Admin

- Use slack for communication with me and each other
- In class **quiz** next **monday**
- Wednesday we discuss the quiz
- *Office hours will be once a week in an Asia friendly time: 6pm ?*
- I have been adding to the project doc. Start now if you like
- We will do mentor sign up this week
- **Any admin questions for me?**

About me …

What is Reinforcement Learning?

- Agent-oriented learning—learning by interacting with an environment to **achieve a goal**
- Learning by trial and error, with only **delayed evaluative feedback** (reward)
  - the kind of machine learning like natural learning (animals)
  - learning that can tell for itself when it is right or wrong

Plan for today

Plan for today

Goal: Remind you about RL & sequential decision making

Plan for today

Goal: Remind you about RL & sequential decision making

- The basics of RL:
  - states, actions, rewards, time, MDPs, policies, value

Plan for today

Goal: Remind you about RL & sequential decision making

- The basics of RL:
  - states, actions, rewards, time, MDPs, policies, value
- Q-learning

Plan for today

Goal: Remind you about RL & sequential decision making

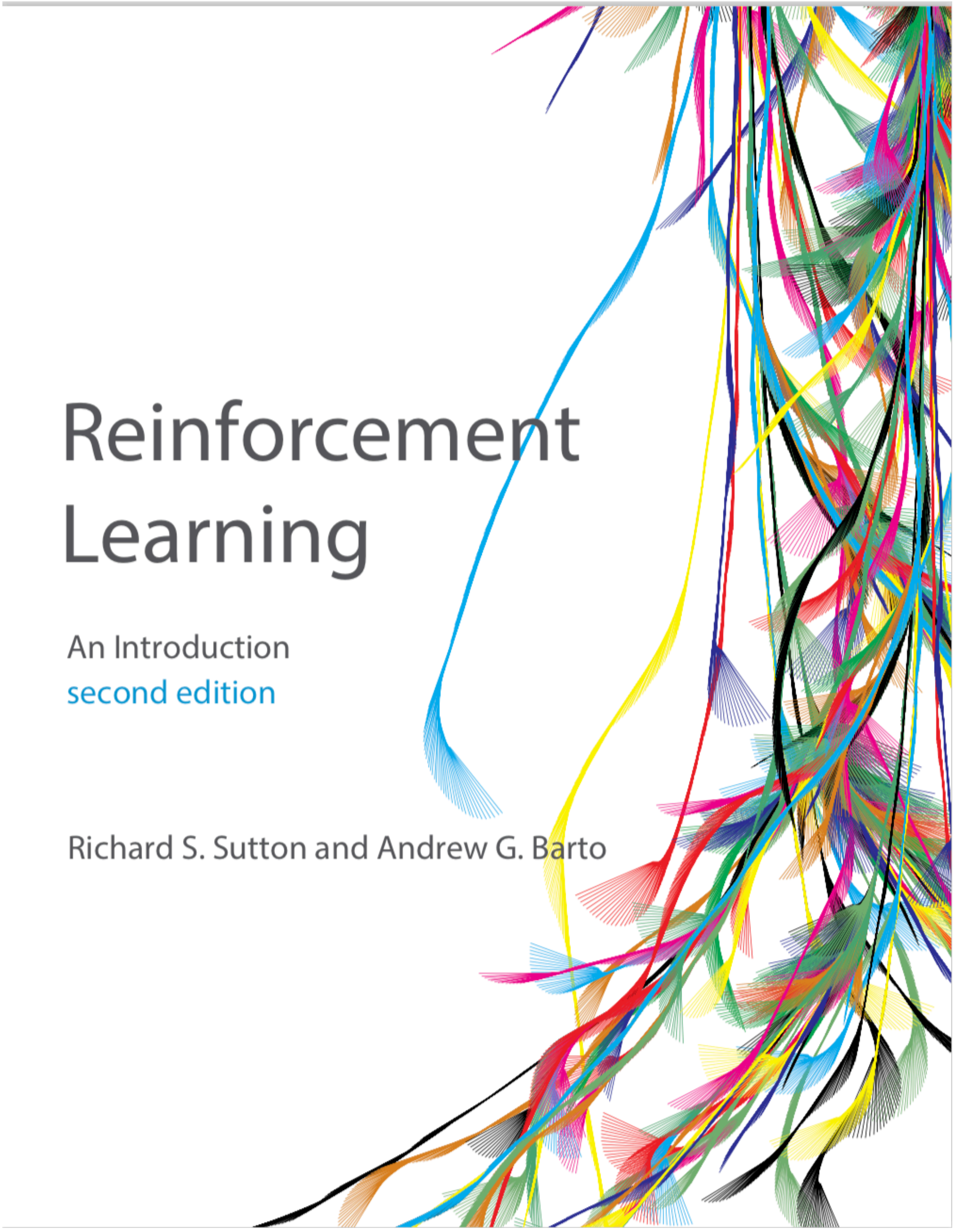
- The basics of RL:
  - states, actions, rewards, time, MDPs, policies, value
- Q-learning
- Function approximation in RL

Plan for today

Goal: Remind you about RL & sequential decision making

- The basics of RL:
  - states, actions, rewards, time, MDPs, policies, value
- Q-learning
- Function approximation in RL
- Comments throughout on open research challenges, particularly related to learning in the real world!

Many ways to learn about RL



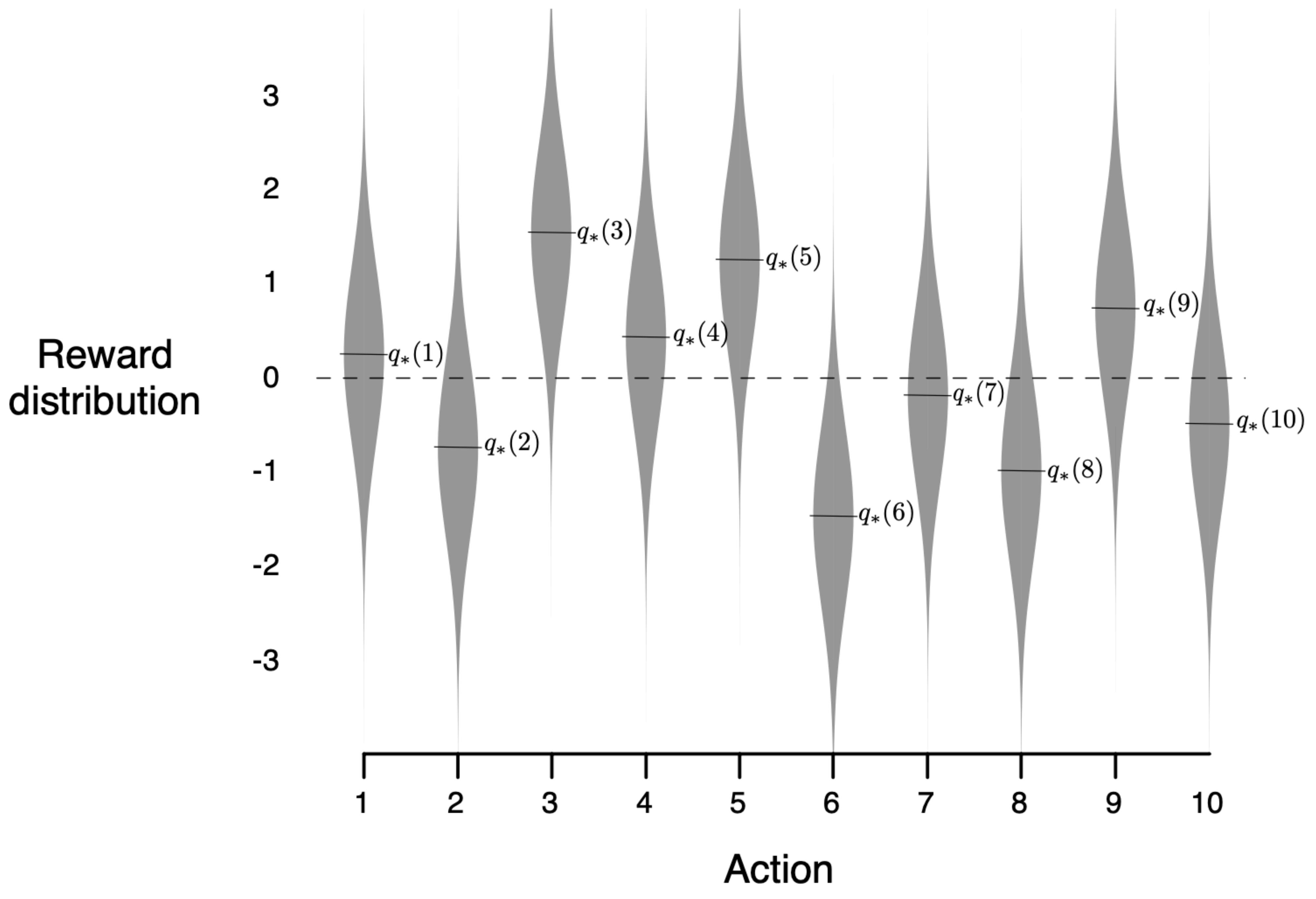
2nd edition: free and online

4 course RL specialization
(uab.ca/RLMOOC)

Key characteristics of RL

- **Evaluative feedback (reward)**
- Delayed consequences
- Must associate different actions with different situations
- Online and Incremental learning
- Need for trial and error, to explore as well as exploit
- Non-stationarity

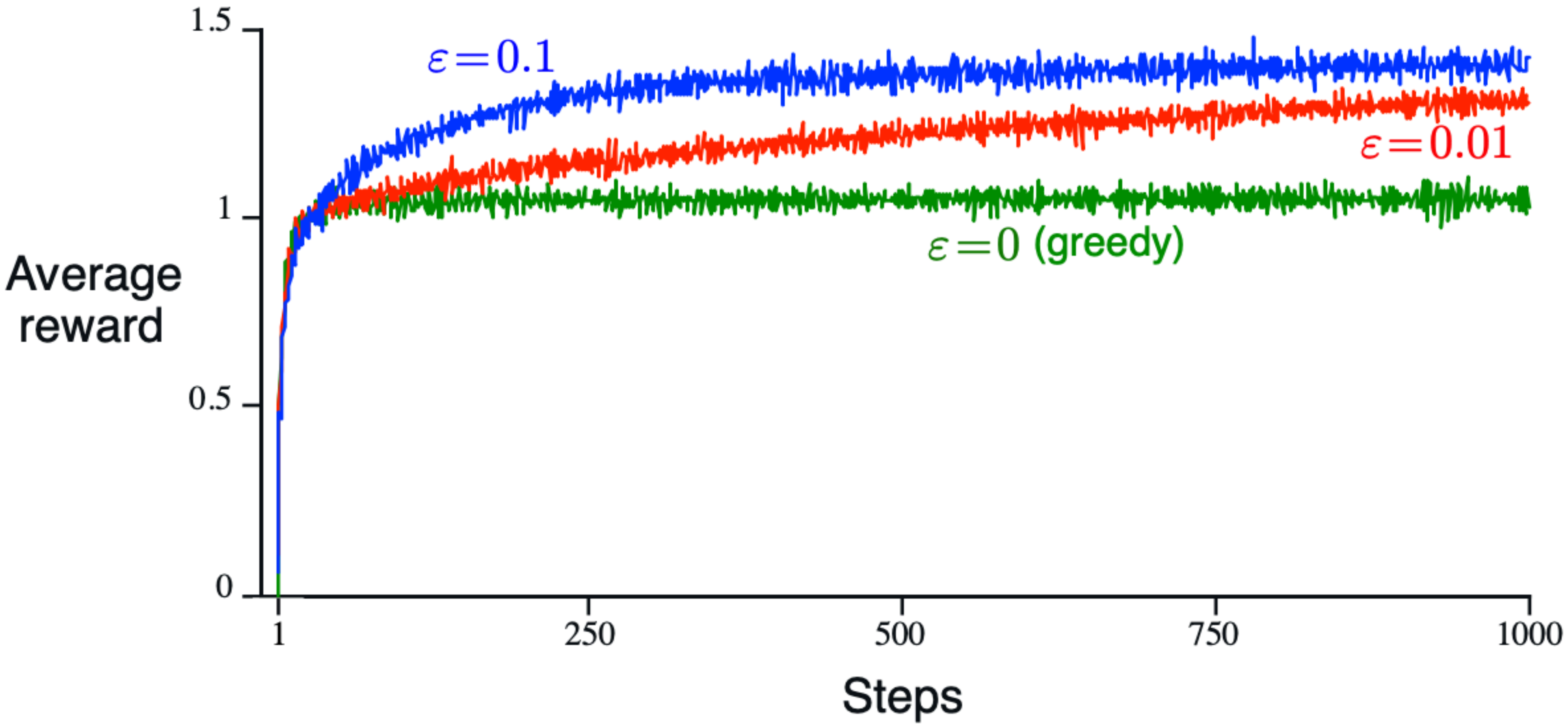
Multi-armed bandits

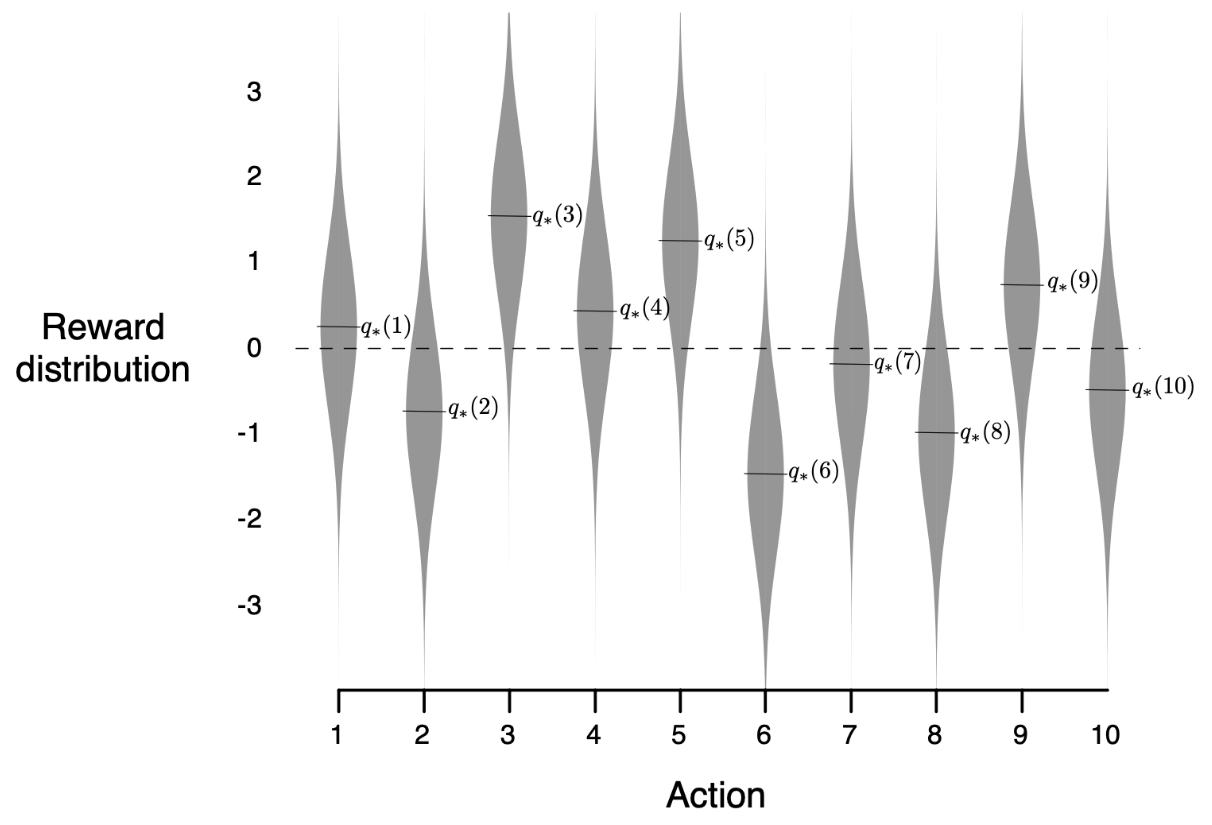


$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a]$$

- Estimate the value of each action in order to find the best
- Only get samples of the reward by trying an action: **rewards of arms not chosen are not revealed**
- Means we need to try each arm enough, but we also don't want to suffer too much loss of potential reward
  - **=> the exploration / exploitation tradeoff**

Learning in a multi-armed bandit





$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a]$$

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

$$A_t \doteq \underset{a}{\arg\max}\, Q_t(a)$$

Dimensions of learning revealed by the MAB problem

- The need to learn online and incrementally

$$NewEstimate \leftarrow OldEstimate + StepSize \Big[Target - OldEstimate \Big]$$

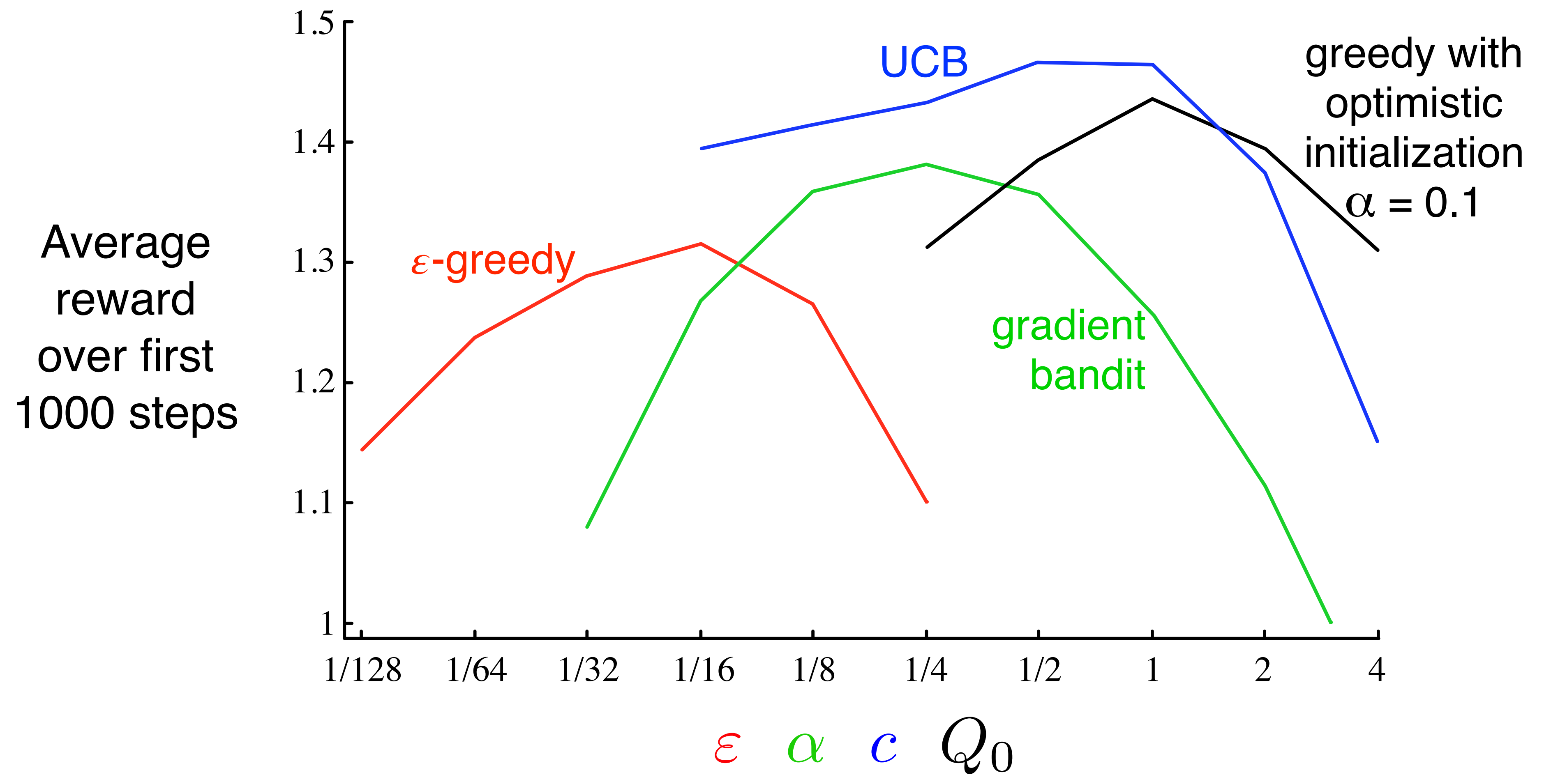
- Tracking and non-stationary tasks

$$Q_{n+1} \doteq Q_n + \alpha \Big[R_n - Q_n \Big]$$

- The role of initializing algorithms (e.g., optimistic init)

- Role of exploration algorithms (e.g., OI, e-greedy, UBC)

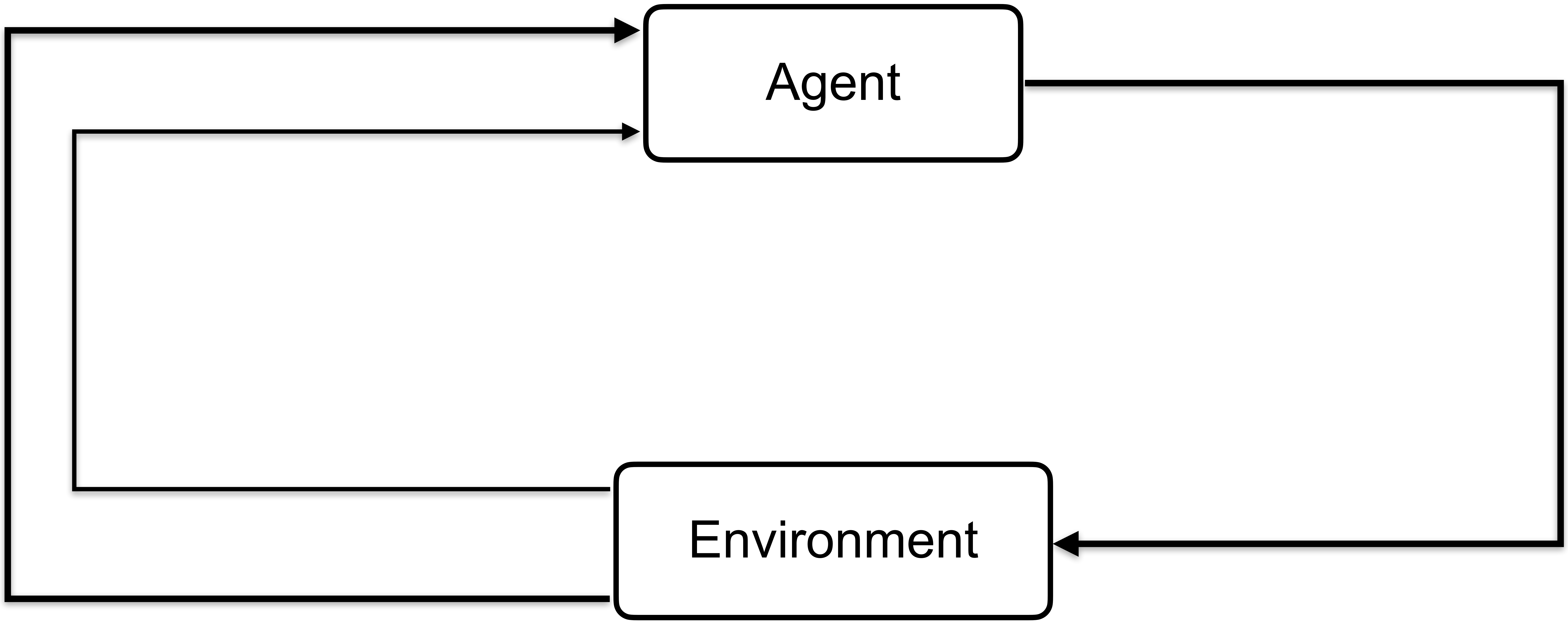
- Gradient methods



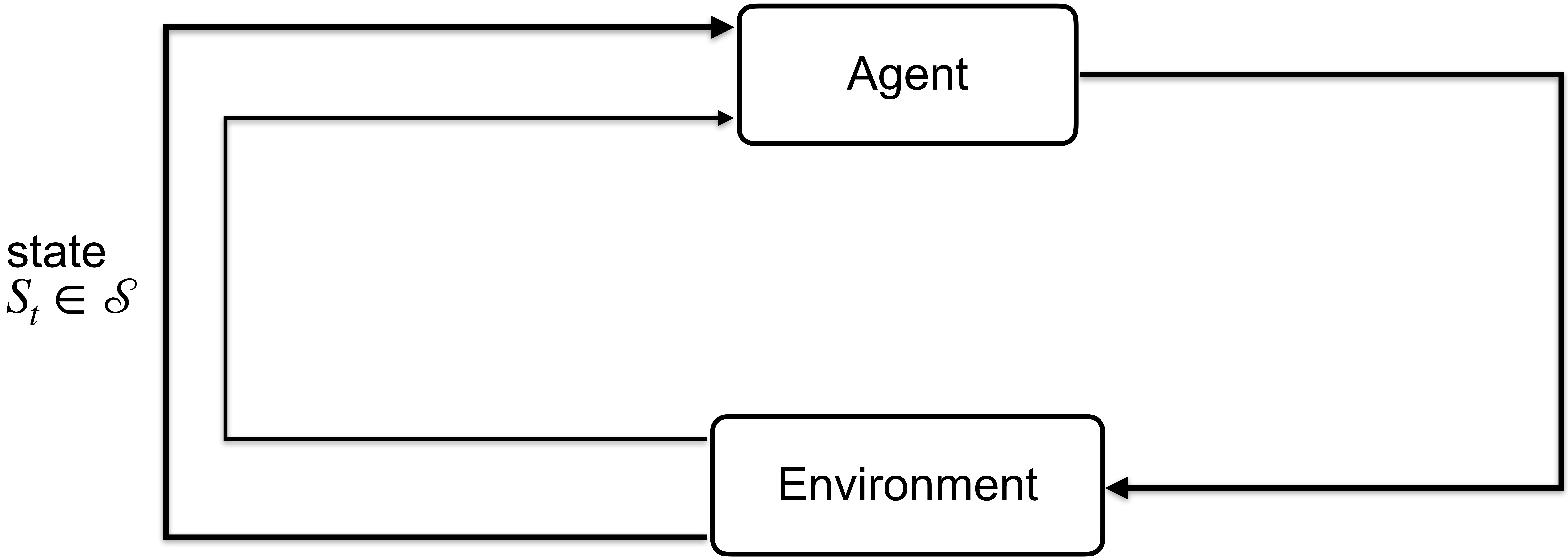
From Bandits to MDPs

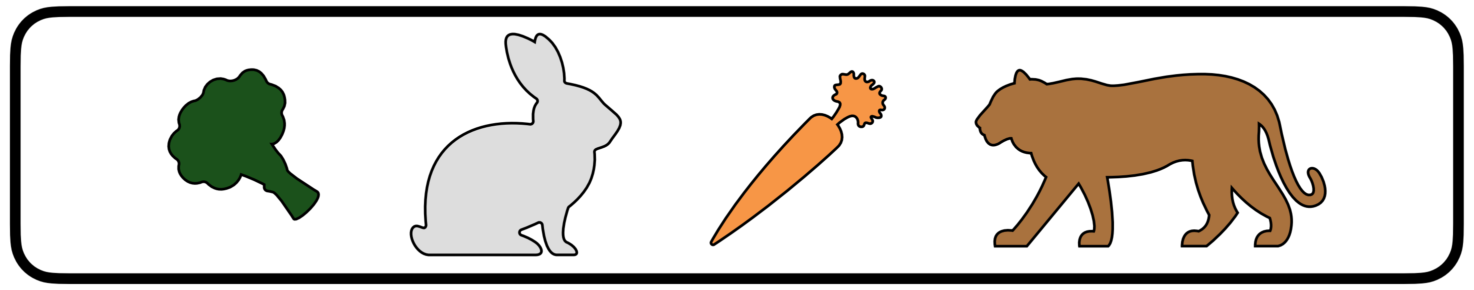
- The k-armed bandit task shares some of the same key characteristics of the RL problem:
  - **Evaluative feedback (reward)**
  - Online and incremental learning
  - Need for trial and error, to **explore** as well as exploit
  - Non-stationary???
- Let's see how Markov Decision Processes and the RL problem differ from Bandits

