# Sequential Decision Making for Cooperative Agents

## Part I: An Introduction to Decision Theory

Stefan J. Witwicki
Robotic Systems Laboratory (LSRO)
École Polytechnique Fédérale de Lausanne

João V. Messias
Institute for Systems and Robotics (ISR)
IST, Universidade de Lisboa

EASSS-2014

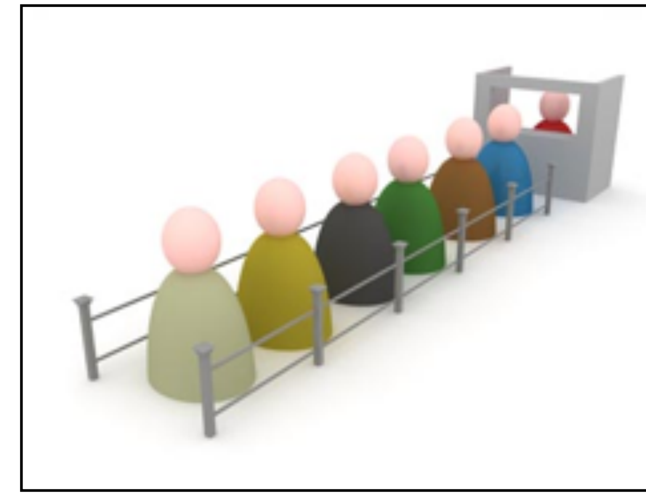
July 15, 2014

# Motivation

## Sequential Decision Making

Maintenance and Scheduling

Autonomous Robots

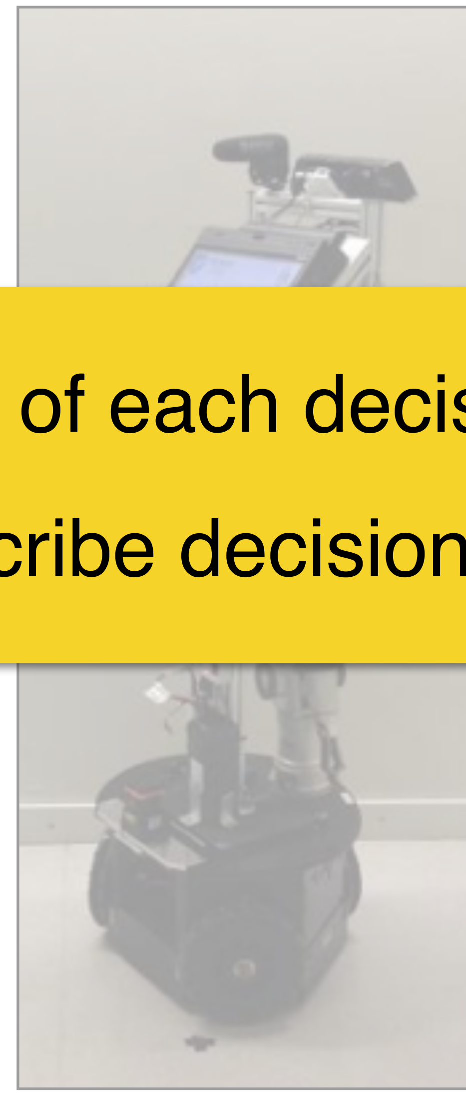


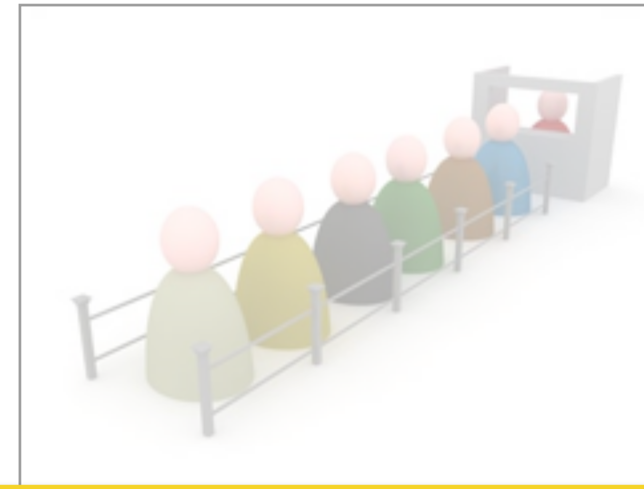
Queue Management

Renewable Energy

Medical Decision Making

What do these domains have in common?





The outcome of each decision is uncertain

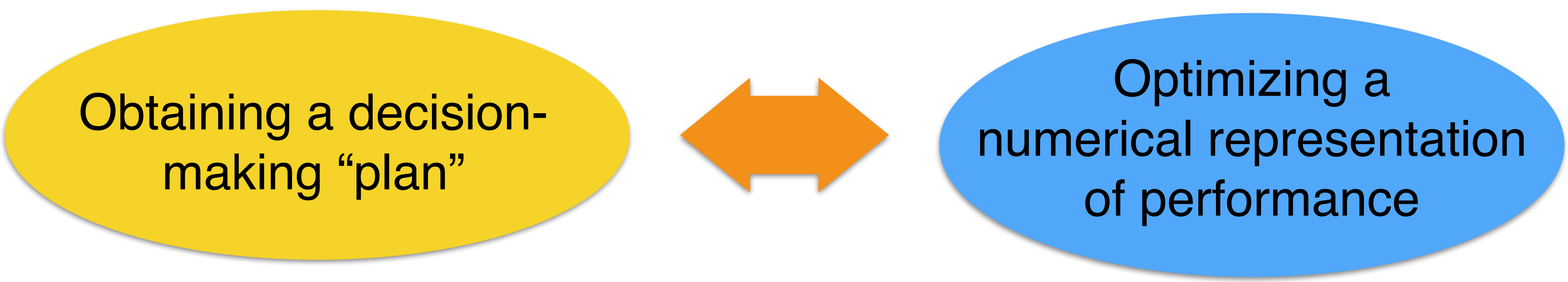
It is hard to manually prescribe decisions for every possible outcome

Renewable Energy

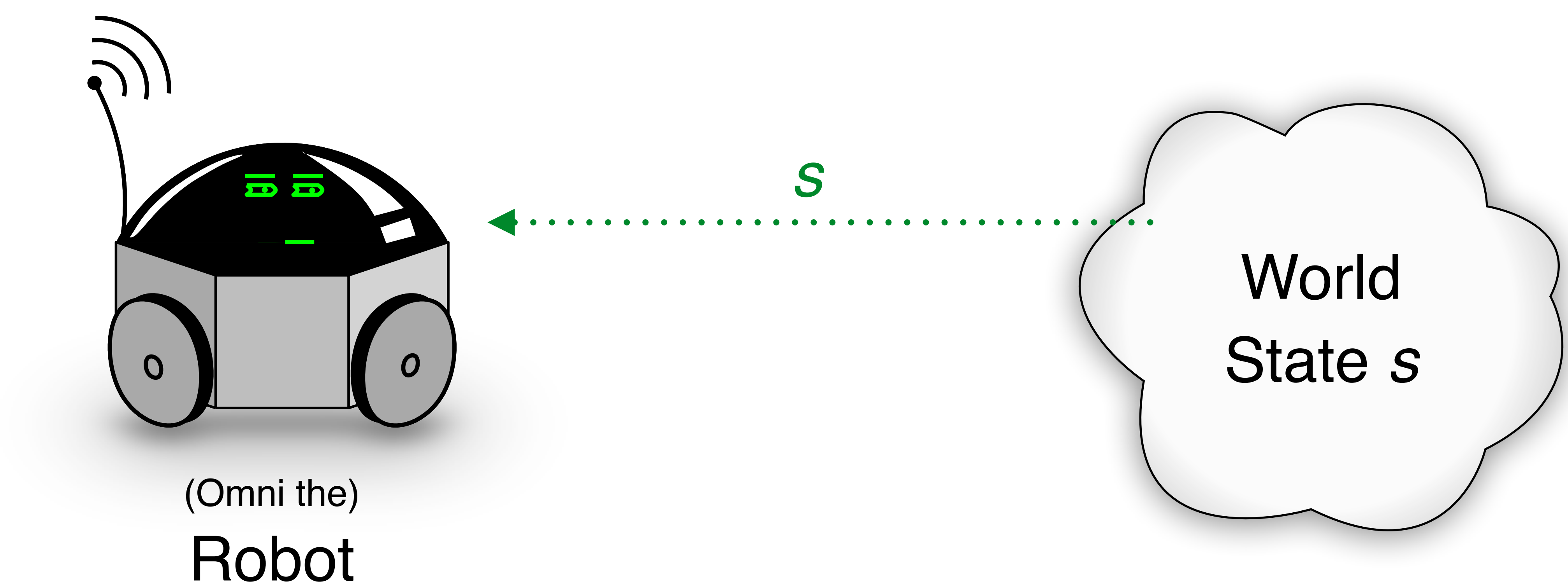
Autonomous Robots

Medical Decision Making

Decision Theory provides mathematical tools to model and solve sequential decision-making problems **in the presence of uncertainty**

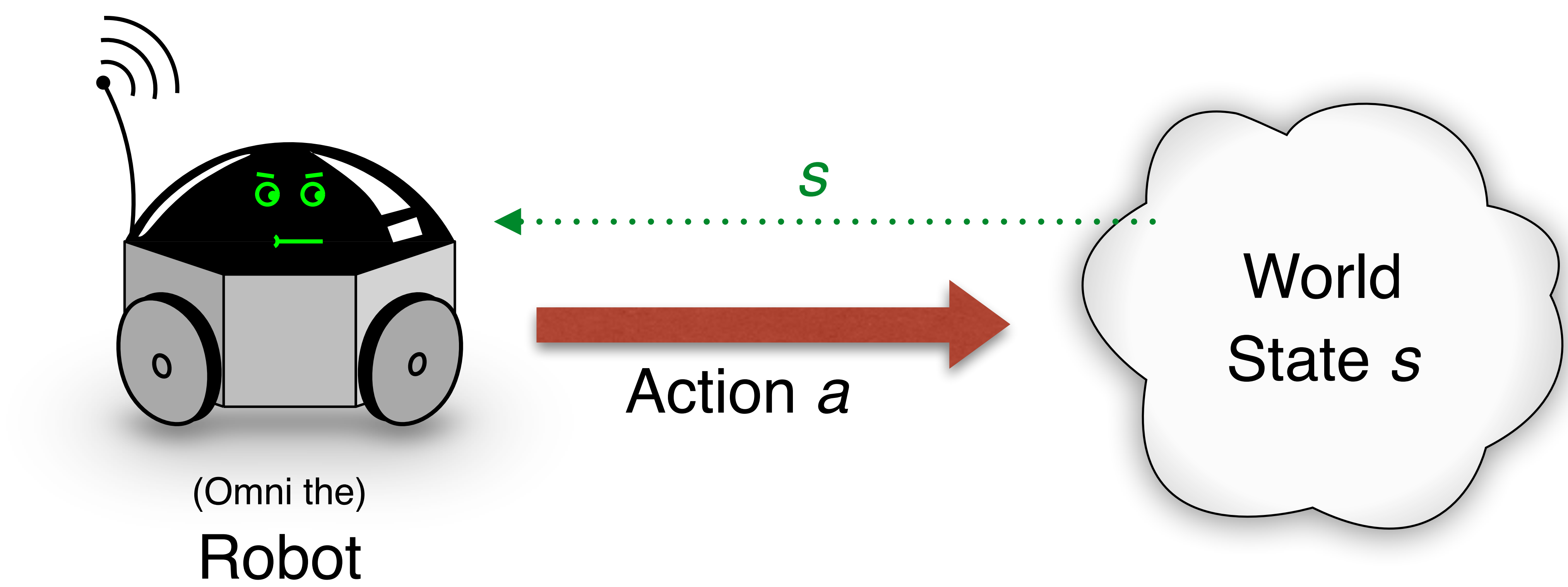


A Markov Decision Process:




**1.** The agent observes state $s$

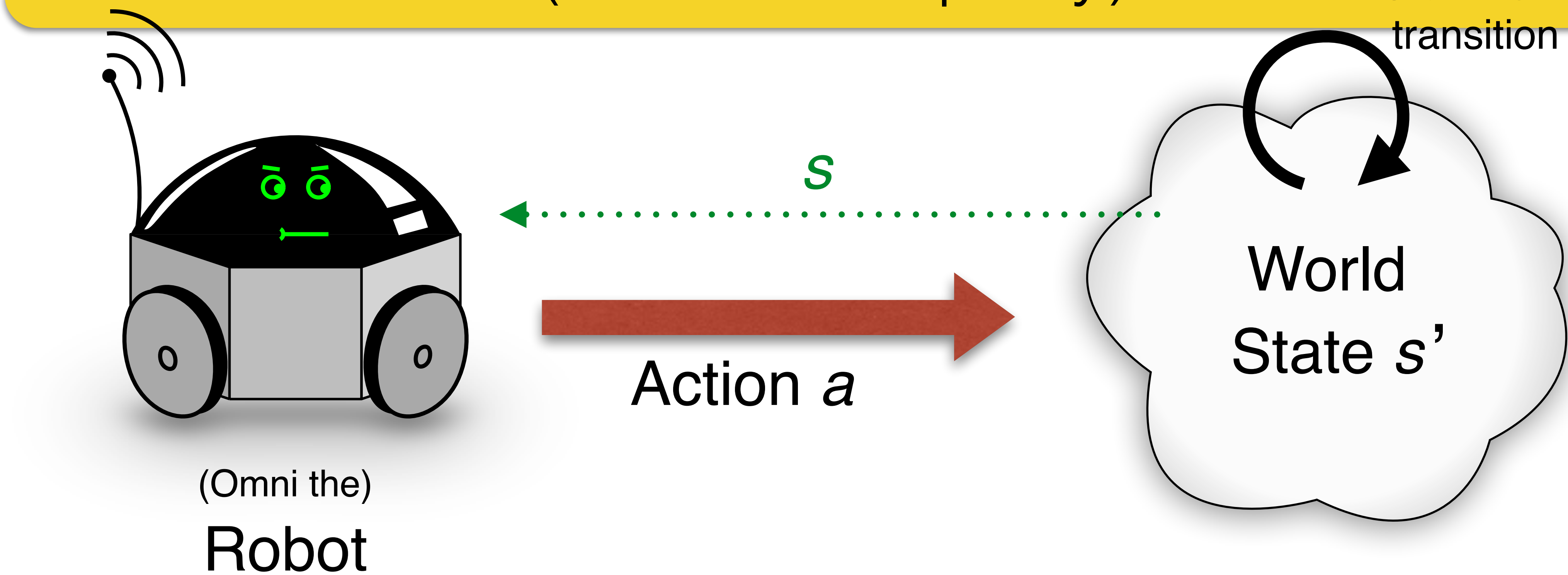
A Markov Decision Process:




**2.** The agent selects action *a*

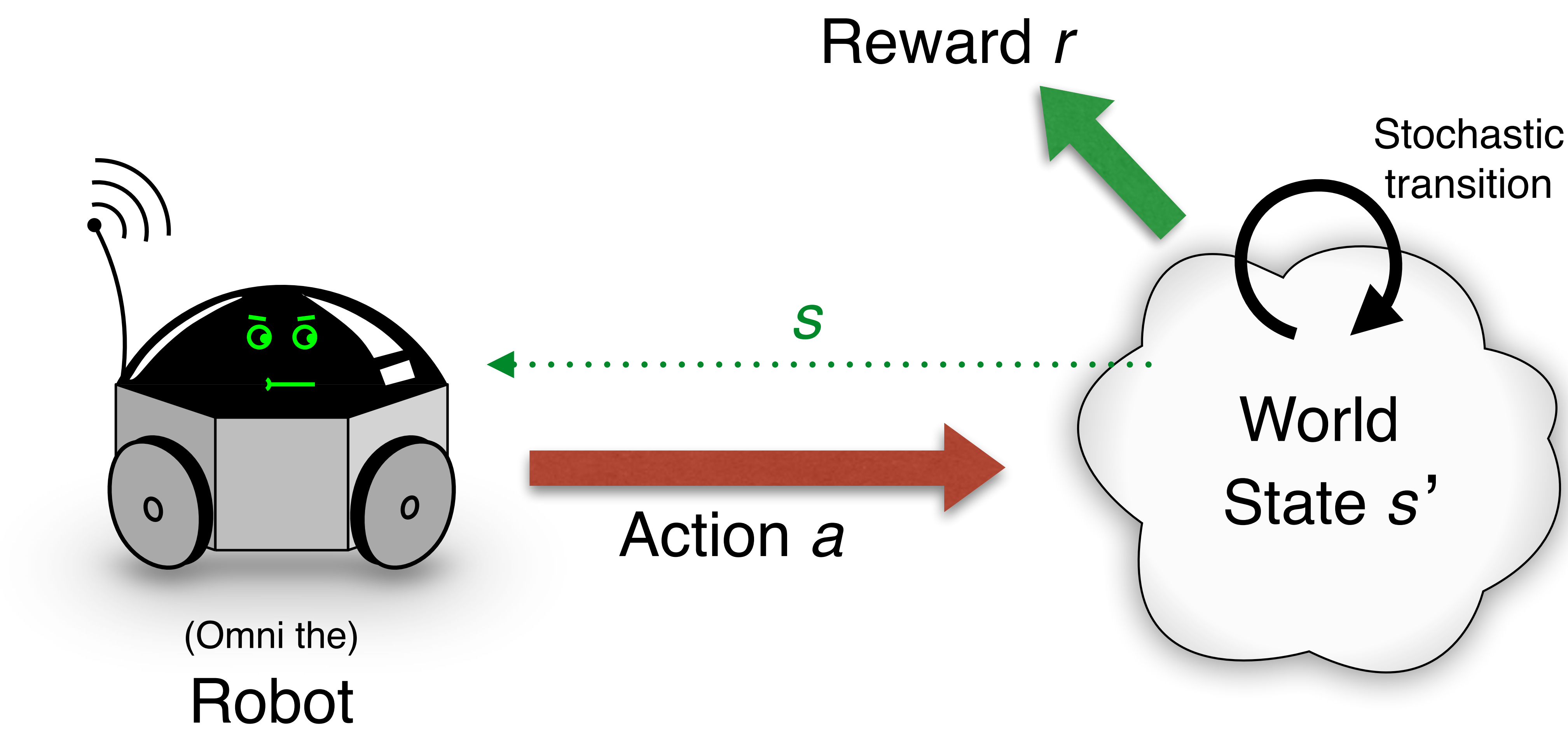
A Markov Decision Process:

**Only depends on the last state and action!**
(Markov Property)




**3.** The state of the system changes randomly to *s'*

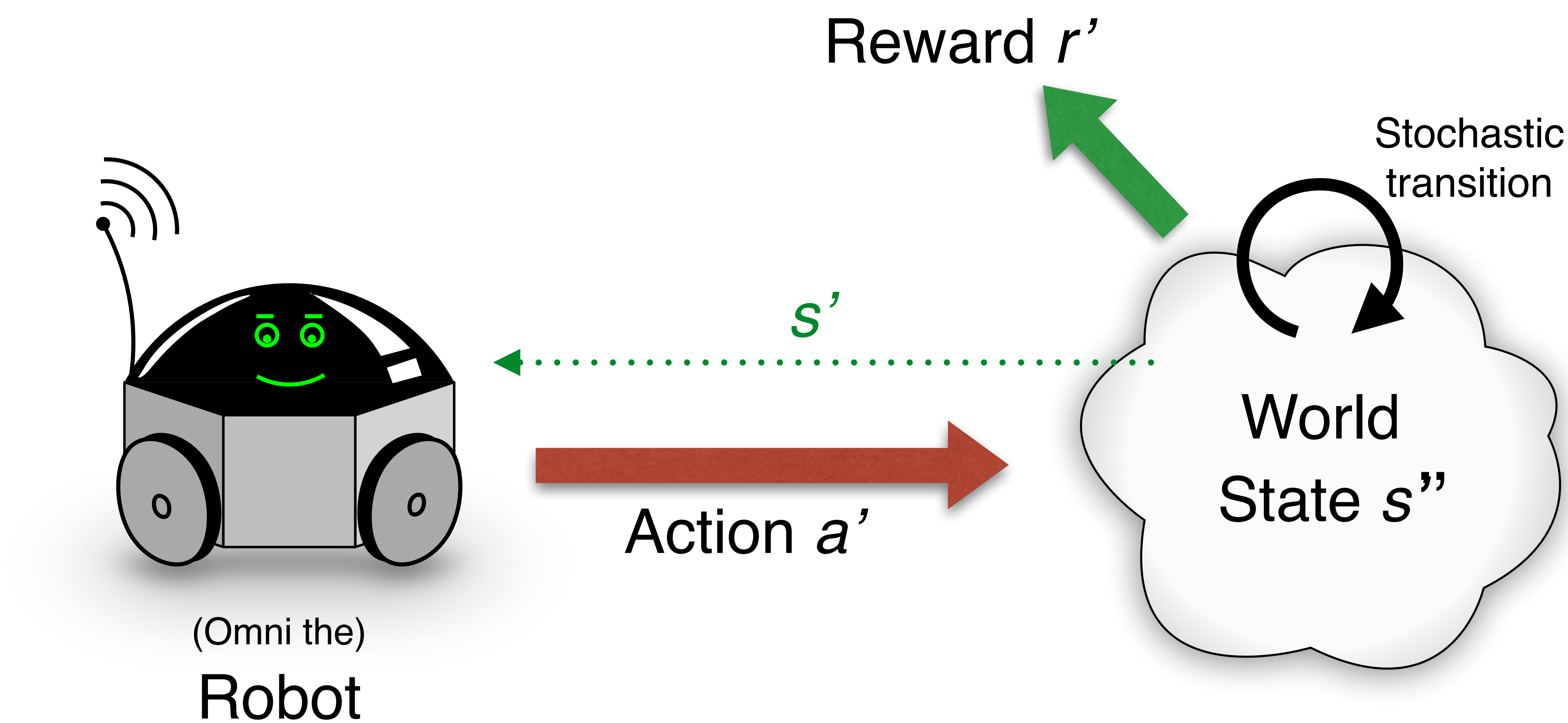
A Markov Decision Process:




**4.** A *reward* is assigned for the execution of action *a* in state *s*

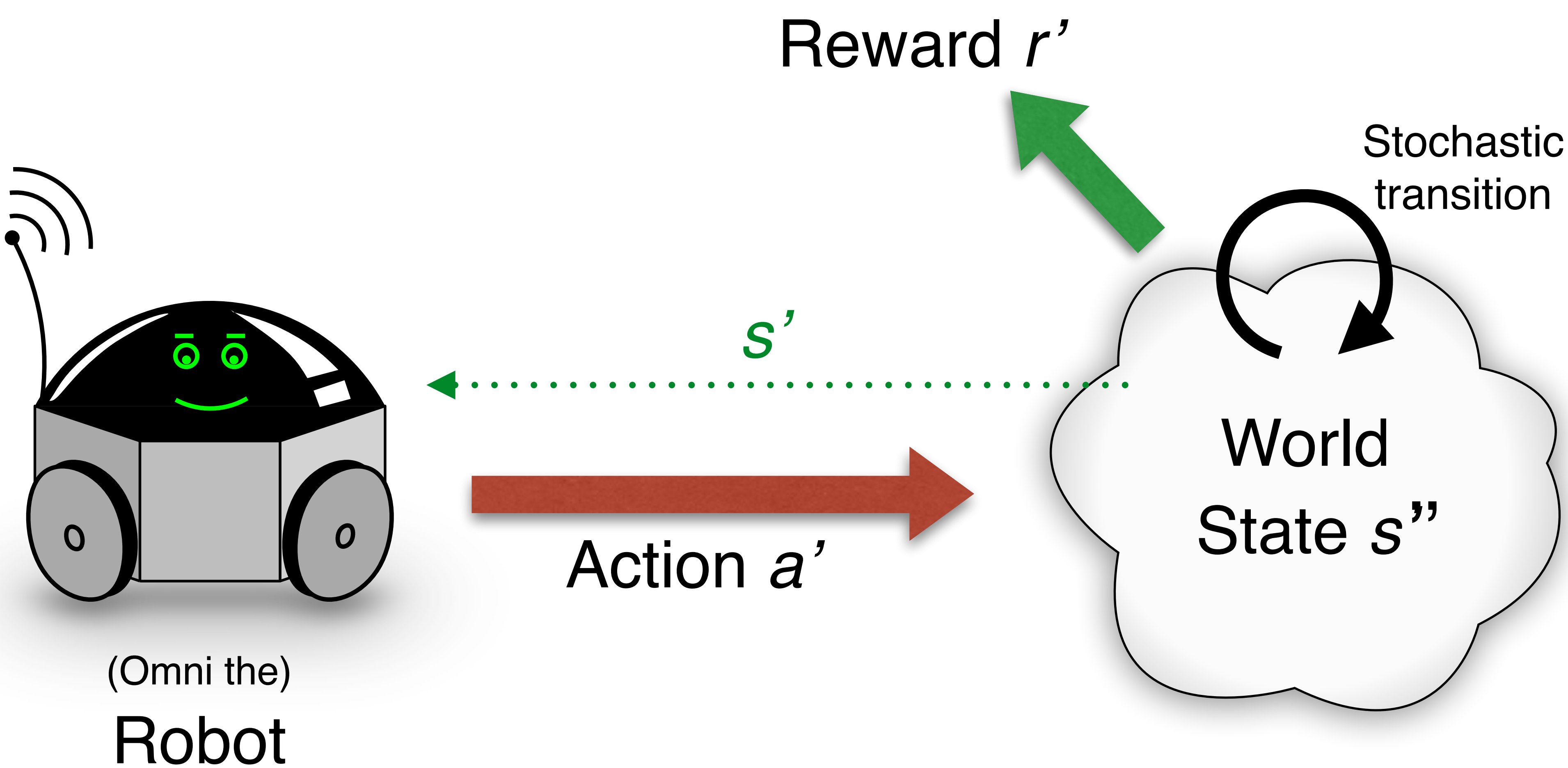
(not necessarily observed by the agent)

A Markov Decision Process:




The process is repeated for a certain number of steps

A Markov Decision Process:




**Objective:**
Find the actions that maximize a function of the reward collected over all steps

## Ingredients:

State space, $\mathcal{S}$ — A set describing the possible states of the system

Action space, $\mathcal{A}$ — A set describing the possible actions of the agent

Transition function, $T$
$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$
— A function describing the stochastic behaviour of the system
$$T(s, a, s') = \Pr(s' \mid s, a)$$

Reward function, $R$
$R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
— A function describing the utility or cost of each state and action

**The tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ is an MDP**

# An Example of a Real-World MDP

Autonomous Surveillance

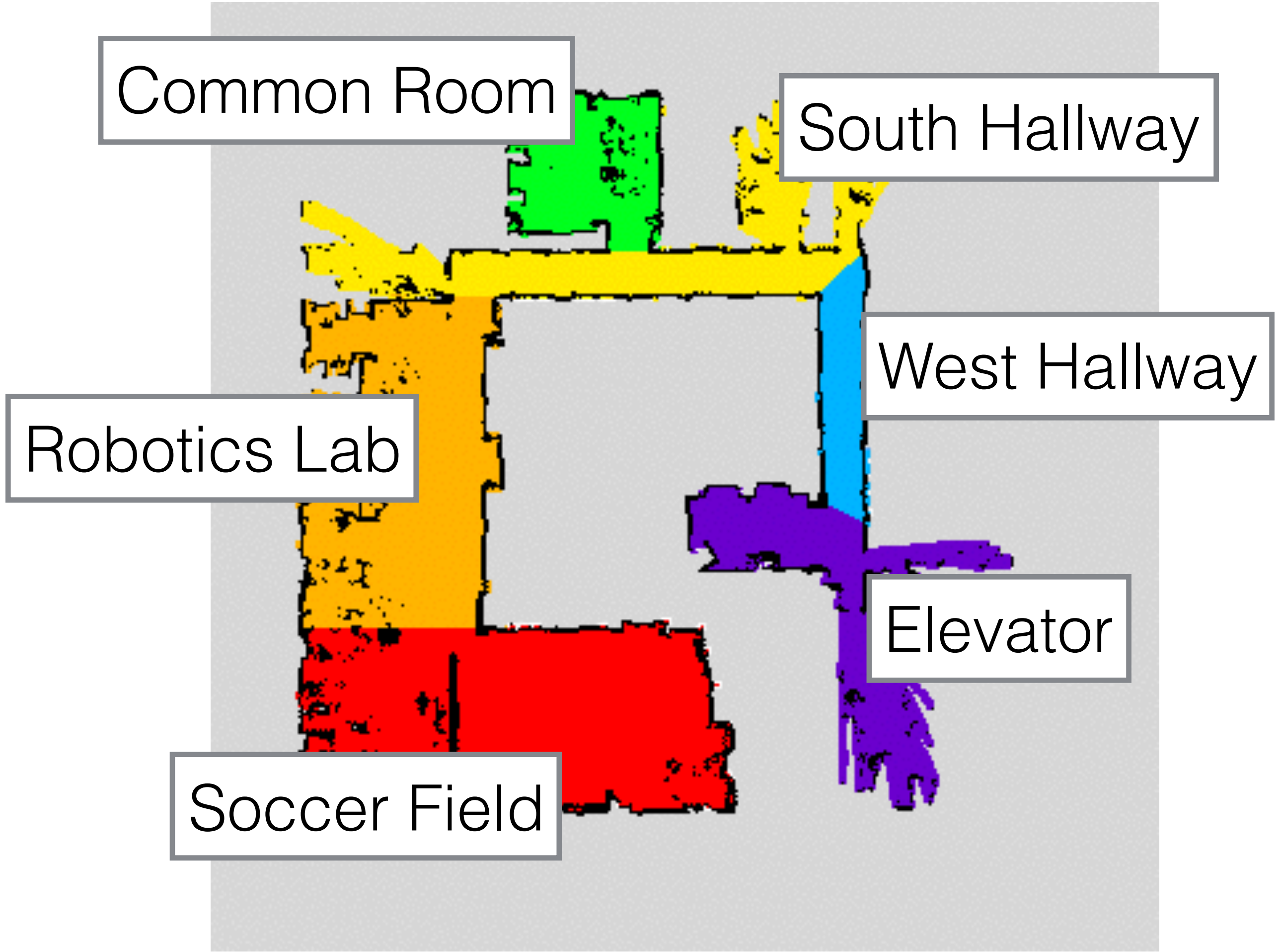
Problem statement: a mobile robot should patrol its environment in search of **visitors**, **trespassers**, and **emergencies**

# An Example of a Real-World MDP

Autonomous Surveillance

Real environment

State space abstraction

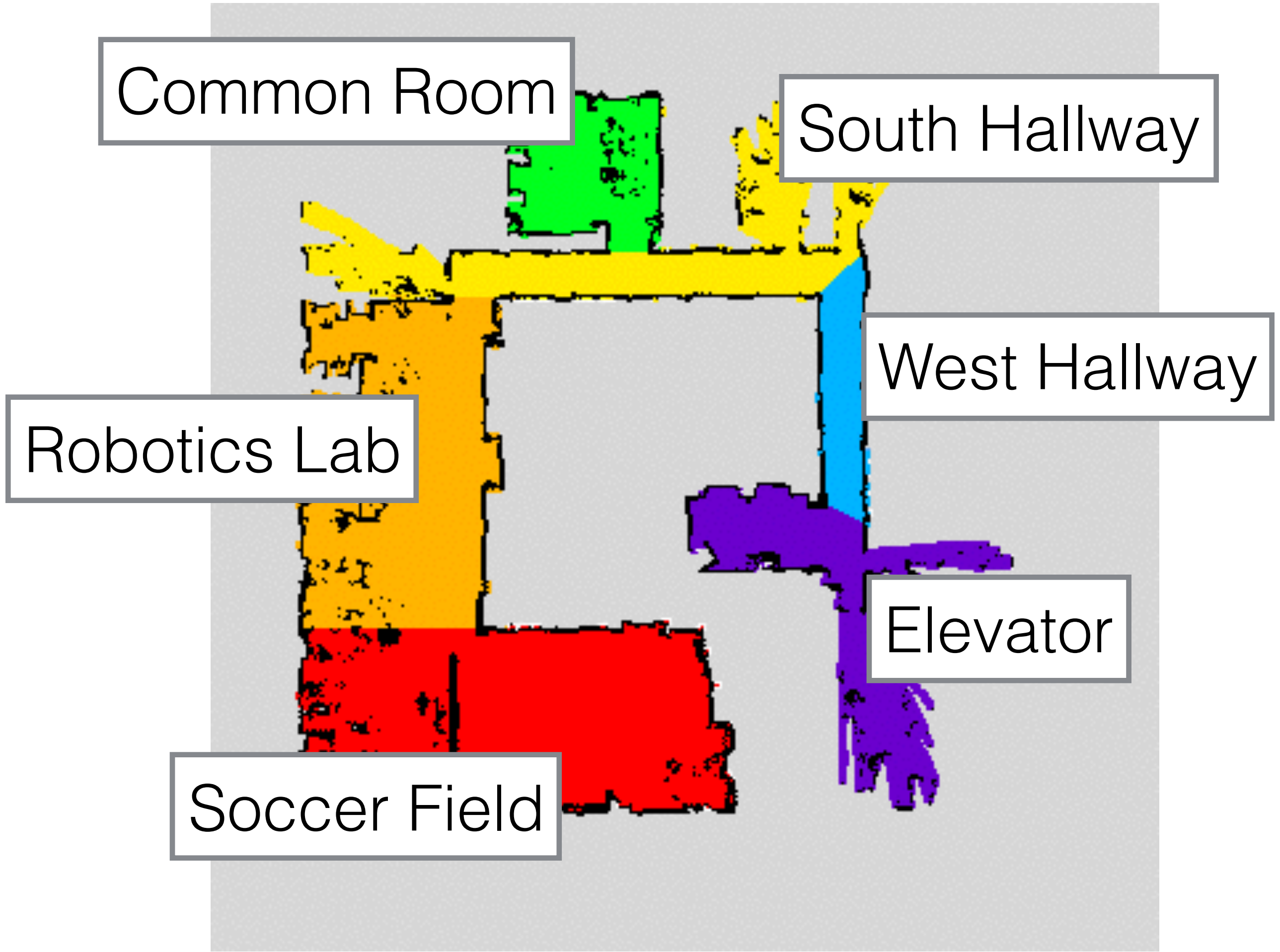


Autonomous Surveillance

State space =
{'location X visited'}
x {'visitor?'}
x {'trespassing?'}
x {'emergency?'}

Action space =
{'Up','Down','Left','Right'}

State space abstraction

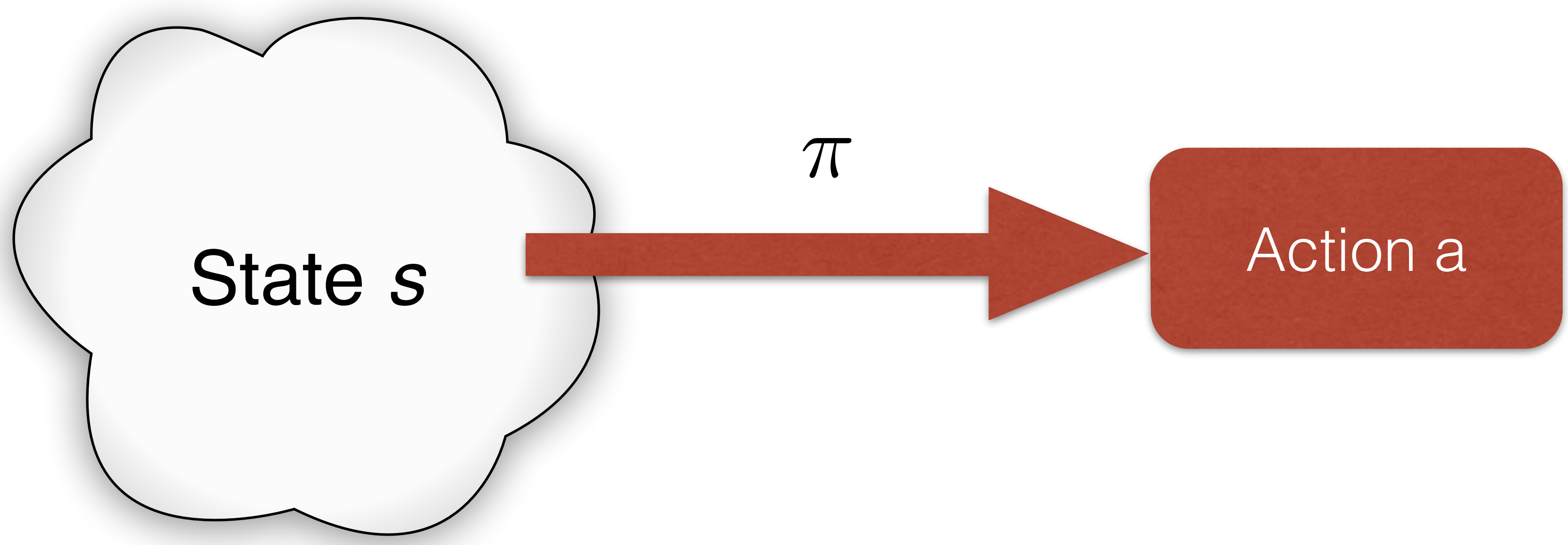


# MDP Policies

We are interested in determining a set of *decision rules*:

$$\pi = \{\delta_0, \delta_1, \ldots, \delta_{h-1}\}\,,\;\; \delta_n : \mathcal{S} \rightarrow \mathcal{A}$$

This is an *h*-horizon *policy* for the MDP agent




Remember: a policy is **not** simply a sequence of actions!

