Lecture 8: Scaling Reinforcement Learning and Gradient Boosting

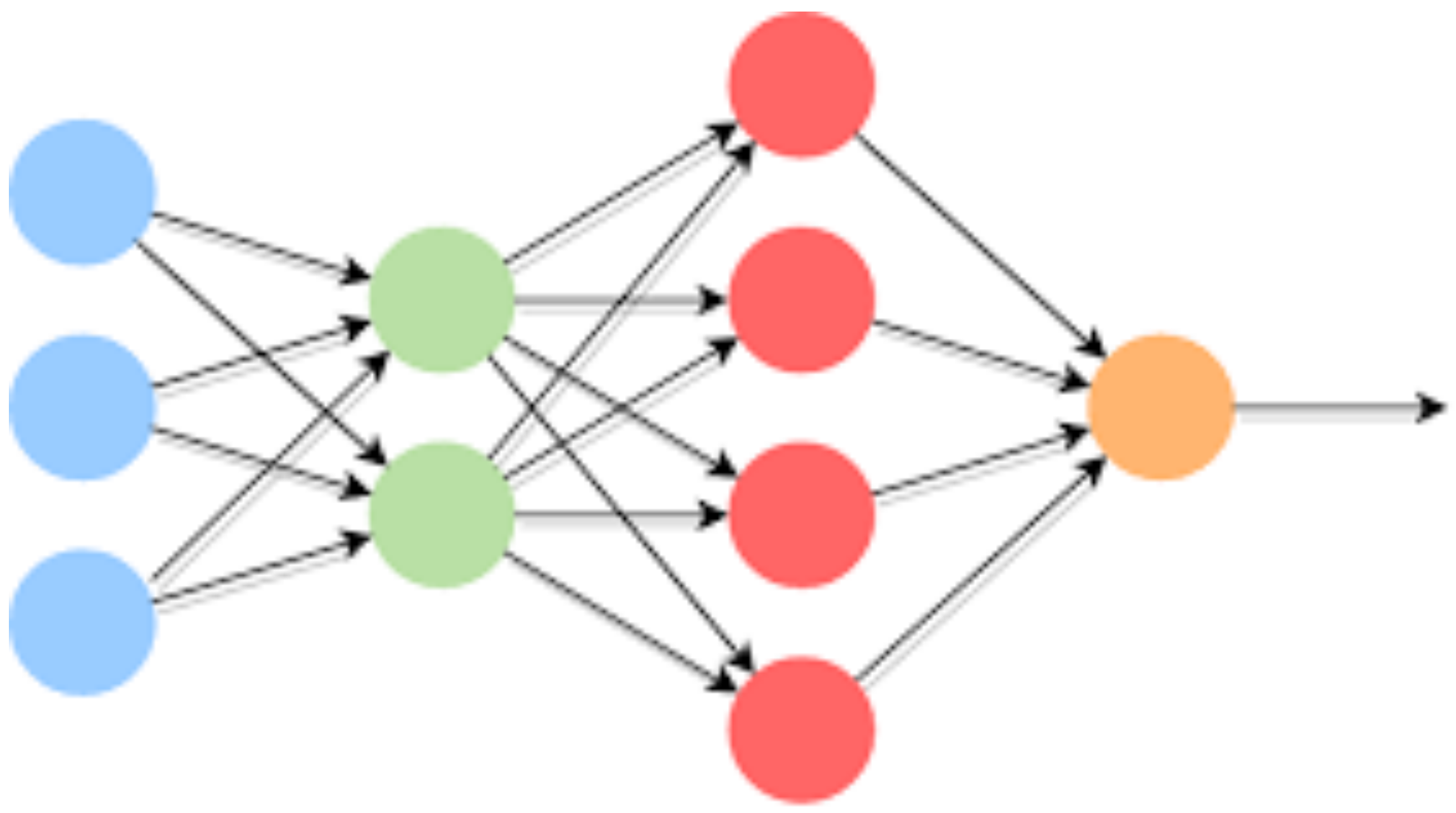
CS 256: Systems and Machine Learning

Sangeetha Abdu Jyothi

UCIRVINE

Parts of this lecture were adapted from talks on XGBoost and DeepRL course at Berkeley

