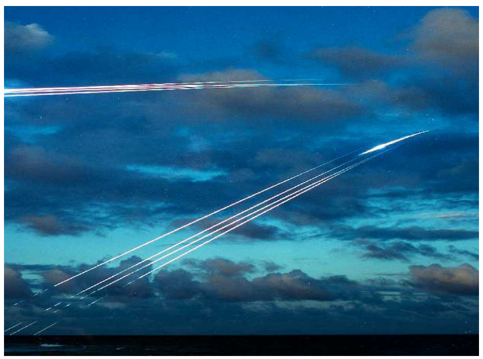
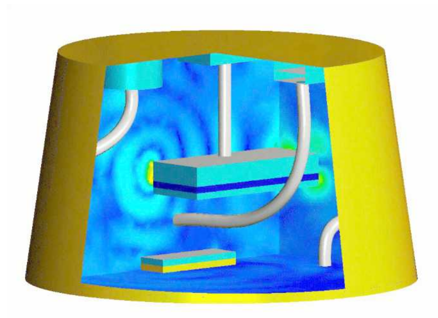
SAND2016-0709PE

Exceptional service in the national interest

Sandia National Laboratories





Sandia's Next Generation Code Architecture & Tools

Rob Hoekstra & ASC Team

January 27, 2016

U.S. DEPARTMENT OF ENERGY

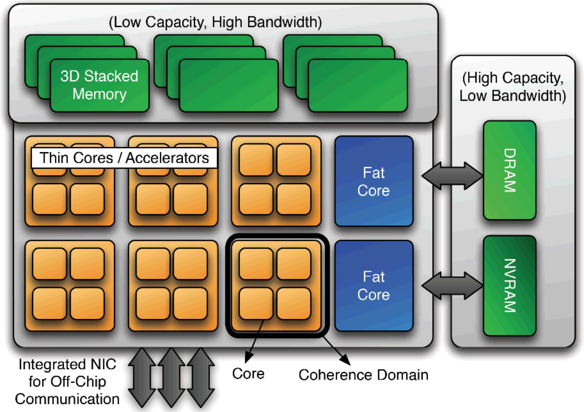
NNSA National Nuclear Security Administration

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. SAND NO. 2011-XXXXP

NextGen Software Architecture

Sandia National Laboratories

- ASC/ATDM gives us an opportunity to target new programming models and environments
  - Massive Parallelism
  - Performance Portability
  - Data Management
  - Asynchronous Multi-Tasking



COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

- We are building on DOE research efforts
  - ASC
    - Co-design, CSSE, Testbeds, PSAAP2 Centers, etc.
  - ASCR
    - Co-design, X-Stack, RX-Solvers, CAL, etc.

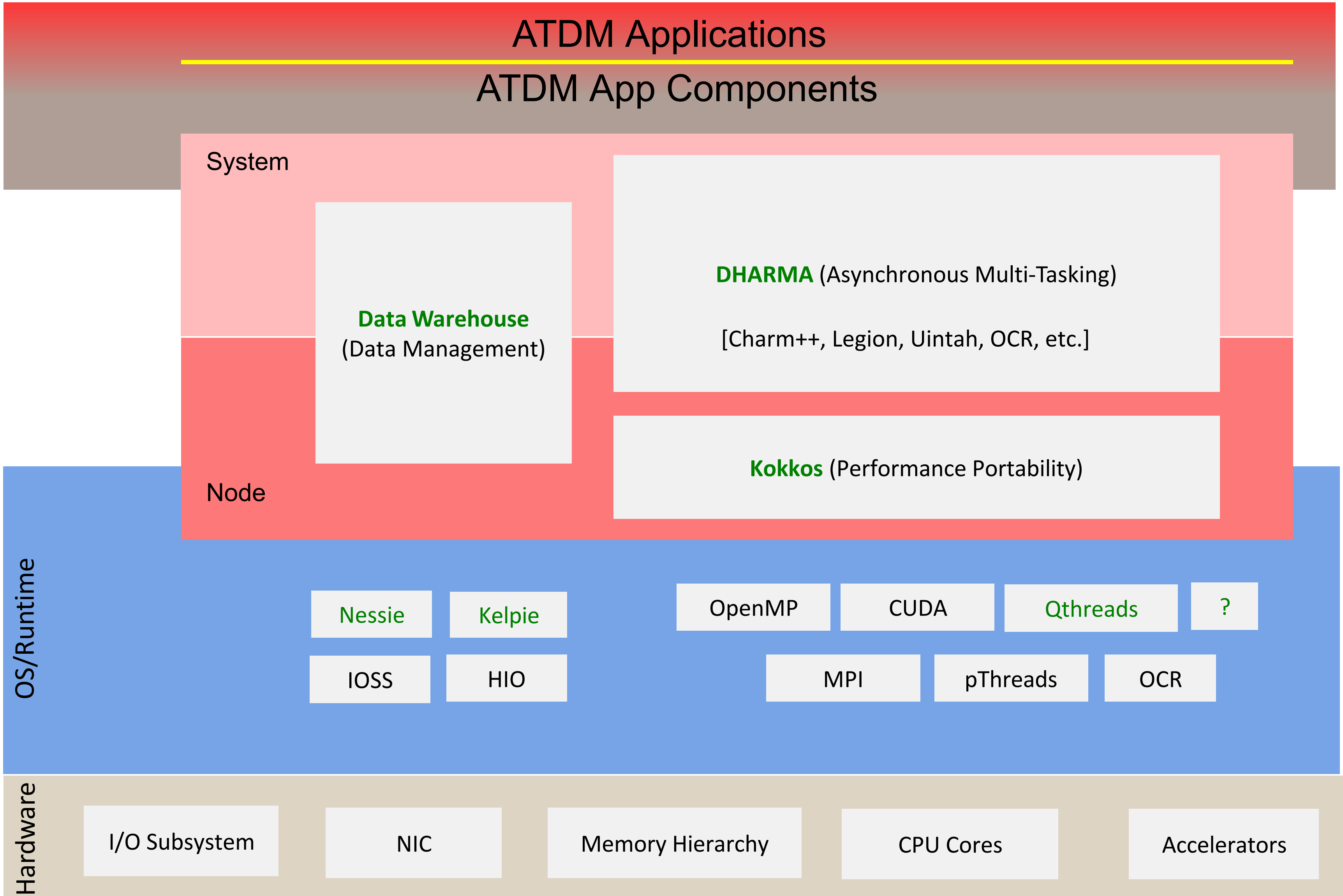
Future Programming Model/Envir.