The RL Interface

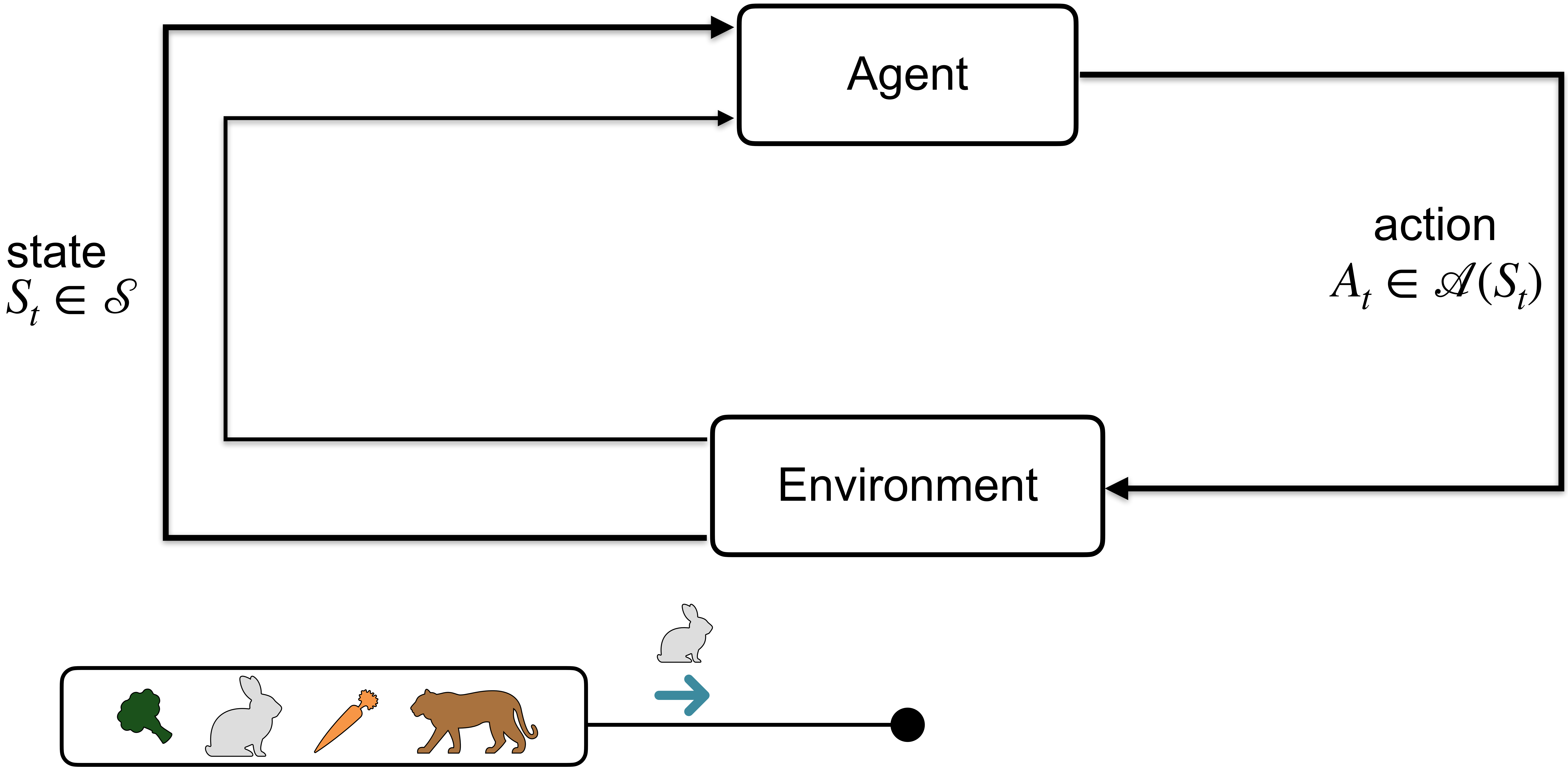


The RL Interface

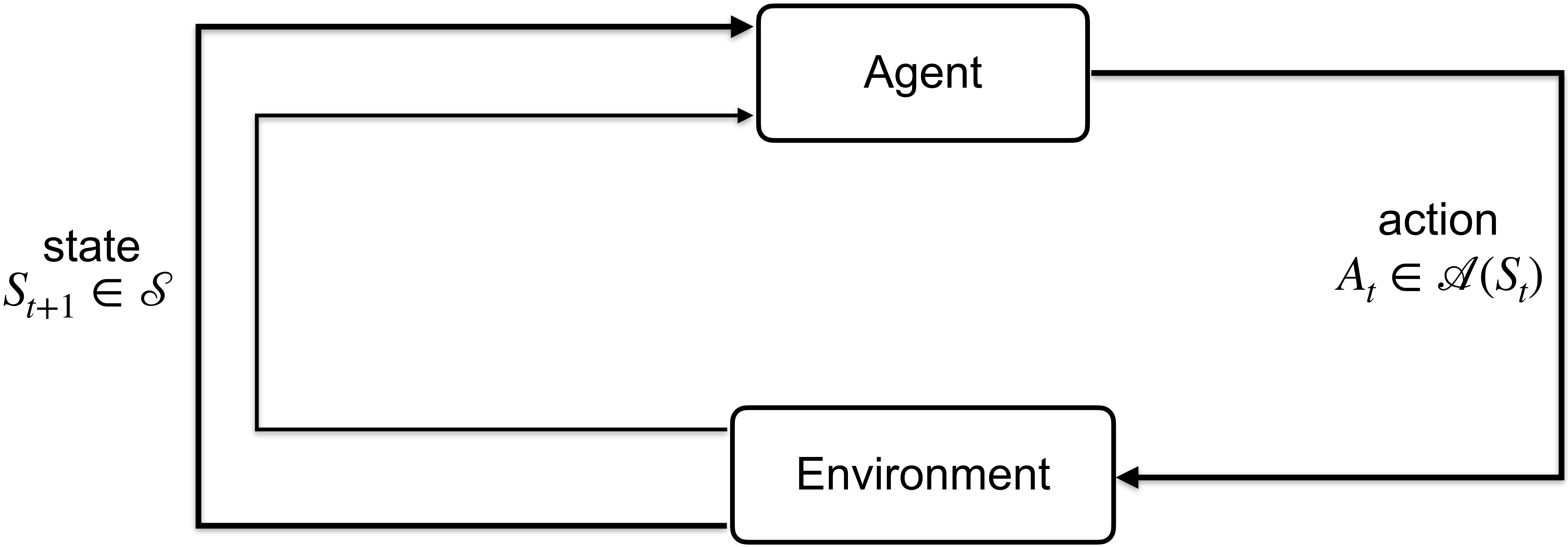


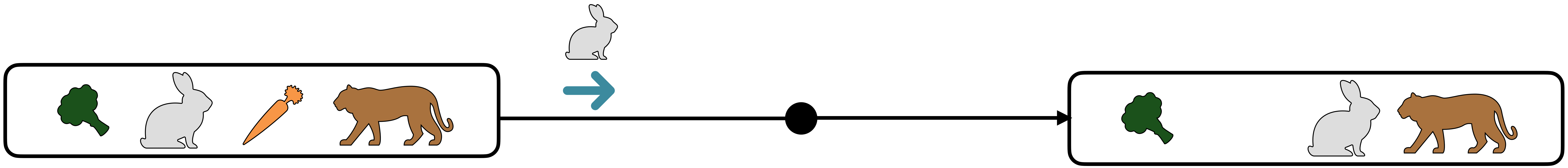


The RL Interface

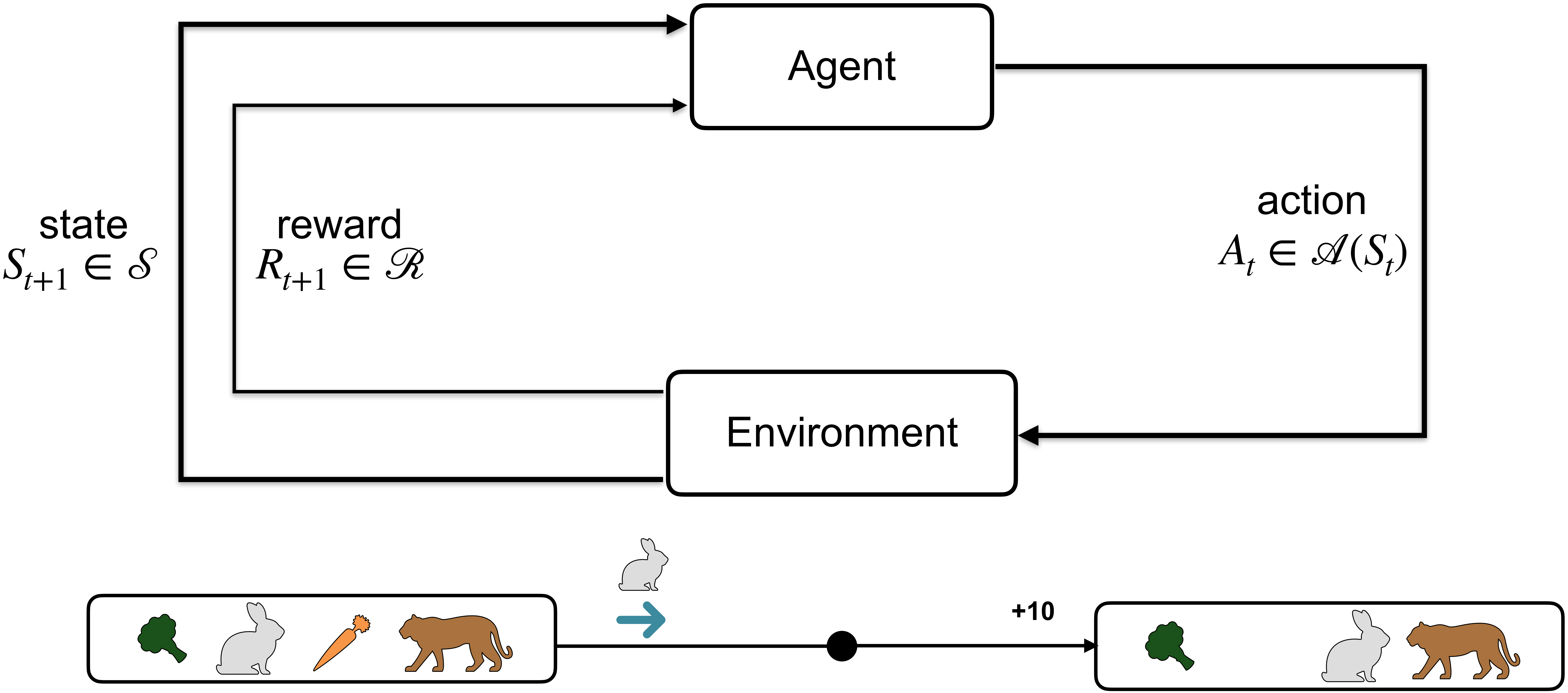


The RL Interface

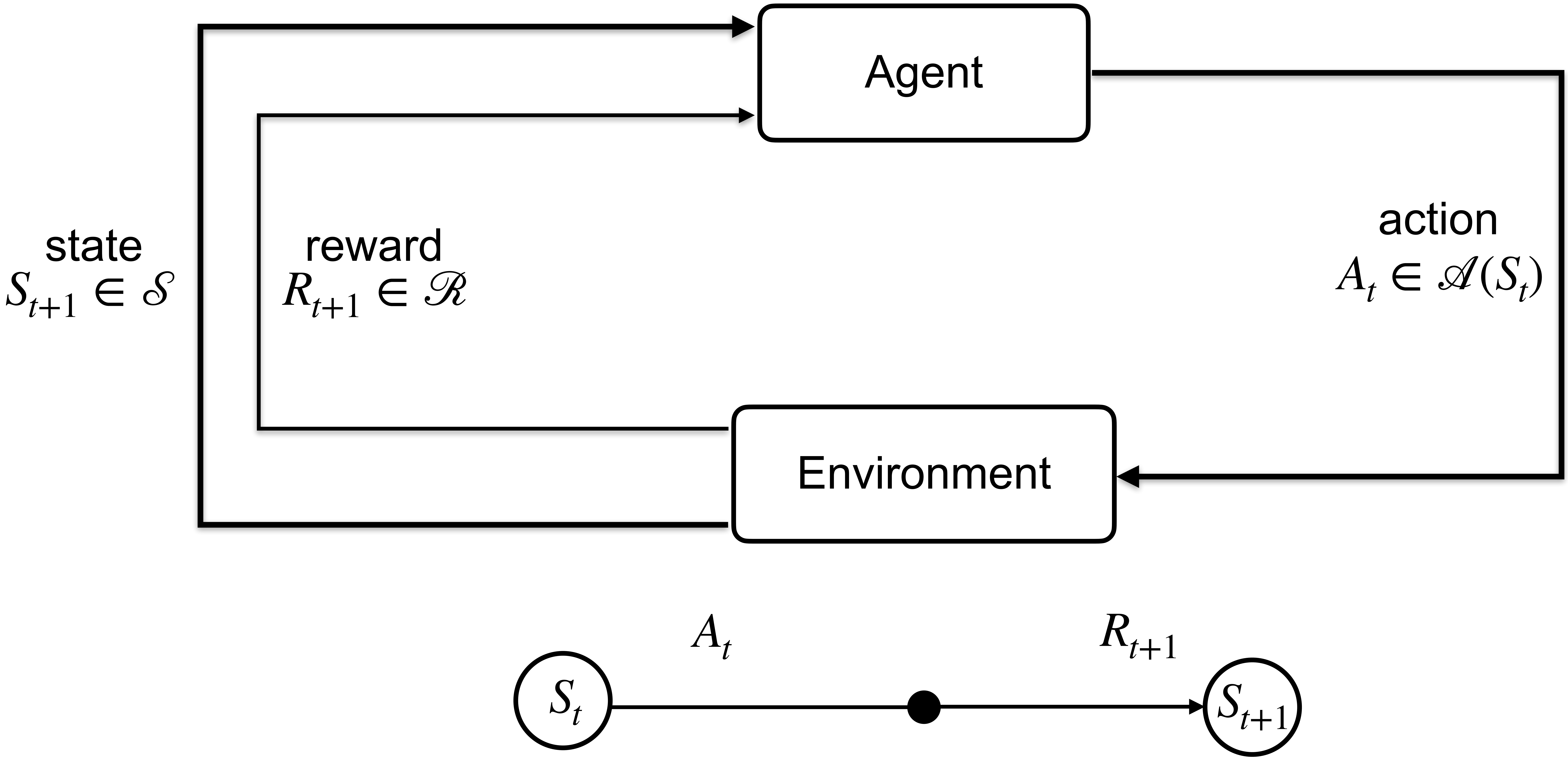




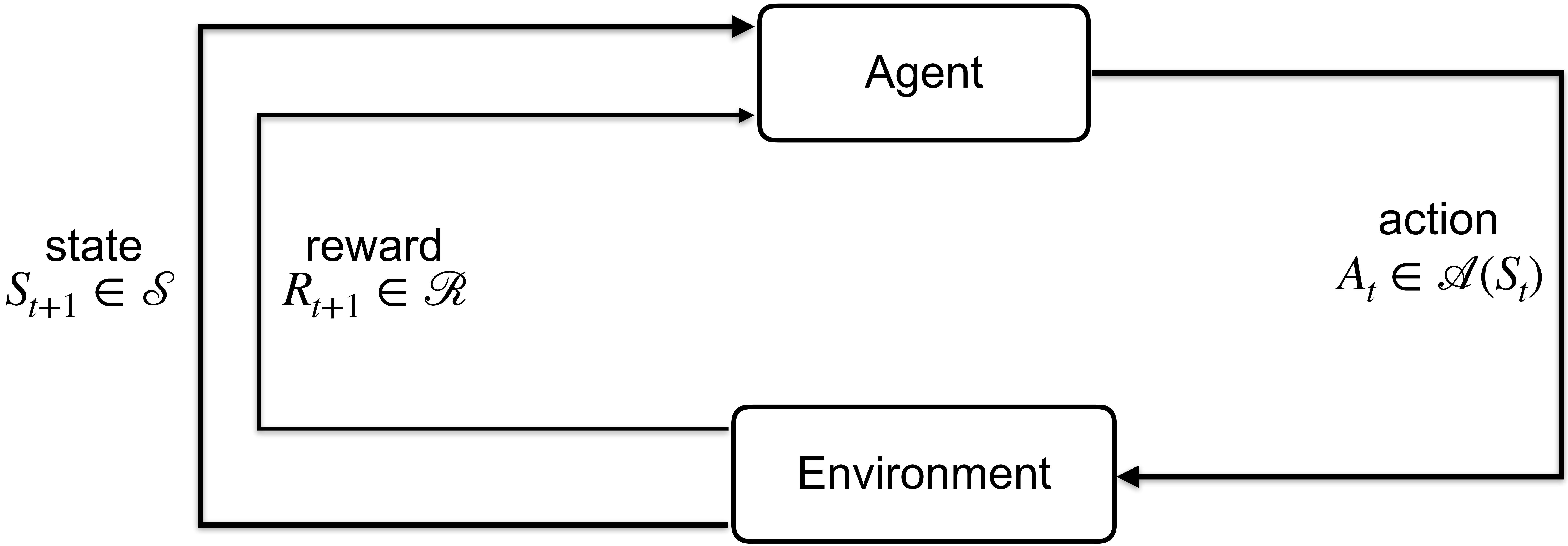
The RL Interface



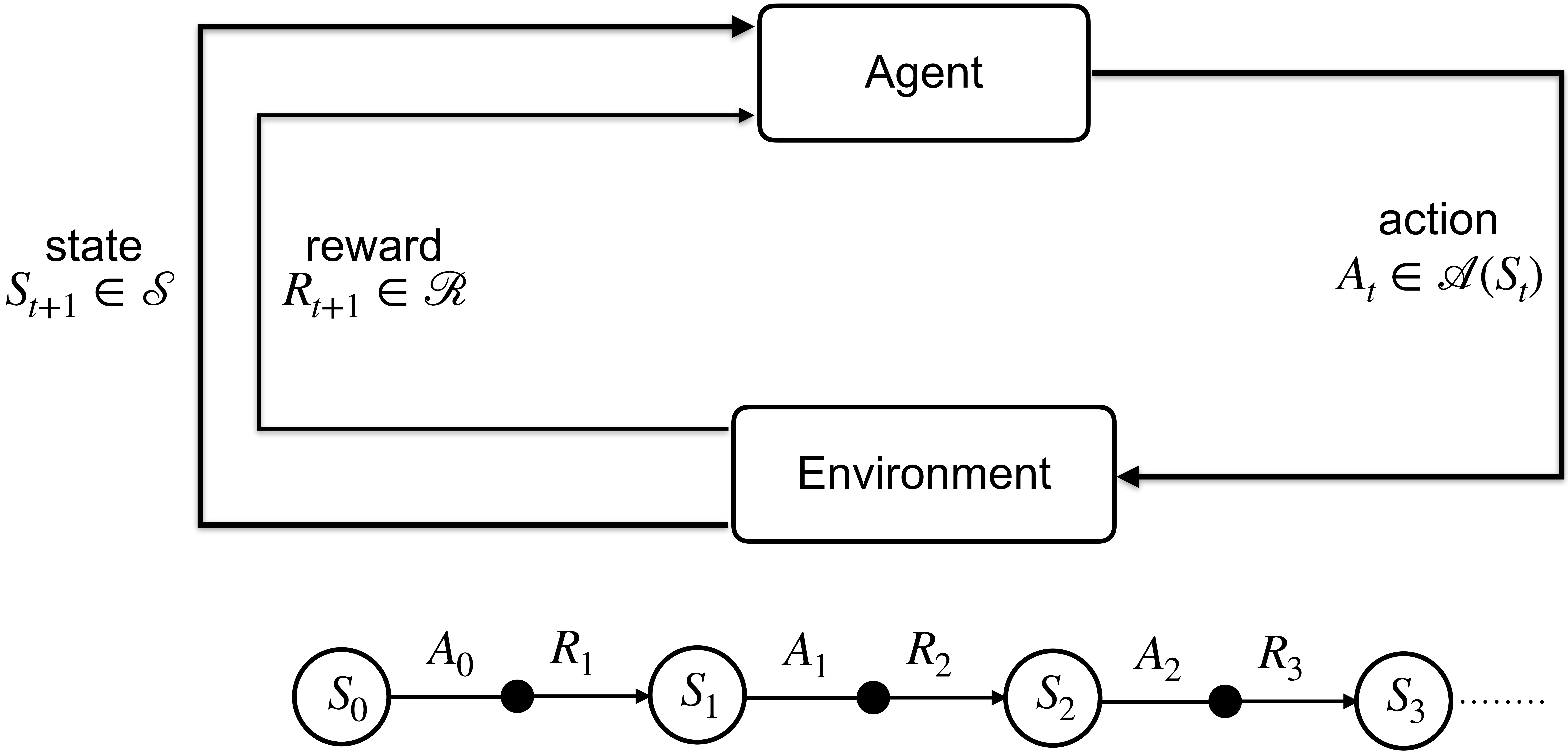
The RL Interface



The interaction generates a stream of experience!



The interaction generates a stream of experience!



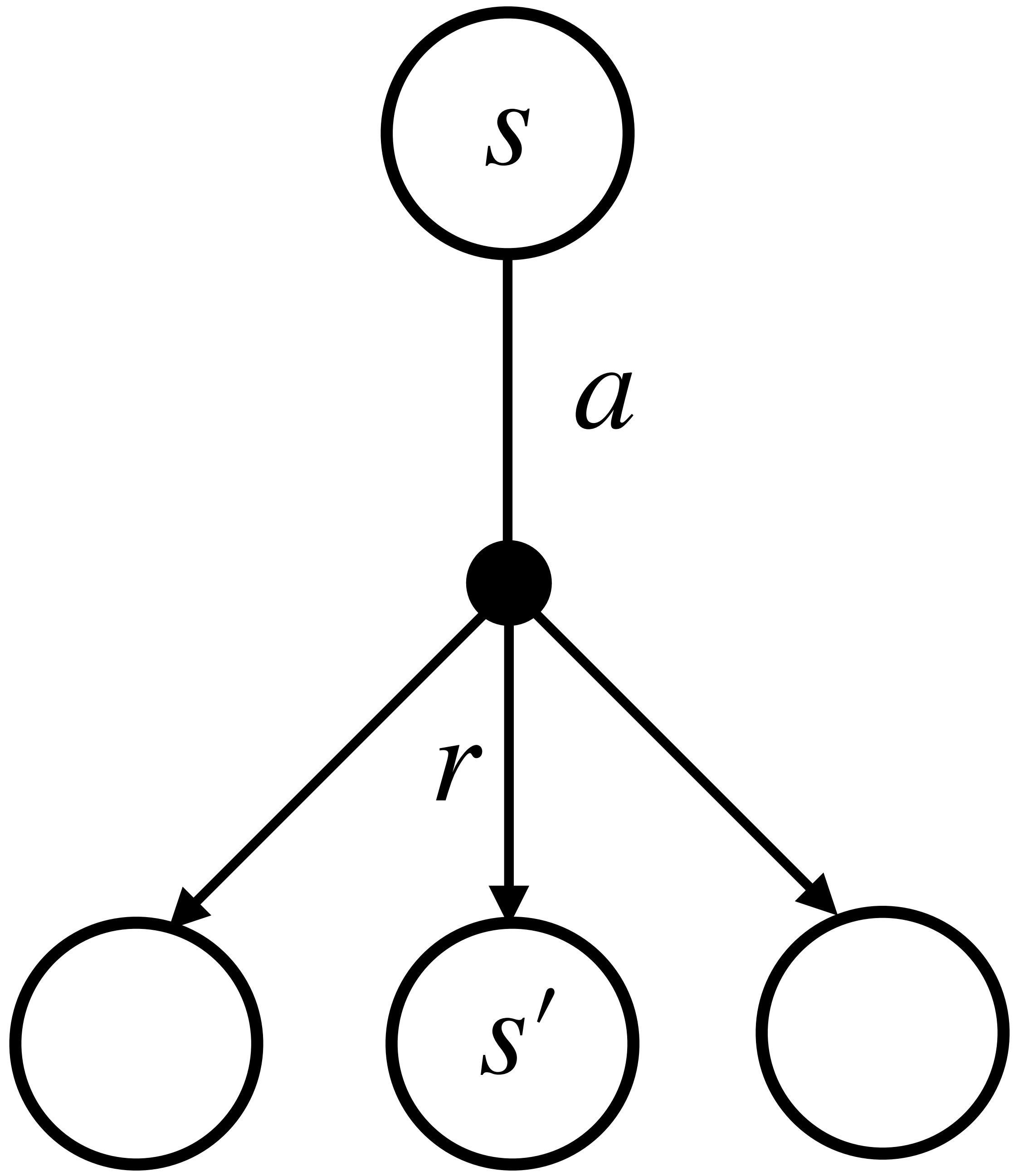
Finite Markov Decision Processes

- Environment may be unknown, stochastic and complex
  - we formalize this with the language of MDPs
- An RL problem is a finite MDP if:
  - the set of states, actions, and rewards are finite
  - there is a transition function that describes the probabilities of all possible next state S', and reward R
    - the state satisfies the Markov Property

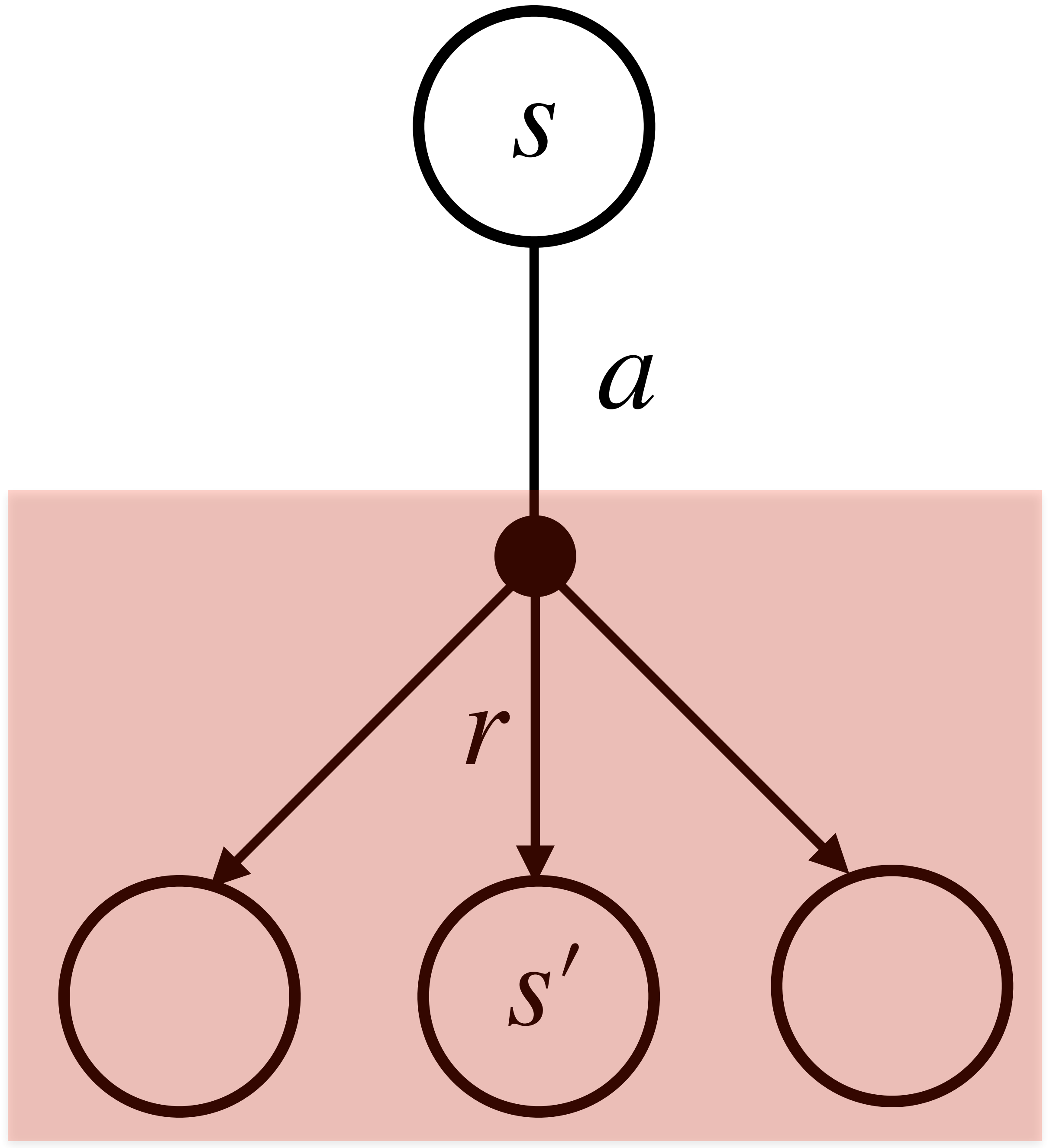
Finite Markov Decision Processes

- Environment may be unknown, stochastic and complex
  - we formalize this with the language of MDPs
- An RL problem is a finite MDP if:
  - the set of states, actions, and rewards are finite
  - there is a transition function that describes the probabilities of all possible next state S', and reward R
    - the state satisfies the Markov Property

The dynamics of an MDP

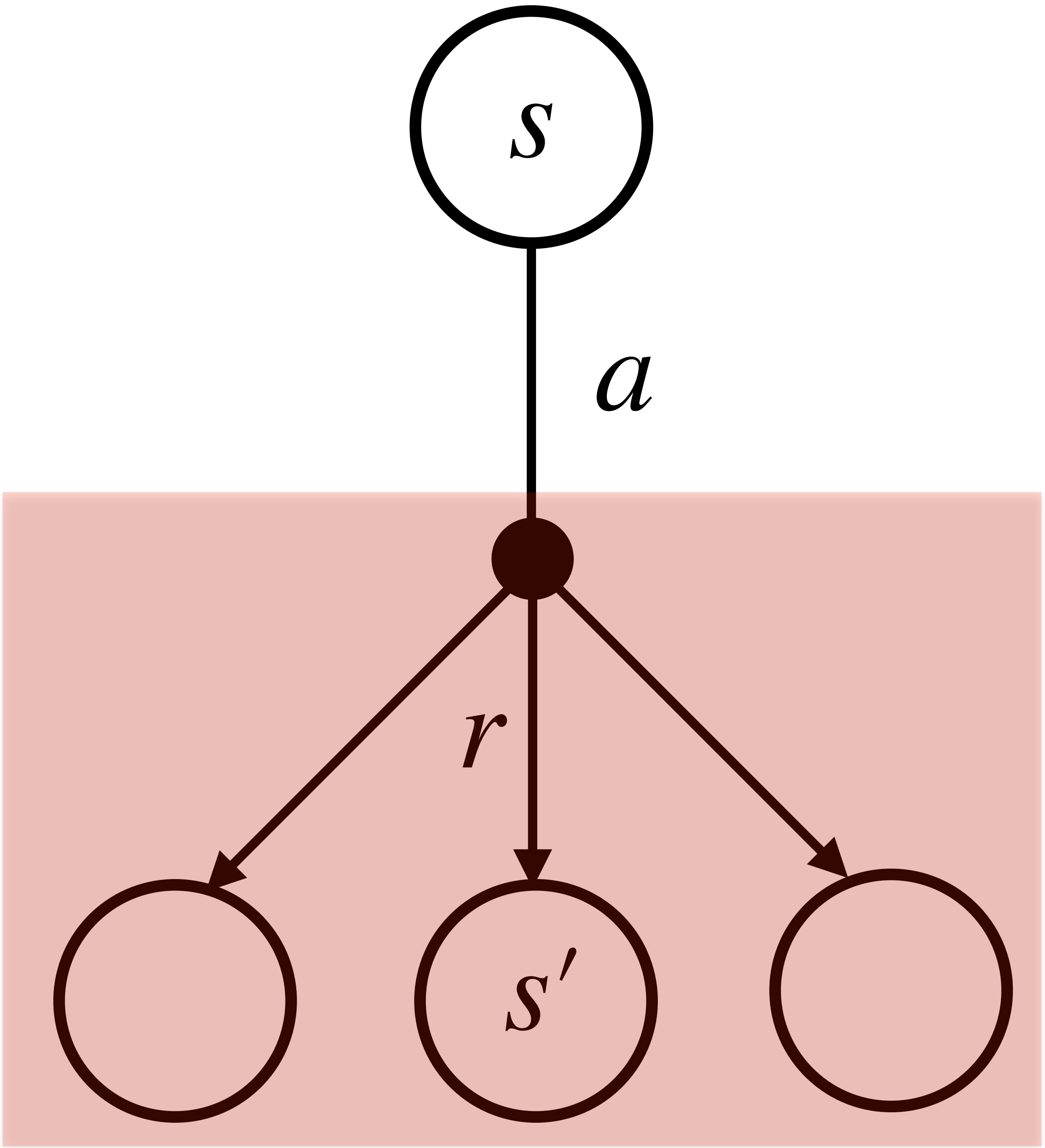


The dynamics of an MDP



$$p(s', r \mid s, a)$$

The dynamics of an MDP

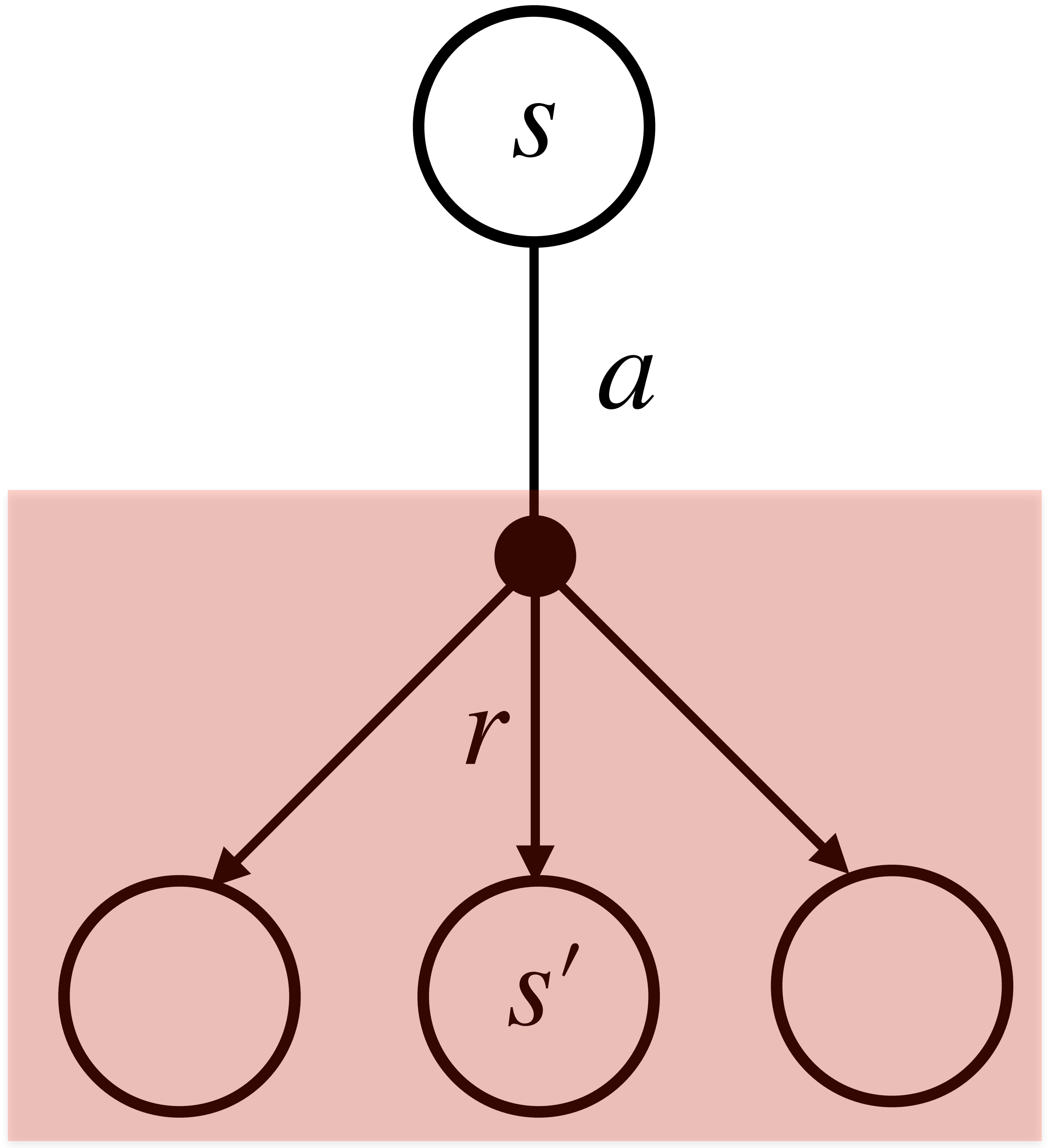


$$p(s', r \mid s, a)$$

$$p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

The dynamics of an MDP



$$p(s', r \mid s, a)$$

$$p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

Remembering earlier states would not improve predictions about the future

The goal of life: more reward

The goal of life: more reward

- The agent's objective is to maximize future total reward

The goal of life: more reward

- The agent's objective is to maximize future total reward
- The scalar return: $G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \dots$

The goal of life: more reward

- The agent's objective is to maximize future total reward
- The scalar return: $G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ldots$
- But, the agent's interaction may never end, so we discount rewards far into the future

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{k-1} R_{t+k} + \ldots$$

$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

The goal of life: more reward

- The agent's objective is to maximize future total reward
- The scalar return: $G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \dots$
- But, the agent's interaction may never end, so we discount rewards far into the future

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{k-1} R_{t+k} + \dots$$

$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Finite as long as $0 \leq \gamma < 1$ and rewards are bounded

The goal of life: more reward

- The agent's objective is to maximize future total reward
- The scalar return: $G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ldots$
- But, the agent's interaction may never end, so we discount rewards far into the future

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{k-1} R_{t+k} + \ldots$$

$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Finite as long as $0 \leq \gamma < 1$ and rewards are bounded

- In each state, the agent should choose the action that results in the highest return, **in expectation—why the expectation?**

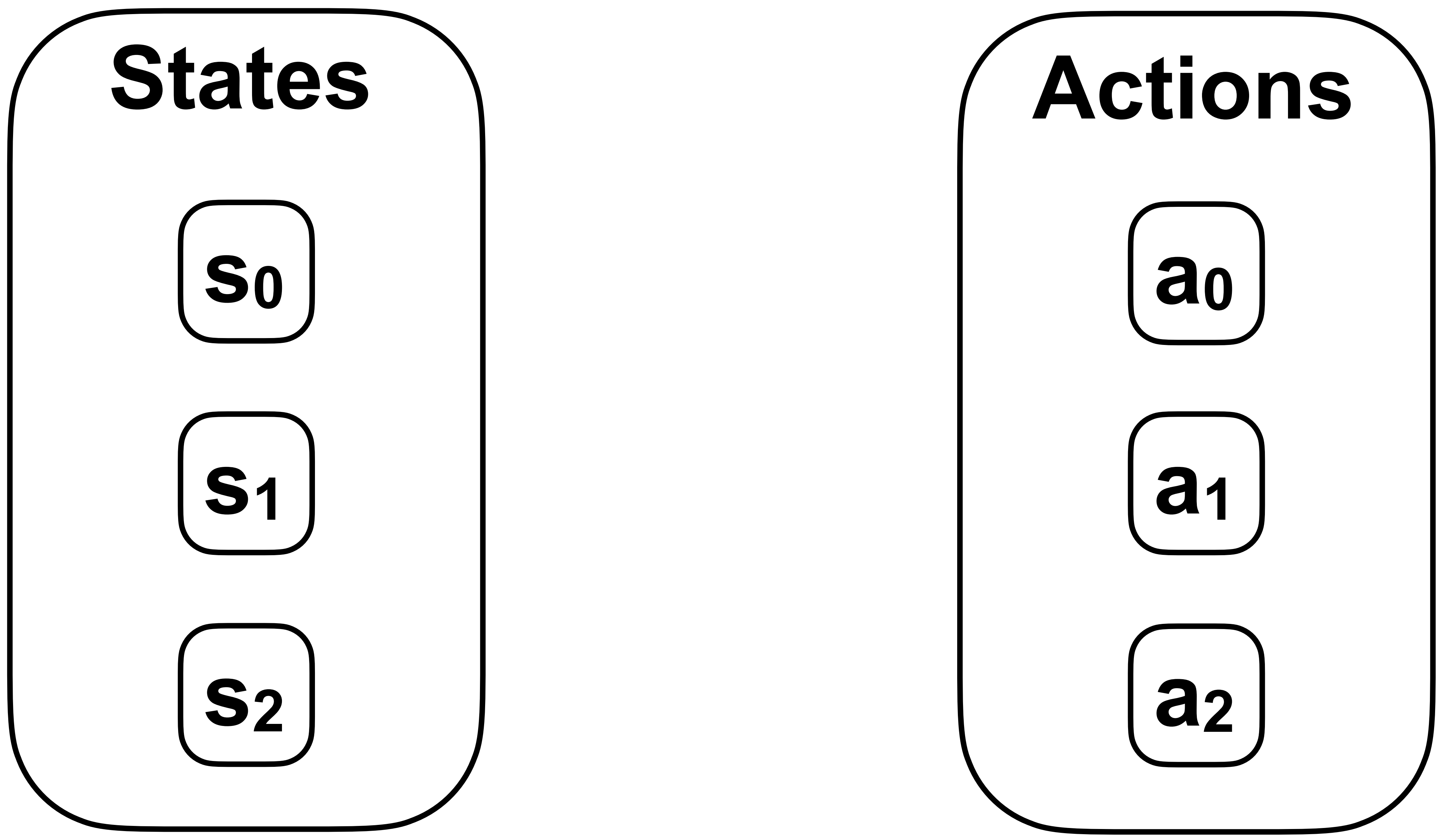
Key characteristics of RL

- ☑ Evaluative feedback (reward)
- ☑ Delayed consequences
- **Must associate different actions with different situations**
- Online and Incremental learning
- Need for trial and error, to explore as well as exploit
- Non-stationarity

