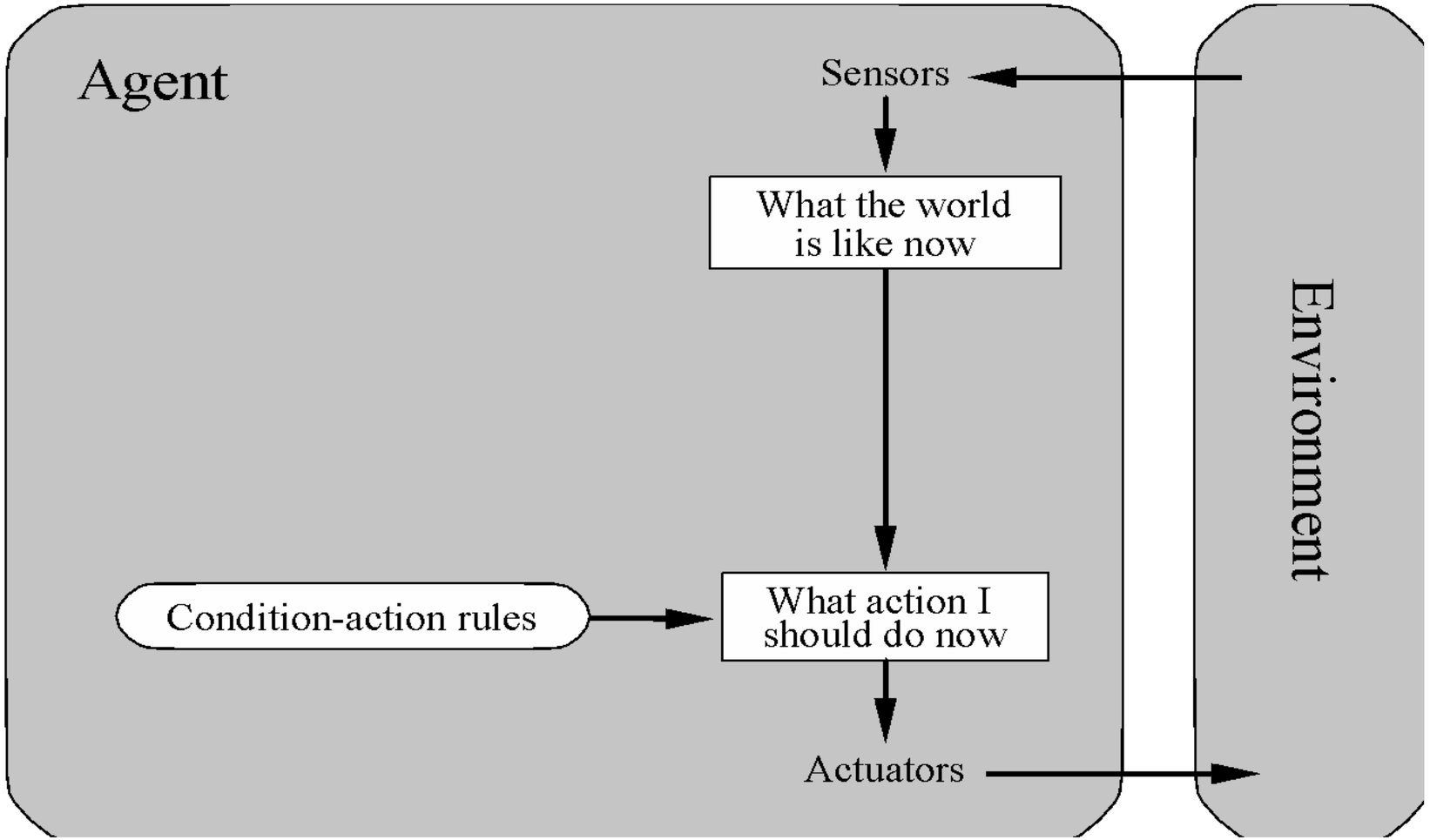
PURDUE CS47100

INTRODUCTION TO AI

RECAP

- **Agent**: perceives the environment via sensors and acts upon the environment via actuators
- **PEAS**: performance measures, environment, actuators, sensors
- **Environment properties**: fully/partially observable, single/multi-agent, deterministic/stochastic, static/dynamic, discrete/continuous, known/unknown
- **Algorithm components**: representation of state of environment/actions, learning/reasoning, action selection, generalization to new scenarios

SIMPLE REFLEX AGENTS

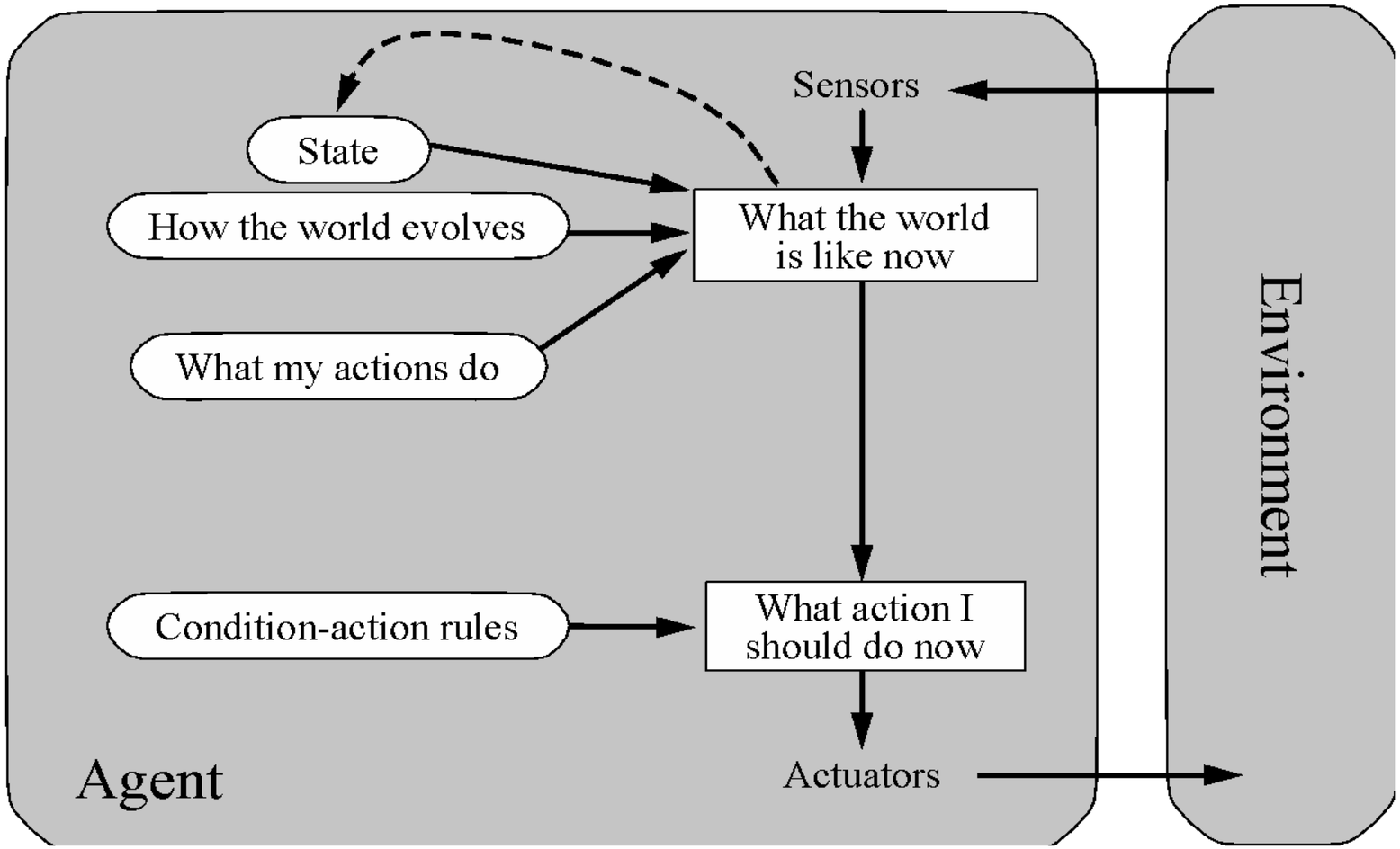


No memory! Not suitable for partially observable environments!

Need a huge table to store all the rules!

Unable to generalize to new situations!

