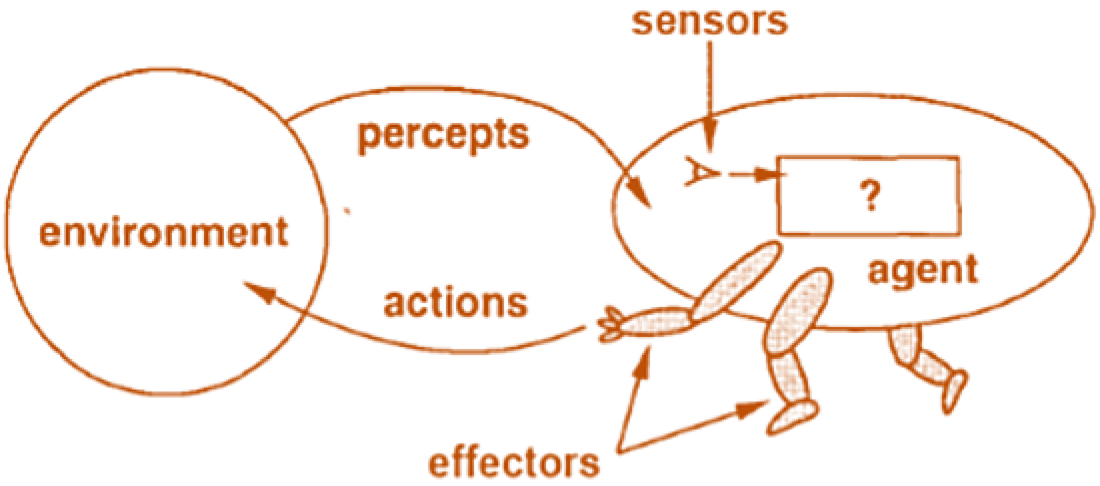
BCSE306

Artificial Intelligence

Module 1: Introduction

Sub topic: Structure of Agent & its type



Dr. Durgesh Kumar

Assistant Professor (Senior)

School of Computer Science and Engineering (SCOPE),

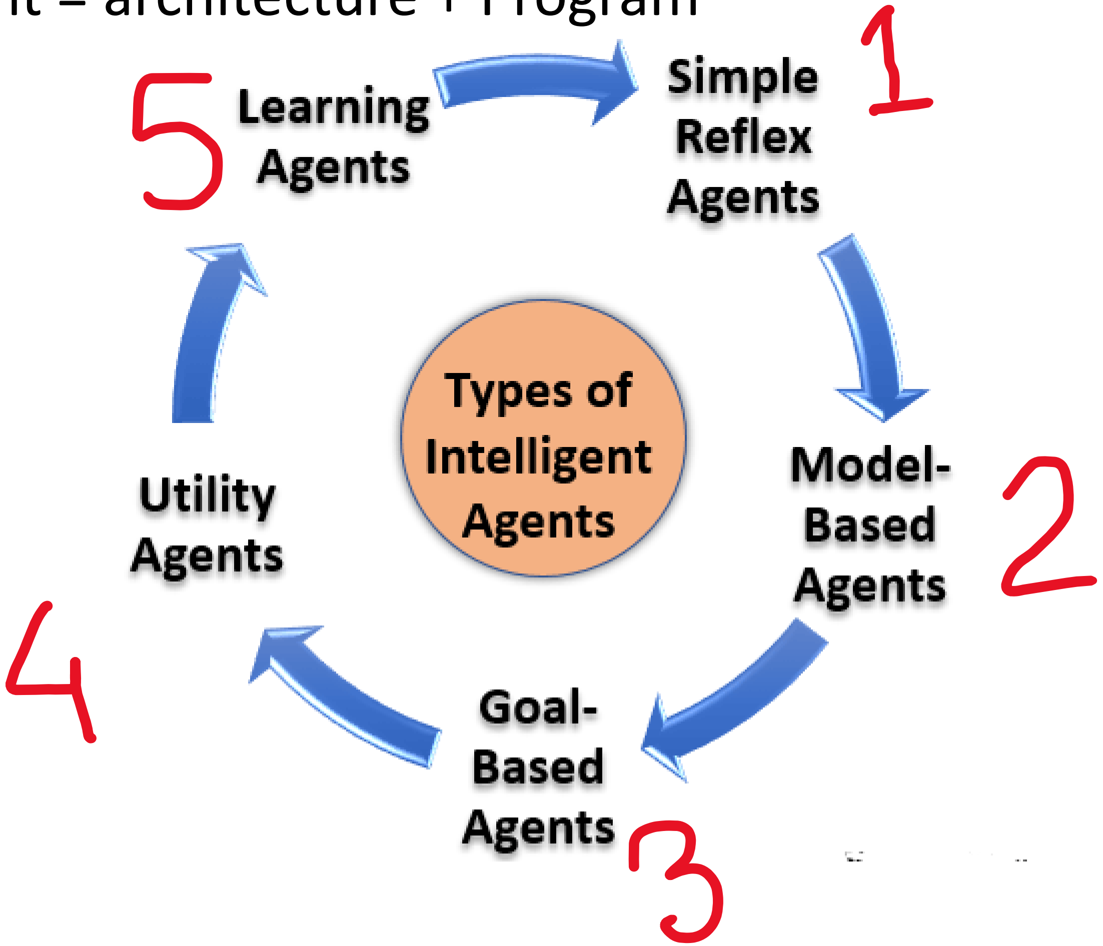
VIT Vellore,

Lecture Outline

- The structure of Agents
  - Agent = architecture + Program
  - Agent Programs
  - Simple Reflex Agents
  - Model-based reflex agents
  - Goal-based agents
  - Utility based agent
  - Learning based agents