Policies

- Deterministic policy

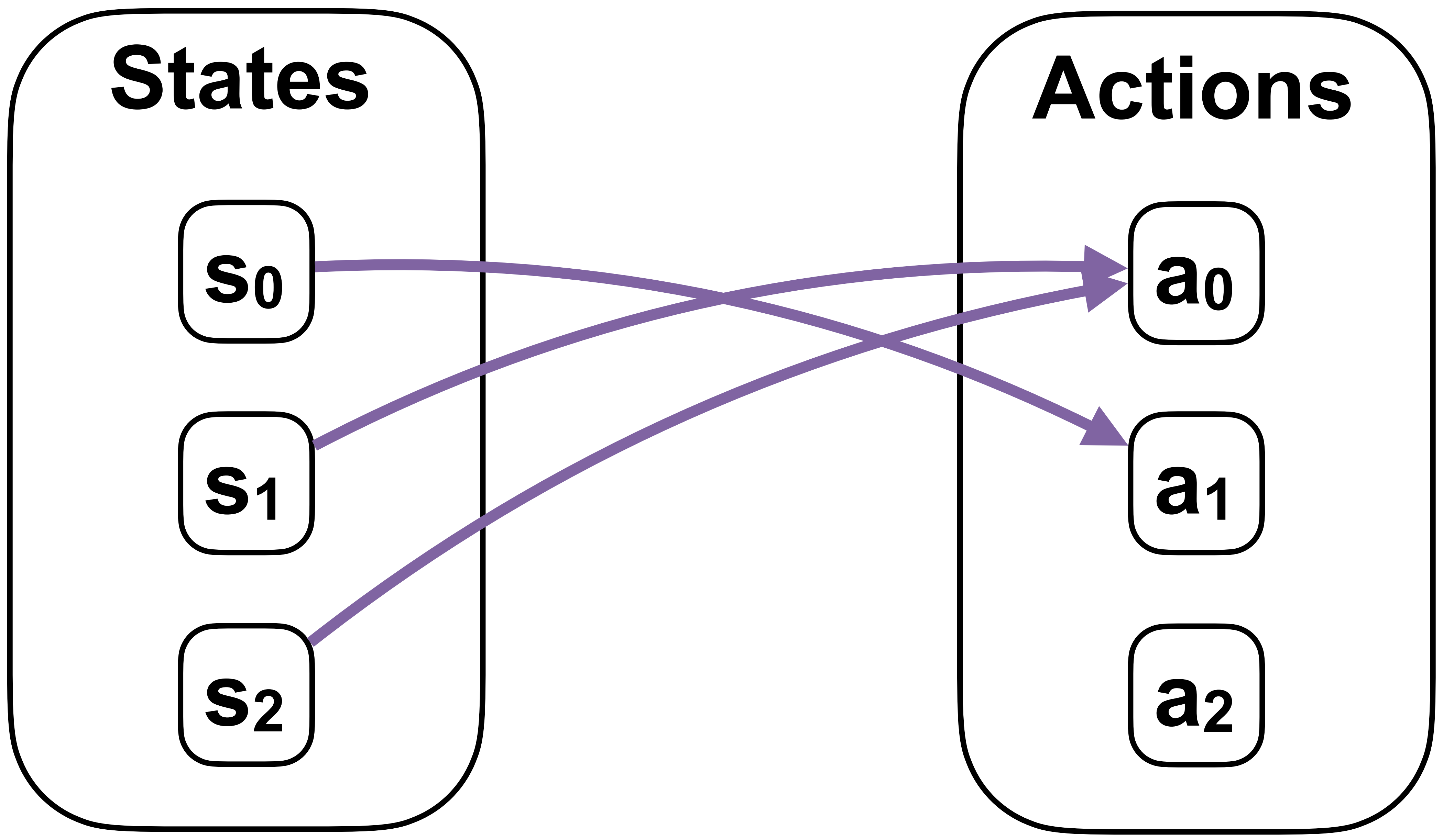
$$\pi(s) = a$$



Policies

- Deterministic policy

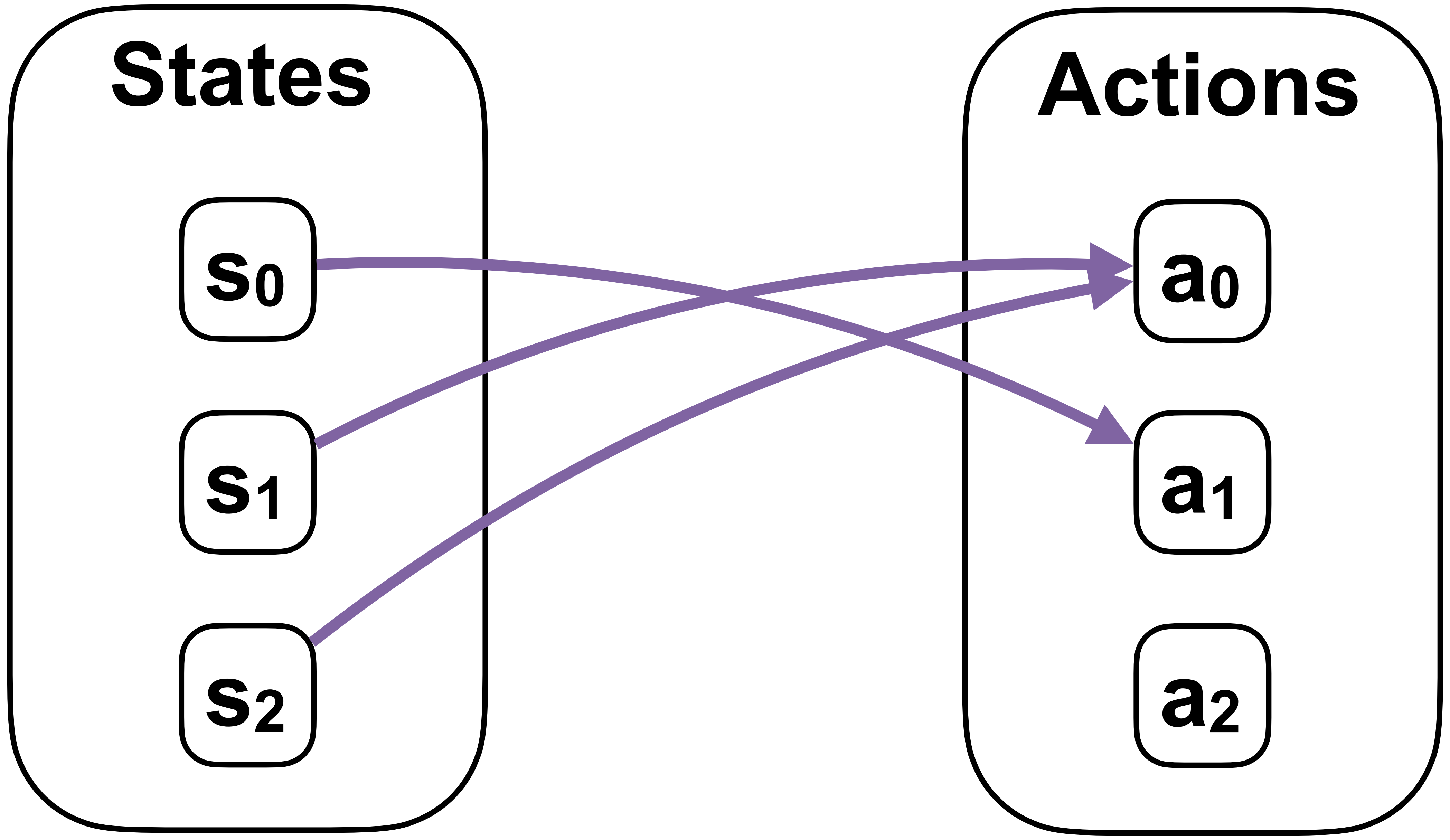
$$\pi(s) = a$$



Policies

- Deterministic policy

$$\pi(s) = a$$



State	Action
s_0	a_1
s_1	a_0
s_2	a_0

Policies

- Stochastic policy:

$$\pi(a \mid s)$$

- where $\sum_{a \in \mathscr{A}(s)} \pi(a \mid s) = 1$

- and $\pi(a \mid s) \geq 0$

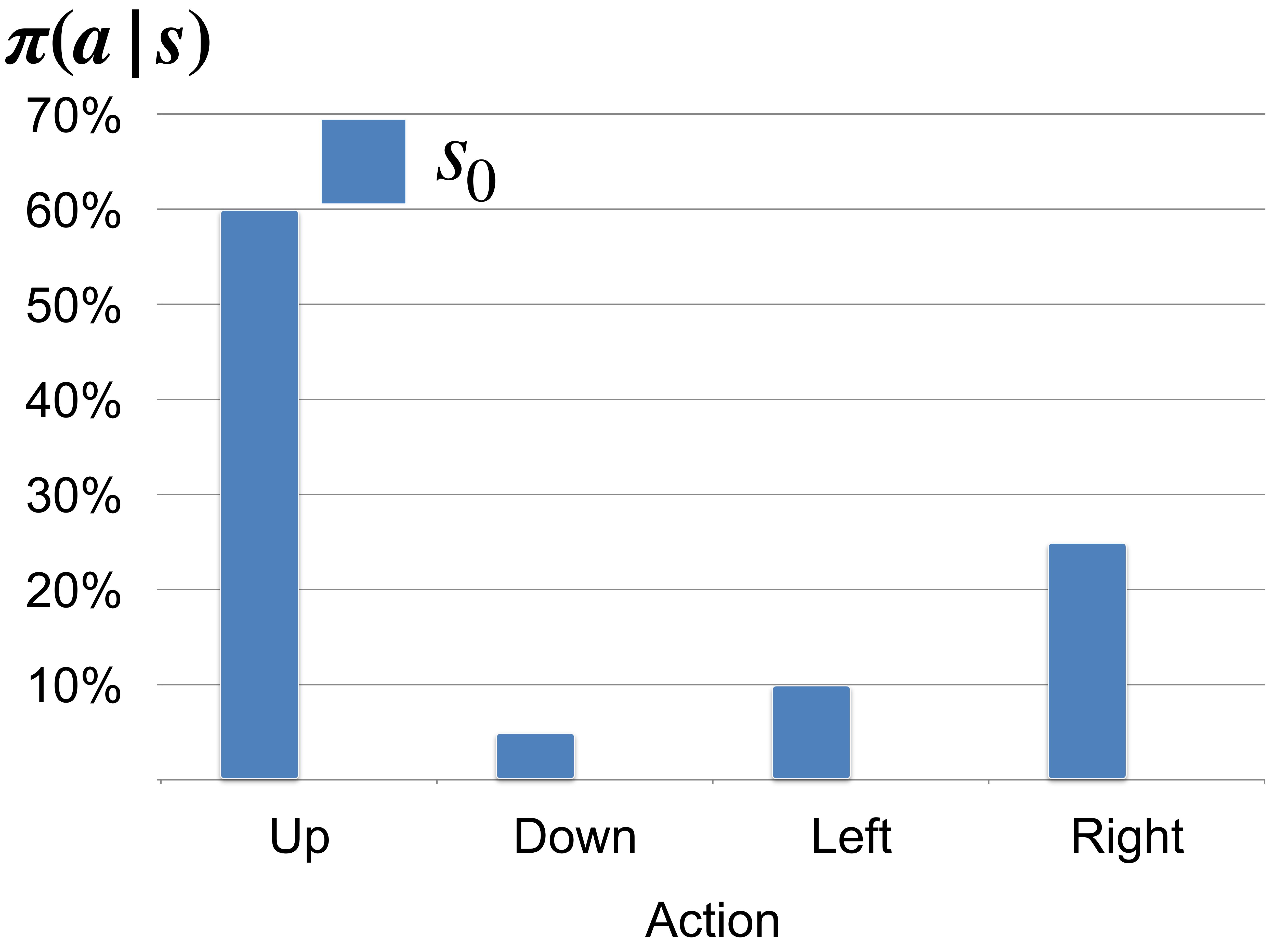
Policies

- Stochastic policy:

$$\pi(a \mid s)$$

- where $\sum_{a \in \mathcal{A}(s)} \pi(a \mid s) = 1$

- and $\pi(a \mid s) \geq 0$



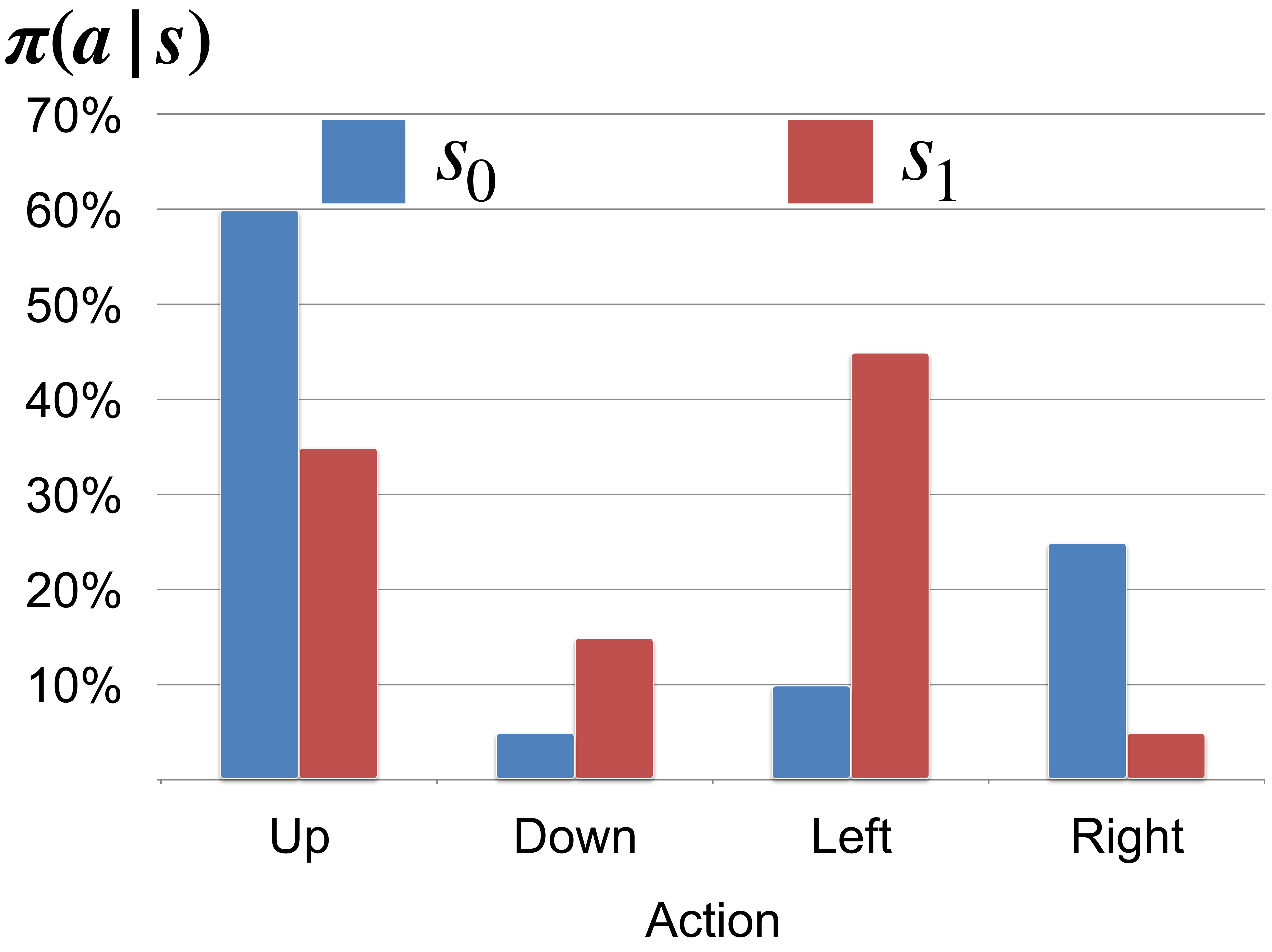
Policies

- Stochastic policy:

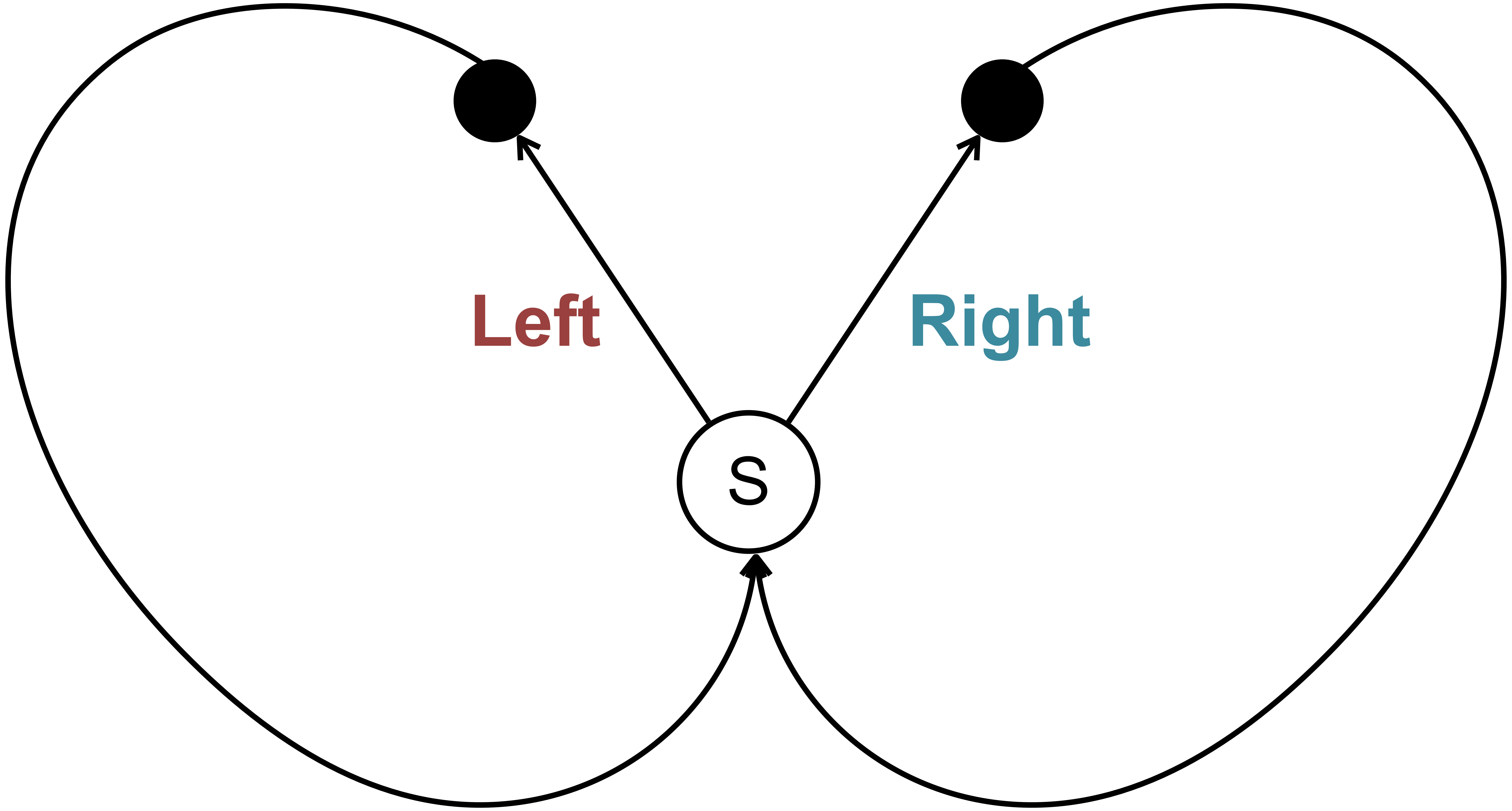
$$\pi(a \mid s)$$

- where $\sum_{a \in \mathscr{A}(s)} \pi(a \mid s) = 1$

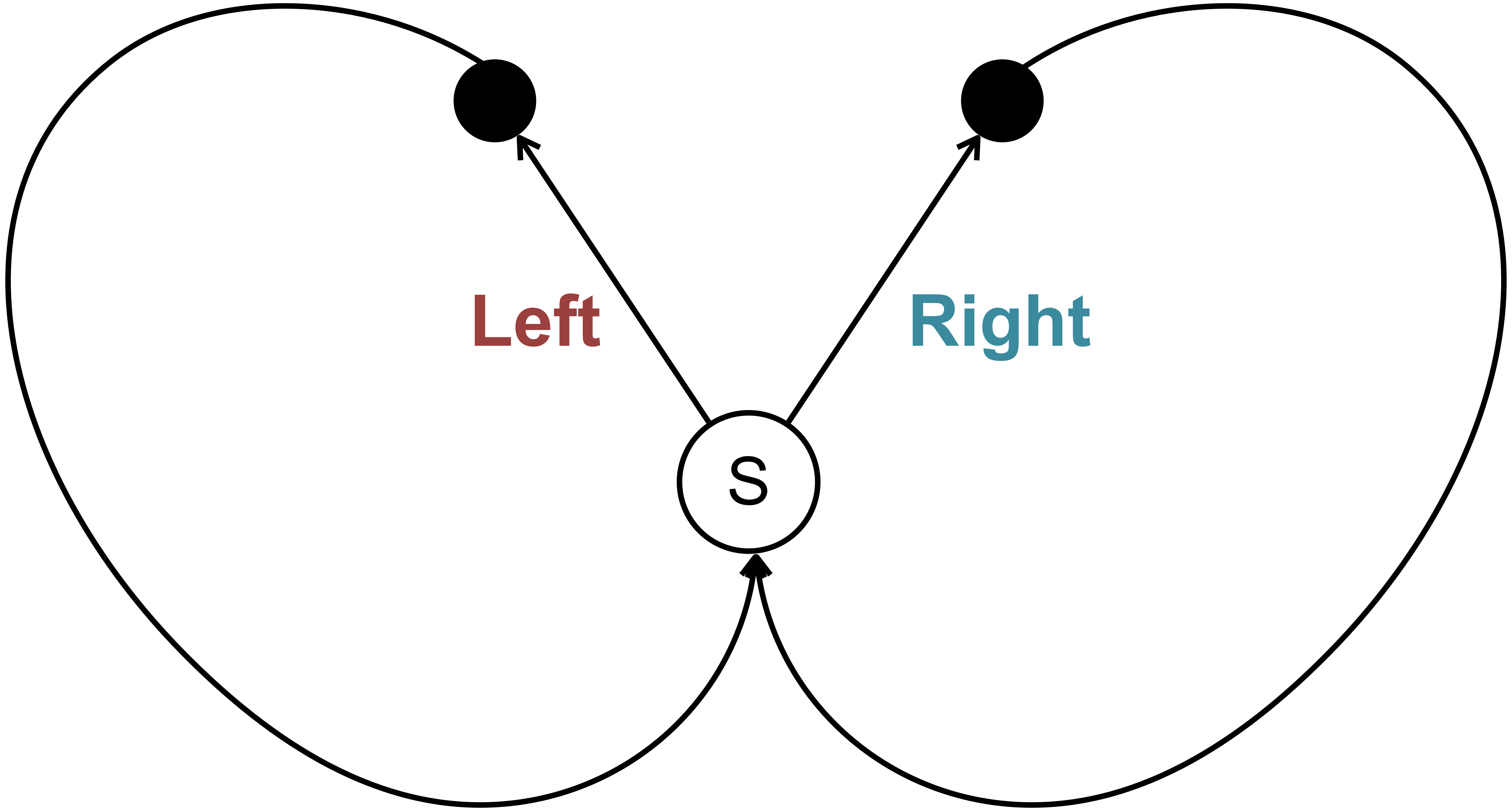
- and $\pi(a \mid s) \geq 0$



Valid and invalid policies



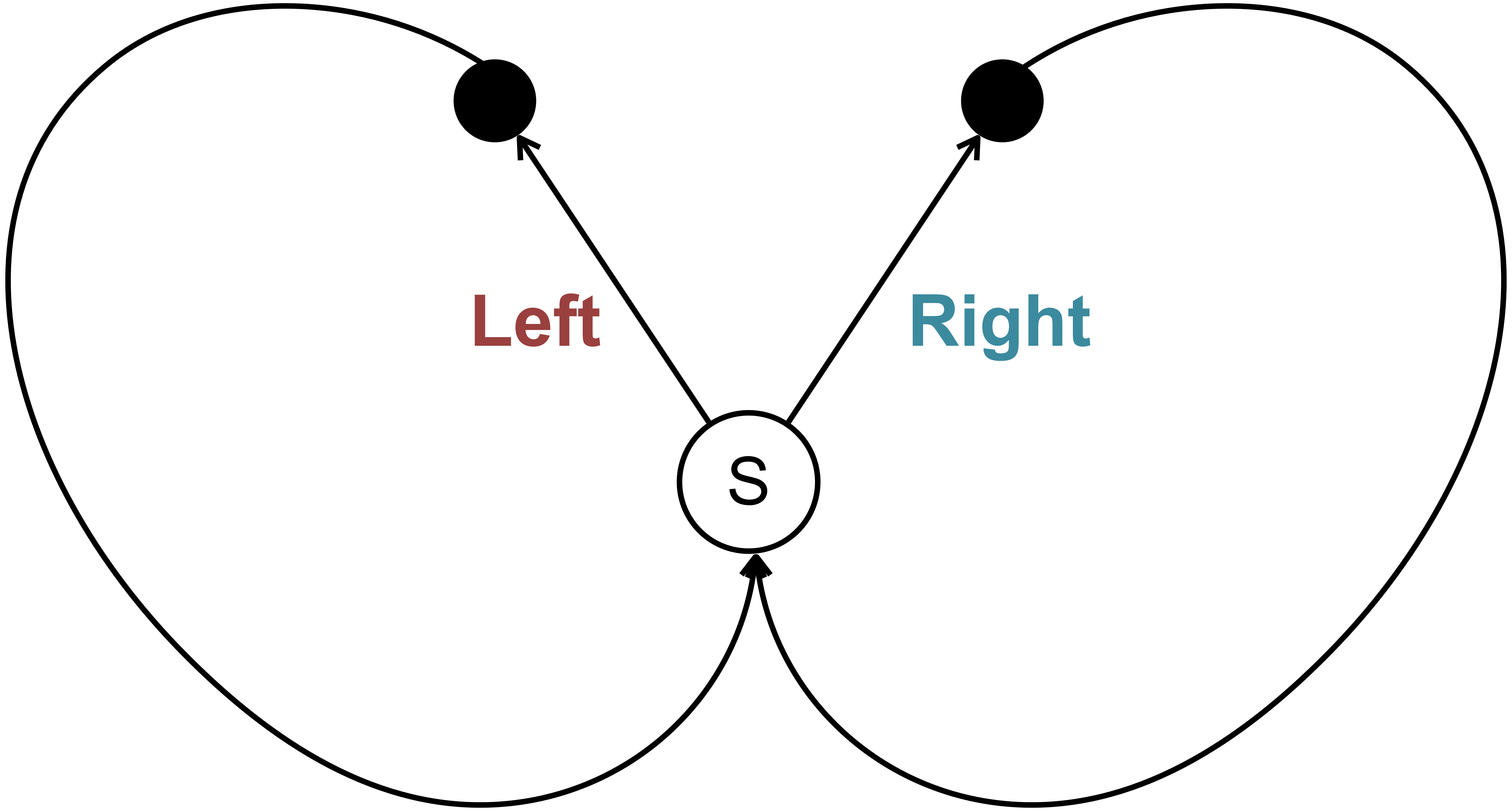
Valid and invalid policies



1: Left with 50% probability
and
Right with 50% probability

L L R L R L R R R ...

Valid and invalid policies



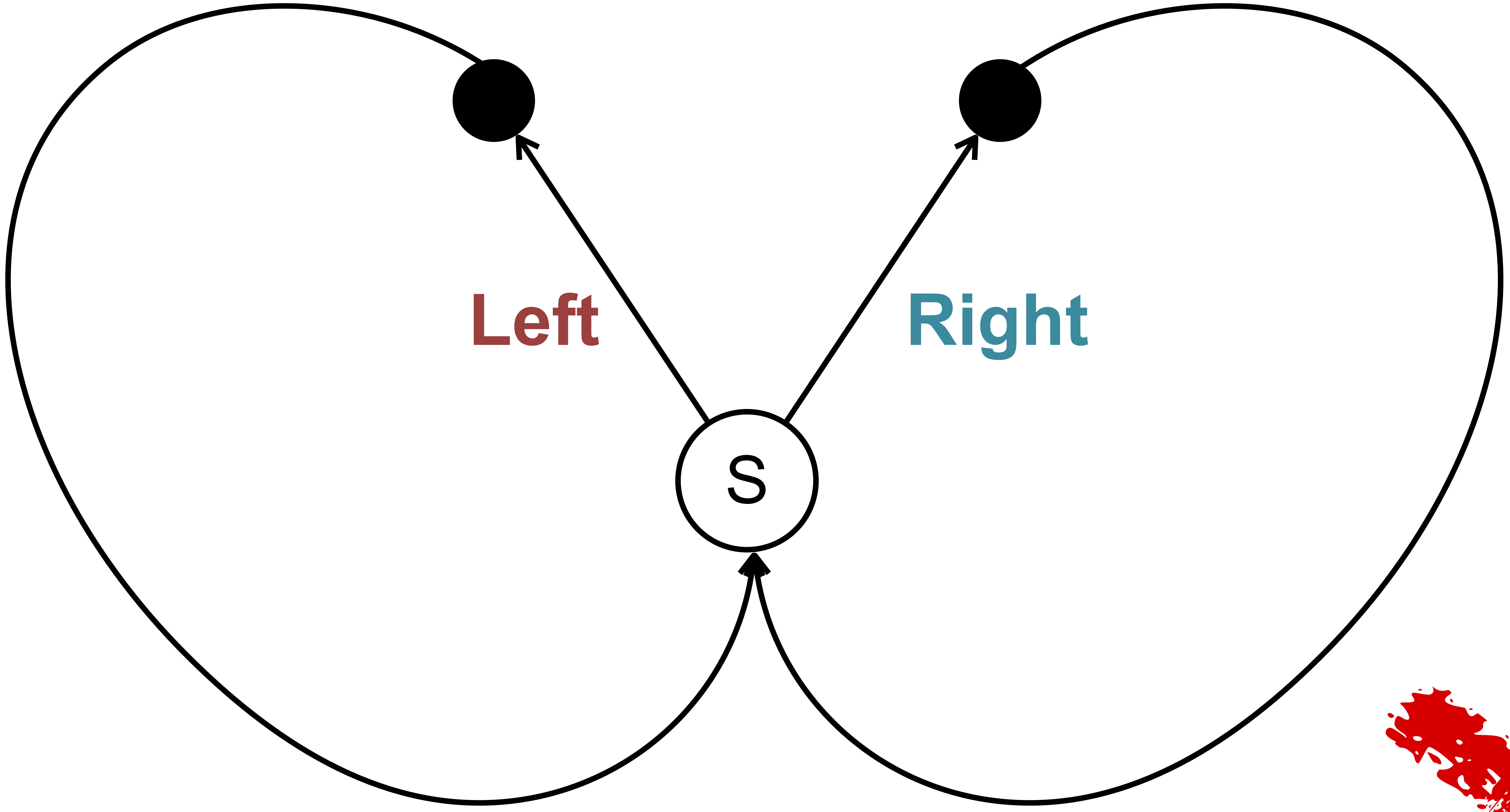
1: Left with 50% probability
and
Right with 50% probability

L L R L R L R R R ...

2: Alternate Left and Right

L R L R L R L R L ...

Valid and invalid policies



1: Left with 50% probability
and
Right with 50% probability

L L R L R L R R R ...

2: Alternate Left and Right

L R L R L R L R L ...

