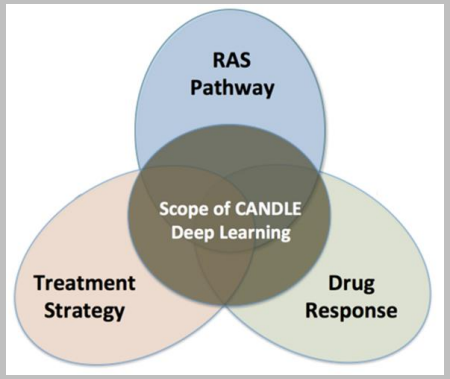
OPTIMAL DEEP LEARNING ON EXASCALE COMPUTERS

Argonne NATIONAL LABORATORY

AN INTRODUCTION TO SCALABLE DEEP LEARNING WORKFLOWS WITH CANDLE



JUSTIN M WOZNIAK
Computer Scientist
Mathematics & Computer Science
Argonne National Laboratory

CANDLE Deep Learning Workshop @ NIH
February 22, 2018

COLLABORATORS

- CANDLE Infrastructure:
  Tom Brettin, Jon Ozik, Nick Collier, Rajeev Jain (ANL)
  Jamal Mohd-Yusof, Cristina Garcia Cardona (LANL)
  George Zaki (NIH)

- Pilot benchmarks
  Fangfang Xia (ANL), Brian Van Essen (LLNL), Arvind Ramanathan (ORNL)

- PI
  Rick Stevens (ANL)

OUTLINE

- Overview of CANDLE project

- Overview of hyperparameter optimization
  - Introduction to hyperparameter optimization
  - Workflow-based solution: EMEWS

- Afternoon tutorial:
  - Hyperparameter optimization of a CANDLE Benchmark

CANDLE TUTORIALS

- May be found here: `https://github.com/ECP-CANDLE/Tutorials`
  - Subdirectory `2018/NIH`

- See the top-level README to get started with the installation

Argonne NATIONAL LABORATORY

CANDLE OVERVIEW

CANDLE WORKFLOWS: GOALS

- Develop an exascale deep learning environment for cancer
- Building on open source deep learning frameworks and middleware
- Optimization for CORAL and exascale platforms
- Support all three pilot project needs for deep learning – common abstractions
- Collaborate with DOE computing centers, HPC vendors and ECP co-design and software technology projects
- Mission statement: Enable the most challenging deep learning problems in cancer research to run on the most capable supercomputers in the DOE

CANDLE SOFTWARE STACK

Hyperparameter Sweeps, Data Management (e.g. DIGITS, Swift, etc.)	**Workflow**
Network description, Execution scripting API (e.g. Keras, Mocha)	**Scripting**
Tensor/Graph Execution Engine (e.g. Theano, TensorFlow, LBANN-LL, etc.)	**Engine**
Architecture Specific Optimization Layer (e.g. cuDNN, MKL-DNN, etc.)	**Optimization**

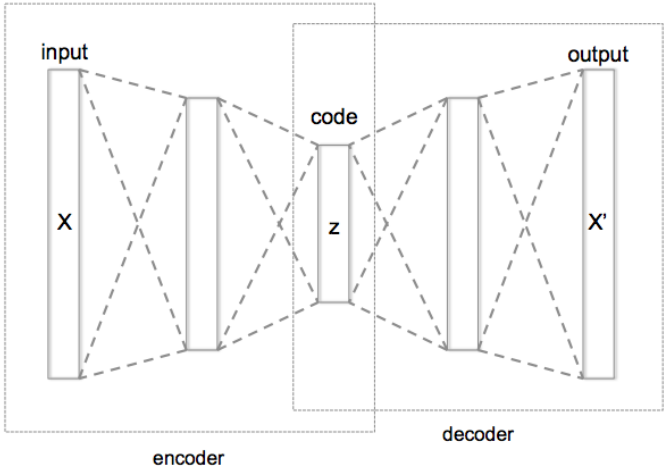
HYPERPARAMETER OPTIMIZATION

WHAT IS HYPERPARAMETER OPTIMIZATION

Hyperparameter optimization = HPO

- Neural networks have a large number of possible configuration parameters, called *hyperparameters*
  - Avoids collision with NN *weights*, which are sometimes called *parameters*
- Applying optimization can automate part of the design of the neural network

- In the cancer Pilot 1 autoencoder shown, the system can determine
  - How many neurons to put in each layer
  - What activation function to use
  - What batch size to use
  - Etc.



MATHEMATICAL EXPRESSION FOR HPO

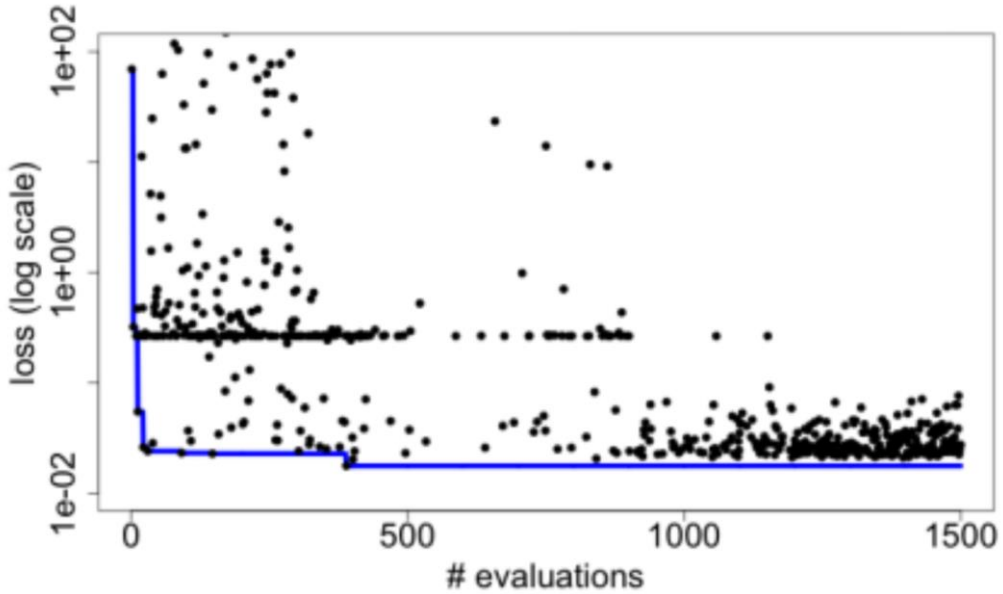
- For a given problem:
  - A loss function ***F*** is determined on a given NN (usually accuracy)
  - The hyperparameter optimization problem is to minimize ***F(p)***,
    - for all hyperparameter sets ***p*** in the valid parameter space ***P***,
    - however, ***P*** is large and ***F*** is expensive.
    - ***P*** is the cross product of all valid network settings,
      - some of which may be categorical, some integer, some continuous.
    - Evaluating ***F*** involves training the network on a training data set and applying it to the validation set
- We can use a generic, previously developed method to optimize ***F*** !
- These methods require and can use large compute resources

