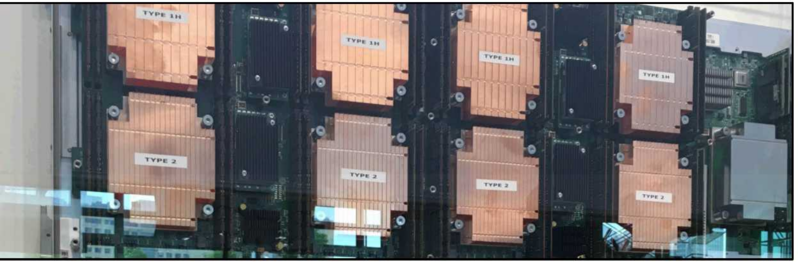
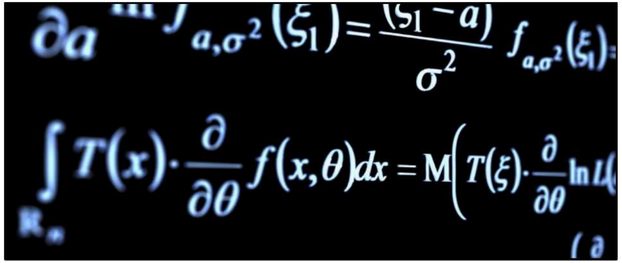
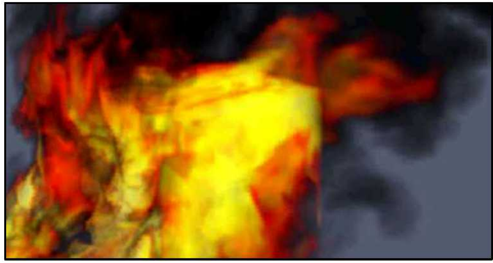
This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2019-2045C







Kokkos Libraries and Applications