Previous Lectures



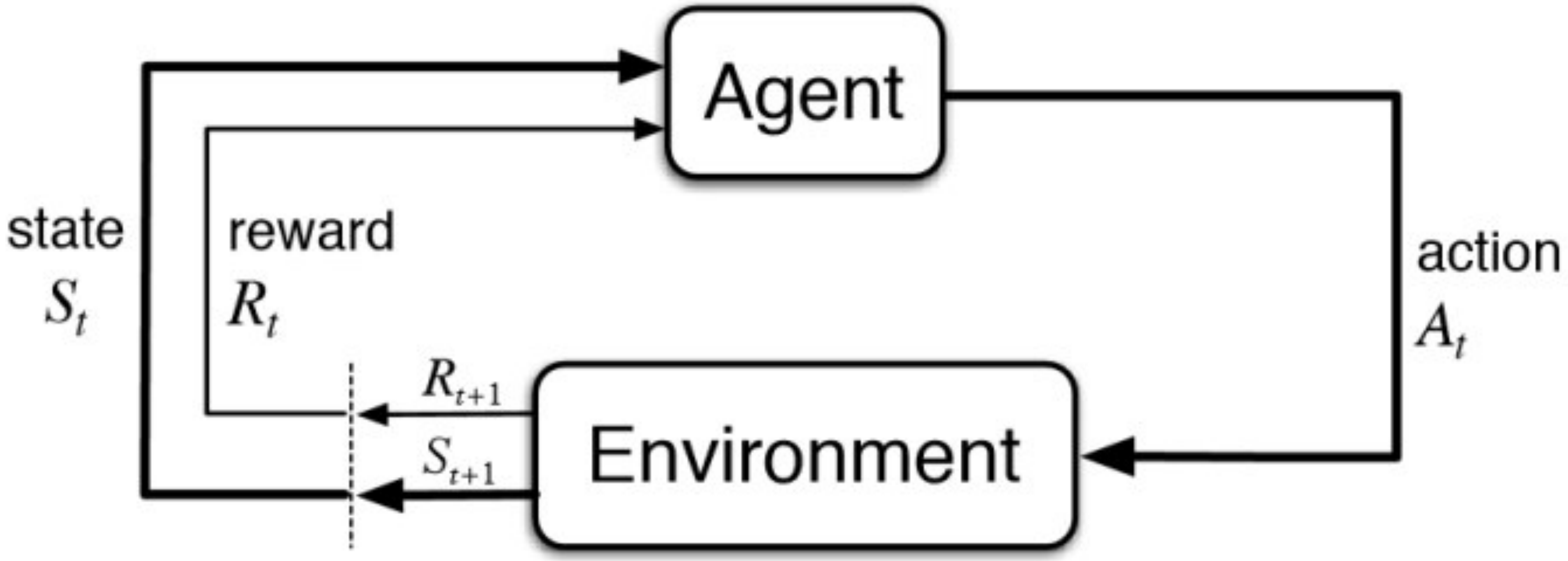
DNNs

Supervised Learning

Today's Lecture

- Distributed Reinforcement Learning
- XGBoost: A Scalable Tree Boosting System

Reinforcement Learning



Not easy to parallelize

Policy-based Methods

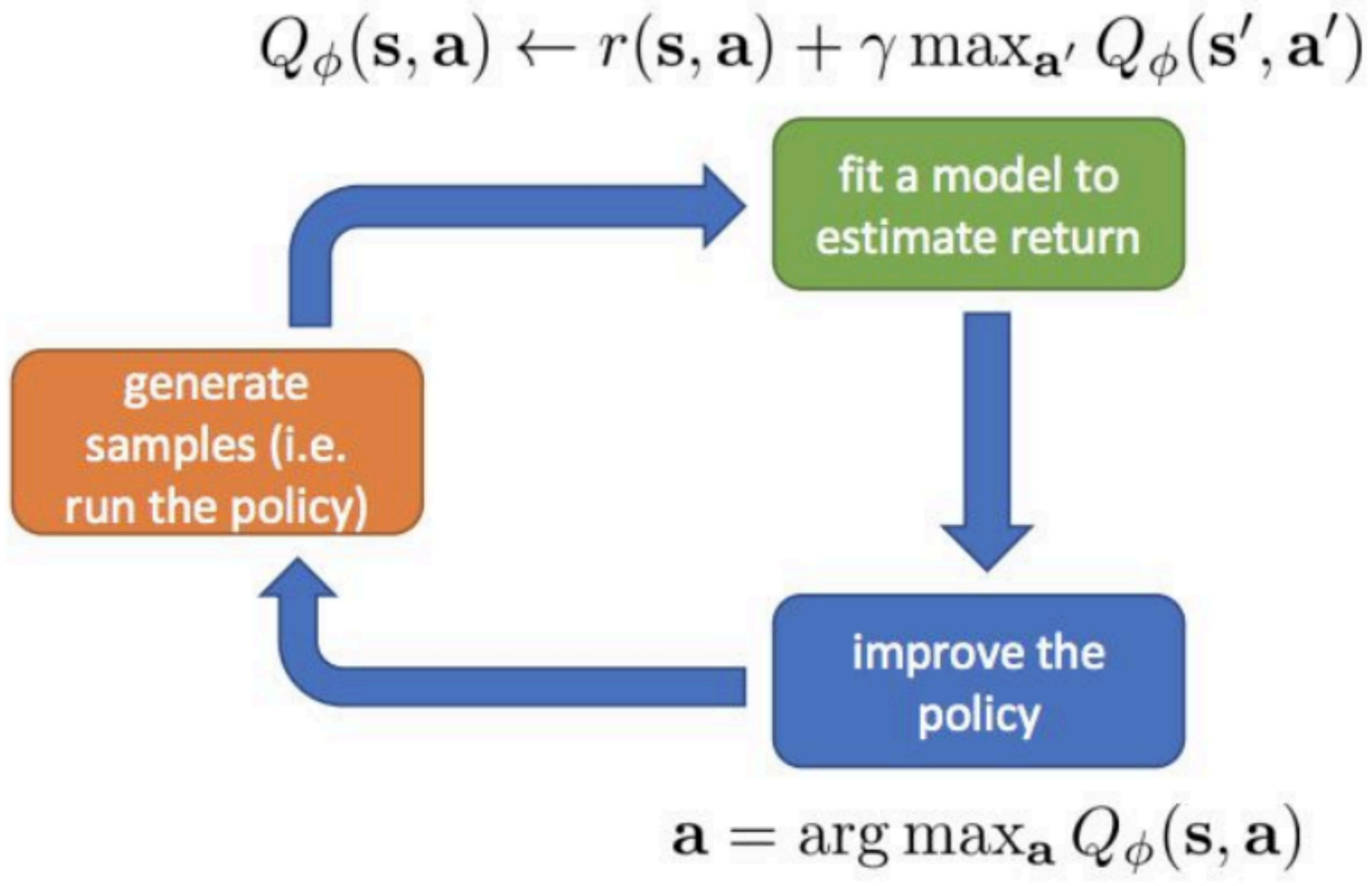
- Parametrize policy with theta and update theta with gradient descent