BASIC STRATEGIES

- Grid search
- Random search
- Generic optimization
  - Stochastic gradient descent
  - Evolutionary algorithms
  - Model-based optimization (mlrMBO in R)
- NN hyperparameter-specific optimization
  - Hyperopt, NEAT, Optunity, …

CANDLE HYPERPARAMETER LEARNING

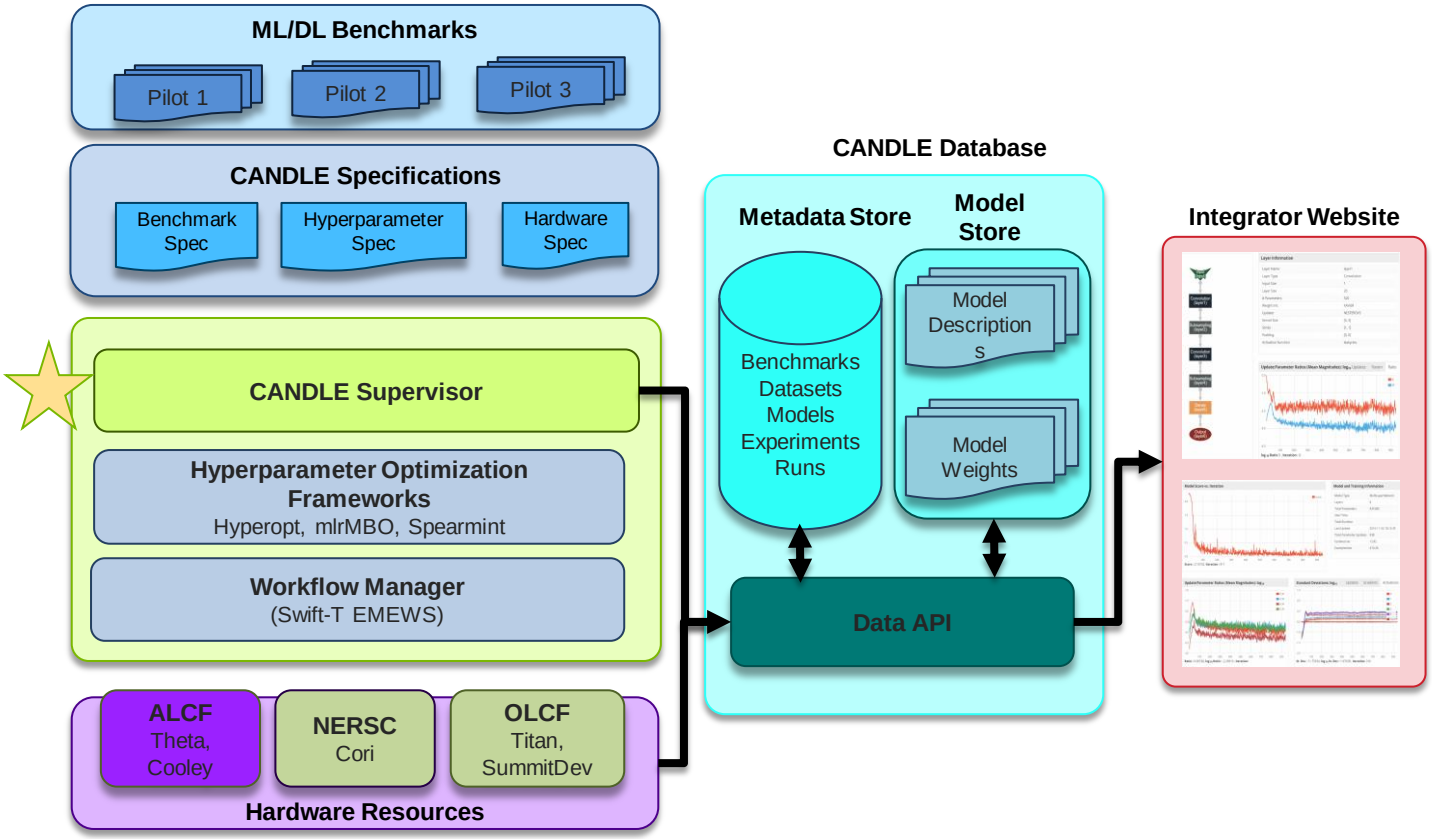
- Search trajectory of mlrMBO (R model-based optimization) algorithm
- Each iteration does 300 evaluations (batch size)
- Minimum and average performance on validation data set decreases as the ME algorithm learns



Predicting Tumor Cell Line Response to Drug Pairs with Deep Learning, F. Xia, M. Shukla, T. Brettin, C. Garcia-Cardona, J. Cohn, J. Allen, S. Maslov, Y. Evrard, S. Holbeck, J. Doroshow, E. Stahlberg, and R. Stevens (Computational Approaches for Cancer Workshop @ SC 2017)

