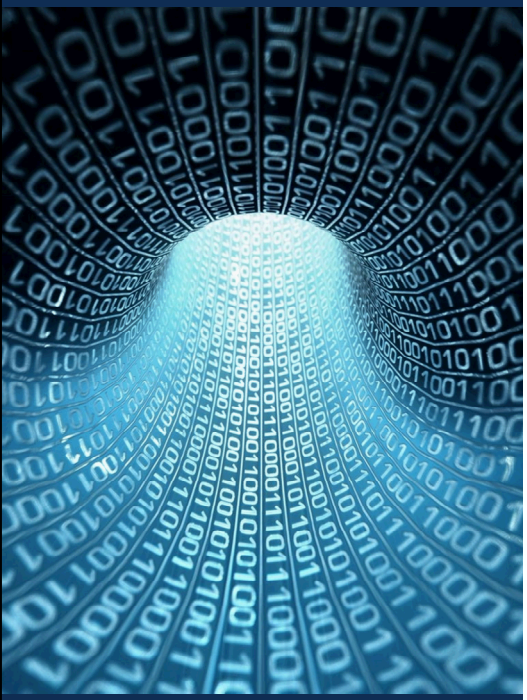


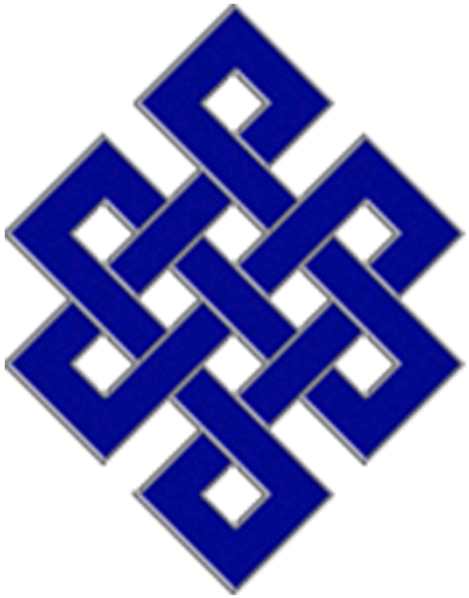
Sandia National Laboratories

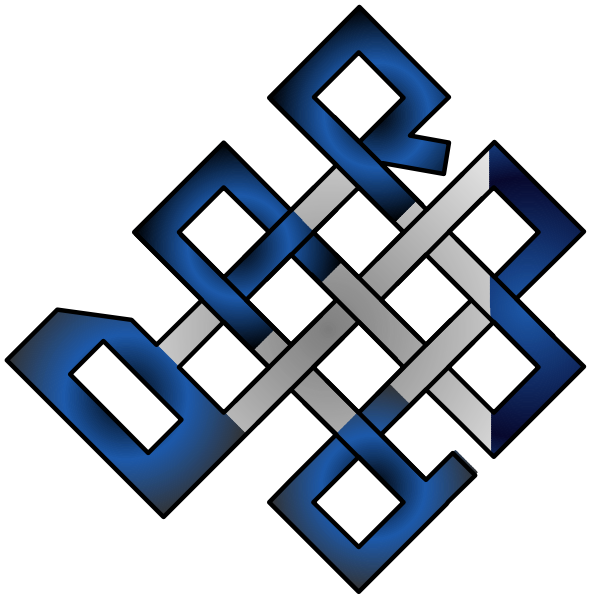
*Exceptional service in the national interest*

SAND2016-3270PE

# The DARMA Approach to Asynchronous Many-Task (AMT) Programming

***Jeremiah J. Wilke***, David S. Hollman, Nicole Slattengren, Hemanth Kolla, Francesco Rizzi, Keita Teranishi, Janine C. Bennett (PI), Robert L. Clay (PM)





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

SAND2016-0930 C

# CHAPTER 1:
# THE ORIGIN STORY

# Asynchronous Many-Task (AMT) runtimes address key performance challenges posed by future architectures

Sandia National Laboratories

- Performance challenges:
  - Utilizing whole machine requires more parallelism
  - Managing deep memory hierarchies requires flexible staging of data/assigning work
  - Handling dynamic workloads requires flexible task scheduling

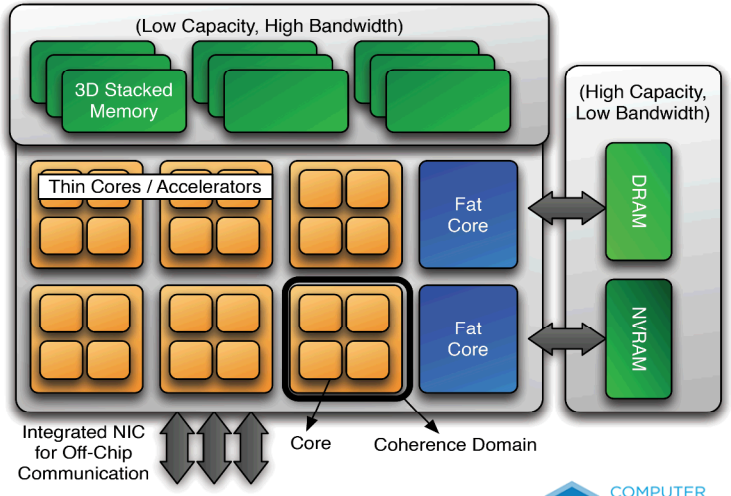
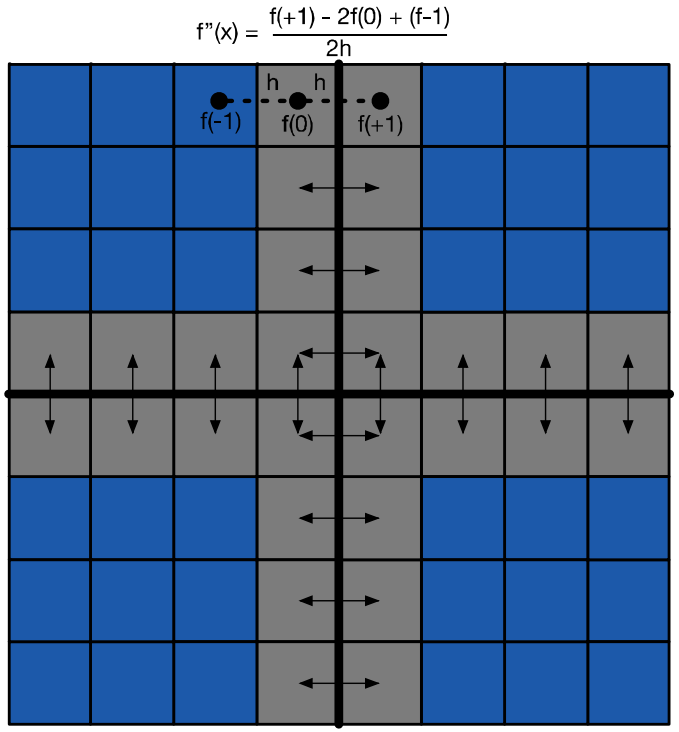


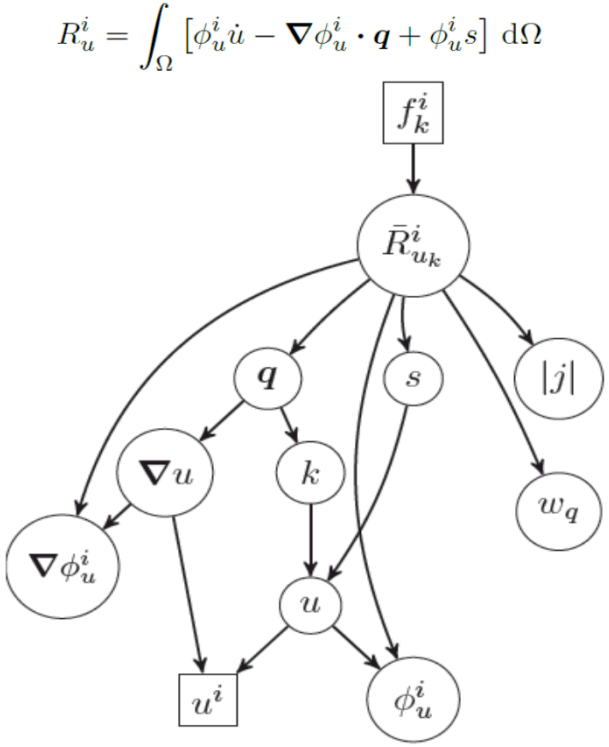

Image courtesy of www.cal-design.org

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

- **Asynchronous**: express all possible parallelism and minimize/hide communication/scheduling latency
- **Many-task**: Chunks of work of ``correct'' granularity that can be flexibly assigned to different memory/execution spaces