SAND2018-XXXX C

Unclassified Unlimited Release

Christian R. Trott, - Center for Computing Research

Sandia National Laboratories/NM

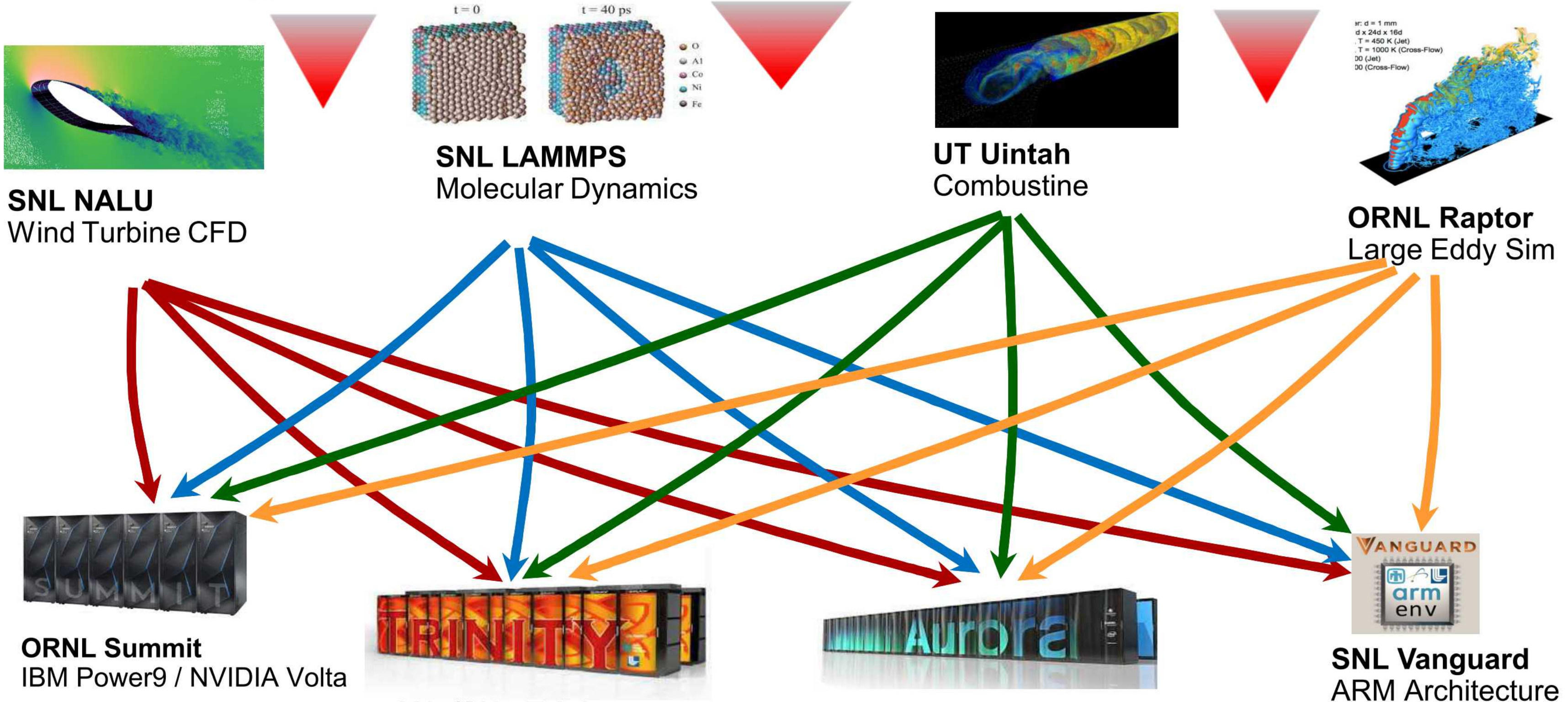
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

