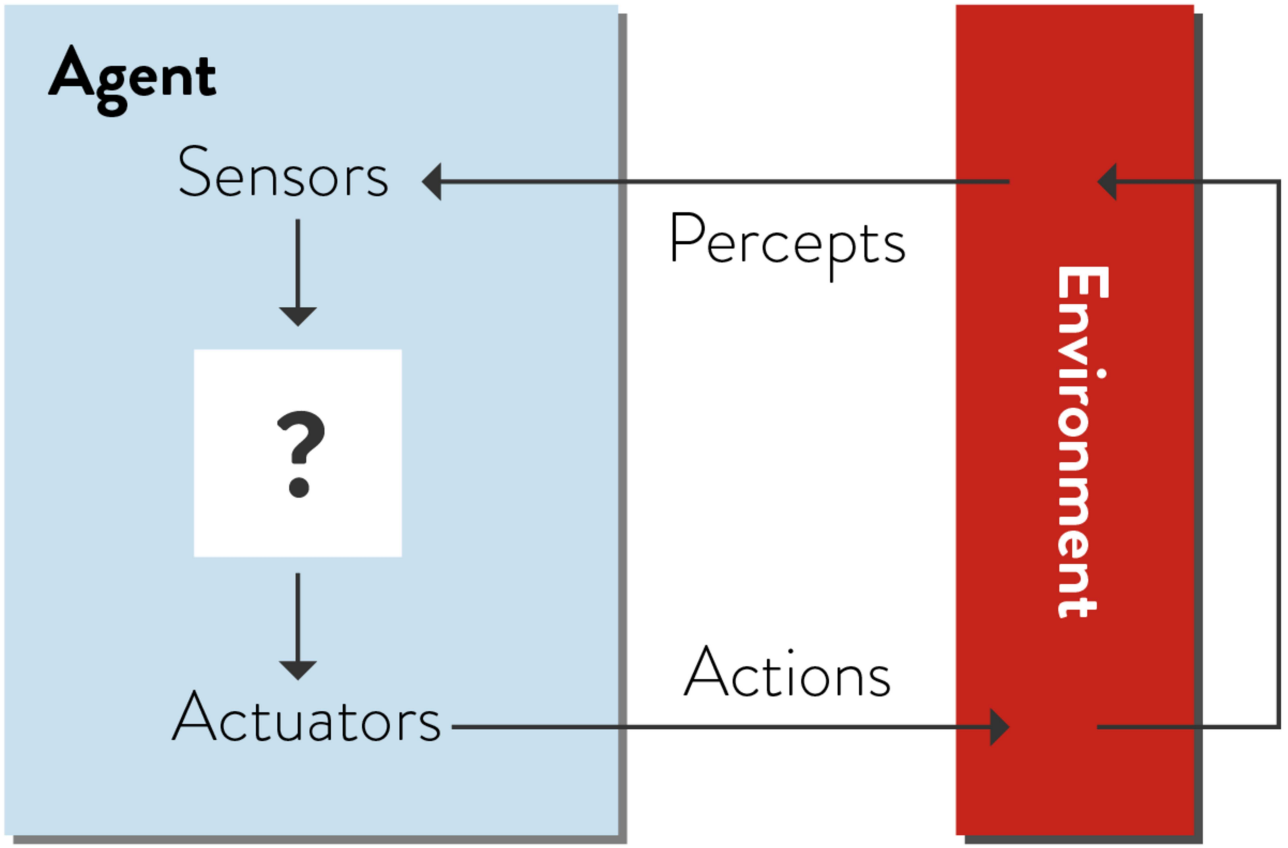
COMP3411/9814: Artificial Intelligence

Intelligent Agents – Agent Types

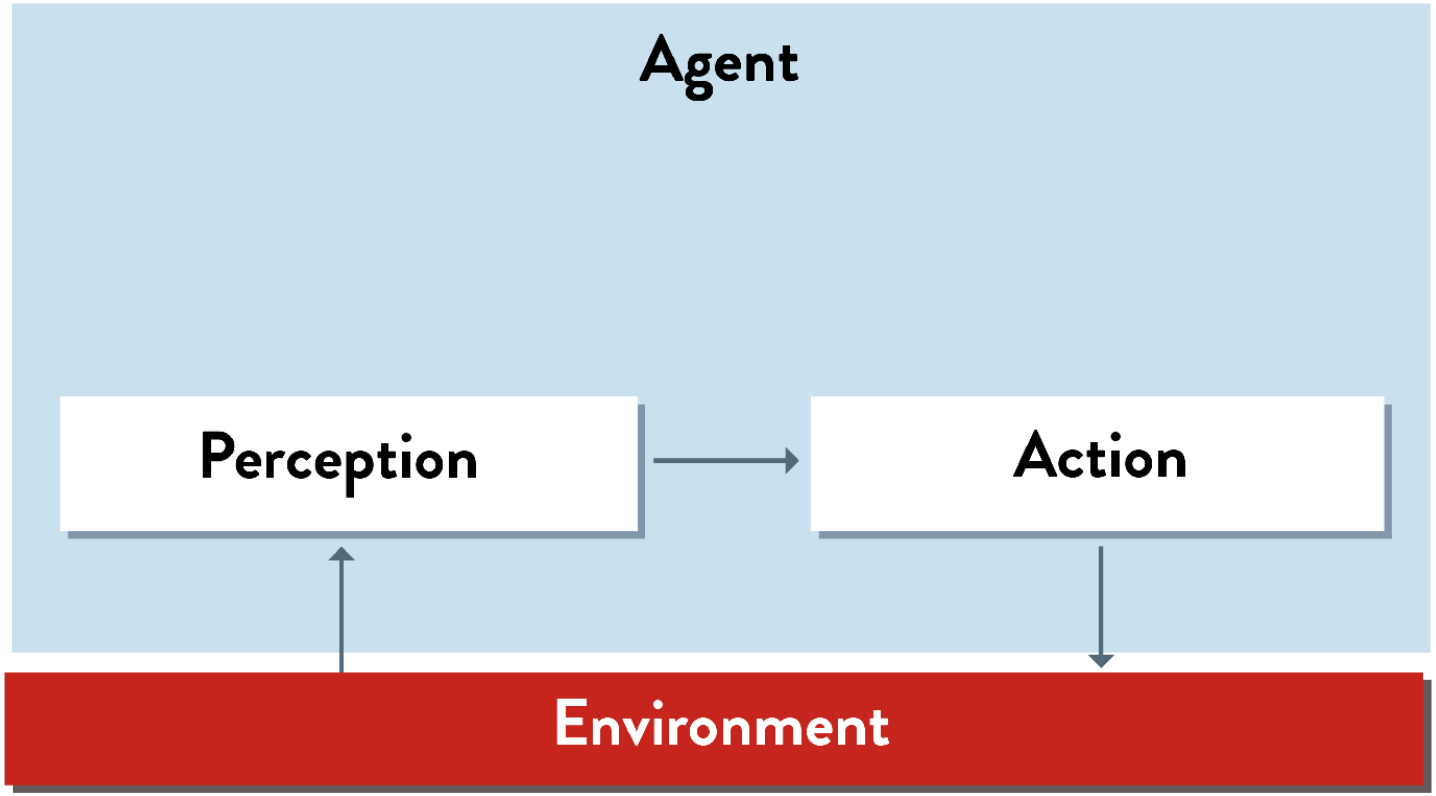
Lecture Overview

- Agent types
  - Reactive Agent
  - Model-Based Agent
  - Planning Agent
  - Utility-based agent
  - Game Playing Agent
  - Learning Agent
  - Agent Programs and Architectures