- Opportunity to start from a “clean sheet”:

- Architect the codes targeting potential future HPC PM/Es
  - AMT for moderate to coarse-grained tasking
  - Kokkos data parallelism (and possibly DAG tasking) within AMT tasks
  - Data Warehouse to enable AMT task movement and efficient movement across “memory spaces” and complex I/O layers

- Benefits/Goals
  - Dynamic, system managed workload distribution
  - Infrastructure which supports resiliency mechanisms
  - New algorithmic approaches (asynchrony, hierarchy, thread scalability)
  - Productivity/reduced complexity for application developer?

ATDM CS Core Components



DHARMA project: Distributed asyncHronous Adaptive and Resilient Models for Applications

- A co-design driven specification effort at the top of the AMT runtime system stack
  - Start by gathering requirements across Sandia's ATDM application space
  - Approach is extensible to incorporating requirements from other Labs
- An effort to implement that spec
  - Implementation is an integration effort
  - Includes all ATDM CS Components
    - not just the DHARMA team
- A long term engagement with the community to define best practices and ultimately standards
  - Communication of Sandia ATDM requirements
  - Defining common vocabularies and shared abstractions across runtimes

FY15 L2 milestone to assess leading AMT runtimes in the context of ASC workloads

We face a spectrum of choices/risks in developing technical roadmap

Build system from scratch and take ownership

Risk: potential lack of vendor support/buy in

Lots of control, but lots of extra investment

Rely completely on external partners

Risk: current academic runtimes lack features to support our workloads

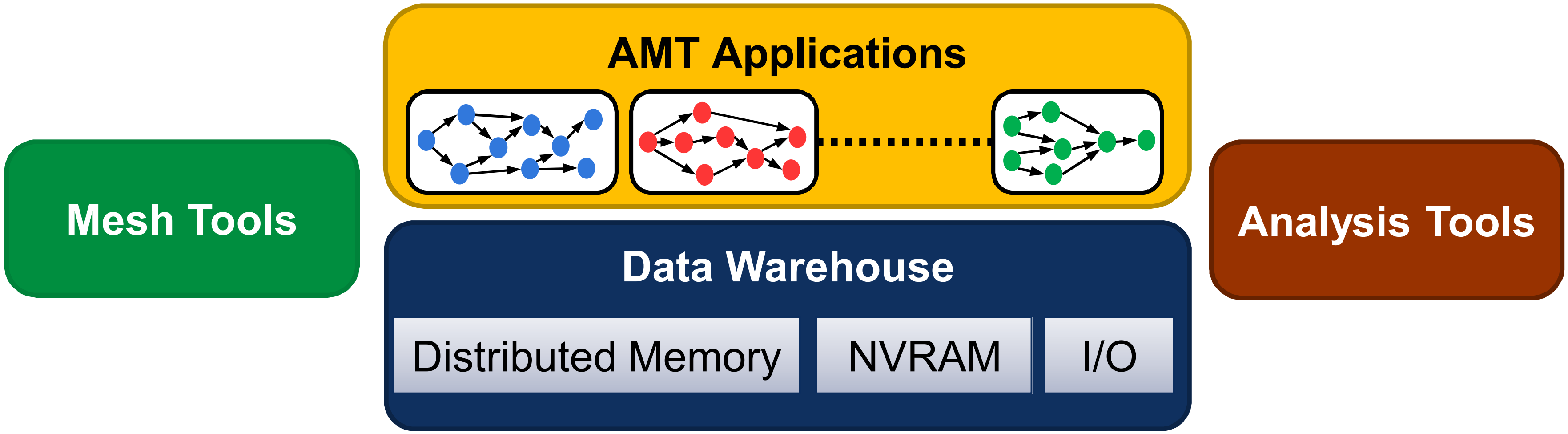
Less control, but less investment

- **Programmability:** Does this solution enable expression of our workloads
- **Mutability:** Ease of adopting this solution, modifying it to suit our needs
- **Performance:** Strong and weak scaling studies, load-balancing under system heterogeneity, task- and data-granularity studies

Data Management: Warehouse

Sandia National Laboratories

- *Data management* is a large challenge at scale
  - Practical: How do we move data between existing tools and AMT?
  - Persistence: How will AMT codes leverage new I/O capabilities?
  - Future Proof: How can we mitigate reliability risks?
- Build a flexible, multipurpose *data warehouse*
  - Engine by which application data is moved and stored in ATDM