# Planning vs. Learning

A common measure of performance is the *expected discounted reward:*

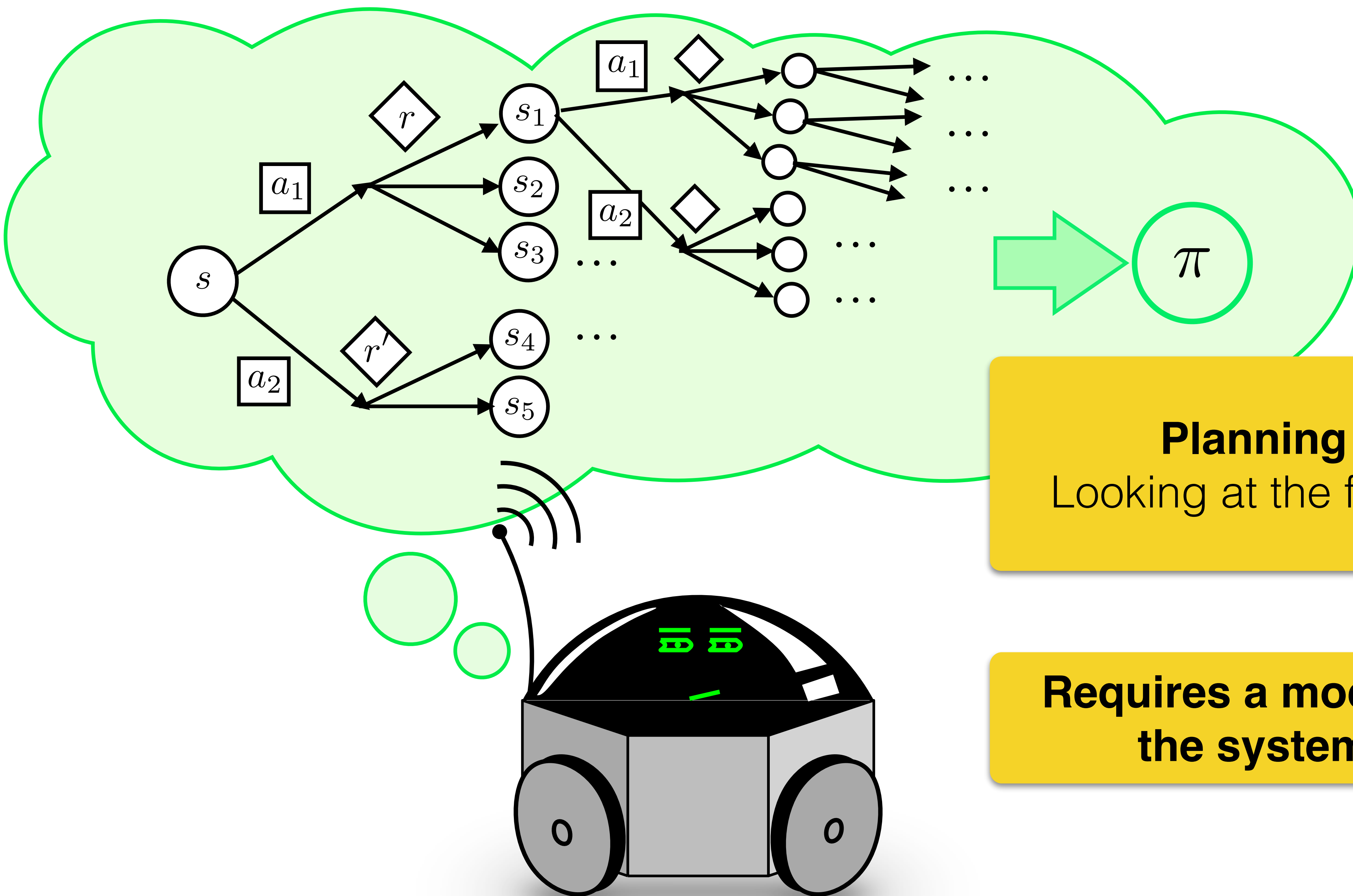
$$E_\pi \left\{ \sum_{n=0}^{h-1} \gamma^n R(s_n, \delta_n(s_n)) \right\}$$

$\gamma \in (0, 1]$ is a discount factor.

How do we find $\pi$ that maximizes this quantity?

# Planning vs. Learning

Two approaches: **Planning** and **Learning**

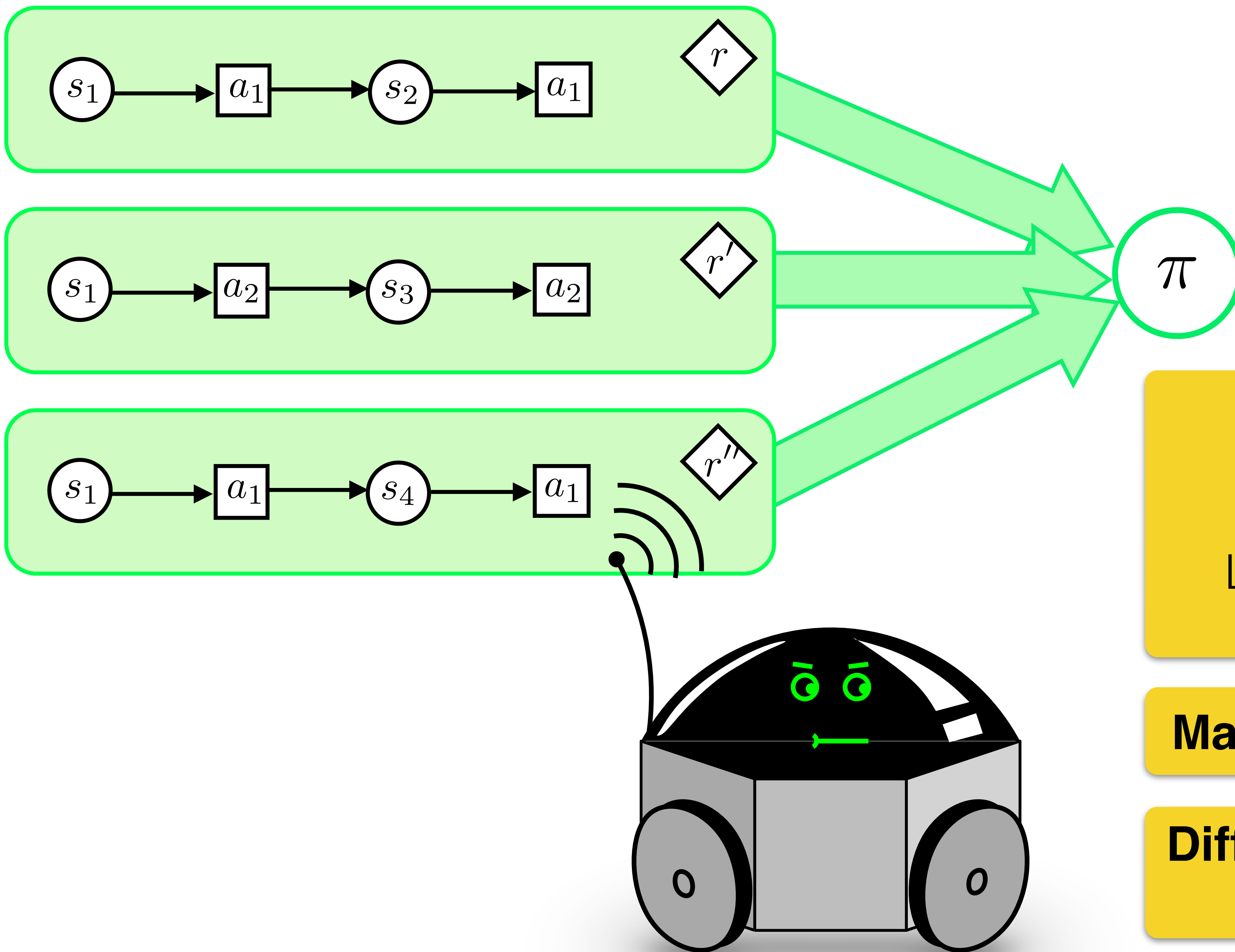



**Planning**
Looking at the future

**Requires a model of the system**

# Planning vs. Learning

Two approaches: **Planning** and **Learning**




**(Reinforcement) Learning**
Looking at the past

**Many trials are needed**

**Difficult for multi-agent systems.**

# The Value Function

The *expected discounted reward*, a.k.a. the *Value*

$$V_0^{\pi}(s) = E_{\pi}\left\{\sum_{n=0}^{h-1} \gamma^n R(s_n, \delta_n(s_n))\right\}$$

Can be calculated recursively as:

$$V_n^{\pi}(s) = \underbrace{R(s, \delta_n(s))}_{\text{Immediate Reward}} + \underbrace{\gamma}_{\text{Discount}} \underbrace{\sum_{s' \in \mathcal{S}} T(s, \delta_n(s), s') V_{n+1}^{\pi}(s')}_{\text{Future Reward}}$$

## Value Iteration

The best policy is the one that maximizes the expected value:

$$V_n^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{n+1}^*(s') \right\}$$

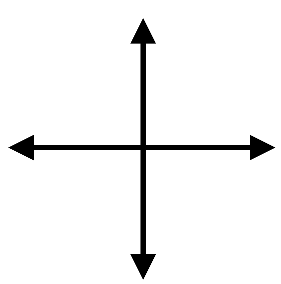
$$\delta_n^*(s) = \arg\max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{n+1}^*(s') \right\}$$

## Value Iteration

$$V_n^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{n+1}^*(s') \right\}$$

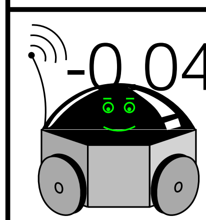
Reward of -0.04 for each step

Actions:



0.8 Prob. to move correctly;

0.2 Prob. to move in right angles.

| | | | |
|---|---|---|---|
| -0.04 | -0.04 | -0.04 | 1 |
| -0.04 | | -0.04 | -1 |
|  -0.04 | -0.04 | -0.04 | -0.04 |

Last Step

(reward = utility)

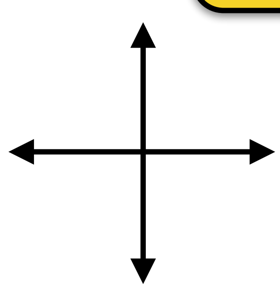
## Value Iteration

$$V_n^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{n+1}^*(s') \right\}$$

Reward of -0.04

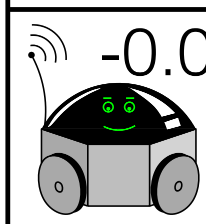
Action

$V^{east}(s) = -0.04 +$
$\Pr(\text{"move correctly"}) \times 1 +$
$\Pr(\text{"move incorrectly"}) \times -0.04$



0.8 Prob. to move correctly;

0.2 Prob. to move in right angles.

| | | | |
|---|---|---|---|
| | | 0.752 | 1 |
| -0.08 | | -0.08 | -1 |
| -0.08  | -0.08 | -0.08 | -0.08 |

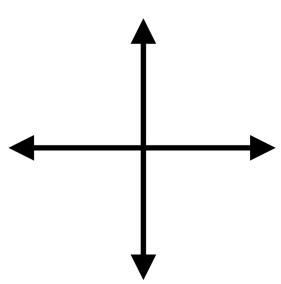
1 Step to go

## Value Iteration

$$V_n^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{n+1}^*(s') \right\}$$

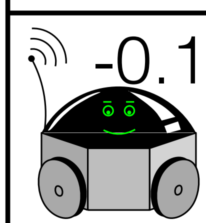
Reward of -0.04 for each step

Actions:



0.8 Prob. to move correctly;

0.2 Prob. to move in right angles.

| | | | |
|---|---|---|---|
| -0.12 | 0.546 | 0.827 | 1 |
| -0.12 | | 0.454 | -1 |
|  -0.12 | -0.12 | -0.12 | -0.12 |

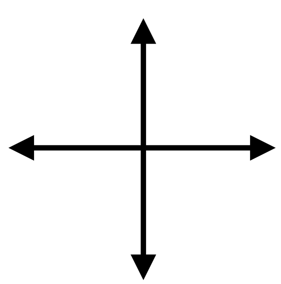
2 Steps to go

## Value Iteration

$$V_n^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{n+1}^*(s') \right\}$$

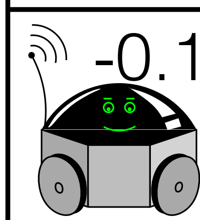
Reward of -0.04 for each step

Actions:



0.8 Prob. to move correctly;

0.2 Prob. to move in right angles.

| | | | |
|---|---|---|---|
| 0.372 | 0.731 | 0.888 | 1 |
| -0.16 | | 0.567 | -1 |
|  -0.16 | -0.16 | 0.299 | -0.16 |

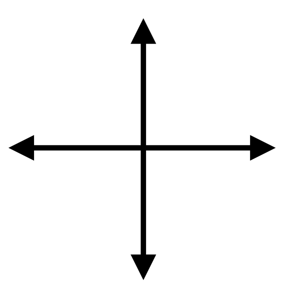
3 Steps to go

## Value Iteration

$$V_n^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{n+1}^*(s') \right\}$$

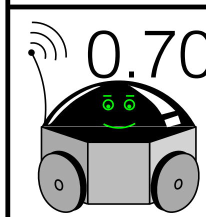
Reward of -0.04 for each step

Actions:



0.8 Prob. to move correctly;

0.2 Prob. to move in right angles.

| | | | |
|---|---|---|---|
| 0.812 | 0.868 | 0.918 | 1 |
| 0.762 | | 0.660 | -1 |
|  0.705 | 0.655 | 0.611 | 0.388 |

Final utilities

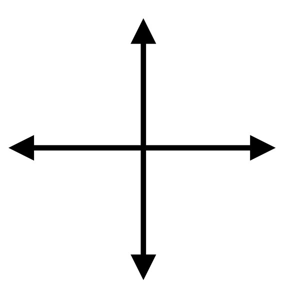
# Dynamic Programming

## Value Iteration

$$V_n^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{n+1}^*(s') \right\}$$

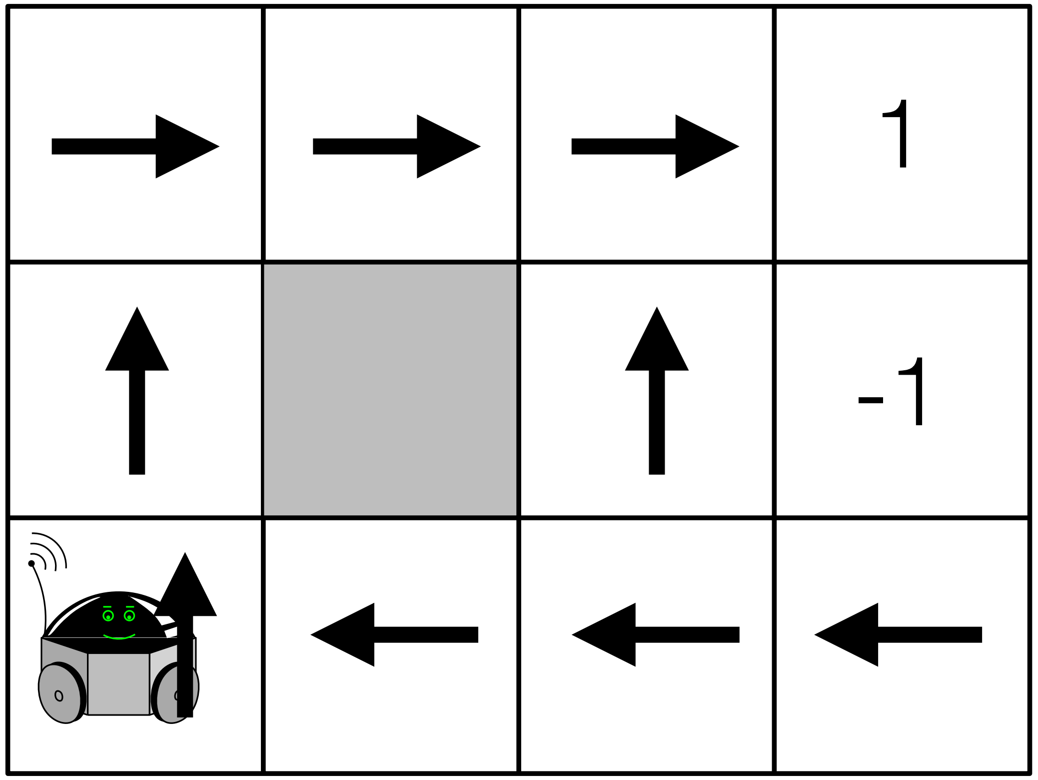
Reward of -0.04 for each step

Actions:



0.8 Prob. to move correctly;

0.2 Prob. to move in right angles.




Optimal Policy

## Policy Iteration

Another option is to search directly in the space of policies:

1. Pick a policy, $\pi$

2. Calculate the value

$$V_n^\pi(s) = R(s, \delta_n(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \delta_n(s), s') V_{n+1}^\pi(s')$$

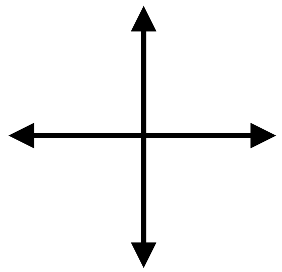
3. If we can improve the value by changing the first action, update $\pi$ accordingly.

