SAND2017-1430C

Status of the DARMA Asynchronous Many Task Abstraction Layer

Janine C. Bennett, Jonathan Lifflander, David S. Hollman, Jeremiah Wilke, Hemanth Kolla, Aram Markosyan, Nicole Slattengren, Robert L. Clay (PM)

Lawrence Livermore National Laboratory

Feb 8, 2017

What is DARMA?

DARMA is a C++ abstraction layer
for asynchronous many-task (AMT) runtimes

Goals:

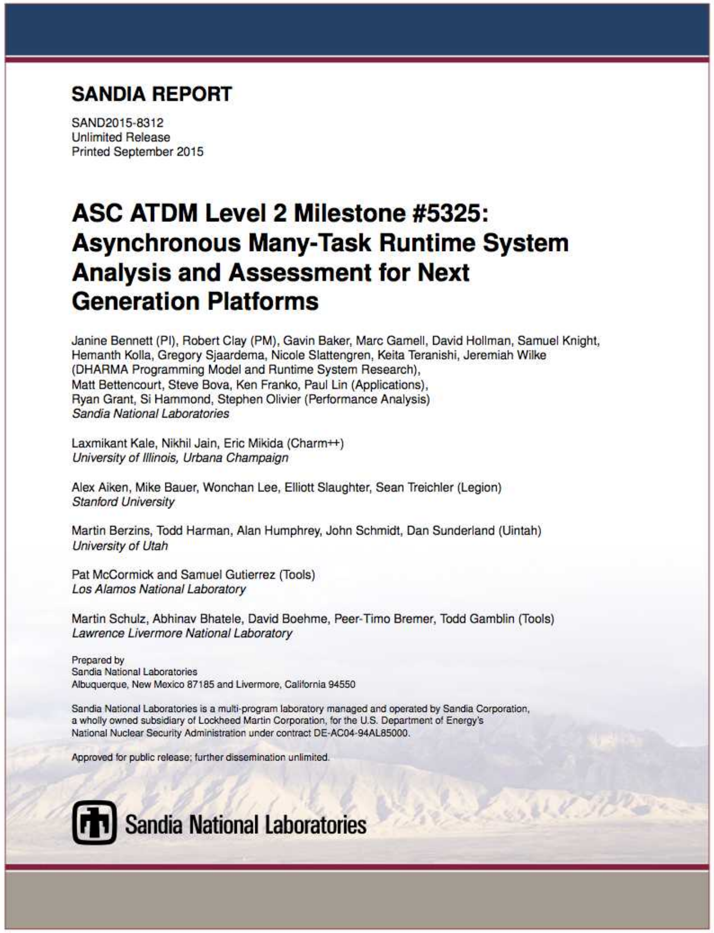
1. Enable Sandia ATDM application scientists to explore a variety of underlying runtime system technologies
2. Facilitate the expression of coarse-grained tasking

2015 study to assess leading AMT runtimes led to DARMA

Aim: inform Sandia's technical roadmap for next generation codes

- Broad survey of many AMT runtime systems
- Deep dive on Charm++, Legion, Uintah

- ***Programmability:*** Does this runtime enable efficient expression of ATDM workloads?
- ***Performance:*** How performant is this runtime for our workloads on current platforms and how well suited is this runtime to address future architecture challenges?
- ***Mutability:*** What is the ease of adopting this runtime and modifying it to suit our code 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.

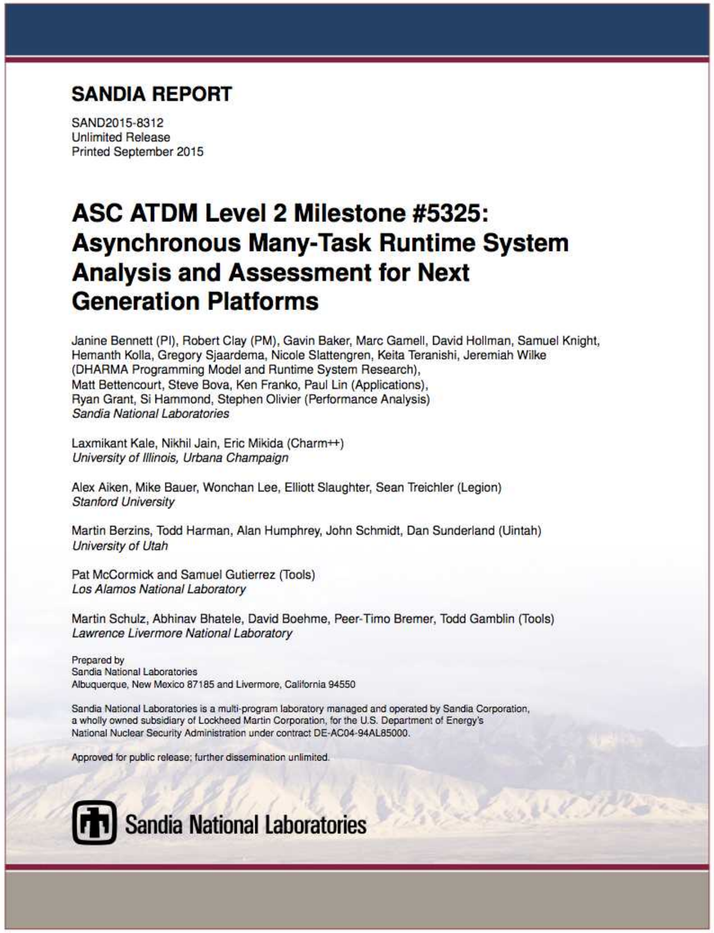
Sandia National Laboratories

2015 study to assess leading AMT runtimes led to DARMA

Aim: inform Sandia's technical roadmap for next generation codes

- ***Conclusions***
  - AMT systems show great promise
  - Gaps in requirements for Sandia applications
  - No common user-level APIs
  - Need for best practices and standards

- ***Survey recommendations led to DARMA***
  - C++ abstraction layer for AMT runtimes
  - Requirements driven by Sandia ATDM applications
  - A single user-level API
  - Support multiple AMT runtimes to begin identification of best practices



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

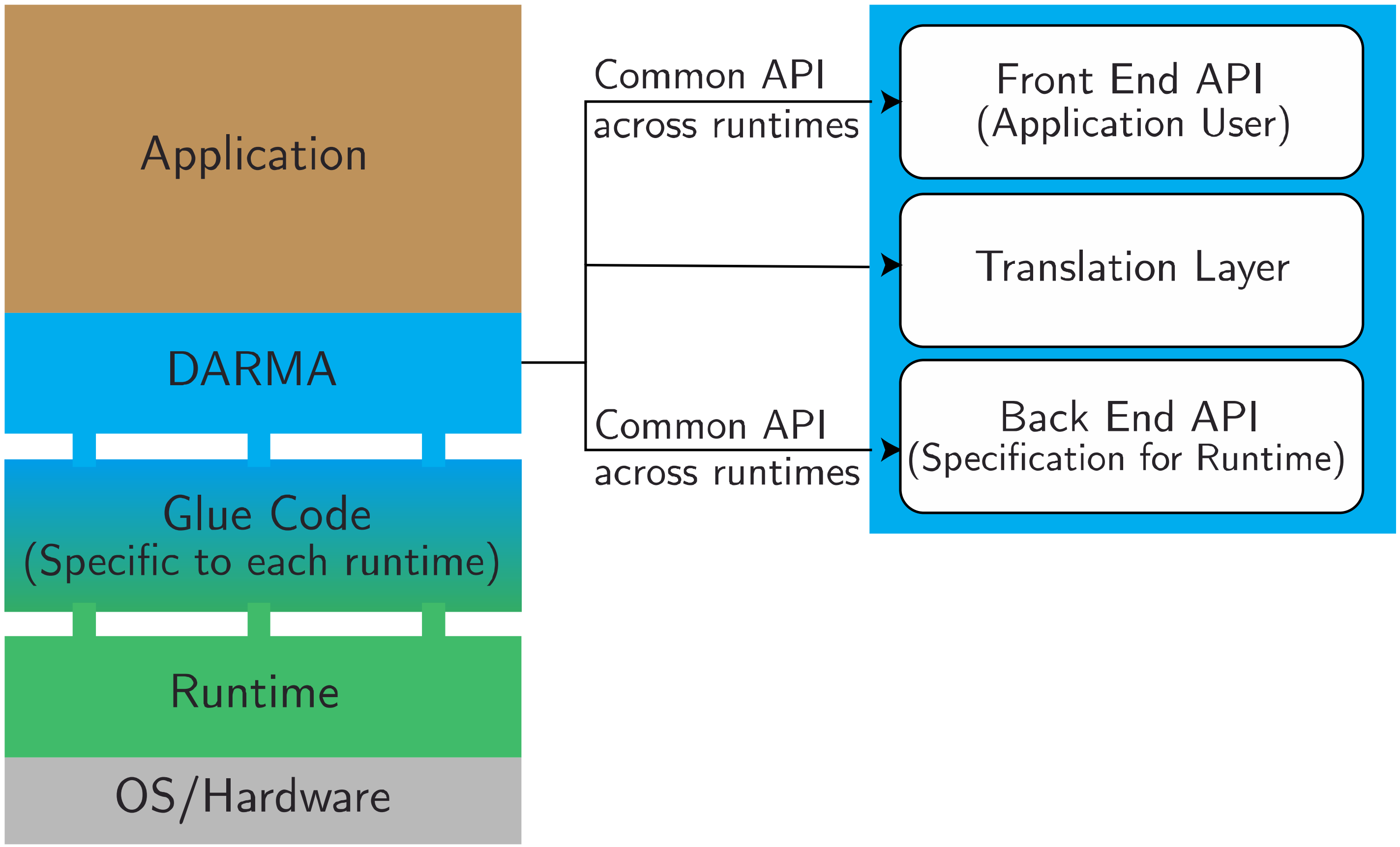
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

