How do we maximize the value of future actions?

Readings for today

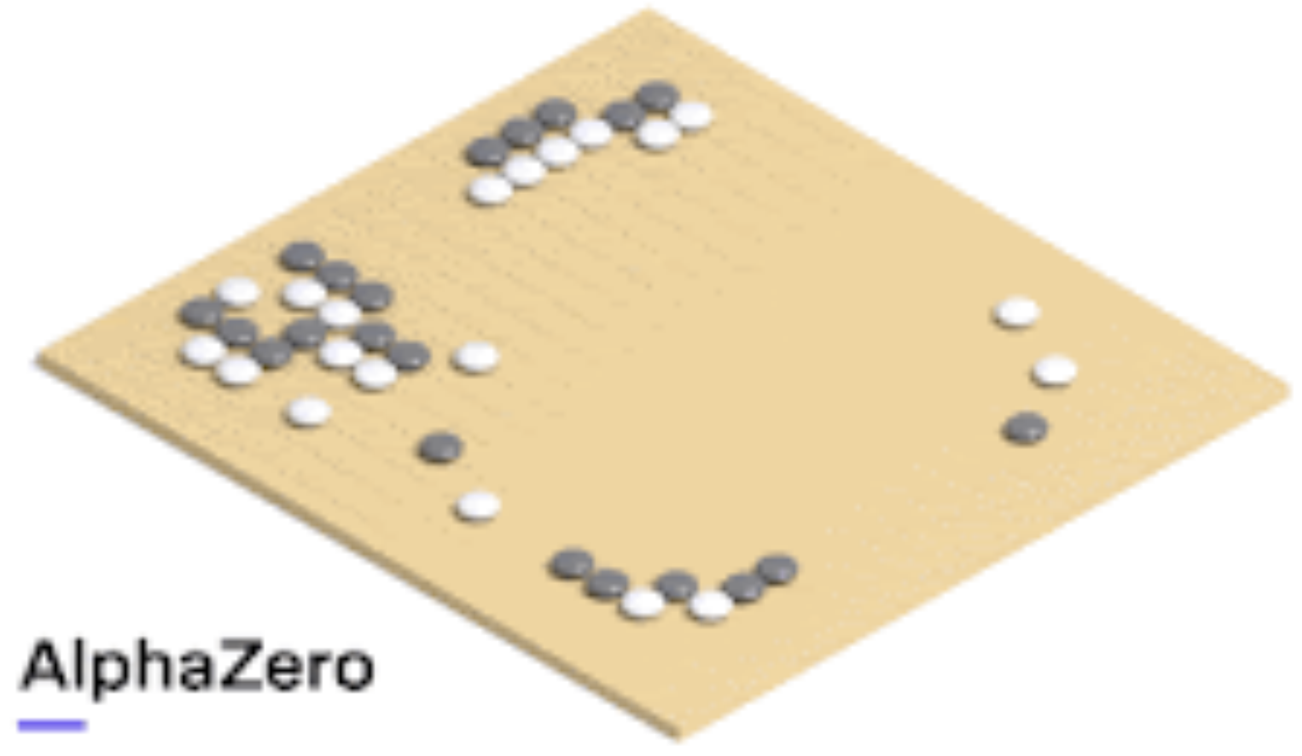
- Sutton, R. S., & Barto, A. G. (1998). Chapter 1: Introduction. In Reinforcement learning: An introduction (2nd edition). MIT press.
- Sutton, R. S., & Barto, A. G. (1998). Chapter 2: Multi-armed bandits. In Reinforcement learning: An introduction (2nd edition). MIT press.

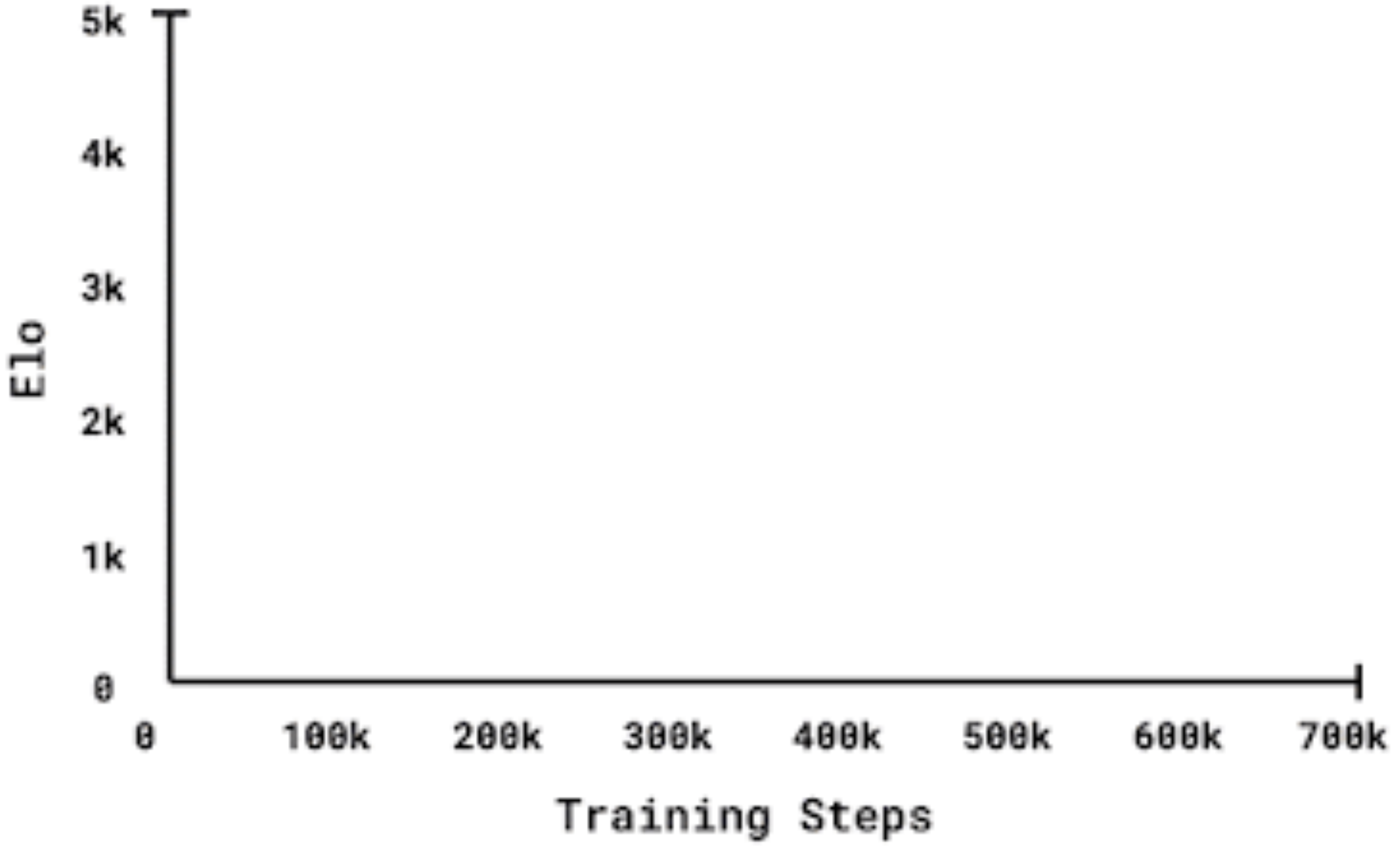
Topics

- Reinforcement learning
- Q-learning and multi-arm bandits

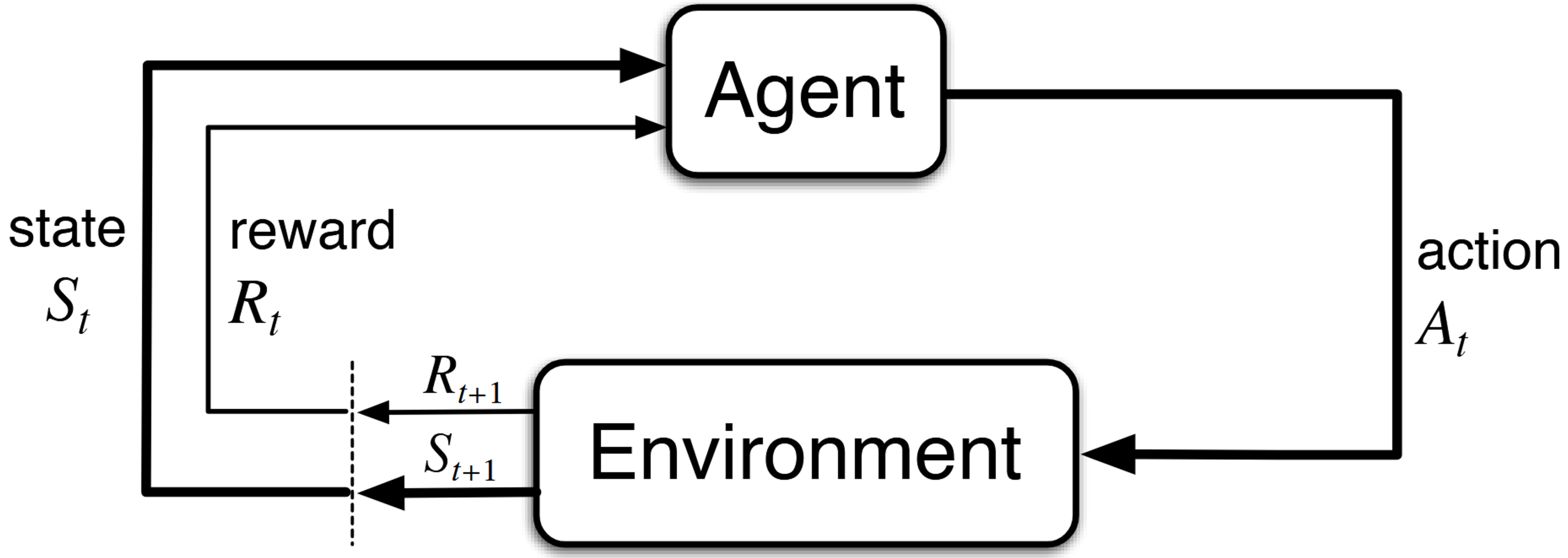
Reinforcement Learning (RL)

