

Introduction to RL

2021. 03. 30.

김준호

Bandit based 추천시스템

David Silver

- A principal research scientist at DeepMind
- A professor at UCL(University College London)

- on indefinite leave of absence from UCL

- Research

- Starcraft (Grandmaster level in StarCraft II using multi-agent reinforcement learning)
- AlphaGo (Human-level control through deep reinforcement learning)
- Rainbow (Rainbow: Combining improvements in deep reinforcement learning)
- DQN (Playing atari with deep reinforcement learning)



David Silver

Google DeepMind
google.com의 이메일 확인됨 - 홈페이지

Artificial Intelligence Machine Learning Reinforcement Learning Monte-Carlo Search Computer Games

팔로우

내 프로필 만들기



Introduction [1/12]

Characteristics of RL

What makes reinforcement learning different from other machine learning paradigms?

- There is no supervisor, only a reward signal : trial and error paradigm
- Feedback is delayed, not instantaneous delay 될 수 있다.
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

Introduction [2/12]

Rewards

- A **reward** R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximise cumulative reward

Reinforcement learning is based on the **reward hypothesis**

Definition (Reward Hypothesis)

All goals can be described by the maximisation of expected cumulative reward

Do you agree with this statement?

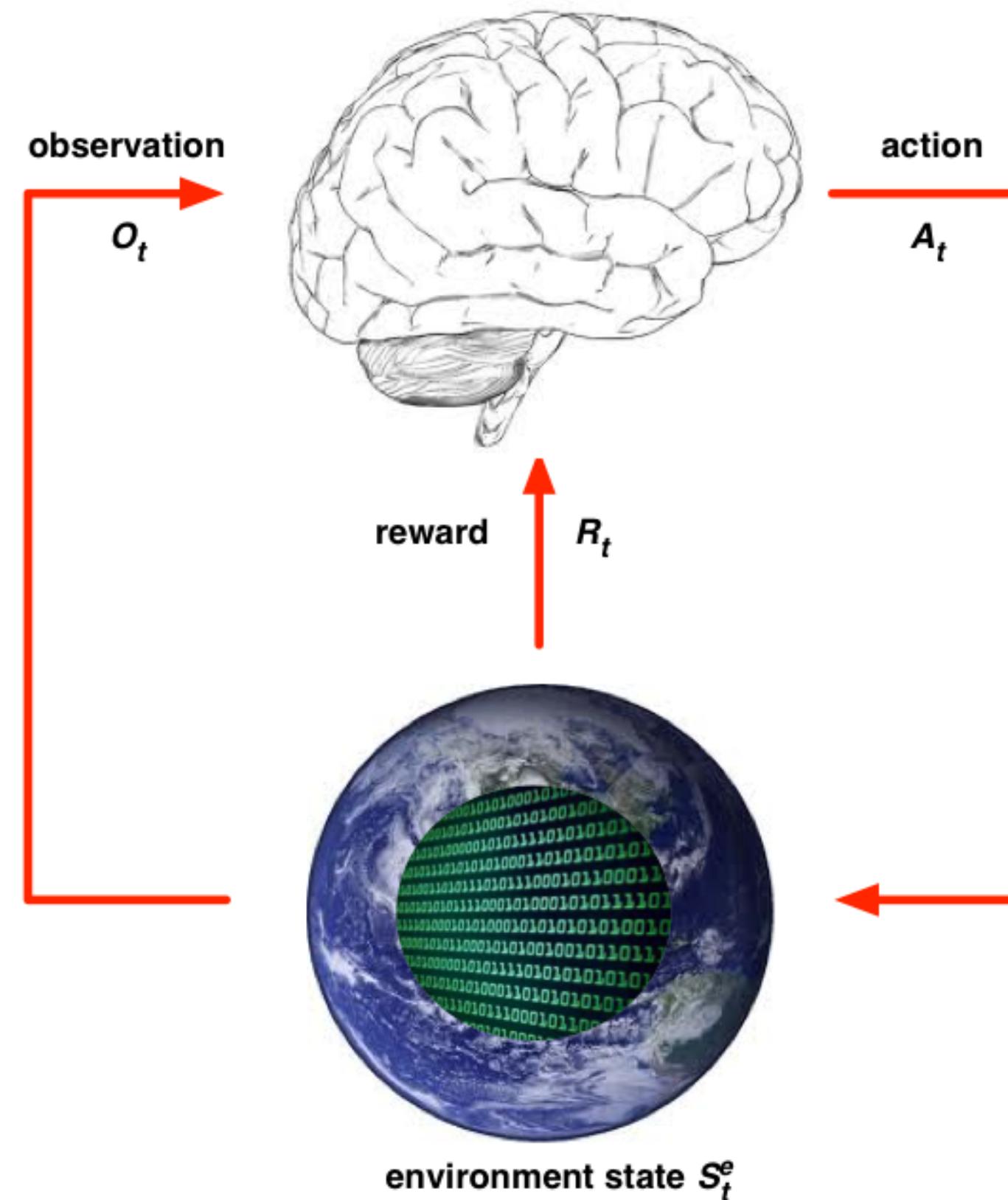
Introduction [3/12]

Sequential Decision Making

- Goal: *select actions to maximise total future reward*
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
- Examples:
 - A financial investment (may take months to mature)
 - Refuelling a helicopter (might prevent a crash in several hours)
 - Blocking opponent moves (might help winning chances many moves from now)

Introduction [4/12]

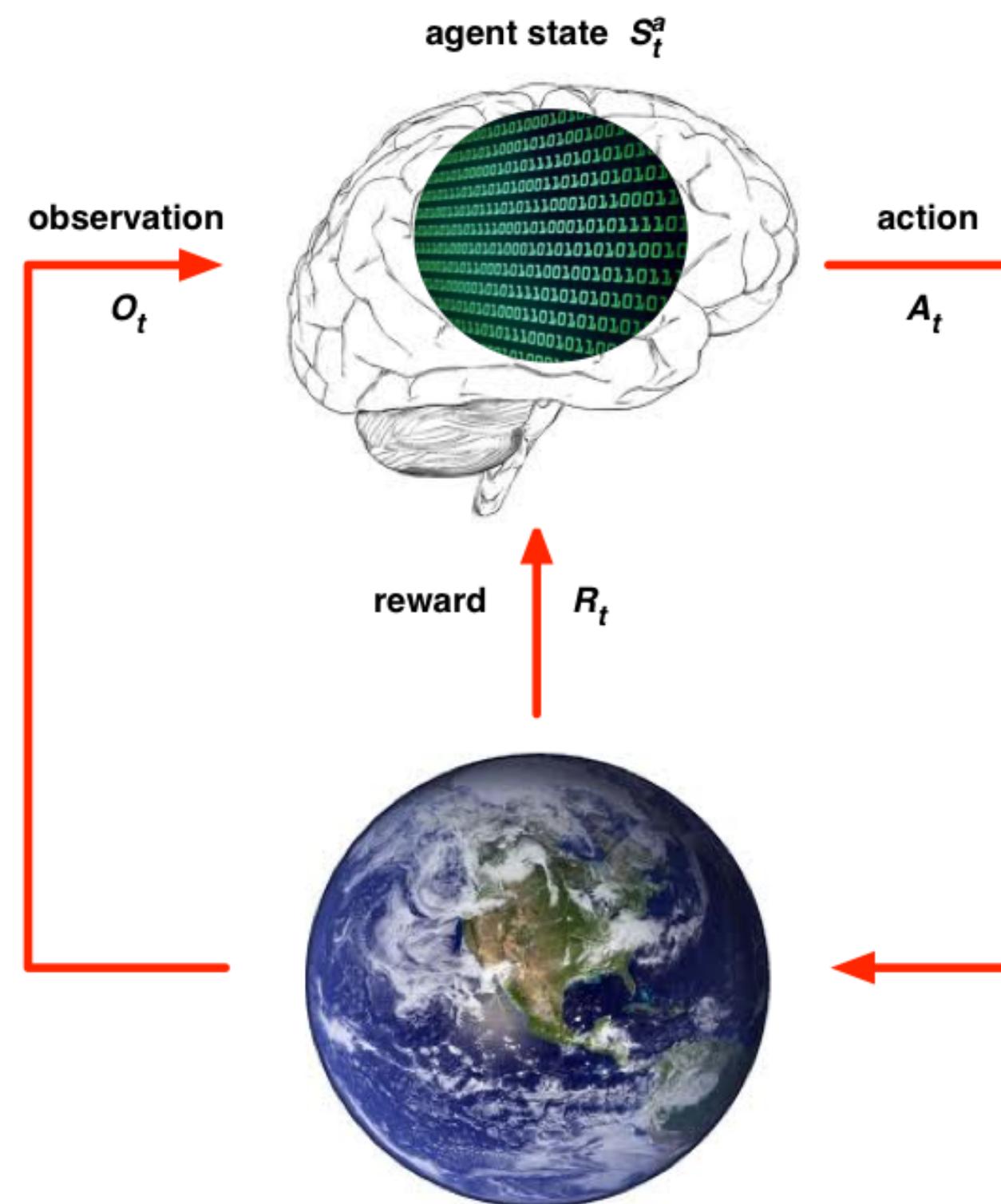
Environment State



- The **environment state** S_t^e is the environment's private representation
- i.e. whatever data the environment uses to pick the next observation/reward
- The environment state is not usually visible to the agent
- Even if S_t^e is visible, it may contain irrelevant information

Introduction [5/12]

Agent State

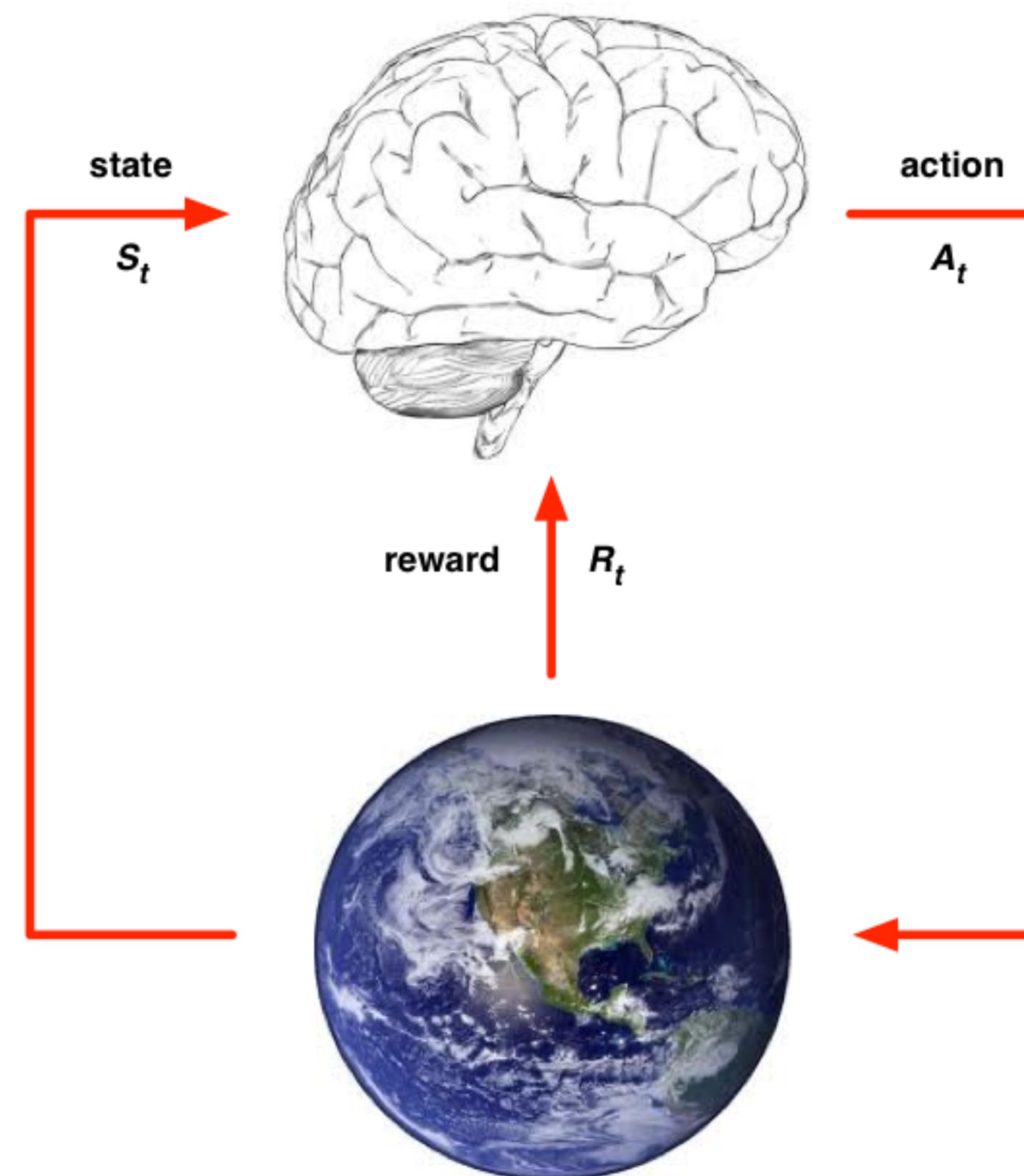


- The **agent state** S_t^a is the agent's internal representation
- i.e. whatever information the agent uses to pick the next action
- i.e. it is the information used by reinforcement learning algorithms
- It can be any function of history:

$$S_t^a = f(H_t)$$

Introduction [6/12]