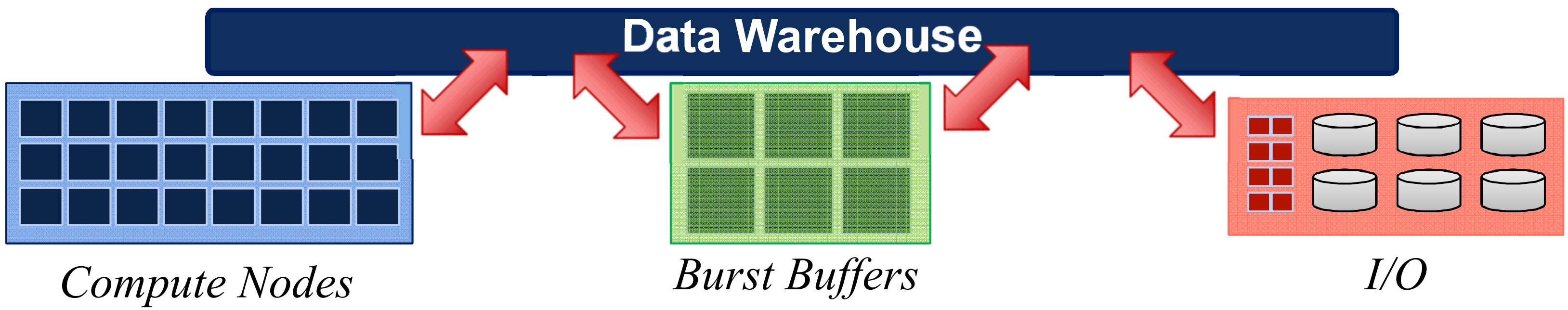


Data Warehouse Supports Key Application Use Cases

Sandia National Laboratories

Workflows

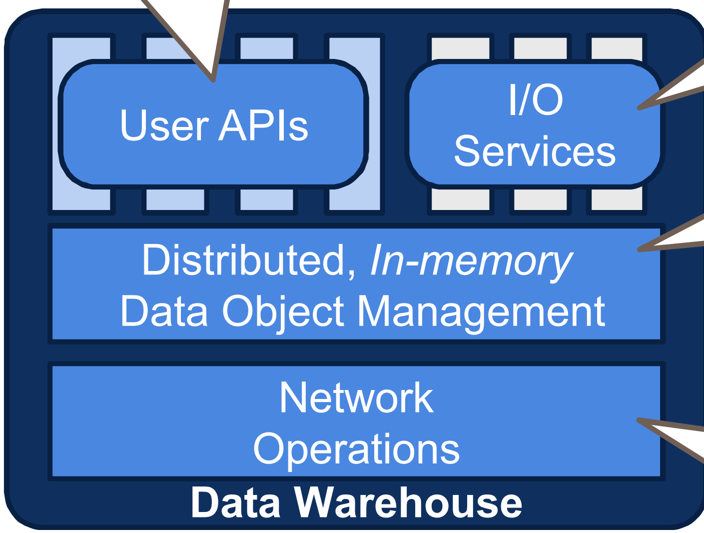
Resource Abstraction



Task-DAG Communication

Warehouse Software Architecture

Custom views of data warehouse
- App-specific APIs
- Data distribution



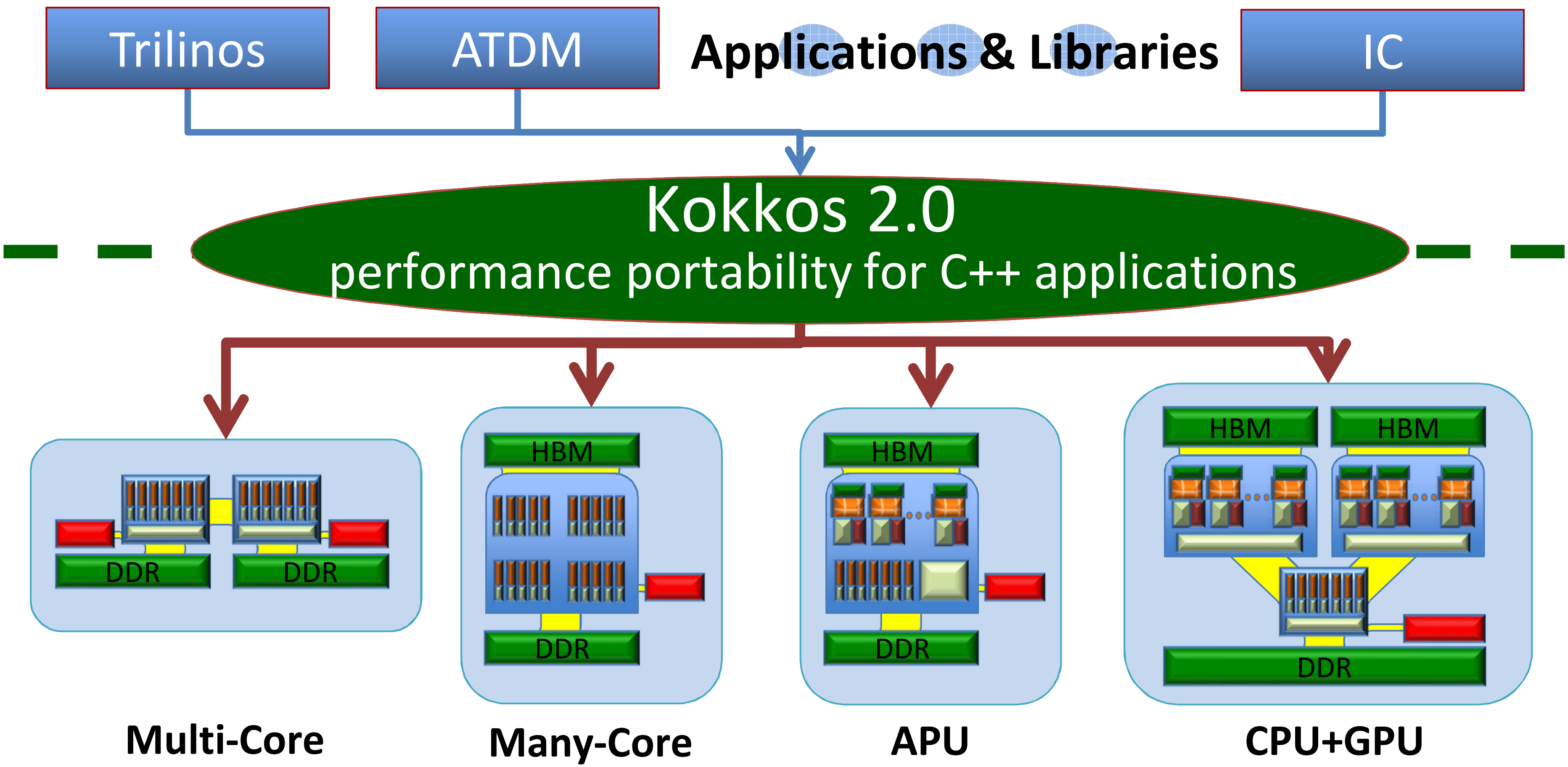
Short/long-term persistence
- Burst Buffers, DFS
- Interest in HIO

