SMU: Lecture 1

(Intro to RL and Recap of MDPs)

Monday, February 14, 2022

(Heavily inspired by the Stanford RL Course of Prof. Emma Brunskill, but all potential errors are mine.)

Markov Decision Processes

(You've heard of them already and it is quite likely that you know them very well but they are important for understanding where RL algorithms come from… that's why we will review them anyways)

Part 1: Markov Processes

Random Process (Not yet MP)

- Let us have:
  - a set of **states** S, called the **state space**,
  - a random process $X_1, X_2, X_3, \ldots, X_t, \ldots$ taking values from S,
  - the state of the process at time t is the value (outcome) of X_t.

Random Processes

Markov Process (Not yet MDP)

- Let us have:
  - a set of **states** S, called the **state space**,
  - a random process $X_1, X_2, X_3, \ldots, X_t, \ldots$ taking values from S,
  - the state of the process at time t is the value (outcome) of X_t.
- **Markov property**:
  - $P[X_{t+1} = s_{t+1} \mid X_t = s_t, X_{t-1} = s_{t-1}, \ldots, X_1 = s_1] = P[X_{t+1} = s_{t+1} \mid X_t = s_t]$ for all $s_1, s_2, \ldots s_{t+1} \in S$.

The probability of transition to the next state does not depend on how we got to the present state!

Markov Property

- Markov property will be exploited in RL algorithms that we will meet in the next lectures. *(So let us spend little bit of time with it.)*

$$P[X_{t+1} = s_{t+1} \,|\, \underbrace{X_t = s_t, X_{t-1} = s_{t-1}, \ldots, X_1 = x_1}_{\textbf{History}}] = P[X_{t+1} = s_{t+1} \,|\, X_t = s_t]$$

In other words, what we are saying is that the state transition probability does not depend on the history, just on the current state. Yet in other words: Future is independent of the past given the present.

- What if a process is not Markov? Then we can make it Markov by including more information in its state.

Markov Property

Notation

- We will use the notation

$$P[X_{t+1} = s' | X_t = s] = P(s' | s)$$

whenever there will be no risk of confusion what we mean by $P(\, . \, | \, . \,)$.

Note on Stationarity

State Transition Matrix

- **State transition probabilities can be written in the form of a state transition matrix.**

$$\begin{pmatrix} P[X_{t+1} = s_1] \\ P[X_{t+1} = s_2] \\ \vdots \\ P[X_{t+1} = s_k] \end{pmatrix} = \begin{pmatrix} P(s_1 \mid s_1) & P(s_2 \mid s_1) & \dots & P(s_k \mid s_1) \\ P(s_1 \mid s_2) & P(s_2 \mid s_2) & \dots & P(s_k \mid s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1 \mid s_k) & P(s_2 \mid s_k) & \dots & P(s_k \mid s_k) \end{pmatrix}^T \begin{pmatrix} P[X_t = s_1] \\ P[X_t = s_2] \\ \vdots \\ P[X_t = s_k] \end{pmatrix}$$

Example of a Markov Process I (1/3)

- We have a six-sided die 🎲
- The state space is $S = \{0,1,2,3,4,5,6\}$.
- The "dynamics" are given as follows. If you are in a state $i \in \{0,1,\ldots,6\}$ then through the die and let the new state be: 🎲 + *"current state" mod 7.*

Example of a Markov Process I (2/3)

- We have a six-sided die 🎲
- The state space is $S = \{0,1,2,3,4,5,6\}$.
- The "dynamics" are given as follows. If you are in a state $i \in \{0,1,\ldots,6\}$ then through the die and let the new state be: 🎲 + *"current state" mod 7.*
- From this description, we can write down the transition probabilities:

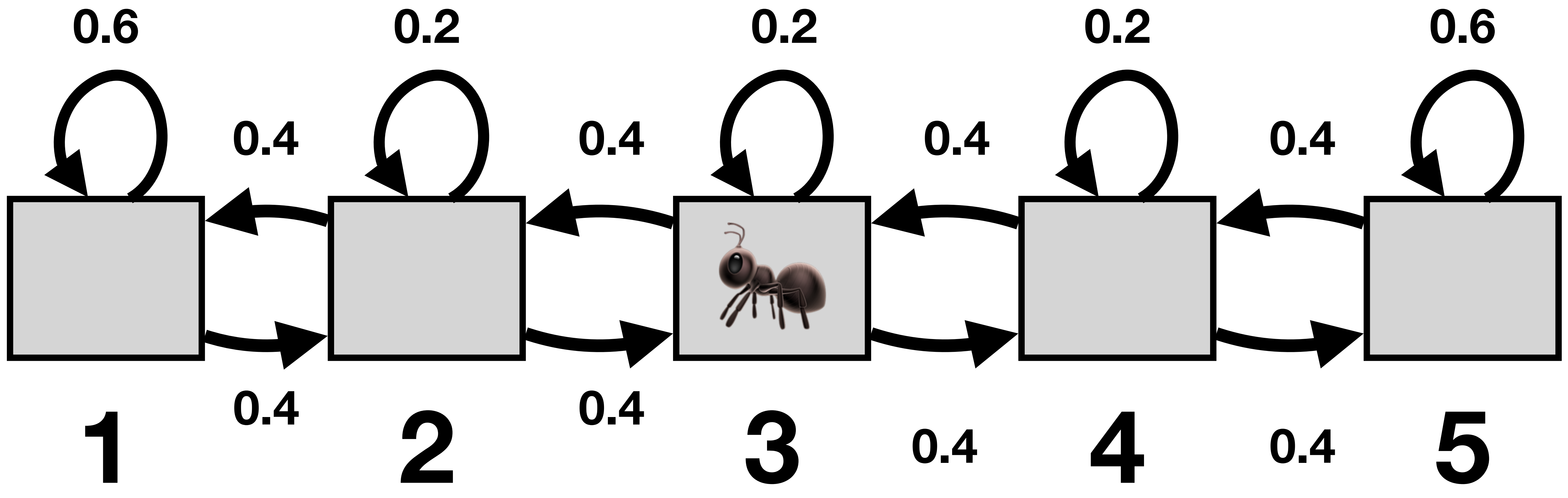
$$\begin{array}{ccccc}
P(0\,|\,0) = 0, & P(1\,|\,0) = \frac{1}{6}, & P(2\,|\,0) = \frac{1}{6}, & \ldots, & P(6\,|\,0) = \frac{1}{6} \\
P(0\,|\,1) = \frac{1}{6}, & P(1\,|\,1) = 0, & P(2\,|\,1) = \frac{1}{6}, & \ldots, & P(6\,|\,1) = \frac{1}{6} \\
\vdots, & \vdots, & \vdots, & \ddots, \vdots & \\
P(0\,|\,6) = \frac{1}{6}, & P(1\,|\,6) = 0, & P(2\,|\,6) = \frac{1}{6}, & \ldots, & P(6\,|\,6) = 0
\end{array}$$

Example of a Markov Process I (3/3)

$$P = \begin{pmatrix} 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 \end{pmatrix}$$

Example of a Markov Process (3/3)

Another Example 🐜 (1/2)

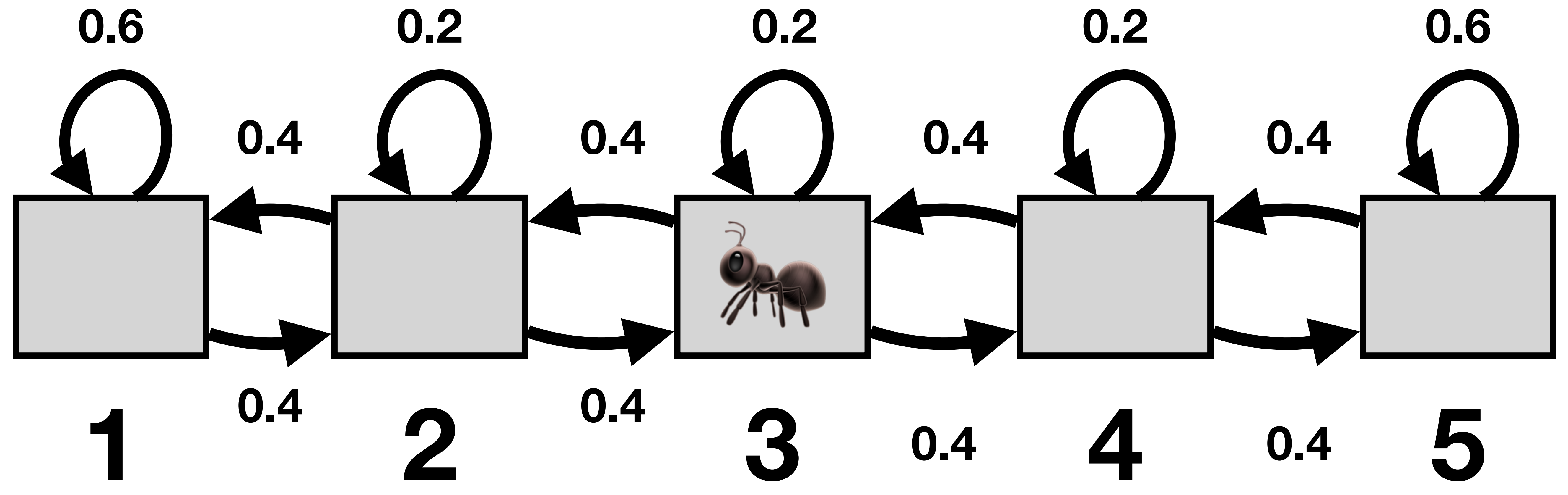


The ant moves left with probability 0.4, right with probability 0.4 and stays where it is with probability 0.2, except for the borders (s1 and s5) where it stays with probability 0.6.

