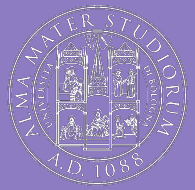


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Intelligenza Artificiale

Intelligent Agents
Part II

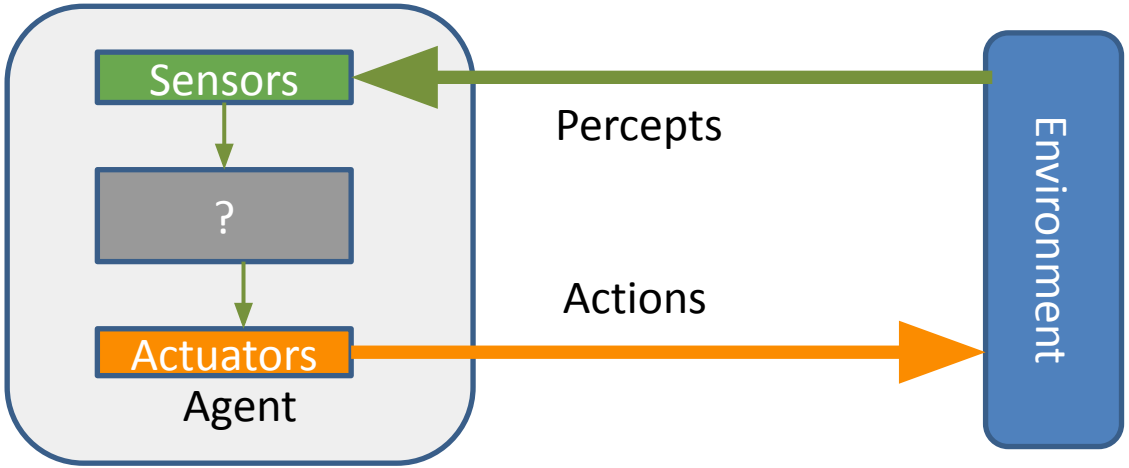
Ivan Heibi
Dipartimento di Filologia Classica e Italianistica (FICLIT)
Ivan.heibi2@unibo.it

What is an agent? (recap)

An **agent** is anything that can be viewed as:

- **perceiving** its environment through sensors
- **acting** upon that environment through actuators

An agent's choice of action at any given instant can depend on the entire percept sequence observed to date.



Agent Function (recap)

Mathematically speaking, an agent behaviour can be described by a function that maps any given percept sequence to an action.

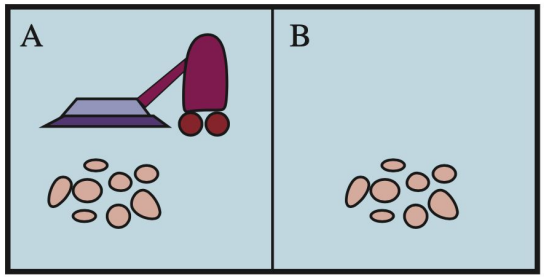
We can imagine this function as a table with (possibly infinite) number of rows

Example: Vacuum agent

Environment: *Square A*, *Square B*

Perceptions: *Square Dirty*, *Square Clean*

Actions: Move left, Move right, Draw up the dirt



Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Draw up
[B, Clean]	Left
[B, Dirty]	Draw up
[A, Clean][A, Clean]	Right
[A, Clean][A, Dirty]	Draw up
....	
[A, Clean][A, Clean] [A, Clean]	Right
[A, Clean][A, Clean] [A, Dirty]	Draw up
...	

PEAS description (recap)

The **PEAS description** is an acronym for the elements needed for specifying the problem:

- **Performance measure**
  It defines what the agent aspires.
- **Environment**
  It is where the agent operates.
- **Actuators**
  The “devices” that allow the agent to operate
- **Sensors**
  The “devices” that allow the agent to percept

Agent type	Performance measure	Environment	Actuators	Sensors
Automated Taxi Driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, pedestrian, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS

Dimensions (recap)

Dimension	*Chess w clock*	*Poker*	*Object recognition*	*Taxi*
Fully / Partially Observable	Fully	Partially	Fully	Partially
Single / Multi Agent	Multi	Multi	Single	Multi
Deterministic / Stochastic	Deterministic	Stochastic	Deterministic	Stochastic
Episodic / Sequential	Sequential	Sequential	Episodic	Sequential
Static / Dynamic	Semi-Dynamic	Static	Static	Dynamic
Discrete / Continuous	Discrete	Discrete	Continuous	Continuous

The real world is partially observable, multiagent, stochastic, sequential, dynamic, continuous, unknown

