Tutorial: Analytics and AI on Cray Systems

CUG 2019

Mustafa Mustafa (NERSC), Kristyn Maschhoff, Mike Ringenburg (Cray)

Mustafa Mustafa <mmustafa@lbl.gov>

Kristyn Maschhoff <kristyn@cray.com>

Mike Ringenburg <mikeri@cray.com>

Agenda

- **Session I: 8:30-10:00**
  - Welcome and Introductions
  - Urika XC and CS overview (Mike)
  - Introduction to Deep Learning (Mike)
  - Hyperparameter Optimization (Kristi)
- **Break: 10:00-10:30**
- **Session II: 10:30-12:00**
  - Distributed Deep Learning (Mustafa)
  - Cray PE ML Plugin Update (Kristi/Pete)
- **Lunch:12:00-1:00**
- **Session III: 1:00-2:30**
  - Hands-on Deep Learning (Mustafa)
  - Spark and Dask (Mike)
- **Break: 2:30-3:00**
- **Session IV: 3:00-4:30**
  - R and pbdR (Kristi)
  - Cray Graph Engine (Kristi)
  - Hands-on: HPO, R, Cray Graph Engine (Kristi)

Don’t miss the Data Science Bird of a Feather Session right after this tutorial!

Tools and Utilities for Data Science Workloads and Workflows

Speakers from KAUST, CSCS, and Cray

Salle Edward C

Monday, 4:40-6:00PM

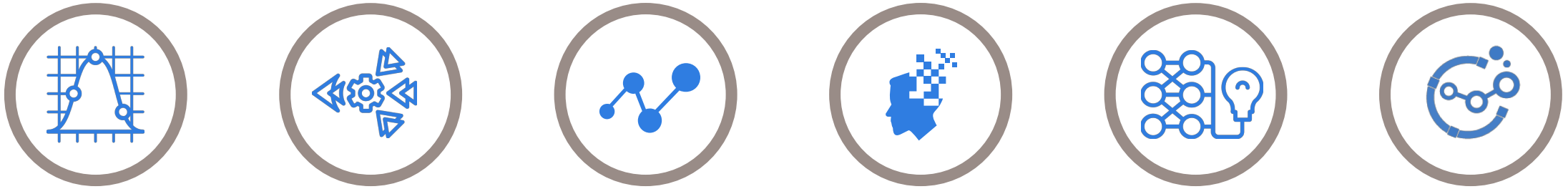
Urika-XC and Urika-CS

CRAY URIKA® SUITES

	CRAY URIKA-XC	CRAY URIKA-CS
Focus	Analytics and AI software suite on a leadership supercomputer platform for Analysis, Machine Learning, Deep Learning and Graph applications	
Scale	Any scale form single node to 1000s of nodes	
Infrastructure	Cray Linux Environment, Centos, Cray Aries, InfiniBand™, Omni Path, Shifter containers (XC), Singularity containers (CS)	
Key Capabilities	• Apache Spark Environment • Anaconda and Dask Python-based Environment • Programming with Big Data in R (pbdR) Distributed Environment • Keras, TensorFlow, PyTorch and BigDL for Deep Learning • Distributed DL Training Frameworks – Horovod, CrayPE ML plugin • Distributed Hyperparameter Optimization (HPO) • Cray Graph Engine (XC only)	

CRAY URIKA SUITES

INTEGRATED AND READY FOR THE CRAY XC AND CS SERIES



Reduce Time and Complexity of AI Environments

Pre-integrated AI software package that is current with evolving AI frameworks/tools

- Right tools for today's data-intensive environments
- Easy to use parallelization for distributed deep learning

Make your researchers and data scientists more productive

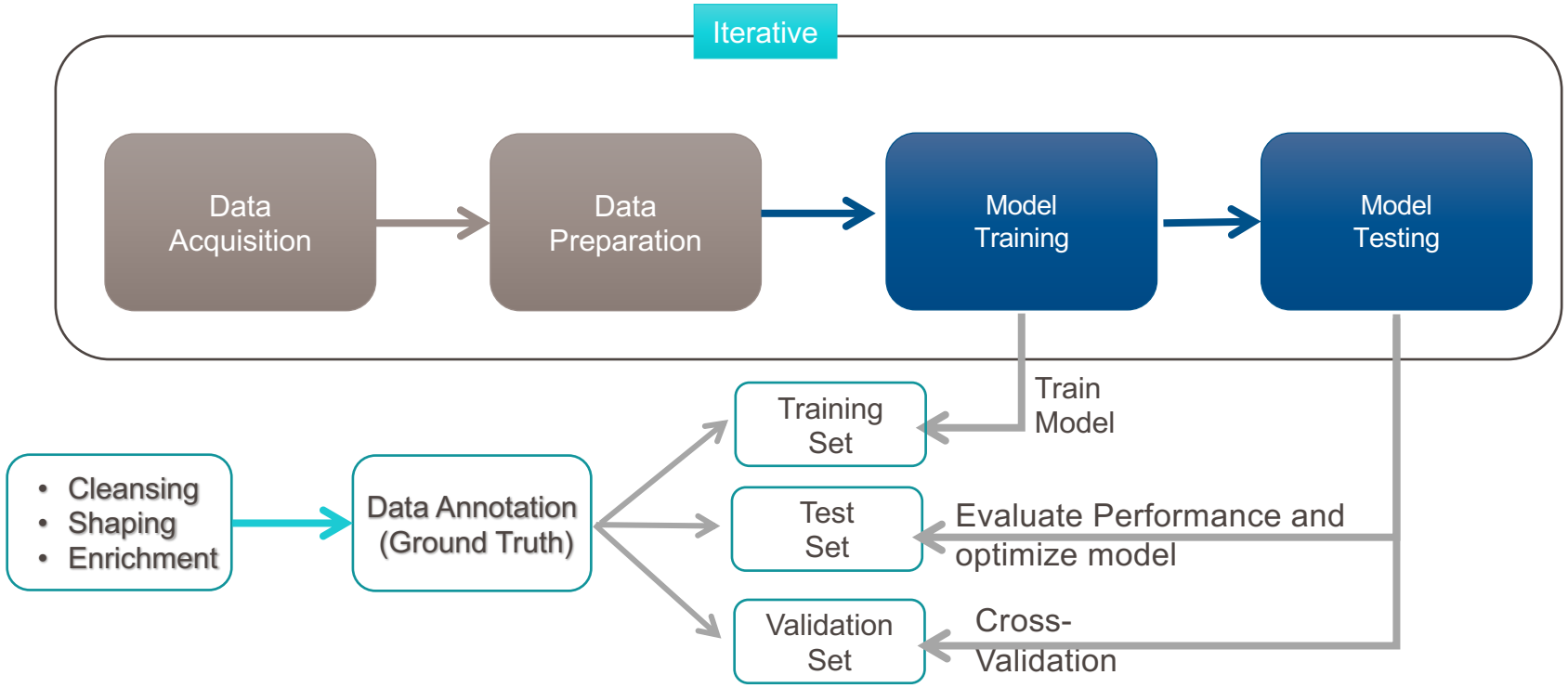
Expand the use of your XC/CS Series Investments

AI and Analytics side-by-side with research and simulation

Urika-XC/CS Focus: Support the Entire AI Workflow

Deep Learning workflows are not limited to training

Similar to other HPC and analytics workloads, significant portions of DL jobs are devoted to data collection, preparation and management.



CRAY URIKA-CS AI & ANALYTICS SUITE

Pre-integrated and supported AI stack with popular open source AI frameworks and libraries delivered in a container for ease of development and deployment

<table>
<tr><td colspan="6">UIs: Jupyter Notebooks, TensorBoard</td></tr>
<tr><td>MLlib,Spark SQL, Spark Streaming, GraphX</td><td>BigDL</td><td rowspan="2">Anaconda Python, Dask</td><td rowspan="2">pbdR</td><td>PyTorch</td><td>Keras, TensorFlow™</td></tr>
<tr><td colspan="2">Apache Spark™</td><td colspan="2">Distributed Training Framework Horovod, CrayPE ML Plugin</td></tr>
<tr><td colspan="6">Intel® MKL, Intel MKL-DNN, OpenMPI</td></tr>
</table>

CRAY CS-STORM

CRAY CS500

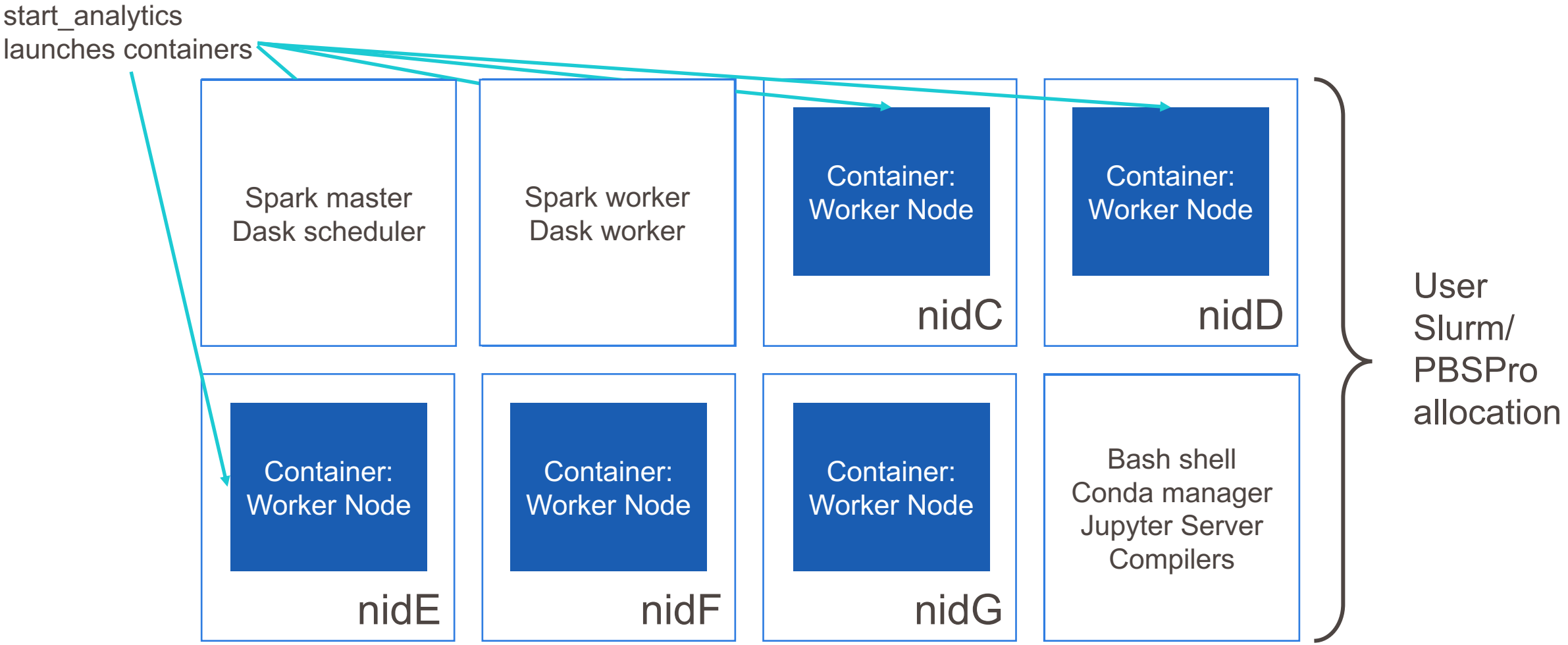
CRAY URIKA-XC AI & ANALYTICS SUITE

Pre-integrated and supported AI stack with popular open source AI frameworks and libraries delivered in a container for ease of development and deployment

<table>
<tr><td colspan="6">UIs: Jupyter Notebooks, TensorBoard</td><td></td></tr>
<tr><td colspan="2">MLlib,Spark SQL, Spark Streaming, GraphX</td><td>BigDL</td><td rowspan="2">Anaconda Python, Dask</td><td rowspan="2">pbdR</td><td>PyTorch</td><td>Keras, TensorFlow™</td><td rowspan="2">Cray Graph Engine (CGE)</td></tr>
<tr><td colspan="3">Apache Spark™</td><td colspan="2">Distributed Training Framework
Horovod, CrayPE ML Plugin</td></tr>
<tr><td colspan="7">Intel® MKL, Intel MKL-DNN, Cray MPI</td></tr>
</table>

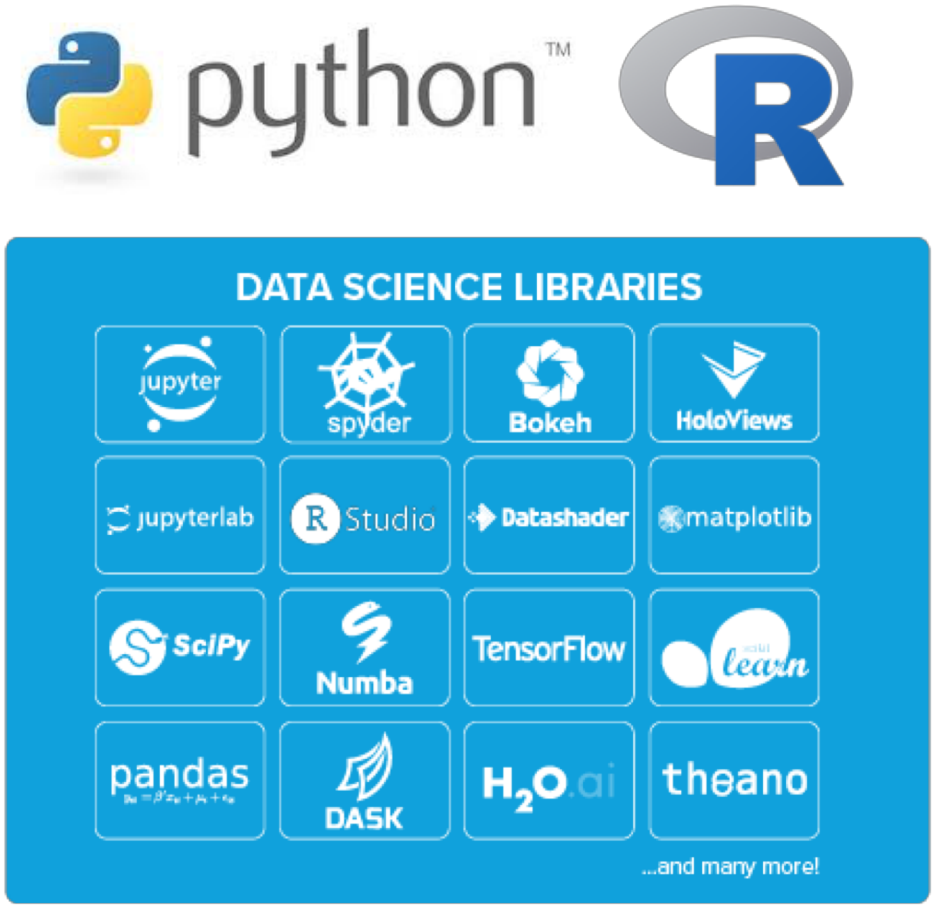
CRAY XC SERIES

EASILY LEVERAGE SYSTEM RESOURCES FOR BIG DATA ANALYTICS & AI



INCLUDES ANACONDA DISTRIBUTION

- Large set of data science packages
  - 250+ Python and R packages preinstalled
  - >1000 available in repositories
  - Many optimized – e.g., work with Intel Python team
- Conda environment manager
  - Linked to conda repos for more Python and R packages
  - Ability to create, clone, share custom environments with your own python/package versions
  - Handles all dependencies
  - Allows sharing environments



OPTIMIZED DISTRIBUTED DEEP LEARNING TRAINING WITH CRAY PE ML PLUGIN

Users can easily achieve ideal scaling performance across multiple DL frameworks, such as TensorFlow, utilizing stochastic gradient descent

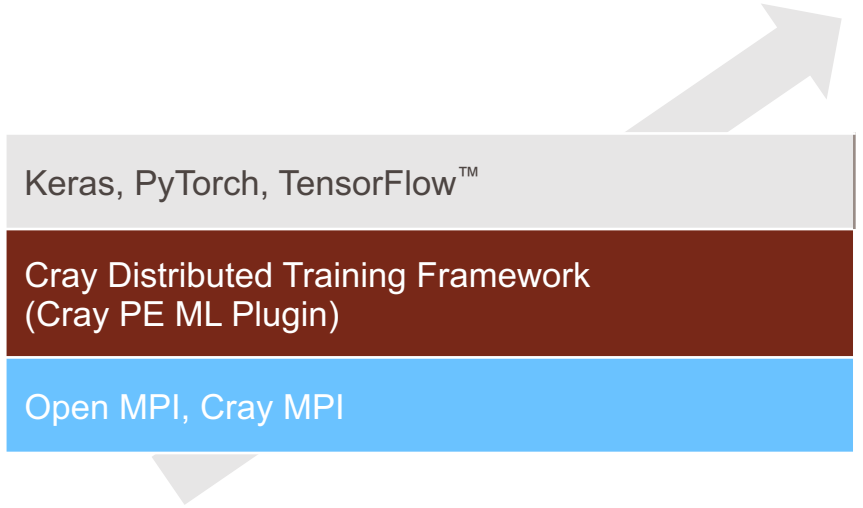
- Load a module
- Plug in few simple lines to the serial Python or C-based training script
- Tunable through API and environment variables
- Scale up training workload to hundreds of nodes on Cray systems

Highly optimized communication plugin tuned for Cray XC and CS Series systems and DL workloads

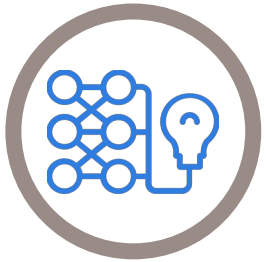
- Perfectly overlaps gradient communication with computation

Delivers high performance and scale for XC and CS Series node architectures

- NVIDIA® Tesla® GPUs, Intel® Xeon® , and AMD EPYC™



A scalable solution to accelerate Deep Learning training



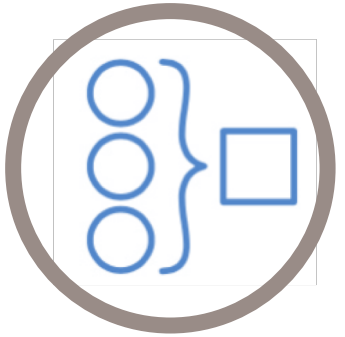
OPTIMIZED DISTRIBUTED DEEP LEARNING TRAINING WITH HOROVOD

- Urika-CS and Urika-XC now include Horovod distributed training framework for TensorFlow, Keras, and PyTorch
- Enables optimization of CNN distributed model training on dense GPU systems
- Enables faster time to model accuracy
- Leverages the Cray MPI library on Urika-XC and OpenMPI on Urika-CS
- Open sourced by Uber, significant ongoing contributions from NVIDIA and others
- General Recommendation for use (current release*):
  - Use Cray PE ML plugin on XC Series and CS500 systems
  - Use Horovod on CS-Storm dense-GPU systems
  - In session II, Pete Mendygral will discuss some next-release enhancements to the Cray PE ML plugin for dense-GPU systems

HYPERPARAMETER OPTIMIZATION

Cray HPO

- New Distributed Hyper Parameter Optimization (HPO) tools for machine learning that automates the time-consuming process of model selection and parameter setting – enabling data scientists to achieve a desired level of model accuracy in a shorter amount of time
- Leverages Deep learning and Analytics stack
- Allows seamless distribution via familiar WLM



Supports three techniques for HPO

- Genetic optimization
- Grid sweep
- Random search

Supports Population Based Training (PBT)

- Learn a custom training parameter schedule while training
- Trains a better model in less time

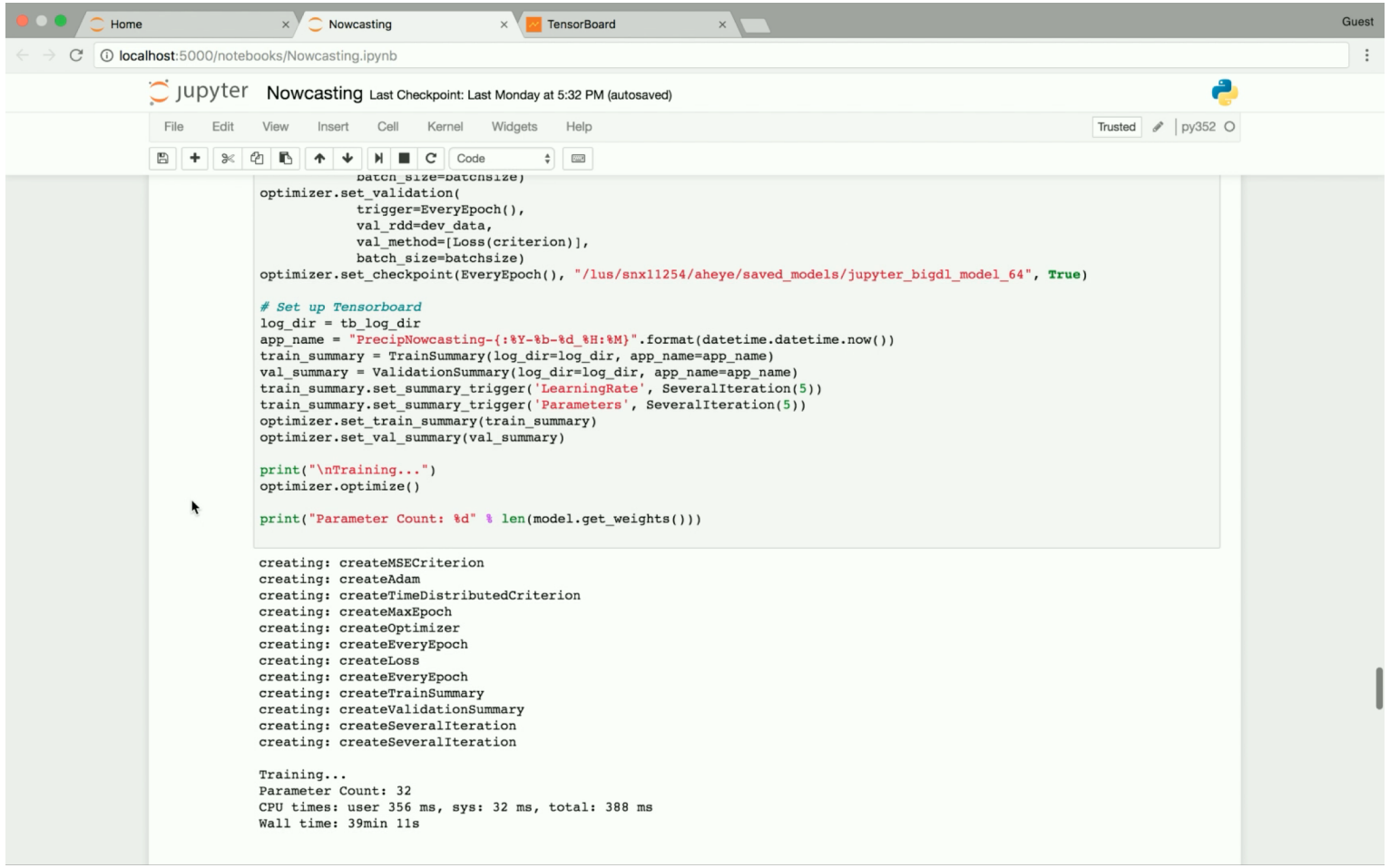
WEB UI FOR INTERACTIVITY & MONITORING

Jupyter Notebooks

Interactive computing and visualization

TensorBoard

- Visualize training
- Examine DNN layers
- Run live (monitor training) or after-the-fact



Introduction to Deep Learning

Mike Ringenburg

A Specific Example

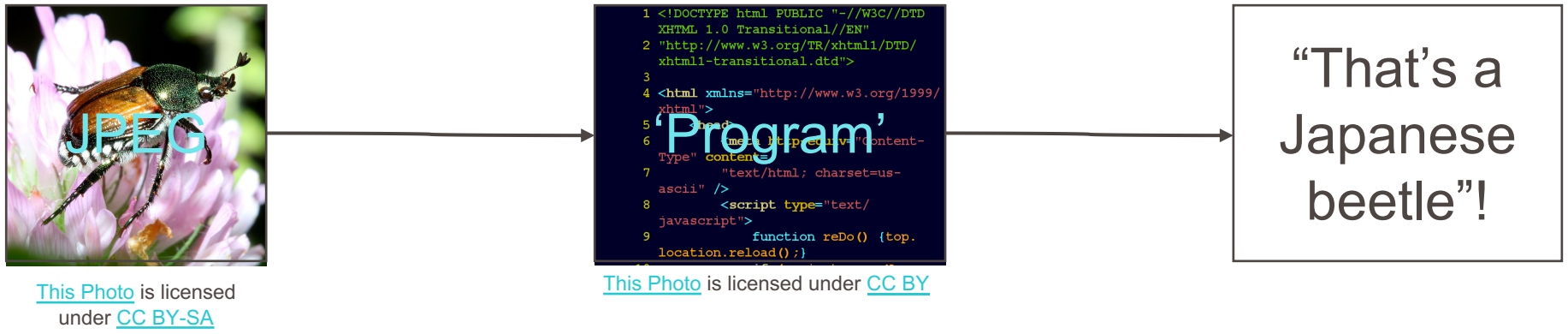
- An organic gardener is building a robot in his garage to recognize the 10 insects found in his garden, and decide which ones to kill with a laser
- The robot will have a camera, and will capture JPEG files of the insects
- The robot needs a ‘program’ to classify each JPEG according to which of the 10 kinds of insect was photographed

This Photo is licensed under CC BY-SA

This Photo is licensed under CC BY

Inputs & Outputs

- Our input is a JPEG
  - 224x224 pixels, 3 colors → a 224x224x3 element vector of the pixel values
- Our output is a classification
  - One of 10 categories → a 10 element vector with a “1” in the position representing the category to which the image belongs



How many “IF” statements will we need to figure out that a bunch of pixel values is a Japanese beetle?

This is an Artificial Intelligence Problem

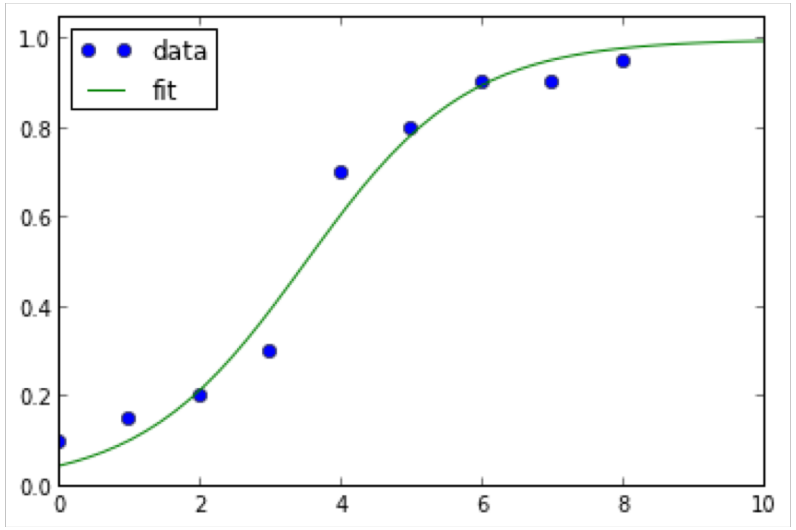
- If you can't get the output from the input with a bunch of loops and conditionals, it's AI
  - But, if that won't work, how can we do it?

This is an Artificial Intelligence Problem

- If you can't get the output from the input with a bunch of loops and conditionals, it's AI
  - But, if that won't work, how can we do it?
- Hint #1: Any mapping of inputs to outputs is a function
- Hint #2: A function can be approximated using a (good) approximating function

An Approximating Function

- How can we determine a good approximating function?
  - Choose its form (linear, polynomial, …)
  - Minimize the overall error at a finite number of inputs with known outputs - - **fit the curve**
    - We have to find the values of the free parameters of the function that minimize the error – it doesn't matter how we do it

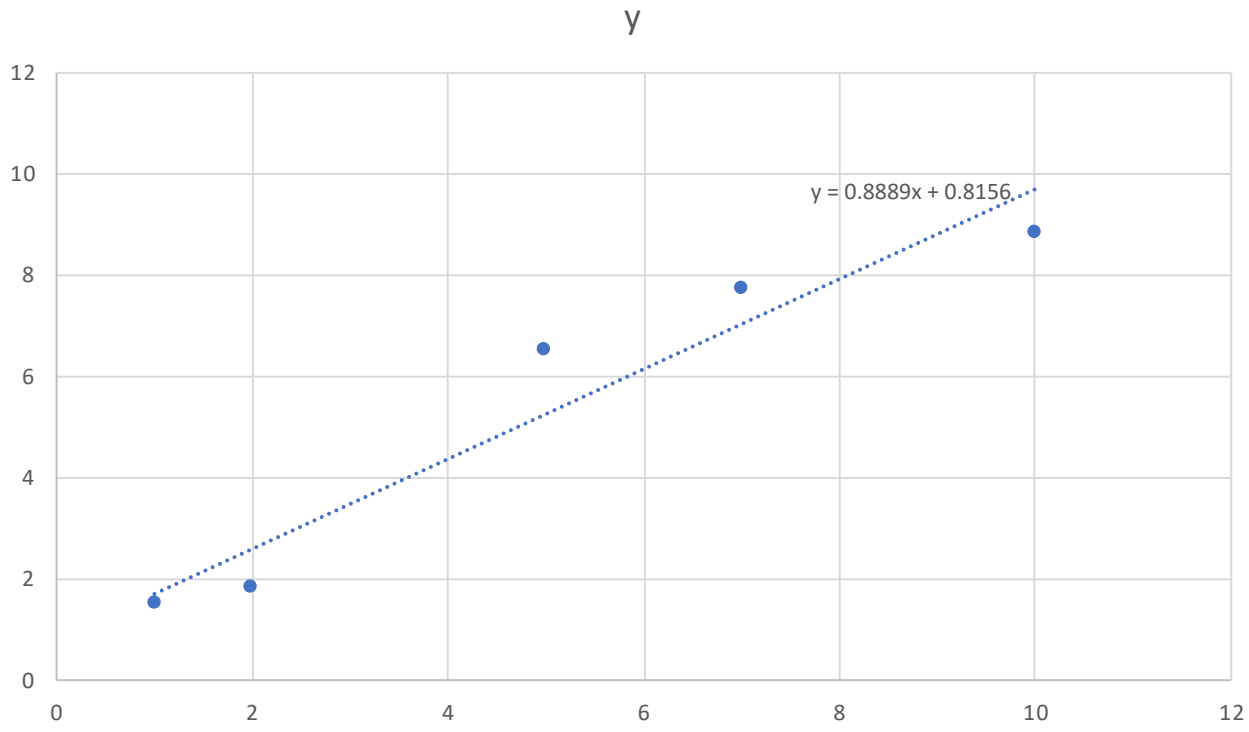


Fitting the curve by adjusting free parameters is *training* the function to know the answer for arbitrary inputs

Training via Gradient Descent

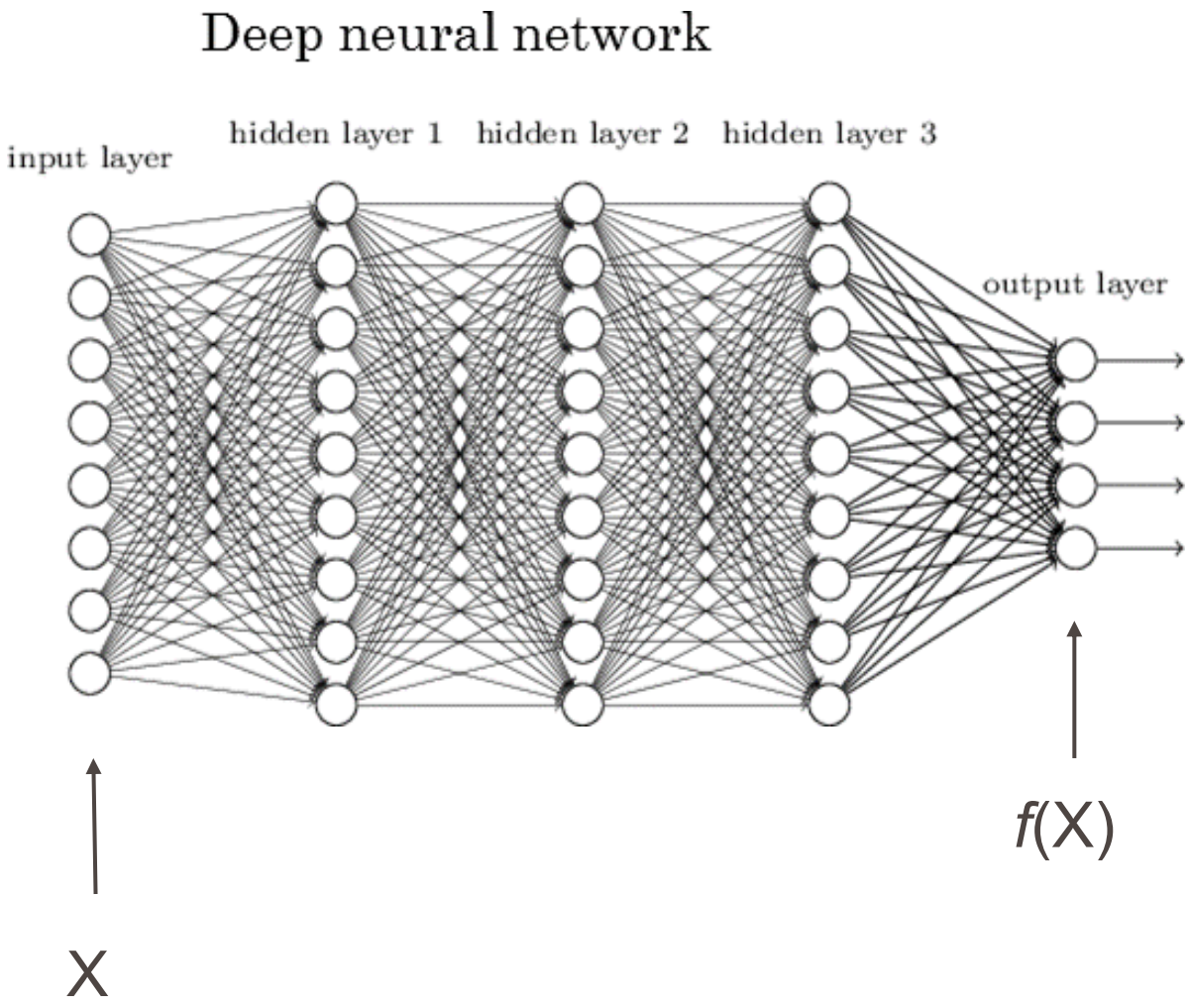
- We want to approximate $y=f(x)$
  - Really, we want to find a function that maps a set of inputs to a set of outputs, to some level of accuracy
- We know $y_i=f(x_i)$, for i=1,N
- Iterate:
  - First iteration only: initialize the free parameters of *f*
  - Calculate error (over our N known points)
  - Calculate gradient of error, as a function of the free parameters of function *f*
  - Adjust the free parameters of function *f* a 'small' distance in the direction of the negative of the error gradient
  - Assess convergence & stop when 'good enough'

Training Error and Validation Error



- Here, we chose the function y=ax+b, with “a” and “b” as the free parameters
- “a” and “b” were chosen to minimize the *training error*, using the 5 points shown
- If we test this function against a distinct set of known data points, we could determine the *validation error*

A Really Useful Kind of Function



- This image shows a *deep neural network*
  - An approximating function, with free parameters called *weights* and *biases*
  - Deep networks have been found to be especially powerful
  - Neural networks can approximate any continuous function arbitrarily well

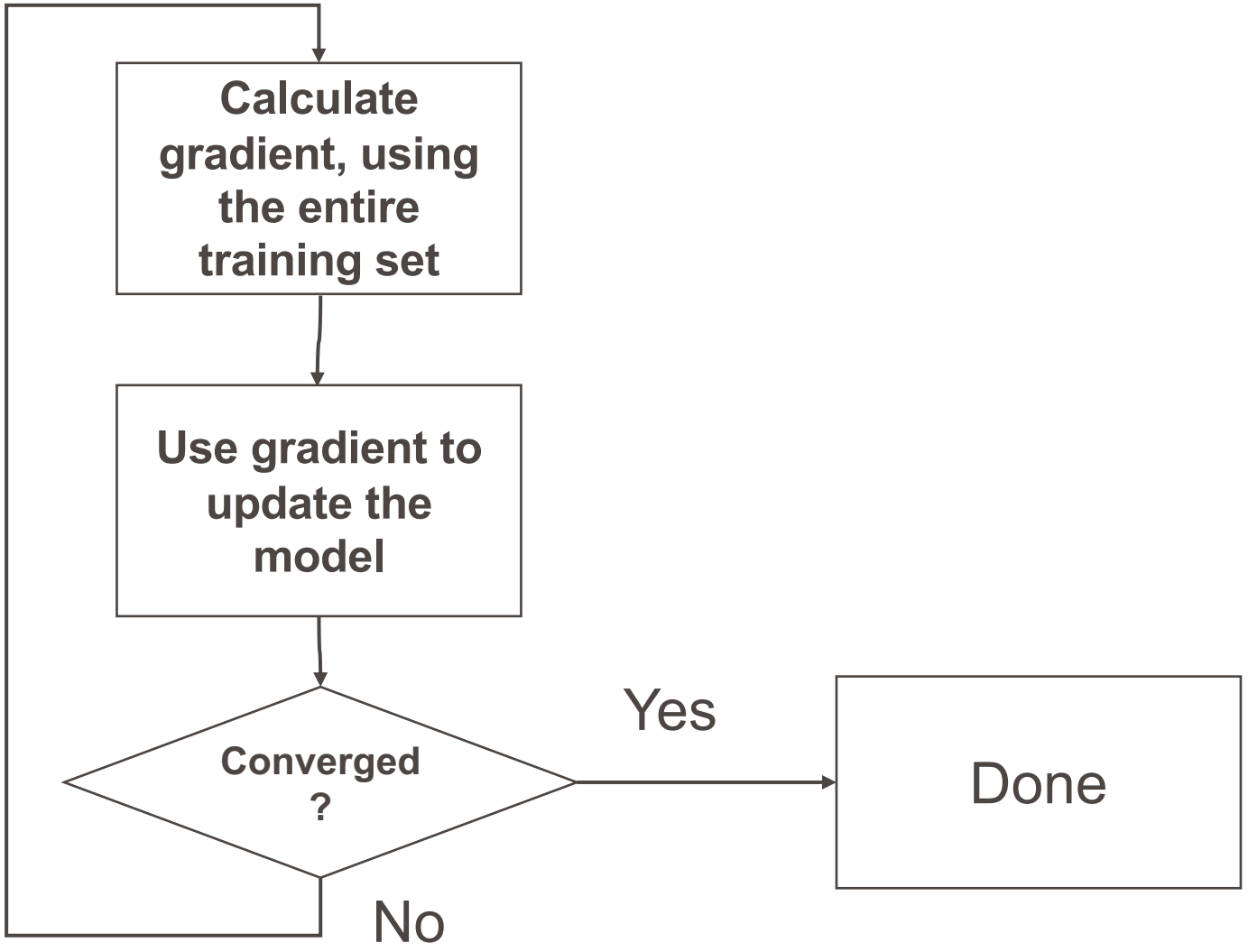
The Big Picture

- Training a “sufficiently complex” neural network on a “large” and “representative” data set should allow it to “know” about novel data
  - If we show the neural network 1,000,000 pictures of cats, it should recognize new pictures of cats
  - If we only show the network pictures of black cats, it might not recognize white cats
  - If the network only has 4 “neurons”, it probably can’t learn to recognize cats

