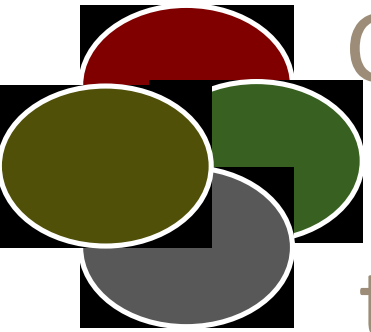
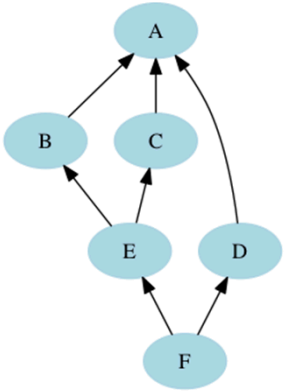
SAND2015-3015PE

Exceptional service in the national interest

Sandia National Laboratories



Can many-task models make data movement and fault-tolerance easier to express?



Jeremiah Wilke, Janine Bennett, Nicole Slattengren, Hemanth Kolla, Keita Teranishi, Gary Templet, Craig Ulmer, Ken Franko, Greg Sjaardema, John Floren

4/27/2015

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

Architecture trends pose serious challenges for how we program apps

Sandia National Laboratories

Challenges
▪ Increases in concurrency
▪ Dynamic app workloads
▪ Deep memory hierarchies
▪ Increased fail-stop errors
▪ Performance heterogeneity ▪ Accelerators ▪ Thermal throttling ▪ General system noise ▪ Responses to transient failures

Overarching abstract machine model of an exascale node

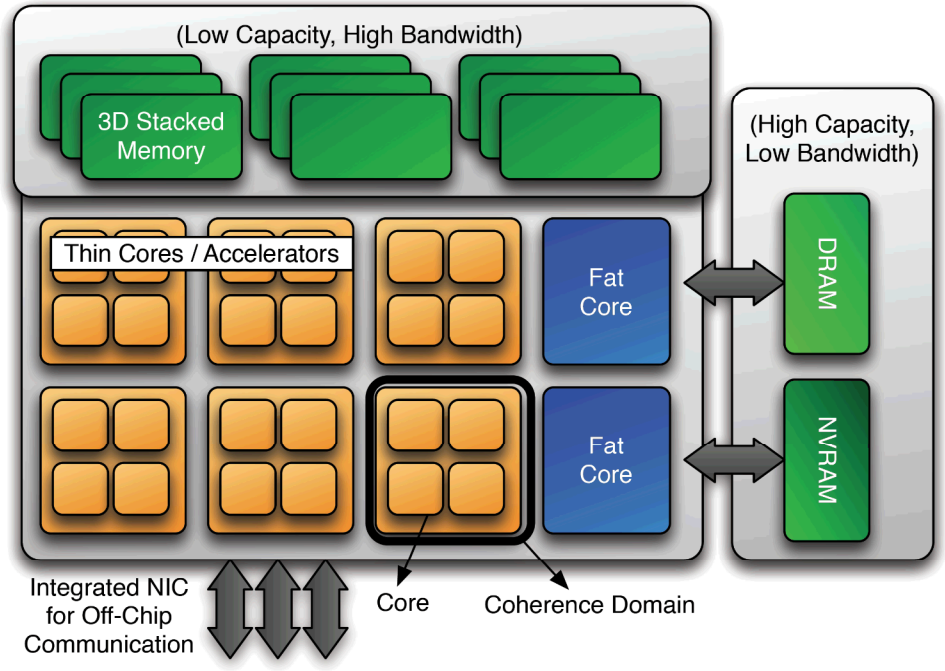


Image courtesy of www.cal-design.org

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Extant models (MPI+X) may not address all the algorithm and architecture challenges

Sandia National Laboratories

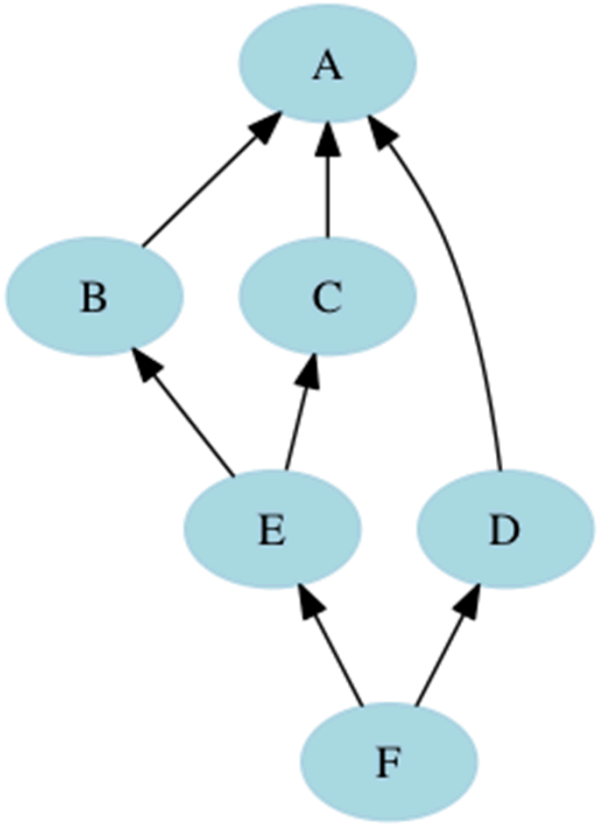
Challenges
Increases in concurrency
Dynamic app workloads
Deep memory hierarchies
Increased fail-stop errors
Performance heterogeneity ▪ Accelerators ▪ Thermal throttling ▪ General system noise ▪ Responses to transient failures

- Complexity of application code increases with proposed solutions

- Requires new execution models enabled by more expressive programming models

Emphasis on *dynamic*, coarse-grained parallelism, mostly affecting distributed memory

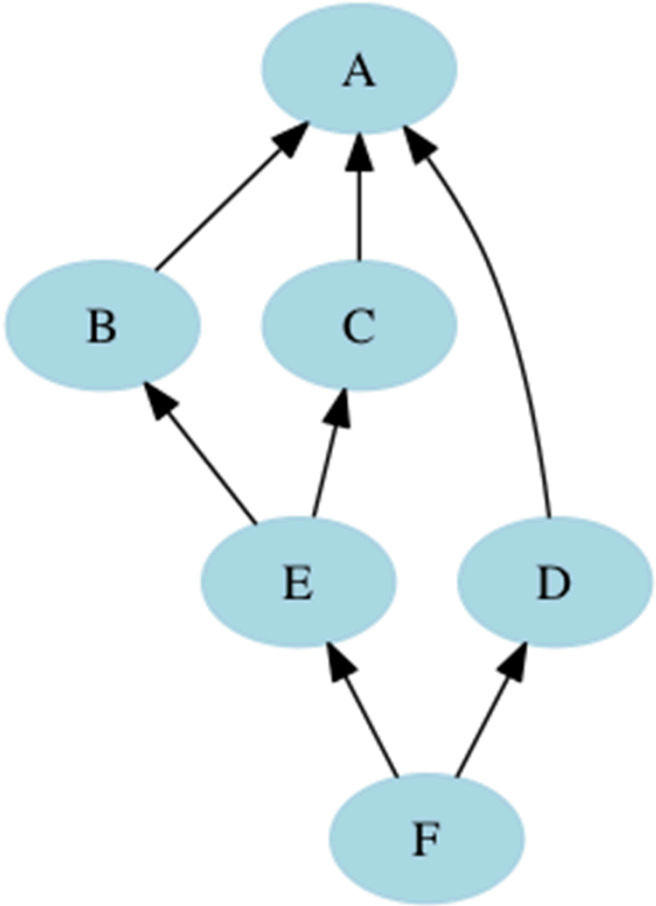
Asynchronous many-task (AMT) utilizes task graph to expose parallelism



