Deep Reinforcement Learning

Lecture 9 — Distributed Reinforcement Learning

國立清華大學
NATIONAL TSING HUA UNIVERSITY

National Tsing Hua University
Department of Computer Science

Prof. Chun-Yi Lee

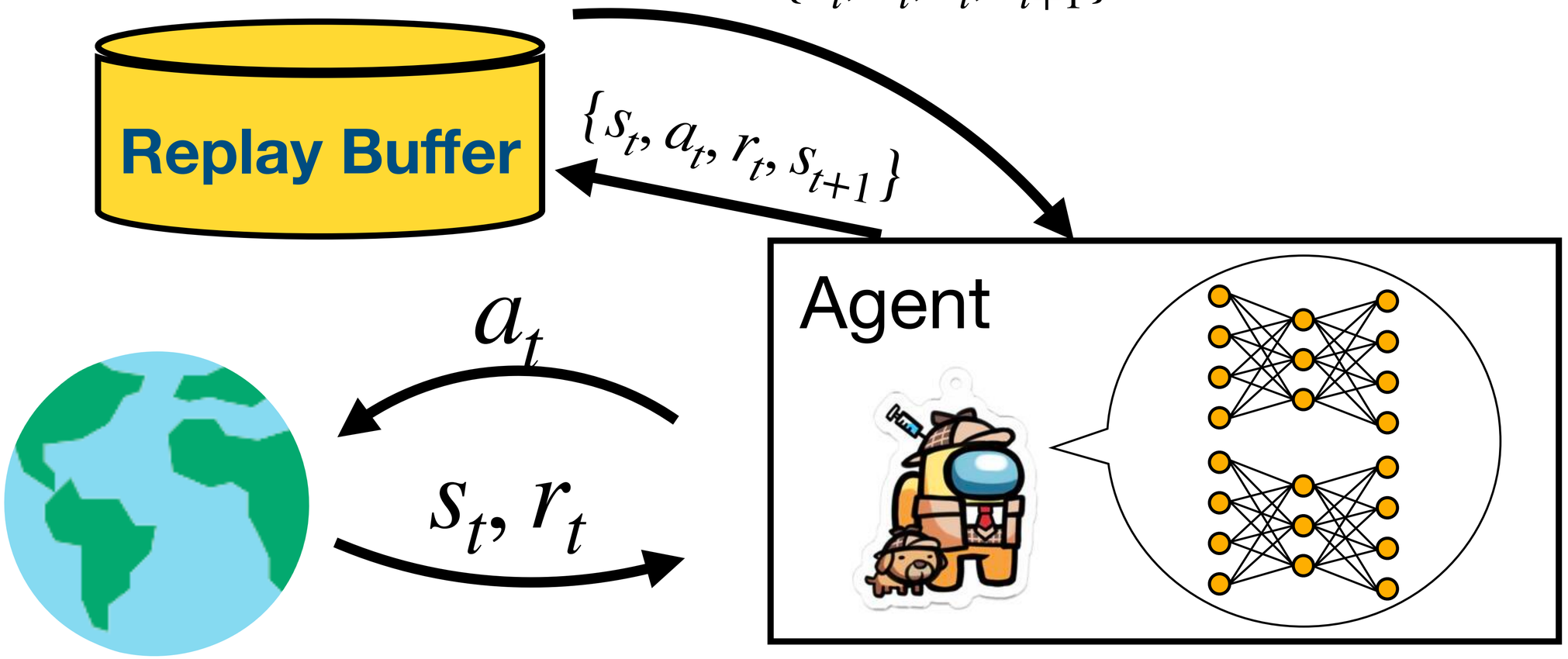
Outline

- Asynchronous Advantage Actor Critic (A3C)
- Ape-X
- IMPALA

Asynchronous Advantage Actor Critic

One thread reinforcement learning

- **Take DQN for example**



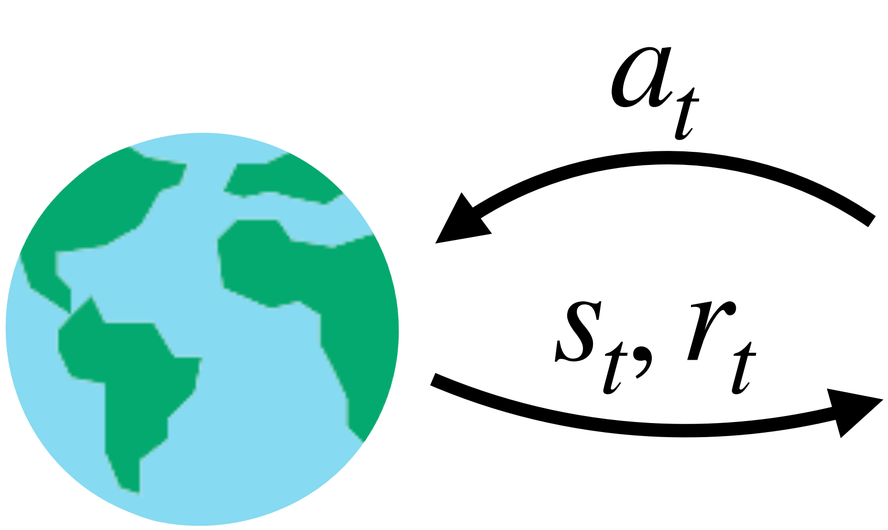
Asynchronous Advantage Actor Critic

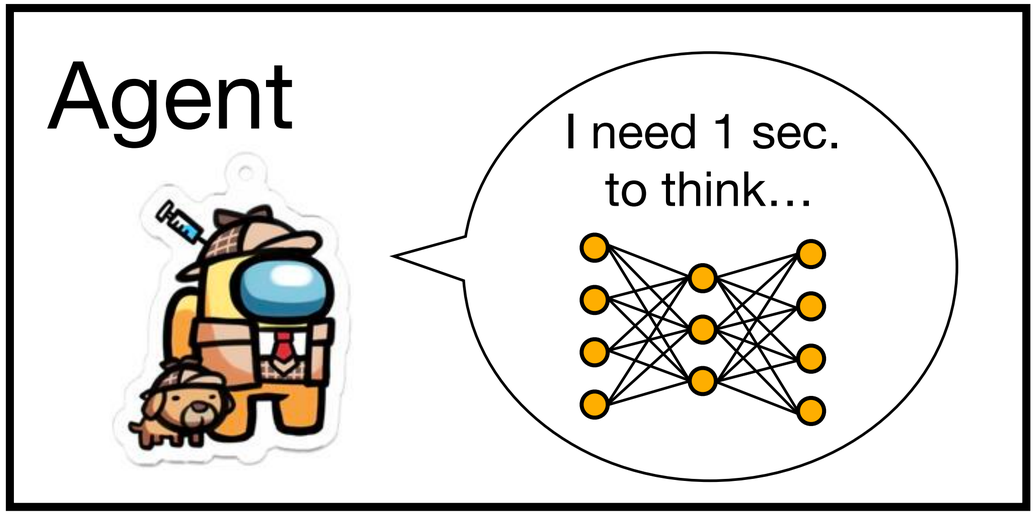
One thread Reinforcement Learning

- **For each timestep *t*, it collects only one data sample**

$$\{s_t, a_t, r_t, s_{t+1}\} \quad \{s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2}\}$$