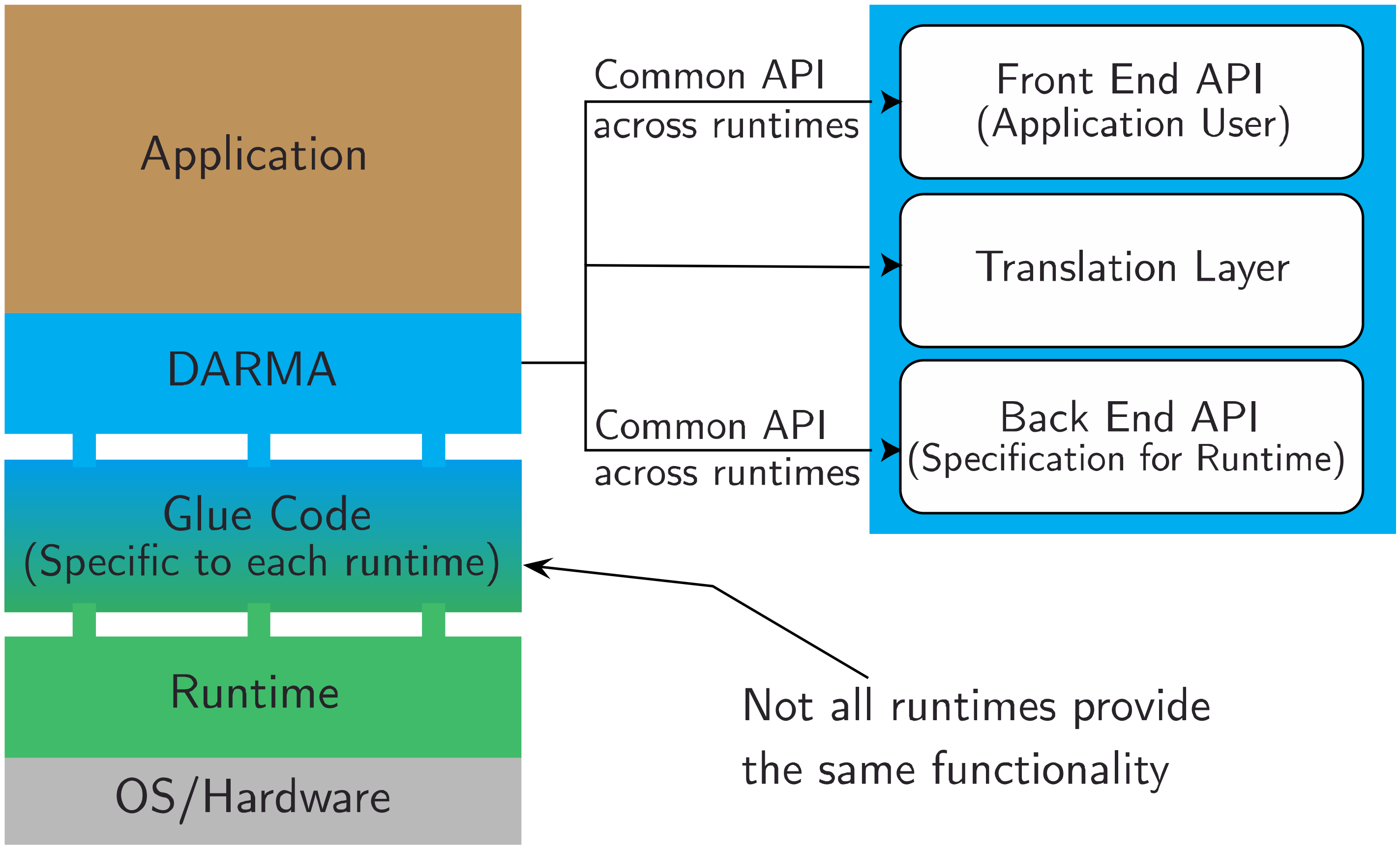
Goal 1: Enabling exploration of a variety of runtime system technologies via a unified API

Application developers use a single API for expressing coarse-grained tasks



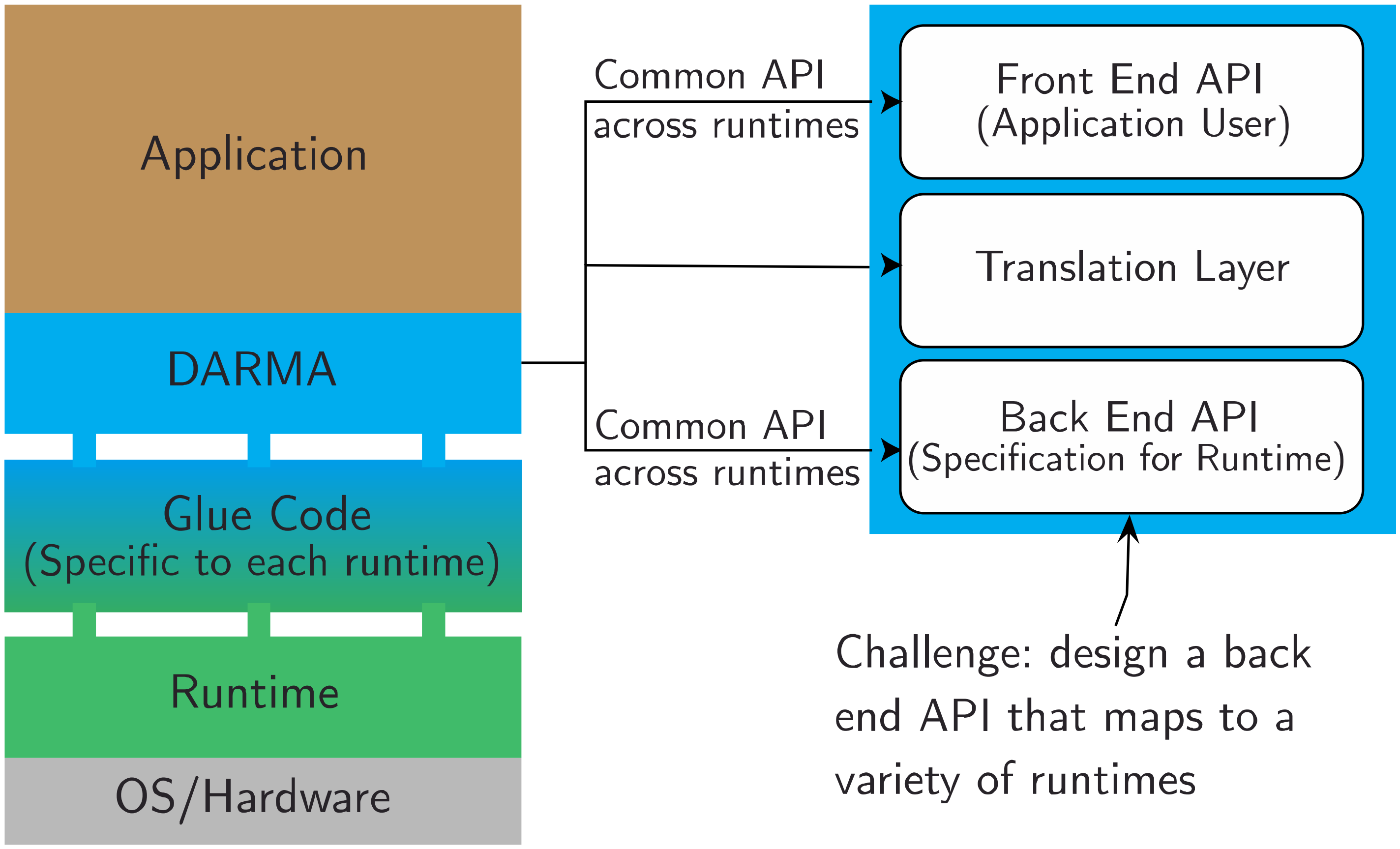
Goal 1) Enabling exploration of a variety of runtime system technologies via a unified API

Application code is translated into a series of backend API calls to an AMT runtime



Goal 1) Enabling exploration of a variety of runtime system technologies via a unified API

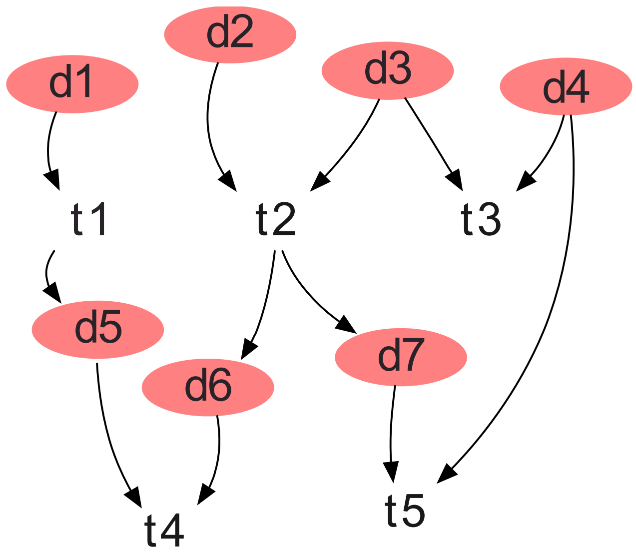
Application code is translated into a series of backend API calls to an AMT runtime



Goal 1) Enabling exploration of a variety of runtime system technologies via a unified API

Considerations when developing a backend API that maps to a variety of runtimes

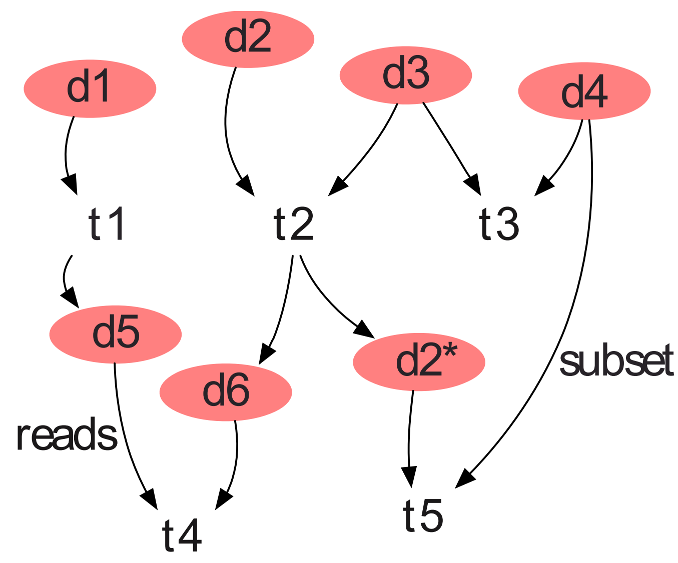
- AMT runtimes often operate with a directed acyclic graph (DAG)
  - Captures relationships between application data and inter-dependent tasks



