LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

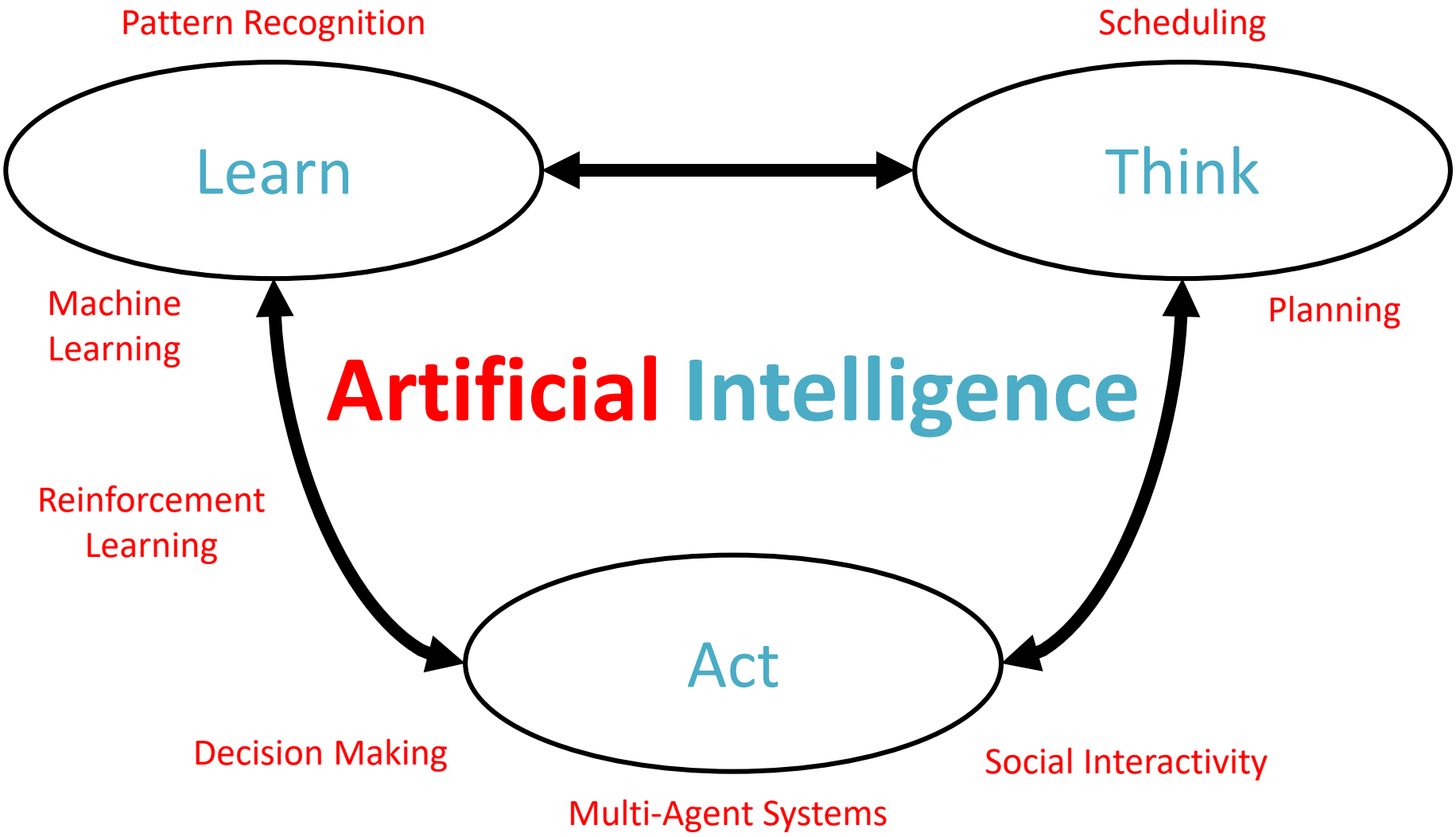
Praktikum Autonome Systeme

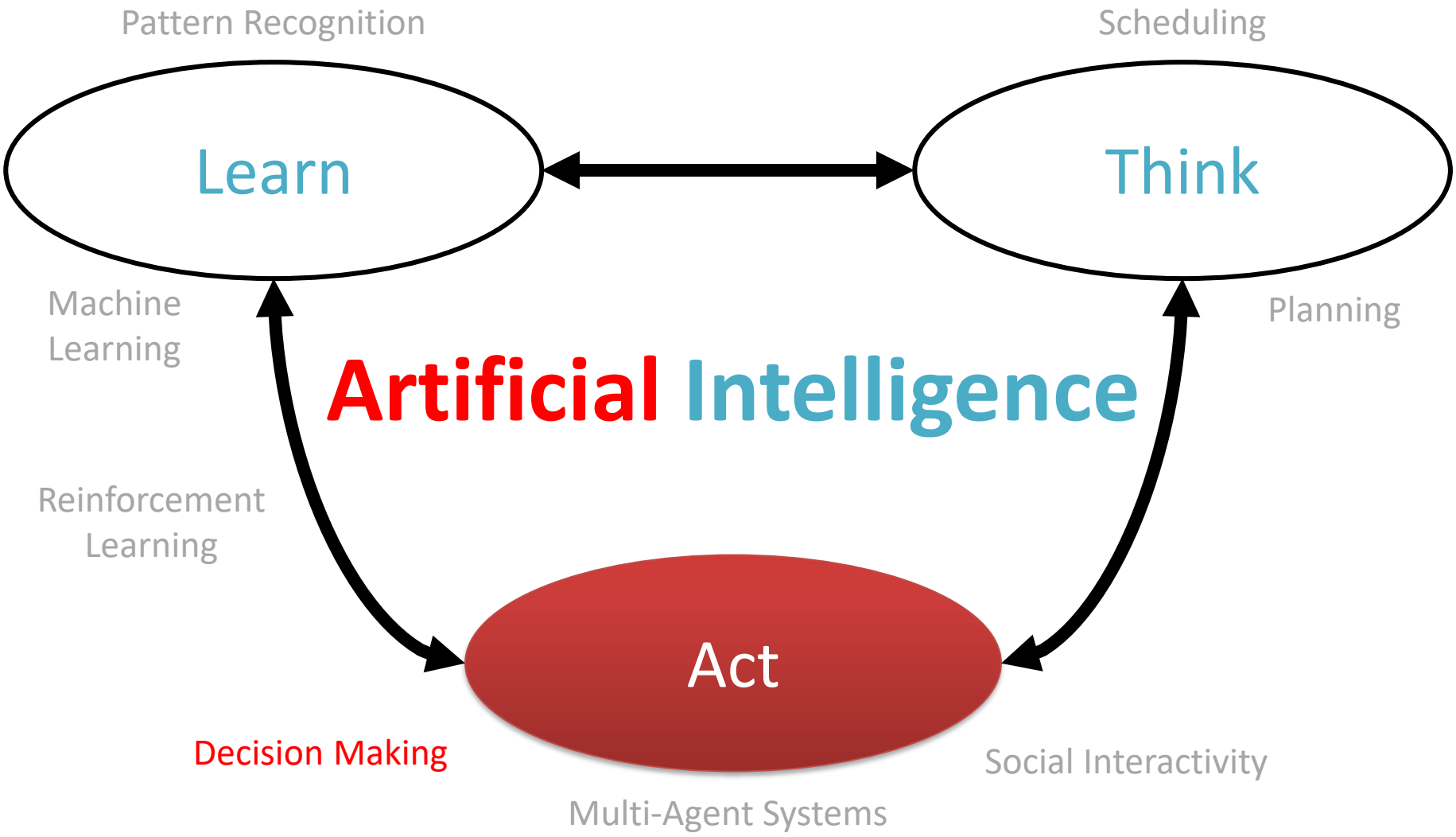
# Automated Planning

Prof. Dr. Claudia Linnhoff-Popien
Thomy Phan, Andreas Sedlmeier, Fabian Ritz
http://www.mobile.ifi.lmu.de

WiSe 2019/20

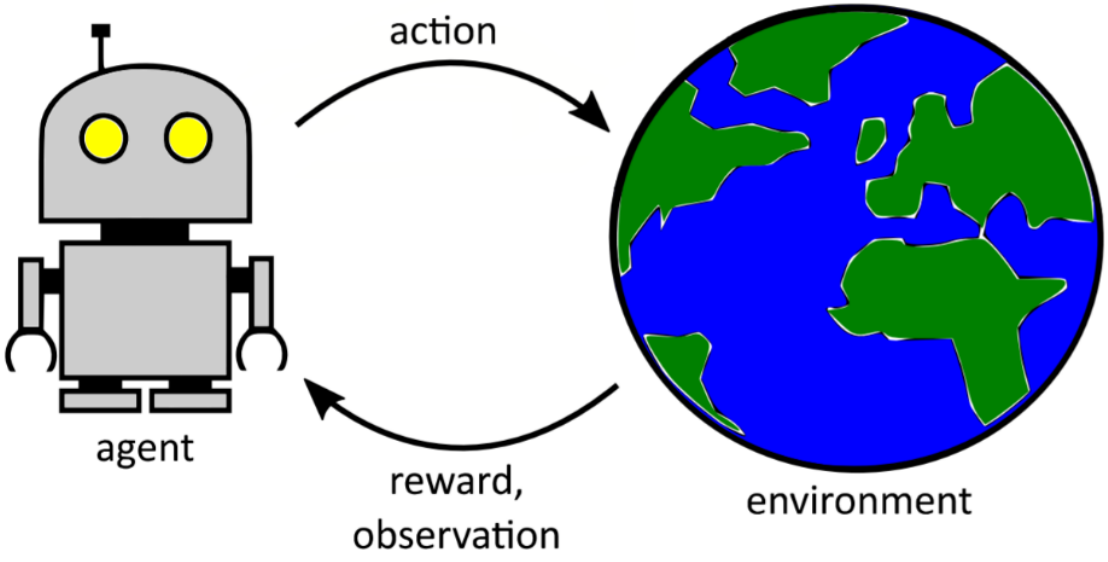
# → Recap: Decision Making


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

# Decision Making

- **Goal:** Autonomously select actions to solve a (complex) task
  - time could be important (but not necessarily)
  - maximize the **expected reward** for each state