# Matrix assembly a priori knows all parallelism, but managing/deriving parallelism is difficult

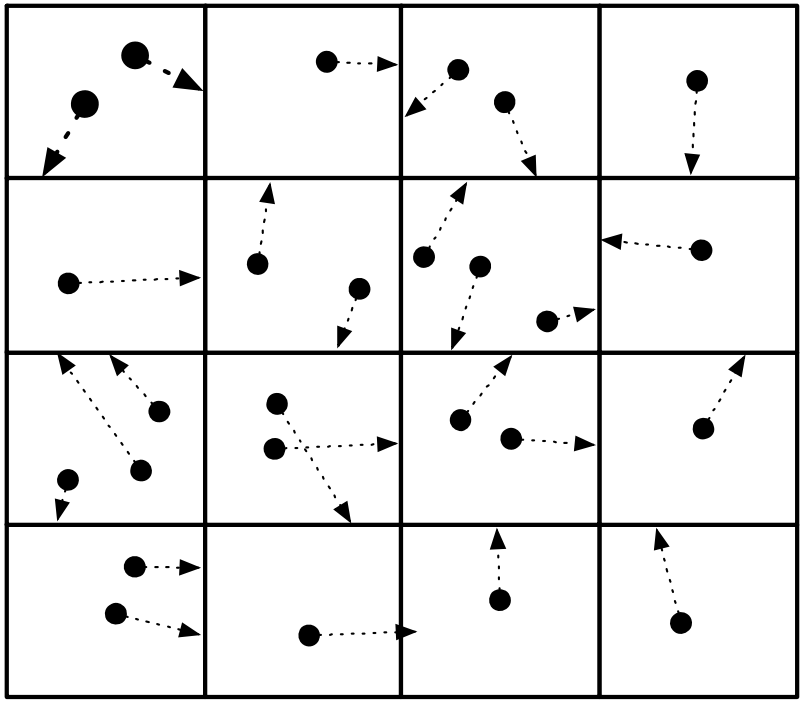



- Ghost exchange has fixed communication pattern
- On-node DAG can be complex, but statically (semi-statically) known
- Given data dependencies, compiler/runtime can maximize parallelism
- App developer doesn't really need to understand execution/concurrency models if data model is good enough

# Particle-in-cell (PIC) exhibits has dynamic data distribution/load balancing challenges

Sandia National Laboratories

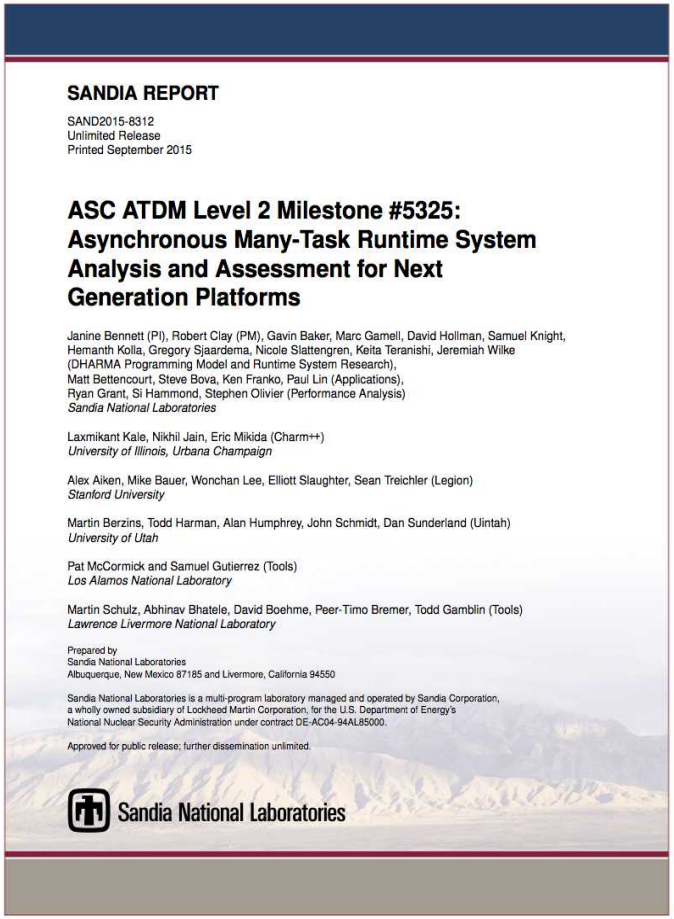


- Particles migrate irregularly through cells
- An idle cell may not STAY idle
- Load-balancing problems for inhomogenous particle distributions
- Some task parallelism, but major challenge is unknown parallelism

# Sandia led a comparative analysis study of leading AMT runtimes to inform our technical roadmap

Sandia National Laboratories

- Broad survey of many AMT runtime systems
- Deep dive on Charm++, Legion, Uintah
- Programmability
  - Does this runtime enable efficient expression of our workloads?
- Performance
  - How performant is this runtime for our workloads on current platforms?
  - How well suited is this runtime to address exascale challenges?
- Mutability
  - What is the ease of adopting this runtime and modifying it to suit our needs?



**SANDIA REPORT**

SAND2015-8312
Unlimited Release
Printed September 2015

**ASC ATDM Level 2 Milestone #5325: Asynchronous Many-Task Runtime System Analysis and Assessment for Next Generation Platforms**

Janine Bennett (PI), Robert Clay (PM), Gavin Baker, Marc Gamell, David Hollman, Samuel Knight, Hemanth Kolla, Gregory Sjaardema, Nicole Slattengren, Keita Teranishi, Jeremiah Wilke (DHARMA Programming Model and Runtime System Research), Matt Bettencourt, Steve Bova, Ken Franko, Paul Lin (Applications), Ryan Grant, Si Hammond, Stephen Olivier (Performance Analysis)
*Sandia National Laboratories*

Laxmikant Kale, Nikhil Jain, Eric Mikida (Charm++)
*University of Illinois, Urbana Champaign*

Alex Aiken, Mike Bauer, Wonchan Lee, Elliott Slaughter, Sean Treichler (Legion)
*Stanford University*

Martin Berzins, Todd Harman, Alan Humphrey, John Schmidt, Dan Sunderland (Uintah)
*University of Utah*

Pat McCormick and Samuel Gutierrez (Tools)
*Los Alamos National Laboratory*

Martin Schulz, Abhinav Bhatele, David Boehme, Peer-Timo Bremer, Todd Gamblin (Tools)
*Lawrence Livermore National Laboratory*

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

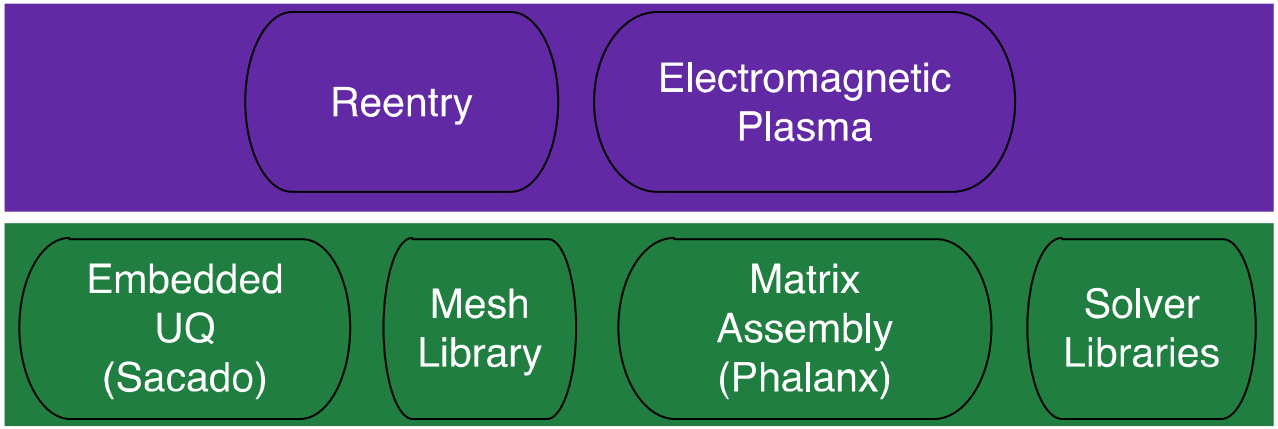
# Lessons learned from study led to application-driven programming model specification co-design effort

Sandia National Laboratories

- Data, task, and pipeline parallelism can be expressed in different ways
  - Explicit parallelism vs apparently sequential semantics
  - Arbitrary data structures vs strong data model
  - Runtime vs user-level control
  - New language vs embedded in C/C++
- Model should enhance performance, productivity, resilience
  - Applications should not be (much) more difficult to write than MPI
  - Make difficult things manageable, e.g. load balancing, fault-tolerance
- Design space tradeoffs need further assessment prior to committing to a single runtime
  - Across variety of applications and architectures
  - Further research required in some aspects of runtime (e.g., resource management)

# Sandia needs a software stack that supports diverse applications and provides performance portability

Sandia National Laboratories




Applications

What goes here?!?!?!?

