Microsoft

人工智能系统 System for AI

强化学习系统
System for Reinforcement Learning

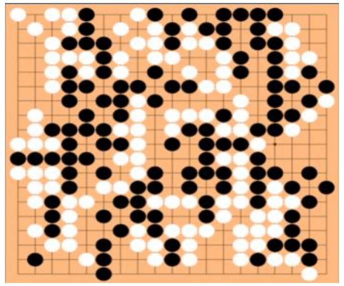
薛卉

微软亚洲研究院

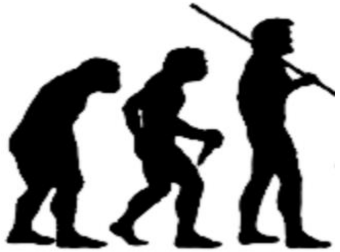
Microsoft

真实世界的问题: 在动态变化的状况下学习如何做出正确的序列选择

游戏

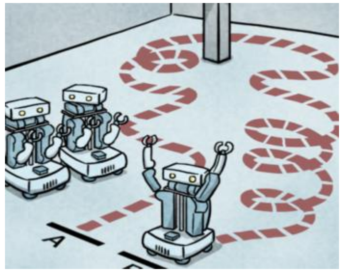


物种的进化



自动驾驶

路径规划



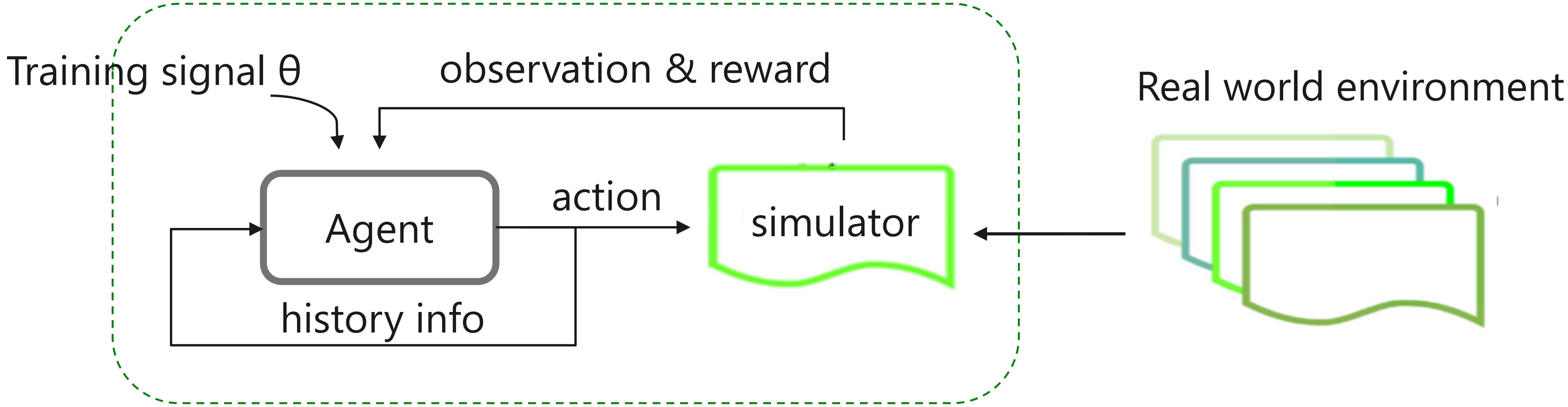
股票交易

电商推荐

…

Microsoft

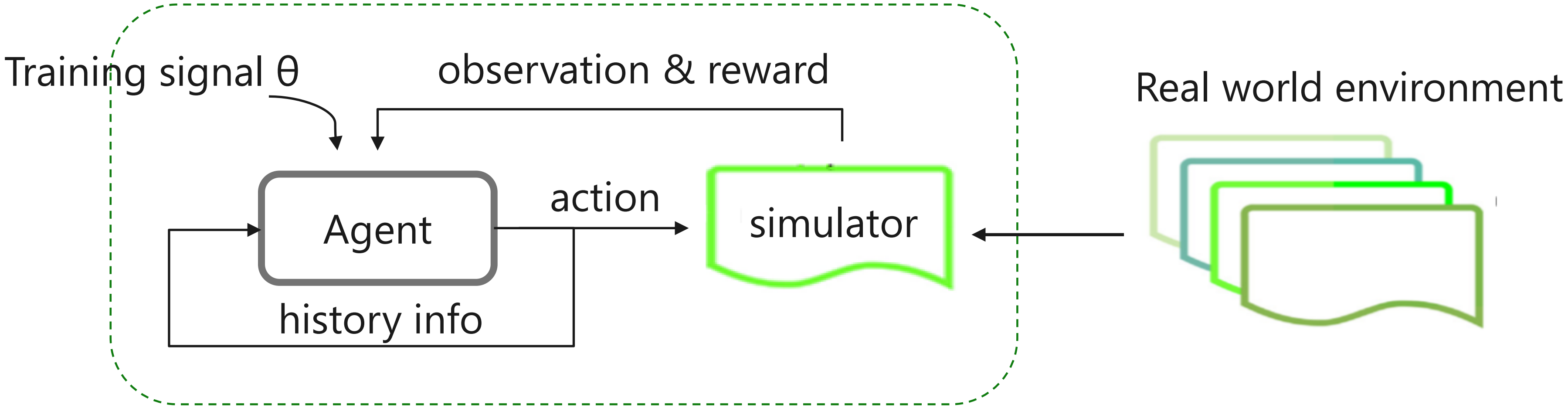
强化学习



- Each time step t
  - *Agent takes an* ***action*** a_t
  - *World updates given* ***action*** *at , emits* ***observation*** o_t *and* ***reward*** r_t
  - *Agent receives* ***observation*** o_t *and* ***reward*** r_t

- *Explore the world (**explore**)*
- *Use experience to guide future decisions (**exploit**)*

强化学习



- ***History*** $h_t = (a_1, o_1, r_1, \ldots, a_t, o_t, r_t)$
- ***Agent*** *chooses action based on history*
- ***State*** *is information assumed to determine what happens next*
  - *Function of history* $s_t = (h_t)$
  - *State* s_t *is* ***Markov*** *if and only if* $p(s_{t+1} \mid s_t, a_t) = p(s_{t+1} \mid h_t, a_t)$

Microsoft

强化学习

- **Goal** select actions to maximize total expected future reward
  - *balancing immediate & long-term rewards*

- **Policy** π determines how the agent chooses actions
  - *Deterministic policy*

$$\pi(s) = a$$

  - *Stochastic policy*

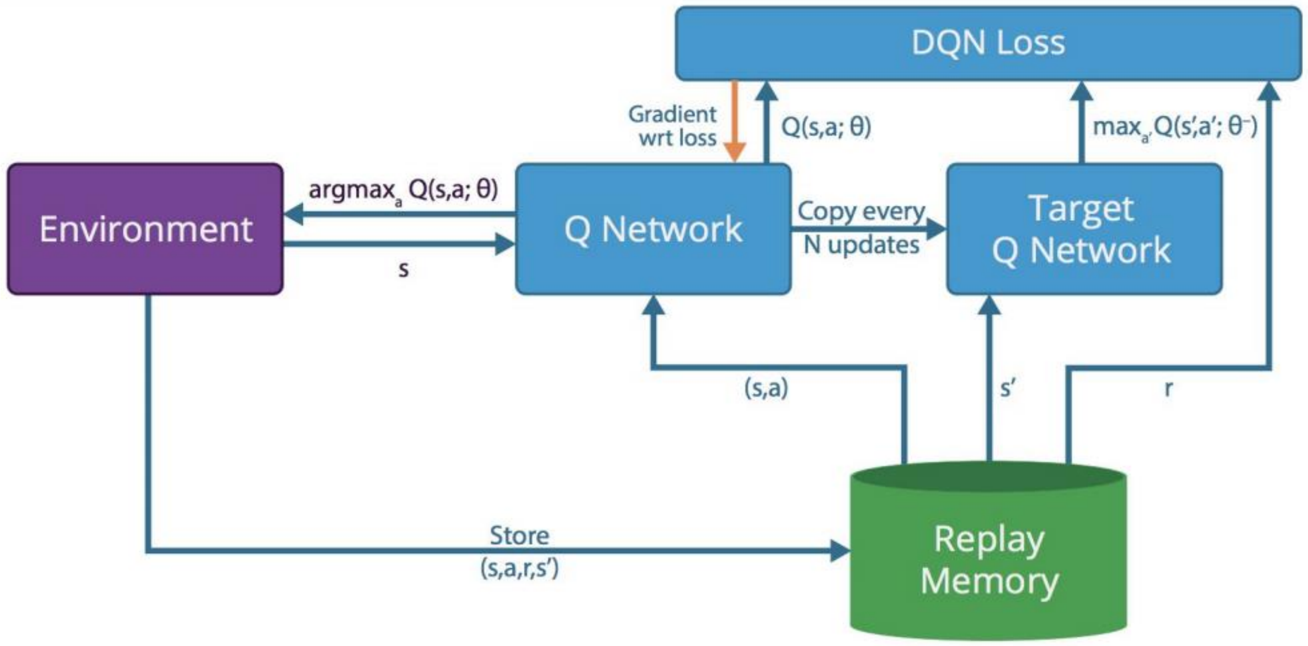
$$\pi(a|s) = Pr(a_t = a|s_t = s)$$

- **Value function** expected discounted sum of future rewards under a policy π

$$V^{\pi}(s_t = s) = \mathbb{E}_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s]$$

Microsoft

同步的单机DQN的例子



同步的单机DQN的例子

Microsoft

initialize env

initialize policy

training loop

Rollout data

Update policy

```python
def cartpole():
    env = gym.make(ENV_NAME)
    score_logger = ScoreLogger(ENV_NAME)
    observation_space = env.observation_space.shape[0]
    action_space = env.action_space.n
    dqn_solver = DQNSolver(observation_space, action_space)
    run = 0
    while True:
        run += 1
        state = env.reset()
        state = np.reshape(state, [1, observation_space])
        step = 0
        while True:
            step += 1
            #env.render()
            action = dqn_solver.act(state)
            state_next, reward, terminal, info = env.step(action)
            reward = reward if not terminal else -reward
            state_next = np.reshape(state_next, [1, observation_space])
            dqn_solver.remember(state, action, reward, state_next, terminal)
            state = state_next
            if terminal:
                print "Run: " + str(run) + ", exploration: " + str(dqn_solver.exploration_rate) + ", score: " + str(step)
                score_logger.add_score(step, run)
                break
            dqn_solver.experience_replay()
```

