SMU Lyle School of Engineering

Reinforcement Learning

Introduction

Textbook Chapter 1

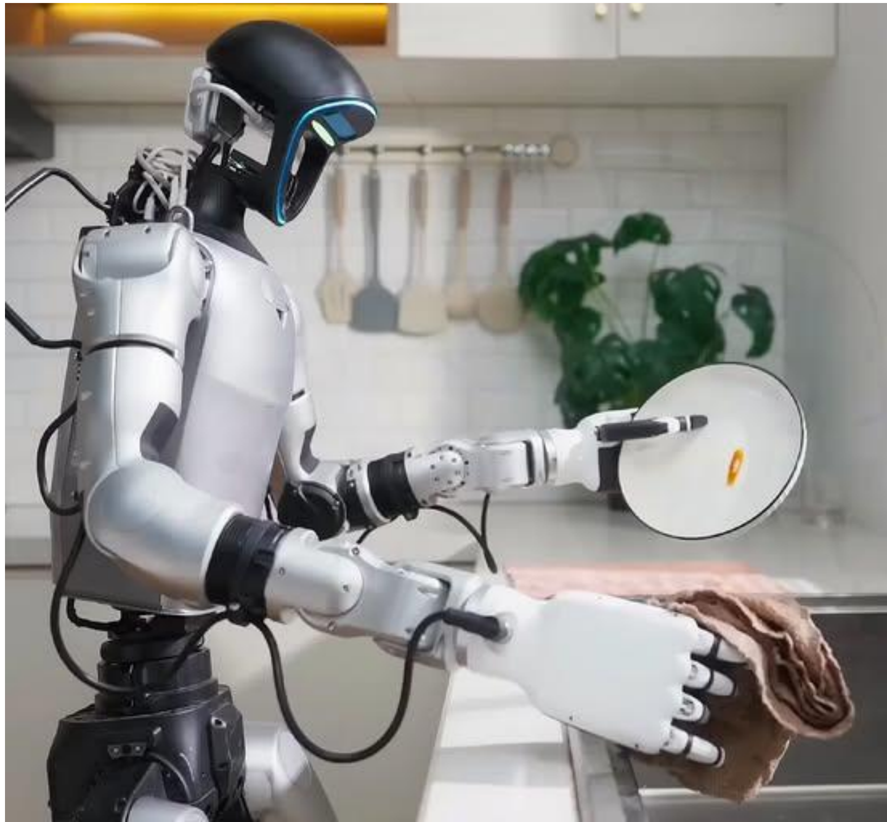
Michael Hahsler, SMU

Based on slides by Oliver Wallscheid (Paderborn University) and material by David Silver (University College London)

How Do We Build Intelligent Systems?

Options:

- Explicitly program behavior: **Rule-based agent, planning**
- Learn behavior from labeled data offline: **Supervised learning**
- Learn behavior using feedback from interacting with the environment (online): **Reinforcement Learning**



Topics of this Course

- **Introduction to reinforcement learning**
- Markov decision processes
- Part I: Tabular Methods
  - Dynamic programming
  - Monte Carlo methods
  - Temporal-difference learning
  - Multi-step bootstrapping
  - Planning and learning with tabular methods
- Part II: Approximate Solution Methods
  - Prediction and Control using Approximation
  - Eligibility Traces
  - Policy Gradient Methods
- Part III: Modern RL Methods
  - Deep Reinforcement Learning
  - Current Applications

Recommended Readings

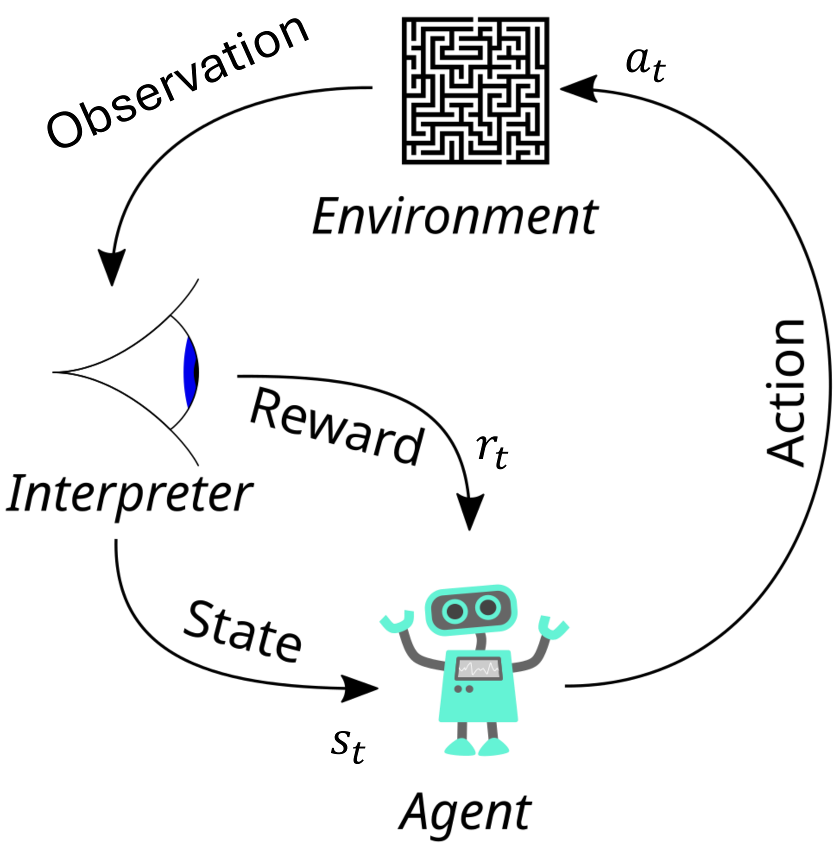
- Richard S. Sutton, Andrew G. Barto,
  Reinforcement Learning: An Introduction,
  2nd edition, MIT Press, Cambridge, MA, 2018.

- Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare and Joelle Pineau,
  An Introduction to Deep Reinforcement Learning,
  Foundations and Trends in Machine Learning, 11:3-4, pp 219-354, 2018.

Table of Contents

What is RL?

The Basic Reinforcement Learning Structure



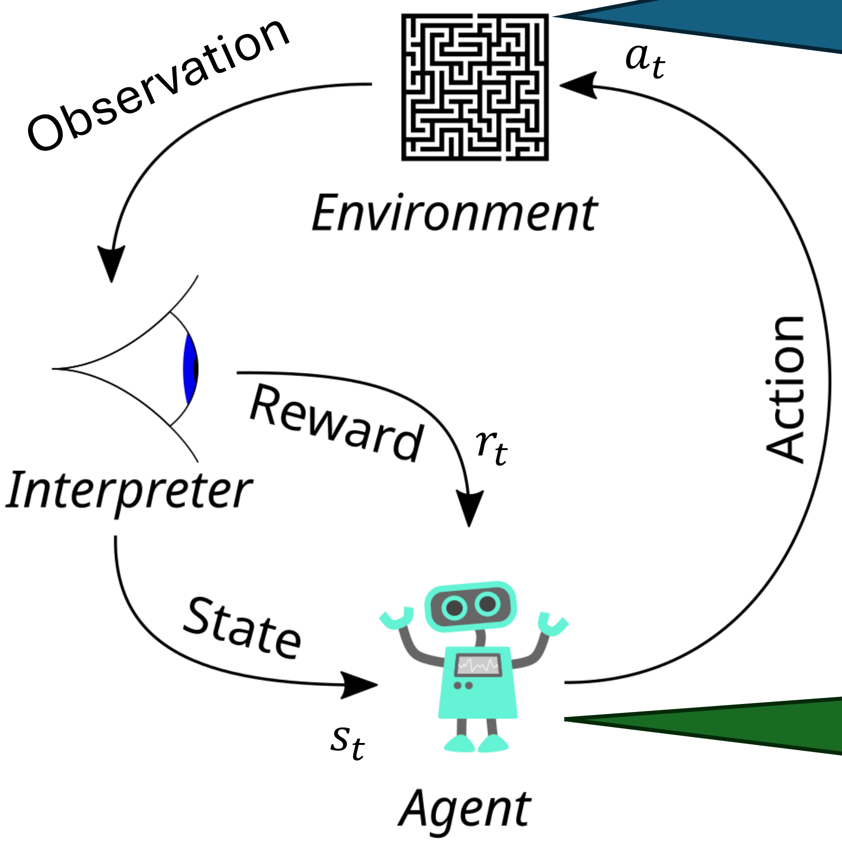
The basic RL operation principle
(derivative of www.wikipedia.org, CC0 1.0)

Goal: Map a situation to an action to maximize the reward

Key characteristics:

- **Uncertainty about the environment** leads to a need for sensing.
- **No supervisor**: discover actions by trying them (data-driven)
- **Sequential decision making**: Agent actions affect subsequent data. Agent maximizes over future (delayed) rewards which need to be estimated!
- Action discovery and maximization lead to a tradeoff: **exploit vs. explore**

Agent and Environment



We often model the interpreter as part of the environment.

