The Legion Programming System

Alex Aiken
Stanford/SLAC

Pat McCormick
LANL

Legion

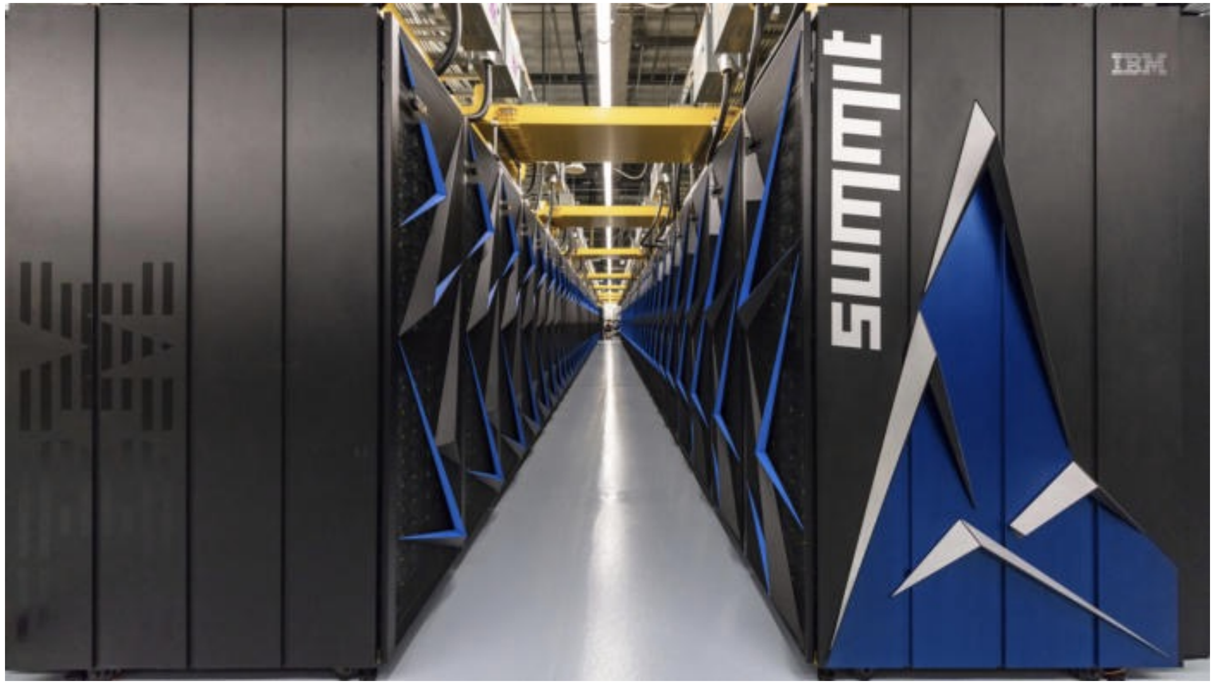
A data-centric, task-based programming model for parallel, accelerated, distributed machines.

History:

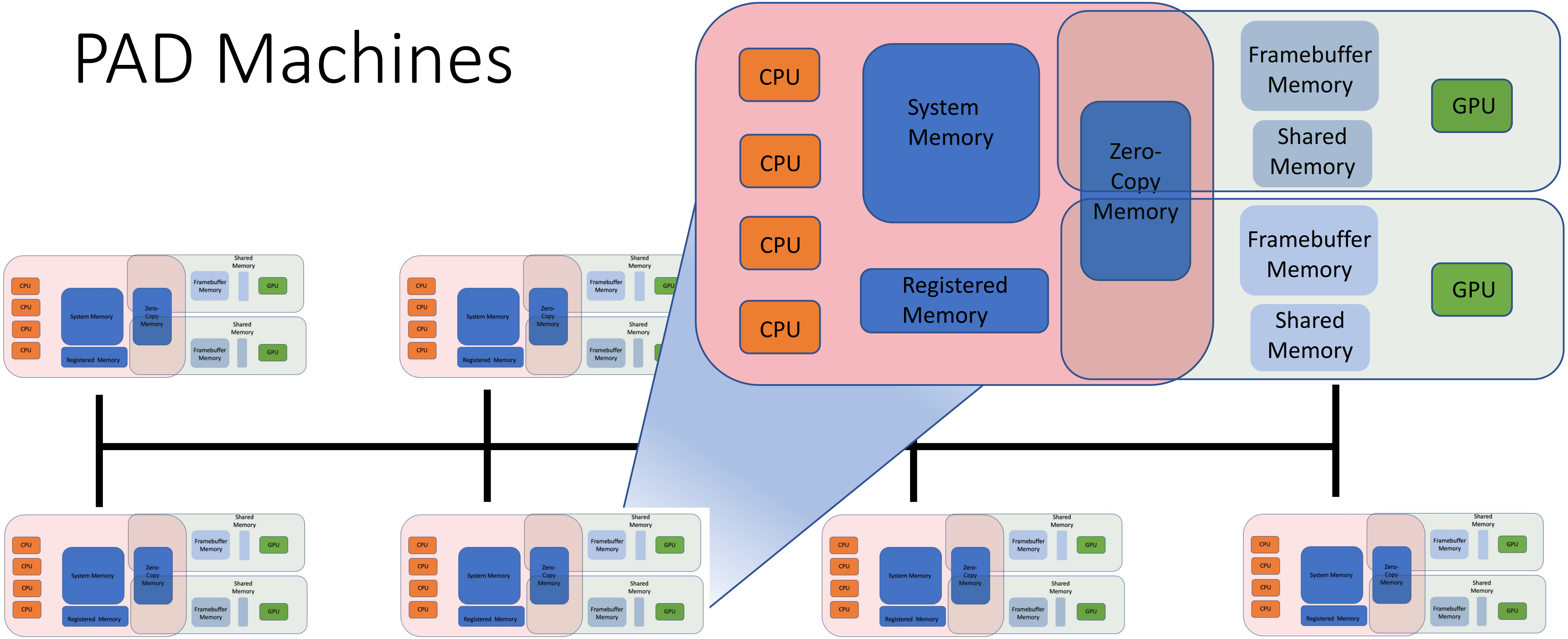
- Initially supported by ASCR through the ExaCT co-design center (~2011) and additional projects (w/ Lucy, Laura, Hal)
- Additional support from DOE NNSA ASC: PSAAP II, III, LANL, BES, SciDAC, DARPA; part of the ST portfolio in ECP
- Industry interest & investment, most notably NVIDIA & Facebook
- R&D 100 Award in 2020

Legion

A data-centric, task-based programming model for parallel, accelerated, distributed machines



PAD Machines



Very complex memory hierarchy & significant memory capacity constraints.

