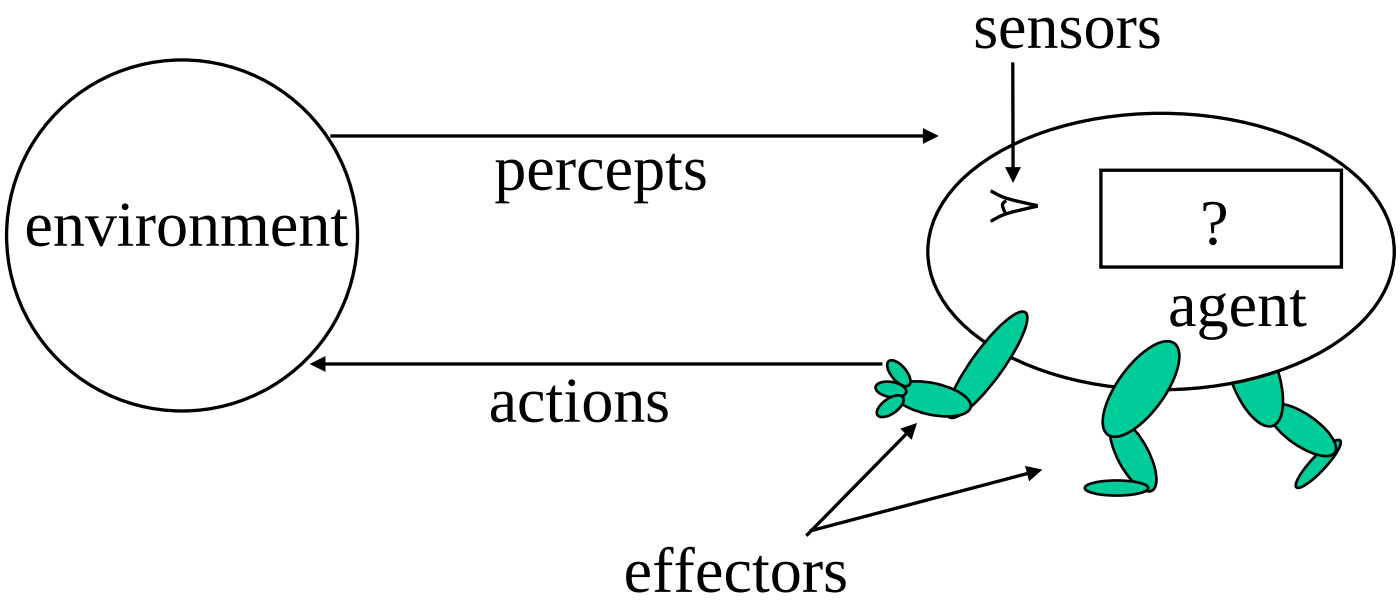
# AI as the Design of Agents

**= a unifying view for the bag of techniques that AI encompasses**

Tuomas Sandholm
Carnegie Mellon University
Computer Science Department

# An agent and its environment



# How to design an intelligent agent?

- **Definition**: An *agent* perceives its environment via sensors and acts in that environment with its effectors.

  Hence, an agent gets percepts one at a time, and maps this percept sequence to actions (one action at a time)

- **Properties**:
  - Autonomous
  - Interacts with other agents plus the environment
  - Reactive to the environment
  - Pro-active (goal-directed)

# Examples of agents in different types of applications

| Agent type | Percepts | Actions | Goals | Environment |
|---|---|---|---|---|
| Medical diagnosis system | Symptoms, findings, patient's answers | Questions, tests, treatments | Healthy patients, minimize costs | Patient, hospital |
| Satellite image analysis system | Pixels of varying intensity, color | Print a categorization of scene | Correct categorization | Images from orbiting satellite |
| Part-picking robot | Pixels of varying intensity | Pick up parts and sort into bins | Place parts in correct bins | Conveyor belts with parts |
| Refinery controller | Temperature, pressure readings | Open, close valves; adjust temperature | Maximize purity, yield, safety | Refinery |
| Interactive English tutor | Typed words | Print exercises, suggestions, corrections | Maximize student's score on test | Set of students |

