LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Praktikum Mobile und Verteilte Systeme

Decision Making in Autonomous Systems

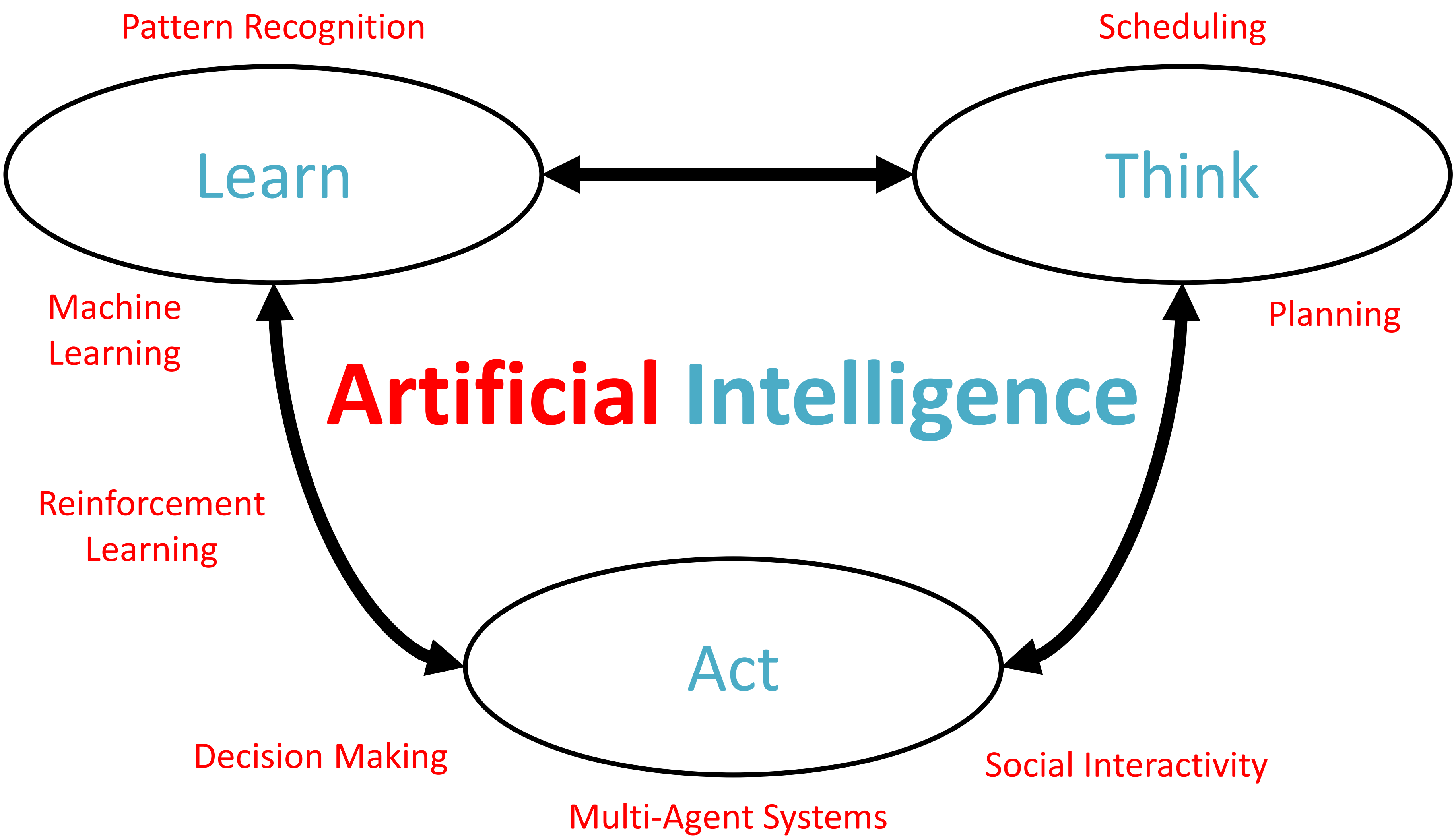
Prof. Dr. Claudia Linnhoff-Popien
André Ebert, Thomy Phan, Robert Müller, Steffen Illium
http://www.mobile.ifi.lmu.de