Lecture Outline

- The structure of Agents
  - Agent = architecture + Program



Types of Intelligent Agents

1 Simple Reflex Agents

2 Model-Based Agents

3 Goal-Based Agents

4 Utility Agents

5 Learning Agents

Agent Architecture

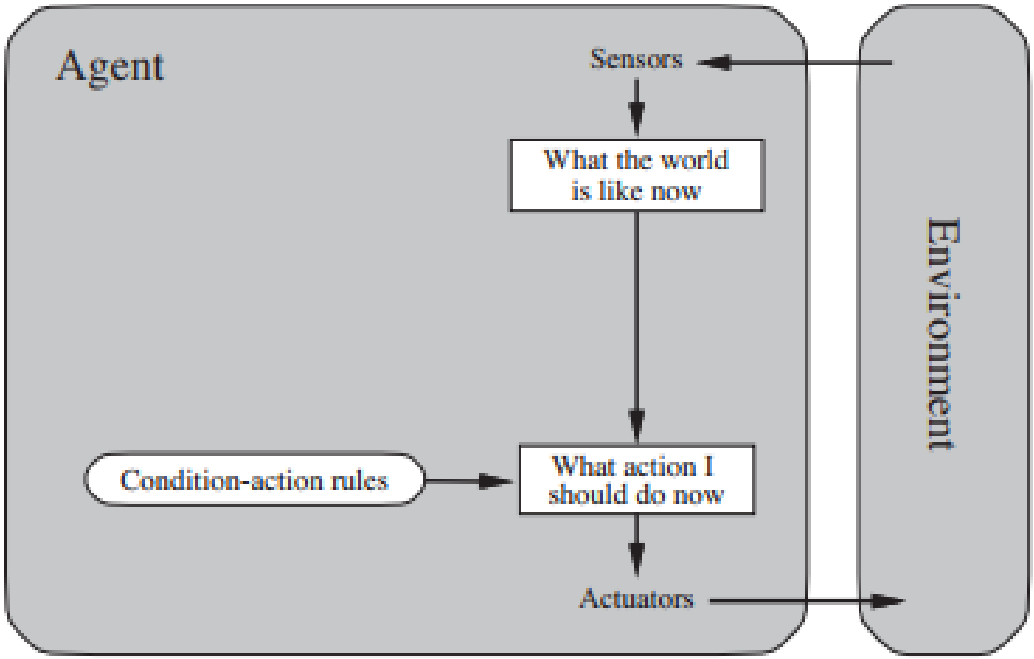
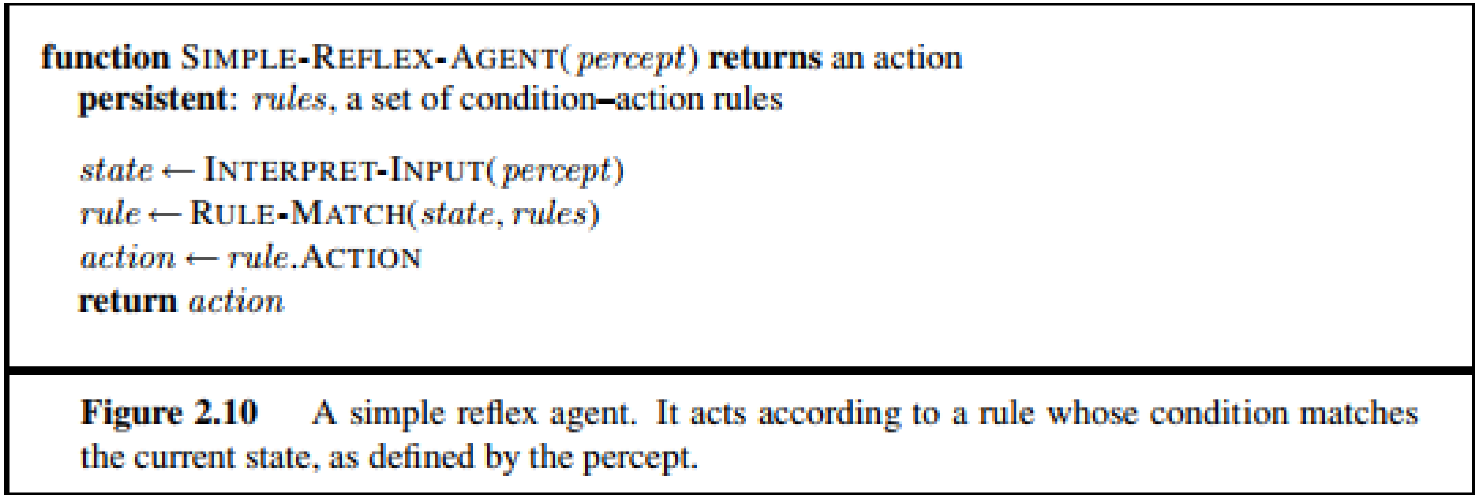


Figure 2.9 Schematic diagram of a simple reflex agent.