Fully Observable Environments



Full observability: agent directly observes environment state

$$O_t = S_t^a = S_t^e$$

- Agent state = environment state = information state
- Formally, this is a **Markov decision process** (MDP)
- (Next lecture and the majority of this course)

Introduction [7/12]

Partially Observable Environments

- Partial observability: agent **indirectly observes** environment:
 - A robot with camera vision isn't told its absolute location
 - A trading agent only observes current prices
 - A poker playing agent only observes public cards
- Now agent state \neq environment state
- Formally this is a **partially observable Markov decision process** (POMDP)
- Agent must construct its own state representation S_t^a , e.g.
 - Complete history: $S_t^a = H_t$
 - **Beliefs** of environment state: $S_t^a = (\mathbb{P}[S_t^e = s^1], \dots, \mathbb{P}[S_t^e = s^n])$
 - Recurrent neural network: $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$

Introduction [8/12]

Major Components of an RL Agent

- An RL agent may include one or more of these components:
 - Policy: agent's behaviour function
 - Value function: how good is each state and/or action
 - Model: agent's representation of the environment

Introduction [9/12]

Policy

- A **policy** is the agent's behaviour
- It is a map from state to action, e.g.
- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

Introduction [10/12]

Value Function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

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

Introduction [11/12]

Model

- A **model** predicts what the environment will do next
- \mathcal{P} predicts the next state
- \mathcal{R} predicts the next (immediate) reward, e.g.
 - *State transition model* $\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
 - *Reward model* $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$

Introduction [12/12]

Learning and Planning

Two fundamental problems in sequential decision making

- Reinforcement Learning:
 - The environment is initially unknown
 - The agent interacts with the environment
 - The agent improves its policy
- Planning:
 - A model of the environment is known
 - The agent performs computations with its model (without any external interaction)
 - The agent improves its policy
 - a.k.a. deliberation, reasoning, introspection, pondering, thought, search

MDP [1/11]

Introduction to MDPs

- *Markov decision processes* formally describe an environment for reinforcement learning
- Where the environment is *fully observable*
- i.e. The current *state* completely characterises the process
- Almost all RL problems can be formalised as MDPs, e.g.
 - Optimal control primarily deals with continuous MDPs
 - Partially observable problems can be converted into MDPs
 - Bandits are MDPs with one state

MDP [2/11]

Markov Property

“The future is independent of the past given the present”

Definition

A state S_t is *Markov* if and only if

$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \dots, S_t]$$

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future

MDP [3/11]

Markov Process

A Markov process is a memoryless random process, i.e. a sequence of random states S_1, S_2, \dots with the Markov property.

Definition

A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- \mathcal{S} is a (finite) set of states
- \mathcal{P} is a state transition probability matrix,
$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

MDP [4/11]

State Transition Matrix

For a Markov state s and successor state s' , the *state transition probability* is defined by

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

State transition matrix \mathcal{P} defines transition probabilities from all states s to all successor states s' ,

The Markov assumption:

The future is conditionally independent of the past, given the present. In Markov Decision problems we assume that successor states are conditionally independent of all states and actions that took place prior to the last state. This assumption is wrong. But useful.

$$\mathcal{P} = \text{from} \begin{matrix} & \text{to} \\ \left[\begin{matrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{matrix} \right] & n \text{ by } n \end{matrix}$$

where each row of the matrix sums to 1.

MDP [5/11]

Markov Reward Process : MP + Reward

A Markov reward process is a Markov chain with values.

Definition

A *Markov Reward Process* is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- \mathcal{S} is a finite set of states
- \mathcal{P} is a state transition probability matrix,
 $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$
- \mathcal{R} is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

MDP [6/11]

Returns

Definition

The *return* G_t is the total discounted reward from time-step t .

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
 - γ close to 0 leads to "myopic" evaluation
 - γ close to 1 leads to "far-sighted" evaluation

MDP [7/11]

Discount Factor

Most Markov reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behaviour shows preference for immediate reward
- It is sometimes possible to use *undiscounted* Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.

MDP [8/11]

Value Function of MRP

The value function $v(s)$ gives the long-term value of state s

Definition

The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E} [G_t \mid S_t = s]$$

MDP [9/11]

Markov Decision Process

A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Definition

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- \mathcal{S} is a finite set of states
- \mathcal{A} is a finite set of actions
- \mathcal{P} is a state transition probability matrix,
$$\mathcal{P}_{ss'}^{\textcolor{red}{a}} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = \textcolor{red}{a}]$$
- \mathcal{R} is a reward function, $\mathcal{R}_s^{\textcolor{red}{a}} = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = \textcolor{red}{a}]$
- γ is a discount factor $\gamma \in [0, 1]$.

MDP [10/11]

Policy

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent),
 $A_t \sim \pi(\cdot|S_t), \forall t > 0$

MDP [11/11]

Value Functions of MDP

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s , and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$$

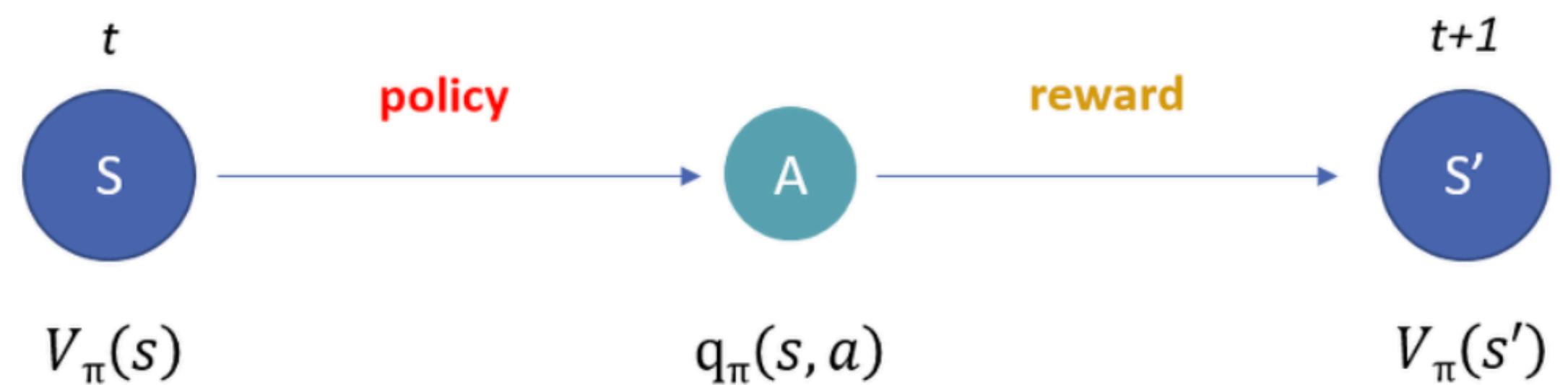
Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s , taking action a , and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a]$$

Bellman Equation [1/5]

Value Functions of MDP



The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

$$v_\pi(s) = \mathbb{E}_\pi [R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

The action-value function can similarly be decomposed,

$$q_\pi(s, a) = \mathbb{E}_\pi [R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

Bellman Equation [2/5]

State-Value Function

$$v_{\pi}(s) = \mathbb{E}_{\pi} [G_t \mid S_t = s]$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$V_{\pi}(s) = E_{\pi} [G_t \mid S_t = s]$$

$$= E_{\pi} [R_{t+1} + \underline{\gamma R_{t+2}} + \gamma^2 R_{t+3} + \dots \mid S_t = s]$$

$$= E_{\pi} [R_{t+1} + \underline{\gamma(R_{t+2} + \gamma R_{t+3} + \dots)} \mid S_t = s]$$

$$= E_{\pi} [R_{t+1} + \gamma \underline{(G_{t+1})} \mid S_t = s]$$

$$= E_{\pi} [\underline{R_{t+1}} + \gamma \underline{V(S_{t+1})} \mid S_t = s] \quad \longrightarrow \text{구현 어떻게?}$$

①

②

Bellman Equation [3/5]

State-Value Function (Cont.)

$$V_{\pi}(s) = E_{\pi} [G_t \mid S_t = s]$$

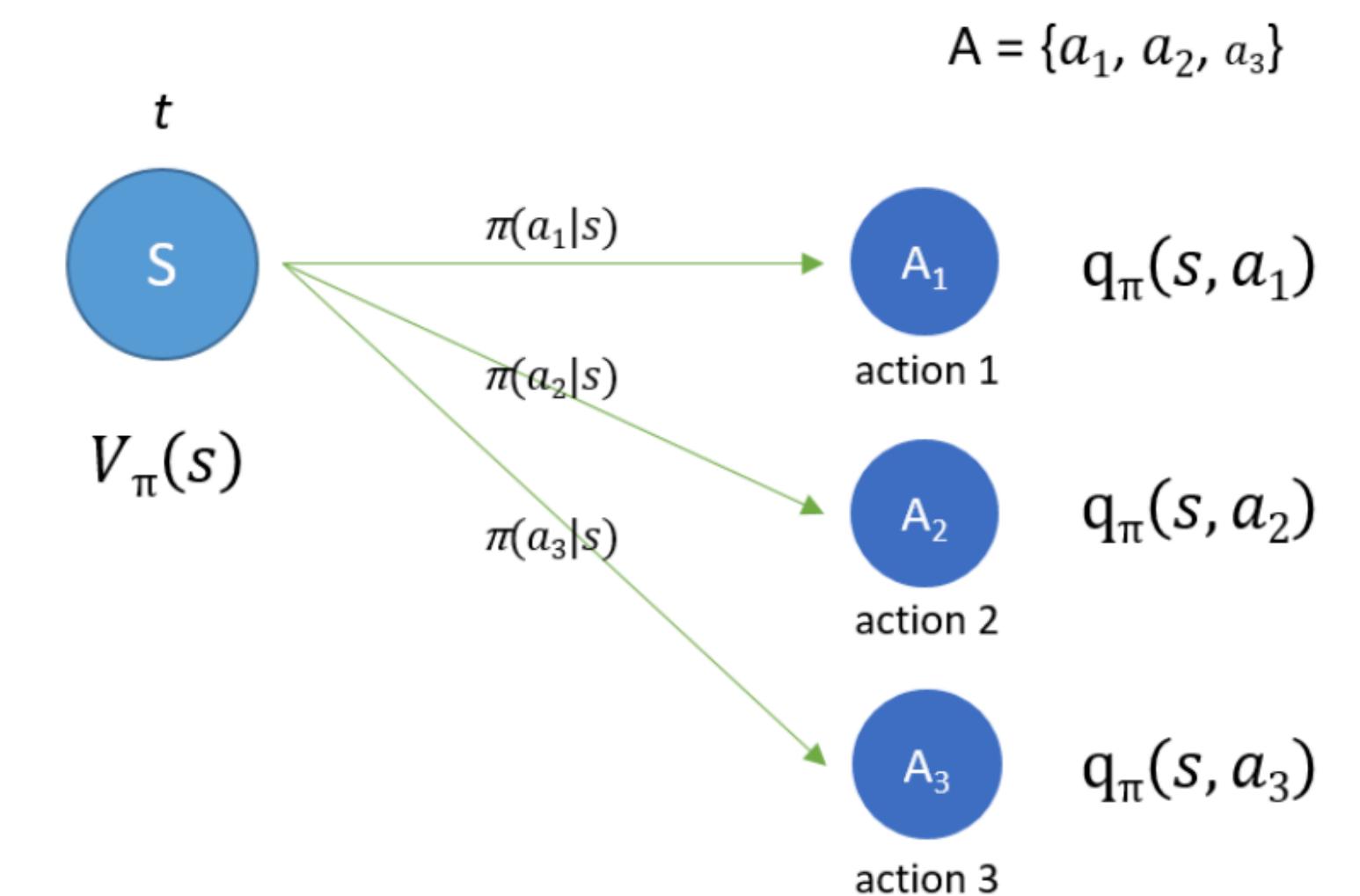
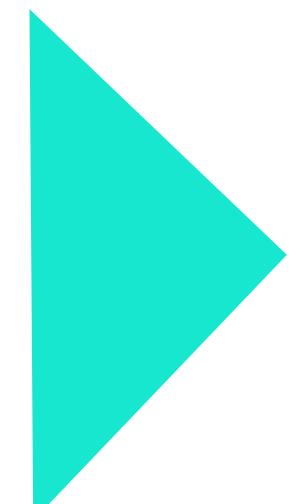
$$= E_{\pi} [R_{t+1} + \underline{\gamma R_{t+2} + \gamma^2 R_{t+3} + \dots} \mid S_t = s]$$

