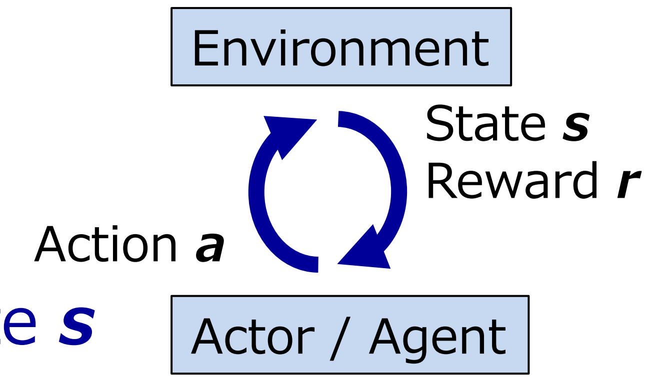
Accelerating Distributed Deep Reinforcement Learning by In-Network Experience Sampling

Masaki Furukawa, Hiroki Matsutani
(Keio University, Japan)

Background: Reinforcement learning

- Goal
  - Acquire an action-selection policy that maximizes a long-term reward by taking actions and observing the environment



- Q-value
  - Expected value for action ***a*** in state ***s***
- Q-learning algorithm
  - ***Q(s, a)*** is updated by taking action $\boldsymbol{a_t}$ and observing the next state $\boldsymbol{s_{t+1}}$ and reward $\boldsymbol{r_t}$

$$Q^{new}(s_t, a_t) \leftarrow \underset{\text{Old value}}{Q(s_t, a_t)} + \overset{\text{Learning rate}}{\alpha} \left\{ \underset{\text{Reward}}{r_t} + \overset{\text{Discount rate}}{\gamma} \underset{\text{Expected future value}}{\max_{a \in A} Q(s_{t+1}, a)} - \underset{\text{Old value}}{Q(s_t, a_t)} \right\}$$

Background: Deep Q-Network

- Q-learning

Estimated optimal future value

$$Q(s_t, a_t) = r_t + \gamma \max_{a \in A} Q(s_{t+1}, a)$$

Set of all the possible actions

- DQN: Q-learning + Deep neural network
  - Q-value is approximated with deep neural network $\boldsymbol{\theta}$
  - $\boldsymbol{\theta}$ is updated to minimize the loss

$$Q(s_t, a_t; \theta) = r_t + \gamma \max_{a \in A} Q(s_{t+1}, a; \theta)$$

$$L(\theta) = \left\{ r_t + \gamma \max_{a \in A} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta) \right\}^2$$

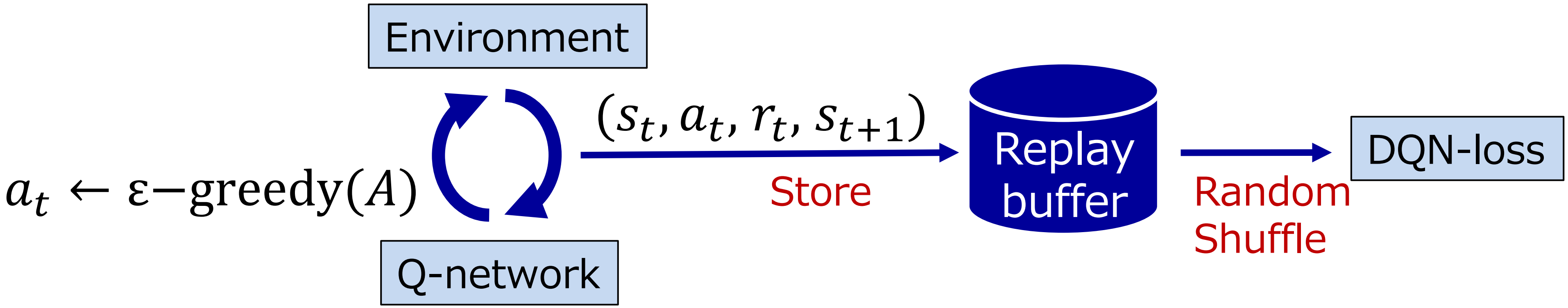
Loss function

Fixed

  - Game AI, robot control, communication control, ⋯
  - Various techniques are used for stability and convergence

Background: DQN techniques

- Experience replay
  - Remove correlations in the observation sequence
  - Reuse experiences to increase sampling efficiency

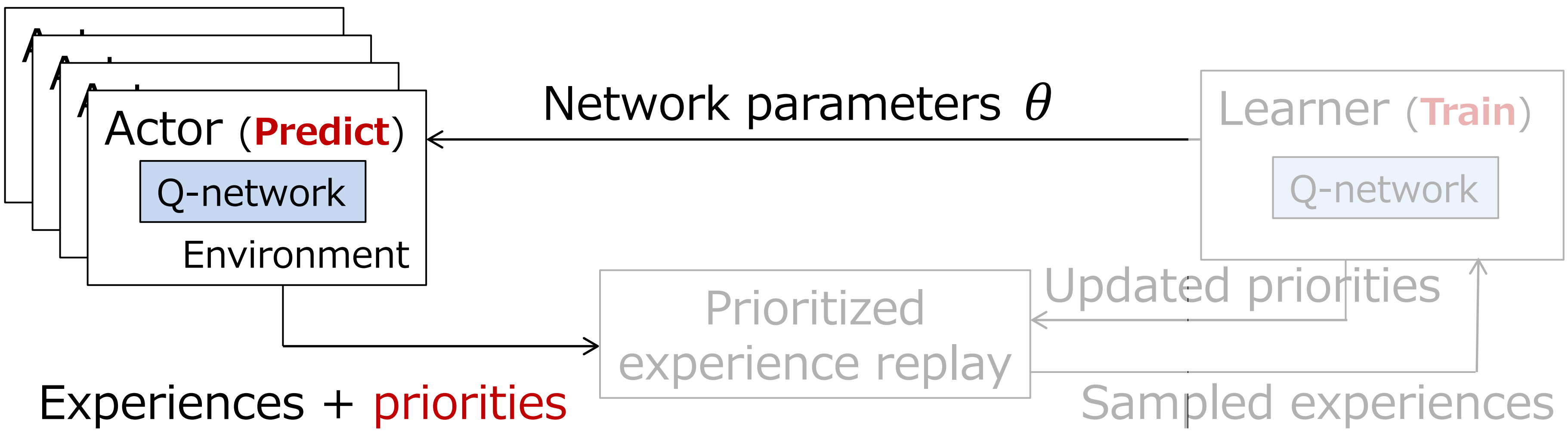


- Distributed deep reinforcement learning
  - Generate more experiences using many machines
  - Ape-X DQN [1] (Distributed prioritized experience replay)

[1] D. Horgan, et al., "Distributed Prioritized Experience Replay", ICLR'18.

