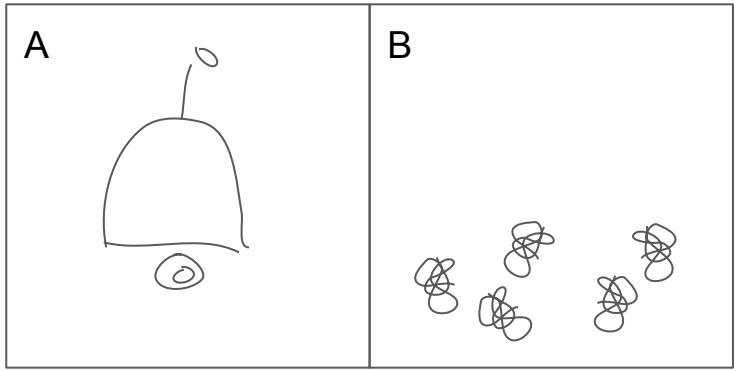
Agents and State Spaces

CS 580
Intro to Artificial Intelligence

Vacuum World Code

```
while(True):
    Clean()
    MoveLeft()
    Clean()
    MoveRight()
```

- Why might this be **bad**?
- Why might this be **good**?
- Can we find a **general** structure that will work for any environment/task?



Need a formal way to describe the agent's capabilities, performance, and environment

Rational Agents

Performance Measure
Evaluates the sequence of configurations of the environment

Environment
Domain the agent will be operating within

Actions
How the agent can modify its environment

Sensors
What information the agent receives about the environment

Definition: “A **rational agent** chooses whichever action maximizes the expected value of the performance measure given the sequence of observed perceptions”

Performance measures for Vacuum World

- One point per clean square
- One point per clean square in time T
- One point per clean square minus one per move

Under which of these was our agent behaving rationally?

Environment Types

Many specific environments can be loosely clustered based on certain shared characteristics.

For many of these, the boundaries are not always clear.

Different characteristics may require different agent designs

Additional type: **Unknown** vs **Known**
What the **designer** of the agent (you) knows about the environment

- **Observable** (vs. Fully/Partially)
  Can the agent sense all the relevant aspects of the environment directly
- **Single-agent** (vs. Multi)
  Is there more than one **agent** acting in the environment at a time?
- **Deterministic** (vs. Stochastic)
  Do actions always result in the same outcomes?
- **Episodic** (vs. Sequential)
  Do past actions affect future performance?
- **Static** (vs. Dynamic)
  Can the environment change while the agent is thinking?
- **Discrete** (vs. continuous)
  State, time, sensors, and actions

Table Driven Agent

```
def table_driven_agent(percept):
    percept_history.append(percept)
    action =
action_table[percept_history]
    return action
```

Achieves what we want, just fill in **`action_table`**, so why aren't we done?

1. **How** do we fill in the table?
2. How **big** does the table have to be?

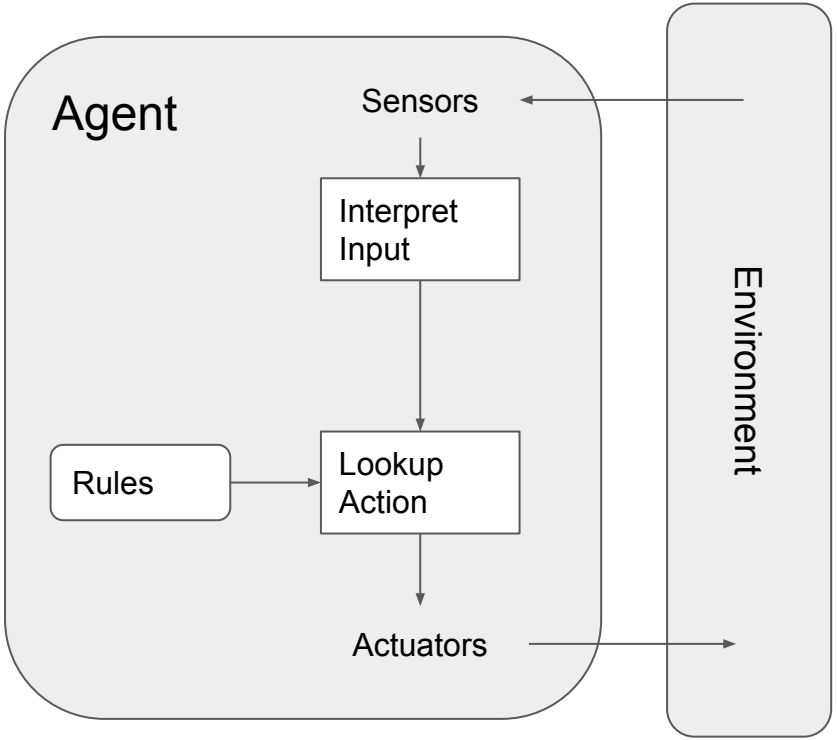
This solution would be fine if it weren't ridiculously inefficient

