Decision Making under Uncertainty

Matthijs Spaan[§] Frans Oliehoek[*]

[§]Delft University of Technology
[*]Maastricht University
The Netherlands

15th European Agent Systems Summer School (EASSS '13)
London, UK

http://www.st.ewi.tudelft.nl/~mtjspaan/tutorialDMuU/

July 2, 2013

Outline

This lecture:

1. Introduction to decision making under uncertainty
2. Planning under action uncertainty (MDPs)
3. Planning under sensing uncertainty (POMDPs)

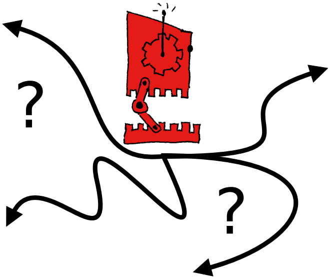
After the break:

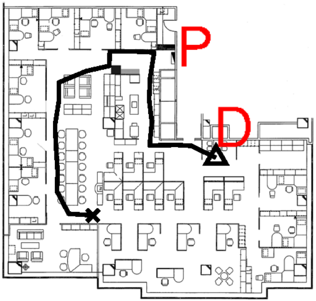
1. Multiagent planning
2. Selected further topics

Introduction

Introduction

- Goal in Artificial Intelligence: to build intelligent agents.
- Our definition of “intelligent”: perform an assigned task as well as possible.
- Problem: how to act?
- We will explicitly model uncertainty.



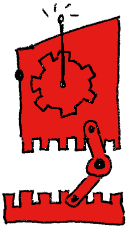


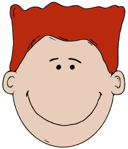
Applications

- Resource planning
- Maintenance
- Queue management
- Medical decision making

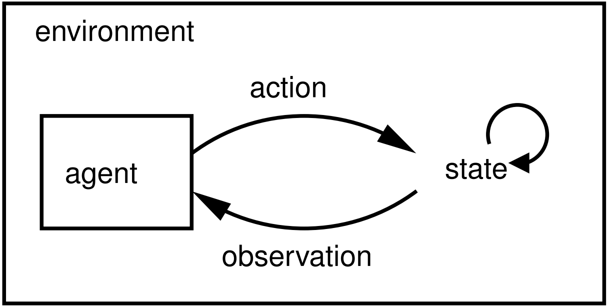
Agents

- An agent is a (rational) decision maker who is able to perceive its external (physical) environment and act autonomously upon it (Russell and Norvig, 2003).
- Rationality means reaching the optimum of a performance measure.
- Examples: humans, robots, some software programs.





Agents



- It is useful to think of agents as being involved in a perception-action loop with their environment.
- But how do we make the right decisions?

Planning

Planning:

- A plan tells an agent how to act.
- For instance
  - A sequence of actions to reach a goal.
  - What to do in a particular situation.
- We need to model:
  - the agent's actions
  - its environment
  - its task

We will model planning as a sequence of decisions.

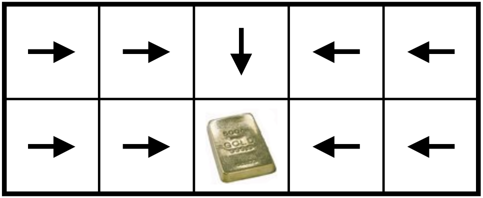
Classic planning

- Classic planning: sequence of actions from start to goal.
- Task: robot should get to gold as quickly as possible.
- Actions: $\rightarrow \downarrow \leftarrow \uparrow$
- Limitations:
  - New plan for each start state.
  - Environment is deterministic.

Classic planning

- Classic planning: sequence of actions from start to goal.
- Task: robot should get to gold as quickly as possible.
- Actions: $\rightarrow \downarrow \leftarrow \uparrow$
- Limitations:
  - New plan for each start state.
  - Environment is deterministic.
- Three optimal plans: $\rightarrow \rightarrow \downarrow$, $\rightarrow \downarrow \rightarrow$, $\downarrow \rightarrow \rightarrow$.

Conditional planning



- Assume our robot has noisy actions (wheel slip, overshoot).
- We need conditional plans.
- Map situations to actions.

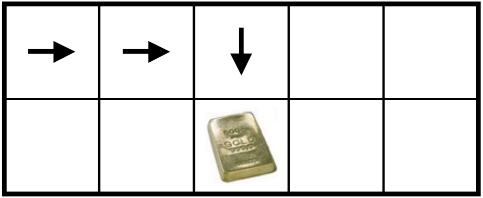
Decision-theoretic planning

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

- Positive reward when reaching goal, small penalty for all other actions.
- Agent's plan maximizes **value**: the sum of future rewards.
- Decision-theoretic planning successfully handles noise in acting and sensing.

Decision-theoretic planning

Plan #1:



Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Values of this plan:

?	?	?		
		10		

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Values of this plan:

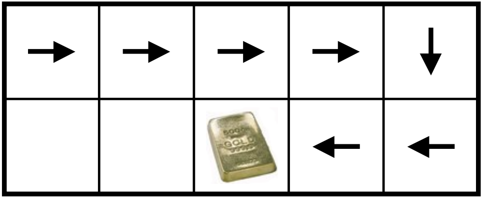
9.7	9.8	9.9		
		10		

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Plan #2:



Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Values of this plan:

?	?	?	?	?
		10	?	?

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Values of this plan:

9.3	9.4	9.5	9.6	9.7
		10	9.9	9.8

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Decision-theoretic planning

Optimal values (encode optimal plan):

9.7	9.8	9.9	9.8	9.7
9.8	9.9	10	9.9	9.8

Reward:

−0.1	−0.1	−0.1	−0.1	−0.1
−0.1	−0.1	10	−0.1	−0.1

Markov Decision Processes

Sequential decision making under uncertainty

- Uncertainty is abundant in **real-world planning** domains.
- **Bayesian** approach $\Rightarrow$ probabilistic models.

Main assumptions:

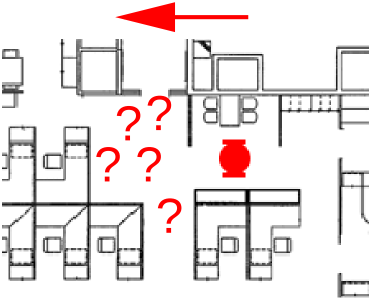
Sequential decisions: problems are formulated as a sequence of "independent" decisions;

Markovian environment: the state at time t depends only on the events at time $t-1$;

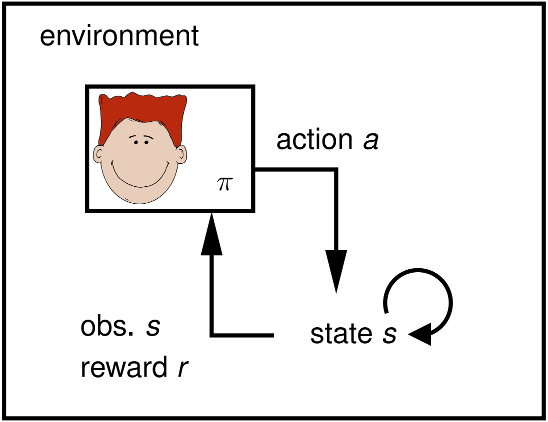
Evaluative feedback: use of a reinforcement signal as performance measure (reinforcement learning);

Transition model

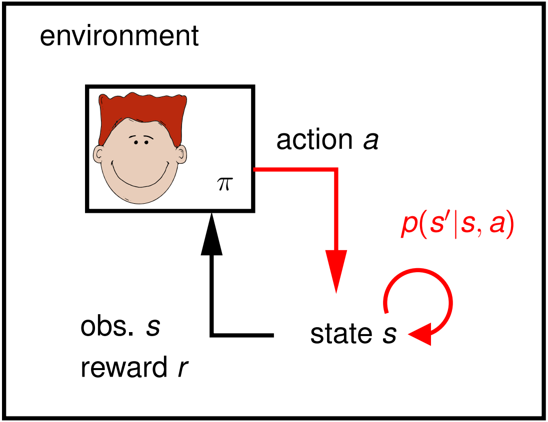
- For instance, robot motion is inaccurate.
- Transitions between states are **stochastic**.
- $p(s'|s,a)$ is the probability to jump from state s to state s' after taking action a.



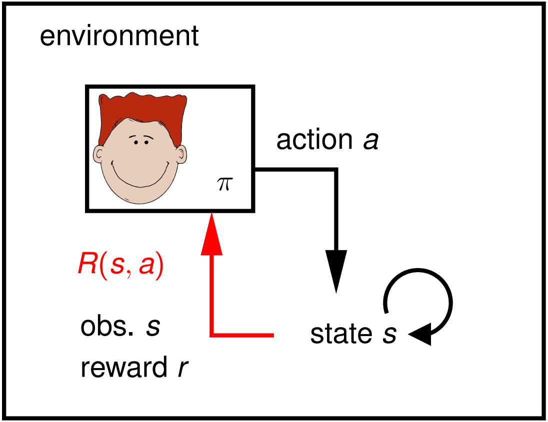
MDP Agent



MDP Agent



MDP Agent



Optimality criterion

For instance, agent should maximize the value

$$E\Big[\sum_{t=0}^{h} \gamma^t R_t\Big], \tag{1}$$

where

- h is the planning horizon, can be finite or ∞
- γ is a discount rate, $0 \leq \gamma < 1$

Reward hypothesis (Sutton and Barto, 1998):
All goals and purposes can be formulated as the maximization of the cumulative sum of a received scalar signal (reward).

Discrete MDP model

Discrete Markov Decision Process model (Puterman, 1994; Bertsekas, 2000):

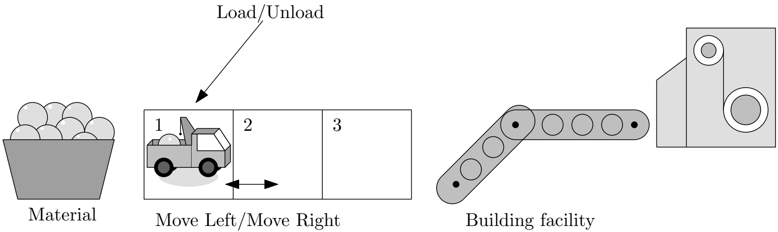
- Time t is discrete.
- State space S.
- Set of actions A.
- Reward function $R : S \times A \mapsto \mathbb{R}$.
- Transition model $p(s'|s, a)$, $T_a : S \times A \mapsto \Delta(S)$.
- Initial state s_0 is drawn from $\Delta(S)$.

The Markov property entails that the next state s_{t+1} only depends on the previous state s_t and action a_t:

$$p(s_{t+1}|s_t, s_{t-1}, \ldots, s_0, a_t, a_{t-1}, \ldots, a_0) = p(s_{t+1}|s_t, a_t). \qquad (2)$$

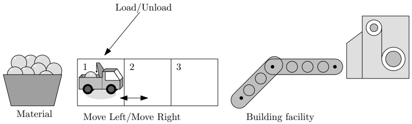
A simple problem

Problem:
An autonomous robot must learn how to transport material from a deposit to a building facility.



(thanks to F. Melo)

Load/Unload as an MDP



- States: $S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$;

  1_U Robot in position 1 (unloaded);
  2_U Robot in position 2 (unloaded);
  3_U Robot in position 3 (unloaded);
  1_L Robot in position 1 (loaded);
  2_L Robot in position 2 (loaded);
  3_L Robot in position 3 (loaded)
- Actions: $A = \{\text{Left, Right, Load, Unload}\}$;

Load/Unload as an MDP (1)

- Transition probabilities: "Left"/"Right" move the robot in the corresponding direction; "Load" loads material (only in position 1); "Unload" unloads material (only in position 3). Ex:

$$\begin{aligned} (2_L, \text{Right}) &\to 3_L; \\ (3_L, \text{Unload}) &\to 3_U; \\ (1_L, \text{Unload}) &\to 1_L. \end{aligned}$$

- Reward: We assign a reward of $+10$ for every unloaded package (payment for the service).

Load/Unload as an MDP (2)

- For each action $a \in A$, T_a is a matrix.
  Ex:

$$T_{\text{Right}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- Recall: $S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$.

Load/Unload as an MDP (3)

- The reward $R(s, a)$ can also be represented as a matrix
  Ex:

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Policies and value

- Policy π: tells the agent how to act.
- A deterministic policy $\pi : S \mapsto A$ is a mapping from states to actions.
- Value: how much reward $E[\sum_{t=0}^{h} \gamma^t R_t]$ does the agent expect to gather.
- Value denoted as $Q^\pi(s, a)$: start in s, do a and follow π afterwards.

Policies and value (1)

- Extracting a policy π from a value function Q is easy:

$$\pi(s) = \arg\max_{a \in A} Q(s, a). \tag{3}$$

- Optimal policy π^*: one that maximizes $E[\sum_{t=0}^{h} \gamma^t R_t]$ (for every state).
- In an infinite-horizon MDP there is always an optimal deterministic stationary (time-independent) policy π^*.
- There can be many optimal policies π^*, but they all share the same optimal value function Q^*.

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} \mathbf{?} & ? & ? & ? \\ \mathbf{?} & ? & ? & ? \\ \mathbf{?} & ? & ? & ? \\ \mathbf{?} & ? & ? & ? \\ \mathbf{?} & ? & ? & ? \\ \mathbf{?} & ? & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s,a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \\ 0 & ? & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & \mathbf{?} & ? & ? \\ 0 & \mathbf{?} & ? & ? \\ 0 & \mathbf{?} & ? & ? \\ 0 & \mathbf{?} & ? & ? \\ 0 & \mathbf{?} & ? & ? \\ 0 & \mathbf{?} & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 0 & \mathbf{?} & ? \\ 0 & 0 & \mathbf{?} & ? \\ 0 & 0 & \mathbf{?} & ? \\ 0 & 0 & \mathbf{?} & ? \\ 0 & 0 & \mathbf{?} & ? \\ 0 & 0 & \mathbf{?} & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \\ 0 & 0 & 0 & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 0 & 0 & \textbf{?} \\ 0 & 0 & 0 & \textbf{?} \\ 0 & 0 & 0 & \textbf{?} \\ 0 & 0 & 0 & \textbf{?} \\ 0 & 0 & 0 & \textbf{?} \\ 0 & 0 & 0 & \textbf{?} \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \qquad Q_2 = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s,a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \qquad Q_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & ? & 0 & 0 \\ 0 & ? & ? & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Since S and A are finite, $Q^*(s, a)$ is a matrix.
Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix} \qquad Q_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 9.5 & 0 & 0 \\ 0 & 9.5 & 9.5 & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Iterations of dynamic programming ($\gamma = 0.95$):

$$Q_5 = \begin{bmatrix} 0 & 0 & 8.57 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 8.57 & 9.03 & 8.57 & 8.57 \\ 8.57 & 9.5 & 9.03 & 9.03 \\ 9.03 & 9.5 & 9.5 & 10 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Iterations of DP:

$$Q_{20} = \begin{bmatrix} 18.53 & 17.61 & 19.51 & 18.54 \\ 18.53 & 16.73 & 17.61 & 17.61 \\ 17.61 & 16.73 & 16.73 & 16.73 \\ 19.51 & 20.54 & 19.51 & 19.51 \\ 19.51 & 21.62 & 20.54 & 20.54 \\ 20.54 & 21.62 & 21.62 & 26.73 \end{bmatrix}$$

$S = \{1_U, 2_U, 3_U, 1_L, 2_L, 3_L\}$, $A = \{\text{Left, Right, Load, Unload}\}$

Dynamic programming

Final Q^* and policy:

$$Q^* = \begin{bmatrix} 30.75 & 29.21 & 32.37 & 30.75 \\ 30.75 & 27.75 & 29.21 & 29.21 \\ 29.21 & 27.75 & 27.75 & 27.75 \\ 32.37 & 34.07 & 32.37 & 32.37 \\ 32.37 & 35.86 & 34.07 & 34.07 \\ 34.07 & 35.86 & 35.86 & 37.75 \end{bmatrix} \qquad \pi^* = \begin{bmatrix} \text{Load} \\ \text{Left} \\ \text{Left} \\ \text{Right} \\ \text{Right} \\ \text{Unload} \end{bmatrix}$$

Value iteration

- Value iteration: successive approximation technique.
- Start with all values set to 0.
- In order to consider one step deeper into the future, i.e., to compute V_{n+1}^* from V_n^*:

$$Q_{n+1}^*(s,a) := R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q_n^*(s',a'), \quad (4)$$

  which is known as the dynamic programming update or Bellman backup.
- Bellman (1957) equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} Q^*(s',a'). \quad (5)$$

Value iteration (1)

Initialize Q arbitrarily, e.g., $Q(s, a) = 0, \forall s \in S, a \in A$
repeat
  $\delta \leftarrow 0$
  for all $s \in S, a \in A$ **do**
    $v \leftarrow Q(s, a)$
    $Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \max_{a' \in A} Q(s', a')$
    $\delta \leftarrow \max(\delta, |v - Q(s, a)|)$
  end for
until $\delta < \epsilon$
Return Q

Value iteration (2)

Value iteration discussion:

- As $n \to \infty$, value iteration converges.
- Value iteration has converged when the largest update δ in an iteration is below a certain threshold ϵ.
- Exhaustive sweeps are not required for convergence, provided that in the limit all states are visited infinitely often.
- This can be exploited by backing up the most promising states first, known as prioritized sweeping.

Solution methods: MDPs

Model based

- Basic: dynamic programming (Bellman, 1957), value iteration, policy iteration.
- Advanced: prioritized sweeping, function approximators.

Model free, reinforcement learning (Sutton and Barto, 1998)

- Basic: Q-learning, TD(λ), SARSA, actor-critic.
- Advanced: generalization in infinite state spaces, exploration/exploitation issues.

POMDPs

Beyond MDPs

- Real agents cannot directly observe the state.
- Sensors provide partial and noisy information about the world.

Beyond MDPs

- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:

- Diagnosis (medical, maintenance)
- Robot navigation
- Tutoring
- Dialog systems
- Vision-based robotics
- Fault recovery

Beyond MDPs

- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:

- Diagnosis (medical, maintenance)
- Robot navigation
- Tutoring
- Dialog systems
- Vision-based robotics
- Fault recovery

Beyond MDPs

- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:

- Diagnosis (medical, maintenance)
- Robot navigation
- Tutoring
- Dialog systems
- Vision-based robotics
- Fault recovery

Beyond MDPs

- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:

- Diagnosis (medical, maintenance)
- Robot navigation
- Tutoring
- Dialog systems
- Vision-based robotics
- Fault recovery

Beyond MDPs

- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:

- Diagnosis (medical, maintenance)
- Robot navigation
- Tutoring
- Dialog systems
- Vision-based robotics
- Fault recovery

Beyond MDPs

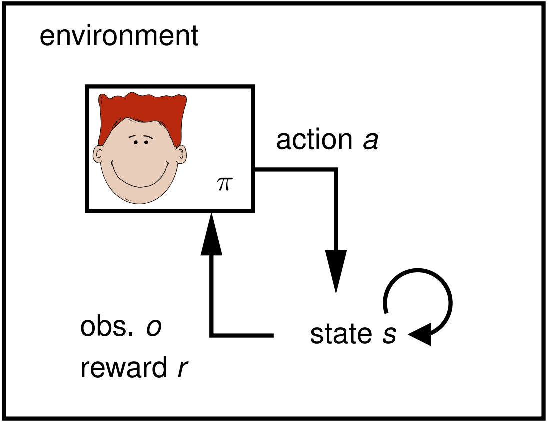
- MDPs have been very successful, but requires to have an observable Markovian state.
- Many domains this is impossible (or expensive) to obtain:

- Diagnosis (medical, maintenance)
- Robot navigation
- Tutoring
- Dialog systems
- Vision-based robotics
- Fault recovery

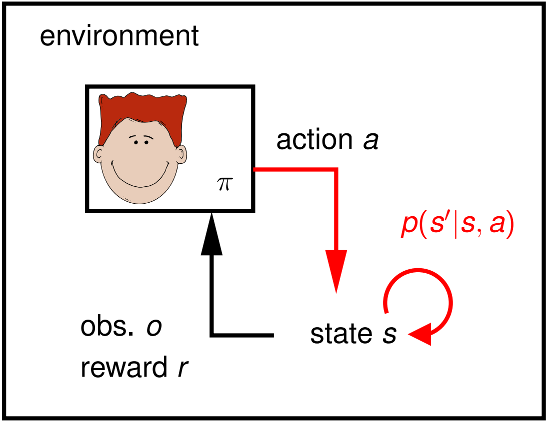
Observation model

- Imperfect sensors.
- Partially observable environment:
  - Sensors are **noisy**.
  - Sensors have a **limited view**.
- $p(o|s', a)$ is the probability the agent receives observation o in state s' after taking action a.

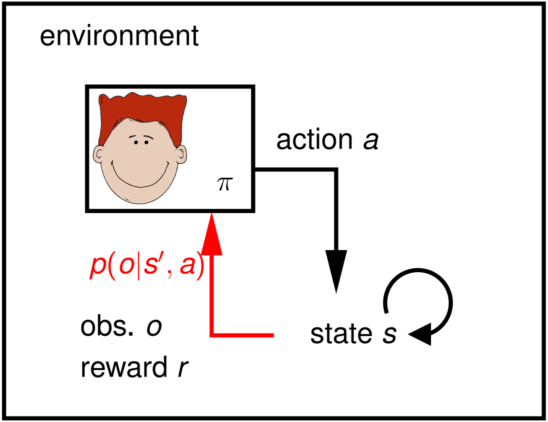
POMDP Agent



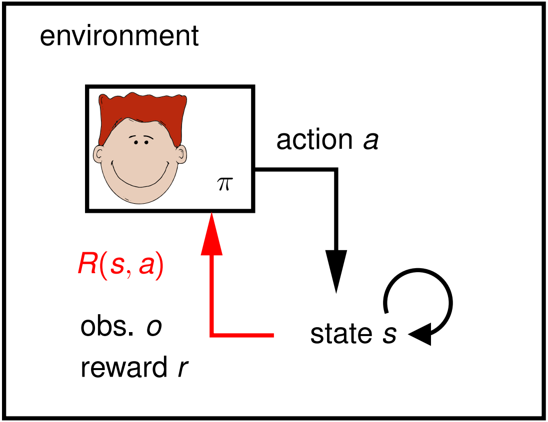
POMDP Agent



POMDP Agent



POMDP Agent



POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s, a)$: models the effect of **actions**.
- Observation model $p(o|s', a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s, a)$.
- A planning horizon h (finite or ∞).
- A discount rate $0 \leq \gamma < 1$.
- Goal is to compute plan, or **policy** π, that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s,a)$: models the effect of **actions**.
- Observation model $p(o|s',a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s,a)$.
- A planning horizon h (finite or ∞).
- A discount rate $0 \leq \gamma < 1$.
- Goal is to compute plan, or **policy** π, that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s, a)$: models the effect of **actions**.
- Observation model $p(o|s', a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s, a)$.
- A planning horizon h (finite or ∞).
- A discount rate $0 \leq \gamma < 1$.
- Goal is to compute plan, or **policy** π, that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s, a)$: models the effect of **actions**.
- Observation model $p(o|s', a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s, a)$.
- A planning horizon h (finite or ∞).
- A discount rate $0 \leq \gamma < 1$.
- Goal is to compute plan, or **policy** π, that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s,a)$: models the effect of **actions**.
- Observation model $p(o|s',a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s,a)$.
- A planning horizon h (finite or ∞).
- A discount rate $0 \leq \gamma < 1$.
- Goal is to compute plan, or **policy** π, that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s,a)$: models the effect of **actions**.
- Observation model $p(o|s',a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s,a)$.
- A planning horizon h (finite or ∞).
- A discount rate $0 \leq \gamma < 1$.
- Goal is to compute plan, or **policy** π, that maximizes long-term reward.

POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s,a)$: models the effect of **actions**.
- Observation model $p(o|s',a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s,a)$.
- A planning horizon h (finite or ∞).
- A discount rate $0 \leq \gamma < 1$.
- Goal is to compute plan, or **policy** π, that maximizes long-term reward.

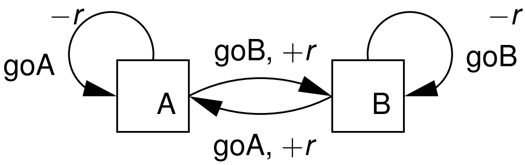
POMDPs

Partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998):

- Framework for agent planning under uncertainty.
- Typically assumes discrete sets of states S, actions A and observations O.
- Transition model $p(s'|s, a)$: models the effect of **actions**.
- Observation model $p(o|s', a)$: relates **observations** to states.
- Task is defined by a **reward** model $R(s, a)$.
- A planning horizon h (finite or ∞).
- A discount rate $0 \leq \gamma < 1$.
- Goal is to compute plan, or **policy** π, that maximizes long-term reward.

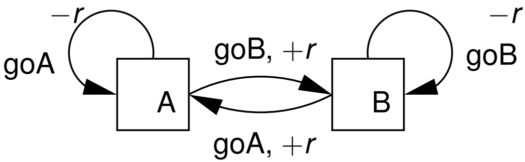
Memory

- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Memory

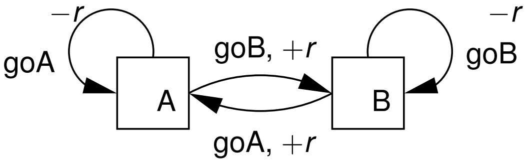
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Policy	Value
MDP: optimal policy	
POMDP: memoryless deterministic	
POMDP: memoryless stochastic	
POMDP: memory-based (optimal)	

Memory

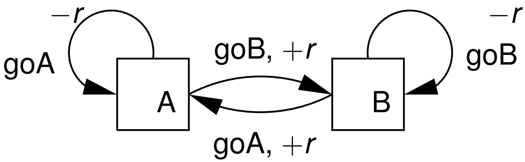
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Policy	Value
MDP: optimal policy	$V = \sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma}$
POMDP: memoryless deterministic	
POMDP: memoryless stochastic	
POMDP: memory-based (optimal)	

Memory

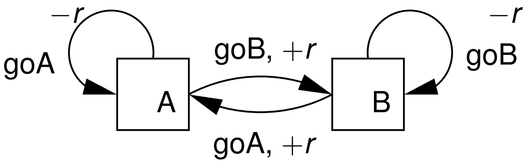
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Policy	Value
MDP: optimal policy	$V = \sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma}$
POMDP: memoryless deterministic	$V_{\max} = r - \frac{\gamma r}{1-\gamma}$
POMDP: memoryless stochastic	
POMDP: memory-based (optimal)	

Memory

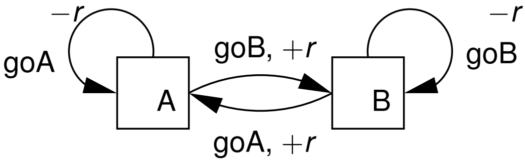
- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Policy	Value
MDP: optimal policy	$V = \sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma}$
POMDP: memoryless deterministic	$V_{\max} = r - \frac{\gamma r}{1-\gamma}$
POMDP: memoryless stochastic	$V = 0$
POMDP: memory-based (optimal)	

Memory

- In POMDPs memory is required for optimal decision making.
- In this non-observable example (Singh et al., 1994):



Policy	Value
MDP: optimal policy	$V = \sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma}$
POMDP: memoryless deterministic	$V_{\text{max}} = r - \frac{\gamma r}{1-\gamma}$
POMDP: memoryless stochastic	$V = 0$
POMDP: memory-based (optimal)	$V_{\text{min}} = \frac{\gamma r}{1-\gamma} - r$

Beliefs

Beliefs:

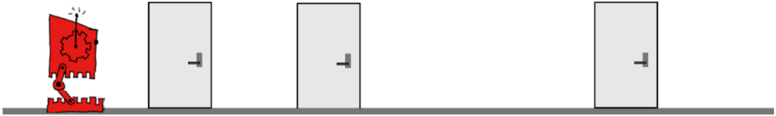
- The agent maintains a **belief** $b(s)$ of being at state s.
- After action $a \in A$ and observation $o \in O$ the belief $b(s)$ can be updated using Bayes' rule:

$$b'(s') \propto p(o|s') \sum_{s} p(s'|s,a) b(s)$$

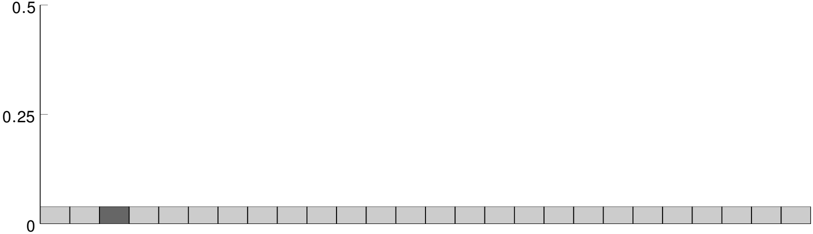
- The belief vector is a **Markov** signal for the planning task.

Belief update example

True situation:



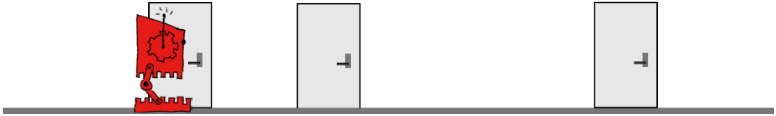
Robot's belief:



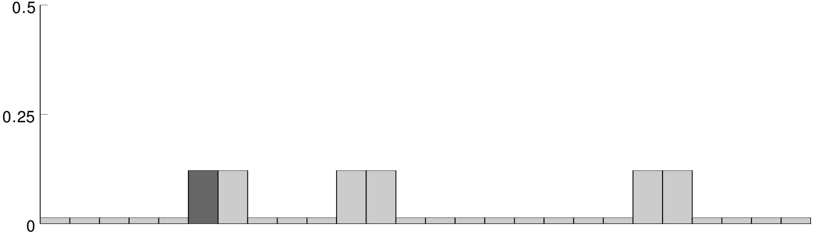
- Observations: *door* or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



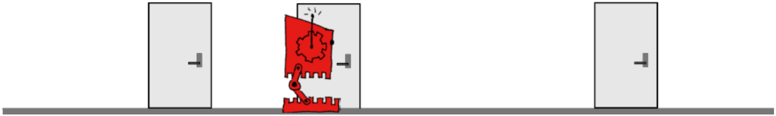
Robot's belief:



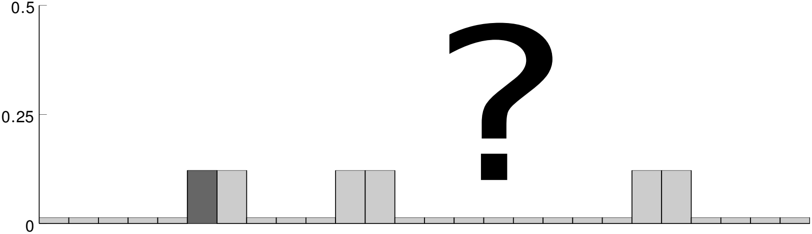
- Observations: **door** or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



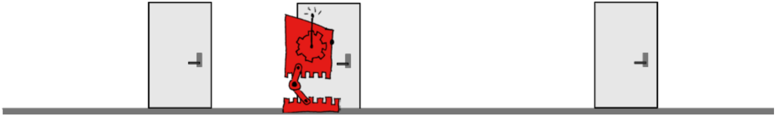
Robot's belief:



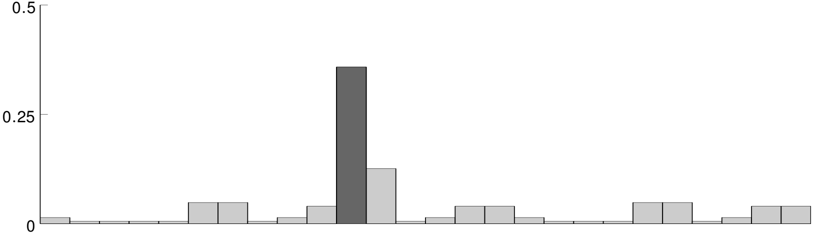
- Observations: **door** or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



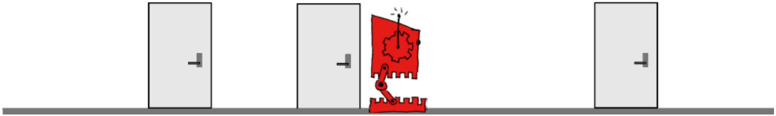
Robot's belief:



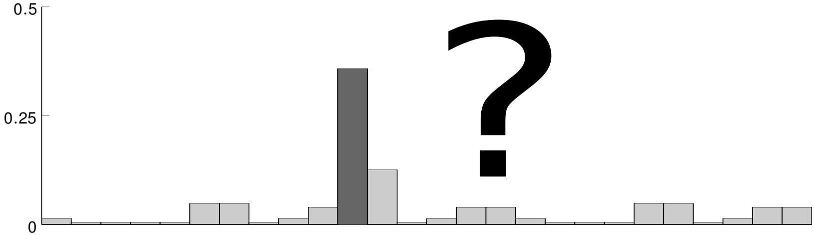
- Observations: **door** or *corridor*, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



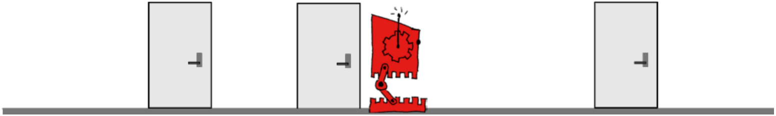
Robot's belief:



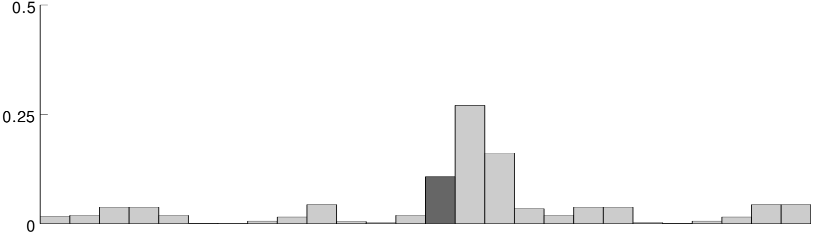
- Observations: *door* or **corridor**, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

Belief update example

True situation:



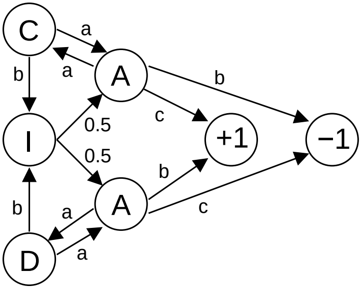
Robot's belief:



- Observations: *door* or **corridor**, 10% noise.
- Action: moves 3 (20%), 4 (60%), or 5 (20%) states.