Some Terminology

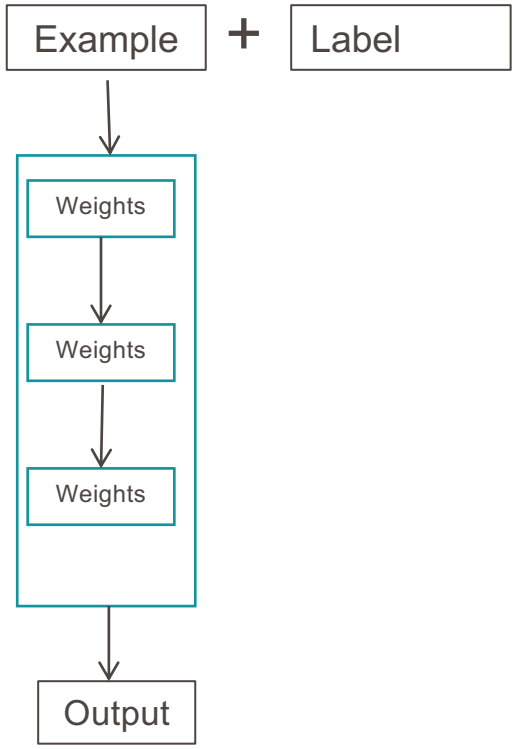
- The training data consists of training *examples*
  - Each example is an input with a known correct output, called a *label*
  - Having labeled examples is a special but common case, and we won't go deeper on this topic today
- A subset of the training data is often called a *minibatch*
- One 'trip' through the whole training set is called an *epoch*
  - Often, bookkeeping, convergence testing, checkpointing, etc. are done after each epoch

Gradient Descent Algorithm



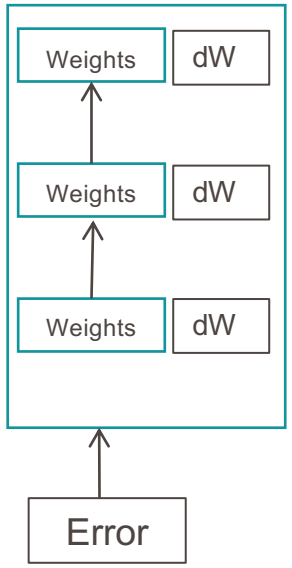
Training Schematic

Feedforward



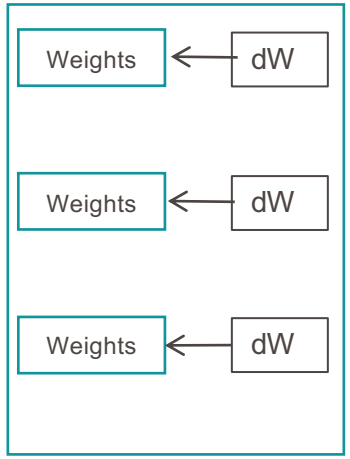
One or more training examples *feedforward* through the layers of *weights*, producing an output

Backpropagate



The error, which is the difference between the label and the output, is *backpropagated* through the layers, producing the *gradients*

Update



The weights are updated by adding the gradients (scaled by a multiplier) to them

Feedforward and backpropagate are much more expensive than update (>100X)

Variations on the Gradient Descent Algorithm

- Stochastic Gradient Descent
  - A gradient is calculated, and the model is updated, for each training example
- Batch Gradient Descent
  - The training examples are divided into minibatches
  - A gradient is calculated and the model is updated for each minibatch
- Strict Gradient Descent is seldom if ever used
- Strict Stochastic Descent is seldom if ever used
- Batch Gradient Descent is almost always used
  - And, everyone calls it Stochastic Gradient Descent (SGD)

Parallelizing SGD

- Data parallel methods
  - “Minibatch Parallel”
  - Every worker independently calculates a “local gradient” using a “local minibatch”
  - All workers participate in an allreduce, or communicate with a parameter server, to average all the gradients and synchronize with other workers
- Model parallel methods
  - Break the neural network up – different layers on different nodes
  - Useful if the model is too large for a single node
  - But often more communication than data parallel methods

For More Information…

A good overview:

Efficient Processing of Deep Neural Networks: A Tutorial and Survey

https://arxiv.org/abs/1703.09039

TensorFlow

- Developed by Google
- Most popular DL framework
- Large open source community
- APIs for
  - Python
  - C++
  - Go
  - Java
- Optimized for CPU and GPU architectures
- Ships with Urika-XC
- Learn TensorFlow
  - Docs: https://www.tensorflow.org/get_started/
  - Programmer's Guide: https://www.tensorflow.org/programmers_guide/
  - Tutorials: https://www.tensorflow.org/tutorials/

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu',input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

- High-level neural networks API – just add layers!
- New versions of TensorFlow include Keras APIs

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu',input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Construct Model

- High-level neural networks API – just add layers!
- New versions of TensorFlow include Keras APIs

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu',input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Configure Model for Training

- High-level neural networks API – just add layers!
- New versions of TensorFlow include Keras APIs

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu',input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Train Model

- High-level neural networks API – just add layers!
- New versions of TensorFlow include Keras APIs

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu',input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Evaluate Model

- High-level neural networks API – just add layers!
- New versions of TensorFlow include Keras APIs

Hyperparameter Optimization

Kristyn Maschhoff, Ph.D., Cray Inc.

kristyn@cray.com

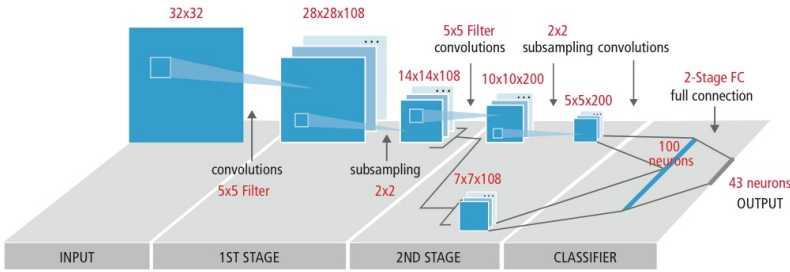
Goals of this Session

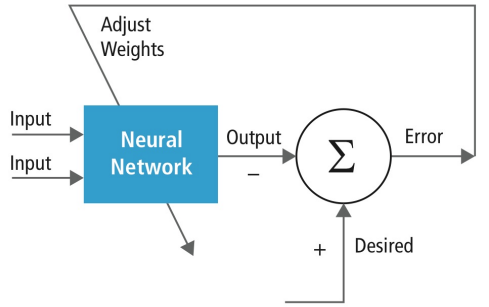
- Introduce the concept of Hyperparameter Optimization (HPO)
  - What are hyperparameters?
  - Why are they important?
- Introduce automatic distributed hyperparameter optimization
  - Using the Cray HPO library
- Demo HPO Jupyter Notebook integration on Cori
- Hands-On Exercises

Background: Hyperparameters

- **Model parameters** – internal values in a model determined from data
  - In neural networks: weights (connection, bias)

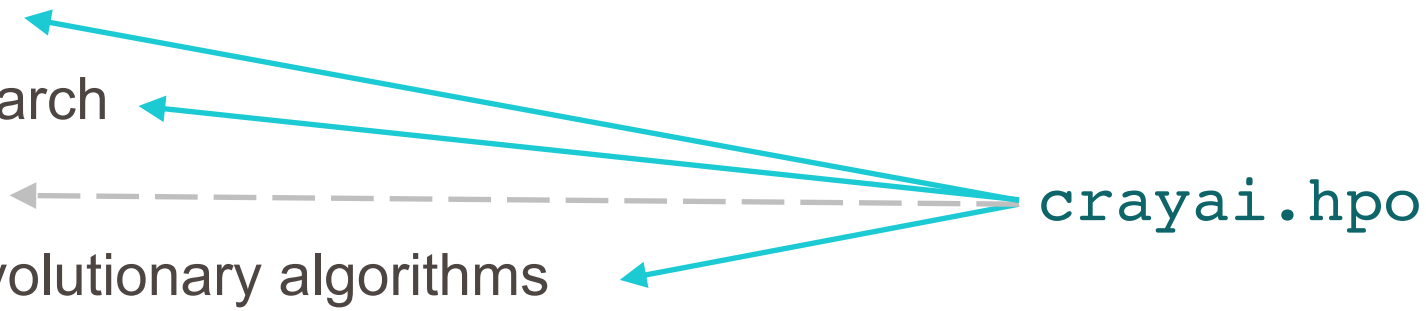
- **Model hyperparameters** – external values to model that determine model capacity
  - In neural networks:
    - **Topology**:
      - Number of neurons in fully connected layers
      - Filters, kernel sizes, convolutional / pooling strides
      - Nonlinearity: logistic, ReLU, tanh
    - **Training**:
      - Learning rate, batch size, momentum
      - Dropout probability, batch normalization
      - Optimizers (SGD, Adam, RMSProp, AdaGrad, etc)



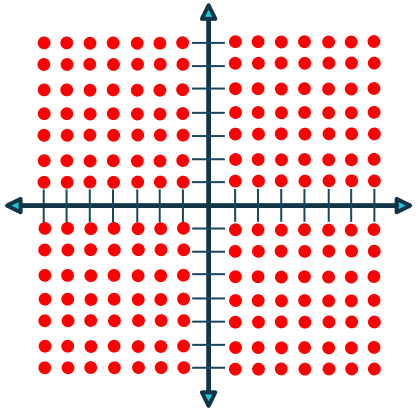


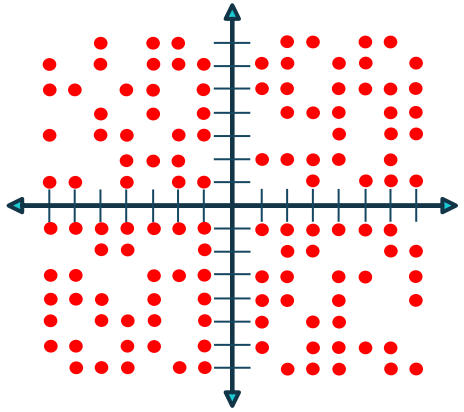
Background: Hyperparameter Optimization

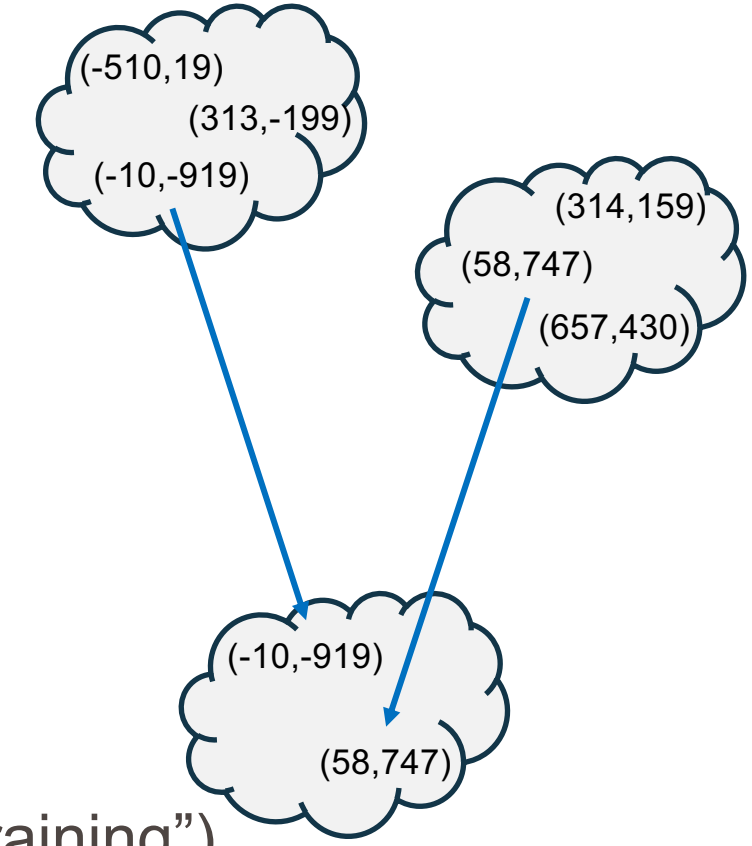
- By hand
  - Hyperparameters are selected and tuned manually
  - Guided by intuition and rules of thumb
- Hyperparameter optimization
  - Brute-force of entire search space intractable
    - More combinations of hyperparameters than atoms in the observable universe
  - Evaluate a subspace
    - Grid search
    - Random search
    - Bayesian
    - Genetic / evolutionary algorithms



Hyperparameter Optimization







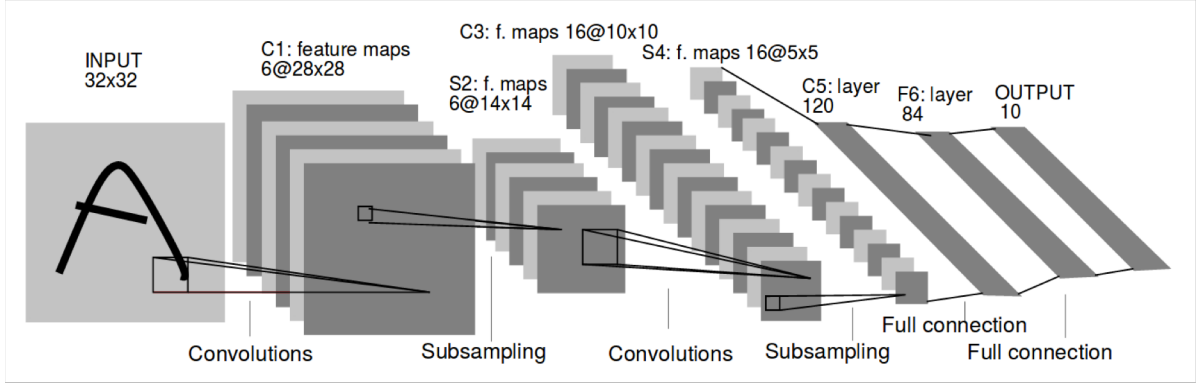
- Grid Search
- Random Search
- Genetic / Evolutionary Algorithms (“population-based training”)

Background: Importance of Hyperparameters

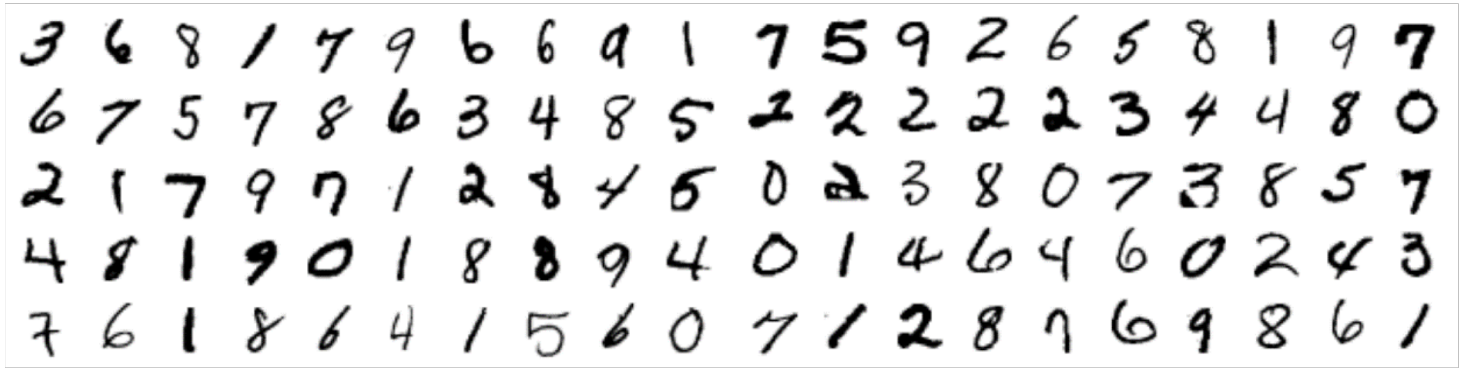
- Finding a good set of hyperparameters can have a big impact:
  - Accuracy
  - Time-to-accuracy
  - Preventing underfitting / overfitting
- Analysts consistently highlight importance of HPO in their workflows

HPO Example – LeNet and MNIST

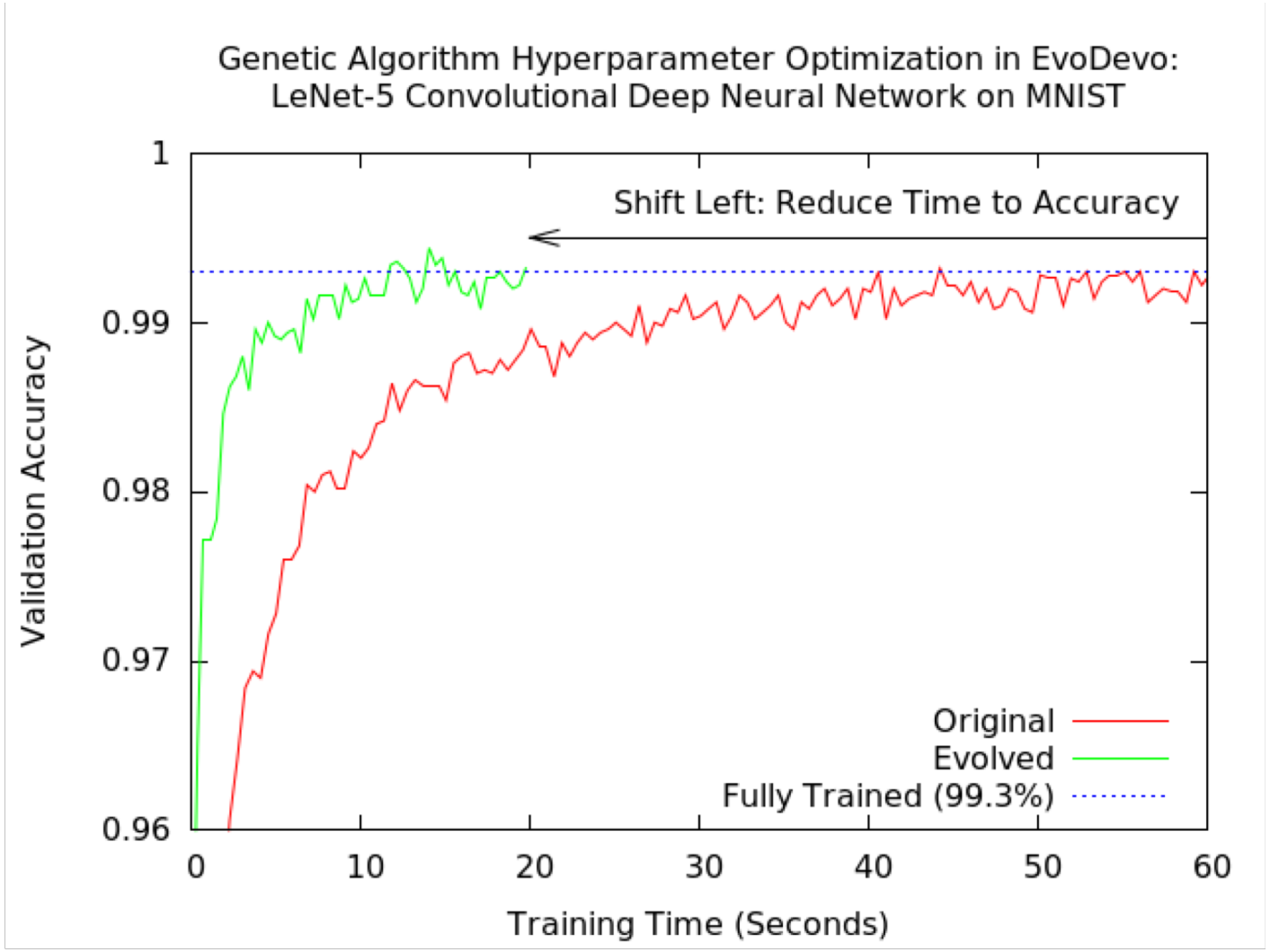
- LeNet-5, 7 layers, 5 hidden:



- MNIST, 70k 28x28 greyscale images, 10 classes:

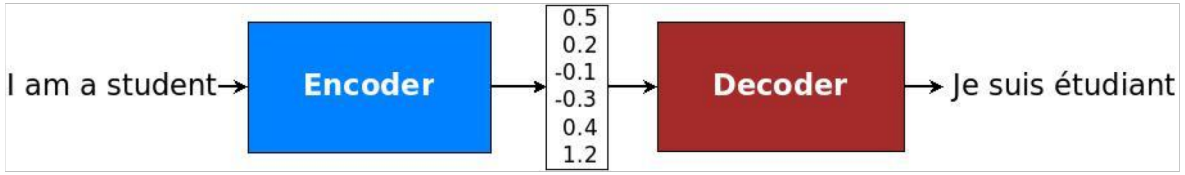


HPO Results – LeNet on MNIST

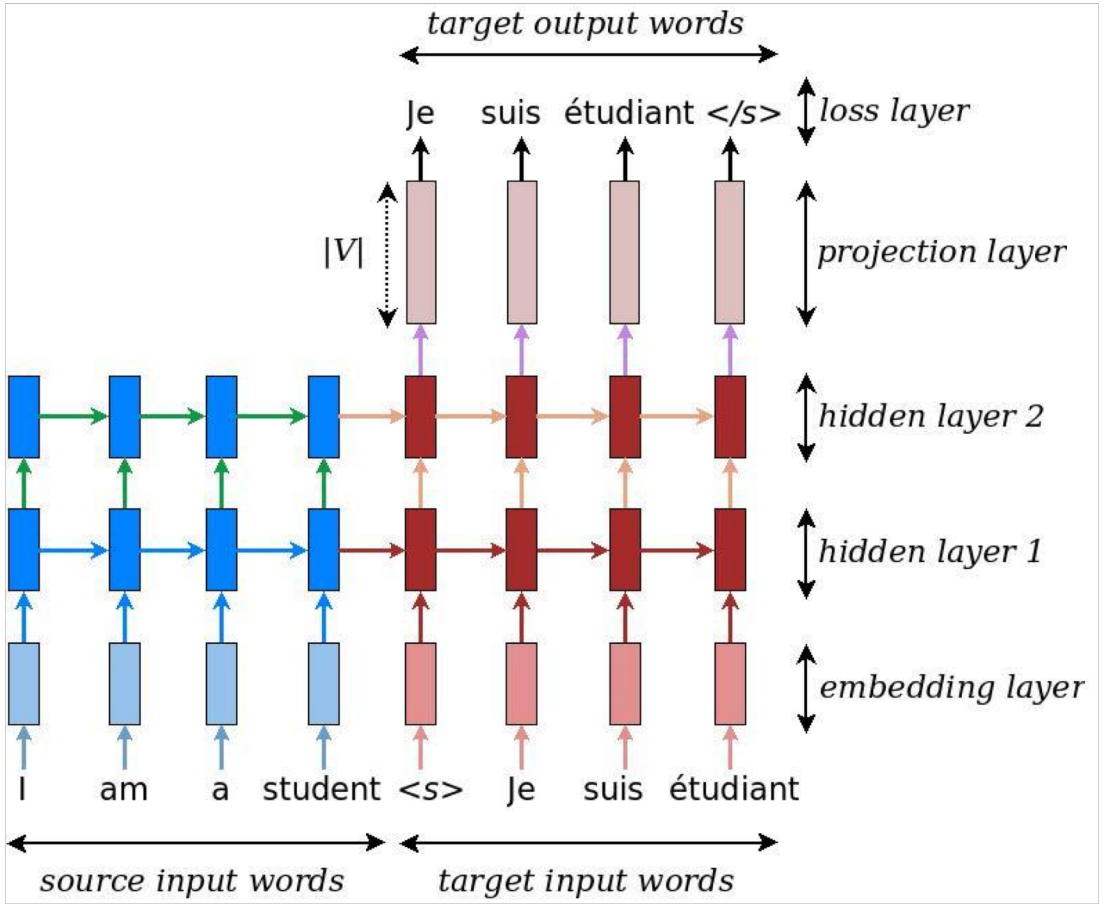


Another Example: Machine Translation PBT

- Neural Machine Translation (NMT)
- Encoder-decoder architecture

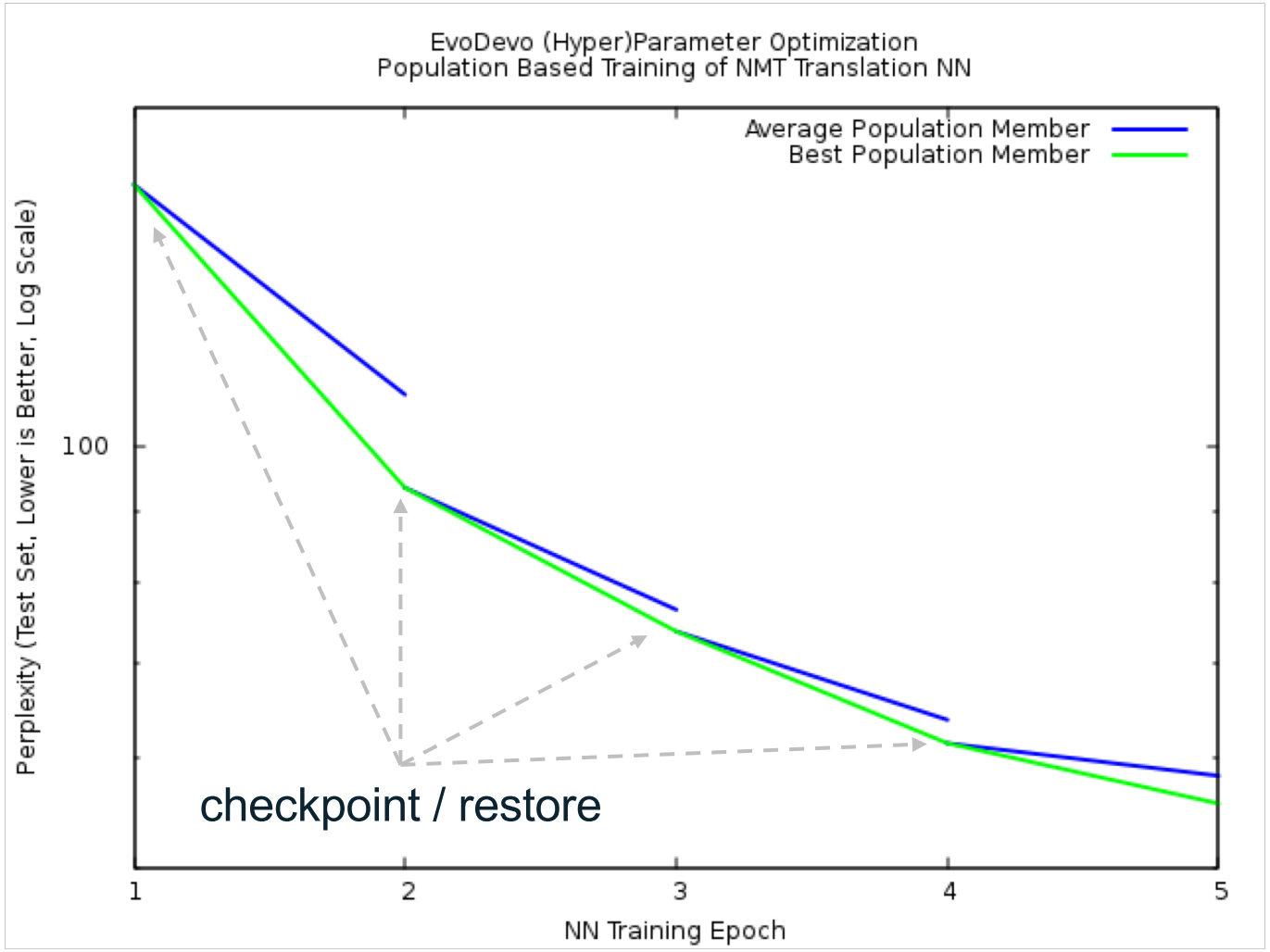


- Recurrent neural network
- English to French



Population Based Training

- Optimize hyperparameters *and parameters*
  - Hyperparameters optimized as usual with GA / EA
  - Parameters optimized with checkpoint / restore:
    - At the end of each epoch, population copies best parameters
  - Creates a “training schedule” with customized epochs



Cray Hyperparameter Optimization

- Cray integrated HPO support with Python interface and Chapel backend
- Supported distributed optimization as well as distributed training
  - E.g., 20 nodes, 5 HPO instances each training on 4 nodes
- Simple steps to use:
  - Create a python wrapper script
  - Define optimizer and configuration
  - Provide parameters, search range and executable command
  - Run wrapper script to optimize

hpo_example.py

```
from crayai import hpo

eval = hpo.Evaluator('python …')
params = ([["--learning_rate", 0.01, (1e-6, 0.1)],
           ["--dropout_rate",  0.5,  (0.3, 0.7)],
           ...]

Optimizer = hpo.{genetic,grid,random}.Optimizer(
               params, eval, launcher="urika",
               generations=5)
```

```
$ python hpo_example.py
```

Cray Hyperparameter Optimization

- The Cray HPO package comes with Urika integration
  - Leverages deep learning and analytics resources of the Urika image
  - Allows seamless distribution with workload managers
- Supports three techniques for HPO
  - Genetic optimization
  - Grid sweep
  - Random search
- Supports Population Based Training
  - Learn a custom training parameter schedule while training
  - Trains a better model in less time

Interface: Model Training Script

```
45
46      parser = argparse.ArgumentParser()
47      parser.add_argument('--lr', '-l', default=0.01, type=float,
48                          help='Learning Rate')
49      parser.add_argument('--dropout', '-d', nargs='?', default=0.5, type=float,
50                          help='drop rate for layers')
51      args = parser.parse_args()
52
```

Expose hyperparameters as command line args

```
149     # constructing hidden layer 1
150     layer1 = tf.layers.Dense(units=layerSize[0], activation=tf.nn.relu, use_bias=True)
151     layer1Out = tf.layers.dropout(inputs=layer1(inputs=trainInput), rate=FLAGS.dropout)
152
153     # constructing hidden layer 2
154     layer2 = tf.layers.Dense(units=layerSize[1], activation=tf.nn.sigmoid, use_bias=True)
155     layer2Out = tf.layers.dropout(inputs=layer2(inputs=layer1Out), rate=FLAGS.dropout)
```

Utilize hyperparameters from command line args

```
168
169     trainOperation = tf.train.GradientDescentOptimizer(learning_rate=FLAGS.lr).minimize(loss=loss)
170     prediction = tf.argmax(output, 1)
171     evalPrediction = tf.argmax(evalOut, 1)
172
```

```
217
218         print("FoM: %e" % lossVal)
219
```

Print field of merit

Interface: HPO Script

```
1  #!/usr/bin/env python3
2  # encoding: utf-8
3
4  from crayai import hpo
5
6  evaluator = hpo.Evaluator('python source/train_model.py')
7
8  params = hpo.params([["--lr",  0.001, (1e-5, 0.1)],
9                       ["--dropout",  0.5, (0.01, 1)]])
10
11 optimizer = hpo.genetic.Optimizer(evaluator,
12                                    generations=20,
13                                    pop_size=10,
14                                    num_demes=4,
15                                    log_fn='genetic.log')
16
17 optimizer.optimize(params)
18
19 print(optimizer.best_fom)
20 print(optimizer.best_params)
21
```

Import `crayai` module

Set model training script

Provide hyperparameters flags, default values, and bounds/possible values

Setup and run hyperparameter optimizer

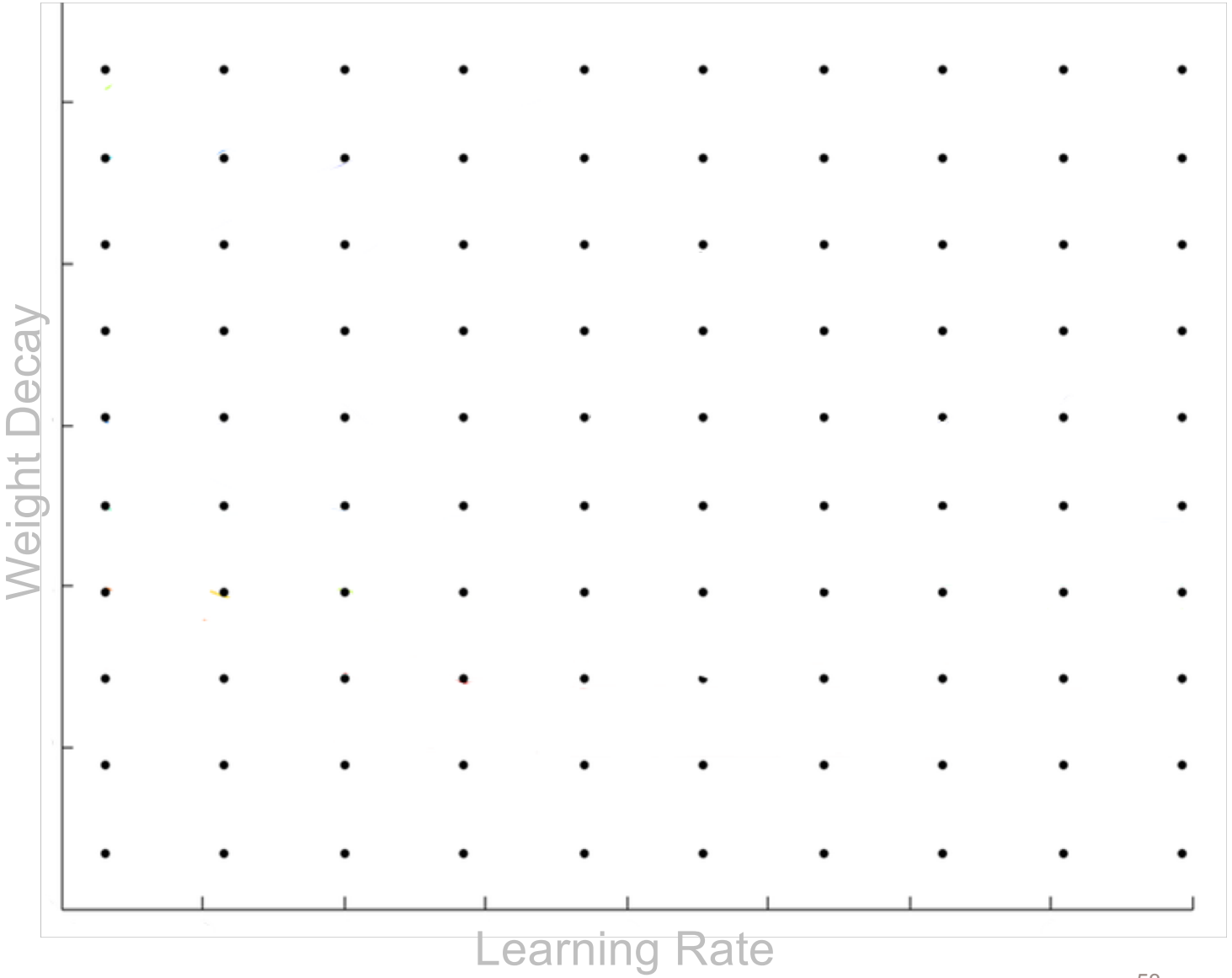
Results stored in optimizer object

CrayAI HPO Strategies

- Traditional HPO
  - Grid Search
  - Random Search
  - Genetic Search
- Schedule training
  - Population-based Training

Grid Search

- Advantages
  - Simple
  - Easily parallelizable
- Disadvantages
  - Curse of dimensionality
  - Computation expense
- Baseline HPO

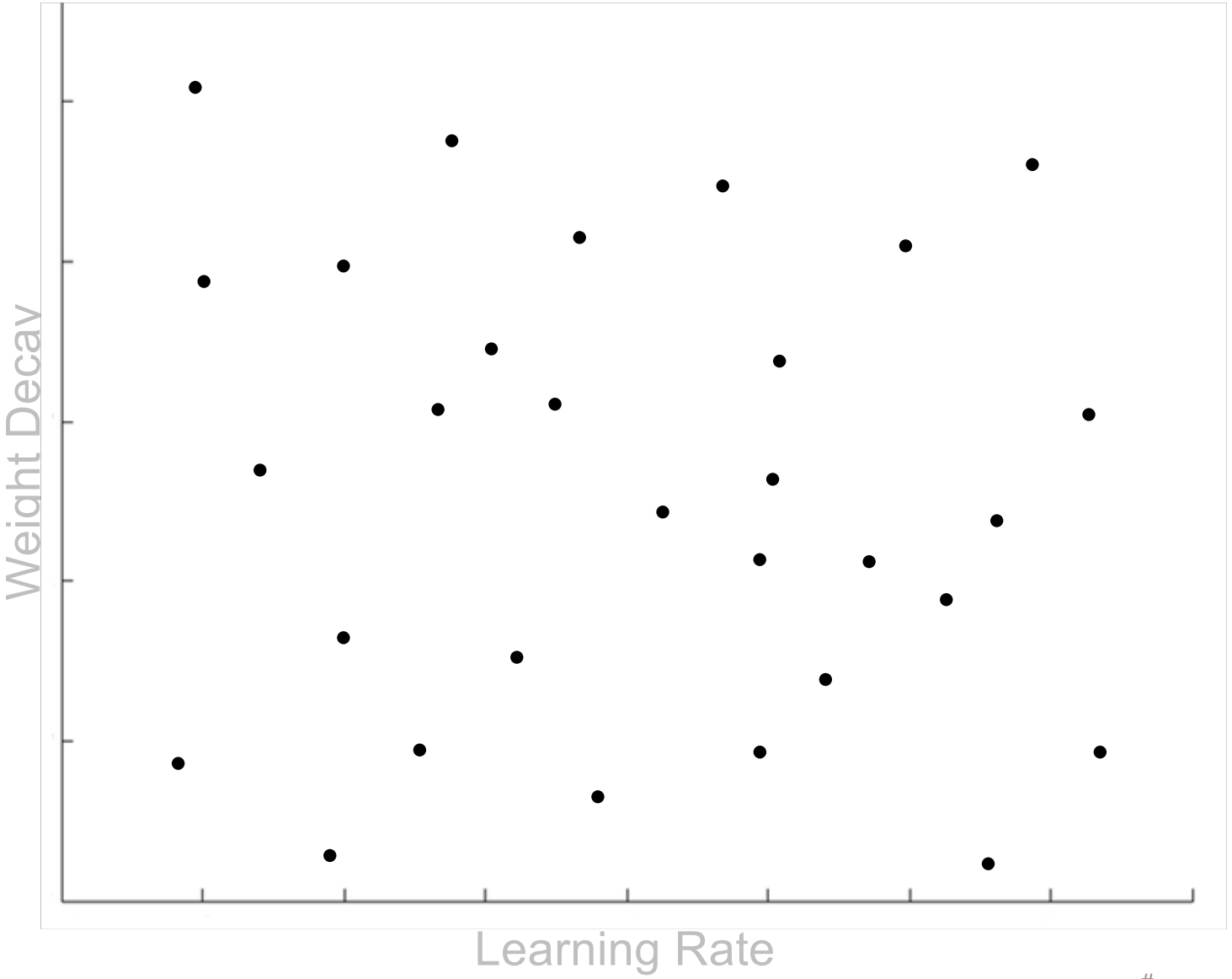


Grid Search: Example

```
#!/usr/bin/env python3
# encoding: utf-8

from crayai import hpo

evaluator = hpo.Evaluator('python source/train_model.py')

params = hpo.params([["--lr",  0.001, (1e-5, 0.1)],
                     ["--dropout",  0.5, (0.01, 1)]])

optimizer = hpo.grid.Optimizer(evaluator,
                               grid_size=3)




optimizer.optimize(params)

print(optimizer.best_fom)
print(optimizer.best_params)

```

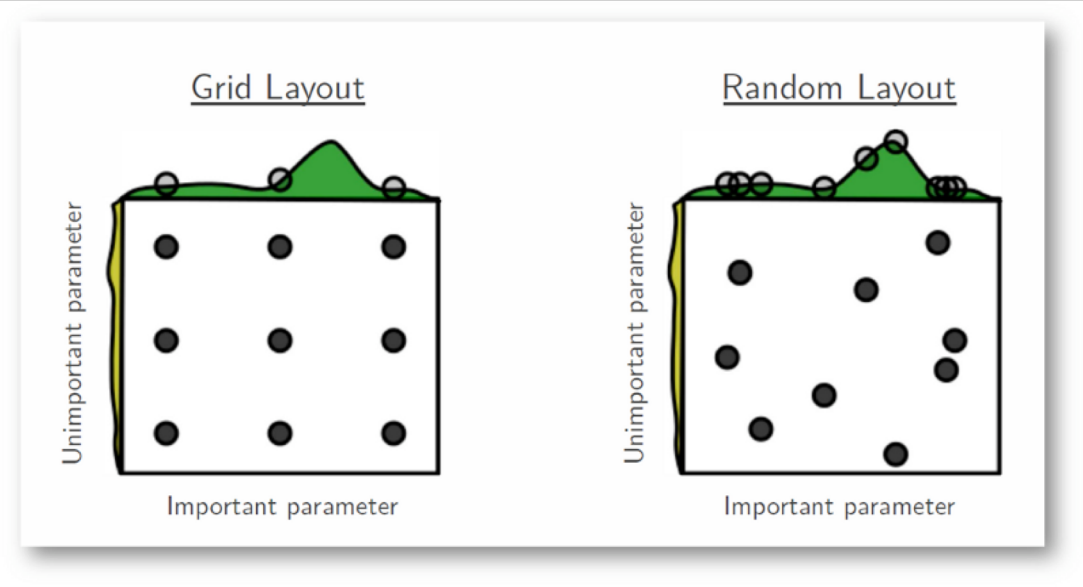
Random Search

- Advantages
  - Simple
  - Easily parallelizable
  - More efficient than grid
- Disadvantages
  - Computation expense



Random Search

- Generate random hyperparameters
  - Simple yet surprisingly effective
- Samples each hyperparameter at a higher rate for a given computation



"for any distribution over a sample space with a finite maximum, the maximum of 60 random observations lies within the top 5% of the true maximum, with 95% probability."