Policy model

Policy inference

Policy update

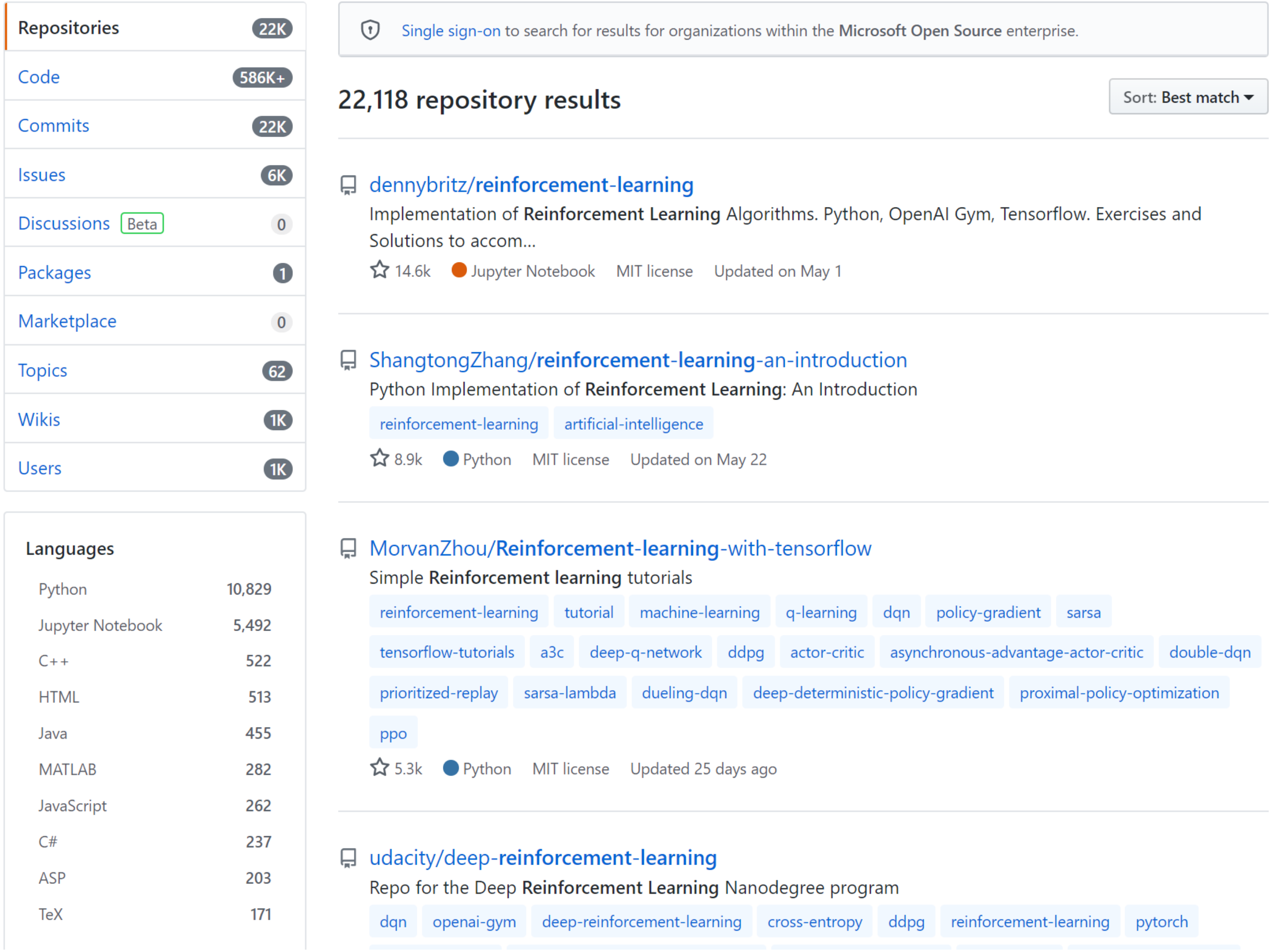
```python
class DQNSolver:

    def __init__(self, observation_space, action_space):
        self.exploration_rate = EXPLORATION_MAX

        self.action_space = action_space
        self.memory = deque(maxlen=MEMORY_SIZE)

        self.model = Sequential()
        self.model.add(Dense(24, input_shape=(observation_space,), activation="relu"))
        self.model.add(Dense(24, activation="relu"))
        self.model.add(Dense(self.action_space, activation="linear"))
        self.model.compile(loss="mse", optimizer=Adam(lr=LEARNING_RATE))

    def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))

    def act(self, state):
        if np.random.rand() < self.exploration_rate:
            return random.randrange(self.action_space)
        q_values = self.model.predict(state)
        return np.argmax(q_values[0])

    def experience_replay(self):
        if len(self.memory) < BATCH_SIZE:
            return
        batch = random.sample(self.memory, BATCH_SIZE)
        for state, action, reward, state_next, terminal in batch:
            q_update = reward
            if not terminal:
                q_update = (reward + GAMMA * np.amax(self.model.predict(state_next)[0]))
            q_values = self.model.predict(state)
            q_values[0][action] = q_update
            self.model.fit(state, q_values, verbose=0)
        self.exploration_rate *= EXPLORATION_DECAY
        self.exploration_rate = max(EXPLORATION_MIN, self.exploration_rate)
```

Microsoft

强化学习和传统的机器学习有什么差别?

强化学习系统面临的挑战和机器学习系统相比，有什么不同?

大量难以复用的强化学习代码库




为什么不能复用这些存在的代码库呢?

Microsoft

算法上微小差别可能会极大地影响结果

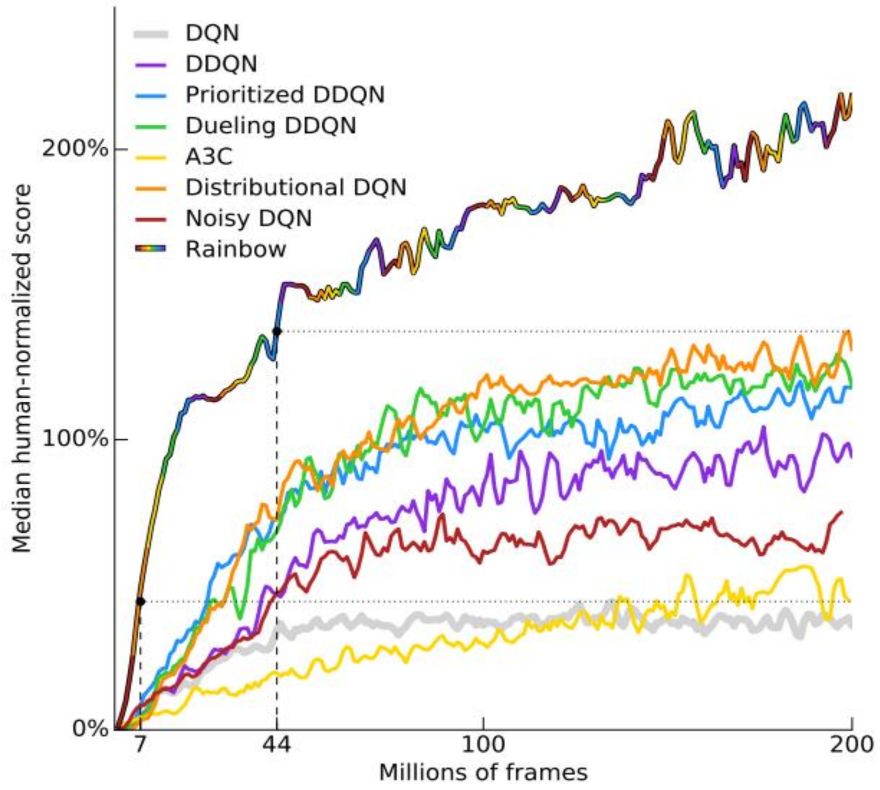


Figure 1 tricks in DQN will performs different performance from rainbow paper.

Hessel, Matteo, et al. "Rainbow: Combining improvements in deep reinforcement learning."

算法上微小差别可能会极大地影响结果

——给PPO带来真正的性能上提升以及将policy约束在trust region内的效果，都不是通过PPO论文中提出的对新的policy和原policy的比值进行裁切（clip）带来的，而是通过code-level的一些技巧带来的。

Engstrom, Logan, et al. "Implementation matters in deep policy gradients: A case study on PPO and TRPO."