Why Do We Need New Programming Models?

- Because the hardware has changed
  - Every new DOE machine is now a PAD machine

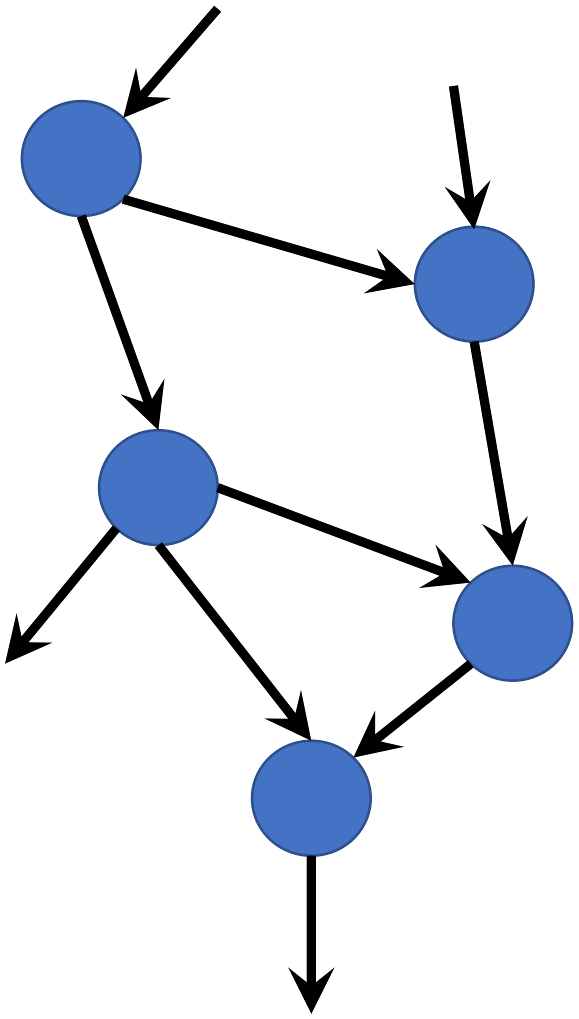
- Current programming models were designed for a different class of machines

- We are betting that task-based programming models are the best fit for PAD machines
  - Or at least a better fit
  - Fellow travelers: PaRSEC, StarPU

Legion

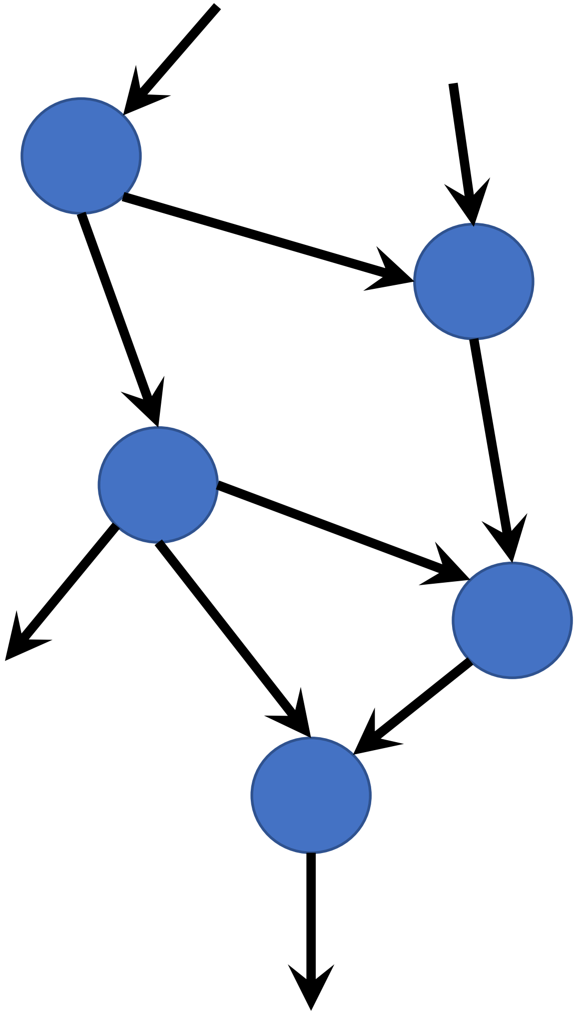
A data-centric, task-based programming model for parallel, accelerated, distributed machines

Task Graphs



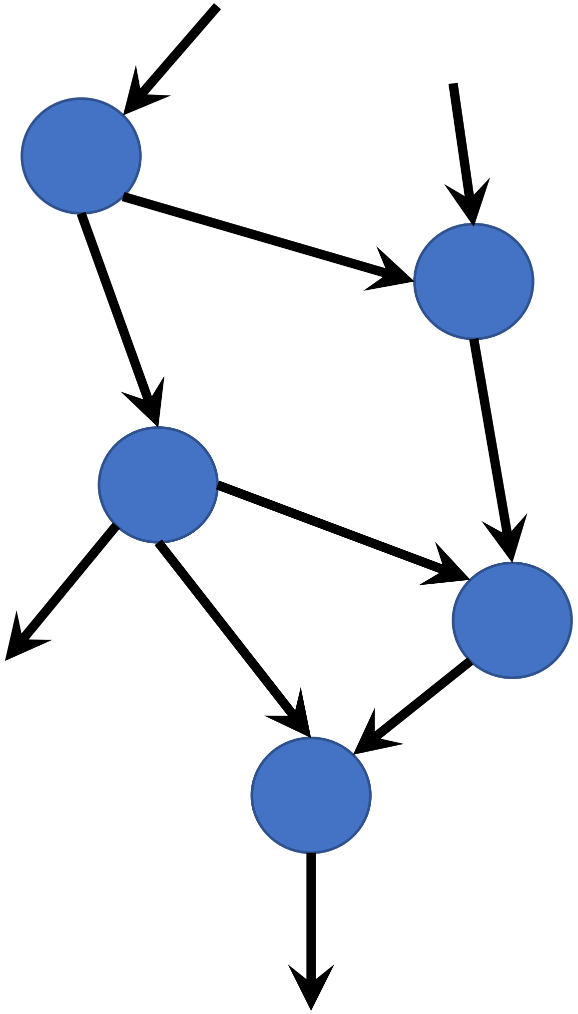
- Nodes are *tasks*
  - Units of work
- Edges are *dependencies*
  - Ordering constraints

