# Practical RL

Episode 3; 2019

# Model-free reinforcement learning

Yandex Data Factory

LAMBDA

British Hedgehog Preservation Society

# Recap: discounted rewards


Objective:

Total action value

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

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

γ ~ patience

Cake tomorrow is γ as good as now

Reinforcement learning:

- Find policy that maximizes the expected reward

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

# Recap: discounted rewards


Objective:
Total action value

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

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

**Trivia: which γ corresponds to “only current reward matters”?**

Reinforcement learning:
- Find policy that maximizes the expected reward

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

# Recap: discounted rewards


Objective:
Total reward

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

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

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)?**

# Previously...

- **V(s)** and **V*(s,a)**

**WTF is V(s)?**

# Previously...

- **V(s)** and **V*(s,a) –** state values
- know V* and P(s'|s,a) → know optimal policy
- We can learn V* with dynamic programming

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

# Recap: notation

- $\mathbf{V_\pi(s)}$ – expected G from state **s** if you follow **π**
- $\mathbf{V^*(s)}$ – expected G from state s if you follow **π***

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

# Recap: notation

- **$V_\pi$(s)** – expected G from state **s** if you follow **π**
- **V*(s)** – expected G from state s if you follow **π***

- **$Q_\pi$(s,a)** – expected G from state **s**
  - if you start by taking action **a**
  - and follow **π** from next state on

- **Q*(s,a)** – guess what it is :)

# Recap: notation

- $\mathbf{V_\pi(s)}$ – expected G from state **s** if you follow **π**
- $\mathbf{V^*(s)}$ – expected G from state s if you follow **π***

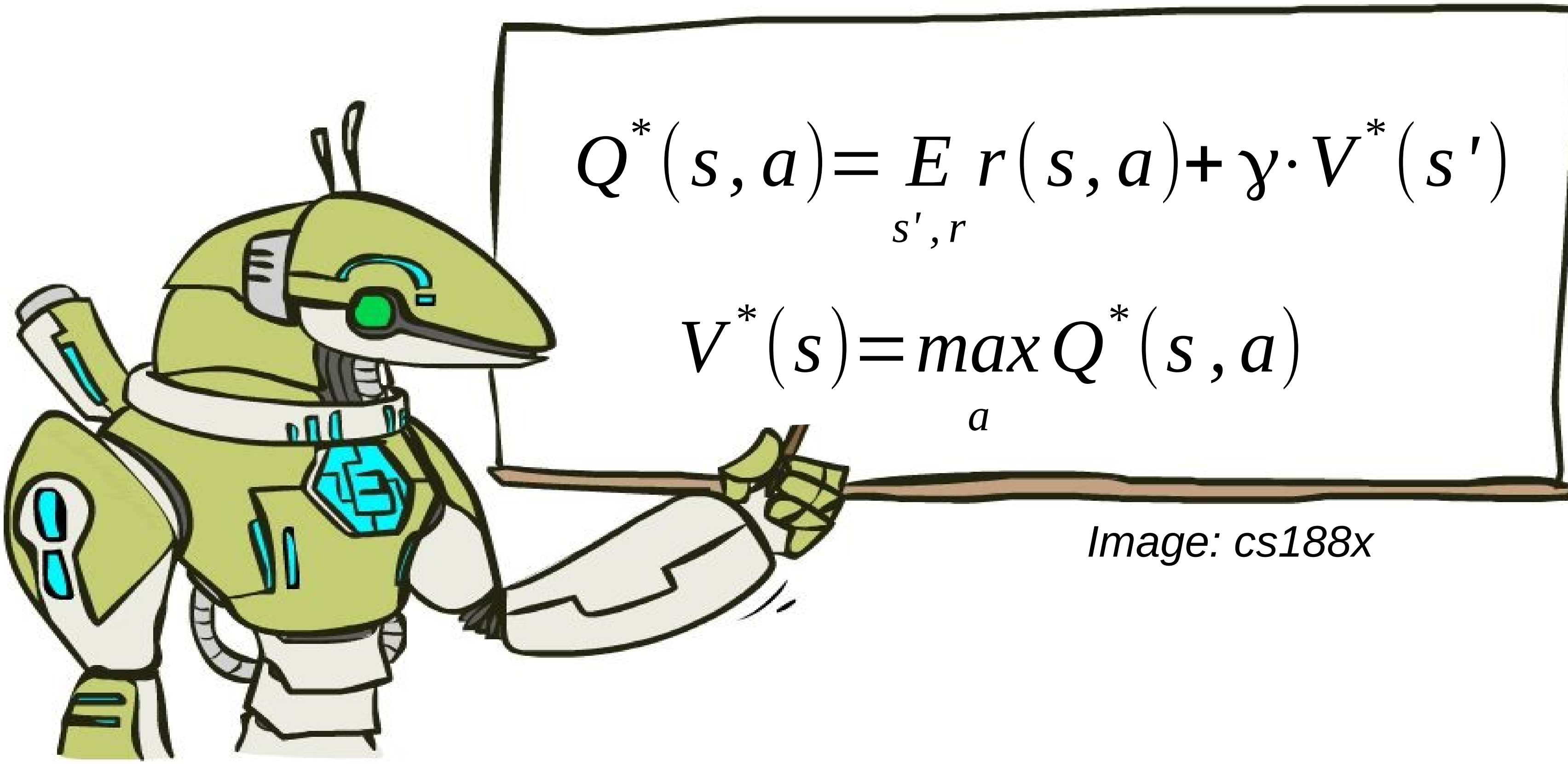
- $\mathbf{Q_\pi(s,a)}$ – expected G from state **s**
  - if you start by taking action **a**
  - and follow **π** from next state on

- $\mathbf{Q^*(s,a)}$ – same as $\mathbf{Q_\pi(s,a)}$ where $\boldsymbol{\pi = \pi^*}$

# Trivia

- Assuming you know **Q*(s,a)**,
  - how do you compute **π***
  - how do you compute **V*(s)**?
- Assuming you know V(s)
  - how do you compute **Q(s,a)?**

# To sum up



$$Q^*(s,a) = \mathop{E}_{s',r} r(s,a) + \gamma \cdot V^*(s')$$

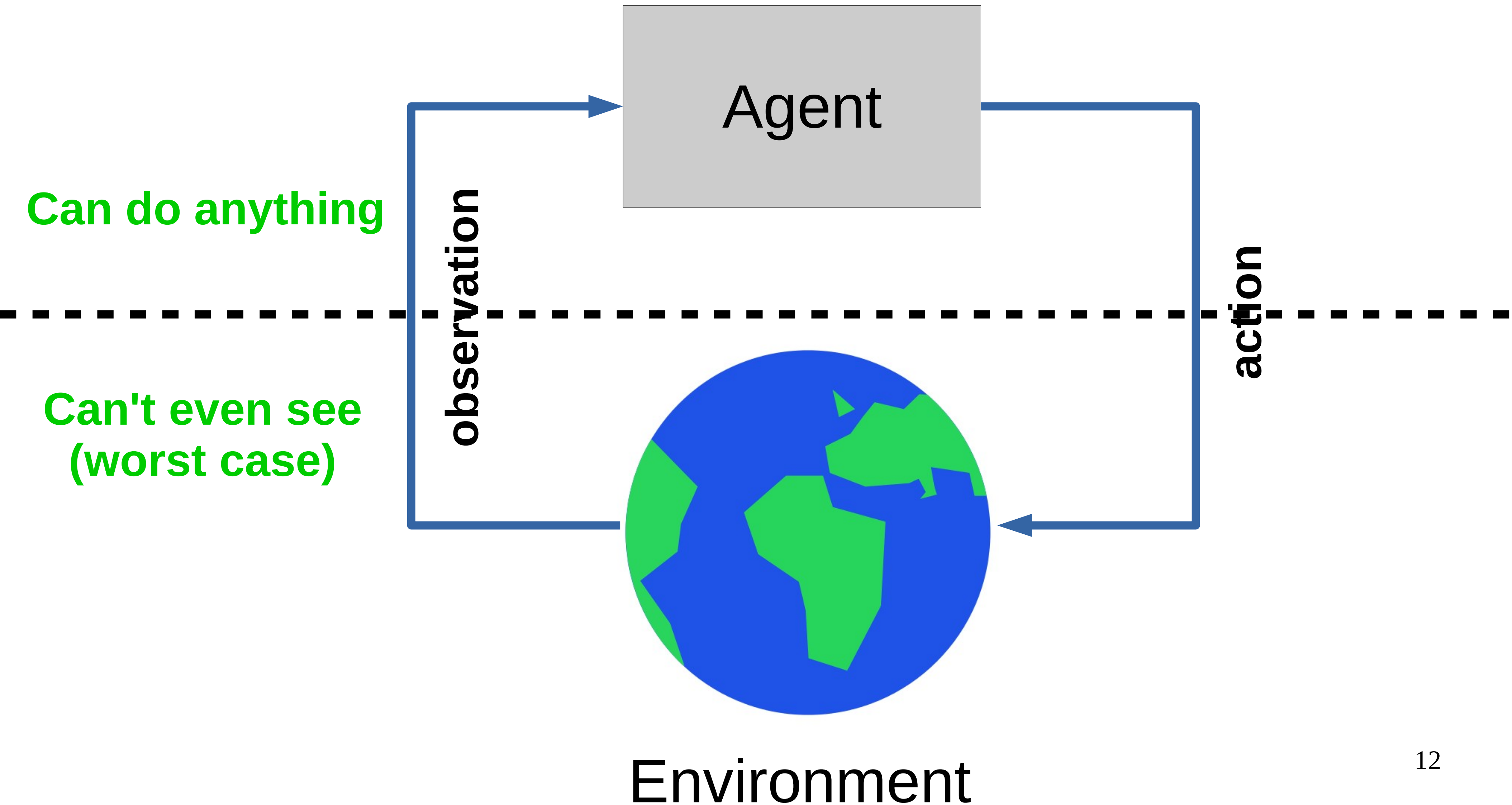
$$V^*(s) = \max_a Q^*(s,a)$$

*Image: cs188x*

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

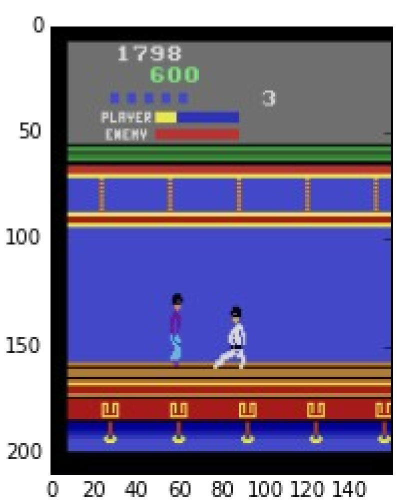
$$\pi(s) : argmax_a Q(s,a)$$

# Decision process in the wild

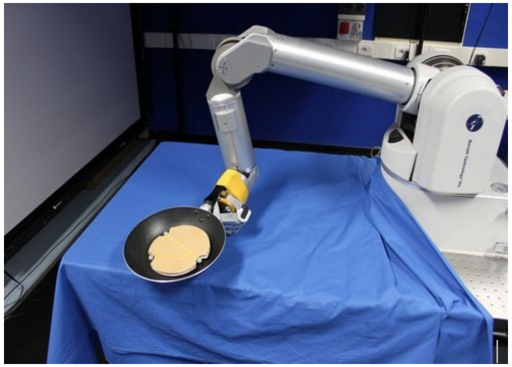


# Decision process in the wild


**Can do anything**







Model-free reinforcement learning:

We don't know actual

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

**Whachagonnado?**

Model-free reinforcement learning:

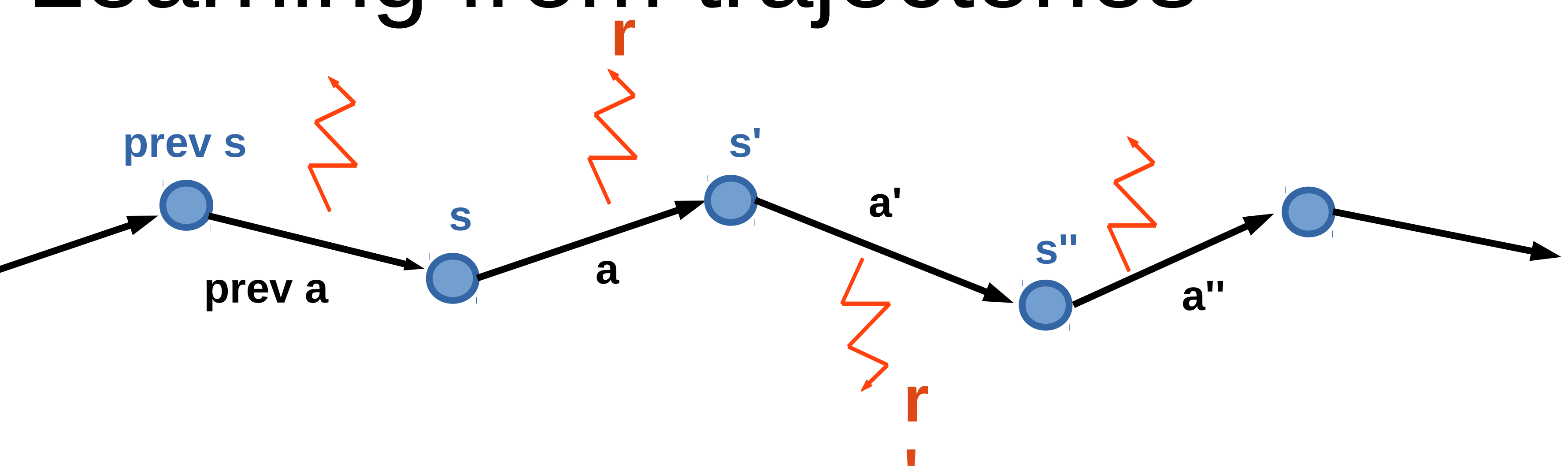
We don't know actual

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

Learn it?