Contrast between declarative and sequential, imperative coding styles

Sandia National Laboratories

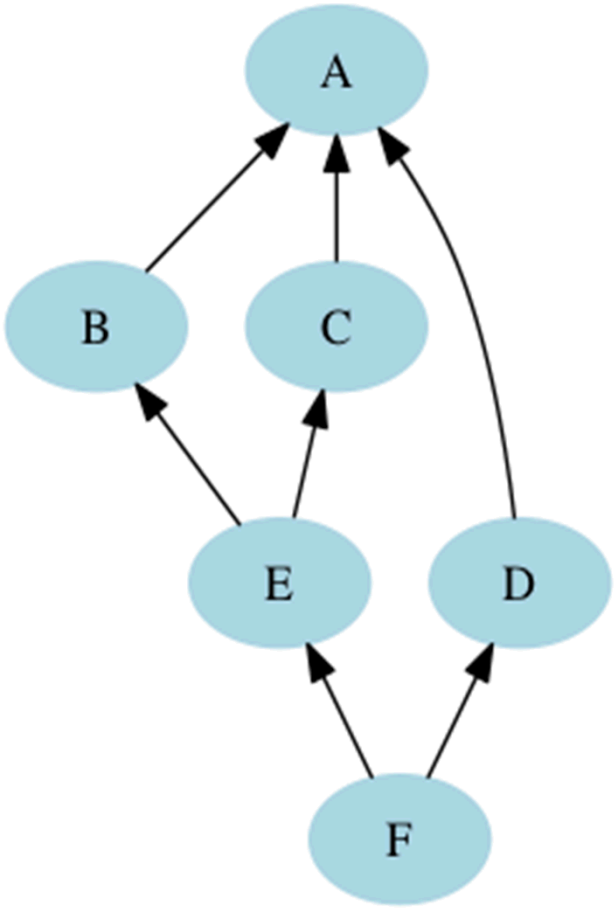
```
Output sequential()
{
  Output A = do_A();
  Output B = do_B(A);
  Output C = do_C(A);
  Output D = do_D(A);
  Output E = do_E(B,C);
  Output F = do_F(D,E);
  return F;
}
```



Contrast between declarative and sequential, imperative coding styles

Sandia National Laboratories

```
Output declarative()
{
  TaskDag dag;
  Task A;
  dag.add(A);
  Task B(A), C(A), D(A);
  dag.add(B,C,D);
  Task E(B,C);
  dag.add(E);
  Task F(E);
  dag.add(F);
  dag.run();
  return dag.result();
}
```

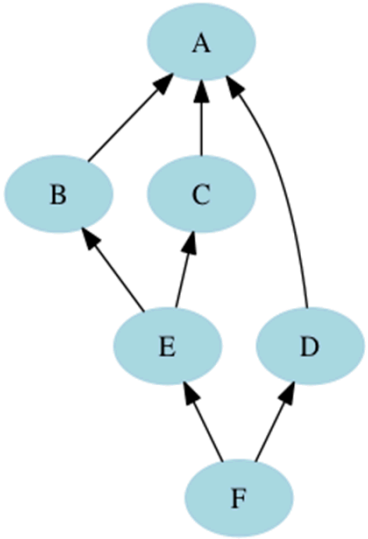


Programming model AND execution model both illustrated

Sandia National Laboratories

- Programming model: Do we code via procedure calls with pointers and scalar types? Or do we code via task declarations on logical handles?
- Execution model: Does the code run in a sequential, step-by-step way? Event-driven?

```
Output sequential()
{
  Output A = do_A();
  Output B = do_B(A);
  Output C = do_C(A);
  Output D = do_D(A);
  Output E = do_E(B,C);
  Output F = do_F(D,E);
  return F;
}
```

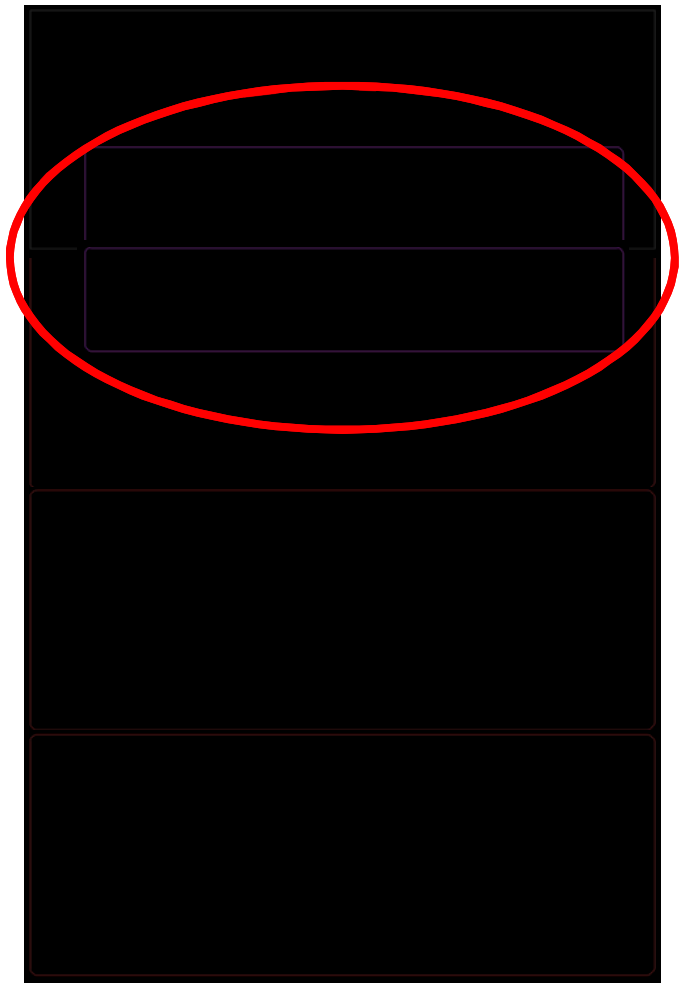


DHARMA project and the software stack

Sandia National Laboratories

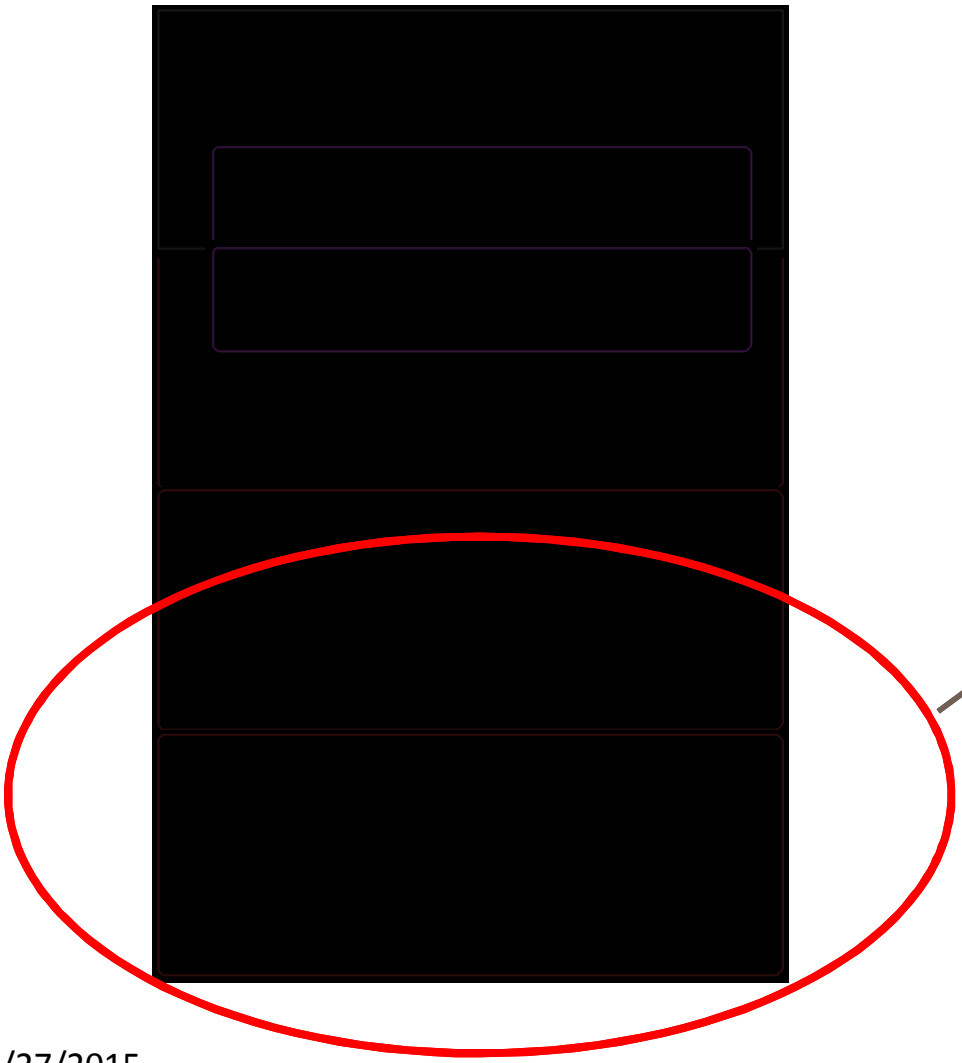
Applications are by far the largest part of the stack =$$$$$$$