Agent Model



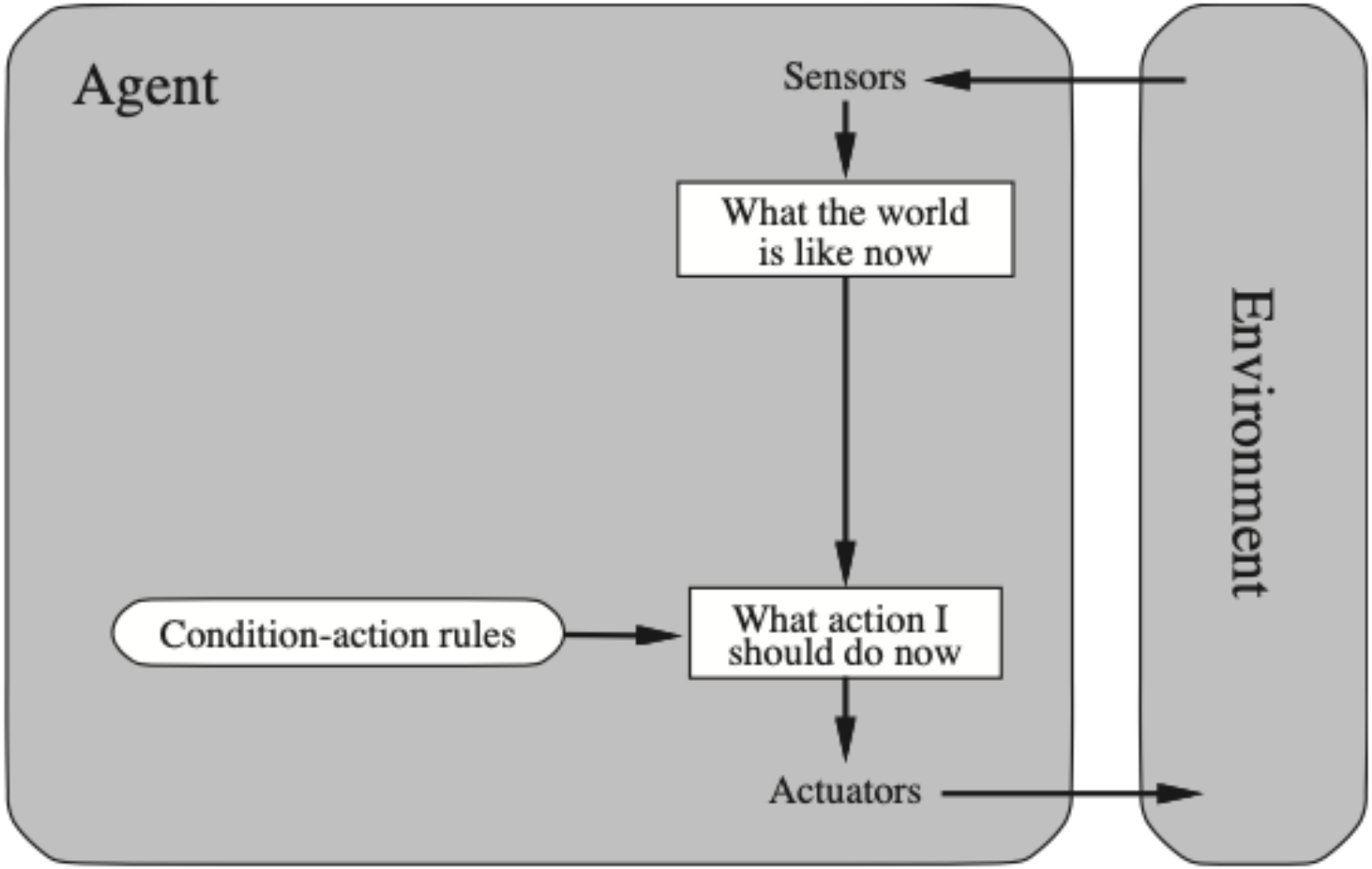
Reactive Agent



Reactive Agent

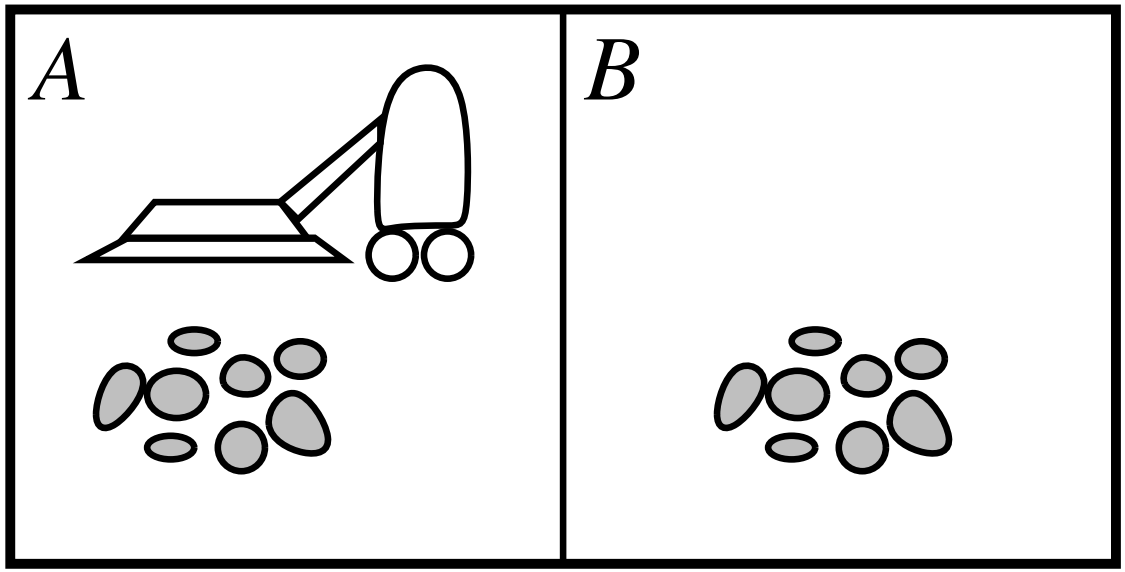
- Choose the next action based only on what they currently perceive, using a "policy" or set of rules which are simple to apply
- Sometimes called "simple reflex agents" – but they can do surprisingly sophisticated things!
  - Swiss robots (https://www.youtube.com/watch?v=B0wM-eKSxhk)
  - Pool balancing
  - Simulated hockey

Reflex (Reactive) Agent



Reflex (reactive) agent — applies condition-action rules to each percept

Vacuum-cleaner world



Percepts: location and contents, e.g., [*A, Dirty*]

Actions: *Left, Right, Suck, NoOp*

A vacuum-cleaner -simple reflex agent

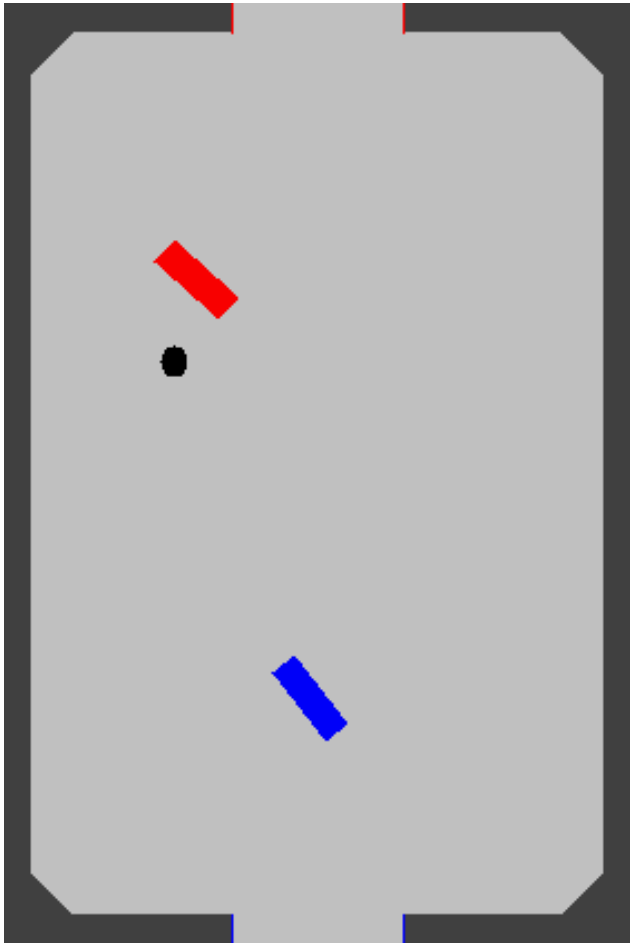
Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
⋮	⋮

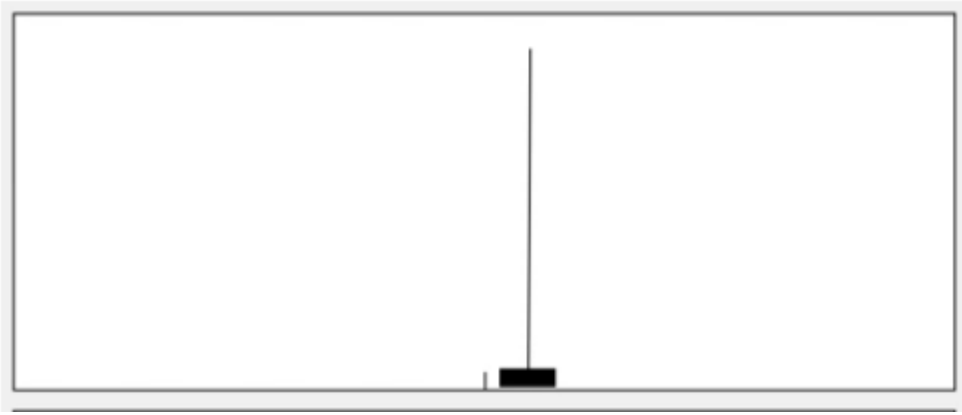
function REFLEX-VACUUM-AGENT([*location*,*status*]) **returns** an action

if $status = Dirty$ **then return** $Suck$
else if $location = A$ **then return** $Right$
else if $location = B$ **then return** $Left$

condition-action rules

Reactive Agent





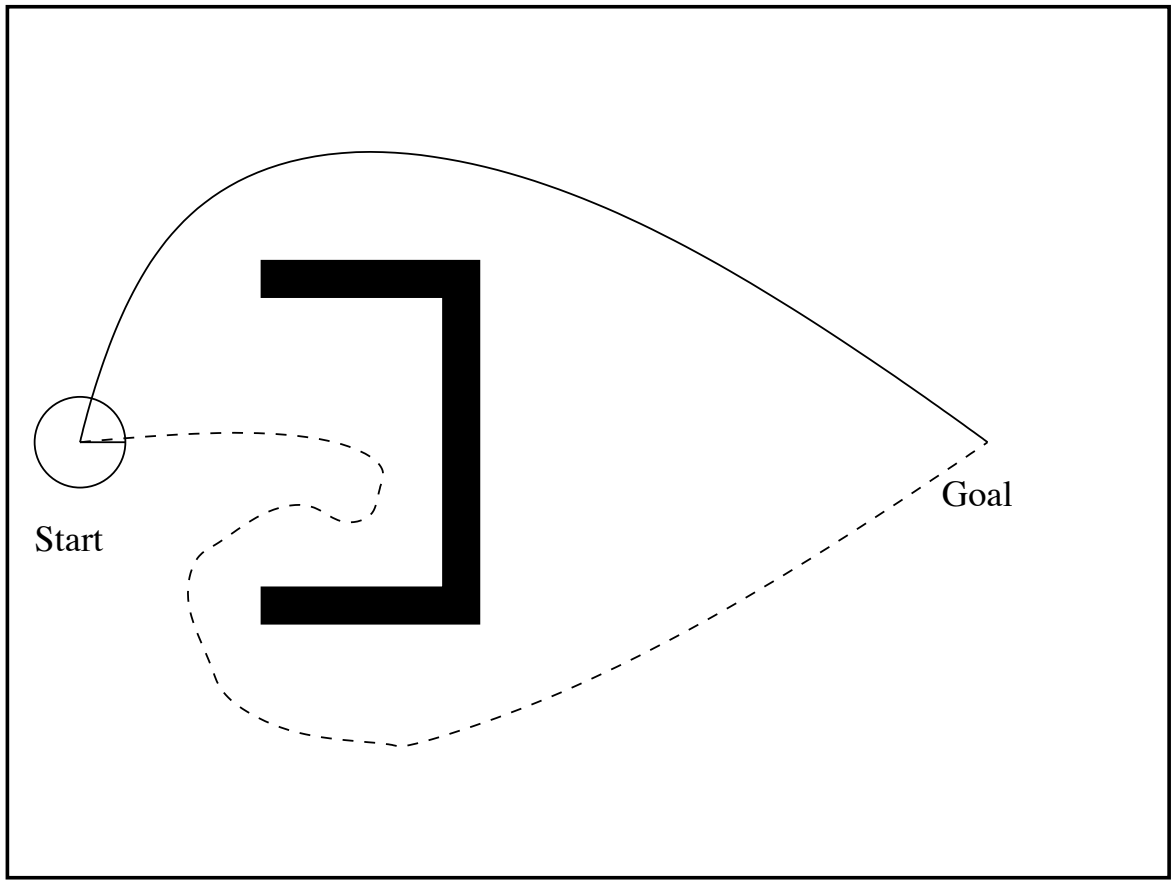
Swiss robot

- Swiss robot

  https://www.youtube.com/watch?v=B0wM-eKSxhk

- An implementation of the Swiss Robot using Lego Mindstorm NXT.
- Is the robot clustering/cleaning up (observer perspective) or just reacting to sensory stimulation (robot perspective)?