At each step t the environment:

- Receives an action a_t.
- Emits an observation s_{t+1}.
- Emits a reward r_{t+1}.

The time increments $t \leftarrow t+1$.

At each step t the agent:

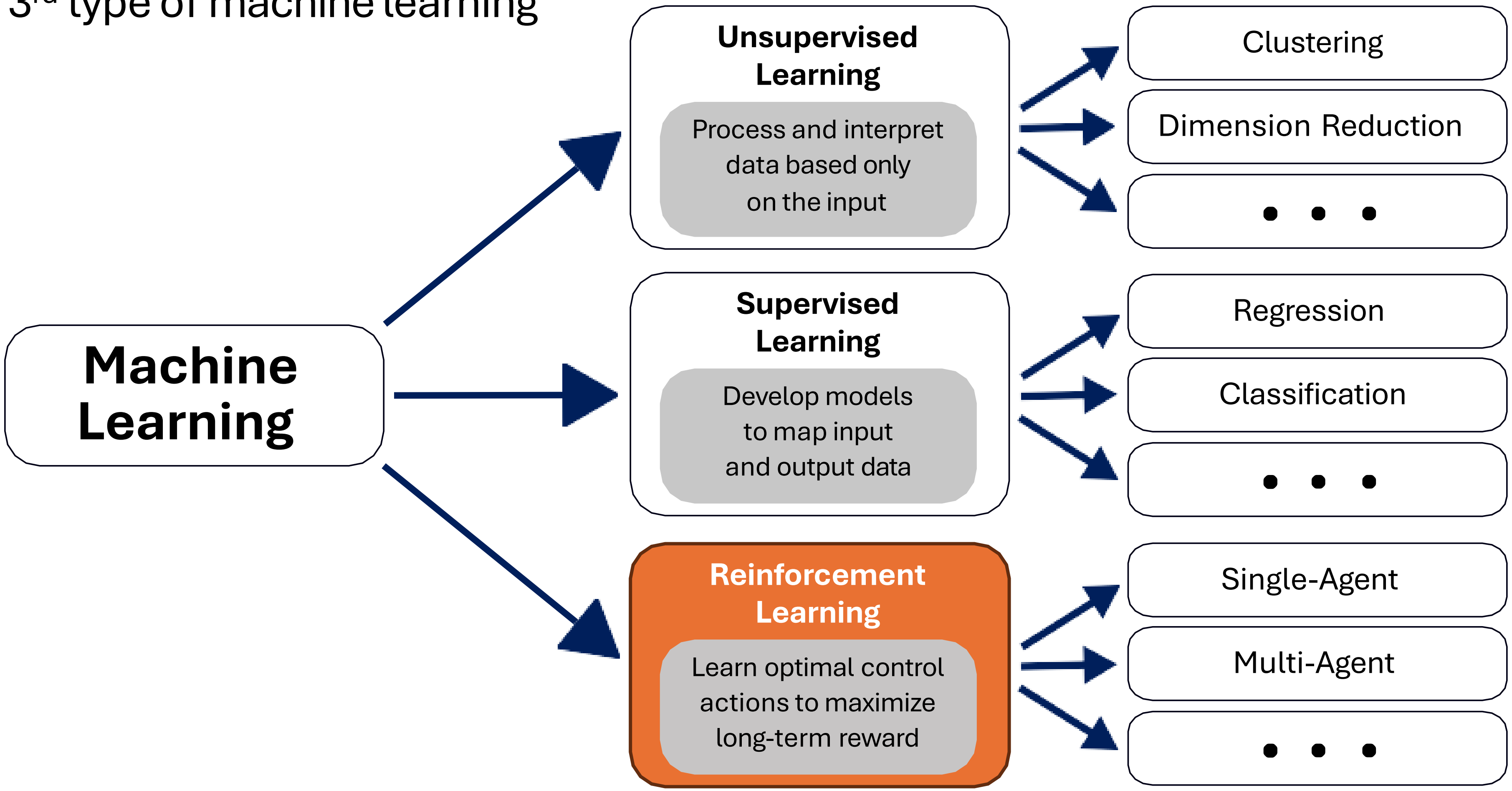
- Receives the state s_t.
- Receives a reward r_t.
- Picks an action a_t.

Remark on time: In AI, we often assume a time delay of one unit between executing the action and receiving the next state as well as the reward. This is called a discrete-time process (vs. a continuous-time process).

RL and Machine Learning

Machine learning is studied in AI to learn from examples.

RL is a 3rd type of machine learning



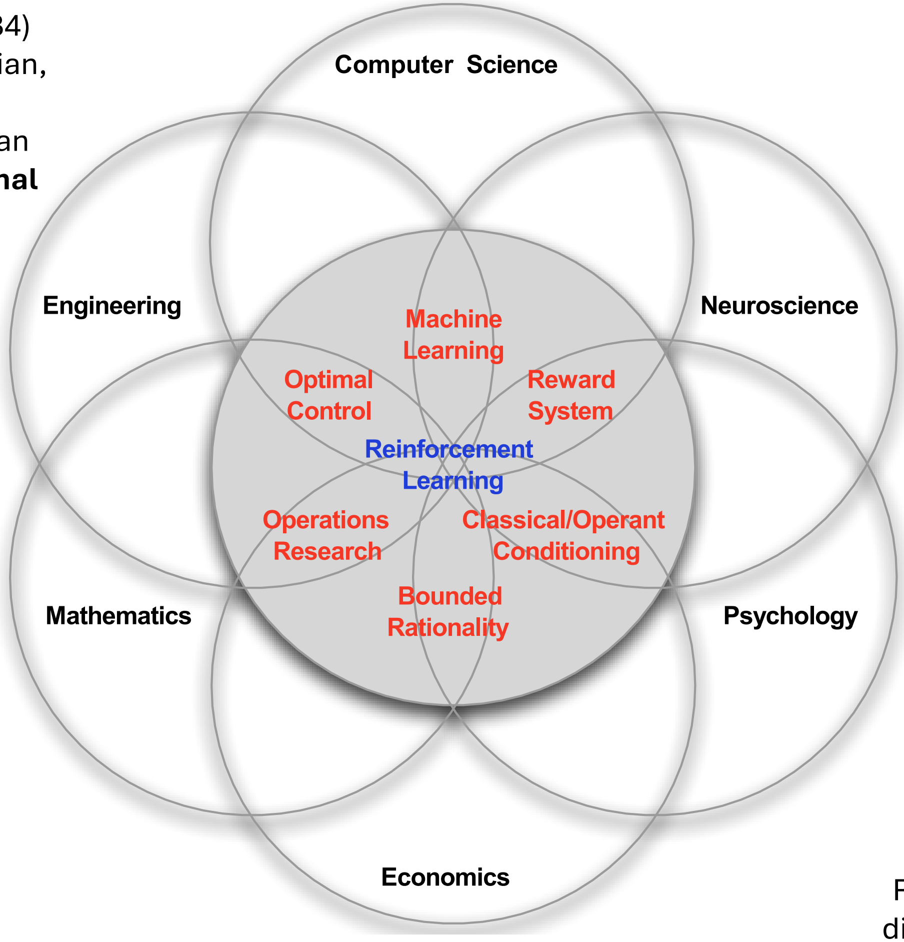
Many Faces of Reinforcement Learning

Optimal sequential decision making

Richard E. Bellman (1920-1984)
American applied mathematician, who introduced **dynamic programming** and the Bellman equation. Foundation of **optimal control theory**.