Background: Ape-X

- Distributed prioritized experience replay [1]



1. Pull parameters θ

2. Prediction

Action:

$$a_t \leftarrow \varepsilon\text{-greedy}(A)$$

Next state & reward:

$$s_{t+1}, r_t \leftarrow \text{Environment}(a_t, s_t)$$

LocalBuffer.ADD(s_t, a_t, r_t, s_{t+1})

if LocalBuffer.Size $\geq$ BatchSize:

Priority:

$$|\delta_t| \leftarrow |r_t + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)|$$

3. Push experience

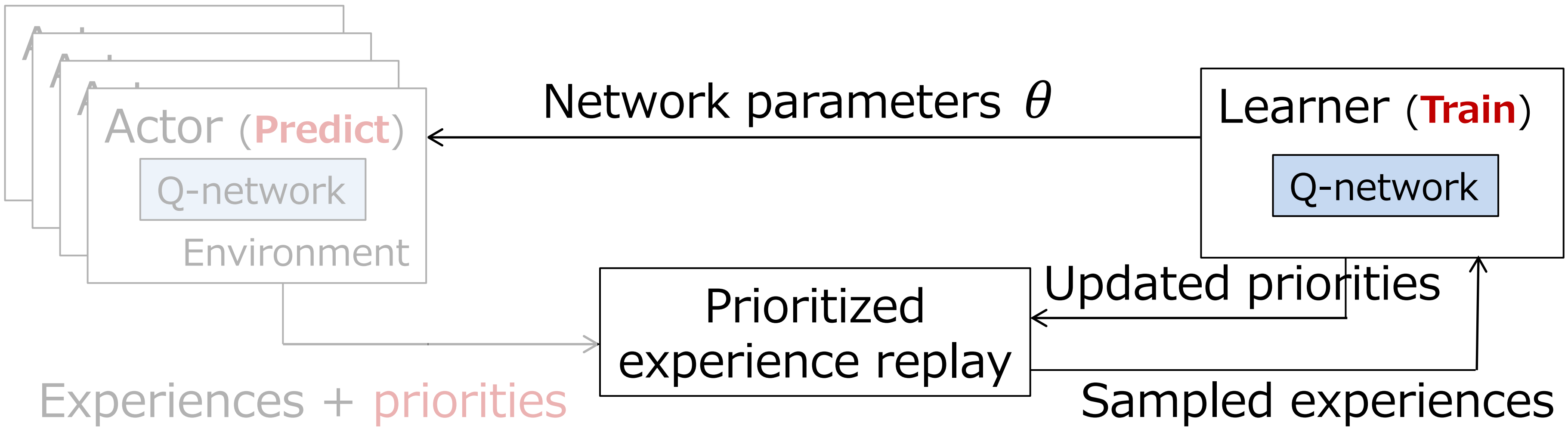
$$\{a_t, s_t, r_t, s_{t+1}, \delta_t\} \times \text{BatchSize}$$

a: Action, s: State, r: Reward,
δ: TD-error , γ: Discount rate

[1] D. Horgan, et al., "Distributed Prioritized Experience Replay", ICLR'18.

Background: Ape-X

- Distributed prioritized experience replay [1]



1. Experience sampling

$$\{a_t, s_t, r_t, s_{t+1}, \delta_t\} \times \mathrm{BatchSize}$$

2. Batch training

$$Q(s_t, a_t; \theta) \leftarrow r_t + \gamma \max_{a \in A} Q(s_{t+1}, a; \theta)$$

3. Update priorities

4. Set parameters $\boldsymbol{\theta}$

a: Action, s: State, r: Reward,
δ: TD-error , γ: Discount rate

[1] D. Horgan, et al., "Distributed Prioritized Experience Replay", ICLR'18.

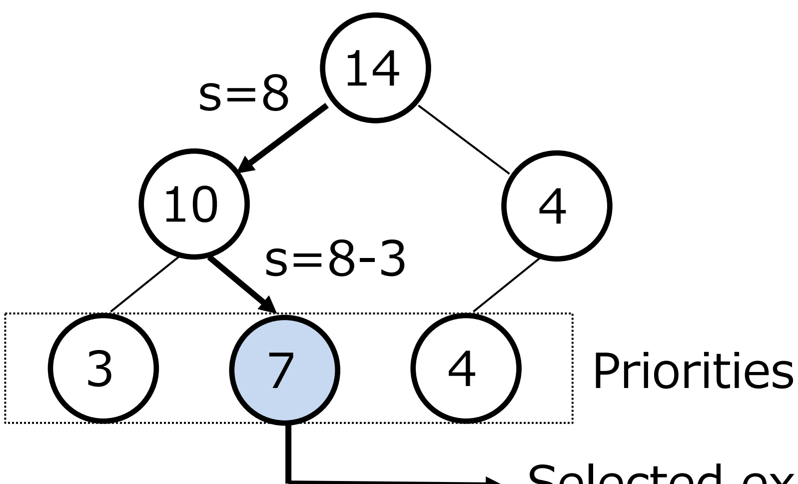
Background: Prioritized exp. replay

- Stochastic sampling [2]
  - Priority: $p_t = |\delta_t| = |r_t + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)|$
  - Sampling probability for experience ***i***

$$P_i = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}} \qquad (p_i \neq 0)$$