MDP-based algorithms

- Exploit belief state, and use the MDP solution as a heuristic.
- Most likely state (Cassandra et al., 1996): $\pi_{MLS}(b) = \pi^*(\arg\max_s b(s))$.
- Q_{MDP} (Littman et al., 1995): $\pi_{Q_{\text{MDP}}}(b) = \arg\max_a \sum_s b(s) Q^*(s, a)$.



(Parr and Russell, 1995)

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- Continuous state space Δ: a simplex in $[0,1]^{|S|-1}$.
- Stochastic Markovian transition model $p(b_a^o|b,a) = p(o|b,a)$. This is the normalizer of Bayes' rule.
- Reward function $R(b,a) = \sum_s R(s,a)b(s)$. This is the average reward with respect to $b(s)$.
- The robot fully 'observes' the new belief-state b_a^o after executing a and observing o.

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- Continuous state space Δ: a simplex in $[0, 1]^{|S|-1}$.
- Stochastic Markovian transition model $p(b_a^o|b, a) = p(o|b, a)$. This is the normalizer of Bayes' rule.
- Reward function $R(b, a) = \sum_s R(s, a)b(s)$. This is the average reward with respect to $b(s)$.
- The robot fully 'observes' the new belief-state b_a^o after executing a and observing o.

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- Continuous state space Δ: a simplex in $[0, 1]^{|S|-1}$.
- Stochastic Markovian transition model $p(b_a^o|b, a) = p(o|b, a)$. This is the normalizer of Bayes' rule.
- Reward function $R(b, a) = \sum_s R(s, a)b(s)$. This is the average reward with respect to $b(s)$.
- The robot fully 'observes' the new belief-state b_a^o after executing a and observing o.

POMDPs as continuous-state MDPs

A belief-state POMDP can be treated as a continuous-state MDP:

- Continuous state space Δ: a simplex in $[0, 1]^{|S|-1}$.
- Stochastic Markovian transition model $p(b_a^o|b, a) = p(o|b, a)$. This is the normalizer of Bayes' rule.
- Reward function $R(b, a) = \sum_s R(s, a)b(s)$. This is the average reward with respect to $b(s)$.
- The robot fully 'observes' the new belief-state b_a^o after executing a and observing o.

Solving POMDPs

- A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.
- The optimal value V^* of a POMDP satisfies the Bellman optimality equation $V^* = HV^*$:

$$V^*(b) = \max_a \left[R(b,a) + \gamma \sum_o p(o|b,a) V^*(b_a^o) \right]$$

- Value iteration repeatedly applies $V_{n+1} = HV_n$ starting from an initial V_0.
- Computing the optimal value function is a hard problem (PSPACE-complete for finite horizon, undecidable for infinite horizon).

Solving POMDPs

- A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.
- The optimal value V^* of a POMDP satisfies the Bellman optimality equation $V^* = HV^*$:

$$V^*(b) = \max_a \left[R(b,a) + \gamma \sum_o p(o|b,a) V^*(b_a^o) \right]$$

- Value iteration repeatedly applies $V_{n+1} = HV_n$ starting from an initial V_0.
- Computing the optimal value function is a hard problem (PSPACE-complete for finite horizon, undecidable for infinite horizon).

Solving POMDPs

- A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.
- The optimal value V^* of a POMDP satisfies the Bellman optimality equation $V^* = HV^*$:

$$V^*(b) = \max_a \left[R(b, a) + \gamma \sum_o p(o|b, a) V^*(b_a^o) \right]$$

- Value iteration repeatedly applies $V_{n+1} = HV_n$ starting from an initial V_0.
- Computing the optimal value function is a hard problem (PSPACE-complete for finite horizon, undecidable for infinite horizon).

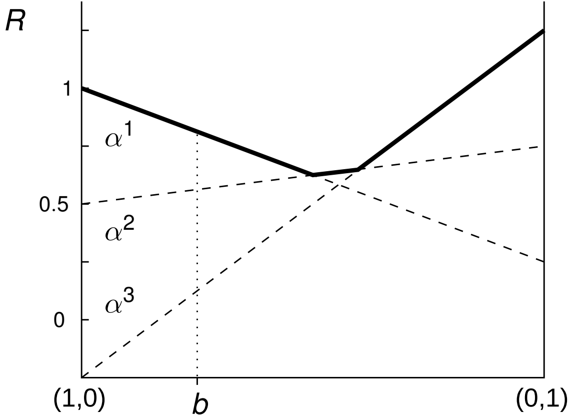
Solving POMDPs

- A solution to a POMDP is a **policy**, i.e., a mapping $\pi : \Delta \mapsto A$ from beliefs to actions.
- The optimal value V^* of a POMDP satisfies the Bellman optimality equation $V^* = HV^*$:

$$V^*(b) = \max_a \Big[R(b,a) + \gamma \sum_o p(o|b,a) V^*(b_a^o) \Big]$$

- Value iteration repeatedly applies $V_{n+1} = HV_n$ starting from an initial V_0.
- Computing the optimal value function is a hard problem (PSPACE-complete for finite horizon, undecidable for infinite horizon).

Example V_0



$R(s,a)$	a_1	a_2	a_3
s_1	1.00	0.50	-0.25
s_2	0.25	0.75	1.25

PWLC shape of V_n

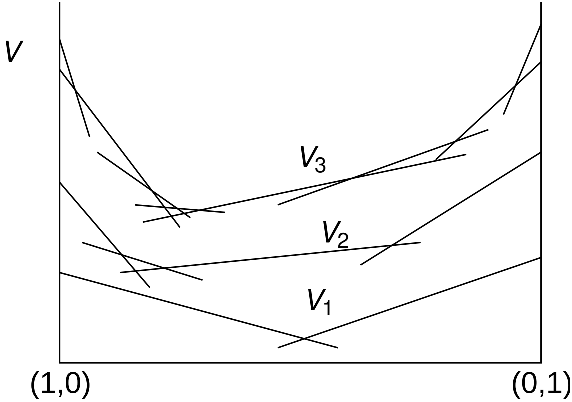
- Like V_0, V_n is as well piecewise linear and convex.
- Rewards $R(b, a) = b \cdot R(s, a)$ are linear functions of b. Note that the value of a point b satisfies:

$$V_{n+1}(b) = \max_a \left[b \cdot R(s, a) + \gamma \sum_o p(o|b, a) V_n(b_a^o) \right]$$

  which involves a maximization over (at least) the vectors $R(s, a)$.
- Intuitively: less uncertainty about the state (low-entropy beliefs) means better decisions (thus higher value).

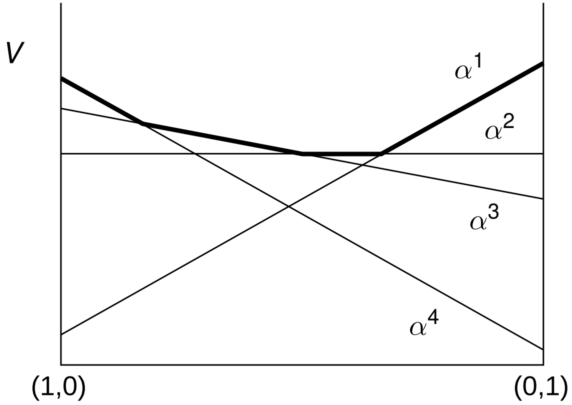
Exact value iteration

Value iteration computes a sequence of value function estimates $V_1, V_2, \ldots, V_n$, using the POMDP backup operator H, $V_{n+1} = HV_n$.

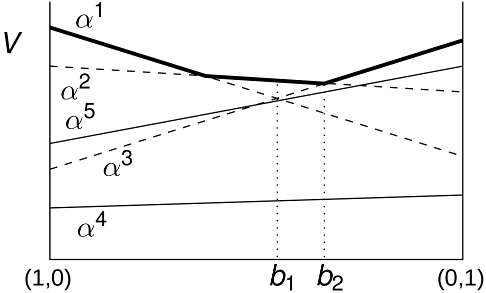


Optimal value functions

The optimal value function of a (finite-horizon) POMDP is piecewise linear and convex: $V(b) = \max_{\alpha} b \cdot \alpha$.



Vector pruning



Linear program for pruning:

variables: $\forall s \in S, b(s); x$
maximize: x
subject to:

$$b \cdot (\alpha - \alpha') \geq x, \forall \alpha' \in V, \alpha' \neq \alpha$$
$$b \in \Delta(S)$$

Optimal POMDP methods

Enumerate and prune:

- Most straightforward: Monahan (1982)'s enumeration algorithm. Generates a maximum of $|A||V_n|^{|O|}$ vectors at each iteration, hence requires pruning.
- Incremental pruning (Zhang and Liu, 1996; Cassandra et al., 1997).

Search for witness points:

- One Pass (Sondik, 1971; Smallwood and Sondik, 1973).
- Relaxed Region, Linear Support (Cheng, 1988).
- Witness (Cassandra et al., 1994).

Sub-optimal techniques

- Grid-based approximations

  (Drake, 1962; Lovejoy, 1991; Brafman, 1997; Zhou and Hansen, 2001; Bonet, 2002).
- Optimizing finite-state controllers

  (Platzman, 1981; Hansen, 1998b; Poupart and Boutilier, 2004).
- Heuristic search in the belief tree

  (Satia and Lave, 1973; Hansen, 1998a).
- Compression or clustering

  (Roy et al., 2005; Poupart and Boutilier, 2003; Virin et al., 2007).
- Point-based techniques

  (Pineau et al., 2003; Smith and Simmons, 2004; Spaan and Vlassis, 2005; Shani et al., 2007; Kurniawati et al., 2008).
- Monte Carlo tree search

  (Silver and Veness, 2010).

Point-based backup

- For finite horizon V^* is piecewise linear and convex, and for infinite horizons V^* can be approximated arbitrary well by a PWLC value function (Smallwood and Sondik, 1973).
- Given value function V_n and a particular belief point b we can easily compute the vector α^b_{n+1} of HV_n such that

$$\alpha^b_{n+1} = \underset{\{\alpha^k_{n+1}\}_k}{\arg\max}\, b \cdot \alpha^k_{n+1},$$

where $\{\alpha^k_{n+1}\}_{k=1}^{|HV_n|}$ is the (unknown) set of vectors for HV_n. We will denote this operation $\alpha^b_{n+1} = \texttt{backup}(b)$.

Point-based backup

- For finite horizon V^* is piecewise linear and convex, and for infinite horizons V^* can be approximated arbitrary well by a PWLC value function (Smallwood and Sondik, 1973).
- Given value function V_n and a particular belief point b we can easily compute the vector α_{n+1}^b of HV_n such that

$$\alpha_{n+1}^b = \arg\max_{\{\alpha_{n+1}^k\}_k} b \cdot \alpha_{n+1}^k,$$

where $\{\alpha_{n+1}^k\}_{k=1}^{|HV_n|}$ is the (unknown) set of vectors for HV_n. We will denote this operation $\alpha_{n+1}^b = \texttt{backup}(b)$.

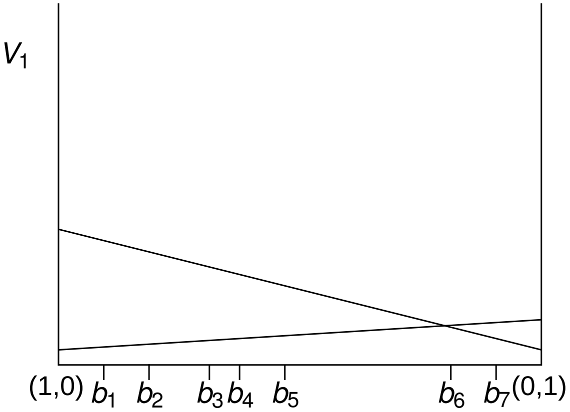
Point-based (approximate) methods

Point-based (approximate) value iteration plans only on a limited set of **reachable** belief points:

1. Let the robot explore the environment.
2. Collect a set B of belief points.
3. Run approximate value iteration on B.

PERSEUS: randomized point-based VI

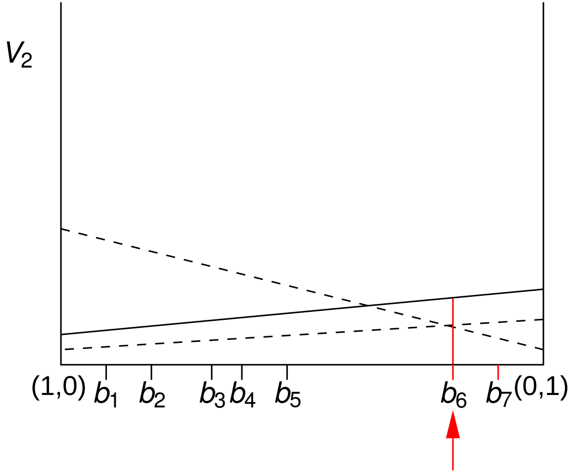
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

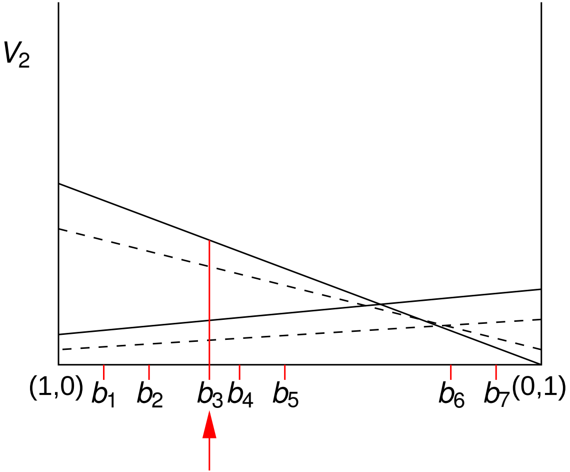
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

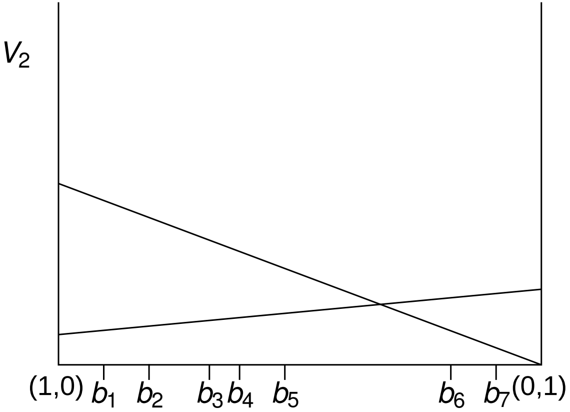
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

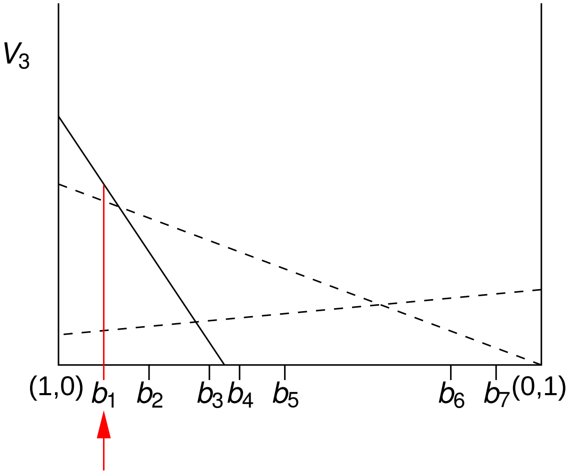
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

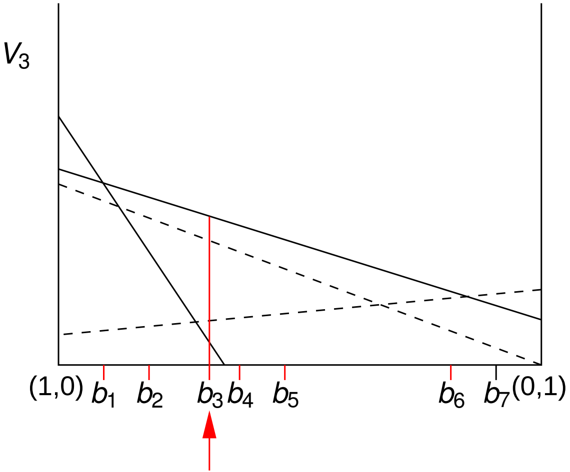
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

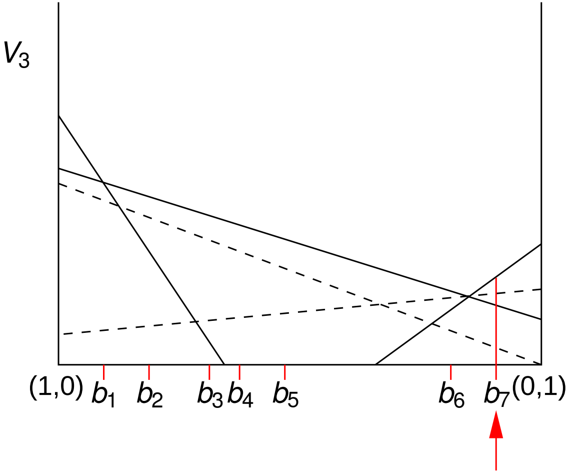
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

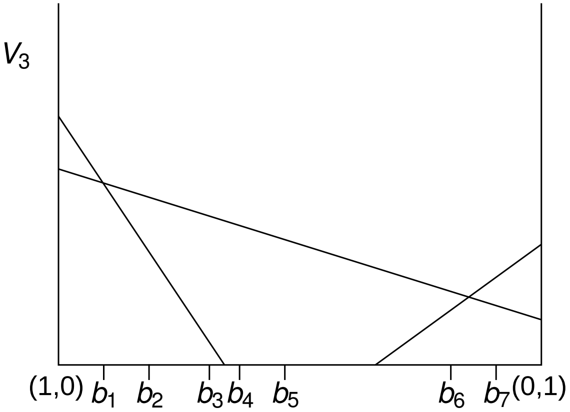
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

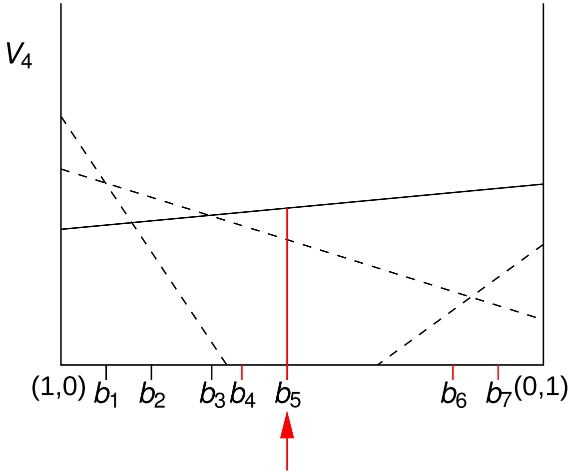
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

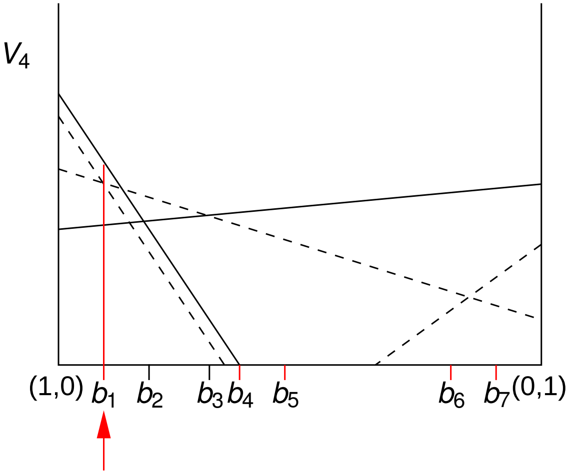
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

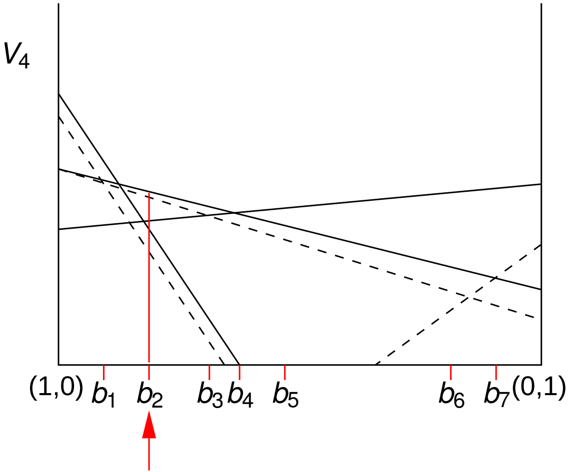
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

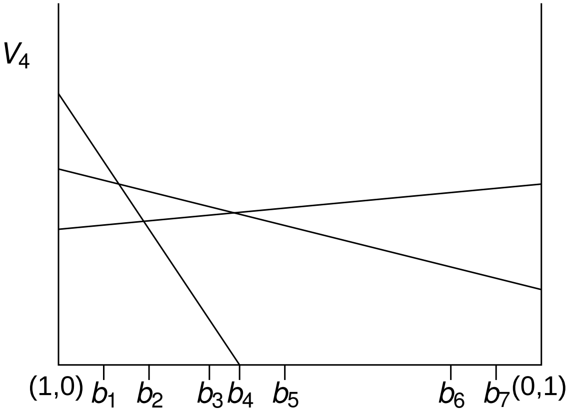
Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

PERSEUS: randomized point-based VI

Idea: at every backup stage improve the value of all $b \in B$.



(Spaan and Vlassis, 2005)

Further reading

- Textbook on reinforcement learning
  - R. S. Sutton and A. G. Barto. “Reinforcement Learning: An Introduction”. MIT Press, 1998.
- Recent book containing chapters on many aspects of decision-theoretic planning (MDPs, POMDPs, Dec-POMDPs):
  - Marco Wiering and Martijn van Otterlo, editors, “Reinforcement Learning: State of the Art”, Springer, 2012.

References I

R. Bellman. *Dynamic programming*. Princeton University Press, 1957.

D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 2nd edition, 2000.

B. Bonet. An epsilon-optimal grid-based algorithm for partially observable Markov decision processes. In *International Conference on Machine Learning*, 2002.

R. I. Brafman. A heuristic variable grid solution method for POMDPs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997.

A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. In *Proc. of International Conference on Intelligent Robots and Systems*, 1996.

A. R. Cassandra, M. L. Littman, and N. L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proc. of Uncertainty in Artificial Intelligence*, 1997.

H. T. Cheng. *Algorithms for partially observable Markov decision processes*. PhD thesis, University of British Columbia, 1988.

A. W. Drake. *Observation of a Markov process through a noisy channel*. Sc.D. thesis, Massachusetts Institute of Technology, 1962.

E. A. Hansen. *Finite-memory control of partially observable systems*. PhD thesis, University of Massachusetts, Amherst, 1998a.

E. A. Hansen. Solving POMDPs by searching in policy space. In *Proc. of Uncertainty in Artificial Intelligence*, 1998b.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

H. Kurniawati, D. Hsu, and W. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, 2008.

M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning*, 1995.

W. S. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39(1):162–175, 1991.

References II

G. E. Monahan. A survey of partially observable Markov decision processes: theory, models and algorithms. *Management Science*, 28(1), Jan. 1982.

R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 1995.

J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2003.

L. K. Platzman. A feasible computational approach to infinite-horizon partially-observed Markov decision problems. Technical Report J-81-2, School of Industrial and Systems Engineering, Georgia Institute of Technology, 1981. Reprinted in working notes AAAI 1998 Fall Symposium on Planning with POMDPs.

P. Poupart and C. Boutilier. Value-directed compression of POMDPs. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003.

P. Poupart and C. Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.

M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

N. Roy, G. Gordon, and S. Thrun. Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research*, 23:1–40, 2005.

S. J. Russell and P. Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, 2nd edition, 2003.

J. K. Satia and R. E. Lave. Markovian decision processes with probabilistic observation of states. *Management Science*, 20(1):1–13, 1973.

G. Shani, R. I. Brafman, and S. E. Shimony. Forward search value iteration for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2007.

D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems 23*, 2010.

S. Singh, T. Jaakkola, and M. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *International Conference on Machine Learning*, 1994.

R. D. Smallwood and E. J. Sondik. The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

References III

T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, 2004.

E. J. Sondik. *The optimal control of partially observable Markov processes*. PhD thesis, Stanford University, 1971.

M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Y. Virin, G. Shani, S. E. Shimony, and R. Brafman. Scaling up: Solving POMDPs through value based clustering. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, 2007.

N. L. Zhang and W. Liu. Planning in stochastic domains: problem characteristics and approximations. Technical Report HKUST-CS96-31, Department of Computer Science, The Hong Kong University of Science and Technology, 1996.

R. Zhou and E. A. Hansen. An improved grid-based approximation algorithm for POMDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2001.

Decision making under uncertainty

Matthijs Spaan[1] and Frans Oliehoek[2]

[1] Delft University of Technology
[2] Maastricht University

Part 3: Multiagent Frameworks

European Agent Systems Summer School (EASSS '13)

www.st.ewi.tudelft.nl/~mtjspaan/tutorialDMuU/

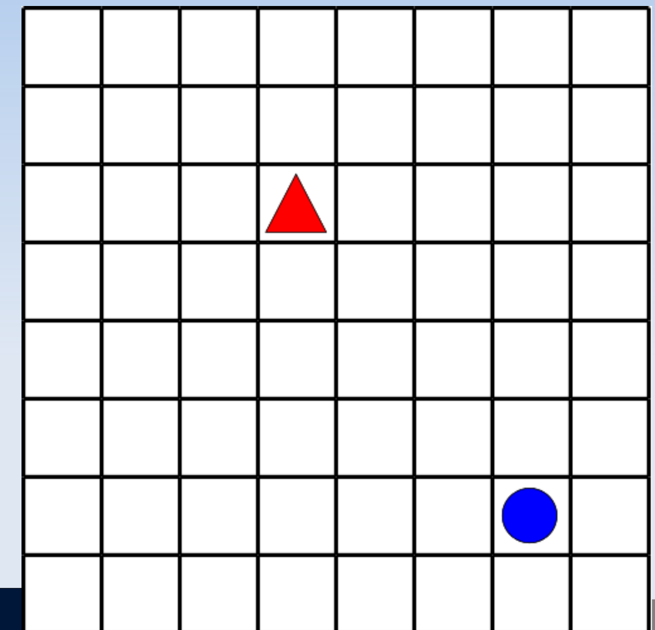
Multiagent Systems (MASs)

Why MASs?

- If we can make intelligent agents, soon there will be many...
- Physically distributed systems: centralized solutions expensive and brittle.
- can potentially provide [Vlassis, 2007,Sycara, 1998]
  - Speedup and efficiency
  - Robustness and reliability (‘graceful degradation’)
  - Scalability and flexibility (adding additional agents)

Example: Predator-Prey Domain

- Predator-Prey domain – still single agent!
  - 1 agent: the predator (blue)
  - prey (red) is part of the environment
  - on a torus ('wrap around world')

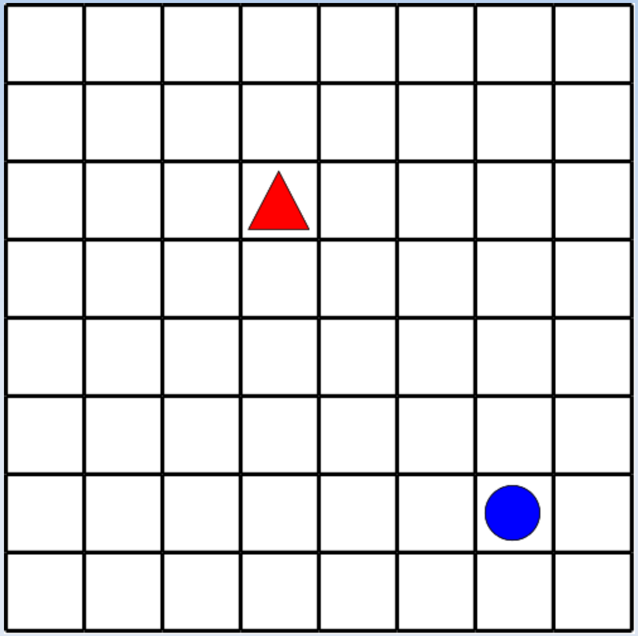


- Formalization:
  - states
  - actions
  - transitions
  - rewards

?

Example: Predator-Prey Domain

- Predator-Prey domain
  - 1 agent: the predator (blue)
  - prey (red) is part of the environment
  - on a torus ('wrap around world')

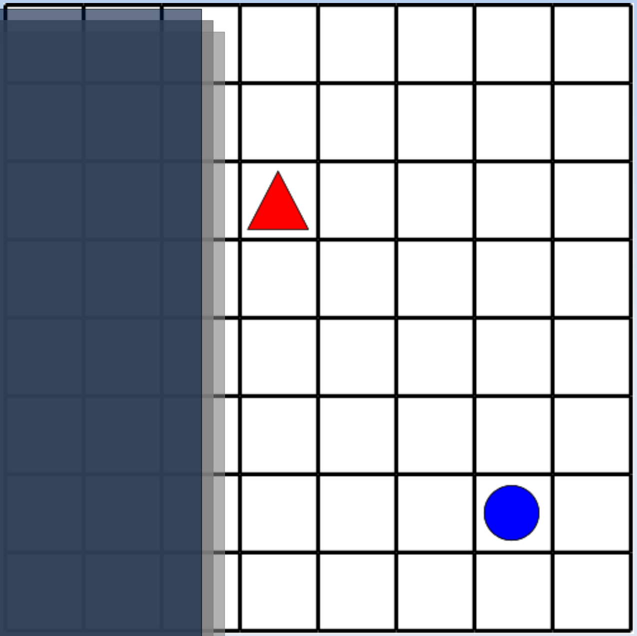


- Formalization:
  - states (-3,4)
  - actions N,W,S,E
  - transitions probability of failing to move, prey moves
  - rewards reward for capturing

Example: Predator-Prey Domain

- Predator-Prey domain
  - 1 agent: the predator (blue)
  - prey (red) is part of the environment
  - on a torus ('wrap around world')

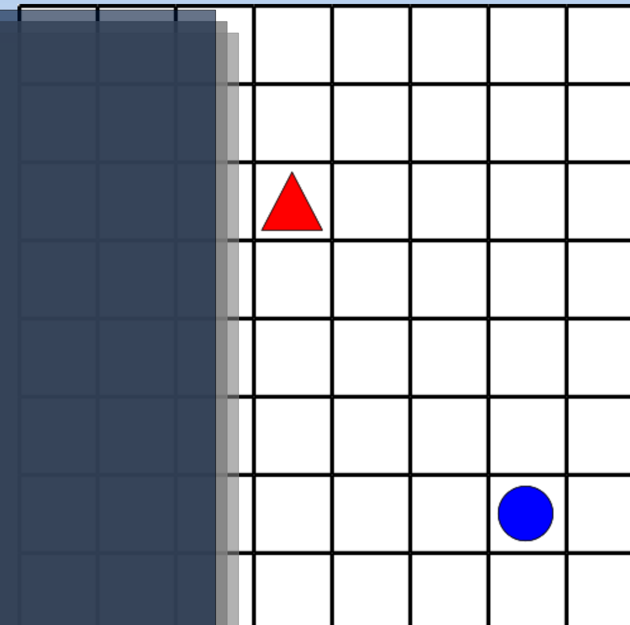
Markov decision process (MDP)



- Formalization:
  - states (-3,4)
  - actions N,W,S,E
  - transitions probability of failing to move, prey moves
  - rewards reward for capturing

Example: Predator-Prey Domain

- Predator-Prey domain



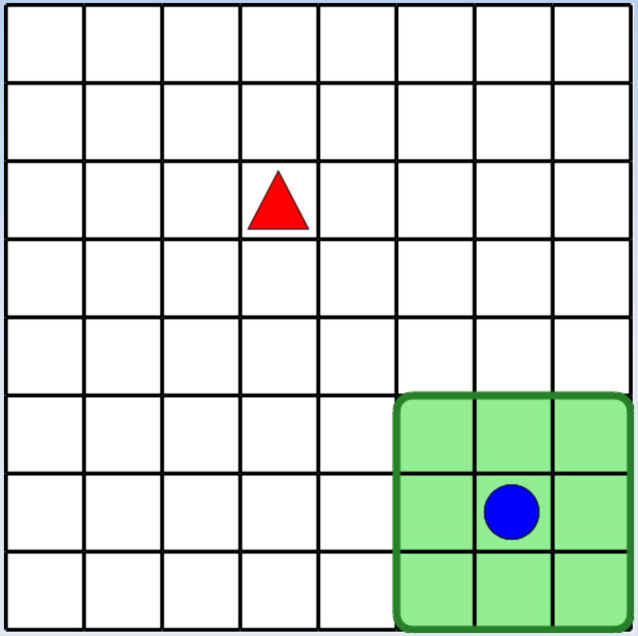
Markov decision process (MDP)

- Markovian state *s*...
- ...which is observed
- policy π maps states → actions
- Value function Q(s,a)
- Value iteration: way to compute it.