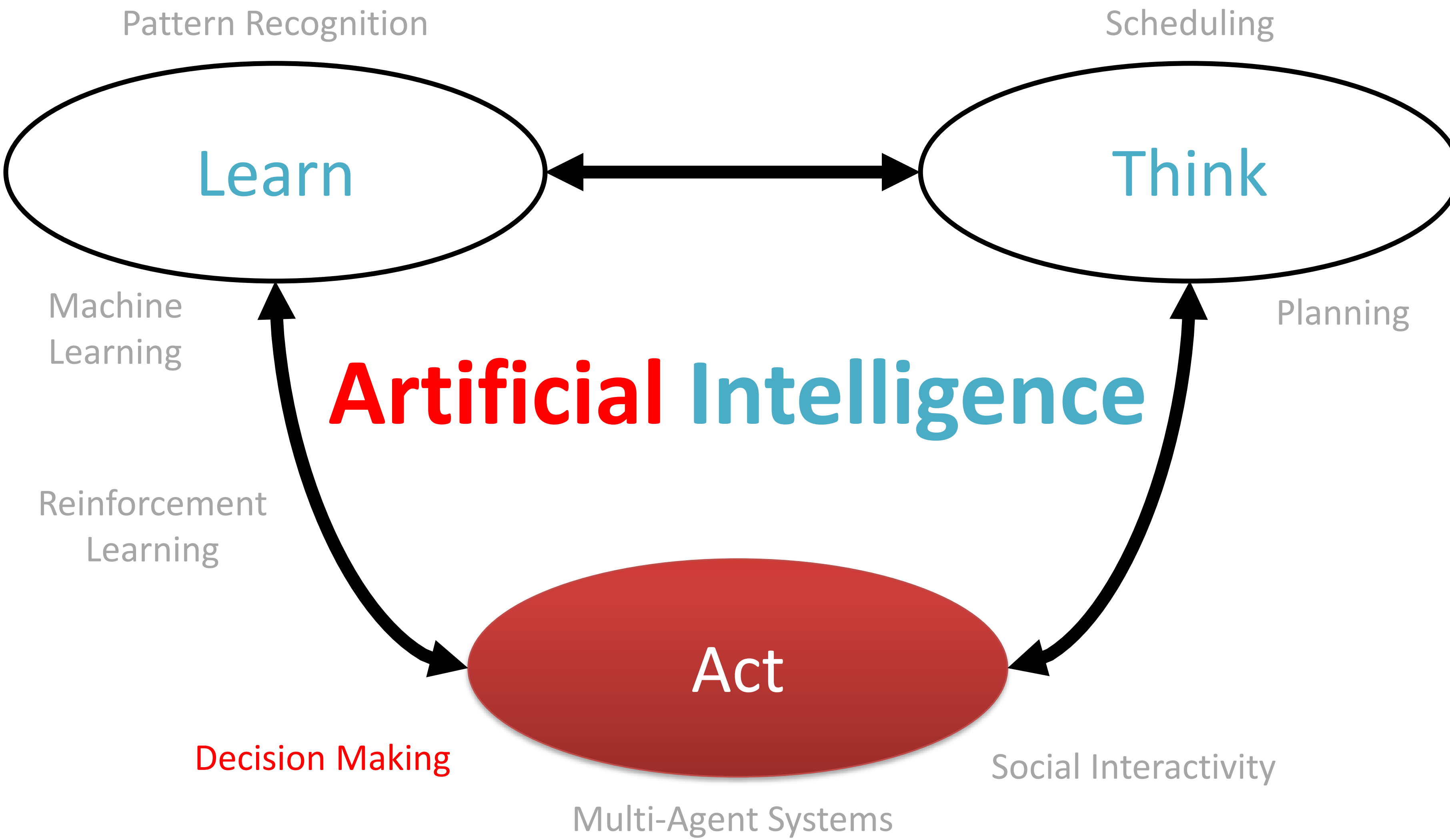
WiSe 2018/19

Outline

- **An Introduction to Autonomous Systems**
  - Motivation, Definition and Challenges
  - Artificial Intelligence
- **Decision Making in Autonomous Systems**
  - Markov Decision Processes
  - Planning
  - Reinforcement Learning
- **Applications and Further Challenges**
  - Deep Reinforcement Learning
  - Combining Planning and Reinforcement Learning
  - Further Challenges

→ Short Recap


Pattern Recognition
Scheduling
Learn
Think
Machine Learning
Planning
Artificial Intelligence
Reinforcement Learning
Act
Decision Making
Social Interactivity
Multi-Agent Systems



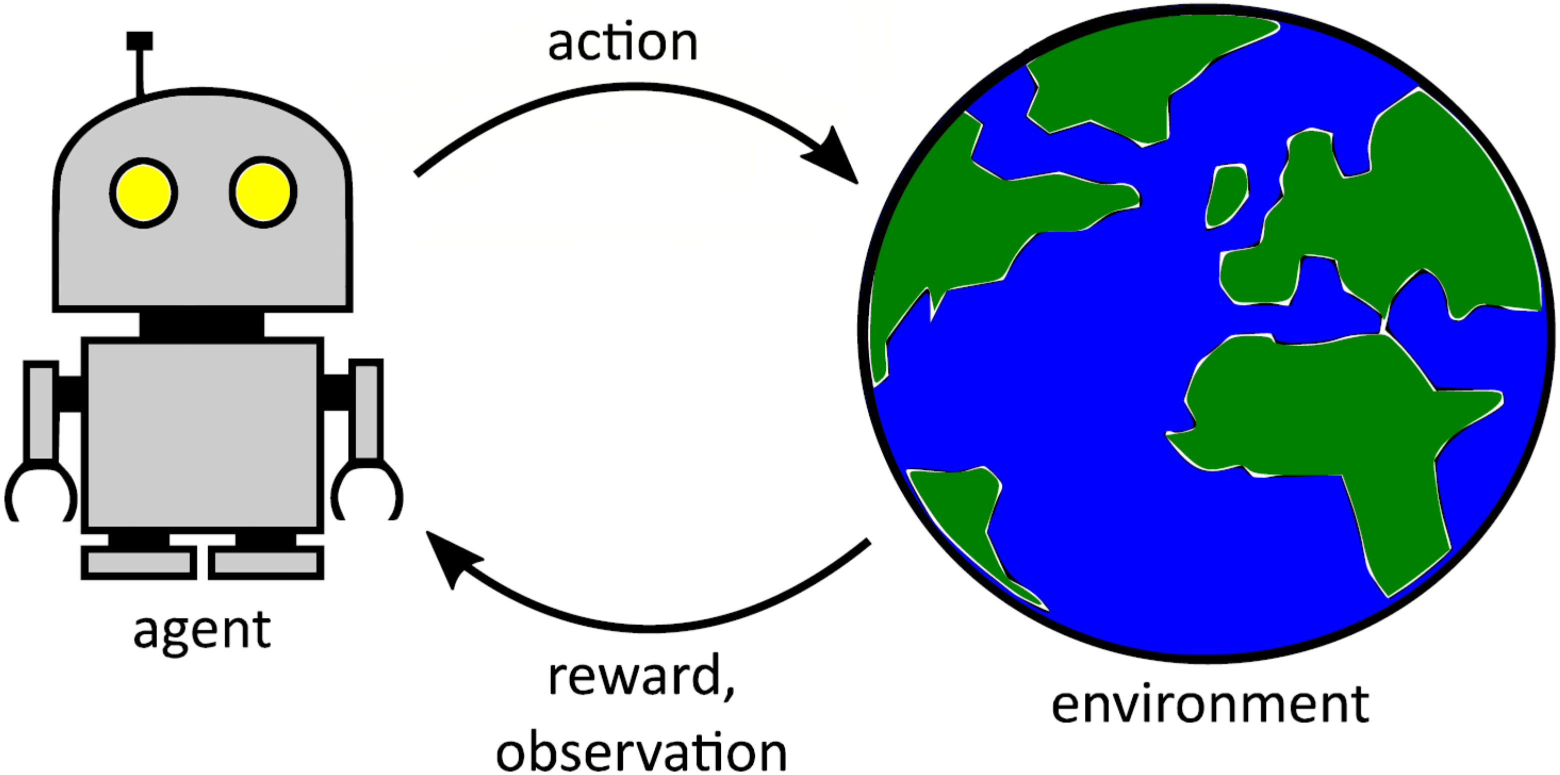
Pattern Recognition
Scheduling
Learn
Think
Machine Learning
Planning
Artificial Intelligence
Reinforcement Learning
Act
Decision Making
Social Interactivity
Multi-Agent Systems