A sample episode starting from s_3:

3,3,2,1,2,2,3,4,…

Another Example 🐜 (2/2)

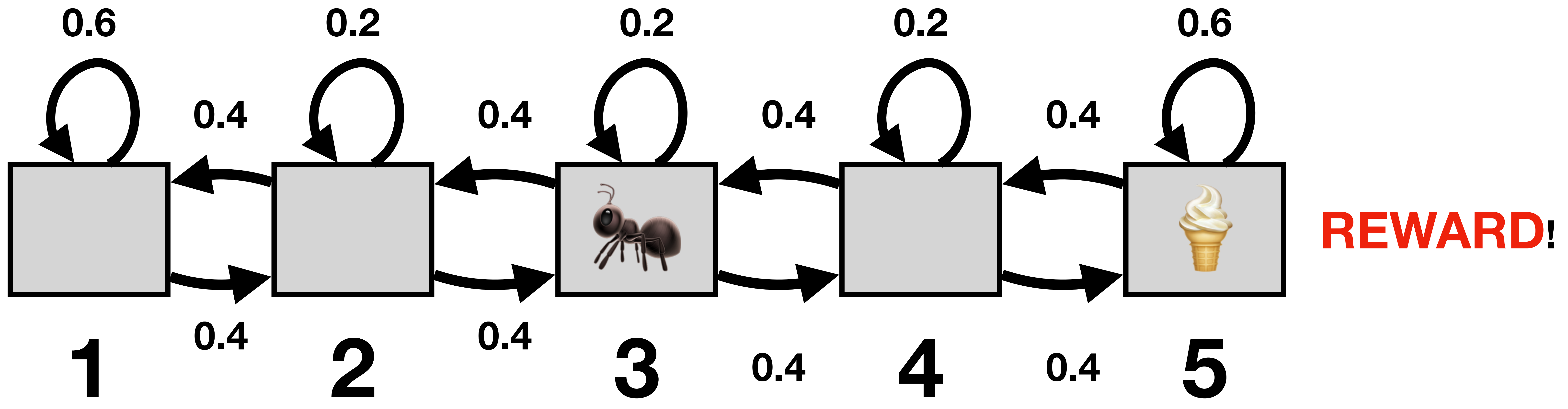


$$P = \begin{pmatrix} 0.6 & 0.4 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 & 0 \\ 0 & 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0.4 & 0.2 & 0.4 \\ 0 & 0 & 0 & 0.4 & 0.6 \end{pmatrix}$$

Part 2: Markov Reward Processes

Markov Reward Process

Markov reward process = Markov process + Reward



Markov Reward Process

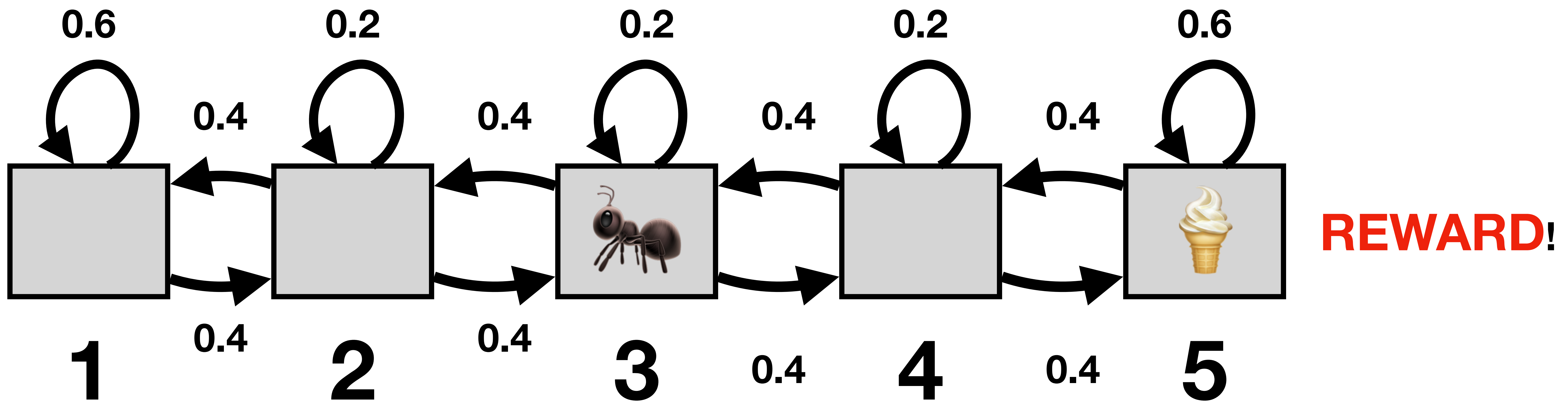
Markov reward process = Markov process + Reward

Formally, MRP is given by:

- A set of states S.
- A transition model $P[X_{t+1} = s' | X_t = s]$, which we also denote by $P(s' | s)$.
- A reward function $R(s) = \mathbb{E}[R_t | X_t = s]$, which is the expected reward the agent receives in state s, $(s \in S)$.
- A discount factor $\gamma \in [0; 1]$.

Markov Reward Process

Markov reward process = Markov process + Reward



For example:

$$R(s) = \begin{cases} 0, & s = 1 \\ 0, & s = 2 \\ 0, & s = 3 \\ 0, & s = 4 \\ 10, & s = 5 \end{cases}$$

$\longleftarrow$ **We expect that each time we visit s_5, there will be ice cream (i.e. we are not running out of it).**

Return from an Episode

- **Horizon:**
  - Number of time steps in an episode (which can also be infinite). **We will first assume infinite horizons** (they are easier because they will lead to stationary, i.e. time-independent, policies!).

- **Return G_t:**
  - **Given:** An episode $s_1, s_2, s_3, s_4, \ldots, s_H$.
  - **Compute:** Return g_t = discounted sum of rewards from time t.
  - **As a formula:**

$$g_t = R(s_t) + R(s_{t+1}) \cdot \gamma + R(s_{t+2}) \cdot \gamma^2 + \ldots = R(s_t) + \sum_{i=1} R(s_{t+i}) \cdot \gamma^i$$

Return (Random Variable)

- What we had on the previous slide was return from one specific sampled episode.
- Next we define **return** of a Markov reward process as a random variable (it is important to understand the distinction between the two):

$$G_t = R(X_t) + \gamma \cdot R(X_{t+1}) + \gamma^2 \cdot R(X_{t+2}) + \ldots = \sum_{i=0}^{\infty} R(X_{t+i}) \cdot \gamma^i$$

Note: Discount Factor

- Honestly, the discount factor and how it is used makes a lot of things mathematically convenient. *(You will see in a moment or maybe you remember it from other courses.)*
- It also makes the return finite even for problems with infinite horizon.
- But the discount also makes sense practically — the same reward today is better than tomorrow.
- Special cases:
  - $\gamma = 0$: only immediate reward counts.
  - $\gamma = 1$: future rewards matter as much as present rewards.

(State) Value Function

- **Definition:**

$$V(s) = \mathbb{E}[G_t \,|\, X_t = s] = \mathbb{E}[R(X_t) + \gamma \cdot R(X_{t+1}) + \gamma^2 \cdot R(X_{t+2}) + \ldots \,|\, X_t = s]$$

It seems from this definition that $V(s)$ should depend on t. But is that really the case? Think of the definition of G_t and of the Markov property (and stationarity of MRP)! Indeed, t can be anything and the value function of a state s will not change.

- **Intuition:** Value function $V(s)$ is the expected return when starting from state s.

Computing Value Function (1/3)

$$V(s) = \mathbb{E}[G_t \,|\, X_t = s] = \mathbb{E}[R(X_t) + \gamma \cdot R(X_{t+1}) + \gamma^2 \cdot R(X_{t+2}) + \ldots \,|\, X_t = s]$$

$$= R(s) + \gamma \mathbb{E}[R(X_{t+1}) + \gamma \cdot R(X_{t+2}) + \ldots \,|\, X_t = s] =$$

$$= R(s) + \gamma \cdot \sum_{s' \in S} P(s' \,|\, s) \cdot \mathbb{E}[R(X_{t+1}) + \gamma \cdot R(X_{t+2}) + \ldots \,|\, X_{t+1} = s']$$

$$= R(s) + \gamma \cdot \sum_{s' \in S} P(s' \,|\, s) \cdot V(s').$$

Computing Value Function (2/3)

$V(s) = R(s) + \gamma \cdot \sum_{s' \in S} P(s' | s) \cdot V(s')$ for all $s \in S$, is nothing else then a system of linear equation, which we can write in the matrix form for finite S as:

$$\begin{pmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_n) \end{pmatrix} = \begin{pmatrix} R(s_1) \\ R(s_2) \\ \vdots \\ R(s_n) \end{pmatrix} + \underbrace{\begin{pmatrix} P(s_1 | s_1) & P(s_2 | s_1) & \dots & P(s_n | s_1) \\ P(s_1 | s_2) & P(s_2 | s_2) & \dots & P(s_n | s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1 | s_n) & P(s_2 | s_n) & \dots & P(s_n | s_n) \end{pmatrix}}_{=P} \begin{pmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_n) \end{pmatrix}$$

Unfortunately, solving the system directly, e.g. as $\mathbf{V} = (I - \gamma P)^{-1} R$, is slow in practice.
We will describe how to solve similar problems for MDPs (hance also for MRPs)

Computing Value Function (3/3)

- **An alternative is to use an iterative algorithm (exploiting dynamic programming)***

```
Set $V_0(s) = 0$ for all $s \in S$
For $k = 1, \dots$
    For $\forall s \in S$:
        $V_k(s) = R(s) + \gamma \cdot \sum_{s' \in S} P(s' | s) \cdot V_{k-1}(s')$
    if converged** (with some tolerance) then return $V_k$
```