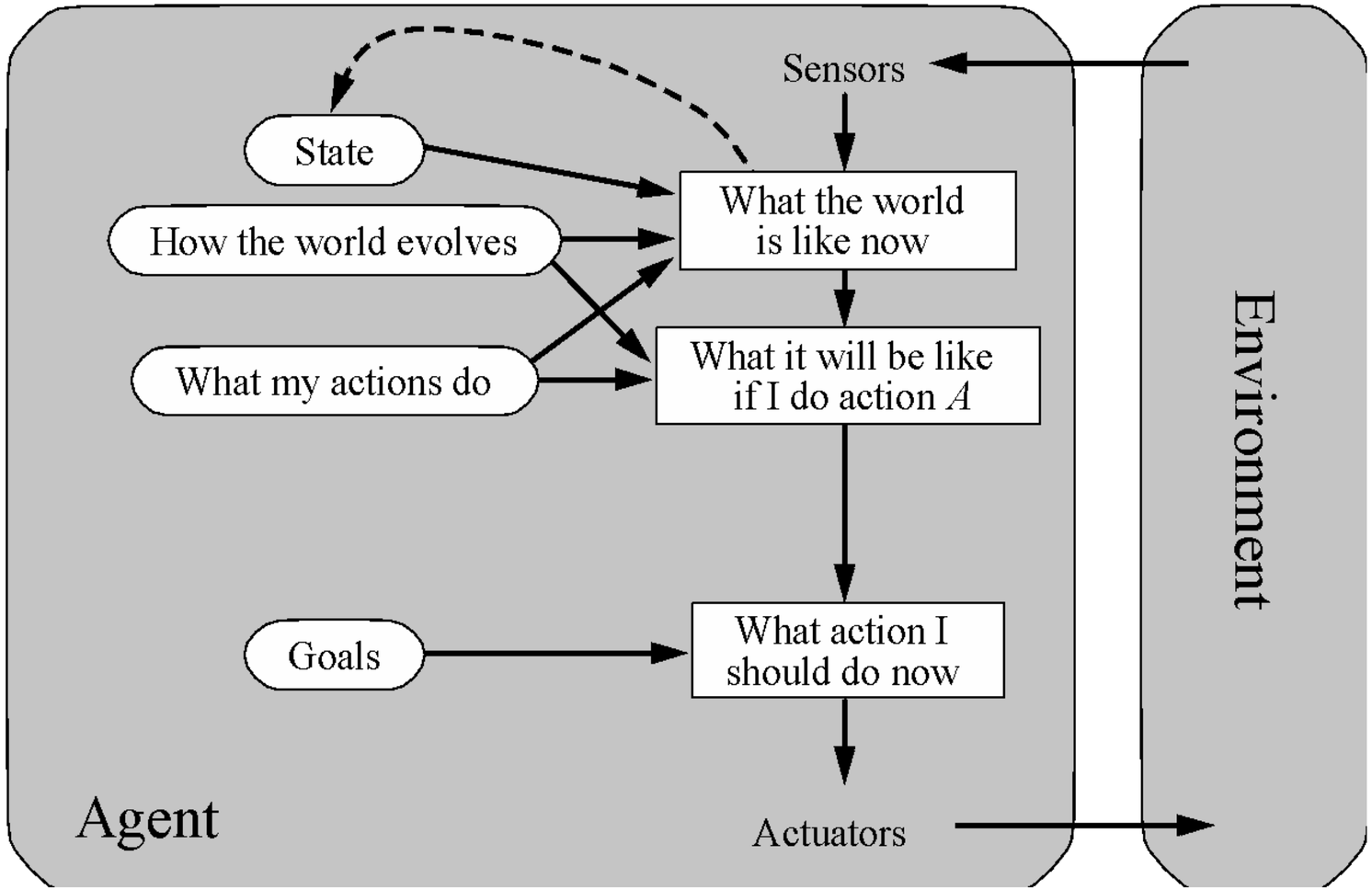
MODEL-BASED AGENTS



Use an **internal state** to keep track of the environment!

Still need a huge table to store all the rules and unable to generalize!

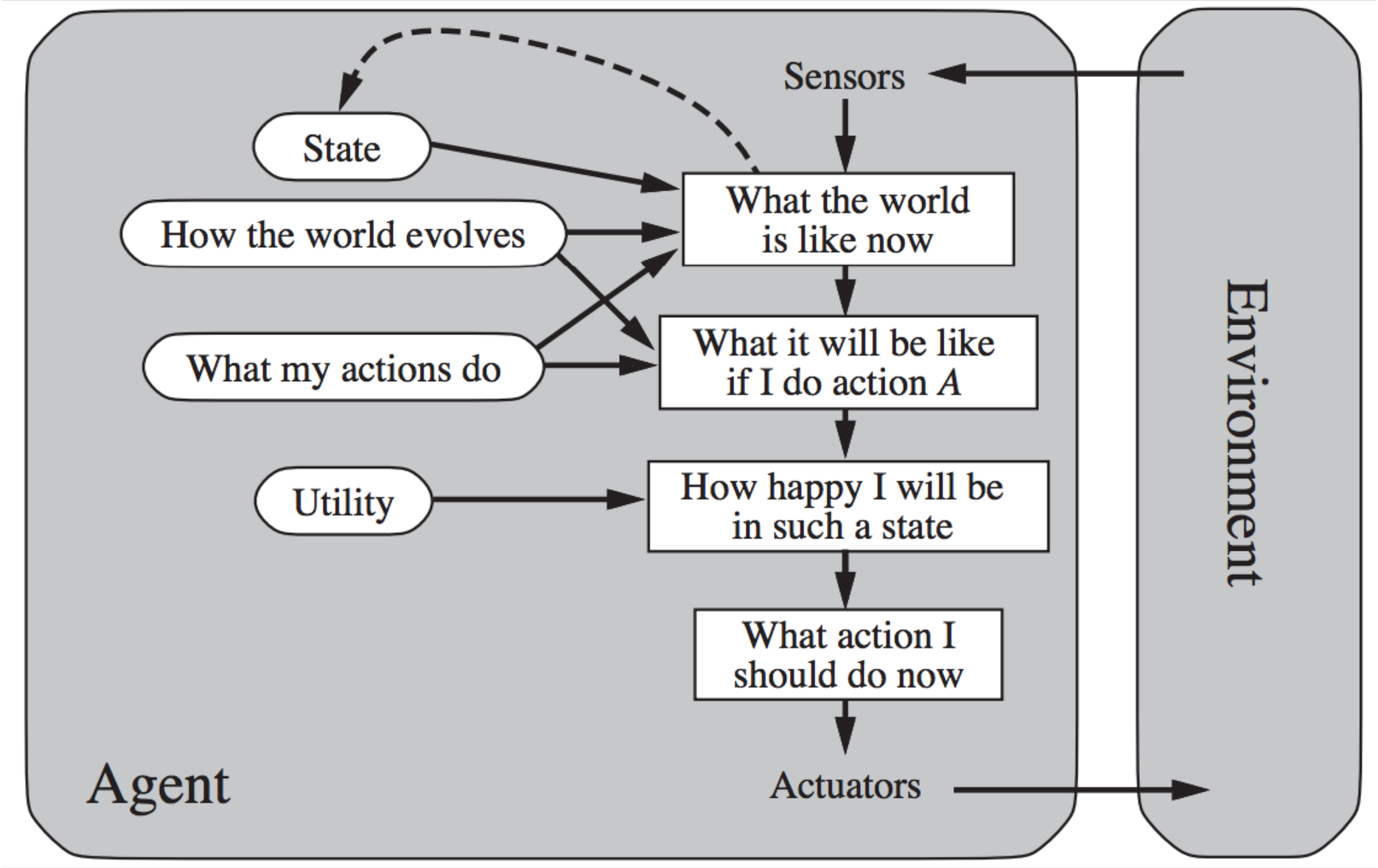
GOAL-BASED AGENTS



Goals provides a crude binary distinction between the agent being “happy” or “not happy”