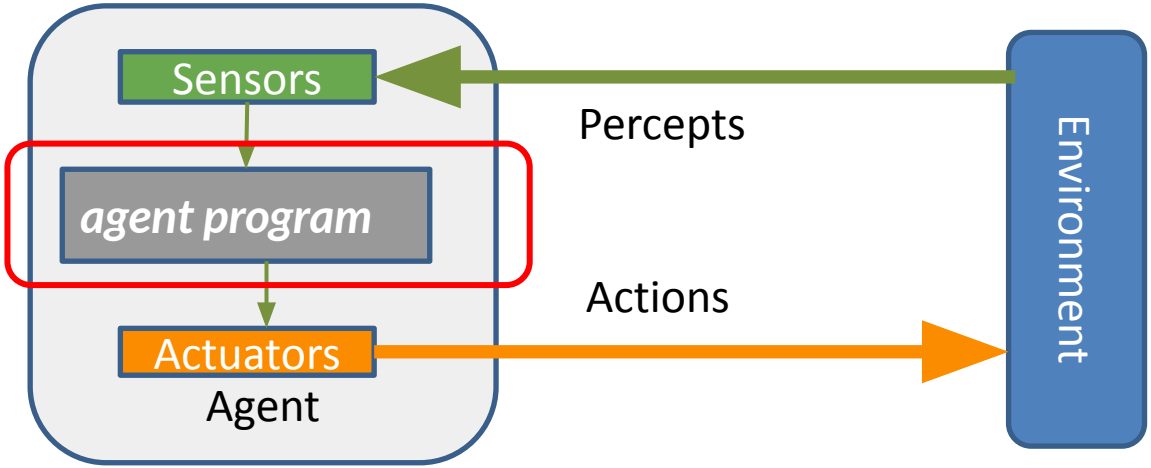
Agent programs

The job of AI is to design an agent program that implements the agent function

We assume this program will run on some sort of computing device with physical sensors and actuators (we call this **architecture**).

```
agent = architecture + program
```

Note: agent program is not the agent function.



Exemplary agent program (pseudocode)

```
1  function TABLE-DRIVEN-AGENT(percept) returns an action

2      persistent:
           percepts, a sequence, initially empty,
           table, a table of actions, indexed by
                  percept sequences, initially fully specified

3      append percept to the end of percepts

4      action←LOOKUP(percepts,table)

5      return action
```

Exemplary agent program (pseudocode)

```
1  function TABLE-DRIVEN-AGENT(percept) returns an action

2      persistent:
           percepts, a sequence, initially empty,
           table, a table of actions, indexed by
                  percept sequences, initially fully specified

3      append percept to the end of percepts

4      action←LOOKUP(percepts,table)

5      return action
```

Exemplary agent program (pseudocode)

```
1  function TABLE-DRIVEN-AGENT(percept) returns an action

2      persistent:
           percepts, a sequence, initially empty,
           table, a table of actions, indexed by
                  percept sequences, initially fully specified

3      append percept to the end of percepts

4      action←LOOKUP(percepts,table)

5      return action
```

Exemplary agent program (pseudocode)

```
1  function TABLE-DRIVEN-AGENT(percept) returns an action

2      persistent:
           percepts, a sequence, initially empty,
           table, a table of actions, indexed by
                  percept sequences, initially fully specified

3      append percept to the end of percepts

4      action←LOOKUP(percepts,table)

5      return action
```

Exemplary agent program (pseudocode)

```
1  function TABLE-DRIVEN-AGENT(percept) returns an action

2      persistent:
           percepts, a sequence, initially empty,
           table, a table of actions, indexed by
                  percept sequences, initially fully specified

3      append percept to the end of percepts

4      action←LOOKUP(percepts,table)

5      return action
```

Exemplary agent program (pseudocode)

```
1  function TABLE-DRIVEN-AGENT(percept) returns an action

2      persistent:
           percepts, a sequence, initially empty,
           table, a table of actions, indexed by
                  percept sequences, initially fully specified

3      append percept to the end of percepts

4      action←LOOKUP(percepts,table)

5      return action
```

Exemplary agent program – example

percepts = [A,clean]

table =

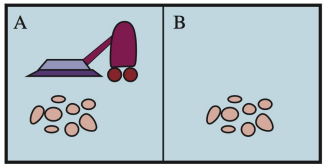
Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Draw up
[B, Clean]	Left
[B, Dirty]	Draw up
[A, Clean] [A, Clean]	Right
[A, Clean] [A, Dirty]	Draw up
...	...

Example: Vacuum agent

Environment: *Square A*, *Square B*

Perceptions: *Square Dirty*, *Square Clean*

Actions: Move left, Move right, Draw up the dirt



Exemplary agent program – example

percept = [A,dirty]

`TABLE-DRIVEN-AGENT (percept)`

percepts = [A,clean]

table =

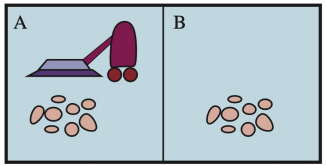
Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Draw up
[B, Clean]	Left
[B, Dirty]	Draw up
[A, Clean] [A, Clean]	Right
[A, Clean] [A, Dirty]	Draw up
...	...

