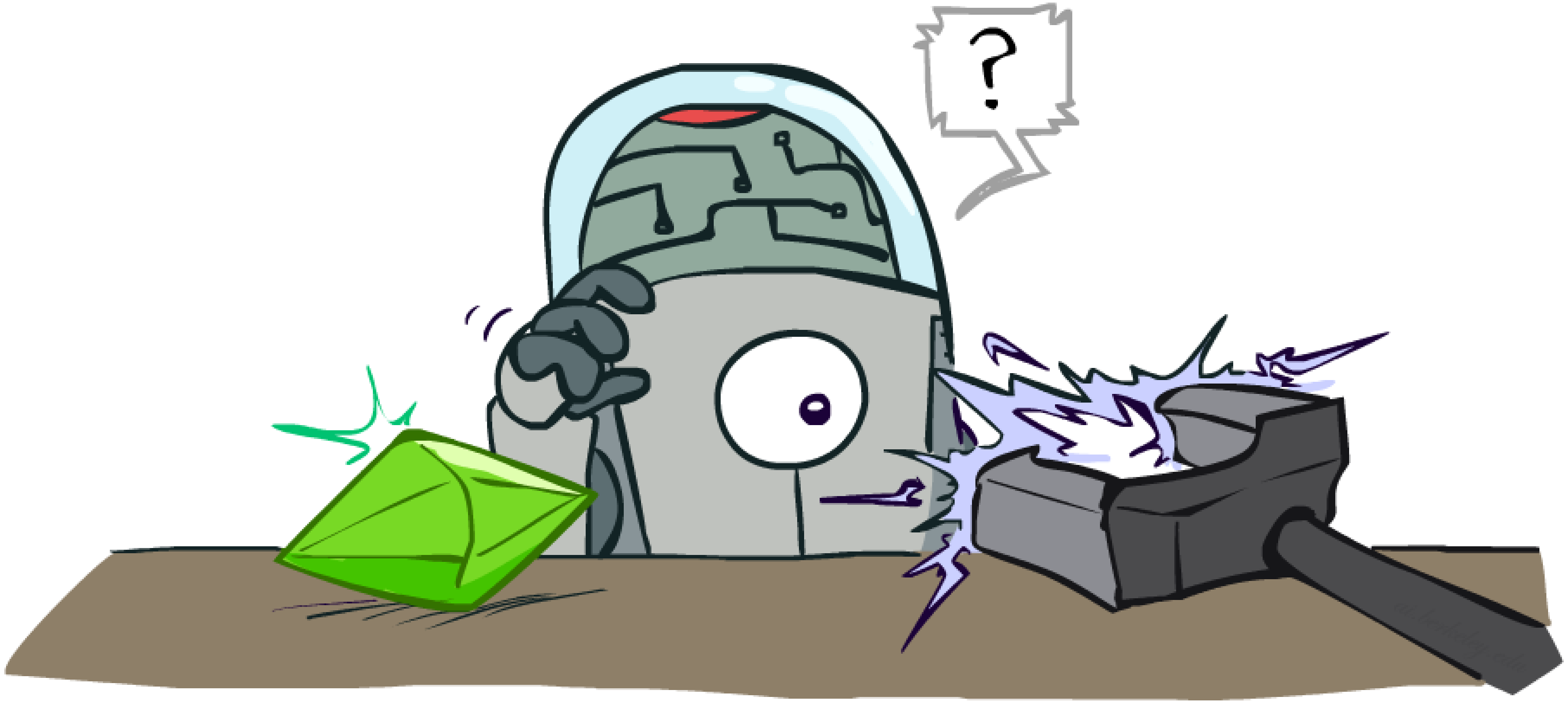
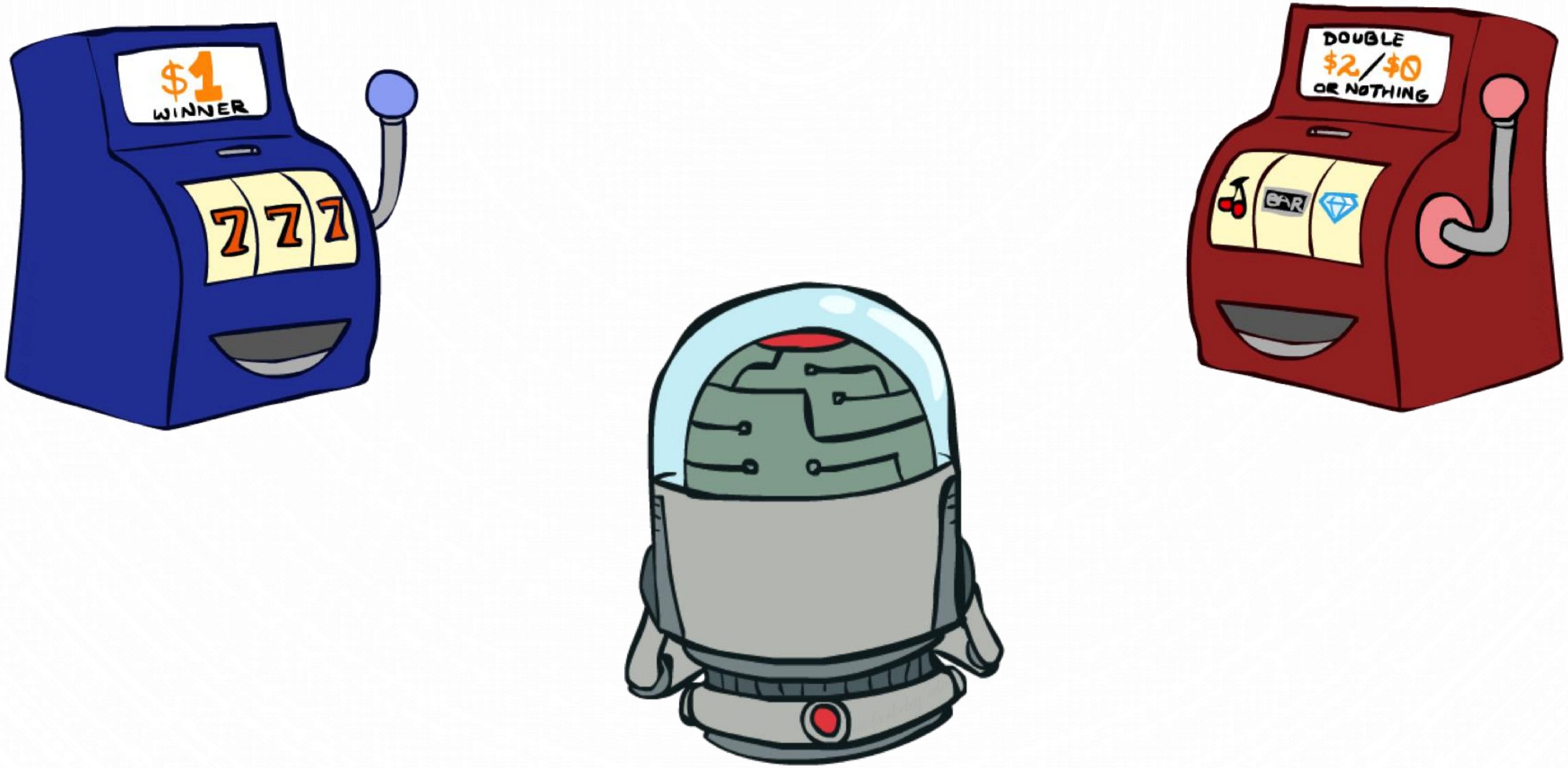
Reinforcement Learning