RL is useful conceptually and practically





A general framework for learning from rewards

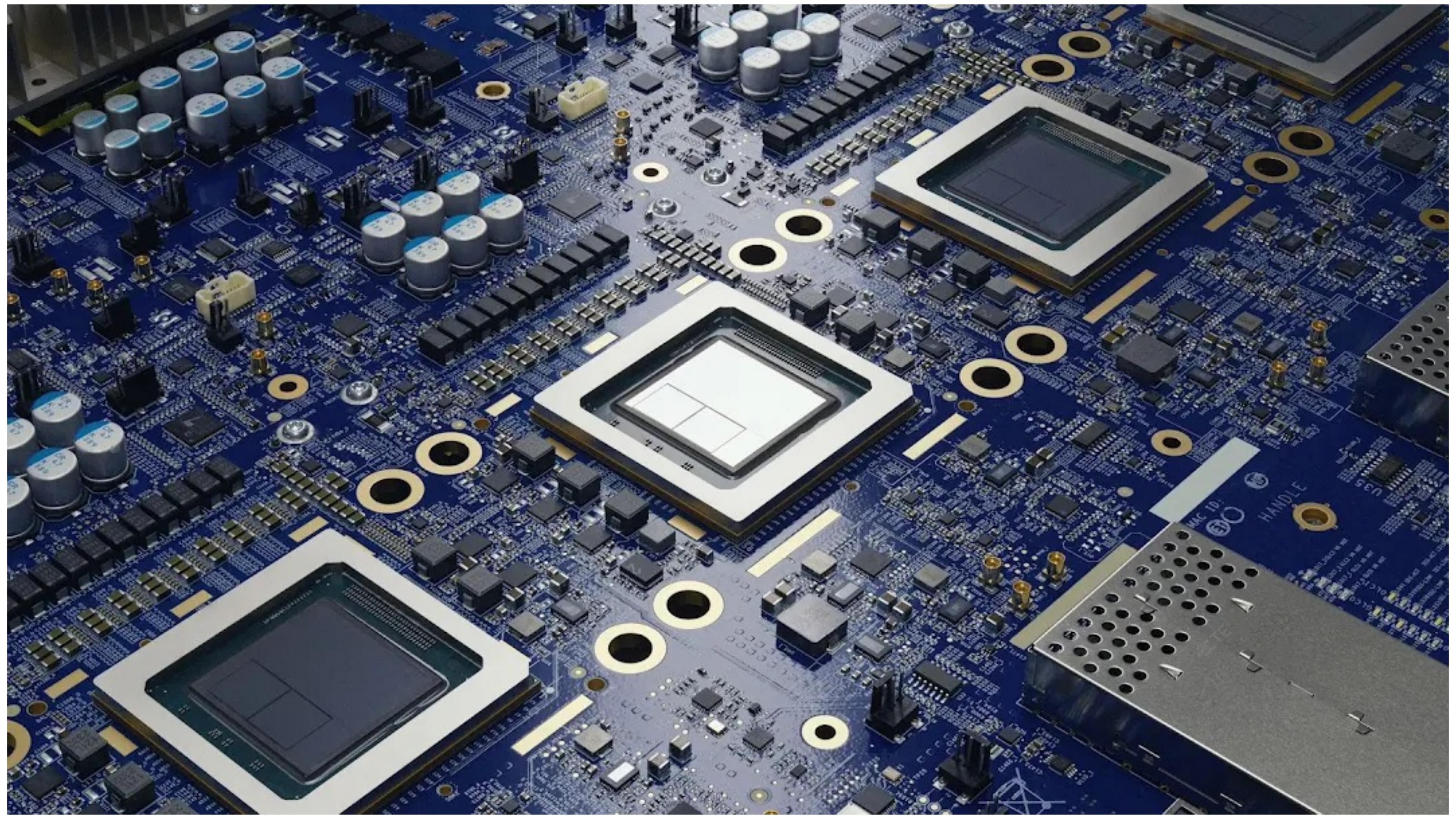


- **Agent** exists in an **environment** described by a **state** S_t
- Chooses an **action** A_t according to a **policy** π
- Receives **reward** R_{t+1} and environment evolves to state S_{t+1}
- Attempts to learn a policy which maximizes **sum of rewards** G_t

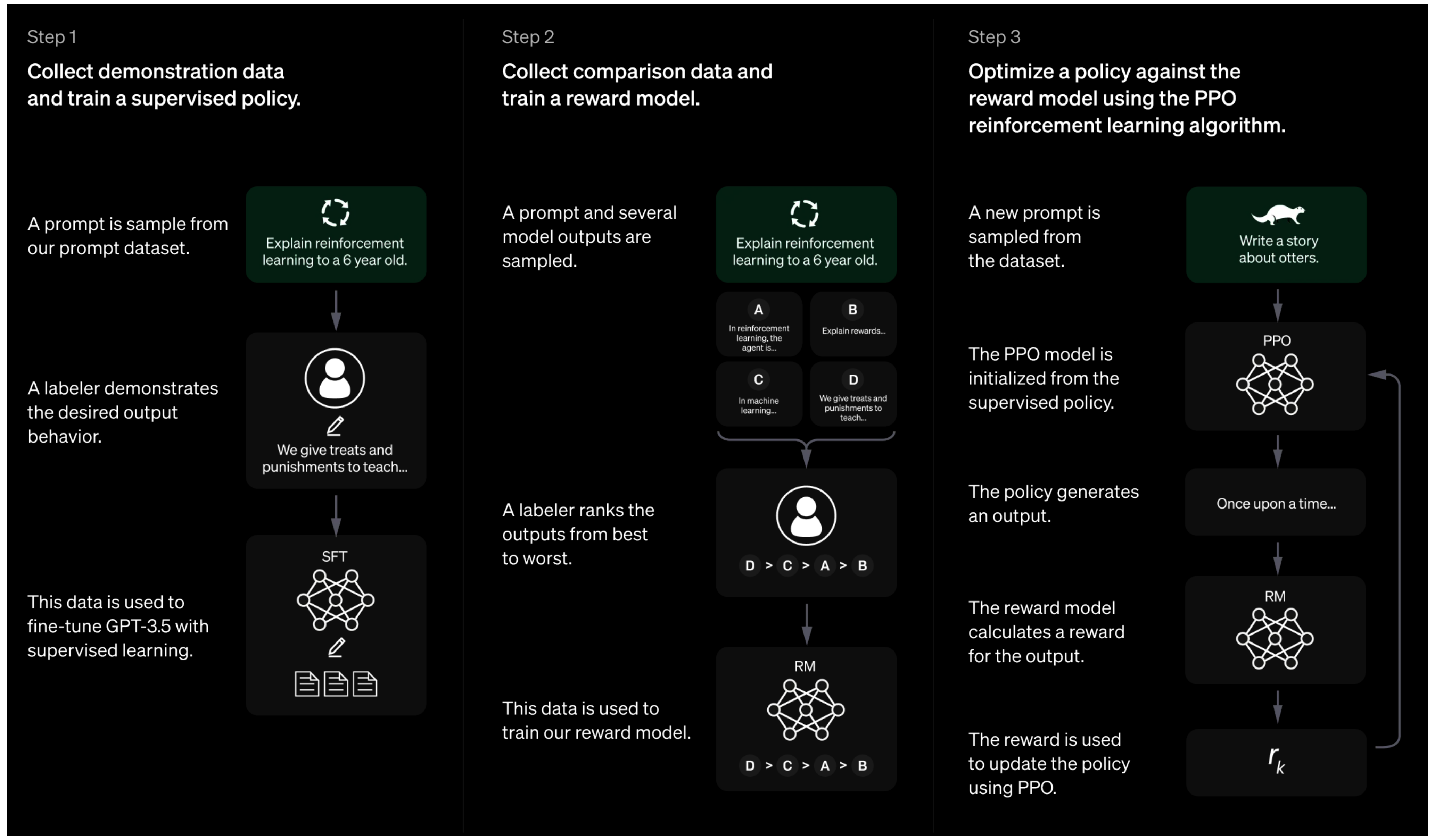
$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