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

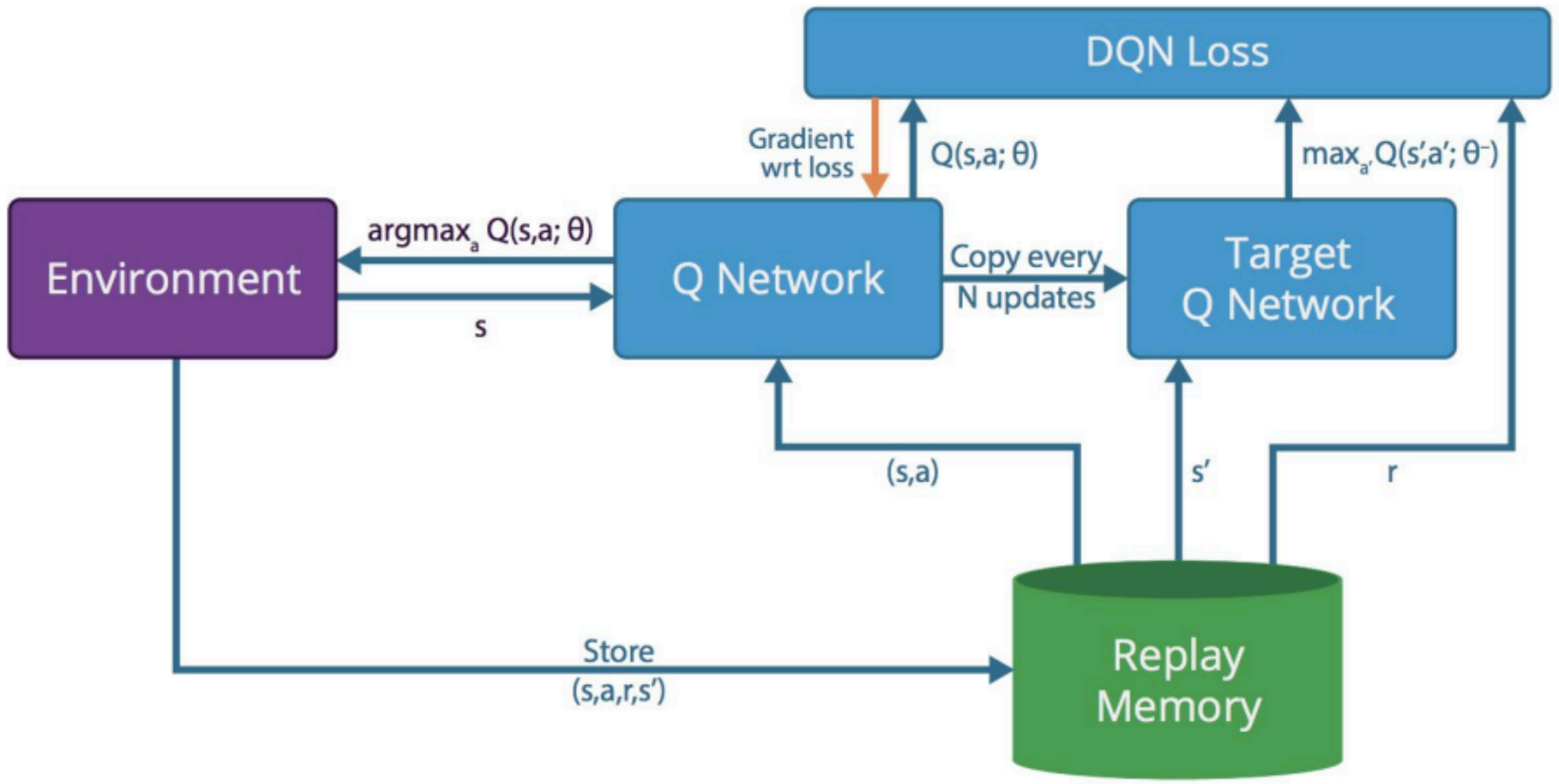
Value Based Methods

- Don't learn policy explicitly
- Learn Q-function
    - Deep RL: Train neural network to approximate Q-function



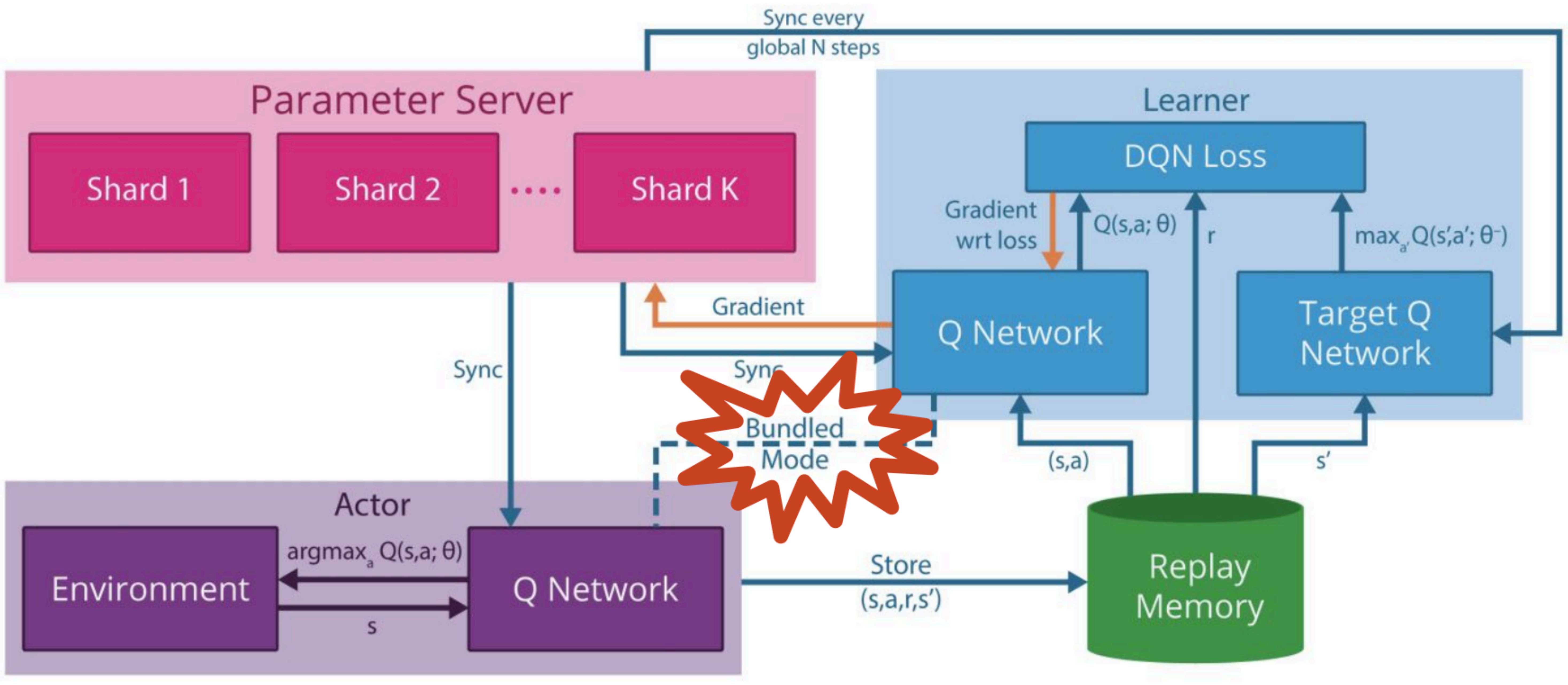
DQN (2013/2015)

- Experience Replay
  - Stores experiences including state transitions, rewards and actions
  - Reuses past transitions to avoid catastrophic forgetting
- Target Network
  - Unstable target function makes training difficult
  - Target Network technique fixes parameters of target function and replaces them with the latest network periodically (e.g., every thousands steps)
- Clipping Rewards
  - Large rewards make training unstable