Kelpie

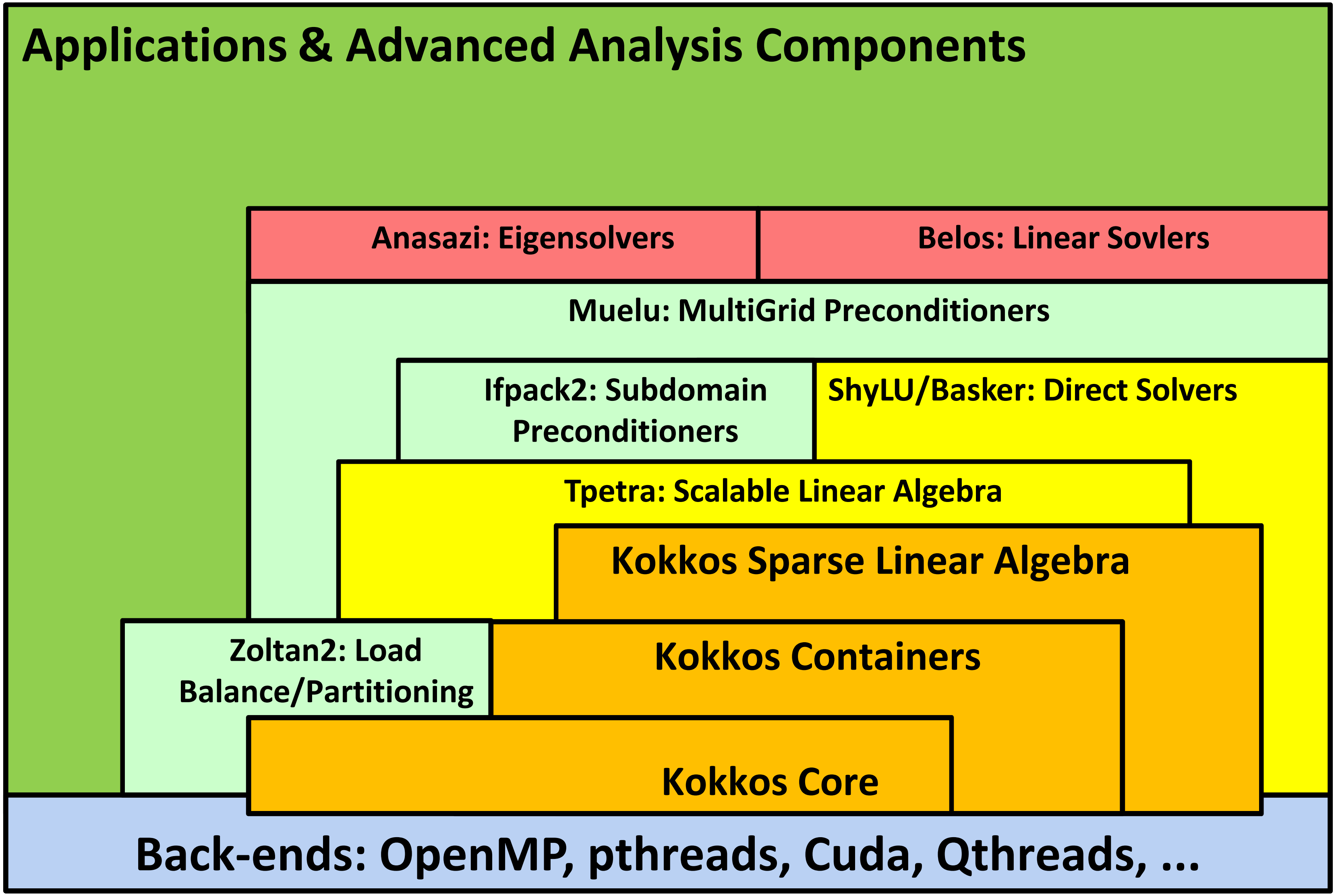
Manage key-value objects via RDMA
Organize resource pools

Portable RDMA/RPC for HPC
- BG/Q, Gemini, IB, MPI, …

Performance Portability (Kokkos)

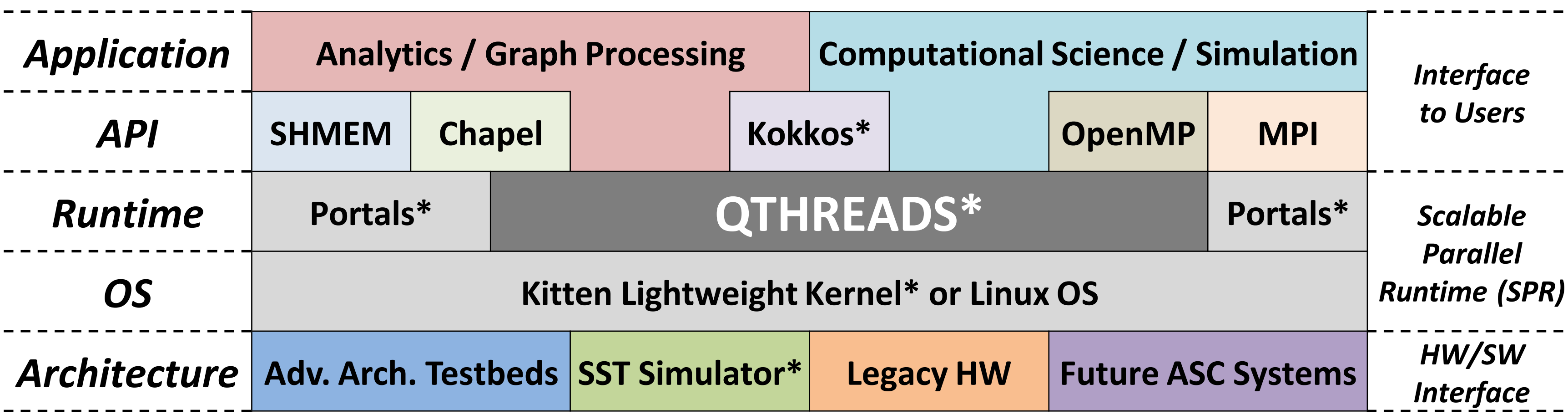


Kokkos Kernels for Trilinos Solvers



On-Node Runtime: Qthreads

- Qthreads is a user-level library for lightweight multithreading on the node, accessible directly using a C API or indirectly through a variety of higher-level programming models.
- Qthreads originated at Notre Dame and moved to Sandia.
- Initial motivation: Support graph processing / analytics applications on commodity hardware using runtime-managed massive multithreading and rich synchronization primitives.