RL in practice

Chip design



ChatGPT

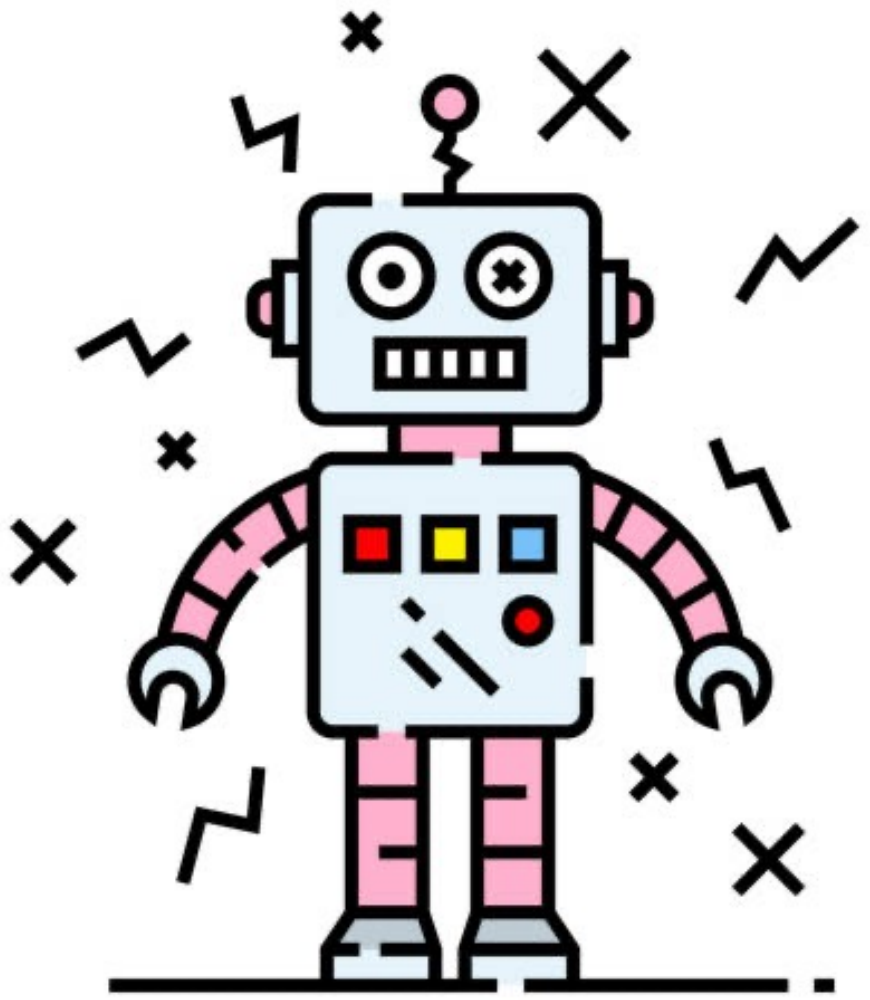


Noisy actions and rewards in the real world

- The world is noisy
- Actions do not always produce the same outcomes (nor the same rewards)

State transition probabilities

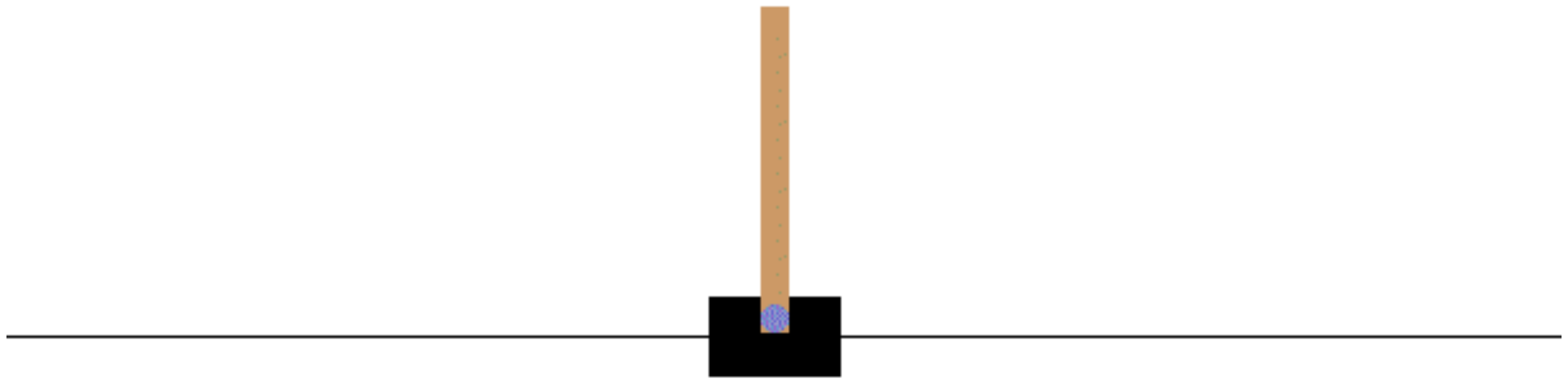
$$p(s', r \,|\, s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

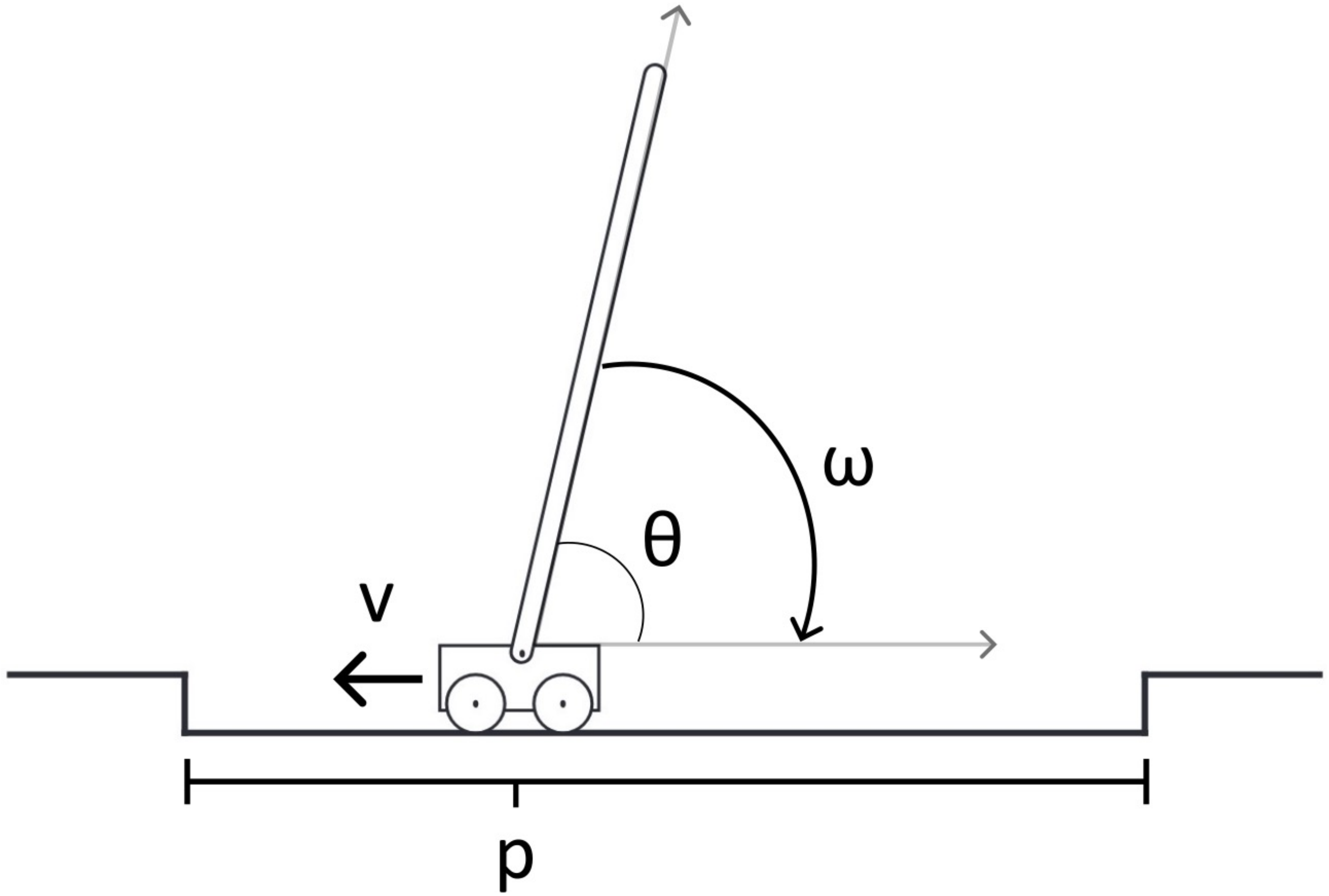


Wu *et al*. **TidyBot: Personalized Robot Assistance with Large Language Models**. *arXiv*. (2023)

The CartPole environment

- Goal: balance a pole on a cart
- $S_t = (p_t, v_t, \theta_t, \omega_t)$
- Possible actions = (push left, push right)
- $R_t = \sin \theta_t$ (1 when vertical, 0 when horizontal)
- Environment resets if pole falls completely





Discounted sum of rewards

$$0 < \gamma < 1 \qquad G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Value functions

- How can we tell which states are preferable?

