Reinforcement Learning – Part II

Khimya Khetarpal

School of Computer Science, McGill University, Mila Montreal

Slides adapted from Doina Precup, David Silver, and Rich Sutton's book

AI4Good Summer Lab 2020

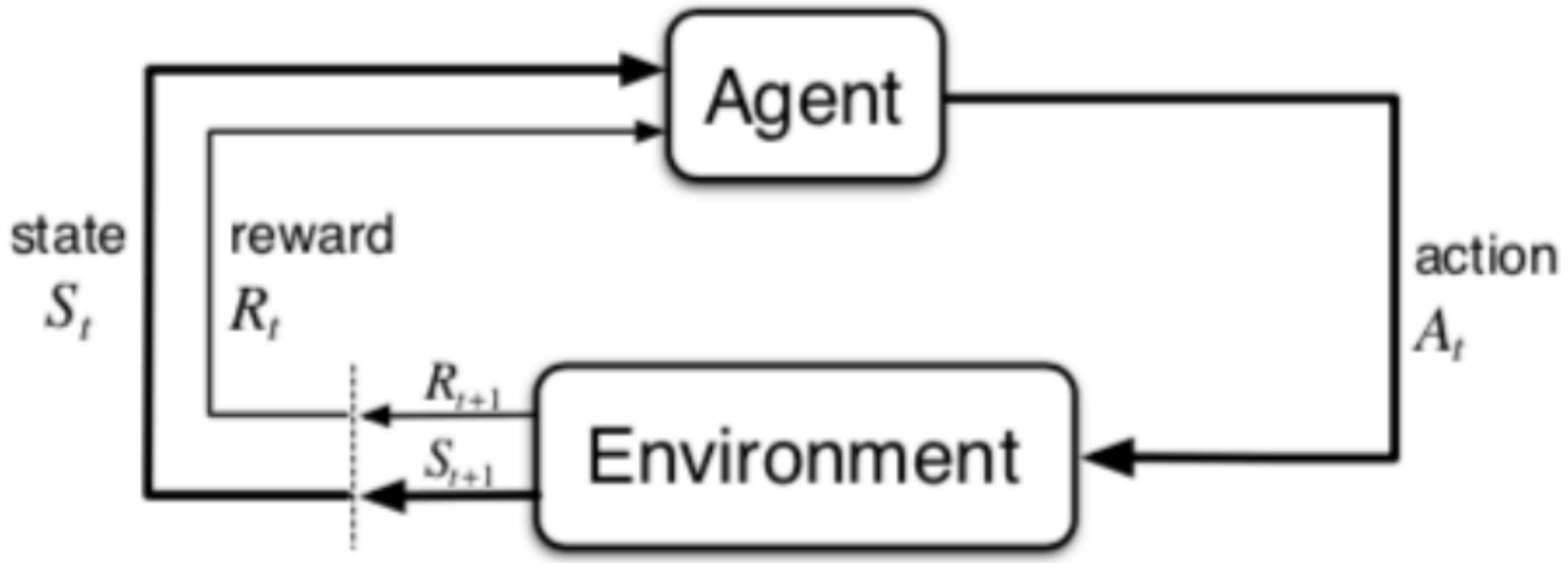
Who am I?

2017-	Mila McGill	Ph.D. in Computer Science
2016-17	intel	Perceptual Computing Engineer
2014-16	UF UNIVERSITY of FLORIDA	Masters in Computer Engineering
2013-14	IIT KANPUR Indian Institute of Technology, Kanpur	Research Associate
2011-13	BOSCH	Embedded Systems Engineer
2007-11	VIT Vellore Institute of Technology (Deemed to be University under section 3 of UGC Act, 1956)	Undergrad in Electronics & Communications

Outline

- ❑ Recap
- ❑ Markov Decision Processes
- ❑ Bellman Equations
- ❑ Dynamic Programming
- ❑ Temporal Difference Learning
- ❑ A Unified View of Reinforcement Learning

Agent-Environment Interaction



Agent-Environment Interaction
(Fig. from Sutton & Barto)

At each time step, the ***agent***:

- Observes state $S_t \in S$
- Executes action $A_t \in A$
- Receives reward R_t

At each time step, the ***environment***:

- Receives action A_{t+1}
- Emits new state S_{t+1}
- Emits scalar reward R_{t+1}

Markov Property

The future is independent of the past given the present.

$$P(S_{t+1} \mid S_t, A_t) = P(S_{t+1} \mid S_1, A_1, S_2, A_2 \ldots . S_t, A_t)$$

- The state captures all relevant information from the history
- The state is a sufficient statistic of the future
- ***Markovian assumption*: ** current state provides sufficient information to describe the distribution of immediate reward and next state

Markov Decision Processes

- **Markov decision processes (MDP)** formally describes an environment for reinforcement learning

- A finite discrete-time MDP is a tuple $\langle S, A, R, P, \gamma \rangle$

Markov Decision Processes

- **Markov decision processes (MDP)** formally describes an environment for reinforcement learning
- A finite discrete-time MDP is a tuple $\langle S, A, R, P, \gamma \rangle$
- One-step ***model*** of the environment:
  - One-step ***state-transition probabilities***

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

  - One-step ***expected rewards***

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

Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

$$v_\pi(s) = E_\pi[G_t \mid S_t = s]$$

Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

$$v_\pi(s) = E_\pi[G_t \,|\, S_t = s]$$

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \,|\, S_t = s]$$

Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

$$v_\pi(s) = E_\pi[G_t \,|\, S_t = s]$$

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \,|\, S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s]$$

Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

$$v_\pi(s) = E_\pi[G_t \,|\, S_t = s]$$

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \,|\, S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma E_\pi[G_{t+1} \,|\, S_{t+1} = s']]$$

Value Function

- The ***value of being in a state*** is the expected return starting from state s, and then following policy π .

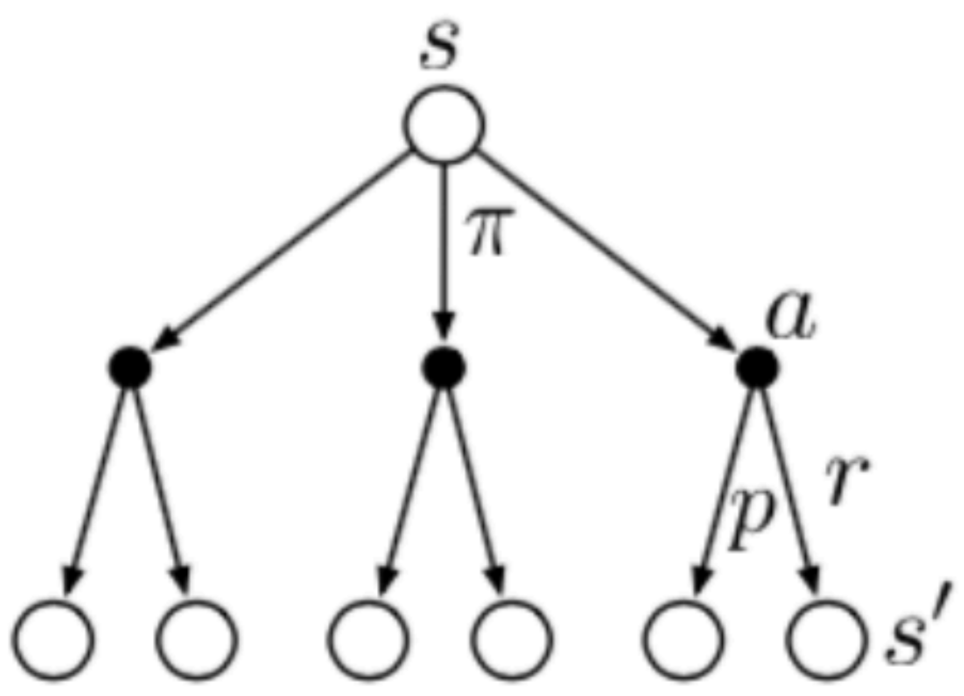
$$v_\pi(s) = E_\pi[G_t \,|\, S_t = s]$$

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \,|\, S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma E_\pi[G_{t+1} \,|\, S_{t+1} = s']]$$

$$= \sum_{a \in A} \pi(a \,|\, s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right]$$



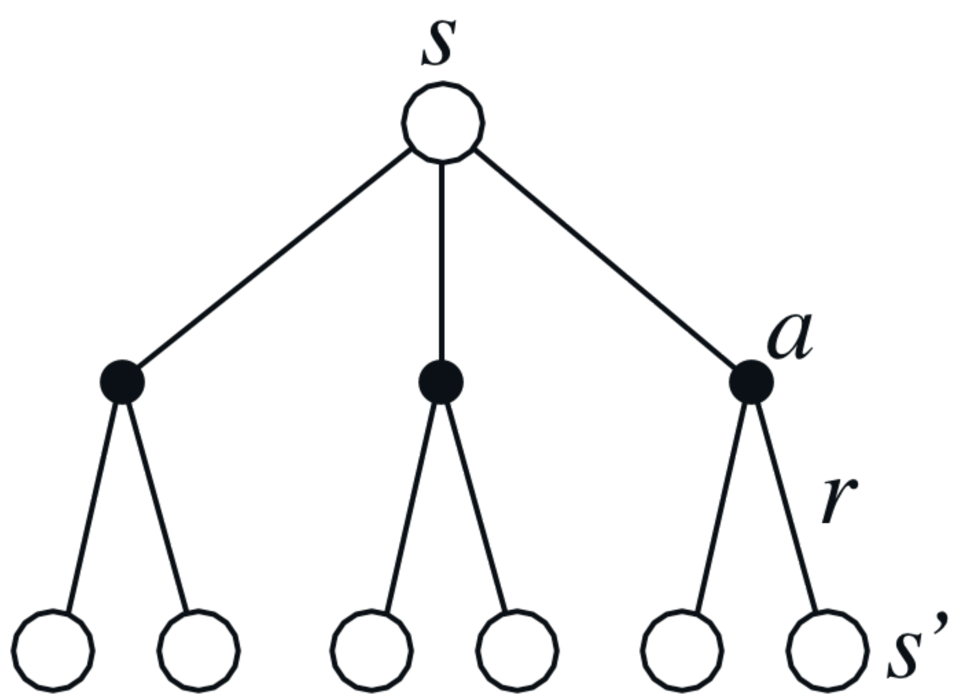
- Values can be written in terms of successor values: ***Bellman equations***

More on the Bellman Equation

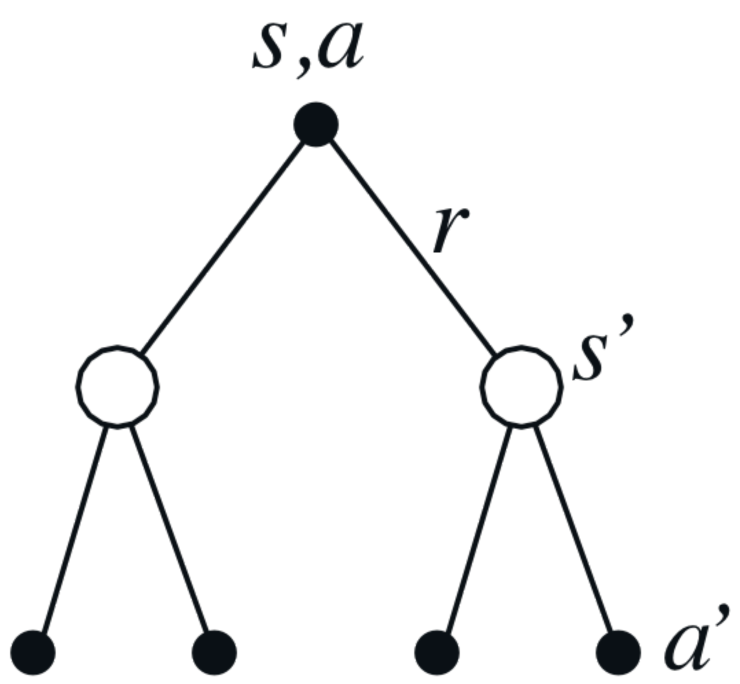
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big]$$

This is a set of equations (in fact, linear), one for each state.
The value function for π is its unique solution.

