

# RL @ PicsArt

Day 1, part 1

## Temporal Difference



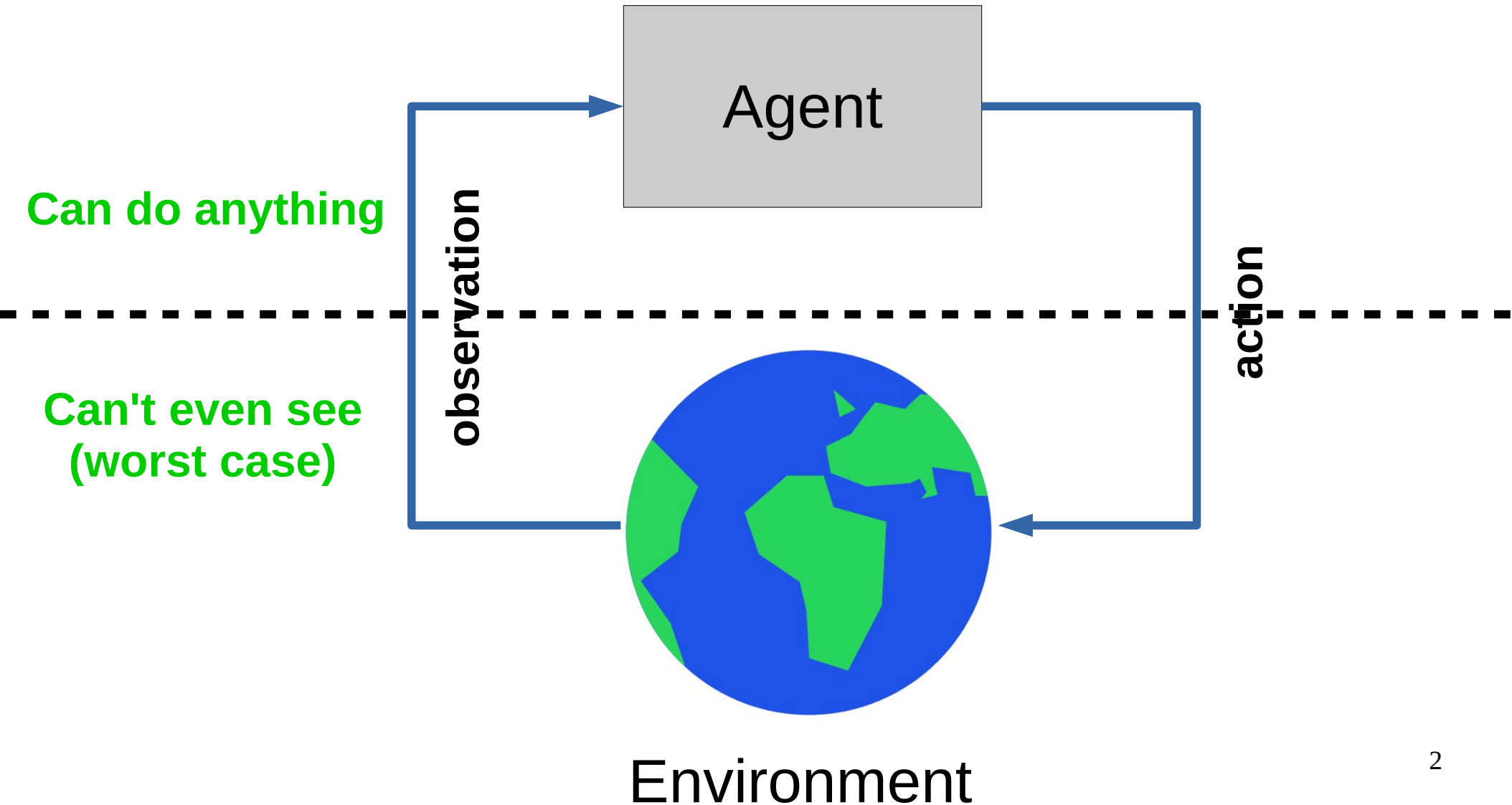
Yandex  
Data Factory

LAMBDA



**British Hedgehog  
Preservation Society**

# Recap: reinforcement learning



# Monte-carlo methods

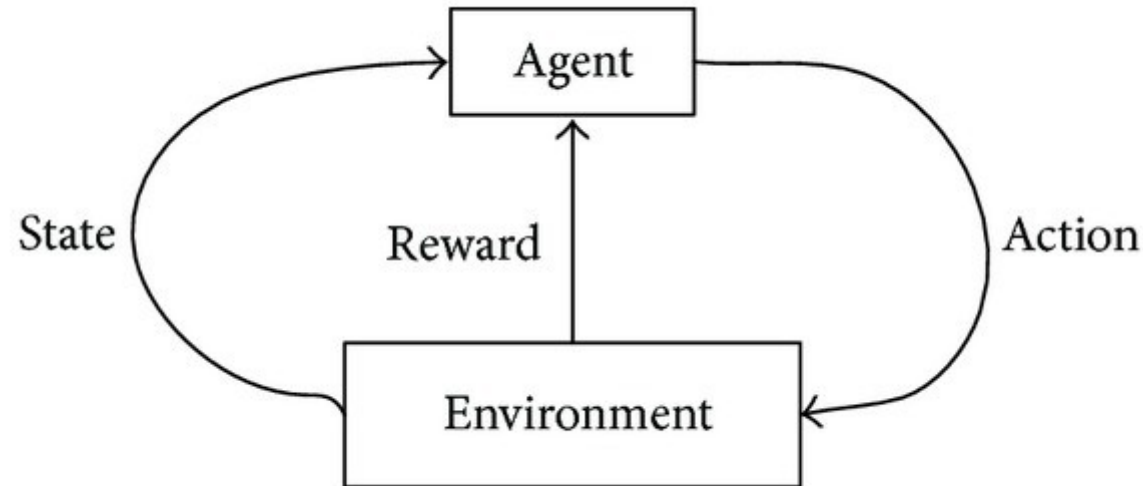
- $R(z)$  – evaluated at the very end
- Metaheuristics (genetic algorithm, etc.)
- Stochastic optimization (crossentropy method)

# Monte-carlo: drawbacks

- Both need a full session to start learning
- Requires a lot of interaction
  - A lot of crashed robots / simulations



# MDP formalism: reward on each tick



Classic MDP(Markov Decision Process)

Agent interacts with environment

- Environment states:  $s \in S$
- Agent actions:  $a \in A$
- State transition:  $P(s_{t+1}|s_t, a_t)$
- Reward:  $r_t = r(s_t, a_t)$

# Discounted reward MDP



Objective:  
Total action value

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots + \gamma^n \cdot r_{t+n}$$

$$G_t = \sum_i \gamma^i \cdot r_{t+i} \quad \gamma \in (0,1) \text{ const}$$

$\gamma \sim$  patience

Cake tomorrow is  $\gamma$  as good as now

Reinforcement learning:

- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[G] \rightarrow \max$$

# Discounted reward MDP



Objective:  
Total action value

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots + \gamma^n \cdot r_{t+n}$$

$$G_t = \sum_i \gamma^i \cdot r_{t+i} \quad \gamma \in (0,1) \text{ const}$$

**Trivia: which  $\gamma$  corresponds to “only current reward matters”?**

Reinforcement learning:

- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[G] \rightarrow \max$$

# Discounted reward MDP



Objective:  
Total reward

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots + \gamma^n \cdot r_{t+n}$$

$$G_t = \sum_i \gamma^i \cdot r_{t+i} \quad \gamma \in (0,1) \text{ const}$$

Reinforcement learning:

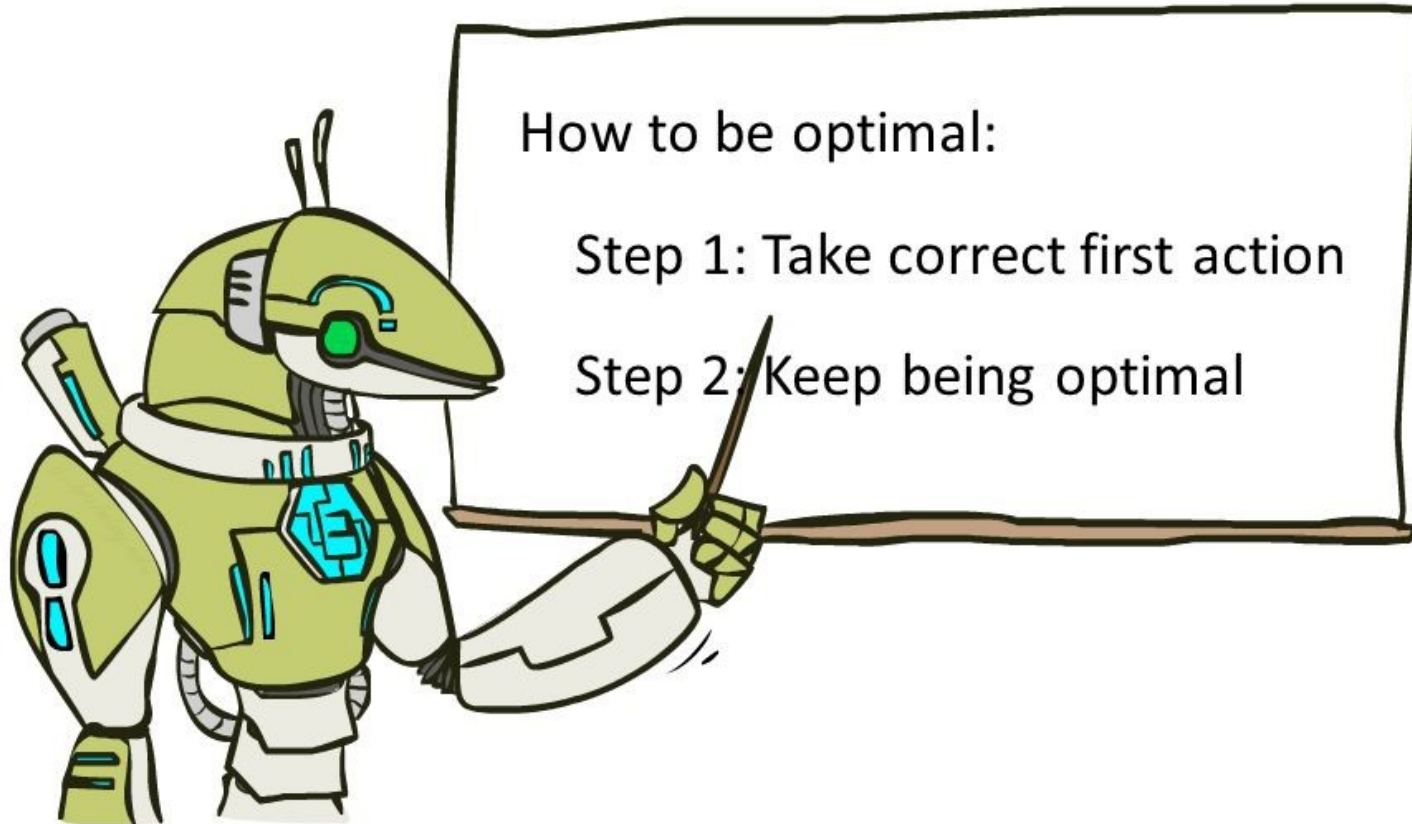
- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[G] \rightarrow \max$$