How much prioritization is used

- Sum-tree structure
  - Add/Sample experience: $O(\log N)$



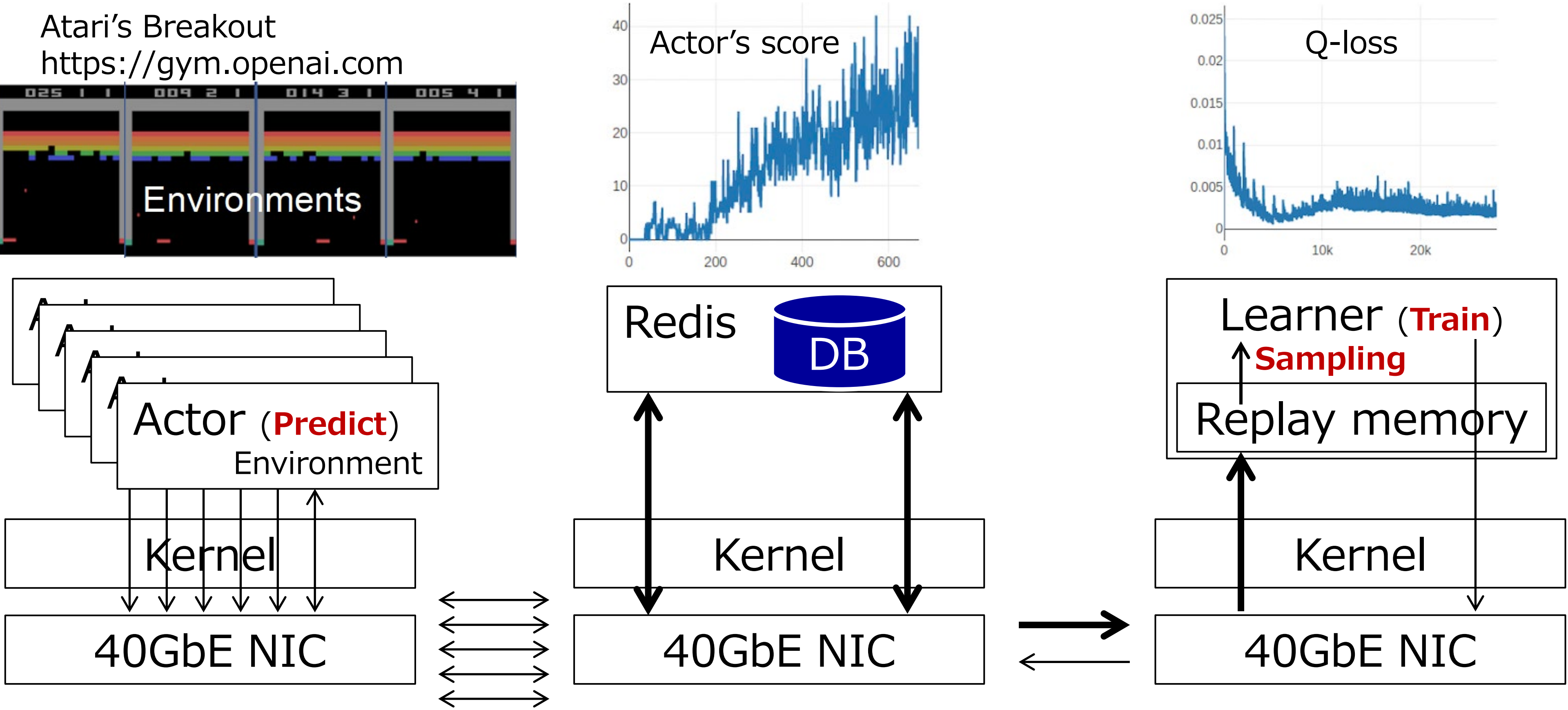

```
Require: 0 <= s (random number) <= Σ_k p_k
Require: n : root

function Sampling(n, s)
    if n is leaf_node then return n
    if n.left.val >= s then
        return Sampling(n.left, s)
    else
        return Sampling(n.right, s - n.left.val)
```

Selected experience is used for training

[2] T. Schaul, et al., "Prioritized Experience Replay", ICLR'16.

Baseline: Our DRL implementation

- Distributed deep reinforcement learning (DRL)
  - Actors, shared memory (Redis), and learner



1. **Pull** parameters (Dueling network architecture [3] 13MB)
2. Prediction
3. **Push** experiences (43MB)

1. **Pull** experiences (43MB*N)
2. Sampling + Training
3. **Push** parameters (13MB)

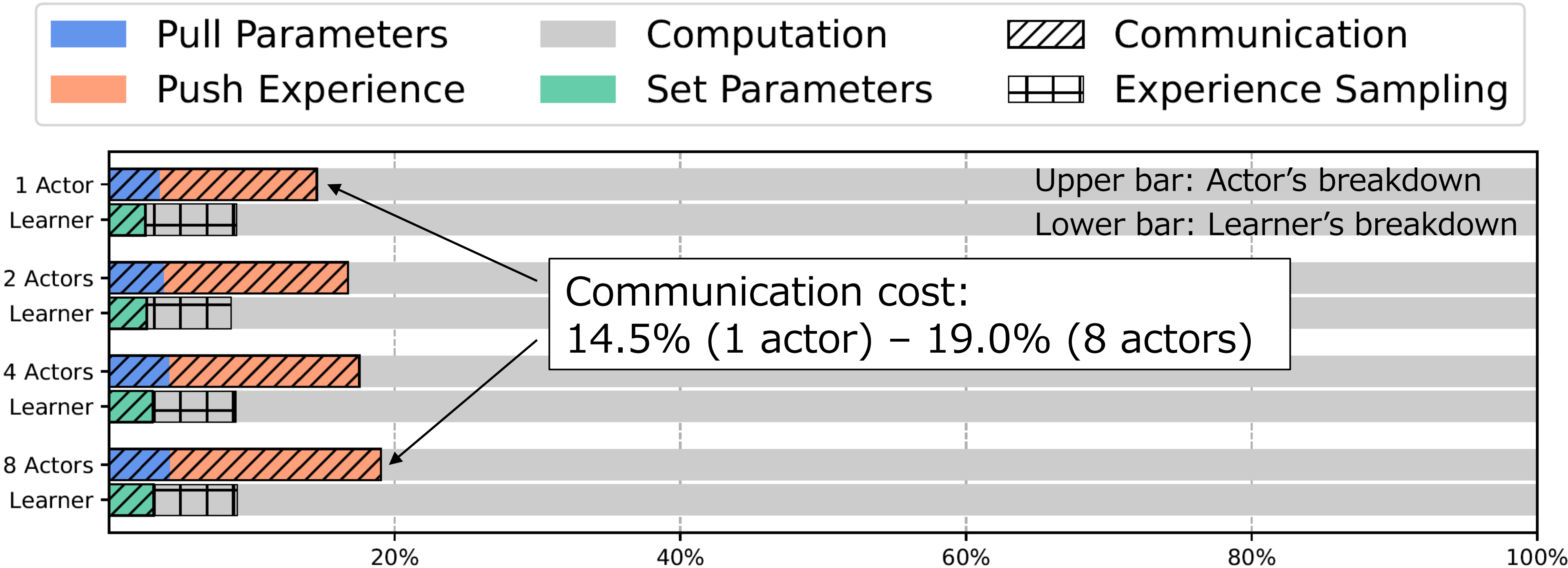
[3] Z. Wang, et al., "Dueling Network Architectures for Deep Reinforcement Learning", ICML'16.