Backup diagrams:



for v_π



for q_π

Action-Value Function

- The ***value of taking an action a in a state*** s under policy π

$$q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a]$$

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' \mid s') q_\pi(s', a')$$

Optimal Policies and Value Functions

- Value functions define a partial order over policies:

$$\pi_1 \geq \pi_2 \textbf{ iff } v_{\pi_1}(s) \geq v_{\pi_2}(s), \forall s \in S$$

- If a policy is better than another policy if and only if, it generates at least the same amount of return at all states

- The optimal state-value function $v*(s)$ is the maximum value function over all policies

$$\mathbf{v}^*(\mathbf{s}) = \max_{\pi} \mathbf{v}_{\pi}(\mathbf{s})$$

- The optimal action-value function $q^*(s, a)$ is the maximum action-value function over all policies

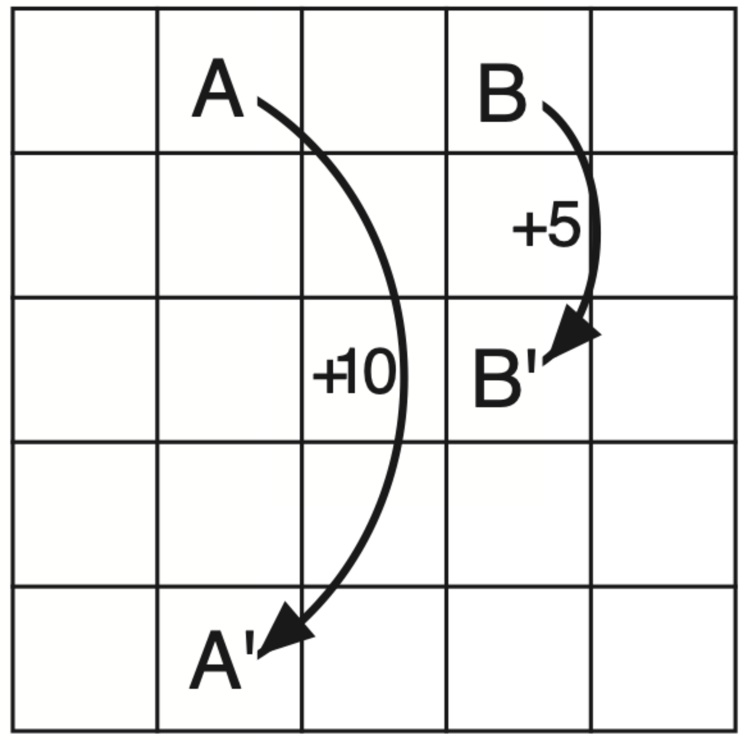
$$\mathbf{q}^*(\mathbf{s}, \mathbf{a}) = \max_{\pi} \mathbf{q}_{\pi}(\mathbf{s}, \mathbf{a})$$

Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to v_* is an optimal policy.

Therefore, given v_*, one-step-ahead search produces the long-term optimal actions.

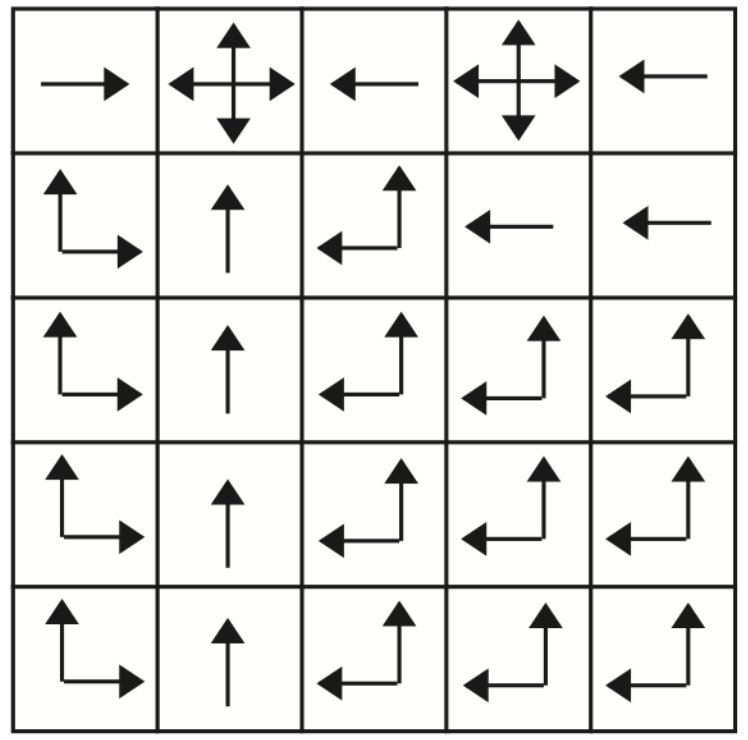
E.g., back to the gridworld:



a) gridworld

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_*

What About Optimal Action-Value Functions?

Given q_*, the agent does not even have to do a one-step-ahead search:

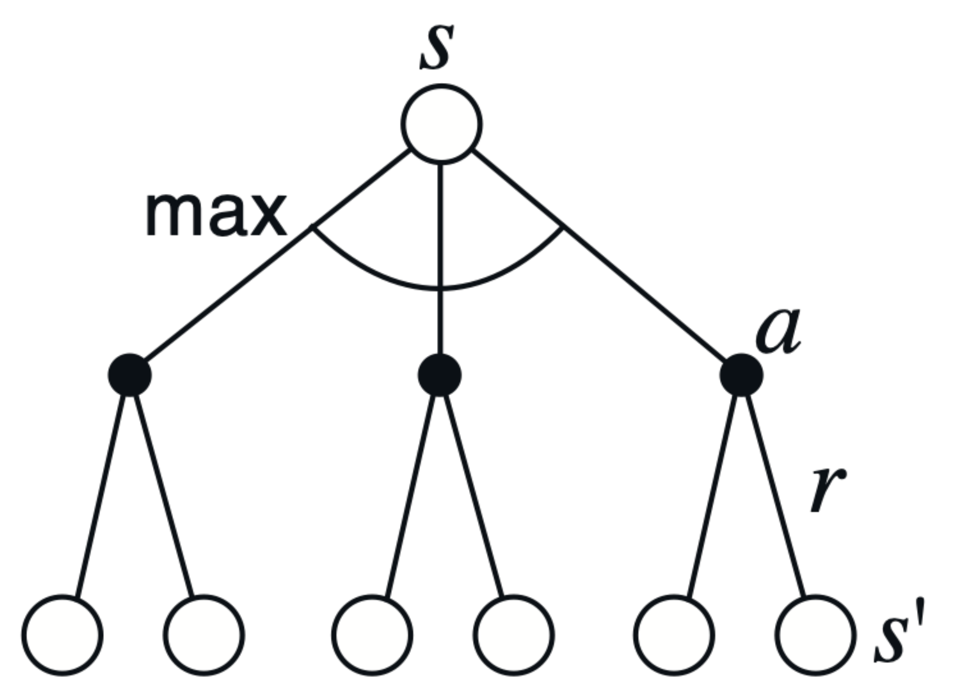
$$\pi_*(s) = \arg\max_a q_*(s,a)$$

Bellman Optimality Equation for v_*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$
\begin{aligned}
v_*(s) &= \max_a q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big].
\end{aligned}
$$

The relevant backup diagram:

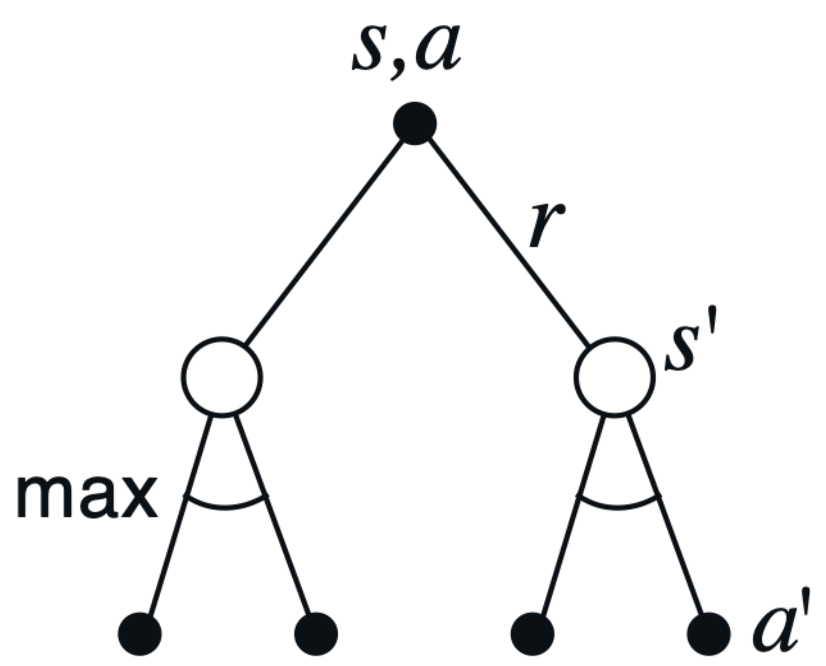


v_* is the unique solution of this system of nonlinear equations.

Bellman Optimality Equation for q_*

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\
&= \sum_{s',r} p(s', r|s,a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big].
\end{aligned}
$$

The relevant backup diagram:



q_* is the unique solution of this system of nonlinear equations.

Dynamic Programming

Key Idea: Turn Bellman equations into update rules

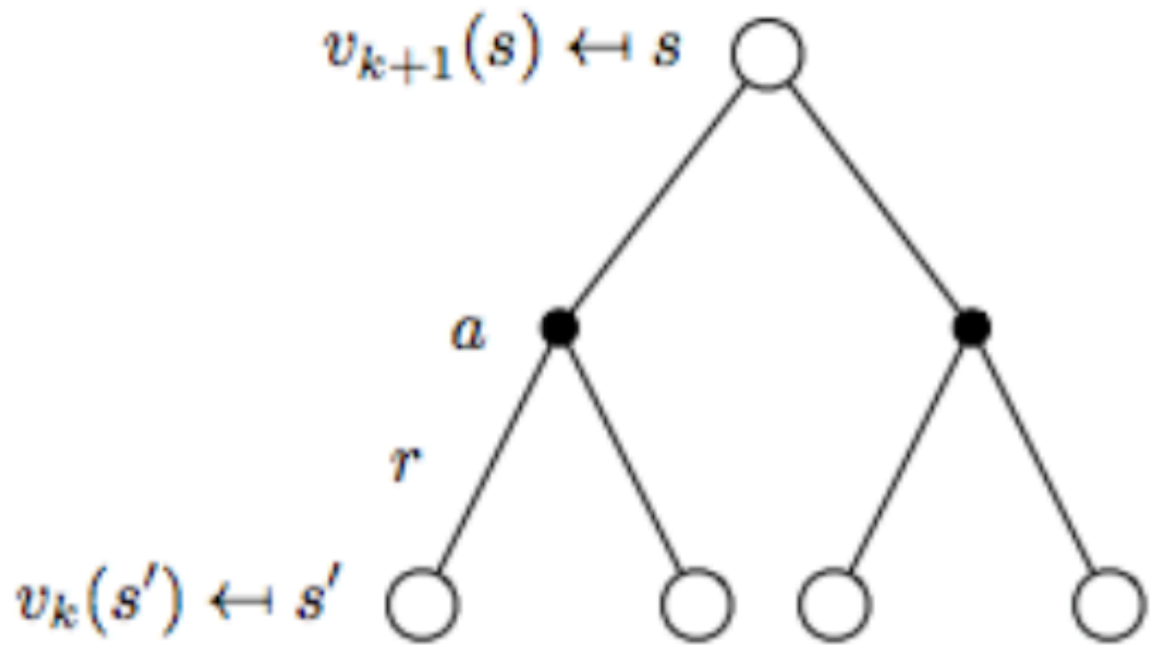
For instance, we can use DP for

- ✓ Iterative Policy Evaluation
- ✓ Policy Iteration
- ✓ Value Iteration

Iterative Policy Evaluation (Prediction)