- states (-3,4)
- actions N,W,S,E
- transitions probability of failing to move, prey moves
- rewards reward for capturing

Partial Observability

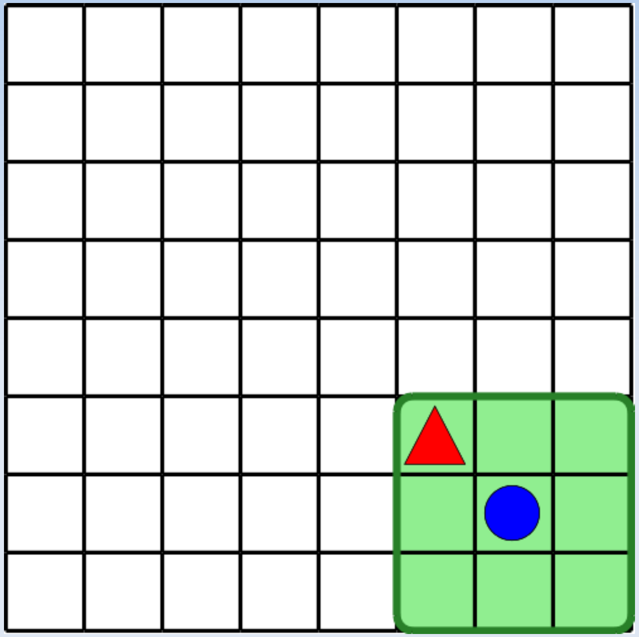
- Now: partial observability
  - E.g., limited range of sight
- MDP + observations
  - explicit observations
  - observation probabilities
    - noisy observations (detection probability)



$o = \text{'}nothing\text{'}$

Partial Observability

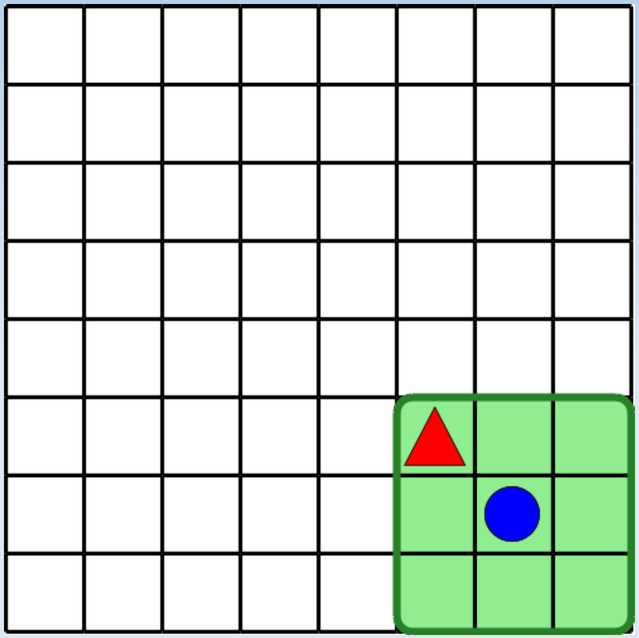
- Now: partial observability
  - E.g., limited range of sight
- MDP + observations
  - explicit observations
  - observation probabilities
    - noisy observations (detection probability)



$o = (-1, 1)$

Partial Observability

- Now: partial observability
  - E.g., limited range of sight
- MDP + observations
  - explicit observations
  - observation probabilities
    - noisy observations (detection probability)



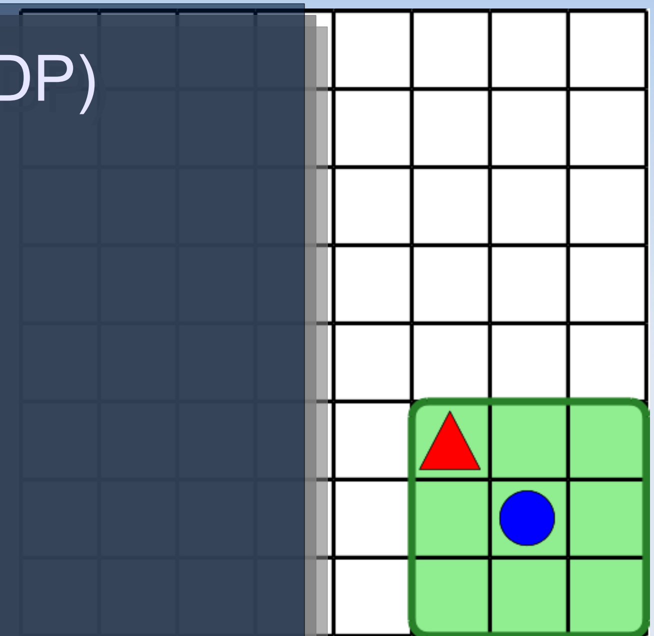
$o = (-1, 1)$

Can not observe the state
→ Need to maintain a belief over states *b(s)*
→ Policy maps beliefs to actions $\pi(b) = a$

Partial Observability

- Now: partial observability
  - E.g., [illegible]
- MDP + observations
  - explicit observations
  - observation probabilities
    - noisy observations (detection probability)

Partially Observable MDP (POMDP)

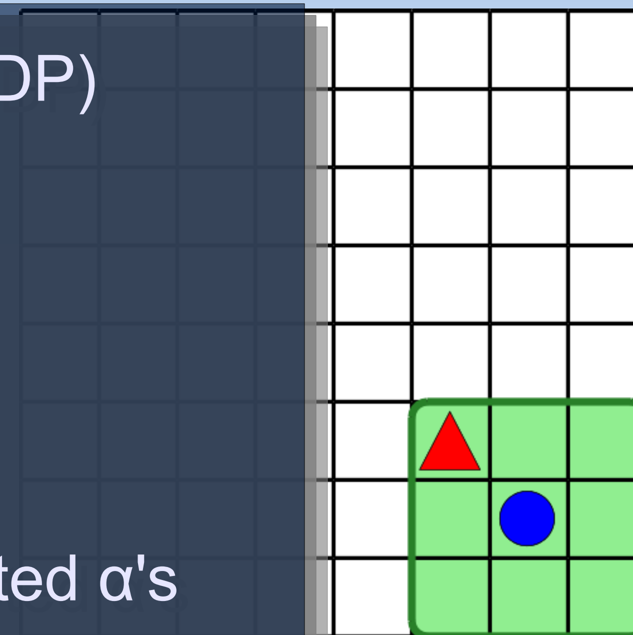


$o = (-1, 1)$

Can not observe the state
→ Need to maintain a belief over states *b(s)*
→ Policy maps beliefs to actions $\pi(b) = a$

Partial Observability

- Now: partial observability



$o=(-1,1)$

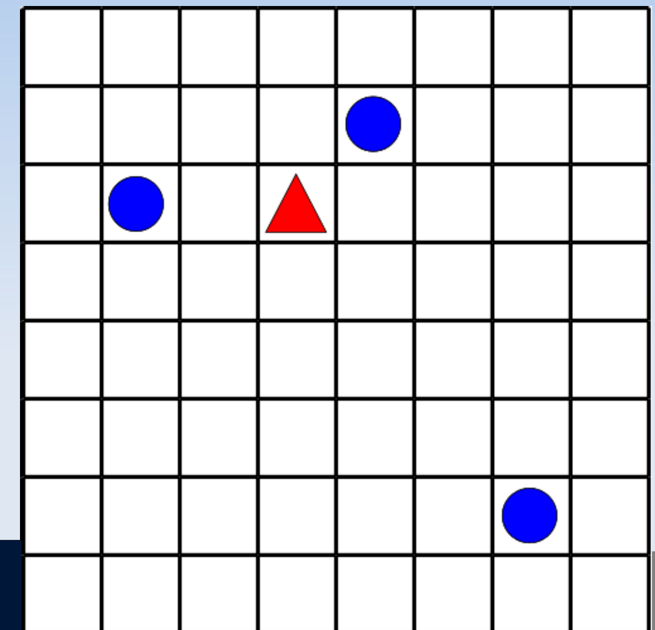
Partially Observable MDP (POMDP)

- reduction → continuous state MDP (in which the belief **is** the state)
- Value iterations:
  - make use of α-vectors (correspond to complete policies)
  - perform pruning: eliminate dominated α's

Can not observe the state
→ Need to maintain a belief over states *b(s)*
→ Policy maps beliefs to actions $\pi(b)=a$

Multiple Agents

- Now: multiple agents
  - fully observable

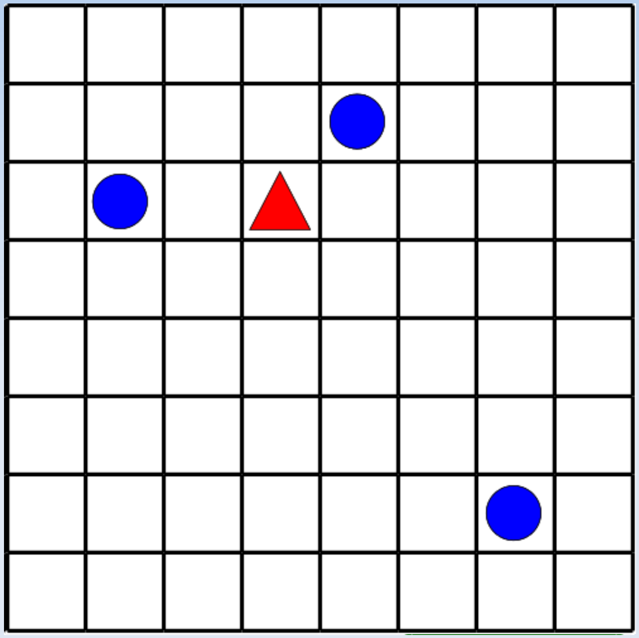


- Formalization:
  - states
  - actions
  - **joint** actions
  - transitions
  - rewards

?

Multiple Agents

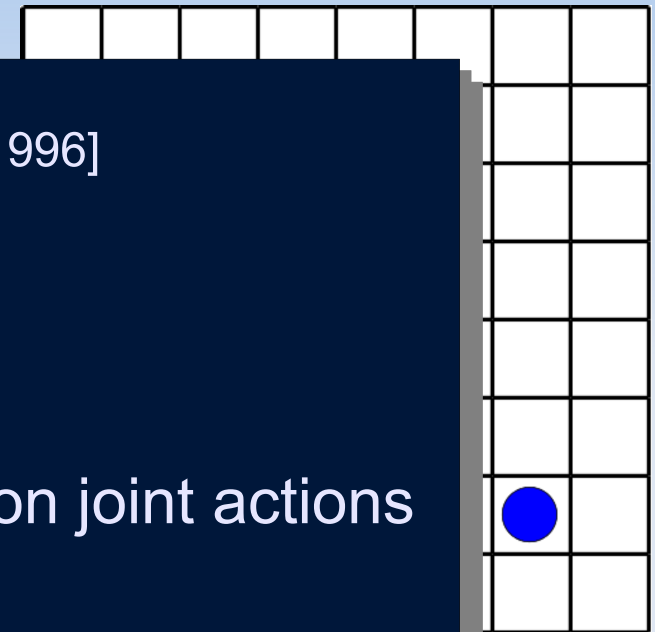
- Now: multiple agents
  - fully observable



- Formalization:
  - states ((3,-4), (1,1), (-2,0))
  - actions {N,W,S,E}
  - **joint** actions {(N,N,N), (N,N,W),...,(E,E,E)}
  - transitions probability of failing to move, prey moves
  - rewards reward for capturing jointly

Multiple Agents

- Now: multiple agents
  - fully observable



Multiagent MDP [Boutilier 1996]

- Differences with MDP
  - *n* agents
  - joint actions $a = \langle a_1, a_2, \ldots, a_n \rangle$
  - transitions and rewards depend on joint actions
- Solution:
  - Treat as normal MDP with 1 'puppeteer agent'
  - Optimal policy $\pi(s) = a$
  - Every agent executes its part

- Fo
  - es
  - rewards reward for capturing jointly

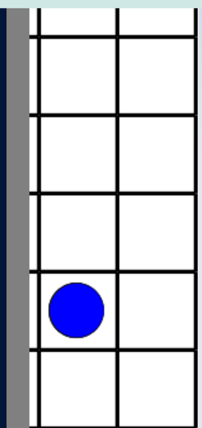
Multiple Agents

- Now: multiple agents
  - fully observable
- Fo

Multiagent

Catch: …?

- Differences with MDP
  - *n* agents
  - joint actions $a = \langle a_1, a_2, \ldots, a_n \rangle$
  - transitions and rewards depend on joint actions
- Solution:
  - Treat as normal MDP with 1 'puppeteer agent'
  - Optimal policy $\pi(s) = a$
  - Every agent executes its part



es

- rewards reward for capturing jointly

Multiple Agents

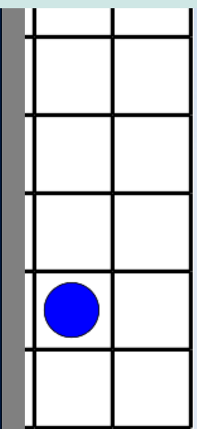
- Now: multiple agents
  - fully observable

Multiagent

- Differences with MDP
  - *n* agents
  - joint actions $a = \langle a_1, a_2, \ldots, a_n \rangle$
  - transitions and rewards depend on joint actions
- Solution:
  - Treat as normal MDP with 1 'puppeteer agent'
  - Optimal policy $\pi(s) = a$
  - Every agent executes its part

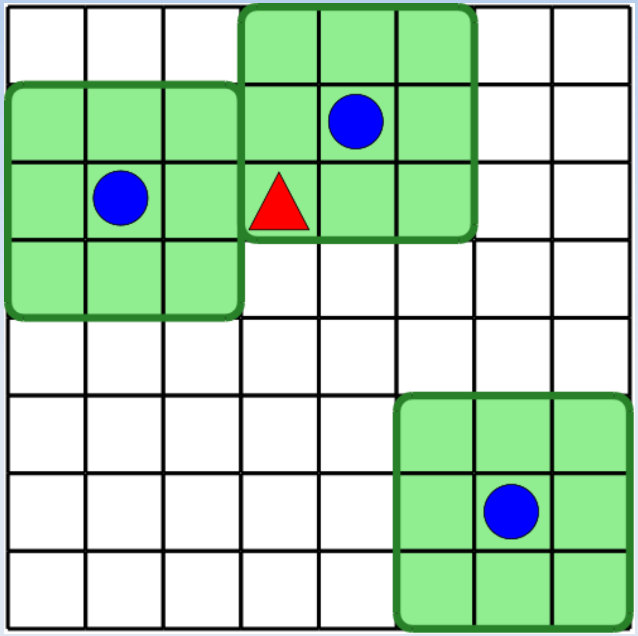
Catch: number of joint actions is **exponential**!
(but other than that, conceptually simple.)

- Fo
  - ...es
  - rewards reward for capturing jointly



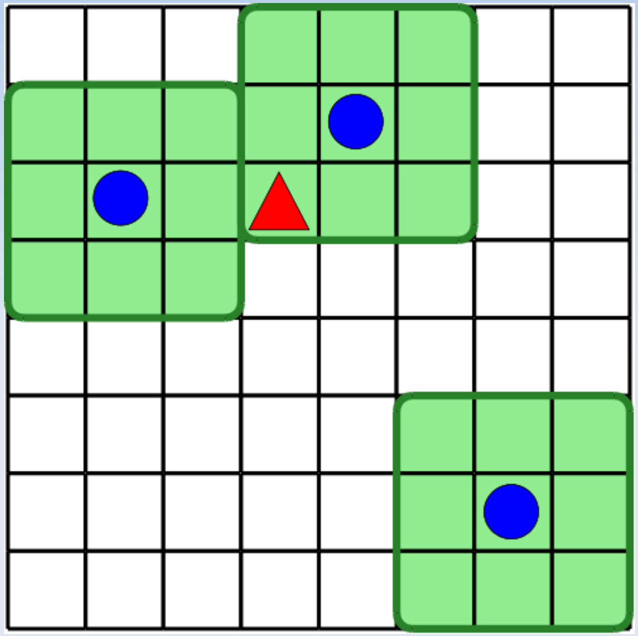
Multiple Agents & Partial Observability

- Now: Both
  - partial observability
  - multiple agents



Multiple Agents & Partial Observability

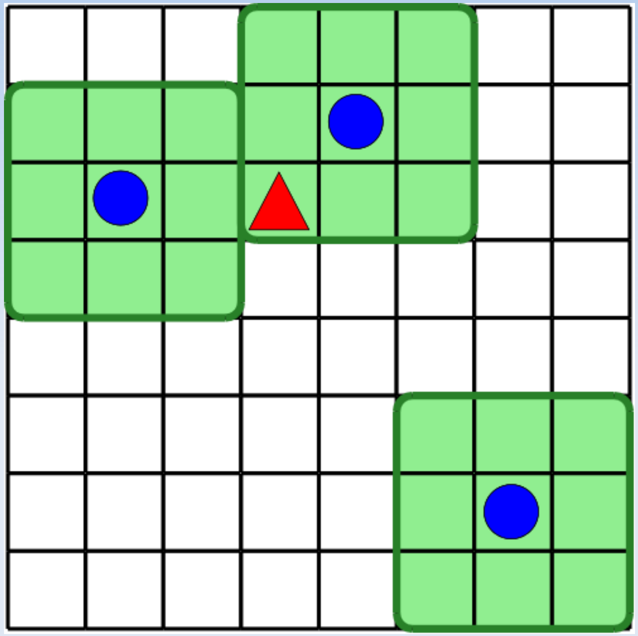
- Now: Both
  - partial observability
  - multiple agents
- Decentralized POMDPs (Dec-POMDPs) [Bernstein et al. 2002]
- both
  - joint actions and
  - joint observations



Multiple Agents & Partial Observability

- Again we can make a reduction...

  any idea?

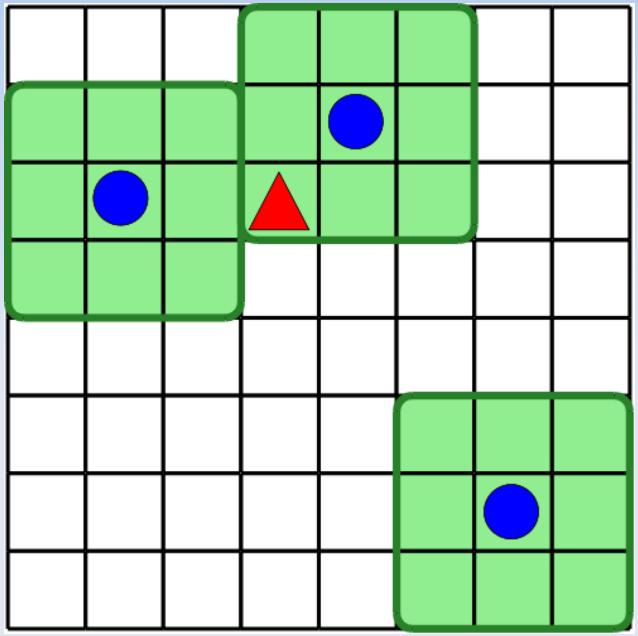


Multiple Agents & Partial Observability

- Again we can make a reduction...

  Dec-POMDPs → MPOMDP

  (multiagent POMDP)

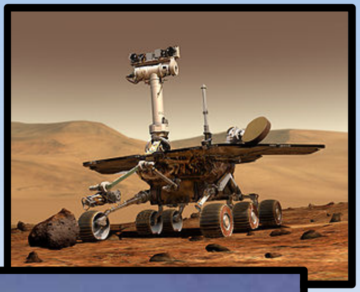


- 'puppeteer' agent that
  - receives joint observations
  - takes joint actions
- requires broadcasting observations!
  - instantaneous, cost-free, noise-free communication → optimal [Pynadath and Tambe 2002]
- Without such communication: no easy reduction.

The Dec-POMDP Model

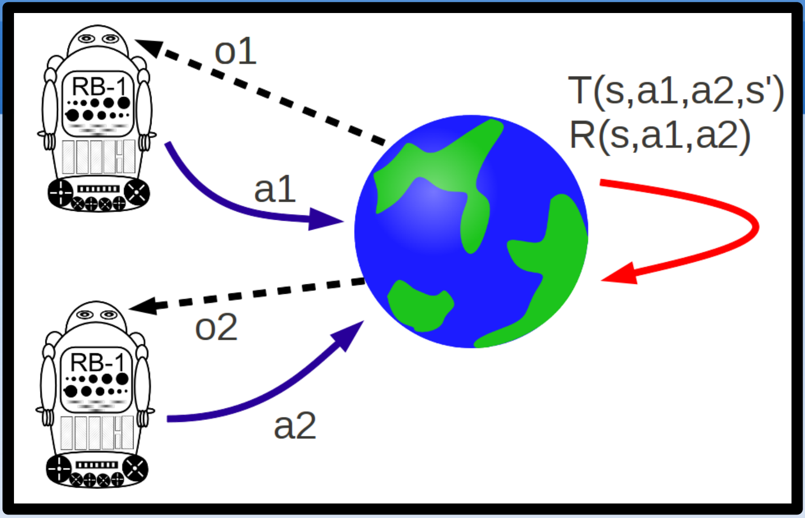
Acting Based On Local Observations

- MPOMDP: Act on global information
- Can be impractical:
  - communication not possible
  - significant cost (e.g battery power)
  - not instantaneous or noise free
  - scales poorly with number of agents!



- Alternative: act based only on local observations
  - Other side of the spectrum: no communication at all
  - (Also other intermediate approaches: delayed communication, stochastic delays)

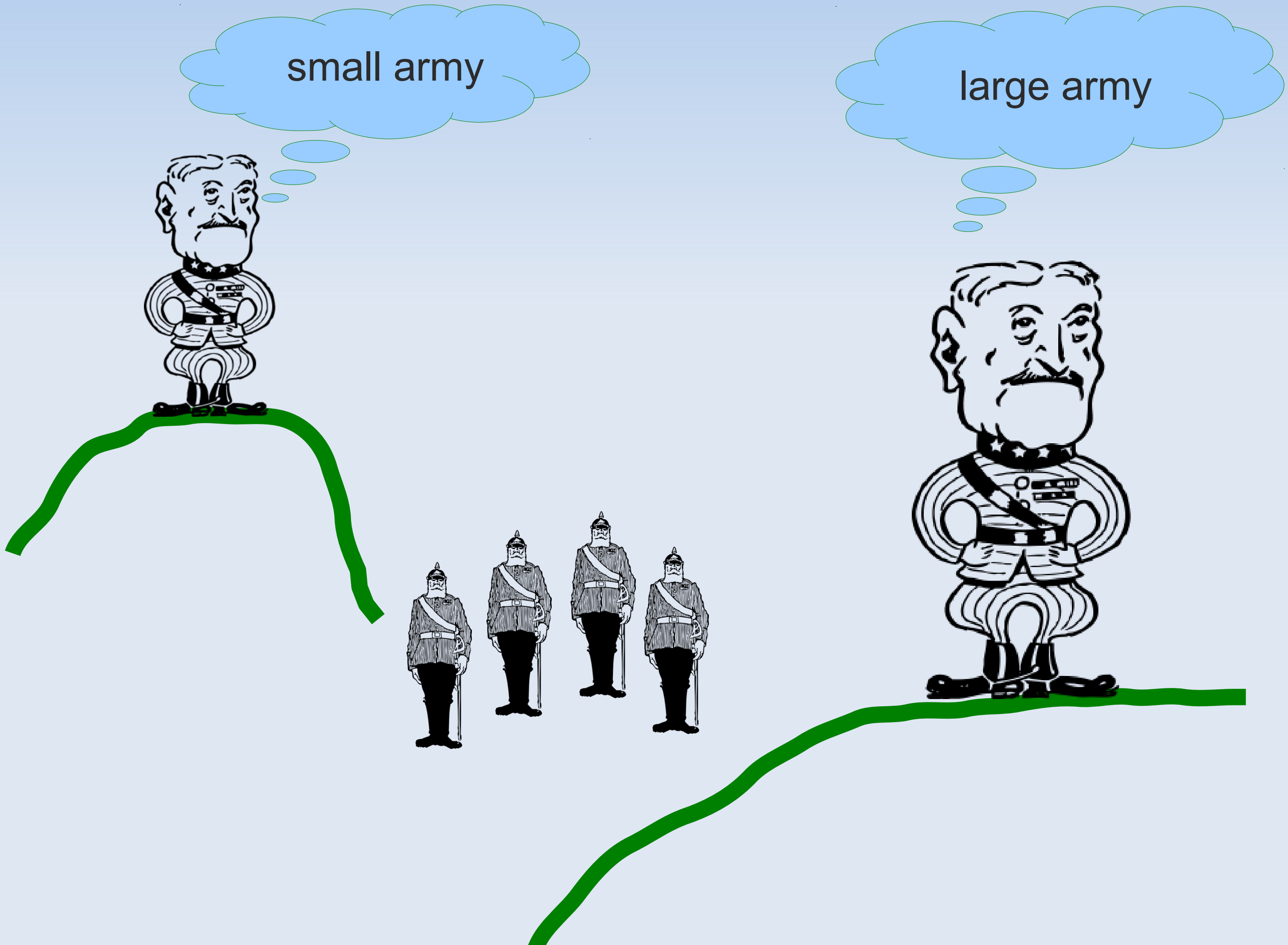
Formal Model



- A Dec-POMDP
  - $\langle S, A, P_T, O, P_O, R, h \rangle$
  - n agents
  - S – set of states
  - A – set of **joint** actions $\quad a = \langle a_1, a_2, \ldots, a_n \rangle$
  - P_T – transition function $\quad P(s' | s, a)$
  - O – set of **joint** observations $\quad o = \langle o_1, o_2, \ldots, o_n \rangle$
  - P_O – observation function $\quad P(o | a, s')$
  - R – reward function $\quad R(s, a)$
  - h – horizon (finite)

Running Example

- 2 generals problem



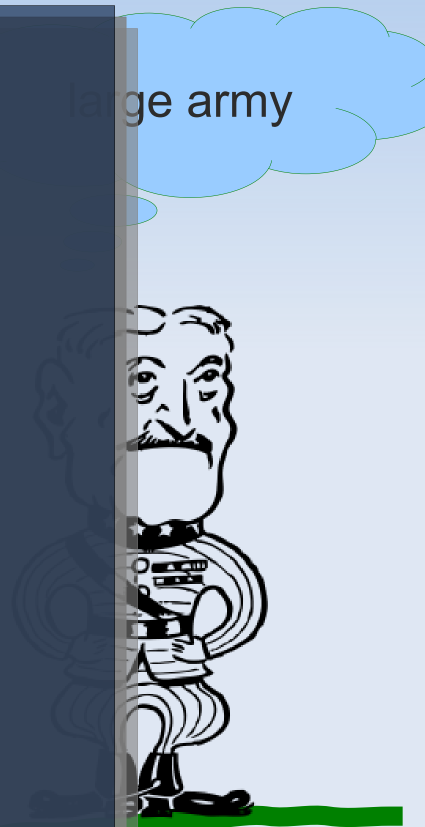
Running Example

- 2 generals problem

S – { s_L, s_S }
A_i – { (O)bserve, (A)ttack }
O_i – { (L)arge, (S)mall }

Transitions
- Both Observe: no state change
- At least 1 Attack: reset with 50% probability

Observations
- Probability of correct observation: 0.85
- E.g., $P(<L, L> | s_L) = 0.85 * 0.85 = 0.7225$



Running Example

- 2 generals problem

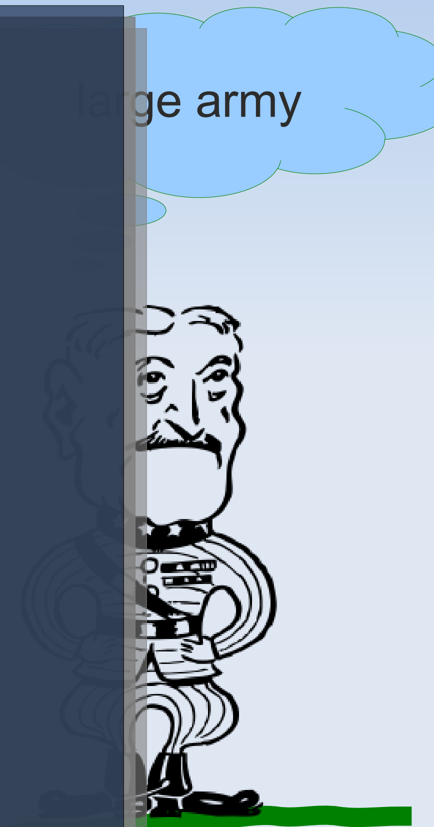
S – { s_L, s_S }

A_i – { (O)bserve, (A)ttack }

O_i – { (L)arge, (S)mall }

Rewards

- 1 general attacks: he loses the battle
  - R(*,<A,O>) = -10
- Both generals Observe: small cost
  - R(*,<O,O>) = -1
- Both Attack: depends on state
  - R(s_L,<A,A>) = -20
  - R(s_R,<A,A>) = +5



Running Example

- 2 generals problem

S – { s_L, s_S }

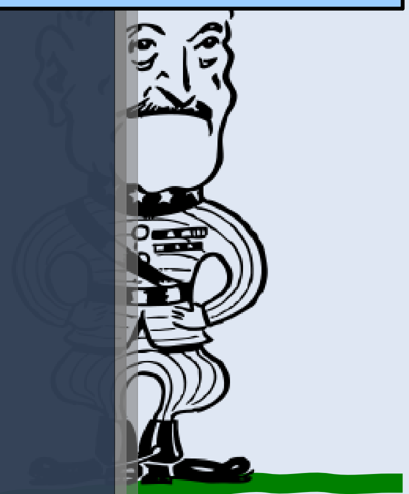
A_i – { (O)bserve, (A)ttack }

O_i – { (L)arge, (S)mall }

Rewards

- 1 general attacks: he loses the battle
  - R(*,<A,O>) = -10
- Both generals Observe: small cost
  - R(*,<O,O>) = -1
- Both Attack: depends on state
  - R(s_L,<A,A>) = -20
  - R(s_R,<A,A>) = +5

suppose h=3,
what do you think is optimal in this problem?

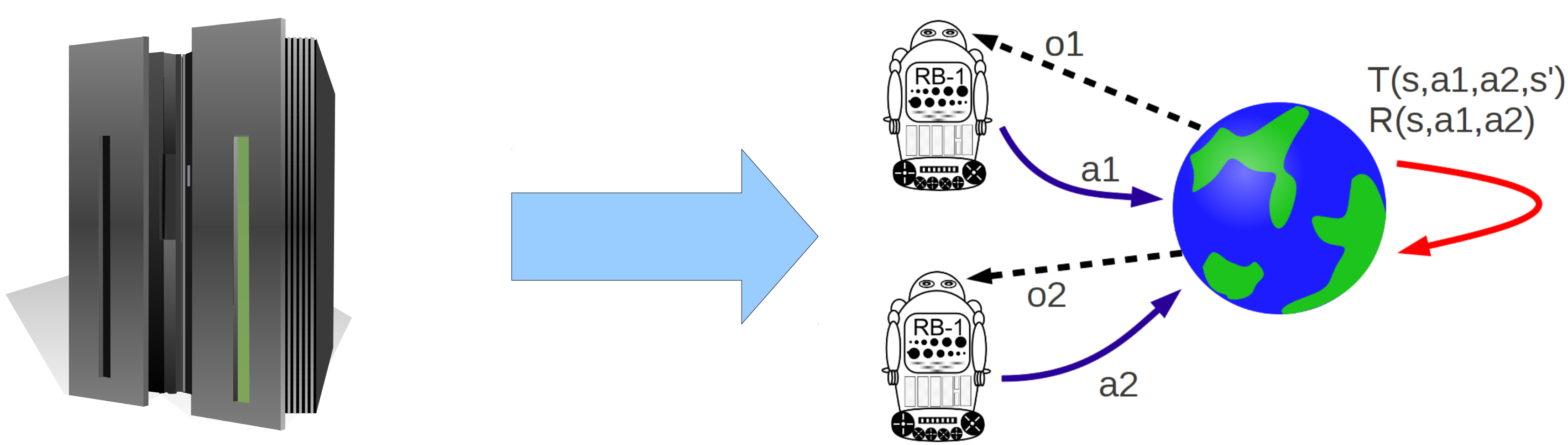


Off-line / On-line phases

- off-line planning, on-line execution is decentralized

Planning Phase

Execution Phase

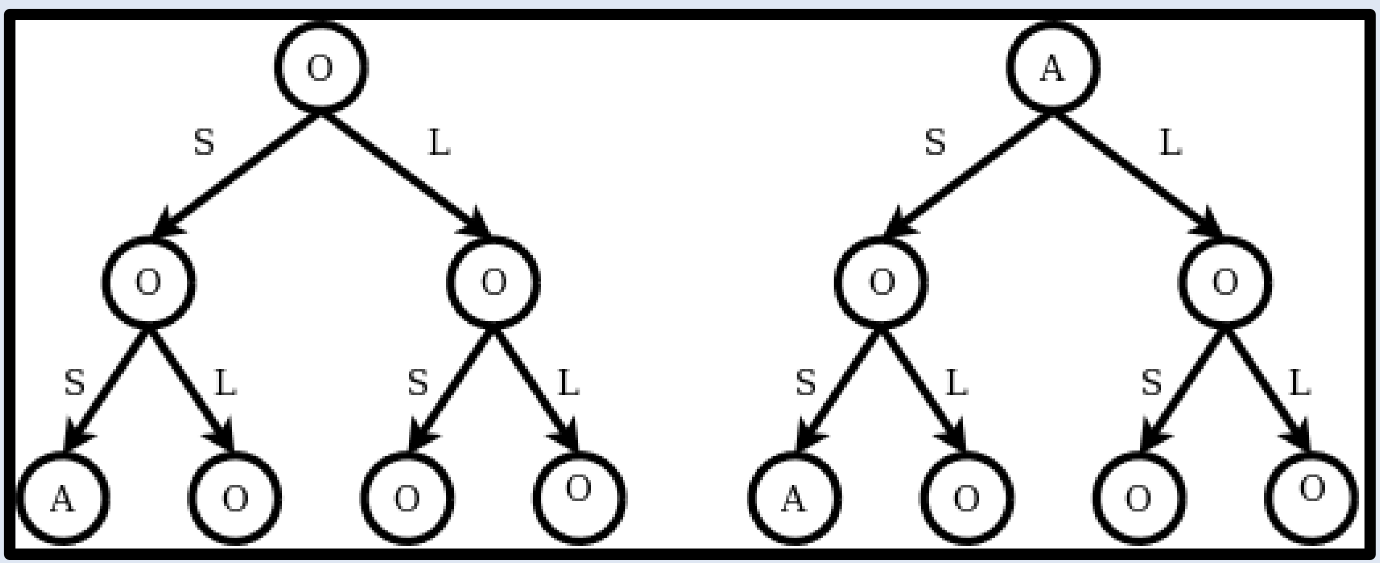


$$\pi = \langle \pi_{1,} \pi_2 \rangle$$

Policy Domain

- What do policies look like?
  - In general histories → actions
  - before: more compact representations...
- Now, this is difficult: no such representation known!