CANDLE: WORKFLOWS

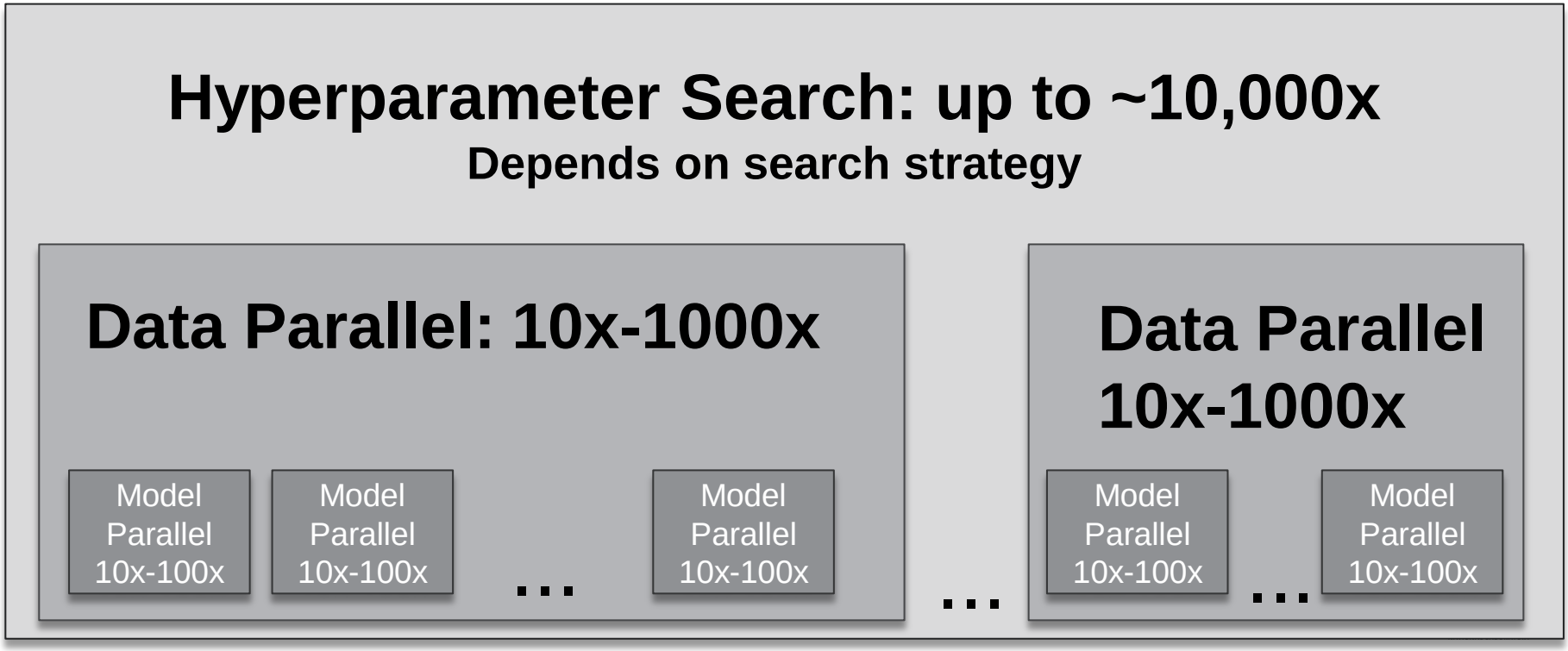
CANDLE SYSTEM OVERVIEW



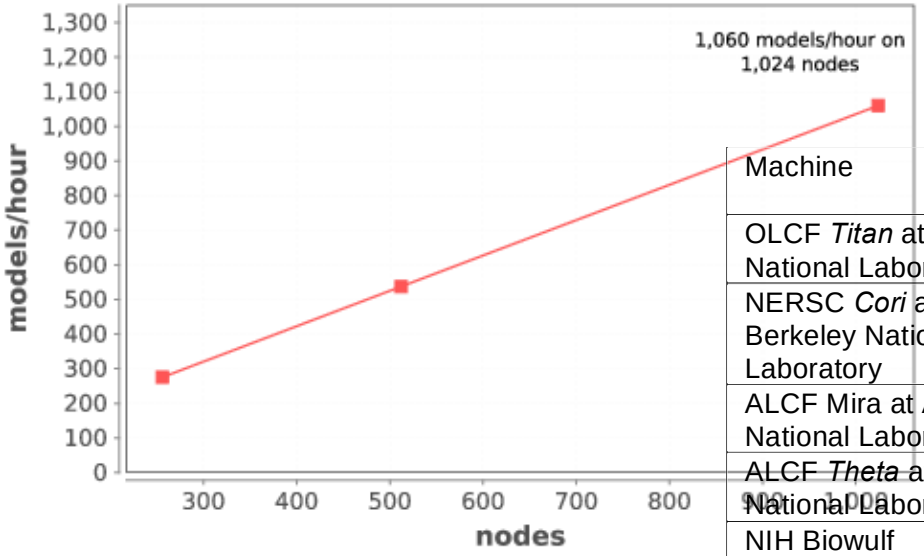
Argonne NATIONAL LABORATORY

PARALLELISM STRATEGIES

10,000 x 10-1000 x 10-100 = 1M – 1000M processing elements



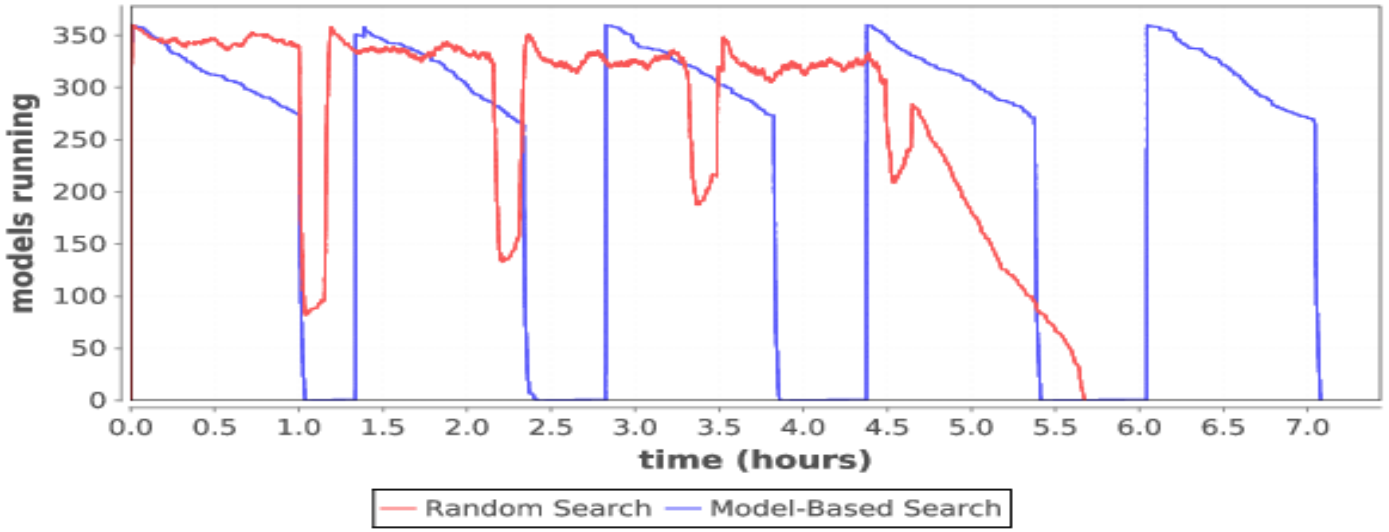
CANDLE PERFORMANCE



- Delivers 1+ petaflop!

