

*Machine Learning*

# Introduction to Reinforcement Learning

Gabriele Russo Russo    Francesco Lo Presti

Laurea Magistrale in Ingegneria Informatica – A.Y. 2023/24



Dipartimento di Ingegneria Civile e  
Ingegneria Informatica

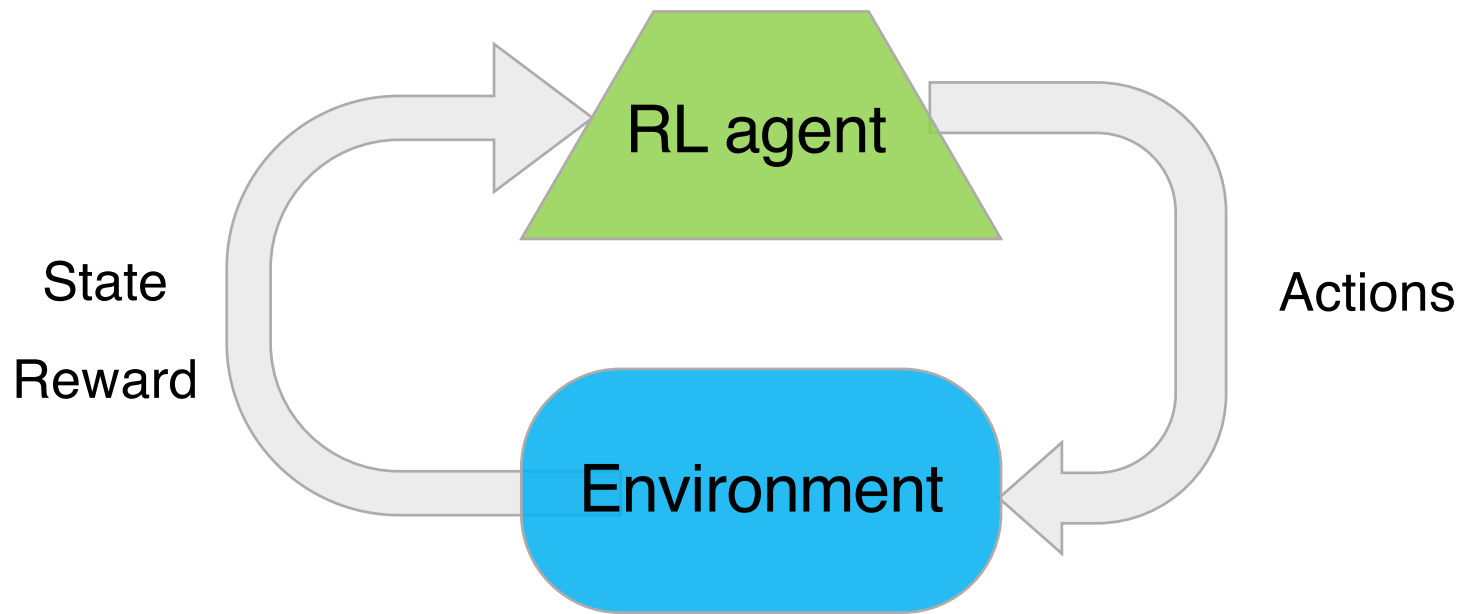
# Reinforcement Learning

- ▶ Supervised learning
- ▶ Unsupervised learning
- ▶ **Reinforcement learning**

# Reinforcement Learning

- ▶ Supervised learning
- ▶ Unsupervised learning
- ▶ **Reinforcement learning**
  - ▶ Branch of ML dealing with sequential decision-making

# Reinforcement Learning



- ▶ Agent interacts with environment through **actions**
- ▶ Feedback in the form of **reward** (or **paid cost**)
- ▶ Goal: maximizing cumulated reward over the long run
- ▶ Trial-and-error experience (no complete knowledge of environment a priori)

# Example: Tic-Tac-Toe

- ▶ **State**: representation of the board (3x3 matrix)
- ▶ **Actions**: available cells to mark
- ▶ **Reward**: 1 for a winning move, 0 otherwise

X	O	O
O	X	X
		X

# Example: AlphaZero by DeepMind

- ▶ Software able to play Go, Chess and Shogi <sup>1</sup>
  - ▶ Board games with huge number of legal positions (i.e., state space)
- ▶ Trained via self-play and advanced deep RL techniques
- ▶ Superhuman level of play with 24-hour training
- ▶ First presented in 2017; in 2019 [MuZero](#), generalization to play Atari games and other board games without prior rule knowledge

---

<sup>1</sup><https://arxiv.org/abs/1712.01815>

# Example: AlphaDev by DeepMind

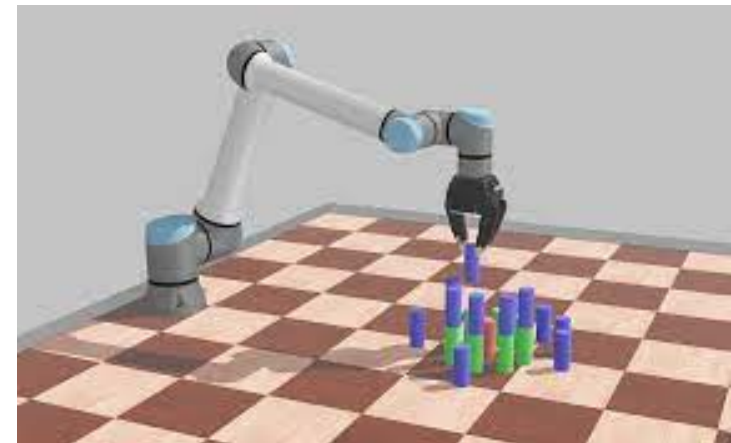
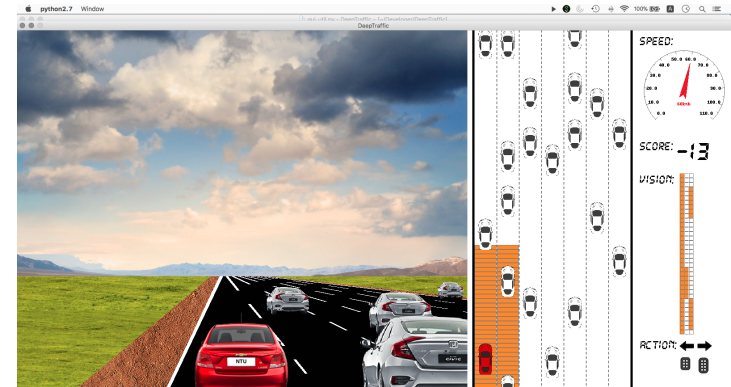
- ▶ Announced in 2023<sup>2</sup>
- ▶ RL used to develop new C++ sorting algorithm, now accepted in the standard library
- ▶ 70% faster on short sequences (2-3 items), 1.7% faster on long sequences
- ▶ State: instructions generated so far and state of the CPU
- ▶ Actions: assembly instructions to add
- ▶ Reward: based on sorting correctness and efficiency

