CS 331: Introduction to Artificial Intelligence

Oregon State University

Lecture 2: Agents

Sandhya Saisubramanian

Announcements

- Quiz 1 will be released on Oct 1 at 2 pm and will be active until Oct 3 11:59pm
- Topics for Quiz 1: Agents and uninformed search (Lectures 2,3)
- HW1 will be posted on Oct 1; due on Oct 10
- Change in TA office hours: Dilumika's office hours will be on Friday 9-10 am (instead of Thursday)

Today's Agenda

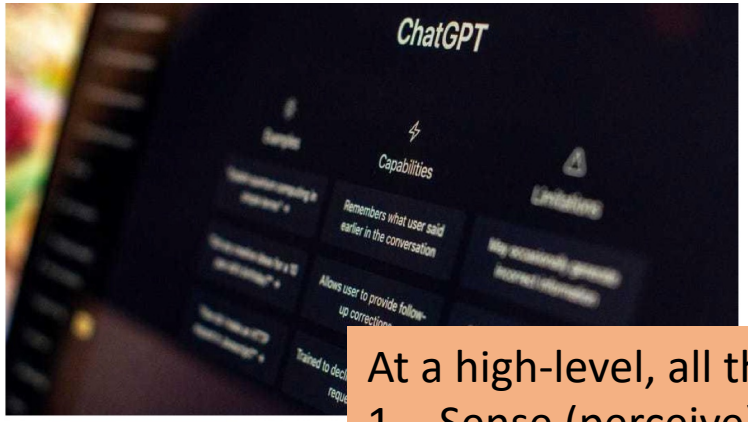
What is an agent?

What does it mean to design an intelligent agent?

How can these agents decide what to do/ how to act?

Readings: Chapters 2 and 3 in course textbook

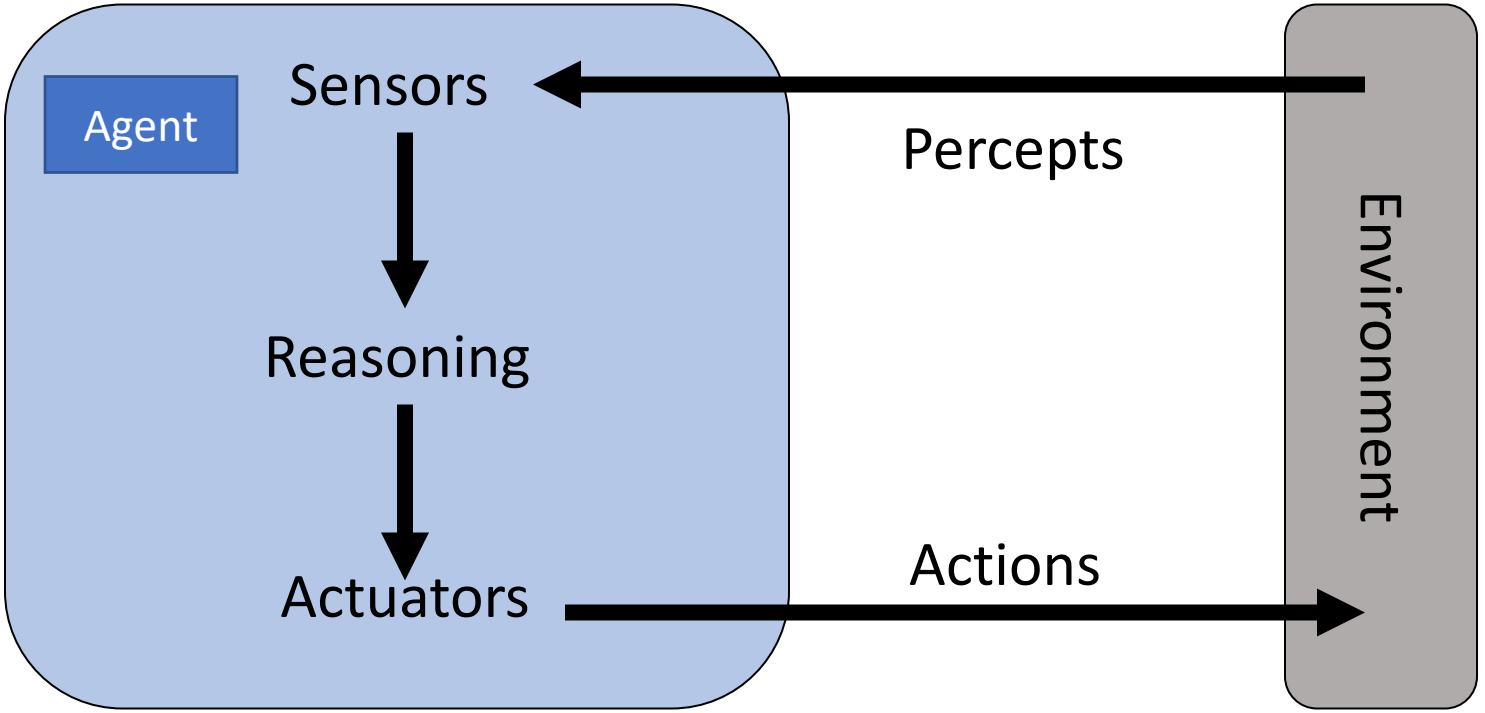
Few Real-World Examples



At a high-level, all these systems perform the following:

1. Sense (perceive)
2. Think (decide what to do)
3. Act (produce some output)

Agent-Centric View of AI

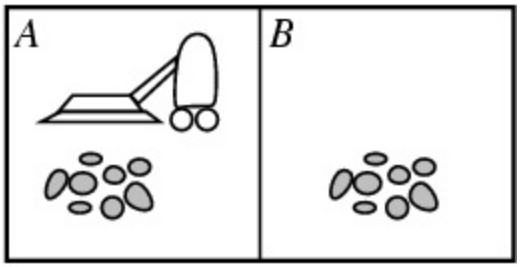


Agent: anything that perceives its environment through sensors and acts on that environment through actuators