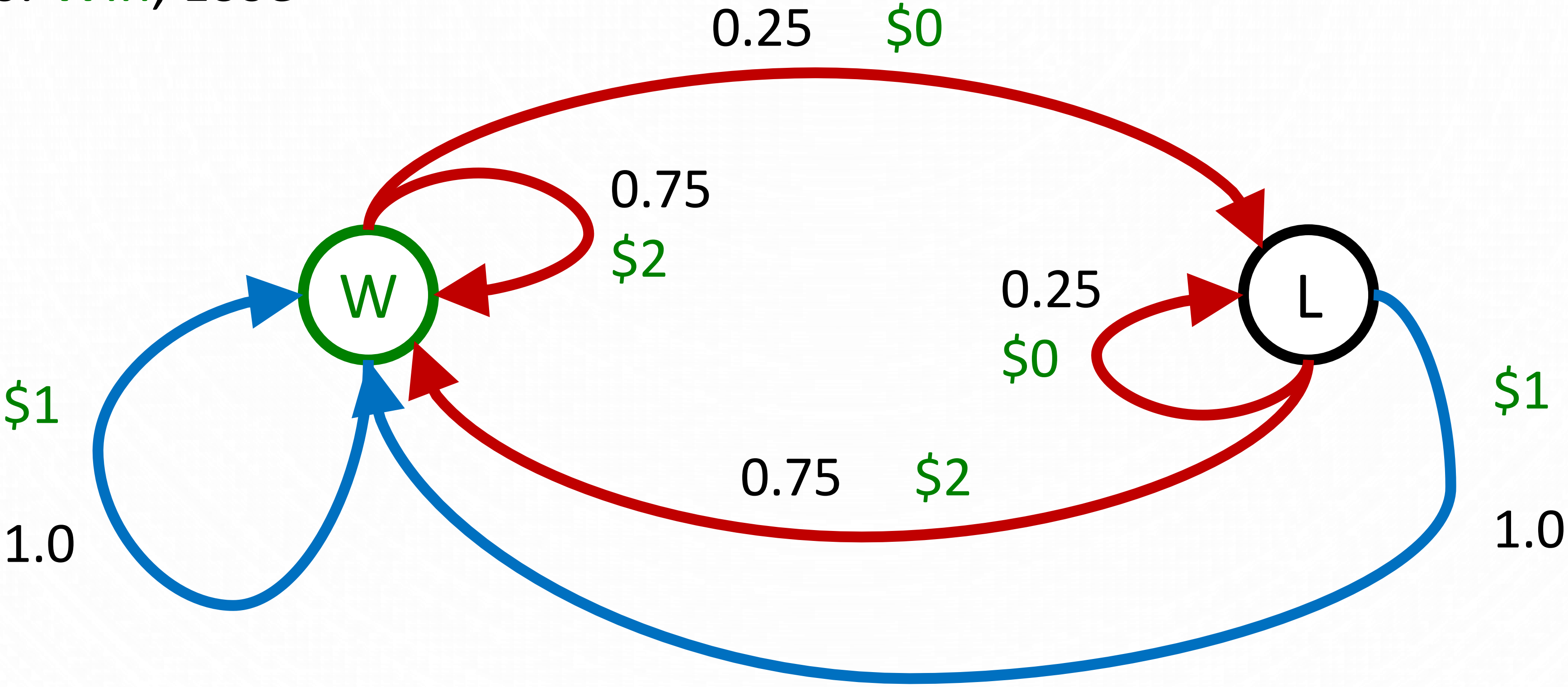


DOUBLE BANDITS



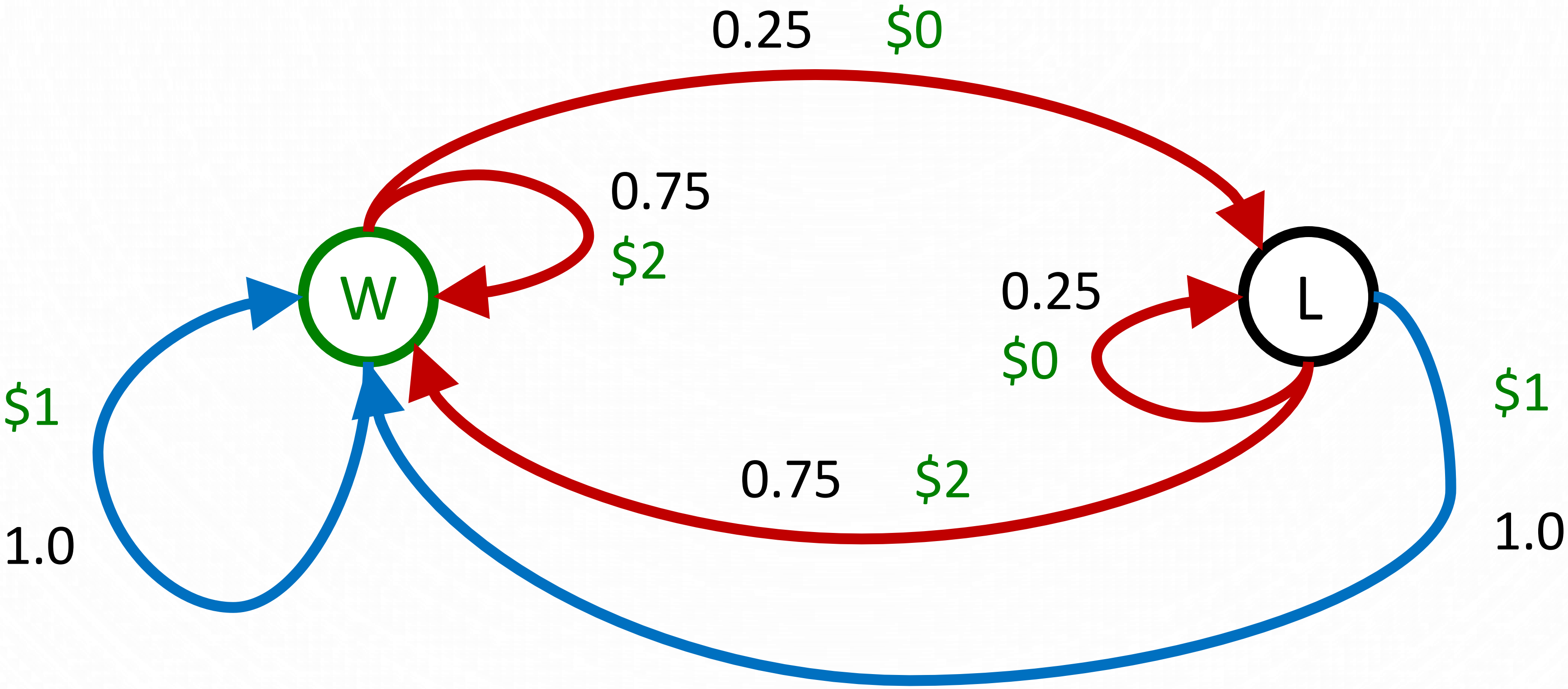
DOUBLE-BANDIT MDP

- Actions: *Blue*, *Red*
- States: Win, Lose



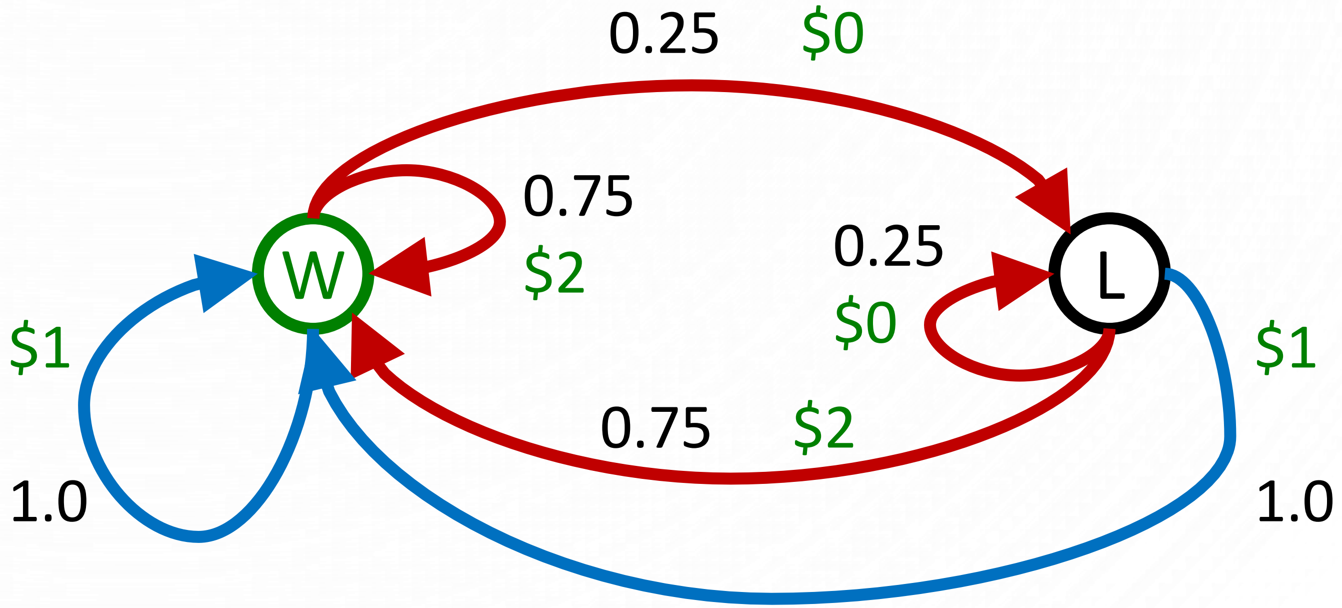
DOUBLE-BANDIT MDP

- Formulate the problem as an MDP, and solve it by Value Iteration, you will find the best policy is to always play the red machine

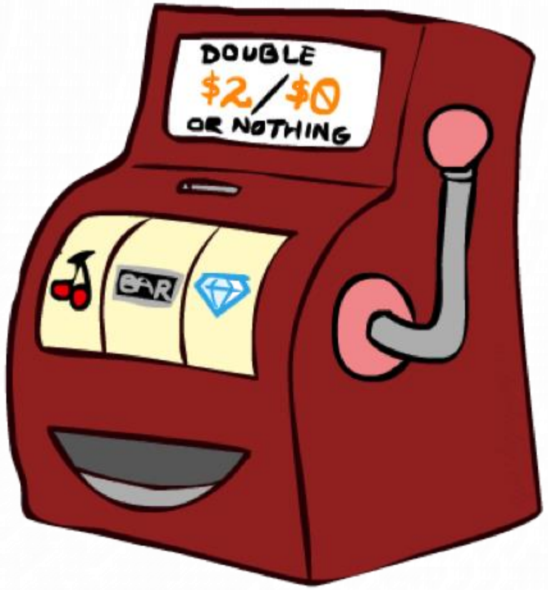


OFFLINE PLANNING

- Solving MDPs is offline planning
  - You determine all quantities through computation
  - You need to know the details of the MDP
  - You do not actually play the game!