$$= E_{\pi} [R_{t+1} + \underline{\gamma(R_{t+2} + \gamma R_{t+3} + \dots)} \mid S_t = s]$$

$$= E_{\pi} [R_{t+1} + \underline{\gamma(G_{t+1})} \mid S_t = s]$$

$$= E_{\pi} [R_{t+1} + \underline{\gamma V(S_{t+1})} \mid S_t = s]$$

① ②

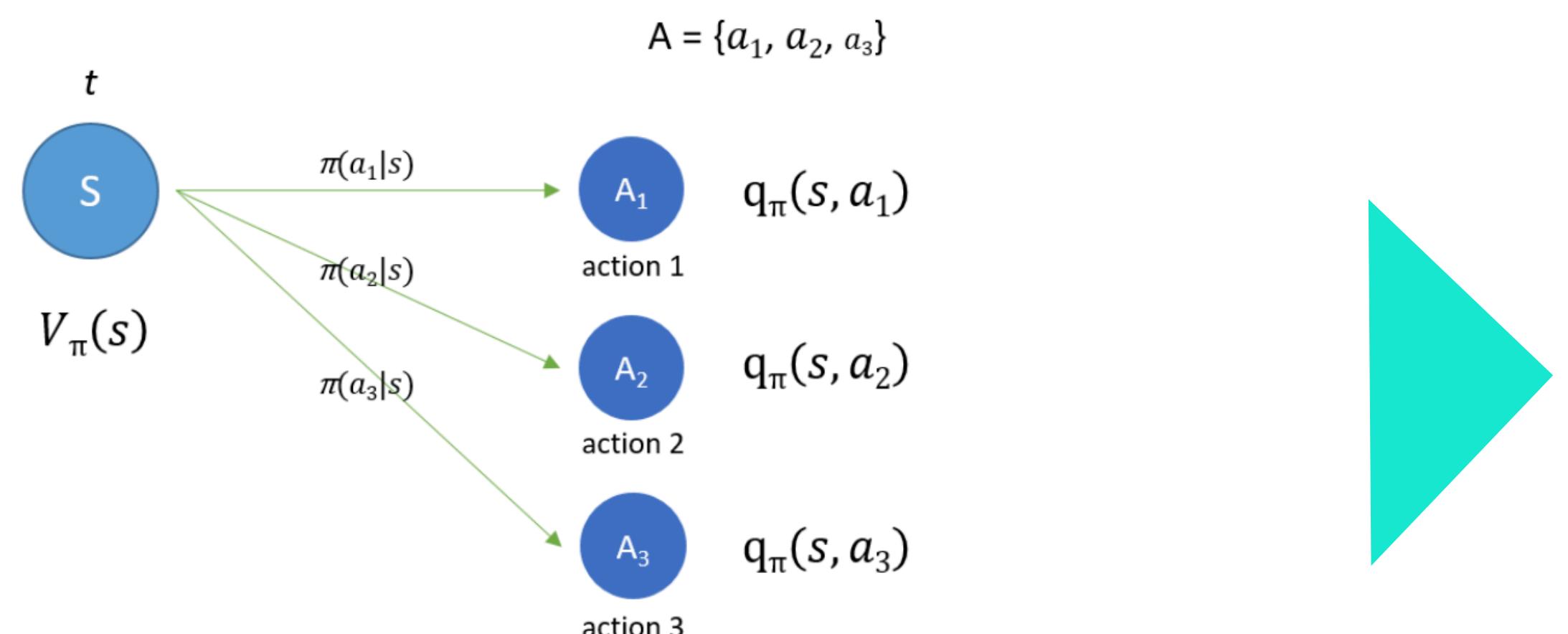
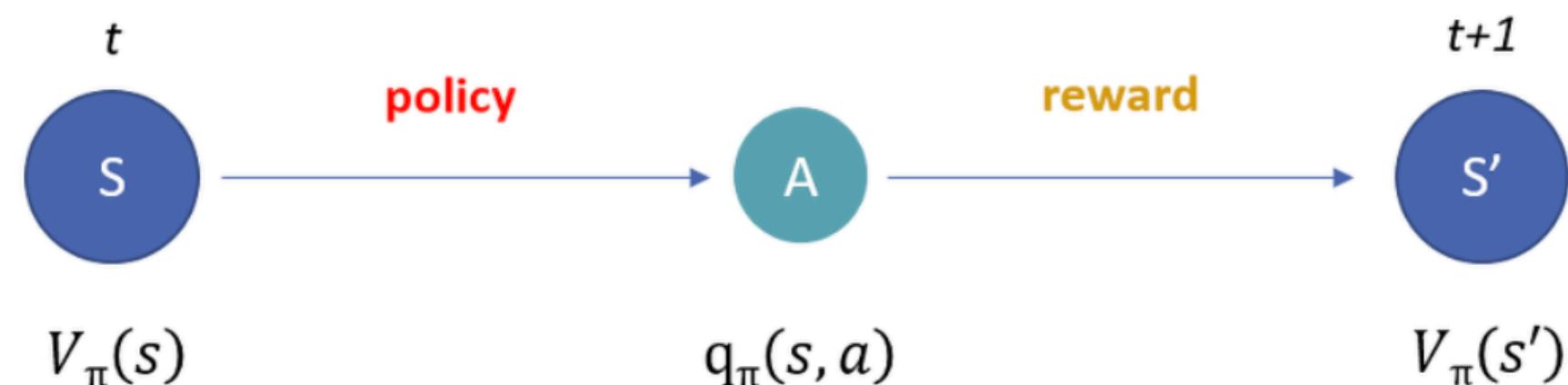


* Policy : mapping func. ($s \rightarrow a$)

$$V_{\pi}(s) = \sum_{a \in A} \pi(a|s) * q_{\pi}(s, a)$$

Bellman Equation [4/5]

Action-Value Function



* Policy : mapping func. ($s \rightarrow a$)

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) * q_\pi(s, a)$$



* Transition Prob. : mapping func. ($s, a \rightarrow s'$)

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma * \sum_{s' \in S} \mathcal{P}_{ss'}^a * V_\pi(s')$$

Bellman Equation [5/5]

Bellman Expectation Equation

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

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

Bellman Optimal Equation

Dynamic Programming [1/5]

Learning and Planning

Two fundamental problems in sequential decision making

- Reinforcement Learning:
 - The environment is initially unknown
 - The agent interacts with the environment
 - The agent improves its policy
- Planning:
 - A model of the environment is known $\xrightarrow{\hspace{1cm}}$ $\mathcal{P}_{ss'}^a, \mathcal{R}_s^a$
 - The agent performs computations with its model (without any external interaction)
 - The agent improves its policy
 - a.k.a. deliberation, reasoning, introspection, pondering, thought, search

Dynamic Programming [2/5]

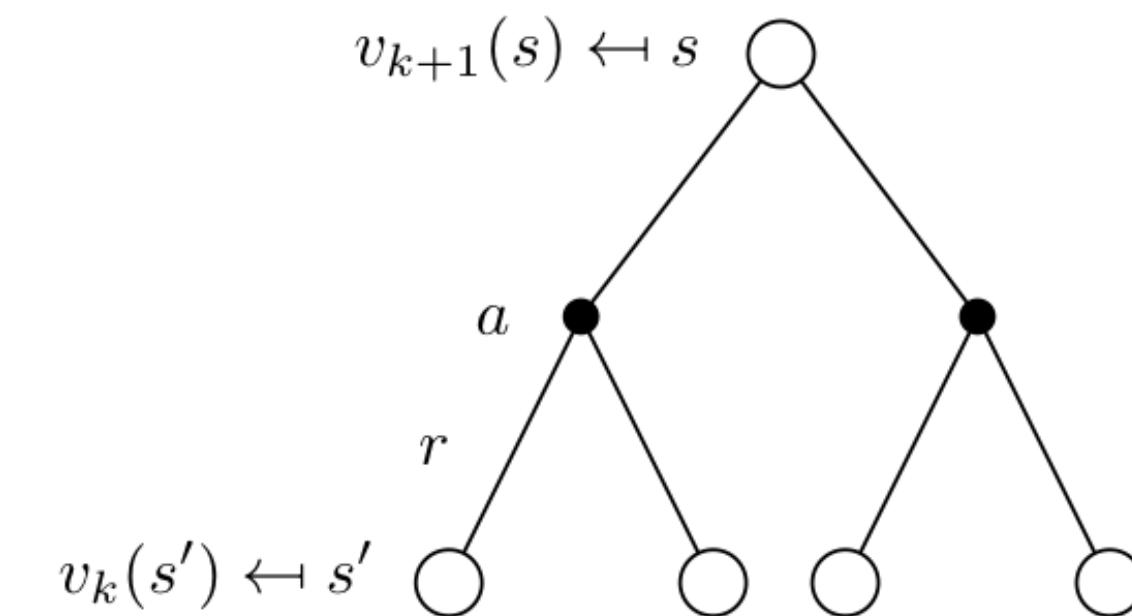
Planning by DP

- Dynamic programming assumes full knowledge of the MDP
- It is used for *planning* in an MDP
- For prediction:
 - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
 - or: MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
 - Output: value function v_π
- Or for control:
 - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
 - Output: optimal value function v_*
 - and: optimal policy π_*

Dynamic Programming [3/5]

1) Prediction : Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_\pi$
- Using *synchronous* backups,
 - At each iteration $k + 1$
 - For all states $s \in \mathcal{S}$
 - Update $v_{k+1}(s)$ from $v_k(s')$
 - where s' is a successor state of s
- We will discuss *asynchronous* backups later
- Convergence to v_π will be proven at the end of the lecture



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$
$$\mathbf{v}^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}^k$$

Dynamic Programming [4/5]

2) Control : Policy improvement

- Given a policy π
 - Evaluate the policy π

$$v_\pi(s) = \mathbb{E} [R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$$

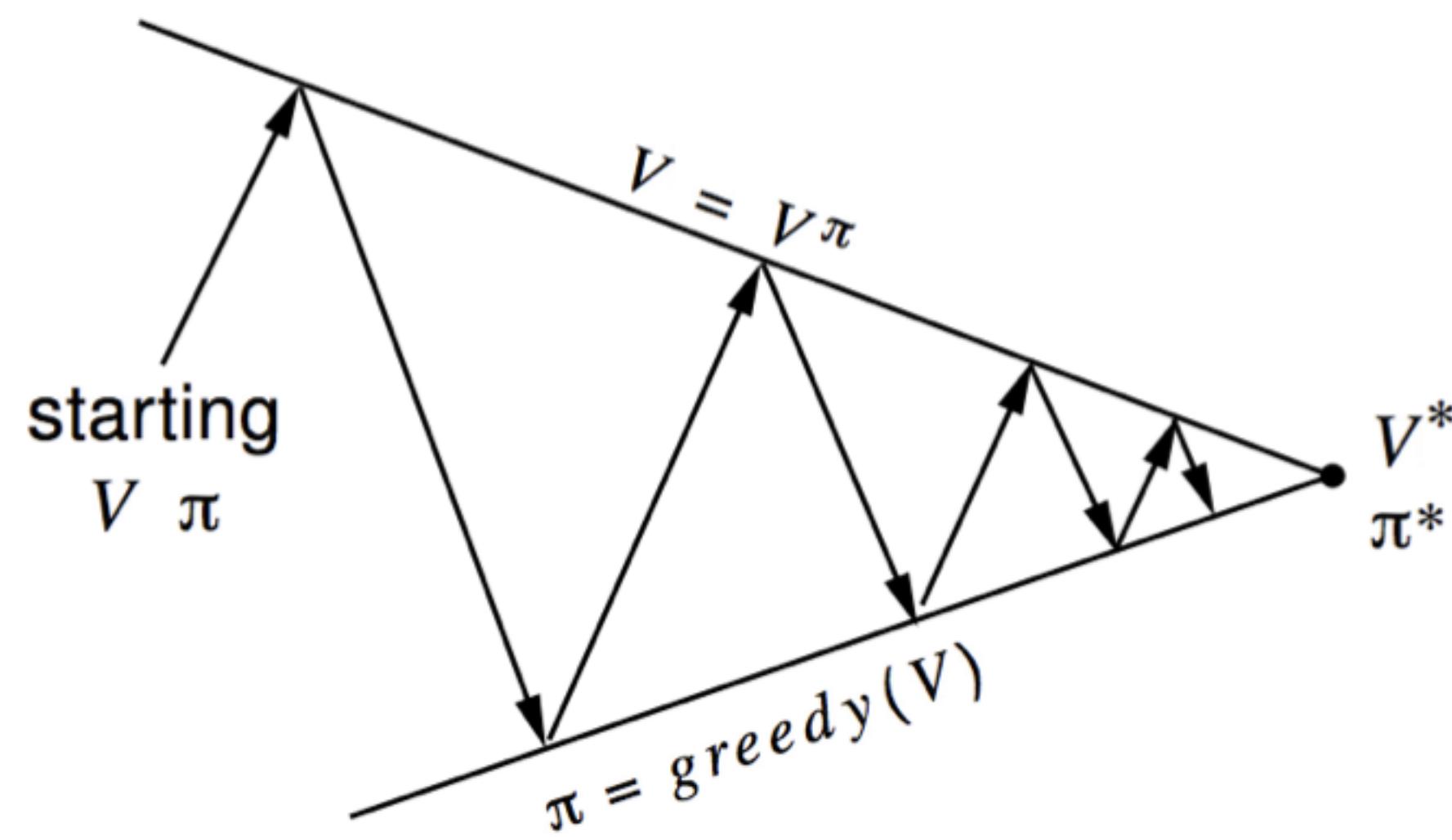
- Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$

- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of **policy iteration** always converges to π^*

Dynamic Programming [5/5]

Policy Iteration

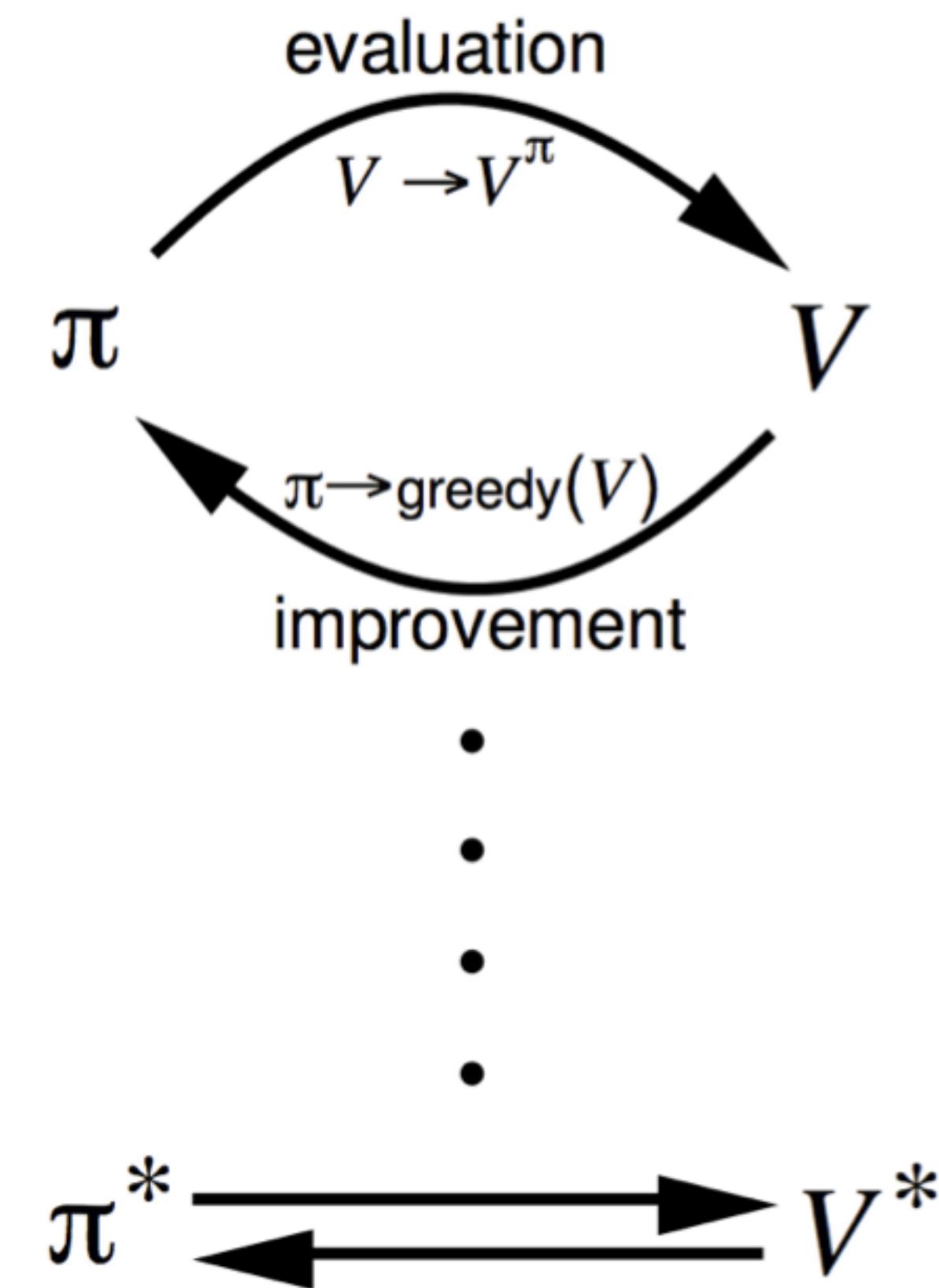


Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement



Model-Free Prediction [1/10]

Monte Carlo RL

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of MDP transitions / rewards
- MC learns from *complete* episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to *episodic* MDPs
 - All episodes must terminate

Model-Free Prediction [2/10]

Monte Carlo Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$$

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

Model-Free Prediction [3/10]

Monte Carlo Policy Evaluation :

First-visit MCPE

- To evaluate state s
- The **first** time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Every-Visit MCPE

- To evaluate state s
- **Every** time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- Again, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Model-Free Prediction [4/10]

Incremental Mean

The mean μ_1, μ_2, \dots of a sequence x_1, x_2, \dots can be computed incrementally,

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} \left(x_k + \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})\end{aligned}$$

Model-Free Prediction [5/10]

Incremental MC Updates

- Update $V(s)$ incrementally after episode $S_1, A_1, R_2, \dots, S_T$
- For each state S_t with return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

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

Model-Free Prediction [6/10]

Temporal-Difference Learning

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards
- TD learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

Model-Free Prediction [7/10]

MC vs. TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
 - Update value $V(S_t)$ toward *actual* return G_t

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

- Simplest temporal-difference learning algorithm: TD(0)
 - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

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

- $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

Model-Free Prediction [8/10]

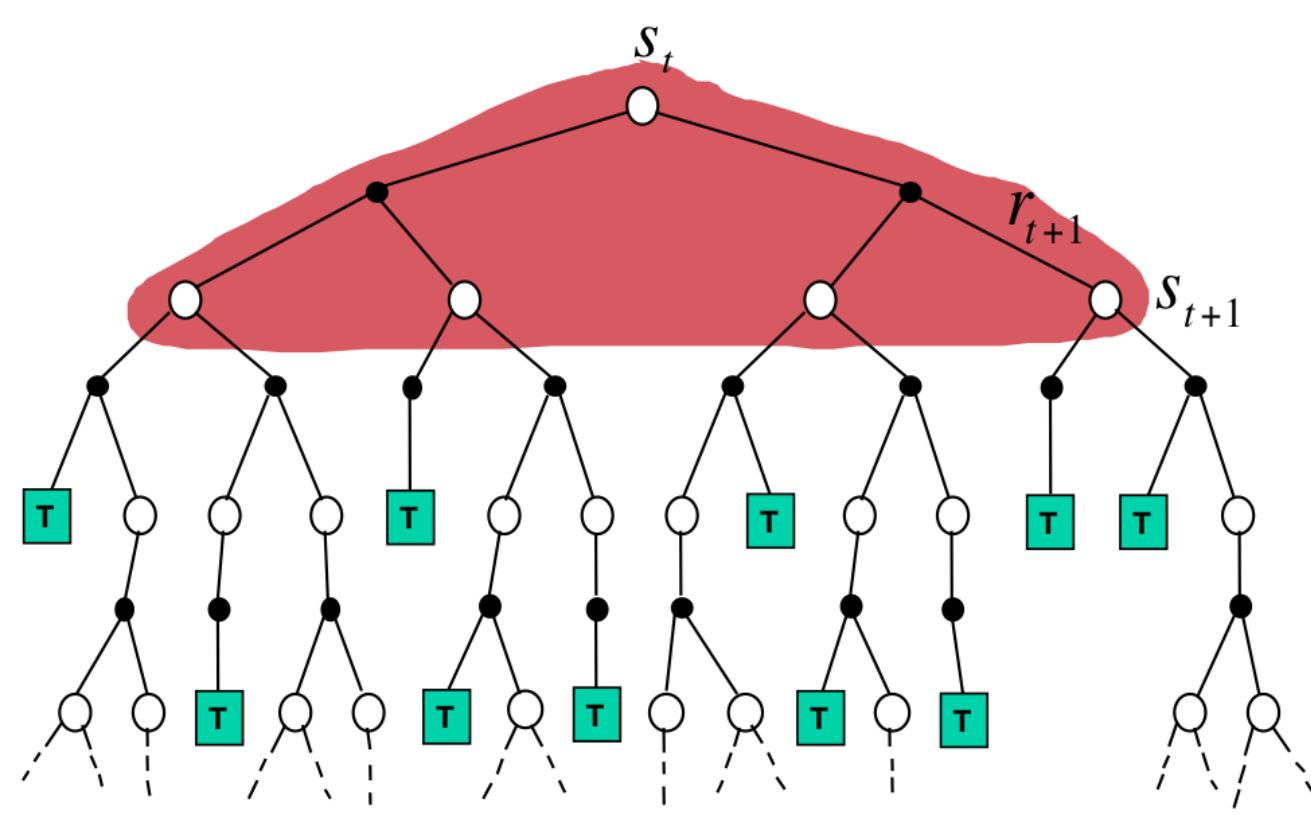
Bias / Variance Trade-Off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$ is *unbiased* estimate of $v_\pi(S_t)$
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is *unbiased* estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is *biased* estimate of $v_\pi(S_t)$
- TD target is much lower variance than the return:
 - Return depends on *many* random actions, transitions, rewards
 - TD target depends on *one* random action, transition, reward

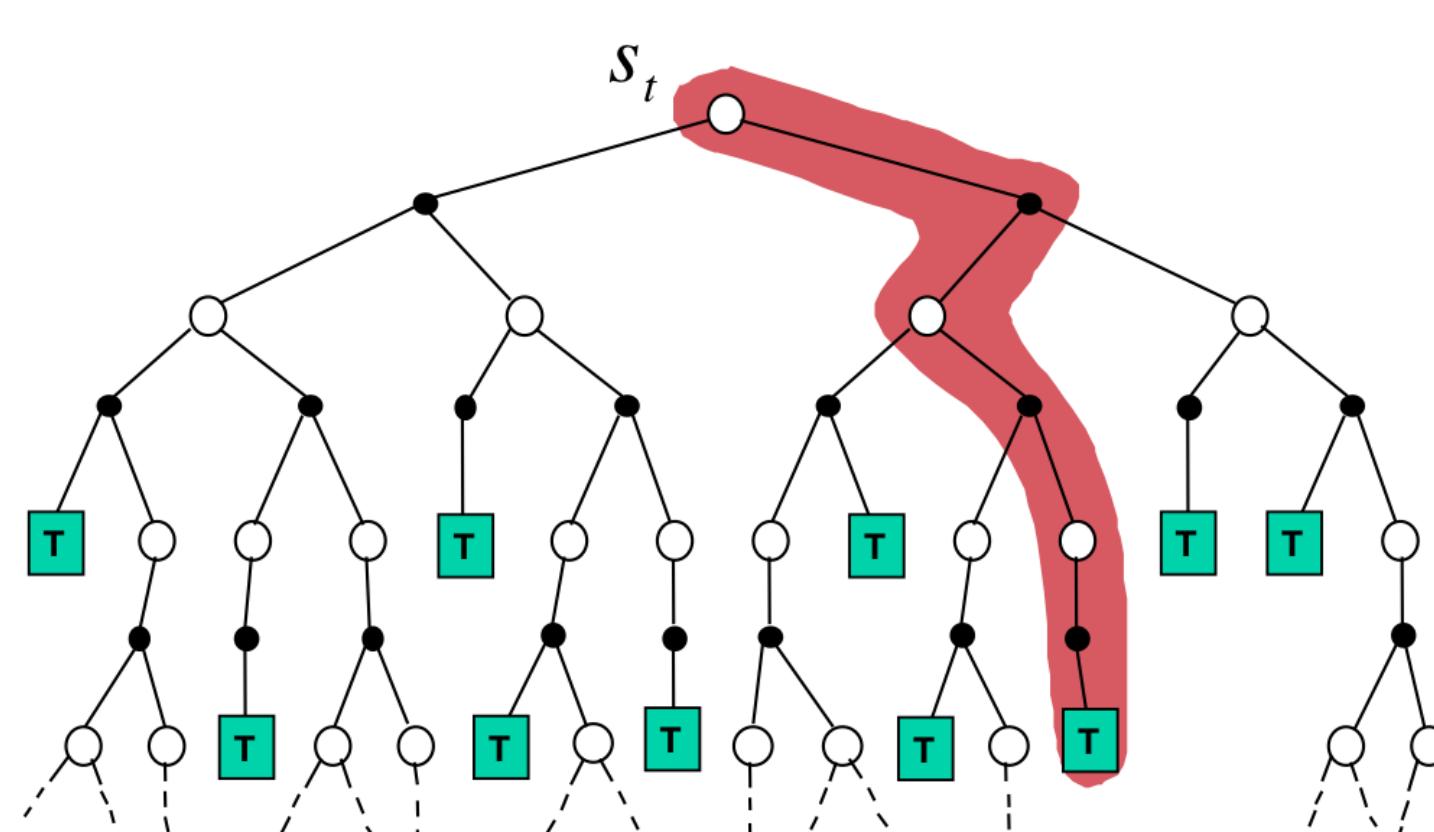
Model-Free Prediction [9/10]

