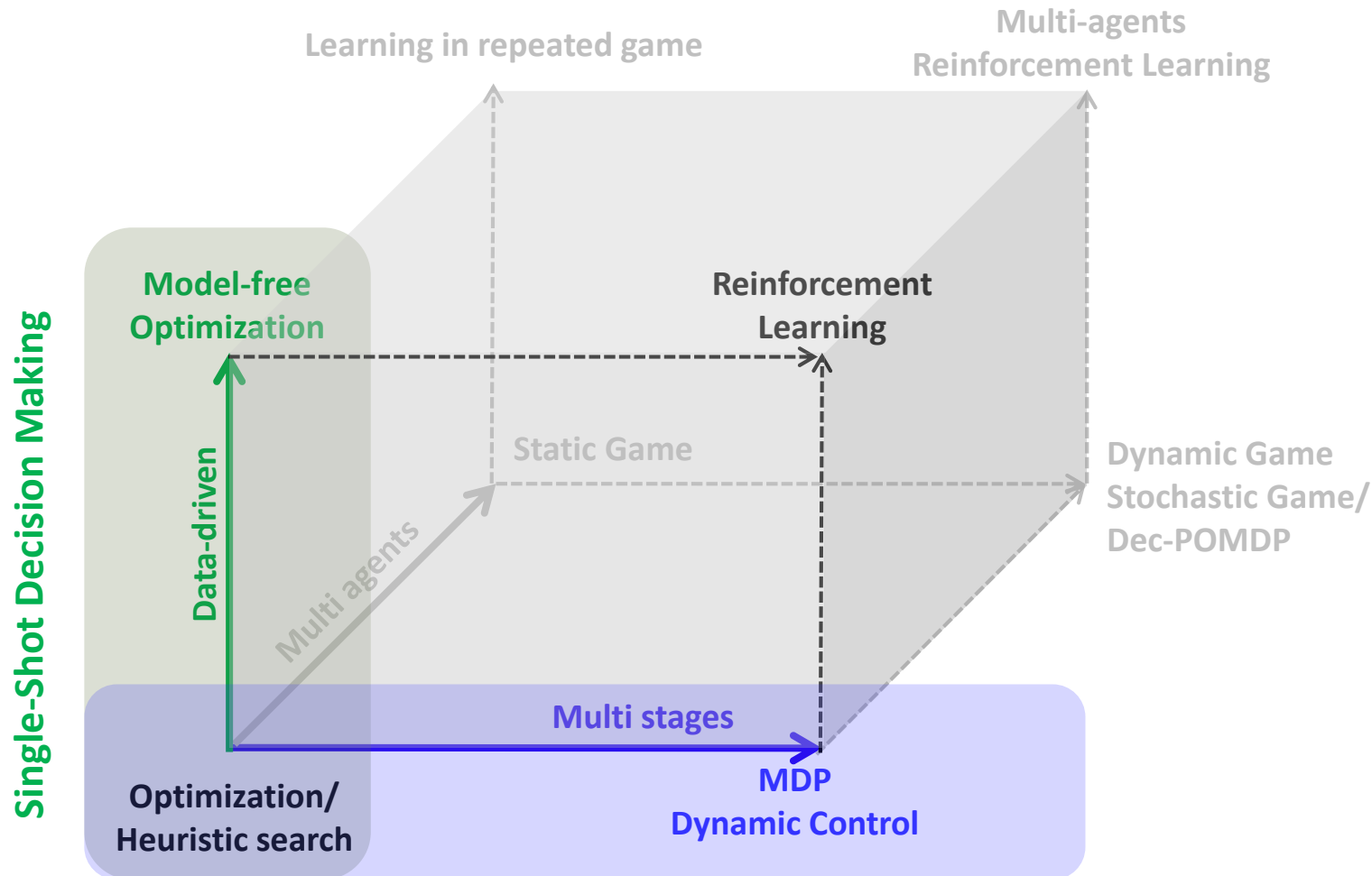


Lecture 20

Markov Decision Process and Dynamic Programming

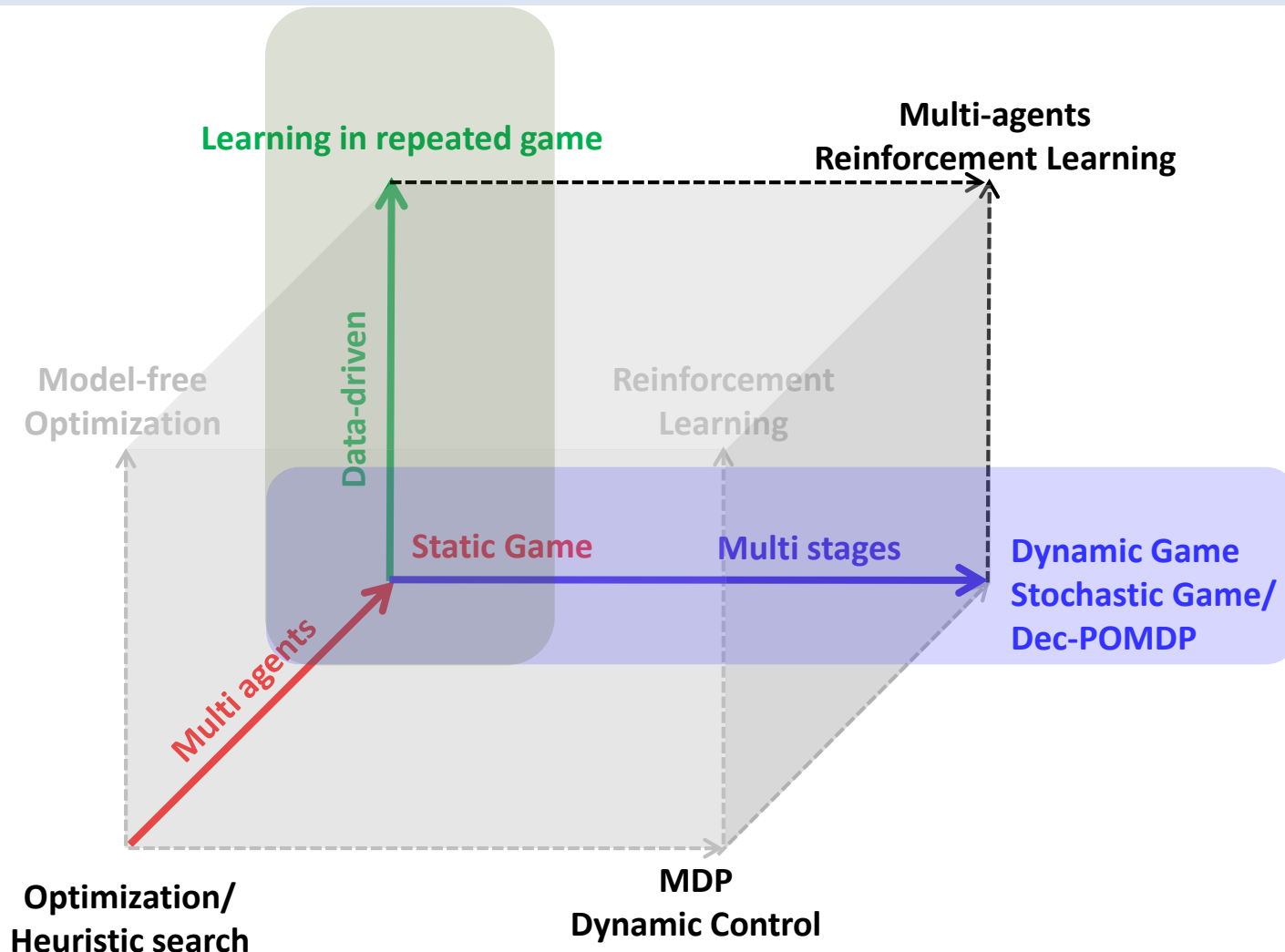
1. Definition of Markov Decision Process
2. Dynamic programming approach
 - Policy Evaluation
 - Policy Improvement
 - Policy iteration
 - Value Iteration

Problem Solving



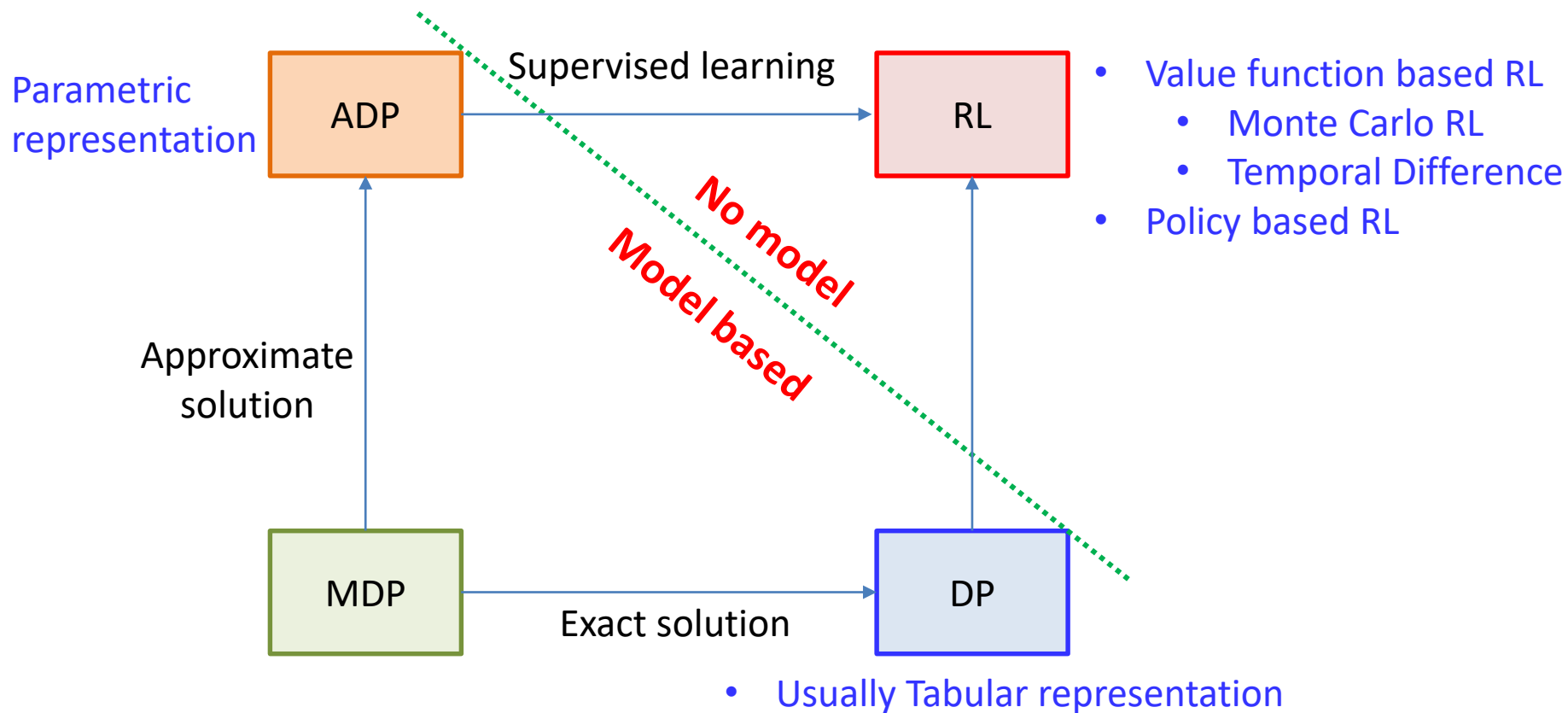
- Requires resonating about future *sequences* of actions and observations
- Requires resonating about actions of other players

Problem Solving



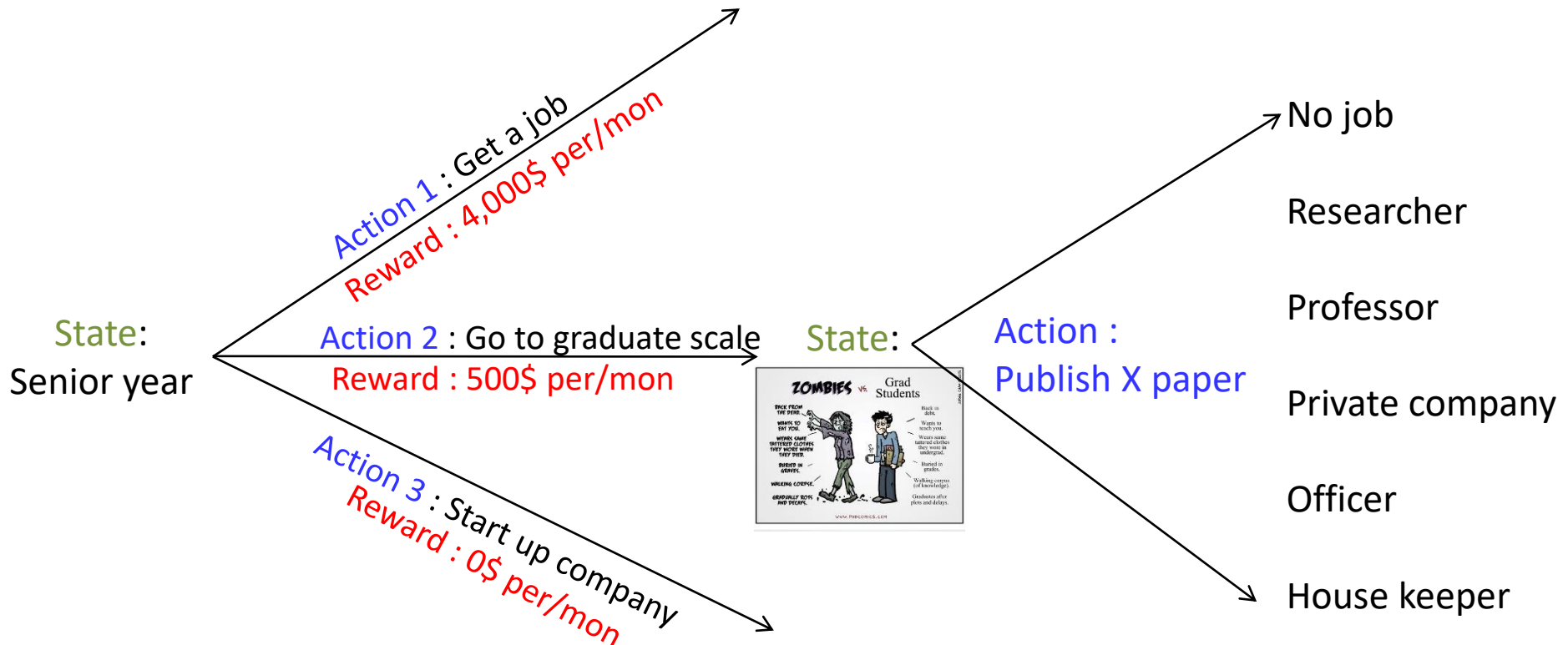
- Requires resonating about future *sequences* of actions and observations
- Requires resonating about actions of other players

Contents



Sequential Decision Making in Uncertainties

- Many important problems require the decision maker to make a series of decisions.
- It requires resonating about future *sequences* of actions and observations

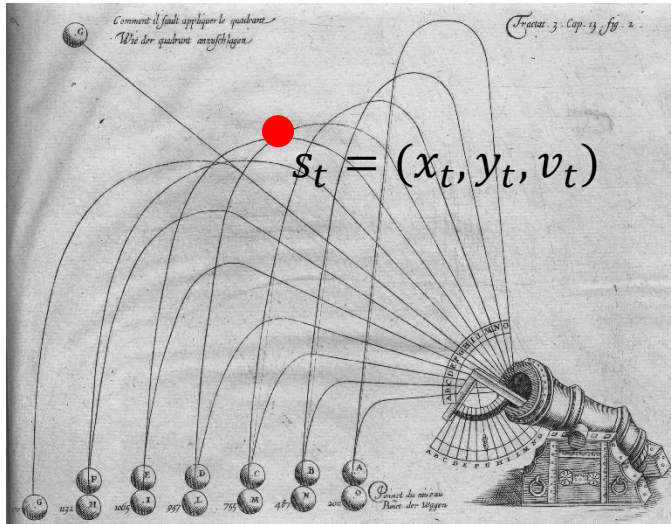


- taking an action might lead to any one of many possible states
- how we can even hope to act optimally in the face of randomness?