Limitations of Reactive Agents

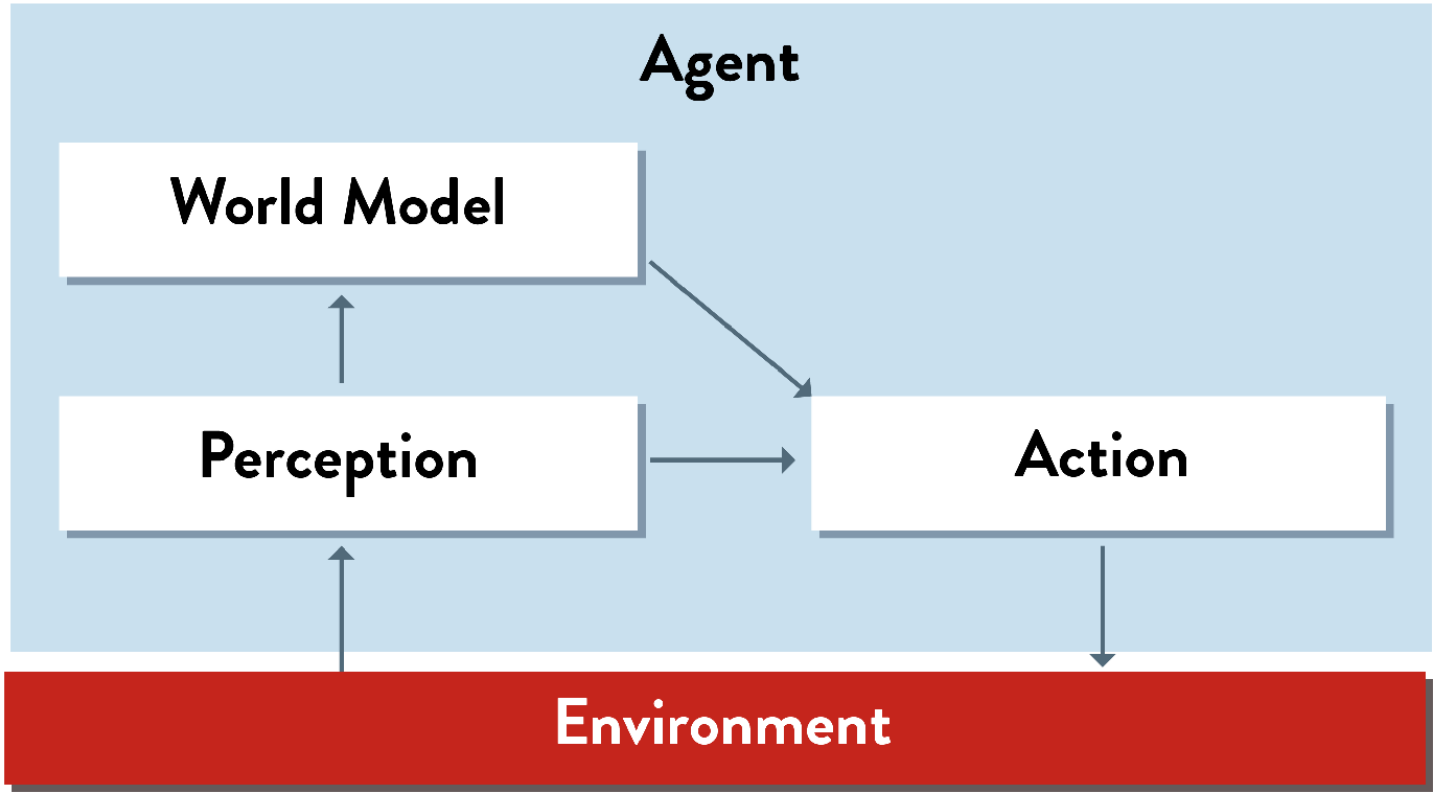


Limitations of Reactive Agents

- Reactive Agents have no memory or "state"
  - unable to base decision on previous observations
  - may repeat the same sequence of actions over and over

Escape from infinite loops is possible if the agent can **randomize** its actions.

Model-Based Agent

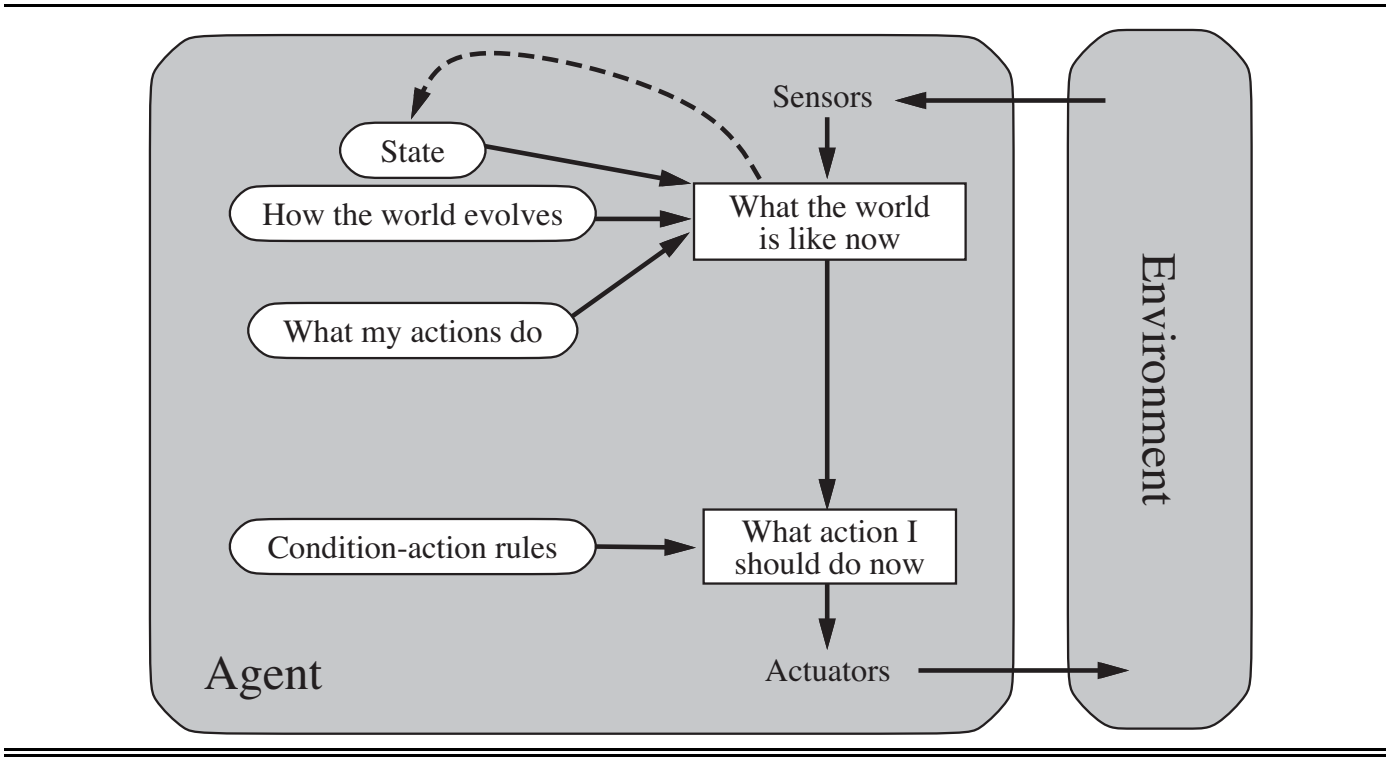


Model-based agents

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.
- Knowledge about "how the world works"—is called a **model** of the world. An agent that uses such a model is called a **model-based agent**.

Model-based agent



A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

Limitations of Model-Based Agents

An agent with a world model but no planning can look into the past, but not into the future; it will perform poorly when the task requires any of the following:

- searching several moves ahead
  - Chess, Rubik's cube
- complex tasks requiring many individual step
  - cooking a meal, assembling a watch
- logical reasoning to achieve goals
  - travel to New York

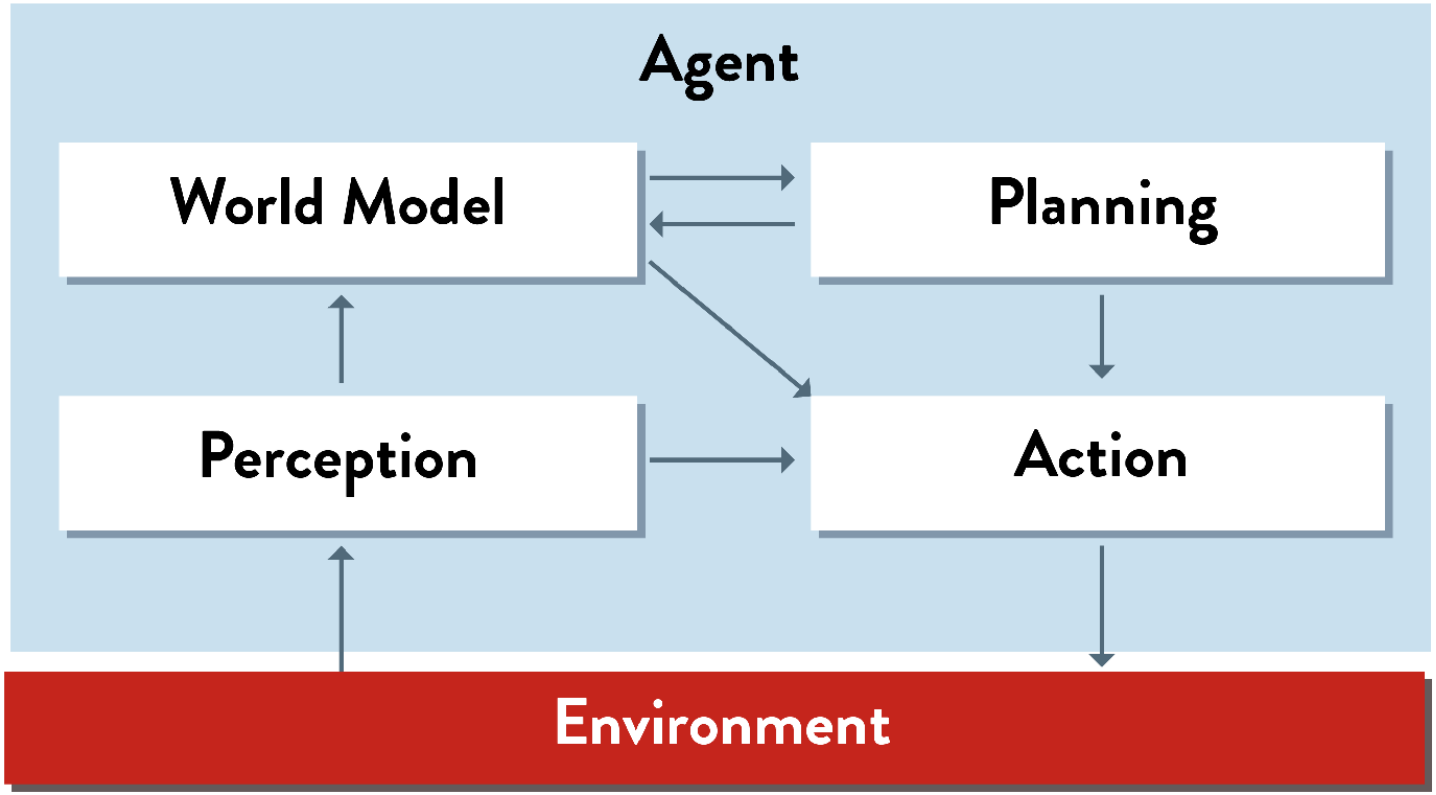
Limitations of Model-Based Agents

An agent with a world model but no planning can look into the past, but not into the future; it will perform poorly when the task requires any of the following:

- searching several moves ahead
  - Chess, Rubik's cube
- complex tasks requiring many individual step
  - cooking a meal, assembling a watch
- logical reasoning to achieve goals
  - travel to New York

Sometimes we may need to plan several steps into the future

Planning Agent



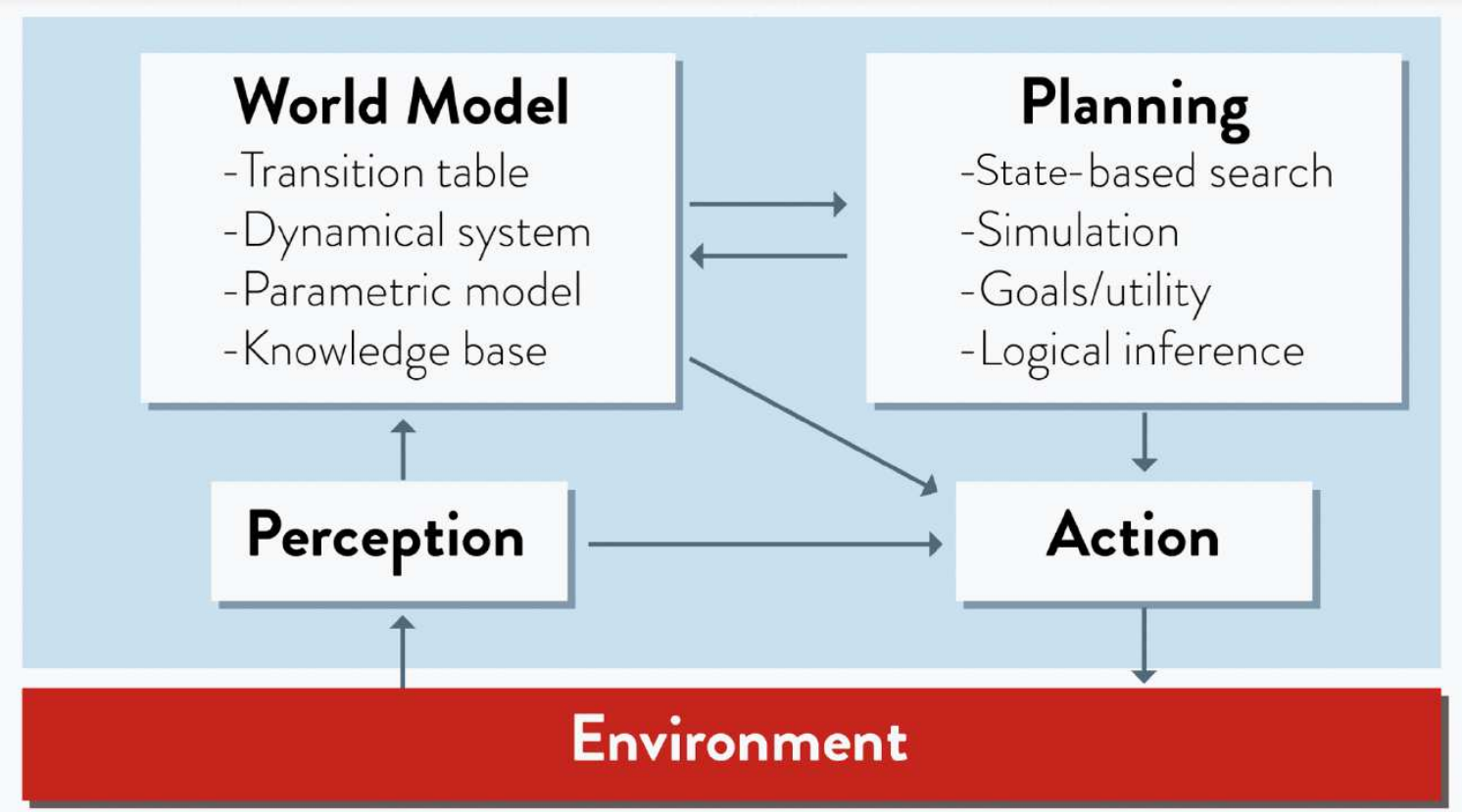
Goal-Based Agent

Planning Agent

- Decision making of this kind is fundamentally different from the condition– action rules
- It involves consideration of the future
  - "What will happen if I do such-and-such?" and
  - "Will that make me happy?"

In the reflex agent designs, this information is not explicitly represented, because the built-in rules map directly from

Models and Planning



Reasoning about Future States

- What is the best action in this situation?

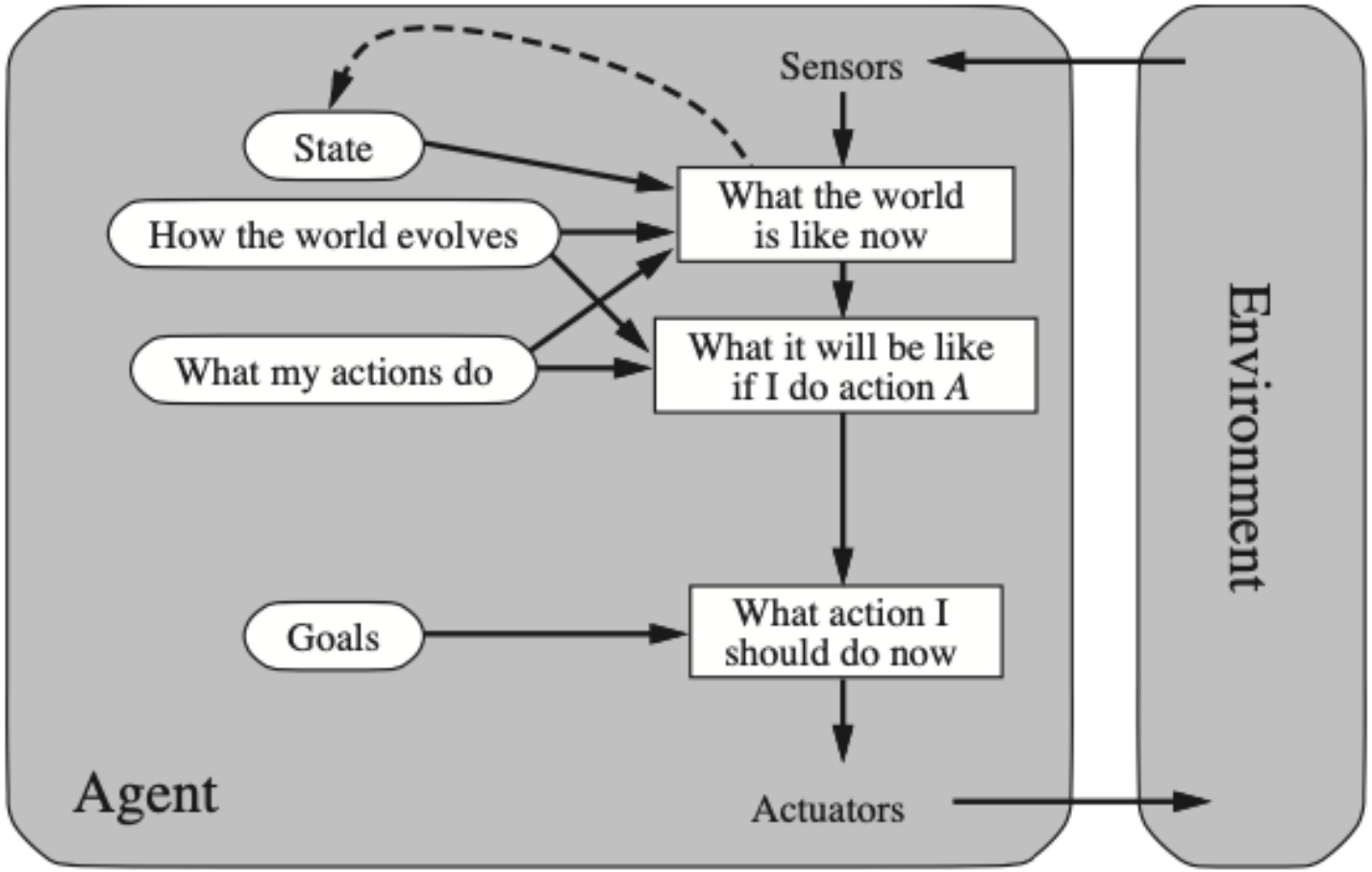
- Faking it
  - Sometimes an agent may appear to be planning ahead but is actually just applying reactive rules.
  - These rules can be hand-coded, or learned from experience.
  - Agent may appear intelligent, but is not flexible in adapting to new situations.

Planning Agent – Goal-based

- The planning agent or goal-based agent is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified.

- The agent's behavior can easily be changed.

