Scalable Reinforcement Leaning on Cray Systems

Benjamin Robbins, Director - AI & Advanced Productivity
Ananda Kommaraju, Kristyn Maschhoff, Mike Ringenburg

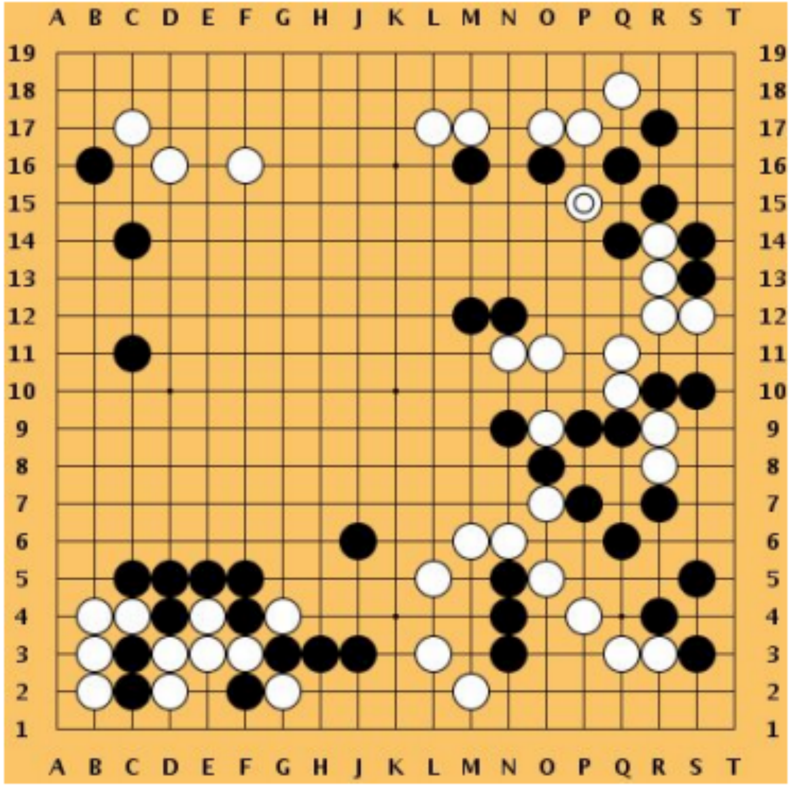
Overview

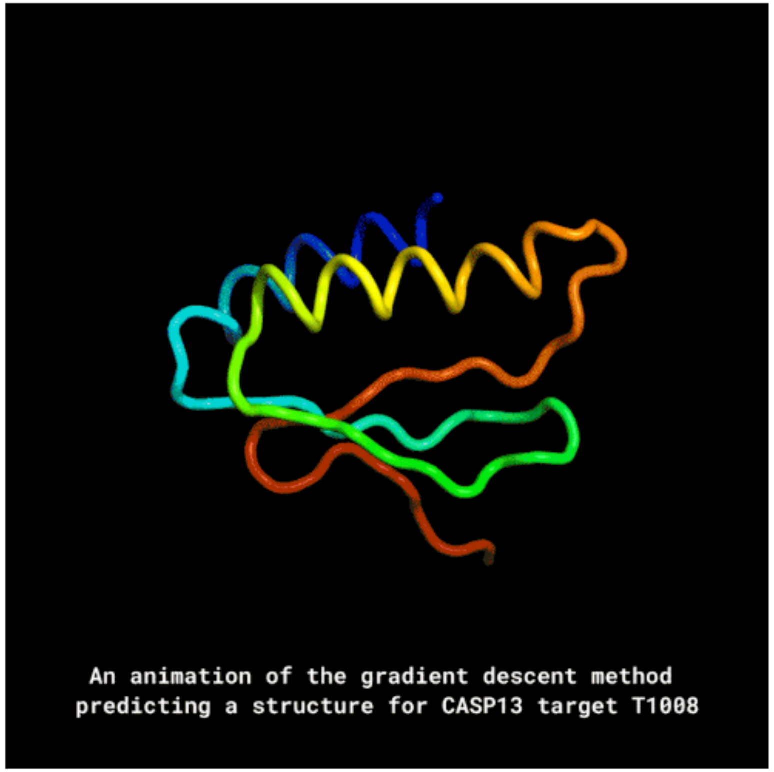
Purpose: Demonstrate steps to reproduce a working distributed reinforcement learning configuration on a Cray system. Platform for further exploration.

Relevance: Reinforcement learning approaches have produced many of the recent state of the art results for Machine Learning. Reinforcement learning is resource and communication intensive and is, therefore, an excellent candidate to take advantage of high-performance-computing.

Results: Cray systems' ability to support mixed node types (GPU, CPUs) and resource configuration flexibility make for an ideal platform to explore and push limits of reinforcement learning.

Recent Advances in AI





Reinforcement Learning

Learning from the rewards of a given action.

Reinforcement Learning (RL)

