# Reinforcement Learning At Exascale

PASC 21 Conference minisymposium:
Exploiting and Supporting Exascale Infrastructure for Deep Learning Applications

Geneva, July 5-9

Jamaludin Mohd Yusof, Los Alamos National Laboratory
Vinay Ramakrishnaiah, Los Alamos National Laboratory

# Talk Overview

- Introduction to Exalearn ECP Project: Co-design Center for Exascale Machine Learning Technologies
  - Application pillars
  - Control pillar overview
  - RL Overview
  - Control examples
- EXARL Software Overview
  - Challenges for scientific reinforcement learning
  - Current software design
  - Path forward

- Control Team:
  - Christine Sweeney, LANL, Lead, Shinjae Yoo, BNL, Co-Lead

    PNNL: Malachi Schram, Josh Suetterlein, Asher Mancinelli, Draguna Vrabie

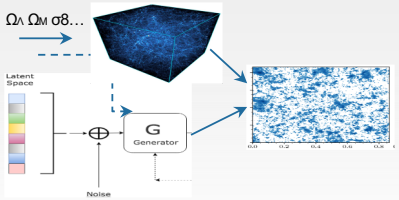
    BNL: Ai Kagawa

    LBL: Brian Friesen, Neil Mehta, Jack Deslippe

# ExaLearn Application Pillars

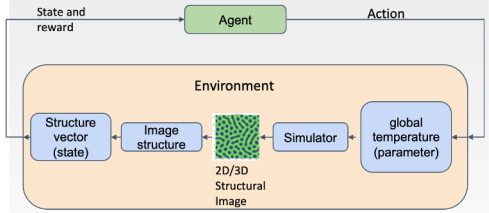
## Surrogates

- ML-created models
- Faster and/or higher fidelity models
- Generative networks
- Use ML to replace complicated physics, learned low fidelity to high fidelity mapping,
- Cosmology
- Fast, accurate Surrogates to replace large-scale epidemiological simulations



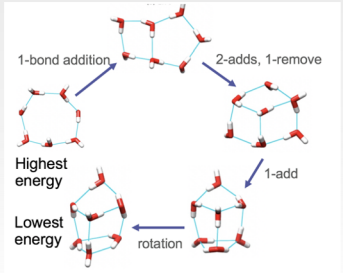

## Control

- ML-controlled processes and ML-controlled design
- Efficient exploration of complex action space
- Reinforcement Learning
- Accelerator parameter control, temperature control for guiding block copolymer self-annealing, water cluster molecule design
- Using RL to design optimal non-pharmaceutical interventions



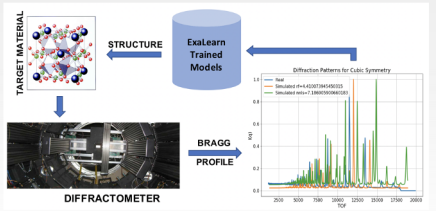

## Design

- ML-generated physical structures
- Optimized proposal for desired behavior of structure within complex design space
- Methodologies include graph neural networks, variational autoencoders, point-cloud based CNNs, deep reinforcement learning
- Molecular Generator and Design for COVID-19 Therapeutics




## Inverse

- ML projection from observation to original form
- Back-out complex input structure from observed data
- Regression models
- Predicting crystal structure from light source imaging
- Material structure from neutron scattering
- ML framework for protein structure prediction from SNS experimental data




EXASCALE COMPUTING PROJECT

# Brief Overview of ExaLearn Control Pillar

- Goals
  - Provide scalable control-related machine learning software for ECP applications
  - Implement use case applications for demonstration and testing
  - Run on exascale DOE Leadership class computing facilities
- Methods
  - Using primarily reinforcement learning for now, but could expand to other methods
  - Science use cases:  Accelerator control, temperature control for block copolymer self-annealing in light source experiments, water cluster molecular design
  - EXARL software framework for exascale reinforcement learning for science and benchmarking
- Collaboration
  - Working toward adoption of molecular design application
  - Leverage related ECP application software (eg CANDLE hyperparameter optimization)
  - ECP Proxy App project collaboration on reinforcement learning proxy app

# Approaches to Solving Control and Design Problems

- Controllers
  - Good if control algorithm is simple/straightforward and experts are available, otherwise could be intractable
- Optimization
  - Can utilize HPC and try many solutions, works well for problems for which there is a response surface
  - Doesn't work well for more complicated problems
- Supervised Learning
  - Good for problems that have a fixed and labeled data set. Like having a supervisor watch and tell which action agent should have taken. Provides an exact answer.
- Bayesian Optimization
  - Doesn't scale well with data size and dimensionality
- **Reinforcement Learning**
  - Problems where agent must learn by interaction with environment, self-teaching, no need for expert control engineer or labelled data.
  - Used when can formulate problem in terms of Finite Markov Decision Process

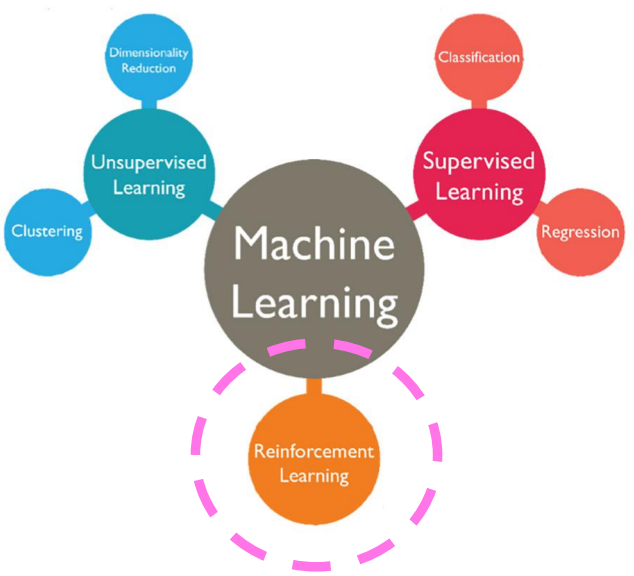
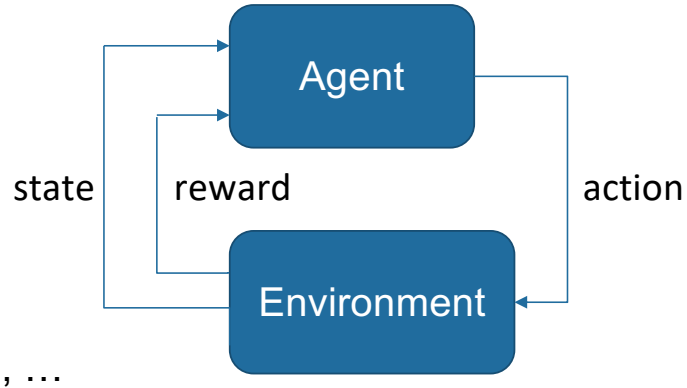
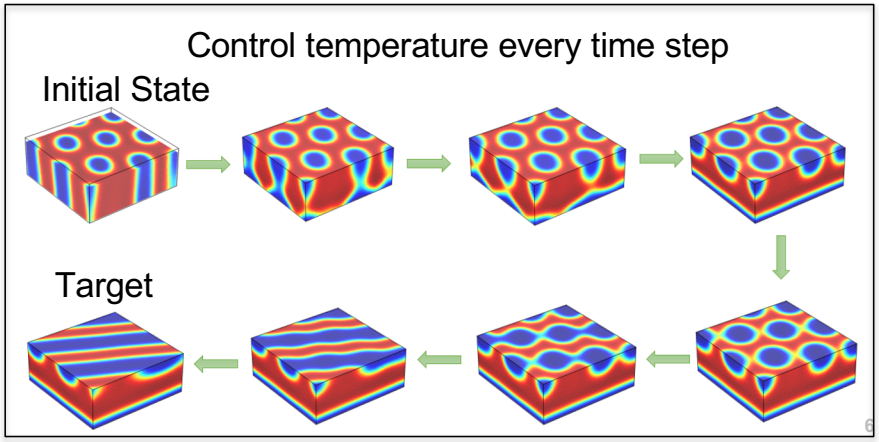