Comparison to Backup

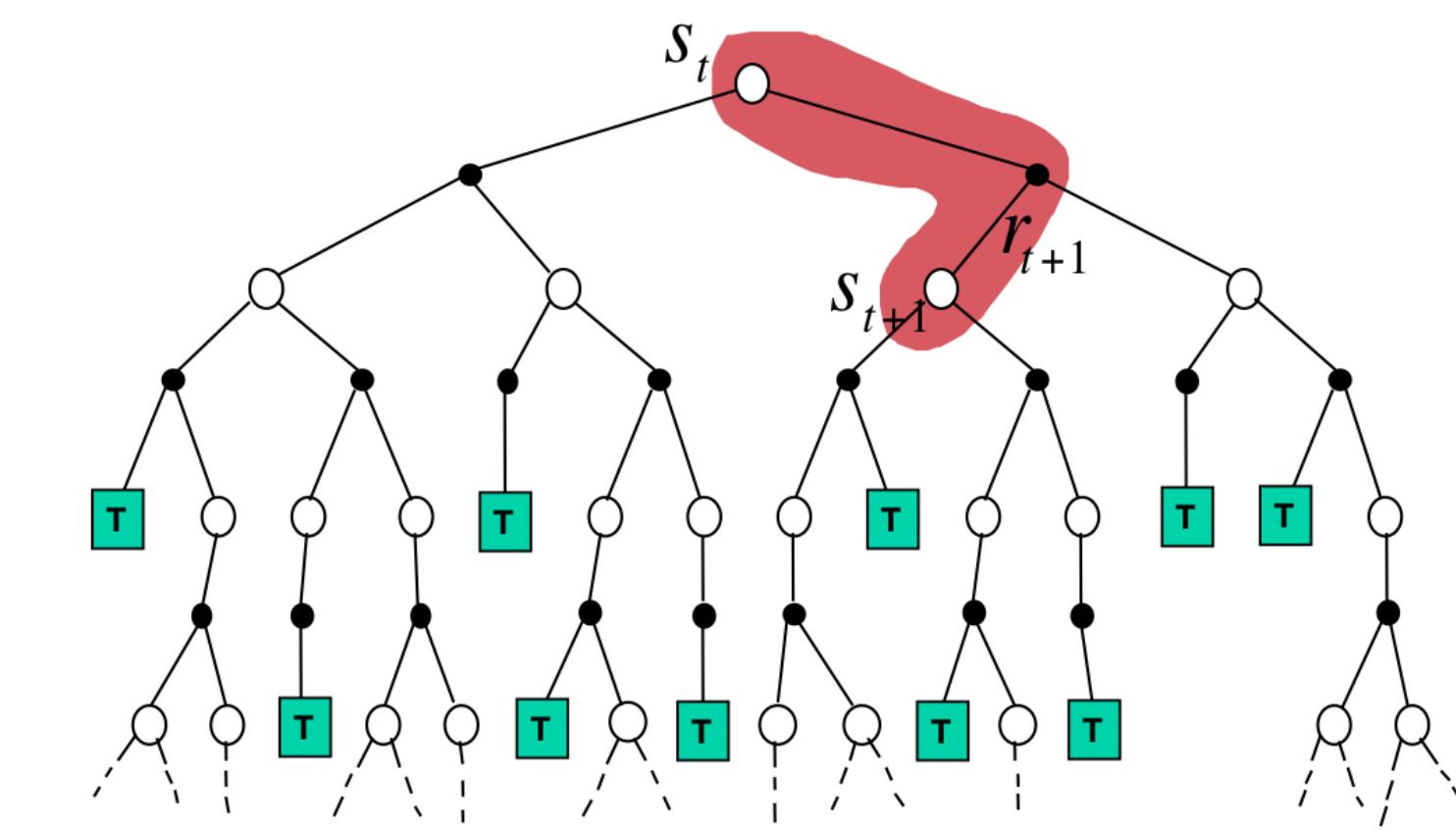
$$V(S_t) \leftarrow \mathbb{E}_\pi [R_{t+1} + \gamma V(S_{t+1})]$$



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

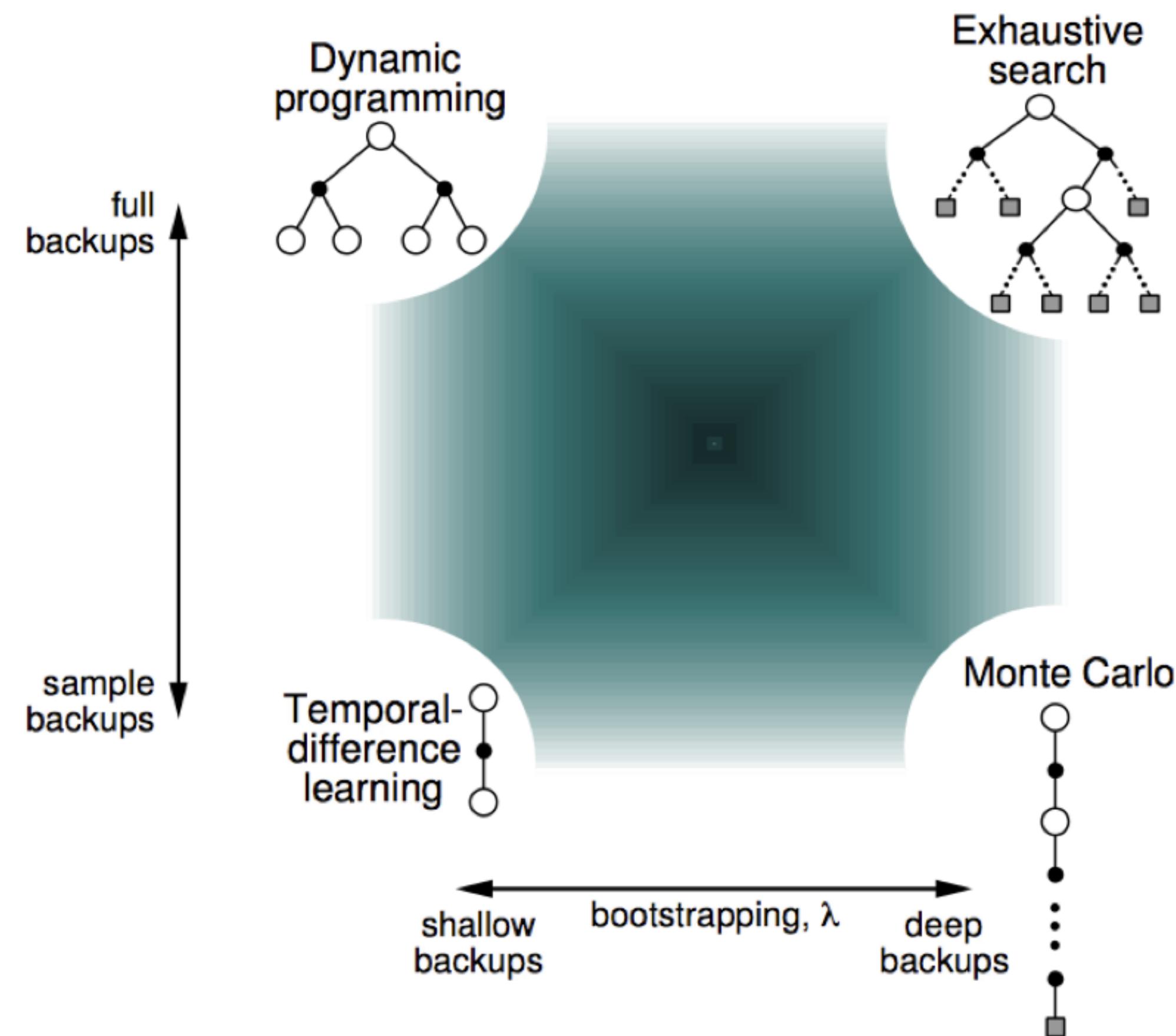


$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$



Model-Free Prediction [10/10]

Unified View of RL



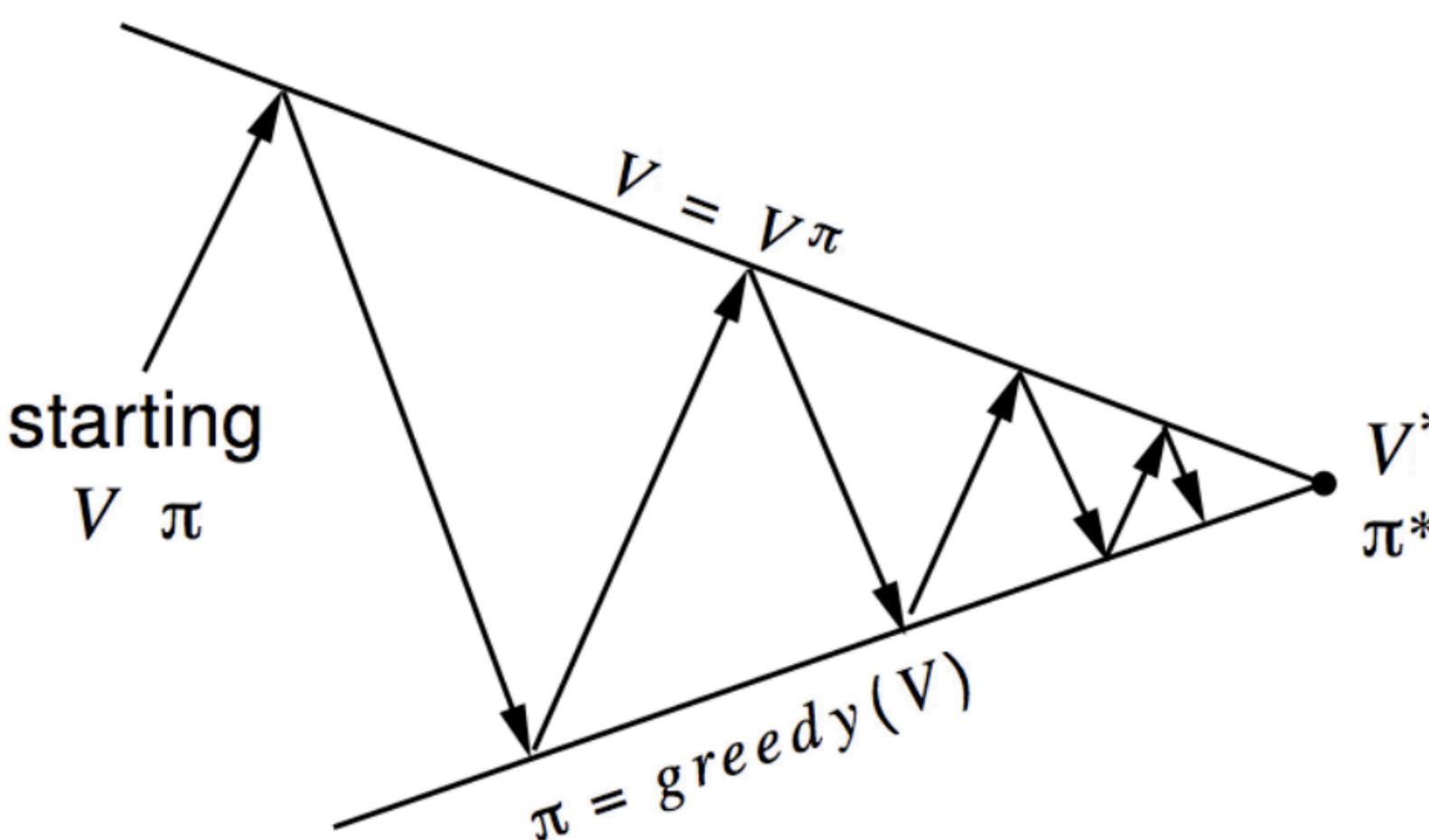
Model-Free Control [1/17]