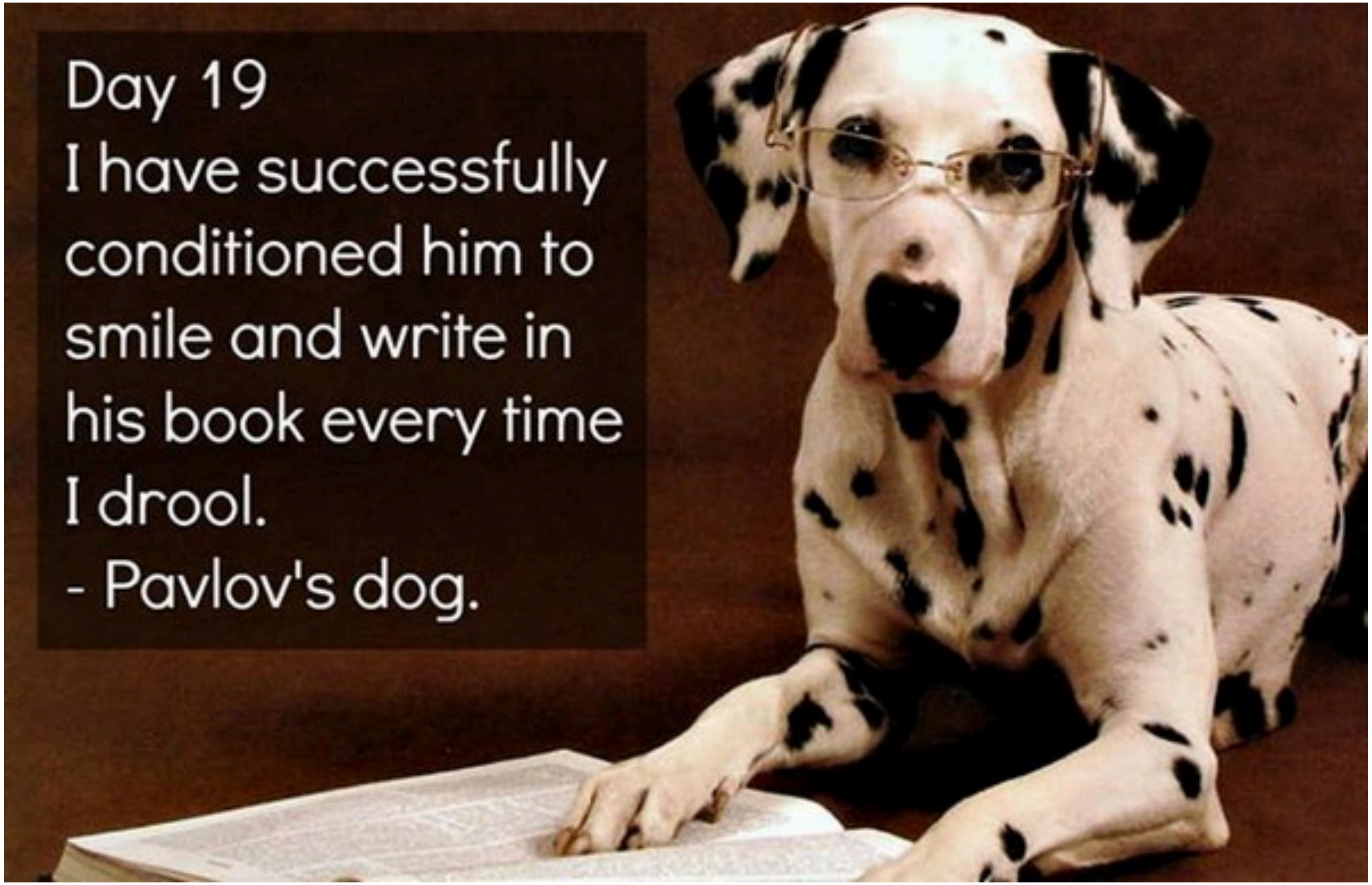
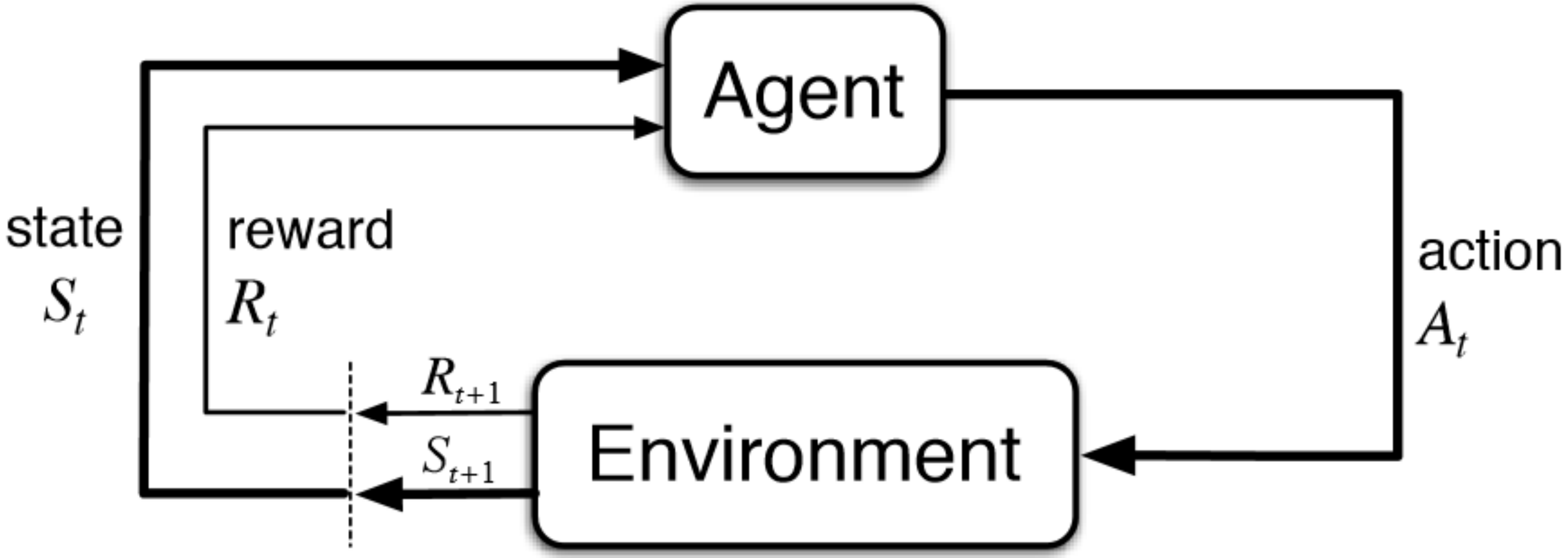


Image: geekshumor.com

Reinforcement Learning (RL) Basics

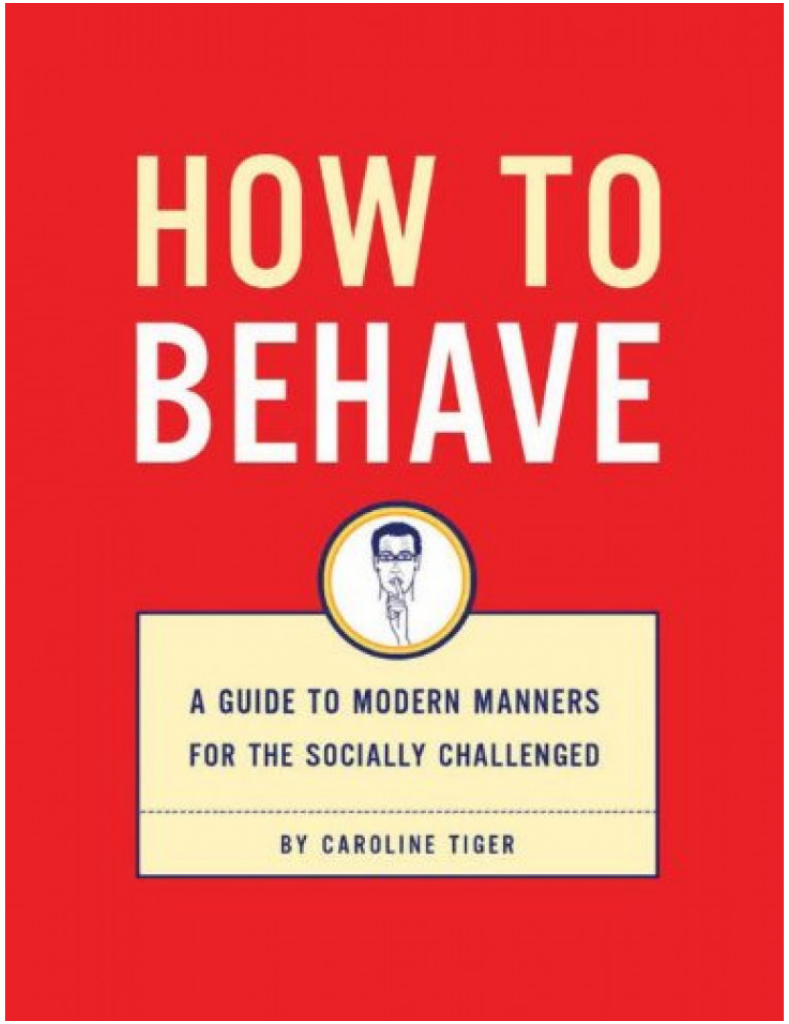


Reinforcement Learning Illustration (https://i.stack.imgur.com/eoeSq.png)

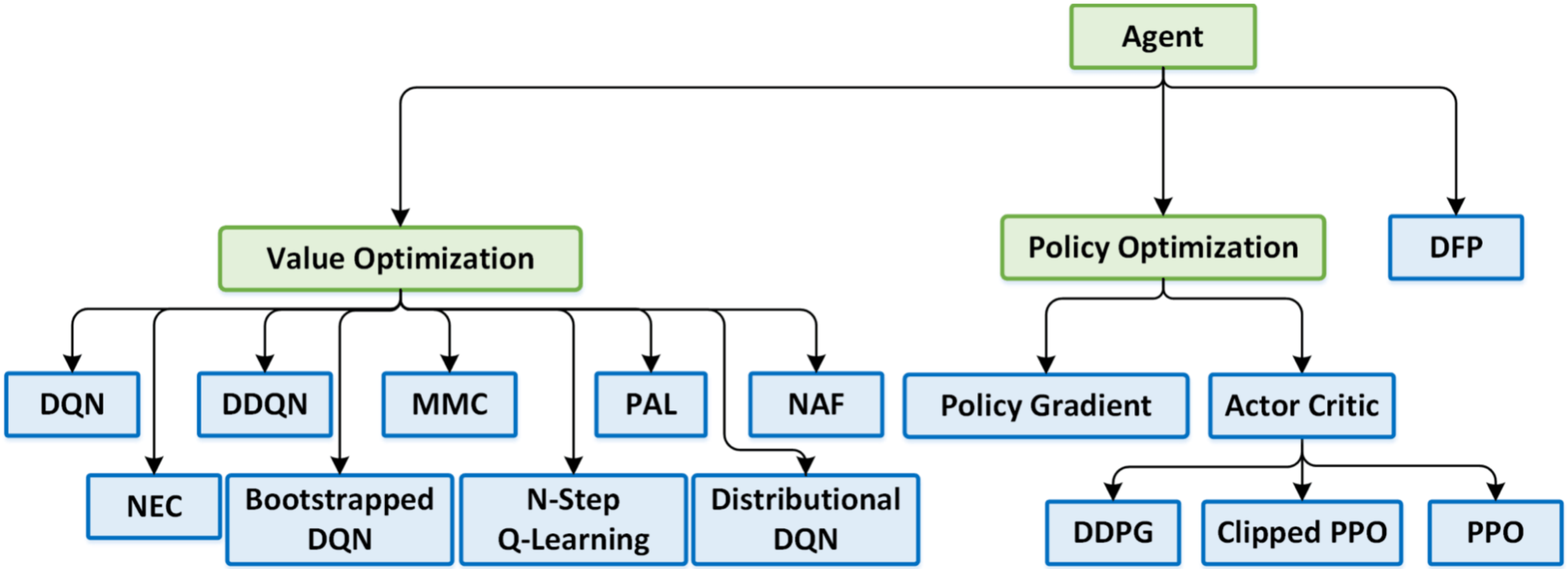
The Goal of RL?

