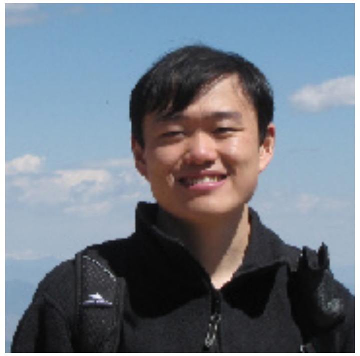
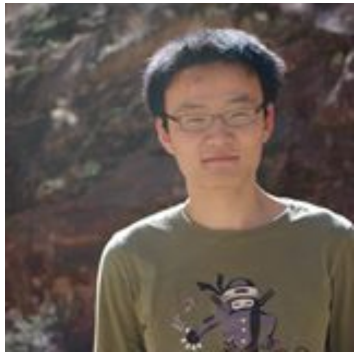
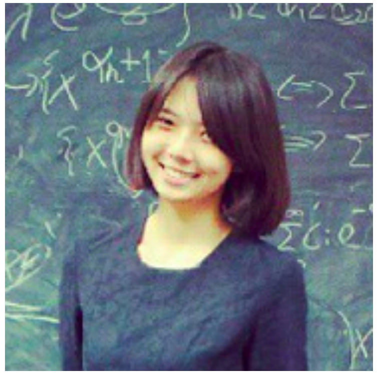
ELF: **E**xtensive, **L**ightweight and **F**lexible Framework for Game Research


Yuandong Tian


Qucheng Gong

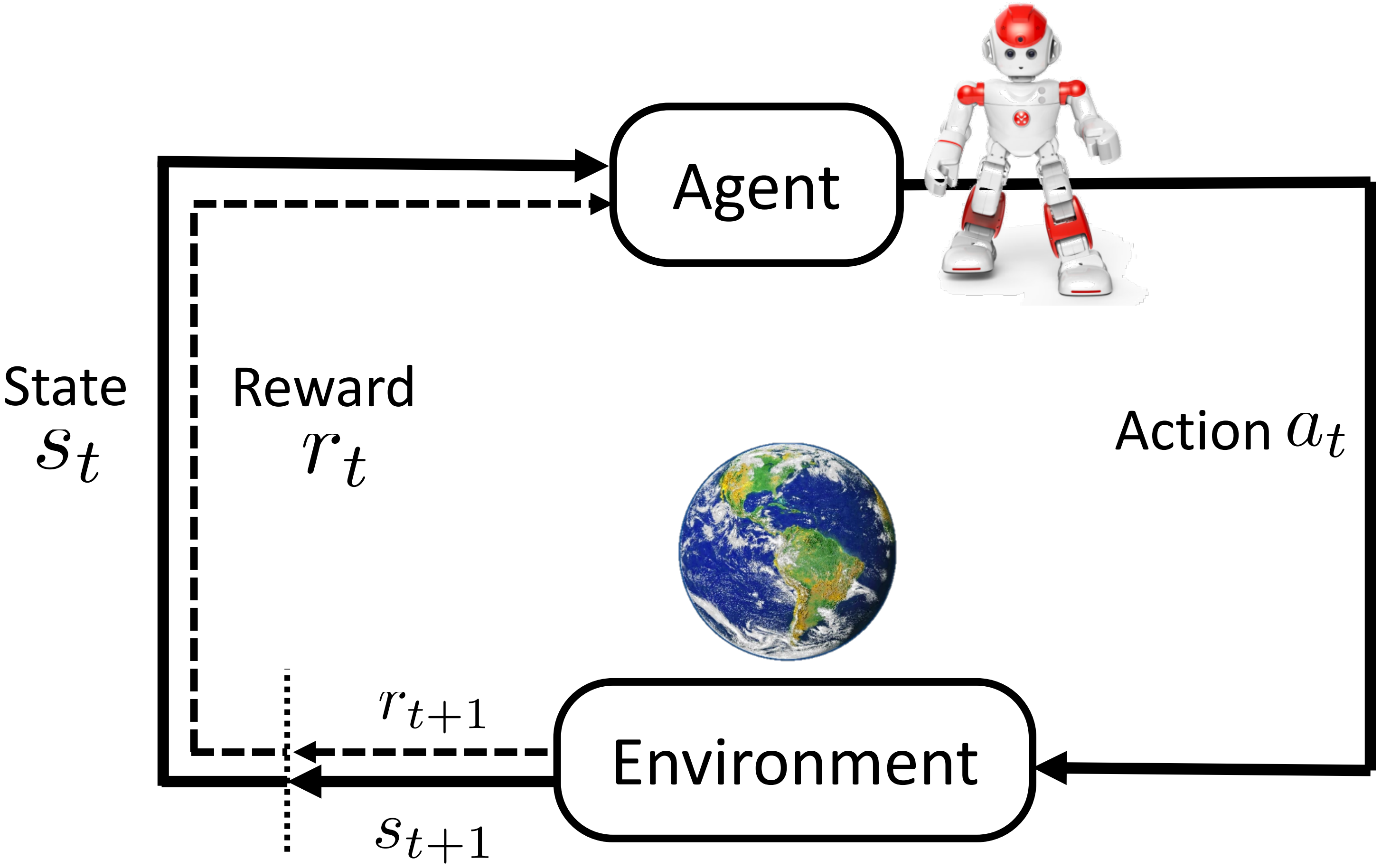

Wenling Shang


Yuxin Wu


Larry Zitnick

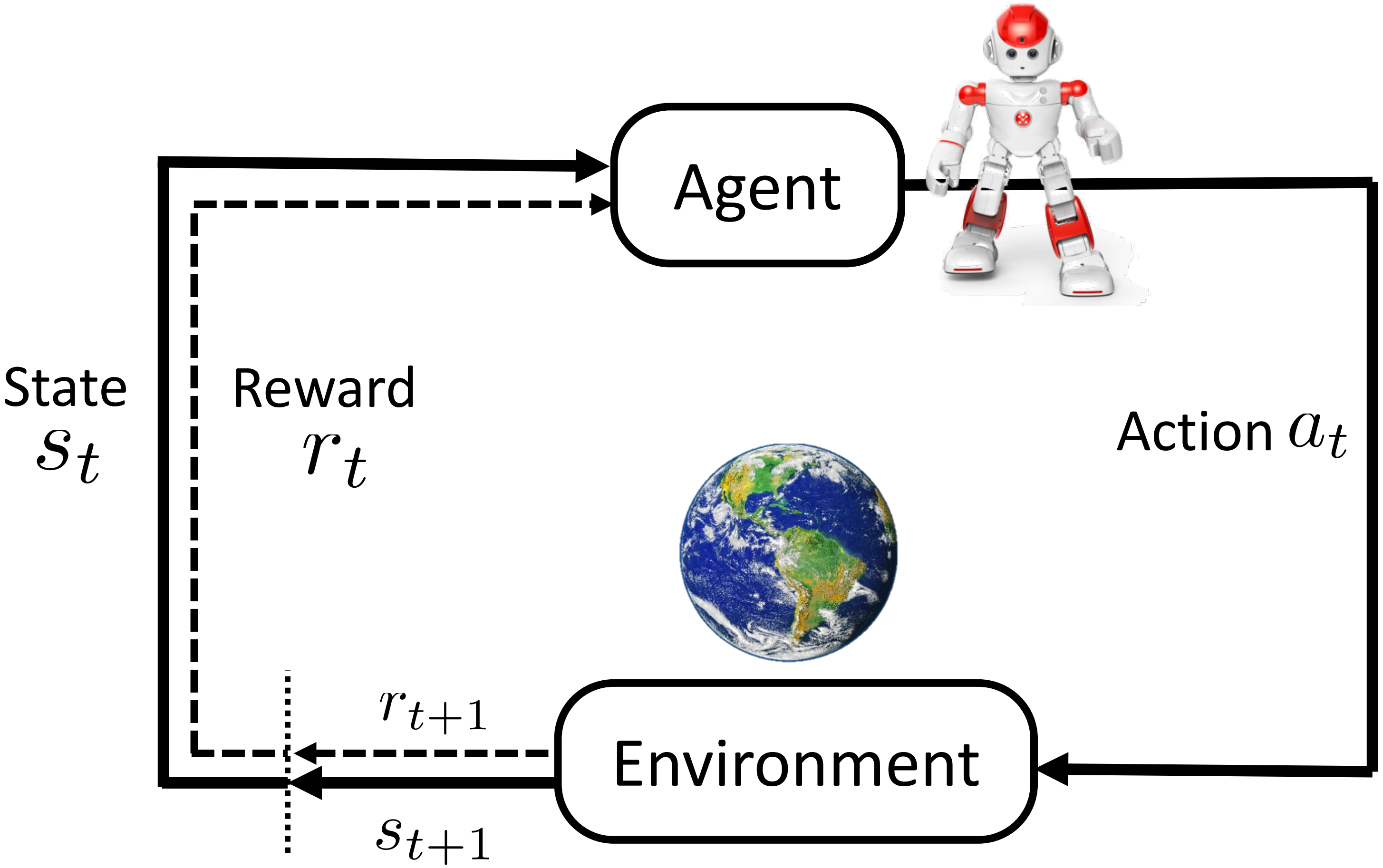
Facebook AI Research

Reinforcement Learning: Ideal and Reality



[R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction]

Reinforcement Learning: Ideal and Reality

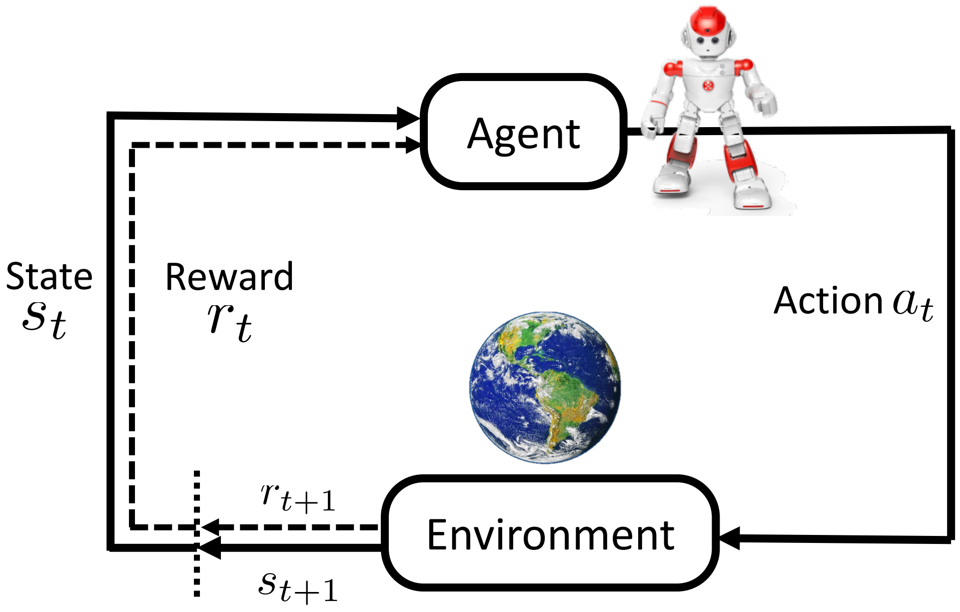


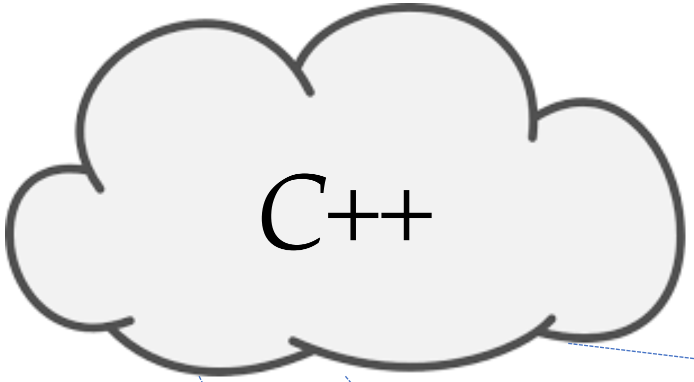
Design Choices:

CPU, GPU?
Simulation, Replays
Concurrency

[R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction]

ELF: A simple for-loop





Python

```
while True:
    batched_states = GameContext.Wait()
    replies = model(batched_states)
    GameContext.Steps(replies)
```

ELF Characteristics

Extensive

Any games with C++ interfaces
can be incorporated.