- **Problem:** Evaluate a given policy π

- **Solution**: iterative application of Bellman expectation backup
  - $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$
  - Using synchronous backups,
    - At each iteration $k+1$
    - For all states $s \in S$
    - Update $v_{k+1}(s)$ from $v_k(s')$
    - where s' is a successor state of s



$$v_{k+1} = \sum_{a \in A} \pi(a \mid s)\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')\right)$$

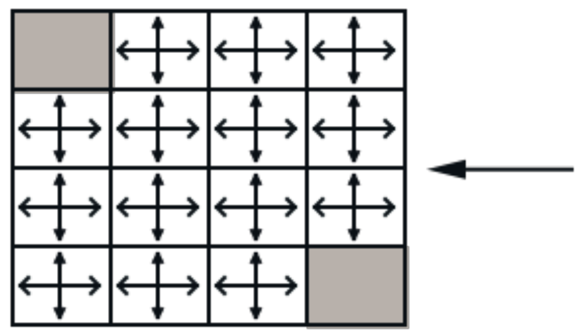
Example: Small Gridworld

V_k for the Random Policy

Greedy Policy w.r.t. V_k

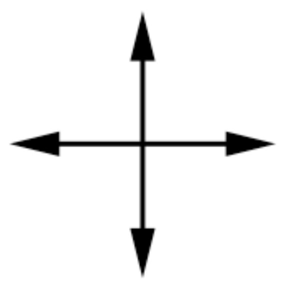
$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

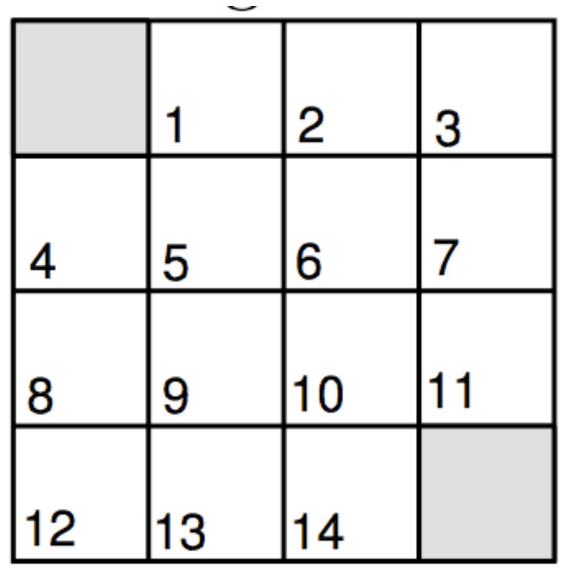


random policy

$\pi =$ equiprobable random action choices



actions



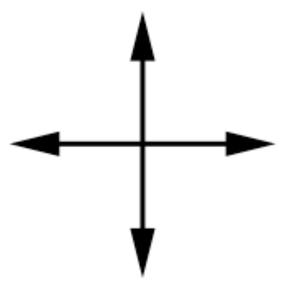
$R = -1$ on all transitions

$\gamma = 1$

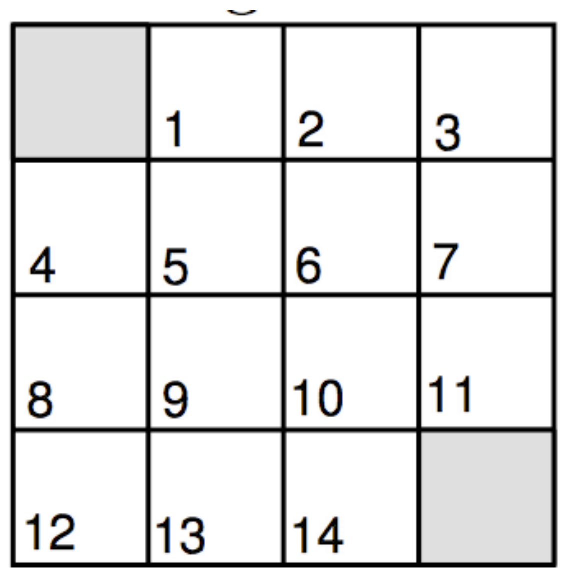
$$v_{k+1} = \sum_{a \in A} \pi(a \mid s)\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')\right)$$

Example: Small Gridworld

$\pi =$ equiprobable random action choices

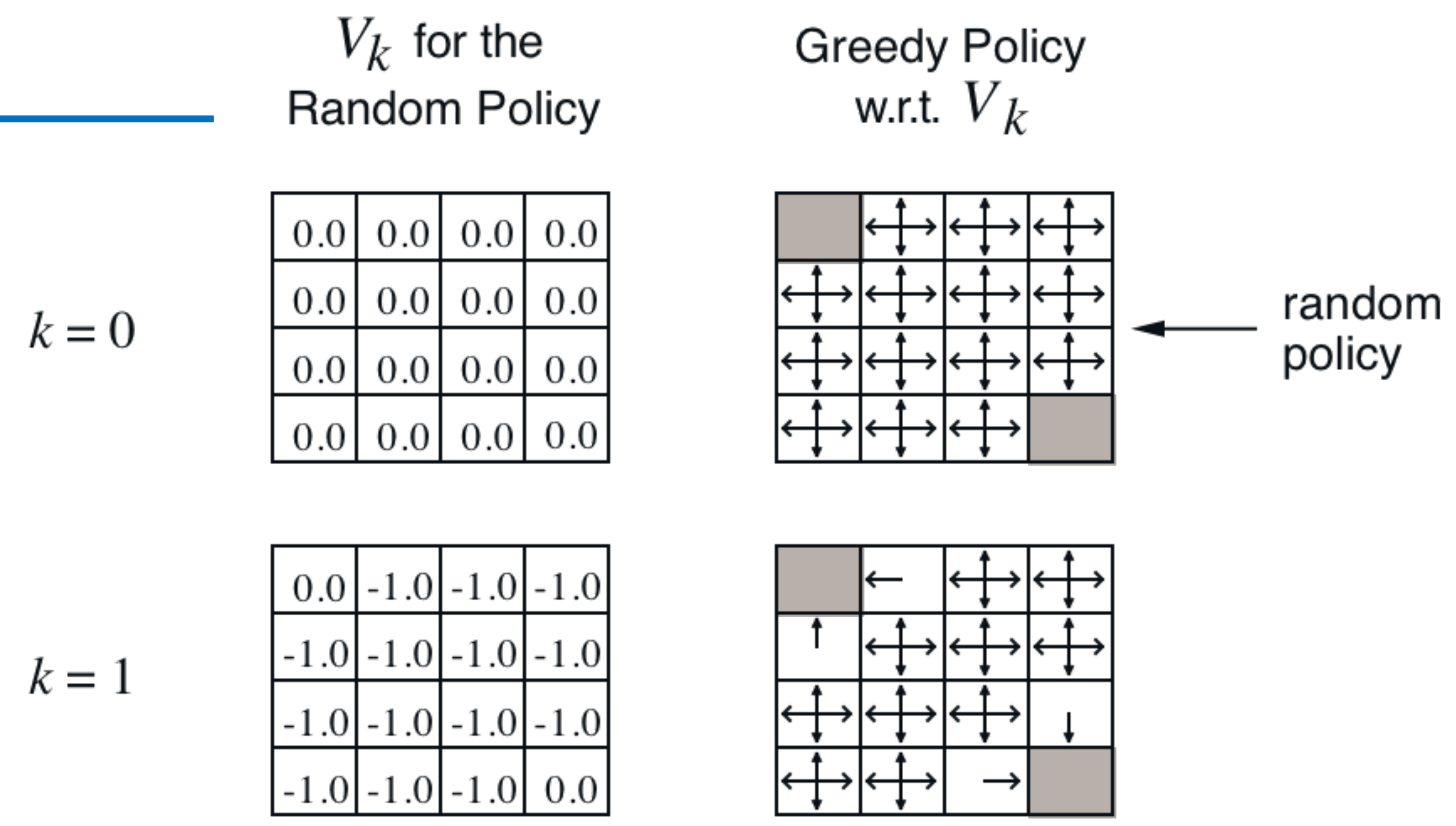


actions



$R = -1$
on all transitions

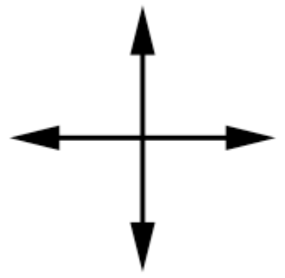
$\gamma = 1$



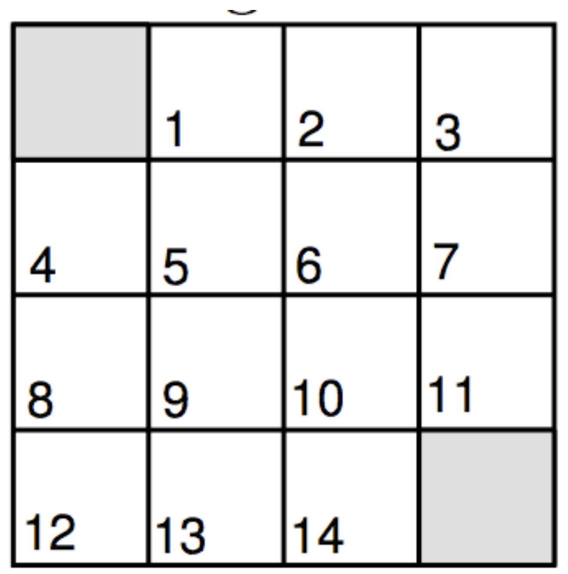
$$v_{k+1} = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

Example: Small Gridworld

$\pi =$ equiprobable random action choices



actions



$R = -1$
on all transitions

$\gamma = 1$

$$v_{k+1} = \sum_{a \in A} \pi(a \mid s)\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')\right)$$