Goal-based (teleological) agent

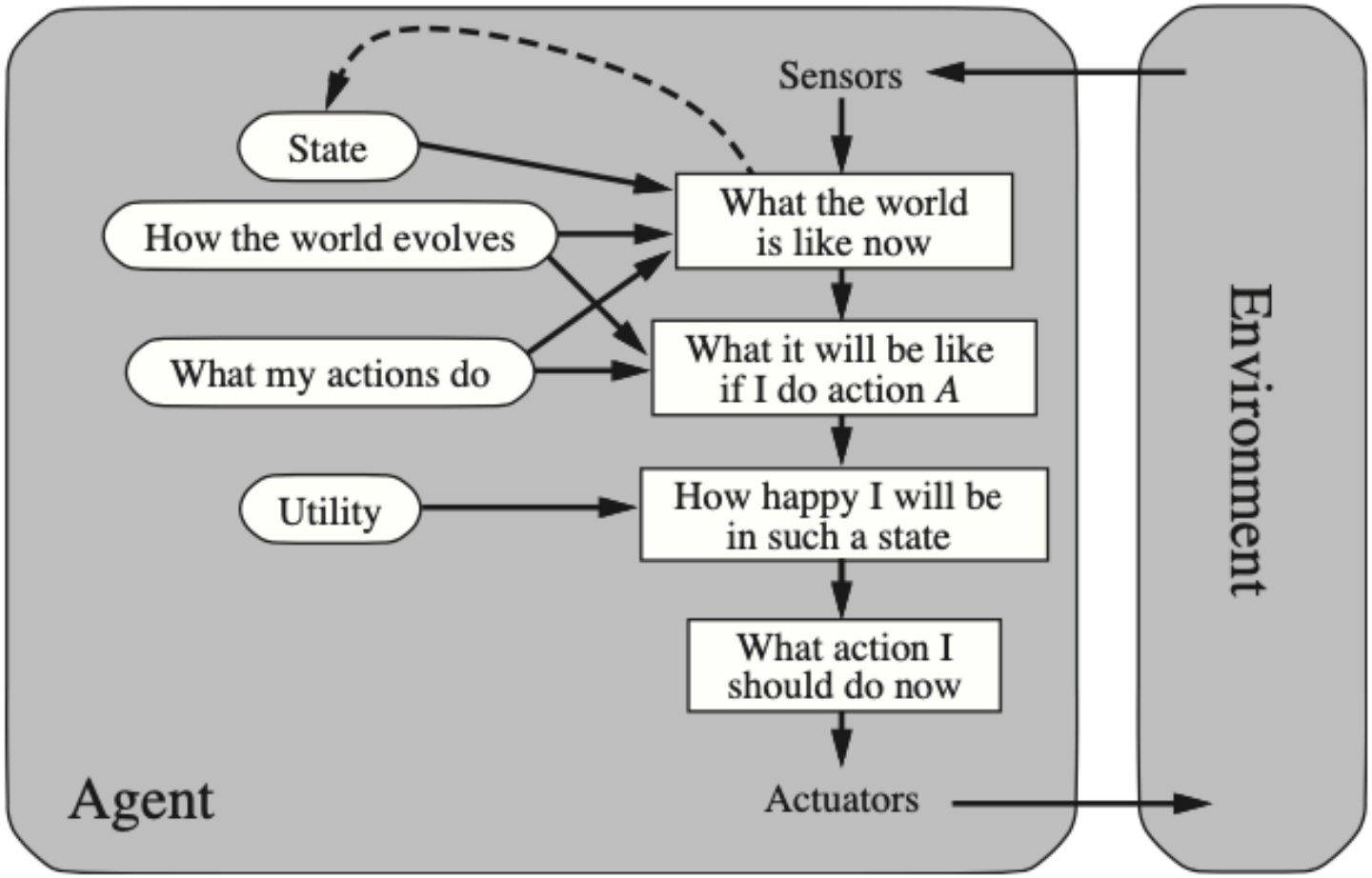


Goal-based (teleological) agent — state description often not sufficient for agent to decide what to do so it needs to consider its goals (may involve searching and planning)

Utility-based agent

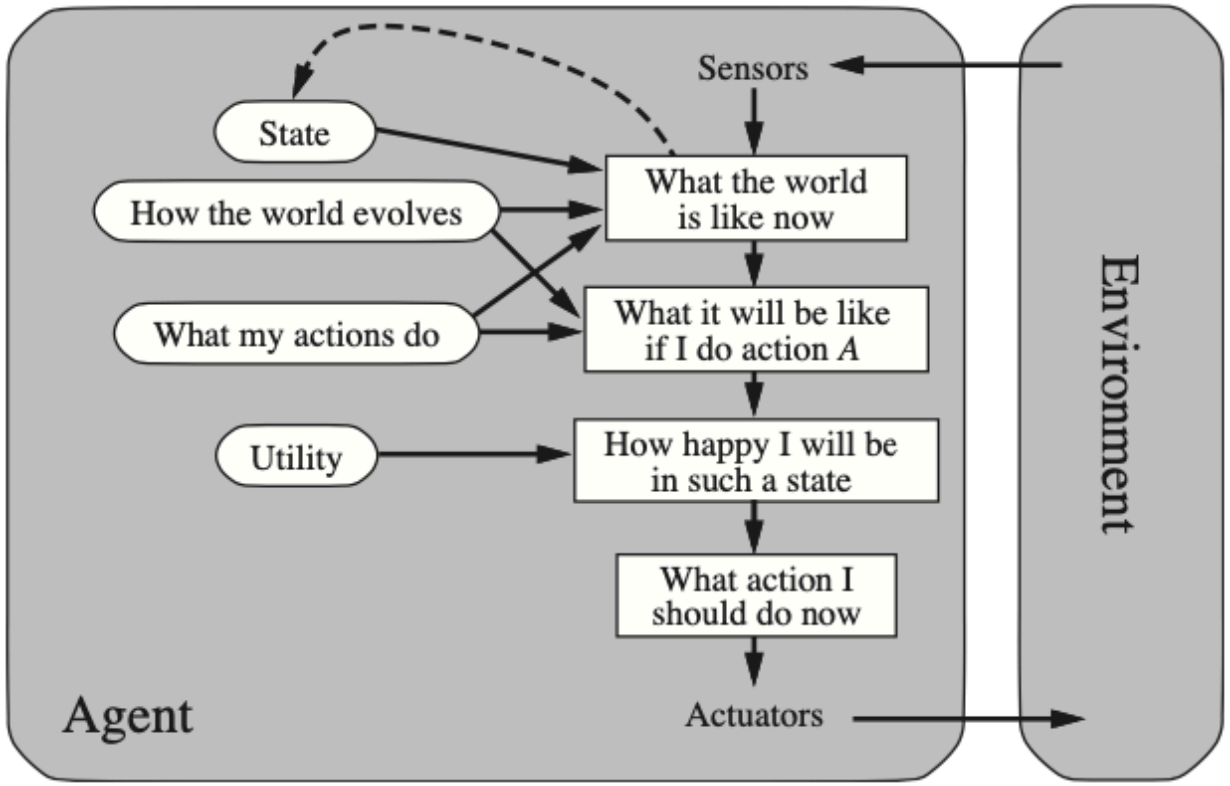
- A rational utility-based agent chooses the action that maximizes the **expected utility** of the action outcomes
  - that is, the utility the agent expects to derive, on average, given the probabilities and utilities of each outcome.
- The utility-based agent is not easy to implement
  - It has to model and keep track of its environment
  - Tasks involved a great deal of research on perception, representation, reasoning, and learning.
  - It can be implemented as a Decision-making agent that must handle the uncertainty inherent in stochastic or partially observable environments.

Utility-based agent



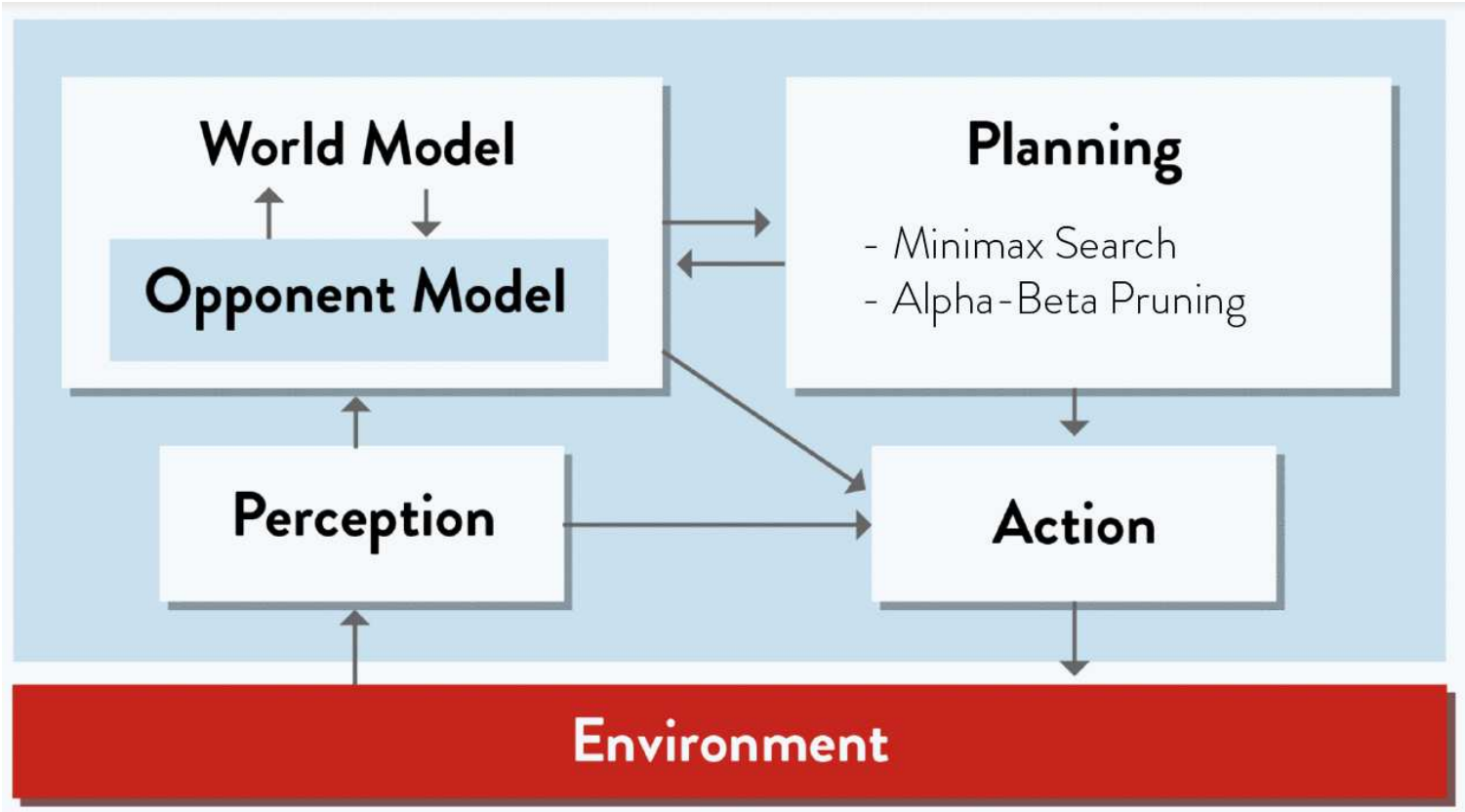
Utility-based agent — considers preference for certain world states over others