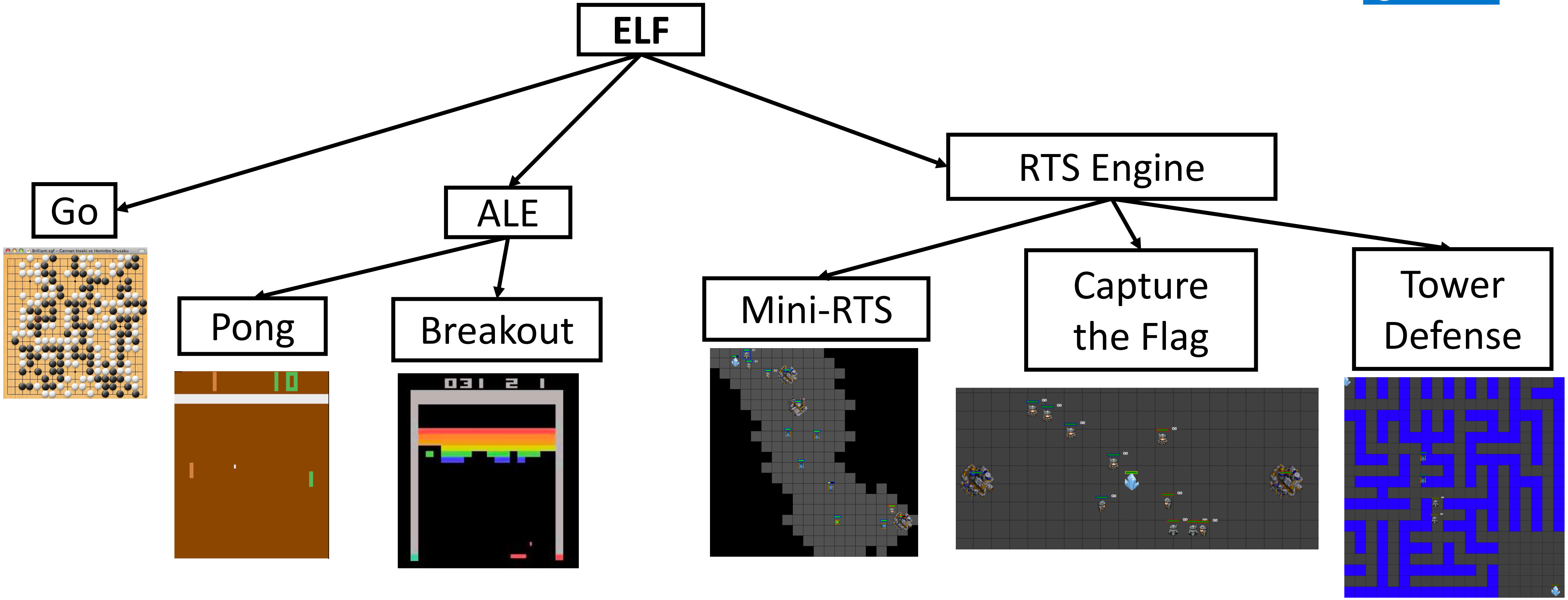
Lightweight

Fast. Mini-RTS (40K FPS per core)
Minimal resource usage (1GPU+several CPUs)
Fast training (half a day for a RTS game)

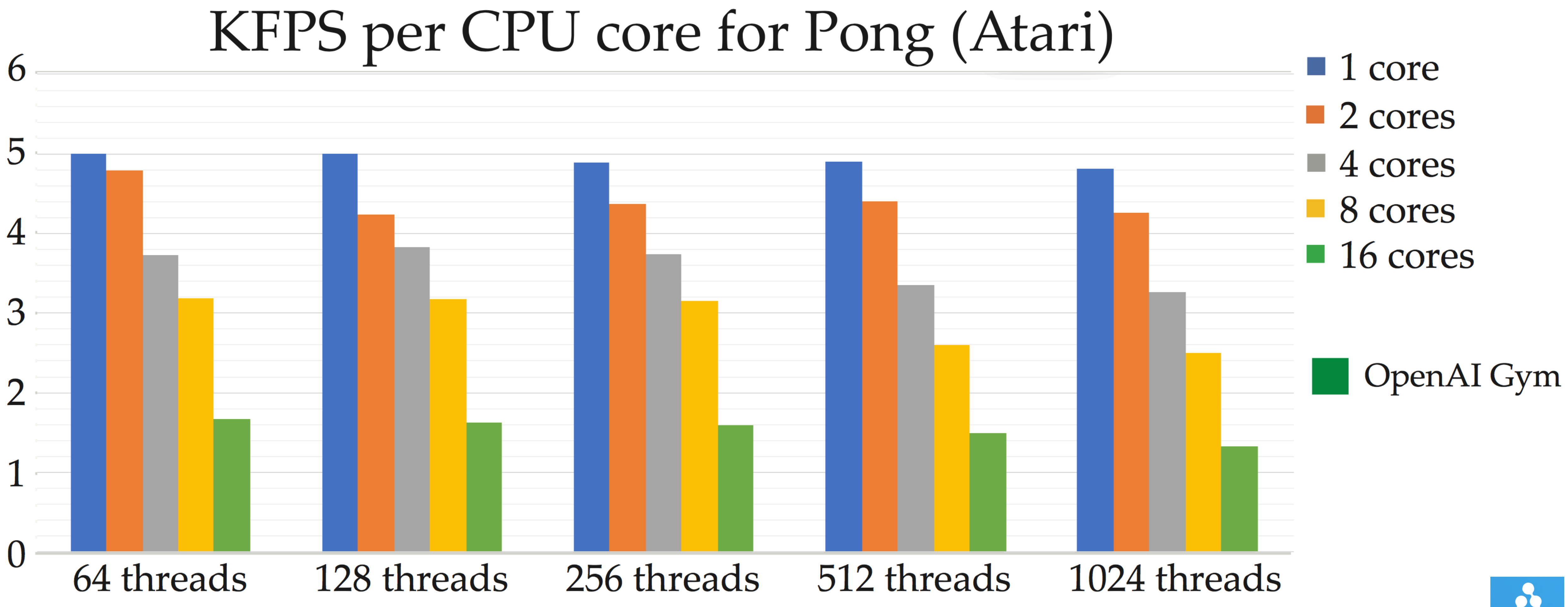
Flexible

Environment-Actor topology
Parametrized game environments.
Choice of different RL methods.

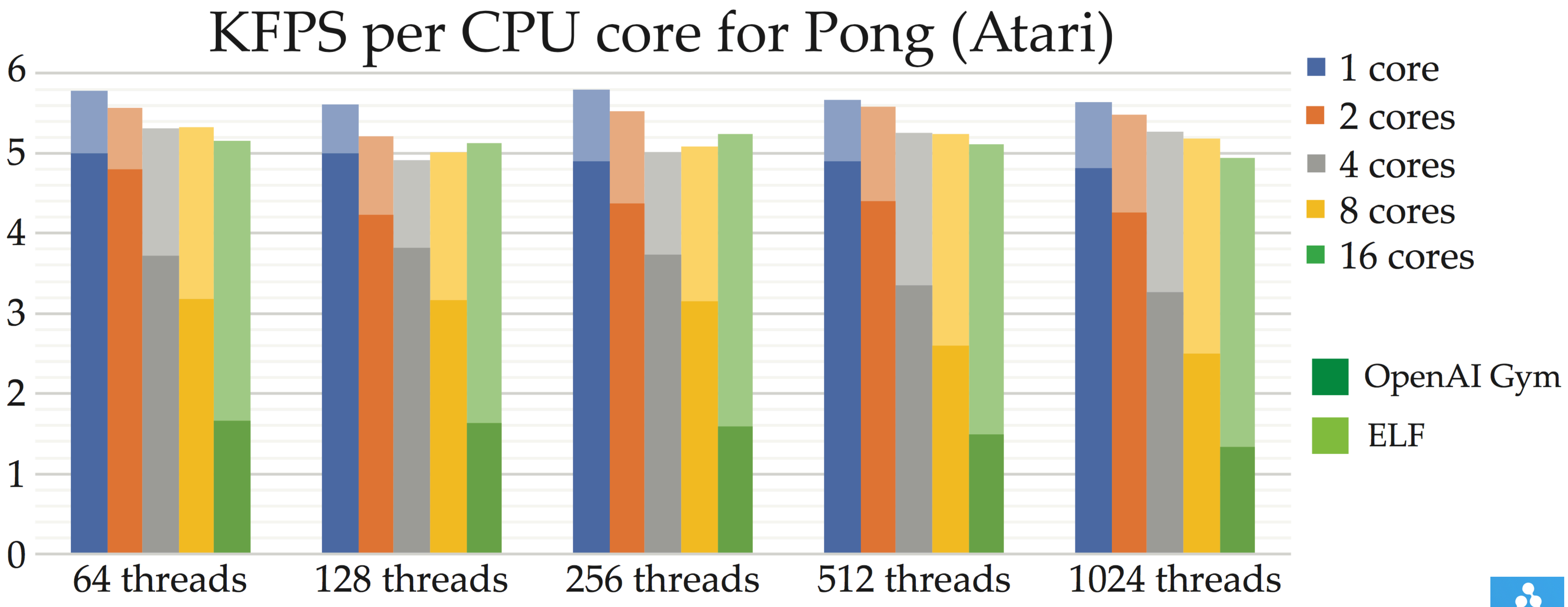
Extensibility



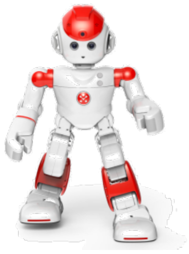
Lightweight

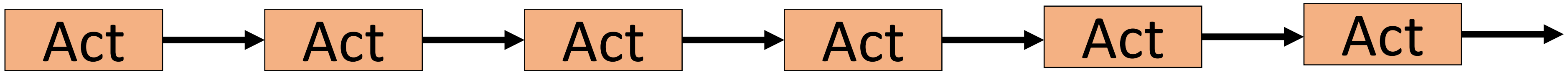


Lightweight



Flexibility

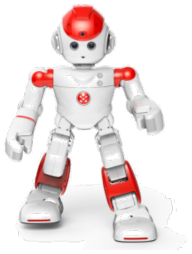


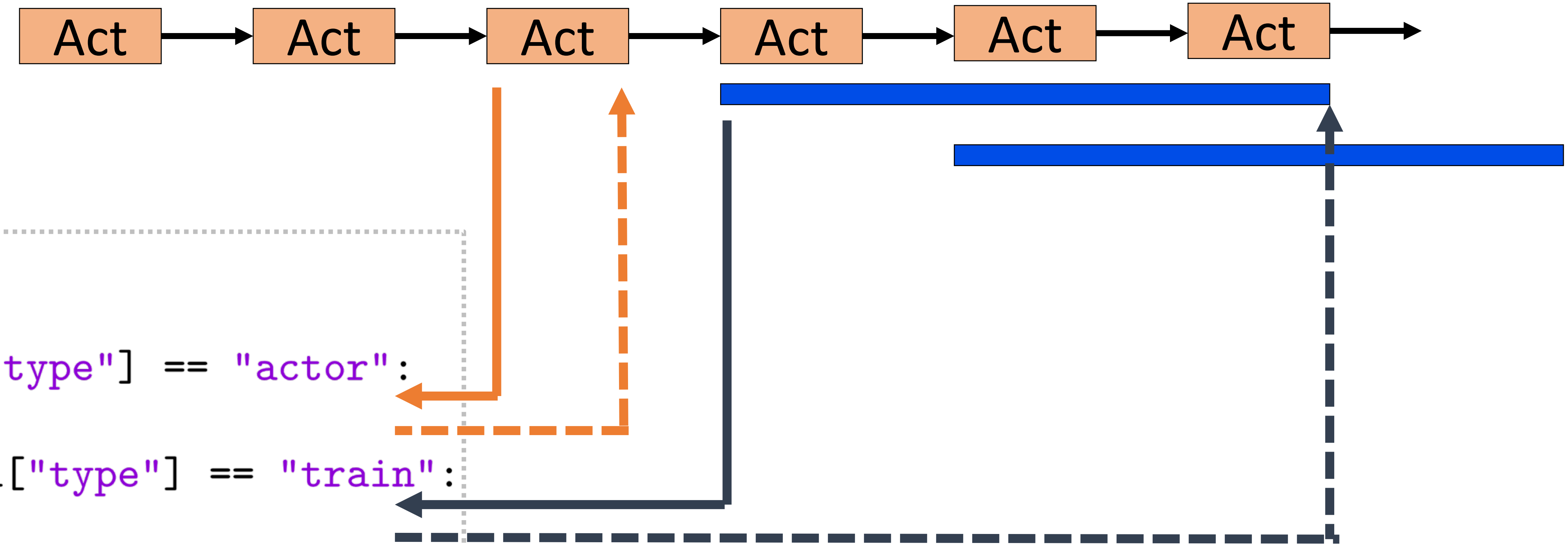



```python
while True:
    batched = GameContext.Wait()
    replies = model(batched)
    GameContext.Steps(replies)
```