V_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

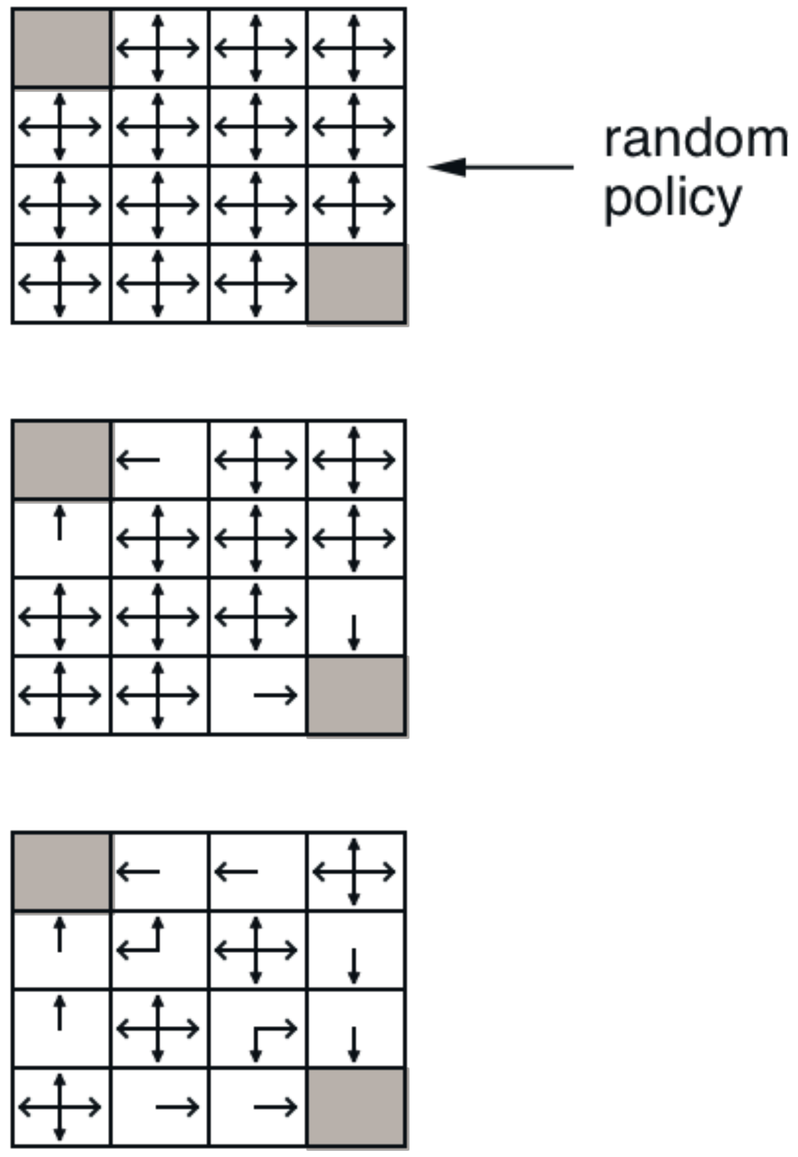
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

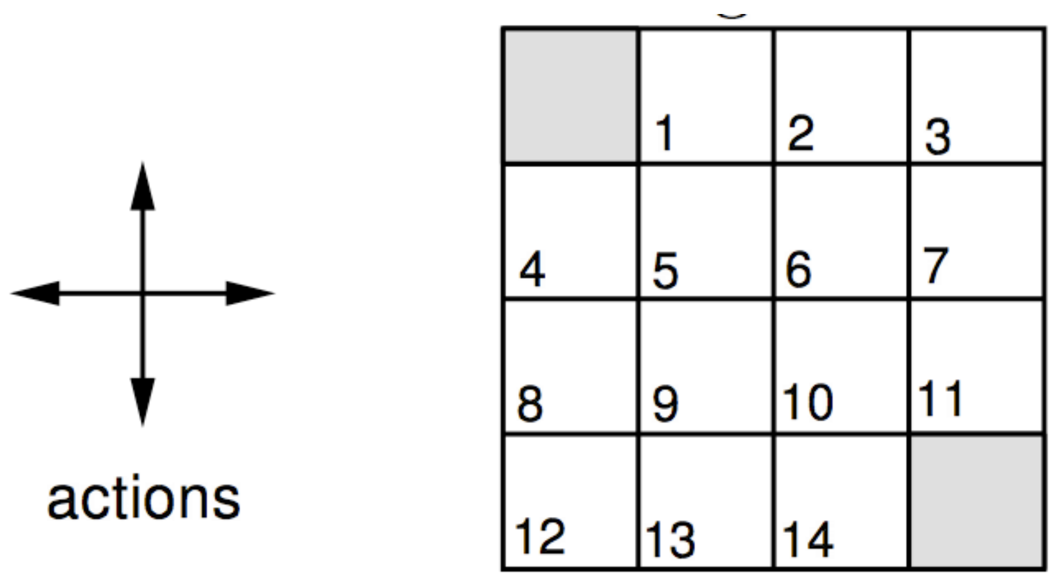
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

Greedy Policy w.r.t. V_k



Example: Small Gridworld

$\pi =$ equiprobable random action choices



$R = -1$
on all transitions

$\gamma = 1$

$$v_{k+1} = \sum_{a \in A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

V_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

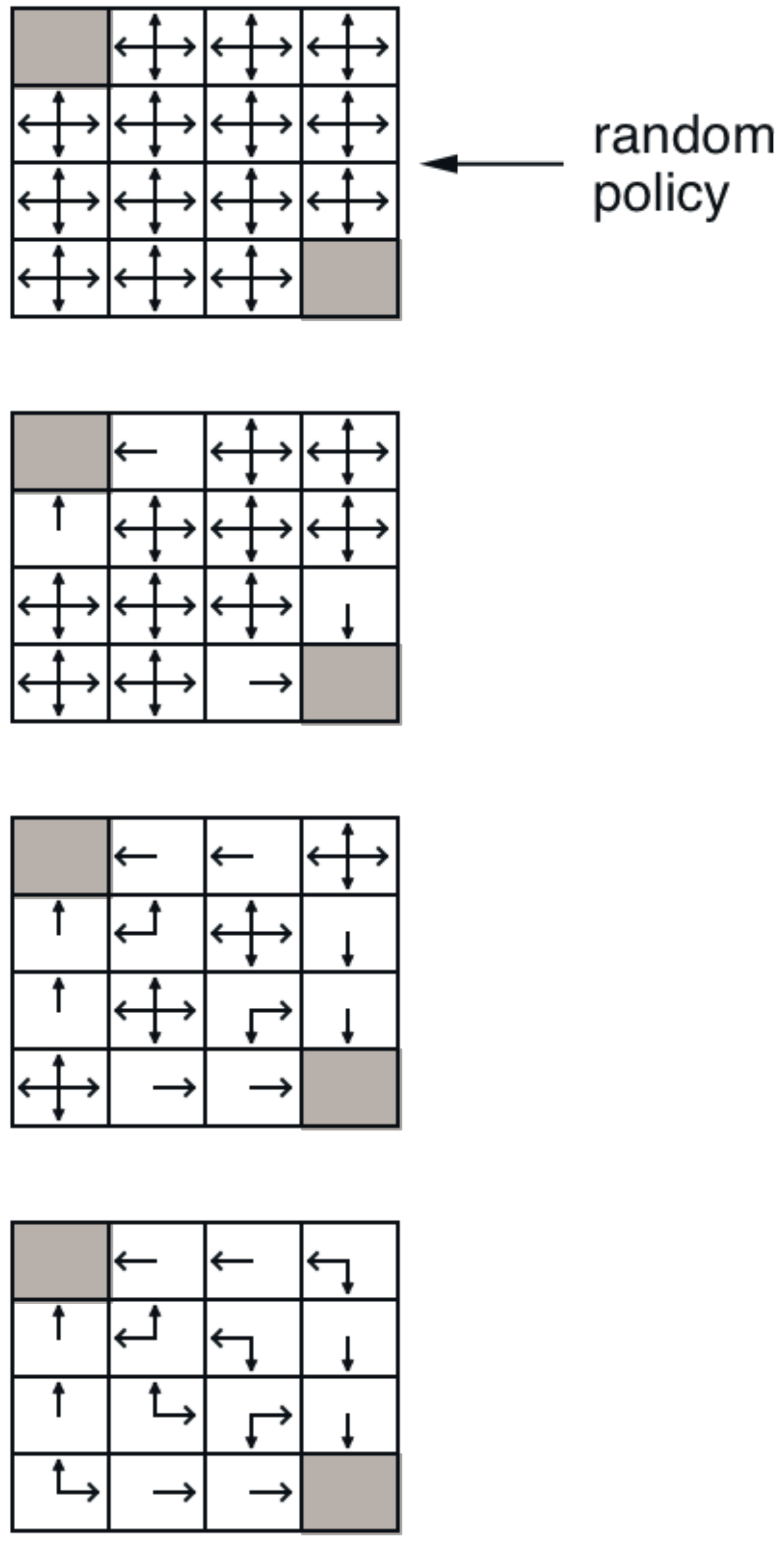
$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

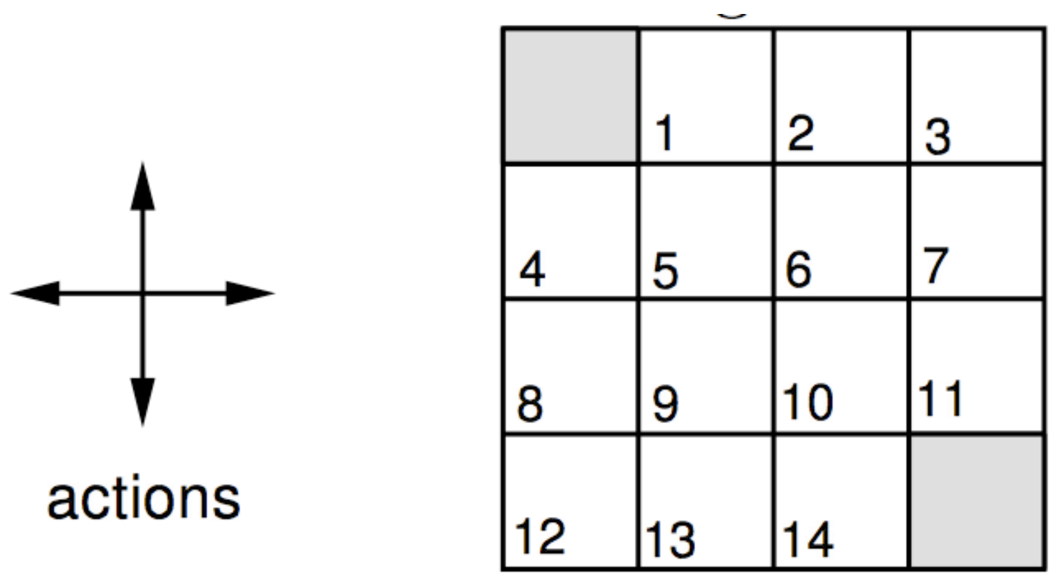
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

Greedy Policy w.r.t. V_k



Example: Small Gridworld

$\pi =$ equiprobable random action choices



$R = -1$
on all transitions

$\gamma = 1$

$$v_{k+1} = \sum_{a \in A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

V_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

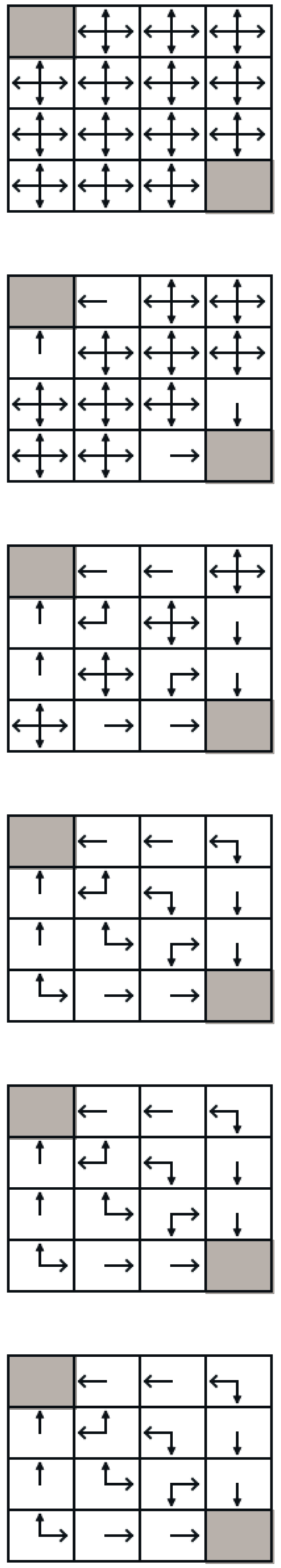
$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Greedy Policy w.r.t. V_k



random policy

optimal policy

Policy improvement theorem (How to improve the policy)