Outline

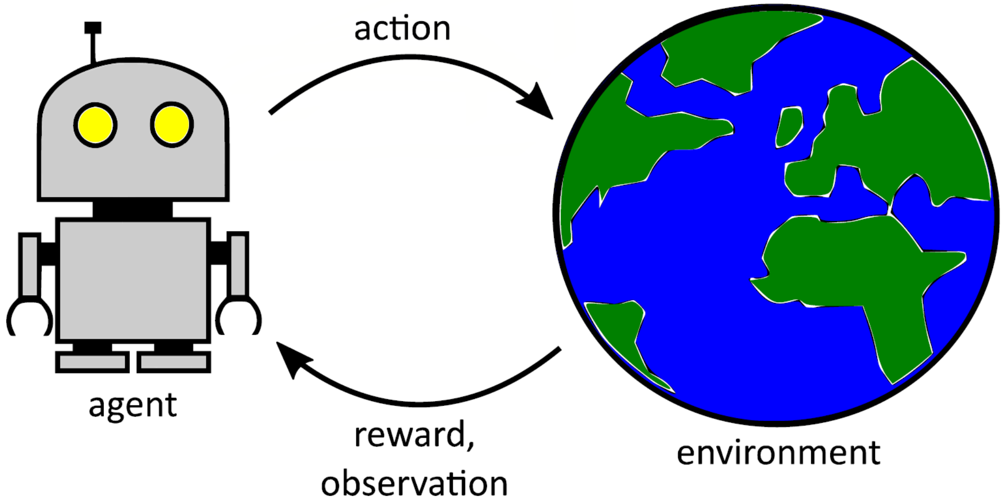
- **An Introduction to Autonomous Systems**
  - Motivation, Definition and Challenges
  - Artificial Intelligence
- **Decision Making in Autonomous Systems**
  - Markov Decision Processes
  - Planning
  - Reinforcement Learning
- **Applications and Further Challenges**
  - Deep Reinforcement Learning
  - Combining Planning and Reinforcement Learning
  - Further Challenges

→ Decision Making Problem

Decision Making



Decision Making



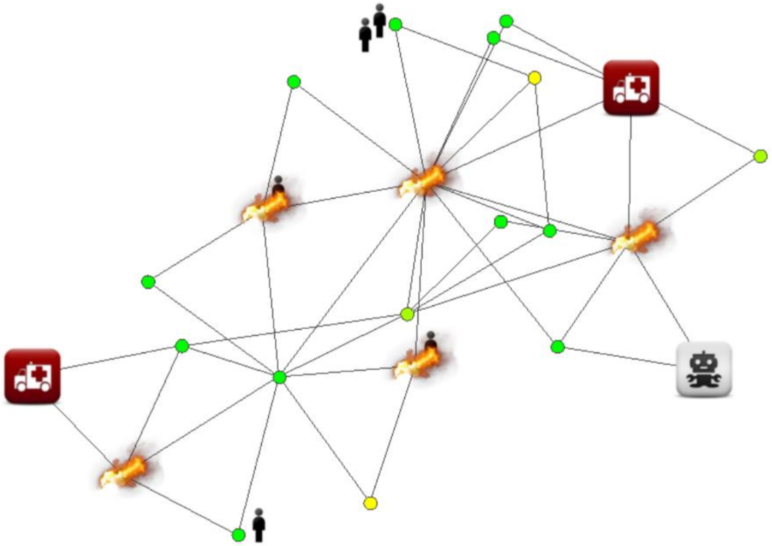
- **Goal:** Select actions to solve a complex task
- Focus on **Sequential Decision Making**: Time matters!
- Actions have possible consequences in the long run
- Feedback/Reward may be delayed

Why Decision Making?