Task Graphs



- Common in data-centric programming systems:
  - TensorFlow
  - PyTorch
  - MapReduce
  - Spark
  - DASK
- Why?
  - Productivity!
  - Provide access to supercomputer-scale resources to programmers who otherwise could not program such machines
- The challenge:
  - Provide the generality, scalability, and performance needed for DOE codes
  - While keeping the productivity

Task Graphs in Legion



- Nodes are *tasks*
  - Application functions & data movement
- Edges are *dependencies*
  - Ordering constraints
  - Inferred automatically as tasks are launched
- Asynchronous model
  - Deals gracefully with variable latencies
- Machine independent
  - No commitment to size of machine, where tasks execute, or where data is placed
  - Separate *mapping* embeds a task graph in a machine

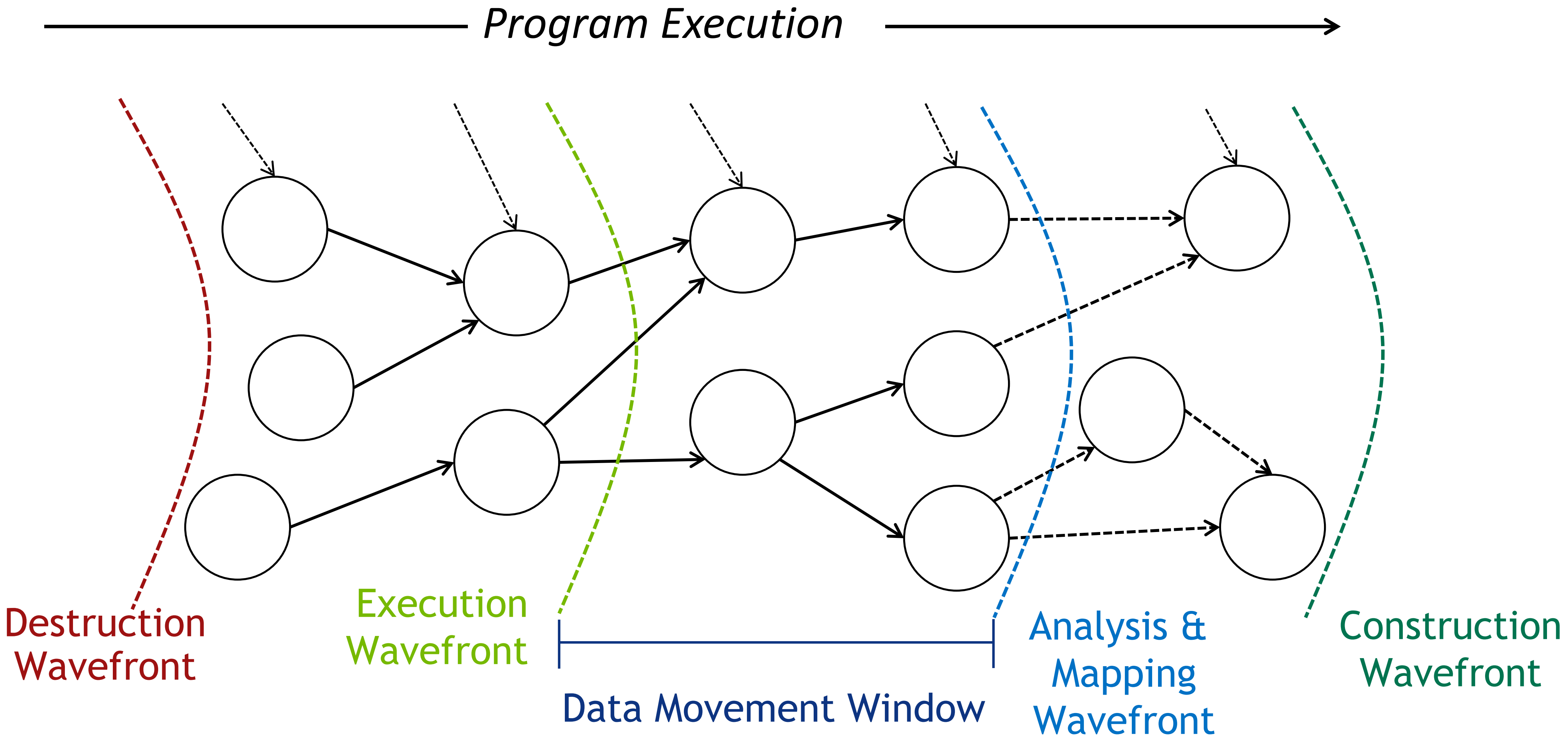
Source Code

```
for j = 0, conf.num_loops do
  for i = 0, conf.num_pieces do
    calculate_new_currents(steps, pn_private[i], pn_shared[i], pn_ghost[i], pw_outgoing[i])
  end
  for i = 0, conf.num_pieces do
   distribute_charge(pn_private[i], pn_shared[i], pn_ghost[i], pw_outgoing[i])
  end
  for i = 0, conf.num_pieces do
   update_voltages(pn_equal[i])
  end
end
```

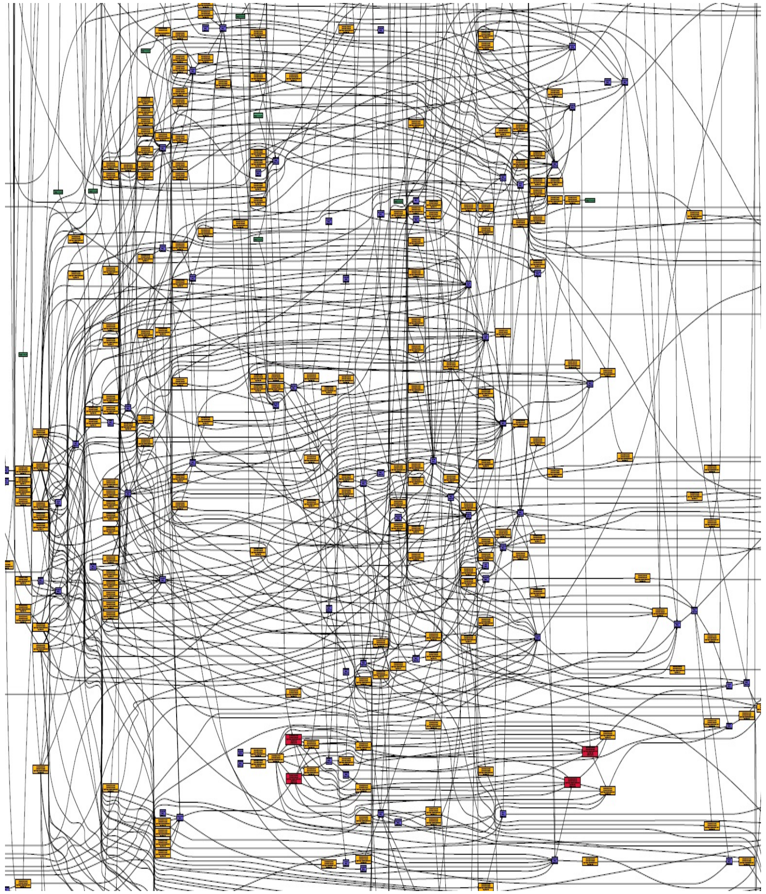
Programs simply launch tasks.