Find the optimal Policy.

The policy tells the agent how to act given a particular state.

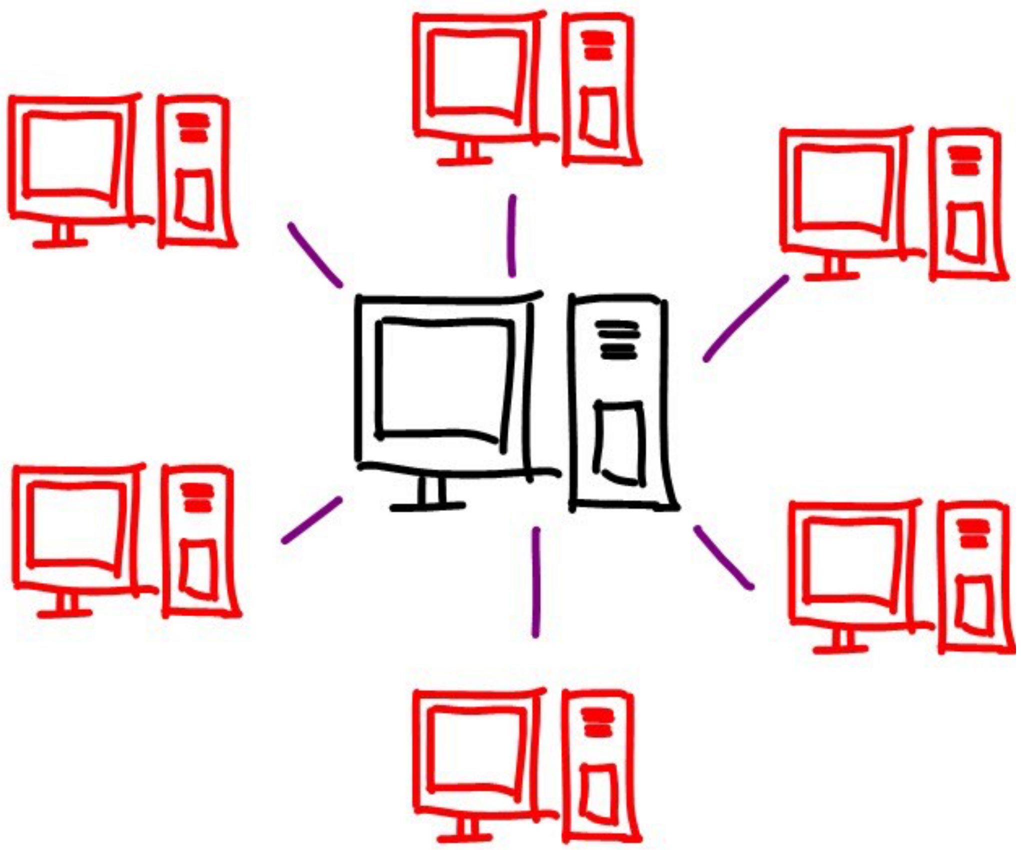


Many RL Methods



https://www.intel.ai/introducing-reinforcement-learning-coach-0-10-0/#gs.91i90q

Distributed Reinforcement Learning



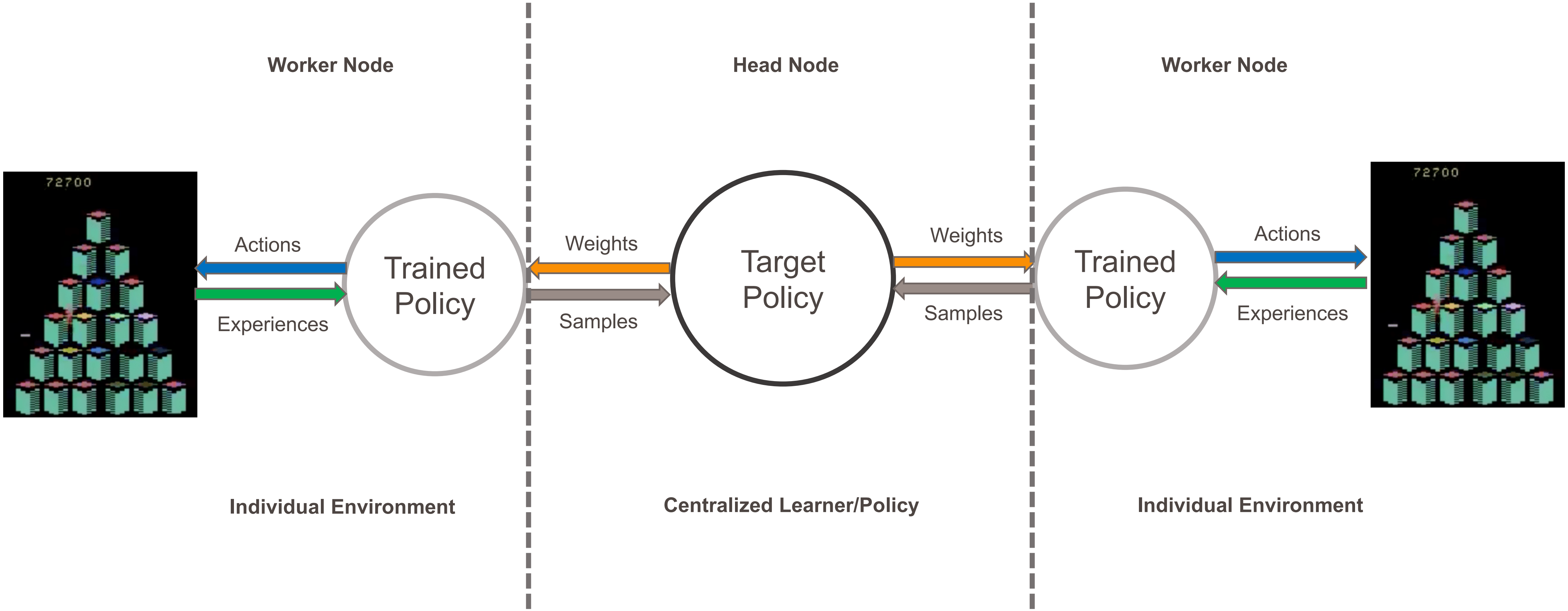
https://cdn-images-1.medium.com/max/1600/1*tYxWuyksovxA1Thu8PggPQ.jpeg

Motivation for Study of Distributed RL on Cray Systems

- Explore Reinforcement Learning algorithms
- Understand if possible to leverage existing open source frameworks
- Study nested parallelism and complex workflows
- Distributed RL is very active area of research. Participate in research and make available for other Cray end-users
- Scalability is a requirement for many real-world problems. How can distributed RL at the scale of Cray help solve these problems?