Baseline: Our DRL implementation

- Distributed deep reinforcement learning (DRL)

Baseline: Execution time breakdown

- Distributed deep reinforcement learning (DRL)

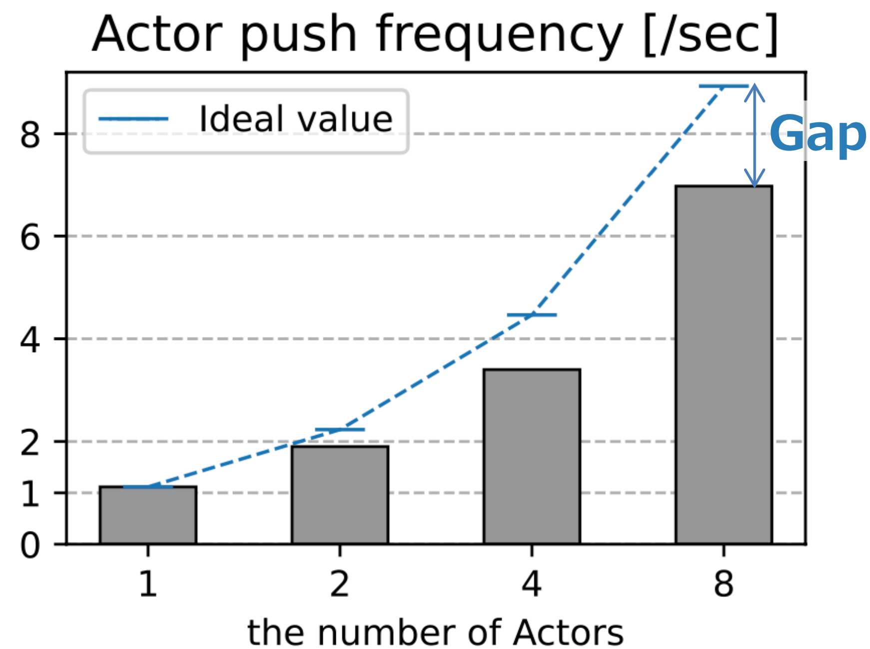


	Actors machine	Shared/Replay memory machine	Learner machine
OS	Ubuntu 20.04	Ubuntu 20.04	Ubuntu 20.04
CPU	Intel Xeon E5-2637 v3 @3.5GHz	Intel Xeon CPU E5-2637 v3 @3.50GHz	Intel Xeon CPU E5-1620 v2 @3.50GHz
Memory	128 GB	512 GB	128 GB
GPU	GeForce RTX 3080 ×1	–	GeForce GTX 1080Ti ×2
CUDA/PyTorch	11.3 / 1.8.0+cu111	11.3 / 1.8.0+cu111	11.3 / 1.8.0+cu111
NIC	Intel Ethernet CNA XL710-QDA2	Intel Ethernet CNA XL710-QDA2	Intel Ethernet CNA XL710-QDA2
DPDK	–	20.11.0	–
Redis	–	5.0.5	–
Network		Mellanox 40G Switch SX1012	

Baseline: Shared mem throughput

- Distributed deep reinforcement learning (DRL)

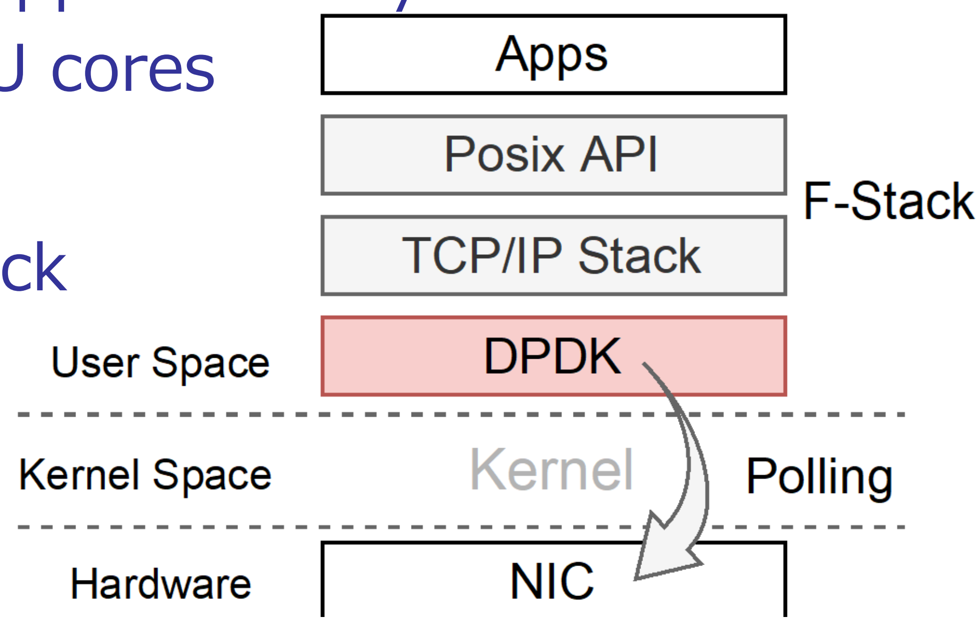
- Actor push frequency
  - Increase as # of actors
- Increase of # of actors
  - Network traffic increases
  - Gap between ideal and actual frequency increases