Markov Property

Action-value functions

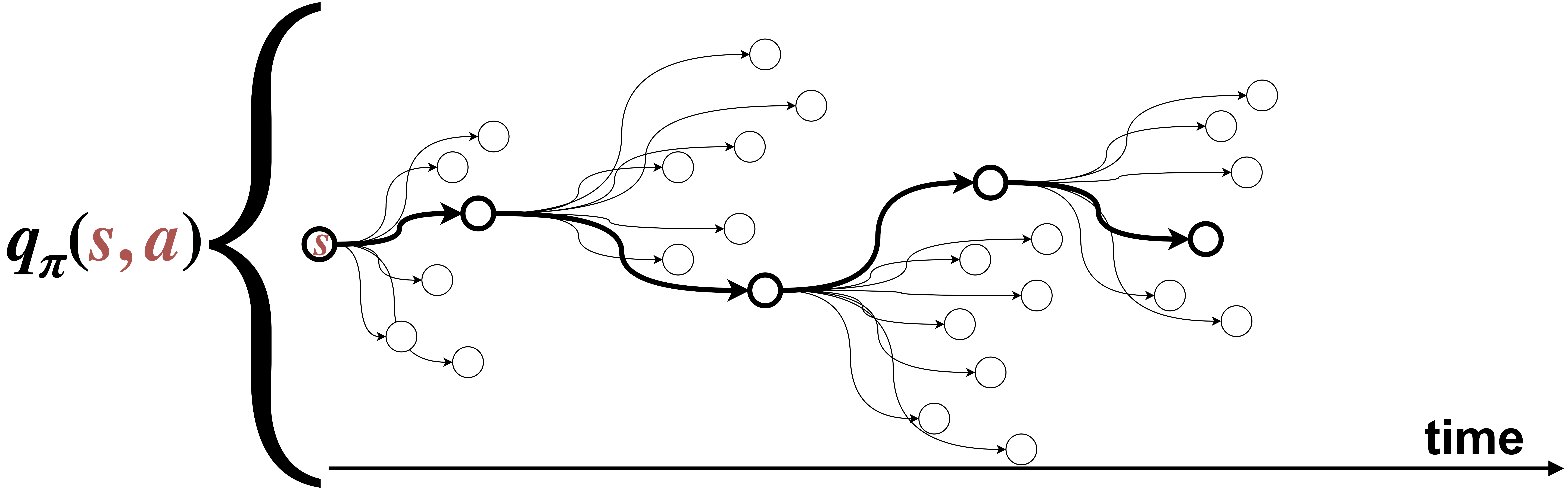
- An action-value function says how good it is to be in a state, take an action, and thereafter follow a policy:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \mid S_t = s, A_t = a\right]$$

Action-value functions

- An action-value function says how good it is to be in a state, take an action, and thereafter follow a policy:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \mid S_t = s, A_t = a\right]$$



Optimal Polices

- A policy $\pi_\star$ is optimal if it maximizes the action-value function:

$$q_{\pi_\star}(s,a) \doteq \max_\pi q_\pi(s,a) = q_\star(s,a) \qquad \forall\, s \in \mathcal{S},\, a \in \mathcal{A}(s)$$

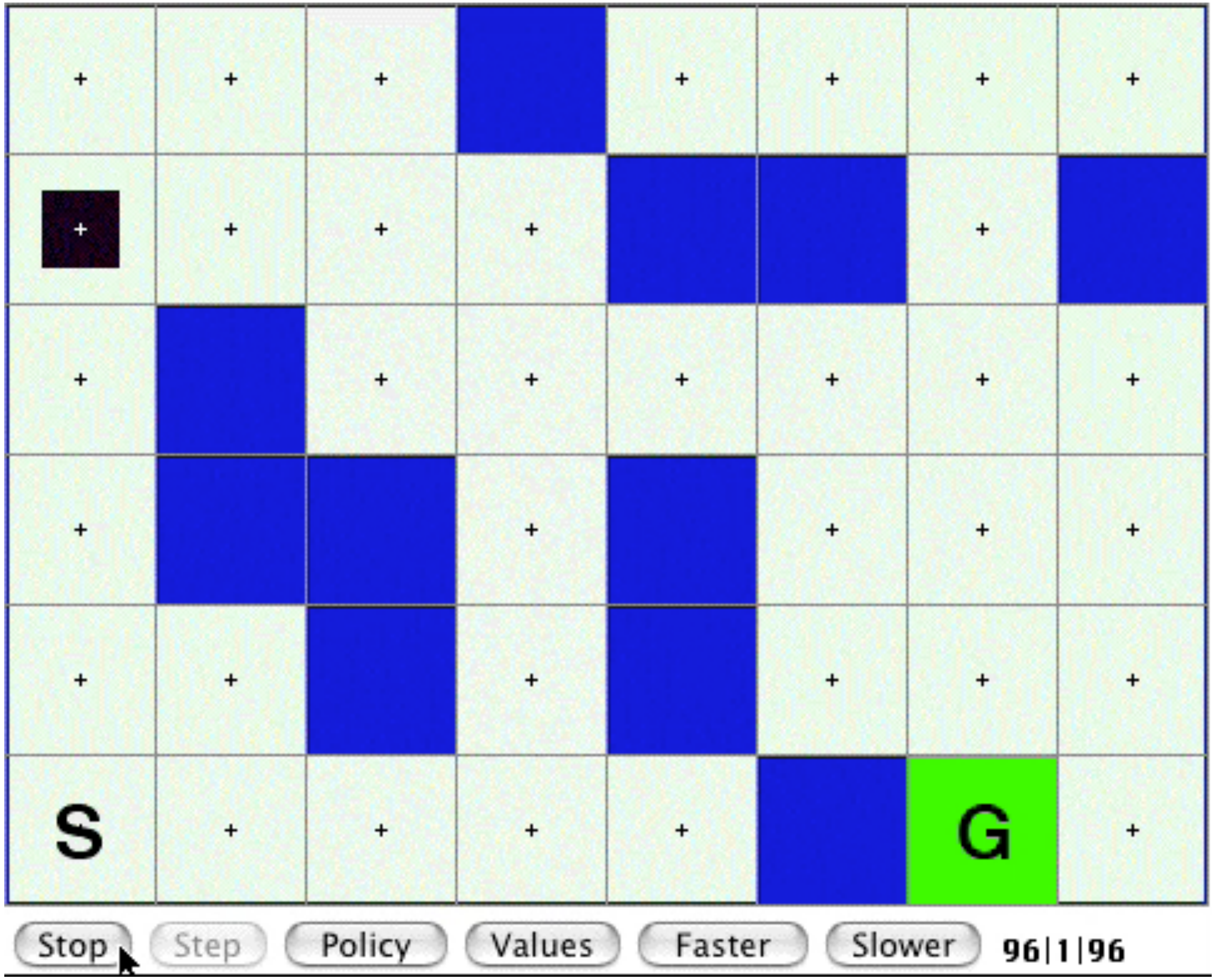
- Thus all optimal policies share the same optimal value function
- Given the optimal value function, it is easy to act optimally:

$$\pi_\star(s) = \arg\max_a q_\star(s,a) \qquad \text{“greedification”}$$

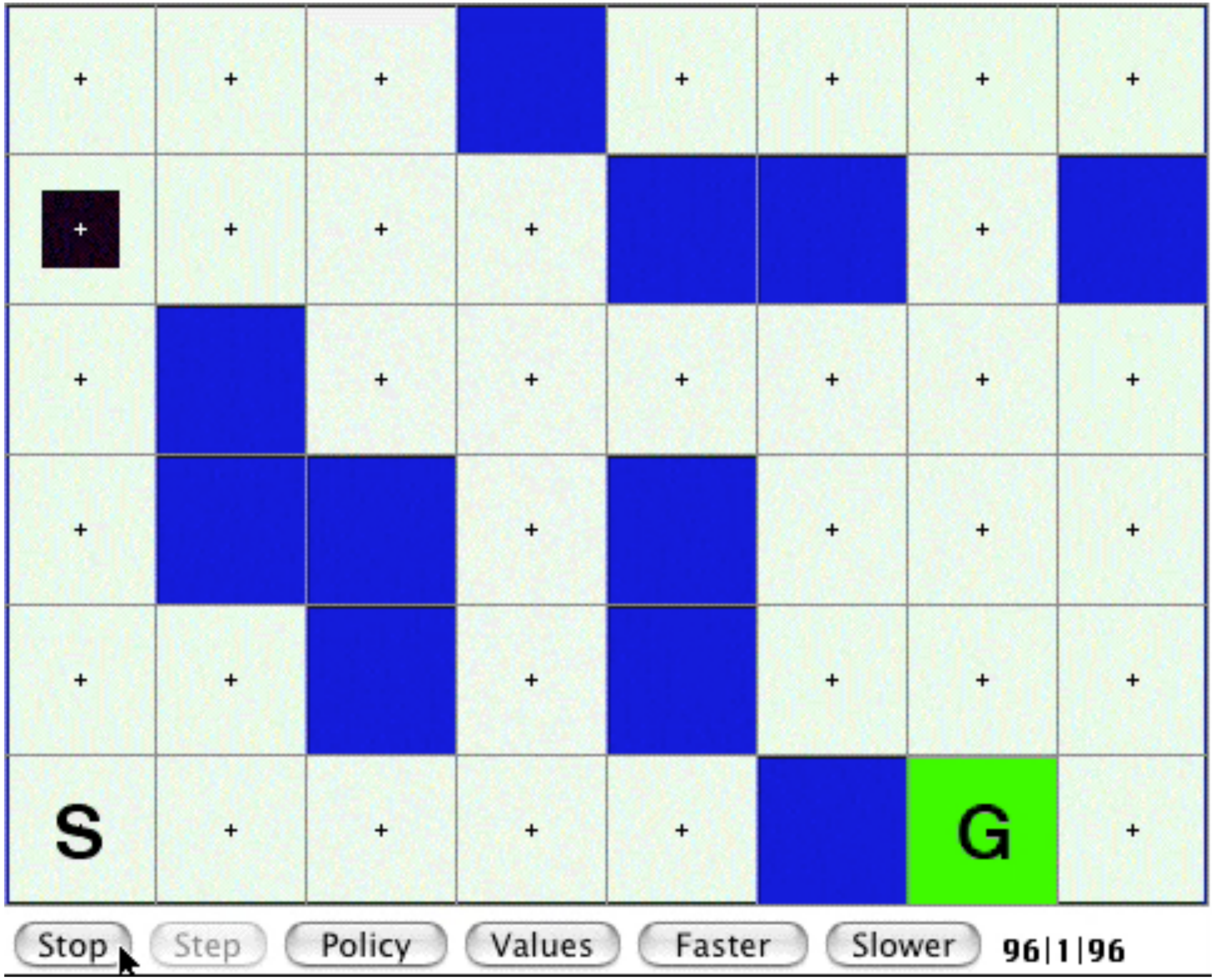
  - we say that the optimal policy is greedy with respect to the optimal value function

- There is always at least one deterministic optimal policy

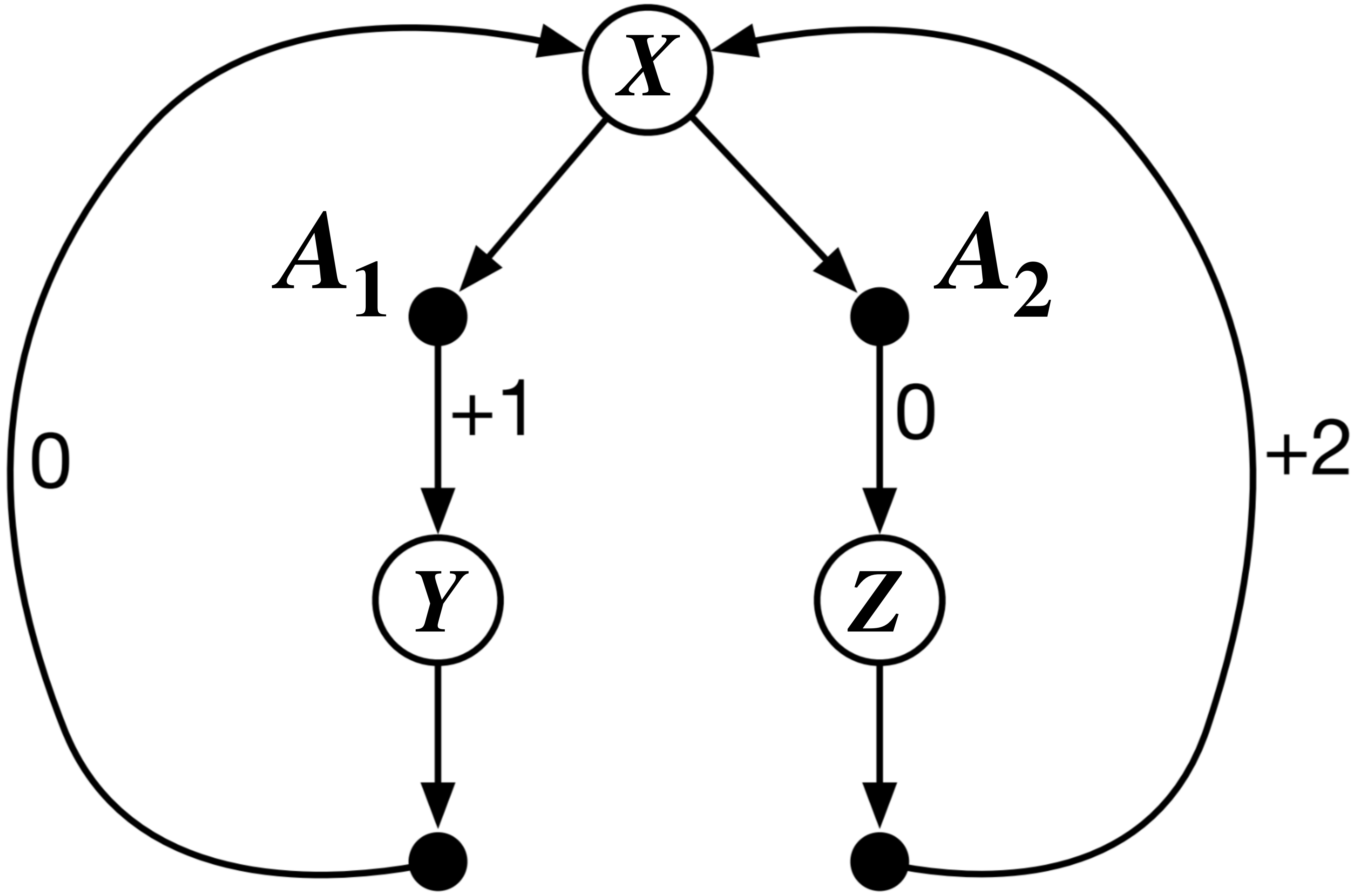
GridWorld Example



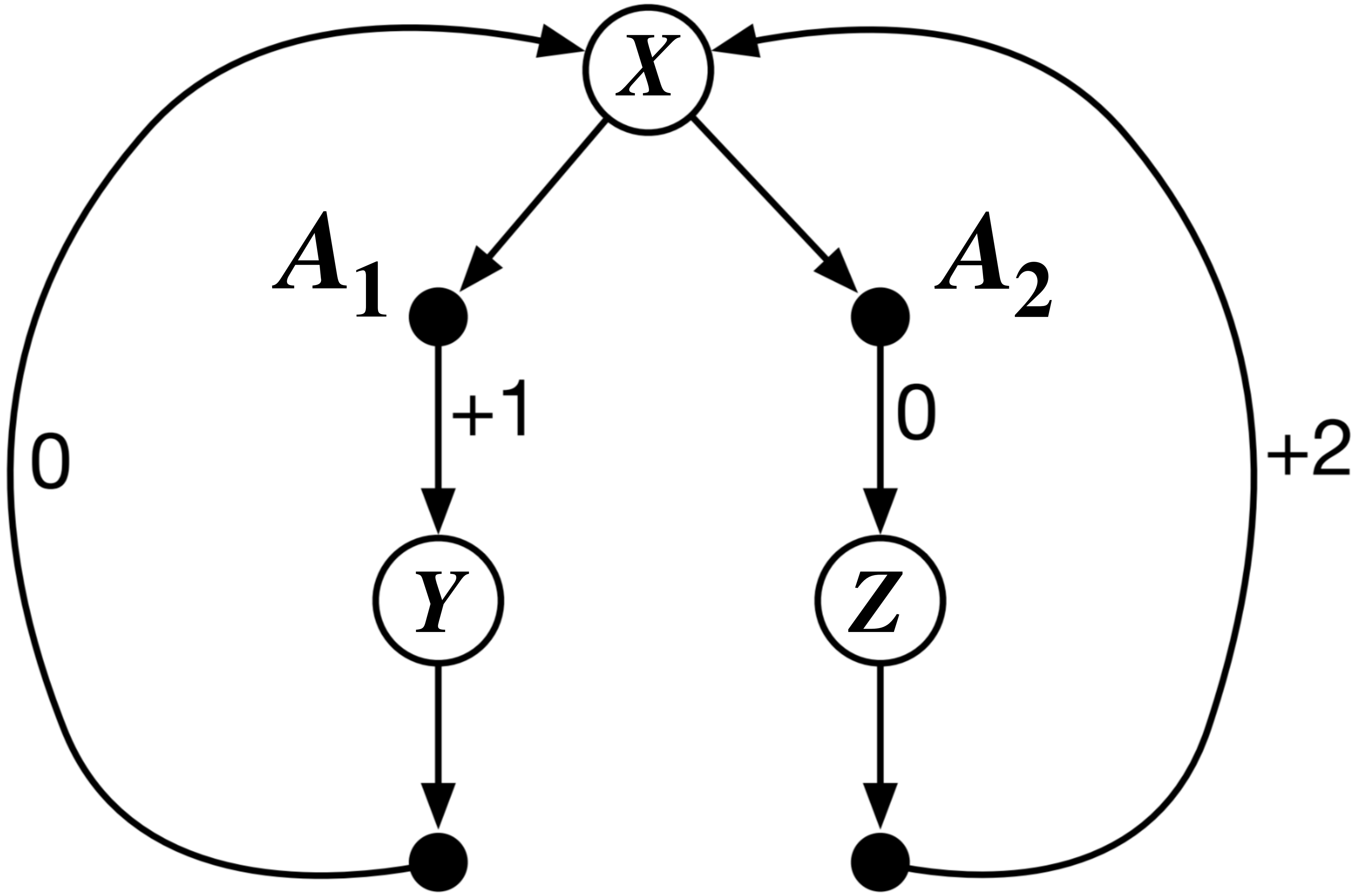
GridWorld Example



Exercise: what's optimal?

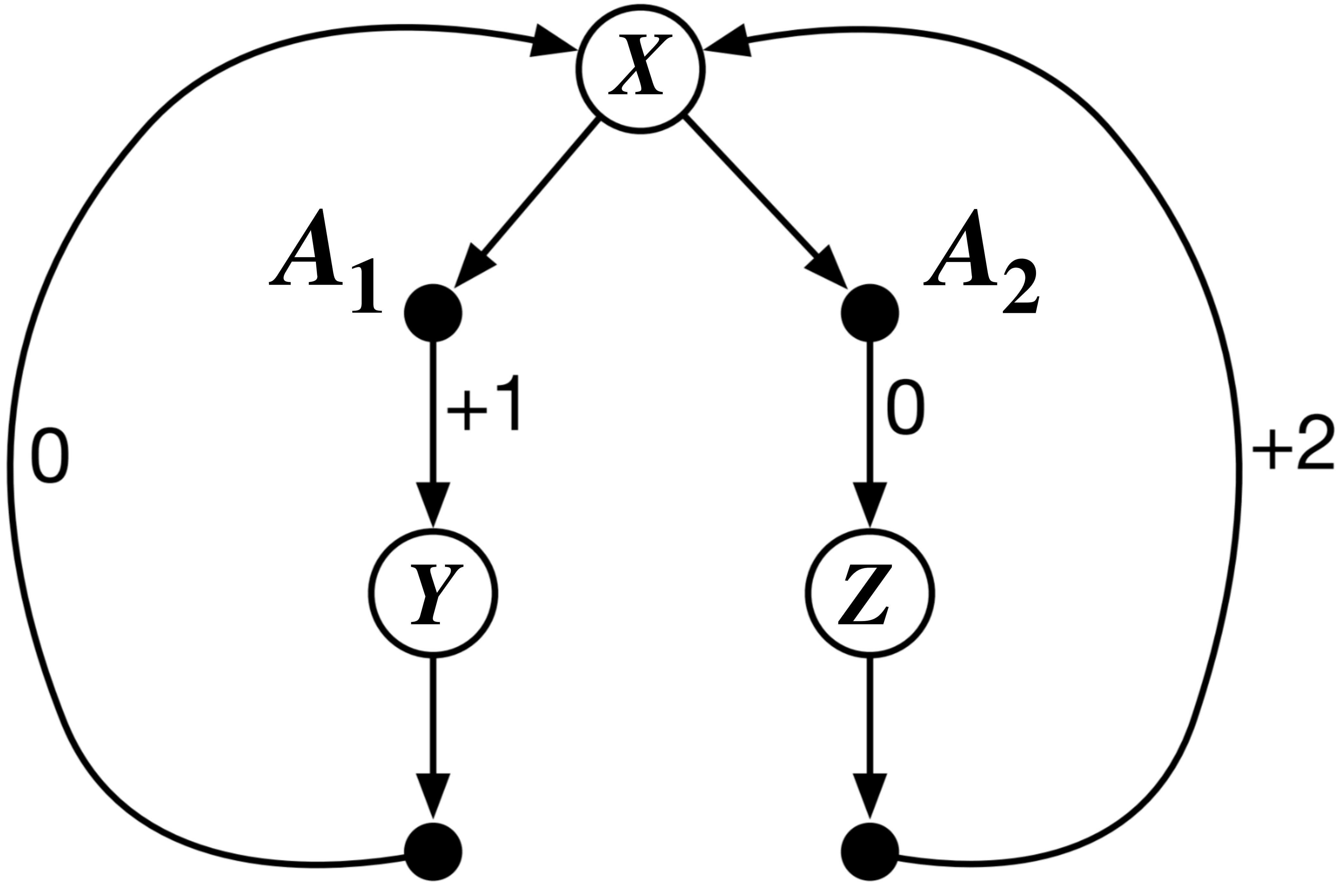


Exercise: what's optimal?



$\pi_1(X) = A_1$ $\pi_2(X) = A_2$

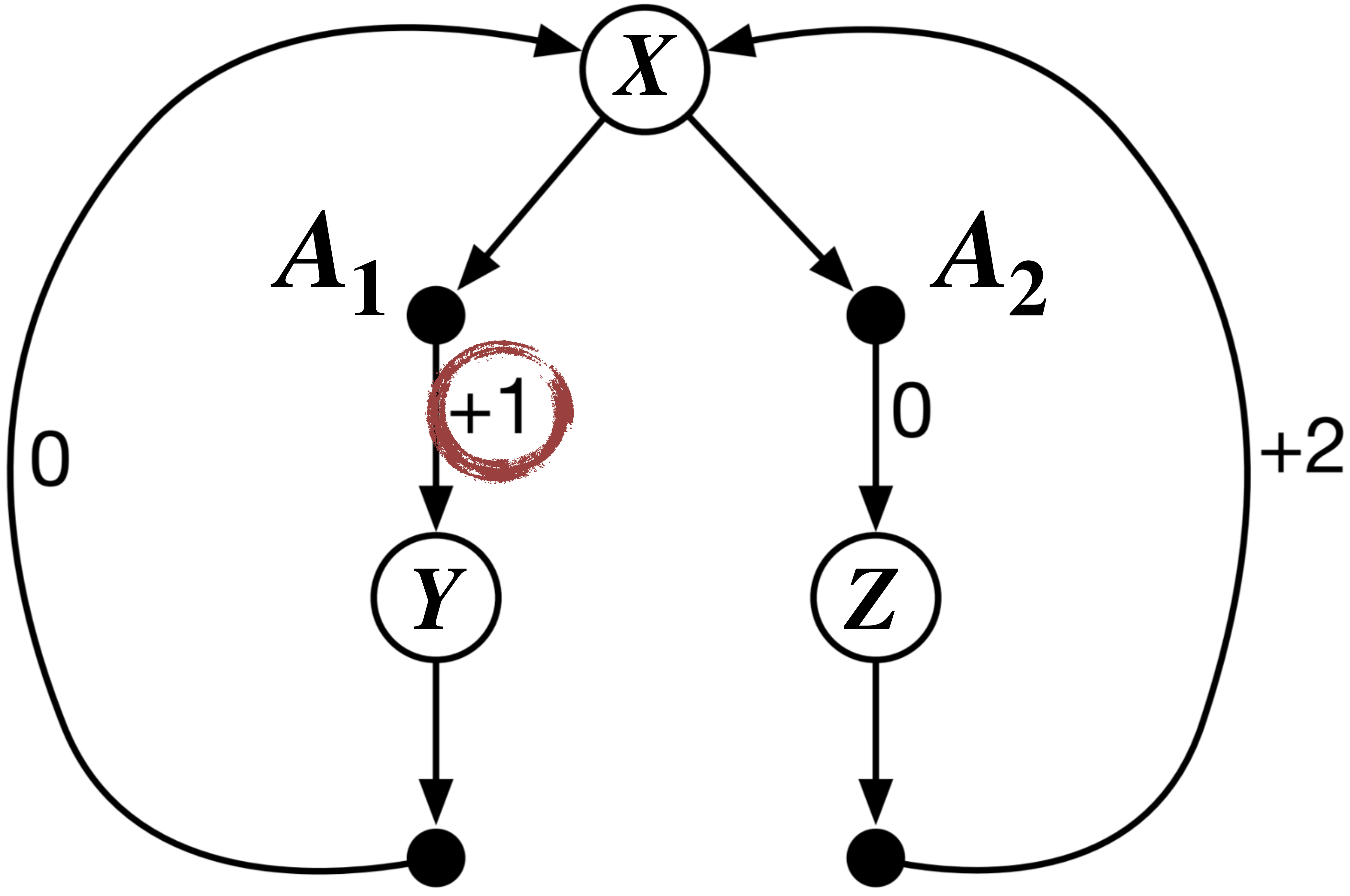
Exercise: what's optimal?



$\gamma = 0$

$\pi_1(X) = A_1$ $\pi_2(X) = A_2$

Exercise: what's optimal?

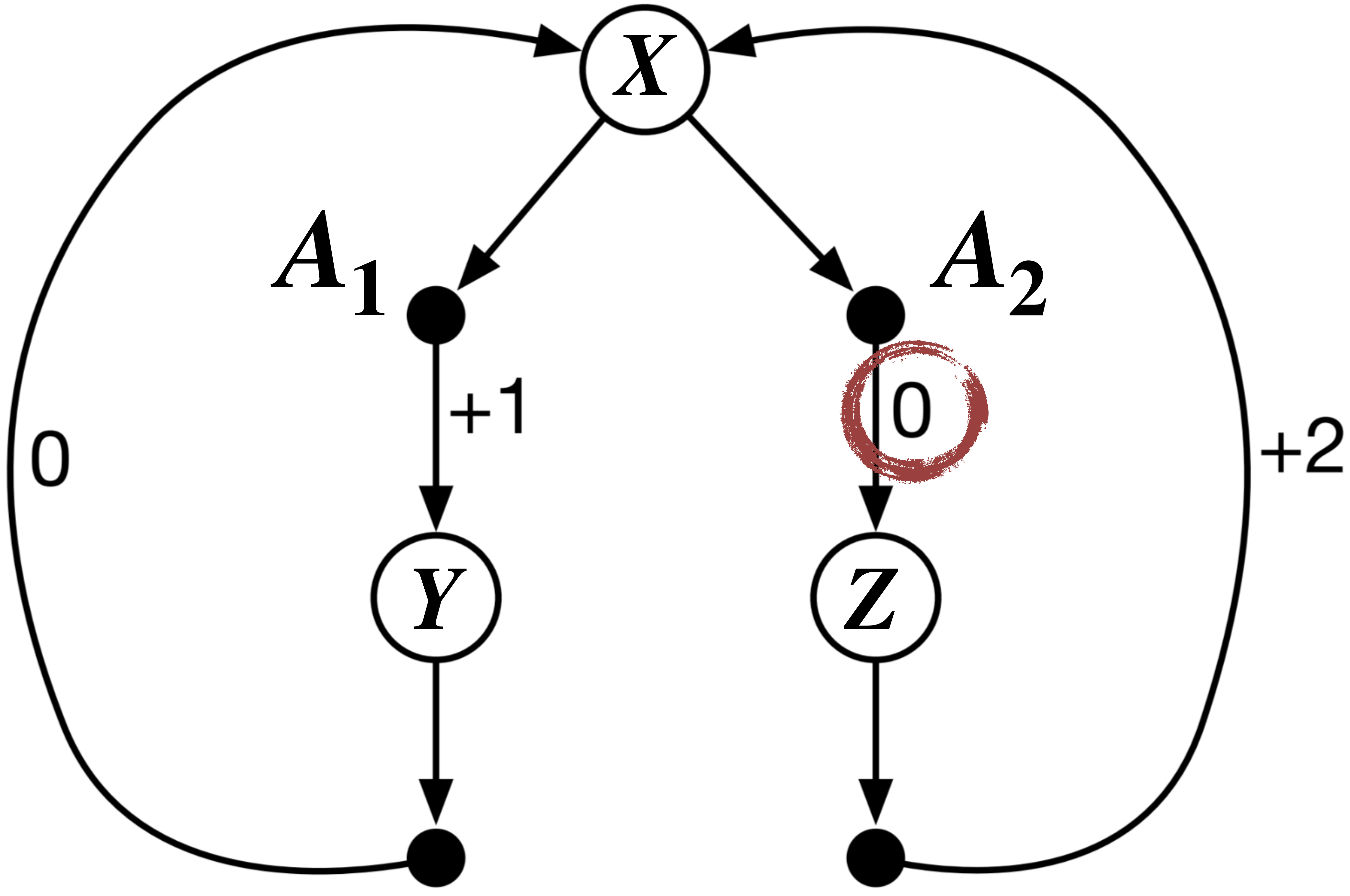


$\gamma = 0$

$v_{\pi_1}(X) = 1$

$\pi_1(X) = A_1$ $\quad$ $\pi_2(X) = A_2$

Exercise: what's optimal?



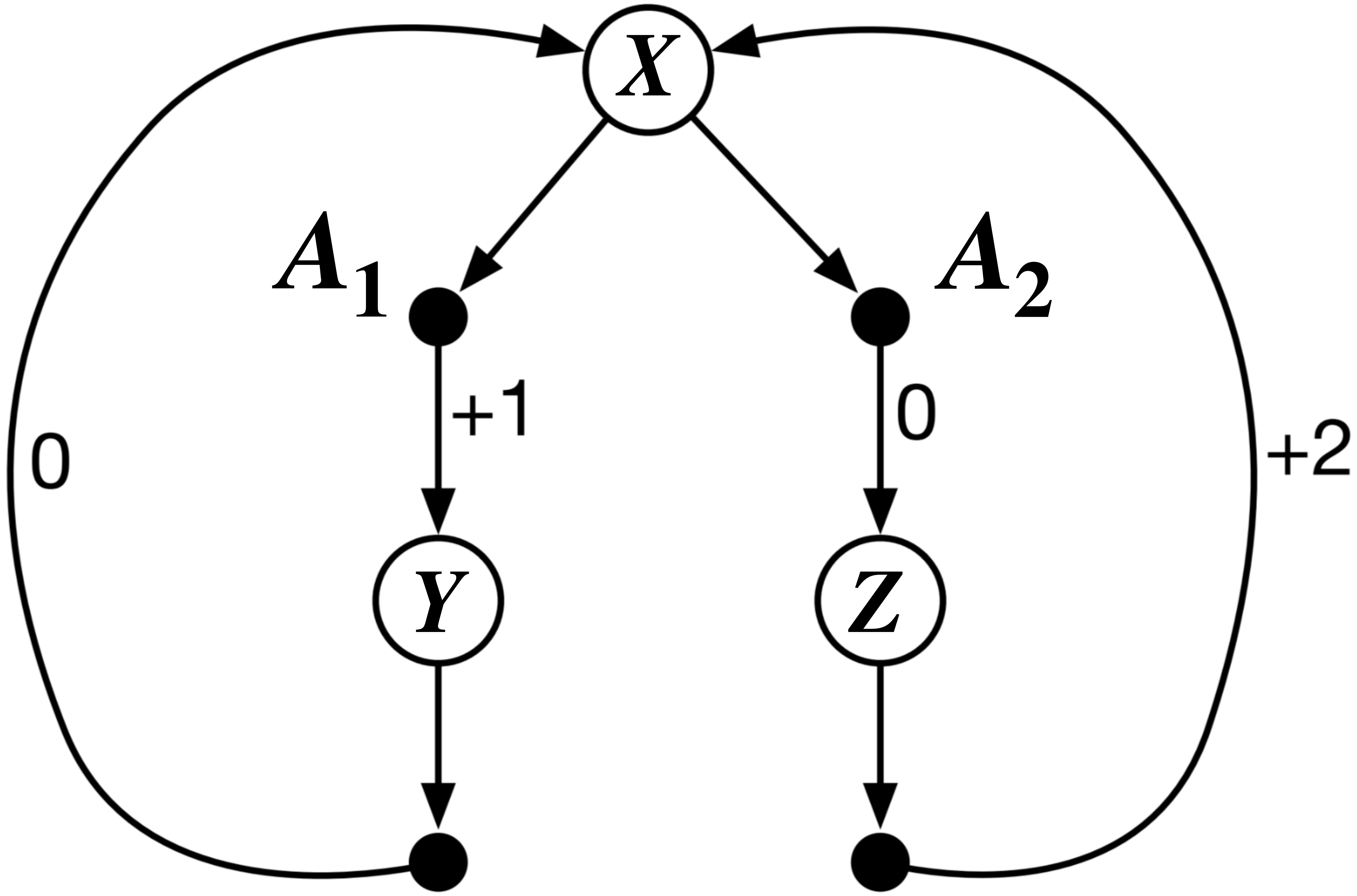
$\pi_1(X) = A_1$ $\quad$ $\pi_2(X) = A_2$

$\gamma = 0$

$v_{\pi_1}(X) = 1$

$v_{\pi_2}(X) = 0$

Exercise: what's optimal?