Agent Program



```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

Structure of Agent

- Till now, we have talked about **agents** by describing **behavior**—the action that is performed after any given sequence of percepts.
- The job of AI is to design an **agent program** that implement the agent function
  - Mapping from percepts to actions
- Assumption: this program will run on some sort of computing device with physical sensors and actuators (called as architecture).

agent = architecture + program

Agent Program

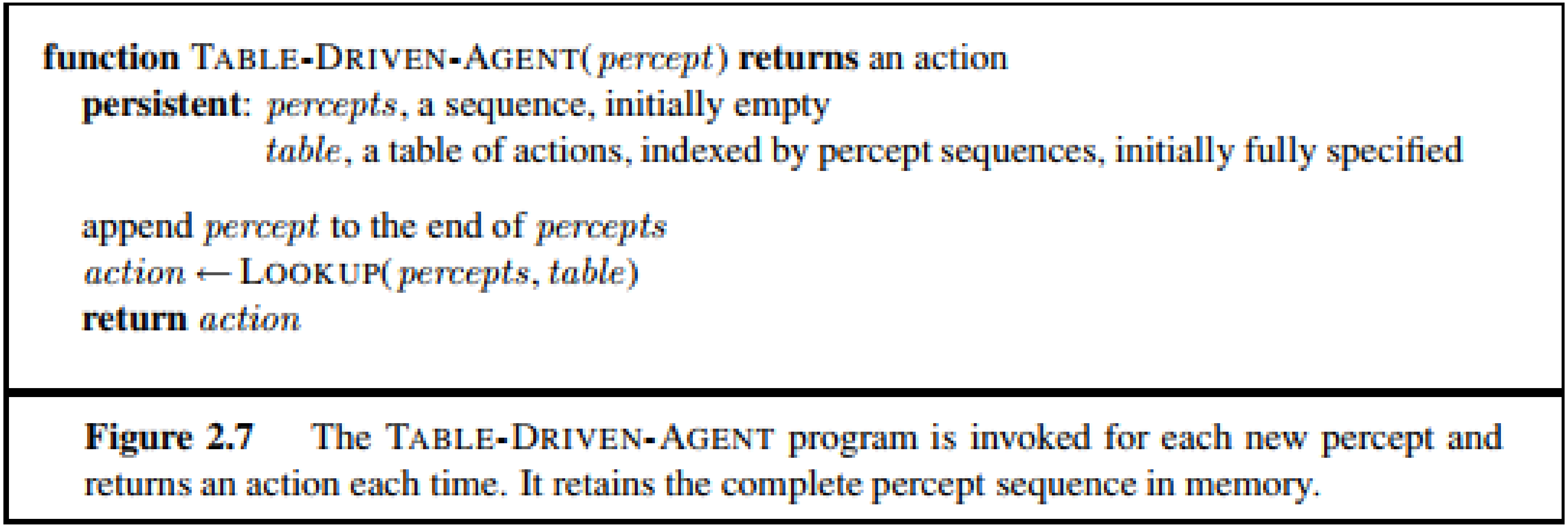
agent program is represented using pseudo code. E.g Table driven Agent.

- Trivial agent program that keeps track of the percept sequence and then uses it to index into a table of action.

- The table represents explicitly the agent function that the agent program embodies.

- In order to build a **rational agent**, the designers must construct a table that contains the appropriate action for every possible percept sequence

Table Driven Agent: Agent Program

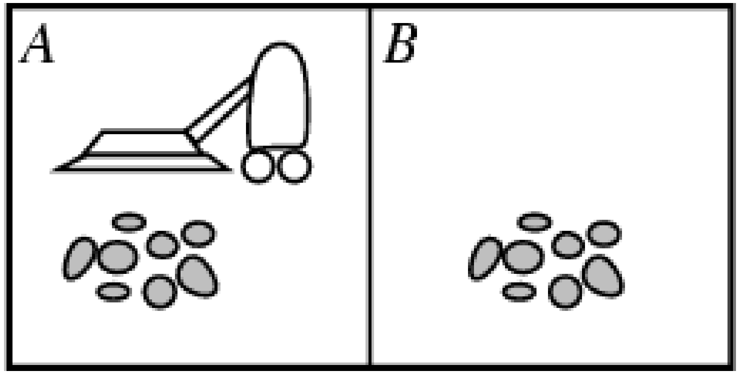


```
function TABLE-DRIVEN-AGENT(percept) returns an action
    persistent: percepts, a sequence, initially empty
                table, a table of actions, indexed by percept sequences, initially fully specified

    append percept to the end of percepts
    action ← LOOKUP(percepts, table)
    return action
```

Figure 2.7 The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.

Table Driven Agent: Example

