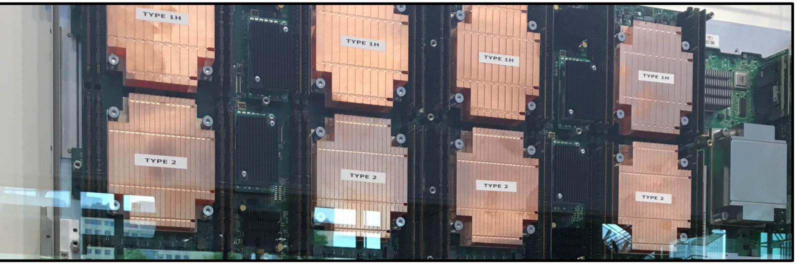
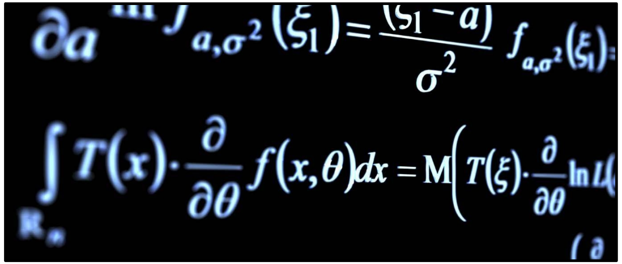
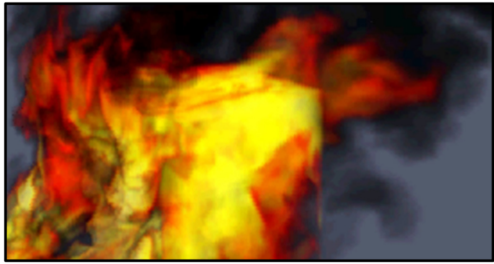
Exceptional service in the national interest







The Kokkos C++ Performance Portability EcoSystem