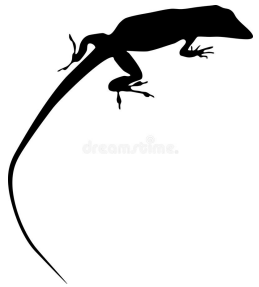
Random Search: Example

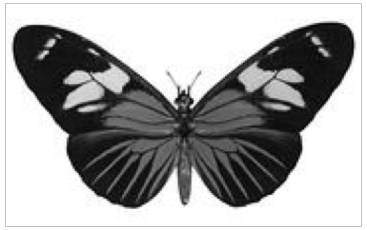
```
#!/usr/bin/env python3
# encoding: utf-8

from crayai import hpo

evaluator = hpo.Evaluator('python source/train_model.py')

params = hpo.params([["--lr",  0.001, (1e-5, 0.1)],
                     ["--dropout",  0.5, (0.01, 1)]])

optimizer = hpo.random.Optimizer(evaluator,
                                 num_iters=800,
                                 seed=10)



optimizer.optimize(params)

print(optimizer.best_fom)
print(optimizer.best_params)

```

Genetic Search

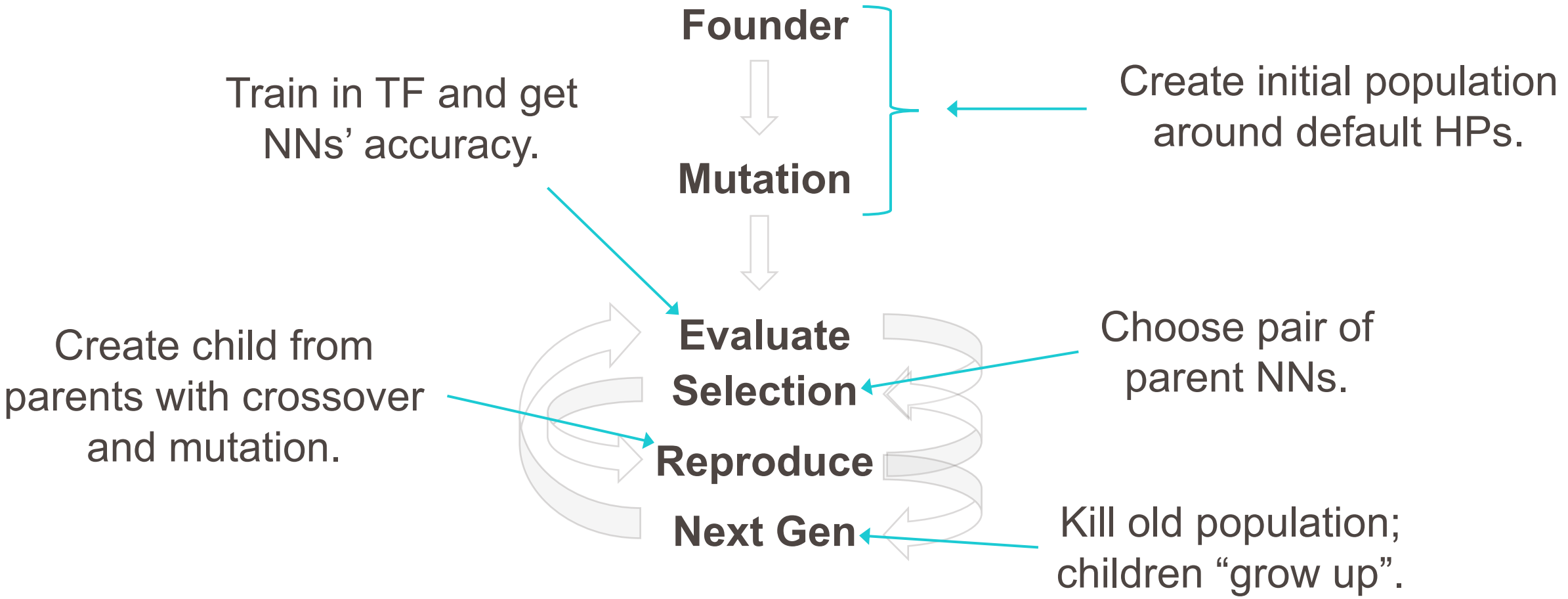
- Think of a genetic algorithms applied to HPO as:
  - “Automatic, iterative, stochastic grid search with pruning.”

- Inspired by biological systems found in nature:
  - Mutation
  - Crossover
  - Selection

