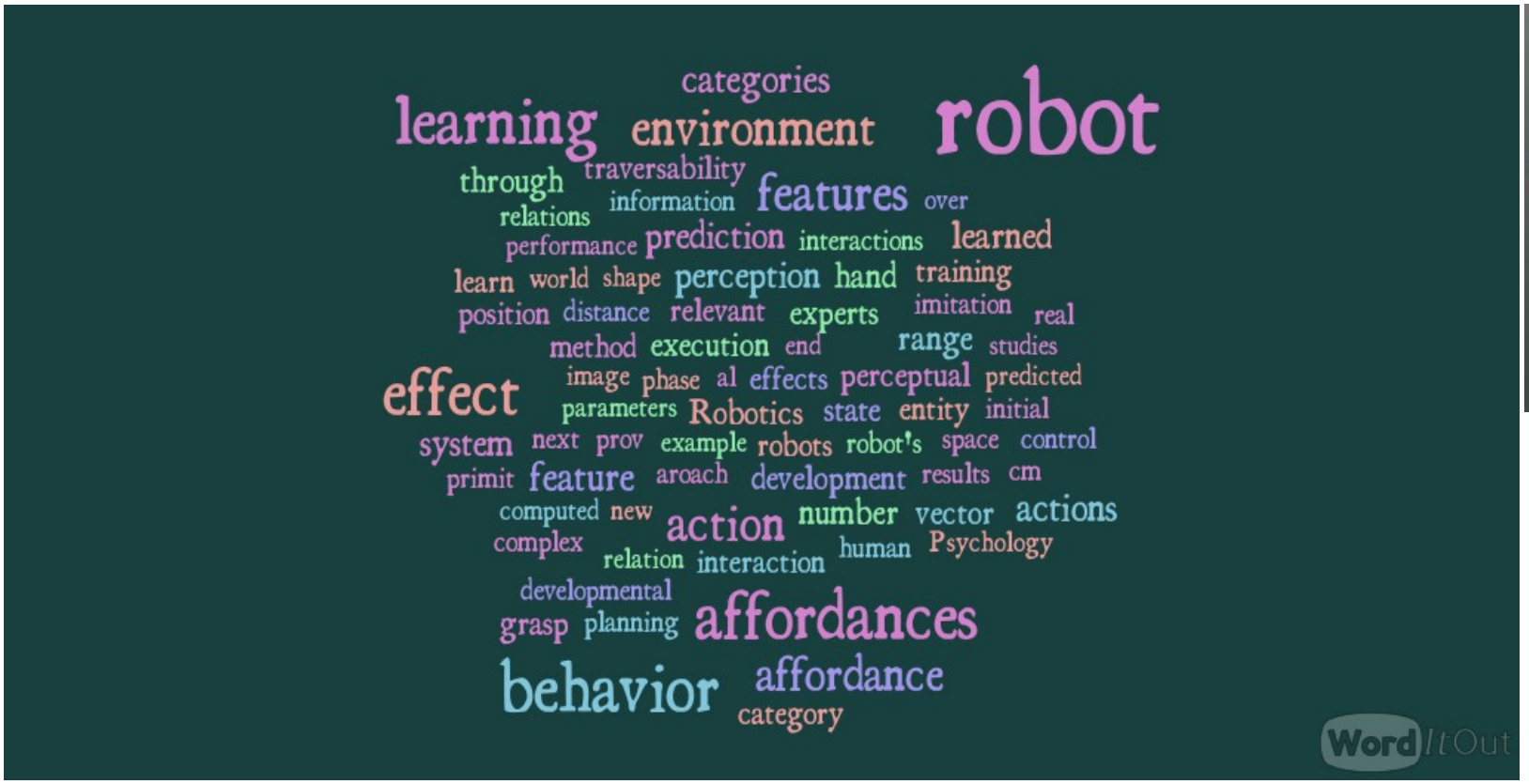
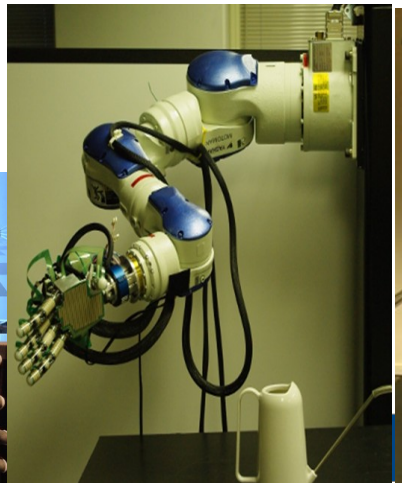
Week 1

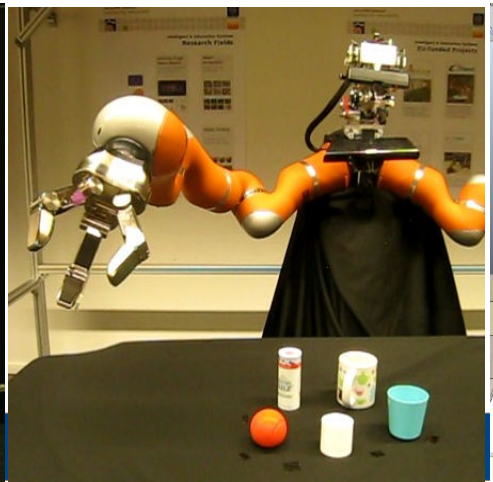
Emre Ugur, BM 33

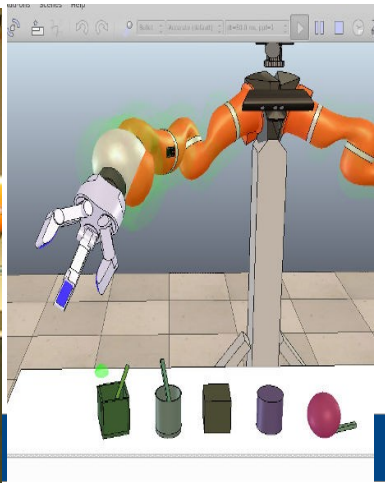
emre.ugur@boun.edu.tr

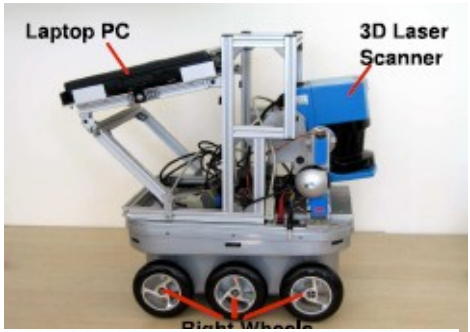
https://www.cmpe.boun.edu.tr/~emre/courses


categories
learning
environment
robot
through
traversability
relations
information
features
over
performance
prediction
interactions
learned
learn
world
shape
perception
hand
training
position
distance
relevant
experts
imitation
real
method
execution
end
range
studies
effect
image
phase
al
effects
perceptual
predicted
parameters
Robotics
state
entity
initial
system
next
prov
example
robots
robot's
space
control
primit
feature
aroach
development
results
cm
computed
new
action
number
vector
actions
complex
relation
interaction
human
Psychology
developmental
grasp
planning
affordances
behavior
affordance
category
WordItOut








Laptop PC
3D Laser Scanner

CoLoRs Lab: www.colors.cmpe.boun.edu.tr/

- Artificial Intelligence
- Machine Learning
- Robot Learning
- Introduction to Cognitive Science


HABER
TURK
FIRTINA NEDENİYLE İDO TÜM İÇ HAT SEFERLERİNİ İPTAL ETTİ
€ 4.6439

CMPE540: Responsibilities – This Year

- Quizzes (10): 0%
- Midterm (1): 30%
- Final: 30%
- Projects (4): 40%
- In-class activity: 0%

Syllabus (subject to change)

	Agents & Uninformed search	Ch 2, 3.1-4
	Introduction to Project Programming Environment	
	A* search, heuristics Local search; search-based agents	Ch. 3.5-6 Ch. 4
	Game playing Constraint satisfaction problems	Ch. 5.1-5 Ch. 6.1, 6.3-5
	Propositional logic: semantics and inference Propositional planning and logical agents	Ch. 7.1-4, 7.6.1 Ch. 7.7
	First-order logic ..	Ch. 8.1-3, 9.1
	Planning	

	Midterm - tentative	
	Probability Bayes nets: Syntax and semantics	Ch. 13.1-5 Ch. 14.1-3
	Bayes nets: Exact inference Bayes nets: Approximate inference	Ch. 14.3 Ch. 14.4
	Markov Models, Hidden Markov Models Applications of HMMs	Ch. 15.1-3, 15.5 Ch. 22.1, 23.5
	Decision theory Markov decision processes	Ch. 16.1-3, 16.5-6 Ch. 17.1
	Machine learning: Classification and regression	Ch. 18.1-4, 18.6
	Advanced Topics: Vision and robotics	Ch. 24,25

Projects – programming + report

- Programming Language: C++
- Programming environment: **Linux**
- Automatically graded

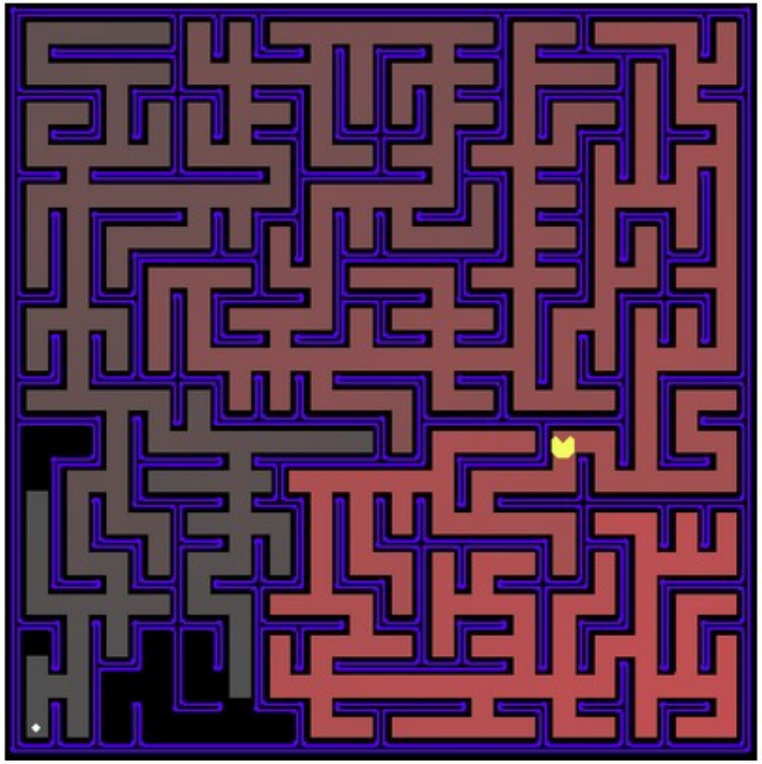
- Will be announced at least 2 weeks before due date
- Late submission
  - Upto 4 days in total.

- Topics (subject to change):
  - Search
  - Adversarial search
  - Logic
  - Regression

Projects – programming + report

- Projects are adapted from inst.eecs.berkeley.edu
  - https://inst.eecs.berkeley.edu/~cs188/fa11/projects/

Project 1: Search in Pacman

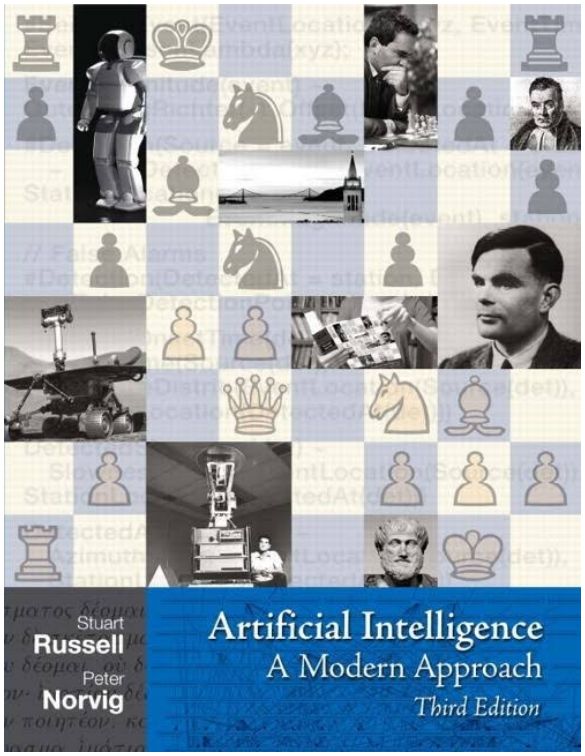


All those colored walls,
Mazes give Pacman the blues,
So teach him to search.

Quizzes

- In every lecture
- Random time
- Mostly from the current lecture – but might be from paper readings or the previous lecture
  - Pseudo-code
  - Problem solving
  - Algorithm application
  - Explanation, definition, etc.
-
-

Sources & credits



- Course notes:
  - Russell and Norvig:
    - http://aima.cs.berkeley.edu/
  - Levent Akin, Pinar Yolum and Albert Ali Salah
    - http://www.cmpe.boun.edu.tr/~akin/
    - http://mas.cmpe.boun.edu.tr/wiki/doku.php?id=courses:cmpe540:info
- Projects and content:
  - UC Berkeley CS188 Intro to AI
    - https://inst.eecs.berkeley.edu/~cs188/fa11/assignments.html

Chapter 1:
Introduction

What is AI?

- The study of creating intelligent agents

What is AI?

Views of AI fall into four categories (Humanly vs. Rationally)

- Acting rationally, Thinking humanly, Acting humanly, Thinking rationally,

``The exciting new effort to make computers think ... machines with minds, in the full and literal sense'' (Haugeland, 1985)  ``The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning ...'' (Bellman, 1978)	``The study of mental faculties through the use of computational models'' (Charniak and McDermott, 1985)  ``The study of the computations that make it possible to perceive, reason, and act'' (Winston, 1992)
``The art of creating machines that perform functions that require intelligence when performed by people'' (Kurzweil, 1990)  ``The study of how to make computers do things at which, at the moment, people are better'' (Rich and Knight, 1991)	``A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes'' (Schalkoff, 1990)  `` AI … is concerned with intelligent behavior in artifacts.” (Nillson, 1998)

Thinking humanly: cognitive modeling

- 1960s "cognitive revolution": information processing psychology
  - Requires scientific theories of internal activities of the brain
- How to validate? Requires
  - 1) Predicting and testing behavior of human subjects (top-down)
  - or 2) Direct identification from neurological data (bottom-up)
- Both approaches (roughly, Cognitive Science and Cognitive Neuroscience) are now distinct from AI

Milestones in understanding and implementing intelligent systems

- Turing, Alan M. "On computable numbers, with an application to the Entscheidungsproblem." Proceedings of the London mathematical society 2.1 (1937): 230-265.
- McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.
- Lettvin, J. Y., Maturana, H. R., McCulloch, W. S., & Pitts, W. H. (1959). What the frog's eye tells the frog's brain. Proceedings of the IRE, 47(11), 1940-1951.
- Hubel, David H., and Torsten N. Wiesel. "Receptive fields of single neurones in the cat's striate cortex." The Journal of physiology 148.3 (1959): 574-591.
- Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological review, 63(2), 81.
- Newell, Allen, and Herbert Simon. "The logic theory machine--A complex information processing system." IRE Transactions on information theory 2.3 (1956): 61-79.
- Chomsky, Noam. "Three models for the description of language." IRE Transactions on information theory 2.3 (1956): 113-124.
- Von Neumann, John. 1958 The computer and the brain. Yale University Press,.
- Putnam, H. 1960. "Minds and Machines." InS. Hook, ed., Dimensions of Mind. New York: New York University Press.
- Marr, D. (1982) Vision: A Computational Investigation info the Human Representation and Processing of Visual Information. San Francisco: W. H. Freeman.

Turing Machine: The idea of algorithm and computation

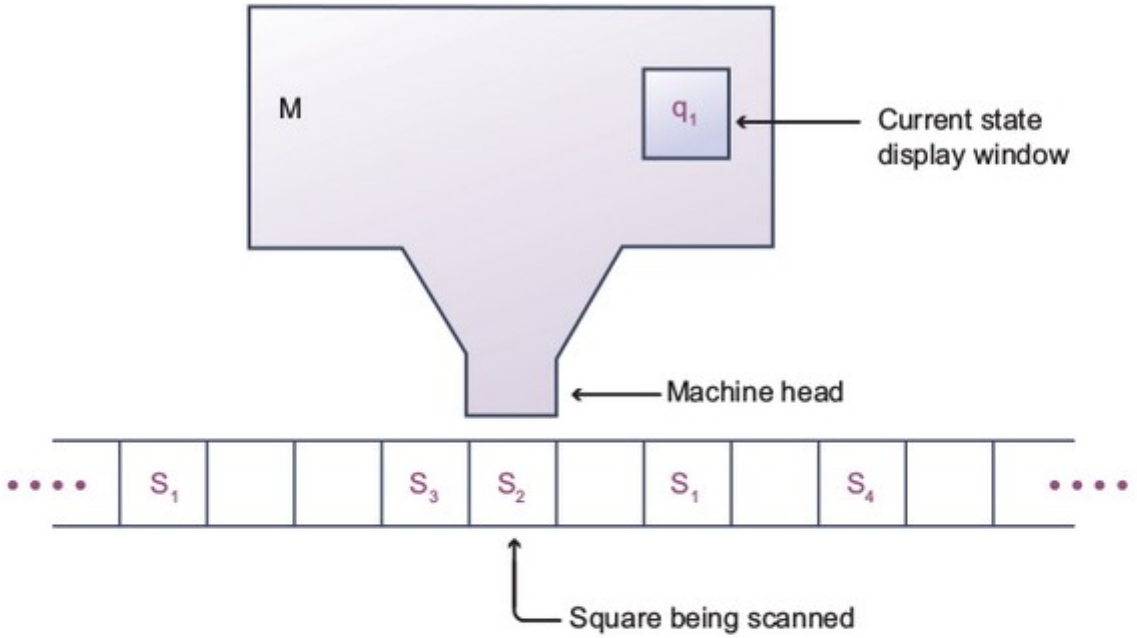
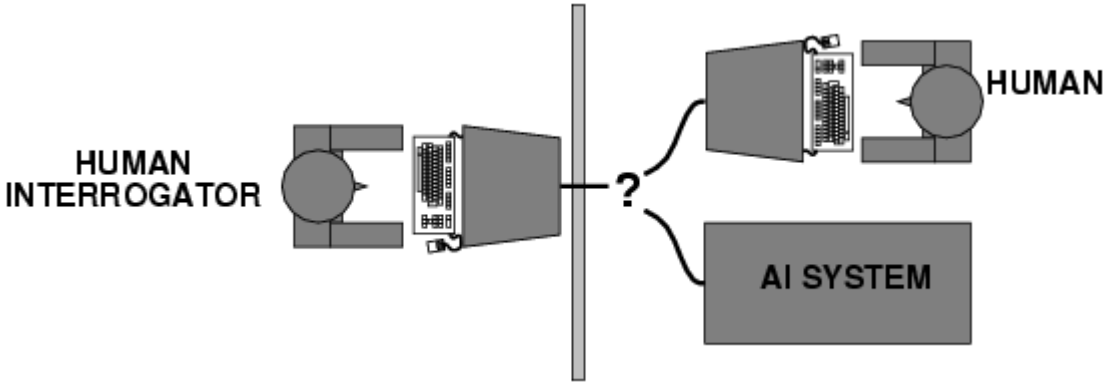