*Image courtesy Vishakha Jha*
*https://www.techleer.com/articles/203-machine-learning-algorithm-backbone-of-emerging-technologies/*

# How to Formulate Problems for Reinforcement Learning

- Finite Markov Decision Process (MDP) fits reinforcement learning:
  - Agent (controller)
  - Environment (controlled system)
  - Action (control signal)
  - Reward (numerical consequence of action)
  - State (representation of environment)
- Sequence of MDP and agent forms trajectory: $S_0, A_0, R_0, S_1, A_1, R_1, \ldots$
- Finite MDP sets of states, actions and rewards have finite number elements
- State includes information on all past agent-environment interaction that make a difference for the future.
- MDP framework is abstract and flexible:
  - Time is fixed intervals or arbitrary successive stages
  - States are low level sensors or high-level abstractions
  - Boundary between agent and environment is flexible



# Motivation for Deep Reinforcement Learning

Complex problems like Go Game have almost infinite possibilities

## Traditional Reinforcement Learning Approach

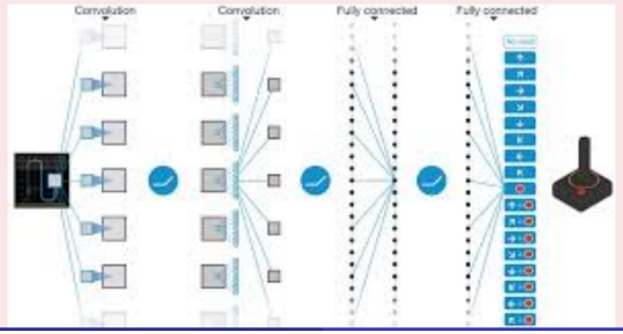
- Generally use a lookup table to decide an action

| | Action 1 | ··· | Action n |
|---|---|---|---|
| State 1 | | | |
| ⋮ | | ⋮ | |
| State m | | | |

## Deep Reinforcement Learning Approach

- A lookup table can be large
  ⇒ **Approximation** by Deep Learning method
- Deep learning generally performs well for this approximation with a big data

(Reference:
`https://skymind.ai/wiki/deep-reinforcement-learning`)

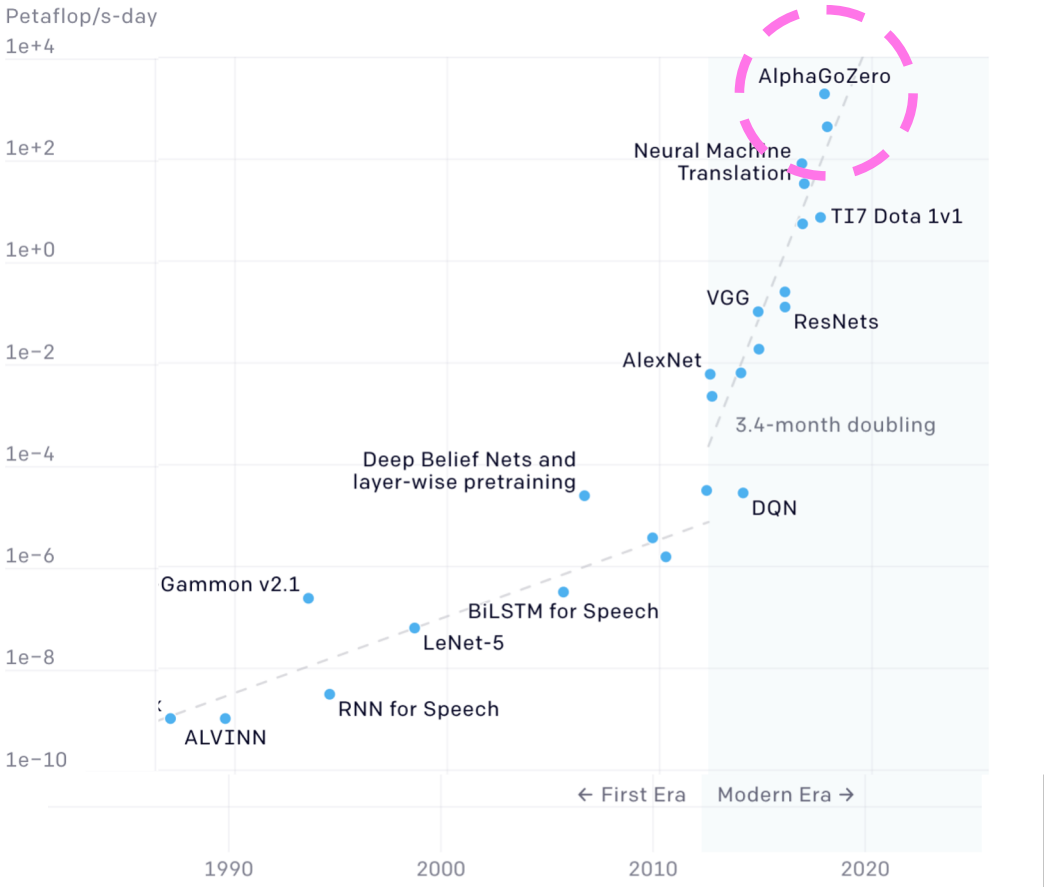


# Why Reinforcement Learning at Scale?

Reasons RL needs to scale:

- Environments may use many computational resources (CPUs, GPUs, etc.)
- Function approximator for complex tasks will use deep ML models
- Large number of actions and/or states
- Policy network ML hyper-parameters
- New RL hyper-parameters will need to be studied

Two Distinct Eras of Compute Usage in Training AI Systems

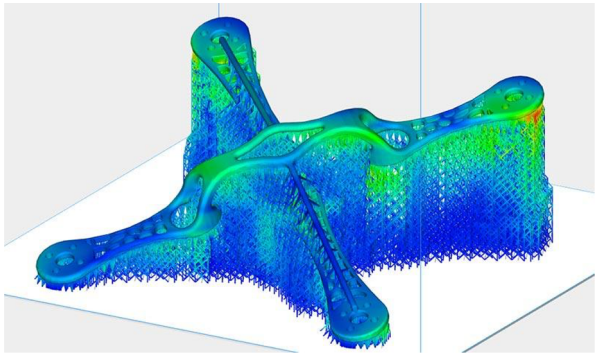



*Excerpted from https://openai.com/blog/ai-and-compute*

# Control Problems in Science

## Simulation

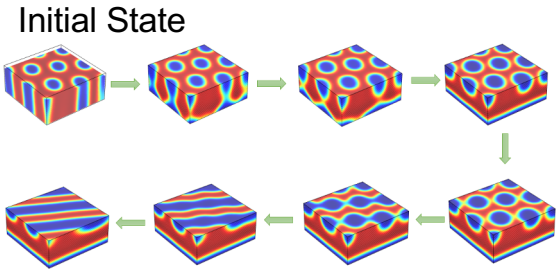
Accelerate sampling in a simulation via search, to reduce computation required for solution



*https://www.materialise.com/en/press-releases/materialise-brings-simulation-for-additive-manufacturing-to-production-floor*

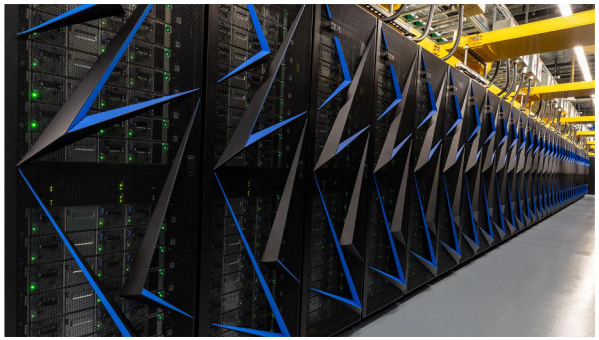
## Experiment

Guide scientific experiments - eg. block copolymer self-annealing




## Operation

- Control HPC facility resource management
- Control experimental facilities (eg x-ray beam )
- Control air, land or space vehicles

