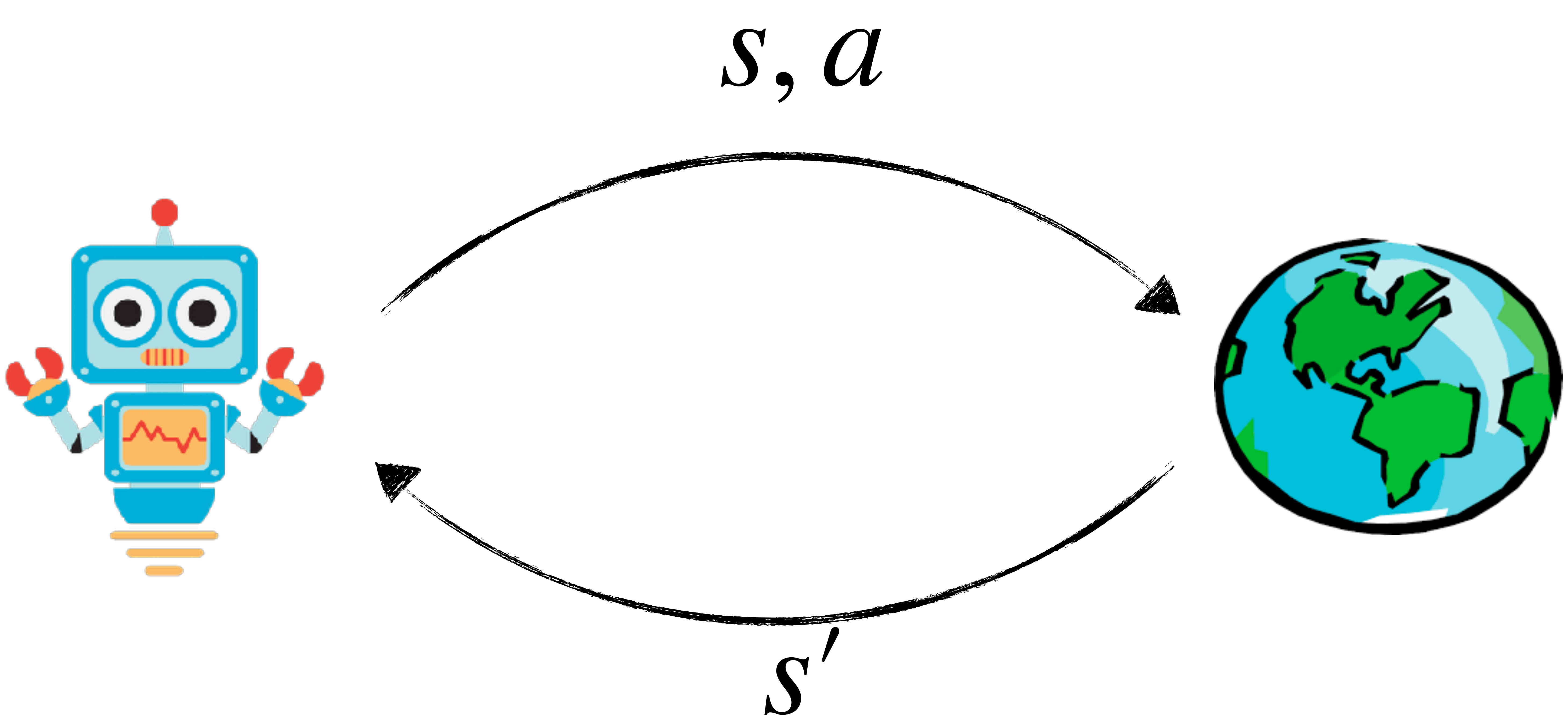
Temporal Difference & Q Learning

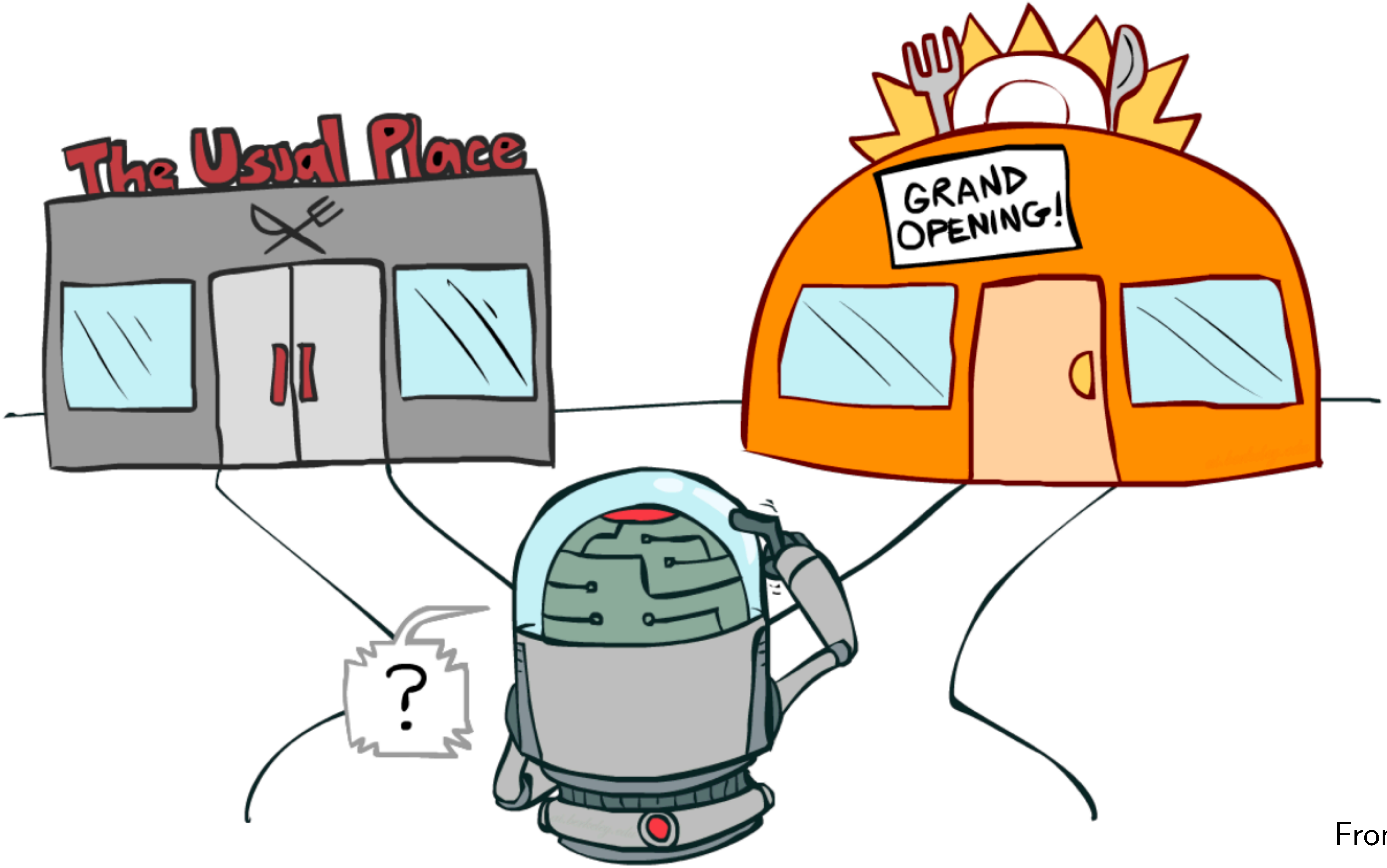
Sanjiban Choudhury

What if the transitions are unknown?

$$< S , A , C , \mathcal{T} >$$

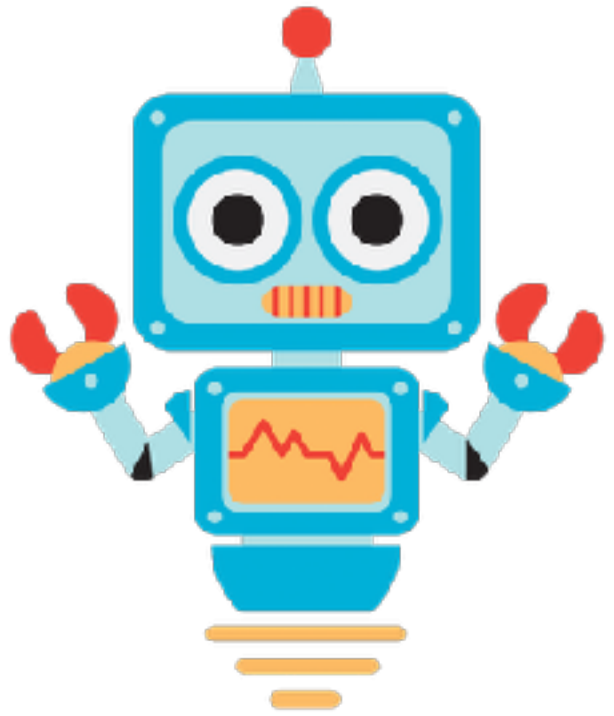


Exploration vs Exploitation



From Dan Klein

Doors



a^1 ?

a^2 ?

a^3 ?