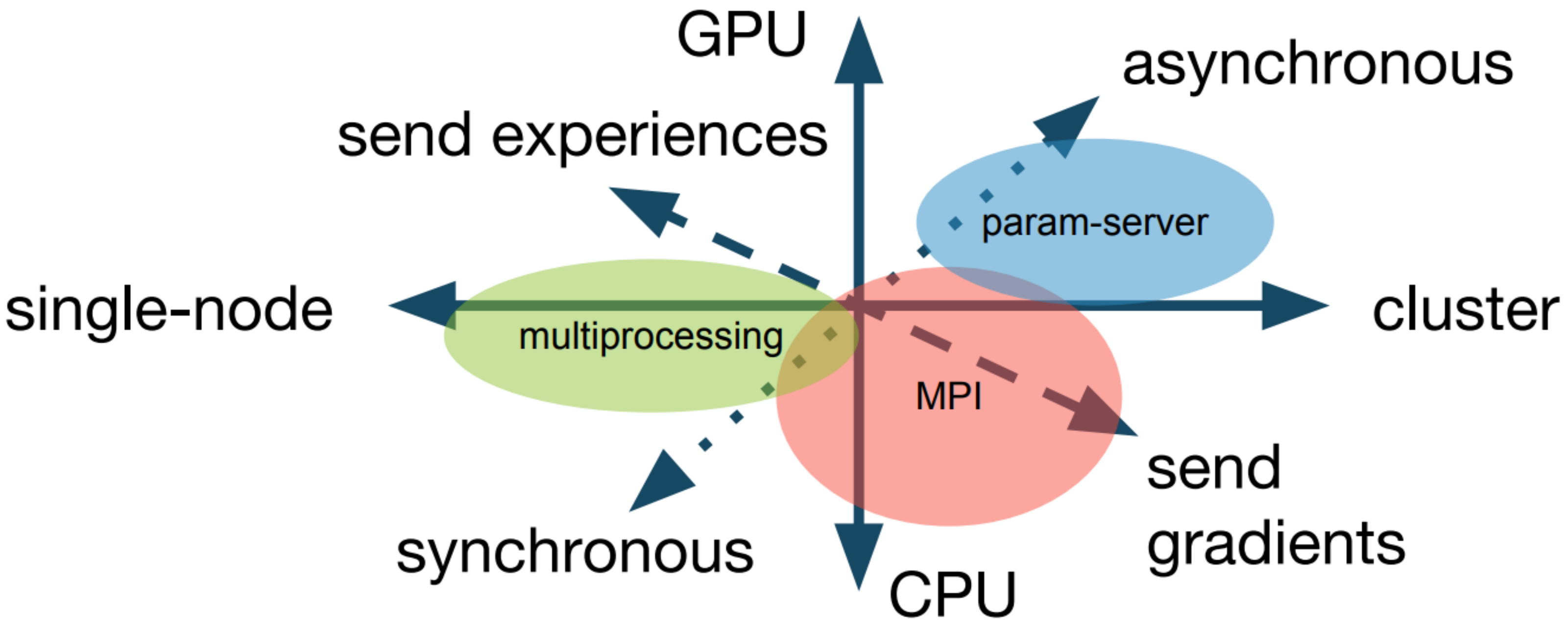
不同的强化学习算法结构差异很大

Algorithm Family	Policy Evaluation	Replay Buffer	Gradient-Based Optimizer	Other Distributed Components
DQNs	X	X	X	
Policy Gradient	X		X	
Off-policy PG	X	X	X	
Model-Based/Hybrid	X		X	Model-Based Planning
Multi-Agent	X	X	X	
Evolutionary Methods	X			Derivative-Free Optimization
AlphaGo	X	X	X	MCTS, Derivative-Free Optimization

Liang, Eric, et al. "Ray rllib: A composable and scalable reinforcement learning library."

Microsoft

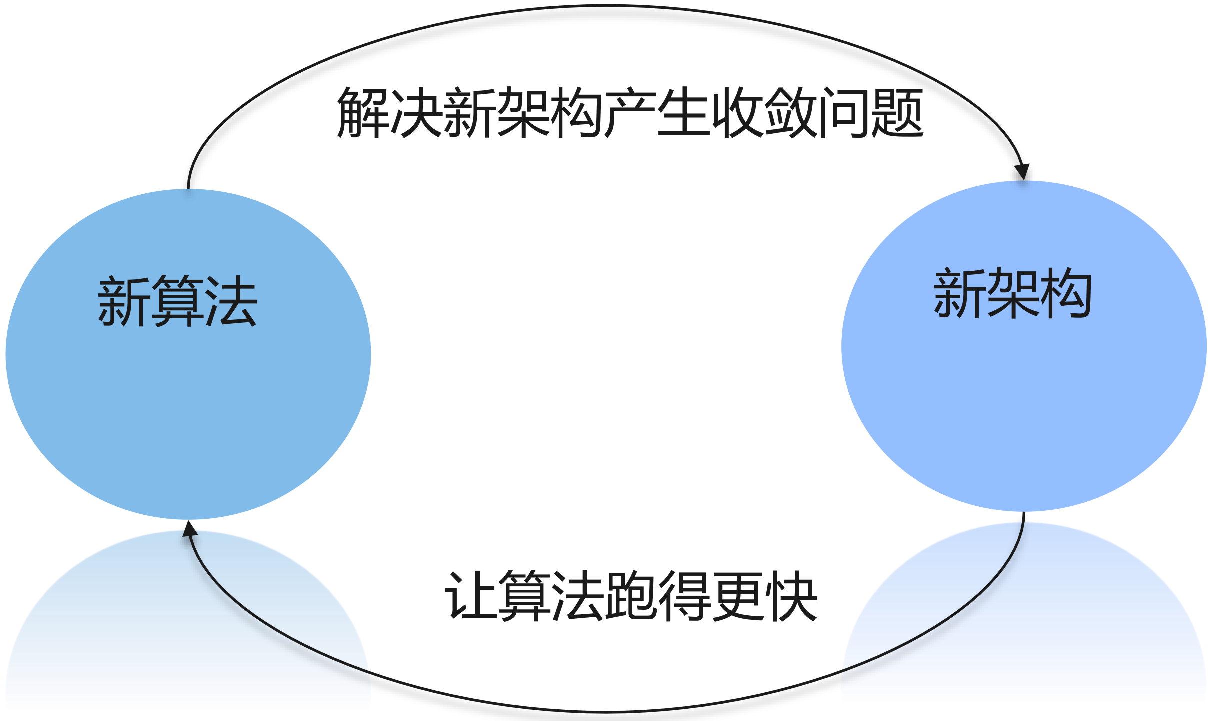
强化学习的执行策略多种多样



Liang, Eric, et al. "Ray rllib: A composable and scalable reinforcement learning library."

Microsoft

分布式强化学习算法和分布式架构互相影响



强化学习算法和分布式架构互相影响

2013 — 2015 — 2016 — 2018 — 2018 — ?

DQN

Playing Atari with Deep Reinforcement Learning (Mnih 2013)

GORILA

Massively Parallel Methods for Deep Reinforcement Learning (Nair 2015)

A3C

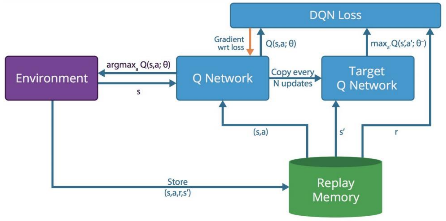
Asynchronous Methods for Deep Reinforcement Learning (Mnih 2016)

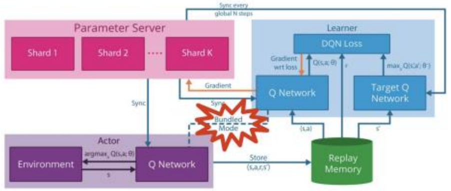
Ape-X

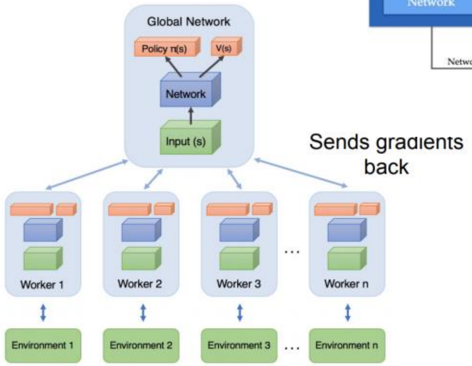
Distributed Prioritized Experience Replay (Horgan 2018)

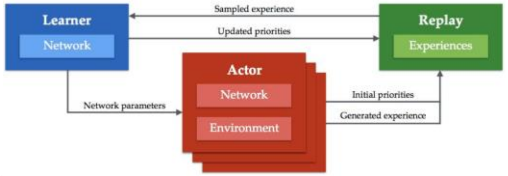
IMPALA

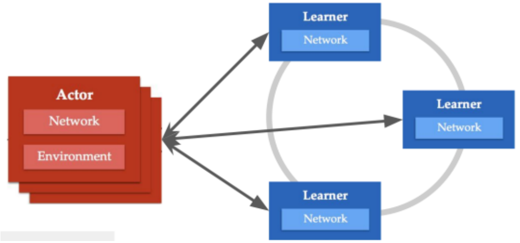
IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures (Espeholt 2018)











为什么不能复用Github上存在的代码库呢?

- **RL算法复现比较困难**
  - e.g., trick, random seed, parameters...
- **不同的RL算法结构存在差异**
  - e.g., on-policy vs off policy...
- **分布式RL算法的执行策略多种多样**
  - e.g., async vs sync, GPU vs TPU, single node vs cluster
- **分布式RL算法和架构互相影响和变化**
  - e.g., Ape-X vs IMPALA

Github上大部分的Repo都只针对特定的算法和架构模式，难以满足RL通用框架的需求。

难以复用的强化学习代码

可扩展性的强化学习框架

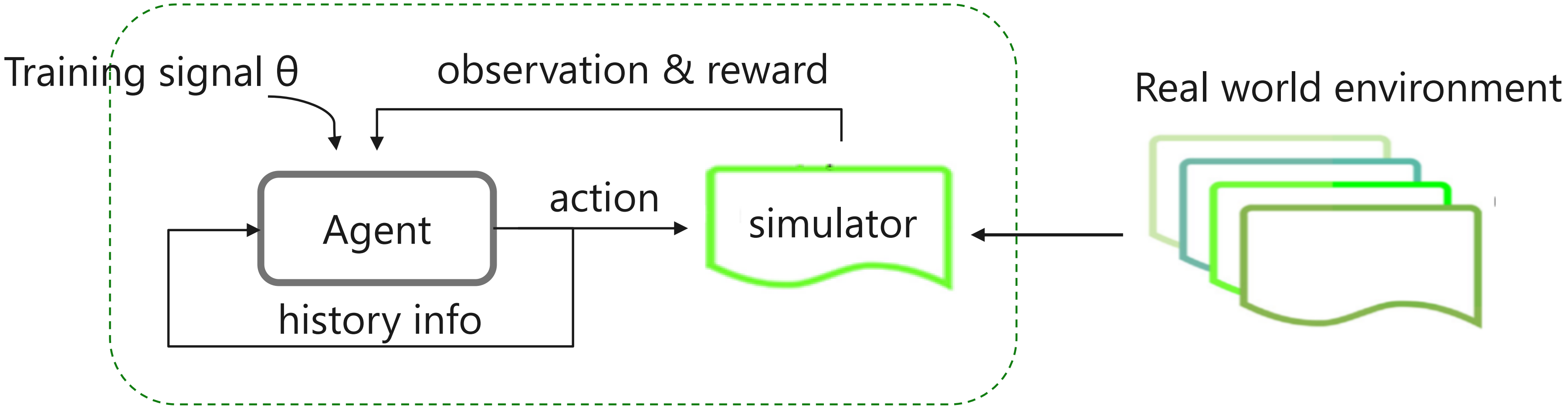
- 用户友好且通用的RL算法的抽象
- 支持复现的各种RL算法
- 支持不同的RL执行策略（e.g.，Sync/Async）
- 支持不同的RL分布式架构