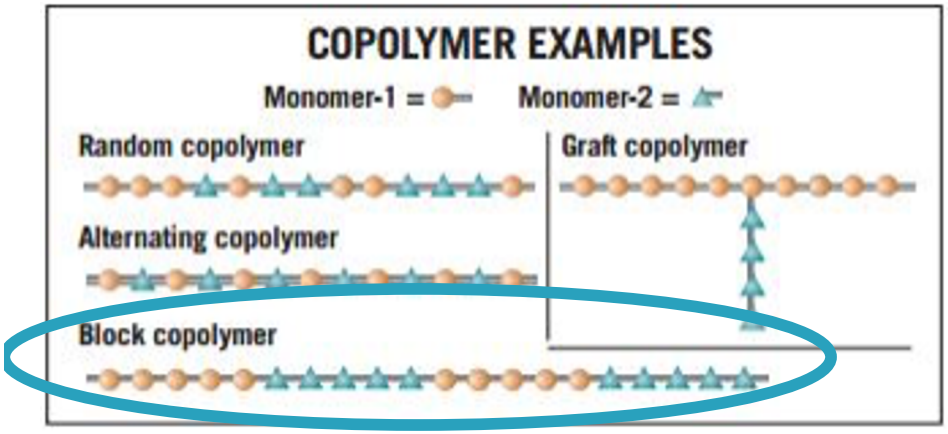


*https://www.olcf.ornl.gov/summit*

# Use Case: Block Co-Polymer Self-Assembly

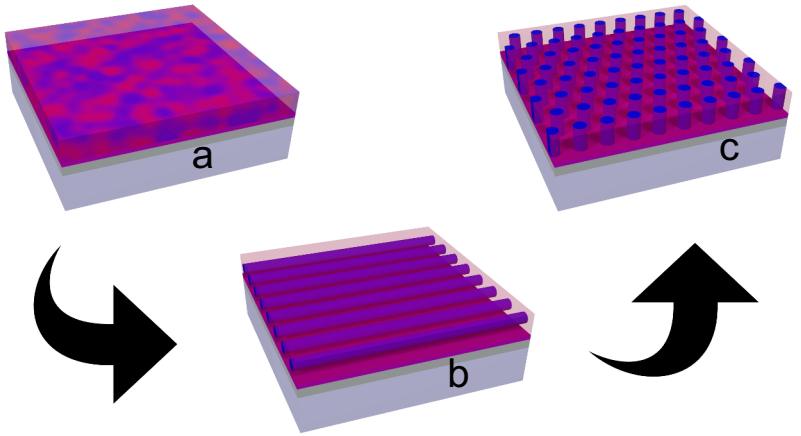
# Use Case #2: Light Source Experiment Control Use Case: Block Copolymer Self-assembly

**What is a Block Copolymer (BCP)?**




*https://www.particlesciences.com/news/technical-briefs/2011/glossary-of-polymer-terms.html*

**How is self-assembly of block copolymers directed?**




*Youngwoo Choo, et al.*

a) Block copolymer is in complete disorder.
b) Laser "orders" BCP into horizontal tubes.
c) High-temp annealing helps achieve desired morphology while maintaining previous order.

**Why do we care about BCPs?**

Combining protein and synthetic polymers can create functional biomaterials **useful for catalysis, sensors, nanotechnology and renewable energy**.

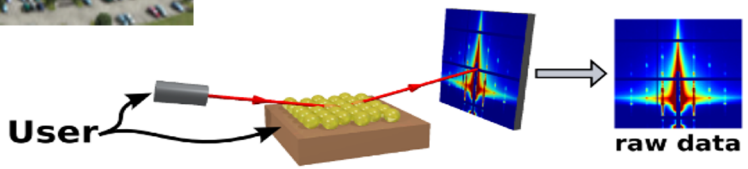
**This process could take hundreds of experimental trials to get right!!**

# Challenges for Block Copolymer Experiments

- BCP experiments are performed at DOE light source user facilities.
- Temperature is adjusted to direct the formation of the block copolymers to a target morphology.

- GISAXS technique is used to detect BCP morphology during directed self-annealing process
- Light source beam shining on sample at small grazing incidence angle produces a diffraction pattern

- The multi-dimensional energy landscape underlying directed block copolymer self-assembly requires engineering a convoluted pathway in order to obtain a target morphology.



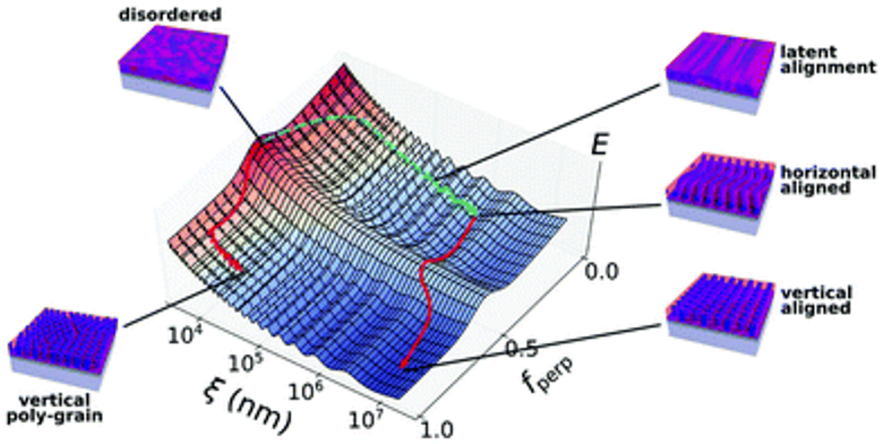
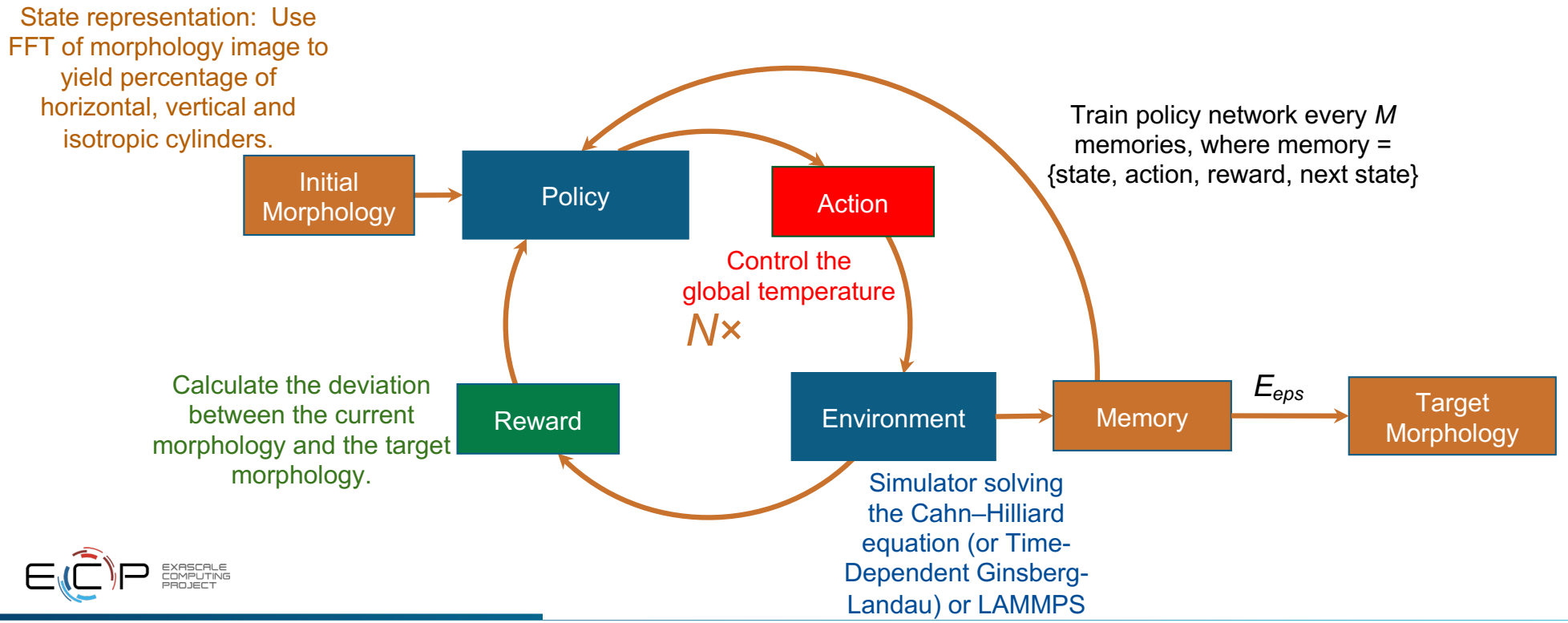