Get rid of it?

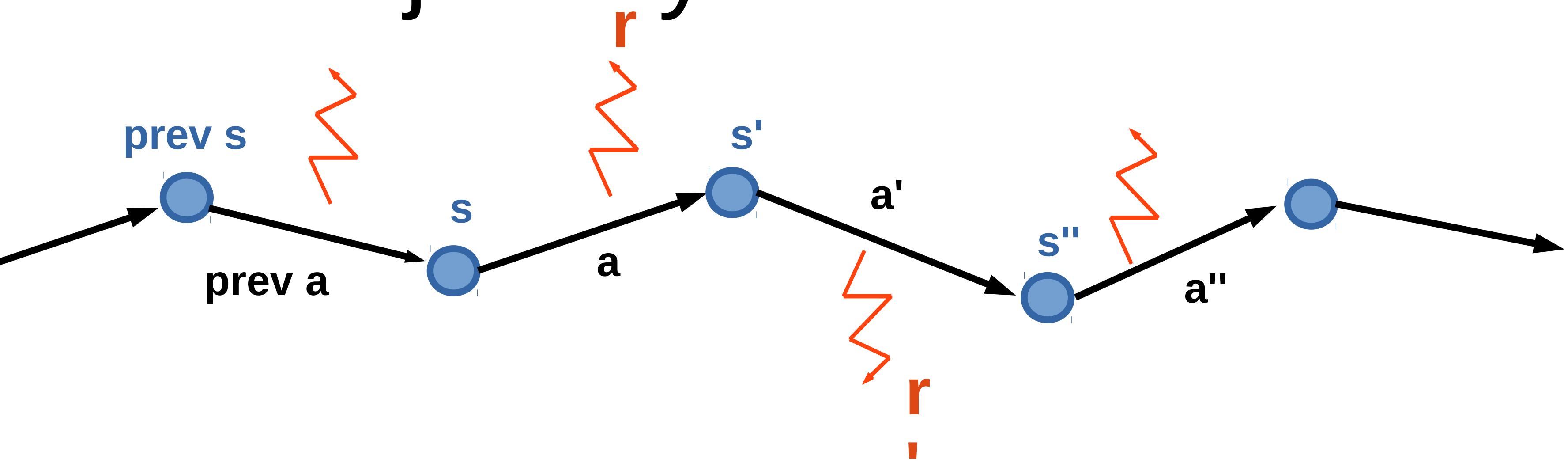
# Learning from trajectories




**Model-based:** you know P(s'|s,a)
- can apply dynamic programming
- can plan ahead

**Model-free:** you can sample trajectories
- can try stuff out
- insurance not included

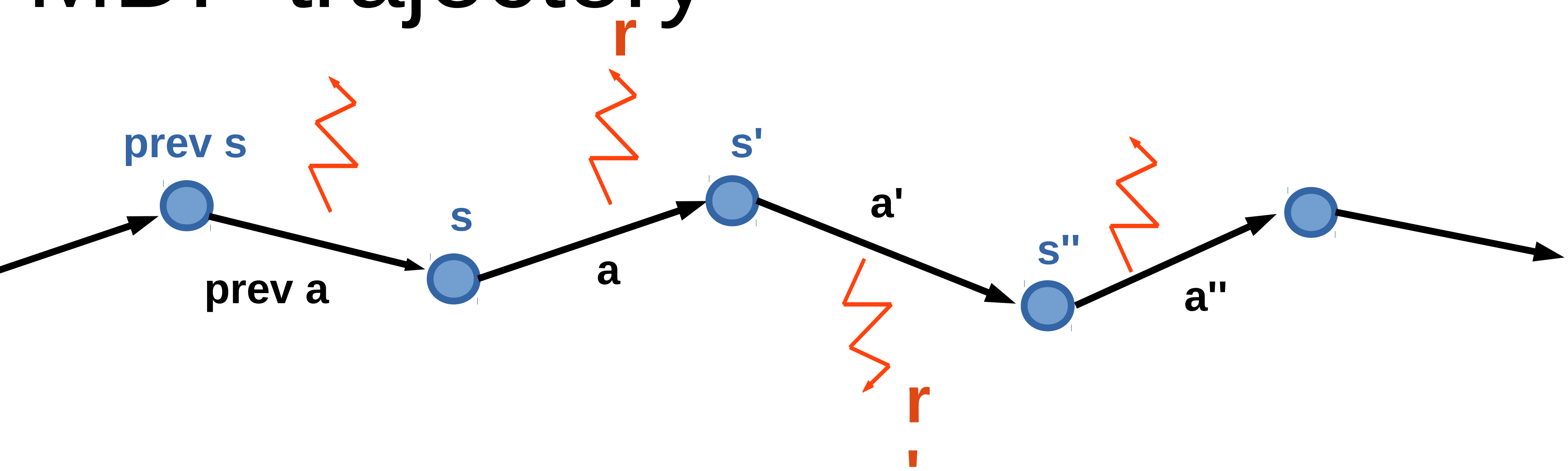
# MDP trajectory




- Trajectory is a sequence of
  - states (s)
  - actions (a)
  - rewards (r)

- We can only sample trajectories

# MDP trajectory

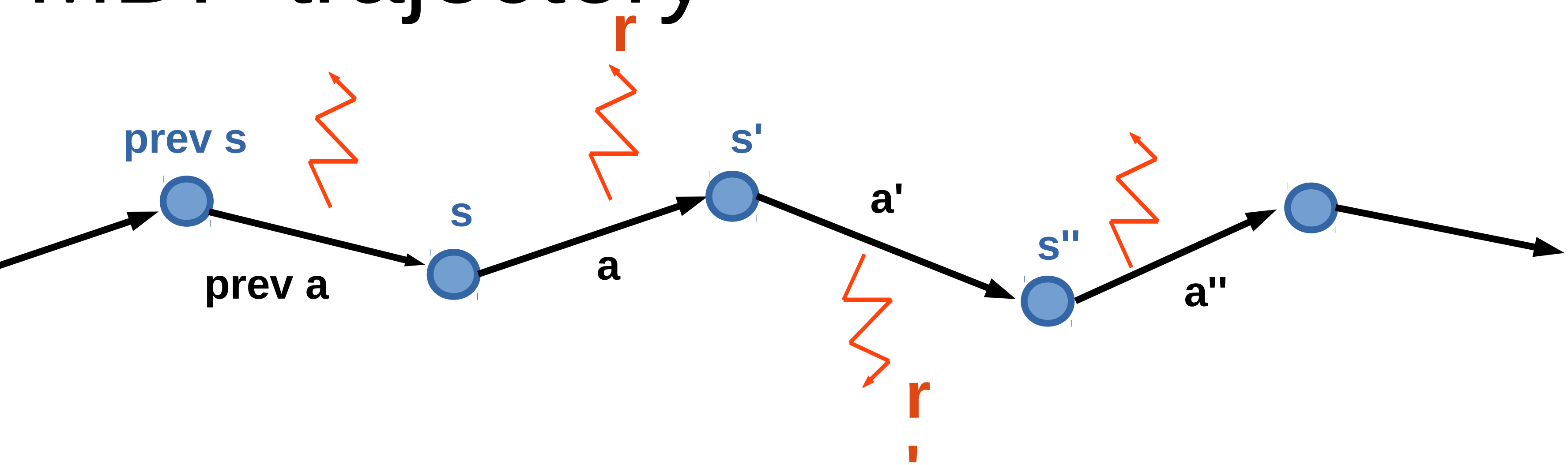



- Trajectory is a sequence of
  - states (s)
  - actions (a)
  - rewards (r)

**Q:** What to learn?
V(s) or Q(s,a)

- We can only sample trajectories

# MDP trajectory




- Trajectory is a sequence of
  - states (s)
  - actions (a)
  - rewards (r)

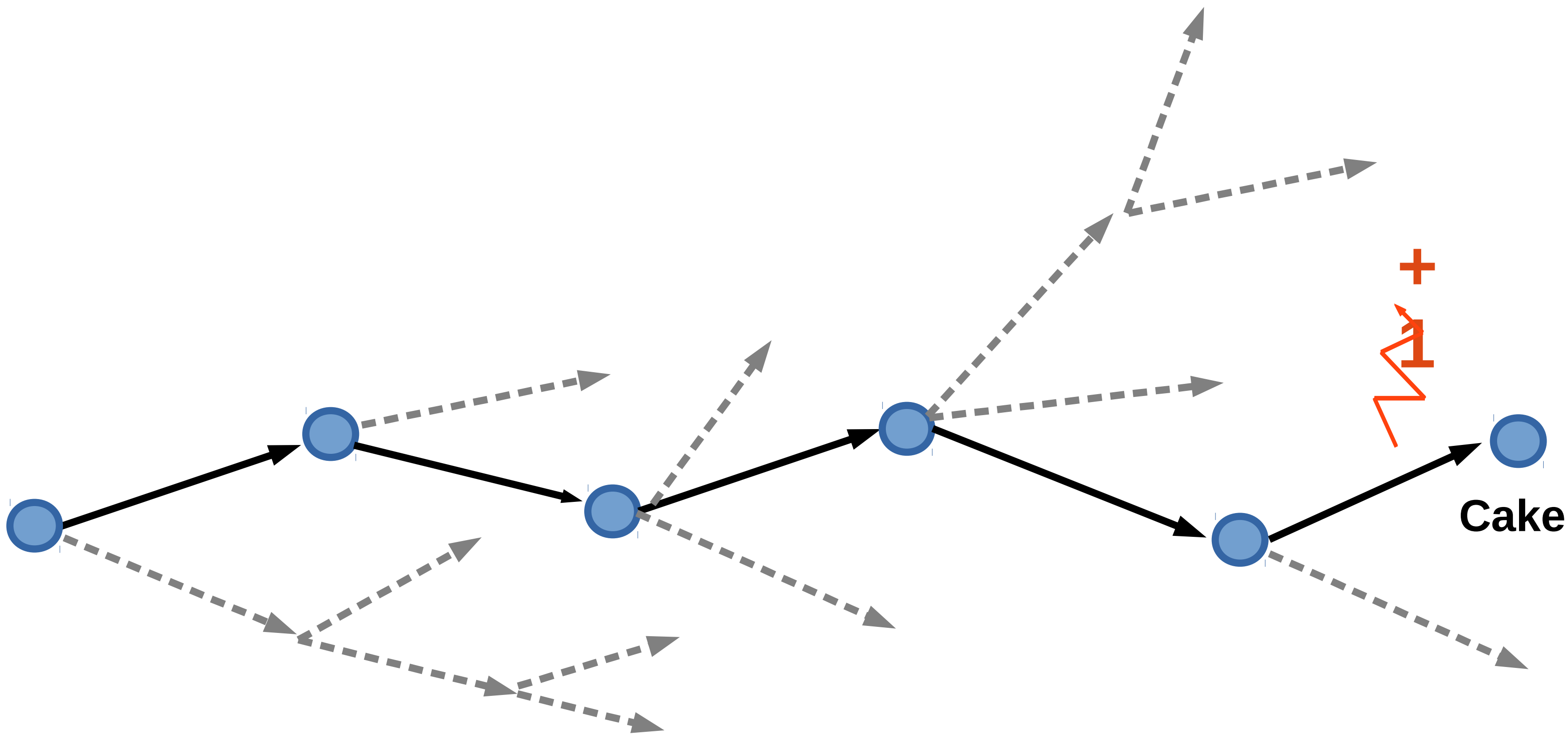
**Q:** What to learn?
V(s) or <u>Q(s,a)</u>

V(s) is useless
without P(s'|s,a)

- We can only sample trajectories

# Idea 1: monte-carlo

- Get all trajectories containing particular (s,a)
- Estimate G(s,a) for each trajectory
- Average them to get expectation



# Idea 1: monte-carlo

- Get all trajectories containing particular (s,a)
- Estimate G(s,a) for each trajectory
- Average them to get expectation

**takes a lot of sessions**

*Image: super meat boy*

# Idea 2: temporal difference

- Remember we can improve Q(s,a) iteratively!

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

# Idea 2: temporal difference

- Remember we can improve Q(s,a) iteratively!

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

**That's Q*(s,a)**

**That's value for π***
aka optimal policy

# Idea 2: temporal difference

- Remember we can improve Q(s,a) iteratively!

