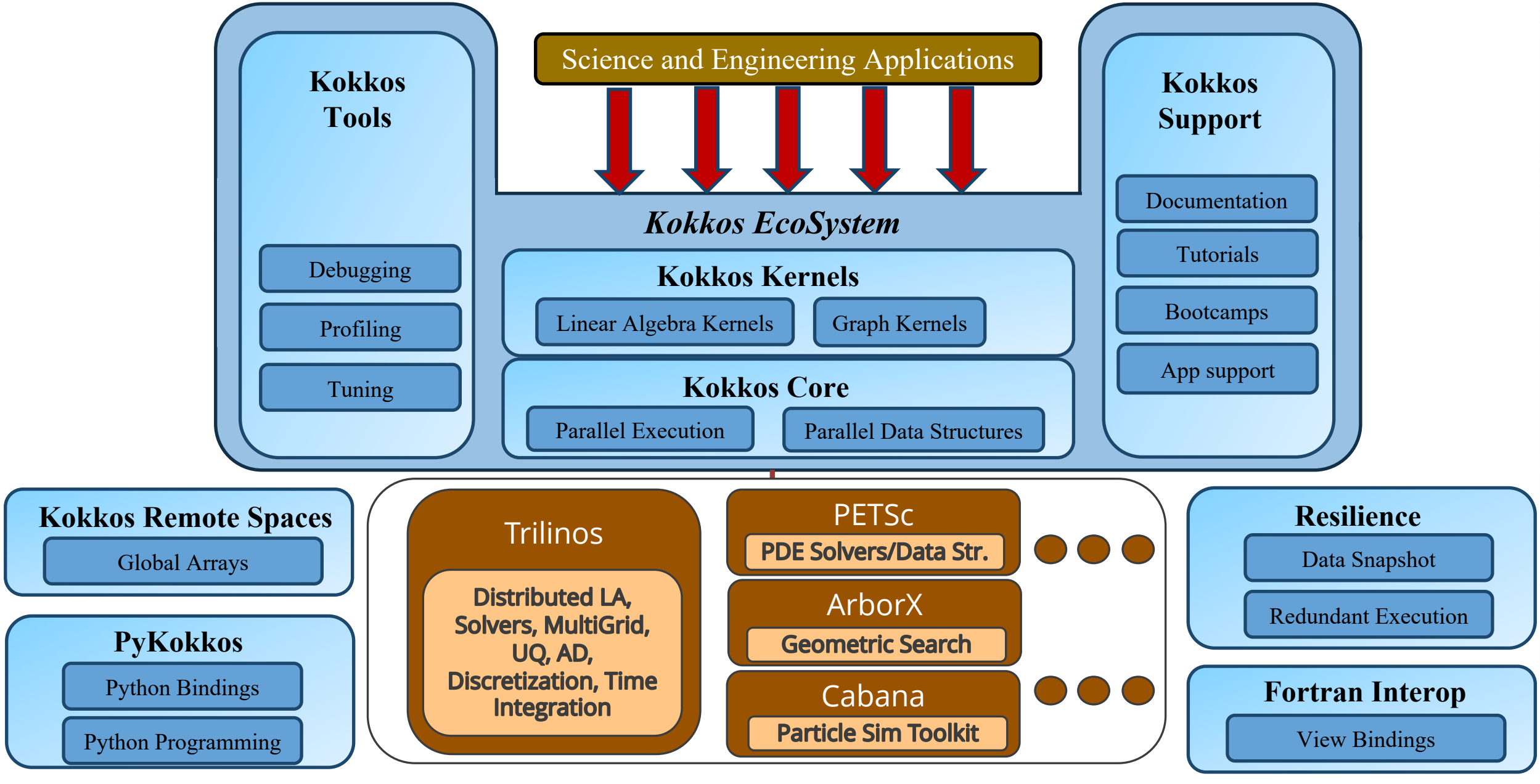
kokkos





Kokkos is NOT just for GPUs!