Goal 1) Enabling exploration of a variety of runtime system technologies via a unified API

Considerations when developing a backend API that maps to a variety of runtimes

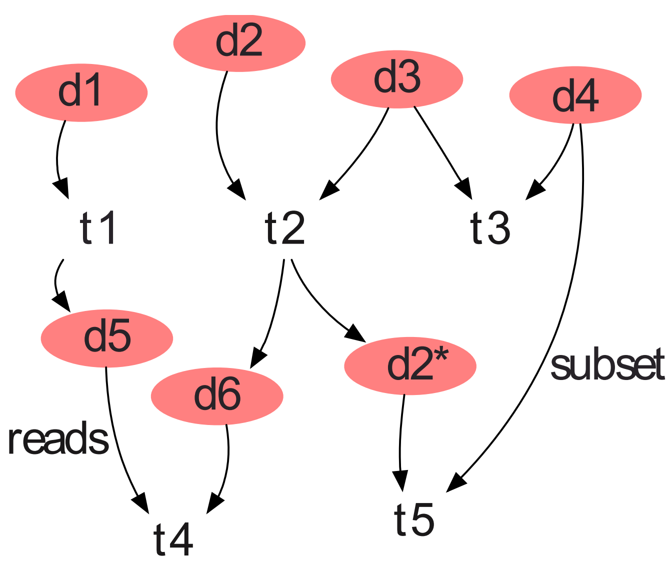
- AMT runtimes often operate with a directed acyclic graph (DAG)
  - Captures relationships between application data and inter-dependent tasks
- DAGs can be annotated to capture additional information
  - Tasks' read/write usage of data
  - Task needs a subset of data



Goal 1) Enabling exploration of a variety of runtime system technologies via a unified API

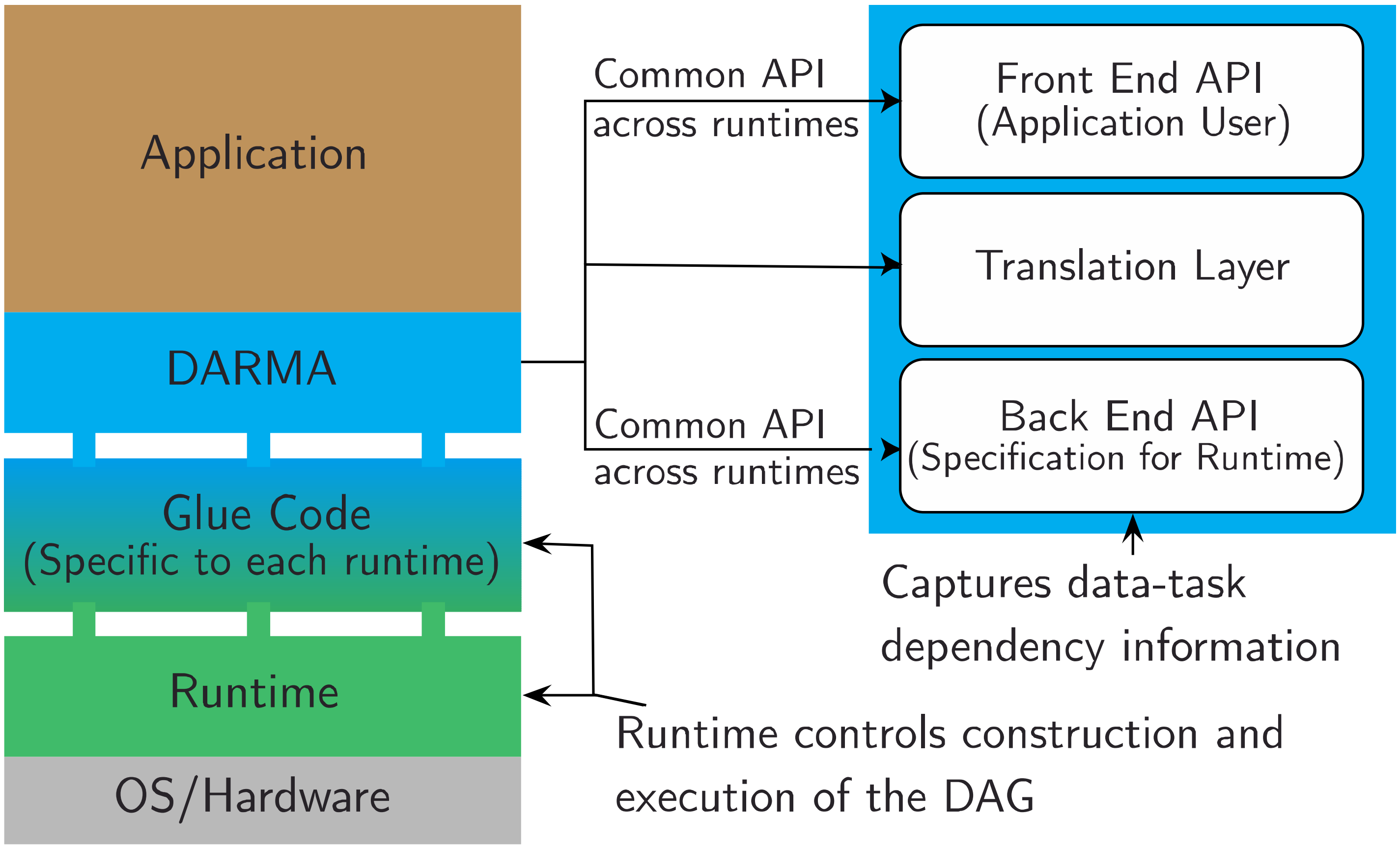
Considerations when developing a backend API that maps to a variety of runtimes

- AMT runtimes often operate with a directed acyclic graph (DAG)
  - Captures relationships between application data and inter-dependent tasks
- DAGs can be annotated to capture additional information
  - Tasks' read/write usage of data
  - Task needs a subset of data
- Additional information enables runtime to reason more completely about
  - When and where to execute a task
  - Whether to load balance
- Existing runtimes leverage DAGs with varying degrees of annotation



Goal 1) Enabling exploration of a variety of runtime system technologies via a unified API

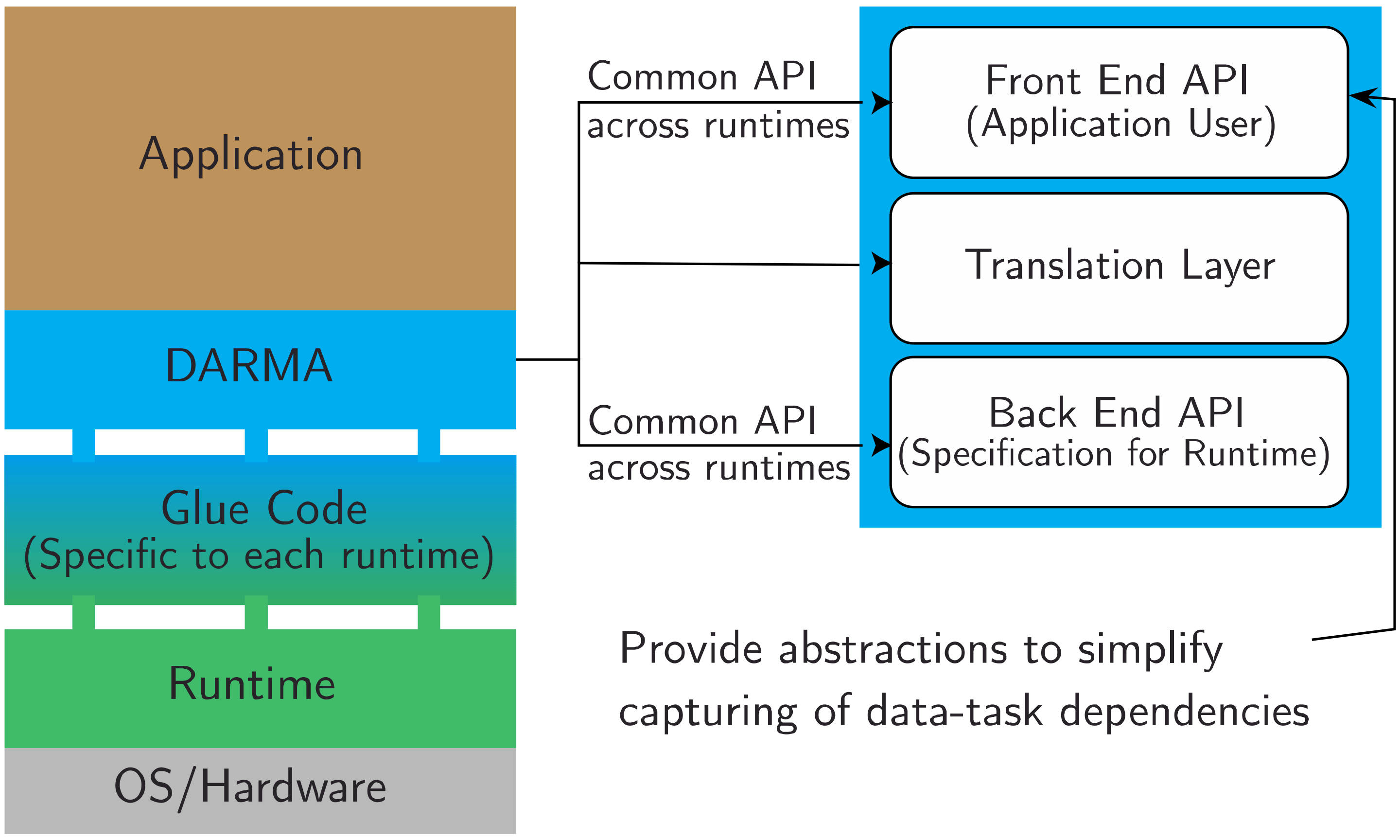
DARMA passes data-task dependency information to the runtime which builds and executes the DAG



Goal 1) Enabling exploration of a variety of runtime system technologies via a unified API

Goal 2: Facilitating the expression of coarse-grained tasking

DARMA front end abstractions are co-designed with Sandia ATDM application scientists



Goal 2) Facilitating the expression of coarse-grained tasking

DARMA introduces a set of abstractions that enable local and distributed tasking

- Asynchronous smart pointers wrap user data
  - darma::AccessHandle<T>
  - darma::AccessHandleCollection<T>

- DARMA tasks
  - darma::create_work
  - darma::create_concurrent_work

Goal 2) Facilitating the expression of coarse-grained tasking

DARMA introduces a set of abstractions that enable local and distributed tasking

- Asynchronous smart pointers wrap user data
  - darma::AccessHandle<T>
  - darma::AccessHandleCollection<T>

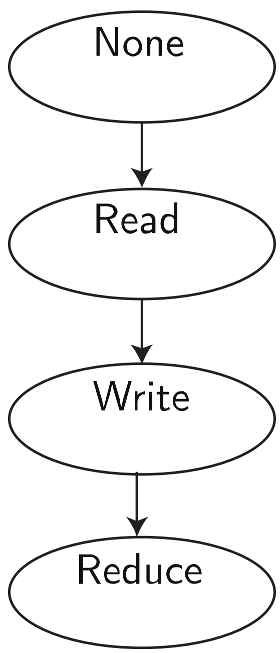
- DARMA tasks
  - darma::create_work
  - darma::create_concurrent_work

Goal 2) Facilitating the expression of coarse-grained tasking

Asynchronous Smart Pointers enable extraction of concurrency in a data-race-free manner

darma::AccessHandle<T> enforces **sequential semantics**: it uses the order in which data is accessed in your program and how it is accessed (read/write/etc.) to automatically extract concurrency

Permission Level



Permission Type

Scheduling

A task with scheduling permission can create deferred tasks that can access the data at the specified permission level.

