Admin

- Two in-person midterms:
  - This Thursday
  - AND next Tuesday
- **Please Bring your ONECARD**
- **If you are auditing mark `Audit' at the top of your test**
- No discussing the midterm under any circumstances

The midterm

- Closed book: no cheatsheet, no electronic devices
- Test designed to take 50 mins, you have 90
- Always best to explain your answers and show all your steps:
  - What does that mean?
- If you write two answers—one wrong and one right—you get zero on the question
- Tuesday’s midterm != Thursday’s midterm

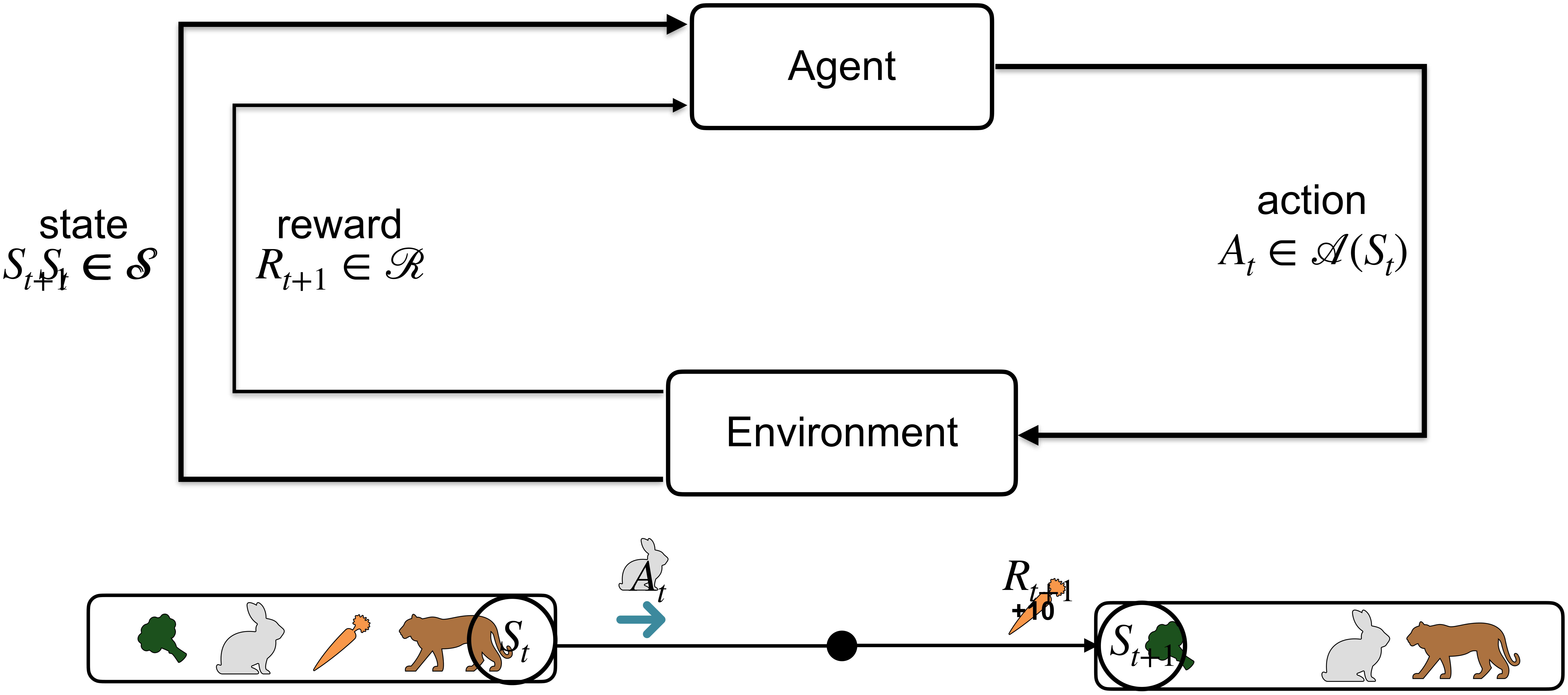
Review Practice Midterm

Course Content Review

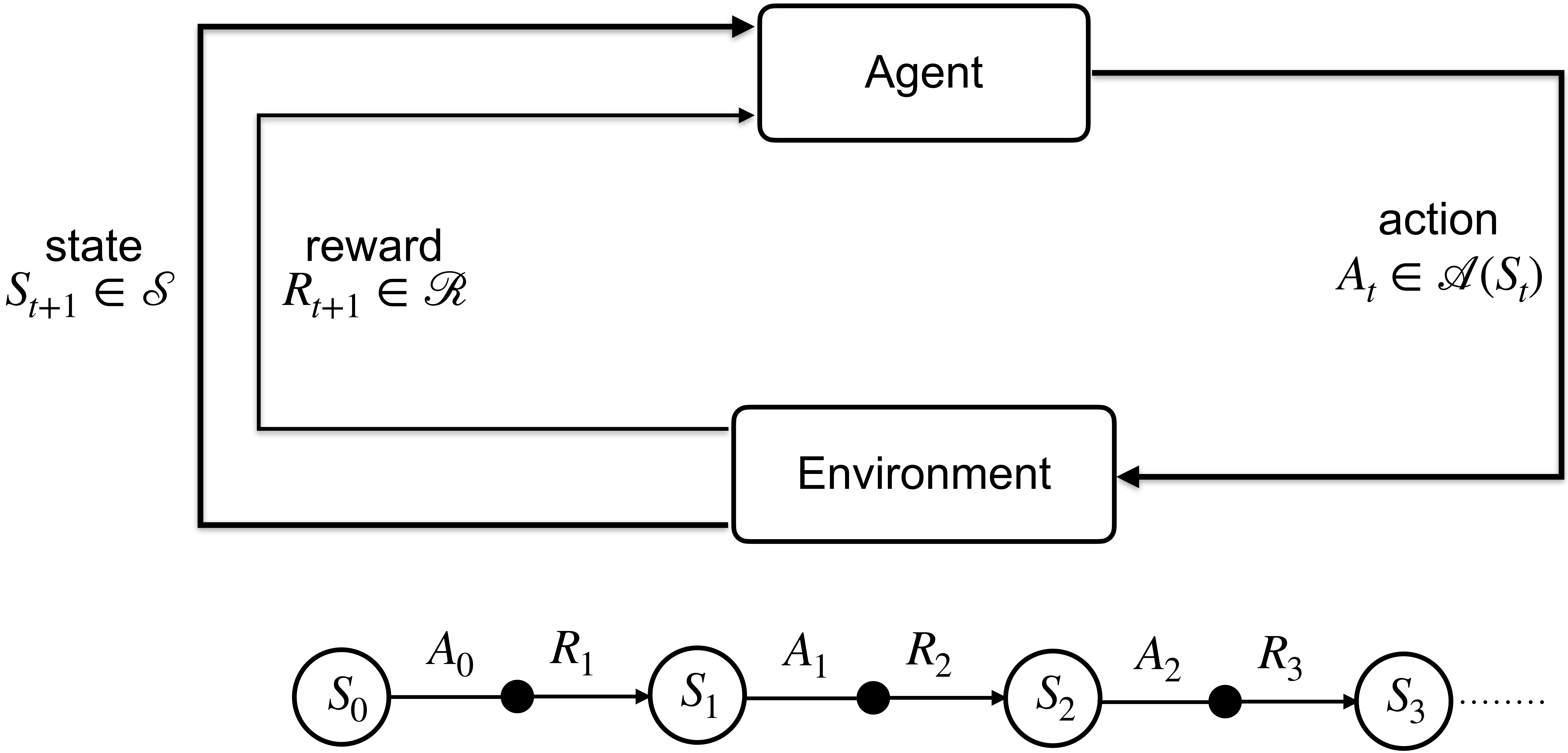
What is Reinforcement Learning?

- 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

The RL Interface



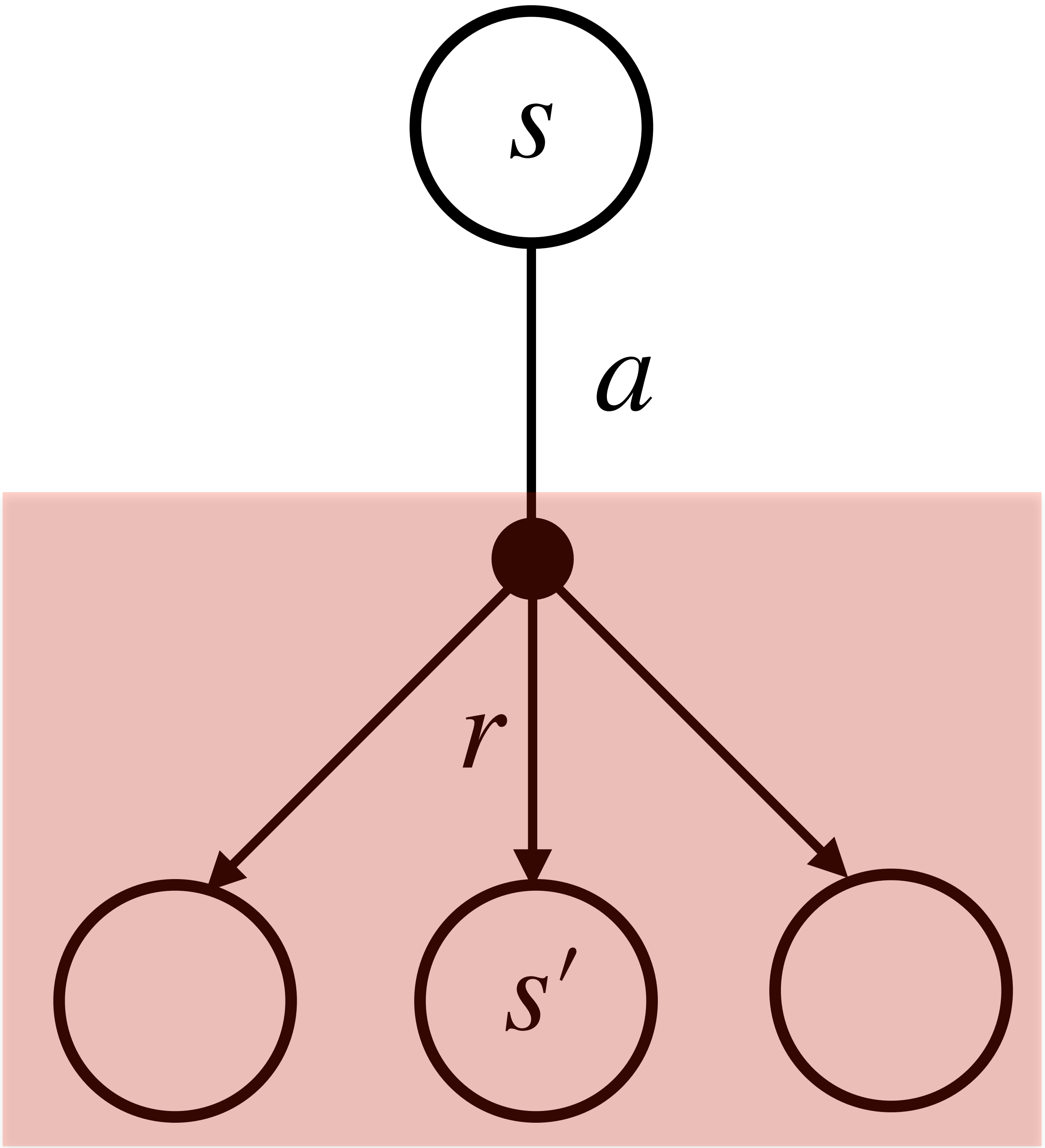
The interaction generates a stream of experience!