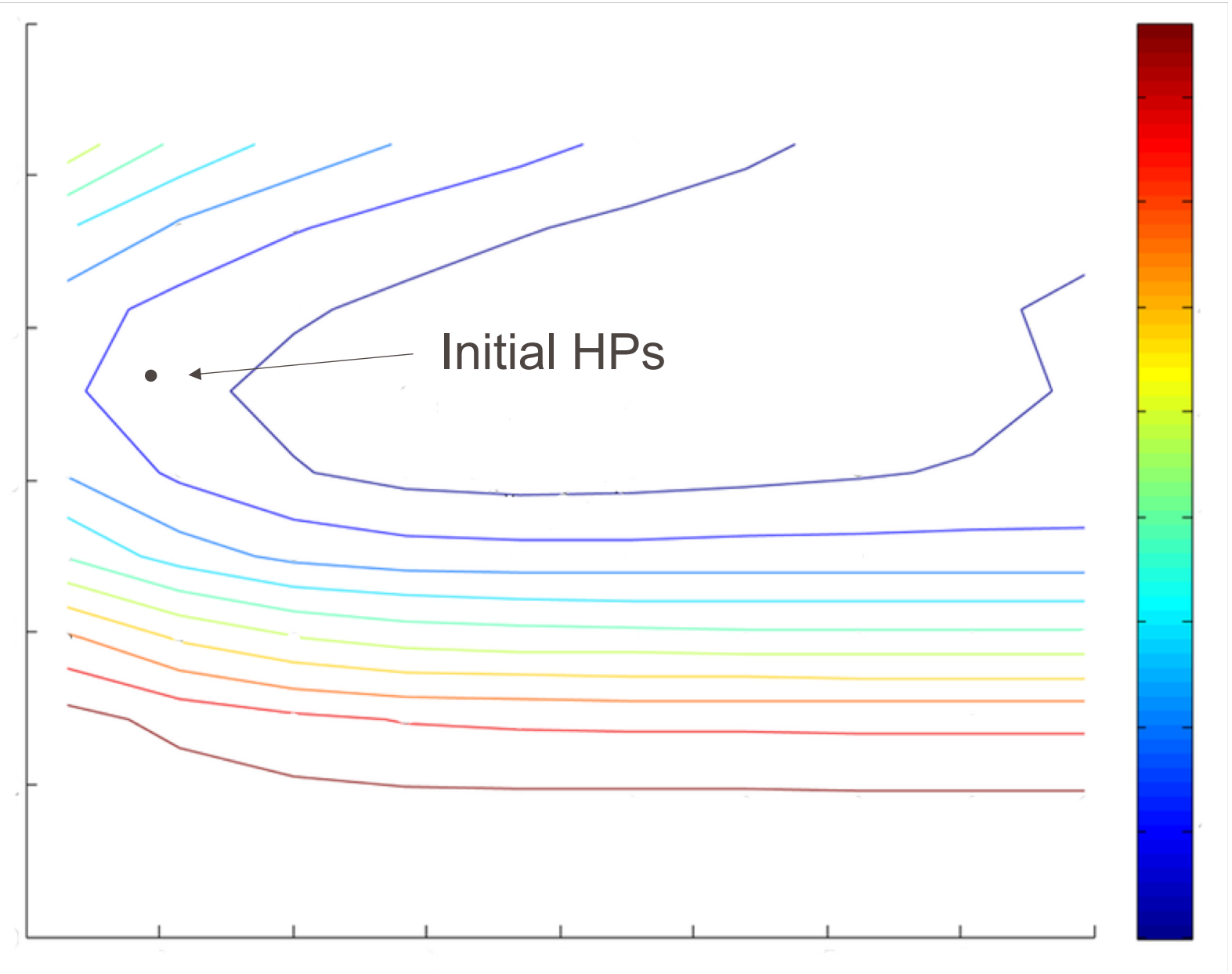


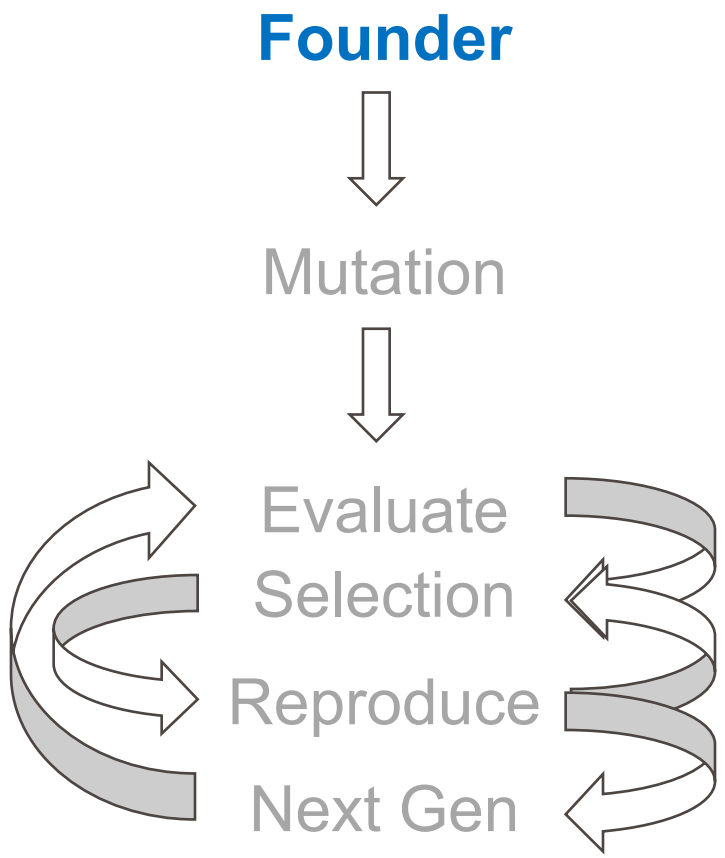


Genetic Search: Generation Cycle

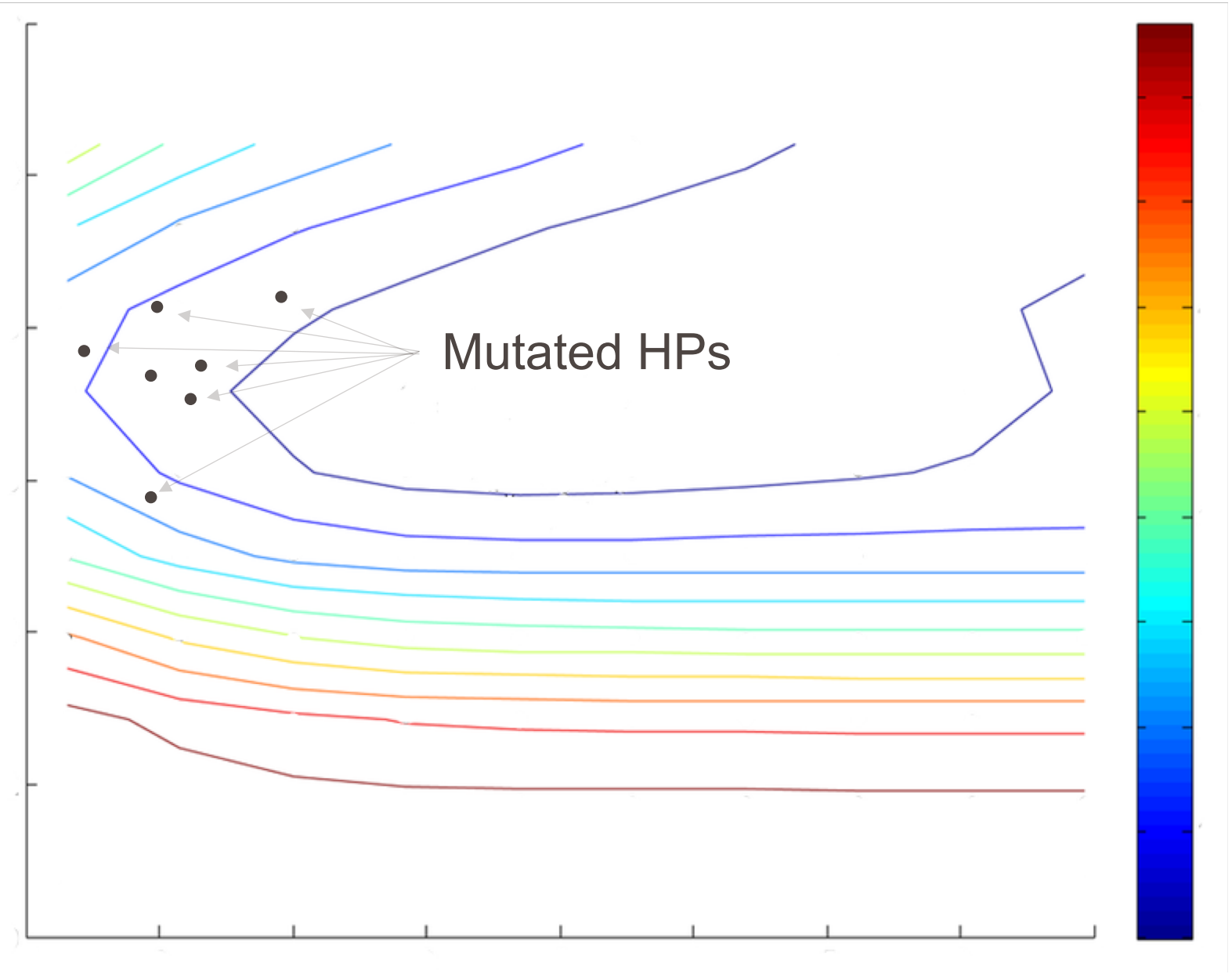


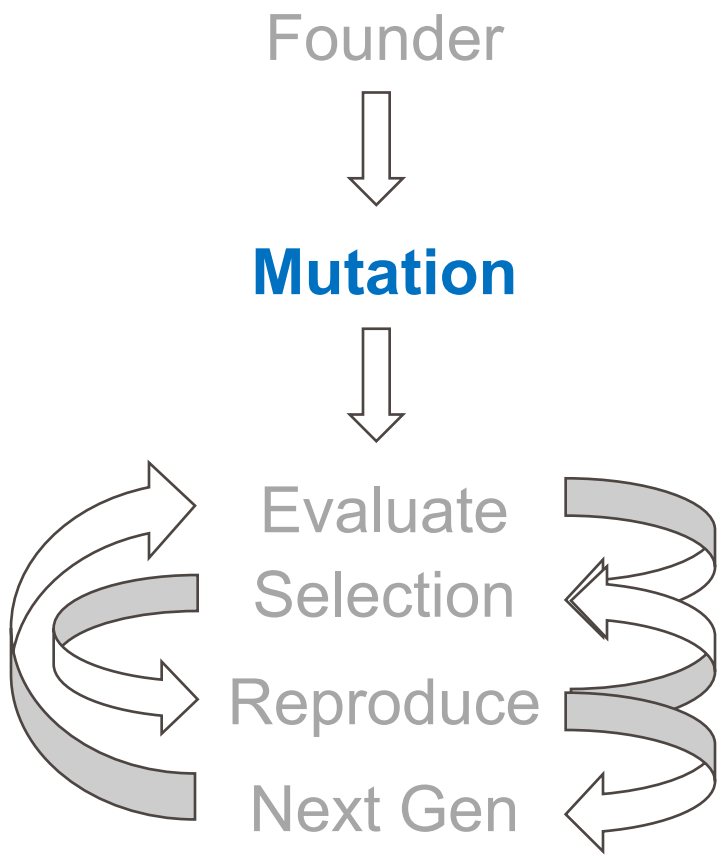
Genetic Search: Founder



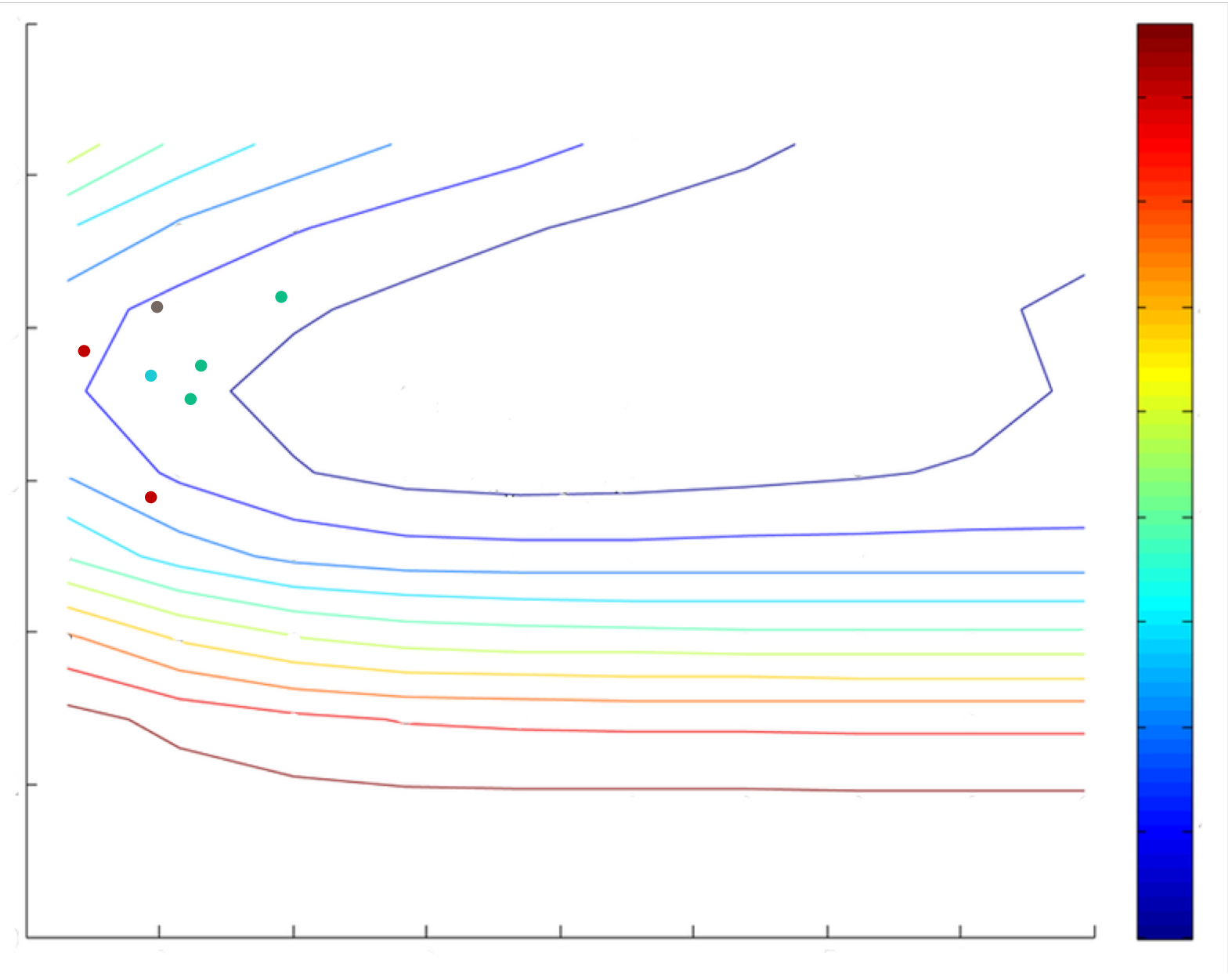


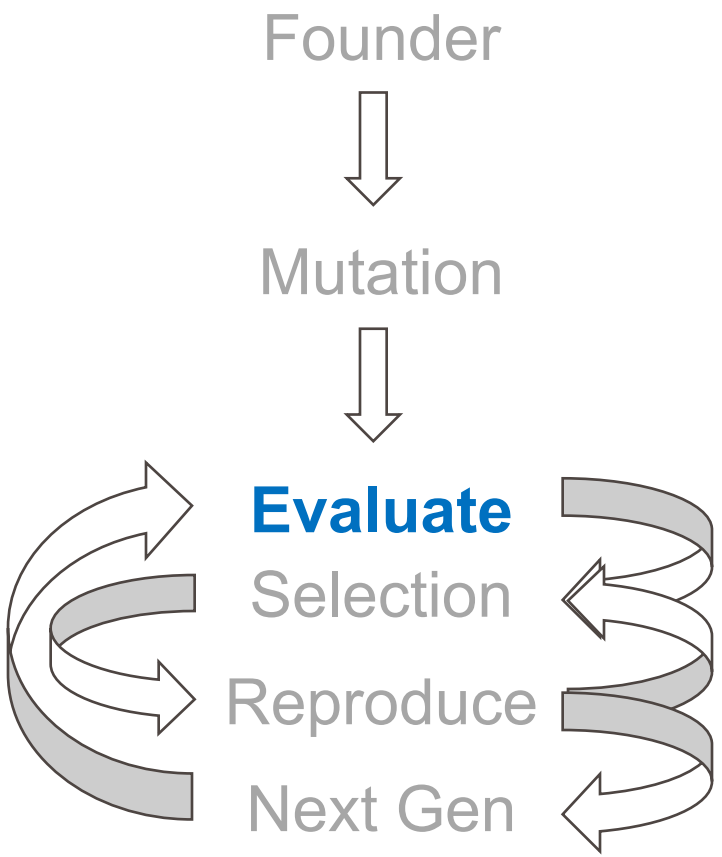
Genetic Search: Initial Mutation



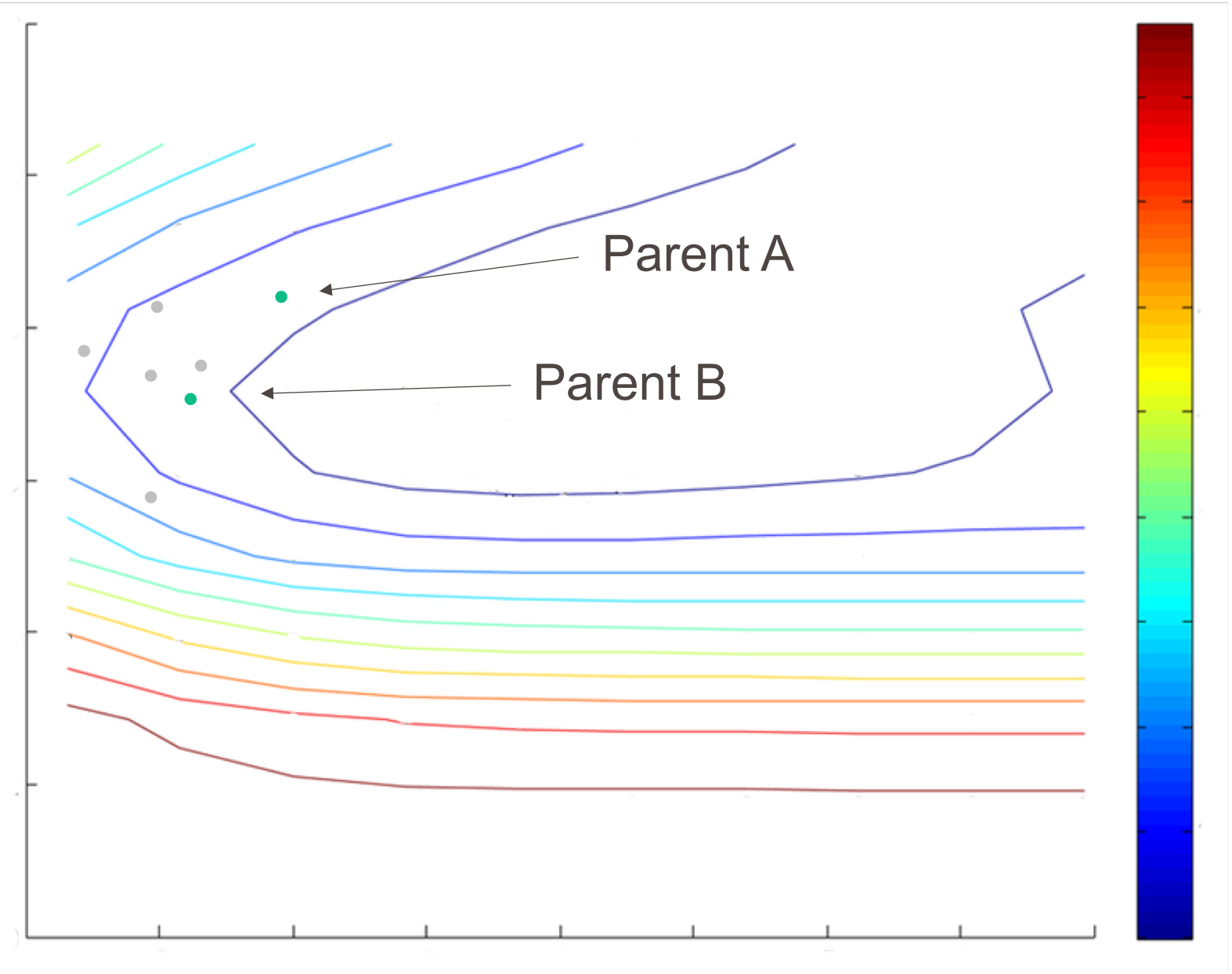


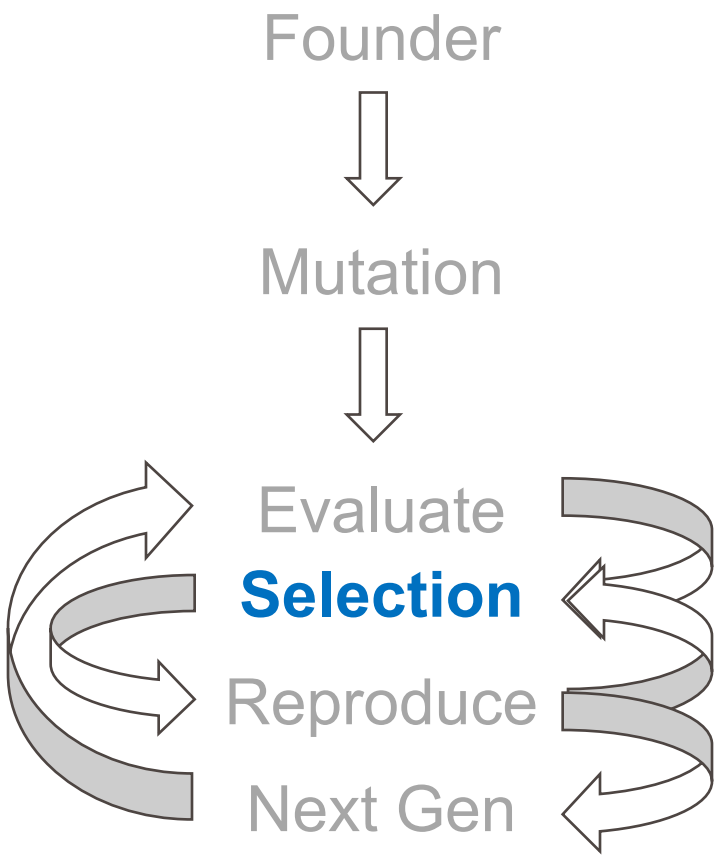
Genetic Search: Evaluate Fitness 1



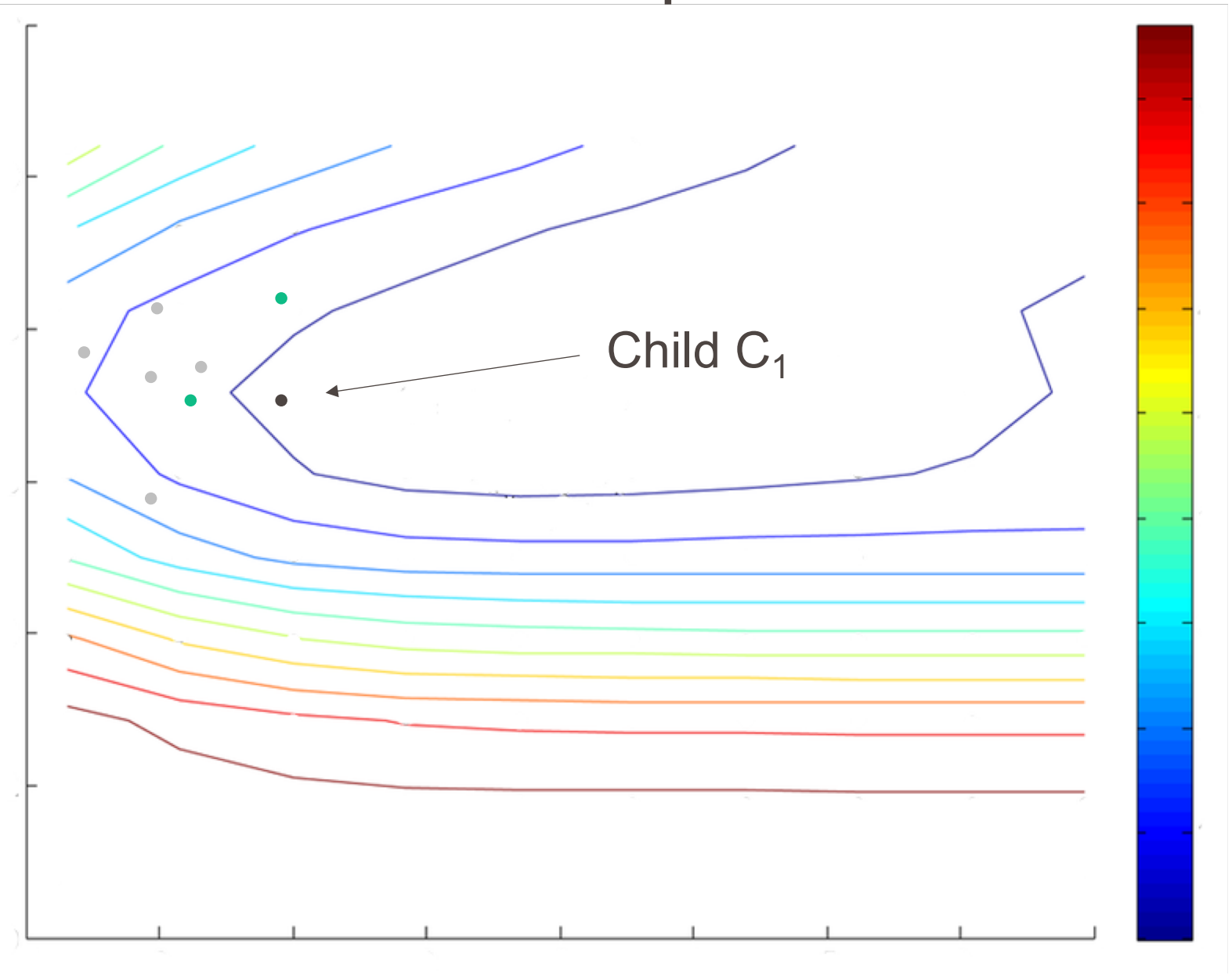


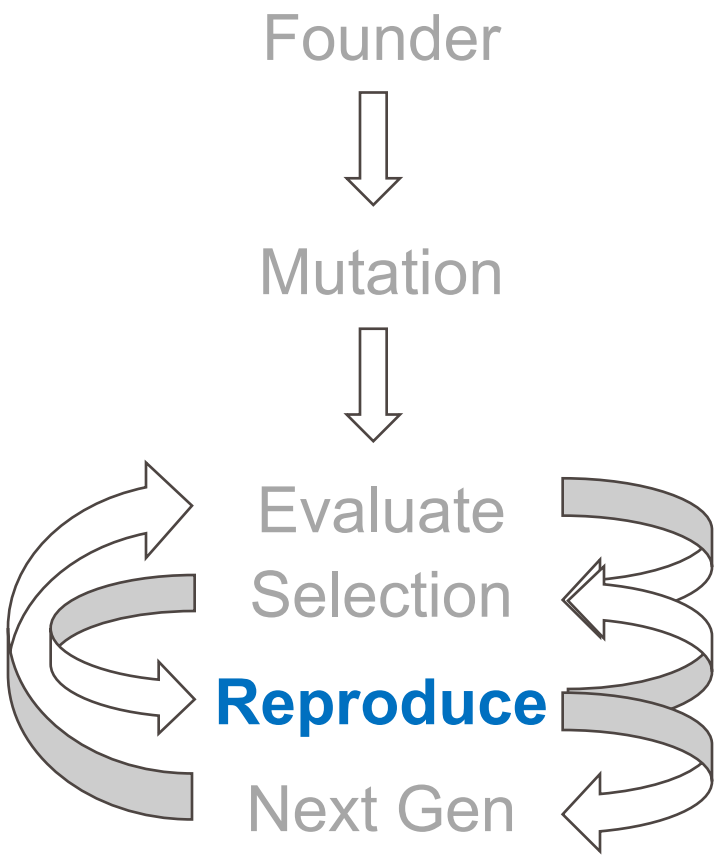
Genetic Search: Mate Selection 1



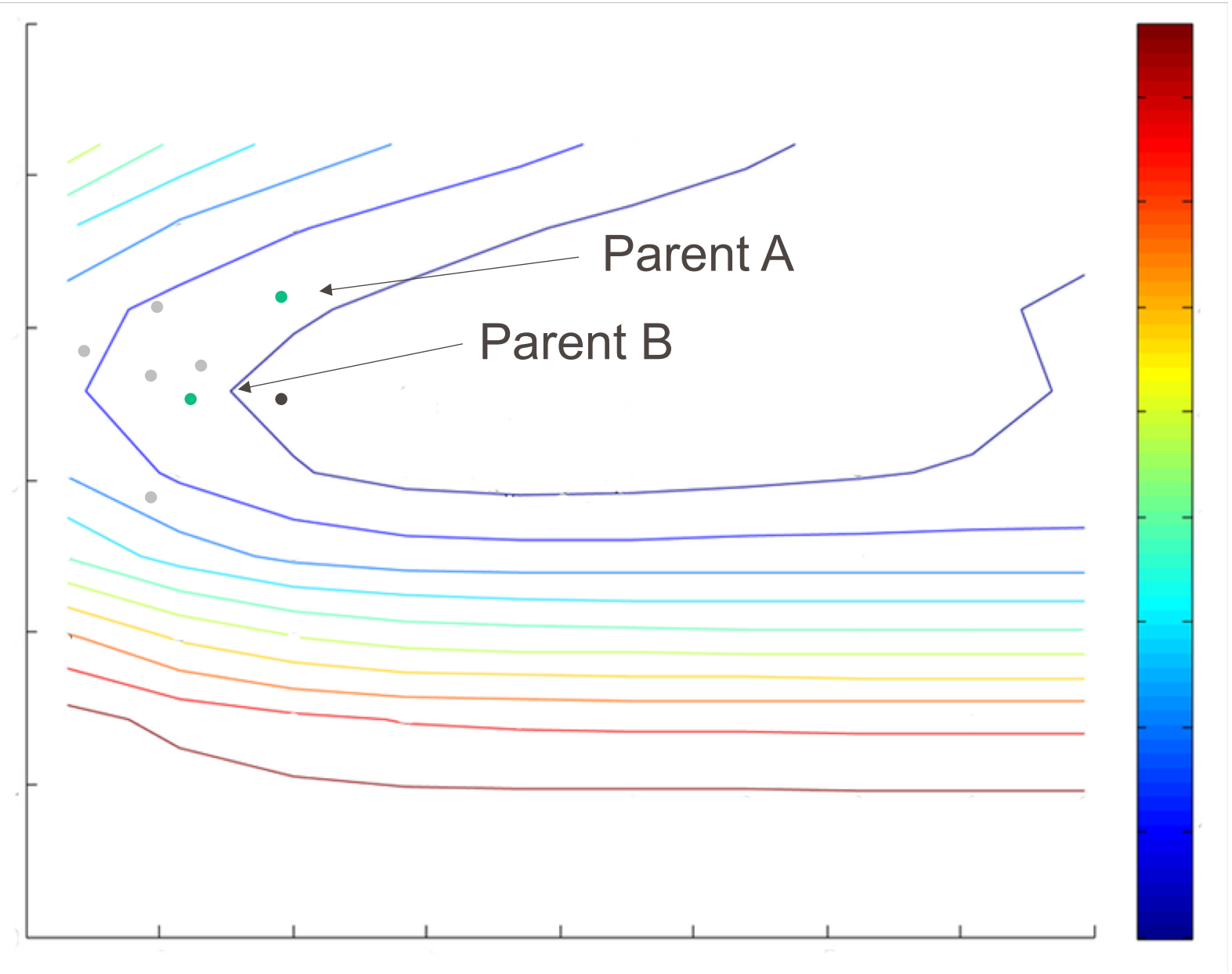


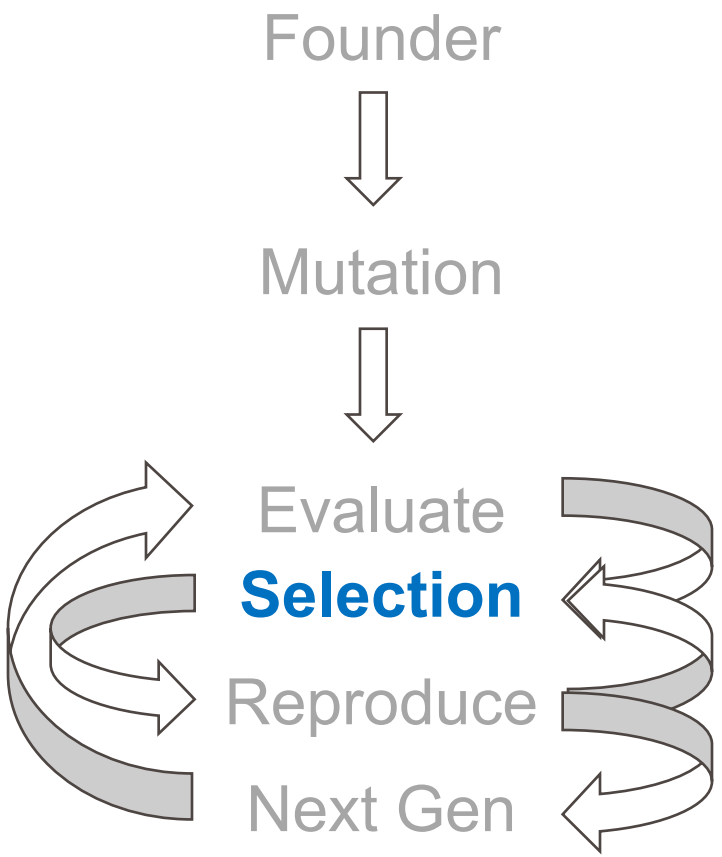
Genetic Search: Reproduction 1



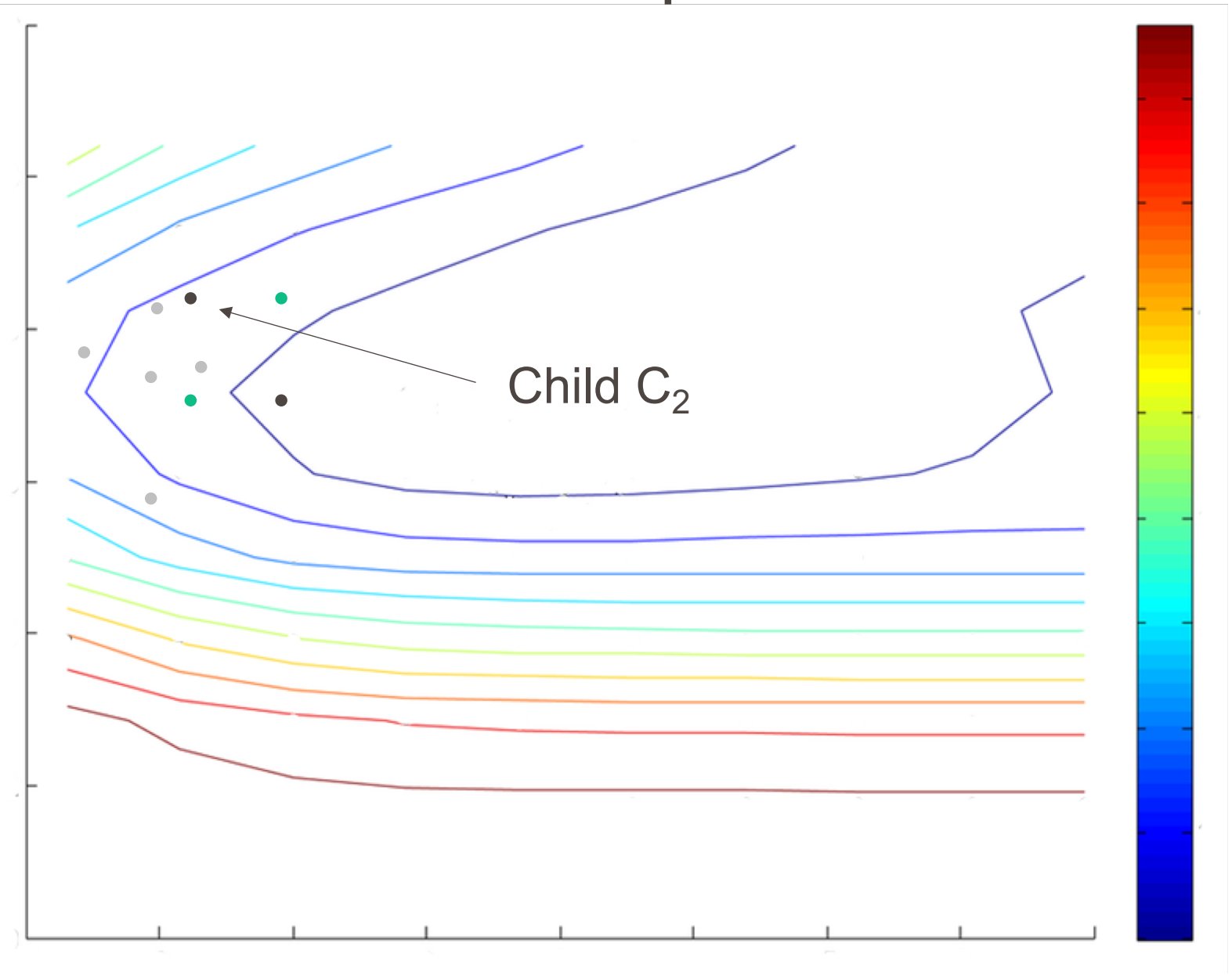


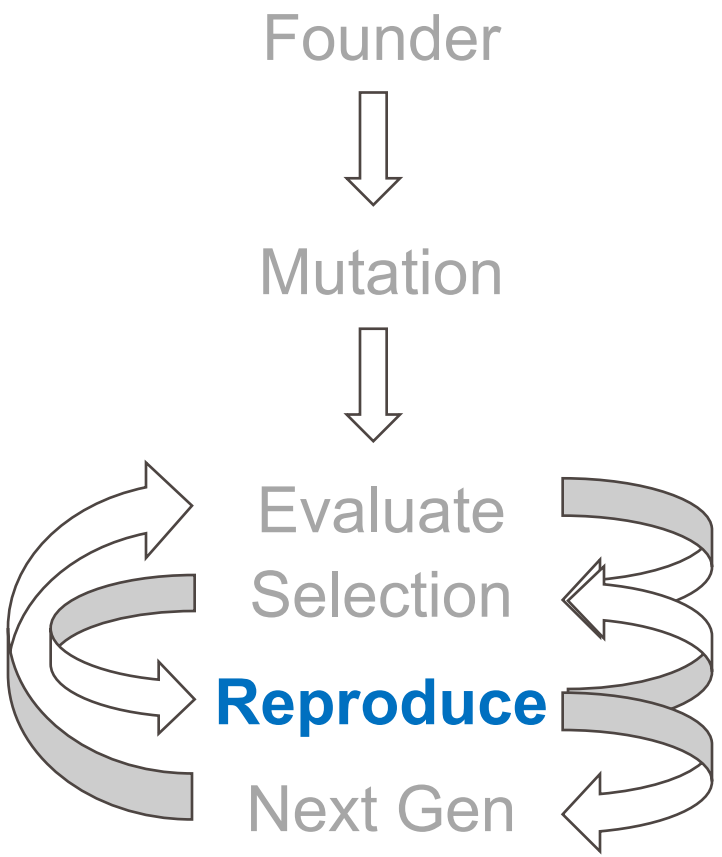
Genetic Search: Mate Selection 2



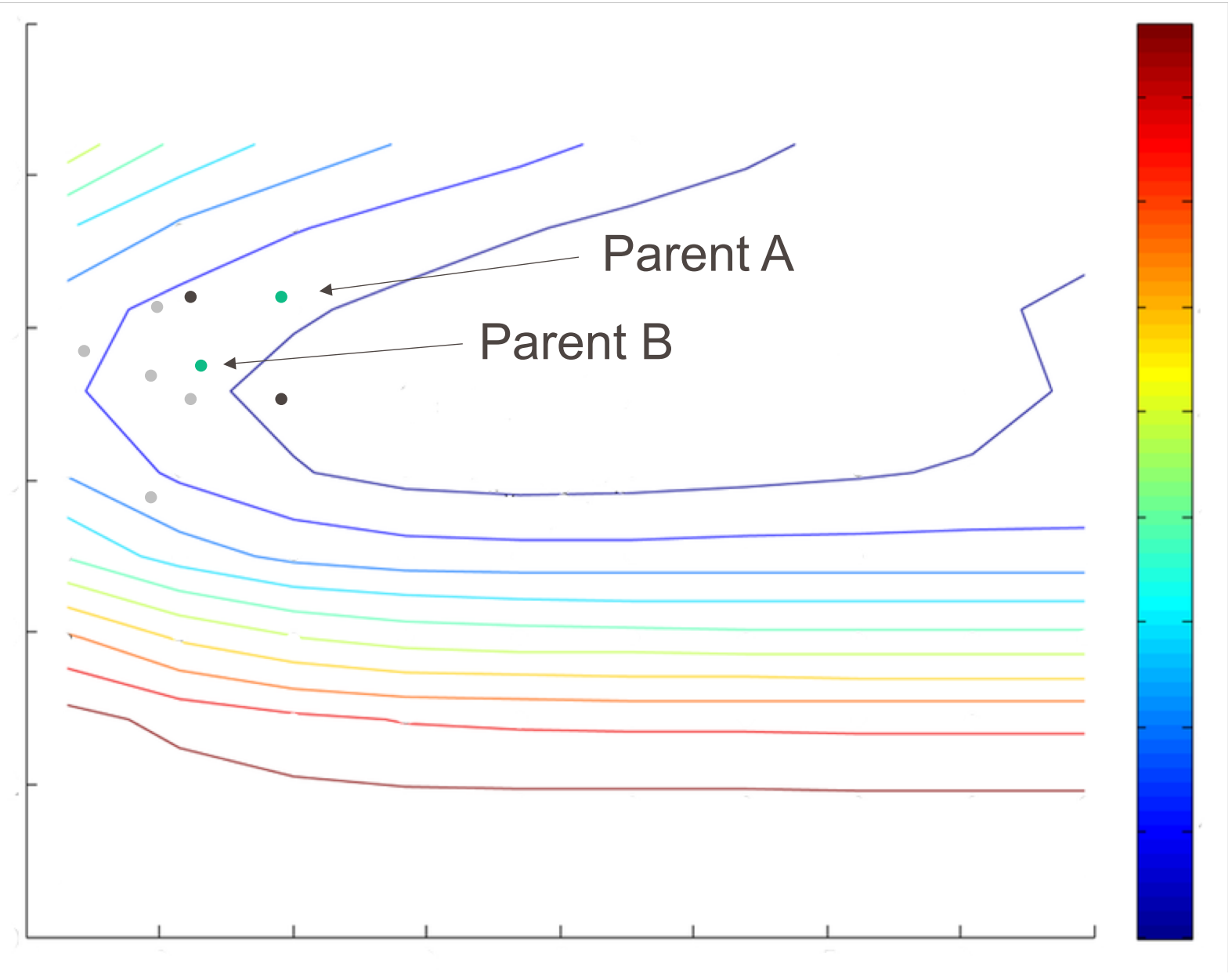


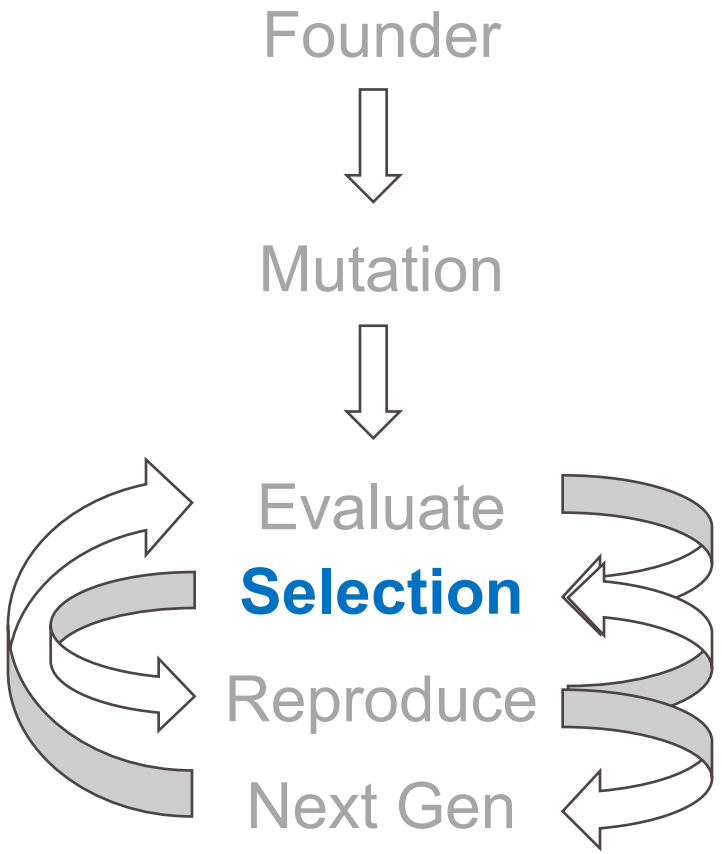
Genetic Search: Reproduction 2



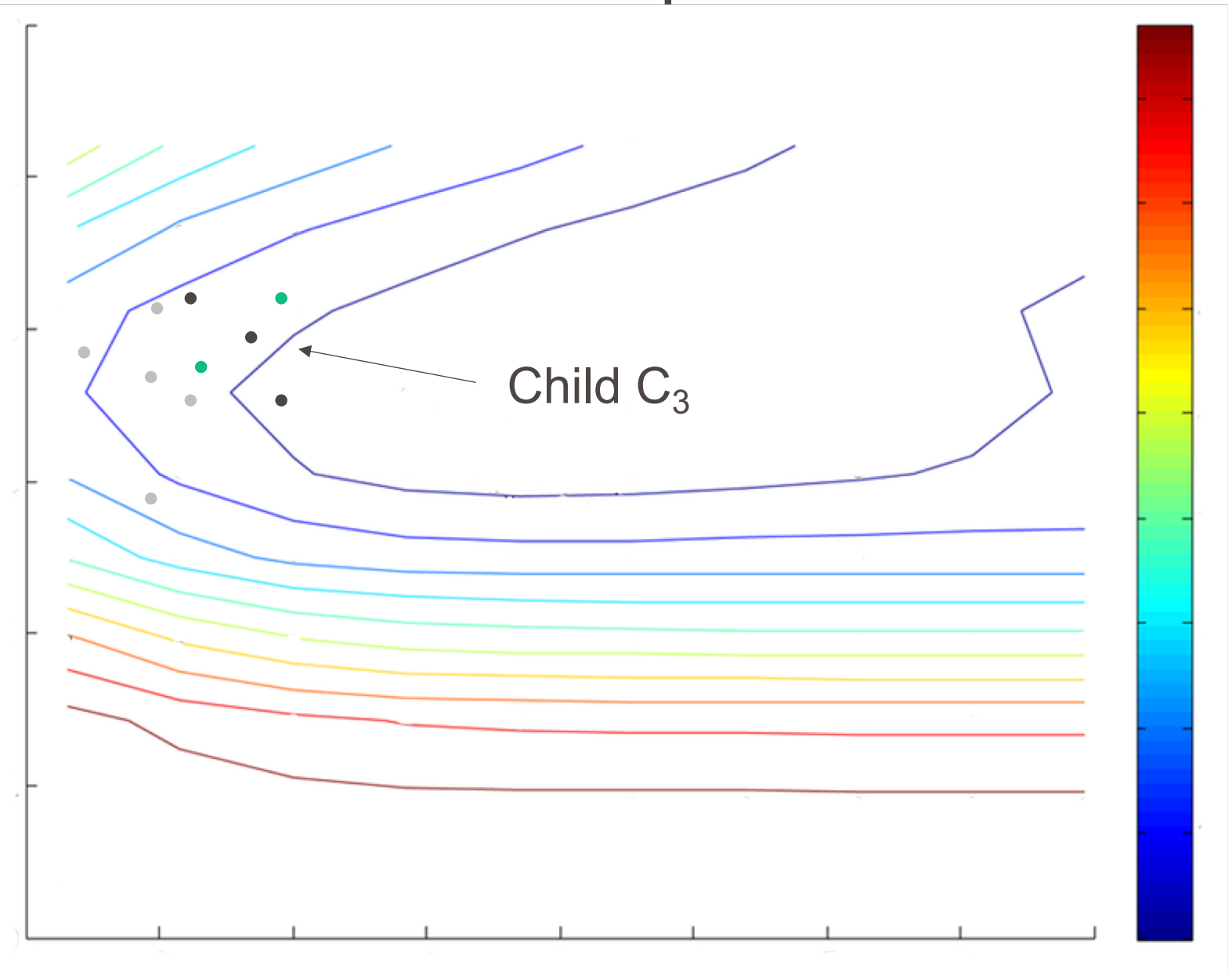


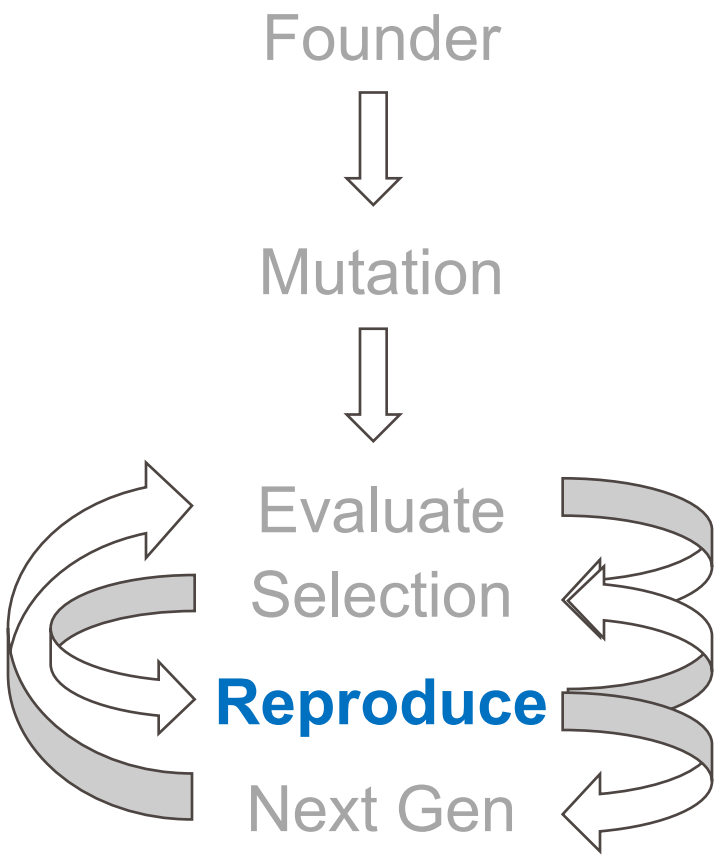
Genetic Search: Mate Selection 3



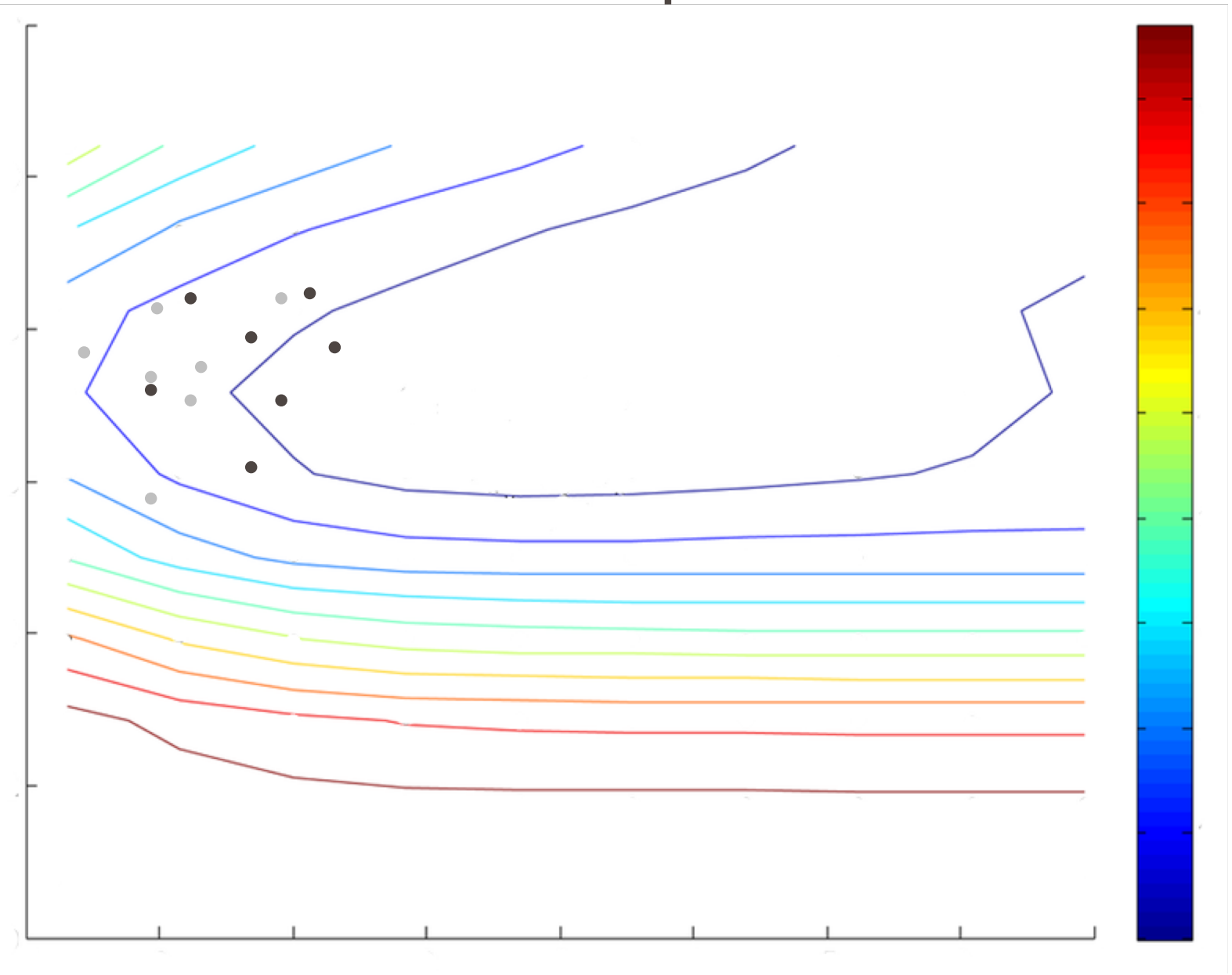


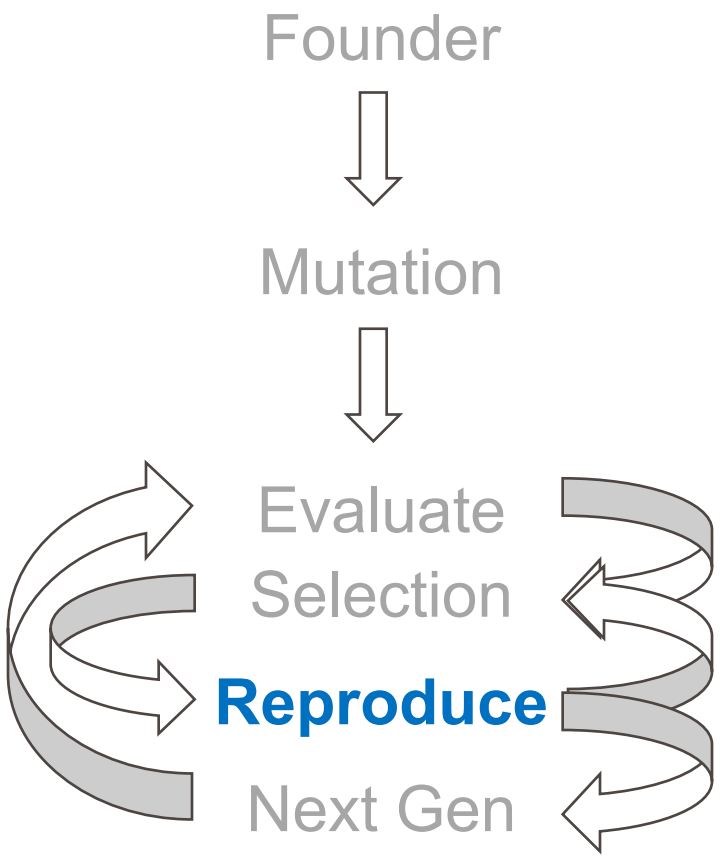
Genetic Search: Reproduction 3



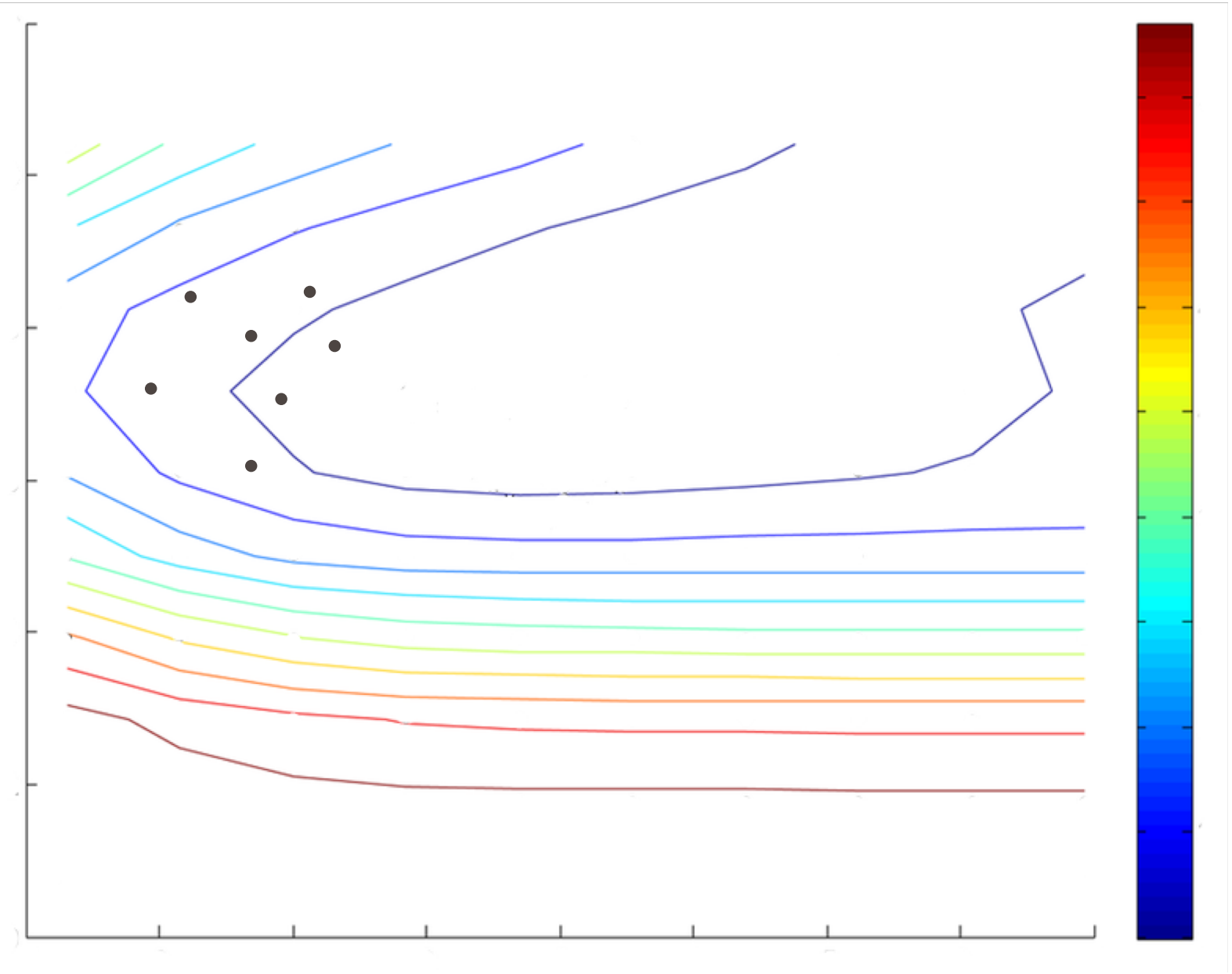


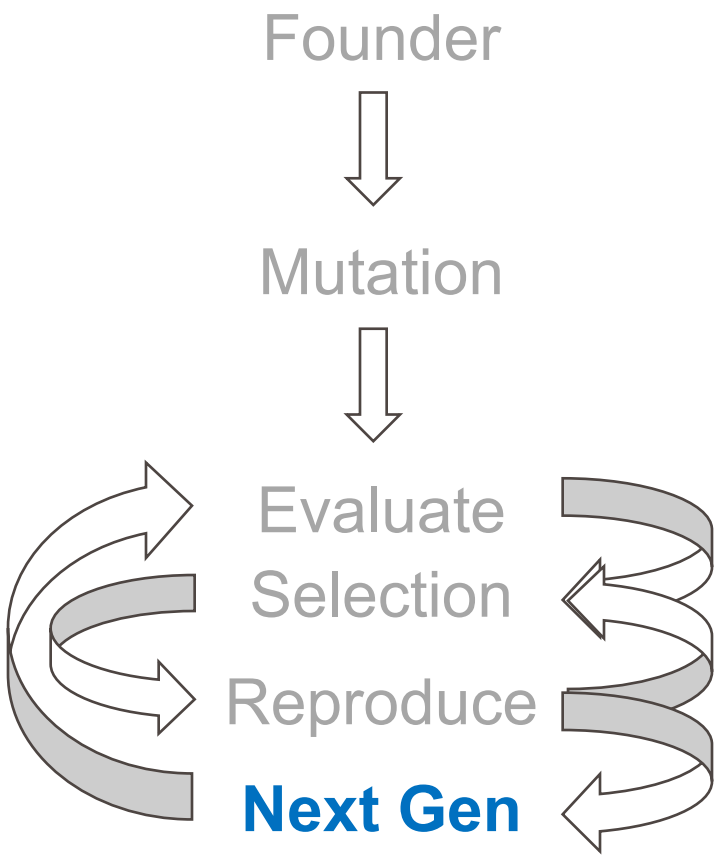
Genetic Search: Reproduction N



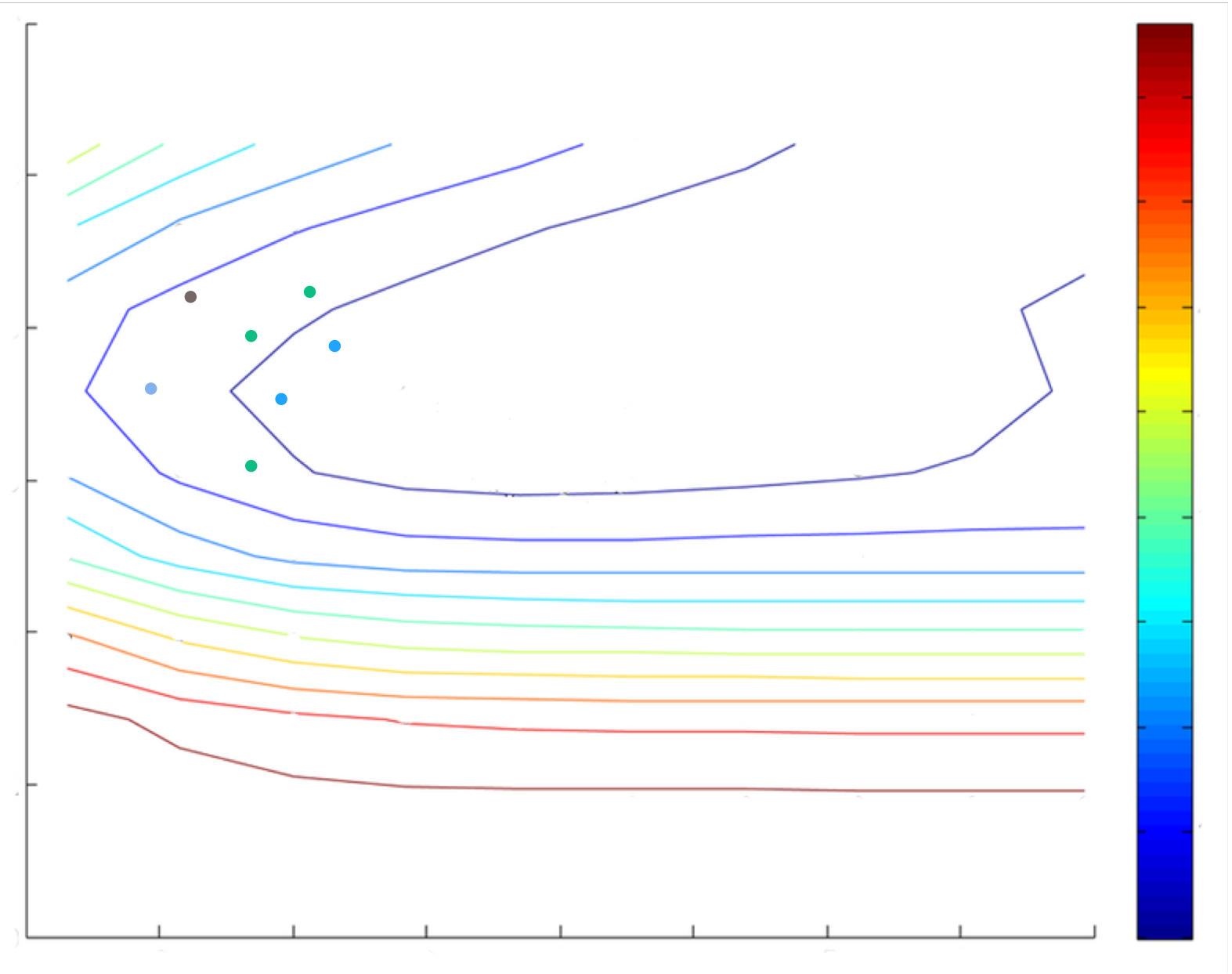


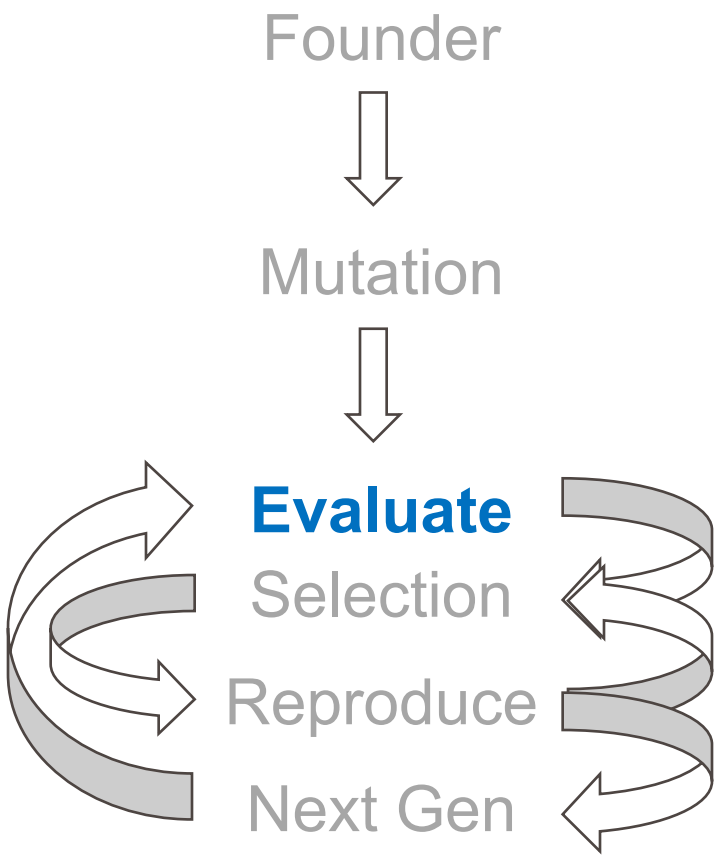
Genetic Search: Next Generation



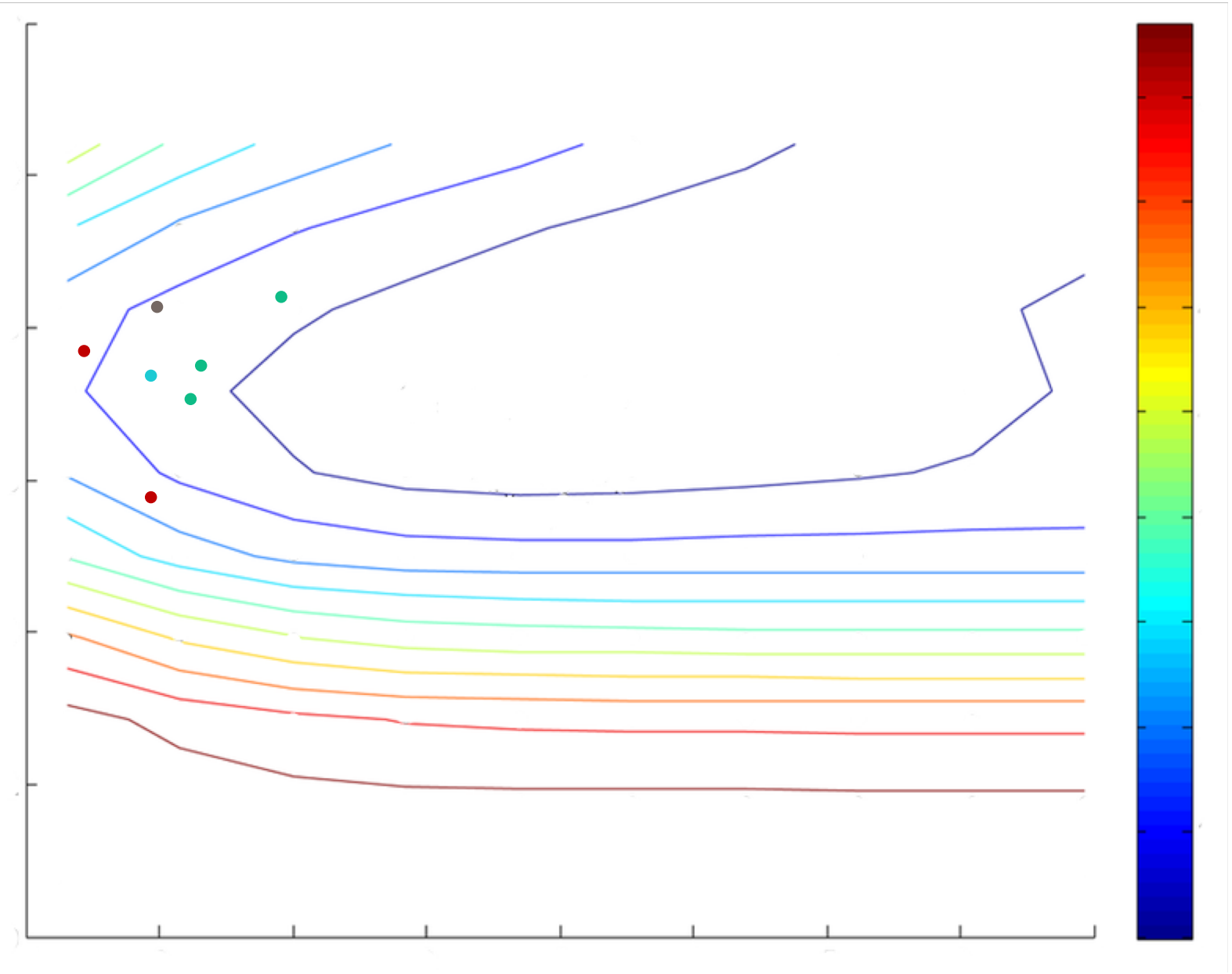


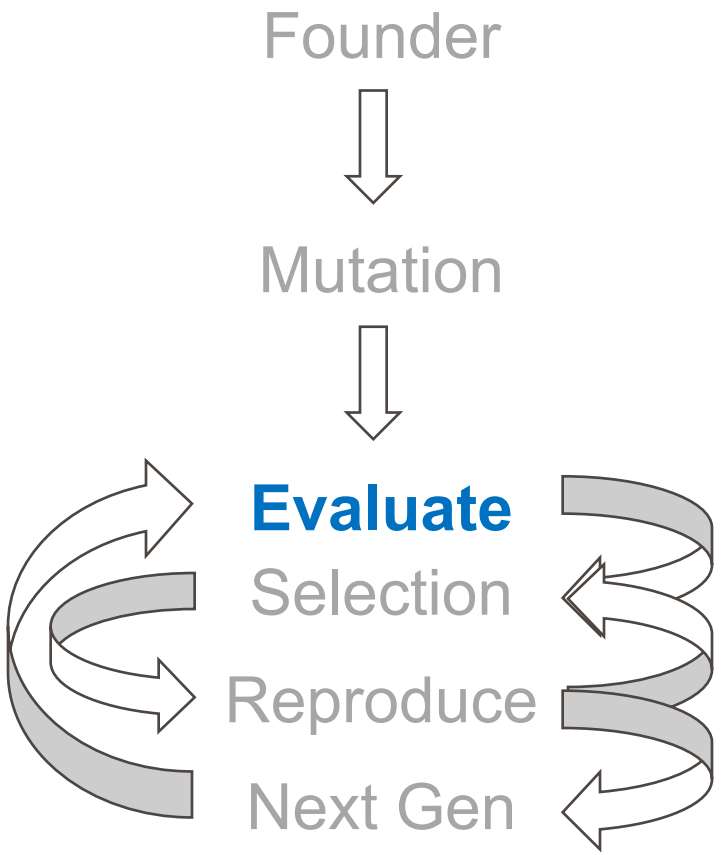
Genetic Search: Evaluate Fitness 2



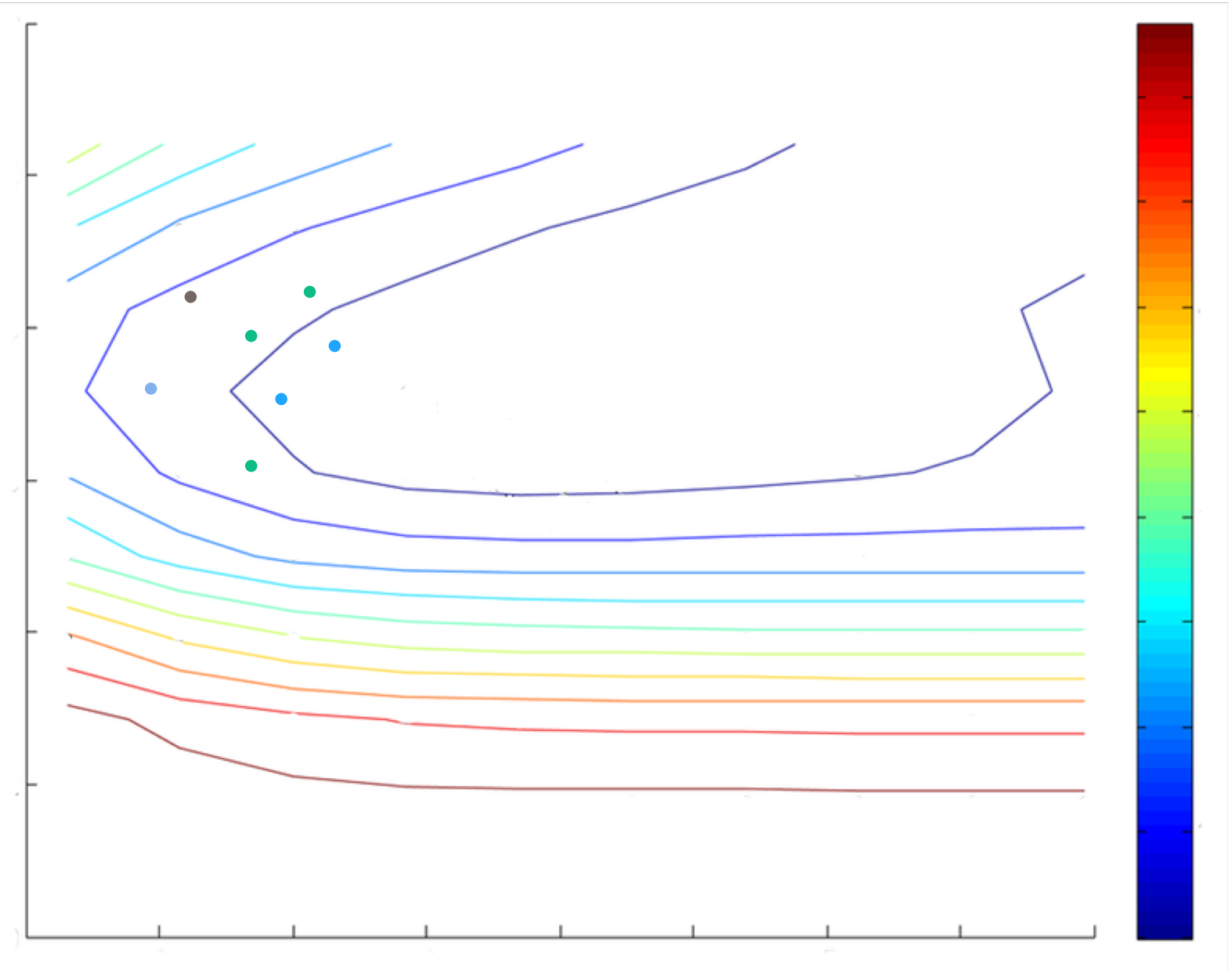


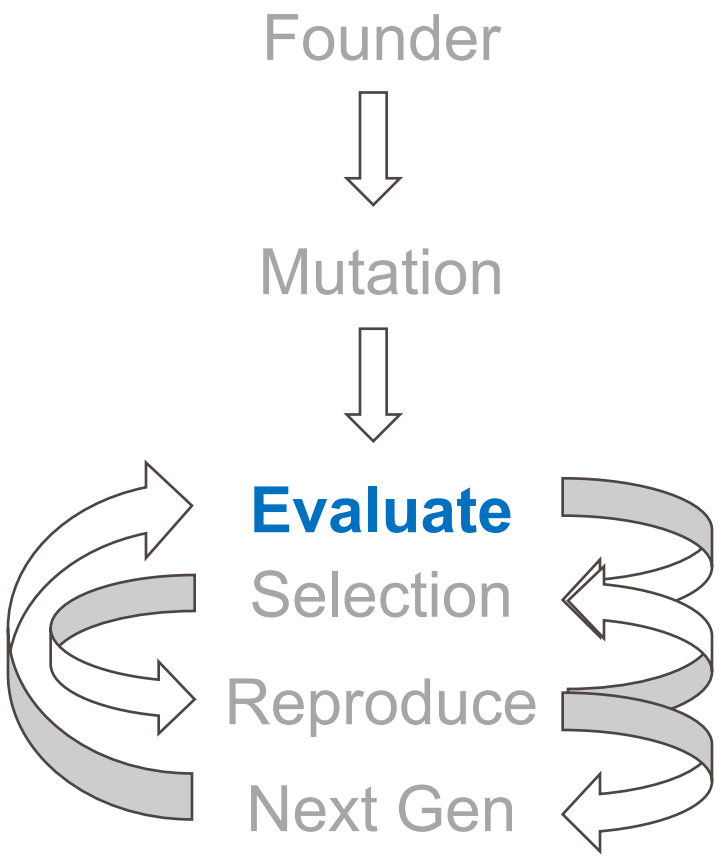
Genetic Search: Evaluate Fitness 1



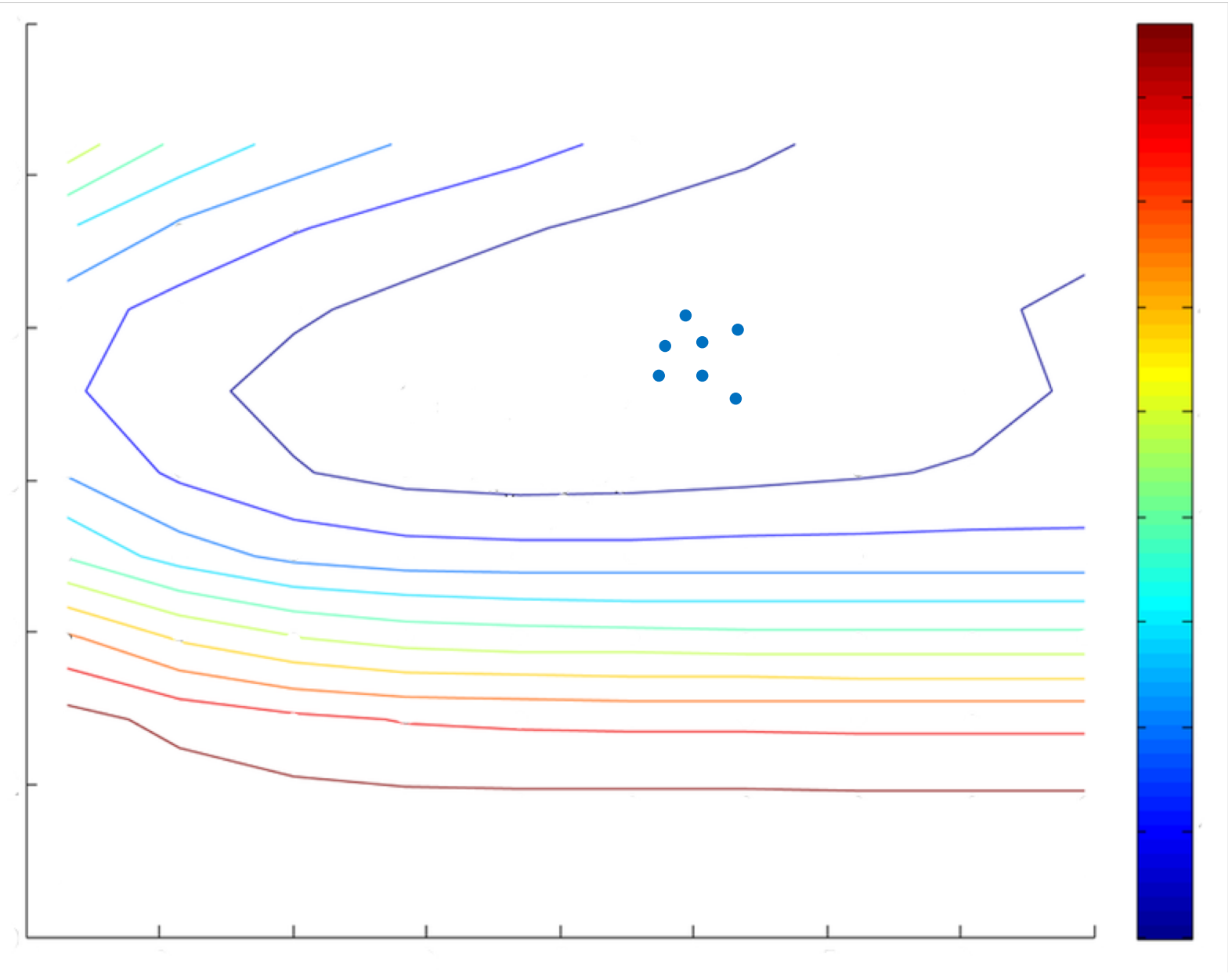


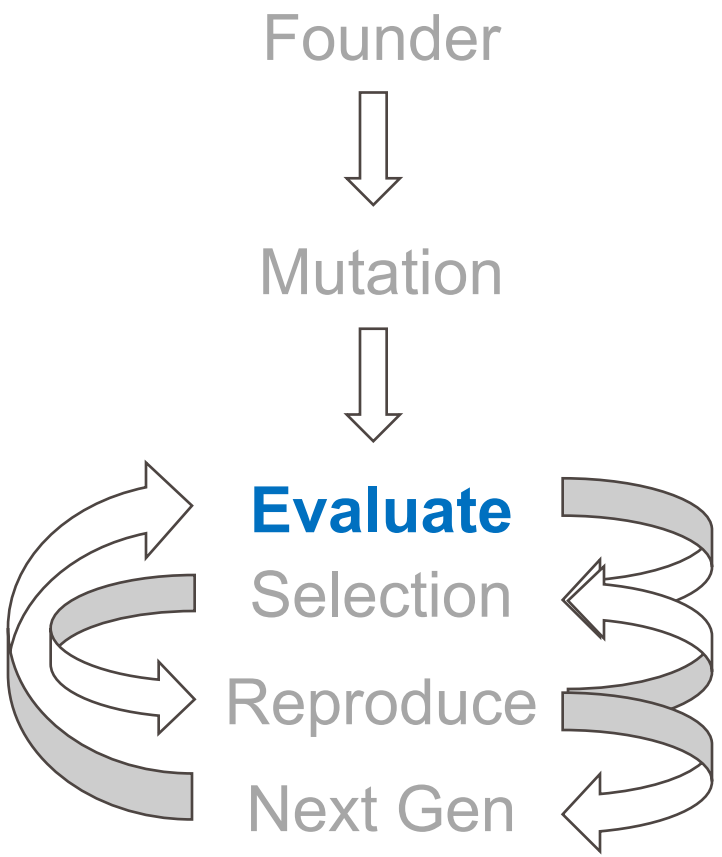
Genetic Search: Evaluate Fitness 2





Genetic Search: Evaluate Fitness N

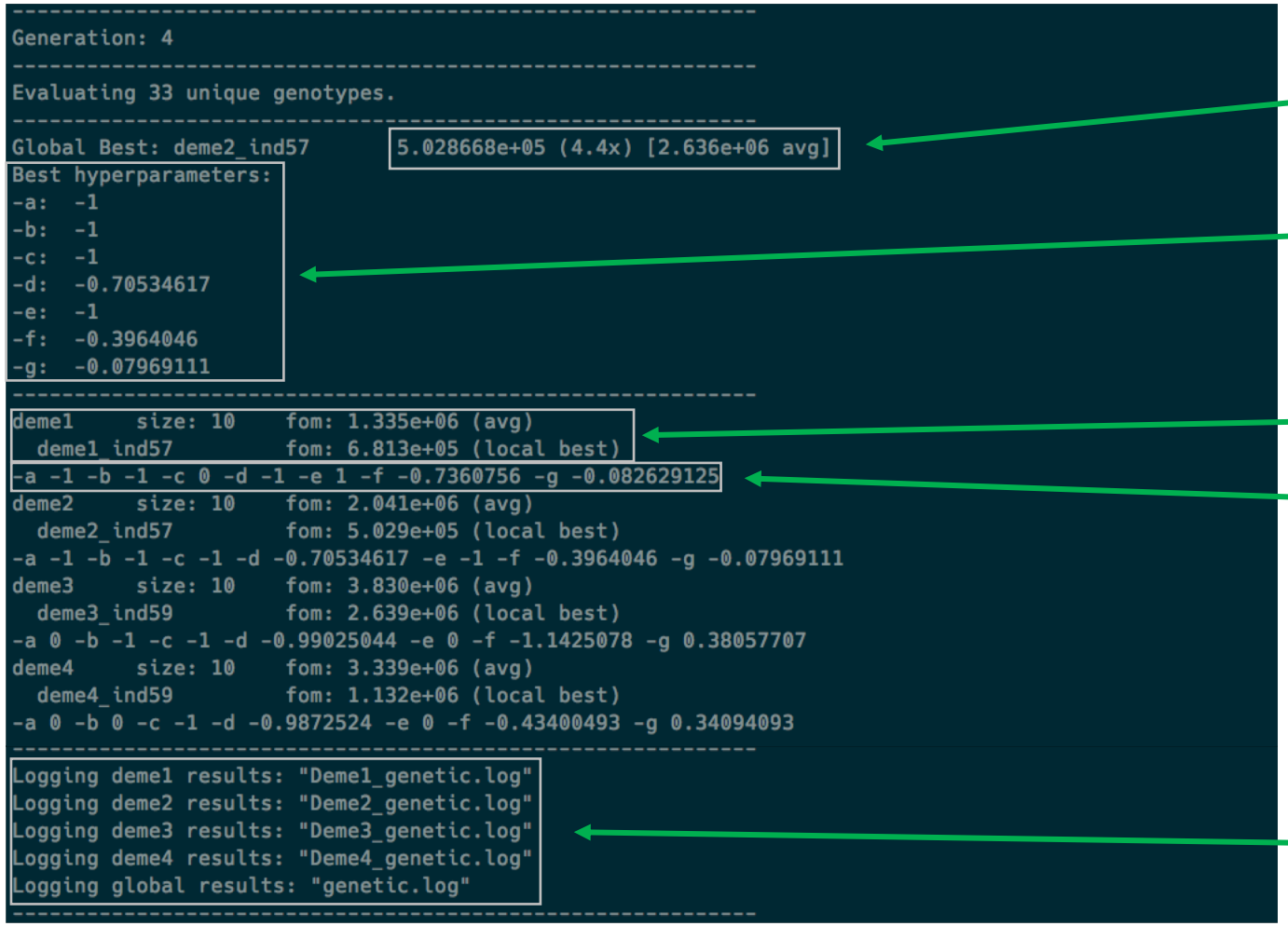




Genetic Search: Example

```
#!/usr/bin/env python3
# encoding: utf-8

from crayai import hpo

evaluator = hpo.Evaluator('python source/train_model.py')

params = hpo.params([["--lr",  0.001, (1e-5, 0.1)],
                     ["--dropout",  0.5, (0.01, 1)]])

optimizer = hpo.genetic.Optimizer(evaluator,
                                  generations=20,
                                  pop_size=10,
                                  num_demes=4,
                                  log_fn='genetic.log')

optimizer.optimize(params)

print(optimizer.best_fom)
print(optimizer.best_params)

```

Genetic Search: Example Verbose Output



Global best FoM, improvement from original HPs, average

Global best HPs

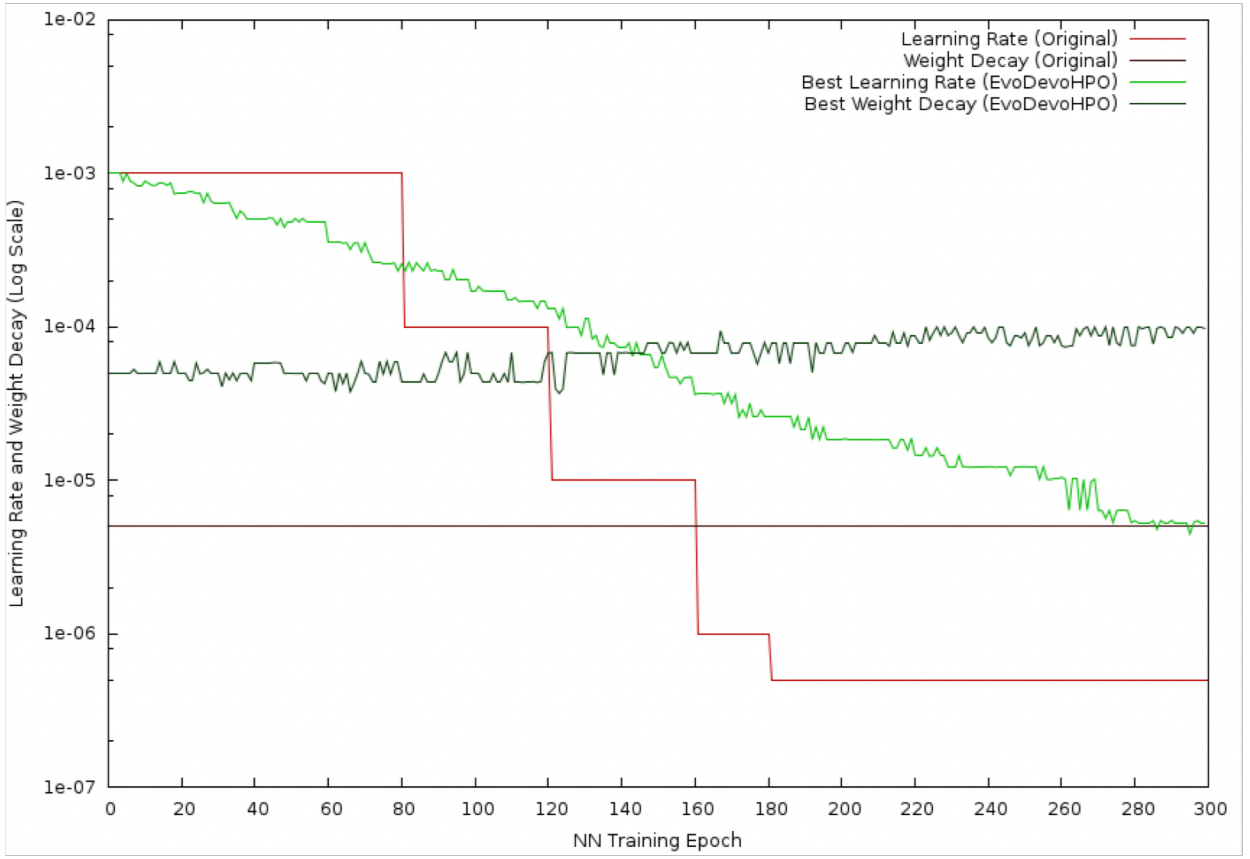
Local best FoM, average per deme

Local best HPs per deme