LET'S PLAY!

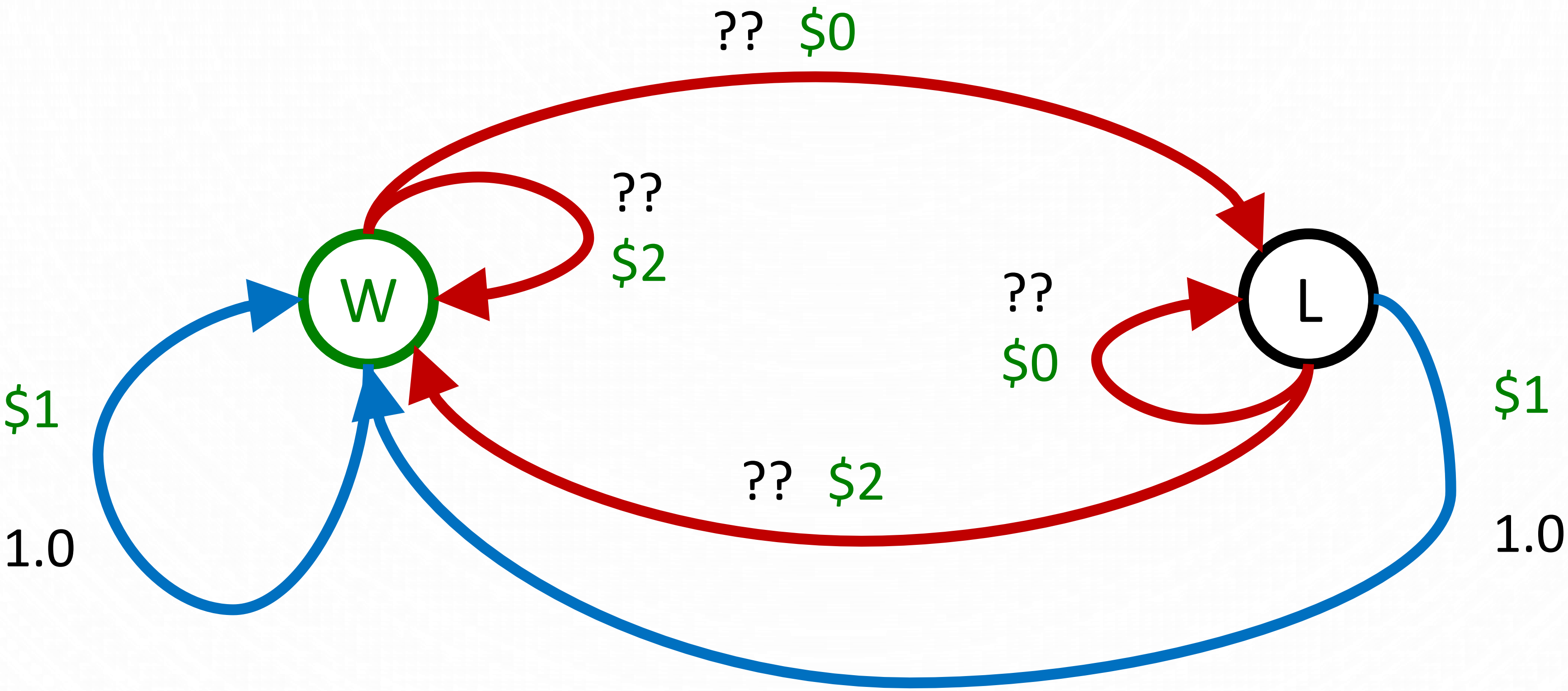


\$2 \$2 \$0 \$2 \$2

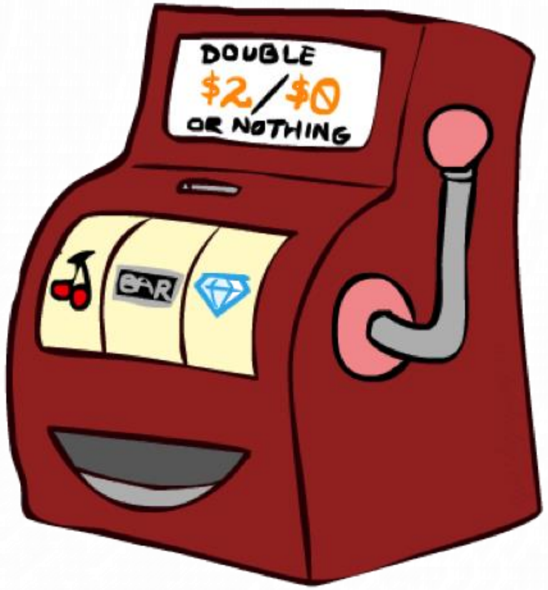
\$2 \$2 \$0 \$0 \$0

ONLINE LEARNING

- Rules changed! Red's win chance is unknown.



LET’S PLAY!



$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

WHAT JUST HAPPENED?

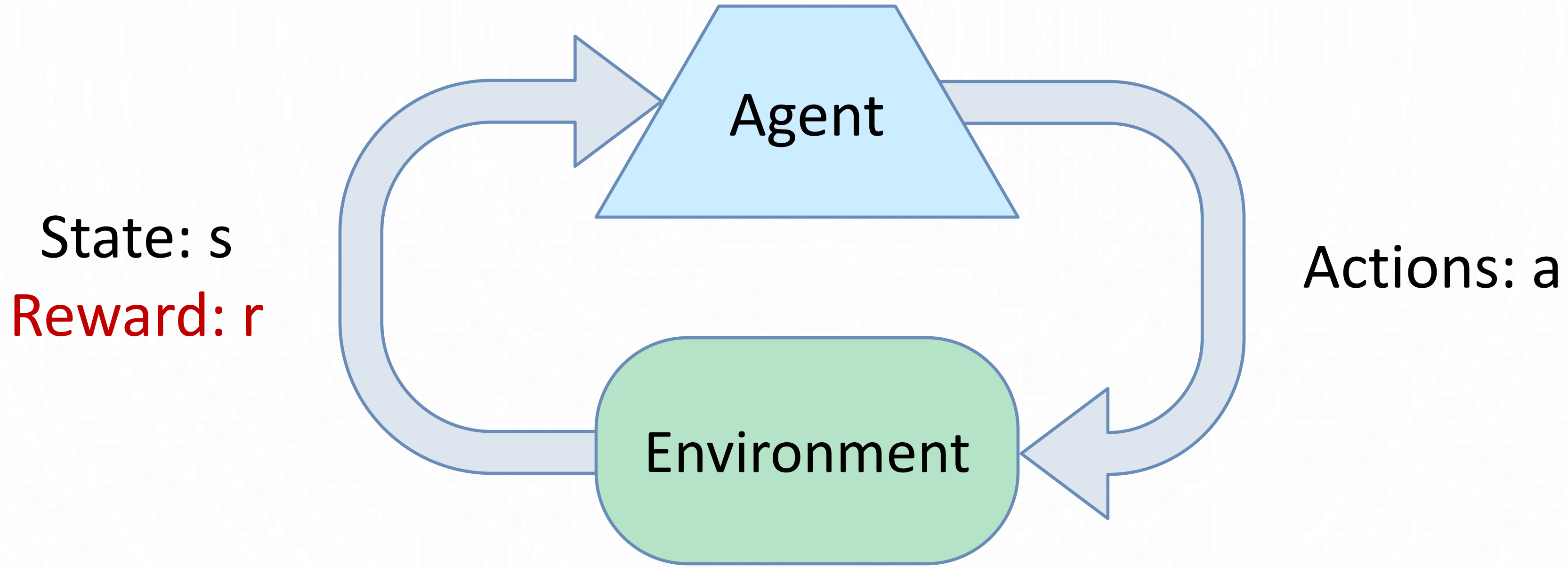
- That wasn't planning, it was learning!
  - Specifically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computation
  - You needed to actually act to figure it out

- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Sampling: because of chance, you have to try things repeatedly