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.

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

SNL NALU
Wind Turbine CFD

SNL LAMMPS
Molecular Dynamics

UT Uintah
Combustine

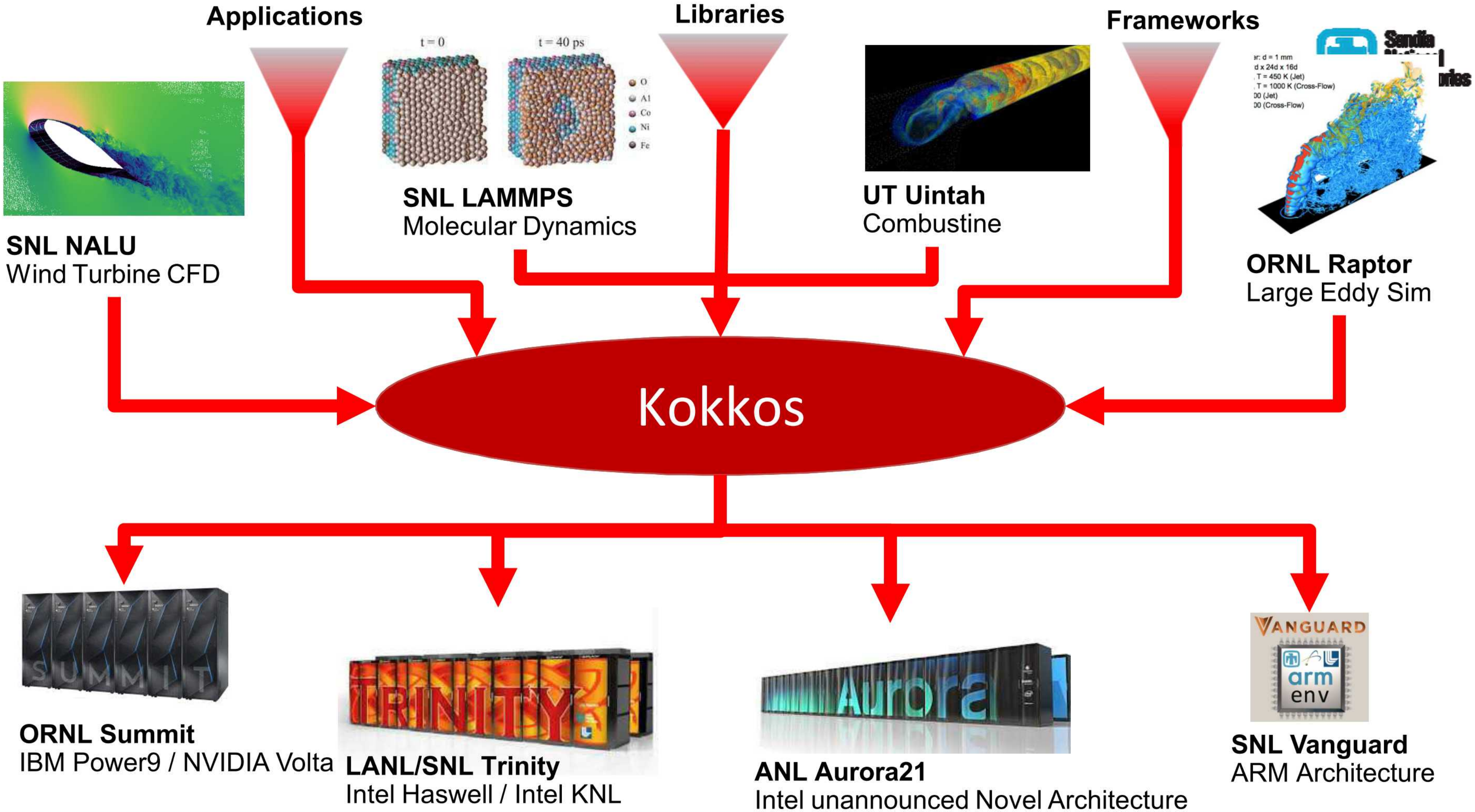
ORNL Raptor
Large Eddy Sim

ORNL Summit
IBM Power9 / NVIDIA Volta

LANL/SNL Trinity
Intel Haswell / Intel KNL

ANL Aurora21
Intel unannounced Novel Architecture

SNL Vanguard
ARM Architecture



Applications

Libraries

Frameworks

SNL NALU