Results logged per generation

Population-based Training

- Train hyperparameters and parameters simultaneously
  - Apply genetic search to find best set of hyperparameters per *epoch*
  - Results in a *training schedule* adapted to model and data

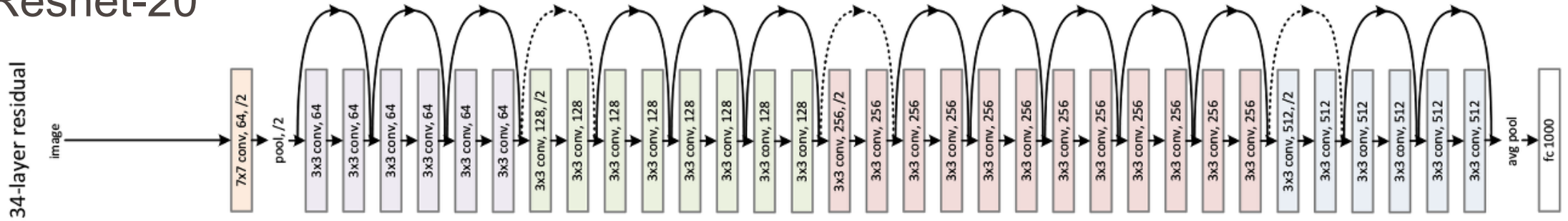


Population-based Training

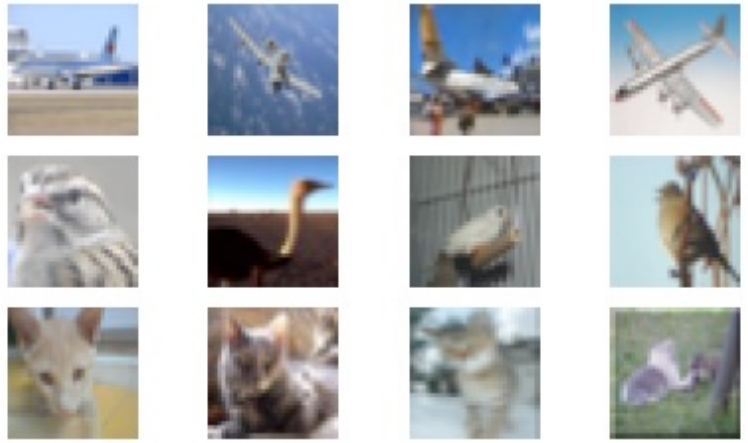
- Originally published from Google DeepMind in November, 2017
  - Not widely available in other HPO frameworks yet
- CrayAI's PBT improves upon DeepMind's PBT (*publication pending*)
  - Reproduction with probabilistic multi-point crossover between 3 parents
    - 2 hyperparameter parents
    - 1 parameter parent
  - Advantages:
    - Increased speed of adaptation
    - Increased ability to shed deleterious genes from population
      - Especially helpful for large numbers of HPs

PBT Example1: ResNet-20 on CIFAR-10

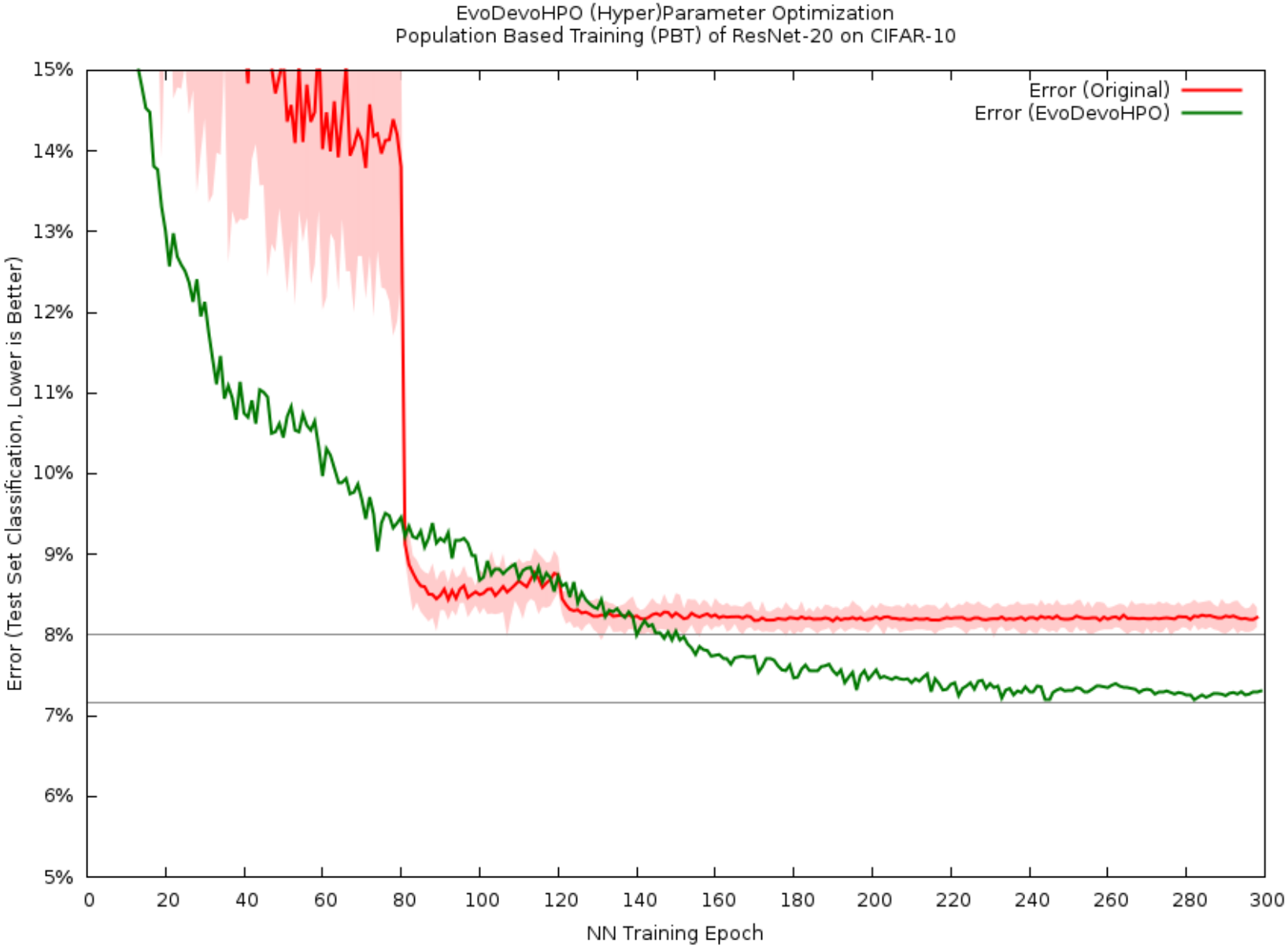
- Resnet-20



- CIFAR-10

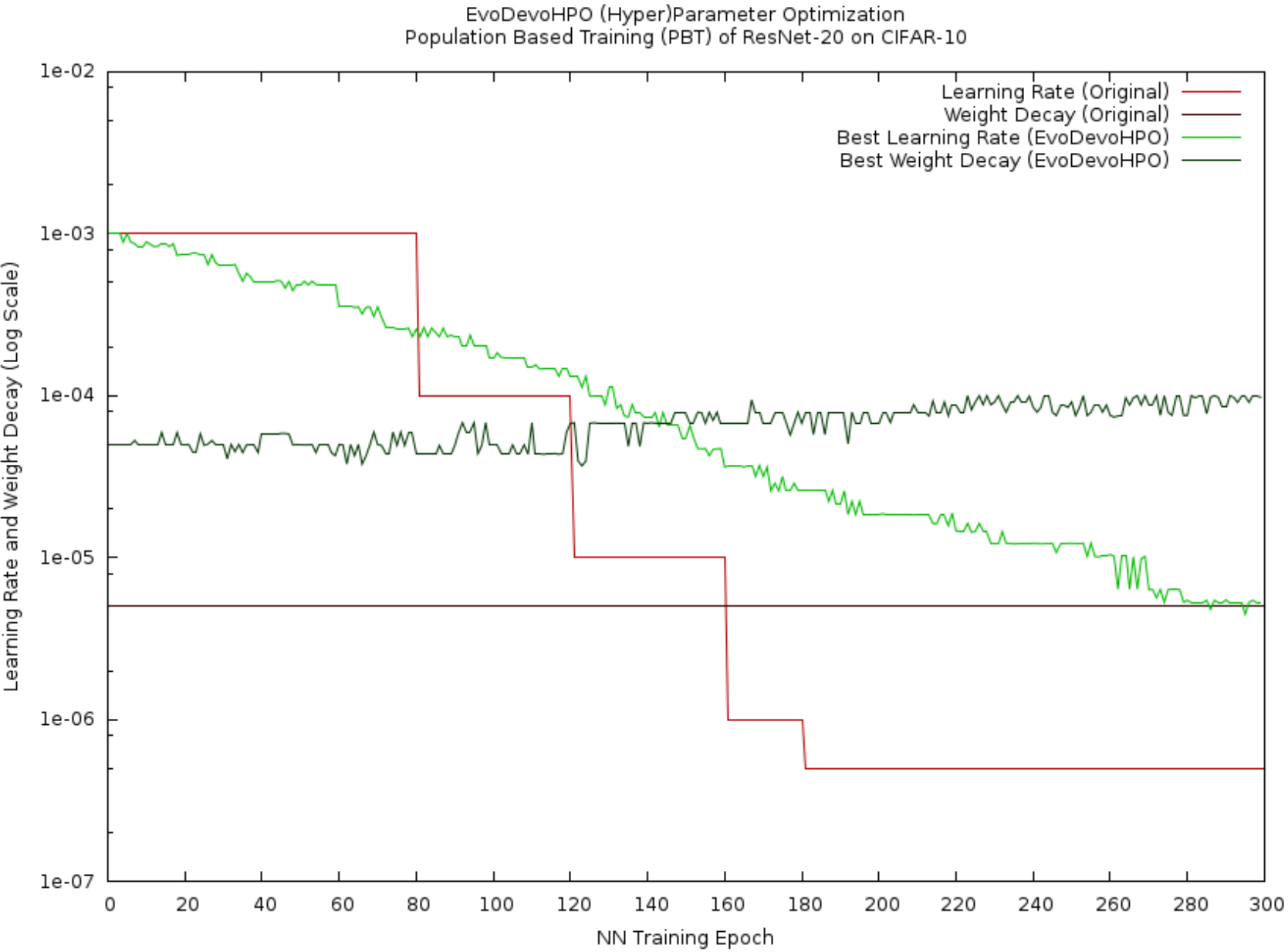


PBT Example1: Results



- 11% reduction in error (relative)
- 103 Cicero nodes (4,944 IVB cores)
- Runtime:
  1 epoch in ~5 min

PBT Example1: Training Schedule



- Discovered improved training schedule
- Optimizes both:
  - Learning rate
  - Weight decay

PBT Example1: Training Schedule

- Discovered improved training schedule
- Optimizes both:
  - Learning rate
  - Weight decay

Distribution

- Current distribution mechanism utilizes system workload managers
  - Existing strategies are trivially parallellizable
  - Easy to support across Cray systems
  - Supports launching multiple distributed model trainings within an allocation
- Launching schemes available to users:
  - local
  - Slurm
    - given a slurm job ID or from an existing salloc
  - Urika
    - Must be on an existing allocation, supports slurm, pbs, moab torque


Status and Next Steps

Status

- Available in Urika XC 1.2 and Urika CS 1.1
  - Accessible through analytics module or crayai module:

```
> module load crayai
> module load analytics
```

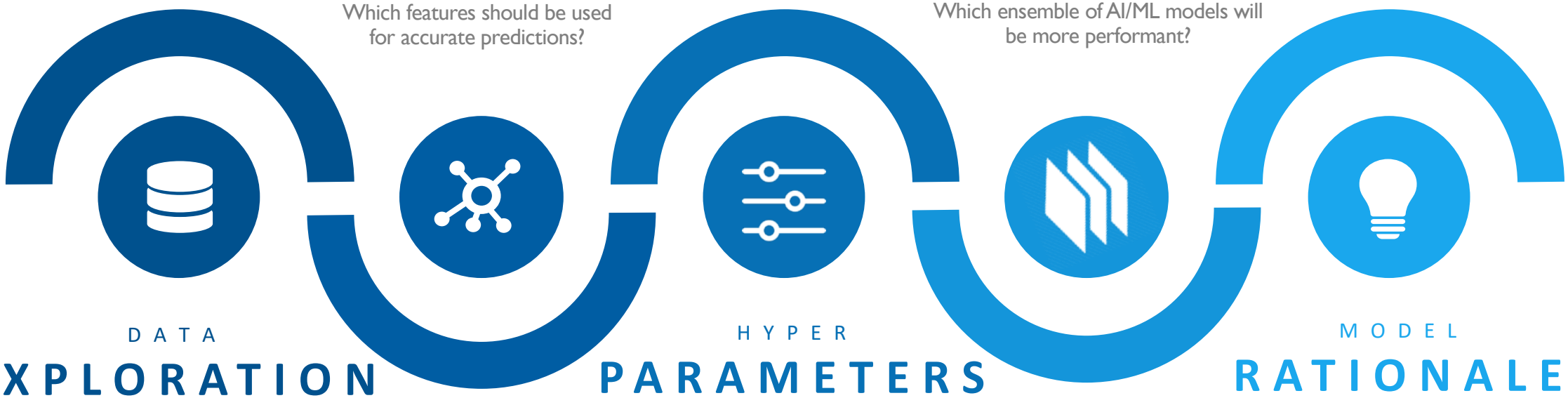
- Being tested early by users
- Updates to documentation
- Actively addressing user feedback

Next Steps

- Continue improving HPO features
  - More options for configuring a given optimizer or evaluator
  - Distinct PBT module
  - Improve workload manager interface
- Implement new strategies
  - Bayesian
- Lower the barrier to contributing new HPO strategies
  - Solidify and document interface between internal objects
  - Allow anyone to easily contribute their own HPO strategy

DISCOVERY **ROADBLOCKS**

Data Science Pain Points



DATA
EXPLORATION
Do I fully understand my data? Does it need to be cleaned?