OS/Hardware

Runtime software and hardware

# Vendor-supported runtime system and standards are ideal but AMTs are still an active research area

Sandia National Laboratories

## 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

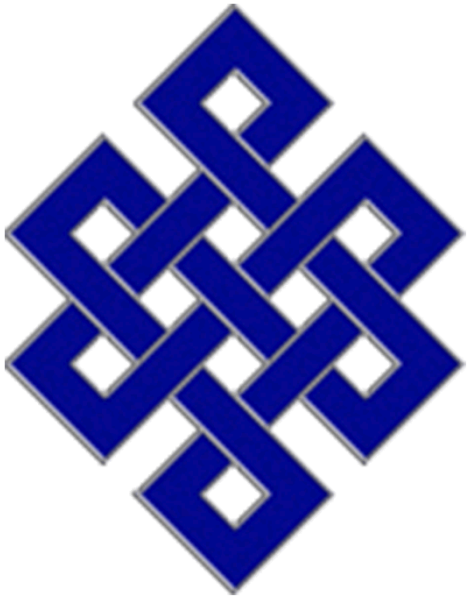
Rely completely on external partners

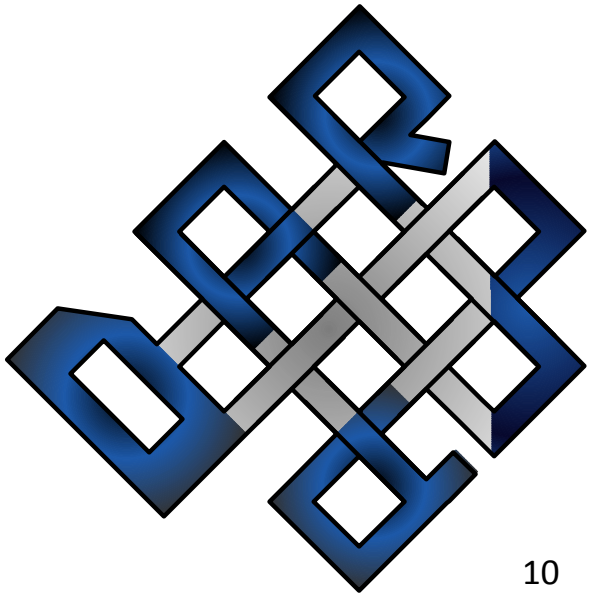
Risk: current academic runtimes may lack features to support our workloads

Less control, but less investment

# DARMA is the reincarnation of DHARMA

- ***D***istributed, ***A***synchronous, ***R***esilient ***M***odels for ***A***pplications
- Originally fault-tolerance: ***D***istributed ***H***ash ***A***rray for ***R***esilience in ***M***assively parallel ***A***pplications
- Dhr is sanskrit meaning to hold, keep
- Programming model concerns for fault-tolerance similar to AMT
  - Simplify reasoning about code correctness
  - Latency hiding
  - Recoverable (migratable) chunks of work

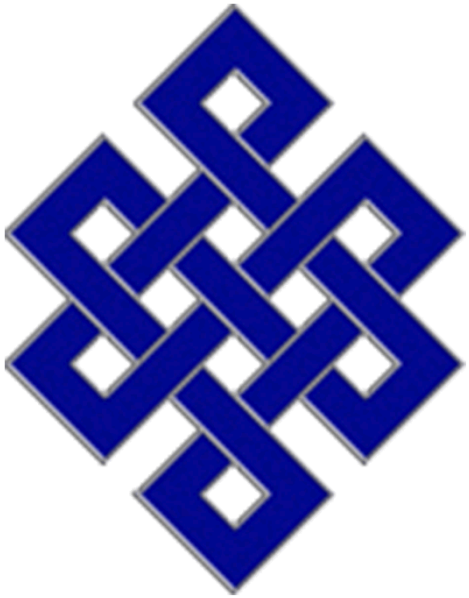


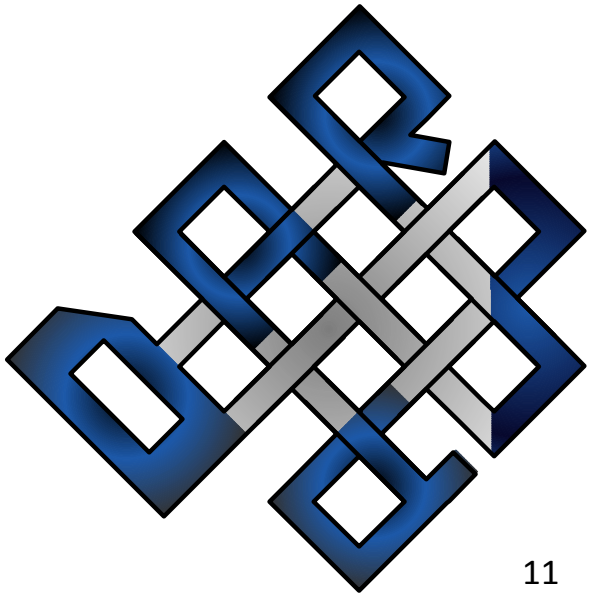


# DARMA is the reincarnation of DHARMA

Sandia National Laboratories

- ***D***istributed, ***A***synchronous, ***R***esilient ***M***odels for ***A***pplications
- Originally fault-tolerance: ***D***istributed ***H***ash ***A***rray for ***R***esilience in ***M***assively parallel ***A***pplications
- Dhr is sanskrit meaning to hold, keep
- Programming model concerns for fault-tolerance similar to AMT
  - Simplify reasoning about code correctness
  - Latency hiding
  - Recoverable (migratable) chunks of work



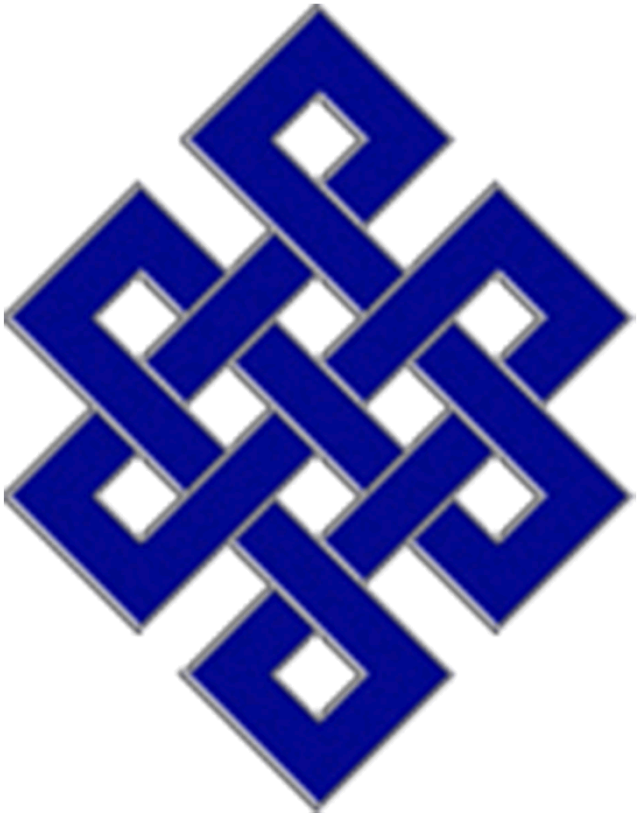


## Noble Truths of HPC

- All life is suffering
- Our desire for more flops is the source of our suffering