Environments for Study

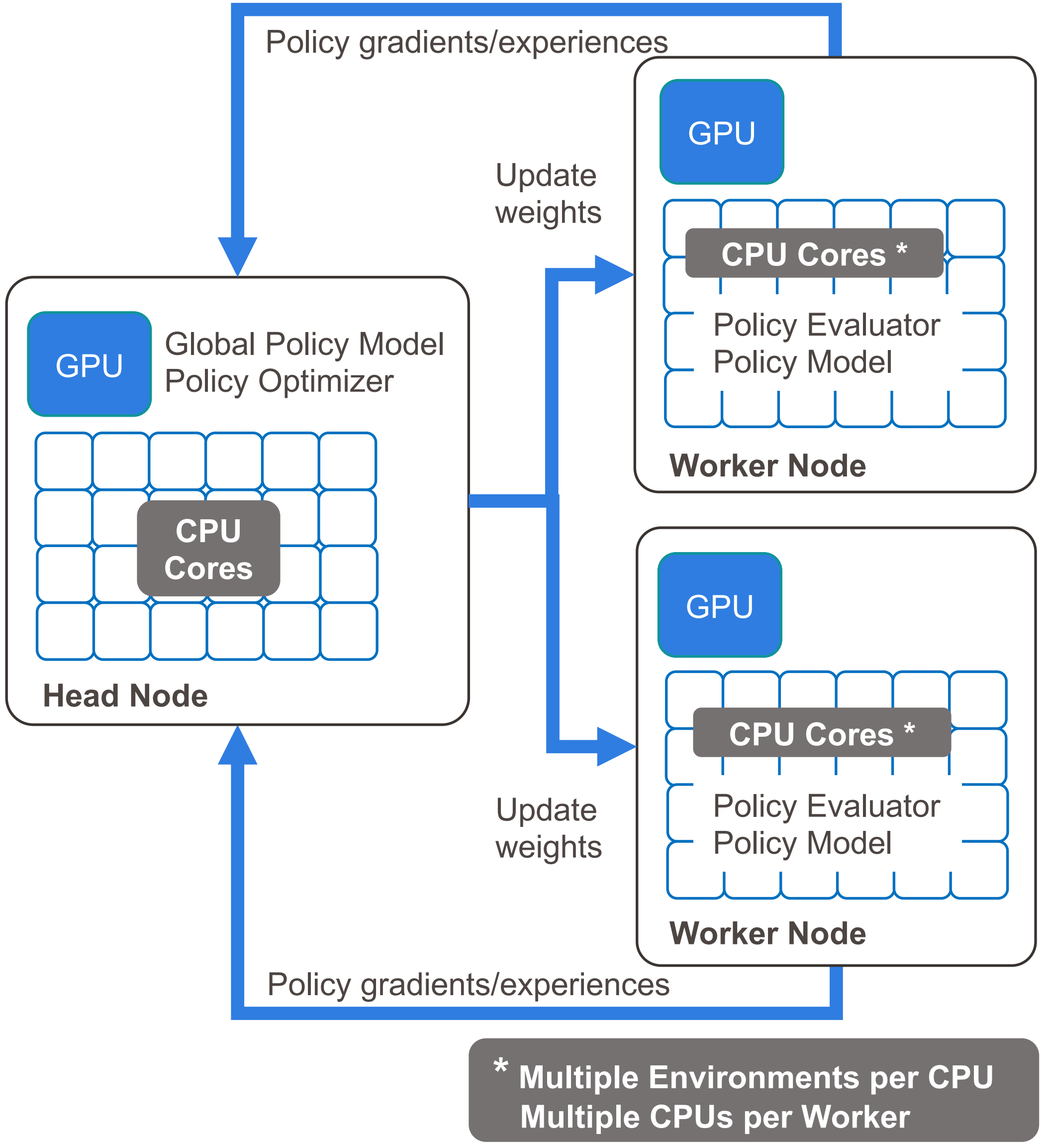
Distributed Reinforcement Learning Generalized



Setup and Configuration

Mapping RL to Cray XC

- Nested Parallelism
- Resource Scaling
- Multiple algorithms can be mapped



What is Ray?

- Ray is a flexible, high-performance distributed execution framework.
- Developed by RISELab at UC Berkeley
- Very active open-source project – https://ray.readthedocs.io/en/latest/index.html
- Ray has libraries for tuning and reinforcement learning
- RLlib is Ray’s reinforcement learning library
- RLlib supports many RL methods.

Deploy a Ray Cluster on Cray XC

- Allocate nodes through SLURM workload manager
- ccmlogin (Cluster compatible mode)

```
$salloc -N 1 -p ccm_queue -C P100 --gres=gpu --
    exclusive
$module load ccm
$ccmlogin -V
```

- Start a Ray head node with initial settings

```
#!/bin/bash
source activate ray
IP=$(ip -oneline -family inet addr list ipogif0 \
| head --lines 1 | grep --perl-regexp \
--only-matching 'inet \K[\d.]+')
echo $IP:6380 > $HOME/ray_head_node

ray start --head --node-ip-address=$IP --redis-port
    =6380
```

Deploy a Ray Cluster on Cray XC

- Independently start Ray workers with initial settings
- Resource scaling - Increase or reduce workers/resources

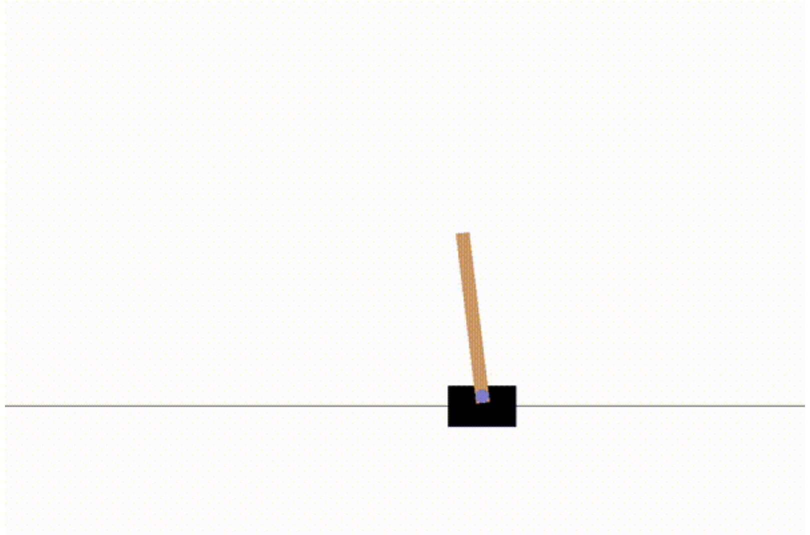
```
$salloc -N 1 -C P100 --gres=gpu --exclusive
$srun sh worker.sh
```

```
#!/bin/bash
source activate ray
HEAD_IP=$(head -n 1 ray_head_node)
ray start --redis-address $HEAD_IP
while [ 1 ];
do
        sleep 1
done
```

Execute Rllib algorithms

```
import ray
from ray.rllib.agents.ppo import PPOAgent
from ray.tune import run_experiments


def train_fn(config, reporter):
    agent1 = PPOAgent(env="CartPole-v0", config=config)
    for _ in range(100000):
        result = agent1.train()
        result["phase"] = 1
        reporter(**result)
        phase1_time = result["timesteps_total"]
    state = agent1.save()
    agent1.stop()
if __name__ == "__main__":
    ray.init(redis_address="10.128.0.225:6380")
    run_experiments({
        "demo": {
            "run": train_fn,
            "local_dir": "/lus/scratch/user/ray_results/custom/",
            "config": {
                "lr": 0.01,
                "num_workers": 64,
            },
        },
    })
```