- High Complexity (autonomous systems instead of hand-coded solutions)
- Risk and Safety (autonomous systems instead of living beings)
- Dynamics (adaptation to changes)
- **Examples:**

Industry 4.0

Search and Rescue

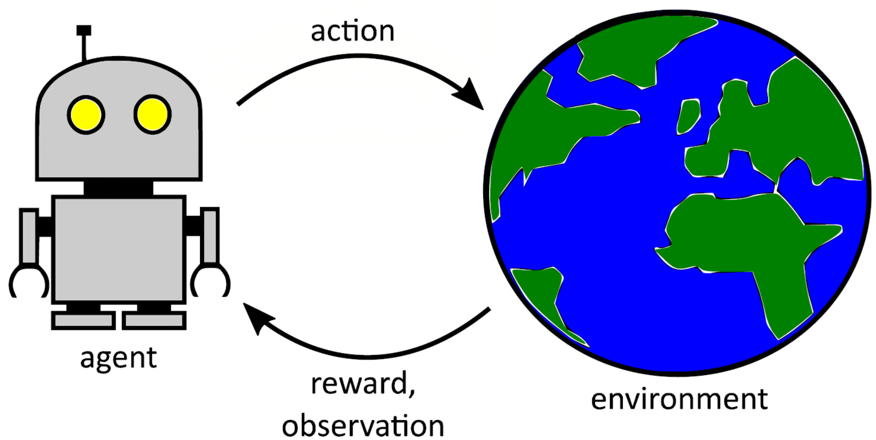


Lenz Belzner: Simulation-Based Autonomous Systems in Discrete and Continuous Domains, PhD Thesis, LMU Munich, 2016

→ Markov Decision Processes

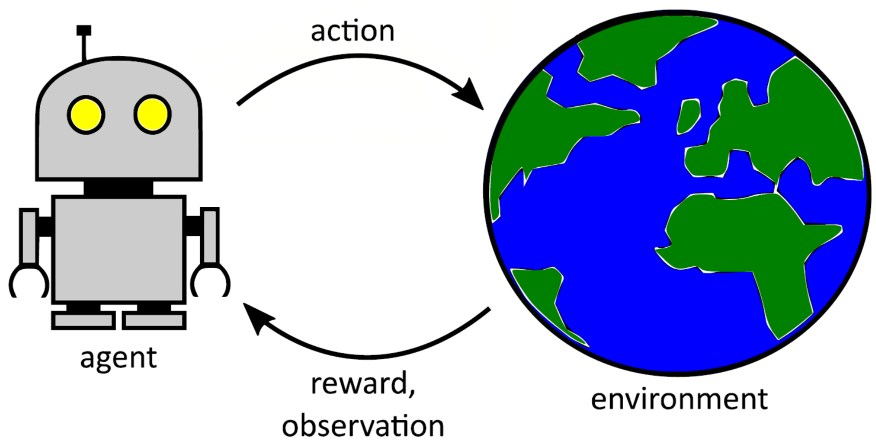
Definition

- A Markov Decision Process (MDP) is defined as $\mathrm{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$:
  - $\mathcal{S}$ is a (finite) set of states
  - $\mathcal{A}$ is a (finite) set of actions
  - $\mathcal{P}(s_{t+1}|s_t, a_t) \in [0, 1]$ is the probability for reaching $s_{t+1} \in \mathcal{S}$ when executing $a_t \in \mathcal{A}$ in $s_t \in \mathcal{S}$
  - $\mathcal{R}(s_t, a_t) \in \mathbb{R}$ is a reward function



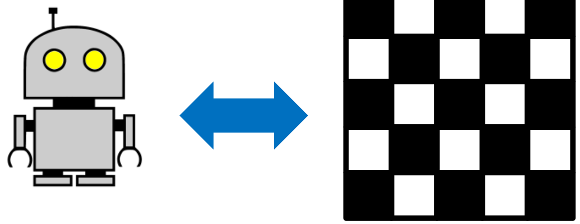
Markov Decision Processes

- MDPs formally describe environments for Sequential Decision Making
- All states $s_t \in \mathcal{S}$ are **Markov** such that
  $\mathbb{P}(s_{t+1}|s_t) = \mathbb{P}(s_{t+1}|s_1, \dots, s_t)$ (no history of past states required)
- Assumes **full observability** of the state
- States and actions may be **discrete** or **continuous**
- Many problems can be formulated as MDPs!

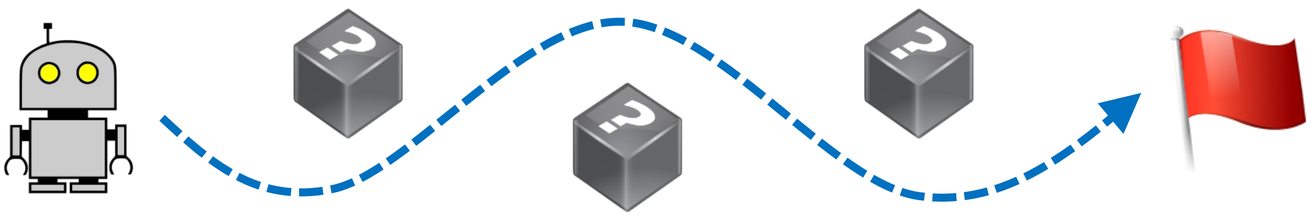


Examples

- Board Games
  - Board positions represent states $s_t \in \mathcal{S}$
  - Moves represent actions $a_t \in \mathcal{A}$
  - Game dynamics (opponents, rules, etc.) represent $\mathcal{P}(s_{t+1}|s_t, a_t)$
  - $\mathcal{R}(s_t, a_t)$ defines when a game is won, lost or a draw.