Evaluation

Flexibility

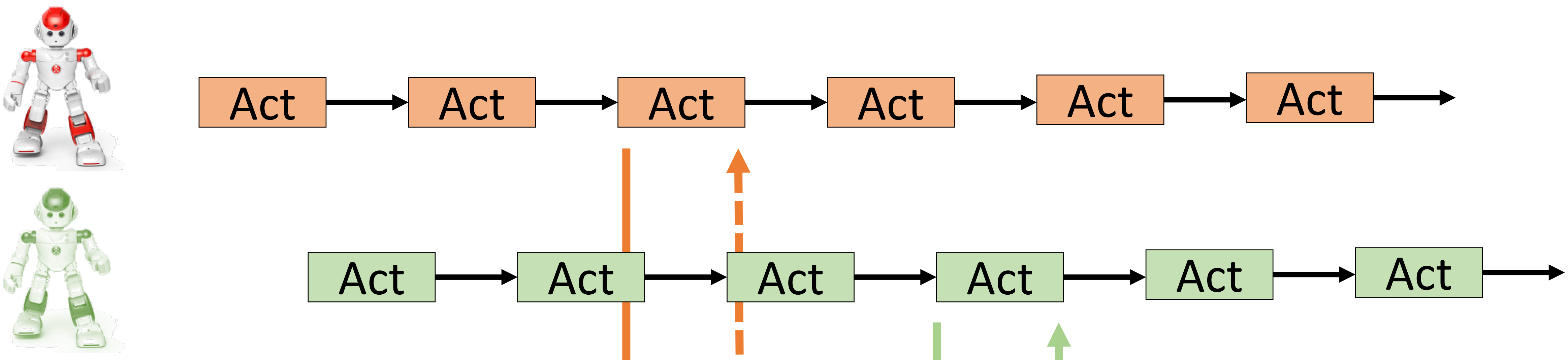





```python
while True:
    ...
    if batch["type"] == "actor":
        ...
    elif batch["type"] == "train":
        ...
```

Training

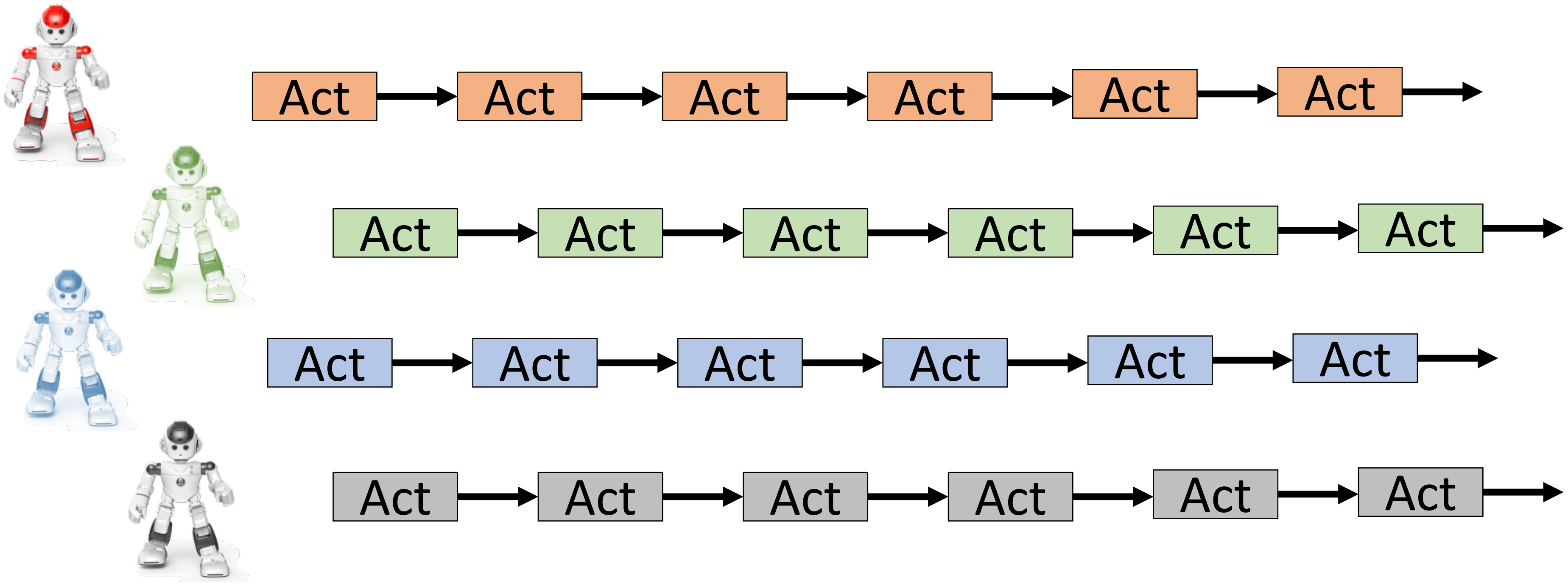
Flexibility




```python
while True:
    ...
    if batch["type"] == "actor0":
        ...
    elif batch["type"] == "actor1":
        ...
```

Self-play

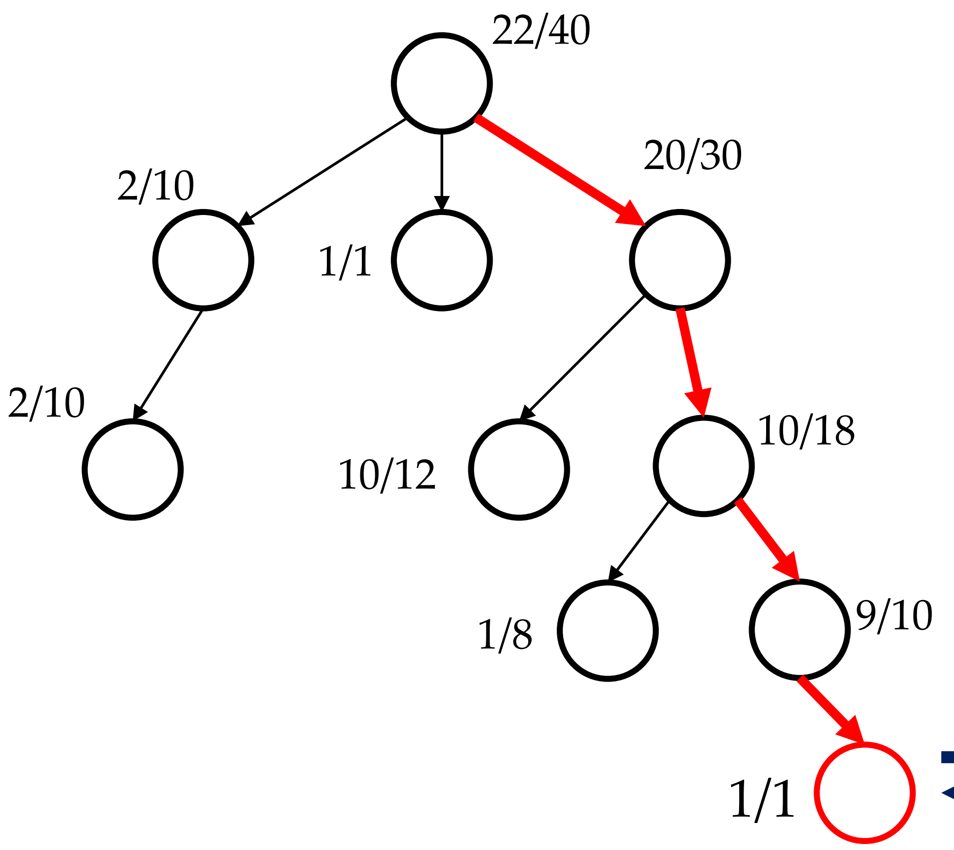
Flexibility




```python
while True:
    ...
    for i in range(n):
        if batch["type"] == "actor%d" % i:
            ...
```