- **If you need collect 100,000, you need to take 100,000 seconds**

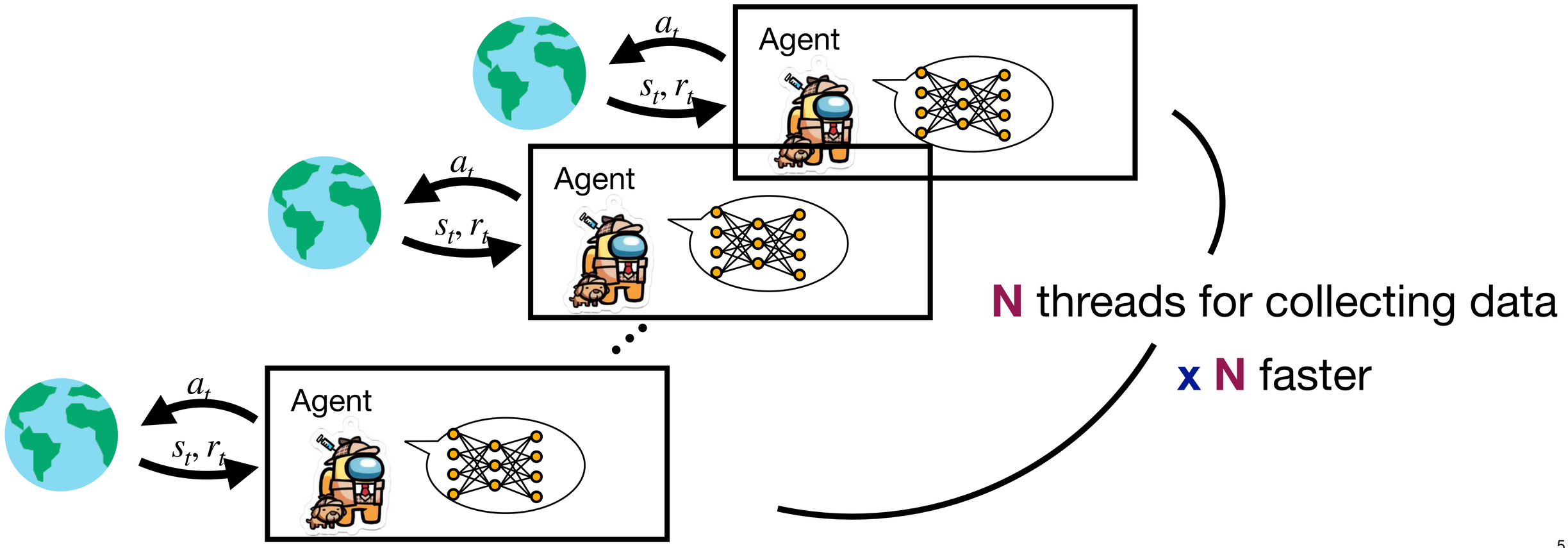




Asynchronous Advantage Actor Critic

One thread Reinforcement Learning

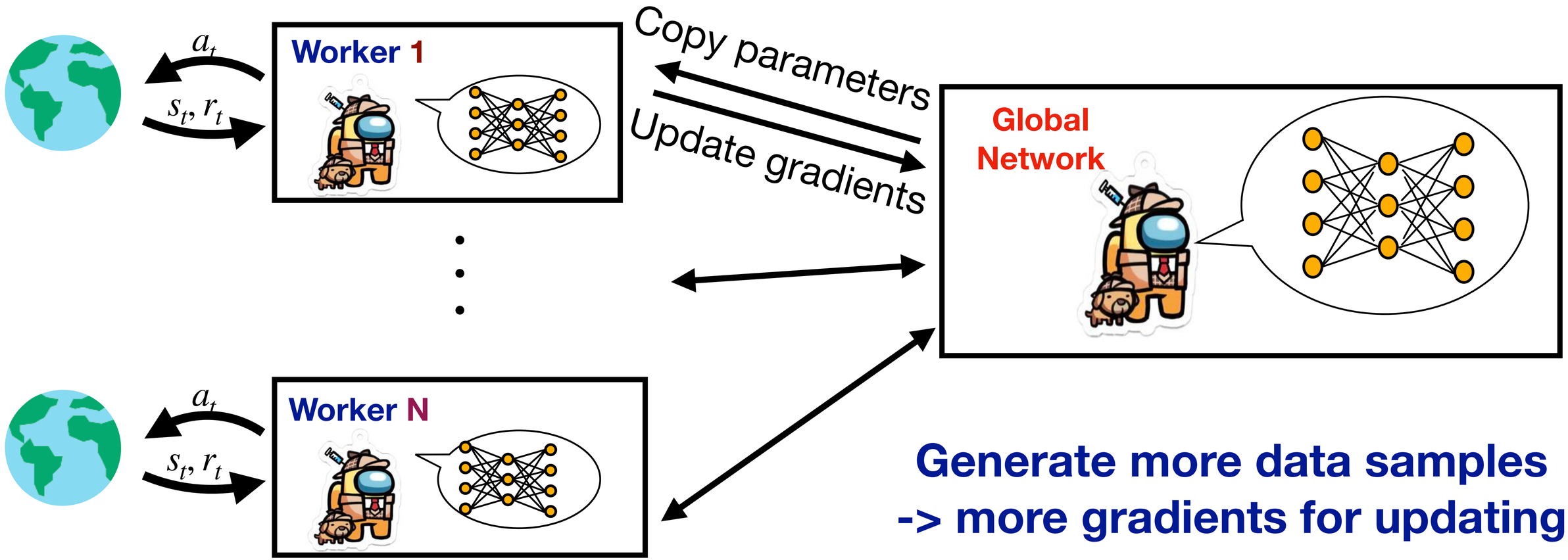
- **Any better way…?**



Asynchronous Advantage Actor Critic

Asynchronous Advantage Actor Critic

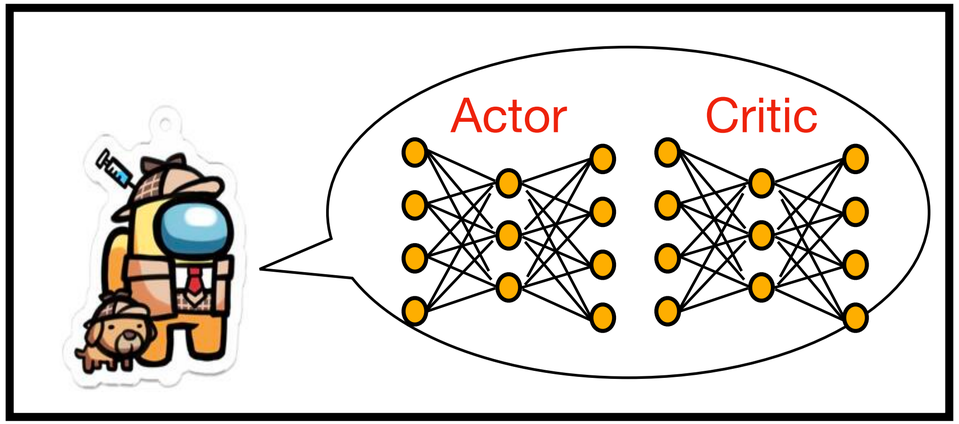
- A3C



Asynchronous Advantage Actor Critic

Asynchronous Advantage Actor Critic

- Asynchronous Advantage Actor Critic
  - For all workers and the global network, each of them has one Actor $\pi_{\theta'}$ and one Critic $V_{\theta}(s)$



Asynchronous Advantage Actor Critic

Asynchronous Advantage Actor Critic

- Asynchronous Advantage Actor Critic
  - It uses advantage function in policy gradient methods to the update policy

$$\nabla_{\theta'} \log \pi(a_t \mid s_t, \theta') A(a_t, s_t)$$

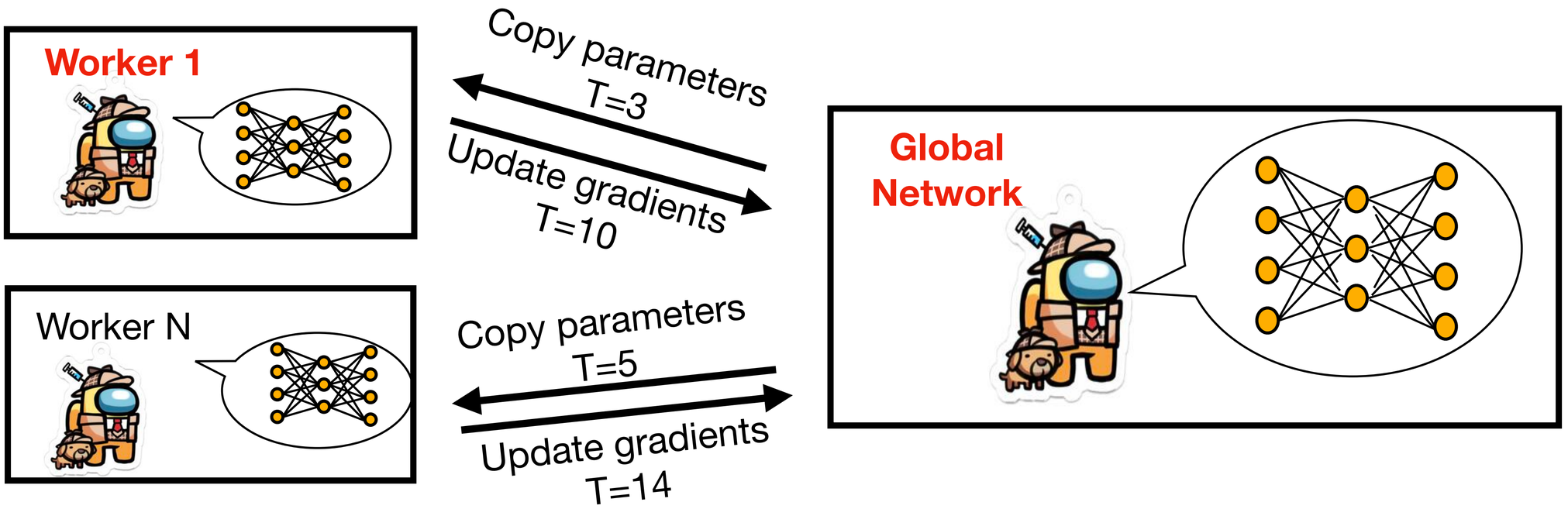
  - It estimates the advantage function by combining the value function and discounted reward R (as an easily estimate as an estimate of Q)

$$A(a_t, s_t) = R - V(s_t, \theta), \quad R = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}, \theta)$$

Asynchronous Advantage Actor Critic

Asynchronous Advantage Actor Critic

- Asynchronous Advantage Actor Critic
  - Different threads copy network and update at different timesteps



Asynchronous Advantage Actor Critic

Asynchronous Advantage Actor Critic

- Why Asynchronous ?
    - The experience of each thread is independent of the experience of the others
    - This enables the experience to become more diverse.
    - Avoid data correlation

Asynchronous Advantage Actor Critic

Algorithm

1. The worker copies the global network parameter
2. The worker interacts with the environment
3. The worker calculates the gradients for the value and the policy
4. The worker applies the gradients to the global network