The Kokkos EcoSystem - Today



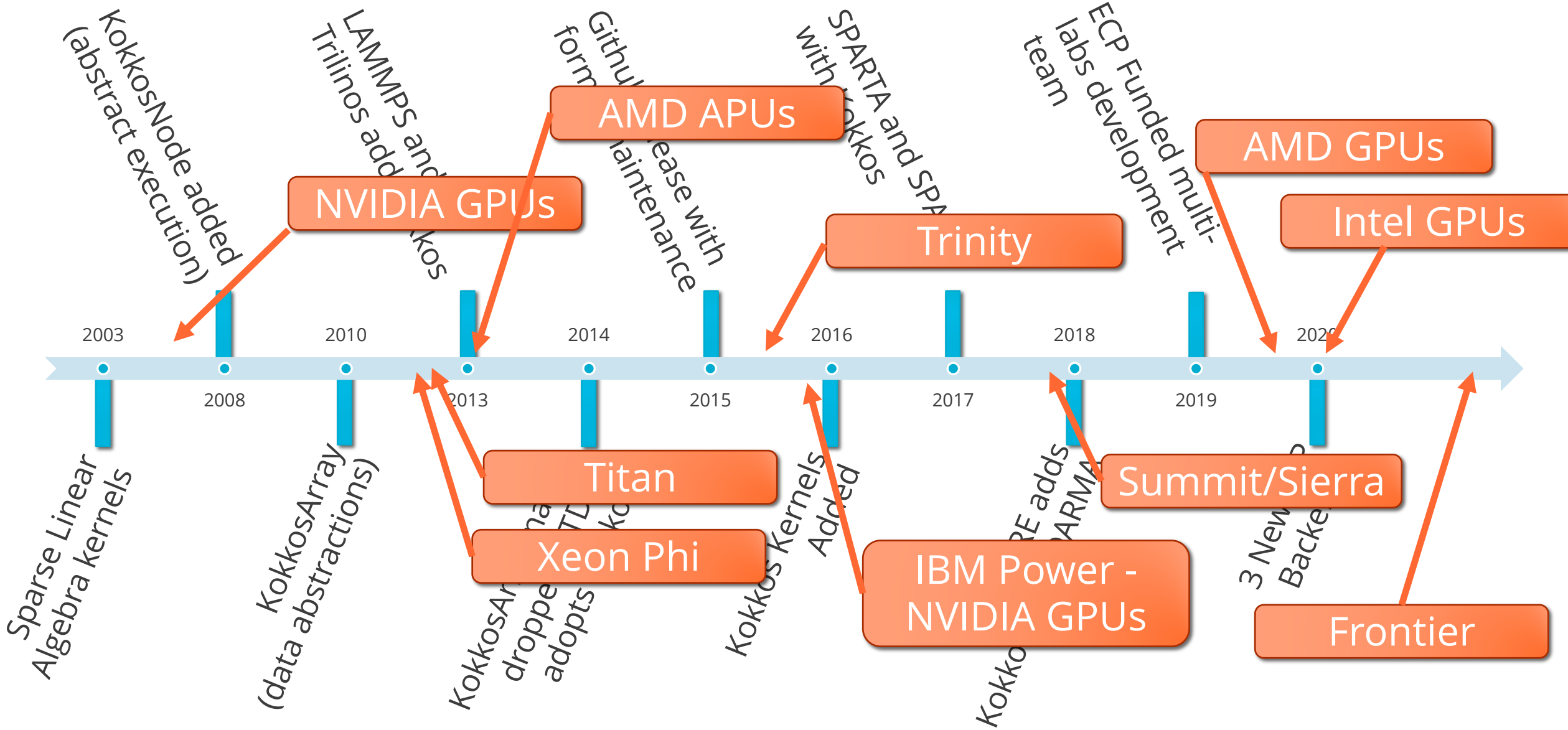
Support Matters



- DOE has service contracts with vendors to fix bug in their software.
- Most large procurement contracts contain Center Of Excellence financing
  - Helps DOE software teams to prepare for a new machine in advance of that machine arriving
  - Helps vendors to find bugs and missing features in their software stack before the machine is deployed
- Vendors work with DOE software technology teams like Kokkos, RAJA, AMReX early on
- But they don't support each others chosen models (CUDA, HIP, SYCL) or compilers

Application teams are reluctant to use non-vendor provided software stacks.

Kokkos Timeline

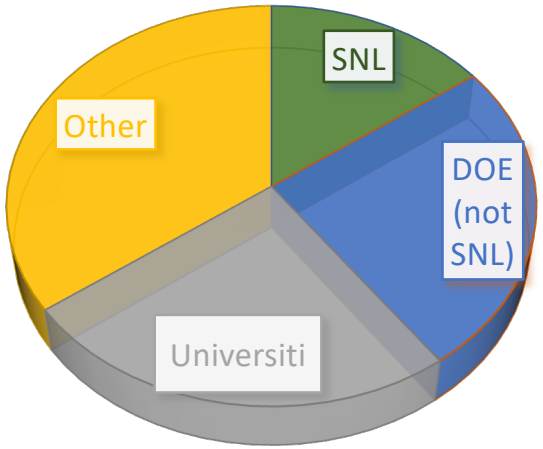


Kokkos Community

Kokkos Slack

https://kokkosteam.slack.com

- 1500 Registered Users
- >150 Institutions
  - Including 34 European



Kokkos Developers

Sandia National Laboratories
OAK RIDGE National Laboratory
cea
Argonne NATIONAL LABORATORY
Los Alamos NATIONAL LABORATORY EST.1943
BERKELEY LAB
CSCS