Agent-Related Terms

- **Percept sequence (P)**: A complete history of everything the agent has ever perceived. Think of this as the state of the world from the agent's perspective.
- **Agent Policy:** Maps percept sequence to action (determines agent behavior)
  - Also called as agent function in some textbooks
  - Let **P** denote the percept history and **A** denote the set of actions available. The agent policy or function is denoted by $\boldsymbol{f}: \boldsymbol{P} \rightarrow \boldsymbol{A}$
  - Ideal mapping or expected behavior ("what it should do")
- **Agent Program**: A concrete implementation of the agent function on a machine.
  - How the agent function is realized in practice. Example: A Python implementation

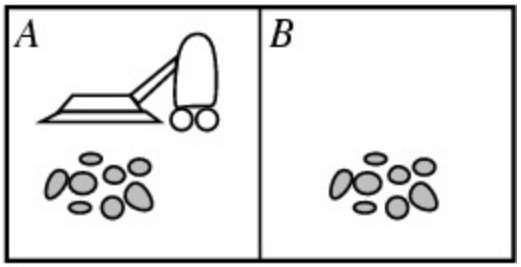
Example: Vacuum Cleaner Agent



Sensors: Camera to detect obstacles and dirt

Percept Sequence: Cleanliness status

Example: Vacuum Cleaner Agent



Sensors: Camera to detect obstacles and dirt

Percept Sequence: Cleanliness status

Actuators: Suction brush, wheels

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
:	:
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
:	:

Agent function

Agent Design and Implementation

Before implementing an agent, we need to think carefully about:

1. The environment in which it will operate
2. What sensors are required or available?
3. How will it achieve the task? What capabilities are available?
   - A vacuum cleaner cannot clean the floor if does not have a mechanism to suck the dirt
4. How should the agent act? How will its behavior or performance be evaluated?

PEAS Descriptions of Task Environments

Performance, Environment, Actuators, Sensors

PEAS is the standard way of specifying an **agent task environment** before you even design the agent

Example: Autonomous taxi

Performance Measure	Environment	Actuators	Sensors
Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

PEAS Descriptions of Task Environments

Performance, Environment, Actuators, Sensors

Example: Medical Diagnosis Agent

Performance Measure	Environment	Actuators	Sensors
Healthy patient, minimize costs, lawsuits	Patient, hospital, staff	Display questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers

Properties of Environments

Fully observable: can access complete state of environment at each point in time	Partially observable: could be due to noisy, inaccurate or incomplete sensor data
Deterministic: next state of the environment completely determined by current state and agent's action	Stochastic: when actions have multiple outcomes, each prescribed by a probability
Episodic: agent's experience divided into independent, atomic episodes in which agent perceives and performs a single action in each episode.	Sequential: current decision affects all future decisions
Static: agent doesn't need to keep sensing while decides what action to take, doesn't need to worry about time	Dynamic: environment changes while agent is thinking (changes with time)
Discrete: (note: applies to states, time, percepts, or actions)	Continuous: continuous values of states and/or actions
Single agent: single decision-making and executing entity	Multiagent: multiple decision-making/executing entities; cooperative or competitive

How Should An Agent Act?

Given:

- A performance measure
- A percept sequence
- Agent's knowledge
- A set of available actions

We still have one key piece before we can implement the agent…

How should an agent act? In other words, how should it select its actions? How should the agent function be constructed?

Intelligence emerges from how agents choose actions

How Should An Agent Act?

Given:

- A performance measure
- A percept sequence
- Agent's knowledge
- A set of available actions

An ideal agent will outperform any other agent in maximizing the performance measure

Is this well-defined?

Does not consider the agent's knowledge, its percept sequence or the environment in which it is operating

Rational Agent

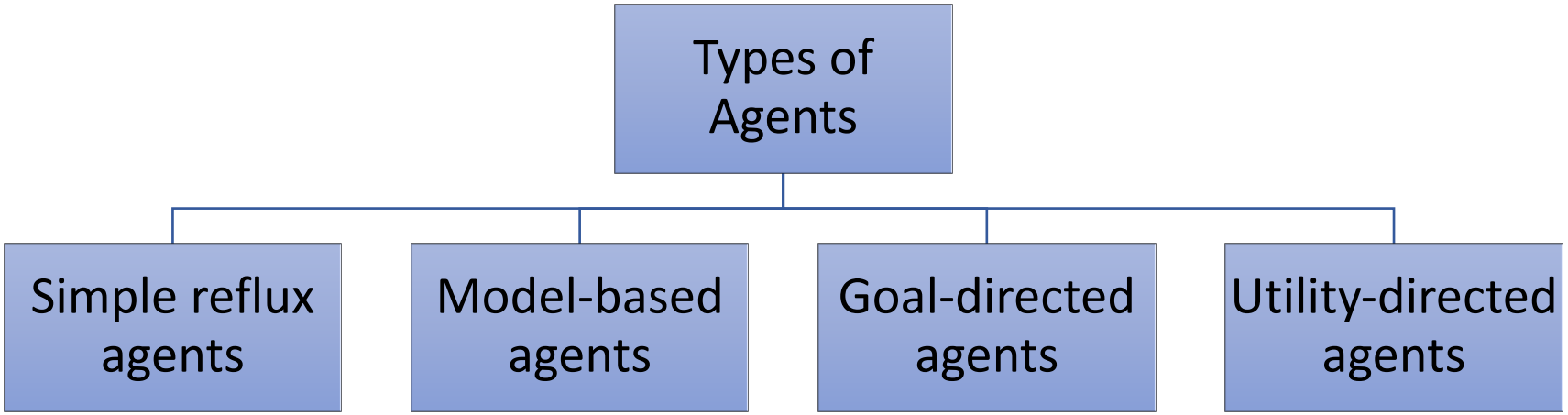
Rational agent: for each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has

Rationality depends on 4 things:

1. Performance measure of success
2. Agent's prior knowledge of environment
3. Actions agent can perform
4. Agent's percept sequence to date