DHARMA project and the software stack



- DHARMA exists in space between applications and runtime
- Programming model and execution model are API/abstractions within the application and runtime software space

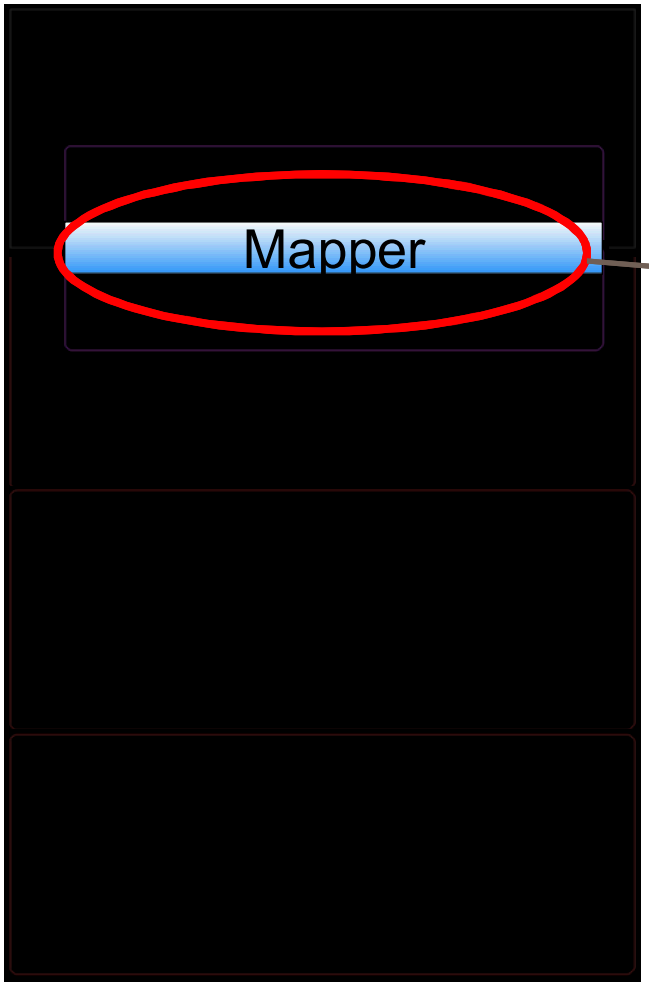
DHARMA project and the software stack



How do you future-proof the stack against changes in hardware?

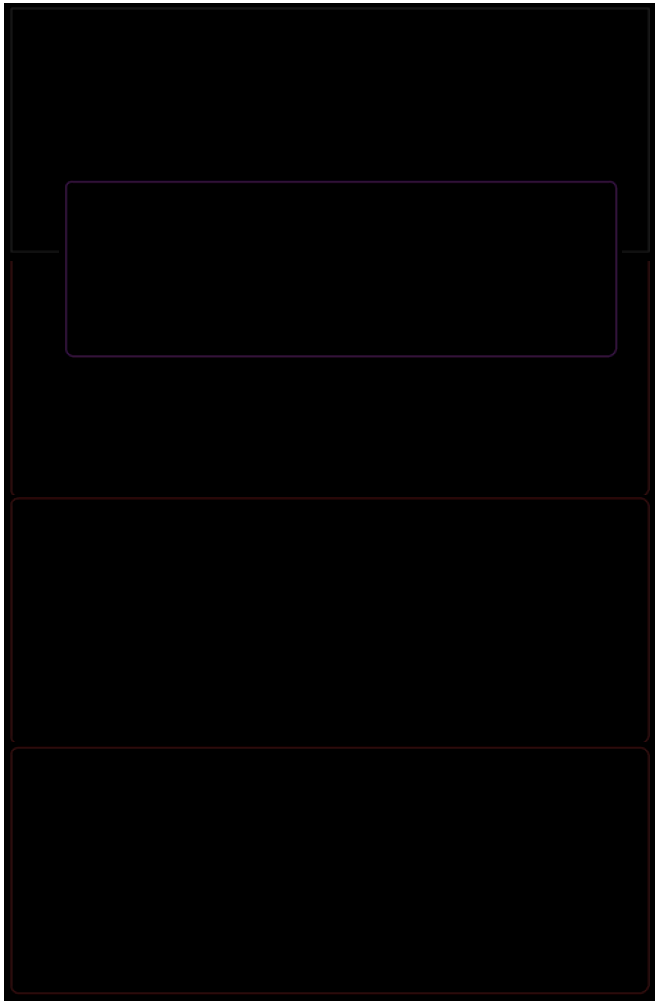
DHARMA project and the software stack

Sandia National Laboratories



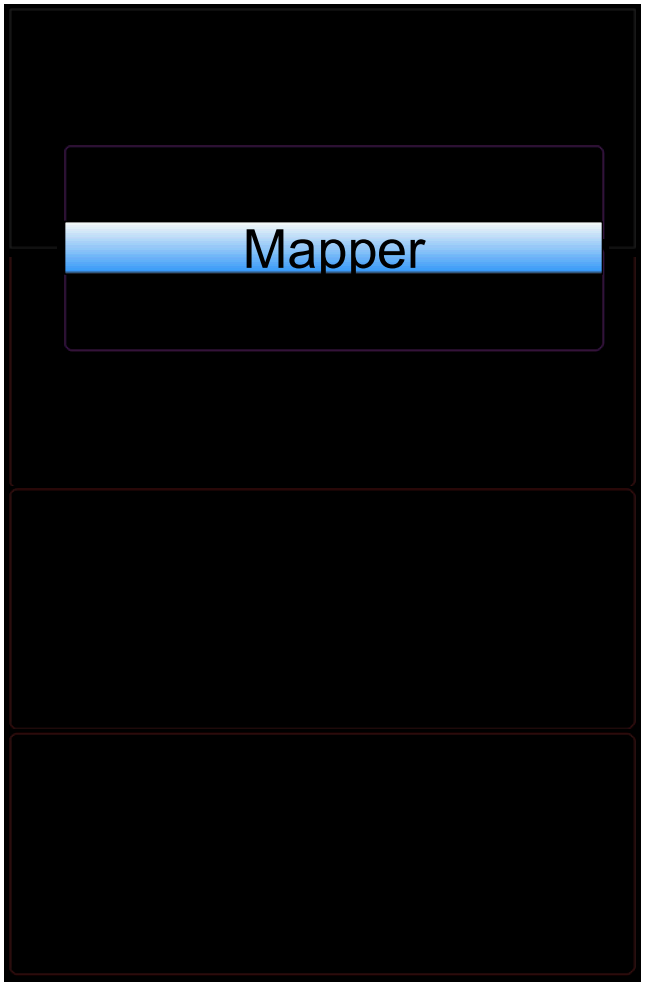
Can performance portability libraries and compilers suffice as shims to future-proof the stack?