Multi-agent

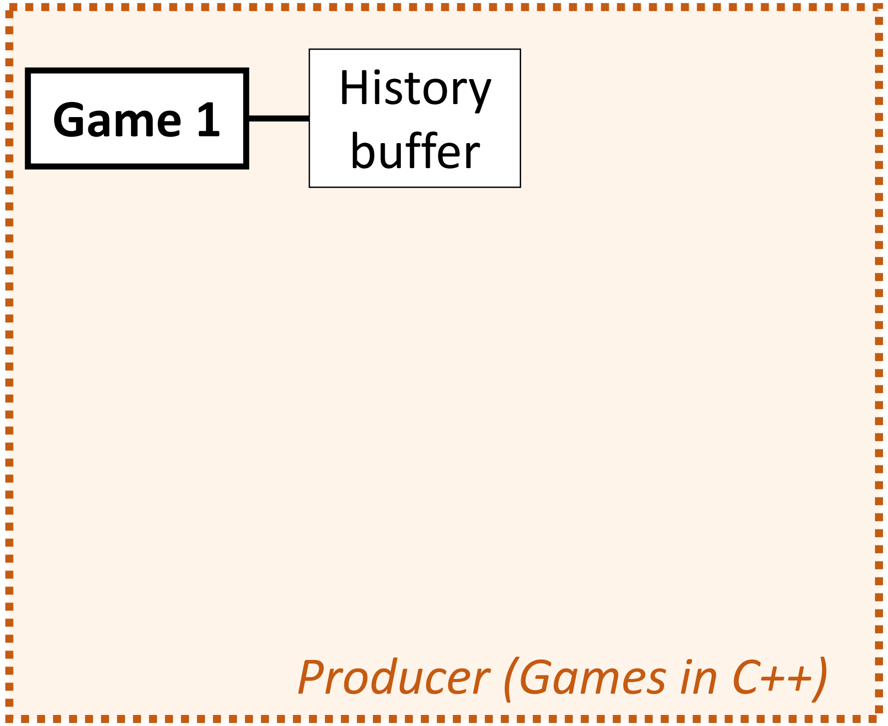
Flexibility



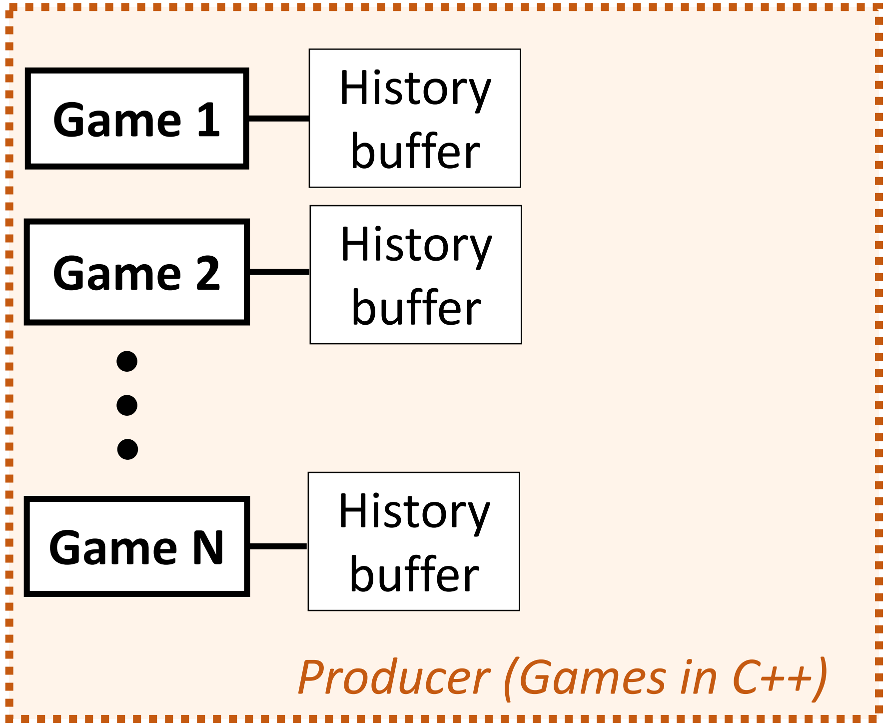

```
while True:
    batched = GameContext.Wait()
    replies = model(batched)
    GameContext.Steps(replies)
```

Monte-Carlo Tree Search

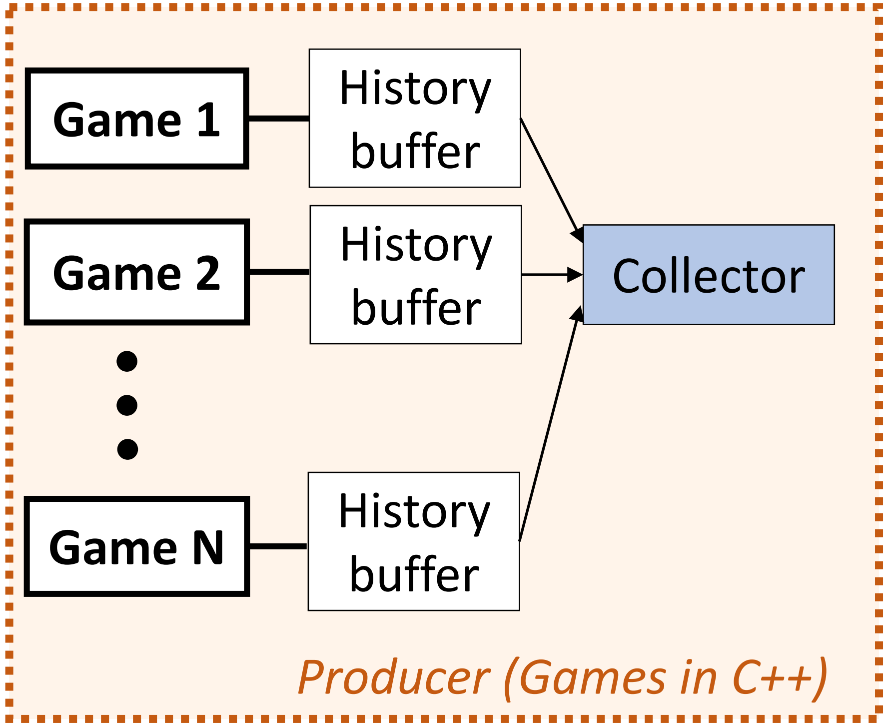
ELF design



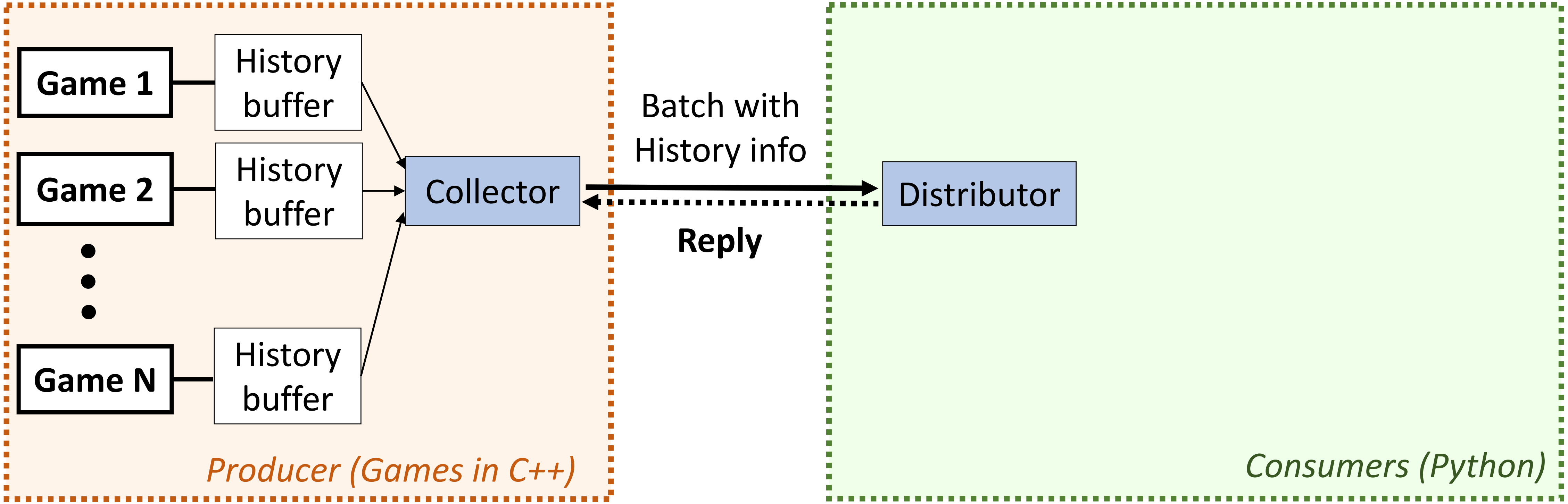
ELF design


Game 1
History buffer
Game 2
History buffer
Game N
History buffer
Producer (Games in C++)

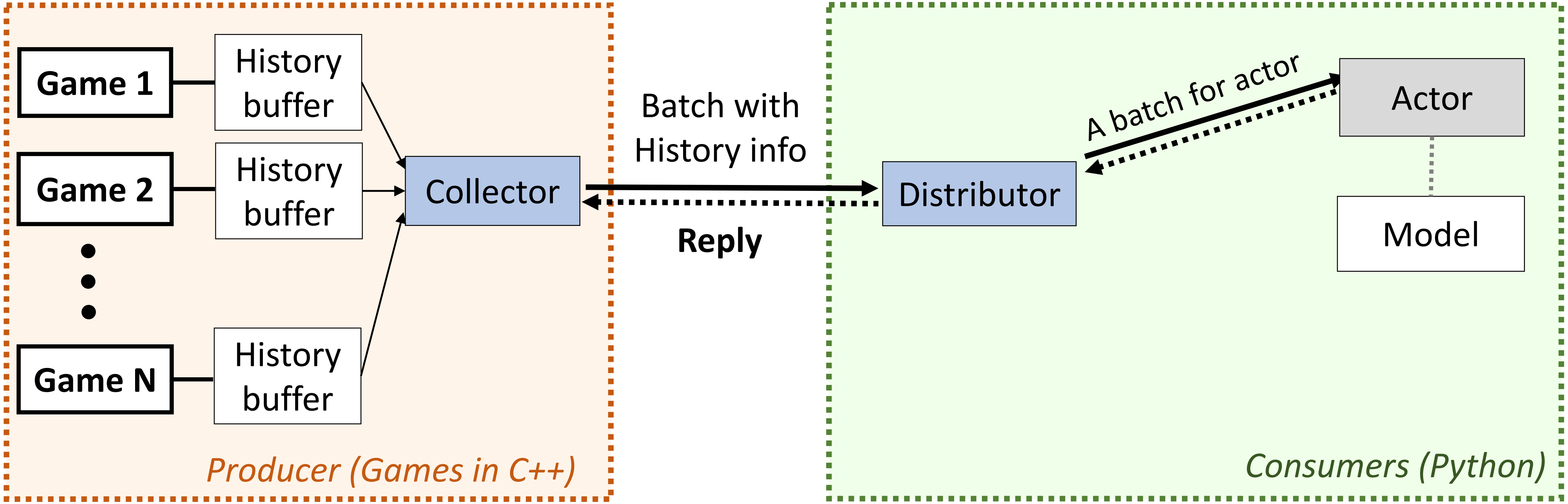
ELF design


Game 1
History buffer
Game 2
History buffer
Collector
Game N
History buffer
Producer (Games in C++)

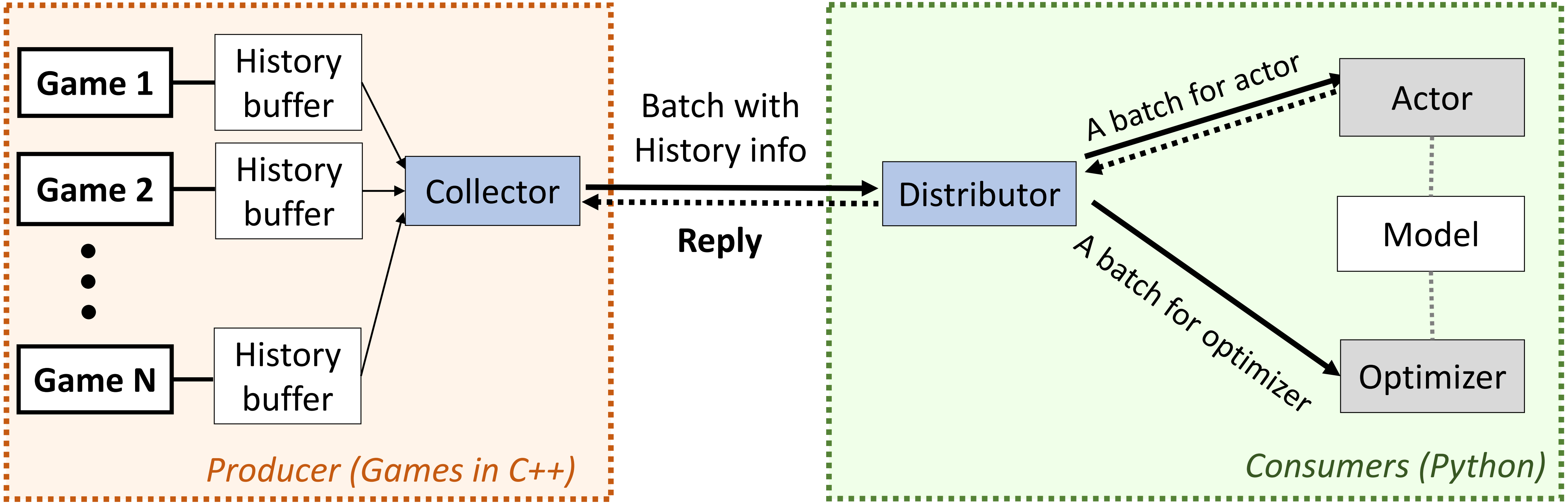
ELF design


Game 1
History buffer
Game 2
History buffer
Game N
History buffer
Collector
Producer (Games in C++)
Batch with History info
Reply
Distributor
Consumers (Python)