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

The dynamics of an MDP



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

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

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

Remembering earlier states would not improve predictions about the future

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

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

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

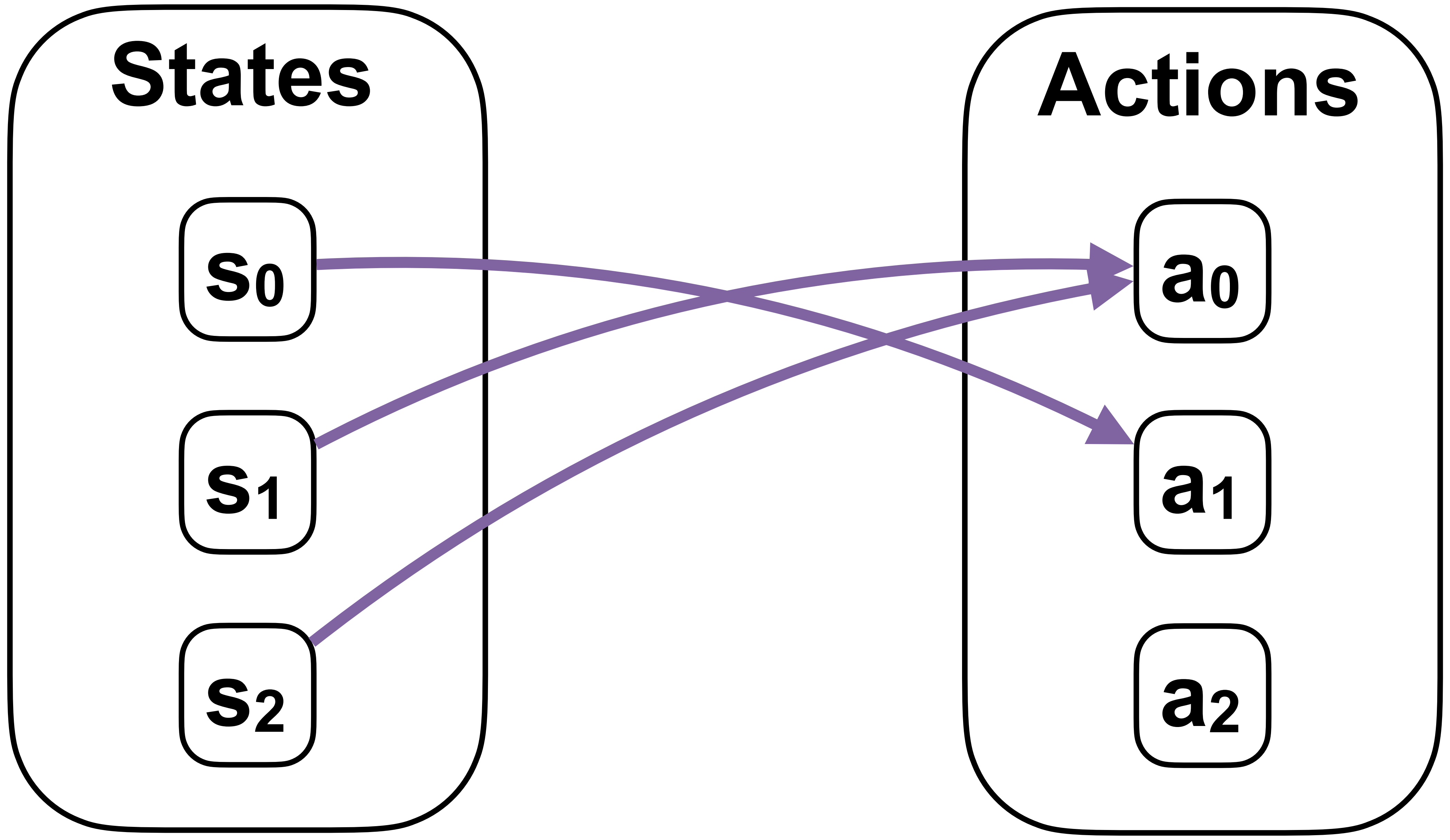
Finite as long as $0 \leq \gamma < 1$
and rewards are bounded

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

Policies

- Deterministic policy

$$\pi(s) = a$$



State	Action
s_0	a_1
s_1	a_0
s_2	a_0

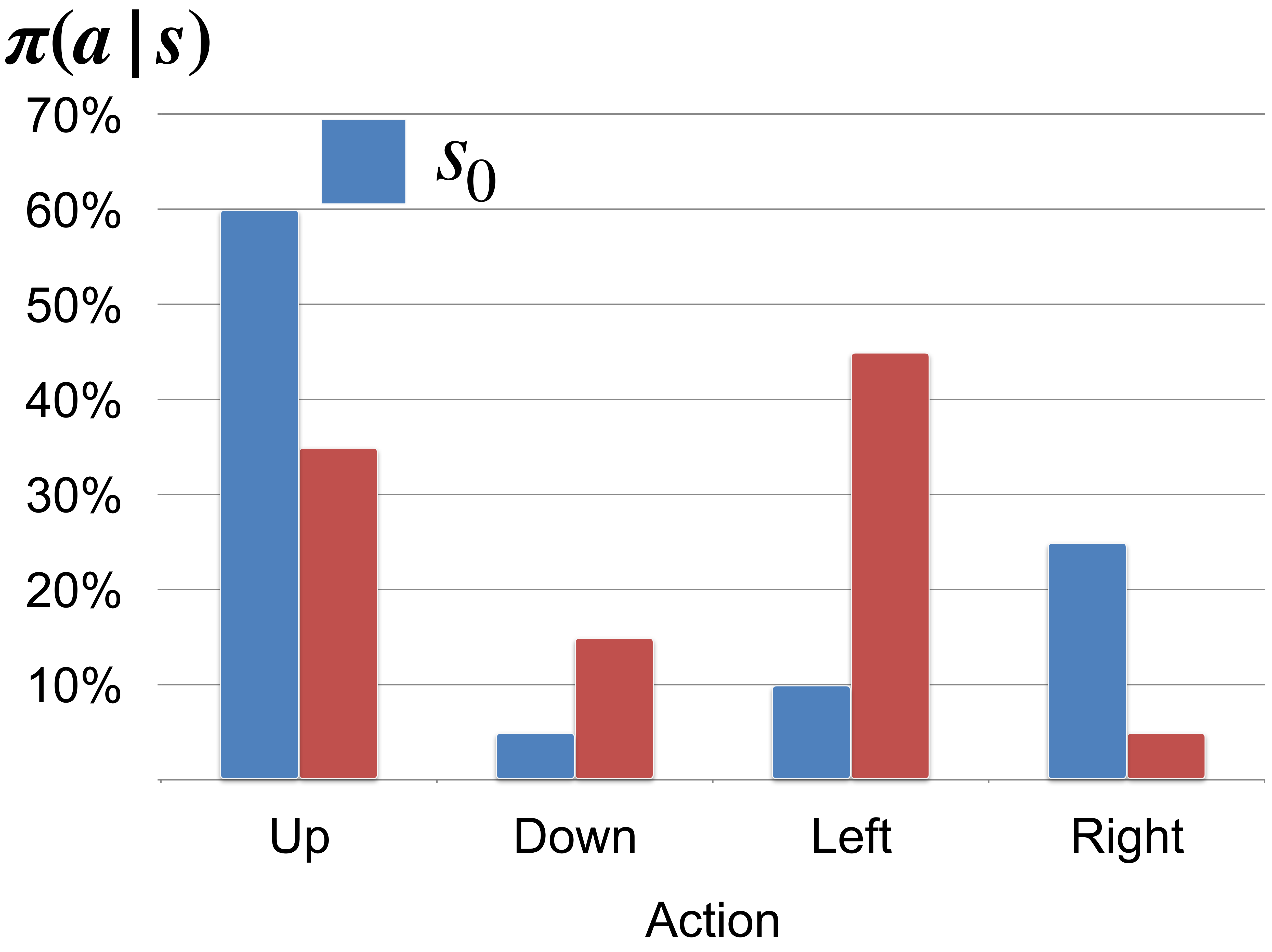
Policies

- Stochastic policy:

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

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

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

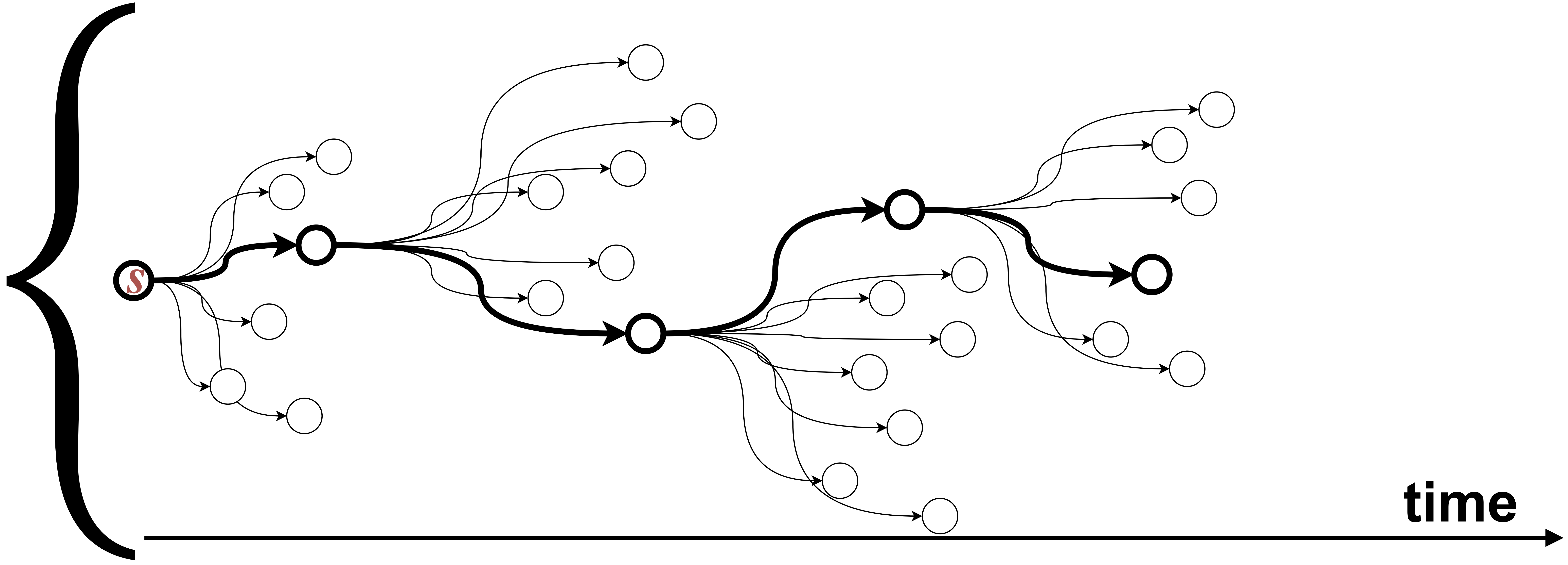


Action-value functions

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

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

$q_\pi(s,a)$



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

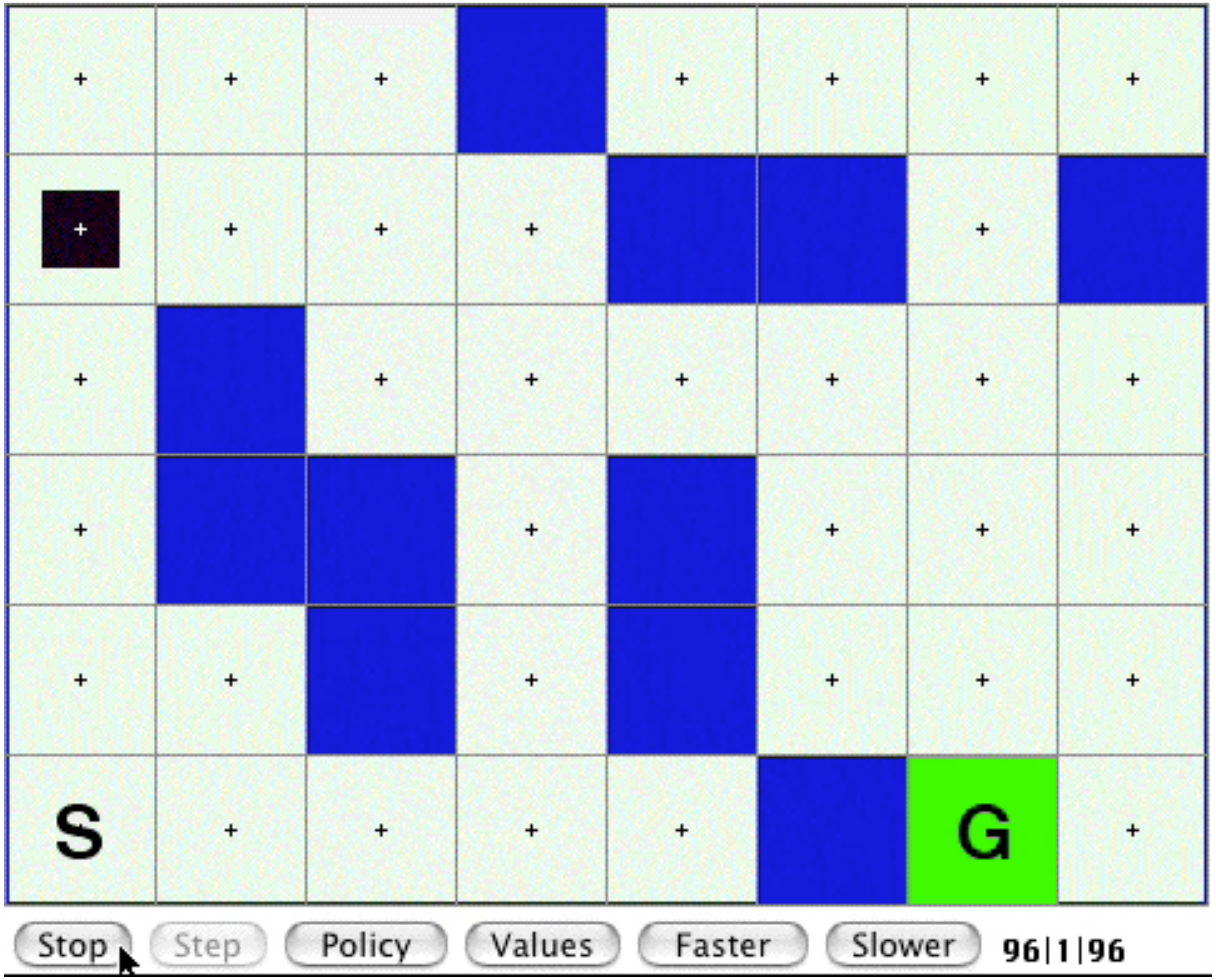
- Given the optimal value function, it is easy to act optimally:

$$\pi_\star(s) = \arg\max_a q_\star(s,a)$$ "greedification"

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

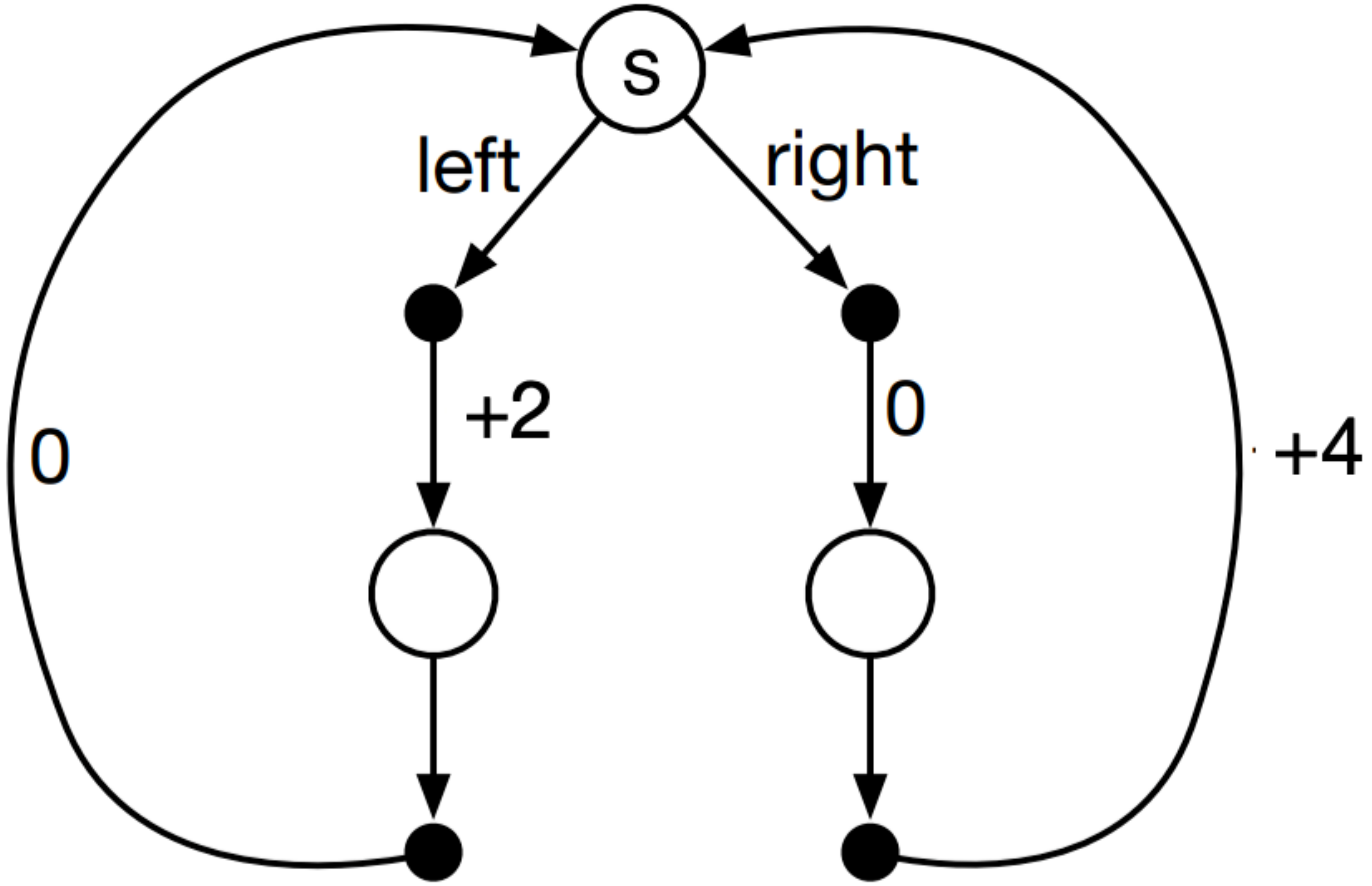
- There is always at least one deterministic optimal policy

GridWorld Example



Selt-test: C1M3

- Is the following policy valid for this MDP (i.e. does if fit our definition of a policy): Choose left for five steps, then right for five steps, then left for five steps, and so on? Explain your answer.



Difference between v and q

- “Why does the lower golf example (figure 3.3) which is supposed to be optimal have a -2 field over most of the green, where the above example with the putter has that area marked as only -1? Isn't q*() supposed to be optimal? There should be no areas where q*() has a worse result than v putt, right?”