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

**That's Q*(s,a)**

**That's something we don't have**

**That's value for π***
aka optimal policy

**What do we do?**

# Idea 2: temporal difference

# Idea 2: temporal difference

- Replace expectation with sampling

$$\underset{r_t, s_{t+1}}{E} r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a') \approx \frac{1}{N} \sum_i r_i + \gamma \cdot max_{a'} Q(s_i^{next}, a')$$

# Idea 2: temporal difference

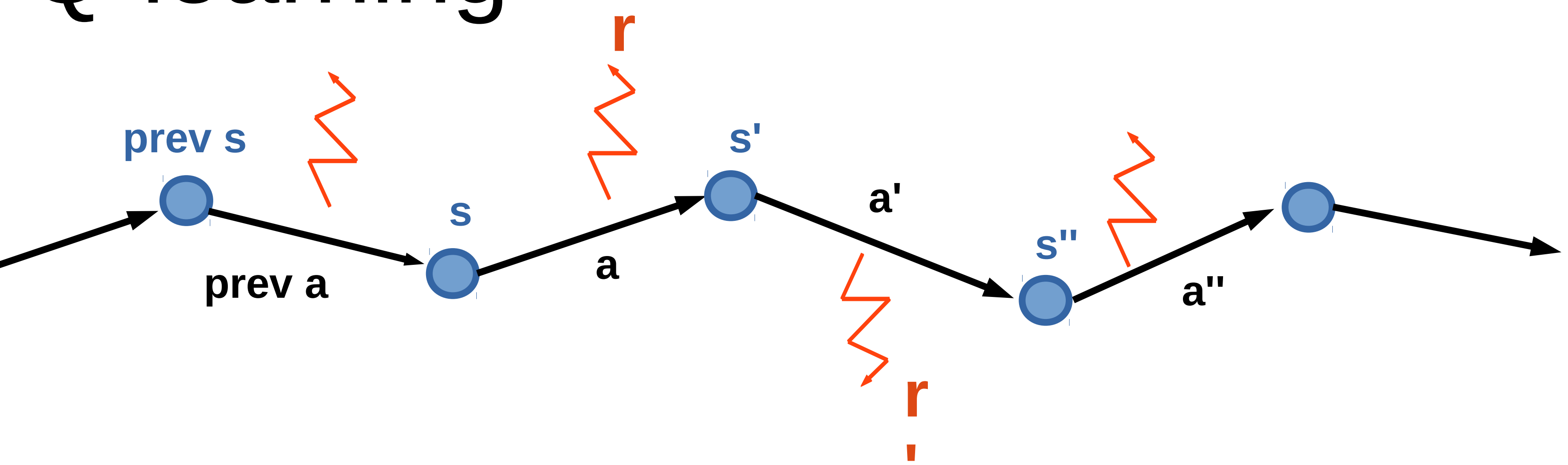
- Replace expectation with sampling

$$\underset{r_t, s_{t+1}}{E} r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a') \approx \frac{1}{N} \sum_i r_i + \gamma \cdot max_{a'} Q(s_i^{next}, a')$$

- Use moving average with just one sample!

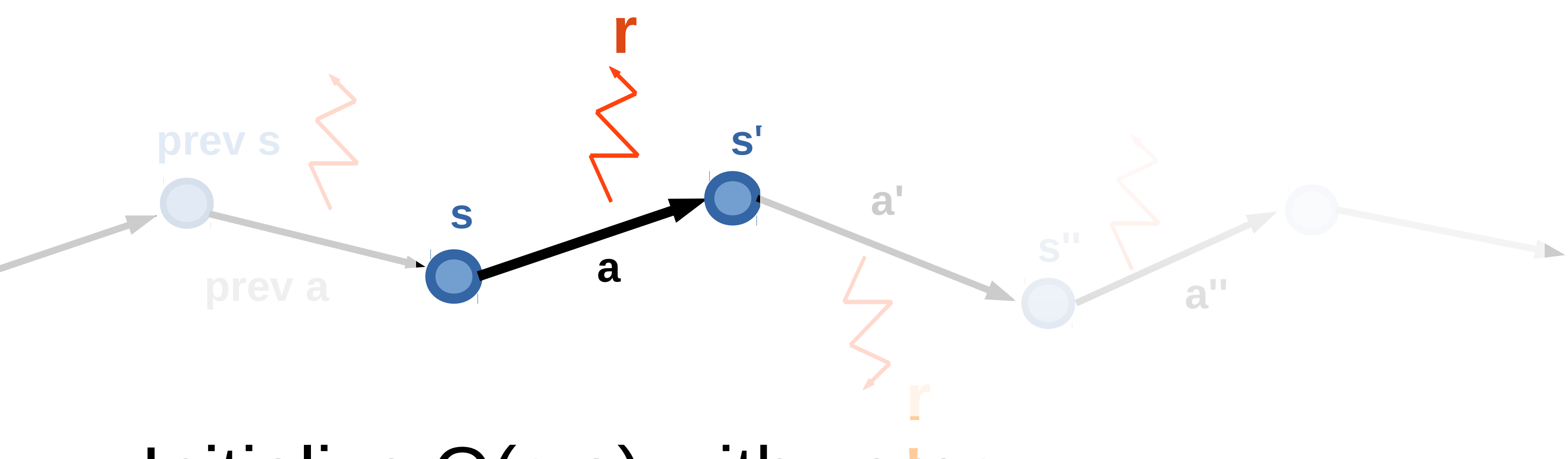
$$Q(s_t, a_t) \leftarrow \alpha \cdot (r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a')) + (1 - \alpha) Q(s_t, a_t)$$

# Q-learning




- Works on a sequence of
  - states (s)
  - actions (a)
  - rewards (r)

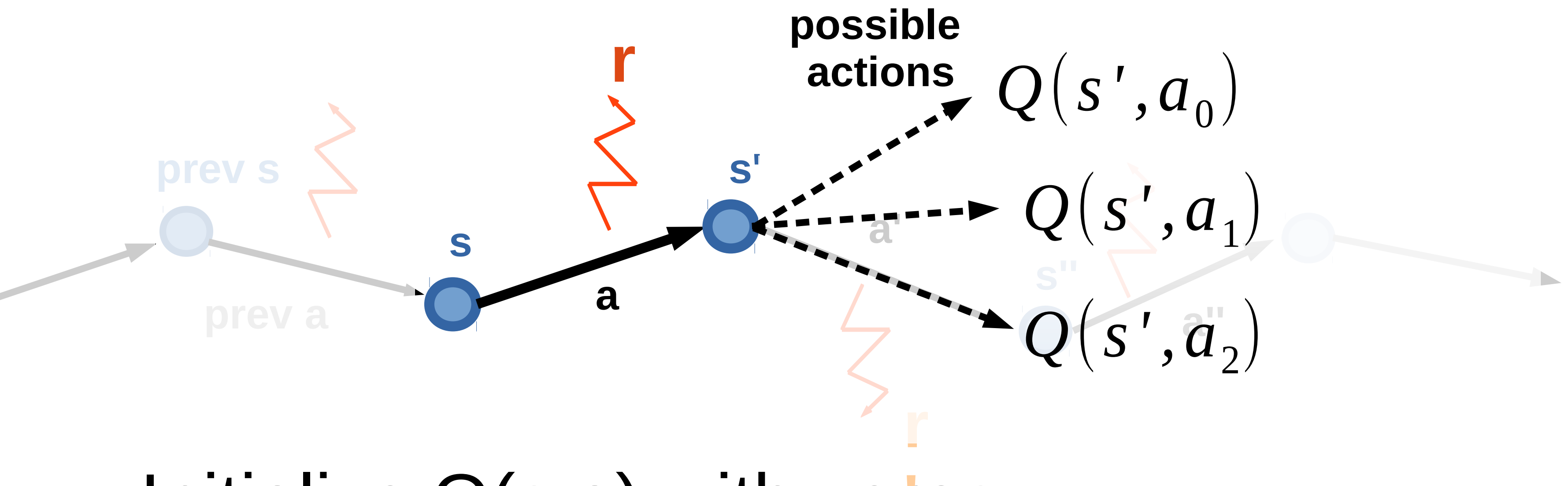
# Q-learning




Initialize Q(s,a) with zeros

- Loop:
  - Sample <**s**,**a**,**r**,**s'**> from env

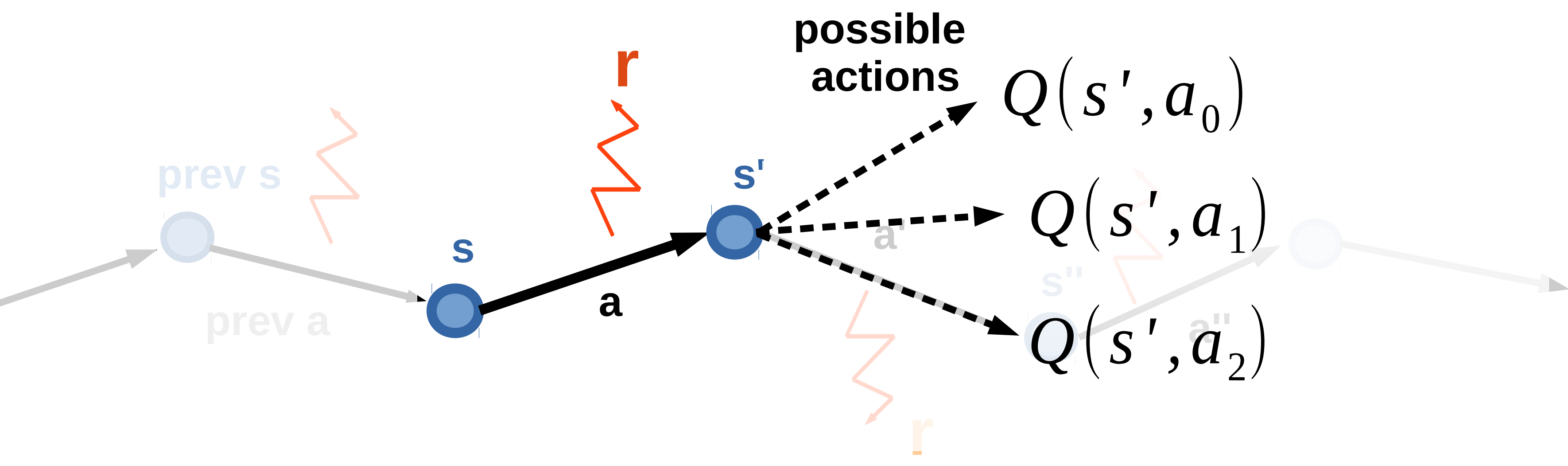
# Q-learning




Initialize Q(s,a) with zeros

- Loop:
  - Sample <**s**,**a**,**r**,**s'**> from env
  - Compute $\hat{Q}(s,a) = r(s,a) + \gamma \max_{a_i} Q(s',a_i)$

# Q-learning



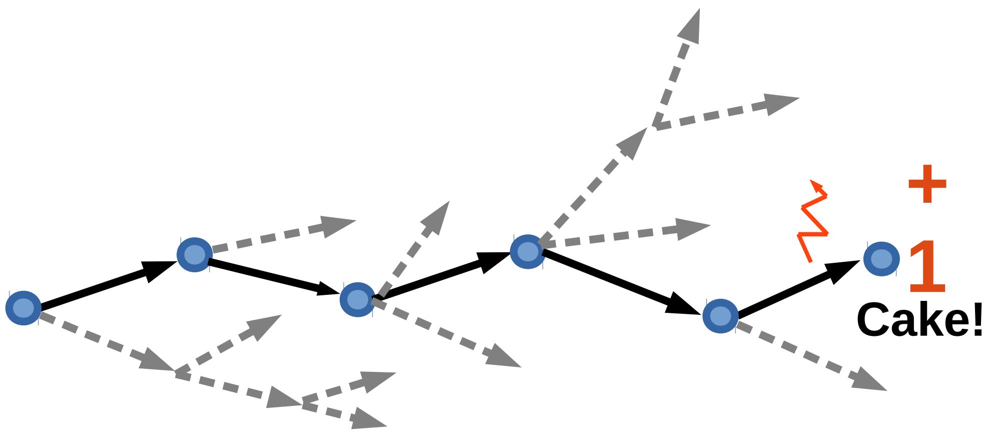

Initialize Q(s,a) with zeros

- Loop:
  - Sample <**s,a,r,s'**> from env
  - Compute $\hat{Q}(s,a) = r(s,a) + \gamma \max_{a_i} Q(s',a_i)$
  - Update $Q(s,a) \leftarrow \alpha \cdot \hat{Q}(s,a) + (1-\alpha) Q(s,a)$