Stochastic process formalism

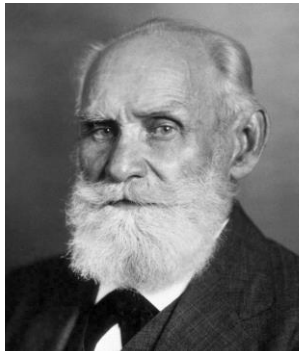
Andrei Markov (1856-1922)
Russian mathematician developed the groundwork for **Markov chains** to model **dynamical systems**.



RL and its neighboring domains

(source: D. Silver, "Reinforcement learning", 2015. CC BY-NC 4.0)

Classical conditioning



Ivan Pavlov (1849-1936)
Russian physiologist and Nobel laureate who discovered **classical conditioning** through his experiments with dogs.

Contemporary application examples

RL has led to superhuman performance for many games and simple toy examples

- Playing Atari Breakout
- Play strategy board game Go at super-human performance
- Simulated classic control problems
- Swinging-up and balance a cart-pole / an inverted pendulum

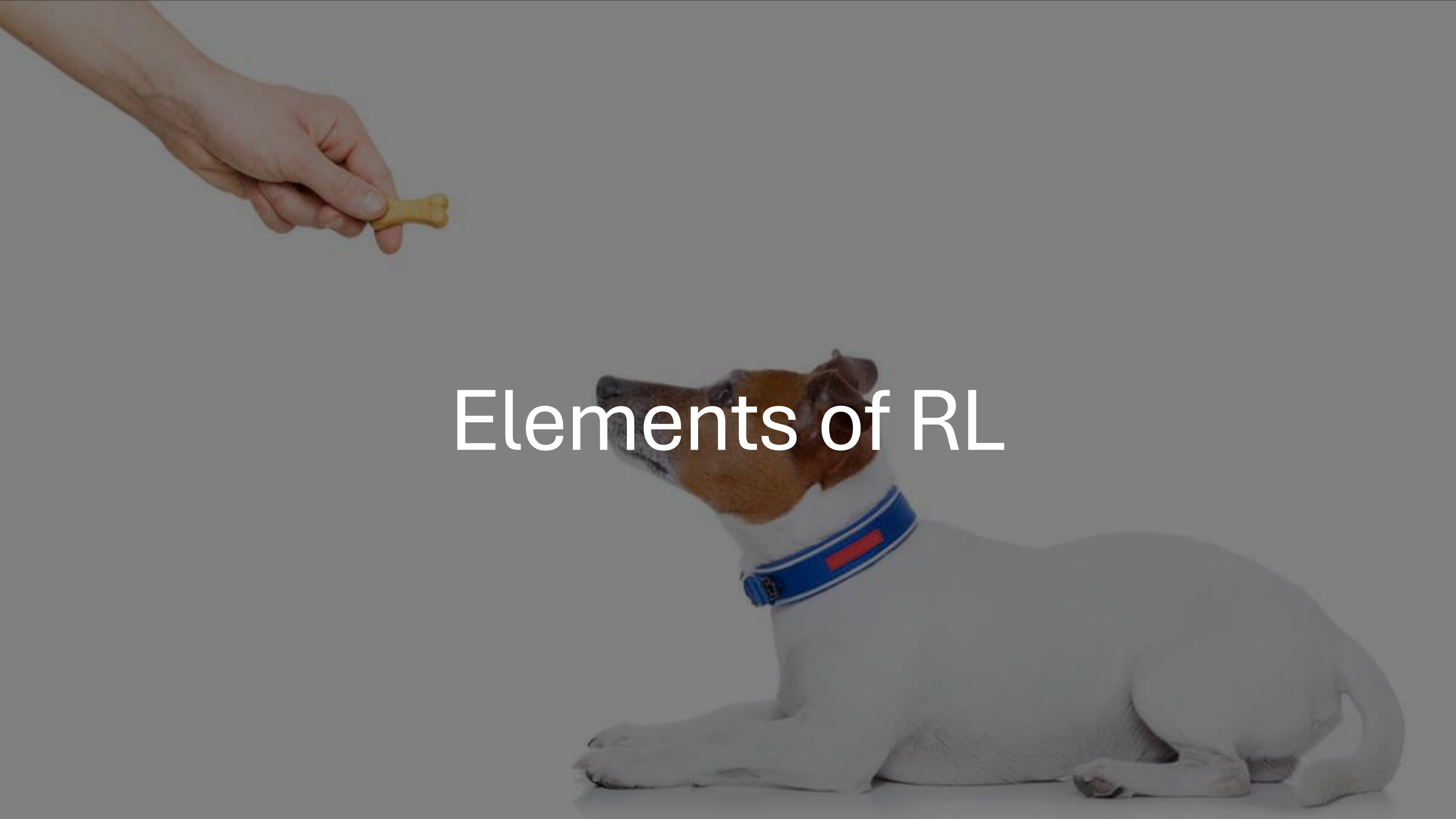
Using RL for real applications is promising, but much more difficult. Here are some examples:

- Controlling electric drive systems
- Flipping pancakes with a roboter arm
- Drifting with a RC-car
- Driving an autonomous car
- Nuclear fusion reactor plasma control
- Training chat bots (like chatGPT)
- …

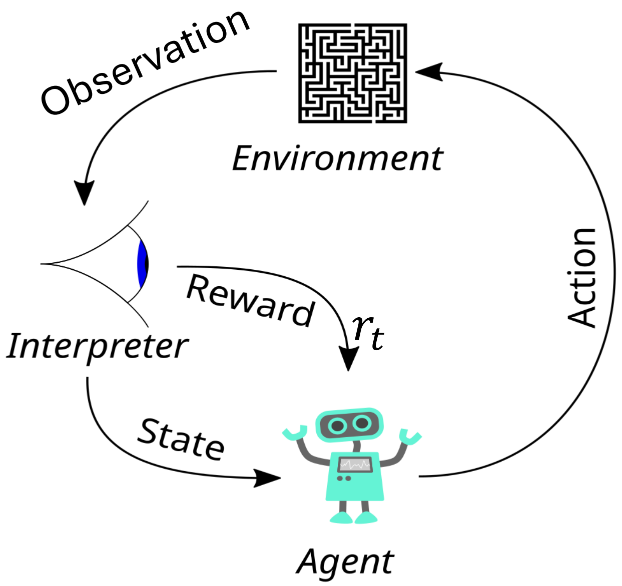
Reasons for difficulty:

- Real environments are complicated.
- Sensors (observations) are noisy.
- We often have partial observability.
- The number of different actions can be large.
- Measuring reward can be difficult.
- No access to a fast, perfect simulation environment.