Unclassified Unlimited Release

Christian R. Trott, - Center for Computing Research

Sandia National Laboratories/NM

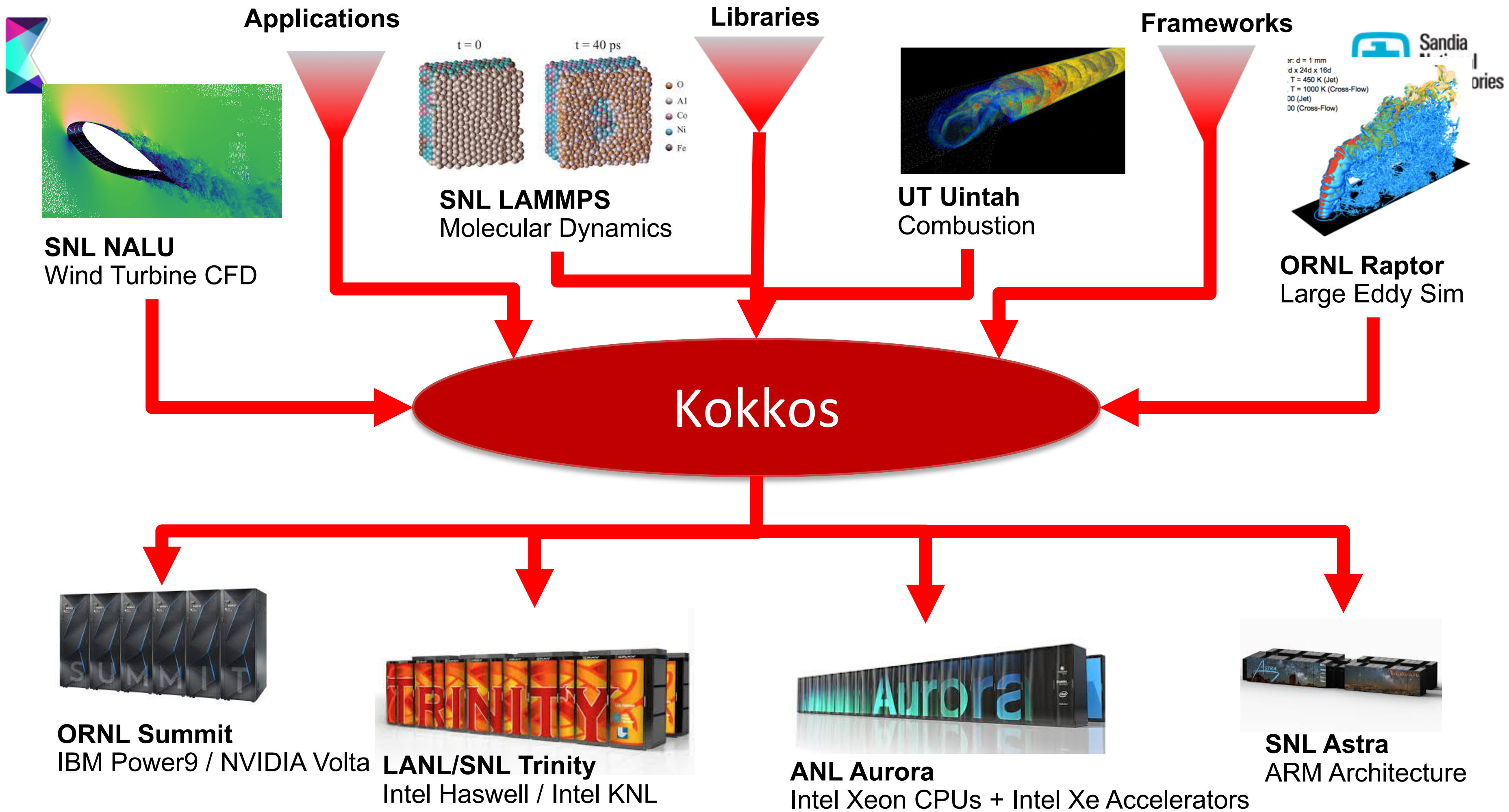
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