Coding directly to an execution model has short-term rewards but major *risks*



- It can be easier to code directly to an execution model
- You cannot beat the performance of a code tuned to a specific execution model and hardware
- Loss of flexibility, greater commitment to smaller set of technologies

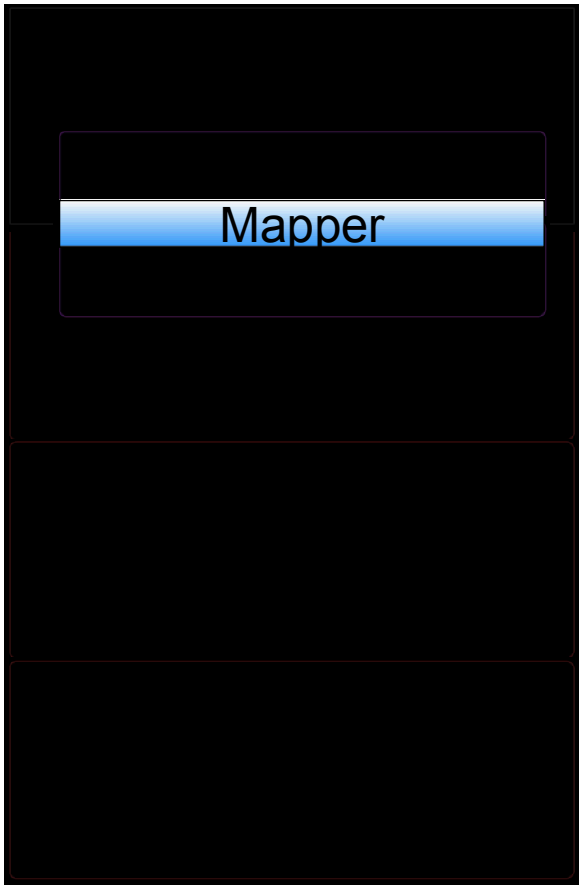
Optimistic assertions for current work



- Everything can pivot around the programming model as long it is sufficiently *expressive*
- Programming models can be “performance portable”
- An execution is only performance portable to similar runtimes/hardware

DHARMA is providing AMT capabilities to enable ATDM apps on ATS-3

Sandia National Laboratories



- Identify requirements for programming and execution model
- Focus on asynchronous many-task parallelism for distributed memory
- FY15 Activity #1: Analyze existing AMT technologies and assess technology gaps ($1.1M)
- FY15 Activity #2: Develop technical roadmap for ATS-3 AMT programming model ($0.3M)

DHARMA addresses key CIS mission area and touches many CIS research challenges

Sandia National Laboratories

- Mission area: Vertically integrated scalable computing
- Enabling new scientific/engineering capabilities via improved programming model broadly applicable
- Research challenge: Revolutionary approaches to the stockpile
- Research challenge: Engineering of materials
- Research challenge: First to high yield fusion
- Research challenge: Data sciences
  - Improved workflows on emerging architectures will benefit greatly from expressive programming model that enhances code coupling

Conclusions (preview)

Sandia National Laboratories

- DHARMA is both *computer science* and *economics*. We must balance reward of improved performance against risk of investing in new/existing codes.
- An expressive programming model provides risk mitigation by enabling many different runtimes/libraries and execution models
- Programming/execution model relationship critical to making sure major rewrite of applications happens ONCE

DHARMA started as AMT-based fault-tolerance research vehicle

Sandia National Laboratories

- Dharma: the order which makes life possible
- Dhr: Sanskrit – to hold, maintain, keep
- Originally: ***D***istributed ***H***ash ***A***rray for ***R***emote ***M***emory ***A***ccess – research vehicle
- Now: **D**istributed async**H**ronous, **A**daptive, **R**esilient **M**odels for **A**pplications – path to production capabilities

DHARMA started as AMT-based fault-tolerance research vehicle

Sandia National Laboratories

- Dharma: the order which makes life possible
- Dhr: Sanskrit – to hold, maintain, keep
- Originally: ***D***istributed ***H***ash ***A***rray for ***R***emote ***M***emory ***A***ccess
- Now: **D**istributed async**H**ronous, **A**daptive, **R**esilient **M**odels for **A**pplications

Noble truth of scalable computing:

All life is suffering. Our desire for more flops is the source of all suffering.

DHARMA has progressed through three phases of enlightenment

Sandia National Laboratories

DHARMA has progressed through three phases of enlightenment

Sandia National Laboratories

- Ep. I: A New Hope
  - Optimism AMT models will revolutionize software and save the world
  - Port mini-app to 3 exemplar AMT systems: Legion, Charm++, Uintah
  - Engaging both app developers and runtime developers

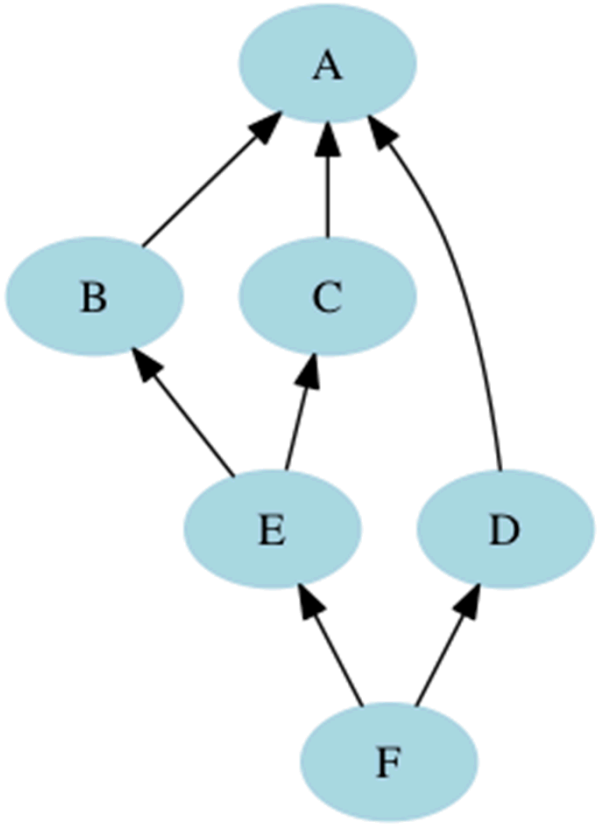
Escalating features in many-task models can improve performance – at a cost

Sandia National Laboratories

Feature	Where it might help	Trade-off	Features prominently
1) Data parallel	Everywhere	Shared/ghost sync	Everywhere
2) Task-parallel on-node	Multiphysics	Task ordering Data coherency	SMPs, Uintah
3) Work stealing on-node	Dynamic work loads (mesh smoothing, feature analysis)	Cache problems Termination detection	Cilk, X10
4) Global sync load balancing	Re-meshing Poor initial distribution	Global synchronization	MPI
5) Distributed memory task parallel	Quantum chemistry	Distributed coherency	NWChem, Charm++
6) Distributed work stealing	Particle-in-cell Molecular dynamics	Global termination detection	TASCEL, Charm++
7) Runtime DAG derivation	Many fields, tasks that are difficult to manage	Rigid data structures, runtime overhead	Legion

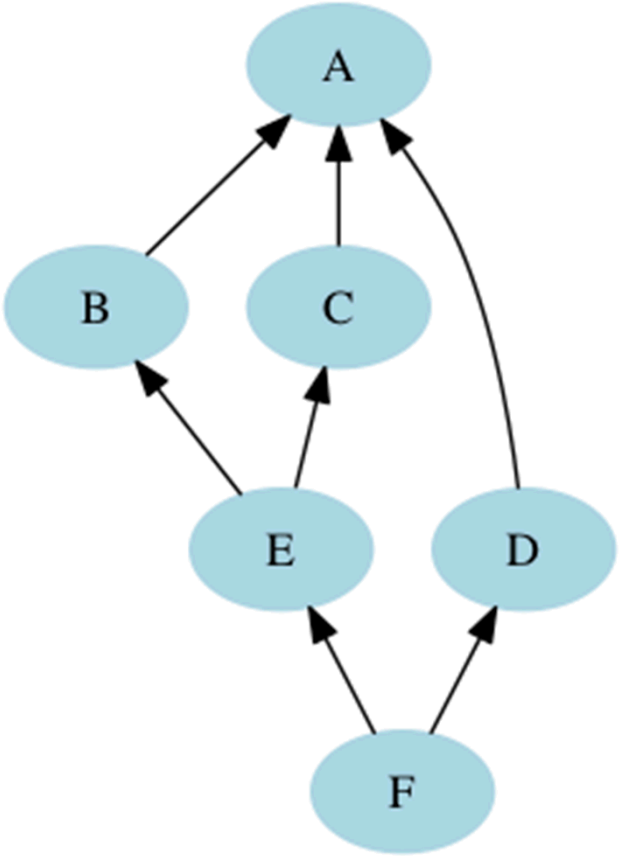
Asynchronous many-task (AMT) utilizes task graph to expose parallelism subject to order constraints