Wind Turbine CFD

SNL LAMMPS
Molecular Dynamics

UT Uintah
Combustine

ORNL Raptor
Large Eddy Sim

Kokkos

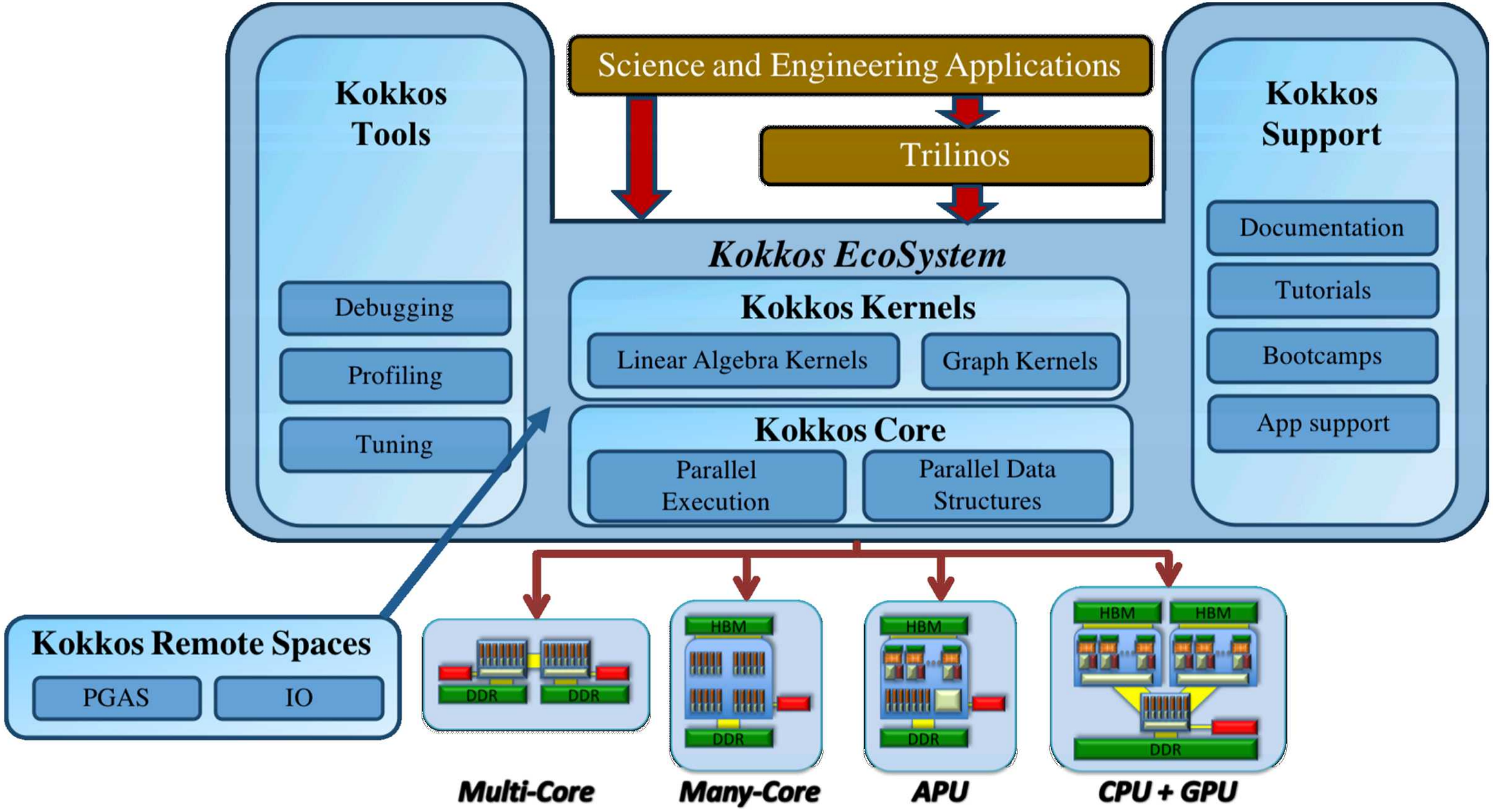
ORNL Summit
IBM Power9 / NVIDIA Volta

LANL/SNL Trinity
Intel Haswell / Intel KNL

ANL Aurora21
Intel unannounced Novel Architecture

SNL Vanguard
ARM Architecture

Kokkos EcoSystem



A Vision of the future

4 Memory Spaces

- Bulk non-volatile (Flash?)
- Standard DDR (DDR4)
- Fast memory (HBM/HMC)
- (Segmented) scratch-pad on die