Bellman update

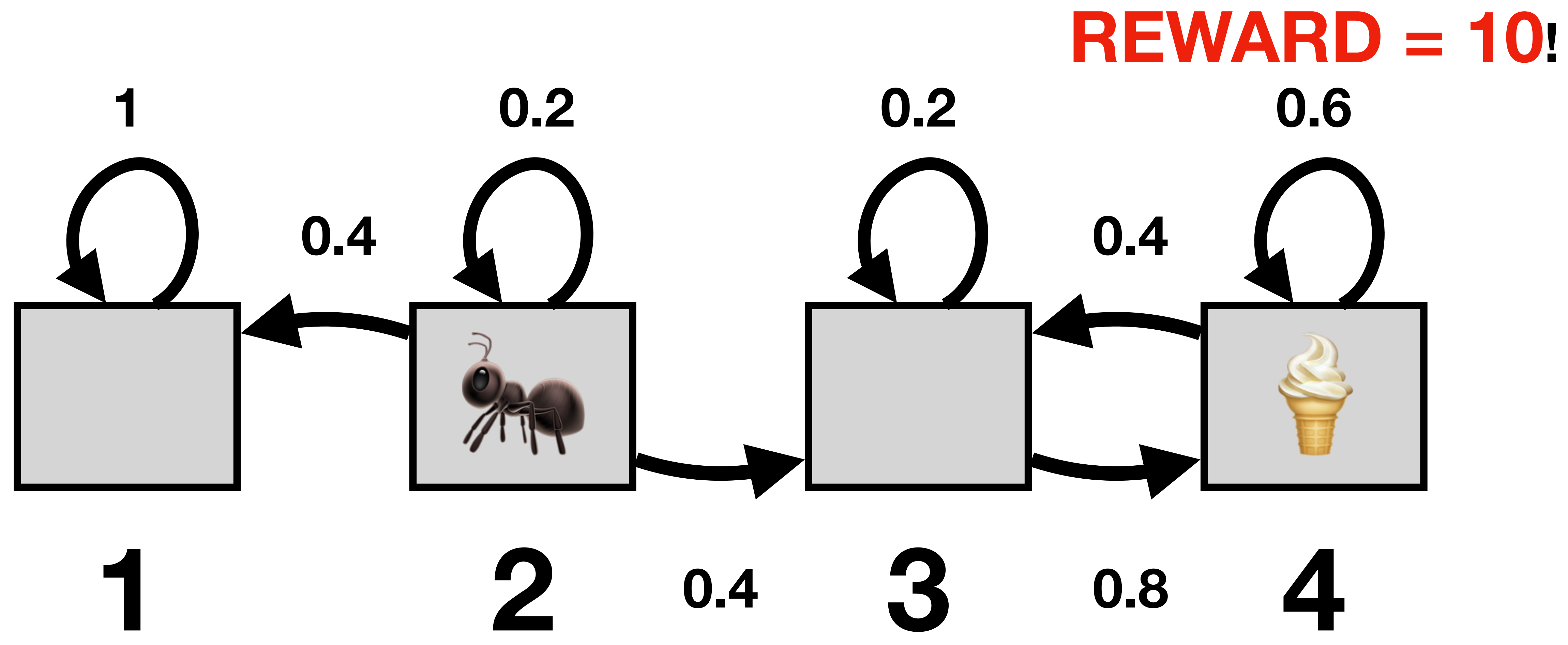
*This is nothing else than an iterative method for solving linear equations but it has a nicer interpretation of you think of it in terms of the MRP.

**For instance, we can use $||V_k - V_{k-1}||_\infty \leq \varepsilon$.

Value Function (Example) $\gamma = 0.5$

$$\boxed{V(s) = R(s) + \gamma \cdot \sum_{s' \in S} P(s' \mid s) \cdot V(s')}$$

$$V(s_1) = \underbrace{R(s_1)}_{=0} + \gamma \cdot \underbrace{P(s_1 \mid s_1)}_{=1} \cdot V(s_1)$$



$$V(s_2) = \underbrace{R(s_2)}_{=0} + \gamma \cdot (\underbrace{P(s_1 \mid s_2)}_{=0.4} \cdot V(s_1) + \underbrace{P(s_2 \mid s_2)}_{=0.2} \cdot V(s_2) + \underbrace{P(s_3 \mid s_2)}_{=0.4} \cdot V(s_3))$$

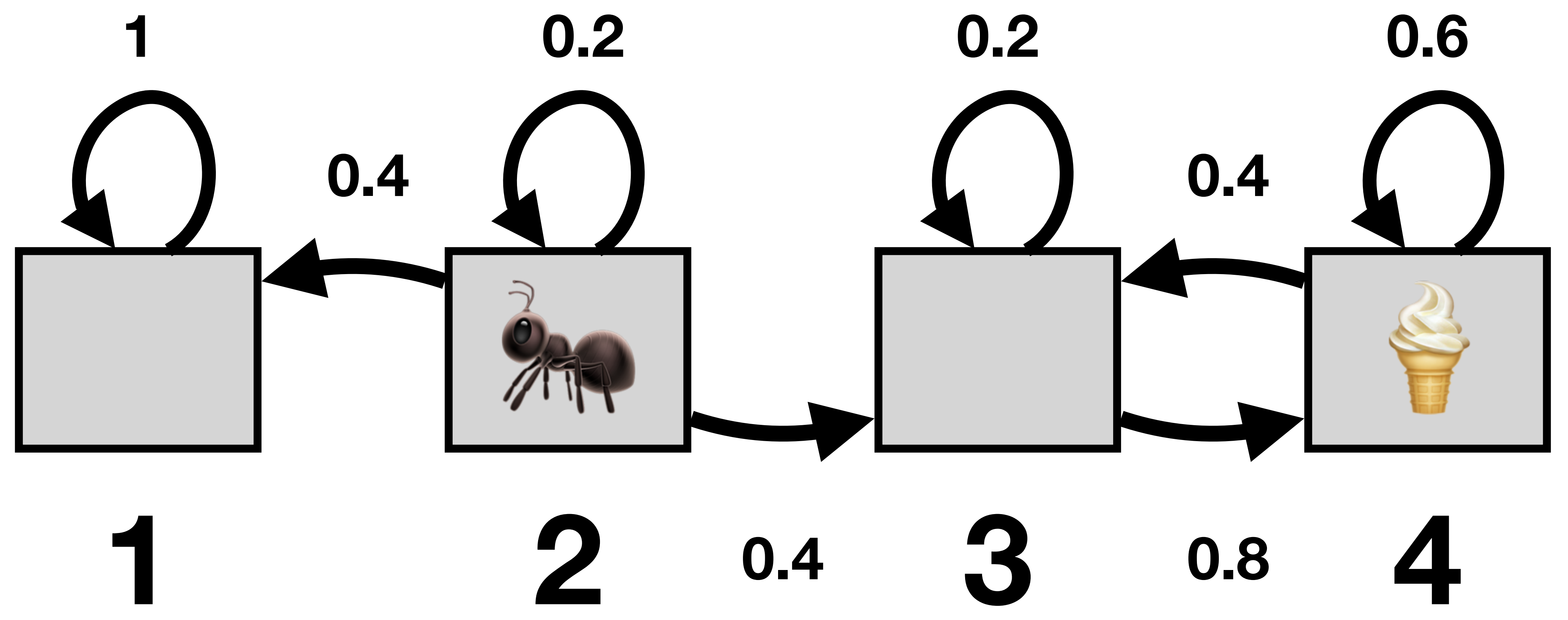
$$V(s_3) = \underbrace{R(s_3)}_{=0} + \gamma \cdot (\underbrace{P(s_3 \mid s_3)}_{=0.2} \cdot V(s_3) + \underbrace{P(s_4 \mid s_3)}_{=0.8} \cdot V(s_4))$$

$$V(s_4) = \underbrace{R(s_4)}_{=10} + \gamma \cdot (\underbrace{P(s_3 \mid s_4)}_{=0.4} \cdot V(s_3) + \underbrace{P(s_4 \mid s_4)}_{=0.6} \cdot V(s_4))$$

Value Function (Example) $\gamma = 0.5$

REWARD = 10!

$$\boxed{V(s) = R(s) + \gamma \cdot \sum_{s' \in S} P(s' \mid s) \cdot V(s')}$$



$$V(s_1) = 0.5 \cdot V(s_1)$$

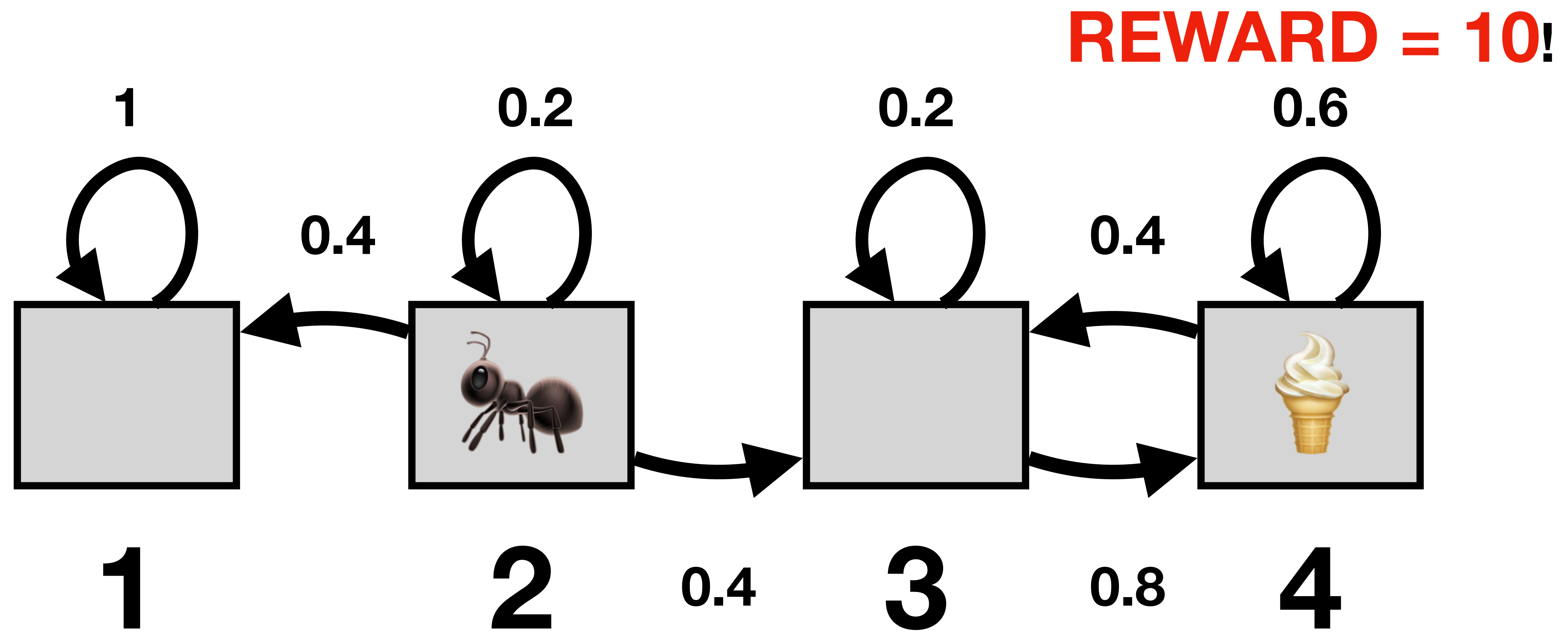
$$V(s_2) = 0.5 \cdot (0.4 \cdot V(s_1) + 0.2 \cdot V(s_2) + 0.4 \cdot V(s_3))$$

$$V(s_3) = 0.5 \cdot (0.2 \cdot V(s_3) + 0.8 \cdot V(s_4))$$

$$V(s_4) = 10 + 0.5 \cdot (0.4 \cdot V(s_3) + 0.6 \cdot V(s_4))$$

Value Function (Example) $\gamma = 0.5$

$$\boxed{V(s) = R(s) + \gamma \cdot \sum_{s' \in S} P(s' \mid s) \cdot V(s')}$$



$$V(s_1) = 0.5 \cdot V(s_1)$$

$$V(s_2) = 0.5 \cdot (0.4 \cdot V(s_1) + 0.2 \cdot V(s_2) + 0.4 \cdot V(s_3))$$

$$V(s_3) = 0.5 \cdot (0.2 \cdot V(s_3) + 0.8 \cdot V(s_4))$$

$$V(s_4) = 10 + 0.5 \cdot (0.4 \cdot V(s_3) + 0.6 \cdot V(s_4))$$

By solving the set of equations directly:

$$V(s_1) = 0,\ V(s_2) \approx 1.62,\ V(s_3) \approx 7.27,\ V(s_4) \approx 16.36$$