**Is optimal policy same as it would be in monte-carlo (if we add-up all  $r_t$ )?**

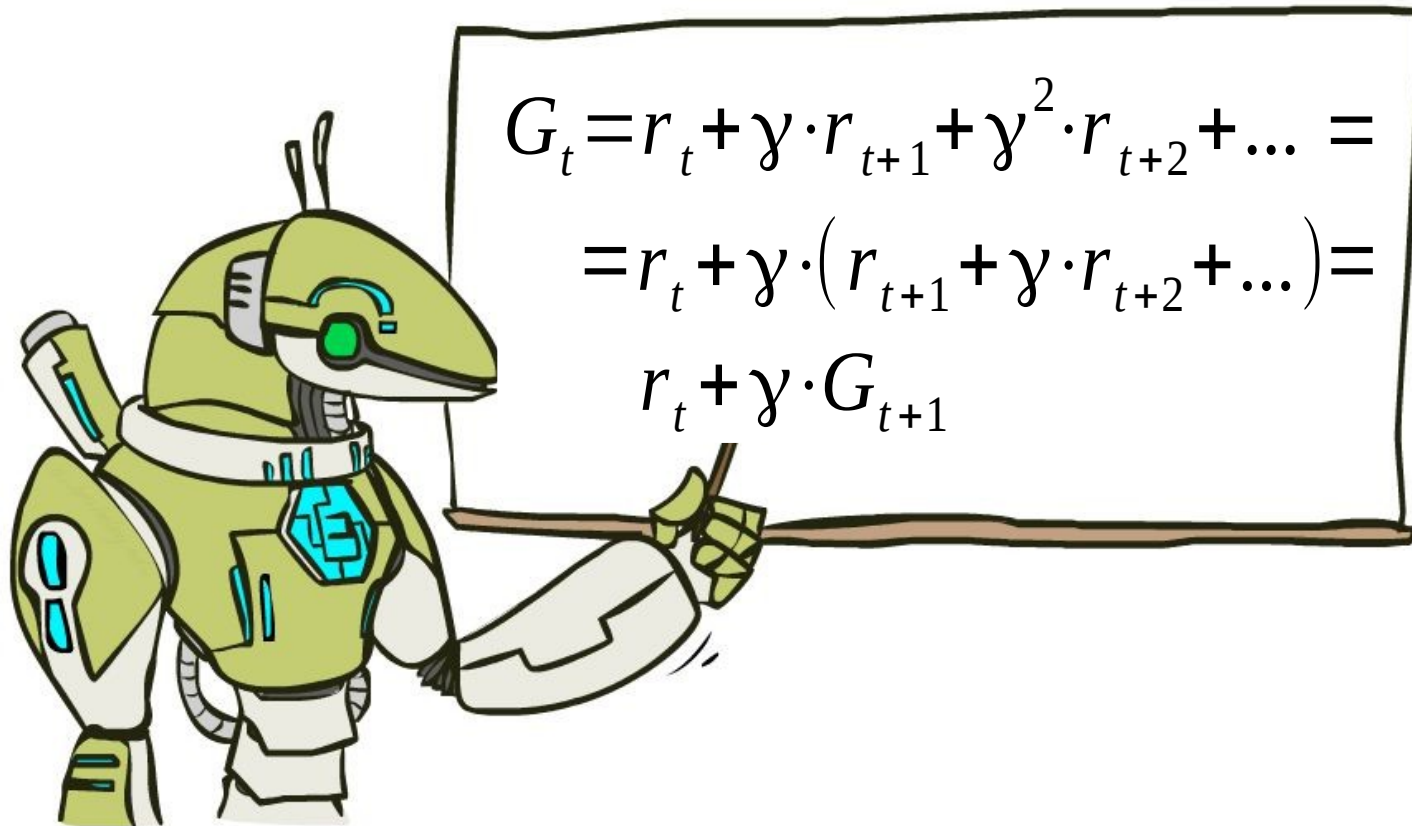


# Optimal policy



Recurrent optimal strategy definition

# Optimal policy



We rewrite  $G$  with sheer power of math!

# Value iteration (Temporal Difference)

Idea:

- For each state, obtain  $V(\text{state})$

$$V(s) = \max_a [r(s, a) + \gamma \cdot V(s'(s, a))]$$

**Definition**  $V(s)$  – expected total reward  $G$  that can be obtained starting from state  $s$  under optimal policy

# Value iteration (Temporal Difference)

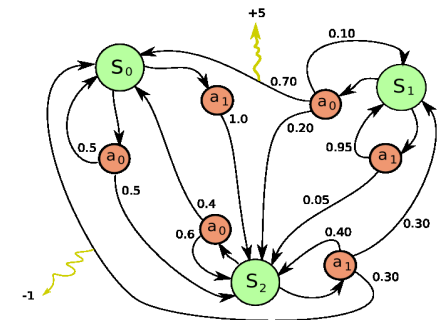
Idea:

- For each state, obtain  $V(\text{state})$

$$V(s) = \max_a [r(s, a) + \gamma \cdot E_{s' \sim P(s'|s, a)} V(s')] ]$$

↑  
Stochastic  
action outcome

**Trivia:** if we know the exact  $V(s)$  for all states, how do we determine the best actions?



# Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := \max_a [r(s, a) + \gamma \cdot E_{s' \sim P(s'|s, a)} V_i(s')]$$

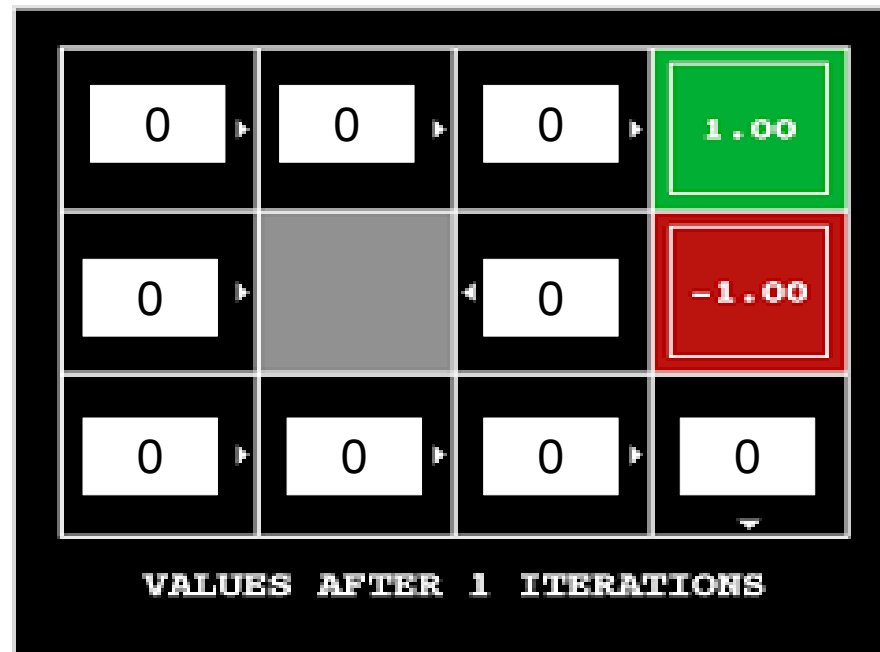
# Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := \max_a [r(s, a) + \gamma \cdot E_{s' \sim P(s'|s, a)} V_i(s')]$$



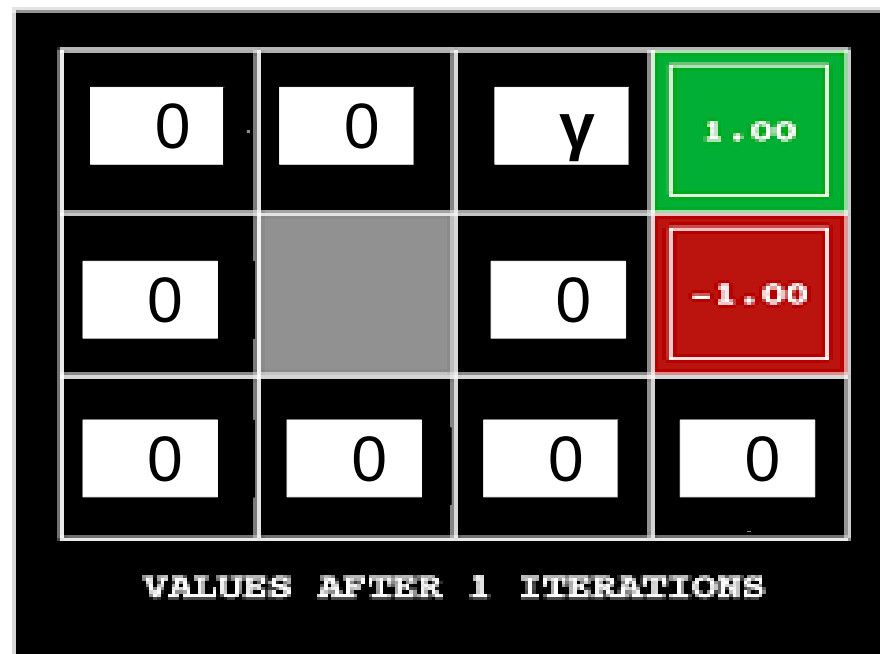
# Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := \max_a [r(s, a) + \gamma \cdot E_{s' \sim P(s'|s, a)} V_i(s')]$$



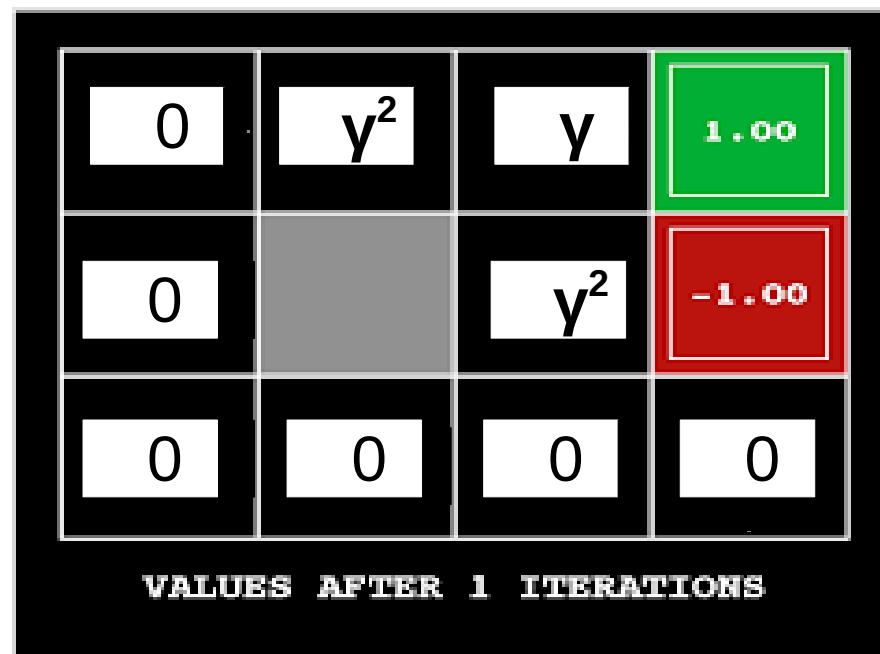
# Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := \max_a [r(s, a) + \gamma \cdot E_{s' \sim P(s'|s, a)} V_i(s')]$$





