Markov Decision Processes

Machine Learning

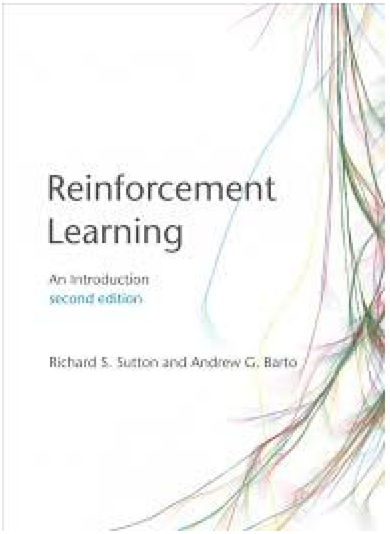
Daniele Loiacono

POLITECNICO
MILANO 1863

Outline and References

- Outline
  - Agent-Environment Interface
  - Markov Decision Process
  - Policy
  - Value Functions
  - Optimality

- References
  - Reinforcement Learning: An Introduction [RL Chapter 3]
  - Fundamentals of Reinforcement Learning (Coursera)



Reinforcement Learning vs Supervised Learning

- In supervised learning we train a model by providing it with the correct output

Q Name of the company that developed ChatGPT

A OpenAI

- In reinforcement learning we train a model by providing it with an evaluation of its output

Q Write a 140 chars tweet about ChatGPT

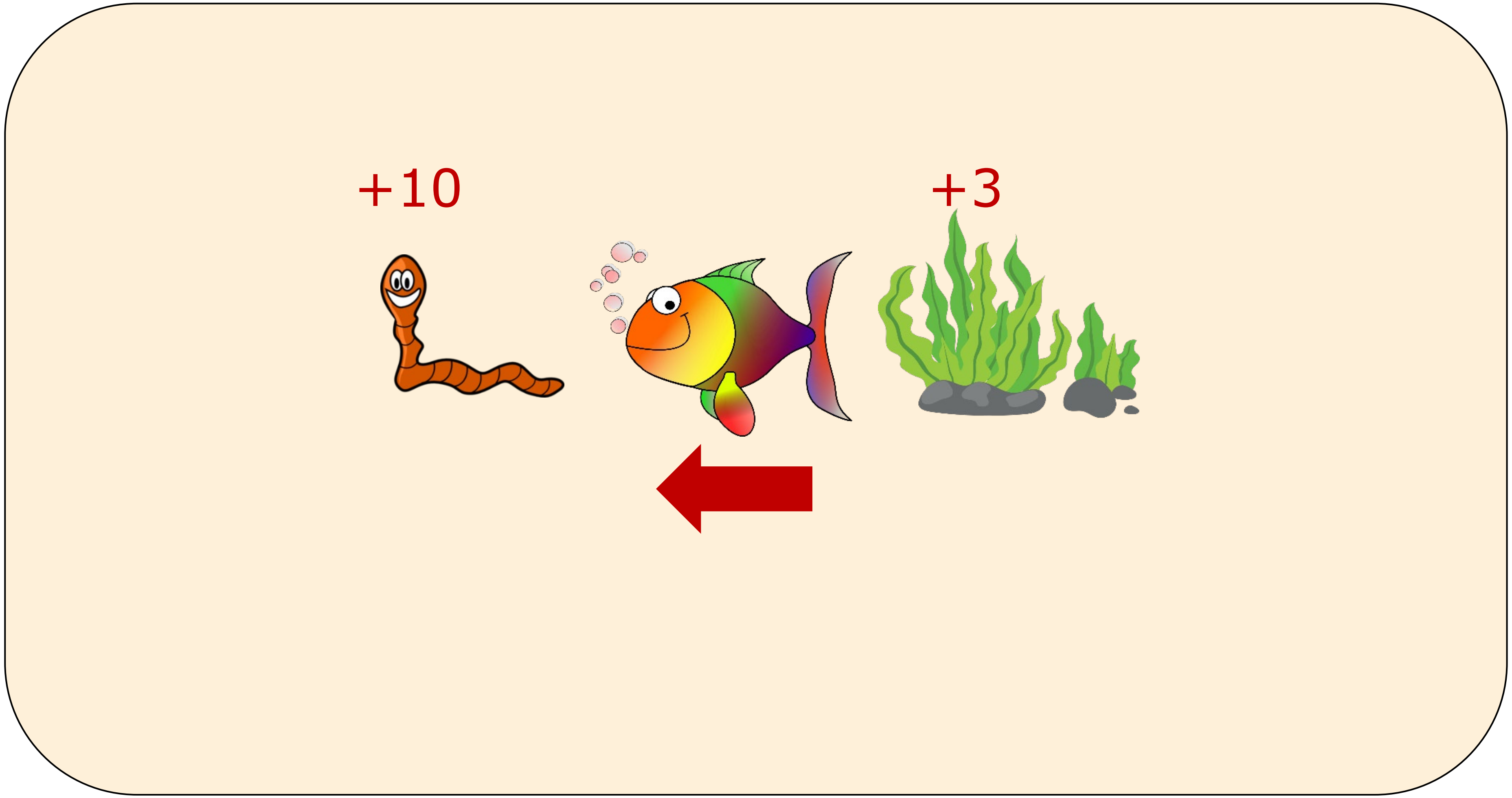
Experience ChatGPT by @OpenAI🚀! Harness AI for human-like conversations, virtual assistance, content creation, and beyond. Join the future!💬🤖 #ChatGPT #OpenAI

Agent-Environment Interface

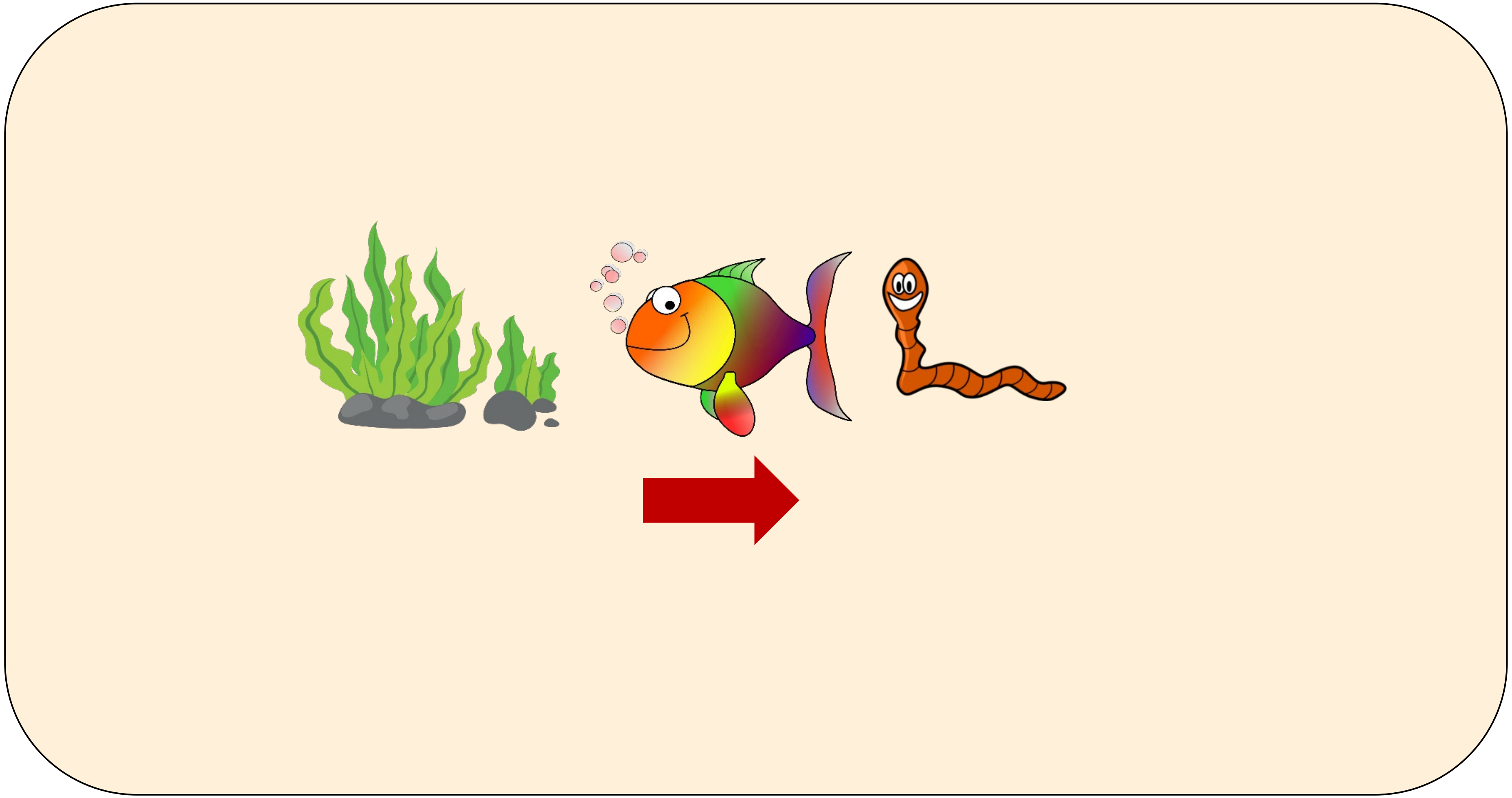
Sequential Decision Making

- In sequential decision making...
  - ... we take a **sequence** of decisions (or **actions**) to reach the **goal**
  - ... the optimal actions are **context-dependent**
  - ... we generally do not have **examples** of correct actions
  - ... actions may have **long-term** consequences
  - ... **short-term consequences** of optimal actions might seem negative