SAND2019-11367 PE

Cost Of Software

10 LOC / hour ~ 20k LOC / year

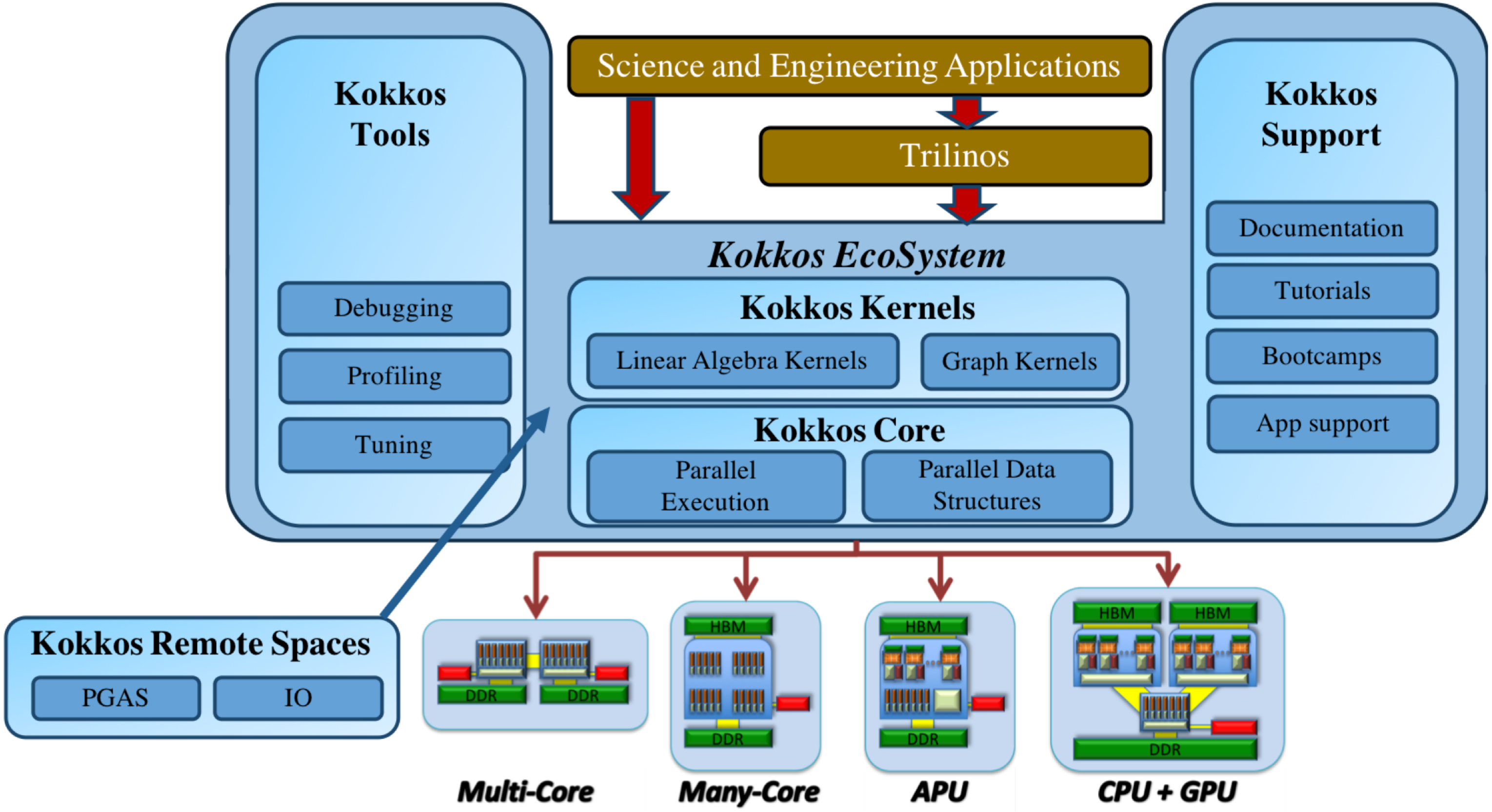
- Optimistic estimate: 10% of an application needs to get rewritten for adoption of Shared Memory Parallel Programming Model
- Typical Apps: 300k – 600k Lines
  - Uintah: 500k, QMCPack: 400k, LAMMPS: 600k; QuantumEspresso: 400k
  - Typical App Port thus 2-3 Man-Years
  - Sandia maintains a couple dozen of those
- Large Scientific Libraries
  - E3SM: 1,000k Lines x 10% => 5 Man-Years
  - Trilinos: 4,000k Lines x 10% => 20 Man-Years