3 Execution Spaces

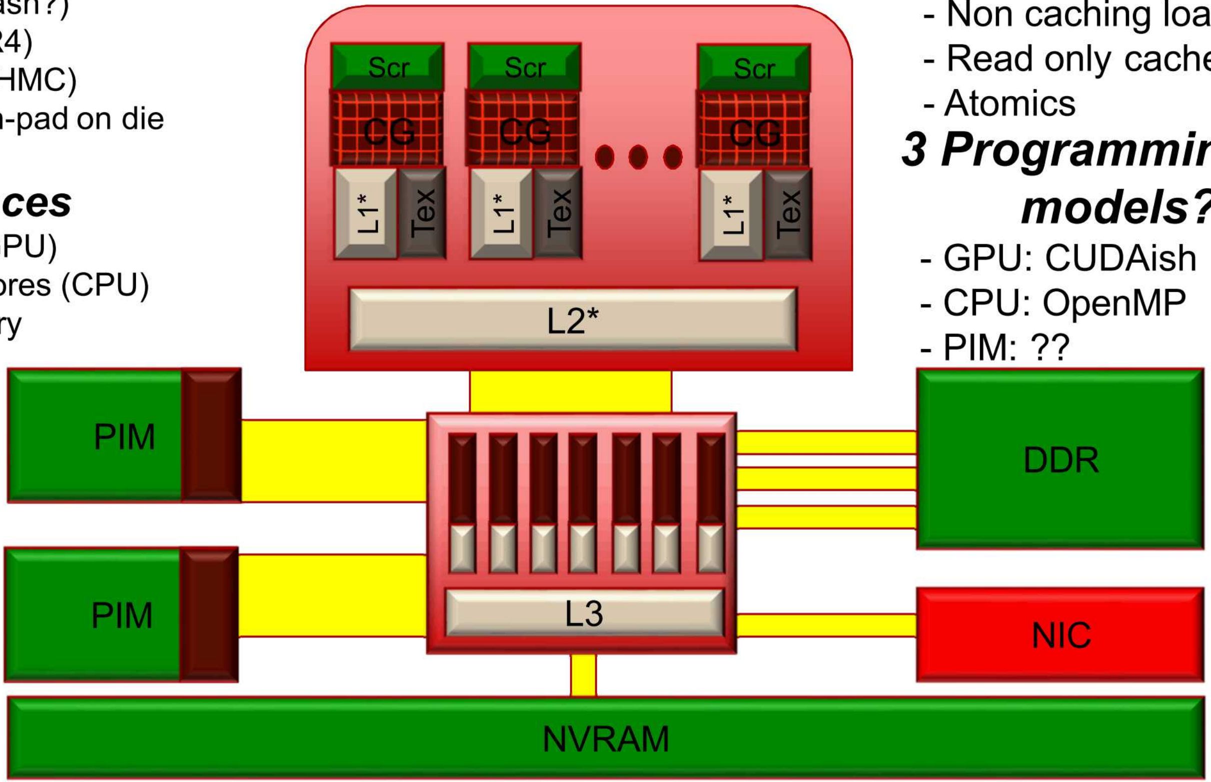
- Throughput cores (GPU)
- Latency optimized cores (CPU)
- Processing in memory