Value Function (Iterative Solution)

Iteration 0:

$$V_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Value Function (Iterative Solution)

Iteration 1:

$$V_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0.2 & 0.8 \\ 0 & 0 & 0.4 & 0.6 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \end{pmatrix}$$

Value Function (Iterative Solution)

Iteration 2:

$$V_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0.2 & 0.8 \\ 0 & 0 & 0.4 & 0.6 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0.4 \\ 13 \end{pmatrix}$$

Value Function (Iterative Solution)

Iteration 3:

$$V_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0.2 & 0.8 \\ 0 & 0 & 0.4 & 0.6 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0.4 \\ 13 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.08 \\ 5.24 \\ 13.98 \end{pmatrix}$$

Value Function (Iterative Solution)

Iteration 4:

$$V_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0.2 & 0.8 \\ 0 & 0 & 0.4 & 0.6 \end{pmatrix} \begin{pmatrix} 0 \\ 0.08 \\ 5.24 \\ 13.98 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.056 \\ 6.116 \\ 15.242 \end{pmatrix}$$

Value Function (Iterative Solution)

Iteration 5:

$$V_5 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0.2 & 0.8 \\ 0 & 0 & 0.4 & 0.6 \end{pmatrix} \begin{pmatrix} 0 \\ 1.056 \\ 6.116 \\ 15.242 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.3288 \\ 6.7084 \\ 15.7958 \end{pmatrix}$$

Value Function (Iterative Solution)

Iteration 6:

$$V_6 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0.2 & 0.8 \\ 0 & 0 & 0.4 & 0.6 \end{pmatrix} \begin{pmatrix} 0 \\ 1.3288 \\ 6.7084 \\ 15.7958 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.47456 \\ 6.98916 \\ 16.08042 \end{pmatrix}$$

Value Function (Iterative Solution)

Iteration 7:

$$V_7 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0.2 & 0.8 \\ 0 & 0 & 0.4 & 0.6 \end{pmatrix} \begin{pmatrix} 0 \\ 1.47456 \\ 6.98916 \\ 16.08042 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.545288 \\ 7.131084 \\ 16.221958 \end{pmatrix}$$

Value Function (Iterative Solution)

Iteration 8:

$$V_8 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0.2 & 0.8 \\ 0 & 0 & 0.4 & 0.6 \end{pmatrix} \begin{pmatrix} 0 \\ 1.545288 \\ 7.131084 \\ 16.221958 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.5807456 \\ 7.2018916 \\ 16.2928042 \end{pmatrix}$$

Value Function (Iterative Solution)

Iteration 8:

$$V_8 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \end{pmatrix} + 0.5 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0.2 & 0.8 \\ 0 & 0 & 0.4 & 0.6 \end{pmatrix} \begin{pmatrix} 0 \\ 1.545288 \\ 7.131084 \\ 16.221958 \end{pmatrix} = \begin{pmatrix} 0 \\ 1.5807456 \\ 7.2018916 \\ 16.2928042 \end{pmatrix}$$

$$|V_8 - V_\infty| \approx \begin{pmatrix} 0 \\ 0.035 \\ 0.071 \\ 0.071 \end{pmatrix}$$

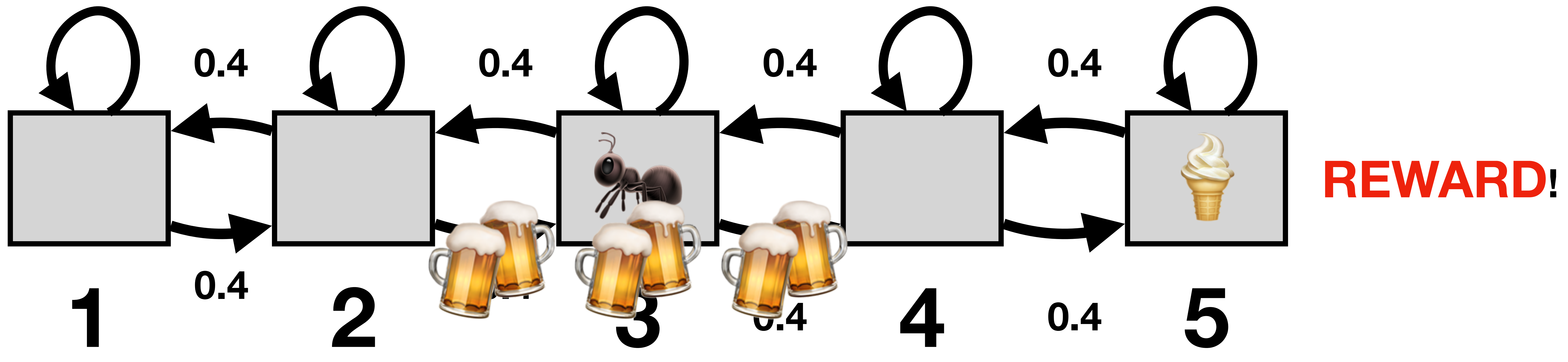
Part 3: Markov Decision Processes

Markov Decision Process

- **Markov decision process = Markov reward process + Actions**
- **An MDP is given by:**
  - A set of states S.
  - A set of actions A.
  - A transition model $P(X_{t+1} = s' \,|\, X_t = s, A_t = a) \underbrace{= P(s' \,|\, s, a)}_{\text{notation}}$
  - A reward $R(s, a) = \mathbb{E}[R_t \,|\, X_t = s, A_t = a]$, i.e. the expected reward that the agent receives when performing action a in state s.
  - Discount factor γ.

Transition Model

- A bit of intuition about $P(X_{t+1} = s' \mid X_t = s, A_t = a)$:
    - Why is this random and not deterministic? Imagine that our ant is drunk and if it wants to go left, it actually goes right with some probability. Or imagine that the action is to throw a die in a game or pick a card from a deck…



MRP vs MDP

- **Compare:**

MRP

Dynamics:

$$P[X_{t+1} = s' | X_t = s]$$

Return:

$$R(s) = \mathbb{E}[R_t | X_t = s]$$

MDP

Dynamics:

$$P[X_{t+1} = s' | A_t = a, X_t = s]$$

Return:

$$R(s,a) = \mathbb{E}[R_t | X_t = s, A_t = a].$$

Policy

- Policy determines which action to take in each state s.
- It can be either deterministic or random — that is also why policy will not simply be a function from states to actions.
- **We define policy:** $\pi(a \mid s) = P(A_t = a \mid X_t = s)$.
- **Example** (policy for our ant 🐜):
  - $A = \{\text{left}, \text{right}\}$
  - $\pi(\text{left} \mid 1) = 0$, $\pi(\text{right} \mid 1) = 1$, $\pi(\text{left} \mid 2) = 0.5$, $\pi(\text{right} \mid 1) = 0.5, \ldots$

MDP+Policy = MRP

- When we specify a policy for a given MDP, we are effectively turning the MDP into a corresponding MRP.

- **Formally:**

    - Given an MDP (A, S, P, R, γ), we turn it into an MRP $(S, P^\pi, R^\pi, \gamma)$ where

$$P^\pi(s' \mid s) = \sum_{a \in A} \pi(a \mid s) \cdot P(s' \mid s, a) \text{ *}$$

$$R^\pi(s) = \sum_{a \in A} \pi(a \mid s) \cdot R(s, a)$$

* In the more verbose notation: $P^\pi[X_{t+1} = s' \mid X_t = s] = \sum_{a \in A} \pi(a \mid s) \cdot P[X_{t+1} = s' \mid A_t = a, X_t = s]$.

State Value Function of MDP (1/3)

$$V^{\pi}(s) = R^{\pi}(s) + \gamma \cdot \sum_{s' \in S} P^{\pi}(s' \mid s) \cdot V^{\pi}(s')$$

State Value Function of MDP (2/3)

$$V^\pi(s) = \underbrace{\boxed{\sum_{a\in A} \pi(a\,|\,s)\cdot R(s,a)}}_{} + \gamma \cdot \sum_{s'\in S} \boxed{\sum_{a\in A} \pi(a\,|\,s)\cdot P(s'\,|\,s,a)} \cdot V^\pi(s')$$

$$R^\pi(s) = \sum_{a\in A} \pi(a\,|\,s)\cdot R(s,a)$$

$$P^\pi(s'\,|\,s) = \sum_{a\in A} \pi(a\,|\,s)\cdot P(s'\,|\,s,a)$$

State Value Function of MDP (3/3)

$$V^{\pi}(s) = \sum_{a \in A} \pi(a, s) \cdot \left[R(s, a) + \gamma \cdot \sum_{s' \in S} P(s' \mid s, a) \cdot V^{\pi}(s') \right]$$

(Bellman equation for MDP)

MDP Policy Evaluation - Iteration (1/3)

- Since we reduced MDP (A, S, P, R, γ) + policy to the MRP $(S, P^\pi, R^\pi, \gamma)$, we can use the same iterative method for computing the value function $V^\pi(s)$.

```
*Set* $V_0(s) = 0$ *for all* $s \in S$
***For*** $k = 1,\dots$
  ***For*** $\forall s \in S$:
    $V_k^\pi(s) = R^\pi(s) + \gamma \cdot \sum_{s' \in S} P^\pi(s' \mid s) \cdot V_{k-1}^\pi(s')$
  ***if*** *converged (with some tolerance) then* ***return*** $V_k^\pi$
```

MDP Policy Evaluation - Iteration (2/3)

- Since we reduced MDP (A, S, P, R, γ) + policy to the MRP $(S, P^\pi, R^\pi, \gamma)$, we can use the same iterative method for computing the value function $V^\pi(s)$.

```
*Set* $V_0(s) = 0$ *for all* $s \in S$
**For** $k = 1, \ldots$
  **For** $\forall s \in S$:
    $V_k^\pi(s) = \overset{R^\pi(s)}{\overset{||}{\boxed{\sum_{a\in A} \pi(a|s) \cdot R(s,a)}}} + \gamma \cdot \sum_{s'\in S} \overset{P^\pi(s'|s)}{\overset{||}{\boxed{\sum_{a\in A} \pi(a|s) \cdot P(s'|s,a)}}} \cdot V_{k-1}^\pi(s')$
  *if converged (with some tolerance) then* ***return*** $V_k$
```

MDP Policy Evaluation - Iteration (3/3)

- Since we reduced MDP (A, S, P, R, γ) + policy to the MRP $(S, P^{\pi}, R^{\pi}, \gamma)$, we can use the same iterative method for computing the value function $V^{\pi}(s)$.

```
Set $V_0(s) = 0$ for all $s \in S$
For $k = 1, \ldots$
  For $\forall s \in S$:
    $V_k^{\pi}(s) = \sum_{a \in A} \pi(a \mid s) \cdot \left( R(s, a) + \gamma \cdot \sum_{s' \in S} P(s' \mid s, a) \cdot V_{k-1}^{\pi}(s') \right)$
  if converged (with some tolerance) then return $V_k$
```

Part 4: MDP Control

MDP Control: What is it?