⋮

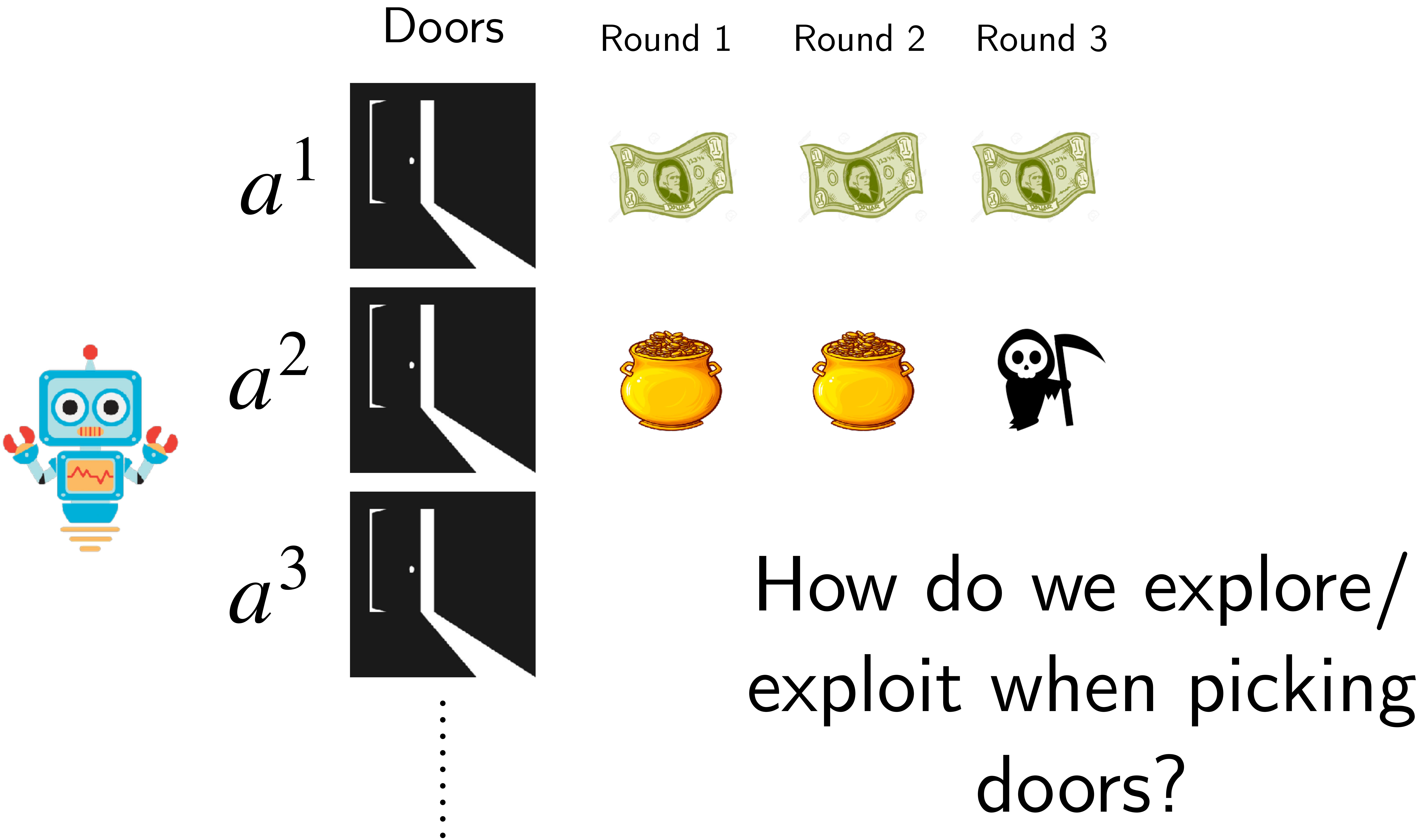
-100

-1

1000


Doors
Round 1
Round 2
Round 3
a^1
a^2
a^3
-100
-1
1000

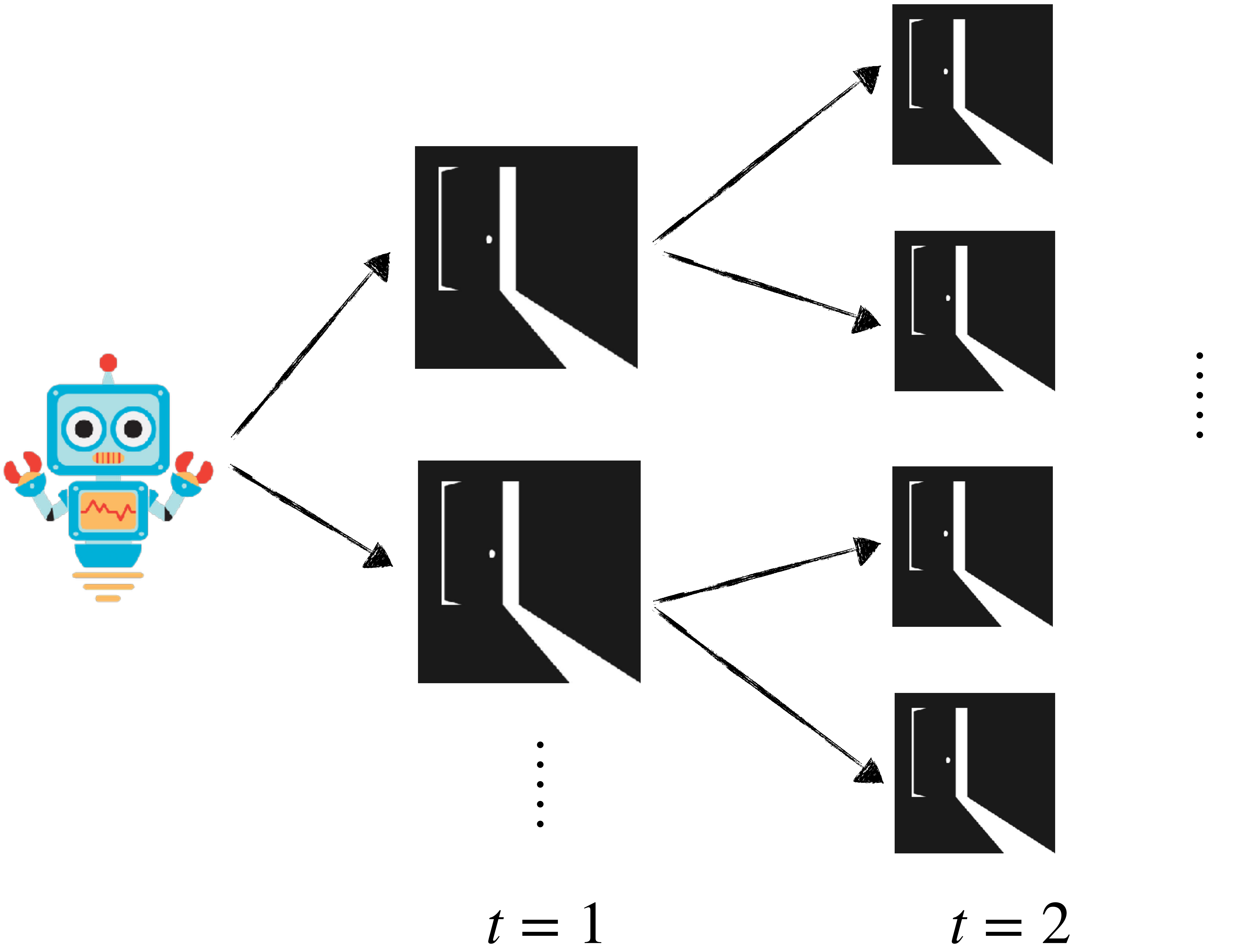

Doors
Round 1
Round 2
Round 3
a^1
a^2
a^3
-100
-1
1000



Doors
Round 1
Round 2
Round 3
a^1
a^2
a^3
How do we explore/
exploit when picking
doors?

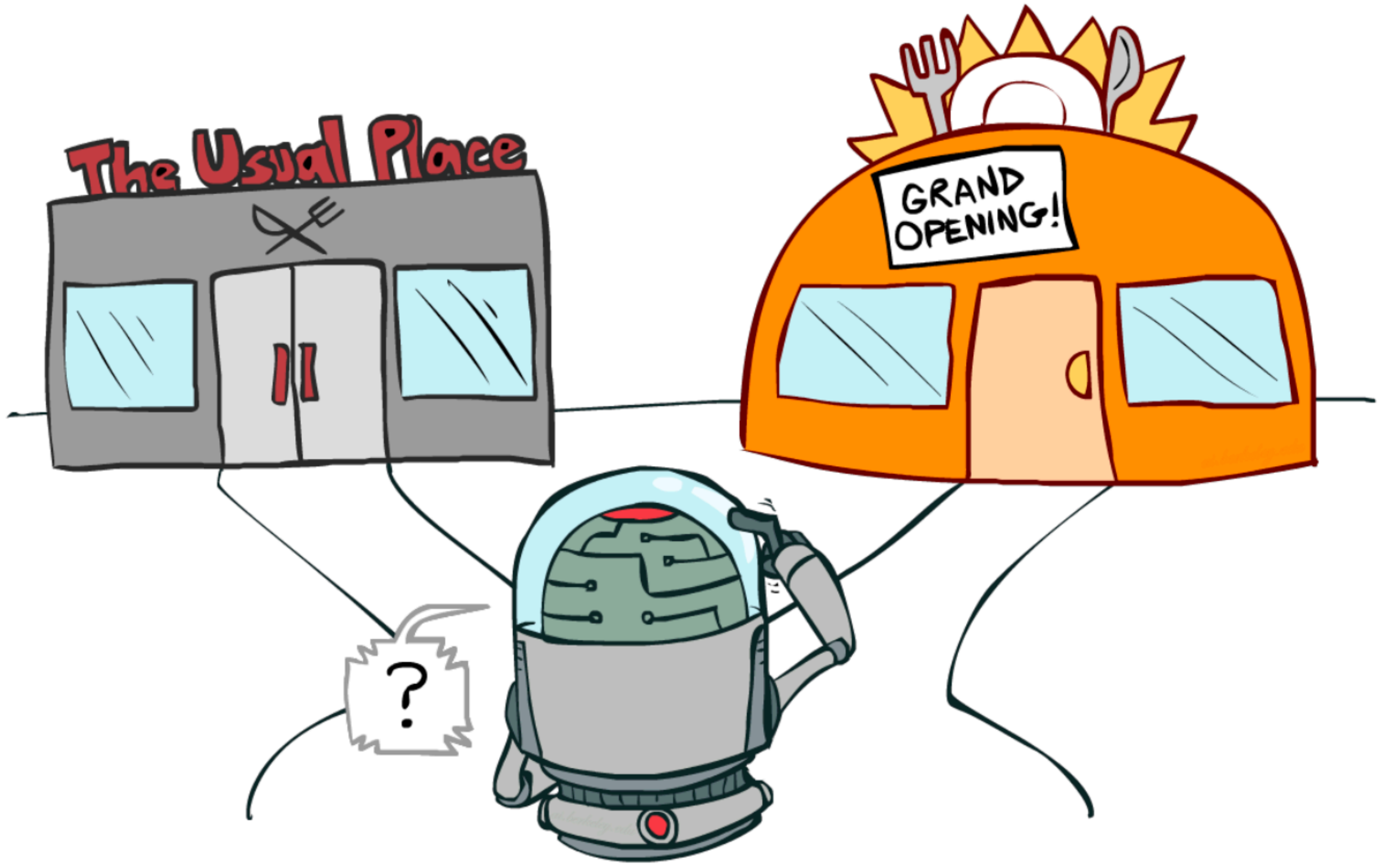
What if we played the game over multiple time steps?


-100
-1
1000
$t = 1$
$t = 2$

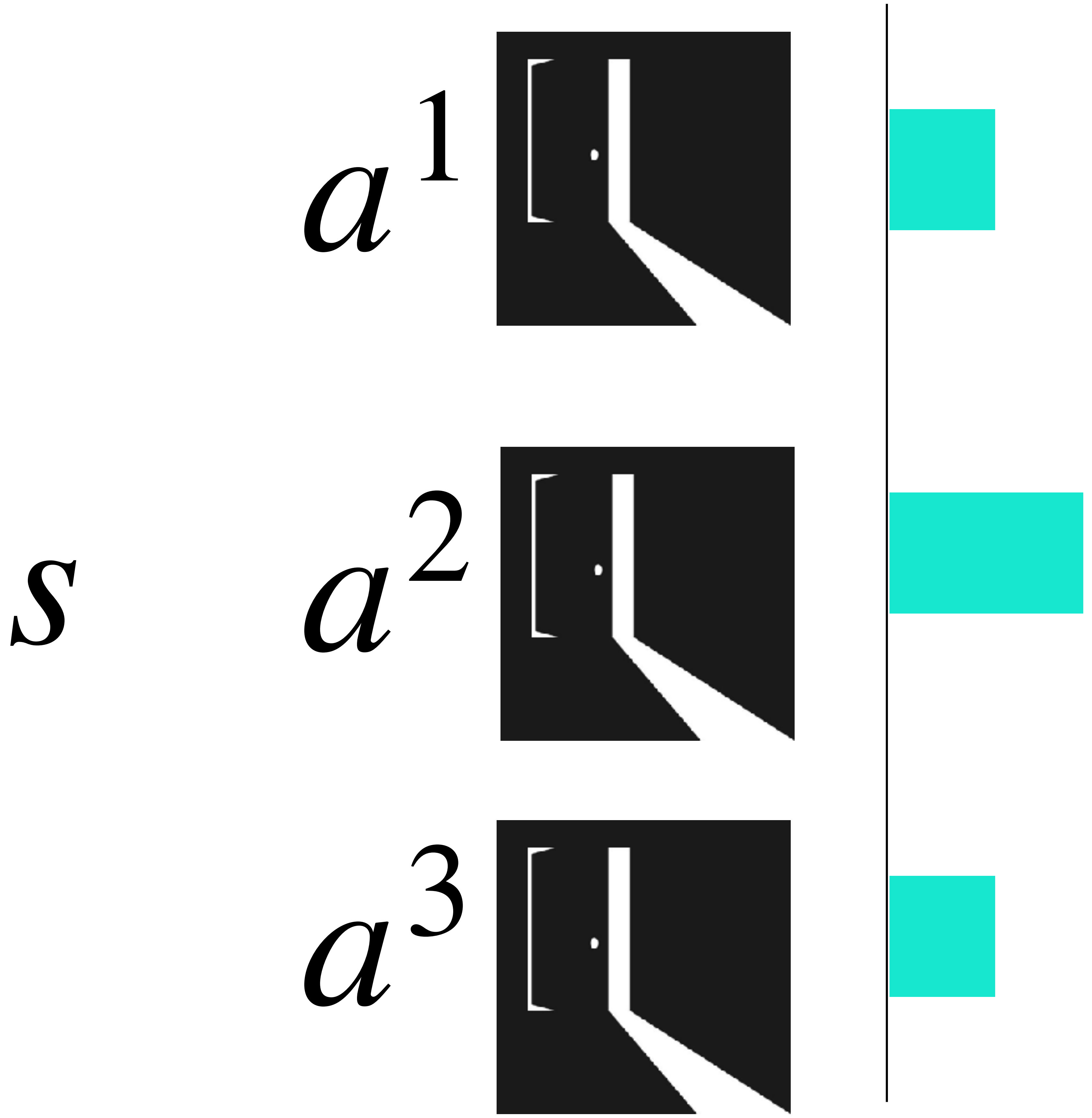


How do we estimate values of each door?