On-Policy / Off-Policy Learning

- **On-policy** learning
 - “Learn on the job”
 - Learn about policy π from experience sampled from π
- **Off-policy** learning
 - “Look over someone’s shoulder”
 - Learn about policy π from experience sampled from μ

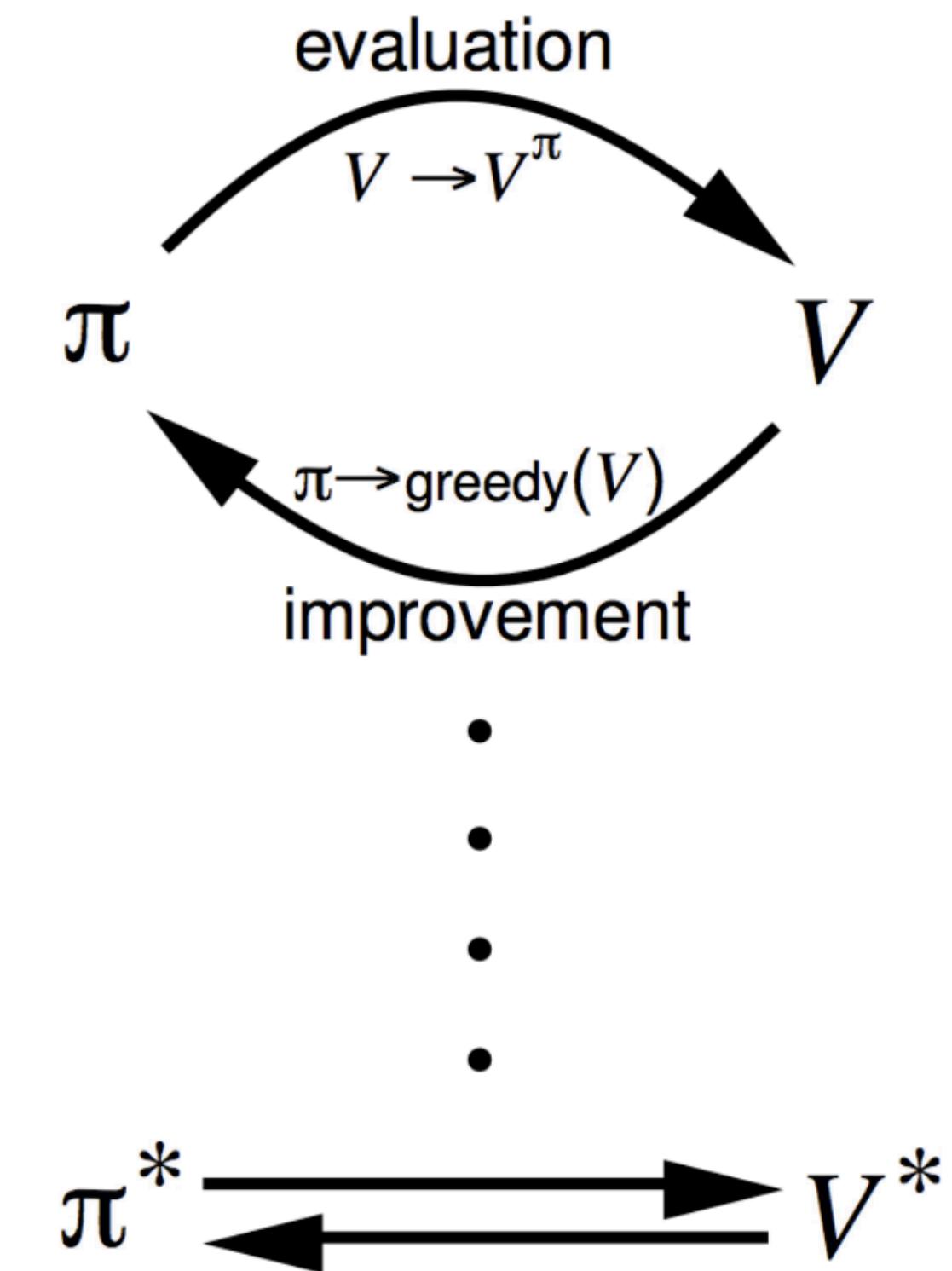
Model-Free Control [2/17]

Generalized Policy Iteration (Remind)



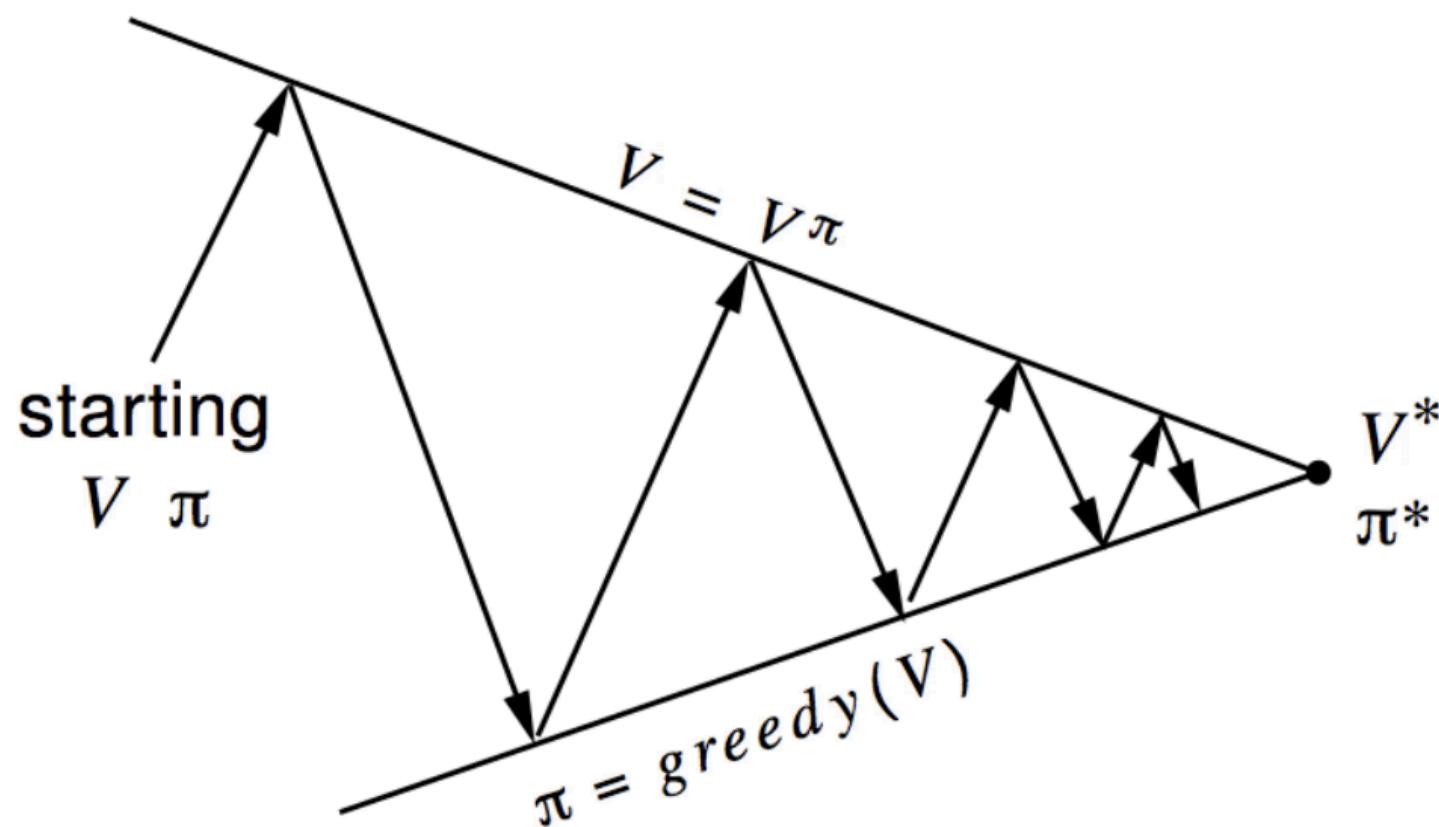
Policy evaluation Estimate v_π
e.g. Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
e.g. Greedy policy improvement



Model-Free Control [3/17]

Model-Free Policy Evaluation with MC Prediction



Policy evaluation Monte-Carlo policy evaluation, $V = v_\pi$?

Policy improvement Greedy policy improvement?

- Greedy policy improvement over $V(s)$ requires model of MDP

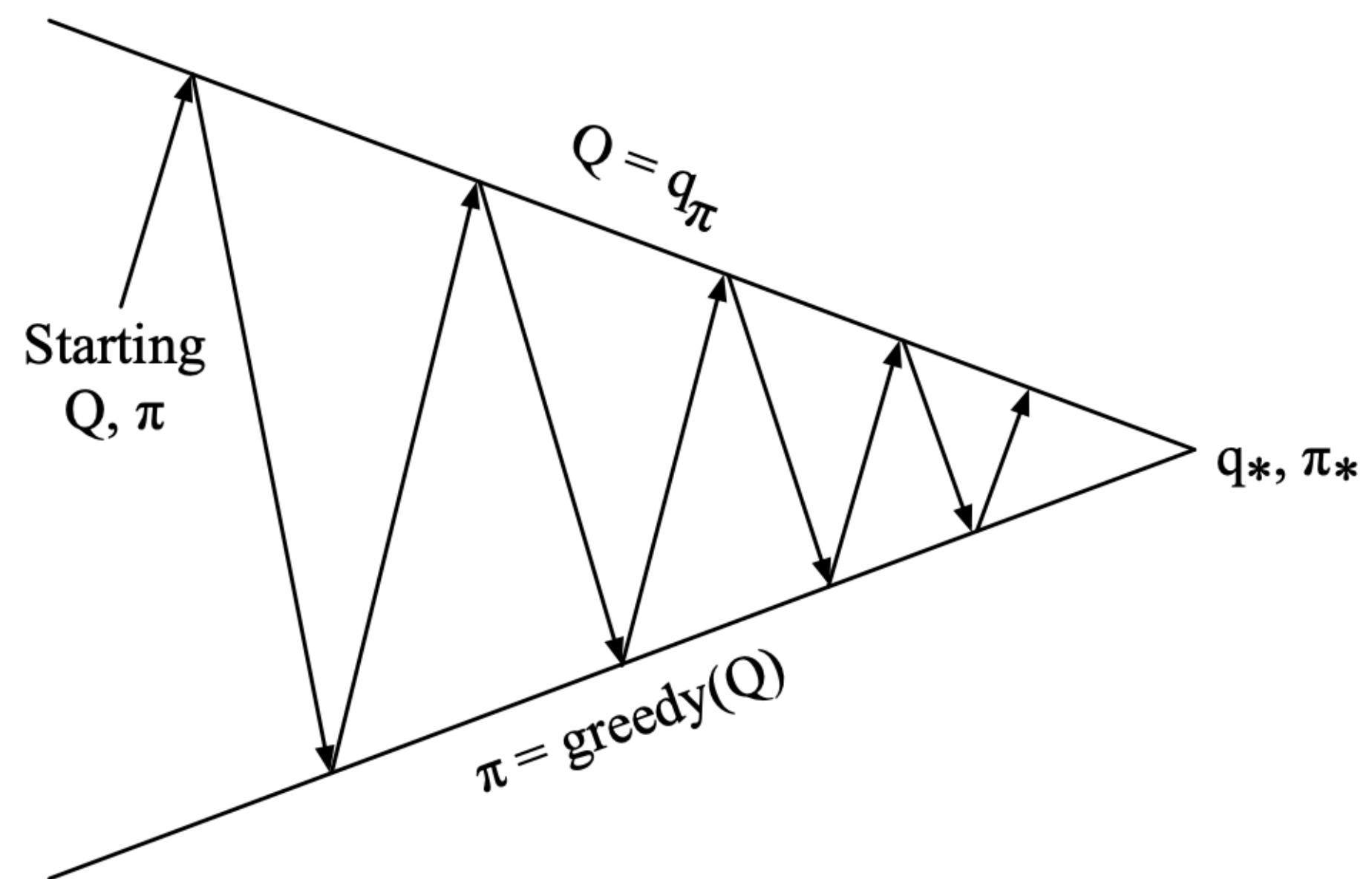
$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- Greedy policy improvement over $Q(s, a)$ is model-free

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

Model-Free Control [4/17]

Model-Free Policy Evaluation using Action-Value function



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement Greedy policy improvement?