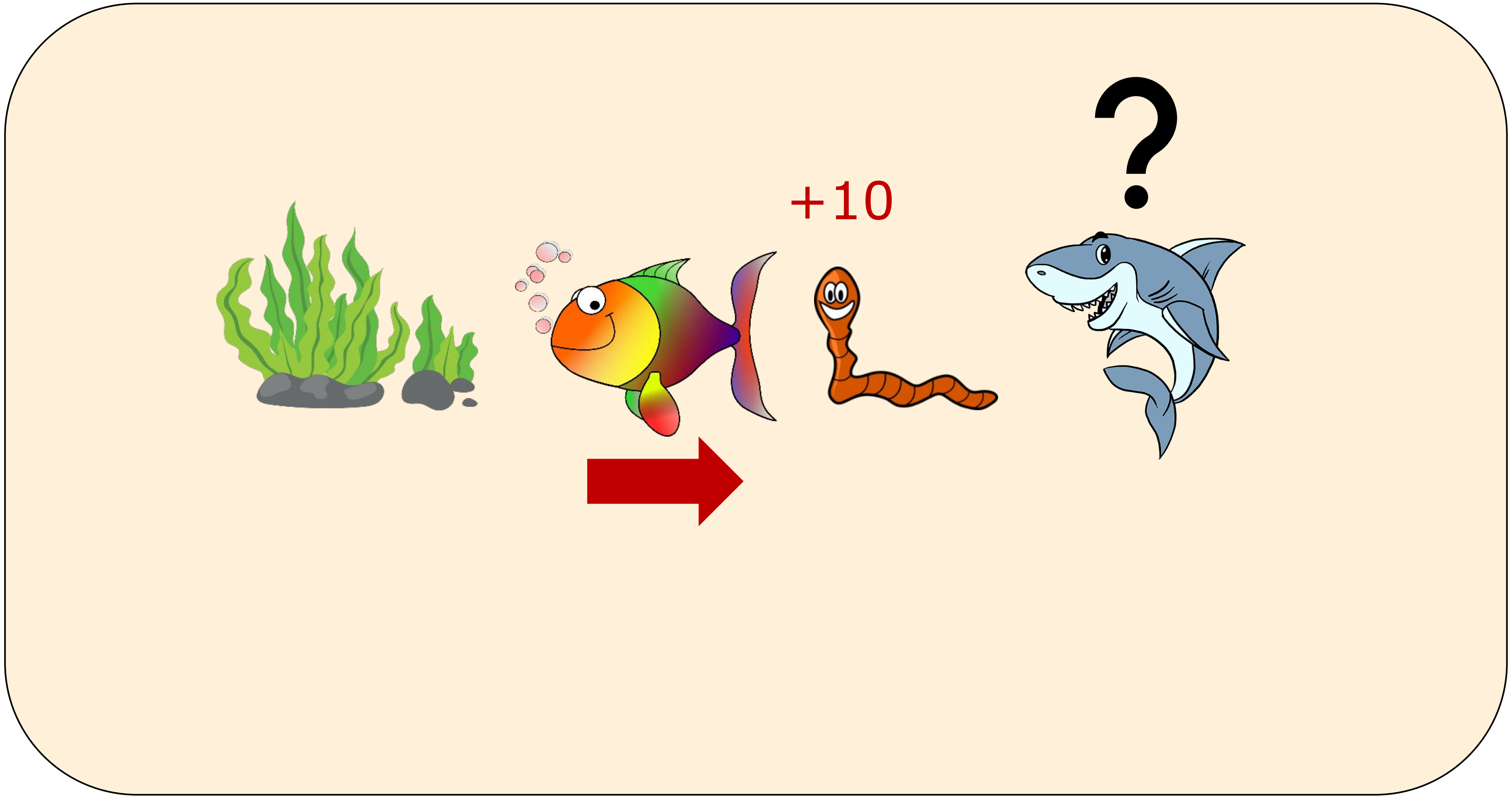
Sequential Decision Making



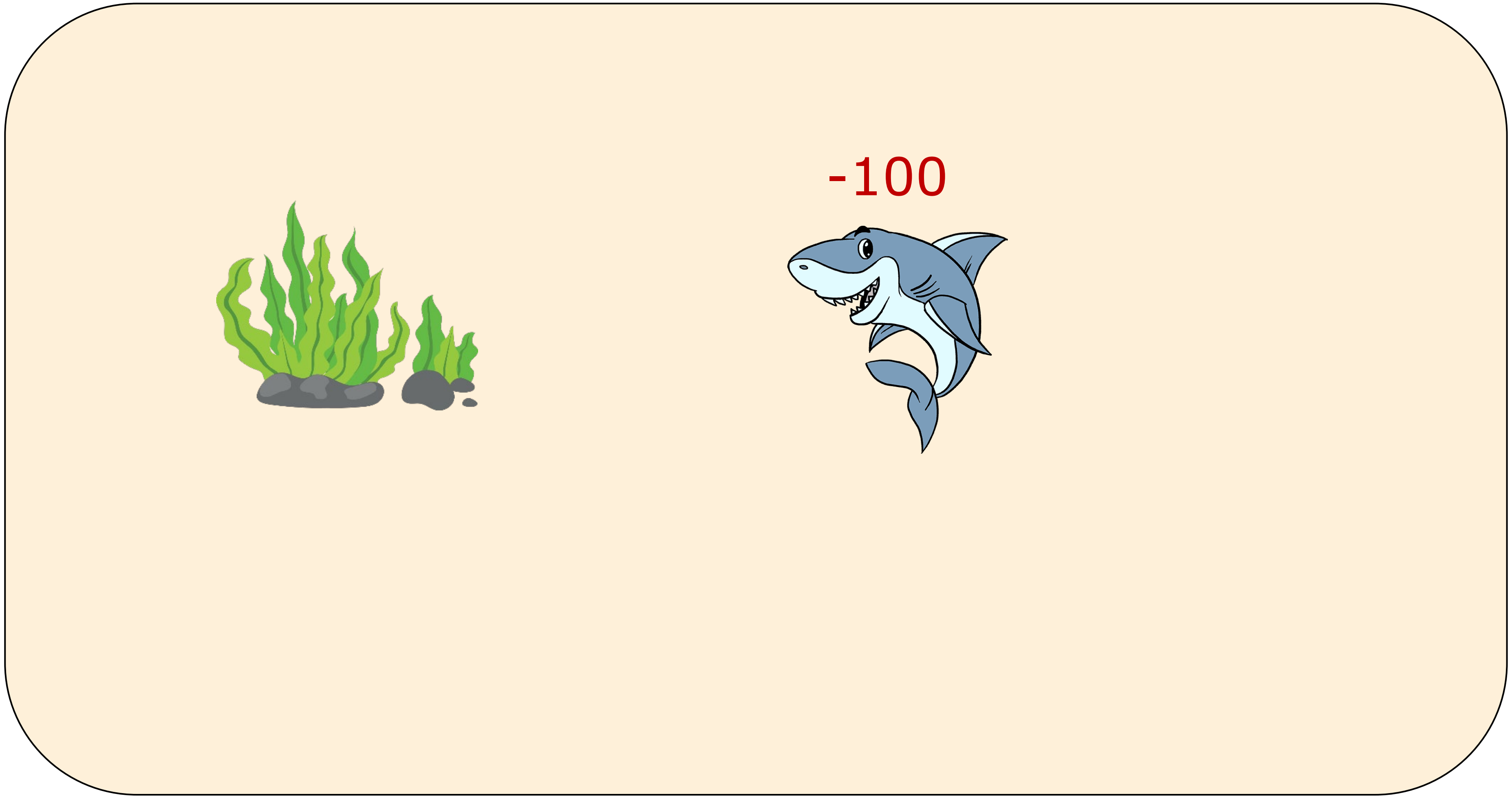
Sequential Decision Making



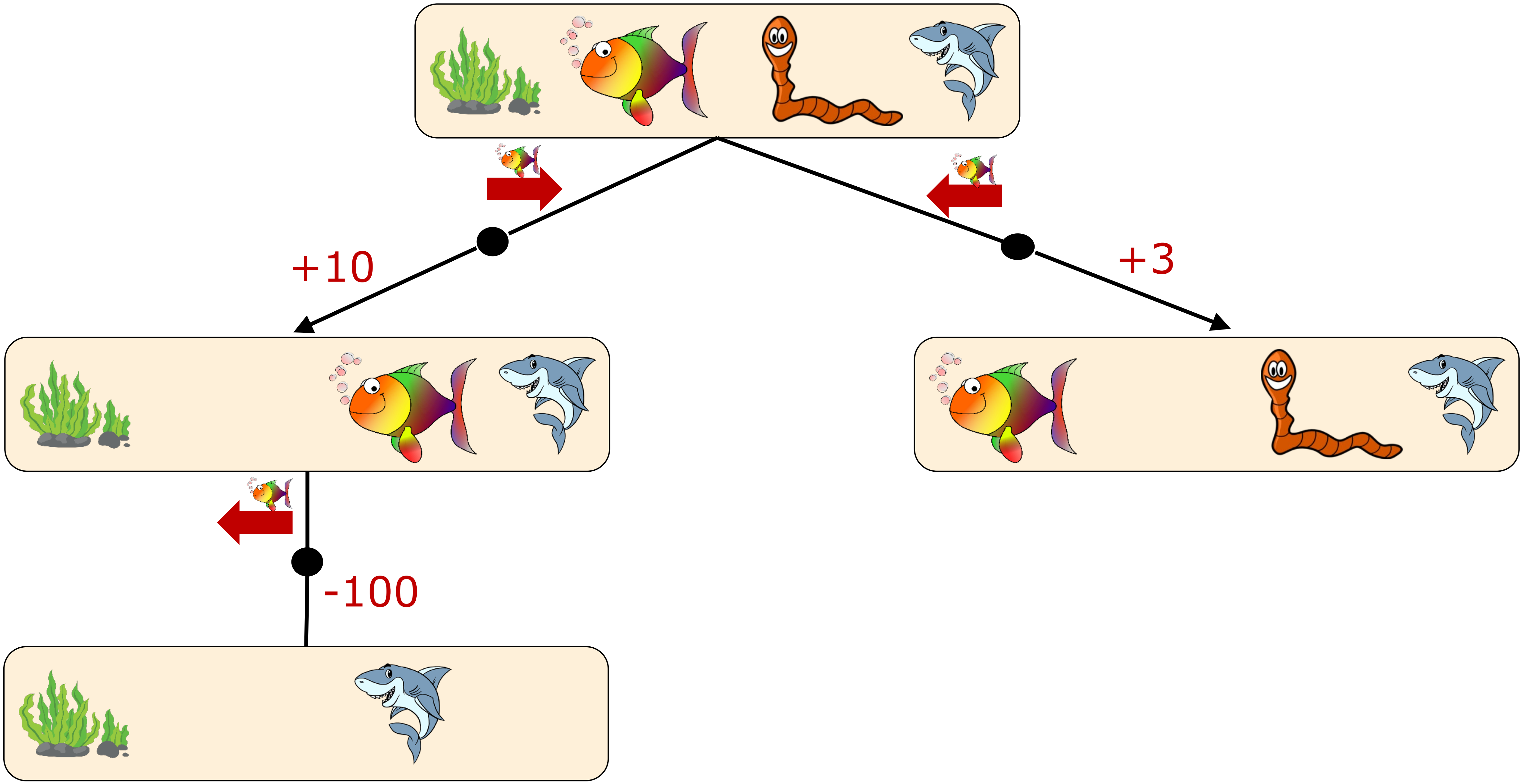
Sequential Decision Making



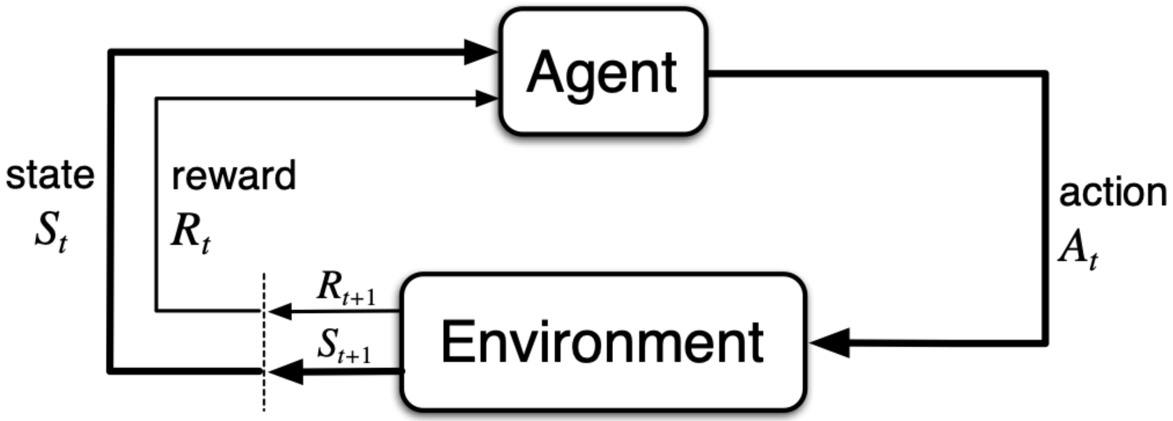
Sequential Decision Making



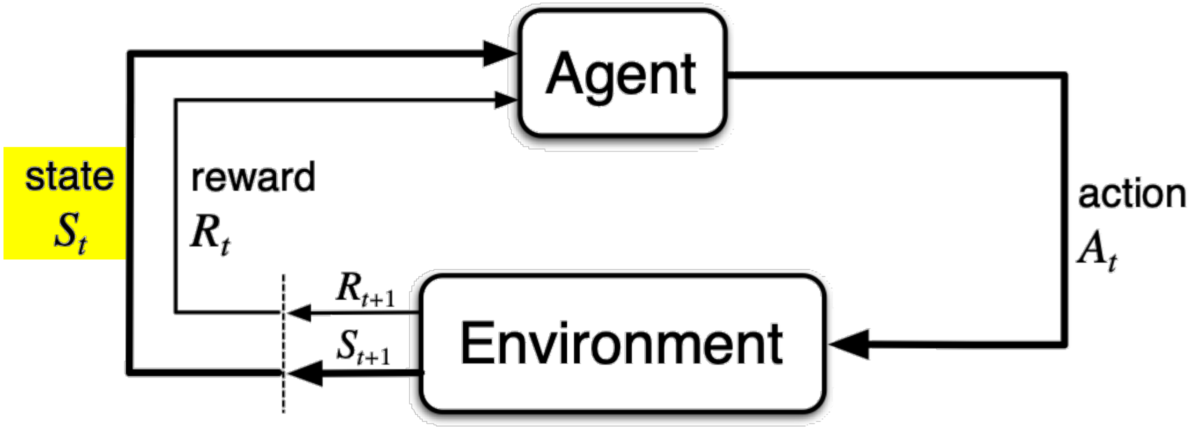
Sequential Decision Making

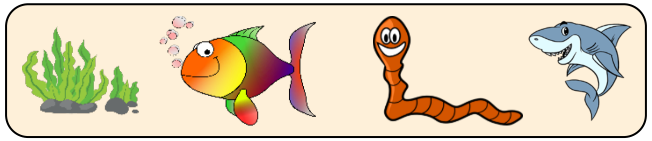


The Agent-Environment Interface

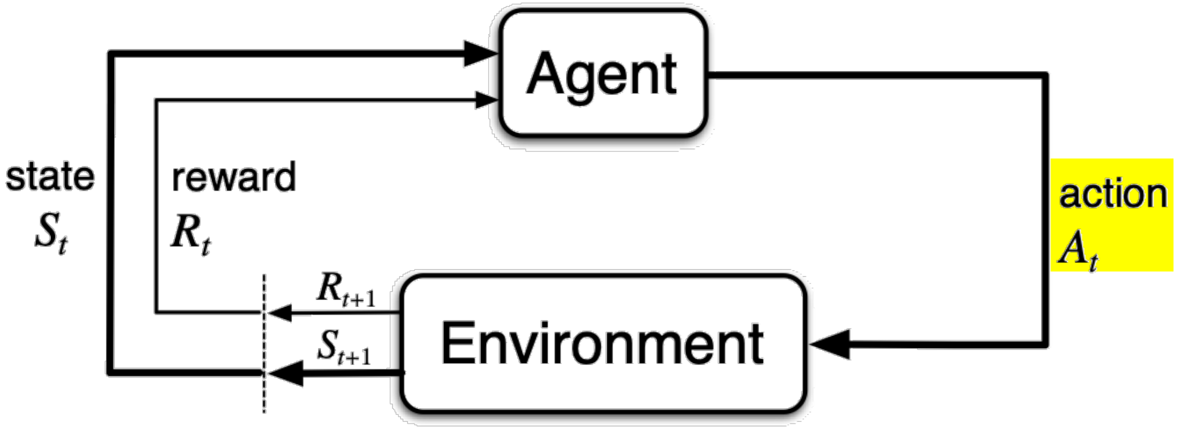


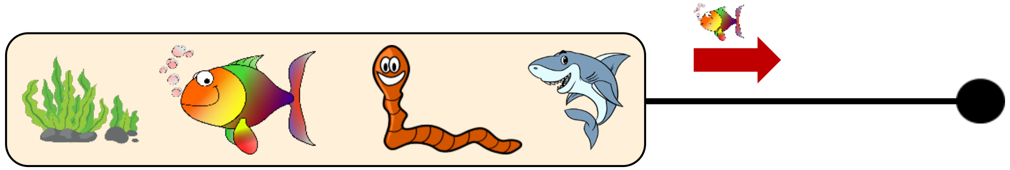
The Agent-Environment Interface



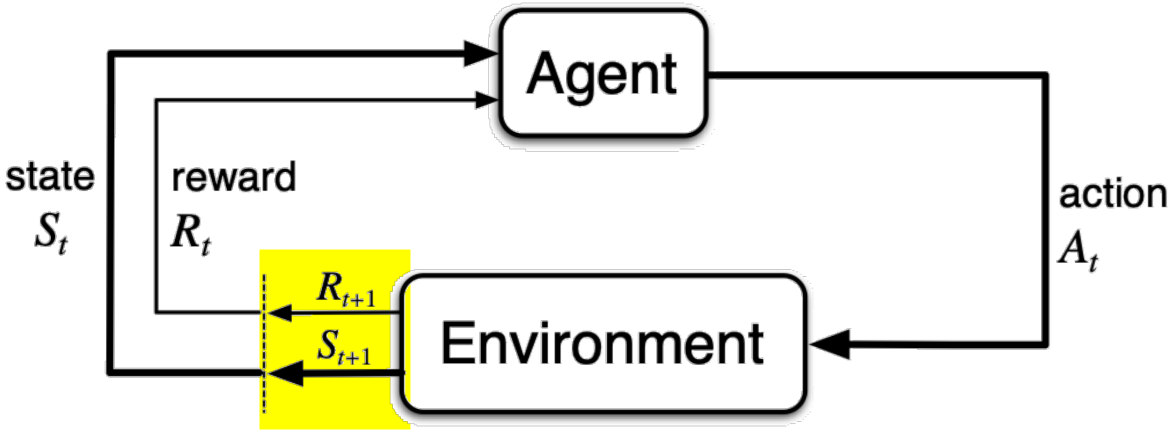