强化学习需要实时采集数据

经典机器学习

强化学习

- 迭代地采集数据和学习
- 自主决定采集什么样的数据



Microsoft

采集数据的效率是收敛的关键

面临的问题

- 与环境交互等待时间较长，资源利用率低
- 分布式rollout数据可行，但分布式代码的开发有成本

新的需求

- 支持复杂环境的并发采集
- 提供易用的分布式的编程模式和API

Apex框架让Actor分布式地rollout data

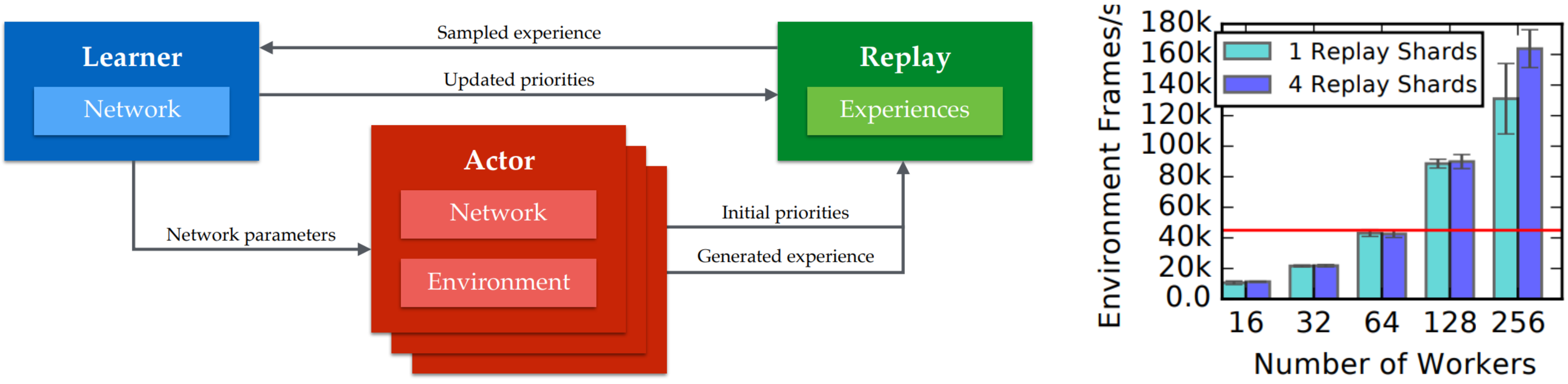


Figure. Apex architecture, multiply actors to rollout data in their own environment.

Horgan, Dan, et al. "Distributed prioritized experience replay."

Microsoft

强化学习训练需要切换context

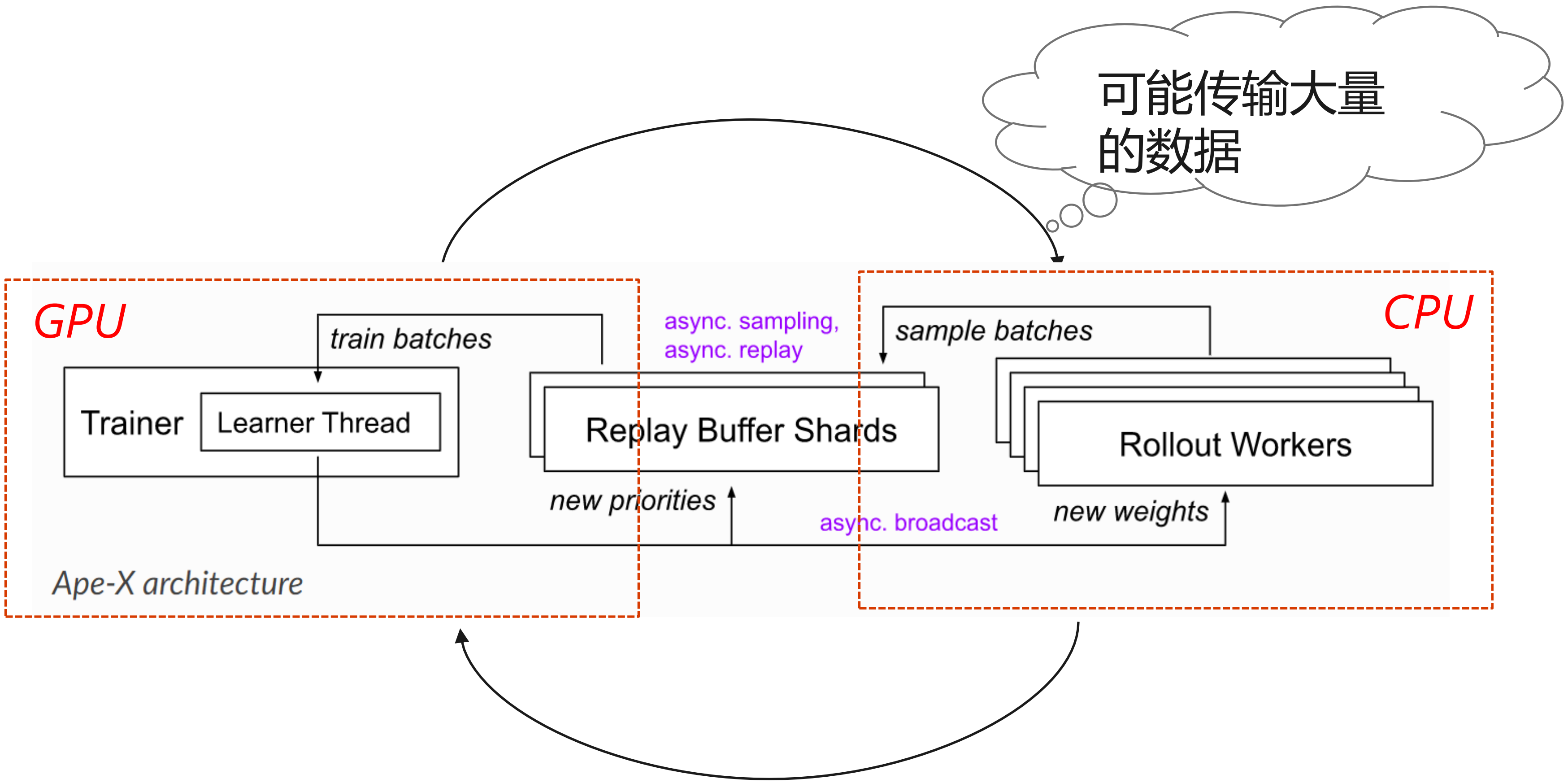
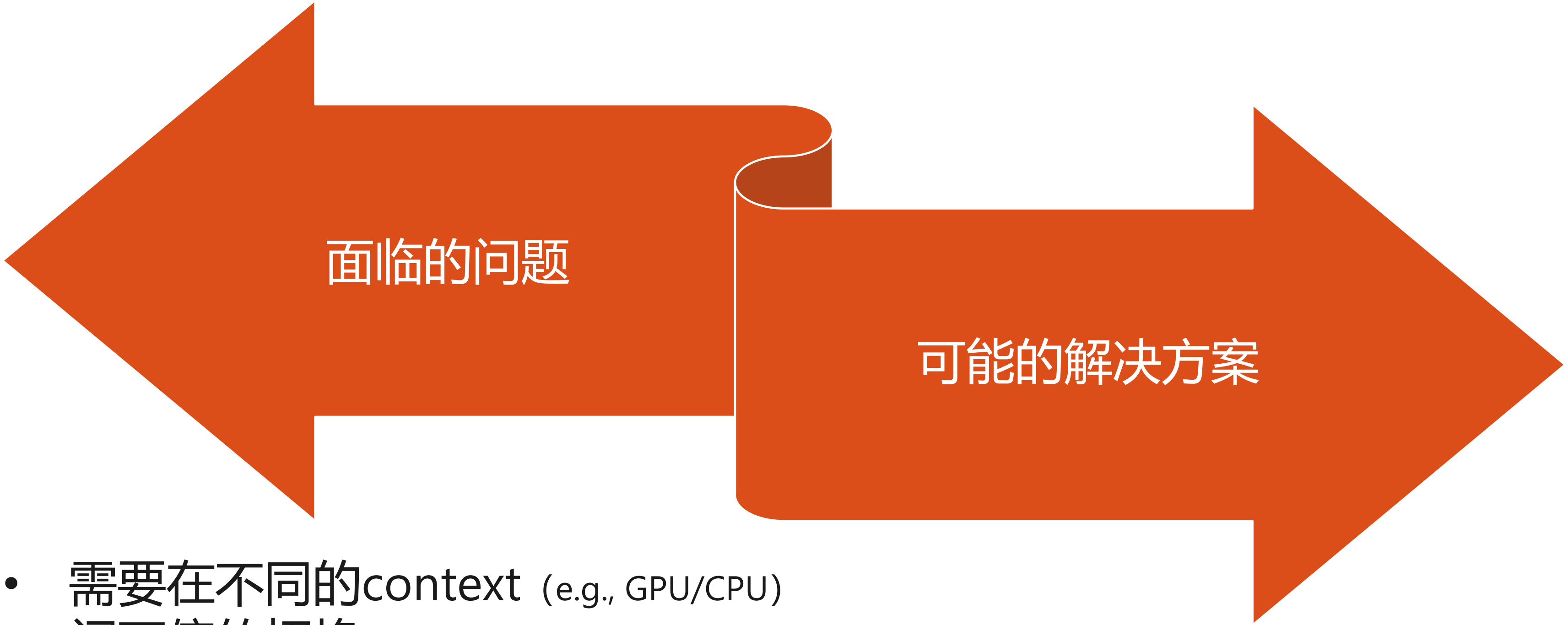


Figure. Context switch in Apex architecture

Microsoft

强化学习训练需要切换context



- 需要在不同的context（e.g., GPU/CPU）间不停的切换
- 同时可能需要传输大量的数据

- 支持高性能的通信框架
- 减少context切换的代价
- 数据的预处理
- 优化数据的传输

当前的强化学习平台

Tensorforce
TENSORFORCE
HUSKARL
Keras-RL
openai gym
PARL
Dopamine
KERAS-RL
ELF
OPENAI GYM
RLLIB
RLlib
BASELINES
COACH
TRFL
TensorLayer
openai gym
baselines
DOPAMINE
TENSORLAYER
Unity
TIANSHOU

当前强化学习平台的分类

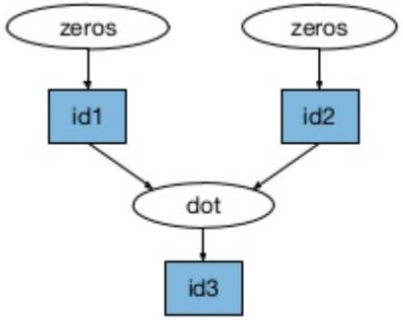
	通用的RL算法	针对Env开发	支持分布式	Star数目	Repo
ACME+Reverb	✓	×	✓	2.1k	https://github.com/deepmind/acme
ELF	×	✓	✓	2k	https://github.com/facebookresearch/ELF
Ray + RLlib	✓	×	✓	16.4k	https://github.com/ray-project/ray
Gym	×	✓	×	24.5k	https://github.com/openai/gym
Baselines	✓	×	×	11.6k	https://github.com/openai/baselines
TorchBeast	×	×	✓	553	https://github.com/facebookresearch/torchbeast
SeedRL	×	×	✓	617	https://github.com/google-research/seed_rl
Tianshuo	✓	×	?	3.2k	https://github.com/thu-ml/tianshou
Keras-RL	✓	×	×	5.1k	https://github.com/keras-rl/keras-rl

案例研究: Ray and RLlib

Ray is a **fast** and **simple** framework for building and running **distributed applications**.