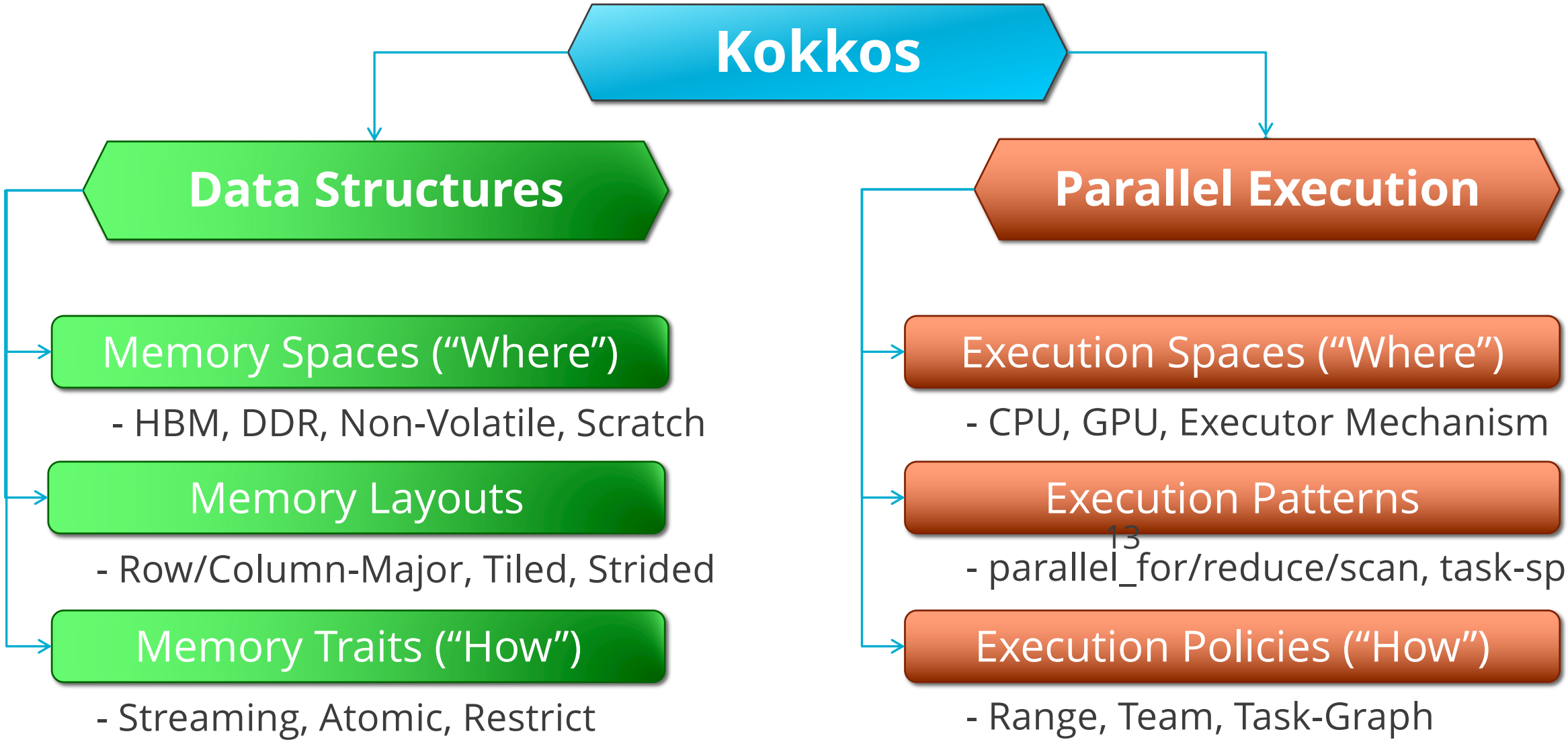
Applications and Libraries

- Estimated 150-300 HPC projects using Kokkos
- On the order of three-dozen apps run science and engineering production runs with Kokkos
  - Many apps use multiple Kokkos based libraries
- Similar distribution as the Slack User



Kokkos Programming Model

Kokkos Core Abstractions



Kokkos View

Multidimensional arrays with layouts

- Supports runtime and compile time dimensions (and mix)
- Can take subslices
- Typesafe with respect to allocator (e.g. device vs host)
- Typesafe data layouts

```
template<class DataType, class Layout, class MemorySpace, class MemoryTraits>
class View;

// Allocate NxMx4 Rank-3 Array of doubles
View<double**[4]> a("Label",N,M);

// Column Major 9x3 matrix created from existing pointer to 27 doubles
View<double[9][3],LayoutLeft> mleft(ptr_m1);
// Row Major 9x3 matrix  created from existing pointer to 27 doubles
View<double[9][3],LayoutRight> mright(ptr_m1);

// access
a(4,5,2) = 5.0;

// Take subslice
auto subslice = subview(a, pair(5,N-5), ALL, 2);
```

Example: Conjugent Gradient Solver



Simple Iterative Linear Solver

For example used in MiniFE

Uses only three math operations:

- Vector addition (AXPBY)
- Dot product (DOT)
- Sparse Matrix Vector multiply (SPMV)

Data management with Kokkos Views:

```
// Wrap existing host data (double* x_in) into Kokkos View

View<double*,HostSpace,MemoryTraits<Unmanaged> > h_x(x_in, nrows);
// Allocate ”device” data
View<double*> x("x",nrows);
// Synchronous copy from h_x to x
deep_copy(x, h_x);
// Asynchronous copy from h_x to x
deep_copy(DefaultExecutionSpace(), x, h_x);
```

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];
}
```

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);
  });
}
```

String Label: Profiling/Debugging

Execution Policy: do n iterations

Loop Body

Iteration handle: integer index

Parallel Pattern: for loop

CG Solve: The Dot Product

Simple data parallel loop with reduction: Kokkos::parallel_reduce

Non trivial in CUDA due to lack of built-in reduction support

Bandwidth bound

Serial Implementation:

```
double dot(int n, const double* x, const double* y) {
  double sum = 0.0;
  for(int i=0; i<n; i++)
    sum += x[i]*y[i];
  return sum;
}
```

Kokkos Implementation:

```
double dot(int n, View<const double*> x, View<const double*> y) {
  double x_dot_y = 0.0;
  parallel_reduce("Dot",n, KOKKOS_LAMBDA (const int i,double& sum) {
    sum += x[i]*y[i];
  }, x_dot_y);
  return x_dot_y;
}
```

Parallel Pattern: loop with reduction

Iteration Index + Thread-Local Red. Varible

CG Solve: Sparse Matrix Vector Multiply

Loop over rows

Dot product of matrix row with a vector

Example of Non-Tightly nested loops

Random access on the vector (Texture fetch on GPUs)

Outer loop over matrix rows

```
void SPMV(int nrows, const int* A_row_offsets, const int* A_cols,
          const double* A_vals, double* y, const double* x) {
  for(int row=0; row<nrows; ++row) {
    double sum = 0.0;
    int row_start=A_row_offsets[row];
    int row_end=A_row_offsets[row+1];
    for(int i=row_start; i<row_end; ++i) {
      sum += A_vals[i]*x[A_cols[i]];
    }
    y[row] = sum;
  }
}
```