- Navigation Task
  - Agent‘s and other object‘s positions represent the state $s_t \in \mathcal{S}$
  - Agent moves represent actions $a_t \in \mathcal{A}$
  - Environment dynamics represent $\mathcal{P}(s_{t+1}|s_t, a_t)$
  - $\mathcal{R}(s_t, a_t)$ defines when a target is reached



Policies and Value Functions

- A **policy** $\pi: \mathcal{S} \rightarrow \mathcal{A}$ represents the behavioural strategy of an agent
  - Policies may also be stochastic $\pi(a_t|s_t) \in [0,1]$
- The **return** of a state $s_t \in \mathcal{S}$ for a horizon h given a policy π is the cumulative (discounted) future reward (h may be infinite!):

$$G_t = \sum_{k=0}^{h-1} \gamma^k \, \mathcal{R}\big(s_{t+k}, \pi(s_{t+k})\big), \gamma \in [0,1]$$

- The **value** of a state $s_t \in \mathcal{S}$ is the expected return of s_t for a horizon $h \in \mathbb{N}$ given a policy π:

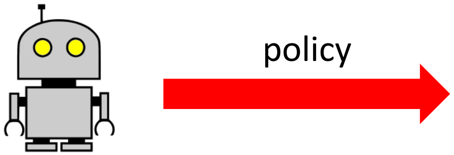
$$V^{\pi}(s_t) = \mathbb{E}[G_t|s_t]$$

- The **action value** of a state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$ is the expected return of executing a_t in s_t for a horizon $h \in \mathbb{N}$ given a policy π:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}[G_t|s_t, a_t]$$

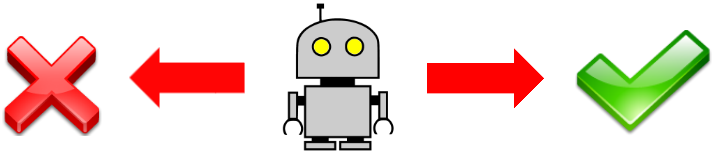
Policies and Value Functions

- The policy π describes **what** an agent should do in a certain state.
  - Recommendation of actions



- The state value function V^{π} describes **how well** an agent will perform in a **certain state** (with policy π).
  - State evaluation

- The action value function Q^{π} describes **how well** agent will perform when executing a **certain action** in a certain state (with subsequent policy π).
  - Action evaluation



Policies and Value Functions

- **Goal:** Find an *optimal policy* π^* which maximizes the expected return for each state:

$$\pi^* = argmax_\pi V^\pi(s_t), \forall s_t \in \mathcal{S}$$

- The *optimal value function* is defined by:

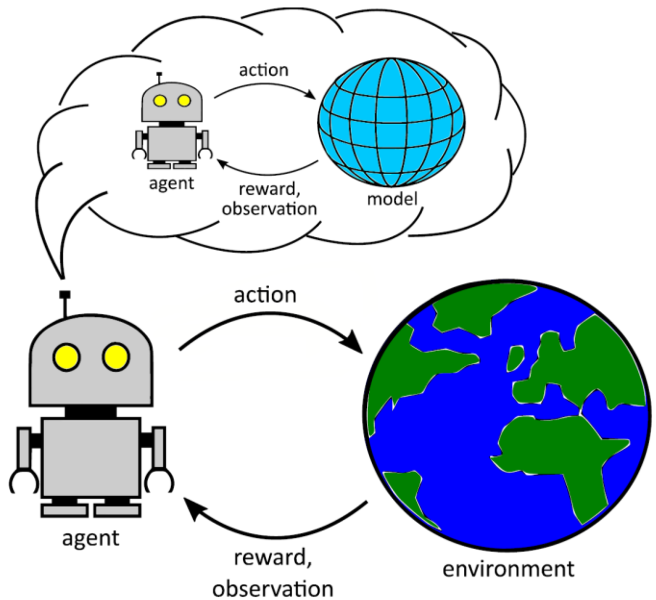
$$V^*(s_t) = V^{\pi^*}(s_t) = max_\pi V^\pi(s_t)$$
$$Q^*(s_t, a_t) = Q^{\pi^*}(s_t, a_t) = max_\pi Q^\pi(s_t, a_t)$$

- When V^* or Q^* is known, π^*can be derived.