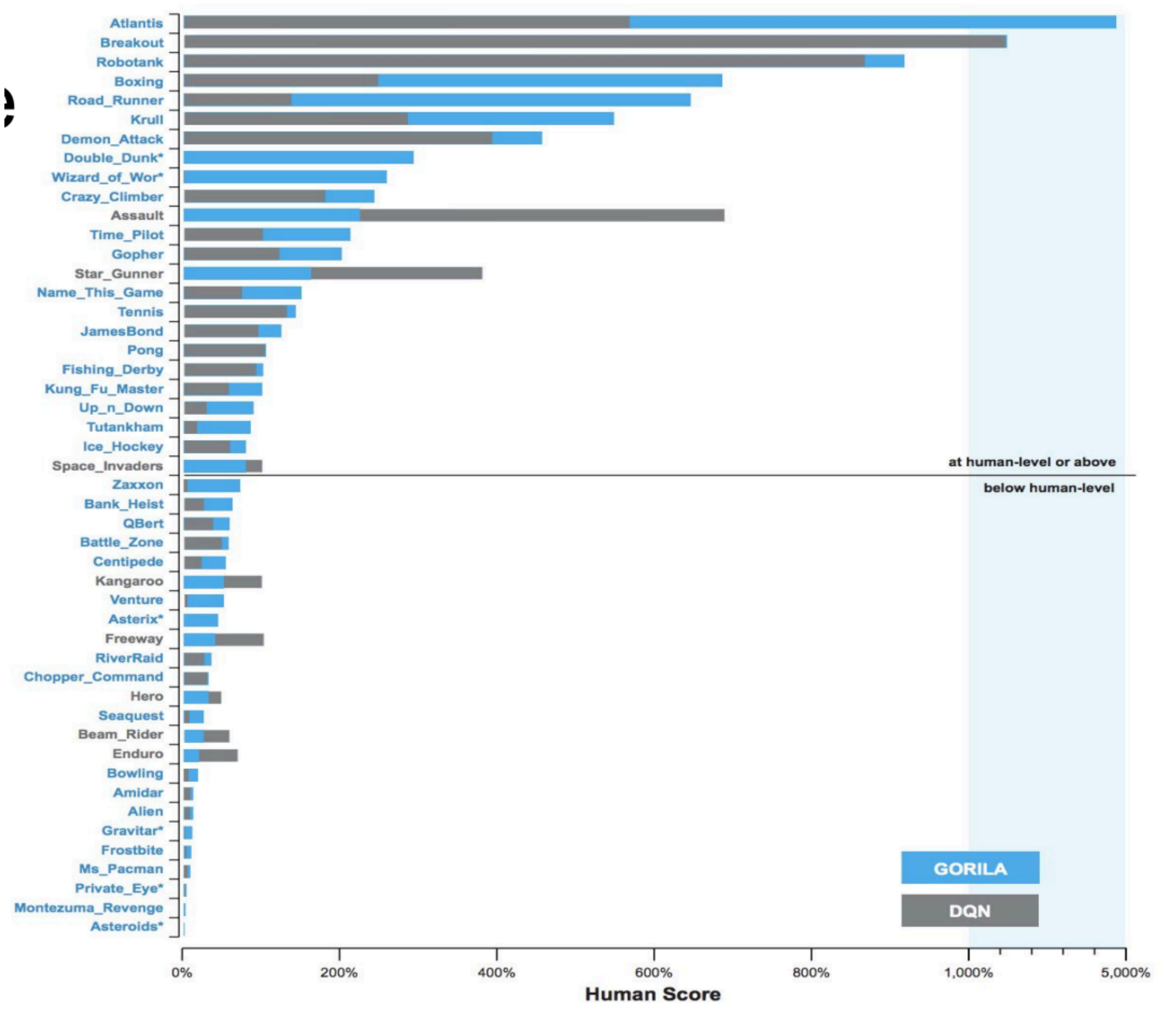


General Reinforcement Learning Architecture (GORILA) (2015)

- Asynchronous training of RL agents in a distributed setting
- Components
    - Actor
    - Replay memory
    - Learner
- Updated parameters sent to actors periodically



GORILLA performance



Asynchronous Actor-Learners

- Rather than separate machines coordinated by a parameter server…
- Multiple CPU threads on single machine coordinated by OS
- Removes communication costs
- Actors walk through environment and send updates to learners
- Learners use observations to compute gradients
- Advantages
    - Multiple actors running in parallel explores different parts of environment and decorrelates observations
    - Different exploration policies in each actor-learner
    - Can avoid instability due to data correlation w/o using replay buffer
    - Reduction in training time roughly linear in the number of parallel actor-learners

Variants

- Asynchronous 1-step Q Learning
- Asynchronous 1-step Sarsa
- Asynchronous n-step Q Learning
- Asynchronous Advantage Actor Critic (A3C)

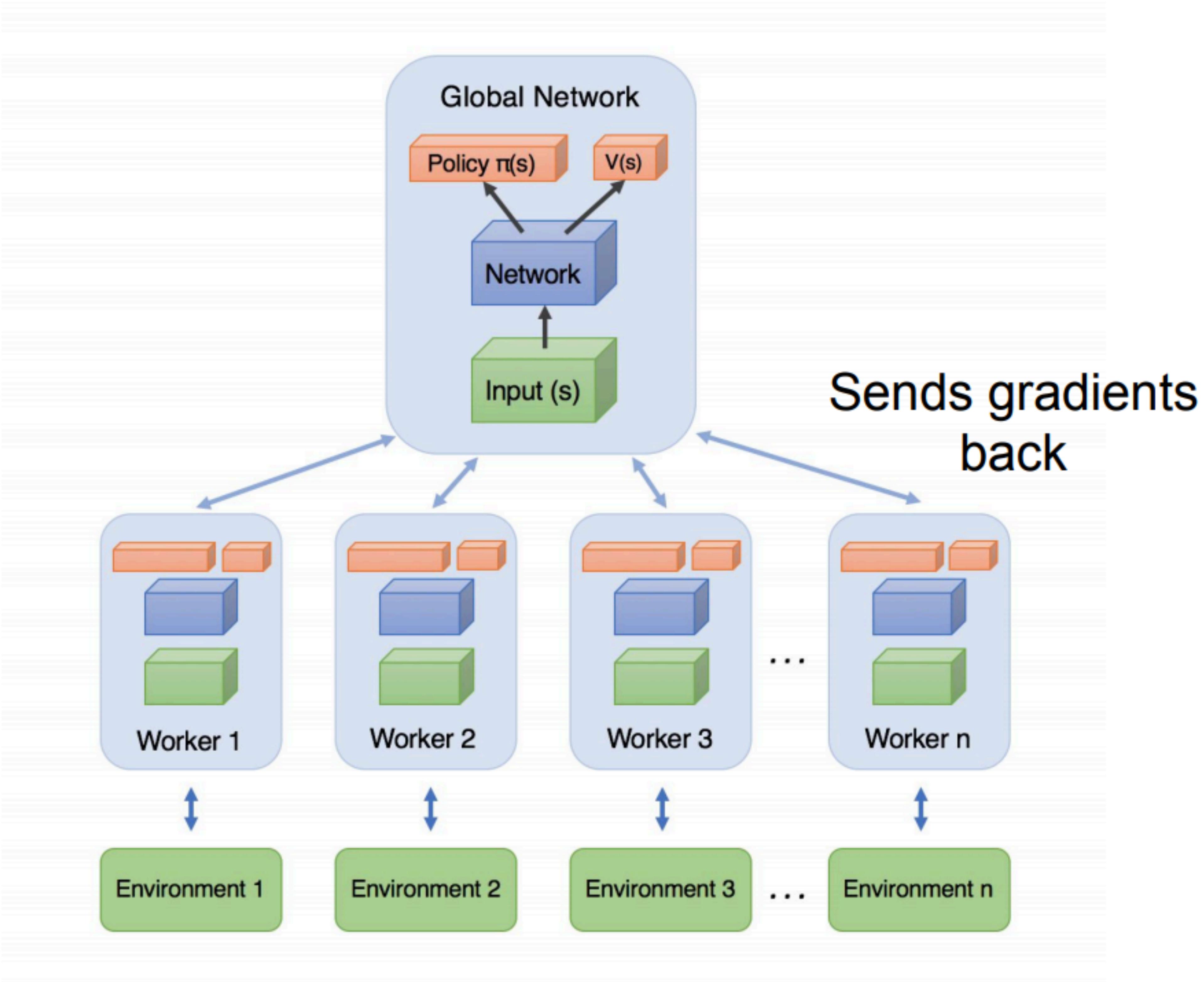
Asynchronous Advantage Actor Critic (A3C) (2016)

```
# Each worker:

while True:
    sync_weights_from_master()

    for i in range(5):
        collect sample from env

    grad = compute_grad(samples)
    async_send_grad_to_master()
```