Example: Vacuum agent

Environment: *Square A*, *Square B*

Perceptions: *Square Dirty*, *Square Clean*

Actions: Move left, Move right, Draw up the dirt



Exemplary agent program – example

percept = [A,dirty]

`TABLE-DRIVEN-AGENT (percept)`

append percept to the end of percepts

percepts = [A,clean] [A,dirty]

table =

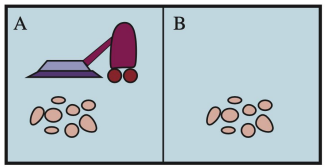
Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Draw up
[B, Clean]	Left
[B, Dirty]	Draw up
[A, Clean] [A, Clean]	Right
[A, Clean] [A, Dirty]	Draw up
...	...

Example: Vacuum agent

Environment: *Square A*, *Square B*

Perceptions: *Square Dirty*, *Square Clean*

Actions: Move left, Move right, Draw up the dirt



Exemplary agent program – example

percept = [A,dirty]

↓

TABLE-DRIVEN-AGENT (*percept*)

↓

append percept to the end of percepts

↓

action←LOOKUP(*percepts*,*table*)

percepts = [A,clean] [A,dirty]

table =

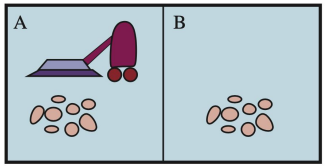
Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Draw up
[B, Clean]	Left
[B, Dirty]	Draw up
[A, Clean] [A, Clean]	Right
[A, Clean] [A, Dirty]	Draw up
...	...

Example: Vacuum agent

Environment: *Square A*, *Square B*

Perceptions: *Square Dirty*, *Square Clean*

Actions: Move left, Move right, Draw up the dirt



Types of agent programs

The key challenge for AI is to find out how to write programs that, to the extent possible, produce rational behavior from a smallish program rather than from a vast table.

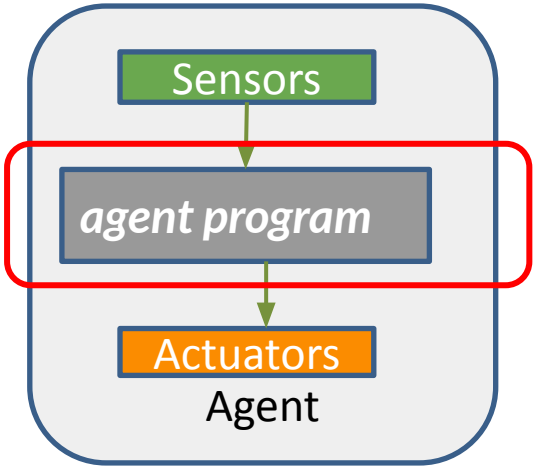
In the remainder of this presentation we will see ***four basic kinds of agent programs*** *that embody the principles underlying almost all intelligent systems:*

Simple reflex agents;

Model-based reflex agents;

Goal-based agents;

Utility-based agents.



Simple reflex agents

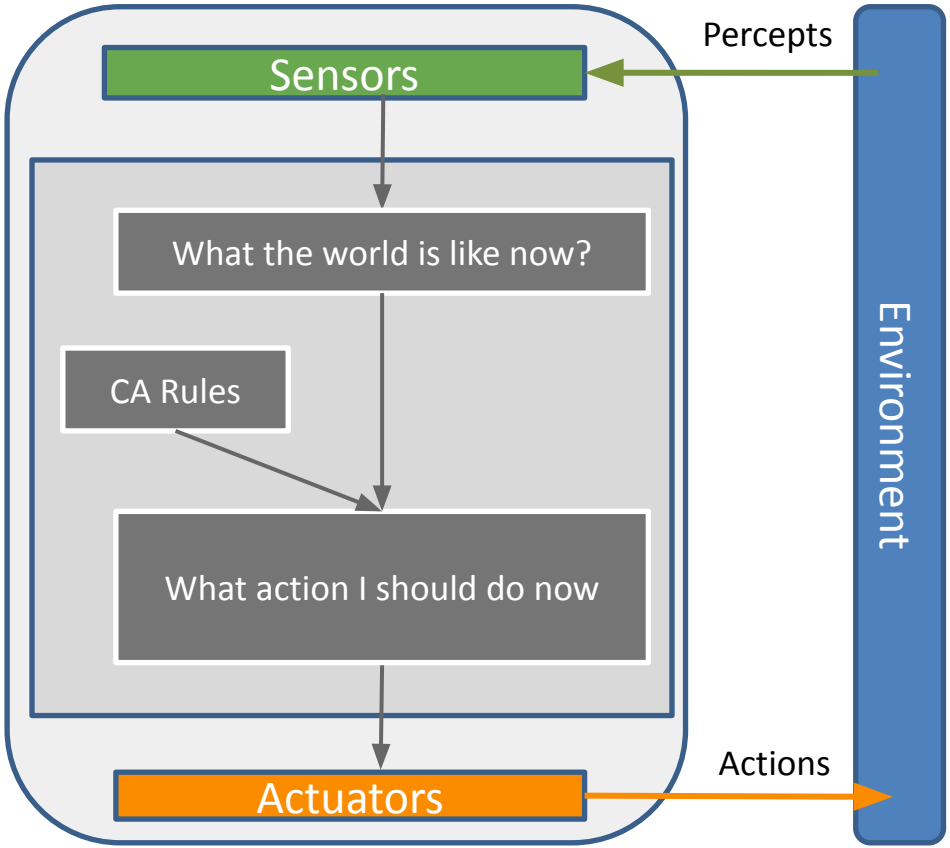
The simplest kind of agent is the **simple reflex agent**