# Multi-Armed Bandits

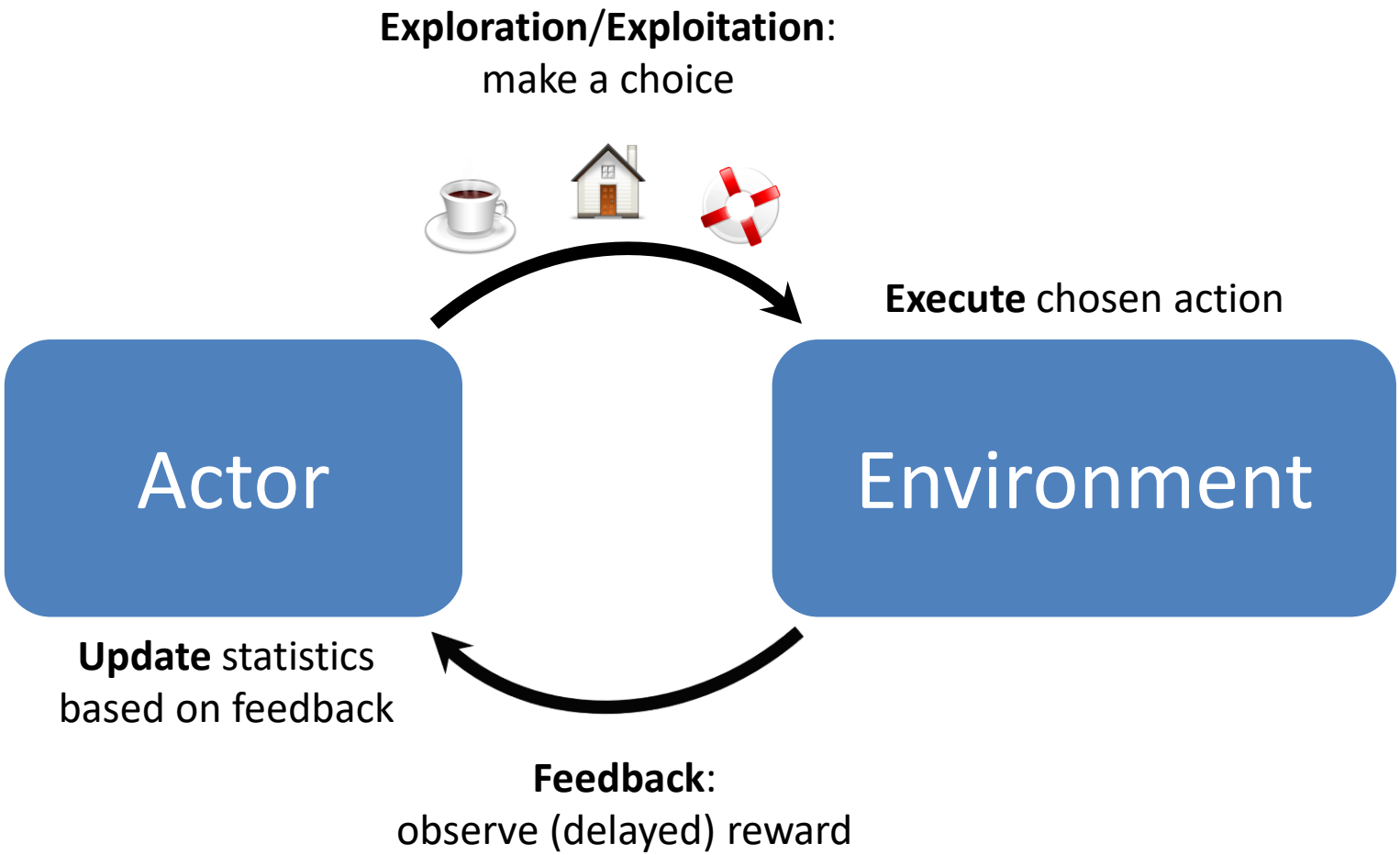
- **Multi-Armed Bandit**: situation, where you have to learn how to make a good (long-term) choice

- **Explore** choices to gather information (= Exploration)
  - Example: random choice

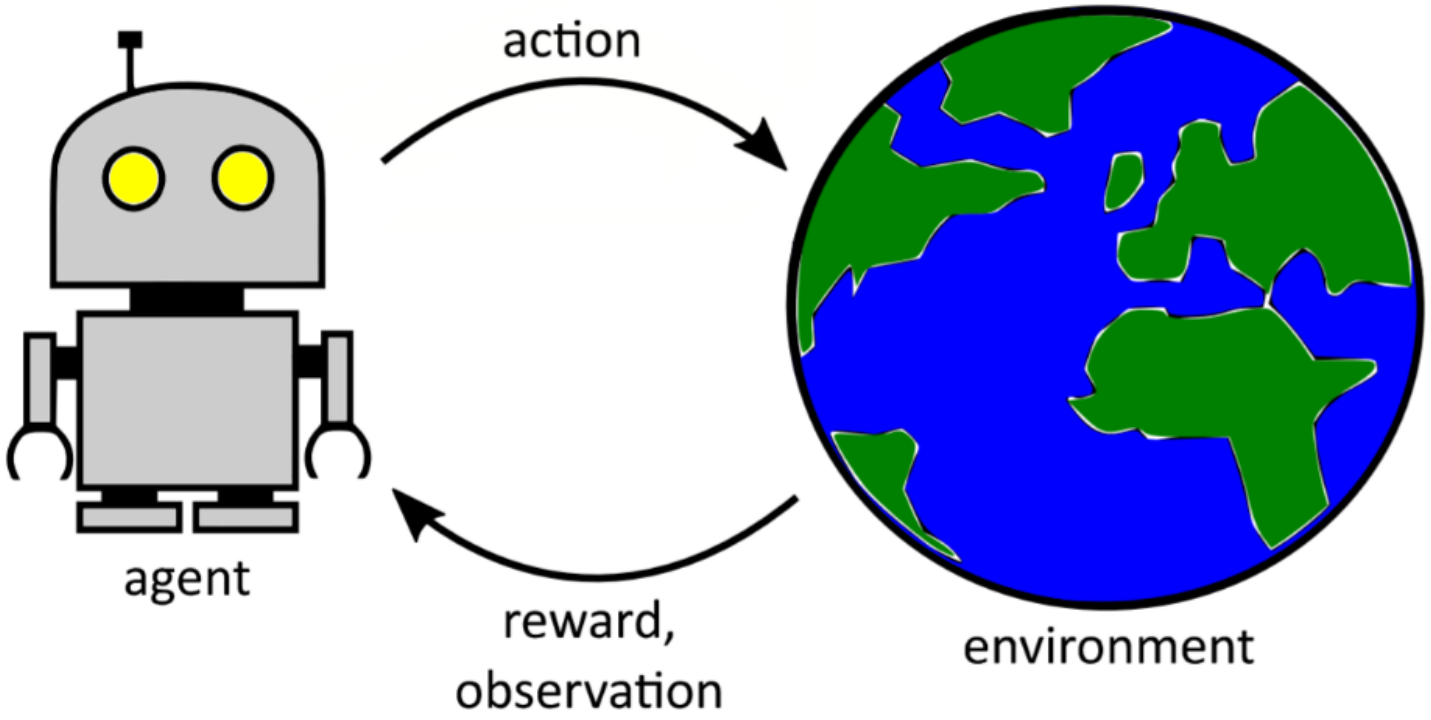

- **Prefer** promising choices (= Exploitation)
  - Example: greedy choice (e.g., using `argmax`)

- A good Multi-Armed Bandit solution should **always** balance between Exploration and Exploitation