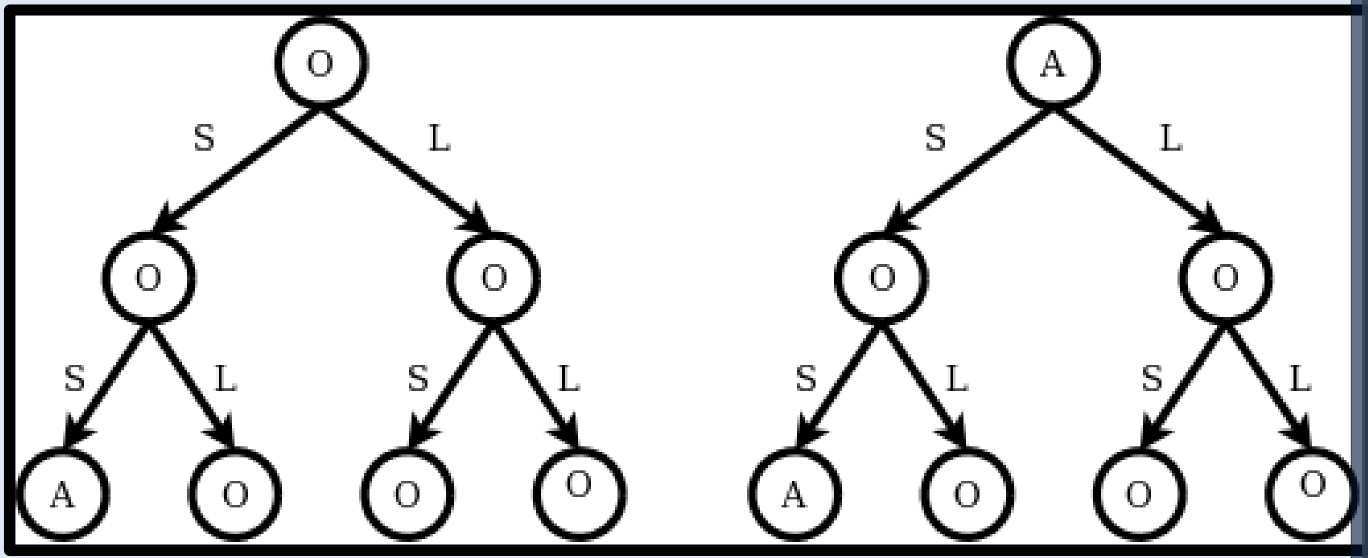
  → So we will be stuck with histories



Policy Domain

- What do policies look like?
  - In general histories → actions
  - before: more compact representations...
- Now, this is difficult: no such representation known!

  → So we will be stuck with histories



Most general, AOHs:

$$\left(a_i^0, o_i^1, a_i^1, \ldots, a_i^{t-1}, o_i^t\right)$$

But: can restrict to deterministic policies
→ only need OHs:

$$\vec{o}_i = \left(o_i^1, \ldots, o_i^t\right)$$

No Compact Representation?

There are a number of types of beliefs considered

- **Joint Belief**, *b(s)* (as in MPOMDP) [Pynadath and Tambe 2002]
  - compute b(s) using joint actions and observations
  - Problem: ?

No Compact Representation?

There are a number of types of beliefs considered

- **Joint Belief**, *b(s)* (as in MPOMDP) [Pynadath and Tambe 2002]
  - compute b(s) using joint actions and observations
  - Problem: agents do not know those during execution

No Compact Representation?

There are a number of types of beliefs considered

- **Joint Belief**, *b(s)* (as in MPOMDP) [Pynadath and Tambe 2002]
  - compute b(s) using joint actions and observations
  - Problem: agents do not know those during execution
- **Multiagent belief**, $b_i(s,q_{-i})$ [Hansen et al. 2004]
  - belief over (future) policies of other agents
  - Need to be able to predict the other agents!
    - for belief update $P(s'|s,a_i,a_{-i})$, $P(o|a_i,a_{-i},s')$, and prediction of $R(s,a_i,a_{-i})$
  - form of those other policies? most general: $\pi_j : \vec{o}_j \rightarrow a_j$
  - if they use beliefs? → infinite recursion of beliefs!

Goal of Planning

- **Find** the **optimal** joint policy $\pi^* = \langle \pi_{1,} \pi_2 \rangle$
  - where individual policies map OHs to actions $\pi_i : \vec{O}_i \rightarrow A_i$
- What is the optimal one?
  - Define **value** as the expected sum of rewards:

$$V(\pi) = \boldsymbol{E}\left[\sum_{t=0}^{h-1} R(s,a) \mid \pi, b^0\right]$$

  - optimal joint policy is one with maximal value (can be more that achieve this)

Goal of Planning

- **Find the optimal joint policy**
  - where individual policies map OHs to actions $\pi_i : \vec{O}_i \rightarrow A_i$
- What is the optimal one?
  - Define value as the expected sum of rewards:
  - optimal joint policy is one with maximal value (can be more that achieve this)

Optimal policy for 2 generals, h=3

value=-2.86743

() --> observe
(o_small) --> observe
(o_large) --> observe
(o_small,o_small) --> attack
(o_small,o_large) --> attack
(o_large,o_small) --> attack
(o_large,o_large) --> observe

() --> observe
(o_small) --> observe
(o_large) --> observe
(o_small,o_small) --> attack
(o_small,o_large) --> attack
(o_large,o_small) --> attack
(o_large,o_large) --> observe

Goal of Planning

Optimal policy for 2 generals, h=3

value=-2.86743

() --> observe
(o_small) --> observe
(o_large) --> observe
(o_small,o_small) --> attack
(o_small,o_large) --> attack
(o_large,o_small) --> attack
(o_large,o_large) --> observe

() --> observe
(o_small) --> observe
(o_large) --> observe
(o_small,o_small) --> attack
(o_small,o_large) --> attack
(o_large,o_small) --> attack
(o_large,o_large) --> observe

conceptually:

what should policy optimize to allow for good coordination (thus high value)

?

Coordination vs. Exploitation of Local Information

- Inherent trade-off

  coordination vs. **exploitation of local information**

  - Ignore own observations → 'open loop plan'
    - E.g., “ATTACK on 2nd time step”

      \+ maximally predictable
      \- low quality
  - Ignore coordination
    - E.g., compute an individual belief $b_i(s)$ and execute the MPOMDP policy
      \+ uses local information
      \- likely to result in mis-coordination

      $$b_i(s) = \sum_{q_{-i}} b(s, q_{-i})$$
  - Optimal policy π^* should balance between these.

Planning Methods

Brute Force Search

- We can compute the value of a joint policy $V(\pi)$
  - using a Bellman-like equation [Oliehoek 2012]
- So the **stupidest algorithm** is:
  - compute $V(\pi)$, for all π
  - select a π with maximum value
- Number of joint policies is huge!
  (doubly exponential in horizon *h*)
- Clearly intractable...

h	num. joint policies
1	4
2	64
3	16384
4	1.0737e+09
5	4.6117e+18
6	8.5071e+37
7	2.8948e+76
8	3.3520e+153

Brute Force Search

- We can compute the value of a joint policy $V(\pi)$
  - using a Bellman-like equation [Oliehoek 2012]

No easy way out...

The problem is
NEXP-complete [Bernstein et al. 2002]

most likely (assuming EXP != NEXP)
doubly exponential time required.

(doubly exponential in horizon h)

- Clearly intractable...

h	num. joint policies
1	4
2	64
3	16384
4	1.0737e+09
5	4.6117e+18
6	8.5071e+37
7	2.8948e+76
8	3.3520e+153

Brute Force Search

- We can compute the value of a joint policy *V(π)*
  - using a Bellman-like equation [Oliehoek 2012]
- So the s… …rithm is:
  - compute *V(π)* for all *π*
- (doubly exponential in horizon *h*)
- Clearly intracta…

No easy way out...

The problem is
NEXP-complete [Bernstein et al. 2002]

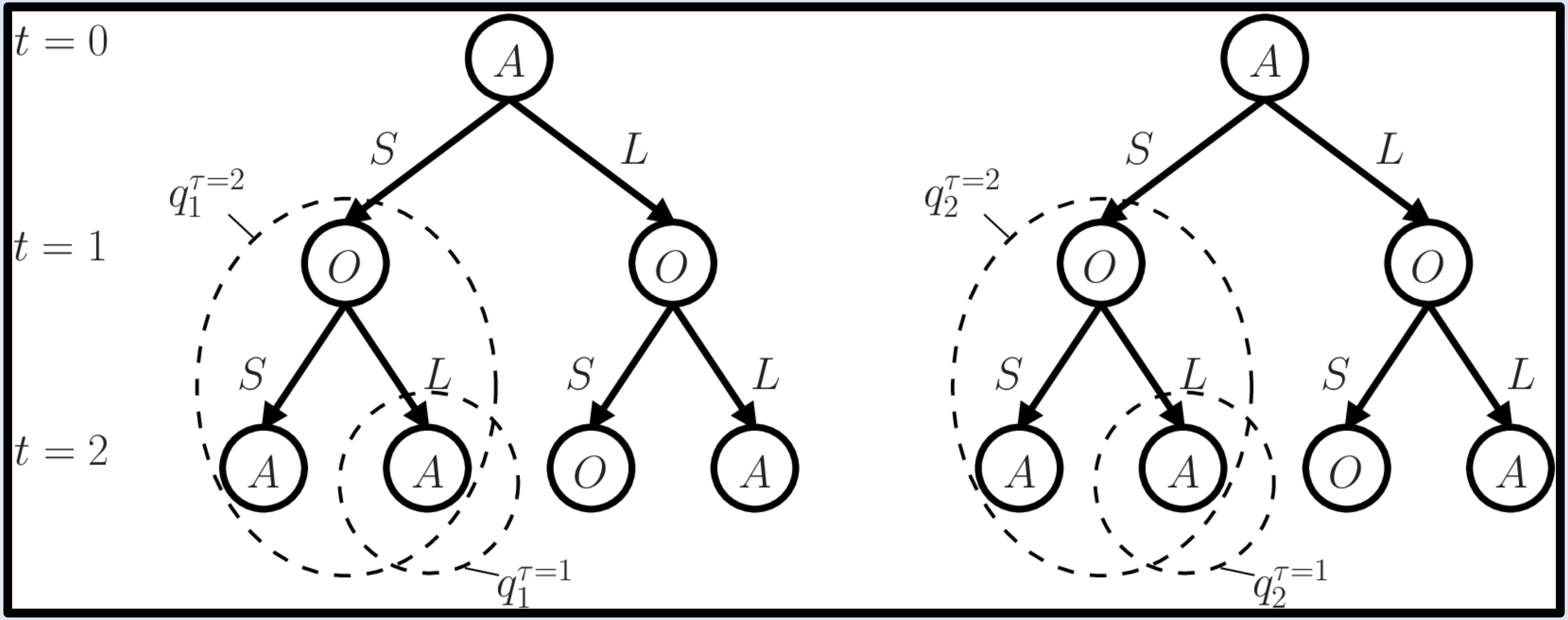
most likely (assuming EXP != NEXP)
doubly exponential time required.

h	num. joint policies
1	4
2	64
3	16384
4	1.0737e+09
5	4.6117e+18
6	8.5071e+37
7	2.8948e+76

- Still, there are better algorithms that work better for at least some problems...
- Useful to understand what optimal really means!
  (trying to compute it helps understanding)

Dynamic Programming – 1

- Generate all policies in a special way:
  - from 1 stage-to-go policies $Q^{\tau=1}$
  - construct all 2-stages-to-go policies $Q^{\tau=2}$, etc.



Dynamic Programming – 1

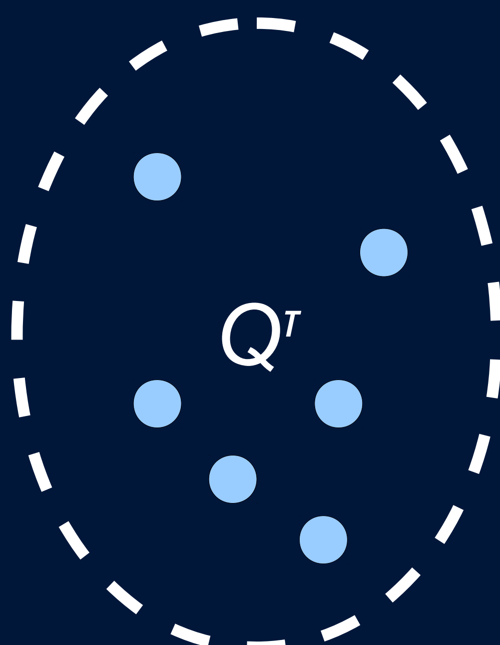
- Generate all policies in a special way:
  - from 1 stage-to-go policies $Q^{\tau=1}$
  - [redacted] etc.

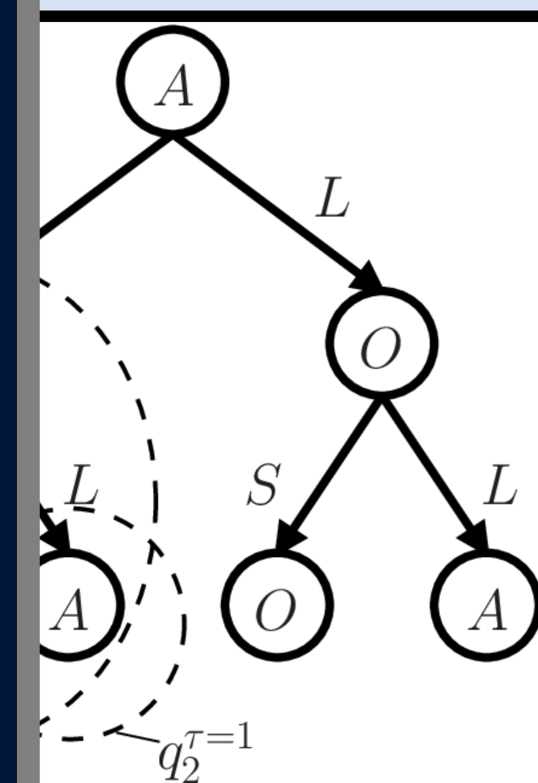
Exhaustive backup operation

Dynamic Programming – 1

- Generate all policies in a special way:
  - from 1 stage-to-go policies $Q^{\tau=1}$
  - … etc.

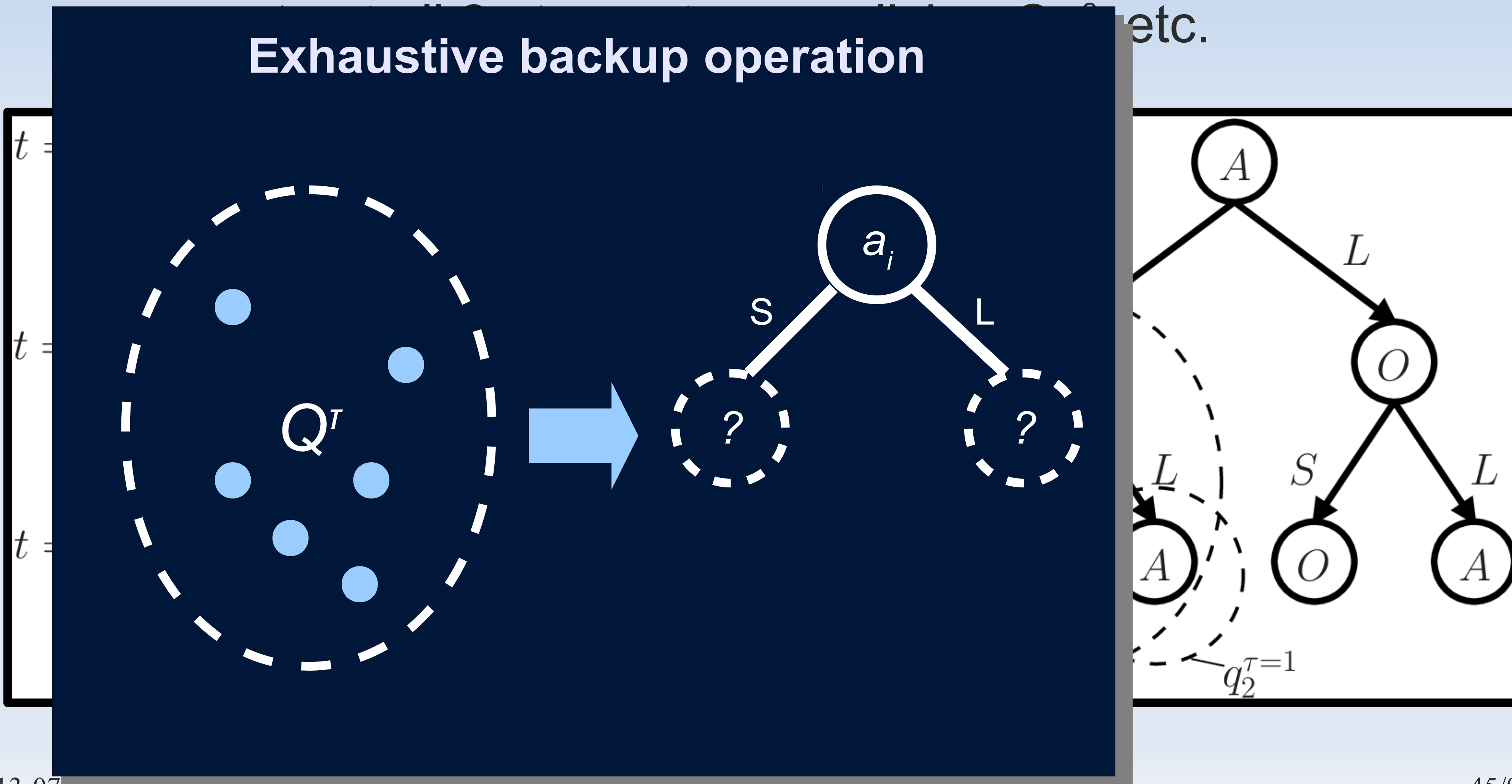
Exhaustive backup operation





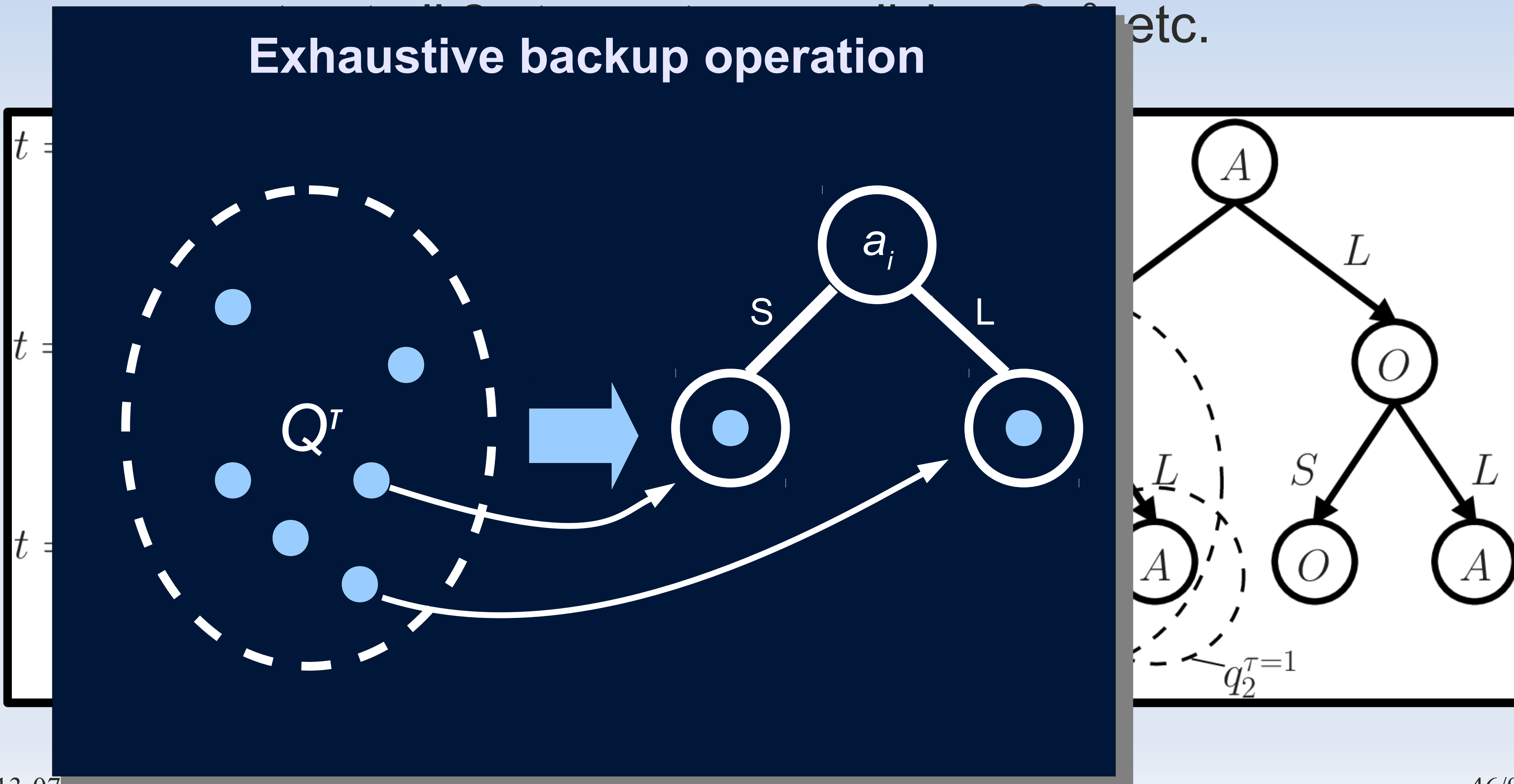
Dynamic Programming – 1

- Generate all policies in a special way:
  - from 1 stage-to-go policies $Q^{\tau=1}$
  - etc.



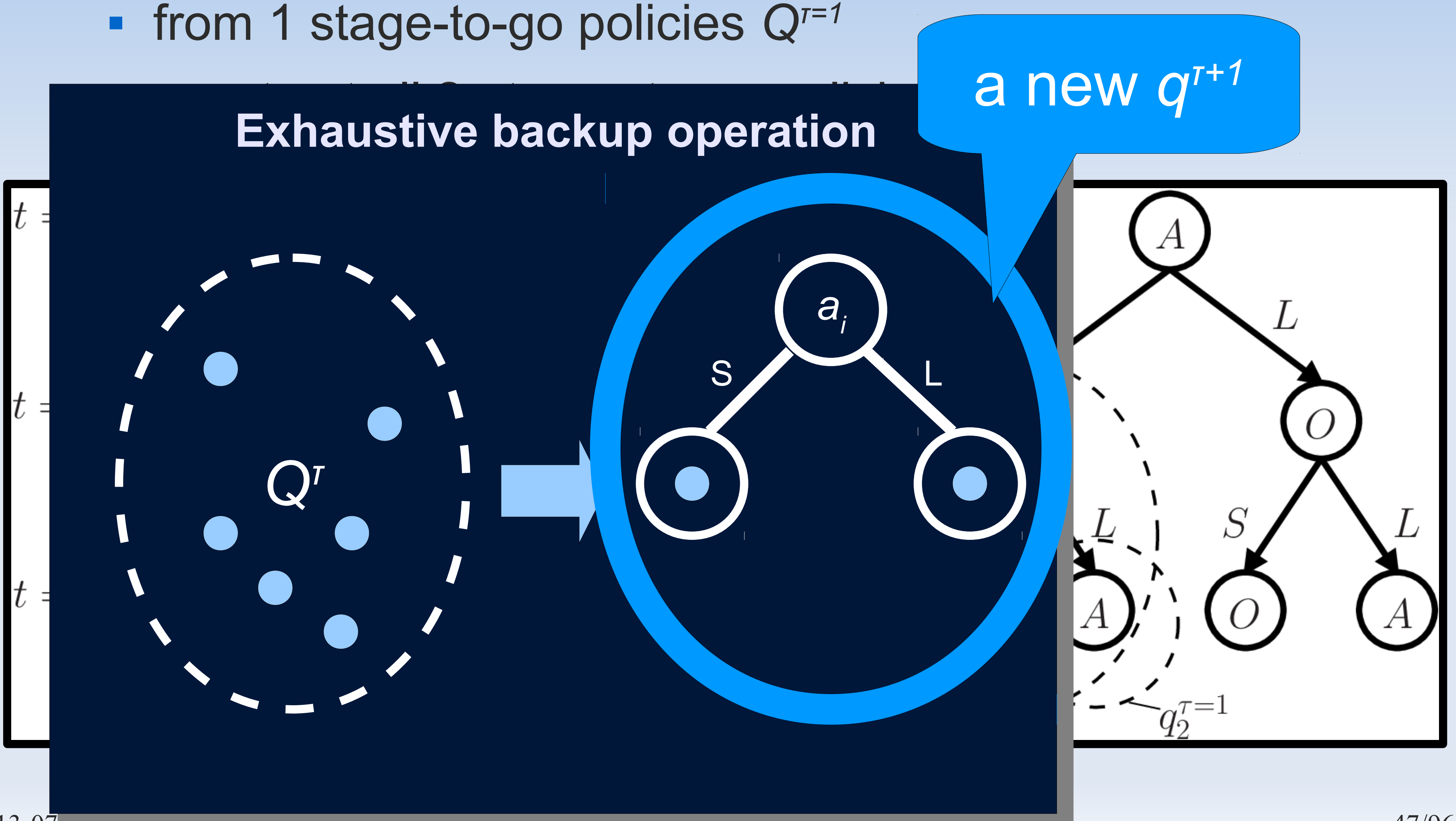
Dynamic Programming – 1

- Generate all policies in a special way:
  - from 1 stage-to-go policies $Q^{\tau=1}$
  - … etc.



Dynamic Programming – 1

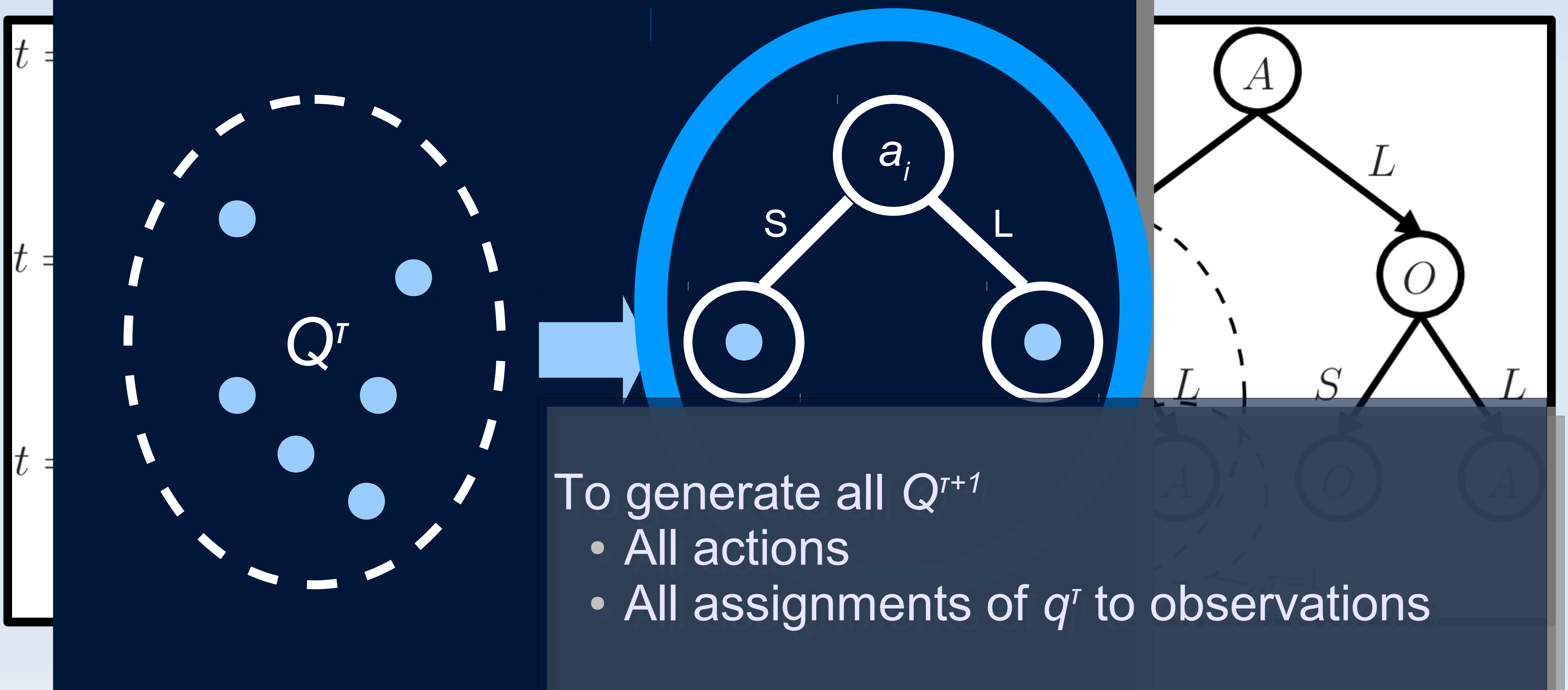
- Generate all policies in a special way:
  - from 1 stage-to-go policies $Q^{\tau=1}$



Dynamic Programming – 1

- Generate all policies in a special way:
  - from 1 stage-to-go policies $Q^{\tau=1}$
  - etc.

Exhaustive backup operation

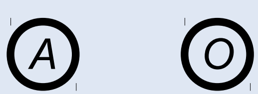


To generate all $Q^{\tau+1}$
- All actions
- All assignments of q^{τ} to observations

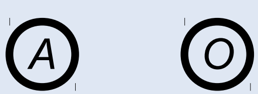
Dynamic Programming – 2

- (obviously) this scales very poorly...