The (distributed) task graph is constructed dynamically by the Legion runtime.

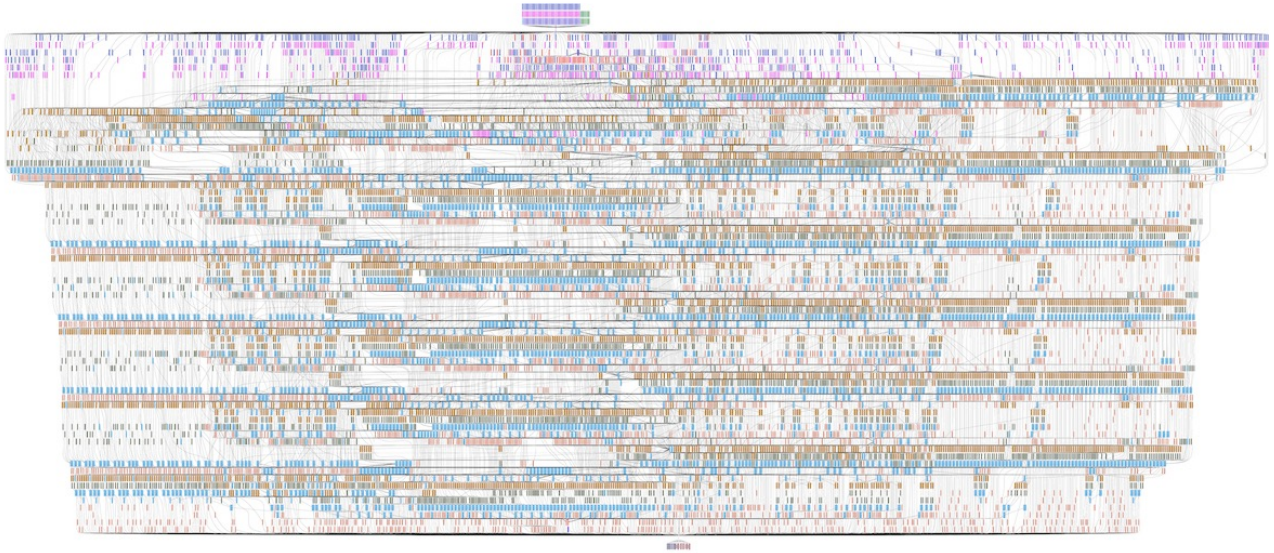
Dynamic Task Graph Construction & Execution



Task Graphs Get Large and Complex



The task graph for one iteration on one node … of a mini-app



Some applications have ~10K tasks/sec/node: E.g., one iteration of distributed memory S3D.

Legion

A data-centric, task-based programming model for parallel, accelerated, distributed machines

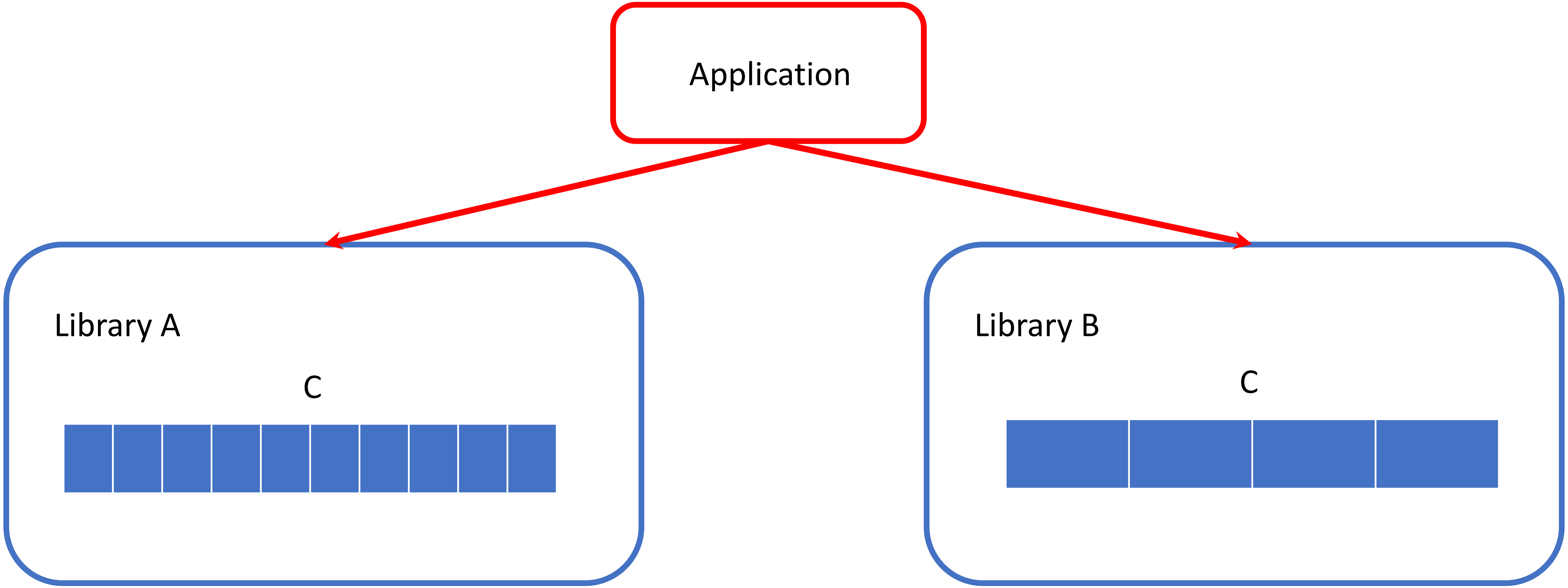
Partitioning

- Tasks work on *collections* of data

- For parallelism, collections can be partitioned into subcollections

- Partitions are first-class objects in Legion
  - Perhaps the most radical aspect of the design
  - Rich sublanguage of partitioning operations
  - Multiple partitions of the same data can exist simultaneously
  - Partitions can be hierarchical