---

<sup>2</sup><https://www.deepmind.com/blog/alphadev-discovers-faster-sorting-algorithms>

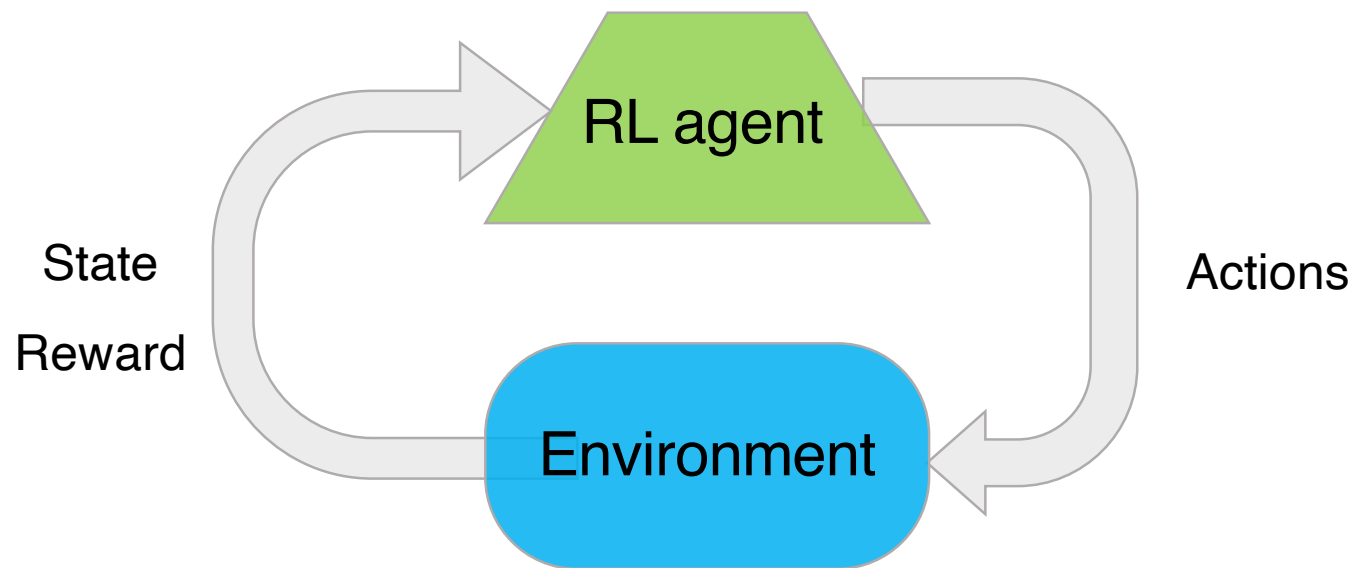
# Other Examples

- ▶ Autonomous vehicles
- ▶ Robot control
- ▶ Trading
- ▶ Autonomous network and computer systems
- ▶ Videogames
- ▶ ...



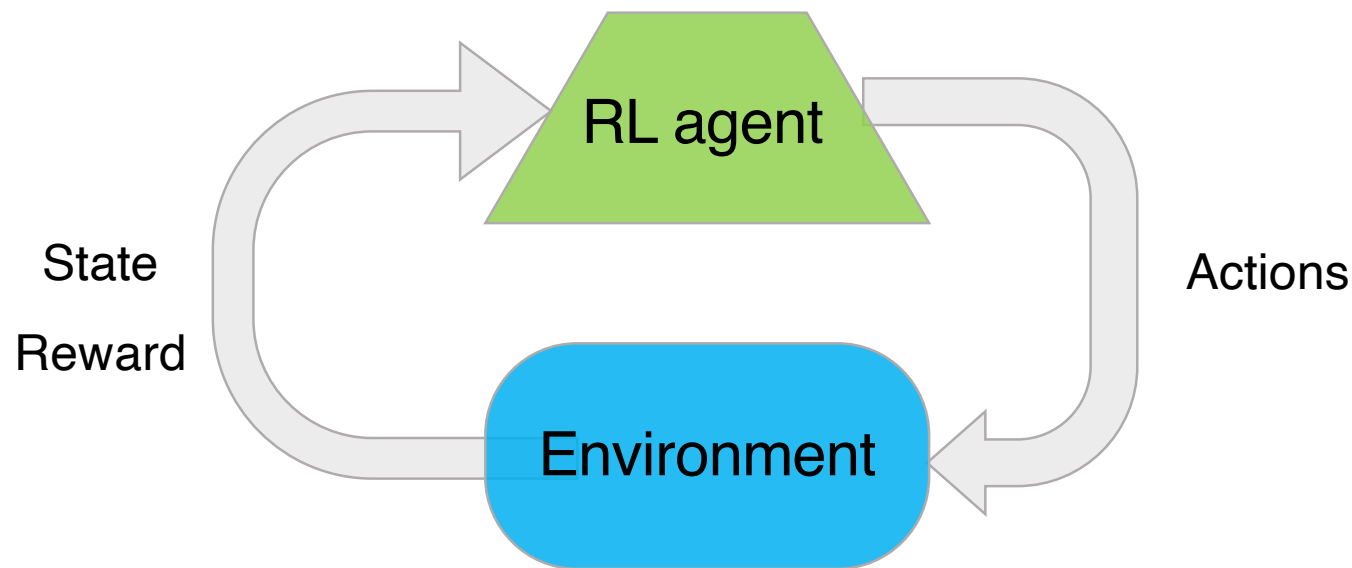


# Reinforcement Learning



- ▶ Agent, environment, actions, state, rewards, ...
- ▶ Modeled depending on the specific task
  - ▶ e.g., autonomous car uses different state information compared to chess player