$Q_1^{\tau=1}$

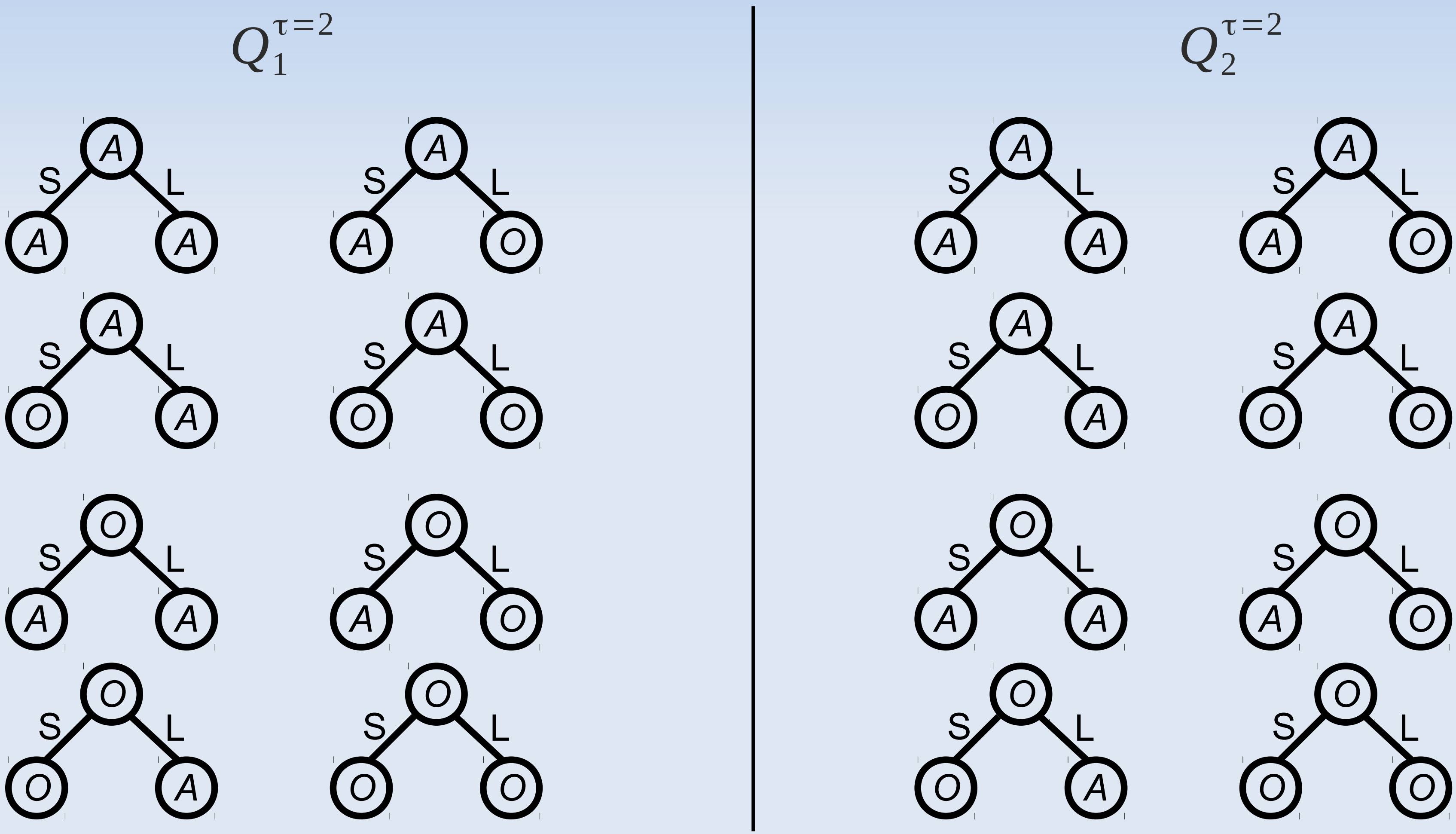


$Q_2^{\tau=1}$



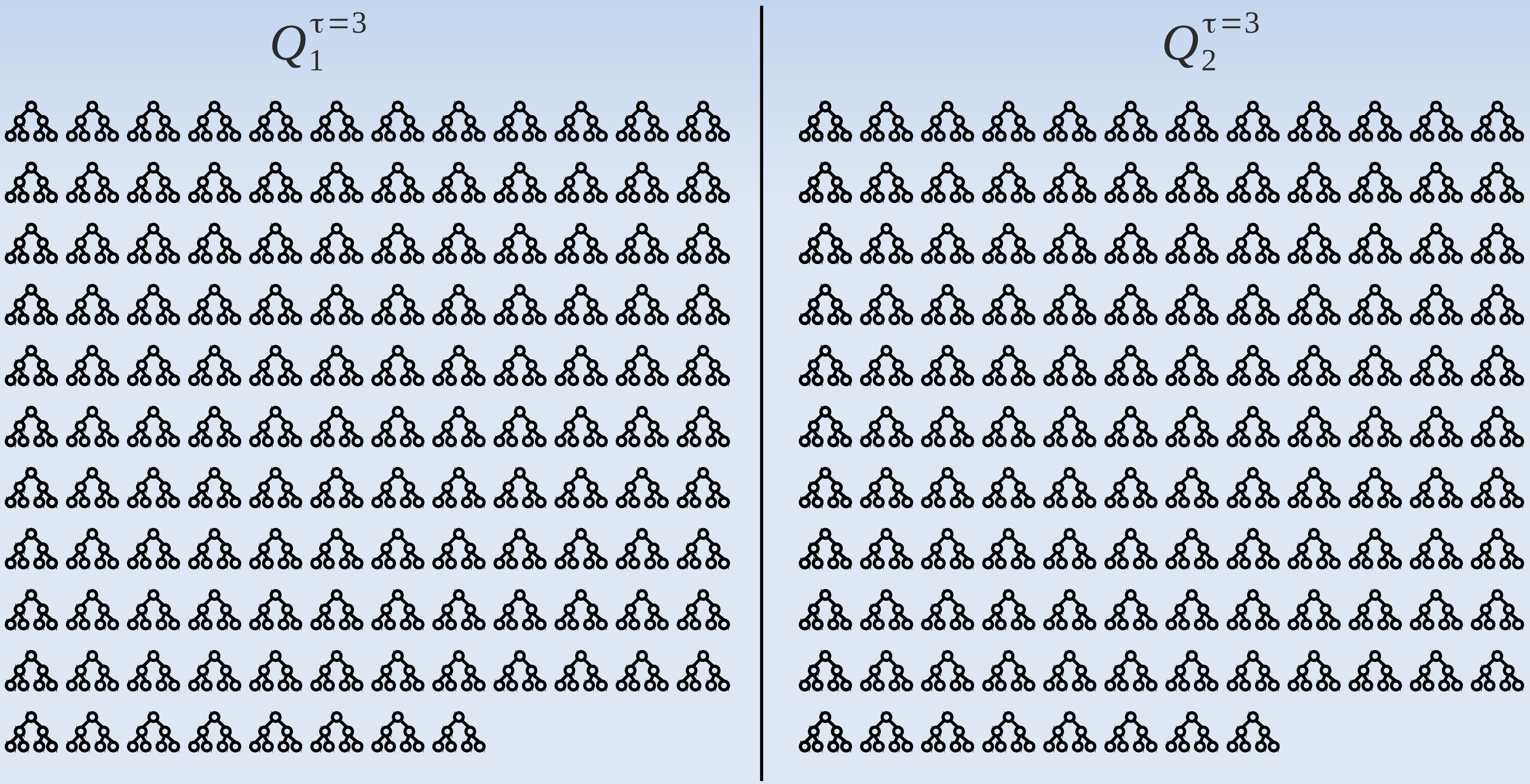
Dynamic Programming – 2

- (obviously) this scales very poorly...



Dynamic Programming – 2

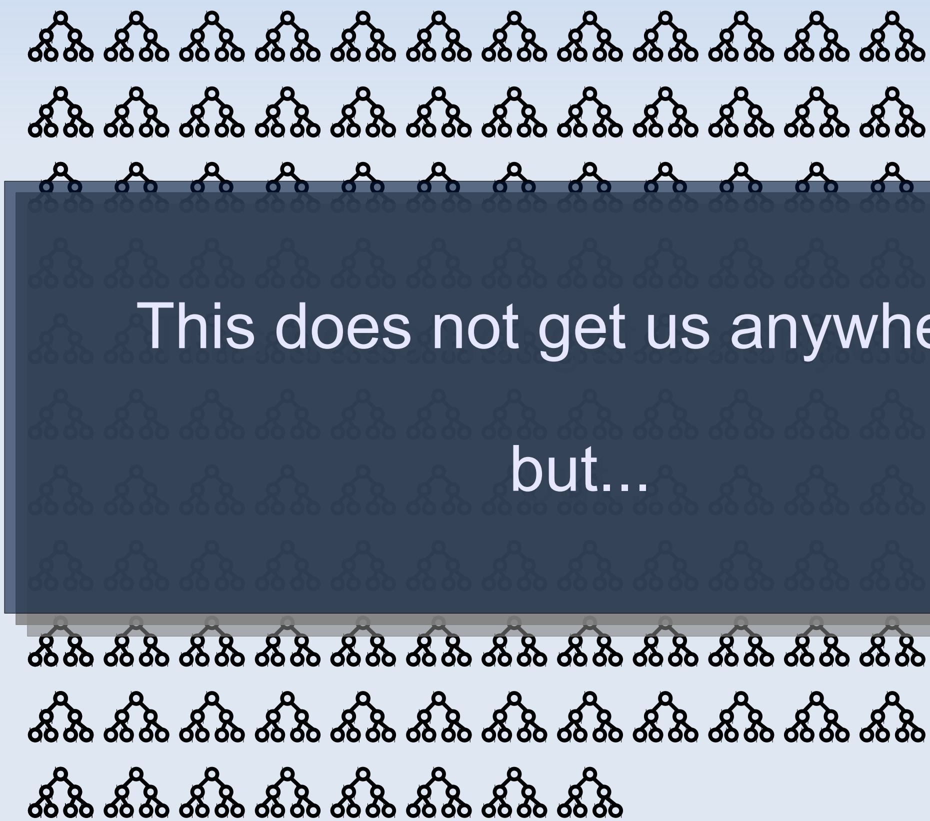
- (obviously) this scales very poorly...



Dynamic Programming – 2

- (obviously) this scales very poorly...

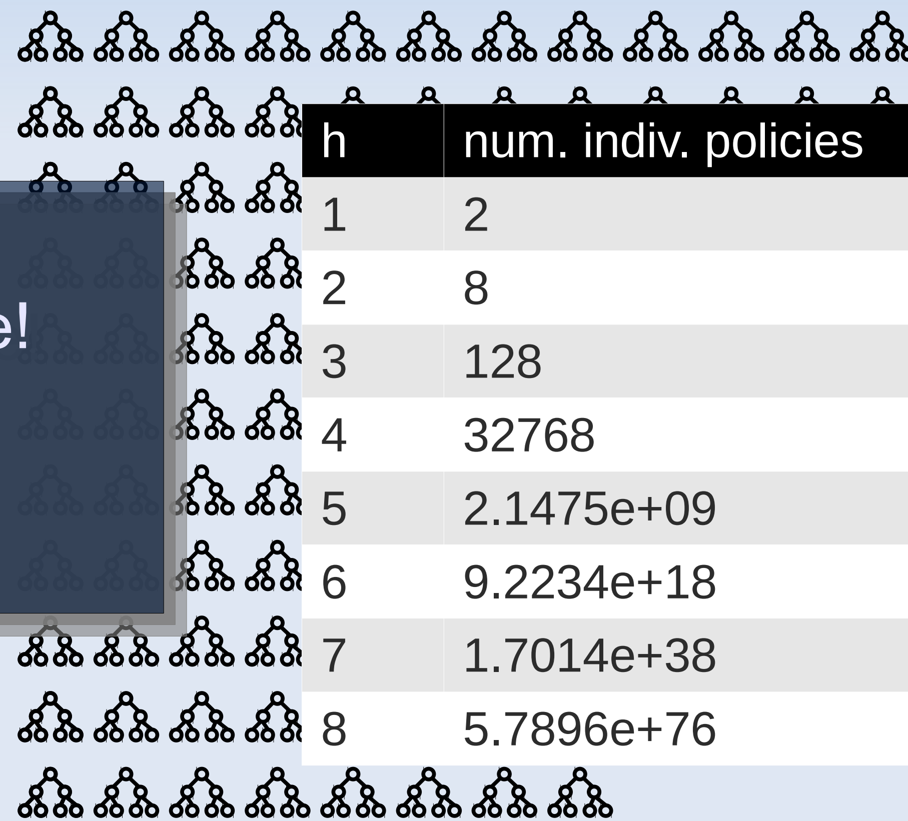
$Q_1^{\tau=3}$



This does not get us anywhere!

but...

$Q_2^{\tau=3}$



h	num. indiv. policies
1	2
2	8
3	128
4	32768
5	2.1475e+09
6	9.2234e+18
7	1.7014e+38
8	5.7896e+76

Dynamic Programming – 3

- Perhaps not all those Q_i^τ are useful!
  - Perform **pruning** of 'dominated policies'!
- Algorithm [Hansen et al. 2004]

$$Q_i^{\tau=1} = A_i$$

```
Initialize Q1(1), Q2(1)
for tau=2 to h
  Q1(tau) = ExhaustiveBackup(Q1(tau-1))
  Q2(tau) = ExhaustiveBackup(Q2(tau-1))
  Prune(Q1,Q2,tau)
end
```

Dynamic Programming – 3

- Perhaps not all those Q_i^τ are useful!
  - Perform **pruning** of 'dominated policies'!
- Algorithm [Hansen et al. 2004]

$$Q_i^{\tau=1} = A_i$$

```
Initialize Q1(1), Q2(1)
for tau=2 to h
  Q1(tau) = ExhaustiveBackup(Q1(tau-1))
  Q2(tau) = ExhaustiveBackup(Q2(tau-1))
  Prune(Q1,Q2,tau)
end
```

Note: cannot prune independently!

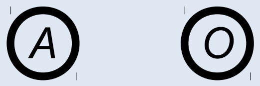
- usefulness of a q_1 depends on Q_2
- and vice versa

→ **Iterated elimination** of policies

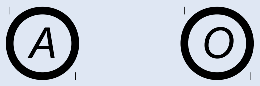
Dynamic Programming – 4

- Initialization

$Q_1^{\tau=1}$



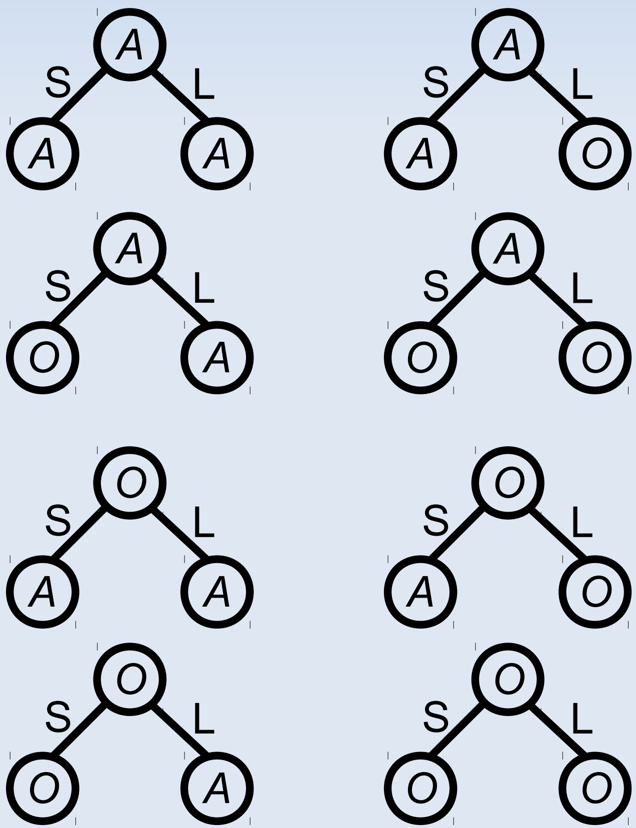
$Q_2^{\tau=1}$



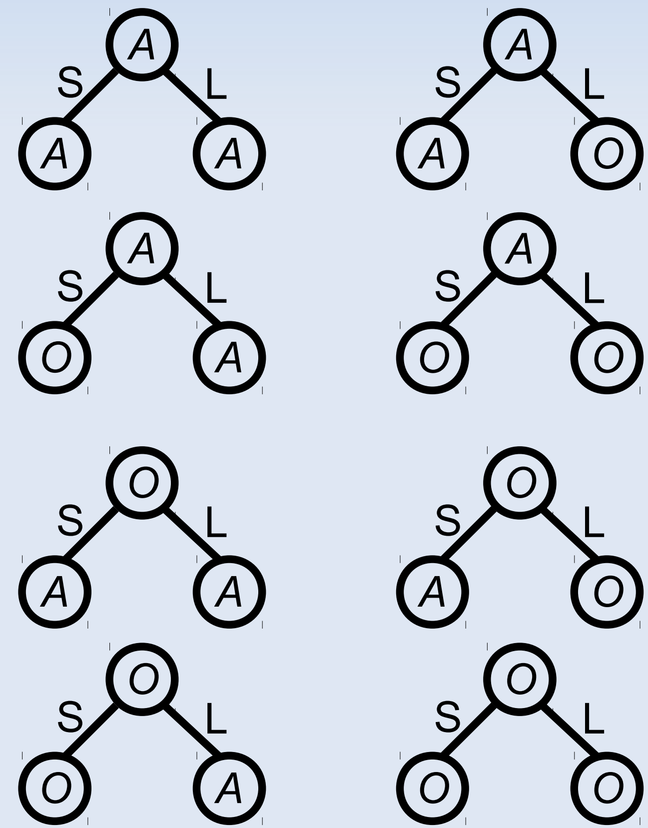
Dynamic Programming – 4

- Exhaustive Backups gives

$Q_1^{\tau=2}$



$Q_2^{\tau=2}$

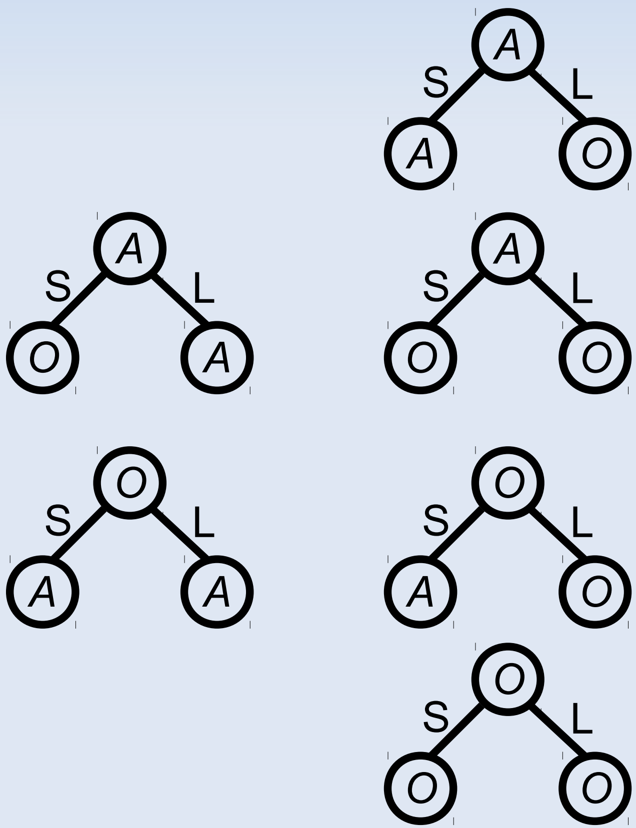


Dynamic Programming – 4

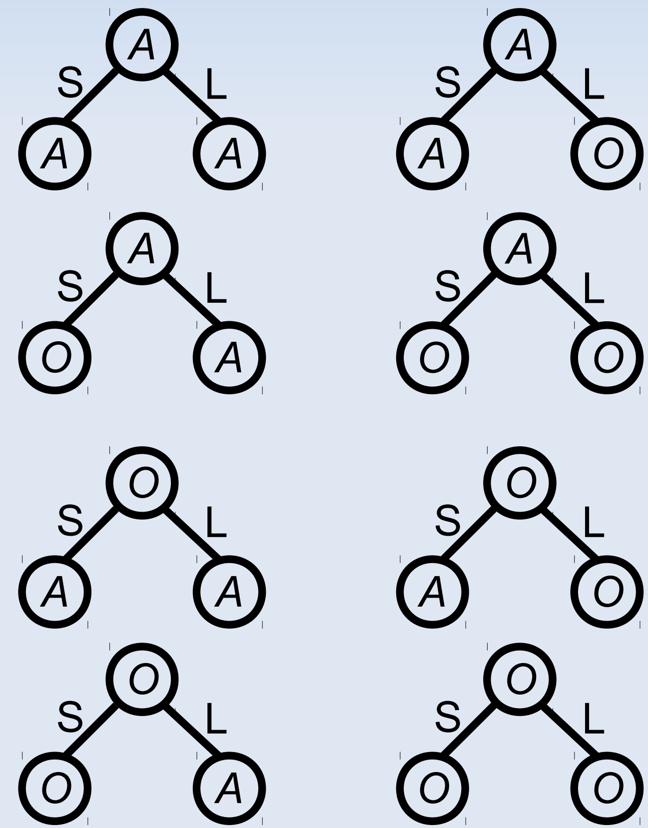
Hypothetical Pruning
(not the result of actual pruning)

- Pruning agent 1...

$Q_1^{\tau=2}$

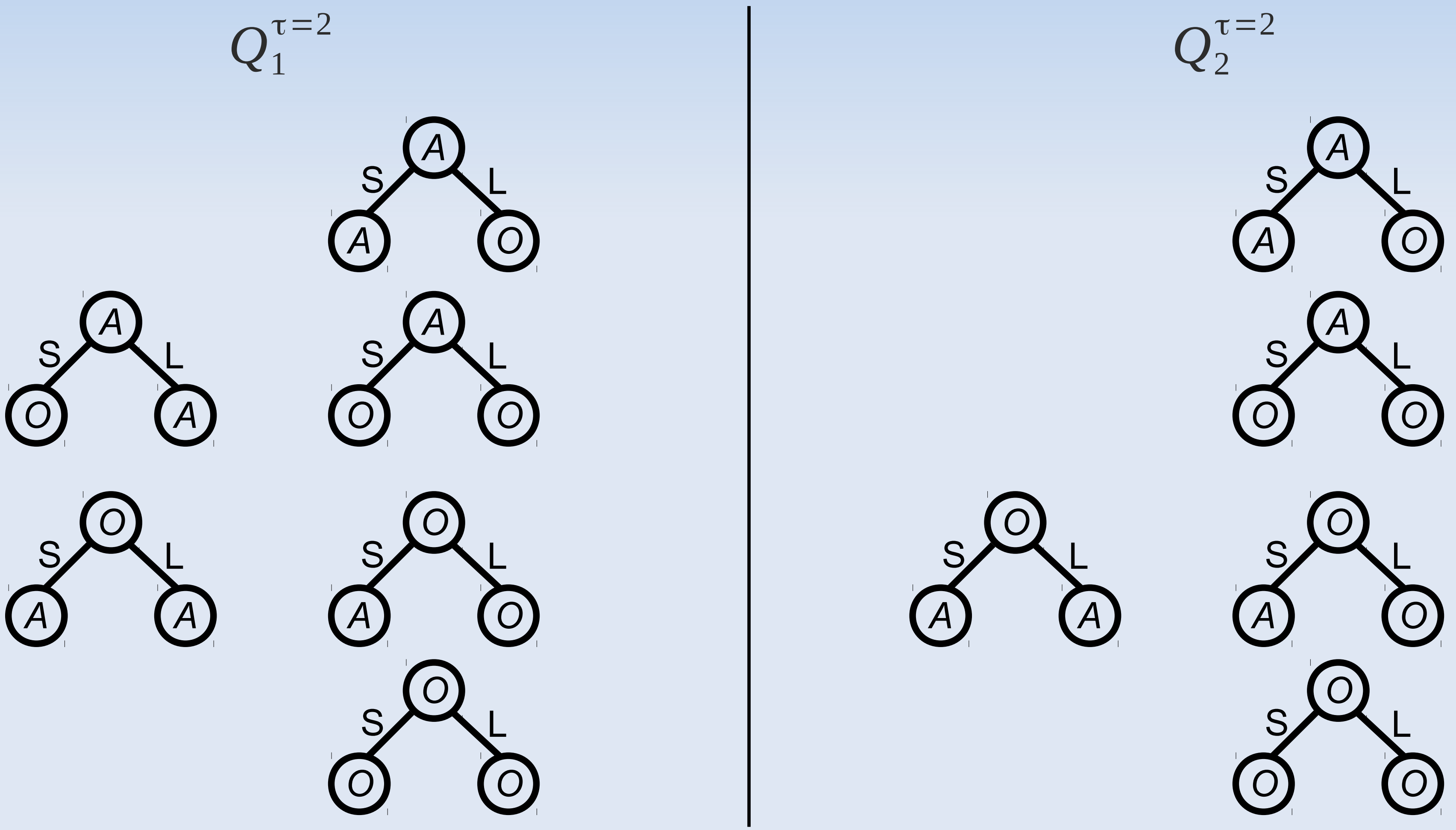


$Q_2^{\tau=2}$



Dynamic Programming – 4

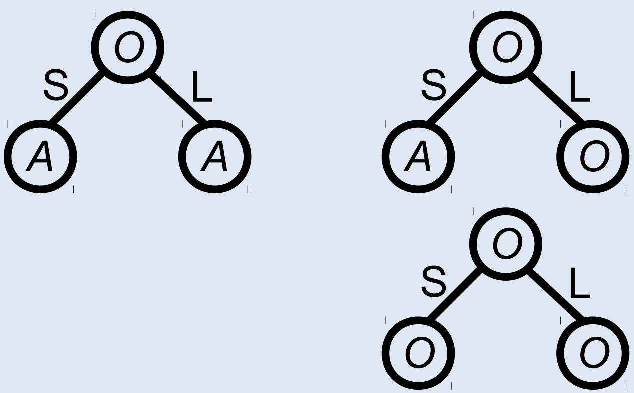
- Pruning agent 2...



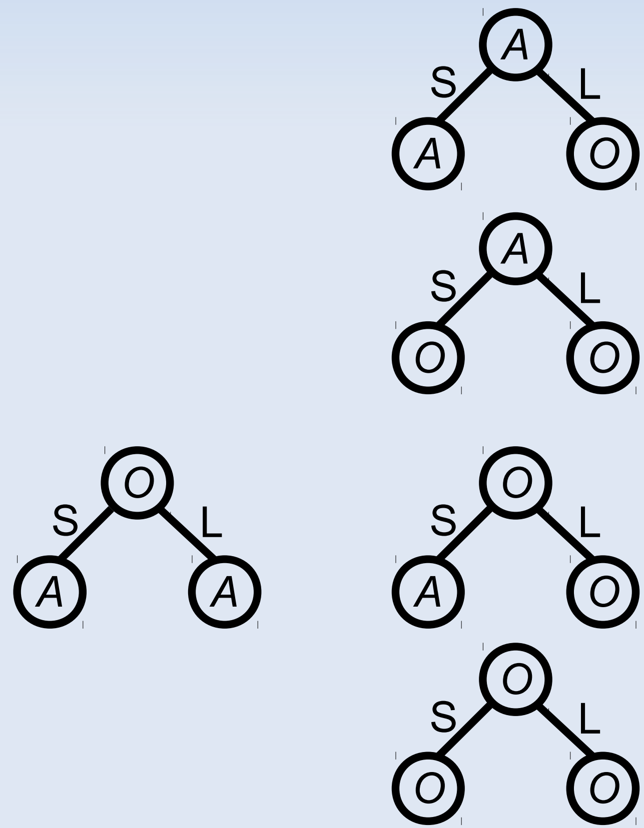
Dynamic Programming – 4

- Pruning agent 1...

$Q_1^{\tau=2}$



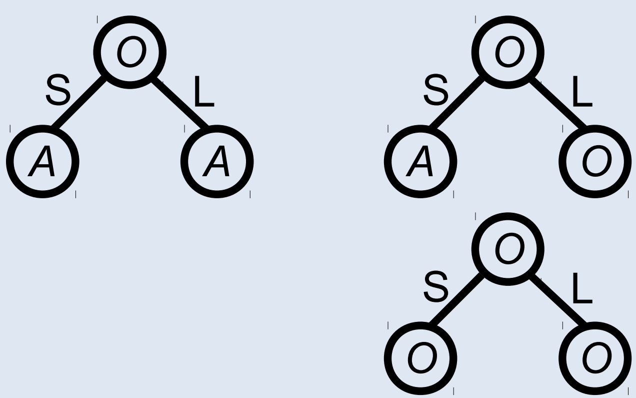
$Q_2^{\tau=2}$



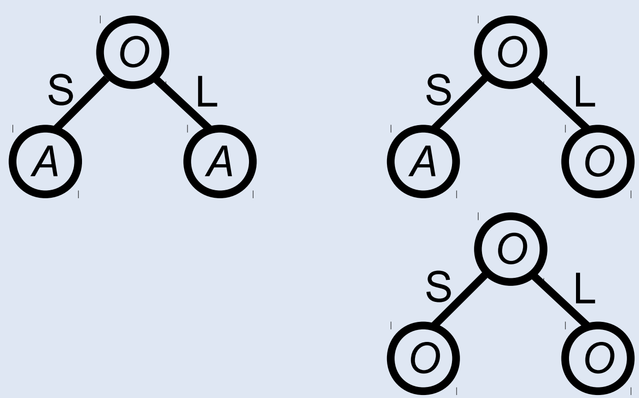
Dynamic Programming – 4

- Etc...

$Q_1^{\tau=2}$



$Q_2^{\tau=2}$



Dynamic Programming – 4

- Etc...

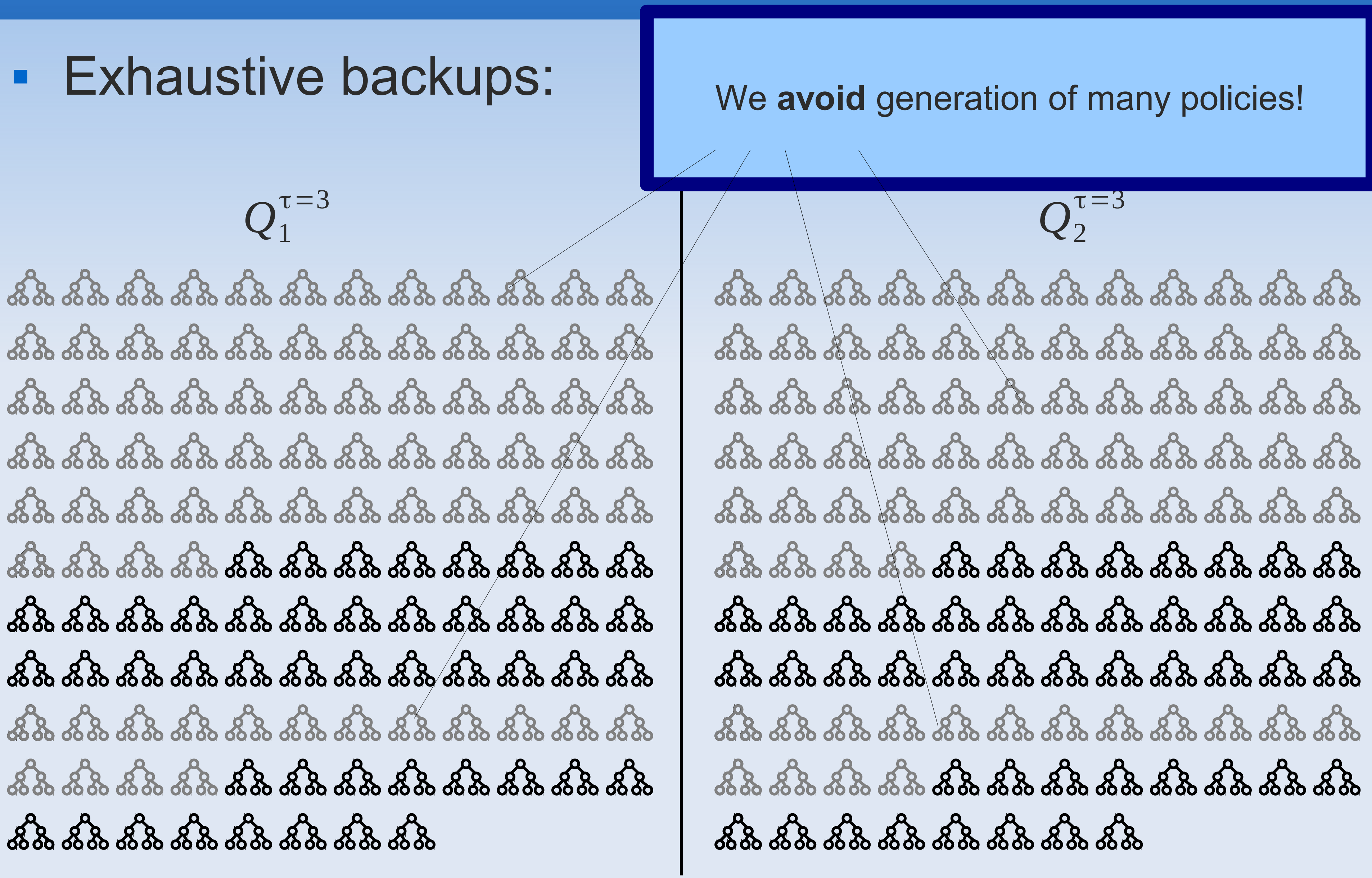
$Q_1^{\tau=2}$

$Q_2^{\tau=2}$

In this case: symmetric
→ but need not be in general!