```
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
```



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

Actions: *Left*, *Right*, *Suck*, *NoOp*

Table Driven Agent: Drawbacks

- Huge table (Example: chess requires $10^{(150)}$ entries in the lookup table)
- Take a long time to build the table
- No autonomy
- Even with learning, need a long time to learn the table entries

Agent Types

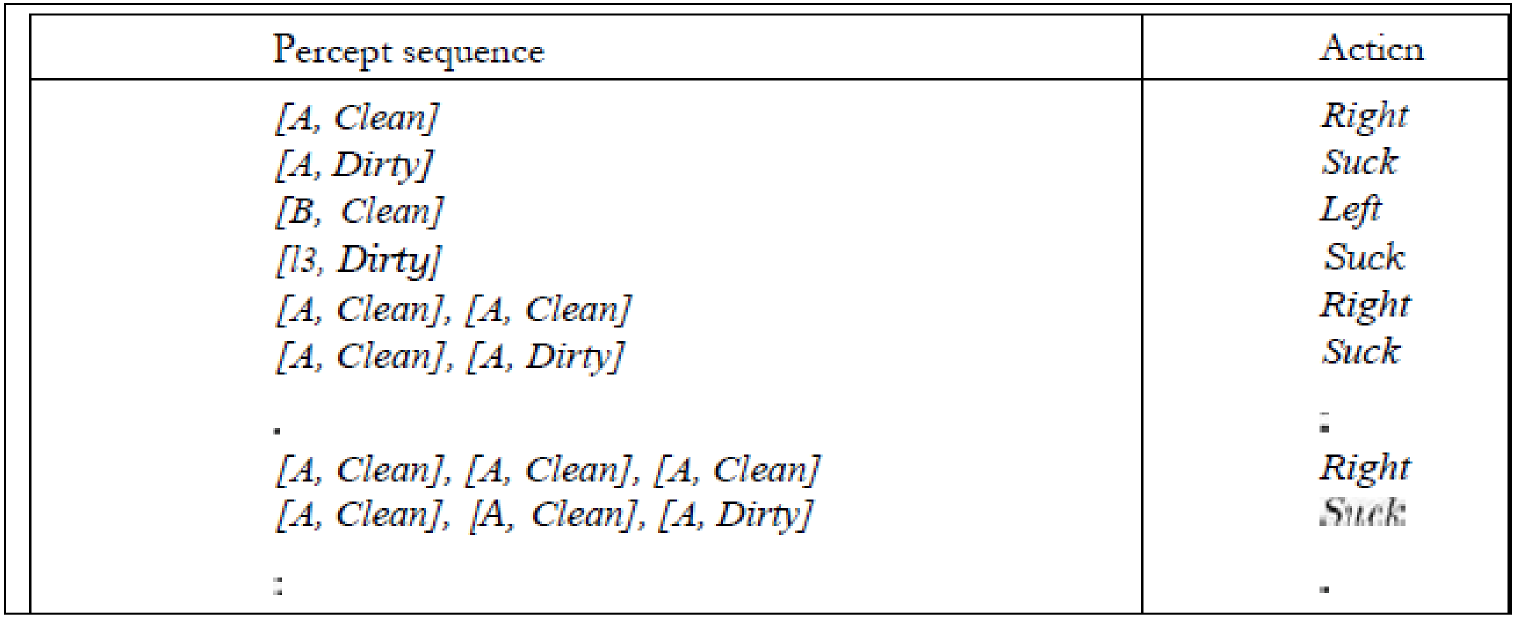
- Five basic types in order of increasing generality

1. Simple reflex agents
2. Model-based reflex agents
3. Goal-based agents
4. Utility-based agents
5. Learning agent

Simple Reflex Agents

- It select actions on the basis of the current percept, ignoring the rest of the percept history

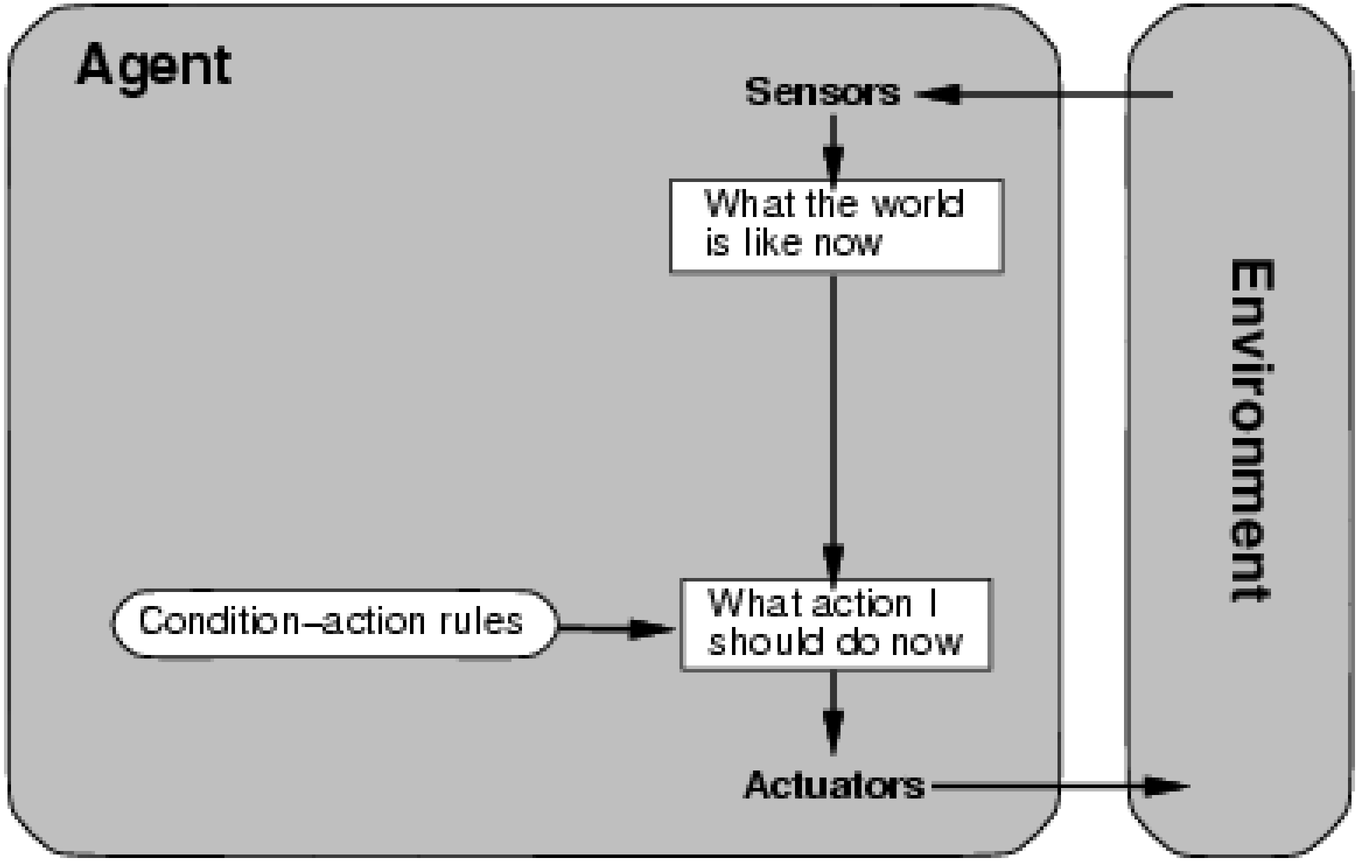
VACUUM-AGENT



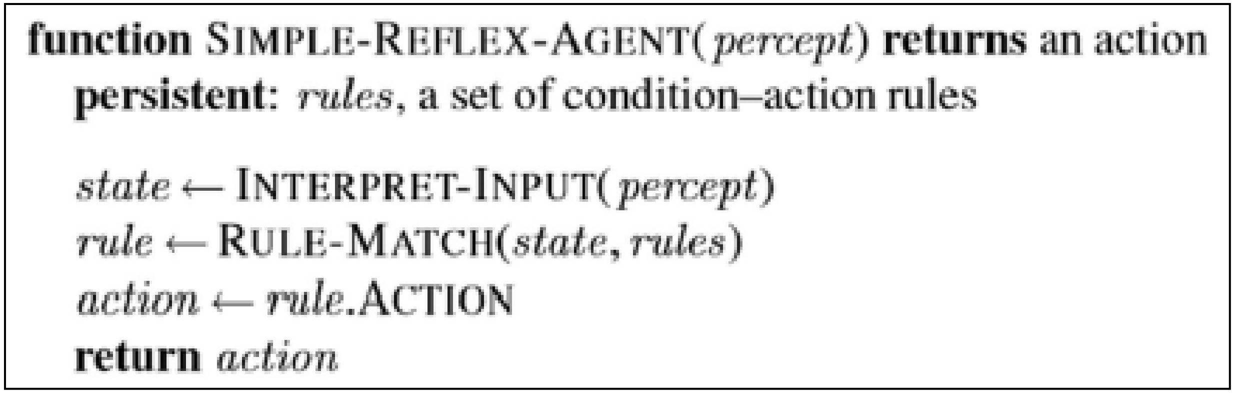
Percept sequence	Acticn
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[l3, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
.	:
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
:	.

Works on condition–action rule !

Simple Reflex Agents



Simple Reflex Agents