Qthreads System

- The programmer exposes application parallelism as massive numbers of lightweight tasks (qthreads).
  - Full/empty bit primitives for powerful, lightweight synchronization
    - Emulates behavior of Cray XMT (ThreadStorm) architecture
  - C API with no special compiler support required
- The run time system dynamically manages the scheduling of tasks for locality and scalable performance.
  - Heavyweight worker pthreads to execute the user's tasks
    - Worker pthreads pinned onto underlying hardware cores
    - Architecture-aware mapping of workers to hardware (e.g., NUMA or Phi)
  - Lightweight task switching
- Used in: Cray's Chapel programming language, Kokkos fine-grained threading, Multi-threaded graph library(MGTL), LLNL ROSE front-ended OpenMP run time

On-Node Runtime Coordination

Scalable performance portable on-node runtime

- FY15: Leverage prior ASC-CSSE and ASCR investment in Qthreads runtime system
- FY16 L2: Support runtimes needs for Kokkos and AMT
- FY19 L1: Performant on-node tasking for high-performance ATDM apps on Advanced Technologies Systems (ATS-3 in particular)

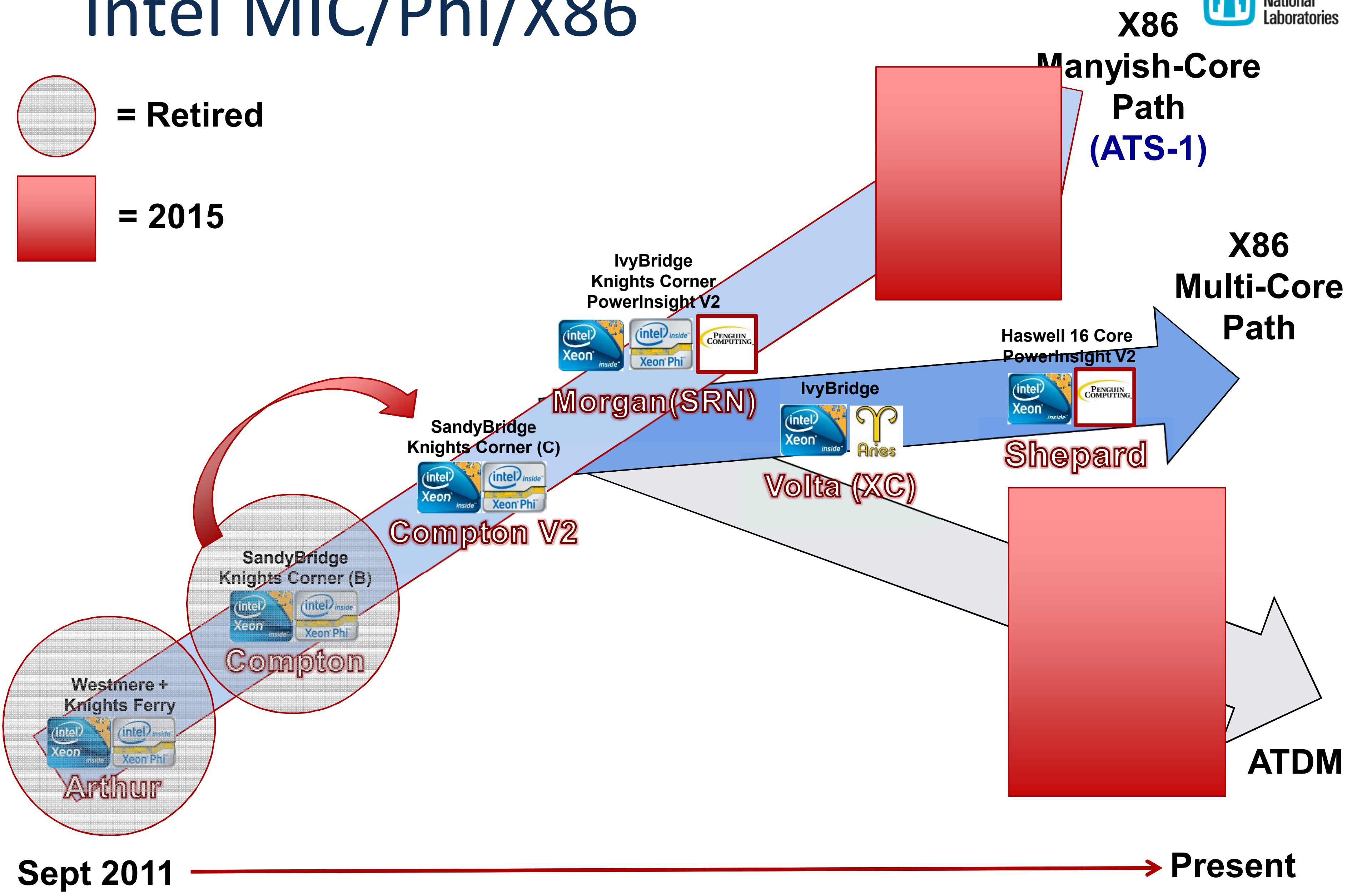
Outcome(s): Provide scalable and performance portable on-node runtime and related system software in support of Kokkos, AMT, and application needs