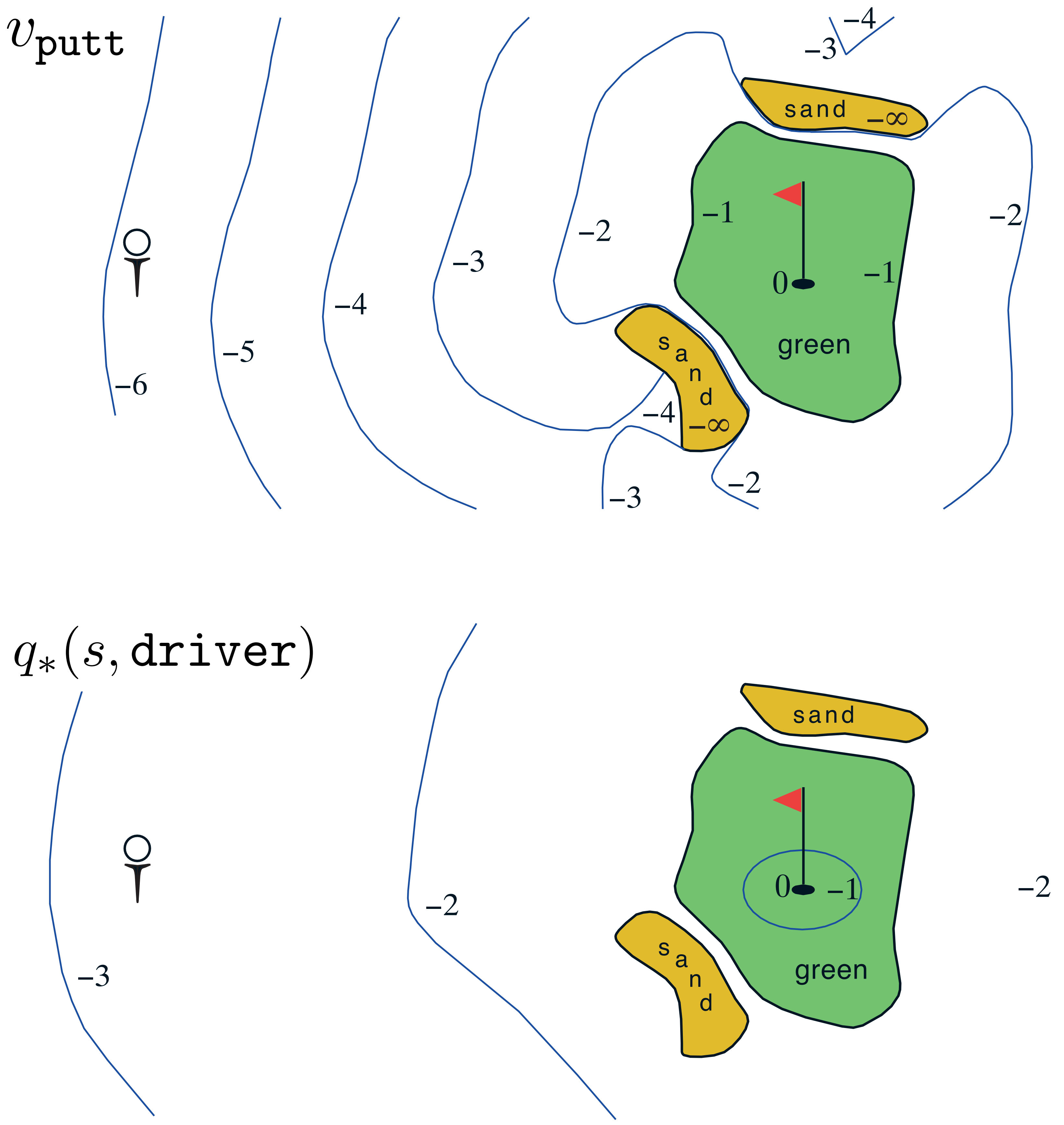
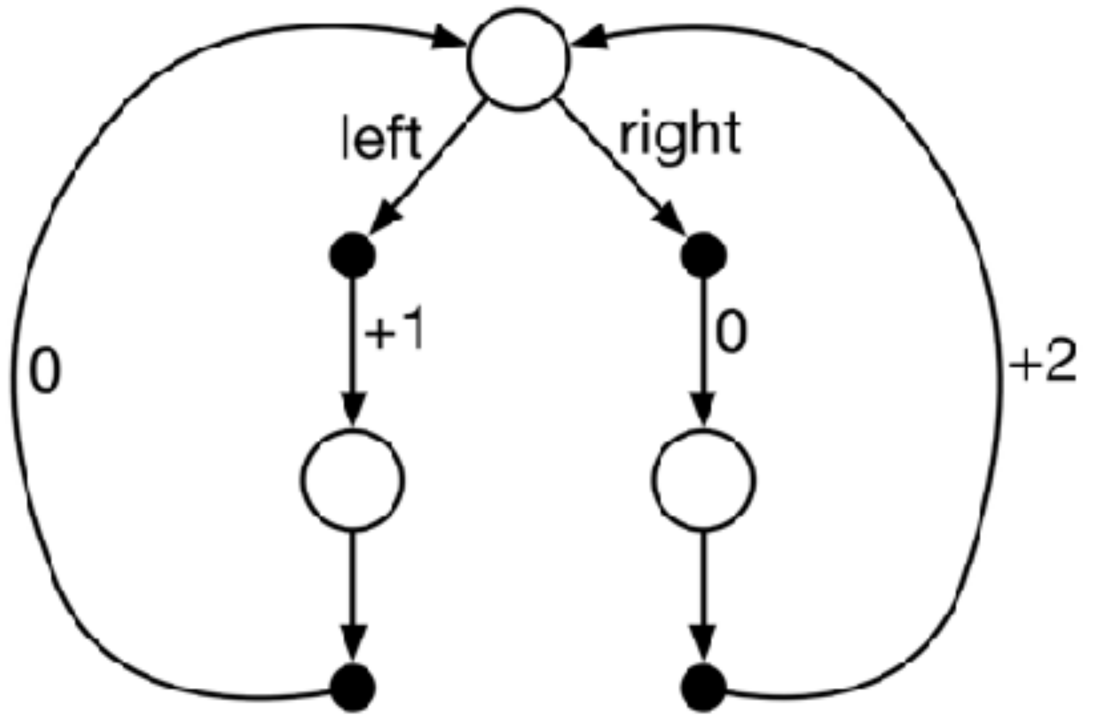


Figure 3.3: A golf example: the state-value function for putting (upper) and the optimal action-value function for using the driver (lower). ■

We can only directly solve small MDPs



2 Deterministic Policies

Brute-Force Search

A General MDP

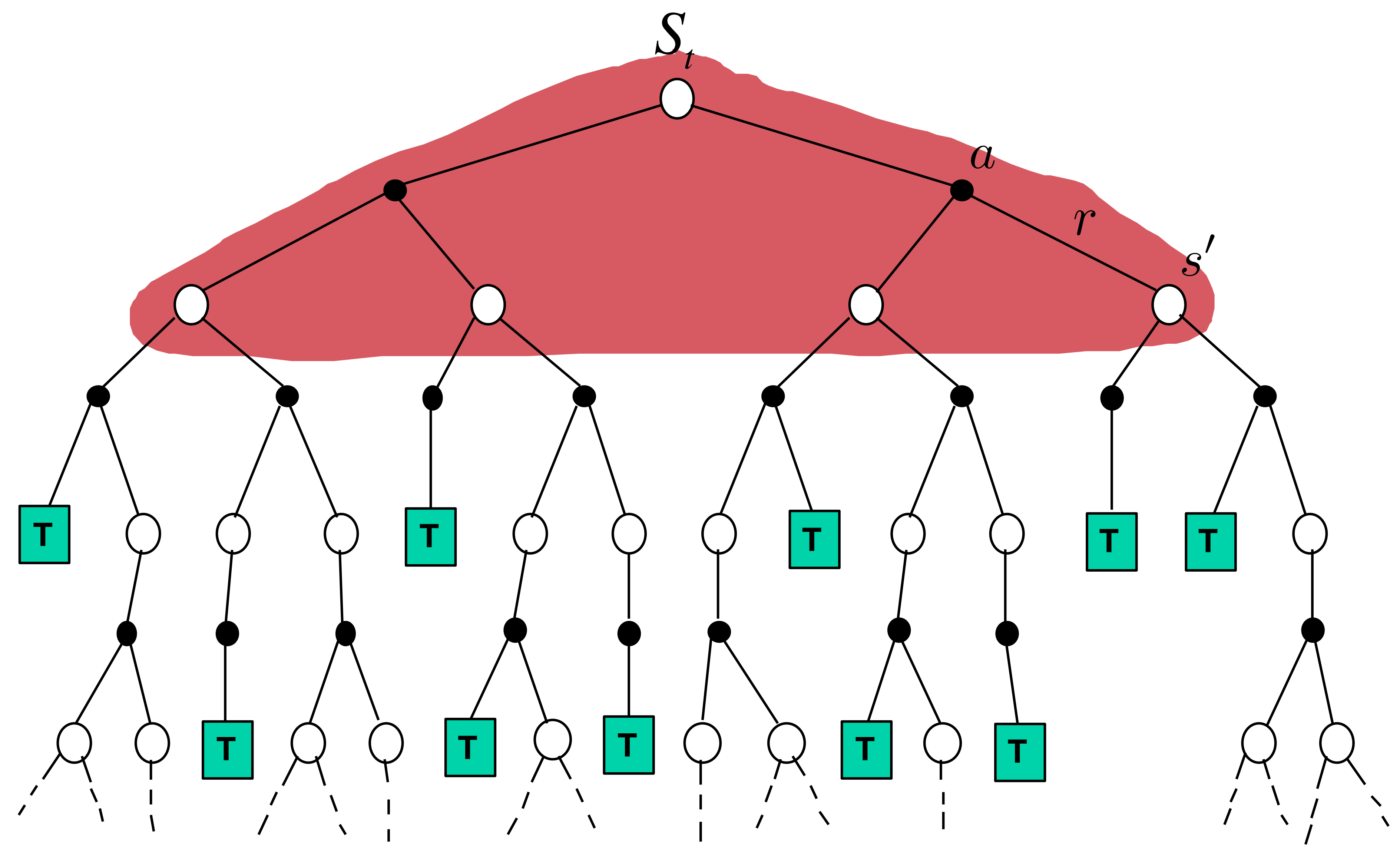
$|\mathscr{A}|^{|\mathscr{S}|}$ Deterministic Policies