These agents select actions on the basis of the **current percept, ignoring the rest of the percept history.**

Their behaviour can be described by condition-action (CA) rules:
`If <percept> then <action>`

Example: `If car-in-front-is-braking then initiate-braking`

Simple reflex agent are designed to work when the environment if fully observable



Simple reflex agent – Vacuum agent example

```
1   function REFLEX-VACUUM-AGENT([location,status]) returns an action

2       if status = Dirty then action←Draw_up
3       else if location = A then action←Right
4       else if location = B then action←Left

5       return action
```

The most obvious reduction comes from **ignoring the percept history**, which cuts down the number of relevant percept sequences

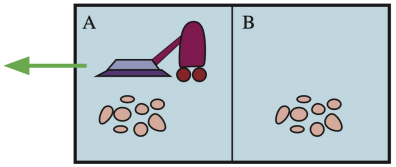
Simple reflex agent – Infinite loops

Suppose that a **simple reflex vacuum agent** is deprived of its location sensor and has only a dirt sensor.

Its behaviour could be described with the following **CA rules**:

> *If Dirt then draw up.*

> *If Clean then move left.*



If the agent is on square A, **it fails forever!**

To avoid from infinite loop the agent can **randomize its actions**.

For example, if the vacuum perceives the square as clean, it might flip a coin to choose between Left and Right

In single-agent environments, randomization is usually not rational
> we can do better with more sophisticated agents

Model-based reflex agents

The most effective way to handle partial observability is for the **agent to keep track of the part of the world it can't see now**.

The agent should maintain some sort of **internal state that depends on the percept history** and thereby reflects at least some of the unobserved aspects of the current state.

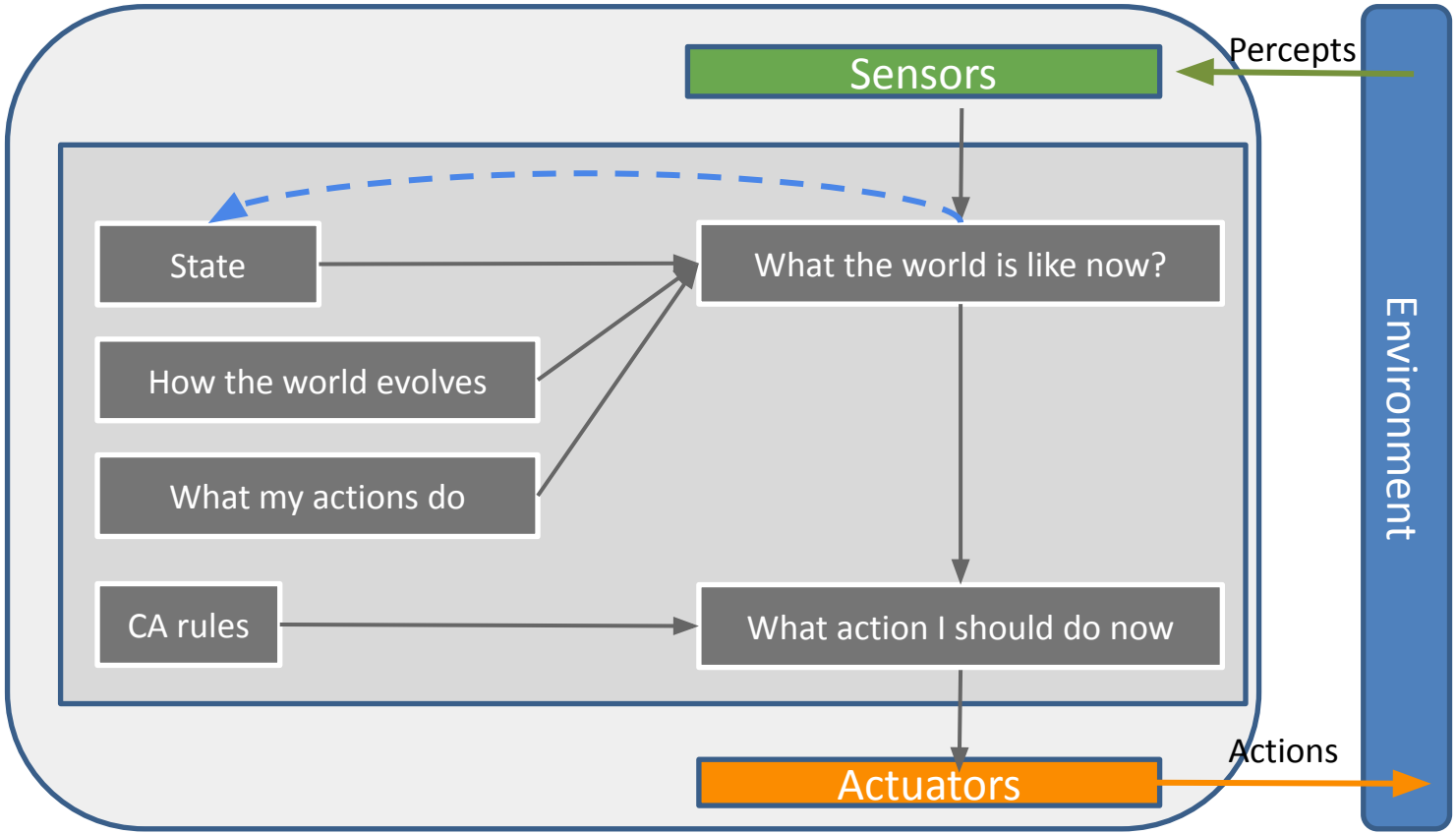
To update this internal state information the agent needs information about how:
> the world evolves independently of his actions;
> his own actions affect the world.