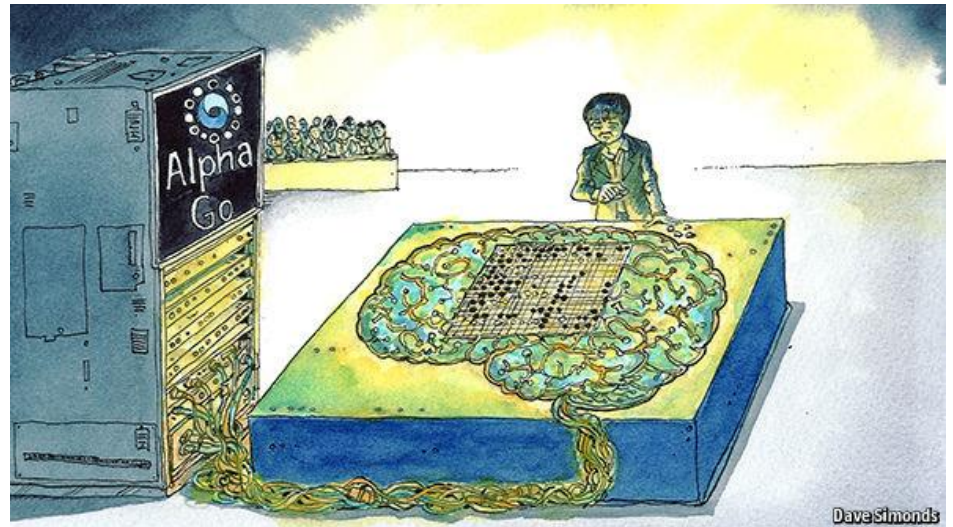
Definition of Markov Decision Process

State and Markov Property

- The state is constructed and maintained on the basis of **immediate sensations** together with **the previous state or some other memory of past sensations**
- Ideal state signal summarizes past sensations compactly, yet in such a way that all relevant information is retained
- A state signal that succeeds in retaining all relevant information is said to be **Markov**



s_t = the location and velocity of the flying canon



s_t = the configuration of the black and white stones

The Markov Property

- “Markov” generally means that given the present state, the future and the past are independent



Andrey Markov (1856-1922)

- For Markov decision processes, “Markov” means action outcomes depend only on the current state

$$\begin{aligned} P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \dots, S_0 = s_0, A_0 = a_0) \\ = P(S_{t+1} = s' | S_t = s_t, A_t = a_t) \end{aligned}$$

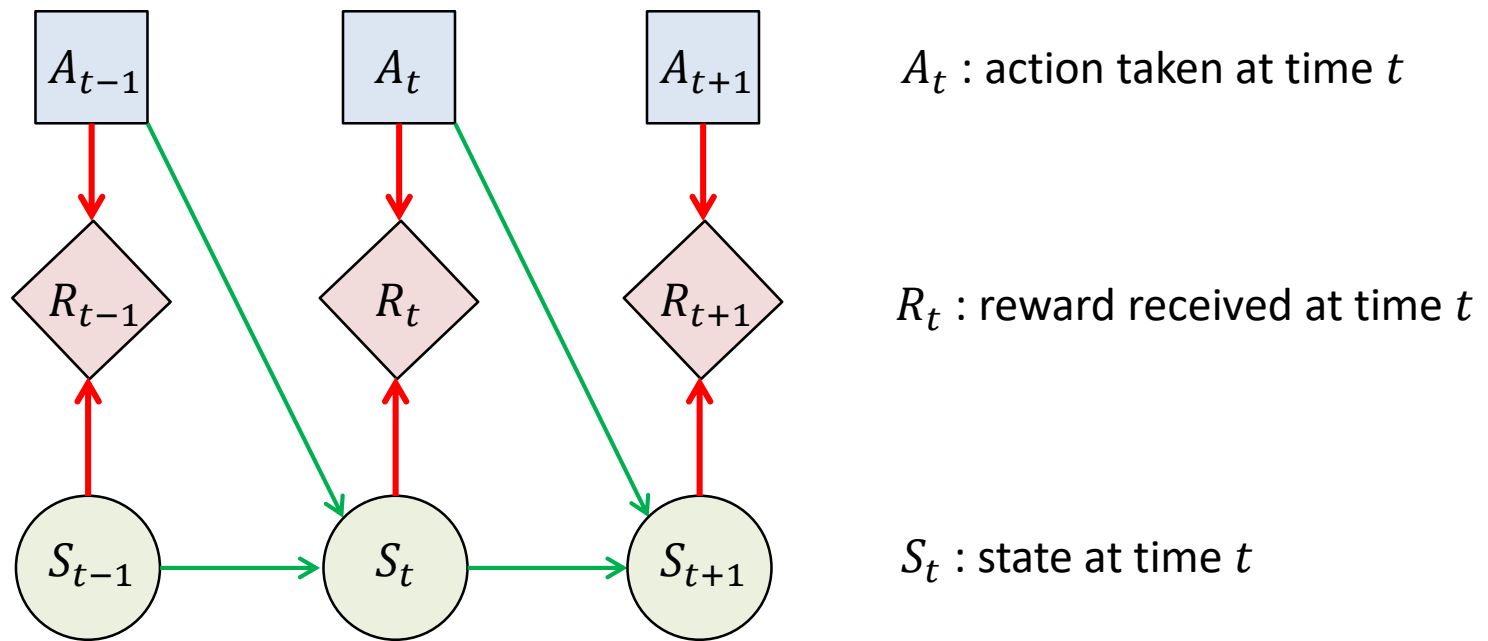
- The best policy for choosing actions as a function of a Markov state is just as good as the best policy for choosing actions as a function of complete history

$$\pi^*(s_t, s_{t-1}, \dots, s_0) = \pi^*(s_t)$$

Note:

- As states become more Markovian, the better performance in MDP solution
- It is useful to think of the state at each time step as an approximation to a Markov value

Markov Decision Processes



- **Transition probability** $T_t(s_t, a_t, s_{t+1}) = P(S_{t+1} = s_{t+1} | S_t = s_t, A_t = a_t) = P(s_{t+1} | s_t, a_t)$
 - ✓ represents the probability of transitioning from state s_t to s_{t+1} after executing action s_t at time t (**Markov assumption**)
- **Reward function:** $r_t = R_t(s_t, a_t)$: represents the probability of receiving reward r_t when executing action a_t from state s_t at time t

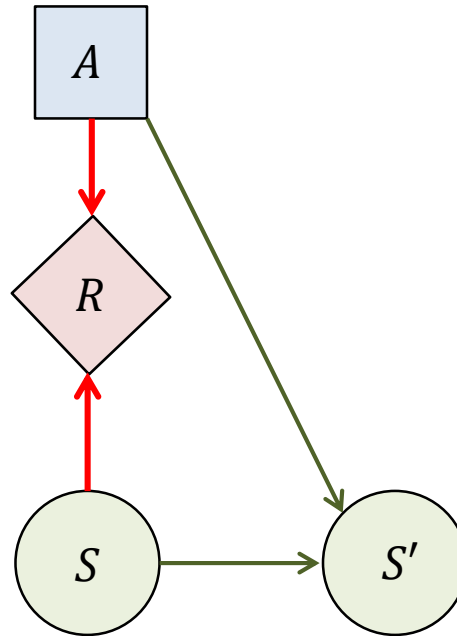
The Agent-Environment Interface

- The time steps $t = 0, 1, \dots$ need not refer to fixed interval of real time:
 - ✓ they can refer to arbitrary successive stages of decision-making and action
- The **actions** can be
 - ✓ Low-level controls, such as the voltages applied to motors of a robot arm
 - ✓ High-level decisions, such as whether or not to go graduate school
- The **states** can take a wide variety of forms
 - ✓ Can be completely determined by low-level sensations, such as sensor readings
 - ✓ Can be high-level and abstract, such as image or mental status
- The **reward** is a consequence of taking an action given a state
 - ✓ Can be specified according to the target tasks
 - ✓ Maximizing reward should result in achieving the goals of a task

In summary,

Actions can be any decisions we want to learn how to make to affect rewards, and the states can be anything we can know that might be useful in making them

(Stationary) Markov Decision Processes

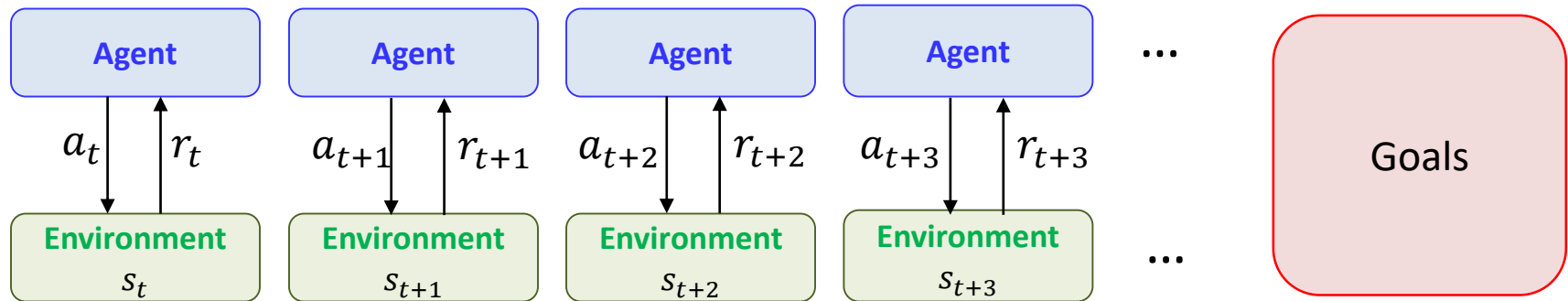


Stationary Markov Decision Process (MDP)

→ transition and reward models are **stationary**

- Transition probability $T(s_t, a_t, s_{t+1})$ are the same for all time step t
- Reward function: $r_t = R(s_t, a_t)$ are the same for all time step t

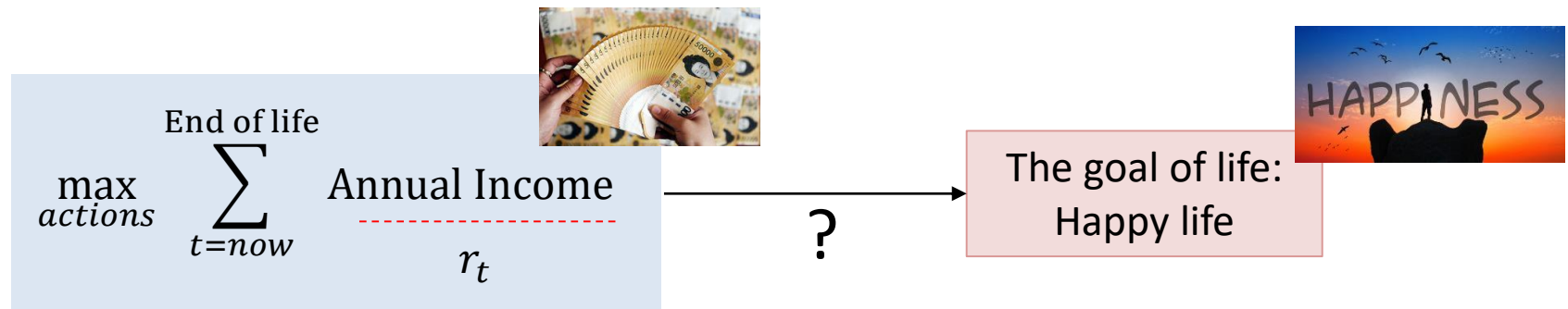
Goals and Rewards



- The agent's goal is to maximize the total amount of reward (cumulative reward) it receives.

$$\max_{a_t, a_{t+1}, \dots} \sum_k r_{t+k}$$

- Rewards we setup truly indicate what we want accomplished
- The reward signal is your way of communicating to the agent **what you want it to achieve**, not how you want it achieved



Utility (returns)

How to formally define the **accumulated reward** in the long run?

- Denote reward : $r_t = R(S_t, A_t)$
- In the episodic tasks, the simplest utility is defined as

$$U_t = r_t + r_{t+1} + r_{t+2} + \dots + r_T = \sum_{k=0}^T r_{t+k}$$

- ✓ T is a final time step associated with the terminal time step T
- ✓ Examples include, maze, go, chess, etc.