RL Methods and Results

RL Method – Actor/Critic

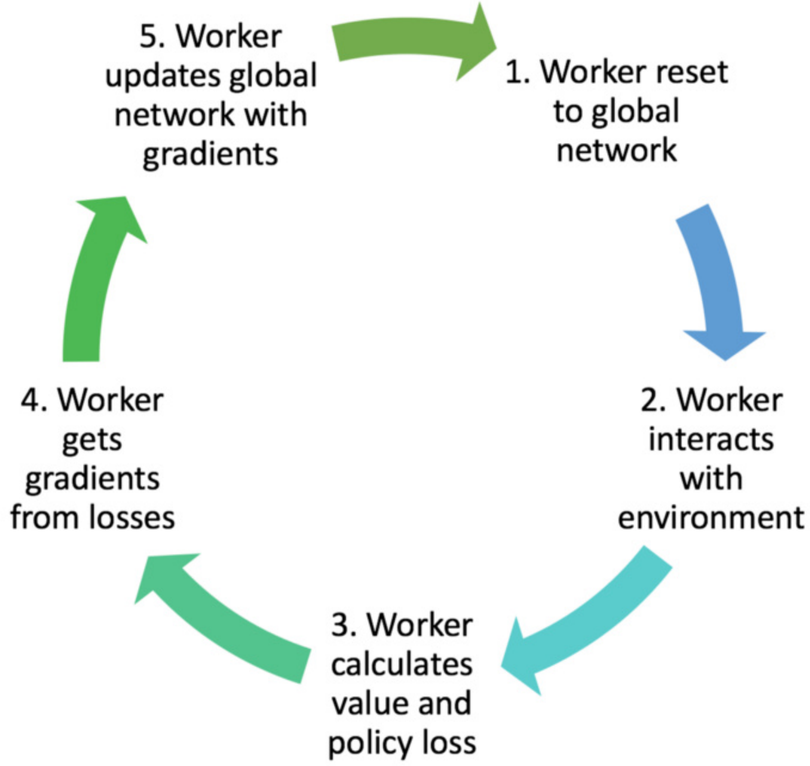
Actor

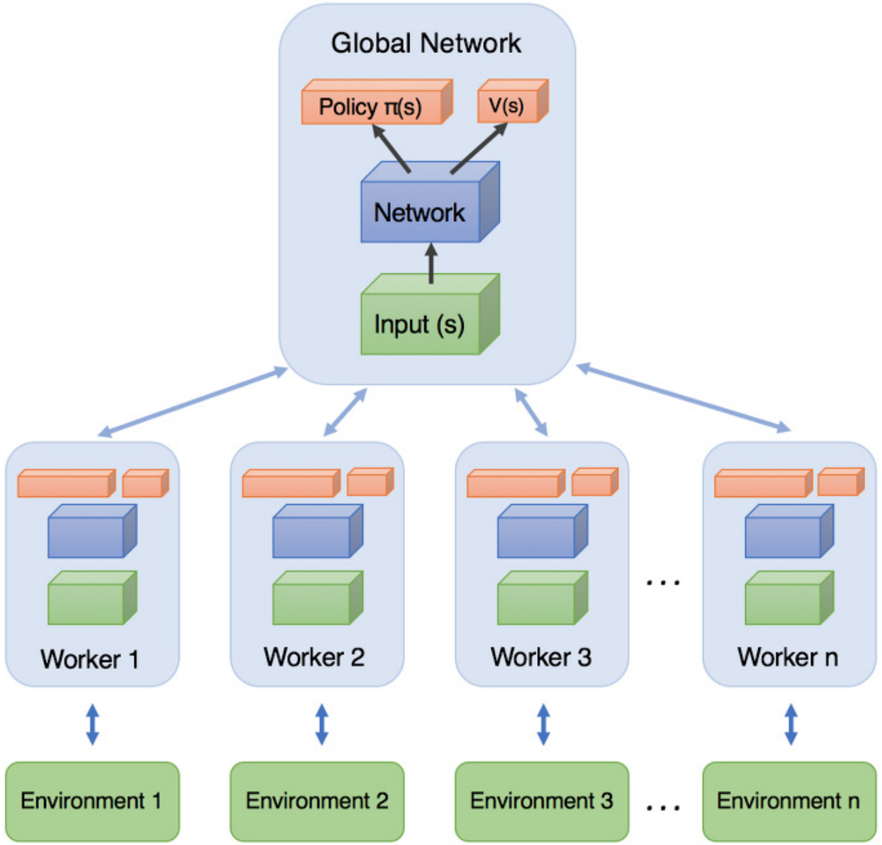
Critic

https://www.cleaninginstitute.org/sites/default/files/assets/1/Photos/700x700/child-coloring-wall.jpg