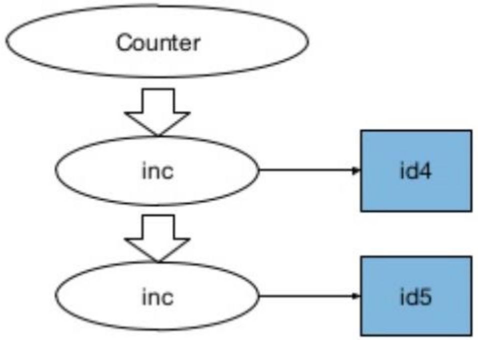
- *Ray provide a task parallel API*

```python
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)

id1 = zeros.remote([5, 5])
id2 = zeros.remote([5, 5])
id3 = dot.remote(id1, id2)
result = ray.get(id3)
```



- *Ray provide an actor API*

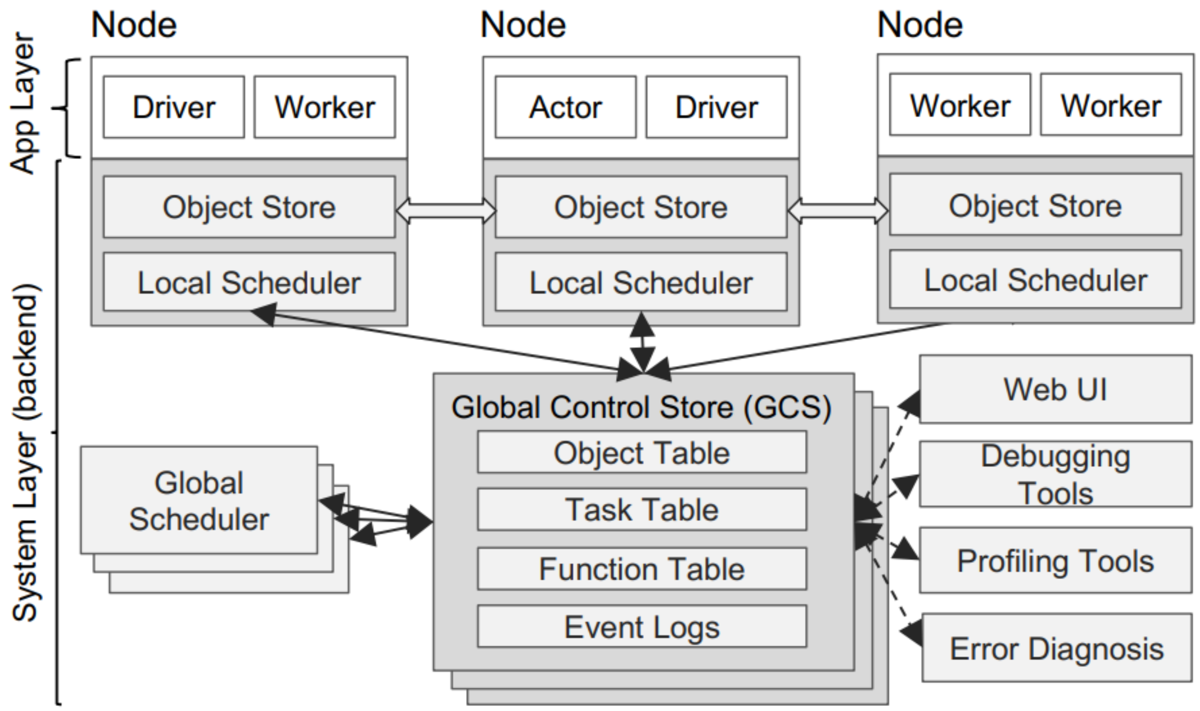
```python
@ray.remote(num_gpus=1)
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.remote()
id4 = c.inc.remote()
id5 = c.inc.remote()
result = ray.get([id4, id5])
```



Github repo: https://github.com/ray-project/ray

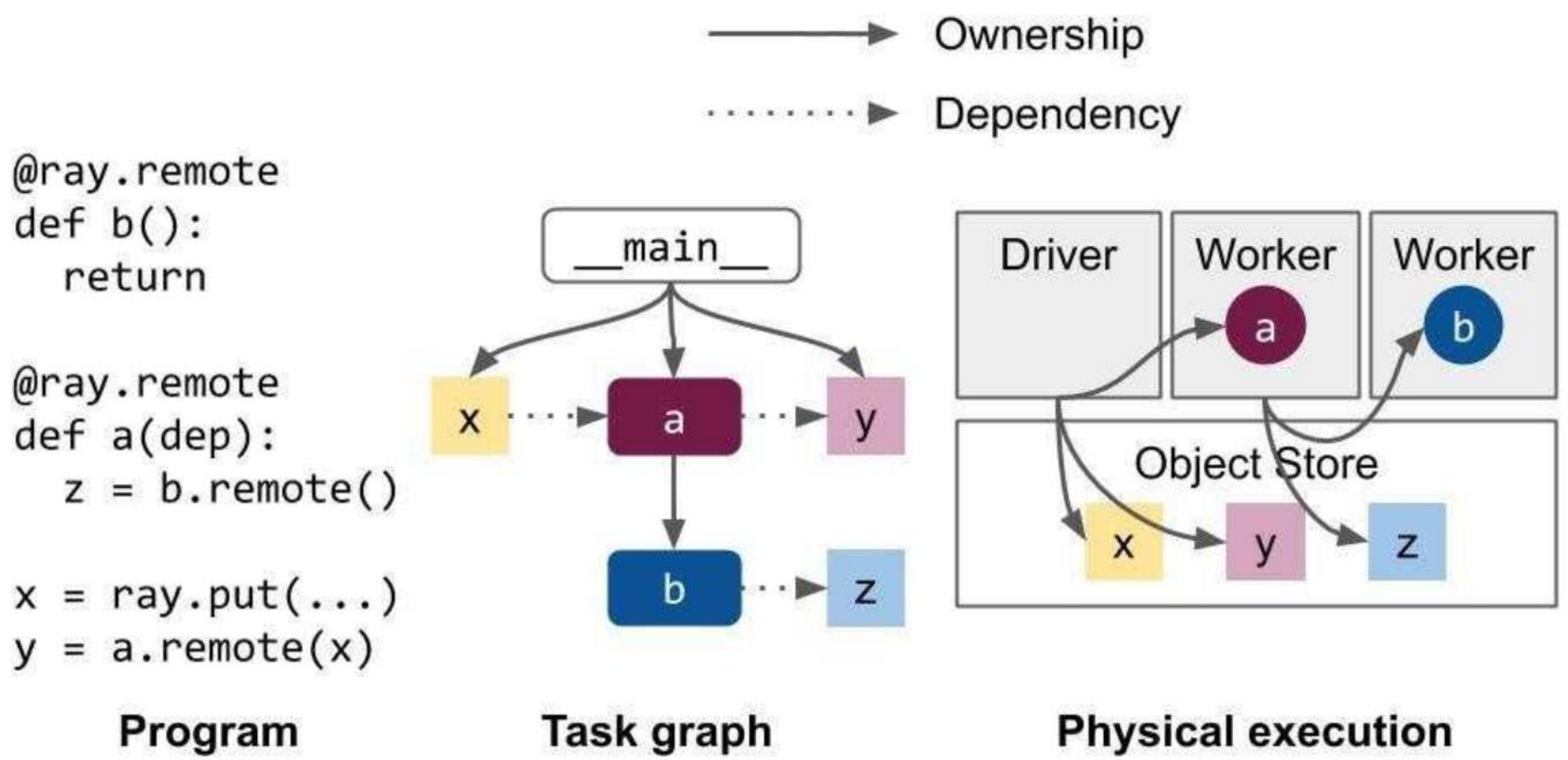
案例研究: Ray and RLlib

Ray is a **fast** and **simple** framework for building and running **distributed applications**.