The rest of this course will be about techniques and algorithms for efficiently computing what the rational action to take is in a given situation

Most of these require that we make certain simplifying assumptions about the environment

Simple Reflex Agent

```
def simple_reflex_agent(percept):
    state = interpret_input(percept)
    action = rules[state]
    return action
```



Simple Reflex Agent

How is this any different from the table driven agent?

State: a description of the **current** configuration of the environment

Reflex or **reactive** agents have simple rules for how the world looks **right now** (no history)

Simplifying assumption: the past doesn't matter

- Episodic vs Sequential

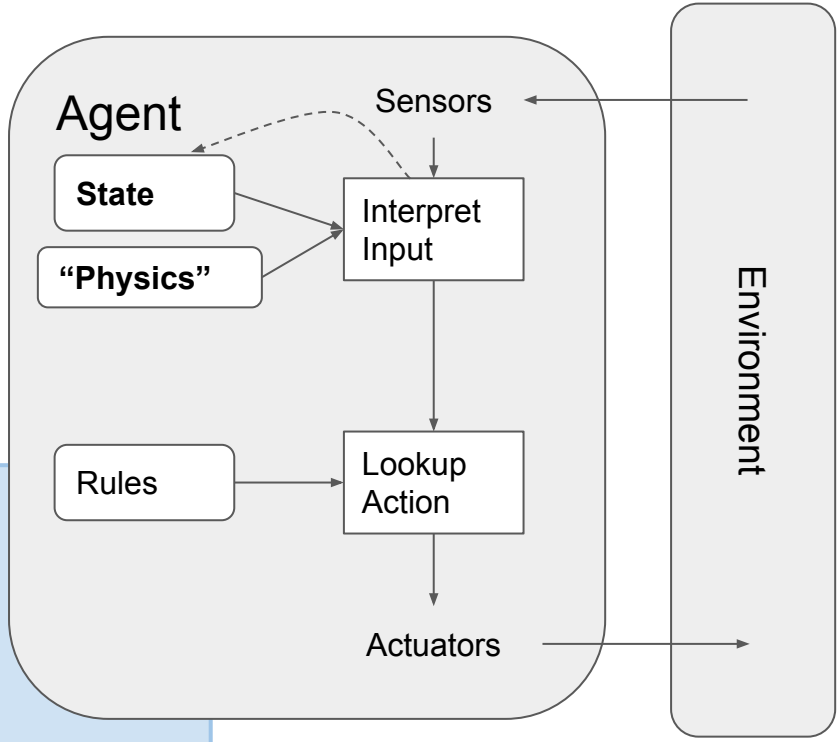
```
def simple_reflex_agent(percept):
    state = interpret_input(percept)
    action = rules[state]
    return action
```

```
vacuum_rules = {
    (A,Clean):"Go_Right",
    (A,Dirty):"Clean",
    (B,Clean):"Go_Left",
    (B,Dirty):"Clean"
}
```

Model-based Reflex Agent

Handling **partial-observability**:

- Keep track of the parts of the environment that are not directly sensed
- Needs domain knowledge about how the state changes over time, and in response to agent action




```
def __init__(self):
     self.is_clean['A'] = 'Unknown'
     self.is_clean['B'] = 'Unknown'
def model_based_vacuum(self,percept):
     if percept.clean==False:
          Clean()
     self.is_clean[percept.loc] = 'Clean'
     if percept.loc=='A' and self.is_clean['B']!='Clean':
          MoveRight()
     elif percept.loc=='B' and self.is_clean['A']!='Clean':
          MoveLeft()
```