# Reinforcement Learning



- ▶ Agent, environment, actions, state, rewards, ...
- ▶ Modeled depending on the specific task
  - ▶ e.g., autonomous car uses different state information compared to chess player
- ▶ Formally defined as a **Markov Decision Process (MDP)**
- ▶ A framework to model decision making in situations where outcomes are partly random

# Markov Decision Process (MDP)

- ▶ Extension of discrete-time Markov chains
- ▶ At each time step  $t$ , the process is in some state  $s_t$

# Markov Decision Process (MDP)

- ▶ Extension of discrete-time Markov chains
- ▶ At each time step  $t$ , the process is in some state  $s_t$
- ▶ The decision maker (the agent) chooses an action  $a_t$  among those available in state  $s_t$ 
  - ▶ e.g., robot observes current position and decides direction to move; some directions might be blocked by obstacles

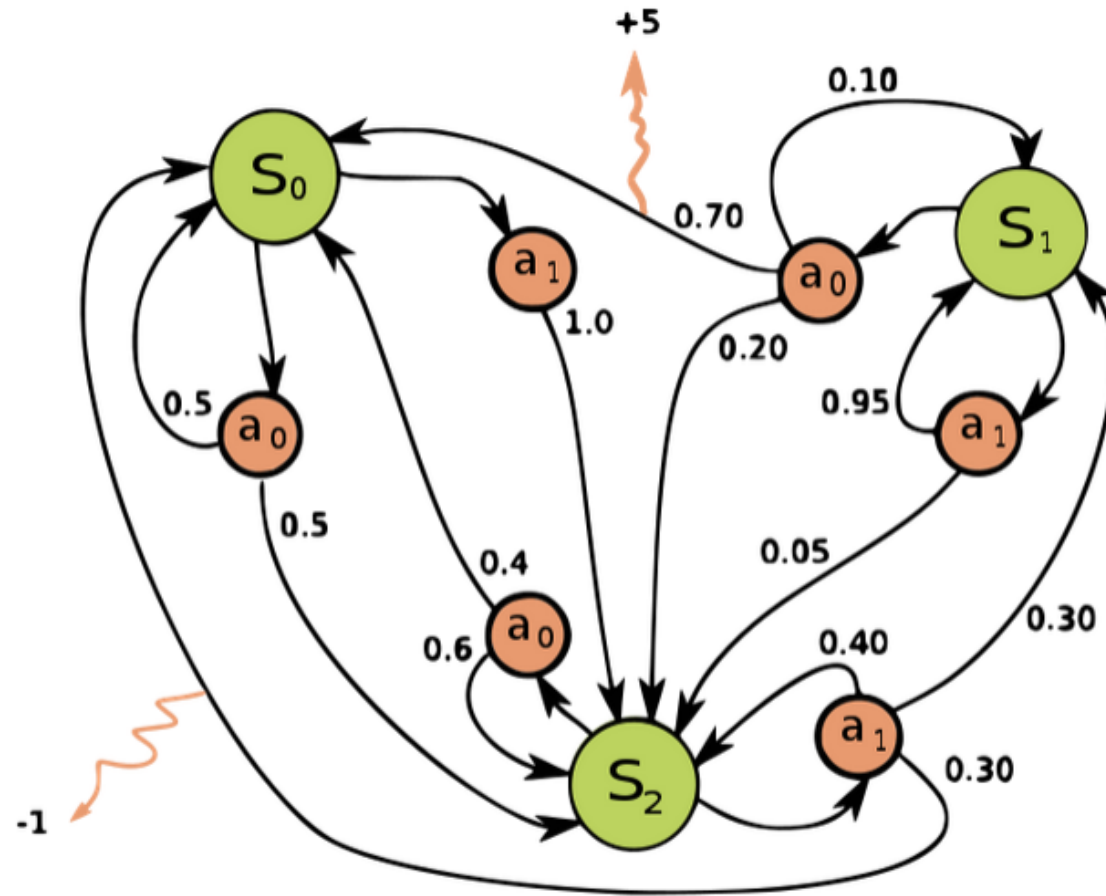
# Markov Decision Process (MDP)

- ▶ Extension of discrete-time Markov chains
- ▶ At each time step  $t$ , the process is in some state  $s_t$
- ▶ The decision maker (the agent) chooses an action  $a_t$  among those available in state  $s_t$ 
  - ▶ e.g., robot observes current position and decides direction to move; some directions might be blocked by obstacles
- ▶ Following  $a_t$ , the process moves to (random) state  $s_{t+1}$ 
  - ▶ e.g., autonomous drone chooses an action to reduce altitude; actual outcome may depend on (unpredictable) wind speed

# Markov Decision Process (MDP)

- ▶ Extension of discrete-time **Markov chains**
- ▶ At each time step  $t$ , the process is in some state  $s_t$
- ▶ The decision maker (the **agent**) chooses an action  $a_t$  among those available in state  $s_t$ 
  - ▶ e.g., robot observes current position and decides direction to move; some directions might be blocked by obstacles
- ▶ Following  $a_t$ , the process moves to (random) state  $s_{t+1}$ 
  - ▶ e.g., autonomous drone chooses an action to reduce altitude; actual outcome may depend on (unpredictable) wind speed
- ▶ Agent receives a **reward** (or, equivalently, pays a **cost**)
  - ▶ e.g., robot may get a reward for reaching its final destination
  - ▶ e.g., chess player rewarded at the end of a match

# Example



Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows).

# Markov Decision Process (2)

What defines an MDP?

- ▶  $\mathcal{S}$ : a (finite) set of states
- ▶  $\mathcal{A}$ : a (finite) set of actions
- ▶  $p$ : state transition probabilities

$$p(s'|s, a) = P[S_{t+1} = s' | S_t = s, A_t = a]$$

- ▶  $r$ : reward function (or,  $c$ : cost function)
  1.  $r(s, a) = E[R_t | S_t = s, A_t = a]$
  2.  $r(s, a, s') = E[R_t | S_t = s, A_t = a, S_{t+1} = s'] \longrightarrow$   
 $r(s, a) = \sum_{s'} p(s'|s, a) r(s, a, s')$



# Markov Decision Process (2)

What defines an MDP?