- Given the value function for *any policy* π, evaluate the policy:

$$q_\pi(s, a) \qquad \text{for all } s, a$$

- Improve the policy by acting greedily with respect to the value function:

$$\pi'(s) = \arg\max_a q_\pi(s, a)$$

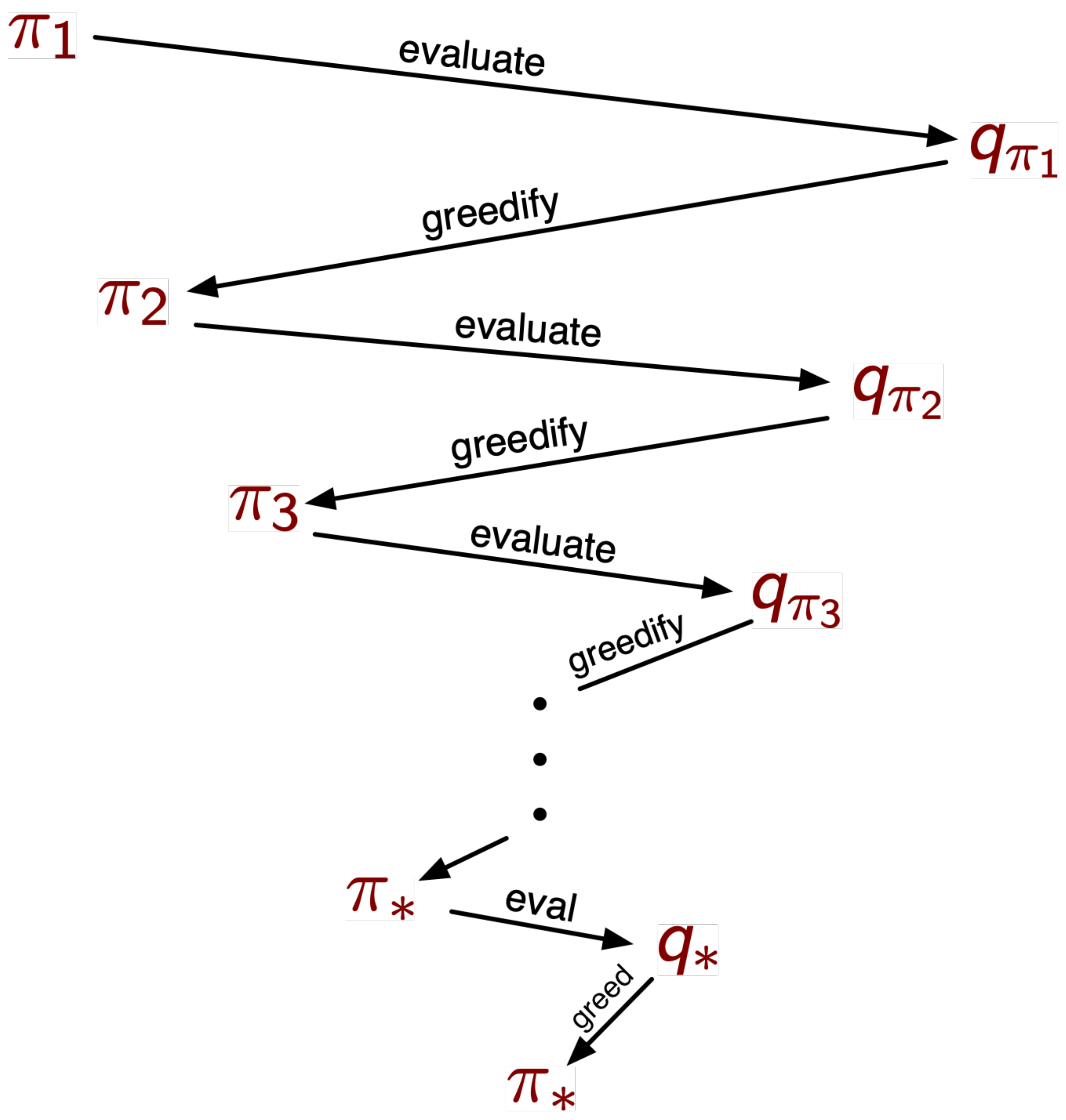
(π' is not unique)

- where better means:

$$q_{\pi'}(s, a) \geq q_\pi(s, a) \qquad \text{for all } s, a$$

- with equality only if both policies are optimal

The dance of policy and value (Policy Iteration)



- Policy evaluation: Estimate value function – Iterative policy evaluation
- Policy improvement: generate better policy by acting greedily – Greedy policy improvement

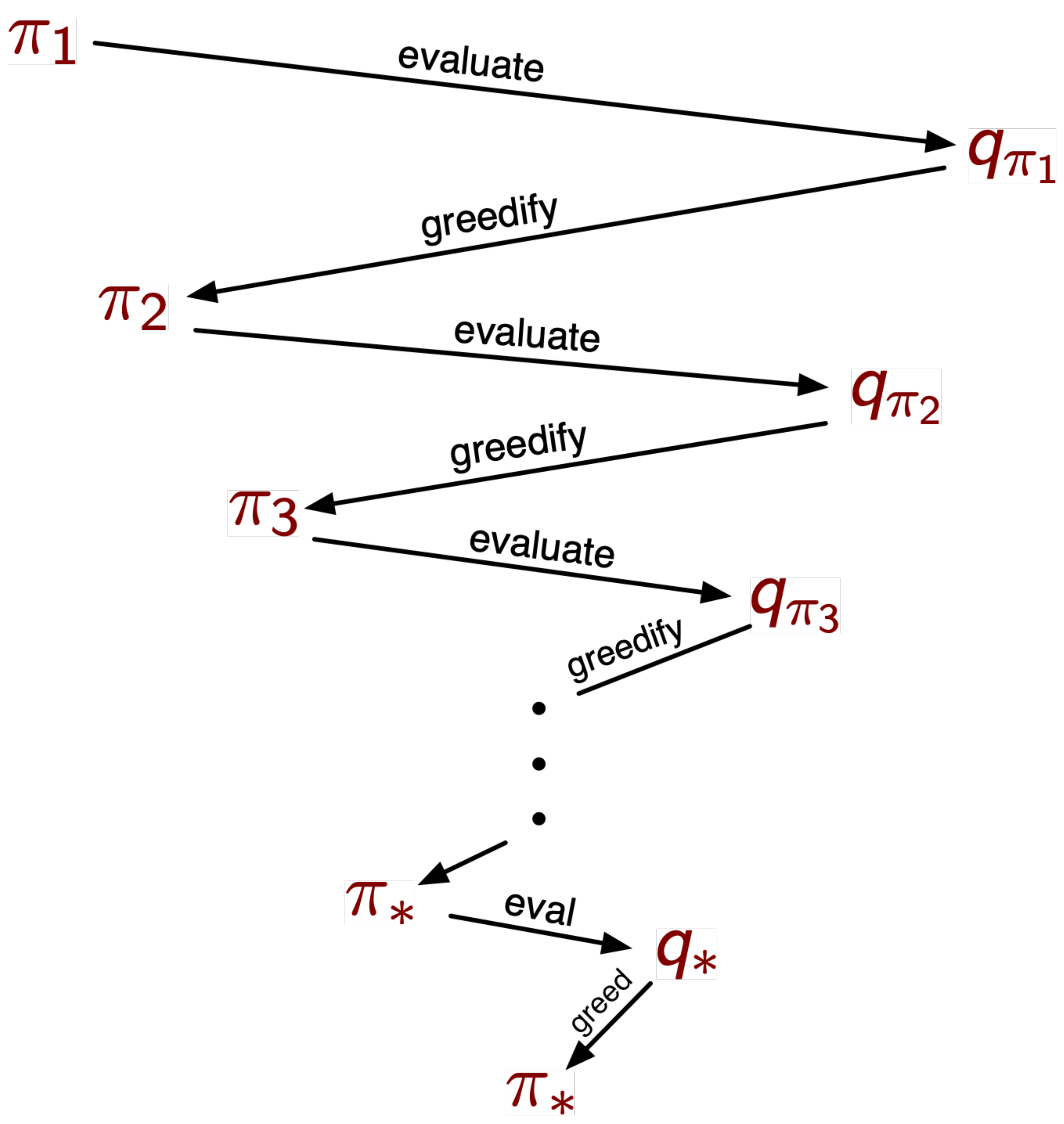
---

Each policy is *strictly better* than the previous, until *eventually both are optimal*

There are *no local optima*

The dance converges in a finite number of steps, usually very few

General Policy Iteration (GPI)



- Policy evaluation: Estimate value function – Any policy evaluation
- Policy improvement: generate better policy – Any policy improvement

Value Iteration

Recall the **full policy-evaluation backup**:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big] \qquad \forall s \in \mathcal{S}$$

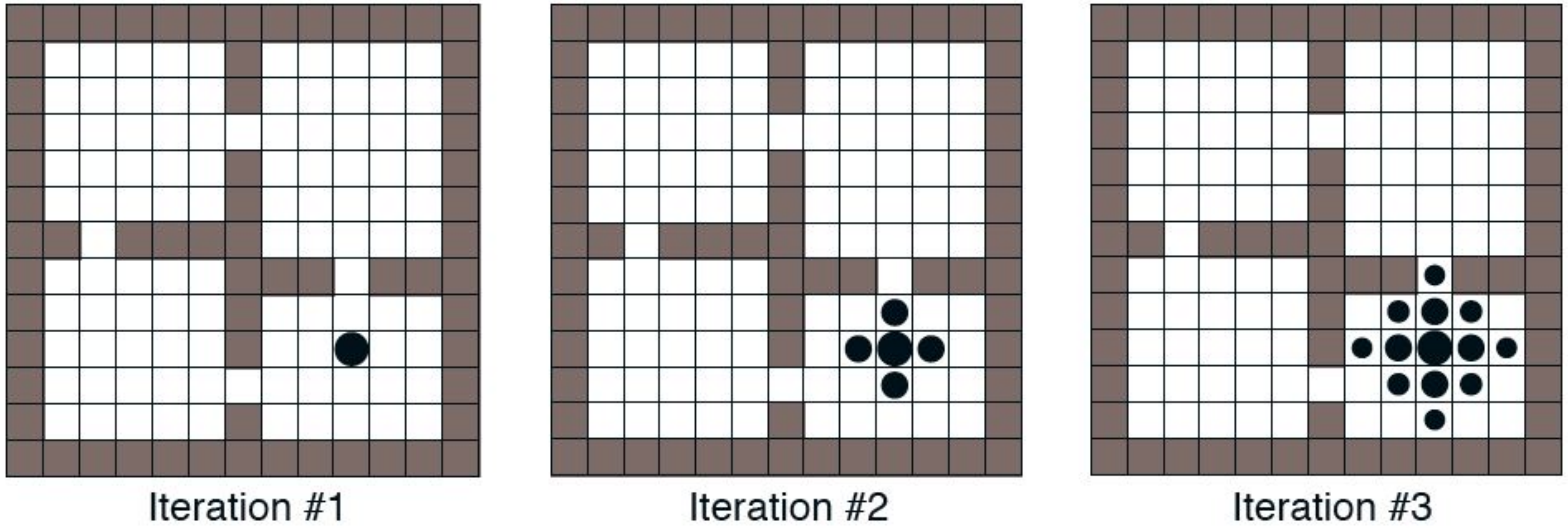
Here is the **full value-iteration backup**:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big] \qquad \forall s \in \mathcal{S}$$