Brute-Force Search

RL methods based on Bellman Equations

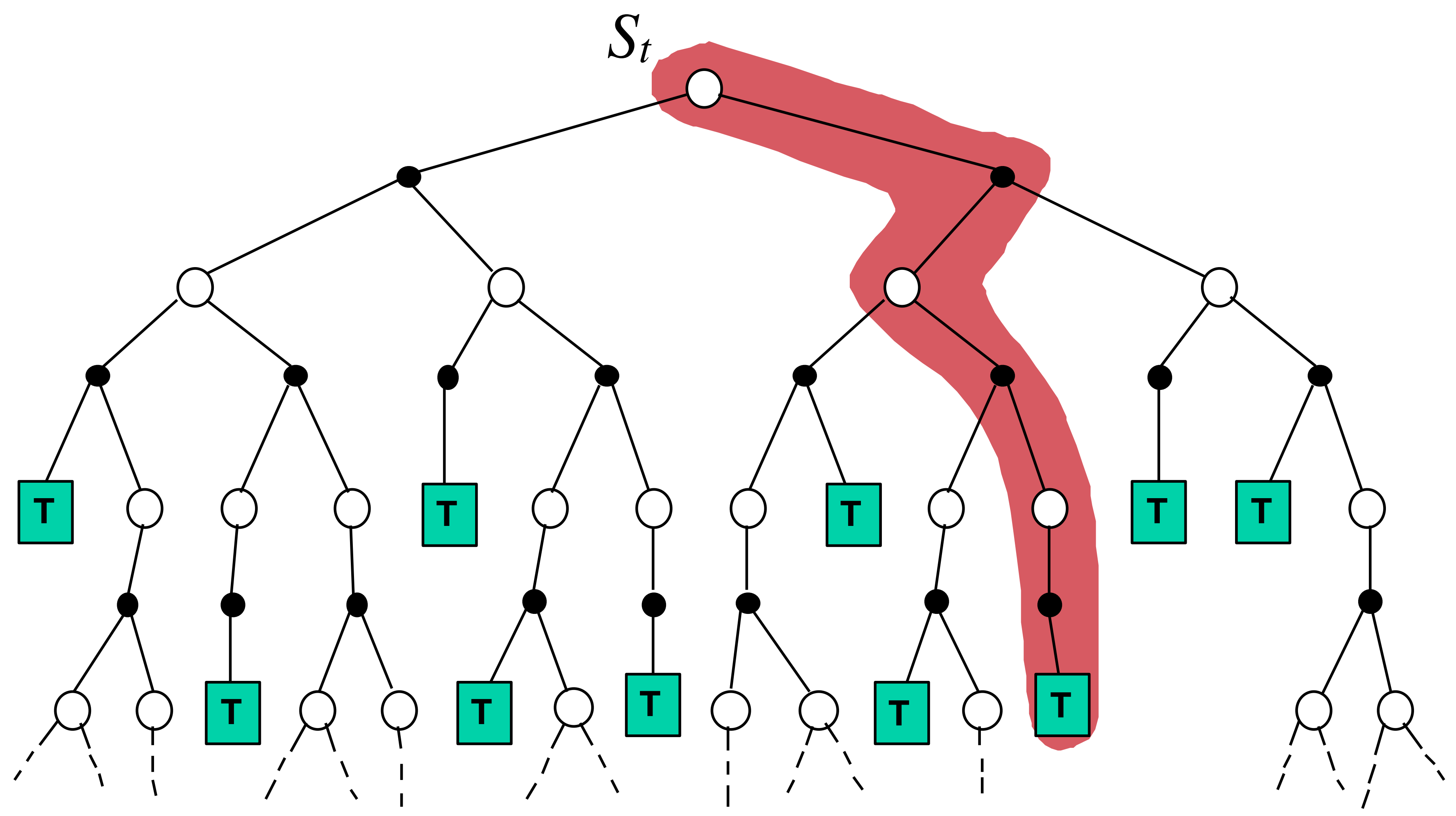
Dynamic programming

$$V(S_t) \leftarrow E_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right] = \sum_a \pi(a|S_t) \sum_{s',r} p(s',r|S_t,a)[r + \gamma V(s')]$$



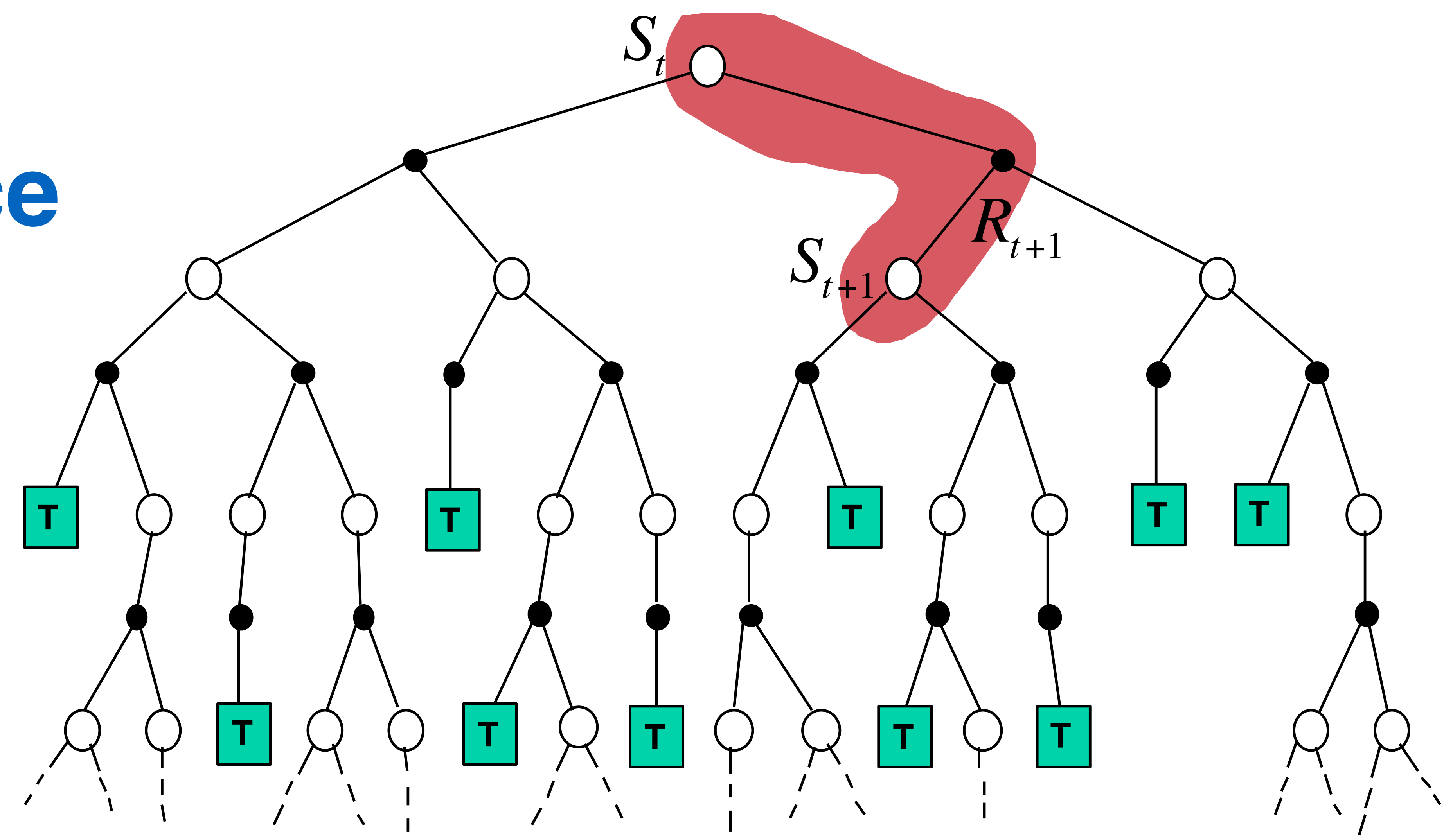
Simple Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha\left[G_t - V(S_t)\right]$$



$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

Temporal Difference Learning



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'

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

$S \leftarrow S'$
until S is terminal

Self-test

- What is the target policy for Q-learning?
- What can the behavior policy be?

Self-test

- What is the target policy for Q-learning?
  - Answer: Q-learning **learns about** the greedy policy (which eventually becomes π^*), while **following a different policy** (e.g., ε-greedy). That is off-policy, but there are no importance sampling corrections!
- What can the behavior policy be?

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

- 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

- 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)

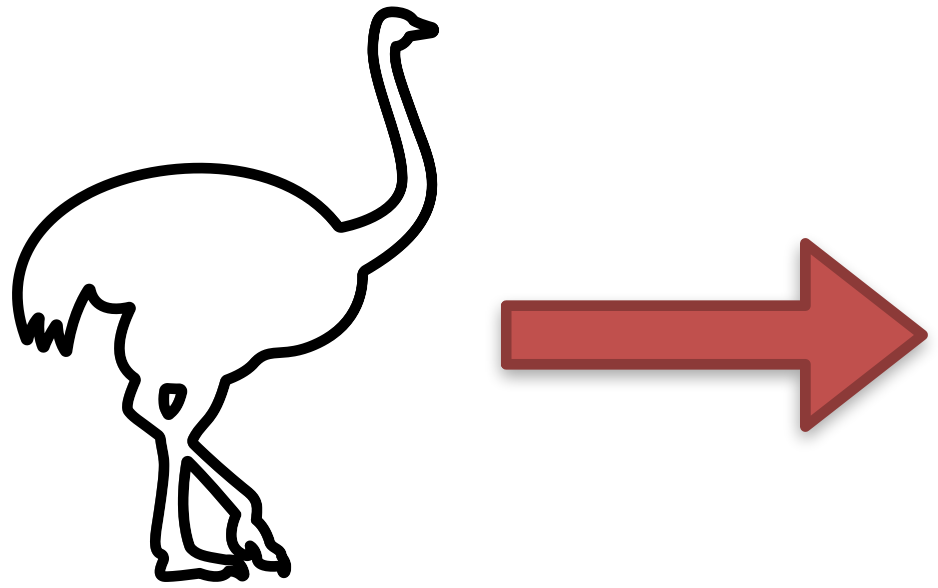
How does Q-learning handle exploration?

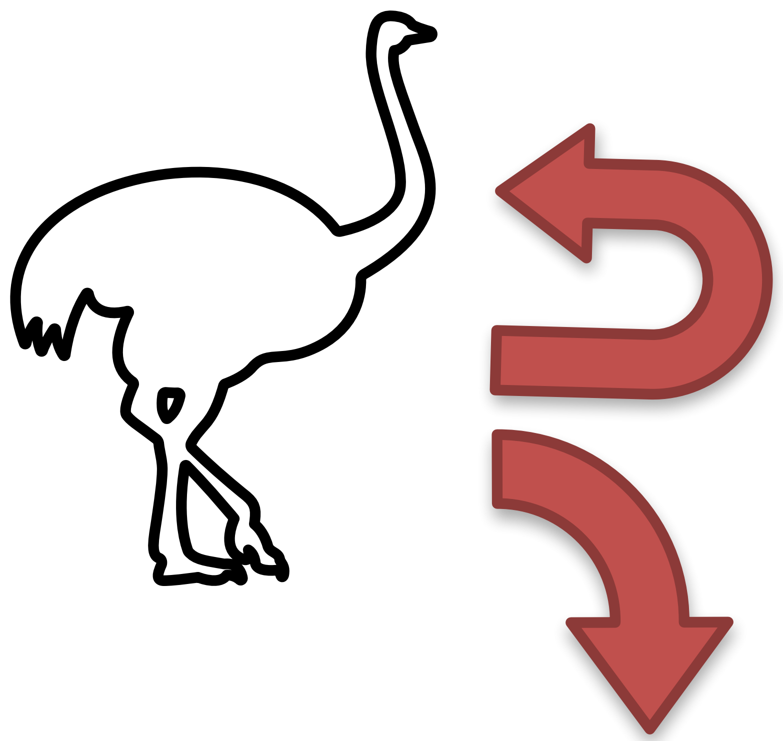
Choose actions in any way, perhaps based on Q, such that all actions are taken in all states (infinitely often in the limit)

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

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

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





$$A_t = \underset{a}{\operatorname{argmax}}\ Q(S_t, a) \qquad A_t = \text{Random action}$$

Self-test: Exploration in MC

- Why did we talk about exploring starts in MC when estimating action-values, but not when estimating state values?
- Can we use state-values for control in MC, like we did in DP?