# CHAPTER 2:
# THE HIGH-LEVEL TECHNICAL STORY

# The exascale software stack should NOT look like this

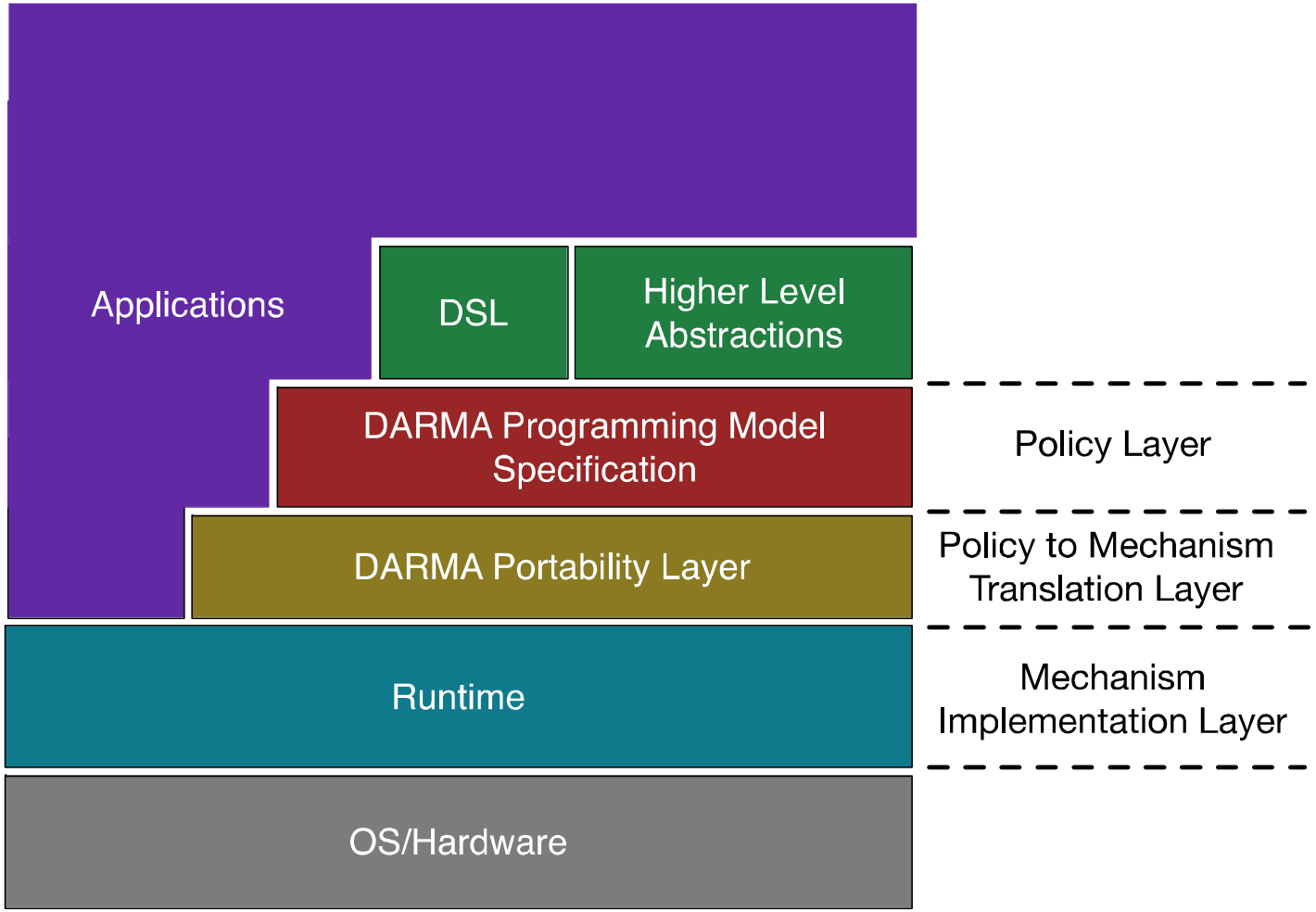


Is the only way to achieve exascale a monolithic, tightly-coupled software stack?

Or can we have well-defined, independent components?

# DARMA software stack separates policy and mechanism

Sandia National Laboratories




Policy: Express correctness and performance requirements

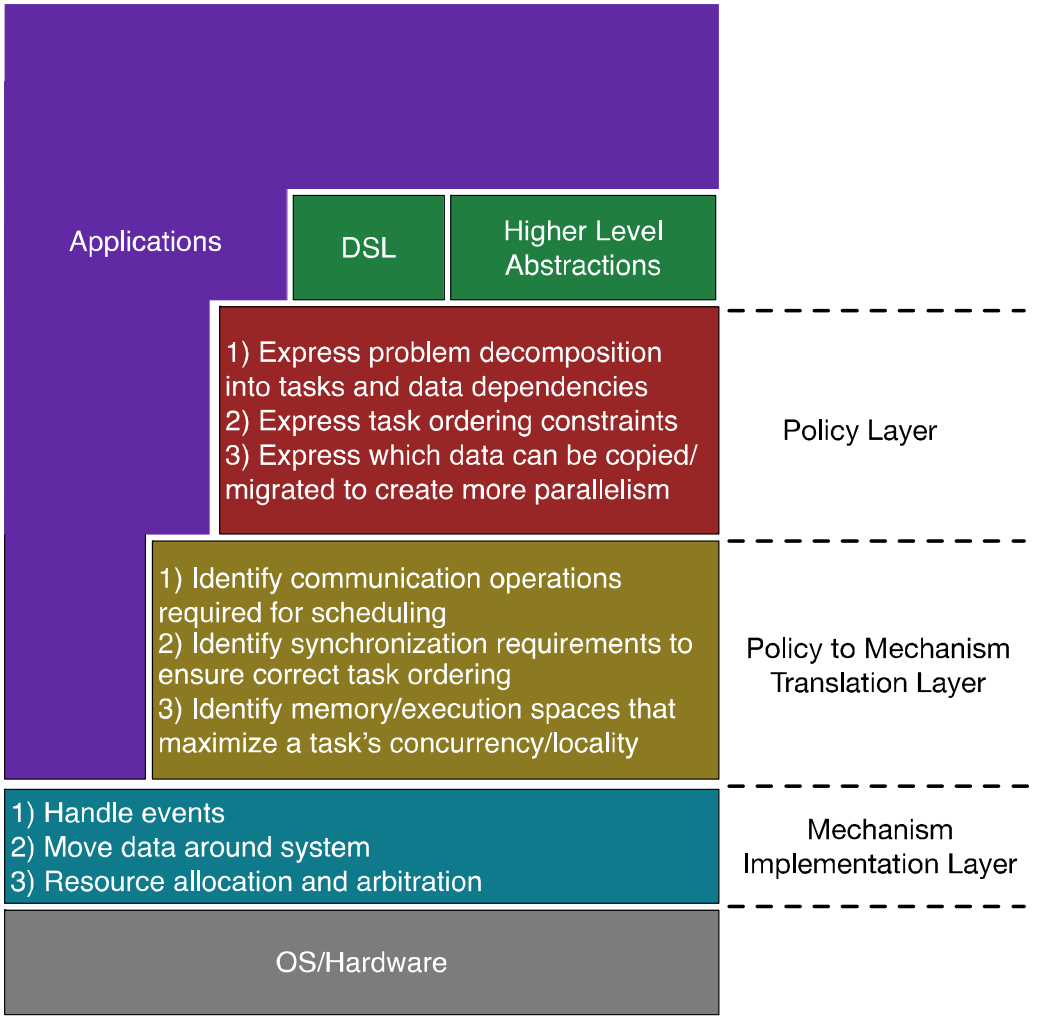
Mechanism: Implement correctness and performance requirements

# Expression of policy enables runtime freedom to make complex performance-oriented decisions

Sandia National Laboratories

## Design Intent:

- Applications specify policy
  - Enable rapid development of correct implementation
- Applications can specify mechanism
  - Enable improvement towards performant implementation