Applications
Libraries
Frameworks
t = 0
t = 40 ps
O
Al
Co
Ni
Fe
SNL NALU
Wind Turbine CFD
SNL LAMMPS
Molecular Dynamics
UT Uintah
Combustion
ORNL Raptor
Large Eddy Sim
Kokkos
ORNL Summit
IBM Power9 / NVIDIA Volta
LANL/SNL Trinity
Intel Haswell / Intel KNL
ANL Aurora
Intel Xeon CPUs + Intel Xe Accelerators
SNL Astra
ARM Architecture

What is Kokkos?

- A C++ Programming Model for Performance Portability
  - Implemented as a template library on top of CUDA, OpenMP, ROCm, …
  - 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 Development Team

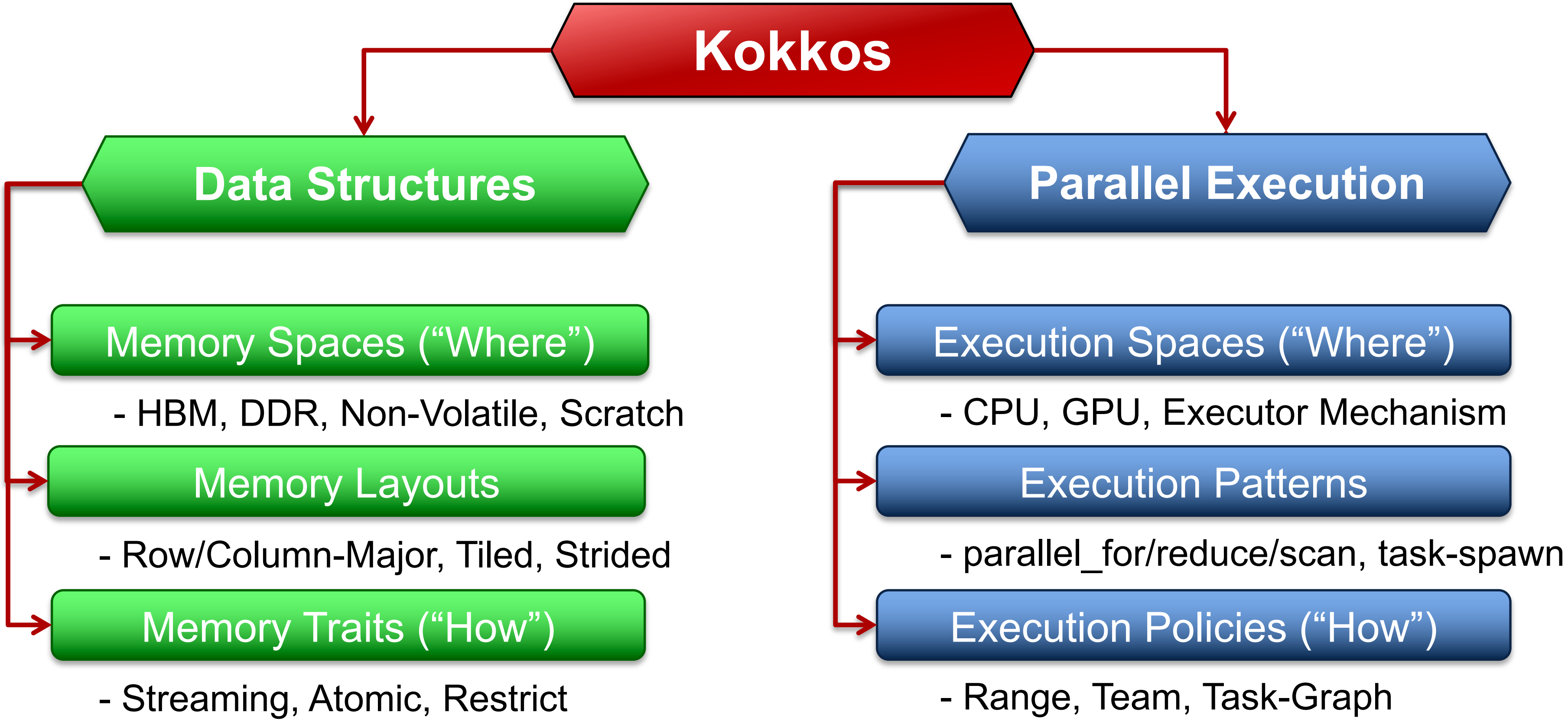
Kokkos Core: ***C.R. Trott,*** *D. Sunderland, N. Ellingwood, D. Ibanez, J. Miles, D. Hollman, V. Dang, H. Finkel, N. Liber, D. Lebrun-Grandie, B. Turcksin, J. Wilke, D. Arndt*
former: ***H.C. Edwards****, D. Labreche, G. Mackey, S. Bova*

Kokkos Kernels: ***S. Rajamanickam,*** *N. Ellingwood, K. Kim, C.R. Trott, V. Dang, L. Berger, J. Wilke, W. McLendon*

Kokkos Tools: ***D. Poliakoff,*** *S. Hammond, C.R. Trott, D. Ibanez, S. Moore*

Kokkos Support: ***C.R. Trott,*** *G. Shipman, G. Lopez, G. Womeldorff, and all of the above as needed*
former: ***H.C. Edwards****, D. Labreche, Fernanda Foertter*

Kokkos Core Abstractions



Kokkos Core Capabilities

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), ROCm (experimental)