- In the continuing tasks, the discounted utility is defined as

$$U_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

- ✓ γ is a parameter, $0 \leq \gamma \leq 1$, called discount rate
- ✓ Examples include, maze game, Go, Chess, etc.
- ✓ $\gamma = 0$ (myopic): concern only with maximizing immediate rewards
- ✓ $\gamma = 1$ (farsighted): the objective takes future rewards take into account more strongly

- A policy π gives an action $a \in \mathcal{A}$ for each state $s \in \mathcal{S}$:

$$\pi: \mathcal{S} \rightarrow \mathcal{A}$$



- An optimal policy** π^* is one that maximizes expected utility if followed:

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}[U^{\pi}(s)] \text{ for all } s$$

$$\text{where } \mathbb{E}[U^{\pi}(s)] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots]$$

- An optimal policy** can be either deterministic or stochastic

deterministic

$$a^* = \pi^*(s)$$

Stochastic

$$p(a|s) = \pi^*(s, a)$$

Take action a^* with a probability one

Take action a with a probability $p(a|s)$

We will exclusively consider deterministic policy

Summary: Markov Decision Processes

Finite Markov Decision Process (MDP) :The state and action space are finite

An MDP is defined by:

- A set of states $s \in \mathcal{S}$
- A set of actions $a \in \mathcal{A}$
- A transition function $T(s, a, s') = P(S_{t+1} = s' | S_t = s, A_t = a) = P(s' | s, a)$
 - ✓ Probability that a from s leads to s' when taking action a
 - ✓ Also called the model or the dynamics
- A reward function $R(s, a, s')$
 - ✓ $r_t = R(s_t, a_t, s_{t+1})$ or $r_{t+1} = R(s_t, a_t)$
 - ✓ If stochastic, $R(s, a, s') = \mathbb{E}[r_t + r_{t+1}, \dots | S_t = s, A_t = a, S_{t+1} = s']$
- A start state $s_0 \in \mathcal{S}$
- A terminal state $s_T \in \mathcal{S}^+$ (for episodic tasks)

Goal of MDP is to find the optimal policy π^* that maps the current state to the best action

$$a^* = \pi^*(s)$$

Dynamic Programming Approach

Value Function & Q function

Value function (**state value function for π**)

“How good it is for the agent to be in a given state”

$V^\pi(s)$: The expected utility received by following policy π from state s

$$V^\pi(s) = \mathbb{E}_\pi(U_t | S_t = s) = \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid S_t = s\right)$$

\mathbb{E}_π : not expectation over policy π but all stochastic state transitions associated with π

Q-function (**action-value function for π**)

“How good it is for the agent to perform a given action in a given state”

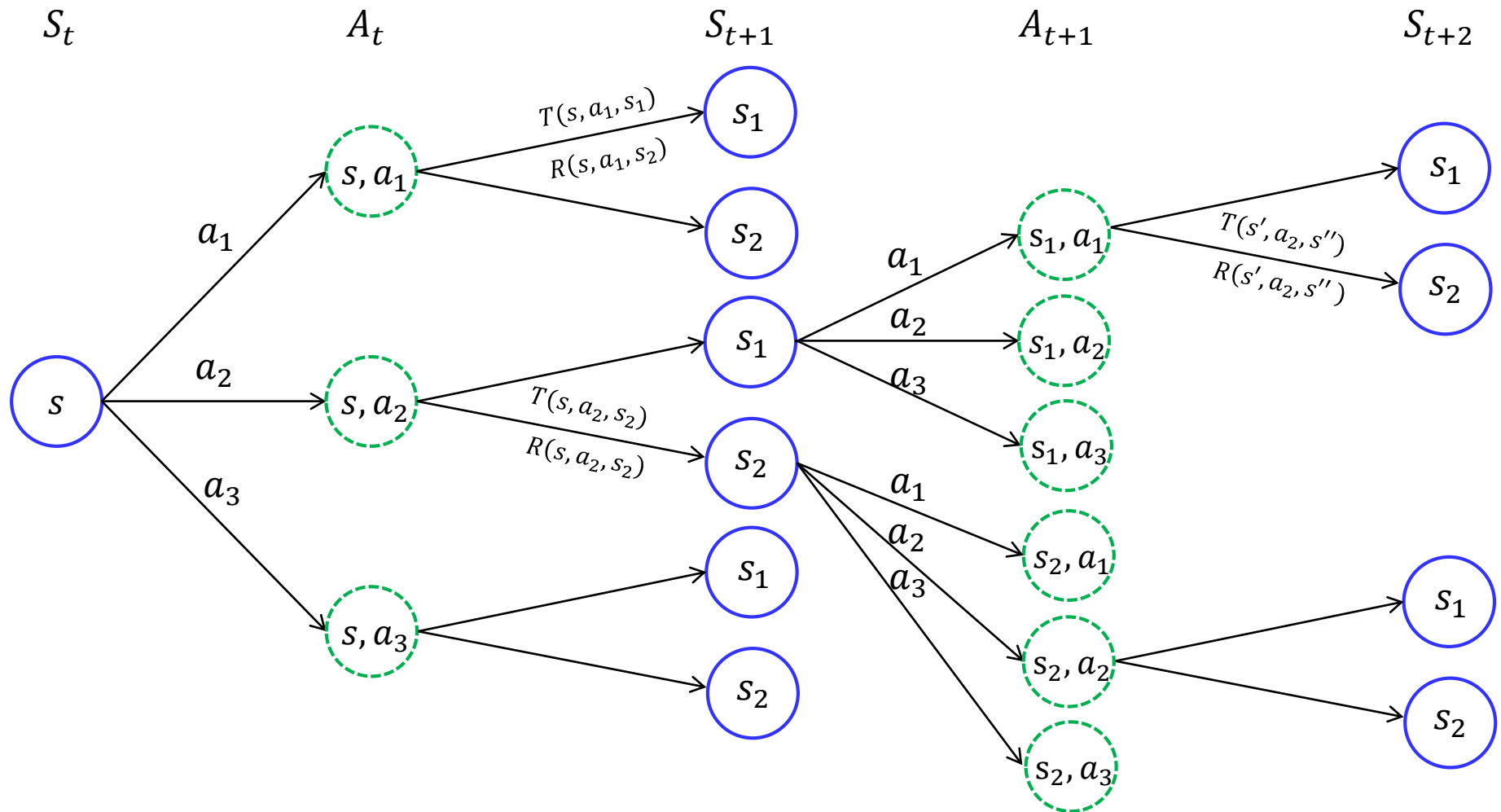
$Q^\pi(s, a)$: The expected utility of taking action a from state s , and then following policy π

$$Q^\pi(s, a) = \mathbb{E}_\pi(U_t | S_t = s, A_t = a) = \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid S_t = s, A_t = a\right)$$

Because the agent can expect to receive in the future depend on what actions it will take
→ Value and Q functions are defined with respect to a particular policy mapping state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$

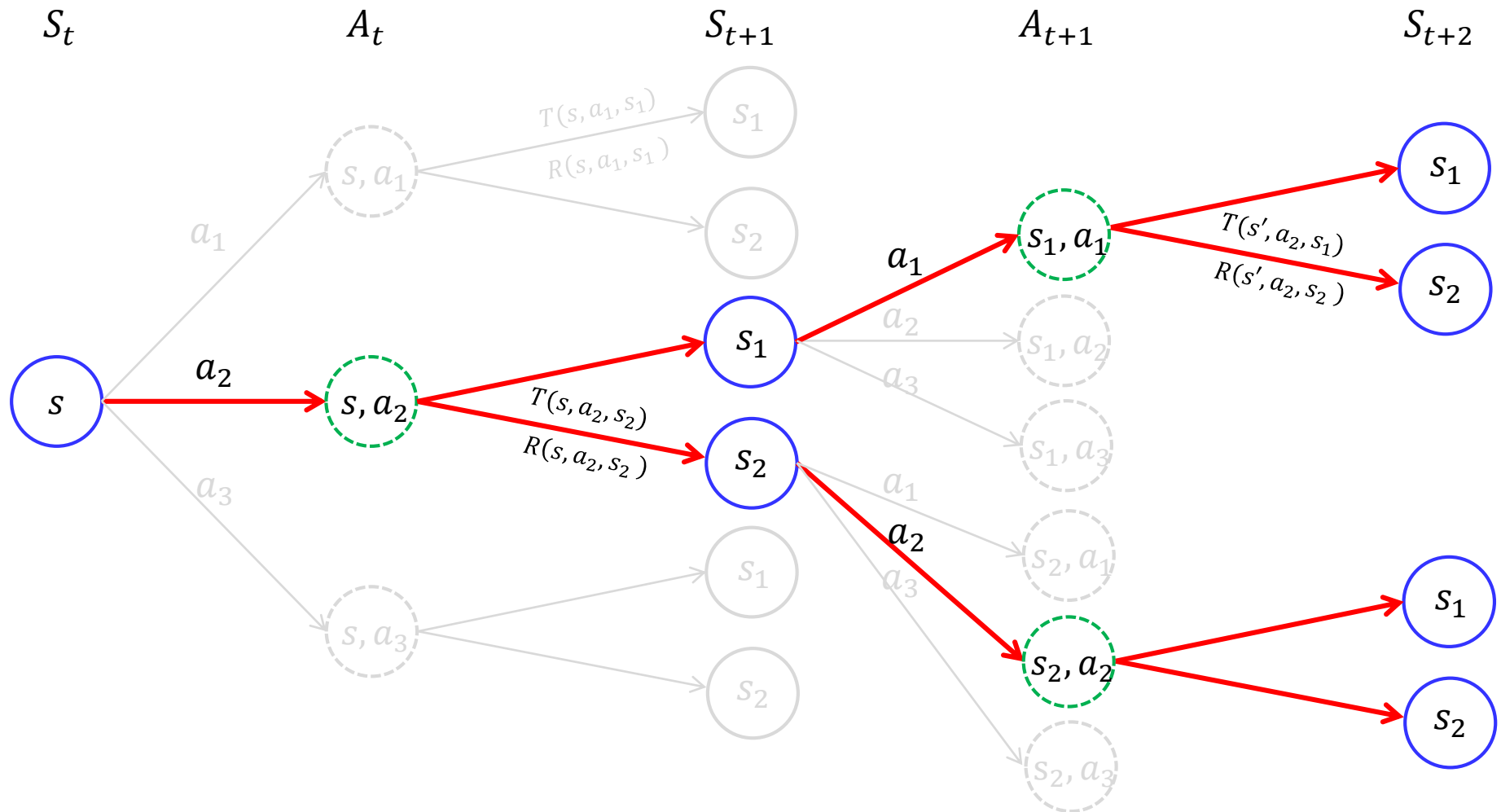
Value Function

All possible trajectories



Value Function

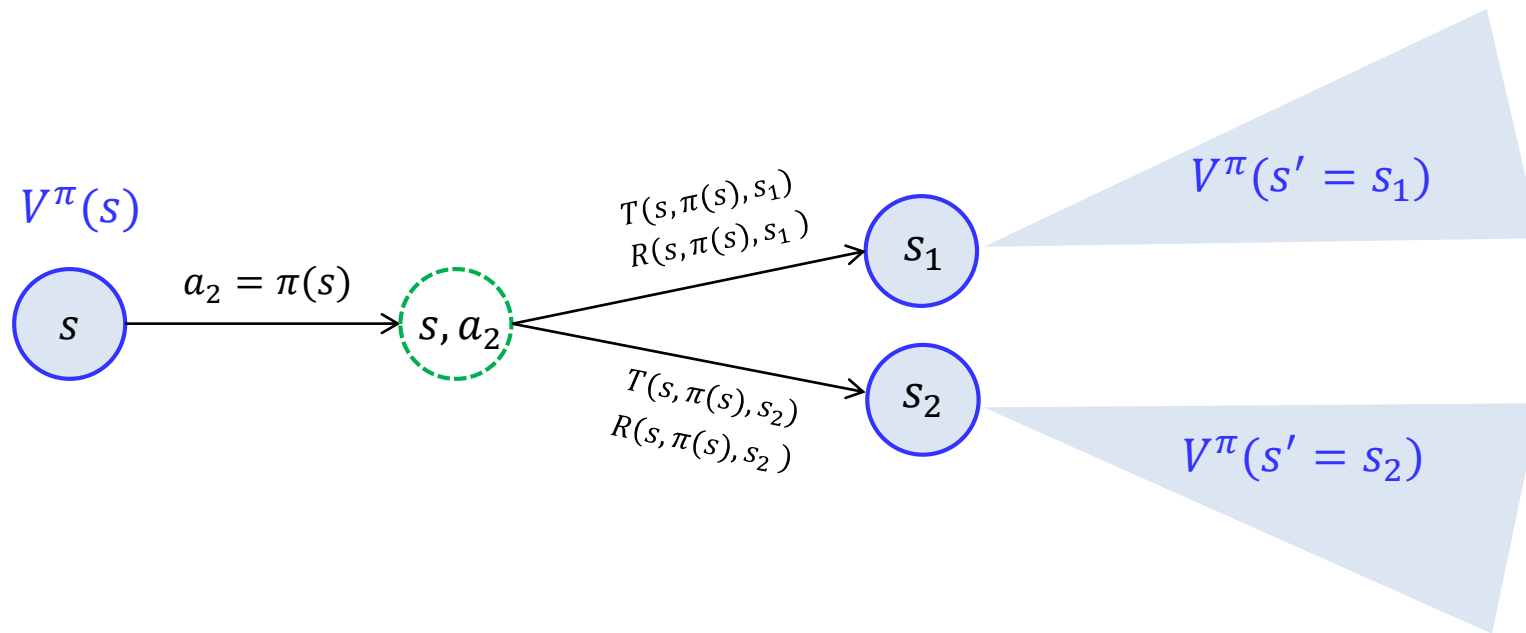
A policy π is given as: $\pi(s) = a_2$; $\pi(s_1) = a_1$; $\pi(s_2) = a_2$



Value Function

A policy π is given as: $\pi(s) = a_2$; $\pi(s_1) = a_1$; $\pi(s_2) = a_2$

S_t A_t S_{t+1} A_{t+1} S_{t+2}



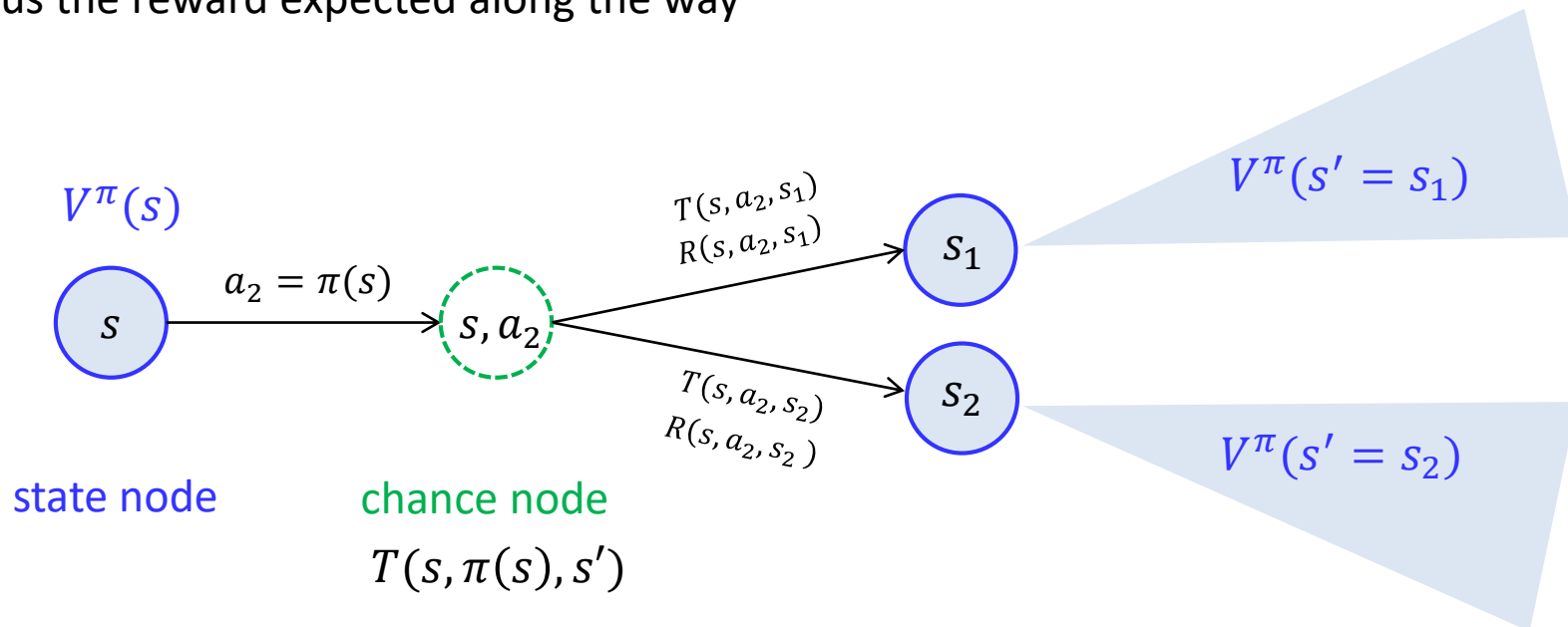
$$V^\pi(s) = T(s, \pi(s), \mathbf{s}_1) \{R(s, \pi(s), s_1) + \gamma V^\pi(s_1)\} + T(s, \pi(s), s_2) \{R(s, \pi(s), s_2) + \gamma V^\pi(\mathbf{s}_2)\}$$