	Actors machine	Shared/Replay memory machine	Learner machine
OS	Ubuntu 20.04	Ubuntu 20.04	Ubuntu 20.04
CPU	Intel Xeon E5-2637 v3 @3.5GHz	Intel Xeon CPU E5-2637 v3 @3.50GHz	Intel Xeon CPU E5-1620 v2 @3.50GHz
Memory	128 GB	512 GB	128 GB
GPU	GeForce RTX 3080 $\times 1$	–	GeForce GTX 1080Ti $\times 2$
CUDA/PyTorch	11.3 / 1.8.0+cu111	11.3 / 1.8.0+cu111	11.3 / 1.8.0+cu111
NIC	Intel Ethernet CNA XL710-QDA2	Intel Ethernet CNA XL710-QDA2	Intel Ethernet CNA XL710-QDA2
DPDK	–	20.11.0	–
Redis	–	5.0.5	–
Network		Mellanox 40G Switch SX1012	

Network optimizations on DRL

- Our approach
  - Accelerating **shared memory** & **experience replay** using DPDK
- DPDK (Data Plane Development Kit)
  - Network processing at application layer
  - Polling by dedicated CPU cores
- F-stack
  - Light-weight TCP/IP stack for DPDK applications

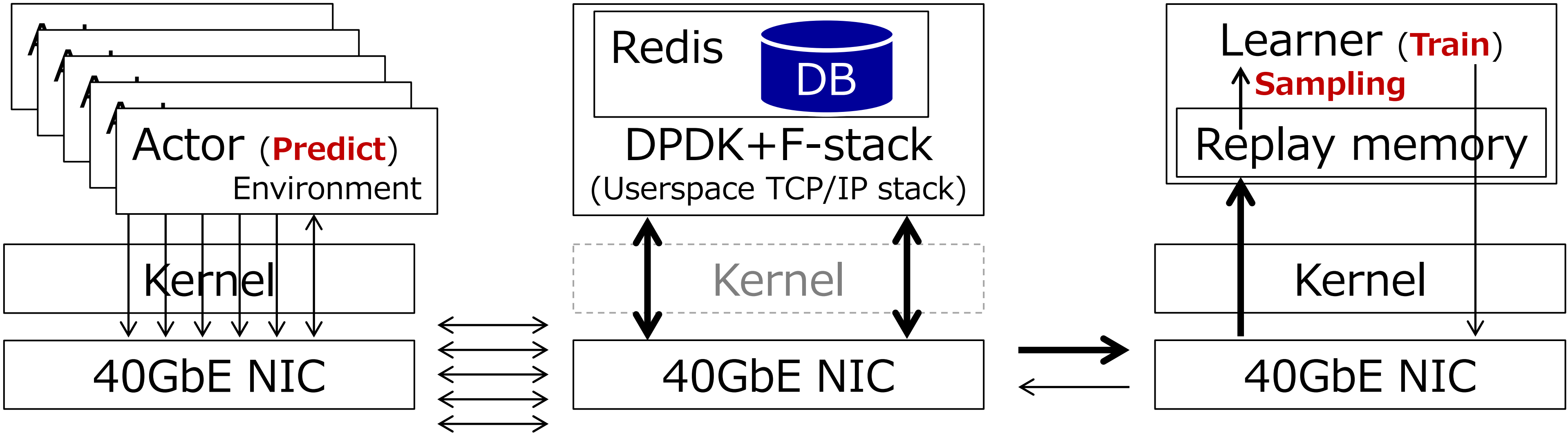


- Related work [4]
  - Accelerating gradient aggregation of distributed deep learning at DPDK-based network switch

[4] M. Furukawa, et al., "An In-Network Parameter Aggregation using DPDK for Multi-GPU Deep Learning", CANDAR'20.

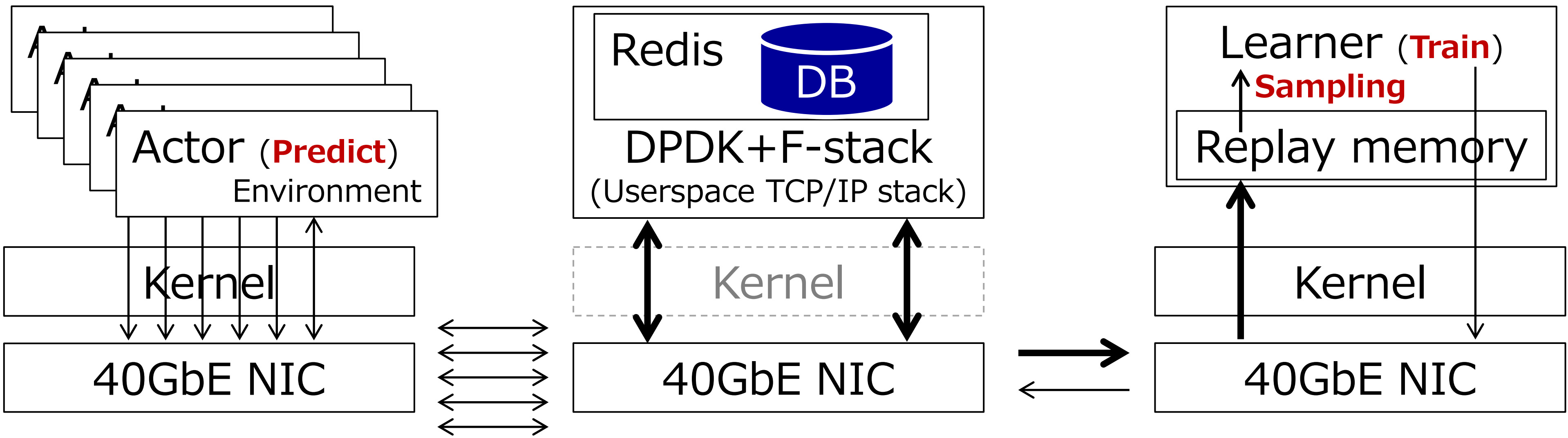
Optimization 1: Shared memory

- In-network shared memory (Redis) by DPDK
  - Low-latency shared memory by DPDK and F-stack