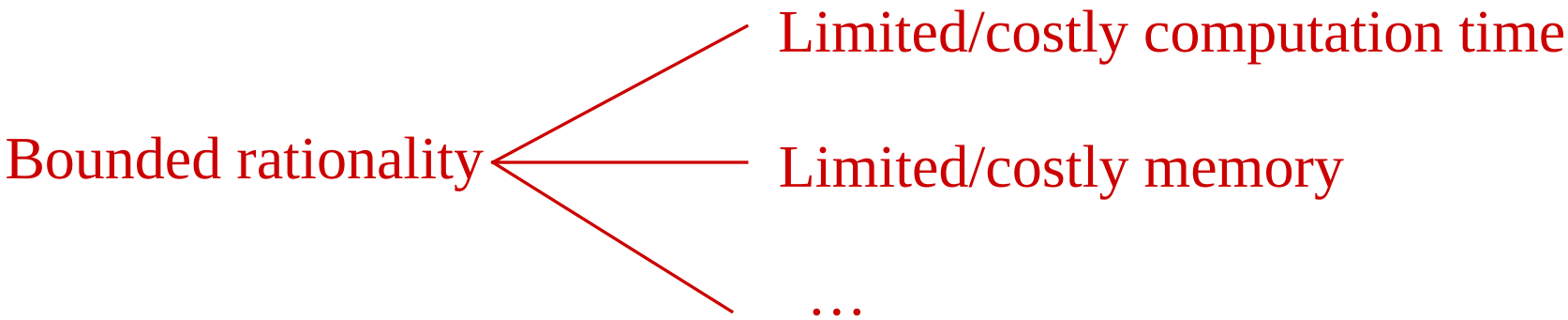
# Definition of ideal rational agent

**Ideal Rational Agent**: For each possible percept sequence, such an agent does whatever action is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

What do you think? Is this an acceptable definition?

Not looking left when crossing the street: If I don't see a car coming from the left, it is rational to cross the street?

No. Should also consider taking *information gathering actions*.



# Agent's strategy

- Agent's *strategy* is a mapping from percept sequence to action
- How to encode an agent's strategy?
  - Long list of what should be done for each possible percept sequence
  - vs. shorter specification (e.g. algorithm)

# WARNING:
# Might not get what you ask for in the performance measure

- Cleaning robot
  - Pick up as much trash as possible
- Vehicle route optimization
  - Maximize utilizations => Driving fully loaded
  - Capitalizing on oddities in tariff list => Renegotiation
  - Don’t include solution method in the criterion

# agent = architecture + **program**

This course concentrates on the program

Physical agents vs. software agents
(software agents = softbots)

# Skeleton agent

**function** SKELETON-AGENT (*percept*) **returns** action
  static: *memory*, the agent's memory of the world

  *memory* ← UPDATE-MEMORY(*memory*,*percept*)
  *action* ← CHOOSE-BEST-ACTION(*memory*)
  *memory* ← UPDATE-MEMORY(*memory*, *action*)
  **return** *action*

On each invocation, the agent's memory is updated to reflect the new percept, the best action is chosen, and the fact that the action was taken is also stored in the memory. The memory persists from one invocation to the next.

Input = Percept, not history

NOTE: Performance measure is not part of the agent

# Examples of how the agent function can be implemented

More sophisticated

1. Table-driven agent
2. Simple reflex agent
3. Reflex agent with internal state
4. Agent with explicit goals
5. Utility-based agent

# 1. Table-driven agent

**function** TABLE-DRIVEN-AGENT (percept) **returns** action
  static: *percepts*, a sequence, initially empty
    *table*, a table, indexed by percept sequences, initially fully specified