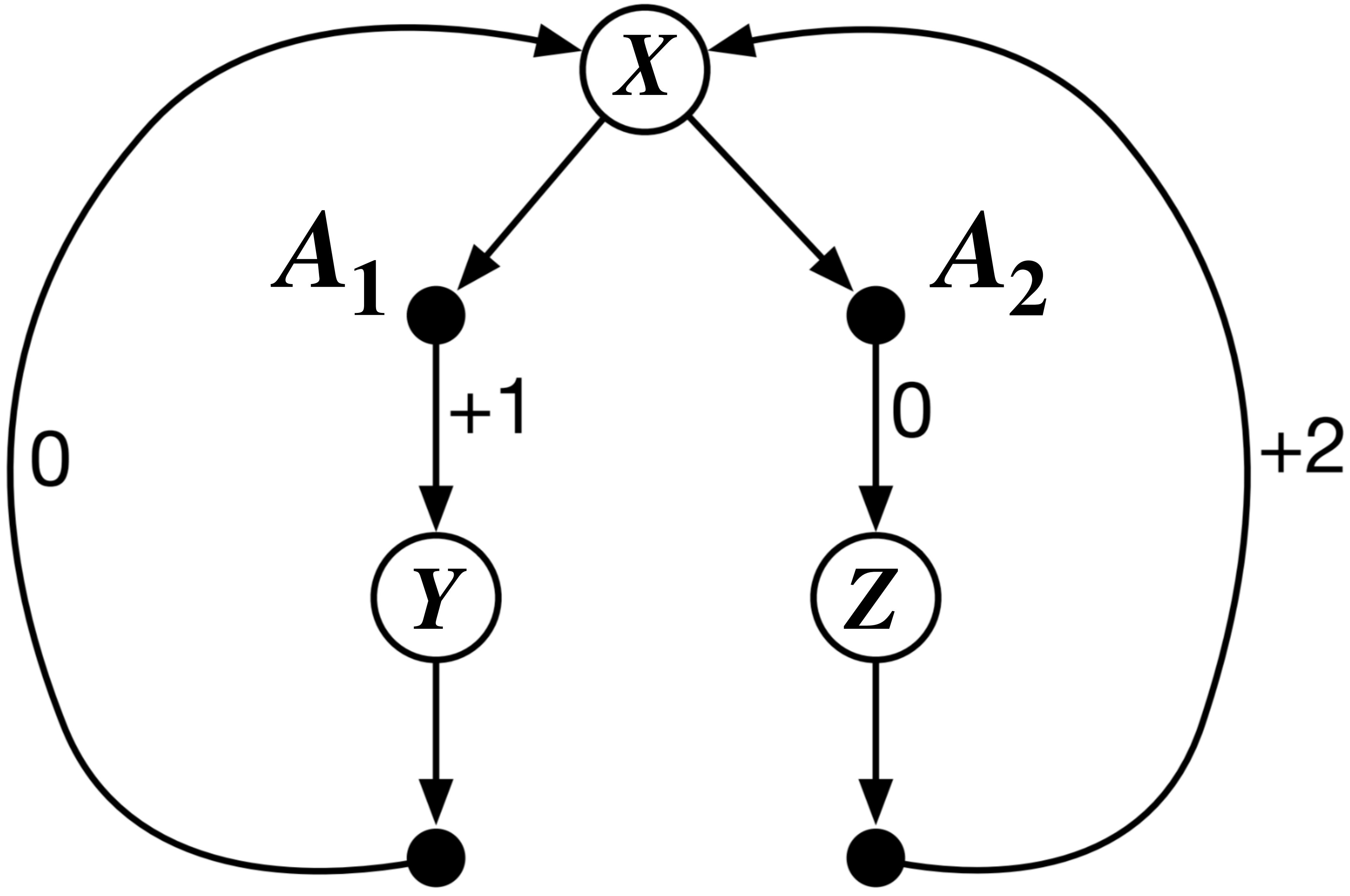
$\pi_1(X) = A_1$ $\quad \pi_2(X) = A_2$

$\gamma = 0$

$v_{\pi_1}(X) = 1$ ✓

$v_{\pi_2}(X) = 0$

Exercise: what's optimal?



$\pi_1(X) = A_1$ $\pi_2(X) = A_2$

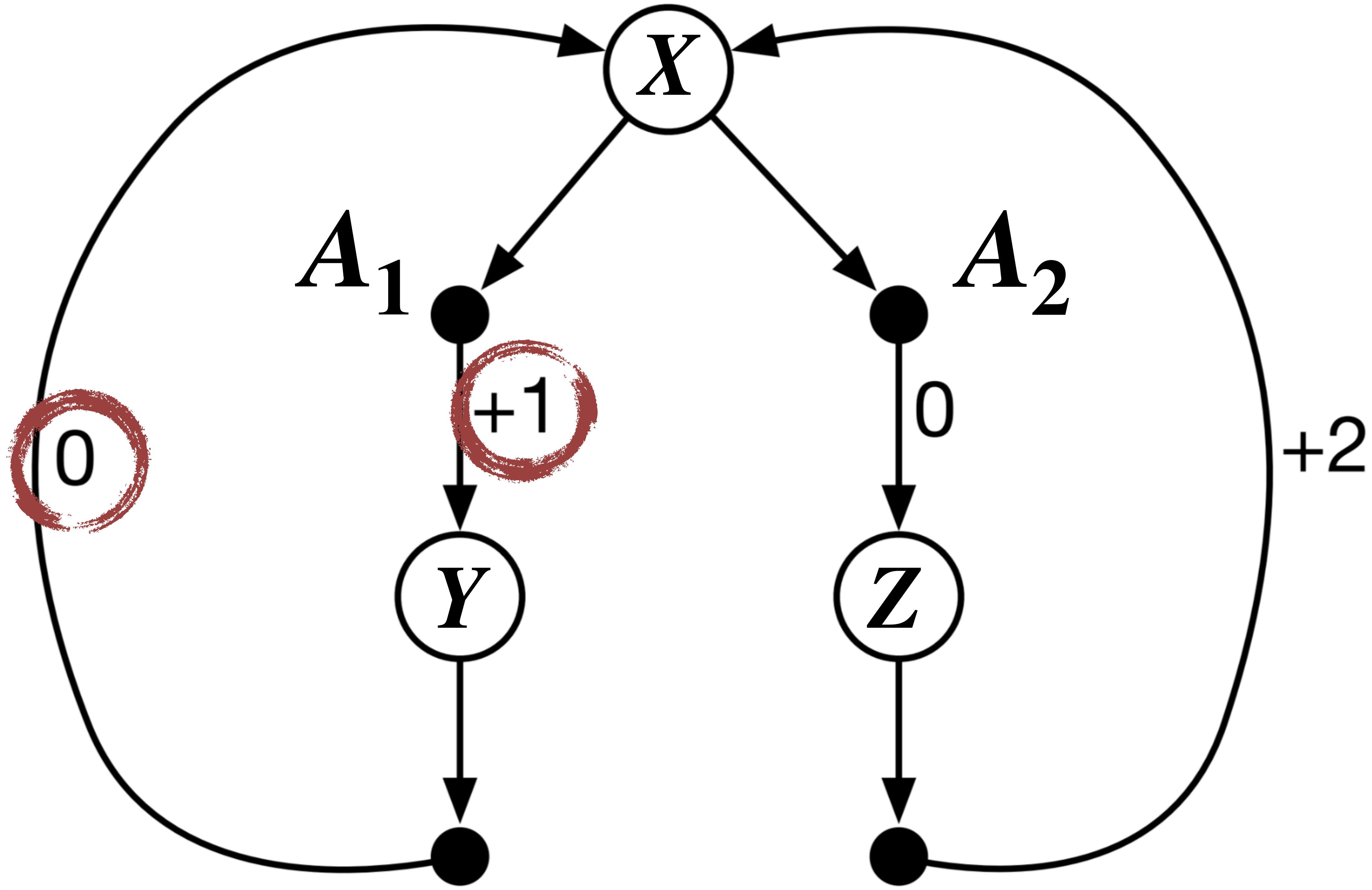
$\gamma = 0$

$v_{\pi_1}(X) = 1$ ✓

$v_{\pi_2}(X) = 0$

$\gamma = 0.9$

Exercise: what's optimal?



$\pi_1(X) = A_1 \quad \pi_2(X) = A_2$

$\gamma = 0$

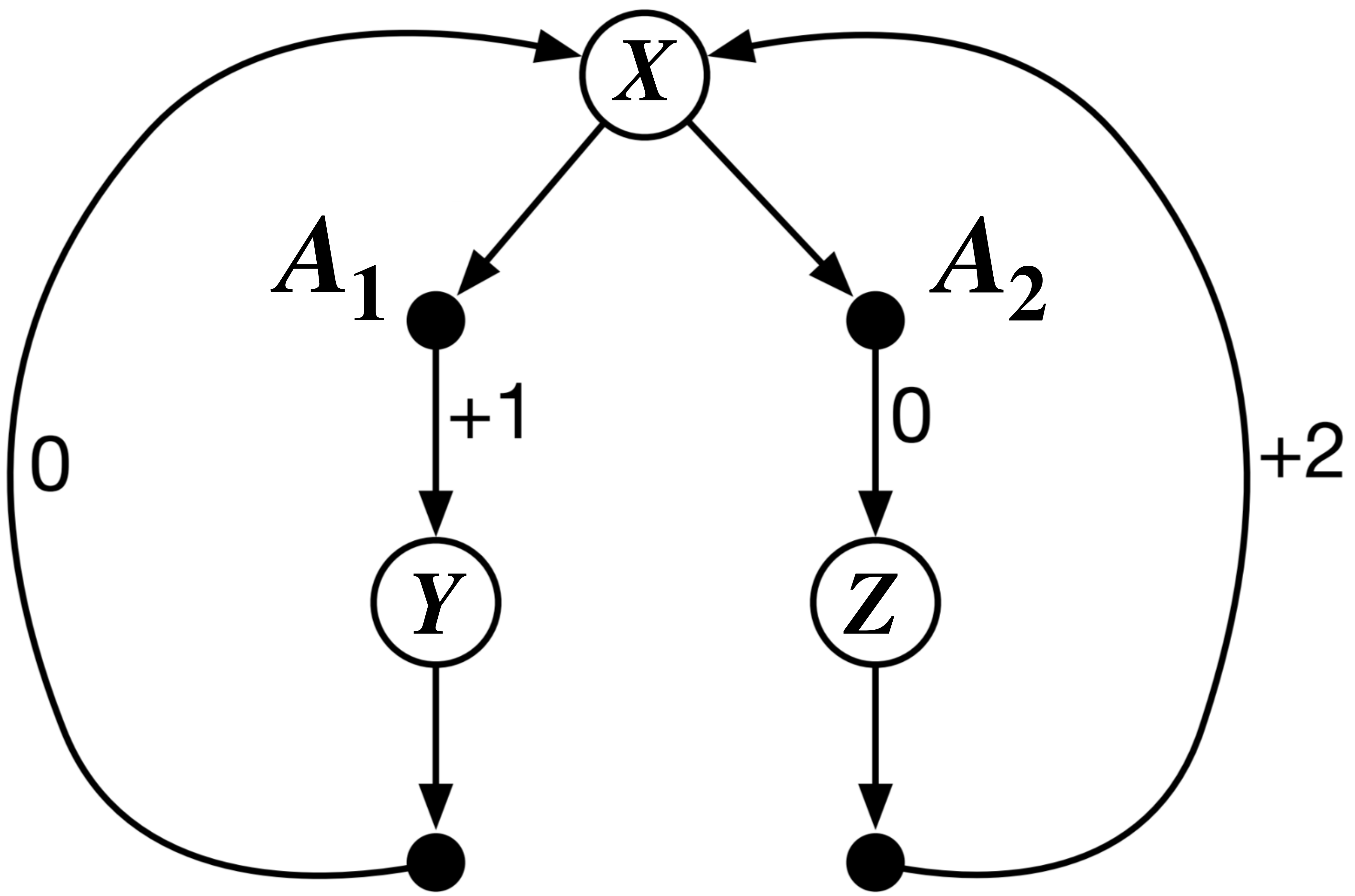
$v_{\pi_1}(X) = 1$ ✓

$v_{\pi_2}(X) = 0$

$\gamma = 0.9$

$v_{\pi_1}(X) = 1 + 0.9 * 0 + (0.9)^2 * 1 + \ldots$

Exercise: what's optimal?



$\pi_1(X) = A_1$ $\pi_2(X) = A_2$

$\gamma = 0$

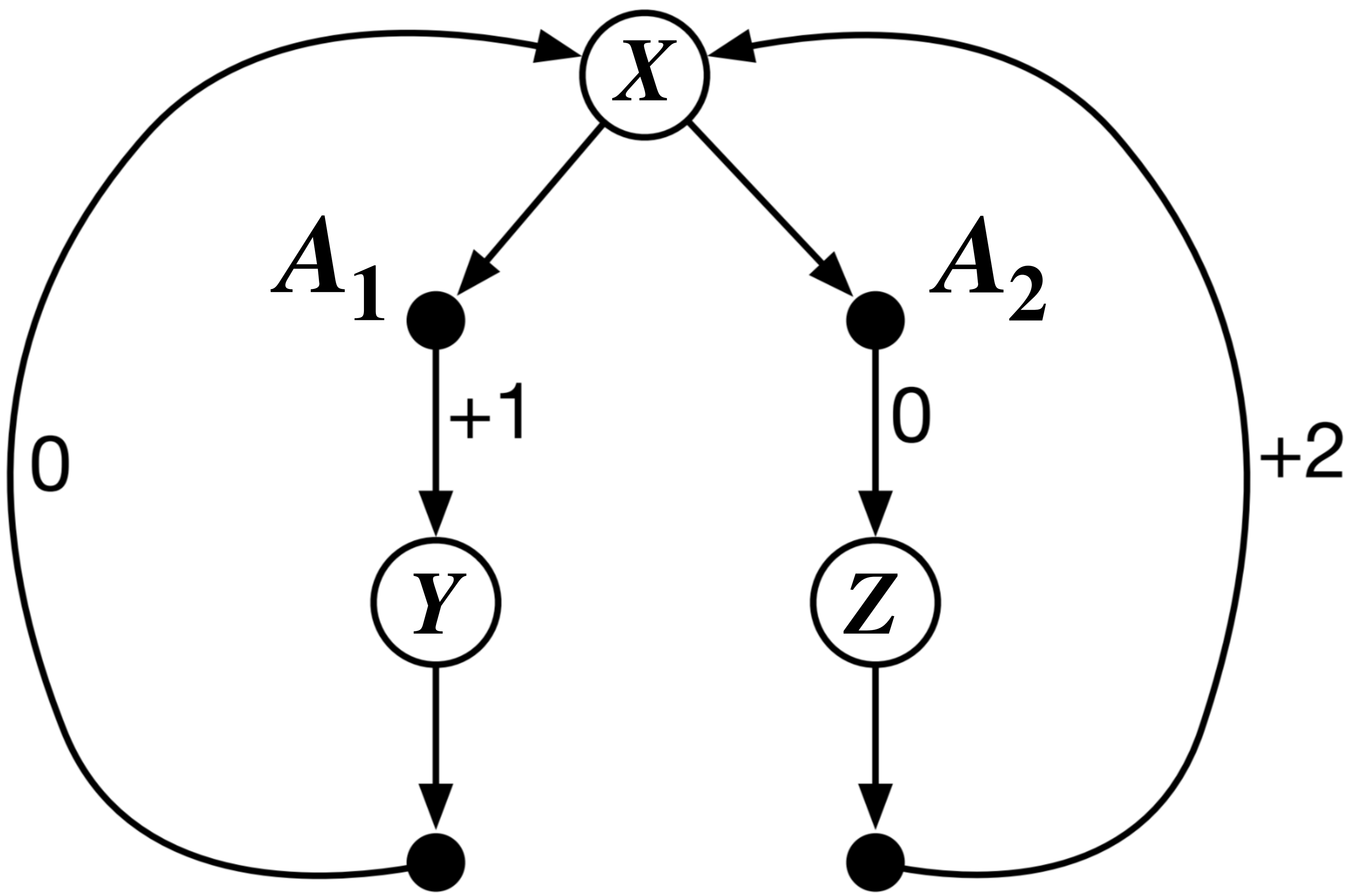
$v_{\pi_1}(X) = 1$ ✓

$v_{\pi_2}(X) = 0$

$\gamma = 0.9$

$$v_{\pi_1}(X) = \sum_{k=0}^{\infty} (0.9)^{2k}$$

Exercise: what's optimal?



$\pi_1(X) = A_1$ $\pi_2(X) = A_2$

$\gamma = 0$

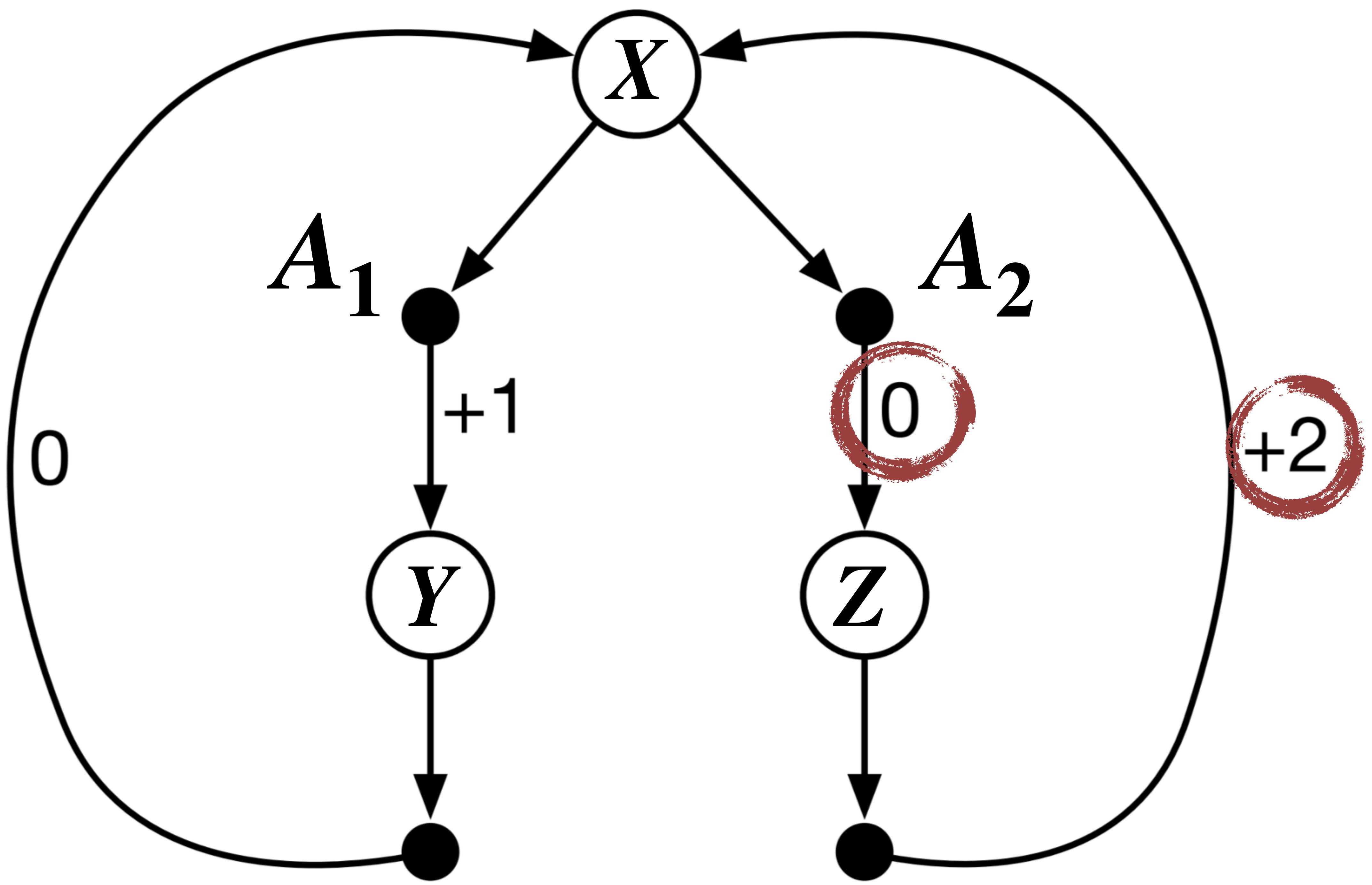
$v_{\pi_1}(X) = 1$ ✓

$v_{\pi_2}(X) = 0$

$\gamma = 0.9$

$$v_{\pi_1}(X) = \sum_{k=0}^{\infty} (0.9)^{2k} = \frac{1}{1 - 0.9^2} \approx 5.3$$

Exercise: what's optimal?



$\pi_1(X) = A_1$ $\pi_2(X) = A_2$

$\gamma = 0$

$v_{\pi_1}(X) = 1$

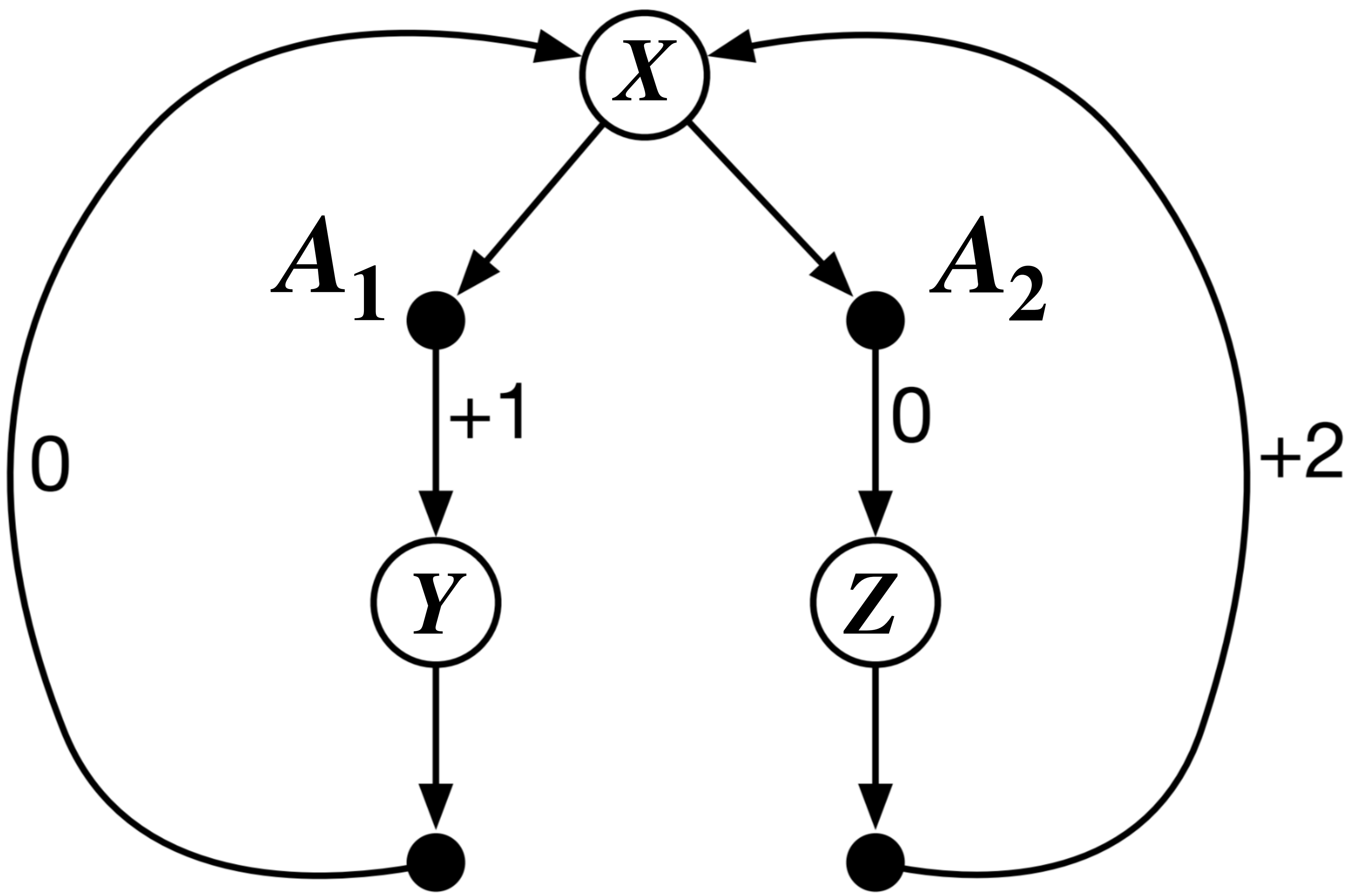
$v_{\pi_2}(X) = 0$

$\gamma = 0.9$

$$v_{\pi_1}(X) = \sum_{k=0}^{\infty} (0.9)^{2k} = \frac{1}{1 - 0.9^2} \approx 5.3$$

$$v_{\pi_2}(X) = 0 + 0.9 * 2 + (0.9)^2 * 0 + \ldots$$

Exercise: what's optimal?



$\pi_1(X) = A_1$ $\quad$ $\pi_2(X) = A_2$

$\gamma = 0$

$v_{\pi_1}(X) = 1$ ✓

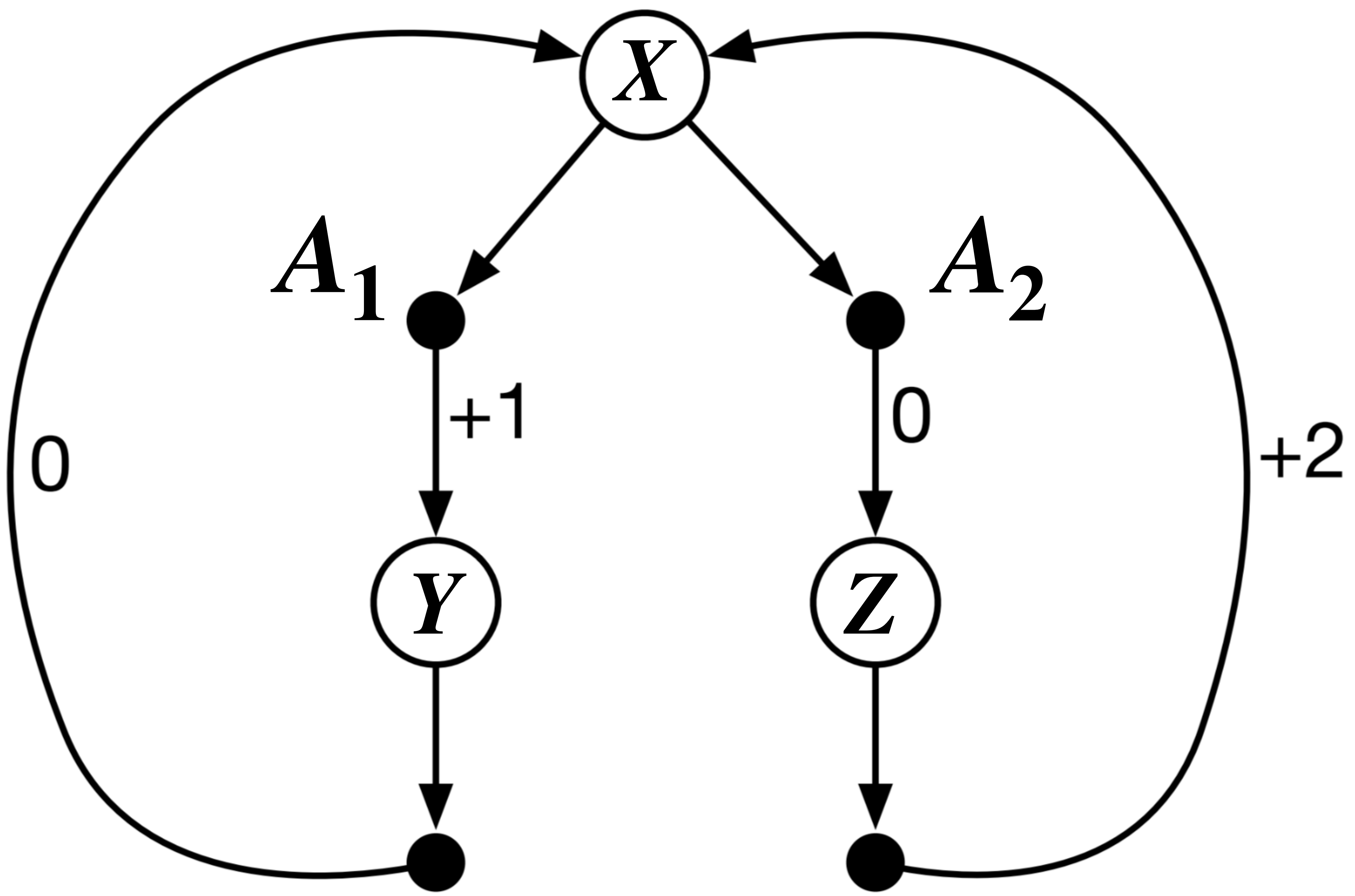
$v_{\pi_2}(X) = 0$

$\gamma = 0.9$

$$v_{\pi_1}(X) = \sum_{k=0}^{\infty} (0.9)^{2k} = \frac{1}{1 - 0.9^2} \approx 5.3$$

$$v_{\pi_2}(X) = \sum_{k=0}^{\infty} (0.9)^{2k+1} * 2$$

Exercise: what's optimal?



$\pi_1(X) = A_1$ $\pi_2(X) = A_2$

$\gamma = 0$

$v_{\pi_1}(X) = 1$ ✓

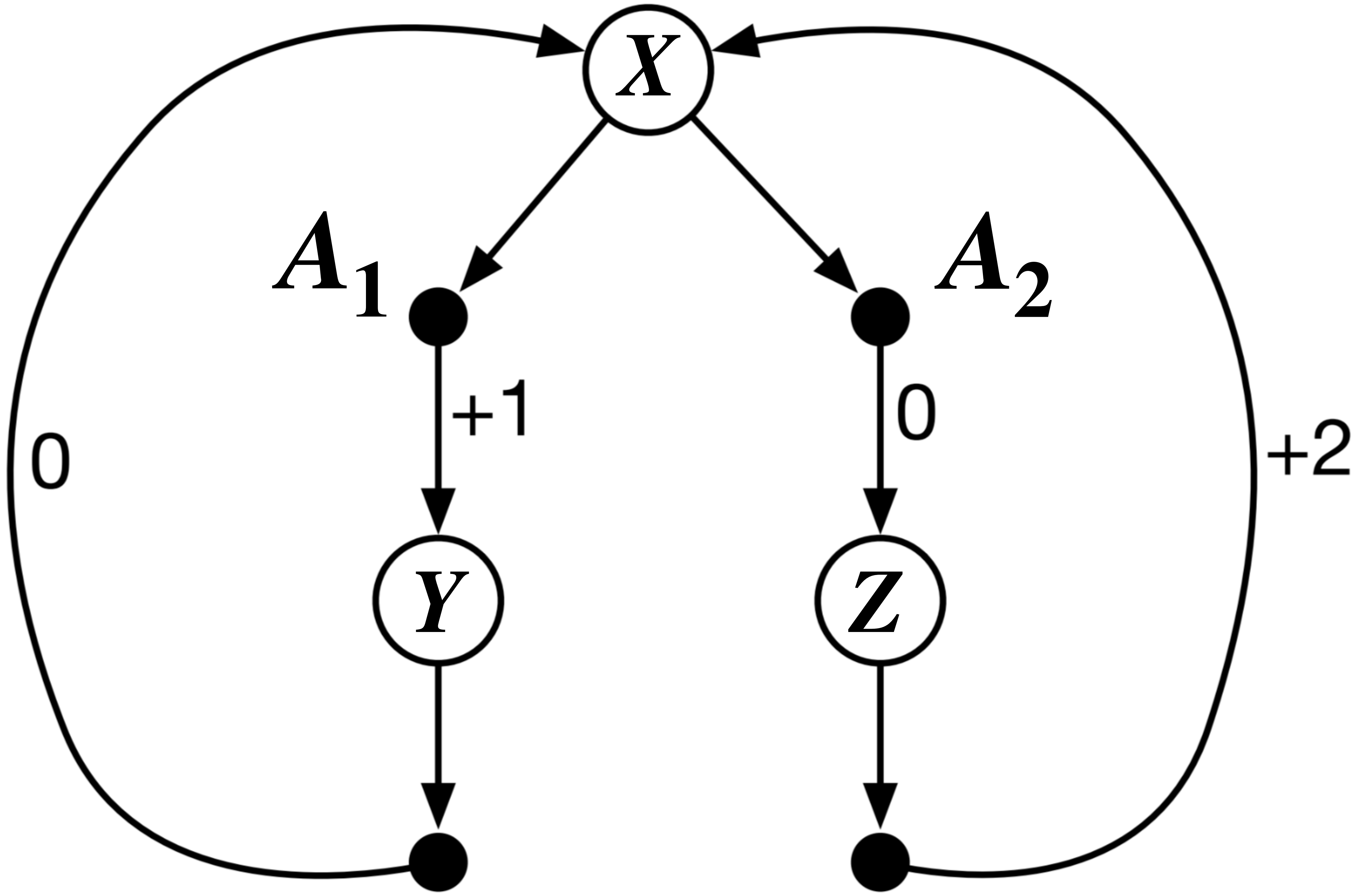
$v_{\pi_2}(X) = 0$

$\gamma = 0.9$

$$v_{\pi_1}(X) = \sum_{k=0}^{\infty} (0.9)^{2k} = \frac{1}{1 - 0.9^2} \approx 5.3$$

$$v_{\pi_2}(X) = \sum_{k=0}^{\infty} (0.9)^{2k+1} * 2 = \frac{0.9}{1 - 0.9^2} * 2 \approx 9.5$$

Exercise: what's optimal?