Immediate

A task with immediate permission can dereference the AccessHandle<T> and use it according to the permission level.

Goal 2) Facilitating the expression of coarse-grained tasking

A task is a block of deferred work that executes sequentially

Tasks can be recursively nested within each other to generate more subtasks

C++ Lambdas

```
darma::create_work(
  [=]{
     /*do some work*/
  }
);
```

This is the C++ 11 syntax for writing an anonymous function that captures variables by value.

C++ Functors

```
struct MyFun {
  void operator()(...) {
     /* do some work */
  }
};

darma::create_work<MyFun>(...)
```

Functors are for larger blocks of code that may be reused and migrated by the backend to another memory space.

Goal 2) Facilitating the expression of coarse-grained tasking

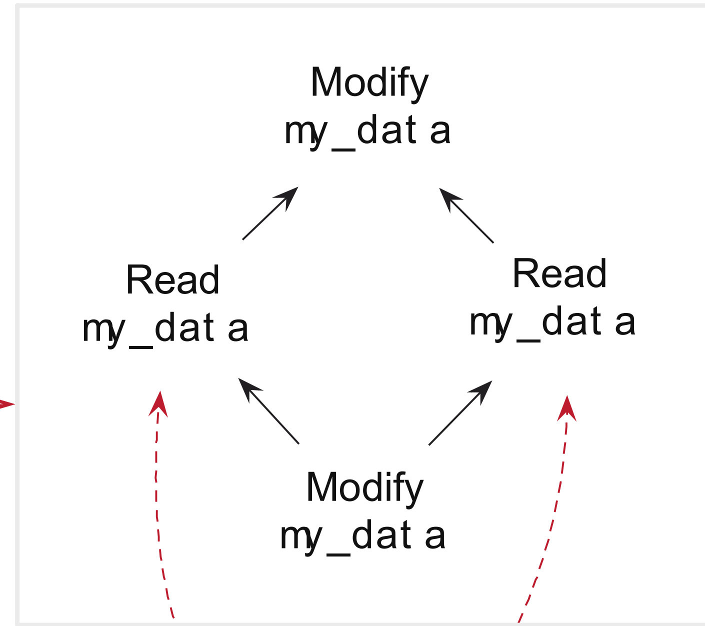
Example: Putting tasks and data together

Example Program

```
AccessHandle<int> my_data;

darma::create_work([=]{
  my_data.set_value(29);
});

darma::create_work(
  reads(my_data), [=]{
    cout << my_data.get_value();
  }
);

darma::create_work(
  reads(my_data), [=]{
    cout << my_data.get_value();
  }
);

darma::create_work([=]{
  my_data.set_value(31);
});
```

Sequential Semantics

DAG (Directed Acyclic Graph)



These two tasks are concurrent and can be run in parallel by a DARMA backend!

Goal 2) Facilitating the expression of coarse-grained tasking

Sandia ATDM applications drive requirements and developers play active role in informing front end API

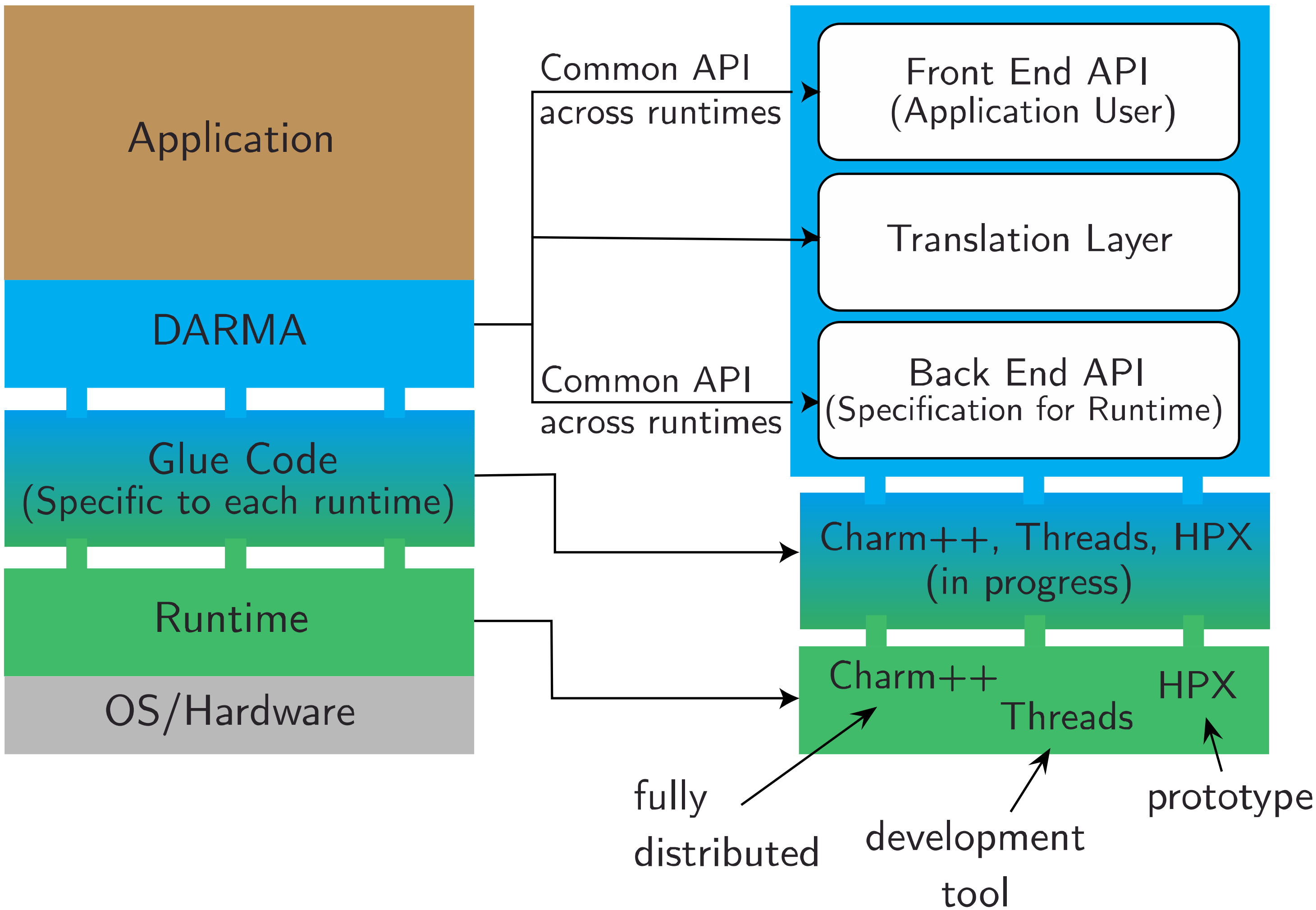
- Application feature requests
  - Sequential semantics
  - MPI interoperability
  - Node-level performance portability layer interoperability (Kokkos)
  - Collectives
  - Runtime-enabled load-balancing schemes

- API has evolved based on application developer usage and feedback

Goal 2) Facilitating the expression of coarse-grained tasking

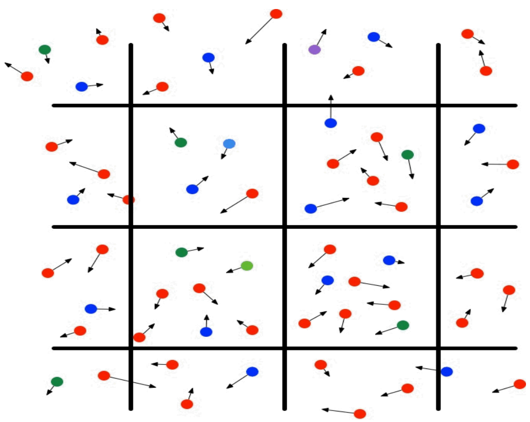
Using DARMA to inform Sandia's technical roadmap