Dev. Environment/Testbeds

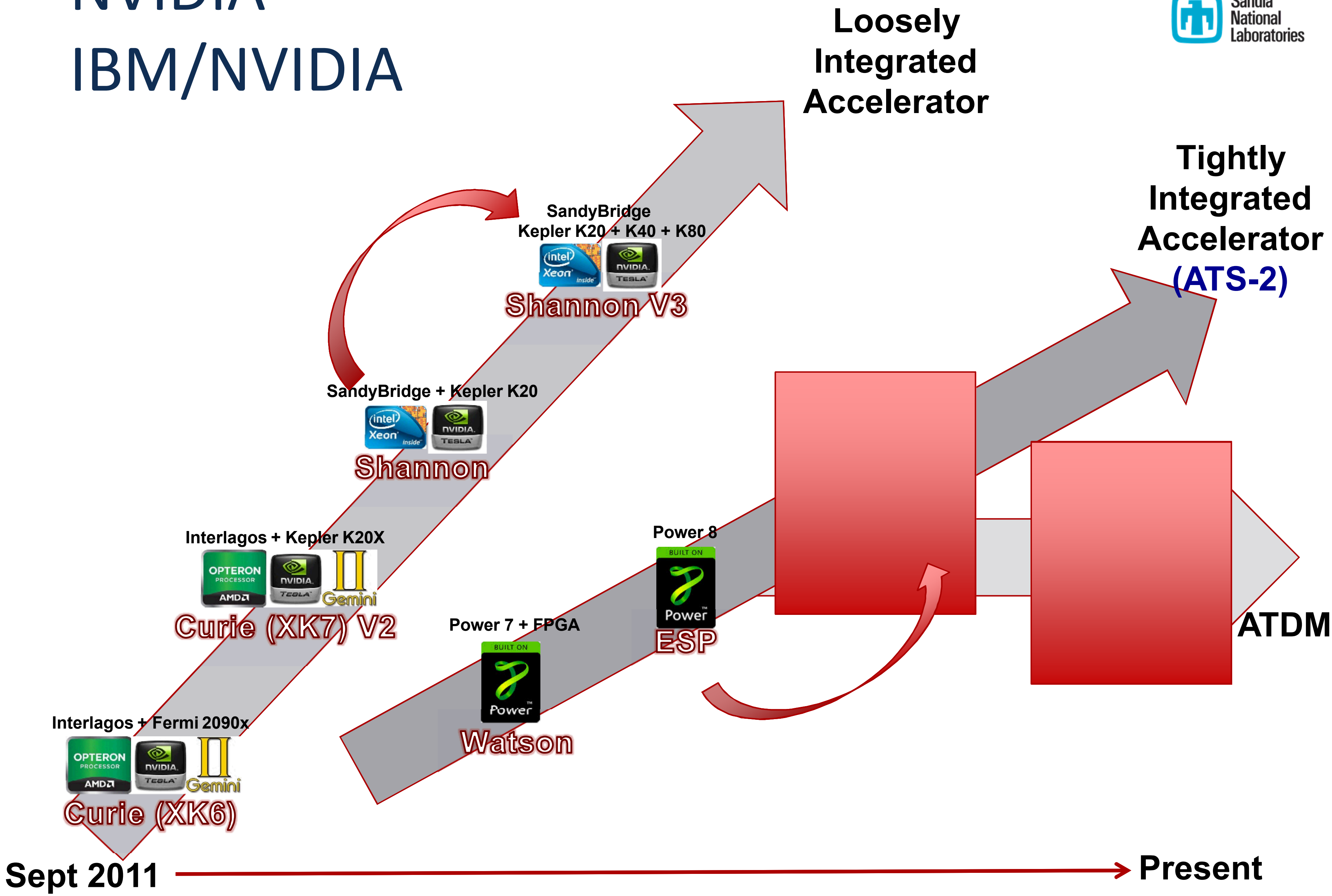
- Next Generation Development Environment
  - Increased performance and efficiency for developers
  - Robust regression testing
  - Deployment across range of platforms including testbeds

- SNL's ASC program has supported a thriving CSSE testbed program
  - Includes pre-production HW from majority of vendors
  - ATDM is leveraging and augmenting this resource
  - Early HW has proven crucial to:
    - Testing of system software stack and compilers prior to production platform delivery
    - Exploration of performance implications for our proxies and codes

Intel MIC/Phi/X86



NVIDIA
IBM/NVIDIA



Risks/Gaps

- Asynchronous Multi-Task
  - Can existing technologies be extended to allow the dynamic behavior required by our extremely complex codes?
  - Can it out perform simpler MPI-based alternatives?
- Data Management
  - Can the overhead be minimized enough to not damage code performance?
  - Can this layer facilitate on-node as well as off-node data movement?
- Performance Portability
  - Can Kokkos significantly reduce HW specific code in our applications?
  - Pros and cons of embedded DSL approach to directive based alternatives such as OpenMP?
- Overarching
  - Instrumentation/Tools that can expose meaningful information throughout this stack

Outcomes

- We are leveraging years of prior ASC and ASCR efforts and ASC Co-design has driven significant NNSA tri-lab collaboration for this space; many more commonalities.
- We are evaluating promising new tools for productivity and performance on new architectures; but they create a much deeper layer of constructs that the developer does not control.
- And doing it with highly complex near production codes
- Synergy, leverage and enhancement with the broader community is critical to success