But it can not tell exactly **how happy the agent is**…

UTILITY-BASED AGENTS

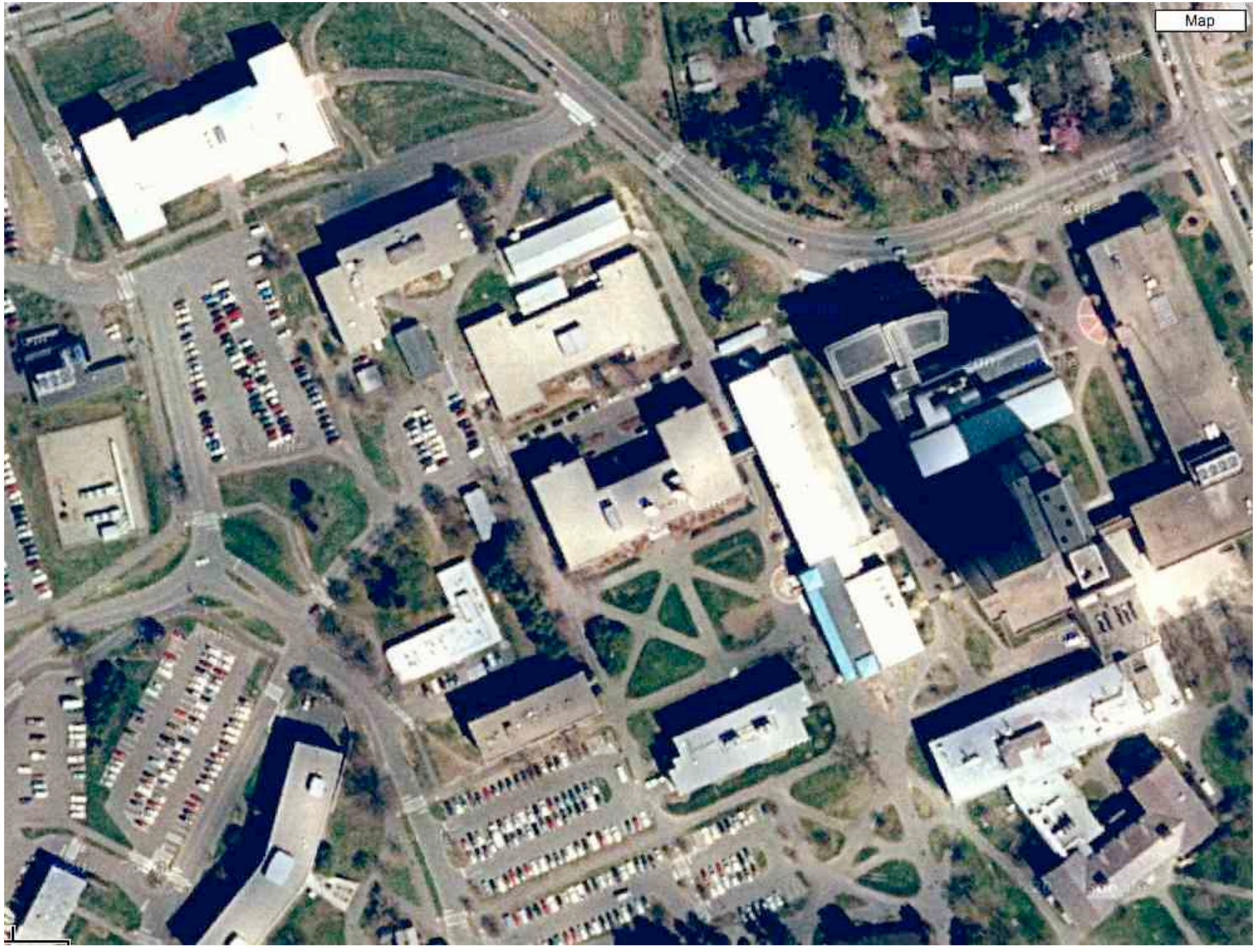


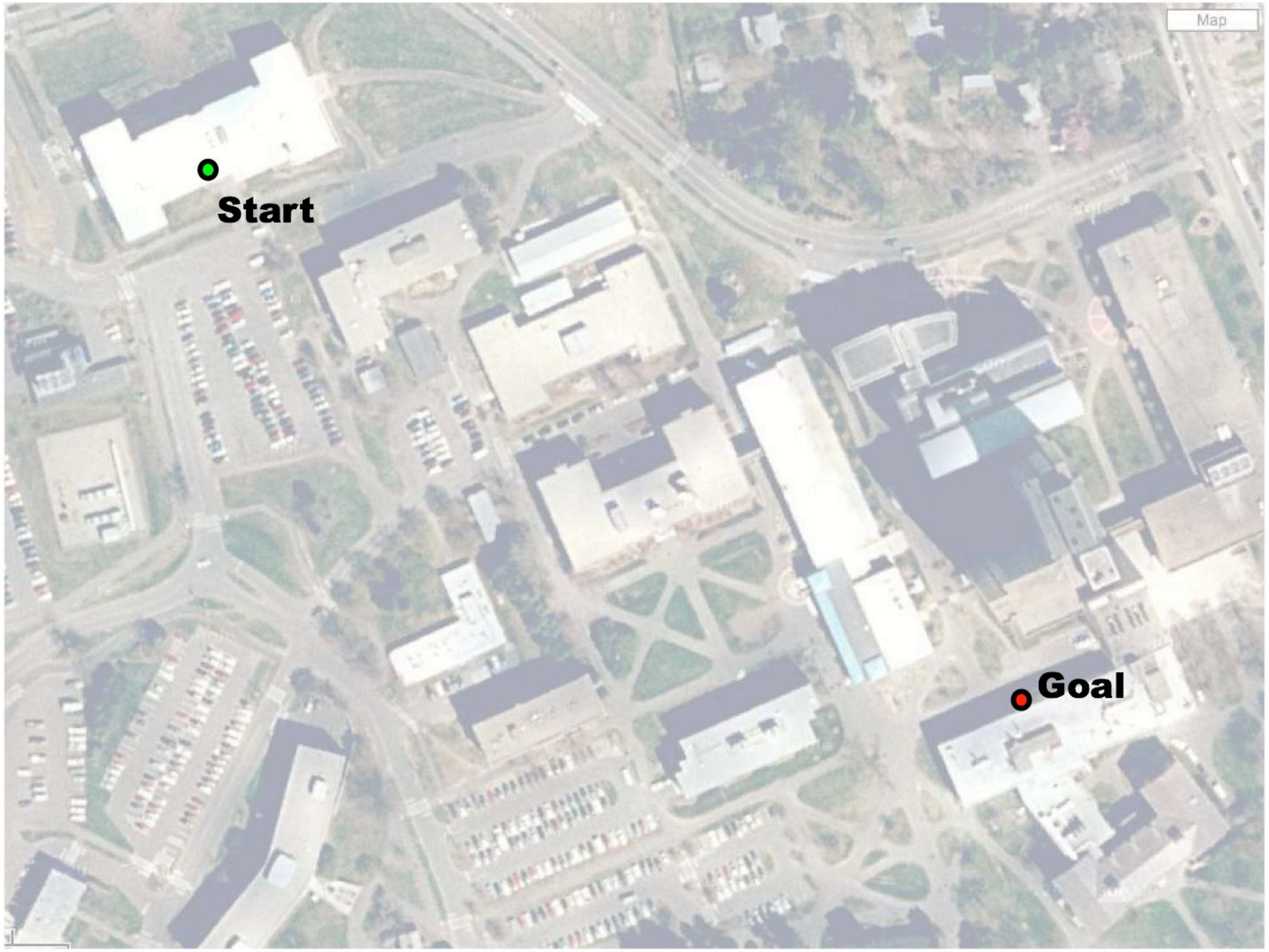
Designing a rational agent is to design an agent that **maximize its expected utility**!