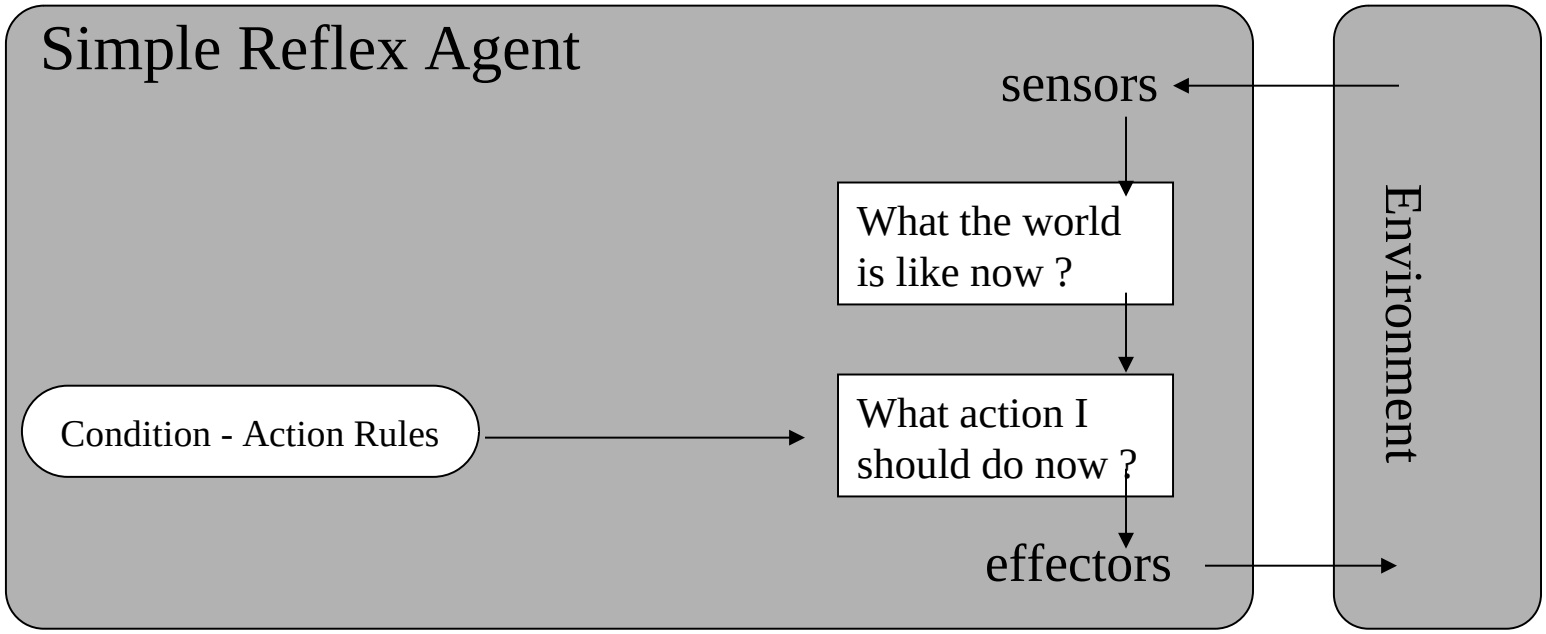
  append *percept* to the end of *percepts*
  *action* ← LOOKUP(*percepts*, *table*)
  **return** *action*

An agent based on a prespecified lookup table. It keeps track of percept sequence and just looks up the best action

- **Problems**
  - Huge number of possible percepts (consider an automated taxi with a camera as the sensor) => lookup table would be huge
  - Takes long time to build the table
  - Not adaptive to changes in the environment; requires entire table to be updated if changes occur

# 2. Simple reflex agent

- Differs from the lookup table based agent in that the condition action rule (that determines the action) is already higher-level interpretation of the percepts
  - Percepts could be e.g. the pixels on the camera of the automated taxi




**function** SIMPLE-REFLEX-AGENT(*percept*) **returns** action
static: *rules*, a set of condition-action rules

*state* ← INTERPRET-INPUT (*percept*)
*rule* ← RULE-MATCH (*state*,*rules*)
*action* ← RULE-ACTION [*rule*]
**return** *action*