- We want to find a policy π^* that will maximize the value function for all states (i.e. we want to learn to behave optimally in every state).
- **Formally:**

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$$

- **One can show that:**
    - A unique optimal **value function** exists, but… the optimal policy does not have to be unique.
    - **For an infinite horizon problem**, there exists a **deterministic** optimal policy (there may also be a non-deterministic optimal policy) and the policy is stationary (*this is why it is convenient to work with infinite-horizon MDPs*).

MDP Control Problem

How to find $\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$???

State-Action Value Q

- **Definition:**

$$Q^{\pi}(s,a) = R(s,a) + \gamma \cdot \sum_{s' \in S} P(s' \mid s,a) \cdot V^{\pi}(s').$$

- **Intuition:**
  - The value of the return that we obtain if we first take the action a in the state s and then follow the policy π (including when we visit s again).
  - *Think of it as perturbing the policy π — we deviate from following the policy π only in the first step in s.*

Policy Improvement Step

- **Given:** An MDP and a **policy π_i that we want to improve** (if possible).
- **DO:**
    - For all $s \in S$, compute $Q^{\pi_i}(s,a)$ as defined on the previous slide, i.e. $Q^{\pi_i}(s,a) = R(s,a) + \gamma \cdot \sum_{s' \in S} P(s' \mid s,a) \cdot V^{\pi_i}(s')$.
    - **Compute new policy for all $s \in S$:**

$$\pi_{i+1}(s) = \arg\max_{a \in S} Q^{\pi_i}(s,a)$$

Here, we use the fact that our policy is deterministic for simpler notation (treating policy as a function). Using our previous notation we could write:

$$\pi(a \mid s) = \begin{cases} 1 & \textit{if } a = \arg\max_{a \in A} Q^{\pi_i}(s,a) \\ 0 & \textit{otherwise} \end{cases}$$

Policy Iteration

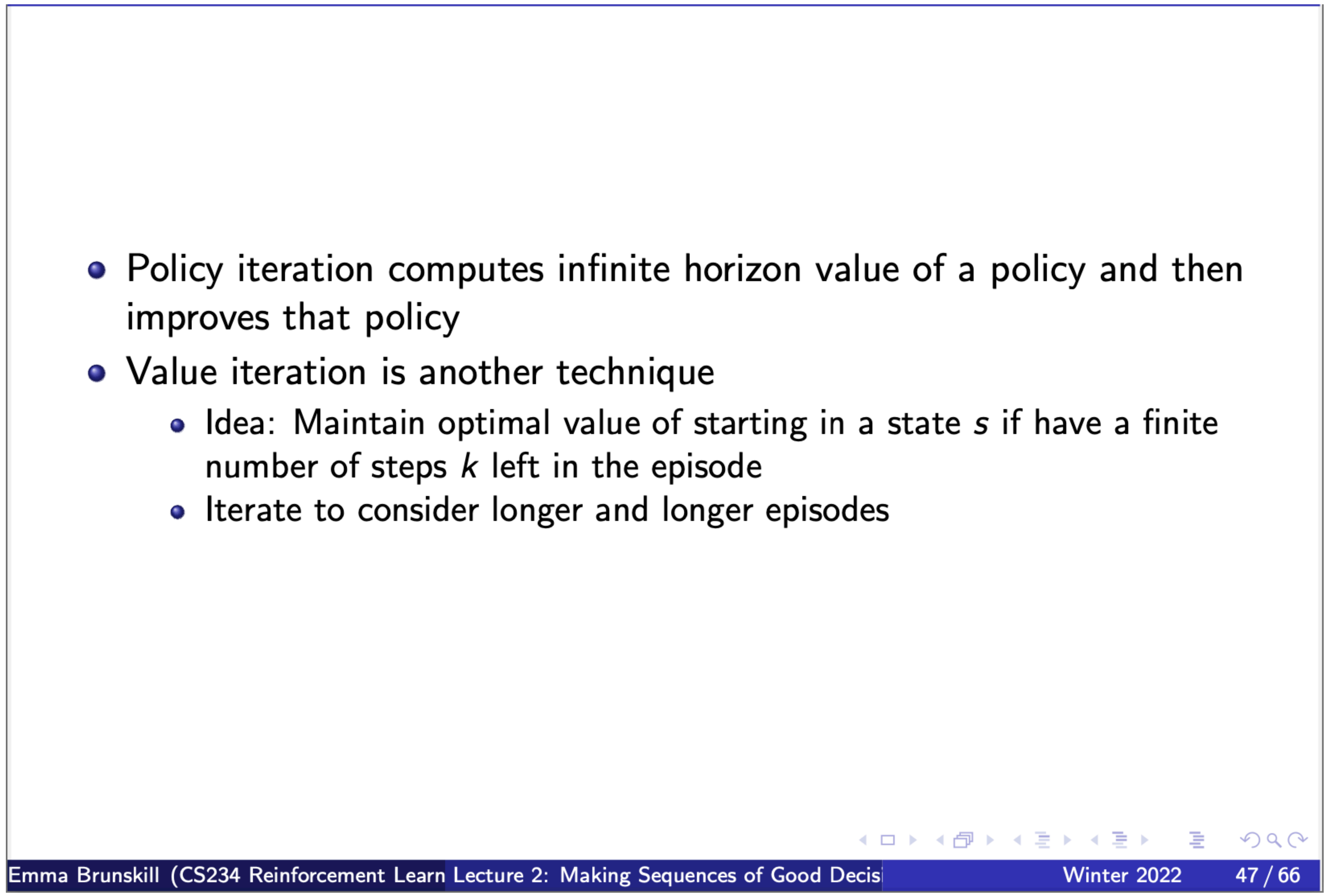
```
$i = 0$
**Initialize $\pi_0$ randomly.**
**DO**
    $V^{\pi_i} =$ Compute the state-value function, evaluating $\pi_i$.
    $\pi_{i+1} =$ Policy improvement of $\pi_i$.
    $i = i + 1$
**WHILE** $||\pi_i - \pi_{i-1}||_1 > 0$ /* if policy changed */
```

Policy iteration finds the globally optimal policy!

Value Iteration

- Value iteration is another way to find the optimal policy.

- Instead of searching for the optimal policy as before (i.e. $\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$), we will be looking directly for the optimal value function: $V^*(s) = \max_{\pi} V^{\pi}(s)$.



- Policy iteration computes infinite horizon value of a policy and then improves that policy
- Value iteration is another technique
  - Idea: Maintain optimal value of starting in a state s if have a finite number of steps k left in the episode
  - Iterate to consider longer and longer episodes

Emma Brunskill (CS234 Reinforcement Learn Lecture 2: Making Sequences of Good Decis Winter 2022 47 / 66

Value Iteration (Bellman Equation)

- Recall we had:

$$V^{\pi}(s) = \sum_{a \in A} \pi(a, s) \cdot \left[R(s, a) + \gamma \cdot \sum_{s' \in S} P(s' | s, a) \cdot V^{\pi}(s') \right]$$

- But now we do not have a policy, so we will have some V without specifying π (but any such V induces some policy π).
- We can define Bellman backup operator $B(\, . \,)$ (the operator will be applied on functions!):
- **Bellman Backup Operator for Value Function:**
  - Notation: $B[V]$ denotes applying B (Bellman backup).

$$B[V] = \max_{a \in A} \left[R(s, a) + \gamma \cdot \sum_{s' \in S} P(s' | s, a) \cdot V(s') \right]$$

  - $B[V]$ is a new value function, Bellman backup improves the old value function (if not yet optimal).

Value Iteration

Set $k = 1$

Initialize $V_0(s) = 0$ for all $s \in S$

DO:

$$V_k(s) = \max_{a \in A} \left[R(s,a) + \gamma \cdot \sum_{s' \in S} P(s' \mid s,a) \cdot V_{k-1}(s') \right]$$

Bellman backup B[V]

WHILE $\|V_k - V_{k-1}\|_\infty \geq \varepsilon$

- To extract an optimal policy, we can extract a deterministic (not necessarily unique) policy:

$$\pi(s) = \arg\max_{a \in A} \left[R(s,a) + \sum_{s' \in S} P(s' \mid s,a) \cdot V(s') \right].$$

Part 5: Proofs

Outline

1. Why value iteration converges to an optimal value function,
2. Why policy iteration converges to an optimal policy.

A Bit More on Bellman Backup Operators

- This slide is about **terminology** (which is also important, after all, we want to understand others!).
- **Bellman Backup B:**

$$B[V] = \max_{a \in A} \left[R(s,a) + \gamma \cdot \sum_{s' \in S} P(s' \mid s, a) \cdot V(s') \right]$$

- **Bellman Backup B^π for policy evaluation:**

$$B^\pi[V(s)] = R^\pi(s) + \gamma \cdot \sum_{s' \in S} P^\pi(s' \mid s) \cdot V(s')$$

Why Value Iteration and Value Evaluation Converge

- **Definition** (Contractive Operator)**:** An operator $T[\,.\,]$ in a space with norm $||.||$ is a contractive operator if there exists $0 \leq \alpha < 1$ such that, for all V, V', it holds: $||T[V] - T[V']|| \leq \alpha \cdot ||V - V'||$.

- By Banach's Fixed-Point Theorem, we have that any such contractive operator has exactly one fixed point.