SEARCH


I deliver to
Boilermakers
starship-purdue.com
STARSHIP


I deliver to
Boilermakers
starship-purdue.com


Map


Map
Start
Goal

PROBLEM SOLVING AS SEARCH: BASIC COMPONENTS

- **States**: *s*, describe the configuration of environment
  - *Initial state*: the state that an agent starts at
- **Actions (Operators):** *a*, activities which move the agent from one state to another
- **Transition model (Successor function):** Result(*s*, *a*), the state resulting from taking action *a* at state *s*

With states, actions, transition model, we can get a **state space**, i.e., the set of states reachable from the initial state by applying any sequence of actions

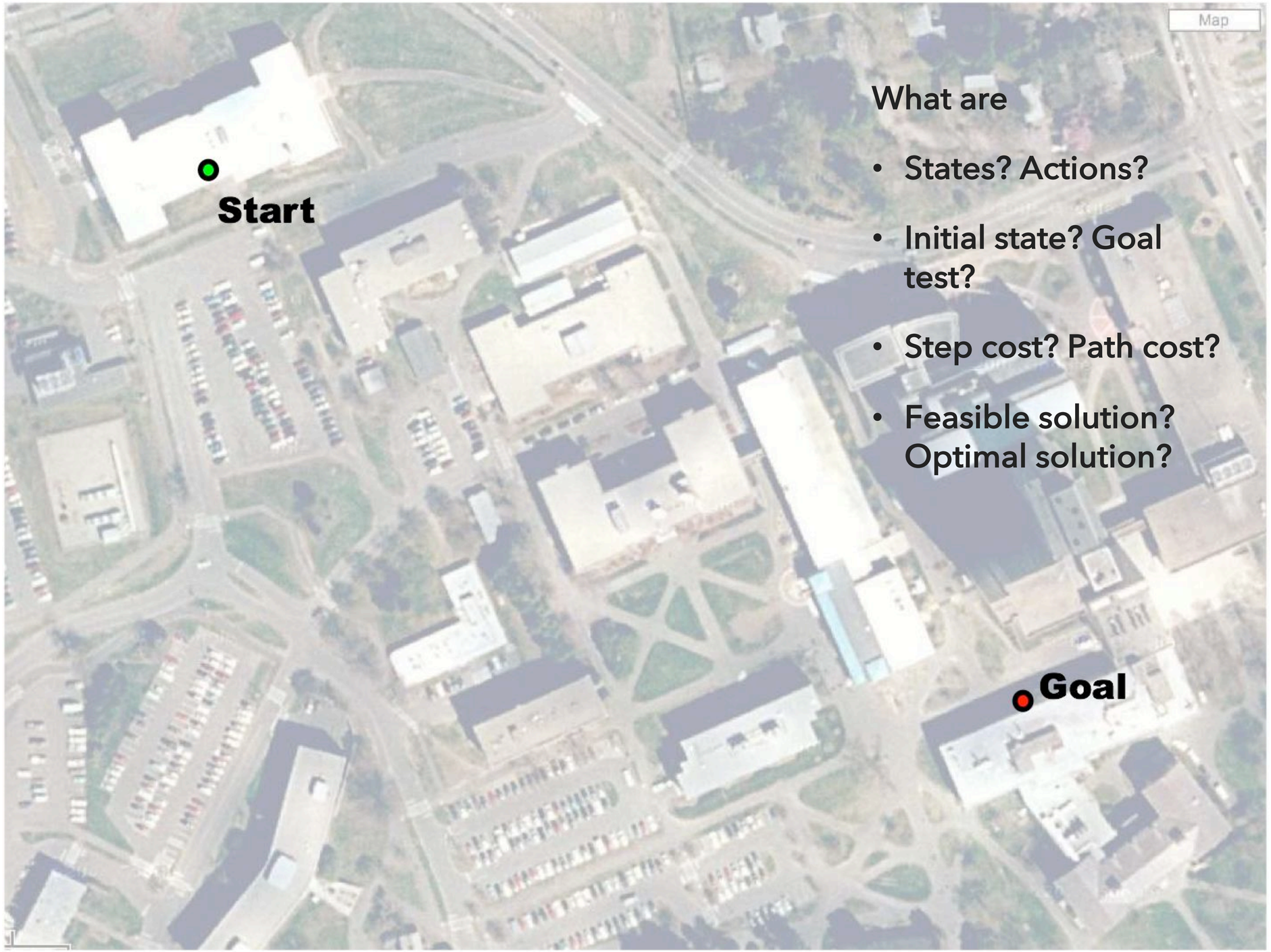
- State space is usually represented as a **graph**

PROBLEM SOLVING AS SEARCH: BASIC COMPONENTS

- **Goal test:** determines whether a state *s* is a goal state
- **Step cost:** *c(s, a, s')*, the cost of taking an action *a* which moves the agent from state *s* to *s'*
- **Path cost:** the cost of a path (i.e., a sequence of actions)

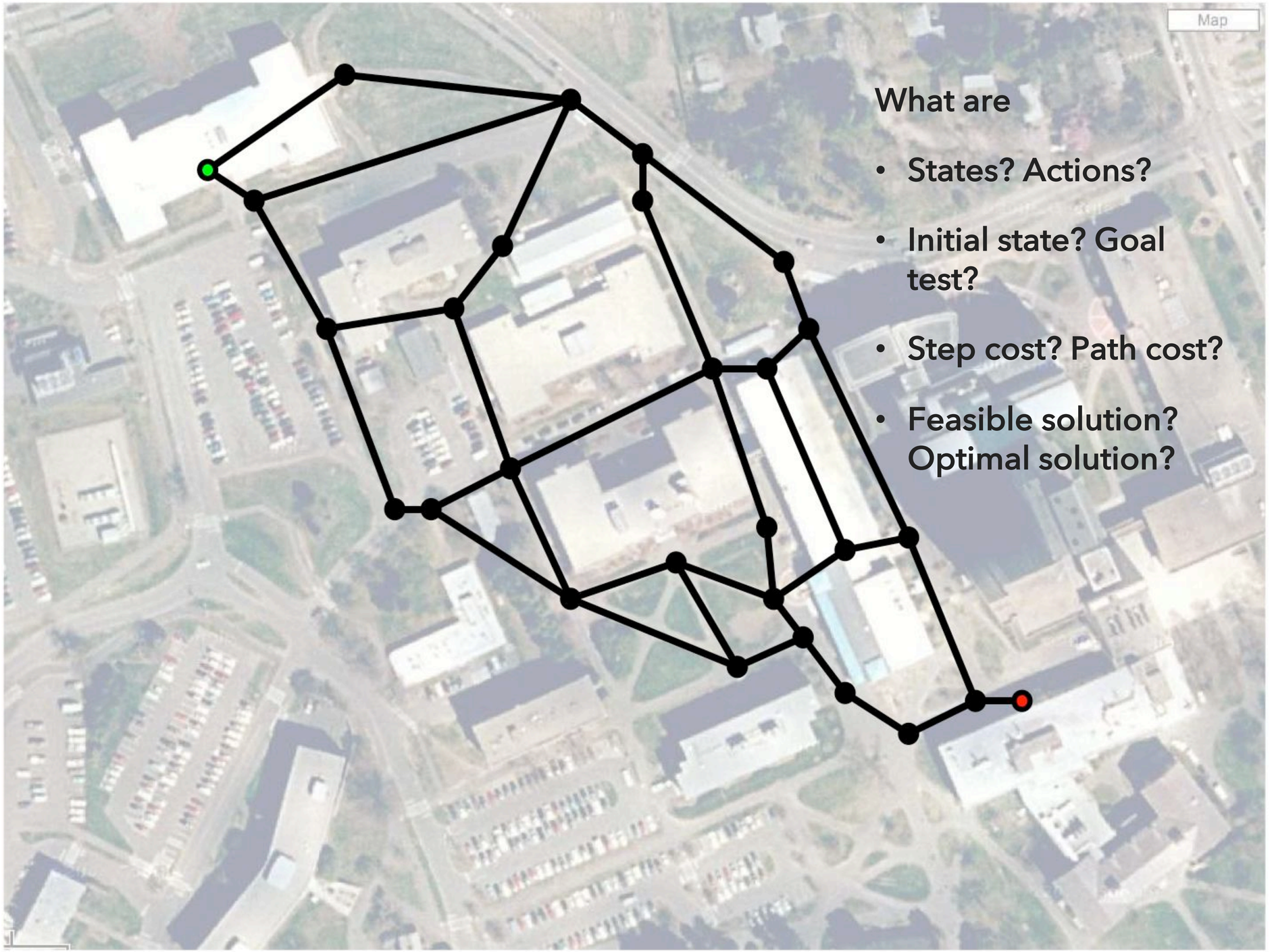
PROBLEM SOLVING AS SEARCH: BASIC COMPONENTS

- **Solution:** A sequence of actions which forms a **path** between the initial and goal states
- **Quality** of a solution is measured by its path cost; the **optimal solutions** have the lowest cost of **any** possible path
- **Search algorithm**: the algorithm that determines which actions should be tried in which order to find a path from the initial state to a goal state



What are

- States? Actions?
- Initial state? Goal test?
- Step cost? Path cost?
- Feasible solution? Optimal solution?



What are

- States? Actions?
- Initial state? Goal test?
- Step cost? Path cost?
- Feasible solution? Optimal solution?