Inner dot product row x vector

CG Solve: Sparse Matrix Vector Multiply

```
void SPMV(int nrows, View<const int*> A_row_offsets,
          View<const int*> A_cols, View<const double*> A_vals,
          View<double*> y,
          View<const double*, MemoryTraits< RandomAccess>> x) {

  // Performance heuristic to figure out how many rows to give to a team
  int rows_per_team = get_row_chunking(A_row_offsets);

  parallel_for("SPMV:Hierarchy", TeamPolicy< Schedule< Static > >
       ((nrows+rows_per_team-1)/rows_per_team,AUTO,8),
    KOKKOS_LAMBDA (const TeamPolicy<>::member_type& team) {

    const int first_row = team.league_rank()*rows_per_team;
    const int last_row = first_row+rows_per_team<nrows? first_row+rows_per_team : nrows;

    parallel_for(TeamThreadRange(team,first_row,last_row),[&] (const int row) {
      const int row_start=A_row_offsets[row];
      const int row_length=A_row_offsets[row+1]-row_start;

      double y_row;
      parallel_reduce(ThreadVectorRange(team,row_length),[&] (const int i, double& sum) {
        sum += A_vals(i+row_start)*x(A_cols(i+row_start));
      } , y_row);
      y(row) = y_row;
    });
  });
}
```

Enable Texture Fetch on x

Row x Vector dot product

Distribute rows in workset over team-threads

Team Parallelism over Row Worksets

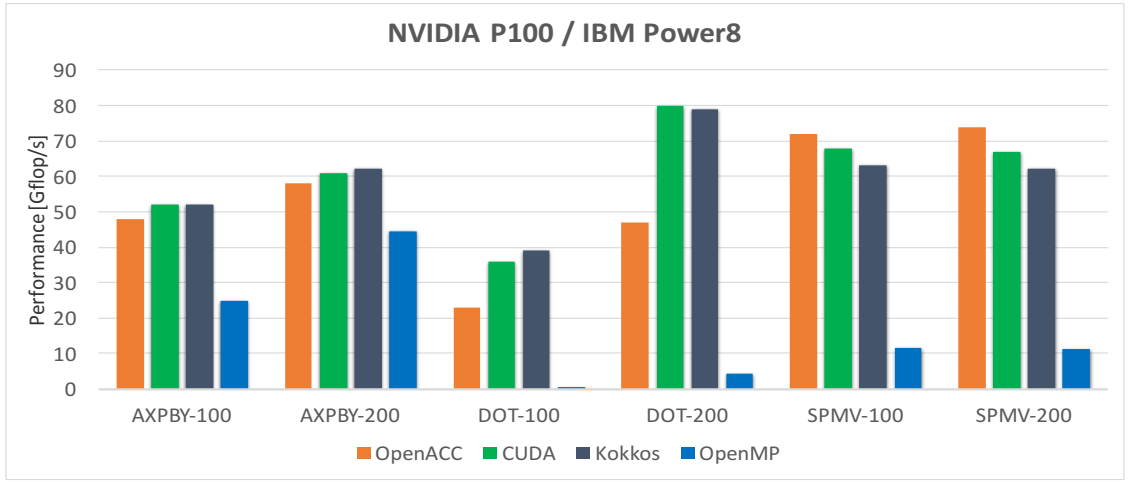
CG Solve: Performance 2016

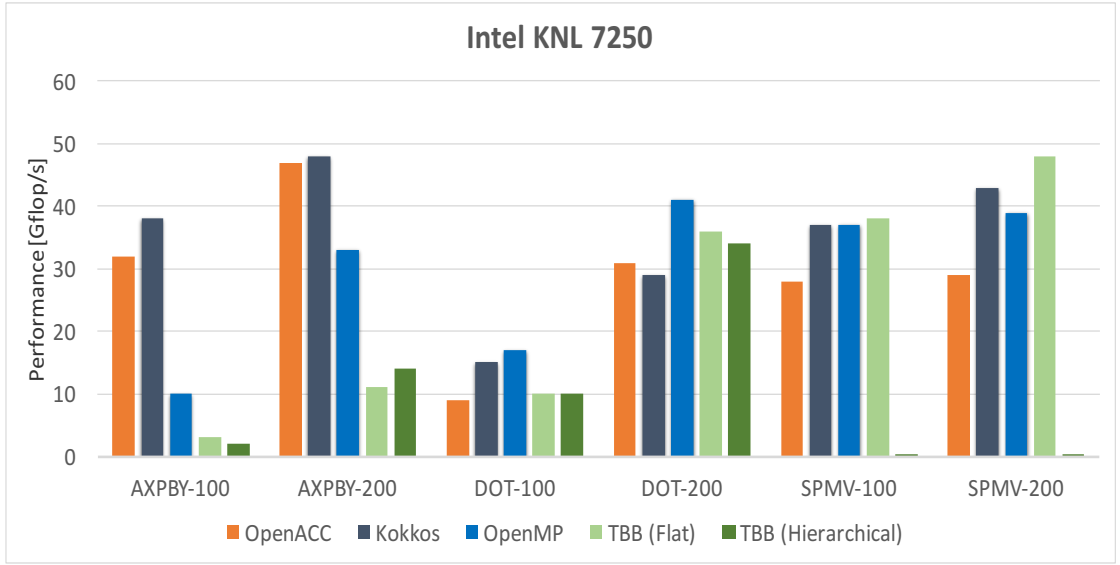
Comparison with other Programming Models