This knowledge about "how the world works" is called **model of the world**.

An agent that uses such a model is called a model-based agent.

The details of how models and states are represented vary widely depending on the environment and the technology used in the agent design.

Model-based reflex agents



Model-based reflex agents – pseudocode

```
1  function MODEL-BASED-REFLEX-AGENT (percept) returns an action

2      persistent:
          state, agent's current conception of the world state
          transition model, a description of how the next state
                depends on the current state and action
          sensor model, a description of how the current world
                state is reflected in the agent's percepts
          rules, a set of condition-action rules
          action, the most recent action


3      state←UPDATE-STATE(state,action,percept,transition model,sensor model)
4      rule←RULE-MATCH(state,rules)
5      action←rule.ACTION

6      return action
```

Goal-based agents

Knowing the **state of the environment is not always enough** to decide what to do (e.g. taxi driver)

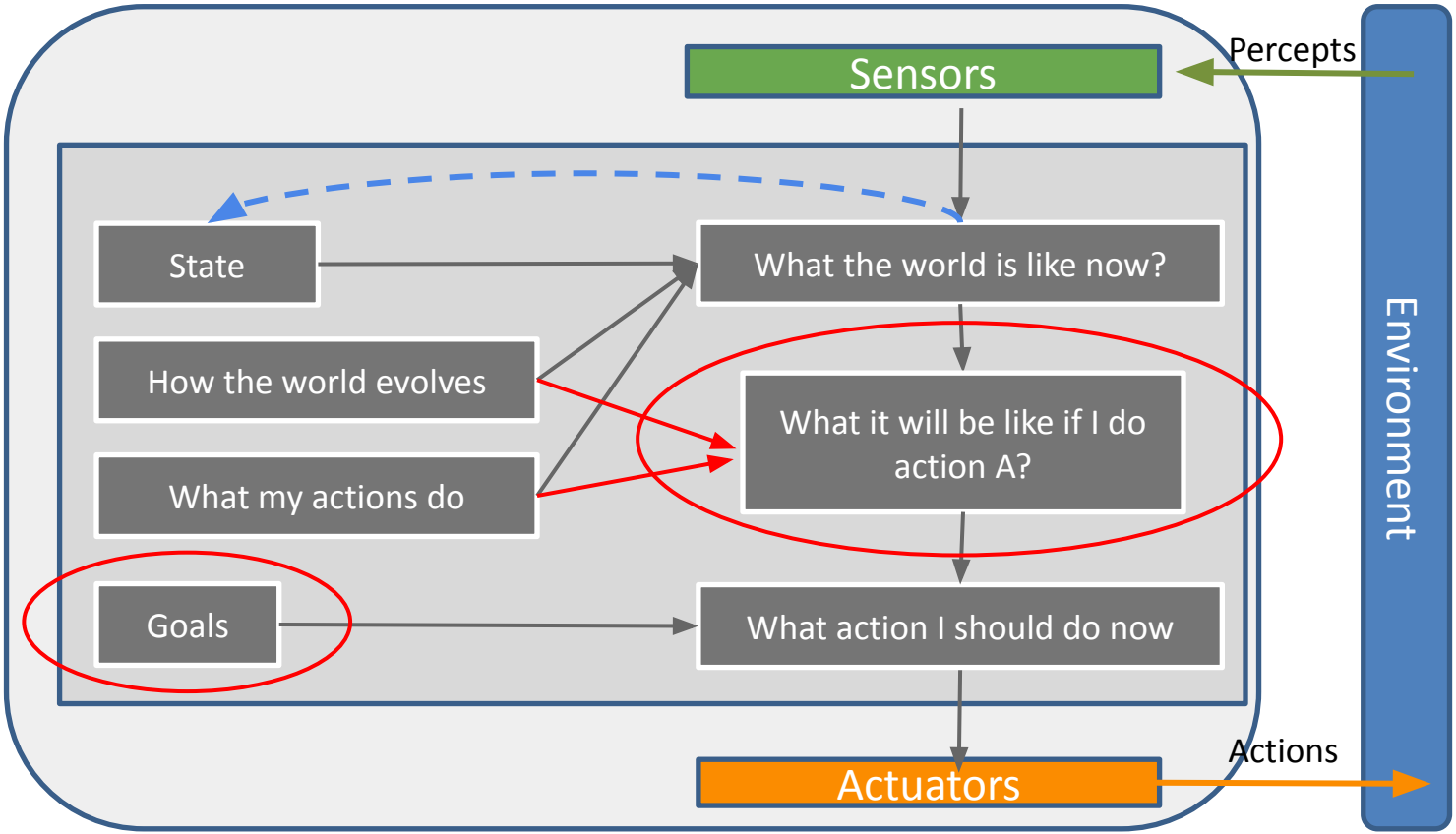
➔ agents needs some sort of **goal information that describes desirable situations**