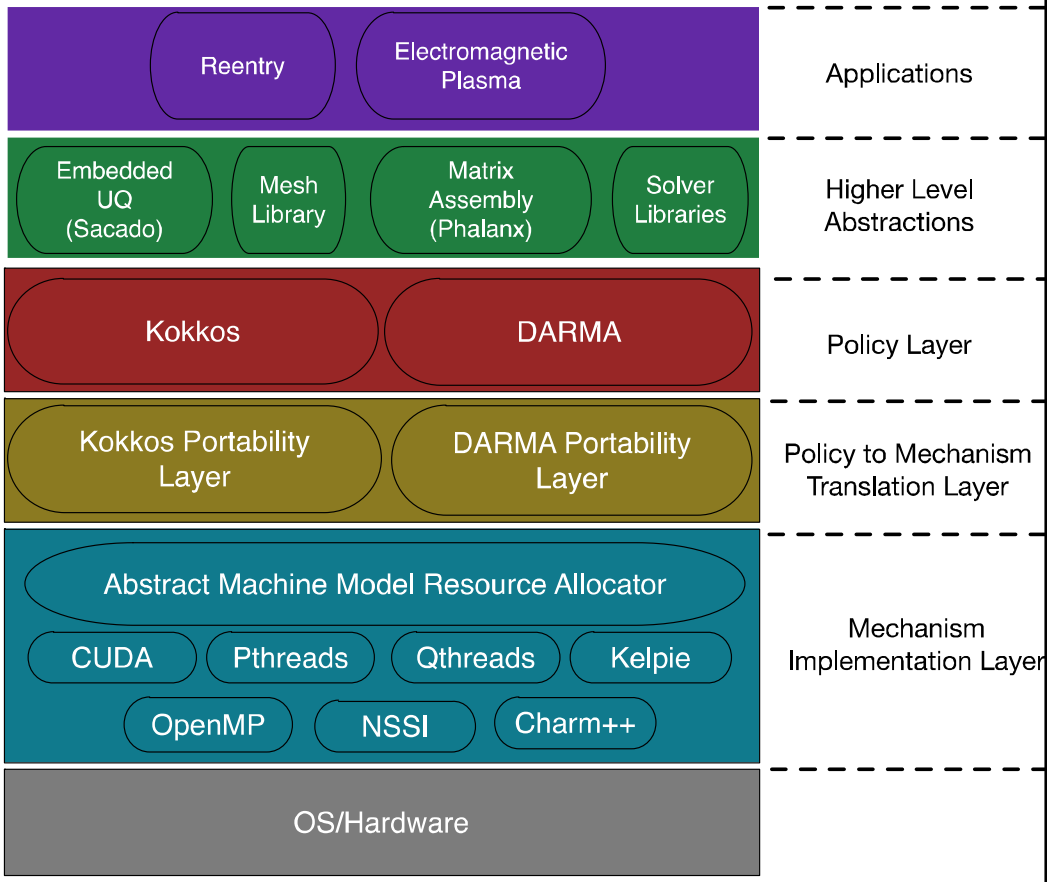


# The separation of policy and mechanism facilitates exploration of runtime design space

Sandia National Laboratories

- AMT software stack working group at Sandia
  - DARMA
  - Kokkos
  - Data Warehouse/Kelpie
  - Resource allocation and management
  - Qthreads
- Initial implementation of stack this year leveraging Charm++
- Working with community to explore alternative stack implementations
  - OCR, REALM

Sample Sandia Software Stack



# CHAPTER 3:
# PHILOSOPHY OF PROGRAMMING MODELS

# Keep simple things simple, keep manageable things manageable , make difficult things manageable

Sandia National Laboratories

- Simple/manageable
  - SPMD launch and initial problem decomposition/distribution
  - Collectives
  - Basic checkpoint/restart fault recovery supported
  - Application-specific data structures/layouts
- Difficult
  - Express/mix all forms of parallelism (data, pipeline, task)
  - Dynamic load balancing, work stealing
  - Data staging (software-managed cache)
  - Performance portability across execution spaces
  - Macro data-flow parallelism (parallelism within a task)

# The Ontology of DARMA: Axioms/assumptions of programming model derived from L2/co-design study

Sandia National Laboratories

- SPMD is the dominant parallelism
- There will too much compute (parallelism) available in the hardware for basic data parallelism to fill
- Extra asynchrony should not complicate reasoning about application correctness (intuitive semantics, debugging tools)
- The traditional MPI abstract machine model (uniform compute elements, flat memory spaces) will get further and further away from actual system architecture
- There exist many applications/algorithms with dynamic load balance, dynamic sparsity, or complex workflow coupling not yet implemented in MPI/OpenMP that need more productive programming model

# All parallel programming require four basic components

Sandia National Laboratories

- Problem decomposition
  - How to decompose data/work into tasks that *might* run in parallel
- Derive parallelism
  - Safe to run parallel tasks based on data disjointness, read/write permissions, or even use of atomics
  - Figure out which tasks *can* run in parallel
- Work distribution
  - How should tasks be distributed based on available parallelism and data relationships
- Data movement
  - Once tasks are assigned to compute units, data must be moved between parallel compute units

# Granularity of tasks is THE problem: Will parallelizing compilers ever succeed in distributed memory?

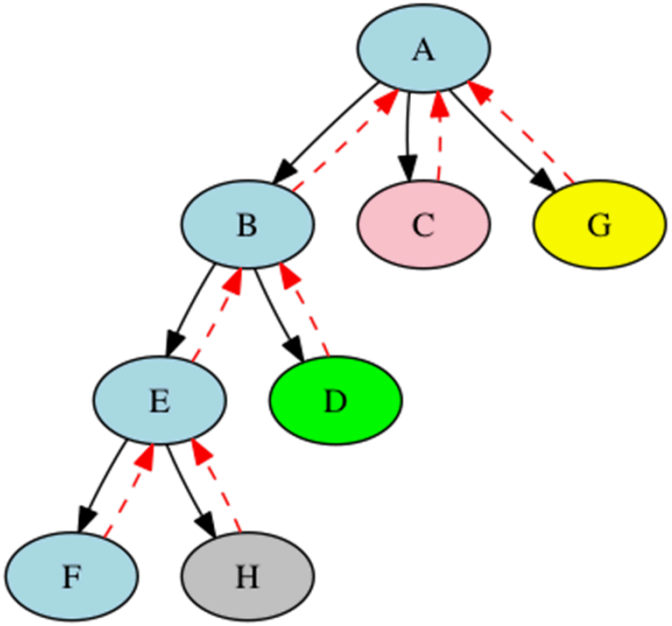
Sandia National Laboratories

- The correct granularity for the problem decomposition has to balance increased parallelism against increased overheads
- Compilers really good at deriving parallelism
  - Compilers understand read/write conflicts on registers
  - Can derive a lot of instruction-level parallelism
- Humans better at decomposing problems, worse at generating correct code
  - Compilers do not understand “big picture’’ of data partitioning
  - Difficult for humans to reason about and write explicitly parallel code
- Is there an optimal human/compiler hybrid?
  - Humans choose decomposition that compiler/runtime works with
  - Compilers/runtime use problem decomposition to generate *correct* and *performant* parallelism

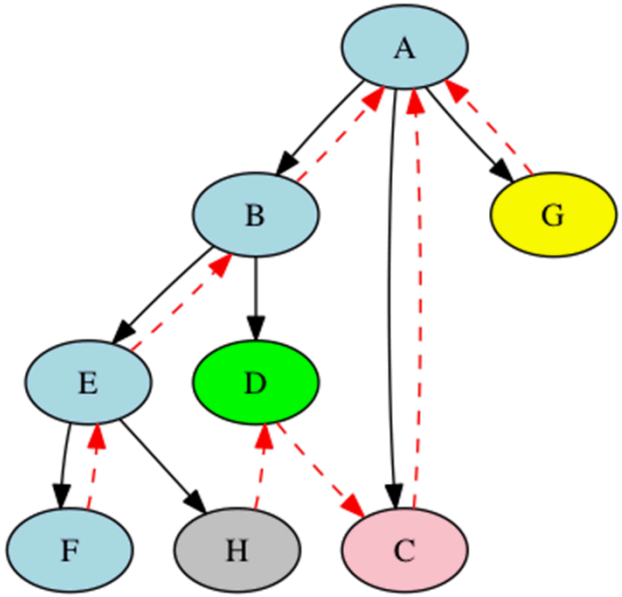
# Balancing the promise of sequential semantics against the peril of poor performance

Sandia National Laboratories

- ‘’Ideal’’ programming model must choose semantics that makes it possible to *verify* and *optimize* code
- Sequential semantics makes code correctness easier
- Mapping sequential semantics to parallel execution is easiest with *strict* tasks, which limits scalability