*Image from Nanoscale, 2018, 10, 416. Choo, Majewski, Fukuto, Osuji and Yager.*

# Reinforcement Learning Framework for Block Copolymer

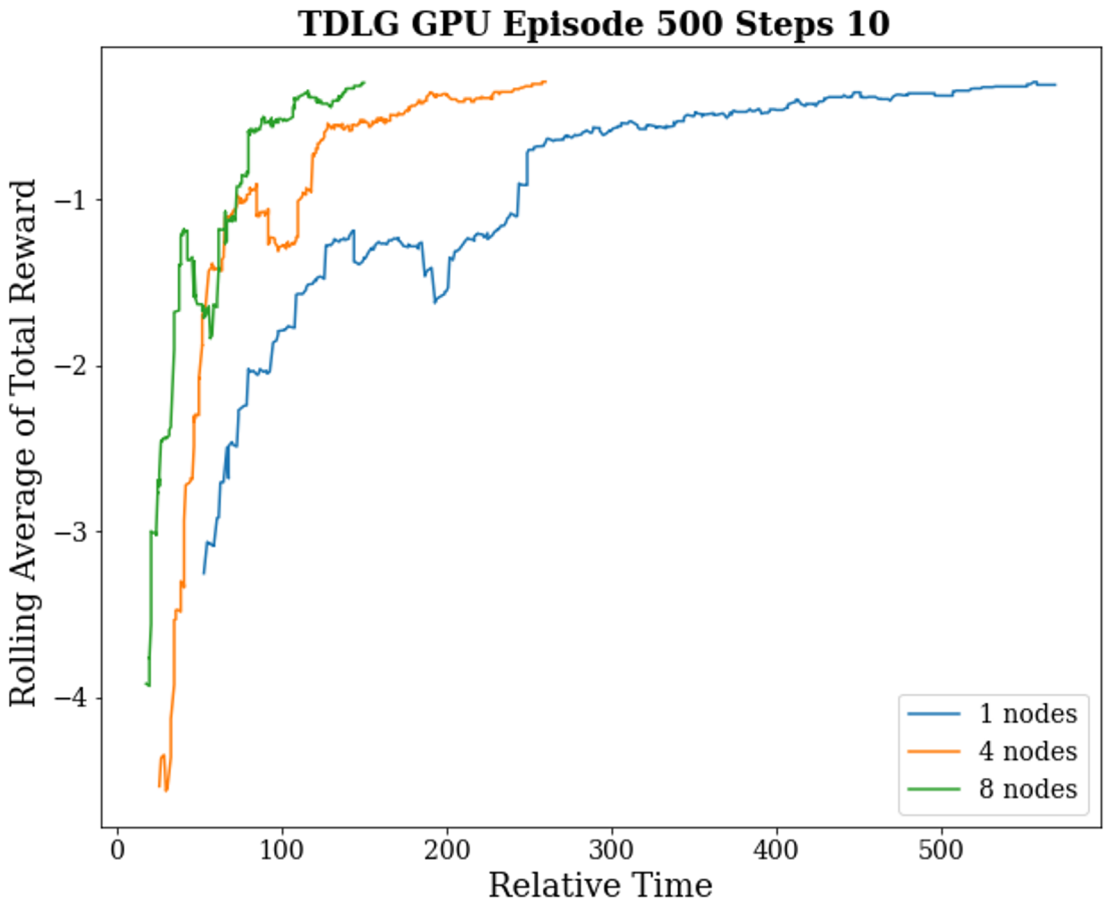
Starting with a initial morphology, can we generate a target morphology?



# Convergence of 3D BCP Reinforcement Learning

Summary

- Scaling studies performed for the 3D Block Copolymer application (TDLG).
- EXARL Workflow: asynchronous learner.
- Six MPI processes per node.
- Approximately 5x improvement in convergence going from 6 processes (1 node) to 48 processes (8 nodes) for equivalent reward.
- Application problem is relatively small and does not scale well beyond 8 nodes.



# EXARL Overview

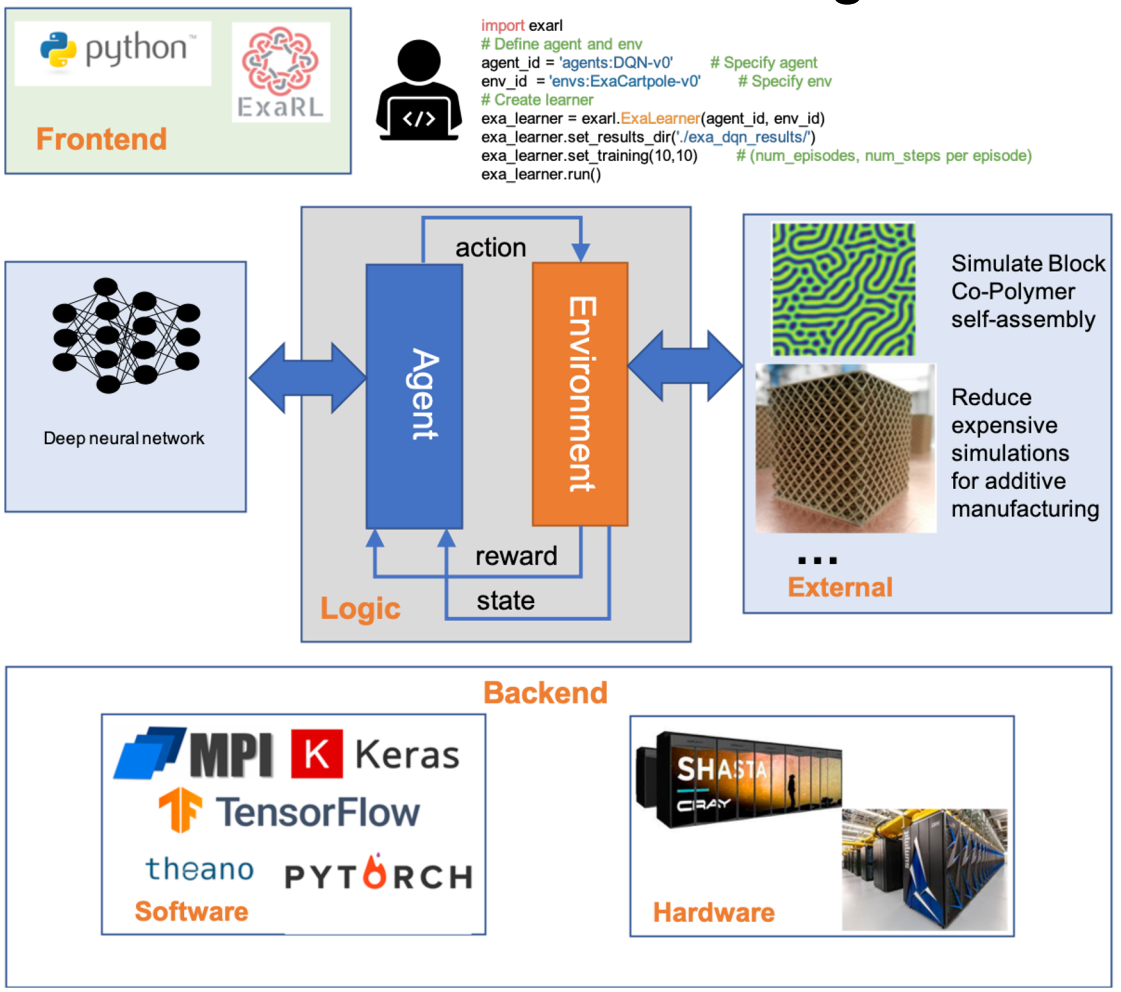
# Requirements for an Exascale RL Framework?