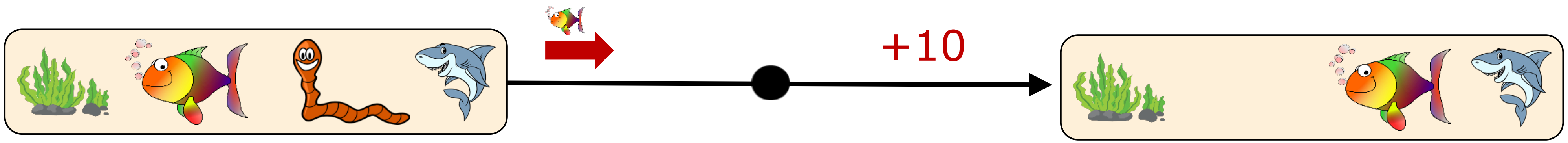
The Agent-Environment Interface



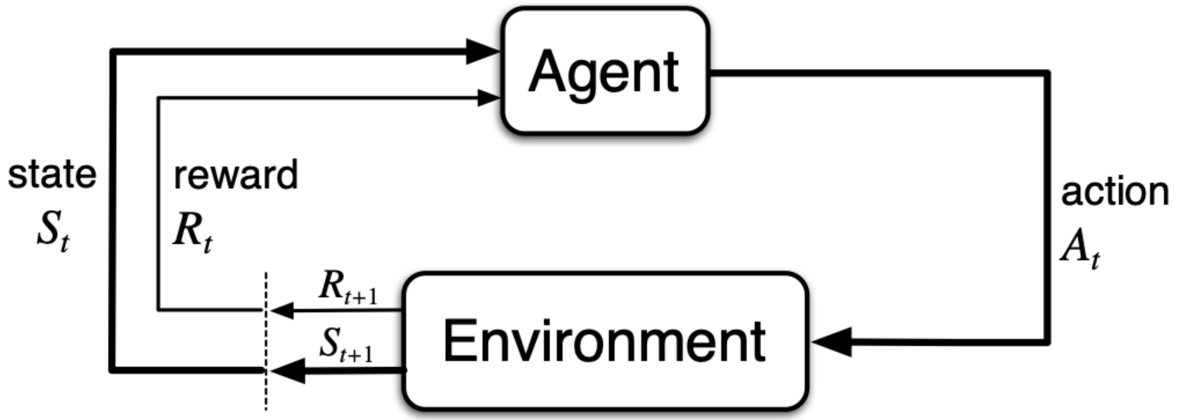


The Agent-Environment Interface





The Agent-Environment Interface



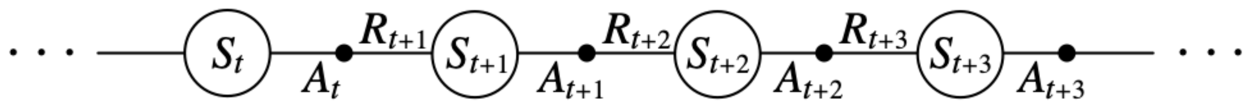
Agent and environment interact at discrete time steps: $t = 0, 1, 2, K$

Agent observes state at step t: $S_t \in \mathcal{S}$

produces action at step t: $A_t \in \mathcal{A}(S_t)$

gets resulting reward: $R_{t+1} \in \mathcal{R}$

and resulting next state: $S_{t+1} \in \mathcal{S}$



Markov Decision Process

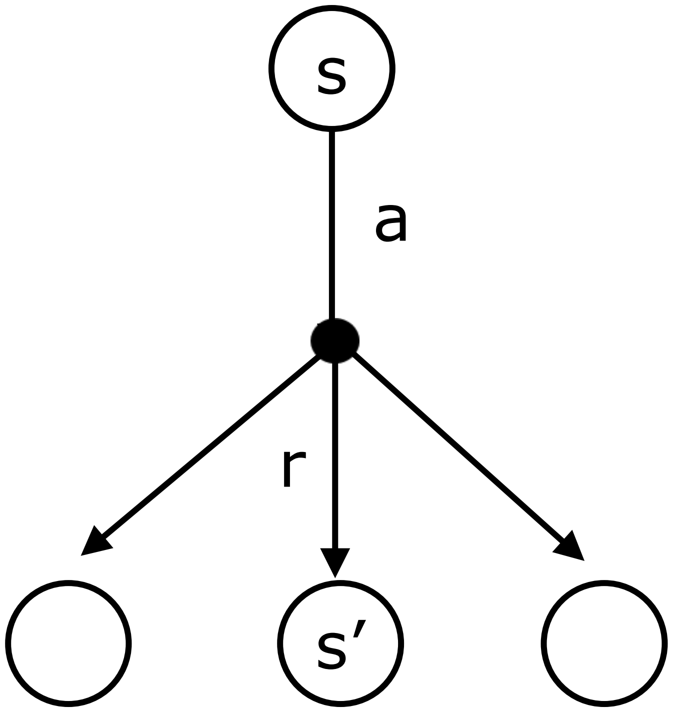
Markov Decision Process: One-Step Dynamics