Utility-based agent

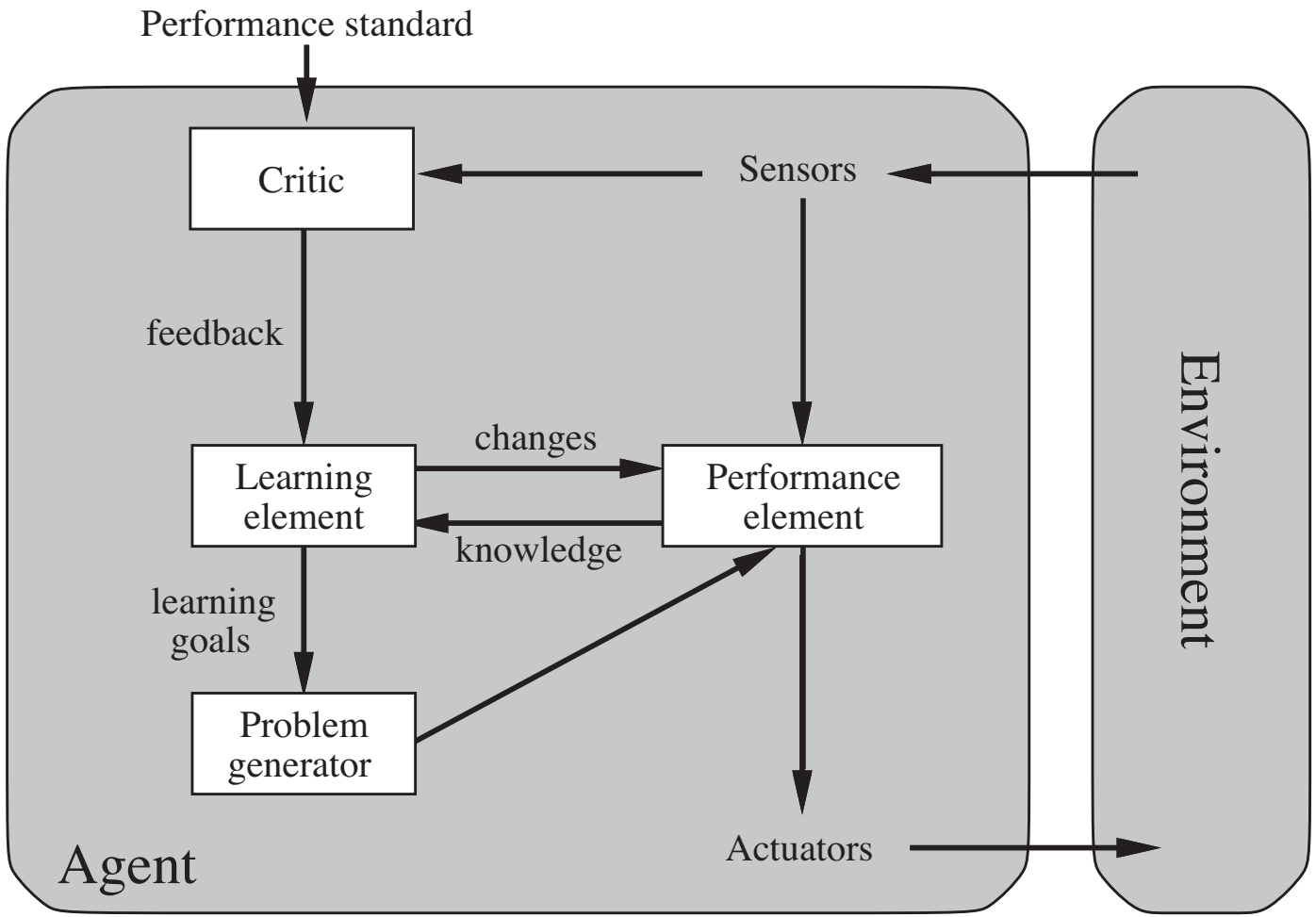


A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world.
It chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

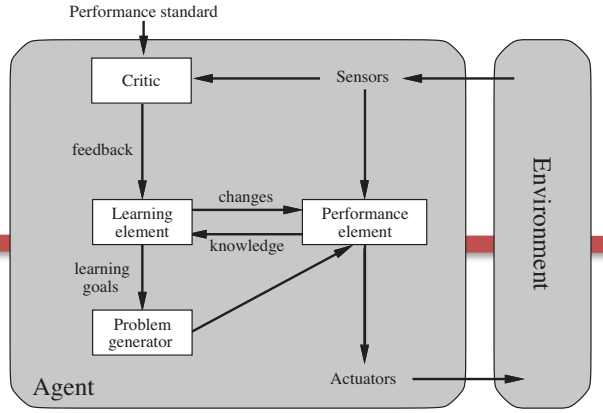
Game Playing Agent



Learning Agent



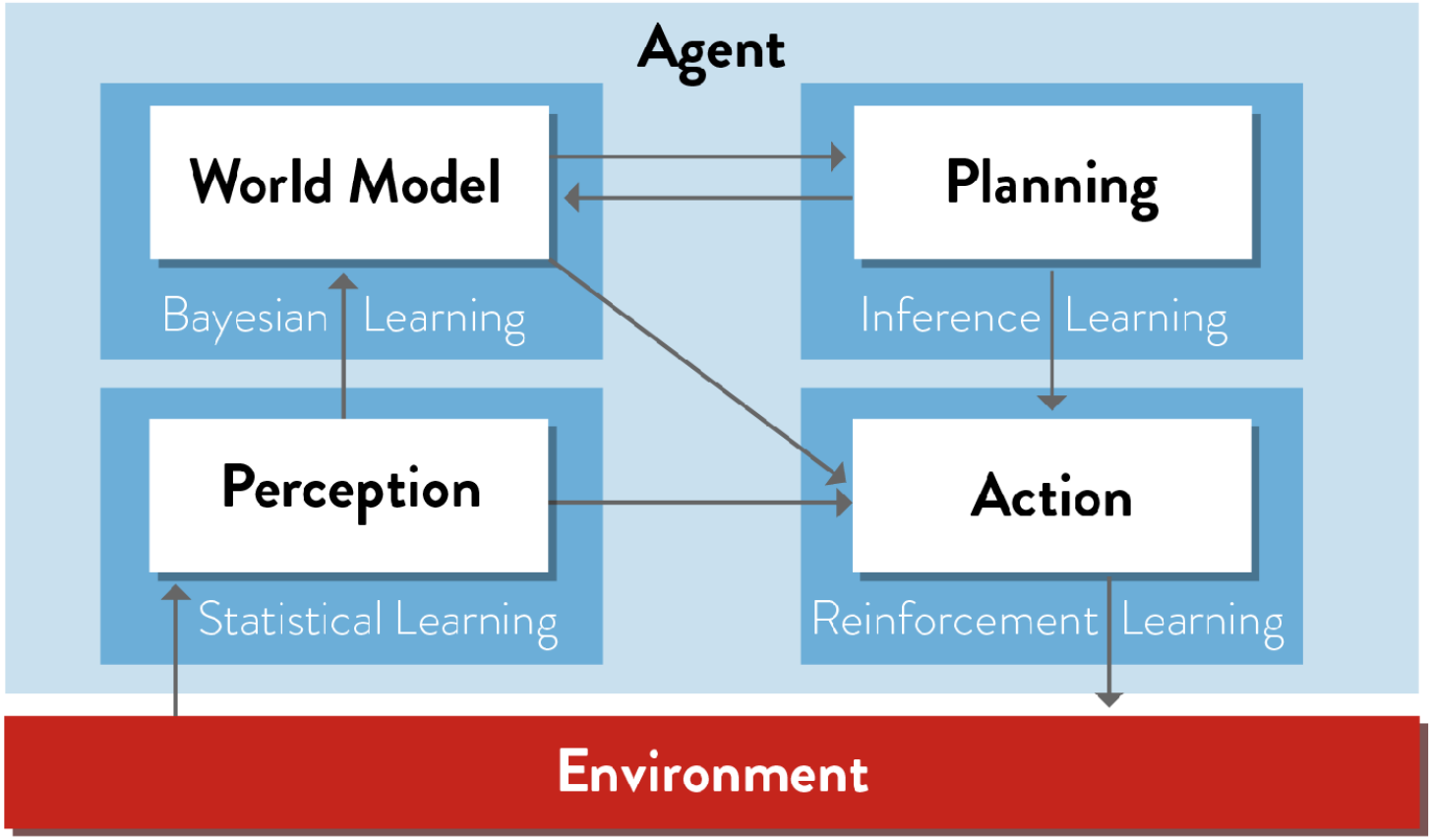
Learning Agent



- A learning agent can be divided into four conceptual components:
  - the **learning element**, which is responsible for making improvements,
  - the **performance element**, which is responsible for selecting external actions.
  - The **critic element -** tells the learning element how well the agent is doing with respect to a fixed performance standard.
  - The **problem generator** -for suggesting actions that will lead to new and informative experiences.

The performance element takes in percepts and decides on actions. The learning element uses feedback from the **critic** on how the agent is doing and determines how the performance element should be modified to do better in the future.

Learning Agent



Learning

- Learning is not a separate module, but rather a set of techniques for improving the existing modules
- Learning is necessary because:
  - may be difficult or even impossible for a human to design all aspects of the system by hand
  - the agent may need to adapt to new situations without being re-programmed by a human

Summary

- The **agent program** implements the agent function.
- A variety of basic agent-program designs reflecting the kind of information made explicit and used in the decision process.
  - The designs vary in efficiency, compactness, and flexibility.
  - The appropriate design of the agent program depends on the nature of the environment.

- **Simple reflex agents** respond directly to percepts,
- **model-based reflex agents** maintain internal state to track aspects of the world that are not evident in the current percept.
- **Planning (Goal-base) agents** act to achieve their goals, and
- **utility-based agents** try to maximize their own expected "happiness."
- All agents can improve their performance through **learning**.