# Multi-Armed Bandits



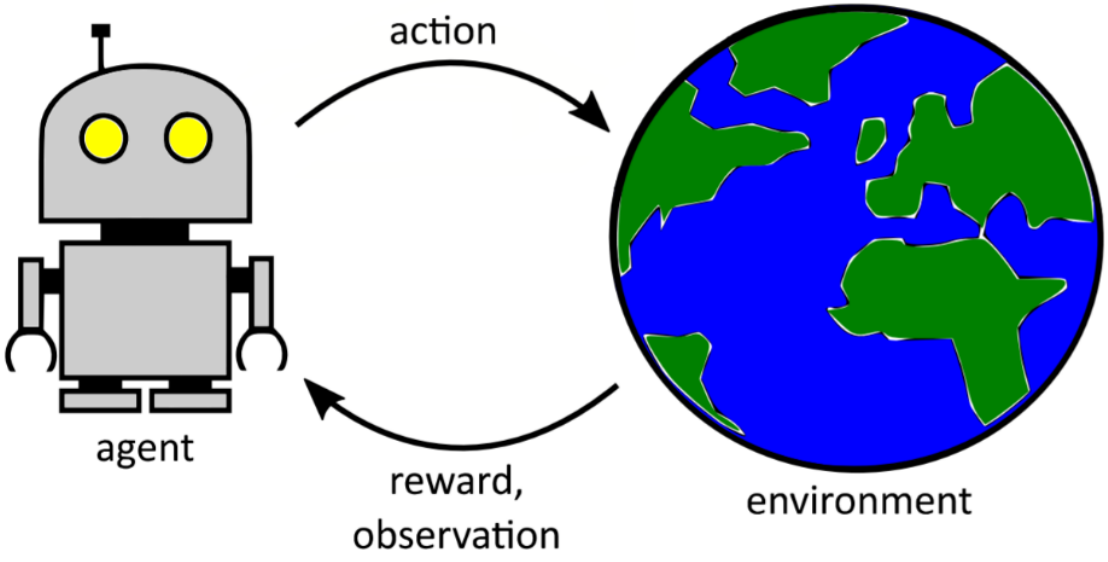
# Multi-Armed Bandits



# → Sequential Decision Making

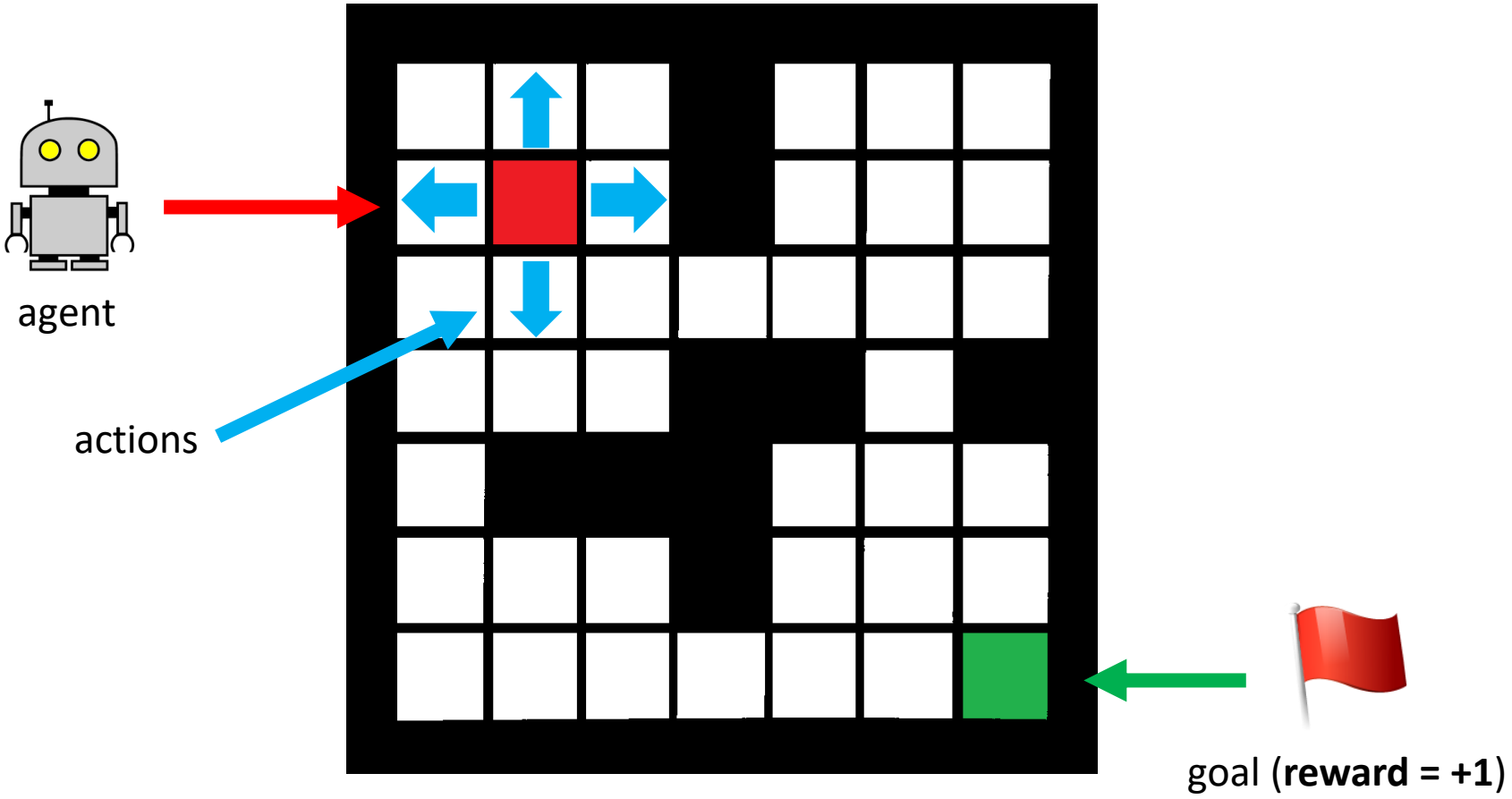
# Sequential Decision Making

- **Goal:** Autonomously select actions to solve a (complex) task
  - time is important (actions might have **long term** consequences)
  - maximize the **expected cumulative reward** for each state



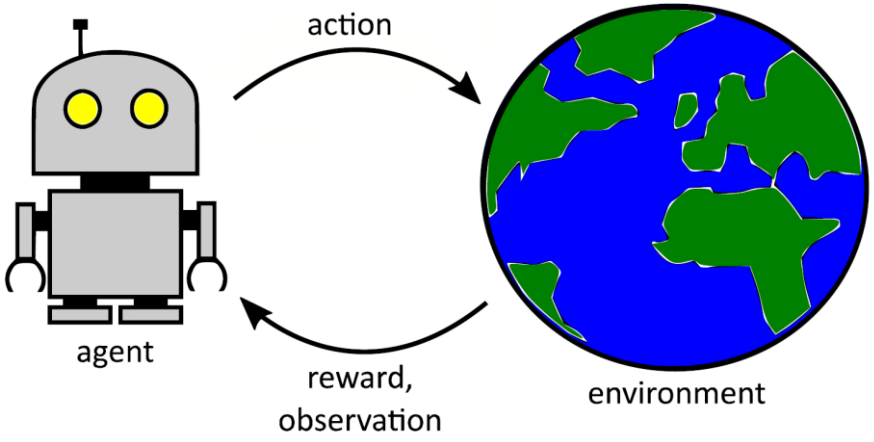
# Sequential Decision Making Example

- **Rooms**: reach a goal as fast as possible



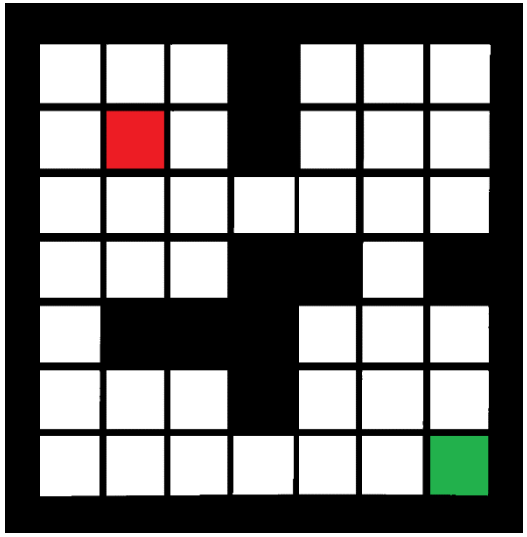
# Markov Decision Processes

- 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



# Rooms as MDP

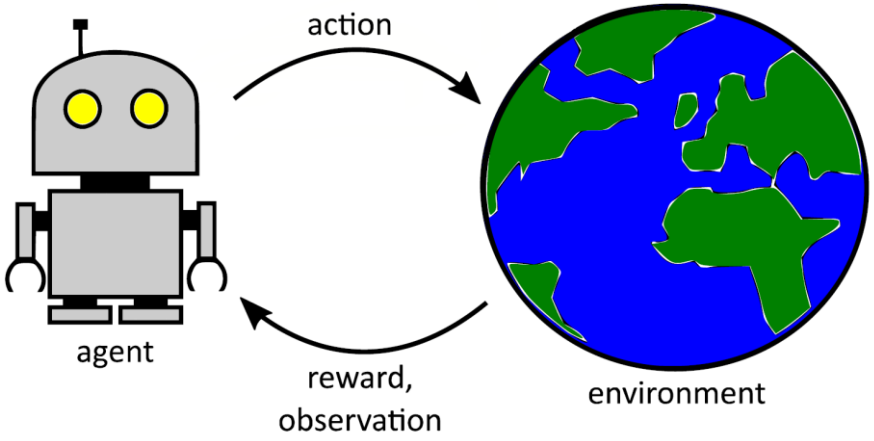
- Define Rooms as MDP $\mathrm{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$:
  - **States $\mathcal{S}$**: position of the agent
  - **Actions $\mathcal{A}$**: move north/south/west/east
  - **Transitions $\mathcal{P}$**: deterministic movement. No transition if moving against wall.
  - **Rewards $\mathcal{R}$**: +1 if goal is reached, 0 otherwise



# Markov Decision Processes

- MDPs formally describe environments for Sequential Decision Making
- All states $s_t \in \mathcal{S}$ are **Markov** such that

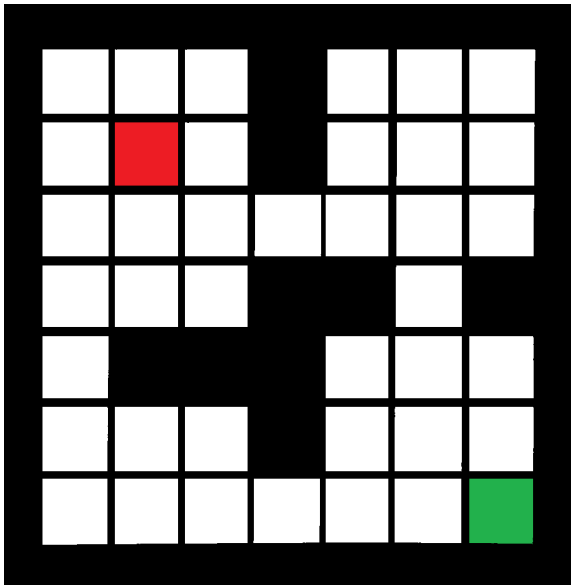
  $\mathbb{P}(s_{t+1}|s_t) = \mathbb{P}(s_{t+1}|s_1, \ldots, s_t)$ (no history of past states required)
- Assumes **full observability** of the state
- States and actions may be **discrete** or **continuous**
- Many problems can be formulated as MDPs!
  - E.g., multi-armed bandits are MDPs with a single state



# Policies

- A **policy** $\pi: \mathcal{S} \rightarrow \mathcal{A}$ represents the behavioral strategy of an agent
  - Policies may also be stochastic $\pi(a_t|s_t) \in [0,1]$

- Policy examples for Rooms:
  - $\pi_0$: maps each state $s_t \in \mathcal{S}$ to a random action $a_t \in \mathcal{A}$
  - $\pi_1$ : maps each state $s_t \in \mathcal{S}$ to action $a_t = MoveSouth \in \mathcal{A}$
  - $\pi_2$ : maps state $s_t \in \mathcal{S}$ to action $a_t = MoveSouth \in \mathcal{A}$ if t is odd and select $a_t$ random otherwise .