# Recap

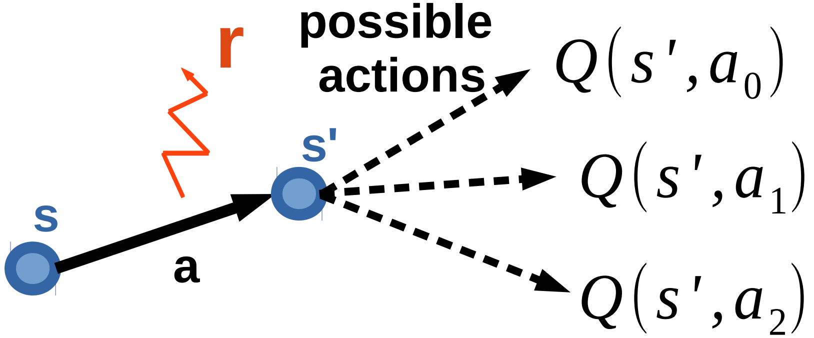
**Monte-carlo**

- Averages Q over sampled paths




**Temporal Difference**

- Uses recurrent formula for Q



# Nuts and bolts: MC vs TD

**Monte-carlo**

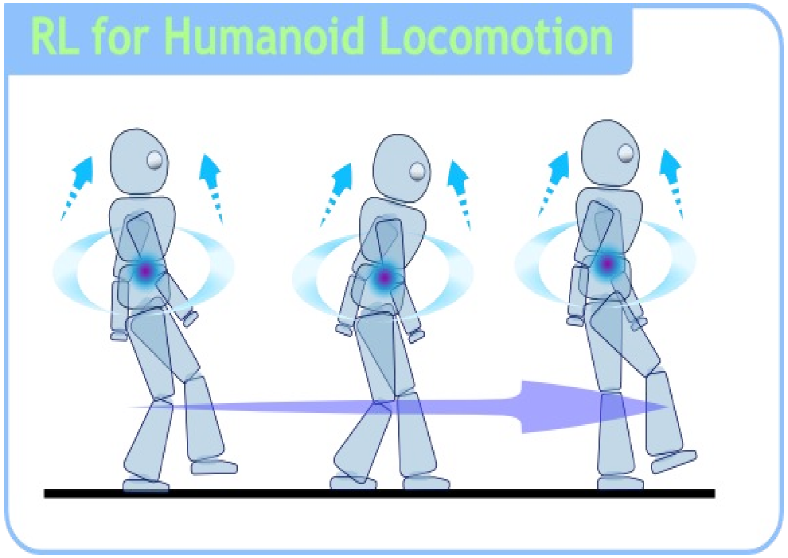
- Averages Q over sampled paths
- Needs full trajectory to learn
- Less reliant on markov property

**Temporal Difference**

- Uses recurrent formula for Q
- Learns from partial trajectory
  Works with infinite MDP
- Needs less experience to learn

# What could possibly go wrong?

Our mobile robot learns to walk.

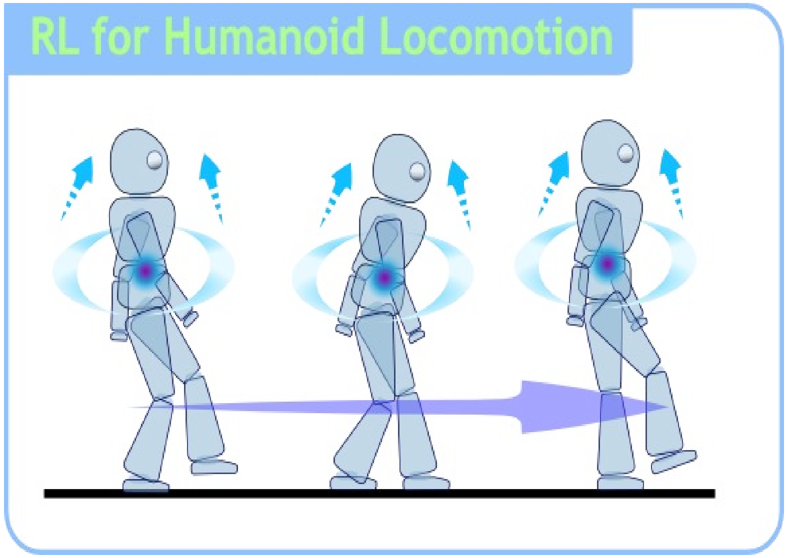



Initial Q(s,a) are zeros
robot uses argmax Q(s,a)

He has just learned to crawl with positive reward!

# What could possibly go wrong?

Our mobile robot learns to walk.




Initial Q(s,a) are zeros
robot uses argmax Q(s,a)

*Too bad, now he will never learn to walk upright =(*

# What could possibly go wrong?

New problem:

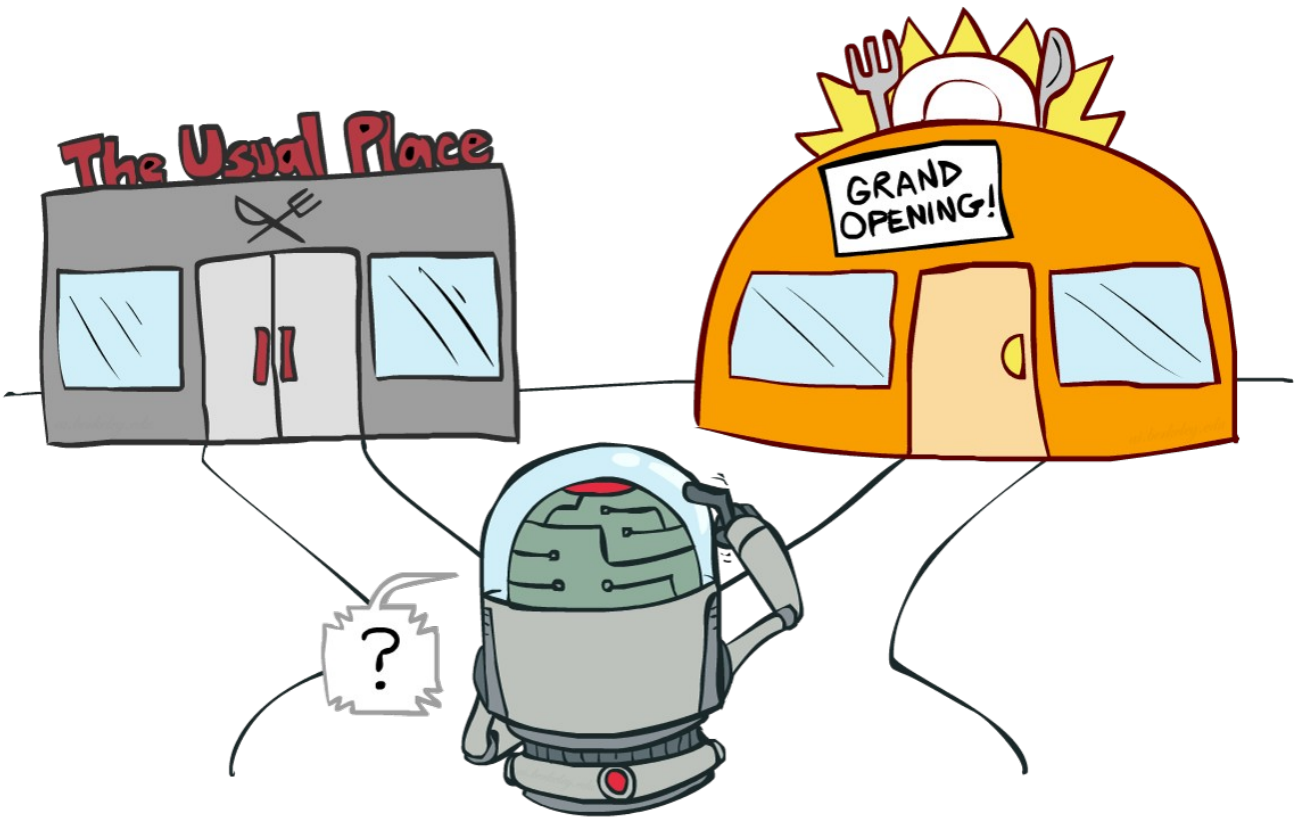
If our agent always takes “best” actions from his current point of view,

How will he ever learn that other actions may be better than his current best one?

Ideas?

# Exploration Vs Exploitation

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



# Exploration Vs Exploitation

Strategies:

- ε-greedy
  - With probability ε take random action; otherwise take optimal action.

# Exploration Vs Exploitation

Strategies:

- ε-greedy
  - With probability ε take random action; otherwise take optimal action.
- Softmax

  Pick action proportional to softmax of shifted normalized Q-values.

$$\pi(a|s) = softmax\left(\frac{Q(s,a)}{\tau}\right)$$

- More cool stuff coming later

# Exploration over time

**Idea:**

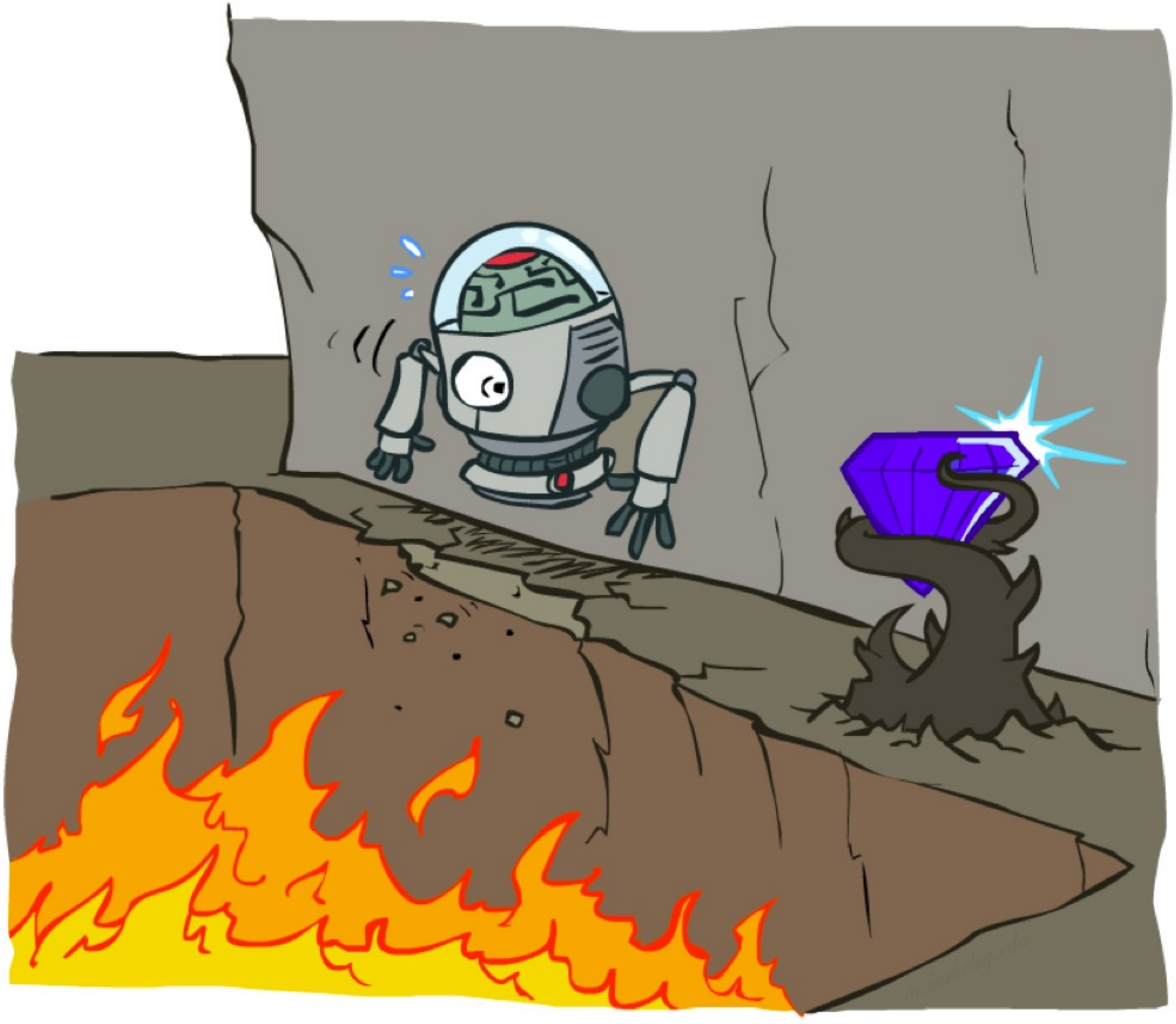
If you want to converge to optimal policy,
you need to gradually reduce exploration