Agent Programs and Architectures

- Program — function implementing mapping from percept sequence to actions, using an internal representation of the percept history

- Architecture — hardware and software components, and their organization, on which agent program executes

Agent = Architecture + Program

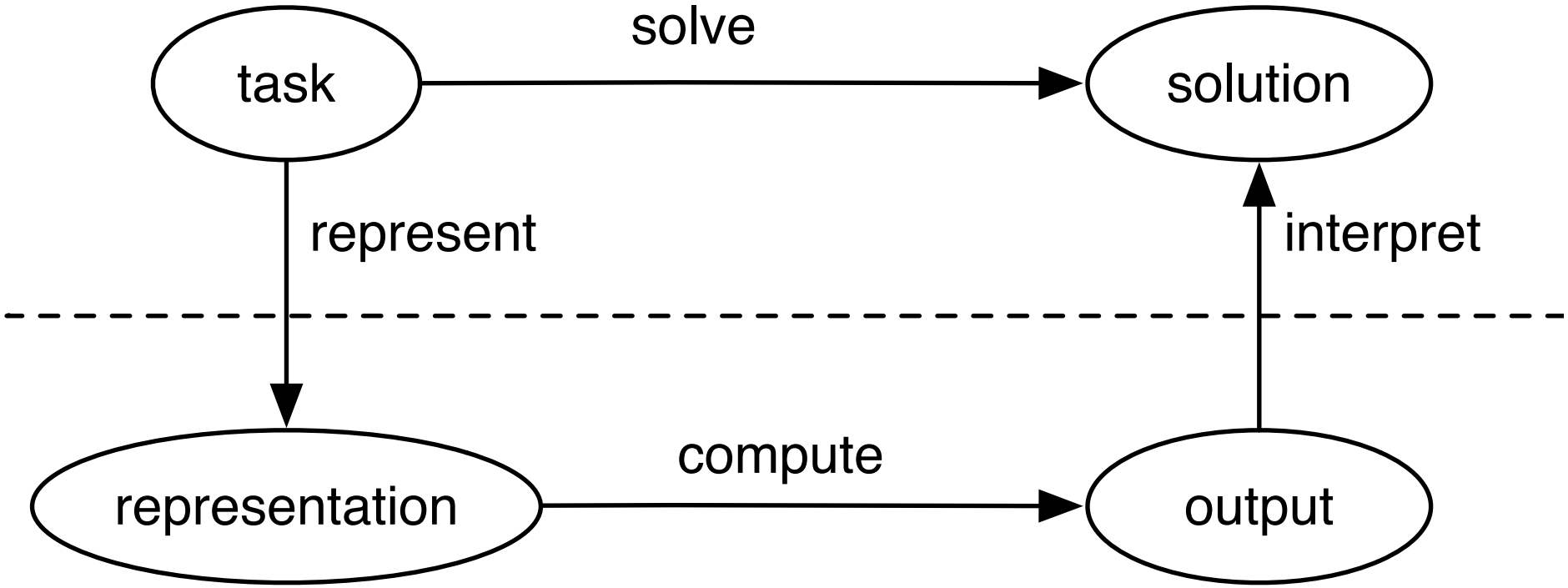
Tasks

- One way that AI representations differ from computer programs in traditional languages.
- An AI representation typically specifies *what needs* to be computed, not how it is to be computed.
  - We might specify that the agent should find the most likely disease a patient has, or specify that a robot should get coffee, but not give detailed instructions on how to do these things.
- Much AI reasoning involves searching through the space of possibilities to determine how to complete a task.

Tasks

- Typically, a task is only given informally, such as "deliver parcels promptly when they arrive" or "fix whatever is wrong with the electrical system of the house."

Agents acting in an environment



The role of representations in solving tasks

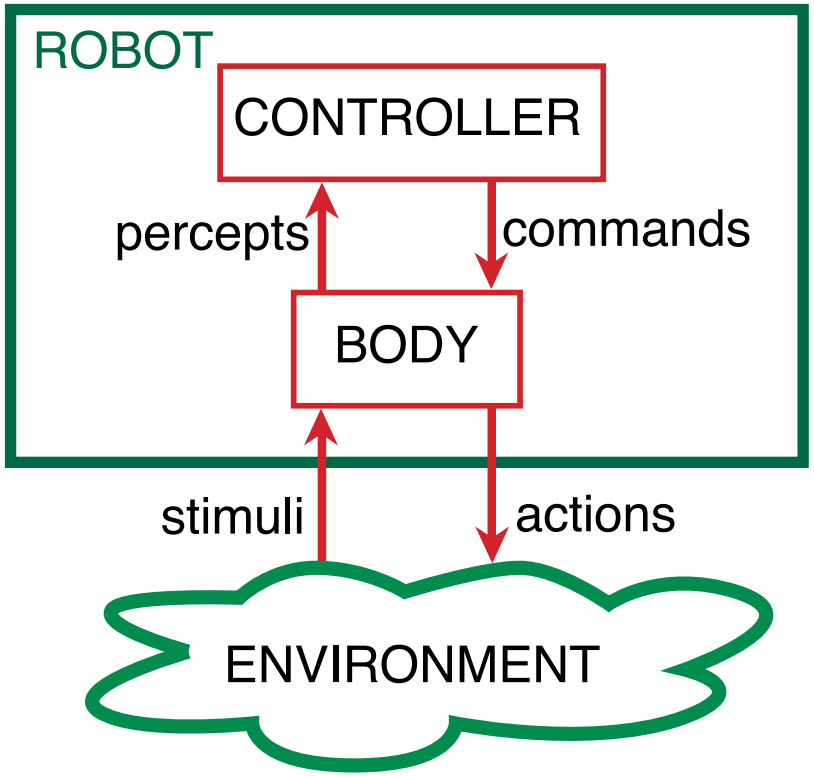
The general framework - solving tasks by computer

- To solve a task, the designer of a system must:
  - determine what constitutes a solution
  - represent the task in a way a computer can reason about
  - use the computer to compute an output, which is answers presented to a user or actions to be carried out in the environment, and
  - interpret the output as a solution to the task.

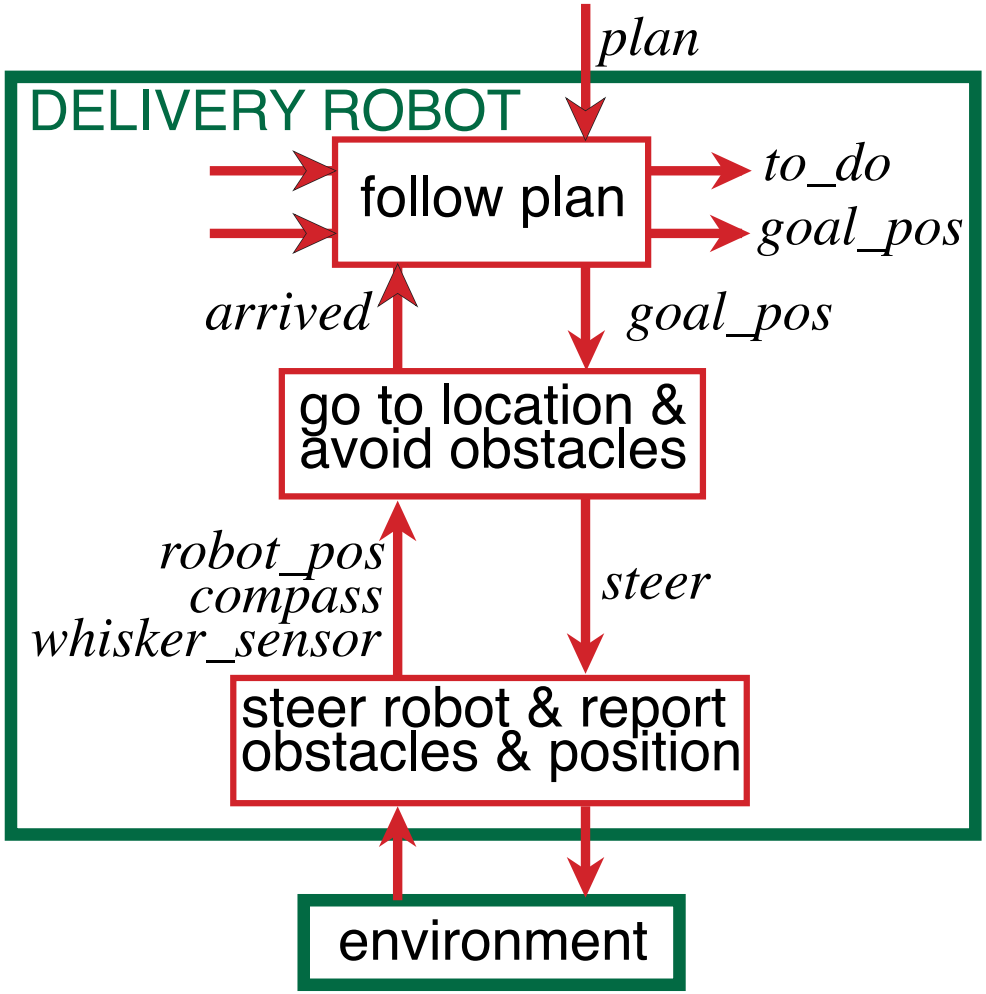
Representation

- Rich enough to express knowledge needed (to solve the problem)
- As close to the problem as possible: compact, natural, maintainable
- Amenable to efficient computation
  - Able to express features of the problem that can be exploited for computational gain
  - Able to trade off accuracy and computation time and/or space
  - Able to be acquired from people, data and past experiences