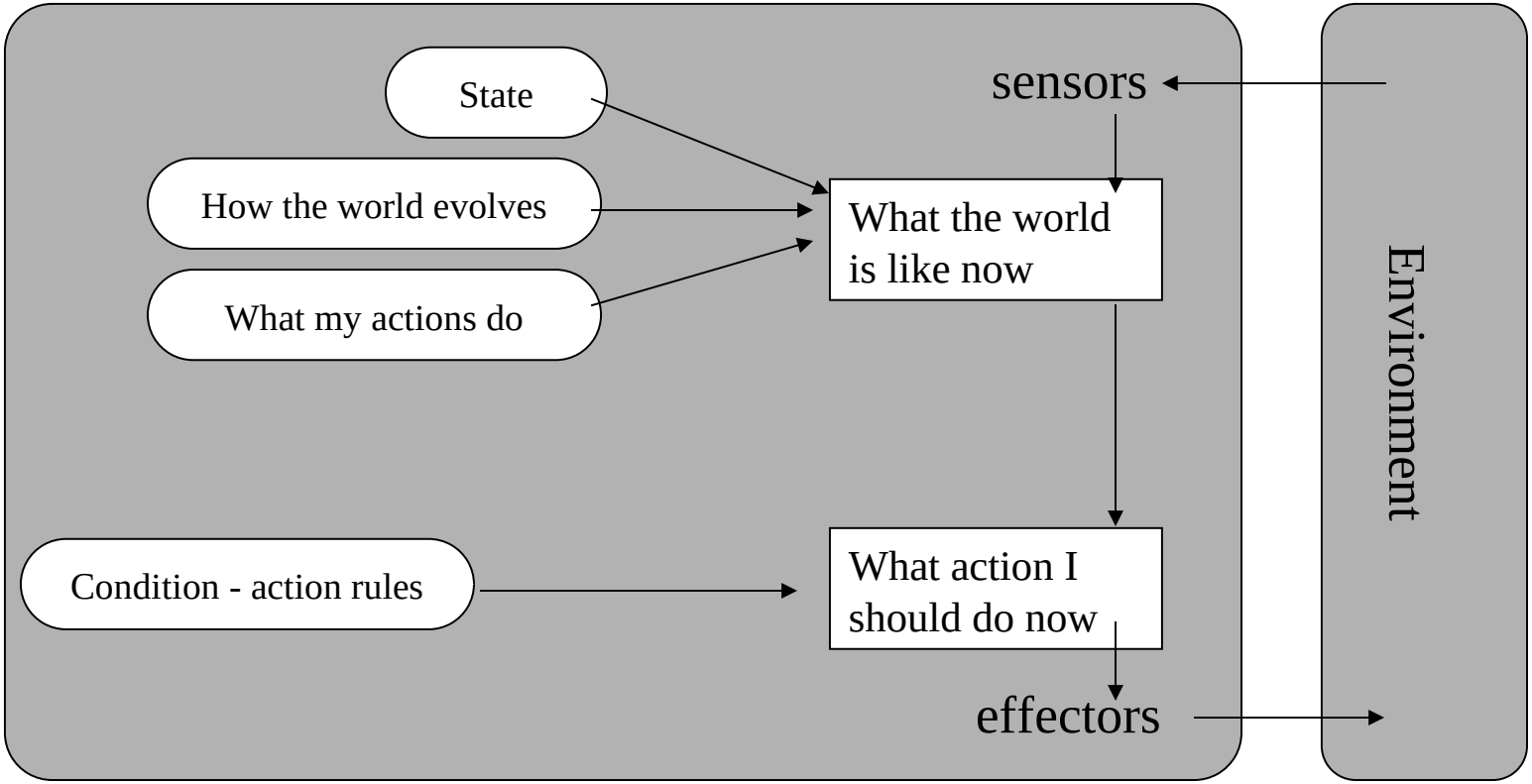
First match.
No further matches sought.
Only one level of deduction.

A simple reflex agent works by finding a rule whose condition matches the current situation (as defined by the percept) and then doing the action associated with that rule.

# Simple reflex agent…

- **Table lookup of condition-action pairs defining all possible condition-action rules necessary to interact in an environment**
  - e.g. if car-in-front-is-breaking then initiate breaking
- **Problems**
  - Table is still too big to generate and to store (e.g. taxi)
  - Takes long time to build the table
  - No knowledge of non-perceptual parts of the current state
  - Not adaptive to changes in the environment; requires entire table to be updated if changes occur
  - Looping: Can’t make actions conditional

# 3. Reflex agent *with internal state*



# Reflex agent with internal state …

**function** REFLEX-AGENT-WITH-STATE (*percept*) **returns** action
  static: *state*, a description of the current world state
    *rules*, a set of condition-action rules

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

A reflex agent with internal state works by finding a rule whose condition matches the current situation (as defined by the percept and the stored internal state) and then doing the action associated with that rule.