Special Hardware

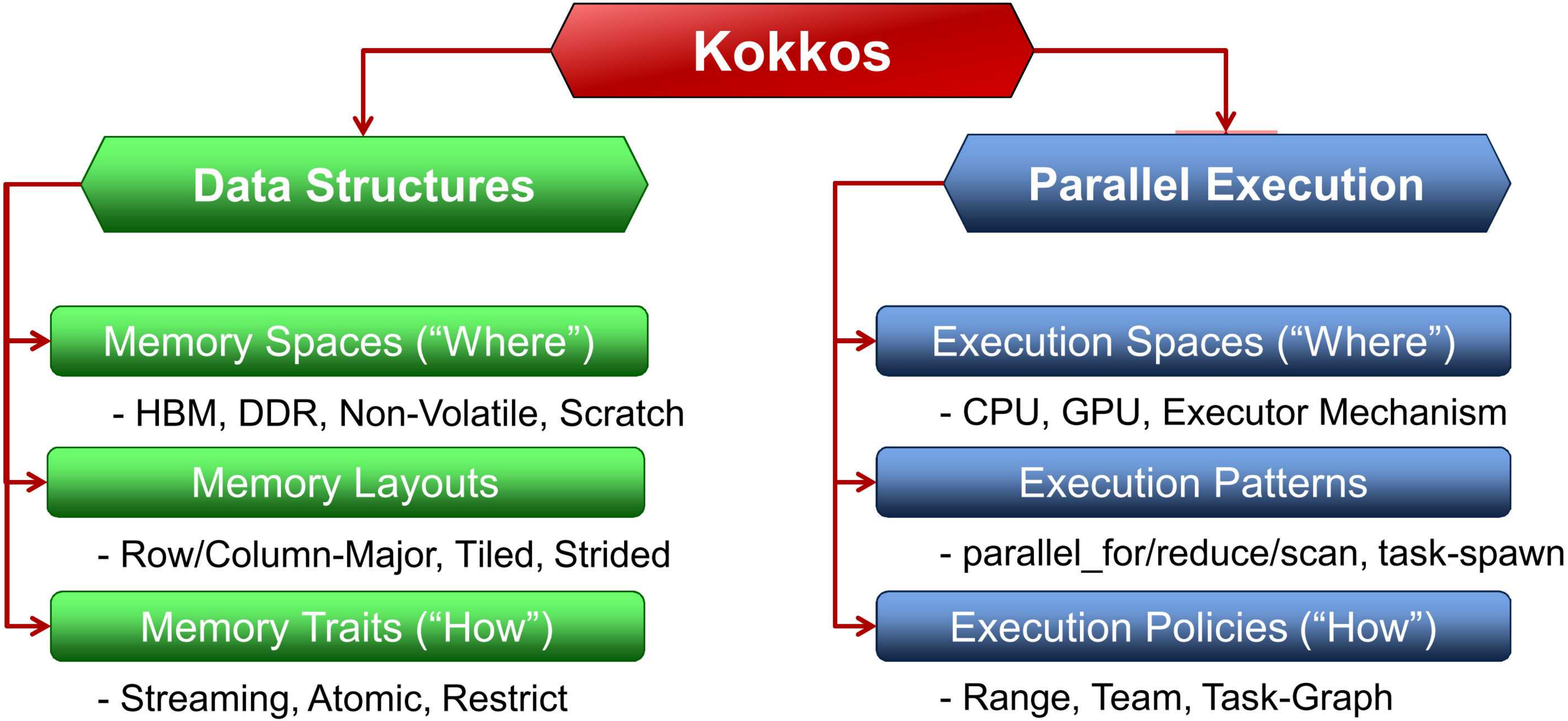
- Non caching loads
- Read only cache
- Atomics

3 Programming models??

- GPU: CUDAish
- CPU: OpenMP
- PIM: ??



Kokkos Abstractions



Kokkos

Data Structures

Memory Spaces (“Where”)

- HBM, DDR, Non-Volatile, Scratch

Memory Layouts

- Row/Column-Major, Tiled, Strided

Memory Traits (“How”)

- Streaming, Atomic, Restrict

Parallel Execution

Execution Spaces (“Where”)

- CPU, GPU, Executor Mechanism

Execution Patterns

- parallel_for/reduce/scan, task-spawn

Execution Policies (“How”)

- Range, Team, Task-Graph

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 += ...
	}, 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);
Exec Spaces	Serial, Threads, OpenMP, Cuda, ROCm *(experimental)*