- So all we need to do to show that VI and VE converge, is to show that the respective Bellman backup operators $B[\,.\,]$ and $B^{\pi}[\,.\,]$ are contraction operators.

$B[\ .\ ]$ is a contractive operator

Infinity norm: $\|V - V'\| = \max_{s\in S} |V(s) - V'(s)|$.

$$\|B(V) - B(V')\|_\infty = \max_{s\in S}\left|\max_{a\in A}\left(R(s,a) + \gamma\cdot\sum_{s'\in S}P(s'|s,a)\cdot V(s')\right) - \max_{a'\in A}\left(R(s,a') + \gamma\cdot\sum_{s'\in S}P(s'|s,a')\cdot V'(s')\right)\right|$$

$$\leq \max_{s\in S}\left|\max_{a\in A}\left(R(s,a) + \gamma\cdot\sum_{s'\in S}P(s'|s,a)\cdot V(s') - R(s,a) - \gamma\cdot\sum_{s'\in S}P(s'|s,a)\cdot V'(s')\right)\right|$$

$$= \max_{s\in S}\left|\max_{a\in A}\left(\gamma\cdot\sum_{s'\in S}P(s'|s,a)\cdot V(s') - \gamma\cdot\sum_{s'\in S}P(s'|s,a)\cdot V'(s')\right)\right| = \max_{s\in S}\left|\max_{a\in A}\left(\gamma\cdot\sum_{s'\in S}P(s'|s,a)\cdot\left(V(s') - V'(s')\right)\right)\right|$$

$$\leq \max_{s\in S}\max_{a\in A}\left(\gamma\cdot\sum_{s'\in S}P(s'|s,a)\cdot\left|V(s') - V'(s')\right|\right) \leq \max_{s\in S}\max_{a\in A}\left(\gamma\cdot\sum_{s'\in S}P(s'|s,a)\cdot\max_{s''\in S}\left|V(s'') - V'(s'')\right|\right)$$

$$\leq \gamma\cdot\max_{s''\in S}|V(s'') - V'(s'')| = \gamma\cdot\|V - V'\|_\infty.$$

So Value Iteration Converges…

- …but does it converge to the right thing (i.e. to the optimal V^*)?

Notation: $B^{(n)}[V] = \underbrace{B[B[\ldots B[V]}_{n-times}\ldots]]]$

Proof (that it does):

Claim 1: $B[V^*] = V^*$.

Claim 2: $\|B^{(n)}[V] - B^{(n)}[V']\|_\infty \leq \gamma^n \cdot \|V - V'\|_\infty$.

Set $V' = V^*$.

Then $\|B^{(n)}[V] - V^*\|_\infty = \|B^{(n)}[V] - B^{(n)}[V^*]\|_\infty \leq \gamma^n \cdot \|V - V'\|_\infty$.

So for $\gamma < 1$, value iteration converges to V^* from any initialization V.

Now the Same for Value Evaluation…. ($B^\pi[\ .\ ]$ is a contractive operator)

$$\|B^\pi(V) - B^\pi(V')\|_\infty = \max_{s \in S} \left| R^\pi(s) + \gamma \cdot \sum_{s' \in S} P^\pi(s' \mid s) \cdot V(s') - R^\pi(s) - \gamma \cdot \sum_{s' \in S} P^\pi(s' \mid s) \cdot V'(s') \right|$$

$$= \max_{s \in S} \left| \gamma \cdot \sum_{s' \in S} P^\pi(s' \mid s) \cdot V(s') - \gamma \cdot \sum_{s' \in S} P^\pi(s' \mid s) \cdot V'(s') \right| = \gamma \cdot \max_{s \in S} \left| \sum_{s' \in S} P^\pi(s' \mid s) \cdot V(s') - \sum_{s' \in S} P^\pi(s' \mid s) \cdot V'(s') \right|$$

$$= \gamma \cdot \max_{s \in S} \left| \sum_{s' \in S} P^\pi(s' \mid s) \cdot V(s') - \sum_{s' \in S} P^\pi(s' \mid s) \cdot V'(s') \right| = \gamma \cdot \max_{s \in S} \left| \sum_{s' \in S} P^\pi(s' \mid s) \cdot \left(V(s') - V'(s') \right) \right|$$

$$\leq \gamma \cdot \sum_{s' \in S} P^\pi(s' \mid s) \cdot \max_{s \in S} \left| V(s') - V'(s') \right| = \gamma \cdot \max_{s \in S} \left| V(s') - V'(s') \right| \leq \gamma \cdot \|V - V'\|_\infty .$$

The rest of the proof is completely analogical to the proof for value iteration…

Recall: Policy Iteration

```
$i = 0$
**Initialize $\pi_0$ randomly.**
**DO**
    $V^{\pi_i} =$ Compute the state-value function, evaluating $\pi_i$.
    $\pi_{i+1} =$ Policy improvement of $\pi_i$.
    $i = i + 1$
**WHILE** $||\pi_i - \pi_{i-1}||_1 > 0$ /* if policy changed */
```

Policy iteration finds the globally optimal policy!

Why It Works

Note that:

$$V^{\pi_i}(s) \leq \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \cdot V^{\pi_i}(s') \right] = \max_{a \in A} Q^{\pi_i}(s,a)$$

We have

$$\begin{aligned} V^{\pi_i}(s) &\leq R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s' \mid s, \pi_{i+1}(s)) \cdot V^{\pi_i}(s') \\ &\leq R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s' \mid s, \pi_{i+1}(s)) \cdot \max_{a \in A} Q^{\pi_i}(s',a) \\ &\leq R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s' \mid s, \pi_{i+1}(s)) \cdot \left[R(s',\pi_{i+1}(s')) + \gamma \sum_{s'' \in S} P(s'' \mid s', \pi_{i+1}(s')) \cdot V^{\pi_i}(s'') \right] \\ &\quad \vdots \quad \textbf{(keep repeating...)} \\ &\leq V^{\pi_{i+1}}(s) \end{aligned}$$

Next Lecture…

- A bit more about MDPs with finite horizons
- Starting reinforcement learning (right now we have the MDP, in RL we will not have it and yet we will try to learn to act optimally!)

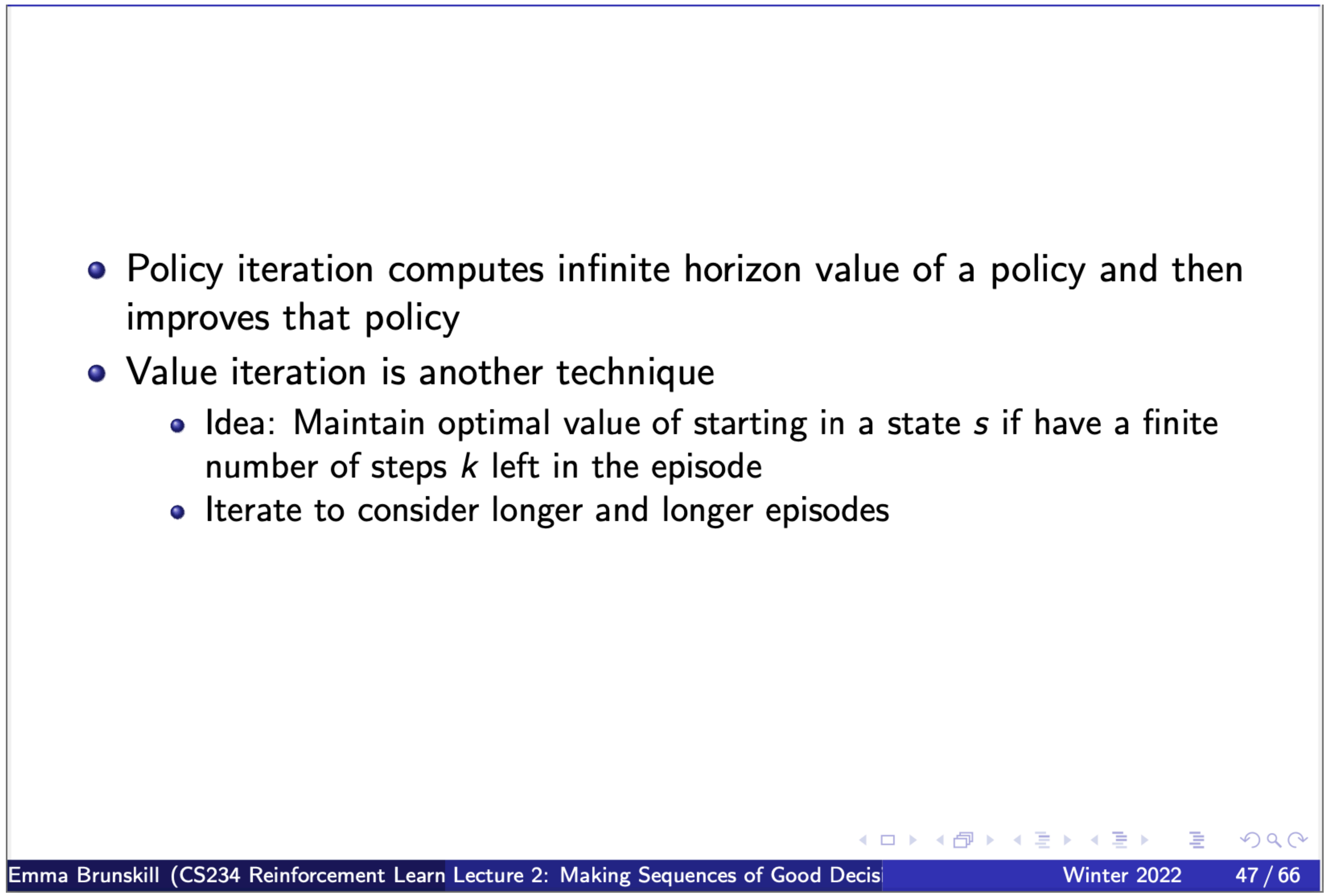
A Bit More About Finite Horizon’s

Non-Stationarity

- One complication with finite horizons is that optimal policies may be non-stationary, which means that the optimal action to take in a state $s \in S$ may depend on the number of time steps remaining until the end of the episode.