Two Ingredients of RL

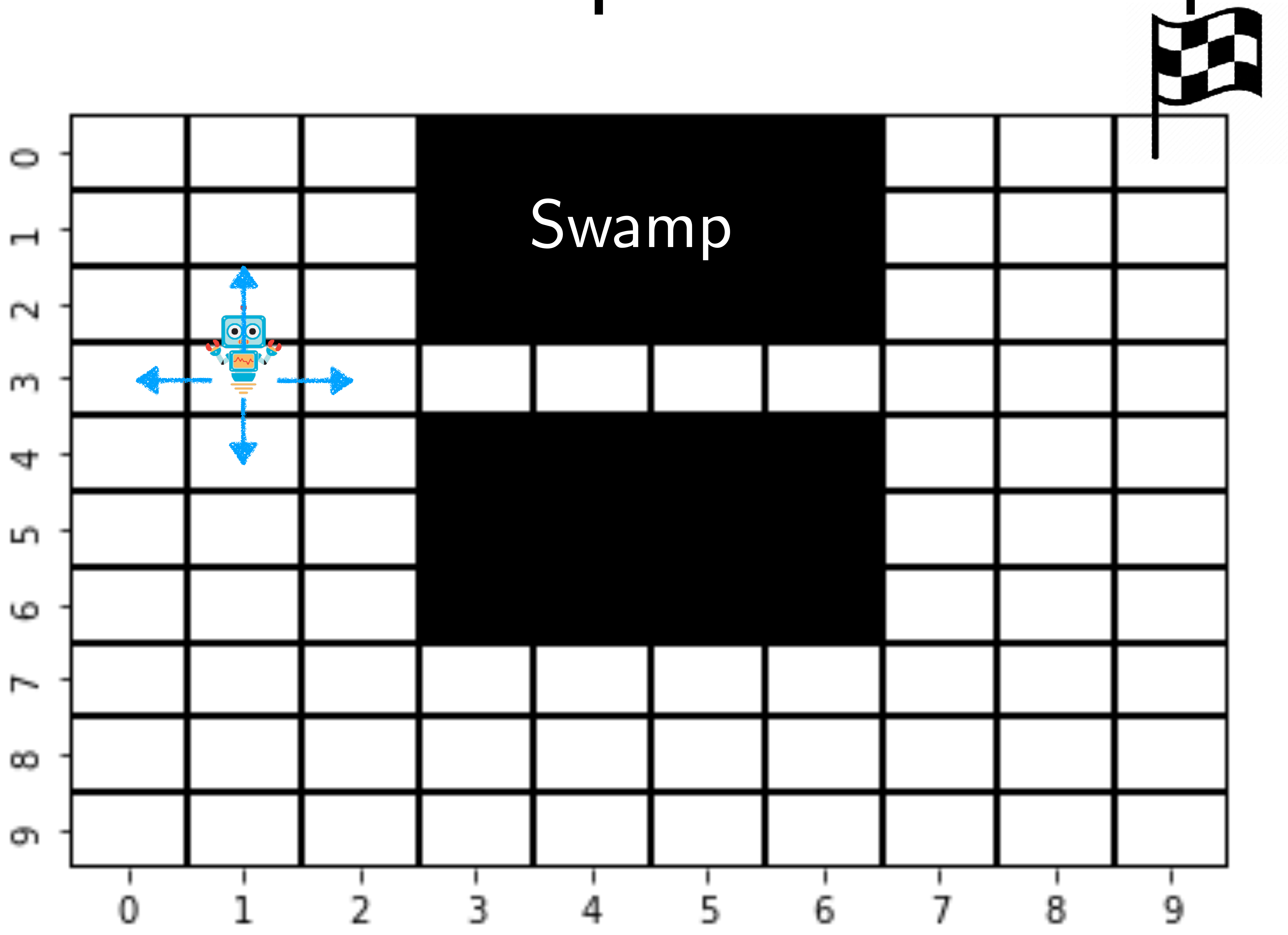


Exploration Exploitation



Estimate Values $Q(s, a)$

Recap: The Swamp MDP



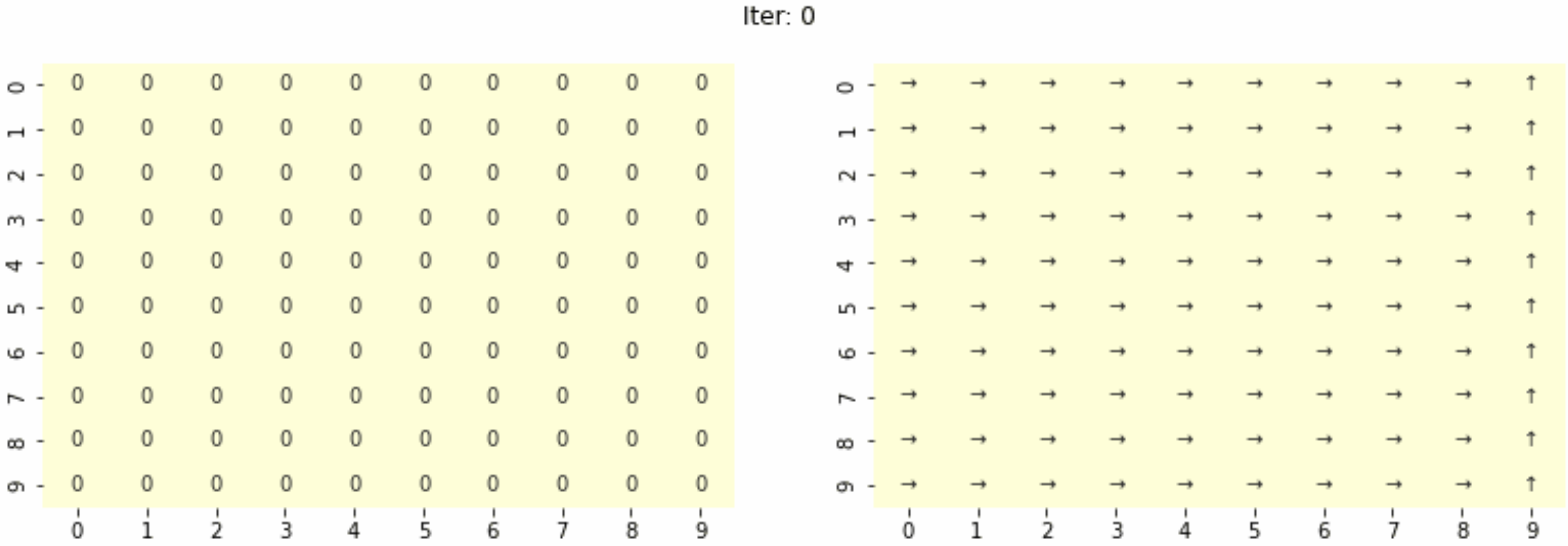
$< S, A, C, \mathcal{T} >$

- Two absorbing states: Goal and Swamp
- Cost of each state is 1 till you reach the goal
- Let's set T = 30

When the MDP is known!

Run Value / Policy Iteration

When MDP is known: Policy Iteration



$$V^{\pi}(s) = c(s, \pi(s)) + \gamma \mathbb{E}_{s' \sim \mathcal{T}(s,a)} V^{\pi}(s')]$$

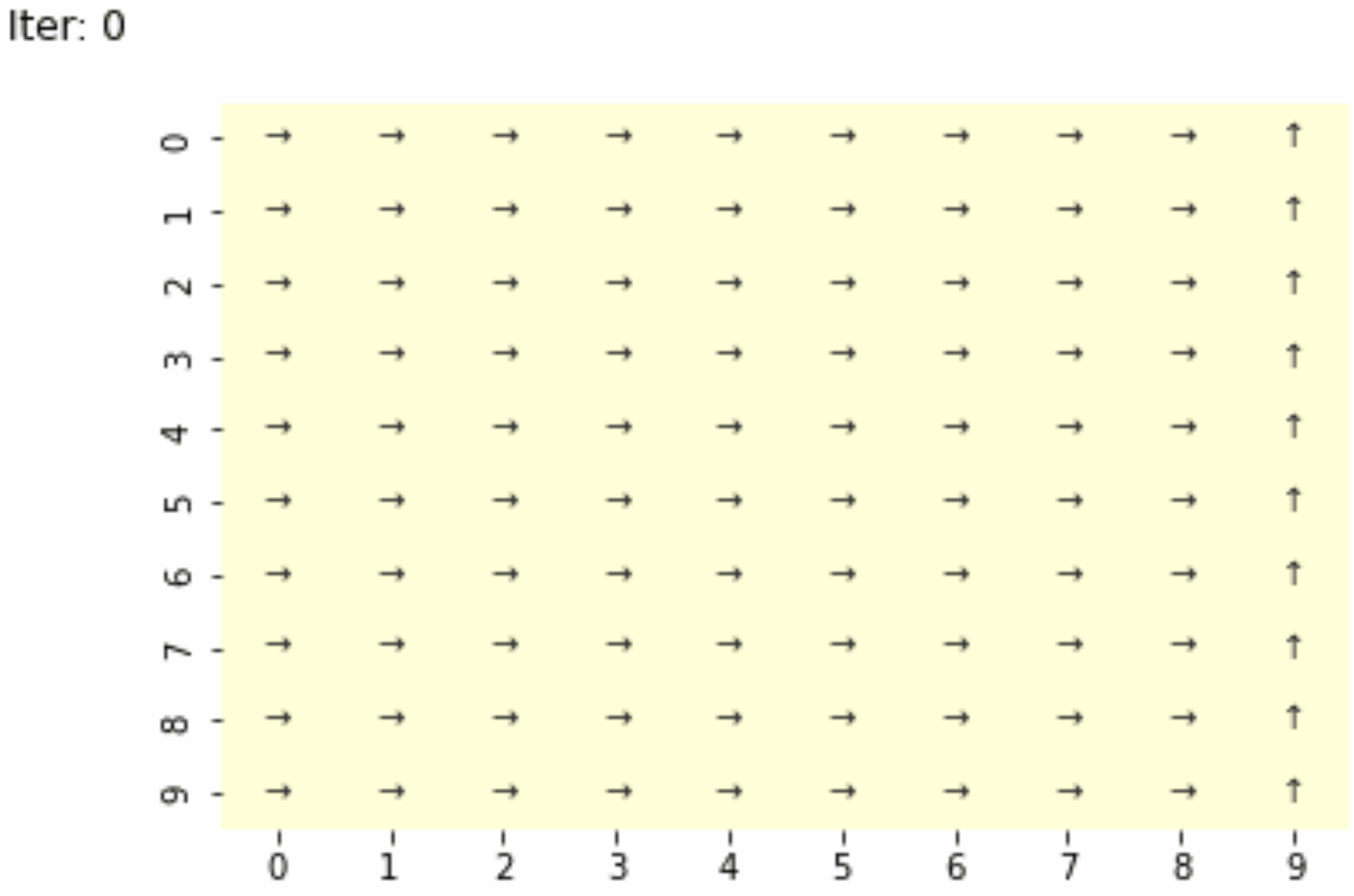
$$\pi^{+}(s) = \arg\min_{a} c(s, a) + \gamma \mathbb{E}_{s' \sim \mathcal{T}(s,a)} V^{\pi}(s')]$$

Estimate value

Improve policy

What happens when the MDP is *unknown*?