Types of Agents

- Categorized based on their performance measure



Simple Reflex Agents

- Simple if-then: **if** *condition* **then** *action*; no internal model
- Selects actions using only the current percept

Example: if temp < 68 → turn heater on

function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action

static: *rules*, a set of condition-action rules

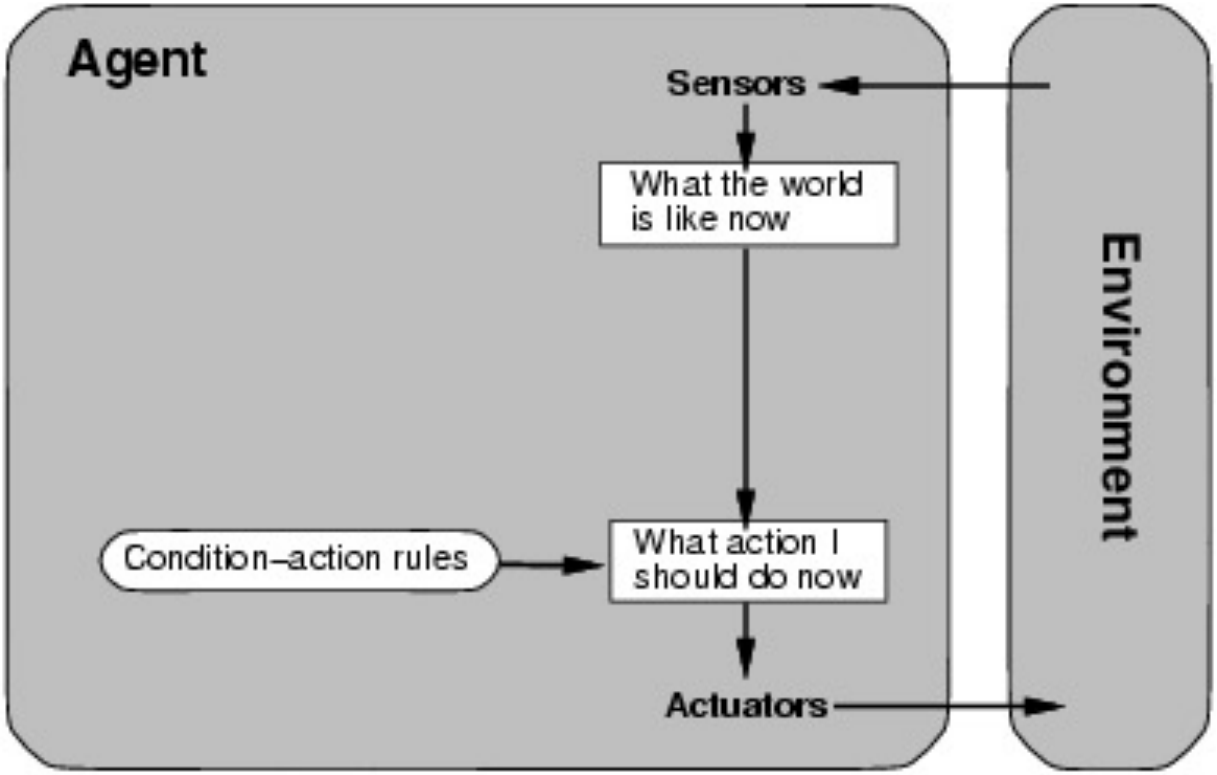
state ← INTERPRET-INPUT(*percept*)

rule ← RULE-MATCH(*state*, *rules*)

action ← RULE-ACTION[*rule*]

return *action*

Simple Reflex Agents



Simple Reflex Agents

- Advantages:
  - Easy to implement
  - Uses much less memory than the table-driven agent
- Disadvantages:
  - Will only work correctly if the environment is fully observable
  - Infinite loops

Model-based Reflex Agents

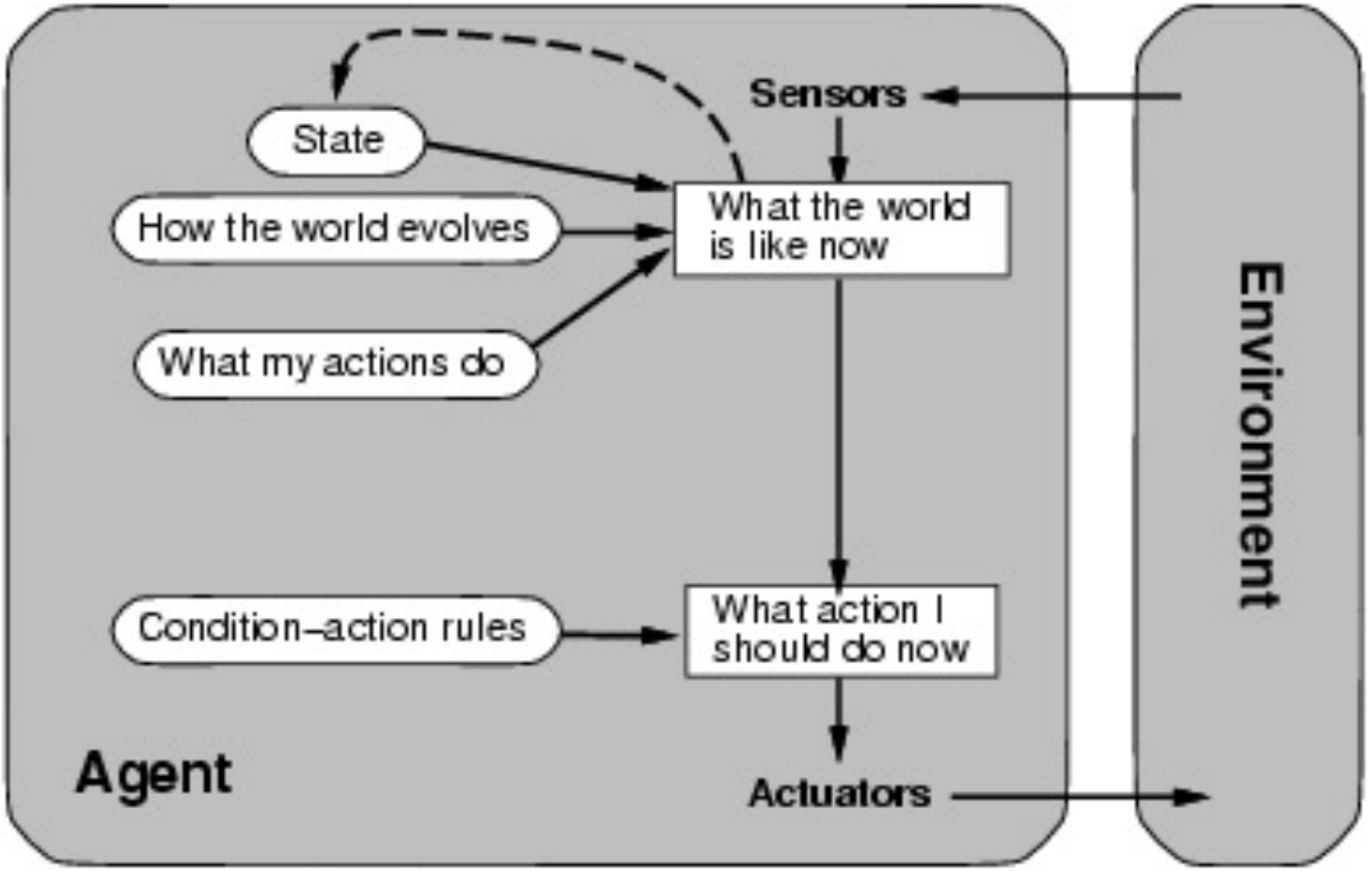
- Maintain some internal state that keeps track of the part of the world it can't see now (partial observability)
- Needs model (encodes knowledge about how the world works)

```
function REFLEX-AGENT-WITH-STATE(percept) returns an action
    static: state, a description of the current world state
        rules, a set of condition-action rules
        action, the most recent action, initially none

    state ← UPDATE-STATE(state, action, percept)
    rule ← RULE-MATCH(state, rules)
    action ← RULE-ACTION[rule]
    return action
```