→ Planning

Planning

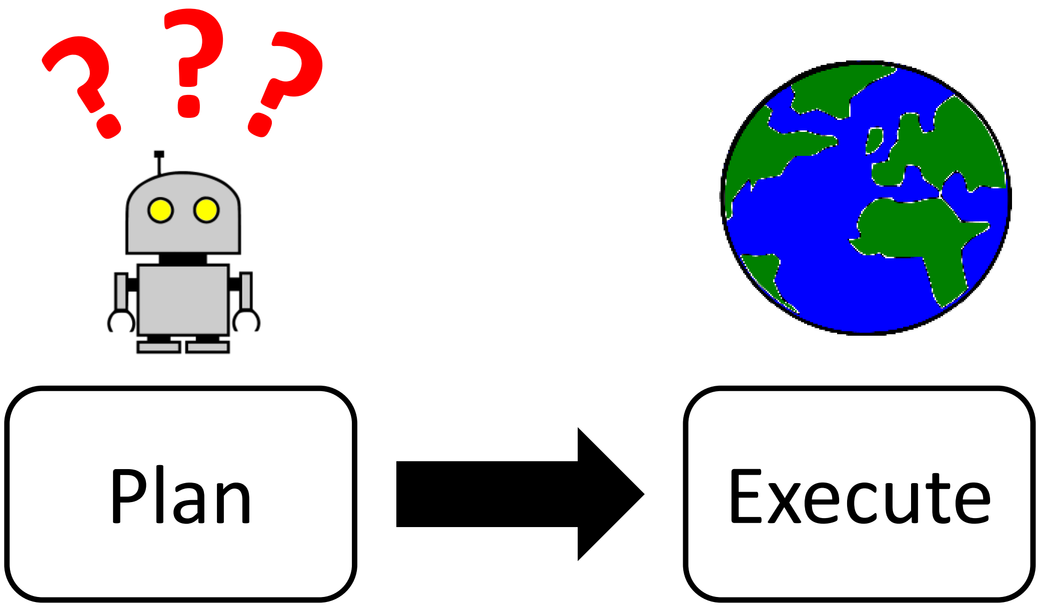
- **Goal:** Find an (near-)optimal policy to solve complex problems



- Use (heuristic) lookahead search on a **given model** of the problem
  - Model can be defined by rules (e.g. physical laws, game rules, etc.) or statistical approximations (e.g. using machine learning)
  - Represented by $\mathcal{P}(s_{t+1}|s_t, a_t)$ in the context of MDPs

Global Planning

- Global Planning considers the **entire state space** $\mathcal{S}$ to approximate π^*
- Produces for each state $s_t \in \mathcal{S}$ a mapping to actions $a_t \in \mathcal{A}$
- Typically performed **offline** (before deploying the agent)



Example: Value Iteration

- Starting with an initial guess V^0
- Refine approximation V^n for each state $s_t \in \mathcal{S}$ according to

$$V^{n+1}(s_t) = \max_{a_t \in \mathcal{A}} \{\mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1 \in \mathcal{S}}} \mathcal{P}(s_{t+1}|s_t, a_t) V^n(s_{t+1})\}$$

- Based on Bellman‘s „principle of optimality“
- Converges to the **optimal value function** V^*

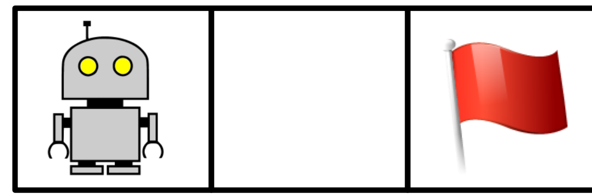
- **Optimal policy** π^* can be derived from V^*

$$\pi^*(s_t) = argmax_{a_t \in \mathcal{A}} \{\mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1 \in \mathcal{S}}} \mathcal{P}(s_{t+1}|s_t, a_t) V^*(s_{t+1})\}$$

- Requires explicit model $\mathcal{P}(s_{t+1}|s_t, a_t)$!

Example: Value Iteration

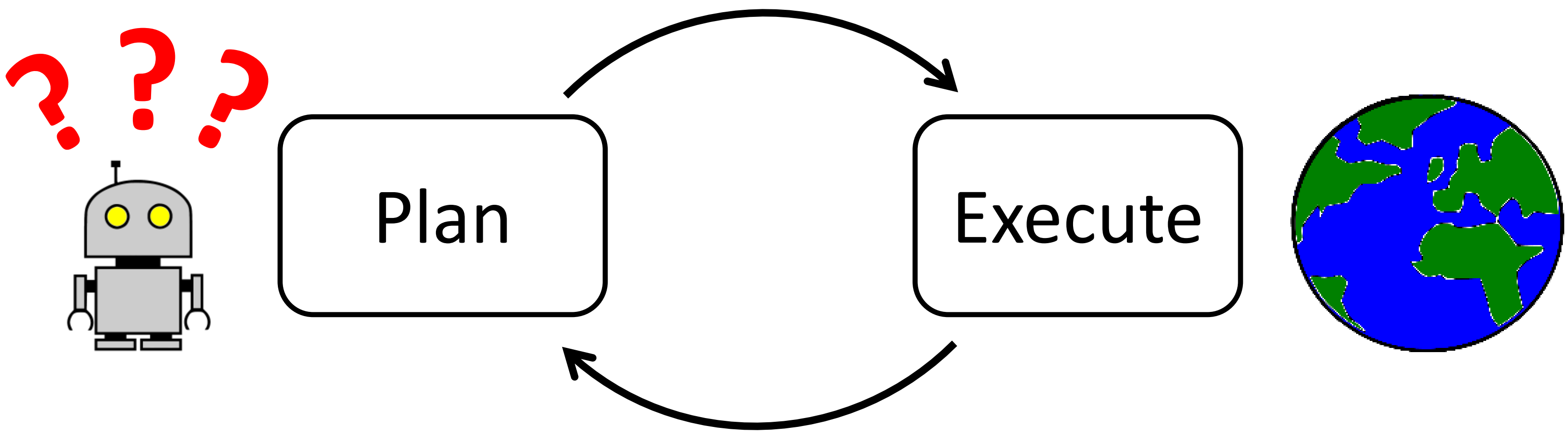
- Simple navigation problem:
  - 3 possible positions / states in $\mathcal{S}$
  - Possible actions $\mathcal{A}$: move left/right
  - state transitions are deterministic
  - Reward $\mathcal{R}(s_t, a_t)$:
    - +1, if flag is reached (terminal state)
    - -1, if agent bumped against wall
    - 0 otherwise
  - Discount factor $\gamma = 0.95$
- Optimal value (function) map is V^*:



0.9025 $\gamma^2(+1)$	0.95 $\gamma^1(+1)$	1 $\gamma^0(+1)$

Local Planning

- Local Planning only considers the **current state** $s_t \in \mathcal{S}$ (and possible future states) to approximate $\pi^*(s_t)$
- Can be performed **online** (interleaving planning and execution)

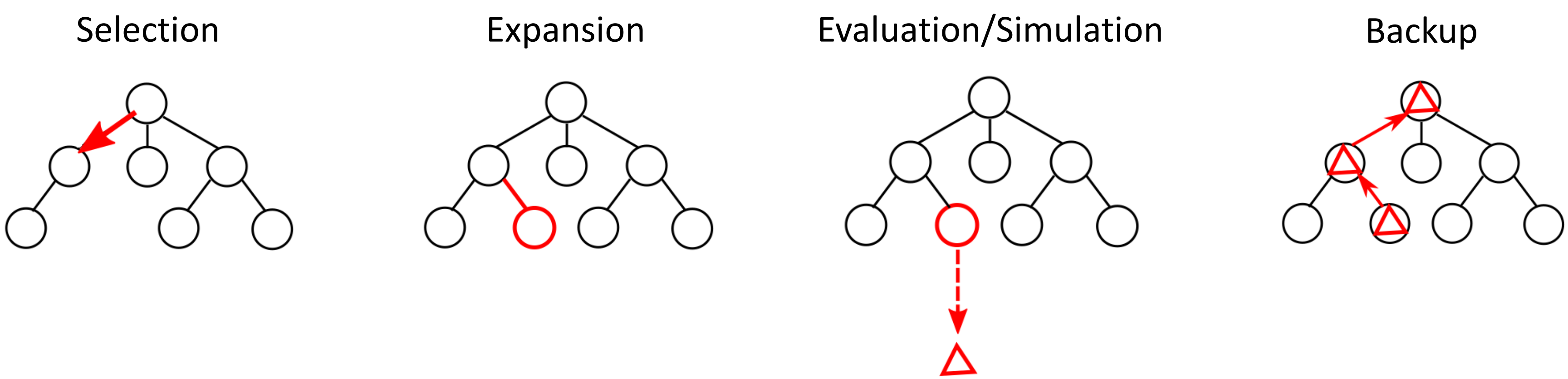


- Typically constrained **time** and **computation budget** (overall performance depends on available resources)

Example: Monte Carlo Planning

- Explicit model $\mathcal{P}(s_{t+1}|s_t, a_t)$ usually unknown for many MDPs
- **Monte Carlo Planning**: uses a generative model $\widehat{M} \approx M$
  - $\widehat{M}$ provides a sample $\langle s_{t+1}, r_t \rangle \sim \widehat{M}(s_t, a_t)$ for $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$
  - Can be used as black box simulator to approximate V^* or Q^*
  - Approximate planning via statistical sampling
  - Requires minimal domain knowledge ($\widehat{M}$ can be easily replaced)

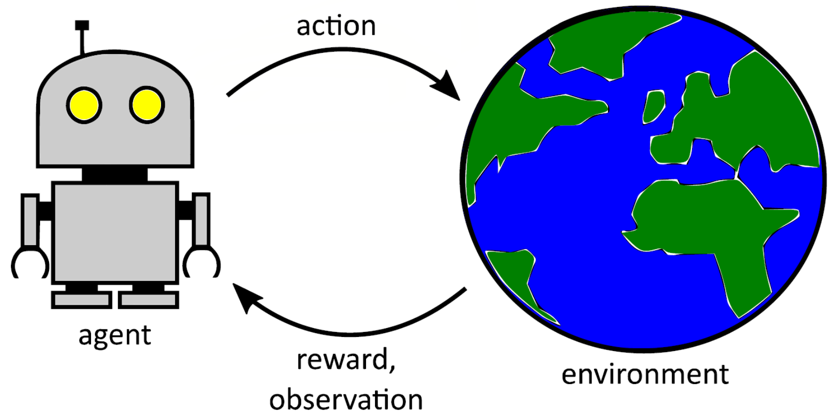
- **Example:** Monte Carlo Tree Search



→ Reinforcement Learning

Reinforcement Learning

- **Goal:** Find an (near-)optimal policy to solve complex problems



- Learn via trial-error from (real) experience:
  - Model $\mathcal{P}(s_{t+1}|s_t, a_t)$ is unknown
  - **Experience samples** $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$ are generated by interacting with a (real or simulated) environment
  - To obtain sufficient experience samples one has to balance between **exploration** and **exploitation** of actions

Model-Free Reinforcement Learning

- Learn policy and/or value function **directly** from experience samples
- Experience samples $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$ are generated by interacting with a (real or simulated) environment
- Approximate π^*, V^* and/or Q^* with experience samples

- **Example:** Temporal Difference (TD) Learning
  - Recap: Value Iteration – iteratively refine guess V^n by using a model

$$V^{n+1}(s_t) = \max_{a_t \in \mathcal{A}} \{ r_t + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1} | s_t, a_t) V^n(s_{t+1}) \}$$

model required!