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
- View have all their 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);
```

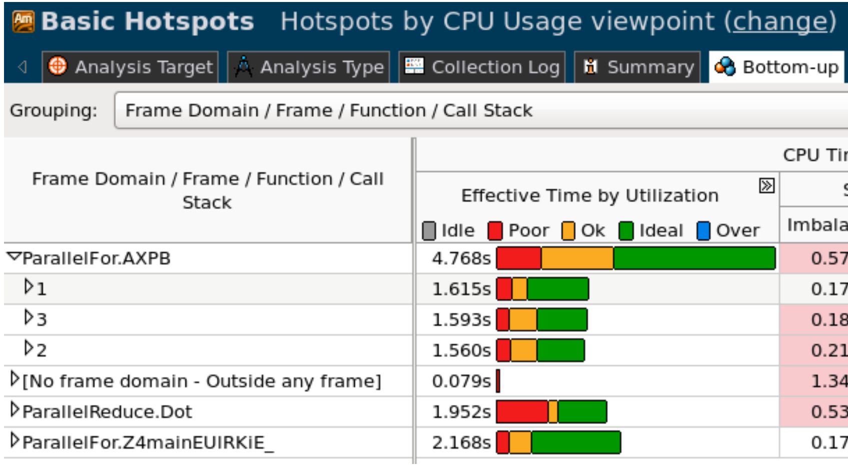
```
// Kokkos Kernels
double alpha, beta; View<double**> A,B,C;
gemm('N','N',alpha,A,B,beta,C);
```

- 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 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
- One common issue abstraction layers obfuscate profiler output
  - Kokkos hooks for passing names on
  - Provide Kernel, Allocation and Region
- No need to recompile
  - Uses runtime hooks
  - Set via env variable



DOE Machine Announcements

- Now publicly announced that DOE is buying both AMD and Intel GPUs
  - Argonne: Cray with Intel Xeon + Intel Xe Compute
  - ORNL: Cray with AMD CPUs + AMD GPUs
  - NERSC: Cray with AMD CPUs + NVIDIA GPUs
- Have been planning for this eventuality:
  - Kokkos ECP project extended and refocused to include developers at Argonne, Oak Ridge, and Lawrence Berkeley - staffing is in place
  - HIP backend for AMD: main development at ORNL
    - The current ROCm backend is based on a compiler which is now deprecated …
  - SYCL for Intel: main development at ANL
  - OpenMPTarget for AMD, Intel and NVIDIA, lead at Sandia

Supporting Aurora

- Two backend plans
  - SYCL: will need Intel proposed extensions
    - ANL will lead development
  - OpenMPTarget: OpenMP 5.x based
    - NERSC/SNL will lead development
- Timeline:
  - Q2 FY20: Initial capabilities, enough for many miniApps
  - Q4 FY20: Functional backends
  - FY21: Production support

OpenMPTarget Backend

- Started work on this more than 2 years ago
  - Hindered by compiler bugs: 15 min work on backend, 6 hours work on compiler bug reproducer, 6 months wait for fix, repeat
  - With Clang 9 first time this isn't the case
- Got some capabilities:
  - RangePolicy: parallel_for, parallel_reduce
  - MDRangePolicy: parallel_for
  - Views

SYCL Backend

- Started recently both with Codeplays and Intels compiler
- Not much working yet
  - RangePolicy: parallel_for works with Codeplay
- Looking into some of the problems around restrictions of SYCL such as kernel naming
- We likely need to rely on Intel proposed extensions