Example: Vacuum cleaner robot
The internal model ("map of which rooms are clean/dirty") allows it to act sensibly despite partial observability.

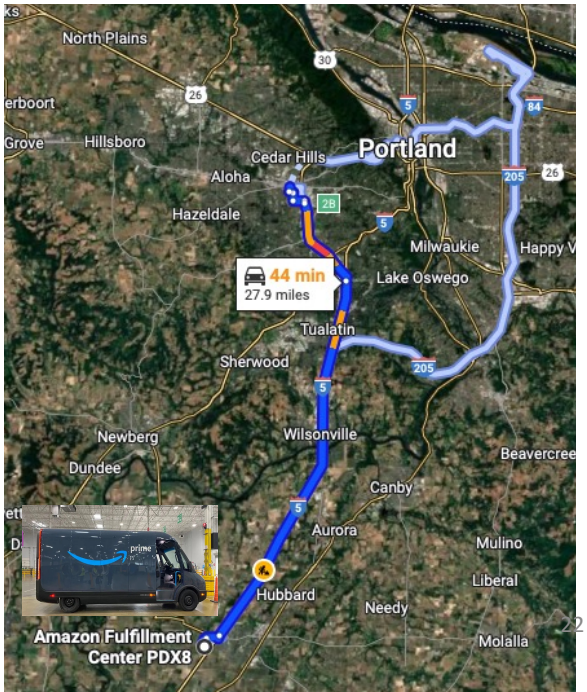
Model-based Reflex Agents



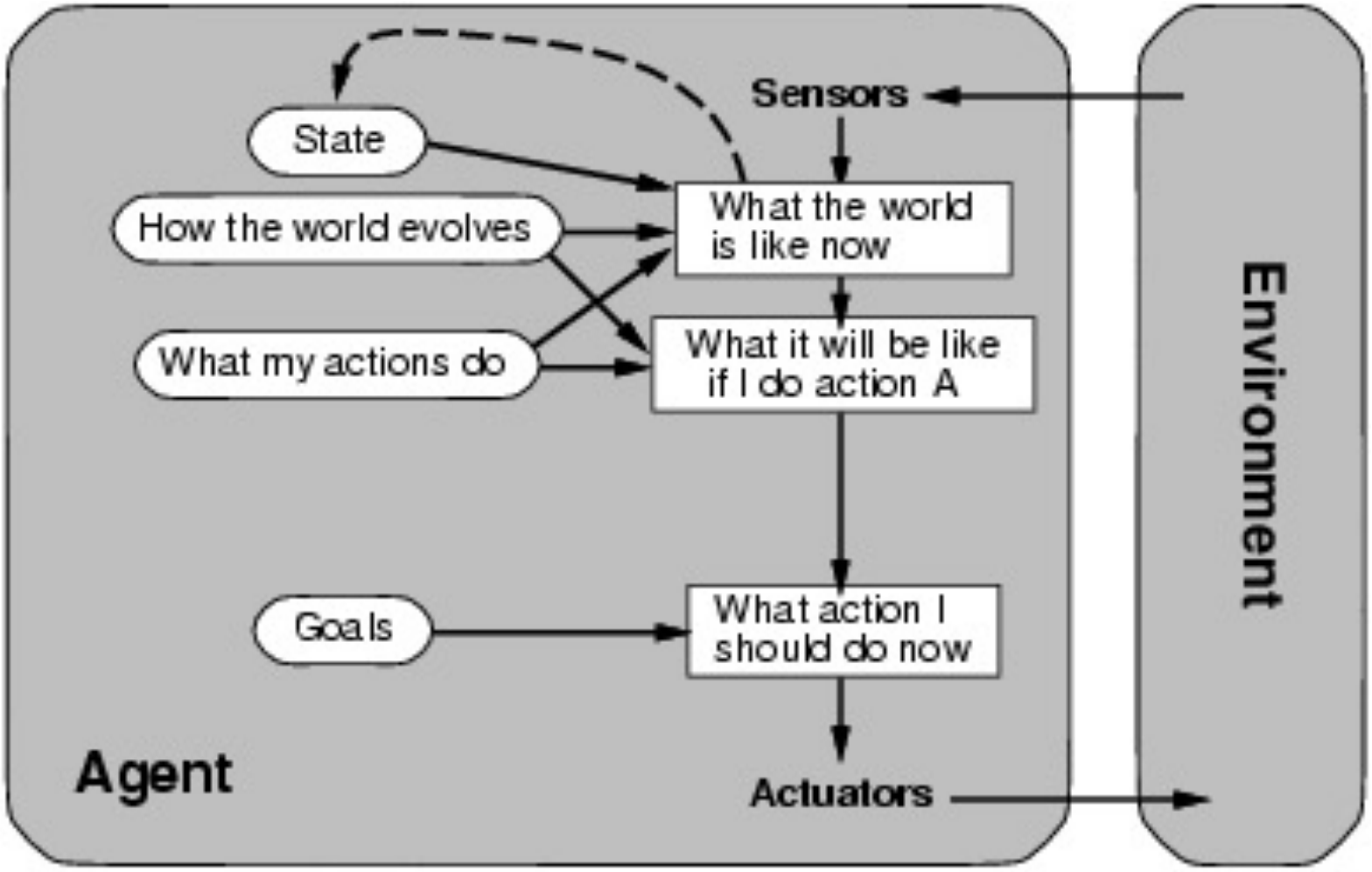
Goal-directed Agents

- Goal information guides agent's actions (looks to the future)
- Sometimes achieving goal is simple e.g. from a single action
- Other times, goal requires reasoning about *long sequences of actions* → accounts for future states