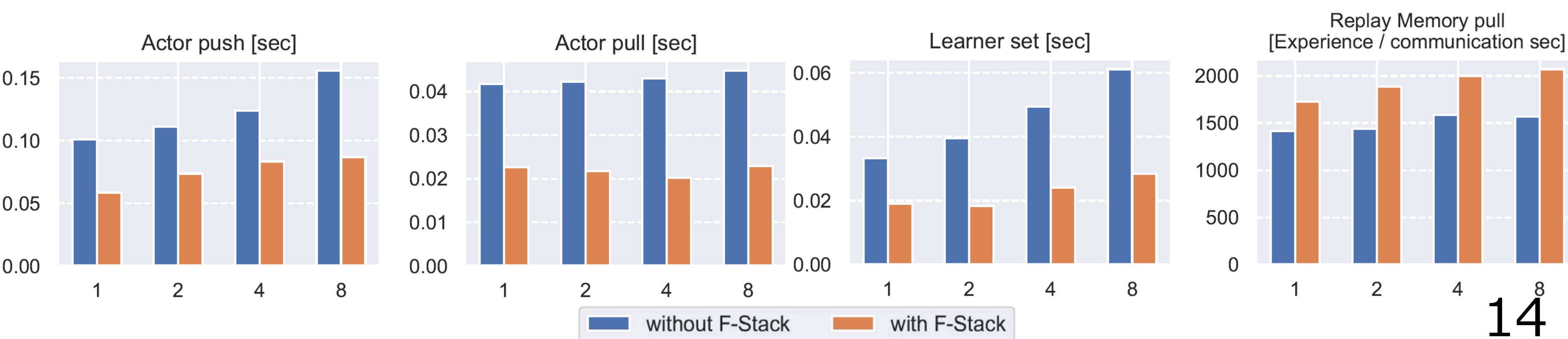
Optimization 1: Shared memory

- In-network shared memory (Redis) by DPDK
  - Low-latency shared memory by DPDK and F-stack



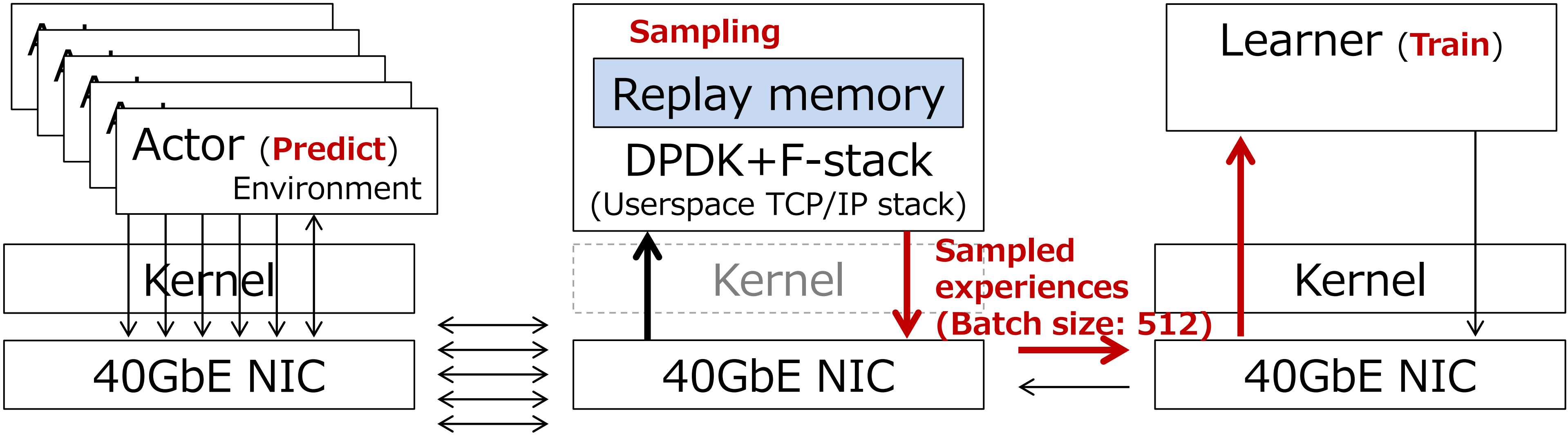
Redis access latency:
32.7% to 58.9% decrease