- Most RL research cases typically have simple environments
  - Actions in scientific environments can take multiple hours to get a response even on large clusters (eg NWChem, LAMMPS)
- Complexity of scientific problems requires scalable RL:
  - Need scalable learning
  - Need ability to run environments in parallel on multiple nodes
- Open AI Gym is not sufficient for scientific problems:
  - No inherent support for scalable environments
  - Doesn't provide scalable algorithms for science problems
    - has 'bring your own algorithm' philosophy
    - No means to register RL agents
    - Create your own workflow

→ Scalable, composable, flexible framework for RL

# Easily eXtendable Architecture for Reinforcement Learning (EXARL)

- EXARL: scalable RL framework for scientific environments
- Extends OpenAI Gym's environment registry to agents and workflows
- Scalable multi-node environments
- Abstract classes to mandate necessary functionality
- Easy to register new agents, environments, and workflows
- Supports different hardware and software infrastructures
  - Use existing prevalent infrastructure



# Modular Agents, Environments, and Workflows in EXARL

**Environment**

- **Init**
  - Initialize global variables
  - Application setup, etc.
- **Step**
  - Takes an action run
  - Update the internal state
- **Reset**
  - Reset the environment
- **Render**
  - Visualize the environment

**Agent**

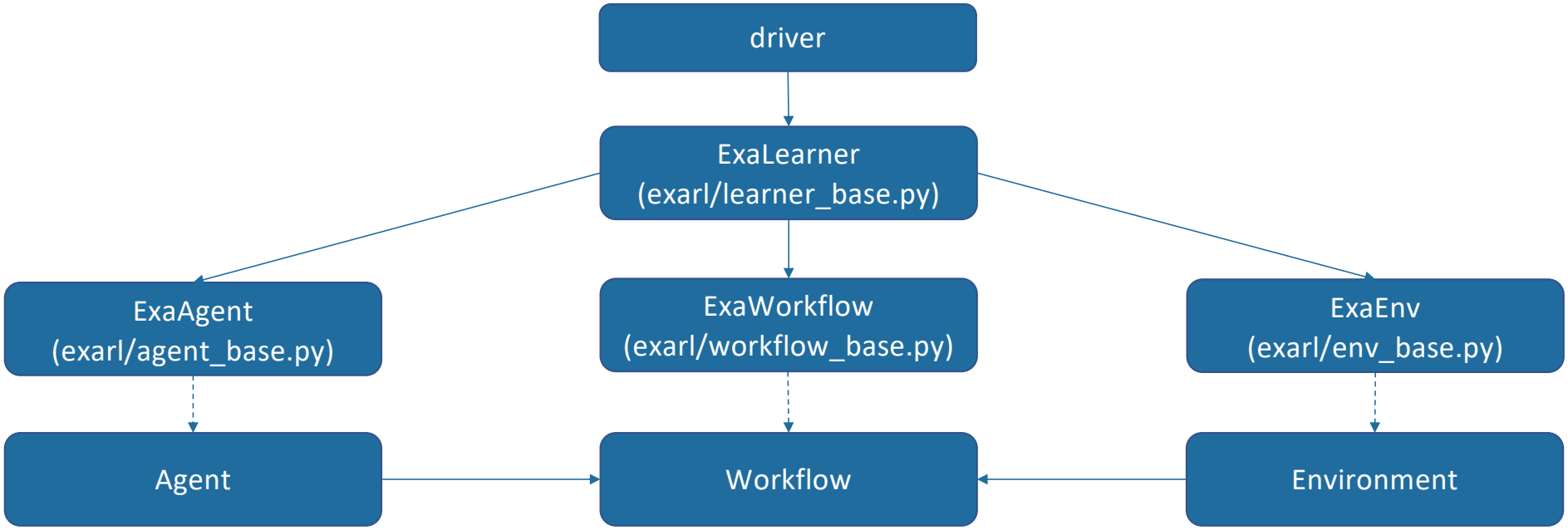
- **Init**
  - Initialize global variables
  - Target networks, active networks, optimizer, learning rate
- **Train**
  - Training the active network using mini batches
- **Action**
  - Use target networks to determine next action
  - Sampling technique
- **Load/Save**
  - Load or save network model

...

**Workflow**

- **Init**
  - Initialize global variables
  - Target networks, active networks, optimizer, learning rate
- **Run**
  - Load or save network model

# Class Hierarchy in EXARL



# Registration Directory Structure

- agents
  - __init__.py
  - agent_vault
    - __init__.py

```python
from .registration import register

register(
  id='ExaFooAgent-v0',
  entry_point=agents.agent_vault:ExaFooAgent
)
```

- envs
  - __init__.py
  - env_vault
    - __init__.py

```python
from gym.envs.registration import register

register(
   id='ExaFooEnv-v0',
   entry_point=envs.env_vault:ExaFooEnv
)
```

- workflows
  - __init__.py
  - workflow_vault
    - __init__.py

```python
from .registration import register

register(
   id='ExaFooWorkflow-v0',
 entry_point='workflows.workflow_vault:ExaFooWorkflow'
)
```

# EXARL is Easy to Use

```python
from mpi4py import MPI
import exarl

# MPI communicator
comm = MPI.COMM_WORLD
# Create learner
exa_learner = exarl.ExaLearner(comm)
# Run the learner
exa_learner.run()
```

- Example configuration:

```
{
    "agent": "DQN-v0",
    "env": "CartPole-v0",
    "workflow": "async",
    "n_episodes": 10,
    "n_steps": 10,
    "output_dir": "./results_dir/",
    "process_per_env": 1,
    "model_type": "LSTM",
    "action_type": "variable",
    "log_level": [3, 3],
    "profile": "none"
}
```


- Use existing agents, environments, and/or workflows
  - If not, register custom agents, environments and/or workflows
- Specify the configuration
- Framework supports and handles scaling
- CANDLE integration

# Config Files

| Components | Directory | Example JSON file |
|---|---|---|
| main | exarl/config/ | learner_cfg.json |
| agent | exarl/config/agent_cfg/ | DQN-v0.json |
| env | exarl/config/env_cfg/ | ExaBoosterDiscrete-v0.json |
| workflow | exarl/config/workflow_cfg/ | default_workflow_cfg.json |

**learner_cfg.json**

```
{
  "agent": "DQN-v0",
  "env": "CartPole-v0",
  "workflow": "async",
  "n_episodes": 10,
  "n_steps": 10,
  "process_per_env": 1,
    ...
}
```

**DQN-v0.json**

```
{
  "gamma": 0.75,
  "epsilon": 1.0,
  "epsilon_min" : 0.01,
  "epsilon_decay" : 0.999,
  "learning_rate" : 0.001,
  "batch_size" : 5,
    ...
}
```

**ExaBoosterDiscrete-v0.json**

```
{
  "model_file": "fullbooster.h5",
  "model_dir": "None",
  "booster_data_dir" : "booster_data"
  "data_file" : "BOOSTR.csv",
  "url" : "https://zenodo.org/…",
  "max_steps" : 100,
    ...
}
```

**default_workflow_cfg.json**

```
{
  "default_workflow_cfg": "You can specify your workflow config here!"
}
```

# Hyperparameter Optimization: CANDLE Integration

- CANDLE provides integration at 2 levels
- Benchmark level
  - Automatically provides command line override of all the relevant parameters
  - Parses the json file to generate a dictionary of parameters which are passed into the code
  - Can perform small-scale experiments to test effects of parameters
    - e.g. `mpirun -np 3 python driver/driver.py --n_steps 100 --epsilon 0.1`
- Supervisor level
  - Manages a large allocation and farms out work to run multiple learners in parallel, with
  - Allows automated sweeps and optimization at scale
    - e.g grid, random, mlrMBO
- CANDLE originally developed for simpler DL workflows
  - Extended for EXARL to accommodate multi-component, multi-node RL workflows
    - added json file parsing
    - added multi-node jobs with heterogeneous components