  - TD-Learning: regression on TD targets y_t generated from experience

$$y_t = \hat{V}^{n+1}(s_t) = r_t + \gamma \hat{V}^n(s_{t+1})$$

Example: Q-Learning

- Deriving a policy $\hat{\pi}$ from $\hat{V}$ still requires a model:

$$\hat{\pi}(s_t) = argmax_{a_t \in \mathcal{A}} \{r_t + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1}|s_t, a_t) \hat{V}(s_{t+1})\}$$

model required!

- **Solution:** approximate Q^* instead!
  - Regression on TD targets y_t generated with $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$

$$y_t = \hat{Q}^{n+1}(s_t, a_t) = r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} \hat{Q}^n(s_{t+1}, a_{t+1})$$

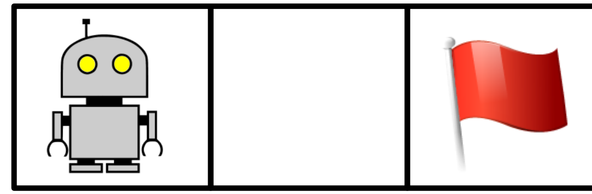
  - Derive a policy $\hat{\pi}$ from $\hat{Q}$

$$\hat{\pi}(s_t) = argmax_{a_t \in \mathcal{A}}(\hat{Q}(s_t, a_t))$$

  - **Challenge:** obtain sufficient samples $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$ by addressing the *exploration-exploitation dilemma.*

Example: Q-Learning

- Simple navigation problem:
  - 3 possible positions / states in $\mathcal{S}$
  - Possible actions $\mathcal{A}$: move left/right
  - state transitions are deterministic
  - Reward $\mathcal{R}(s_t, a_t)$:
    - +1, if flag is reached (terminal state)
    - -1, if agent bumped against wall
    - 0 otherwise
  - Discount factor $\gamma = 0.95$



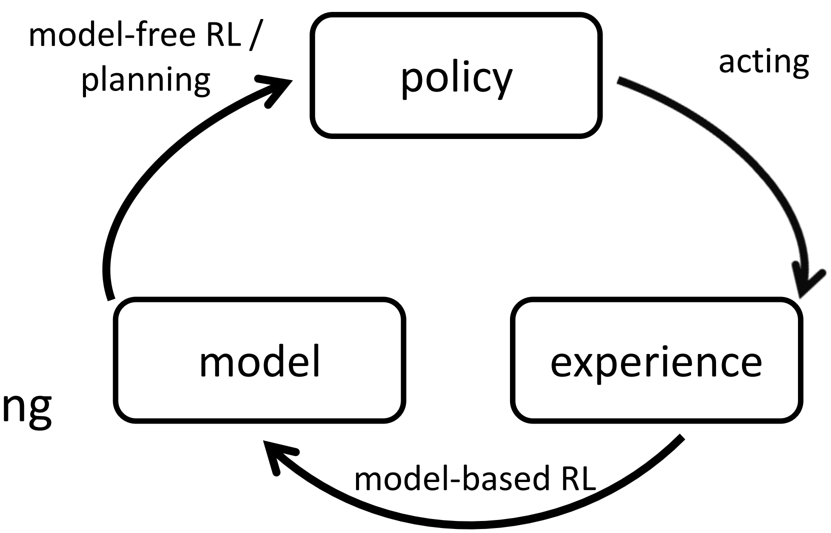
- Optimal Q-values Q^* for given state s_t (see above):

$$Q^*(s_t, a_t = \text{"move left"}) = \mathcal{R}(s_t, a_t) + \gamma V^*(s_{t+1}) = -0.142625$$

$$Q^*(s_t, a_t = \text{"move right"}) = \mathcal{R}(s_t, a_t) + \gamma V^*(s_{t+1}) = 0.9025$$

Model-Based Reinforcement Learning

- Learn a model $\hat{M} \approx M$ from experience samples
- Experience samples $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$ are obtained from interaction with the real environment
- Approximate $\mathcal{P}(s_{t+1}|s_t, a_t)$ (and $\mathcal{R}(s_t, a_t)$) with **supervised learning**
  - Learn $\langle s_t, a_t \rangle \rightarrow s_{t+1}$ via density estimation
  - Learn $\langle s_t, a_t \rangle \rightarrow r_t$ via regression



- Learned model $\hat{M}$ can be used for:
  - Planning
  - Model-free reinforcement learning

→ Case Study: Rockclimbing

How would you try to climb up a rock wall?

- Trial-and-Fall
- Watch how others do it
- Plan own moves based on wall
- Some combination

What is the most intelligent way to you?

How would you try to climb up a rock wall?

- Trial-and-Fall
  - (On-Policy) Reinforcement Learning
- Watch how others do it
  - (Off-Policy) Reinforcement Learning
- Plan own moves based on wall
  - Offline Planning (plan all moves before climbing)
  - Online Planning (plan next moves while climbing)
- Some combination
  - Use own experience or experience of others to plan ahead

What is the most intelligent way to you?

Outline

- **An Introduction to Autonomous Systems**
  - Motivation, Definition and Challenges
  - Artificial Intelligence
- **Decision Making in Autonomous Systems**
  - Markov Decision Processes
  - Planning
  - Reinforcement Learning
- **Applications and Further Challenges**
  - Deep Reinforcement Learning
  - Combining Planning and Reinforcement Learning
  - Further Challenges

Thank you!