- ▶  $\mathcal{S}$ : a (finite) set of states
- ▶  $\mathcal{A}$ : a (finite) set of actions
- ▶  $p$ : state transition probabilities

$$p(s'|s, a) = P[S_{t+1} = s' | S_t = s, A_t = a]$$

- ▶  $c$ : cost function (**Alternative**)
  1.  $c(s, a) = E[C_t | S_t = s, A_t = a]$
  2.  $c(s, a, s') = E[C_t | S_t = s, A_t = a, S_{t+1} = s'] \longrightarrow$   
 $c(s, a) = \sum_{s'} p(s'|s, a) c(s, a, s')$

# Markov Property

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

## Definition

A state  $S_t$  is **Markov** if and only if

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

- ▶ The state captures all relevant information from the history
- ▶ i.e., the state is a sufficient statistic of the future

# Objective: Episodic Tasks

- ▶ Informally, we said that the agent aims to maximize the collected reward over time
- ▶ Let's consider an **episodic** task, where the agent-environment interaction naturally terminates at some final time step  $T$ 
  - ▶ e.g., the end of a chess match
  - ▶ e.g., the time a robot reaches its destination or runs out of battery
- ▶ At time  $t$ , we aim to maximize the **expected return**  $G_t$

$$G_t = R_t + R_{t+1} + \dots + R_T$$

# Objective: Continuing Tasks

- ▶ In many cases the agent–environment interaction does not break naturally into identifiable episodes, but goes on continually without limit
  - ▶ e.g., an agent managing VM migration in a Cloud datacenter
  - ▶ e.g., the control system of RL-based traffic lights
- ▶ In this scenario, the goal of the agent is maximizing the **expected cumulative discounted** reward

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

where  $\gamma \in [0, 1)$  is the discount factor

# Reward vs Cost

- ▶ You can either maximize the expected reward or minimize the expected cost

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

- ▶ The two formulations are equivalent; you can easily switch between them by setting

$$r(s, a) = -c(s, a)$$

- ▶ In the following, we will mostly refer to costs; keep in mind this equivalence

# Policy

## Definition

A **policy**  $\pi$  is a distribution over actions given a state  $s$

$$\pi(a|s) = p(A_t = a | S_t = s)$$

- ▶ A policy fully defines agent's behavior
- ▶ MDP policies depend on the current state only

# Policy

## Definition

A **policy**  $\pi$  is a distribution over actions given a state  $s$

$$\pi(a|s) = p(A_t = a | S_t = s)$$

- ▶ A policy fully defines agent's behavior
- ▶ MDP policies depend on the current state only
- ▶ Special case: **deterministic policy**

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

# Example: Deterministic Policy

<i>State</i>	<i>Action</i>
$s_1$	$a_1$
$s_2$	$a_1$
$s_3$	$a_2$
$s_4$	$a_1$



# Value Function

Value function is a prediction of future costs

- ▶ can be used to evaluate how good/bad states and/or actions are
- ▶ and therefore to select actions e.g.

<i>State</i>	$a_1$	$a_2$	$a_3$
$s_1$	10	5	3
$s_2$	8	6	4
$s_3$	6	5	6
$s_4$	5	4	6
$s_5$	4	3	7
$s_6$	1	5	9
$s_7$	0	9	15

# Value Function

Value function is a prediction of future costs

- ▶ can be used to evaluate how good/bad states and/or actions are
- ▶ and therefore to select actions e.g.

<i>State</i>	$a_1$	$a_2$	$a_3$
$s_1$	10	5	3
$s_2$	8	6	4
$s_3$	6	5	6
$s_4$	5	4	6
$s_5$	4	3	7
$s_6$	1	5	9
$s_7$	0	9	15

# Value Function

Value function is a prediction of future costs

- ▶ can be used to evaluate how good/bad states and/or actions are
- ▶ and therefore to select actions e.g.

State	$a_1$	$a_2$	$a_3$
$s_1$	10	5	3
$s_2$	8	6	4
$s_3$	6	5	6
$s_4$	5	4	6
$s_5$	4	3	7
$s_6$	1	5	9
$s_7$	0	9	15

$s$	$\pi(s)$
$s_1$	$a_3$
$s_2$	$a_3$
$s_3$	$a_2$
$s_4$	$a_2$
$s_5$	$a_2$
$s_6$	$a_1$
$s_7$	$a_1$

# Value Functions

## Action value function (or, $Q$ function)

Expected cost starting from state  $s$ , taking action  $a$  and then following policy  $\pi$

$$Q_{\pi}(s, a) = E_{\pi}[G_t | S_t = s, A_t = a]$$

## State value function

Expected cost starting from state  $s$  and then following policy  $\pi$

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

# Action Value Functions

The **action value function** can be decomposed into two parts:

- ▶ immediate cost
- ▶ discounted costs from successor state  $S_{t+1}$

$$Q_{\pi}(s, a) = E_{\pi}[G_t | S_t = s, A_t = a]$$

# Action Value Functions

The **action value function** can be decomposed into two parts:

- ▶ immediate cost
- ▶ discounted costs from successor state  $S_{t+1}$

$$\begin{aligned} Q_{\pi}(s, a) &= E_{\pi}[G_t | S_t = s, A_t = a] \\ &= E_{\pi}[C_t + \gamma C_{t+1} + \gamma^2 C_{t+2} \dots | S_t = s, A_t = a] \end{aligned}$$

# Action Value Functions

The **action value function** can be decomposed into two parts:

- ▶ immediate cost
- ▶ discounted costs from successor state  $S_{t+1}$

$$\begin{aligned}Q_{\pi}(s, a) &= E_{\pi}[G_t | S_t = s, A_t = a] \\&= E_{\pi}[C_t + \gamma C_{t+1} + \gamma^2 C_{t+2} \dots | S_t = s, A_t = a] \\&= E_{\pi}[C_t + \gamma (C_{t+1} + \gamma C_{t+2} \dots) | S_t = s, A_t = a]\end{aligned}$$