Asynchronous Advantage Actor Critic

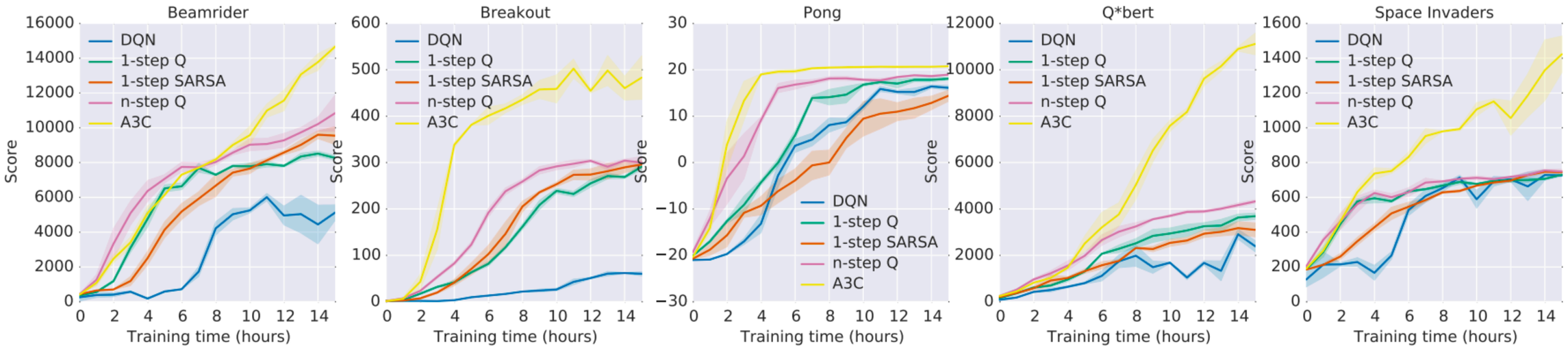
Algorithm

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Asynchronous Advantage Actor Critic

Experimental results in terms of the performance



Asynchronous Advantage Actor Critic

Experimental results in terms of the training time

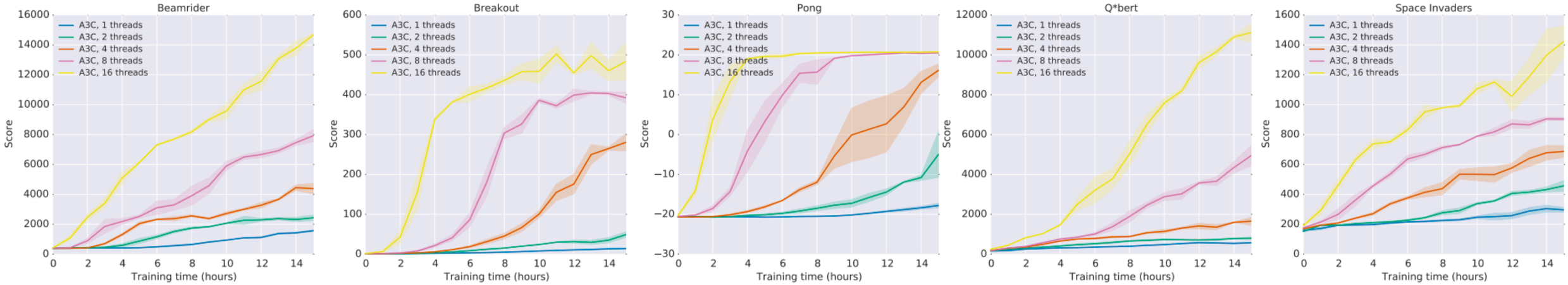
Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Table 1. Mean and median human-normalized scores on 57 Atari games using the human starts evaluation metric. Supplementary Table S3 shows the raw scores for all games.

Asynchronous Advantage Actor Critic

Experimental result for different number of worker threads

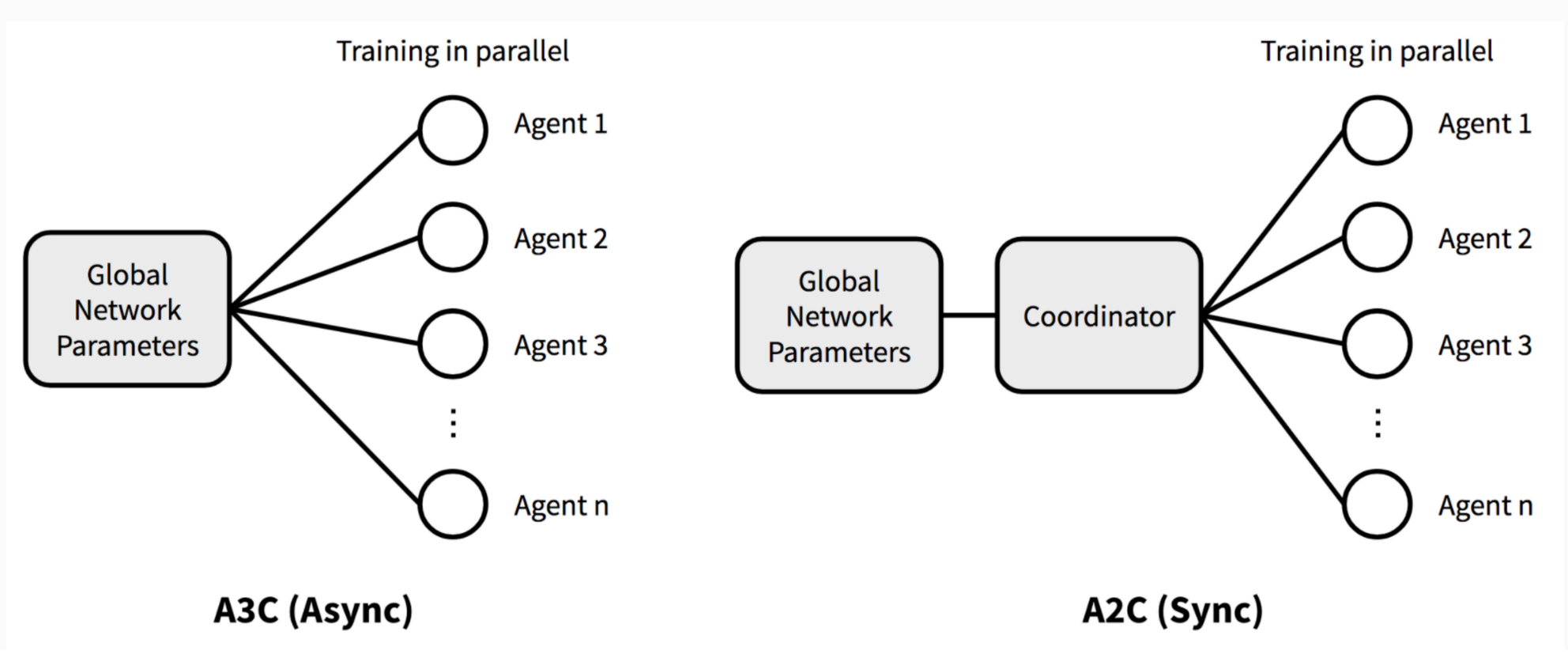
- More threads, performance more better



Advantage Actor Critic

A2C v.s. A3C

- Synchronous advantage Actor Critic



Advantage Actor Critic

A2C v.s. A3C

- A3C uses asynchronous updates, which could cause the global policy to be not **optimal**.
    - Different gradients may go to different direction, making the global model not easy to **converge**)
- A2C collects the gradients (**GPU efficiency**), and update it together in a synchronous manner

Outline

- Asynchronous Advantage Actor Critic (A3C)
- **Ape-X**
- **IMPALA**

Ape-X

Distributed prioritized experience replay

Ape-X

Distributed prioritized experience replay

- A paper accepted and presented at ICLR 2018, a top tier AI conference

DISTRIBUTED PRIORITIZED EXPERIENCE REPLAY

Dan Horgan
DeepMind
horgan@google.com

John Quan
DeepMind
johnquan@google.com

David Budden
DeepMind
budden@google.com

Gabriel Barth-Maron
DeepMind
gabrielbm@google.com

Matteo Hessel
DeepMind
mtthss@google.com

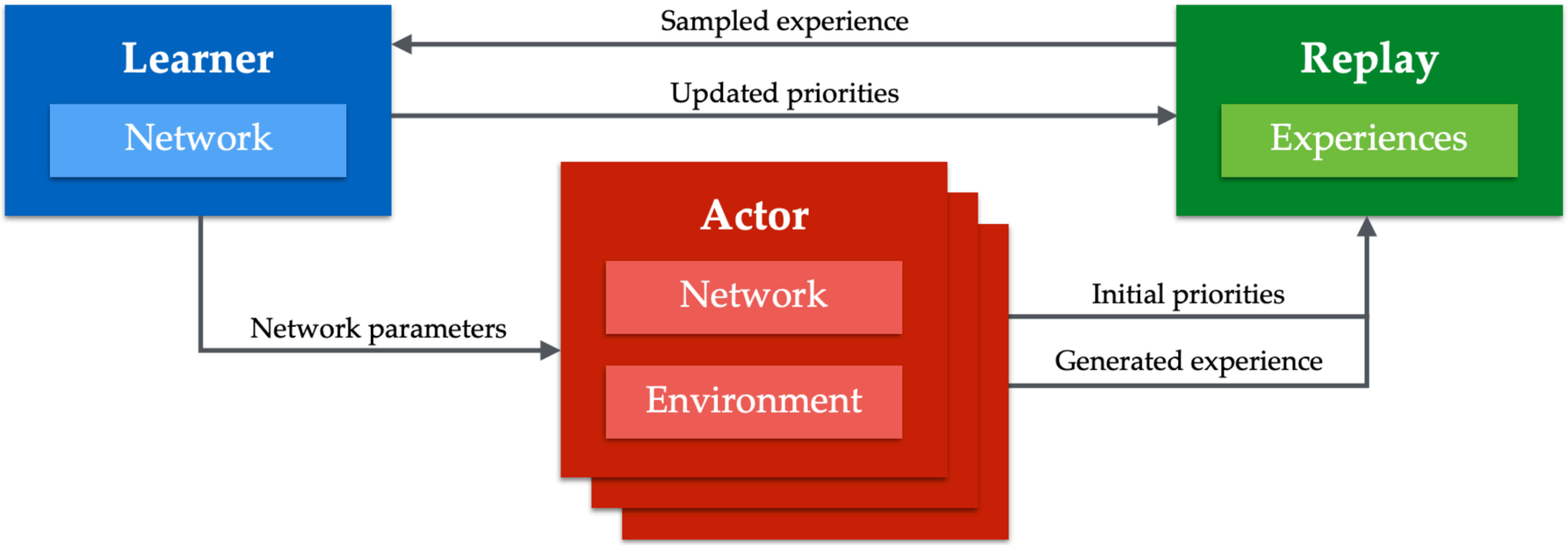
Hado van Hasselt
DeepMind
hado@google.com