Replay memory pull throughput:
21.9% to 31.9% increase



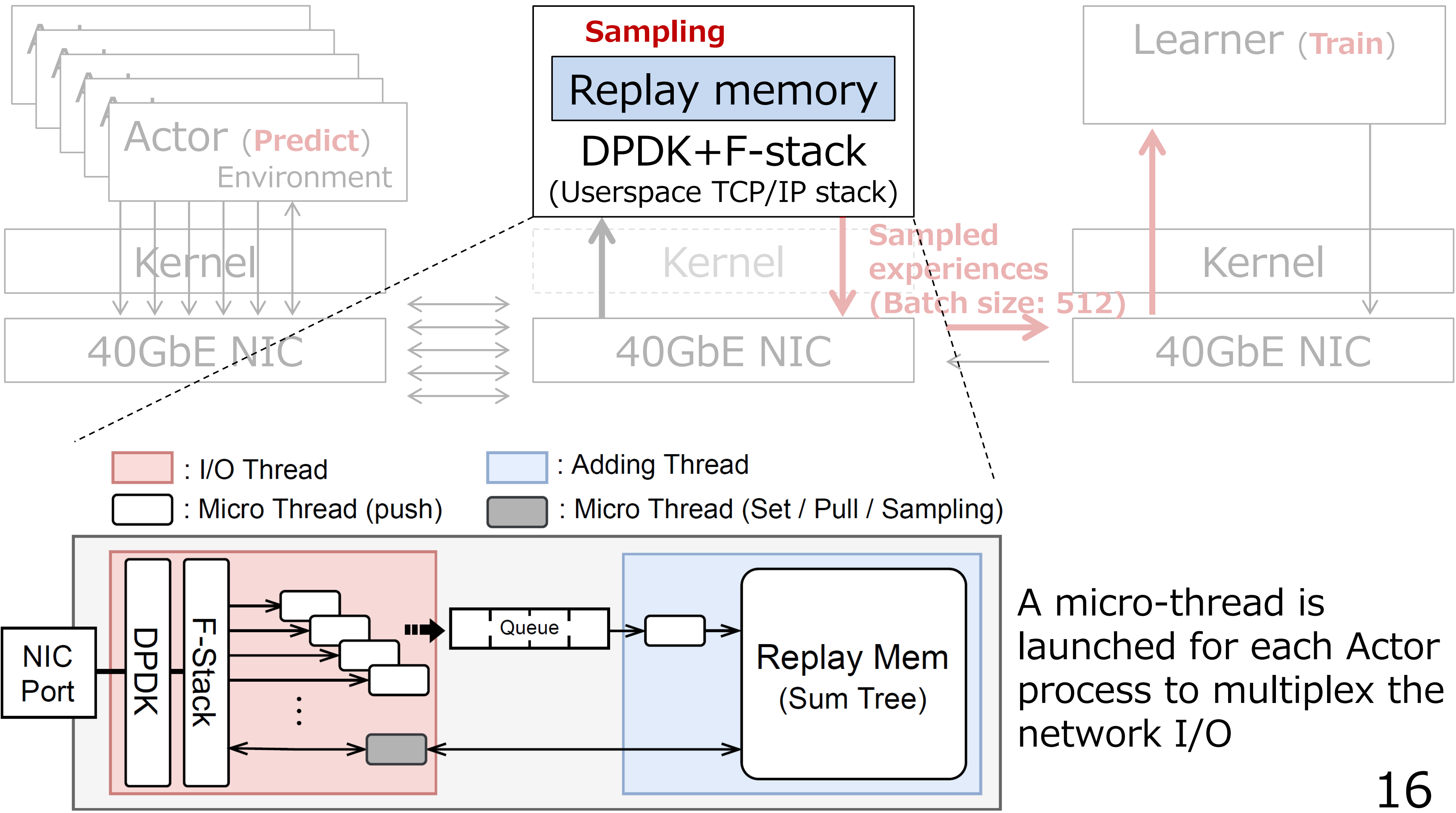
Optimization 2: Experience replay

- In-network experience sampling by DPDK
  - Moved from leaner machine to in-network switch



Optimization 2: Experience replay

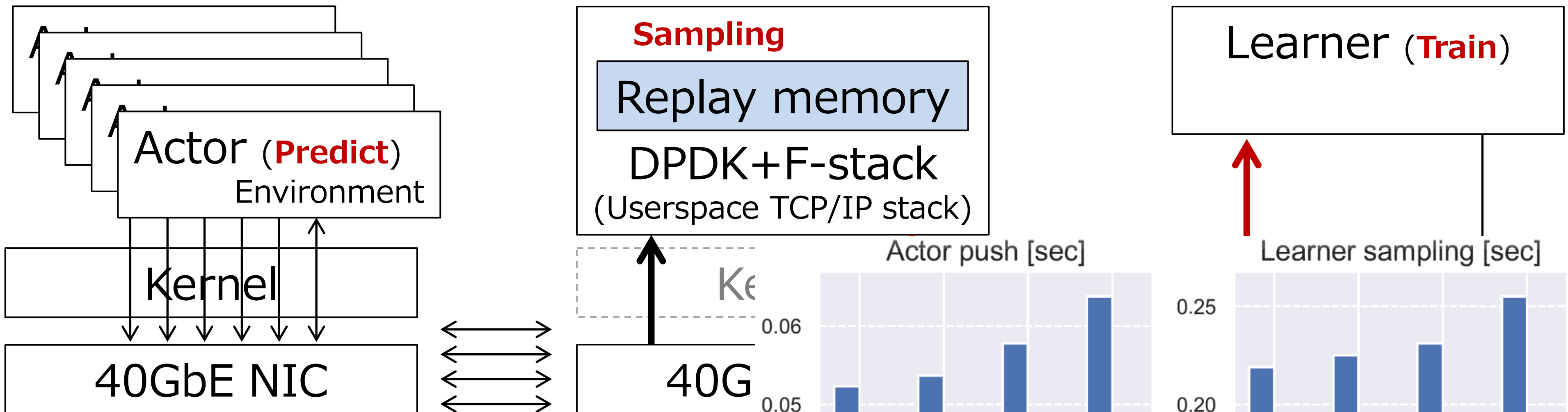
- In-network experience sampling by DPDK
  - Moved from leaner machine to in-network switch

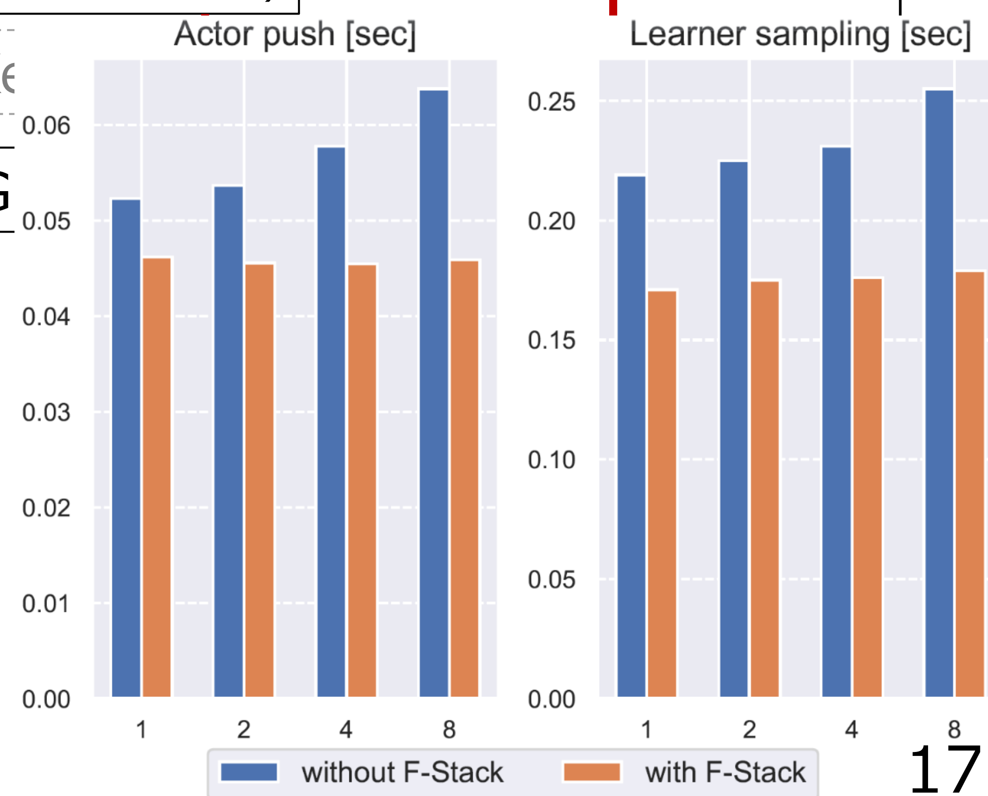


A micro-thread is launched for each Actor process to multiplex the network I/O

Optimization 2: Experience replay

- In-network experience sampling by DPDK
  - Moved from leaner machine to in-network switch