Currently there are three back ends in various stages of development

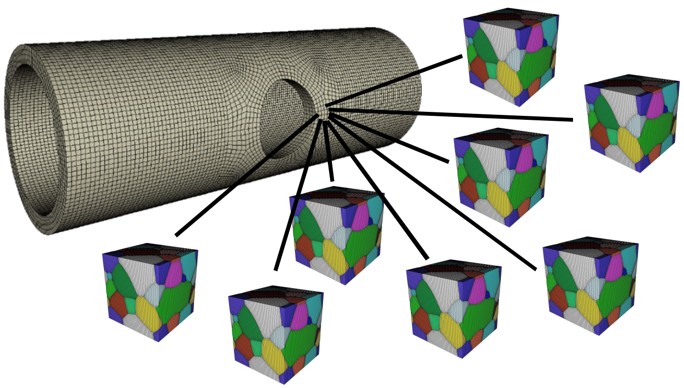


Using DARMA to inform Sandia's ATDM technical roadmap

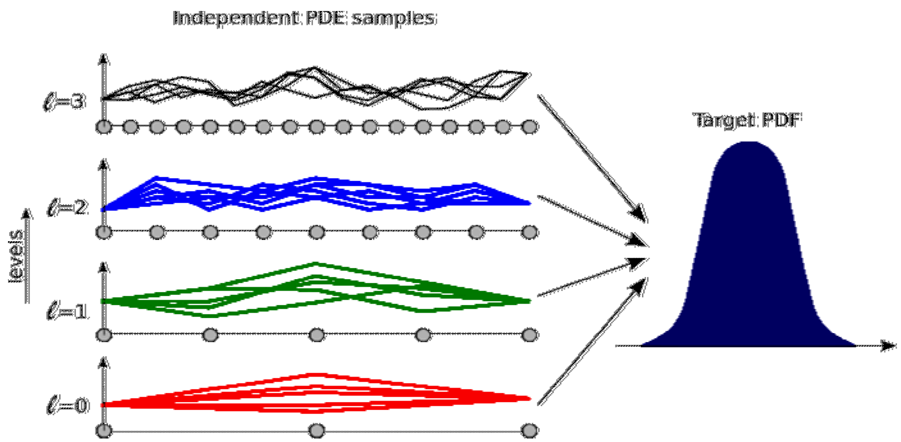
2017 study: Explore programmability and performance of the DARMA approach in the context of ATDM codes



Electromagnetic Plasma Particle-in-cell Kernels



Multiscale Proxy



Multi Level Monte Carlo Uncertainty Quantification Proxy

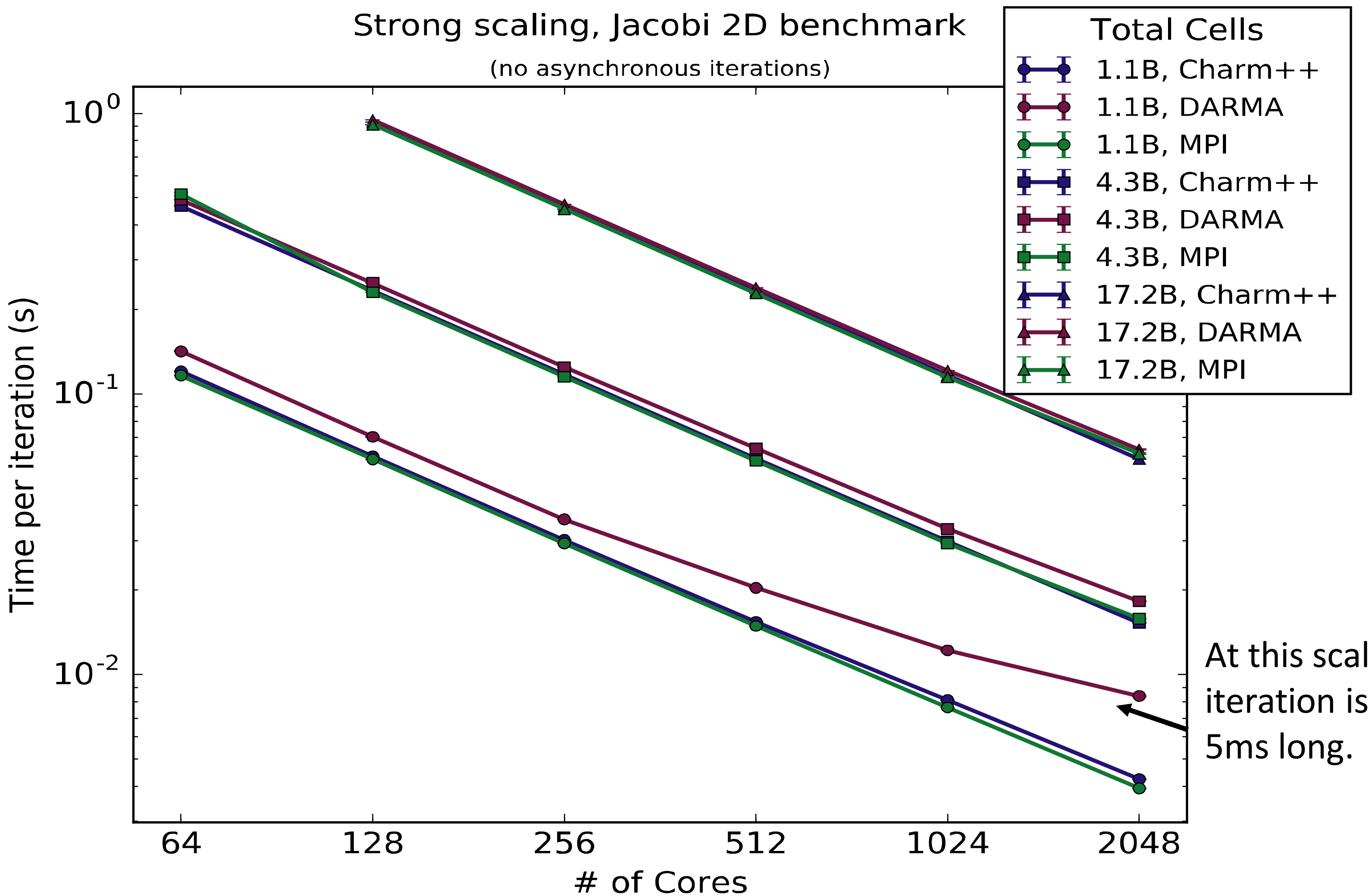
Using DARMA to inform Sandia's ATDM technical roadmap

Performance benchmarks explore how AMT runtime overheads can be masked by several factors

- Kernels and proxies will evolve throughout 2017
- In the meantime simple benchmarks enable studies on
  - Task granularity
  - Overlap of communication and computation
  - Runtime-managed load balancing
- Tests performed on Mutrino
  - Haswell partition of Trinity testbed

- These early results are being used to identify and address bottlenecks in preparation for studies with kernels/proxies

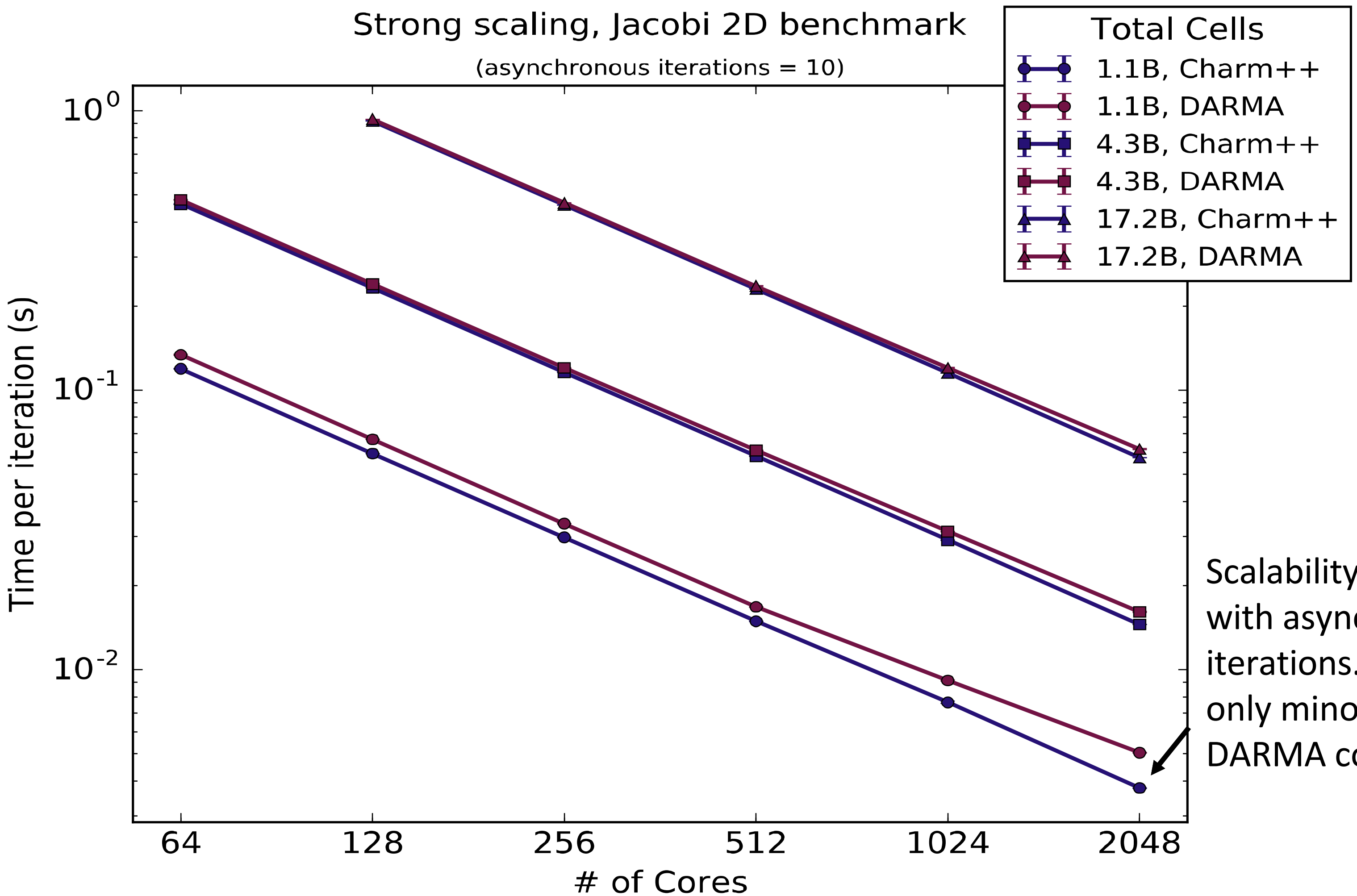
Stencil benchmark is not latency tolerant and highlights runtime overheads when task-granularity is small



At this scale, each iteration is less than 5ms long.