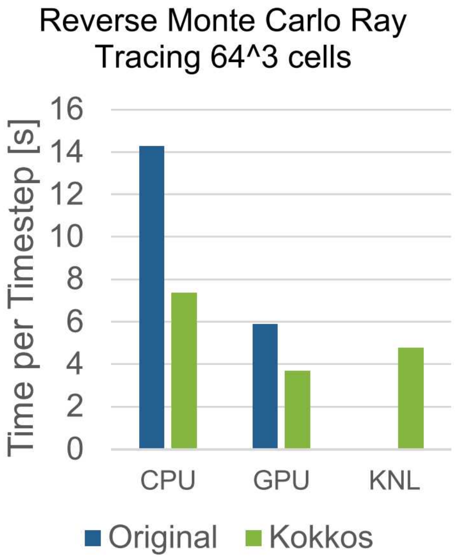
Kokkos Projects

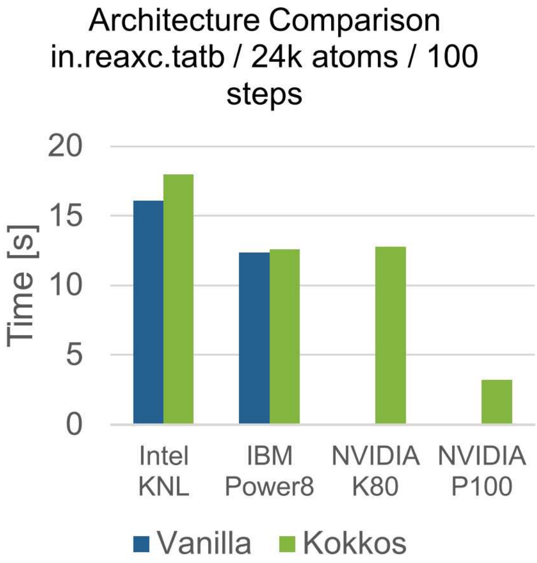
- Production Code Running Real Analysis Today
  - We got about **12** or so.
- Production Code or Library committed to using Kokkos and actively porting
  - Somewhere around **25**
- Packages In Large Collections (e.g. Tpetra, MueLu in Trilinos) committed to using Kokkos and actively porting
  - Somewhere around **50**
- Counting also proxyapps and projects which are evaluating Kokkos (e.g. projects who attended boot camps and trainings).
  - Estimate **80-120** packages.

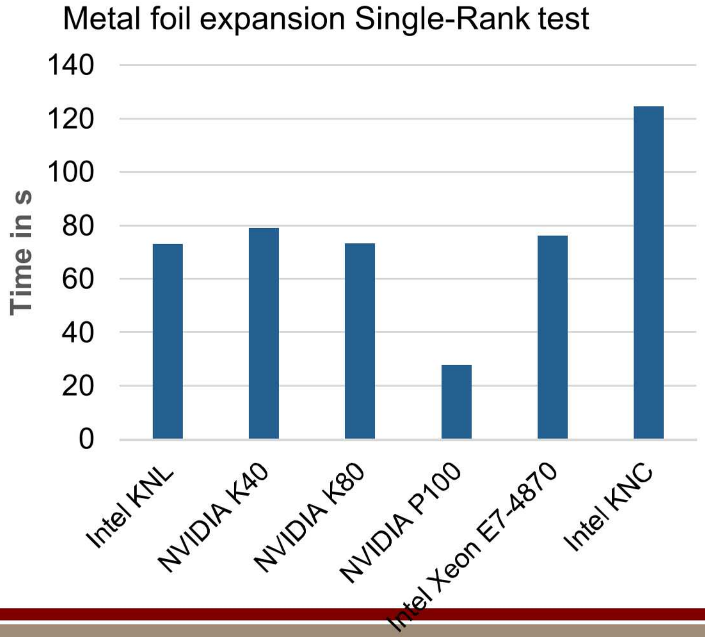
Uintah
- System wide many task framework
- Combustion Codes
- Did Gordon Bell Submissions
- University of Utah

LAMMPS
- Molecular Dynamics Code
- One of most widely used codes
- Often part of procurement benchmarks
- Sandia National Laboratories

Alexa
- Portably performant shock hydrodynamics application
- Solving multi-material problems Molecular Dynamics Code
- Sandia National Laboratories