Value Iteration for Finite Horizon (1/2)

- Value iteration works also for finite horizons. Recall this slide from Prof. Emma Brunskill

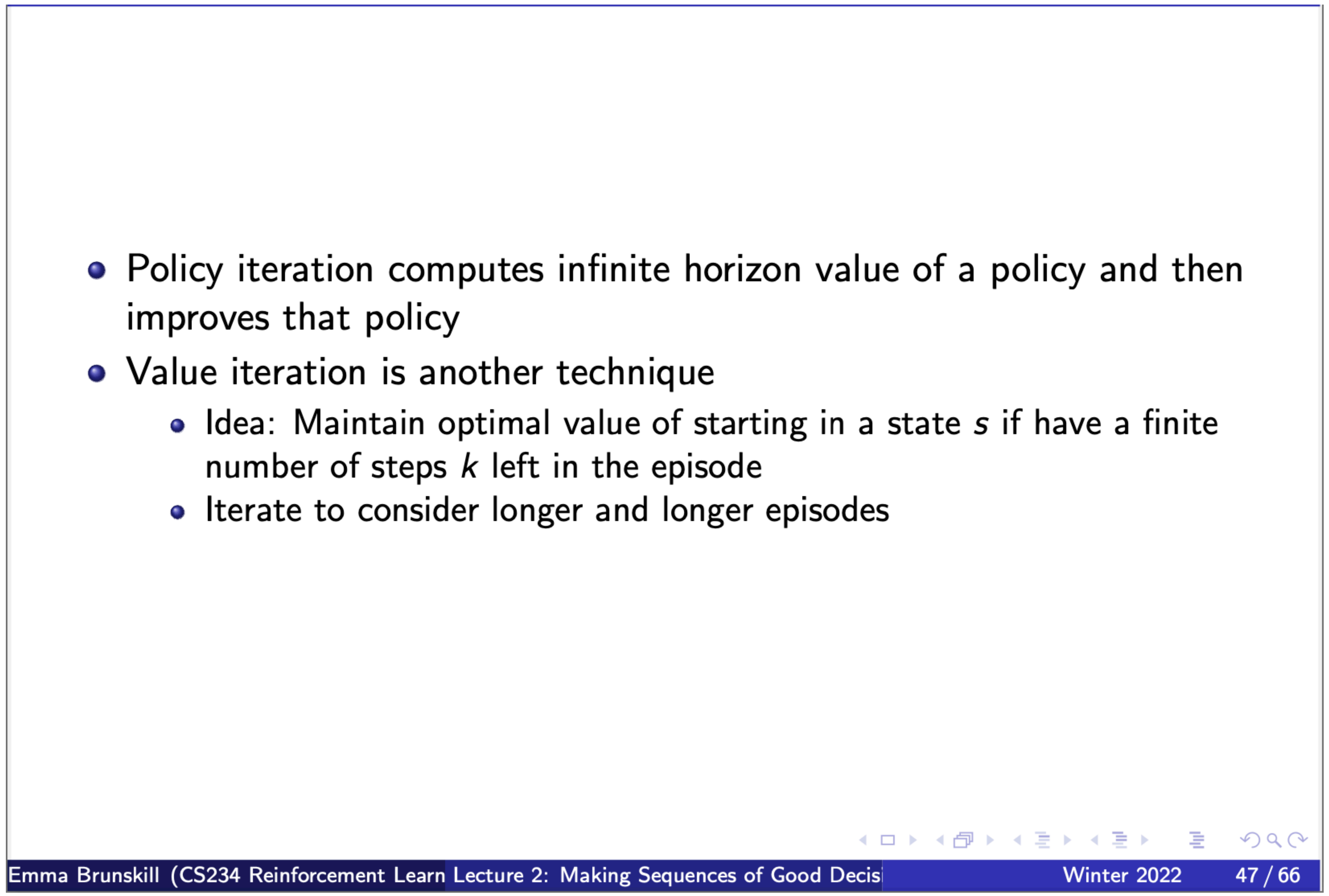


- Policy iteration computes infinite horizon value of a policy and then improves that policy
- Value iteration is another technique
  - Idea: Maintain optimal value of starting in a state s if have a finite number of steps k left in the episode
  - Iterate to consider longer and longer episodes

Emma Brunskill (CS234 Reinforcement Learn Lecture 2: Making Sequences of Good Decis Winter 2022 47 / 66

Value Iteration for Finite Horizon (1/2)

- Value iteration works also for finite horizons. Recall this slide from Prof. Emma Brunskill



- Policy iteration computes infinite horizon value of a policy and then improves that policy
- Value iteration is another technique
  - Idea: Maintain optimal value of starting in a state s if have a finite number of steps k left in the episode
  - Iterate to consider longer and longer episodes

Emma Brunskill (CS234 Reinforcement Learn Lecture 2: Making Sequences of Good Decis Winter 2022 47 / 66

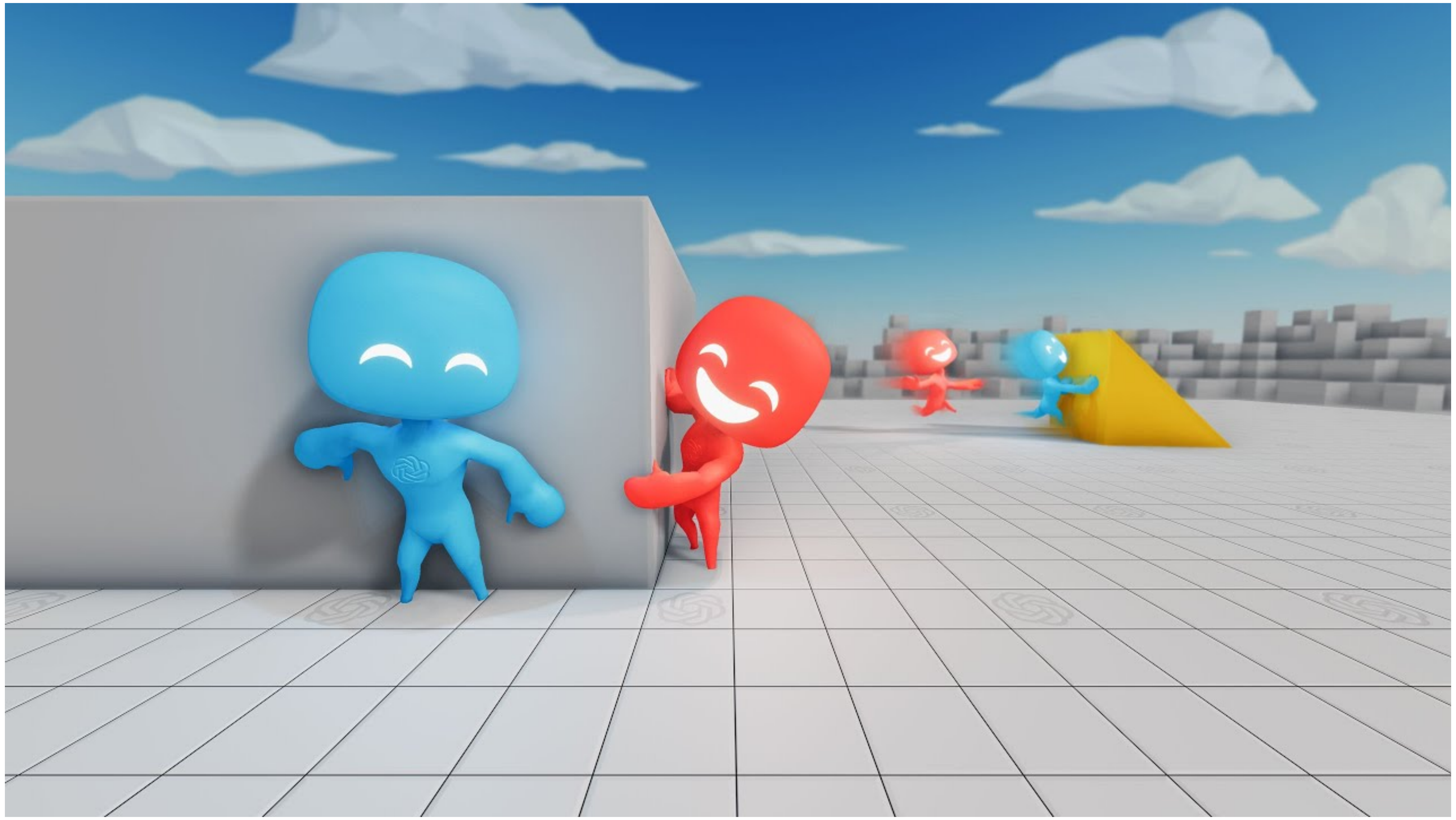
Reinforcement Learning (RL)

- RL: Learning to make sequences of decisions to maximize rewards.