```
function SIMPLE-REFLEX-AGENT(percept) returns an action
    persistent: rules, a set of condition–action rules

    state ← INTERPRET-INPUT(percept)
    rule ← RULE-MATCH(state, rules)
    action ← rule.ACTION
    return action
```

- ➢ The INTERPRET-INPUT function generates an abstracted description of the current state from the percept
- ➢ RULE-MATCH function returns the first rule in the set of rules that matches the given state description

Simple Reflex Agents

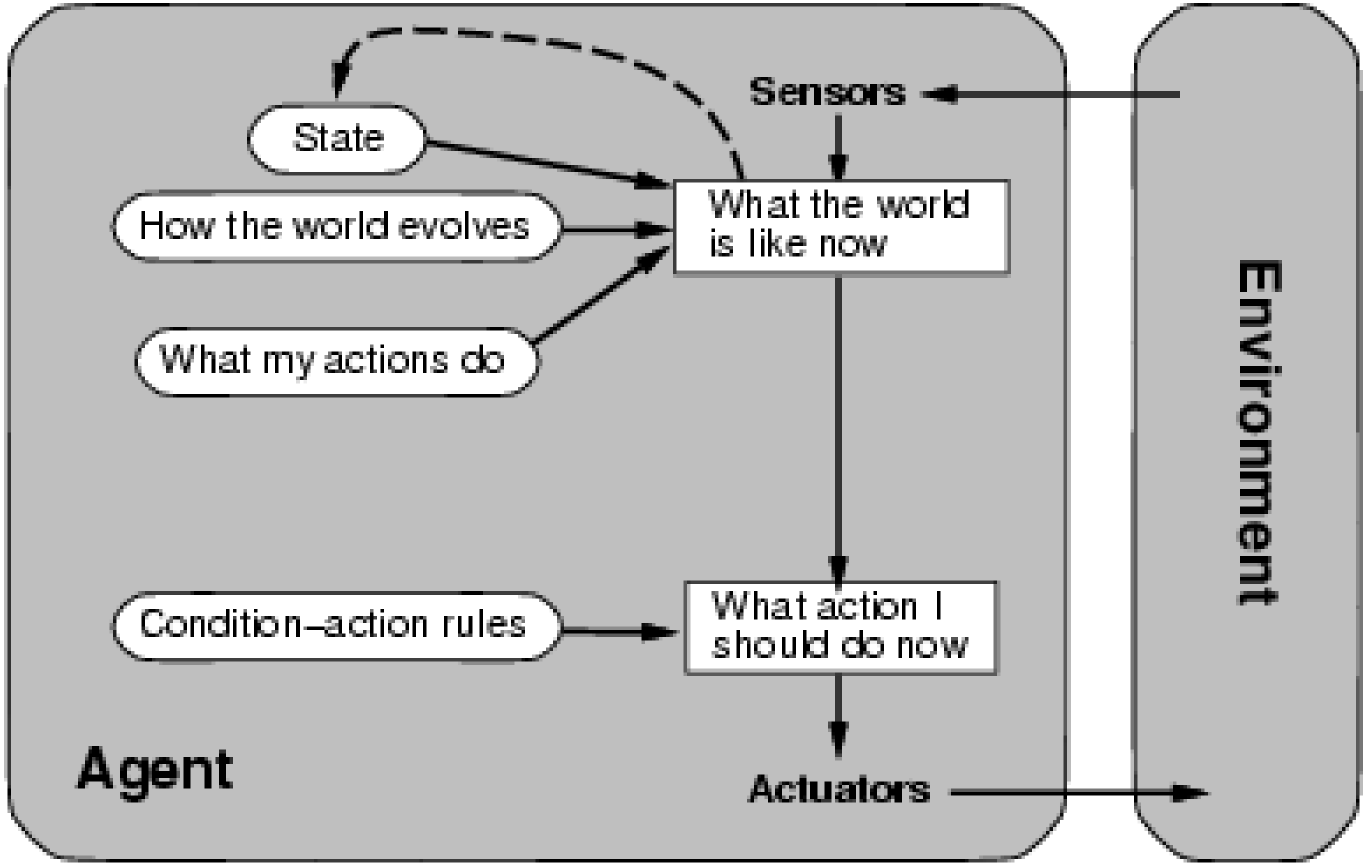
- Advantages:
  - Easy to implement
  - Uses much less memory than the table-driven agent
- Disadvantages:
  - Will only work correctly if the environment is fully observable
  - Infinite loops

Model-based Reflex Agents

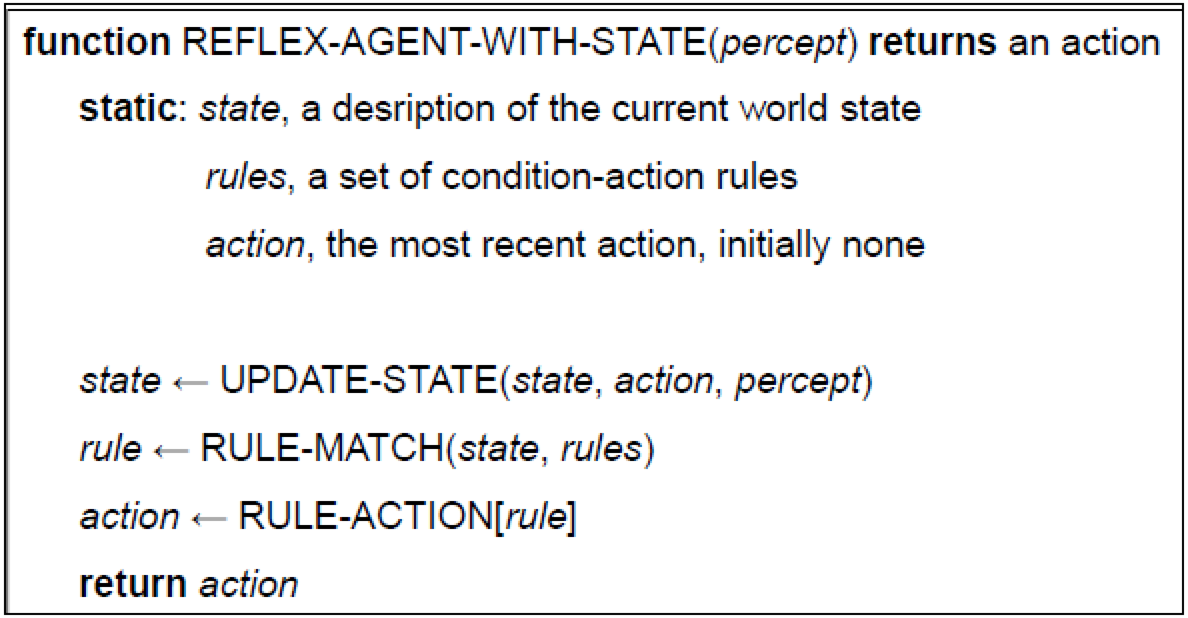