Distributed RL Method – A3C

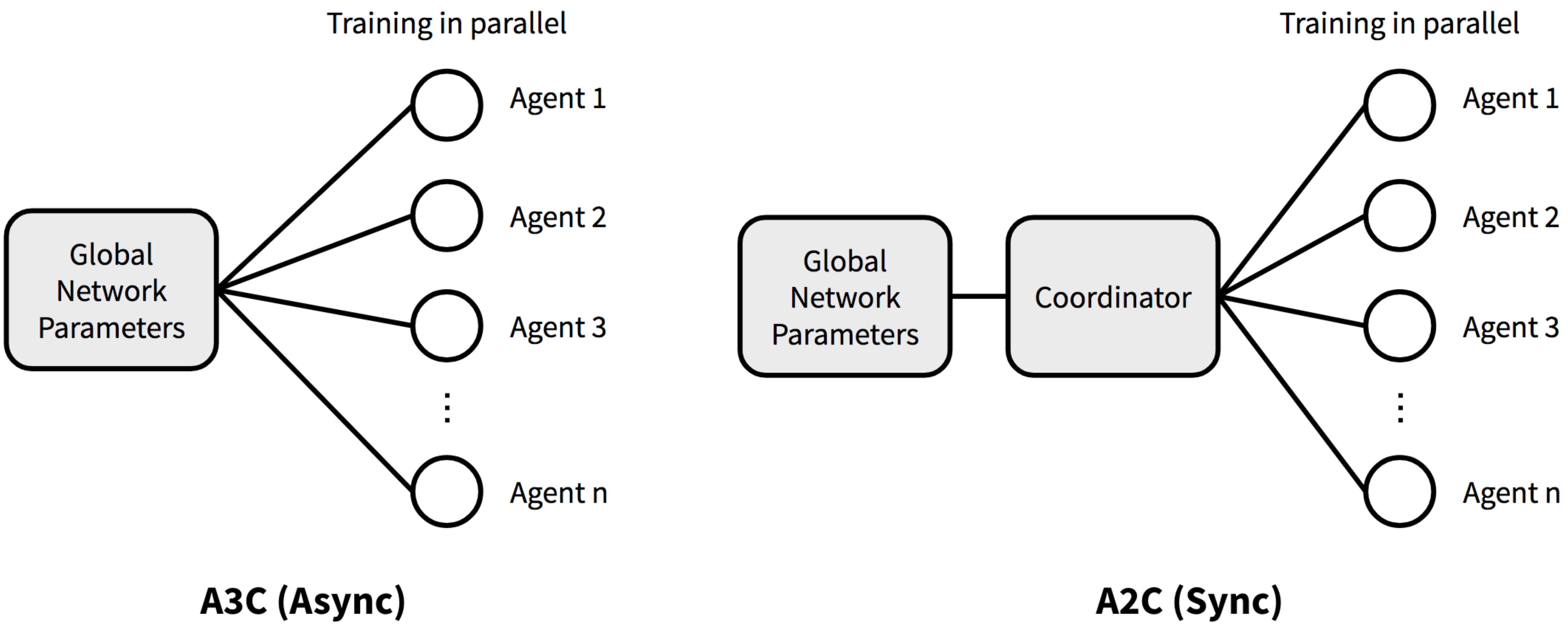
Asynchronous Advantage Actor Critic



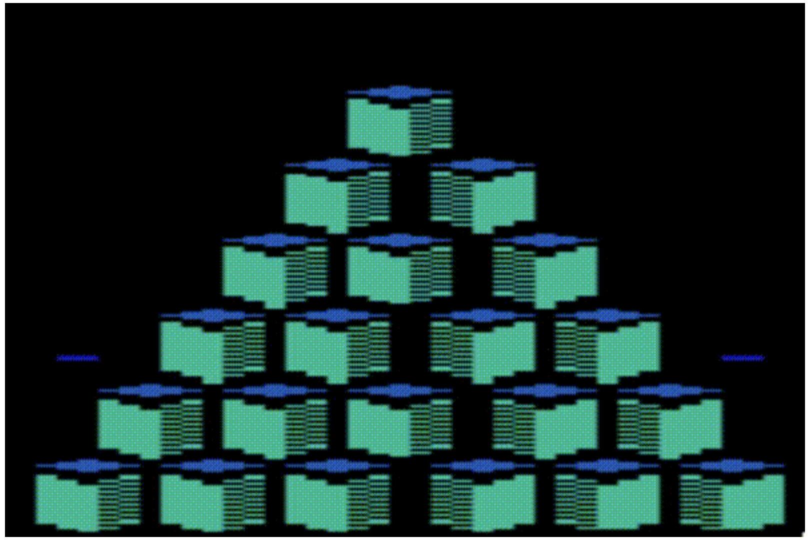


https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

Distributed RL Method – A3C vs A2C



https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#a2c



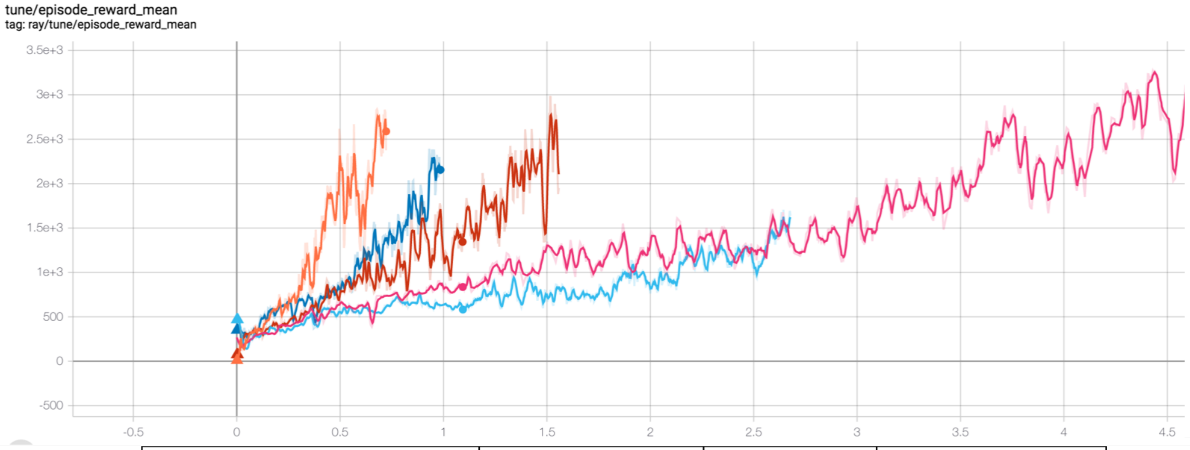
Qbert_16W_PPO

Breakout_7W_PPO

SpaceInvader_16W_A2C

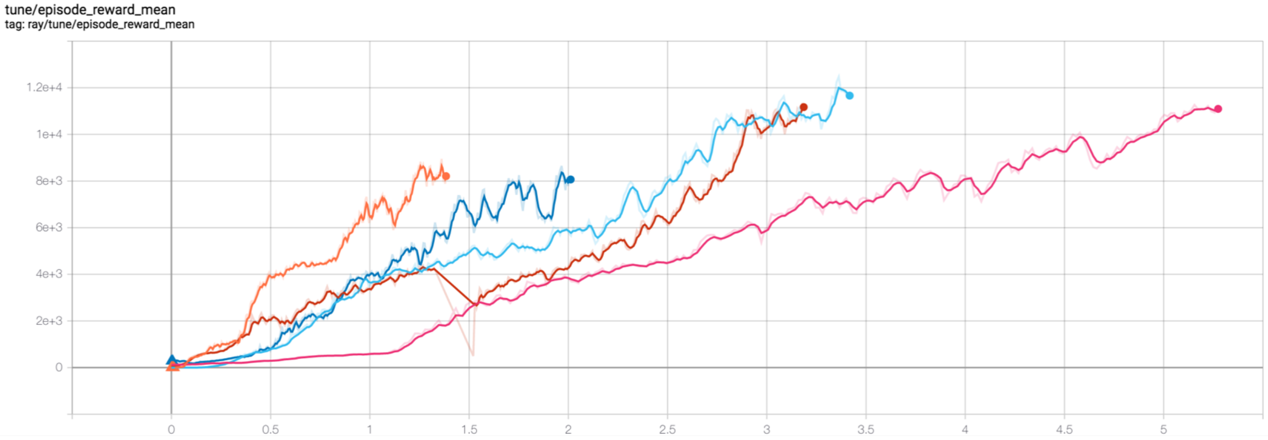
BeamRider_7W_A2C

Multi-Node Scaling



Qbert	Reward Mean	Timestep	Walltime
A2C_16w_1gpu_20cpu	2589	10M	43m 21s
A2C_1w_1gpu_36cpu	3561	10M	4h 50m 8s
A2C_2w_1gpu_36cpu	1621	10M	2h 40m 39s
A2C_4w_1gpu_36cpu	2106	10M	1h 33m 25s
A2C_8w_1gpu_36cpu	2155	10M	59m 6s

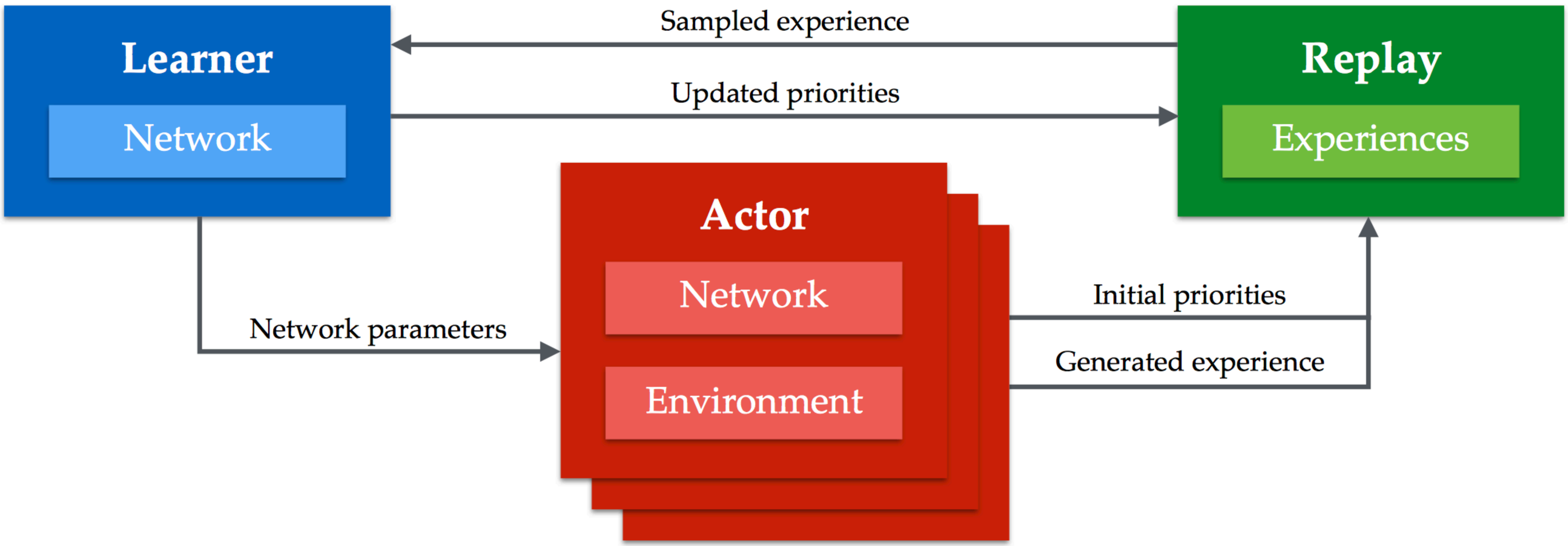
Time in hrs



Qbert	Reward Mean	Timestep	Walltime
PPO_16w_1gpu_20cpu	8211	10M	1h 23m 1s
PPO_1w_1gpu_36cpu	1.1095e+4	10M	5h 16m 27s
PPO_2w_1gpu_36cpu	1.1662e+4	10M	3h 25m 3s
PPO_4w_1gpu_36cpu	1.117e+4	10M	3h 11m 9s
PPO_8w_1gpu_36cpu	8068	10M	2h 0m 41s