Dynamic Programming – 4

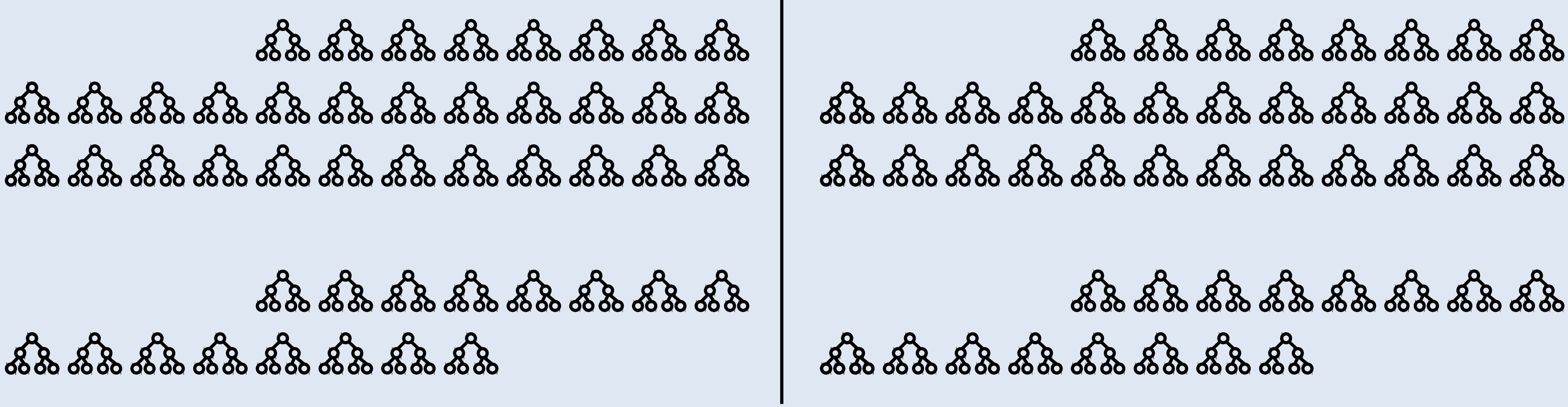
- Exhaustive backups:



Dynamic Programming – 4

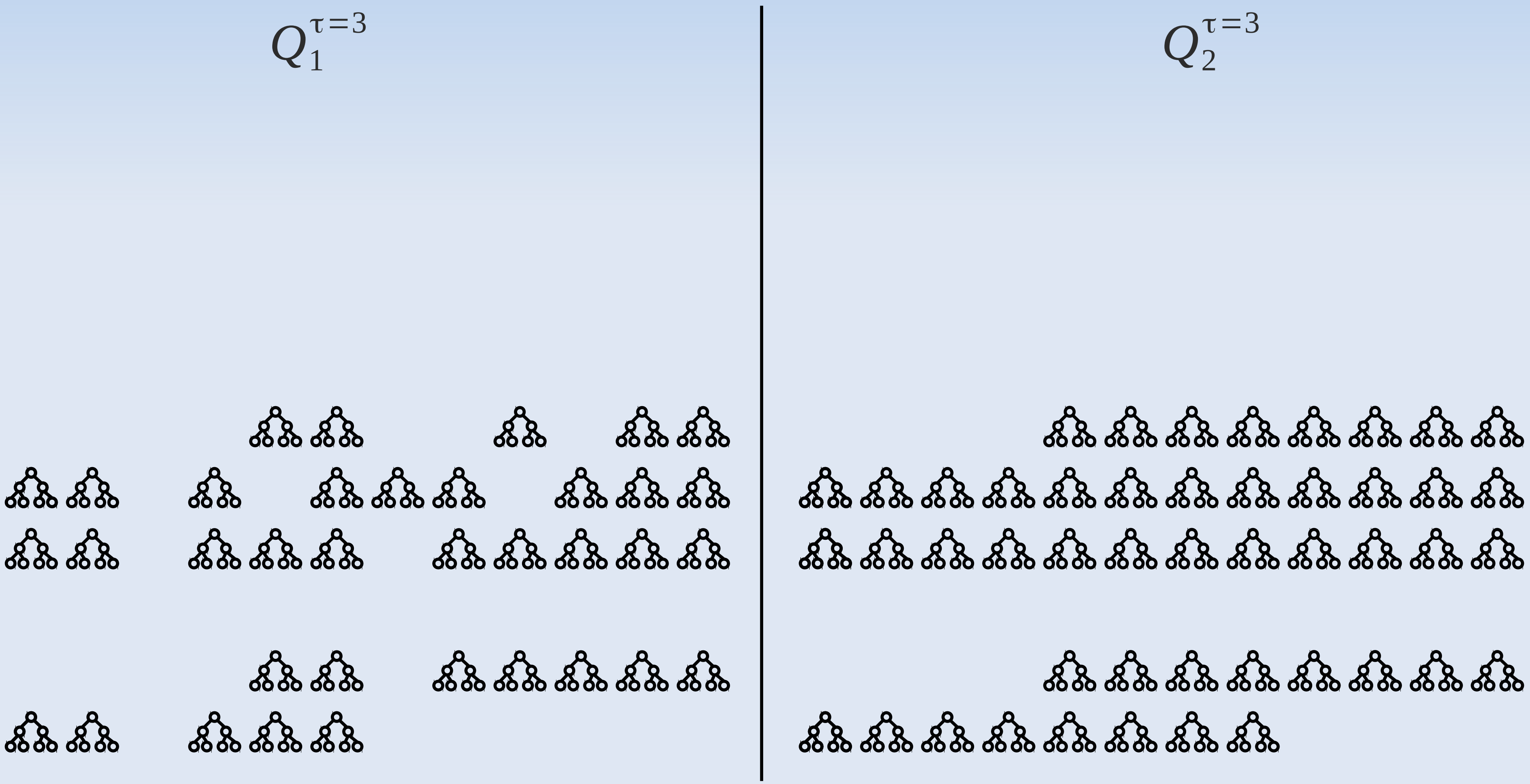
- Exhaustive backups:

$Q_1^{\tau=3}$ $Q_2^{\tau=3}$



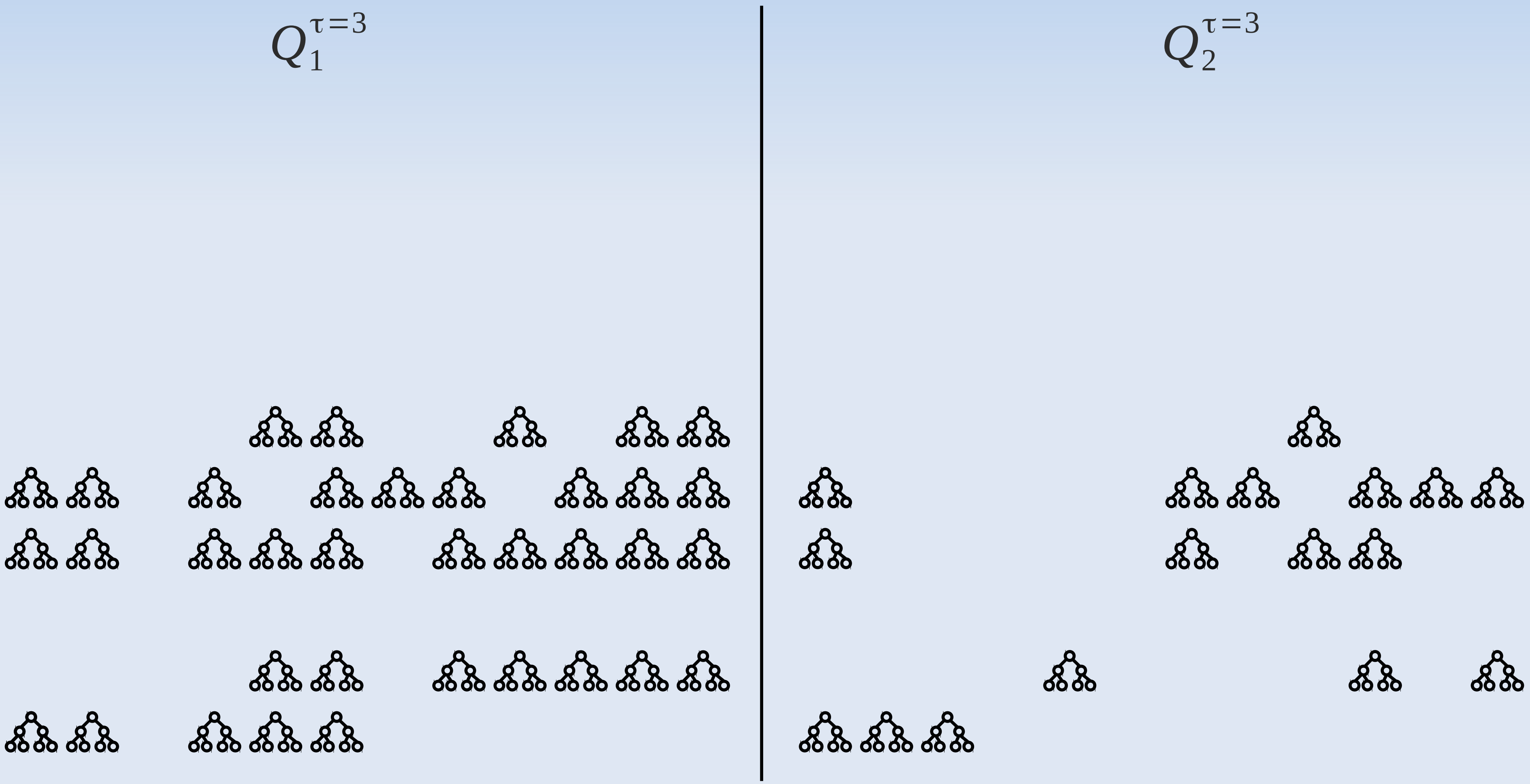
Dynamic Programming – 4

- Pruning agent 1...



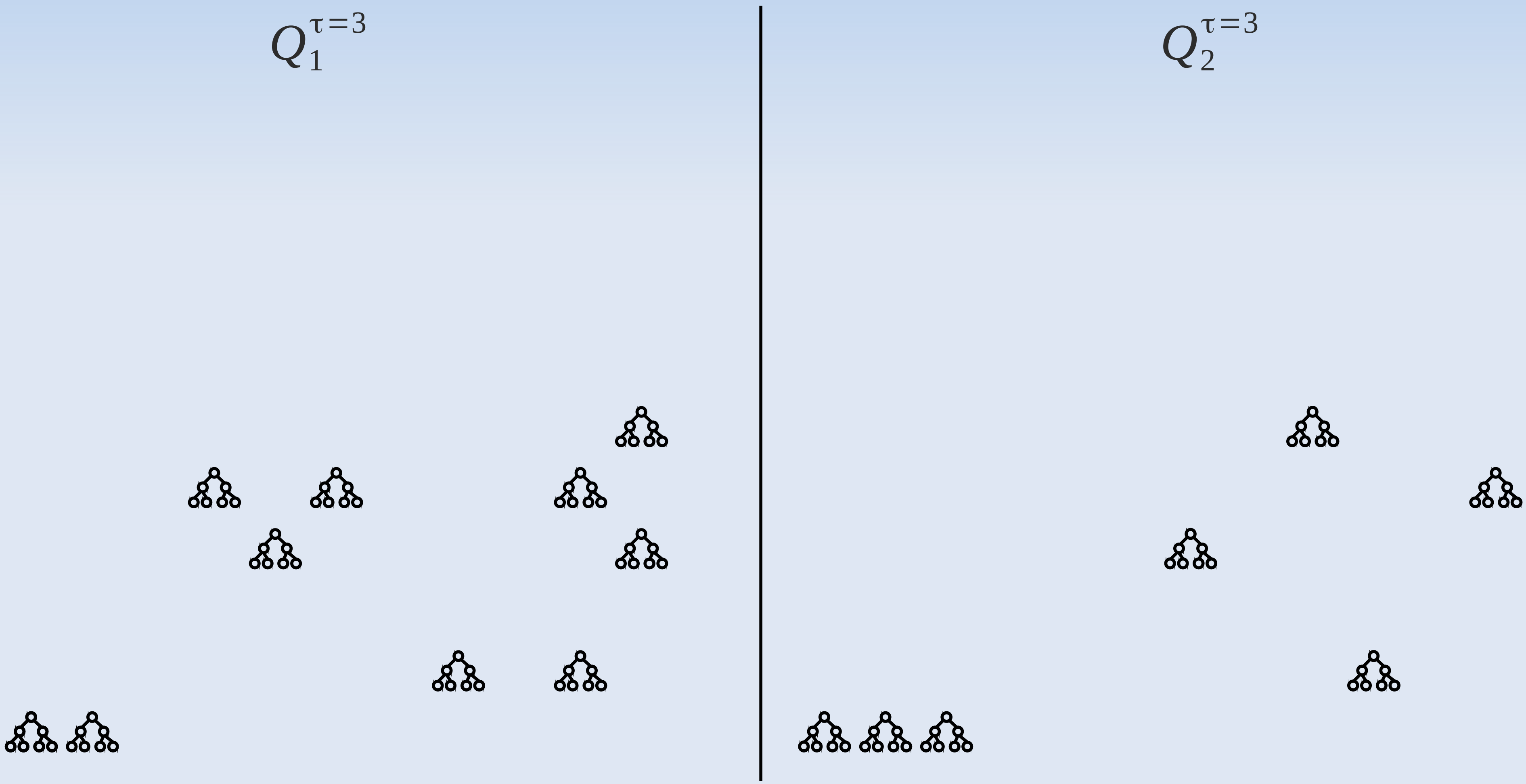
Dynamic Programming – 4

- Pruning agent 2...



Dynamic Programming – 4

- Etc...

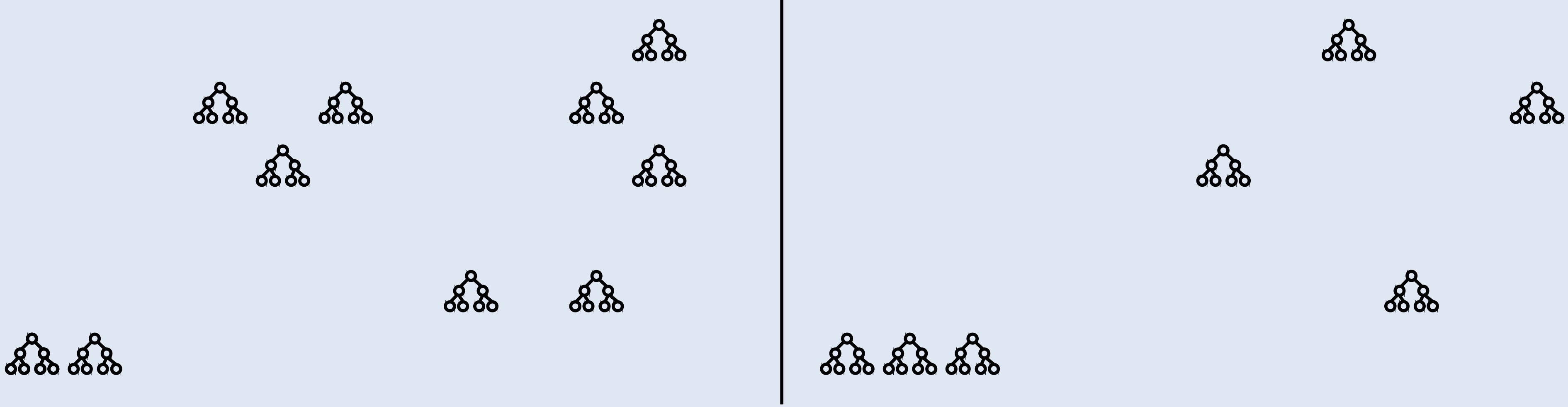


Dynamic Programming – 4

- Etc...

At the very end:

- ...?

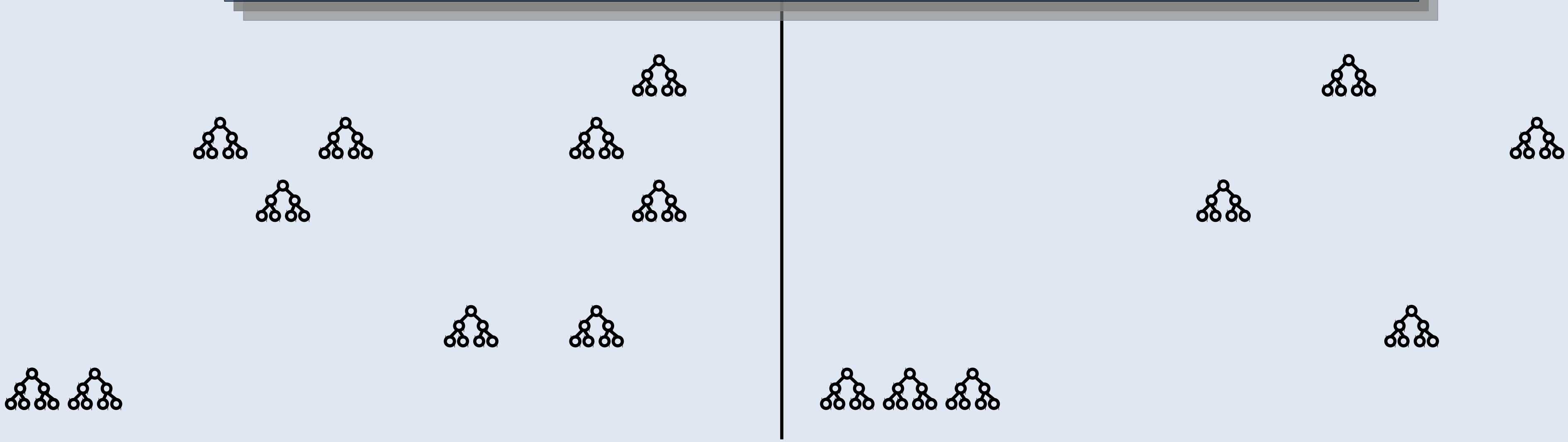


Dynamic Programming – 4

- Etc...

At the very end:

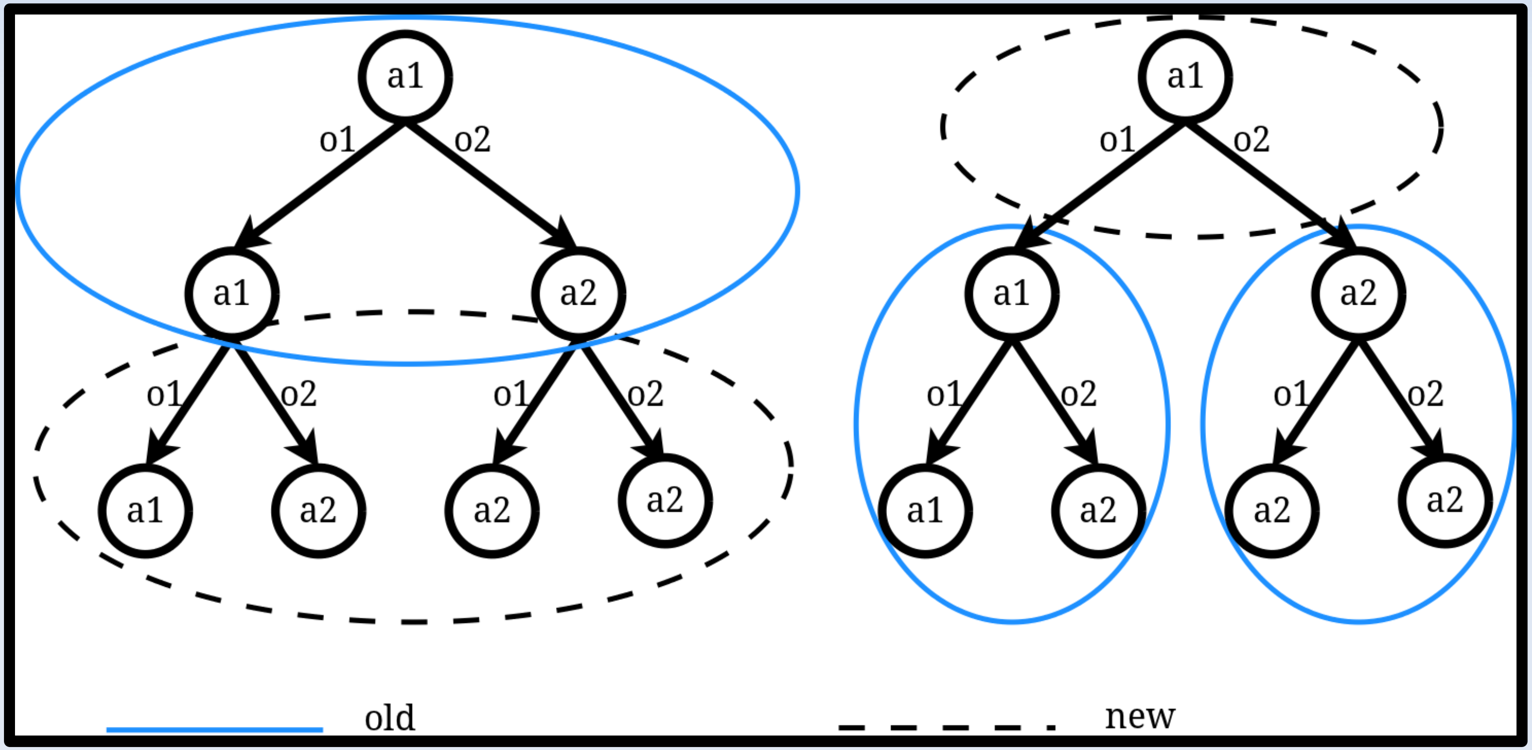
- evaluate all the remaining combinations of policies (i.e., the 'induced joint policies')
- select the best one



Bottom-up vs. Top-down

- DP constructs bottom-up
- Alternatively try and construct top down

  → leads to (heuristic) search [Szer et al. 2005, Oliehoek et al. 2008]

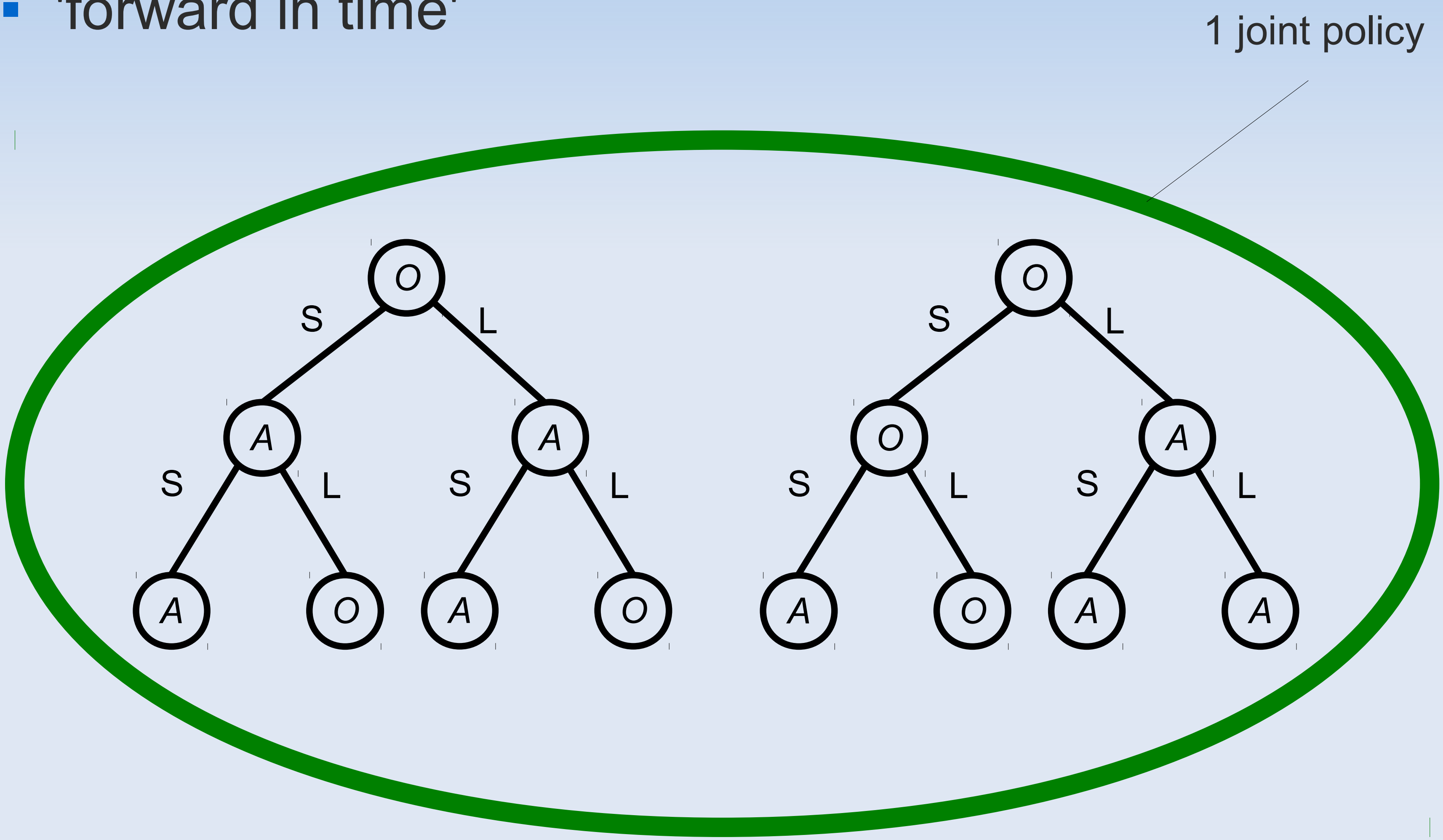


Heuristic Search – Intro

- Core idea is the same as DP:
  - incrementally construct all (joint) policies
  - try to avoid work
- Differences
  - different starting point and increments
  - use **heuristics** (rather than pruning) to avoid work

Heuristic Search – 1

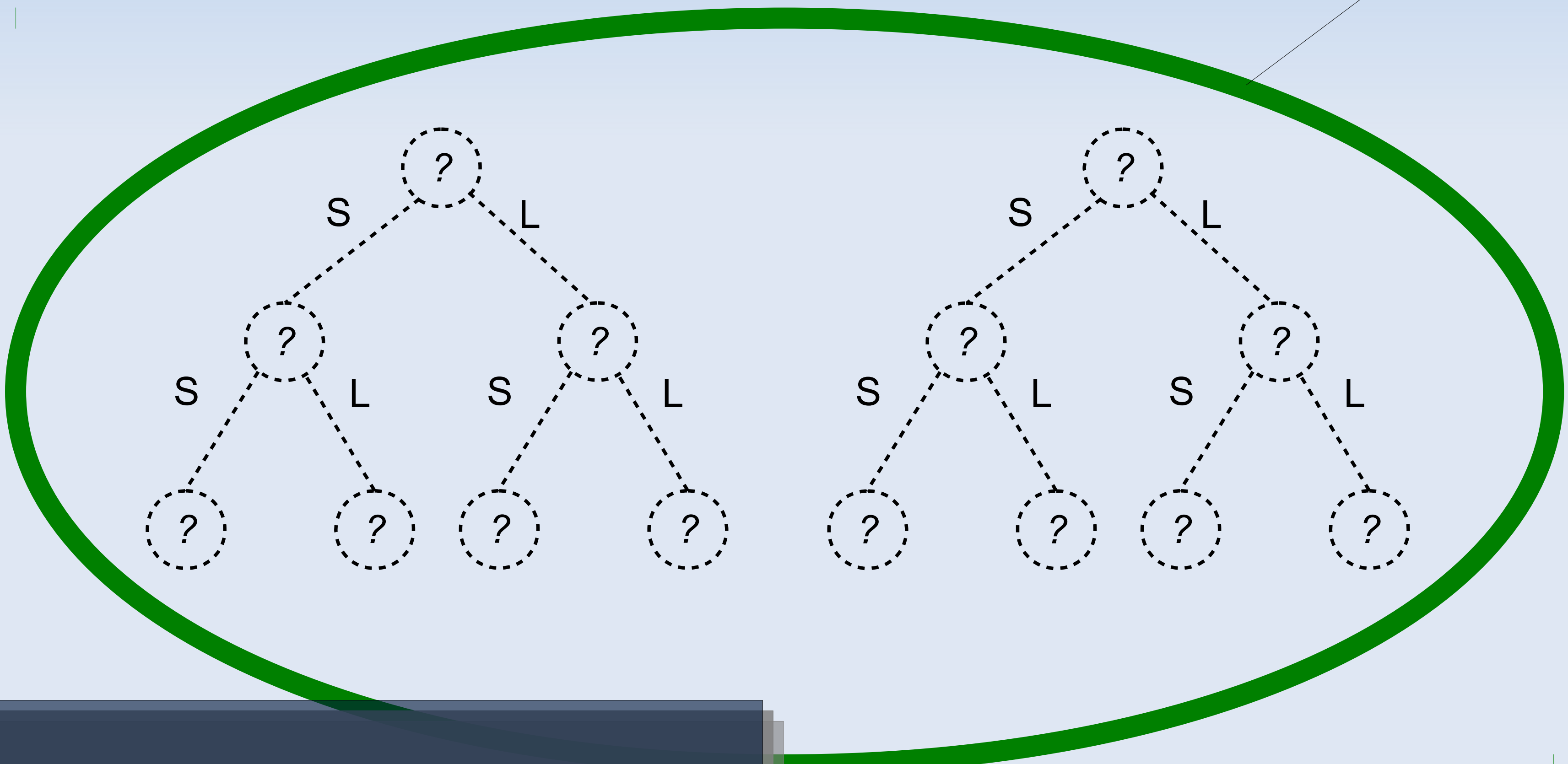
- Incrementally construct all (joint) policies
  - 'forward in time'



Heuristic Search – 1

- Incrementally construct all (joint) policies
  - 'forward in time'

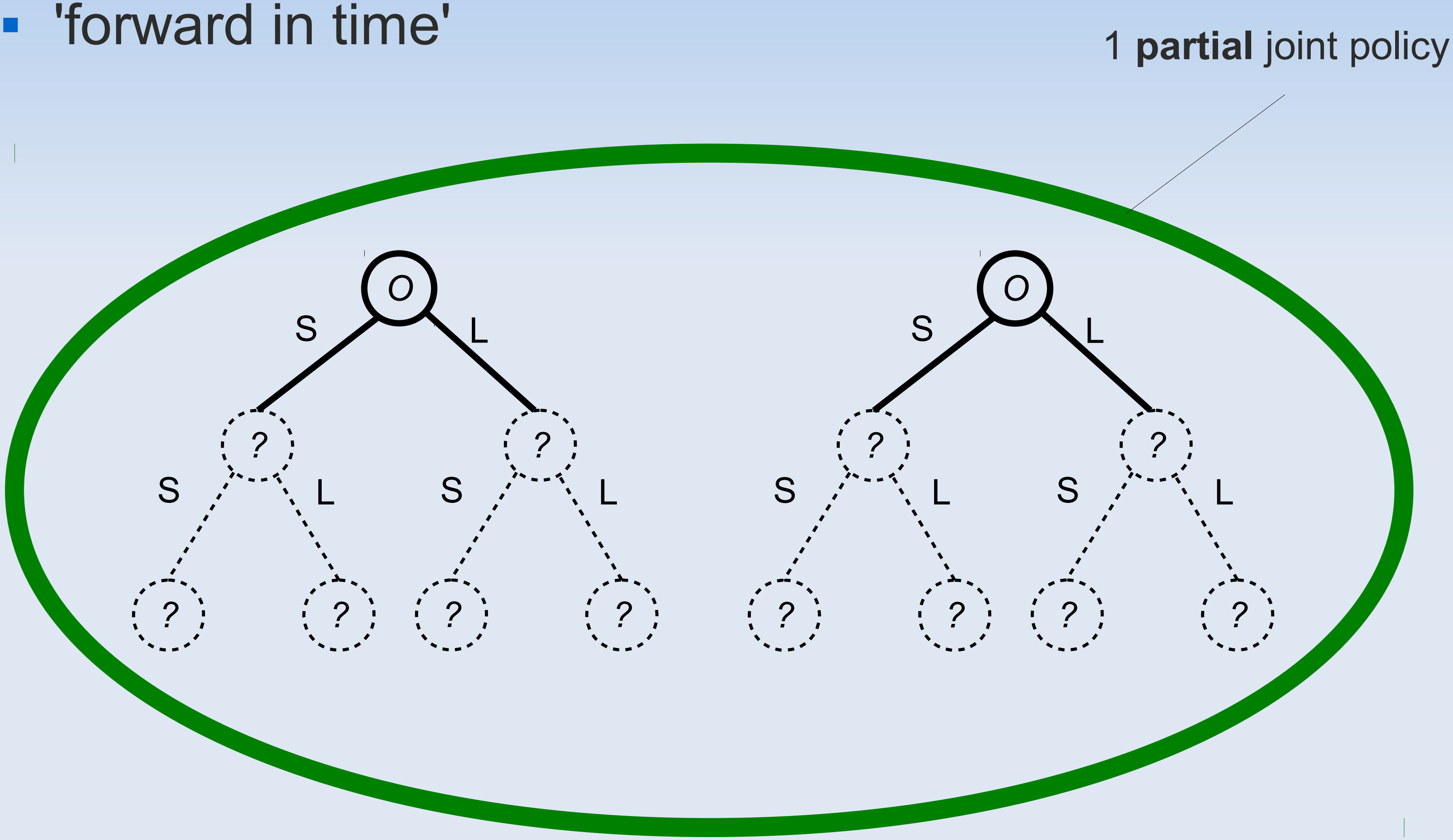
1 **partial** joint policy



Start with unspecified policy

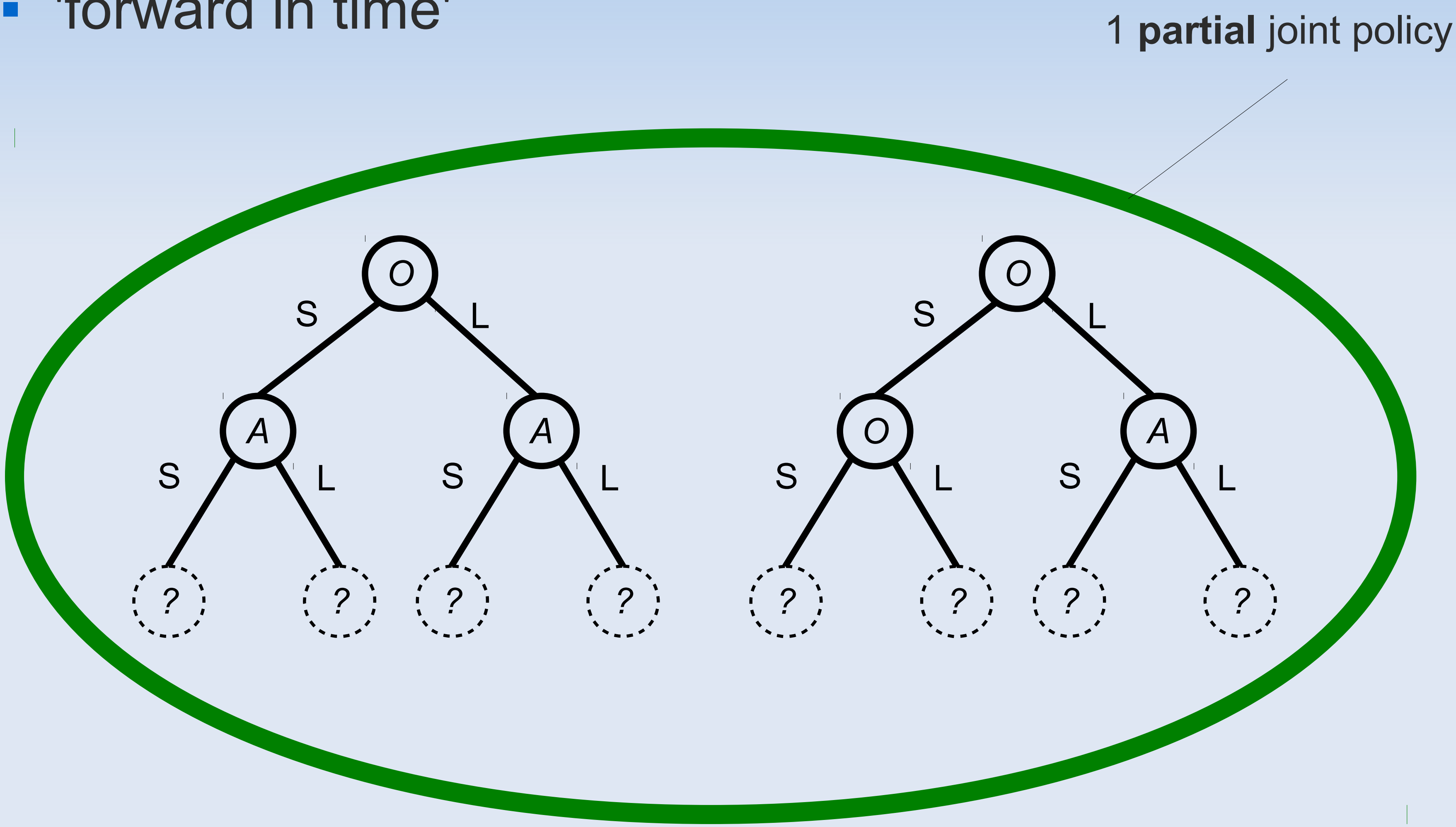
Heuristic Search – 1

- Incrementally construct all (joint) policies
  - 'forward in time'