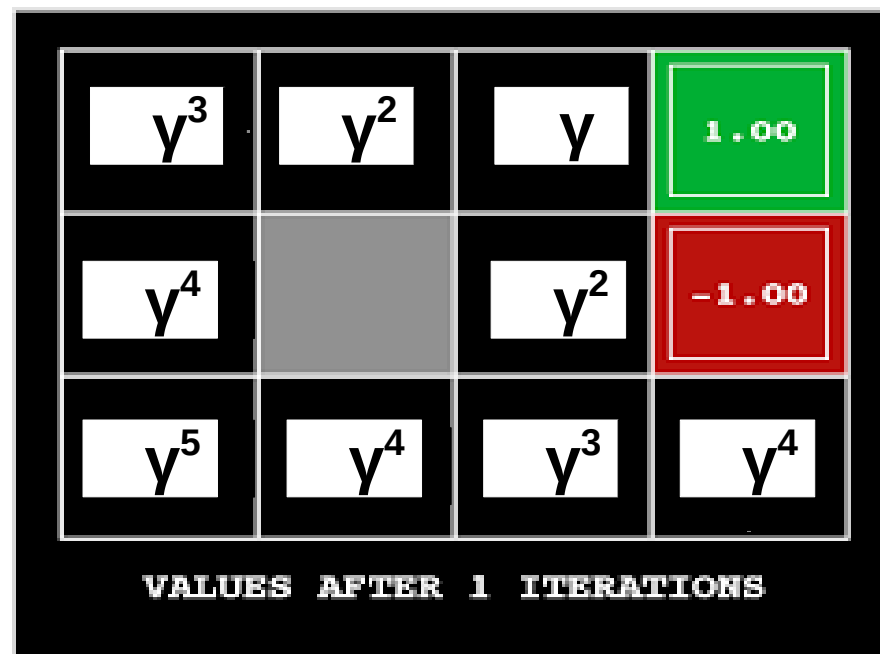
# Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := \max_a [r(s, a) + \gamma \cdot E_{s' \sim P(s'|s, a)} V_i(s')]$$



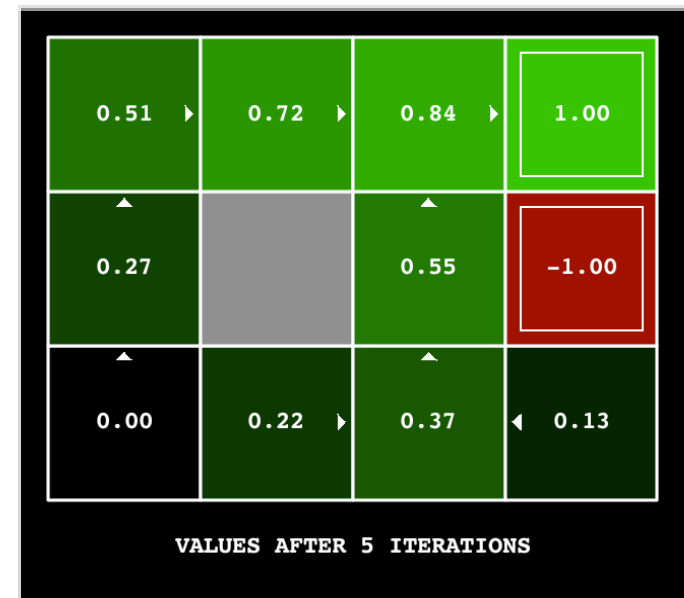
# Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := \max_a [r(s, a) + \gamma \cdot E_{s' \sim P(s'|s, a)} V_i(s')]$$



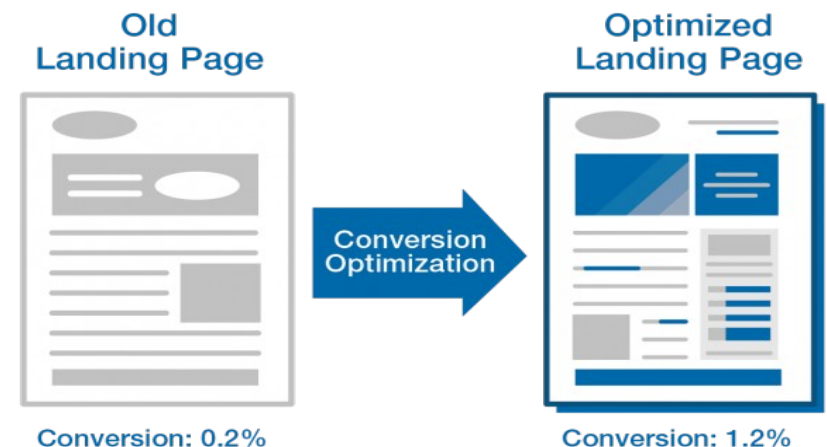
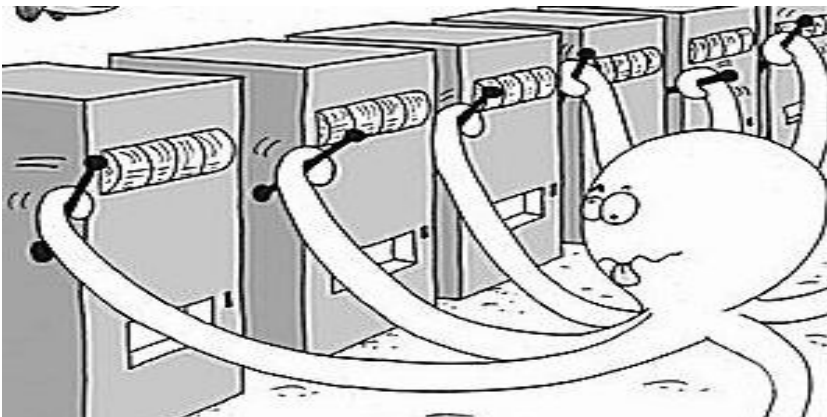
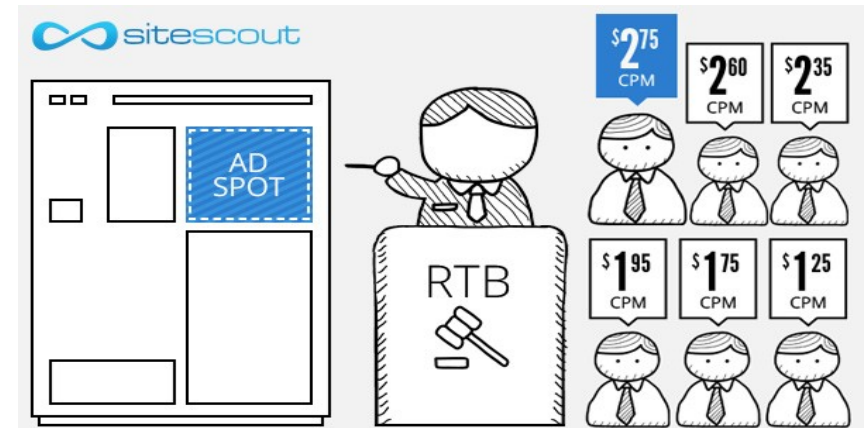
**Voila! We've solved the reinforcement learning!**

**Voila! We've solved the reinforcement learning!**  
Or have we?

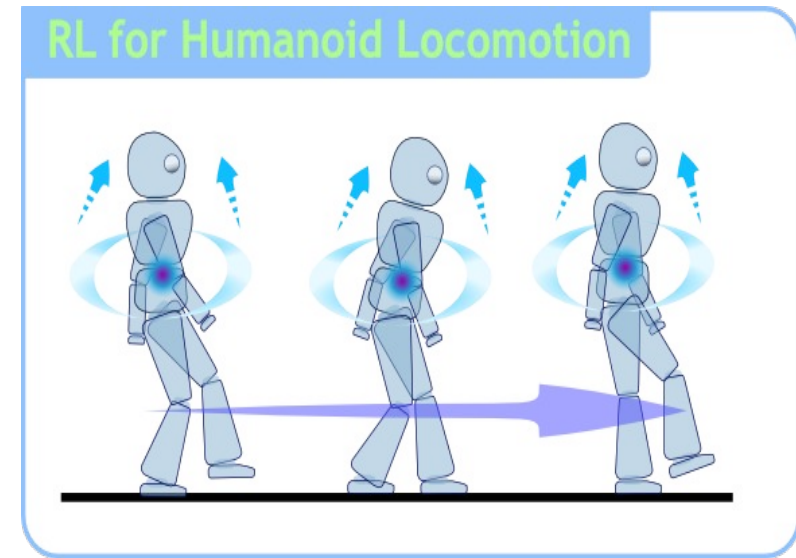
What happens if we apply it to real world  
problems?