## Policy Iteration

### Start with a random policy

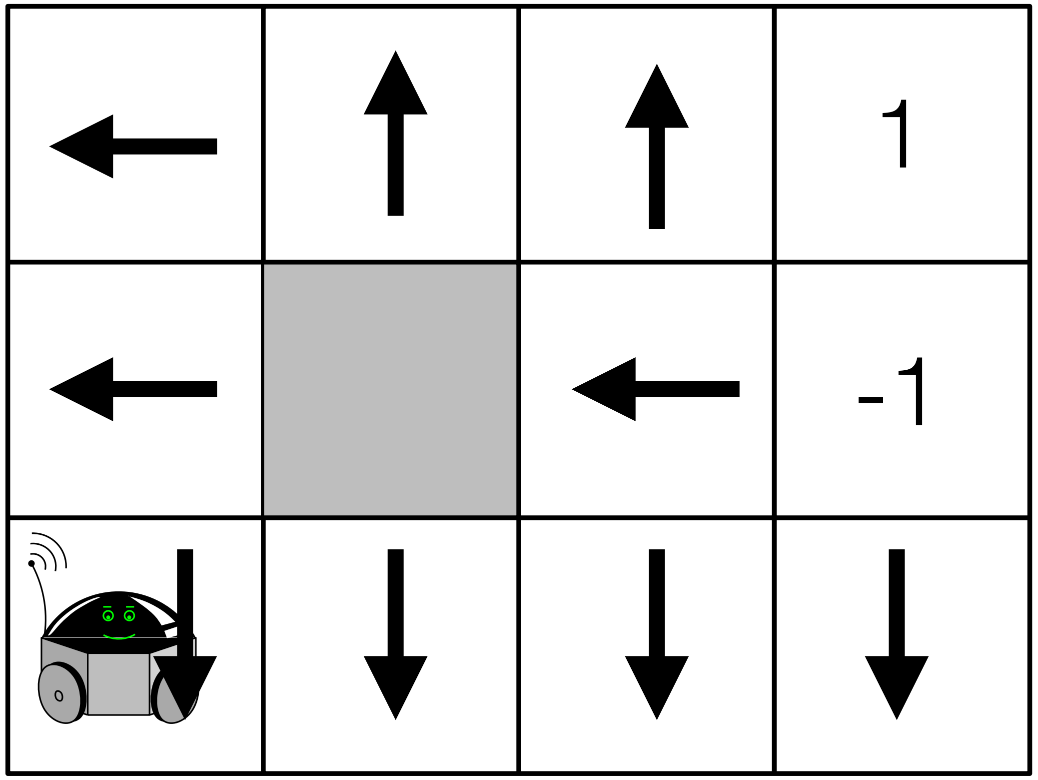
Reward of -0.04 for each step

Actions:



0.8 Prob. to move correctly;

0.2 Prob. to move in right angles.




Remember: a *policy* prescribes actions for **every state.**

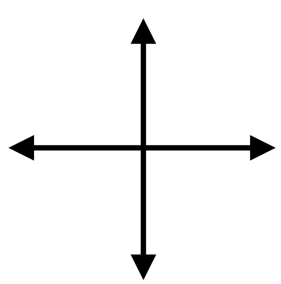
## Policy Iteration

### Evaluate until convergence:

$$V_n^\pi(s) = R(s, \delta_n(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \delta_n(s), s') V_{n+1}^\pi(s')$$

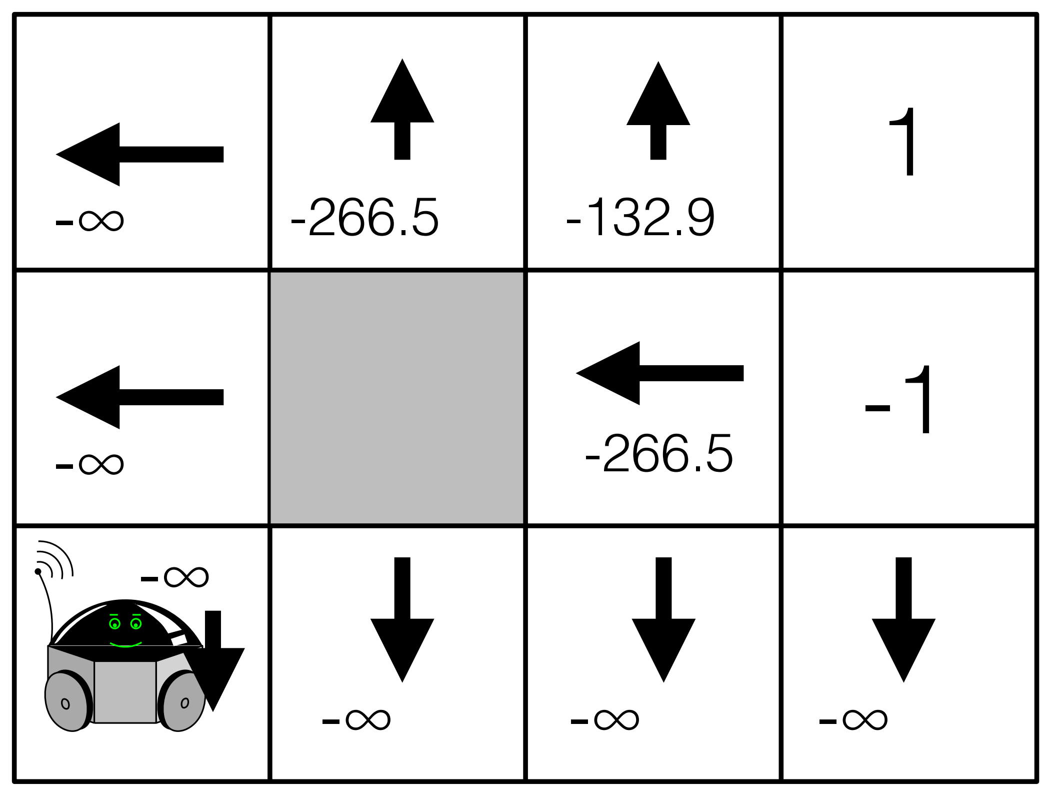
Reward of -0.04 for each step

Actions:



0.8 Prob. to move correctly;

0.2 Prob. to move in right angles.



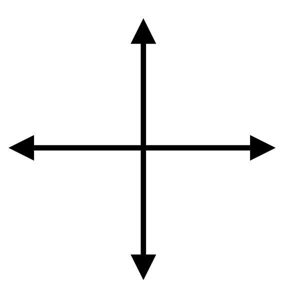
## Policy Iteration

### For **each** state, calculate:

$$\delta_n(s) = \arg\max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^{\pi}_{n+1}(s') \right\}$$

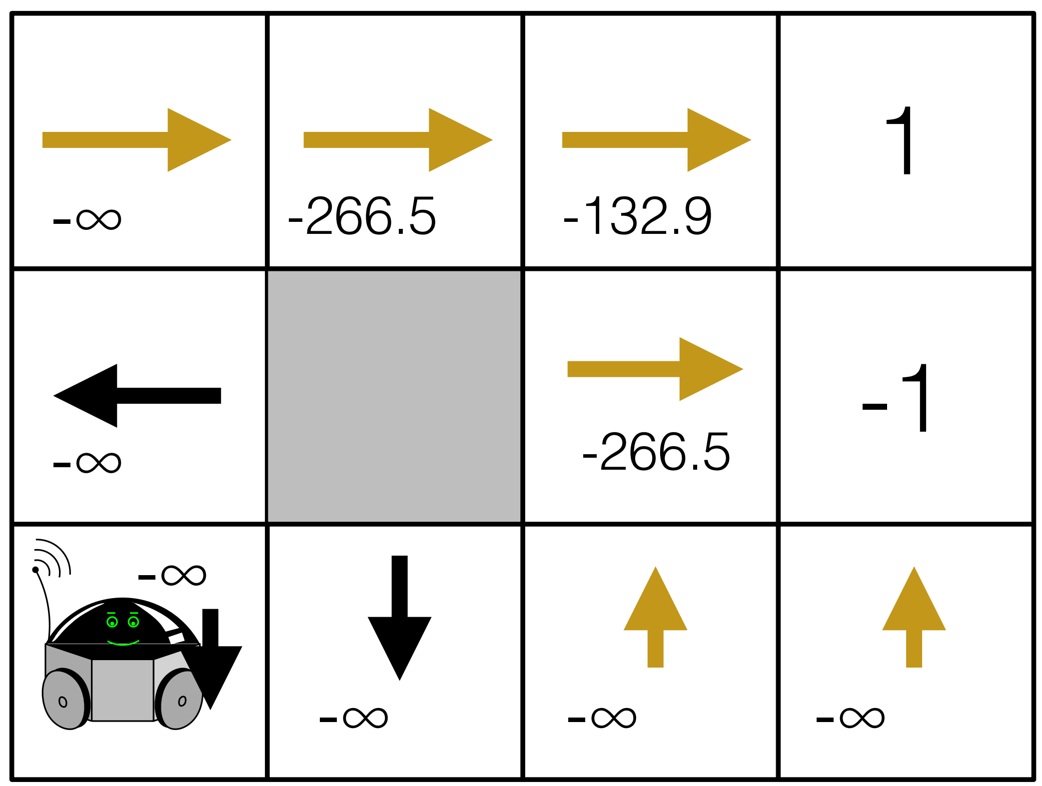
Reward of -0.04 for each step

Actions:



0.8 Prob. to move correctly;

0.2 Prob. to move in right angles.

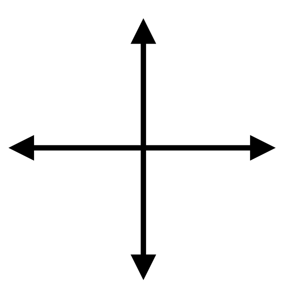


## Policy Iteration

### Repeat evaluation

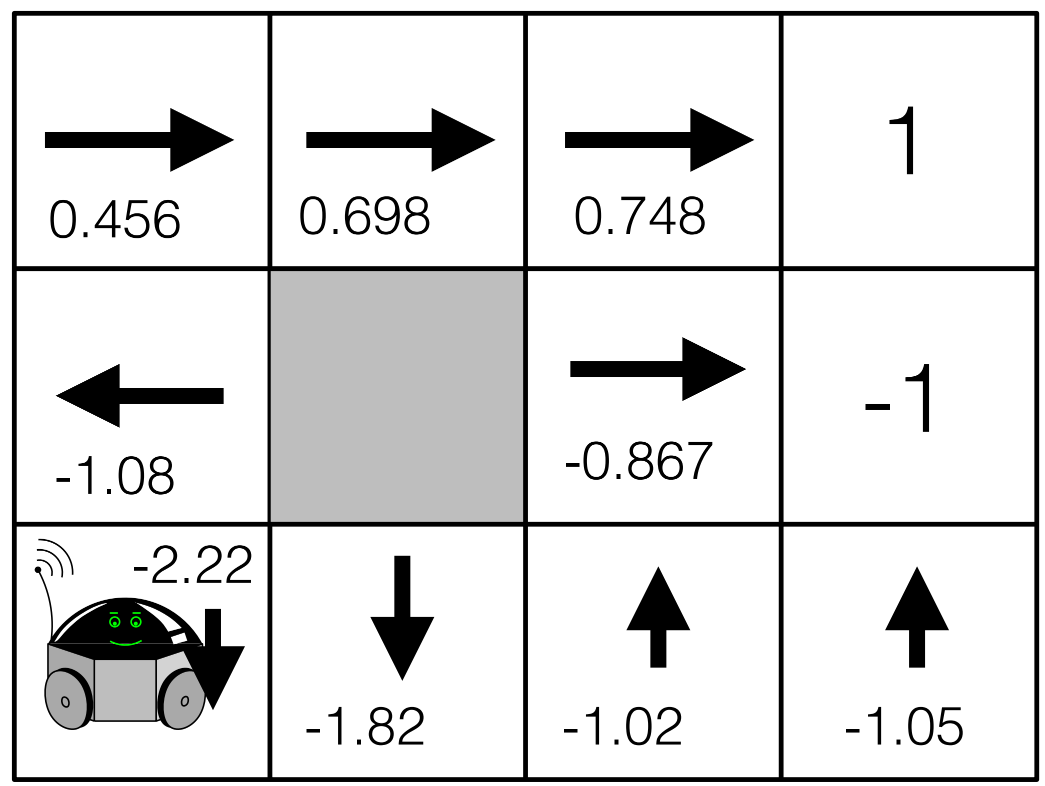
Reward of -0.04 for each step

Actions:



0.8 Prob. to move correctly;

0.2 Prob. to move in right angles.



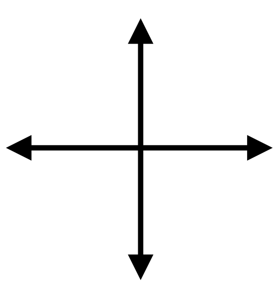
## Policy Iteration

### Update again:

$$\delta_n(s) = \arg\max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^{\pi}_{n+1}(s') \right\}$$

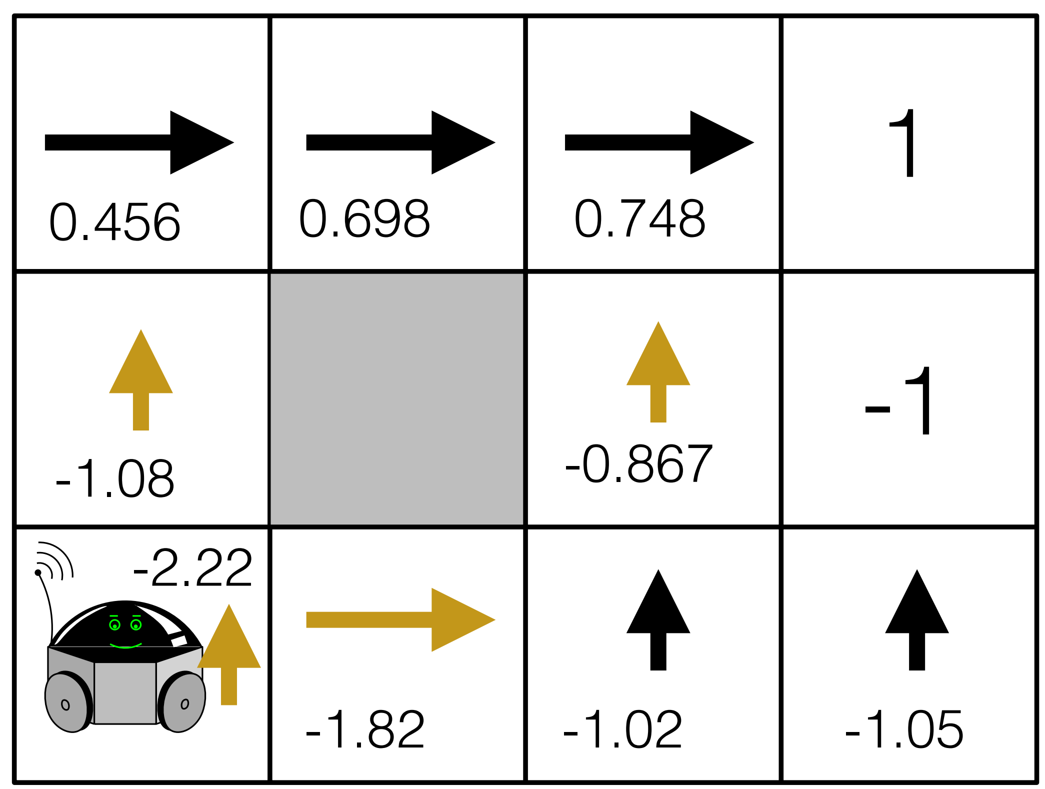
Reward of -0.04 for each step

Actions:



0.8 Prob. to move correctly;

0.2 Prob. to move in right angles.

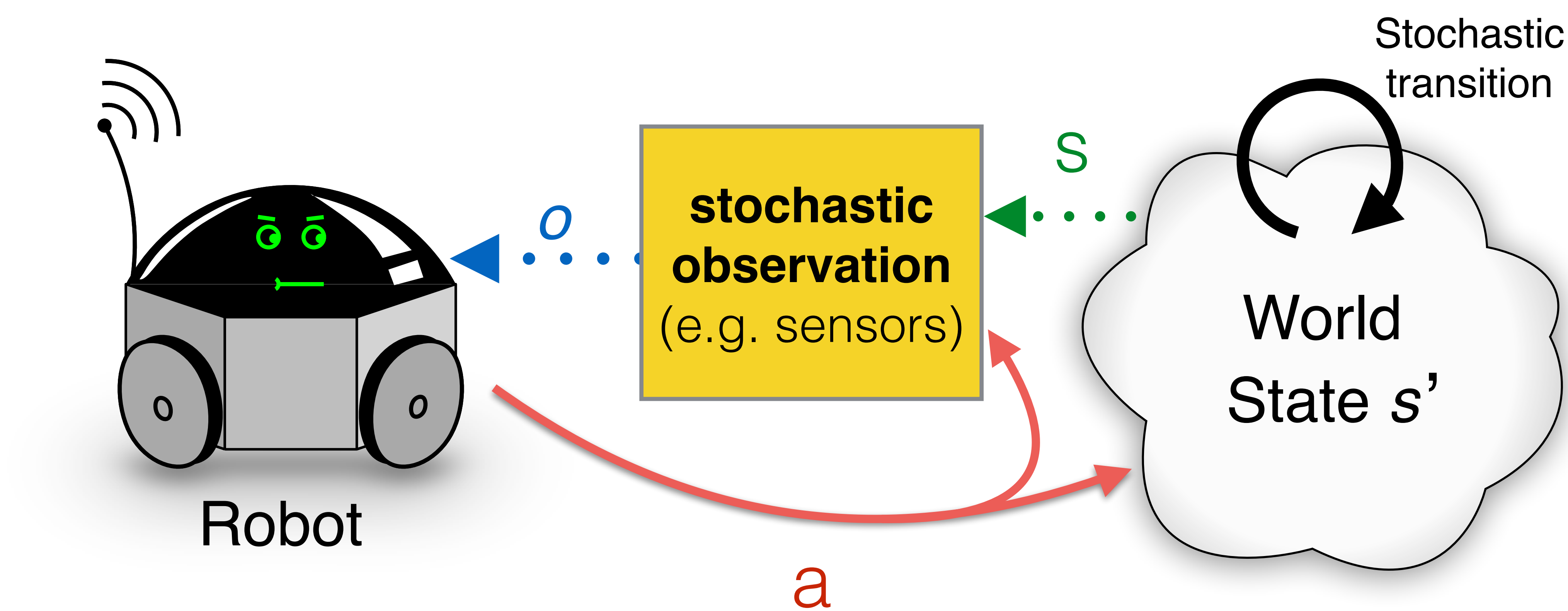



…will converge to the same optimal policy

# Partial Observability

So far we have assumed that the state is known to the agent.

In many domains the state is *not* known with certainty,
but it can be *estimated*.