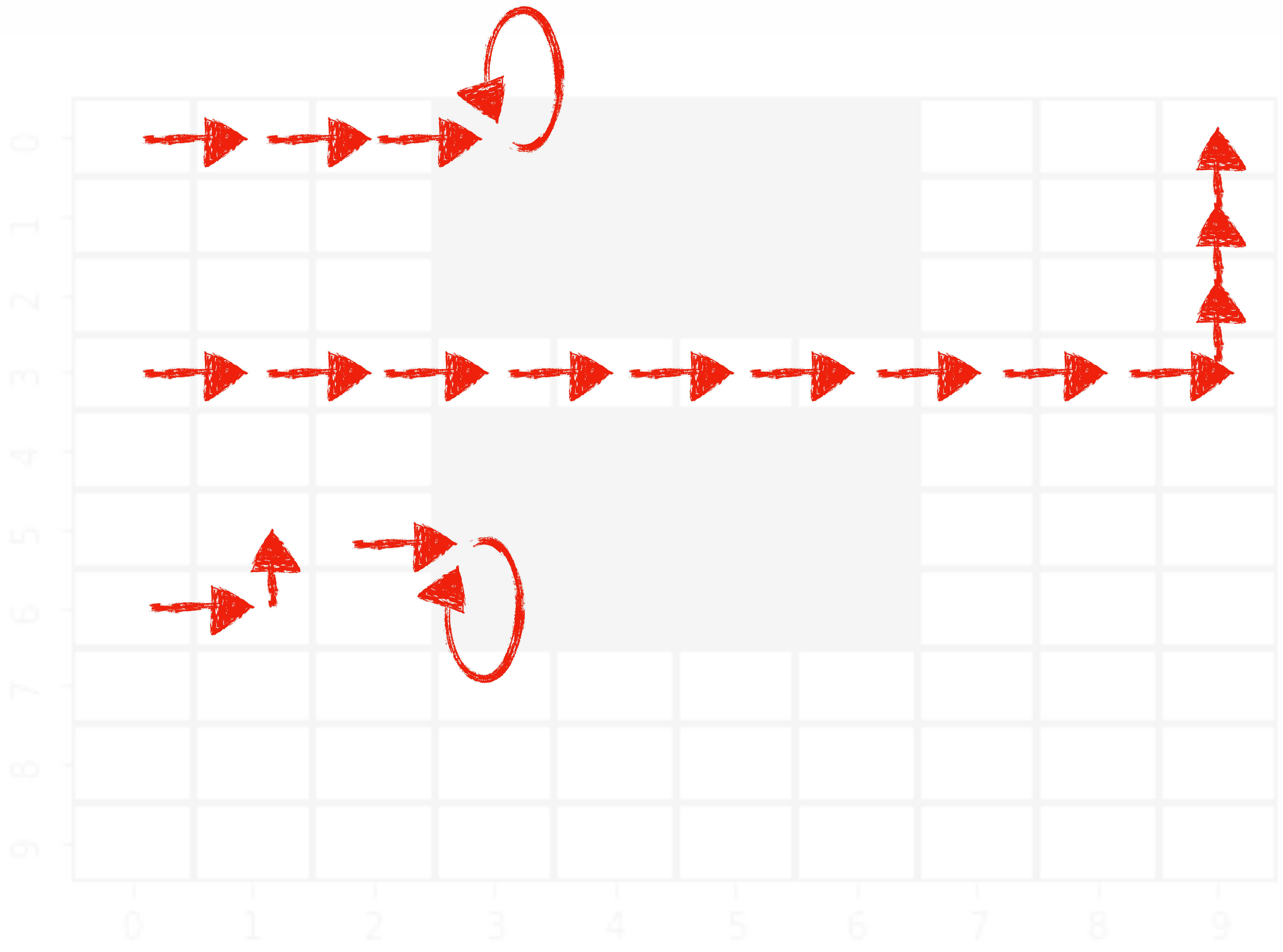
Need to *estimate the value* of policy



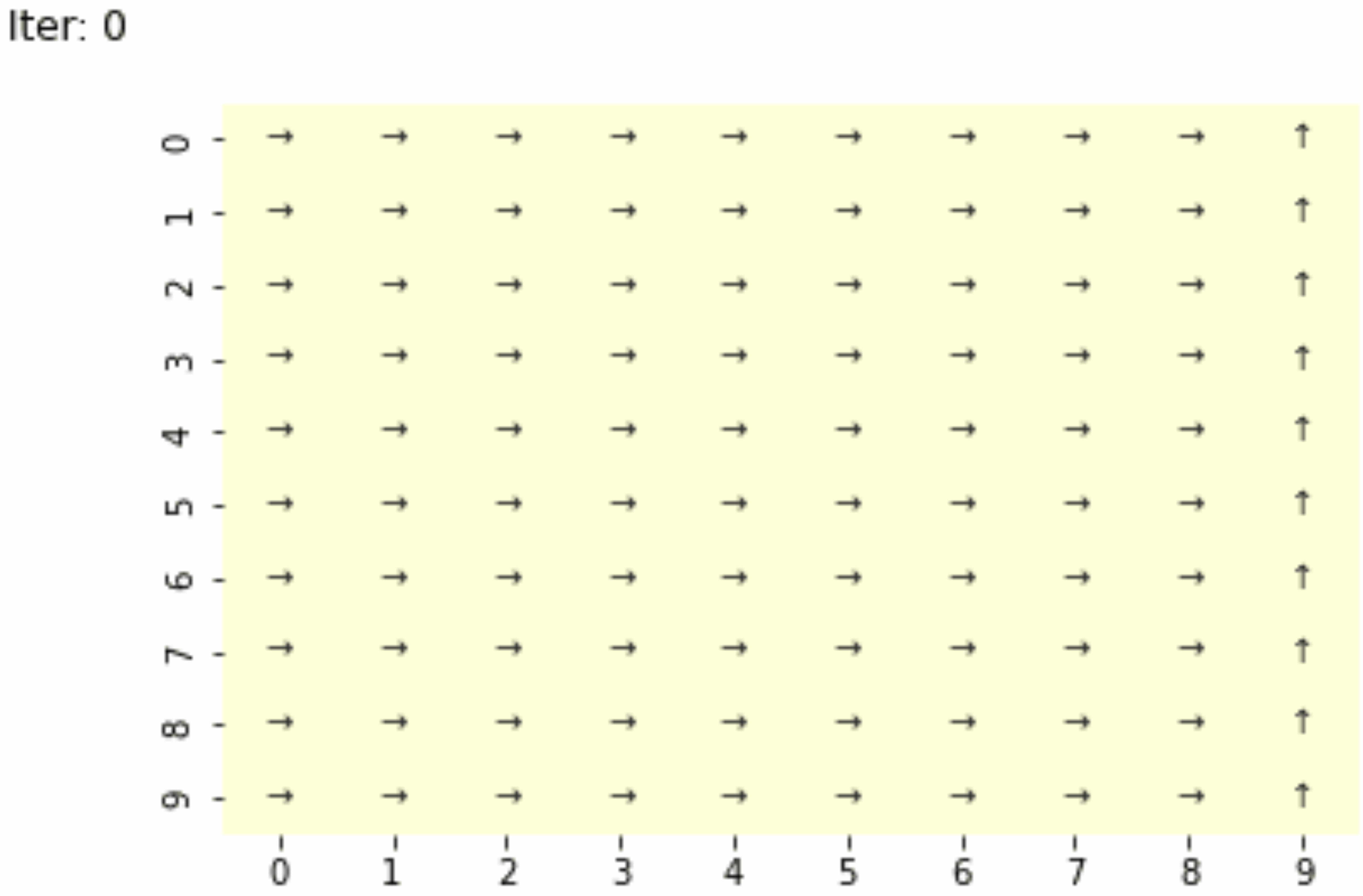
Value $V^{\pi}(s)$

Policy π

Estimate the value of policy from sample rollouts

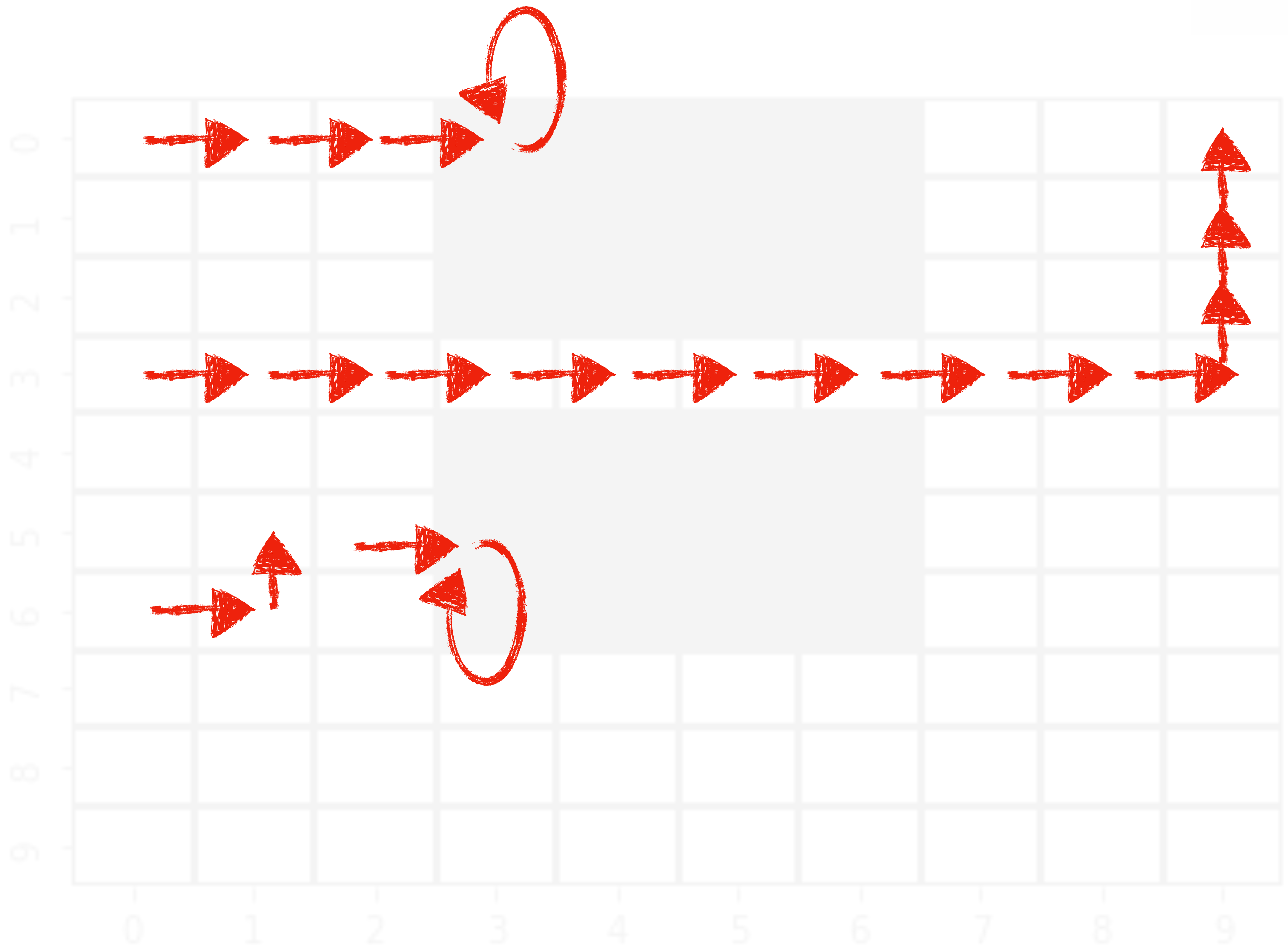


Roll outs

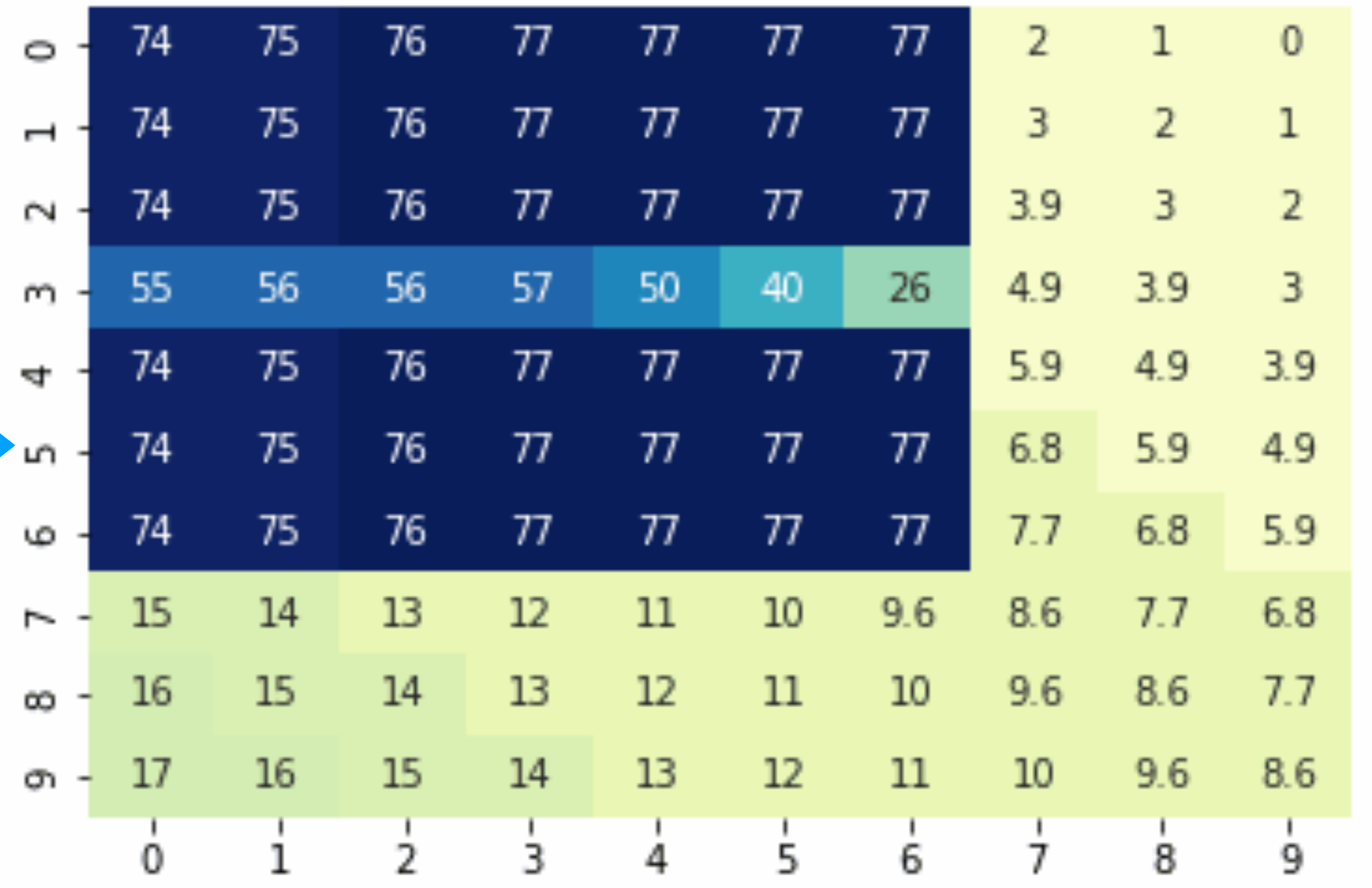


Policy π