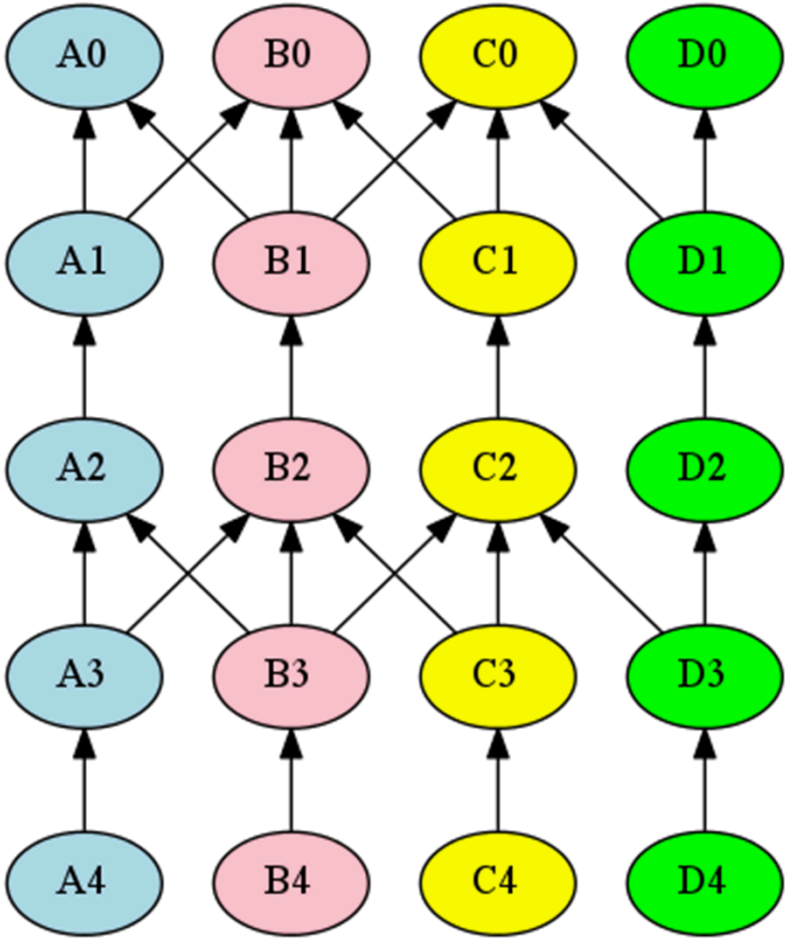



Strict fork-join

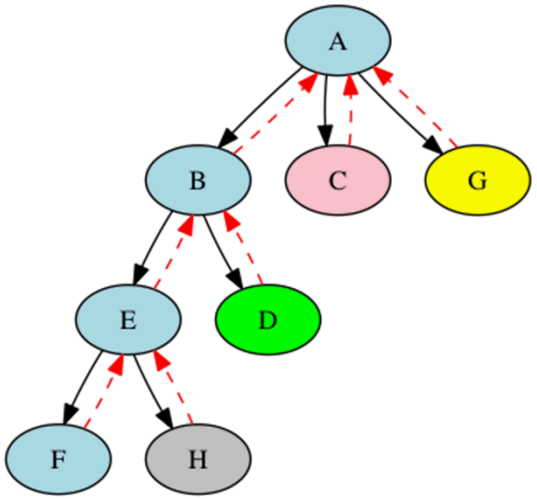



No strictness (allowed in MPI)

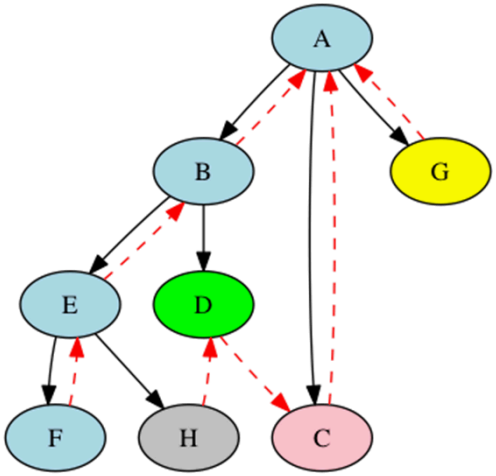
# SPMD is the nightmare scenario for sequential semantics




SPMD task graph




Strict fork-join




No strictness (allowed in MPI)

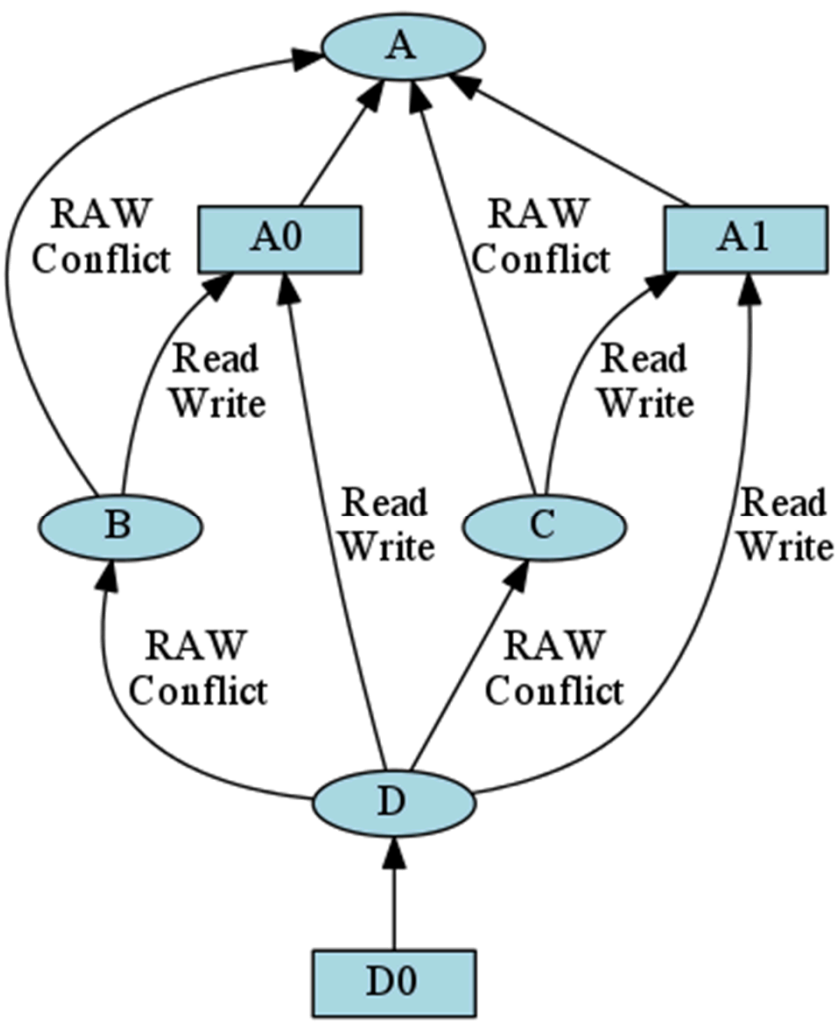
# Specific design choices try to mix best parts of humans with best parts of compilers/runtimes

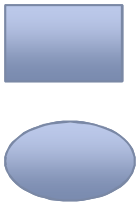
Sandia National Laboratories

- Humans are responsible for problem decomposition
- Humans are responsible for SPMD parallelism using explicitly parallel semantics
- Humans do not explicitly express task parallelism, rely on apparently sequential semantics
- Compilers/runtime are responsible for deriving on-node task parallelism from sequential semantics
- Compilers/runtime are responsible for deferring/re-ordering tasks to achieve communication/computation overlap

# CHAPTER 4:
# THE DETAILED TECHNICAL STORY

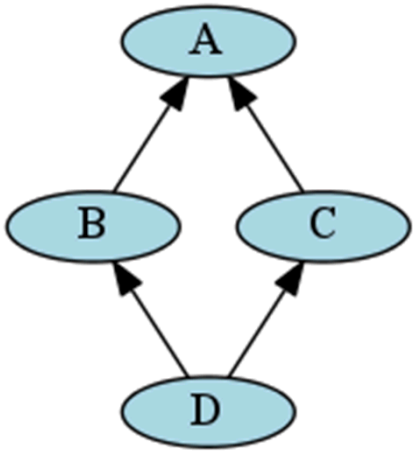
# Use read/write conflicts to derive task parallelism from sequential semantics and build operations/task DAG




RAW = Read-after-write
WAR = Write-after-read

=



# Lambda capture/copy constructors of Handle objects allows runtime to derive parallelism

Sandia National Laboratories

```
typedef AccessHandle<int> IntPtr;
IntPtr a;
IntPtr b;

deferred_work(reads(a),writes(b), [=]{
  *b += *a;
})

deferred_work(reads(b),reads(a), [=]{
  printf("added a=%d to get b=%d\n", *a, *b);
})

deferred_work(writes(a), [=]{
  *a = 5;
})
```

# Lambda capture/copy constructors of Handle objects allows runtime to derive parallelism

Sandia National Laboratories

```
typedef AccessHandle<int> IntPtr;
IntPtr a;
IntPtr b;

deferred_work(reads(a),writes(b), [=]{
  *b += *a;
})

deferred_work(reads(b),reads(a), [=]{
  printf("added a=%d to get b=%d\n", *a, *b);
})

deferred_work(writes(a), [=]{
  *a = 5;
})
```

Read-after-write dependence on b

Write-after-read dependence on a

# Copy constructors are mechanism for building arbitrary operations graph directly in C++11

Sandia National Laboratories

```
AccessHandle::AccessHandle(AccessHandle& prev)
{
 darma::runtime* rt = prev.runtime();
 schedule_state_t state = prev.scheduleState();
 switch (state)
 {
  case Read_Read:
    rt->addReadAfterReadEdge(...)
    break;
  case Read_Write:
    rt->addReadAfterWriteEdge(...)
    break;
  case Write_Read:
    rt->addWriteAfterReadEdge(...)
    break;
```