David Silver
DeepMind
davidsilver@google.com

Ape-X

Distributed prioritized experience replay

- A distributed RL + special prioritized replay buffer
- It can be used on the algorithms with replay buffer (DQN, DDPG …)



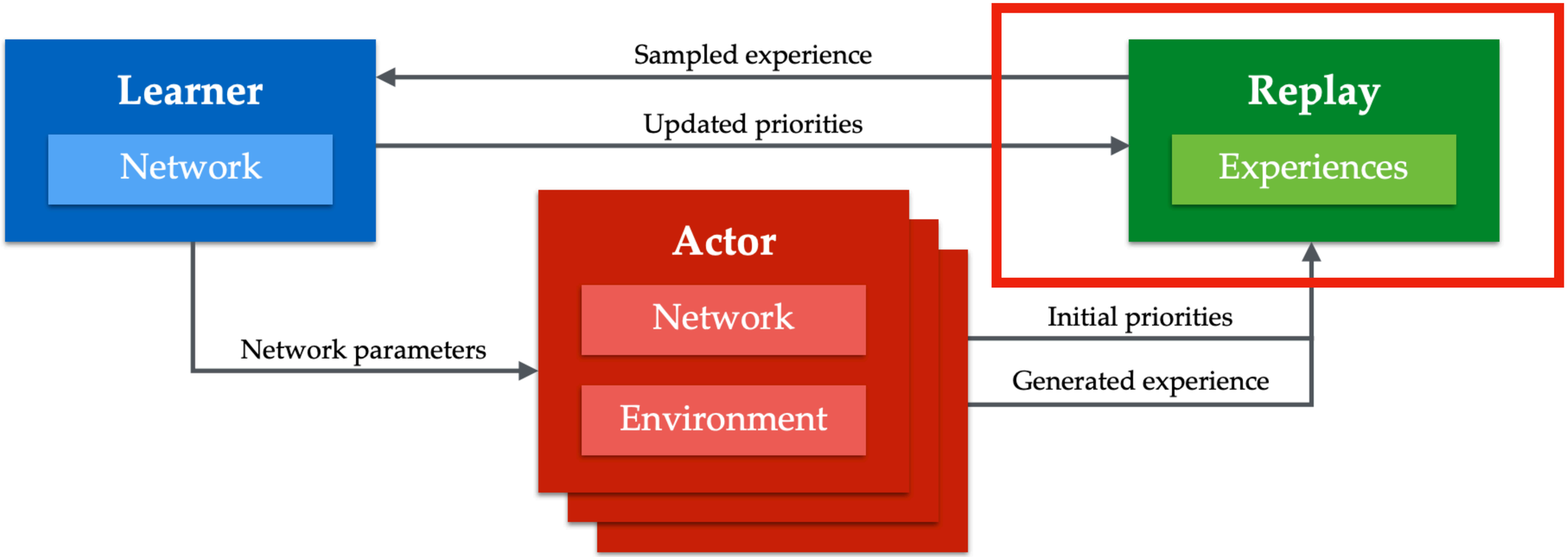
Ape-X

The key components in Ape-X

- Learner * **1**
  - Samples data and computes gradient to update parameters
- Actor * **N**
  - Generates data and pre-computes the priority
- Replay * **1**
  - Shared and centralized prioritized replay buffer
  - Collects and maintains all actor’s data

Ape-X

The prioritized replay buffer in Ape-X



Ape-X

The issue in the shared and centralized replay buffer

- In the original prioritized replay buffer:
  1. Data are initialized to the maximum priority
  2. Updated when they were sampled

- Some problems exist in the distributed version
  - Many actors would collect data
  - Waiting for the learner to update priorities would result in a myopic focus on the most recent data

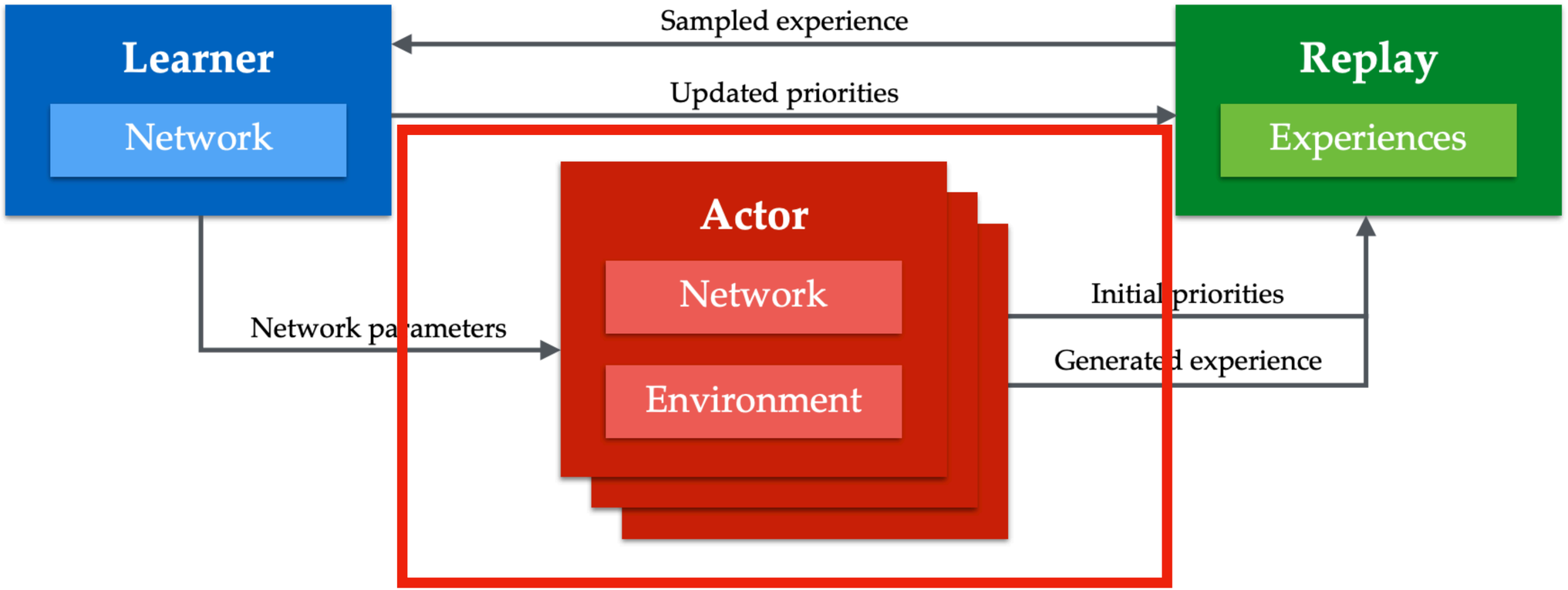
Ape-X

The prioritized replay buffer used by Ape-X

- The Ape-X version of the prioritized replay buffer,
  - The priority of the data would be calculated in the actors first
  - They are then sent back to the global buffer
- The policy in the actors would not be outdated, thus it is suitable to compute priorities for the data online.
- Ape-X ensures the data in the buffer to have more accurate priorities, at no extra cost.