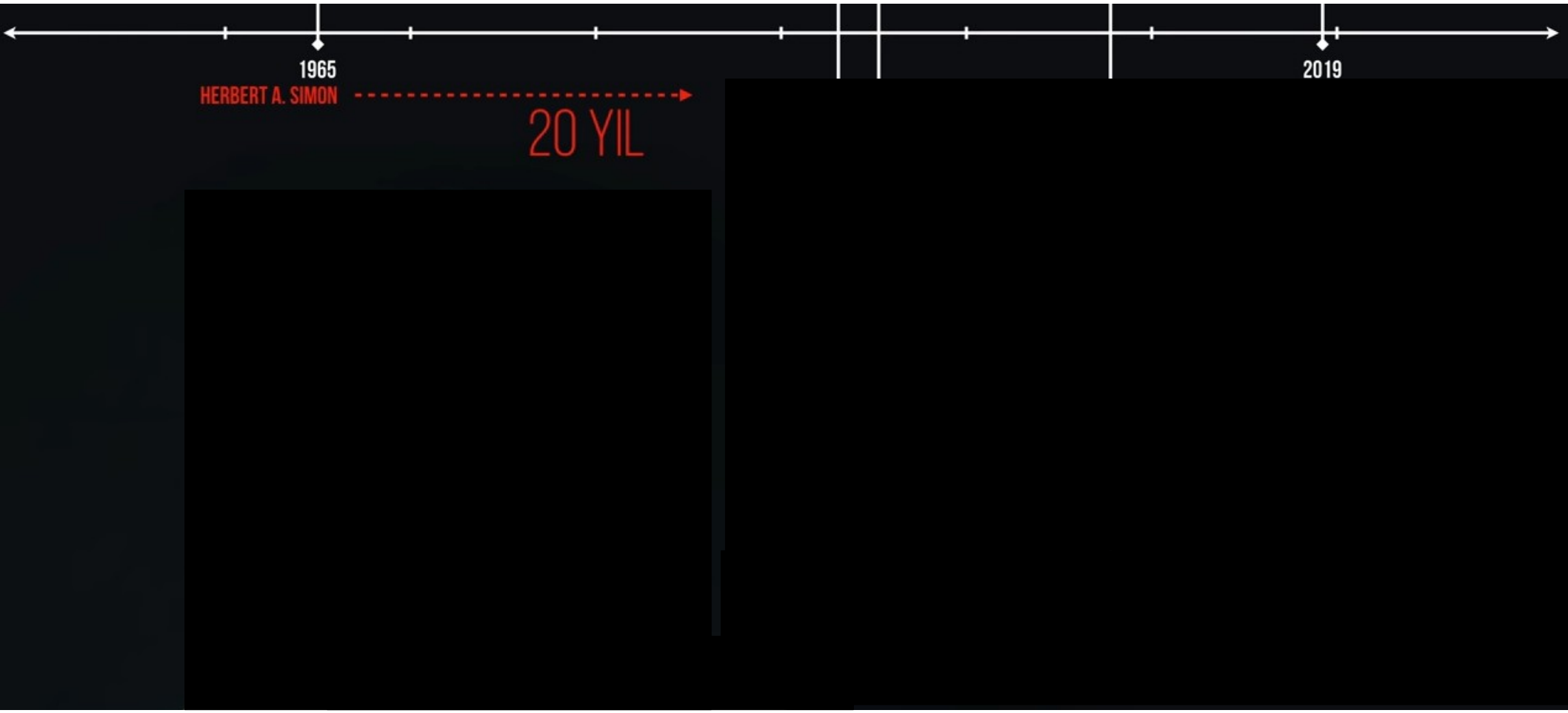
Figure 1.4 Schematic representation of a Turing machine. (Adapted from Cutland 1980)

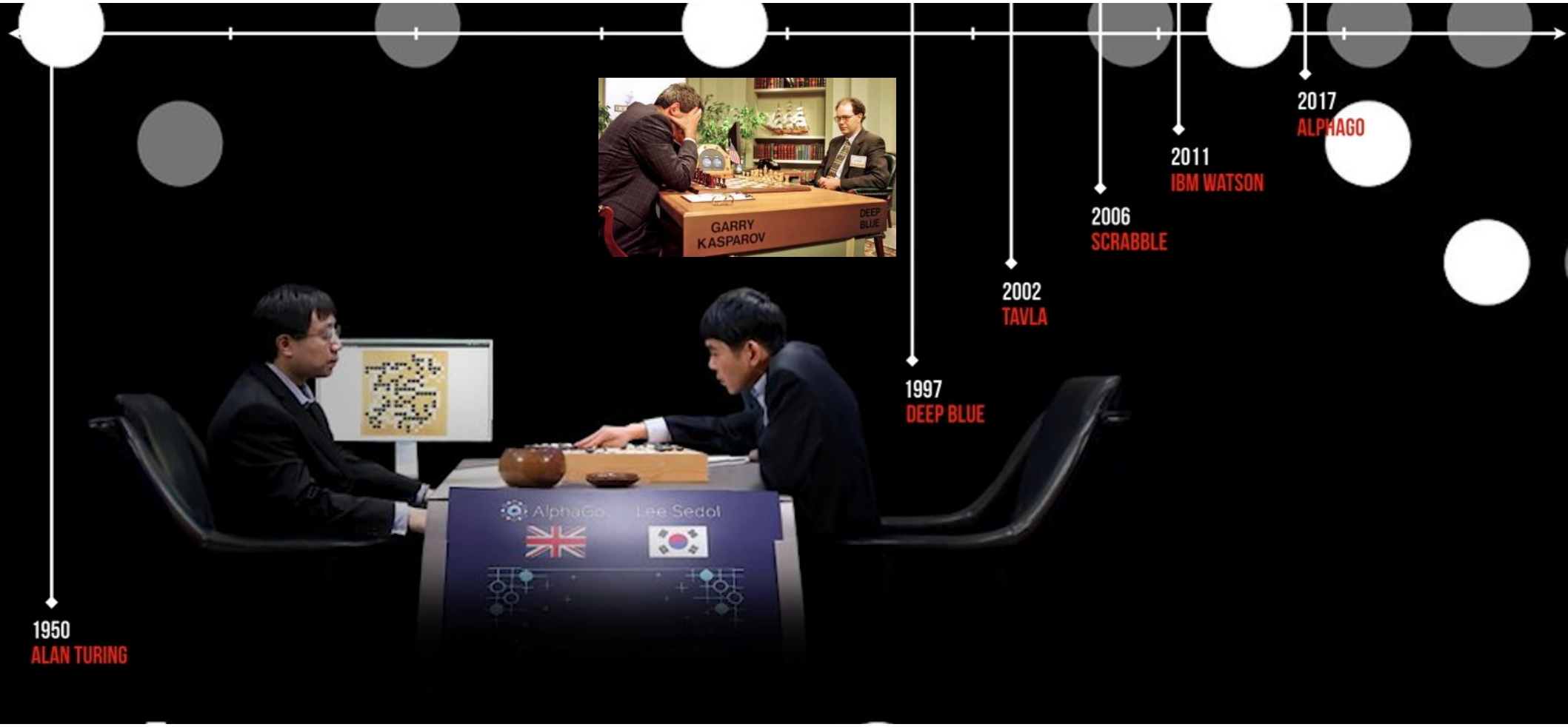
- delete the symbol in the cell
- write a new symbol in the cell
- move the tape one cell to the left
- move the tape one cell to the right

Acting humanly: Turing Test

- Turing (1950) "Computing machinery and intelligence":
- "Can machines think?" "Can machines behave intelligently?"
- Operational test for intelligent behavior: the Imitation Game
- Predicted that by 2000, a machine might have a 30% chance of fooling a lay person for 5 minutes
- Anticipated all major arguments against AI in following 50 years
- Suggested major components of AI: knowledge, reasoning, language understanding, learning

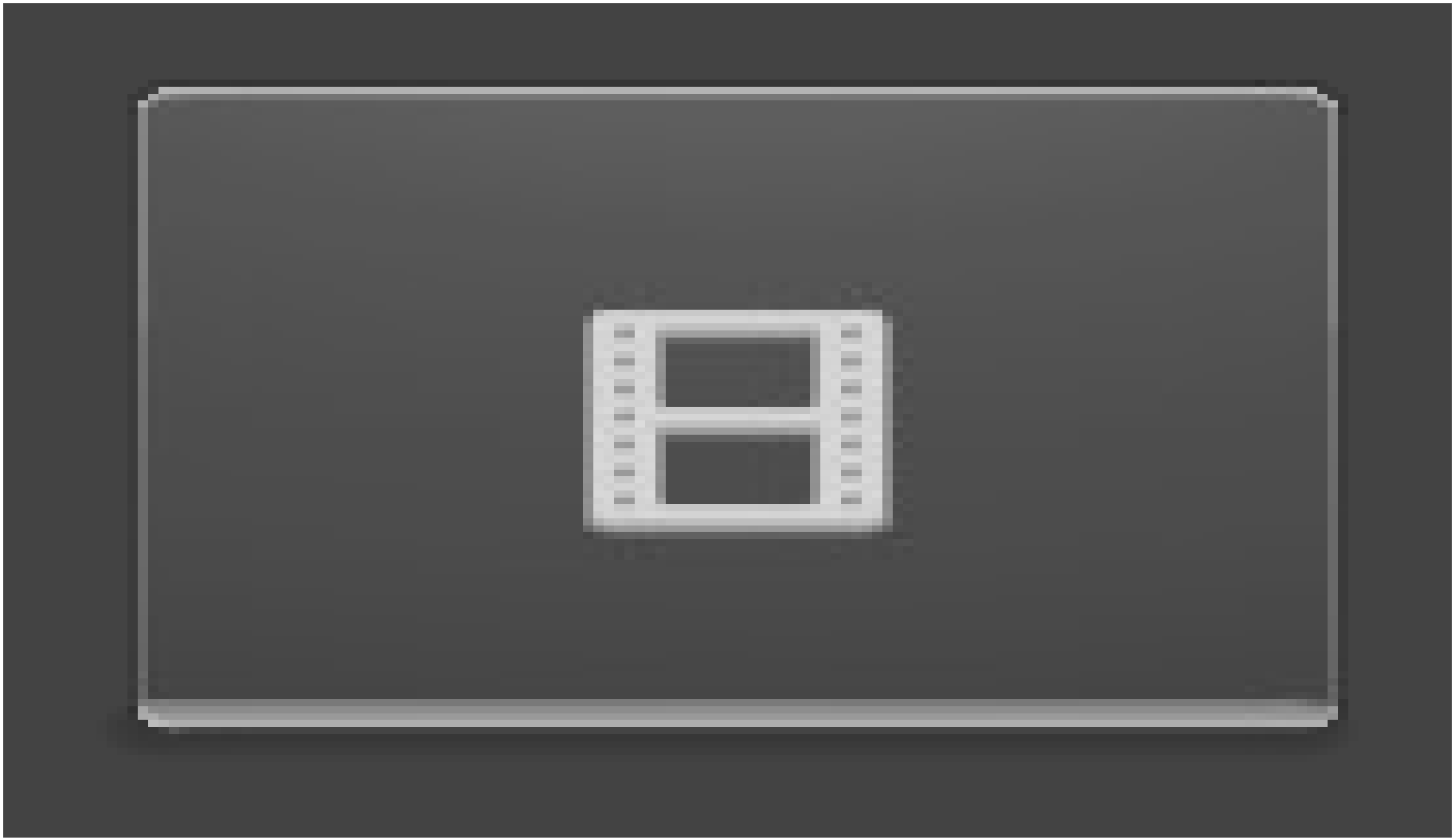



1965
HERBERT A. SIMON
20 YIL
2019

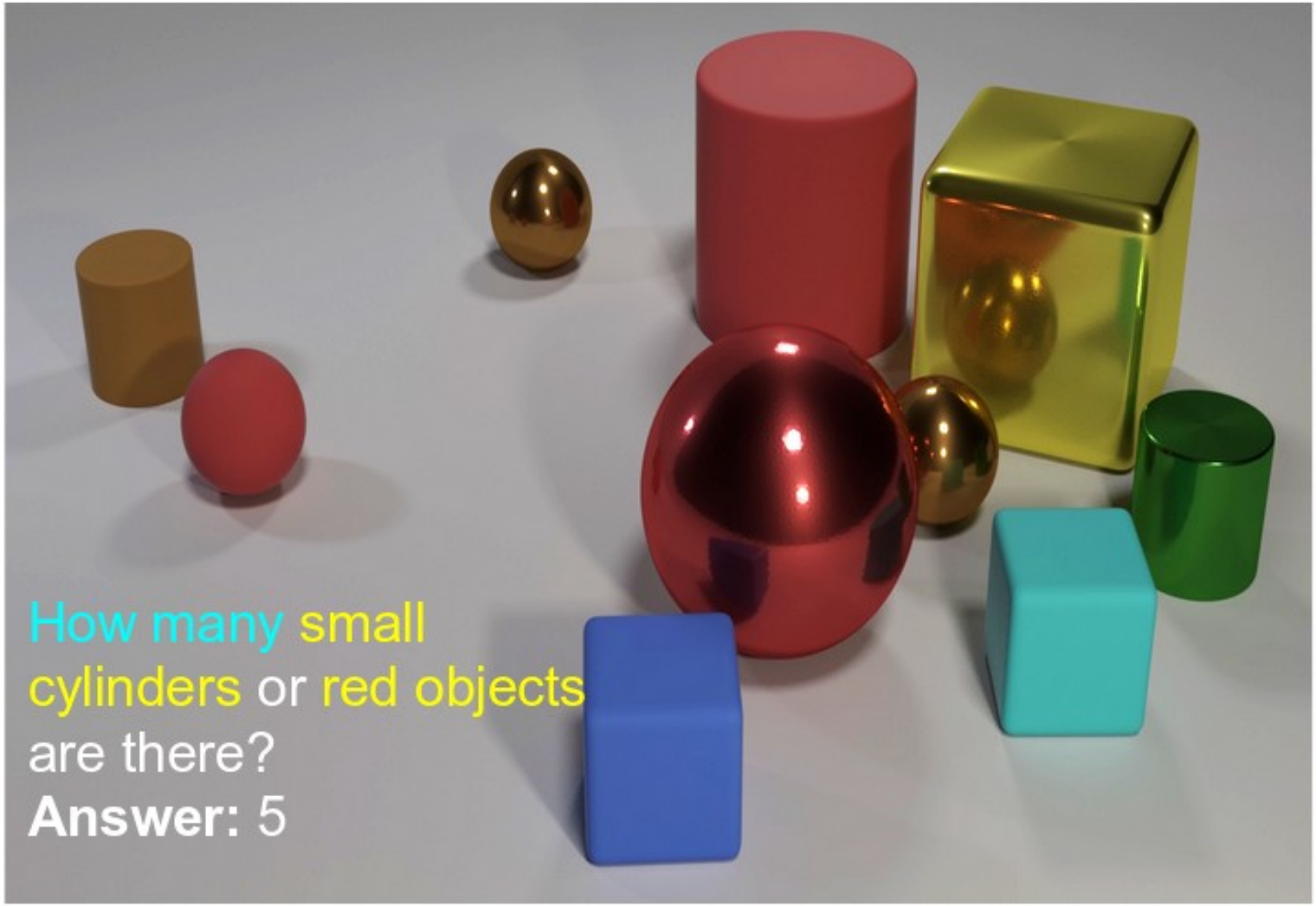

1950
ALAN TURING
1997
DEEP BLUE
GARRY
KASPAROV
DEEP
BLUE
2002
TAVLA
2006
SCRABBLE
2011
IBM WATSON
2017
ALPHAGO
AlphaGo
Lee Sedol