Time in hrs

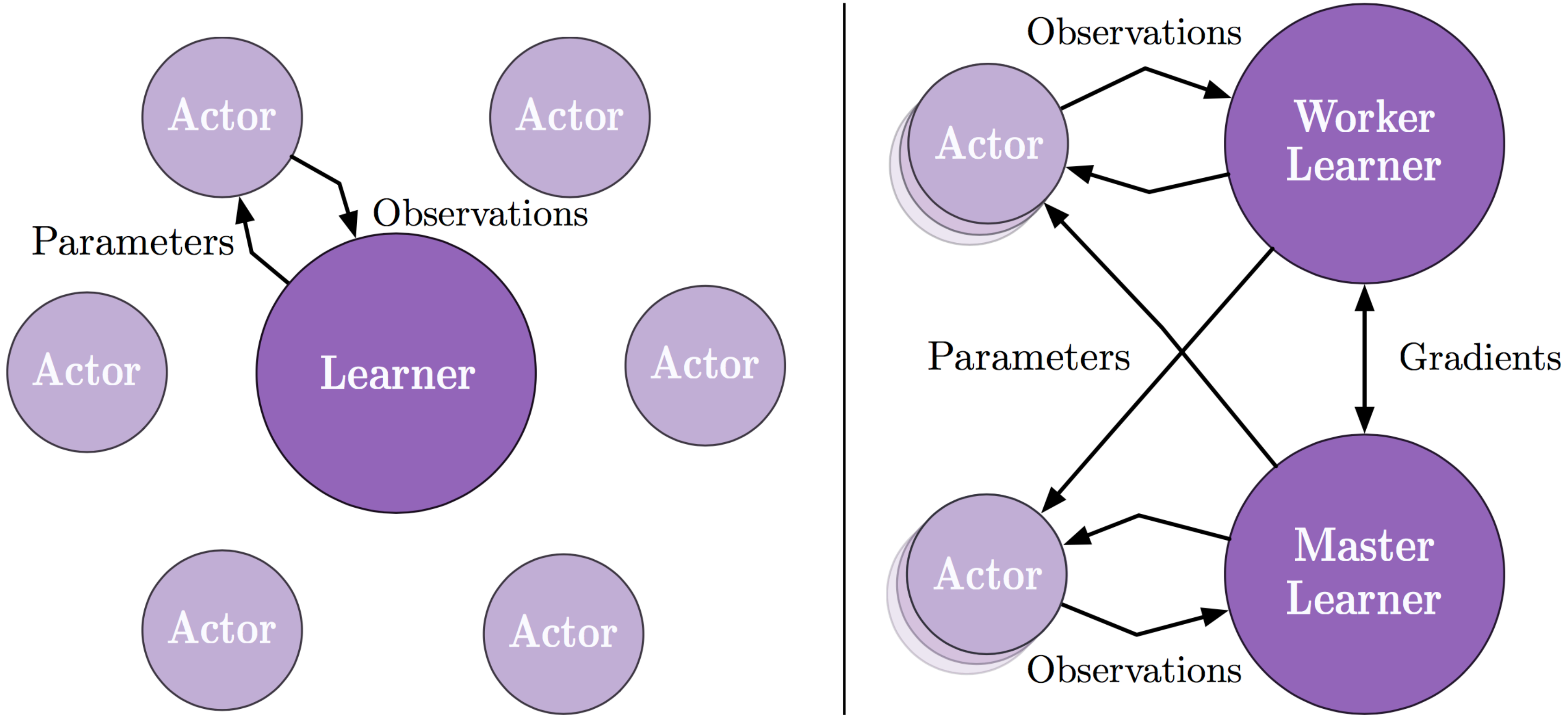
Distributed RL Method – Ape-X



https://openreview.net/pdf?id=H1Dy---0Z

Distributed RL Method - IMPALA

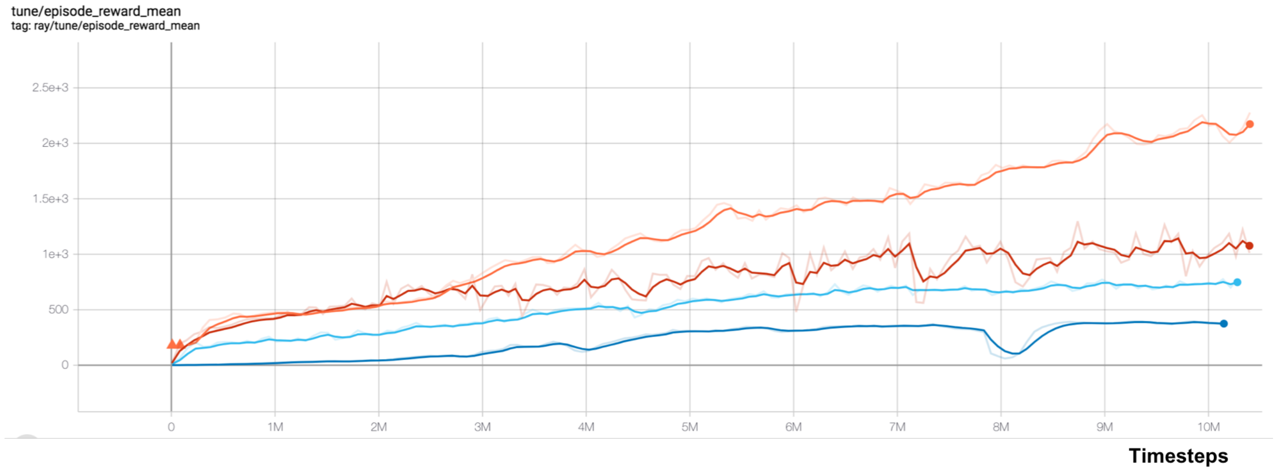
Importance Weighted Actor Learner Architecture



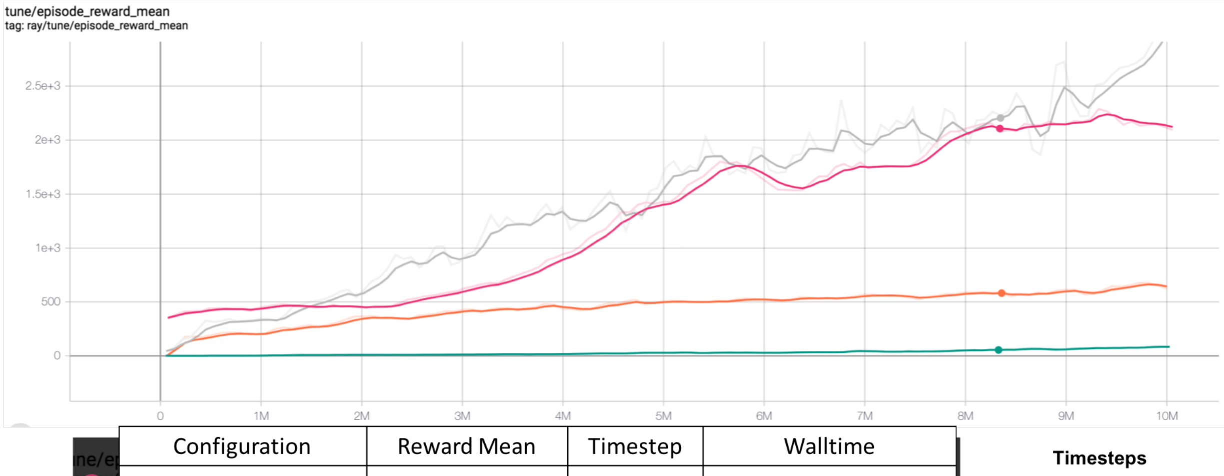
Observations are trajectories of experience (sequence of states, actions, and rewards) not gradients.