Sandia National Laboratories



Sequential schedule has no parallelism

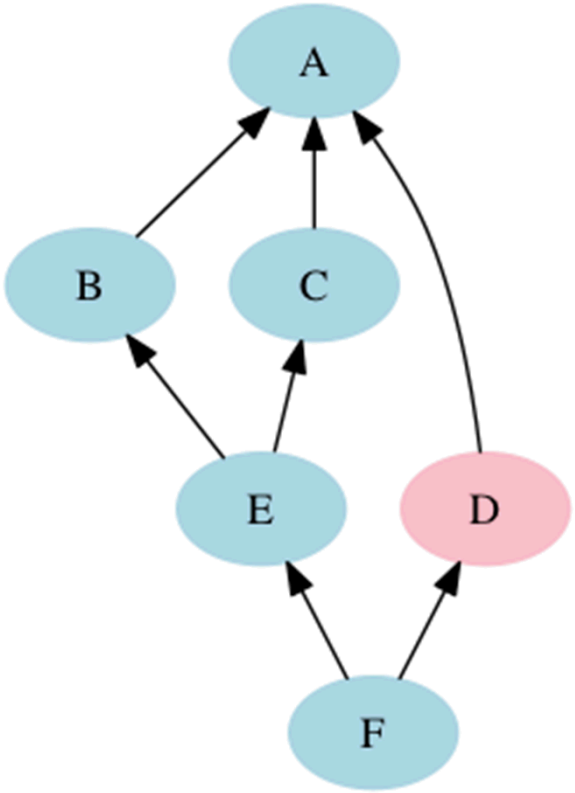
	T=0	T=1	T=2	T=3	T=4	T=5	Time
Proc 0	A	B	C	D	E	F	6



Task parallelism enables multiple workers to lower schedule length

Sandia National Laboratories

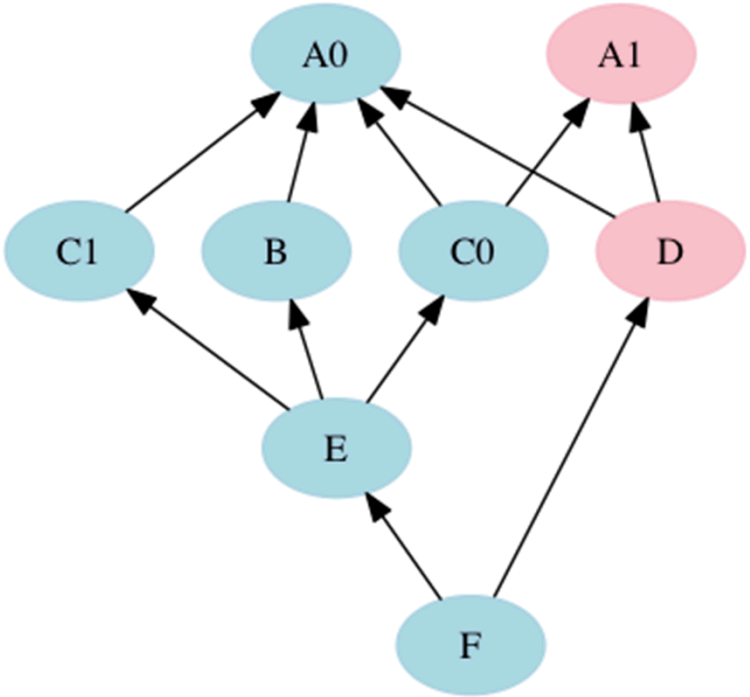
	T=0	T=1	T=2	T=3	T=4	Total
Proc 0	A	B	C	E	F	5
Proc 1		D				



Data parallelism adds another dimension of parallelism to exploit

Sandia National Laboratories

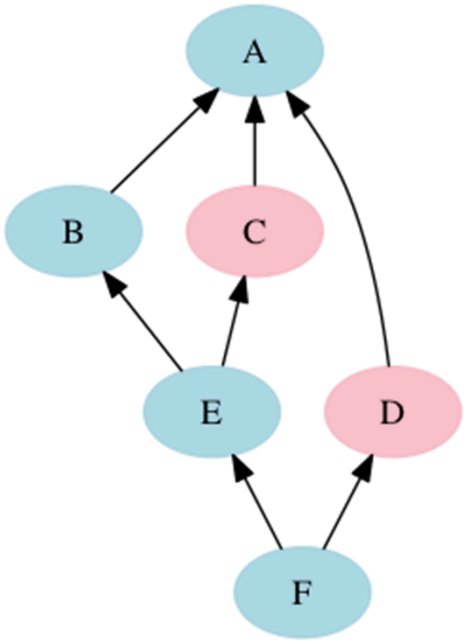
	T=0	T=0.5	T=1.5	T=2.0	T=2.5	T=3.5	Total
Proc 0	A0	B	C0	C1	E	F	4.5
Proc 1	A1	D					



Work stealing can dynamically change schedule to improve performance

Sandia National Laboratories

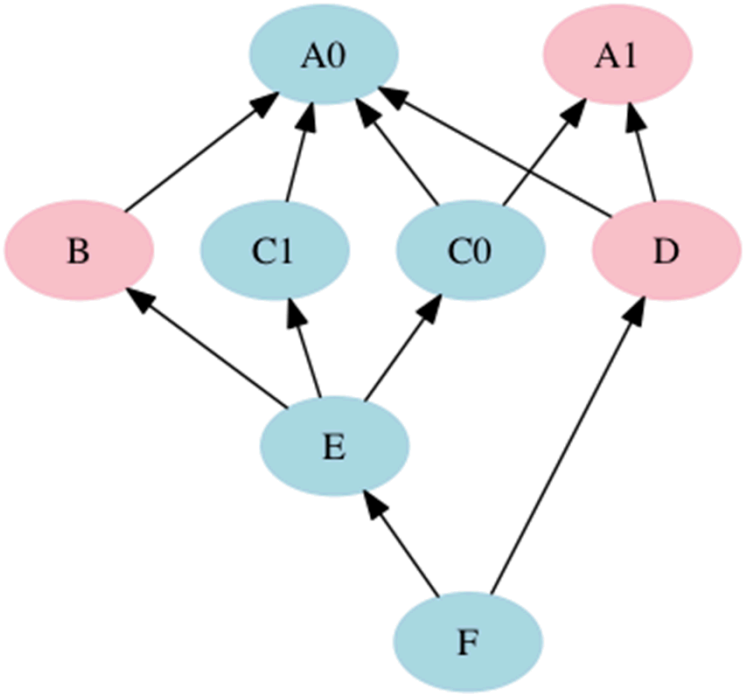
	T=0	T=1	T=2	T=3	Total
Proc 0	A	B	E	F	4
Proc 1		C	D		