Partially Observable Markov Decision Process (POMDP)

***Partial Observability* can mean:**

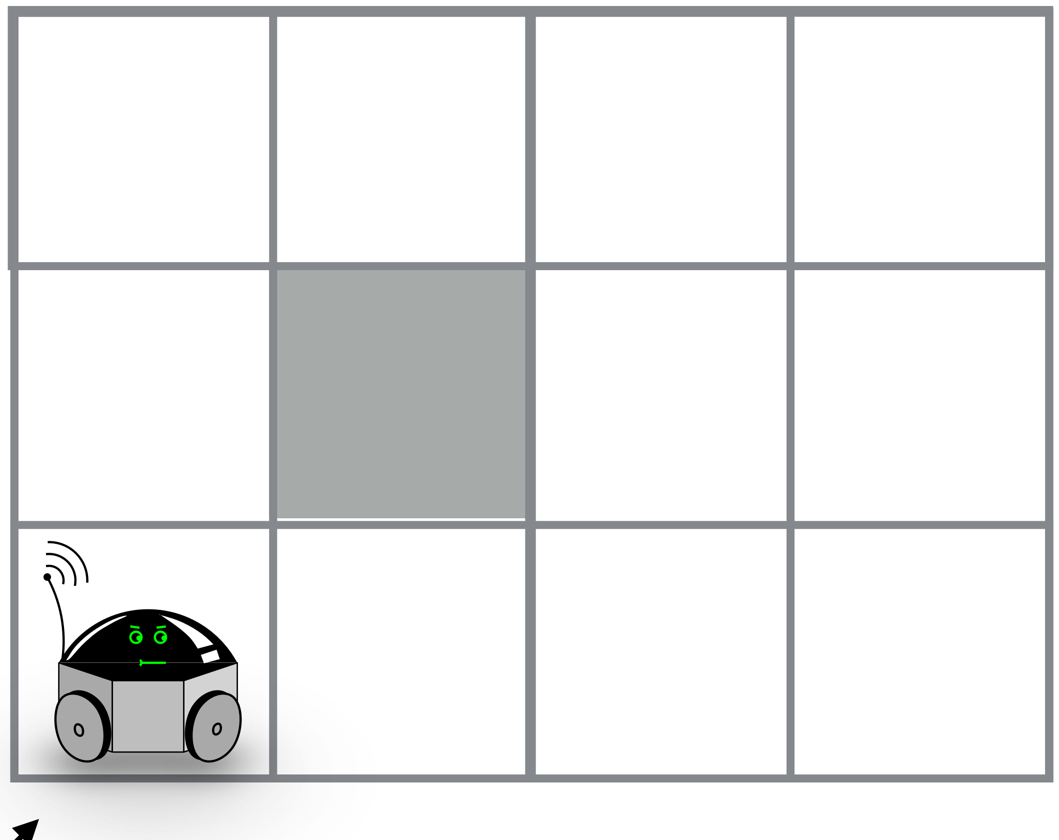
1. Incomplete or partial knowledge regarding the state (*perceptual aliasing*);
2. The observation of noisy, possibly misleading information.

The actions of a POMDP agent depend on a *probability distribution* over the states of the system.

$$b_n(x) = \Pr(s_n = x \,|\, b_0, a_0, o_0, \ldots, a_{n-1}, o_{n-1}, o_n)$$

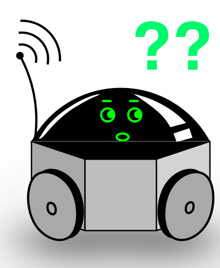
Also known as a *belief state*

# Belief States



Instead of *knowing* that we are in state *s*…

# Belief States

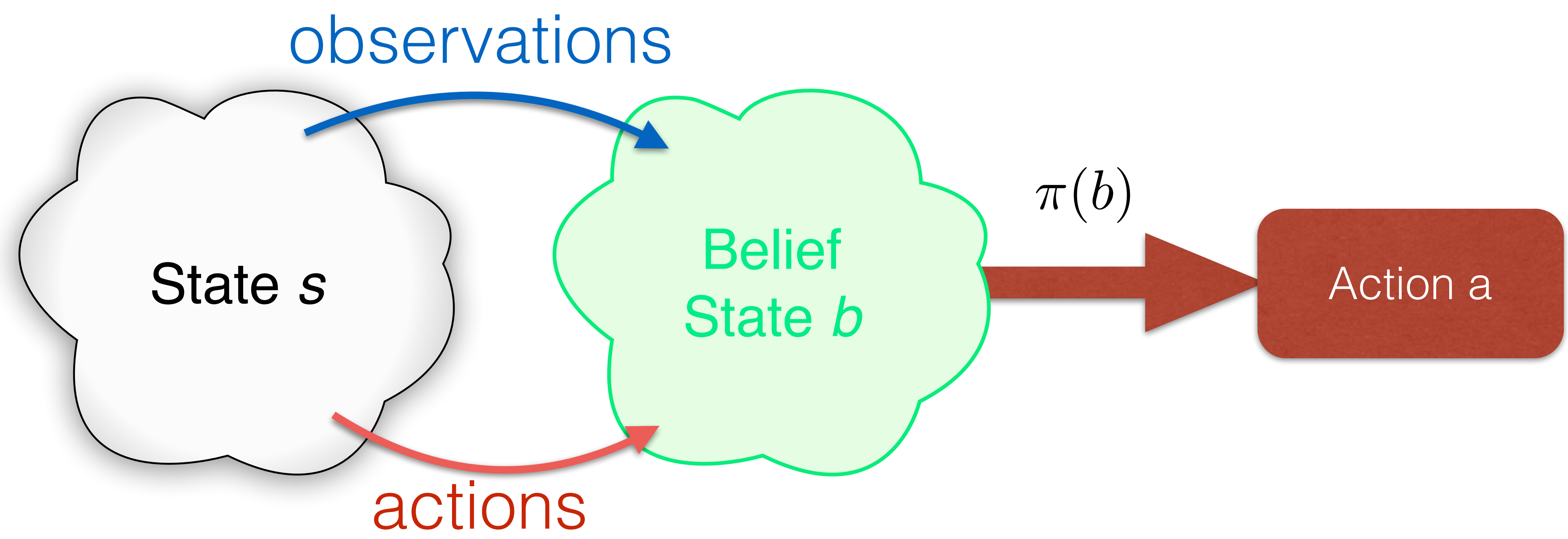



| | | | |
|---|---|---|---|
| Pr = 0.05 | Pr = 0.025 | Pr = 0 | Pr = 0 |
| Pr = 0.1 | | Pr = 0.025 | Pr = 0 |
| Pr = 0.65 | Pr = 0.1 | Pr = 0.05 | Pr = 0 |

…There is a probability distribution over the state of the system.

This distribution depends on the agent's actions and observations.

A POMDP policy is a map from belief states to actions



# Belief States

Belief states are updated after each action and observation:

the probability of the observation *o*

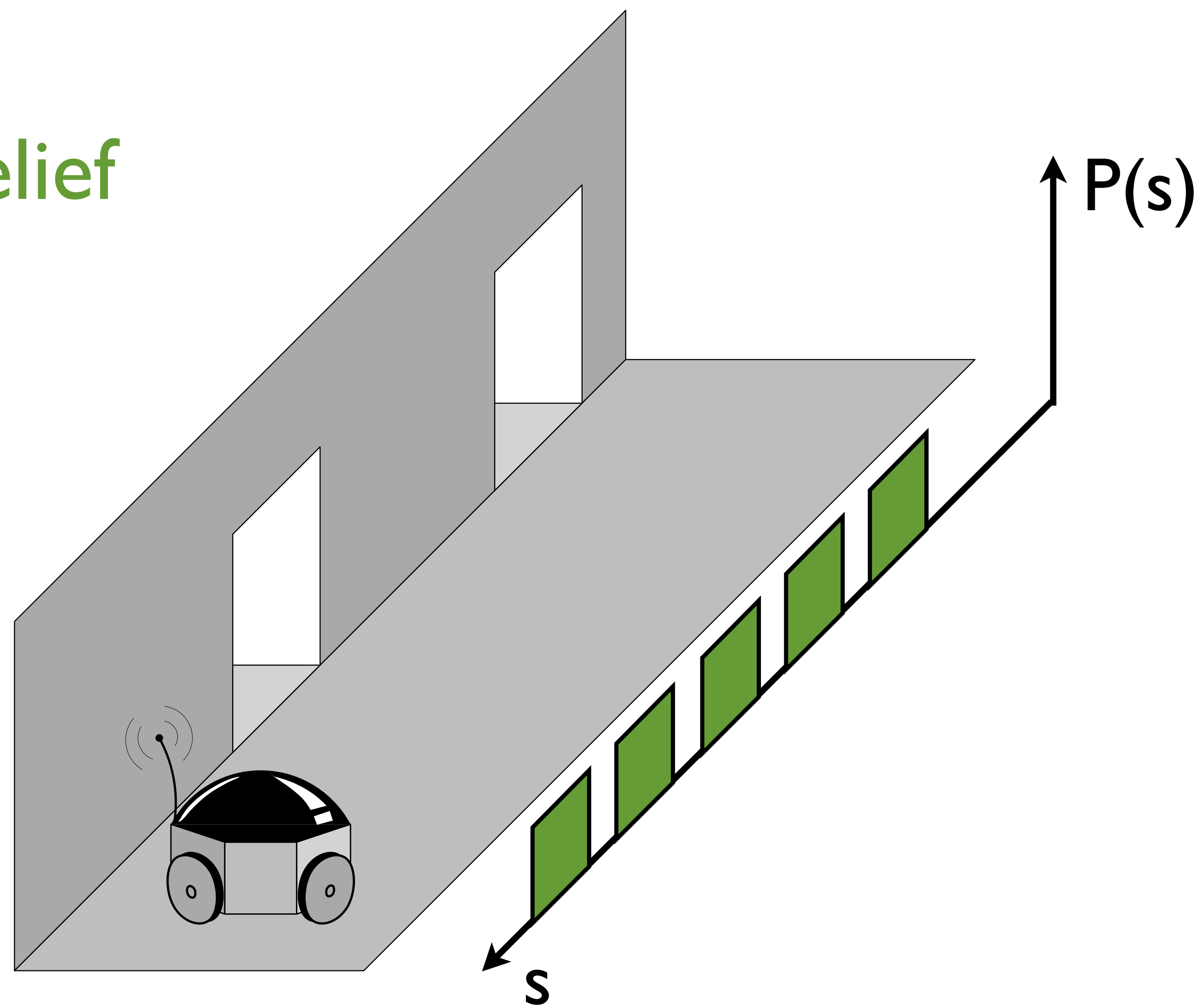
the probability of being in state *s* and jumping to *s'*

$$b^{a,o}(s') = \frac{O(a, s', o) \sum_{s \in \mathcal{S}} b(s) T(s, a, s')}{\sum_{u' \in \mathcal{S}} O(a, u', o) \sum_{u \in \mathcal{S}} b(u) T(u, a, u')}$$

Normalization (all possible ways of observing *o* after any transition)

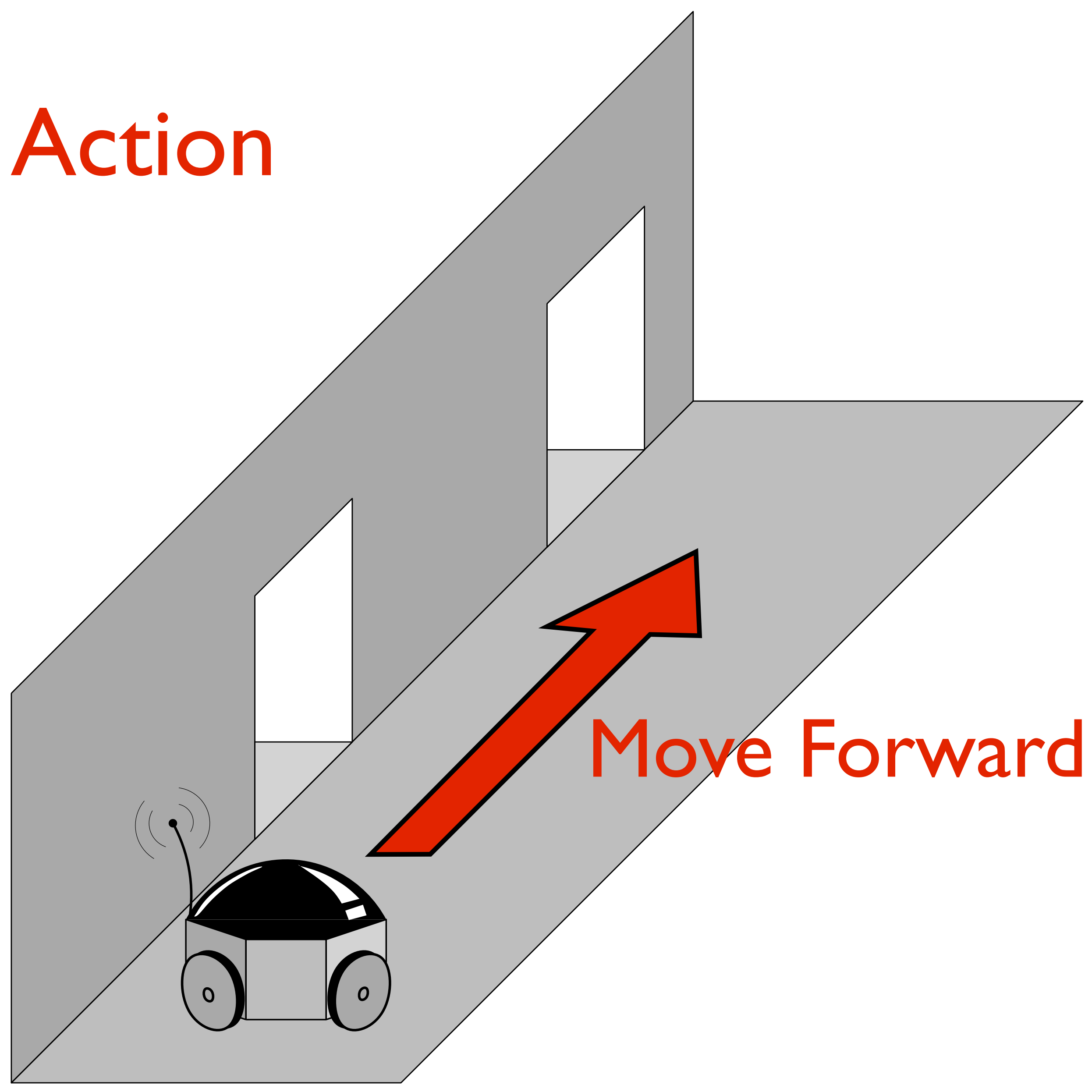
Belief states are updated after each action and observation:

Initial Belief



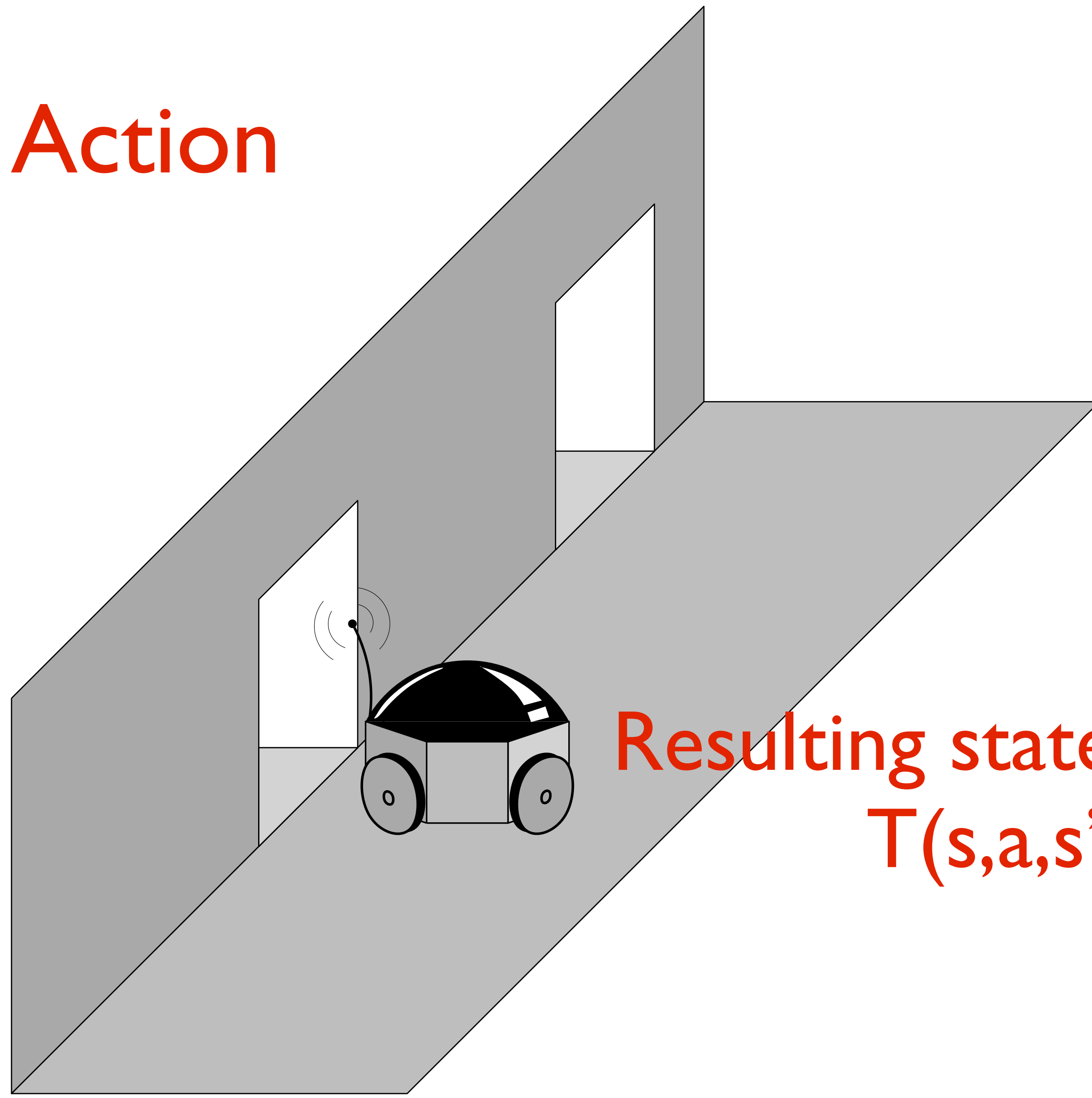
Belief states are updated after each action and observation:

1. Select an Action



Belief states are updated after each action and observation:

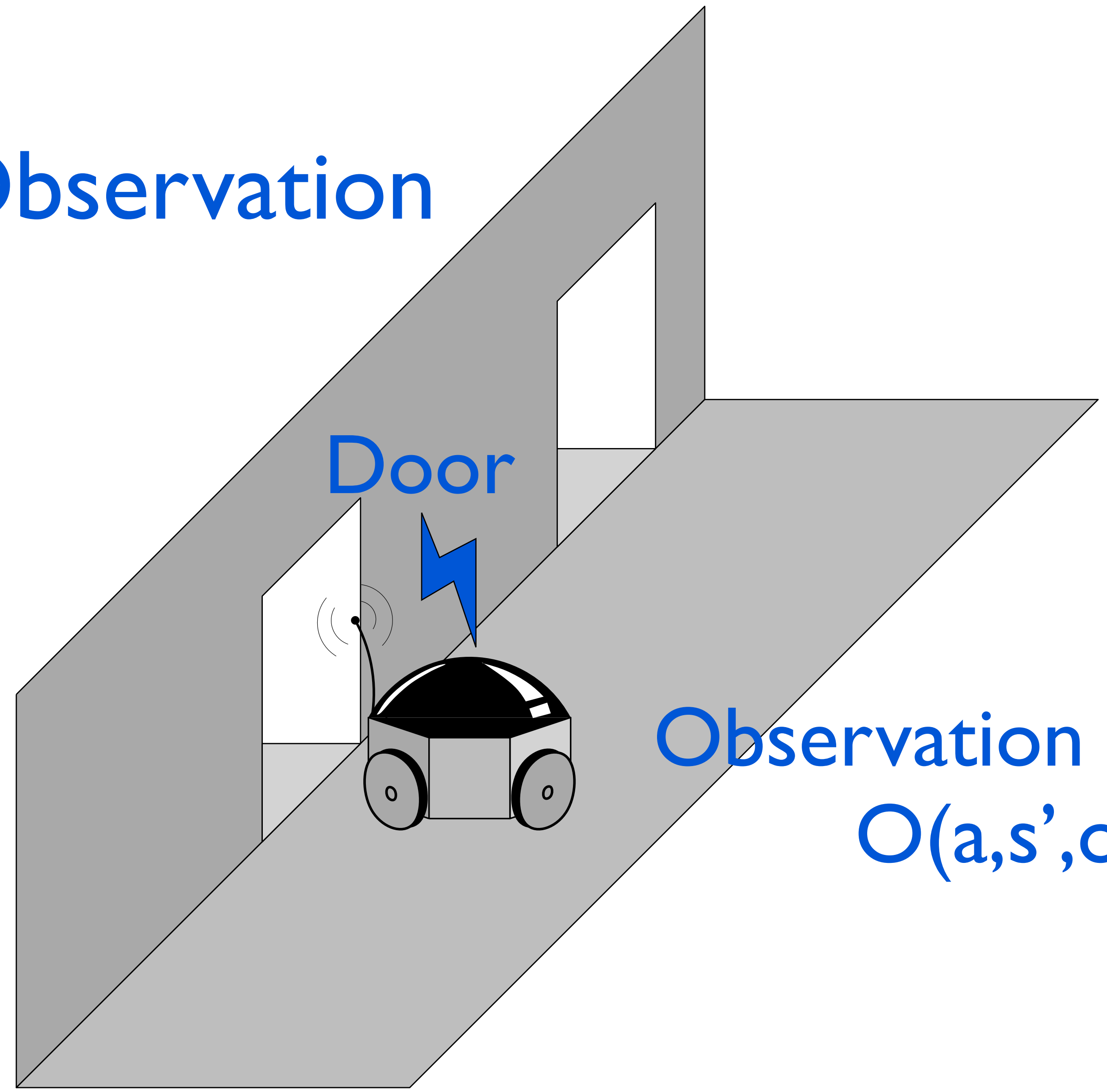
1. Select an Action



Resulting state follows T(s,a,s')

Belief states are updated after each action and observation:

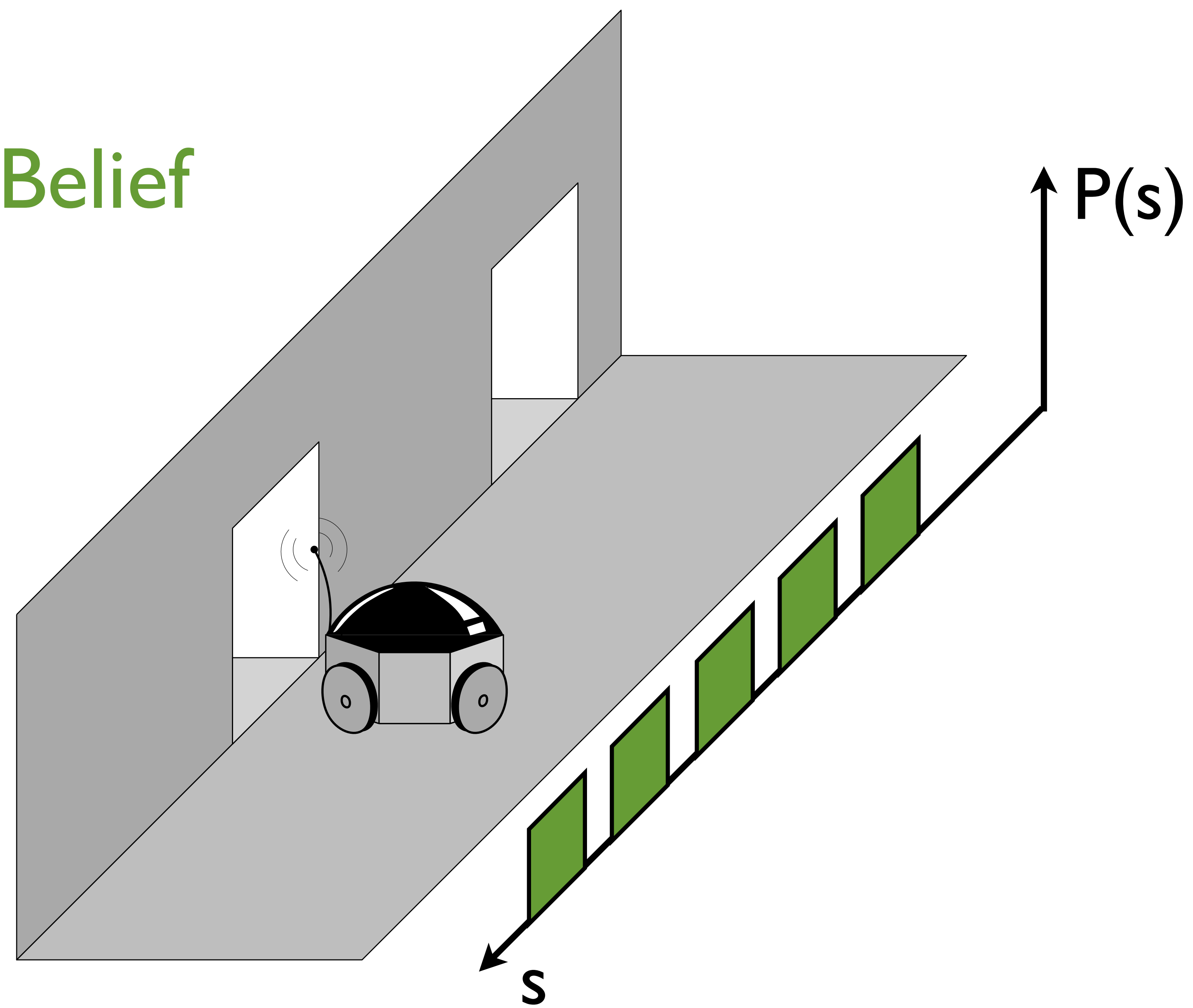
## 2. Receive an Observation




Observation follows
O(a,s’,o)

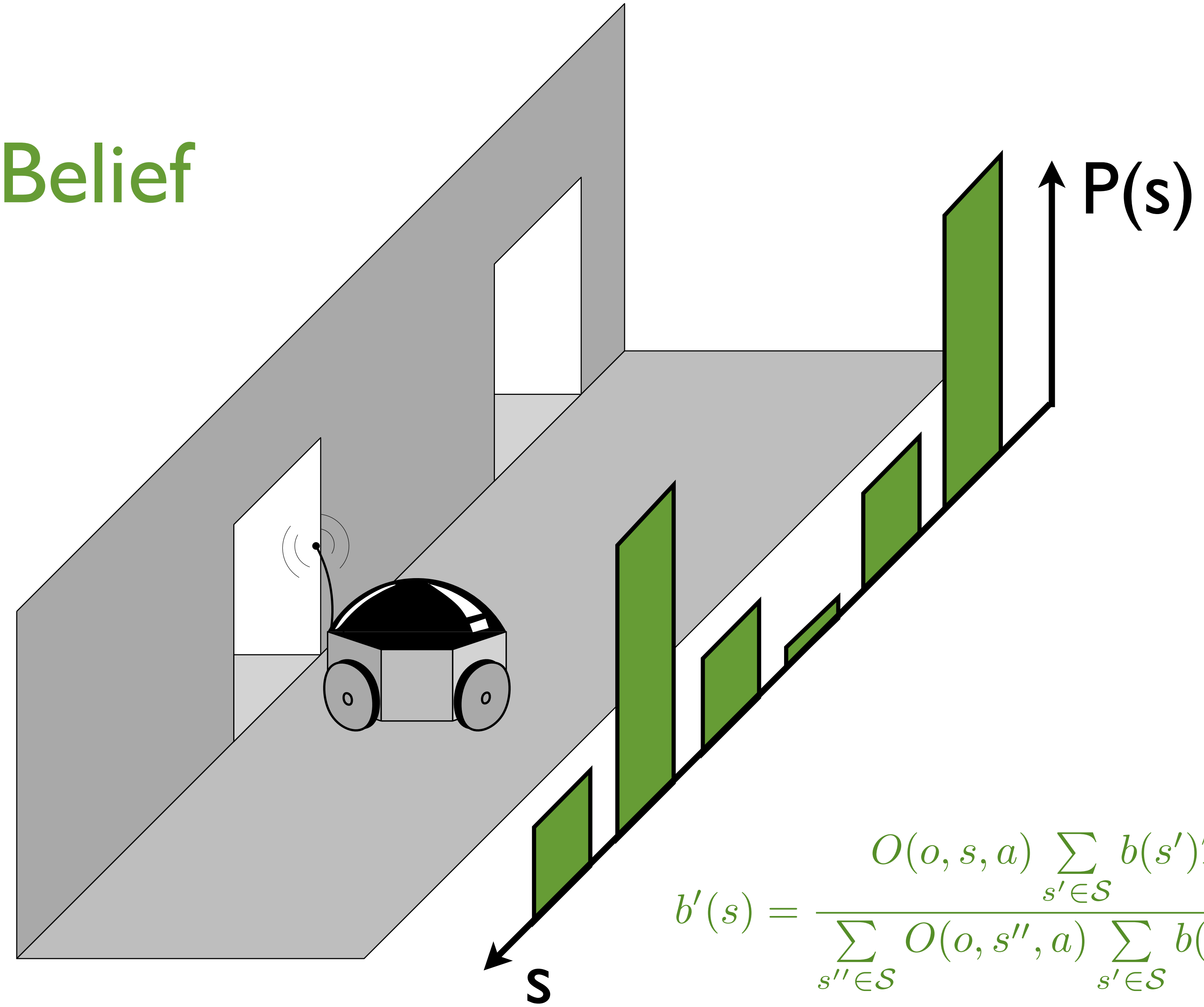
Belief states are updated after each action and observation:

3. Update Belief



Belief states are updated after each action and observation:

3. Update Belief




$$b'(s) = \frac{O(o,s,a)\sum_{s'\in\mathcal{S}} b(s')T(s',s,a)}{\sum_{s''\in\mathcal{S}} O(o,s'',a)\sum_{s'\in\mathcal{S}} b(s')T(s',s,a)}$$

# POMDP Value Functions

Value Functions can also be calculated recursively for POMDPs:

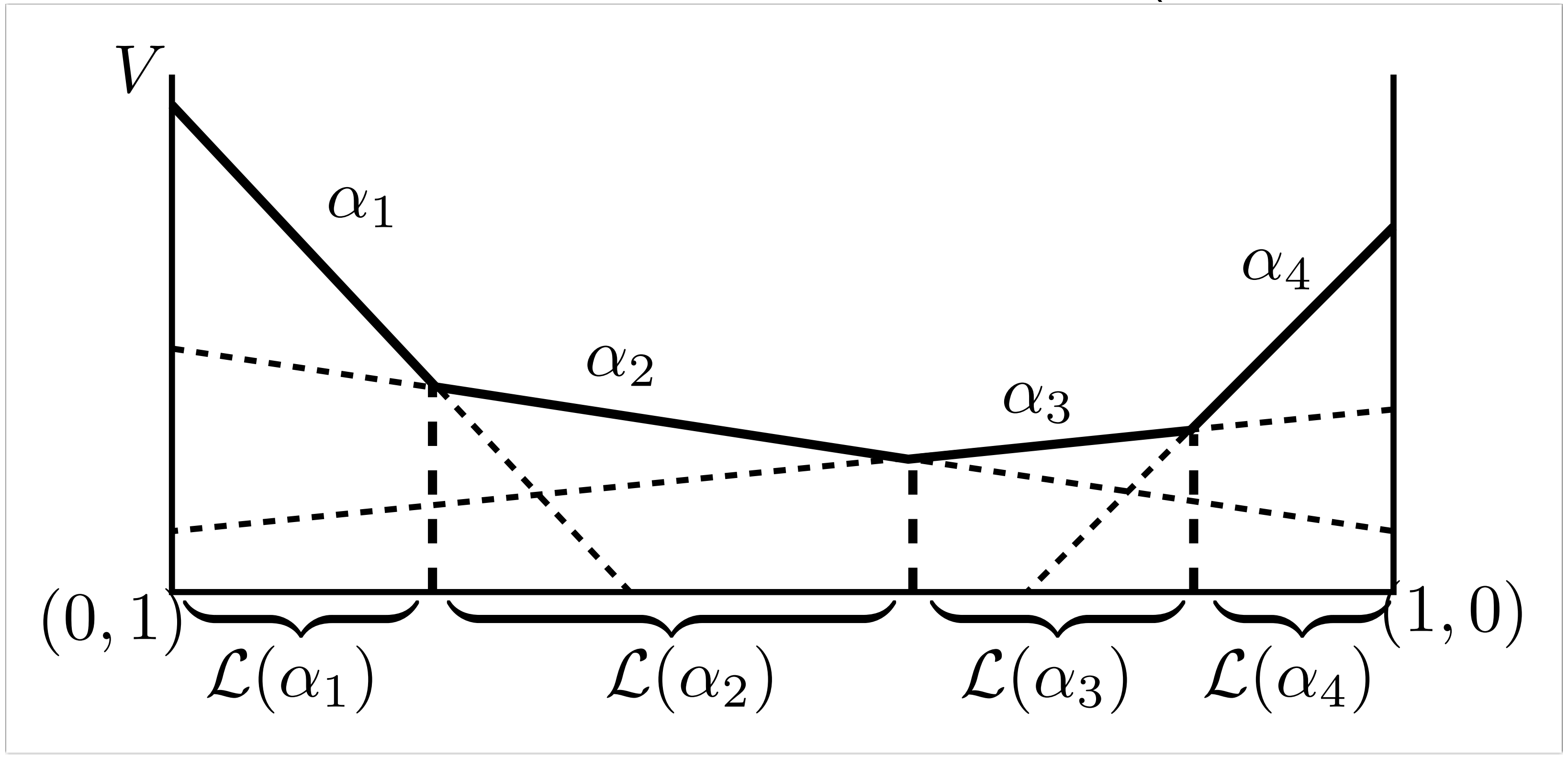
$$V_n^*(b) = \max_{a \in \mathcal{A}} \left\{ \sum_{s \in \mathcal{S}} b(s) \left( R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{o \in \mathcal{O}} T(s,a,s') O(a,s',o) V_{n+1}^*(b^{a,o}) \right) \right\}$$

Instead of a table, this is now a continuous function over the space of possible probability distributions

As it turns out, POMDP Value Functions have useful properties:

$$V_n^*(b) = \max_{\alpha \in \Gamma_n} \left\{ \sum_{s \in S} b(s)\alpha(s) \right\}$$

Inner product with a set of vectors (one for each <a,o> at step n)




Piecewise Linear and Convex (PWLC)

Solving a POMDP optimally is a difficult problem

(MDPs are P-complete, POMDPs are PSPACE-hard)

**Solution by enumeration** (Monahan, '82)

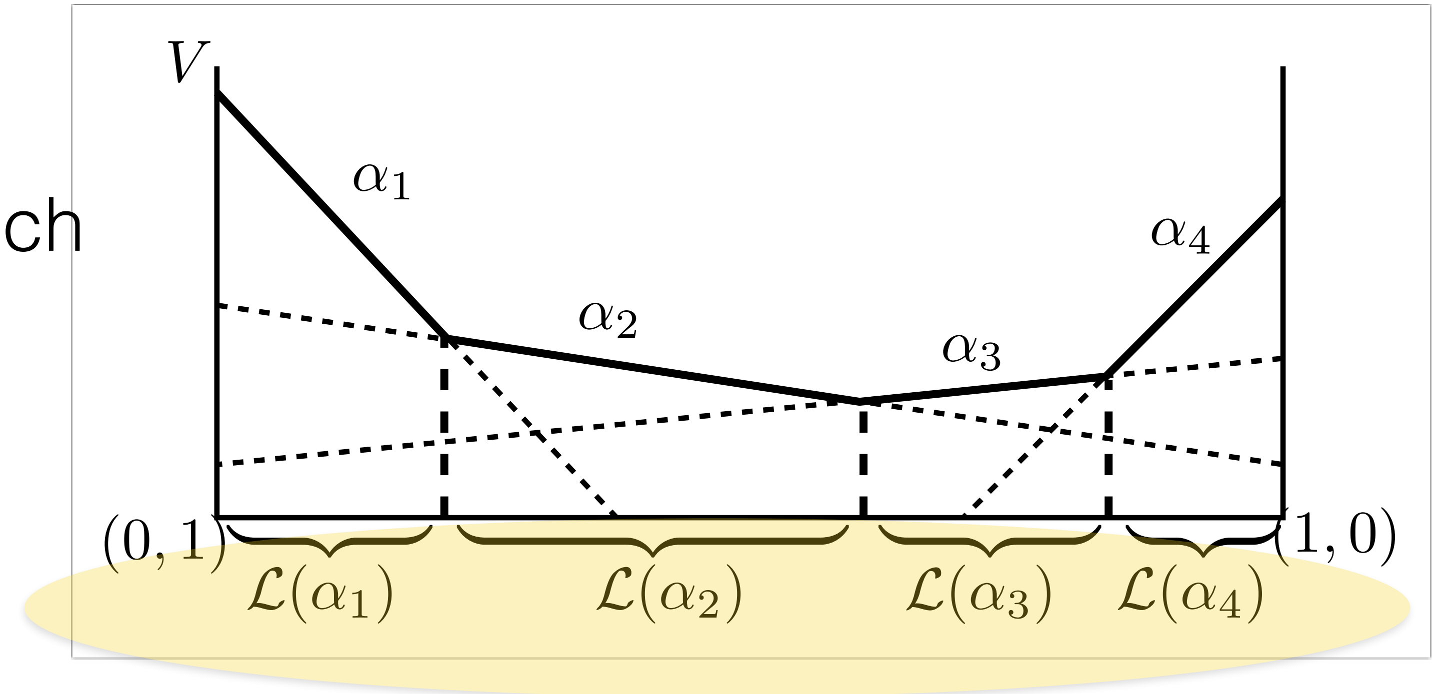
1. Compute all vectors; $$V_n^*(b) = \max_{\alpha \in \Gamma_n} \left\{ \sum_{s \in \mathcal{S}} b(s)\alpha(s) \right\}$$
2. Pick the best at b.