The Bellman Equation for Value Function

Recursive Formulation (The Bellman equation)

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid S_t = s \right) \\ &= \mathbb{E}_\pi \left(r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid S_t = s \right) \\ &= \sum_{s'} T(s, \pi(s), s') \{ R(s, \pi(s), s') + \gamma \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid S_{t+1} = s' \right) \} \\ &= \sum_{s'} T(s, \pi(s), s') \{ R(s, \pi(s), s') + \gamma V^\pi(s') \} \end{aligned}$$

The value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way

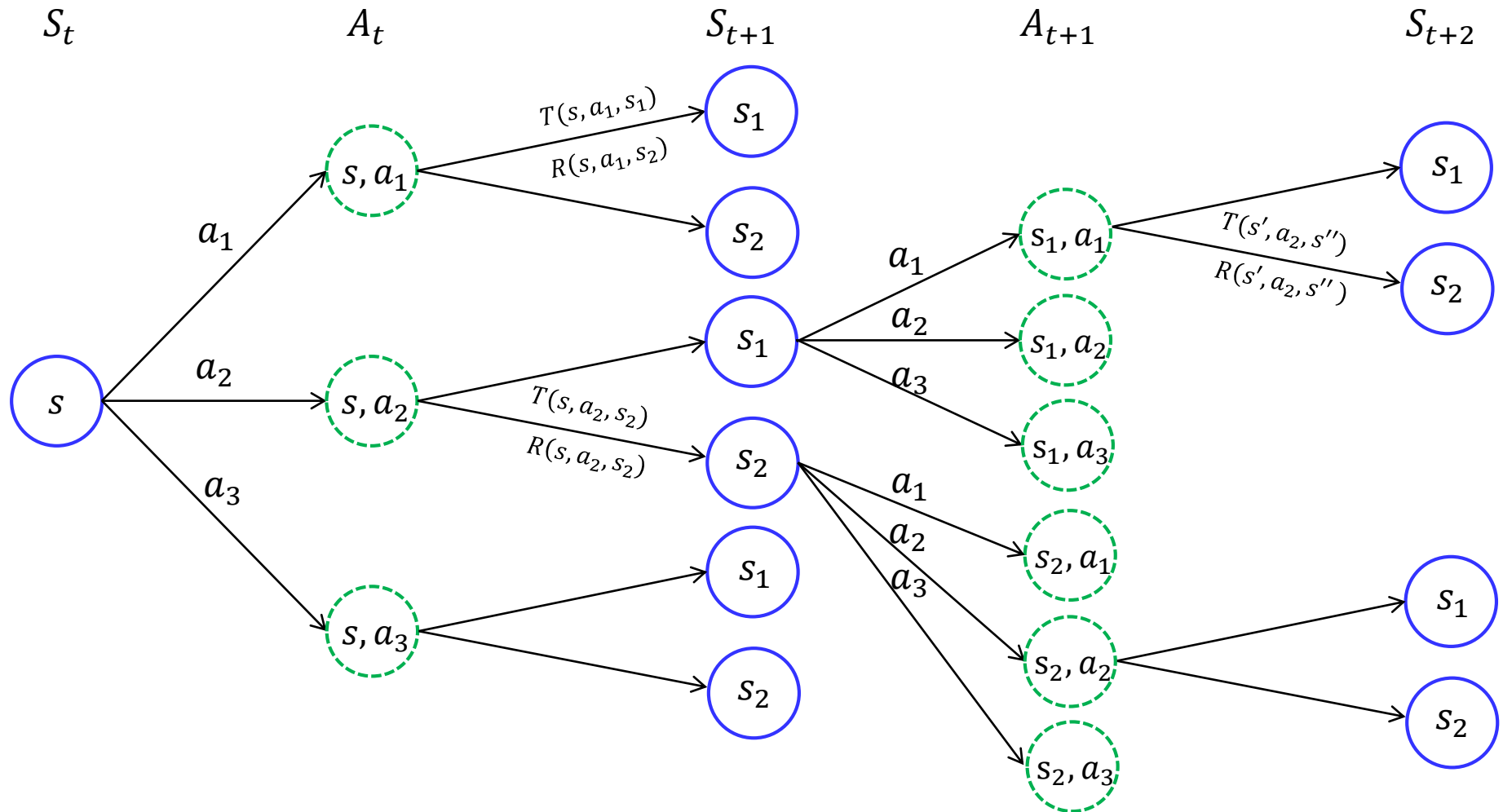


Value Function & Q function



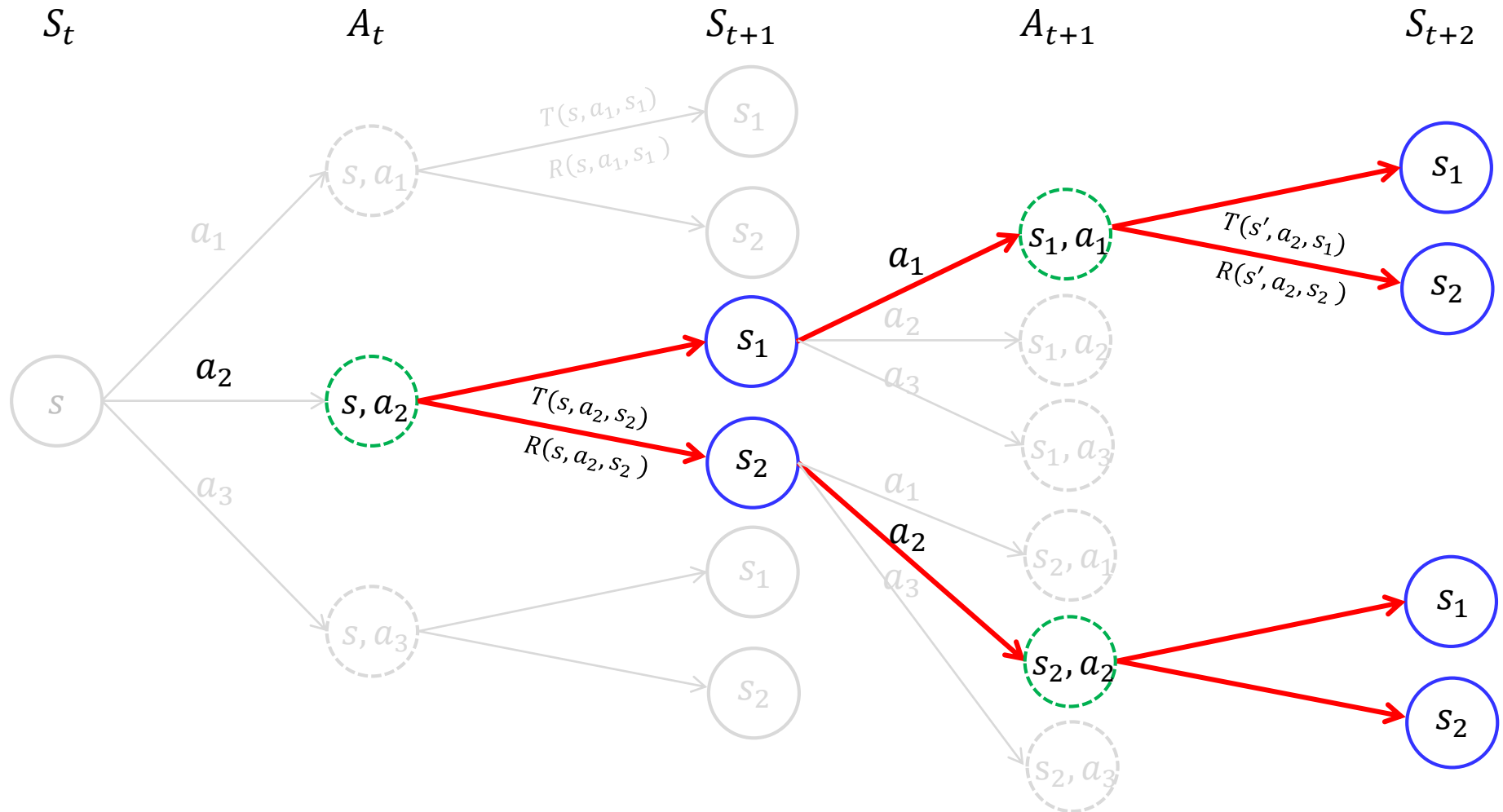
Q function

All possible trajectories



Q function

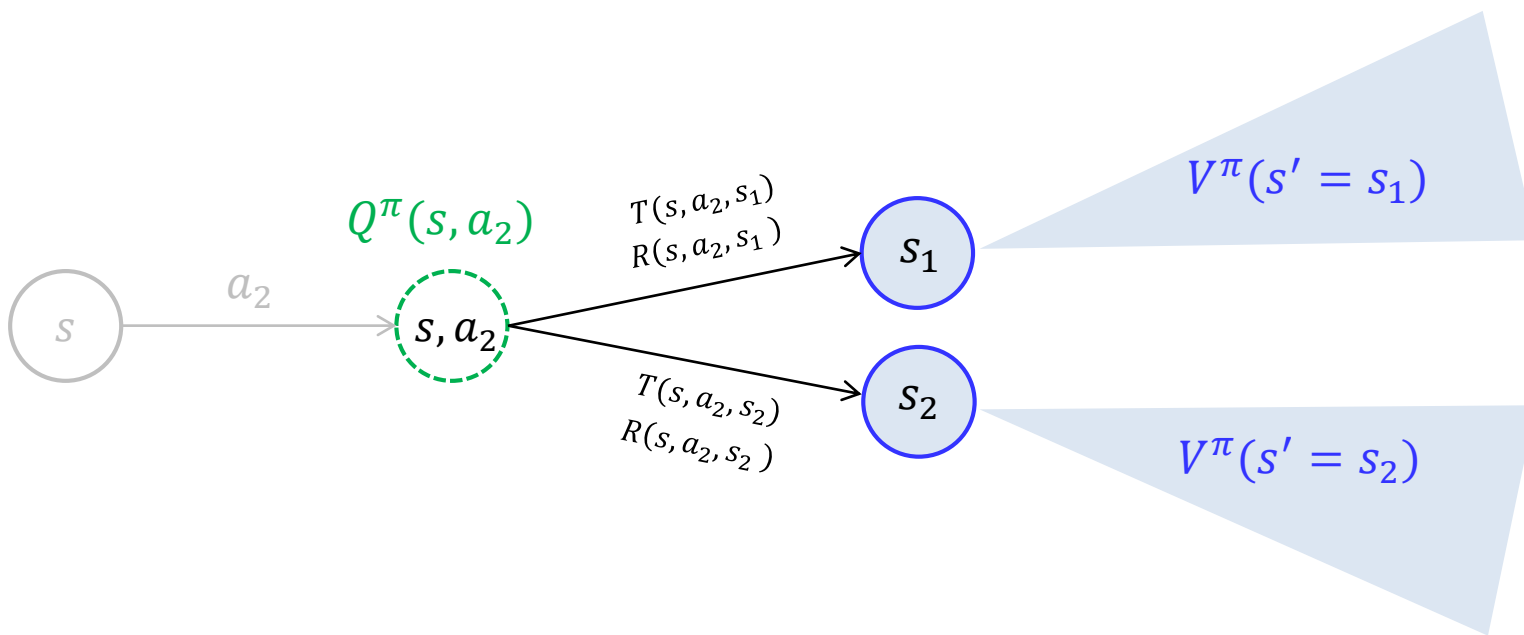
A policy π is given as: $\pi(s) = a_2$; $\pi(s_1) = a_1$; $\pi(s_2) = a_2$



Q Function

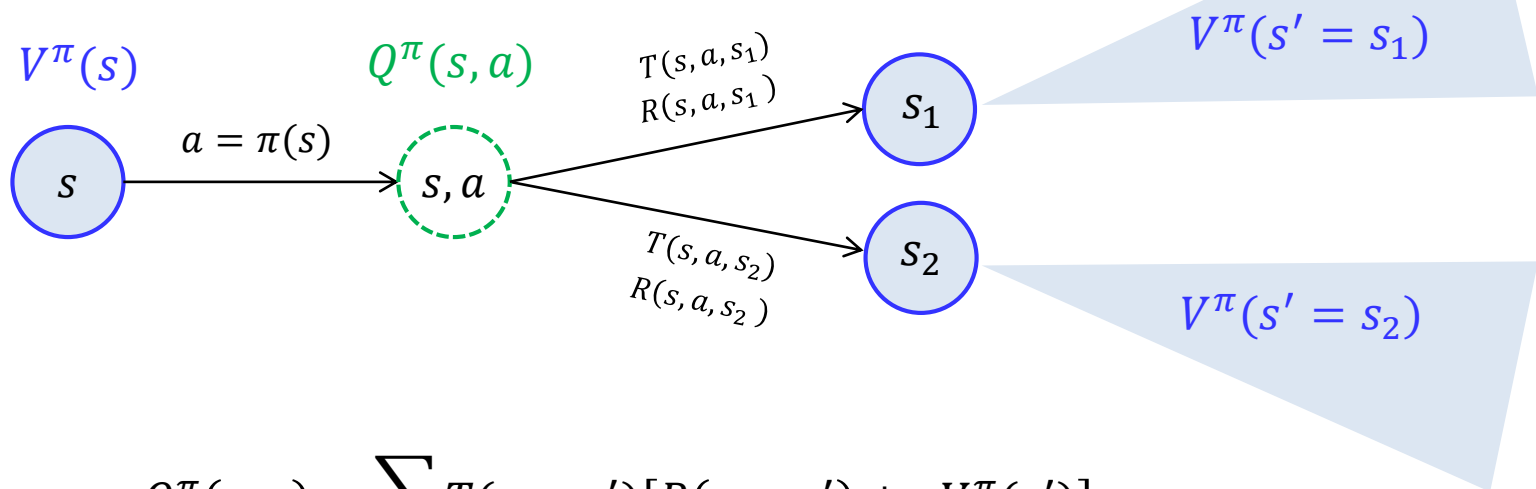
A policy π is given as: $\pi(s) = a_2$; $\pi(s') = a_1$; $\pi(s'') = a_2$

S_t A_t S_{t+1} A_{t+1} S_{t+2}



$$Q^\pi(s, a_2) = T(s, a_2, s_1)\{R(s, a_2, s_1) + \gamma V^\pi(s_1)\} + T(s, a_2, s_2)\{R(s, a_2, s_2) + \gamma V^\pi(s_2)\}$$