Combing multiple strategies can maximize parallelism

Sandia National Laboratories

	T=0	T=0.5	T=1.0	T=1.5	T=2.5	Total
Proc 0	A0	C0	C1	E	F	3.5
Proc 1	A1	B		D		



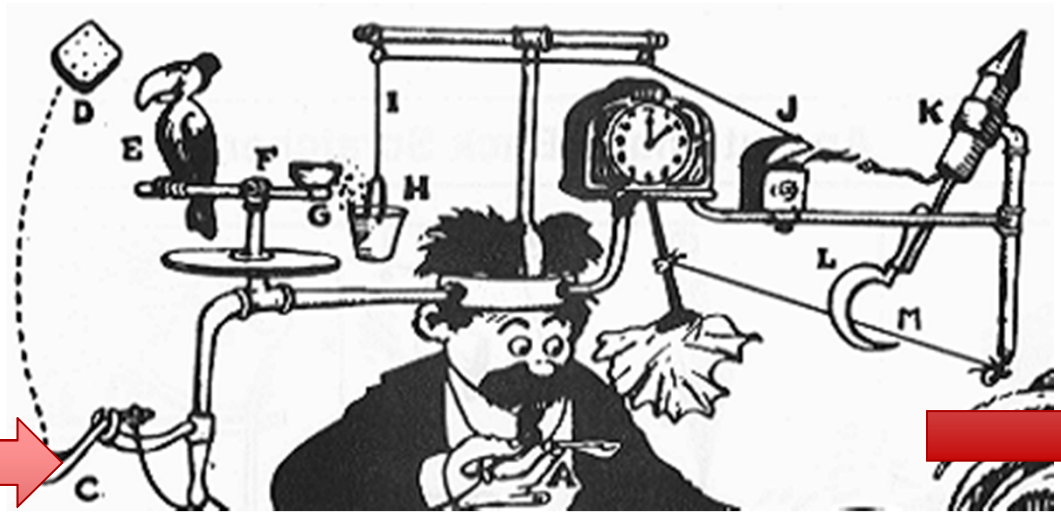
DHARMA has progressed through three phases of enlightenment

Sandia National Laboratories

- Ep. I: A New Hope
  - Optimism AMT models will revolutionize software and save the world
  - Port mini-app to 3 exemplar AMT systems: Legion, Charm++, Uintah
  - Engaging both app developers and runtime developers
- Ep. II: The MPIre Strikes Back
  - Deeper understanding of AMT models – pitfalls and being fair to MPI

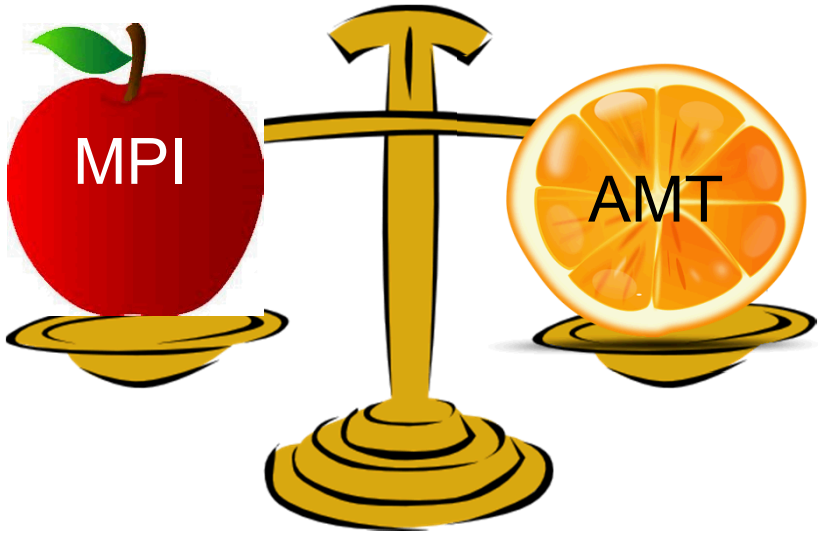
Rube-Goldberg principle of supercomputing benchmarks

Sandia National Laboratories



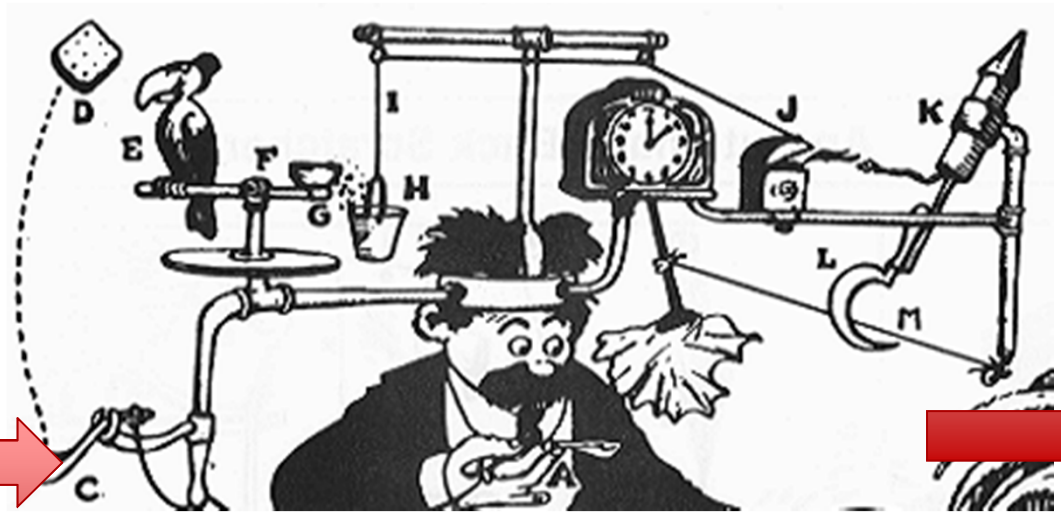
Results
????

Input



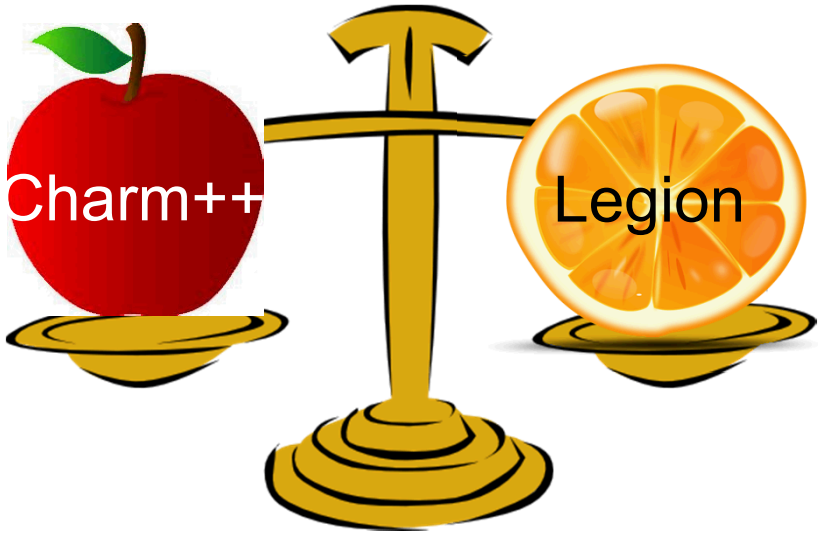
Rube-Goldberg principle of supercomputing benchmarks

Sandia National Laboratories



Results
????