Elements of RL



Elements of RL

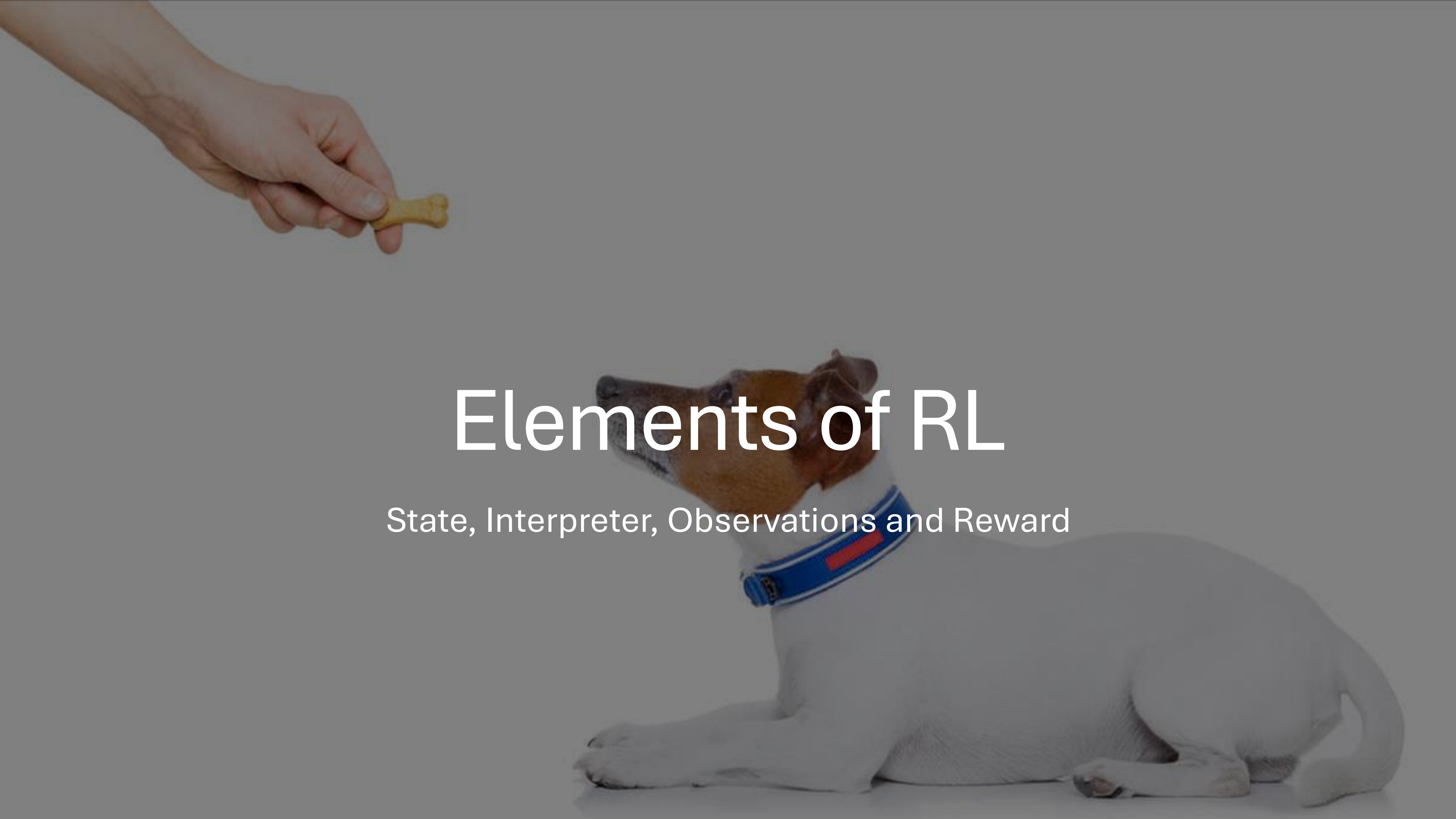


We will discuss:

- State, Interpreter, Observations and Reward
- Actions and Policy

Elements of RL

State, Interpreter, Observations and Reward

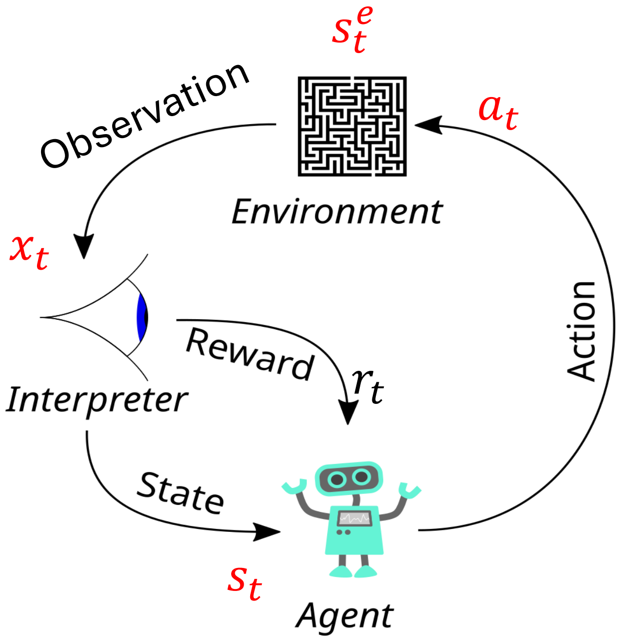


State

- The environment has an **environment state** s_t^e that **contains all information** needed to determine how to react to actions:
  - Physical states, e.g., location in a maze, current car velocity and direction
  - Game states, e.g., current chess board
  - Financial states, e.g., stock market prices, sentiment
- We call such a state **Markovian** or an **information state.**

- **Agent state** s_t is the agent's internal representation of the situation.
  - What does it know about the environmental state?
  - It is the basis of choosing the next action.
  - Can be incomplete and noisy based on observations x_t
  - Can include the agent's internal memory. E.g., agent's current strategy), belief states, learned features, etc.
  - Can be compressed (often using an artificial neural network).

- **Important**: With state (people will even call it "environment" state), we will mean the agent state, and we will use the random variable Xe S_t.
- The state typically has a factored representation with discrete and continuous variables.



Degree of Observability

Full observability

- Agent directly observes the full environment state (e.g., $s_t = s_t^e$).
- s_t is an information state (Markovian): Markov Decision Process (MDP)

Partial observability

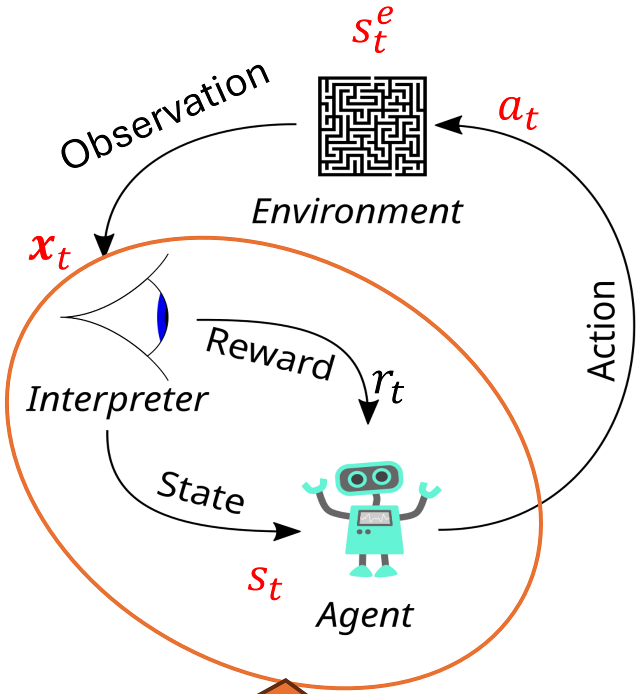
- Agent does not have full access to the environment state but observes state features $\boldsymbol{x}_t = f(s_t^e)$. $\boldsymbol{x}_t$ is typically not Markovian since it is not sufficient to predict the environment's behavior.
- Agent may try to reconstruct the complete state information (state estimation):

$$s_t = \widehat{s_t^e} = \text{update}\big(\text{predict}(\widehat{s_{t-1}^e}, a_t), \boldsymbol{x}_t\big)$$

- This leads to Partially Observable Markov Decision Processes (POMDP)

POMDP examples

- Self-driving cars with only cameras and no radar/lidar (to reduce cost)
- Poker game (opponents' cards and what cards will be dealt next are unknown)
- Human health status (we cannot sense everything and the detailed system dynamics are unknown)