20 vs 361

Deepmind - Reinforcement Learning

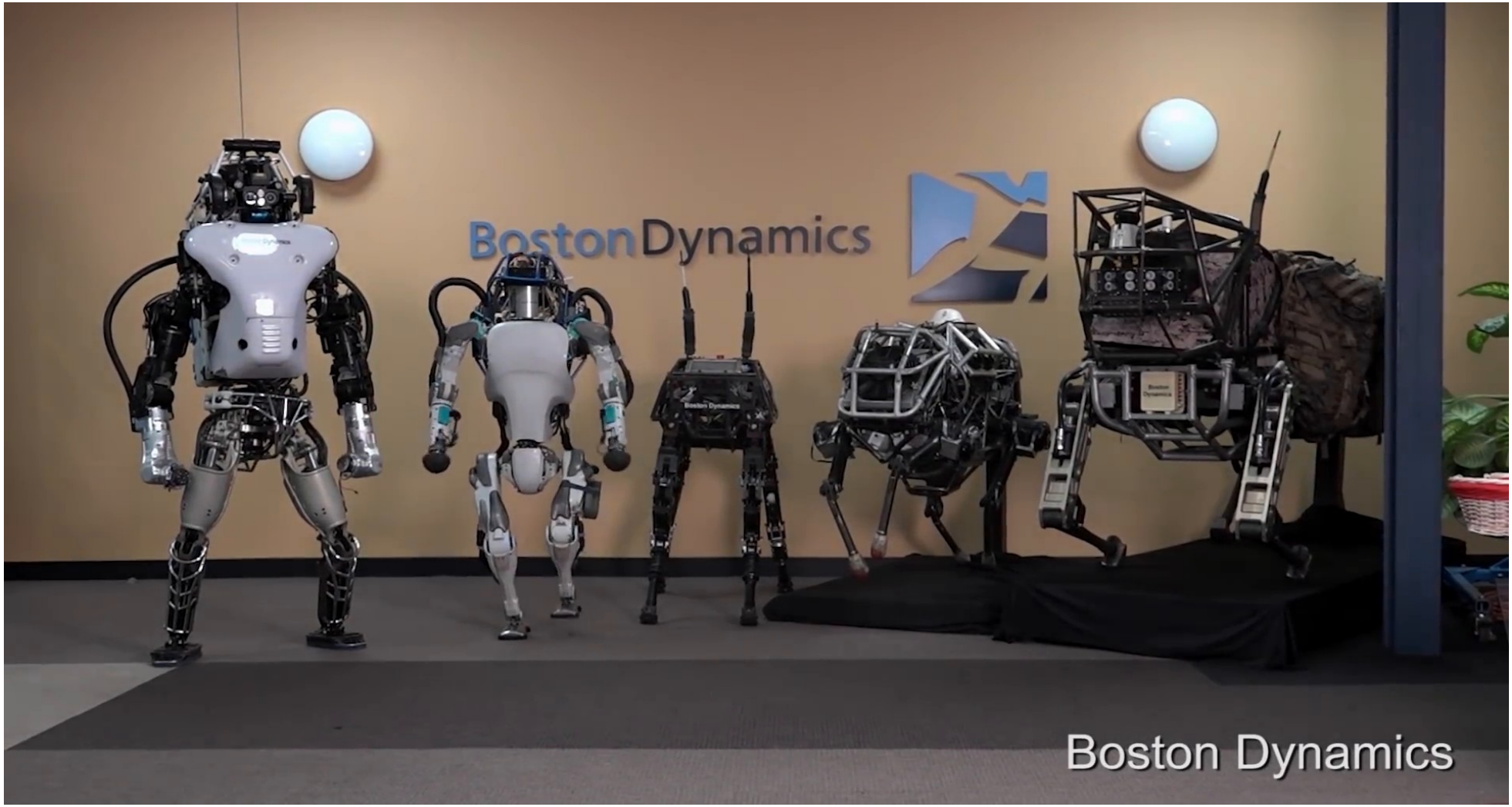


State-of-the-art in Natural Language Processing and Scene Recognition



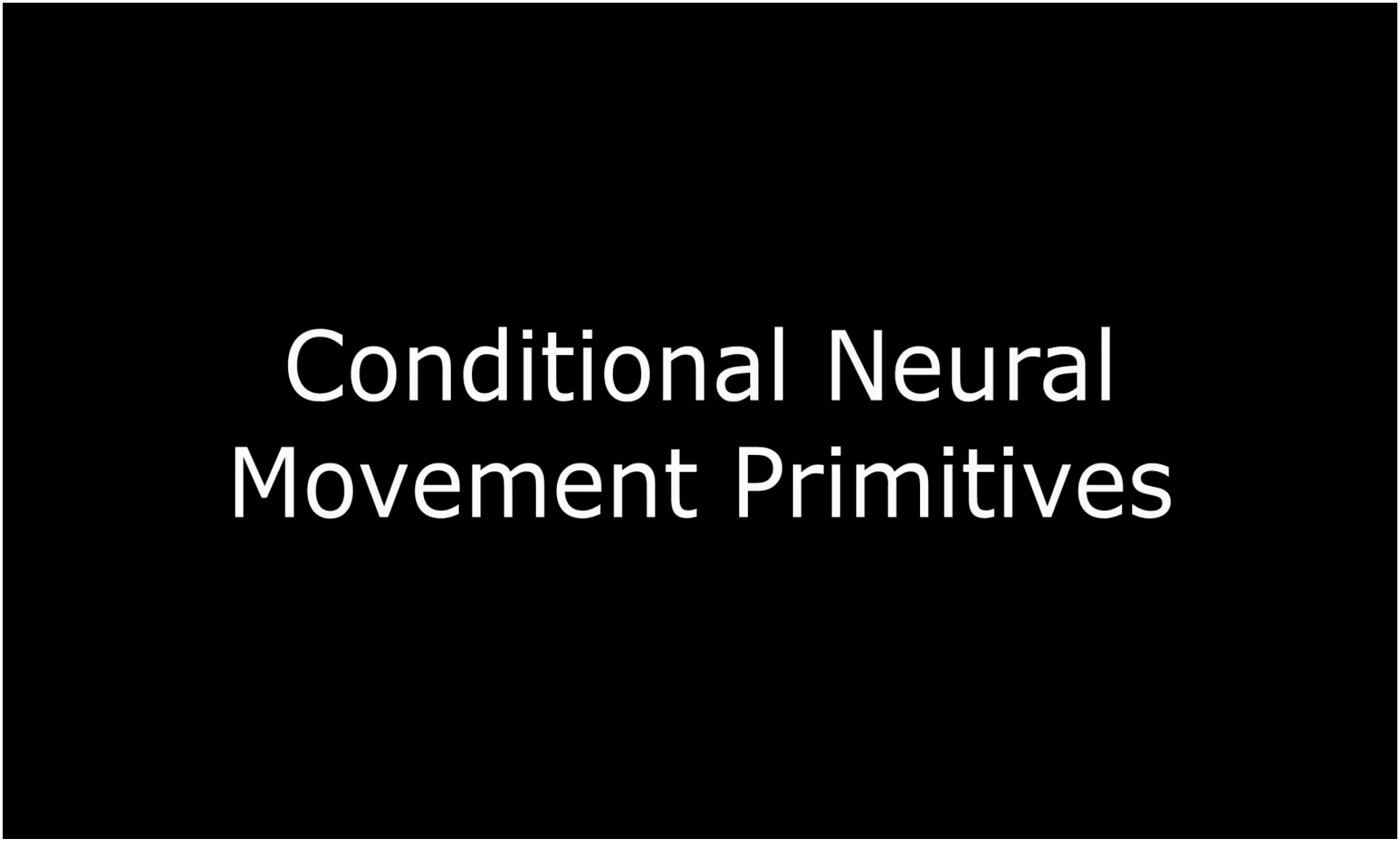
*adapted from Dr. Aykut Erdem

State-of-the-art in robotics



State-of-the-art in robotics

State-of-the-art in robotics



Acting rationally: rational agent

- Rational behavior: doing the right thing.
  - How do you define right thing?
  - The right thing: that which is expected to maximize expected goal achievement or outcome, given the available information

- Doesn't necessarily involve thinking, e.g.?
  - e.g., blinking reflex – but thinking should be in the service of rational action

Rational agents

- An agent is an entity that perceives and acts
- This course is about designing rational agents
- Abstractly, an agent is a function from percept histories to actions:
  - [f: P* → A]
- For any given class of environments and tasks, we seek the agent (or class of agents) with the best performance
- Caveat: computational limitations make perfect rationality unachievable
  - Design best program for given machine resources

State-of-the-art

Real-world interactions
Creativity

- ✓ Play a decent game of table tennis
- ✓ Drive safely along a curving mountain road
- ✗ Drive safely along Minibüs Caddesi
- ✓ Buy a week's worth of groceries on the web
- ✓ Play a decent game of bridge
- ✗ Discover and prove a new mathematical theorem
- ✗ Design and execute a research program in molecular biology
- ✗ Write an intentionally funny story
- ✓ Give competent legal advice in a specialized area of law
- ✓ Translate spoken English into spoken Swedish in real time
- ✗ Converse successfully with another person for an hour
- ✗ Perform a complex surgical operation
- ✓ Clean the floors without further guidance
- ✗ Unload any dishwasher and put everything away

Chapter 2:
Intelligent Agents

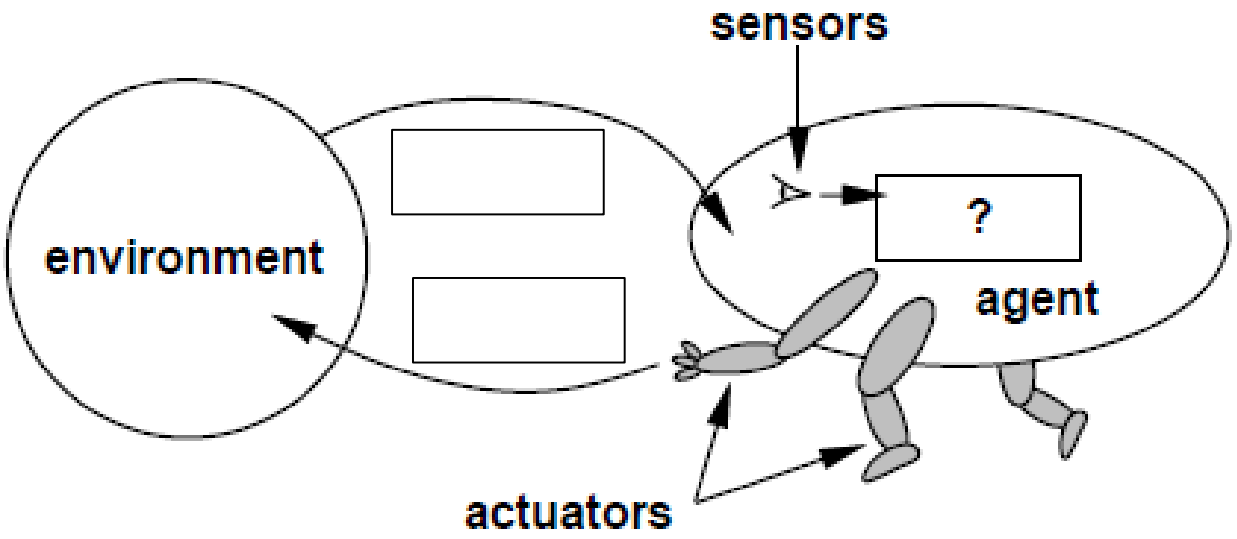
Overview

- PEAS (Performance, Environment, Actuators, Sensors)
- Environment types
- Agent functions and properties
- Agent types

What is an Intelligent Agent?

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators
- Human agent: eyes, ears, and other organs for sensors; hands, legs, mouth, and other body parts for actuators
- Robotic agent: cameras and infrared range finders for sensors; various motors for actuators

Agents



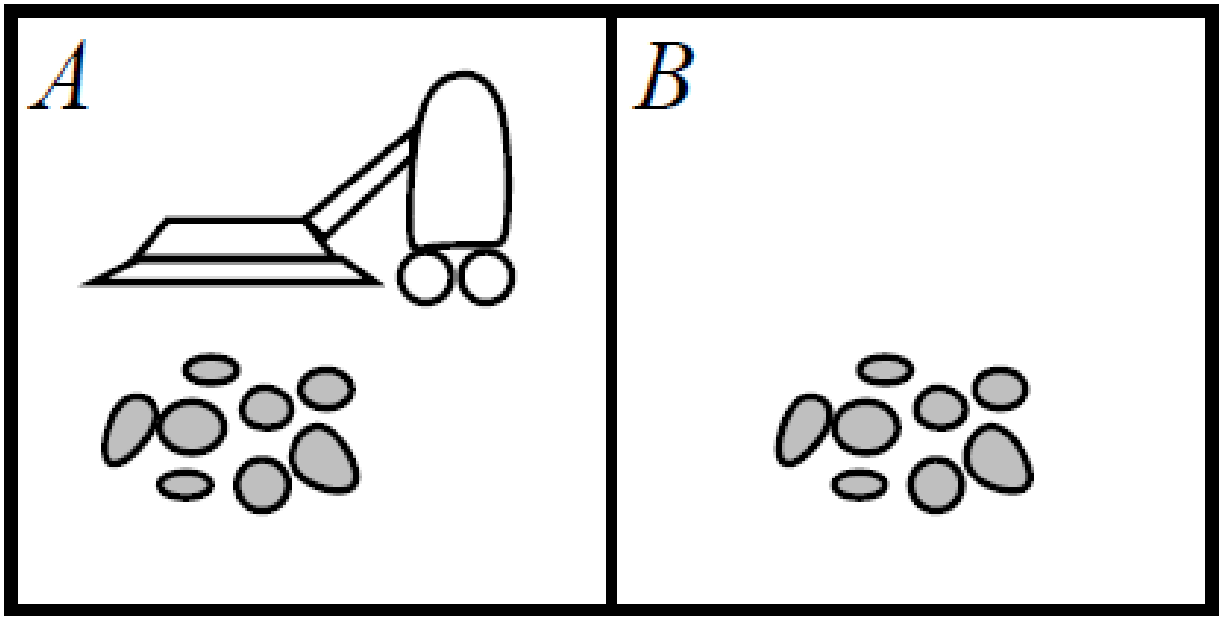
Agents include humans, robots, softbots, thermostats, etc.

The **agent function** maps from [ ] [ ] to [ ]

$f : P^* \rightarrow A$

The **agent program** runs on the physical **architecture** to produce f

Vacuum-cleaner world



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

Actions: *Left, Right, Suck, NoOp*

Simplest reflex agent?

A vacuum-cleaner agent

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

```
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
```

What is the **right** function?

Can it be implemented in a small agent program?

Rationality

- A rational agent is one that does the **right thing**.
- More precisely, what is rational at any given time depends on four things:
  - The performance measure that defines the criterion of success.
  - The agent's prior knowledge of the environment.
  - The actions that the agent can perform.
  - The agent's percept sequence to date.

Performance measure

- 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.
- Design measures according to what you want (as a behavior) and not according to what you think the agent should behave!
- Vacuum-cleaner?
  - Amount of dirt cleaned up.

Rationality

- Rationality maximizes expected performance while perfection maximizes actual performance.

Task Environment

- PEAS: Performance measure, Environment, Actuators, Sensors
- Must first specify the setting for intelligent agent odometer, engine sensors, ?

- Consider, e.g., the task of designing an automated taxi driver ("beyond the capabilities of existing technology") :
- Performance measure:
  -
- Environment:
  -
- Actuators:
  -
- Sensors:
  -

Task Environment

- PEAS: Performance measure, Environment, Actuators, Sensors
- Must first specify the setting for intelligent agent odometer, engine sensors, ?

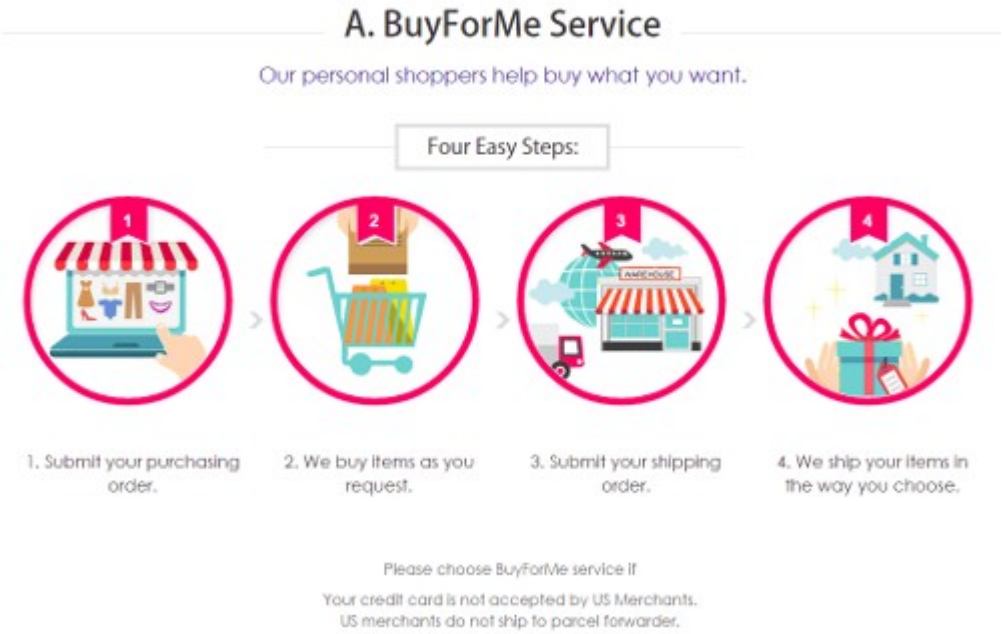
- Consider, e.g., the task of designing an automated taxi driver ("beyond the capabilities of existing technology") :
- Performance measure:
  - Safe, fast, legal, comfortable trip, maximize profits and ?
- Environment:
  - Roads, other traffic (?), pedestrians, customers, weather
- Actuators:
  - Steering wheel, accelerator, brake, signal, horn, ?
- Sensors:
  - Cameras, sonar, speedometer, GPS, odometer, engine sensors, ?

PEAS for Internet shopping agent

- Performance Measure?
  -
- Environment?
  -
- Actuators?
  -
- Sensors?
  -

PEAS for Internet shopping agent

- Performance Measure?
  - price, quality, appropriateness, efficiency
- Environment?
  - current and future WWW sites, vendors, shippers
- Actuators?
  - display to user, follow URL, fill in form
- Sensors?
  - HTML pages (text, graphics, scripts)



PEAS for Part-picking robot

- Performance measure:
  - –
- Environment:
  - –
- Actuators:
  - –
- Sensors:
  - –

PEAS for Part-picking robot

- Performance measure:
  - Percentage of parts in correct bins
- Environment:
  - Conveyor belt with parts,bins
- Actuators:
  - Jointed arm and hand
- Sensors:
  - Camera, joint angle sensors

Agent Characteristics

- **Embodiment**
- **Situatedness**
- **Autonomy**
- **Adaptivity**
- **Sociability**

Agent Characteristics

- **Situatedness:** The agent receives some form of sensory input from its environment, and it performs some action that changes its environment in some way. Examples of environments: the physical world and the Internet.
- **Embodiment:** Having a physical body
- **Autonomy:** The agent can act without direct intervention by humans or other agents and that it has control over its own actions and internal state.

Agent Characteristics

- **Adaptivity :** The agent is capable of
  - (1) reacting flexibly to changes in its environment
  - (2) taking goal directed initiative (i.e., is pro-active), when appropriate;
  - and (3) learning from its own experience, its environment, and interactions with others.
- **Sociability:** The agent is capable of interacting in a peer-to-peer manner with other agents or humans.

Environment Types – Categorize in different dimensions

- **Fully observable** vs. **partially observable**.
  -
- **Deterministic** vs. **stochastic**.
  -
  -
  -
- **Episodic** vs. **sequential**.
  -

Environment Types – Categorize in different dimensions

- **Fully observable** vs. **partially observable**.
  - If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable. Example?
- **Deterministic** vs. **stochastic**.
  - Guaranteed effect.
  - If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic; otherwise it is stochastic. Example? Board games? Chess, backgammon
- **Episodic** vs. **sequential**.
  - In an episodic task environment, the agent's experience is divided into atomic "episodes."
  - Episode: Single cycle of an agent perceiving and taking an action
  - Episodic: If the choice depends on the current episode and not on previous episodes
    - Easier to operate.
  - In sequential environments, the current decision may affect all future decisions. Examples? Chess?

Environment Types

- **Static** vs. **dynamic**.
- **Discrete** vs. **continuous**.
  -
- **Single-agent** vs. **multi-agent**.
  -

Environment Types

- **Static** vs. **dynamic**.
  - If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static.
  - If the environment itself does not change with the passage of time but the agent's **performance score** does, then we say the environment is semidynamic. Example?
- **Discrete** vs. **continuous**.
  - The discrete/continuous distinction can be applied to the *state* of the environment, to the way *time* is handled, and to the *percepts* and *actions* of the agent. Example?
  - Possible to convert continuous environments into discrete environments (with loss of precision)
- **Single-agent** vs. **multi-agent**.
  - Taxi driver? Whether B's behavior is best described as maximizing a performance measure depending on A's performance measure
  - Competitive
  - Cooperative

Environment types

	Backgammon	Internet Shopping	Taxi
Observable?			
Deterministic?			
Episodic?			
Static?			
Discrete?			
Single-agent?			

Environment types

	Backgammon	Internet Shopping	Taxi
Observable?	Yes	No	No
Deterministic?	No	Partly	No
Episodic?	No	No	No
Static?	Yes	Semi	No
Discrete?	Yes	Yes	No
Single-agent?	No	Yes (except auctions)	No

The environment type largely determines the agent design. The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent

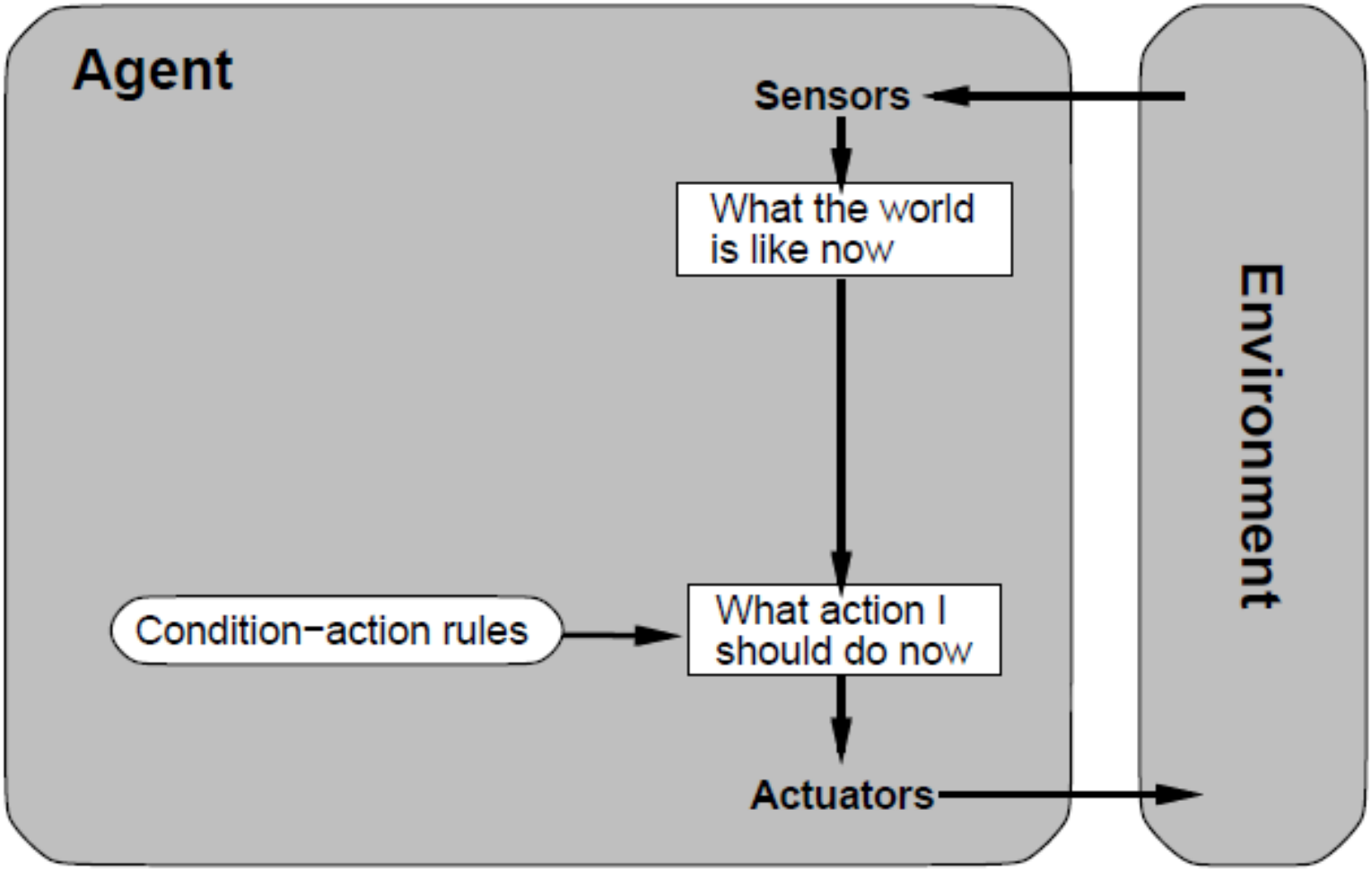
Structure of agent

- **Agent = architecture + program**
- **Agent Function:**
  - Mathematically speaking, we say that an agent's behavior is described by the agent function that maps any given percept sequence to an action.
- **Agent Program:**
  - Take the current percept as input and return action.
  - The implementation of the agent function for an artificial agent is called the agent program.
  - Why only current percept?
    - Only information provided by the environment.