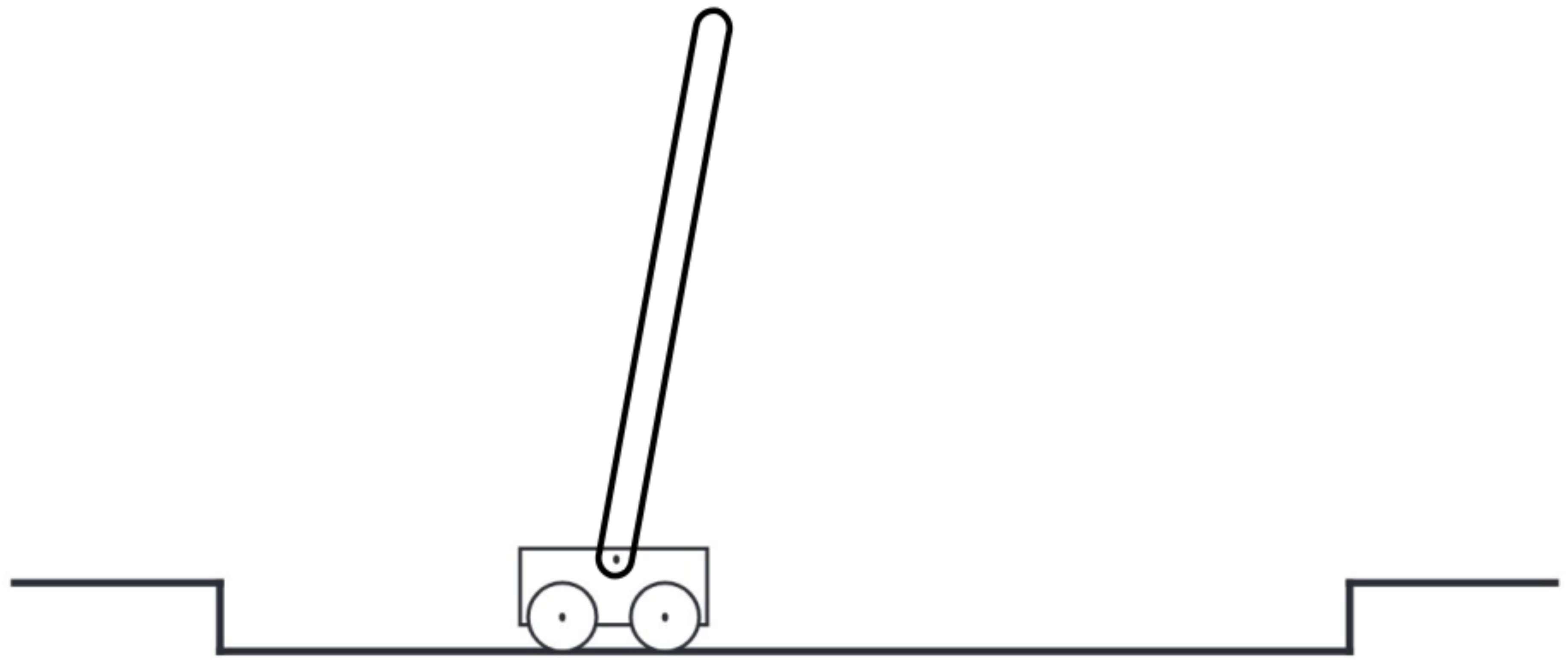
Value functions

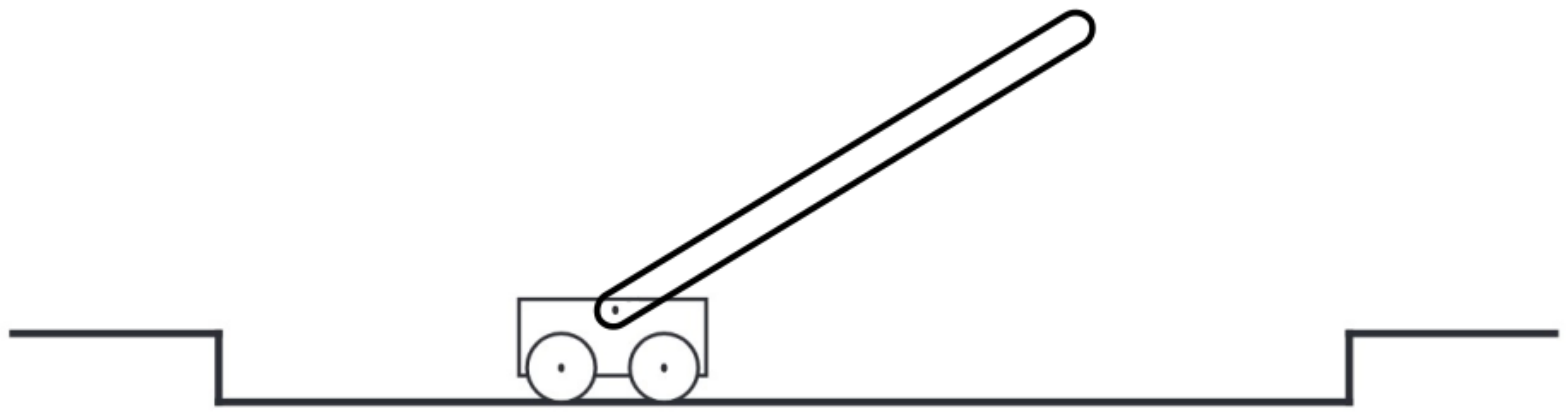
Tells us the total discounted reward we expect to receive starting in a state s under policy π

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

$$v_*(s) \doteq \max_\pi v_\pi(s)$$

Which state has a higher value?





State-action value functions

- Agent cannot directly control state
- Can only choose actions which may produce more preferable state