Goal-based Agent

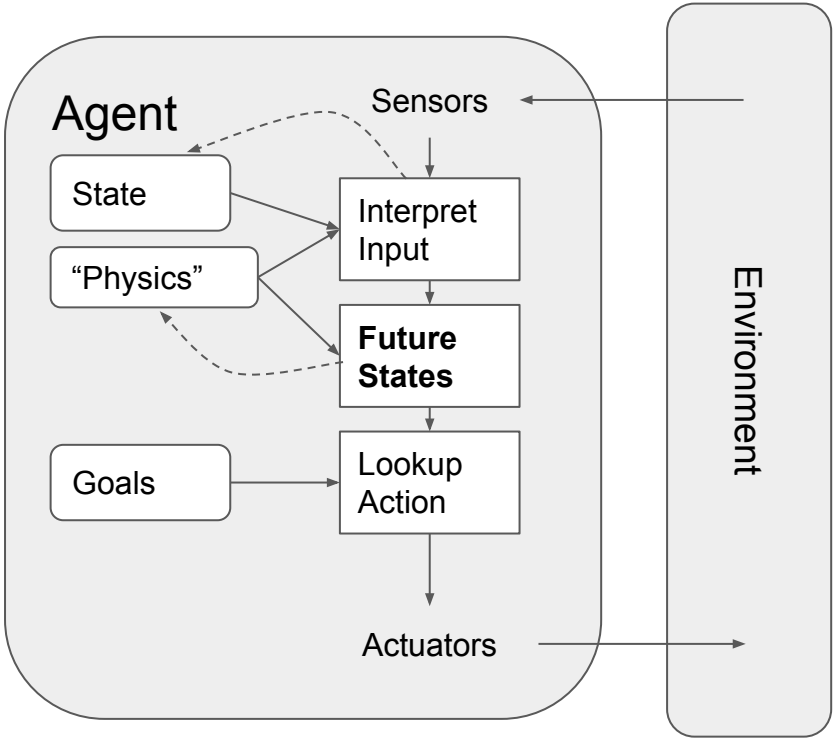
Handling **sequential** environments

- Plan more than one step into the future

Complications

- Dynamic environments
- Continuous environments
- Stochastic environments

Search is one way of implementing a goal-based agent, which we will spend the next few weeks investigating

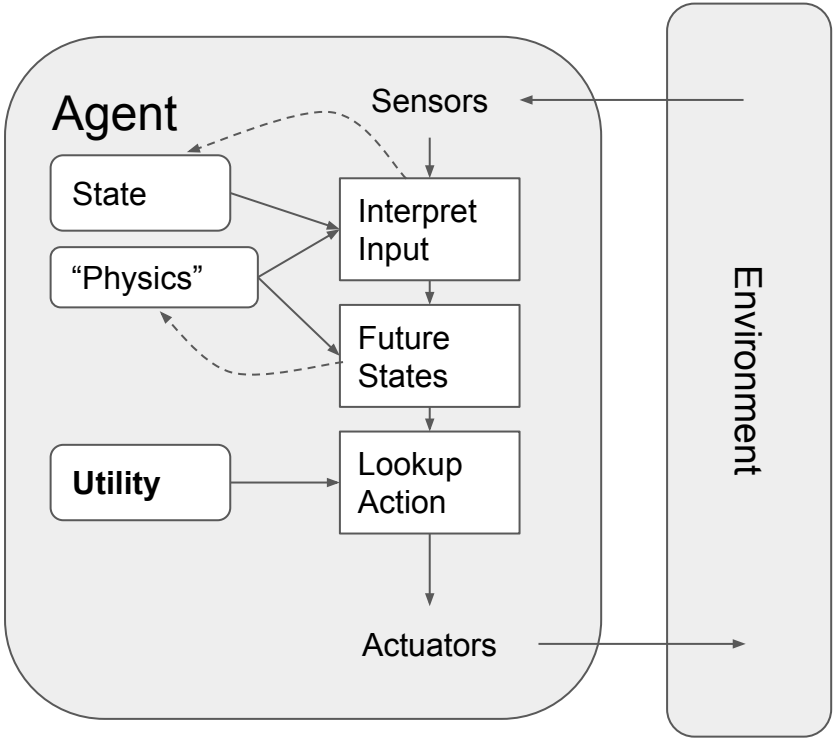


Utility-based Agent

Handling **stochastic** environments

- Use probabilities to quantify expected “goodness” of future states

Markov Decision Processes are a formal mathematical framework for analyzing expected future “goodness” that we will talk about after search