- **Markov Property:** future state (s') and reward (r) only depend on current state (s) and action (a)
  - It is not a limiting assumption, it can be seen as a property of state

- In a Markov Decision Process (MDP), the one-step dynamic can be described as:

$$p(s', r | s, a)$$

  - p: $\mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$
  - $\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1 \quad \forall\, s \in \mathcal{S}, \forall\, a \in \mathcal{A}(s)$



Finite Markov Decision Processes

- When holds the Markov Property and the state and action sets are finite, the problem is a **finite Markov Decision Process**
- To define a finite MDP, you need to define:
  - **state and action sets**
  - one-step dynamics:

$$p(s', r|s, a) = \mathbf{Pr}\{S_{t+1}=s', R_{t+1} = r \mid S_t=s, A_t=a\}$$

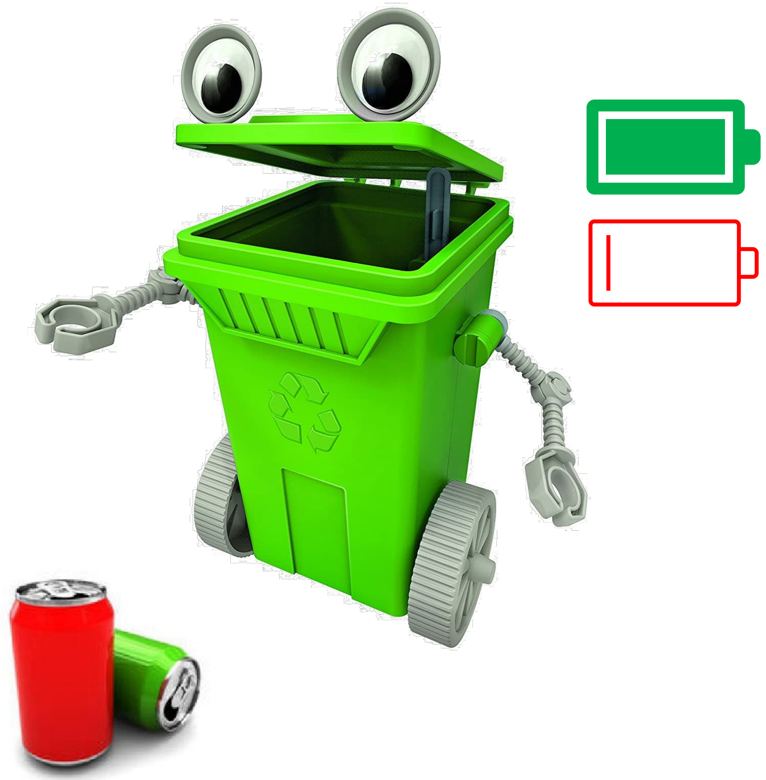
  - we can also derive the next state distribution and expected reward as:

$$p(s'|s, a) \doteq \Pr\{S_{t+1}=s' \mid S_t=s, A_t=a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a)$$

$$r(s, a) \doteq \mathbb{E}[R_{t+1} \mid S_t=s, A_t=a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a)$$

An Example Finite MDP: Recycling Robot

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: `high`, `low`.
- Reward = number of cans collected



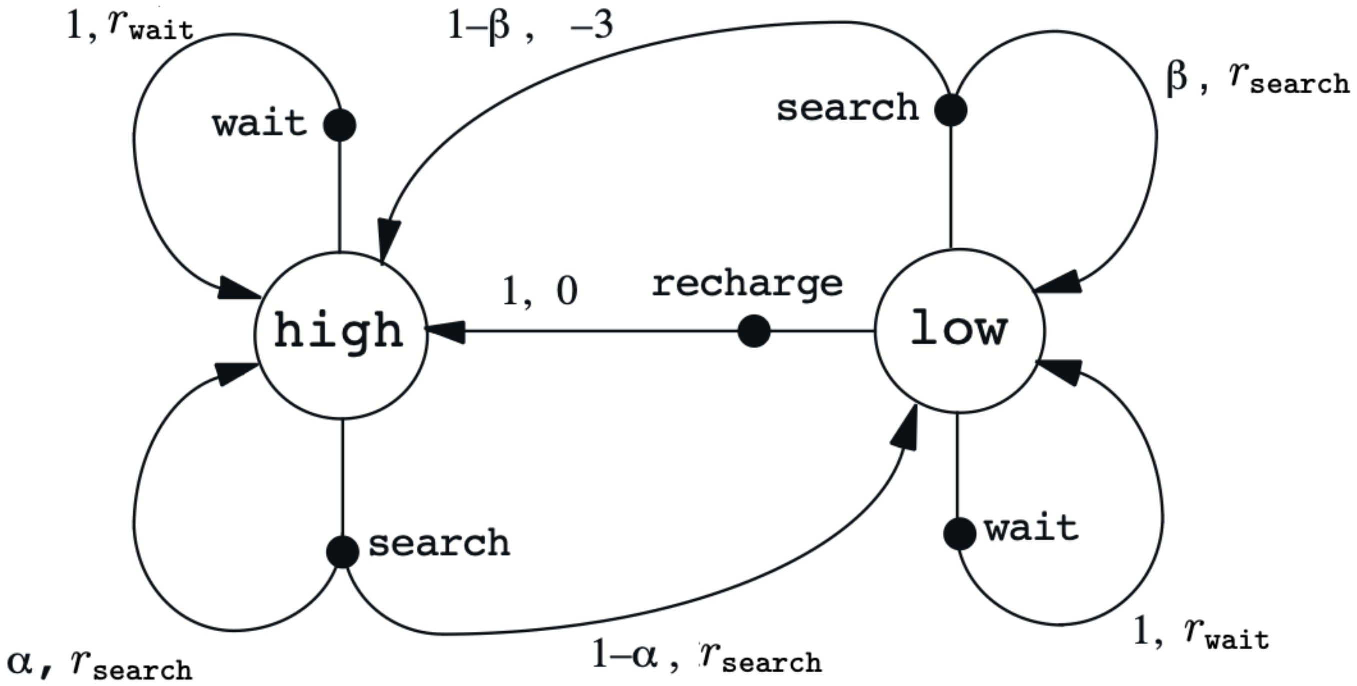
An Example Finite MDP: Recycling Robot

$$\mathcal{S} = \{\texttt{high}, \texttt{low}\}$$
$$\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$$
$$\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$$

$r_{\texttt{search}}$ = expected no. of cans while searching

$r_{\texttt{wait}}$ = expected no. of cans while waiting

$$r_{\texttt{search}} > r_{\texttt{wait}}$$



Return

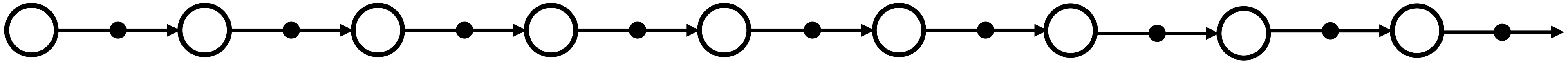
- Agent should not choose actions on the basis of **immediate reward**
- In fact, **long-term** consequences are more important than **short-term** reward
- So, we need to take into account the **sequence of future rewards**
  - We define **return**, G_t, as a function of the sequence of future rewards

$$G_t \doteq f(R_{t+1} + R_{t+2} + R_{t+3} + \dots)$$

  - To succeed, the agent will have to maximize the expected return $\mathbb{E}[G_t]$
- Different definition of return are possible:
  - Total reward
  - Discounted reward
  - Average reward
  - …

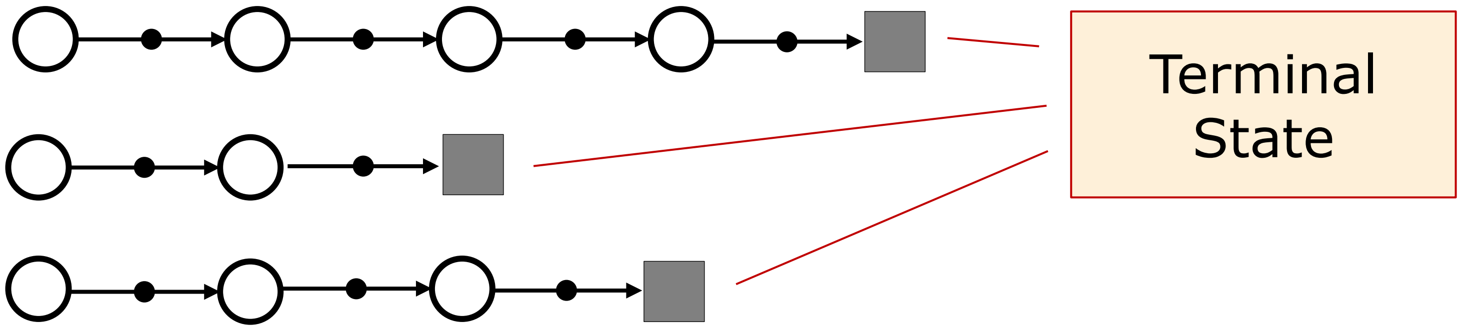
Episodic Task

- In **episodic task** the agent-environment interaction naturally breaks into chunks called **episodes**



Episodic Task

- In **episodic task** the agent-environment interaction naturally breaks into chunks called **episodes**



- It is possible to maximize the expected **total reward**:

$$\mathbb{E}[G_t] = \mathbb{E}[R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T]$$

Final time step

Continuing Tasks

- In **continuing task** the agent-environment interaction goes on **continually** and there are no terminal state
- The total reward is a sum over an **infinite sequence** and might not be finite:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_{t+k} + \ldots \stackrel{?}{=} \infty$$

- To solve this issue we can **discount** the future rewards by a factor γ $(0 < \gamma < 1)$:

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

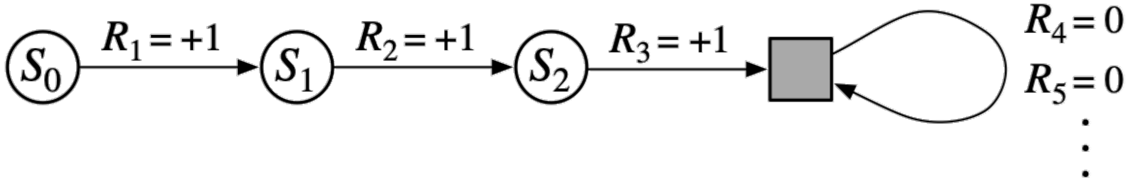
- Thus, the expected reward to maximize will be defined as:

$$\mathbb{E}[G_t] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]$$

$$\leq R_{max} \frac{1}{1-\gamma}$$

Unify Notation for Returns

- In episodic tasks, we number from zero the time steps for each episode
- We can design terminal state as **absorbing states** that always produce zero reward:



- We can use the same definition of expected reward for episodic and continuing tasks:

$$\mathbb{E}[G_t] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]$$

  - where $\gamma = 1$ can be used if an absorbing state is always reached
  - if γ =0, agent would only care about immediate reward
  - As $\gamma \rightarrow 1$, agent would take future rewards into account more strongly

Goal and Reward

- A goal should specify **what** we want to achieve, not **how** we want to achieve it
- **The Reward Hypotesis:** *That all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward).*
- Examples of reward design
  - goal-reward representation: 1 for goal, 0 otherwise
  - action-penalty representation: -1 for not goal, 0 once goal reached
- Challenges to reward hypothesis
  - How to represent risk-sensitive behavior?
  - How to capture diversity in behavior?