Using DARMA to inform Sandia's ATDM technical roadmap

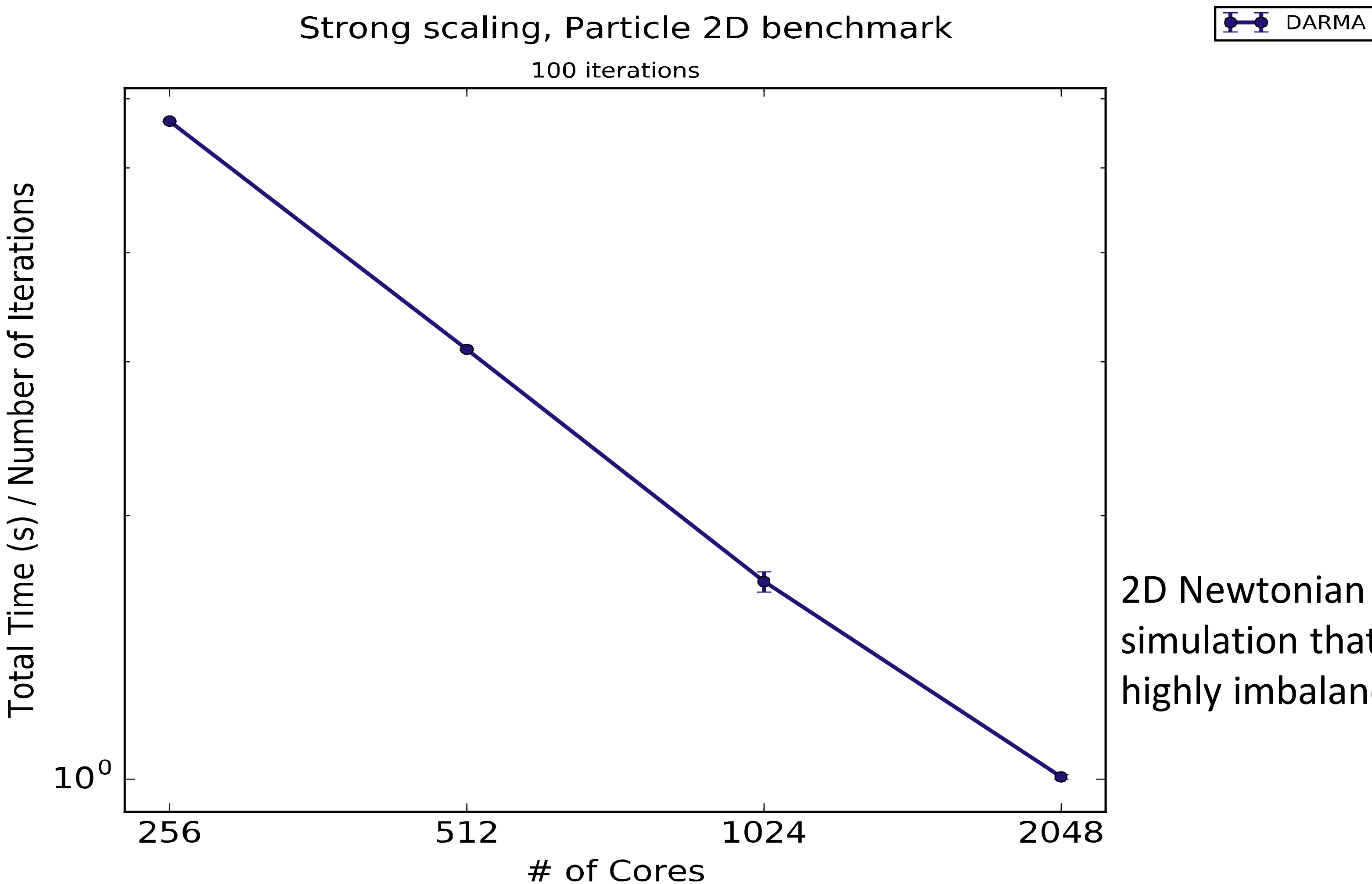
Increased asynchrony in application enables runtime to overlap communication and computation



Scalability improves with asynchronous iterations. Requires only minor changes to DARMA code.

Using DARMA to inform Sandia's ATDM technical roadmap

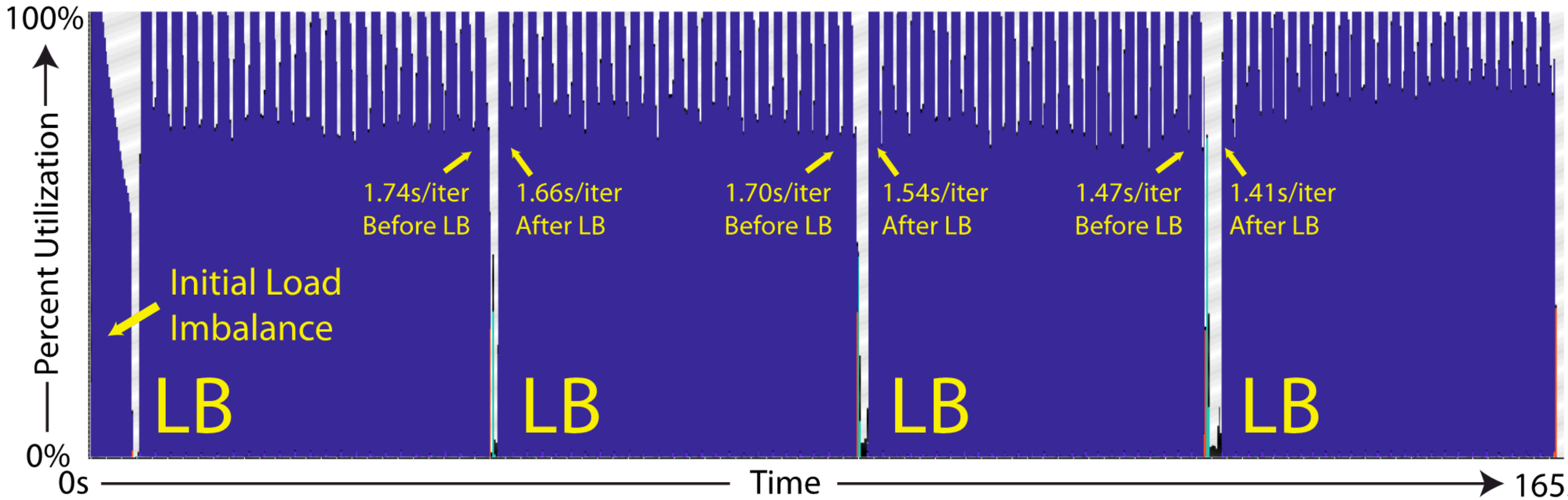
DARMA's programming model enables runtime-managed, measurement-based load balancing



2D Newtonian particle simulation that starts highly imbalanced.

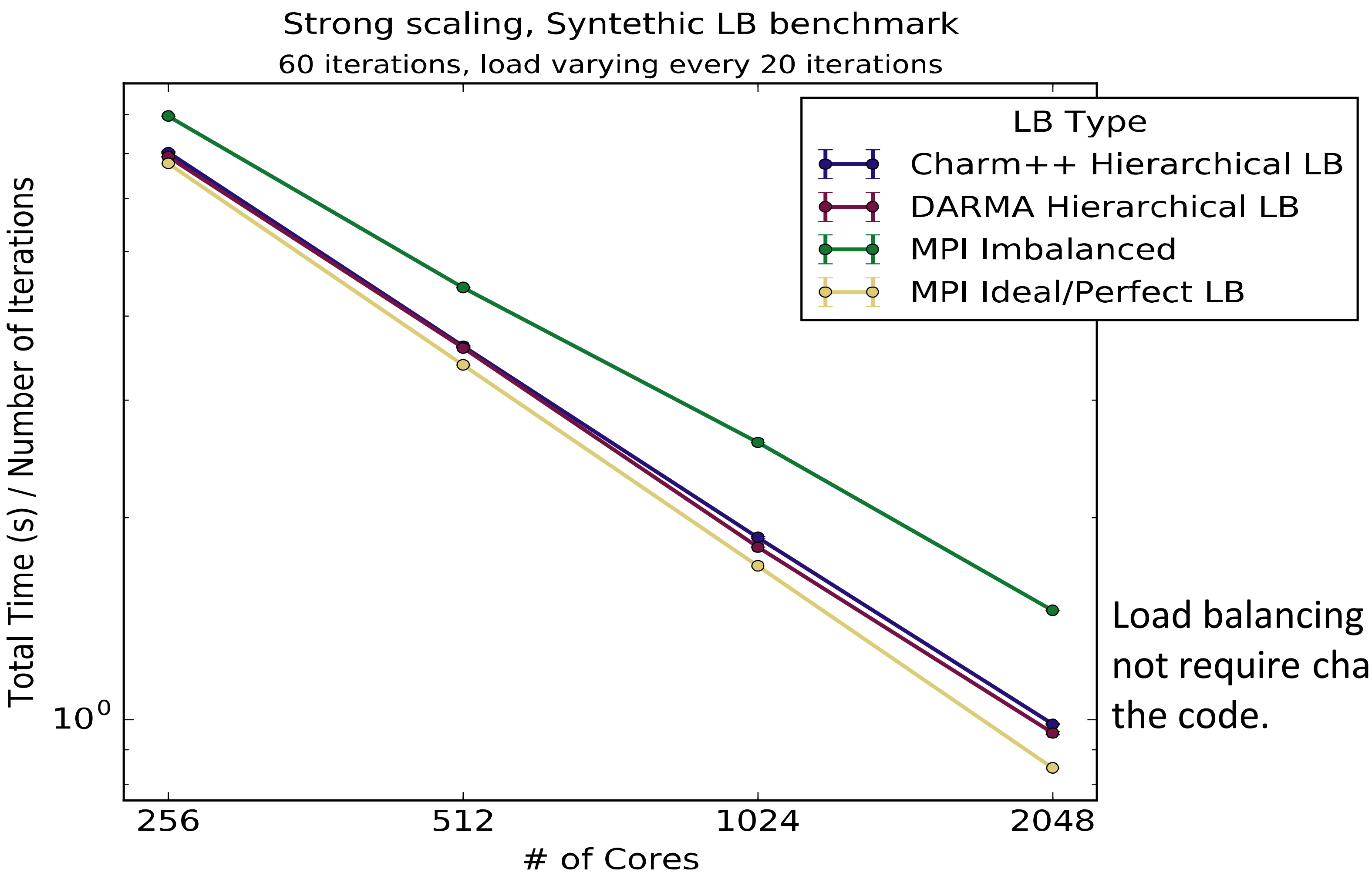
Using DARMA to inform Sandia's ATDM technical roadmap

DARMA's programming model enables runtime-managed, measurement-based load balancing



The load balancer incrementally runs as particles migrate and the work distribution changes.

DARMA's programming model enables runtime-managed, measurement-based load balancing



Load balancing does not require changes to the code.