ELF design

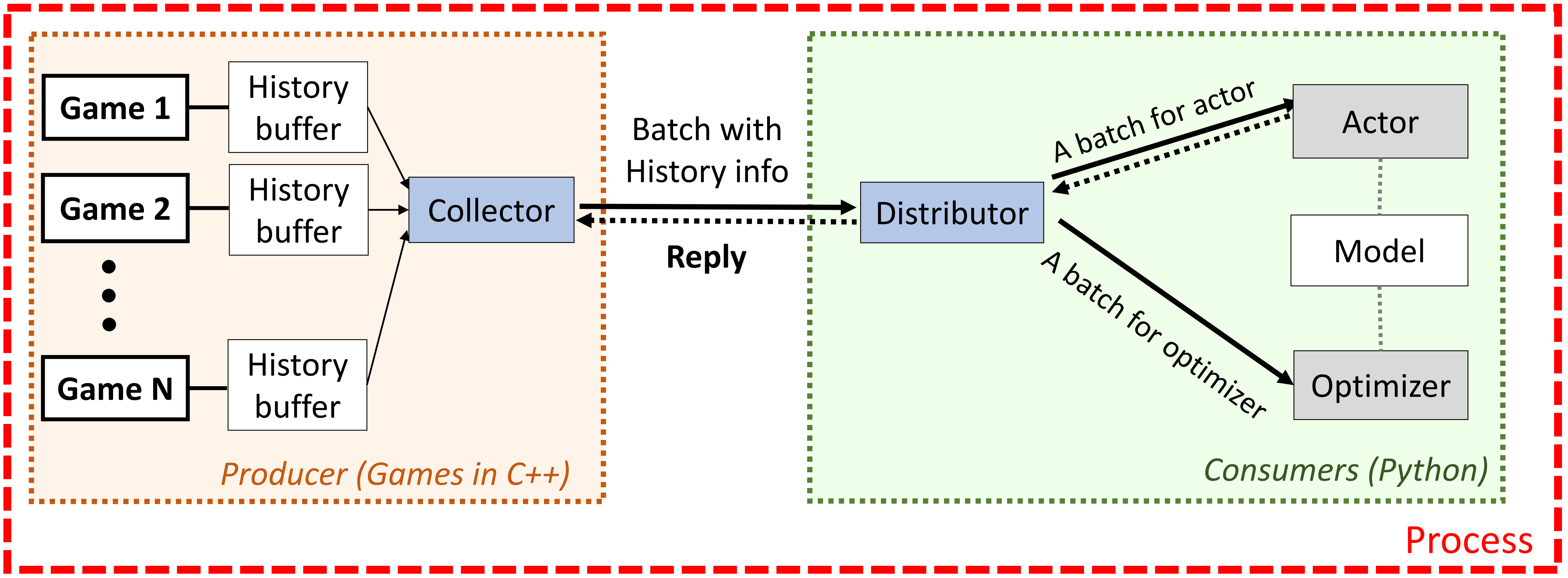

Game 1
History buffer
Game 2
History buffer
Game N
History buffer
Collector
Producer (Games in C++)
Batch with History info
Reply
Distributor
A batch for actor
Actor
Model
Consumers (Python)

ELF design



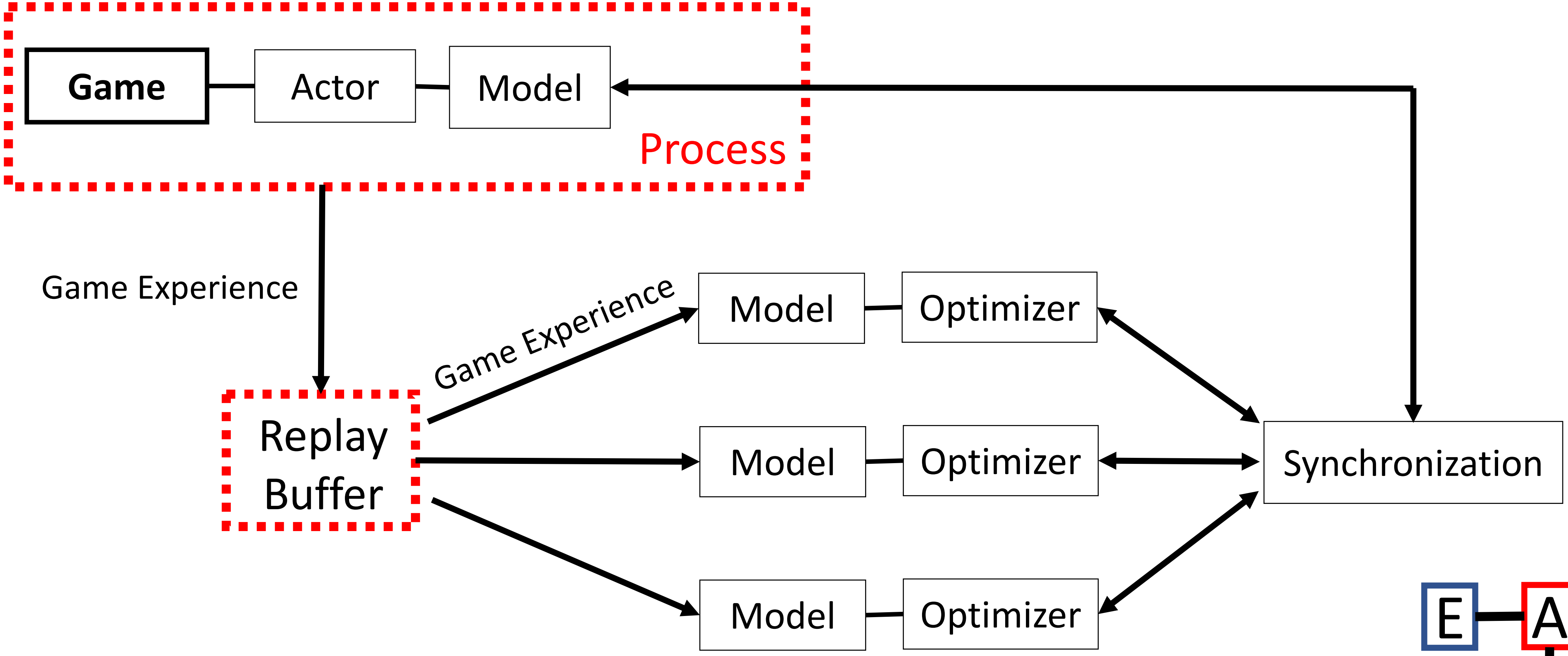
Game 1
Game 2
Game N
History buffer
History buffer
History buffer
Collector
Producer (Games in C++)
Batch with History info
Reply
Distributor
A batch for actor
A batch for optimizer
Actor
Model
Optimizer
Consumers (Python)

ELF design



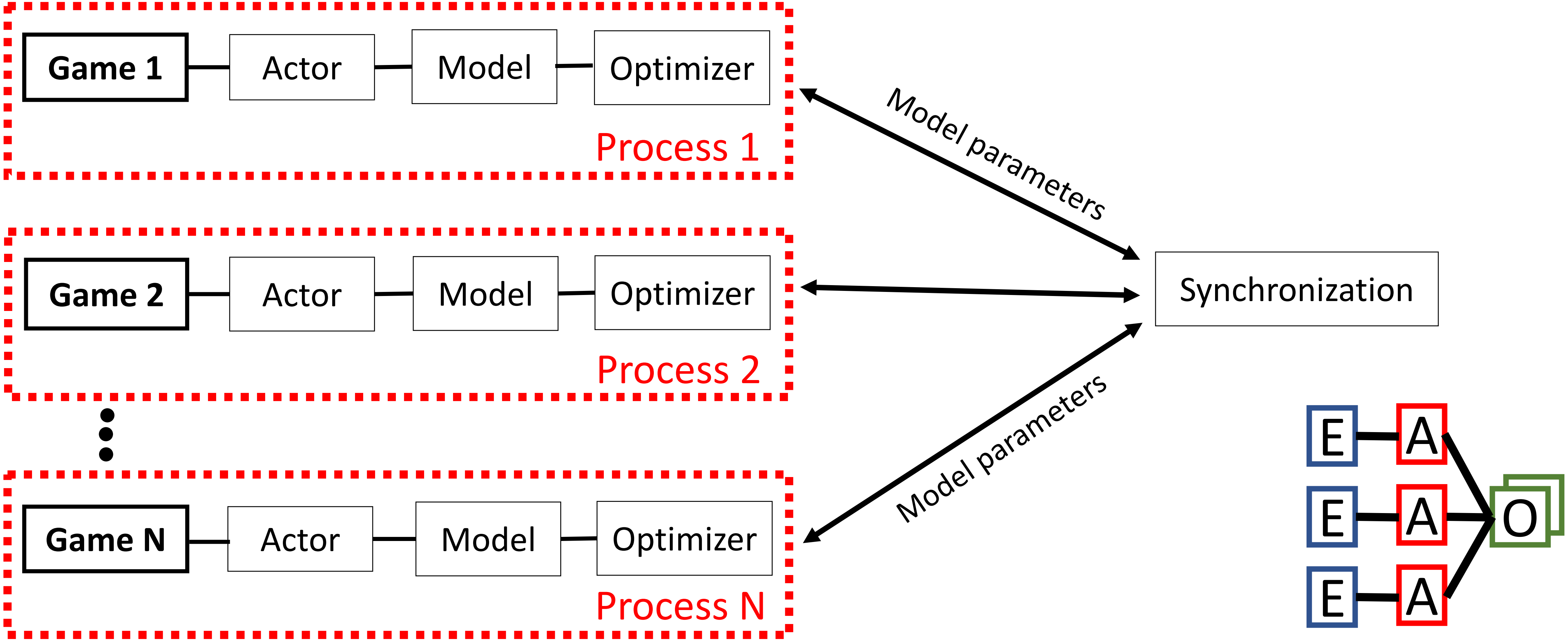
Game 1
History buffer
Game 2
History buffer
Game N
History buffer
Collector
Producer (Games in C++)
Batch with History info
Reply
Distributor
A batch for actor
Actor
Model
A batch for optimizer
Optimizer
Consumers (Python)
Process