# Explicit parallel launch to produce independent execution streams: "coordinating" sequential processes

Sandia National Laboratories

```
MPI_Send(...) -> publish(key={…}, readers={…})
MPI_Recv(...) ->  fetch(key={…}), read_access(key={…})
```

```
int main(int argc, char** argv)
{
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(comm, &rank);
  MPI_Comm_size(comm, &size);
  if (rank == 0){
    MPI_Send(...)
  } else {
    MPI_Recv(...)
  }
}
```

```
int darma_main()
{
  int rank = darma_spmd_rank();
  int size = darma_spmd_size();
  AccessHandle<int> data;
  if (rank == 0){
    data.publish(...);
  } else {
    data.fetch(...);
  }
}
```

# Explicit parallel launch to produce independent execution streams: “coordinating” sequential processes

Sandia National Laboratories

```
MPI_Send(...) -> publish(key={…}, readers={…})
MPI_Recv(...) ->  fetch(key={…}), read_access(key={…})
```

- Data in distributed memory not allowed to be ``anonymous’’
- No message-ordering semantics
- All data must be explicitly published with unique name in tuple space (key-value store)
- Program expresses all *true dependencies (RAW)* via unique keys
- Extra concept of “readership” and “versions” allows efficient data reuse, zero-copy transfers, in-place updates

# Extended coordination semantics for explicit parallelism (usually SPMD)

Sandia National Laboratories

```
Process 0:
DoublePtr residual = initial_access<double>("res");
DoublePtr vec0 = read_access<vector<double>>("block", 0);
DoublePtr vec1 = read_access<vector<double>>("block", 1);
deferred_work([=]{
  ddot(residual.get(), vec0.get(), vec1.get(), vec0.size());
});


Process 1:
int nelems = 100;
DoublePtr vec = initial_access<vector<double>>("block", 0);
deferred_work([=]{
  auto& v = vec.get();
  v.resize(nelems);
  for (int i=0; i < nelems; ++i){
    v[i] = rand();
  }
})
vec.publish()
```

# Extended coordination semantics enables explicit SPMD parallelism

Sandia National Laboratories

```
typedef AccessHandle<double> DoublePtr;
typedef AccessHandle<vector<double>> DoubleArray;
```

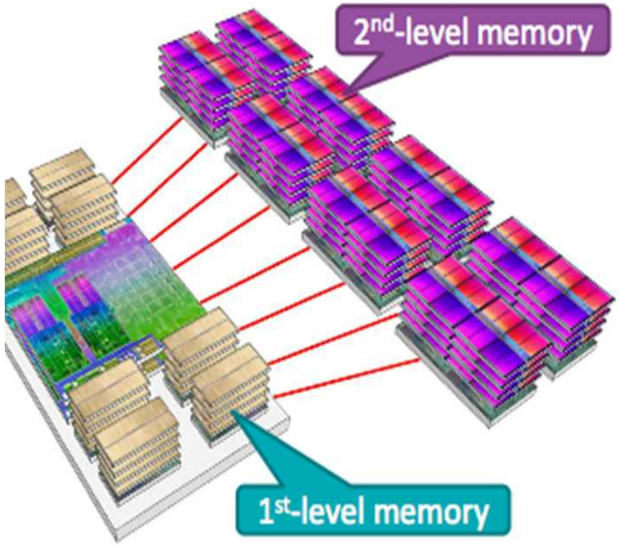
1. AccessHandle is a lightweight wrapper (aka smart pointer) that can be freely copied with minimal cost
2. AccessHandle is unique to a task – each task gets its own copy of the handle with different read/write privileges, memory space, any other relevant metadata
3. Multiple AccessHandle instances can point to same data. Each handle gets a unique identifier (key) for its data block.

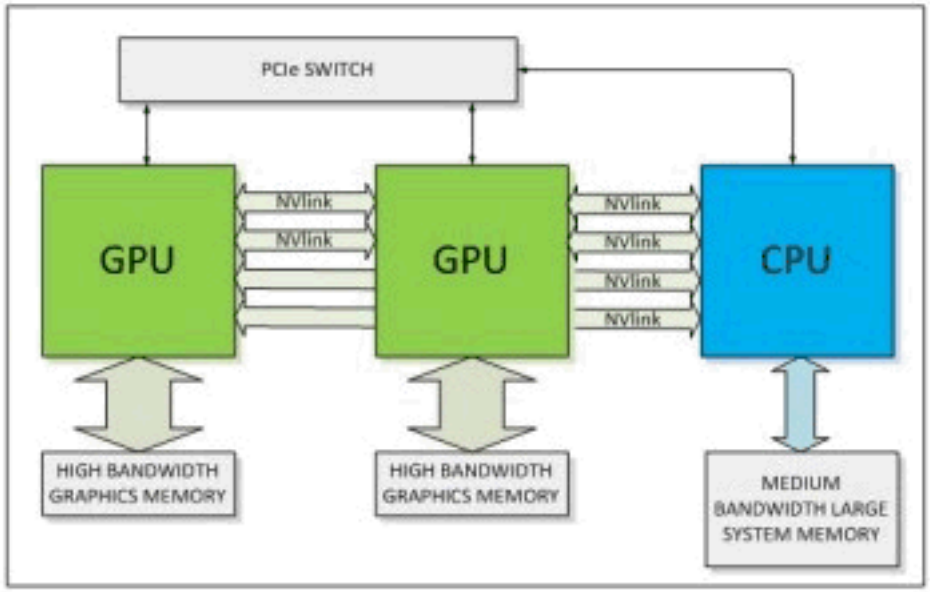
```
  auto& v = vec.get();
  v.resize(nelems);
  for (int i=0; i < nelems; ++i){
    v[i] = rand();
  }
})
vec.publish()
```

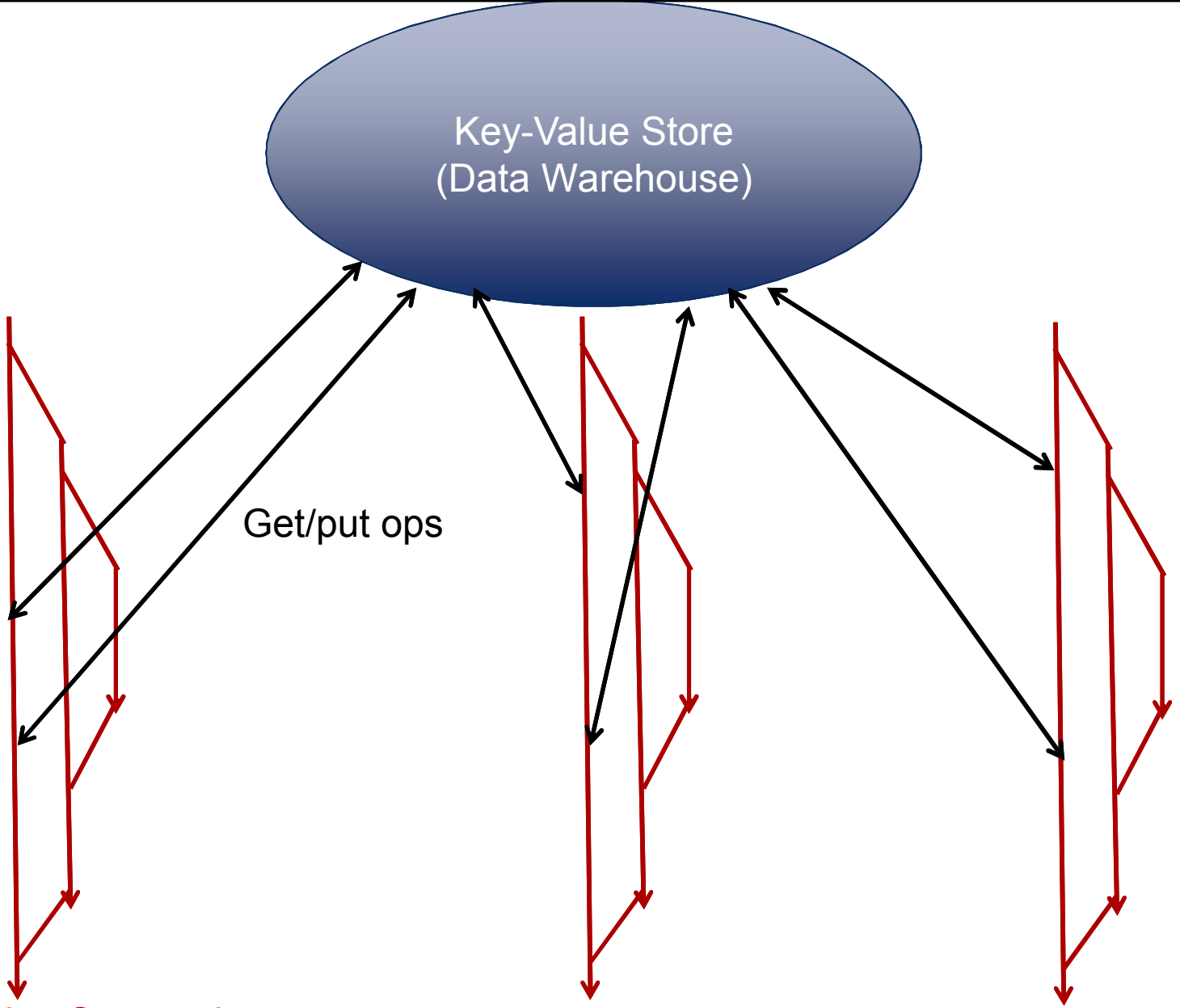
# Copy constructors are mechanism for building arbitrary operations graph directly in C++11