The agent design often includes designing the interpreter

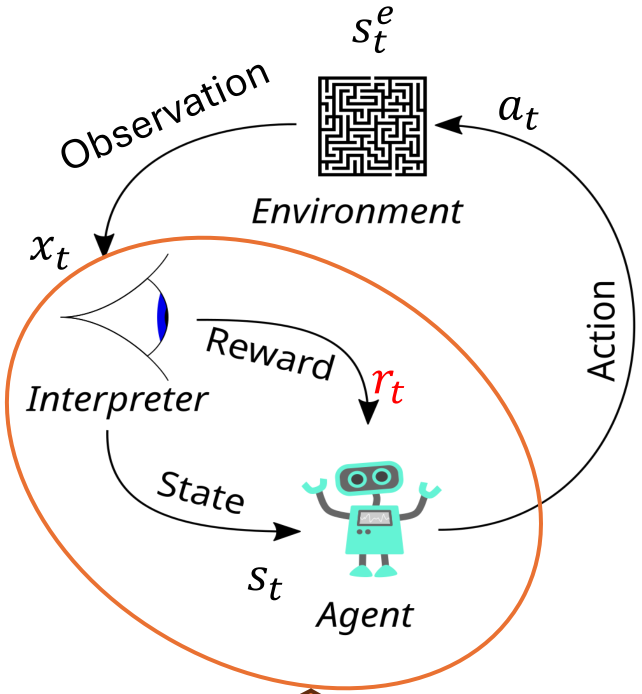
Reward

- An **immediate reward** is a scalar random variable R_t with realizations $r_t \in \mathbb{R}$ that the agent "receives" at time t.
- **Reward function**: The reward is often defined as a function of the current state $r(s_t)$. Examples:
  - 1 point for winning a game (state is a winning board).
- The **agent's task** is to choose actions to maximize the long-term reward called the return.

Reward hypothesis: *"All of what we mean by goals and purposes can be well thought of as the maximization of the expected cumulative reward."*

$$\max \mathbb{E}\left[\sum_{t=0}^{\infty} w_t R_t\right]$$

- The reward hypothesis is a **foundational assumption** of RL! The reward structure fully determines the goal.
- The **designer's task** is to design the reward signal so the agent can learn how to achieve the goal. This is called **reward shaping**. E.g., add a reward for intermediate goals, such as achieving 3 in a row in Connect-4.



The agent design often includes designing the interpreter

Note: w_t is just a weight that depends only on t .

Reward Characteristics

Rewards are highly dependent on the given problem:

- Actions may have **short and long-term consequences**.
  - The reward for a certain action may be delayed, leading to the **credit assignment problem**.
  - Examples: Stock trading, strategic board games,…

- Rewards can be **positive and negative values**.
  - Certain situations (e.g., car hits a wall) might lead to a negative reward (= a cost).

- Exogenous impacts might introduce **stochastic reward components**.
  - Example: A wind gust pushes an autonomous helicopter into a tree.

Reward Examples

Flipping a pancake:

- Pos. reward: catching the flipped pancake
- Neg. reward: dropping the pancake on the floor

Stock trading:

- Trading portfolio monetary value

Playing Atari games:

- Highscore value at the end of a game episode

Driving an autonomous car:

- Pos. reward: getting safely from A to B without crashing
- Neg. reward: hitting another car, pedestrian, bicycle,…

Classical control task (e.g., electric drive, inverted pendulum,…):

- Pos. reward: following a given reference trajectory precisely
- Neg. reward: violating a system constraint or producing a large control error

Issues With the Reward Function

"Be careful what you wish for - you might get it" (proverb)

"…it grants what you ask for, not what you should have asked for or what you intend."
(Norbert Wiener, American mathematician)

Midas and daughter (good as gold)
(source: www.flickr.com, by Robin Hutton CC BY-NC-ND 2.0)

Designing an effective reward function is crucial for successful RL applications!

It is hard to define a good reward function:

- The RL agent's learning process heavily depends on the reward distribution over time.
- Alignment problem and reward hacking.

Definition: Return

Definition: The expected cumulative reward that the agent wants to maximize is called the **return**.

Episodic tasks

- A problem which naturally breaks into subsequences (episodes) with a **finite horizon**.
- Examples: most games like chess, solving a maze,...
- The return becomes a finite sum with horizon T:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T = \sum_{k=1}^{T} R_{t+k}$$

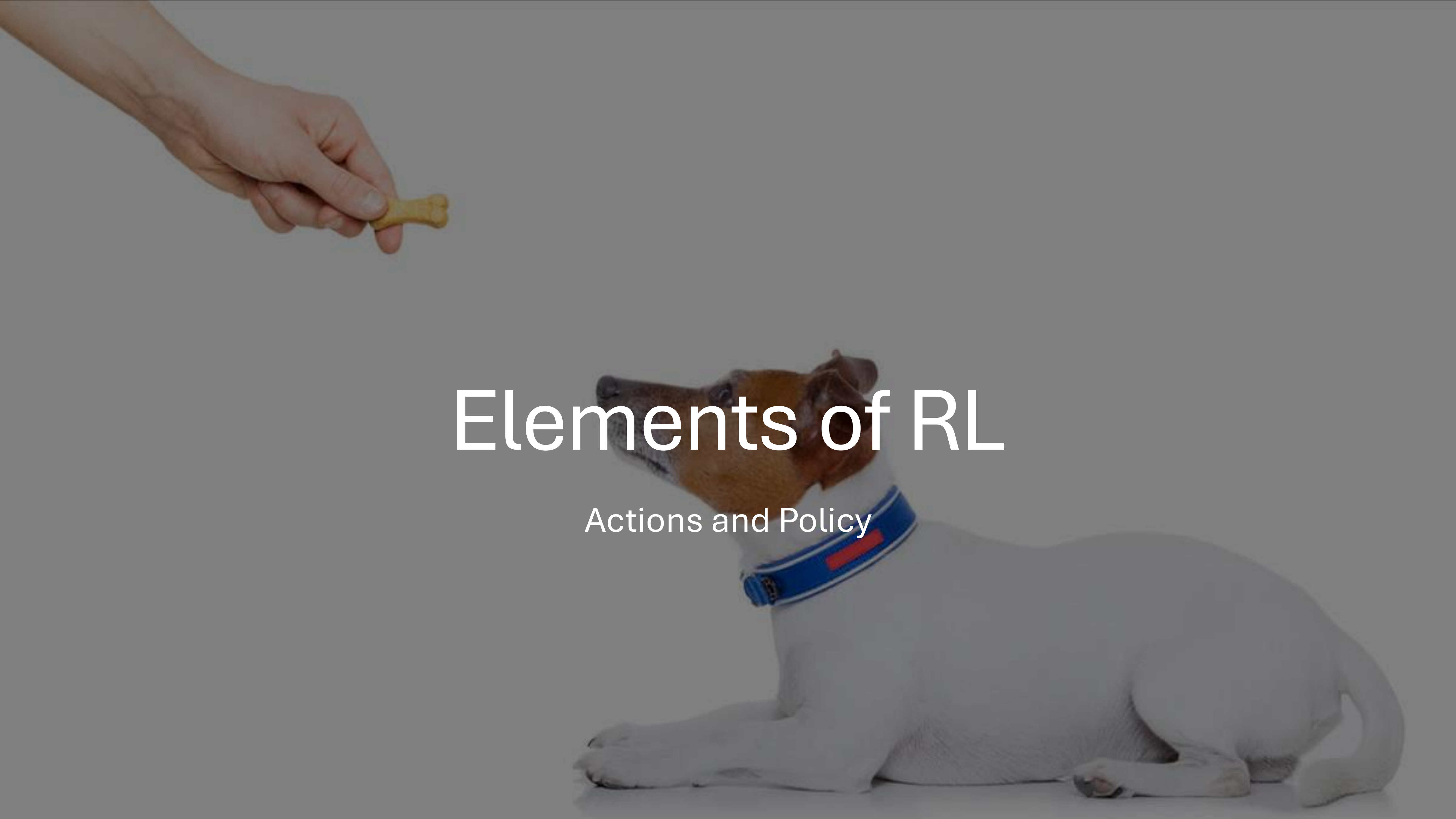
Continuing tasks

- A problem without a natural end (**infinite horizon**).
- Example: A smart thermostat controlling the room temperature
- We use **discounting** and have a **recursive relationship**:

$$\begin{aligned} G_t &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \\ &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\ &= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

- The discount rate $0 \leq \gamma < 1$ is an exponentially decaying weight that prevents infinite sums.