SELECTING
FEATURES
Which features should be used for accurate predictions?

HYPER
PARAMETERS
What are the correct values to set the variables to before training?

MODEL
ENSEMBLES
Which ensemble of AI/ML models will be more performant?

MODEL
RATIONALE
Do I trust my model? Why does it predict that way?

References

- Population Based Training of Neural Networks

  https://arxiv.org/pdf/1711.09846.pdf
- Random Search for Hyper-parameter Optimization

  http://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf
- The MNIST Database of Handwritten Digits (MNIST Dataset)

  http://yann.lecun.com/exdb/mnist/
- Recombination of Artificial Neural Networks

  https://arxiv.org/abs/1901.03900
- Gradient Based Learning Applied to Document Recognition (LeNet CNN)

  http://www.dengfanxin.cn/wp-content/uploads/2016/03/1998Lecun.pdf

Acknowledgements

- Aaron Vose – early development of EvoDevo
- HPO framework
  - Ben Albrecht
    - balbrecht@cray.com
  - Alex Heye
    - aheye@cray.com

HPO Hands-on Exercises

NERSC Cori system

Hands-on examples

- On Cori:
  - /global/cscratch1/sd/kristyn/CUG2019/crayai_hpo/README.cori
  - /global/cscratch1/sd/kristyn/CUG2019/crayai_hpo/examples
- Example 1: HPO with Topology (Tensorflow)
  - Traditional approach using simultaneous optimization of hyperparameters controlling NN topology and training hyperparameters
- Example 2: HPO with PBT (Tensorflow)
  - Population-based training example with genetic search
  - Generating the learning rate schedule
- Example 3: HPO with the Cray ML PE Plugin (PyTorch)

Example 1: HPO with Topology

- Genetic algorithm optimization applied to LeNet-5 (MNIST) Tensorflow example
- Optimization of hyperparameters controlling NN topology and training hyperparameters
- LeNet-5 consists of two convolutional layers, each of which is followed by a subsampling layer, and then a pair of fully-connected layers with a final output layer
  - Hyperparameters used for HPO

```
params = hpo.Params([["--dropout",    0.5,     (0.005,  0.9)],
                     ["--momentum",   1.0e-4,  (1.0e-6, 1.0e-2)],
                     ["--c1_sz",      5,       (2, 8)],
                     ["--c1_ft",      32,      (8, 128)],
                     ["--c2_sz",      5,       (2, 8)],
                     ["--c2_ft",      64,      (16, 256)],
                     ["--fullyc_sz",  1024,    (64, 4096)]])
```

SAFE HARBOR STATEMENT

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts.

These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

THANK YOU

QUESTIONS?

Spark and Dask on Cray Systems

Mike Ringenburg

Agenda

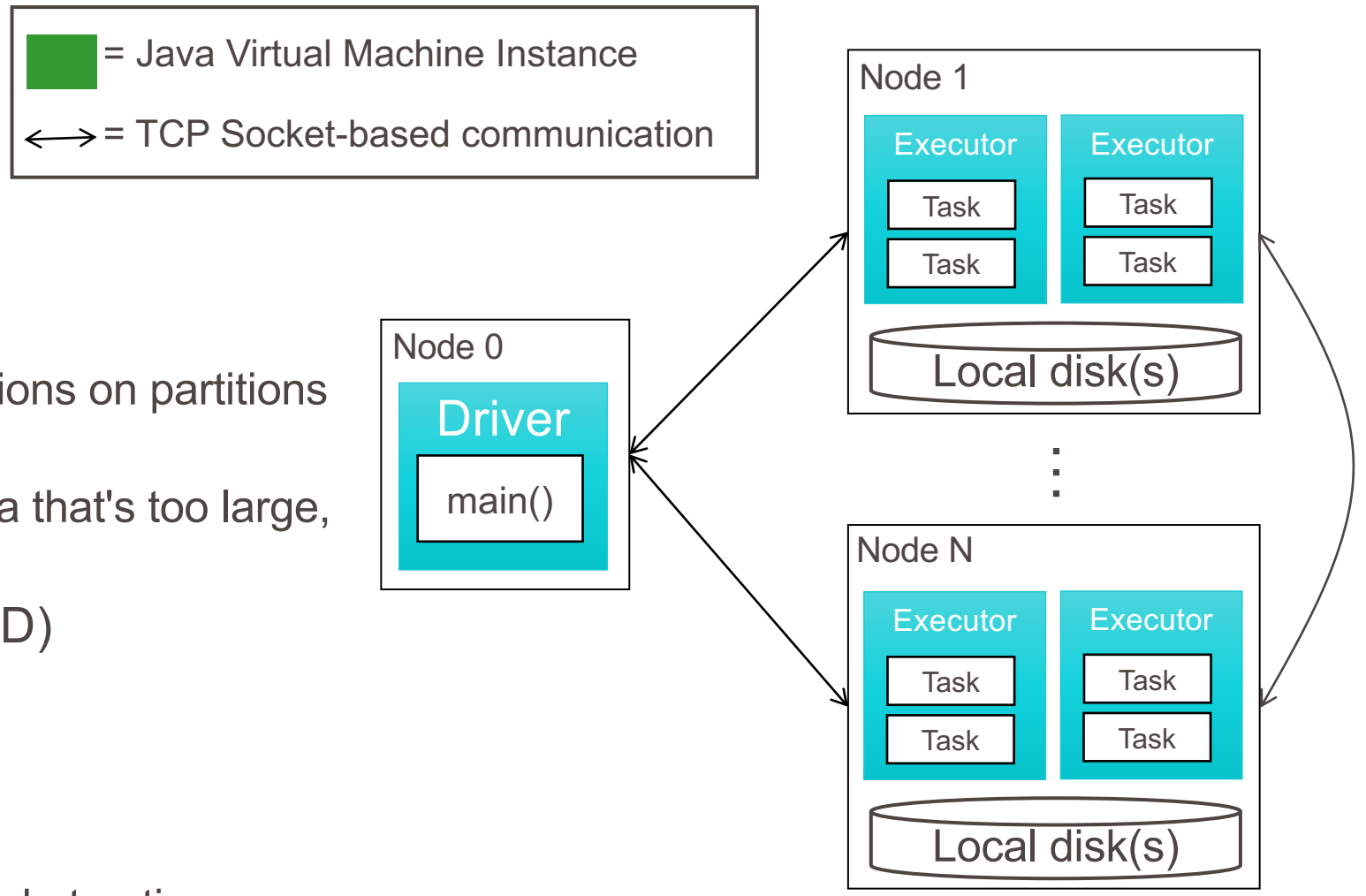
- Introduction to Spark
  - History and Background
  - Computation and Communication Model
- Spark on the XC40
  - Installation and Configuration
  - Local storage

What is Spark?

- Analytics and ML framework released in 2014
  - Originally from Berkeley AMPLab/BDAS stack, now Apache project
  - Native APIs in Scala. Java, Python, and R APIs available as well.
  - Many view as successor to Hadoop MapReduce
- Aimed to address some shortcomings of Hadoop MapReduce
  - More programming flexibility – not constrained to one map, one reduce, write, repeat.
  - Many operations can be pipelined into a single in-memory task
  - Can "persist" intermediate data rather than regenerating every stage

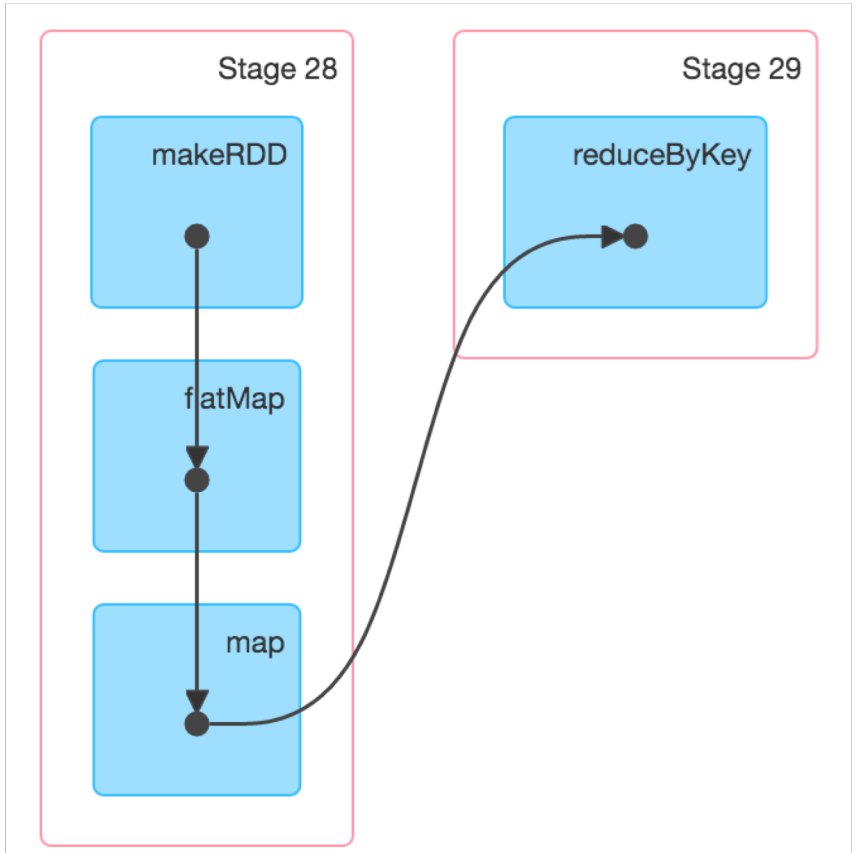
Spark Execution Model

- Master-slave parallelism
- Driver (master)
  - Executes main
  - Distributes work to executors
- Executors (slaves)
  - Lazily execute tasks (local operations on partitions of the RDD)
  - Rely on local disks for spilling data that's too large, and storing shuffle data
- Resilient Distributed Dataset (RDD)
  - Spark's original data abstraction
  - Partitioned amongst executors
  - Fault-tolerant via lineage
  - Dataframes/Datasets extend this abstraction



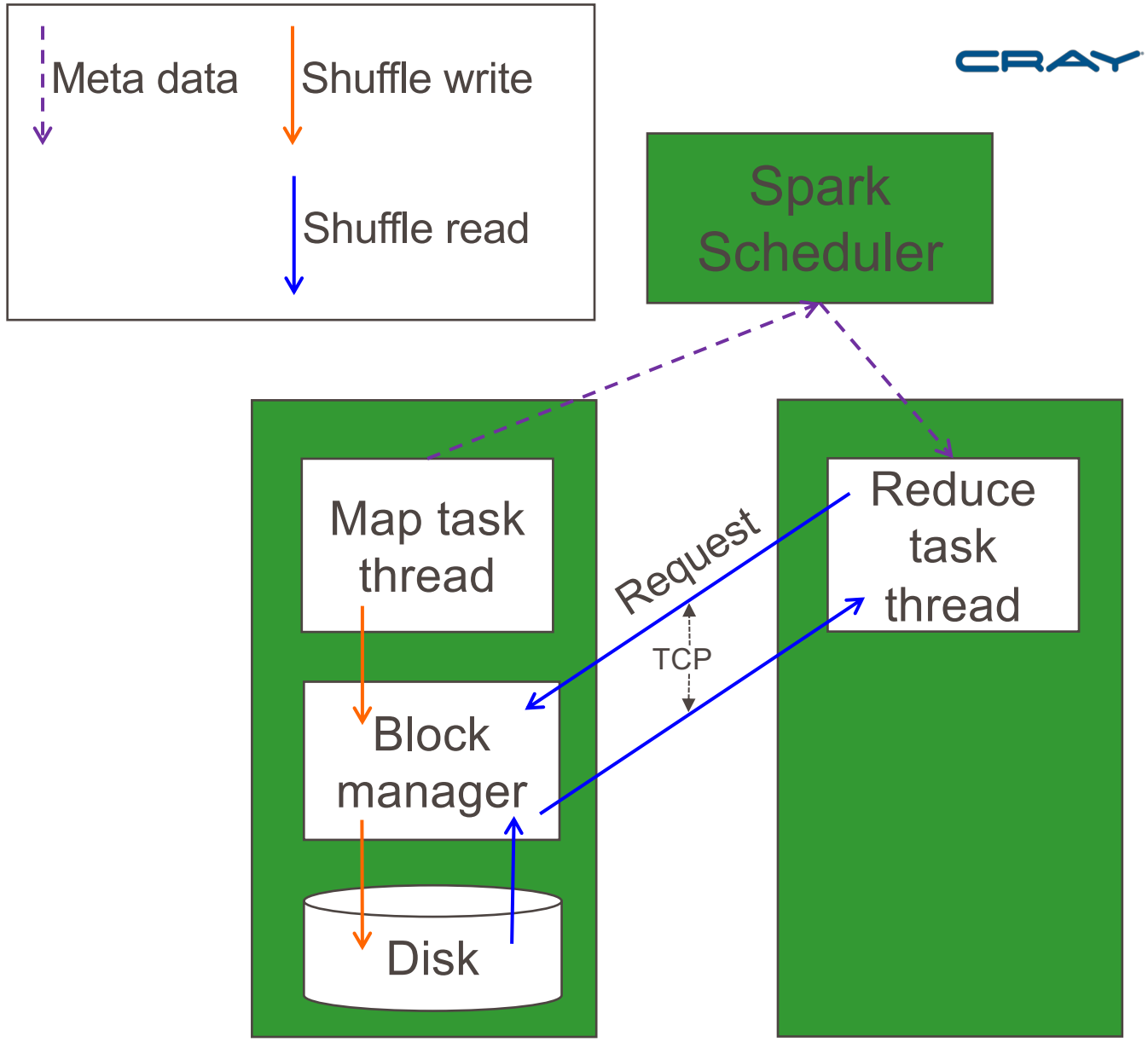
DAGs, Pipelining, and Lazy Evaluation

- Spark is lazily evaluated
  - Operations are only executed when and if needed
  - Needed operations: return result, or parent of needed operation
- Spark DAG (Directed Acyclic Graph)
  - Transformation APIs (operations that produce new RDDs) just add a new node to the DAG, indicating data dependencies and operation
  - Action APIs (return data) trigger execution of DAG nodes
- If an node's dependencies are exclusively on local data from its parent(s), the operations can be *pipelined* into a single *task*
  - *Spark stage:* Execution of task on all RDD partitions
  - Every stage ends with a *shuffle* (all-to-all communication), an output, or returning data back to the driver.
  - Global barrier between stages.



Spark Communication Model (Shuffles)

- All data exchanges between executors implemented via *shuffle*
  - Senders (“mappers”) send data to block managers; block managers write to disks, tell scheduler *how much* destined for each reducer
  - Barrier until all mappers complete shuffle writes
  - Receivers (“reducers”) request data from block managers *that have data for them*; block managers read and send



Spark Example: Word Count

Load file

flatMap maps one value to (possibly) many, instead of one-to-one like map

groupByKey combines all key-value pairs with the same key (k, v1), …, (k,vn) into a single key-value pair (k, (v1, …, vn)).

Collect returns all elements to the driver

More efficient: replace group and sum with reduceByKey

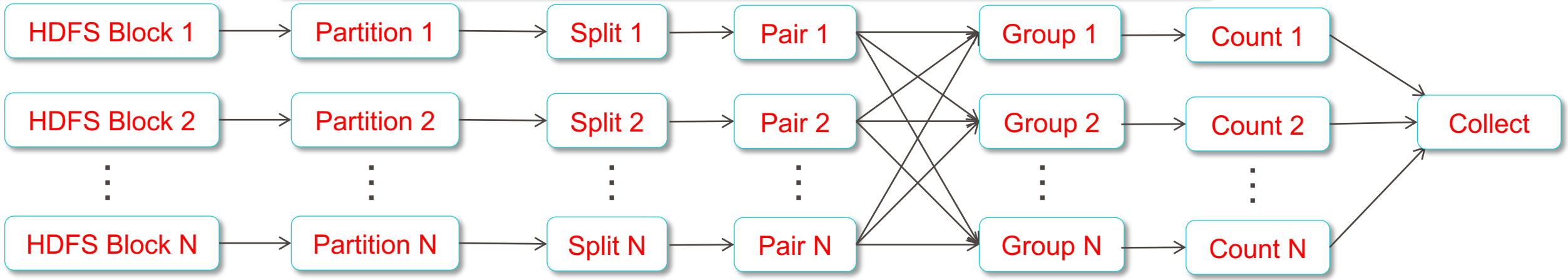
```
val lines = sc.textFile("mytext")
val words = lines.flatMap (
              line => line.split(" ")
            )
val wordKV = words.map(s => (s, 1))
val groupedWords = wordKV.groupByKey()
val wordCounts = groupedWords.map(
                   t => (t._1, t._2.sum)
                 )
val counts = wordCounts.collect()
```

- Let's like at a simple example: computing the number of times each word occurs
  - Load a text file
  - Split it into words
  - Group same words together (all-to-all communication)
  - Count each word

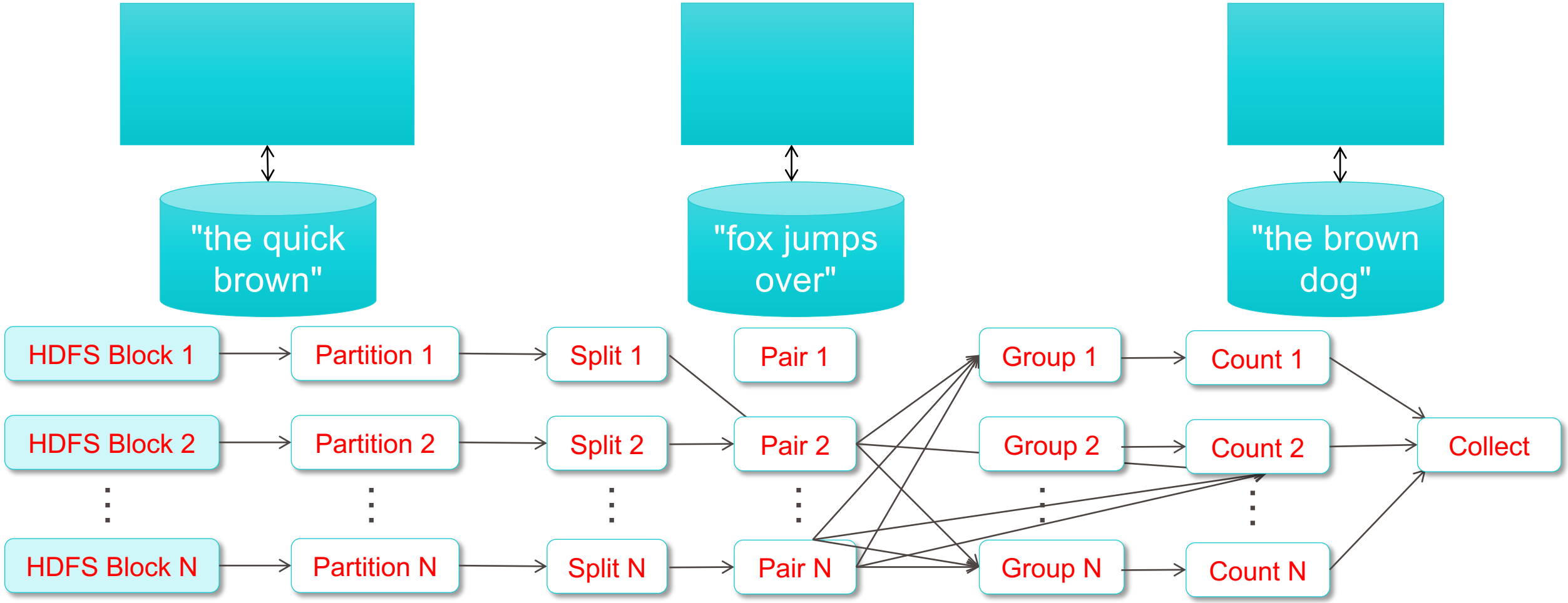
The Spark DAG

```
val lines = sc.textFile("mytext")
val words = lines.flatMap (
                line => line.split(" ")
              )
val wordKV = words.map(s => (s, 1))
val groupedWords = wordKV.groupByKey()
val wordCounts = groupedWords.map(
                   t => (t._1, t._2.sum)
                 )
val counts = wordCounts.collect()
```

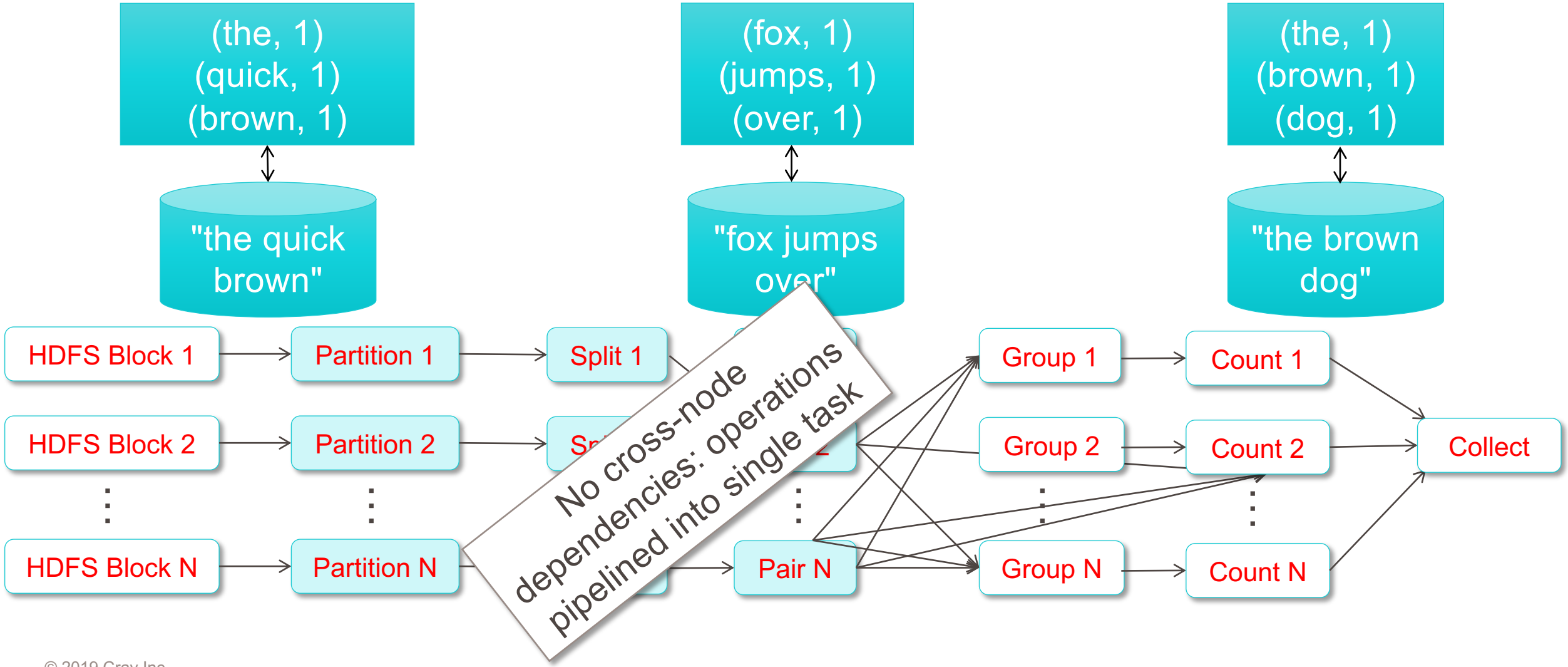
Execute!