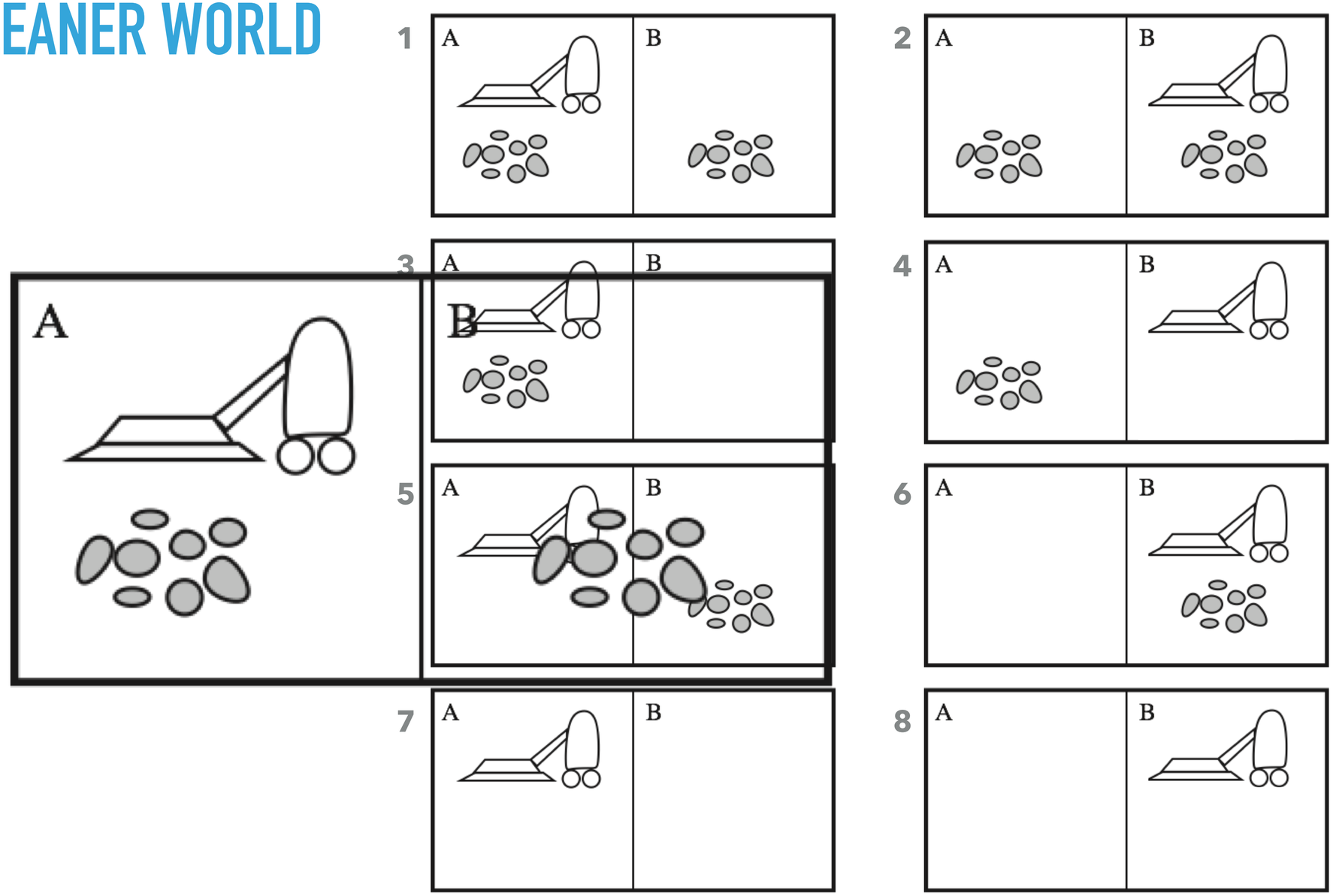
PROBLEM SOLVING AS SEARCH

Claim:
Many problems faced by intelligent agents, when properly formulated, can be solved by a single family of generic approaches

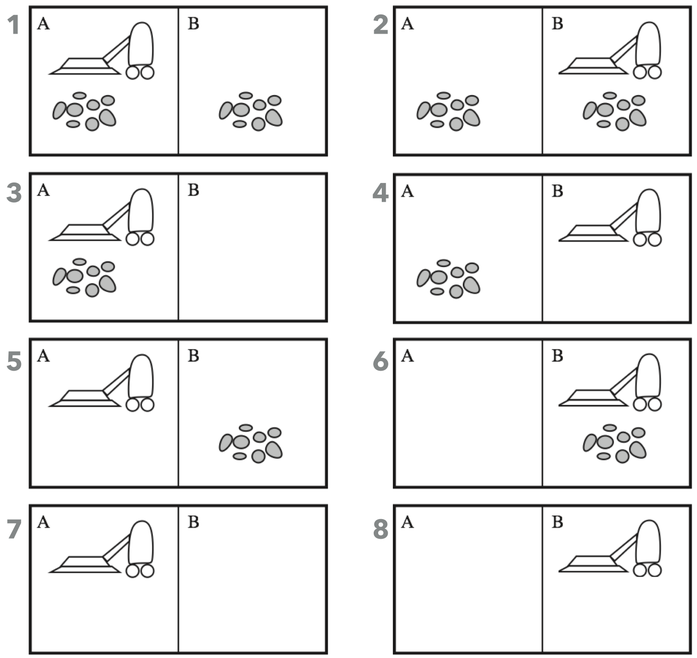
Search

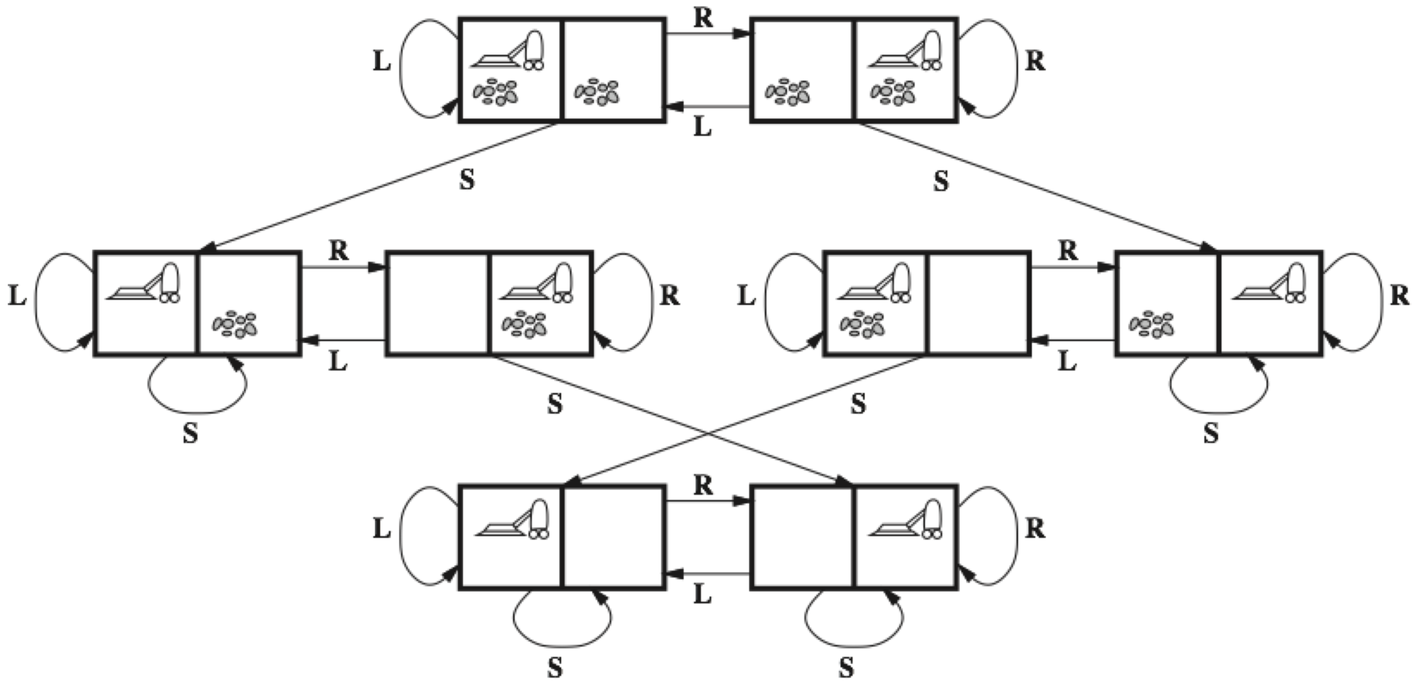
EXAMPLE: VACUUM CLEANER WORLD

- States –
- Actions –



EXAMPLE: VACUUM CLEANER WORLD



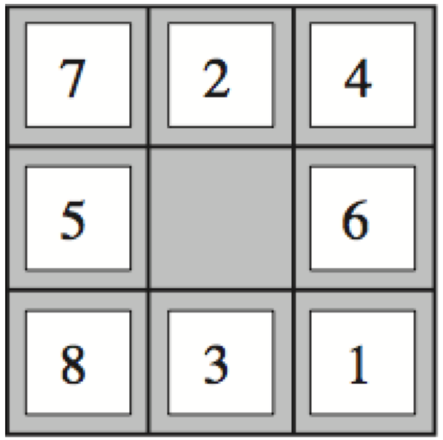


- **States** – Robot locations, A dirty? B dirty?
- **Actions** – Left, Right, Suck
- **Goal test** – no dirt
- **Path cost** – 1 per operation