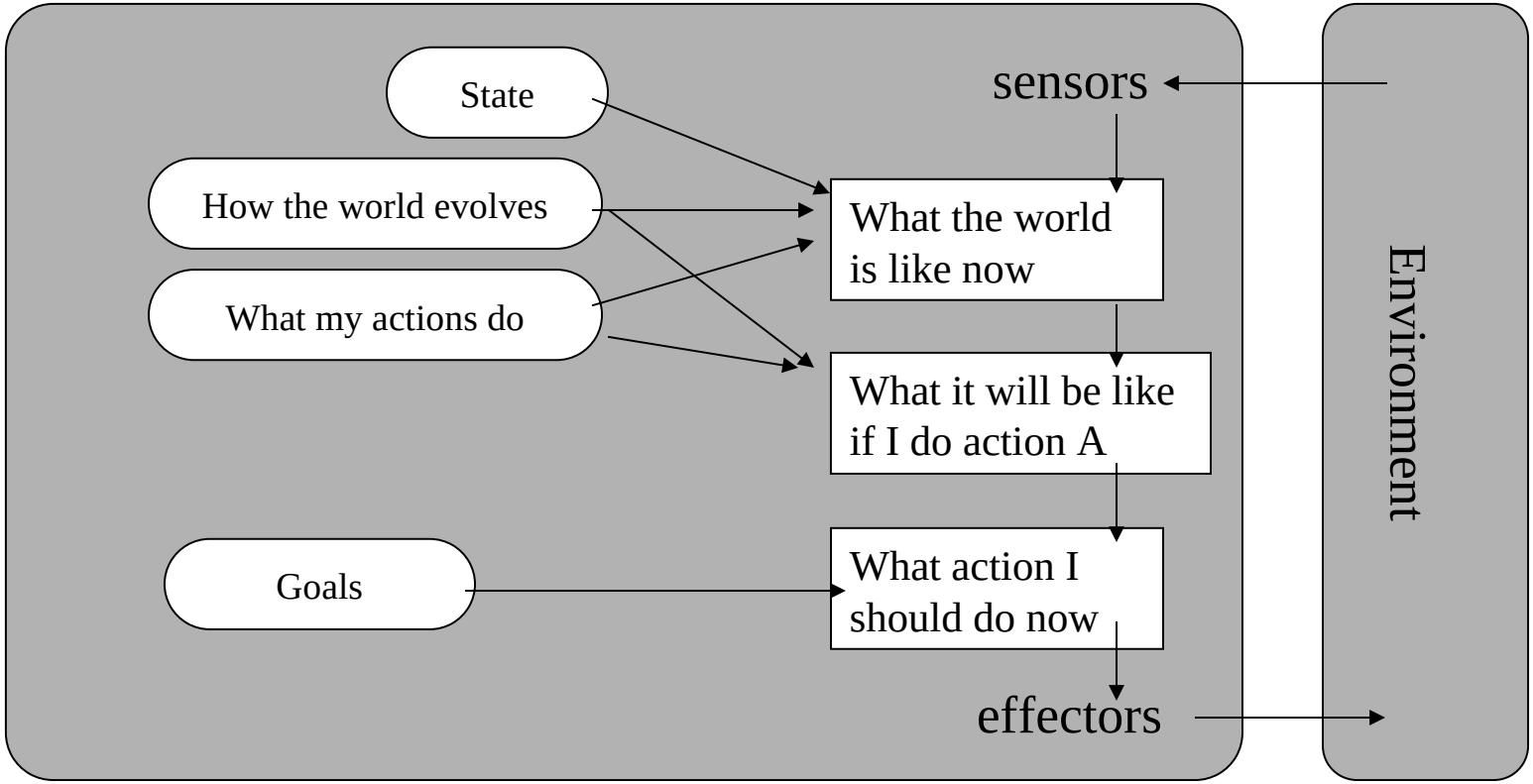
# Reflex agent with internal state …

- Encode “internal state of the world to remember the past as contained in earlier percepts