$\pi_1(X) = A_1 \quad \pi_2(X) = A_2$

$\gamma = 0$

$v_{\pi_1}(X) = 1$ ✓

$v_{\pi_2}(X) = 0$

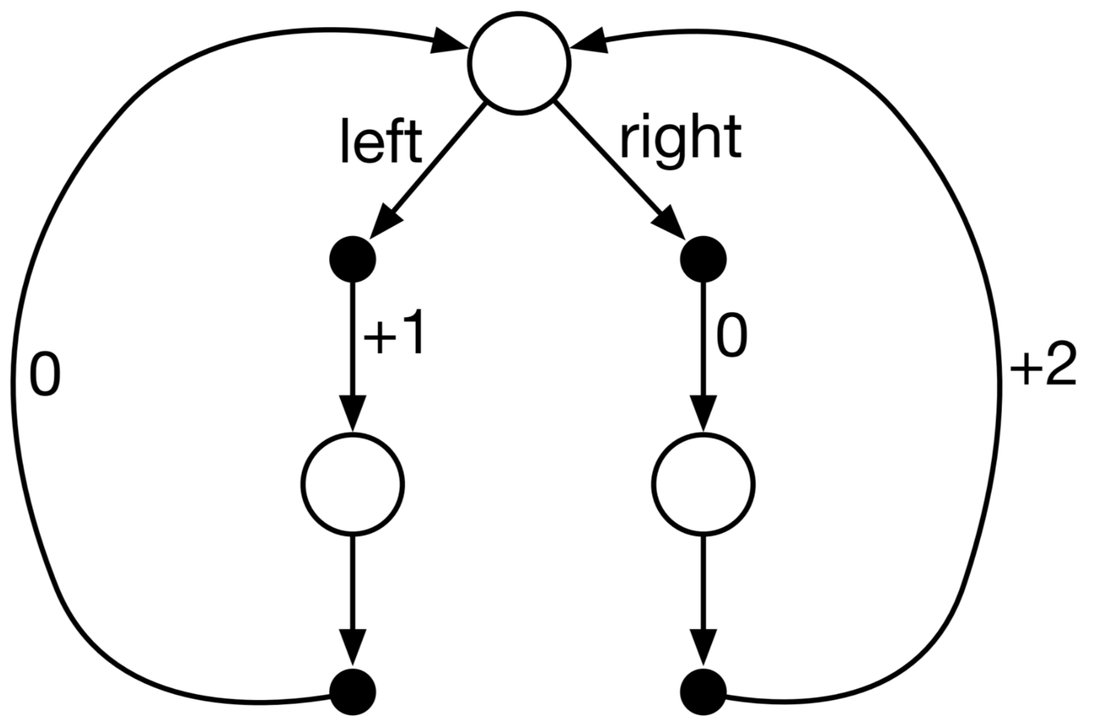
$\gamma = 0.9$

$$v_{\pi_1}(X) = \sum_{k=0}^{\infty} (0.9)^{2k} = \frac{1}{1 - 0.9^2} \approx 5.3$$

$$v_{\pi_2}(X) = \sum_{k=0}^{\infty} (0.9)^{2k+1} * 2 = \frac{0.9}{1 - 0.9^2} * 2 \approx 9.5$$ ✓

We can only directly solve small MDPs

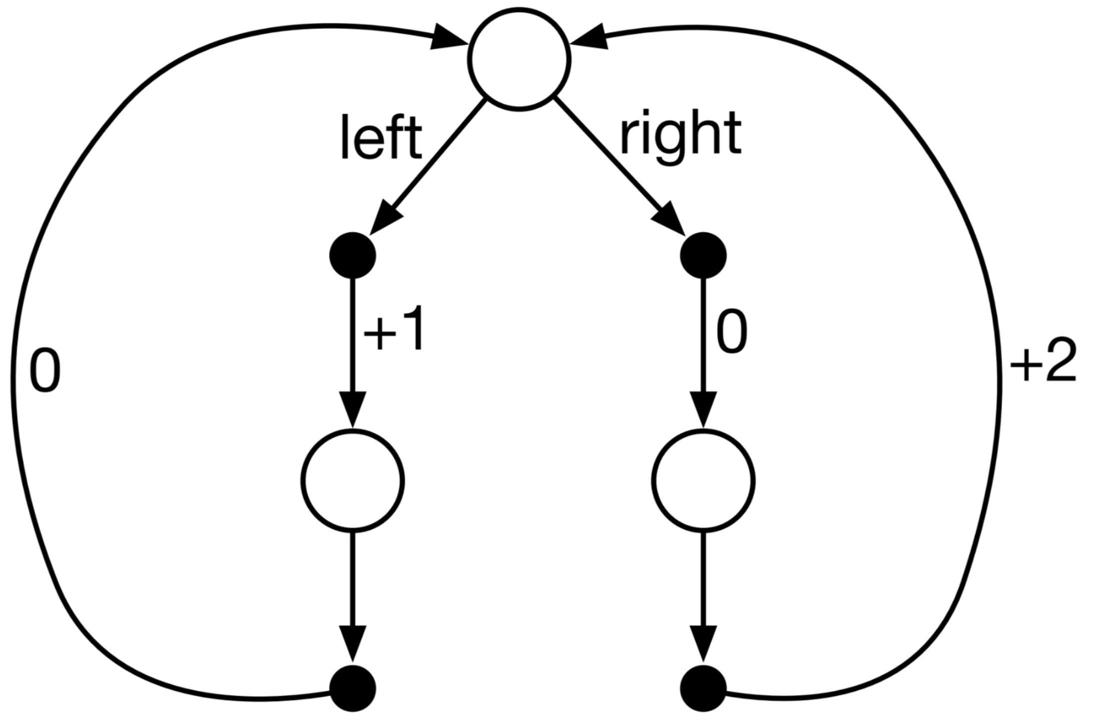
We can only directly solve small MDPs



→ 2 Deterministic Policies → Brute-Force Search

A General MDP → $|\mathcal{A}|^{|\mathcal{S}|}$ Deterministic Policies → Brute-Force Search

We can only directly solve small MDPs



→ 2 Deterministic Policies → Brute-Force Search

A General MDP → $|\mathscr{A}|^{|\mathscr{S}|}$ Deterministic Policies → ~~Brute-Force Search~~

$|\mathscr{A}|^{|\mathscr{S}|}$ Deterministic Policies → RL methods based on Bellman Equations

Key characteristics of RL

- ☑ Evaluative feedback (reward)
- ☑ Delayed consequences
- ☑ **Must associate situations with actions**
- Online and Incremental learning
- Need for trial and error, to explore as well as exploit
- Non-stationarity

Key characteristics of RL

- ☑ Evaluative feedback (reward)
- ☑ Delayed consequences
- ☑ **Must associate situations with actions**
- Online and Incremental learning

Characteristics of the problem

- Need for trial and error, to explore as well as exploit
- Non-stationarity

Key characteristics of RL

- ☑ Evaluative feedback (reward)
- ☑ Delayed consequences
- ☑ **Must associate situations with actions**
- Online and Incremental learning
- Need for trial and error, to explore as well as exploit
- Non-stationarity

Characteristics of the problem

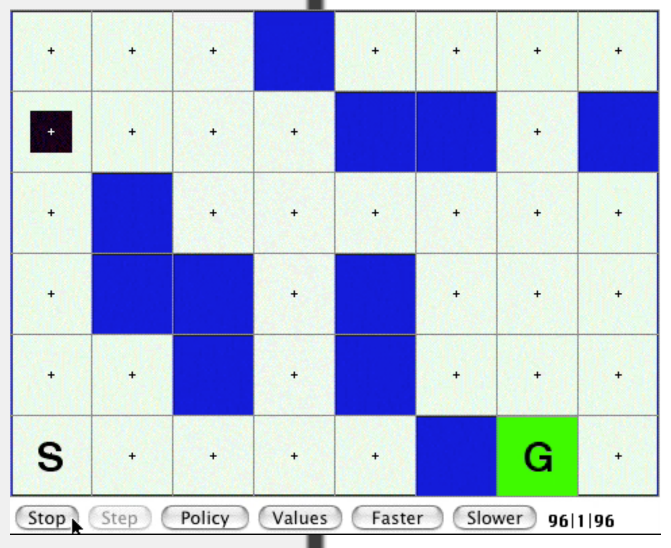
Characteristics of solution/alg

Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

Initialize S
Loop for each step

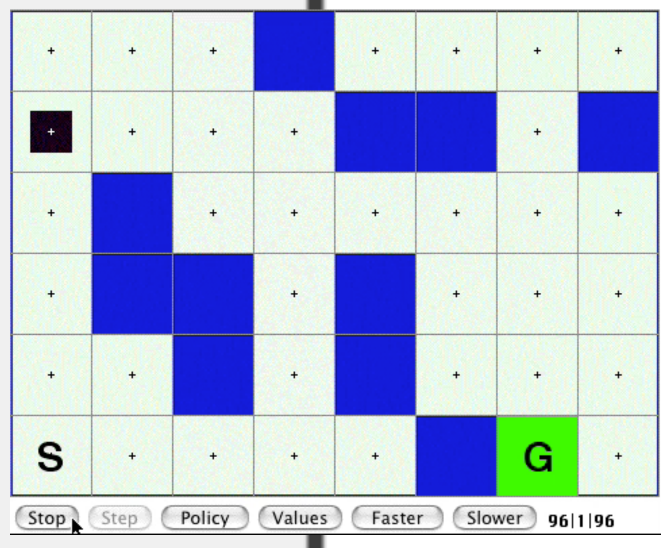


Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

Initialize S
Loop for each step

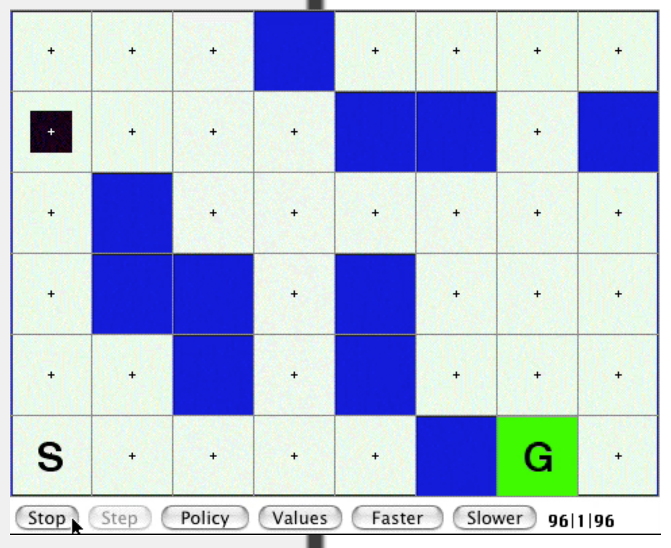


Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

  Initialize S
  Loop for each step
    Choose A from S using policy derived from Q (e.g., ε-greedy)

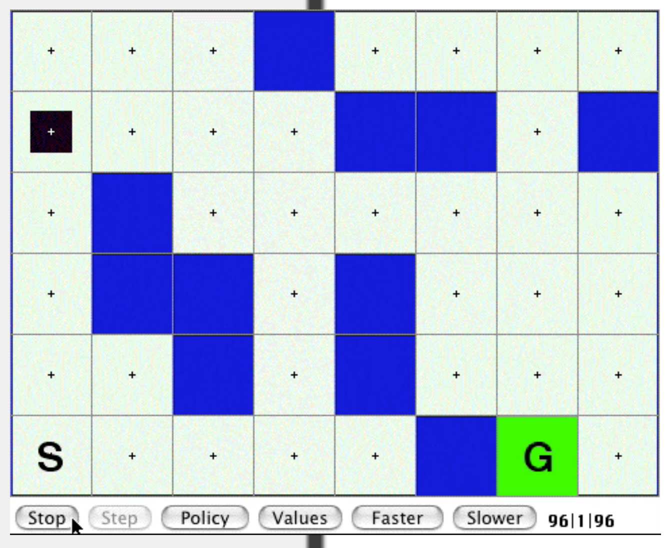


Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

Initialize S
Loop for each step
Choose A from S using policy derived from Q (e.g., ε-greedy)
Take action A, observe R, S'



Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

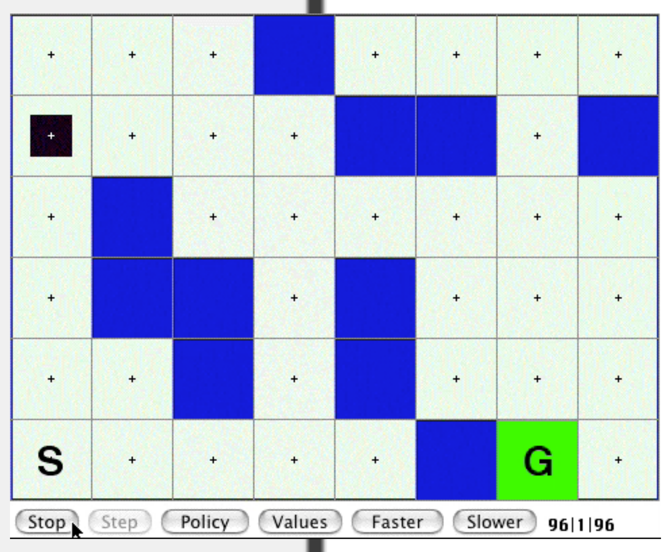
Initialize S

Loop for each step

Choose A from S using policy derived from Q (e.g., ε-greedy)

Take action A, observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$

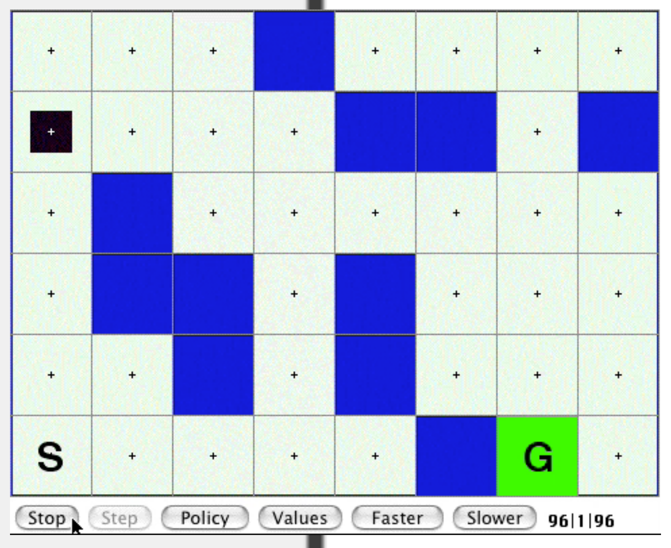


Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

Initialize S
Loop for each step
Choose A from S using policy derived from Q (e.g., ε-greedy)
Take action A, observe R, S'
$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
$S \leftarrow S'$



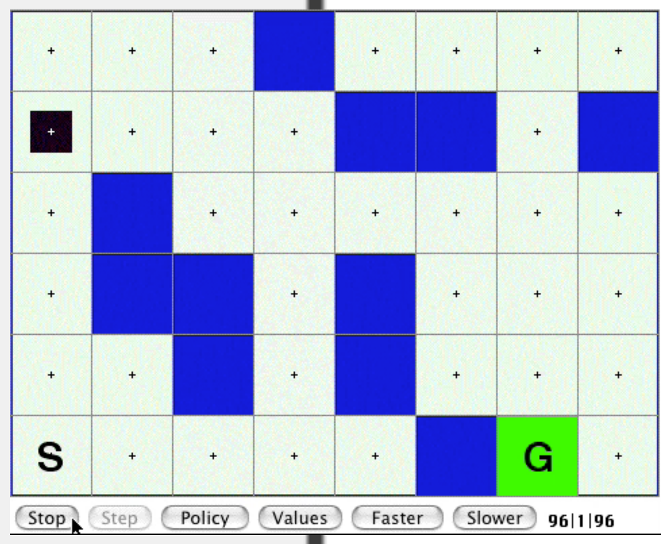
Q-learning

Q converges to $q_\star$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

Initialize S
Loop for each step
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Take action A, observe R, S'
  $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
  $S \leftarrow S'$



- Q-learning converges (Watkins & Dayan 1992) — learning long-term optimal behavior without any **model** of the environment, for arbitrary MDPs!

Key characteristics of RL

- ☑ Evaluative feedback (reward)
- ☑ Delayed consequences
- ☑ Must associate different actions with different situations
- Online and Incremental learning
- Need for trial and error, to explore as well as exploit
- Non-stationarity

Characteristics of solution/alg

Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

Initialize S
Loop for each step of episode:
Choose A from S using policy derived from Q (e.g., ε-greedy)
Take action A, observe R, S'
$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
$S \leftarrow S'$
until S is terminal

Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

Initialize S
Loop for each step of episode:
Choose A from S using policy derived from Q (e.g., ε-greedy)
Take action A, observe R, S'
$Q(S, A) \leftarrow Q(S, A) + \alpha \underbrace{\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]}_{\text{error term}}$
$S \leftarrow S'$
until S is terminal

Q-learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

Initialize S
Loop for each step of episode:
Choose A from S using policy derived from Q (e.g., ε-greedy)
Take action A, observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha \big[\underbrace{\overbrace{R + \gamma \max_a Q(S', a)}^{\text{target}} - Q(S, A)}_{\text{error term}}\big]$$

$S \leftarrow S'$
until S is terminal

Bootstrapping: key idea in Q-learning and all temporal-difference (TD) learning

- You might think we need a complete trajectory of rewards to estimate values

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \mid S_t = s, A_t = a \right]$$

Bootstrapping: key idea in Q-learning and all temporal-difference (TD) learning

- You might think we need a complete trajectory of rewards to estimate values

$$q_\pi(s,a) \doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots \mid S_t = s, A_t = a \right]$$

- We don’t have to wait!

Bootstrapping: key idea in Q-learning and all temporal-difference (TD) learning

- You might think we need a complete trajectory of rewards to estimate values

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \mid S_t = s, A_t = a\right]$$

- We don't have to wait!

- Lets use q_π(next-state,next-action) as a replacement for $R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \ldots$

Bootstrapping: key idea in Q-learning and all temporal-difference (TD) learning

- You might think we need a complete trajectory of rewards to estimate values

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \mid S_t = s, A_t = a\right]$$

- We don't have to wait!

- Lets use q_π(next-state,next-action) as a replacement for $R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \ldots$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \mid S_t = s, A_t = a\right]$$

Bootstrapping: key idea in Q-learning and all temporal-difference (TD) learning

- You might think we need a complete trajectory of rewards to estimate values

$$q_\pi(s,a) \doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \mid S_t = s, A_t = a\right]$$

- We don't have to wait!

- Lets use q_π(next-state,next-action) as a replacement for $R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \ldots$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma \boxed{R_{t+2} + \gamma^2 R_{t+3} \ldots} \mid S_t = s, A_t = a\right]$$

Bootstrapping: key idea in Q-learning and all temporal-difference (TD) learning

- You might think we need a complete trajectory of rewards to estimate values

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \mid S_t = s, A_t = a\right]$$

- We don't have to wait!

- Lets use q_π(next-state,next-action) as a replacement for $R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \ldots$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma \boxed{R_{t+2} + \gamma^2 R_{t+3} \ldots} \mid S_t = s, A_t = a\right]$$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

Bootstrapping: key idea in Q-learning and all temporal-difference (TD) learning

- You might think we need a complete trajectory of rewards to estimate values

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \mid S_t = s, A_t = a\right]$$

- We don't have to wait!

- Lets use q_π(next-state,next-action) as a replacement for $R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \ldots$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma \boxed{R_{t+2} + \gamma^2 R_{t+3} \ldots} \mid S_t = s, A_t = a\right]$$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma \underbrace{q_\pi(S_{t+1}, A_{t+1})}_{\text{use Q-learning's estimate in its update}} \mid S_t = s, A_t = a\right]$$

Bootstrapping: key idea in Q-learning and all temporal-difference (TD) learning

- You might think we need a complete trajectory of rewards to estimate values

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots \mid S_t = s, A_t = a\right]$$

- We don't have to wait!

- Lets use q_π(next-state,next-action) as a replacement for $R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \ldots$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma \boxed{R_{t+2} + \gamma^2 R_{t+3} \ldots} \mid S_t = s, A_t = a\right]$$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma \underbrace{q_\pi(S_{t+1}, A_{t+1})}_{\text{use Q-learning's estimate in its update}} \mid S_t = s, A_t = a\right]$$

- Q-learning update is based on the Bellman optimality equation:

$$q_\star(s,a) = \mathbb{E}_\pi\left[\underbrace{R_{t+1} + \gamma \max_{a'} q_\star(S_{t+1}, a')}_{\text{Q-learning's target for } Q(S_t, A_t)} \mid S_t = s, A_t = a\right]$$

Bellman equations

Bellman equations

- Define a relationship between the value of a state and the value of its possible successor states

Bellman equations

- Define a relationship between the value of a state and the value of its possible successor states

$$q_\pi(s,a) = \sum_{s'} \sum_{r} p(s',r \mid s,a)\left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s',a')\right]$$

Bellman equations

- Define a relationship between the value of a state and the value of its possible successor states

$$q_\pi(s,a) = \sum_{s'} \sum_{r} p(s',r \mid s,a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s',a') \right]$$

- There are Bellman Equations for v_π, v_*, and q_*

Bellman equations

- Define a relationship between the value of a state and the value of its possible successor states

$$q_\pi(s,a) = \sum_{s'} \sum_{r} p(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a') \right]$$

- There are Bellman Equations for v_π, v_*, and q_*

- Classical Dynamic Programming algorithms (planning), compute value functions and optimal policies using Bellman Equations, given **p** (the model)

Bellman equations

- Define a relationship between the value of a state and the value of its possible successor states

$$q_\pi(s,a) = \sum_{s'} \sum_r p(s',r \mid s,a)\left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s',a')\right]$$

- There are Bellman Equations for v_π, v_*, and q_*
- Classical Dynamic Programming algorithms (planning), compute value functions and optimal policies using Bellman Equations, given **p** (the model)
- **Many** algorithms in RL, like Q-learning, can be seen as approximately solving the Bellman Equation with samples from the environment (model-free)

Key characteristics of RL

- ☑ Evaluative feedback (reward)
- ☑ Delayed consequences
- ☑ Must associate different actions with different situations
- ☑ Online and Incremental learning
- Need for trial and error, to explore as well as exploit
- Non-stationarity

} Characteristics of solution/alg

The Exploration-Exploitation dilemma

The Exploration-Exploitation dilemma

- You cannot choose the action with the max value every time
  - what if your estimates of q_π are wrong?
  - you must try all the actions…an infinite number of times, in each state!

The Exploration-Exploitation dilemma

- You cannot choose the action with the max value every time
  - what if your estimates of q_π are wrong?
  - you must try all the actions…an infinite number of times, in each state!
- But, you can't explore all the time

The Exploration-Exploitation dilemma

- You cannot choose the action with the max value every time
  - what if your estimates of q_π are wrong?
  - you must try all the actions…an infinite number of times, in each state!
- But, you can’t explore all the time
- You must balance exploiting (picking what you think is the best), and exploring (refining your estimates)

How does Q-learning handle exploration?

How does Q-learning handle exploration?

Choose actions in any way, perhaps based on Q, such that all actions are taken in all states (infinitely often in the limit)

How does Q-learning handle exploration?

Choose actions in any way, perhaps based on Q, such that all actions are taken in all states (infinitely often in the limit)

e.g., ε-greedy:

How does Q-learning handle exploration?

Choose actions in any way, perhaps based on Q, such that all actions are taken in all states (infinitely often in the limit)

e.g., ε-greedy:

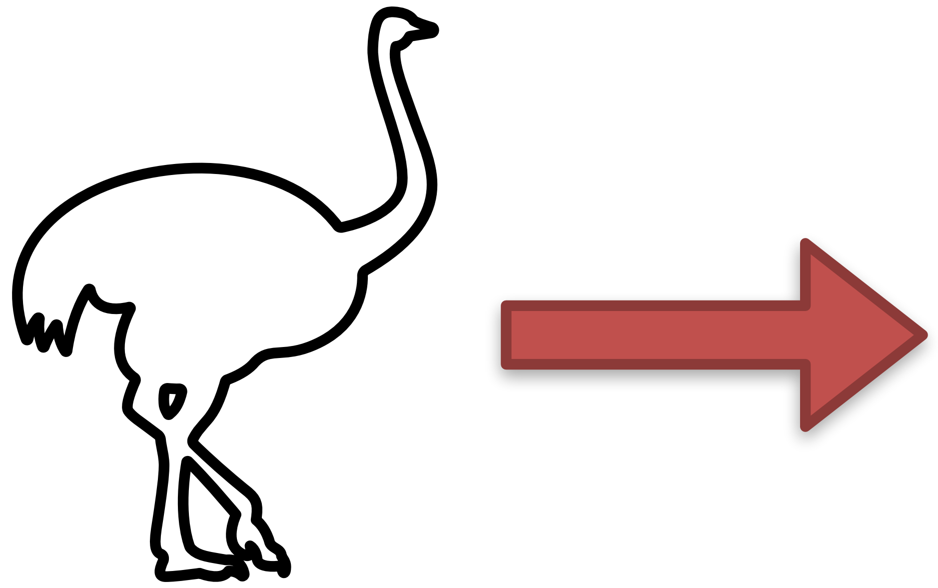
$$1 - \epsilon$$

How does Q-learning handle exploration?

Choose actions in any way, perhaps based on Q, such that all actions are taken in all states (infinitely often in the limit)

e.g., ε-greedy:

$$1 - \epsilon$$



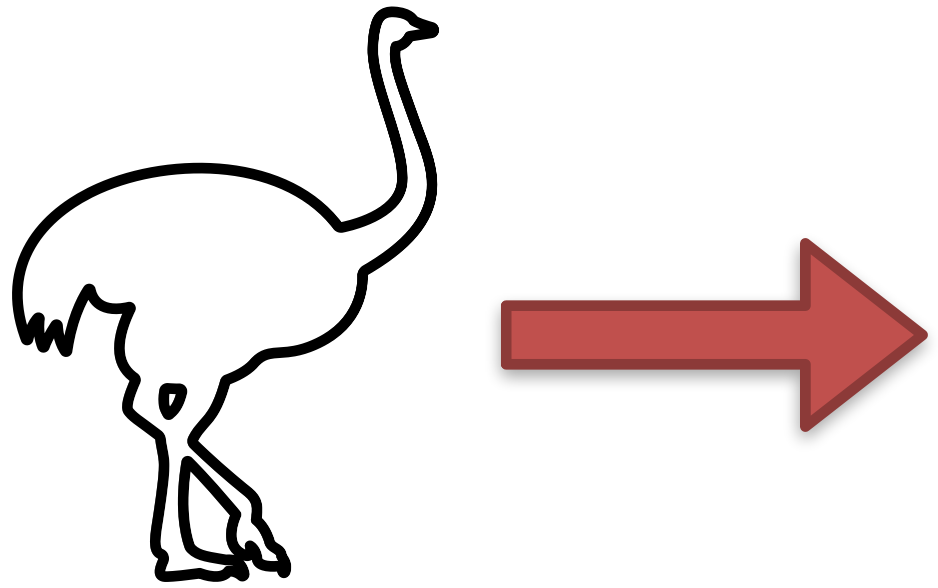
$$A_t = \underset{a}{\operatorname{argmax}}\ Q(S_t, a)$$

How does Q-learning handle exploration?

Choose actions in any way, perhaps based on Q, such that all actions are taken in all states (infinitely often in the limit)

e.g., ε-greedy:

$$1 - \epsilon \qquad\qquad\qquad \epsilon$$



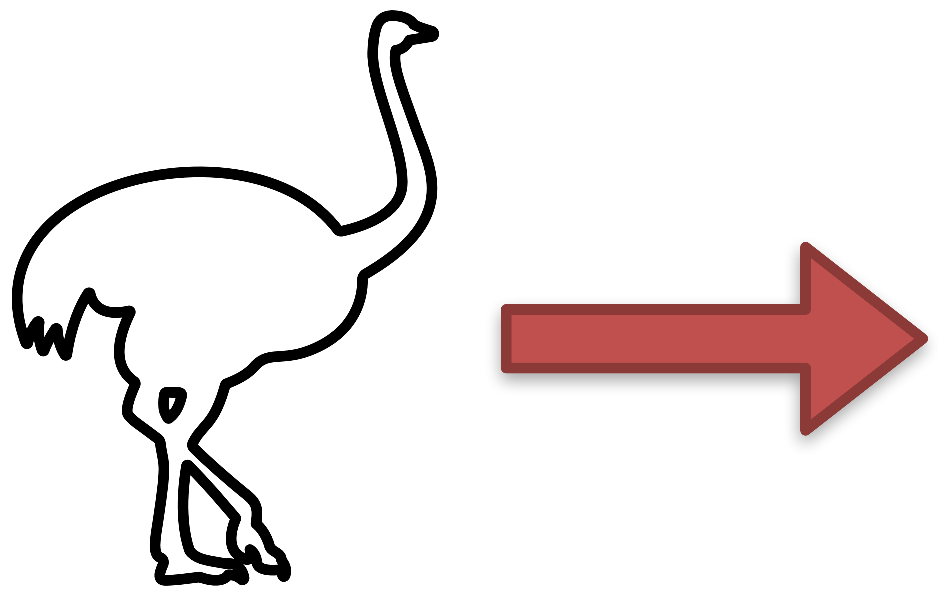
$$A_t = \underset{a}{\operatorname{argmax}}\ Q(S_t, a)$$

How does Q-learning handle exploration?

Choose actions in any way, perhaps based on Q, such that all actions are taken in all states (infinitely often in the limit)

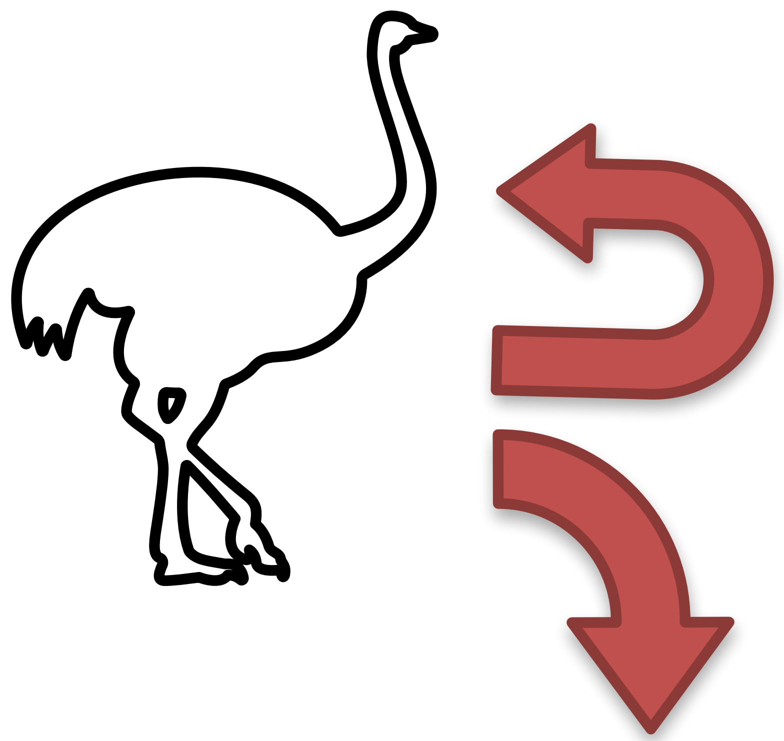
e.g., ε-greedy:

$$1 - \epsilon$$



$$A_t = \underset{a}{\operatorname{argmax}}\; Q(S_t, a)$$

$$\epsilon$$



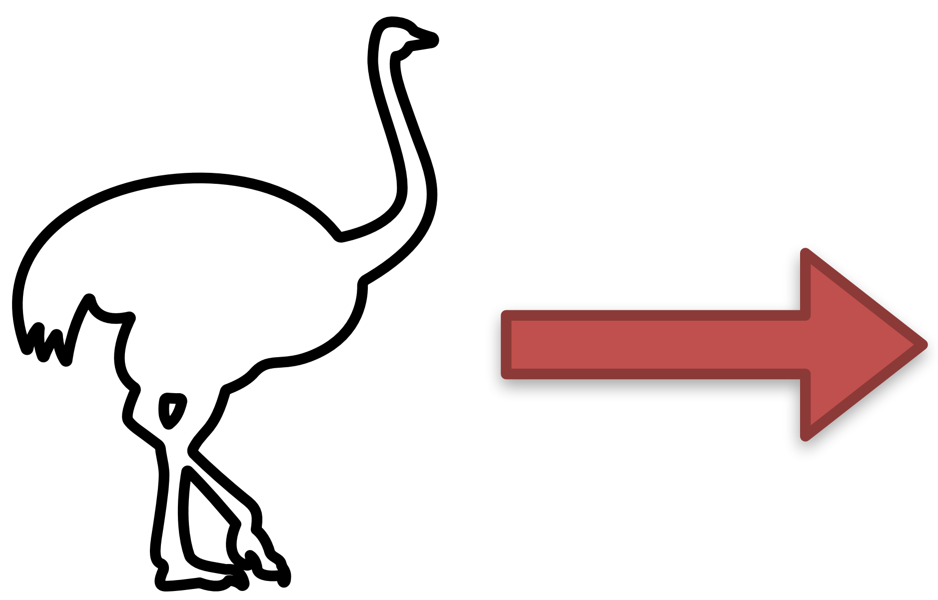
$$A_t = \text{Random action}$$

How does Q-learning handle exploration?

Choose actions in any way, perhaps based on Q, such that all actions are taken in all states (infinitely often in the limit)

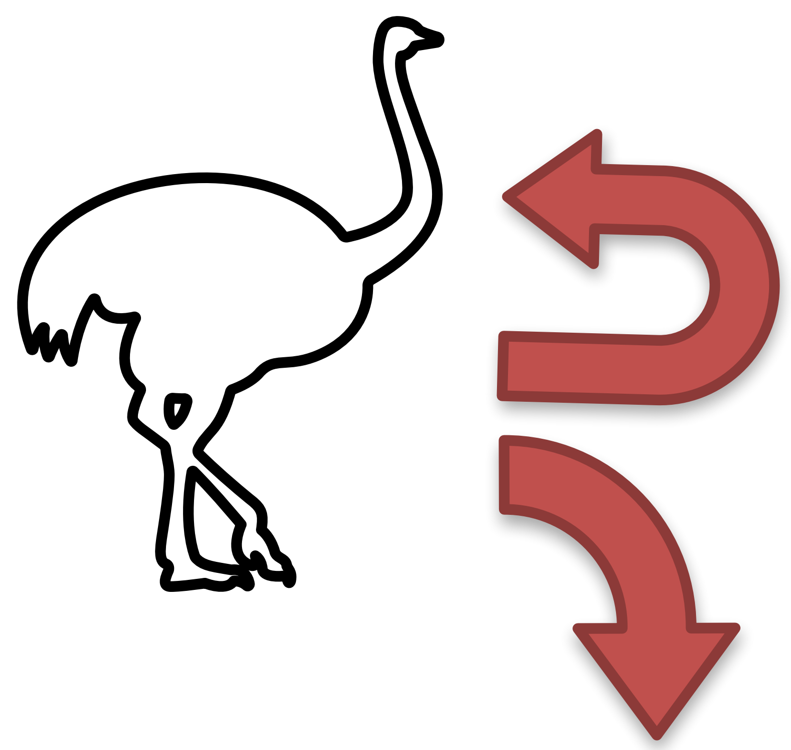
e.g., ε-greedy: dithering or undirected exploration

$1 - \epsilon$



$$A_t = \underset{a}{\operatorname{argmax}}\ Q(S_t, a)$$

ϵ



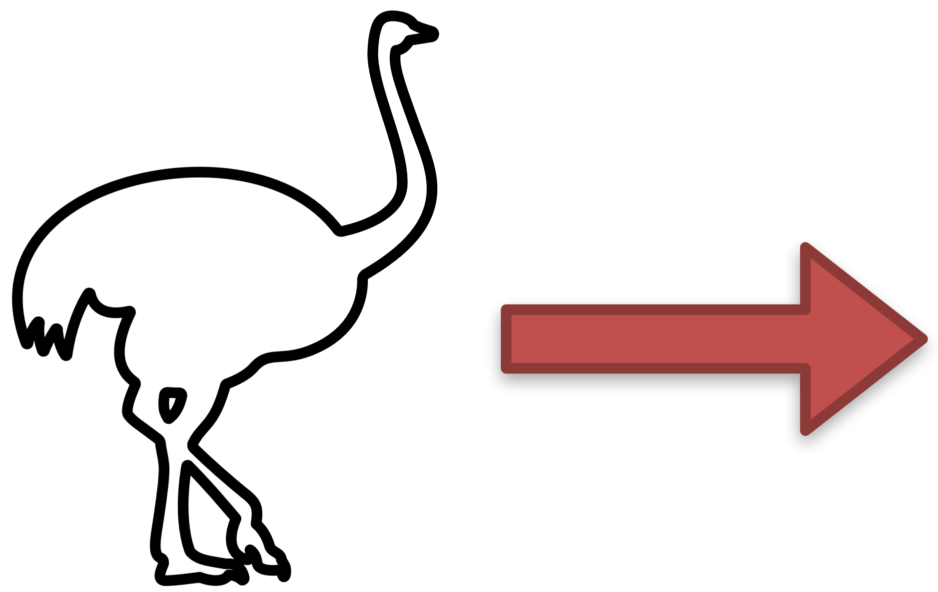
$$A_t = \text{Random action}$$

How does Q-learning handle exploration?