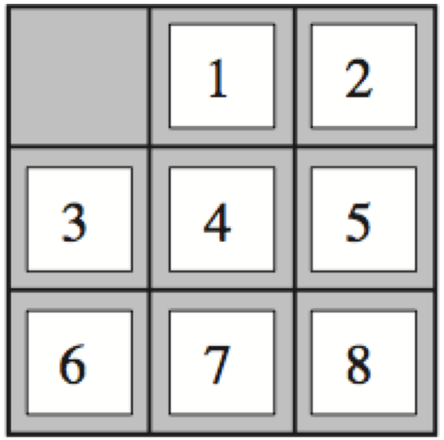
The state representation does not need to be atomic!

EXAMPLE: THE 8-PUZZLE

- States –
- Actions –
- Goal test –
- Path cost –



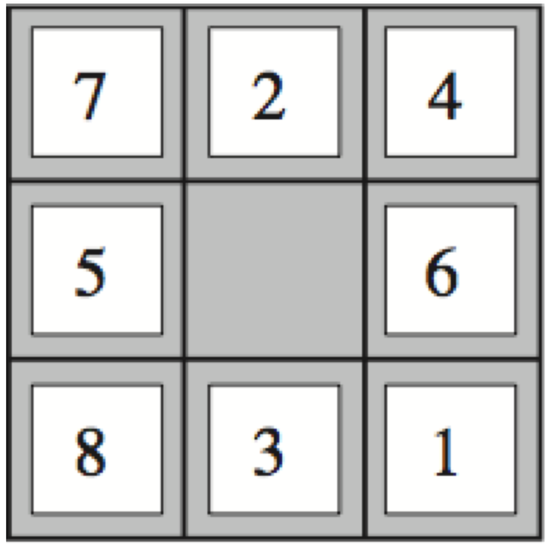
Start State



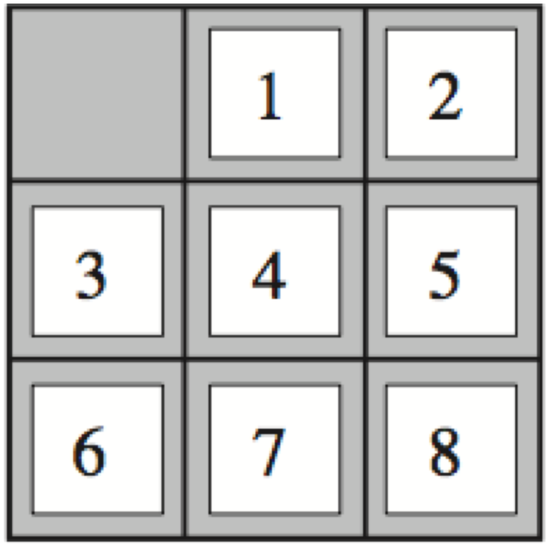
Goal State

EXAMPLE: THE 8-PUZZLE

- **States** – integer locations of tiles
- **Actions** – move blank left, right, up, down
- **Goal test** – goal state (given)
- **Path cost** – 1 per move