The number of vectors is really big!

$$|\mathcal{A}|^{\frac{|O|^{h+1}-1}{|O|-1}}$$

## Linear Support Methods

Calculate the regions for which each vector is best (Linear Programming)



## Point-Based Methods

Backing up **one** belief state is easy!

3. 2. 1.

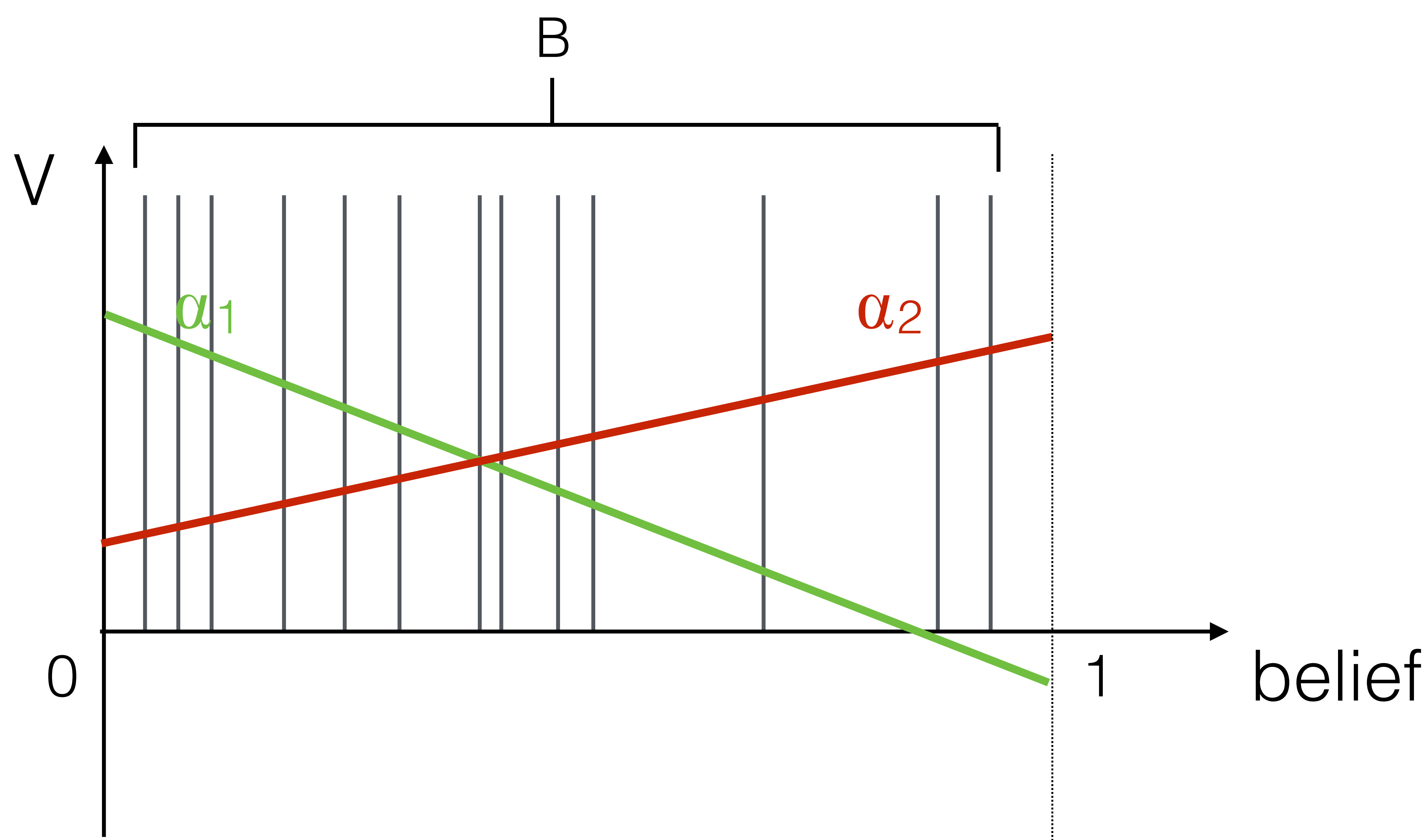
$$\alpha_n^{a,b} = \alpha_h^a + \gamma \sum_{o \in \mathcal{O}} \arg \max_{\alpha_{n+1} \in \Gamma_{n+1}} \sum_{s \in \mathcal{S}} b(s) \sum_{s' \in \mathcal{S}} T(s,a,s') O(a,s',o) \alpha_{n+1}(s)$$

1. Take vectors at n+1;
2. Plug in b;
3. Get one optimal vector at n (at b).

Select belief points randomly
(or by exploration)
and find the vectors for each.

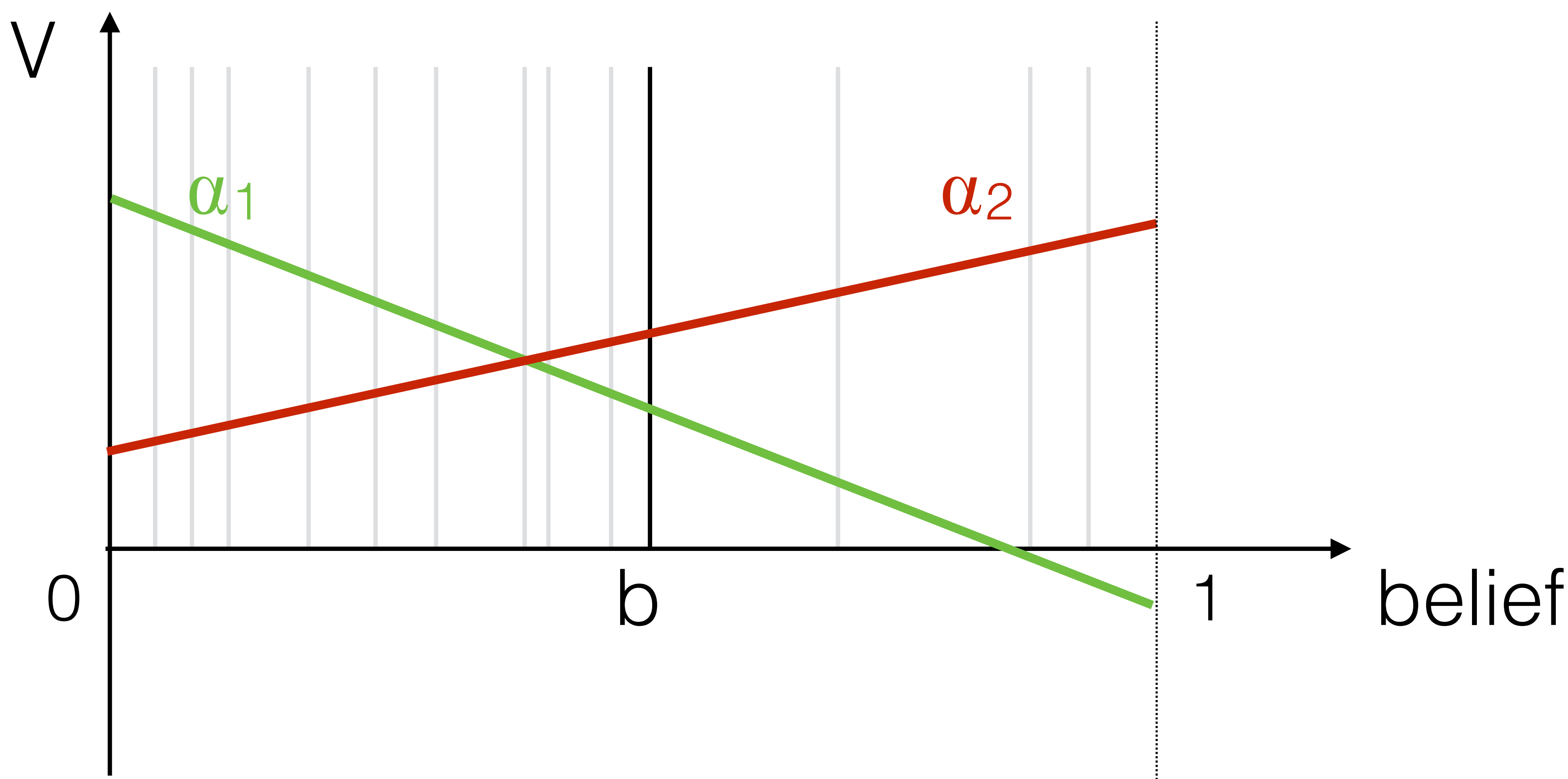
## Point-Based Methods (PERSEUS)

1. Find a set of belief points



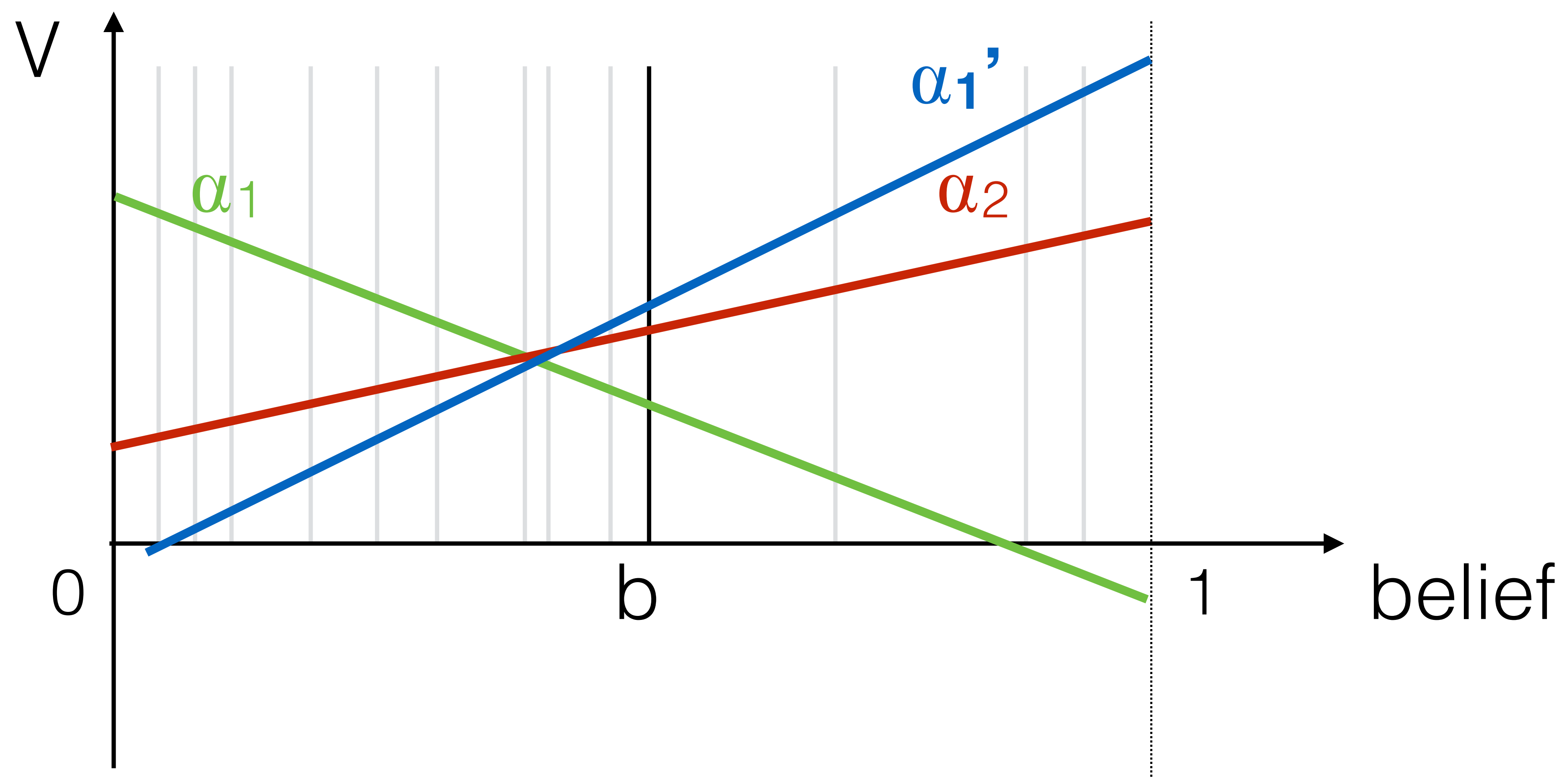
## Point-Based Methods (PERSEUS)

2. From that set, pick a point randomly



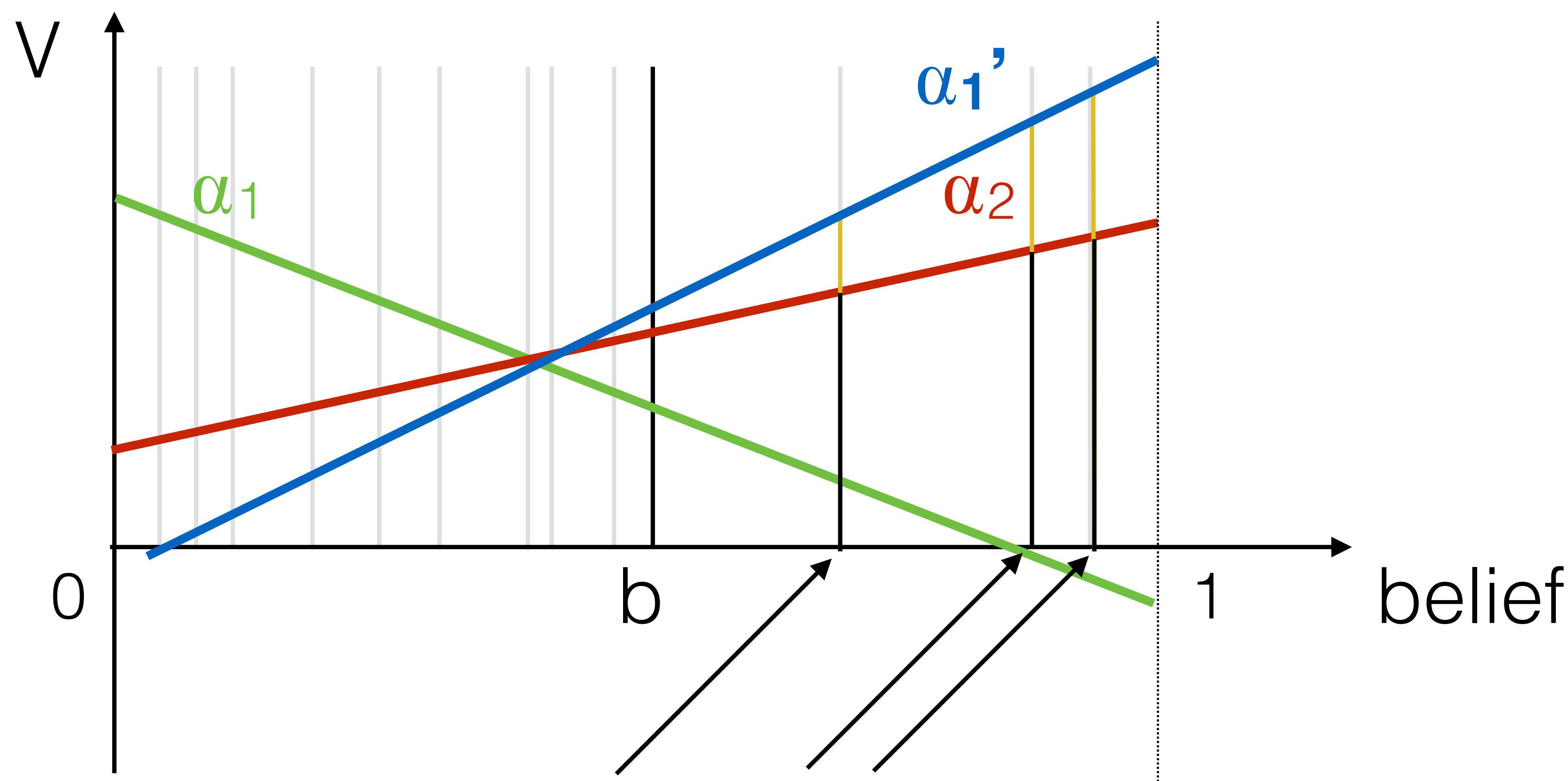
## Point-Based Methods (PERSEUS)