Terminology Review

- TD methods we have learned about are **tabular, one-step, model-free** learning algorithms
- **Tabular:** we store the value function in a table. One entry in the table per value, so each value is stored independently of the others. We are implicitly assuming the state-space ($\mathcal{S}$) is small
- **One-step**: we update a single state or state-action value on each time-step. Only the value of Q(S,A) from S -- A --->S',R. We never update more than one value per learning step
- **Model-free**: we don't assume access to or make use of a model of the world. All learning is driven by sample experience. Data generated by the agent interacting with the environment

Tabular Dyna-Q

Terminology Review

- **Model:** a model of the environment. Anything that can predict how the environment will respond to the agent's actions: M(S,A) --->S',R

- **Planning**: the computational process that takes the model as input and produces or improves the policy

- **Sample Model**: a model that can produce a possible next state and reward, in agreement with the underlying transition probabilities of the world. We need not store all the probabilities to do this (think about epsilon-greedy)

- **Simulate:** sample a transition from the model. Given an S and A, ask the model for a possible next state S' and reward R

- **Simulated Experience:** samples generated by a sample model. Like dreaming or imagining things that could happen

- **Real Experience:** the states, actions, and rewards that are produced when an agent interacts with the real world.

- **Search Control:** the computational process that selects the state and action in the planning loop