Agent types

- Four basic agent types in order of increasing generality:
  - simple reflex agents
  - reflex agents with state
  - goal-based agents
  - utility-based agents
- All these can be turned into learning agents

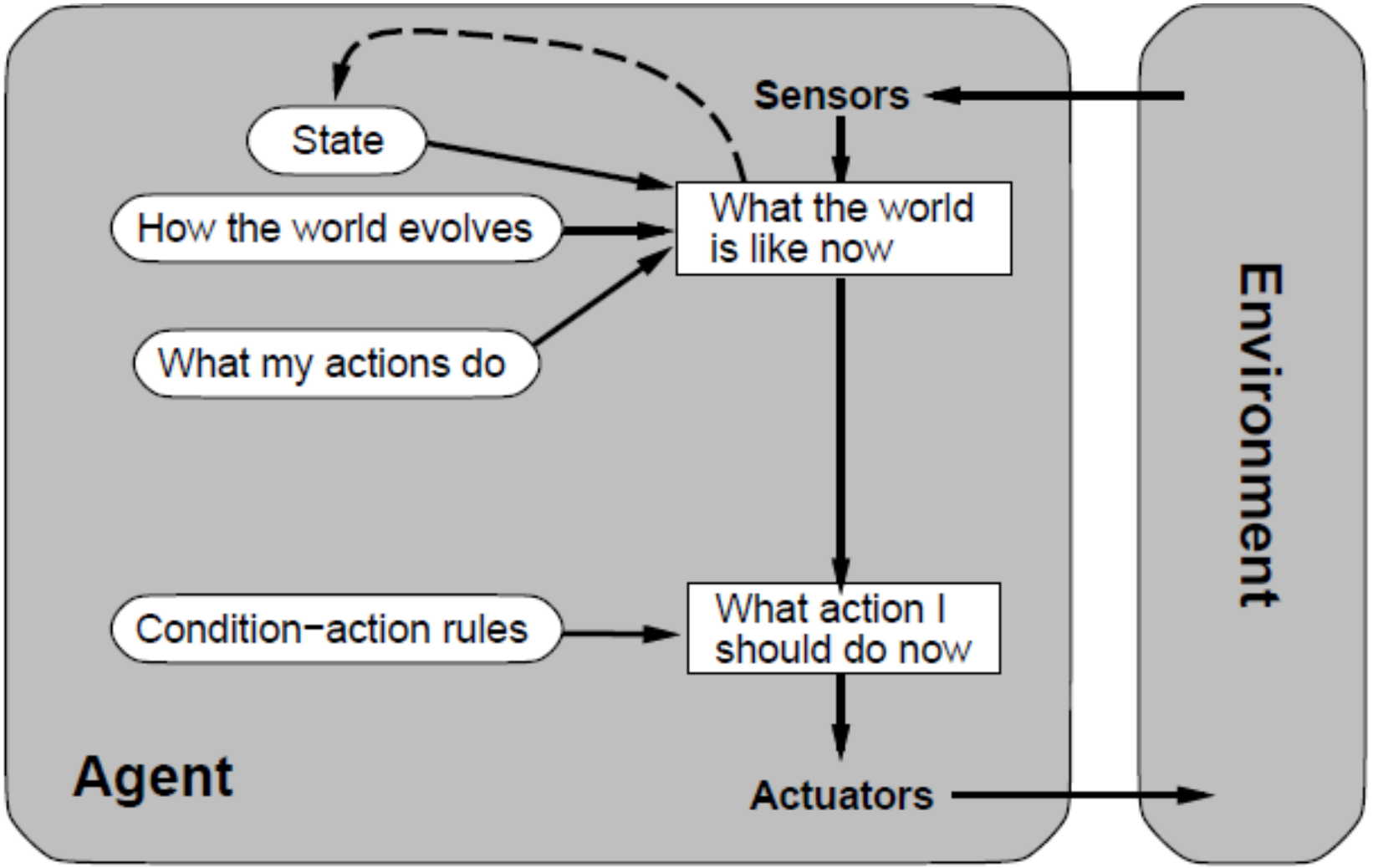
Simple Reflex Agents



Simple Reflex Agents

- Select actions based only on the current percept, ignoring the rest of the percept history.
- Table lookup of percept-action pairs defining all possible condition-action rules necessary to interact in an environment
- Problems
  - Possible condition-action rules too big to generate and to store (Chess has about 10^{120} states, for example)
  - No knowledge of non-perceptual parts of the current state
    - Get into loops - randomize
  - Not adaptive to changes in the environment; requires entire table to be updated if changes occur

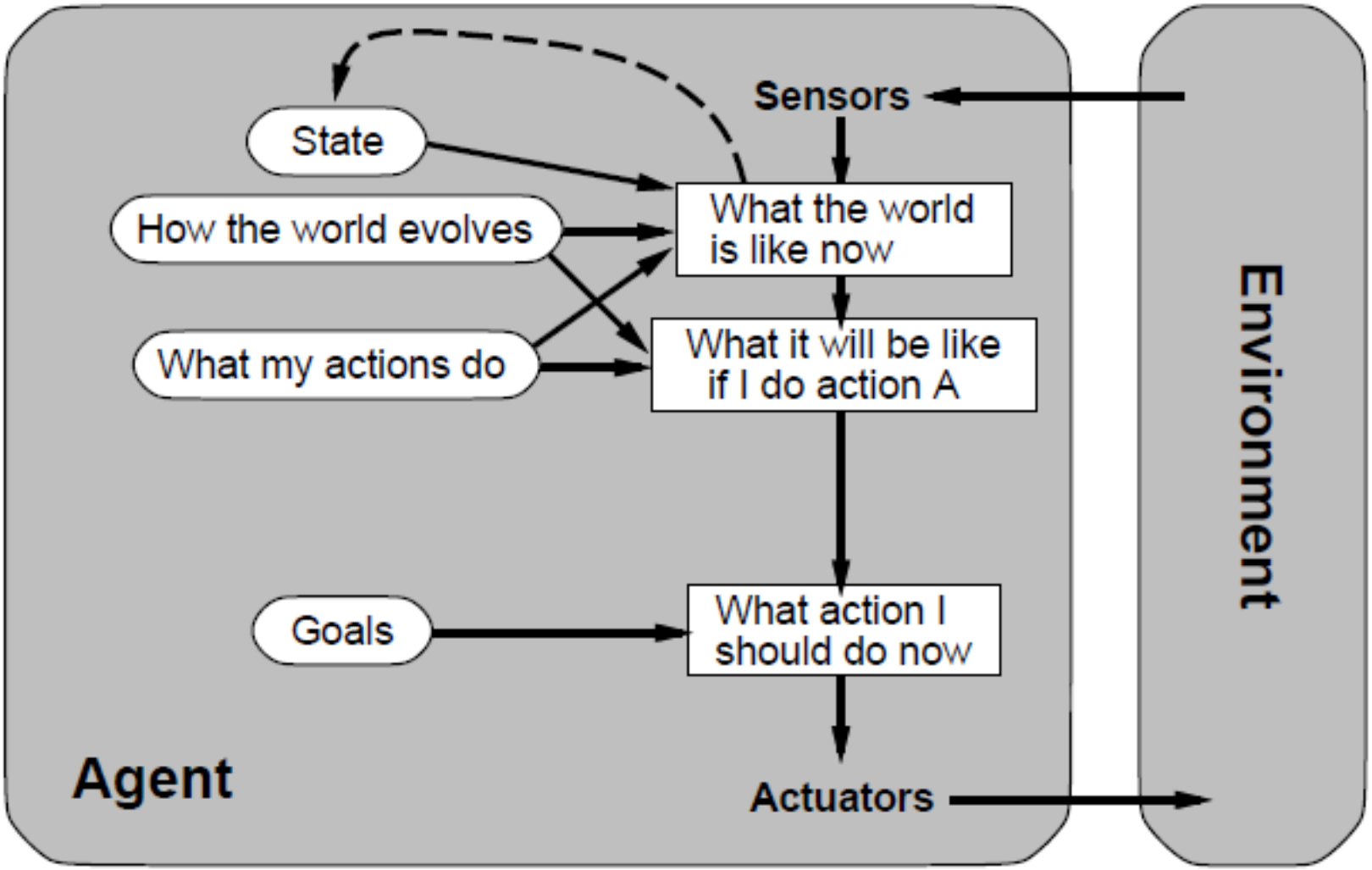
Reflex Agents with state



Reflex Agent with State
Model-based reflex agents

- The knowledge about “how the world works” is called a model of the world.
- An agent that uses such a model is called a model-based agent.
- Encode "internal state" of the world to remember the past as contained in earlier percepts
- **Needed** because sensors do not usually give the entire state of the world at each input (what did we call this environment?), so perception of the environment is captured over time.
- Requires ability to represent change in the world; one possibility is to represent just the latest state, but then can't reason about hypothetical courses of action
  - Ability to “model” how world evolves independent of agent – other driver
  - Ability to “model” how world evolves as a result of agent actions

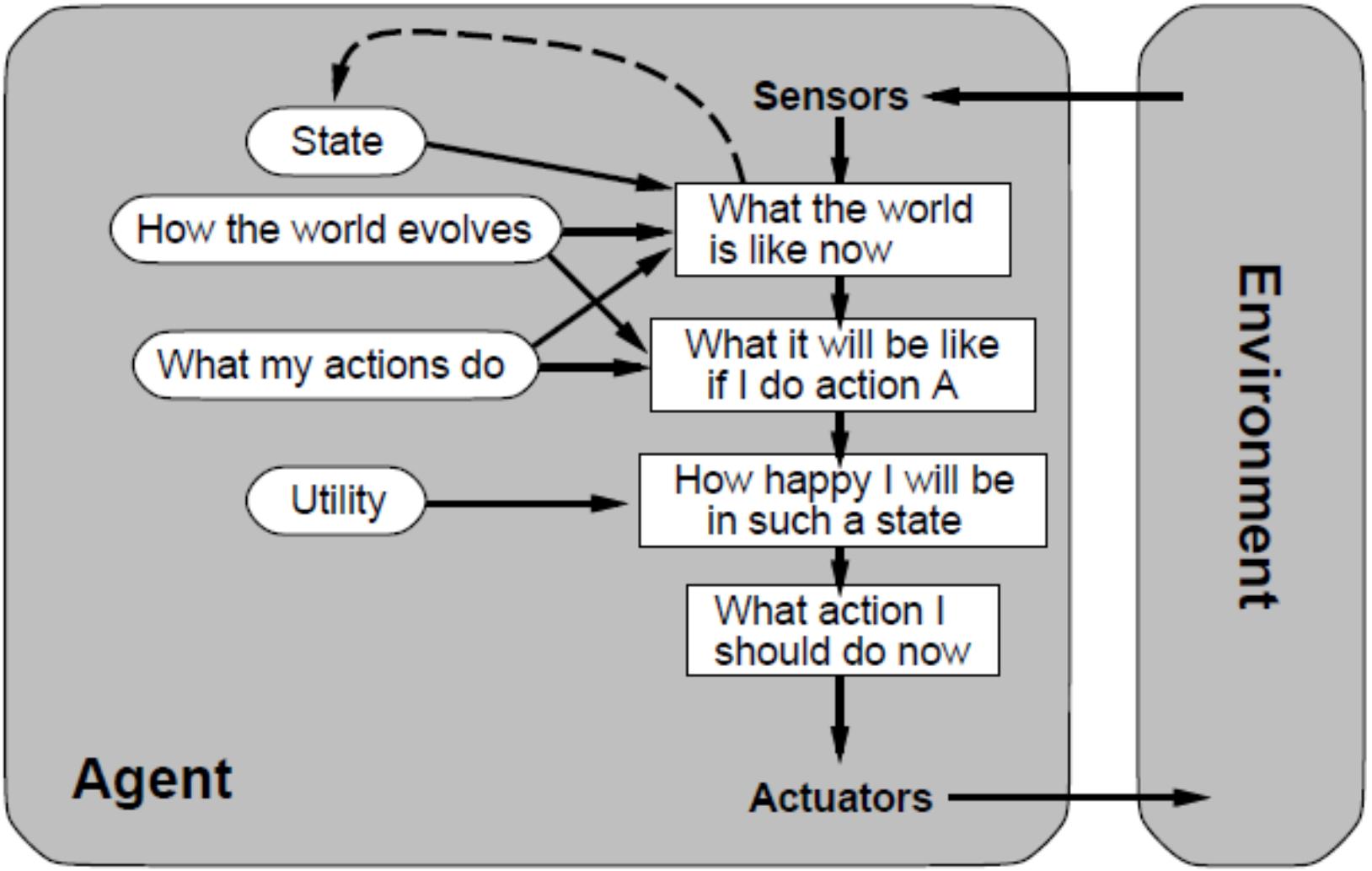
Goal-based Agent



Goal-based Agents

- A taxi in a junction, turn left or right?
- Choose actions so as to achieve a (given or computed) goal = a description of a desirable situation
- Keeping track of the current state is often not enough --- need to add goals to decide which situations are good
  - Think about a **goal** state where the desired goal holds
  - **Plan** or **search** a sequence of actions such that applying those actions will transform the current state into the goal state
  - Combine goal info and possible next states.
- Deliberative instead of reactive. Not reflex anymore.
- May have to consider long sequences of possible actions before deciding if goal is achieved --- involves consideration of the future, "what will happen if I do...?"
- Goal-based agents are more flexible – no need to rewrite lookup tables.