Model-Free Control [5/17]

Epsilon-Greedy Exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Model-Free Control [6/17]

Epsilon-Greedy Policy Improvement

Theorem

For any ϵ -greedy policy π , the ϵ -greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s, a) \\ &= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} q_\pi(s, a) \\ &\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/m}{1 - \epsilon} q_\pi(s, a) \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) = v_\pi(s) \end{aligned}$$

Therefore from policy improvement theorem, $v_{\pi'}(s) \geq v_\pi(s)$

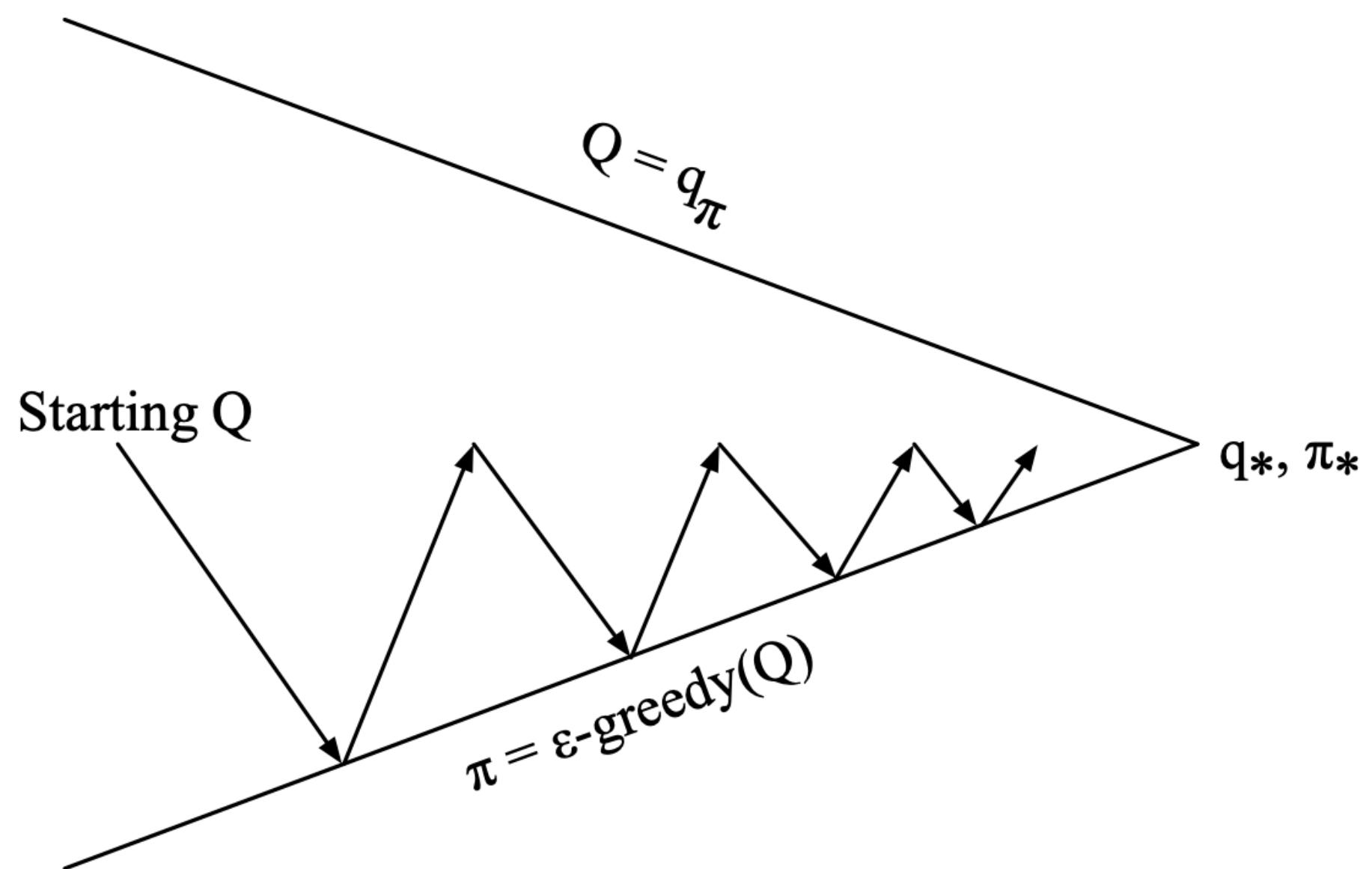
Model-Free Control [7/17]

MC vs. TD Control

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
 - Lower variance
 - Online
 - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
 - Apply TD to $Q(S, A)$
 - Use ϵ -greedy policy improvement
 - Update every time-step

Model-Free Control [8/17]

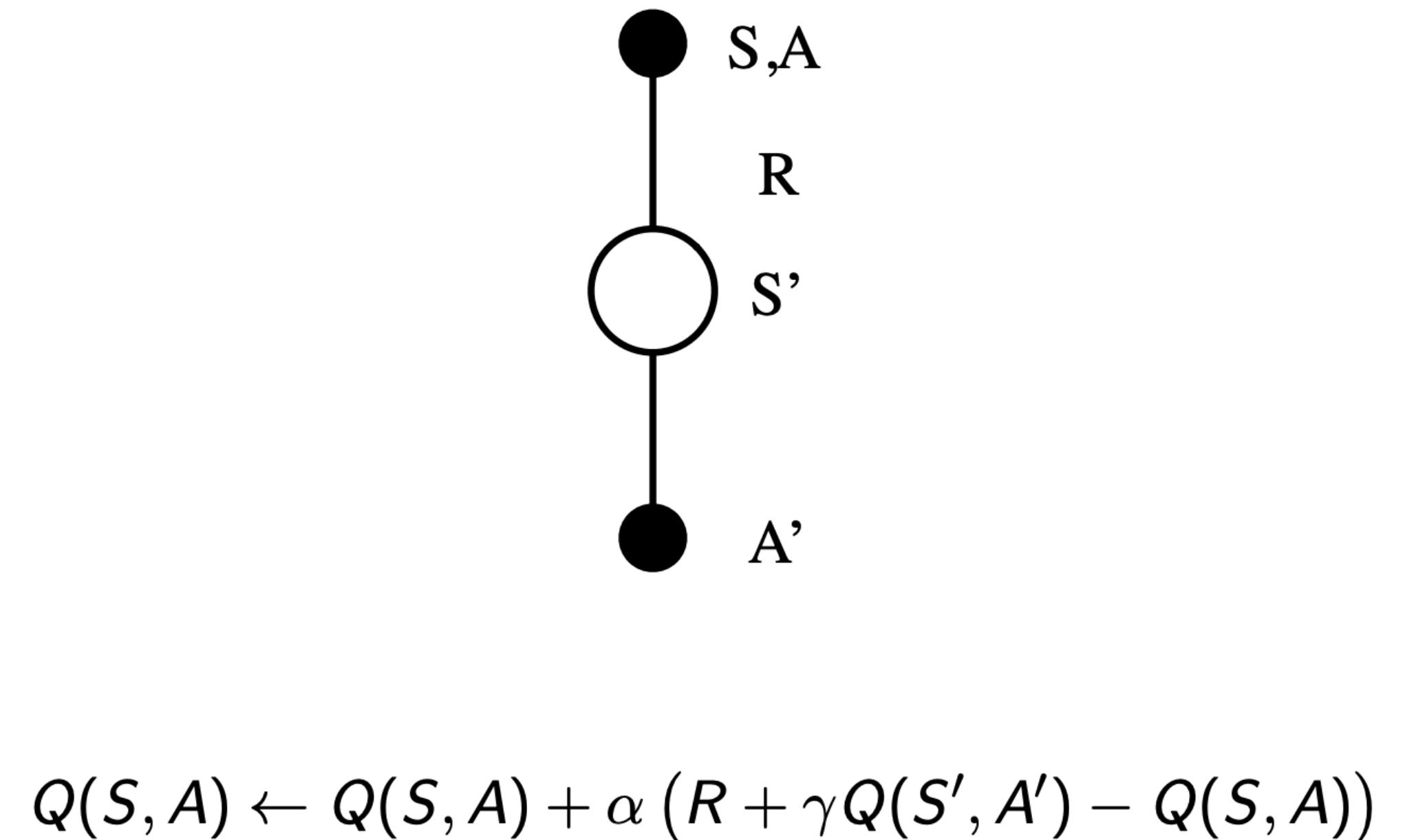
On-Policy Control with Sarsa



Every time-step:

Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement ϵ -greedy policy improvement



Model-Free Control [9/17]

Sarsa Algorithm for On-Policy

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$
Repeat (for each episode):

 Initialize S

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Repeat (for each step of episode):

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ε -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$$

$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal

Model-Free Control [10/17]

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$
- While following behaviour policy $\mu(a|s)$

$$\{S_1, A_1, R_2, \dots, S_T\} \sim \mu$$

- Why is this important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, \dots, \pi_{t-1}$
- Learn about *optimal* policy while following *exploratory* policy
- Learn about *multiple* policies while following *one* policy

Model-Free Control [11/17]

Off-Policy Learning : (1) Importance sampling

- Estimate the expectation of a different distribution

$$\begin{aligned}\mathbb{E}_{X \sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X) \frac{P(X)}{Q(X)} f(X) \\ &= \mathbb{E}_{X \sim Q} \left[\frac{P(X)}{Q(X)} f(X) \right]\end{aligned}$$

Model-Free Control [12/17]

Off-Policy Learning : (1) Importance sampling with Off-Policy MC

- Use returns generated from μ to evaluate π
- Weight return G_t according to similarity between policies
- Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \cdots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

- Update value towards *corrected* return

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{\pi/\mu} - V(S_t) \right)$$

- Cannot use if μ is zero when π is non-zero
- Importance sampling can dramatically increase variance

Model-Free Control [13/17]

Off-Policy Learning : (1) Importance sampling with Off-Policy TD

- Use TD targets generated from μ to evaluate π
- Weight TD target $R + \gamma V(S')$ by importance sampling
- Only need a single importance sampling correction

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} (R_{t+1} + \gamma V(S_{t+1})) - V(S_t) \right)$$

- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

Model-Free Control [14/17]

Off-Policy Learning : (2) Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$
- **No** importance sampling is required
- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot | S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot | S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

Model-Free Control [15/17]

Off-Policy Control with Q-Learning

- We now allow both behaviour and target policies to **improve**
- The target policy π is **greedy** w.r.t. $Q(s, a)$

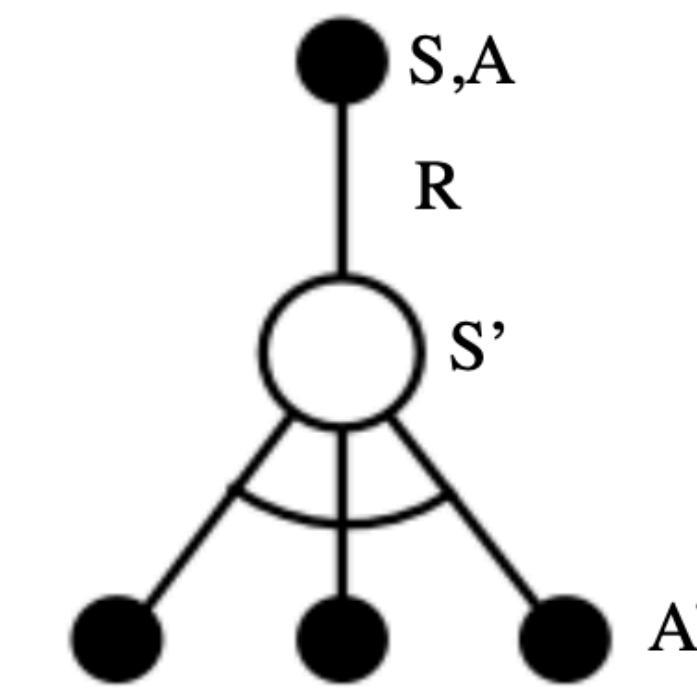
$$\pi(S_{t+1}) = \operatorname{argmax}_{a'} Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. **ϵ -greedy** w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ &= R_{t+1} + \gamma Q(S_{t+1}, \operatorname{argmax}_{a'} Q(S_{t+1}, a')) \\ &= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

Model-Free Control [16/17]

Q-Learning Algorithm



$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

Theorem

*Q-learning control converges to the optimal action-value function,
 $Q(s, a) \rightarrow q_*(s, a)$*

Model-Free Control [17/17]

Q-Learning Algorithm for Off-Policy Control

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Take action A , observe R, S'