**Example:**

Initialize ε-greedy $\varepsilon = 0.5$, then gradually reduce it

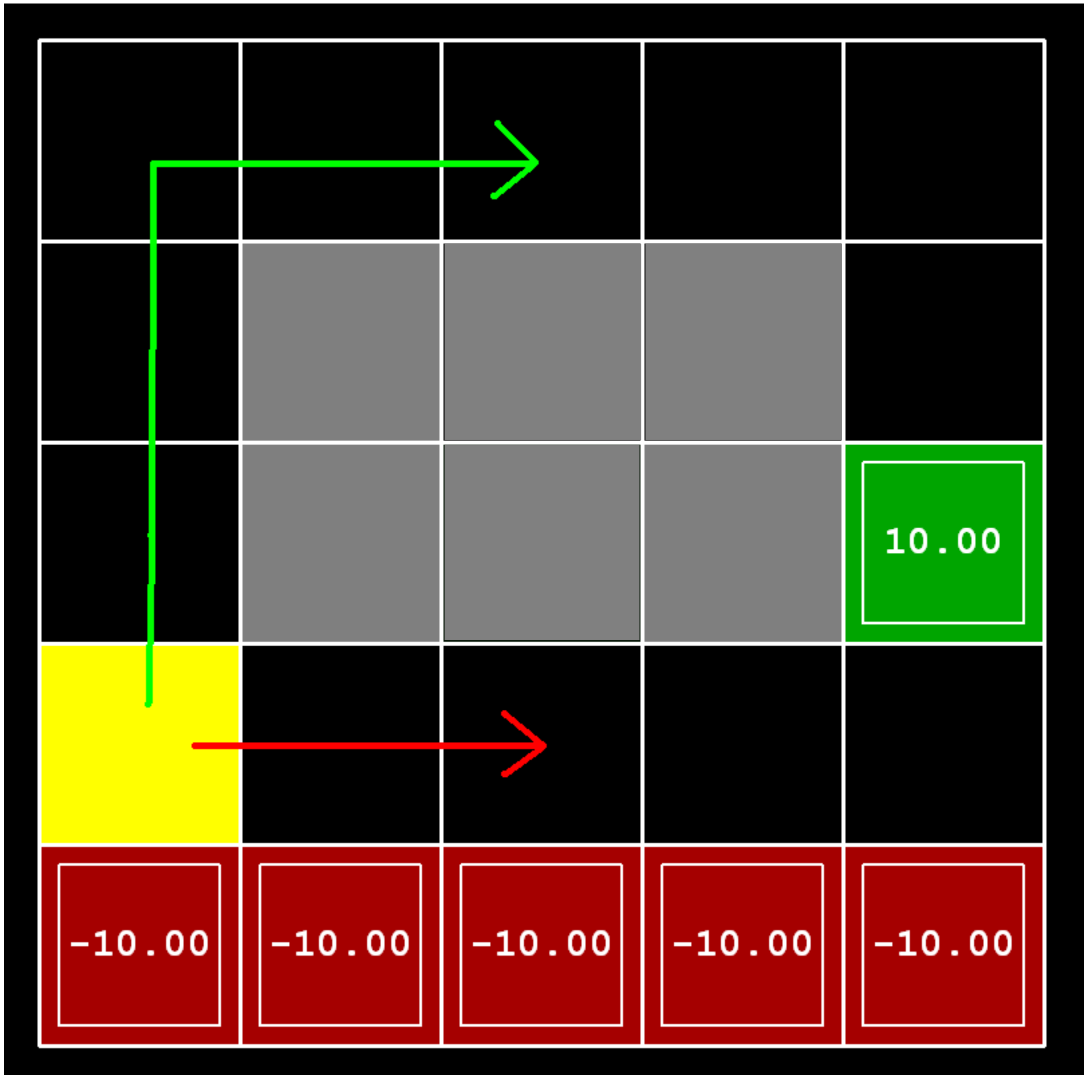
- If $\varepsilon \to 0$, it's **greedy in the limit**
- Be careful with non-stationary environments

# Cliff world



Picture from Berkeley CS188x

# Cliff world




Conditions

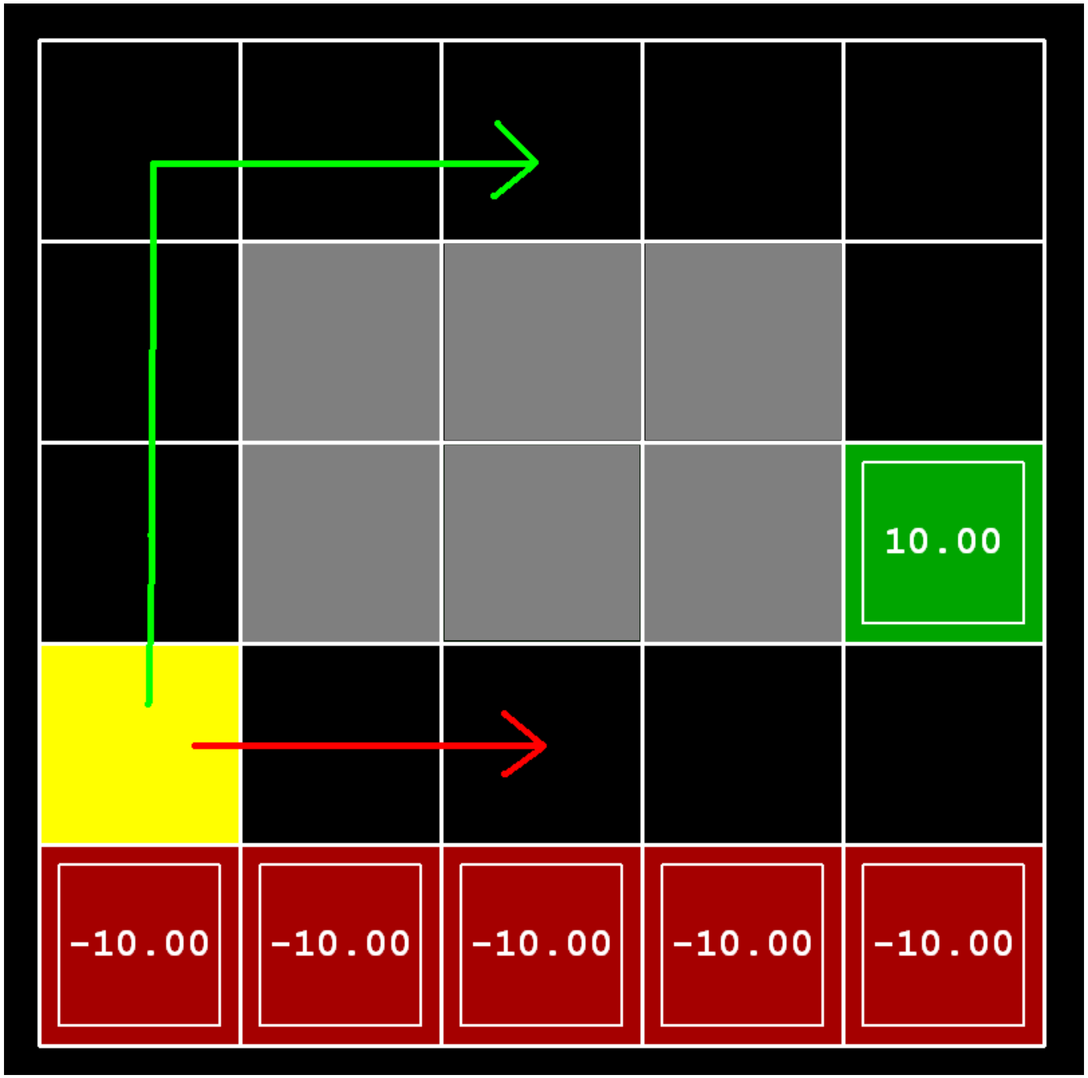
- Q-learning

$$\gamma = 0.99 \quad \epsilon = 0.1$$

- no slipping

**Trivia:**
What will q-learning learn?

# Cliff world




Conditions

- Q-learning

$\gamma = 0.99 \quad \epsilon = 0.1$

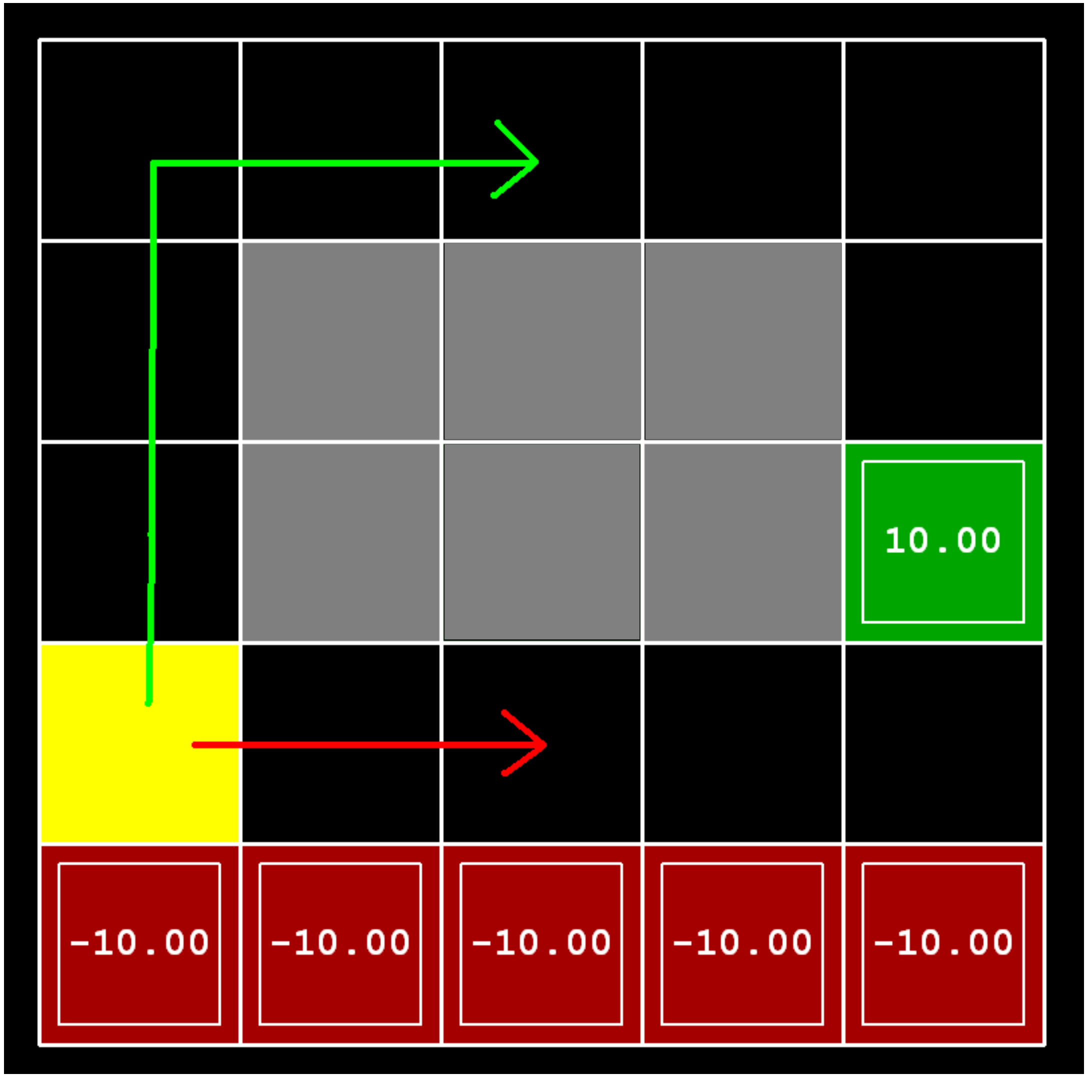
- no slipping

**Trivia:**
What will q-learning learn?

**follow the short path**

Will it maximize reward?

# Cliff world




Conditions

- Q-learning

$$\gamma=0.99 \quad \epsilon=0.1$$

- no slipping

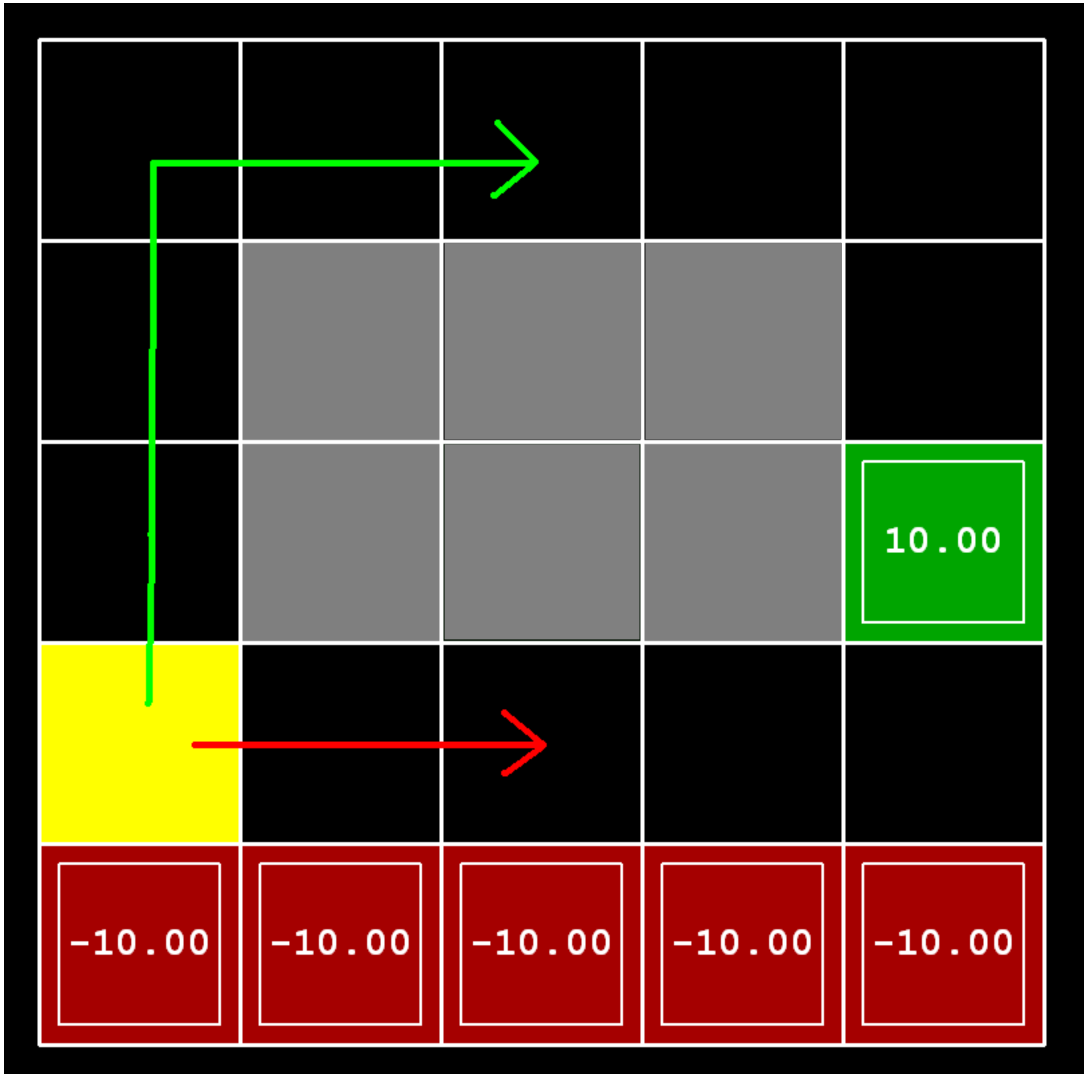
**Trivia:**
What will q-learning learn?