- Needed because sensors do no usually give the entire state of the world at each input, so perception of the environment is captured over time. “State” used to encode different “world states” that generate the same immediate percept

- Requires ability to represent change in the world with/without the agent
  - one possibility is to represent just the latest state, but then cannot reason about hypothetical courses of action

# 4. Agent with explicit goals



# Agent with explicit goals …

- Choose actions so as to achieve a (given or computed) goal = a description of desirable situations. e.g. where the taxi wants to go

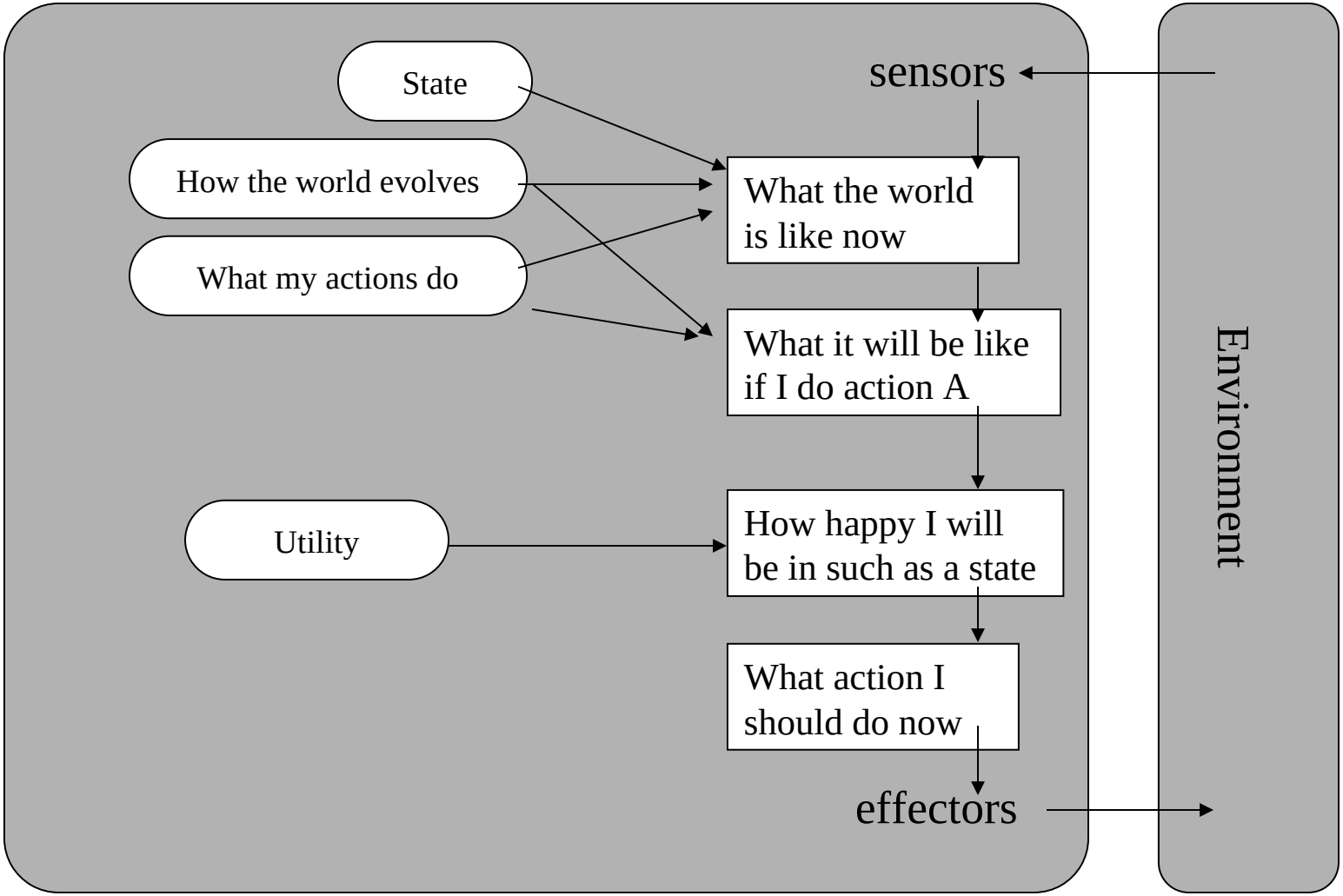
- Keeping track of the current state is often not enough – need to add goals to decide which situations are good