# Action Value Functions

The **action value function** can be decomposed into two parts:

- ▶ immediate cost
- ▶ discounted costs from successor state  $S_{t+1}$

$$\begin{aligned} Q_{\pi}(s, a) &= E_{\pi}[G_t | S_t = s, A_t = a] \\ &= E_{\pi}[C_t + \gamma C_{t+1} + \gamma^2 C_{t+2} \dots | S_t = s, A_t = a] \\ &= E_{\pi}[C_t + \gamma (C_{t+1} + \gamma C_{t+2} \dots) | S_t = s, A_t = a] \\ &= E_{\pi}[C_t + \gamma G_{t+1} | S_t = s, A_t = a] \end{aligned}$$



# Action Value Functions

The **action value function** can be decomposed into two parts:

- ▶ immediate cost
- ▶ discounted costs from successor state  $S_{t+1}$

$$\begin{aligned}Q_{\pi}(s, a) &= E_{\pi}[G_t | S_t = s, A_t = a] \\&= E_{\pi}[C_t + \gamma C_{t+1} + \gamma^2 C_{t+2} \dots | S_t = s, A_t = a] \\&= E_{\pi}[C_t + \gamma (C_{t+1} + \gamma C_{t+2} \dots) | S_t = s, A_t = a] \\&= E_{\pi}[C_t + \gamma G_{t+1} | S_t = s, A_t = a] \\&= c(s, a) + \gamma E_{\pi}[G_{t+1} | S_t = s, A_t = a]\end{aligned}$$

# Action Value Functions

The **action value function** can be decomposed into two parts:

- ▶ immediate cost
- ▶ discounted costs from successor state  $S_{t+1}$

$$\begin{aligned}Q_{\pi}(s, a) &= E_{\pi}[G_t | S_t = s, A_t = a] \\&= E_{\pi}[C_t + \gamma C_{t+1} + \gamma^2 C_{t+2} \dots | S_t = s, A_t = a] \\&= E_{\pi}[C_t + \gamma (C_{t+1} + \gamma C_{t+2} \dots) | S_t = s, A_t = a] \\&= E_{\pi}[C_t + \gamma G_{t+1} | S_t = s, A_t = a] \\&= c(s, a) + \gamma E_{\pi}[G_{t+1} | S_t = s, A_t = a]\end{aligned}$$

**Bellman equation:**

$$Q_{\pi}(s, a) = c(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) Q_{\pi}(s', \pi(s'))$$

# State Value Functions

The **value function** can be similarly decomposed into two parts:

- ▶ immediate cost  $C_t$
- ▶ discounted cost from successor state  $V(S_{t+1})$

$$\begin{aligned} V_{\pi}(s) &= E_{\pi}[G_t | S_t = s] \\ &= E_{\pi}[C_t + \gamma C_{t+1} + \gamma^2 C_{t+2} \dots | S_t = s] \\ &= E_{\pi}[C_t + \gamma (C_{t+1} + \gamma C_{t+2} \dots) | S_t = s] \\ &= E_{\pi}[C_t + \gamma G_{t+1} | S_t = s] \end{aligned}$$

**Bellman equation:**

$$V_{\pi}(s) = c(s, \pi(s)) + \gamma \sum_{s'} p(s' | s, \pi(s)) V_{\pi}(s')$$

# Optimal Value Function

## Optimal action value function

$Q^*(s; a)$  is the minimum action-value function over all policies

$$Q^*(s, a) = \min_{\pi} Q_{\pi}(s, a)$$

## Optimal state value function

$V^*(s)$  is the minimum value function over all policies

$$V^*(s) = \min_{\pi} V_{\pi}(s)$$

# Bellman Optimality Equations

$$Q_{\pi}(s, a) = c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) Q_{\pi}(s', \pi(a'))$$



$$Q^*(s, a) = c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \min_{a'} Q^*(s', a')$$

$$V^*(s) = \min_a Q^*(s, a)$$

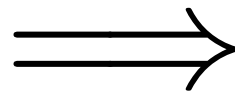
$$Q^*(s, a) = c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V^*(s')$$

# Optimal Policy

Given  $Q^*(s, a)$  the optimal action when the system is in state  $s$  is:

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

State	$a_1$	$a_2$	$a_3$
$s_1$	10	5	3
$s_2$	8	6	4
$s_3$	6	5	6
$s_4$	5	4	6
$s_5$	4	3	7
$s_6$	1	5	9
$s_7$	0	9	15



Optimal Action
$a_3$
$a_3$
$a_2$
$a_2$
$a_2$
$a_1$
$a_1$

# How to compute $V^*$ ?

- ▶ If we know the optimal value function, we have an optimal policy!
- ▶ **But...** how do we compute the optimal value function??

# Value Iteration

## Bellman Equation

$$Q^*(s, a) = c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \min_{a'} Q^*(s', a')$$



# Value Iteration

## Bellman Equation

$$Q^*(s, a) = c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \min_{a'} Q^*(s', a')$$

- ▶ Suppose we know the solution to subproblems  $Q^*(s', a')$
- ▶  $Q^*(s, a)$  can be computed by one-step lookahead

$$Q^*(s, a) \leftarrow c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \min_{a'} Q^*(s', a')$$

# Value Iteration

## Bellman Equation

$$Q^*(s, a) = c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \min_{a'} Q^*(s', a')$$

- ▶ Suppose we know the solution to subproblems  $Q^*(s', a')$
- ▶  $Q^*(s, a)$  can be computed by one-step lookahead

$$Q^*(s, a) \leftarrow c(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \min_{a'} Q^*(s', a')$$

- ▶ The idea is to apply these updates iteratively
- ▶ Proven to converge (see, Contraction Mapping Theorem in Sutton's book)