EXPLORATION VS EXPLOITATION

- Restaurant Selection
  - Exploitation: Go to your favorite restaurant
  - Exploration: Try a new restaurant
- Oil Drilling
  - Exploitation: Drill at the best-known location
  - Exploration: Drill at a new location
- Game Playing
  - Exploitation: Play the move you believe is best
  - Exploration: Play an experimental move

REINFORCEMENT LEARNING



- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

EXAMPLE: LEARNING TO WALK

Initial

A Learning Trial

After Learning [1K Trials]

[Kohl and Stone, ICRA 2004]

EXAMPLE: LEARNING TO WALK

Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initial]

EXAMPLE: LEARNING TO WALK

Training

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – training]

EXAMPLE: LEARNING TO WALK

Finished

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finished]

EXAMPLE: SIDEWINDING

[Andrew Ng]

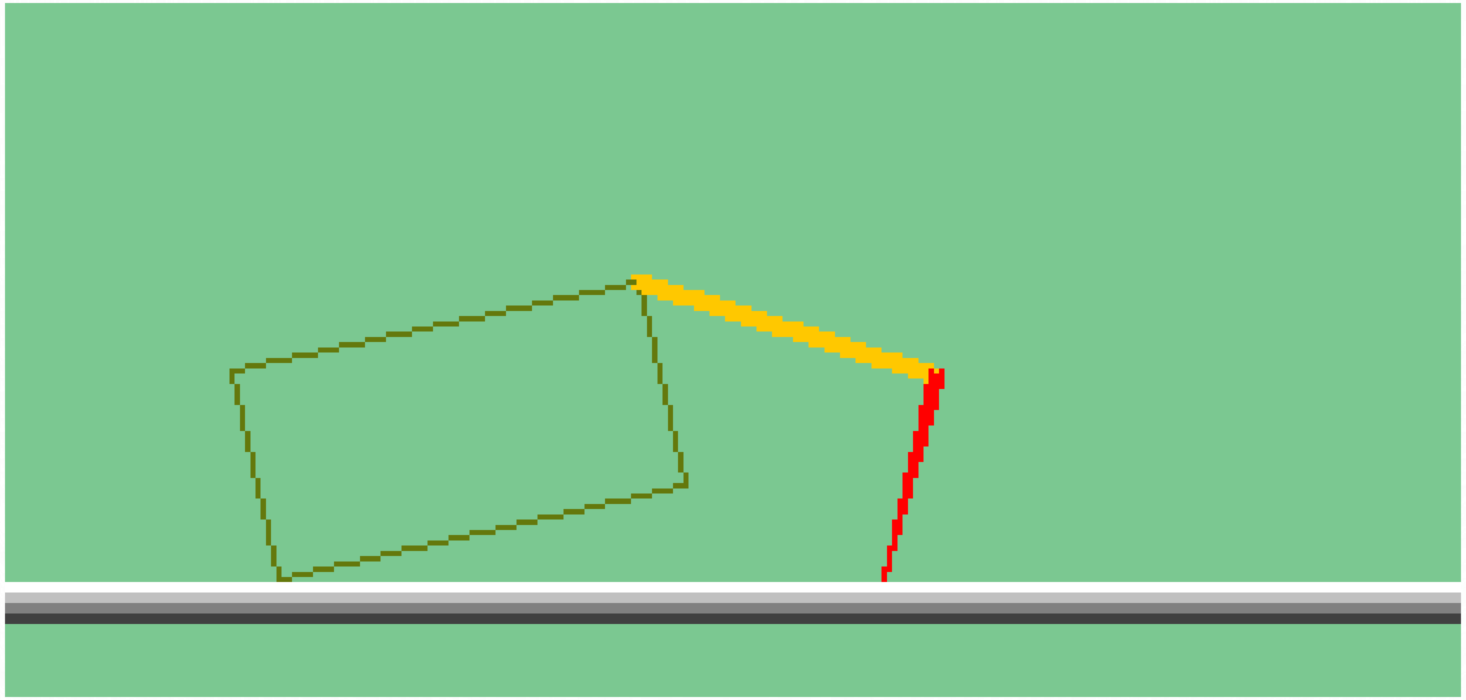
[Video: SNAKE – climbStep+sidewinding]

EXAMPLE: TODDLER ROBOT

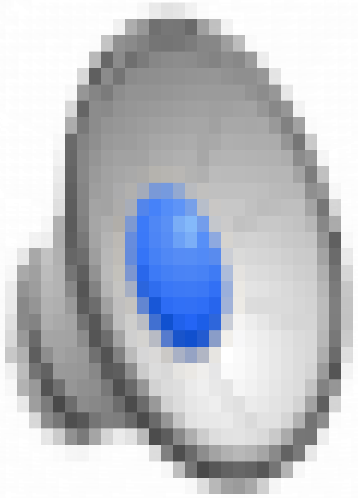
[Tedrake, Zhang and Seung, 2005]

[Video: TODDLER – 40s]

THE CRAWLER!

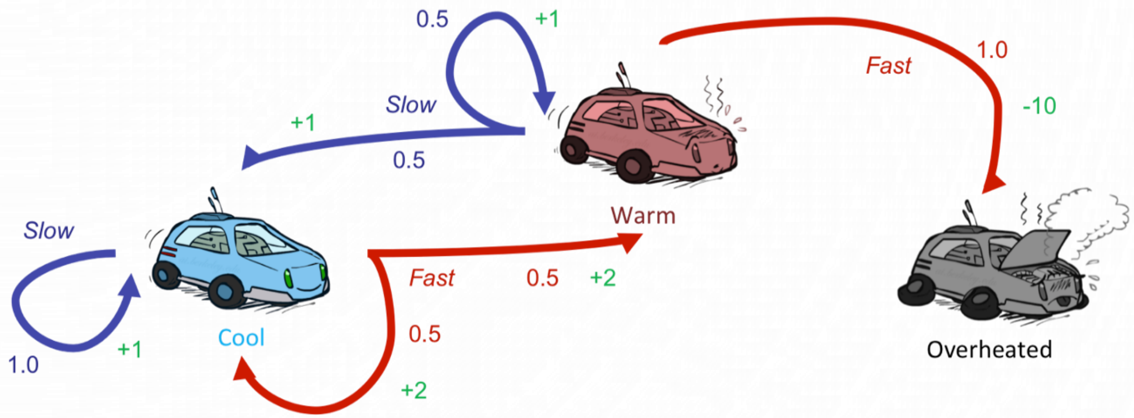


VIDEO OF DEMO CRAWLER BOT

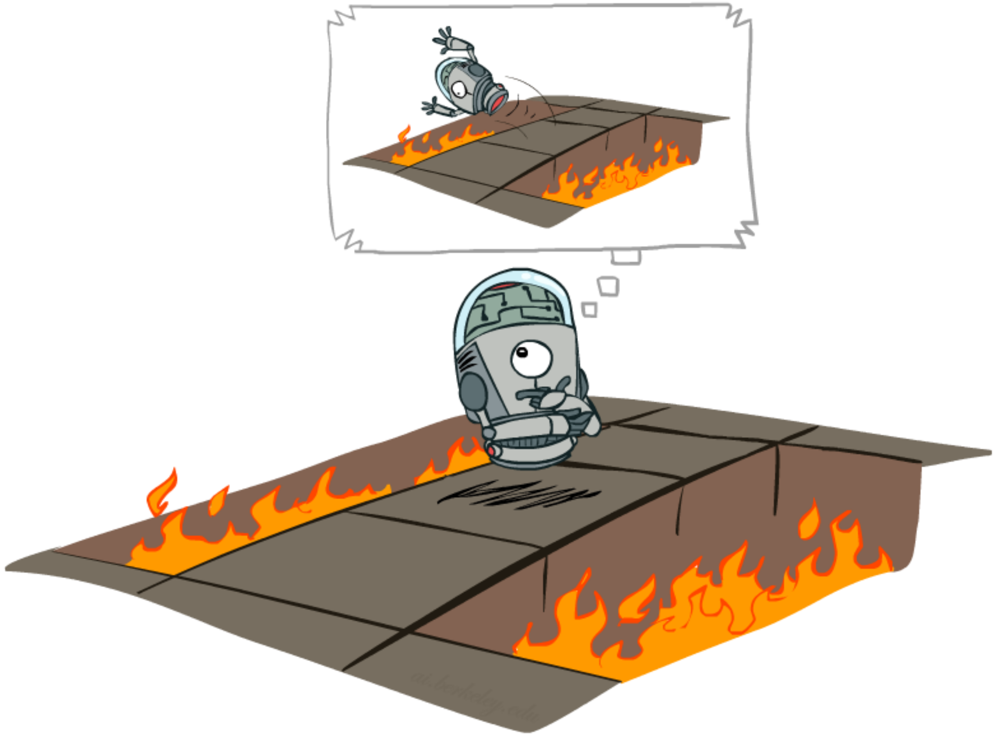


REINFORCEMENT LEARNING

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy $\pi(s)$
- New twist: don't know T or R
  - i.e. we don't know which states are good or what the actions do
  - Must actually try out actions and states to learn