3. Find the best vector for that point (for horizon n+1)



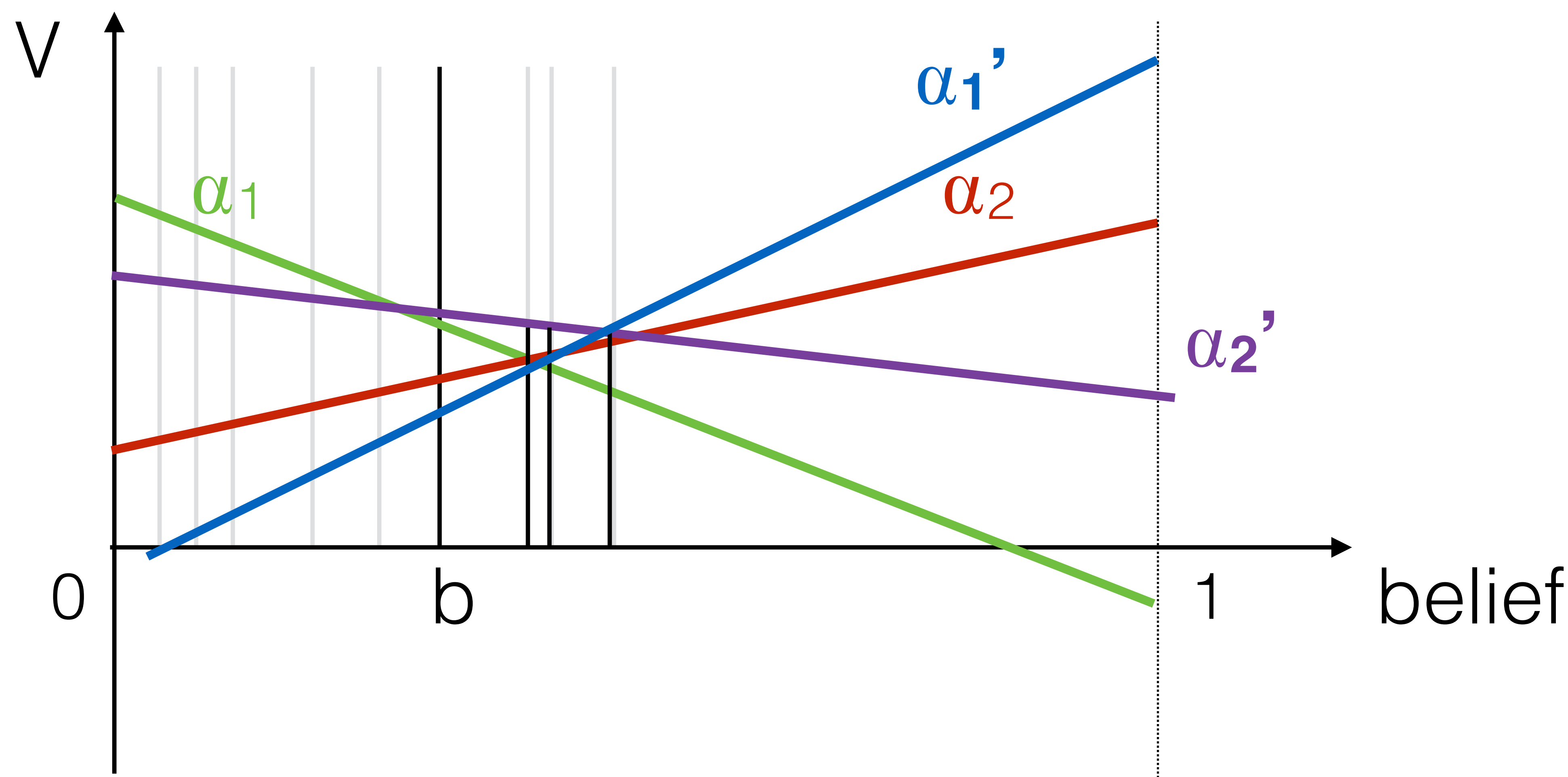
## Point-Based Methods (PERSEUS)

4. If that vector improves the value at a belief point, remove it.



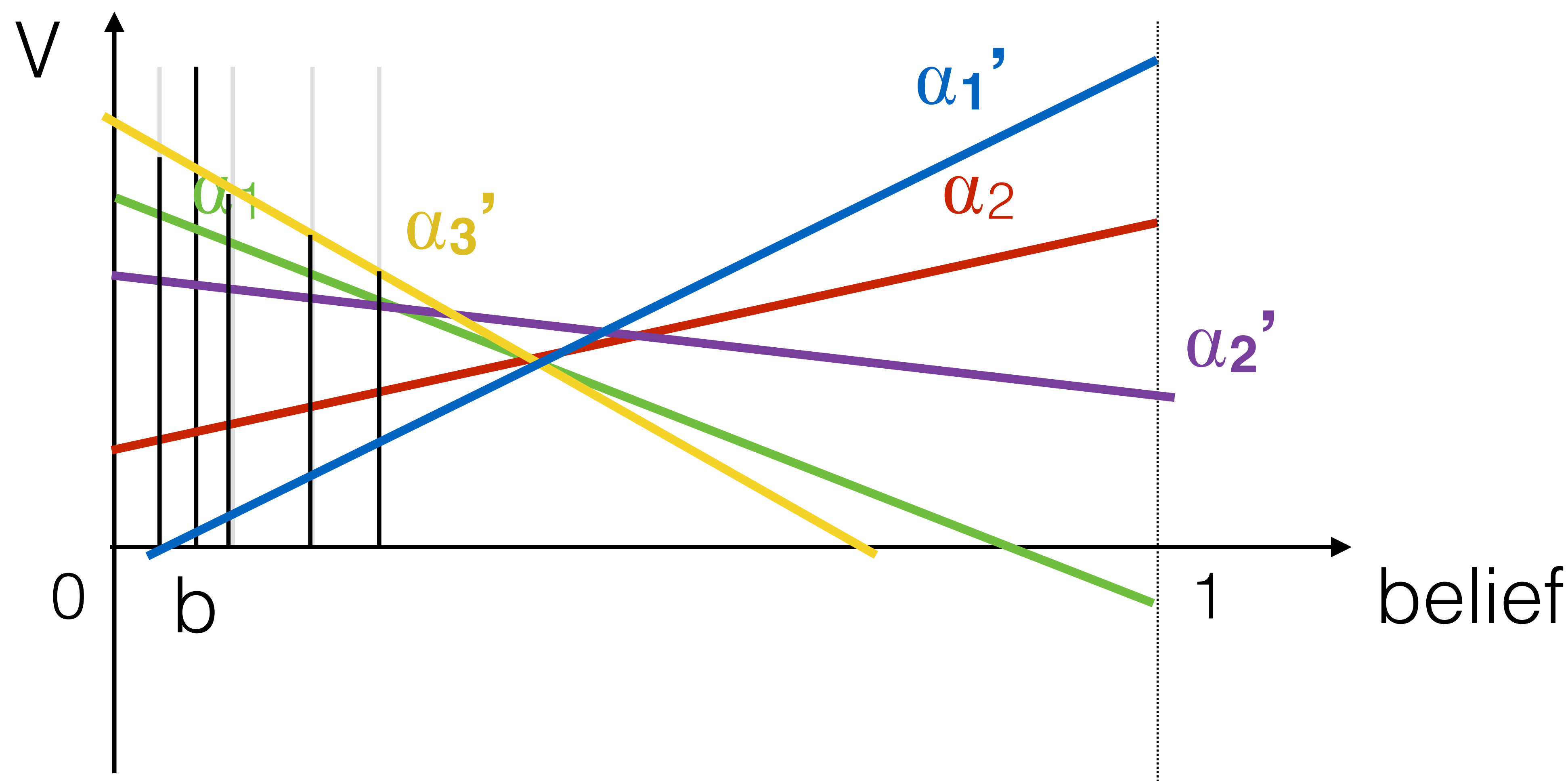
## Point-Based Methods (PERSEUS)

5. Repeat until there are no more points left.



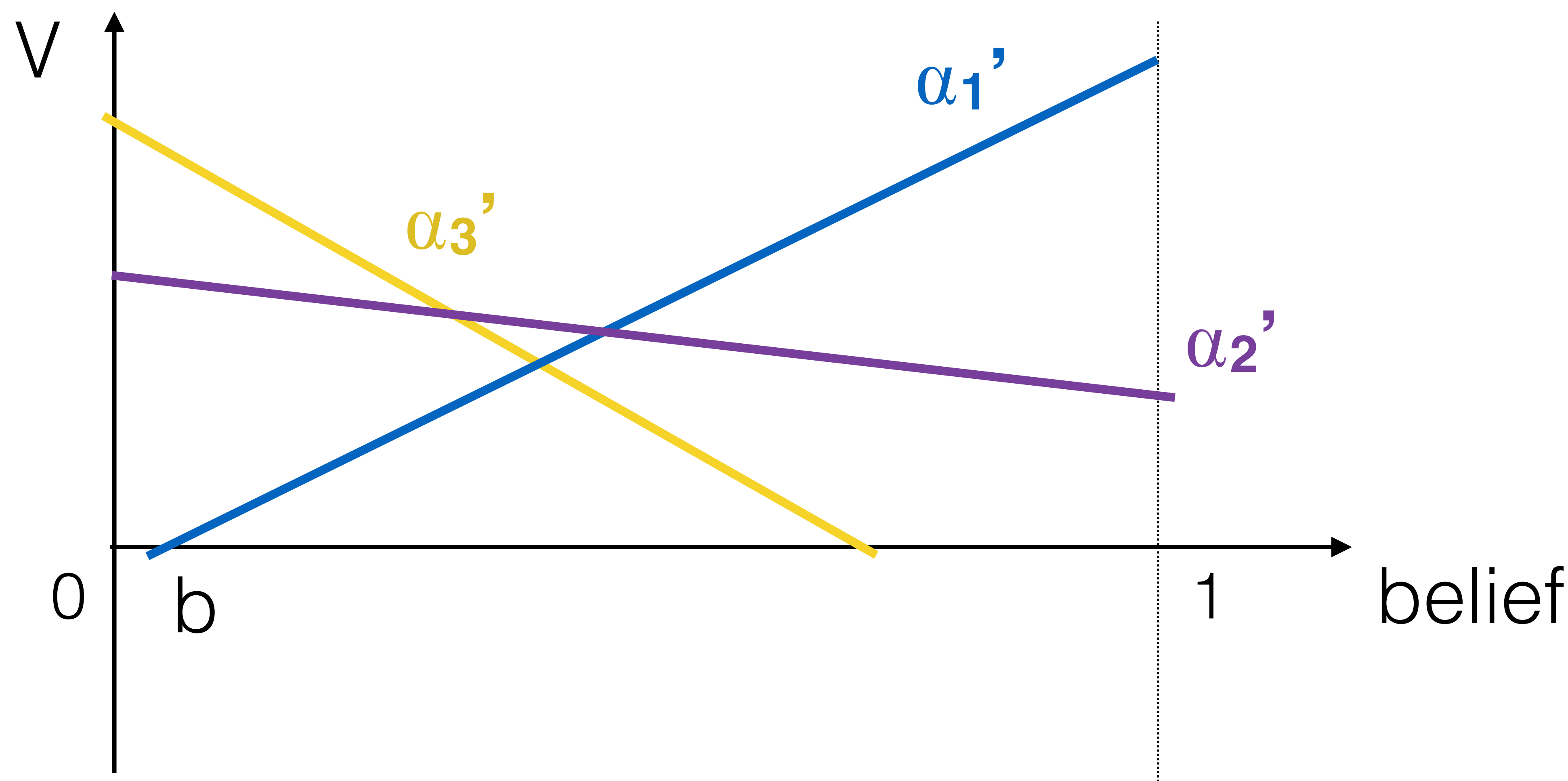
## Point-Based Methods (PERSEUS)

5. Repeat until there are no more points left.



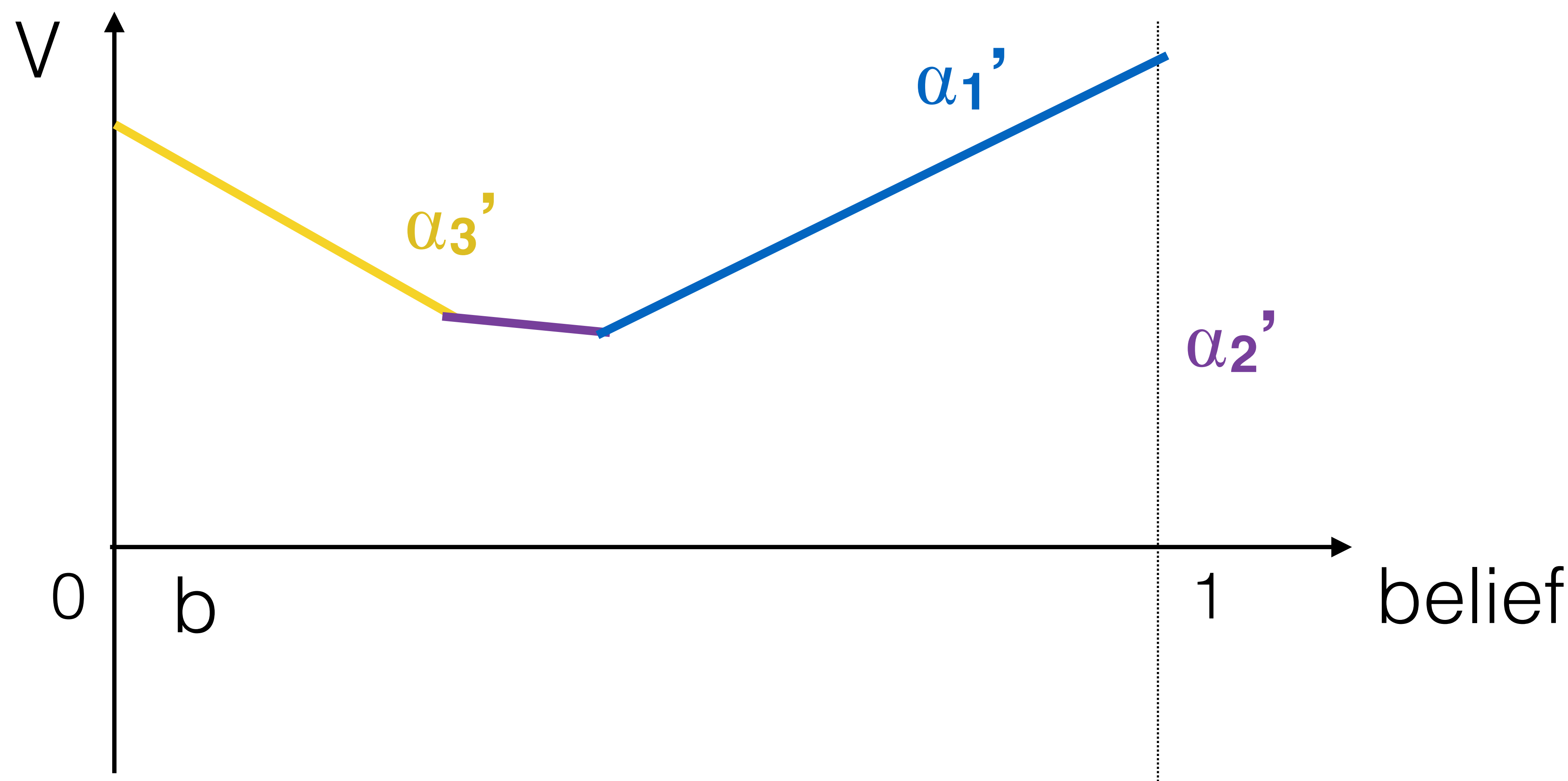
## Point-Based Methods (PERSEUS)

Approximate value function for horizon n+1:



## Point-Based Methods (PERSEUS)

Remember that we only care about the maximum!



In some cases, it is easier to define states, actions, and observations as combinations (tuples) of variables.

**Example:**

State "Factors"

$\mathcal{S} =$ {battery High, battery Low} x {room 1, room 2, …., room N} $= \mathcal{X}_1 \times \mathcal{X}_2$

$$\mathbf{s} = \langle x_1, x_2 \rangle$$

$\mathcal{A} =$ {up, down, left, right} x {move slow, normal speed, move fast} $= \mathcal{A}_{dir} \times \mathcal{A}_{speed}$

$$\mathbf{a} = \langle a_{dir}, a_{speed} \rangle$$

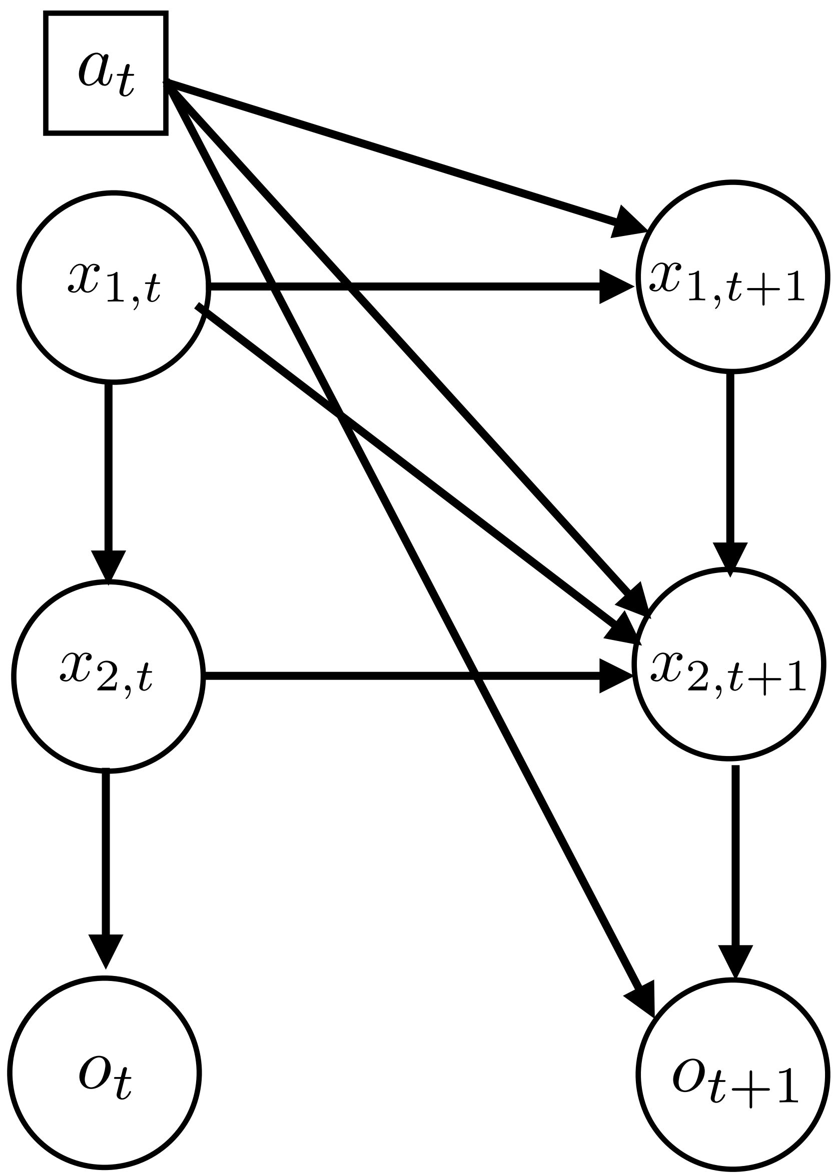
Such models are said to be **factored**.

All factored models have an equivalent "flat" representation.

But they can expose the structure of the decision-making problem, making it easier to solve.

# Factored Models

Factored models can be represented as **Dynamic Bayesian Networks** (DBNs)




Arrows represent conditional dependence

Each variable at time *t+1* has a Conditional Probability Distribution (CPD)

Can be a table (CPT) or a decision diagram