- Flexible: simply reprogram the agent by changing goals

Example: Navigation



Goal-directed Agents



Utility-directed Agents

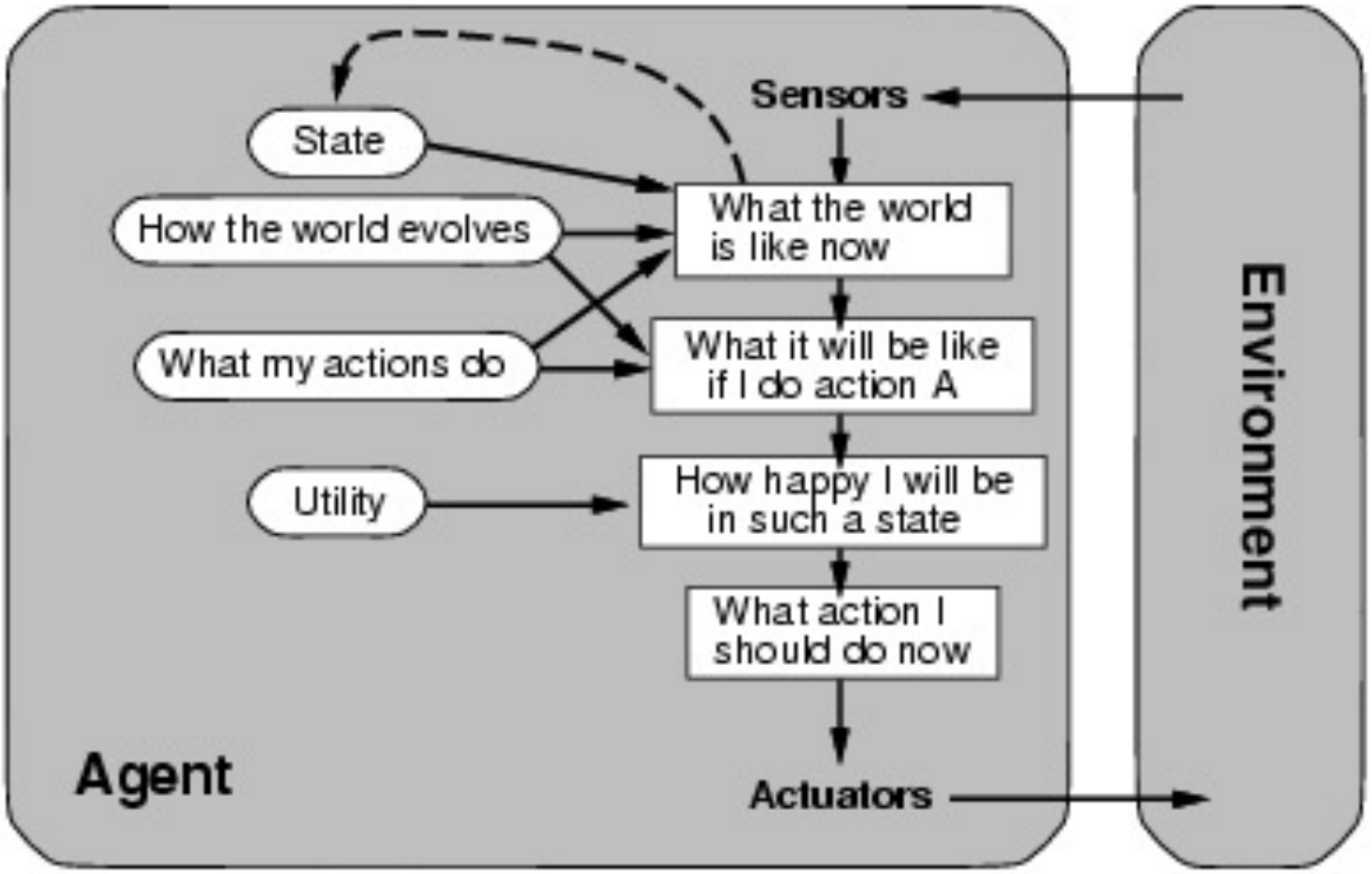
- What if there are many paths to the goal? → optimize trade-offs when multiple outcomes are possible
- Utility measures which states are preferable to other states
- Assign numeric values to each possible outcome (utility or "happiness")

Common types of utility definitions:

- Multidimensional utility (quality, failure rate, etc.)
- Time-dependent utility (hard/soft deadlines)
- Subjective vs. objective utility functions
- Qualitative vs. quantitative

Example: Self-driving car balancing speed vs. safety vs. comfort

Utility-directed Agents



Types of Agents

- Categorized based on their performance measure

- Simple reflux agents
- Model-based reflex agents
- Goal-directed agents
- Utility-directed agents
- Learning agents

Learning Agents

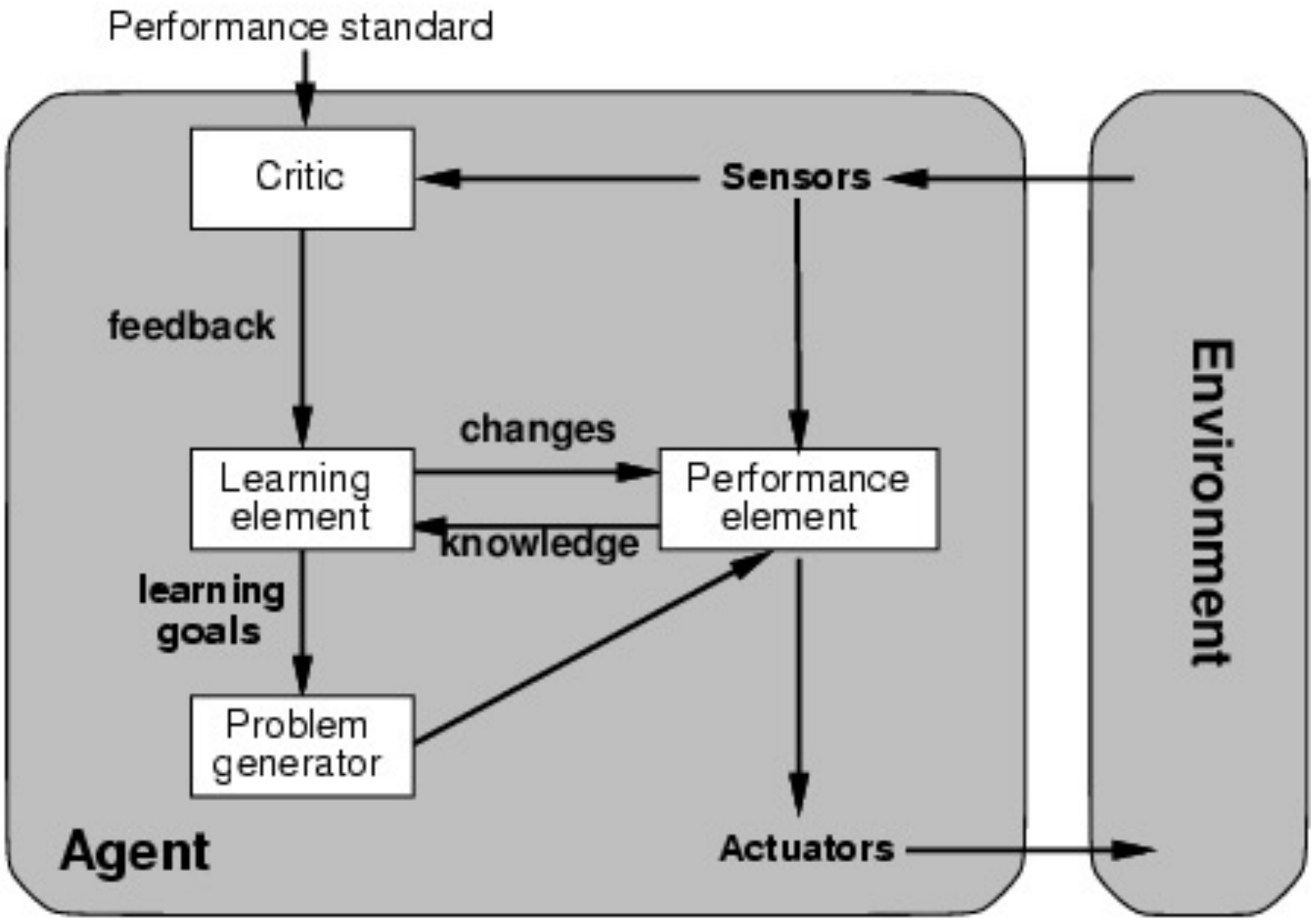
Successful agents split task of computing policy in 3 periods:

1. Initially, designers compute some prior knowledge to include in policy
2. When deciding its next action, agent does some computation
3. Agent learns from experience to modify its behavior