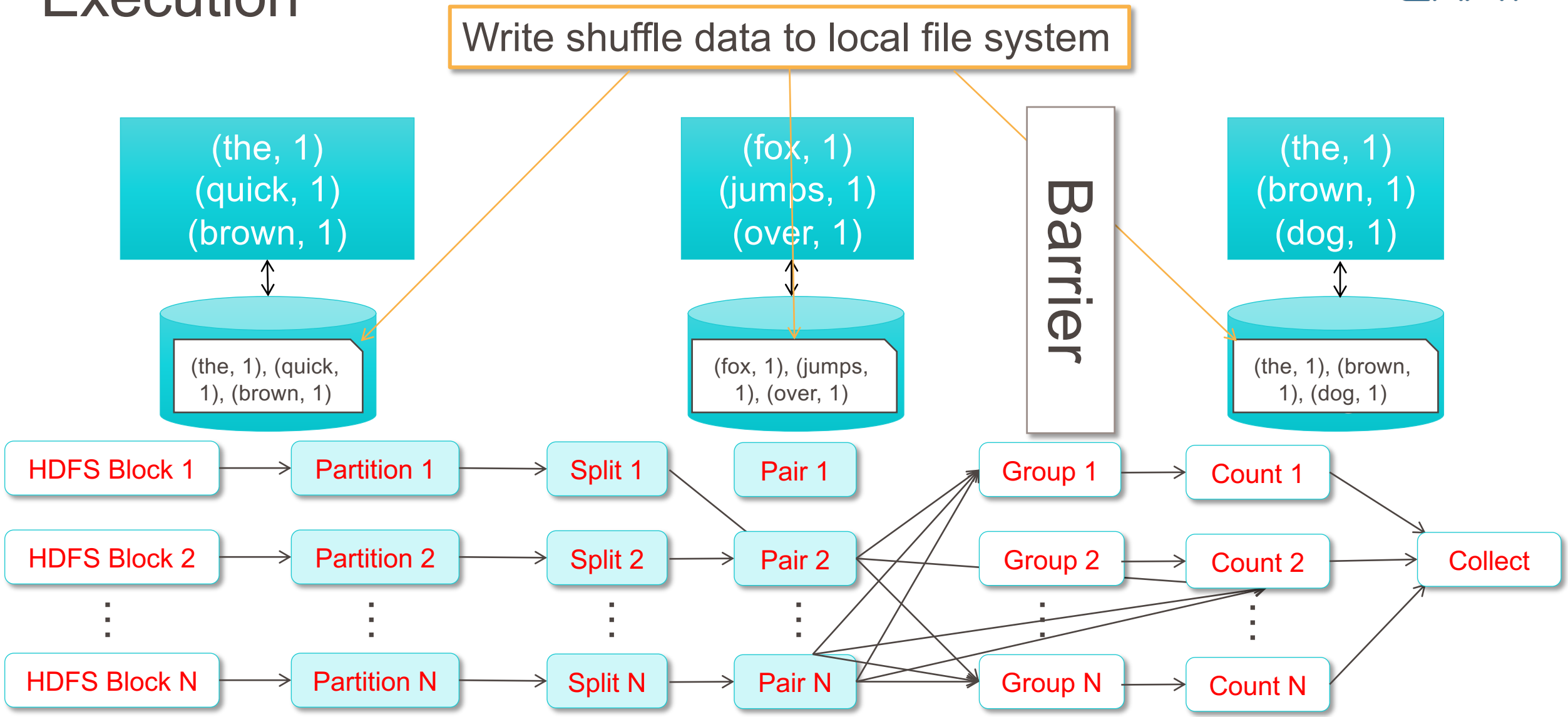
Execution



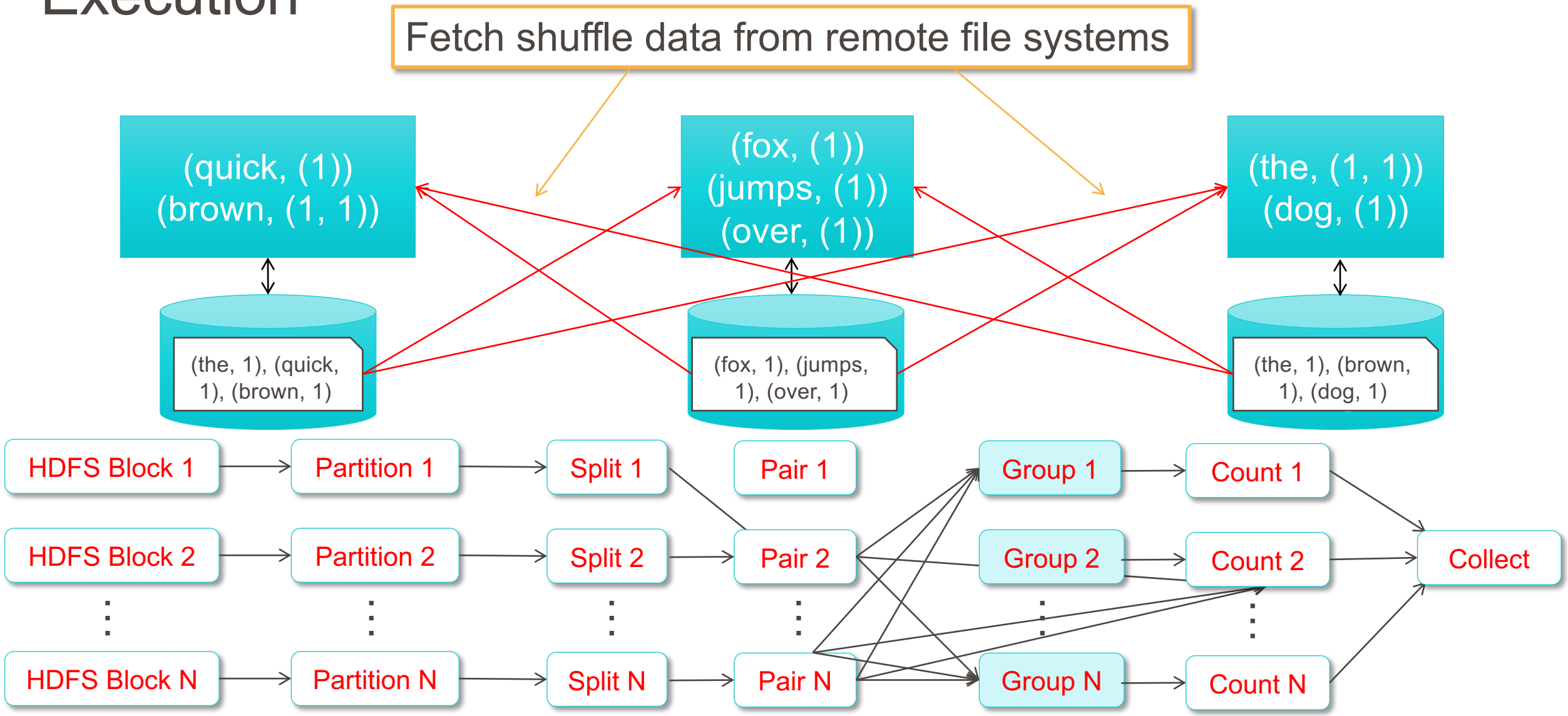
Execution



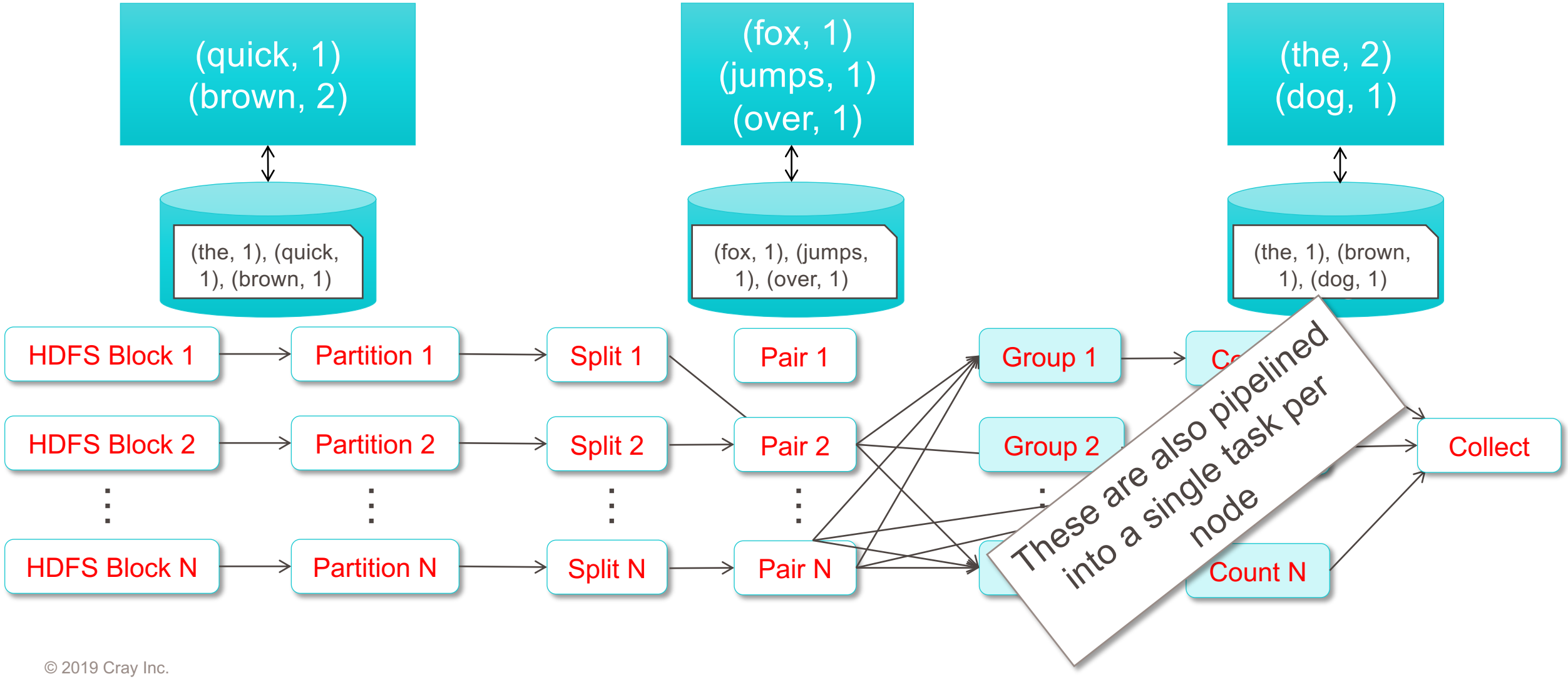
Execution



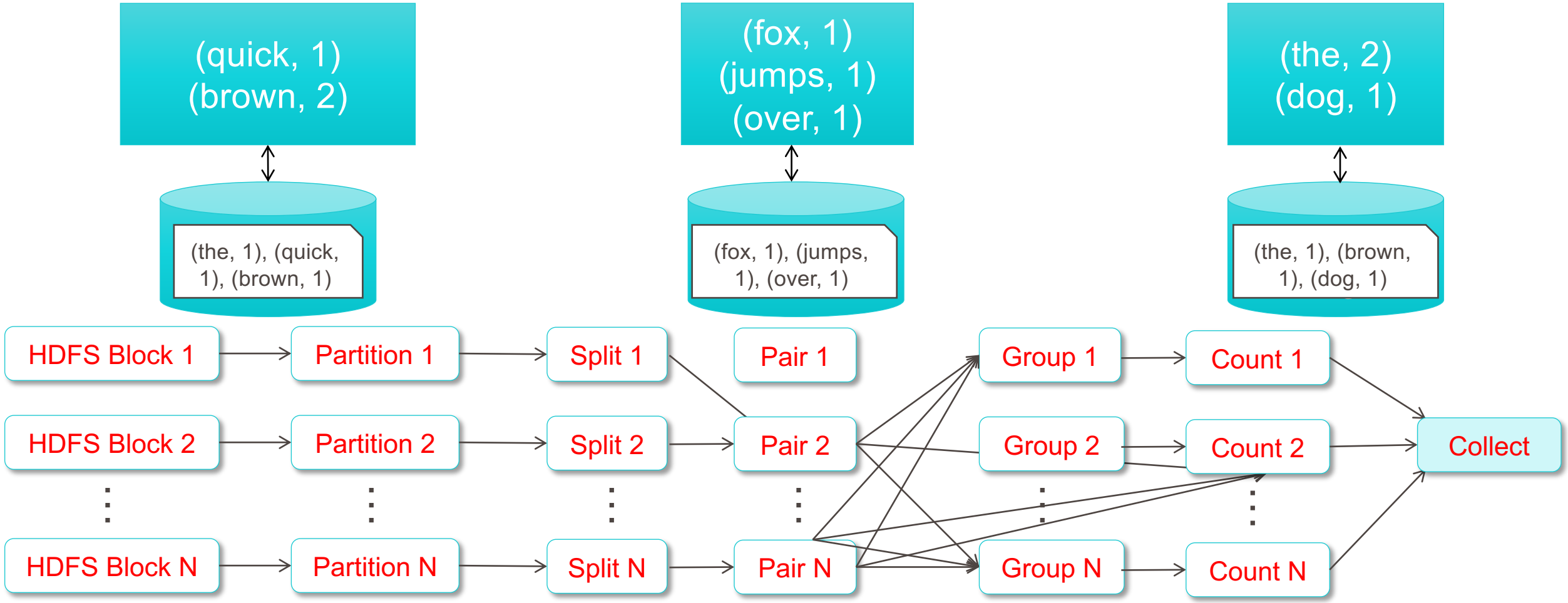
Execution



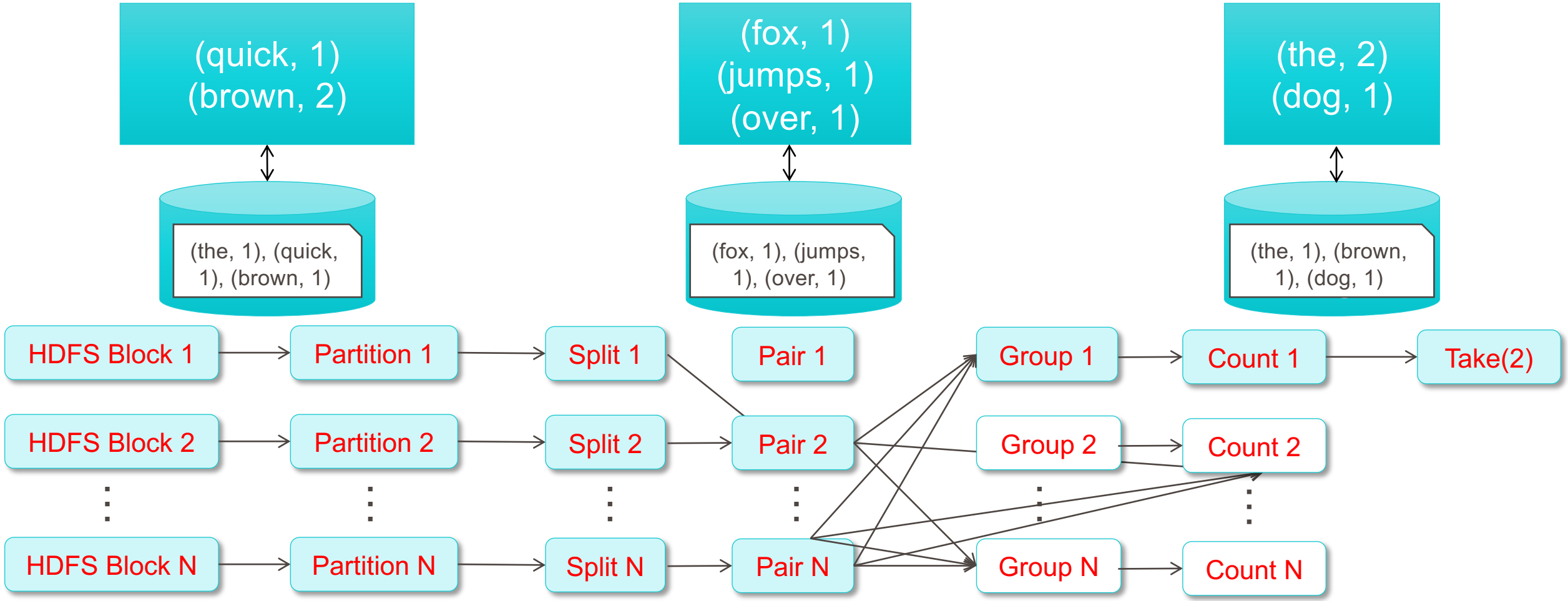
Execution



Execution



Execution



Spark on Cray XC

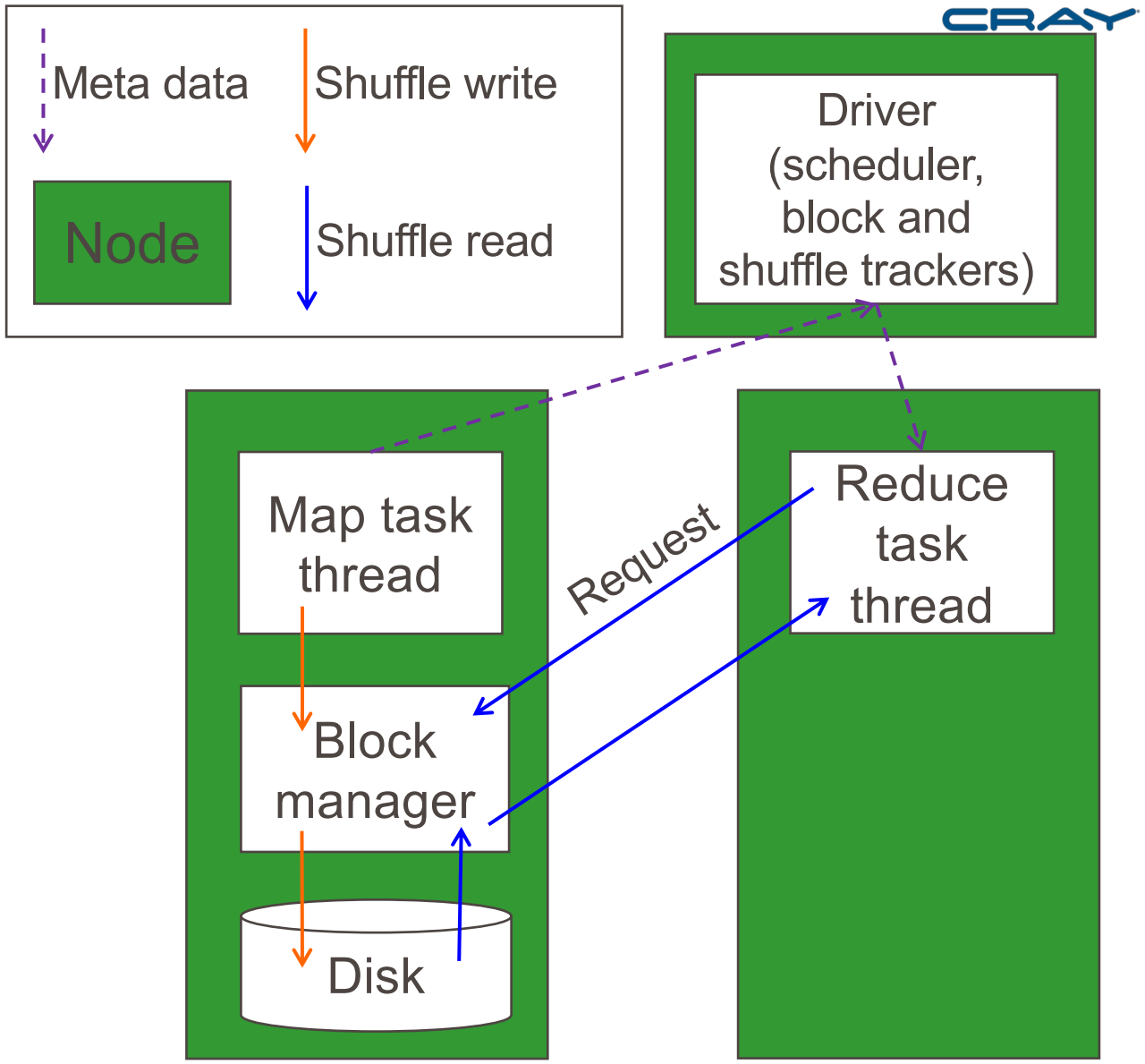
Spark on XC: Typical Setup Options

- Cluster Compatibility Mode (CCM) option
  - Set up and launch standalone Spark cluster in CCM mode; run interactively from Mom node or submit batch script
  - An example recipe can be found in:

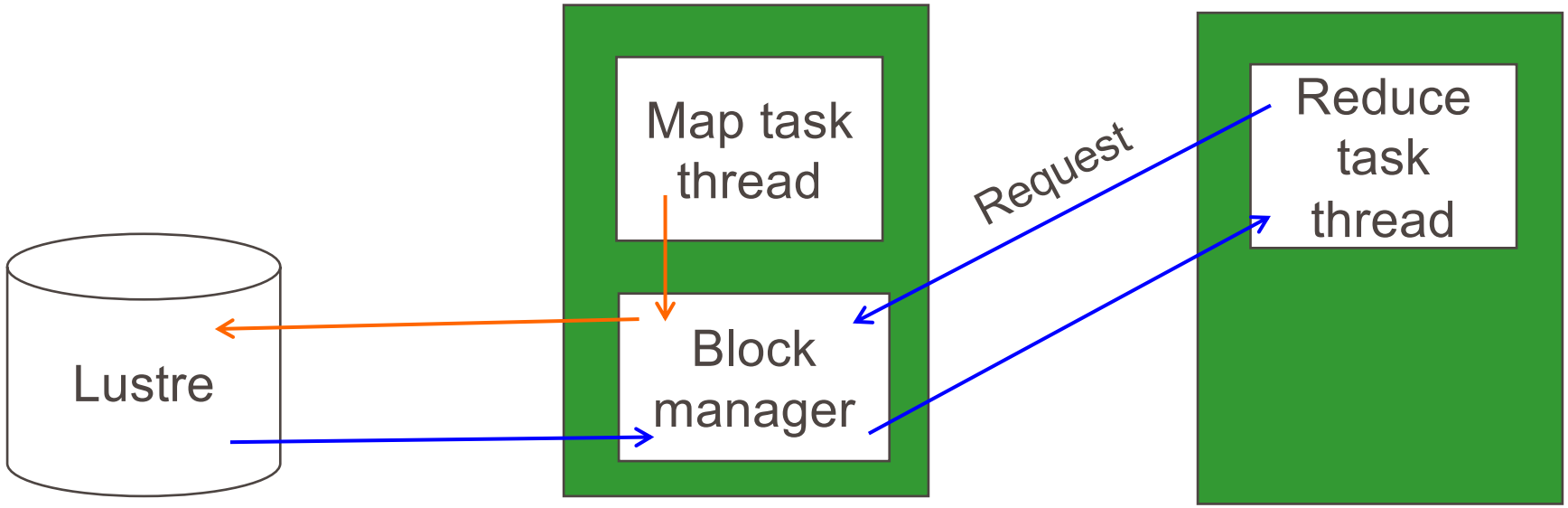
    “Experiences Running and Optimizing the Berkeley Data Analytics Stack on Cray Platforms”, Maschhoff and Ringenburg, CUG 2015
- Container option
  - Shifter container runtime (think “Docker for XC”) developed at NERSC
  - Acquire node allocation: run master image on one node, interactive image on another, worker images on rest
  - Cray’s Urika-XC analytics suite uses this approach
- Challenge: Lack of local storage for Spark shuffles and spills.

Reminder: Spark Shuffle – Standard Implementation

- Senders (“mappers”) send data to block managers; block managers write to local disks, tell driver how much destined for each reducer
- Barrier until all mappers complete shuffle writes
- Receivers (“reducers”) request data from block managers that have data for them; block managers read from local disk and send
- Key assumption: large, fast local block storage device(s) available on executor nodes

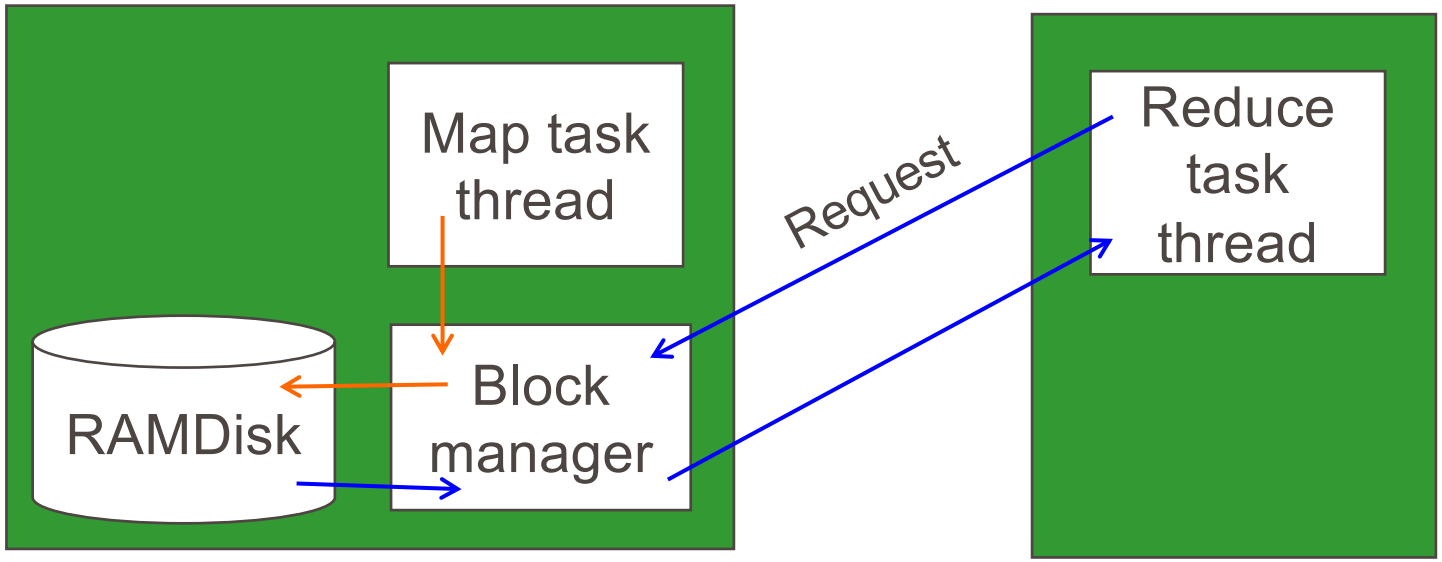


Shuffle on XC – Version 1



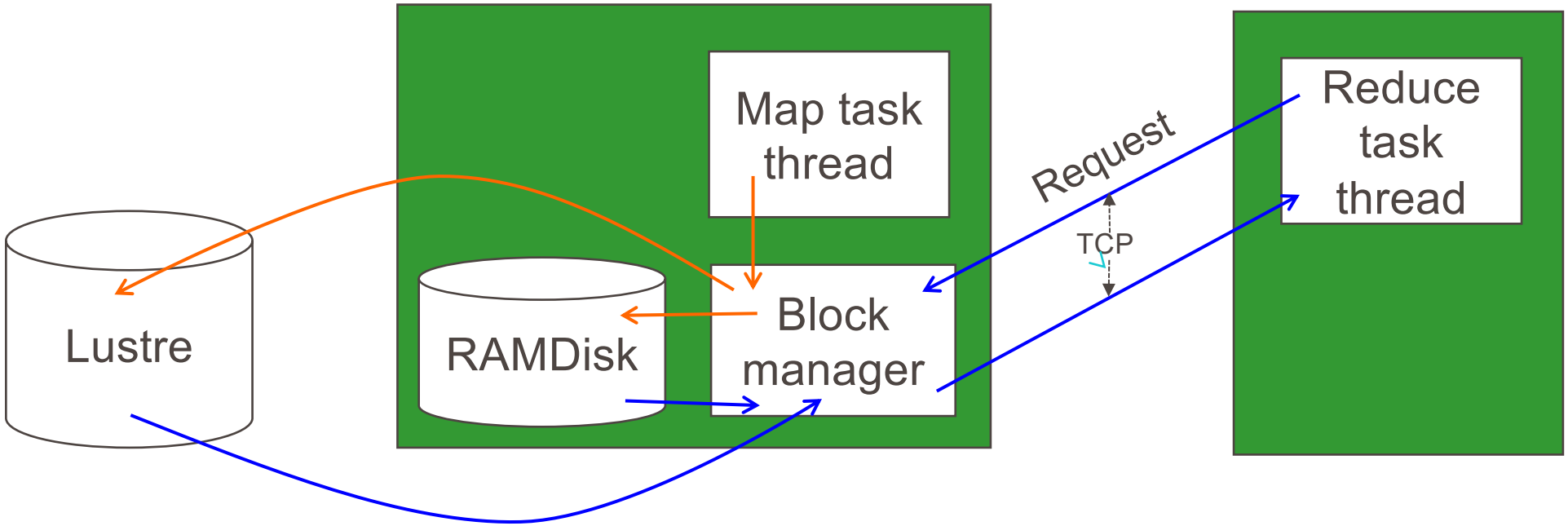
- Problems: No local disk on standard XC40
- First try: Write to Lustre instead
  - Biggest Issue: Poor file access pattern for lustre (lots of small files, constant opens/closes). Creates a major bottleneck on Lustre Metadata Server (MDS).
  - Issue 2: Unnecessary extra traffic through network

Shuffle on XC – Version 2



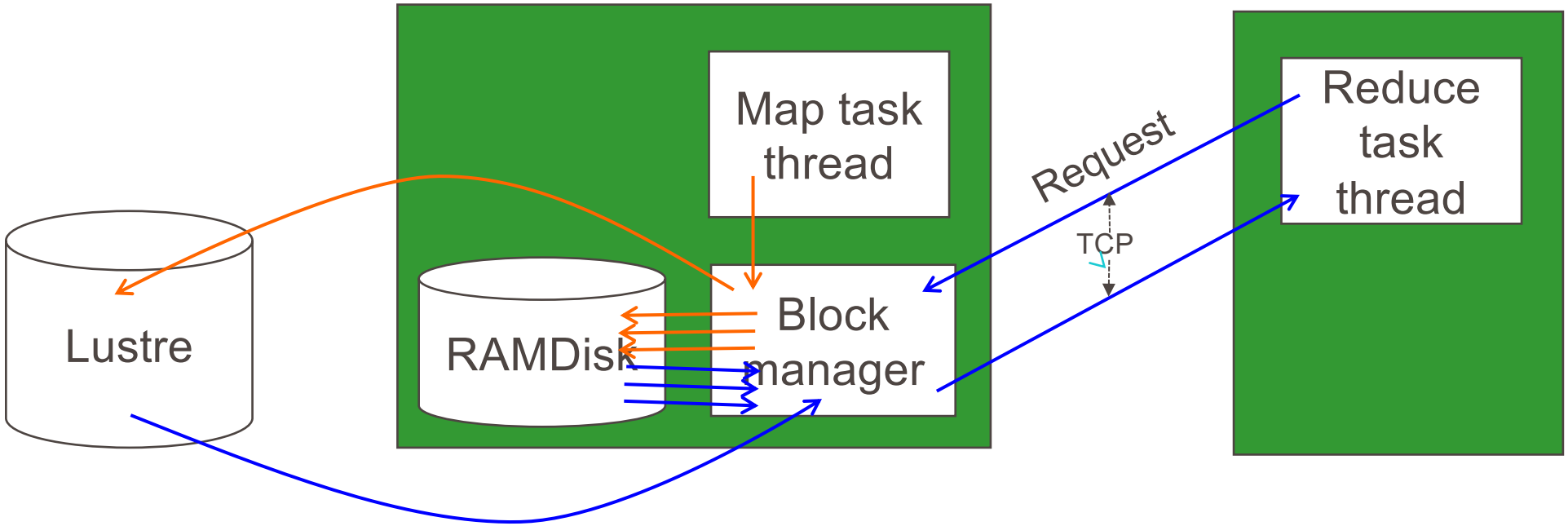
- Second try: Write to RAMDisk
  - Much faster, but …
  - Issues: Limited to lessor of: 50% of node DRAM or unused DRAM; Fills up quickly; takes away memory that could otherwise be allocated to Spark
  - Spark behaves unpredictably when it's local scratch space fills up (failures not always simple to diagnose)

Shuffle on XC – Version 3



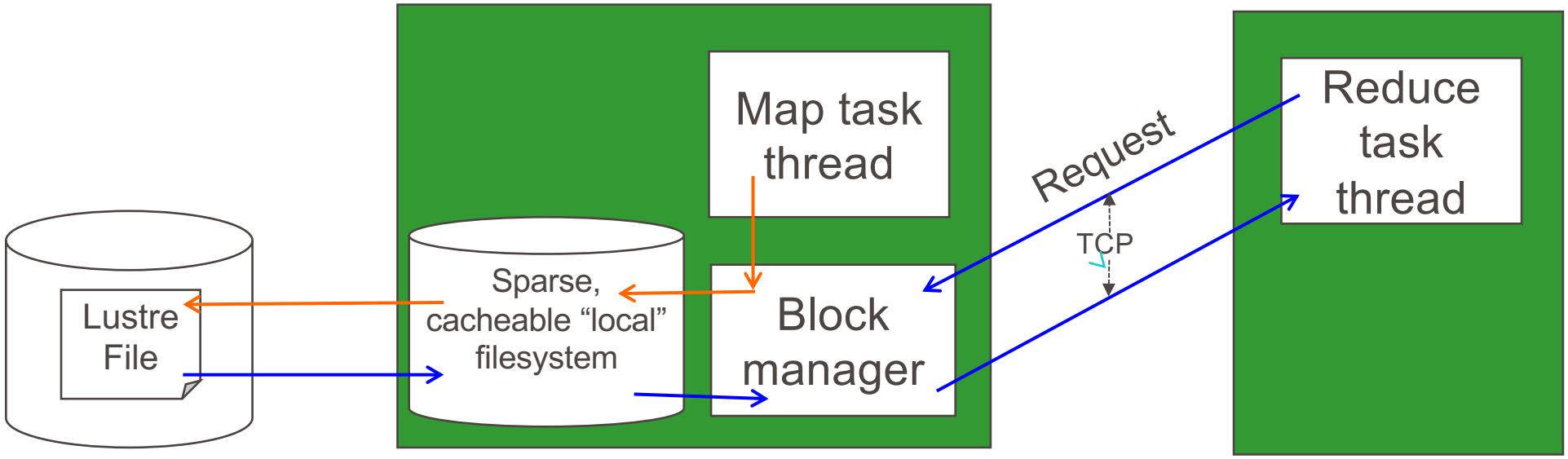
- Third try: Write to RAMDisk and Lustre
  - Set local directories to RAMdisk and lustre (can be list)
  - Initially fast and keeps working when RAMDisk full
  - Issues: Slow once RAMDisk fills; Round robin between directories (no bias towards faster RAM)

Shuffle on XC – Version 3



- Third try: Write to RAMDisk and Lustre
  - Set local directories to RAMdisk and lustre (can be list)
  - Initially fast and keeps working when RAMDisk full
  - Issues: Slow once RAMDisk fills; Round robin between directories (no bias towards faster RAM), *but can specify multiple RAM directories*

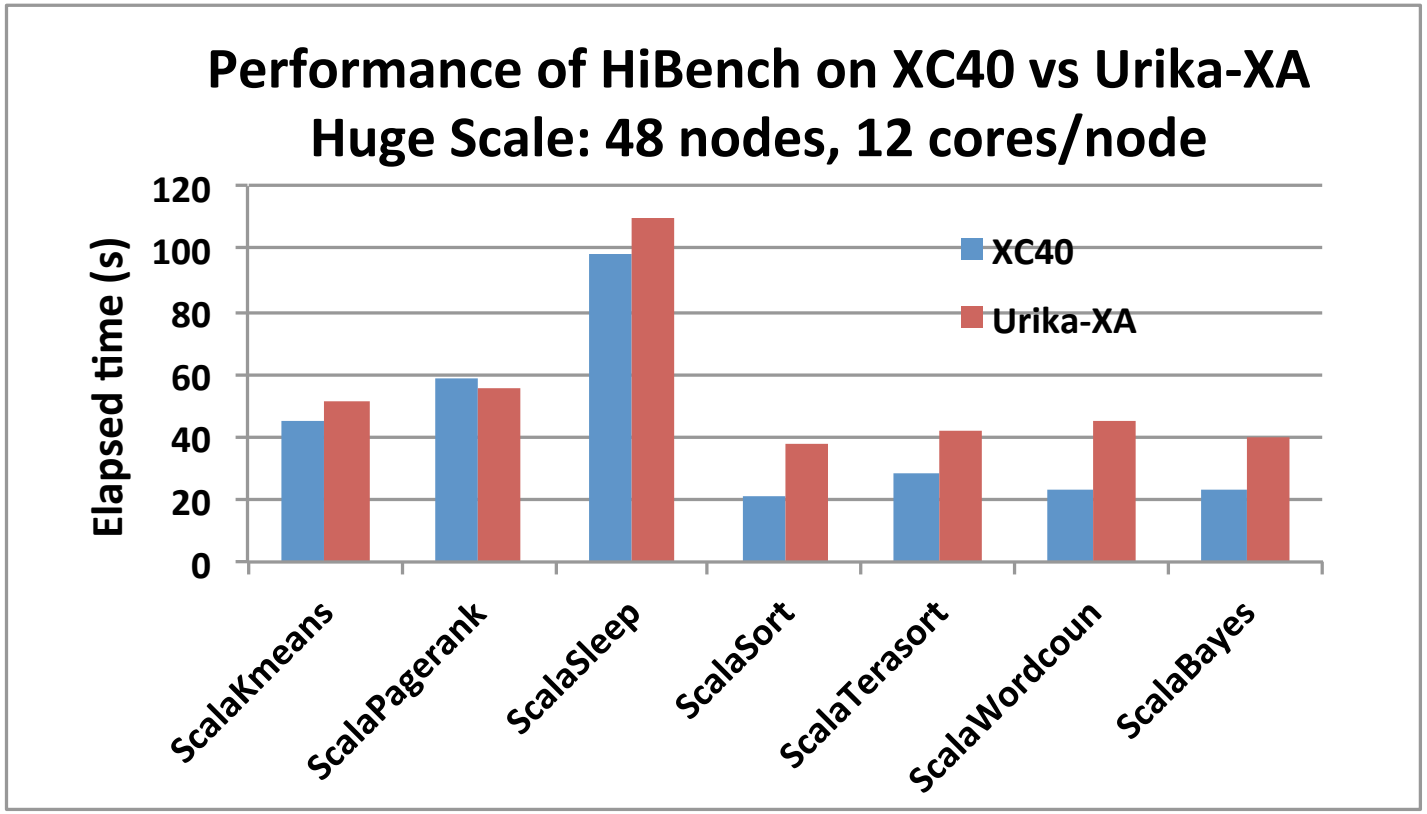
Shuffle on XC – with Shifter PerNodeCache



- Shifter implementation: Per-node loopback file system
  - NERSC's Shifter containerization (in Cray CLE6) provides optional loopback-mounted per-node temporary filesystem
  - Local to each node – fully cacheable
  - Backed by a single sparse file on Lustre – greatly reduced MDS load, plenty of capacity, doesn't waste space
  - Performance comparable to RAMDisk, without capacity constraints (Chaimov et al, CUG '16)
- Urika-XC ships as a Shifter image and uses this approach

Spark Performance on XC: HiBench

- Intel HiBench
  - Originally MapReduce, Spark added in version 4
- Compared performance with Urika XA system
  - XA: FDR Infiniband, XC40: Aries
  - Both: 32 core Haswell nodes
  - XA: 128 GB/node, XC40: 256 GB/node (problems fit in memory on both)
- Similar performace on Kmeans, PageRank, Sleep
- XC40 faster for Sort, TeraSort, Wordcount, Bayes



How can we make this even better?

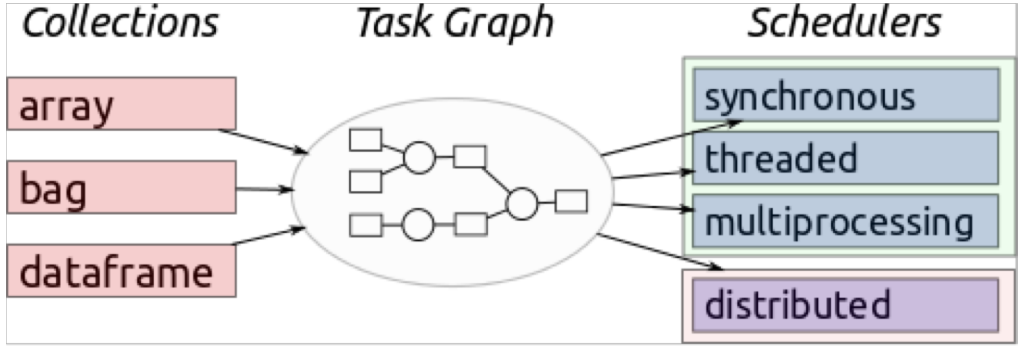
Don’t miss the Alchemist talk on Wednesday!

Salle Edward C

Wednesday, 1:00 PM

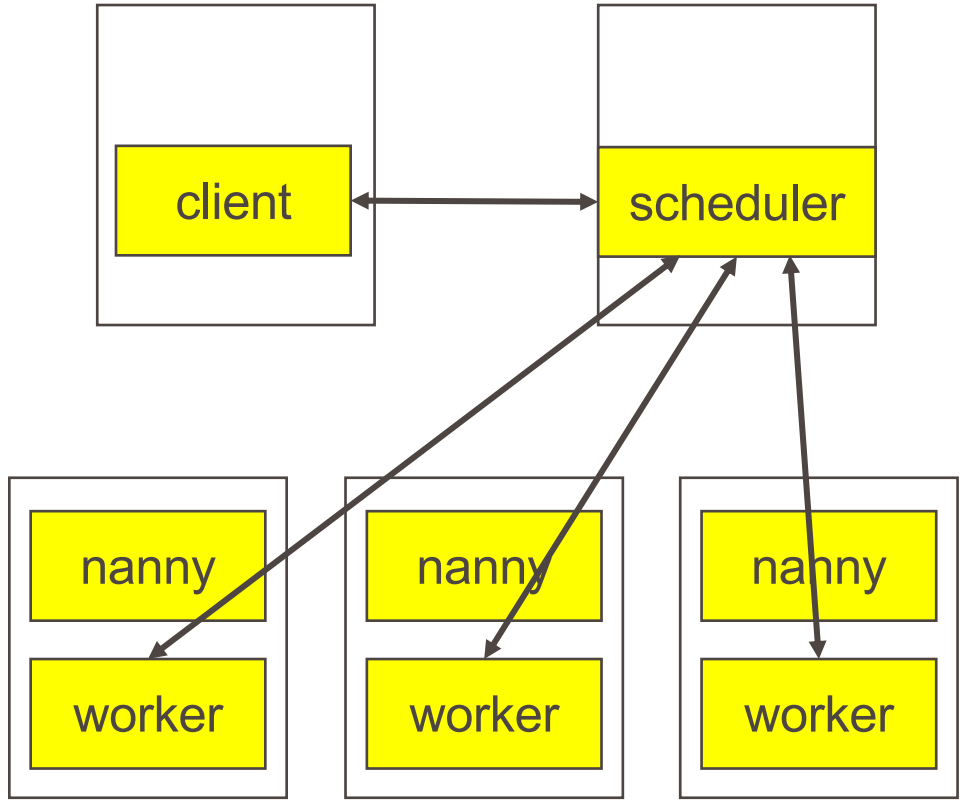
Dask and Dask Distributed

- Dask
  - Set of parallel collections and operations for Python
  - Integrated with most common packages, e.g., parallel version of numpy arrays
  - Supports multiple task schedulers
- Threaded scheduler
  - Backed by low-overhead thread pool
  - Subject to Python Global Interpreter Lock (GIL)
  - Best if application dominated by non-Python code
- Multi-process scheduler
  - Tasks shipped to separate local processes
  - Not subject to Python GIL – allows true on-node parallelism
  - Low overhead to launch/utilize pool, but overhead of moving data
  - Best for mostly Python code (allows parallelism even with GIL)



Dask and Dask Distributed

- Distributed scheduler
  - Dask scheduler for multi-node parallelism
  - Runs a scheduler on one node, workers across allocated nodes
  - Nanny processes for fault tolerance
  - Supports distributed versions of all Dask data structures
  - Allows asynchronous execution (futures)



Setting Up a dask.distributed Cluster on Cray

- Set up a dask distributed environment in anaconda python
  - `conda create --name mydask dask distributed`
- Get allocation
  - `salloc -N 4`
- Activate dask distributed
  - `source activate mydask`
- Start scheduler on one node, start workers on rest
  - Urika-XC can do this automatically:
    - `start_analytics --dask-env mydask`
  - Otherwise can use ssh or srun/aprun (details will vary based on your system)

THANK YOU

QUESTIONS?

mikeri@cray.com

cray.com
@cray_inc
linkedin.com/company/cray-inc-/

HPC and Analytics with R

Kristyn Maschhoff, Ph.D., Cray, Inc.

kristyn@cray.com

What is R?

- R project for Statistical Computing
  - https://www.r-project.org
  - Environment for statistical computing and graphics
  - “GNU S”
  - Freely available – but note most R packages have licenses
    - (GPL-2, GPL-3, MIT, Apache, etc.)
  - Latest Version R 3.6.0 (Planting of a Tree)
    - R version 3.6.0 (2019-04-26) -- ”Planting of a Tree"
- CRAN - The Comprehensive R Archive Network
  - https://cran.r-project.org
  - Network of ftp and web servers that store identical, up-to-date, versions of code and documentation for R
  - R manuals
    - https://cran.r-project.org/doc/manuals/

Interactivity and R

- R community was developed with the goal of interactive exploration of data
  - Basic R interactive console – provided with standard distribution
  - Many R users work use R using an IDE
- RStudio is by far the most popular IDE for R
  - R Markdown files and R Notebooks
  - Files have extension .Rmd
- R can also be run using Jupyter Notebooks
  - Install IRKernel

What we plan to cover in the tutorial

- Versions of R on Cray-XC
  - Cray PE version
  - R provided with Urika-XC
- Installing R packages
- Using Anaconda to manage R packages and multiple R versions (environments)
- pbdR Ecosystem provided on Urika-XC 1.2

Versions of R provided on XC

- Cray PE provides R prebuilt with Cray libsci using the GNU compiler
  - module load cray-R
  - Currently supported version is 3.4.2 (2017-09-28)

- R is also provided within Urika-XC image
  - Urika-XC 1.2
    - R version 3.5.1 (2018-07-02) -- "Feather Spray"
    - R prebuilt with openBLAS + GNU compiler
    - R built as a shared/dynamic library
    - pbdR Ecosytem pre-installed using Cray MPI (initial base set of packages)
    - Support for using Jupyter notebooks via IRKernel
  - Urika-XC 1.3
    - R version 3.6.0 (2019-04-26) -- ”Planting of a Tree"

Installing R Packages from CRAN

- Bring up R on login node and install needed packages
  - Need external access to download packages
  - In general, most tested, and most reliable compiler for R packages are the GNU compilers (gcc, gfortran)
  - Note, if using a site-installed version, any additional installed packages will be saved to a location in your home directory
    - ~/R/x86_64-pc-linux-gnu-library/3.5

> R packages we will be using for the tutorial

> install.packages("foreach")

> install.packages("doParallel")

> install.packages("rlecuyer")

> install.packages("randomForest")

> install.packages("SPARQL")

Installing R packages within Urika-XC

- Use start_analytics -d
  - Specify interactive node to run on login node
  - Better connectivity than from XC compute node
- For Urika-XC 1.2
  - R version 3.5.1 (2018-07-02) -- "Feather Spray"
  - User packages installed to
    - ~/R/x86_64-pc-linux-gnu-library/3.5

Running R using Urika-XC

kristyn@cicero:~> module load analytics

kristyn@cicero:~> start_analytics –d

Once inside the container, bring up R interactive shell to install packages

bash-4.2$ R

Inside R interactive shell

> install.packages("foreach")

```
Installing package into ‘/usr/lib64/R/library’
(as ‘lib’ is unspecified)
Warning in install.packages("foreach") :
  'lib = "/usr/lib64/R/library"' is not writable
Would you like to use a personal library instead?  (y/n) y
Would you like to create a personal library
~/R/x86_64-pc-linux-gnu-library/3.5
to install packages into?  (y/n) y
```

Managing R using Anaconda

- Anaconda R
  - Quite useful for managing R packages and multiple R environments on XC
  - List of R language packages available for install from conda is located at http://repo.continuum.io/pkgs/r/
  - R Essentials bundle includes about 100 of the most popular packages for R
  - Most recent version available: r-essentials 3.5.1

```
> conda create --name myR -c r r-essentials
> source activate myR
```

- Also can specify specific versions of R

```
> conda create --name myR_3.2.2 -c r r=3.2.2
```

- When using an older version of R it works better to create the conda environment first, activate this, then install the allowing packages, allowing conda to manage the package version dependencies

```
> source activate myR_3.2.2
> conda install -c r r-essentails r-xml
```

Enabling ssh between nodes using start_analytics

- On SLURM-based systems (with Shifter SPANK plugin installed)
  - salloc -N 4 --image=custom:analytics-1.01.0000.201712122205_0082-latest start_analytics –ssh
- On interactive node

```
># Determine nid allocations
>echo “$SLURM_NODELIST” or env | grep SLURM
>SLURM_NODELIST=nid0000[4-7]


># Start up R
>R
```

Simple Parallel Socket Cluster

- Basic functionality
  - Runs 'Rscript' on the specified host(s) to set up a worker process which listens on a socket for expressions to evaluate, and returns the results (as serialized objects).
- Commonly used R packages which then build upon the “parallel” package
  - “foreach” package
    - Provides looping construct
  - “doParallel” package
    - Provides mechanism needed to execute **foreach** loops in parallel
  - https://cran.r-project.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf

Example Code: using foreach and doParallel

```
> library(parallel)
> library(foreach)
> library(doParallel)
> machineVec <- c(rep("nid00004",4), rep("nid00005",4), rep("nid00006",4), rep("nid00007",4))

> cl <- makeCluster(machineVec)
> # To use the "foreach", we need to register the cluster with
> registerDoParallel(cl)
> getDoParWorkers()

> # sequential execution
> system.time(foreach(i=1:100000) %do% sum(tanh(1:i)))
> # parallel execution
> system.time(foreach(i=1:10000) %dopar% sum(tanh(1:i)))

> mcoptions <- list(preschedule=FALSE, set.seed=FALSE, cores=4)
> system.time(foreach(i=1:100000,.options.multicore=mcoptions) %dopar% sum(tanh(1:i)))
```

R profiling

- Standard approach – use Rprof
  - Profile R code is to use the Rprof function to profile and the summaryRprof function to summarize the result

```
>help(Rprof)


>Rprof(tmp <- tempfile())
>example(glm)
>Rprof()
>summaryRprof(tmp)
```

Programming with Big Data in R (pbdR)