Straight forward implementation of kernels

OpenMP 4.5 is immature at this point

Two problem sizes: 100x100x100 and 200x200x200 elements





CG Solve Performance 2022

CG-Solve as discussed above

Also try replacing SPMV with TPL

Running 100x100x100 heat conduction problem

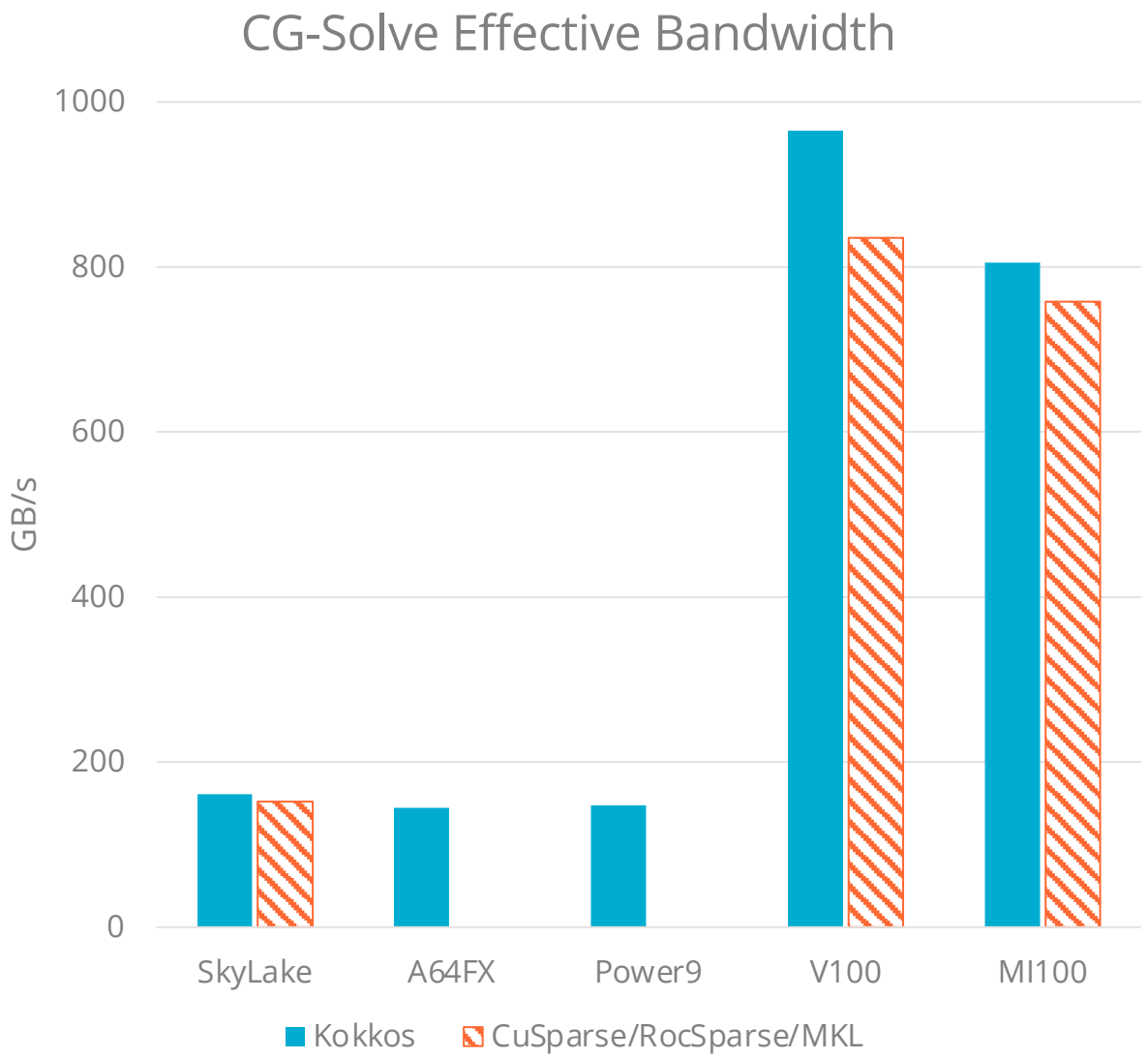
- ”MiniFE” Proxyapp setup

Measure effective Bandwidth

- Algorithmical memory ops per time

Why is this beating vendor libs?

- Its complicated, but a real effect



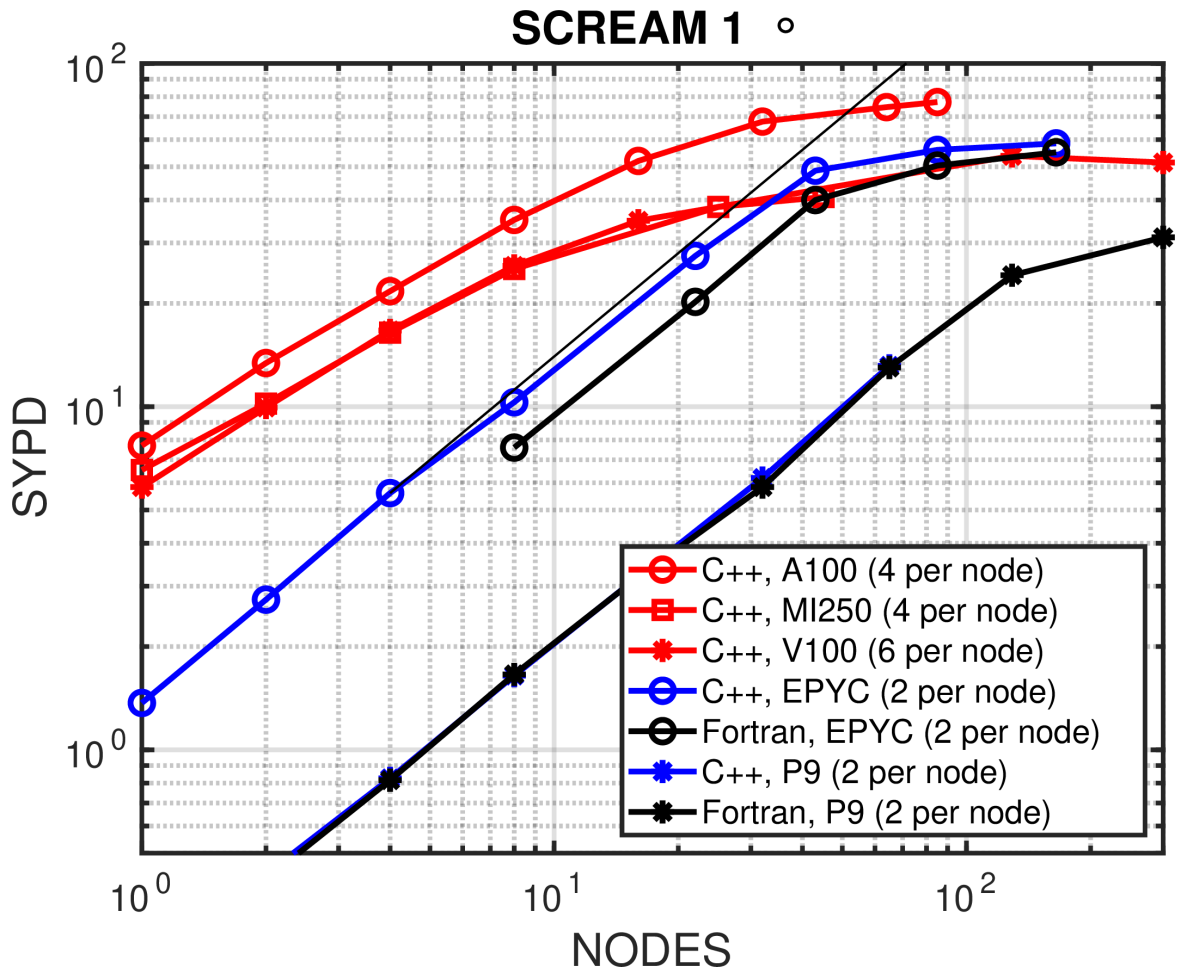
Kokkos and E3SM: Energy Exascale Earth System Model