  - A good chunk of which are already implemented!

Kokkos Based Projects

- Production Code Running Real Analysis Today
  - We got about **12** or so.
- Production Code or Library committed to using Kokkos and actively porting
  - Somewhere around **35**
- Packages In Large Collections (e.g. Tpetra, MueLu in Trilinos) committed to using Kokkos and actively porting
  - Somewhere around **65**
- Counting also proxy-apps and projects which are evaluating Kokkos (e.g. projects who attended boot camps and trainings).
  - Estimate **100-150** packages.

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

GT

TUM TECHNISCHE UNIVERSITÄT MÜNCHEN

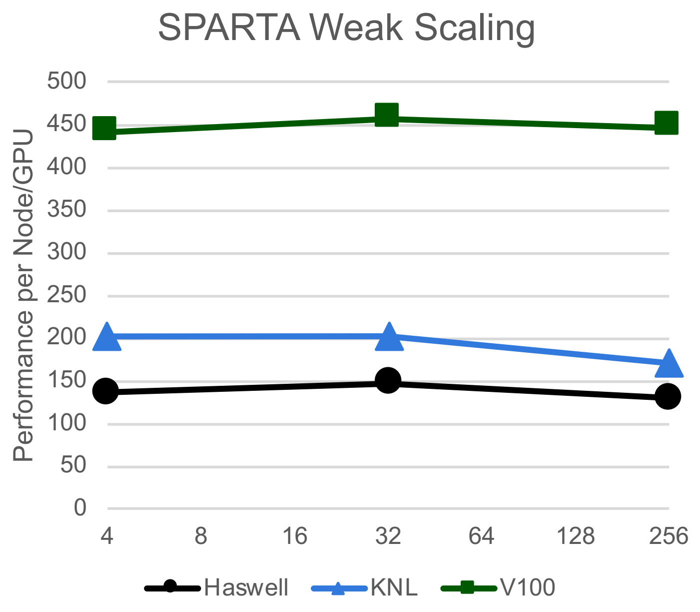
IPP Max-Planck-Institut für Plasmaphysik

BERKELEY LAB

Rensselaer

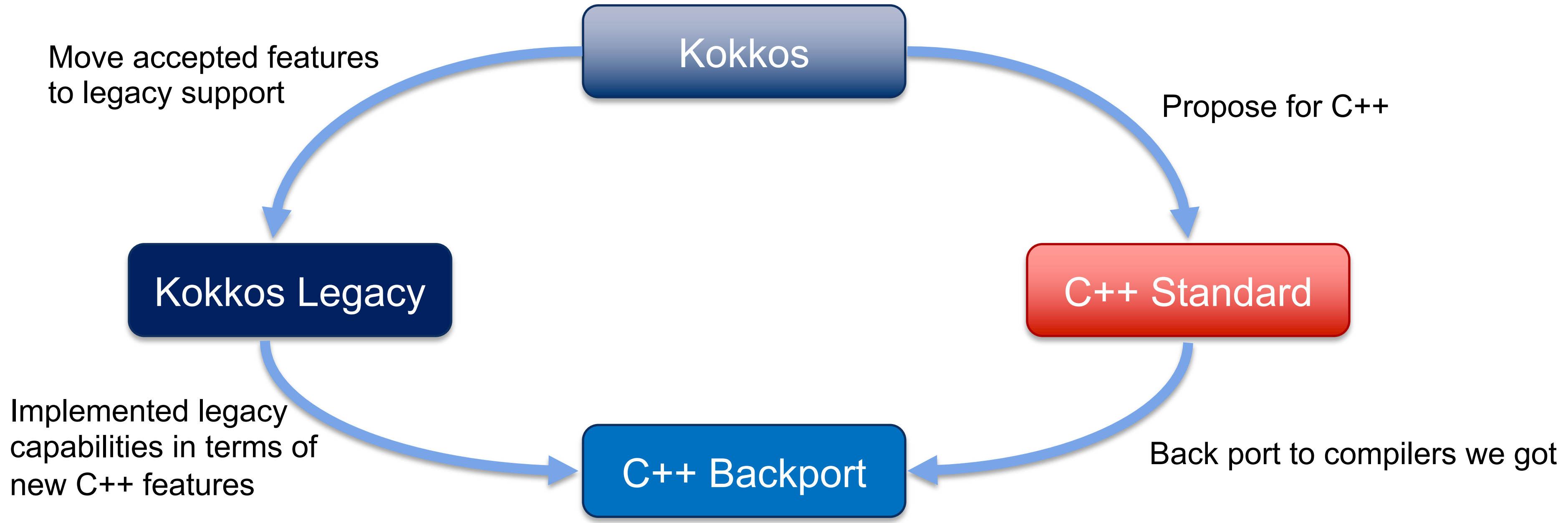
Sparta: Production Simulation at Scale

- **S**tochastic **PA**rallel **R**arefied-gas **T**ime-accurate **A**nalyzer
- A direct simulation Monte Carlo code
- Developers: *Steve Plimpton, Stan Moore, Michael Gallis*
- Only code to have run on all of Trinity
  - 3 Trillion particle simulation using both HSW and KNL partition in a single MPI run (~20k nodes, ~1M cores)
- Benchmarked on 16k GPUs on Sierra
  - Production runs now at 5k GPUs
- Co-Designed Kokkos::ScatterView



Aligning Kokkos with the C++ Standard

- Long term goal: move capabilities from Kokkos into the ISO standard
  - Concentrate on facilities we really need to optimize with compiler



C++ Features in the Works

- First success: **atomic_ref**<T> in C++20
  - Provides atomics with all capabilities of atomics in Kokkos
  - **atomic_ref**(a[i])+=5.0; instead of **atomic_add**(&a[i],5.0);
- Next thing: **Kokkos::View** => **std::mdspan**
  - Provides customization points which allow all things we can do with **Kokkos::View**
  - Better design of internals though! => Easier to write custom layouts.
  - Also: arbitrary rank (until compiler crashes) and mixed compile/runtime ranks
  - We hope will land early in the cycle for C++23 (i.e. early in 2020)
  - Production reference implementation: https://github.com/kokkos/mdspan
- Also C++23: Executors and **Basic Linear Algebra** (just began design work)

Links

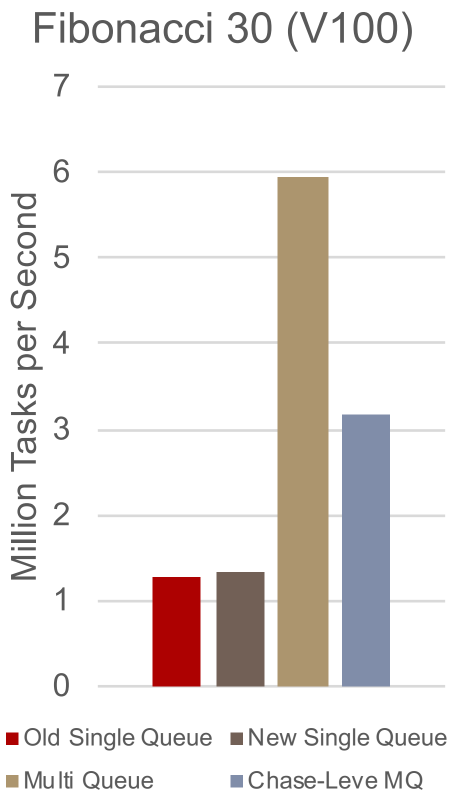