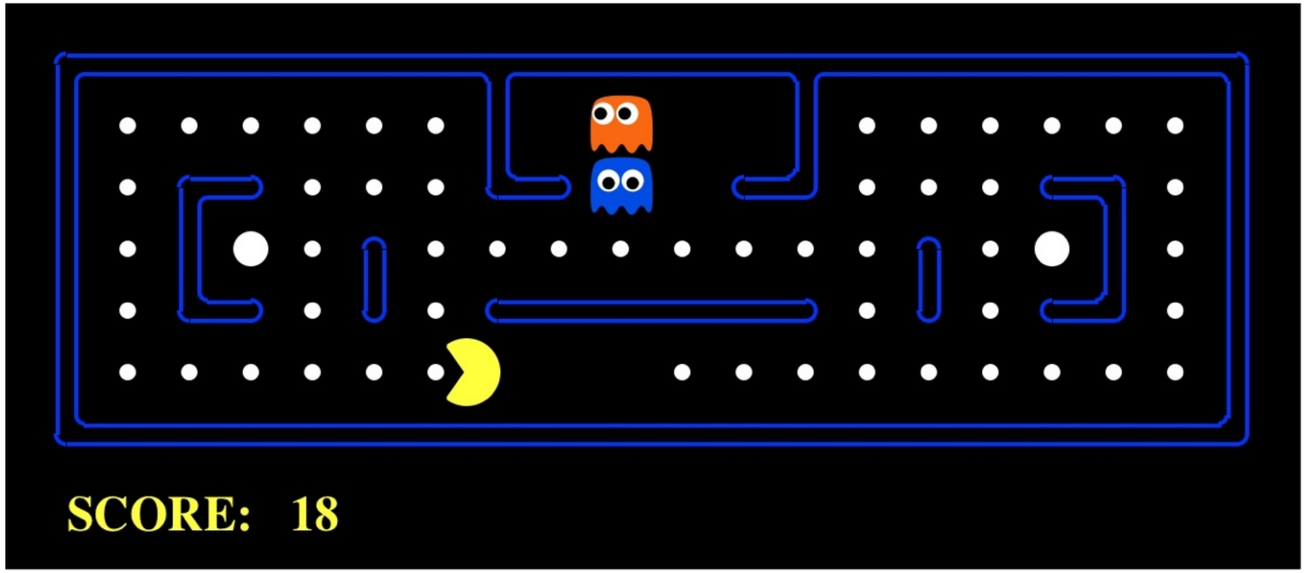


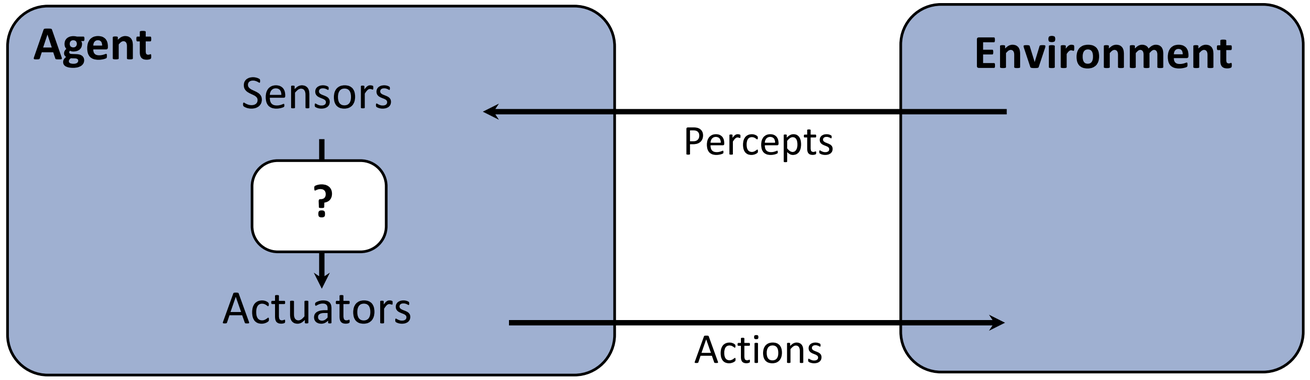
Start State



Goal State

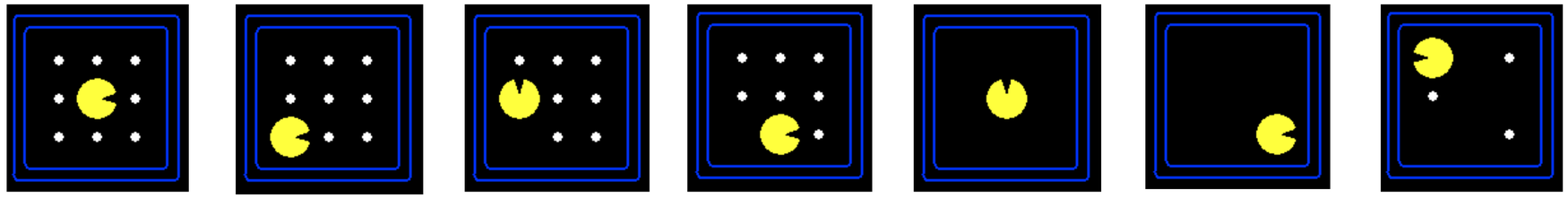
PACMAN AS AN AGENT



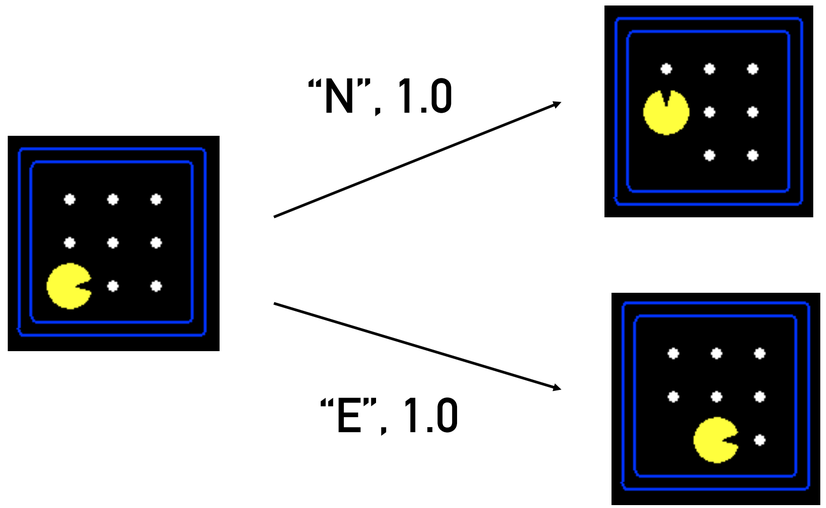


PACMAN EXAMPLE

- State space:

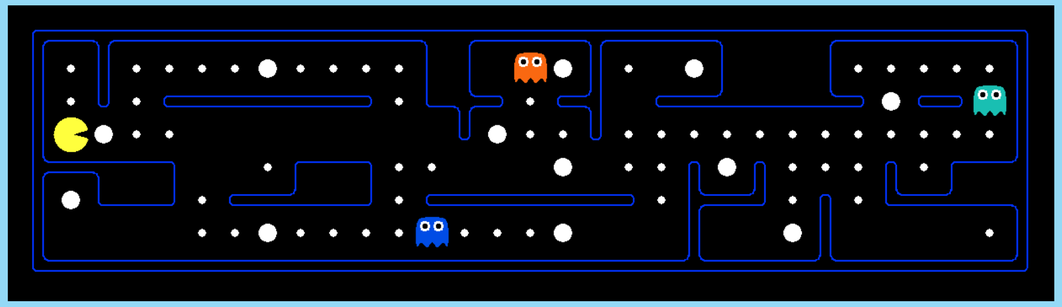


- Transition model:



STATE ≠ WORLD STATE!

The world state includes every last detail of the environment

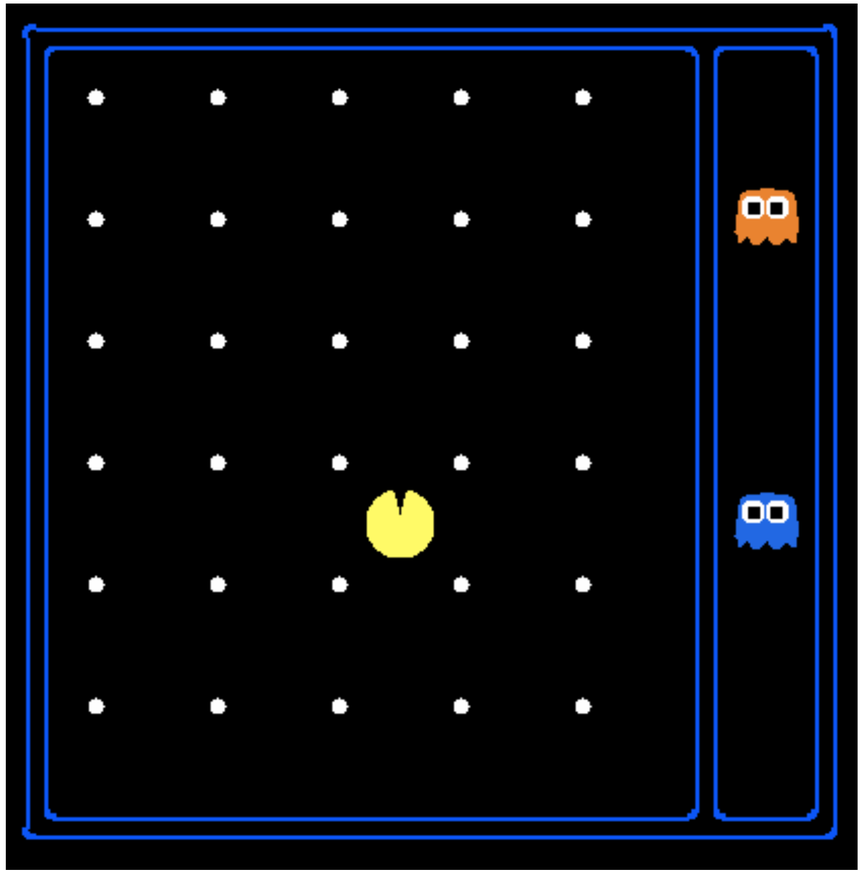


A search state keeps only the details needed for problem solving (abstraction)

- **Problem: Pathing**
  - States: (x,y) location
  - Actions: NSEW
  - Transition model: update location only
  - Goal test: is (x,y)=END
- **Problem: Eat-All-Dots**
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Transition model: update location and dots
  - Goal test: dots all false

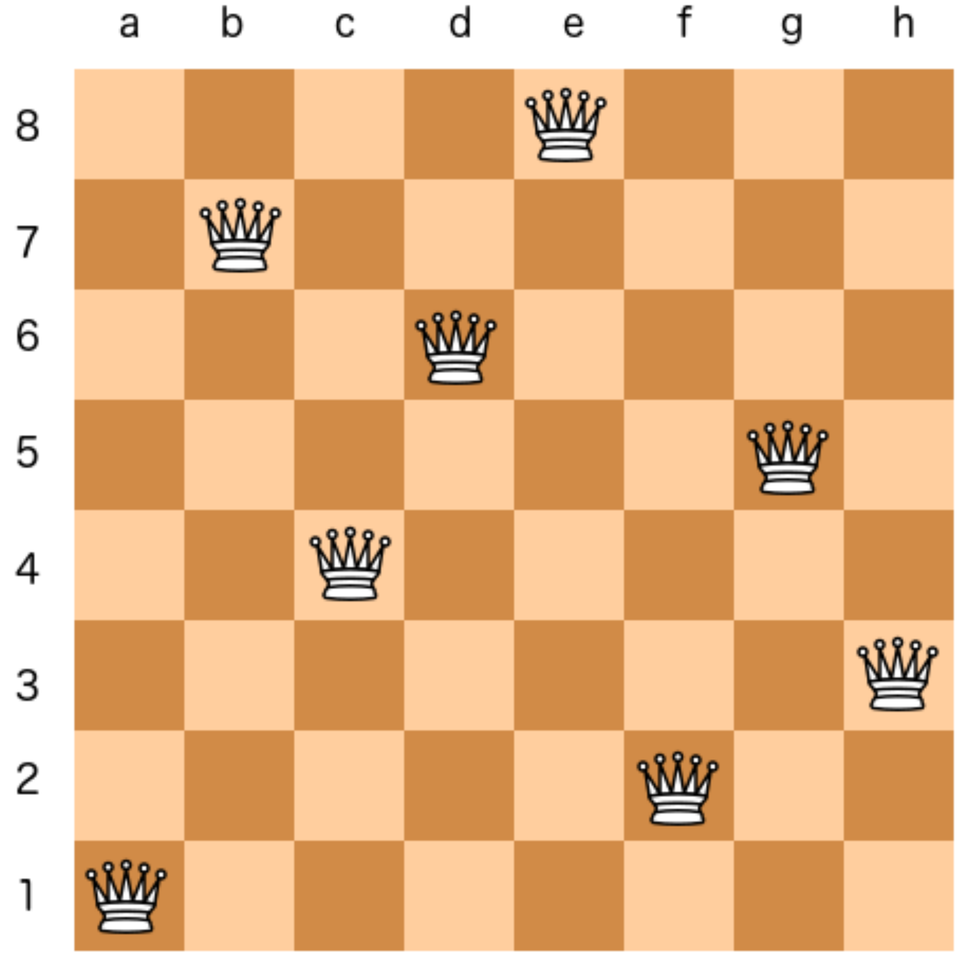
THE POWER OF ABSTRACTION

- World state: ???
  - Pacman positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many
  - World states? 120 x (2^{30}) x (12^2) x 4
  - States for pathing? 120
  - States for eat-all-dots? 120 x (2^{30})