2023 Winner of Gordon Bell Prize for Climate Modeling for SCREAM component

- Eight-lab project led by Lawrence Livermore National Lab
- Original code in Fortran, now C++
- Different components developed by different subteams
- Some use Kokkos some YAKL
- SCREAM is written in Kokkos

E3SM Atmosphere, running at 1 deg resolution: 128L, NH dycore, 10 tracers, P3/SHOC physics with prescribed aerosols, no convective parameterization

- CPU performance:
  - C++/Kokkos as fast or faster than Fortran
- GPU performance:
  - Large scaling range where GPU nodes are 4-10x faster than CPU nodes



Kokkos std::algorithms

Provide well known interfaces for C++ standard algorithms

Limited to 1D Kokkos Views for now

- Can use iterator interface, or directly passing views which is safer

>60 algorithms available

```
// C++17 standard
void foo(std::vector<double>& a, std::vector<double>& b) {
  assert(a.size() == b.size());
  std::transform(std::execution::par,
    std::begin(a), std::end(a), std::begin(b),
    [=](const double& in, double& out) { /* do something */ });
}

// Kokkos using View interface (size check inside transform)
void foo(Kokkos::View<double*> a, Kokkos::View<double*> b) {
  Kokkos::transform(Kokkos::DefaultExecutionSpace(), a, b,
    [=](const double& in, double& out) { /* do something */ });
}
```

ISO C++ SIMD Types



- Previously available as separate repository (github.com/kokkos/simd-math)
- Now part of Kokkos Core
- Implements proposed ISO C++ SIMD types (for C++26)
- Work across all backends, may just fall back to scalar operations

```
using simd_t = Kokkos::Experimental::simd<double, Kokkos::Experimental::simd_abi::native<double>>;

int N = N_in / simd_t::size();
  Kokkos::View<simd_t **> data("D", N, M);
  Kokkos::View<simd_t *> results("R", N);

  double a = 3.0;
  Kokkos::parallel_for( "Combine", data.extent(0), KOKKOS_LAMBDA(const int i) {
    simd_t tmp = 0.0;
    double b = a;
    for (int j = 0; j < data.extent(1); j++) {
      tmp += b * data(i, j);
      b += a + 1.0 * (j + 1);
    }
    results(i) = tmp;
  });
```



Programming model is not enough!

Kokkos-Tools Profiling & Debugging

Performance tuning requires insight, but tools are different on each platform

KokkosTools: Provide common set of basic tools + hooks for 3rd party tools

Common issue: abstraction layers obfuscate profiler output:

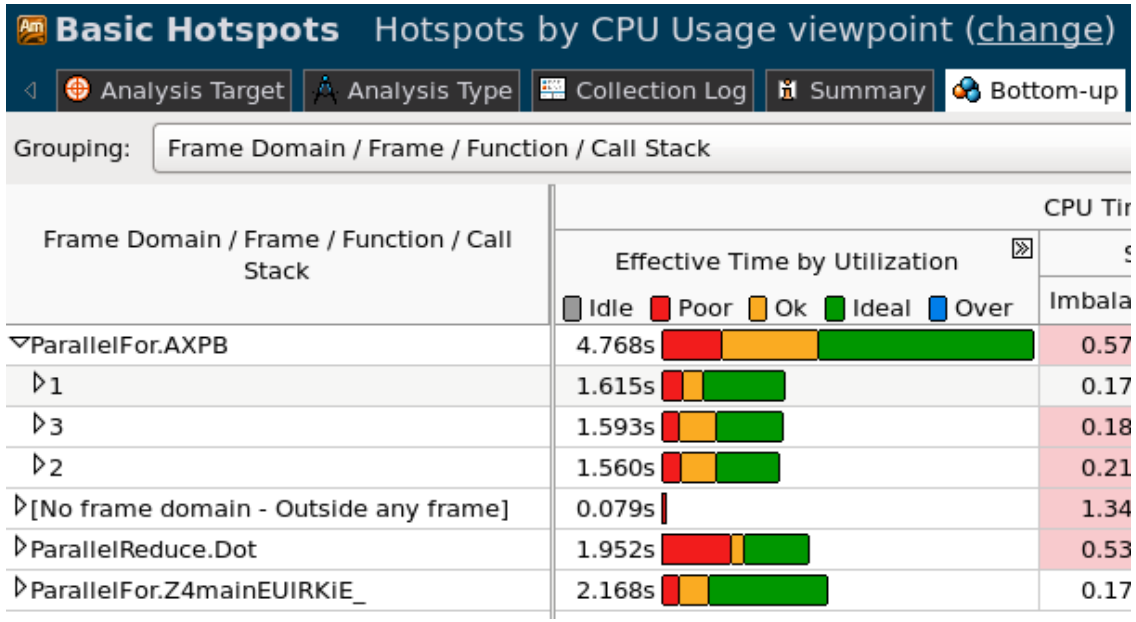
- Kokkos hooks for passing names on
- Mandator names for allocation