Estimate the value of policy from sample rollouts



Roll outs



Value $V^{\pi}(s)$

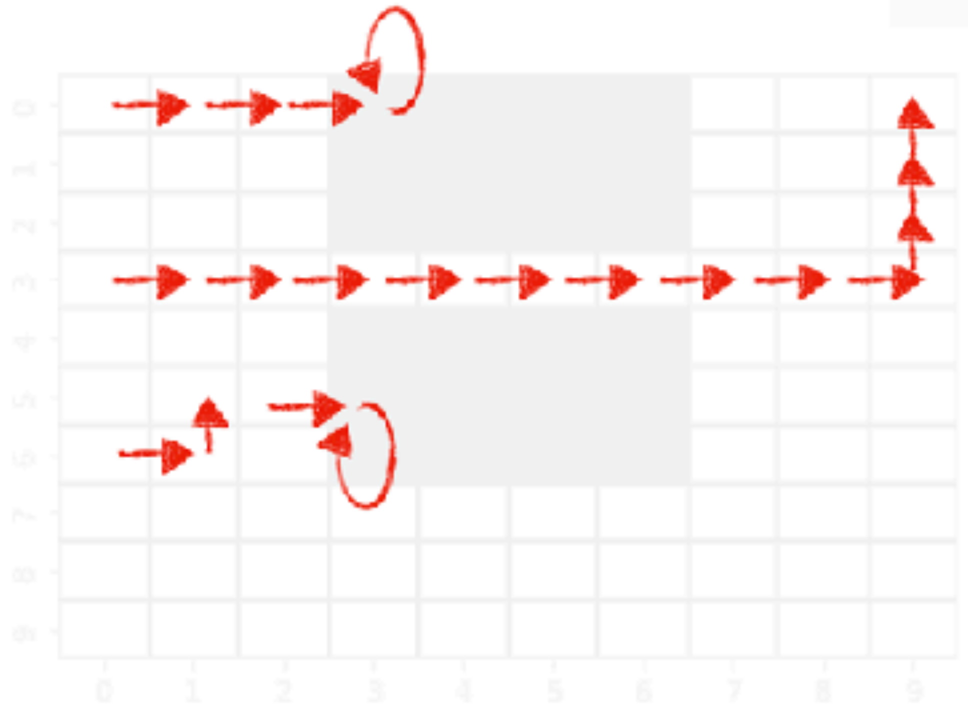
Activity!

Think-Pair-Share

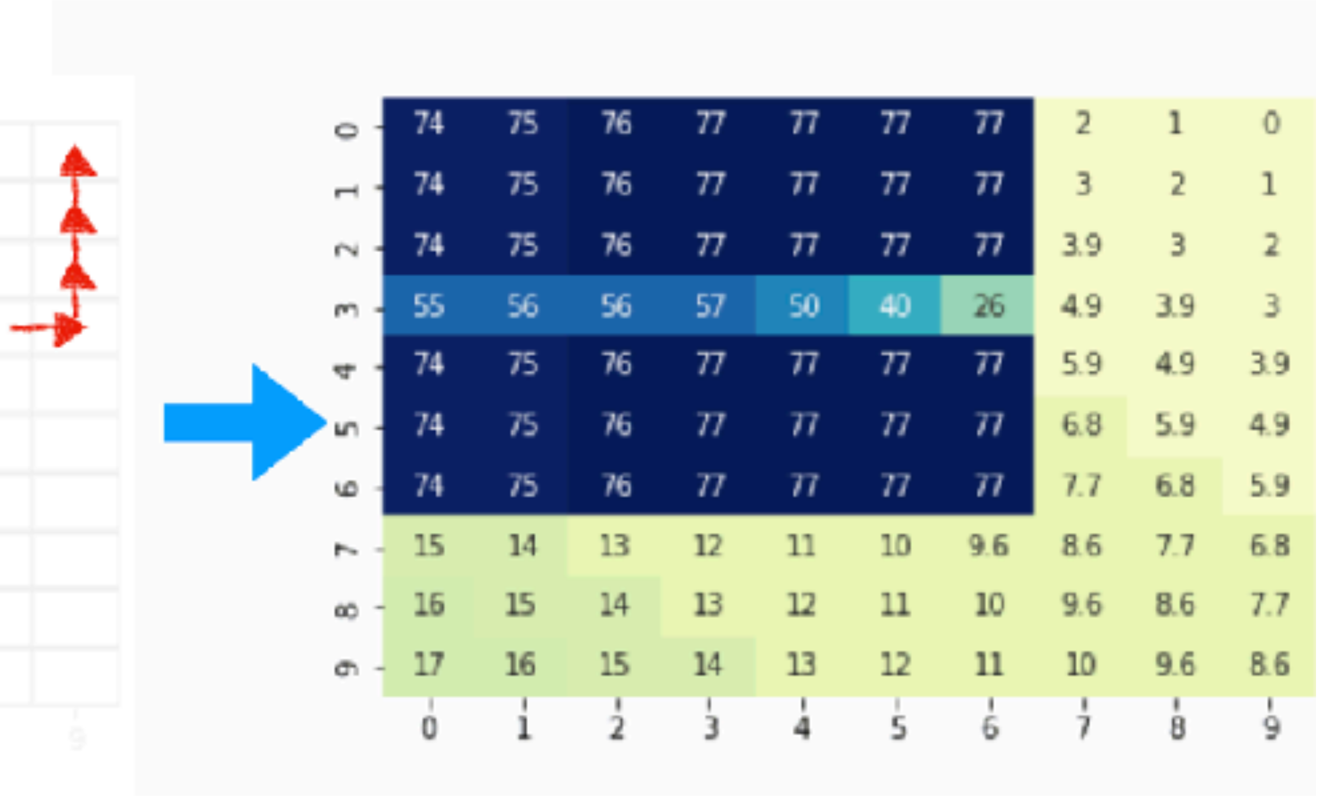
Think (30 sec): Given a bunch of roll-outs, how can you estimate value of a state? (Hint: More than one way!)