# Value Iteration: Algorithm

## Value Iteration

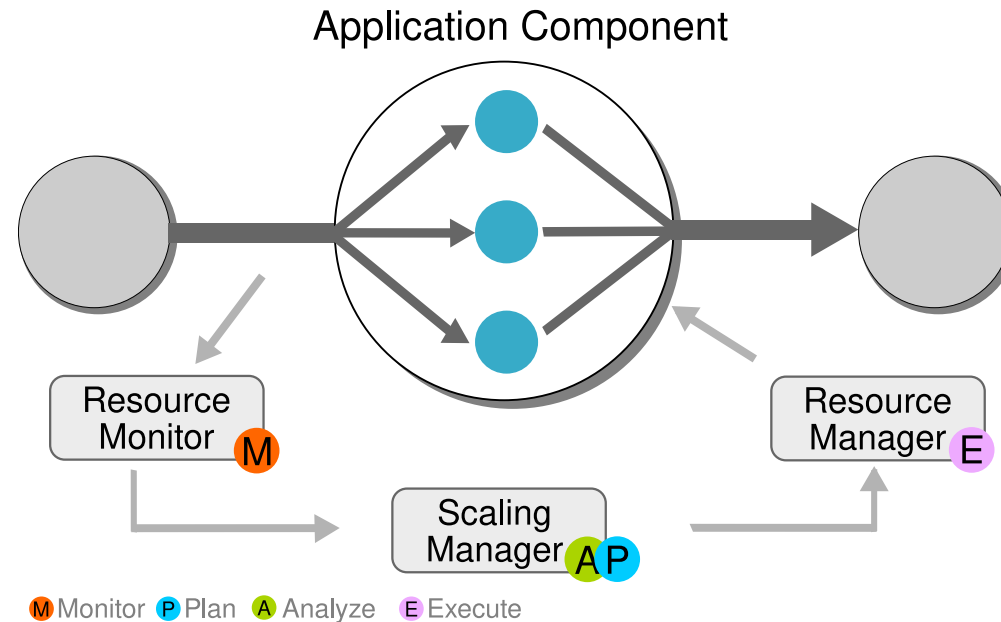
```
1  $i \leftarrow 0$ 
2  $Q_i(s, a) \leftarrow 0, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$ 
3 repeat
4   forall  $s \in \mathcal{S}$  do
5     forall  $a \in \mathcal{A}(s)$  do
6        $Q_{i+1}(s, a) \leftarrow$ 
7          $c(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \min_{a' \in \mathcal{A}(s')} Q_i(s', a')$ 
8     end
9   end
10   $i \leftarrow i + 1$ 
11 until  $\max_{s,a} |Q_i(s, a) - Q_{i-1}(s, a)| < \epsilon$ 
12  $\pi^*(s) = \arg \min_a Q_i(s, a), \forall s \in \mathcal{S}$ 
```

# Value Iteration: Alternative Algorithm

## Value Iteration - Alternative

```
1  $i \leftarrow 0$ 
2  $Q_i(s, a) \leftarrow 0, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$ 
3  $V_i(s) \leftarrow 0, \forall s \in \mathcal{S}$ 
4 repeat
5   forall  $s \in \mathcal{S}$  do
6     forall  $a \in \mathcal{A}(s)$  do
7        $Q_{i+1}(s, a) \leftarrow c(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V_i(s)$ 
8     end
9      $V_{i+1}(s) = \min_{a' \in \mathcal{A}(s)} Q_{i+1}(s, a')$ 
10  end
11   $i \leftarrow i + 1$ 
12 until  $\max_{s,a} |Q_i(s, a) - Q_{i-1}(s, a)| < \epsilon$ 
13  $\pi^*(s) = \arg \min_a Q_i(s, a), \forall s \in \mathcal{S}$ 
```

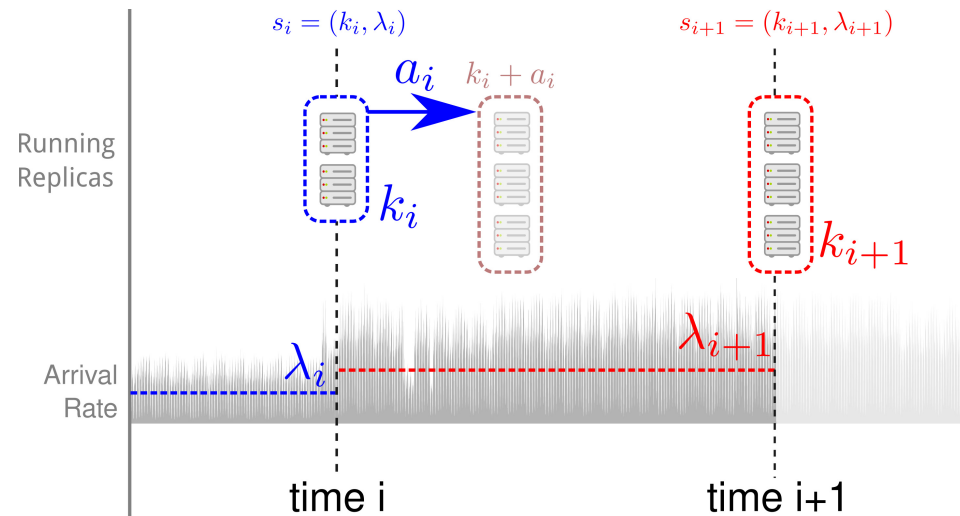
# Horizontal Auto-scaling: MDP formulation



- ▶ We periodically make a decision about scaling in/out an app component
- ▶ We are concerned with 3 objectives:
  - ▶ Monetary resource cost (or, resource usage in general)
  - ▶ Performance req. satisfaction (e.g., max response time)
  - ▶ Scaling overhead

# Horizontal Auto-scaling: MDP formulation

- ▶ We model the problem as a **Markov Decision Process (MDP)**



- ▶ State at time slot  $i$ :  $s_i = (k_i, \lambda_i)$ 
  - ▶  $k_i$  component parallelism
  - ▶  $\lambda_i$  avg. arrival rate (of requests, jobs, data, ...)
- ▶ Action at time slot  $i$ :  $a_i \in \{0, +1, -1\}$