Each has different exploration -> more diverse samples!

A3C Performance

Changes to GORILA:

1. **Faster updates**
2. **Removes** the replay buffer
3. **Moves** to Actor-Critic (from Q learning)

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%

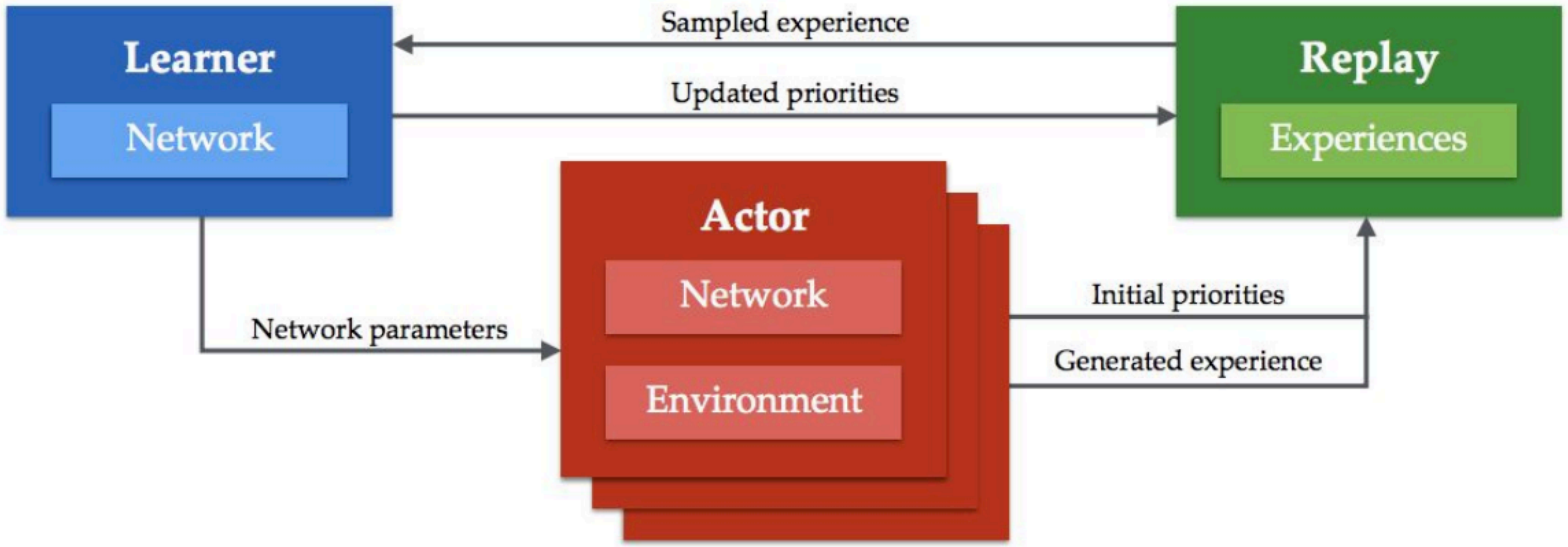
Table 1. Mean and median human-normalized scores on 57 Atari games using the human starts evaluation metric. Supplementary

Distributed Prioritized Experience Replay (Ape-X) (2018)

A3C doesn't scale very well…

Ape-X:

1. Distributed DQN/DDPG
2. Reintroduces replay
3. **Distributed Prioritization**



APEX performance

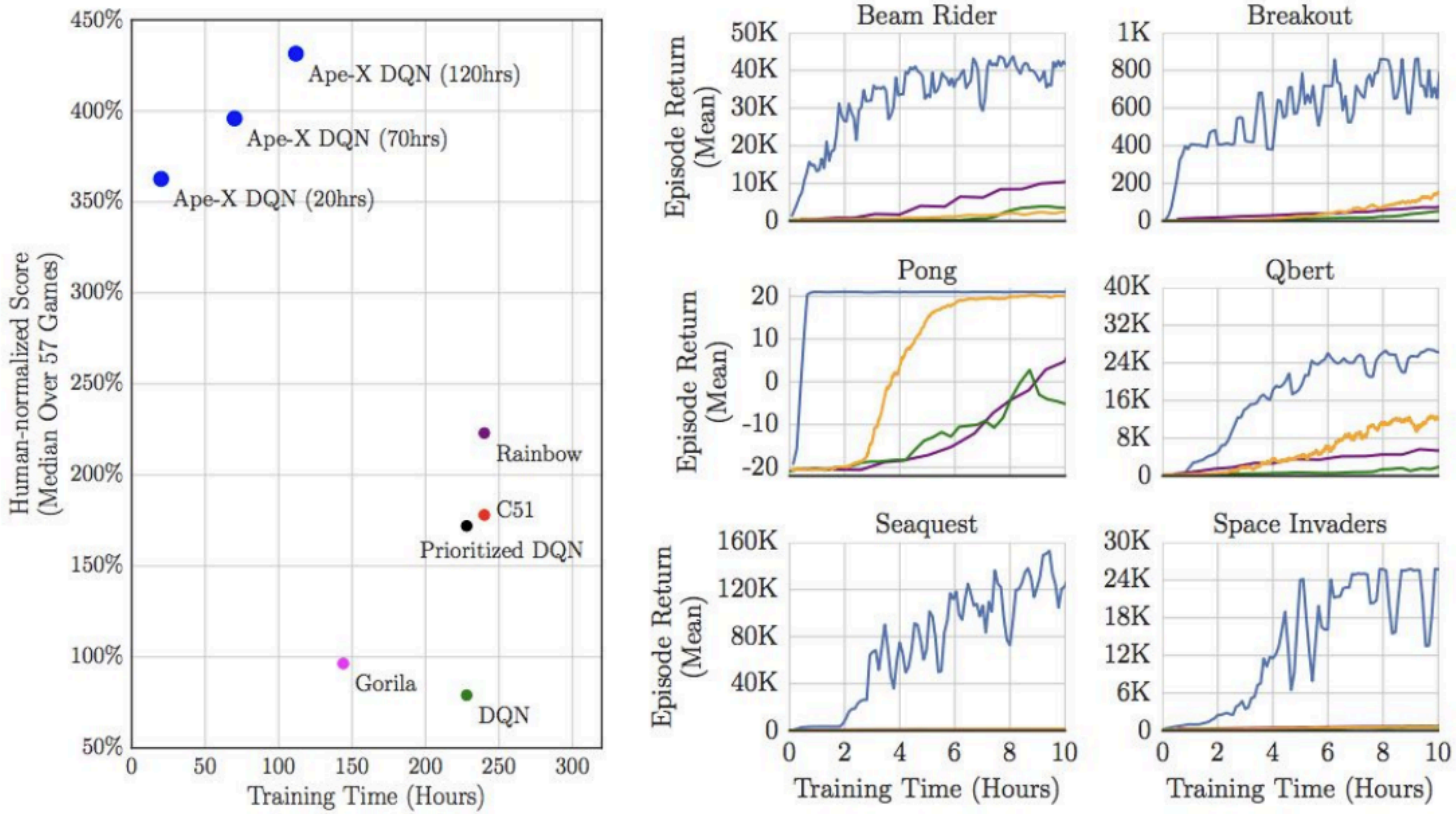


Figure 2: Left: Atari results aggregated across 57 games, evaluated from random no-op starts. Right: Atari training curves for selected games, against baselines. Blue: Ape-X DQN with 360 actors; Orange: A3C; Purple: Rainbow; Green: DQN. See appendix for longer runs over all games.

Ray RLlib

Ray is an open-source unified compute framework that makes it easy to scale AI and Python workloads

RLlib algorithms

RLlib Algorithms

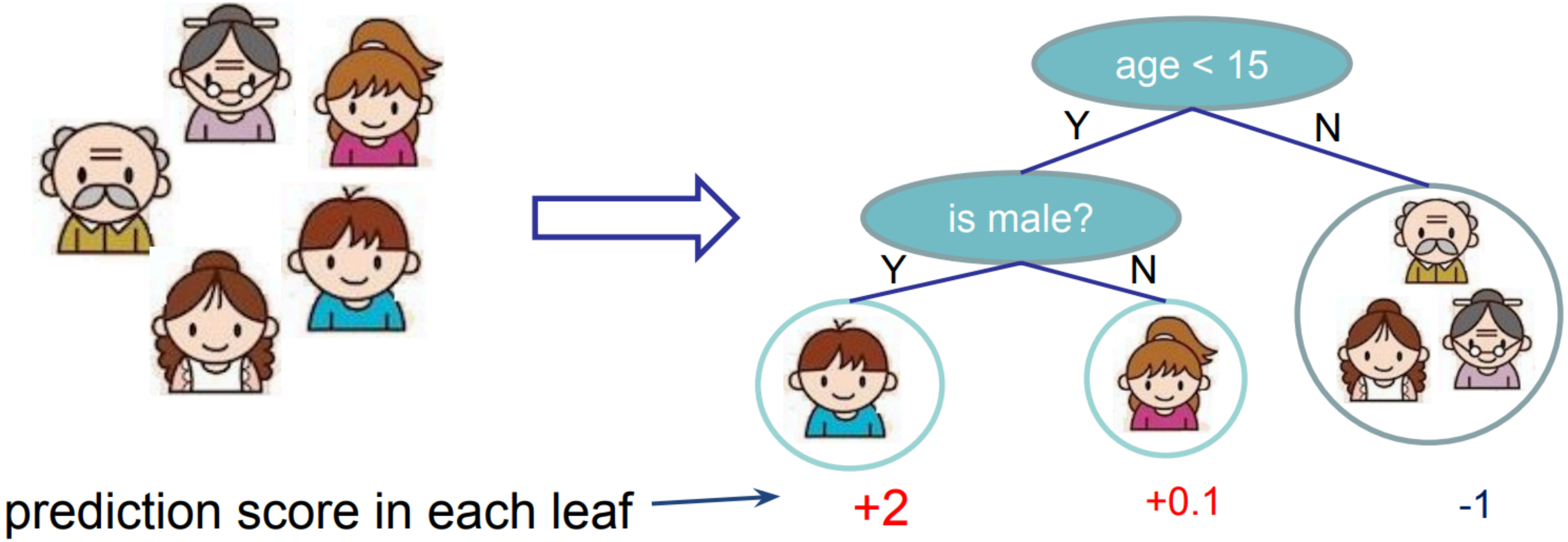
- High-throughput architectures
  - Distributed Prioritized Experience Replay (Ape-X)
  - Importance Weighted Actor-Learner Architecture (IMPALA)
  - Asynchronous Proximal Policy Optimization (APPO)
  - Decentralized Distributed Proximal Policy Optimization (DD-PPO)
- Gradient-based
  - Advantage Actor-Critic (A2C, A3C)
  - Deep Deterministic Policy Gradients (DDPG, TD3)
  - Deep Q Networks (DQN, Rainbow, Parametric DQN)
  - Policy Gradients
  - Proximal Policy Optimization (PPO)
  - Soft Actor Critic (SAC)
  - Slate Q-Learning (SlateQ)
- Derivative-free
  - Augmented Random Search (ARS)
  - Evolution Strategies
- Model-based / Meta-learning / Offline
  - Single-Player AlphaZero (contrib/AlphaZero)
  - Model-Agnostic Meta-Learning (MAML)
  - Model-Based Meta-Policy-Optimization (MBMPO)
  - Dreamer (DREAMER)
  - Conservative Q-Learning (CQL)
- Multi-agent
  - QMIX Monotonic Value Factorisation (QMIX, VDN, IQN)
  - Multi-Agent Deep Deterministic Policy Gradient (contrib/MADDPG)
- Offline
  - Advantage Re-Weighted Imitation Learning (MARWIL)
- Contextual bandits
  - Linear Upper Confidence Bound (contrib/LinUCB)
  - Linear Thompson Sampling (contrib/LinTS)
- Exploration-based plug-ins (can be combined with any algo)
  - Curiosity (ICM: Intrinsic Curiosity Module)