Q Function



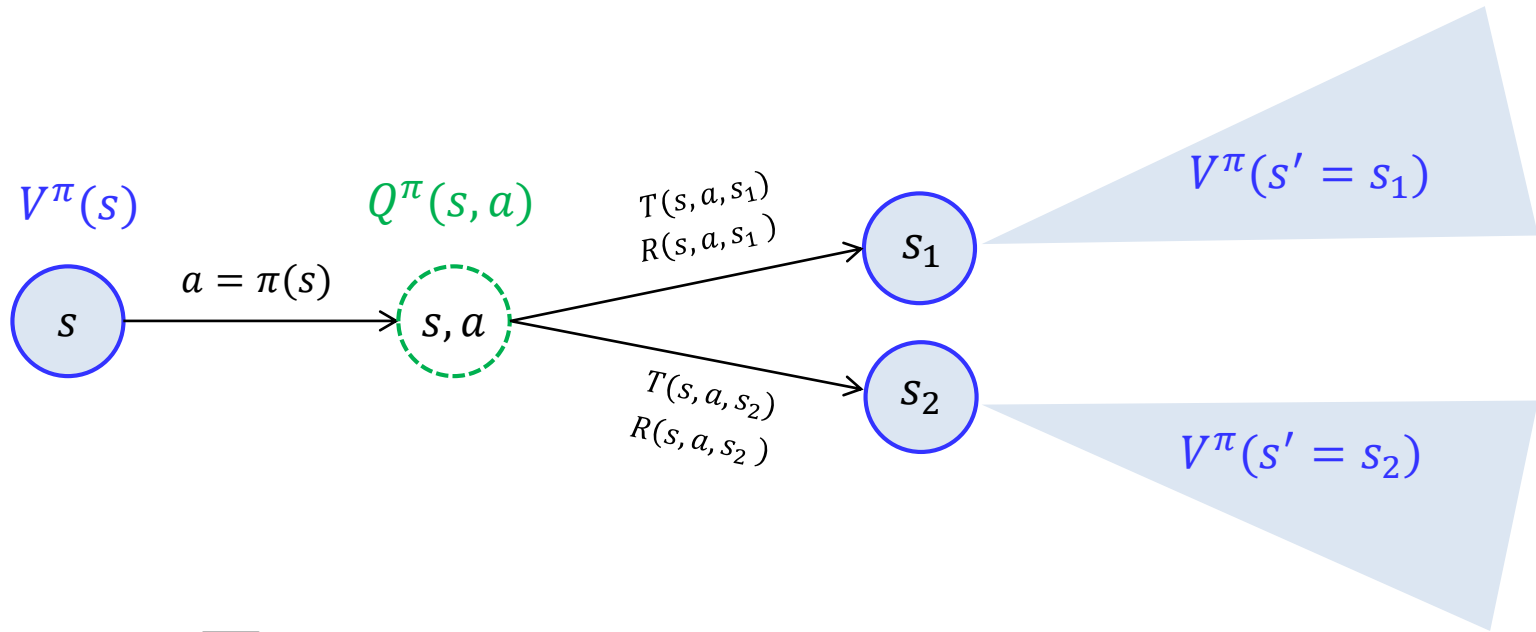
$$Q^\pi(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')]$$

Note that:

$$\begin{aligned} V^\pi(s) &= \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V^\pi(s')\} \\ &= Q^\pi(s, \pi(s)) \end{aligned}$$

- $Q^\pi(s, a)$ is more general since it has the option to select an action a given state s
- If the action is enforced to select $a = \pi(s)$ according to the policy π , $V^\pi(s) = Q^\pi(s, \pi(s))$

Summary for Value function and Q function



$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V^\pi(s')\}$$

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

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

Optimal Value Function & Q function

Optimal policy

$$\pi^* \geq \pi \text{ if and only if } V^{\pi^*}(s) \geq V^\pi(s) \text{ for all } s$$

Optimal state- value function

$$V^*(s) = \max_{\pi} V^\pi(s) \text{ for all } s$$

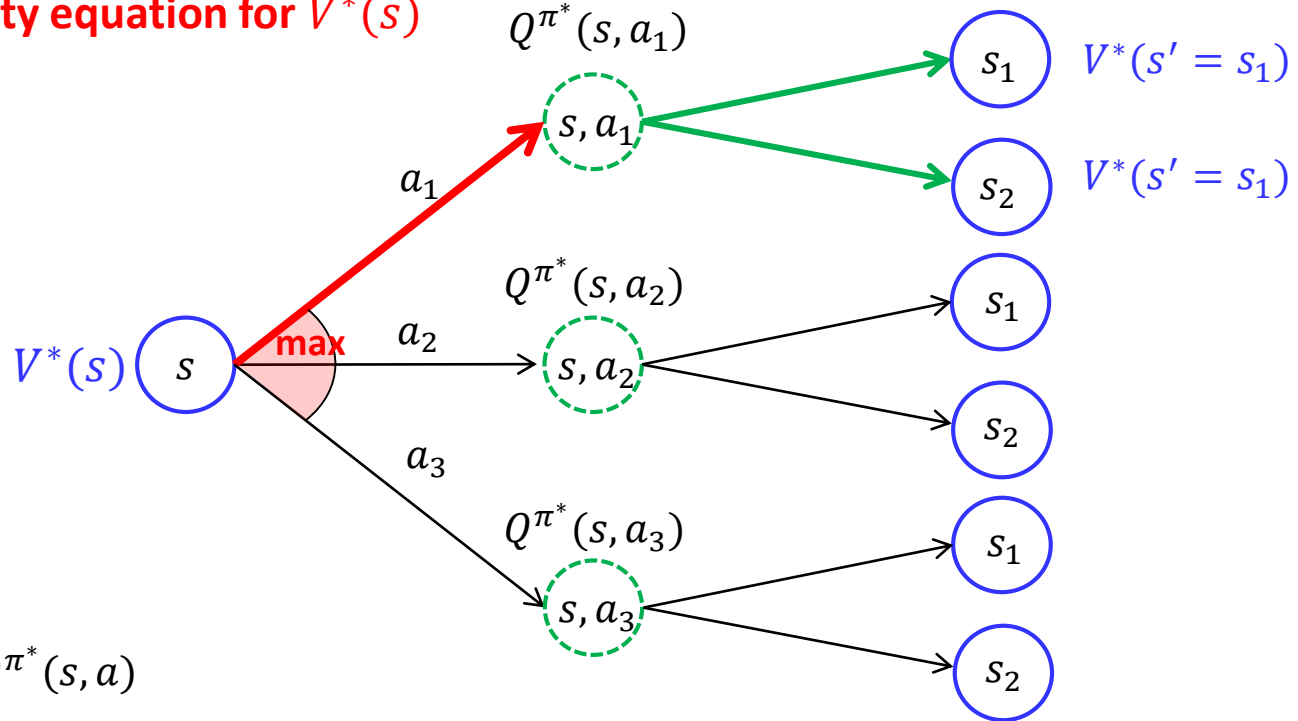
Optimal action-value function

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a) \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}(s)$$

$$\begin{aligned} Q^*(s, a) &= \max_{\pi} Q^\pi(s, a) && \because Q^\pi(s, a) = \mathbb{E}[R(s, a, s') + \gamma V^\pi(s') | s_t = s, a_t = a] \\ &= \max_{\pi} \mathbb{E}[R(s, a, s') + \gamma V^\pi(s') | s_t = s, a_t = a] && \mathbb{E} \text{ is over the next state } s' \\ &= \mathbb{E} \left[R(s, a, s') + \gamma \max_{\pi} V^\pi(s') \mid s_t = s, a_t = a \right] \\ &= \mathbb{E}[R(s, a, s') + \gamma V^*(s') | s_t = s, a_t = a] \end{aligned}$$

Bellman Optimality Equation for State-Value Function

Bellman optimality equation for $V^*(s)$



$$V^*(s) = \max_{a \in \mathcal{A}(s)} Q^{\pi^*}(s, a)$$

$$= \max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi^*} \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid S_t = s, A_t = a \right)$$

$$= \max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi^*} \left(r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid S_t = s, A_t = a \right) \quad \mathbb{E} \text{ is over transitions associated with } \pi^*$$

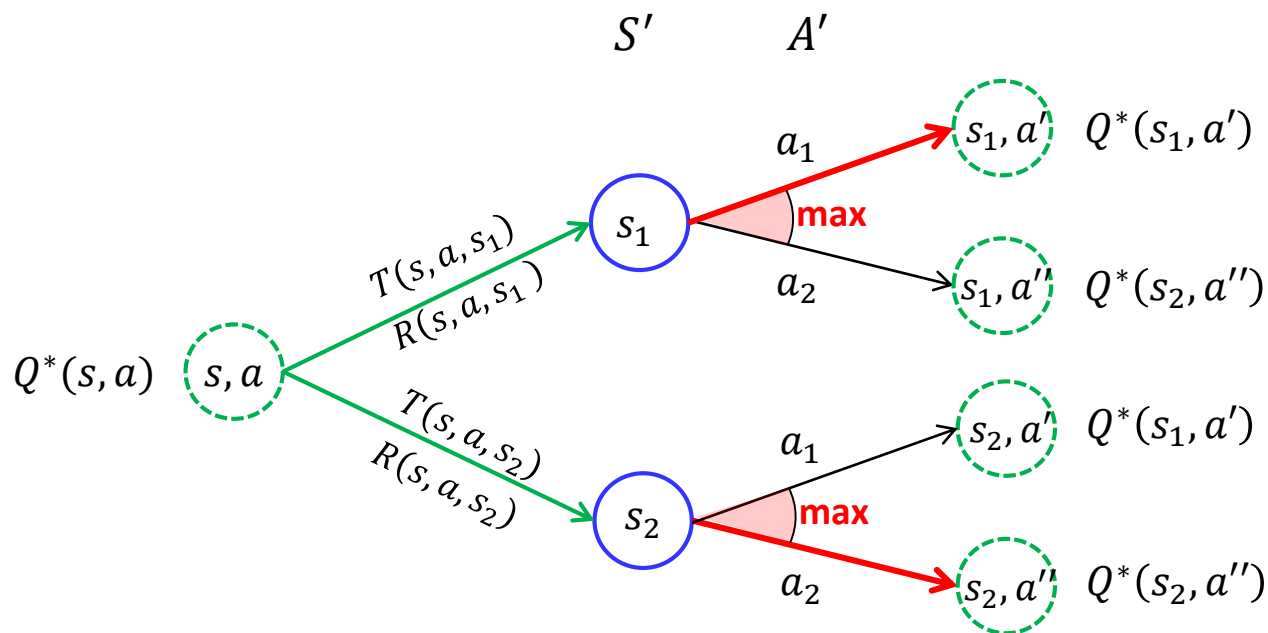
$$= \max_{a \in \mathcal{A}(s)} \mathbb{E} (r_t + \gamma V^*(s_{t+1}) \mid S_t = s, A_t = a) \quad \mathbb{E} \text{ is over transitions}$$

$$= \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\}$$

First take optimum action and follow the optimum policy

Bellman Optimality Equation for State-Action Value Function

Bellman optimality equation for $Q^*(s, a)$



$$\begin{aligned} Q^*(s, a) &= \mathbb{E} \left\{ r_t + \gamma \max_{a'} Q^*(s', a') \mid s_t = s, a_t = a \right\} \\ &= \sum_{s'} T(s, a, s') \left\{ R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right\} \end{aligned}$$

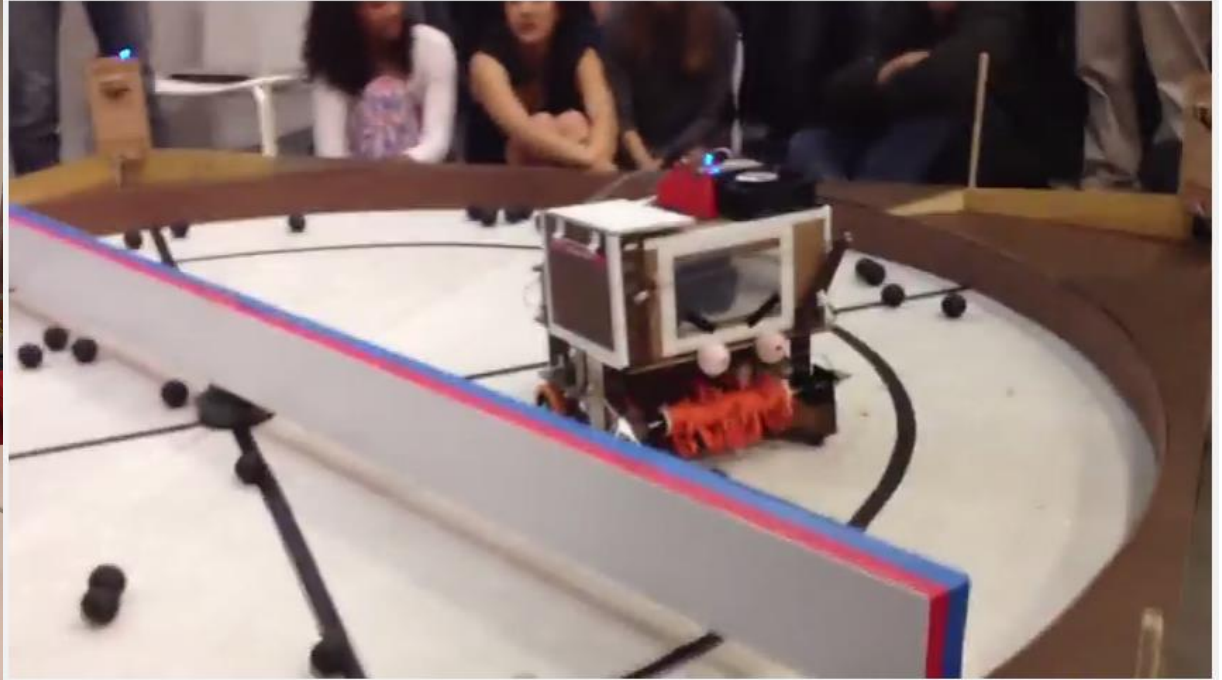
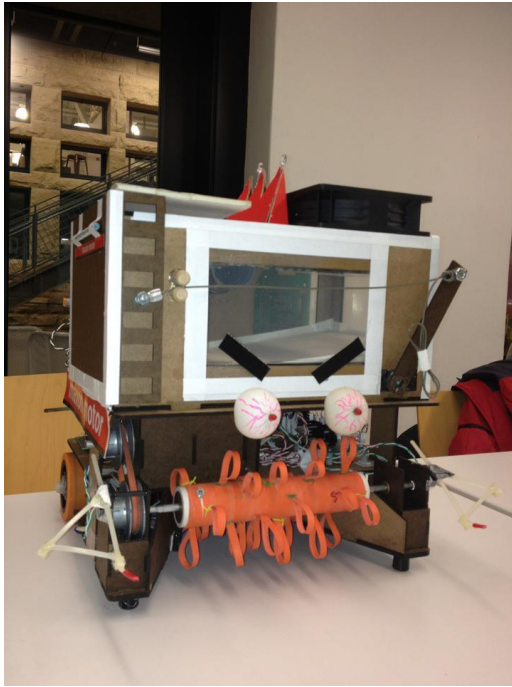
\mathbb{E} is over transitions $s' \rightarrow s'$

First transits by transition probability and take the optimum action for each consequent states