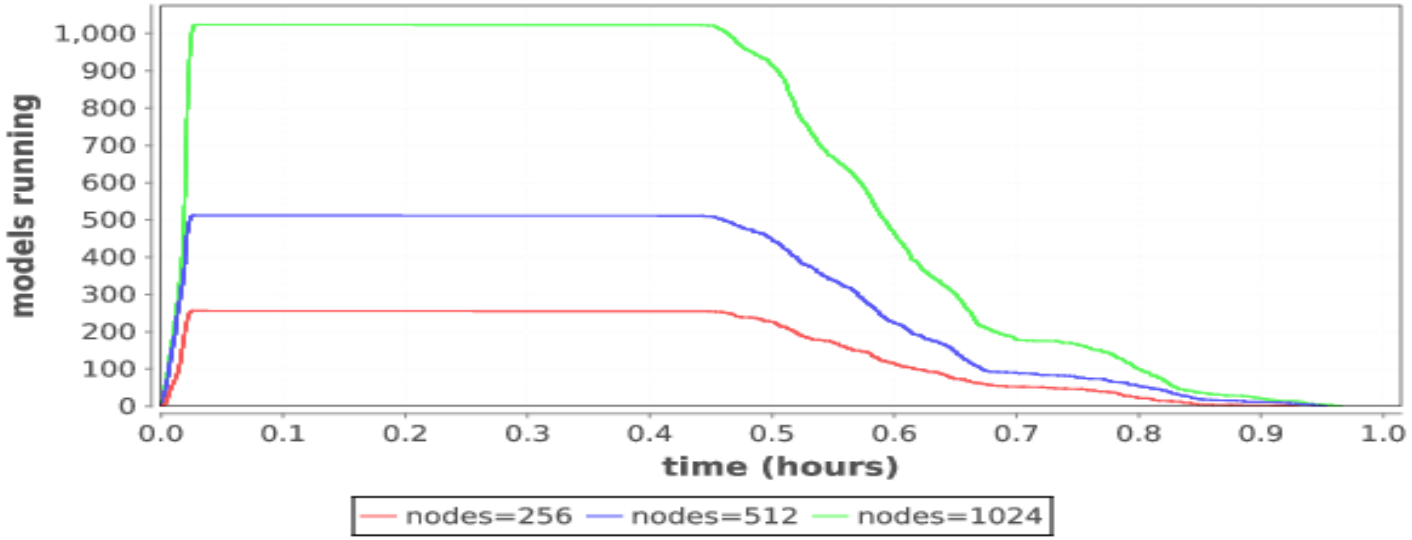
Machine	Architecture (Nov 2017 Top 500 ranking)	Scheduler
OLCF *Titan* at Oak Ridge National Laboratory	Cray XK7, AMD Opteron / K20X (5)	PBS
NERSC *Cori* at Lawrence Berkeley National Laboratory	Cray XC40, Intel Xeon Phi / E5 (8)	SLURM
ALCF Mira at Argonne National Laboratory	BlueGene/Q, PowerPC A2 (11)	Cobalt
ALCF *Theta* at Argonne National Laboratory	Cray XC40, Intel Xeon Phi (18)	Cobalt
NIH Biowulf	HP Apollo XL1x0r, Intel Xeon E5 (66)	SLURM
Blues, Bebop at Argonne National Laboratory	Intel Xeon Phi / E5 (148)	PBS/SLURM
Beagle2 at University of Chicago	Cray XE6, AMD Operton (NA)	PBS
Midway, Midway2 at University of Chicago	Intel various / AMD Opteron (NA)	SLURM

LOAD OVER TIME FOR SEARCH



- Typical load plot for NT3 workflow on Cori

RAMP UP / RAMP DOWN



- Zoom in on single iteration on Titan

SYSTEMS CHALLENGES

NOTES ON REQUIREMENTS

Simulation Applications

- **64bit floating point**
- Memory Bandwidth
- Random Access to Memory
- Sparse Matrices
- **Distributed Memory jobs**
- Synchronous I/O multinode
- Scalability Limited comm
- Low Latency High Bandwidth
- Large Coherency Domains help sometimes
- **O typically greater than I**
- **O rarely read**
- Output is data

Big Data Applications

- 64 bit and Integer important
- Data analysis Pipelines
- **DB including No SQL**
- **MapReduce/SPARK**
- Millions of jobs
- I/O bandwidth limited
- Data management limited
- **Many task parallel**
- Large-data in and Large-data out
- I and O both important
- O is read and used
- Output is data

Deep Learning Applications

- **Lower Precision (fp32, fp16)**
- FMAC @ 16 summing to 32
- Inferencing can be 8 bit (TPU)
- Scaled integer possible
- Training dominates dev
- Inference dominates pro
- Data pipelines needed
- Dense FP typical SGEMM
- Small DFT, CNN
- **Ensembles and search**
- Single Models Small
- **I more important than O**
- **Reuse of training data**
- **Output is models**

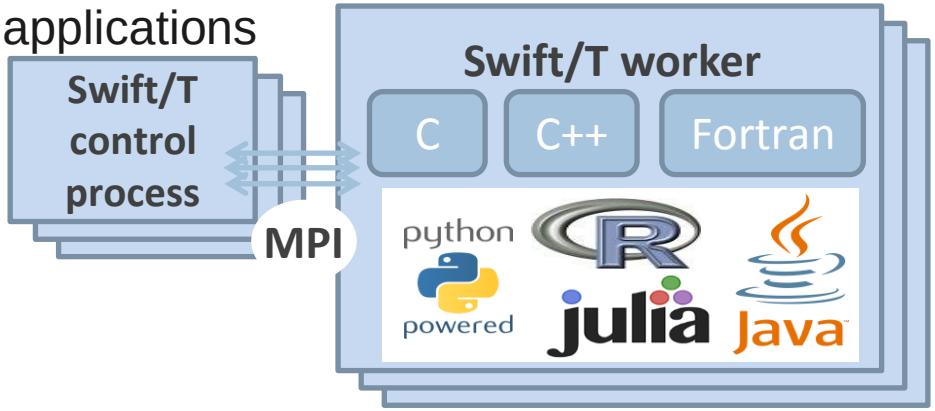
WORKFLOW SUPPORT FOR ML FRAMEWORKS

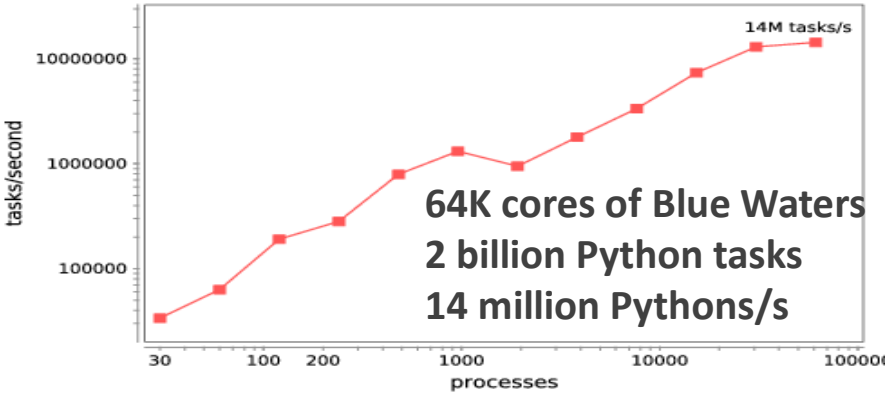
- Concurrency:
  - Scalable task distributor
  - Intranode concurrency, accelerators left up to the framework
  - Multinode ML tasks are future work (already basically supported)
- Data management:
  - Input staging methods have been developed
  - Intermediate caches via DataSpaces
- Software integration:
  - Usually launch frameworks in separate process
  - Launching within process is a configuration challenge
  - Search methods launched within process