Heuristic Search – 1

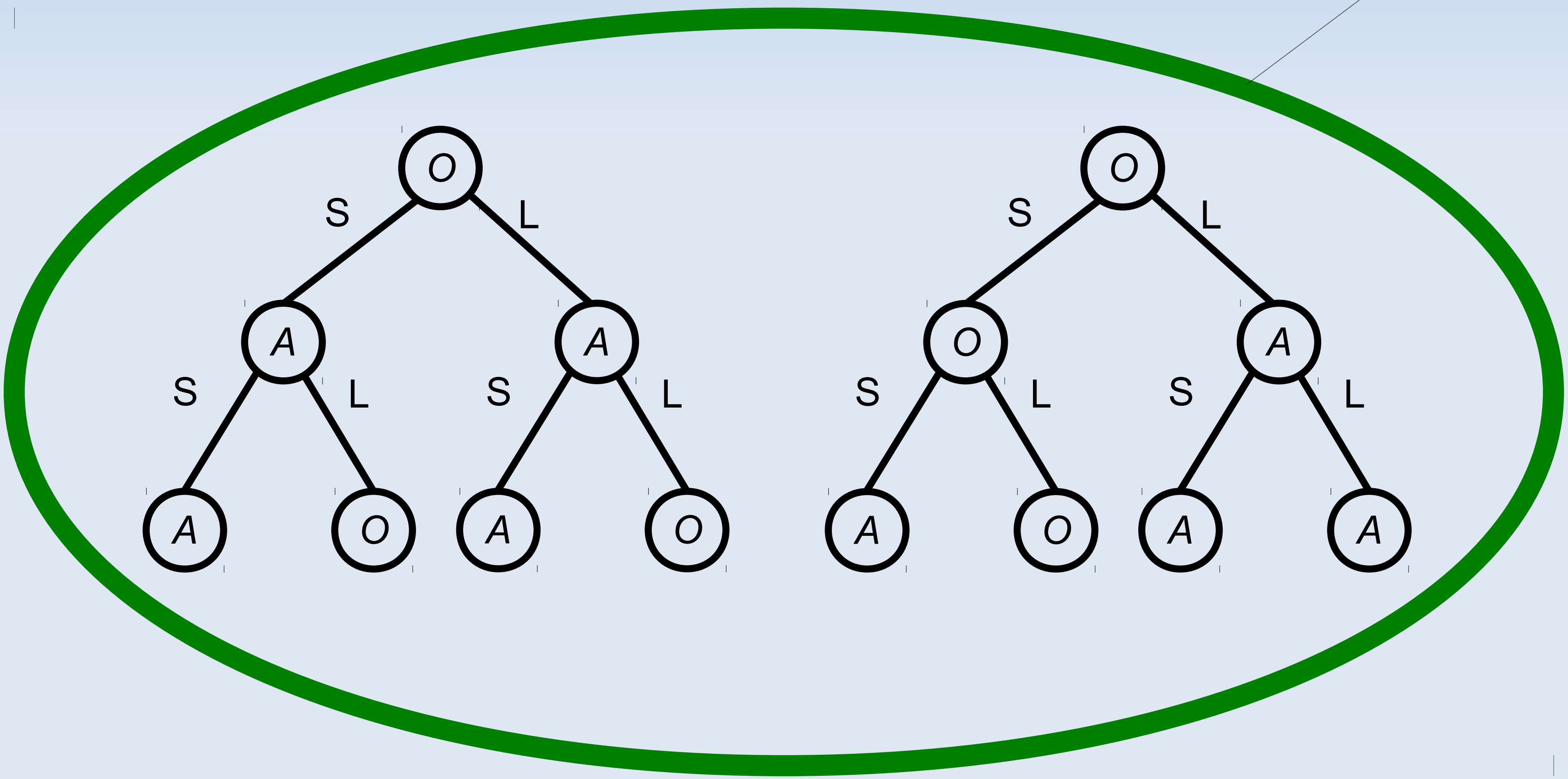
- Incrementally construct all (joint) policies
  - 'forward in time'



Heuristic Search – 1

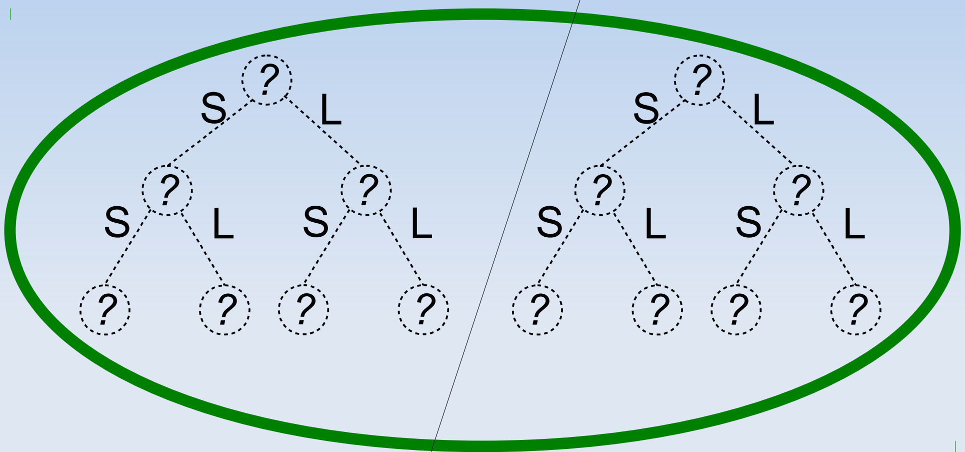
- Incrementally construct all (joint) policies
  - 'forward in time'

1 **complete** joint policy (full-length)



Heuristic Search – 2

- Creating **ALL** joint policies → tree structure!

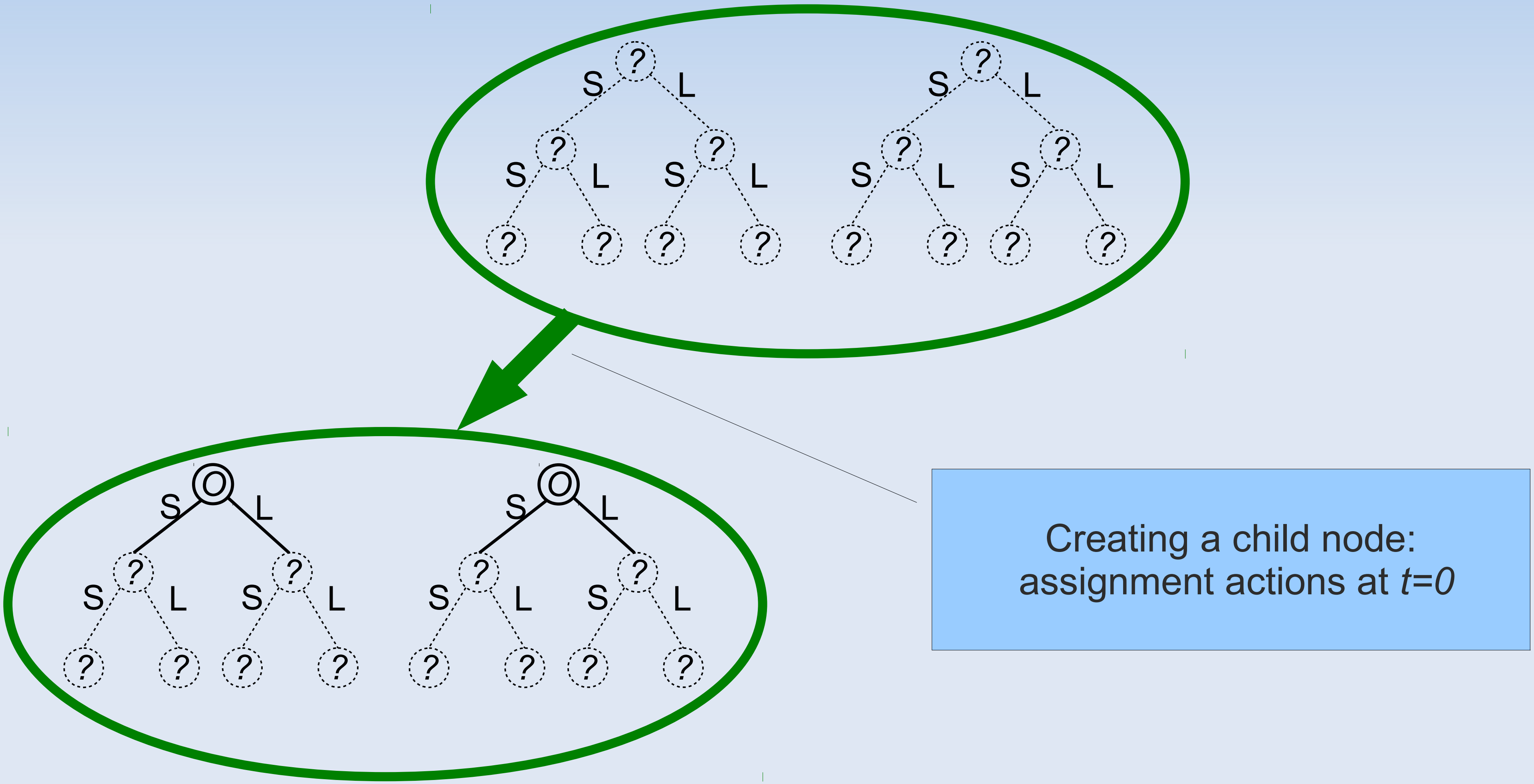


Root node:
unspecified joint policy

why?

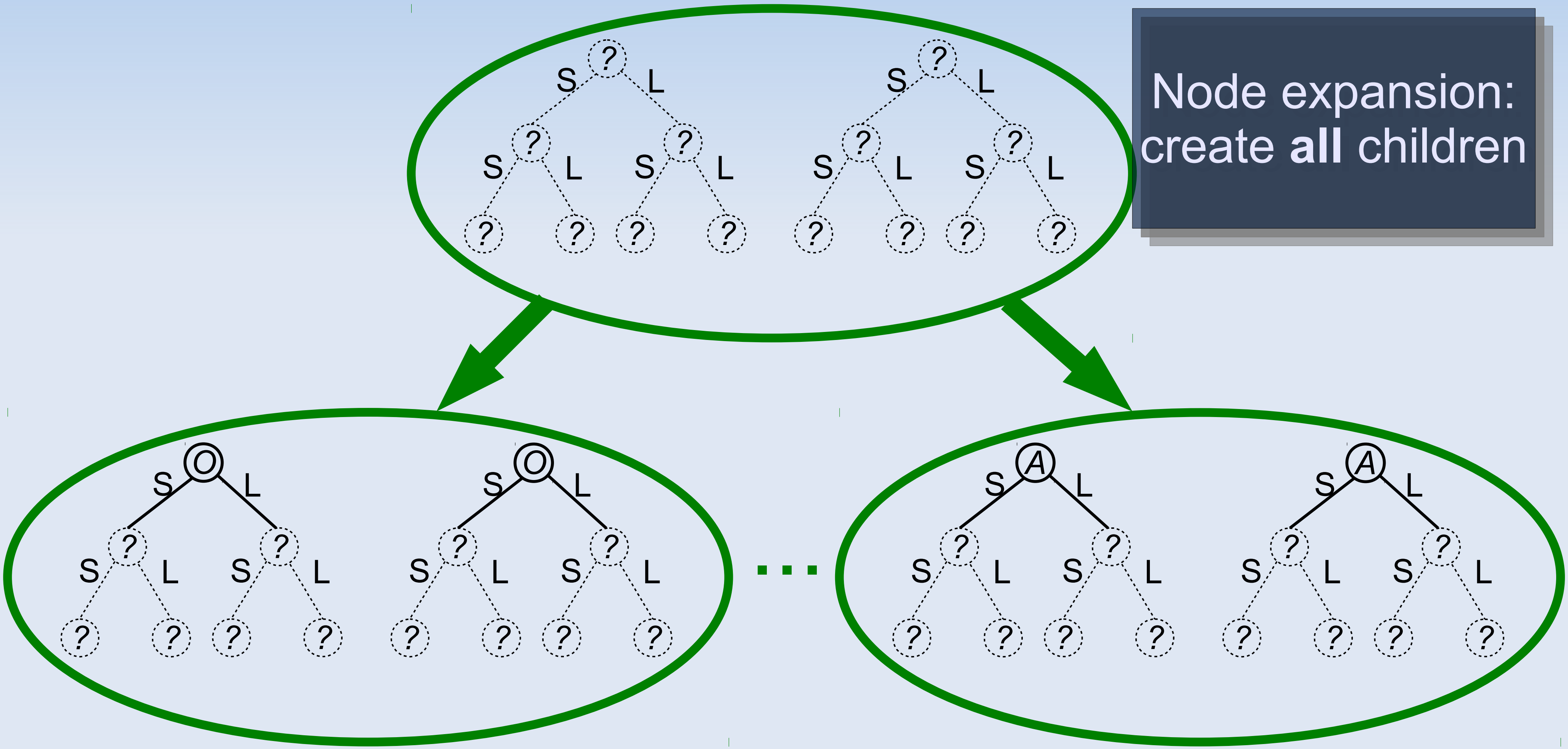
Heuristic Search – 2

- Creating **ALL** joint policies → tree structure!



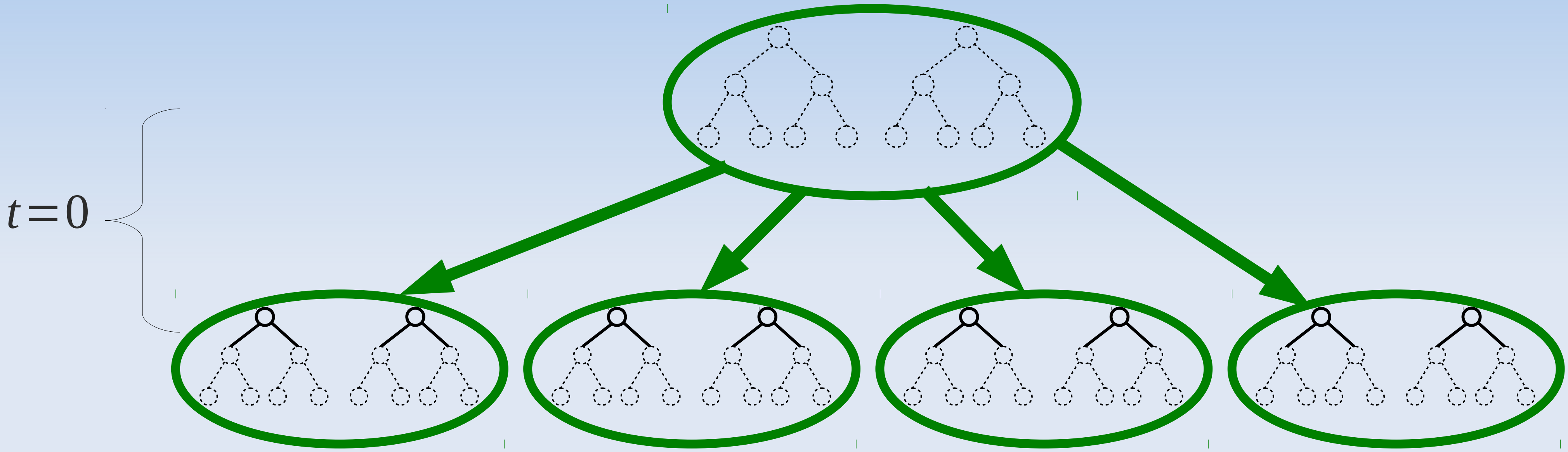
Heuristic Search – 2

- Creating **ALL** joint policies → tree structure!



Heuristic Search – 2

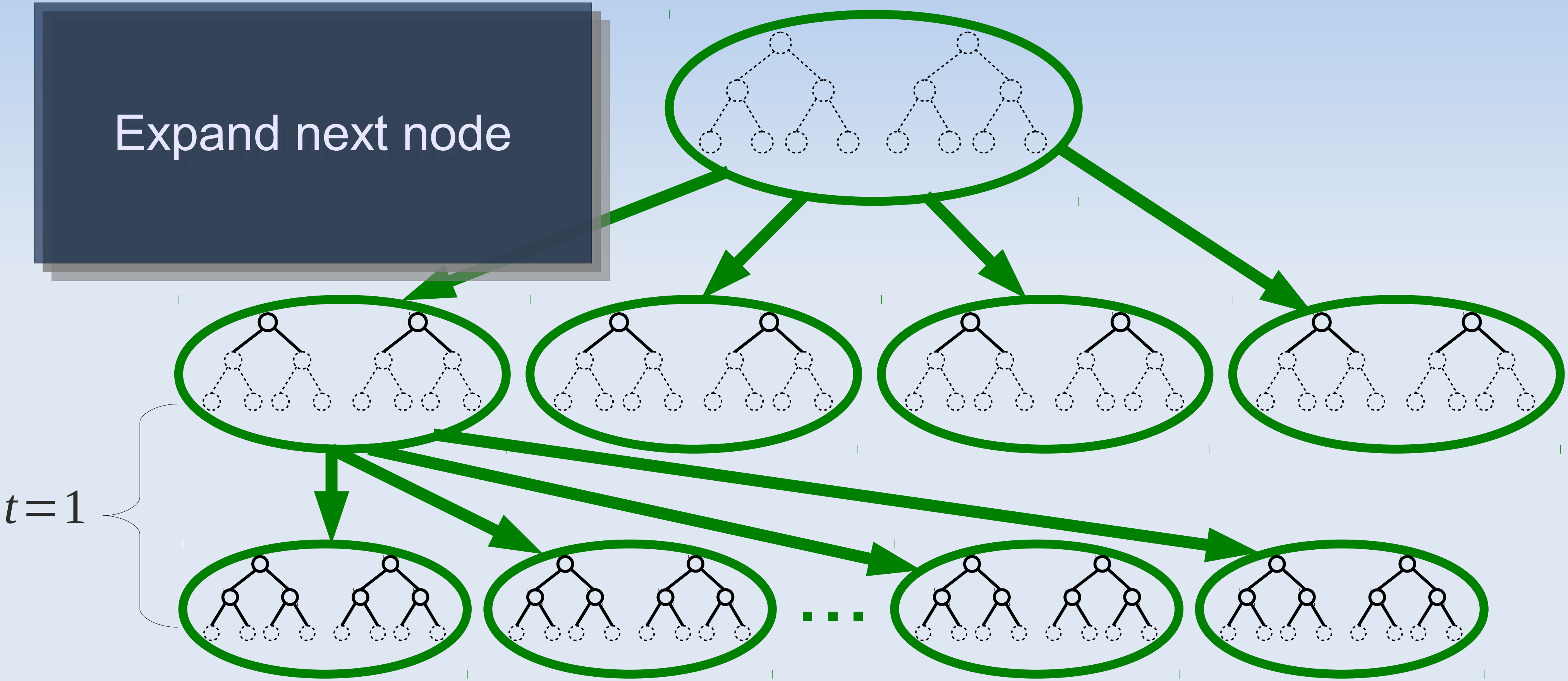
- Creating **ALL** joint policies → tree structure!



Heuristic Search – 2

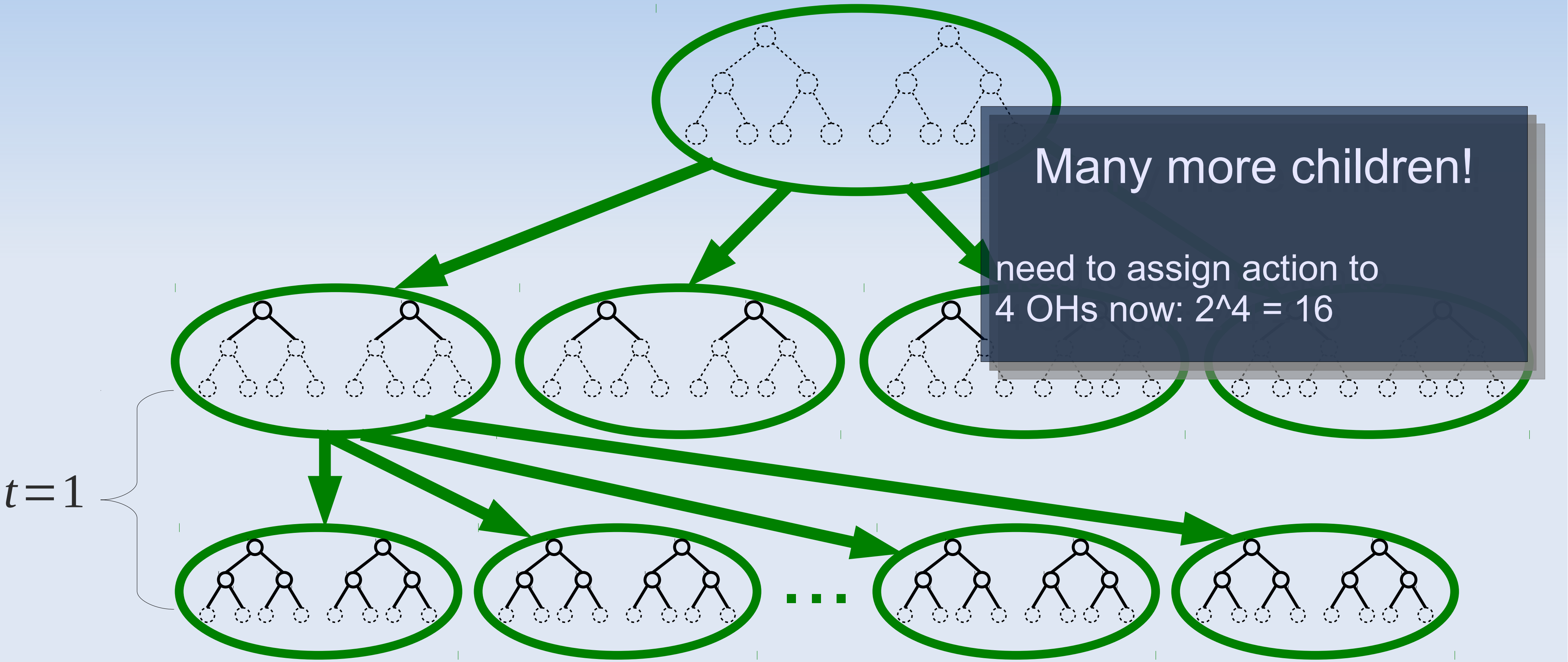
- Creating **ALL** joint policies → tree structure!

Expand next node



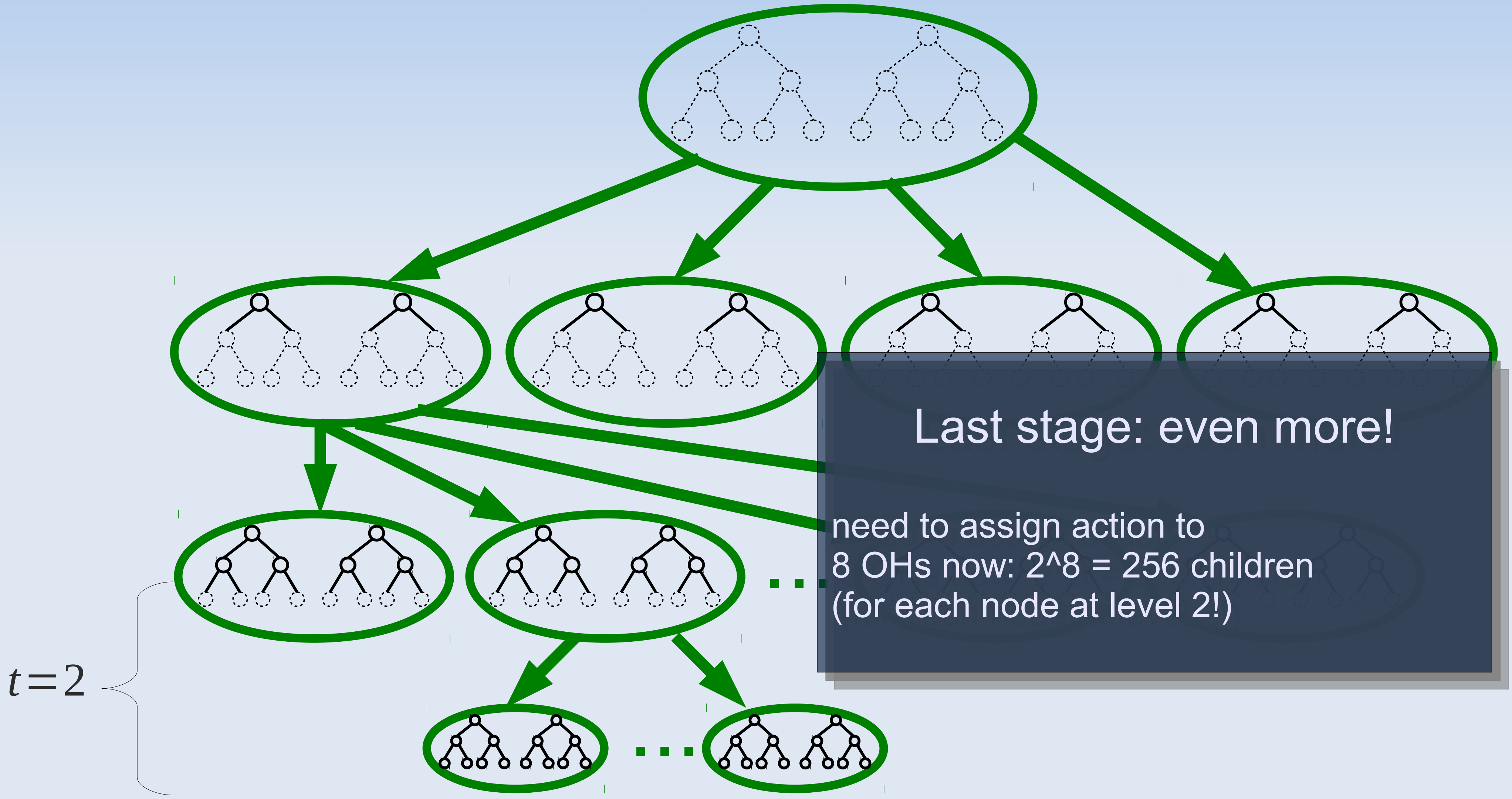
Heuristic Search – 2

- Creating **ALL** joint policies → tree structure!



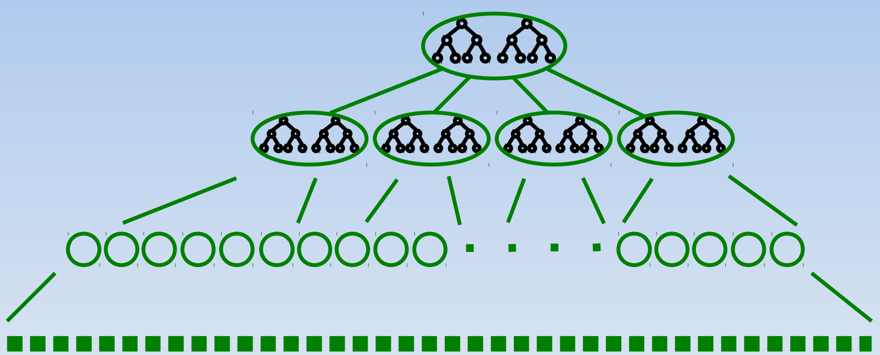
Heuristic Search – 2

- Creating **ALL** joint policies → tree structure!