- Set of highly scalable R packages for distributed computing in data science
  - http://r-pbd.org/
- George Ostrouchov, Wei-Chen Chen, Drew Schmidt, Pragneshkumar Patel
- Winner of the Oak Ridge National Laboratory 2016 Significant Event Award for "Harnessing HPC Capability at OLCF with the R Language for Deep Data Science

pbdR Ecosystem – Urika-XC & Urika-CS

- The Urika-XC 1.2UP00 image ships with an optimized version of R, built with OpenBLAS
  - R version 3.5.1 (2018-07-02) -- ”Feather Spray"
  - OpenBLAS 0.2.19 (from latest Debian libopenblas-dev package)
- Added new run_pbdR command
  - Allows the user to execute a distributed R application inside the Urika image using the pbdMPI package
  - Sets up the run-time environment to utilize optimized R library provided in the Urika-XC image along with the pbdR ecosystem and Cray MPI communication libraries.
- pbdR on Urika-CS 1.1
  - Singularity
  - Utilizes system optimized openMPI libraries

pbdR Ecosystem – initial set of packages

pbdR component	Description
pbdMPI	simple R interface for MPI programming
pbdBASE	Base utilities for distributed matrices
pbdSLAP	The Scalable Linear Algebra Package (distribution of ScaLAPACK)
pbdDMAT	A distributed matrix of classes and methods. This package includes numerous methods for manipulating and reshaping distributed matrices, as well as linear algebra and statistics routines
pbdML	Machine learning algorithms
pmclust	Tools for parallel model-based clustering. These include k-means and Gaussian mixture modeling
pbdIO	Parallel I/O packages (SPMD)

Using run_pbdR

Procedure

Load the analytics module

```
$ module load analytics
```

Obtain job allocation

```
$ salloc -N 4
```

Run distributed R application

```
$ run_pbdR -n 4 --ppn 16 “Rscript ./mpi_hello_world.r”
```

```
# mpi_hello_world.r

# load the package
suppressMessages(library(pbdMPI, quietly = TRUE))

# initialize the MPI communicators
 init()

# Hello world
message <- paste("Hello from rank", comm.rank(),
      "of", comm.size())
       comm.print(message, all.rank=TRUE, quiet=TRUE)

# shut down the communicators and exit
 finalize()
```

pbdR – Getting started

- Laptop
  - pbdR Docker builds hosted on dockerhub
    - https://hub.docker.com/u/rbigdata/
  - pbdR Dockerfiles on github
    - https://github.com/RBigData/docker
  - Tutorial/Workshop
    - https://github.com/RBigData/docker/tree/master/special/workshop

pbdMPi – beyond “Hello World”

- HPSC Cookbook – Wei-Chen Chen
- https://snoweye.github.io/hpsc/cookbook.html

- In addition there are several tutorials available with source code available for download
- Tutorials 1 and 2 both use the Iris dataset already available with base R install

THANK YOU

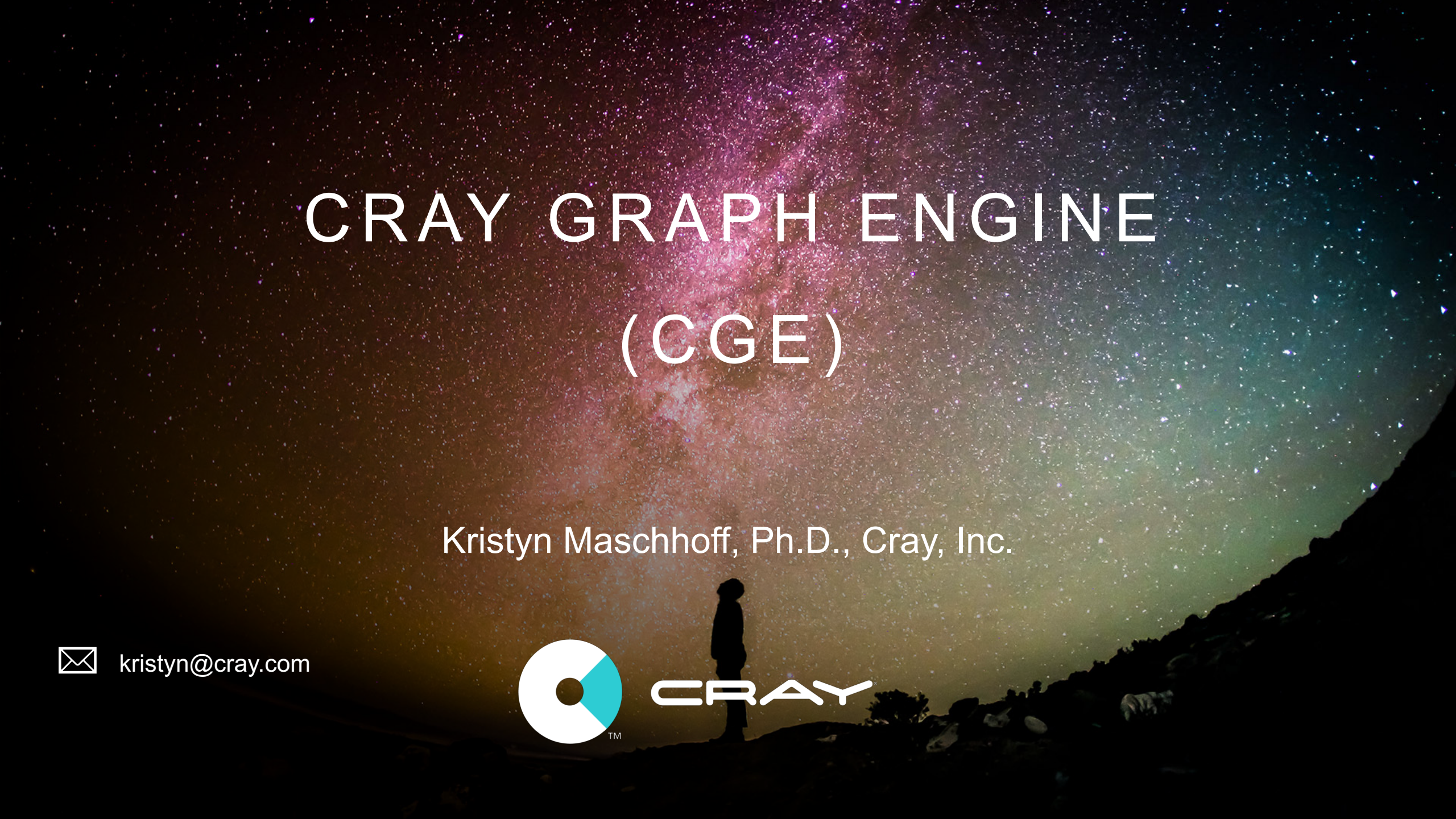
QUESTIONS?

 kristyn@cray.com

CRAY GRAPH ENGINE (CGE)

Kristyn Maschhoff, Ph.D., Cray, Inc.

kristyn@cray.com



What we plan to cover in the tutorial

- Background on CGE
  - Pattern matching, whole-graph analysis
  - Trillion Triples benchmark (CUG 2018)
- Hands-on exercises
  - Build and start up a database (cge-launcher)
  - Run queries
    - Using the cge-cli command line
    - Using the CGE Web UI
  - Integration with R and Python
    - Connecting to the CGE SPARQL endpoint
      - Using R SPARQL package
      - Using Python SPARQLwrapper package

Cray Graph Engine (CGE)

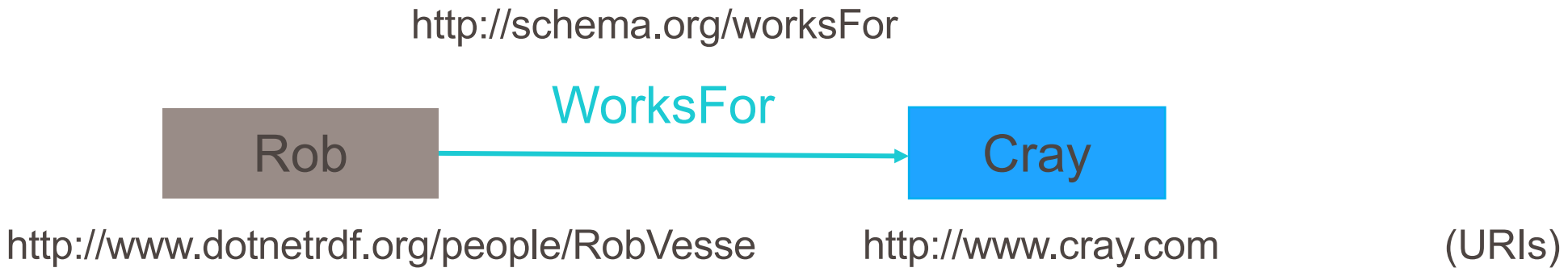
- Scalable parallel graph analytics framework
  - Semantic in-memory graph database
    - Basic graph pattern search
    - Graph-theoretic algorithms (whole graph algorithms)
  - W3C Standards Based
    - Uses RDF Data representation
    - Uses SPARQL as query language
  - Built for “vertical scaling” based on parallel and distributed computing principles — competitors are all horizontally scaled
- Brings interactivity to graph-based discovery
  - Scaling and performance enables interactive analysis of very large datasets

Cray Graph Engine: Updates and Features

- Multi-Architecture Support
  - CGE is available on the Urika-GX and the XC platforms.
  - Strong scaling becomes a key differentiator
    - Bigger datasets => more nodes => better performance
- Integration with Spark
  - Interface to data sources - support for end-end analytic workflow realization
- Integration with Python/Jupyter Notebooks
  - Connect to SPARQL endpoint using sparqlwrapper or sparql-client packages
  - CGE Python API – utilizes the CGE Java API
    - Start up server, run queries, updates, checkpoint, shut down
- Integration with R
  - SPARQL package – connect to SPARQL endpoint, run queries, updates

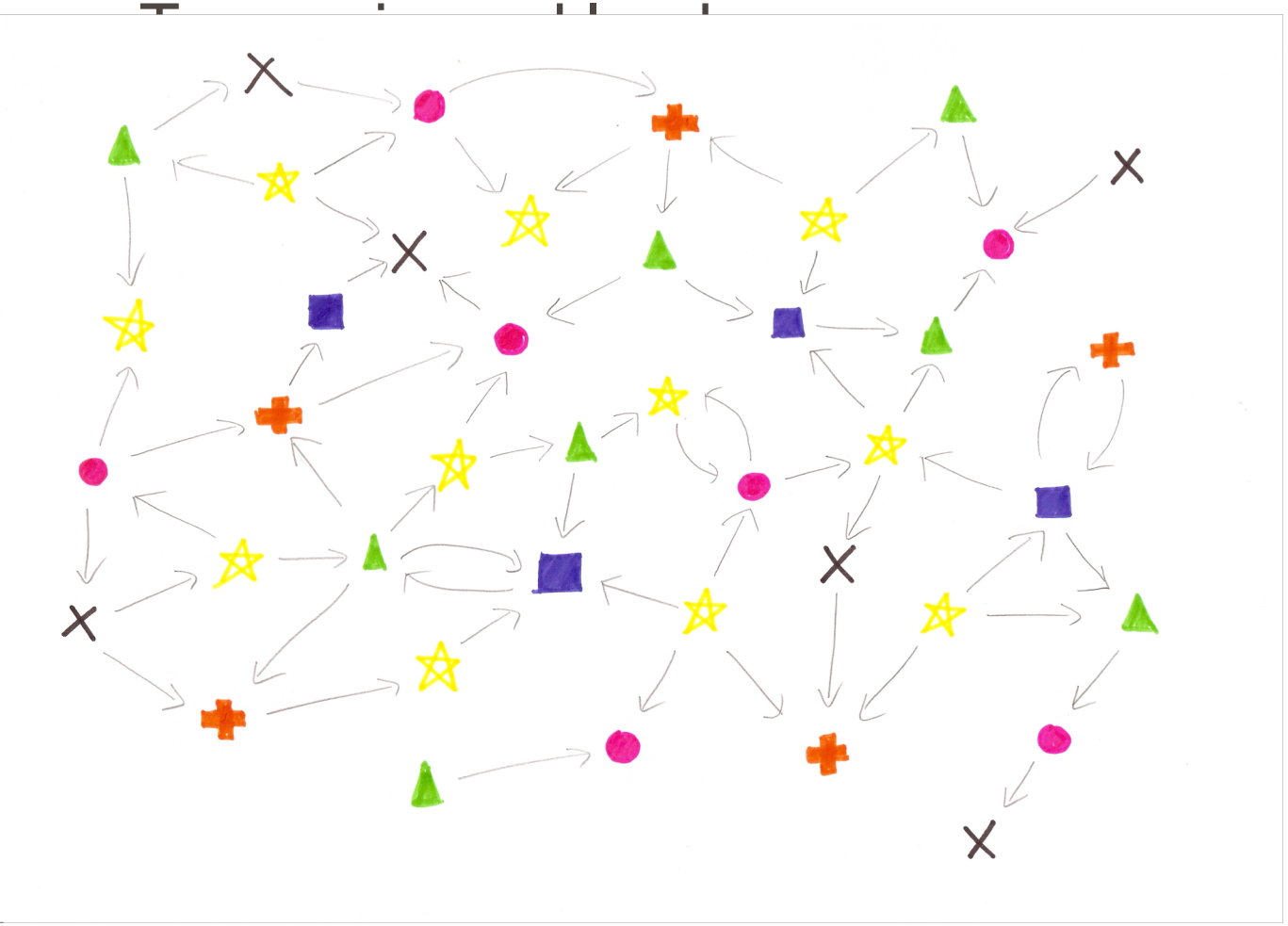
What is RDF?

- Resource Description Framework (RDF)
- A standardized abstract data model centered around the notion of Triples
- A Triple expresses a directed relationship between two entities e.g.



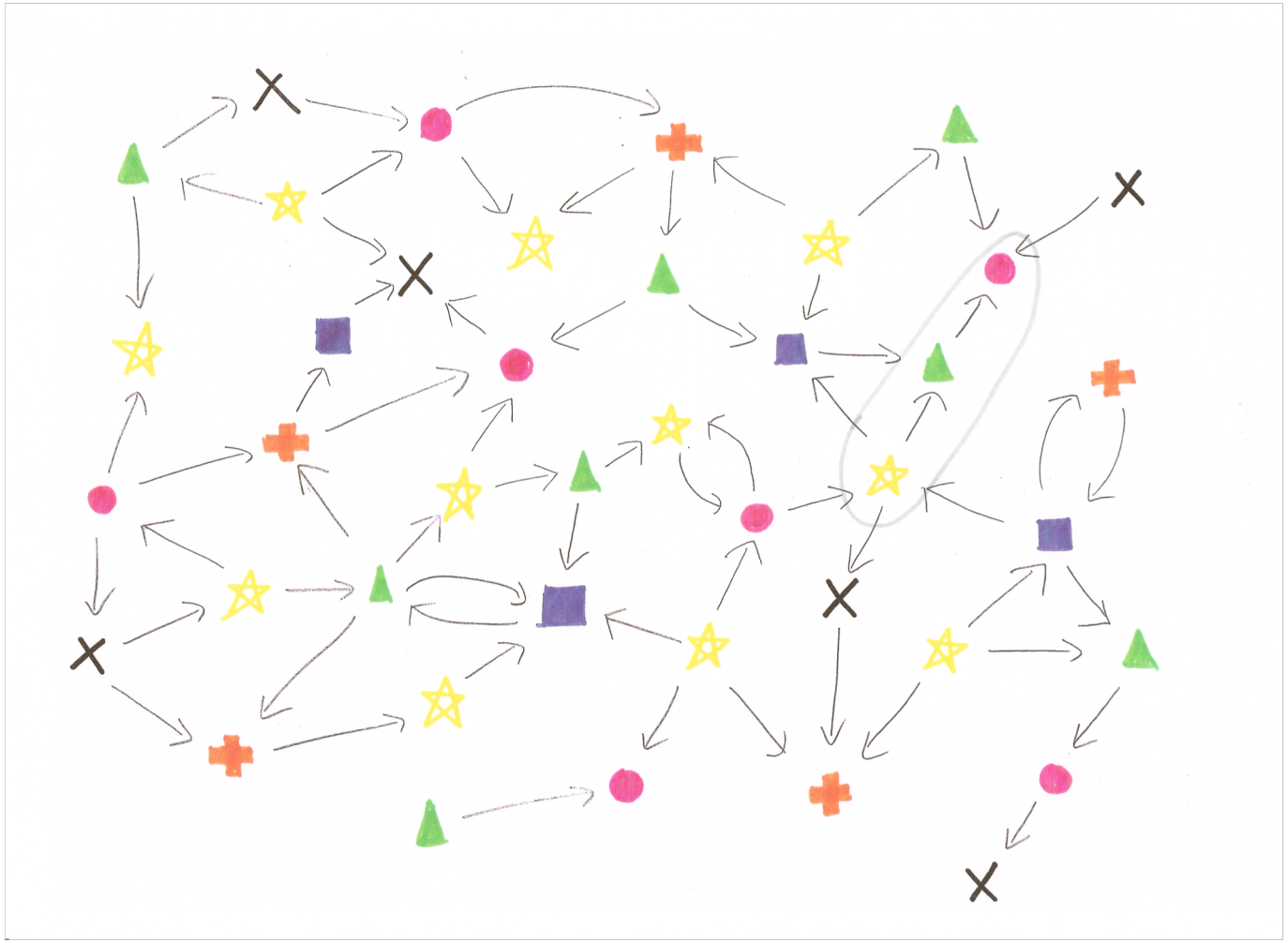
- Components of a Triple are commonly known as Subject, Predicate and Object
  - Subject – The thing I am making a statement about
  - Predicate – The relationship being stated
  - Object – The thing which is related

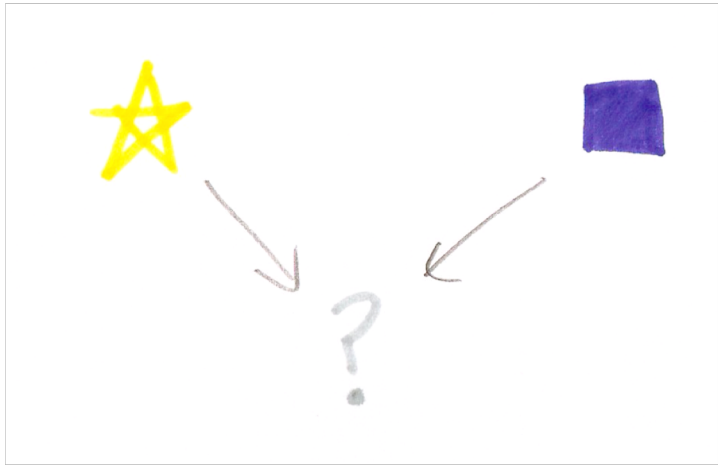
Graph analysis workloads



A Graph-pattern matching workload

Given a pattern of interest find all instances thereof…





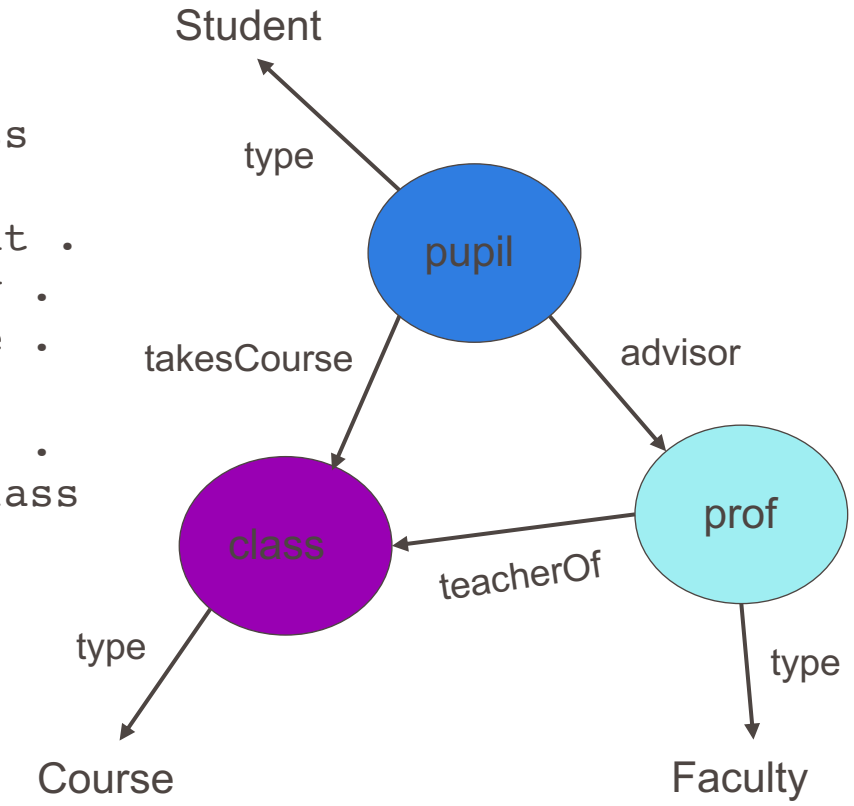
What SPARQL Can Do

LUBM Query 9

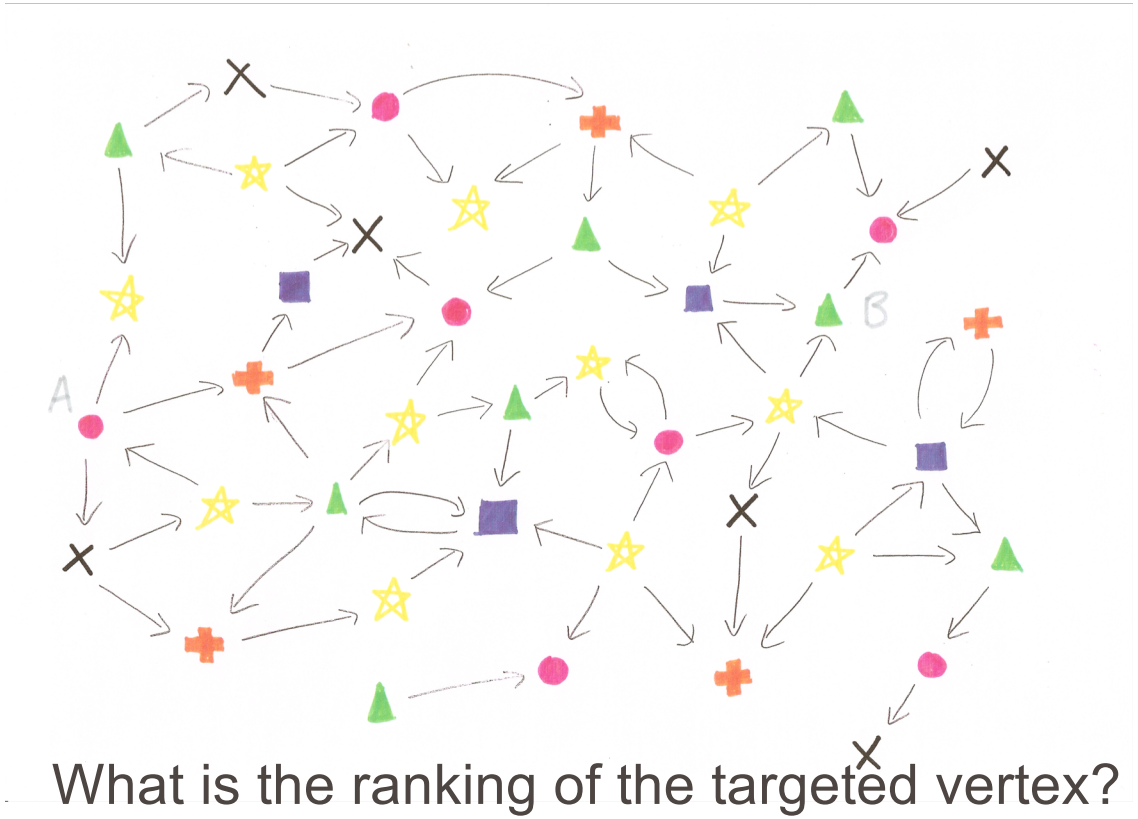
Basic Graph Pattern (BGP)

- Primary unit of search for the SPARQL query language
- A SPARQL query always starts with a BGP, followed by additional filters, joins, or other operators
- Subgraph isomorphism problem
- Consists of SCAN, JOIN, MERGE phases

```
SELECT ?pupil, ?prof, ?class
WHERE
{ ?pupil rdf:type ub:Student .
  ?prof rdf:type ub:Faculty .
  ?class rdf:type ub:Course .
  ?pupil ub:advisor ?prof .
  ?prof ub:teacherOf ?class .
  ?pupil ub:takesCourse ?class
}
```

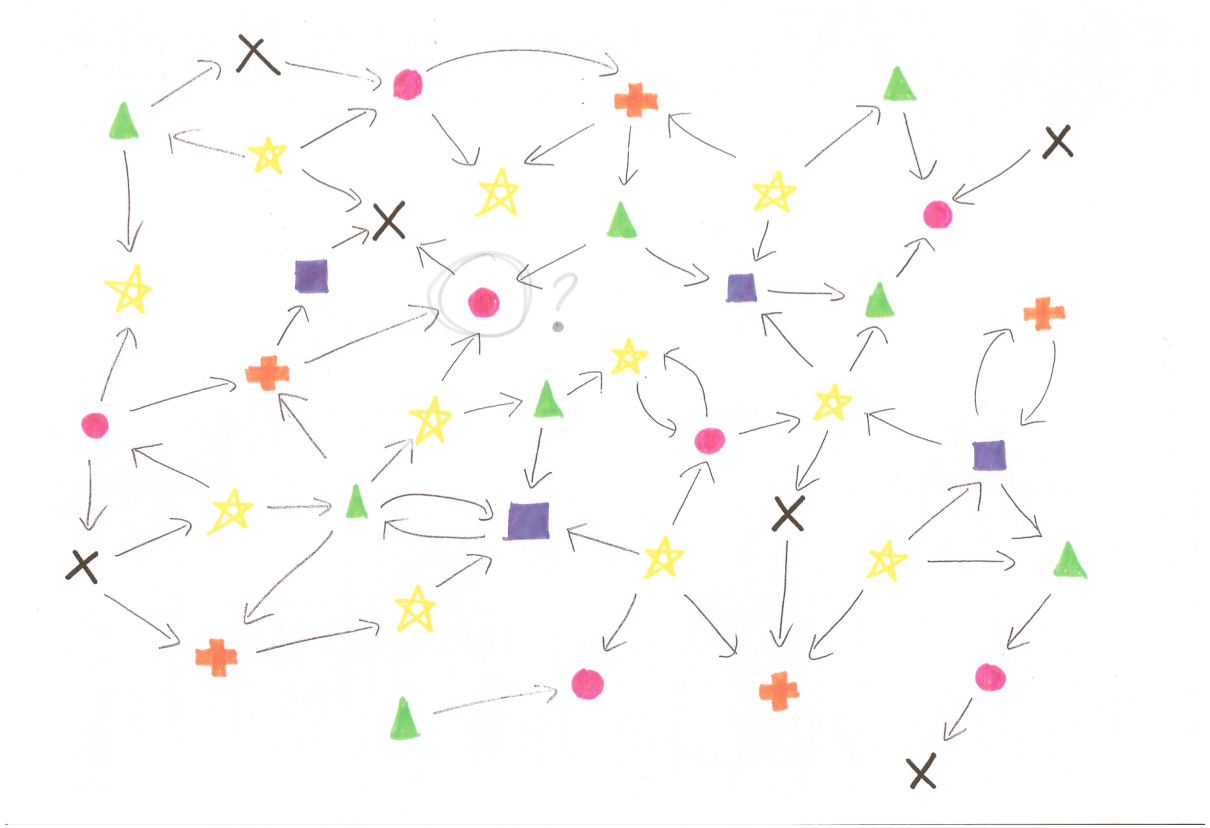


A Graph-theoretic Workload



What is the ranking of the targeted vertex?

What's the shortest route from A to B?



SPARQL versions

- Current CGE release supports SPARQL 1.1 features
  - SPARQL 1.1 Released 2011
    - Basic pattern matching, filters, unions
  - SPARQL 1.1 Update (SPARUL) functionality added
    - INSERT, DELETE, LOAD, DROP

- Not supported in Current CGE release
  - Some features of Property paths
  - SERVICE keyword

- Special CGE features not in SPARQL 1.1
  - Built-in Graph Functions (BGFs)
  - Interval Analytics Functions
  - Haversine Functions
  - Square Root Functions

Built-in Graph Functions (BGFs)

- RDF and SPARQL are graph-oriented, but SPARQL is limited in its ability to express graph processing
- We augmented SPARQL with a capability of calling library graph algorithms
- You can go from SPARQL to a graph algorithm and back to SPARQL for further refinement
- The whole is greater than the sum of its parts

Status: what we now have on Urika-XC

- BadRank
- PageRank
- S-T connectivity
- Betweenness centrality
- Community detection via Label Propagation
- S-T Set connectivity
- Triangle Counting
- Triangle Finding
- Vertex Triangle Counting

Directions and input parameters are in the manual…

Interval Analytics Functions For Temporal Analysis

Interval functions can be used to gather fine-grained detail about intervals. For example, you can use them to:

- Determine whether or not two or more time intervals intersect.
- Determine if a time period that ends at the same time is contiguous with one that starts at the same time.
- Determine the continuity of a given time period.

Geospatial Functions

- Square Root
  - sqrt(argument)
  - Fairly self-explanatory, not in SPARQL 1.1

- Urika Haversine Functions
  - haversinemeters(latStart, longStart, latEnd, longEnd)
  - haversinemiles(latStart, longStart, latEnd, longEnd)
  - The haversinemeters() function returns the distance between two points in meters, whereas the haversinemiles() function returns the distance between two points in miles.

State of the Art for Trillion Triples

Cambridge Semantics, Oct 2016

- Second to achieve a trillion triples
- Dedicated in-memory semantic database, but it is horizontally scaled and more cloud-oriented
- LUBM 4400K, Turtle format, also created 1T triples after inferencing. Data generated and loaded from local SSD on-node
- Software / hardware:
  - Anzo Graph Query Engine (AGQE)
  - 200 nodes Google Compute Platform, each 32 vCPUs, 208 GB

Oracle, September 2014

- First to achieve trillion triples
- Uses a semantic layer over their standard Relational database product
- Disk-based with an SSD cache
- LUBM 4400K, N-Triples format, created 1T triples after inferencing
- Software / hardware:
  - Oracle Database 12c
  - Exadata X4-2 High capacity full rack, 8 DB nodes, 14 storage nodes

Comparison of Trillion Triples benchmarks

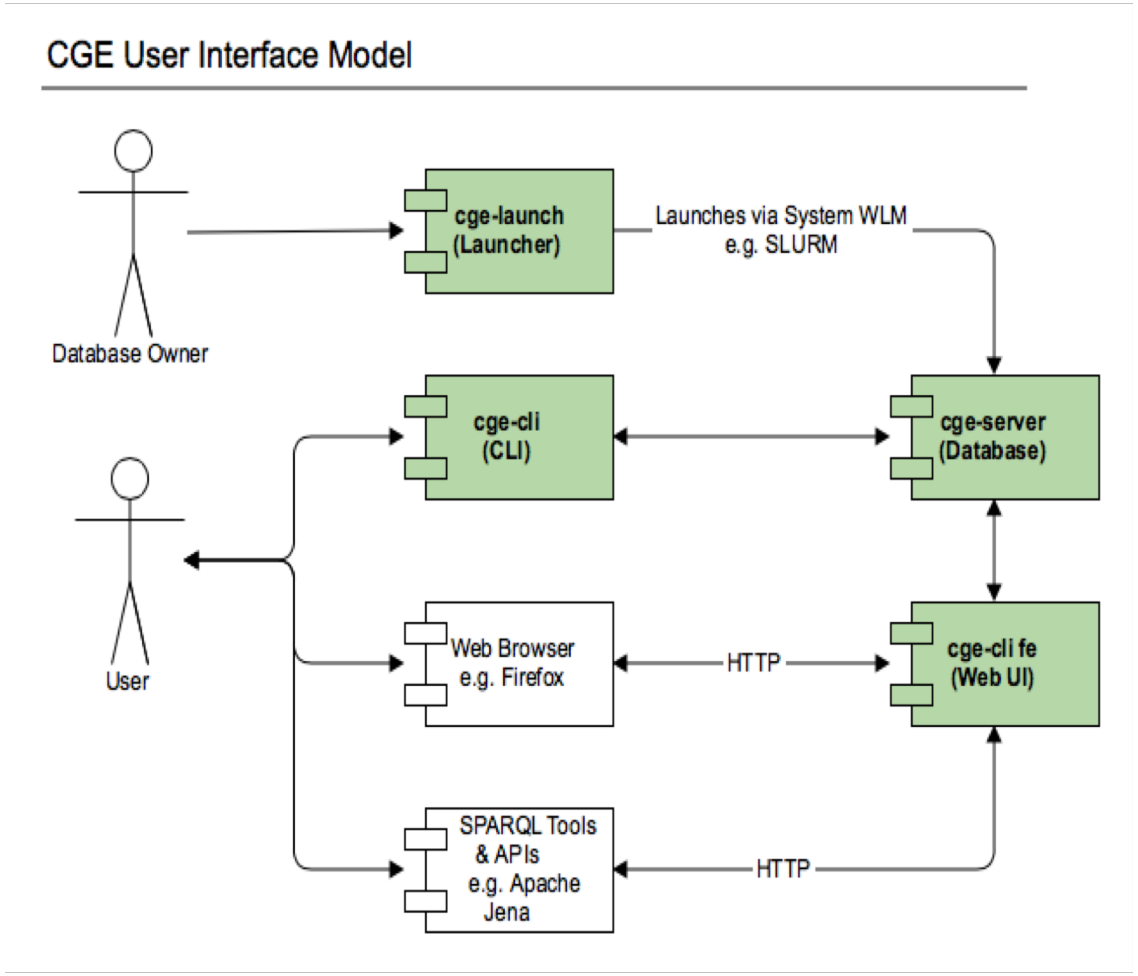
Software	Load time	Inference time	Query time
Oracle Database 12c	115.2 hours	86.5 hours	22.5 hours
Cambridge Semantics AGQE	1764 seconds	4574 seconds	840 seconds
Cray CGE (2018)	4124 seconds	535 seconds	98 seconds

Integrated Life Sciences Dataset

- CGE can accommodate all relevant databases in one environment
- Enables seamless cross-database queries
- Able to load relevant and realistic life sciences datasets in minutes rather than hours or days
  - Build time: 10 minutes (256 nodes)
  - Load time: 105 sec (256 nodes)

Dataset	Size (GB)	# of Triples
Biomodels (r31)	0.2	1057465
Biosamples (v20160912)	61	352661330
ChEMBL (23.0)	75	496019419
Ensembl (Jan 2018)	307	1926736803
Expressionatlas (18-05-2017)	138	685755602
OLS (March 2018)	9.6	73898957
Reactome (r61)	3.6	22354615
UniProt (Feb 2018)	6274	42157935260
OrthoDB (9v2)	137	1128153578
Integrated Total	7 TB	47 Billion

CGE User Interface Model



- Database owner launches the database server
- Users interact via their preferred interface
  - Commands Line
  - Web Browser
  - SPARQL Tools & APIs
  - CLI may be used for scripted workflows

Building and launching

- cge-launch is used to build databases:

```
cge-launch –N 8 –I 16 –o /mnt/lustre/myresults –d
/mnt/lustre/mydata –l logfile
```

- `cge-launch` is a script that takes care of resource allocation for the user!

- After a successful build, the database directory will contain:

  dataset.nt
  rules.txt
  dbQuads
  string_table_chars
  string_table_chars.index
  graph.info

The database port

- A TCP port used for communication with this server instance:

```
cge-launch –N 8 –I 16 –p 3750 …
```

- The default is 3750
- Changing this port allows multiple versions

The database directory

- The database directory, typically:

```
/mnt/lustre/user/datasets/lubm0
```

- Is the start of a directory tree containing all checkpoints, and potentially authorized_keys
- It can be moved, archived and returned (!)
- Multiple users can access it, with permissions

The Command Line Interface (CLI)

- The CLI is used for most interactions with the server, and has many options…

```
cge-cli –db-port 3750 query myquery.rq
```

- `cge-cli help` (or `cge-cli help checkpoint`) will give verbose information on options

- Designed for scripted control, querying and updates with database server

- Communications are secure SSH

CLI — most common options

`query` – submits SPARQL queries

`update` – submits SPARUL updates

`sparql` – submits both queries and updates

`checkpoint` – creates a database checkpoint

`echo` – check status of server

Customization using NVPs and cge.properties file

- Retrieve the Default NVP Configurations

```
$ cge-cli nvp-info
```

- For some systems, need to modify internal memory allocator defaults settings due to accommodate smaller 64GB nodes
  - CGE uses a internal memory allocator to avoid issues with observed memory fragmentation on XC systems
  - cge.server.BuddyMemPercent 20 (current default 35)
  - cge. server.PersistBuddyMemPercent 20 (current default 25)

- More information
  - https://pubs.cray.com/content/S-3014/3.2.UP01/cray-graph-engine-user-guide

Hands on Exercises: Running CGE on Cori (1)

- See README for instructions and exercises
  - /global/cscratch1/sd/kristyn/CUG2019/CGE/README
- To use CGE Web UI, need to set up ssh tunneling
  - Current cge-launch script for XC depends on xtprocadmin
    - Only available on internal Cori MOM nodes cmom02 and cmom05, need to ssh to these nodes from login node
  - Create tunnel from my laptop to internal cmom02 node on Cori
    - Use a random port number (8022) to connect to ssh port 22
      - ssh –L localhost:8022:cmom02:22 cori.nersc.gov

        Enter password+OSA
    - Then ssh directly into cmom02 from laptop, choosing another random port number (15000) for CGE fe
      - ssh –p 8022 –L localhost:15000:localhost:15000 localhost

        Enter password (no OSA)
- The command line interface does not need tunneling
  - cge-cli commands may be scripted for batch submission

Hands on Exercises: Running CGE on Cori (2)

- Set up database directory on Lustre
  - Make sure Lustre striping is set
    - lfs setstripe –c 16 –stripe-size 16m .
  - Needed files: dataset.nt, graph.info, rules.txt
- Set up query_results directory on Lustre
  - Make sure Lustre stripiing is set
- Be sure to set passwordless ssh
  - ssh-keygen
  - cat id_dsa.pub >> authorized_keys

THANK YOU

QUESTIONS?

 kristyn@cray.com

SAFE HARBOR STATEMENT

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts.

These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.