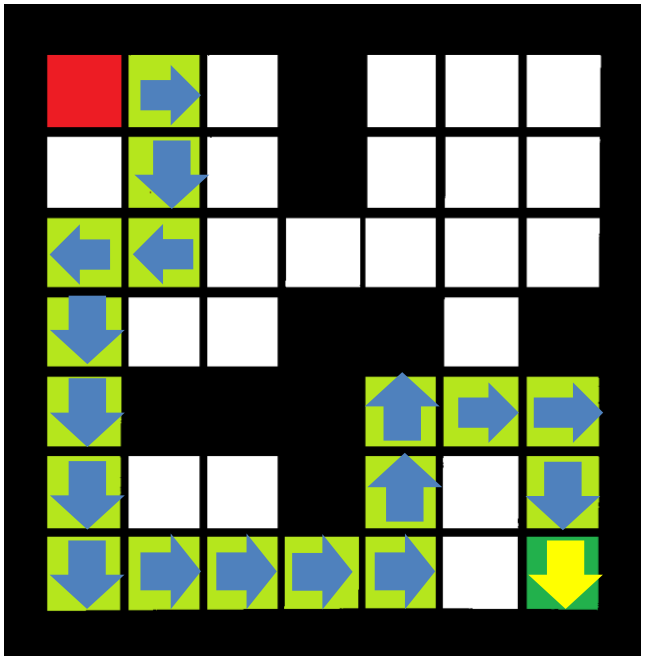
1. **How do we know which policy is better?**
2. **How can we improve a given policy?**

# Returns

- The **return** of a state $s_t \in \mathcal{S}$ for a horizon $h$ given a policy $\pi$ is the cumulative (discounted) future reward ($h$ may be infinite!):

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

- Rooms Example: $\gamma = 0.99$
  - The chosen paths needs **18 steps** to reach the goal
  - Thus, the return from the starting point is: $G_1 = r_1 + \gamma r_2 + \ \ldots + \gamma^{17} r_{18} = \\ = \gamma^{17} r_{18} = 0.99^{17} \sim 0.843$



- What would be the return $G_1$, if the goal isn‘t reached at all?
- What is the optimal value of $G_1$?

# Value Functions

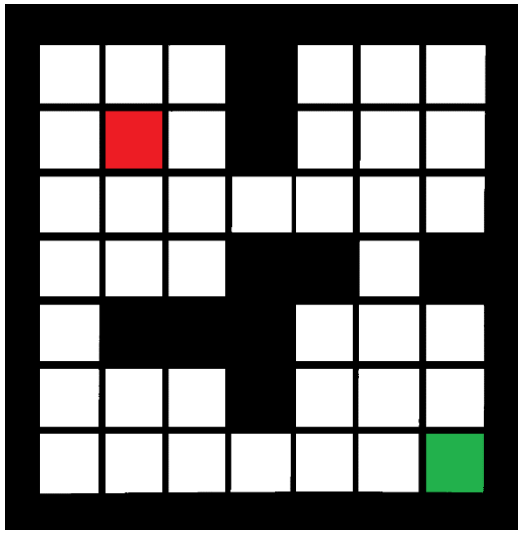
- The **value** of a state $s_t \in \mathcal{S}$ is the expected return of $s_t$ for a horizon $h \in \mathbb{N}$ given a policy $\pi$:

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

- 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 $\pi$:

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

- Rooms Example:
  - $V^{\pi}$ and/or $Q^{\pi}$ can be estimated by **averaging** over **several returns** $G_t$ observed by executing a (fixed) policy $\pi$
- **Value functions ($V^{\pi}$ and/or $Q^{\pi}$) can be used to evaluate policies $\pi$**

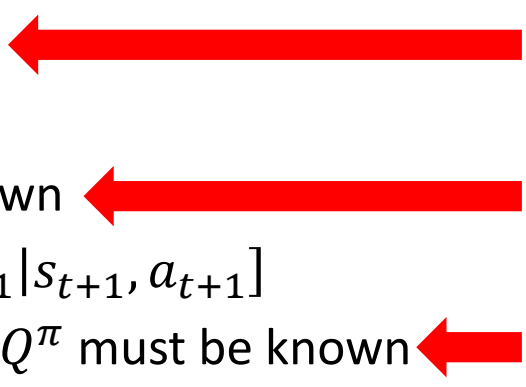


# Remark: Return / Value Estimation

- Estimating the return $G_t$ or the value $Q^\pi(s_t, a_t)$ of state-action pair $\langle s_t, a_t \rangle$ always has the following form:

$$\mathcal{R}(s_t, a_t) + \gamma X$$

- $X$ could be:
  - The successor return $G_{t+1}$
    - reward seqence must be known
  - The successor value $Q^\pi(s_{t+1}, a_{t+1})$
    - $\langle s_{t+1}, a_{t+1} \rangle$ and $Q^\pi$ must be known
  - The expected successor value $\mathbb{E}[G_{t+1} | s_{t+1}, a_{t+1}]$
    - $\mathcal{P}(s_{t+1} | s_t, a_t)$, $\langle s_{t+1}, a_{t+1} \rangle$, and $Q^\pi$ must be known



Monte Carlo Planning / Learning

Temporal-Difference Learning

Dynamic Programming

# Optimal Policies and Value Functions

- **Goal:** Find an *optimal policy* $\pi^*$ which maximizes the expected return $\mathbb{E}[G_t|s_t]$ for each state:

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

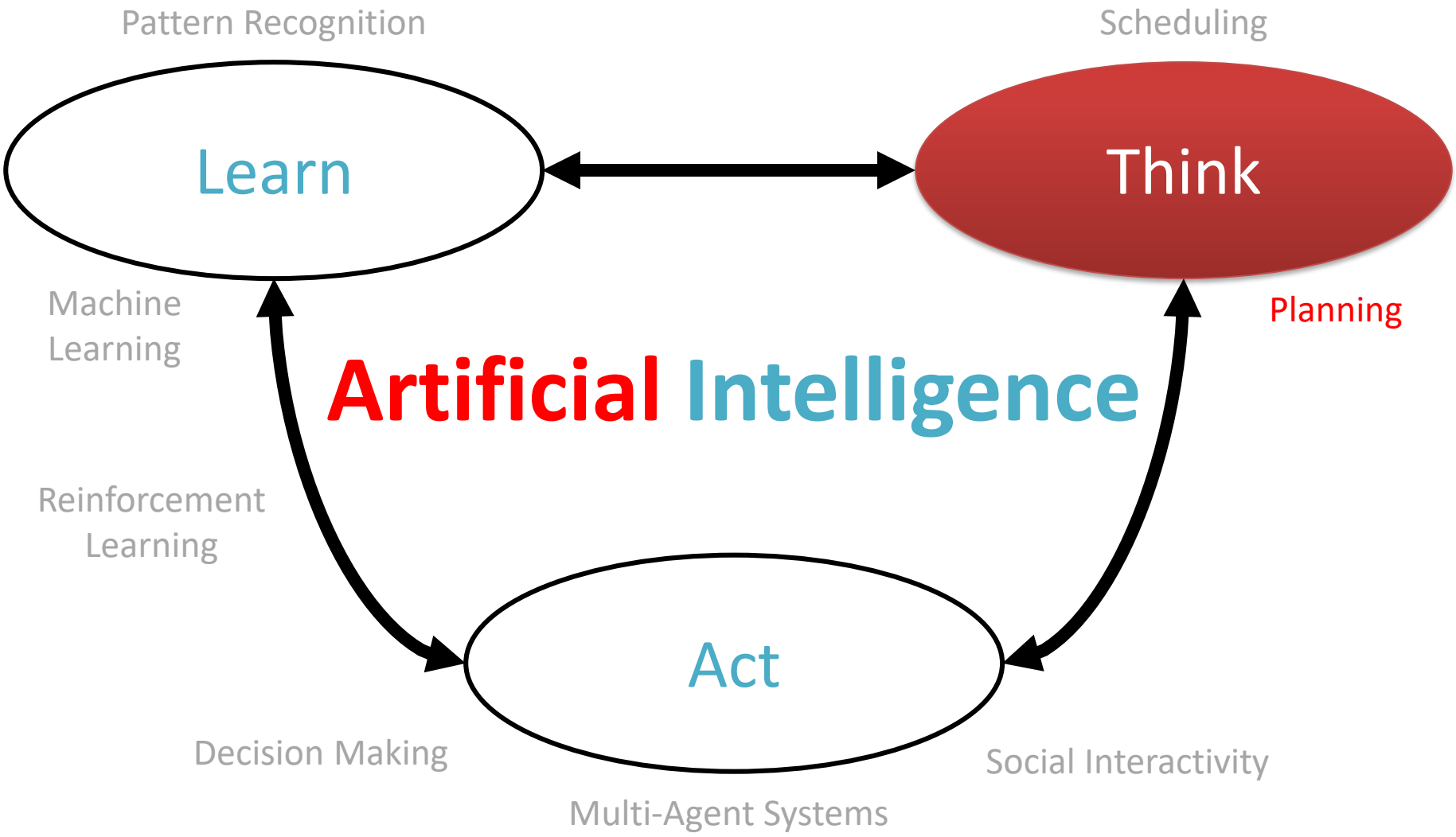
- The *optimal value function* is defined by:

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

- When $V^*$ or $Q^*$ is known, $\pi^*$can be derived.