# Reality check: web

- **Cases:**
  - Pick ads to maximize profit
  - Design landing page to maximize user retention
  - Recommend items to users
- **Common traits:**
  - Independent states
  - Large action space



# Reality check: dynamic systems



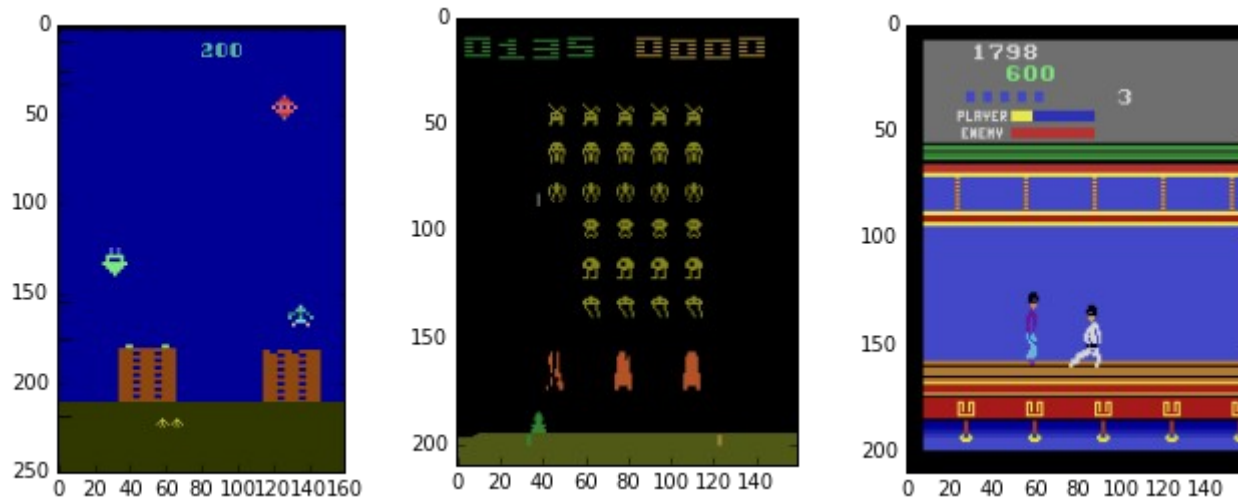
# Reality check: MOAR

- **Cases:**
  - Robots
  - Self-driving vehicles
  - Pilot assistant
  - More robots!
- **Common traits:**
  - Continuous state space
  - Continuous action space
  - Partially-observable environment
  - LONG sessions





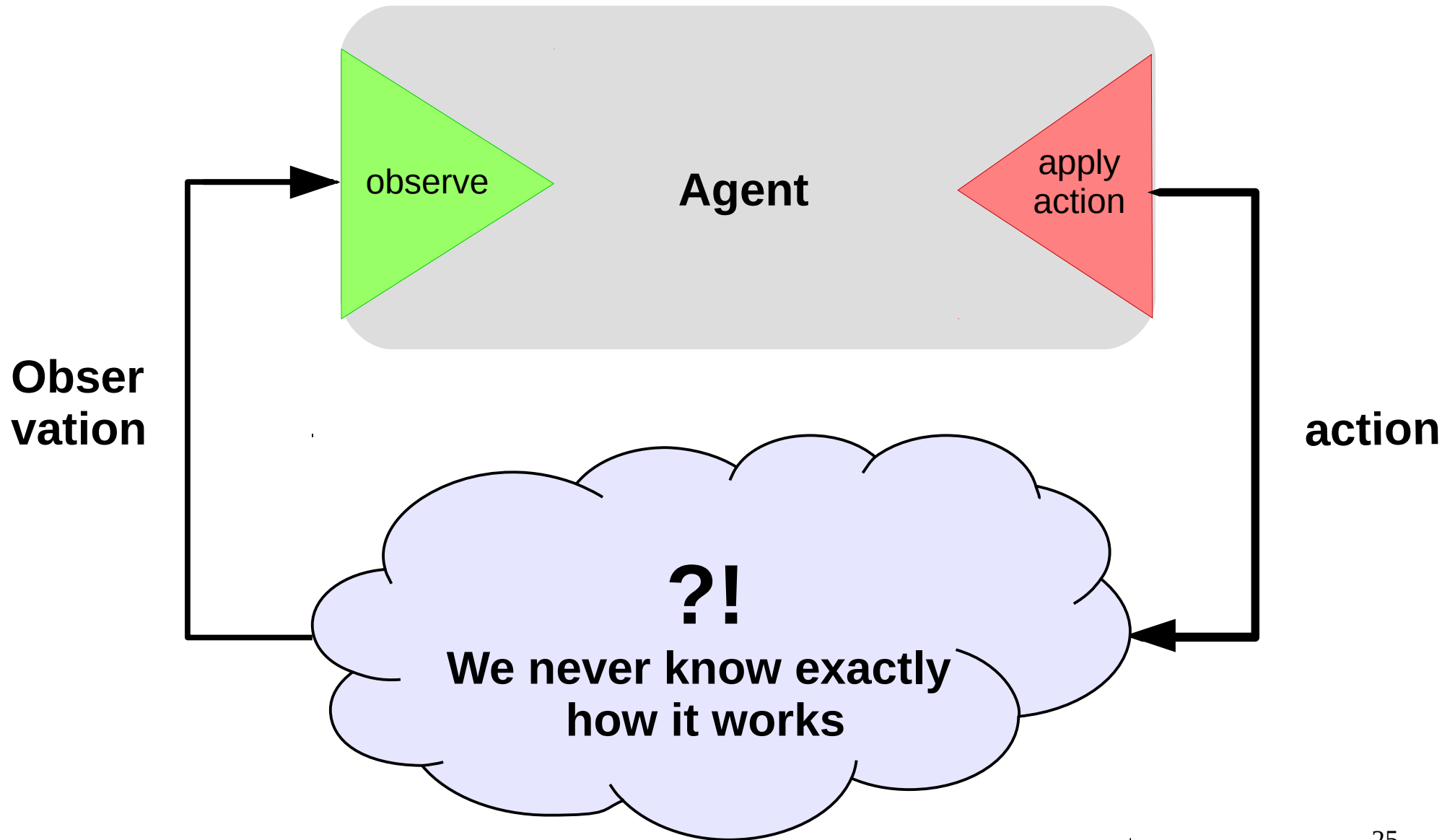
# Reality check: videogames



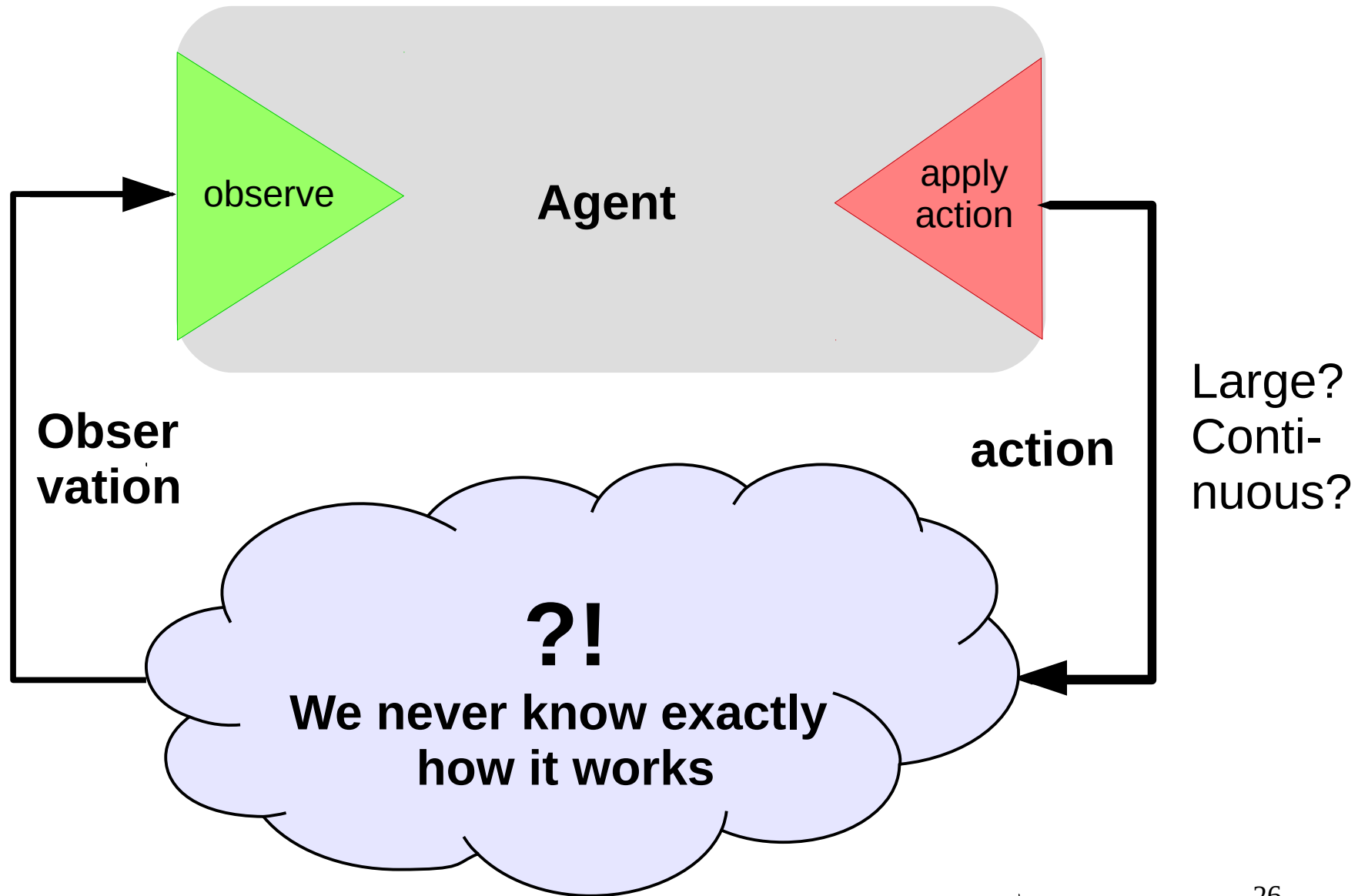
- **Trivia:** What are the states and actions?  
What are the problems?