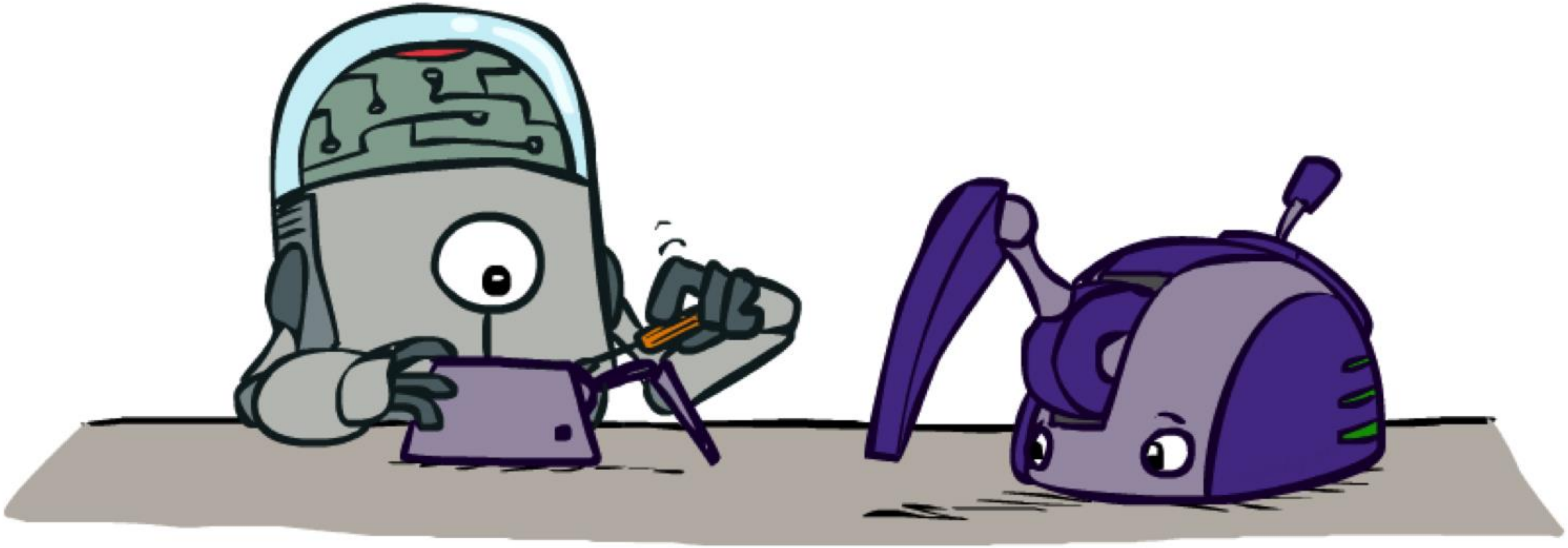
OFFLINE (MDPS) VS. ONLINE (RL)



Offline Solution

Online Learning

MODEL-BASED LEARNING



MODEL-BASED LEARNING

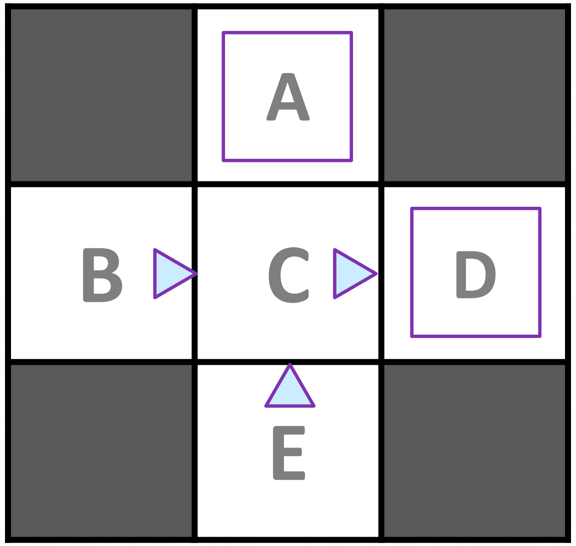
- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- Step 1: Learn empirical MDP model
  - Count outcomes s’ for each (s, a)
  - Normalize to give an estimate of $\widehat{T}(s,a,s')$
  - Discover each $\widehat{R}(s,a,s')$ when we experience (s, a, s’)

- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

EXAMPLE: MODEL-BASED LEARNING

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$$\hat{T}(s, a, s')$$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
…

$$\hat{R}(s, a, s')$$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
…

EXAMPLE: EXPECTED AGE

Goal: Compute expected age of students in a class

Known P(A)

$$E[A] = \sum_a P(a) \cdot a \qquad = 0.35 \times 20 + \ldots$$

Without P(A), instead collect samples [a_1, a_2, … a_N]

Unknown P(A): “Model Based”

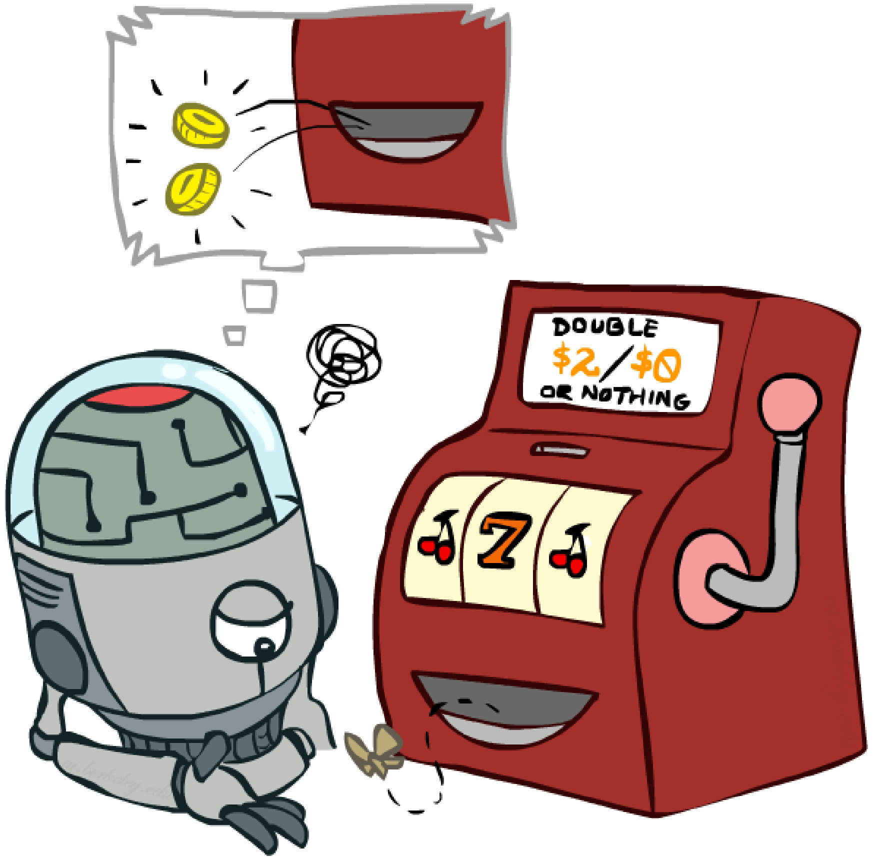
$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