- App Layer
  - Driver - *A process executing the user program*
  - Worker - *A stateless process that executes remote functions invoked by a driver*
  - Actor - *A stateful process that executes*
- System Layer
  - Distributed object store
    - *In-memory distributed storage to store the inputs/outputs, or stateless computation.*
    - *Implement the object store via shared memory*
    - *Use Apache Arrow as data formats*
  - Distributed scheduler
    - *Submitted first to local scheduler*
    - *Global scheduler considers each node's load and task's constraints to make scheduling decisions*
  - Global Control Store(GCS)
    - *A key-value store with pub-sub functionality*

Microsoft

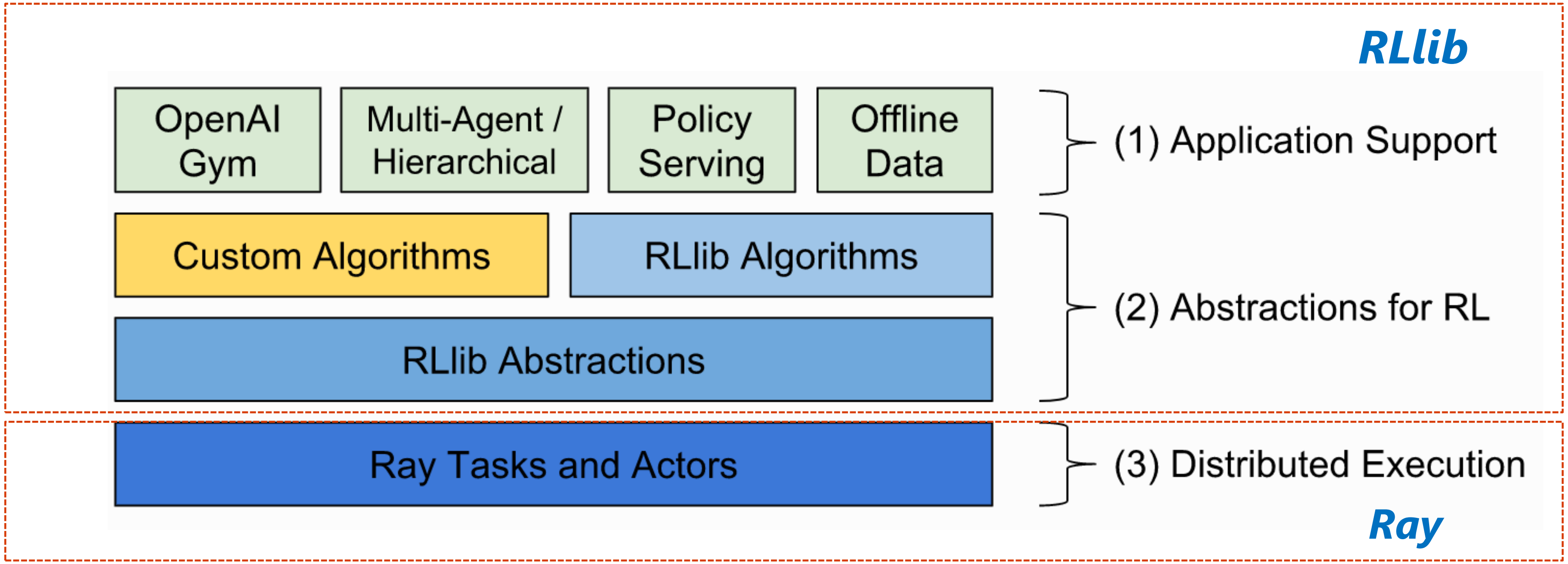


Ownership
Dependency
@ray.remote
def b():
  return
@ray.remote
def a(dep):
  z = b.remote()
x = ray.put(...)
y = a.remote(x)
__main__
x
a
y
b
z
Driver
Worker
Worker
a
b
Object Store
x
y
z
Program
Task graph
Physical execution

Microsoft

案例研究: Ray and RLlib

RLlib is an open-source library for reinforcement learning that offers both **high scalability** and a **unified API** for a variety of applications.



Github repo: https://github.com/ray-project/ray/tree/master/rllib

Microsoft

友好的分布式编程接口

Ray's distributed scheduler is a natural fit for the hierarchical control model, as nested computation can be implemented in Ray with no central task scheduling bottleneck.

```
if mpi.get_rank() <= m:
    grid = mpi.comm_world.split(0)
else:
    eval = mpi.comm_world.split(
        mpi.get_rank() % n)
...
if mpi.get_rank() == 0:
    grid.scatter(
        generate_hyperparams(), root=0)
    print(grid.gather(root=0))
elif 0 < mpi.get_rank() <= m:
    params = grid.scatter(None, root=0)
    eval.bcast(
        generate_model(params), root=0)
    results = eval.gather(
         result, root=0)
    grid.gather(results, root=0)
elif mpi.get_rank() > m:
    model = eval.bcast(None, root=0)
    result = rollout(model)
    eval.gather(result, root=0)
```

a. Distributed control in MPI

```
@ray.remote
def rollout(model):
    # perform a rollout and
    # return the result

@ray.remote
def evaluate(params):
    model = generate_model(params)
    results = [rollout.remote(model)
       for i in range(n)]
    return results

param_grid = generate_hyperparams()
print(ray.get([evaluate.remote(p)
   for p in param_grid]))
```

b. Hierarchical control in ray.

基于Ray的简单的异步DQN的例子

Trainer

```python
import ray
from collections import deque
import time
import threading

from dummy import DQN, Repl

@ray.remote
class Trainer:
    def __init__(self):
        self.steps = 0
        self.thread = None
        self.dqn = DQN()
        self.buffer = ReplayBuffer()
        self.worker = None
        self.checkpoint_interval = 5

    def _run(self):
        for _ in range(10000):
            self.steps += 1
            batch = self.buffer.sample()
            self.dqn.train(batch)
            if self.steps % self.checkpoint_interval:
                weight = self.dqn.dump_weights()
                if self.worker is not None:
                    self.worker.update_weights.remote(weight)

    def run(self, worker):
        self.worker = worker
        self.thread = threading.Thread(target=self._run)
        self.thread.start()

    def add_transitions(self, trans):
        for row in trans:
            self.buffer.append(row)
```

Remote decorator for run in remote

Start thread for async training

Actors/Workers

```python
import ray
import threading

from dummy import DQN, Env

BATCH_SIZE = 10

@ray.remote
class Worker:
    def __init__(self):
        self.dqn = DQN()
        self.env = Env()
        self.s0 = self.env.reset()
        self.trainer = None

        self.buffer = []

    def _run(self):
        for _ in range(10000):
            a = self.dqn.act(self.s0)
            s1, r, done, _ = self.env.step(a)

            if done:
                self.s0 = self.env.reset()
            else:
                self.s0 = s1
            self.buffer.append((self.s0, a, r, s1, done))

            if len(self.buffer) == BATCH_SIZE:
                if self.trainer is not None:
                    self.trainer.add_transitions.remote(self.buffer)
                self.buffer = []

    def run(self, trainer):
        self.trainer = trainer
        self.thread = threading.Thread(target=self._run)
        self.thread.start()

    def update_weights(self, weights):
        self.dqn.load_weights(weights)
```

Run script

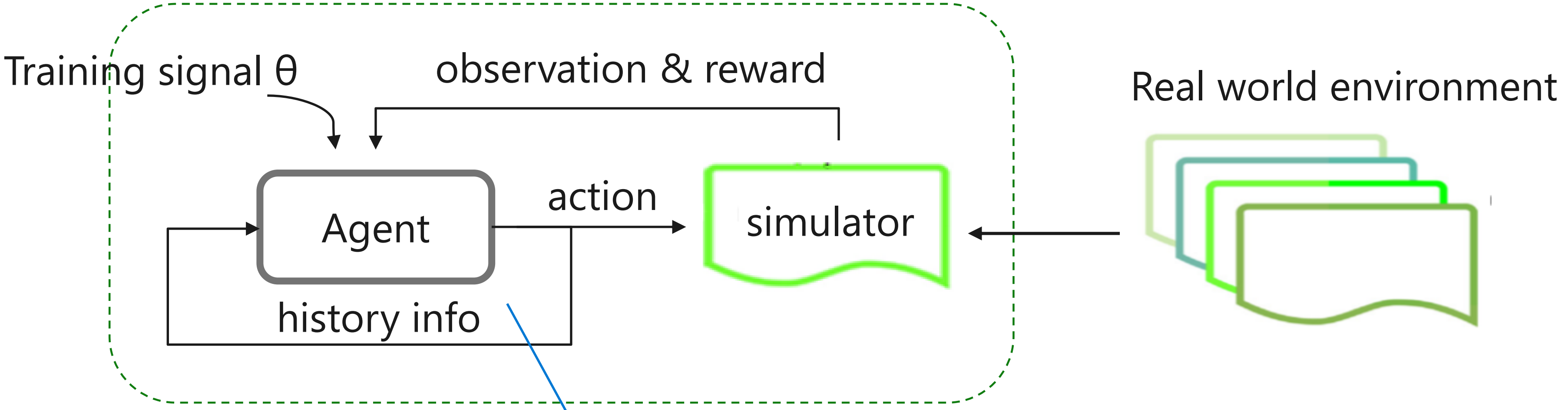
```python
import ray
import time
from trainer import Trainer
from worker import Worke

ray.init()

worker = Worker.remote()
trainer = Trainer.remote()
t1 = worker.run.remote(trainer)
t2 = trainer.run.remote(worker)
ray.get([t1, t2])
time.sleep(100)
ray.shutdown()
```