Ape-X

The actors in Ape-X



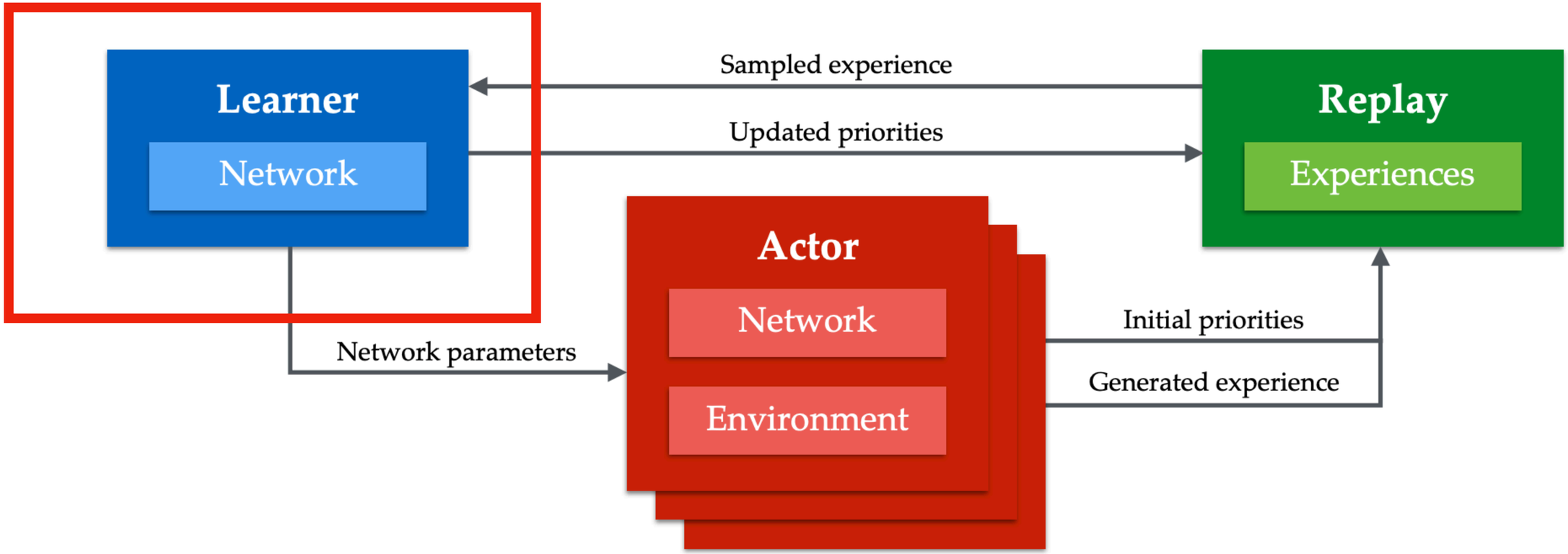
Ape-X

The actors in Ape-X

- Periodically copies the learner’s network parameters
- Step environment in each actor to generate data samples
- Add data into each actor’s local buffer
- Periodically send data and their priorities to the global buffer

Ape-X

The learner in Ape-X



Ape-X

The learner in Ape-X

- Samples data from the experience replay buffer
- Computes the loss (e.g. MSE) and updates the parameters
- Updates the priorities of the sampled data
- Removes old experience from the experience replay buffer

Ape-X

The experimental results compared to DQN and other RL methods

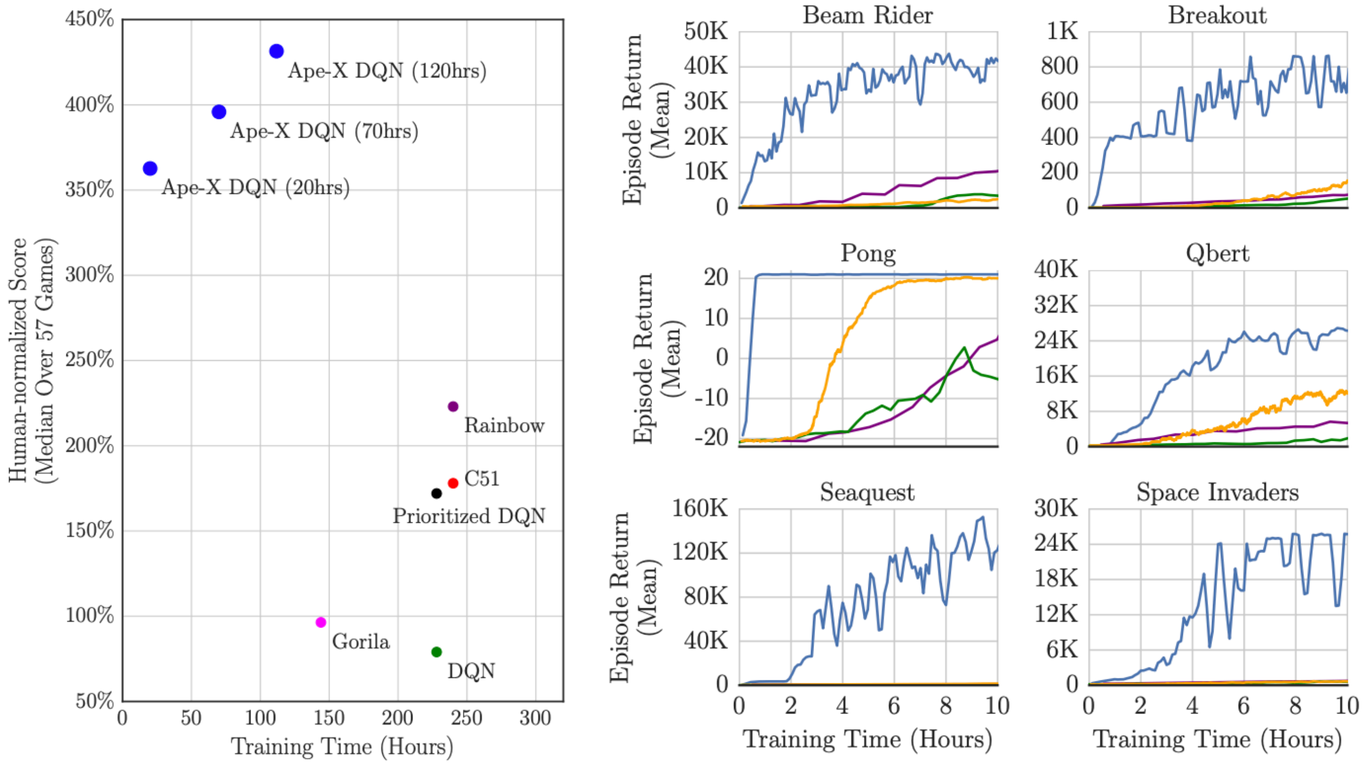
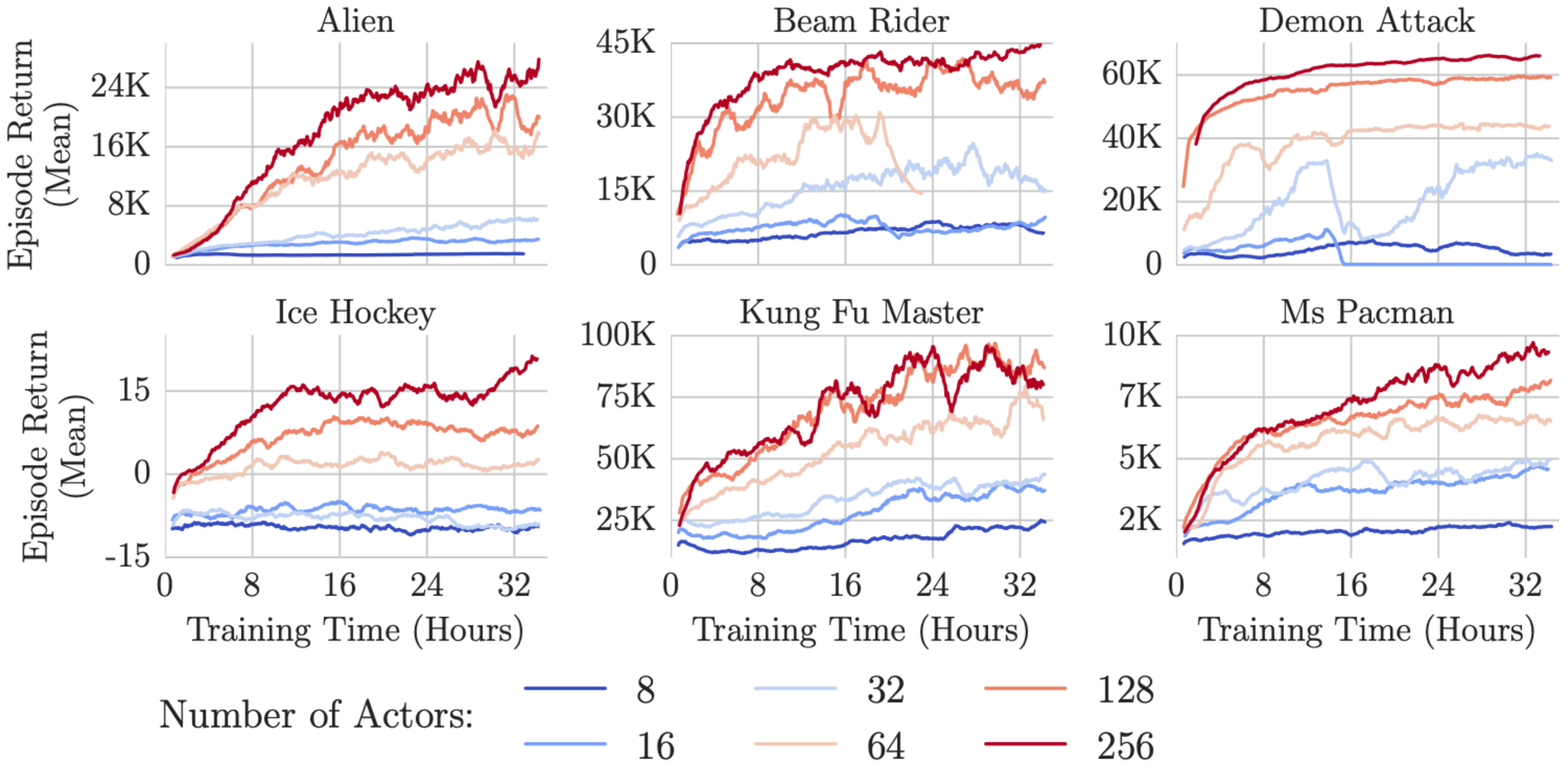


Figure 2: Left: Atari results aggregated across 57 games, evaluated from random no-op starts. Right: Atari training curves for selected games, against baselines. Blue: Ape-X DQN with 360 actors; Orange: A3C; Purple: Rainbow; Green: DQN. See appendix for longer runs over all games.