SWIFT/T: ENABLING HIGH-PERFORMANCE SCRIPTED WORKFLOWS

Supports tasks written in many languages

- Write site-independent scripts
- Automatic parallelization and data movement
- Run native code, script fragments as applications
- Rapidly subdivide large partitions for MPI jobs





- **Interlanguage parallel scripting for distributed-memory scientific computing.** Proc. WORKS @ SC 2015

Argonne NATIONAL LABORATORY

THE SWIFT PROGRAMMING MODEL

All progress driven by concurrent dataflow

```
(int r) myproc (int i, int j)
{
    int x = F(i);
    int y = G(j);
    r = x + y;
}
```

- `F()` and `G()` implemented in native code or external programs
- `F()` and `G()` run in concurrently in different processes
- `r` is computed when they are both done
- This parallelism is *automatic*
- Works recursively throughout the program's call graph

LANGUAGE GOALS

Hierarchical, naturally parallel, script-like programming

- Make it easy to run large batteries of external program or library executions
- Provide rich programming language at the top level – fully generic
- Support implicit concurrency and conventional programming constructs
- Enable complex tasks based in other scripting languages (e.g., Python) or parallel MPI tasks

SWIFT SYNTAX

- Data types

```
int i = 4;
string s = "hello world";
file image<"snapshot.jpg">;
```

- Shell access

```
app (file o) myapp(file f, int i)
{ mysim  "-s" i @f @o; }
```

- Structured data

```
typedef image file;
image A[];
type protein_run {
  file pdb_in; file sim_out;
}
bag<blob>[] B;
```

- Conventional expressions

```
if (x == 3) {
    y = x+2;
    s = strcat("y: ", y);
}
```

- Parallel loops

```
foreach f,i in A {
    B[i] = convert(A[i]);
}
```

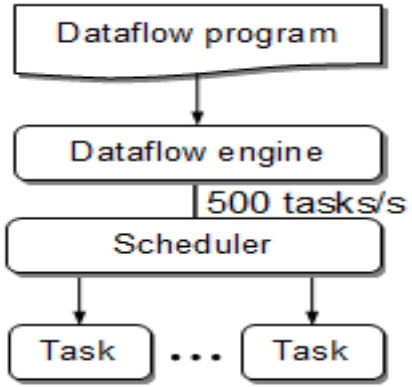
- Data flow

```
merge(analyze(B[0], B[1]),
      analyze(B[2], B[3]));
```

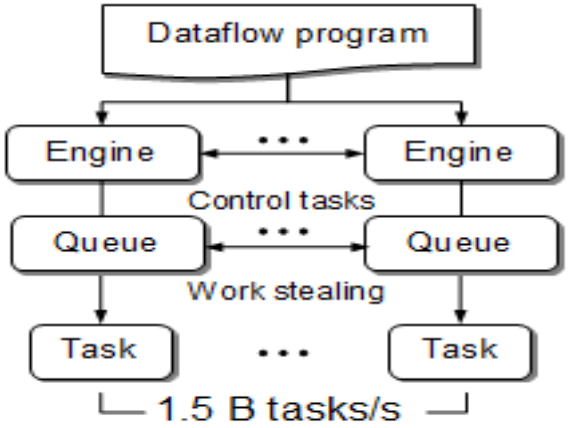
- **Swift: A language for distributed parallel scripting.** J. Parallel Computing 2011
- **Compiler techniques for massively scalable implicit task parallelism.** Proc. SC 2014

CENTRALIZED EVALUATION IS A BOTTLENECK AT EXTREME SCALES

Had this (Swift/K):



Now have this (Swift/T):

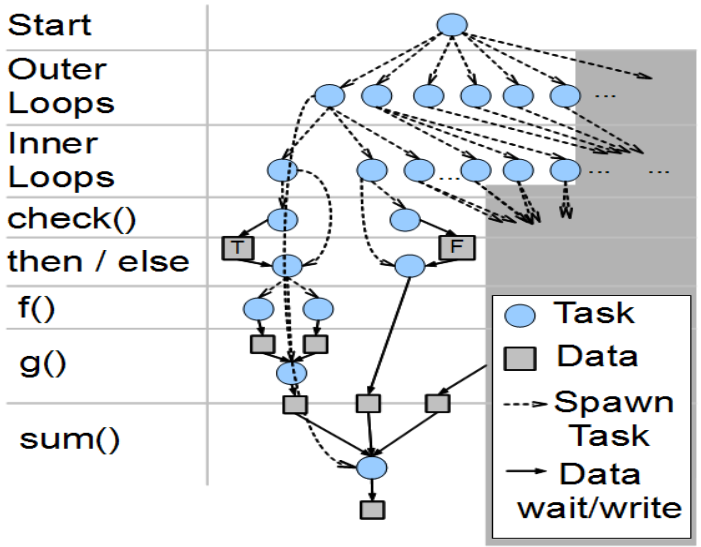


Centralized evaluation | Distributed evaluation

- **Turbine: A distributed-memory dataflow engine for high performance many-task applications.** Fundamenta Informaticae 28(3), 2013

SWIFT/T: FULLY PARALLEL EVALUATION OF COMPLEX SCRIPTS

```
int X = 100, Y = 100;
int A[][];
int B[];
foreach x in [0:X-1] {
  foreach y in [0:Y-1] {
    if (check(x, y)) {
      A[x][y] = g(f(x), f(y));
    } else {
      A[x][y] = 0;
    }
  }
  B[x] = sum(A[x]);
}
```



- **Swift/T: Scalable data flow programming for distributed-memory task-parallel applications .** Proc. CCGrid, 2013.