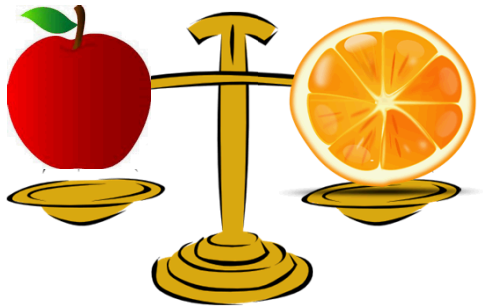
Input



DHARMA is taking a measured approach looking towards best long-term solution

Sandia National Laboratories

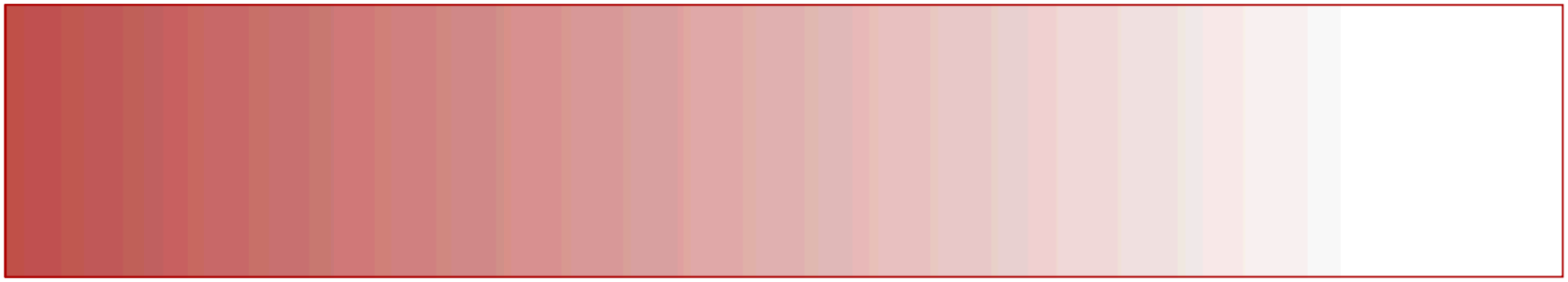
- The AMT technical roadmap addresses:
  ①Performance portability
  ②Programmability/expressiveness
  ③Interoperability (both code and organization)
- 4-5 year roadmap for building programming model solution
- Major overhaul of code base should only happen ONCE!
- Not just about performance of existing AMT systems on current platforms!



Sandia faces spectrum of choices/risks in developing technical roadmap

Sandia builds system from scratch and takes ownership

Sandia relies completely on external academic partners



Lots of control, but lots of extra investment

Less control, but less investment

DHARMA has progressed through three phases of enlightenment

Sandia National Laboratories

- Ep. I: A New Hope
  - Optimism AMT models will revolutionize software and save the world
  - Port mini-app to 3 exemplar AMT systems: Legion, Charm++, Uintah
  - Engaging both app developers and runtime developers
- Ep. II: The MPIre Strikes Back
  - Deeper understanding of AMT models – pitfalls and being fair to MPI
- Ep. III: Return of the Linda
  - Draft programming model spec based on Sandia's application workload
  - Expressiveness is major theme of coordination languages like Linda
  - Linda concepts have informed active AMT projects like concurrent collections

C++ : Assembly :: AMT : MPI

```
int main(){
  int a = 0;
  int N = 100;
  for (int idx=0; idx < N; ++idx){
    a += idx;
  }
}
```

```
int main(){
  int a = 0;
  int N = 100;
  reduce(int idx=0; idx < N; ++idx){
    a += idx;
  }
}
```

```
LBB0_1:
  movl  -16(%rbp), %eax
  cmpl  -12(%rbp), %eax
  jge LBB0_4
  .loc  1 7 0
  movl  -16(%rbp), %eax
  movl  -8(%rbp), %ecx
  addl  %eax, %ecx
  movl  %ecx, -8(%rbp)
  .loc  1 6 0
  movl  -16(%rbp), %eax
  addl  $1, %eax
  movl  %eax, -16(%rbp)
  jmp LBB0_1
```

Variable names are logical identifiers
Assembly is physical implementation (prescribes execution)

C++ : Assembly :: AMT : MPI

Sandia National Laboratories

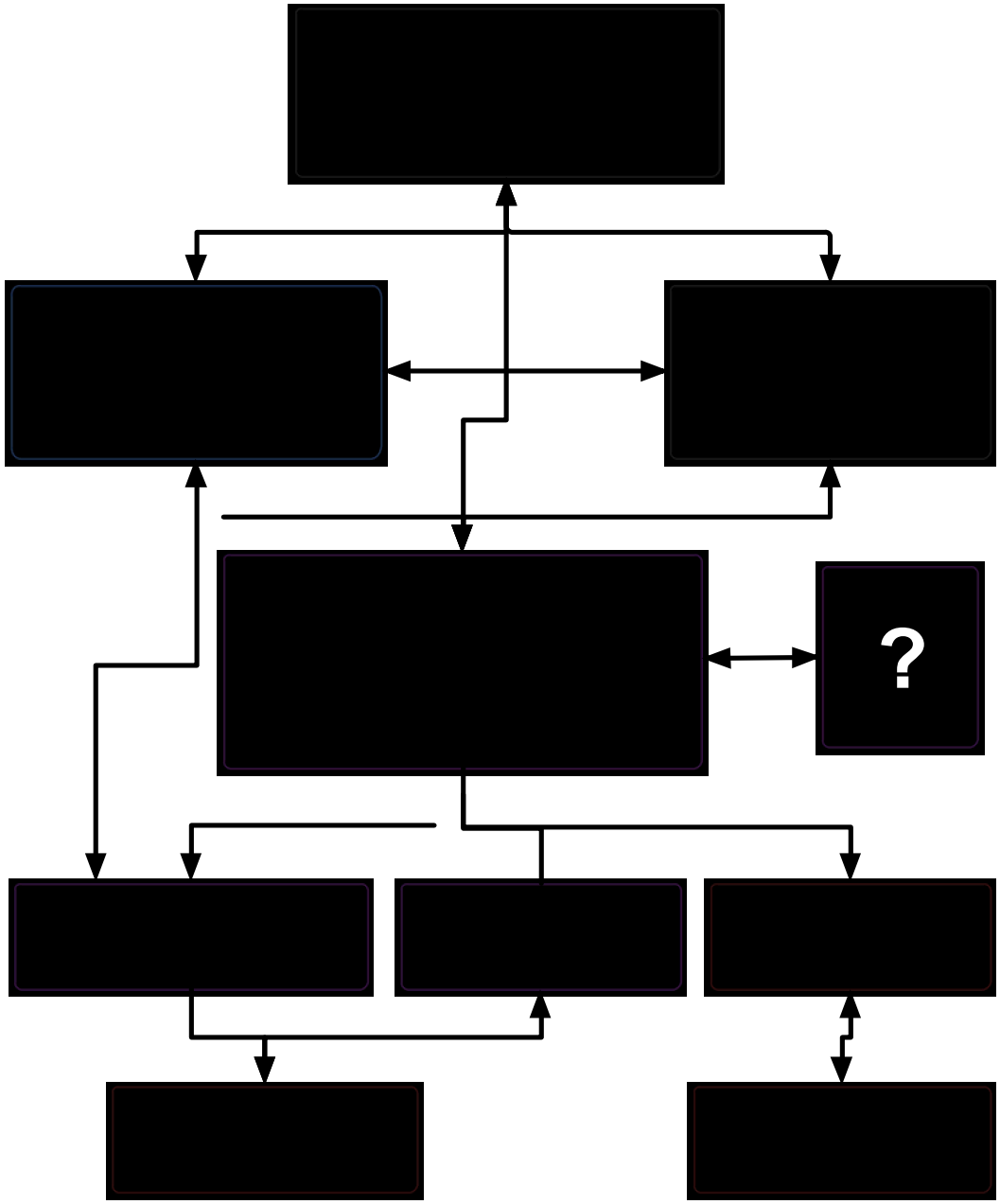
```
double* ghost = ...
double* shared = ...
MPI_Isend(ghost);
MPI_Irecv(shared);
MPI_Waitall();
compute_stencil();
```

Physical pointers passed to runtime. Execution explicitly defined.