Choose actions in any way, perhaps based on Q, such that all actions are taken in all states (infinitely often in the limit)

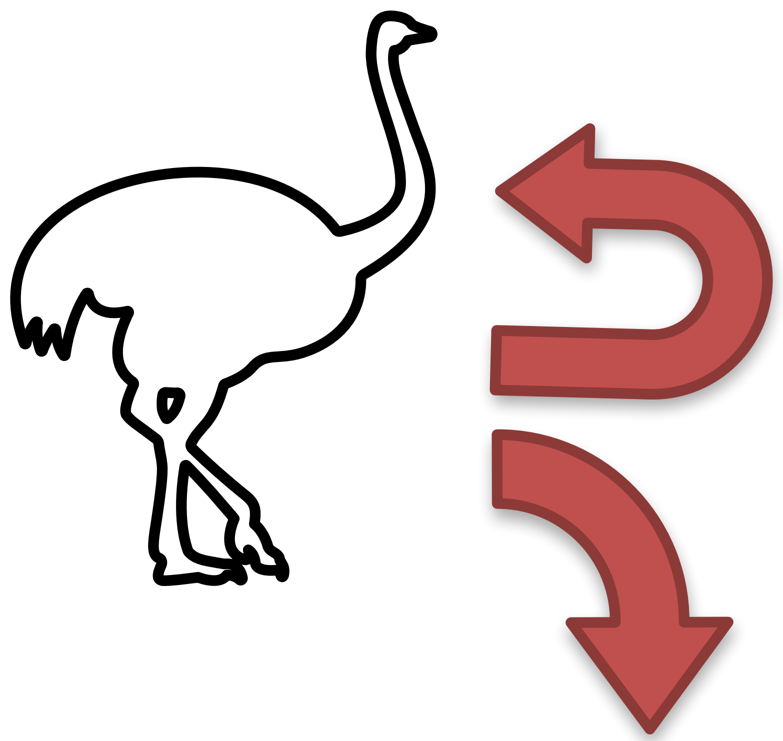
e.g., ε-greedy: dithering or undirected exploration

$$1 - \epsilon$$



$$A_t = \operatorname*{argmax}_a Q(S_t, a)$$

$$\epsilon$$



$$A_t = \text{Random action}$$

- optimistic initial values
- R-max, MBIE
(require models)

Off-policy learning

Off-policy learning

- Learning about the value of **one policy** while using **another policy** to generate the trajectory

Off-policy learning

- Learning about the value of **one policy** while using **another policy** to generate the trajectory

- **Q-learning** is off-policy:

  - the agent learns about the value of its *deterministic greedy policy*

    - which gradually becomes *optimal*

  - from data generated while behaving in a more **exploratory** manner

Off-policy learning

- Learning about the value of **one policy** while using **another policy** to generate the trajectory

- **Q-learning** is off-policy:

  - the agent learns about the value of its *deterministic greedy policy*

    - which gradually becomes *optimal*

  - from data generated while behaving in a more **exploratory** manner

- Also useful for batch-RL, learning from demonstration, and parallel learning (e.g., many value functions, many policies, option-models)

Key characteristics of RL

- ☑ Evaluative feedback (reward)
- ☑ Delayed consequences
- ☑ Must associate different actions with different situations
- ☑ Online and Incremental learning
- ☑ Need for trial and error, to explore as well as exploit
- Non-stationarity

Q-learning: learning never ends

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

Initialize S
Loop for each step
Choose A from S using policy derived from Q (e.g., ε-greedy)
Take action A, observe R, S'
$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
$S \leftarrow S'$

Q-learning: learning never ends

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily

Initialize S
Loop for each step
Choose A from S using policy derived from Q (e.g., ε-greedy)
Take action A, observe R, S'
$Q(S, A) \leftarrow Q(S, A) + \boxed{\alpha}\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
$S \leftarrow S'$

Key characteristics of RL

- ☑ Evaluative feedback (reward)
- ☑ Delayed consequences
- ☑ Must associate different actions with different situations
- ☑ Online and Incremental learning **Bootstrapping**
- ☑ Need for trial and error, to explore as well as exploit
- ☑ Non-stationarity **Never-ending learning**

MDPs, value-functions, policies

Randomization, Off-policy

Now how do we do this with approximation?

The need for approximation

- In real world problems, **tables** of values would become intractably large
  - sometimes the state-space is too large (e.g., Go)
  - sometimes the state-space is continuous
- Instead using tables for our value functions, we will use parameterized functions
- Frame learning these approximate value functions as a supervised learning problem:
  - new challenge balancing **Generalisation** and **Discrimination**

Function approximation

- Represent the action-value function by a parameterized function with parameters $\mathbf{w} \in \mathbb{R}^n$

$$\hat{q}(s, a, \mathbf{w}) \approx q_{\star}(s, a)$$

- The approximator could be a NN, with the weights being the parameters of the network
  - or simply a linear weighting of fixed features
- For large applications, it is important that all computations scale linearly with the number of parameters

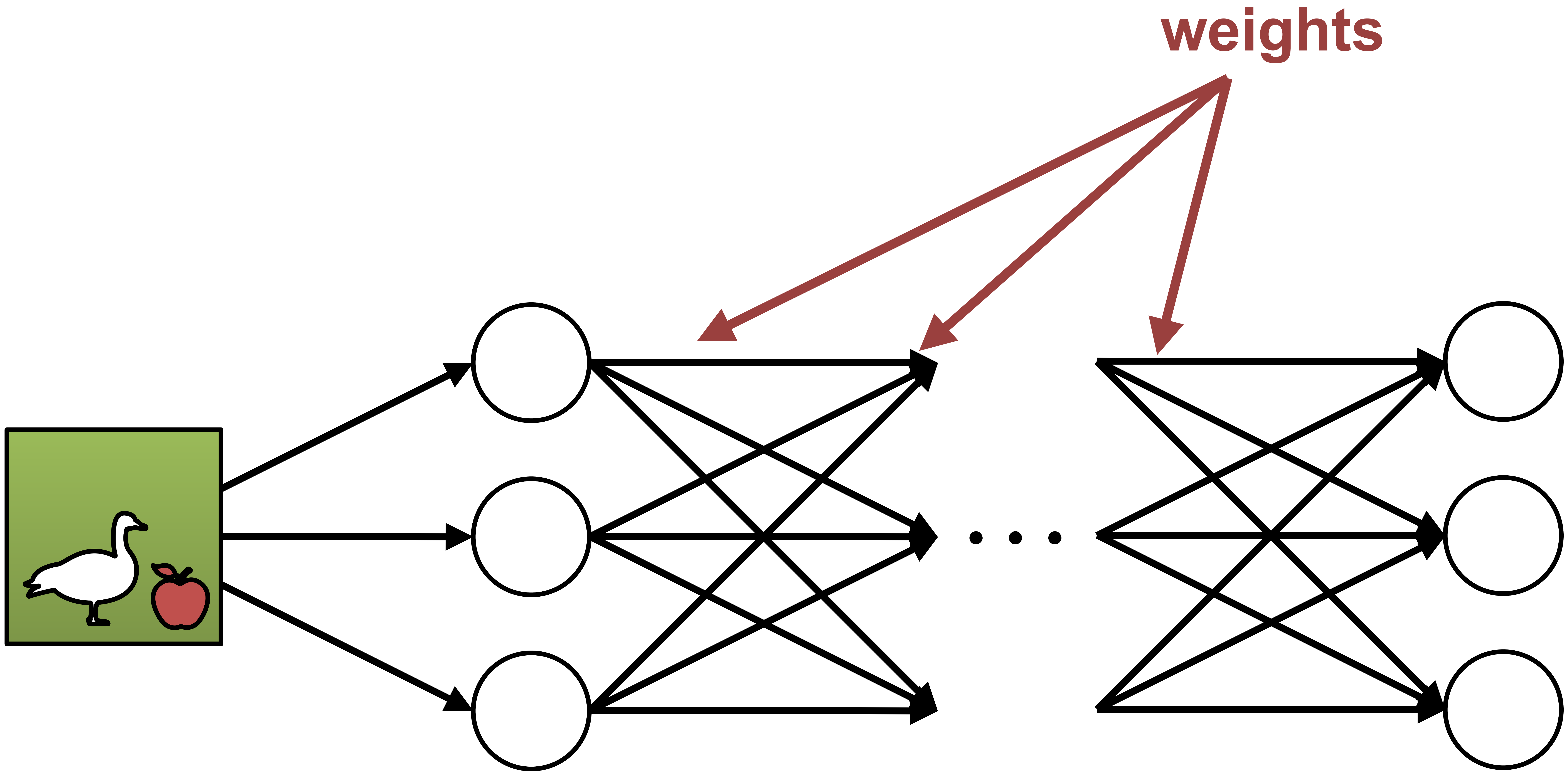
Function approximation

- Represent the action-value function by a parameterized function with parameters $\mathbf{w} \in \mathbb{R}^n$

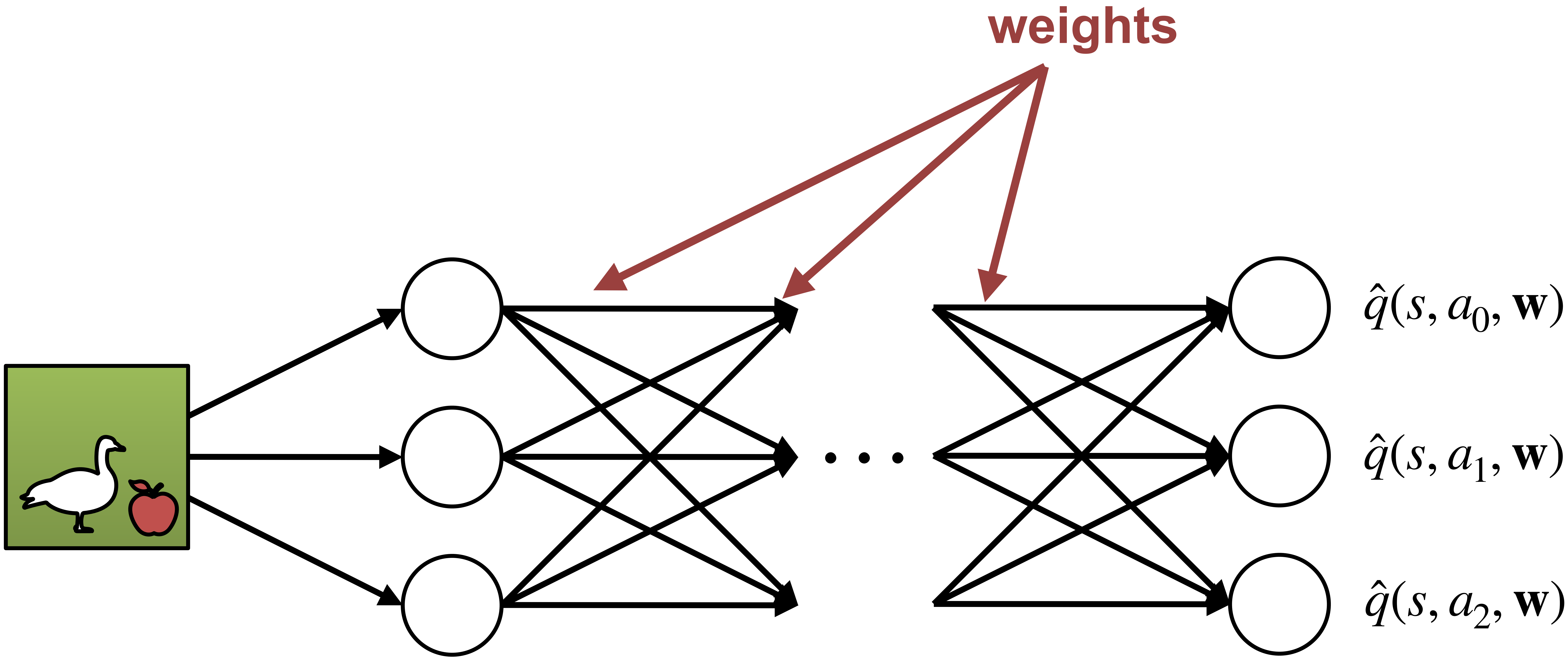
$$\hat{q}(s, a, \mathbf{w}) \approx q_\star(s, a) \approx q_\pi(s, a)$$

- The approximator could be a NN, with the weights being the parameters of the network
    - or simply a linear weighting of fixed features
- For large applications, it is important that all computations scale linearly with the number of parameters

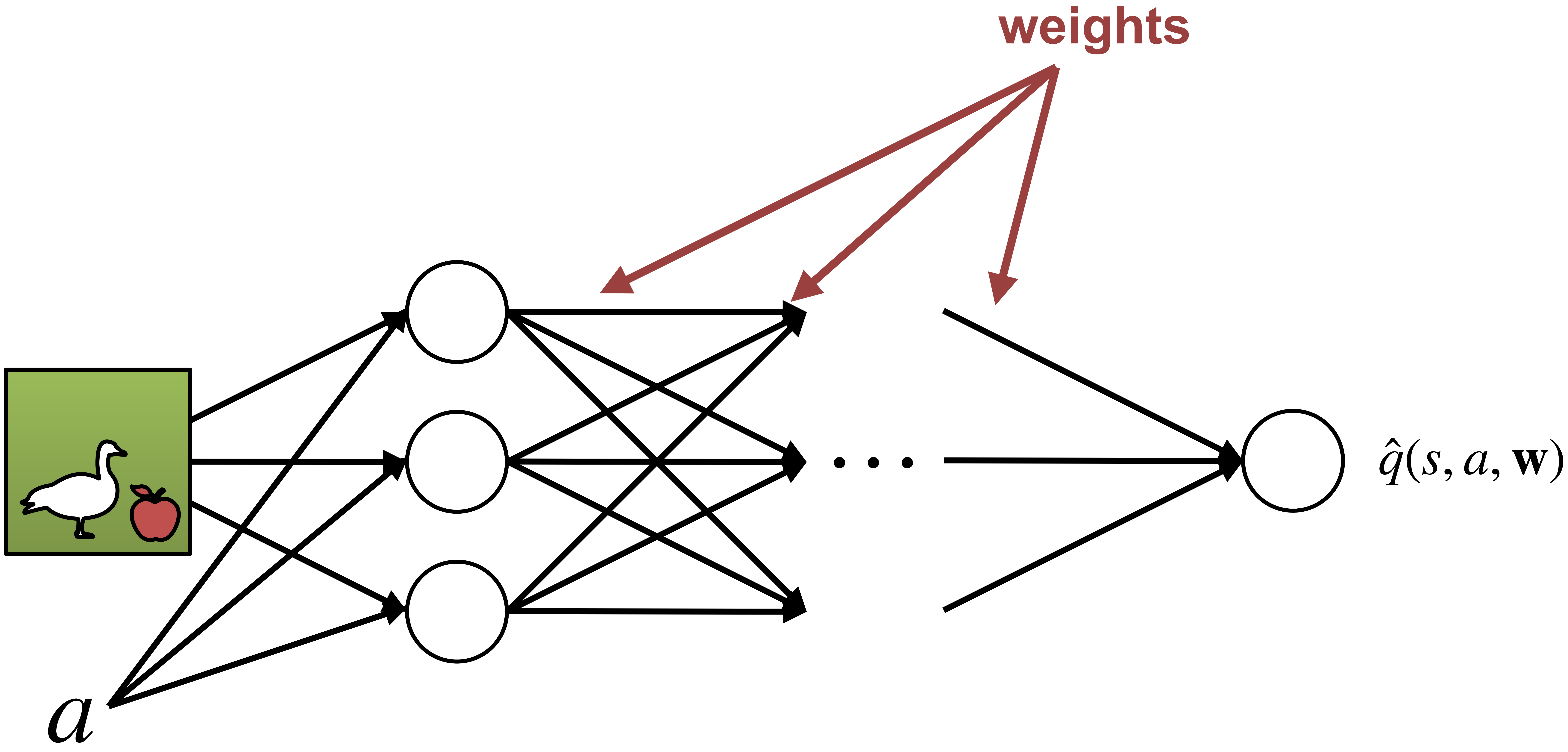
Approximating q_π with an NN



Approximating q_π with an NN



Approximating q_π with an NN



Generalization: Updates to One State Affect the Value of Other States

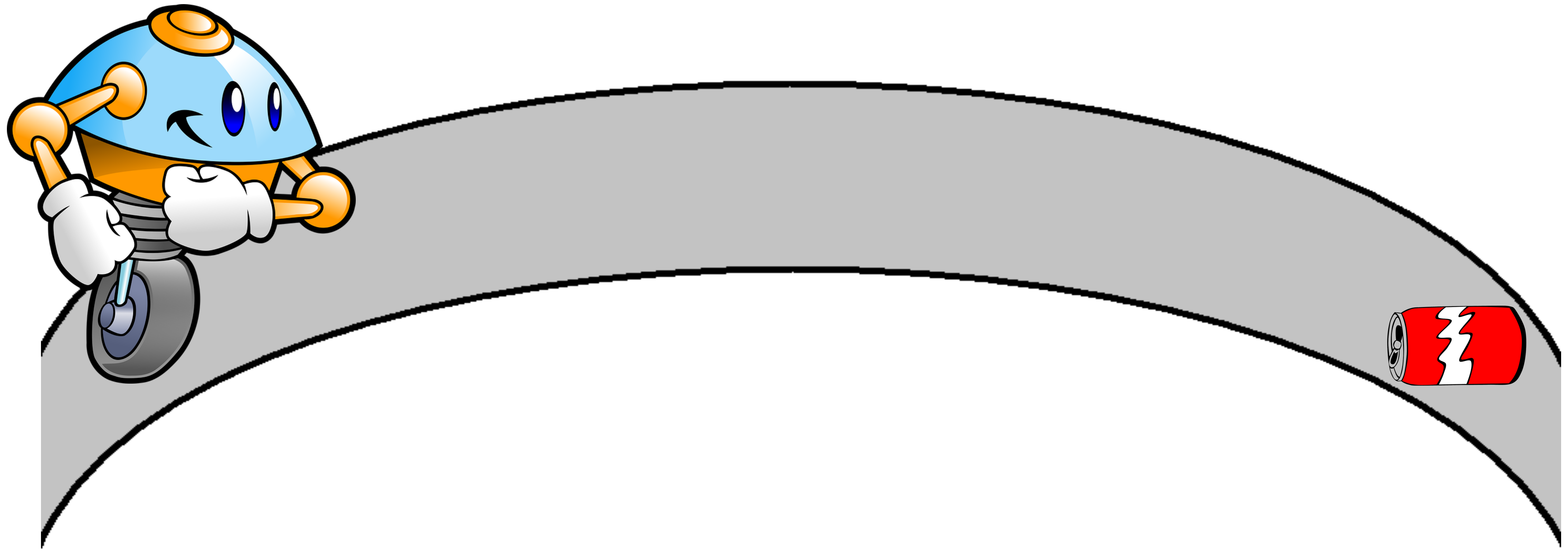
State	Action	Q
s_1	F	**4**
s_2	F	**-4**
s_3	F	**2**
s_4	F	**10**
s_5	F	**4**

Generalization: Updates to One State Affect the Value of Other States

State	Action	Q
s_1	F	**-3**
s_2	F	**-2**
s_3	F	**2**
s_4	F	**-1.7**
s_5	F	**4**

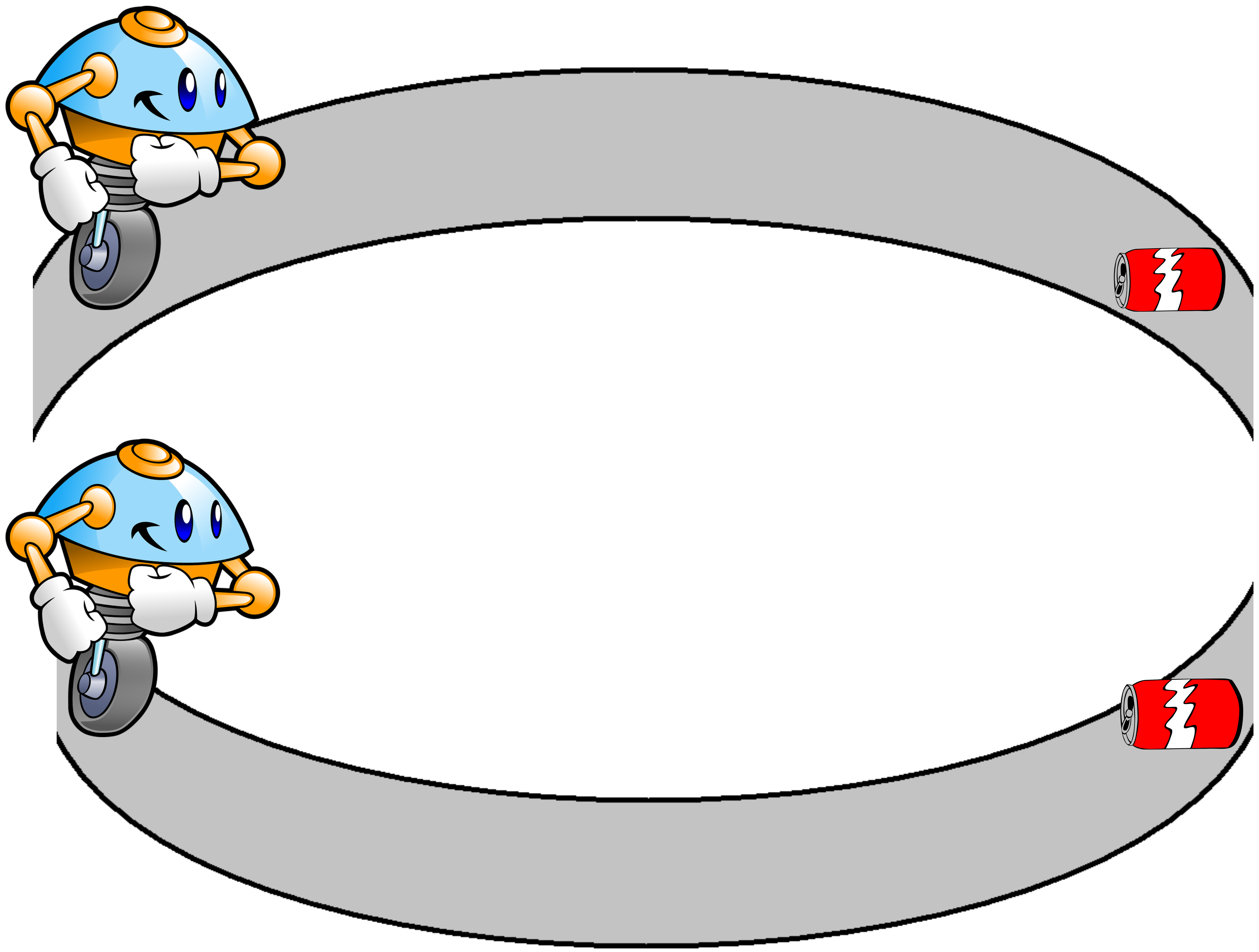
Generalization: Updates to One State Affect the Value of Other States

State	Action	Q
s_1	F	**-3**
s_2	F	**-2**
s_3	F	**2**
s_4	F	**-1.7**
s_5	F	**4**



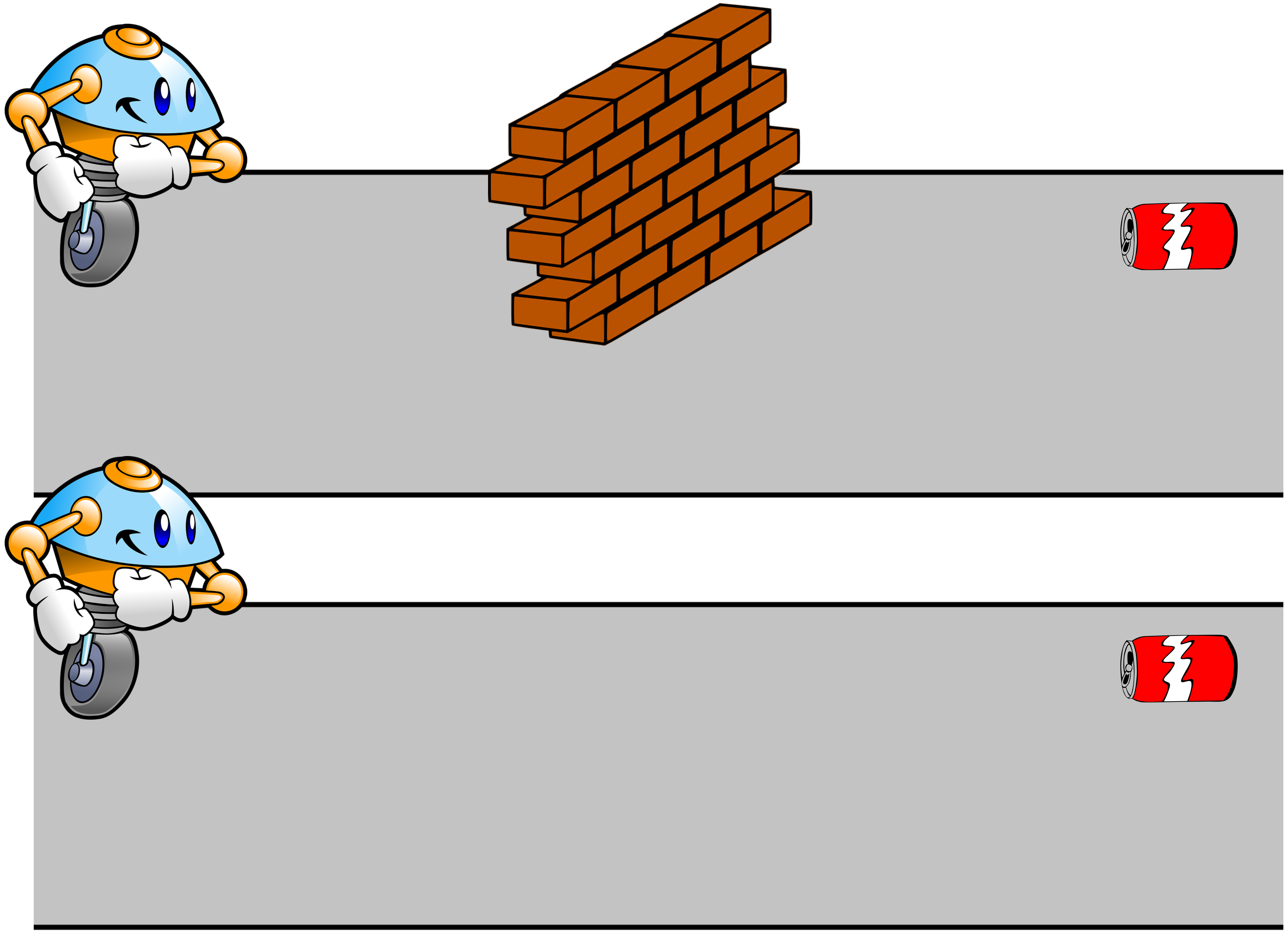
Generalization: Updates to One State Affect the Value of Other States

State	Action	Q
s_1	F	**-3**
s_2	F	**-2**
s_3	F	**2**
s_4	F	**-1.7**
s_5	F	**4**

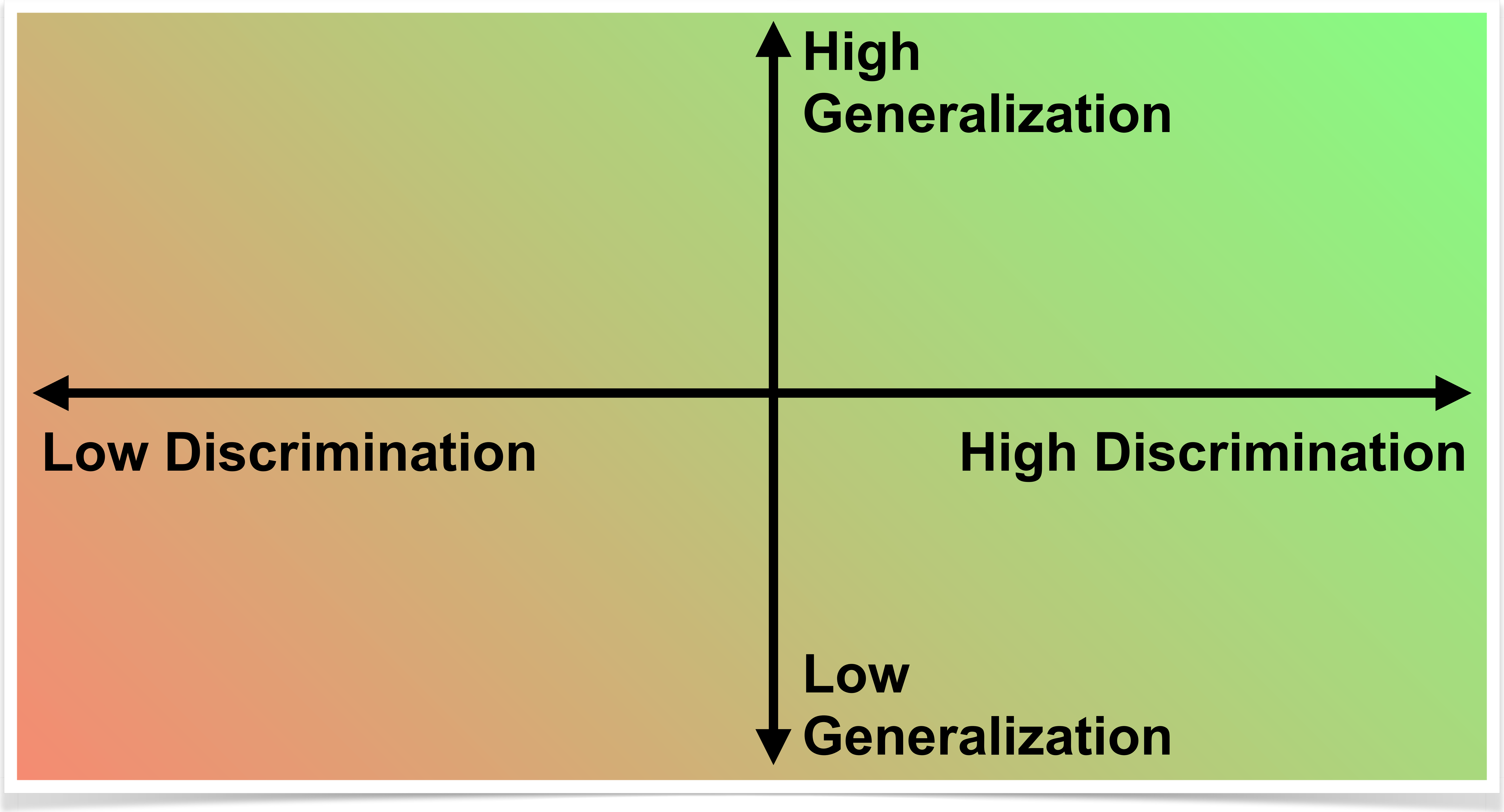


Discrimination: The ability to make the value of two states different

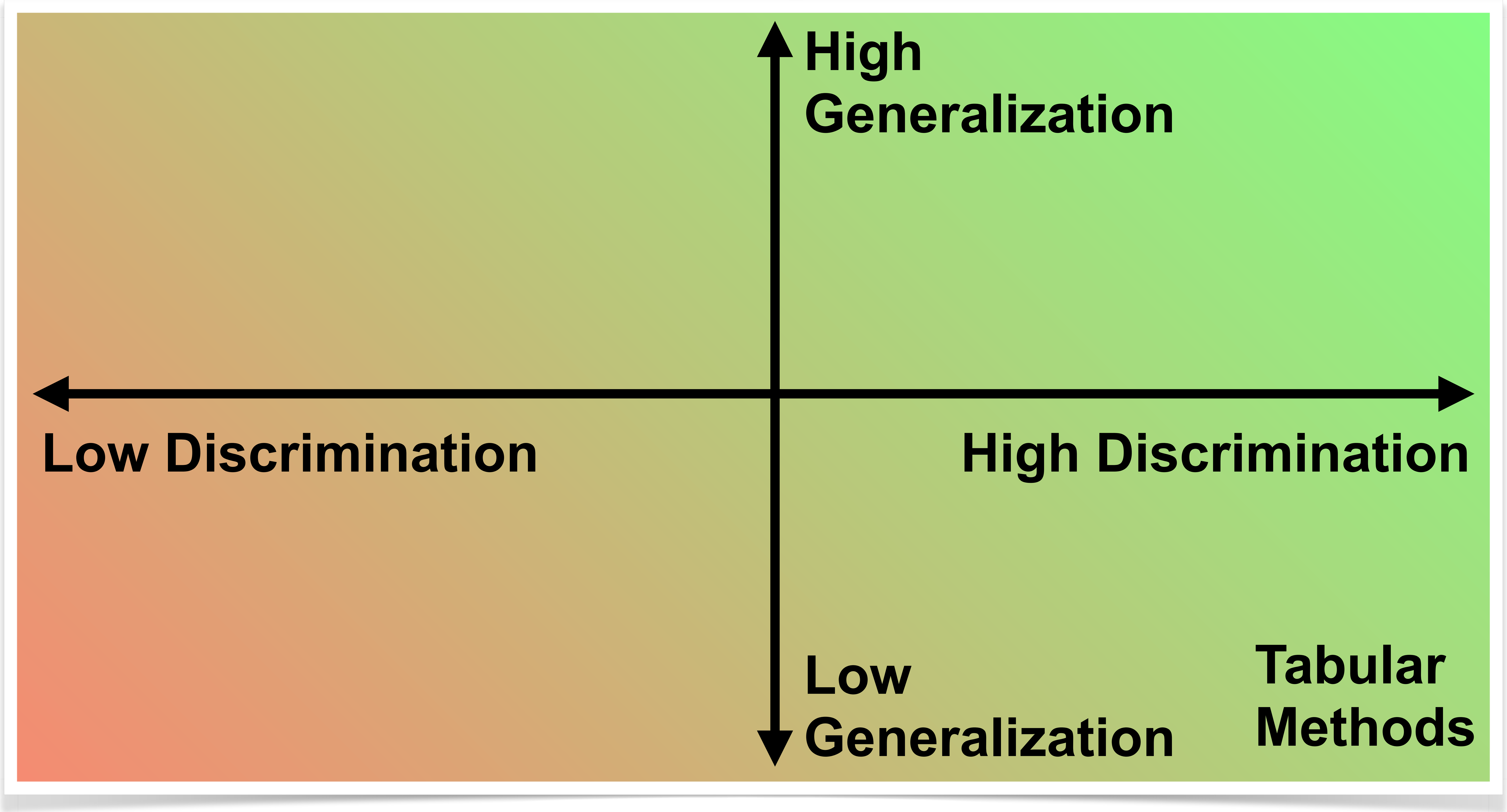
Discrimination: The ability to make the value of two states different