Gorilla



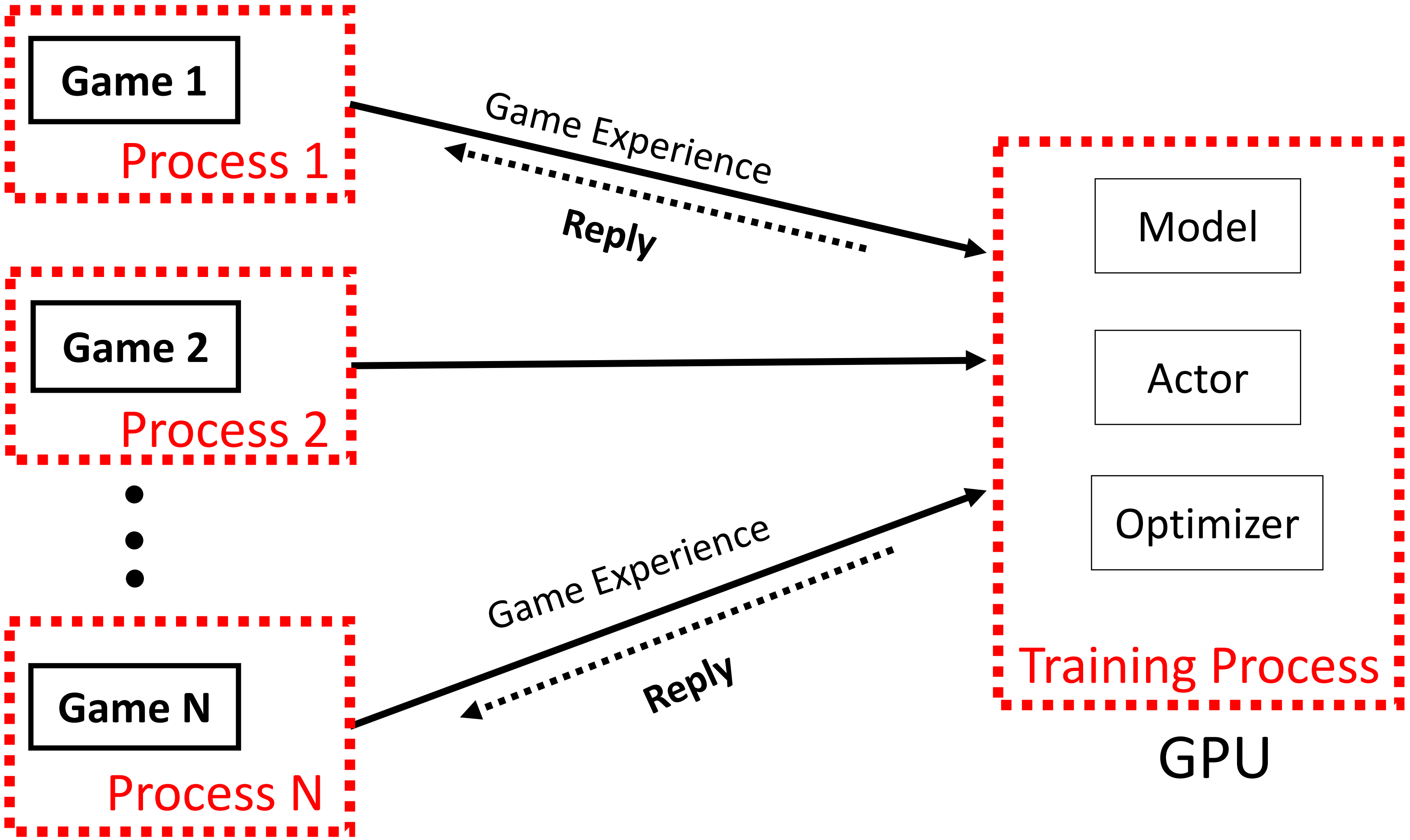
[Nair et al, Massively Parallel Methods for Deep Reinforcement Learning, ICML 2015]

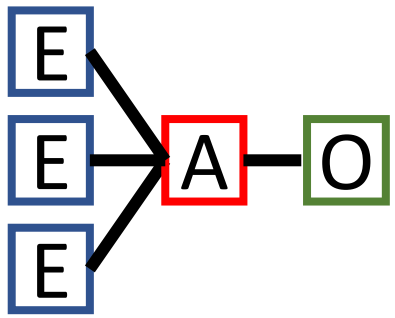
Asynchronized Advantageous Actor-Critic (A3C)



[Mnih et al, Asynchronous Methods for Deep Reinforcement Learning, ICML 2016]

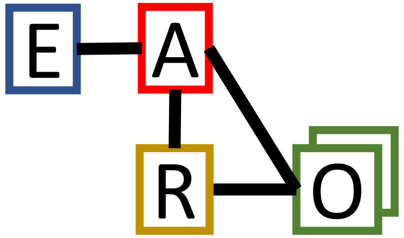
GA3C / BatchA2C





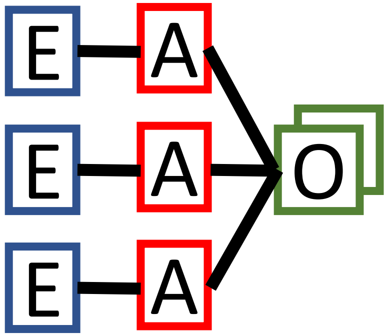
[Babaeizadeh et al, Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU, ICLR 2017]

ELF: A unified framework



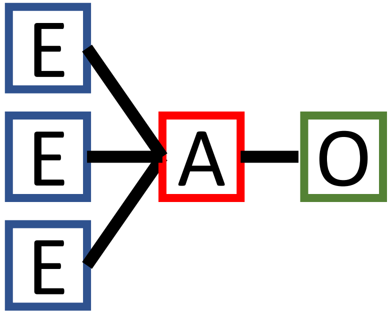
Off-policy training

Deep Q-learning



One-to-One

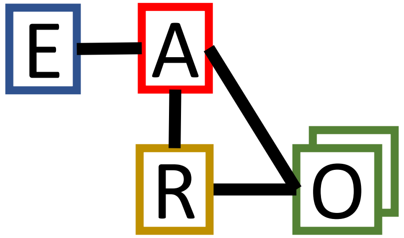
Vanilla A3C



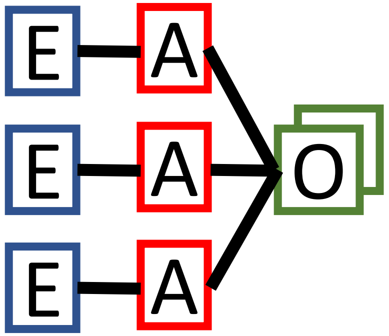
Many-to-One

BatchA2C, GA3C

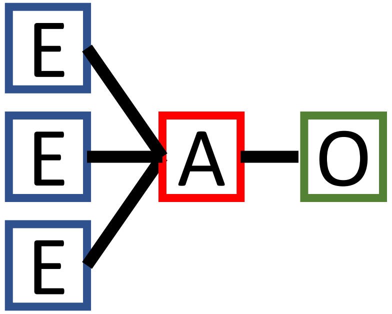
ELF: A unified framework



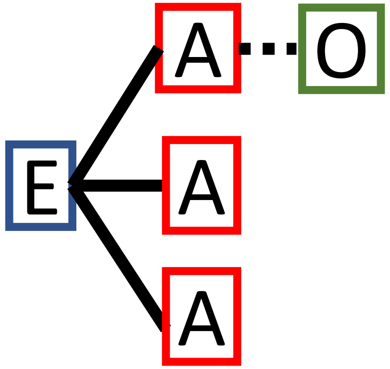
Off-policy training
Deep Q-learning



One-to-One
Vanilla A3C



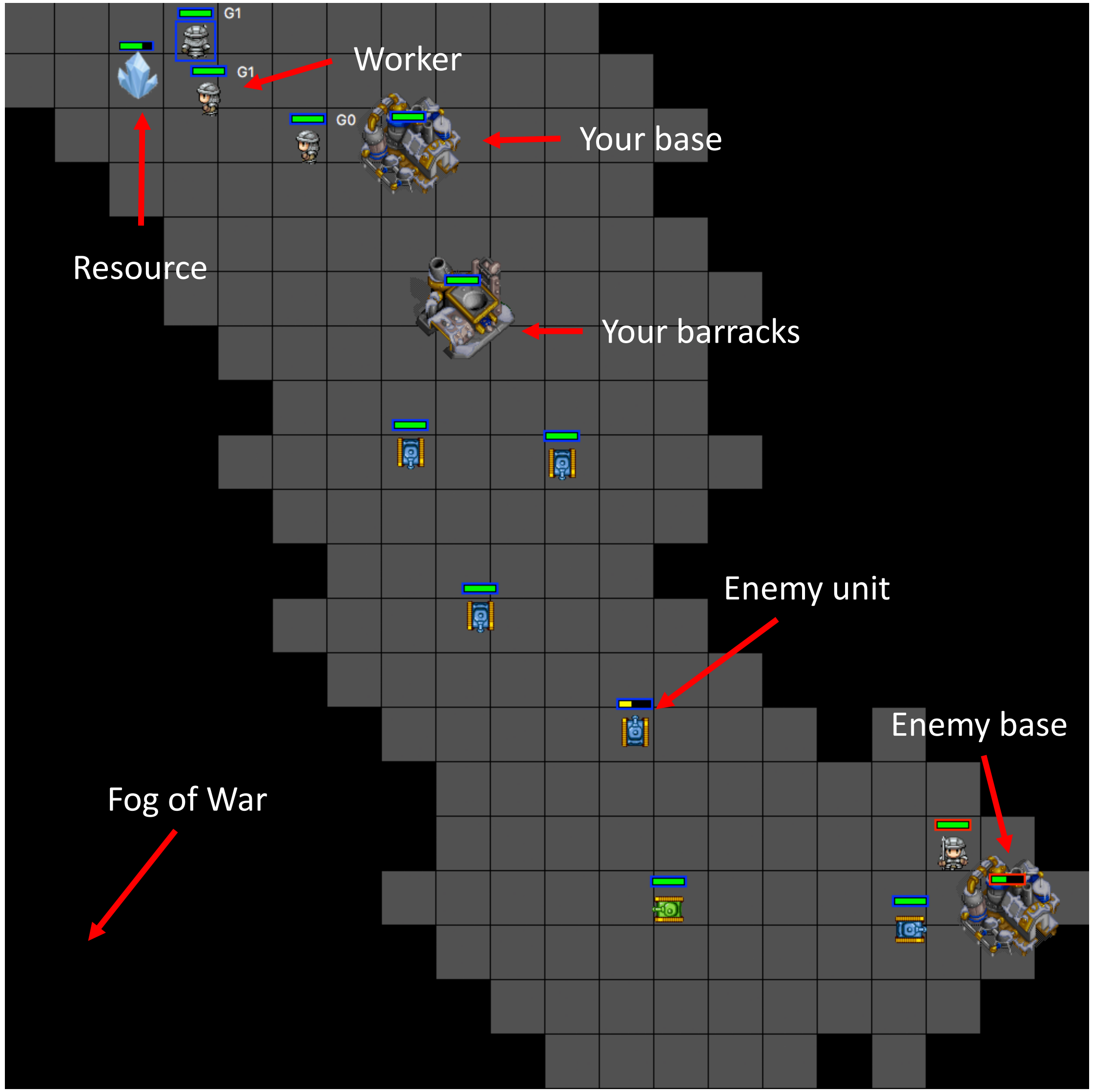
Many-to-One
BatchA3C, GA3C



One-to-Many
Self-Play,
Monte-Carlo Tree Search

Part II. MiniRTS Training

MiniRTS: A miniature RTS engine



Platform	Frame per second
ALE	6,000
Open AI Universe	60
Malmo	120
DeepMind Lab	287*/866**
VizDoom	7,000
TorchCraft	2,000
MiniRTS	40,000

* Using CPU only　　** Using CPUs and GPU

MiniRTS

Base  Build workers and collect resources.

Resource  Contains 1000 minerals.

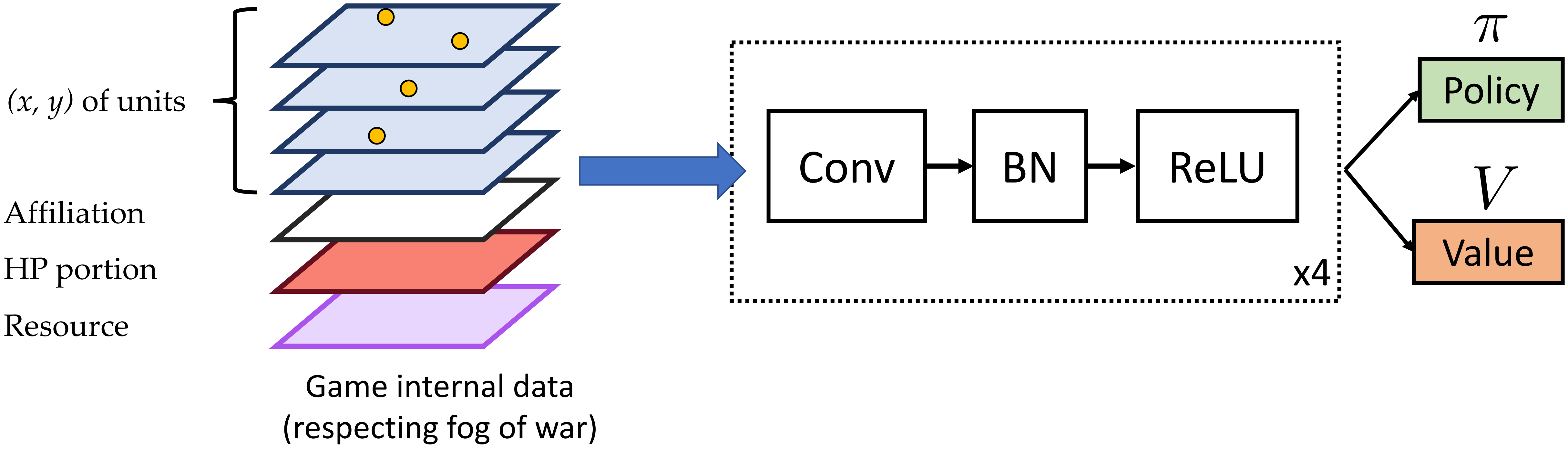
Barracks  Build melee attacker and range attacker.