Learning Agent

Handling **unknown** environments

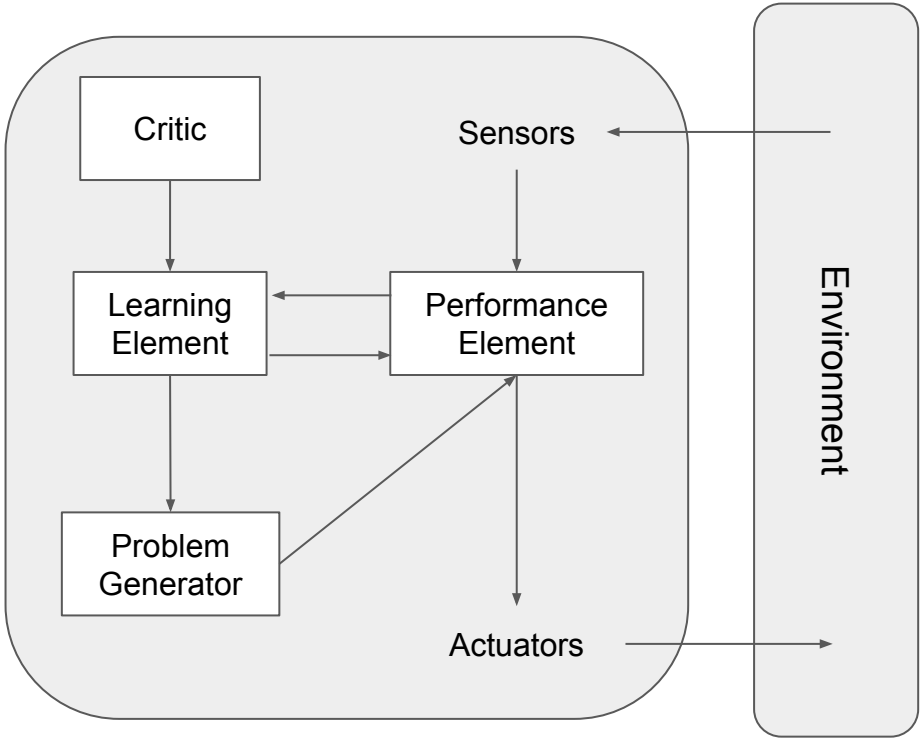
Performance Element: evaluates states/actions, provides information about environment

Learning Element: improves internal model

Problem Generator: suggest actions to improve learning

Critic: evaluates overall performance of agent

After MDPs we'll talk about **Reinforcement Learning** and later in the semester the broader field of **Machine Learning**

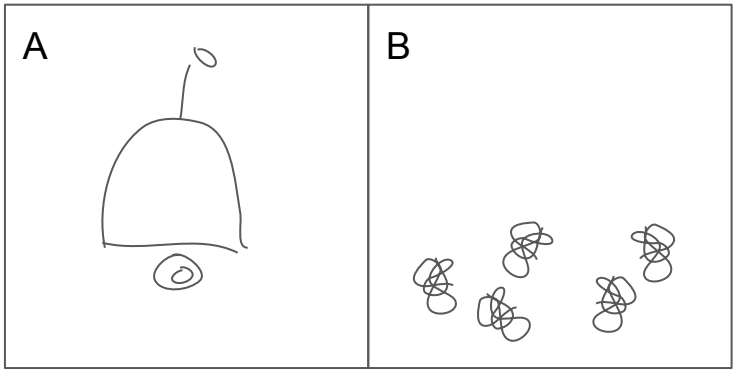


States

State: an encoding of all the relevant information about the current configuration of the environment

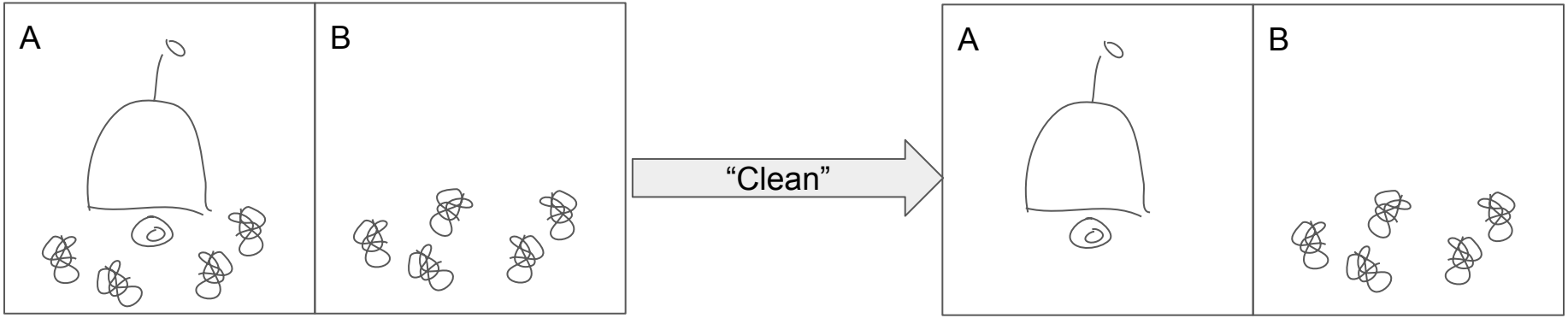
The “physics” of the environment dictate how the state changes over time or in response to the agent’s actions