https://arxiv.org/pdf/1802.01561.pdf

IMPALA and Ape-X



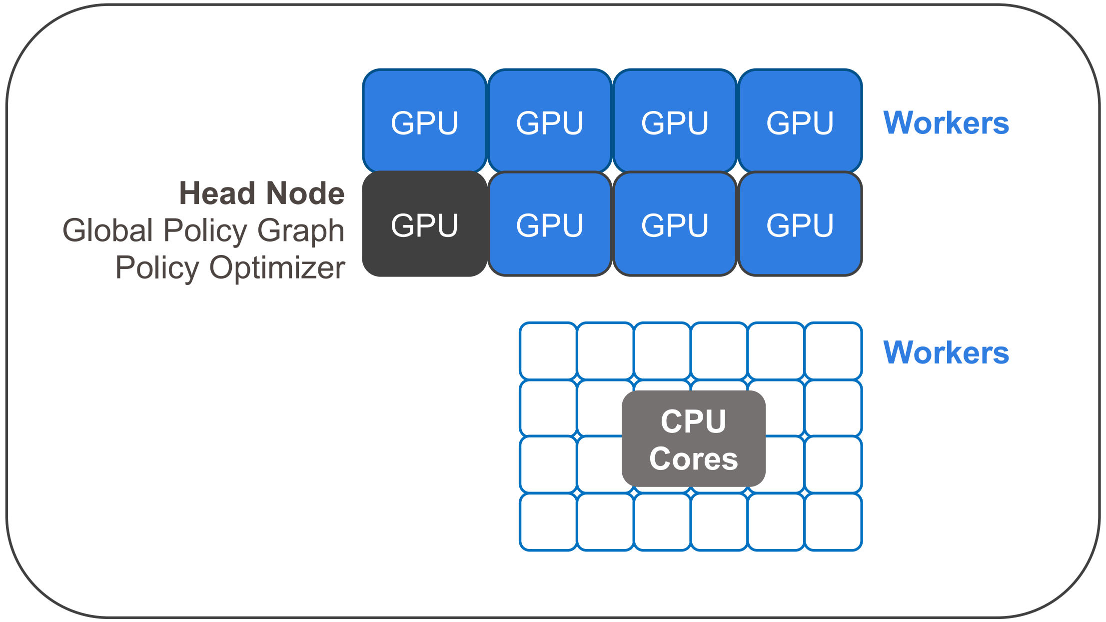
Configuration	Reward Mean	Timestep	Walltime
IMPALA_Beamrider	2173	10M	28m 18s
IMPALA_Breakout	374	10M	28m 14s
IMPALA_Qbert	1077	10M	28m 18s
IMPALA_SpaceInvader	747	10M	28m 9s



Configuration	Reward Mean	Timestep	Walltime
Ape-X_Beamrider	2122	10M	1h 2m 44s
Ape-X_Breakout	84.11	10M	1h 4m 17s
Ape-X_Qbert	2989	10M	1h 4m 54s
Ape-X_SpaceInvader	644.9	10M	56m 25s

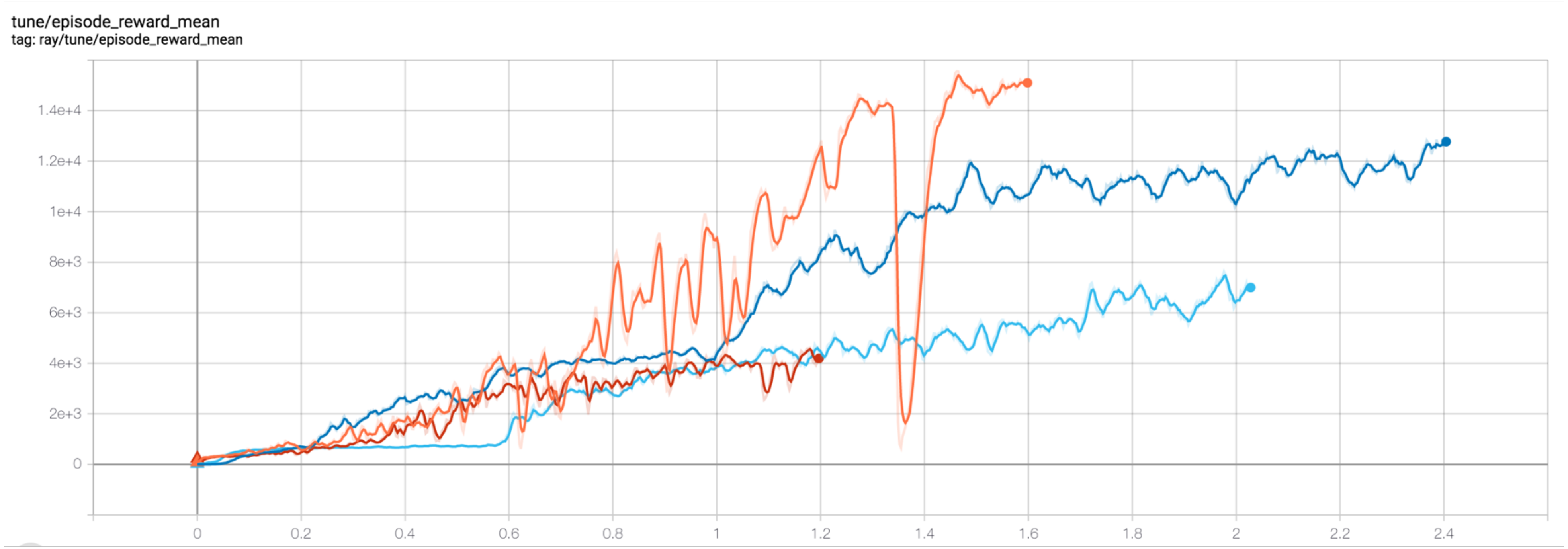
Mapping RL to Cray CS

- 8 GPUs total
- Assign 1 GPU as Head Node
- 7 GPUs as Worker Nodes
- 9 CPU cores per Worker



Single GPU Node on CS Storm

XC vs Dense GPU Node on CS

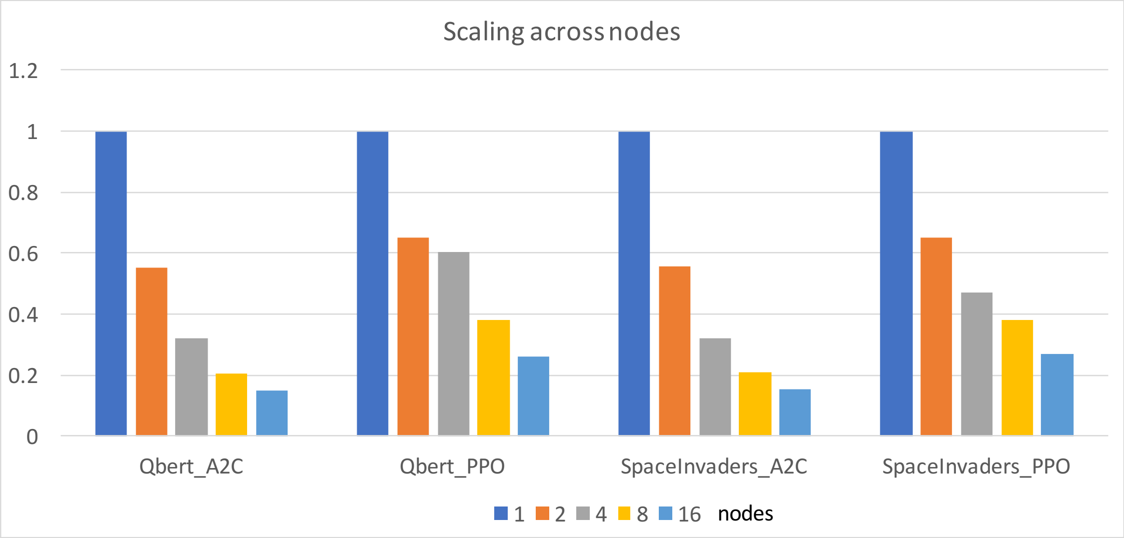


Time in hrs

Configuration	Reward Mean	Timestep	Walltime
Qbert_CS_A2C	1.5101e+4	10M	1h 35m 54s
Qbert_CS_PPO	1.2775e+4	10M	2h 24m 15s
Qbert_XC_A2C	4193	10M	1h 11m 48s
Qbert_XC_Ppo	6998	10M	2h 1m 40s

Key Results

- Explored Reinforcement Learning as a HPC workload
- Deploy a UC Berkeley's RISElab's Ray cluster on XC system
- Trained state-of-art RL agents
- UC Berkeley's RLlib using Ray's distributed execution
- IMPALA, Ape-X, PPO, A2C/A3C on Atari games (from gym library)
- Variety of resource configurations
- Scaled training on multi-node and single-node XC with mixed CPU and GPU



What's Next?

- Explore Optimization of Ray Libraries
- Identify problems sets that map well to Distributed RL
- Comparative studies against other published results
- Explore architecture needs for computational node support and future network requirements

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?

brobbins@cray.com

cray.com

@cray_inc

linkedin.com/company/cray-inc-