Using DARMA to inform Sandia's ATDM technical roadmap

Summary: DARMA seeks to accelerate discovery of best practices

- Application developers
  - Use a unified interface to explore different runtime system technologies
  - Directly inform DARMA's user-level API via co-design requirements/feedback
- System software developers
  - Acquire a synthesized set of requirements via the backend specification
  - Directly inform backend specification via co-design feedback
  - Can experiment with proxy applications written in DARMA
- Sandia ATDM is using DARMA to inform its technology roadmap in the context of AMT runtime systems

Backup Slides

Smart pointer collections can be mapped across memory spaces in a scalable manner

AccessHandleCollection<T, R> is an extension to AccessHandle<T> that expresses a collection of data

Every element in the collection contains a `vector<double>`

```
AccessHandleCollection<vector<double>, Range1D> mycol =
  darma::initial_access_collection(
    index_range = Range1D(10)
  );
```

`Range1D` is a potentially user-defined (or domain-specific) ***index range***, a C++ object that describes the extents of the collection along with providing a corresponding index class for accessing an element.

mycol

Each indexed element is an `AccessHandle<vector<double>>`

Goal 2) Facilitating the expression of coarse-grained tasking

Tasks can be grouped into collections that make concurrent forward progress together

Task collections are a scalable abstraction to efficiently launch communicating tasks across large-scale distributed systems

```
create_concurrent_work<MyFun>(
  index_range = Range1D(5)
);
```

This call to `create_concurrent_work` launches a set of tasks, the size of which is specified by an index range, `Range1D`, that is passed as an argument.

```
struct MyFun {
  void operator()(Index1D i) {
     int me = i.value;
     /* do some work */
  }
};
```

Each element in the task collection is passed an `Index1D` within the range, used by the programmer to express communication patterns across elements in the collection.

Goal 2) Facilitating the expression of coarse-grained tasking

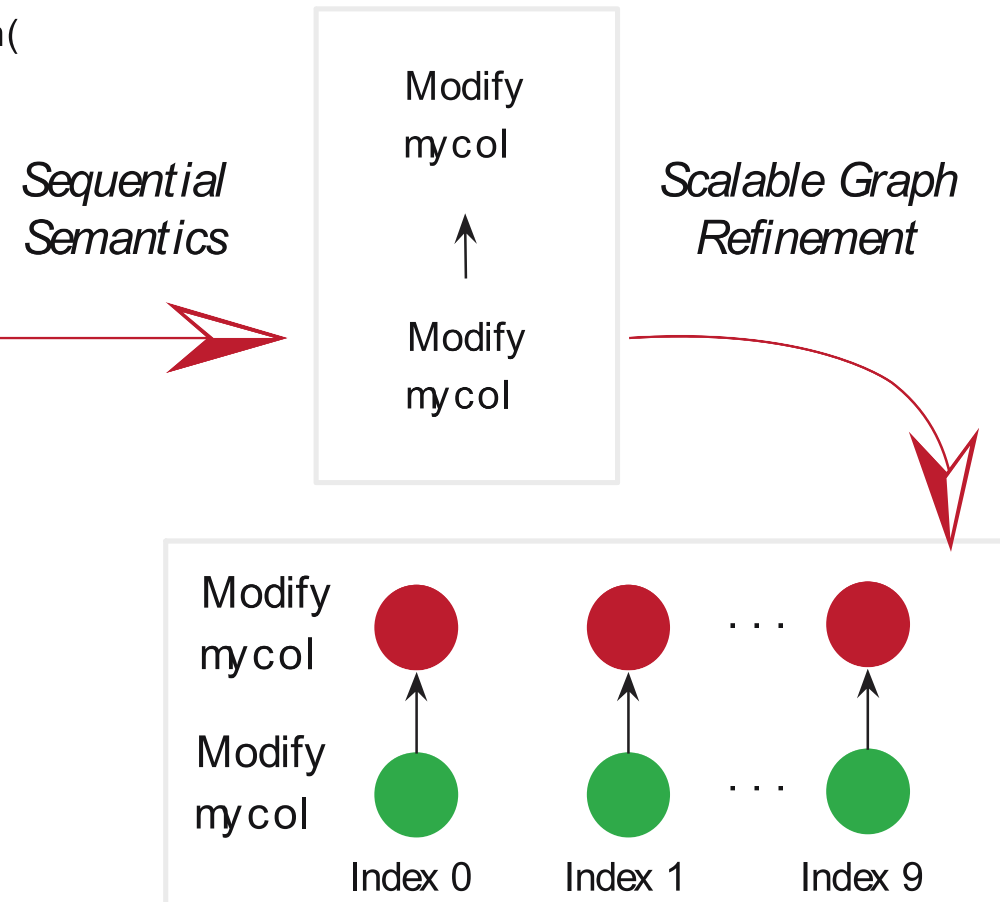
Putting task collections and data collections together

Example Program

```
auto mycol = initial_access_collection(
  index_range = Range1D(10)
);

create_concurrent_work<MyFun>(
  mycol, index_range = Range1D(10)
);

create_concurrent_work<MyFun>(
  mycol, index_range = Range1D(10)
);
```

A mapping must exist between the data index ranges and task index range. In this case, since the three ranges are identical in size and type, a one-to-one ***identity map*** is automatically applied.

Generated DAG



Goal 2) Facilitating the expression of coarse-grained tasking

Tasks in different execution streams can communicate via publish/fetch semantics

Sandia National Laboratories

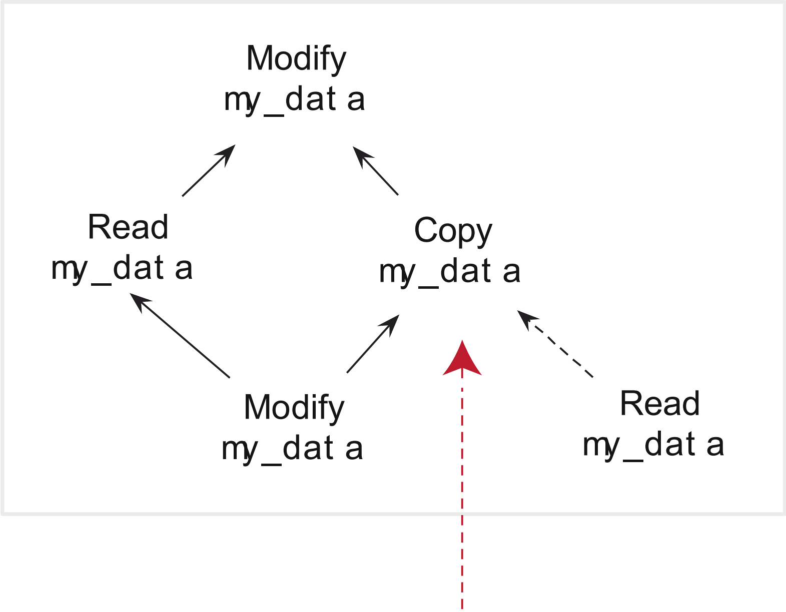
Execution Stream A

```
AccessHandle<int> my_data =
   initial_access<int>("my_key");

darma::create_work([=]{
   my_data.set_value(29);
});

my_data.publish(version="a");

darma::create_work([=]{
   my_data.set_value(31);
});
```

Execution Stream B

```
AccessHandle<int> other_data =
   read_access("my_key", version="a");

darma::create_work([=]{
   cout << other_data.get_value();
});

other_data = nullptr;
```

Potential DAG 1



If the read_access is on another node it might be send across the network.

Goal 2) Facilitating the expression of coarse-grained tasking

Tasks in different execution streams can communicate via publish/fetch semantics

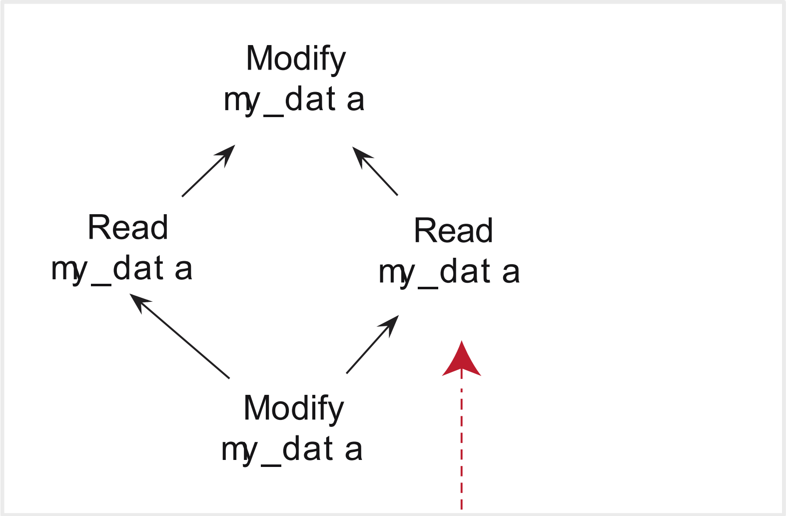
Execution Stream A

```
AccessHandle<int> my_data =
   initial_access<int>("my_key");

darma::create_work([=]{
   my_data.set_value(29);
});

my_data.publish(version="a");

darma::create_work([=]{
   my_data.set_value(31);
});
```

Execution Stream B

```
AccessHandle<int> other_data =
   read_access("my_key",  version="a");

darma::create_work([=]{
   cout  << other_data.get_value();
});

other_data = nullptr;
```

Potential DAG 2



If the read_access is on the same node a back end runtime can generate an alternative DAG without the transfer.