Strategic viewpoint of discounting:
- If $\gamma \approx 1$: agent is farsighted.
- If $\gamma \approx 0$: agent is shortsighted (only interested in immediate reward).



Elements of RL

Actions and Policy

Action

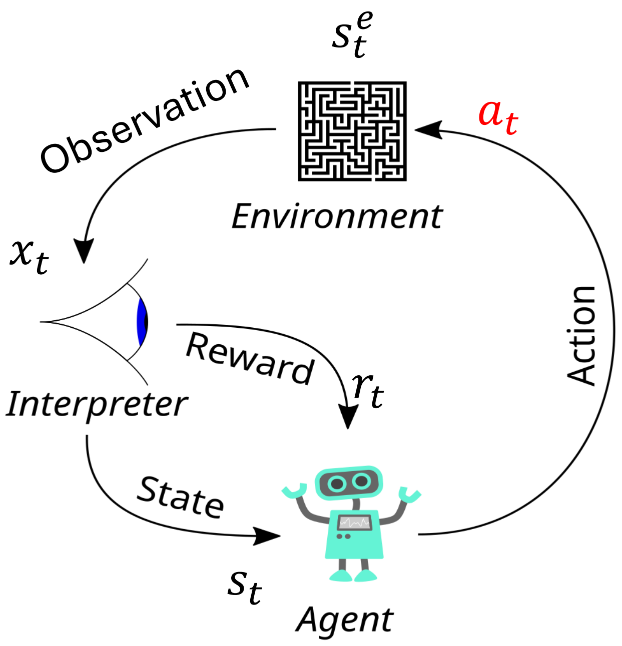
- Choosing actions is the agent's means to affect the environment so it can maximize its long-term reward.

- **Notation**: Random variable A_t or a known action $a_t \in \mathcal{A}$

- **Finite action set (FAS):** $\mathcal{A}$ is a finite set.
  Available actions often depend on the current state $a_t \in \mathcal{A}(s_t)$.

- **Continuous action set (CAS):** $\mathcal{A} = \mathbb{R}^m$
  a_t is an m-dimensional vector. Can be discretized into a FAS.

- Examples:
  - Take another card during a Blackjack game? Yes/No (FAS)
  - Steering while driving an autonomous car. Angle? (CAS)
  - Buy stock options for your trading portfolio (FAS/CAS)



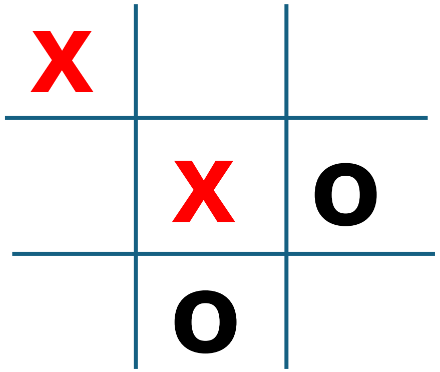
Deterministic Policy

- A policy is the rule of how an agent chooses actions.
- Choice depends on what the agent currently knows about the environment. This is the current state s.
- A deterministic policy prescribes an action for each state:

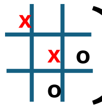
$$\pi: \mathcal{S} \to \mathcal{A} \qquad \text{or} \qquad a = \pi(s)$$

- Example:

State: $s =$



Action a: x always plays bottom right corner.

$\pi($$)$ = play bottom right

Stochastic Policy

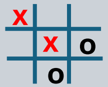
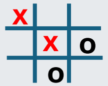
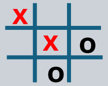
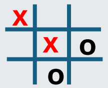
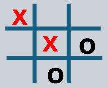
$\Delta(\mathcal{A})$ … Probability distribution over the action space

- A mapping from states to the probability distribution of selecting each action:

$$\pi \colon \mathcal{S} \to \Delta(\mathcal{A})$$

often written as $\pi(a|s)$ giving the probability of choosing a given the agent is in state s.

- **Example:** Probability distribution indicating to play bottom right 60% of the time.

State s	Action a	Probability $\pi(a\|s)$
	Top center	.1
	Top right	.1
	Middle left	.1
	Bottom left	.1
	Bottom right	.6

Note: The deterministic policy is just a special case with 1 for the chosen action and 0 for all others.

Exploration vs. Exploitation

- **Issue**: The agent has to learn a good policy using the reward signal **while it is evaluated** at the same time.
- At any point in time, the agent can choose actions to perform:
  - **Exploitation**: Follow the currently known best policy to maximize the return.
  - **Exploration**: Try new actions to find out more about the environment. This could improve the policy.

- **Trade-off problem**: What is the best split between both strategies?

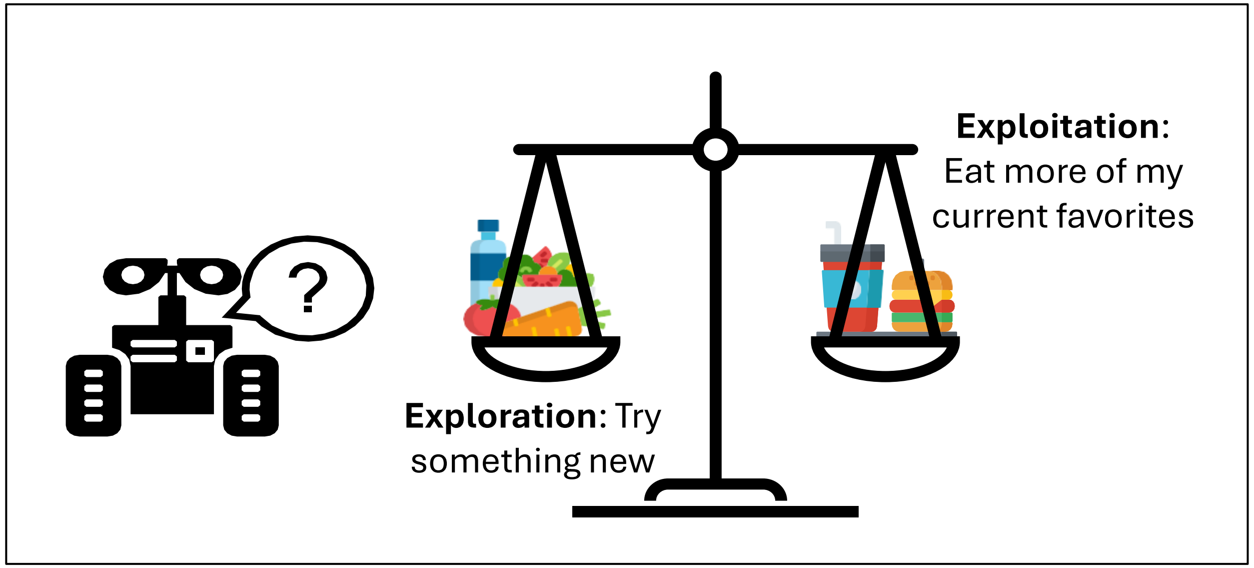
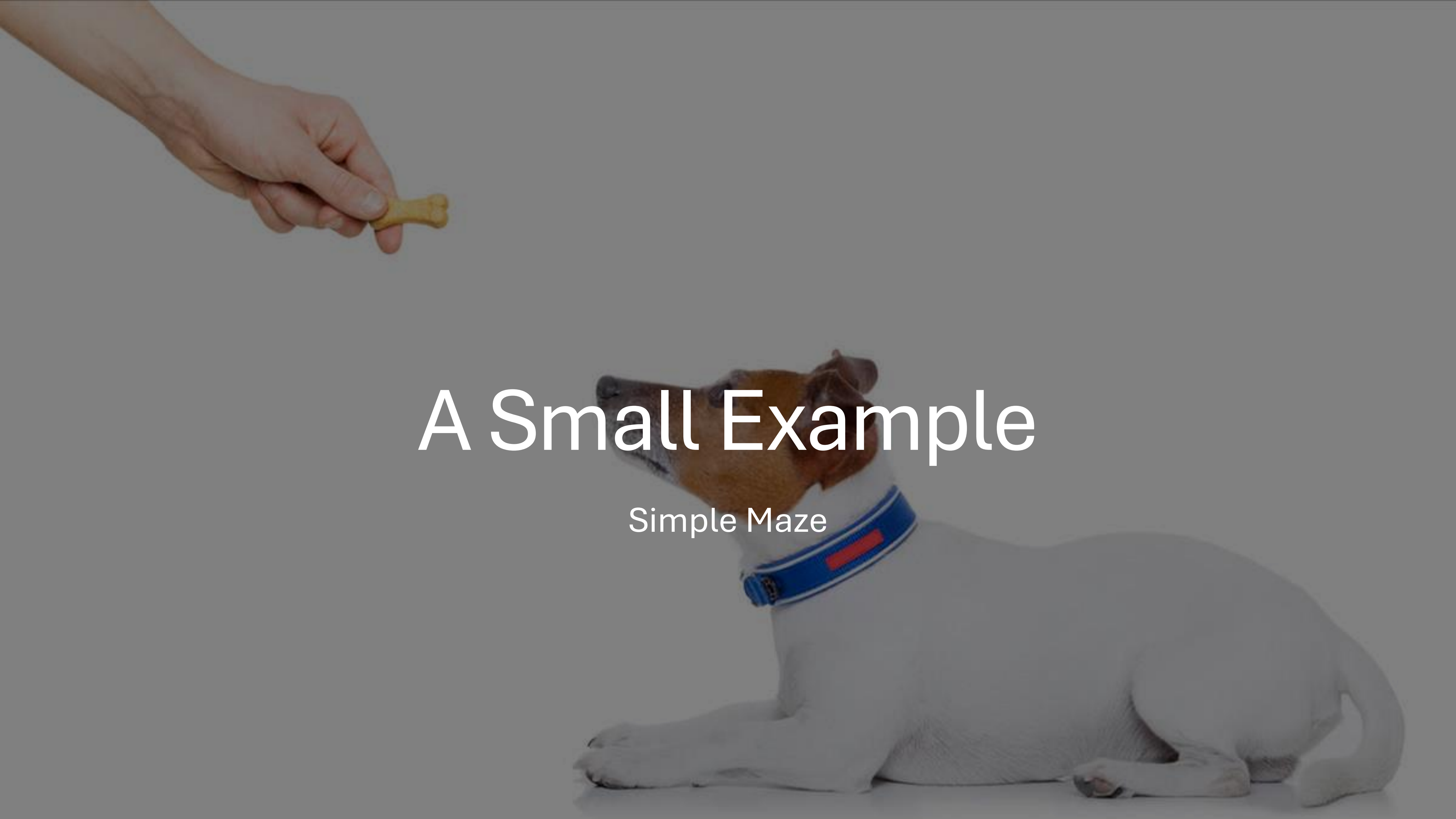