Pair: Find a partner

Share (45 sec): Partners exchange ideas

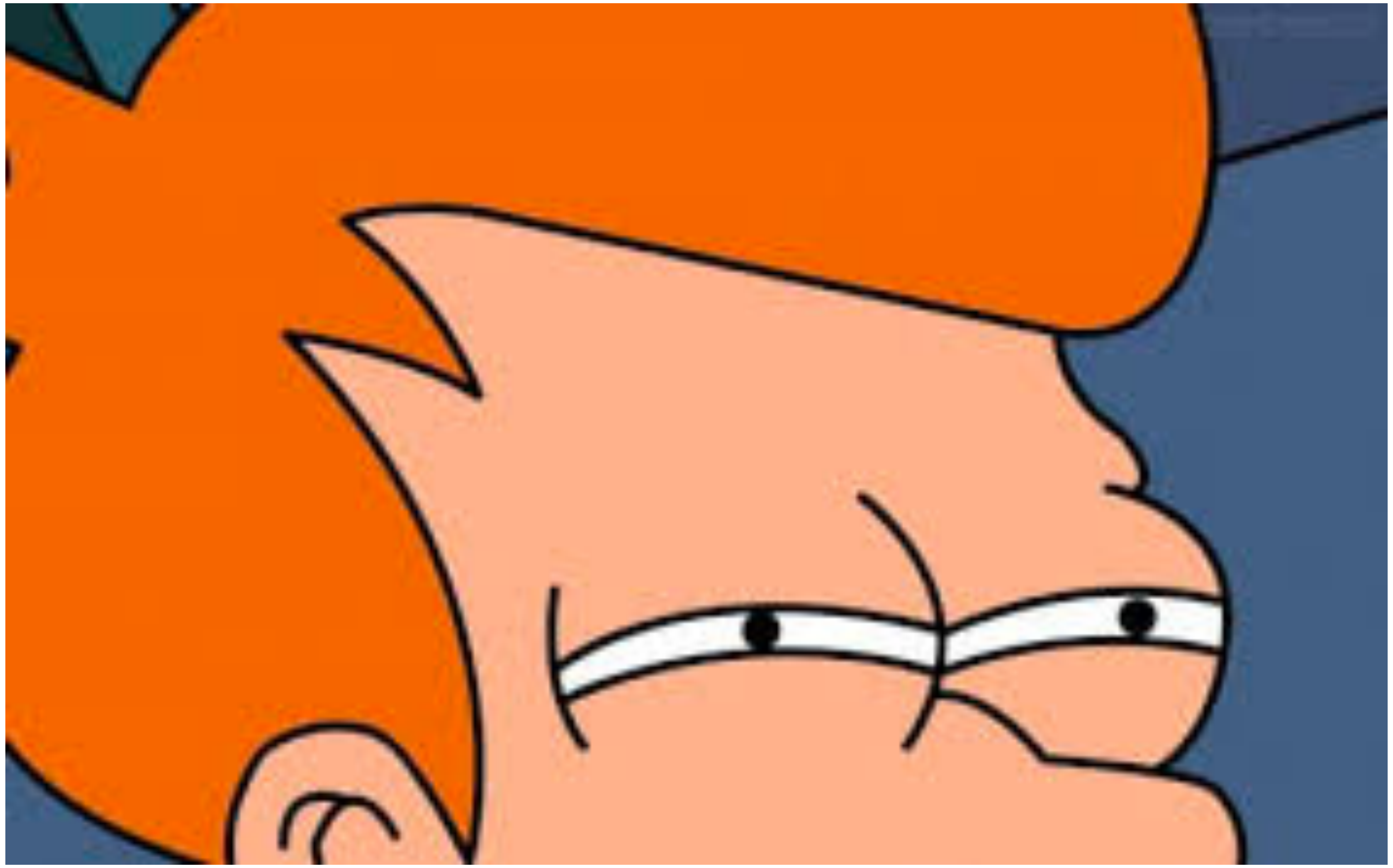


Roll outs



Value $V^{\pi}(s)$

Option 1: Just execute the damn policy!



and look at the returns ..

Monte Carlo Evaluation

Goal: Learn $V^{\pi}(s)$ from complete rollout $s_1, a_1, c_1, s_2, a_2, c_2, \ldots \sim \pi$

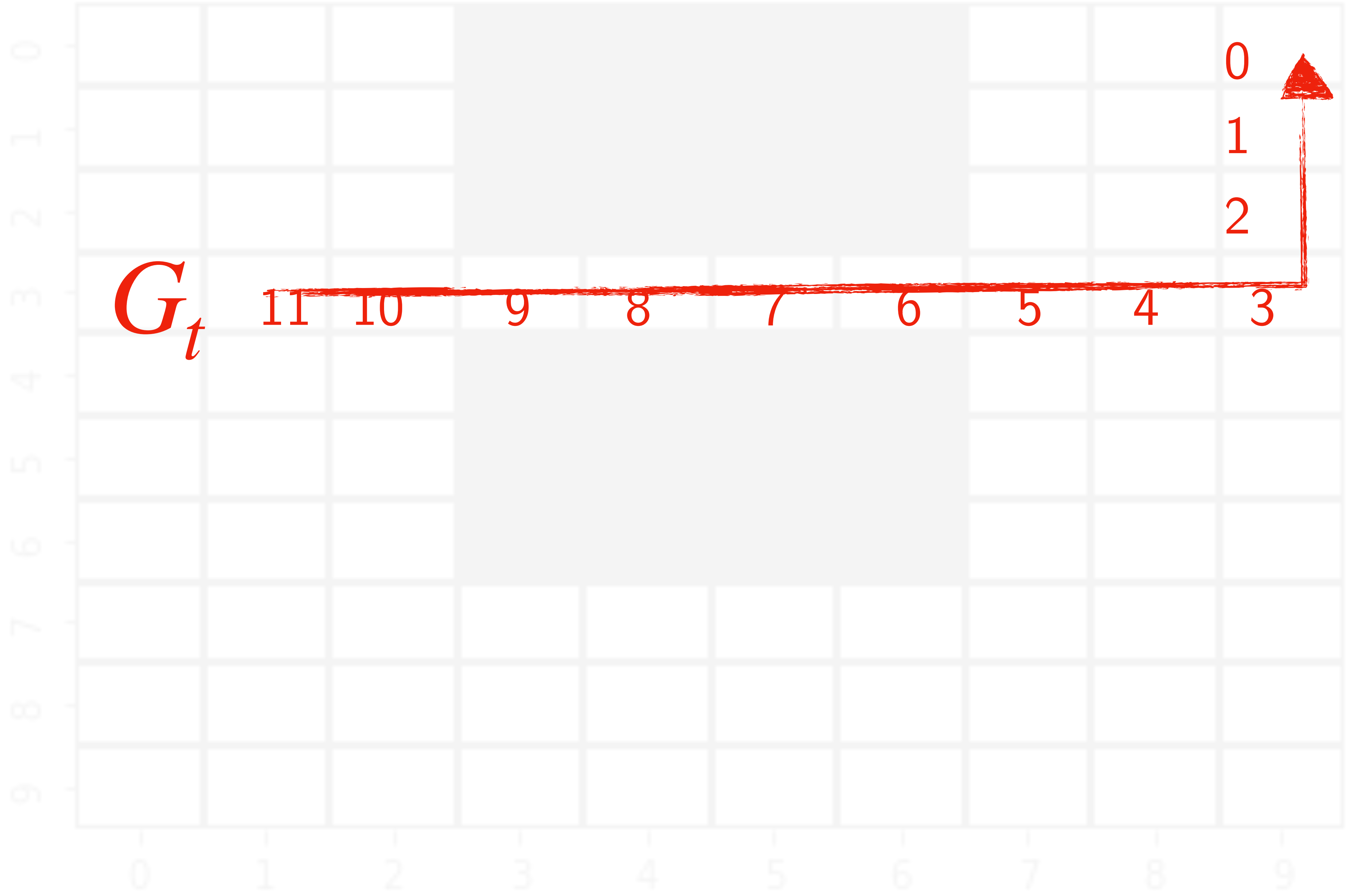
Define: *Return* is the total discounted cost

$$G_t = c_{t+1} + \gamma c_{t+2} + \gamma^2 c_{t+3} + \ldots$$

Value function is the expected return

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t \mid s_t = s]$$

Monte Carlo



For episode in rollouts:

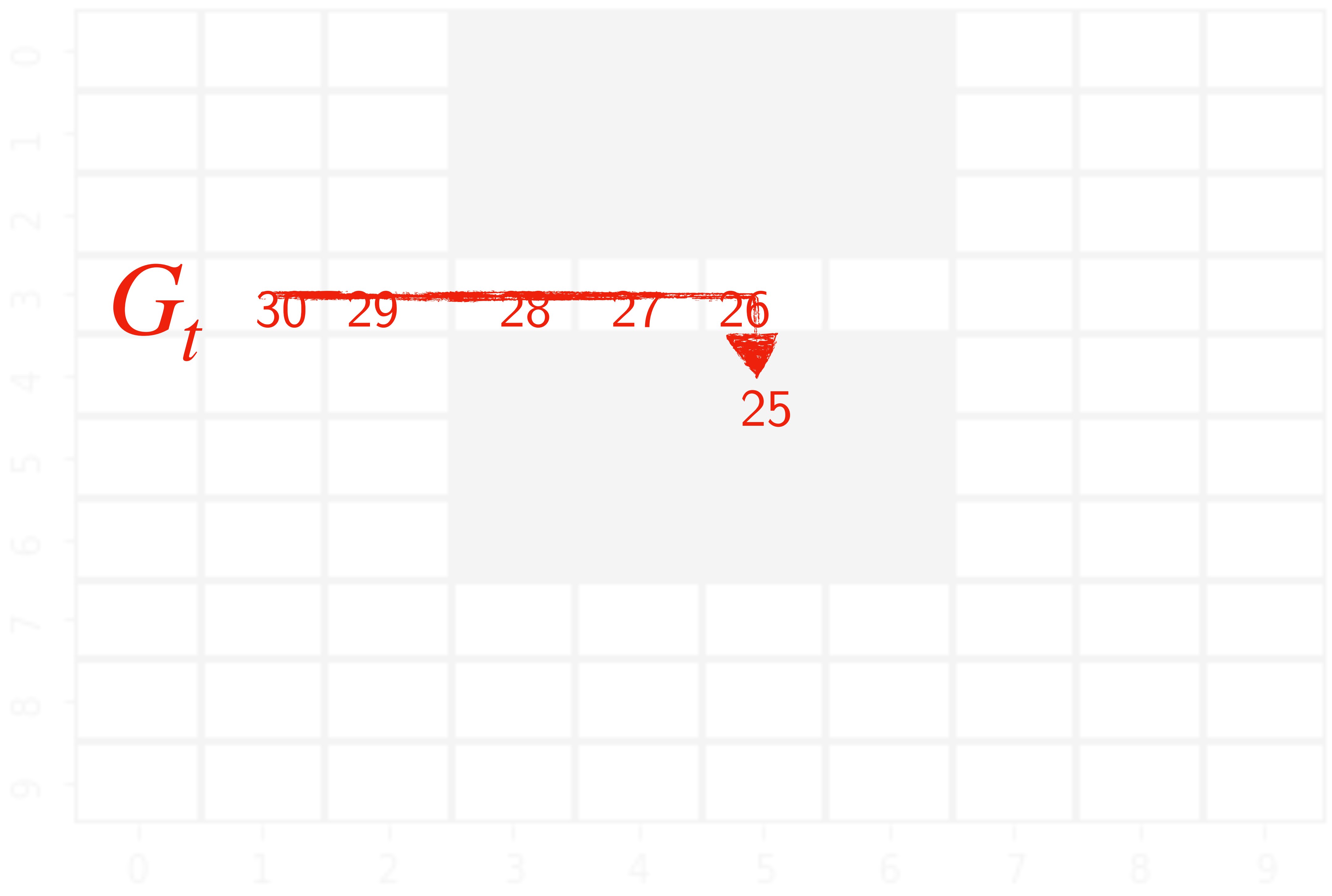
Increment counter $N(s) \leftarrow N(s) + 1$

Increment total return
$S(s) \leftarrow S(s) + G_t$

Update $V(s) = S(s)/N(s)$

Law of large numbers: $V(s) \rightarrow V^{\pi}(s)$ as $N(s) \rightarrow \infty$

Monte Carlo



For episode in rollouts:

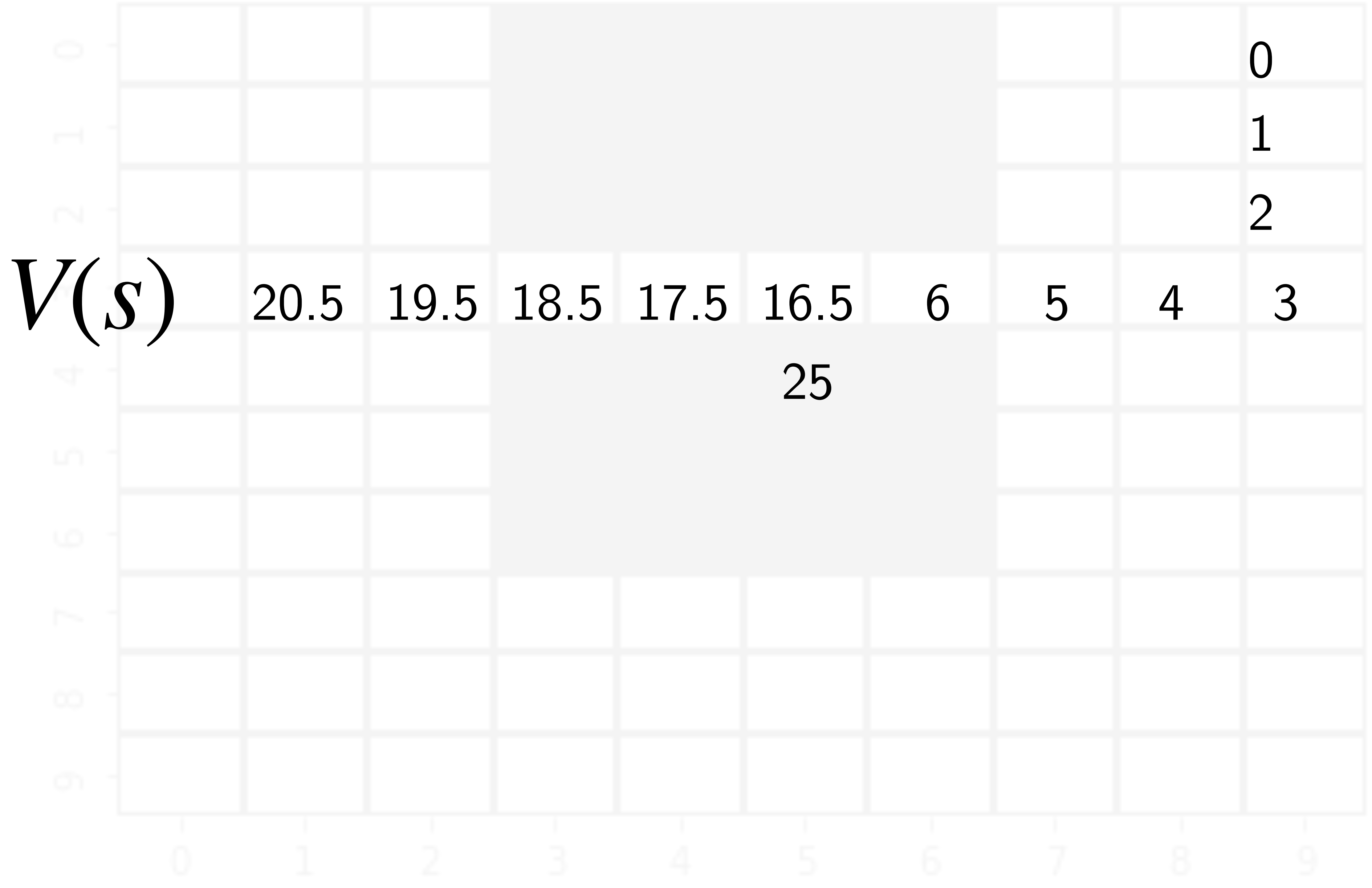
Increment counter $N(s) \leftarrow N(s) + 1$

Increment total return
$S(s) \leftarrow S(s) + G_t$

Update $V(s) = S(s)/N(s)$

Law of large numbers: $V(s) \rightarrow V^{\pi}(s)$ as $N(s) \rightarrow \infty$

Monte Carlo



For episode in rollouts:

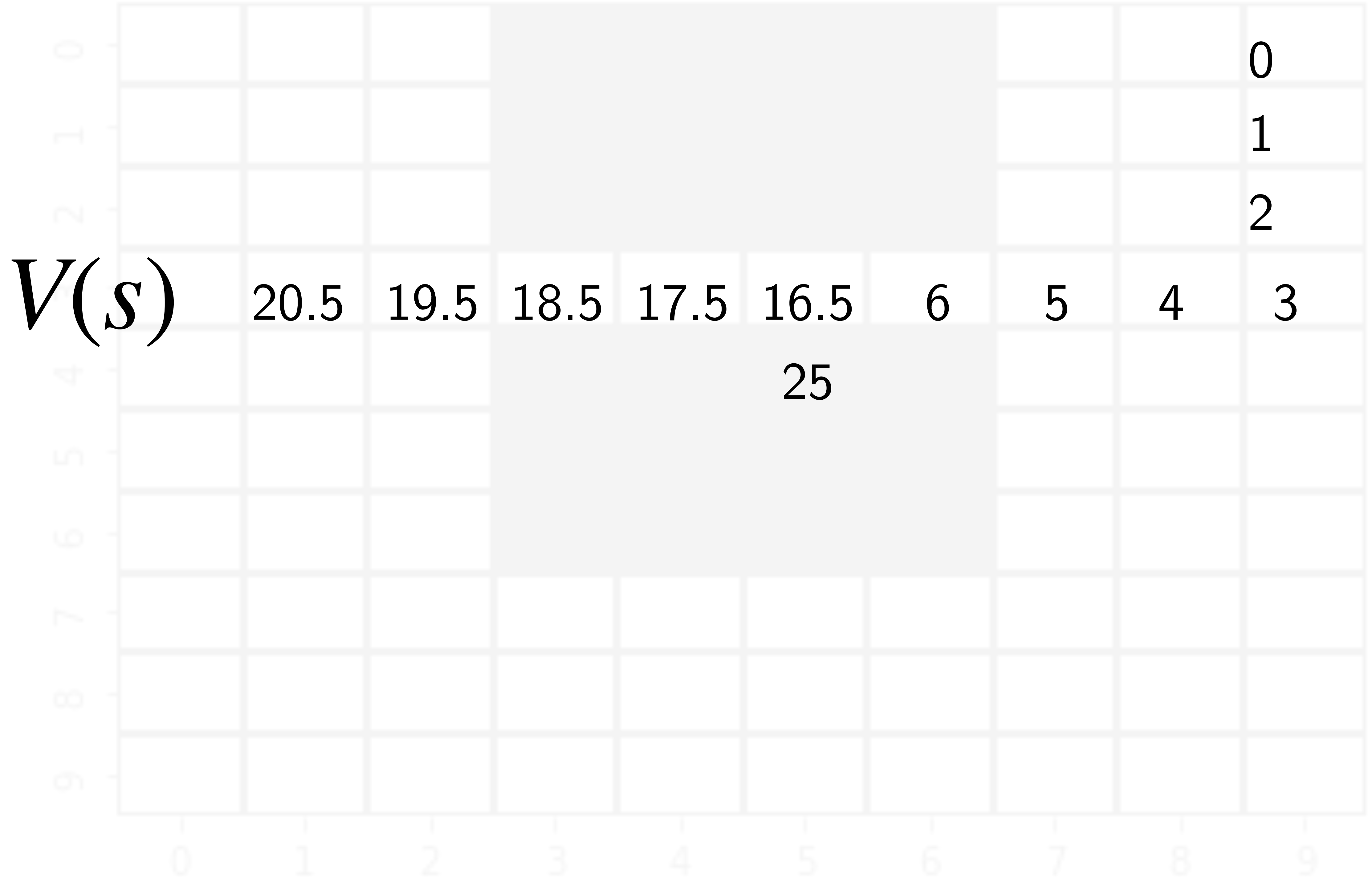
Increment counter $N(s) \leftarrow N(s) + 1$

Increment total return
$S(s) \leftarrow S(s) + G_t$

Update $V(s) = S(s)/N(s)$

Law of large numbers: $V(s) \rightarrow V^{\pi}(s)$ as $N(s) \rightarrow \infty$

Exponential Moving Average MC



For episode in rollouts:

Update $V(s) \leftarrow V(s) + \alpha(G_t - V(s))$

Law of large numbers: $V(s) \rightarrow V^{\pi}(s)$ as $N(s) \rightarrow \infty$

Can we do better than Monte Carlo?

What if we want quick updates?
(No patience to wait till end)

What if we don't have complete episodes?

Option 2: Trust your value estimate

Temporal Difference (TD) learning

Goal: Learn $V^{\pi}(s)$ from traces

$$\left(s_t, a_t, c_t, s_{t+1}\right) \quad \left(s_t, a_t, c_t, s_{t+1}\right) \quad \left(s_t, a_t, c_t, s_{t+1}\right) \quad \left(s_t, a_t, c_t, s_{t+1}\right)$$

Recall value function $V^{\pi}(s)$ satisfies

$$V^{\pi}(s) = c(s, \pi(s)) + \gamma \mathbb{E}_{s'} V^{\pi}(s')$$

TD Idea: Update value using estimate of next state value

$$V(s_t) \leftarrow V(s_t) + \alpha \left(c_t + \gamma V(s_{t+1}) - V(s_t) \right)$$

Temporal Difference Error

TD Learning

For every $\left(s_t, a_t, c_t, s_{t+1}\right)$

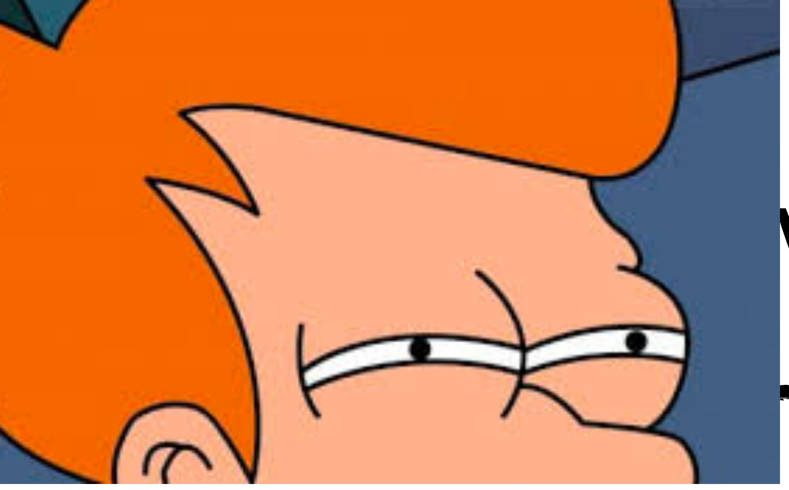
$$V(s_t) \leftarrow V(s_t) + \alpha(c_t + \gamma V(s_{t+1}) - V(s_t))$$

Did you spot the trick?

$$V^{\pi}(s) = c(s, \pi(s)) + \gamma \mathbb{E}_{s'} V^{\pi}(s')$$

$$\Updownarrow$$

$$V(s_t) \leftarrow V(s_t) + \alpha(c_t + \gamma V(s_{t+1}) - V(s_t))$$



Monte-Carlo

Temporal Difference

$$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$$

$$V(s) \leftarrow V(s) + \alpha(c + \gamma V(s') - V(s))$$

Monte-Carlo	Temporal Difference
Zero Bias	Can have bias
High Variance	Low Variance
Always convergence (Just have to wait till heat death of the universe)	May *not* converge if using function approximation

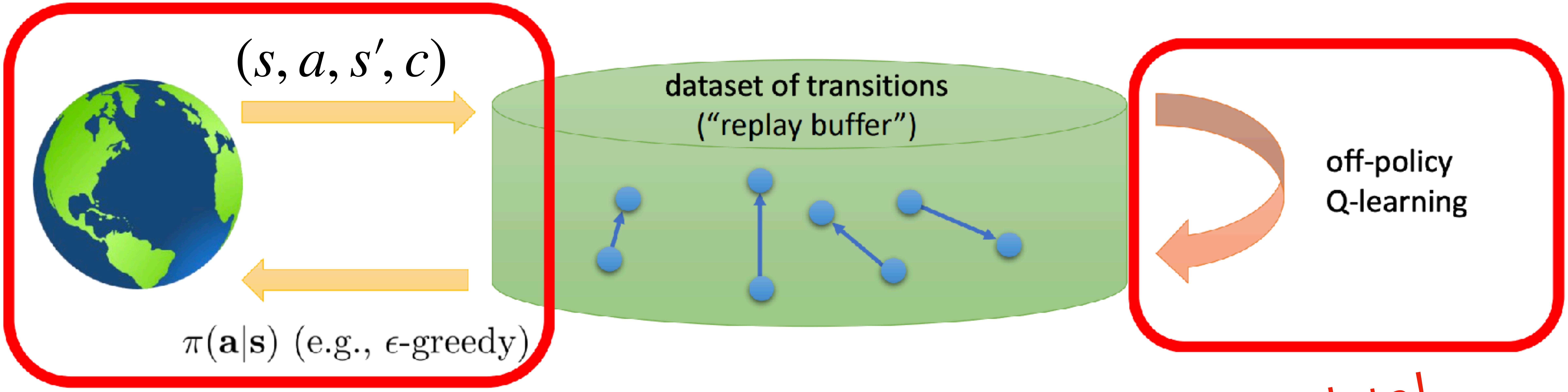
We have been talking about trying to learn the value of a given policy π

$V^{\pi}(s)$ / $Q^{\pi}(s, a)$

What if we wanted to learn the optimal value function

$V^{*}(s)$ / $Q^{*}(s, a)$

Q-learning: Learning off-policy



Can learn from any data!