# MDP Model: Transition Probabilities

- ▶ State of the system  $s = (k, \lambda)$ 
  - ▶  $1 \leq k \leq K^{max}$  Component parallelism
  - ▶  $\lambda$  avg. input rate
    - ▶  $\lambda$  is discretized, i.e.,  $\lambda_i \in \{0, \Delta\lambda, 2\Delta\lambda, (L-1)\Delta\lambda\}$
    - ▶  $\Delta\lambda$  quantization step size,  $L$  number of discrete values
- ▶ Available actions  $\mathcal{A} = \{-1, 0, +1\}$
- ▶ Transition probabilities  $p(s'|s, a) = p((k', \lambda')|(k, \lambda), a)$

$$\begin{aligned} p(s'|s, a) &= P[s_{t+1} = (k', \lambda') | s_t = (k, \lambda), a_t = a] = \\ &= \begin{cases} P[\lambda_{t+1} = \lambda' | \lambda_t = \lambda] & k' = k + a \\ 0 & \text{otherwise} \end{cases} = \\ &= \mathbb{1}_{\{k'=k+a\}} P[\lambda_{t+1} = \lambda' | \lambda_t = \lambda] \end{aligned}$$

# MDP Model: Cost Function

- ▶ Simple Additive Weighting method
- ▶ Cost associated with action execution and state transition  
 $(s, a) \rightarrow s'$

$$c(s, a, s') = \underbrace{w_{res} \frac{k + a}{K^{max}}}_{\text{Resource Cost}} + \underbrace{w_{perf} \mathbb{1}_{\{R(s, a, s') > R^{max}\}}}_{\text{Performance}} + \underbrace{w_{rcf} \mathbb{1}_{\{a \neq 0\}}}_{\text{Reconfiguration}}$$

Resource Cost

Performance

Reconfiguration

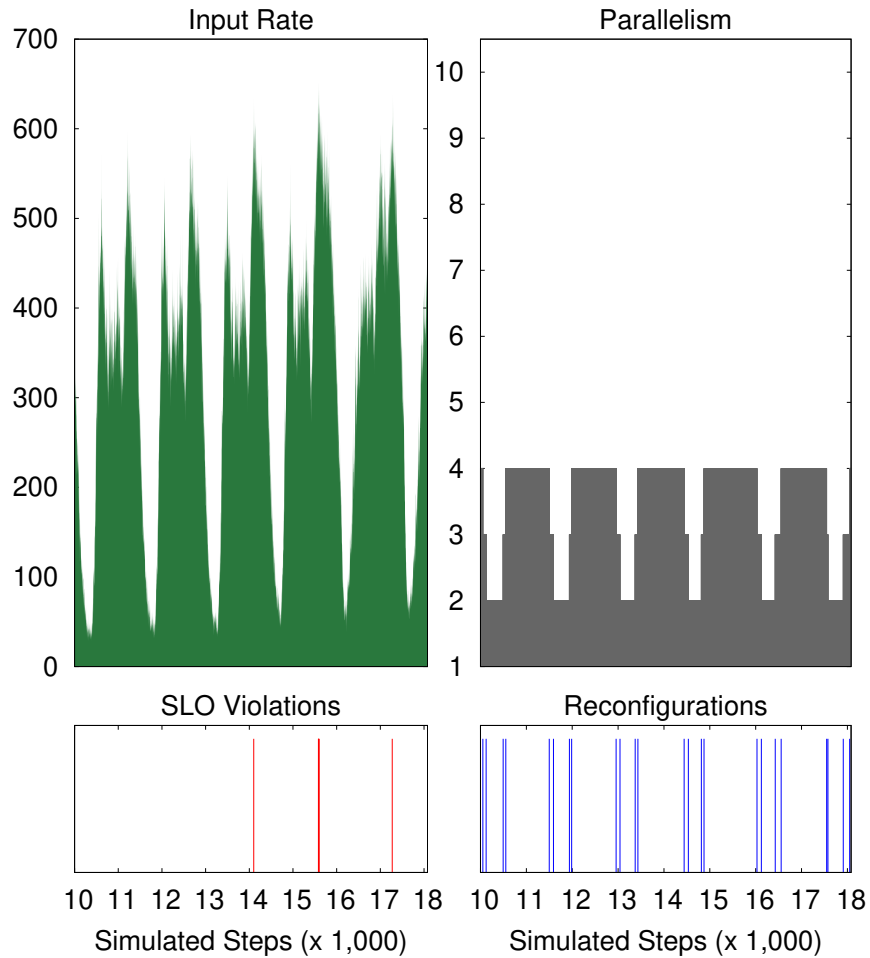
- ▶  $w_{res} + w_{perf} + w_{rcf} = 1, w_x \geq 0, x \in \{res, perf, rcf\}$
- ▶  $R(s, a, s')$  is the component performance index, e.g, response time,
- ▶  $R^{max}$  is the component reference performance value

We want to minimize  $\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t, s_{t+1}), \quad \gamma \in [0, 1)$