XGBoost

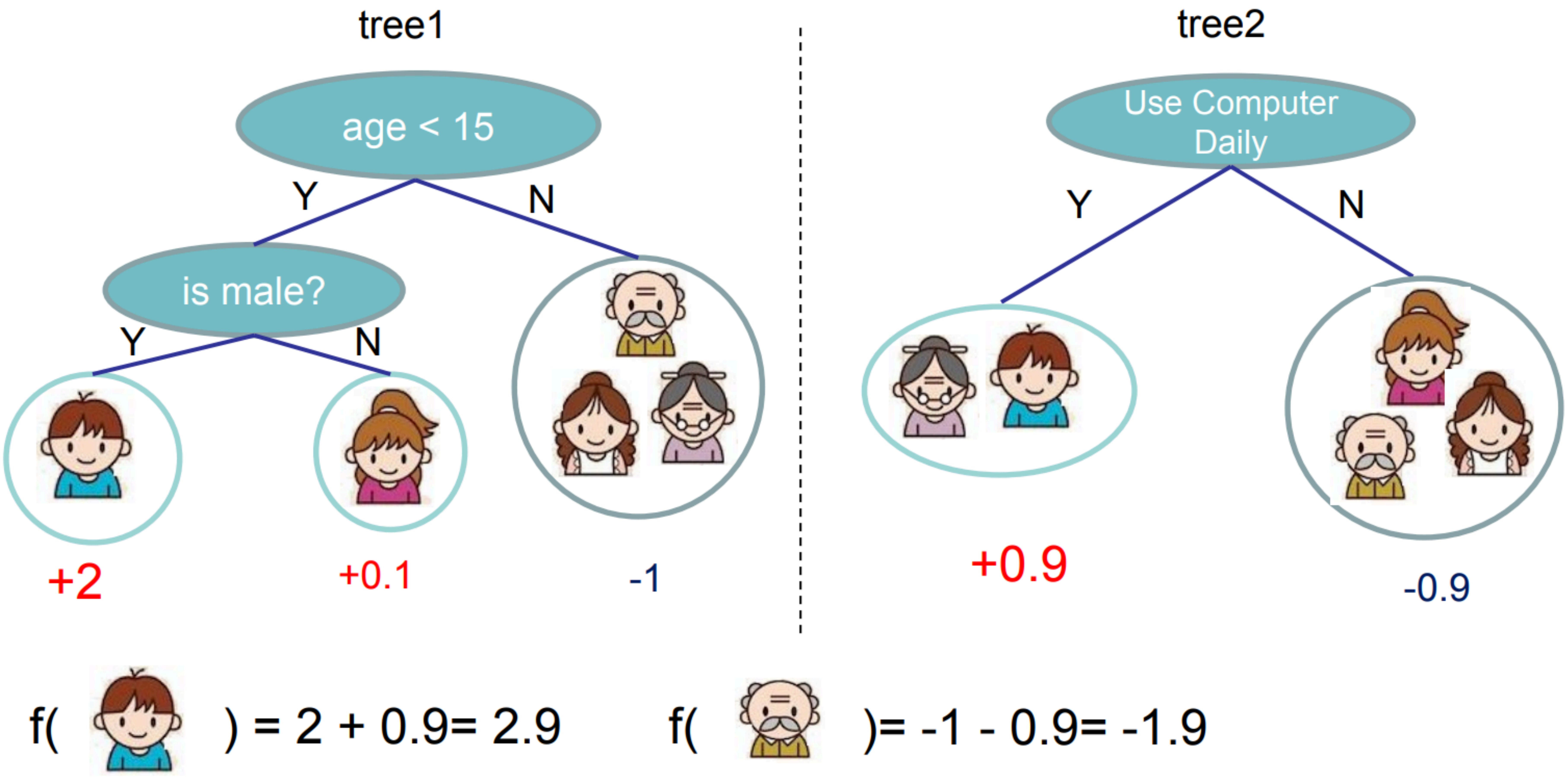
Regression Tree (CART)

Input: age, gender, occupation, …

Does the person like computer games



Tree Ensembles



Algorithms to learn Tree Ensembles

- Random Forest (Breiman 1997)

- Gradient Tree Boosting (Friedman 1999)

- Gradient Tree Boosting with Regularization (variant of original GBM)

Learning Trees : Advantages and Challenges

- Advantages of tree-based methods
  - Highly accurate: several data science challenges are won by tree based methods
  - Easy to use: invariant to input scale, get good performance with little tuning
  - Easy to interpret and control
- Challenges on learning tree(ensembles)
  - Control over-fitting
  - Improve training speed and scale up to larger dataset