- Deliberative instead of reactive

- May have to consider long sequences of possible actions before deciding if goal is achieved – involves considerations of the future, “what will happen if I do…?” (search and planning)

- More flexible than reflex agent. (e.g. rain / new destination) In the reflex agent, the entire database of rules would have to be rewritten

# 5. Utility-based agent



# Utility-based agent …

• When there are multiple possible alternatives, how to decide which one is best?

• A goal specifies a crude destination between a happy and unhappy state, but often need a more general performance measure that describes "degree of happiness"

• Utility function U: State → Reals indicating a measure of success or happiness when at a given state

• Allows decisions comparing choice between conflicting goals, and choice between likelihood of success and importance of goal (if achievement is uncertain)

# Properties of environments

# Properties of environments

**Accessible** (observable) : The agent's sensory apparatus gives it access to the complete state of the environment

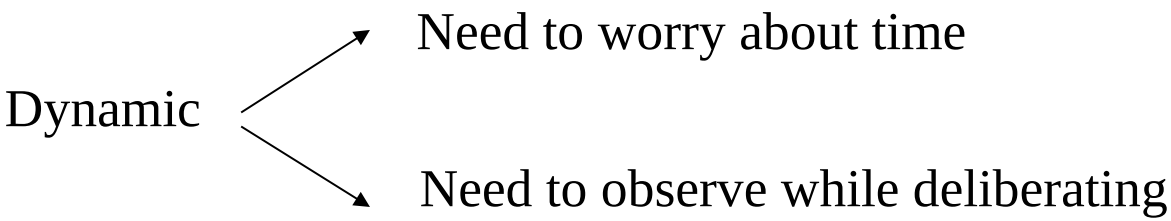
**Deterministic**: The next state of the environment is completely determined by the current state and the actions selected by the agent

*Subjective* non-determinism
- Limited memory (poker)
- Too complex environment to model directly (weather, dice)
- Inaccessibility

**Episodic**: The agent's experience is divided into independent "episodes," each episode consisting of agent perceiving and then acting. Quality of action depends just on the episode itself, because subsequent episodes do not depend on what actions occur in previous episodes. → Do not need to think ahead

# Properties of environments …

**Static**: If the environment can change while the agent is deliberating, then the environment is dynamic; otherwise it's static.




**Discrete**: There are a limited number of distinct, clearly defined percepts and actions we say that environment is discrete.

| Environment | Accessible | Deterministic | Episodic | Static | Discrete |
|---|---|---|---|---|---|
| Chess with a clock | Yes | Yes | No | Semi | Yes |
| Chess without a clock | Yes | Yes | No | Yes | Yes |
| Poker | No | No | No | Yes | Yes |
| Backgammon | Yes | No | No | Yes | Yes |
| Taxi driving | No | No | No | No | No |
| Medical diagnosis system | No | No | No | No | No |
| Image-analysis system | Yes | Yes | Yes | Semi | No |
| Part-picking robot | No | No | Yes | No | No |
| Refinery controller | No | No | No | No | No |
| Interactive English tutor | No | No | No | No | Yes |

# Running the agents and the environment

**procedure** RUN-ENVIRONMENT (*state*, UPDATE-FN, *agents*, *termination*)
  **inputs**: *state*, the initial state of the environment
    UPDATE-FN, function to modify the environment
    *agents*, a set of agents
    *termination*, a predicate to test when we are done

  **repeat**
    **for each** *agent* **in** *agents* **do**
      PERCEPT[*agent*] ← GET-PERCEPT(*agent*,*state*)
    **end**
    **for each** *agent* **in** *agents* **do**
      ACTION[*agent*] ← PROGRAM[*agent*] (PERCEPT[*agent*])
    **end**
    *state* ← UPDATE-FN(*actions*, *agents*, *state*)
  **until** termination (state)