# Real world



# Real world



**Problem:**

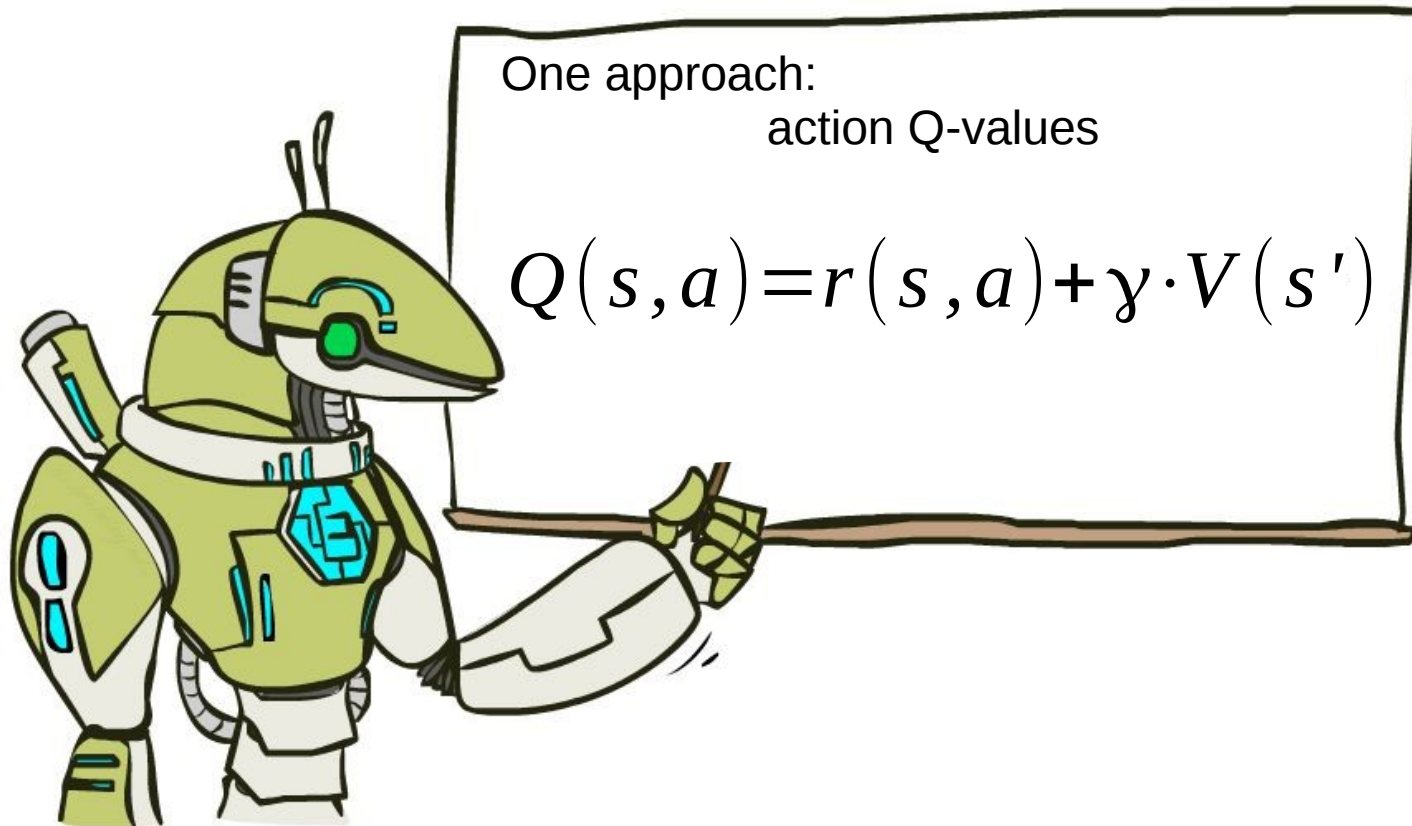
We never know actual

$$P(s'|s,a)$$

Learn it?

Get rid of it?

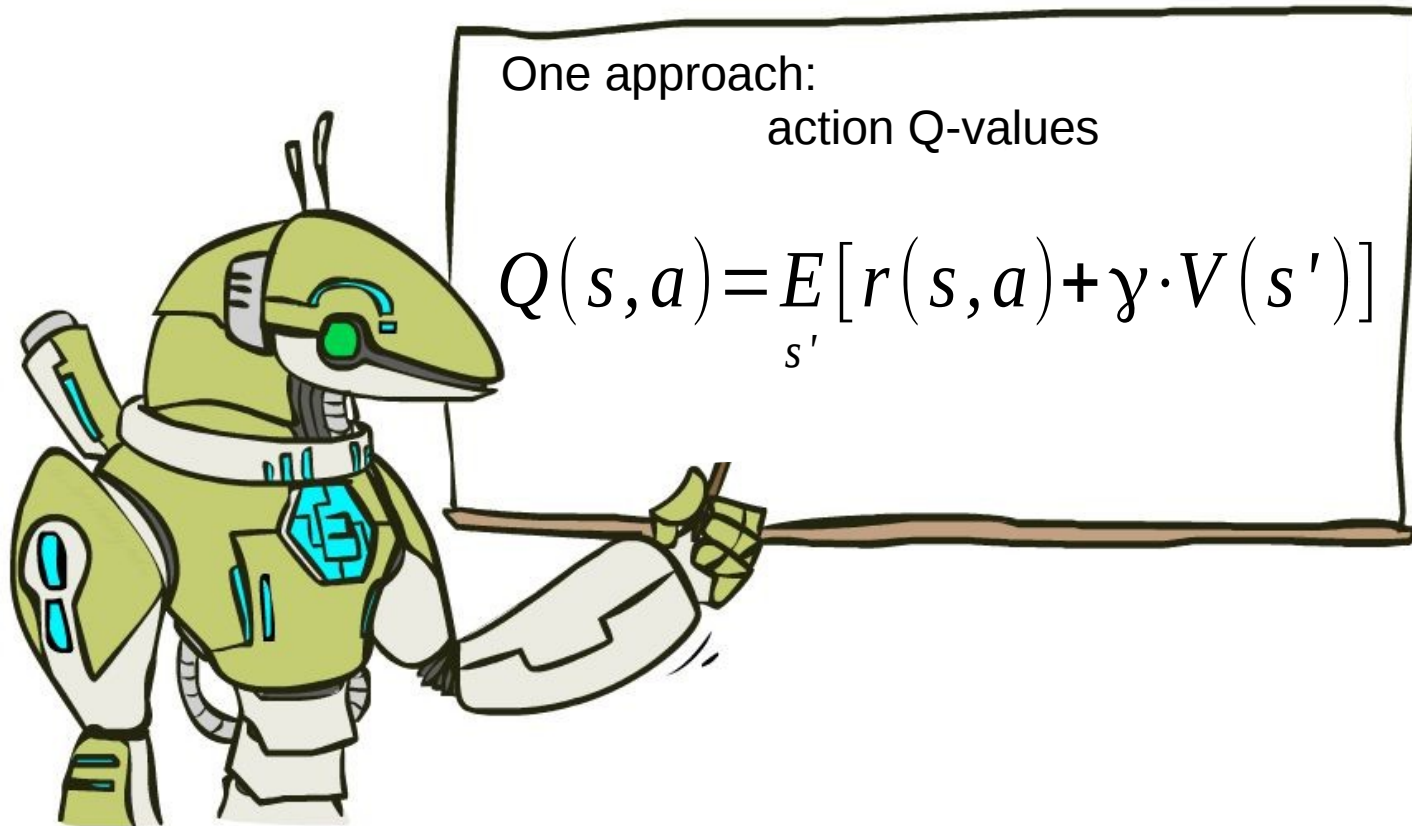
# From V to Q



**Action value  $Q(s, a)$**  is the expected total reward **G** agent gets from state **s** by taking action **a** and following policy  $\pi$  from next state.

$$\pi(s) : \operatorname{argmax}_a Q(s, a)$$

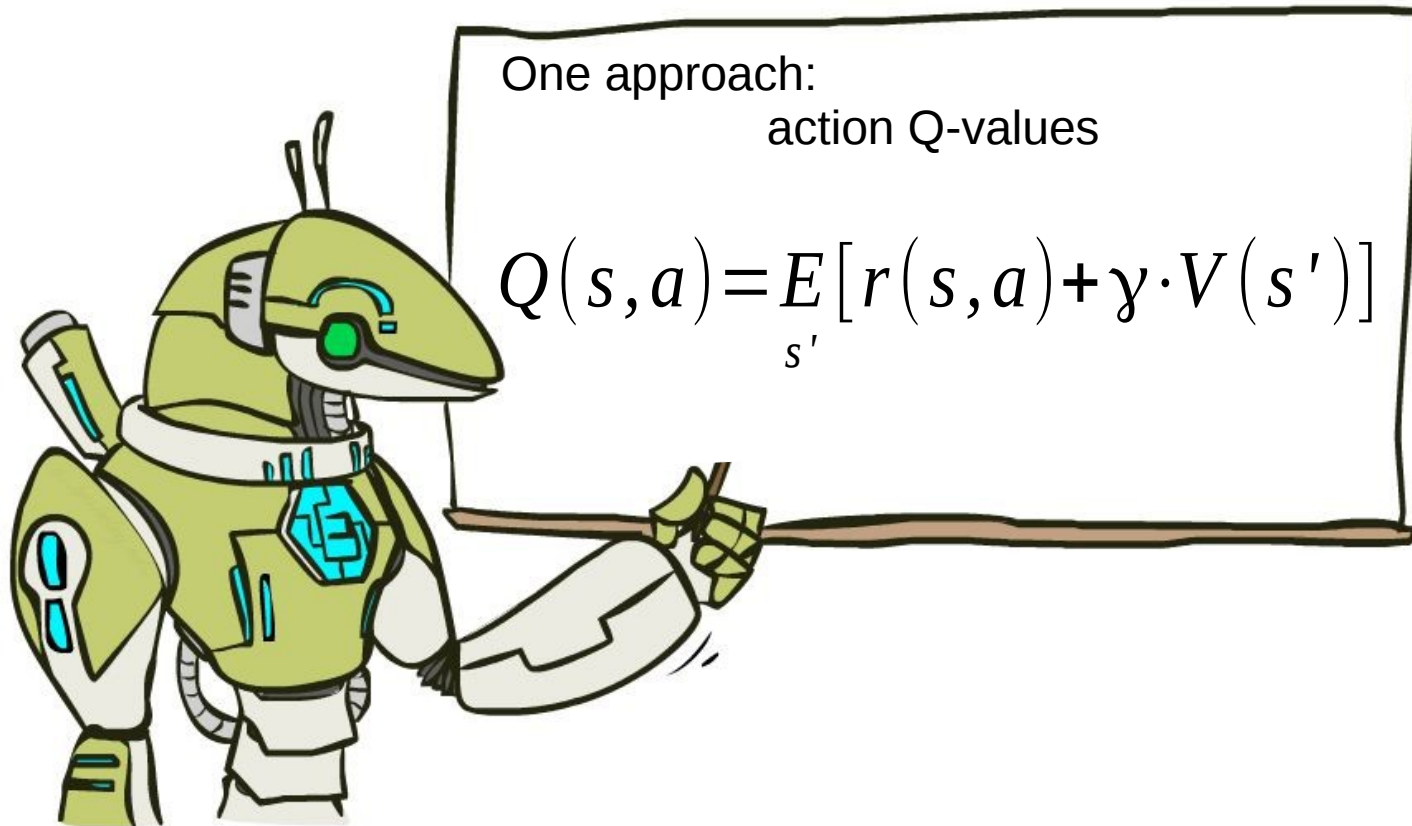
# From V to Q



**Action value  $Q(s, a)$**  is the expected total reward **G** agent gets from state **s** by taking action **a** and following policy  $\pi$  from next state.