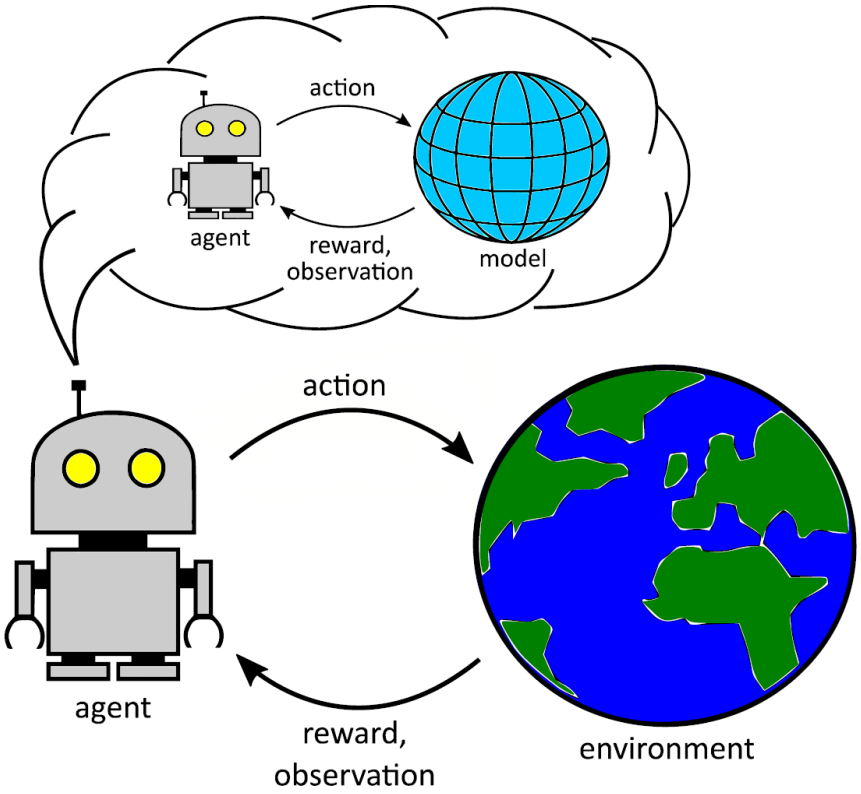
How to find an **optimal** policy or the **optimal** value function?

# → Automated Planning

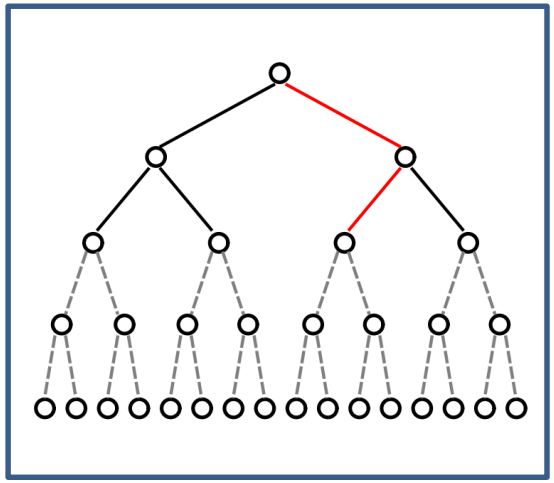

Pattern Recognition
Scheduling
Learn
Think
Machine Learning
Planning
Artificial Intelligence
Reinforcement Learning
Act
Decision Making
Social Interactivity
Multi-Agent Systems

# Automated Planning

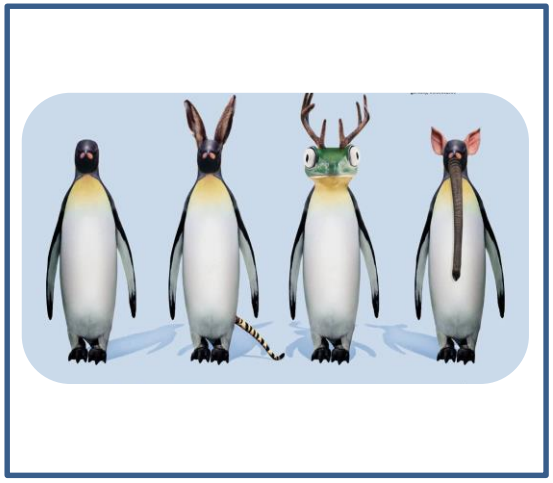
- **Goal:** Find (near-)**optimal policies** $\pi^*$ to solve complex problems
- Use (heuristic) lookahead search on a **given model** $\widehat{M} \approx M$ of the problem



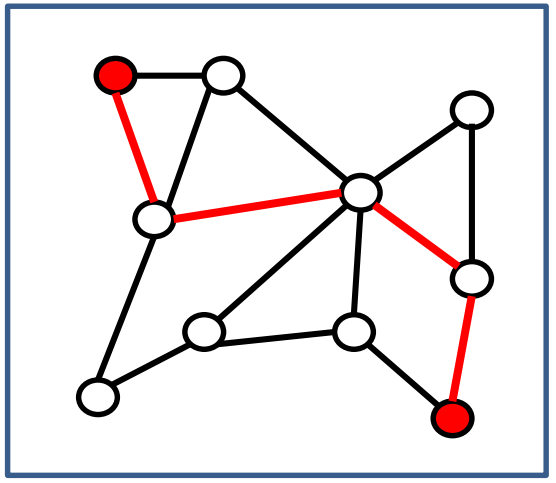
# Planning Approaches (Examples)



Tree Search



Evolutionary Computation



Dynamic Programming

# → Dynamic Programming

# Dynamic Programming

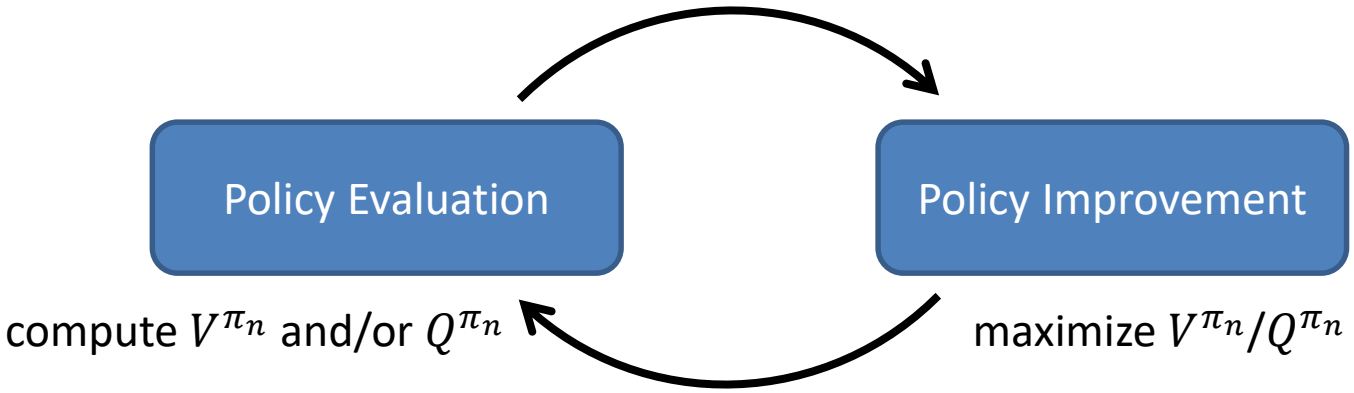
- **Dynamic** refers to sequential / temporal component of a problem
- **Programming** refers to optimization of a program

- We want to solve Markov Decision Processes (MDPs):
  - MDPs are **sequential** decision making problems
  - To find a solution, we need to optimize a **program** (policy $\pi$)

# Policy Iteration

- Dynamic Programming approach to find an optimal policy $\pi^*$

- Starts with a (random) guess $\pi_0$
- Consists of two alternating steps given $\pi_n$:




- Terminates when $\pi_{i+1} = \pi_i$ or when a time budget runs out

- **Policy Iteration forms the basis for most Planning and Reinforcement Learning algorithms!**

# Value Iteration

- Dynamic Programming approach to find the optimal value function $V^*$

- Starts with a (random) guess $V^0$
- Iteratively updates the value estimate $V^n(s_t)$ for **each state** $s_t \in \mathcal{S}$

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

Policy Improvement — Policy Evaluation

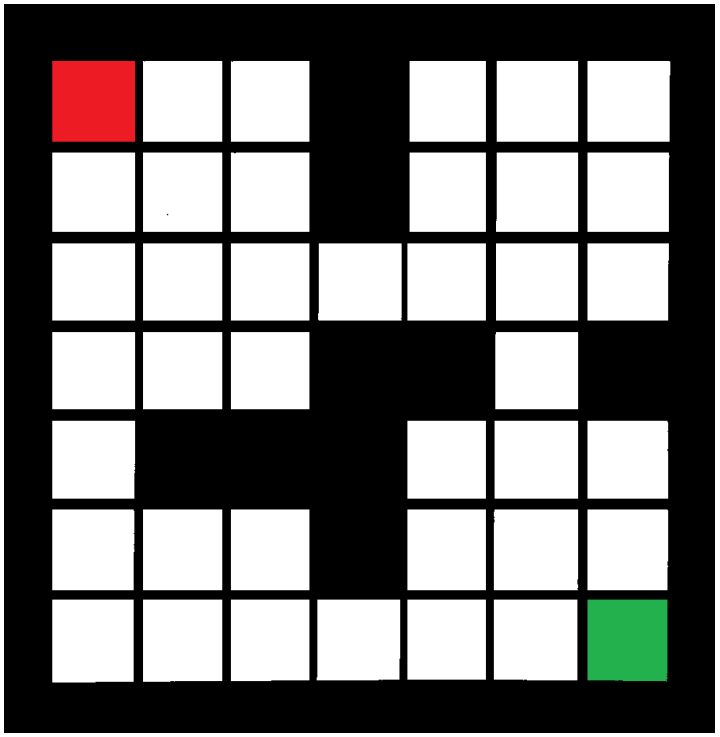
- Terminates when $V^{n+1} = V^n$ or when a time budget runs out
- The optimal action-value function $Q^*$ is computed analogously