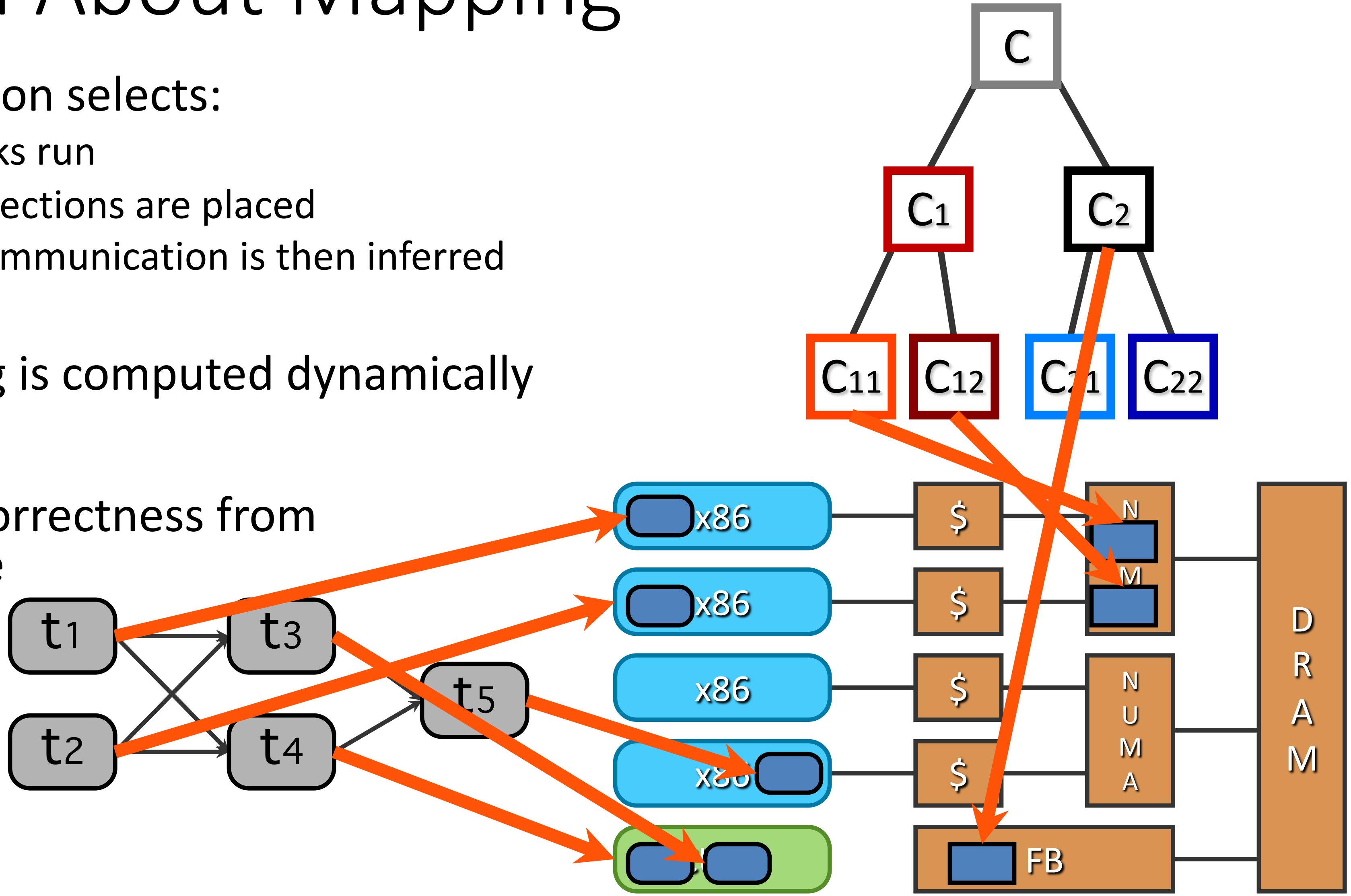
Why Allow Multiple Partitions? Composition!



Different libraries, written independently, may require different views of the same data.

A Word About Mapping

- The application selects:
  - Where tasks run
  - Where collections are placed
  - Needed communication is then inferred
- The mapping is computed dynamically
- Decouples correctness from performance