Worker  Build barracks and gather resource.
Low speed in movement and low attack damage.

Melee Tank  High HP, medium movement speed, short attack range, high attack damage.

Range Tank  Low HP, high movement speed, long attack range and medium attack damage.

Training AI



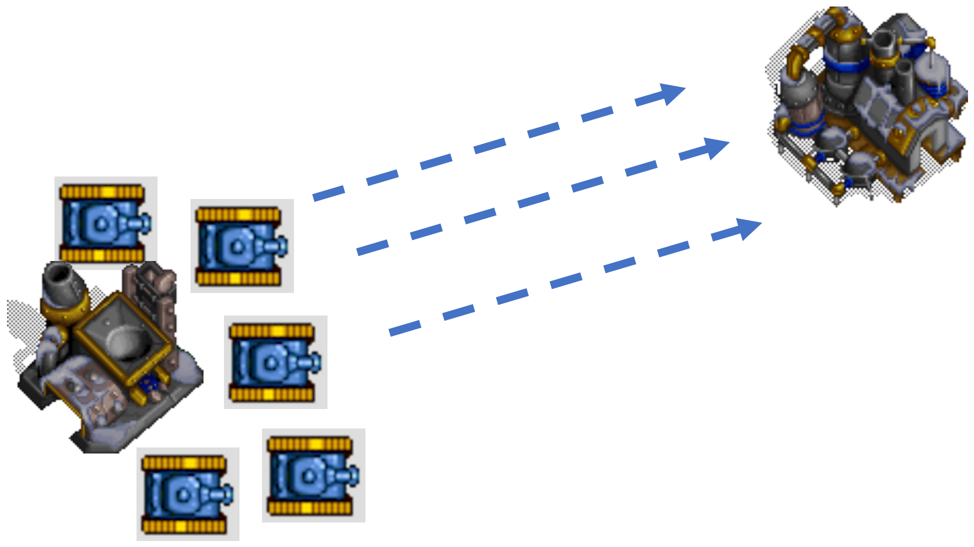
Using Internal Game data and Actor-Critic Models.

Reward is only available once the game is over.

9 Discrete Strategic Actions

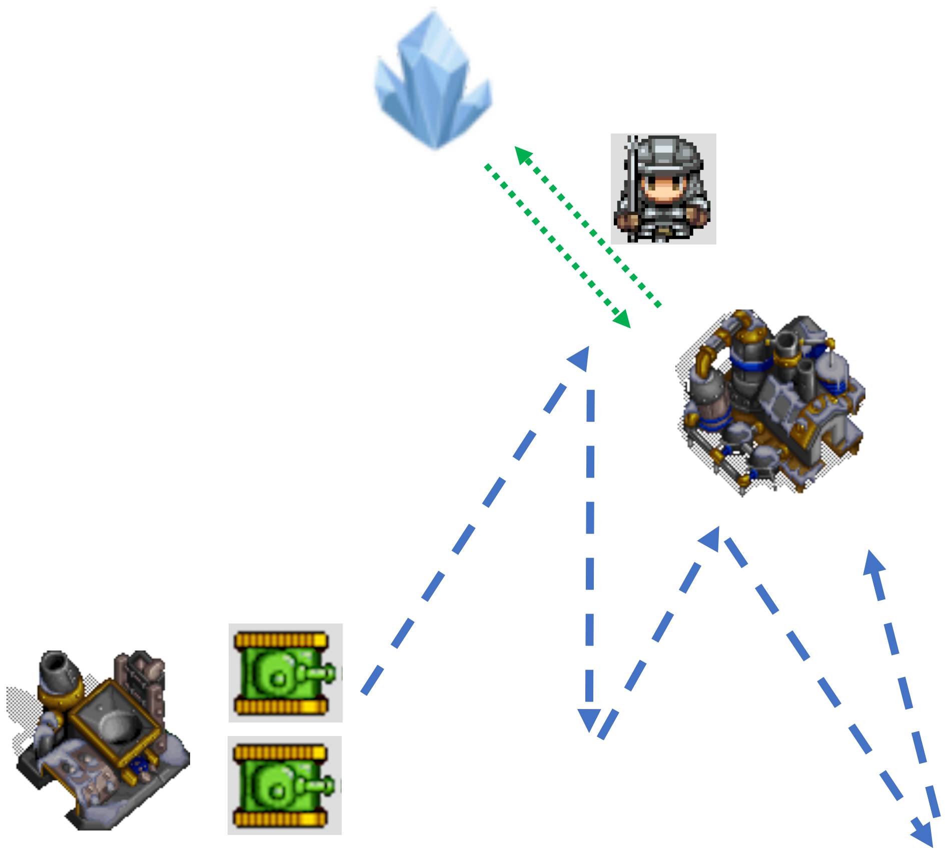
No.	Action name	Descriptions
1	IDLE	Do nothing
2	BUILD WORKER	If the base is idle, build a worker
3	BUILD BARRACK	Move a worker (gathering or idle) to an empty place and build a barrack.
4	BUILD MELEE ATTACKER	If we have an idle barrack, build an melee attacker.
5	BUILD RANGE ATTACKER	If we have an idle barrack, build a range attacker.
6	HIT AND RUN	If we have range attackers, move towards opponent base and attack. Take advantage of their long attack range and high movement speed to hit and run if enemy counter-attack.
7	ATTACK	All melee and range attackers attack the opponent's base.
8	ATTACK IN RANGE	All melee and range attackers attack enemies in sight.
9	ALL DEFEND	All troops attack enemy troops near the base and resource.

Rule-based AIs



AI_SIMPLE

Build 5 tanks and attack

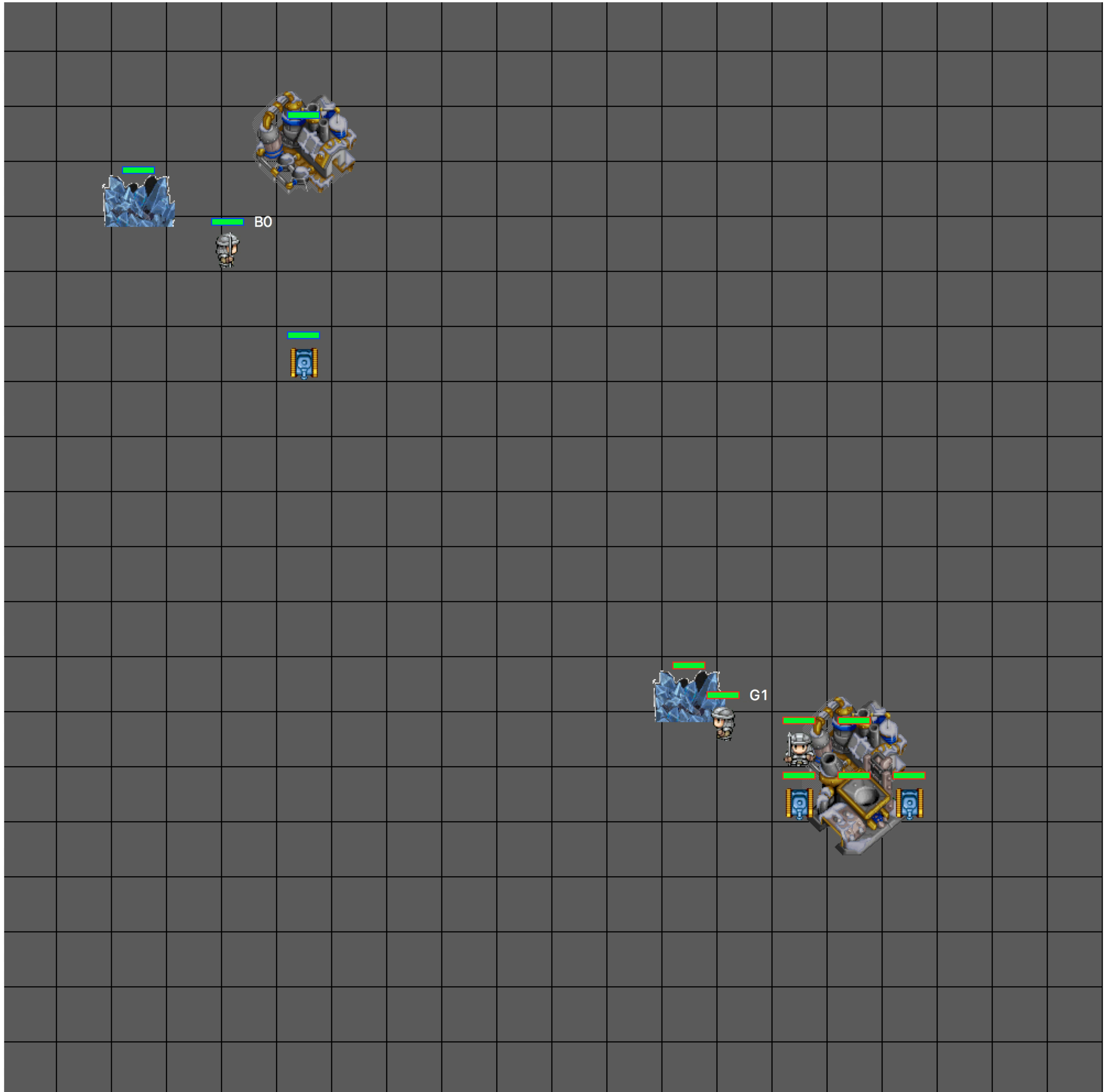


AI_HIT_AND_RUN

Build 2 tanks and harass

<u>MiniRTS trains with a single GPU and 6 CPUs in half a day.</u>

Trained AI



AI_SIMPLE

Win rate against rule-based AI

Frame skip (how often AI makes decisions)

Opponent Frame skip	AI_SIMPLE	AI_HIT_AND_RUN
50	68.4(±4.3)	63.6(±7.9)
20	61.4(±5.8)	55.4(±4.7)
10	52.8(±2.4)	51.1(±5.0)