- $V^*$ and/or $Q^*$ can be used to derive an optimal policy $\pi^*$

- **Do you see the link to Policy Iteration?**

# Value Iteration - Example

- Optimal „Value Map“ in Rooms ($\gamma = 0.99$): for each state $s_t \in \mathcal{S}$
  - $V^{n+1}(s_t) = \max_{a_t \in \mathcal{A}} \{\mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1}|s_t, a_t) V^n(s_{t+1})\}$



**Remember:**

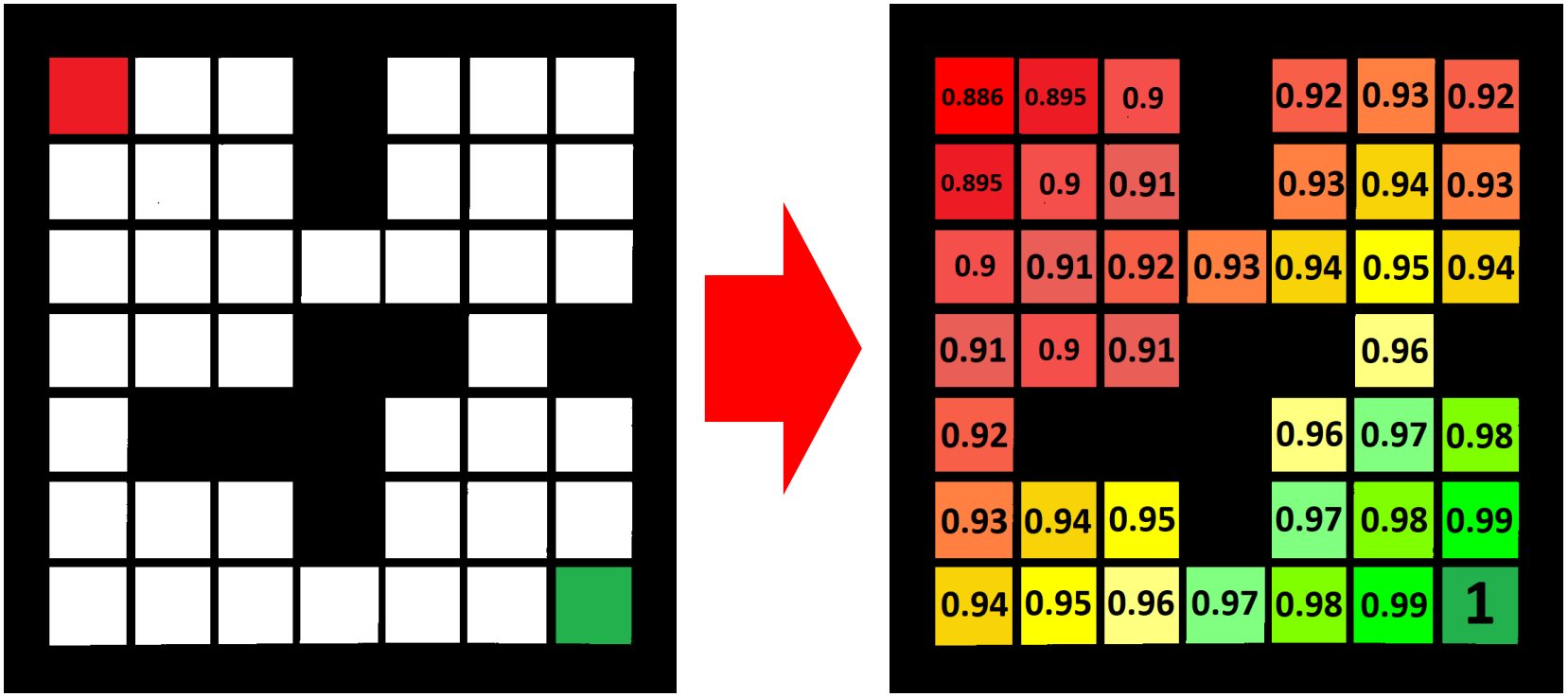
$$\mathcal{R}(s_t, a_t) + \gamma X$$

In this case

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

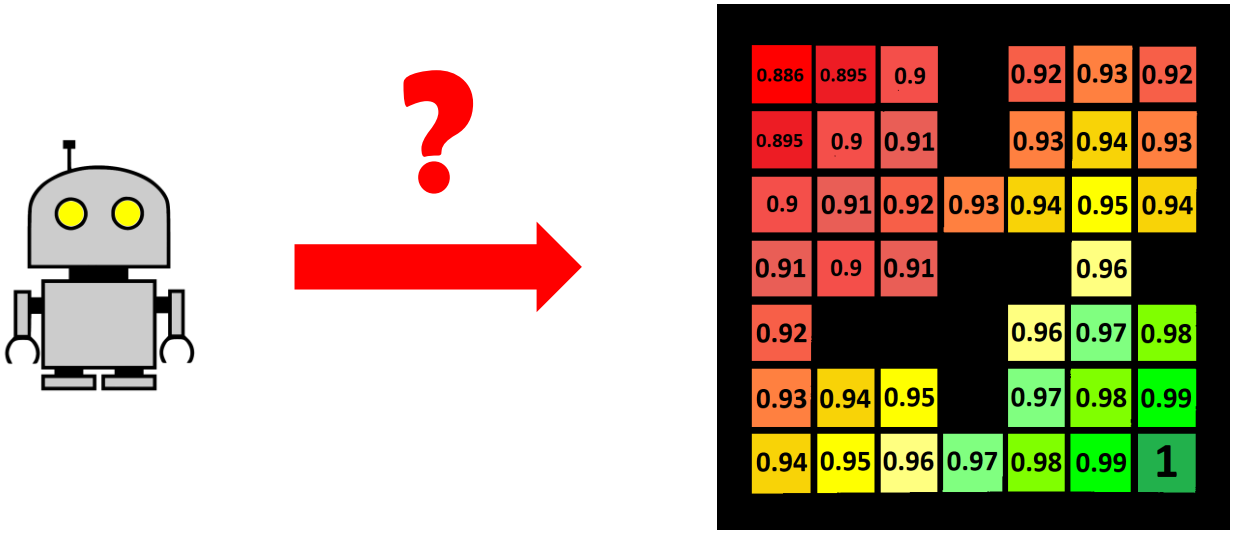
# Value Iteration - Example

- Optimal „Value Map“ in Rooms ($\gamma = 0.99$): for each state $s_t \in \mathcal{S}$
  - $V^{n+1}(s_t) = \max_{a_t \in \mathcal{A}} \{\mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1}|s_t, a_t) V^n(s_{t+1})\}$



# Advantages and Disadvantages of DP

- General approach (does not require **explicit** domain knowledge)
- Converges to optimal solution
- Does not require exploration-exploitation (**all states** are visited anyway)
- Computational costs
- Memory costs
- Availability of an explicit model $\mathrm{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$



# Intermediate Summary

- What we know so far:
  - Markov Decision Processes (MDPs)
  - Policies and Value Functions
  - Optimally solve MDPs with Dynamic Programming

- What we don‘t know (yet):
  - How to find solutions in a more scalable way?
  - How to react to unexpected events?
  - How to find solutions without a model?

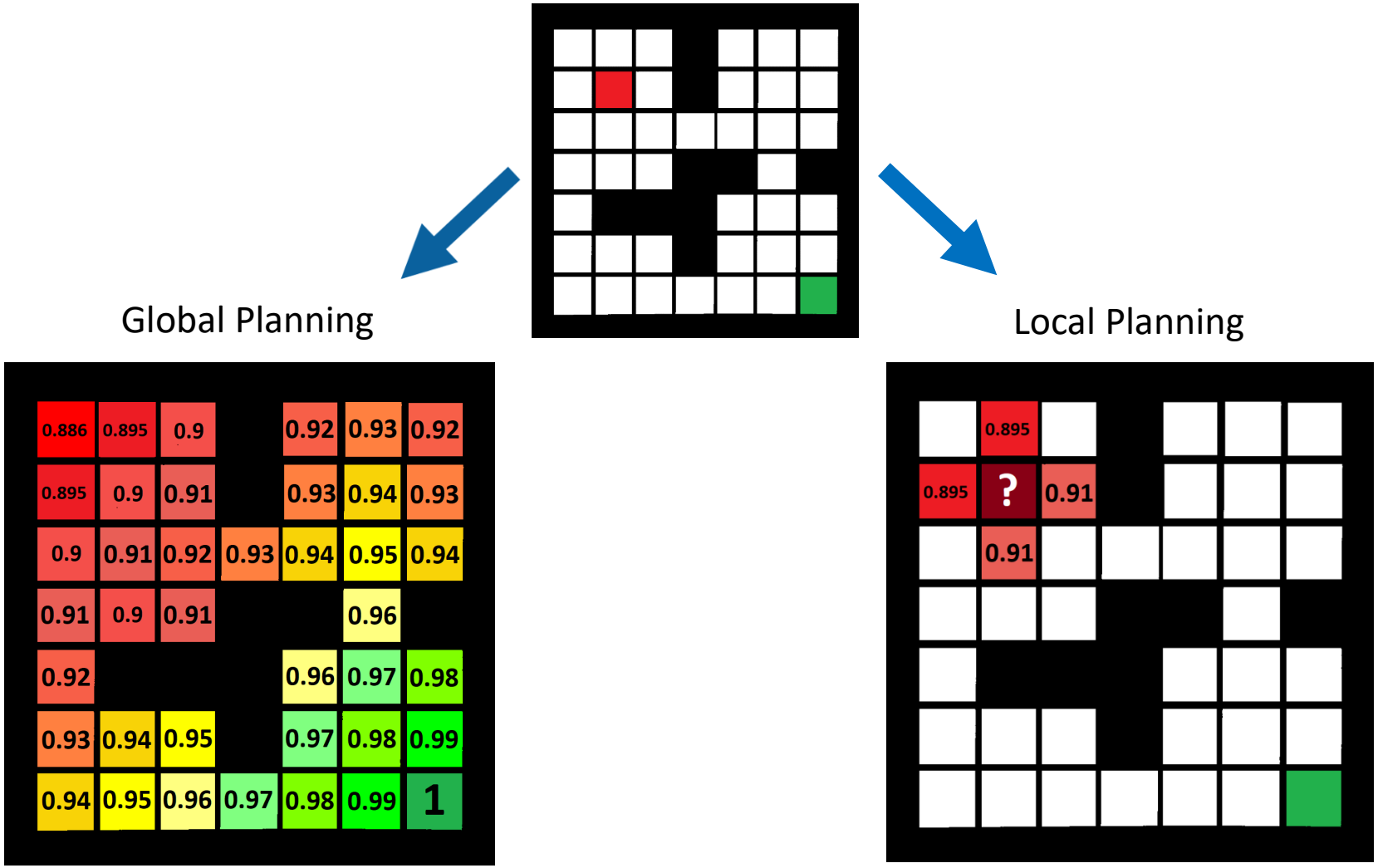
# → Monte Carlo Planning

# Global Planning and Local Planning

- Global Planning
  - considers the **entire state space** $\mathcal{S}$ to approximate $\pi^*$
  - 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)
  - Examples: Dynamic Programming (Policy and Value Iteration)