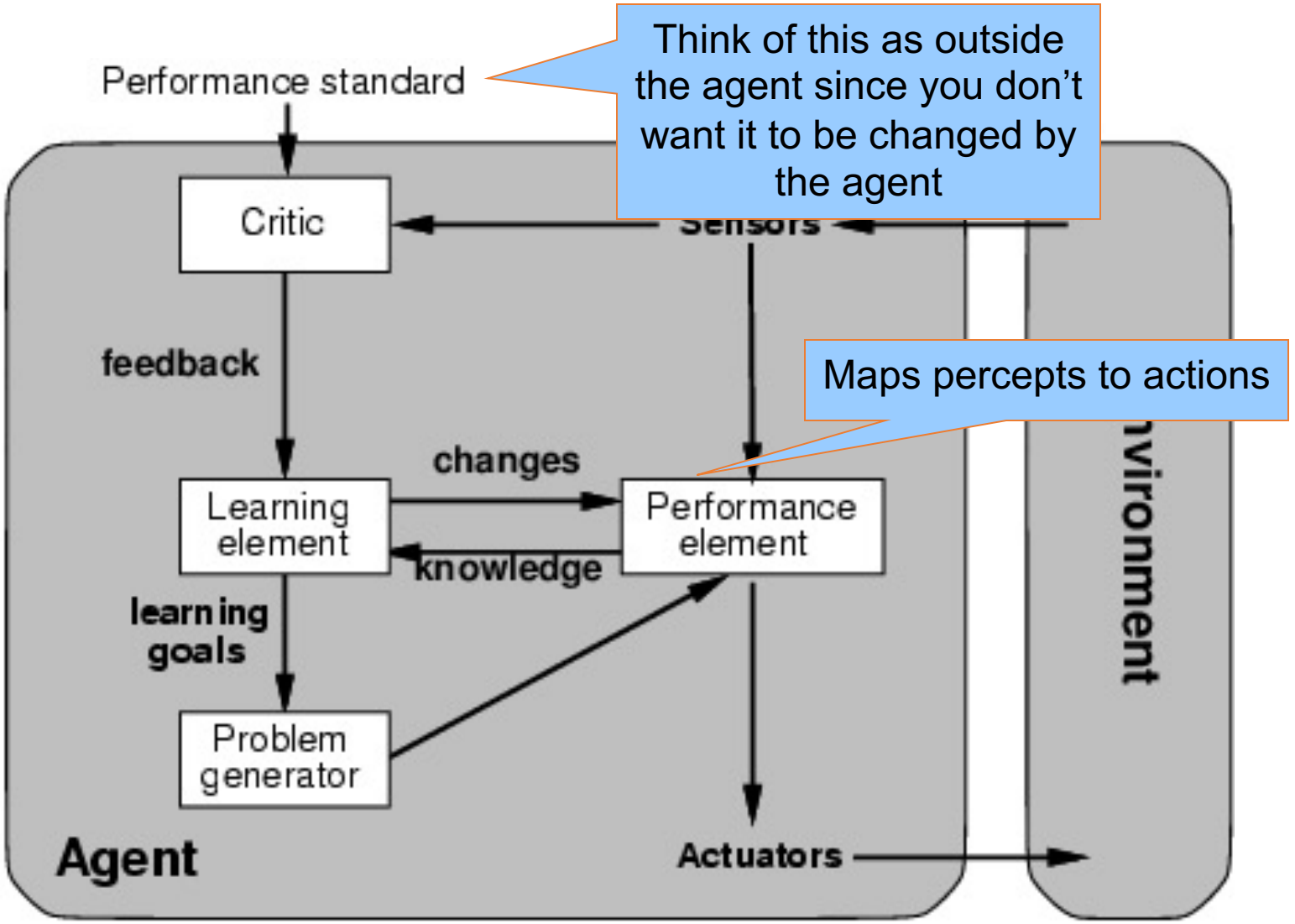
Learn from experience to compensate for partial or incorrect prior knowledge

Example: Spam filter adapting over time. Initially misclassifies some emails, but improves by learning from user corrections ("mark as spam" / "not spam")

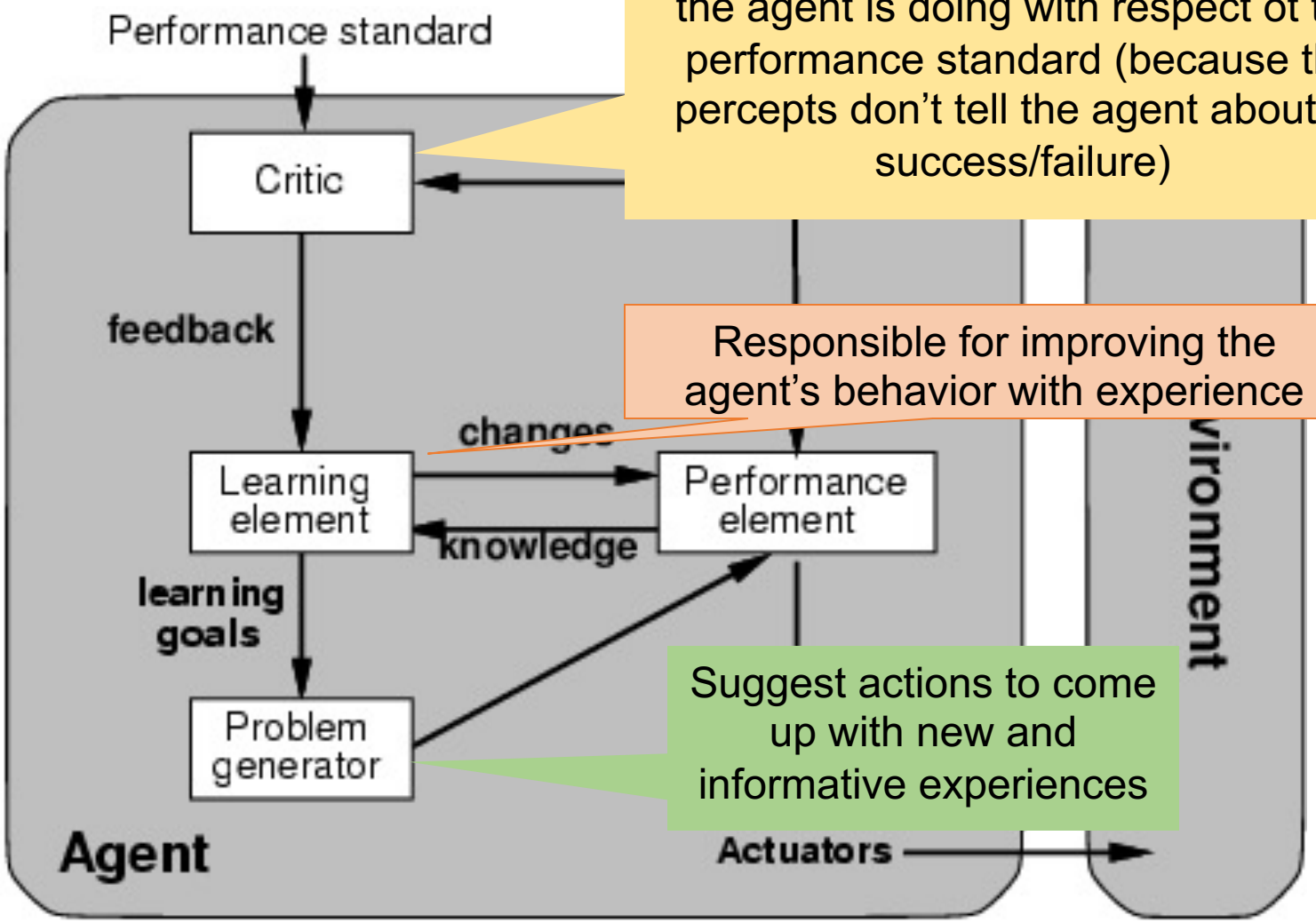
Learning Agents



Learning Agents



Learning Agents



Critic: Tells learning element how well the agent is doing with respect ot the performance standard (because the percepts don't tell the agent about its success/failure)

Responsible for improving the agent's behavior with experience

Suggest actions to come up with new and informative experiences

In-Class Exercise: Designing Agents

Design an intelligent vacuum cleaner for a house.