Policy

What is a policy?

- A policy, at any given point in time, **decides** which action the agent selects
- A policy fully defines the **behavior** of an agent
- Policies can be:
  - Markovian / Non Markovian
  - Deterministic / Stochastic
  - Stationary / Non Stationary

Deterministic Policy

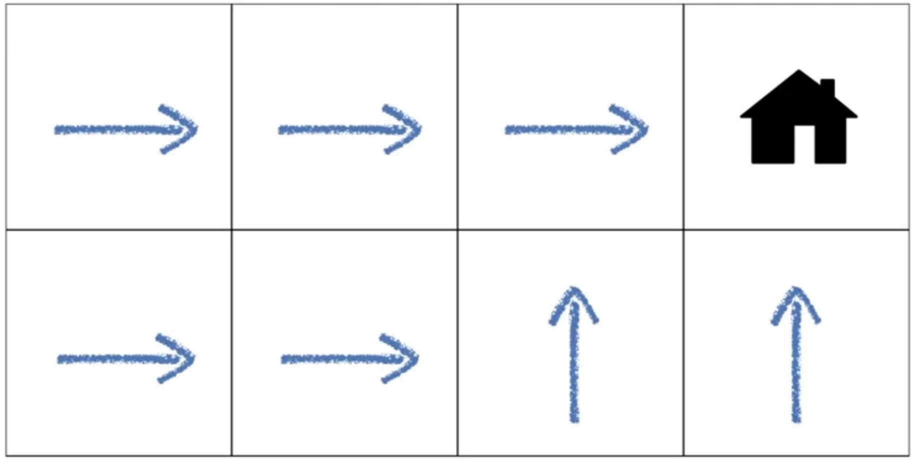
- In the simplest case the policy can be modeled as a function ($\pi: \mathcal{S} \rightarrow \mathcal{A}$):

$$\pi(s) = a$$

- Accordingly, the policy maps each state into an action
- This type of policy can be conveniently represented using a table

State	Action
s_0	a_1
s_1	a_0
s_2	a_0

- This is an example of deterministic policy in a gridworld environment



Stochastic Policy

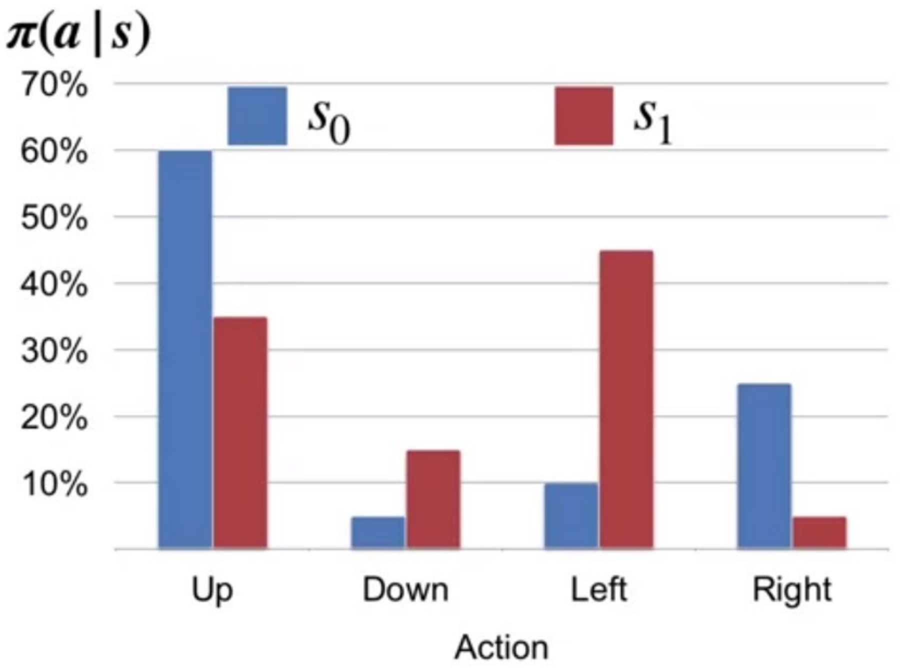
- A more general approach is to model policy as function that maps each state to a probability distribution over the actions:

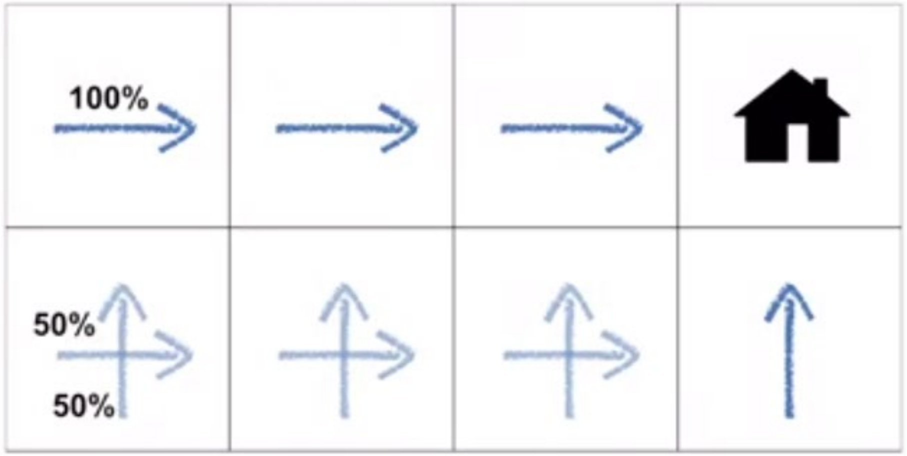
$$\pi(a|s)$$

  - $\sum_{a \in \mathcal{A}(s)} \pi(a|s) = 1$
  - $\pi(a|s) \geq 0$

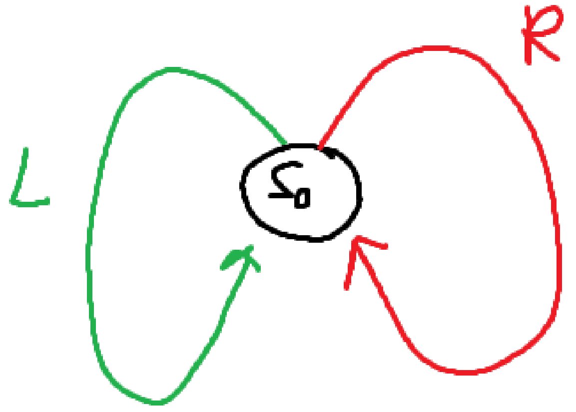
- A stochastic policy can be used to represent also a deterministic policy

- This is an example of stochastic policy in a gridworld environment





Markovian/Non Markovian Policy



π_1: choose 50% R and 50% L

π_2: alternate R and L

- Are π_1 and π_2 both markovian?
- No! π_2 does not depend only from current state, so it is not a valid policy for us!
- However, we can overcome this limitation by extending the state definition (e.g., in this case including previous action)

Value Functions

Value Functions

- Given a policy π we can compute the **state-value function** as:

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

  - it represents the expected return from a given state s, following policy π

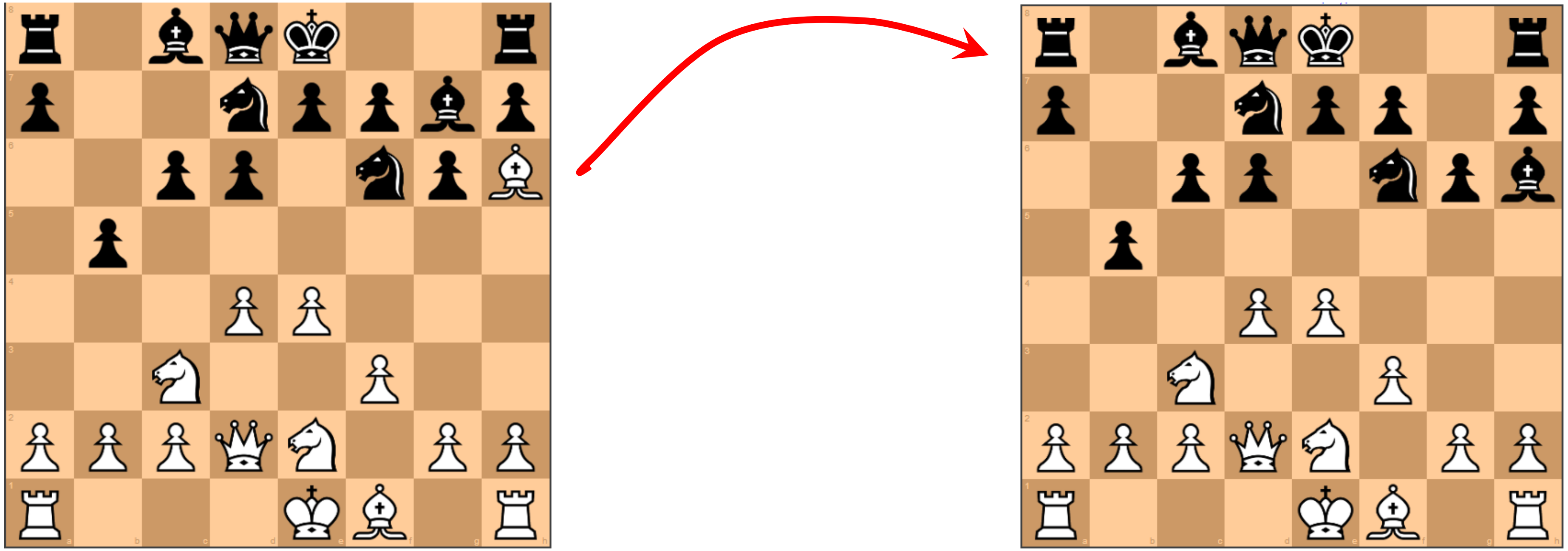
- We can also compute the **action-value function** as:

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

  - it represents the expected return from a given state s, when a given action a is selected and then policy π is followed

Why do we compute value functions?