**follow the short path**

Will it maximize reward?

**no, robot will fall due to epsilon-greedy “exploration”**

# Cliff world




Conditions

- Q-learning

  $\gamma = 0.99 \quad \epsilon = 0.1$

- no slipping

**Decisions must account for actual policy!**
e.g. ε-greedy policy

# Generalized update rule

Update rule (from Bellman eq.)

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1 - \alpha) Q(s_t, a_t)$$

**"better Q(s,a)"**

# Q-learning VS SARSA

Update rule (from Bellman eq.)

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1-\alpha) Q(s_t, a_t)$$

**"better Q(s,a)"**

Q-learning

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

# Q-learning VS SARSA

Update rule (from Bellman eq.)

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1 - \alpha) Q(s_t, a_t)$$

**"better Q(s,a)"**

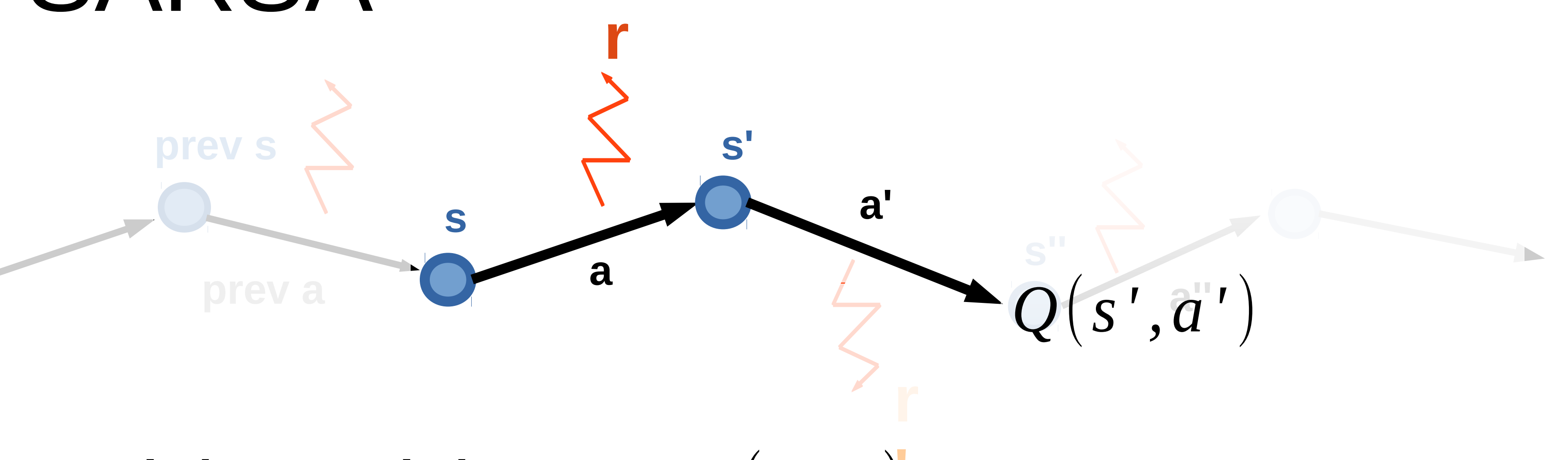
Q-learning

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

SARSA

$$\hat{Q}(s, a) = r(s, a) + \gamma \cdot \underset{a' \sim \pi(a'|s')}{E} Q(s', a')$$

# SARSA

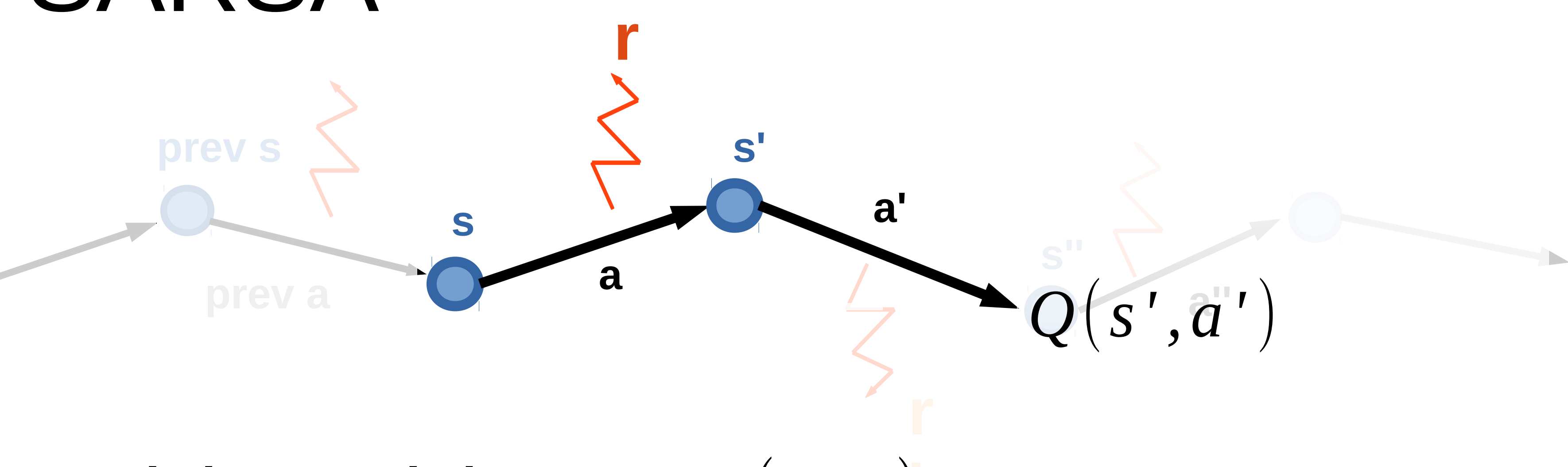



$$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$$

Loop:

- Sample <**s**,**a**,**r**,**s'**,**a'**> from env
- Compute $\hat{Q}(s,a) = r(s,a) + \gamma Q(s',a')$
- Update $Q(s,a) \leftarrow \alpha \cdot \hat{Q}(s,a) + (1-\alpha) Q(s,a)$

# SARSA




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

Loop:

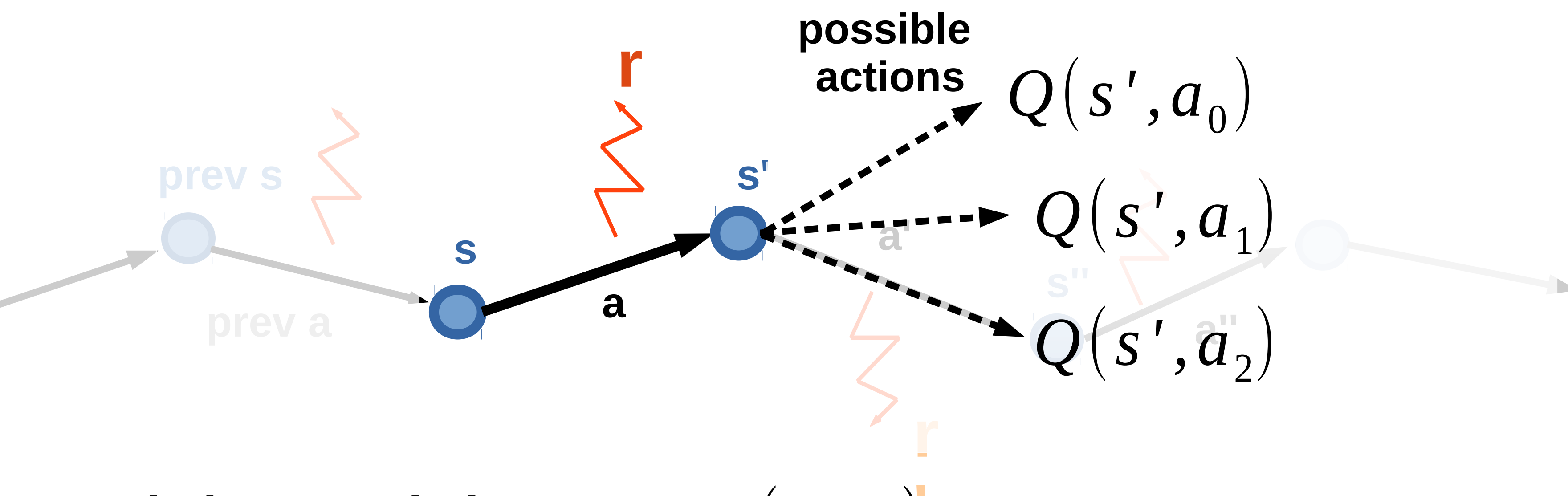
hence “SARSA”

- Sample <**s**,**a**,**r**,**s'**,**a'**> from env
- Compute $\hat{Q}(s,a) = r(s,a) + \gamma Q(s',\underline{a'})$

next action (not max)

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

# Expected value SARSA

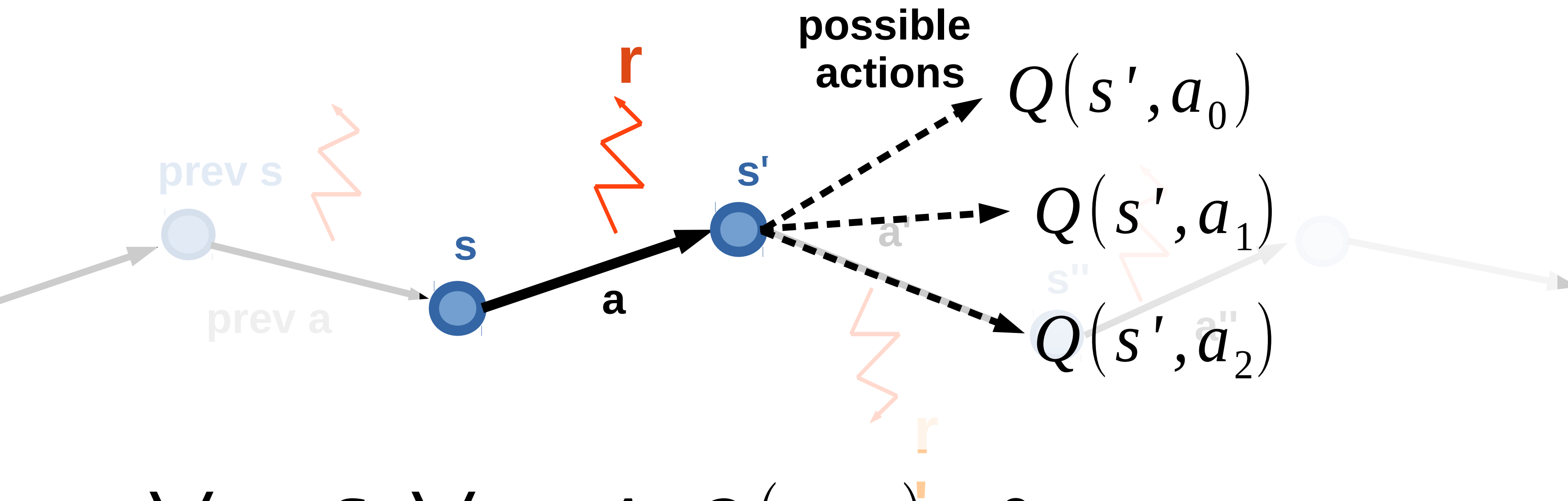



$$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$$

Loop:

- Sample <**s**,**a**,**r**,**s'**> from env
- Compute $\hat{Q}(s,a) = r(s,a) + \gamma \underset{a_i \sim \pi(a|s')}{E} Q(s',a_i)$
- Update $Q(s,a) \leftarrow \alpha \cdot \hat{Q}(s,a) + (1-\alpha) Q(s,a)$

# Expected value SARSA




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

Loop:

- Sample <**s**,**a**,**r**,**s'**> from env

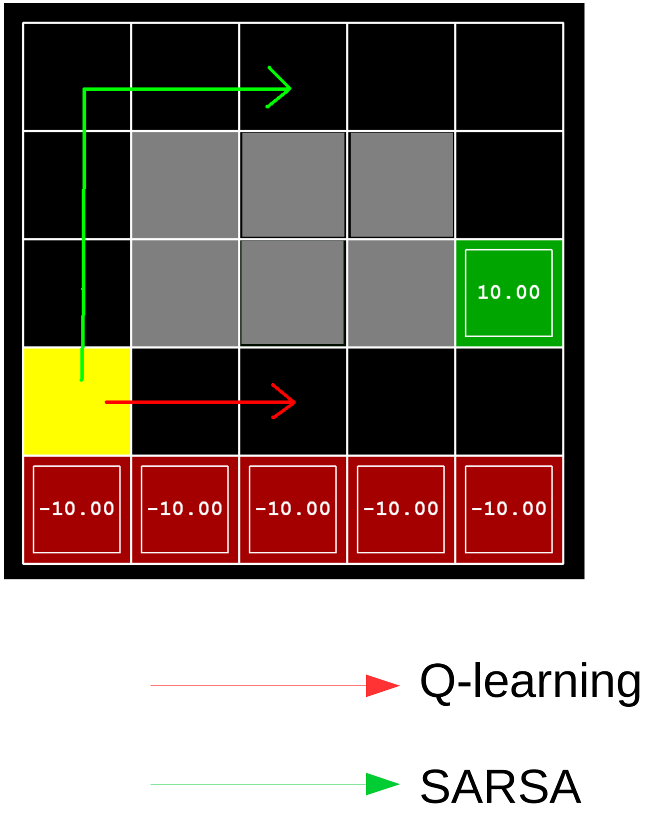
Expected value

- Compute $\hat{Q}(s,a) = r(s,a) + \gamma \underset{a_i \sim \pi(a|s')}{E} Q(s',a_i)$
- Update $Q(s,a) \leftarrow \alpha \cdot \hat{Q}(s,a) + (1-\alpha) Q(s,a)$

# Difference

- SARSA gets optimal rewards under current policy

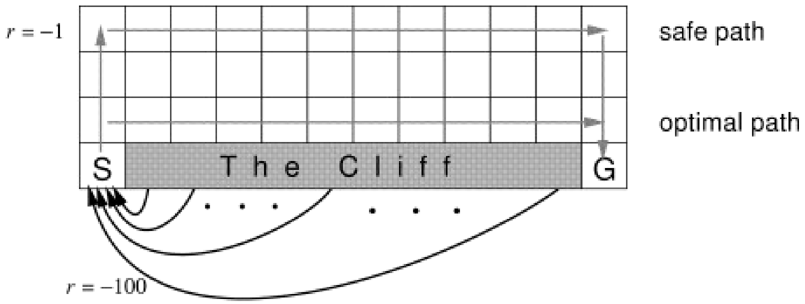
- Q-learning policy **would be** optimal under



# Difference

- SARSA converges to optimal policy

- Q-learning policy **would be** optimal without exploration




# On-policy vs Off-policy

## Two problem setups

**on-policy**

Agent **can** pick actions

- Most obvious setup :)
- Agent always follows his **own** policy

**off-policy**

Agent **can't** pick actions

- Learning with exploration, playing without exploration
- Learning from expert (expert is imperfect)
- Learning from sessions (recorded data)

# On-policy vs Off-policy

Two problem setups

| **on-policy** | **off-policy** |
| --- | --- |
| Agent **can** pick actions | Agent **can't** pick actions |
| – On-policy algorithms **can't** learn off-policy | – Off-policy algorithms **can** learn on-policy |
| | learn optimal policy even if agent takes random actions |

**Q:** which of Q-learning, SARSA and exp. val. SARSA will **only** work on-policy?

# On-policy vs Off-policy

## Two problem setups

**on-policy**

Agent **can** pick actions

- On-policy algorithms **can't** learn off-policy
- SARSA
- more later

**off-policy**

Agent **can't** pick actions

- Off-policy algorithms **can** learn on-policy
- Q-learning
- Expected Value SARSA

# On-policy vs Off-policy

## Two problem setups

**on-policy**

Agent **can** pick actions

- On-policy algorithms **can't** learn off-policy
- SARSA
- more coming soon

**off-policy**

Agent **can't** pick actions

- Off-policy algorithms **can** learn on-policy
- Q-learning
- Expected Value SARSA

# On-policy vs Off-policy

## Two problem setups

**on-policy**

Agent **can** pick actions

- On-policy algorithms **can't** learn off-policy
- SARSA
- more coming soon

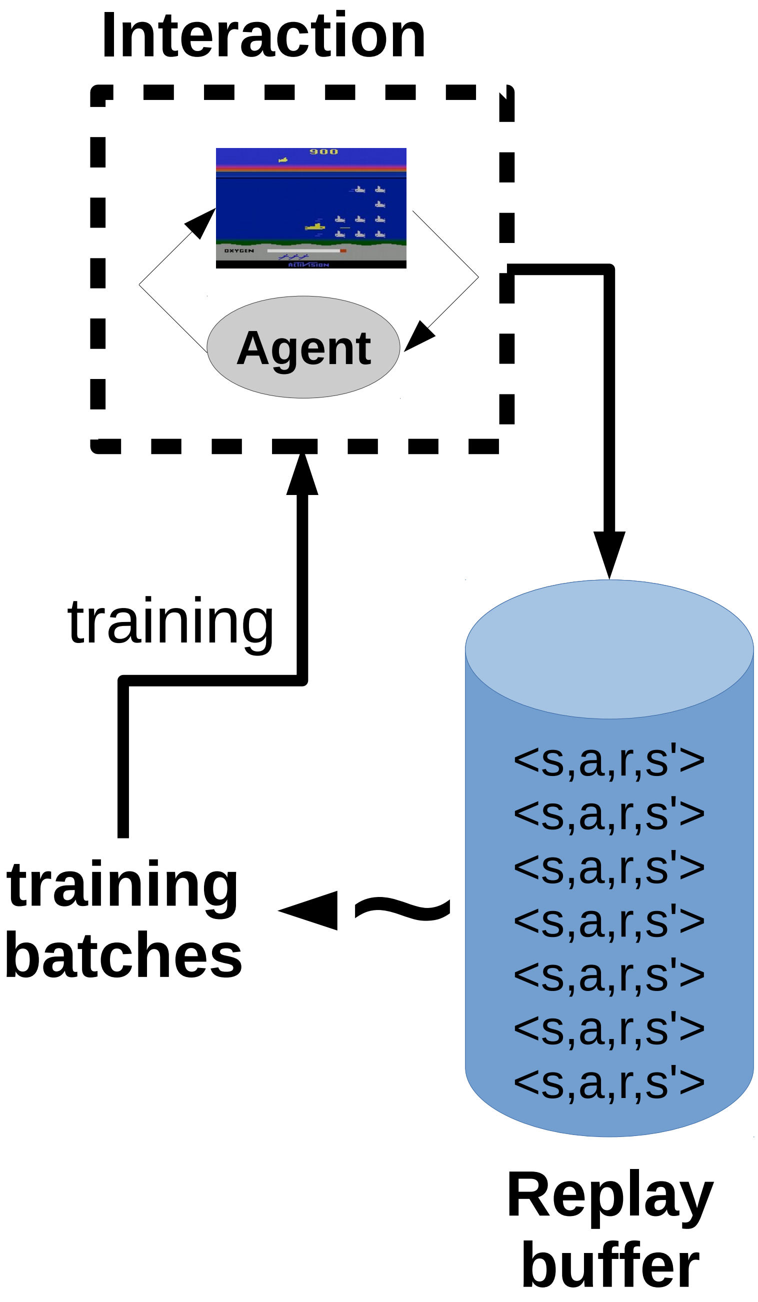
**off-policy**

Agent **can't** pick actions

- Off-policy algorithms **can** learn on-policy
- Q-learning
- Expected Value SARSA