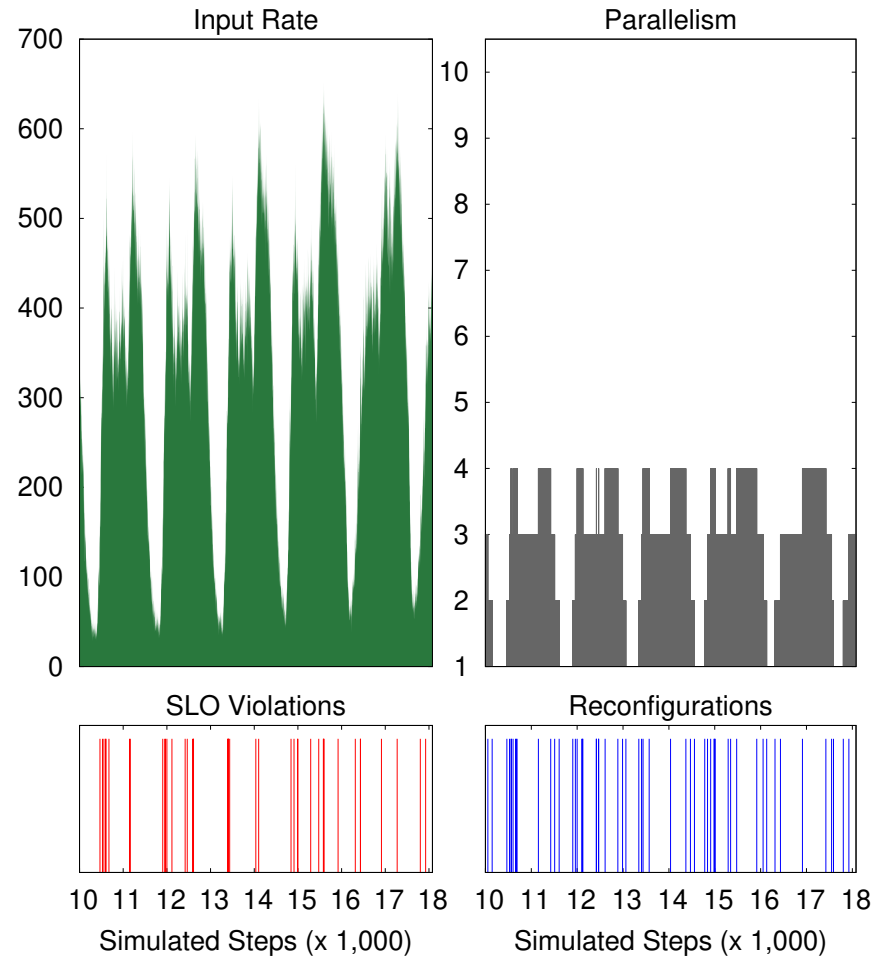


# Trading-off Objectives

$$w_{perf}=0.6, w_{res} = w_{rcf}=0.2$$



$$w_{res}=0.6, w_{perf} = w_{rcf}=0.2$$



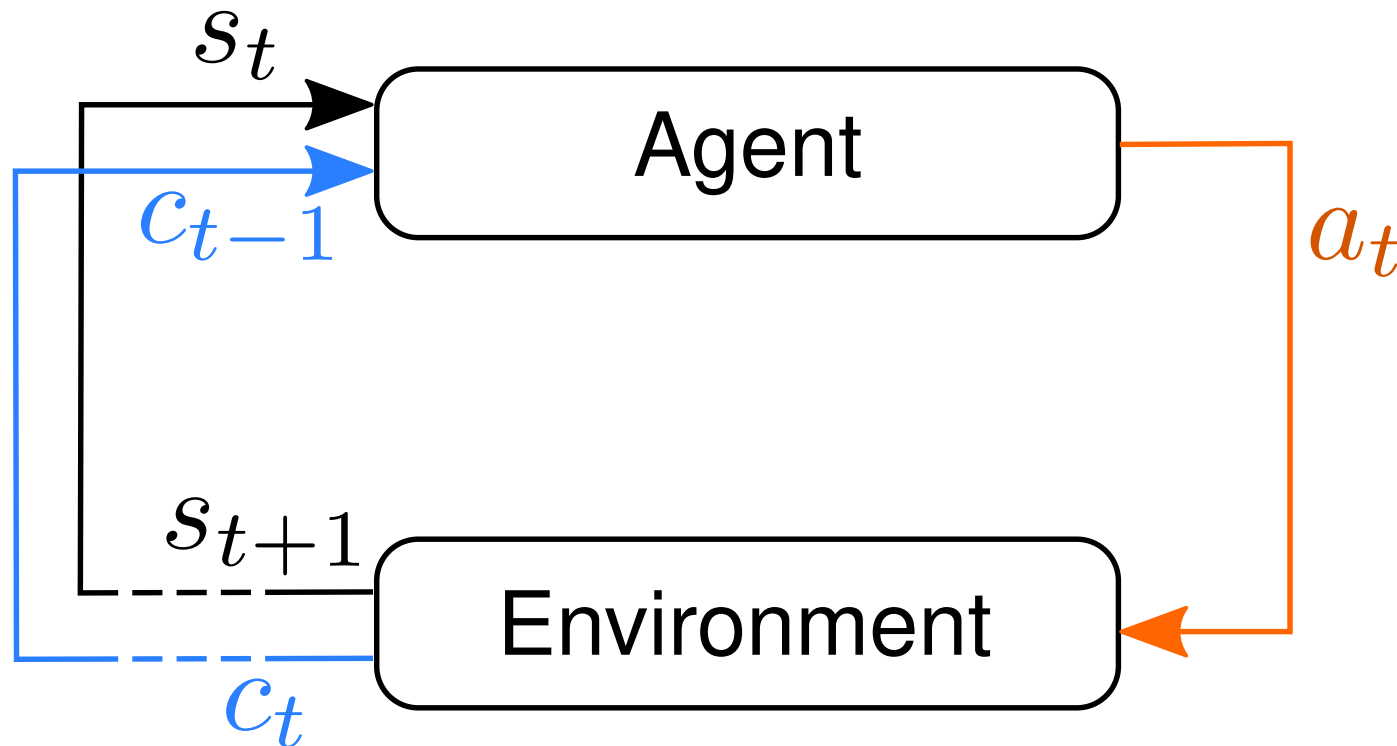
# MDP Resolution

- ▶ We can use the Value Iteration algorithm to solve the MDP
  - ▶ i.e., finding the optimal policy
- ▶ Is this enough?

# MDP Resolution

- ▶ We can use the Value Iteration algorithm to solve the MDP
  - ▶ i.e., finding the optimal policy
- ▶ Is this enough?
- ▶ Unfortunately, solving the MDP requires exact and complete knowledge of the underlying model
  - ▶ state transition probabilities
  - ▶ cost function
- ▶ In practice, we don't have such information!

# Reinforcement Learning



- RL aims to learn the optimal policy through interaction and evaluative feedback

# Model-free vs Model-based RL

- ▶ **Model-free** RL: no model of the environment is available or used; the optimal policy is learned through experience only
- ▶ **Model-based** RL: a (possibly partial) model of the environment is available and used to derive the optimal policy
  - ▶ a partial model can boost learning speed
  - ▶ RL may also be used in presence of a complete model instead of VI; e.g., with a large number of rarely visited states VI would unnecessarily run for a long time!
  - ▶ You may also try to learn the model online and use it to compute a policy

# Value-based vs Policy-based RL

- ▶ **Value-based** RL: aims to learn the optimal value function through experience; the policy is derived from it
  - ▶ Simplest RL algorithms belong to this group
  - ▶ We will mainly focus on this group in the following
- ▶ **Policy-based** RL: aims to directly learn the optimal policy through experience; no explicit computation/learning of the value function
- ▶ **Hybrid** approaches: e.g., the Actor-Critic framework