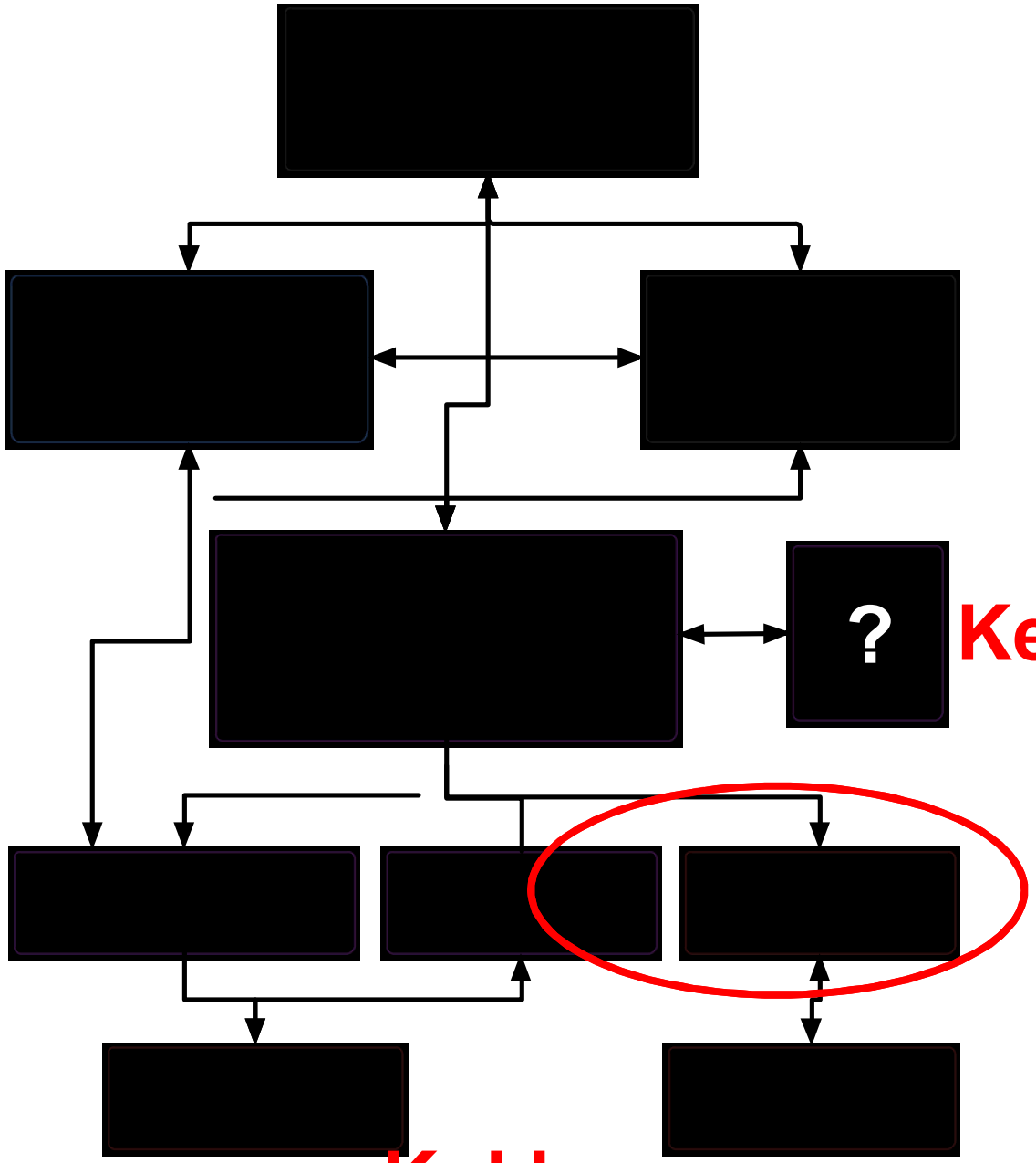
Data movement is described, but can be mapped to multiple different executions

```
double* reg0 = ...
int rightNode = rt->rank() + 1;
rt->put("R0 it1", reg0);
rt->subset("R0 it1", "ghost", reg0, ...);
double* shared;
rt->get("R1 iter 1",
  SUBSET, "ghost",
  SOURCE, rightNode,
  &shared);
```

Key-value store semantics are low-hanging fruit to add expressiveness to with minimal intrusiveness



Flexible key-value store facilitates development of many interacting components



Strong connection to other Sandia projects

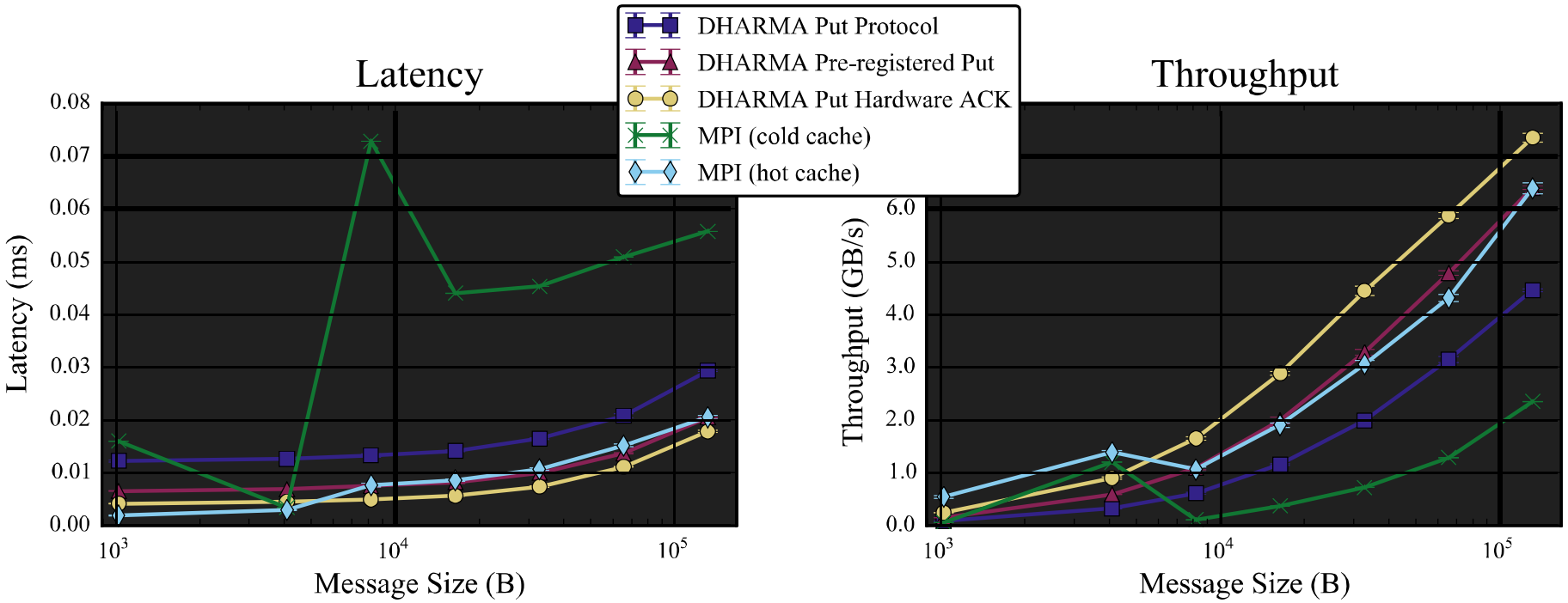
Logical declaration of program simplifies failure recovery (if no pessimistic message logging)

Sandia National Laboratories

```
int dharma_main(){
  /* add many other tasks */
  for (int i=0; i < N; ++i){
    rt->add_task<small>(i);
  }
  rt->add_task<ddot>()
      ->input("A",0)
      ->input("B",0);
  rt->run();
}
```

```
int mpi_main()
{
  int rc;
  rc = MPI_IRecv(A0);
  if (rc == PARTNER_FAILED){
    //do something - but what?
  }
  rc = MPI_Irecv(B0);
  if (rc == PARTNER_FAILED){
    //do something - but what?
  }
  rc = MPI_Wait(A0_request);
  if (rc == PARTNER_FAILED){
    //do something - but what?
  }
  rc = MPI_Wait(B0_request);
  if (rc == PARTNER_FAILED){
    //do something - but what?
  }
```

Key-value store overheads negligible relative to memory registration or eager copy



Cray XC30
Point-to-point exchanges

Conclusions

- DHARMA is both *computer science* and *economics*. We must balance reward of improved performance against risk of shifting new/existing codes.
- An expressive programming model provides risk mitigation by enabling many different runtimes/libraries and execution models – performance portability
- Programming/execution model relationship critical to making sure major rewrite of applications happens ONCE
- The community doesn't yet have all the answers, but we are asking the right questions.