# Software Requirements to Run EXARL (build from scratch)

- Python 3.7
- Tensorflow 2.0
- OpenAI gym
- Numpy
- MPI4py 3.0.3 or above

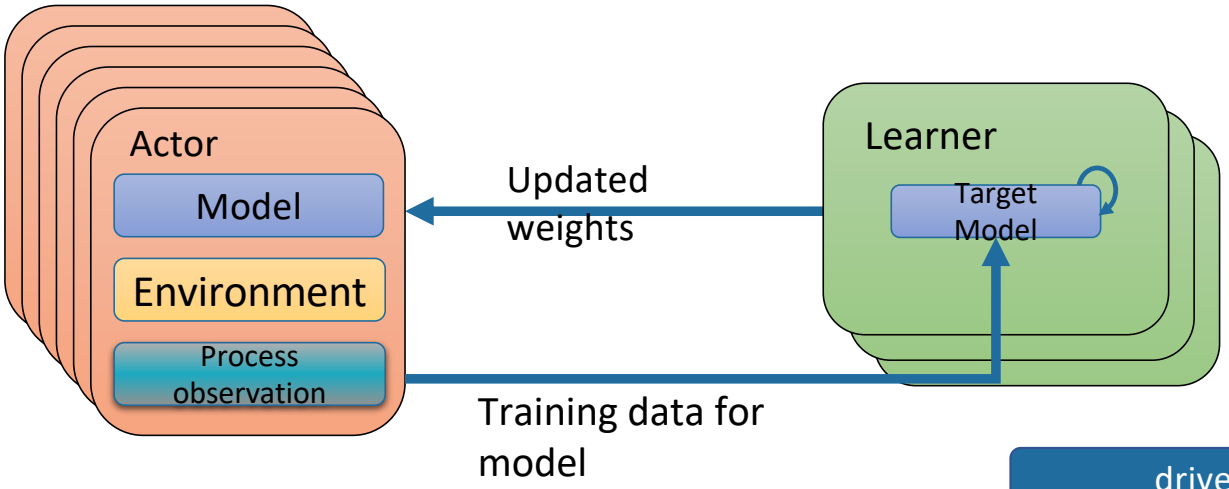
Additional dependencies are listed in setup.py

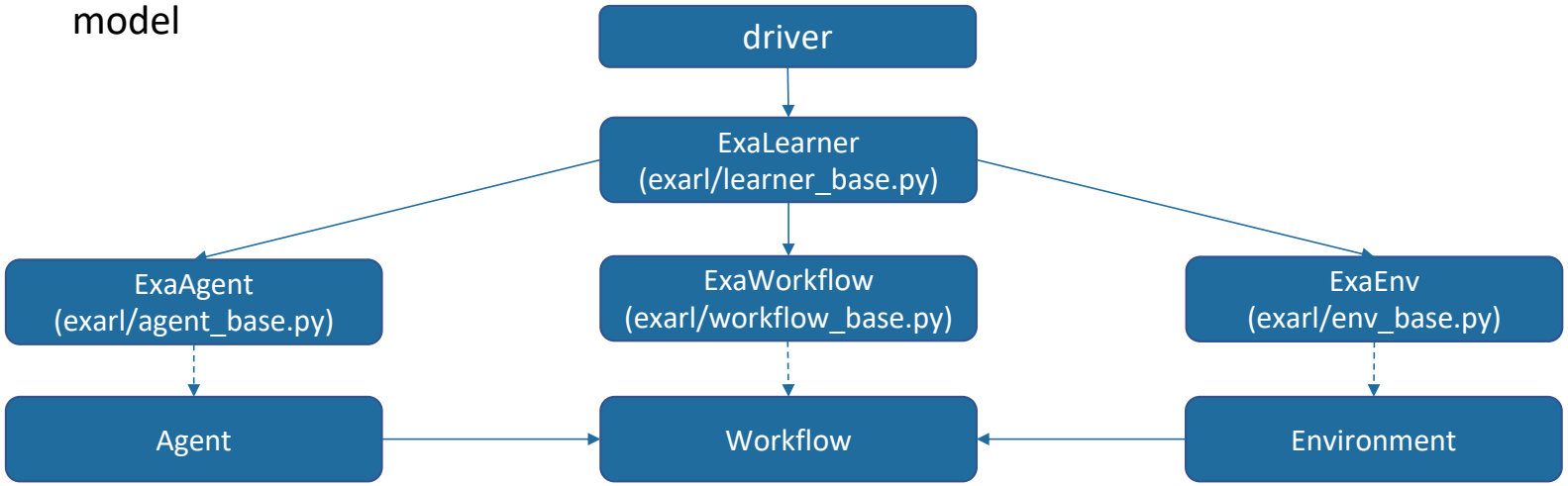
For some systems it is required to build MPI4py using local MPI libraries.

```
wget https://github.com/mpi4py/mpi4py/archive/3.0.3.tar.gz
    tar -xvzf 3.0.3.tar.gz
    python setup.py build --mpicc `which mpicc`
    python setup.py install --user
```

# Scientific Application Implementations and EXARL Customization

# Components: Agent, Environment, Workflow Overview



# EXARL Available Environments

All OpenAI Gym Environments*

- ExaCartPole-v1
- ExaCH-v0
- ExaCovid-v0
- ExaBoosterDiscrete-v0
- ExaWaterClusterDiscrete-v0

```
# Creating custom environments
import gym

class customEnvironment(gym.Env):
        metadata = {'render.modes': ['human']}
        def __init__(self):
                pass
        def step(self, action):
                pass
        def reset(self):
                pass
        def render(self, mode='human',
close=False):
                pass
```

*For purposes of the tutorial on Cori, ExaCartPole-v1 and ExaBoosterDiscrete-v0 are currently ported to Cori. The others require building low level C/C++/Fortran code and have been tested on Summit.

# Custom Environment

EXARL uses OpenAI Gym's environment registry

Example:

```
class ExaFooEnv(gym.Env):
        ...
```

Environment must include the following functions:

```
__init__() # constructor for initialization
step()     # environment step
reset()    # reset the environment
render()   # render for visualization
```

Register the environment in ExaRL/envs/__init__.py

```
from gym.envs.registration import register

register(
   id='ExaFooEnv-v0',
   entry_point=envs.env_vault:ExaFooEnv
)
```

The id variable will be passed to exarl.make() to call the agent.

The file ExaRL/envs/env_vault/__init__.py should include:

```
from envs.env_vault.foo_env import ExaFooEnv
```

where ExaRL/envs/env_vault/foo_env.py is the file containing your environment.

# EXARL Available Agents

**Deep Q-Network (DQN)**

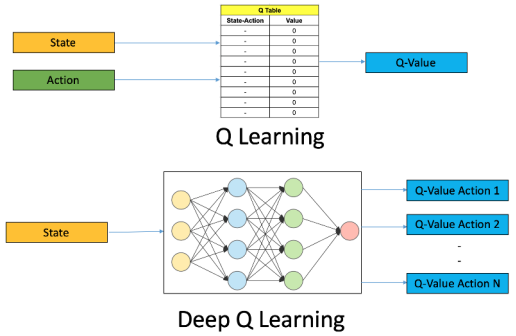



Image source: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

**Deep Deterministic Policy Gradient (DDPG)**

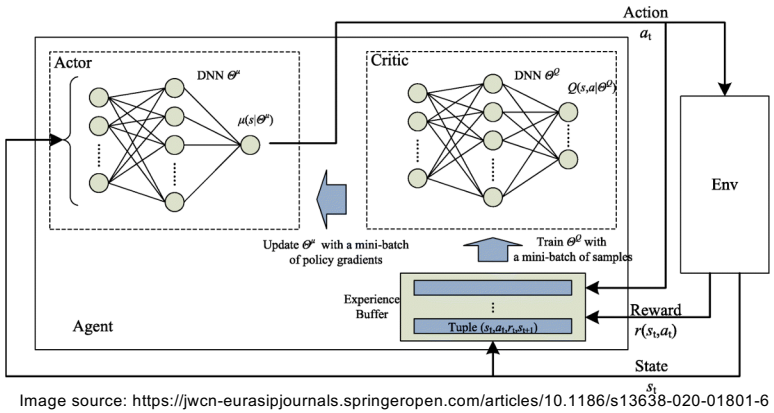



Image source: https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-020-01801-6

```python
# Creating custom agents
import exarl as erl

class customAgent(erl.ExaAgent):
        def __init__(self):
                pass
        def get_weights(self):
                pass
        def set_weights(self):
                pass
        def train(self):
                pass
        def action(self):
                pass
        def load(self):
                pass
        def save(self):
                pass
        def set_learner(self):
                pass
```

# Custom Agents

- EXARL extends OpenAI Gym's environment registry to agents
- Agents inherit from exarl.ExaAgent

Example:

```
class ExaFooAgent(erl.ExaAgent):
        ...
```

Agents must include the following functions:

```
__init__()       # constructor for
initialization
get/set_weights()# get/set weights
load/save()      # load/save weights
action()         # infer action
train()          # train  NN
set_learner()    # set process as learner
```

- Register the agent in ExaRL/agents/__init__.py

```
from .registration import register

register(
  id='ExaFooAgent-v0',
  entry_point=agents.agent_vault:ExaFooAgent
)
```

The id variable will be passed to exarl.make() to call the agent.