S_t $\quad R_t = 0 \quad$ S_{t+1}

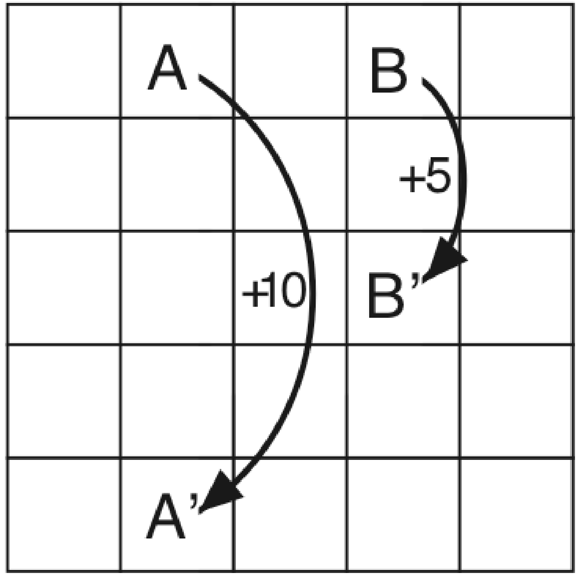


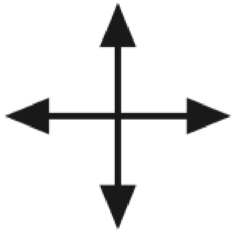
$V_\pi(S_t) = 0.51$ $\quad$ $V_\pi(S_t) = 0.49$

- Let assume the reward is +1 at if white wins and 0 elsewhere
- In this setting, $V_\pi(s)$ represents the probability of winning for white in s

State-Value Function: an example

- Let consider the following gridworld evironment:
  - Actions: `north`, `south`, `east`, `west` (deterministic dynamics)
  - Reward: -1 for bumping into the wall, positive from state A and B, 0 otherwise
  - Policy: random movement (25% north, 25% south, 25% east, 25% west)





Actions

State-Value Function ($\gamma = 0.9$)

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Bellman Expecation Equation

- The state–value function can again be **decomposed** into immediate reward plus discounted value of successor state:

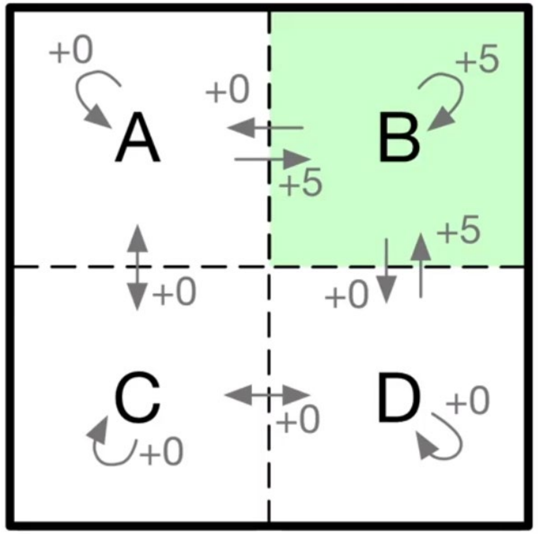
$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right) \end{aligned}$$

- The action-value function can be similarly decomposed:

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \\ &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s',a') \end{aligned}$$

Why Bellman Equations?

- Let see how Bellman equations can be used in practice with an example



$$V_\pi(A) \doteq \mathbb{E}_\pi[G_t \mid S_t = A]$$

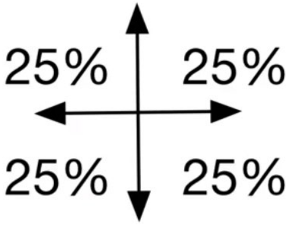
$$V_\pi(B) \doteq \mathbb{E}_\pi[G_t \mid S_t = B]$$

$$V_\pi(C) \doteq \mathbb{E}_\pi[G_t \mid S_t = C]$$

$$V_\pi(D) \doteq \mathbb{E}_\pi[G_t \mid S_t = D]$$

?

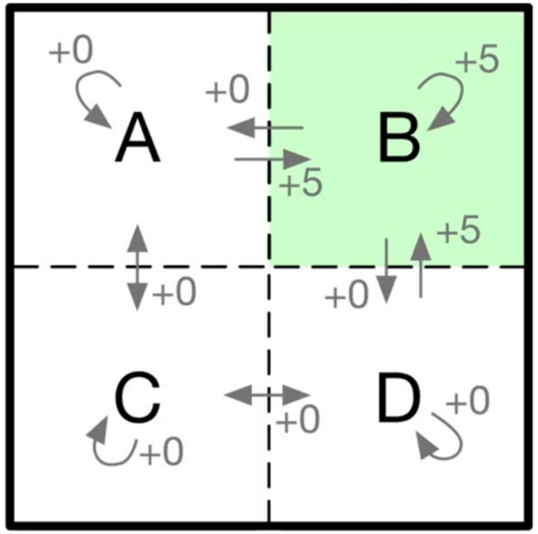
$\pi(a|s)$



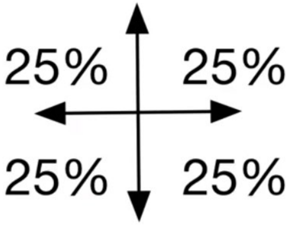
$\gamma = 0.7$

Why Bellman Equations?

- Let see how Bellman equations can be used in practice with an example



$\pi(a|s)$



$\gamma = 0.7$

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s] \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right)
\end{aligned}
$$

$$
\begin{aligned}
V_\pi(A) &= \frac{1}{4}(5 + 0.7V_\pi(B)) + \frac{1}{4}0.7V_\pi(C) + \frac{1}{2}0.7V_\pi(A) \\
V_\pi(B) &= \frac{1}{2}(5 + 0.7V_\pi(B)) + \frac{1}{4}0.7V_\pi(A) + \frac{1}{4}0.7V_\pi(D) \\
V_\pi(C) &= \frac{1}{4}0.7V_\pi(A) + \frac{1}{4}0.7V_\pi(D) + \frac{1}{2}0.7V_\pi(C) \\
V_\pi(D) &= \frac{1}{4}(5 + 0.7V_\pi(B)) + \frac{1}{4}0.7V_\pi(C) + \frac{1}{2}0.7V_\pi(D)
\end{aligned}
$$