Figure: The exploration-exploitation dilemma

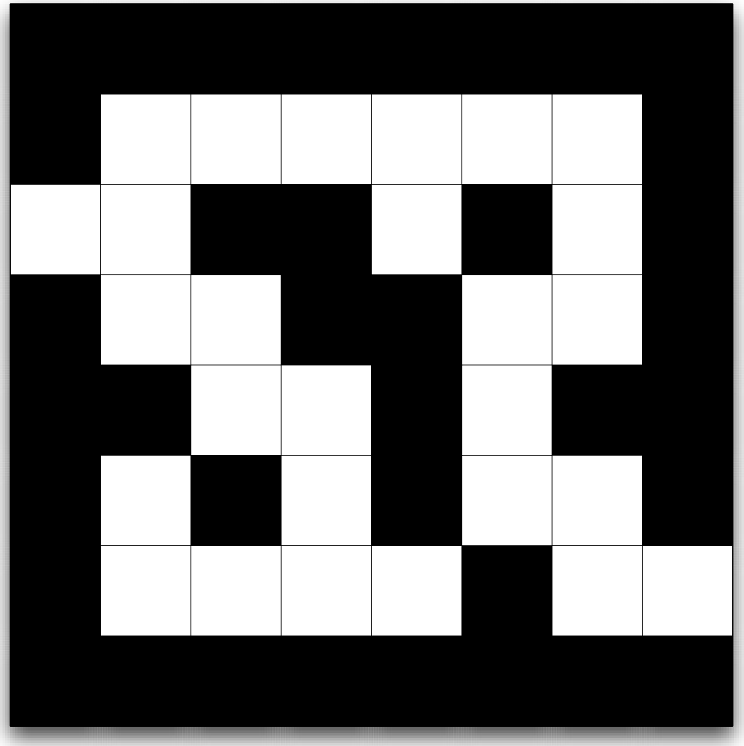
A Small Example

Simple Maze



Maze Example

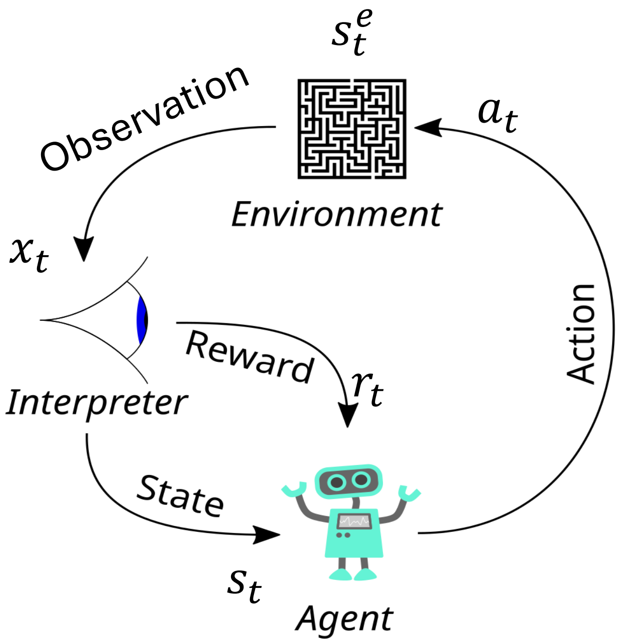
Start



Goal

Maze setup

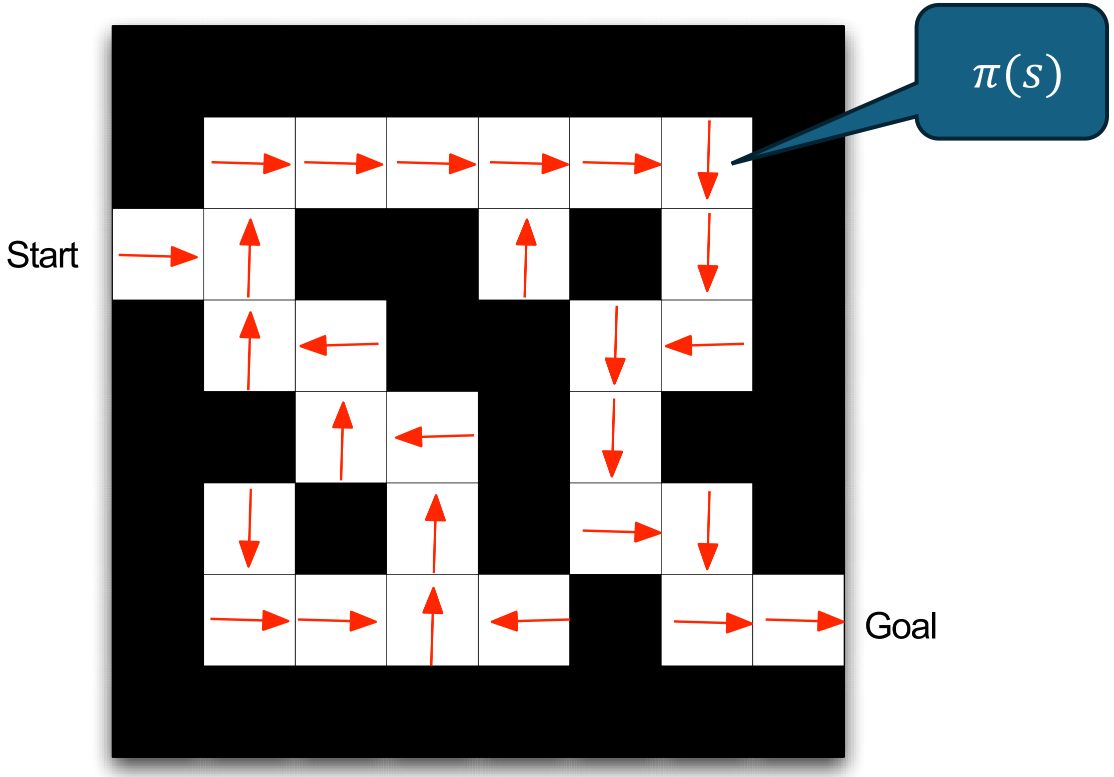
(source: D. Silver, “Reinforcement learning”, 2015. CC BY-NC 4.0)



Problem statement
- State: agent’s location is observable
- Start and goal are given
- Actions: $a_t \in \{N, E, S, W\}$
- Transition model: deterministic
- Immediate Reward: $r_t = -1$
- Episode terminates at goal state

Episodic problem! To maximize the return, it needs to minimize the number of steps to get to the goal!