Sandia National Laboratories

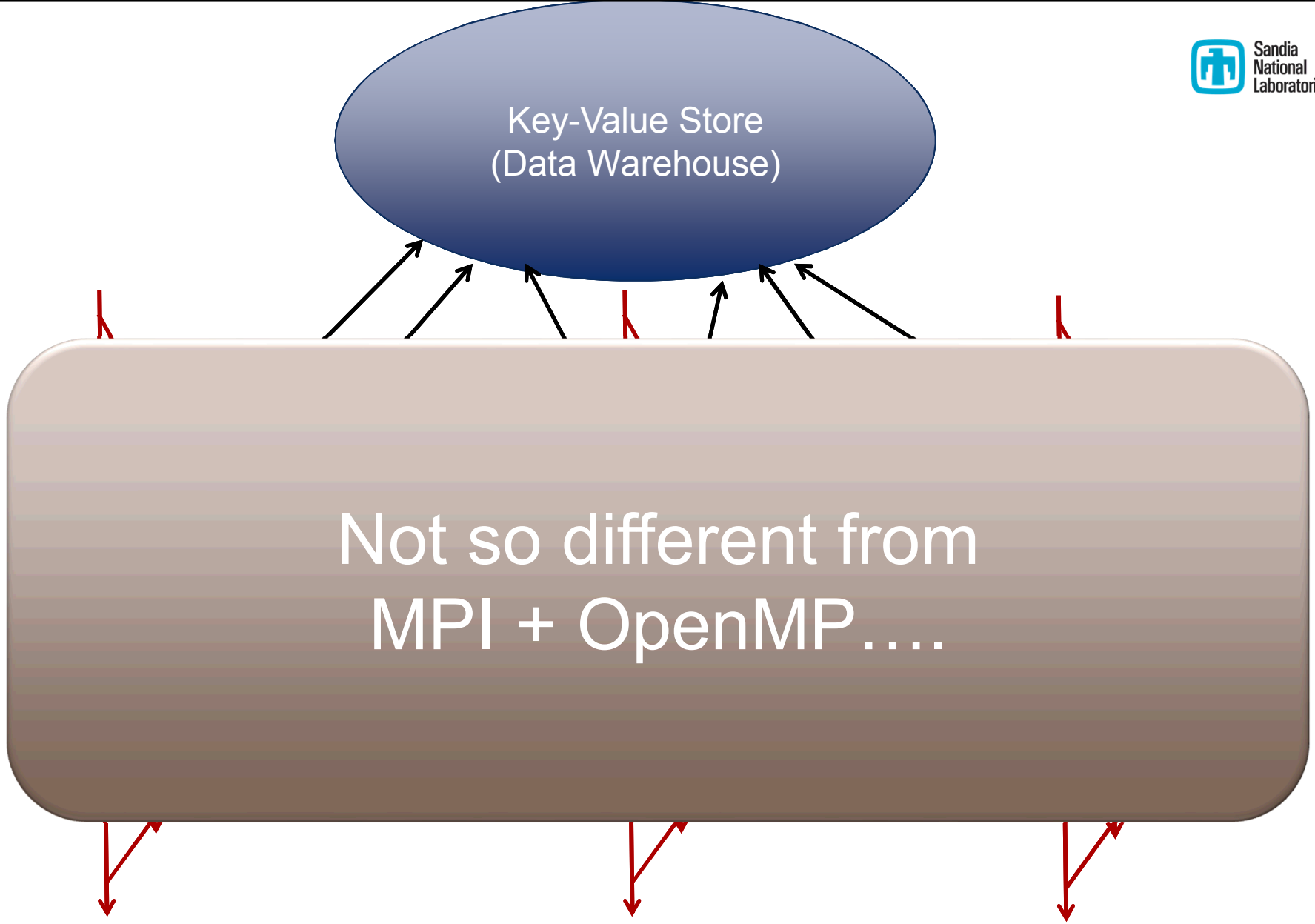
```
AccessHandle::AccessHandle(AccessHandle& prev)
{
 darma::runtime* rt = prev.runtime();
 ...
 if (prev.memorySpace() != this->memorySpace()){
  rt->addDataStagingEdge(...);
 }
```






Execution Stream that forks task parallelism


Key-Value Store
(Data Warehouse)
Not so different from
MPI + OpenMP….

# Quick summary of programming model design

Sandia National Laboratories

- Application controls initial problem decomposition/distribution through coordination
  - Explicit parallelism at user-level
- Extra task/pipeline parallelism added through read/write qualifiers and task annotations
  - Implicit parallelism compliant with sequential semantics
- Embedded in C++11 – no compiler support needed
- Global memory space (tuple space/key-value store) instead of a global address space

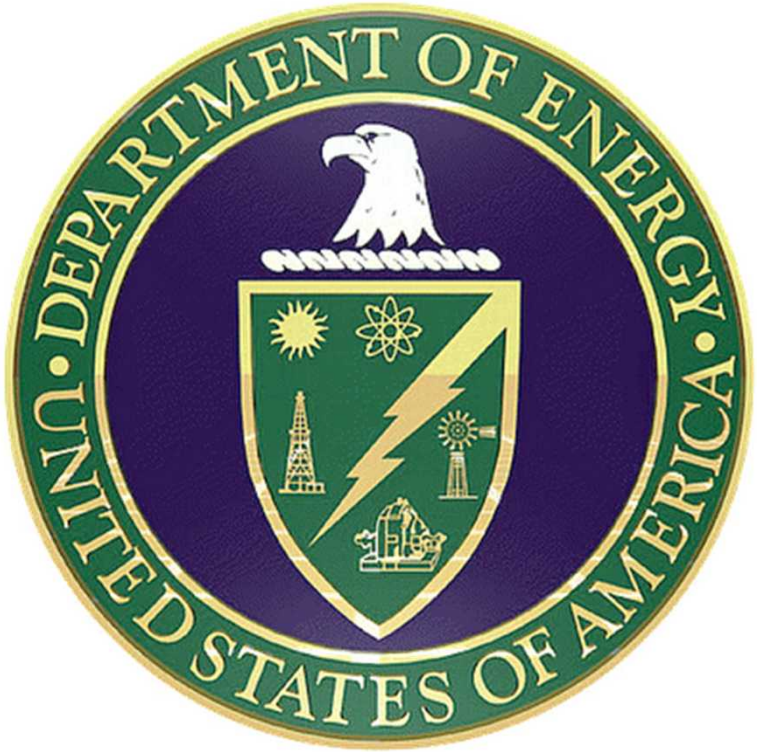
# Future work

Sandia National Laboratories

- Data staging – use Lambda capture/copy operations to start data movement to GPU/high-bandwidth memory if data not resident
- Work stealing algorithms
  - Data exists in global memory space, not global address space
  - Data can be safely migrated to other address spaces
- Elastic (parallel) tasks
  - DARMA not designed to exploit fine-grained data parallelism
  - Still need CUDA/Kokkos/OpenMP for multi-threading within a task
- Macro data-flow scheduling
  - Task graph can run ahead of execution
  - Run scheduling heuristics on tasks graph to choose best schedule

# Acknowledgments

This work was supported by the U.S. Department of Energy (DOE) National Nuclear Security Administration (NNSA) Advanced Simulation and Computing program and the DOE Office of Advanced Scientific Computing Research. SNL is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the DOE NNSA under contract DE-AC04-94AL85000.




National Nuclear Security Administration

Sandia National Laboratories

Sandia
National
Laboratories

*Exceptional service in the national interest*