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**

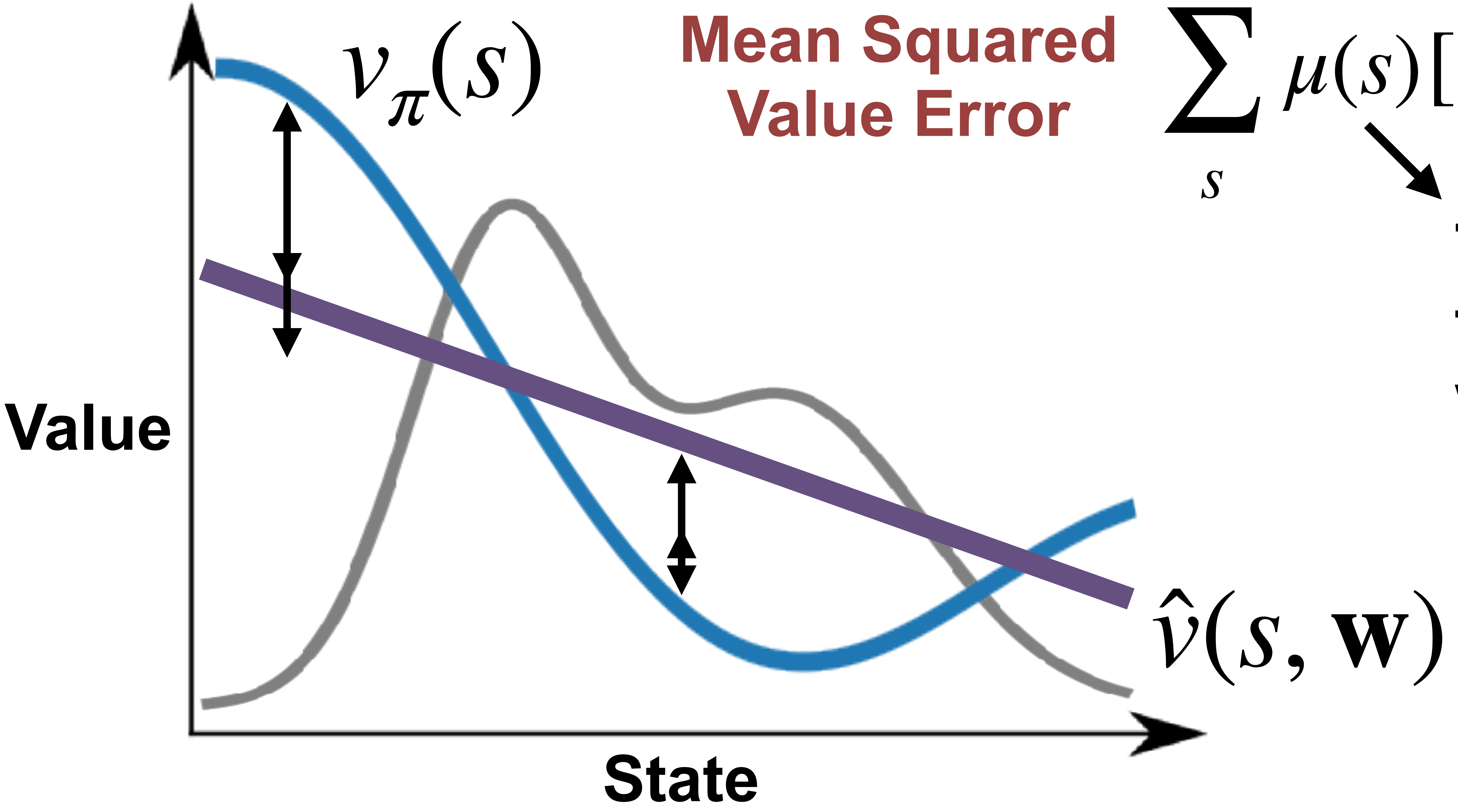
Function approximation

- 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

The Mean Squared Value Error Objective



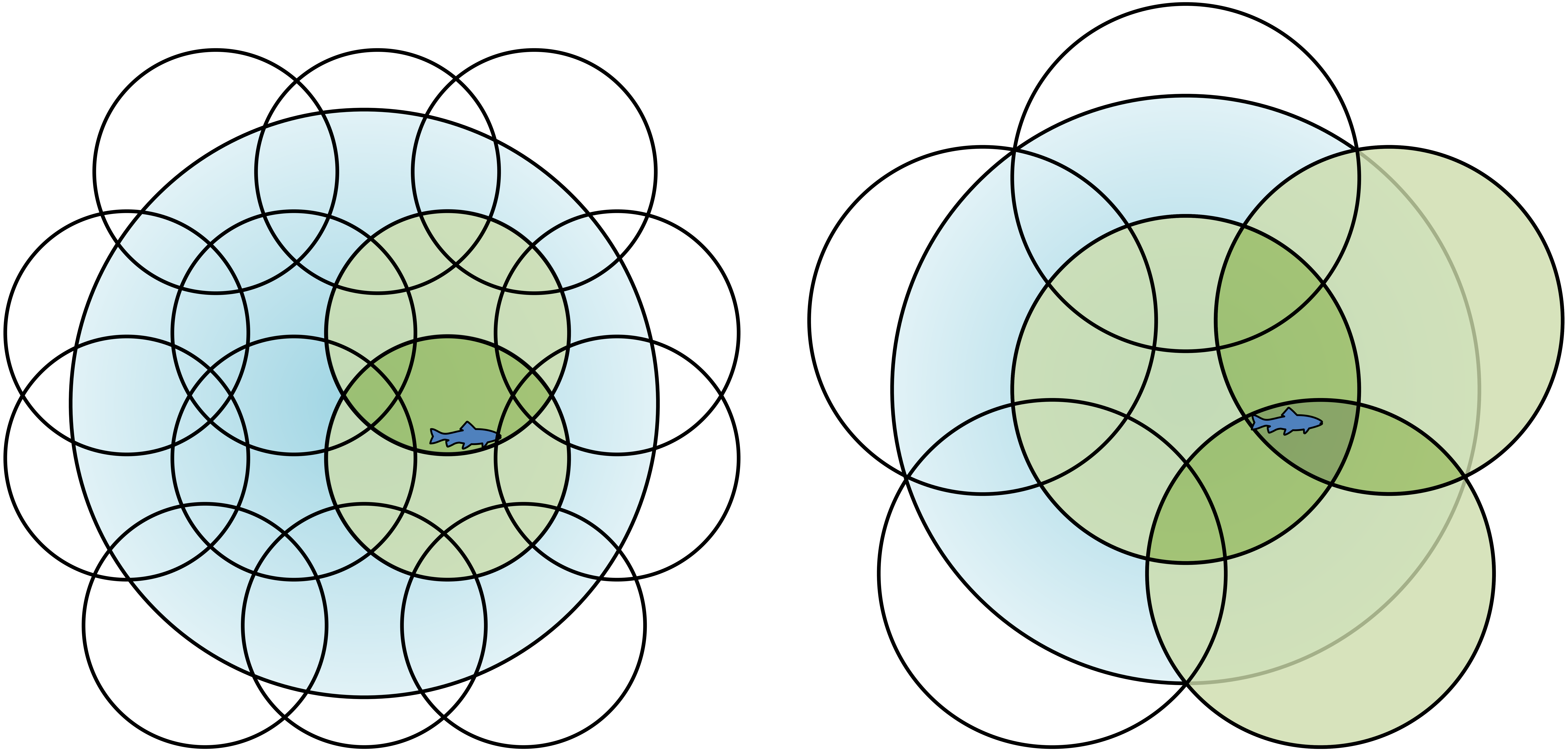
Mean Squared Value Error

$$\sum_{s} \mu(s)[v_\pi(s) - \hat{v}(s, \mathbf{w})]^2$$

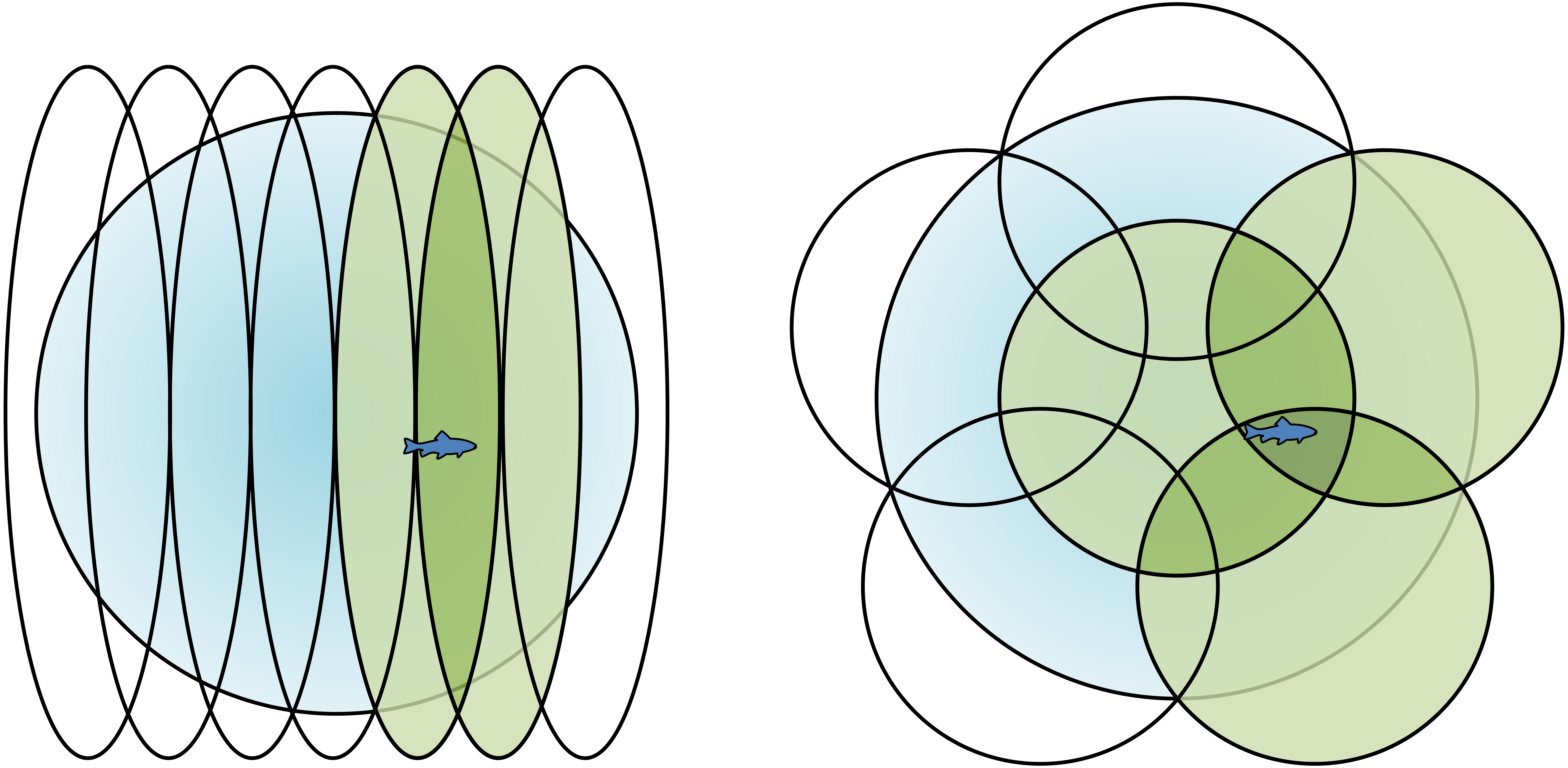
The fraction of time we spend in S when following policy π

Question: Why didn't we use the Value Error in the tabular setting?