Goal 2) Facilitating the expression of coarse-grained tasking

Tasks in different execution streams can communicate via publish/fetch semantics

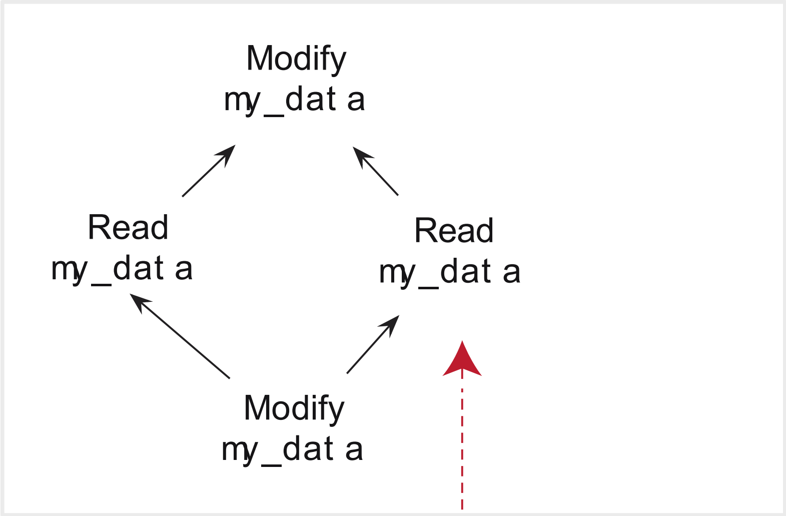
Execution Stream A

```
AccessHandle<int> my_data =
   initial_access<int>("my_key");

darma::create_work([=]{
   my_data.set_value(29);
});

my_data.publish(version="a");

darma::create_work([=]{
   my_data.set_value(31);
});
```

Execution Stream B

```
AccessHandle<int> other_data =
   read_access("my_key", version="a");

darma::create_work([=]{
   cout << other_data.get_value();
});

other_data = nullptr;
```

Potential DAG 2



If the `read_access` is on the same node a back end runtime can generate an alternative DAG without the transfer.

Goal 2) Facilitating the expression of coarse-grained tasking

A mapping between data and task collections determines access permissions between tasks and data

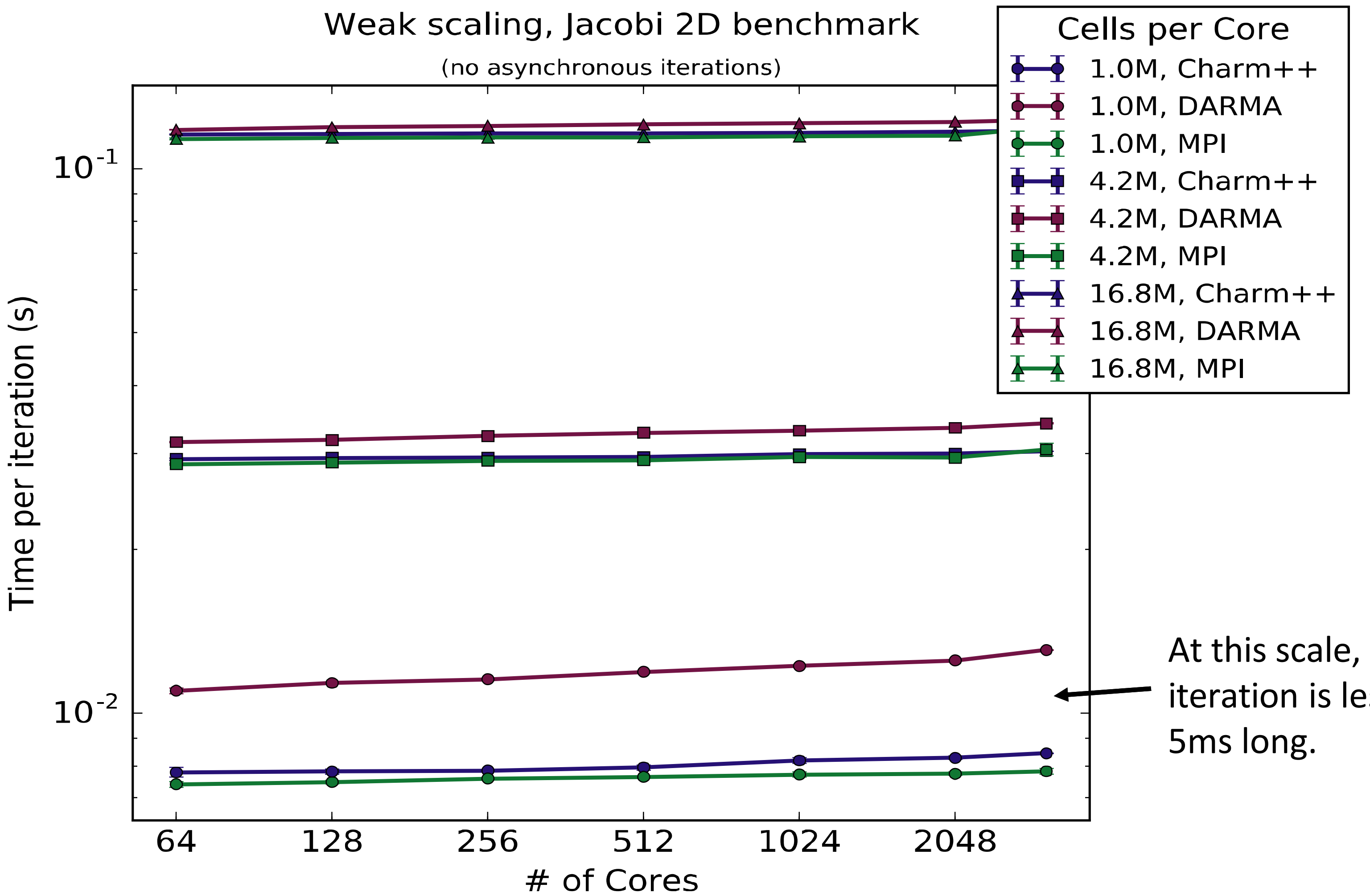
```
auto mycol = initial_access_collection<int>(
  index_range = Range1D(10)
);
create_concurrent_work<MyFun>(
  mycol, index_range = Range1D(10)
);

struct MyFun {
  void operator()(
    Index1D i, AccessHandleCollection<int> col
  ) {
    int me = i.value, mx = i.max_value;

    auto my_elm = col[i].local_access();

    my_elm publish(version=”x”);

    auto neighbor = me-1 < 0 ? mx : me-1;
    auto other_elm = col[neighbor].read_access(version=”x”);
    create_work([=]{
      cout << “neighbor = ” << other_elm get_value() << endl;
    });
  }
};
```

Identity map between these data and tasks. Thus, index i has local access to data index i.

Any other index must be read using `read_access`, which actually may be a remote or local operation depending on the backend mapping, but is always a deferred operation.

Goal 2) Facilitating the expression of coarse-grained tasking

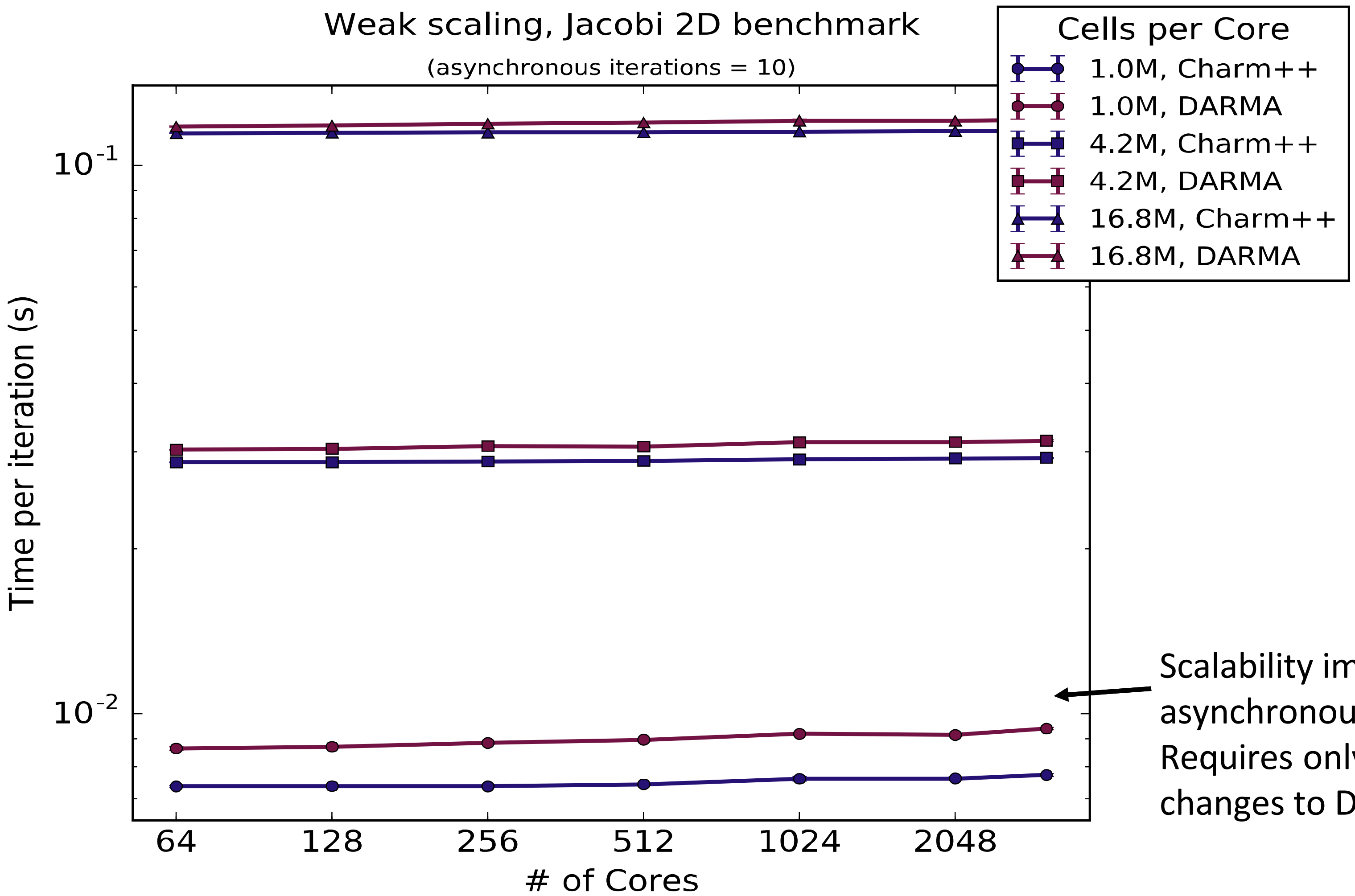
Stencil benchmark is not latency tolerant and highlights runtime overheads when task-granularity is small



At this scale, each iteration is less than 5ms long.

Using DARMA to inform Sandia's ATDM technical roadmap

Increased asynchrony in application enables runtime to overlap communication and computation



Scalability improves with asynchronous iterations. Requires only minor changes to DARMA code.

Using DARMA to inform Sandia's ATDM technical roadmap