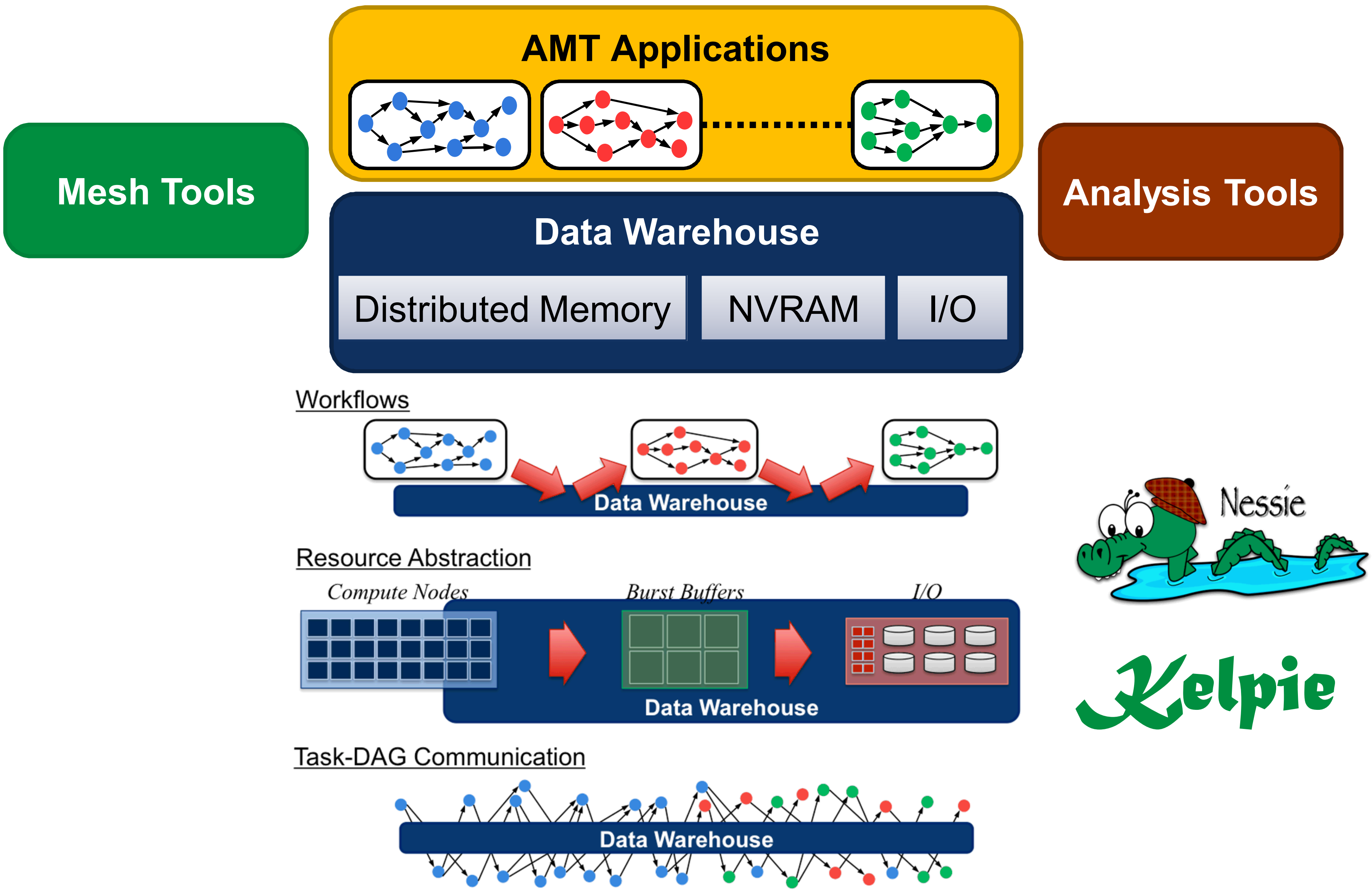
Acronyms

- AMT – Ansynchronous Multi-Tasking
- API – Application Programming Interface
- ATDM – Advanced Technology Development & Mitigation
- CAL – Computer Architecture Lab
- CSSE – Computational Systems & Software Environment
- DAG – Directed Acyclic Graph
- DFS – Distributed File System
- DSL – Domain Specific Language
- EM - Electromagnetics
- IB - Infiniband
- MPI – Message Passing Interface
- PM/Es – Programming Models/Environments
- PSAAP – Predictive Science Academic Alliance Program
- RDMA – Remote Direct Memory Access
- RPC – Remote Procedure Call
- SGEMP – System Generated Electromagnetic Pulse