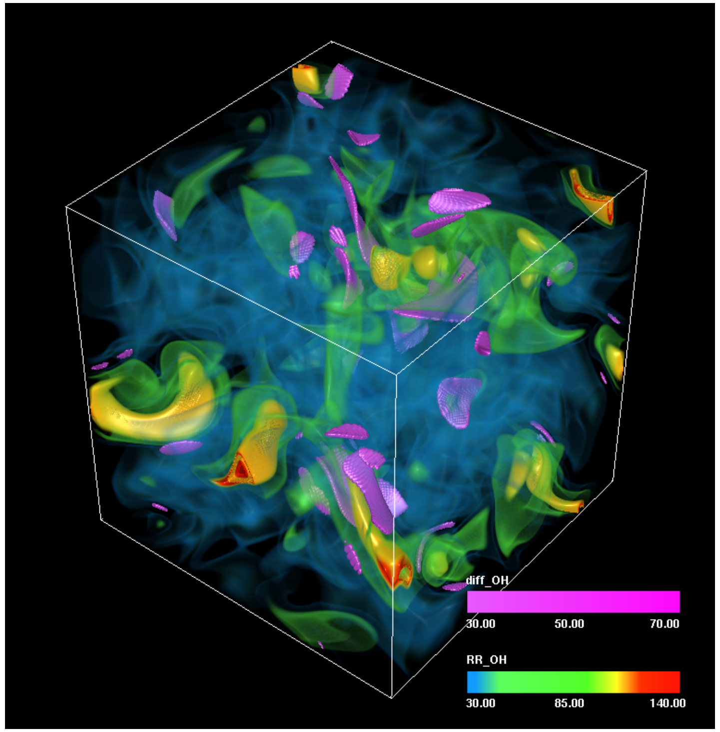


Applications

S3D: Combustion Simulation

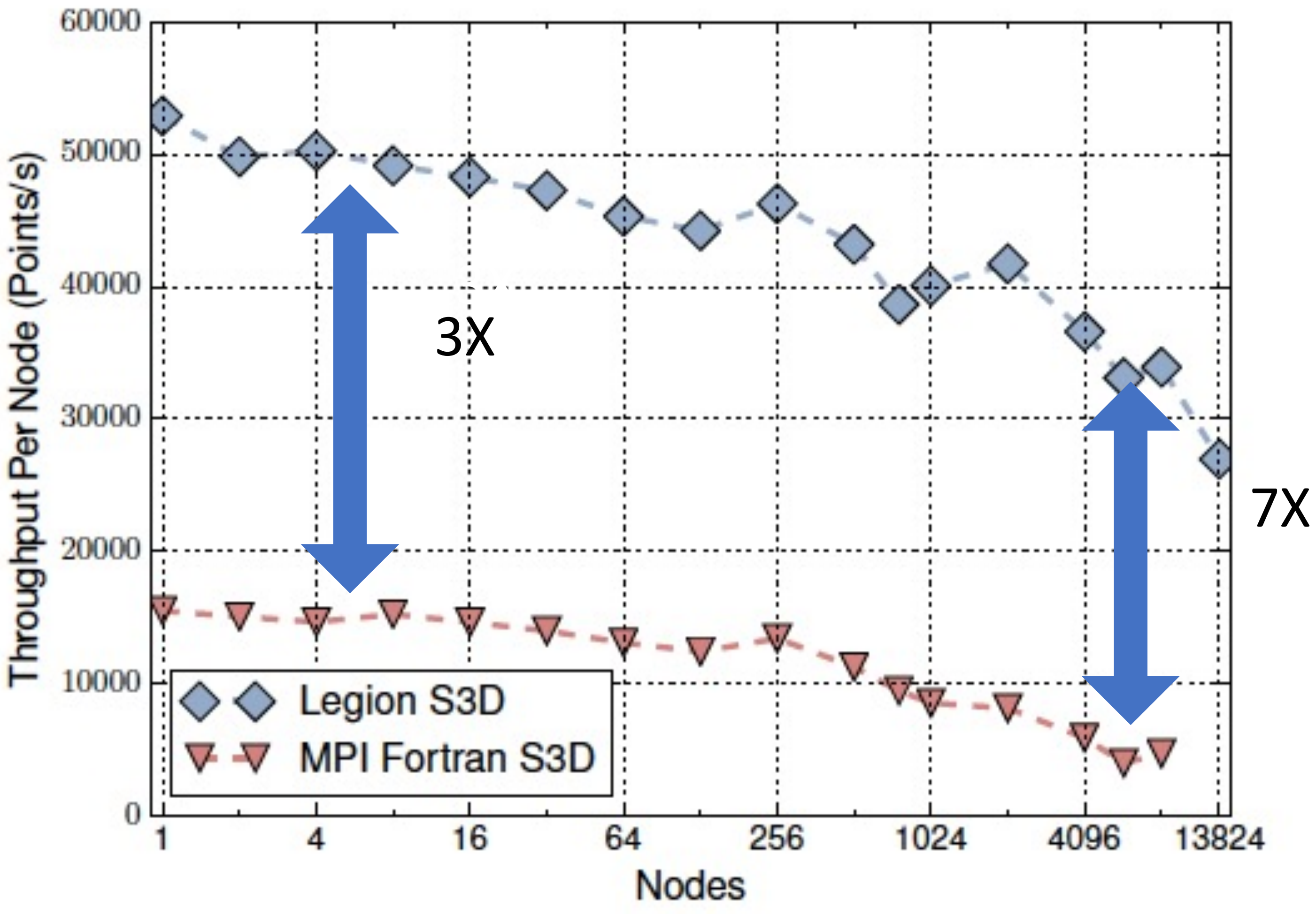
- Simulates chemical reactions
  - DME (30 species)
  - Heptane (52 species)
  - PRF (116 species)

- Two parts
  - Physics
    - Nearest neighbor communication
    - Data parallel
  - Chemistry
    - Local
    - Complex task parallelism
  - Large working sets/task



Recent 3D DNS of auto-ignition with 30-species DME chemistry (Bansal *et al*. 2011)

Weak Scaling: PRF on Titan

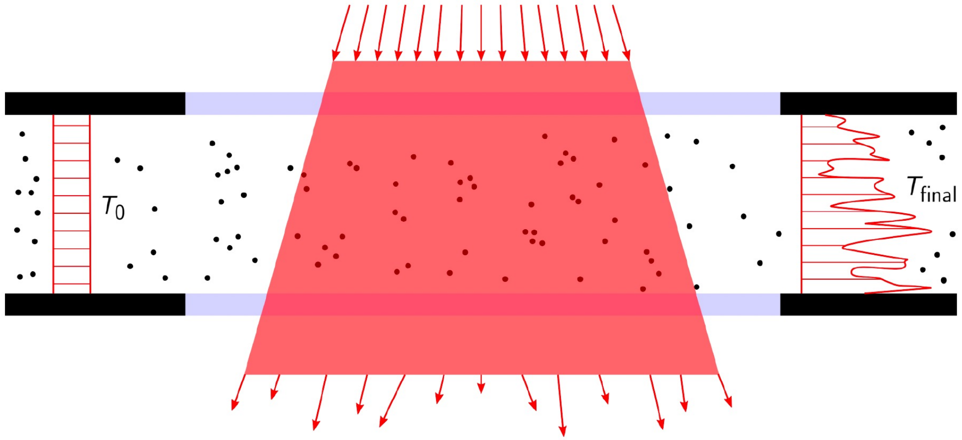


Observations

- More productive for domain scientists to write code
  - They don't deal with parallelism, synchronization, or data movement
  - Three versions of S3D:
    - S3D Legion C++: 23KLOC
    - S3D Regent: 14KLOC
    - S3D Fortran+MPI: ~100KLOC
- We use different mappings for different chemical reactions
  - More species means more expensive chemistry relative to physics
  - Changes the best way to place data and compute
- Legion's late-binding of performance/mapping decisions is key
  - Rapidly explore the best way to execute the program without code changes