Argonne NATIONAL LABORATORY

SWIFT FOR REALLY PARALLEL BUILDS

Plus language features- typed files, arrays, string processing

App definitions

```
app (object_file o) gcc(c_file c, string cflags[])
{
// Example:
//  gcc   -c   -O2    -o  f.o f.c
   "gcc" "-c" cflags "-o" o   c;
}

app (x_file x) ld(object_file o[], string ldflags[])
{
// Example:
//  gcc           -o  f.x f1.o f2.o ...
   "gcc" ldflags "-o" x   o;
}

app (output_file o) run(x_file x)
{
  "sh" "-c" x @stdout=o;
}

app (timing_file t) extract(output_file o)
{
  "tail" "-1" o "|" "cut" "-f" "2" "-d" " " @stdout=t;
}
```

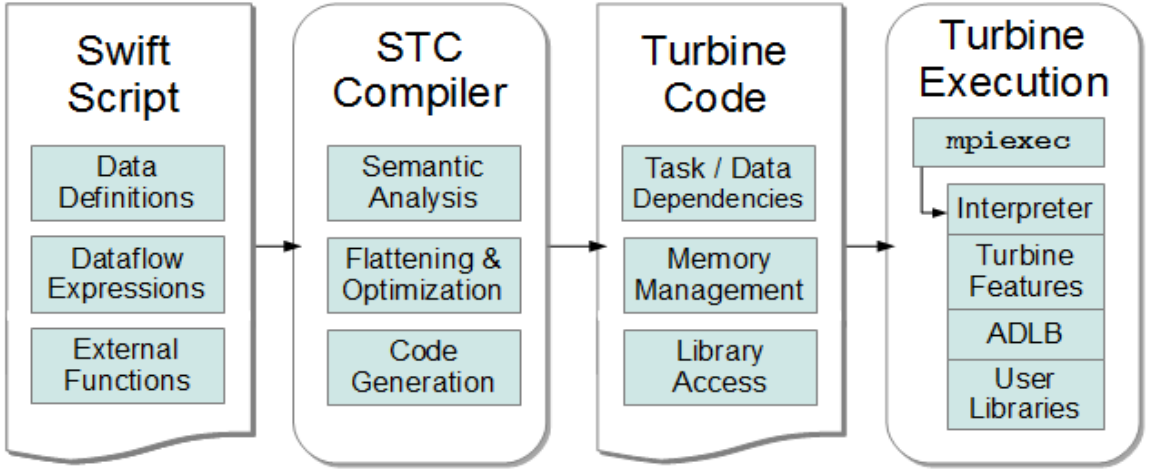
Swift code

```
string program_name = "programs/program1.c";
c_file c = input(program_name);

foreach O_level in [0:3]
{
  // Construct the compiler flags
  string O_flag = "-O" + O_level;
  string cflags[] = [ "-fPIC", O_flag ];

  object_file o<my_object> = gcc(c, cflags);
  object_file objects[] = [ o ];
  string ldflags[] = [];
  // Link the program
  x_file x<my_executable> = ld(objects, ldflags);
  // Run the program
  output_file out<my_output> = run(x);
  // Extract the run time from the program output
  timing_file t<my_time> = extract(out);
}
```

SWIFT/T COMPILER AND RUNTIME



- STC translates high-level Swift expressions into low-level Turbine operations:
  - Create/Store/Retrieve typed data
  - Manage arrays
  - Manage data-dependent tasks

Argonne NATIONAL LABORATORY

ACCESSING PYTHON FROM SWIFT/T

```
global const string numpy = "from numpy import *\n\n";
typedef matrix string;

(matrix A) eye(int n) {
  command = sprintf("repr(eye(%i))", n);
  code = numpy+command;
  matrix t = python(code);
  A = replace_all(t, "\n", "", 0);
}

(matrix R) add(matrix A1, matrix A2) {
  command = sprintf("repr(%s+%s)", A1, A2);
  code = numpy+command;
  matrix t = python(code);
  R = replace_all(t, "\n", "", 0);
}
```

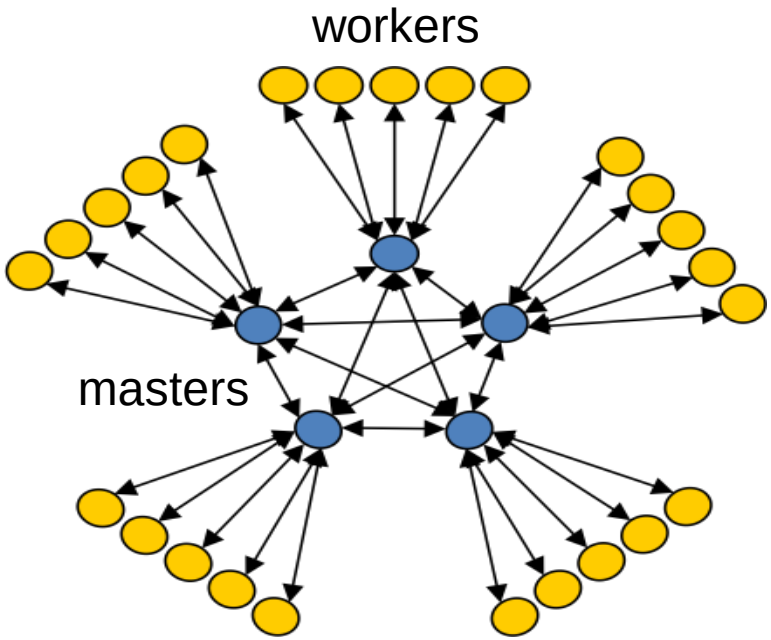
```
a1 = eye(3);
a2 = eye(3);
sum = add(a1, a2);
printf("2*eye(3)=%s", sum);
```

ASYNCHRONOUS DYNAMIC LOAD BALANCER

ADLB for short

- An MPI library for master-worker workloads in C
- Uses a variable-size, scalable network of servers
- Servers implement work-stealing
- The work unit is a byte array
- Optional work priorities, targets, types

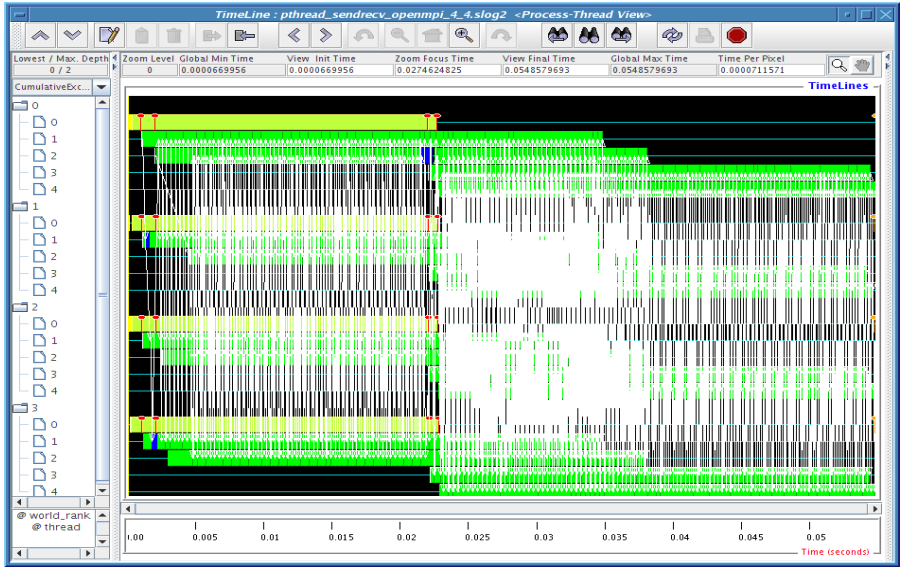
- For Swift/T, we added:
  - Server-stored data
  - Data-dependent execution
  - Parallel tasks