- The agent should maintain some sort of *internal state that depends on the percept history* and thereby *reflects* at least some of the unobserved aspect of the current state

- Updating this internal state information as requires two kinds of knowledge to be encoded in the agent program
  - Information about how the world evolves independently of the agent
  - Information about how the agent's own actions affect the world

Model-based Reflex Agents



Model-based Reflex Agents



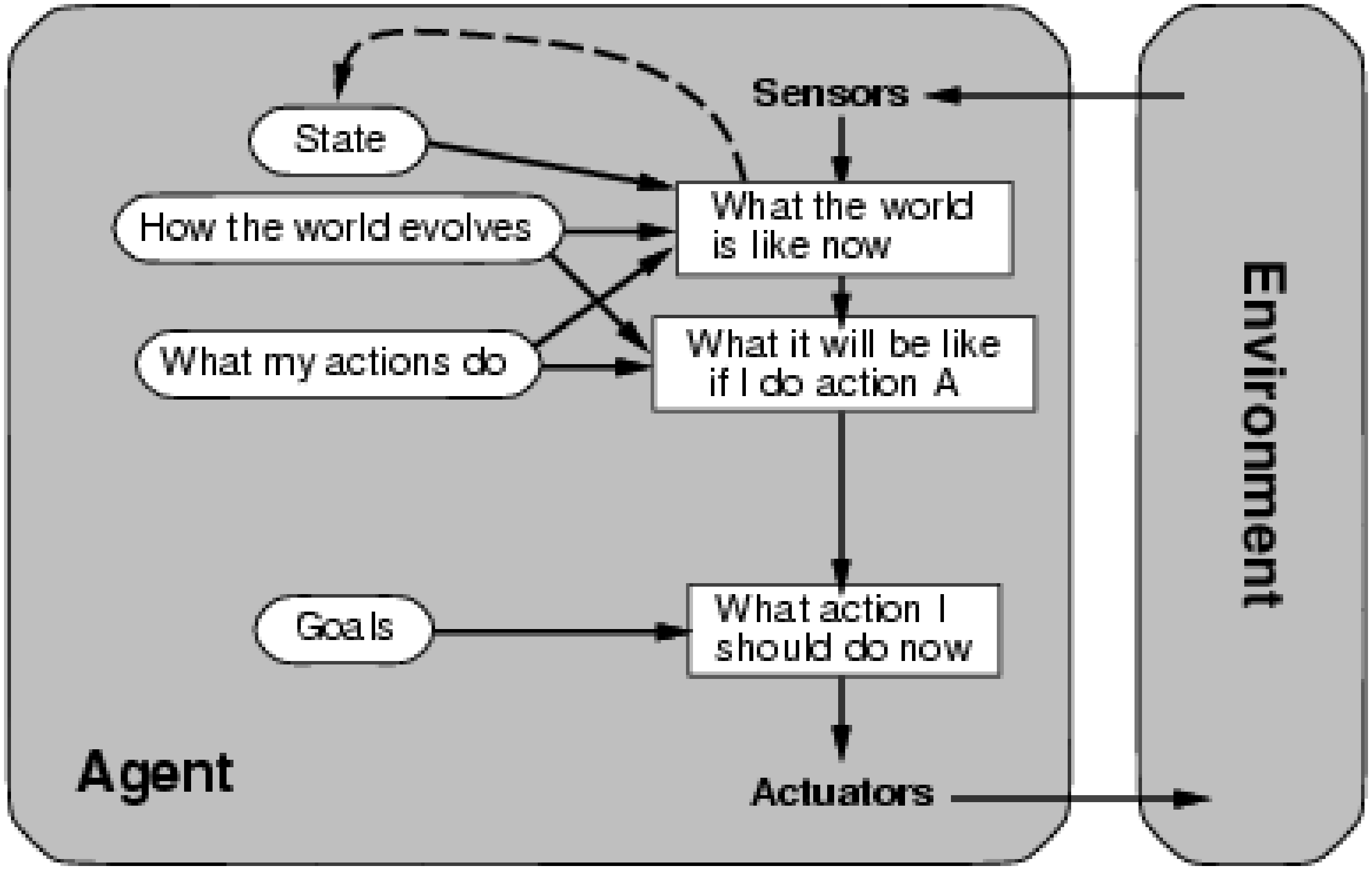
```
function REFLEX-AGENT-WITH-STATE(percept) returns an action
    static: state, a desription of the current world state
            rules, a set of condition-action rules
            action, the most recent action, initially none