- Local Planning
  - only considers the **current state** $s_t \in \mathcal{S}$ (and possible future states) to approximate $\pi^*(s_t)$
  - recommends an action $a_t \in \mathcal{A}$ for current state $s_t \in \mathcal{S}$
  - can be performed **online** (interleaving planning and execution)
  - Examples: Monte Carlo Tree Search

# Global Planning vs. Local Planning



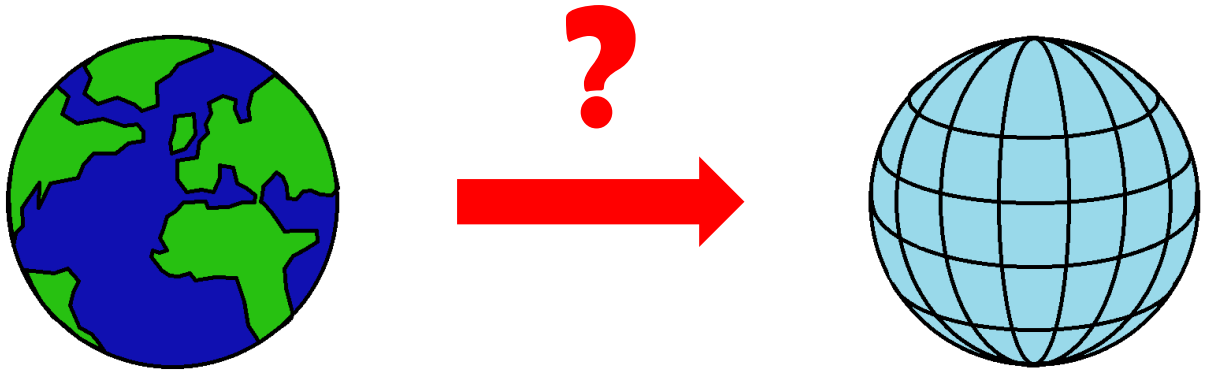
# Monte Carlo Planning

- Dynamic Programming always assumes **full knowledge** of the underlying MDP $\mathrm{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$
  - Most real-world applications have extremely large state spaces
  - Especially $\mathcal{P}(s_{t+1}|s_t, a_t)$ is hard to pre-specify in practice!

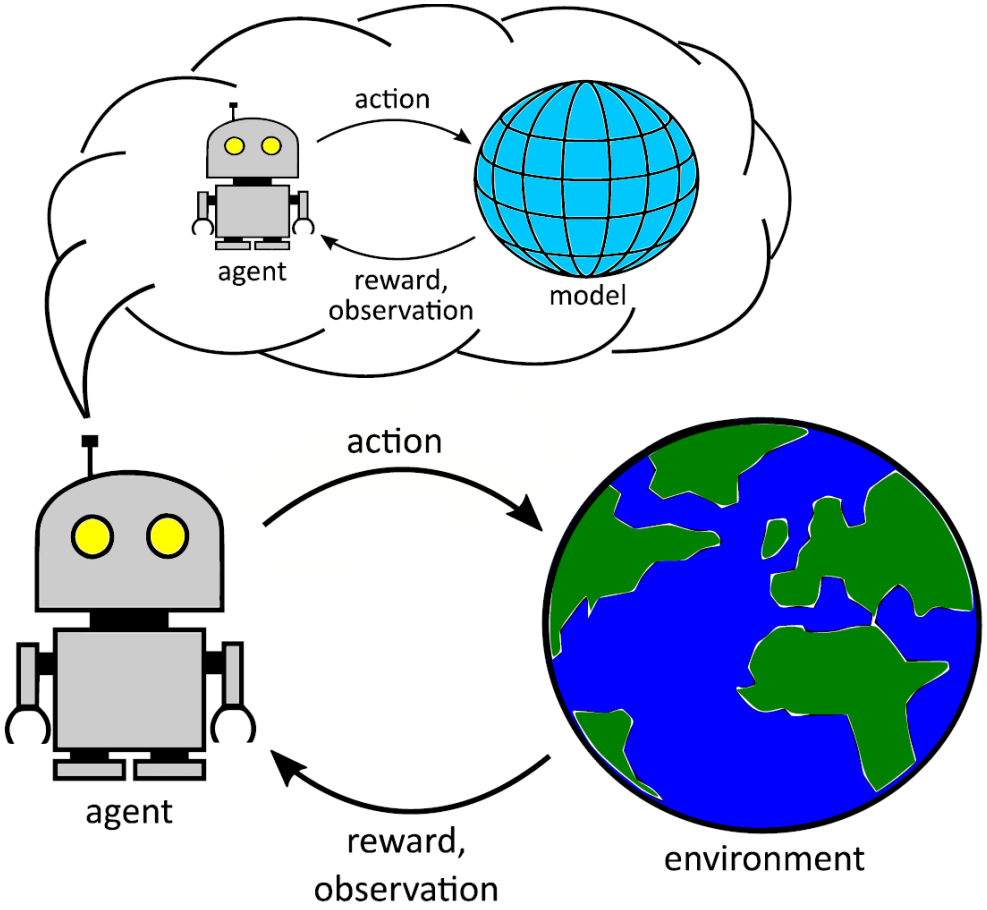
- Monte Carlo Planning only requires a generative model as **blackbox simulator** $\widehat{M} \approx \mathrm{M}$
  - Given some state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$, the generative model provides a sample $s_{t+1} \in \mathcal{S}$ and $r_t = \mathcal{R}(s_t, a_t)$
  - Can be used to approximate $V^*$ or $Q^*$ via statistical sampling
  - Requires minimal domain knowledge ($\widehat{M}$ can be easily replaced)

# Explicit Model vs. Generative Model

- Generative model can be easier implemented than explicit probability distributions!



# Planning with Generative Model



# Monte Carlo Rollouts (MCR)

- **Goal:** Given a state $s_t \in \mathcal{S}$ and a policy $\pi_{rollout}$ , we want to find the action $a_t \in \mathcal{S}$ which maximizes $Q^{\pi_{rollout}}(s_t, a_t) = \mathbb{E}[G_t | s_t, a_t]$

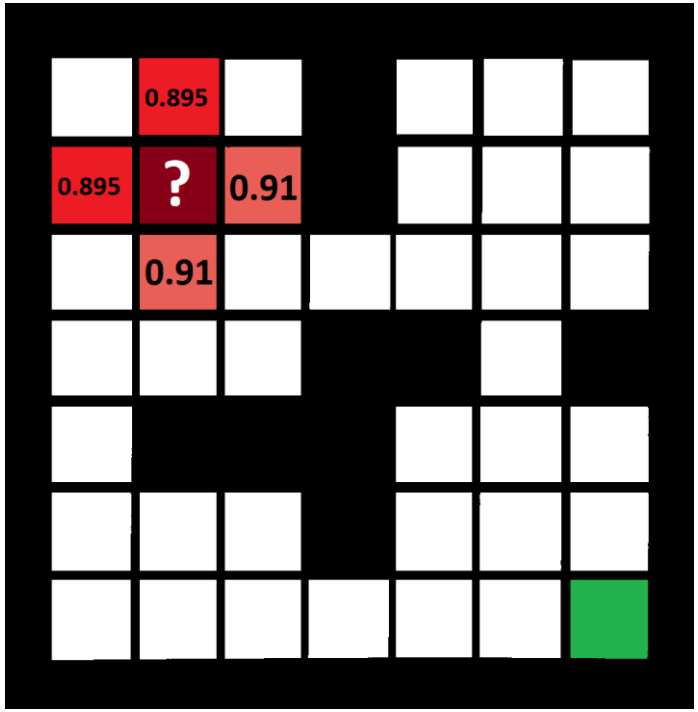
- **Approach**: Given a computation budget of $K$ simulations and a horizon $h$
  - Sample $K$ action sequences (= plans) of length $h$ from $\pi_{rollout}$
  - Simulate all plans with generative model $\widehat{\mathrm{M}}$ and compute the return $G_t$ for each plan
  - Update estimate of $Q^{\pi_{rollout}}(s_t, a_t) = \mathbb{E}[G_t | s_t, a_t]$*
  - **Finally:** Select action $a_t \in \mathcal{S}$ with highest $Q^{\pi_{rollout}}(s_t, a_t)$

*only estimate $Q^{\pi_{rollout}}(s_t, a_t)$ of the first action $a_t$ in each plan!