Sometimes the goal selection is straightforward, sometimes is more tricky!

Search and Planning are the subfields of AI devoted to finding action sequences that achieve the agent's goal.

In other cases, additional reasoning capabilities are needed to decide among different **(possibly conflicting) goals.** For example, consider the taxi driver example, maximize the speed and the safety conflict!

Goal-based agents



Utility-based agents

Goals alone are not enough to generate high-quality behaviour in most environments.

A more general performance measure should allow a comparison of different world states according to exactly how “happy” they would make the agent.

”Happy” doesn’t sound very scientific, it would be better use the term **utility**.

An utility function is essentially an internalization of the performance measure.

➔ If the internal utility function and the external performance measure (based on the objectives) are in agreement, then an agent that chooses actions to maximize its utility will be rational to its aim.

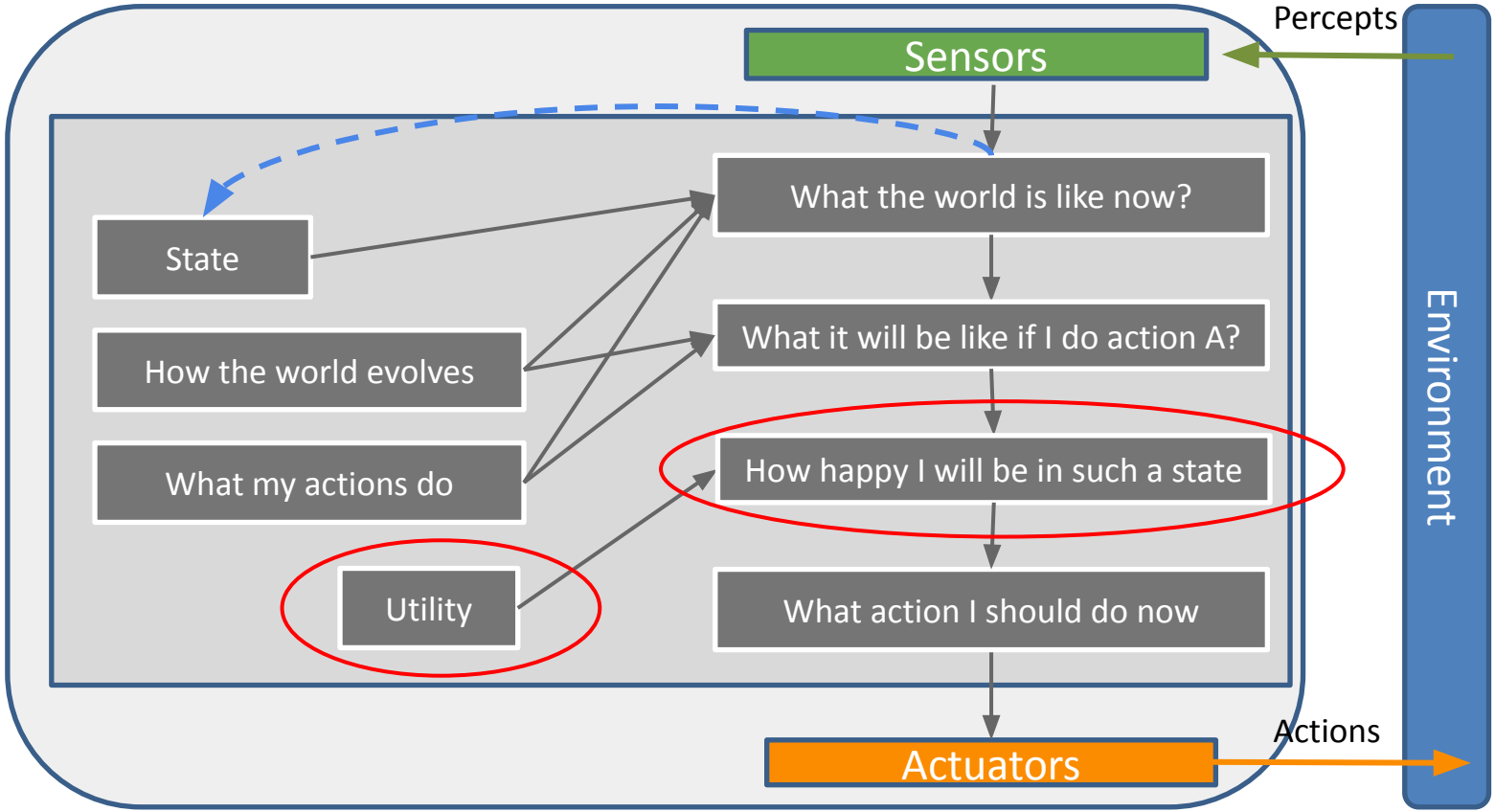
Utility-based agents

In two kinds of cases, goal-based agents are inadequate but a utility-based agent can still make rational decisions:

- When there are **conflicting goals**, only some of which can be achieved, the utility function specifies the tradeoff (for example, safety and speed)
- When there are **several goals** that the agent can aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighted against the importance of the goal.

Partial observability and stochasticity are frequent in the real world, and so, therefore, is decision making under uncertainty.
> A rational utility-based agent chooses the action that maximizes the **expected utility** of the action outcomes.

Utility-based agents



Overview

Simple Reflex Agents	Model-Based Reflex Agents