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

- The above said state information is implemented in simple Boolean circuits or in complete scientific theories called **model** of the world

Goal-based Agents

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

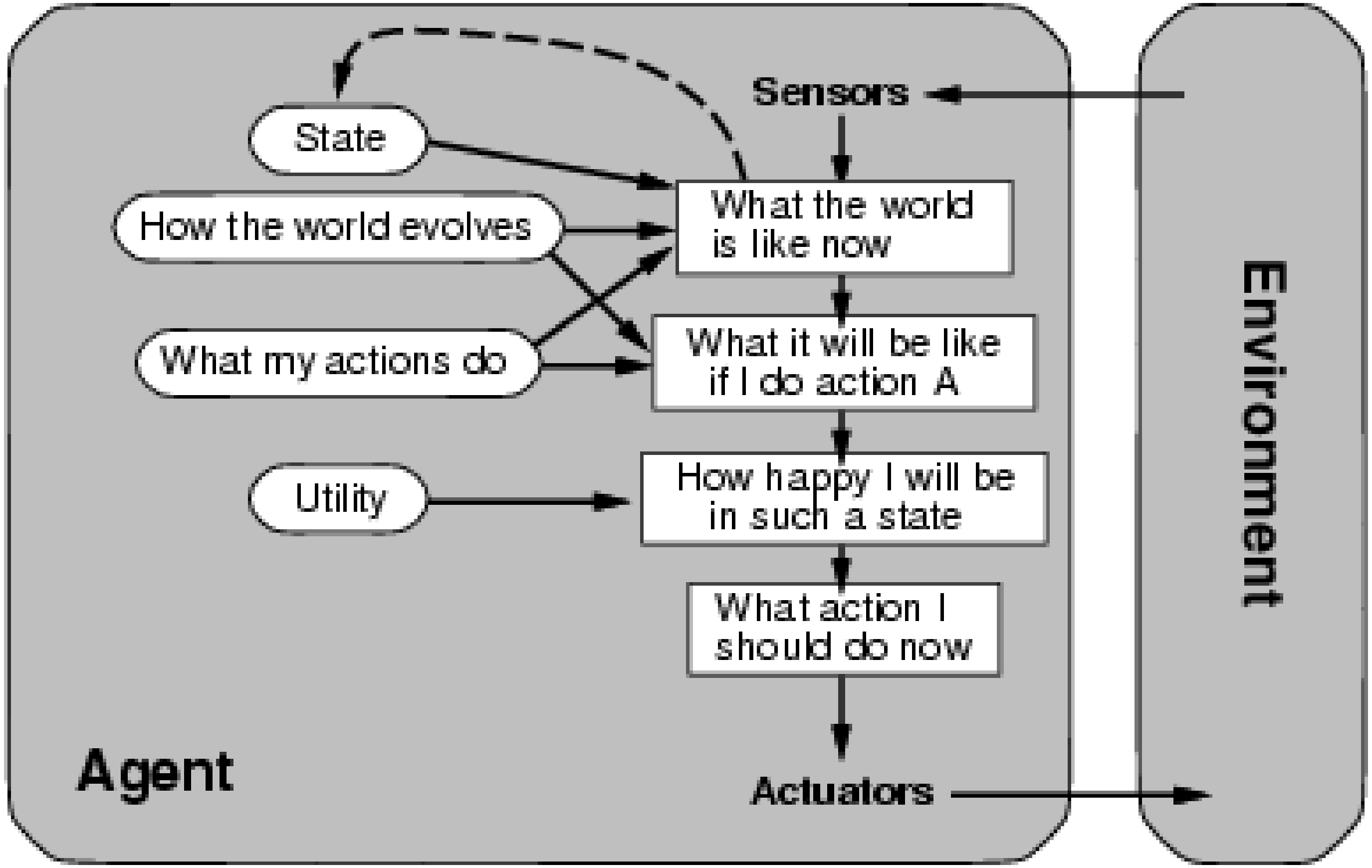
Goal-based Agents


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

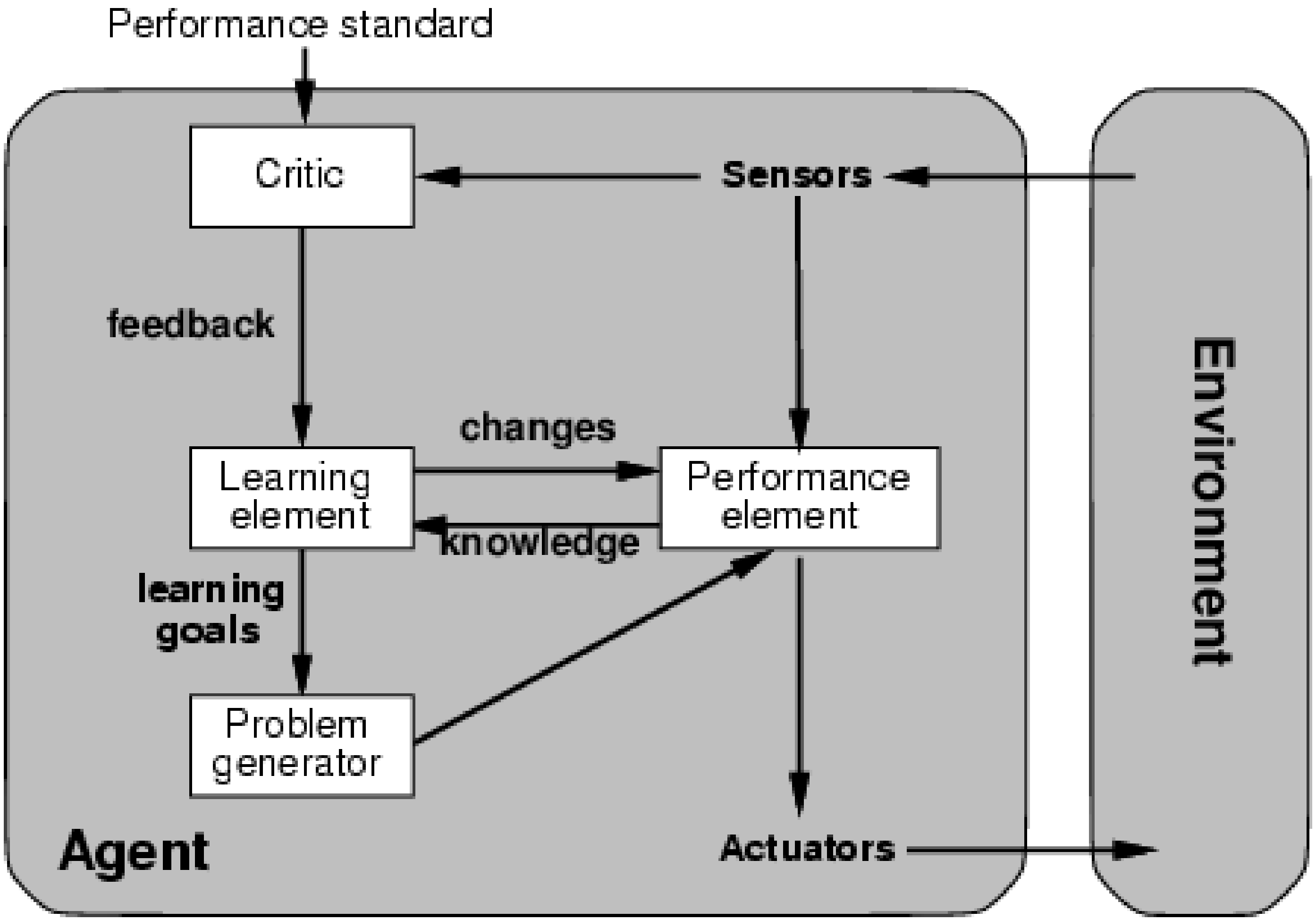
Utility-based Agents

- What if there are many paths to the goal?
- Utility measures which states are preferable to other state
- Maps state to real number (utility)