For every (s_t, a_t, c_t, s_{t+1})

$$Q^*(s_t, a_t) = Q^*(s_t, a_t) + \alpha(c(s_t, a_t) + \gamma \min_{a'} Q^*(s_{t+1}, a') - Q^*(s_t, a_t))$$

Is this ... magic?

We just learned in IL how distribution shift is a big deal ...

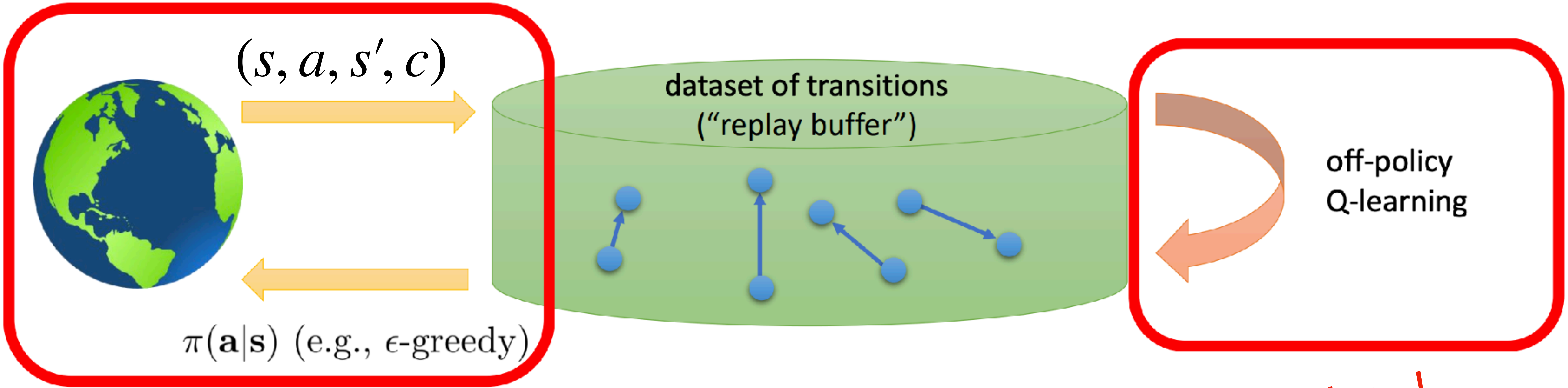
It's not magic. Q-learning relies on a set of assumptions:

1. Each state-action is visited *infinite* times
2. Learning rate α must be annealed over time

QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation

Training time
-
=
Reward: Grasp success determined by subtracting pre and post-drop images
Distributed RL
State, Action, Reward
Learned weights
Inference time
State: 472x472 Image and gripper aperture
Camera
Robot
Action: Gripper displacement and aperture
Critic Function
Q(State, Action)
Action proposals
Q-Values
Cross-Entropy Method
arg max Q(State, Action)
Action

Q-learning: Learning off-policy

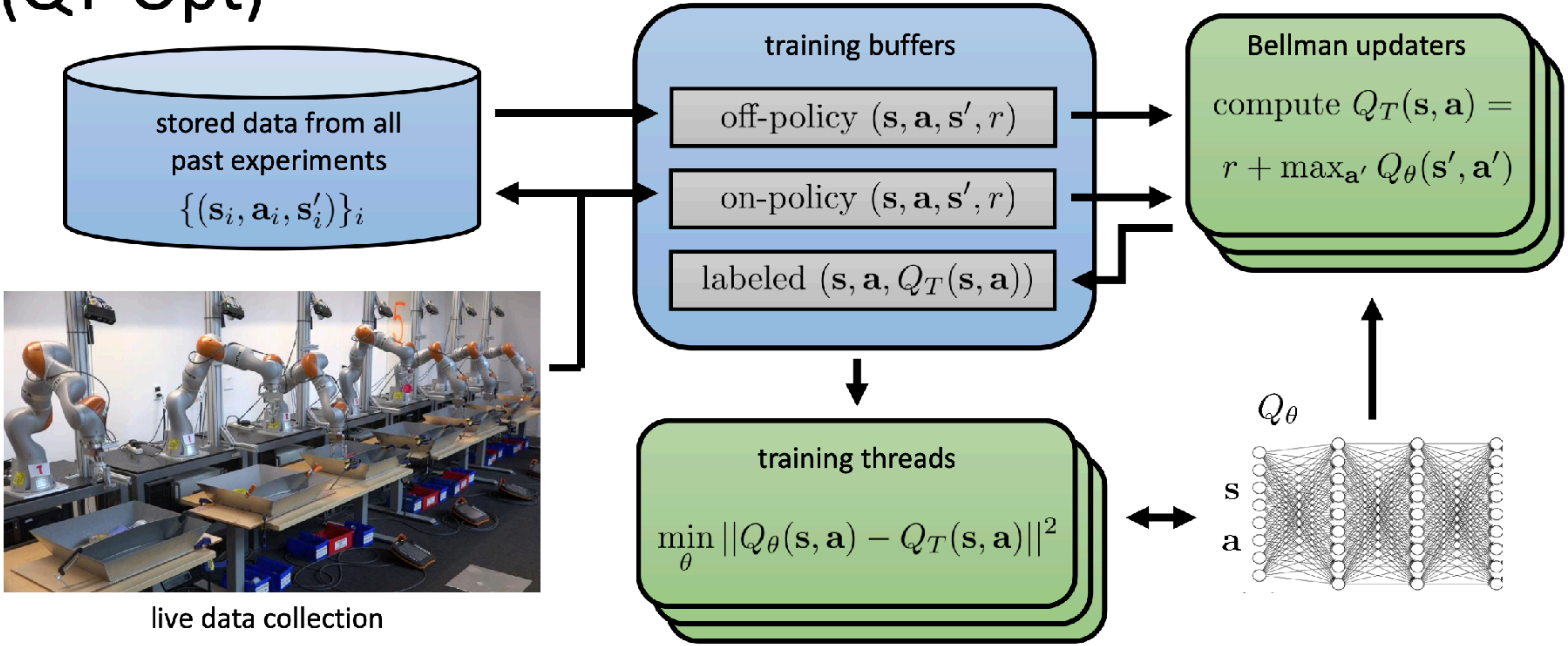


Can learn from any data!

For every (s_t, a_t, c_t, s_{t+1})

$$Q^*(s_t, a_t) = Q^*(s_t, a_t) + \alpha(c(s_t, a_t) + \gamma \min_{a'} Q^*(s_{t+1}, a') - Q^*(s_t, a_t))$$

Large-scale Q-learning with continuous actions (QT-Opt)



Kalashnikov, Irpan, Pastor, Ibarz, Herzong, Jang, Quillen, Holly, Kalakrishnan, Vanhoucke, Levine. **QT-Opt: Scalable Deep Reinforcement Learning of Vision-Based Robotic Manipulation Skills**

Making Q-learning better!

Problem: Q-learning suffers from an estimation bias $\min_{a'} Q^*(s_{t+1}, a')$

Solution: Double Q-learning $Q^*(s_{t+1}, \arg\min_{a'} \tilde{Q}(s_{t+1}, a'))$

Problem: Q-learning samples uniformly from replay buffer

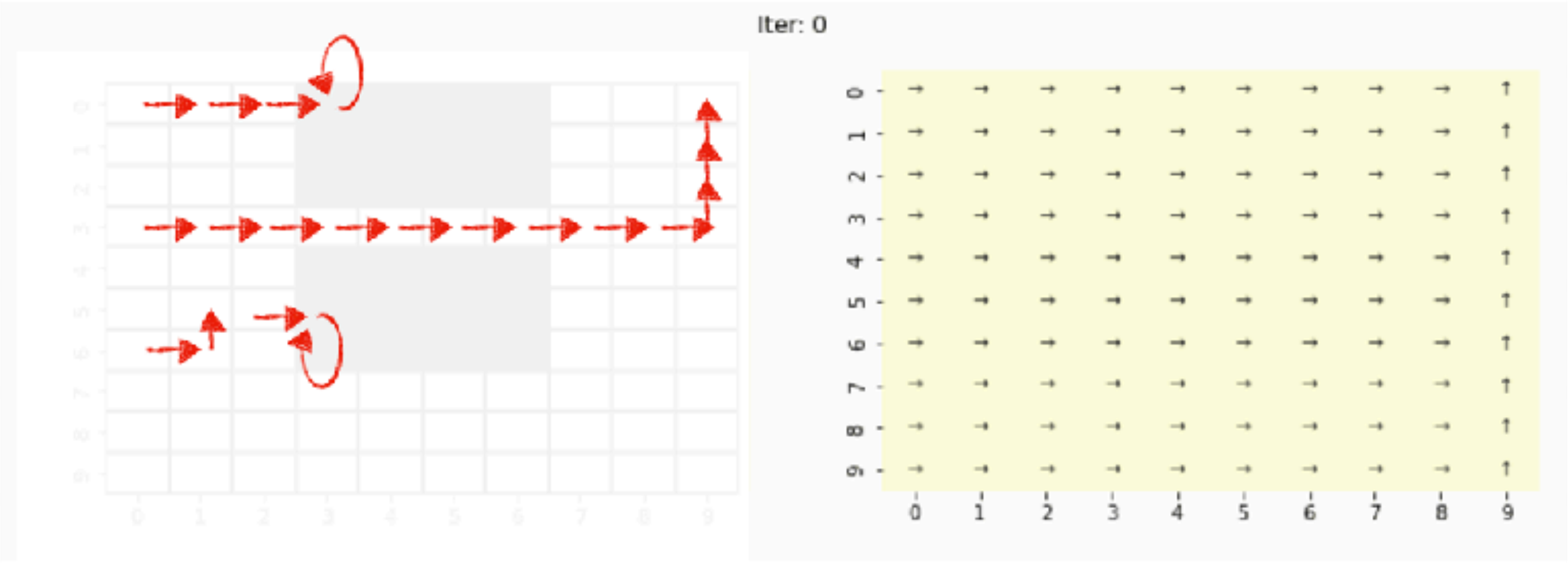
Solution: Prioritized DQN - samples states with higher bellman error

Problem: Q-learning doesn't seem to learn ....

Solution: Start with high exploration + learning rate, anneal!

tl;dr

Estimate the value of policy from sample rollouts



Roll outs | Policy π

Monte-Carlo	Temporal Difference
$V(s) \leftarrow V(s) + \alpha(G_t - V(s))$	$V(s) \leftarrow V(s) + \alpha(c + \gamma V(s') - V(s))$
Zero Bias	Can have bias
High Variance	Low Variance
Always convergence (Just have to wait till heat death of the universe)	May *not* converge if using function approximation

Q-learning: Learning off-policy

For every (s_t, a_t, c_t, s_{t+1})

$$Q^*(s_t, a_t) = Q^*(s_t, a_t) + \alpha(c(s_t, a_t) + \gamma \min_{a'} Q^*(s_{t+1}, a') - Q^*(s_t, a_t)$$

Notice we are *not* approximating $Q^\pi(s_t, a_t)$

We don't even care about π

We can learn from any data!