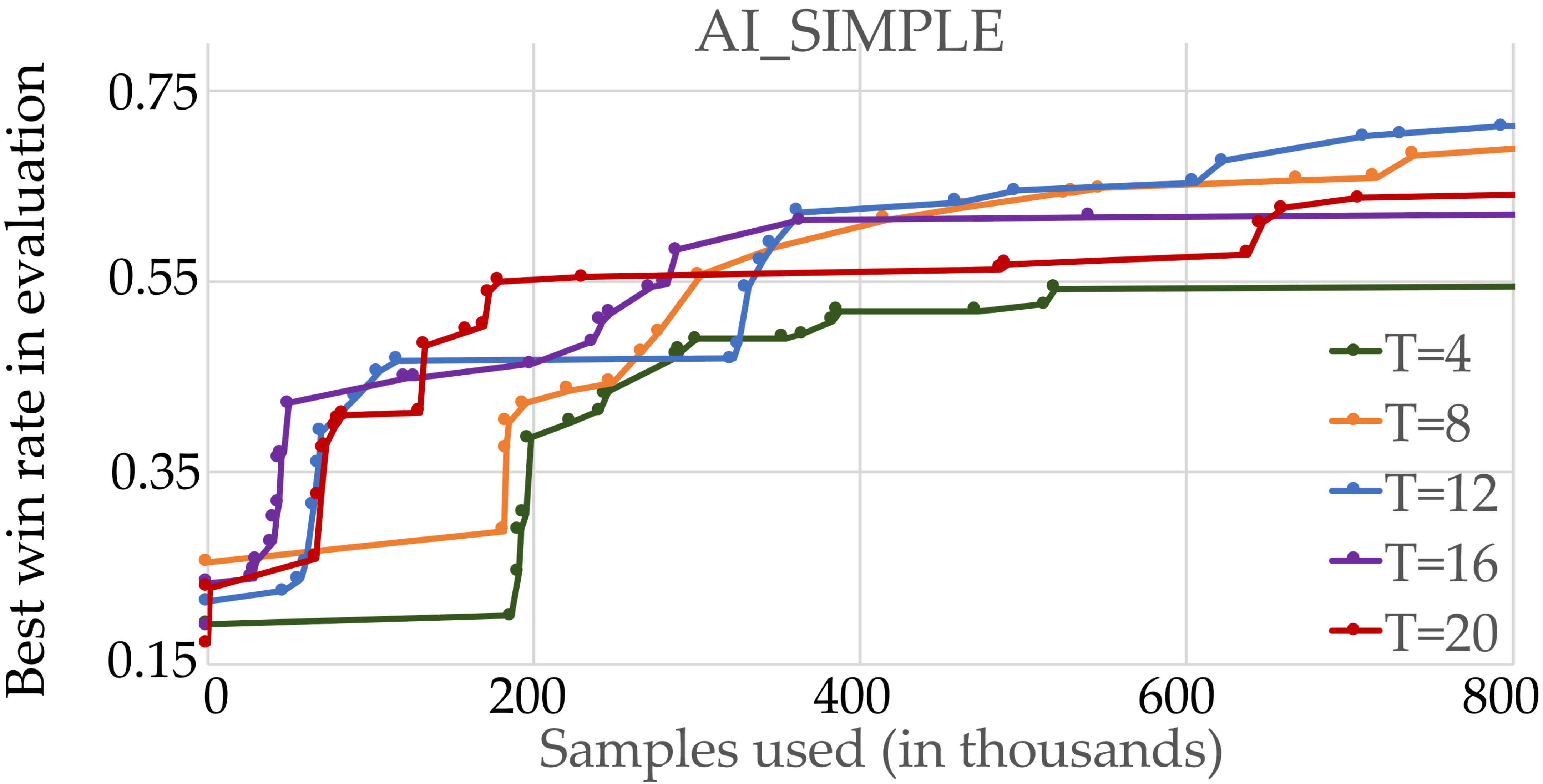
*The frameskip of learned AI is always 50

Win rate against rule-based AI

Network Architecture: Conv → BN → ReLU

Win Rate (10K games)	SIMPLE (median)	SIMPLE (mean/std)	HIT_AND_RUN (median)	HIT_AND_RUN (mean/std)
ReLU	52.8	54.7(±4.2)	60.4	57.0(±6.8)
Leaky ReLU	59.8	61.0(±2.6)	60.2	60.3(±3.3)
ReLU + BN	61.0	64.4(±7.4)	55.6	57.5(±6.8)
Leaky ReLU + BN	**72.2**	**68.4(±4.3)**	**65.5**	**63.6(±7.9)**

Effect of Multi-step Training

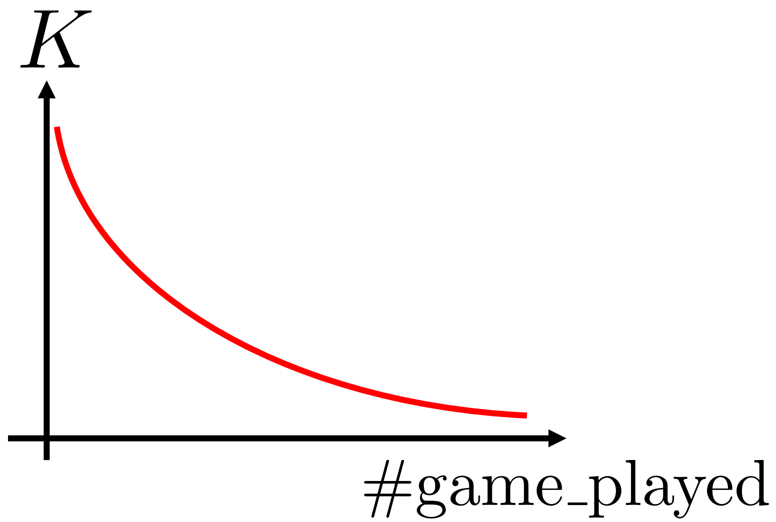


Curriculum Training

Win Rate	Without curriculum training	With curriculum training
AI_SIMPLE	66.0 (±2.4)	**68.4 (±4.3)**
AI_HIT_AND_RUN	54.4 (±15.9)	**63.6 (±7.9)**

First k decisions made by `AI_SIMPLE`
then made by trained AI

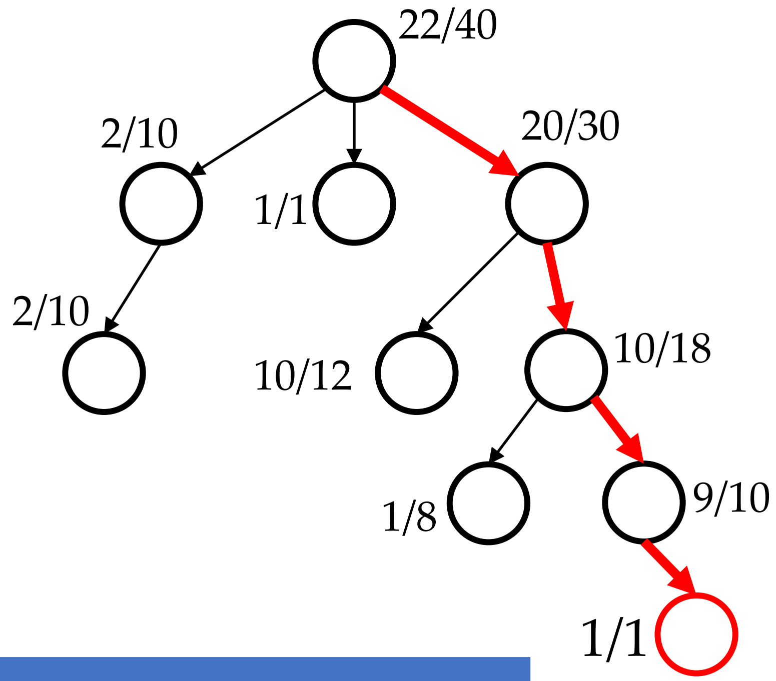
$$k \sim \mathrm{Uniform}[0, K]$$
$$K \propto \beta^{-\#\mathrm{game\_played}}$$



Transfer Learning

Win Rate	AI_SIMPLE	AI_HIT_AND_RUN	Combined (50%SIMPLE+50% H&R)
SIMPLE	**68.4 (±4.3)**	26.6(±7.6)	47.5(±5.1)
HIT_AND_RUN	34.6(±13.1)	**63.6 (±7.9)**	49.1(±10.5)
Combined	51.8(±10.6)	54.7(±11.2)	**53.2(±8.5)**

Monte Carlo Tree Search



Win Rate	AI_SIMPLE	AI_HIT_AND_RUN
Random	24.2 (±3.9)	25.9 (±0.6)
MCTS*	73.2 (±0.6)	62.7 (±2.0)
Trained AI	**68.4(±4.3)**	**63.6(±7.9)**

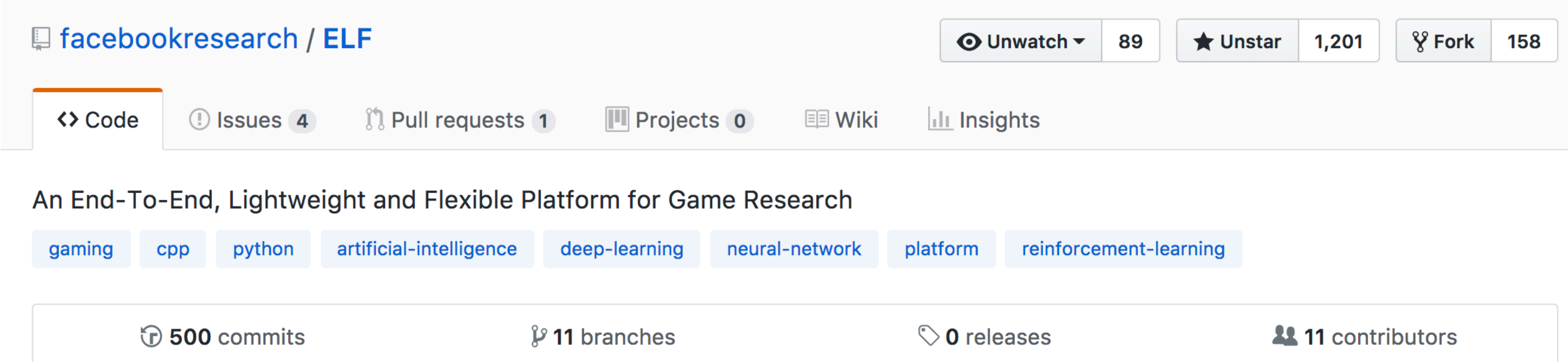
* repeat on 1000 games, each using 800 rollouts.

MCTS uses complete information and perfect dynamics

Ongoing Work

- One framework for different games.
  - DarkForest remastered: https://github.com/facebookresearch/ELF/tree/master/go
- Richer game scenarios for MiniRTS.
  - LUA scripting support
  - Multiple bases (Expand? Rush? Defending?)
  - More complicated units.
- Realistic action space
  - One command per unit
- Model-based Reinforcement Learning
- Self-Play (Trained AI versus Trained AI)

Open Source



https://github.com/facebookresearch/ELF

LUA Interface for MiniRTS

- Easy to change game dynamics
  - Don’t need to touch C++.
- Comparable speed to C++
  - 1.5x slower than compiled code.

```lua
g_funcs = { }
function g_funcs.attack(env, cmd)
    local target = env:unit(cmd.target)
    local u = env:self()

    if target:isdead() or not u:can_see(target) then
        -- c_print("Task finished!")
        return global.CMD_COMPLETE
    end
    local att_r = u:att_r()
    local in_range = env:dist_sqr(target:p()) <= att_r * att_r
    if u:cd_expired(global.CD_ATTACK) and in_range then
        -- print("Attacking .. ")
        -- Then we need to attack.
        if att_r <= 1.0 then
            env:send_cmd_melee_attack(cmd.target, u:att())
        else
            env:send_cmd_emit_bullet(cmd.target, u:att())
        end
        env:cd_start(global.CD_ATTACK)
    else
        if not in_range then
            -- print("Moving towards target .. ")
            env:move_towards(target)
        end
    end
    -- print("Done with Attacking .. ")
end
```

RLPytorch

- A RL platform in PyTorch
- A3C in 30 lines.

```
# A3C
def update(self, batch):
    ''' Actor critic model '''
    R = deepcopy(batch["V"][T - 1])
    batchsize = R.size(0)
    R.resize_(batchsize, 1)

    for t in range(T - 2, -1, -1):
        # Forward pass
        curr = self.model_interface.forward("model", batch.hist(t))

        # Compute the reward.
        R = R * self.args.discount + batch["r"][t]
        # If we see any terminal signal, do not backprop
        for i, terminal in enumerate(batch["terminal"][t]):
            if terminal: R[t][i] = curr["V"].data[i]

        # We need to set it beforehand.
        self.policy_gradient_weights = R - curr["V"].data

        # Compute policy gradient error:
        errs = self._compute_policy_entropy_err(curr["pi"], batch["a"][t])
        # Compute critic error
        value_err = self.value_loss(curr["V"], Variable(R))

        overall_err = value_err + errs["policy_err"]
        overall_err += errs["entropy_err"] * self.args.entropy_ratio
        overall_err.backward()
```

Questions?

Tonight Poster: #96

https://github.com/facebookresearch/ELF