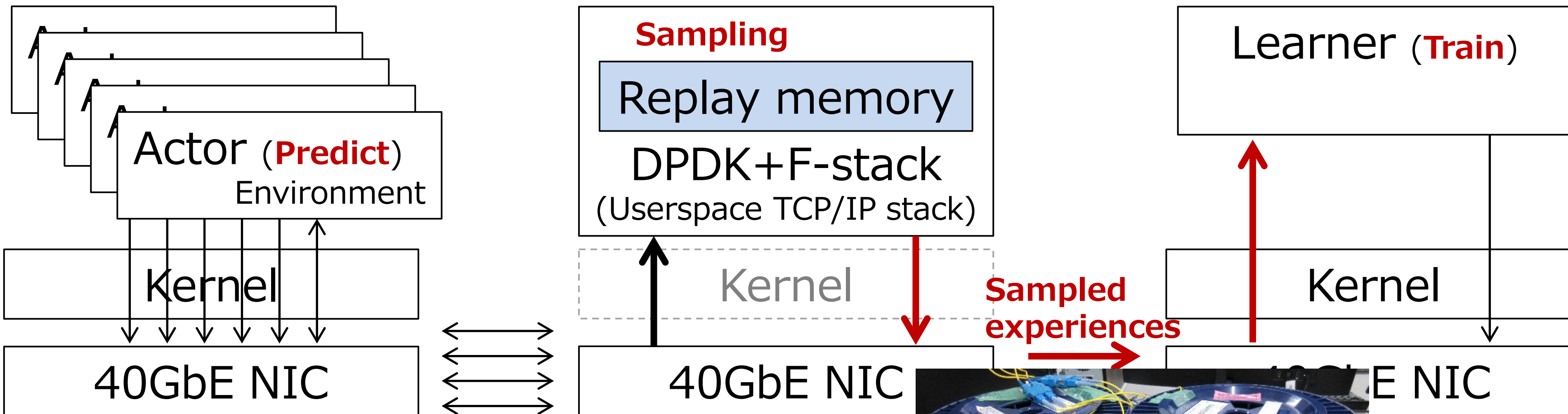


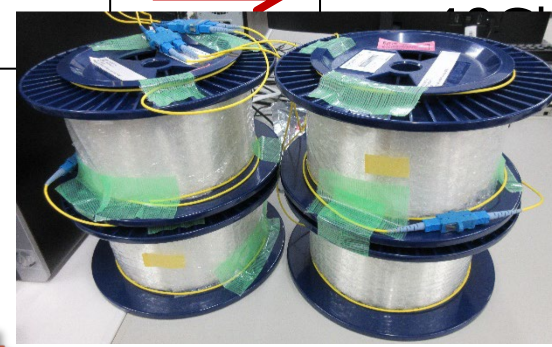
Replay memory access latency is reduced by 11.7%-28.1%

Sampling latency of learner is reduced by 21.9%-29.8%

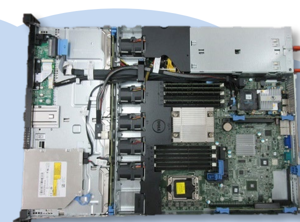
Future work: Deployed in edge server

- Experience sampling in edge server
  - Actors located in edge, and learner located in cloud





10km optical cable



Edge

Cloud

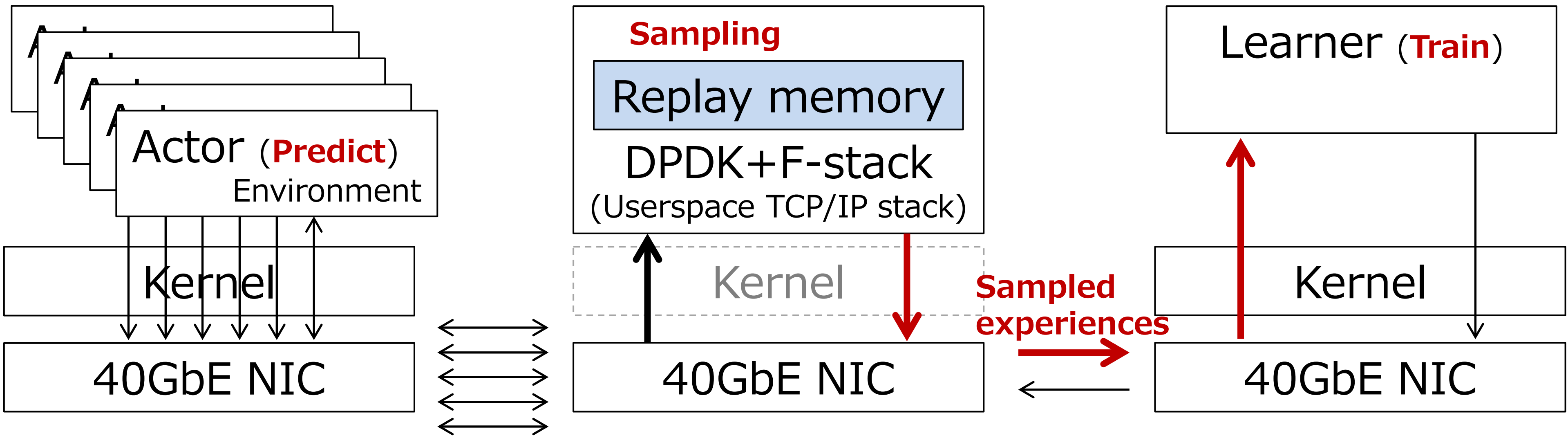
Edge computers

Edge server

Datacenter

Summary: Optimizations in DRL

- Communication cost reduction by DPDK+F-stack
  - Optimization 1: In-network shared memory (Redis)
  - Optimization 2: In-network experience sampling



- Future work: Deployment in edge-cloud system