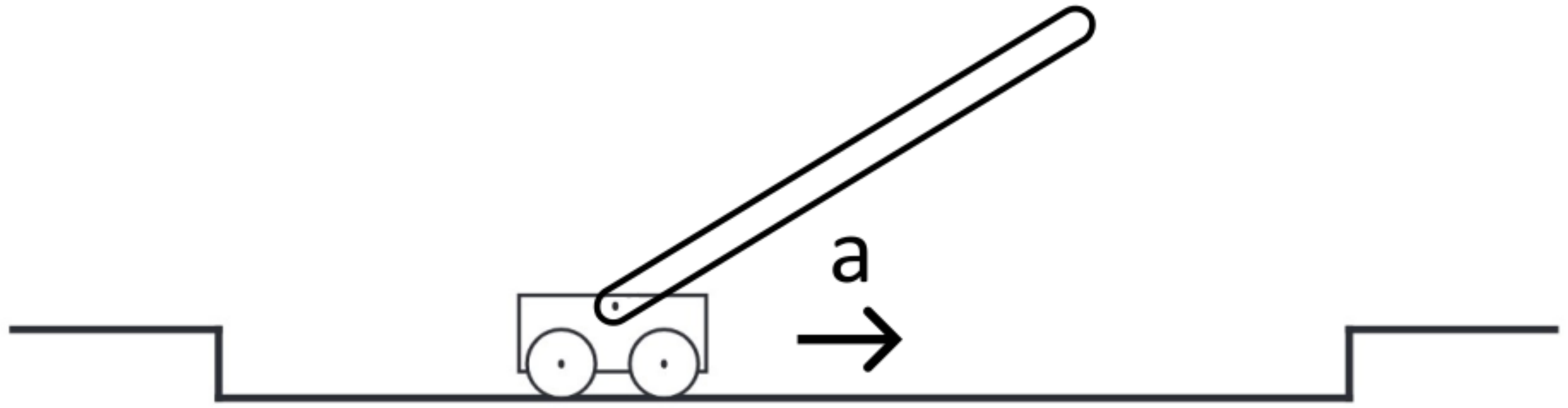
State-action value (q) functions

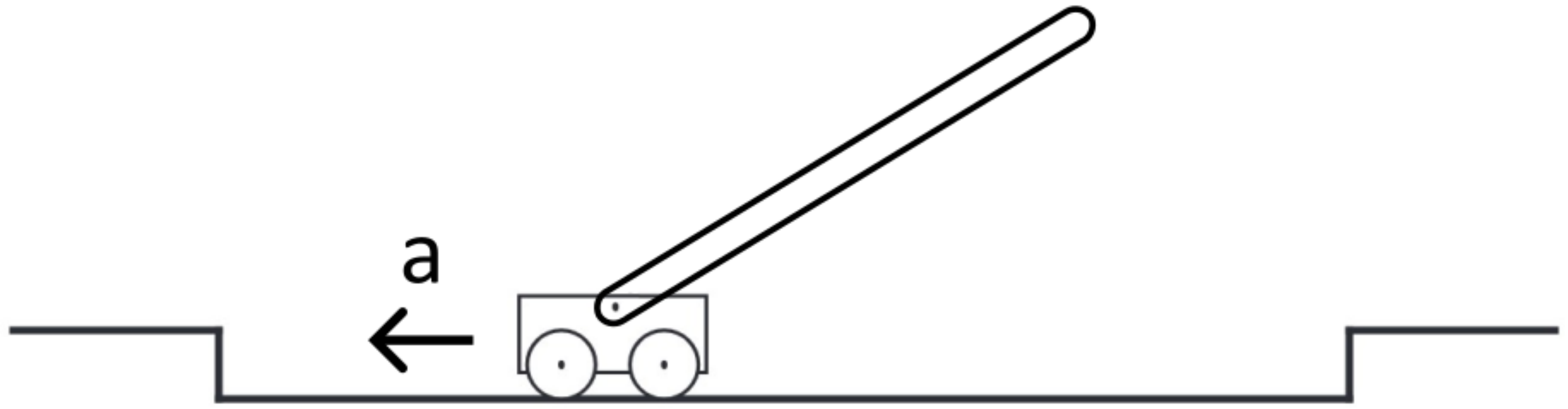
Tells us the total discounted reward we expect to receive after performing action *a* in a state *s* under policy π

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a)$$

Which action has a higher value?





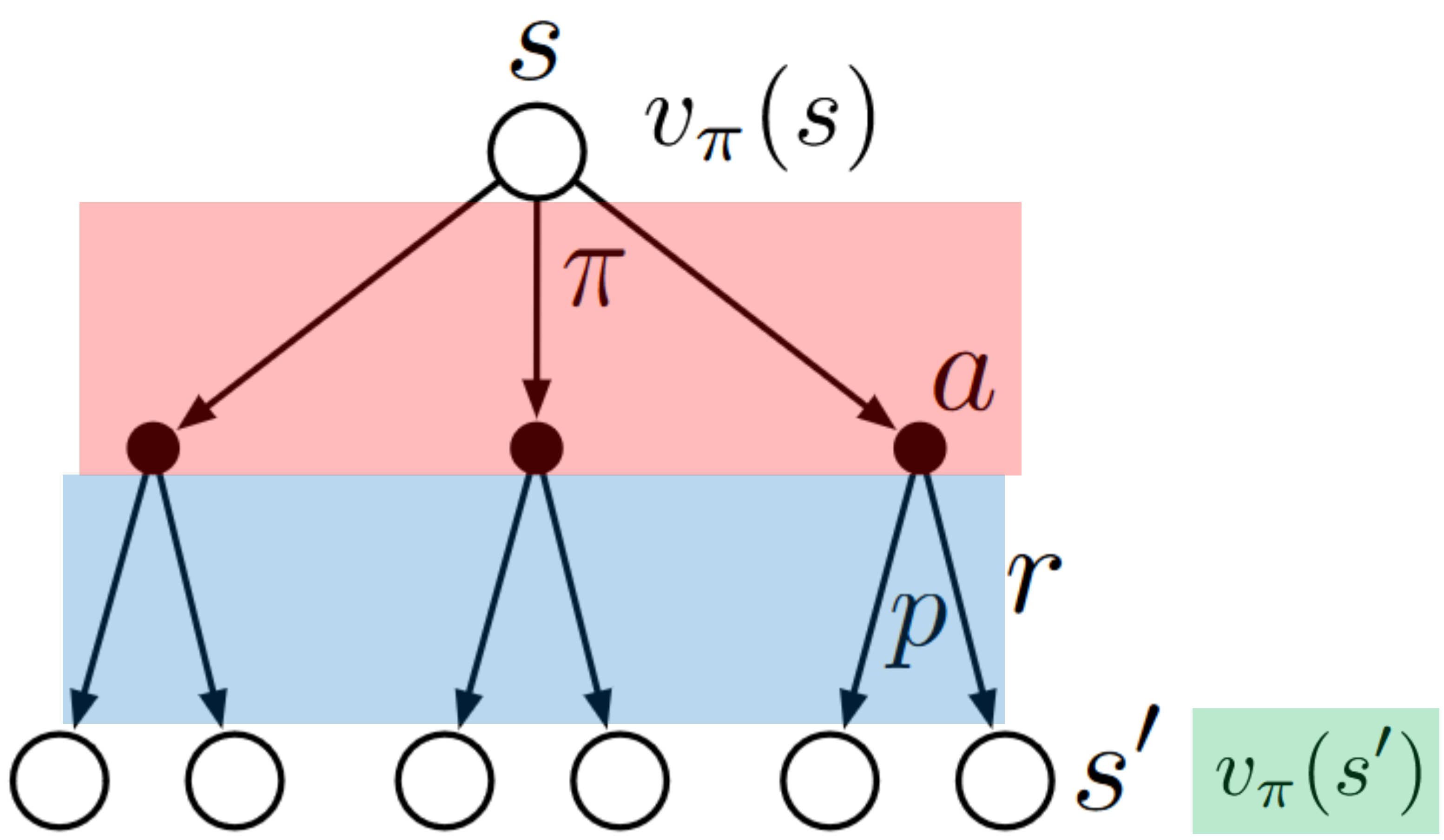
The Bellman equations

- The **Bellman equations** allow us to compute the value of states and actions
- One key idea: **recursion**
- A variable is recursive if it can be defined with a version of itself
- Our value function can be expressed recursively because G_t is an infinite sum

The Bellman equations for state values

Our value function can be expressed recursively because G_t can be expressed recursively

How can we express $v_\pi(s)$ using $v_\pi(s')$?



The Bellman equations for CartPole

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

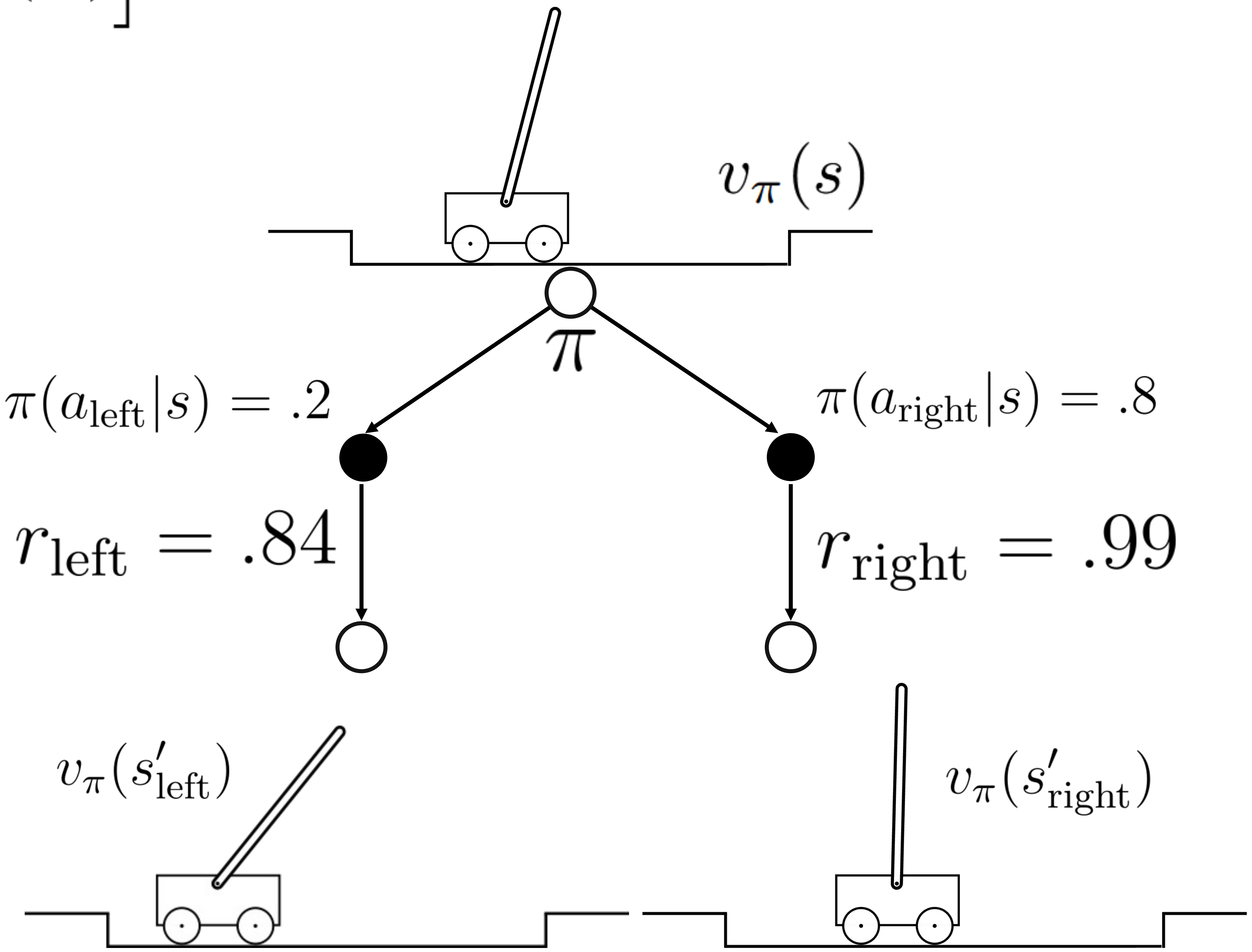
$$\pi(a_{\text{left}}|s) = .2$$

$$\pi(a_{\text{right}}|s) = .8$$

$$p(s',r|s,a) = 1 \qquad \begin{aligned} r_{\text{left}} &= .84 \\ r_{\text{right}} &= .99 \end{aligned}$$