Illustration: Rooms Example

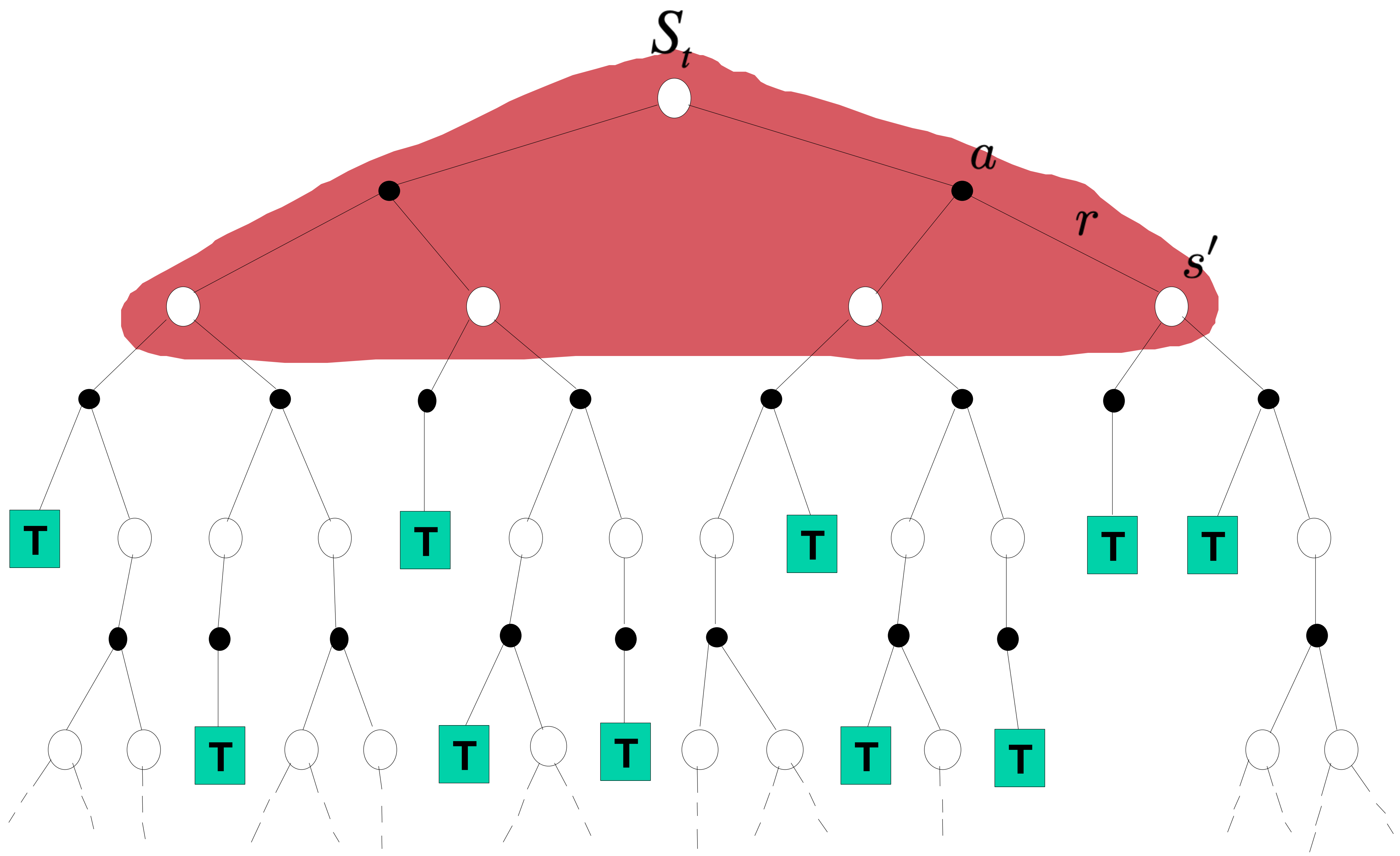
Four actions, fail 30% of the time

No rewards until the goal is reached, $\gamma = 0.9$.



cf. Dynamic Programming

$$V(S_t) \leftarrow E_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right] = \sum_a \pi(a|S_t) \sum_{s',r} p(s',r|S_t,a)[r + \gamma V(s')]$$



Curse of dimensionality

- Values are governed by nice recursive equations:

$$V_{k+1}(s) \leftarrow \max_{a \in A} \left(r^a_{ss'} + \gamma \sum_{s' \in S} p^a_{ss'} V_k(s') \right), \forall s \in S$$

- The number of states grows *exponentially* with the number of state variables (the dimensionality of the problem)

  E.g. in Go, there are 10^{170} states
- The *action set* may also be very large or continuous

  E.g. in Go, branching factor is ≈ 100 actions
- The solution may require *chaining many steps*

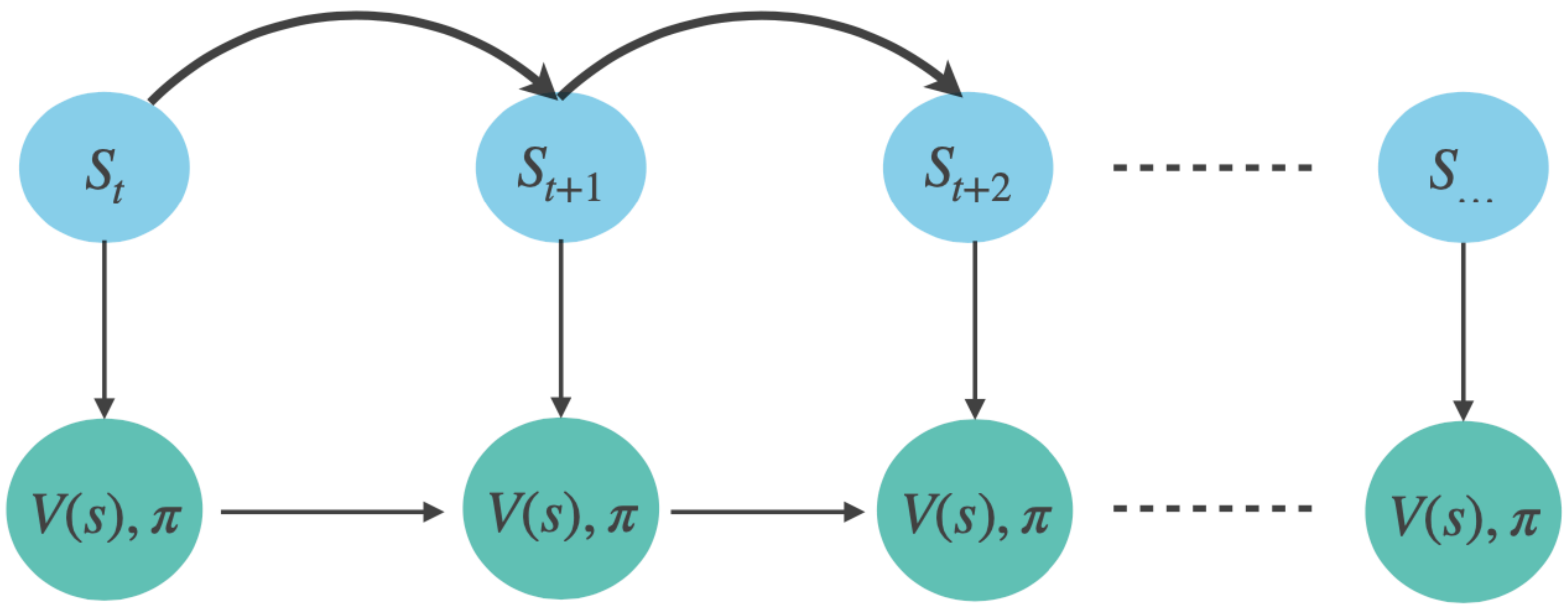
  E.g. in Go games take ≈ 200 actions

Key Challenges in RL

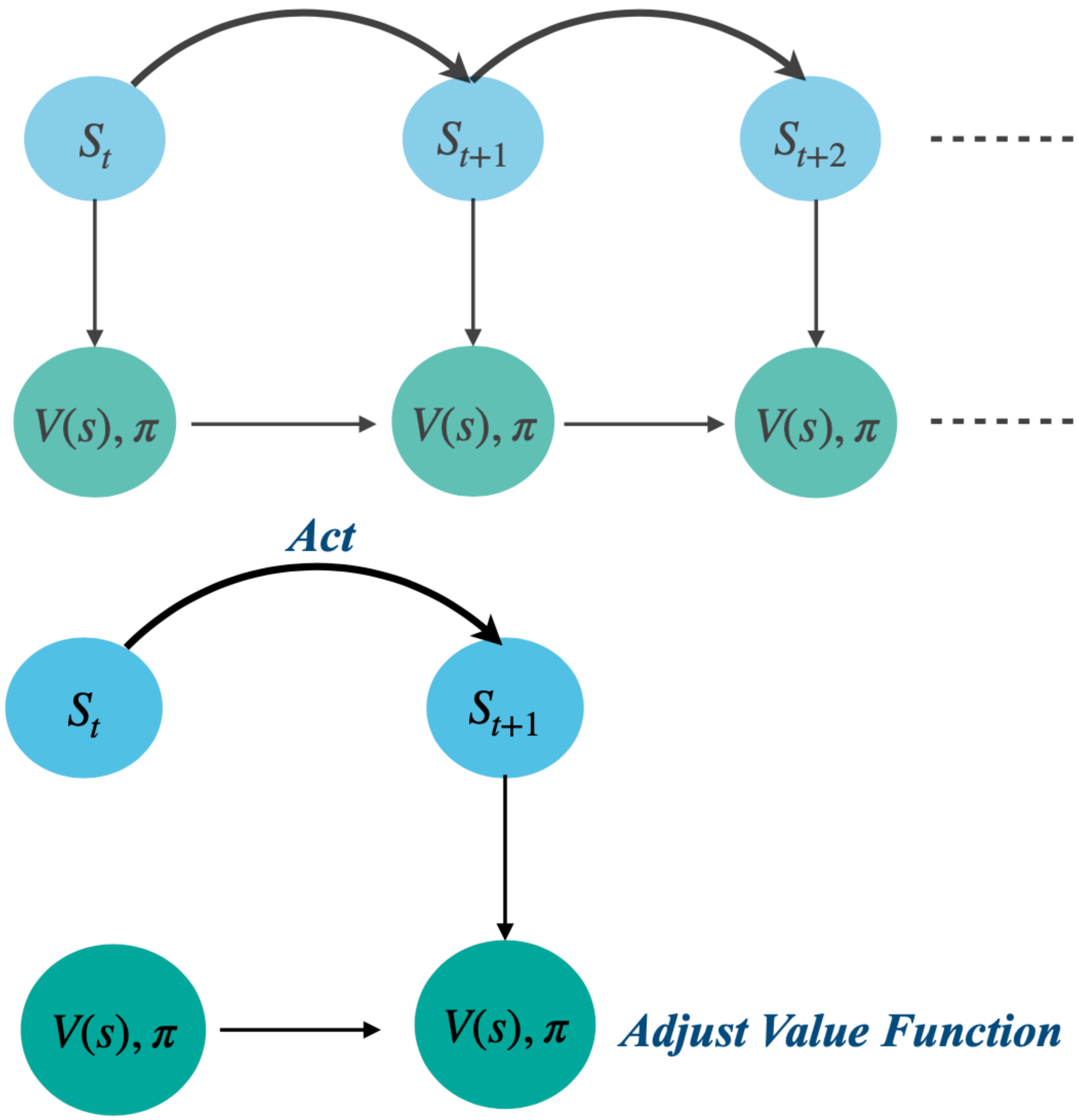
To solve large problems, we need to:

- ***Approximate the iterations*** (using sampling, cf. asynchronous dynamic programming, temporal-difference learning)
- ***Generalize*** the value function to unseen states using **function approximation**