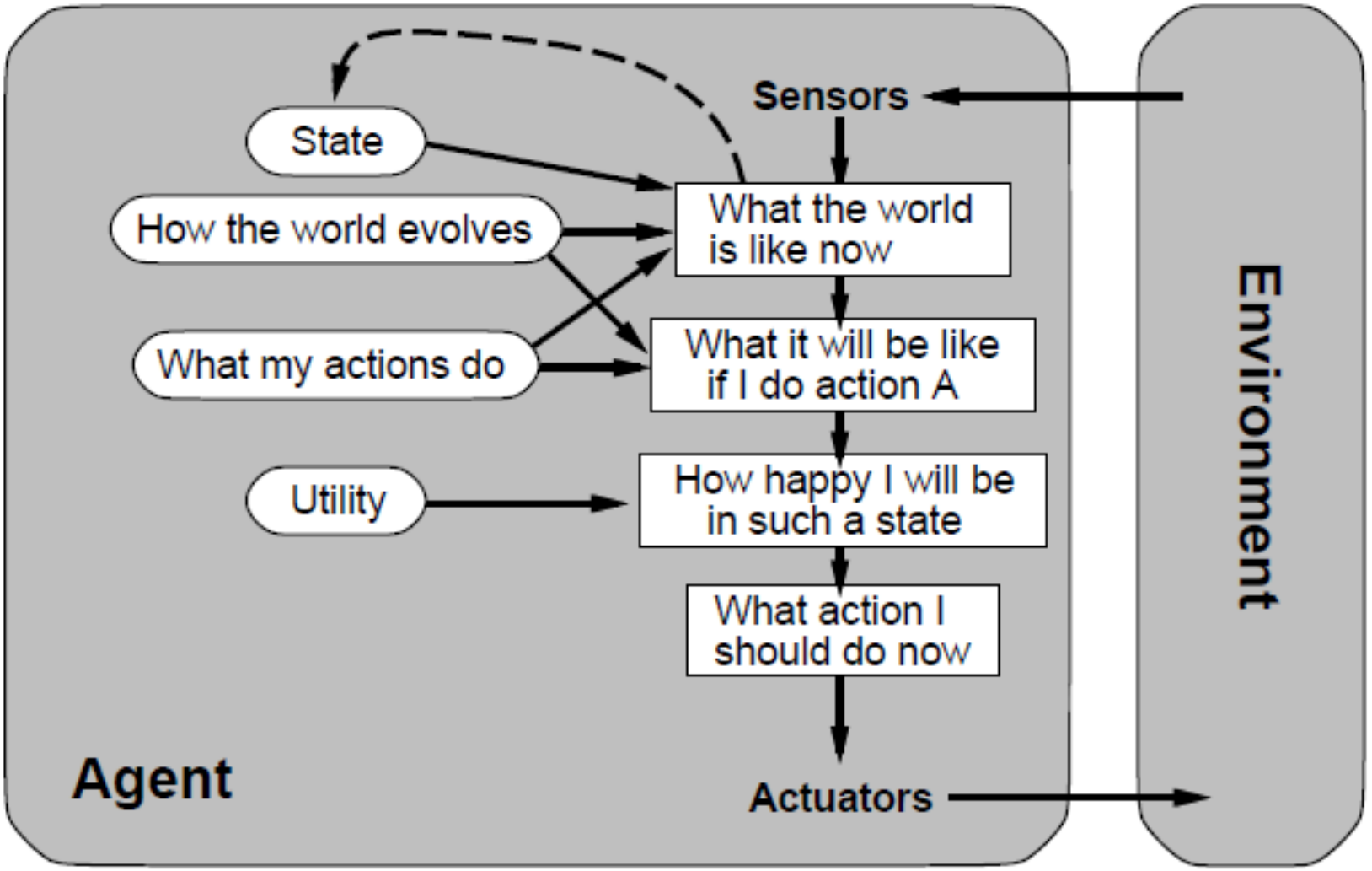
Utility-based Agents



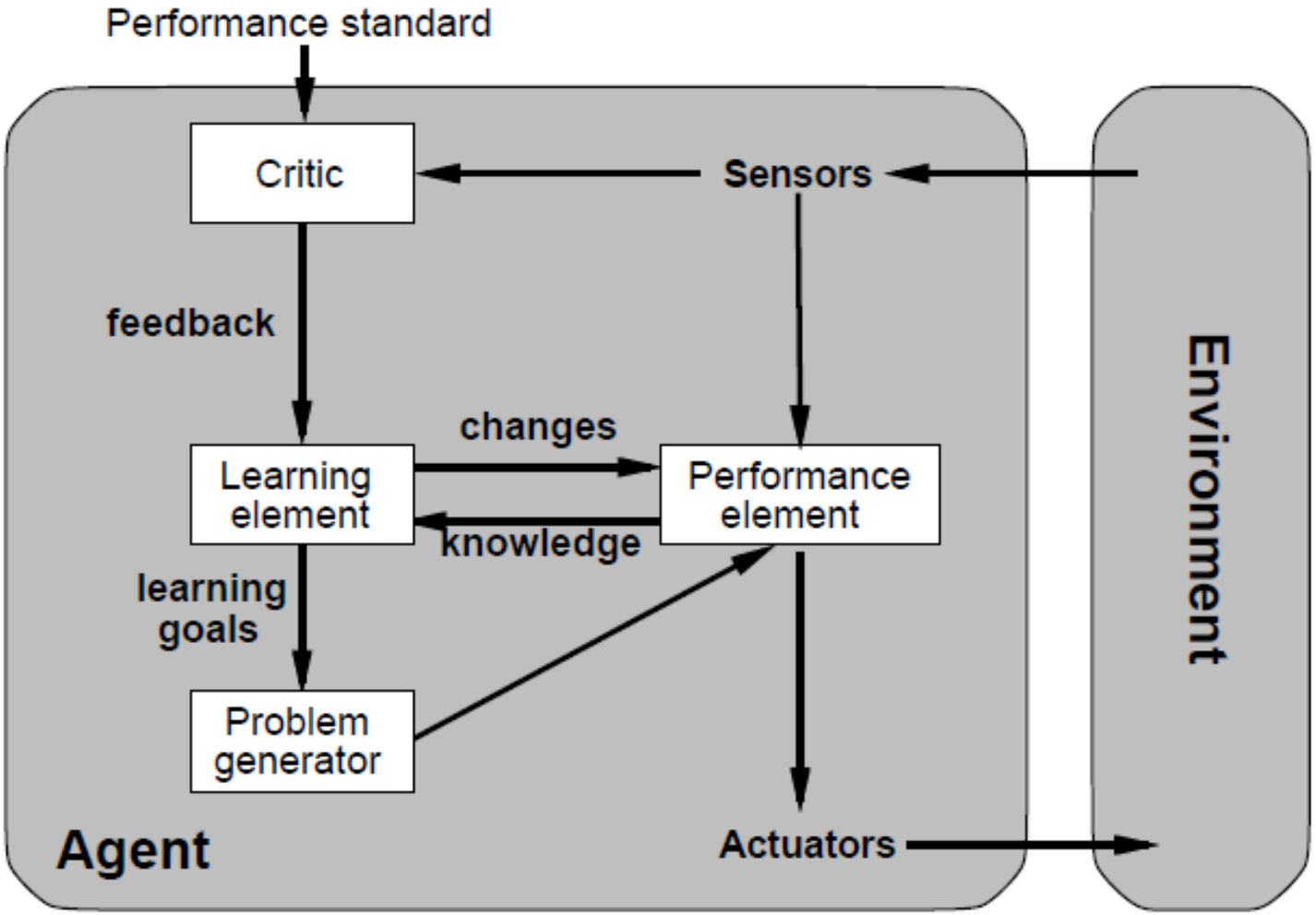
Utility-based Agents

- When there are multiple possible alternatives, how to decide which one is best?
- A goal specifies a crude distinction between a happy and unhappy state, but often need a more general performance measure that describes "degree of happiness"
- **Goals alone are not really enough.**
  - Taxi: different sequences to the same goal are safer, quicker, more reliable or cheaper.
- **Utility function** U: State → Reals
  - indicates a measure of success or happiness when at a given state
  - A state has higher utility if it is preferred over another.
  - Allows decisions comparing choice between conflicting goals, and choice between likelihood of success and importance of goal (if achievement is uncertain)

What is missing?



Learning Agents



Learning agents - components

- **learning element**, which is responsible for making improvements,
- **performance element**, which is responsible for selecting external actions. The performance element is the entire agent: it takes in percepts and decides on actions.
- **critic** gives feedback from the on how the agent is doing with respect to a fixed performance standard.
- **problem generator** is responsible for suggesting actions that will lead to new and informative experiences.

Summary

- Agents interact with environments through actuators and sensors
- The agent function describes what the agent does in all circumstances
- The performance measure evaluates the environment sequence
- A perfectly rational agent maximizes expected performance

Summary

- Agent programs implement (some) agent functions
- PEAS descriptions define task environments
- Environments are categorized along several dimensions:
  - Observable? Deterministic? Episodic? Static? Discrete? Single-agent?
- Several basic agent architectures exist:
  - Reflex, reflex with state, goal-based, utility-based
  - Learning vs. non-learning

Problem solving and search

Chapter 3

Week 1: Uninformed Search

No-problem specific information

Outline

- ♢ Problem-solving agents
- ♢ Problem types
- ♢ Problem formulation
- ♢ Example problems
- ♢ Basic search algorithms

Problem-solving agents

Restricted form of general agent:

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
  action ← RECOMMENDATION(seq, state)
  seq ← REMAINDER(seq, state)
  return action
```

Note: this is offline problem solving; solution executed "eyes closed."
Online problem solving involves acting without complete knowledge.

open-loop

Example: Romania

On holiday in Romania; currently in Arad.
Flight leaves tomorrow from Bucharest

Formulate goal:

be in Bucharest

Formulate problem:

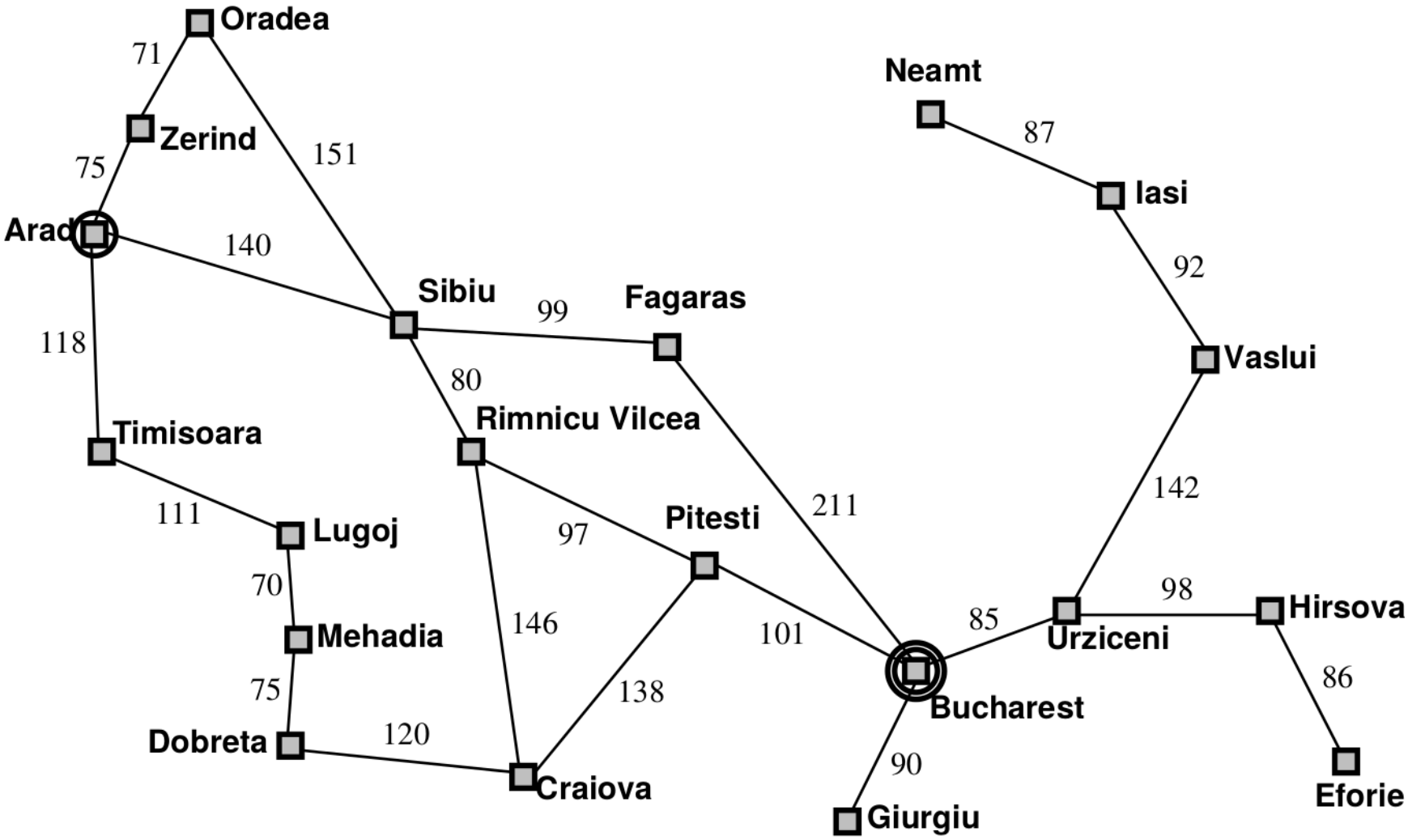
states: various cities

actions: drive between cities

Find solution:

sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Example: Romania



Problem types

Deterministic, fully observable $\Longrightarrow$ single-state problem

Agent knows exactly which state it will be in; solution is a sequence

Non-observable $\Longrightarrow$ conformant problem

Agent may have no idea where it is; solution (if any) is a sequence

Nondeterministic and/or partially observable $\Longrightarrow$ contingency problem

percepts provide **new** information about current state

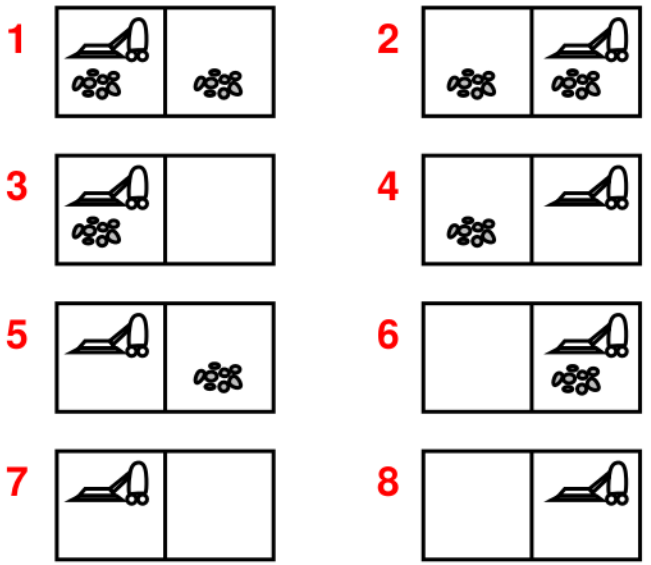
solution is a contingent plan or a policy

often **interleave** search, execution

Unknown state space $\Longrightarrow$ exploration problem ("online")

Example: vacuum world

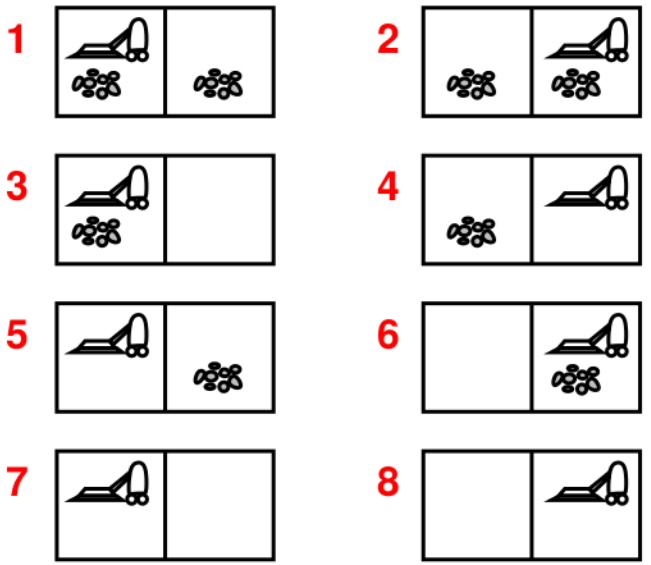
Single-state, start in #5. Solution??



Example: vacuum world

Single-state, start in #5. Solution??
$[Right, Suck]$

Conformant, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
e.g., $Right$ goes to $\{2, 4, 6, 8\}$. Solution??

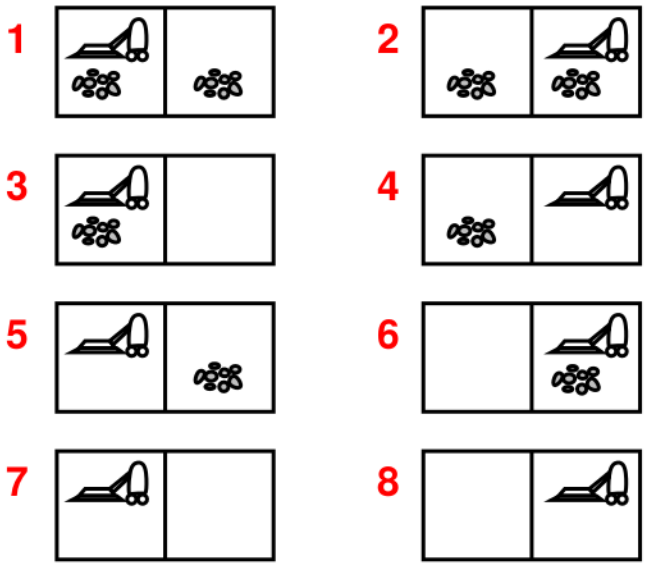


Example: vacuum world

Single-state, start in #5. Solution??
$[Right, Suck]$

Conformant, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
e.g., $Right$ goes to $\{2, 4, 6, 8\}$. Solution??
$[Right, Suck, Left, Suck]$

Contingency, start in #5
Murphy's Law: $Suck$ can dirty a clean carpet
Local sensing: dirt, location only.
Solution??

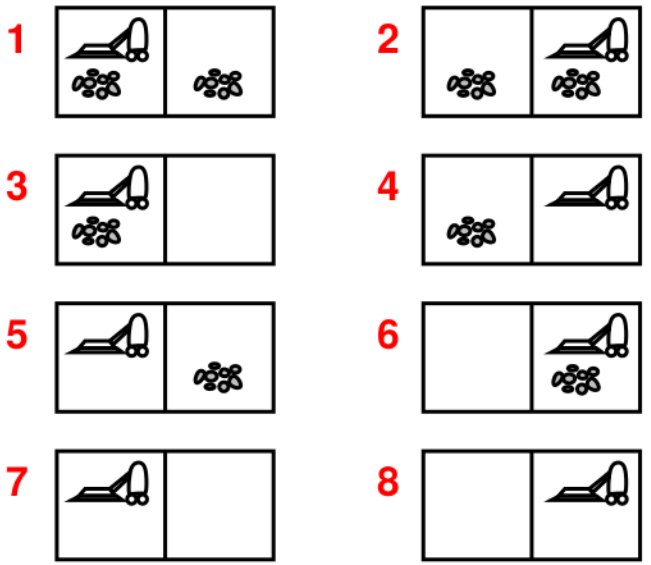


Example: vacuum world

Single-state, start in #5. Solution??
$[Right, Suck]$

Conformant, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
e.g., $Right$ goes to $\{2, 4, 6, 8\}$. Solution??
$[Right, Suck, Left, Suck]$

Contingency, start in #5
Murphy's Law: $Suck$ can dirty a clean carpet
Local sensing: dirt, location only.
Solution??
$[Right, \textbf{if } dirt \textbf{ then } Suck]$



Single-state problem formulation

A problem is defined by four items:

initial state e.g., "at Arad"

successor function $S(x)$ = set of action–state pairs
e.g., $S(Arad) = \{\langle Arad \rightarrow Zerind, Zerind\rangle, \ldots\}$

goal test, can be
explicit, e.g., $x =$ "at Bucharest"
implicit, e.g., $NoDirt(x)$

path cost (additive)
e.g., sum of distances, number of actions executed, etc.
$c(x, a, y)$ is the step cost, assumed to be ≥ 0

A solution is a sequence of actions
leading from the initial state to a goal state

Selecting a state space

Real world is absurdly complex

$\Rightarrow$ state space must be **abstracted** for problem solving

(Abstract) state = set of real states

(Abstract) action = complex combination of real actions

e.g., "Arad $\rightarrow$ Zerind" represents a complex set of possible routes, detours, rest stops, etc.

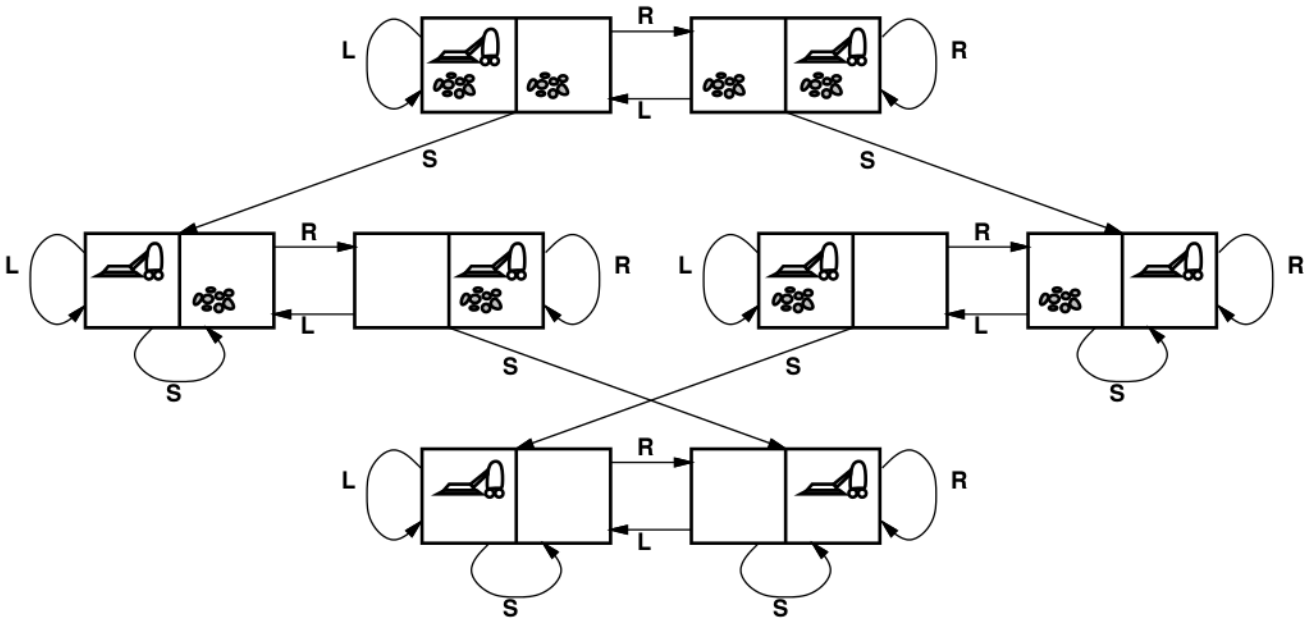
For guaranteed realizability, **any** real state "in Arad" must get to some real state "in Zerind"

(Abstract) solution =

set of real paths that are solutions in the real world

Each abstract action should be "easier" than the original problem!

Example: vacuum world state space graph



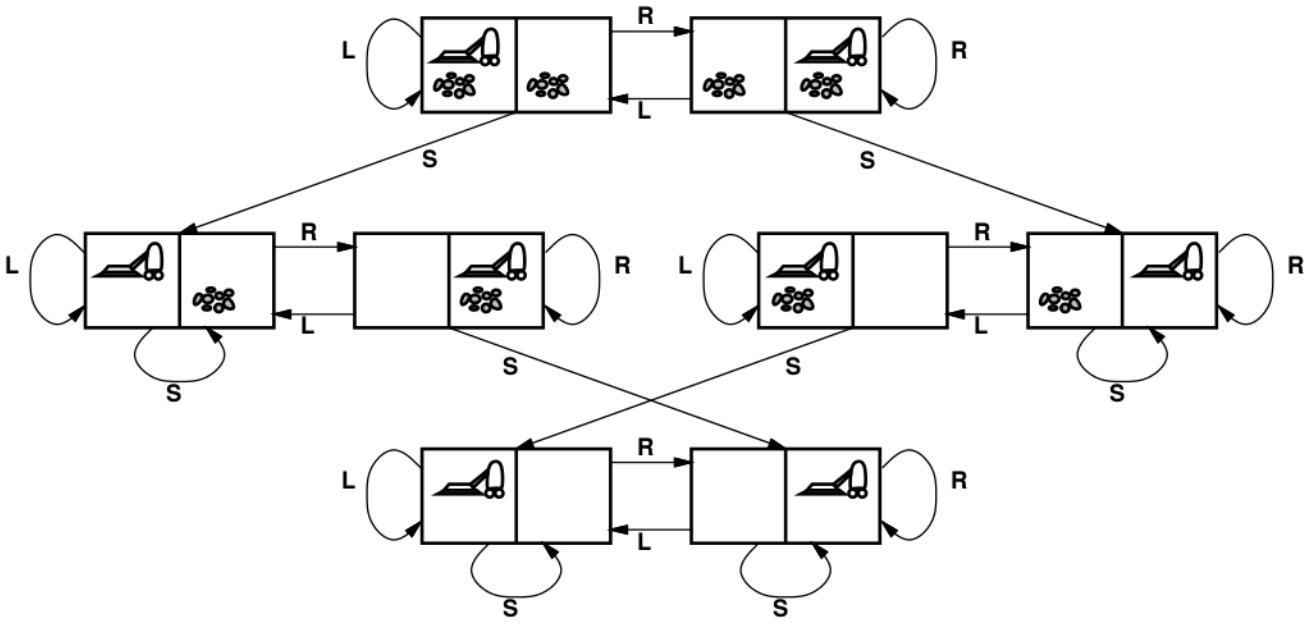
states??

actions??

goal test??

path cost??

Example: vacuum world state space graph



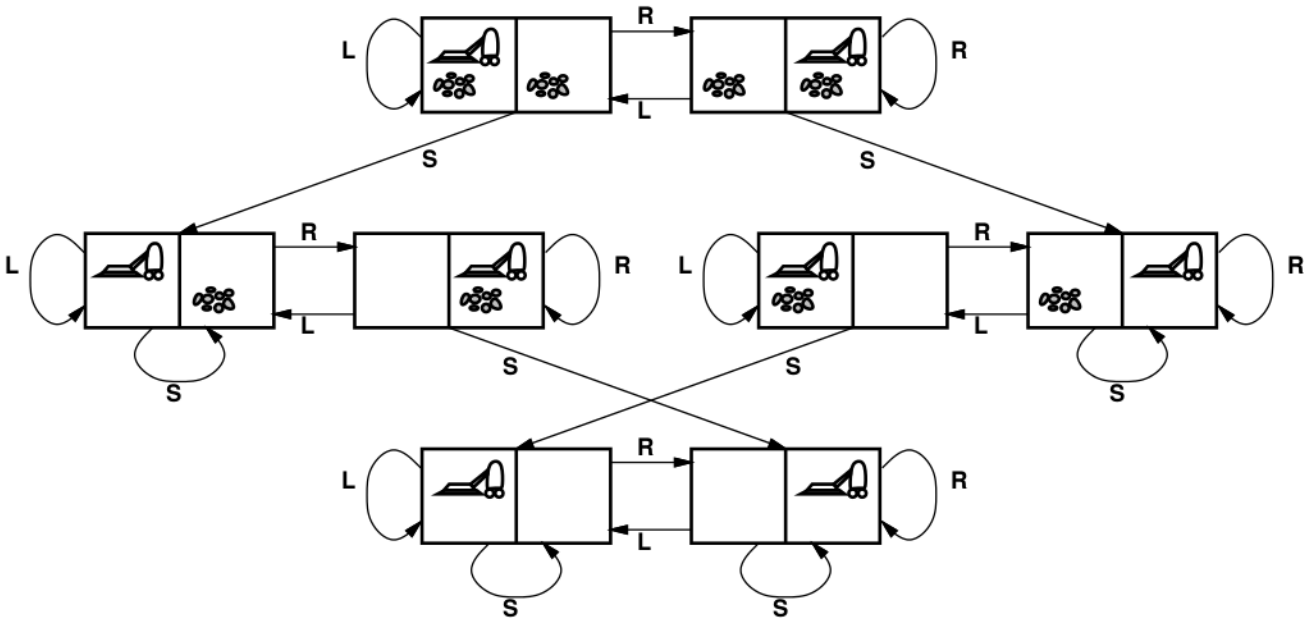
states??: integer dirt and robot locations (ignore dirt amounts etc.)

actions??

goal test??

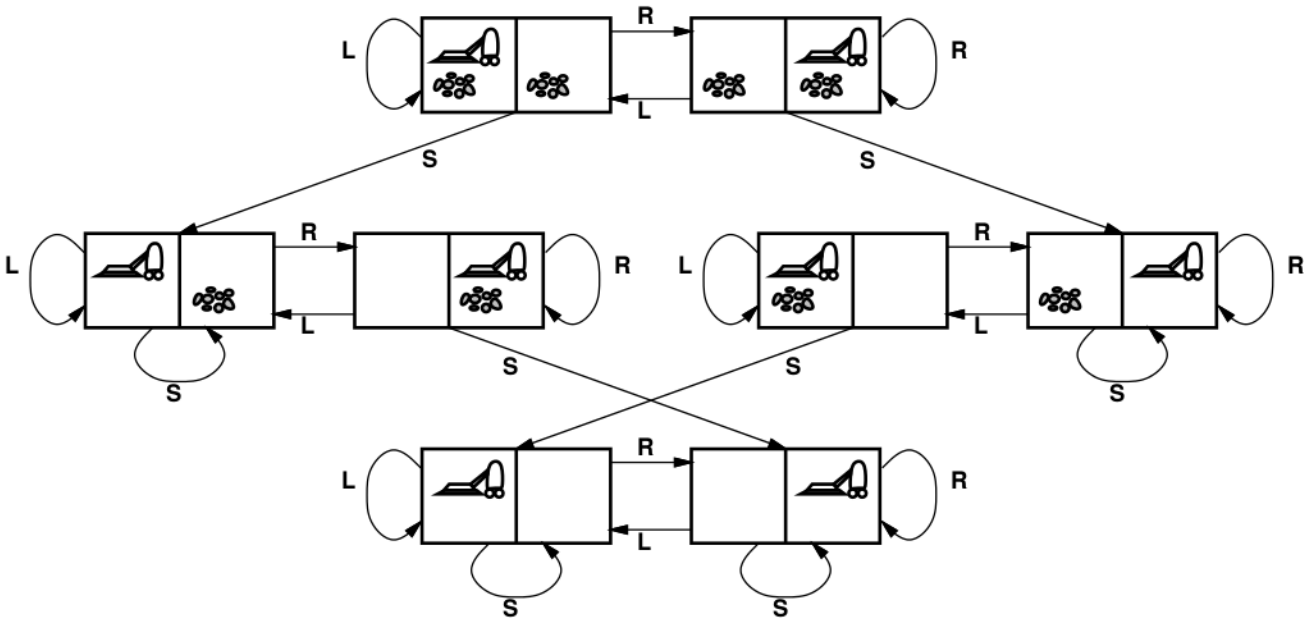
path cost??

Example: vacuum world state space graph



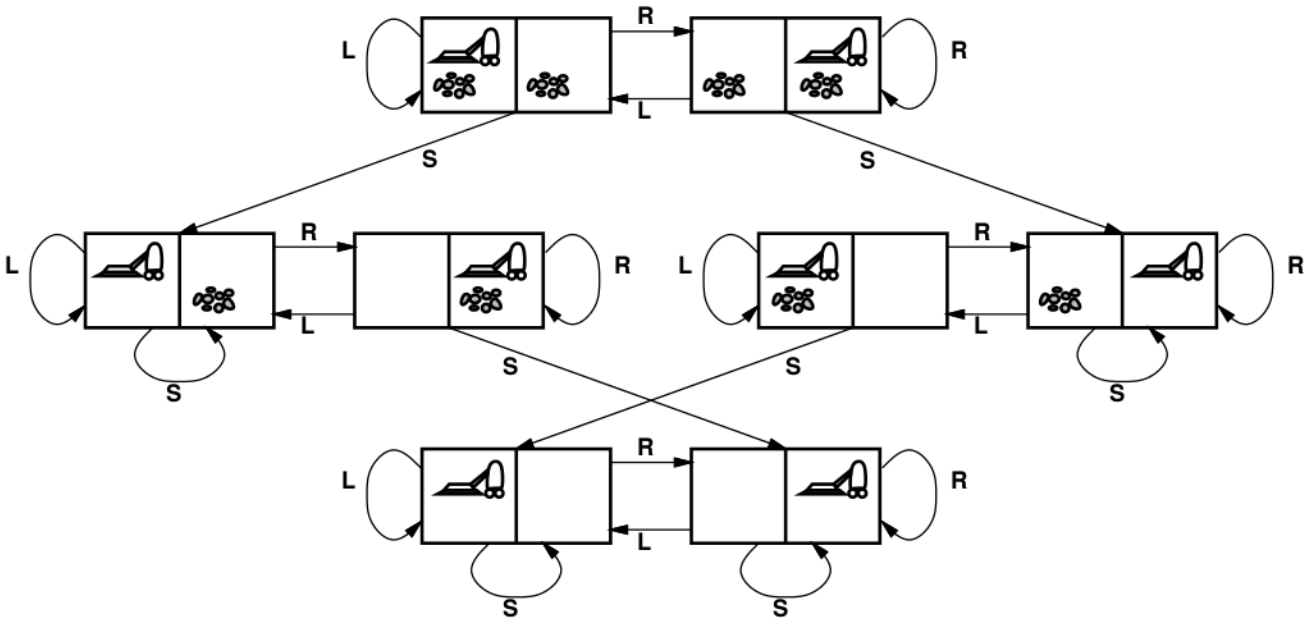
states??: integer dirt and robot locations (ignore dirt amounts etc.)
actions??: $Left$, $Right$, $Suck$, $NoOp$
goal test??
path cost??

Example: vacuum world state space graph



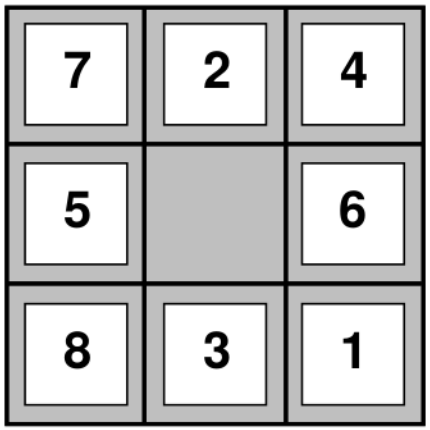
states??: integer dirt and robot locations (ignore dirt amounts etc.)
actions??: $Left$, $Right$, $Suck$, $NoOp$
goal test??: no dirt
path cost??

Example: vacuum world state space graph

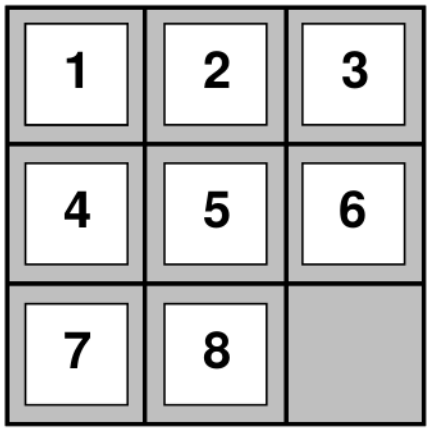


states??: integer dirt and robot locations (ignore dirt amounts etc.)
actions??: $Left$, $Right$, $Suck$, $NoOp$
goal test??: no dirt
path cost??: 1 per action (0 for $NoOp$)

Example: The 8-puzzle



Start State



Goal State

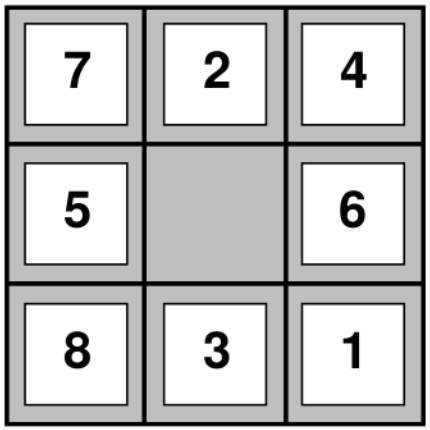
states??

actions??

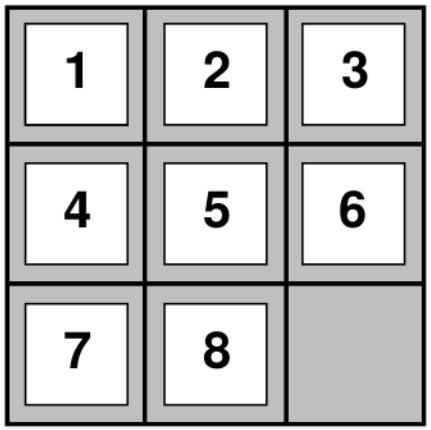
goal test??

path cost??

Example: The 8-puzzle



Start State



Goal State

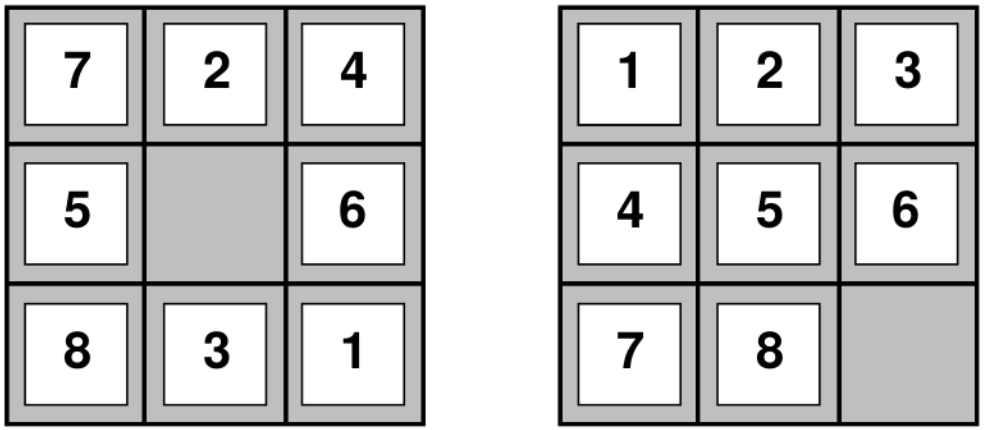
states??: integer locations of tiles (ignore intermediate positions)

actions??

goal test??

path cost??

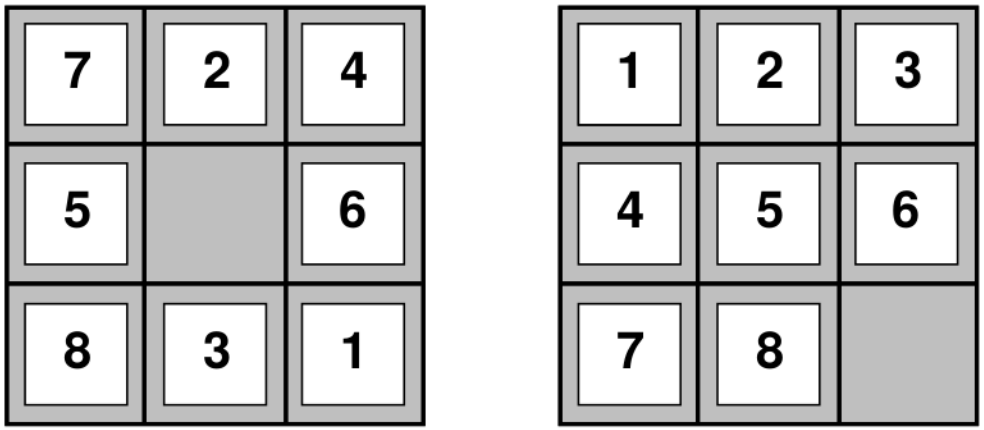
Example: The 8-puzzle



Start State Goal State

states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.)
goal test??
path cost??

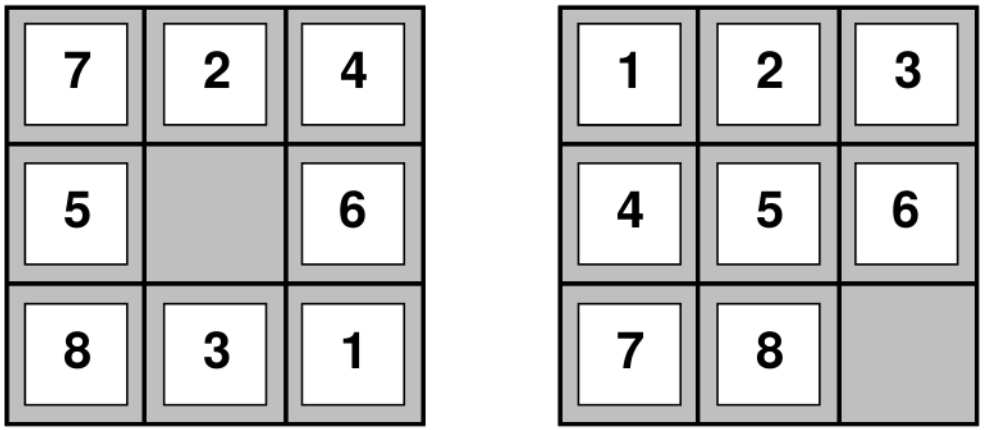
Example: The 8-puzzle



Start State Goal State

states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.)
goal test??: = goal state (given)
path cost??

Example: The 8-puzzle

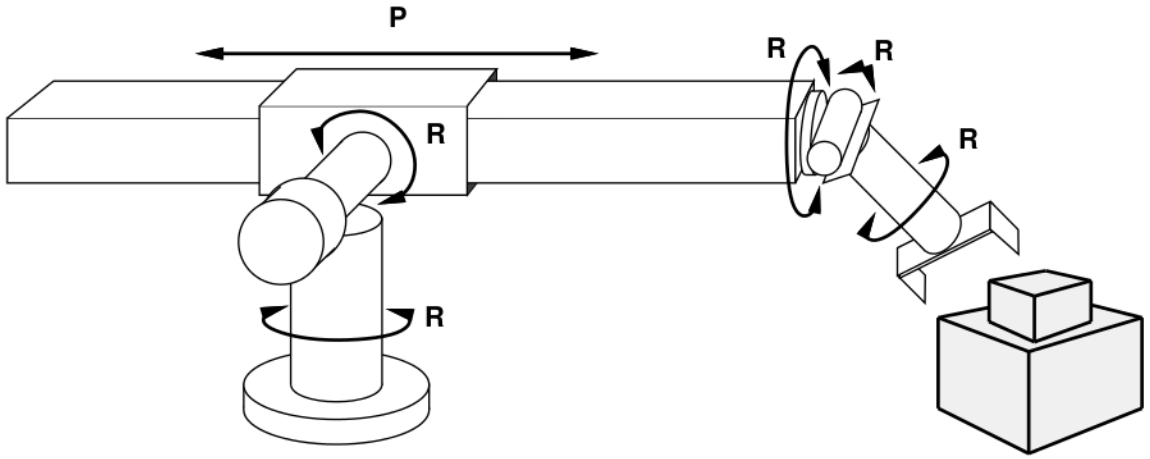


Start State **Goal State**

states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.)
goal test??: $=$ goal state (given)
path cost??: 1 per move

[Note: optimal solution of n-Puzzle family is NP-hard]

Example: robotic assembly



states??: real-valued coordinates of robot joint angles
parts of the object to be assembled

actions??: continuous motions of robot joints

goal test??: complete assembly **with no robot included!**

path cost??: time to execute

Tree search algorithms

Basic idea:

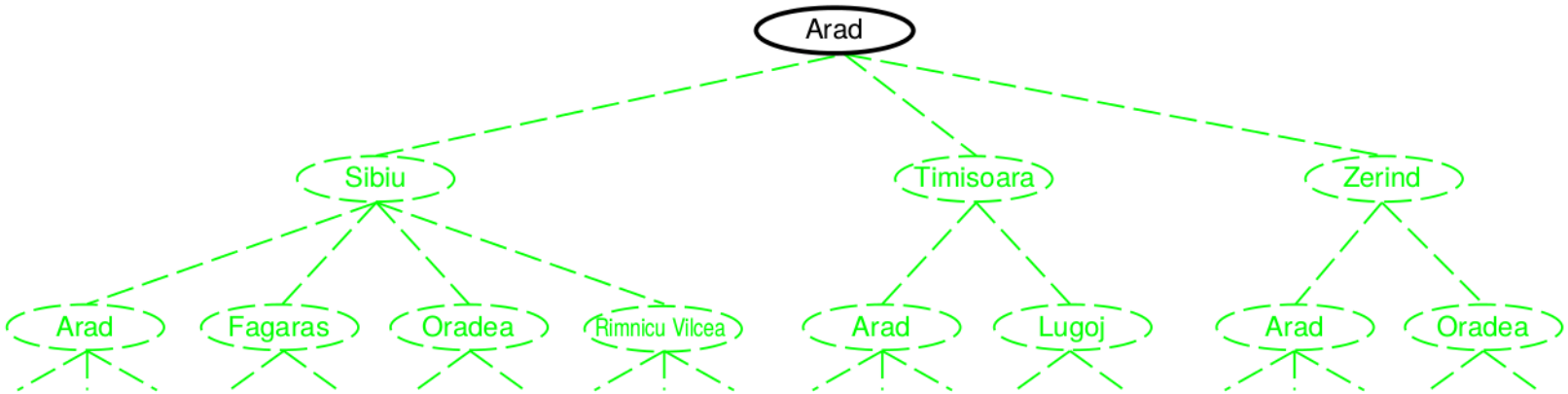
offline, simulated exploration of state space

by generating successors of already-explored states

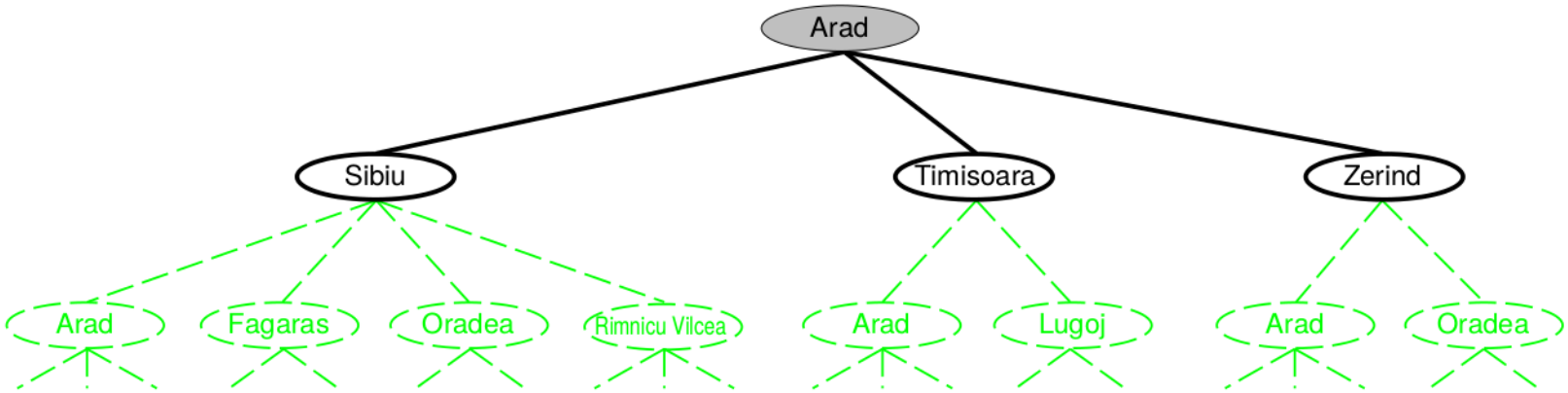
(a.k.a. expanding states)

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end
```