- Lusk et al. **More scalability, less pain: A simple programming model and its implementation for extreme computing.** SciDAC Review 17, 2010

MPI: THE MESSAGE PASSING INTERFACE

- Programming model used on large supercomputers
- Can run on many networks, including sockets, or shared memory
- Standard API for C and Fortran; other languages have working implementations
- Contains communication calls for
  - Point-to-point (send/recv)
  - Collectives (broadcast, reduce, etc.)
- Interesting concepts
  - Communicators: collections of communicating processing and a context
  - Data types: Language-independent data marshaling scheme
  - Can recursively create subordinate MPI contexts in a variety of ways



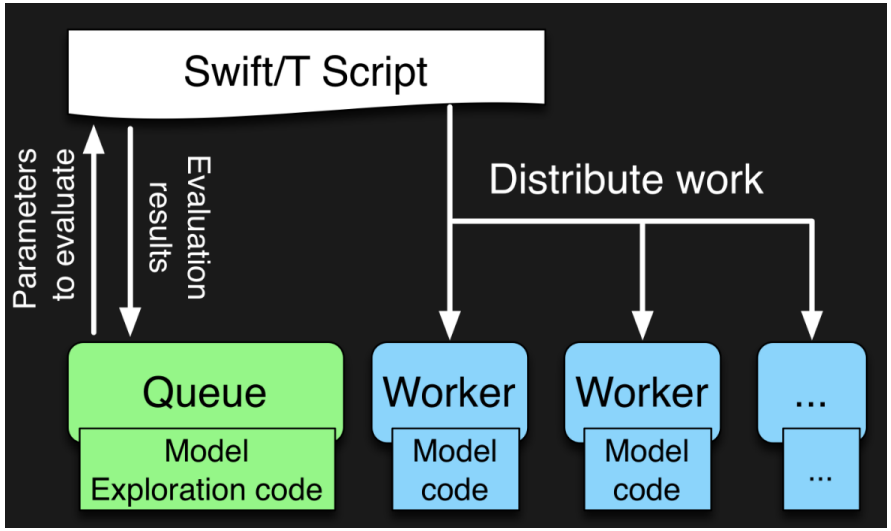
PARALLEL TASKS IN CANDLE WORKFLOWS

Complex concurrency structures

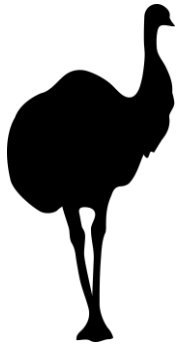
- Model parallelism: running the same network across nodes
- Library approach:
  - Use Swift/T @par syntax
  - Uses MPI 3 to dynamically create communicator from group
  - User task library accepts communicator via function input
  - Approach developed for other scientific computing cases, LAMMPS, NAMD, DIY, etc.
- MPI_Launch approach
  - Use Swift/T launch() function
  - Creates MPI 3 group
  - Launches mpiexec on those resources, creating a new MPI_COMM_WORLD and separate processes (fault tolerance)
  - Works on clusters
  - Working with Cray on support – available soon

EXTREME-SCALE MODEL EXPLORATION WITH SWIFT (EMEWS)

EMEWS WORKFLOW STRUCTURE

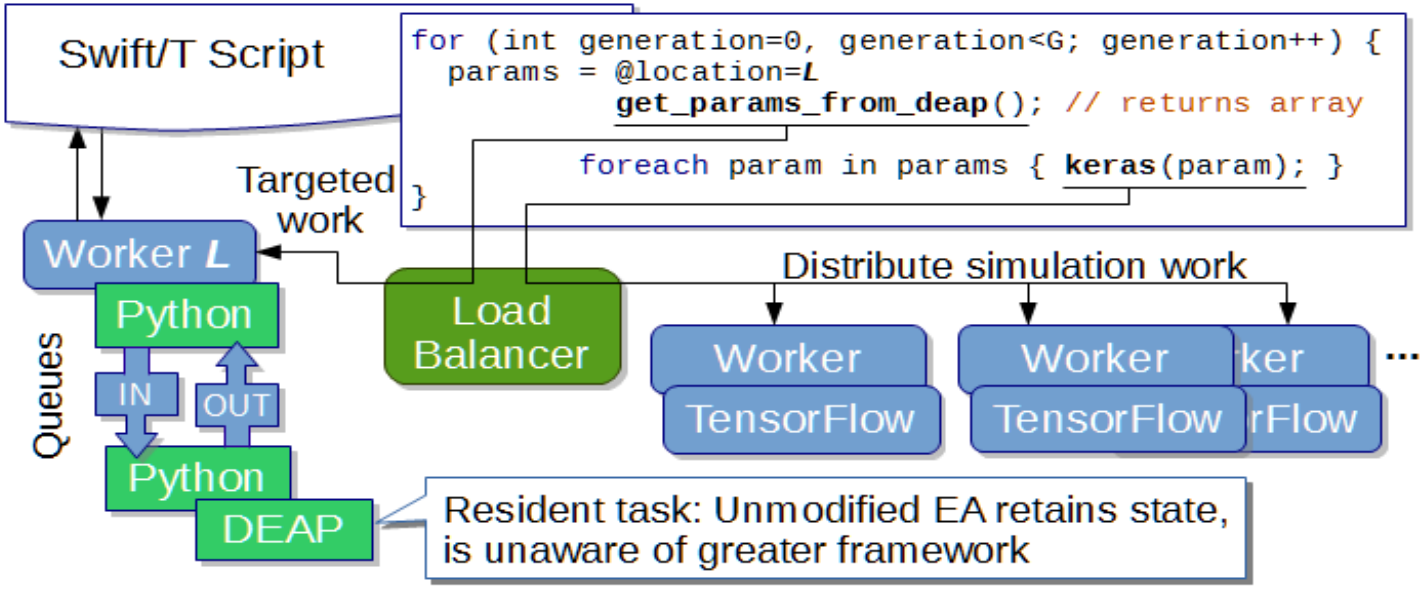


- The core novel contributions of EMEWS are shown in green, these allow the Swift script to access a running **Model Exploration (ME)** algorithm, and create an **inversion of control (IoC)** workflow
- Both green and blue boxes accept **existing multi-language code**



EMEWS: EXTREME-SCALE MODEL EXPLORATION WORKFLOWS IN SWIFT/T

- To query the state of the EA, we designate one worker on location L for exclusive use by DEAP. Other optimizers can easily be used (e.g., mlrMBO in CANDLE)