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\}$$

- The Bellman optimality equation is a system of equations, one for each state
 - ✓ For N states, N unknown and N equations
 - ✓ With known $T(s, a, s')$ and $R(s, a, s')$, the equations can be solved using various mathematical programming (i.e., Linear programming, Quadratic programming)

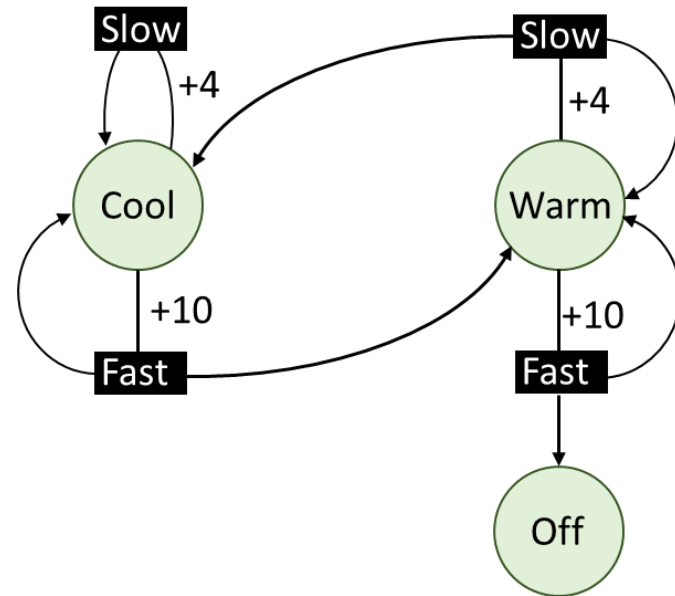
Robot Cleaner



<https://youtu.be/bFTvQt3Sj3Y?t=66>

Optimum Planning as a Greedy Search

s	a	s'	$T(s, a, s')$
cool	slow	cool	1
cool	fast	cool	1/2
cool	fast	warm	1/2
warm	slow	cool	1/2
warm	slow	warm	1/2
warm	fast	warm	1/2
warm	fast	off	1/2



$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\}$$

$$V^*(\text{cool}) = \max \left\{ \begin{array}{l} 1(4 + 0.9V^*(\text{cool})) \\ 0.5(10 + 0.9V^*(\text{cool})) + 0.5(10 + 0.9W) \end{array} \right\}$$

$$V^*(\text{warm}) = \max \left\{ \begin{array}{l} 0.5(4 + 0.9V^*(\text{cool})) + 0.5(4 + 0.9V^*(\text{warm})), \\ 0.5(10 + 0.9V^*(\text{warm})) + 0.5(10 + 0.9V^*(\text{off})) \end{array} \right\}$$

Optimum Planning as a Greedy Search

- **Reconstructing optimal policy with $Q^*(s, a)$ and $V^*(s)$**

$$\begin{aligned} a^* &= \operatorname{argmax}_a Q^*(s, a) \\ &= \operatorname{argmax}_a \mathbb{E}[R(s, a, s') + \gamma V^*(s') | s_t = s, a_t = a] \end{aligned}$$

- Any greedy policy with respect to the optimal value function $V^*(s)$ is an optimal policy
→ because $V^*(s)$ already takes into account the reward consequences of all possible future behavior
- The Q function effectively caches the results of all one-step-ahead search

Dynamic Programming

- The term **dynamic programming (DP)** refers to a collection of algorithms that can be used to compute optimal policies given a **perfect model of the environment** as a Markov decision process (MDP)
- The key idea of DP (and reinforcement learning) is the use of **value functions** to organize and structure the search for good policies
- Optimal policies can be derived from the optimal value functions that satisfy the Bellman optimality equations

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\}$$
$$Q^*(s, a) = \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma \max_{a'} Q^*(s', a')\}$$
$$= \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\} \quad \because V^*(s') = \max_{a'} Q^*(s', a')$$



Optimal policy

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$$
$$= \operatorname{argmax}_a \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\}$$

Policy Evaluation

Policy evaluation :

A method to compute the state-value function $V^\pi(s)$ for an arbitrary policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$

$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V^\pi(s')\}$$

➤ A system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknown

Algorithm

Initialize $V_{t=0}^\pi(s) \leftarrow 0$ for all states $s \in \mathcal{S}$

Repeat (iteration $t = 0, \dots$):

For each state s :

$$V_{t+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V_t^\pi(s')\}$$

Until $\max_{s \in \mathcal{S}} |V_{t+1}^\pi - V_t^\pi(s)| \leq e$

Full backup:

Each iteration of iterative policy evaluation backs up the value of every state once to produce the new approximate value function V_{t+1}^π

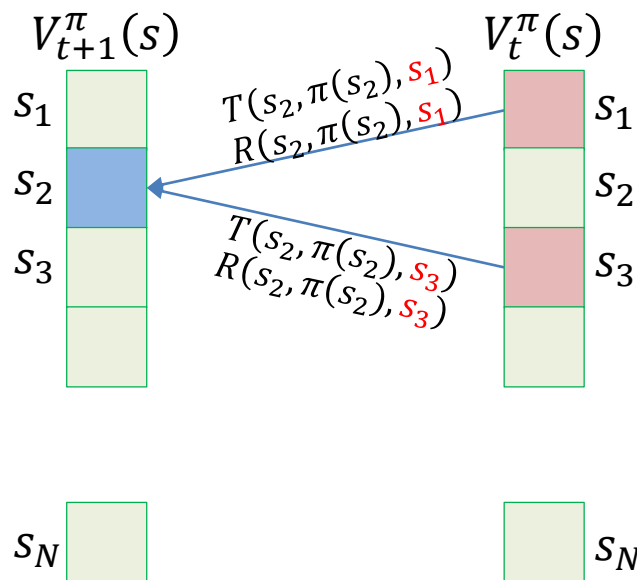
Policy Evaluation

$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$$

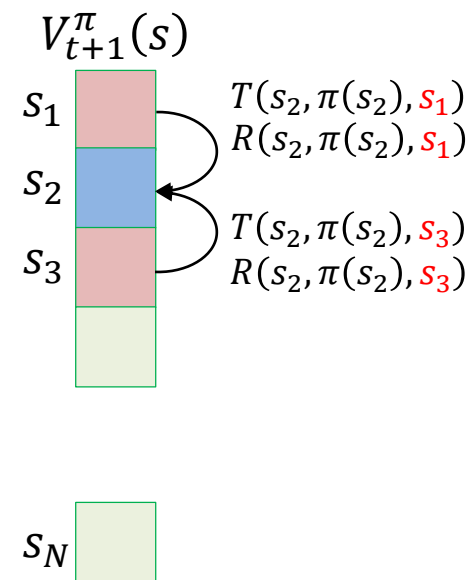
Example:

$$\begin{aligned} V_{t+1}^{\pi}(s_2) &= \sum_{s'} T(s_2, \pi(s_2), s') \{R(s_2, \pi(s_2), s') + \gamma V_t^{\pi}(s')\} \\ &= T(s_2, \pi(s_2), s_1) \{R(s_2, \pi(s_2), s_1) + \gamma V_t^{\pi}(s_1)\} + T(s_2, \pi(s_2), s_3) \{R(s_2, \pi(s_2), s_3) + \gamma V_t^{\pi}(s_3)\} \end{aligned}$$

“Two-arrays” update





“In place” update



Usually faster!
Less memory

Policy Evaluation

Example : Grid world

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$$MDP = \{\mathcal{S}, \mathcal{A}, T, R, \gamma\}$$

- $\mathcal{S} = \{1, 2, \dots, 14\}$
- $\mathcal{A} = \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$
- $T(s, s', a) = \begin{cases} 1, & \text{if move is allowed} \\ 0, & \text{if move is not allowed} \end{cases}$

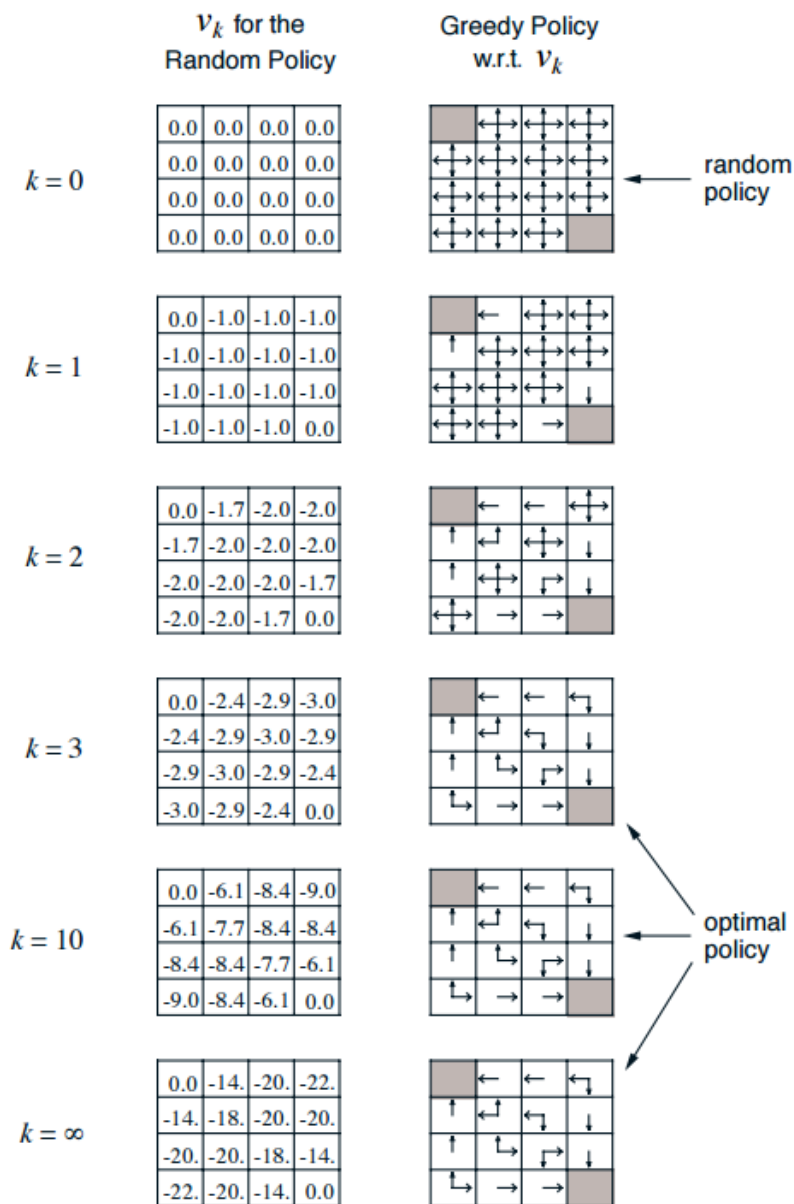
$$T(5, 6, \rightarrow) = 1 \quad T(5, 10, \rightarrow) = 0 \quad T(7, 7, \rightarrow) = 1$$

The actions that would take the agent off the grid leave the state unchanged

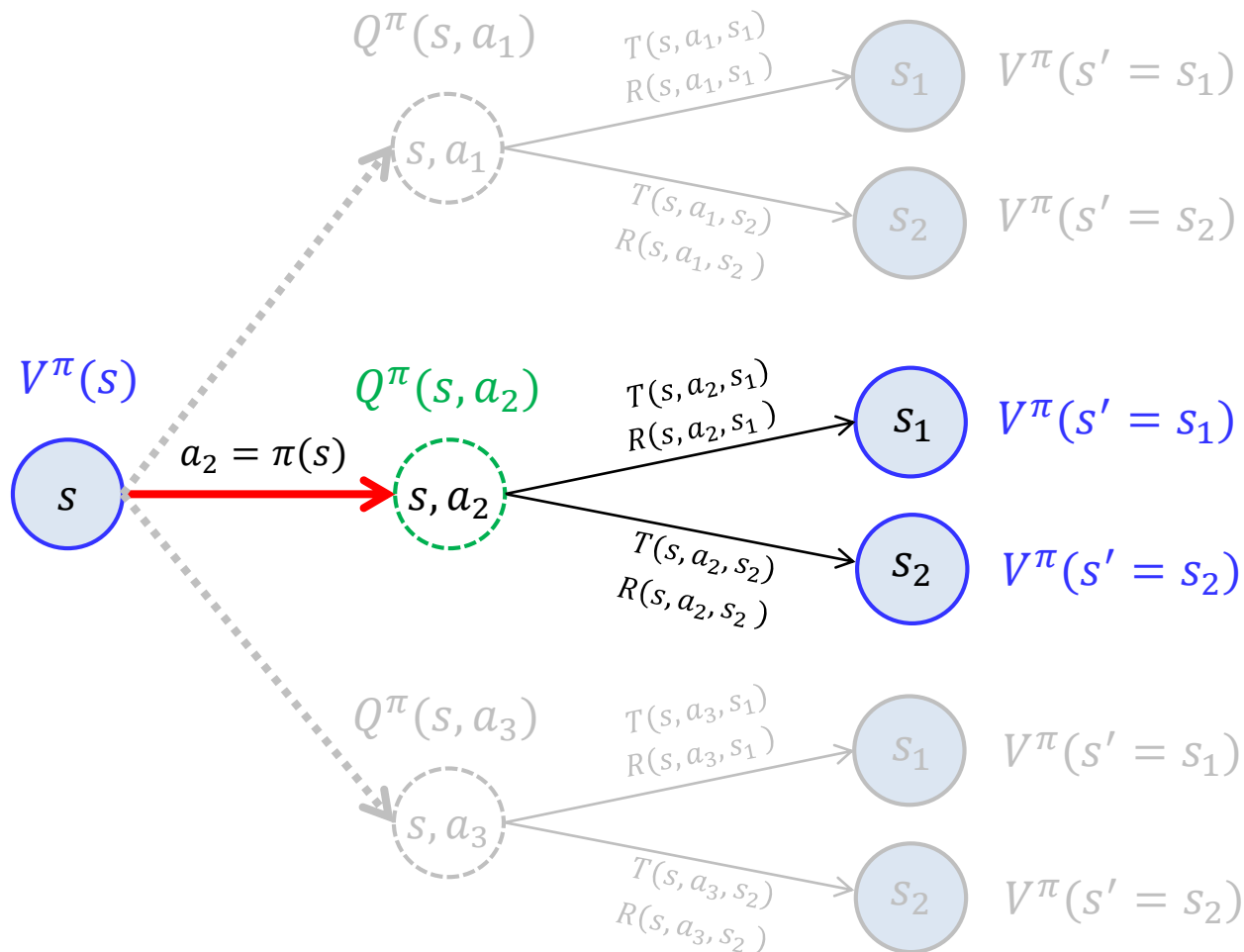
- $R(s, s', a) = -1$ for all s, s', a
- $\gamma = 1$

Suppose the agent follows the equiprobable random policy (all actions equally likely), what is the value function?