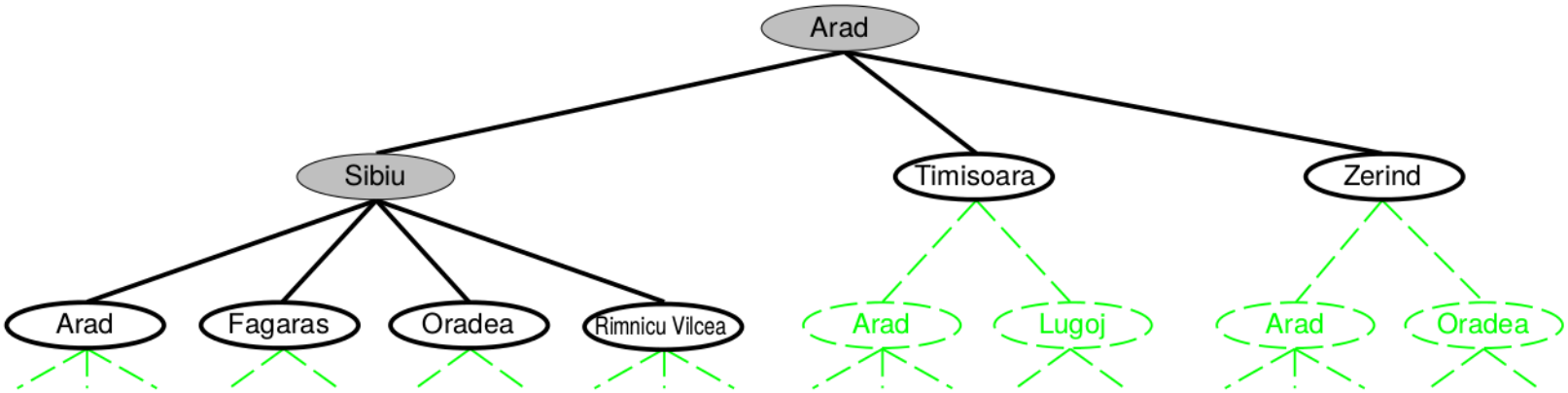
Tree search example



Tree search example



Tree search example



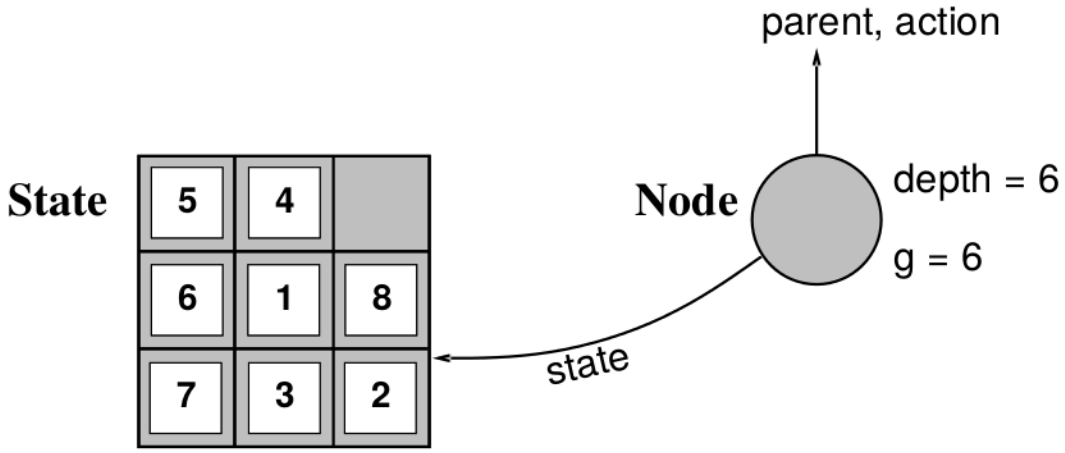
Implementation: states vs. nodes

A state is a (representation of) a physical configuration

A node is a data structure constituting part of a search tree

includes parent, children, depth, path cost $g(x)$

States do not have parents, children, depth, or path cost!



The EXPAND function creates new nodes, filling in the various fields and using the SUCCESSORFN of the problem to create the corresponding states.

Implementation: general tree search

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Search strategies

A strategy is defined by picking the **order of node expansion**

Strategies are evaluated along the following dimensions:

- completeness—does it always find a solution if one exists?
- time complexity—number of nodes generated/expanded
- space complexity—maximum number of nodes in memory
- optimality—does it always find a least-cost solution?

Time and space complexity are measured in terms of

- b—maximum branching factor of the search tree
- d—depth of the least-cost solution
- m—maximum depth of the state space (may be ∞)

Uninformed search strategies

Uninformed strategies use only the information available in the problem definition

Breadth-first search

Uniform-cost search

Depth-first search

Depth-limited search

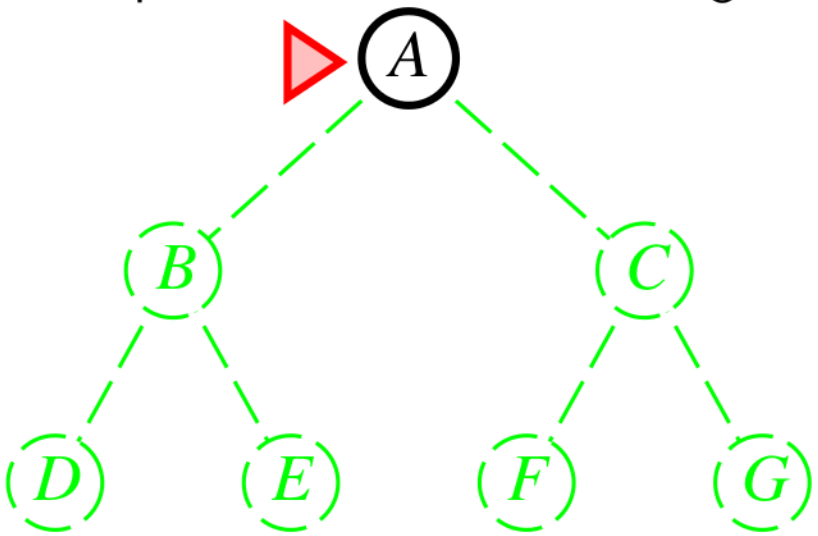
Iterative deepening search

Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

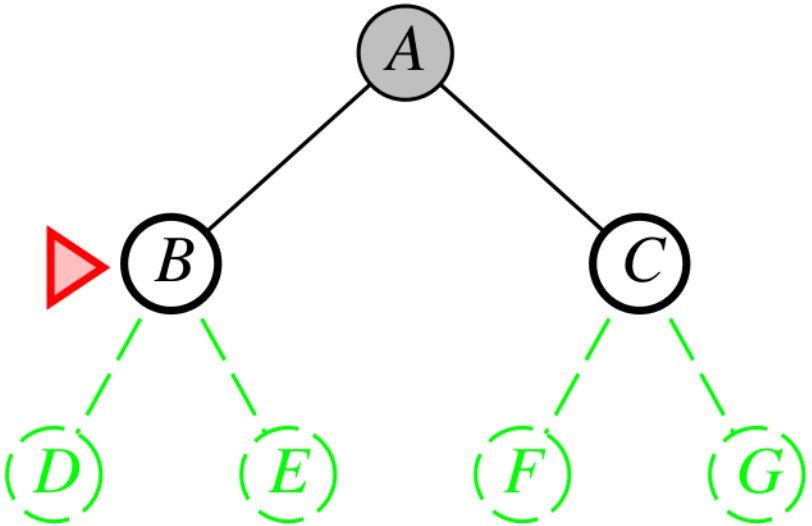


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

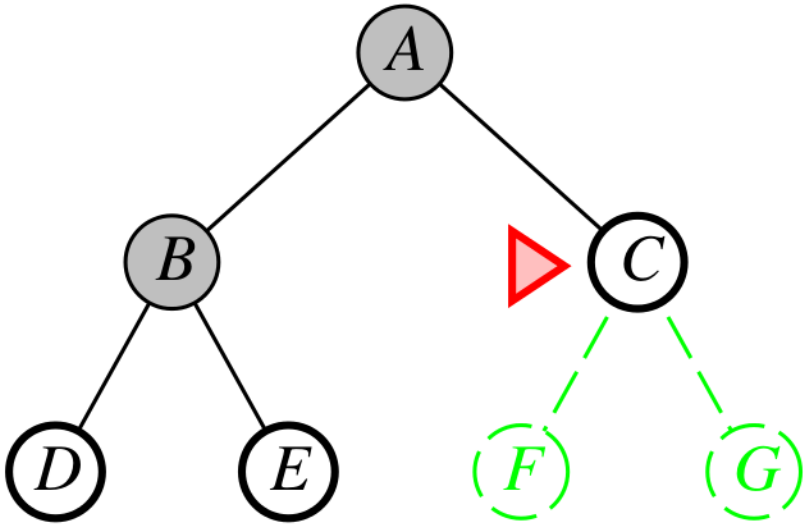


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

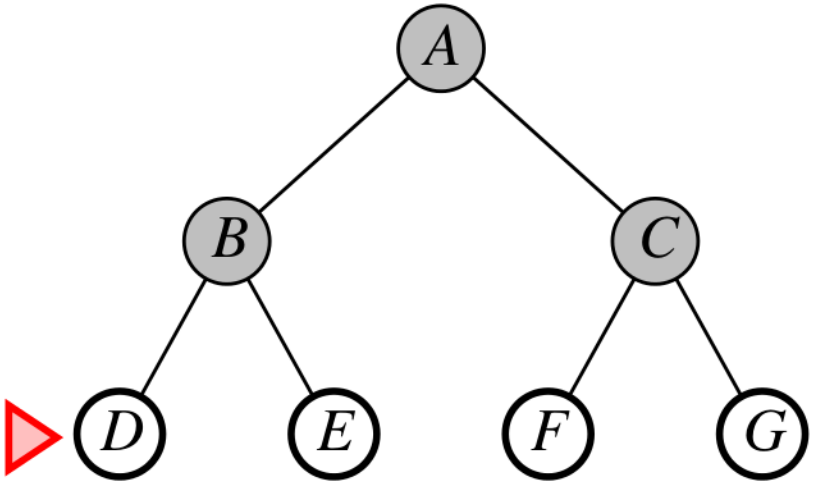


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end



Properties of breadth-first search

Complete??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space?? $O(b^{d+1})$ (keeps every node in memory)

Optimal??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space?? $O(b^{d+1})$ (keeps every node in memory)

Optimal?? Yes (if cost $= 1$ per step); not optimal in general

Space is the big problem; can easily generate nodes at 100MB/sec
so 24hrs $=$ 8640GB.

Uniform-cost search

Expand least-cost unexpanded node

Implementation:

$fringe$ = queue ordered by path cost, lowest first

Equivalent to breadth-first if step costs all equal

Complete?? Yes, if step cost $\geq \epsilon$

Time?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$
where C^* is the cost of the optimal solution

Space?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$

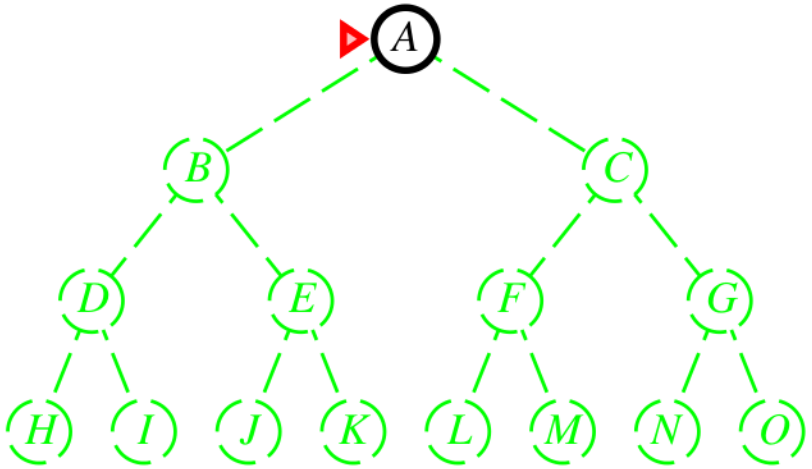
Optimal?? Yes—nodes expanded in increasing order of $g(n)$

Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

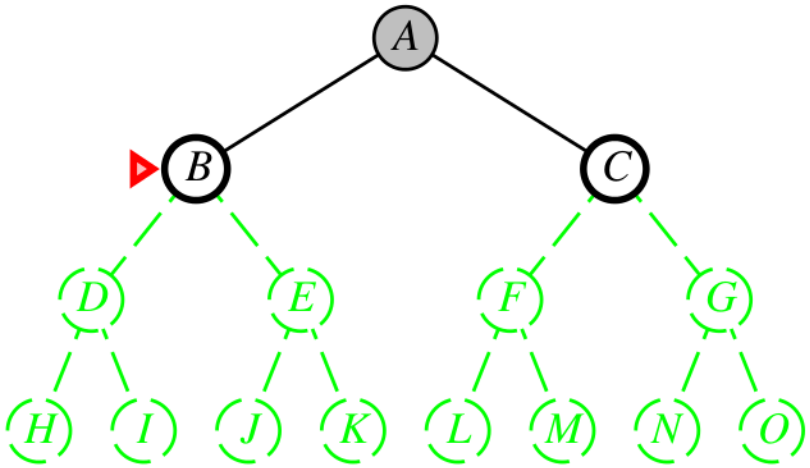


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

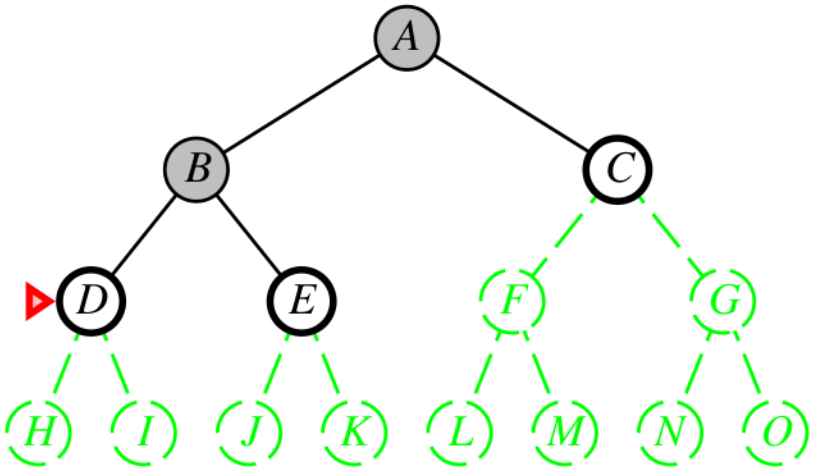


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

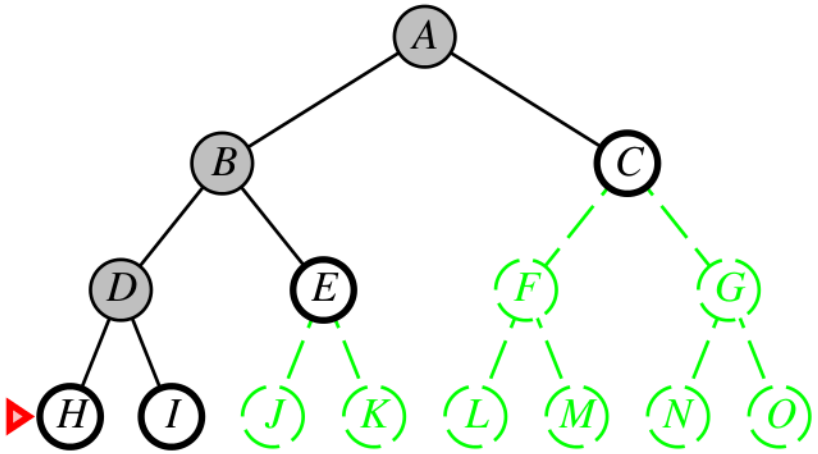


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

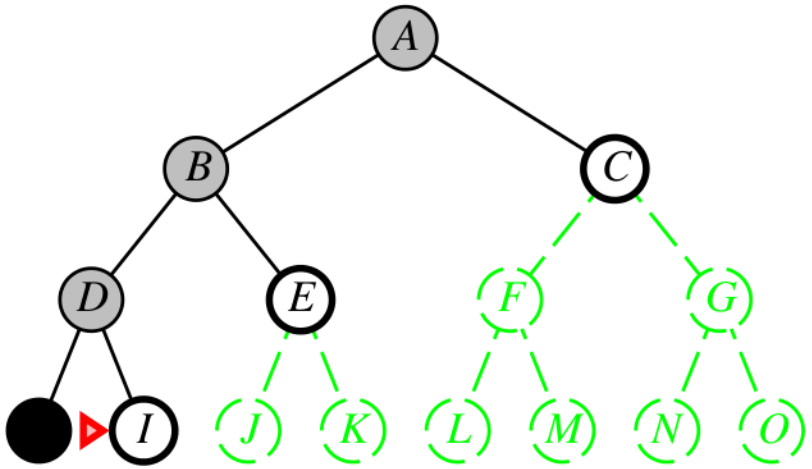


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

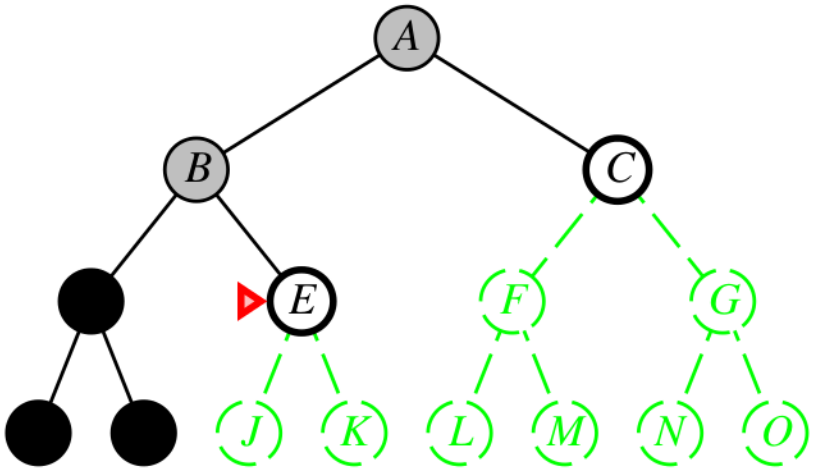


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

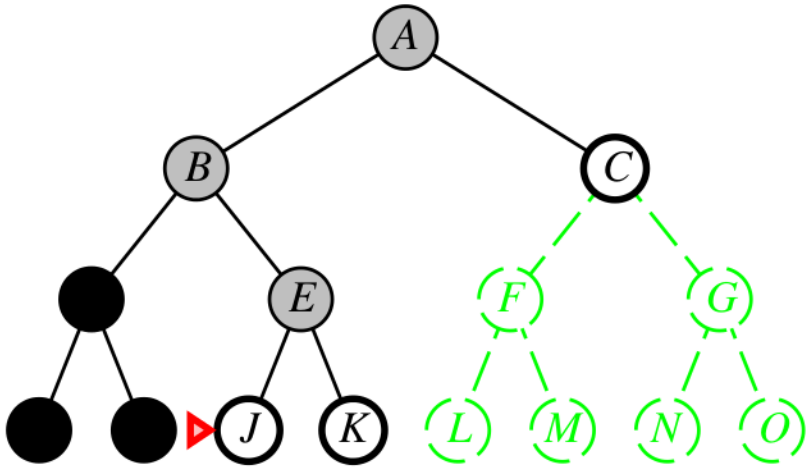


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

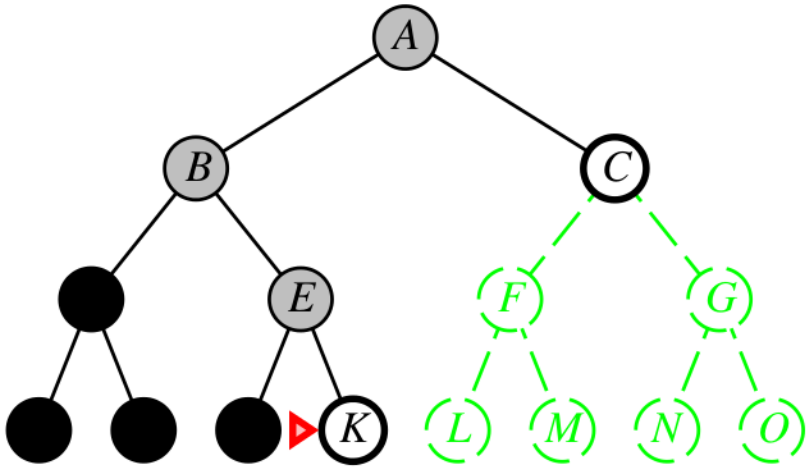


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

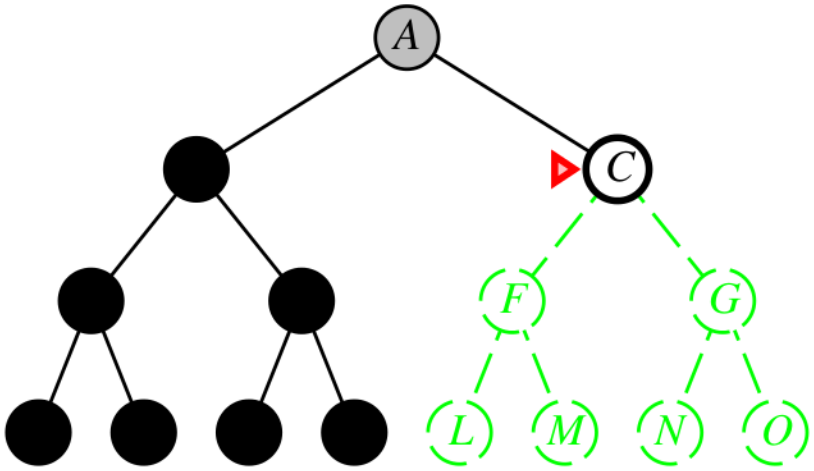


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

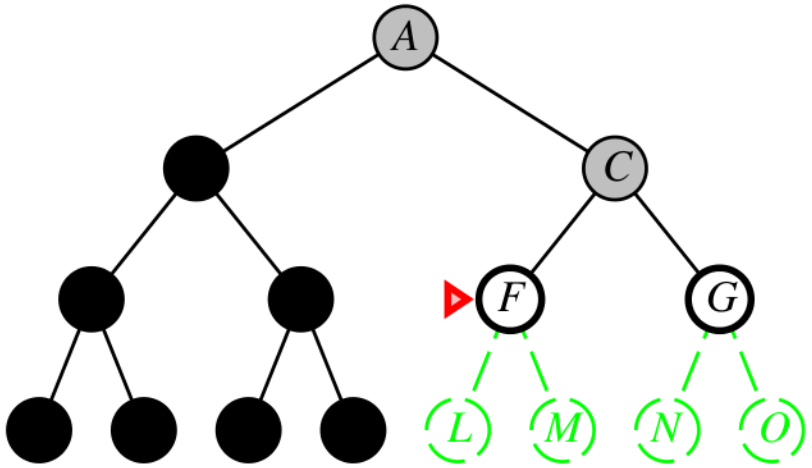


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

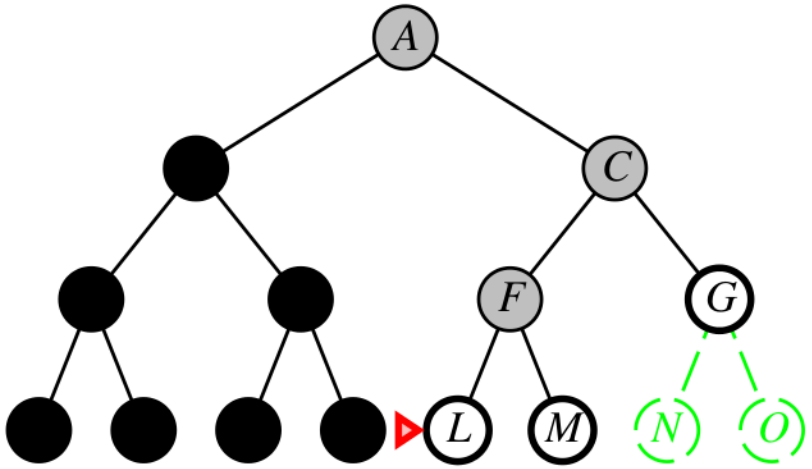


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

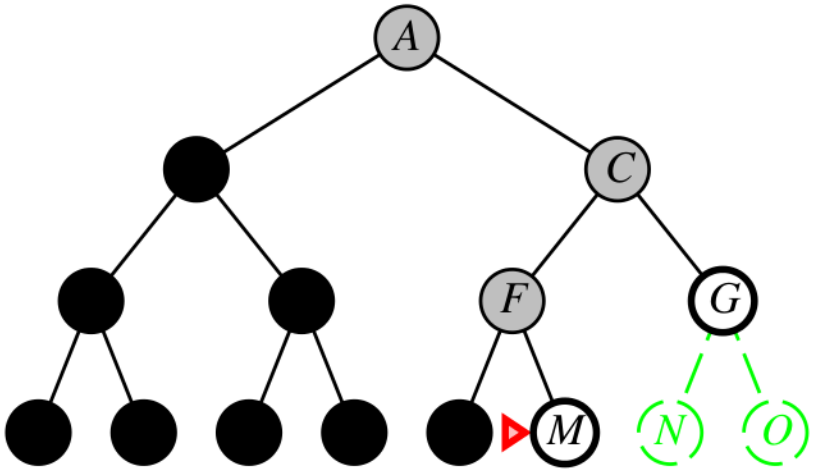


Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front



Properties of depth-first search

Complete??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops

Modify to avoid repeated states along path

$\Rightarrow$ complete in finite spaces

Time??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops

Modify to avoid repeated states along path

$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d

but if solutions are dense, may be much faster than breadth-first

Space??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops

Modify to avoid repeated states along path

$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d

but if solutions are dense, may be much faster than breadth-first

Space?? $O(bm)$, i.e., linear space!

Optimal??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops

Modify to avoid repeated states along path

$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d

but if solutions are dense, may be much faster than breadth-first

Space?? $O(bm)$, i.e., linear space!

Optimal?? No

Depth-limited search

= depth-first search with depth limit l,
i.e., nodes at depth l have no successors

Recursive implementation:

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST(problem, STATE[node]) then return node
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Iterative deepening search

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem

    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

Iterative deepening search $l = 0$

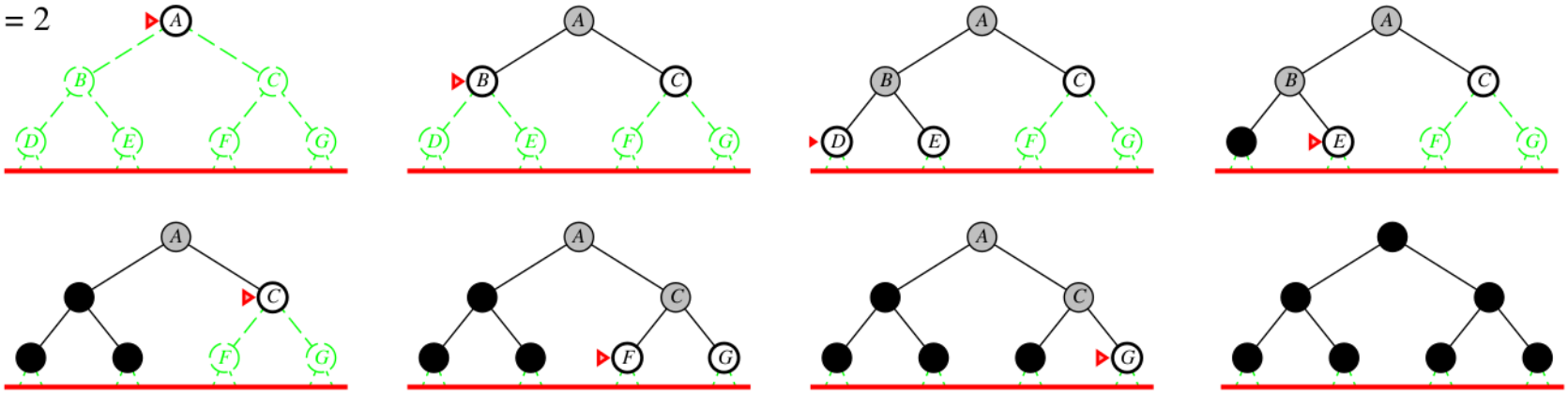
Limit = 0

Iterative deepening search $l = 1$

Limit = 1

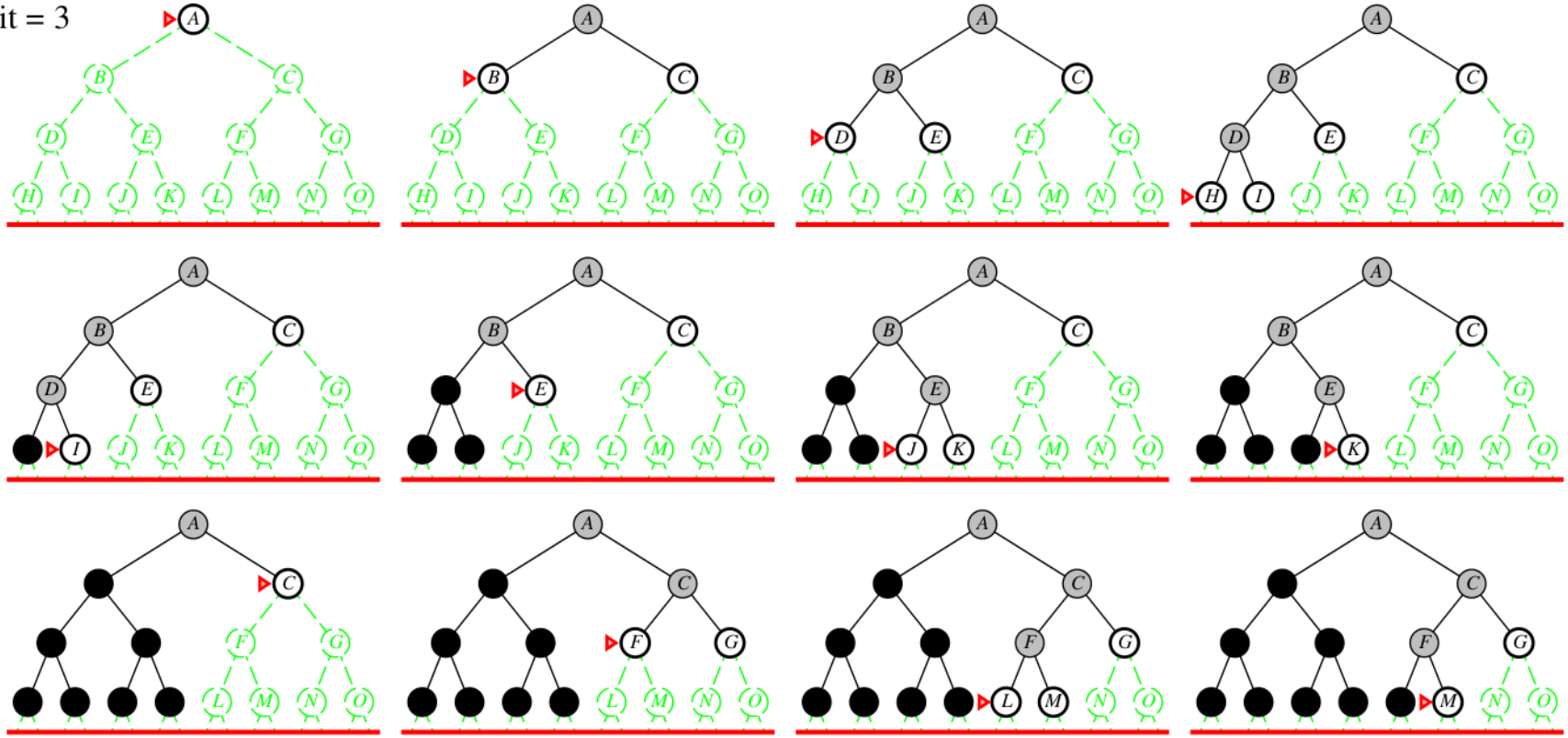
Iterative deepening search $l = 2$

Limit = 2



Iterative deepening search $l = 3$

Limit = 3



Properties of iterative deepening search

Complete??

Properties of iterative deepening search

Complete?? Yes

Time??

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space??

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal??

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? Yes, if step cost $= 1$
Can be modified to explore uniform-cost tree

Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$
$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100$$

IDS does better because other nodes at depth d are not expanded

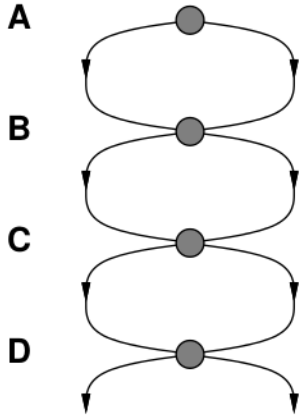
BFS can be modified to apply goal test when a node is **generated**

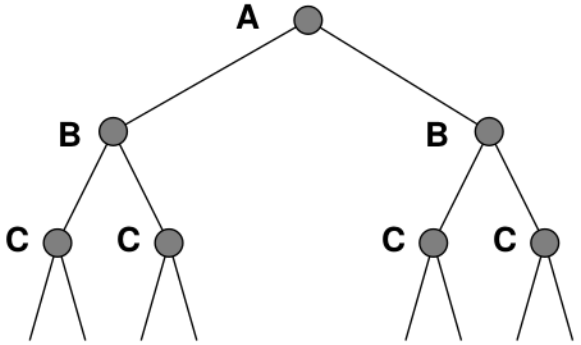
Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

Repeated states

Failure to detect repeated states can turn a linear problem into an exponential one!





Graph search

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure

    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

Summary

Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored

Variety of uninformed search strategies

Iterative deepening search uses only linear space
and not much more time than other uninformed algorithms

Graph search can be exponentially more efficient than tree search