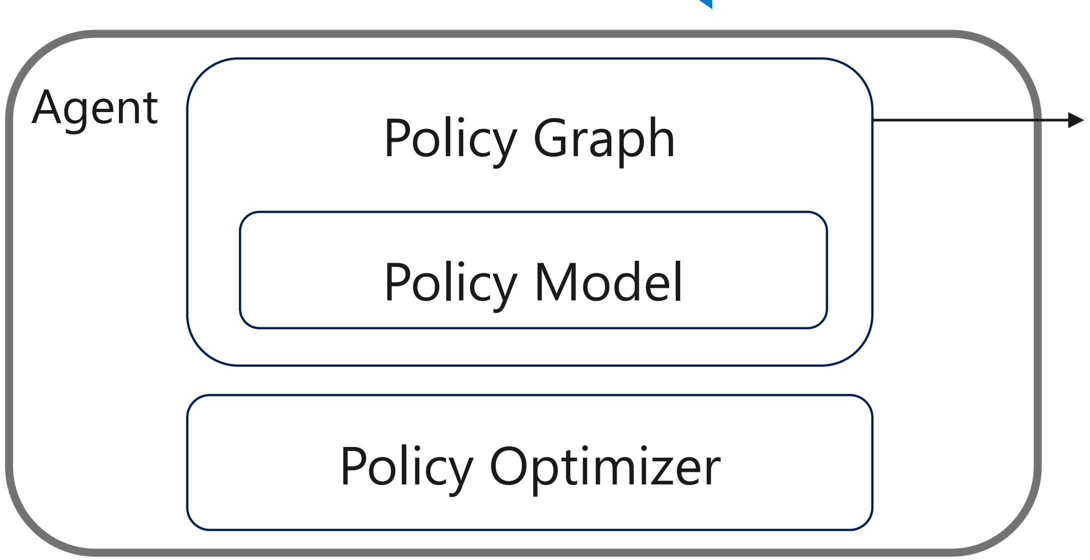
Init ray

Execute the trainer and actor in remote

Microsoft

清晰的模块化的RL接口

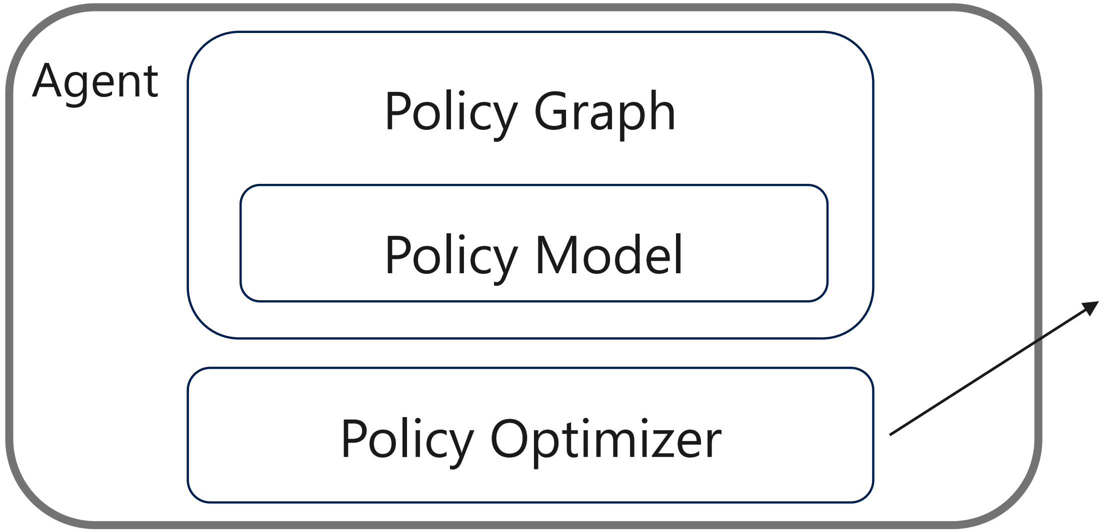





```
abstract class rllib.PolicyGraph:
  def act(self, obs, h): action, h, y*
  def postprocess(self, batch, b*): batch
  def gradients(self, batch): grads
  def get_weights; def set_weights;
  def u*(self, args*)
```

$$\pi_\theta(o_t, h_t) \Rightarrow (a_t, h_{t+1}, y_t^1 \ldots y_t^N)$$

清晰的模块化的RL接口



<u>**The policy optimizer** is responsible for the performance-critical tasks of distributed sampling, parameter updates, and managing replay buffers.</u>

```
grads = [ev.grad(ev.sample())
    for ev in evaluators]
avg_grad = aggregate(grads)
local_graph.apply(avg_grad)
weights = broadcast(
    local_graph.weights())
for ev in evaluators:
    ev.set_weights(weights)
```

(a) Allreduce

```
samples = concat([ev.sample()
    for ev in evaluators])
pin_in_local_gpu_memory(samples)
for _ in range(NUM_SGD_EPOCHS):
   local_g.apply(local_g.grad(samples)
weights = broadcast(local_g.weights())
for ev in evaluators:
    ev.set_weights(weights)
```

(b) Local Multi-GPU

```
grads = [ev.grad(ev.sample())
    for ev in evaluators]
for _ in range(NUM_ASYNC_GRADS):
    grad, ev, grads = wait(grads)
    local_graph.apply(grad)
    ev.set_weights(
        local_graph.get_weights())
    grads.append(ev.grad(ev.sample()))
```

(c) Asynchronous

```
grads = [ev.grad(ev.sample())
    for ev in evaluators]
for _ in range(NUM_ASYNC_GRADS):
    grad, ev, grads = wait(grads)
    for ps, g in split(grad, ps_shards):
        ps.push(g)
    ev.set_weights(concat(
        [ps.pull() for ps in ps_shards])
    grads.append(ev.grad(ev.sample()))
```

(d) Sharded Param-server

Figure. Pseudocode for four RLlib policy optimizer step methods. Each step() operates over a local policy graph and array of remote evaluator replicas.

多种多样的可复现的强化学习算法

Microsoft

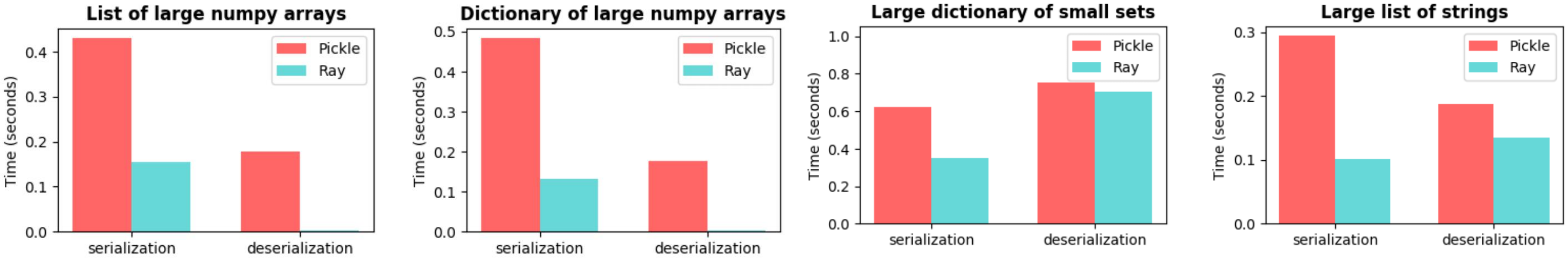
- High throughput architectures
  - Distributed Prioritized Experience Replay(Ape-X-DQN, Ape-X-DDPG)
  - Importance Weighted Actor-Learner Architecture(IMPALA)
- Gradient-based
  - Advantage Actor-Critic(A2C, A3C)
  - Deep Deterministic Policy Gradients(DDPG, TD3)
  - Deep Q Networks(DQN, Rainbow)
  - Policy Gradients
  - Proximal Policy Optimization(PPO, APPO)
  - Soft Actor-Critic(SAC)
  - Single player AlphaZero
- Derivative-free
  - Augment Random Search(ARS)
  - Evolution Strategies
- Multi-agent
  - Monotonic Value Function Factorization(QMIX, VDN, IQN)
  - MADDPG

```
tune.run(
    "DQN",
    stop={"episode_reward_mean": 100},
    config={
        "env": "CartPole-v0",
        "num_gpus": 0,
        "num_workers": 1,
        "lr": tune.grid_search([0.01, 0.001, 0.0001]),
        "monitor": False,
    },
)
```

快速的序列化和反序列化

<u>Serialization and deserialization are **bottlenecks in parallel and distributed computing**, especially in machine learning applications with large objects and large quantities of data.</u>

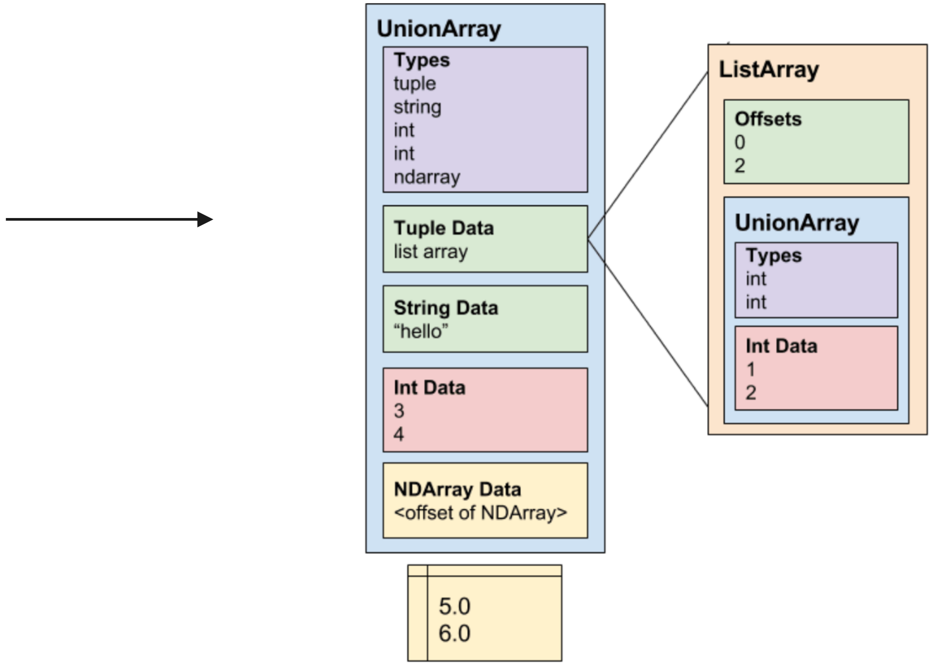
- Goals
    - Very efficient with **large numerical data** (e.g. Numpy arrays and Pandas dataframes)
    - As fast as Pickle for **general Python types**
    - Compatible with **shared memory** (allowing multiple processes to use the same data without copying it)
    - **Deserialization** should be extremely fast
    - **language independent**



快速的序列化和反序列化

- Making **deserialization** fast is important.
  - An object may be serialized once and then deserialized many times
  - A common pattern is for many objects to be serialized in parallel and then aggregated and deserialized one at a time on a single worker making deserialization the bottleneck
- Deserialization is fast and barely visible
  - **Using only the schema, can compute the offsets of each value in the data blob without scanning through the data blob** (unlike Pickle, this is what enables fast deserialization)
  - Avoid copying or otherwise converting large arrays and other values during deserialization(the savings largely come from the lack of memory movement)

```
[(1, 2), 'hello', 3, 4, np.array([5.0, 6.0])]
```



Microsoft

如何评价分布式强化学习框架?