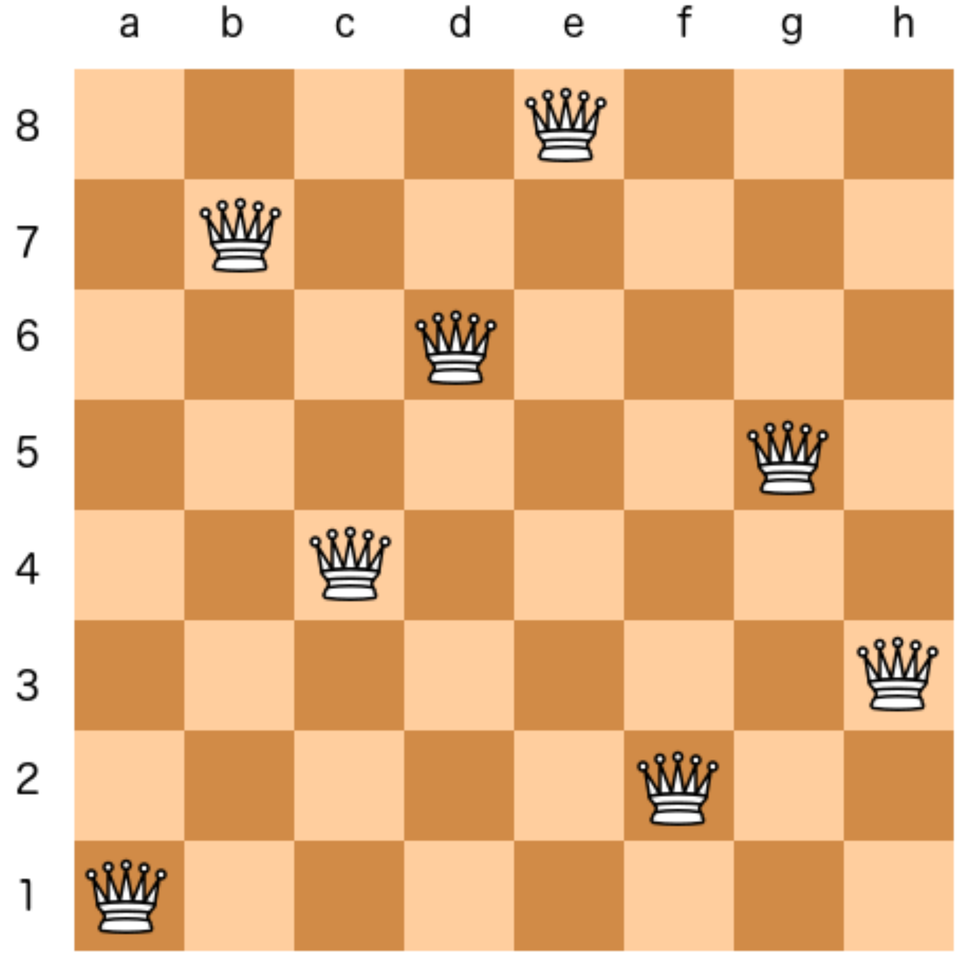
FORMULATE YOUR PROBLEM SMARTLY

- **Example**: 8-Queens Problem
- Place 8 queens on the chessboard such that no queen attacks any other
  - A queen attacks any piece in the same row, column, or diagonal
- How to define states and actions?



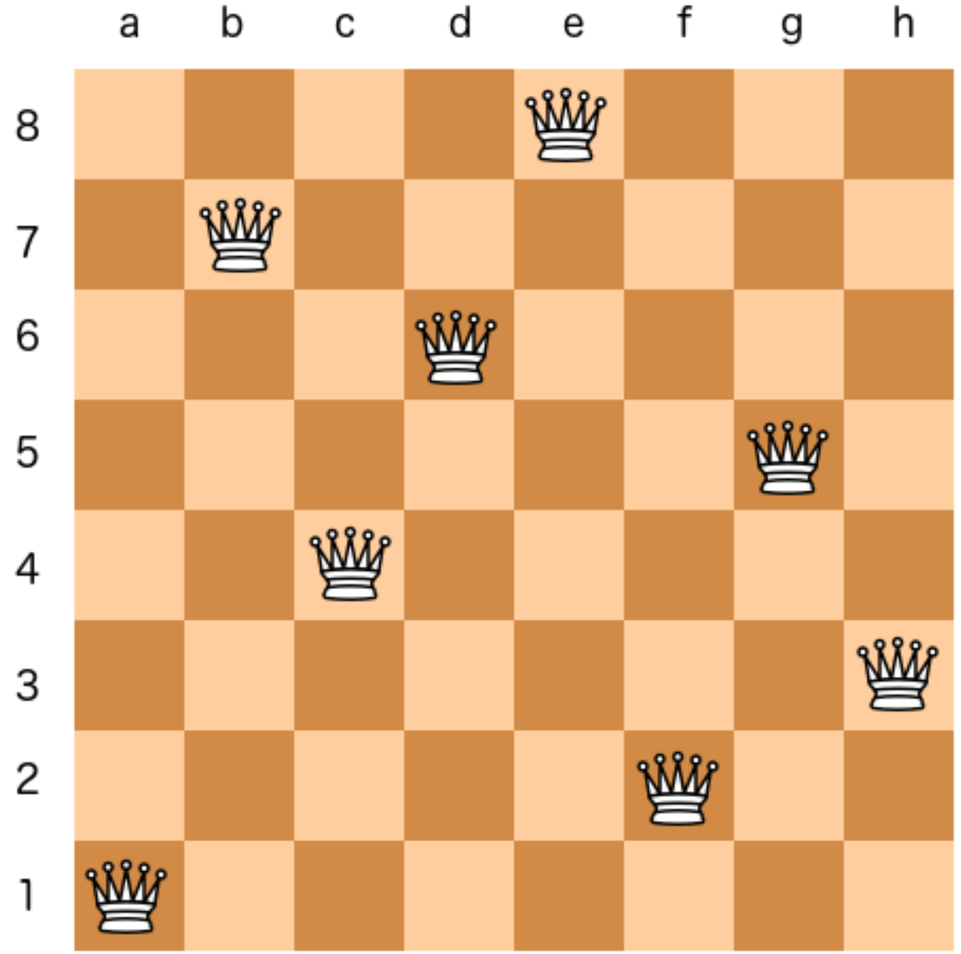
FORMULATE YOUR PROBLEM SMARTLY

- Option 1:
  - **State:** Position of all 8 queens on the board, e.g., [(a,1), (b,7), …, (h,3)]
  - **Actions:** randomly select one queen and move it up/down/to the left/to the right



FORMULATE YOUR PROBLEM SMARTLY

- Option 2:
  - **State:** The row number of the *n* queens in the leftmost *n* columns ($0 \leq n \leq 8$), one per column, e.g., [1, 7, 4]
  - **Actions**: Add a queen to any row in the leftmost empty column such that it is not attached by any existing queens



WHICH ONE IS BETTER?

- Option 1: (Complete-state formulation)
  - **State:** Position of all 8 queens on the board
  - **Actions**: randomly select one queen and move it up/down/to the left/to the right

Size of state space: $64\times63\times\cdots\times57 \approx 1.8\times10^{14}$!

- Option 2: (Incremental formulation)
  - **State:** The row number of the *n* queens ($0 \leq n \leq 8$) in the leftmost *n* columns, one per column
  - **Actions**: Add a queen to any row in the leftmost empty column such that it is not attached by any existing queens

Size of state space: 2057 !

The state does not need to be of the same "length"!