Soleil-X

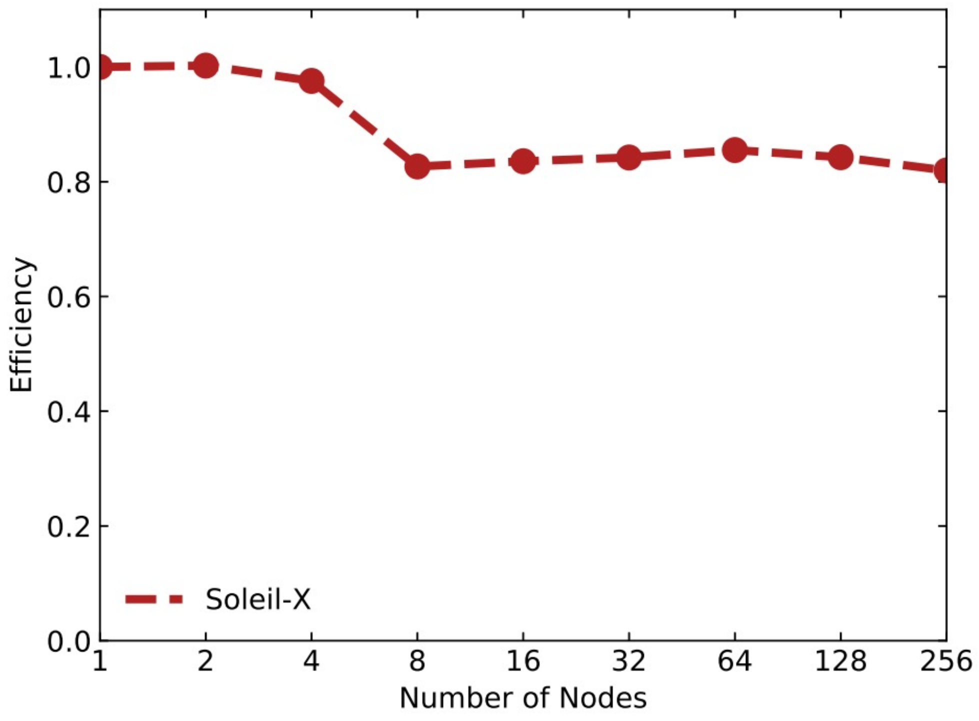
- Solar collector heating nickel particles in a channel
- Multi-physics
  - Fluid, particles & radiation
- Stanford PSAAP II center code
  - All written in Regent



Soleil-X Results

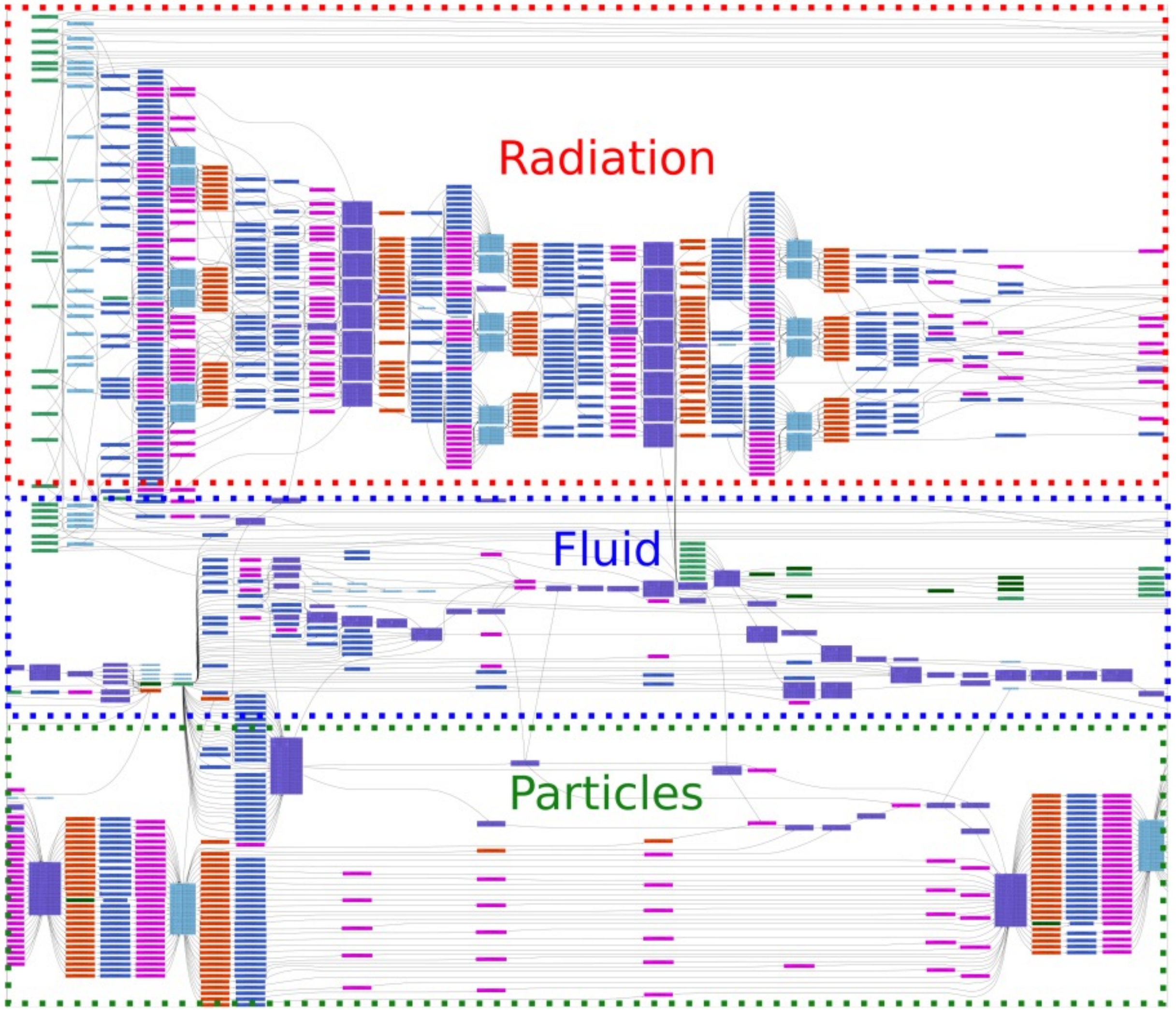
Ported to:

Titan
Summit
Sierra
Lassen
Piz Daint
Certainty (CPU only)
Sherlock



Weak scaling on Sierra

Soleil-X Task Graph



One timestep on one node

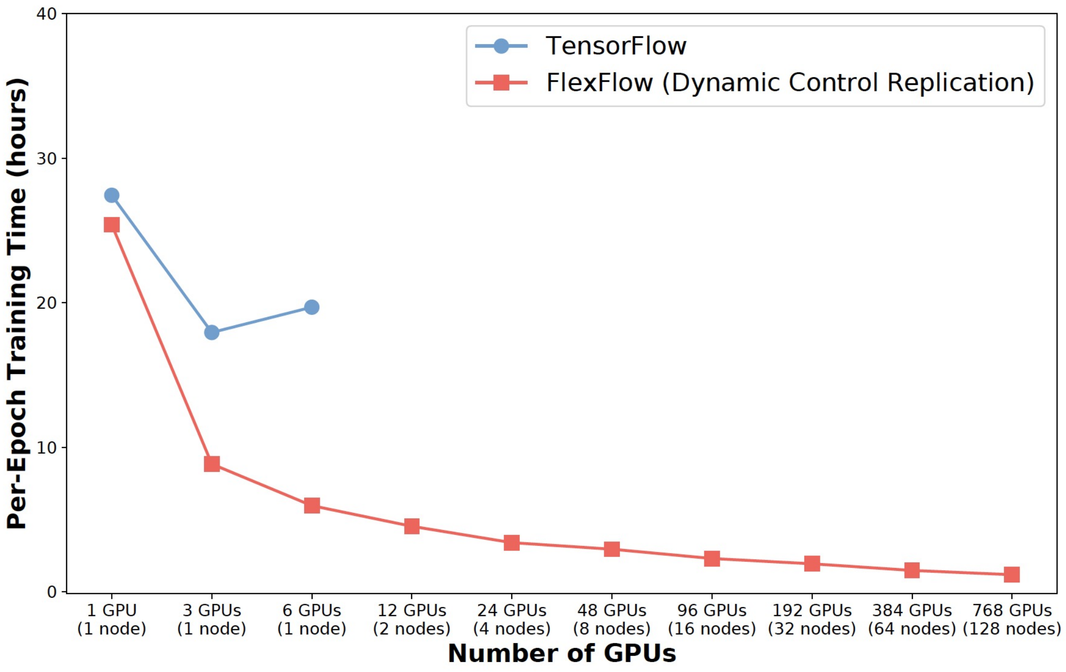
FlexFlow: Deep Neural Networks

- In deep learning, data is commonly organized as tensors.
  - tensor = [image, height, width, channel]

- Existing tools parallelize in one of two ways
  - In one data dimension (*data parallel)*
  - By dividing up the operations across compute resources (*model parallel)*

- Allow each layer to be parallelized differently in any dimensions
  - Exploits Legion's expressive data partitioning

Deep Learning: The Candle Project (ECP)

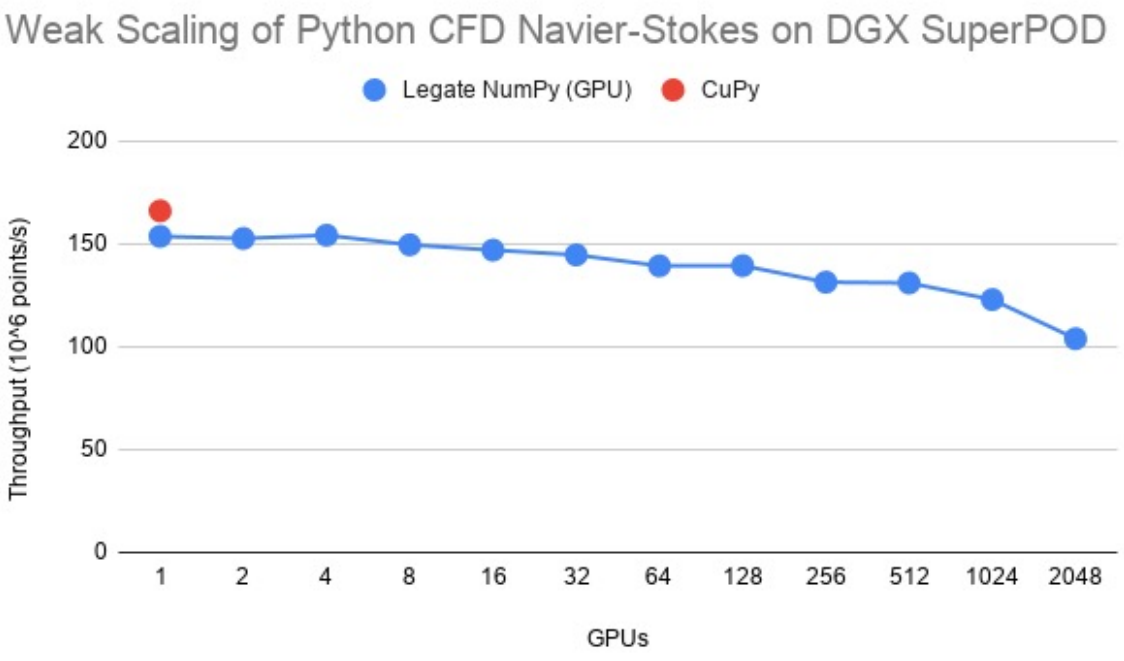


- Training the model on Summit
- TensorFlow's data parallel strategy does not scale past 1 node/6 GPUs
- FlexFlow finds a different parallelization strategy that scales to 128 nodes/768 GPUs

Legate

- Parallel/accelerated/distributed support for Python
  - Built on Legion
  - Provide access to supercomputers to people with little to no HPC background
- Drop-in replacement for
  - NumPy
  - Pandas
- Idea: Automatically partition NumPy arrays and create tasks for NumPy operators
  - Relies on Legion's partitioning support and dynamic task graph creation
- Developed by NVIDIA
  - Open source, as with all Legion packages

CFD Solver in Legate



- Unmodified CFD solver taken from Lorena Barba's CFD Python course
    - ~200 lines of NumPy

- Achieves good weak scaling out to 2,048 A100 GPUs

Important ideas

- Important ideas
  - First class data partitioning
  - Dynamic task graph construction
  - Late binding of performance decisions (e.g., mapping)
  - Compositional model supports writing libraries

- Applications at scale in
  - Simulation
  - Deep learning and data analytics

Other, Current and Future Work

Libraries	*Developing libraries that exploit Legion capabilities*
Interoperability	*Support for interop with Fortran, C++, Python, MPI*
Exascale machines	*Runs on early Frontier and Aurora hardware today*
Kernel support	*Write CPU/GPU portable kernels in Kokkos, (a subset of) OpenMP, and Regent*
Automapping	*Working on automating the mapping process*

And more … some interesting aspects of Legion omitted for lack of time… Full list of project's publications available on-line: https://legion.stanford.edu/publications/

Questions?

Legion: legion.stanford.edu

Regent: regent-lang.org

FlexFlow: flexflow.ai

Legate: github.com/nv-legate/