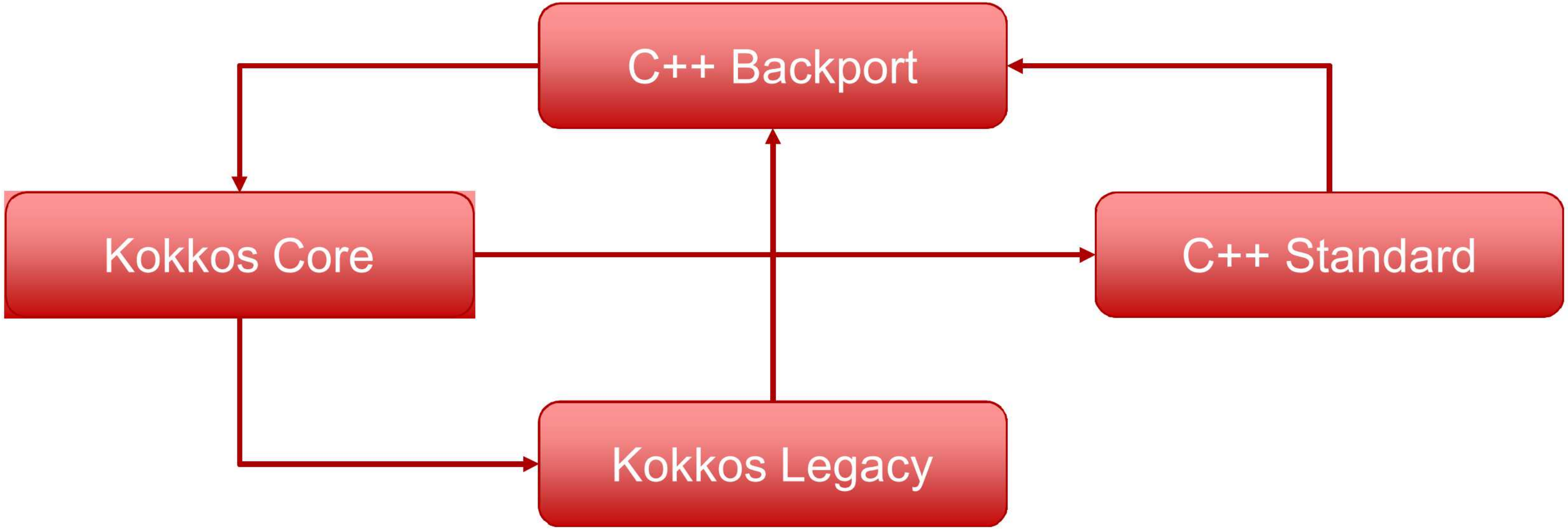






Aligning Kokkos with the C++ Standard

- 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)
- Also C++23: Executors and **Basic Linear Algebra** (just began design work)
- David will talk about most mdspan, atomic_ref and Executors

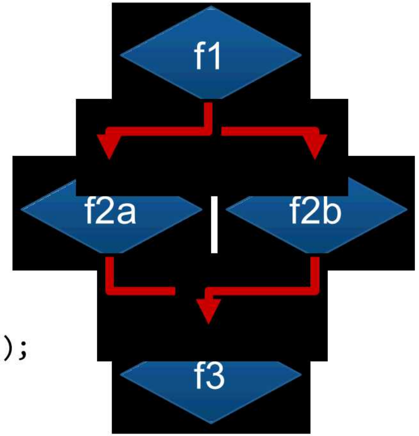
Towards C++23 Executors

- C++ standard is moving towards more asynchronicity with Executors
  - Dispatch of parallel work consumes and returns new kind of future
- Aligning Kokkos with this development means:
  - Introduction of Execution space instances

```
DefaultExecutionSpace spaces[2];
partition( DefaultExecutionSpace(), 2, spaces);
// f1 and f2 are executed simultaneously
parallel_for( RangePolicy<>(spaces[0], 0, N), f1);
parallel_for( RangePolicy<>(spaces[1], 0, N), f2);
// wait for all work to finish
fence();
```

  - Patterns return futures and Execution Policies consume them

```
auto fut_1 = parallel_for( RangePolicy<>(“Funct1”, 0, N), f1 );
auto fut_2a = parallel_for( RangePolicy<>(“Funct2a”, fut_1,0, N), f2a);
auto fut_2b = parallel_for( RangePolicy<>(“Funct2b”, fut_1,0, N), f2b);
auto fut_3 = parallel_for( RangePolicy<>(“Funct3”, all(fut_2a,fut2_b),0, N), f3);
fence(fut_3);
```



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

***C.R. Trott,** D. Sunderland, N. Ellingwood, D. Ibanez, S. Bova, J. Miles, D. Hollman, D. Lanbreche, V. Dang*