Policy Evaluation



Policy Improvement



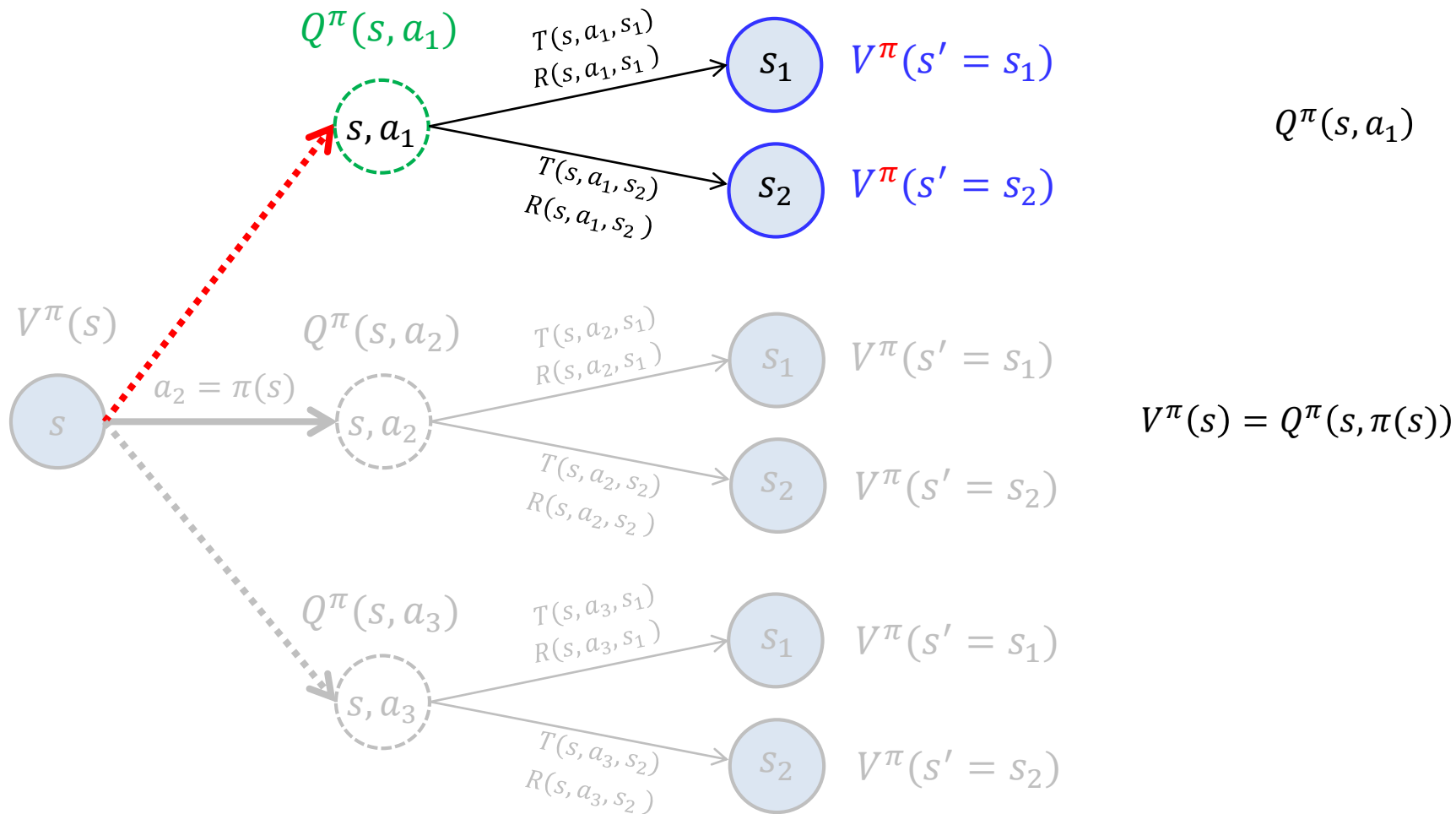
$$V^\pi(s) = Q^\pi(s, \pi(s))$$

We know how good it is to follow the current policy from s based on $V^\pi(s)$

Would it be better or worse to change to the new policy?

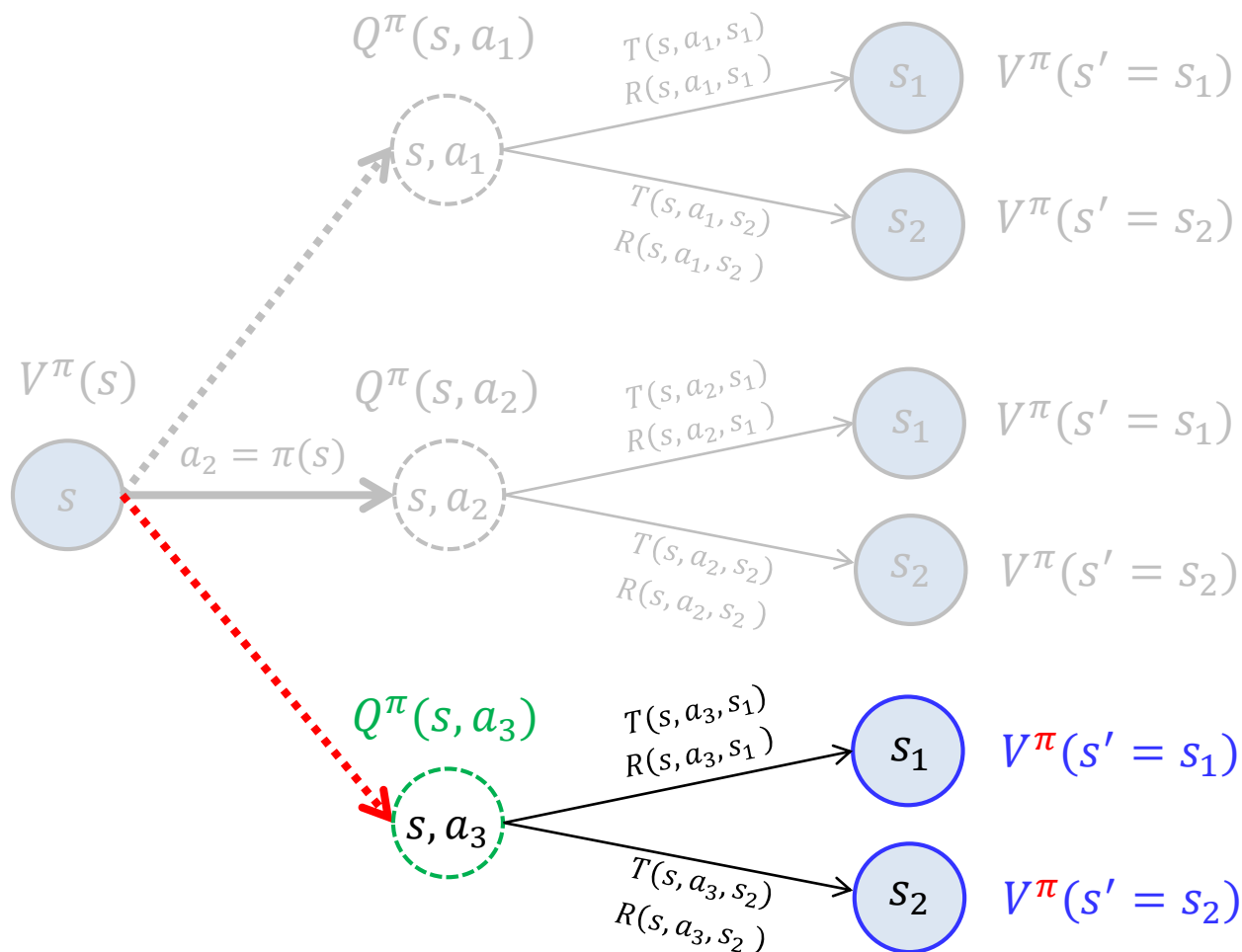
→ Select a given s and thereafter following the existing policy π (a single step change)

Policy Improvement



$$Q^\pi(s, a_1) \geq V^\pi(s) = Q^\pi(s, \pi(s))?$$

Policy Improvement



$$V^\pi(s) = Q^\pi(s, \pi(s))$$

$$Q^\pi(s, a_3)$$

$$Q^\pi(s, a_3) \geq V^\pi(s) = Q^\pi(s, \pi(s))?$$

Policy Improvement

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}(s)} Q^\pi(s, a)$$

$$\rightarrow Q^\pi(s, \pi'(s)) \geq Q^\pi(s, \pi(s)) = V^\pi(s)$$

$$x^* = \operatorname{argmax}_x f(x)$$

$$\rightarrow f(x^*) \geq f(x) \text{ for all } x$$

Improvement criterion =

Expected reward provided by **changing one step action** and **following the original policy**

If it is better to select $a = \pi'(s)$ once in s and thereafter follow π than it would be to follow π all the time,



It is better still to select $a = \pi'(s)$ whenever s is encountered
(The new policy $\pi'(s)$ is a better policy overall)

Policy Improvement

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}(s)} Q^\pi(s, a)$$

$$\rightarrow Q^\pi(s, \pi'(s)) \geq Q^\pi(s, \pi(s)) = V^\pi(s)$$

$$x^* = \operatorname{argmax}_x f(x)$$

$$\rightarrow f(x^*) \geq f(x) \text{ for all } x$$

Improvement criterion =

Expected reward provided by **changing one step action** and **following the original policy**

Proof (Policy improvement Theorem)

Policy improvement must give us a strictly better policy $\pi'(s)$ than the older policy $\pi(s)$ except when the original policy is already optimal $\pi(s) = \pi^*(s)$

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s) \rightarrow \underline{V^{\pi'}(s) \geq V^\pi(s) \text{ for all states } s \in \mathcal{S}}$$

$$\pi' \geq \pi$$

Policy Improvement

Proof (Policy improvement Theorem)

Policy improvement must give us a strictly better policy $\pi'(s)$ than the older policy $\pi(s)$ except when the original policy is already optimal $\pi(s) = \pi^*(s)$

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s) \rightarrow V^{\pi'}(s) \geq V^\pi(s) \quad \text{for all states } s \in \mathcal{S}$$

$$V^\pi(s) \leq Q^\pi(s, \pi'(s))$$

Given

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s]$$

$\mathbb{E}_{\pi'}$ is expectation over s_{t+1} induced by π'

$$\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) | s_t = s] \quad \because V^\pi(s_{t+1}) \leq Q^\pi(s_{t+1}, \pi'(s_{t+1}))$$

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma \mathbb{E}_{\pi'}[r_{t+2} + \gamma V^\pi(s_{t+2})] | s_t = s]$$

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V^\pi(s_{t+2}) | s_t = s]$$

$$\leq \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 Q^\pi(s_{t+2}, \pi'(s_{t+2})) | s_t = s]$$

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 \mathbb{E}_{\pi'}[r_{t+3} + \gamma V^\pi(s_{t+3})] | s_t = s]$$

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^\pi(s_{t+3}) | s_t = s]$$

\vdots

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots | s_t = s]$$

$$= V^{\pi'}(s)$$

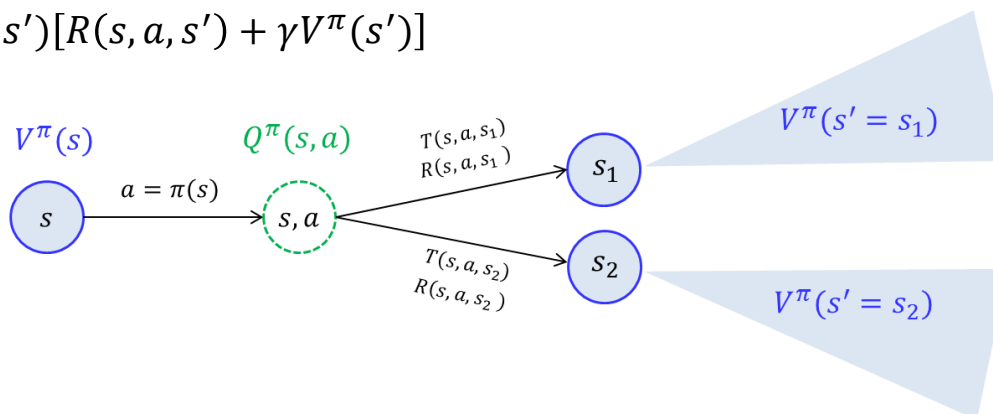
Thus, $\pi \leq \pi'$

Policy Improvement

Policy improvement :

The process of making a new policy π^{new} that improves the original policy π , by making it greedy or nearly greedy with respect to the value function of the original policy

Recall:
$$Q^\pi(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')]$$



Algorithm

Input : value of policy $V^\pi(s)$

Output: new policy π'

For each state $s \in \mathcal{S}$

1. Compute $Q^\pi(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')]$ for each a

2. Compute $\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}(s)} Q^\pi(s, a)$

$$= \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')]$$

Policy Iteration

Policy iteration :

Iterative way of finding the optimum policy through sequence of policy evaluation and policy improvement



Algorithm

```
 $\pi \leftarrow \text{arbitrary}$   
For  $t = 1, \dots, t_{PI}$  (or until  $\pi$  stops changing)  
  Run policy evaluation to compute  $V^\pi$   
  Run policy improvement to get new improved policy  $\pi'$   
   $\pi \leftarrow \pi'$ 
```

- Policy evaluation require iterative computation, requiring multiple sweeps through the state set
- policy evaluation starts with the value function for the *previous policy*

→ *Fast convergence*

Policy Iteration

Policy iteration :

Iterative way of finding the optimum policy through sequence of policy evaluation and policy improvement

Iteration

For $t = 0, \dots$ until convergence

For each state s :