Utility-based Agents


Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action A
How happy I will be in such a state
What action I should do now
Actuators
Agent
Environment

Learning Agents



Learning Agents

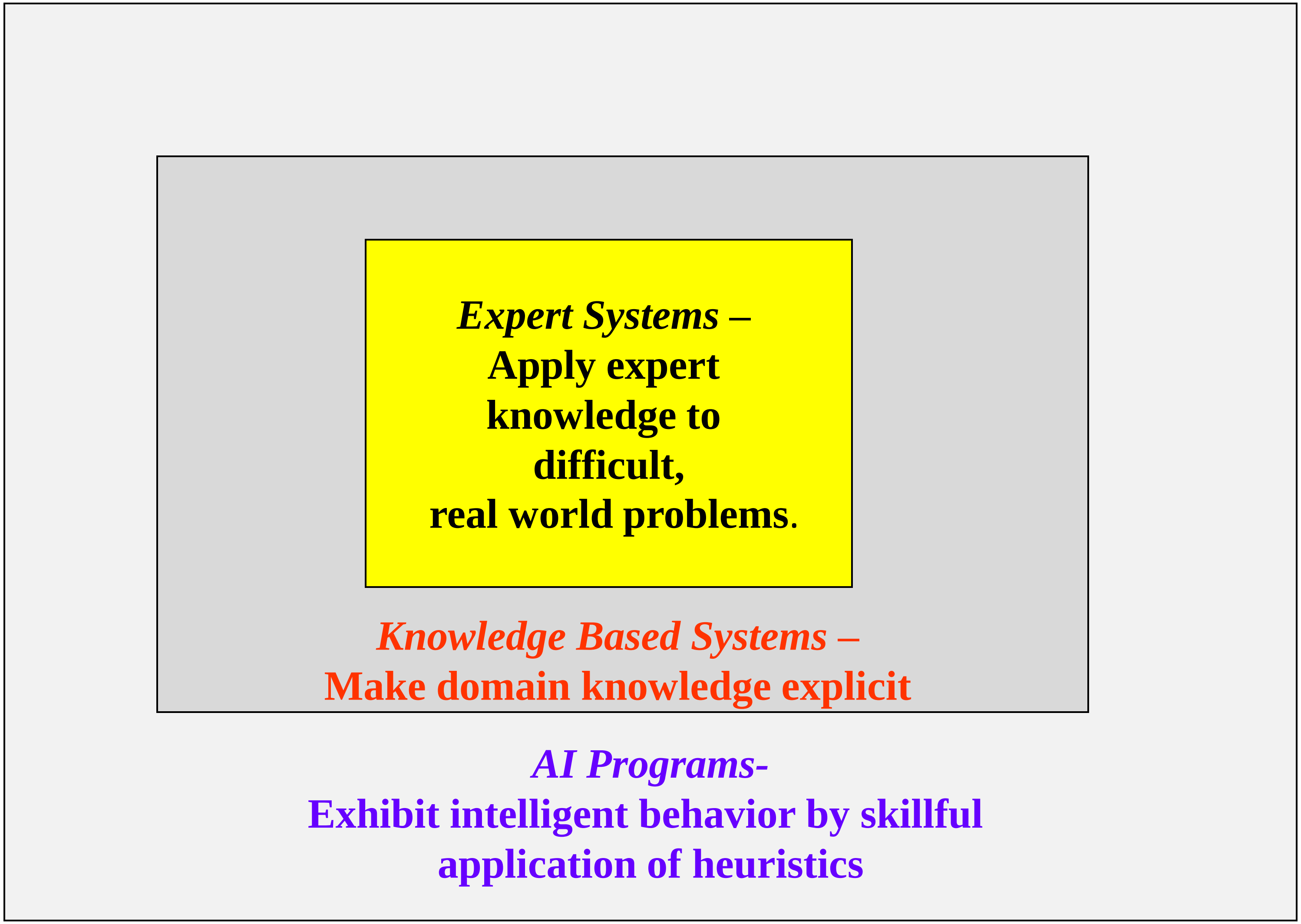
- **Learning element** is responsible for making improvements

- **Performance element** is responsible for selecting external actions.

- **Critic** provides feedback on how the agent is doing and determines how the performance element should be modified to do better in the future.

- **Problem generator** is responsible for suggesting actions that will lead to new and informative experiences.

AI based Systems



***Expert Systems* –**
Apply expert knowledge to difficult, real world problems.

***Knowledge Based Systems* –**
Make domain knowledge explicit

AI Programs-
Exhibit intelligent behavior by skillful application of heuristics

Project Work

- **Sample projects / ideas :**

1. http://web.stanford.edu/class/cs221/2017/project-list.html
2. https://www.cse.iitb.ac.in/~ananddhoot/projects.html
3. http://www.cs.cornell.edu/courses/cs478/2001sp/mllinks/interesting_ai_demos_and_project.htm
4. http://web.cs.iastate.edu/~cs572/projects.html
5. https://web.stanford.edu/class/cs221/2018/project-list.html
6. http://cs229.stanford.edu/projects2011.html
7. web.stanford.edu/class/cs224w/projects.html

- Note:
  - Plagiarism is not permitted, kindly don't copy someone's project and pass it off as yours. If found you will get 0 marks for the whole project.

Research Work/Survey

- Areas:
  - Geo Sensing/ Geo-forecasting/ Geo-Science
  - Image Reorganization
  - NLP
  - Vehicular Networks
  - Intelligent Transportation Systems
  - Autonomous Vehicles
  - Data Science (Text Analysis, Sentiment Analysis Clustering)
  - Bio Technology / Medical Bio Engineering/ Bio-Informatics
  - Robotics
  - Gaming
  - Pervasive Computing
  - …