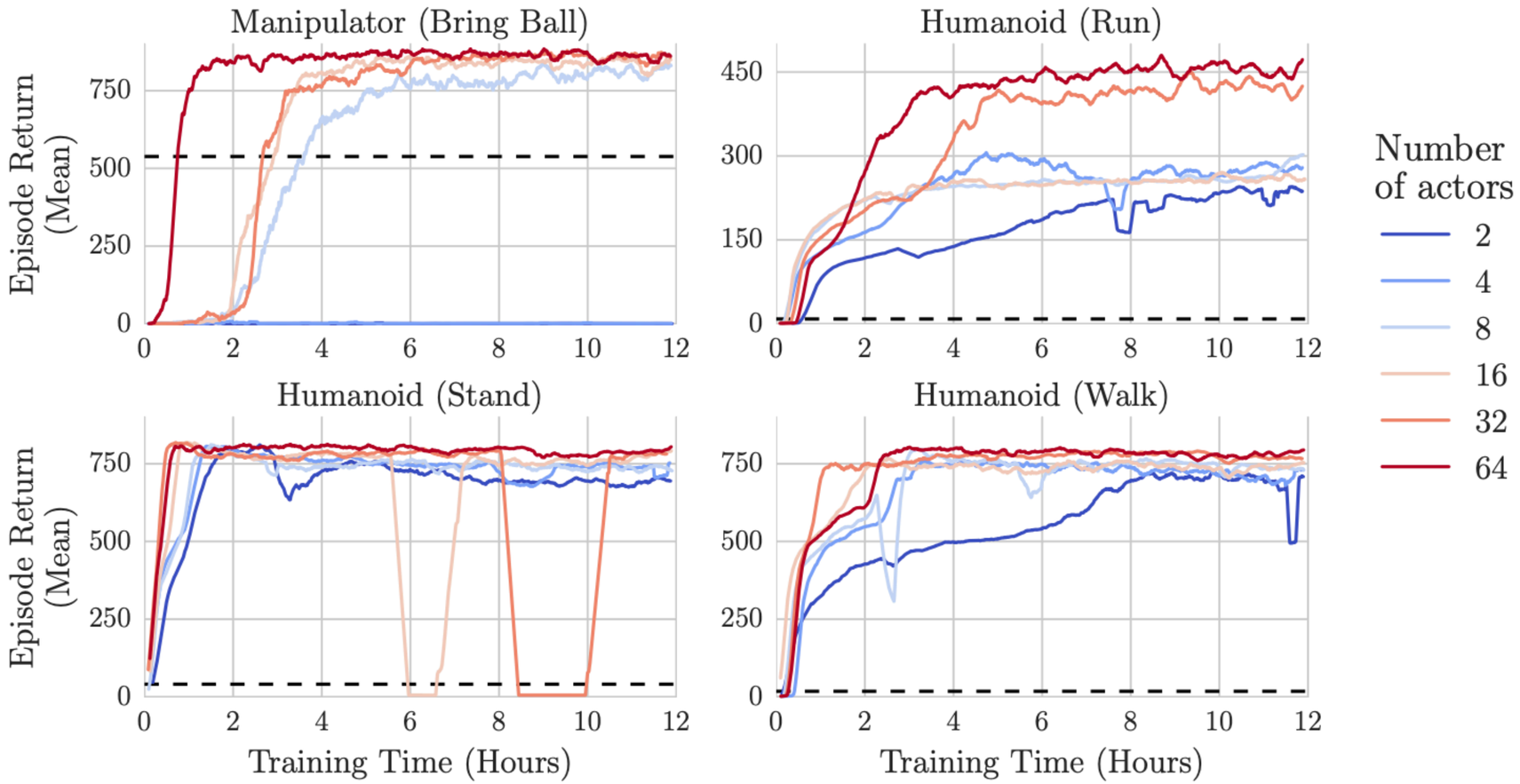
Ape-X

The analysis for different number of actors



Ape-X

The experimental resultful continuous control tasks



The black dashed lines correspond to the vanilla DDPG

Outline

- Asynchronous Advantage Actor Critic (A3C)
- Ape-X
- **IMPALA**

CHARACTER VISUAL
Matsumoto

IMPALA

Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures

- A paper accepted and presented at ICML 2018, a top tier AI conference

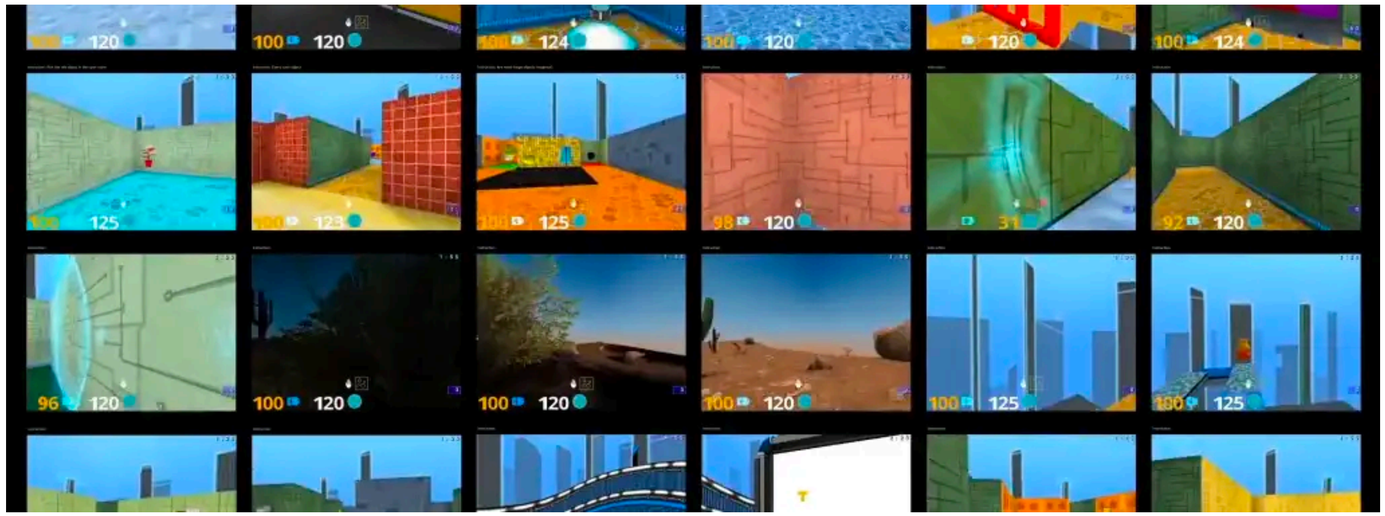
IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures

Lasse Espeholt [*1] **Hubert Soyer** [*1] **Remi Munos** [*1] **Karen Simonyan** [1] **Volodymyr Mnih** [1] **Tom Ward** [1]
Yotam Doron [1] **Vlad Firoiu** [1] **Tim Harley** [1] **Iain Dunning** [1] **Shane Legg** [1] **Koray Kavukcuoglu** [1]

IMPALA

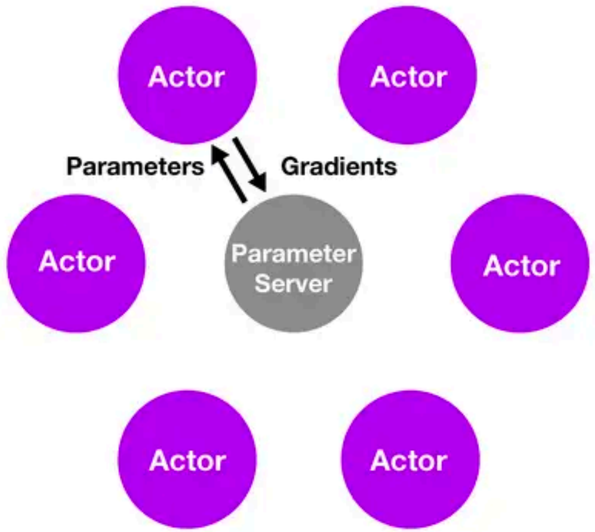
Main contributions

- Decouples the actor phase and the learner phase
- Proposed “V-trace” for off-policy correction

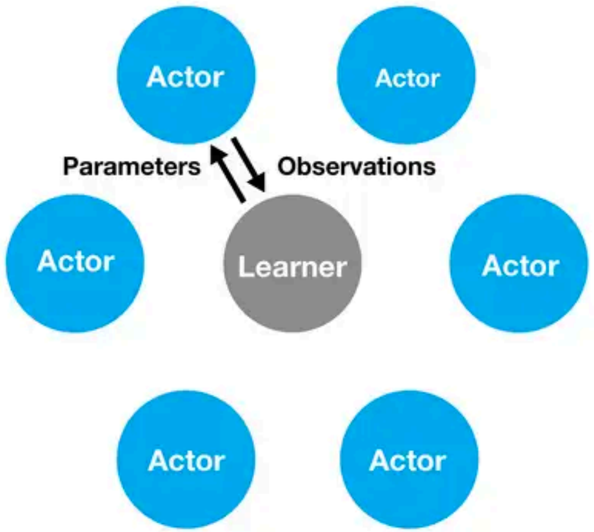


IMPALA

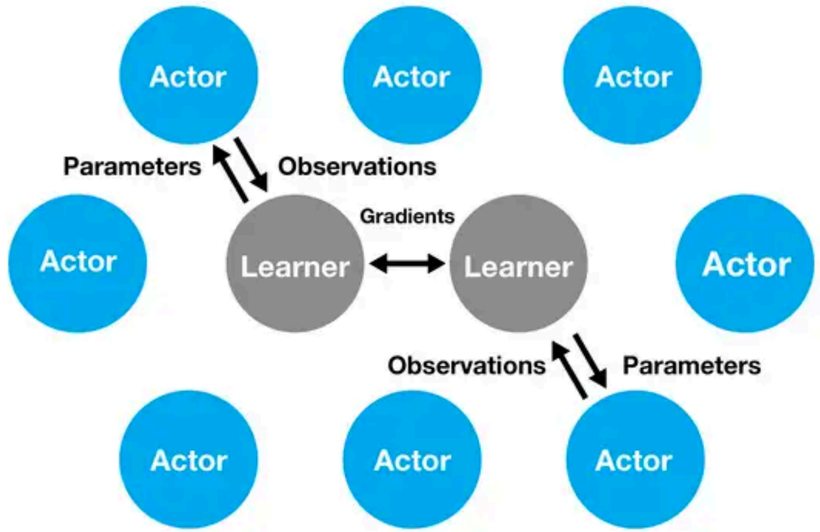
A comparison of different distributed RL schemes