# Why do Monte Carlo Rollouts work?

- MCR estimates value function $Q^{\pi_{rollout}}(s_t, a_t)$ of $\pi_{rollout}$ via **sampling**
- Final decision is a **maximization** of $Q^{\pi_{rollout}}(s_t, a_t)$

- MCR makes always decisions with **the same or better** quality than $\pi_{rollout}$
- Thus, decision quality depends on $\pi_{rollout}$ and the simulation model



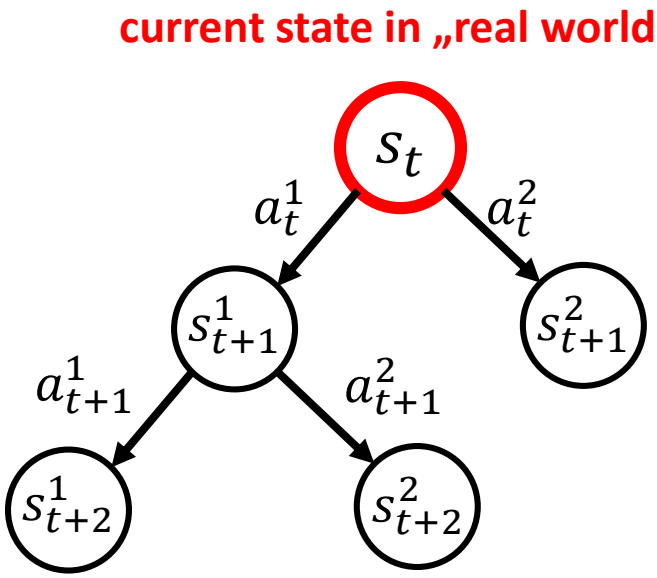
# Monte Carlo Tree Search (MCTS)

- Current state-of-the-art algorithm for Monte Carlo Planning. Used for:
  - board games like Go, Chess, Shogi
  - combinatorial optimization problems like Rubix Cube

- **Approach**: Incrementally construct and traverse a search tree given a computation budget of $K$ simulations and a horizon $h$
  - nodes represent states $s_t \in \mathcal{S}$ (and actions $a_t \in \mathcal{A}$)
  - search tree is used to „learn“ $\hat{Q} \approx Q^*$ via blackbox simulation

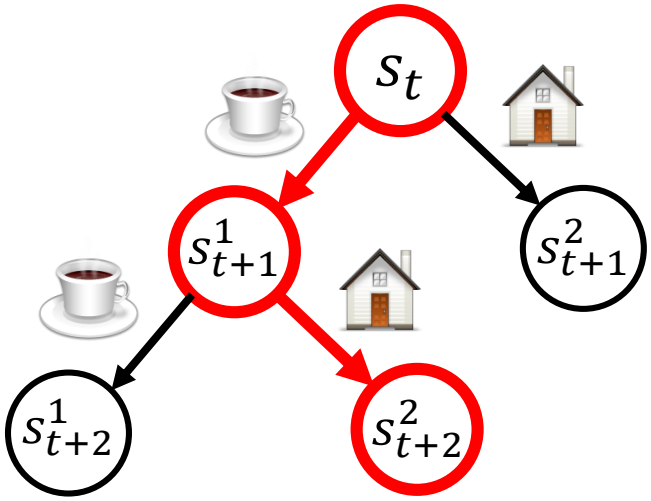
# Monte Carlo Tree Search Phases

- Selection
- Expansion
- Evaluation/Simulation
- Backup



# Monte Carlo Tree Search - Selection

- **Selection**
- Expansion
- Evaluation/Simulation
- Backup




**Example Selection Strategies:**

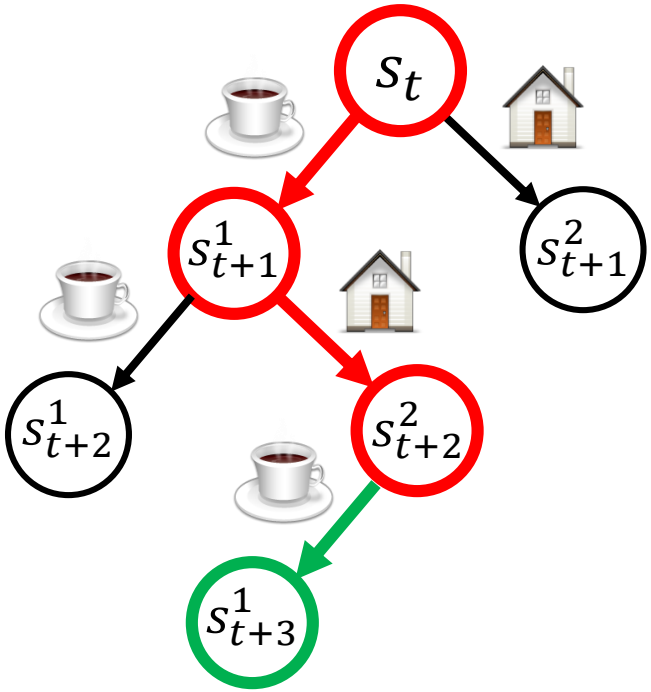
- Random
- Greedy
- $\epsilon$-Greedy
- UCB1

} **Multi-Armed Bandits**

➡ Exploration-Exploitation!!!

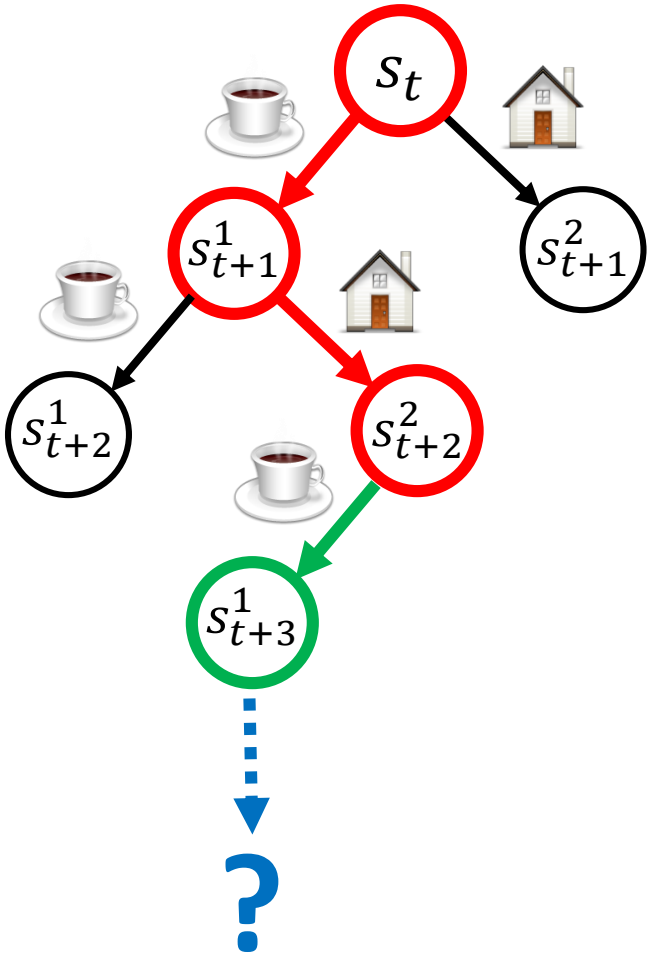
# Monte Carlo Tree Search - Expansion

- Selection
- **Expansion**
- Evaluation/Simulation
- Backup



# Monte Carlo Tree Search - Expansion

- Selection
- Expansion
- **Evaluation/Simulation**
- Backup

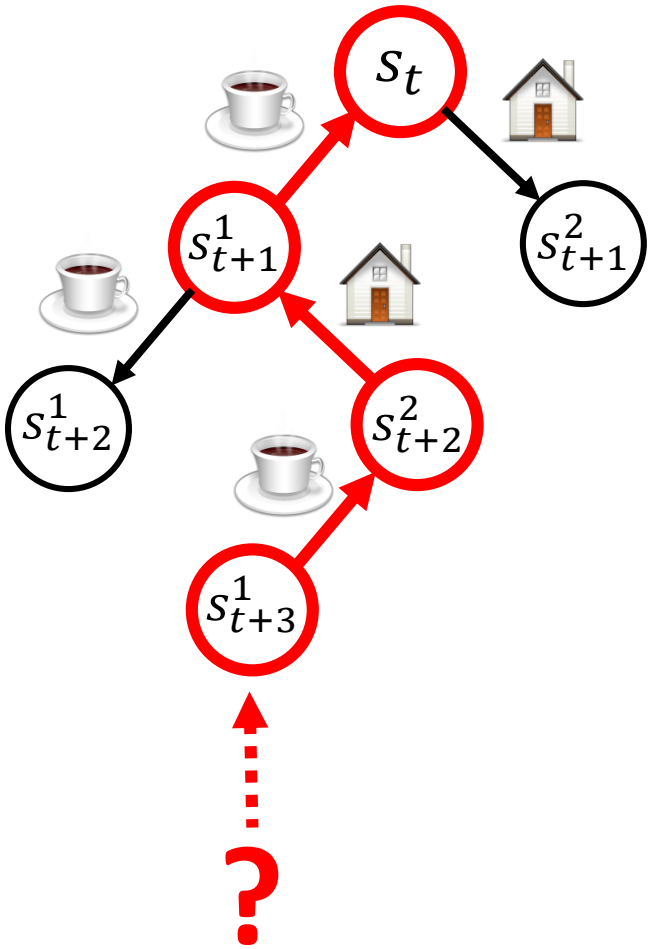



**Example Evaluation Strategies:**

- Rollouts (e.g., Random)
- Value Function $V^{\pi}(s_t)$ (e.g., Reinforcement Learning)

# Monte Carlo Tree Search - Backup

- Selection
- Expansion
- Evaluation/Simulation
- **Backup**




**Remember:**

$$\mathcal{R}(s_t, a_t) + \gamma X$$

In this case $X = G_{t+1}$
(return from next state $s_{t+1}$)

# Summary

- What we know so far:
  - Markov Decision Processes (MDPs)
  - Policies and Value Functions
  - Optimally solve MDPs with Dynamic Programming
  - Approximately solve MDPs with Monte Carlo Search

- What we don‘t know (yet):
  - How to find solutions without a model?

# Thank you!