To make it more interesting, we will analyze design choices for different house settings that will challenge us to think about what type of agent design works best in each setting.

Scenario 1: Small, single-room apartment

Simple reflex agent may be sufficient: If sensor = dirt → suck; else → move.

Scenario 2: Multi-room house

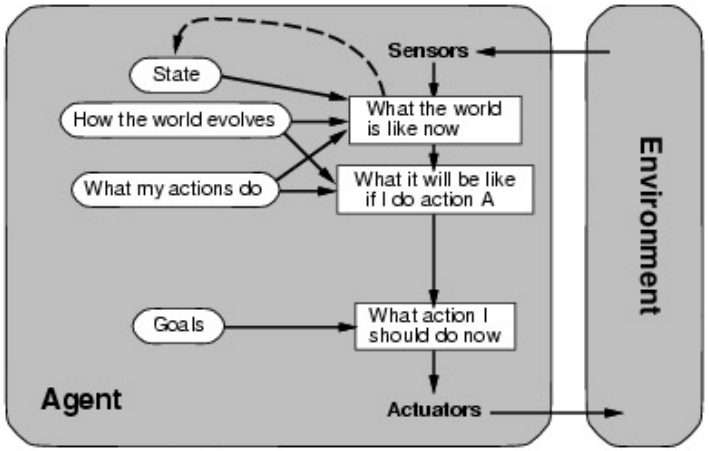
At least need a *model-based agent*, if not more complex designs: must remember which rooms have been cleaned

In-Class Exercise: Designing Agents

Design an intelligent vacuum cleaner for a house.

Scenario 3: Multi-room house with user-specific requests such as ”Clean the kitchen first and then the living room”

Goal-based agents



In-Class Exercise: Designing Agents

Design an intelligent vacuum cleaner for a house.

Scenario 4: Multi-room house and the agent must determine the order of cleaning the rooms

1. Utility-based agents, because it must find the order in which the rooms must be cleaned such that the utility is maximized

2. Learning agent that learns the dirt distribution (and therefore the utility associated with each room)

Designing Modern Agents

Many of the modern AI agents are designed to learn from data, to adapt better to the environment and user

Example:

1. ChatGPT or other LLMs: utility-based, with learned utilities (preferences)
2. Robots in warehouses that perform pickup-and-place tasks: model-based (uses an internal model of the environment) + learning (preferences, utilities, safety constraints)

From Classical AI to Modern Agents

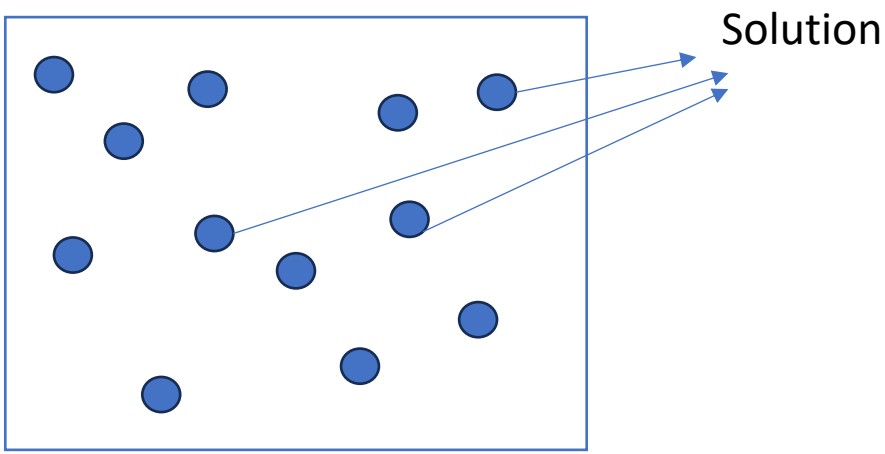
Agent Type	Modern AI Example	Connection
Simple reflex agent	Basic rule-based systems such as thermostat control	Direct condition–action mapping, no memory
Model-based agent	Roomba with mapping, warehouse robots	Maintains internal state of environment to act under partial observability
Goal-based agent	Delivery drones, planning in robotics, navigation apps (Google Maps, Waze)	Chooses actions to achieve explicit goals
Utility-based agent	Self-driving cars, recommendation systems, medical diagnosis AIs	Balances multiple objectives (e.g., safety vs. speed, accuracy vs. coverage)
Learning agent	Spam filters, ChatGPT, AlphaGo, adaptive robots	Improves behavior over time from data and feedback

Planning Agents

- Need to plan a sequence of actions to reach the goal or complete a task
- Except simple reflex agents, all other types of agents perform some level of planning
- Planning can be framed as a search over possible action sequences

Search Problem

- Perform a search in the solution space intelligently
- Evaluate solutions to identify the best



Solution space: set of all possible solutions to the problem

Formalizing the Search Problem

- A finite set of states → set of all decision points
- Initial state
- Goal test
- Successor function: A successor function ***succ(s)*** which takes a state ***s*** as input and returns as output the set of states you can reach from state ***s*** in one step
- Path cost: A cost function ***cost(s,s')*** which returns the non-negative one-step cost of travelling from state ***s*** to ***s'***. The cost function is only defined if ***s'*** is a successor state of ***s***.

Formalizing the Search Problem: Example from Textbook

- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?