A3C



IMPALA - Single Learner

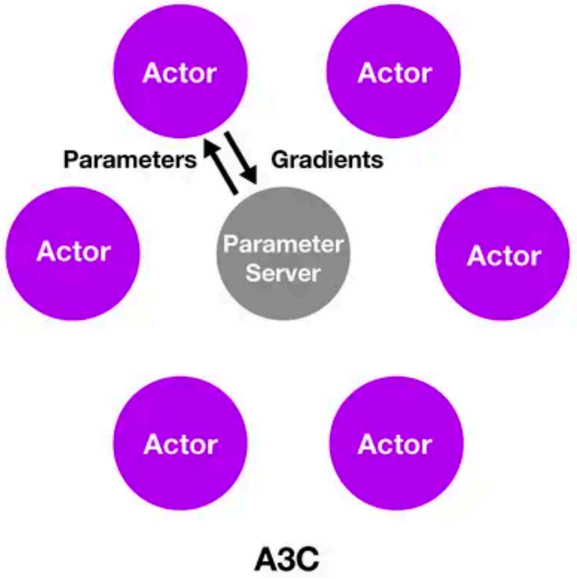


IMPALA - Multiple Learners

IMPALA

In A3C

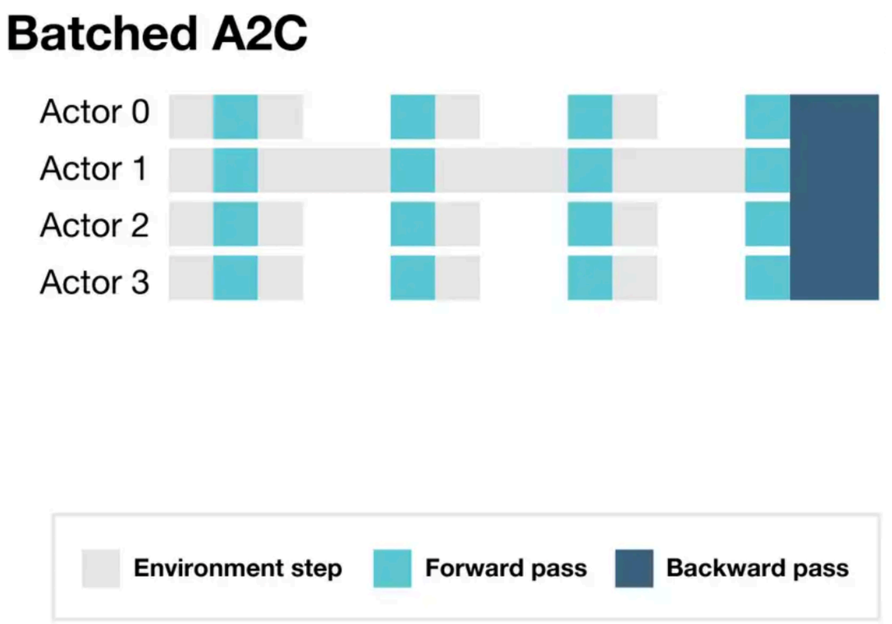
- The actors uses a clone of the policy parameters
- Periodically, the actors share the computed gradients to the global network to apply updates
- Each actor asynchronously shares and copies the parameters
- However, the results show that A2C has better performance than A3C



IMPALA

A2C

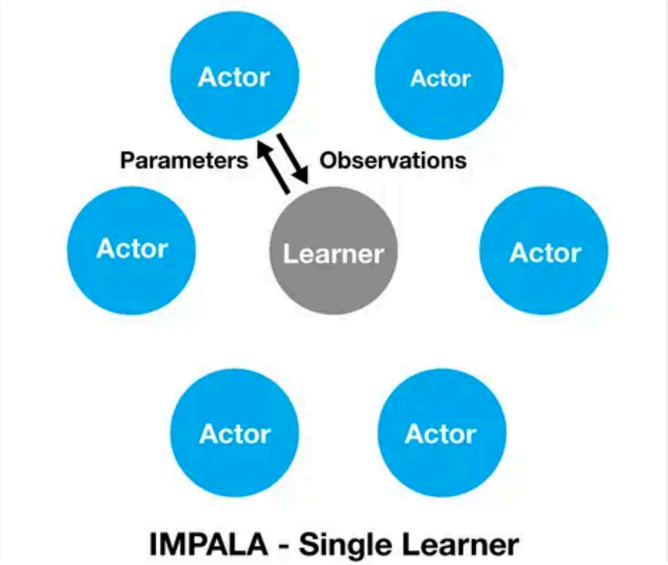
- Each actor synchronously shares and copys the parameters
- However, the learning phase need to wait until all the actors complete their data generation
- Not efficient to utilized CPUs



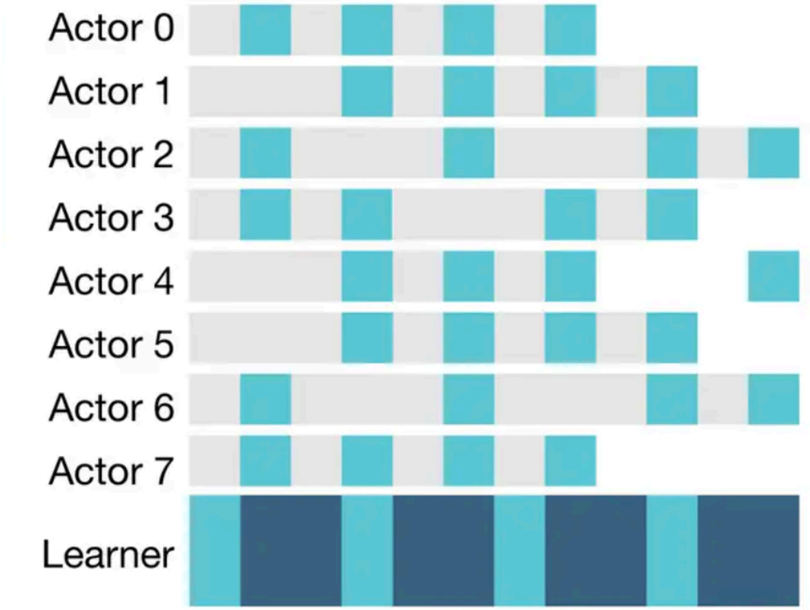
IMPALA

IMPALA

- For each actor, they do not need to wait for each other or the learner
- For the learner, it can always update itself without waiting for the actors
- More efficient in CPU utilization



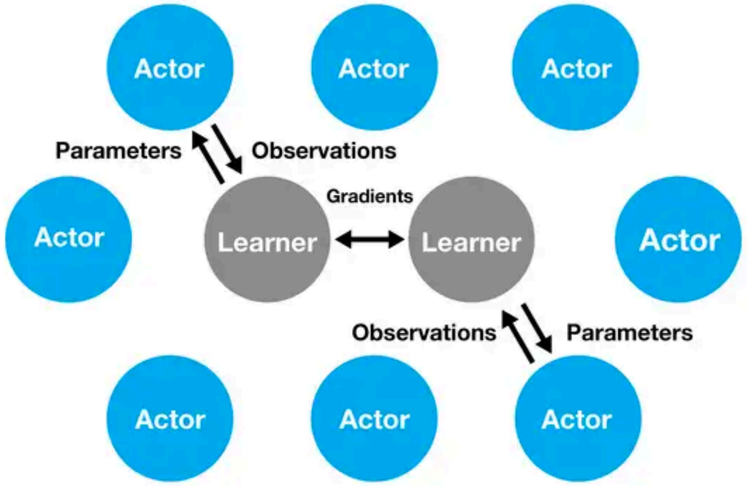
IMPALA - Single Learner



IMPALA

IMPALA —Multi learners

- Since IMPALA decouples the learning and the acting phase, it could have multi learners
- Each learner could maintain its own policy and share gradients to each other



IMPALA - Multiple Learners

IMPALA

IMPALA

- However, decoupling the actor and learner causes the policy in the actor much lag behind the learner. (Off-policy update)
- It needs some term to off-policy corretion
- It proposed “V-trace” term