- **This lecture:**
  - Motivation
  - Review of Markov Decision Processes

Some Cool Applications

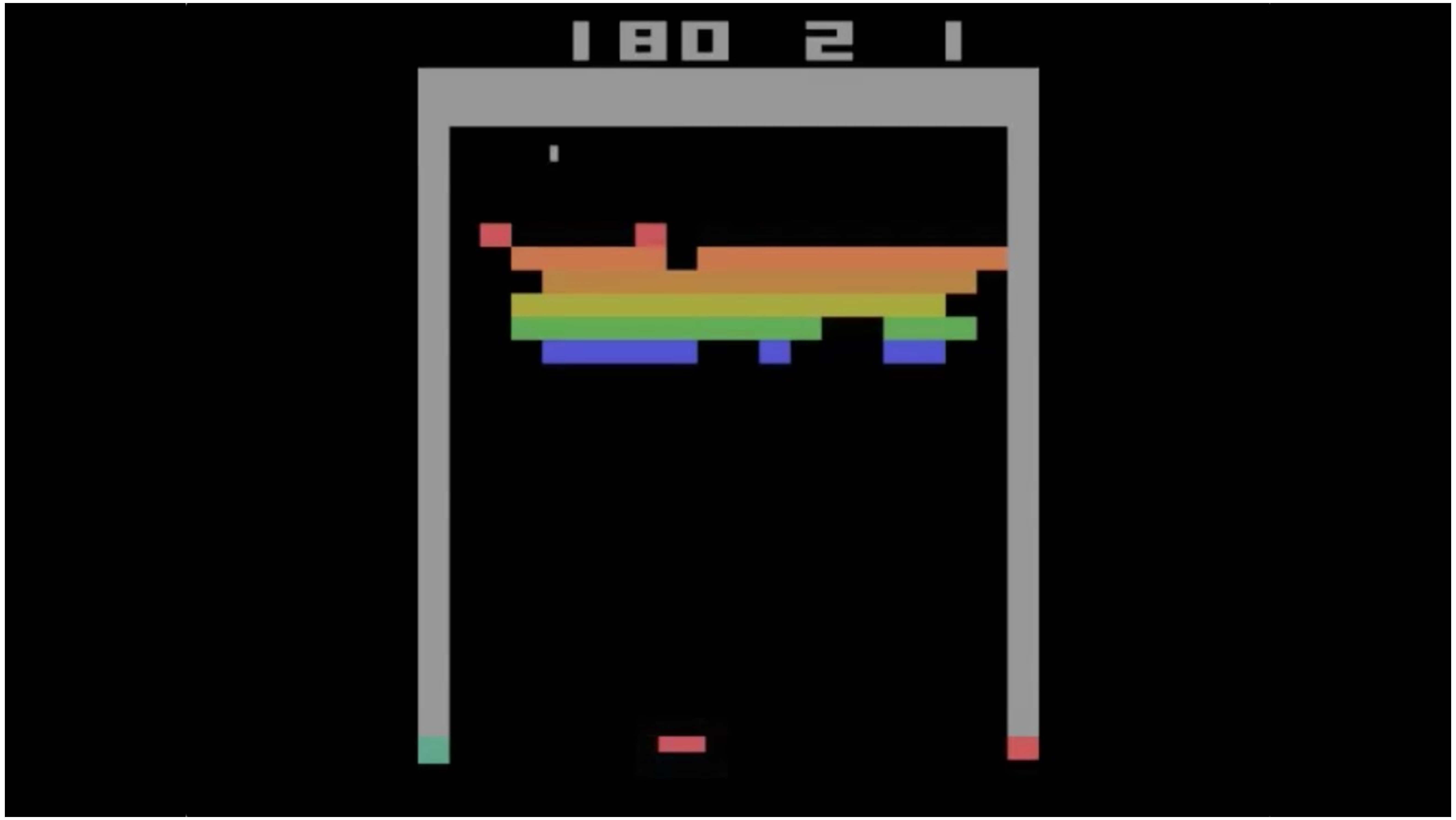
OpenAI's Hide and Seek



Paper: Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew, Igor Mordatch: Emergent Tool Use From Multi-Agent Autocurricula. ICLR 2020

Video: https://www.youtube.com/embed/kopoLzvh5jY

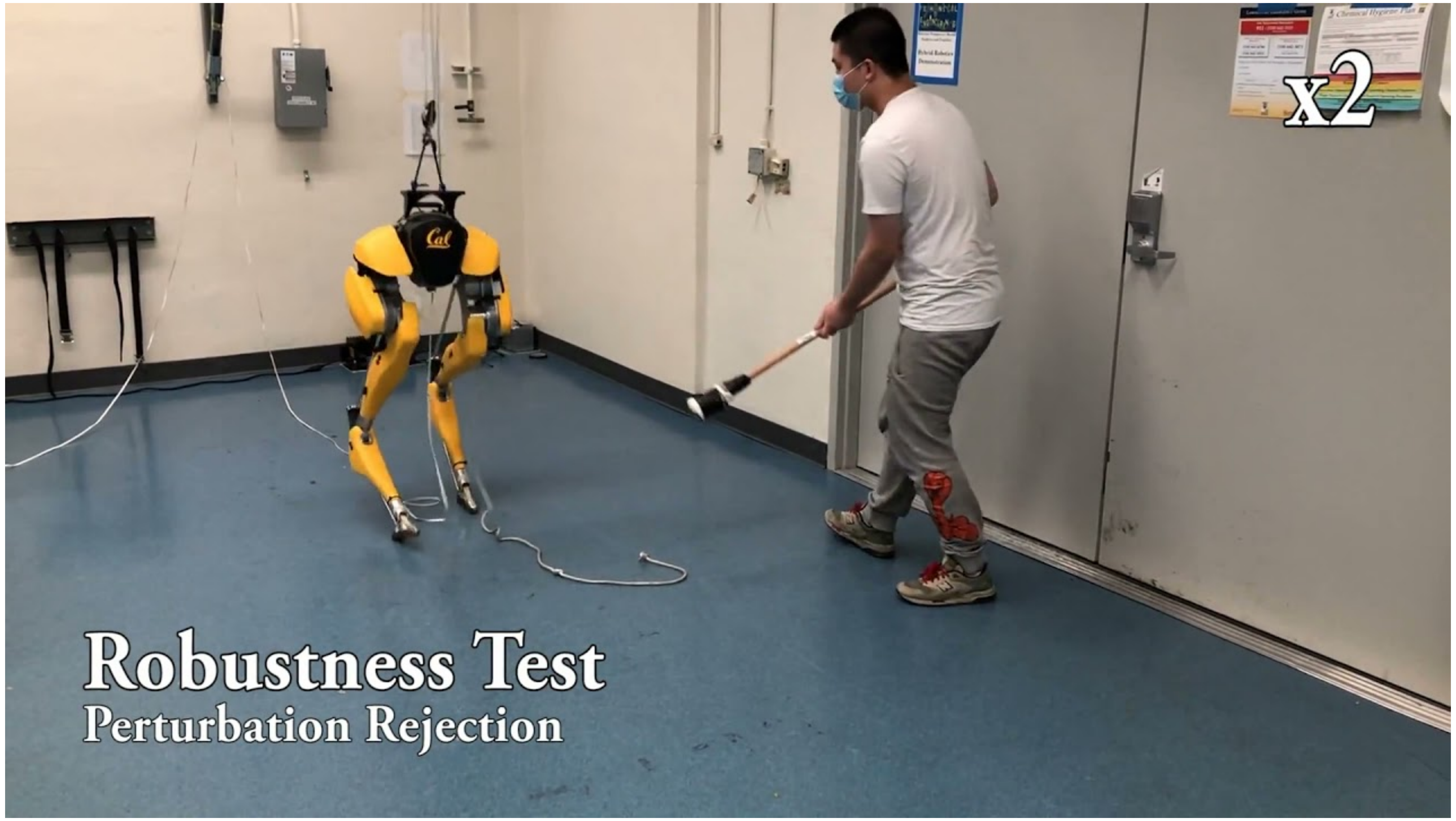
DeepMind's Atari Games



Paper: Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529-533.

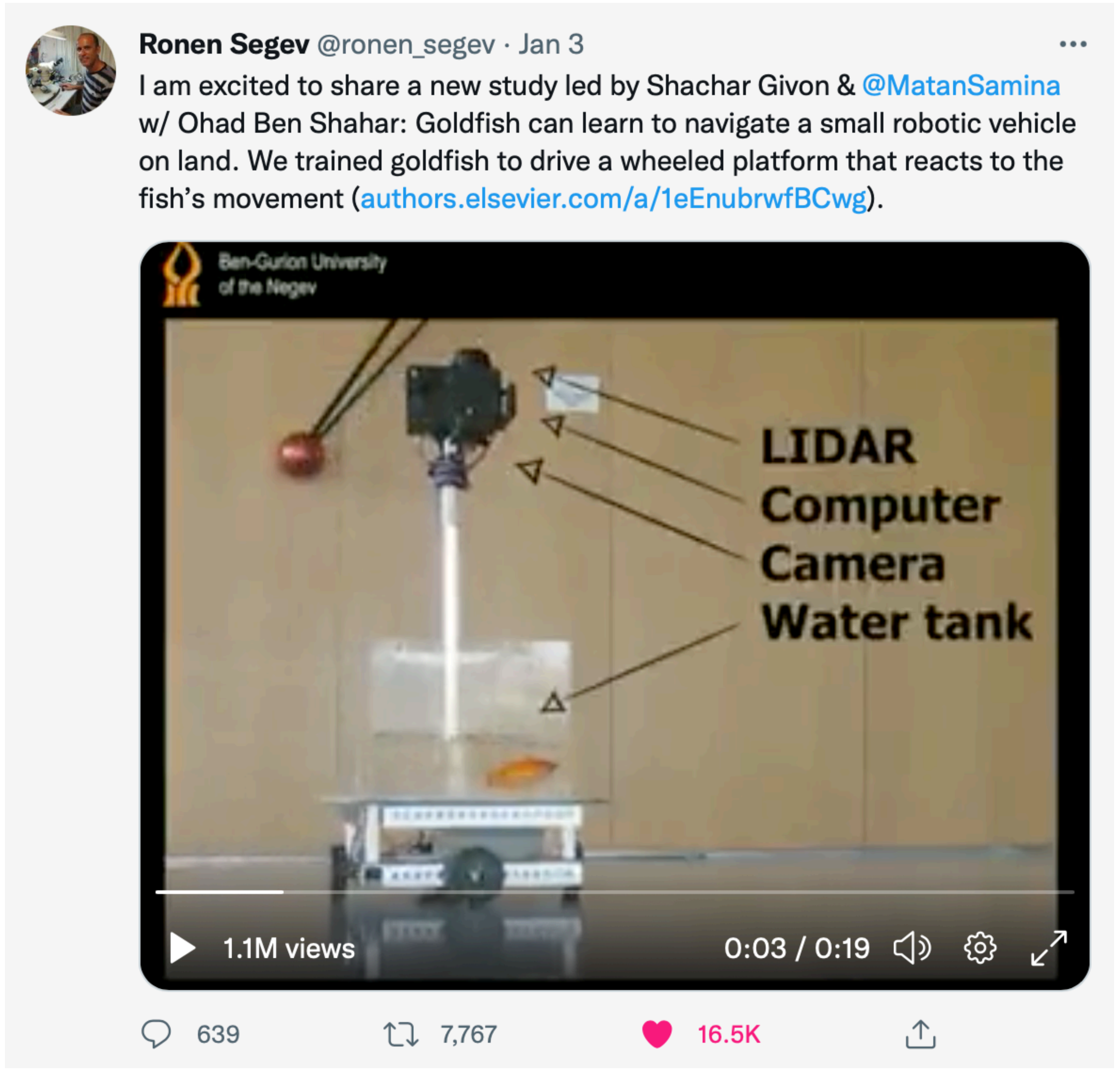
Video: https://www.youtube.com/watch?v=TmPfTpjtdgg

Robots Learning to Walk



Article: https://www.technologyreview.com/2021/04/08/1022176/boston-dynamics-cassie-robot-walk-reinforcement-learning-ai/
Video: https://www.youtube.com/watch?v=goxCjGPQH7U&t=52s

Even Goldfish Can Do Some Interesting Learning and Generalize 🐡



Ronen Segev @ronen_segev · Jan 3

I am excited to share a new study led by Shachar Givon & @MatanSamina w/ Ohad Ben Shahar: Goldfish can learn to navigate a small robotic vehicle on land. We trained goldfish to drive a wheeled platform that reacts to the fish's movement (authors.elsevier.com/a/1eEnubrwfBCwg).

639 7,767 16.5K

https://www.sciencedirect.com/science/article/pii/S0166432821005994