$$\pi(s) : \operatorname{argmax}_a Q(s, a)$$

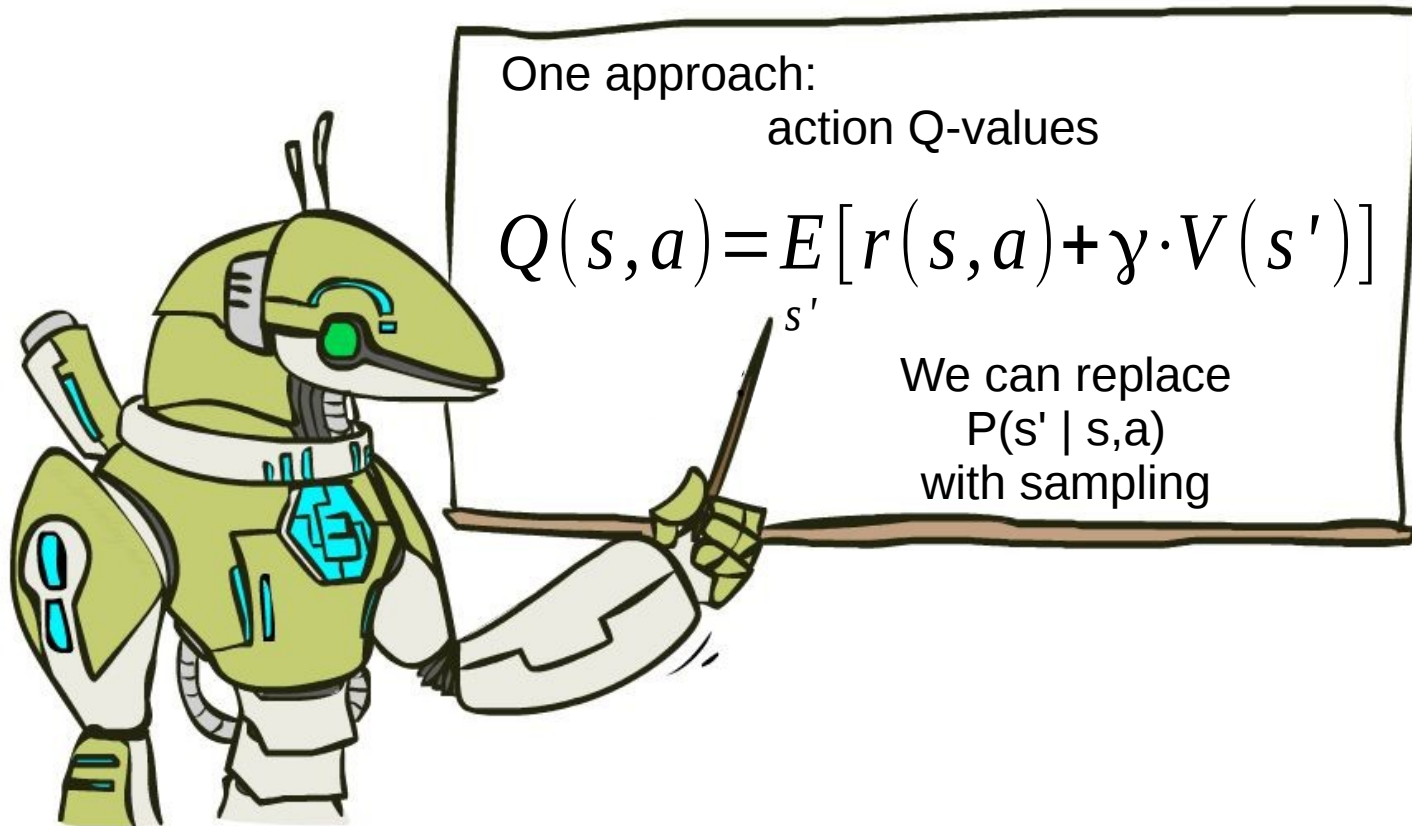
# From V to Q



**Action value  $Q(s, a)$**  is the expected total reward **G** agent gets from state **s** by taking action **a** and following policy  **$\pi$**  from next state.

$$\pi(s) : \operatorname{argmax}_a Q(s, a)$$

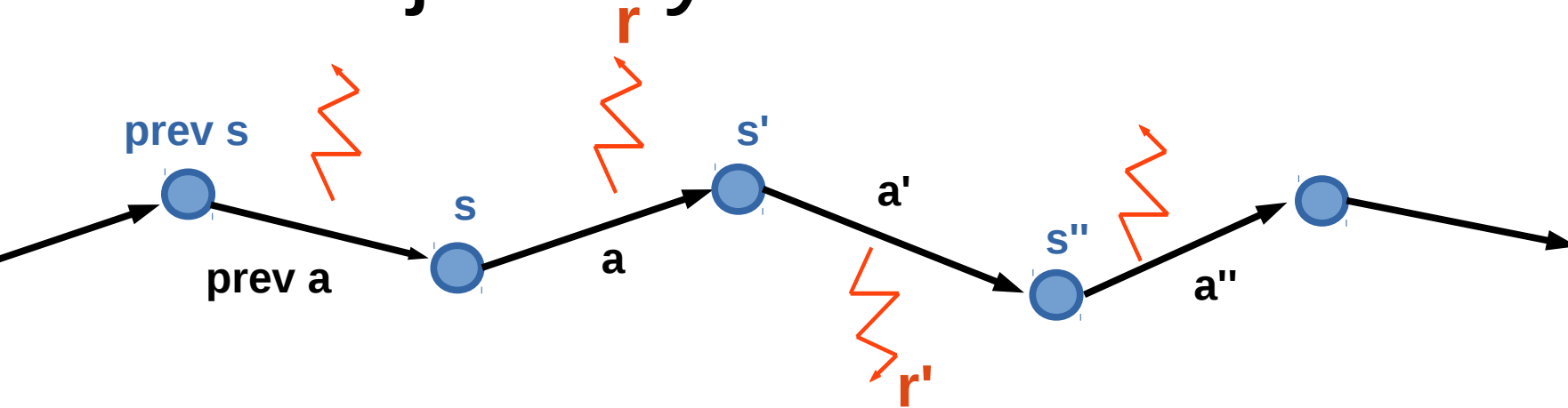
# From V to Q



$$Q(s_t, a_t) \leftarrow \alpha \cdot (r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')) + (1 - \alpha) Q(s_t, a_t)$$

$$\pi(s) : \operatorname{argmax}_a Q(s, a)$$

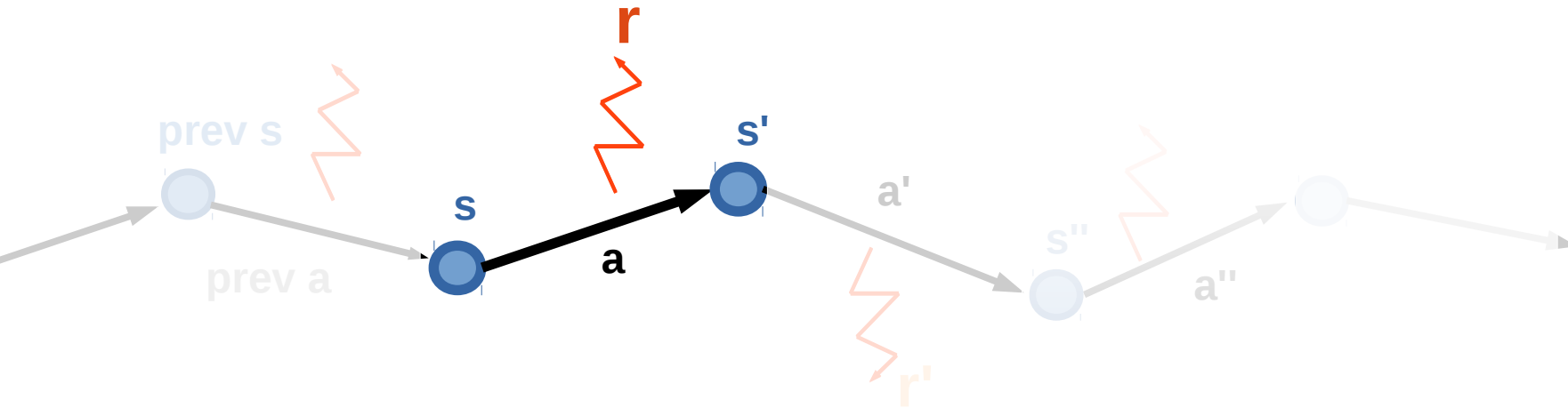
# MDP trajectory



- sample sequence of
  - states ( $s$ )
  - actions ( $a$ )
  - rewards ( $r$ )
- Can be infinite, we can't wait that long



# Q-learning

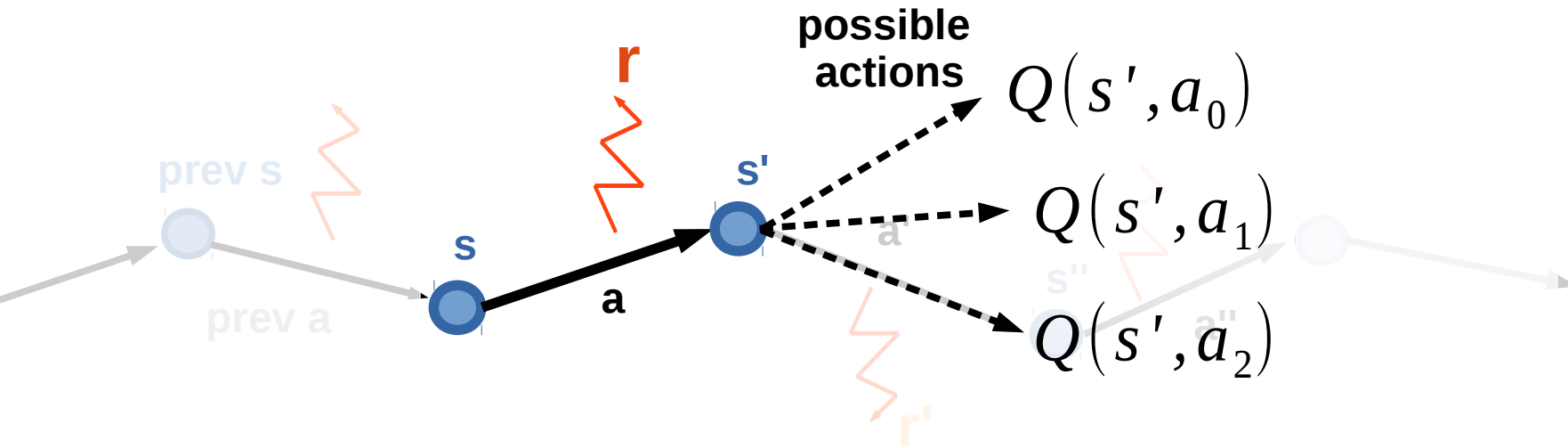


$$\forall s \in S, \forall a \in A, Q(s, a) \leftarrow 0$$

Loop:

- Sample  $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}' \rangle$  from env

# Q-learning



$$\forall s \in S, \forall a \in A, Q(s, a) \leftarrow 0$$

Loop:

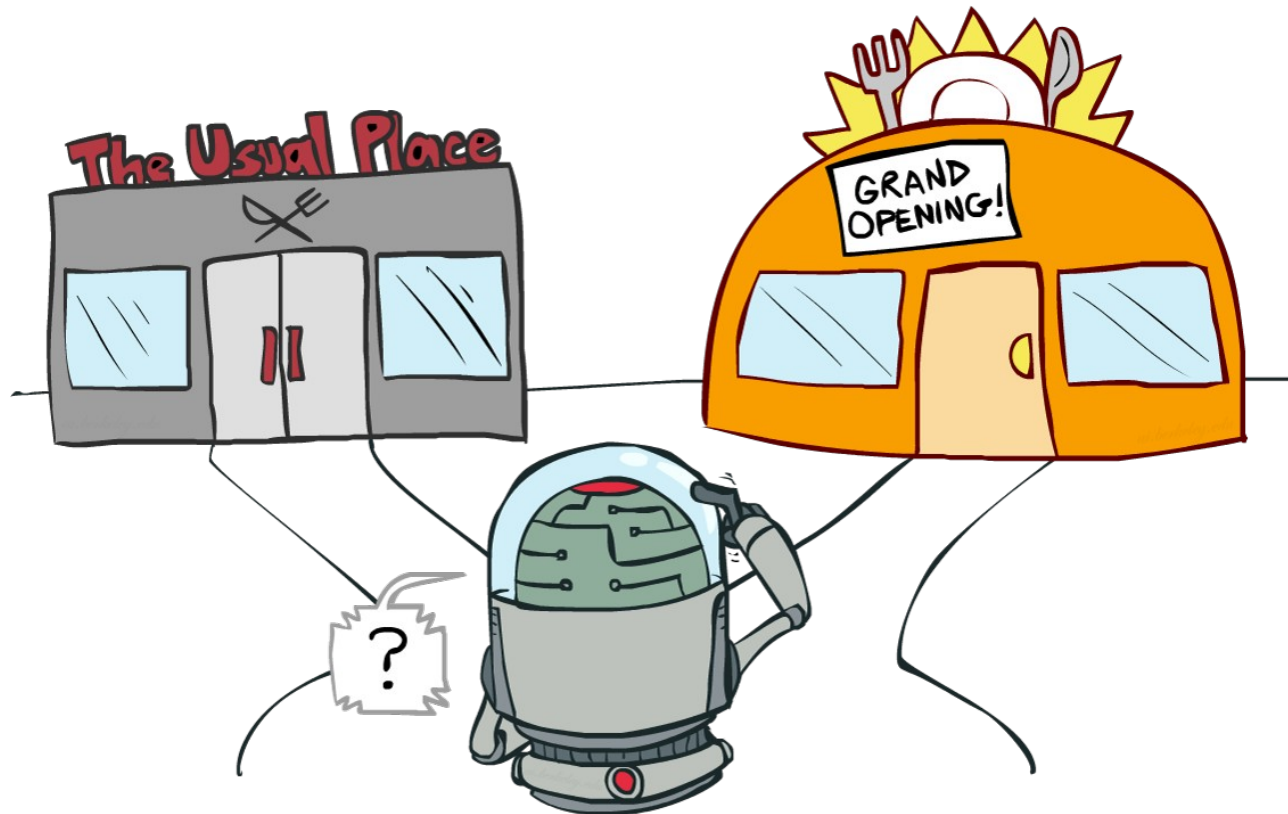
- Sample  $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}' \rangle$  from env

- Compute  $\hat{Q}(s, a) = r(s, a) + \gamma \max_{a_i} Q(s', a_i)$

- Update  $Q(s, a) \leftarrow \alpha \cdot \hat{Q}(s, a) + (1 - \alpha) Q(s, a)$

# Exploration Vs Exploitation

Balance between using what you learned and trying to find something even better



# Exploration Vs Exploitation

Strategies:

- $\epsilon$ -greedy
  - With probability  $\epsilon$  take a uniformly random action; otherwise take optimal action.
- Softmax
  - Pick action proportional to softmax of shifted normalized Q-values.

$$P(a) = \text{softmax}\left(\frac{Q(a)}{\tau}\right)$$

- Some methods have a built-in exploration strategy (e.g. crossentropy, A2c)<sup>36</sup>

**Problem:**

State space is usually large,  
sometimes continuous.

And so is action space;

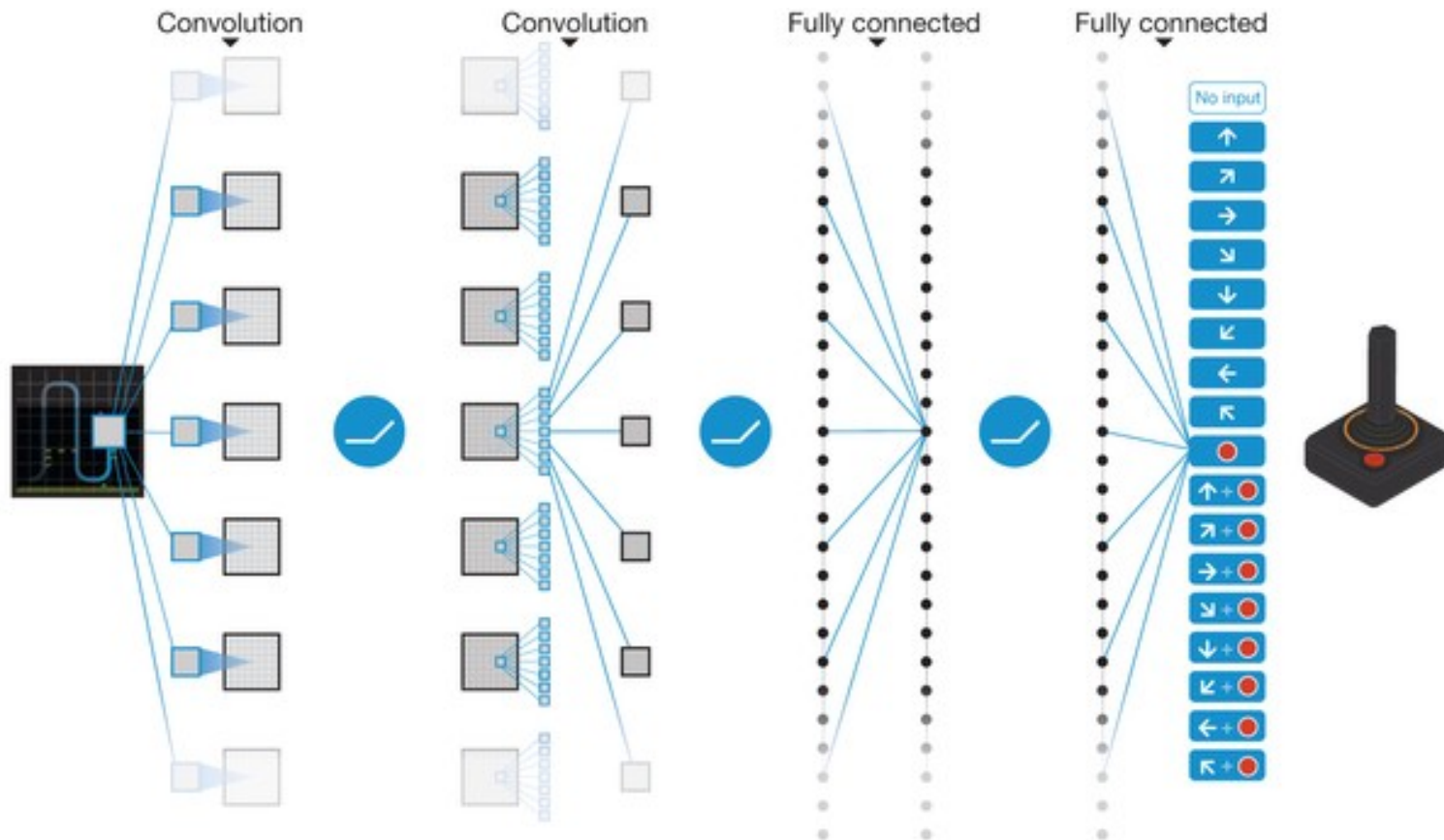
However, states do have a structure, similar  
states have similar action outcomes.

# From tables to approximations

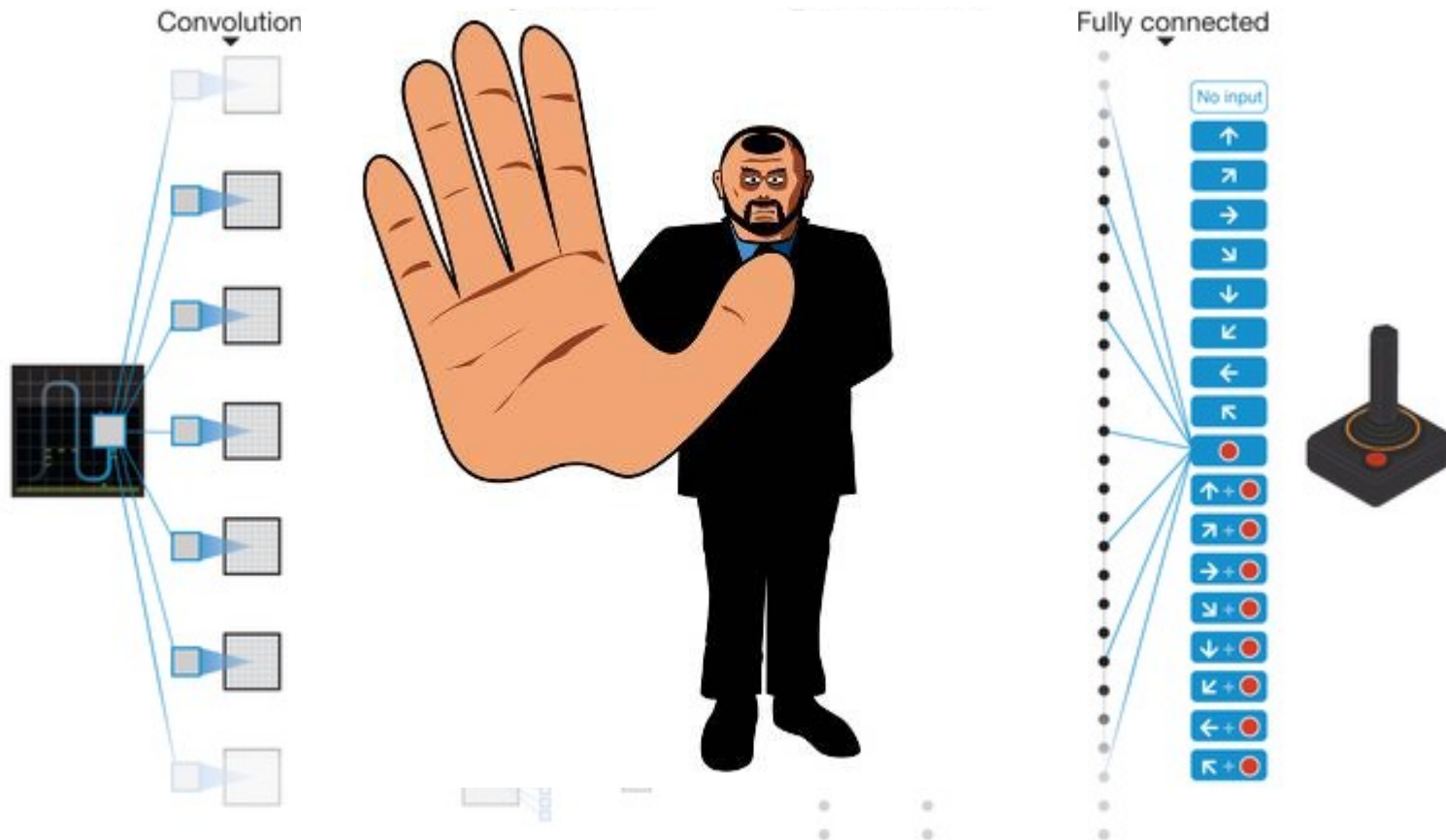
- Before:
  - For all states, for all actions, remember  $Q(s,a)$
- Now:
  - Approximate  $Q(s,a)$  with some function
  - e.g. linear model over state features

$$\operatorname{argmin}_{w,b} \left( Q(s_t, a_t) - [r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')] \right)^2$$