The file ExaRL/agents/agent_vault/__init__.py should include:

```
from agents.agent_vault.foo_agent import ExaFooAgent
```

where ExaRL/agents/agent_vault/agent_env.py is the file containing your environment.

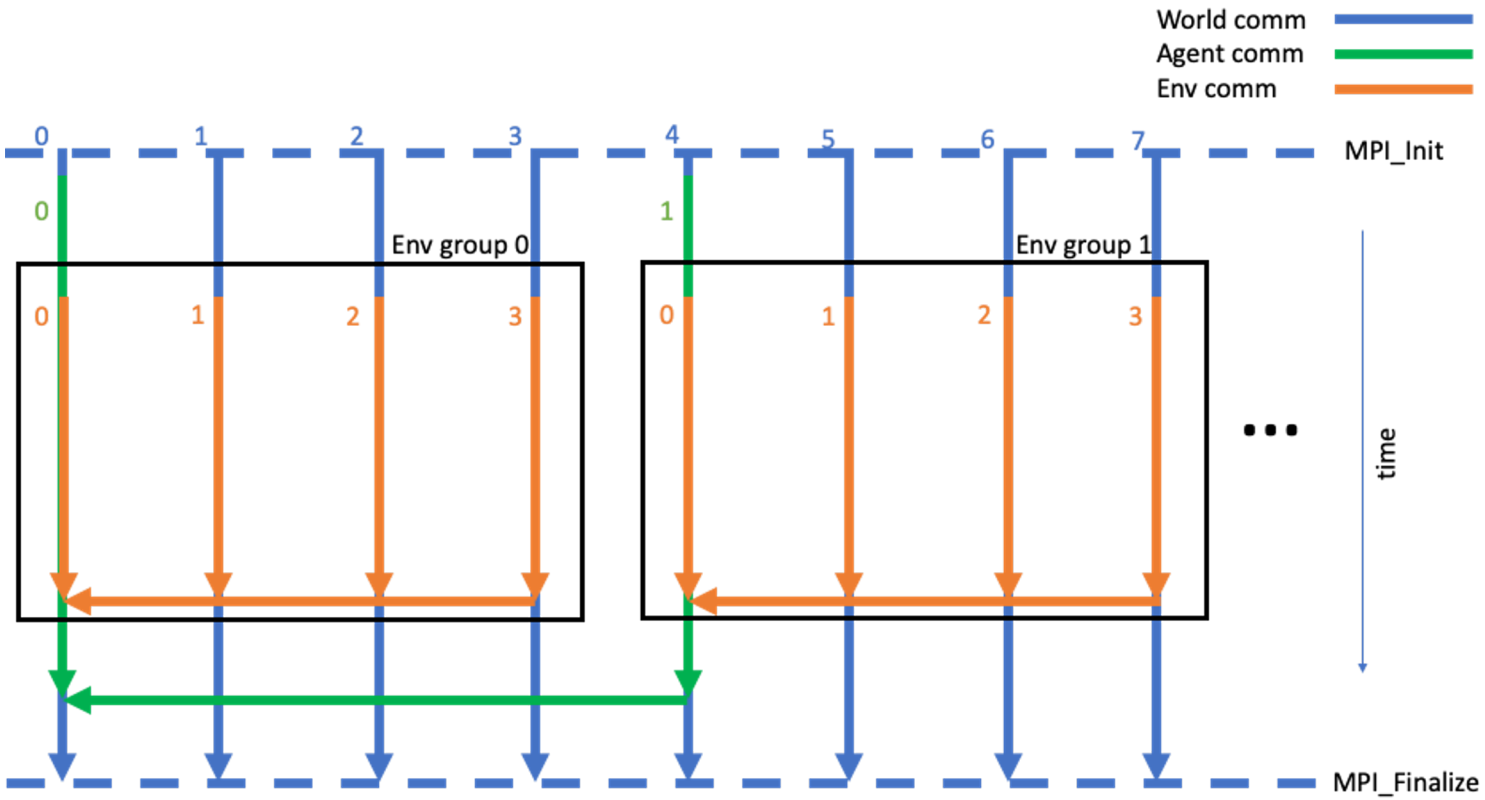
# EXARL Available Workflows

- SYNC - Synchronous MPI process launch
- ASYNC - MPI spinner
- RMA - Uses one sided communication

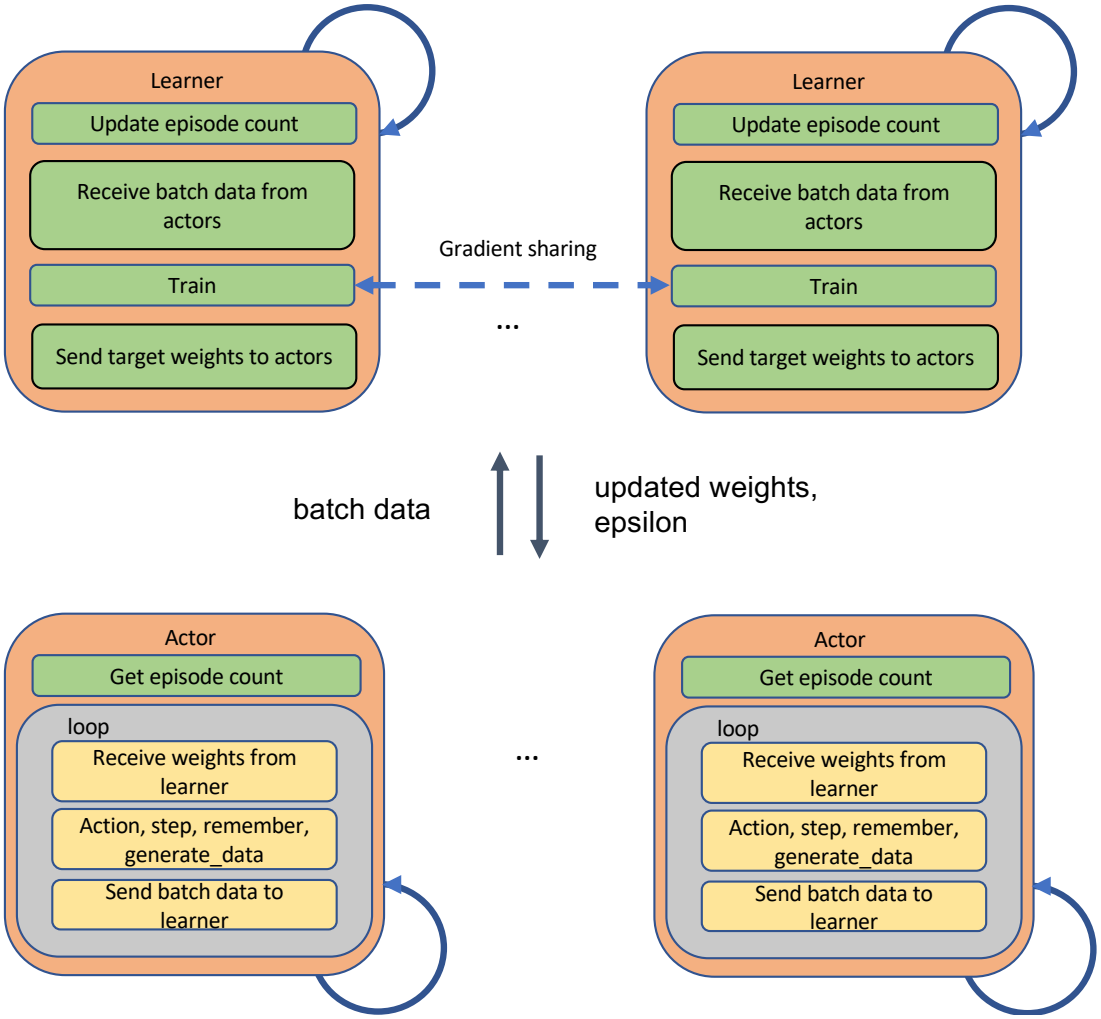
```
# Creating custom workflows
import exarl as erl

class customWorkflow(erl.ExaWorkflow):
      def __init__(self):
            pass
      def run(self):
            pass
```