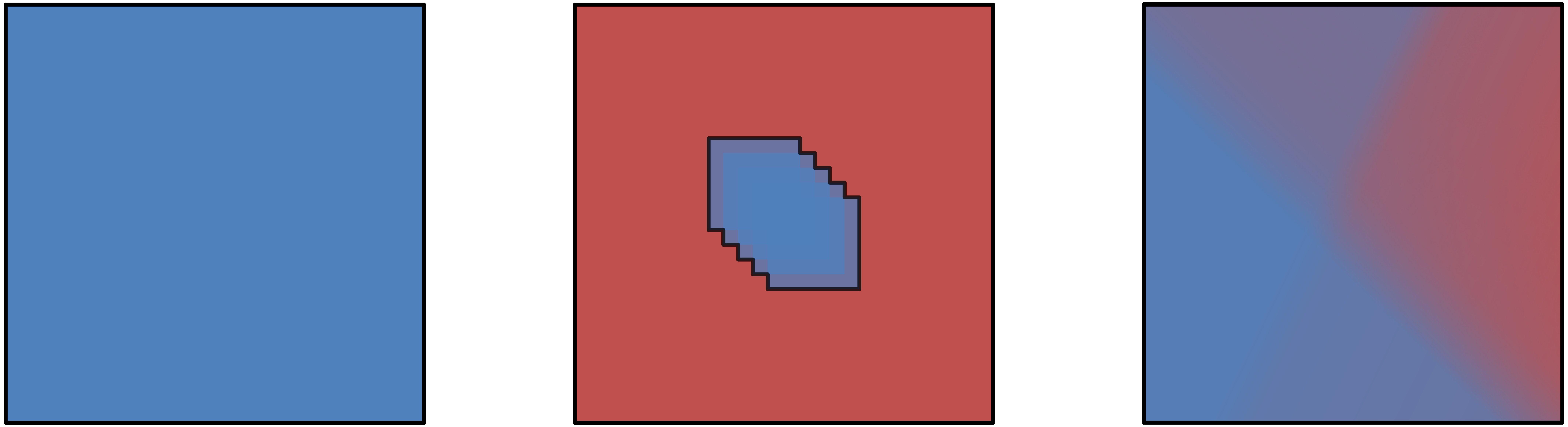
Broadness of generalization



Direction of generalization

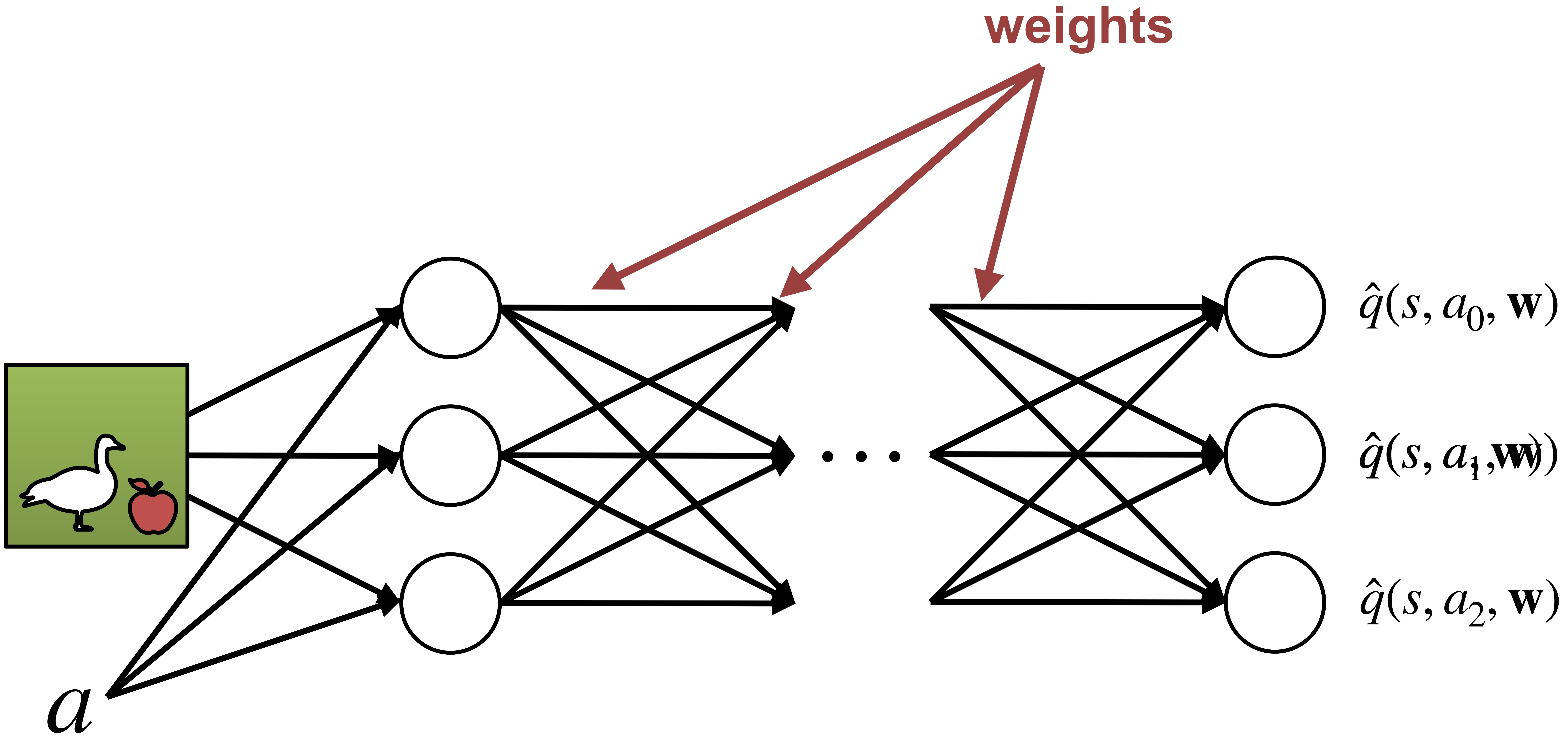


How Optimism Interacts with Generalization



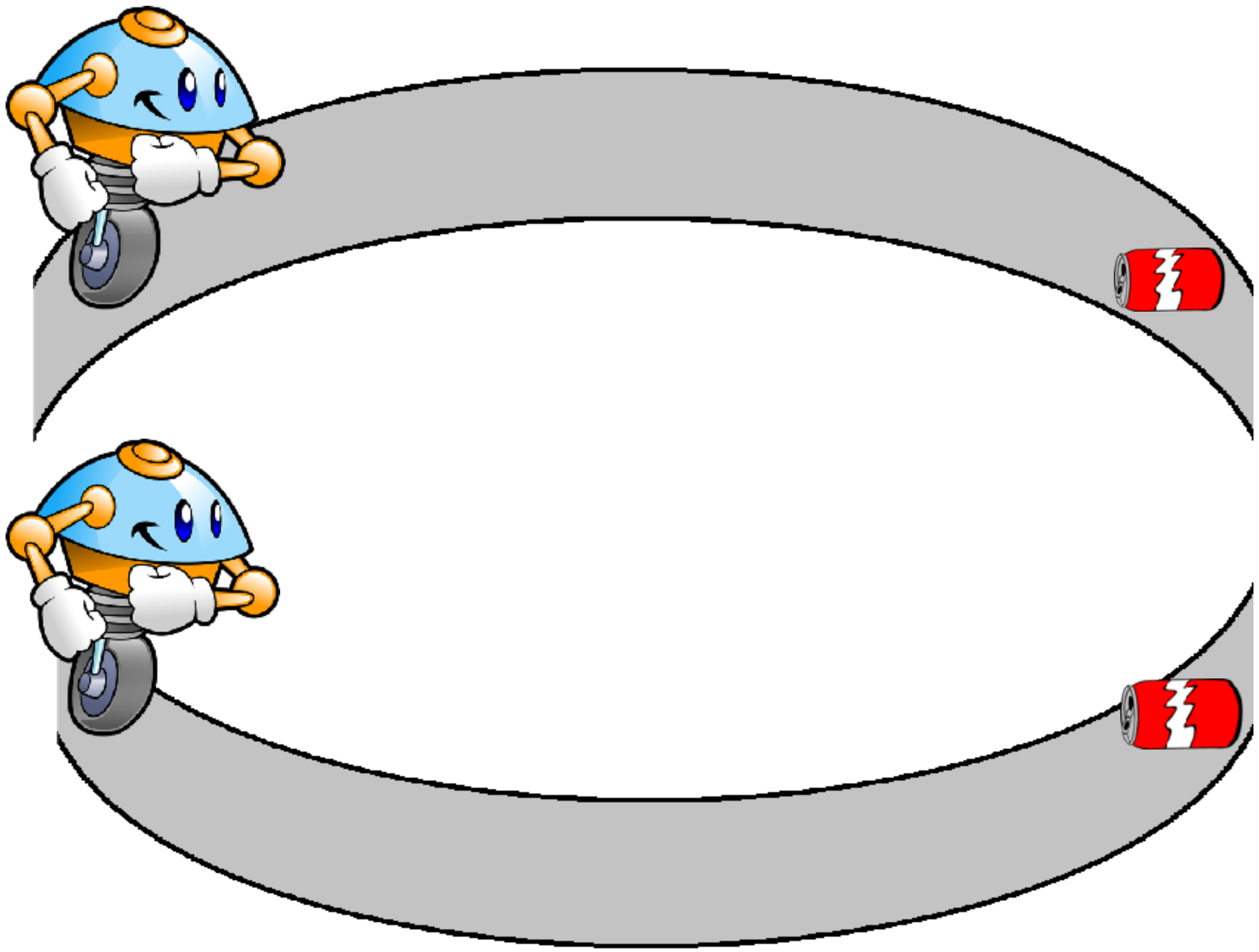
Single feature | Tile coding | Neural network

Approximating q_π with an NN

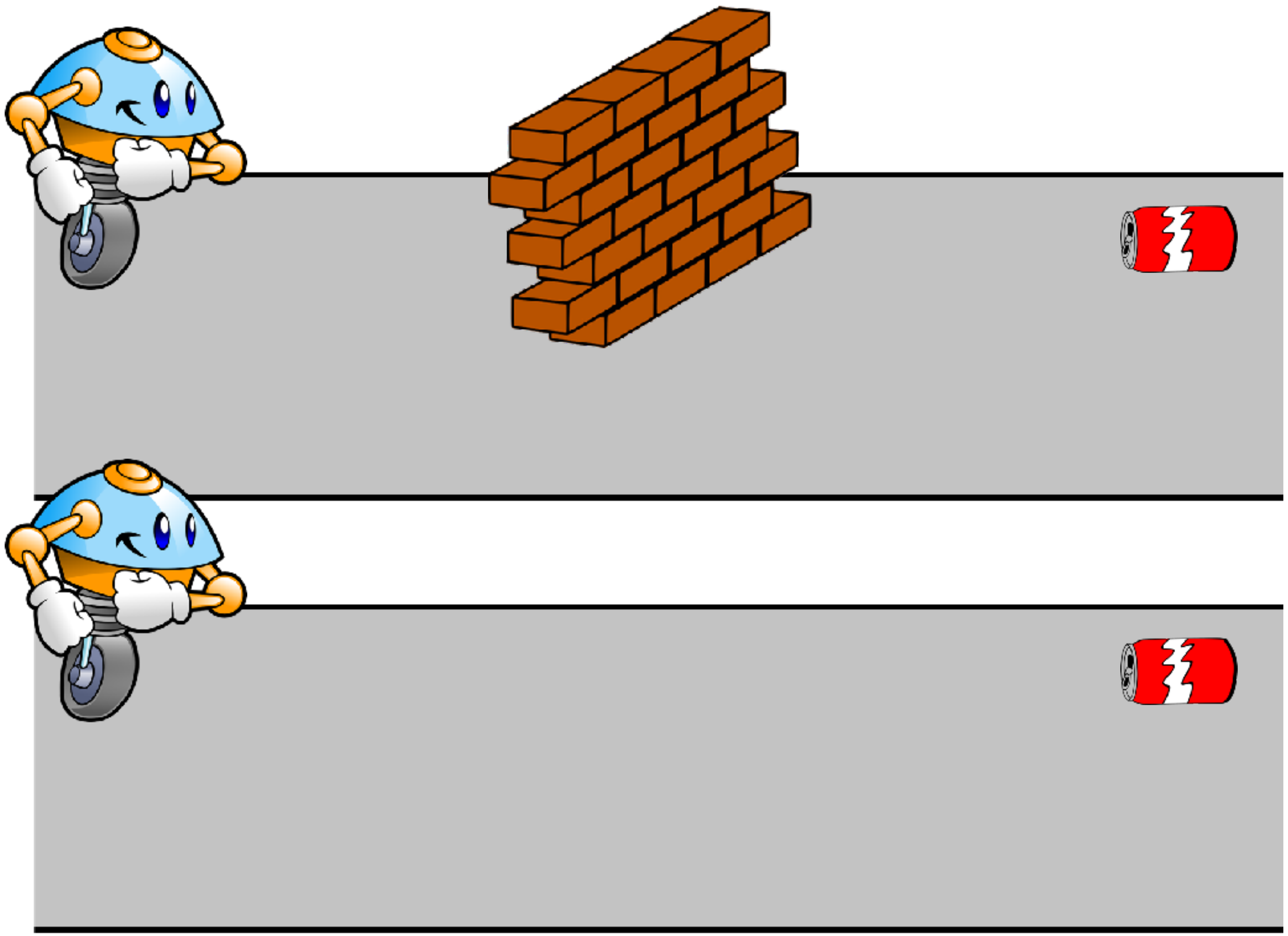


Generalization: Updates to One State Affect the Value of Other States

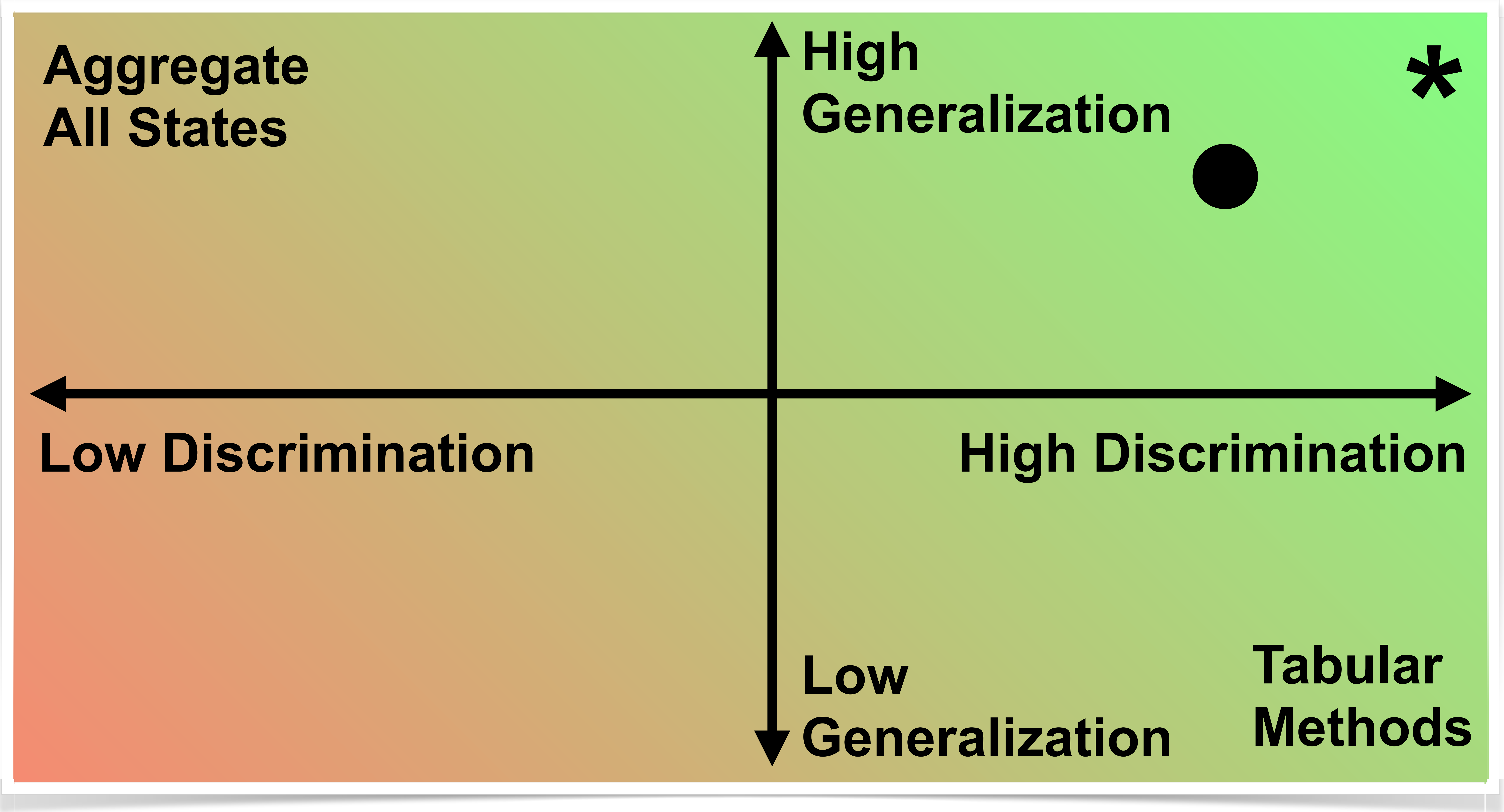
State	Action	Q
s_1	F	-3 / -4
s_2	F	-2 / -4
s_3	F	2
s_4	F	-10 / -7
s_5	F	4



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



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

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

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

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

C3M4: Policy Gradient

- I will not test you on average reward, nor on policy gradient
- I am skipping this in the review