- **Sampling Efficiency**
- Large Scale Test
- Multi-GPU

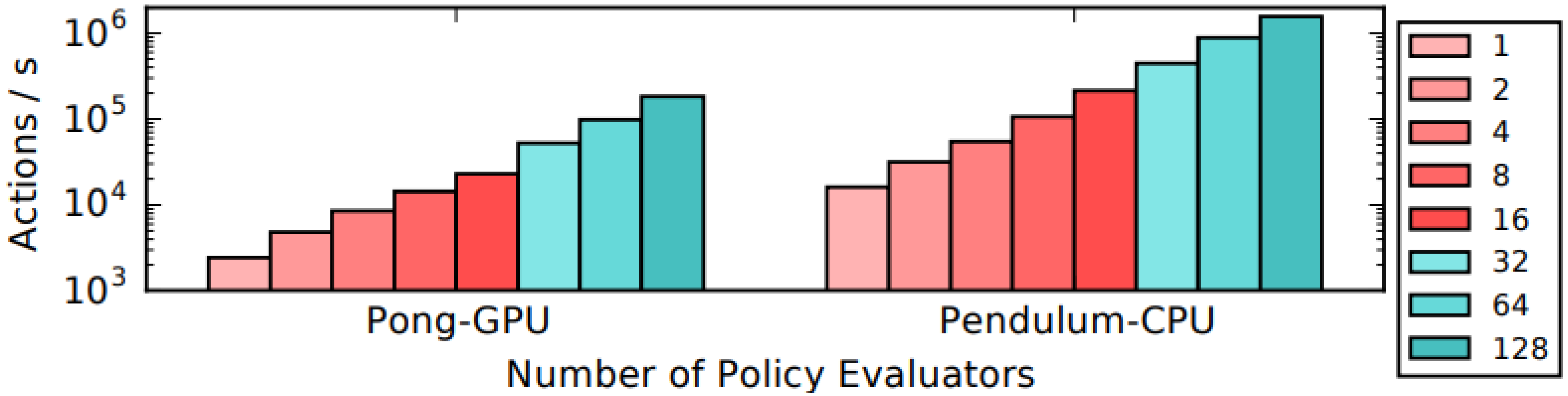
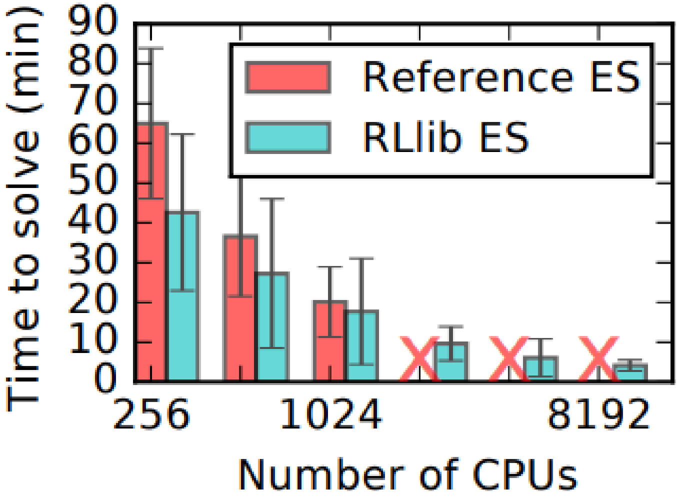


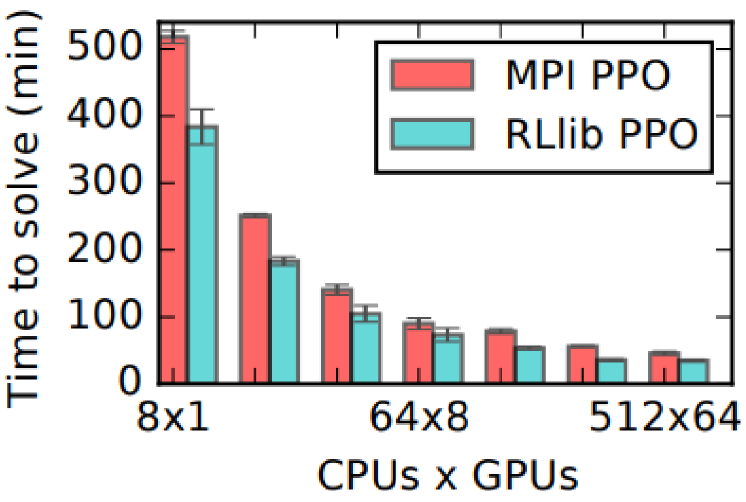
Figure. Policy evaluation throughput scales nearly linearly from 1 to 128 cores.

如何评价分布式强化学习框架?

- Sampling Efficiency
- **Large Scale Test**
- Multi-GPU



(a) Evolution Strategies



(b) PPO

Microsoft

如何评价分布式强化学习框架?

- Sampling Efficiency
- Large Scale Test
- **Multi-GPU**

Policy Optimizer	Gradients computed on	Environment	SGD throughput
Allreduce-based	4 GPUs, Evaluators	Humanoid-v1	330k samples/s
		Pong-v0	23k samples/s
	16 GPUs, Evaluators	Humanoid-v1	**440k samples/s**
		Pong-v0	**100k samples/s**
Local Multi-GPU	4 GPUs, Driver	Humanoid-v1	**2.1M samples/s**
		Pong-v0	N/A (out of mem.)
	16 GPUs, Driver	Humanoid-v1	1.7M samples/s
		Pong-v0	**150k samples/s**

Microsoft

RLlib的小总结

- 优雅而简单的分布式编程语言
- 容错和高并发的分布式框架
- 通用的强化学习接口
- 为python对象优化的高效通信框架

在Rllib还有什么可以改进的地方？新一代的强化学习平台是怎样的?

强化学习的其他挑战

- 可复现性 *(e.g. SURREAL)*
- 可解释性
- 从少量的数据中学习
- 安全限制
- 实时推理
- …

Microsoft

路漫漫其修远兮，吾将上下而求索~

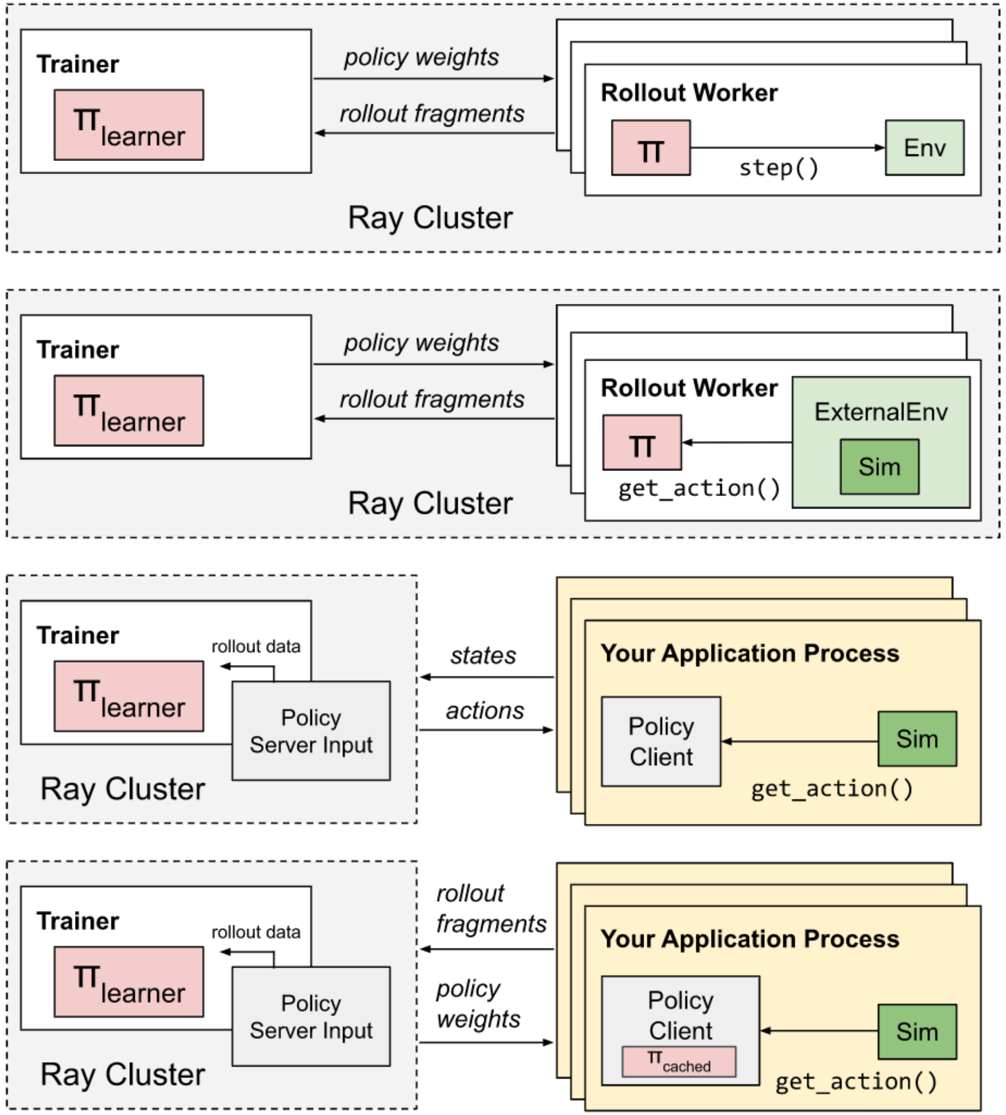
Microsoft

参考资料

- Ray: A Distributed Framework for Emerging AI Applications
- RLlib: Abstractions for Distributed Reinforcement Learning
- DISTRIBUTED PRIORITIZED EXPERIENCE REPLAY
- Rainbow: Combining Improvements in Deep Reinforcement Learning
- SEED RL: Scalable and Efficient Deep-RL with Accelerated Central Inference
- IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures
- Asynchronous Methods for Deep Reinforcement Learning
- SURREAL: Open-Source Reinforcement Learning Framework and Robot Manipulation Benchmark
- Challenges of Real-World Reinforcement Learning
- Apache Arrow https://arrow.apache.org/
- https://wesmckinney.com/blog/arrow-streaming-columnar/
- Modin(speed up the pandas in ray) https://github.com/modin-project/modin
- https://www.zhihu.com/question/377263715
- https://www.slideshare.net/databricks/enabling-composition-in-distributed-reinforcement-learning-with-ray-rllib-with-eric-liang-and-richard-liaw
- https://github.com/deepmind/reverb

Microsoft

支持的复杂的与环境的交互方式



(1) Standard environments (e.g., gym.Env, MultiAgentEnv types) are created and stepped by RLlib rollout workers.

(2) External environments (ExternalEnv) run in their own thread and pull actions as needed. RLlib still creates one external env class instance per rollout worker.

(3) Applications running outside the Ray cluster entirely can connect to RLlib using PolicyClient, which computes actions remotely over RPC.

(4) PolicyClient can be configured to perform inference locally using a cached copy of the policy, improving rollout performance.