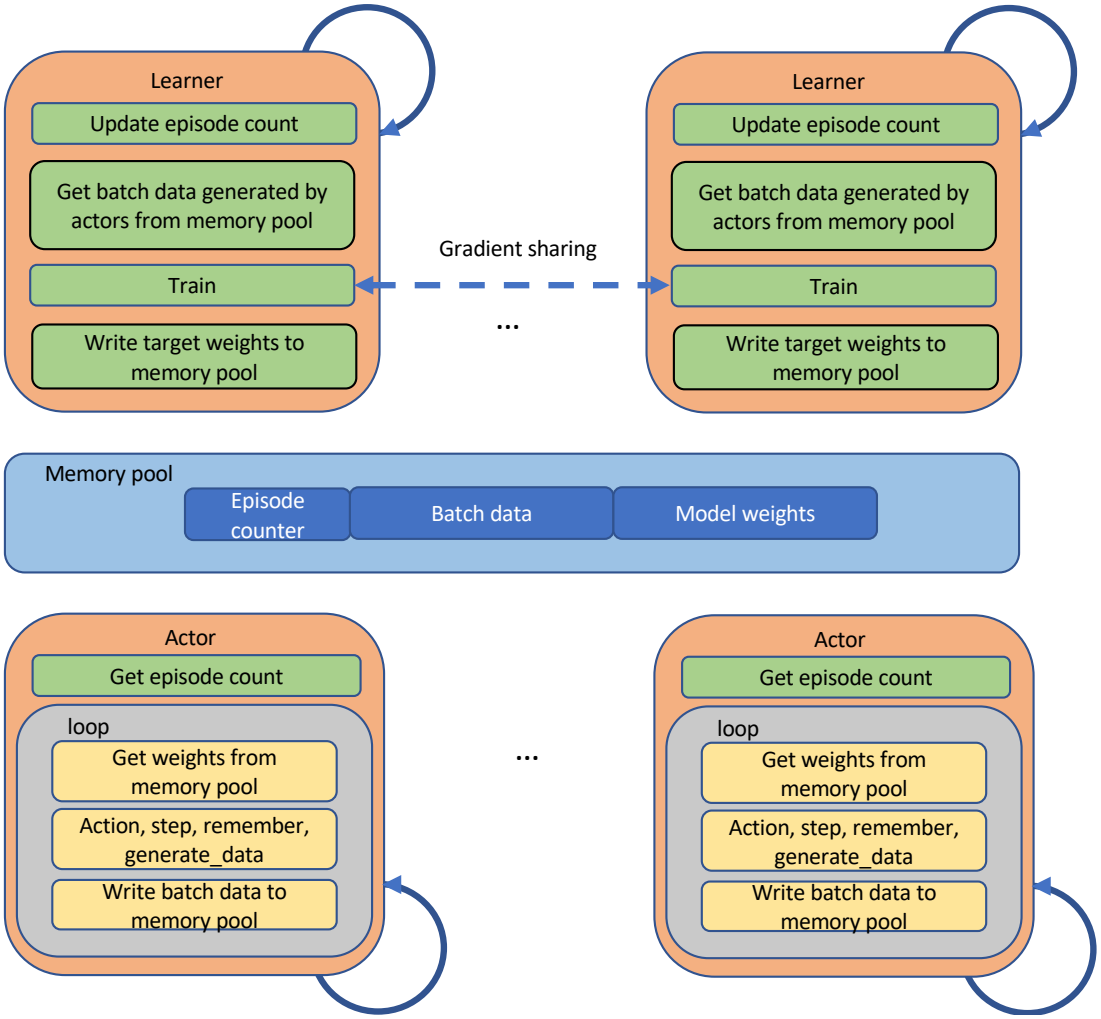
# Custom Workflows - SYNC



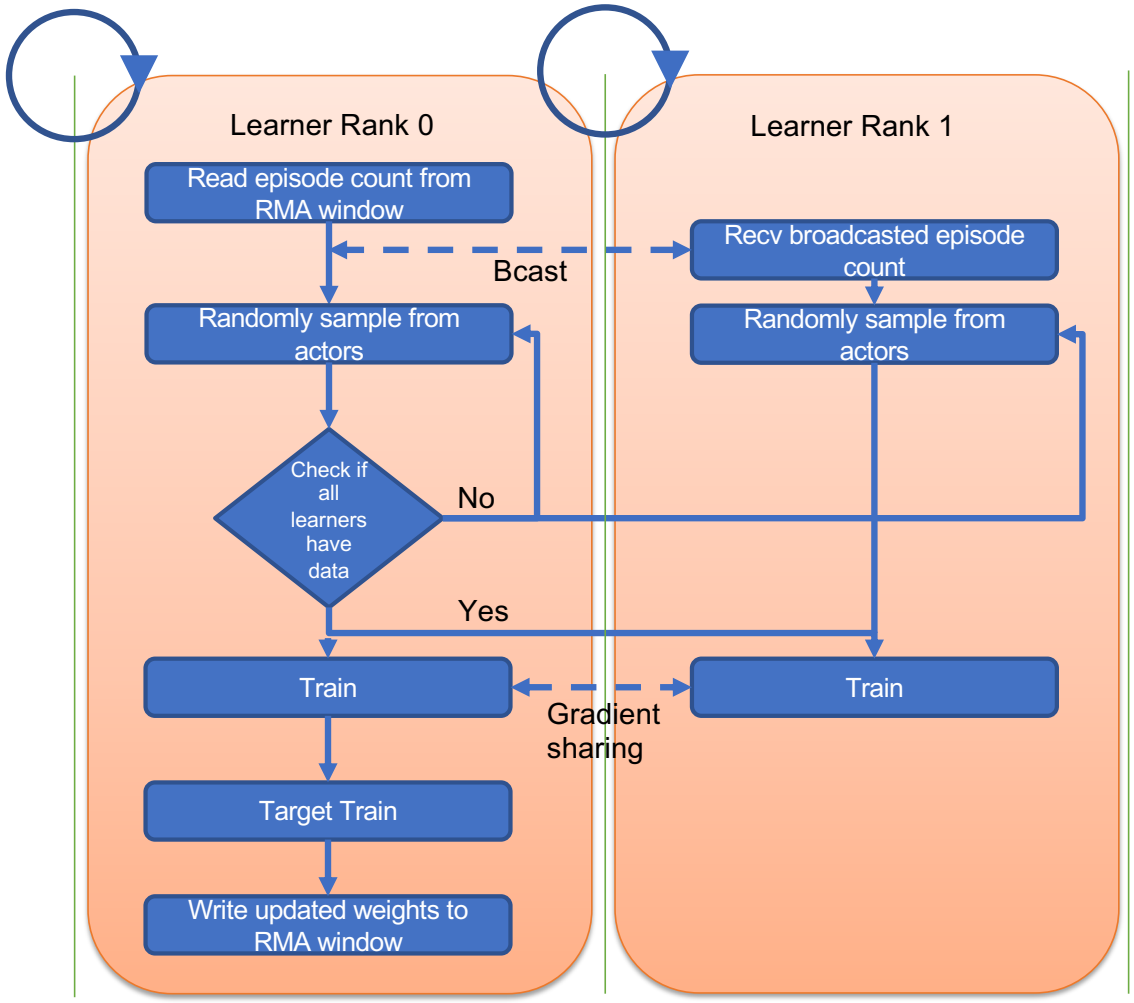
# Custom Workflows - ASYNC



# Custom Workflows - RMA



# Multi-learner using Horovod




...

# Questions?

*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.*

EXASCALE COMPUTING PROJECT