States are different from **percepts** because they can encode things which are not directly observable



Percept: (A, Clean)
State: {
‘robot-loc’:’A’,
‘A-clean’:True,
‘B-clean’:False
}

Transforming State with Actions



{‘loc’:’A’, ’A-clean’:False, ‘B-clean’:False}

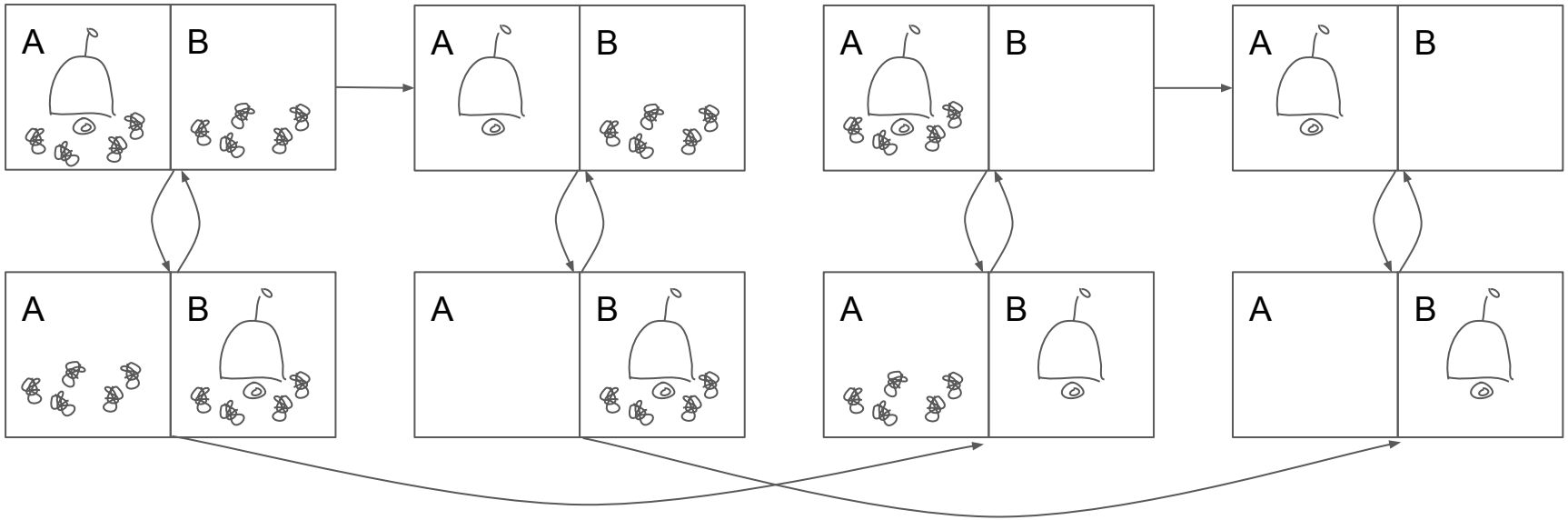
{‘loc’:’A’, ’A-clean’:True, ‘B-clean’:False}

Actions can be thought of as linking together “nearby” states, which leads to two natural representations of the **state space**: the set of all possible states

State Space - Graph form

For problems with a **finite** number of states and a **finite** number of actions, it's often possible to explicitly represent the entire state space in memory as a **graph**

Compare this with the "history of perceptions" from earlier. What is different?



Search based Agent - Preview

1. Formulate problem as a graph where **nodes** represent states and **edges** represent actions
2. Identify the **current state** from most recent perception
3. Search the graph for a path from the current state to a **goal state**
4. Return the path as a **sequence of actions**

Summary and preview

Wrapping up

- A **rational** agent acts to maximize expected performance given experience
- **Environments** can be roughly categorized along five dimensions
- There are several different **agent architectures** for dealing with the challenges presented by these environments

For next time

- Using **search agents** to solve problems