Argonne NATIONAL LABORATORY

PREVIOUS WORK ON HPC WORKFLOWS

Other uses of workflows to control model exploration (ME) typically take one of two approaches

1. They provide rich support for arithmetic operations so that ME algorithms can be constructed (ported)
   - requires that algorithm be **coded from scratch**
   - **impossible to reuse code** in other languages
2. The ME algorithm is provided as a built-in feature of the system
   - does not allow the end users much **control over the algorithm** used
   - may require **access to workflow system source code** in order to incorporate external ME algorithms or to modify built-in algorithms

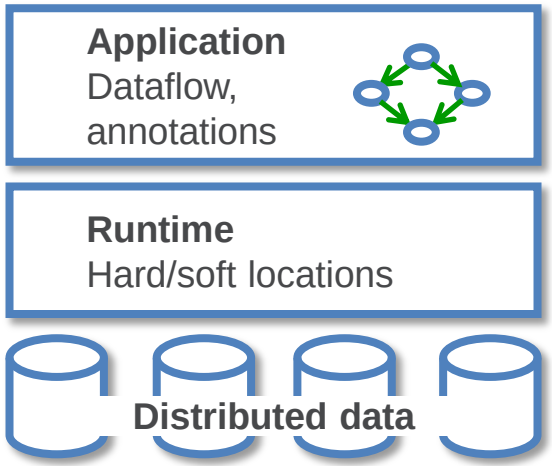
In both cases, the many libraries being actively developed and implemented as free and open source software in programming languages such as R and Python **cannot be directly/easily utilized**.

SUMMARY OF KEY SYSTEM POINTS

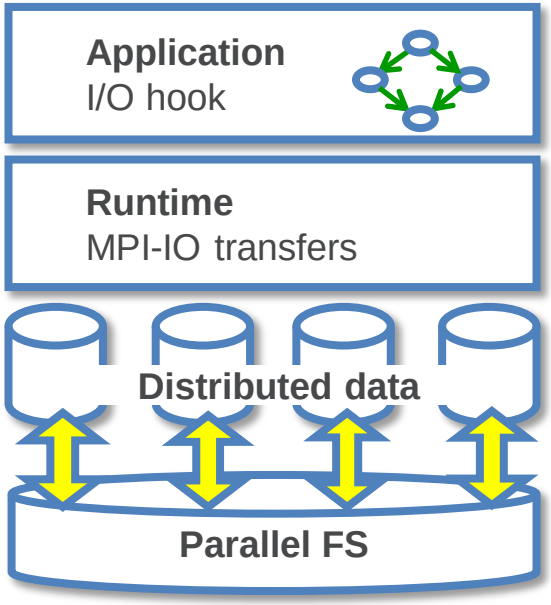
- What about Swift/T enables CANDLE?
  - A workflow system that is actually a hierarchical programming language
  - Runs entirely on the compute nodes
  - Uses standard APIs for HPC (MPI), allows for minimal OS environment
  - Very scalable
  - Supports MPI tasks, embedded Python, R interpreters
- What about EMEWS enables CANDLE?
  - Allows user to focus on two sequential codes
    - The optimizer
    - Their objective function code
  - Everything else is managed by the system

FEATURES FOR BIG DATA ANALYSIS

- **Location-aware scheduling**
  User and runtime coordinate data/task locations



- **Collective I/O**
  User and runtime coordinate data/task locations

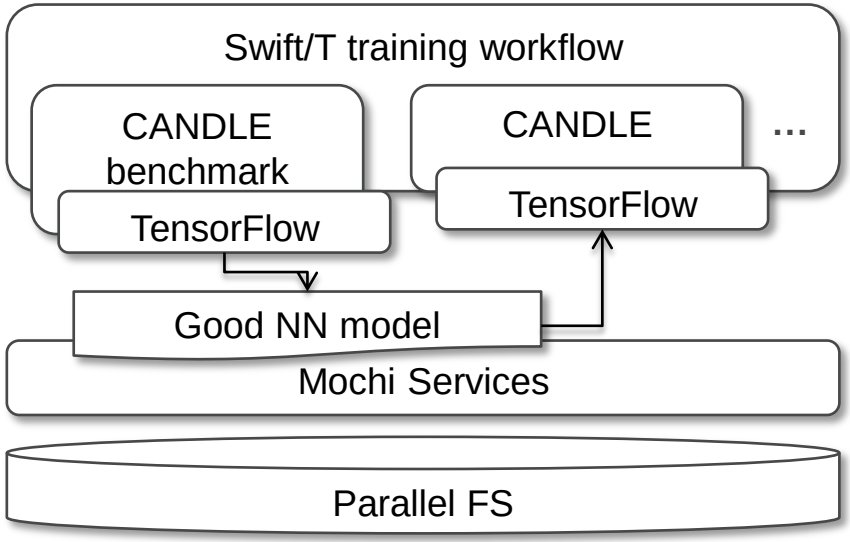


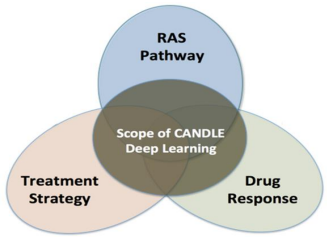
- F. Duro et al**. Flexible data-aware scheduling for workflows over an in-memory object store**. Proc. CCGrid, 2016.

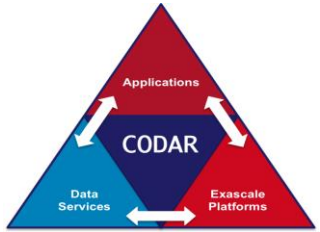
INTERACTION WITH ECP CODAR, ECP CANDLE

With Justin M. Wozniak (CANDLE) and Tong Shu (CODAR)

- **CANDLE** workflows produce a great number of medium-sized ML models
- **Goal:** Cache these on compute node storage for *possible* later use
- Need to flush to global FS before end of run, but many models will be discarded
- **Plan:** Integrate Swift/T workflow system used in CANDLE with Mochi client
- Accelerate CANDLE workflow performance, enable novel training strategies (parameter sharing)
- Provide an opportunity for workflow-based data analysis and I/O reduction
- Demonstrate the utility of node-local storage for complex workflows







THANKS

- Thanks to the organizers

- Code and guides:
  - CANDLE GitHub Organization: https://github.com/ECP-CANDLE
  - Swift/T Home: http://swift-lang.org/Swift-T
  - EMEWS Tutorial: http://emews.org

- This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

TUTORIAL: SUPERVISOR

Argonne NATIONAL LABORATORY

HANDS-ON TUTORIAL: SUPERVISOR

- May be found here: `https://github.com/ECP-CANDLE/Tutorials` directory `2018/NIH`

- See the top-level README to get started with the installation