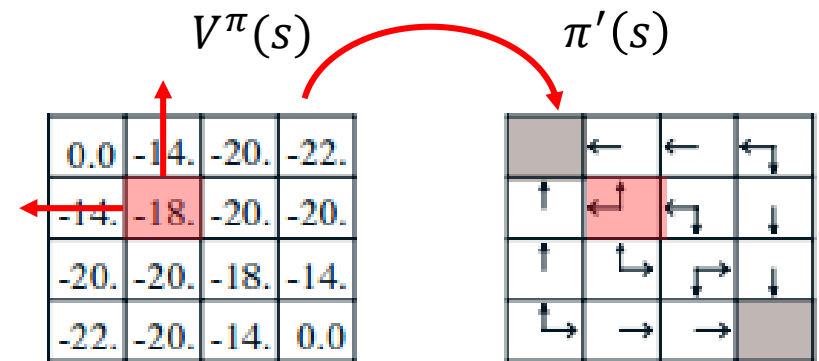
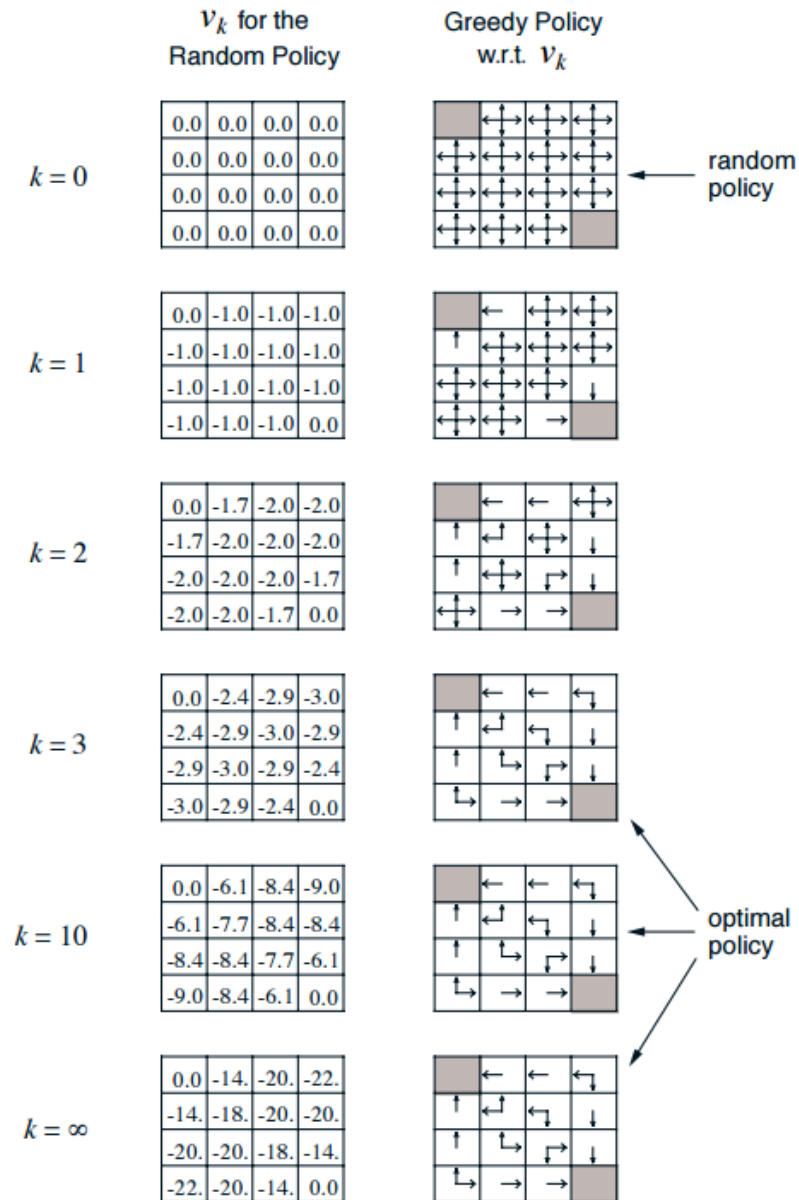
$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$$

Converged state value function $V^{\pi}(s)$

For each state s :

$$\begin{aligned} \pi'(s) &= \operatorname{argmax}_{a \in \mathcal{A}(s)} Q^{\pi}(s, a) \\ &= \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi}(s')] \end{aligned}$$

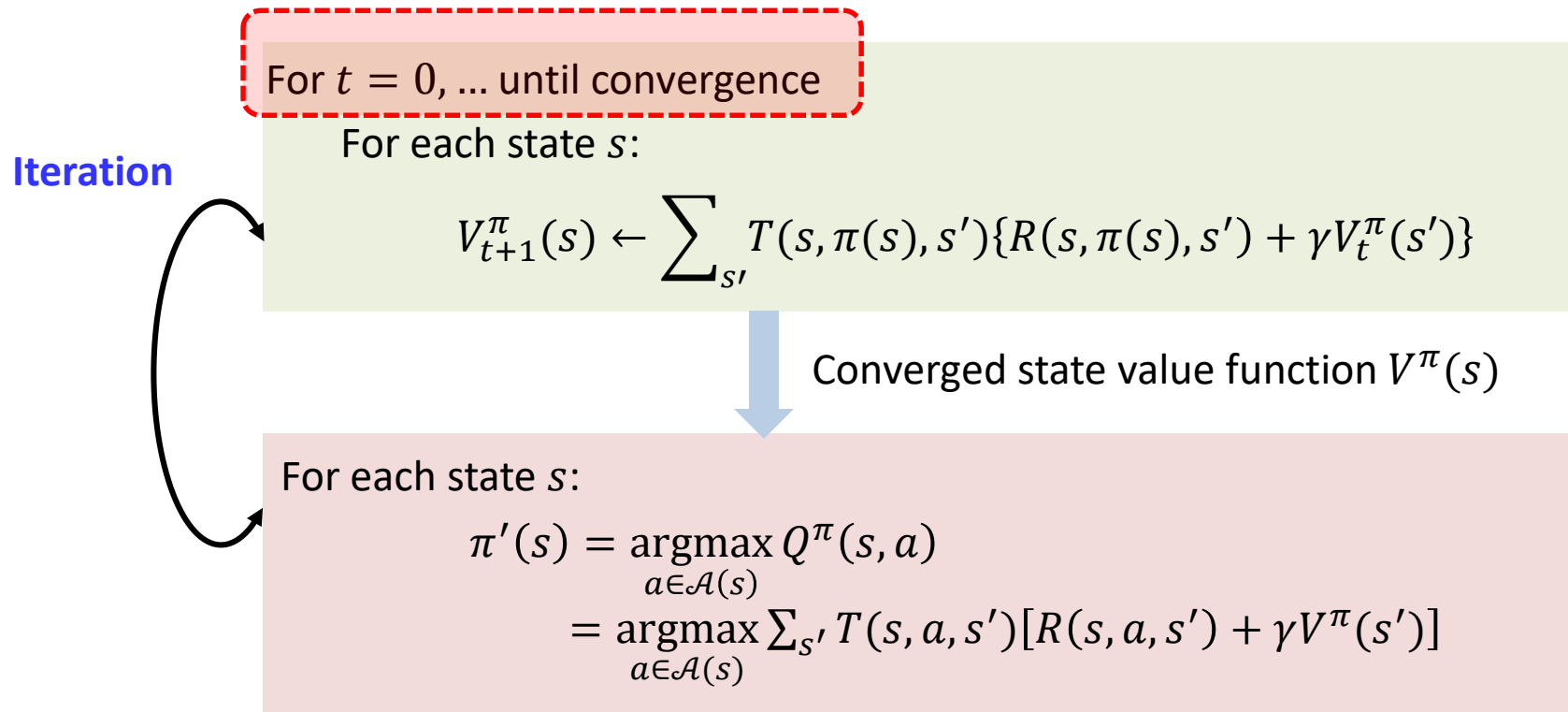
Policy Iteration



Policy Iteration

Policy iteration :

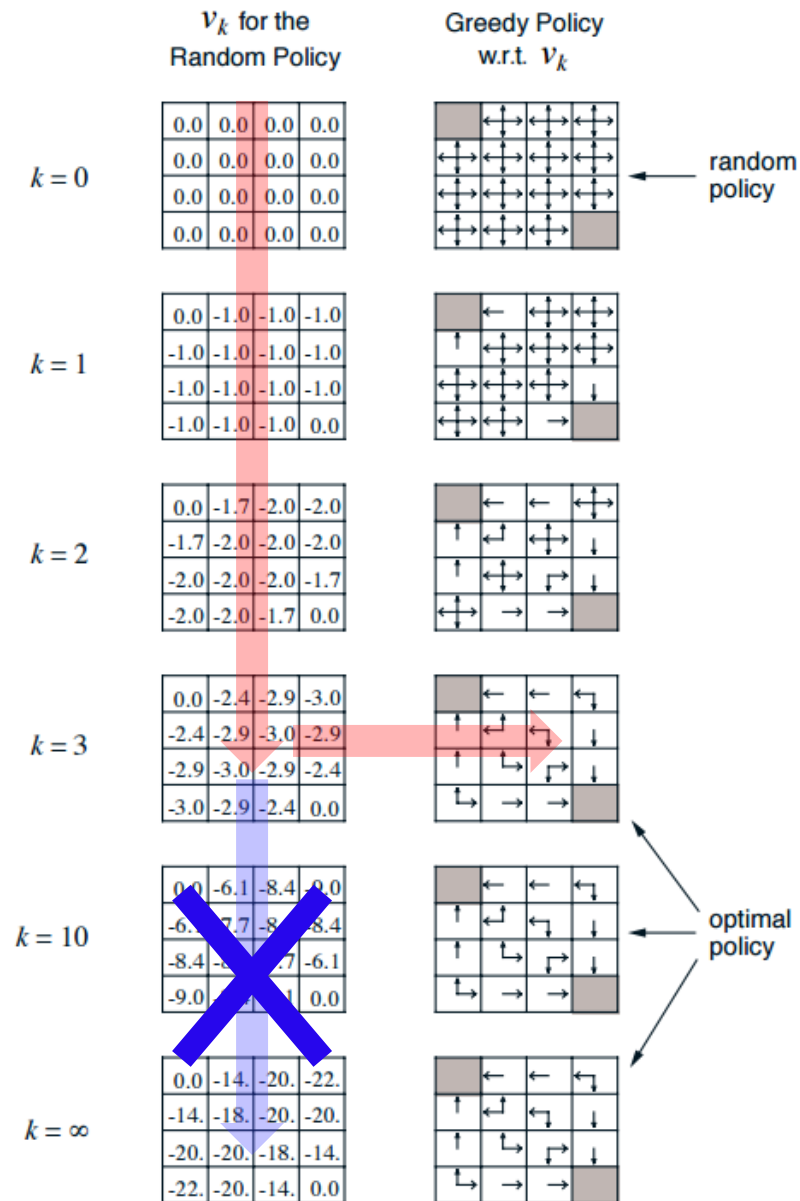
Iterative way of finding the optimum policy through sequence of policy evaluation and policy improvement



Issues :

Policy evaluation requires iterative computation, requiring multiple sweeps through the state set → slow to converge

Policy Iteration



Value Iteration

Solution:

- Stop policy evaluation after just one sweep (one backup of each state)
- Combine one sweep of policy evaluation and one sweep of policy improvement

For each state s :

$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$$

+

For each state s :

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_{t+1}^{\pi}(s')]$$

↓

For each state s :

Value Iteration

$$V_{t+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V_t(s')\}$$

or, can be obtained simply by turning the Bellman optimality equation into an update rule :

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s')\}$$

Value Iteration

Value Iteration:

A method to compute the optimum state-value function $V^*(s)$ by combining one sweep of policy evaluation and one sweep of policy improvement

Algorithm

Initialize $V(s) \leftarrow 0$ for all states $s \in S$

Repeat

For each state s :

$$V(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V(s)\}$$

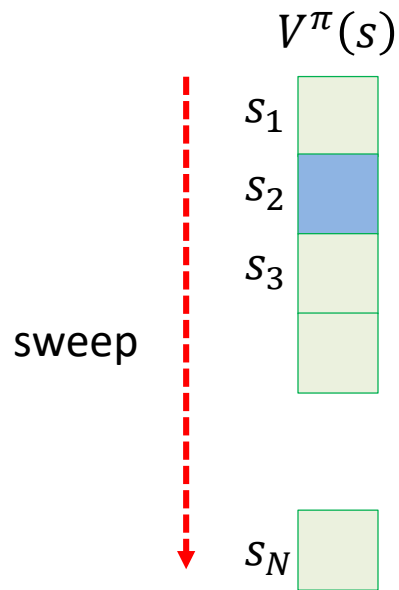
Until $\max_{s \in S} |V_t(s) - V_{t-1}(s)| \leq e$

Optimum policy can be obtained from the converged $V^*(s)$:

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V^*(s)\}$$

Asynchronous DP Algorithms

A major drawback to the DP methods is that they involve operations over the entire state set of the MDP

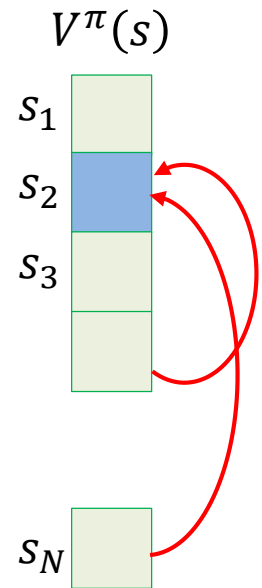


Conventional DP Algorithms

- Black mon game has 10^{20} states
- Go game has $3^{(19 \times 19)}$ states
- ...



- Take forever to sweep all states
- Does not improve policy until value functions are full backed up



Asynchronous DP Algorithms

- Back up the values of states in any order whatsoever, using whatever values of other states happen to be available
- Allow great flexibility in selecting states to which backup operations are applied
- Make it easier to intermix computation with real-time interaction: To solve a given MDP, we can run iterative DP algorithm at the same time that an agent is actually experiencing the MDP (Reinforcement Learning !!!!)

Value Iteration

Policy Evaluation

For $t = 1, \dots$

For each state s :

$$V_{t+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V_t^{\pi}(s')\}$$

Policy Improvement

For each state s :


$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi}(s')]$$

Policy Iteration




Value Iteration

For each state s :

$$V_{t+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V_t(s')\}$$


Asynchronous Value iteration

For any single state s :

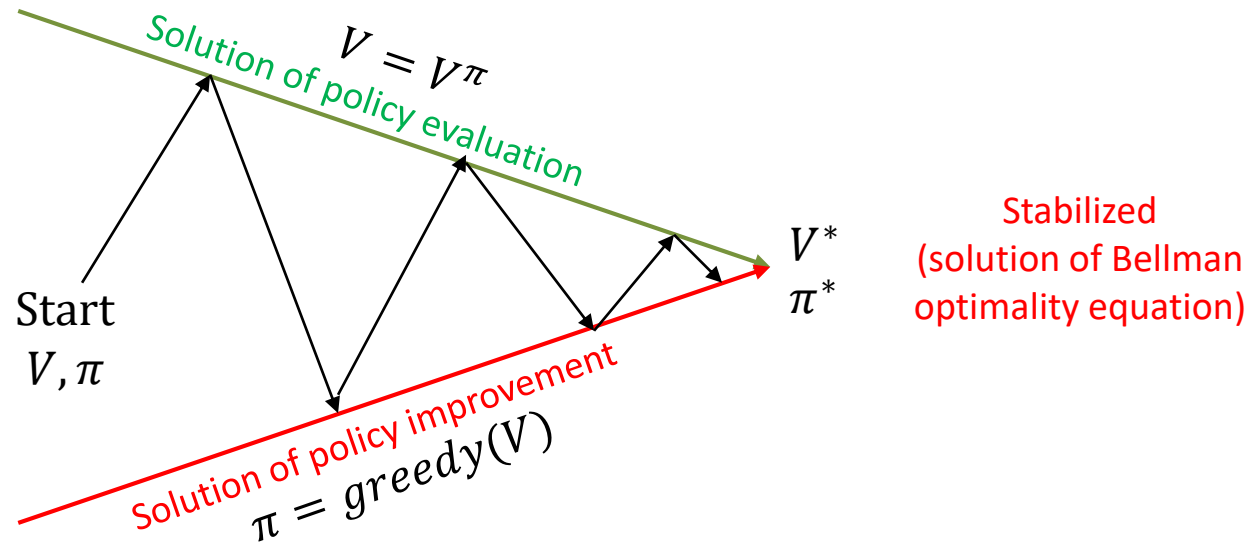
$$V(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} T(s, a, s') \{R(s, a, s') + \gamma V(s')\}$$


As long as both processes continue to update all states, the ultimate result is typically the same-convergence to the optimal value function and an optimal policy

Generalized Policy Iteration

Generalized Policy Iteration :

an general idea of interaction between policy evaluation and policy improvement processes



- Making policy greedy with respect to the value function typically makes the value function incorrect for the changed policy
- Making the value function consistent with the policy typically causes that policy no longer to be greedy
- In the long run, however, these two processes interact to find a single joint solution: the optimal value function and optimal policy