Example – Delivery Robot

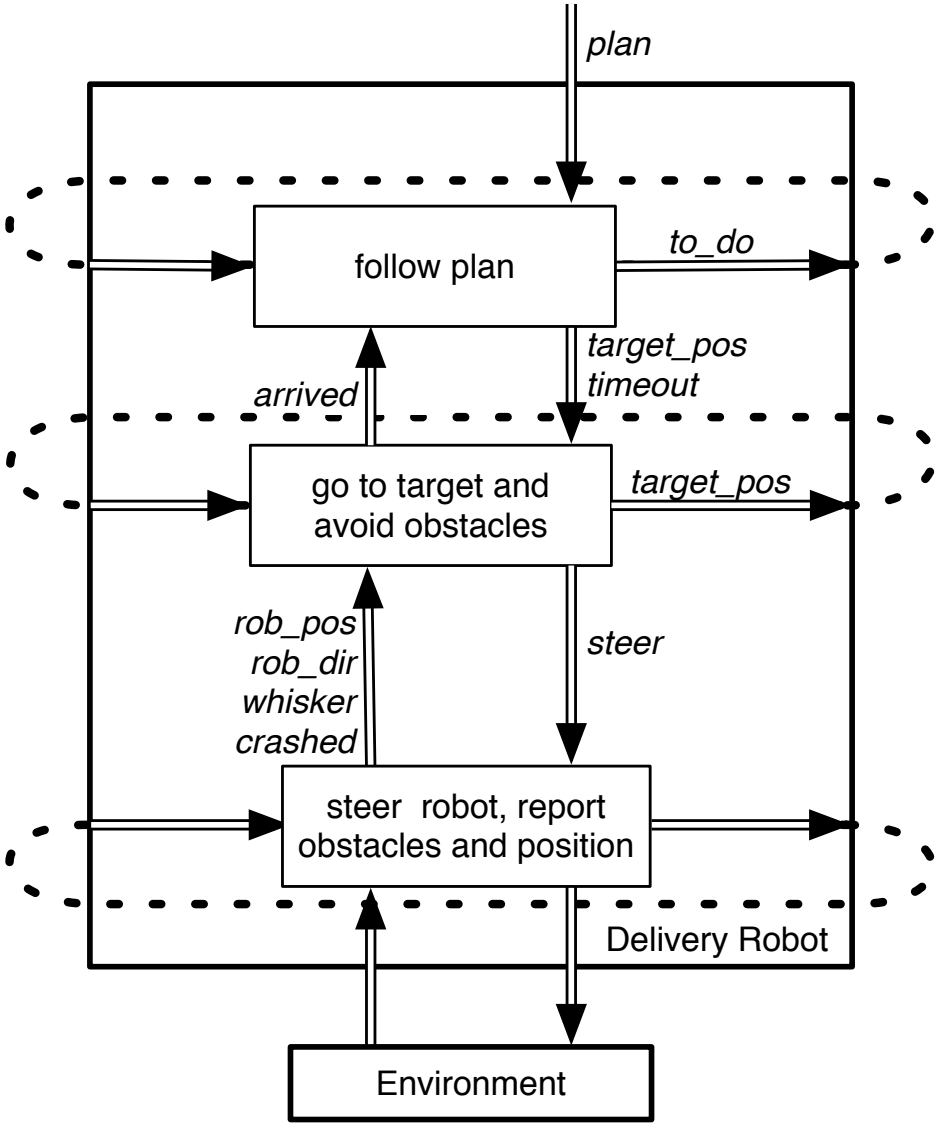


Example – Delivery Robot

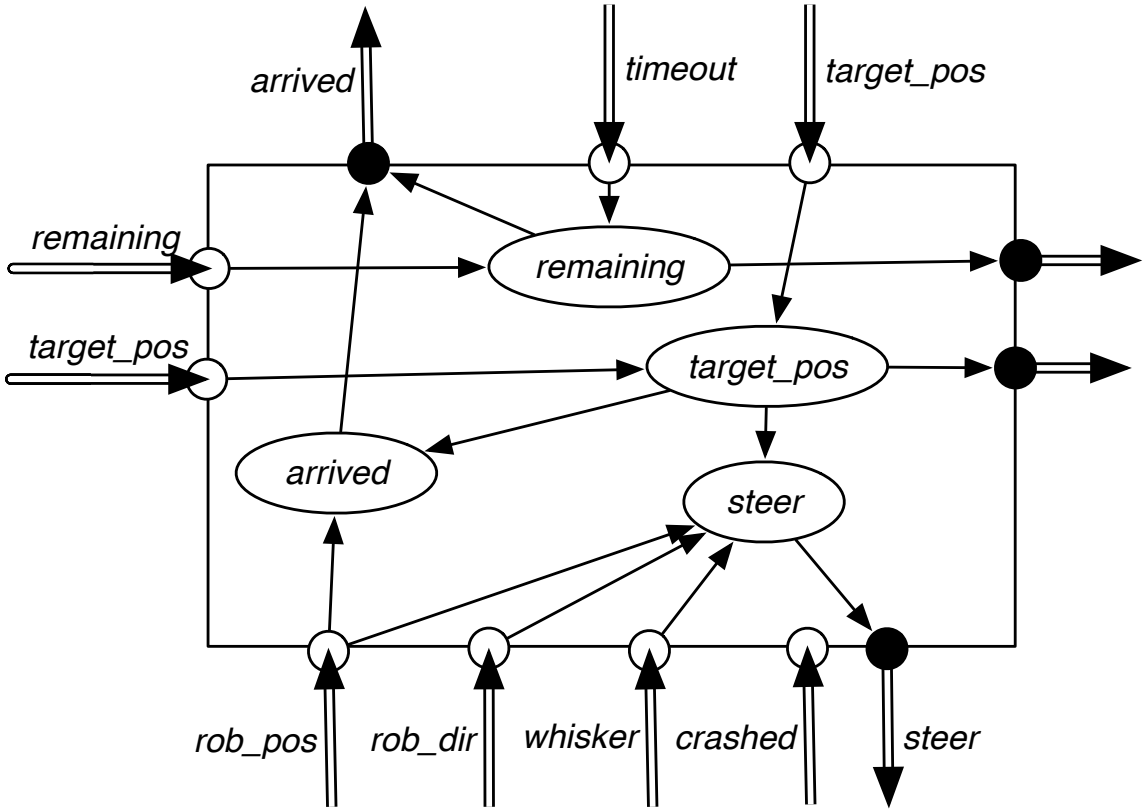


Layered Architecture

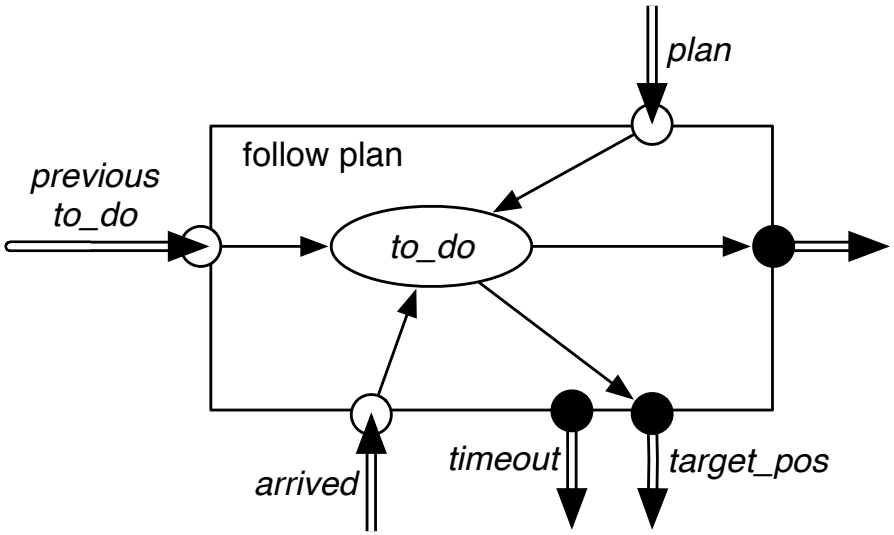
- Hierarchy of controllers
- Controller gets percepts from and sends commands to the lower layer
  - Abstracts low level features into higher level (perception)
  - Translates high level commands into actuator instructions (action)
- The controllers have different representations, programs
- The controllers operate at different time scales
- A lower-level controller can override its commands


plan
follow plan
to_do
target_pos
timeout
arrived
go to target and
avoid obstacles
target_pos
rob_pos
rob_dir
whisker
crashed
steer
steer robot, report
obstacles and position
Delivery Robot
Environment

Delivery Robot – Middle Layer



Delivery Robot – Top Layer



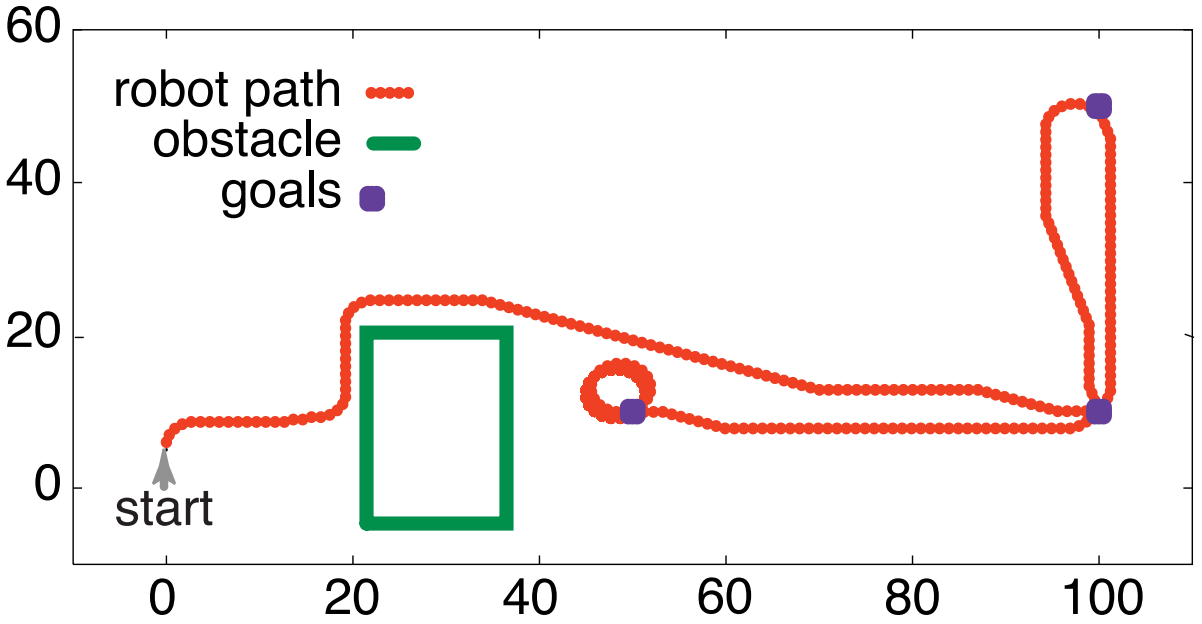
Middle Layer Code

```
given timeout and target pos:
   remaining := timeout
   while not arrived() and remaining =/= 0
      if whisker sensor = on
            then steer := left
      else
       if straight ahead(rob pos; robot dir; target pos)
            then steer := straight
      else
        if left of (rob pos; robot dir; target pos)
            then steer := left
         else steer := right
         do(steer)
         remaining := remaining – 1
      tell upper layer arrived()
```

Top Layer Code

```
given plan:
   to do := plan
   timeout := 200
   while not empty(to do)
      target pos := coordinates(first(to do))
      do(timeout; target pos)
      to do := rest(to do)
```

Delivery Robot – Simulation



References

- Poole &Mackworth, Artificial Intelligence: Foundations of Computational Agents, Chapter 1 & 2
- Russell & Norvig, *Artificial Intelligence: a Modern Approach*, Chapter 2.