Characteristics: * Make decisions based solely on the current percept. * Do not have internal state representations. * Use pre-defined rules or condition-action pairs to respond to stimuli.	**Characteristics:** * Maintain an internal model or representation of the world. * Use the internal model to make decisions based on both current and past percepts. * Can handle partially observable environments more effectively than simple reflex agents.
Limitations: - Limited intelligence in dynamic or partially observable environments.	**Limitations:** - May face challenges in environments with high uncertainty

Goal-Based Agents	Utility-Based Agents
Characteristics: * Have explicit goals that guide decision-making. * Evaluate actions based on their contribution to achieving goals. * Consider sequences of actions to accomplish objectives.	**Characteristics:** * Evaluate actions based on a utility function, capturing preferences. * Seek to maximize the expected utility of outcomes. * Provide a more flexible and nuanced approach to decision-making
Limitations: - Need well-defined goals and may struggle in highly dynamic or uncertain environments (conflict and multiple goals)	**Limitations:** - Require a well-defined utility function and can be computationally intensive

Overview

The choice of an agent program depends on the complexity and nature of the problem at hand. The criticality of each type depends on the specific characteristics of the task or environment an agent is designed to handle.

This decision should consider the environment and check how the agent model works in terms of:

- **Adaptability:** The ability to adapt to changes in the environment or goals.
- **Computational Complexity:** The computational requirements for decision-making.
- **Knowledge Representation:** How knowledge about the world is represented and utilized.
- **Handling Uncertainty:** The ability to make decisions in the presence of incomplete or uncertain information.

Learning agent

Explicitly designing (a-priori) world models, actions' causality, events, condition action rules **constitutes a significant bottleneck!**