Learning *online* using experience

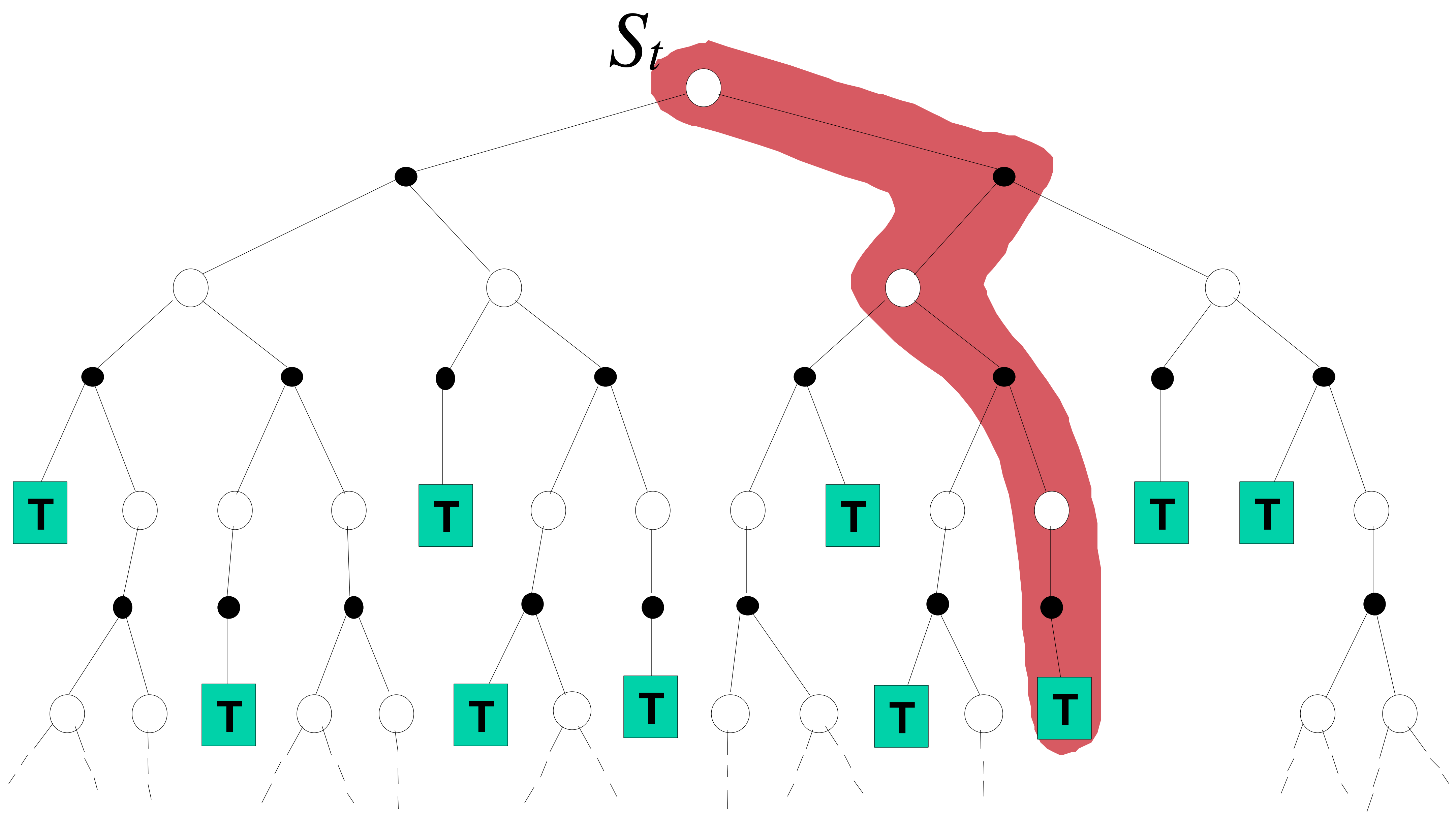


Learning *online* using experience



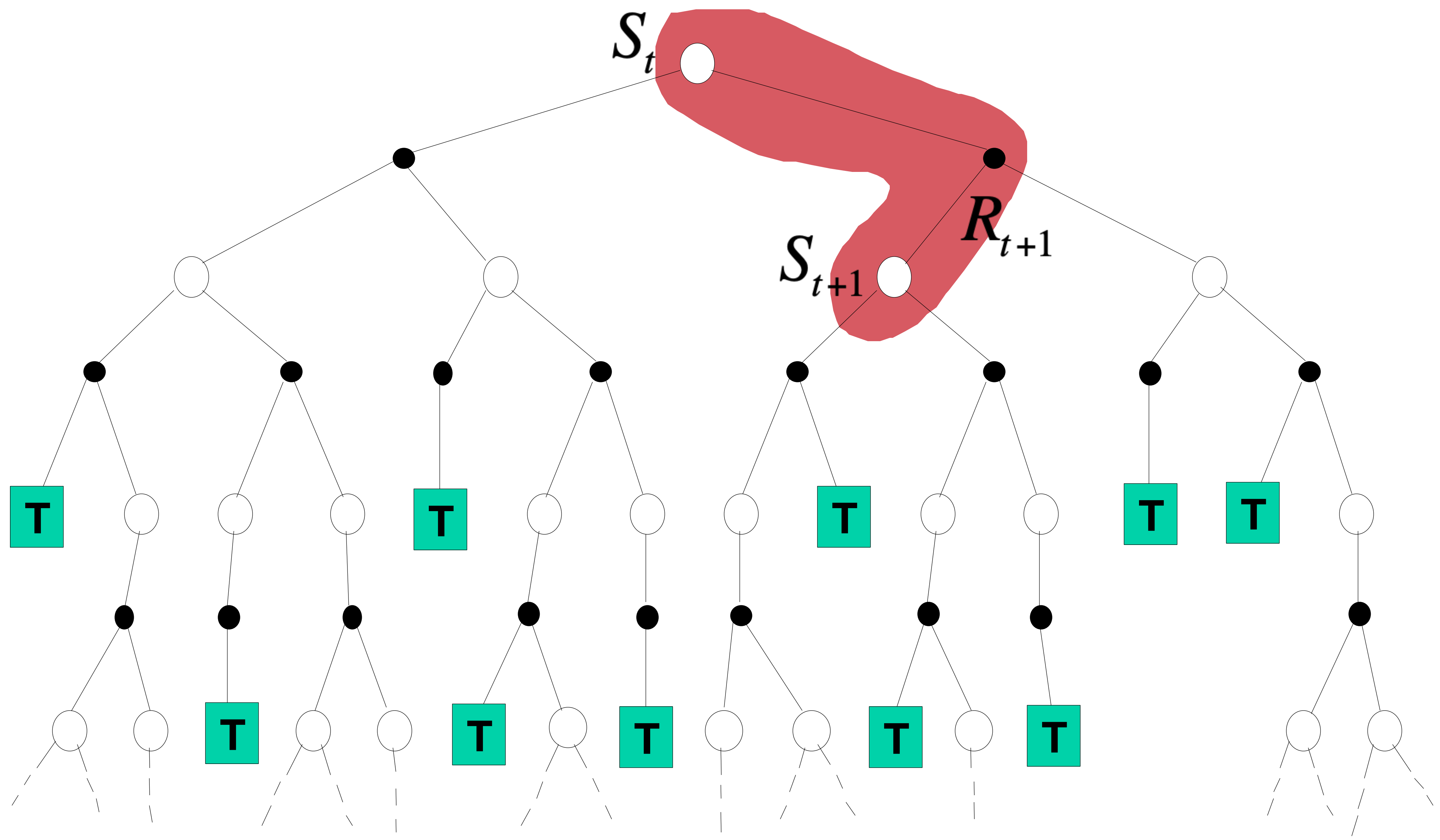
Recall: Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha\left[G_t - V(S_t)\right]$$



Temporal Difference (TD) Learning

$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$



Temporal Difference (TD) Learning

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

TD Prediction

Policy Evaluation (the prediction problem): for a given policy π, compute the state-value function v_π

Recall: Simple every-visit Monte Carlo method:

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[G_t - V(S_t) \Big]$$

target: the actual return after time *t*

TD Prediction

Policy Evaluation (the prediction problem): for a given policy π, compute the state-value function v_π

Recall: Simple every-visit Monte Carlo method:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

target: the actual return after time t

The simplest temporal-difference method TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[\underline{R_{t+1} + \gamma V(S_{t+1})} - V(S_t)\Big]$$

TD target: an estimate of the return

You are the Predictor

Suppose you observe the following 8 episodes:

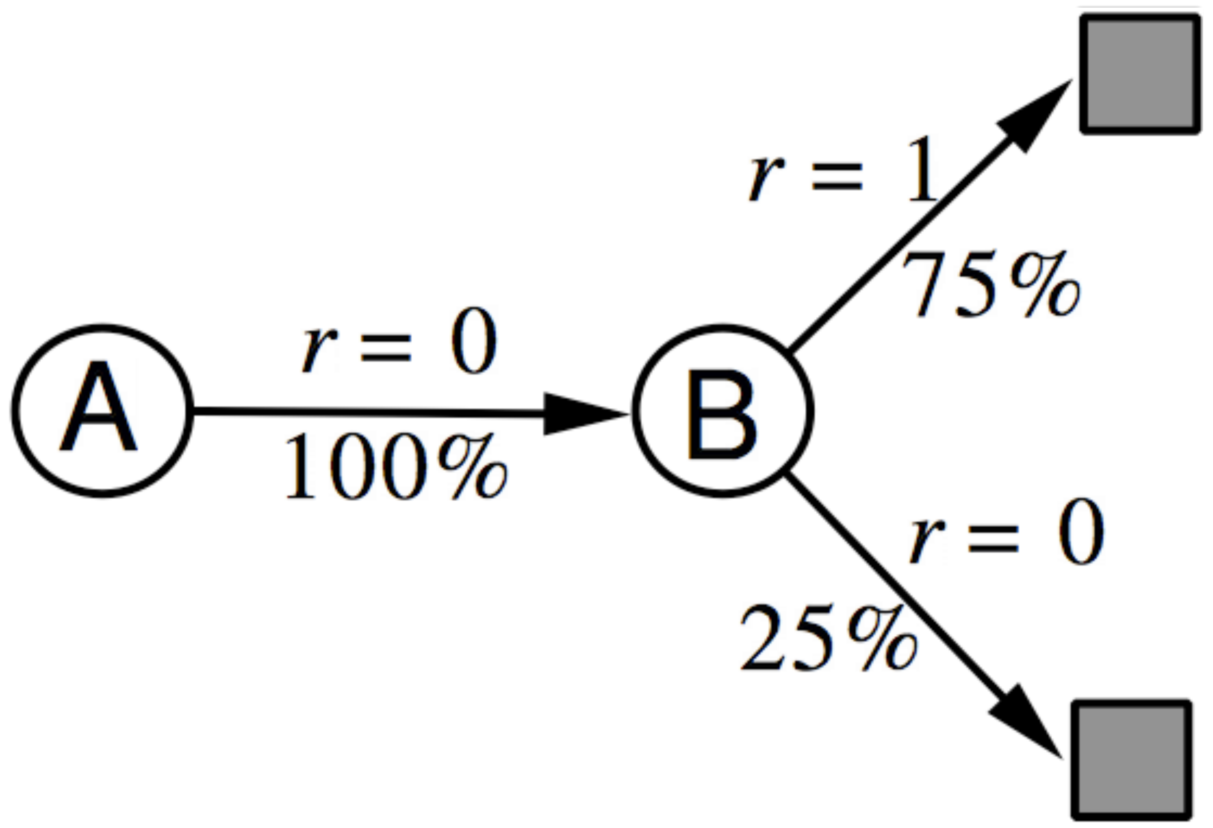
A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

$V(\mathrm{B})$?

$V(\mathrm{A})$?

Assume Markov states, no discounting ($\boldsymbol{\gamma} = 1$)

You are the Predictor



$V(\mathrm{A})$?