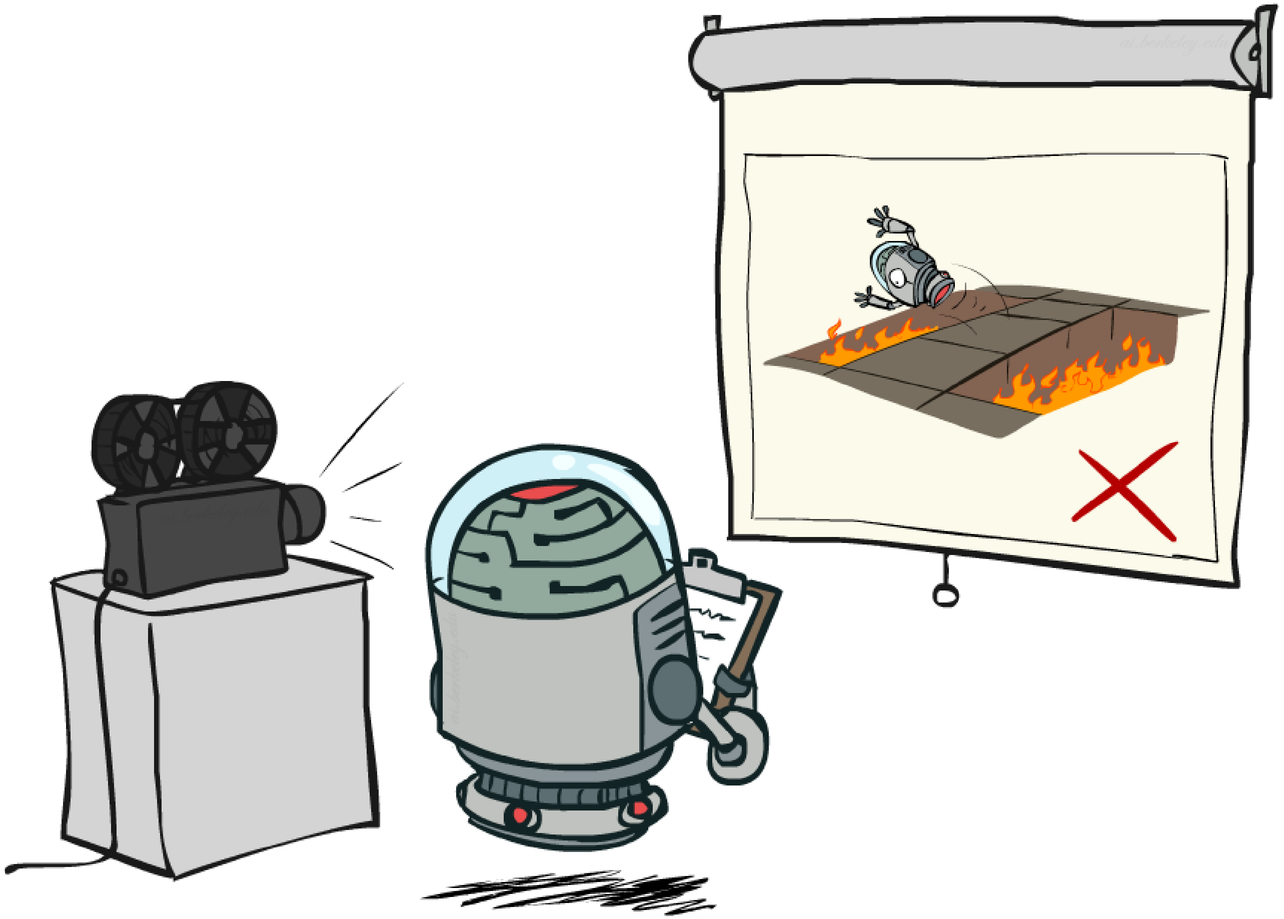
Unknown P(A): “Model Free”

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

MODEL-FREE LEARNING

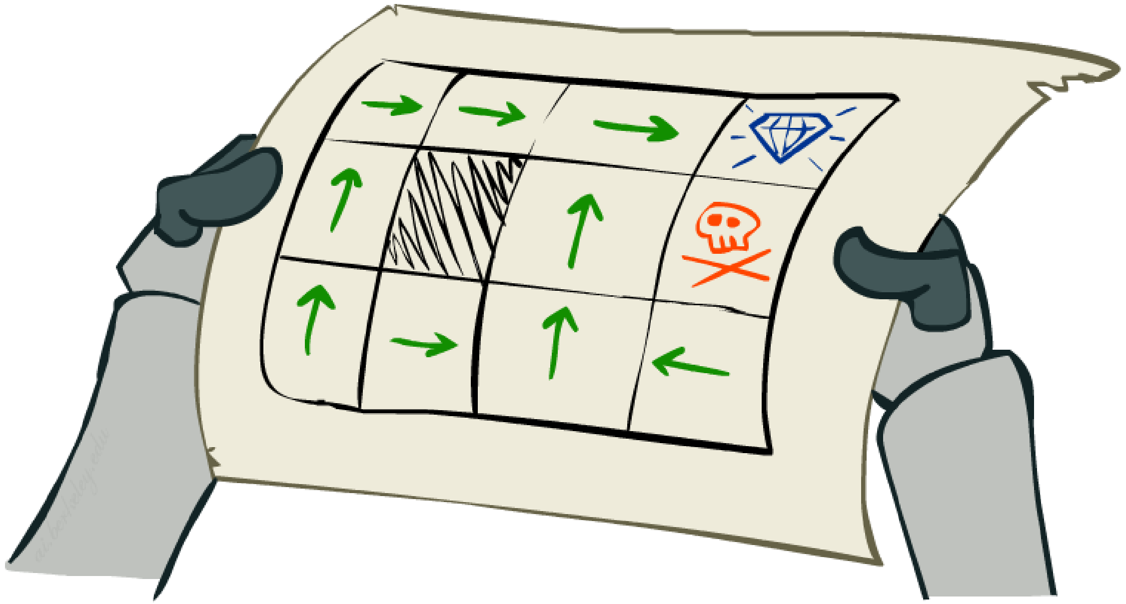


PASSIVE REINFORCEMENT LEARNING



PASSIVE REINFORCEMENT LEARNING

- Given a fixed policy $\pi(s)$
  - You don't know the transitions $T(s, a, s')$
  - You don't know the rewards $R(s, a, s')$
- Goal: learn the state values $V^{\pi}(s)$



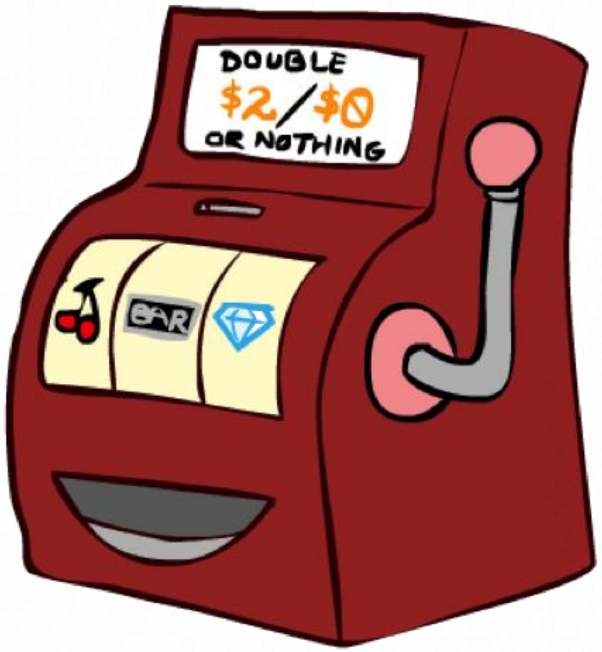
- In this case:
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.

$$V_0^{\pi}(s) = 0$$

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

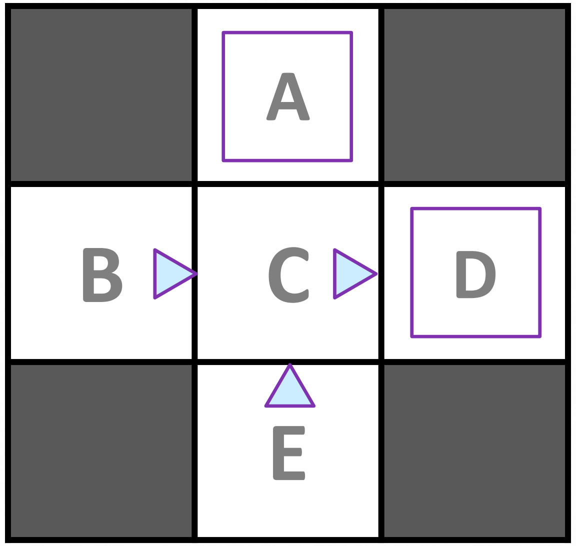
DIRECT EVALUATION

- Goal: Compute the value for each state under a given π, i.e. $V^{\pi}(s)$
- Idea: Average together observed sample values
  - Let the agent act according to π
  - For each state, calculate the sum of the discounted rewards from that state to the end of the game
  - Average those samples from different episodes
- This is called direct evaluation



EXAMPLE: DIRECT EVALUATION

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

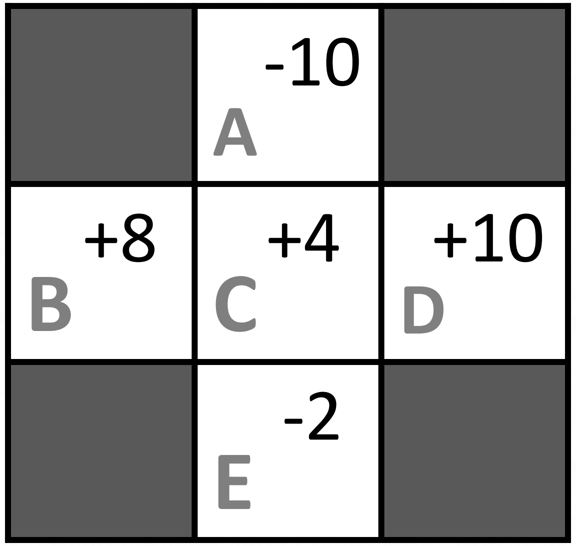
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