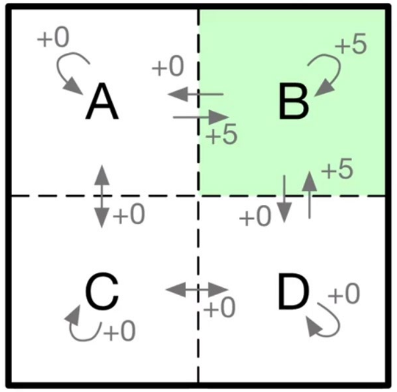
$V_\pi(A) = 4.2$

$V_\pi(B) = 6.1$

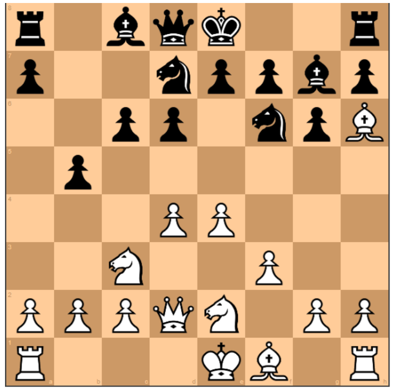
$V_\pi(C) = 2.2$

$V_\pi(D) = 4.2$

Limitations of Bellman Equations



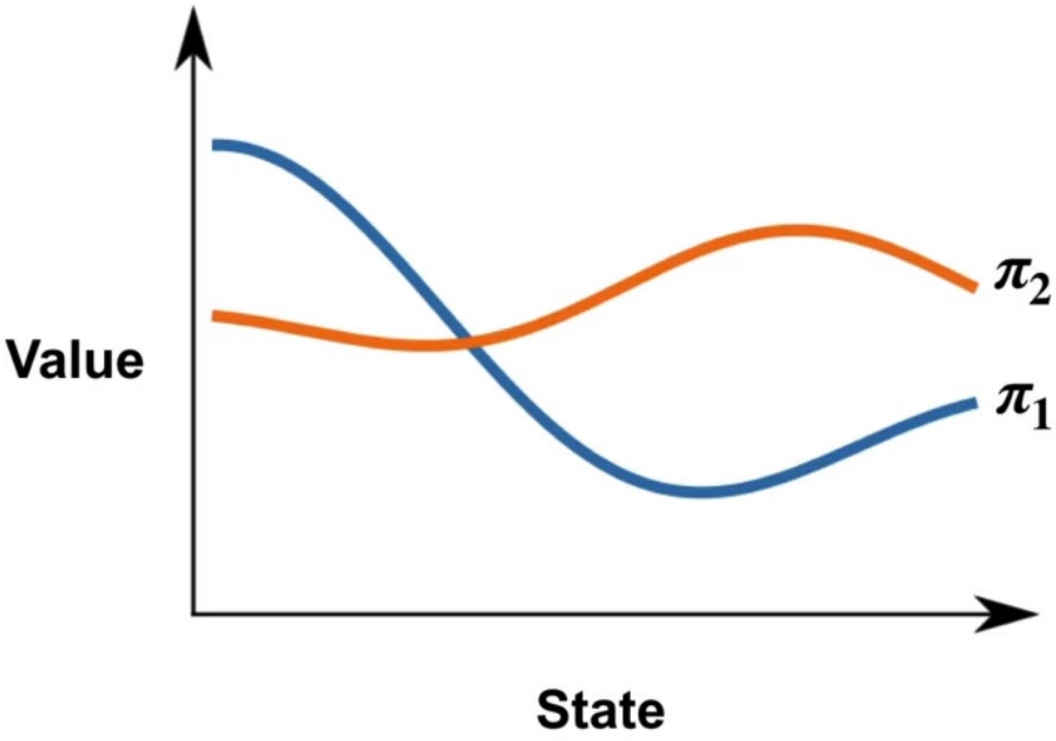
Solving 4 Linear Equations → V_π

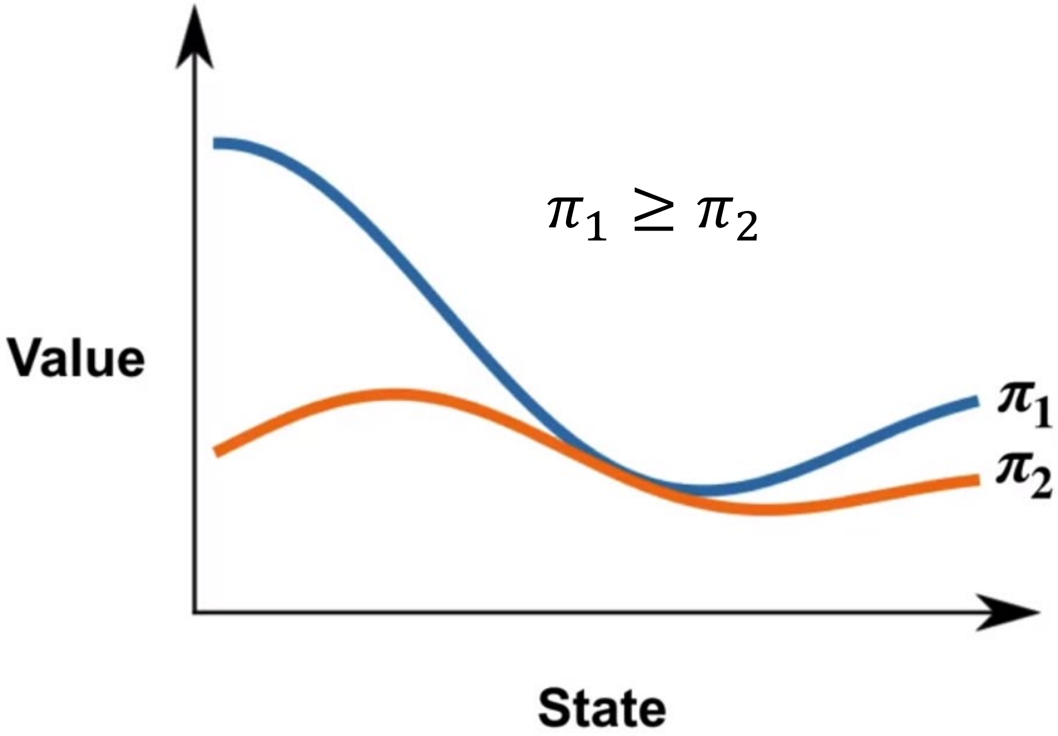


Solving 10^{4}[illegible] Linear Equations → V_π

Optimality

Comparing two policies

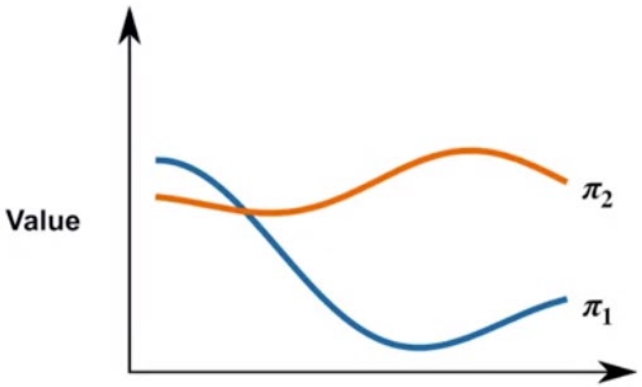


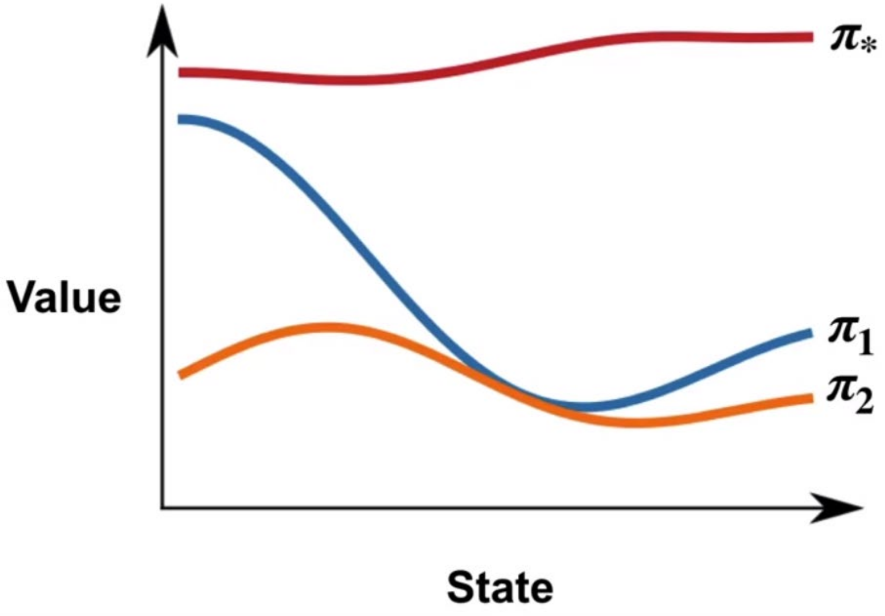


- We say that $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s), \quad \forall s \in \mathcal{S}$

Optimal Policy

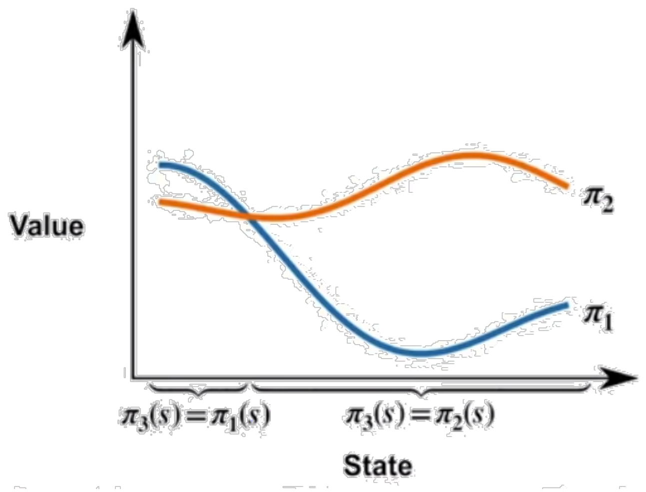
- For any Markov Decision Process, there exists always at least one **optimal deterministic policy** π^* that is better or equal to all the others ($\pi^* \geq \pi, \ \forall \pi$)
- **Sketch of proof**

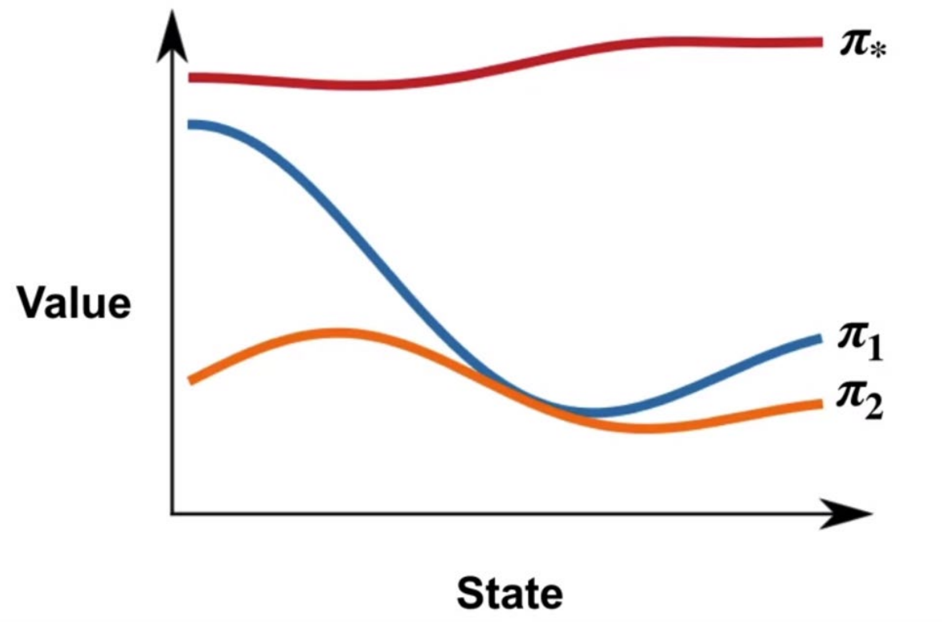




Optimal Policy

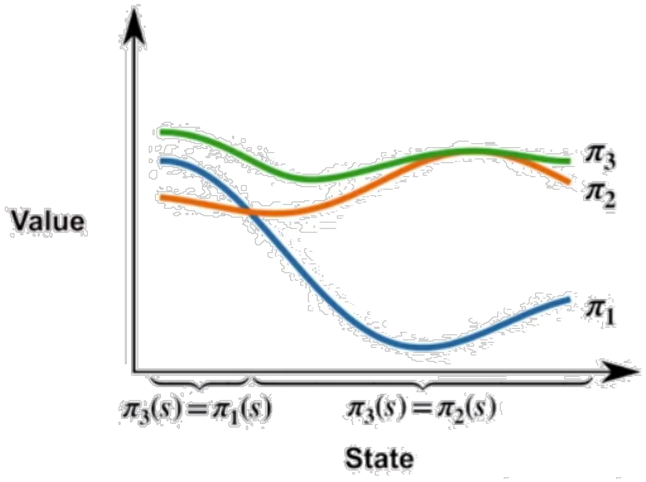
- For any Markov Decision Process, there exists always at least one **optimal deterministic policy** π^* that is better or equal to all the others ($\pi^* \geq \pi, \ \forall \pi$)
- **Sketch of proof**

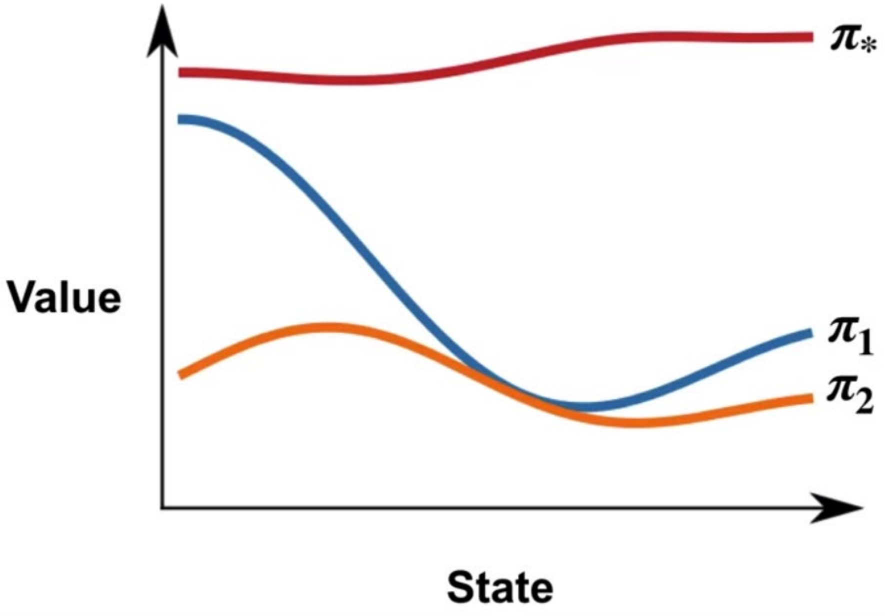




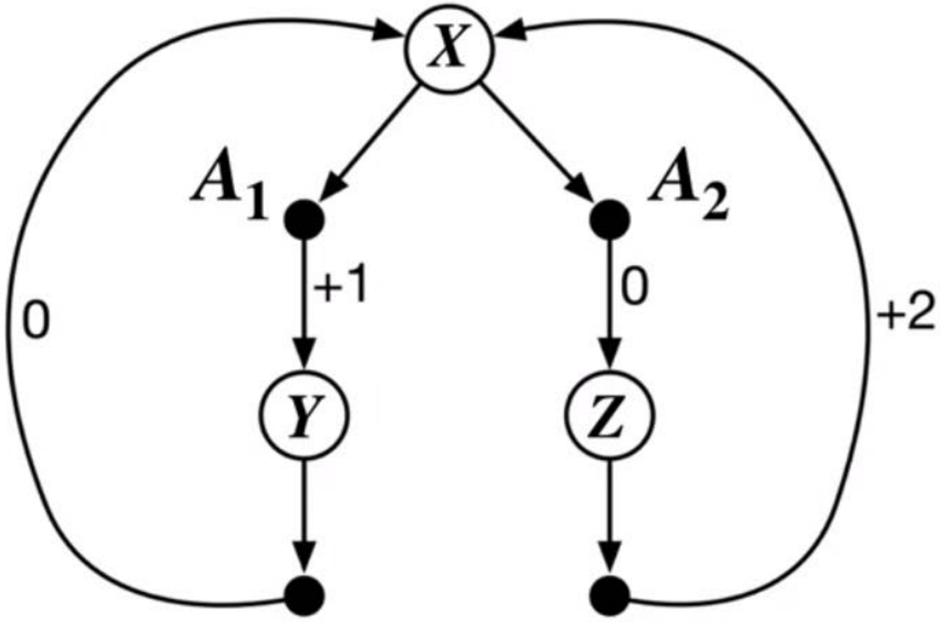
Optimal Policy

- For any Markov Decision Process, there exists always at least one **optimal deterministic policy** π^* that is better or equal to all the others ($\pi^* \geq \pi, \ \forall\pi$)
- **Sketch of proof**