# Smells like a neural network



# Not so fast...





# Discounted reward MDP



Objective:  
Total reward

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots + \gamma^n \cdot r_{t+n}$$

$$G_t = \sum_i \gamma^i \cdot r_{t+i} \quad \gamma \in (0,1) \text{ const}$$

Reinforcement learning:

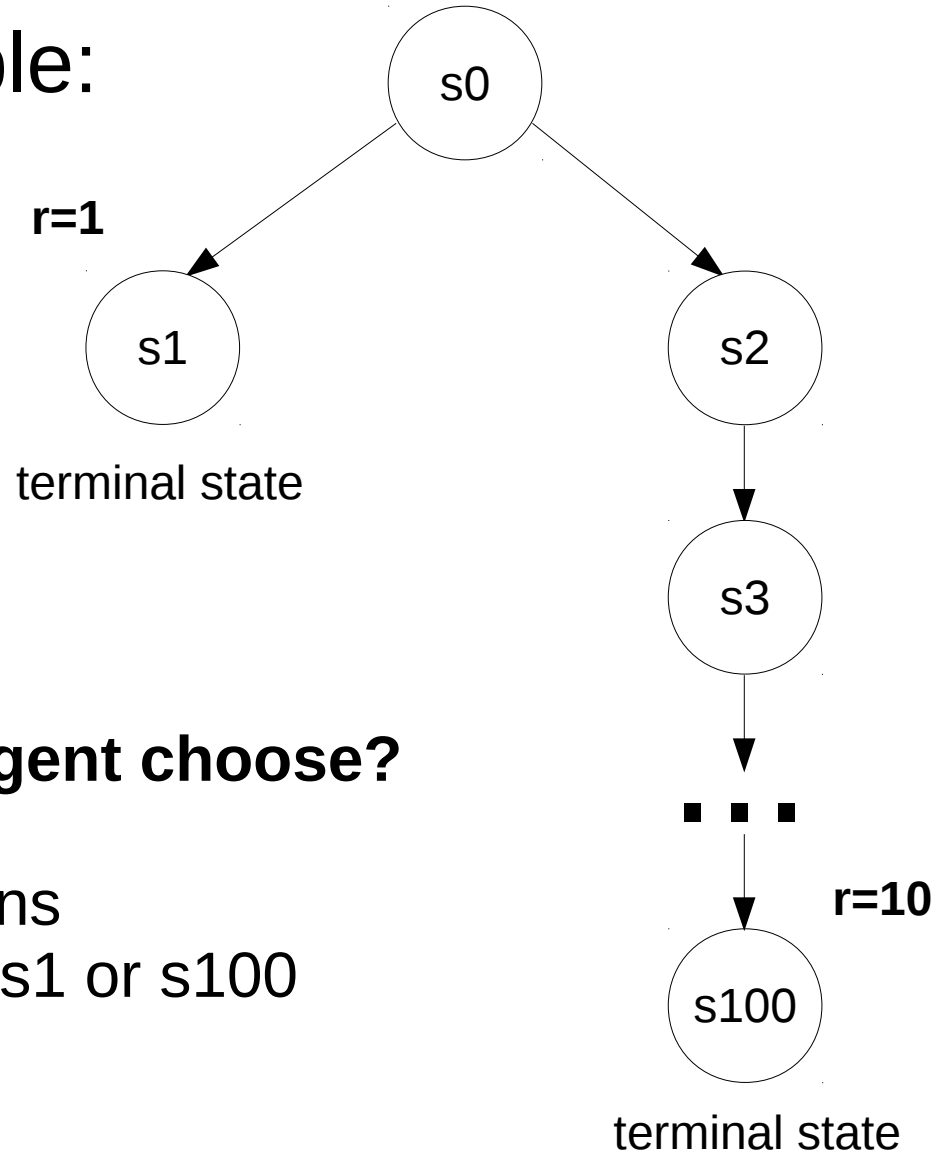
- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[G] \rightarrow \max$$

**Optimal policy isn't always maximizing monte-carlo reward!**

# Discounted reward **fails** #1

Trivial example:

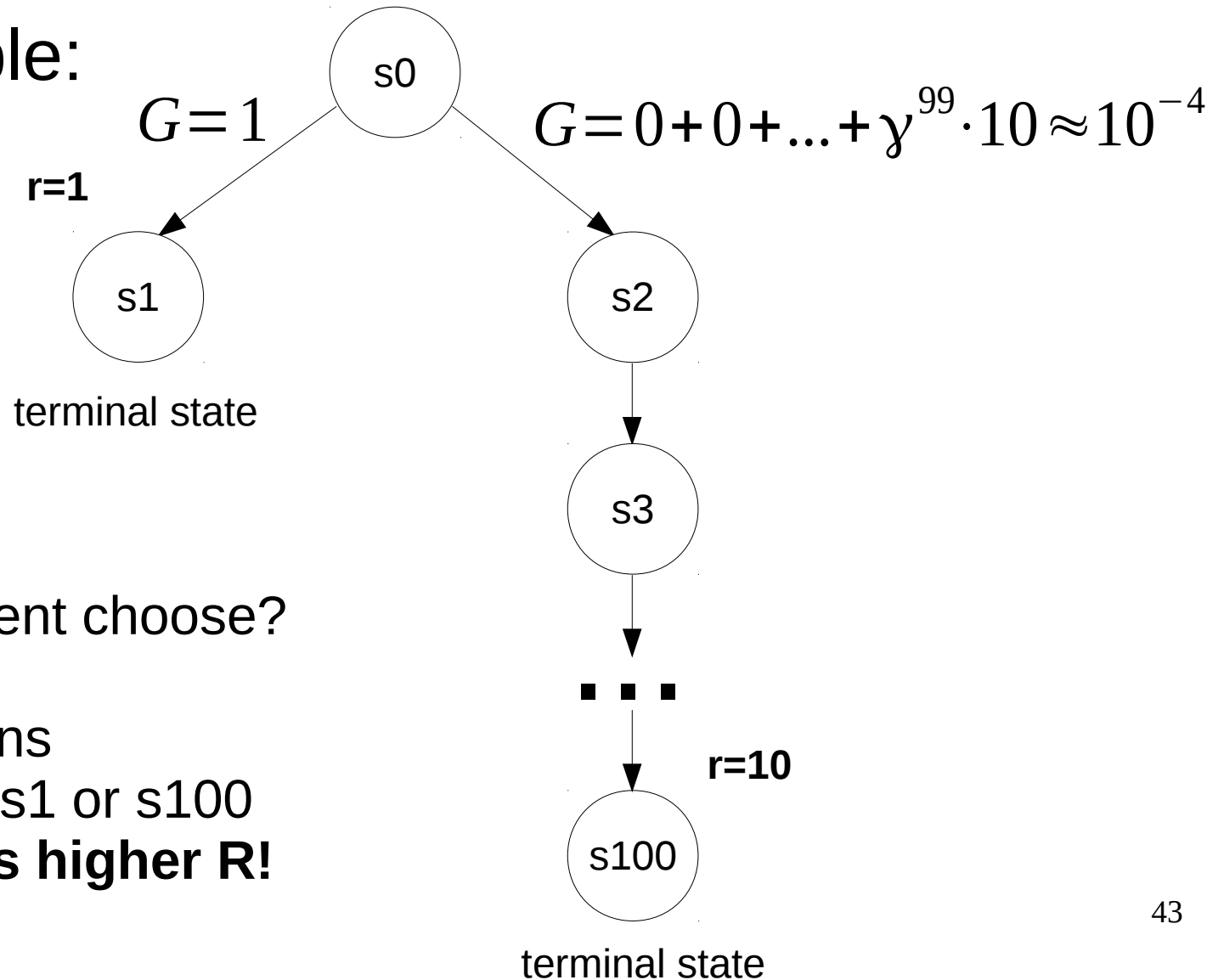


**What path will agent choose?**

- $\gamma=0.9$
- arrows = actions
- game ends at s1 or s100

# Discounted reward **fails** #1

Trivial example:



What path will agent choose?

- $\gamma=0.9$
- arrows = actions
- game ends at s1 or s100
- **left action has higher R!**

# Discounted reward **fails** #2

## Deephack'17 qualification round, Atari Skiing



- You steer the red guy
- Session lasts ~5k steps
- You get -3~-7 reward each tick (faster game = better score)
- At the end of session, you get up to  $r=-30k$  (based on passing gates, etc.)
- Q-learning with  $\gamma=0.99$  fails it doesn't learn to pass gates

**What's the problem?**

# Discounted reward **fails** #2

## Deephack'17 qualification round, Atari Skiing



- You steer the red guy
- Session lasts ~5k steps
- You get -3~-7 reward each tick (faster game = better score)
- At the end of session, you get up to  $r=-30k$  (based on passing gates, etc.)
- Q-learning with  $\gamma=0.99$  fails

# Discounted reward **fails** #3

## CoastRunner7 experiment (openAI)



- You control the boat
- Rewards for getting to checkpoints
- Rewards for collecting bonuses
- What could possibly go wrong?
- “Optimal” policy video:  
<https://www.youtube.com/watch?v=tlOIHko8ySg>

# Nuts and bolts: MC vs TD

## Monte-carlo

- Ignores intermediate rewards  
doesn't need  $\gamma$  (discount)
- Needs full episode to learn  
Infinite MDP are a problem
- Doesn't use Markov property  
Works with non-markov envs

## Temporal Difference

- Uses intermediate rewards  
trains faster under right  $\gamma$
- Learns from incomplete episode  
Works with infinite MDP
- Requires markov property  
Non-markov env is a problem



# Nuts and bolts: discount

- Effective horizon  $1 + \gamma + \gamma^2 + \dots = \frac{1}{(1 - \gamma)}$

Heuristic: your agent stops giving a damn in *this many* turns.

Typical values:

- $\gamma=0.9$ , 10 turns
- $\gamma=0.95$ , 20 turns
- $\gamma=0.99$ , 100 turns
- $\gamma=1$ , infinitely long

Higher  $\gamma$  = less stable algorithm.

$\gamma=1$  only works for episodic MDP (finite amount of turns).



# Nuts and bolts: discount

- Effective horizon  $1 + \gamma + \gamma^2 + \dots = \frac{1}{(1 - \gamma)}$

Heuristic: your agent stops giving a damn in *this many* turns.

- Atari Skiing, reward was delayed by in 5k steps
- $\gamma=0.99$  is not enough
- $\gamma=1$  and a few hacks works better
- Or use a better reward function



Let's write some code!