XGBoost

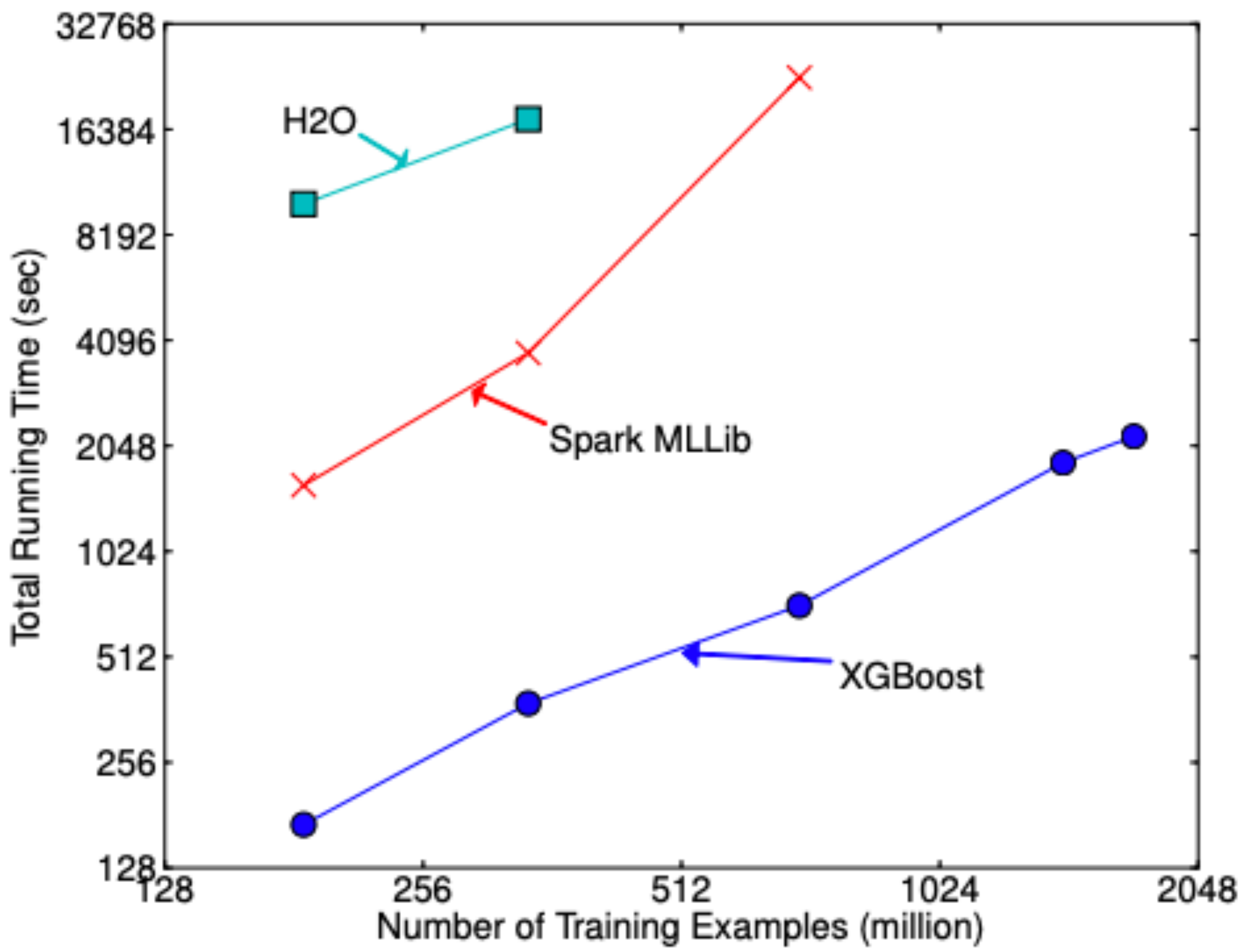
- eXtreme Gradient Boosted trees
- Model improvement
    - Regularized objective for better model
- Systems optimizations
    - Out of core computing
    - Parallelization
    - Cache optimization
    - Distributed computing
- Algorithm improvements
    - Sparse aware algorithm
    - Weighted approximate quantile sketch

How can we learn tree ensembles?

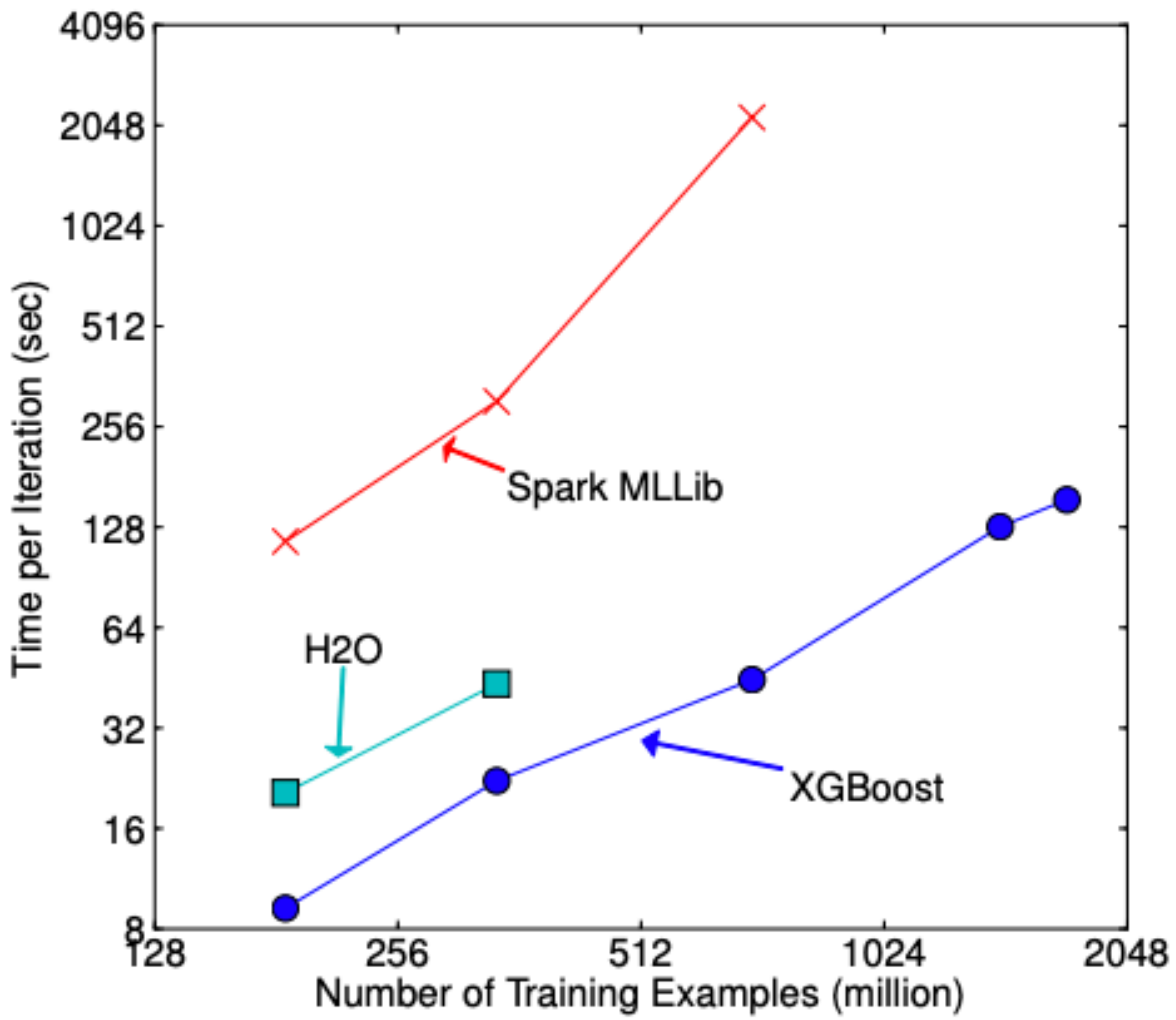
$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

- We cannot use methods like SGD
- Solution: Additive Training (Boosting)
  - Start from constant prediction, add a new function each time

Performance



(a) End-to-end time cost include data loading



(b) Per iteration cost exclude data loading

Thanks!