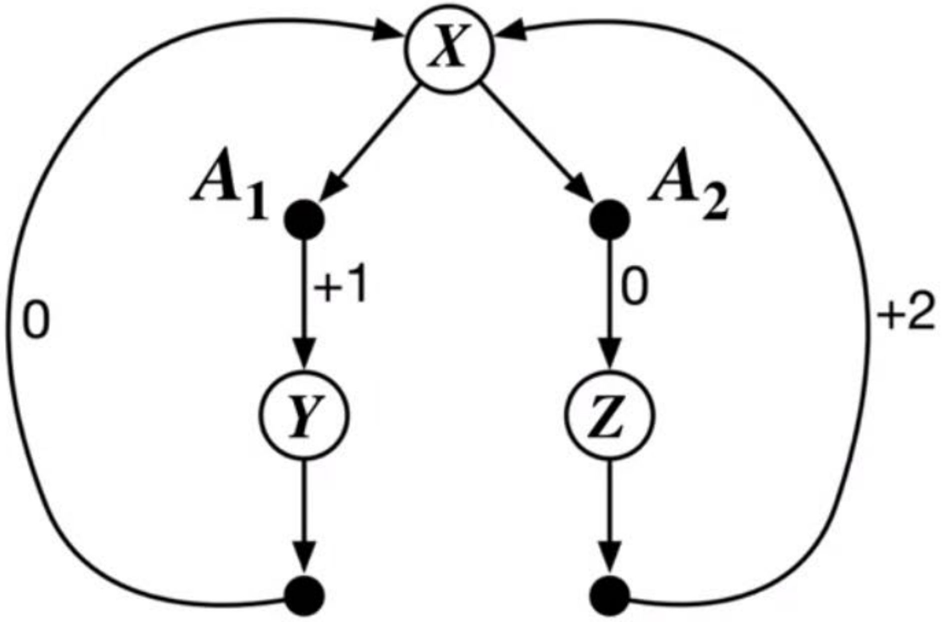




Optimal Policy: an example

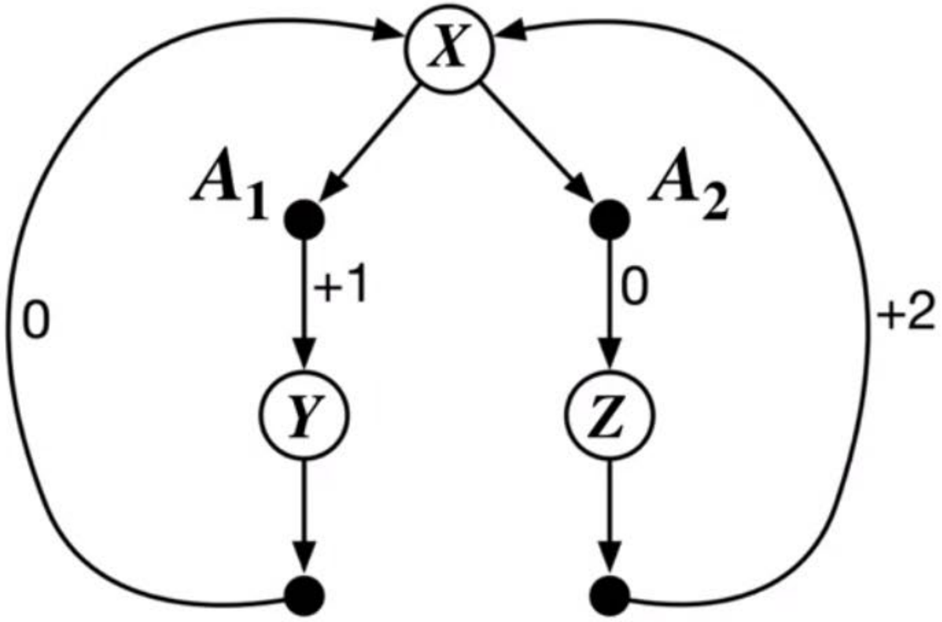


Optimal Policy: an example



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

Optimal Policy: an example



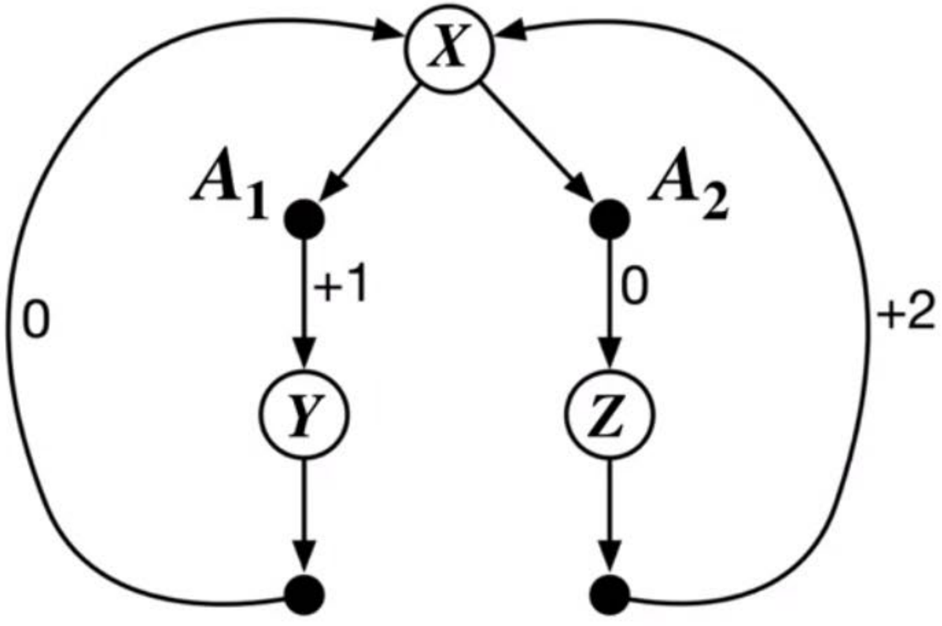
$\pi_1(X) = A_1$ $\quad$ $\pi_2(X) = A_2$

$\gamma = 0$

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

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

Optimal Policy: an example



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

$\gamma = 0$

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

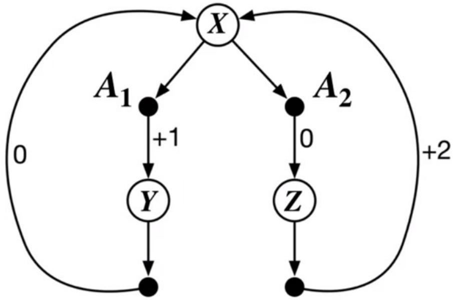
$v_{\pi_2}(X) = 0$

$\gamma = 0.9$

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

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

Limitations of solving MDP to find optimal policy



2 Deterministic Policies → Brute-Force Search

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

Optimal Value Function

- We said that that $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s), \quad \forall s \in \mathcal{S}$
- As a consequence we can compute optimal state-value function and optimal action-value function as:

$$V^*(s) \doteq \max_\pi V_\pi(s) \quad \forall s \in \mathcal{S}$$

$$Q^*(s,a) \doteq \max_\pi Q_\pi(s,a) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

Bellman Optimality Equations

- Bellman Optimality Equation for V*

$$
\begin{aligned}
V^*(s) &= \sum_{a \in \mathcal{A}} \pi^*(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right) \\
&= \max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}
\end{aligned}
$$

- Bellman Optimality Equation for Q*

$$
\begin{aligned}
Q^*(s,a) &= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi^*(a'|s') Q^*(s',a') \\
&= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a'} Q^*(s',a')
\end{aligned}
$$

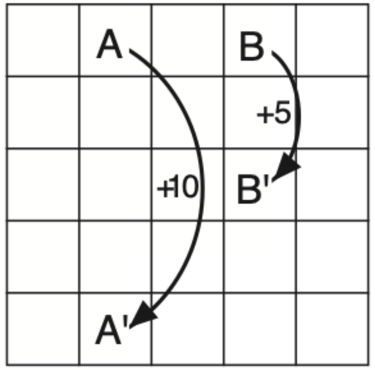
- Why do we care?

Why do we care?

- From V^* and Q^* we can easily compute the optimal policy π^*

$$\pi^*(s) = \arg\max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\} = \arg\max_a Q^*(s,a)$$

- Example:

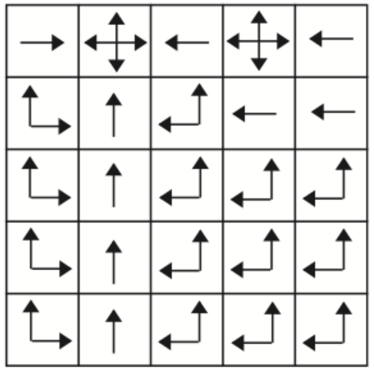


a) gridworld

$\gamma = 1$

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*

Computing the Optimal Value Function

$$V^*(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}$$

$$\mathrm{r, p, \gamma} \longrightarrow \boxed{\text{Linear System Solver}} \longrightarrow \mathrm{V^*}$$

Computing the Optimal Value Function

$$V^*(s) = \boxed{\max_a} \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}$$

$$r, p, \gamma \longrightarrow \boxed{\text{Linear System Solver}} \longrightarrow V^*$$

Computing the Optimal Value Function

$$V^*(s) = \boxed{\max_a} \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}$$

$r, p, \gamma \longrightarrow$ Linear System Solver $\longrightarrow V^*$

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s') \right)$$

$\pi^*, r, p, \gamma \longrightarrow$ Linear System Solver $\longrightarrow V^*$

Computing the Optimal Value Function

$$V^*(s) = \boxed{\max_a} \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}$$

r, p, γ $\longrightarrow$ ~~Linear System Solver~~ $\longrightarrow$ V^*

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s') \right)$$

?

π^*, r, p, γ $\longrightarrow$ Linear System Solver $\longrightarrow$ V^*

Computing the Optimal Value Function

$$V^*(s) = \boxed{\max_a} \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^*(s') \right\}$$

r, p, γ ⟶ ~~Linear System Solver~~ ⟶ V*

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s') \right)$$

?

π*, r, p, γ ⟶ Linear System Solver ⟶ V*

➡ Dynamic Programming and Reinforcement Learning Algorithms