- https://github.com/kokkos Kokkos Github Organization
  - **Kokkos:** *Core library, Containers, Algorithms*
  - **Kokkos-Kernels:** *Sparse and Dense BLAS, Graph, Tensor (under development)*
  - **Kokkos-Tools:** *Profiling and Debugging*
  - **Kokkos-MiniApps:** *MiniApp repository and links*
  - **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

Sandia
National
Laboratories

Improved Fine Grained Tasking

Sandia National Laboratories

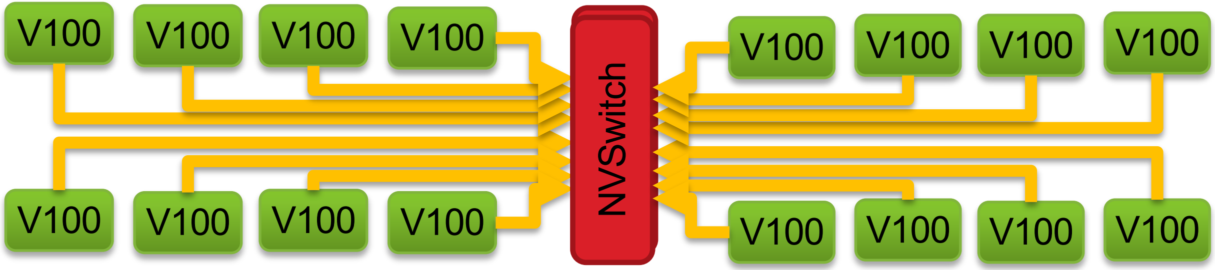
- Generalization of `TaskScheduler` abstraction to allow user to be generic with respect to scheduling strategy and queue
- Implementation of new queues and scheduling strategies:
  - Single shared LIFO Queue (this was the old implementation)
  - Multiple shared LIFO Queues with LIFO work stealing
  - Chase-Lev minimal contention LIFO with tail (FIFO) stealing
  - Potentially more
- Reorganization of Task, Future, TaskQueue data structures to accommodate flexible requirements from the TaskScheduler
  - For instance, some scheduling strategies require additional storage in the Task



Questions: David Hollman

Kokkos Remote Spaces: PGAS Support

- PGAS Models may become more viable for HPC with both changes in network architectures and the emergence of “super-node” architectures
  - Example DGX2
  - First “super-node”
  - 300GB/s per GPU link



- Idea: Add new memory spaces which return data handles with shmem semantics to Kokkos View
  - `View<double**[3], LayoutLeft, NVShmemSpace> a(“A”,N,M);`
  - Operator `a(i,j,k)` returns:

```
template<>
struct NVShmemElement<double> {
   NVShmemElement(int pe_, double* ptr_):pe(pe_),ptr(ptr_) {}
   int pe; double* ptr;
   void operator = (double val) { shmem_double_p(ptr,val,pe); }
};
```

PGAS Performance Evaluation: miniFE

- Test Problem: CG-Solve
  - Using the miniFE problem N^3
  - Compare to optimized CUDA
  - MPI version is using overlapping
  - DGX2 4 GPU workstation
  - Dominated by SpMV (Sparse Matrix Vector Multiply)
  - Make Vector distributed, and store global indicies in Matrix
- 3 Variants
  - Full use of SHMEM
  - Inline functions by ptr mapping
    - Store 16 pointers in the View
  - Explicit by-rank indexing
    - Make vector 2D
    - Encode rank in column index