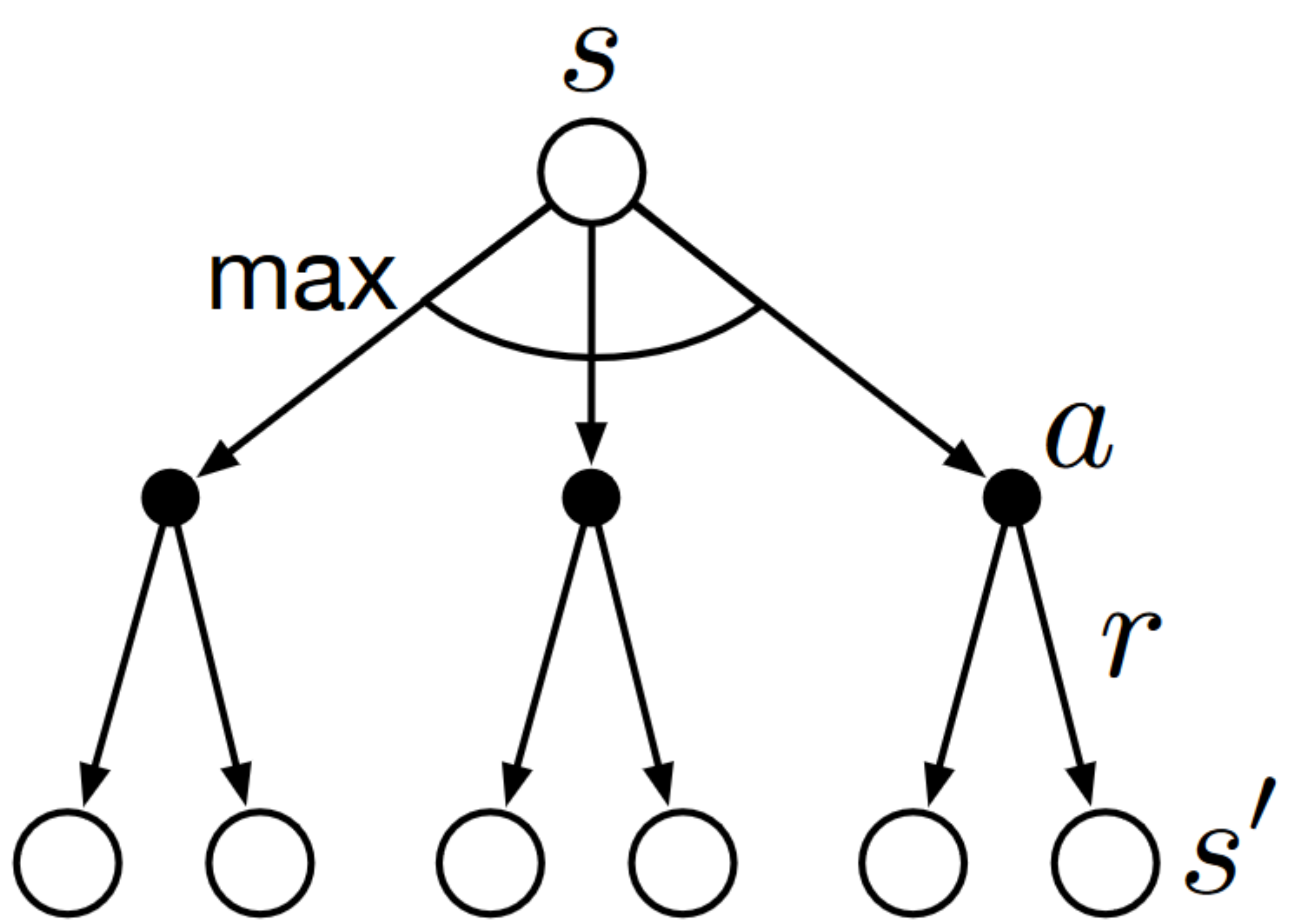
$$\begin{aligned} v_\pi(s) =& \pi(a_{\text{left}}|s)\left[r_{\text{left}} + \gamma v_\pi(s'_{\text{left}})\right] + \\ & \pi(a_{\text{right}}|s)\left[r_{\text{right}} + \gamma v_\pi(s'_{\text{right}})\right] \end{aligned}$$

$$\begin{aligned} v_\pi(s) =& .2\left[.84 + \gamma v_\pi(s'_{\text{left}})\right] + \\ & .8\left[.99 + \gamma v_\pi(s'_{\text{right}})\right] \end{aligned}$$



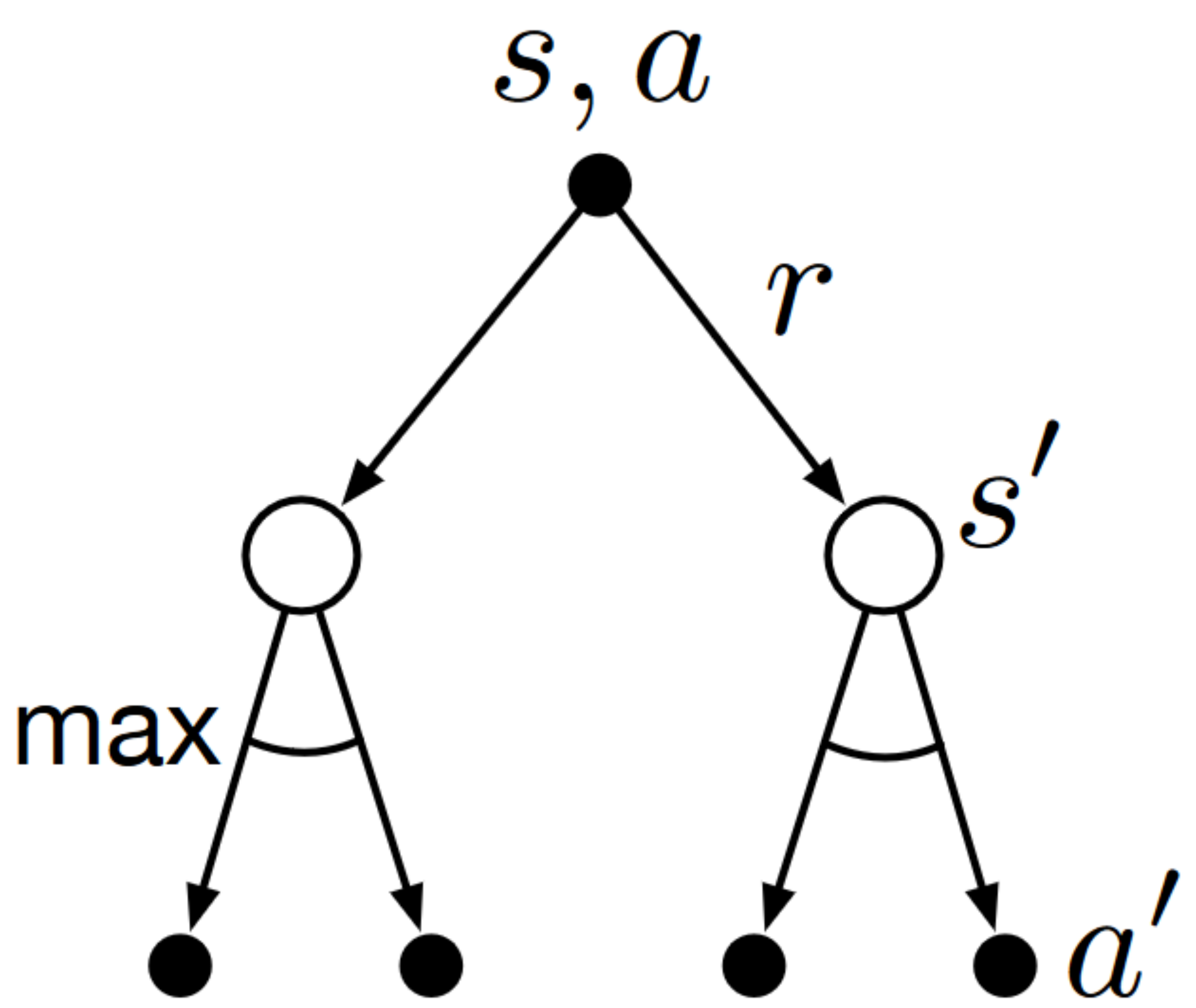
The Bellman equations for optimal state values

We can identify the optimal value function by maximizing over our actions:



The Bellman equations state-action values

We can identify the optimal state-action value function similarly:



Using Bellman in practice

The Bellman equations tell us how to behave optimally

Can they be applied directly in practice?