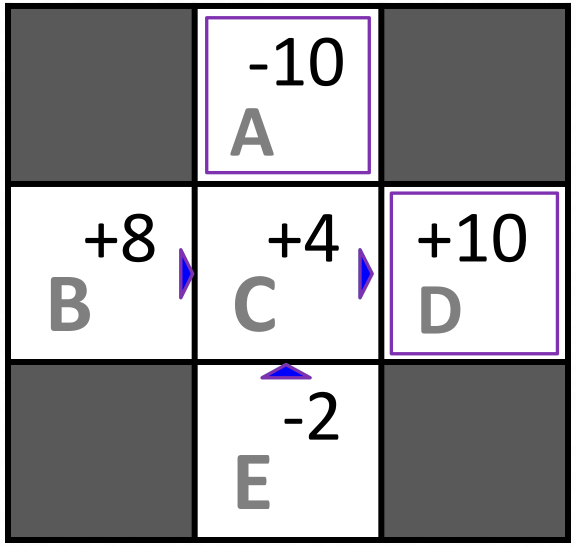
EXAMPLE: DIRECT EVALUATION

	Episode 1	Episode 2	Episode 3	Episode 4	average
A	none	none	none	-10	-10
B	-1-1+10=8	-1-1+10=8	none	none	+8
C	-1+10=9	-1+10=9	-1+10=9	-1-10=-11	+4
D	10	10	10	none	+10
E	none	none	-1-1+10=8	-1-1-10=-12	-2

PROBLEMS WITH DIRECT EVALUATION

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of T, R
  - It eventually computes the correct average values, using just sample transitions
- What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

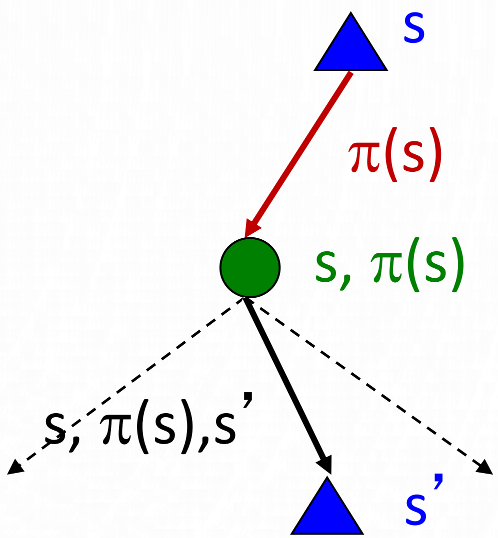
Output Values



If B and E both go to C under this policy, how can their values be different?

WHY NOT USE POLICY EVALUATION?

- Simplified Bellman updates calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer over V



$$V_0^{\pi}(s) = 0$$

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')]$$

  - This approach fully exploited the connections between the states
  - Unfortunately, we need T and R to do it!

- **Key question:** how can we do this update to V without knowing T and R?
  - In other words, how do we take a weighted average without knowing the weights?

SAMPLE-BASED POLICY EVALUATION?

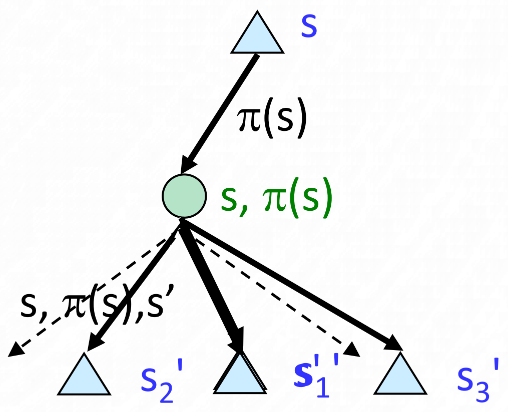
- We want to improve our estimate of V by computing these averages:

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')]$$

- Idea: Take samples of outcomes s' (by doing the action!) and average

$$sample_1 = R(s,\pi(s),s_1') + \gamma V_k^{\pi}(s_1')$$
$$sample_2 = R(s,\pi(s),s_2') + \gamma V_k^{\pi}(s_2')$$
$$\dots$$
$$sample_n = R(s,\pi(s),s_n') + \gamma V_k^{\pi}(s_n')$$

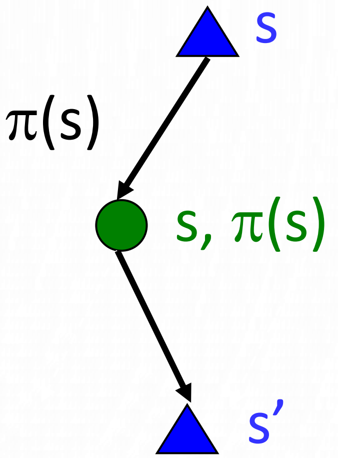
$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n}\sum_i sample_i$$



Almost! But we can't rewind time to get sample after sample from state s.

TEMPORAL DIFFERENCE LEARNING

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s’, r)
  - Likely outcomes s’ will contribute updates more often
- Temporal difference learning of values
  - Policy still fixed, still doing policy evaluation!



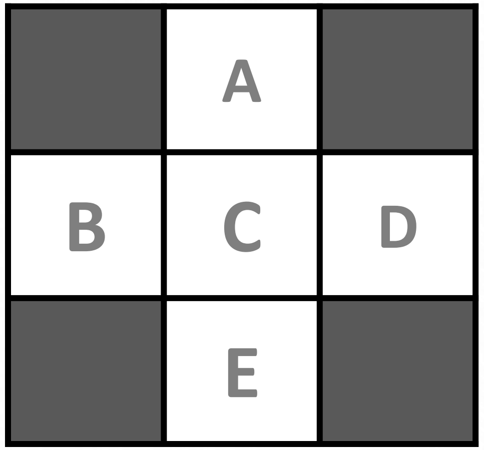
Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

EXAMPLE: TEMPORAL DIFFERENCE LEARNING

States

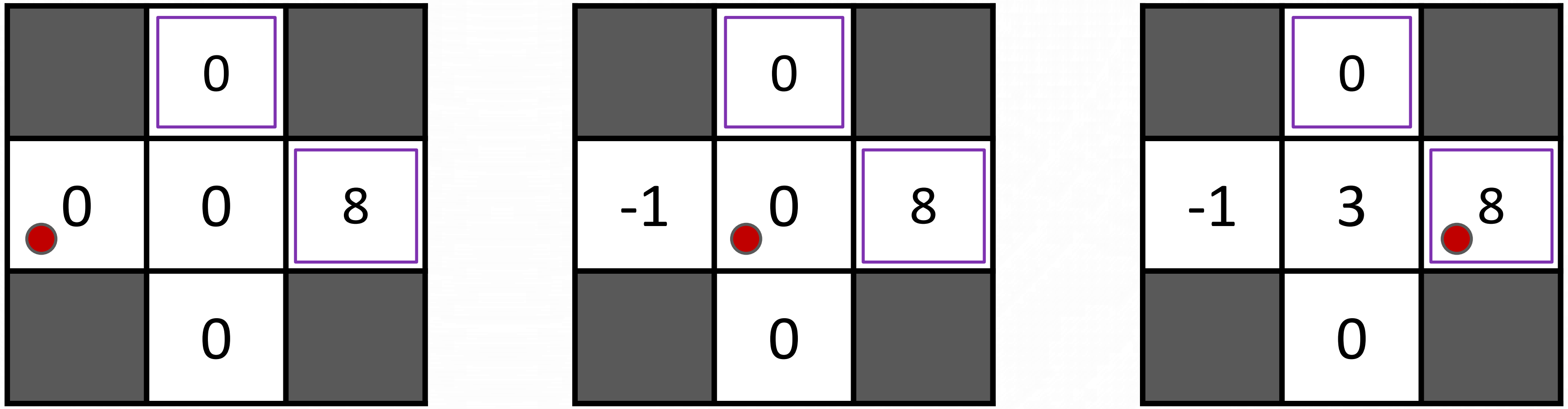


Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions

B, east, C, -2

C, east, D, -2



$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

PROBLEMS WITH TD VALUE LEARNING

- TD value leaning is a model-free way to do policy evaluation (calculate the values V), mimicking Bellman updates with running sample averages
- However, if we want to turn values into a (new) policy, we're sunk:

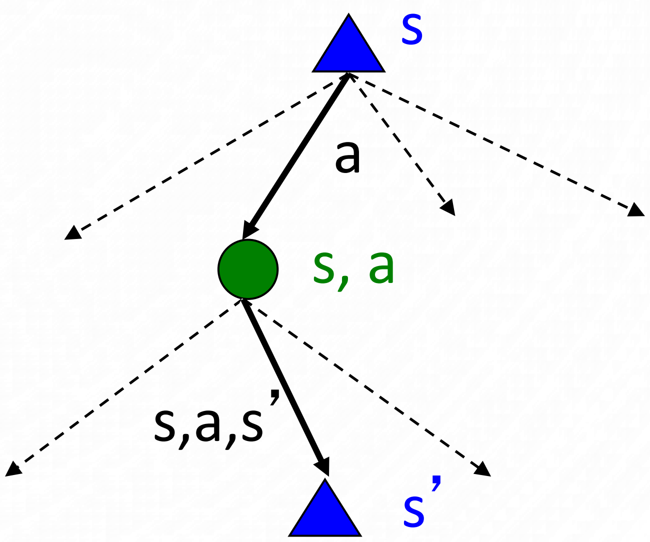
$$\pi(s) = \arg\max_a Q(s,a)$$

$$Q(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V(s')\right]$$

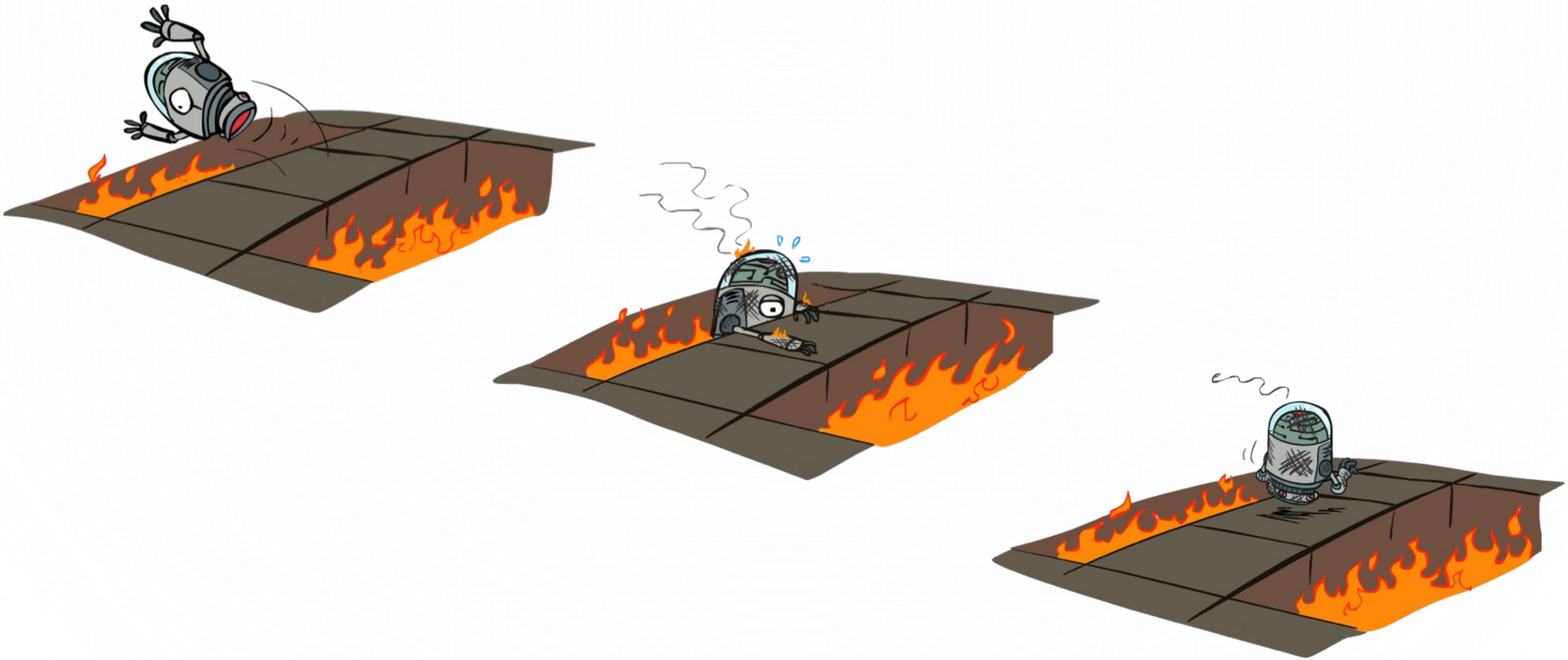
- Recall: policy extraction:

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s') \times [R(s,a,s') + \gamma V^*(s')]$$

- Idea: learn Q-values, not values
- Makes action selection model-free too!

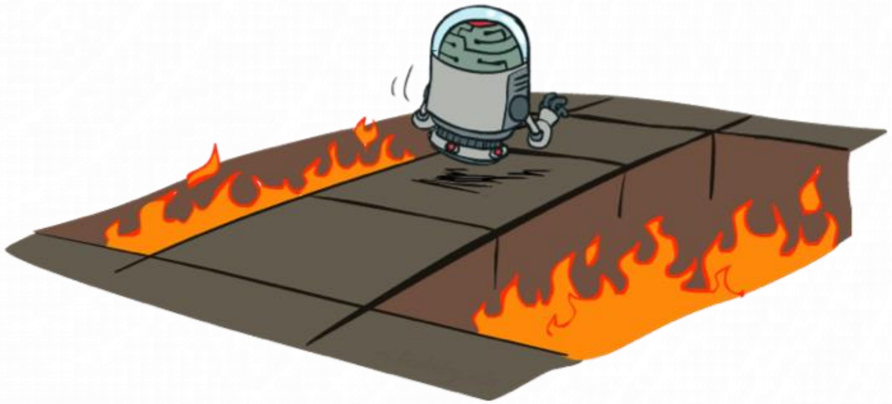


ACTIVE REINFORCEMENT LEARNING



ACTIVE REINFORCEMENT LEARNING

- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / values



- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

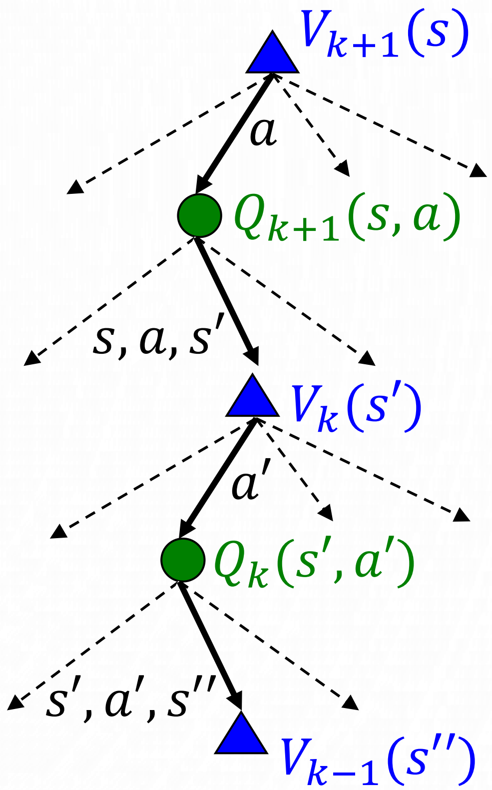
DETOUR: Q-VALUE ITERATION

- Value iteration:
  - Start with $V_0(s) = 0$
  - Given V_k, calculate the depth k+1 values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$
  - Given Q_k, calculate the $(k+1)^{th}$ iteration q-values for all q-states:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$



Q-LEARNING

- Q-Learning: sample-based Q-value iteration

$$Q(s,a) \leftarrow Q(s,a) + \alpha(\, sample - Q(s,a)\,)$$

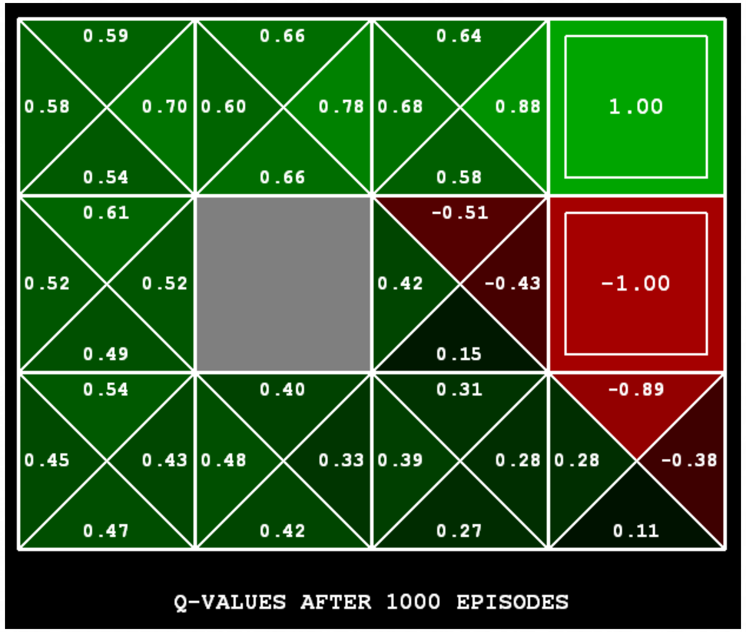
- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha(\, sample - Q(s,a)\,)$$



[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

VIDEO OF DEMO Q-LEARNING -- GRIDWORLD

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

Let $\gamma = 1, \alpha = 0.5$

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$