# Experience replay

**Idea:** store several past interactions
*<s,a,r,s'>*
Train on random subsamples

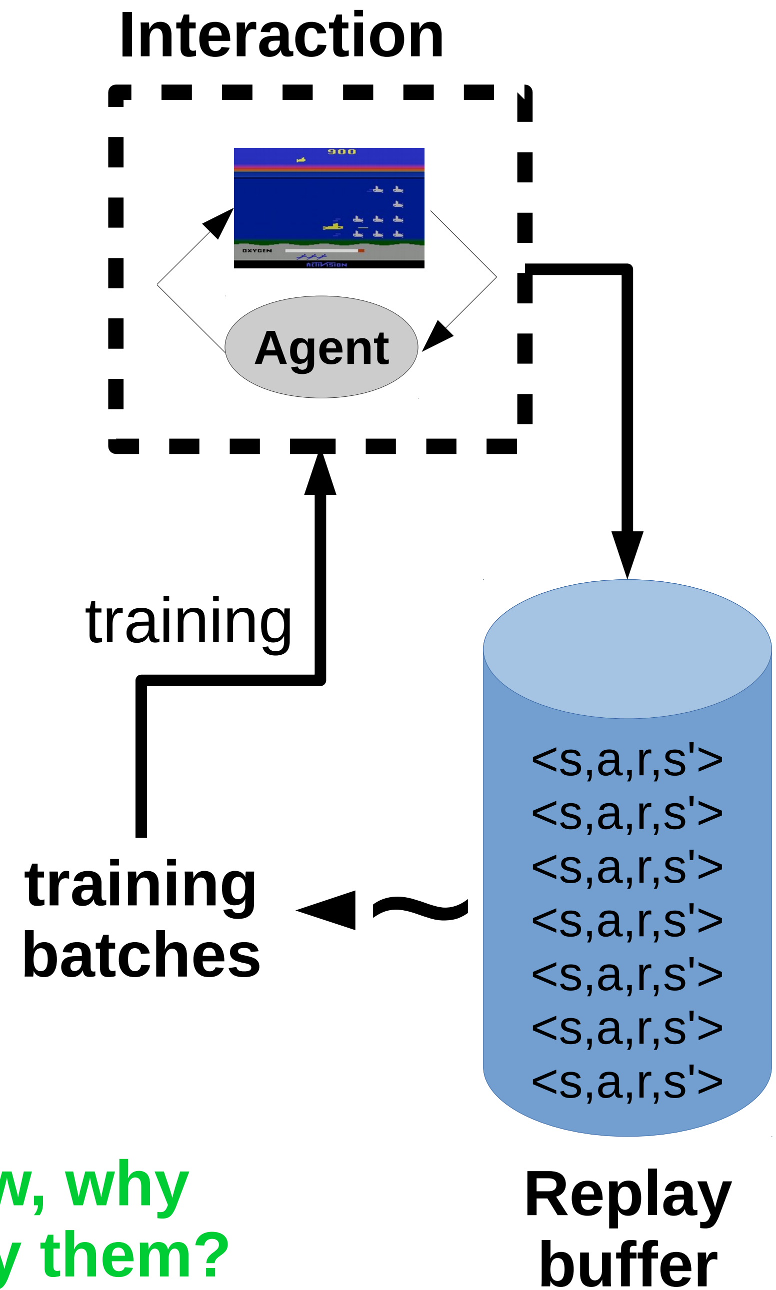


# Experience replay

**Idea:** store several past interactions
*<s,a,r,s'>*
Train on random subsamples

**Training curriculum:**
- play 1 step and record it
- pick N random transitions to train

**Profit:** you don't need to re-visit same (s,a) many times to learn it.




**Only works with off-policy algorithms!**

**Btw, why only them?**

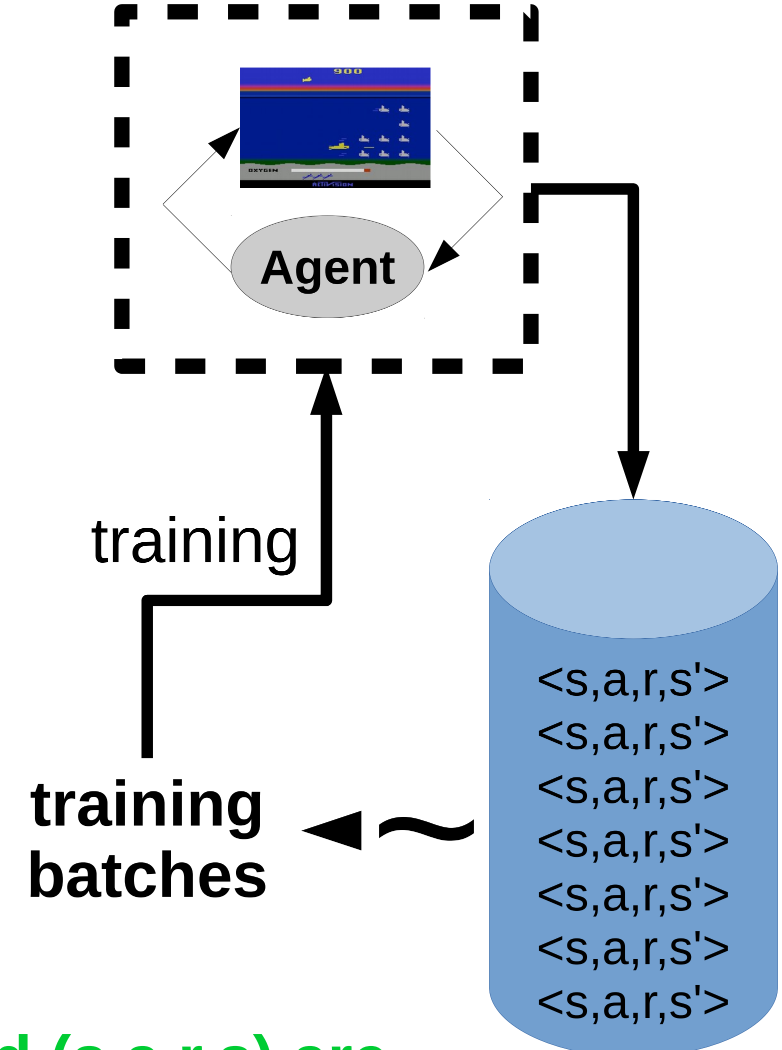
# Experience replay

**Idea:** store several past interactions
*<s,a,r,s'>*
Train on random subsamples

**Training curriculum:**
- play 1 step and record it
- pick N random transitions to train

**Profit:** you don't need to re-visit same (s,a) many times to learn it.

**Only works with off-policy algorithms!**




**Old (s,a,r,s) are from older/weaker version of policy!**

# N-step algorithms

Recall R?

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots =$$

$$= r_t + \gamma \cdot \left( r_{t+1} + \gamma \cdot r_{t+2} + \ldots \right) =$$

$$= r_t + \gamma \cdot R_{t+1} =$$

$$= r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot R_{t+2}$$

# N-step SARSA

- General formula

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1-\alpha) Q(s_t, a_t)$$

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1})$$

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 Q(s_{t+2}, a_{t+2})$$

# N-step SARSA

- General formula

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1-\alpha) Q(s_t, a_t)$$

**1-step SARSA**

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1})$$

**2-step SARSA**

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 Q(s_{t+2}, a_{t+2})$$

**3-step SARSA**

$\hat{Q}(s_t, a_t) =$ **???**

# N-step SARSA

- General formula

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1-\alpha) Q(s_t, a_t)$$

**1-step SARSA**

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1})$$

**2-step SARSA**

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 Q(s_{t+2}, a_{t+2})$$

**3-step SARSA**

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \gamma^3 Q(s_{t+3}, a_{t+3})$$

# N-step SARSA

- General formula

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1-\alpha) Q(s_t, a_t)$$

**1-step SARSA**

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1})$$

**N-step SARSA**

$$\hat{Q}(s_t, a_t) = \left[\sum_{\tau=t}^{\tau<t+n} \gamma^{\tau} r(s_{t+\tau}, a_{t+\tau})\right] + \gamma^{n} Q(s_{t+n}, a_{t+n})$$

# N-step algorithms

- General formula

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1 - \alpha) Q(s_t, a_t)$$

**N-step SARSA**

$$\hat{Q}(s_t, a_t) = [\sum_{\tau=t}^{\tau<t+n} \gamma^{\tau} r(s_{t+\tau}, a_{t+\tau})] + \gamma^{n} Q(s_{t+n}, a_{t+n})$$

**N-step Q-learning**

$$\hat{Q}(s_t, a_t) = [\sum_{\tau=t}^{\tau<t+n} \gamma^{\tau} r(s_{t+\tau}, a_{t+\tau})] + \gamma^{n} \cdot \max_{a} Q(s_{t+n}, a)$$



**Trivia:** which of these methods work off-policy?

# N-step algorithms

- General formula

$$Q(s_t, a_t) \leftarrow \alpha \cdot \hat{Q}(s_t, a_t) + (1-\alpha) Q(s_t, a_t)$$

**N-step SARSA**

$$\hat{Q}(s_t, a_t) = [\sum_{\tau=t}^{\tau<t+n} \gamma^{\tau} r(s_{t+\tau}, a_{t+\tau})] + \gamma^n Q(s_{t+n}, a_{t+n})$$

**N-step Q-learning**

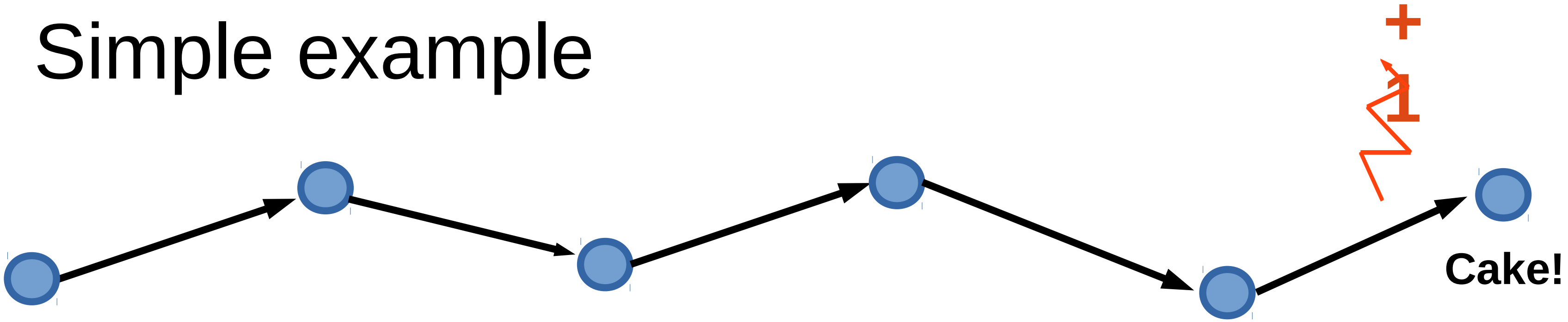
$$\hat{Q}(s_t, a_t) = [\sum_{\tau=t}^{\tau<t+n} \gamma^{\tau} r(s_{t+\tau}, a_{t+\tau})] + \gamma^n \cdot \max_a Q(s_{t+n}, a)$$



**Trivia:** which of these methods work off-policy? **None of them!**

# 1-step Vs n-step

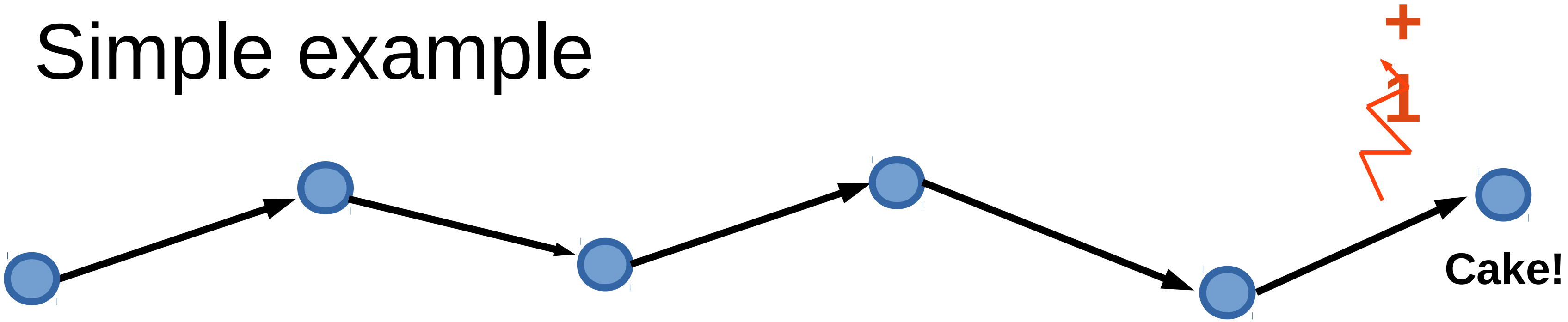
- Simple example




How many games does it take for **SARSA** to learn the optimal policy?

# 1-step Vs n-step

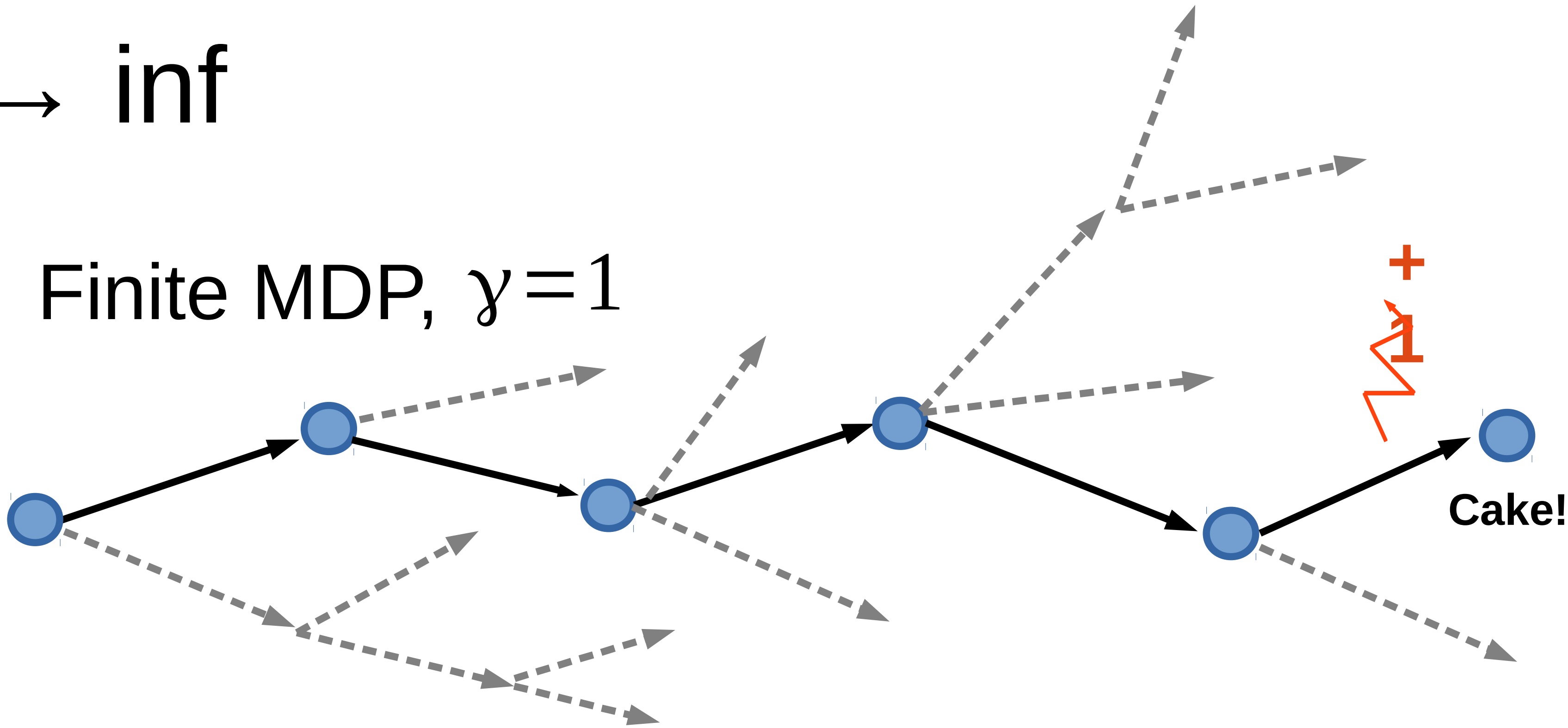
- Simple example




- SARSA needs 5 steps, n-step SARSA needs 1
- Nuts and bolts
  - Nonlinear approximations learn much faster!
  - Play for N steps, than learn (batched)

# n $\to$ inf

- Finite MDP, $\gamma = 1$




- Sample many trajectories (or tree search)
- Compute expected $Q(s,a) = \underset{\substack{s' \sim p(s'|s,a), \\ a' \sim \pi(a'|s') \\ s'' \sim p(s''|s',a') \\ \ldots}}{E} R(s,a)$

minimal assumptions, unbiased, large variance

# New stuff we learned

- Anything?

# New stuff we learned

- $Q(s,a), Q^*(s,a)$
- Q-learning, SARSA
  - We can learn from trajectories (model-free)
- Exploration vs exploitation (basics)
- Learning On-policy vs Off-policy
  - Using experience replay

# Coming next...

- What if state space is large/continuous
  - Deep reinforcement learning