Requires complete knowledge of:

- Rewards / state value function
- Transition probabilities

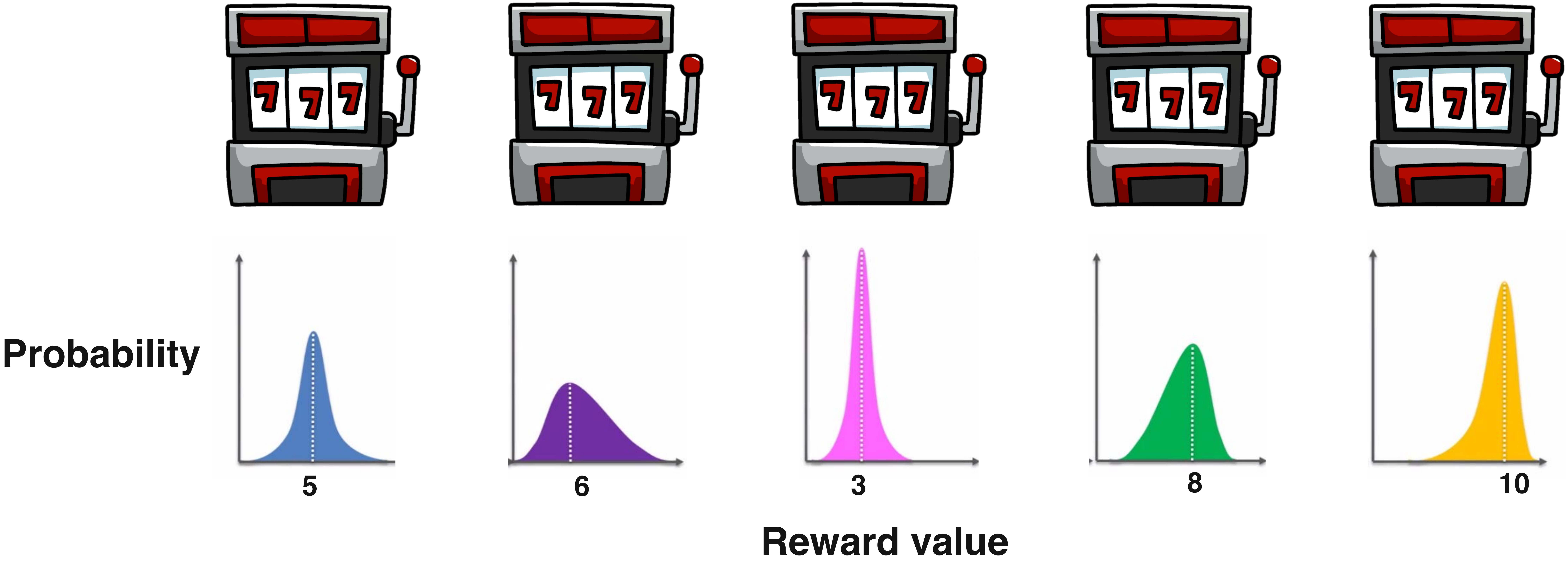
In practice, we have to estimate all of these quantities jointly

Q-learning and multi-arm bandits

Multi-arm bandits

A multi-arm bandit is any problem that involves ***k* separate choices**, each with a (possibly) different reward distribution

Named for slot machines with multiple levers (choices)



Q-learning with bandits

$$a_t \sim \pi$$

$$r_t \sim p(a_t)$$

$$\pi_* = \arg\max_{\pi} \mathbb{E}_t \left[r_t \right]$$

Choosing policies

Greedy action selection

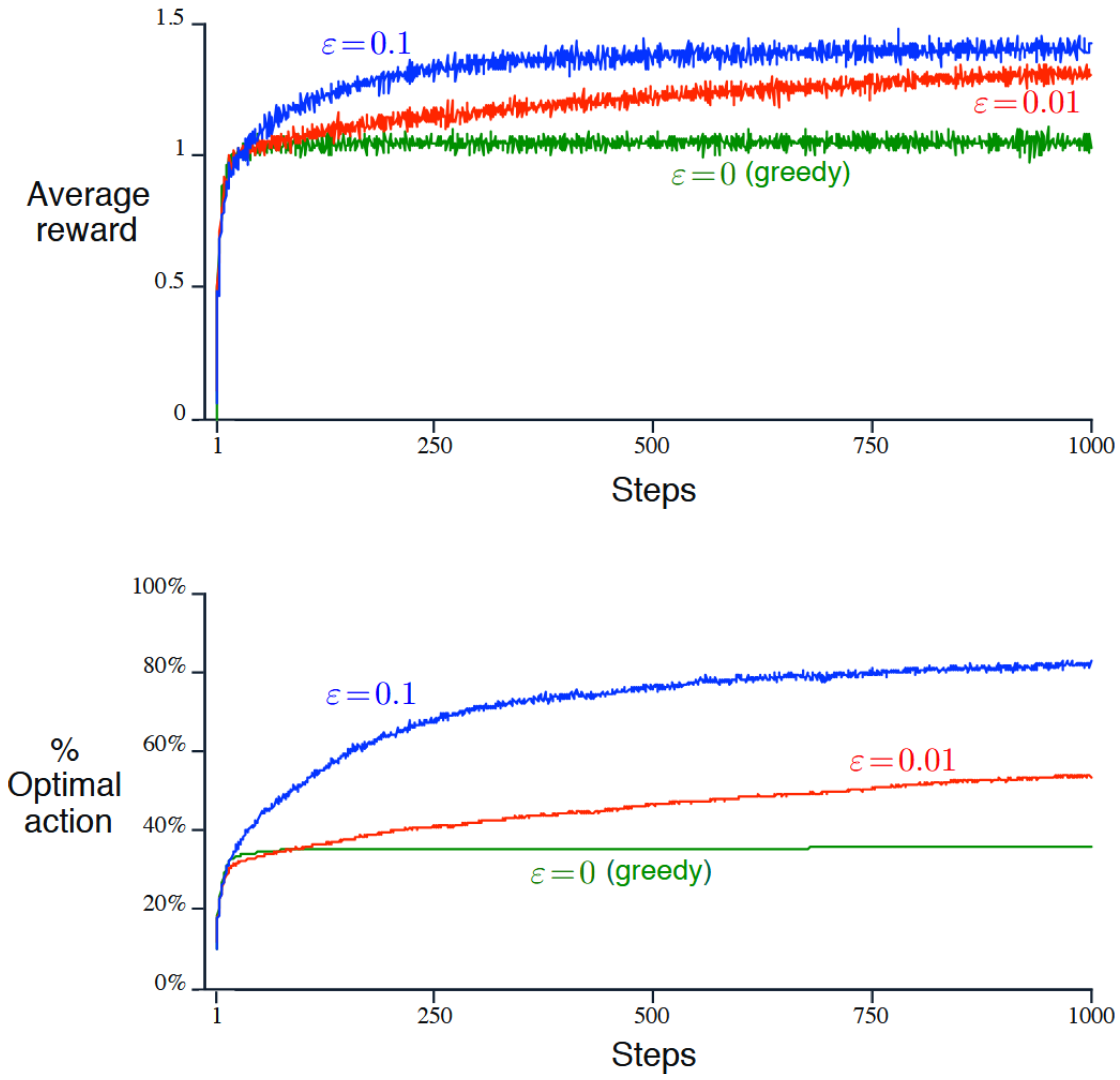
$$A_t \doteq \underset{a}{\arg\max}\, Q_t(a)$$

We can induce greater exploration by choosing a random action some percentage of timesteps

ε-greedy action selection

- With probability ε, we pick randomly
- With probability 1 – ε, we pick greedily

Some exploration is useful



Temporal difference learning (TD)

A general approach for learning q-values is called TD learning:

$$q(s,a) \leftarrow q(s,a) + \alpha * (r - q(s,a))$$

- Learning rate
  - Higher → more variable q-values
- Temporal difference (reward prediction error)
  - Only change q-values if reward is different from expected

Once we have q-values, we can choose actions probabilistically with a **softmax** strategy:

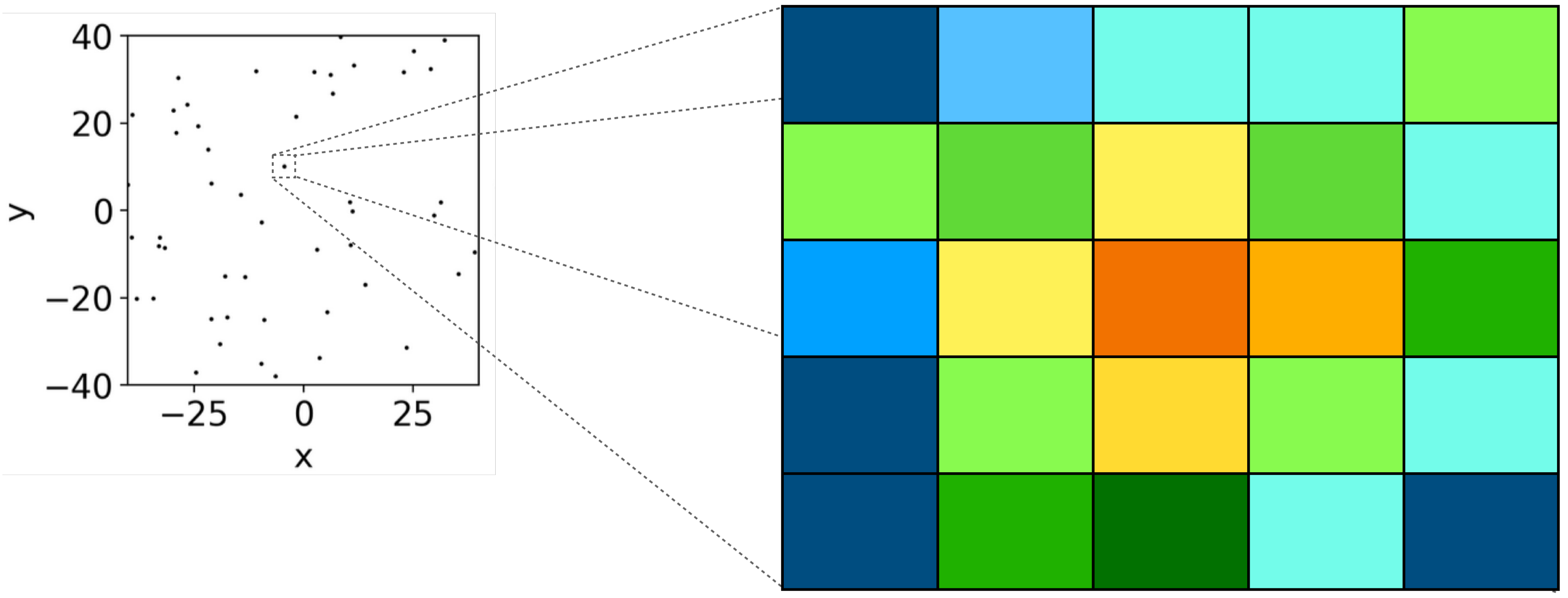
$$\mathrm{Prob}(a') = \frac{e^{q(s,a')}}{\sum_{a \in A(s)} e^{q(s,a)}} \quad q = [-.5, 1] \quad \mathrm{softmax}(q) = [.18, .82]$$

A reward seeking agent

The RL agent

Algorithm 5 RL

Set n_{max} number steps
Set β softmax parameter
Set α learning rate parameter
Initialize grid value memory $\forall i, j\ Q(i, j) = 0.5$
for *step* $= 1, \ldots, n_{max}$ **do**
    Sample value at 4 adjacent positions: *(up, down, left, right)*
    Set action policy for each adjacent position: $P(x_k) = \frac{e^{\beta x_k}}{\sum_{m=1}^{4} e^{\beta x_m}}$
    Move to the position at $\max(P(x))$
    Sample odor concentration o_s
    Change position value: $Q(i, j) = Q(i, j) + \alpha(o_s - Q(i, j))$
end for

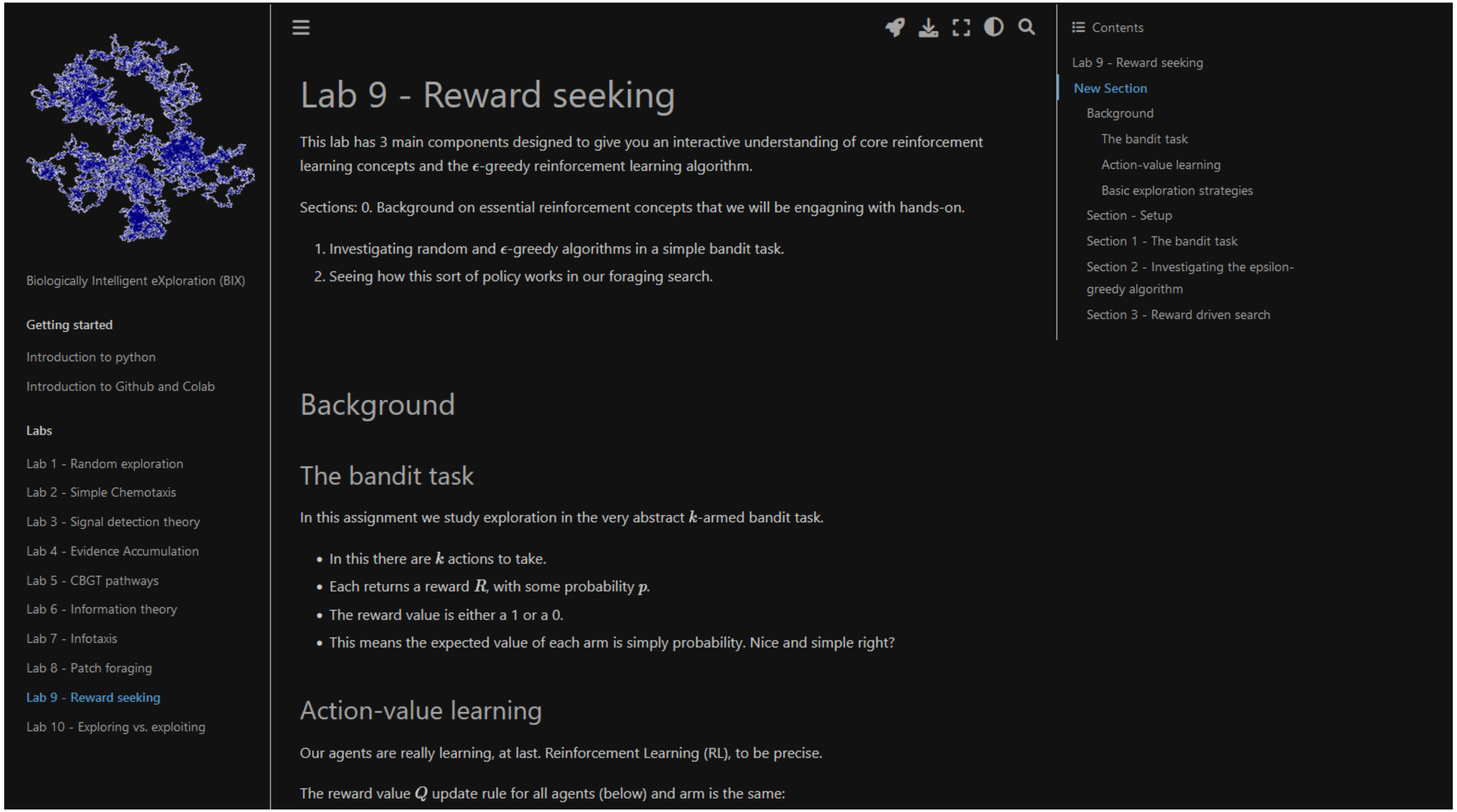


Take home message

- Reinforcement learning is broadly useful for both solving difficult problems and understanding the behavior of agents driven by rewards

- Balancing exploratory and exploitative behavior is useful for maximizing rewards

Lab 9: Reward seeking



Biologically Intelligent eXploration (BIX)

Getting started

Introduction to python

Introduction to Github and Colab

Labs

Lab 1 - Random exploration

Lab 2 - Simple Chemotaxis

Lab 3 - Signal detection theory

Lab 4 - Evidence Accumulation

Lab 5 - CBGT pathways

Lab 6 - Information theory

Lab 7 - Infotaxis

Lab 8 - Patch foraging

Lab 9 - Reward seeking

Lab 10 - Exploring vs. exploiting

Lab 9 - Reward seeking

This lab has 3 main components designed to give you an interactive understanding of core reinforcement learning concepts and the ϵ-greedy reinforcement learning algorithm.

Sections: 0. Background on essential reinforcement concepts that we will be engaging with hands-on.

1. Investigating random and ϵ-greedy algorithms in a simple bandit task.
2. Seeing how this sort of policy works in our foraging search.

Background

The bandit task

In this assignment we study exploration in the very abstract k-armed bandit task.

- In this there are k actions to take.
- Each returns a reward R, with some probability p.
- The reward value is either a 1 or a 0.
- This means the expected value of each arm is simply probability. Nice and simple right?

Action-value learning

Our agents are really learning, at last. Reinforcement Learning (RL), to be precise.

The reward value Q update rule for all agents (below) and arm is the same:

Contents

Lab 9 - Reward seeking

New Section

Background

The bandit task

Action-value learning

Basic exploration strategies

Section - Setup

Section 1 - The bandit task

Section 2 - Investigating the epsilon-greedy algorithm

Section 3 - Reward driven search