  View<int*> a(“Label”, N);
- Recommended names for kernels

  parallel_for(”Label”, N, callable);

No need to recompile

- Uses runtime hooks
- Set via env variable or API calls



Kokkos Support

The Kokkos Lectures

- 8 lectures covering most aspects of Kokkos
- 15 hours of recordings
- > 500 slides
- >20 exercises

Extensive Wiki

- API Reference
- Programming Guide

Slack as primary direct support

https://kokkos.link/the-lectures

- Module 1: Introduction
  - Introduction, Basic Parallelism, Build System
- Module 2: Views and Spaces
  - Execution and Memory Spaces, Data Layout
- Module 3: Data Structures and MDRangePolicy
  - Tightly Nested Loops, Subviews, ScatterView,…
- Module 4: Hierarchical Parallelism
  - Nested Parallelism, Scratch Pads, Unique Token
- Module 5: Advanced Optimizations
  - Streams, Tasking and SIMD
- Module 6: Language Interoperability
  - Fortran, Python, MPI and PGAS
- Module 7: Tools
  - Profiling, Tuning , Debugging, Static Analysis
- Module 8: Kokkos Kernels
  - Dense LA, Sparse LA, Solvers, Graph Kernels

Kokkos Kernels



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);
```

vs.

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

Reminder Kokkos code calling NVIDIA CUBLAS

```
template <class... Args1, class... Args2, class... Args3>
void cublas_gemv(Kokkos::View<double**, Args1...> A,
                 Kokkos::View<double*, Args2...> x,
                 Kokkos::View<double*, Args3...> y) {
  double alpha = 1.0;
  double beta  = 0.0;
  bool A_is_lr =
      std::is_same<Kokkos::LayoutRight, decltype(A)::array_layout>::value;
  const int M       = static_cast<int>(A_is_lr ? A.extent(1) : A.extent(0));
  const int N       = static_cast<int>(A_is_lr ? A.extent(0) : A.extent(1));
  constexpr int one = 1;
  const int LDA     = A_is_lr ? A.stride(0) : A.stride(1);

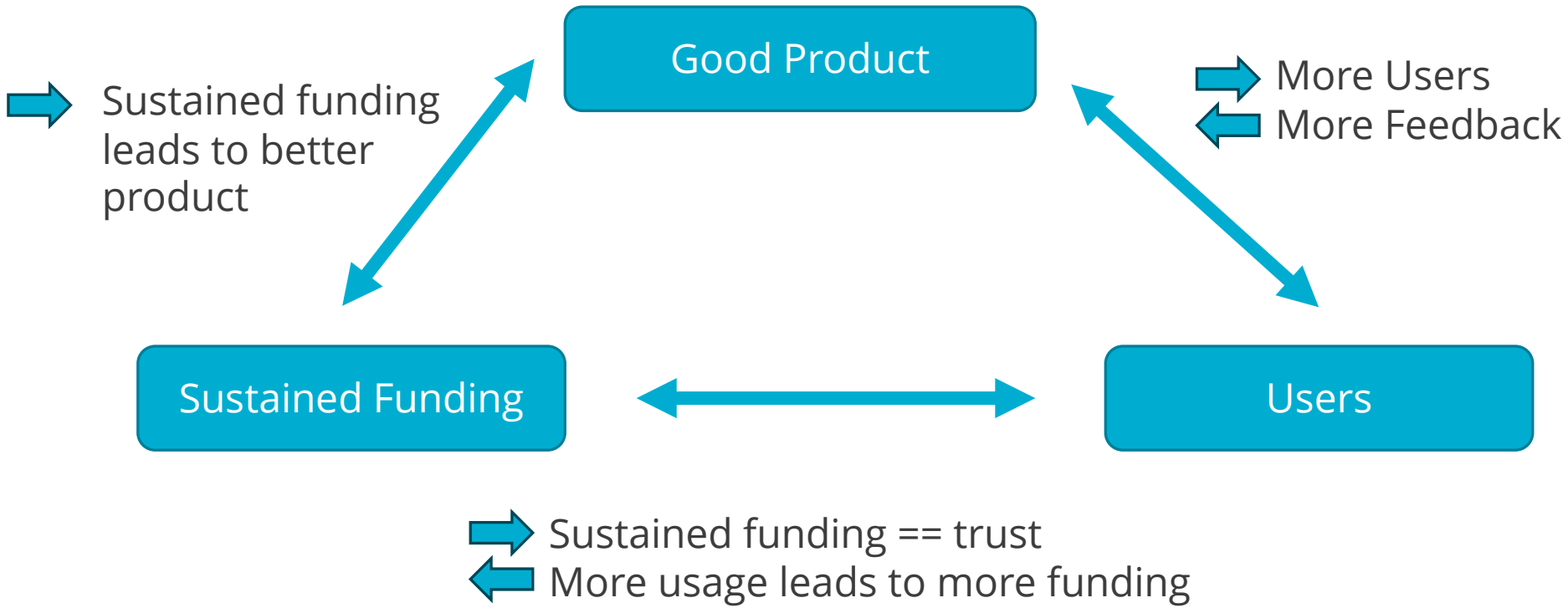

  cublasOperation_t transa;
  transa = A_is_lr ? CUBLAS_OP_T : CUBLAS_OP_N;
  cublasStatus_t handle;
  cublasCreate(&handle)
  cublasDgemv(handle, transa, M, N, &alpha,
              A.data(), LDA,
              x.data(), one, &beta,
              y.data(), one);
}
```

With KokkosKernels this can be replaced by this call:

```
KokkosBlas::gemm('N','N', 1.0, A, x, 0.0, y);
```

Sustainability

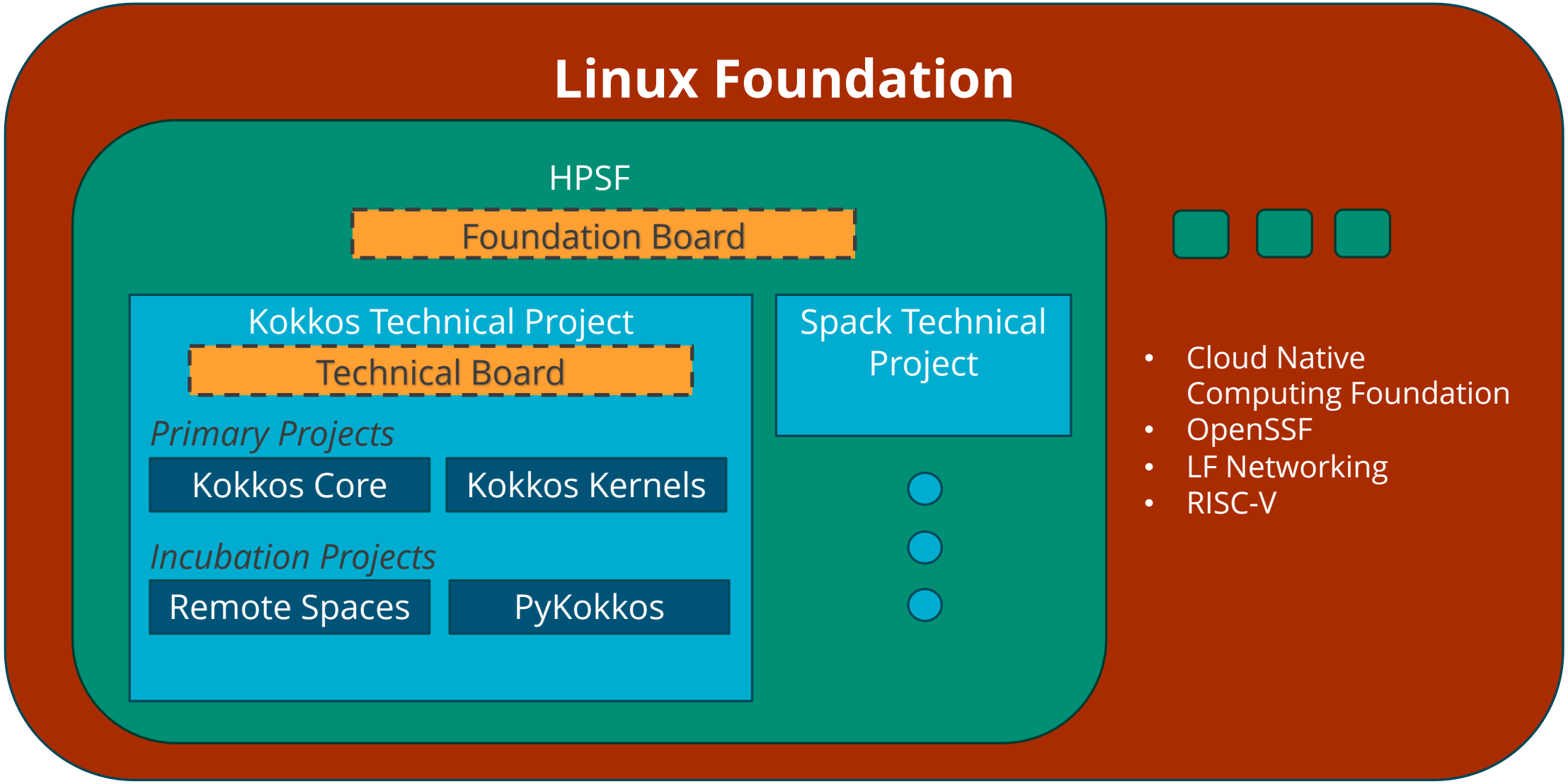
Sustainment: A self reinforcing Cycle?



There is strength in numbers: collaboration on core product good for everyone!

High Performance Software Foundation

Primary Goal: *Enable true partnerships on Kokkos via open governance.*



A few more foundation details

Primary Organizers: Todd Gamblin (Spack) and Christian Trott (Kokkos)

Sponsors:

- *US Labs:* SNL, LLNL, ORNL, LBL, LANL and ANL
- *Tentative Industry:* HPE, Amazon, NVIDIA, AMD and Intel

Kokkos paperwork for joining Linux Foundation already signed by DOE.

HPSF will launch at ISC in Hamburg in May.

What does it do:

- Guarantees Open Governances (e.g. github.com/kokkos will be owned by LF)
- Help organize and pay for meetings and workshops
- Help finance and organize project infrastructure (e.g. Slack)

Takeaways



Kokkos enables performance portability for complex science and engineering applications today.

Kokkos is supported by a large developer team from 6 institutions.

Kokkos has a user-base with thousands of developers from more than 150 institutions.

Kokkos is moving to the Linux Foundation to enable wider adoption and foster a growing and diverse developer community.


kokkos