Maze Example: RL-Solution as a Policy



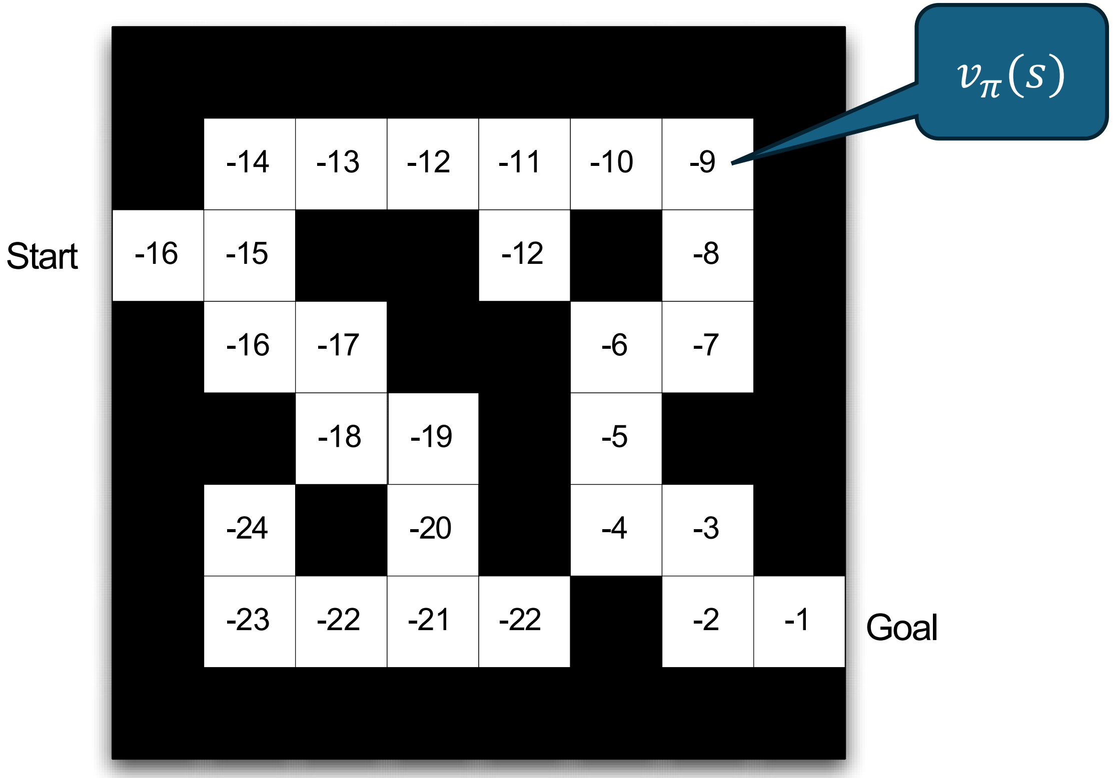
Maze setup: Arrows represent policy $\pi(s)$

(source: D. Silver, "Reinforcement learning", 2015. [CC BY-NC 4.0])

Key characteristics of the policy:

- Specify an action (direction) for each state (position in the maze).
- Policy can be explicitly stored as a table.

Maze Example: RL-Solution as a Value Function



Numbers represent value $v_\pi(s)$: Each step costs 1, i.e. $r = -1$

(source: D. Silver, “Reinforcement learning”, 2015.)

The (state) value function gives the expected return when following a given policy.

$$v_\pi(s)$$

Key characteristics

- The policy is implicitly defined by the state values:
  The agent always chooses actions to move to states with better state values.
  This is called a “greedy” policy.

RL and Planning

Model-based vs. Model-free Methods

Model-based vs. Model-free RL

Model-based RL

- **Known Environment**: The agent has a complete model of the environment.
    - Transition model
    - Reward model
- The agent does not need to interact with the environment: The agent can use the model to **plan** offline and find the optimal policy.
- Important issues:
    - Computational complexity (time and space)
    - Model errors compound

Model-free RL

- **Unknown environment**: Model is unknown.
- The agent needs to interact with the environment: The agent uses online **trial & error** by trying actions and updating its policy based on the received reward.
- Important issues:
    - Sample efficiency (how fast can we learn)
    - Exploitation vs. exploration

Side Note: Optimal Control Theory

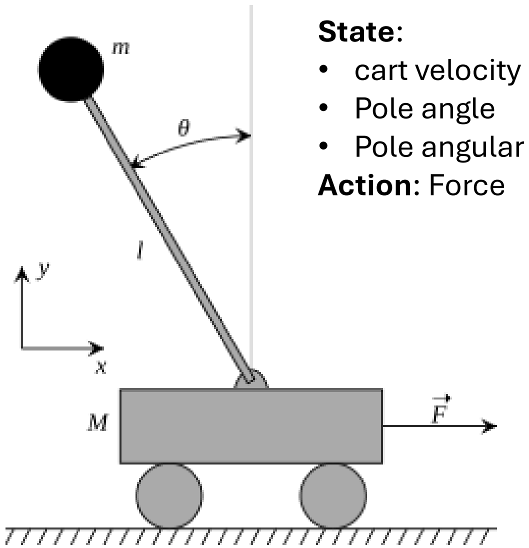
Methods for optimal control of dynamical systems. They are typically

- **model-based**,
- **continuous**,
- solved **offline** (often with a closed-form solution), and
- results in **stable** and robust solutions.

- Example: **Linear Quadratic Regulator (LQR)**
  - Minimize the infinite-horizon quadratic continuous-time cost function:
    $$J = \frac{1}{2}\int_0^\infty [\boldsymbol{x}^\top(t)\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}^\top(t)\boldsymbol{R}\boldsymbol{u}(t)]\,\mathrm{d}t$$
  - With linear time-invariant first-order dynamic constraints
    $$\boldsymbol{x}(t+1) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)$$
  - This optimization problem can be solved to create a feedback control law
    $$\boldsymbol{u}(t) = \boldsymbol{K}\boldsymbol{x}(t)$$

- $\boldsymbol{x}$ is the state (vector)
- $\boldsymbol{u}$ is the action.
- $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the reward (cost) model

- $\boldsymbol{A}$ and $\boldsymbol{B}$ are the transition model.



State:
- cart velocity
- Pole angle
- Pole angular velocity

Action: Force

Example: Cart Pole Balancing
Balance the weight by moving forth and back.

- **Model Predictive Control (MPC):** Similar to optimal control, but
  - reoptimizes online in each step given the feedback
  - uses numerical optimization instead of closed form solutions
  - can deal with non-linear problems.

This course will focus on RL and not Optimal Control Theory or MPC.

Summary: What You Should Know

- Understand the basic **RL interaction loop**: agent, environment, and the role of the interpreter.
- **Basic RL vocabulary**: state, action, immediate reward, return, policy, value function.
- Appreciate the significance of proper **reward formulation**.
- Differentiate between **model-based** (planning) and **model-free** reinforcement learning methods.