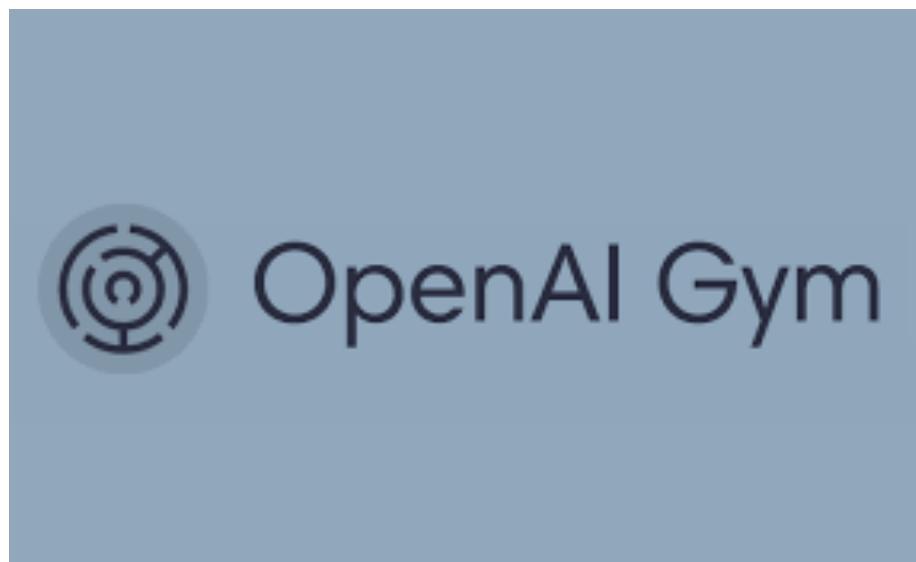
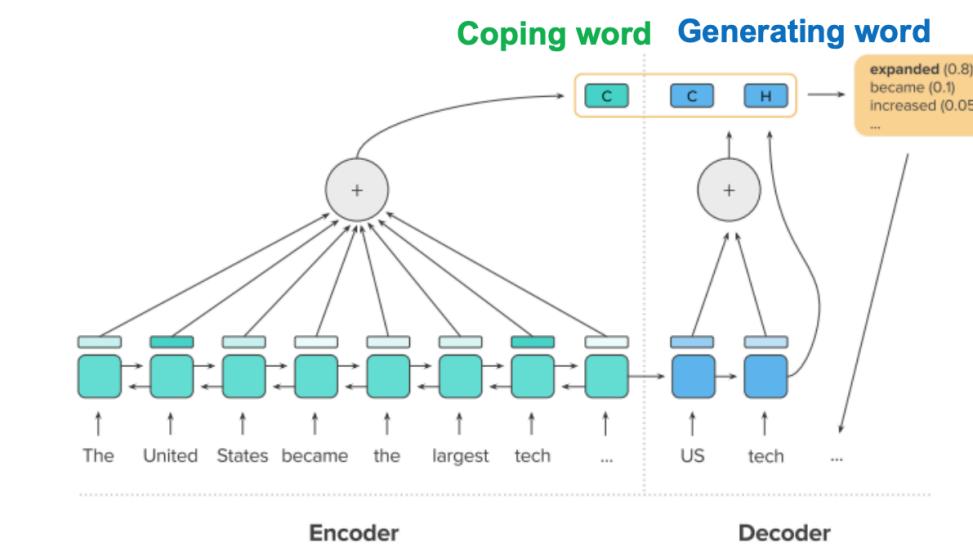
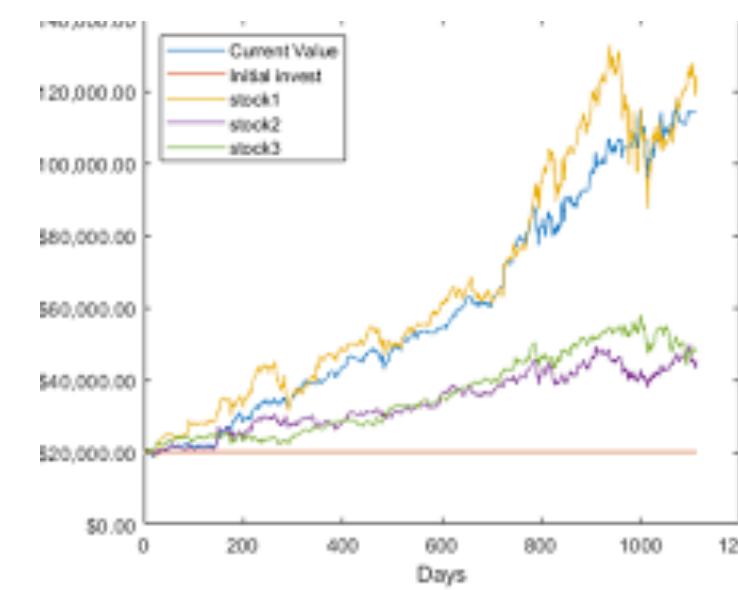
$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$;

 until S is terminal

RL application [1/3]

DeepMind



RL application [2/3]

DeepMind

DeepMind > Blog > DeepMind AI Reduces Google Data Centre Cooling Bill by 40%



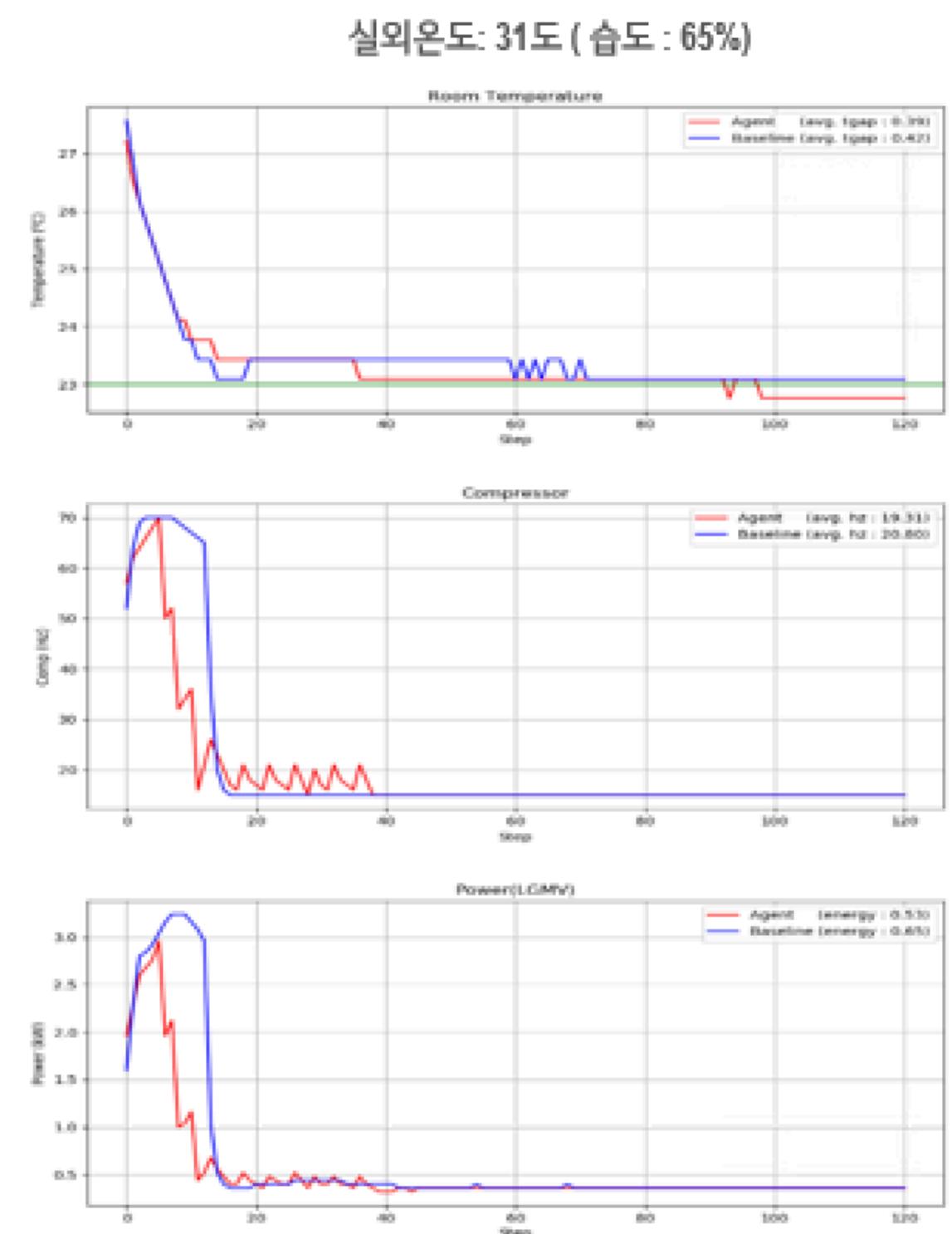
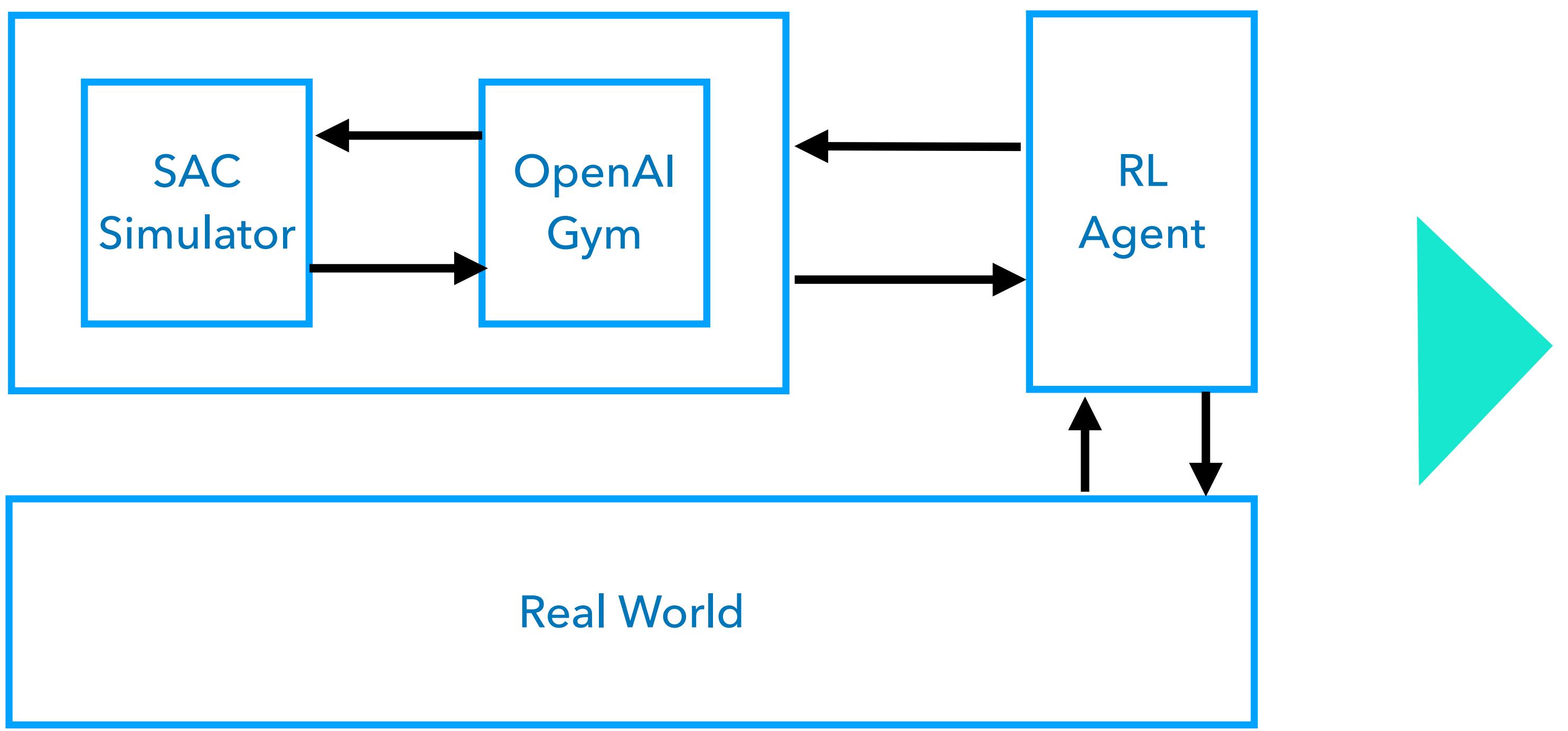
 BLOG POST
RESEARCH

20 JUL 2016

DeepMind AI Reduces Google Data Centre Cooling Bill by 40%

RL application [3/3]

Energy Optimal Control : SAC + RL



Reference

David Silver Lecture (UCL) : <https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver>

- Lecture 1 : Introduction to Reinforcement Learning
- Lecture 2 : Markov Decision Process
- Lecture 3 : Planning by Dynamic Programming
- Lecture 4 : Model-Free Prediction
- Lecture 5 : Model-Free Control

Richard S. Sutton et el., "Reinforcement learning" : <http://incompleteideas.net/book/bookdraft2017nov5.pdf>

지민님이 참고하신 **Blog** : <https://sumniya.tistory.com/>

Reinforcement learning is defined not by characterizing learning methods,
but by characterizing a learning problem.

- Richard S. Sutton

Q n A