V-trace

The formulation of V-trace

- For V-trace term v_s (return target)

$$v_s = V(x_s) + \delta_s V + \gamma c_s \big(v_{s+1} - V(x_{s+1})\big).$$

- Use the v_s term to update value function

$$\big(v_s - V_\theta(x_s)\big)\nabla_\theta V_\theta(x_s),$$

IMPALA

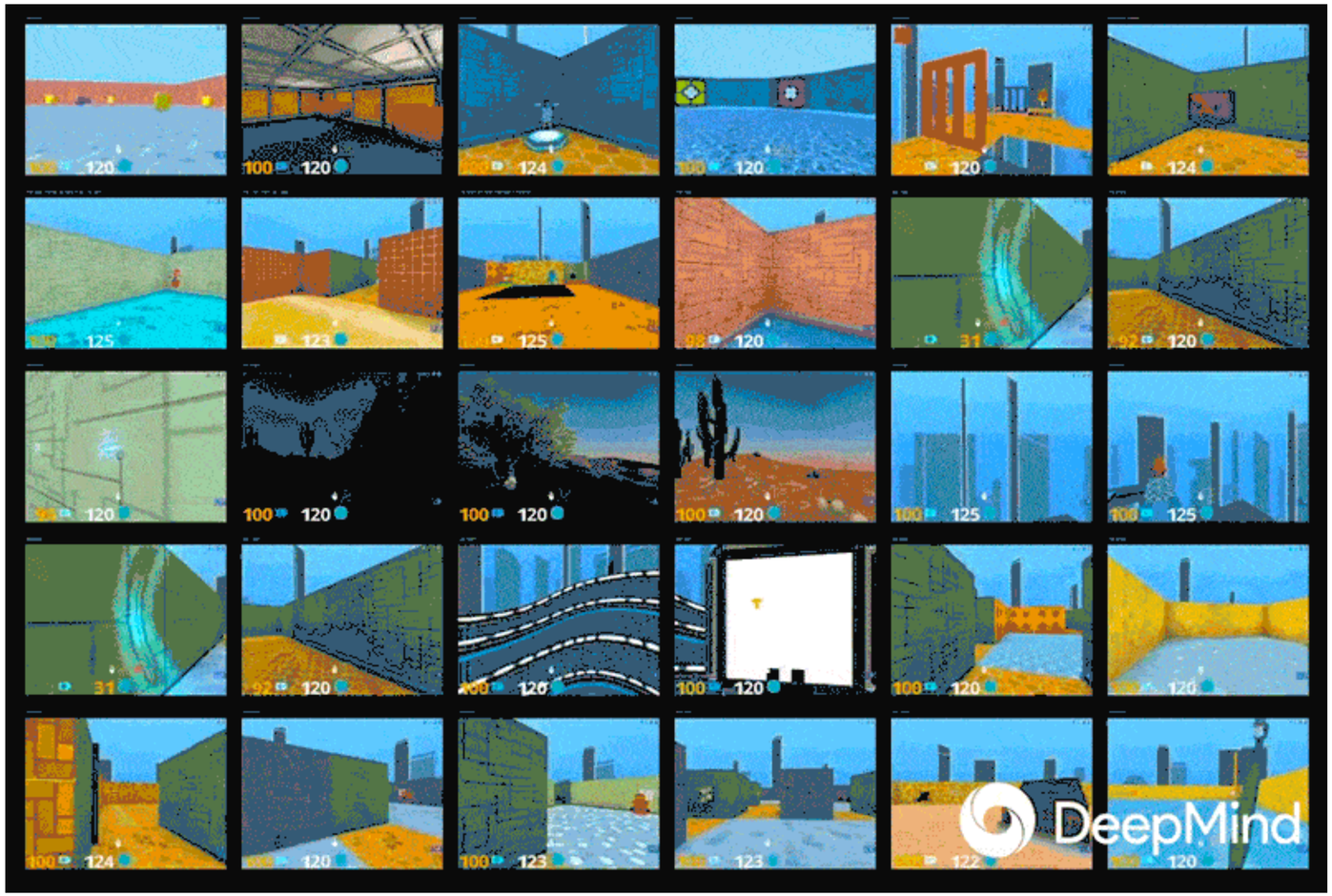
Experimental results — Atari

Human Normalised Return	Median	Mean
A3C, shallow, experts	54.9%	285.9%
A3C, deep, experts	117.9%	503.6%
Reactor, experts	187%	N/A
IMPALA, shallow, experts	93.2%	466.4%
IMPALA, deep, experts	191.8%	957.6%
IMPALA, deep, multi-task	59.7%	176.9%

Table 4. Human normalised scores on Atari-57. Up to 30 no-ops at the beginning of each episode. For a level-by-level comparison to ACKTR (Wu et al., 2017) and Reactor see Appendix C.1 .

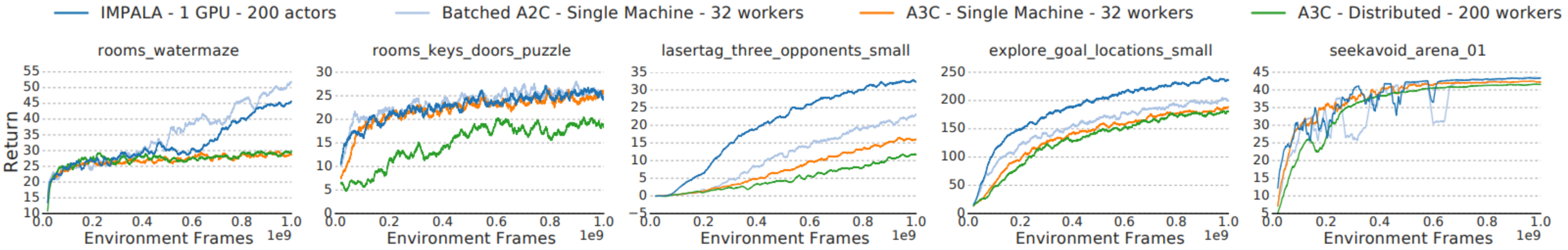
IMPALA

DeepMind Lab tasks



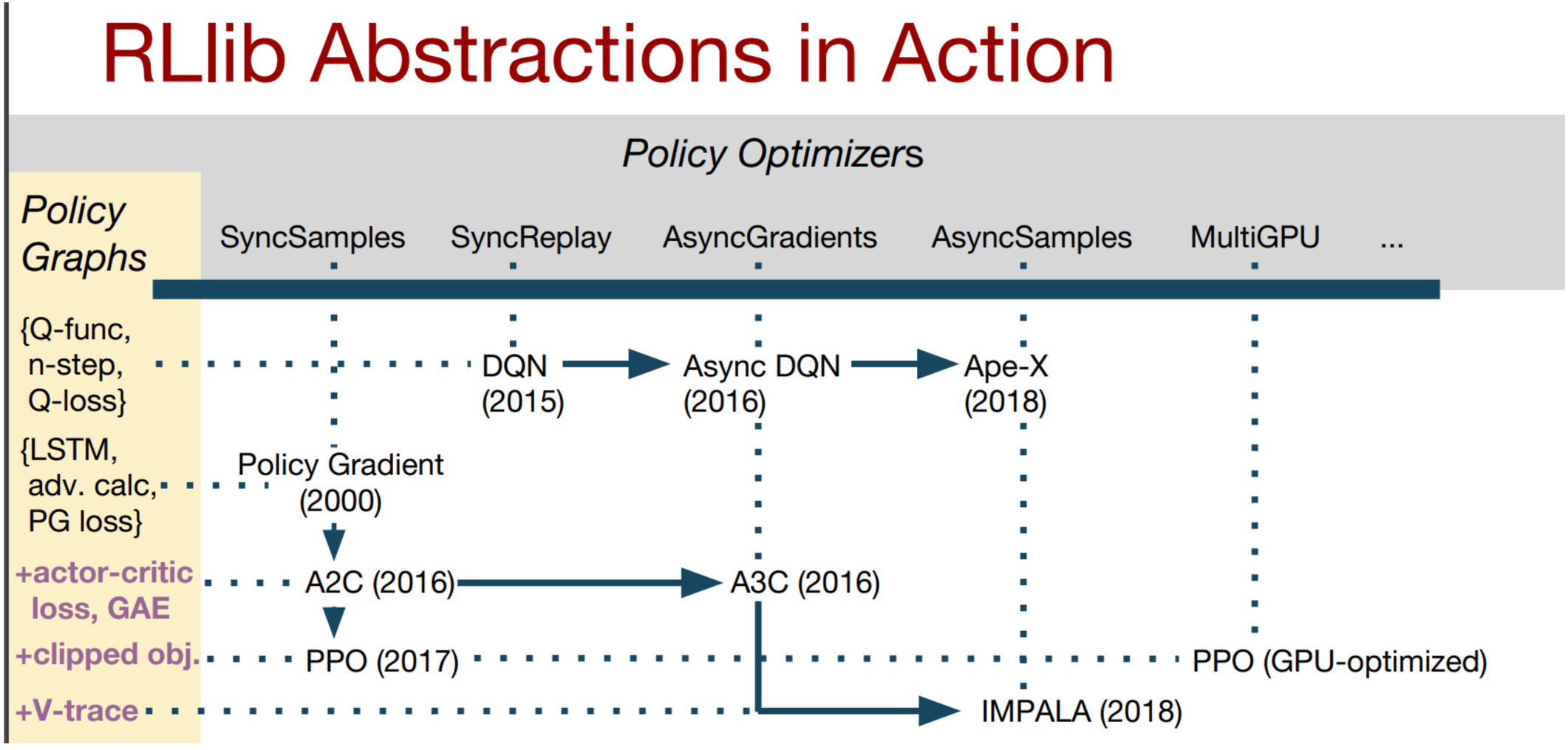
IMPALA

Result — DeepMind Lab tasks



Summary

Evolution of Distributed RL



ありがとう ございます