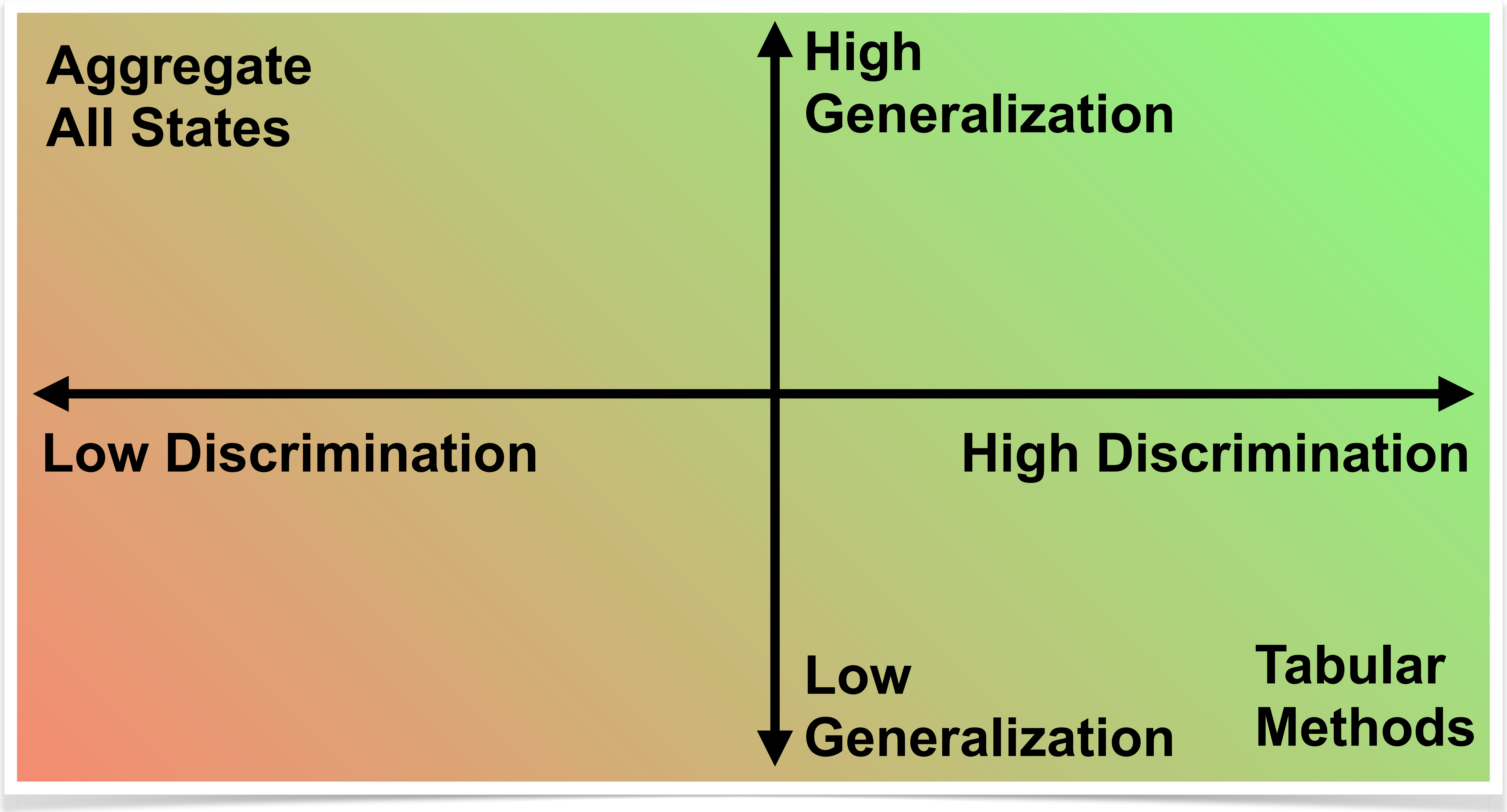
Categorizing methods based on Generalization and Discrimination



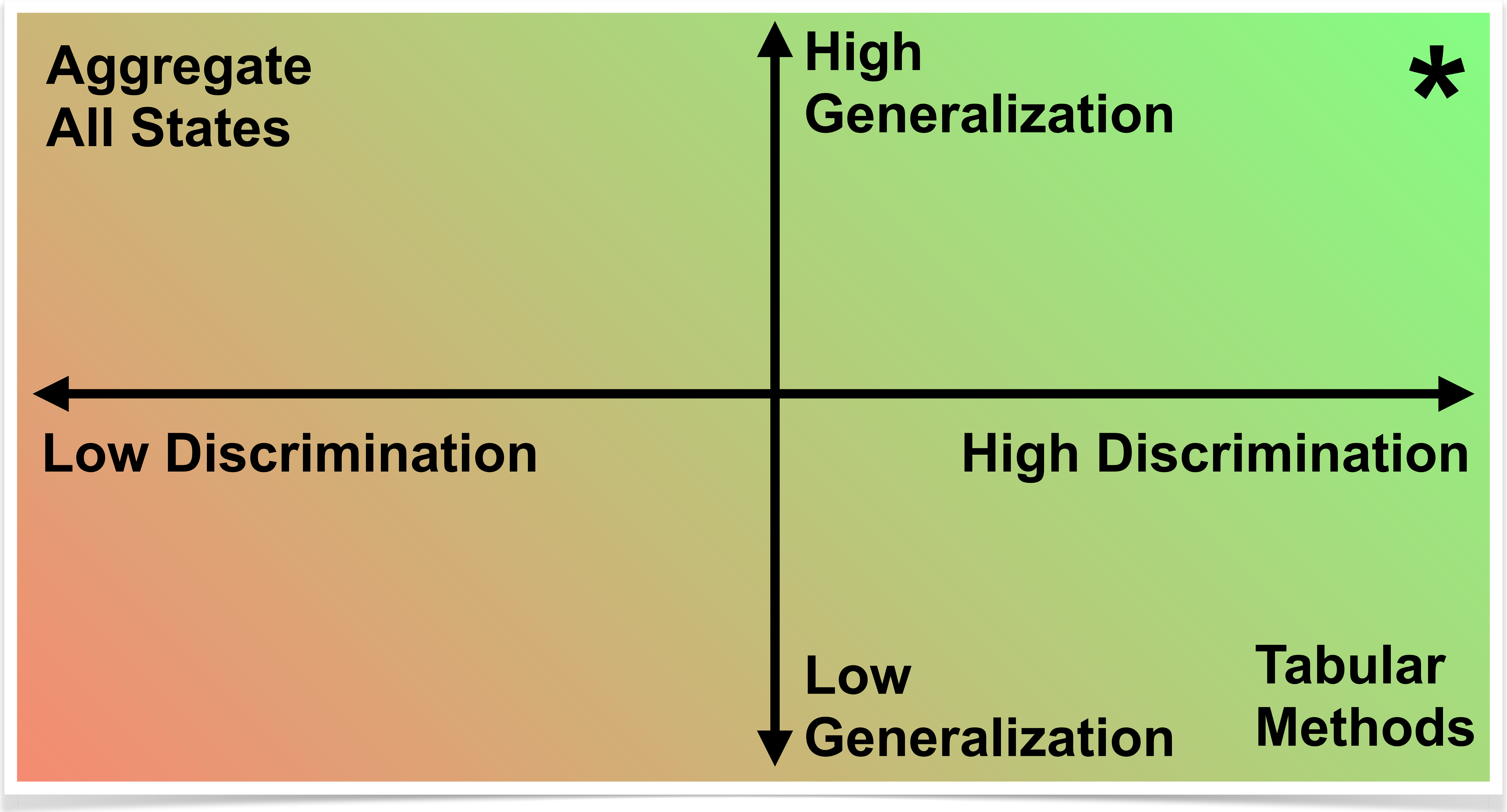
Categorizing methods based on Generalization and Discrimination



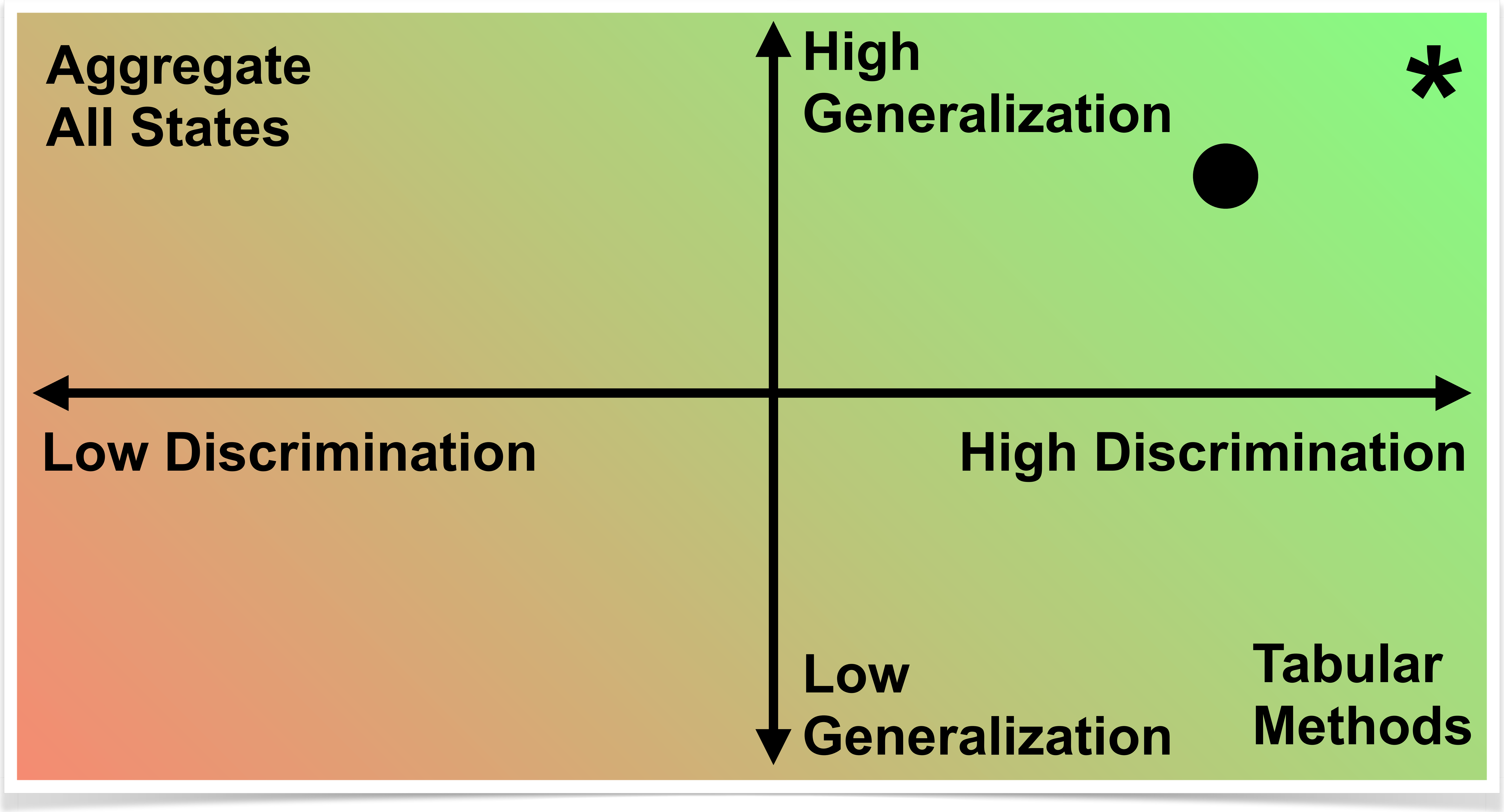
Categorizing methods based on Generalization and Discrimination



Categorizing methods based on Generalization and Discrimination



Categorizing methods based on Generalization and Discrimination



Semi-gradient Q-learning

- There is an obvious generalization of Q-learning to function approximation (Watkins 1989)

- Consider the following objective function:

$$\mathscr{L}(\mathbf{w}) = \mathbb{E}\left[\left(R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})\right)^2\right]$$

- and the update used in Q-learning with function approximation

$$\Delta\mathbf{w} = \alpha\left(R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)\right)\frac{\partial\hat{q}(S_t, A_t, \mathbf{w}_t)}{\mathbf{w}_t}$$

Semi-gradient Q-learning

- There is an obvious generalization of Q-learning to function approximation (Watkins 1989)

- Consider the following objective function:

$$\mathscr{L}(\mathbf{w}) = \mathbb{E}\left[\left(R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}) \ - \ \hat{q}(S_t, A_t, \mathbf{w}) \right)^2 \right]$$

  - and the update used in Q-learning with function approximation

$$\Delta \mathbf{w} = \alpha \left(R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right) \frac{\partial \hat{q}(S_t, A_t, \mathbf{w}_t)}{\mathbf{w}_t}$$

- The target here depends on the **w.** It's like we ignored the gradient of the value of the next state

TD + function approximation can lead to instability

TD + function approximation can lead to instability

- Q-learning can diverge

TD + function approximation can lead to instability

- Q-learning can diverge
- It is not because of **control**,
  - nor exploration, greedification, or sampling

TD + function approximation can lead to instability

- Q-learning can diverge
- It is not because of **control**,
  - nor exploration, greedification, or sampling
- It is not completely due to **non-linear** function approximation

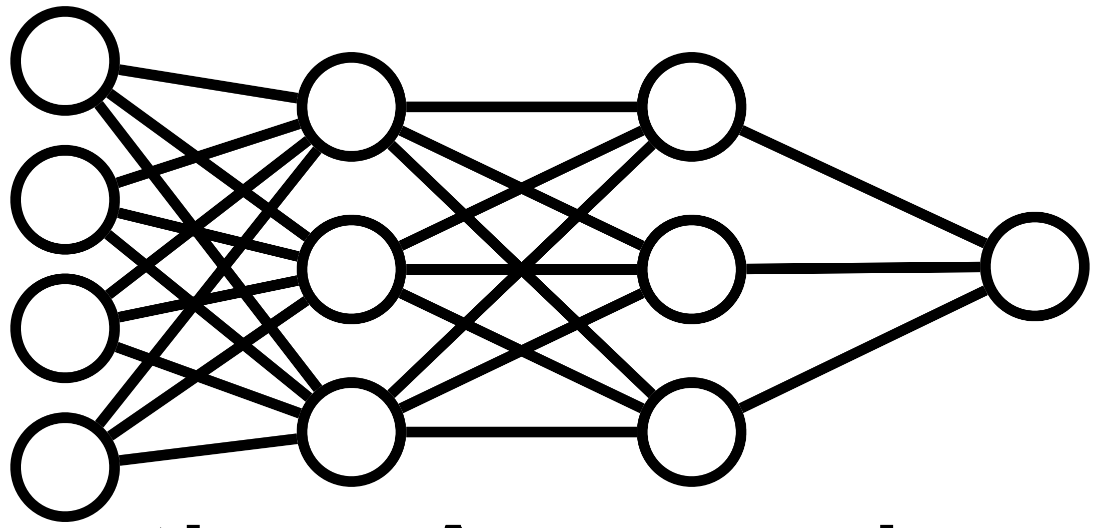
TD + function approximation can lead to instability

- Q-learning can diverge
- It is not because of **control**,
  - nor exploration, greedification, or sampling
- It is not completely due to **non-linear** function approximation
- Dynamic programming methods diverge with function approximation!

TD + function approximation can lead to instability

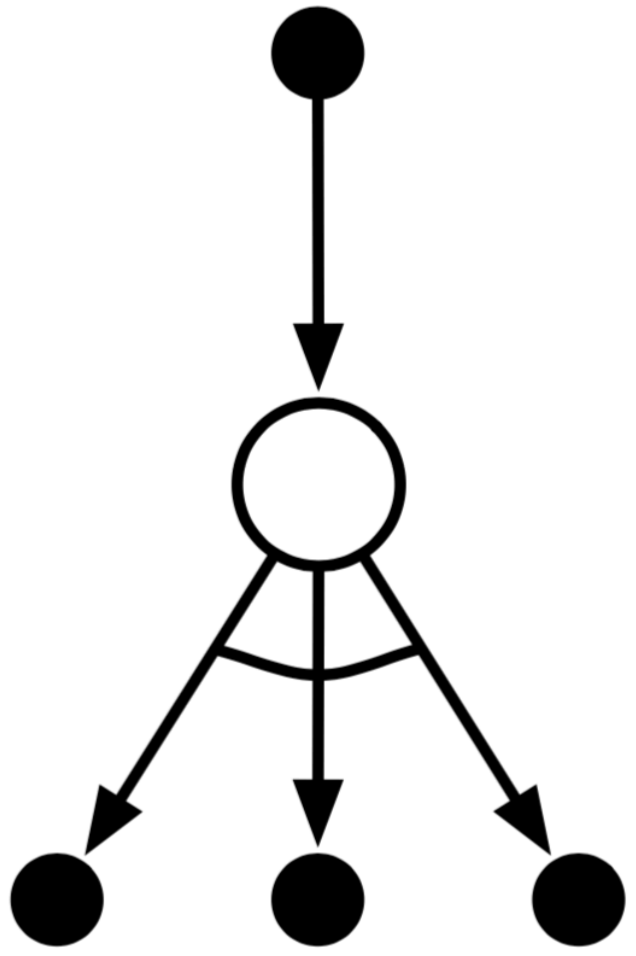
- Q-learning can diverge
- It is not because of **control**,
  - nor exploration, greedification, or sampling
- It is not completely due to **non-linear** function approximation
- Dynamic programming methods diverge with function approximation!
- Even TD with linear function approximation can diverge!
  (in off-policy prediction)

The deadly Triad



Function Approximation

Bootstrapping



Off-policy Learning

Algorithmic solutions to the Triad

- Newish Gradient-TD methods (TDC, GQ, proximal-gradientTD) developed by Maei (2011) and Mahadevan et al (2015) are **sound with off-policy + function approximation**
  - limited practical experience
  - basically unexplored with non-linear function approximation
- New methods to reduce variance in off-policy training (Re-Trace, V-trace, ABQ)
  - can diverge
- Divergence with control and NN is a complex story (van Hasselt et al, 2018)
  - its more likely with larger differences between the policies (common in prioritized replay, sample-based planning, parallel learning)
  - its more likely with larger networks ... both things we might want in our learning systems!

Significant progress in the application of RL

- Learned the world's best player of Backgammon (Tesauro 1995)
- Learned acrobatic helicopter autopilots (Ng, Abbeel, Coates et al 2006+)
- Widely used in the placement and selection of advertisements and pages on the web (e.g., A-B tests)
- Used by Watson to make strategic decisions in Jeopardy!, beating the best human players (IBM 2011)
- Achieved human-level performance on Atari games from pixel-level visual input, in conjunction with deep learning (Deepmind 2015)
- AlphaGo to defeat the world's best Go players (DeepMind, 2016, 2017), AlphaZero to decisively defeat all in Go, chess, and shogi

The good and bad of simulations

The good and bad of simulations

- Simulations are great: they facilitate rapid progress, careful scientific experimentation

The good and bad of simulations

- Simulations are great: they facilitate rapid progress, careful scientific experimentation
- Simulations also allow us to cheat a bit:
  - run on parallel copies of the environment
  - easy to reset states, learn from death, ignore damage
  - knowledge of the underlying dynamics of the world

The good and bad of simulations

- Simulations are great: they facilitate rapid progress, careful scientific experimentation
- Simulations also allow us to cheat a bit:
  - run on parallel copies of the environment
  - easy to reset states, learn from death, ignore damage
  - knowledge of the underlying dynamics of the world
- Lose focus on **data efficiency, parameter sensitivity, exploration challenges**

The good and bad of simulations

- Simulations are great: they facilitate rapid progress, careful scientific experimentation
- Simulations also allow us to cheat a bit:
  - run on parallel copies of the environment
  - easy to reset states, learn from death, ignore damage
  - knowledge of the underlying dynamics of the world
- Lose focus on **data efficiency, parameter sensitivity, exploration challenges**
- Many of the shortcuts we take in simulations are not possible on robots

We are not done!

, while DQN was trained on only 10 M frames.

Game	ES	DQN w/ ϵ-greedy	DQN w/ param noise
Alien	994.0	1535.0	**2070.0**
Amidar	112.0	281.0	**403.5**
BankHeist	225.0	510.0	**805.0**
BeamRider	744.0	**8184.0**	7884.0
Breakout	9.5	**406.0**	390.5
Enduro	95.0	1094	**1672.5**
Freeway	31.0	**32.0**	31.5
Frostbite	370.0	250.0	**1310.0**
Gravitar	**805.0**	300.0	250.0
MontezumaRevenge	**0.0**	**0.0**	**0.0**
Pitfall	**0.0**	-73.0	-100.0
Pong	**21.0**	**21.0**	20.0
PrivateEye	100.0	**133.0**	100.0
Qbert	147.5	**7625.0**	7525.0
Seaquest	1390.0	8335.0	**8920.0**
Solaris	**2090.0**	720.0	400.0
SpaceInvaders	678.5	1000.0	**1205.0**
Tutankham	130.3	109.5	**181.0**
Venture	**760.0**	0	0
WizardOfWor	**3480.0**	2350.0	1850.0
Zaxxon	6380.0	**8100.0**	8050.0

(Plappert et al, 2017)

We are not done!

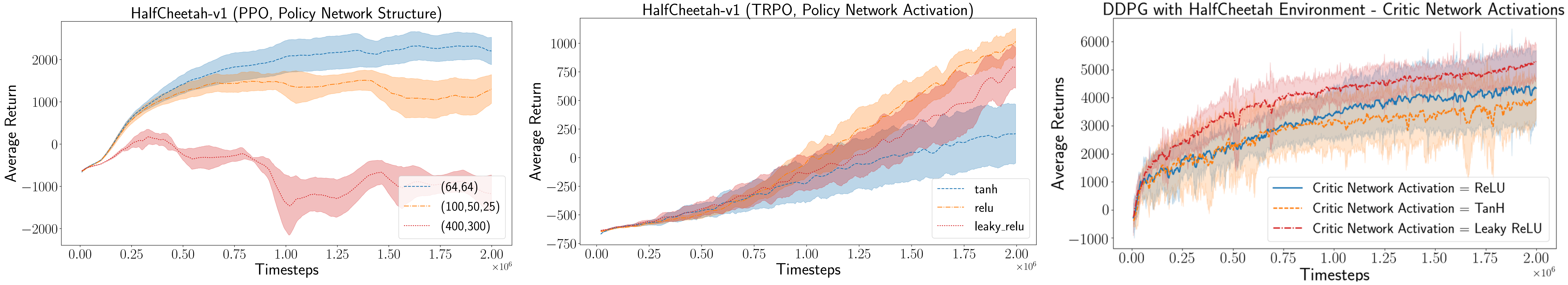
Game	ES	DQN w/ ϵ-greedy	DQN w/ param noise
Alien	994.0	1535.0	**2070.0**
Amidar	112.0	281.0	**403.5**
BankHeist	225.0	510.0	**805.0**
BeamRider	744.0	**8184.0**	7884.0
Breakout	9.5	**406.0**	390.5
Enduro	95.0	1094	**1672.5**
Freeway	31.0	**32.0**	31.5
Frostbite	370.0	250.0	**1310.0**
Gravitar	**805.0**	300.0	250.0
MontezumaRevenge	**0.0**	**0.0**	**0.0**
Pitfall	**0.0**	-73.0	-100.0
Pong	**21.0**	**21.0**	20.0
PrivateEye	100.0	**133.0**	100.0
Qbert	147.5	**7625.0**	7525.0
Seaquest	1390.0	8335.0	**8920.0**
Solaris	**2090.0**	720.0	400.0
SpaceInvaders	678.5	1000.0	**1205.0**
Tutankham	130.3	109.5	**181.0**
Venture	**760.0**	0	0
WizardOfWor	**3480.0**	2350.0	1850.0
Zaxxon	6380.0	**8100.0**	8050.0

(Plappert et al, 2017)

We are not done!

Game	ES	DQN w/ ϵ-greedy	DQN w/ param noise
Alien	994.0	1535.0	**2070.0**
Amidar	112.0	281.0	**403.5**
BankHeist	225.0	510.0	**805.0**
BeamRider	744.0	**8184.0**	7884.0
Breakout	9.5	**406.0**	390.5
Enduro	95.0	1094	**1672.5**
Freeway	31.0	**32.0**	31.5
Frostbite	370.0	250.0	**1310.0**
Gravitar	**805.0**	300.0	250.0
MontezumaRevenge	**0.0**	**0.0**	**0.0**
Pitfall	**0.0**	-73.0	-100.0
Pong	**21.0**	**21.0**	20.0
PrivateEye	100.0	**133.0**	100.0
Qbert	147.5	**7625.0**	7525.0
Seaquest	1390.0	8335.0	**8920.0**
Solaris	**2090.0**	720.0	400.0
SpaceInvaders	678.5	1000.0	**1205.0**
Tutankham	130.3	109.5	**181.0**
Venture	**760.0**	0	0
WizardOfWor	**3480.0**	2350.0	1850.0
Zaxxon	6380.0	**8100.0**	8050.0

(Plappert et al, 2017)

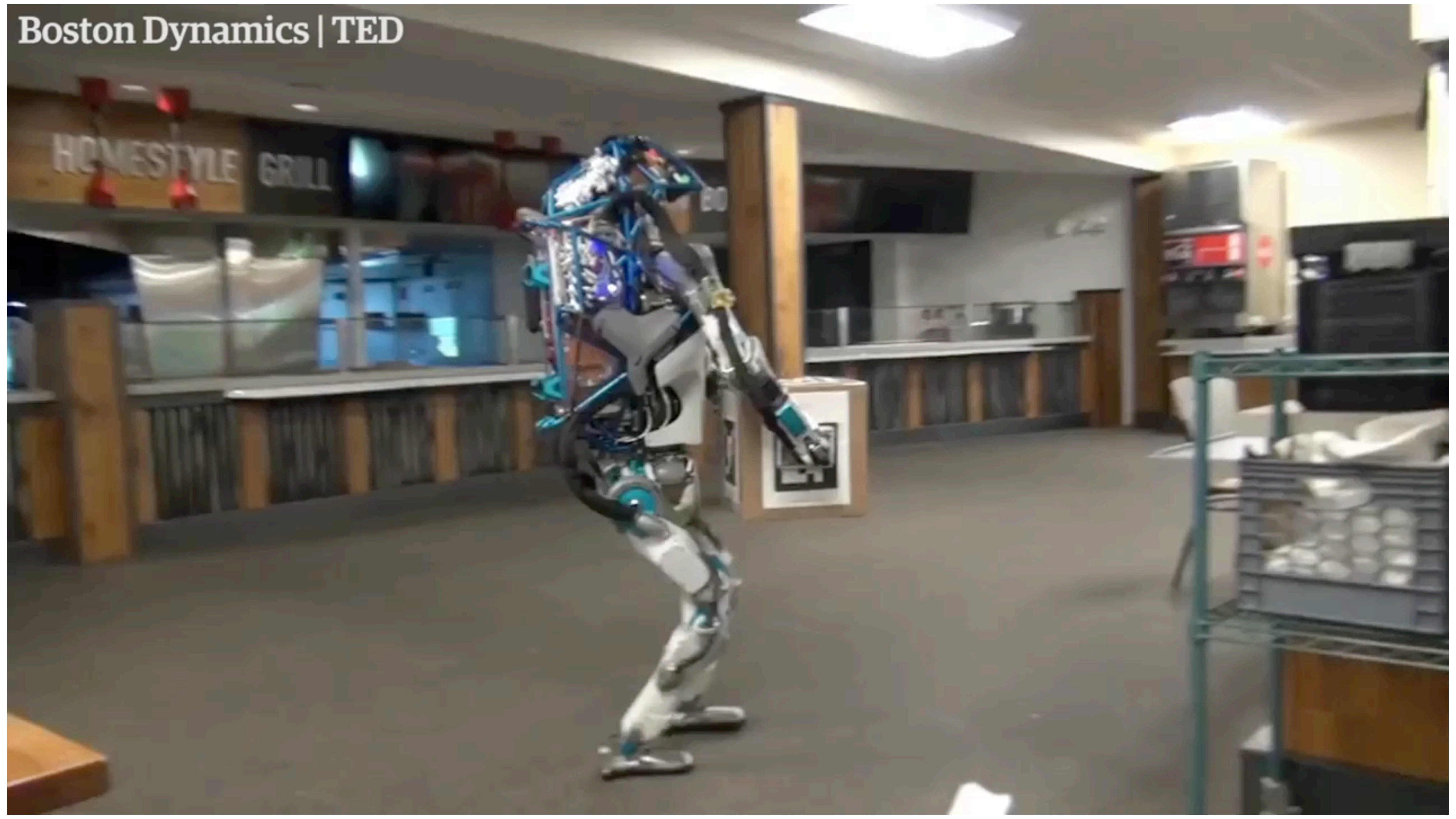


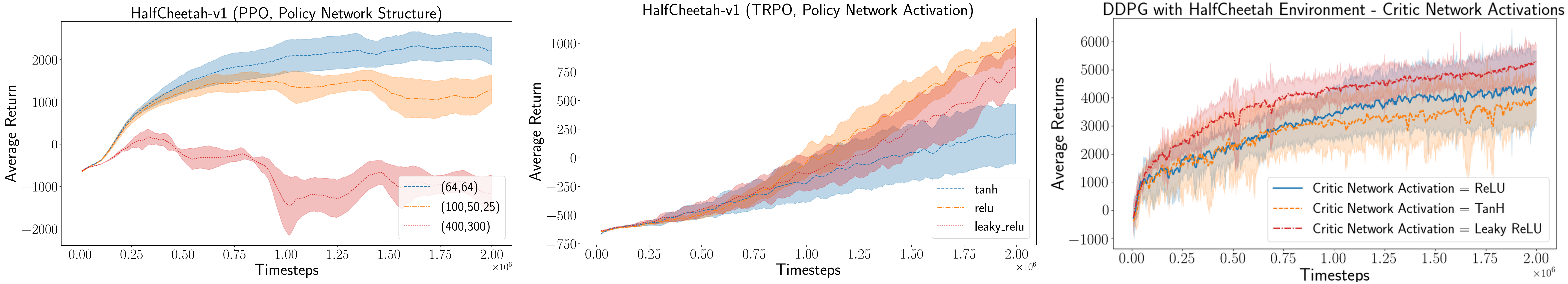
(Henderson et al, 2018)

We are not done!

Game	ES	DQN w/ ϵ-greedy	DQN w/ param noise
Alien	994.0	1535.0	**2070.0**
Amidar	112.0	281.0	**403.5**
BankHeist	225.0	510.0	**805.0**
BeamRider	744.0	**8184.0**	7884.0
Breakout	9.5	**406.0**	390.5
Enduro	95.0	1094	**1672.5**
Freeway	31.0	**32.0**	31.5
Frostbite	370.0	250.0	**1310.0**
Gravitar	**805.0**	300.0	250.0
MontezumaRevenge	**0.0**	**0.0**	**0.0**
Pitfall	**0.0**	-73.0	-100.0
Pong	**21.0**	**21.0**	20.0
PrivateEye	100.0	**133.0**	100.0
Qbert	147.5	**7625.0**	7525.0
Seaquest	1390.0	8335.0	**8920.0**
Solaris	**2090.0**	720.0	400.0
SpaceInvaders	678.5	1000.0	**1205.0**
Tutankham	130.3	109.5	**181.0**
Venture	**760.0**	0	0
WizardOfWor	**3480.0**	2350.0	1850.0
Zaxxon	6380.0	**8100.0**	8050.0

(Plappert et al, 2017)





(Henderson et al, 2018)

The dimensions of RL

- Problems
  - Prediction and control
  - MDPs, Contextual-DP, Contextual Bandits, and simple Bandits
- Solutions
  - Bootstrapping and Monte Carlo (unified by eligibility traces)
  - Tabular and function approximation
  - On-policy and off-policy
  - Model-based and model-free
  - Value-based and policy-based
  - Primitive actions and temporal abstraction

The dimensions of RL