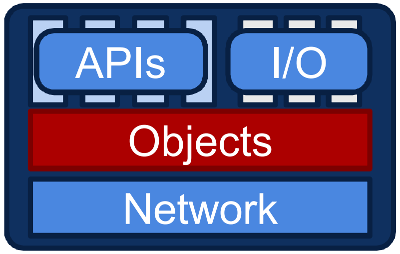
BACKUP SLIDES

Sandia National Laboratories

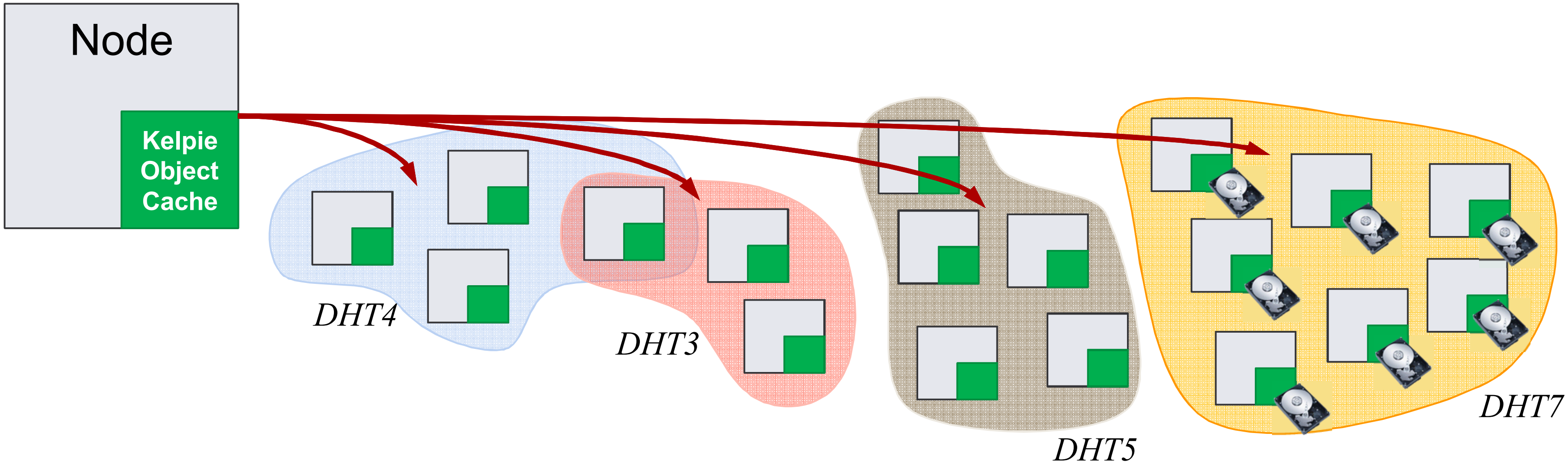
Data Warehouse



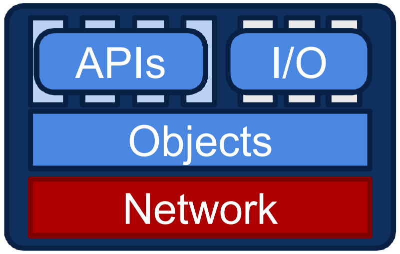
Distributed Objects: Kelpie



- **Kelpie**: An in-memory, distributed object store
  - Peer-based: each host manages a cache of RDMA-able objects
  - Multi-dimensional key that maps to 2D data organization
  - Common interfaces (put/get, pub/sub, prefetch,…)
- FY15: Manage collections of nodes with resource pools
- FY16: Internal rework to streamline API

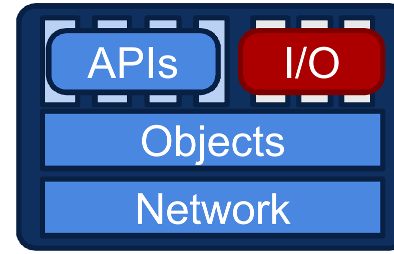


Network: Nessie

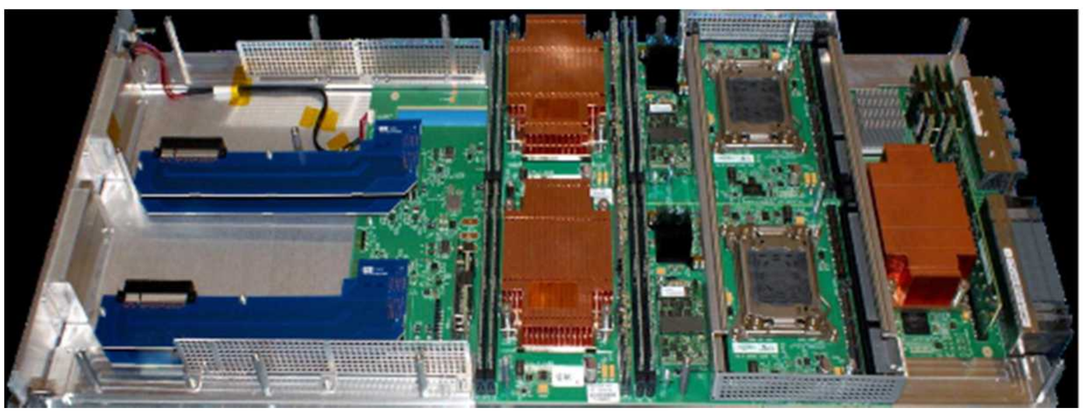


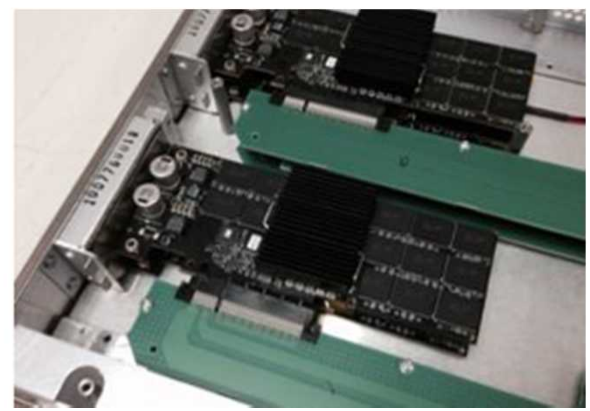
- Task DAGs: Advantageous to work below MPI
  - Examples: GASnet, Dataspaces, libfabrics, uGNI, PAMI
- Hardships: Memory registration, Serialization
- **Nessie**: Sandia-developed RDMA+RPC layer
  - *Explicit* buffer management
- FY15 Progress
  - Lunasa: Dynamic memory manager
  - Nssi-Lite: Stripped down RPC layer
  - NNTI 3.0: Event driven
- FY16 Work
  - New fabrics, Performance optimization

I/O Services



- Enable objects to be made persistent
  - Nonvolatile Memory (burst buffers)
  - Parallel File Systems
- FY16: Prototype storage interface
  - Leverage Hierarchical I/O (HIO) , target Trinity





Qthreads Objectives and Outcomes

- Be efficient for both analytics and computational science
  - Many other run time systems address only one or the other
- Be a vehicle for run time system research
  - Enable co-design leveraging Sandia expertise across the system stack
  - Use modularity for flexibility, extensibility
    - Support for diverse architectures and programming models
- Improve understanding of system and application behavior
  - Test and challenge efficiency and scalability limits
- Impact deployed run time system technologies
  - Develop solutions to unsolved problems in adaptive run time systems
  - Present lessons learned to industry and the community
    - Vendors apply the new techniques to their implementations
    - In the case of the Chapel language, Cray has adopted our run time

Qthreads-based Software

- Chapel programming language
  - Chapel is Cray's next generation parallel programming language.
  - Qthreads provides the default tasking layer for the Chapel run time.
- Multithreaded graph library (MTGL)
  - MTGL is Sandia's toolkit for graph processing algorithms.
  - Qthreads allows MTGL, originally designed for the Cray XMT, to execute on commodity systems such as x86 and POWER machines.
- Kokkos task-parallel extensions *[In progress]*
  - Kokkos is Sandia's C++ library for efficient management of data layout and parallelism for manycore processors.
  - Qthreads supports extensions to Kokkos that add task parallelism (futures) to the existing Kokkos data parallel capabilities.
- Qthreads also serves as an OpenMP run time using the LLNL ROSE compiler as the front-end.