TD vs MC (I)

- TD can learn *before* knowing the final outcome

  It can learn online after every step

  MC must wait until the end of the episode before return is known

- TD can learn *without* the final outcome

  TD can learn from incomplete sequences as opposed to MC (needs complete sequences)

  TD works in continuing environments, MC only works for episodic (terminating) environments

TD vs MC (II)

- Bias/Variance trade off

  MC target i.e. the return is an unbiased estimate of the value function

  TD target is a biased estimate

  TD target is much lower variance than the return:

  - Return depends on *many* random actions, transitions, rewards
  - TD target depends on *one* random actions, transitions, rewards

- MC has high variance, zero bias
- TD has low variance, some bias

TD vs MC (III)

- Monte Carlo converges to solution with minimum mean-squared error (MSE)

  Best fit to observed returns $\sum_{k=1}^{K}\sum_{t=1}^{T_k}\left(G_t^k - V(s_t^k)\right)^2$

  In the AB example, V(A) = 0

- TD(0) converges to solution of max likelihood Markovian model

  Solution to MDP that best fits the data

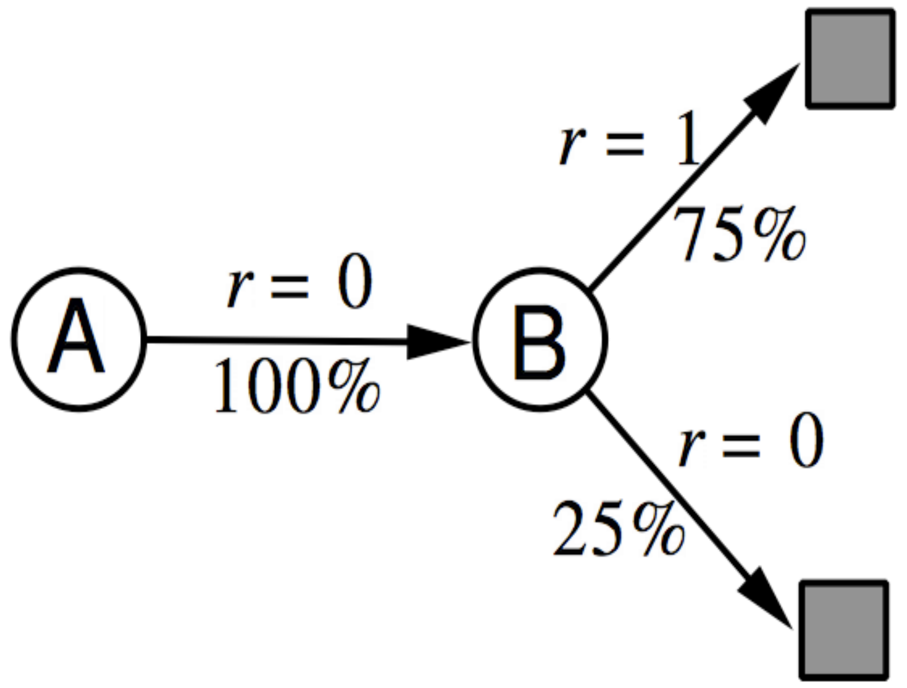
$$\hat{\mathcal{P}}^a_{s,s'} = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}\mathbf{1}(s_t^k, a_t^k, s_{t+1}^k = s, a, s')$$

$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}\mathbf{1}(s_t^k, a_t^k = s, a)r_t^k$$

  In the AB example, V(A) = 0.75

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

V(B)?
V(A)?

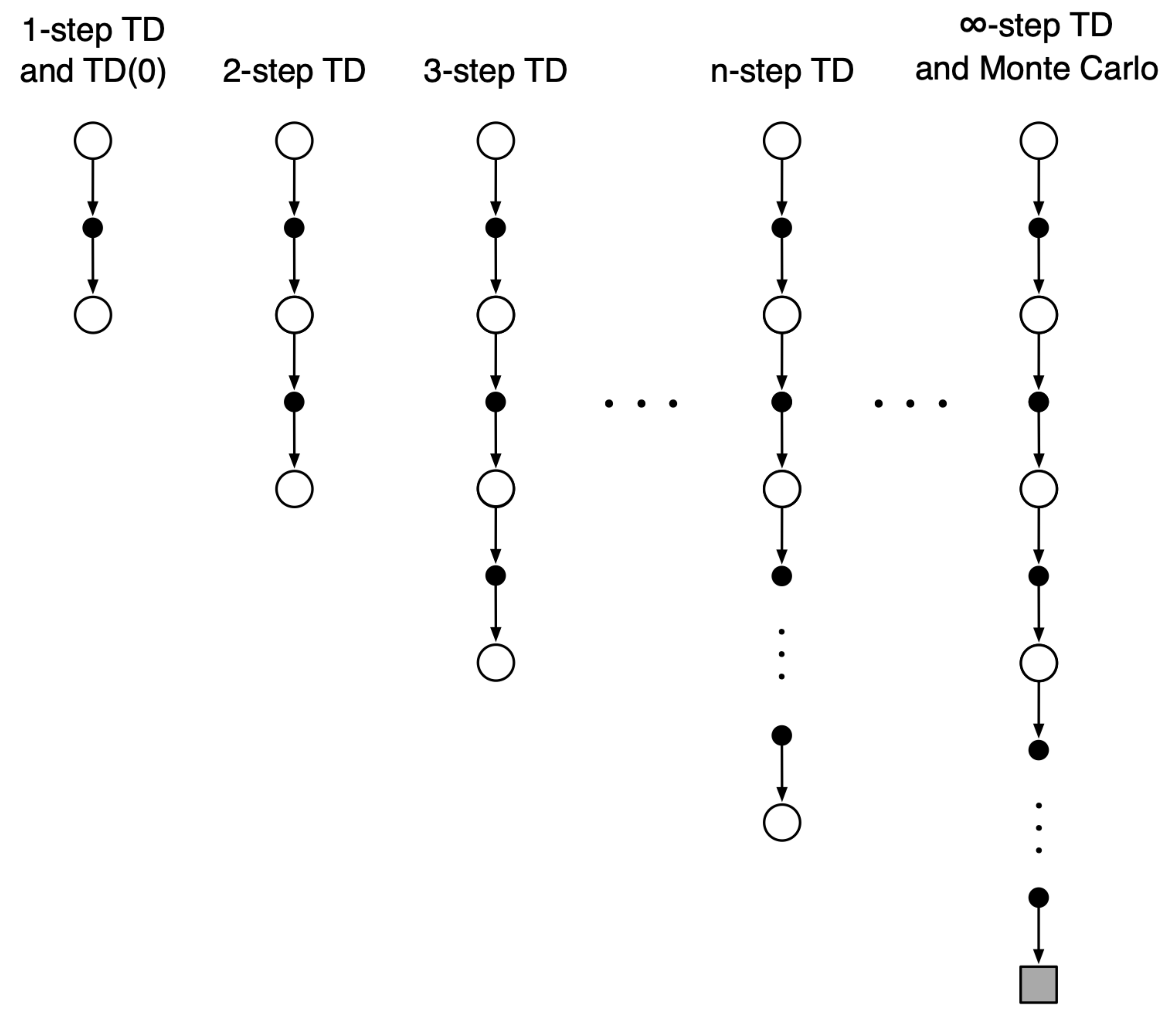


n-step TD Prediction

Idea: Look farther into the future when you do TD —
backup (1, 2, 3, …, n steps)

n-step TD Prediction

Idea: Look farther into the future when you do TD —
backup (1, 2, 3, …, n steps)



Mathematics of *n*-step TD Targets

- Monte Carlo:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

- TD:
  - Use V_t to estimate remaining return

$$G_t^{(1)} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

- *n*-step TD:
  - 2 step return:

$$G_t^{(2)} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_t(S_{t+2})$$

  - *n*-step return:

$$G_t^{(n)} \doteq G_t \text{ if } t + n \geq T)$$

Bootstrapping & Sampling

- Bootstrapping update involves an estimate

  MC does not bootstrap

  DP bootstraps

  TD bootstraps

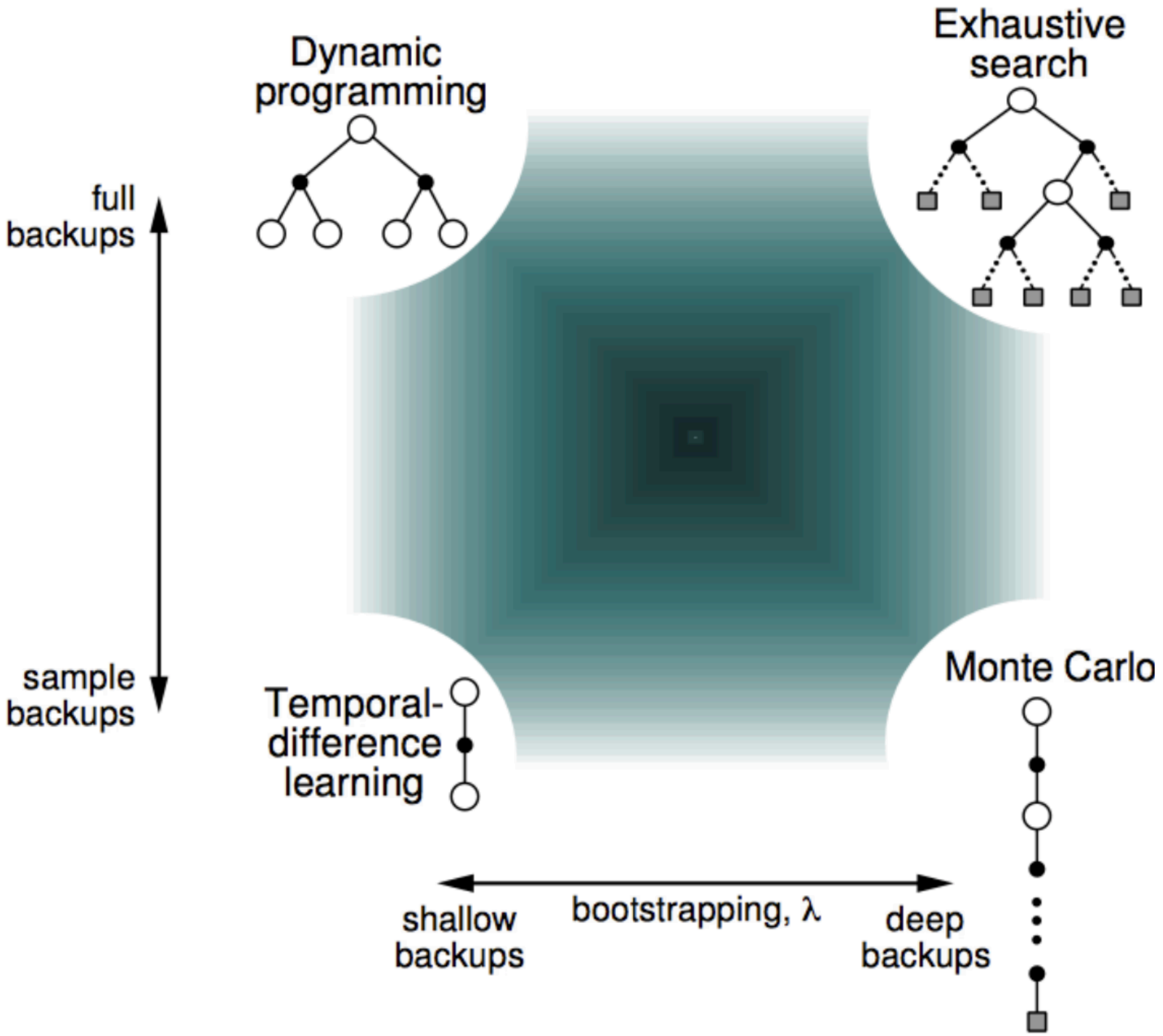
- Sampling update samples an expectation

  MC samples

  DP does not sample

  TD samples

Unified View of Reinforcement Learning



Thank You