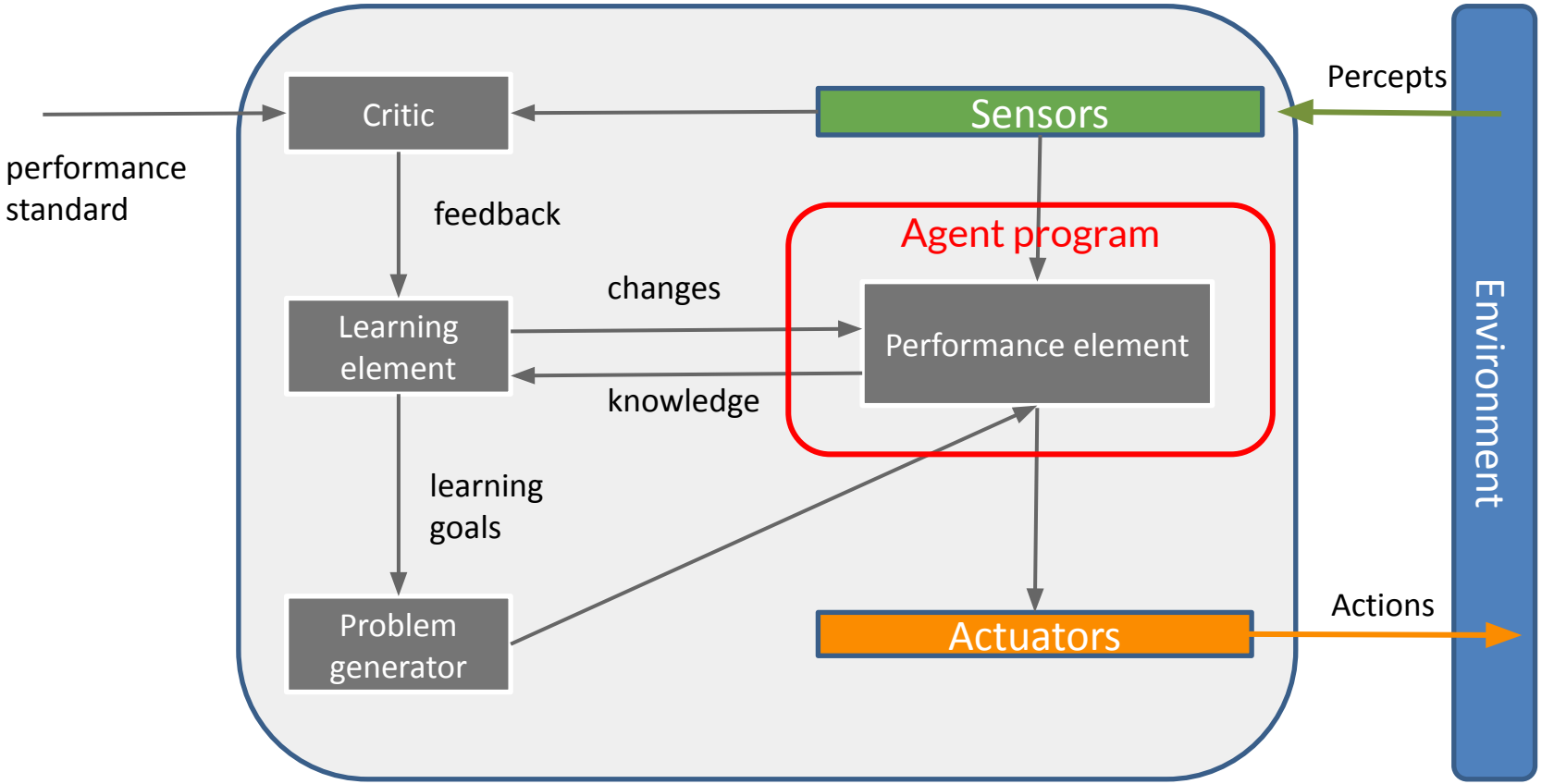
An agent able to **learn** would require less a-priori knowledge thus streamlining the design process.

Learning has another advantage, it allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow.

A learning agent can be composed by four conceptual components:

- **Learning element** (which is responsible for making improvements)
- **Performance element** (which is responsible for selecting actions)
- **Critic** (which provides feedback on how the agent is doing and determines how the performance element should be modified)
- **Problem generator** (which is responsible for suggesting actions that lead to new and informative experiences.

Learning agent



How the components of agent programs actually work?

The whole course will be devoted to begin to answer this question properly.

We can draw some basic distinctions among the various ways that the components can represent the environments that agent inhabits.

We can place the representations along **an axis of increasing complexity and expressive power**:

- Atomic
- Factored
- Structured

Let's focus on the What my actions do component.

Atomic representation

States of the environment are represented as a **single black box with no internal structure**.

Transitions across states are represent as relations (visualized as edges).

In Atomic representation each state of the world is indivisible.

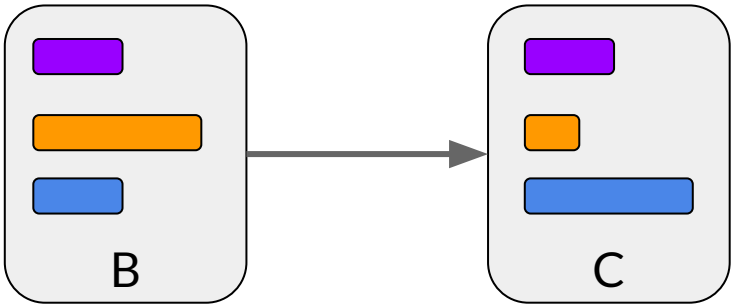
Example: reaching a destination => each state represent my current city.
No attributes are needed from states B and C, only thing needed is to the ability to stay if B equal C or not

Factored representation

In factored representation each state is split up into a fixed set of attributes, each of which can have a value.

Different states may share attributes.

Factored representations also allow to represent uncertainty (e.g. attribute without value).



Example: reaching a destination => possible variables: level of gas, GPS location etc.

Structured representation

For many purposes, we need to understand the world as having things in it that are related to each other (not just values).