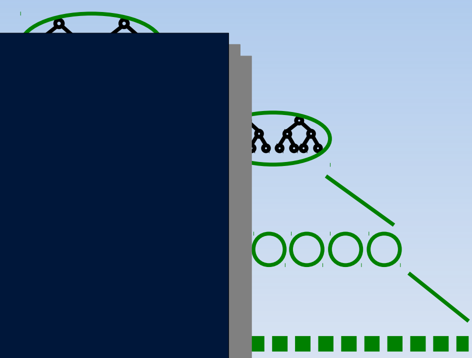
Heuristic Search – 3

- too big to create completely...
- Idea: use **heuristics**
  - avoid going down non-promising branches!
- Apply A* → **Multiagent A*** [Szer et al. 2005]

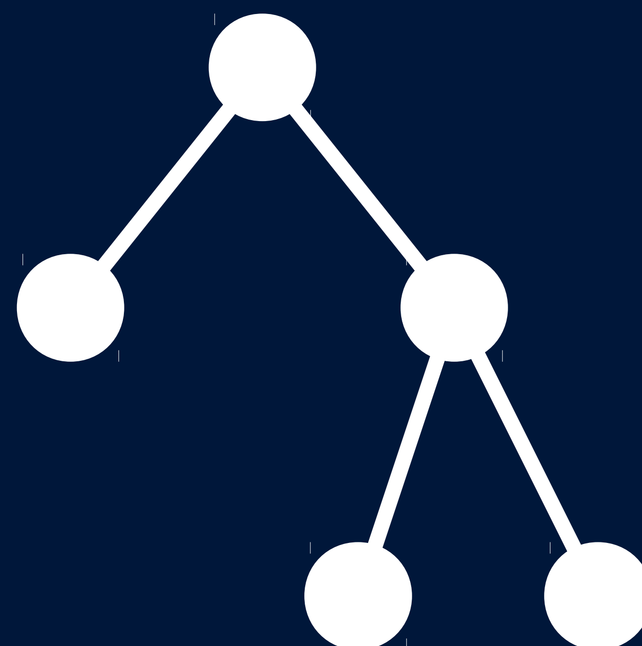


Heuristic Search – 3

- too big to create completely
- Idea:
  - avo…
    nor…
- Apply…



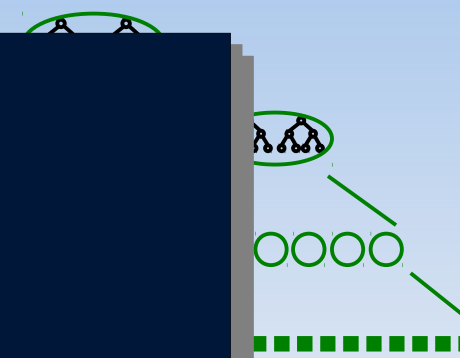
Main intuition A*



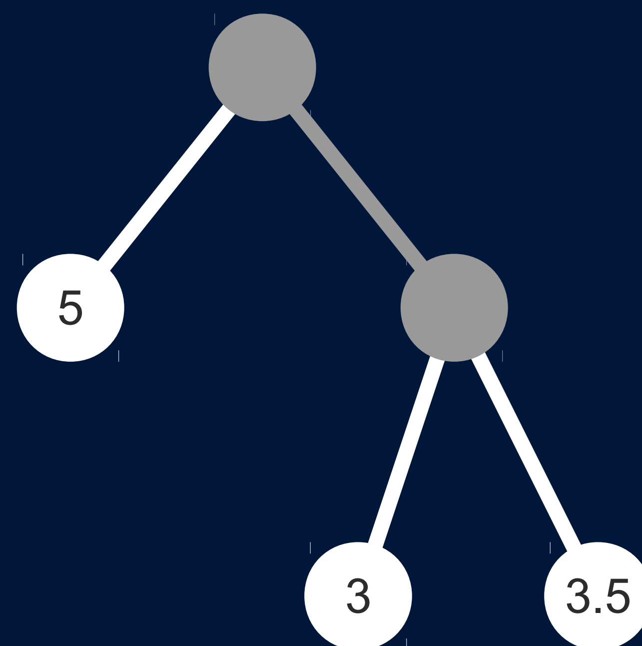
- For each node, compute F-value
- Select next node based on F-value
- More info: [Russel&Norvig 2003]

Heuristic Search – 3

- too big to create completely
- Idea:
  - avo
    nor
- Apply



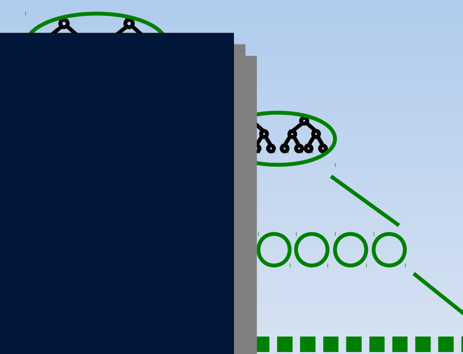
Main intuition A*



- For each node, compute F-value
- Select next node based on F-value
- More info: [Russel&Norvig 2003]

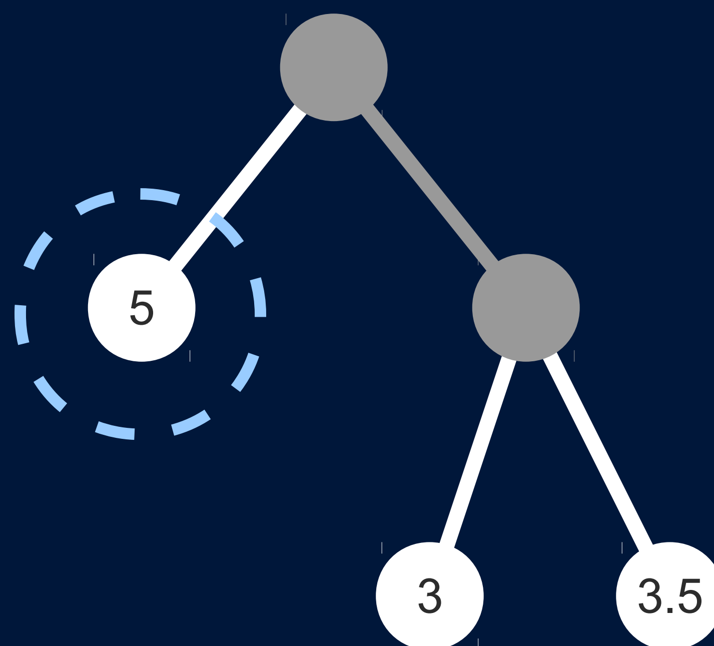
Heuristic Search – 3

- too big to create completely
- Idea:
  - avo
    nor
- Apply



Main intuition A*

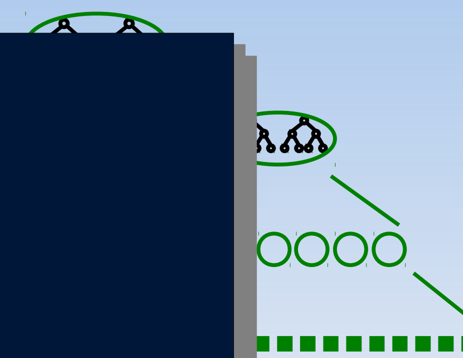
Select highest valued node & expand...



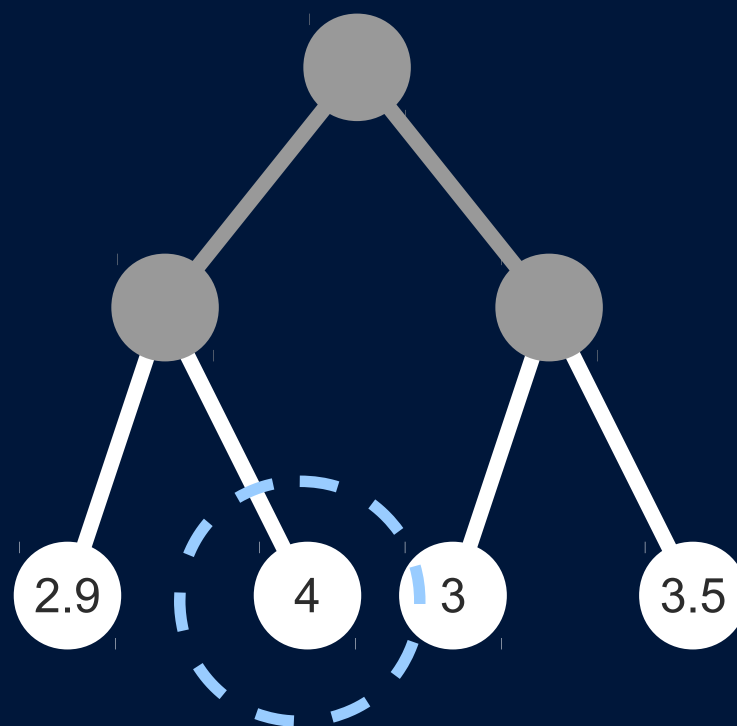
- For each node, compute F-value
- Select next node based on F-value
- More info: [Russel&Norvig 2003]

Heuristic Search – 3

- too bi…
- Idea:
  - avo…
    nor…
- Apply…



Main intuition A*

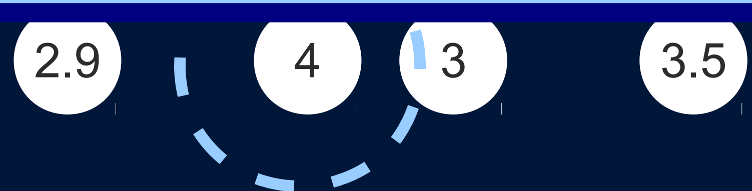


- For each node, compute F-value
- Select next node based on F-value
- More info: [Russel&Norvig 2003]

Heuristic Search – 3

- too big to create
- Idea:
  - avo
    non
- Apply

Main intuition



- For each node, compute F-value
- Select next node based on F-value
- More info: [Russel&Norvig 2003]

F-Value of a node n

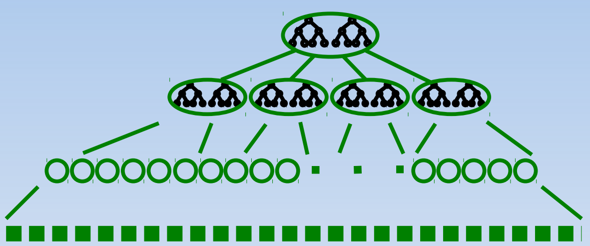
- F(n) is a optimistic estimate
- I.e., F(n) >= V(n') for any descendant n' of n

- F(n) = G(n) + H(n)

reward up to n
(for first *t* stages)

Optimistic estimate of reward below n
(reward for stages t,t+1,...,h-1)

Heuristic Search – 4



- Use heuristics F(n) = G(n) + H(n)
- G(n) – actual reward of reaching n
  - a node at depth t specifies φ^t (i.e., actions for first t stages)

    → can compute $V(\varphi^t)$ over stages 0...t-1
- H(n) – should overestimate!
  - E.g., pretend that it is an MDP
  - compute

$$H(n) = H(\phi^t) = \sum_s P(s \mid \phi^t, b^0) \hat{V}_{MDP}(s)$$

Heuristics – 1

- QPOMDP: Solve 'underlying POMDP'
  - corresponds to immediate communication

$$H(\phi^t) = \sum_{\vec{\theta}^t} P(\vec{\theta}^t | \phi^t, b^0) \hat{V}_{POMDP}(b^{\vec{\theta}^t})$$

- QBG corresponds to 1-step delayed communication
- Hierarchy of upper bounds [Oliehoek et al. 2008]

$$Q^* \leq \hat{Q}_{kBG} \leq \hat{Q}_{BG} \leq \hat{Q}_{POMDP} \leq \hat{Q}_{MDP}$$

Further Developments

- DP
  - Improvements to exhaustive backup [Amato et al. 2009]
  - Compression of values (LPC) [Boularias & Chaib-draa 2008]
  - (Point-based) Memory bounded DP [Seuken & Zilberstein 2007a]
  - Improvements to PB backup [Seuken & Zilberstein 2007b, Carlin and Zilberstein, 2008; Dibangoye et al, 2009; Amato et al, 2009; Wu et al, 2010, etc.]
- Heuristic Search
  - No backtracking: just most promising path [Emery-Montemerlo et al. 2004, Oliehoek et al. 2008]
  - Clustering of histories: reduce number of child nodes [Oliehoek et al. 2009]
  - Incremental expansion: avoid expanding all child nodes [Spaan et al. 2011]
- MILP [Aras and Dutech 2010]

State of the Art

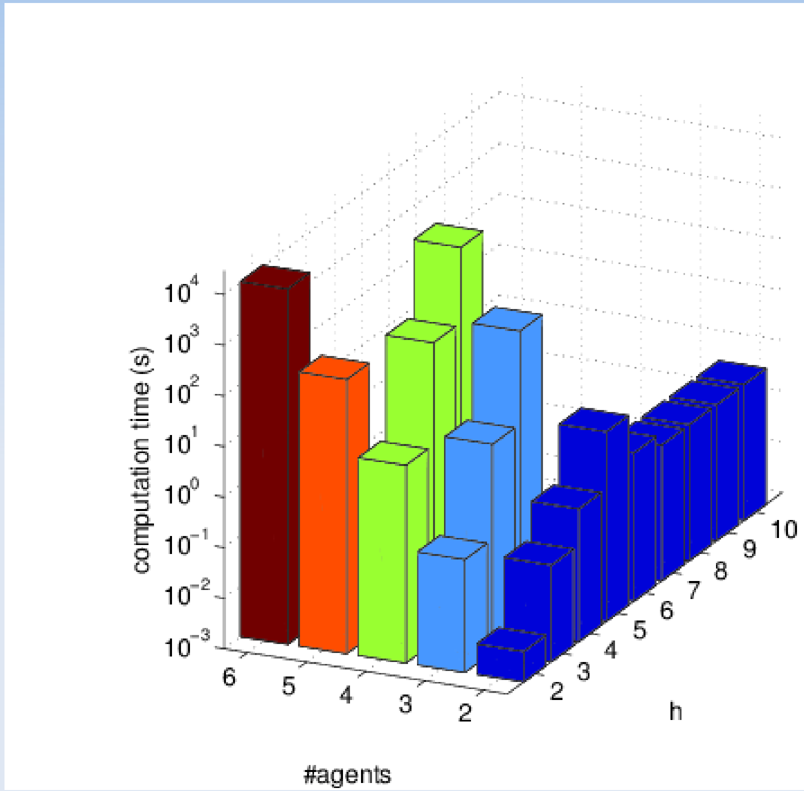
	problem primitives			
	n	$\|\mathcal{S}\|$	$\|\mathcal{A}_i\|$	$\|\mathcal{O}_i\|$
Dec-Tiger	2	2	3	2
BroadcastChannel	2	4	2	2
GridSmall	2	16	5	2
Cooperative Box Pushing	2	100	4	5
Recycling Robots	2	4	3	2
Hotel 1	2	16	3	4
FireFighting	2	432	3	2

h	MILP	DP-LPC	DP-IPG	GMAA — Q_{BG}		
				IC	ICE	heur
BroadcastChannel, ICE solvable to $h = 900$						
2	0.38	≤ 0.01	0.09	≤ 0.01	≤ 0.01	≤ 0.01
3	1.83	0.50	56.66	≤ 0.01	≤ 0.01	≤ 0.01
4	34.06	*	*	≤ 0.01	≤ 0.01	≤ 0.01
5	48.94			≤ 0.01	≤ 0.01	≤ 0.01
Dec-Tiger, ICE solvable to $h = 6$						
2	0.69	0.05	0.32	≤ 0.01	≤ 0.01	≤ 0.01
3	23.99	60.73	55.46	≤ 0.01	≤ 0.01	≤ 0.01
4	*	–	2286.38	0.27	≤ 0.01	0.03
5			–	21.03	0.02	0.09
FireFighting (2 agents, 3 houses, 3 firelevels), ICE solvable to $h \gg 1000$						
2	4.45	8.13	10.34	≤ 0.01	≤ 0.01	≤ 0.01
3	–	–	569.27	0.11	0.10	0.07
4			–	950.51	1.00	0.65
GridSmall, ICE solvable to $h = 6$						
2	6.64	11.58	0.18	0.01	≤ 0.01	≤ 0.01
3	*	–	4.09	0.10	≤ 0.01	0.42
4			77.44	1.77	≤ 0.01	67.39
Recycling Robots, ICE solvable to $h = 70$						
2	1.18	0.05	0.30	≤ 0.01	≤ 0.01	≤ 0.01
3	*	2.79	1.07	≤ 0.01	≤ 0.01	≤ 0.01
4		2136.16	42.02	≤ 0.01	≤ 0.01	0.02
5		–	1812.15	≤ 0.01	≤ 0.01	0.02
Hotel 1, ICE solvable to $h = 9$						
2	1.92	6.14	0.22	≤ 0.01	≤ 0.01	0.03
3	315.16	2913.42	0.54	≤ 0.01	≤ 0.01	1.51
4	–	–	0.73	≤ 0.01	≤ 0.01	3.74
5			1.11	≤ 0.01	≤ 0.01	4.54
9			8.43	0.02	≤ 0.01	20.26
10			17.40	#	#	
15			283.76			
Cooperative Box Pushing (Q_{POMDP}), ICE solvable to $h = 4$						
2	3.56	15.51	1.07	≤ 0.01	≤ 0.01	≤ 0.01
3	2534.08	–	6.43	0.91	0.02	0.15
4	–		1138.61	*	328.97	0.63

'–' memory limit violations
'*' time limit overruns
'#' heuristic bottleneck

State of the Art

h	V^*	T_{GMAA*}(s)	T_{IC}(s)	T_{ICE}(s)
		RECYCLING ROBOTS		
3	10.660125	≤ 0.01	≤ 0.01	≤ 0.01
4	13.380000	713.41	≤ 0.01	≤ 0.01
5	16.486000	–	≤ 0.01	≤ 0.01
6	**19.554200**		≤ 0.01	≤ 0.01
10	**31.863889**		≤ 0.01	≤ 0.01
15	**47.248521**		≤ 0.01	≤ 0.01
20	**62.633136**		≤ 0.01	≤ 0.01
30	**93.402367**		0.08	0.05
40	**124.171598**		0.42	0.25
50	**154.940828**		2.02	1.27
70	**216.479290**		–	28.66
80			–	–
		BROADCASTCHANNEL		
4	3.890000	≤ 0.01	≤ 0.01	≤ 0.01
5	4.790000	1.27	≤ 0.01	≤ 0.01
6	**5.690000**	–	≤ 0.01	≤ 0.01
7	**6.590000**		≤ 0.01	≤ 0.01
10	**9.290000**		≤ 0.01	≤ 0.01
25	**22.881523**		≤ 0.01	≤ 0.01
50	**45.501604**		≤ 0.01	≤ 0.01
100	**90.760423**		≤ 0.01	≤ 0.01
250	**226.500545**		0.06	0.07
500	**452.738119**		0.81	0.94
700	**633.724279**		0.52	0.63
800			–	–
900	**814.709393**		9.57	11.11
1000			–	–



Scalability w.r.t. #agents

Cases that compress well

* excluding heuristic

State of The Art

Approximate (no quality guarantees)

- MBDP: linear in horizon [Seuken & zilberstein 2007a]
- Rollout sampling extension: up to 20 agents [Wu et al. 2010b]
- Transfer planning: use smaller problems to solve large (structured) problems (up to 1000) agents [Oliehoek et al. 2013]

Related Areas

- Partially observable stochastic games [Hansen et al. 2004]
  - Non-identical payoff
- Interactive POMDPs [Gmytrasiewicz & Doshi 2005, JAIR]
  - Subjective view of MAS
- Imperfect information extensive form games
  - Represented by game tree
  - E.g., poker [Sandholm 2010, AI Magazine]

References

- References can be found on the tutorial website:

  www.st.ewi.tudelft.nl/~mtjspaan/tutorialDMuU/

- Further references can be found in

  Frans A. Oliehoek. **Decentralized POMDPs**. In Wiering, Marco and van Otterlo, Martijn, editors, *Reinforcement Learning: State of the Art*, Adaptation, Learning, and Optimization, pp. 471–503, Springer Berlin Heidelberg, Berlin, Germany, 2012.

  - Available from http://people.csail.mit.edu/fao/

Decision making under uncertainty

Matthijs Spaan[1] and Frans Oliehoek[2]

[1] Delft University of Technology
[2] Maastricht University

Part 4: Selected Further Topics

European Agent Systems Summer School (EASSS '13)

www.st.ewi.tudelft.nl/~mtjspaan/tutorialDMuU/

Some Further Topics

High-level overview:

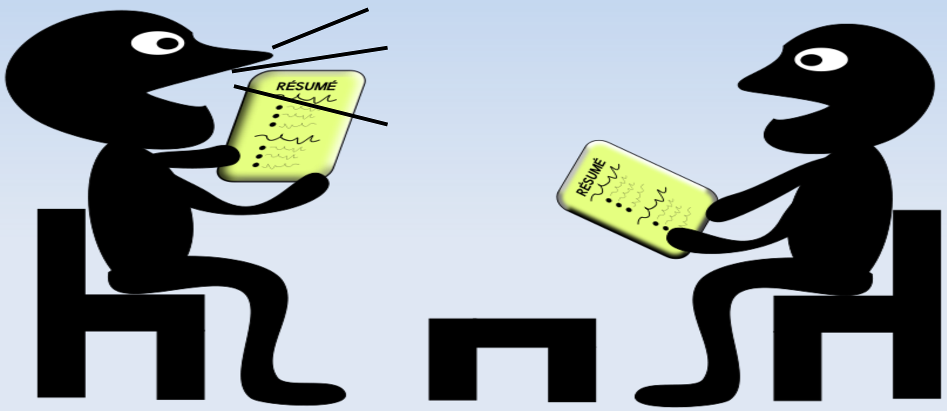
- Communication
- Factored Models
  - Single Agent
  - Multiple agents

Communication

- Already discussed:
  instantaneous cost-free and noise-free communication
  - Dec-MDP → multiagent MDP (MMDP)
  - Dec-POMDP → multiagent POMDP (MPOMDP)
- but in practice:
  - probability of failure
  - delays
  - costs
- Also: implicit communication!
  (via observations and actions)

Implicit Communication

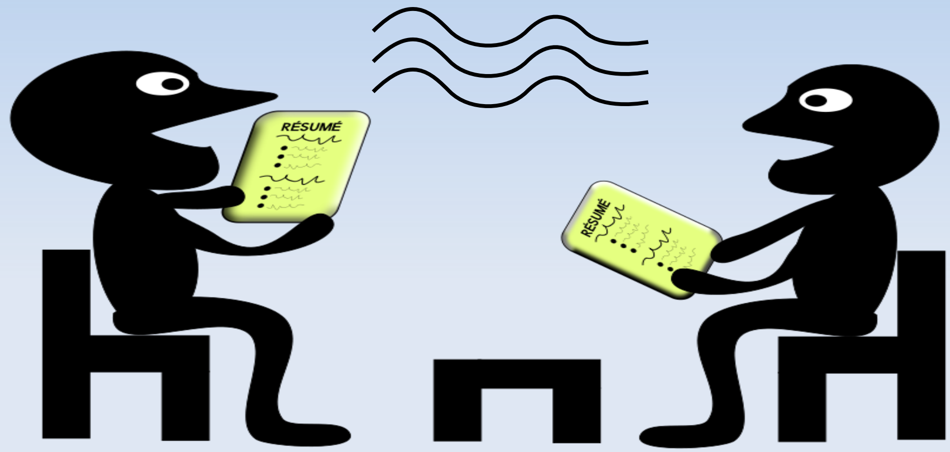
- Encode communications by actions and observations



- Embed the **optimal meaning** of messages by finding the optimal plan [Goldman and Zilberstein 2003, Spaan et al. 2006]

Implicit Communication

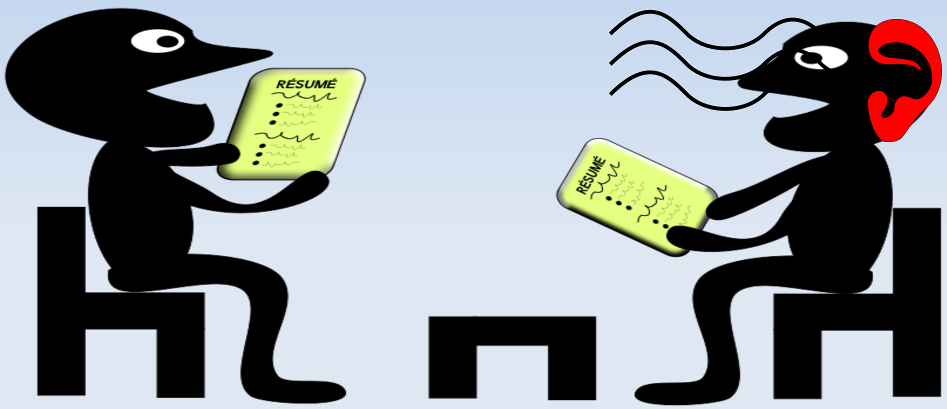
- Encode communications by actions and observations



- Embed the **optimal meaning** of messages by finding the optimal plan [Goldman and Zilberstein 2003, Spaan et al. 2006]

Implicit Communication

- Encode communications by actions and observations



- Embed the **optimal meaning** of messages by finding the optimal plan [Goldman and Zilberstein 2003, Spaan et al. 2006]
- E.g. communication bit
  - doubles the #actions and observations!
  - Clearly, useful... but intractable for general settings
    (perhaps for analysis of very small communication systems)

Explicit Communication

- perform a particular information update (e.g., sync) as in the MPOMDP:
  - each agent broadcasts its information, and
  - each agent uses that to perform joint belief update
- Other approaches:
  - Communication cost [Becker et al. 2005]
  - Delayed communication [Hsu 1982, Spaan 2008, Oliehoek 2012]
  - communicate every k stages [Goldman & Zilberstein 2008]

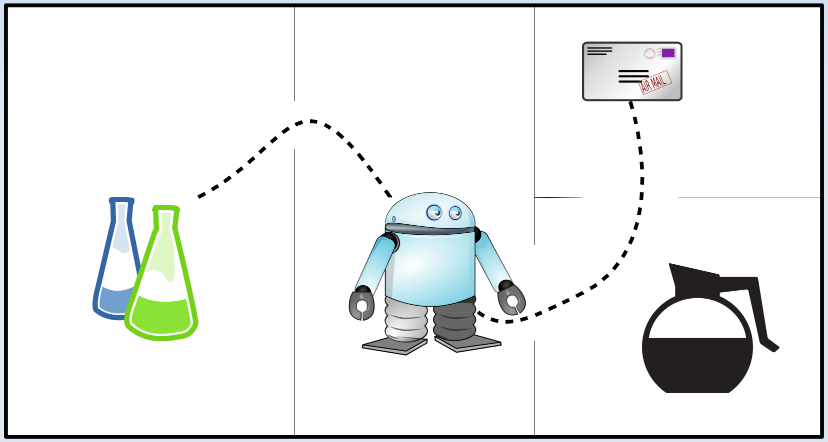
Some Further Topics

Overview:

- On-line planning
- Communication
- **Factored Models**
  - Single Agent
  - Multiple agents

Factored MDPs

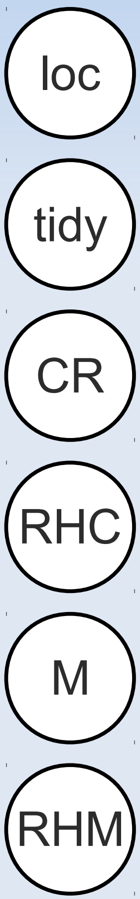
- So far: used 'states'
- But in many problems states are **factored**
  - state is an assignment of variables $s=<f_1,f_2,...,f_k>$
  - *factored MDP* [Boutilier et al. 99 JAIR]



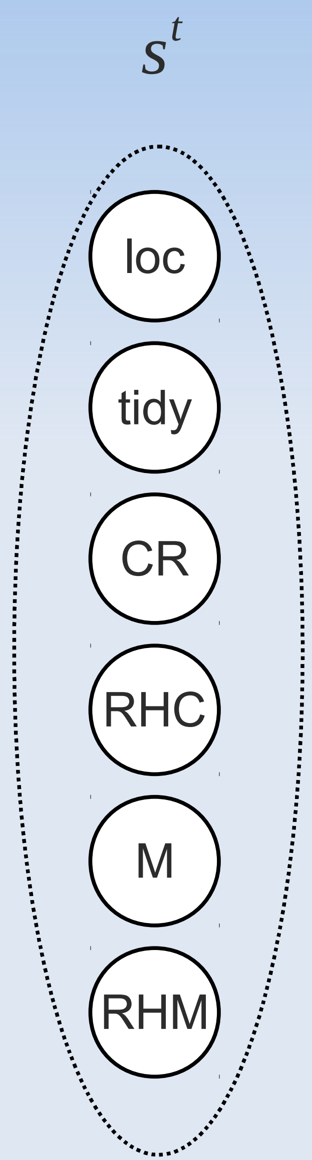
Examples:

- Predator-prey: x, y coordinate!
- Robotic P.A.
  - location of robot (lab, hallway, kitchen, mail room), tidiness of lab, coffee request, robot holds coffee, mail present, robot holds mail, etc.
  - Actions: move (2 directions), pickup coffee/mail, deliver coffee/mail

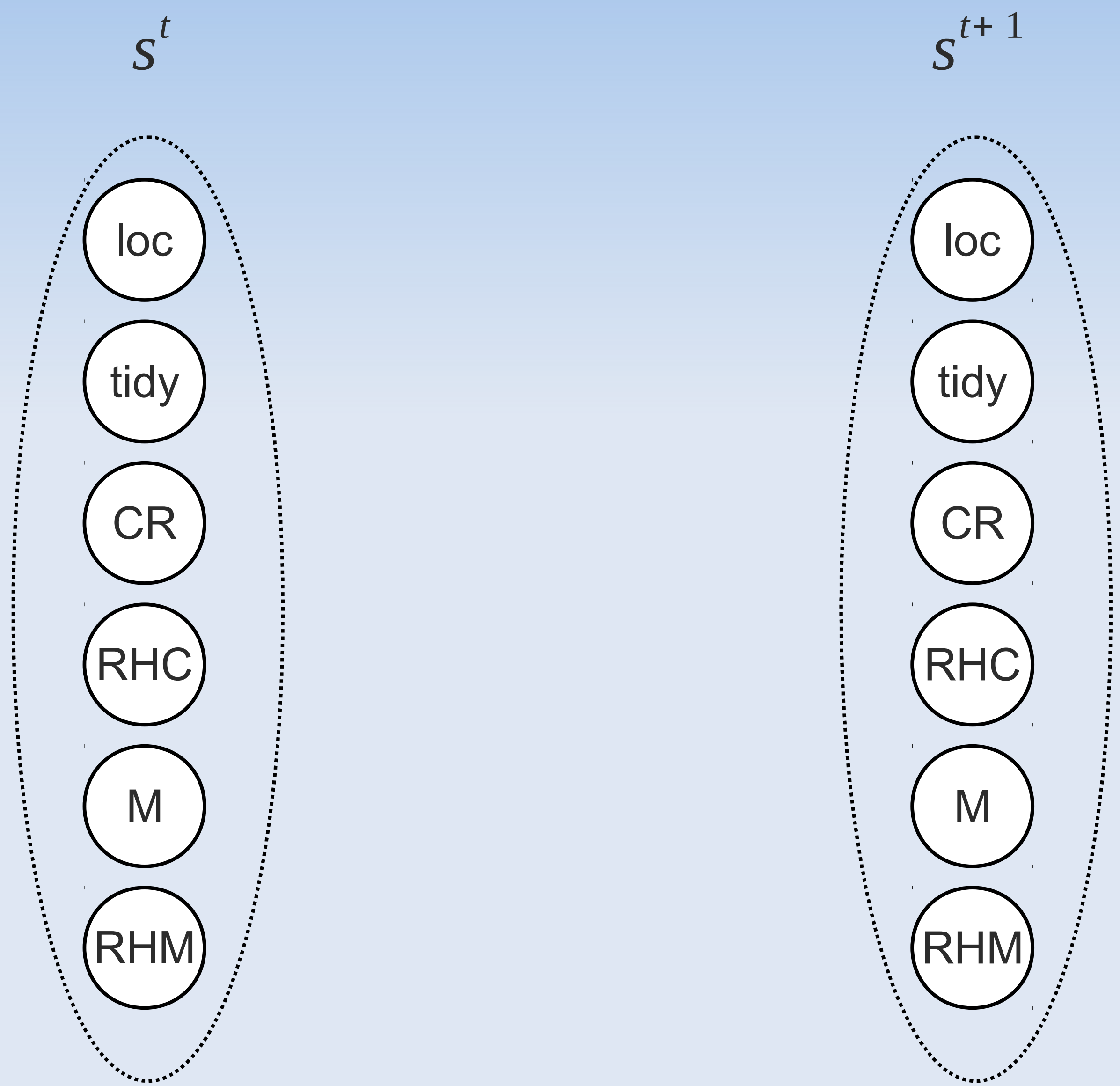
Factored States & Transitions



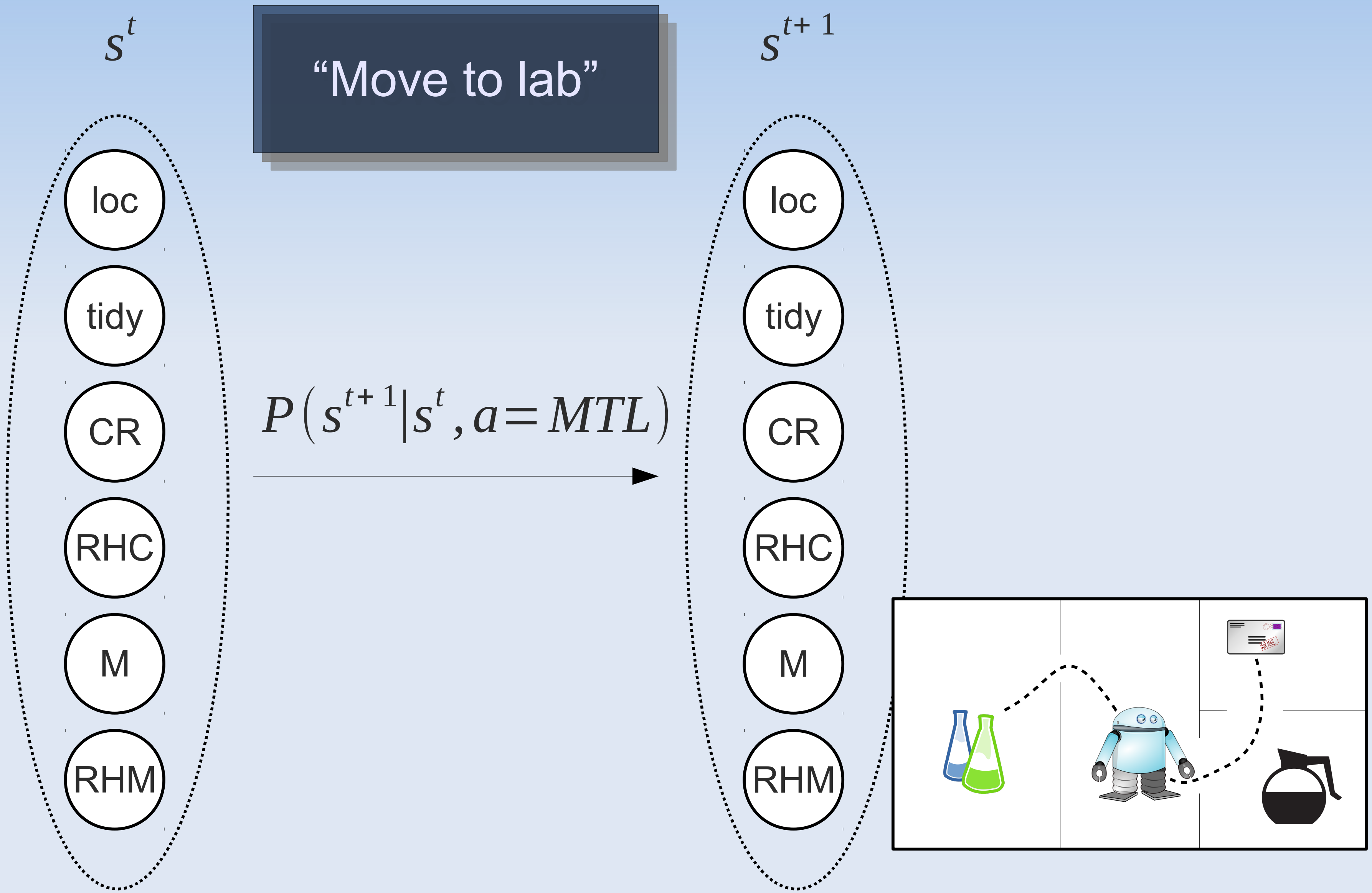
Factored States & Transitions



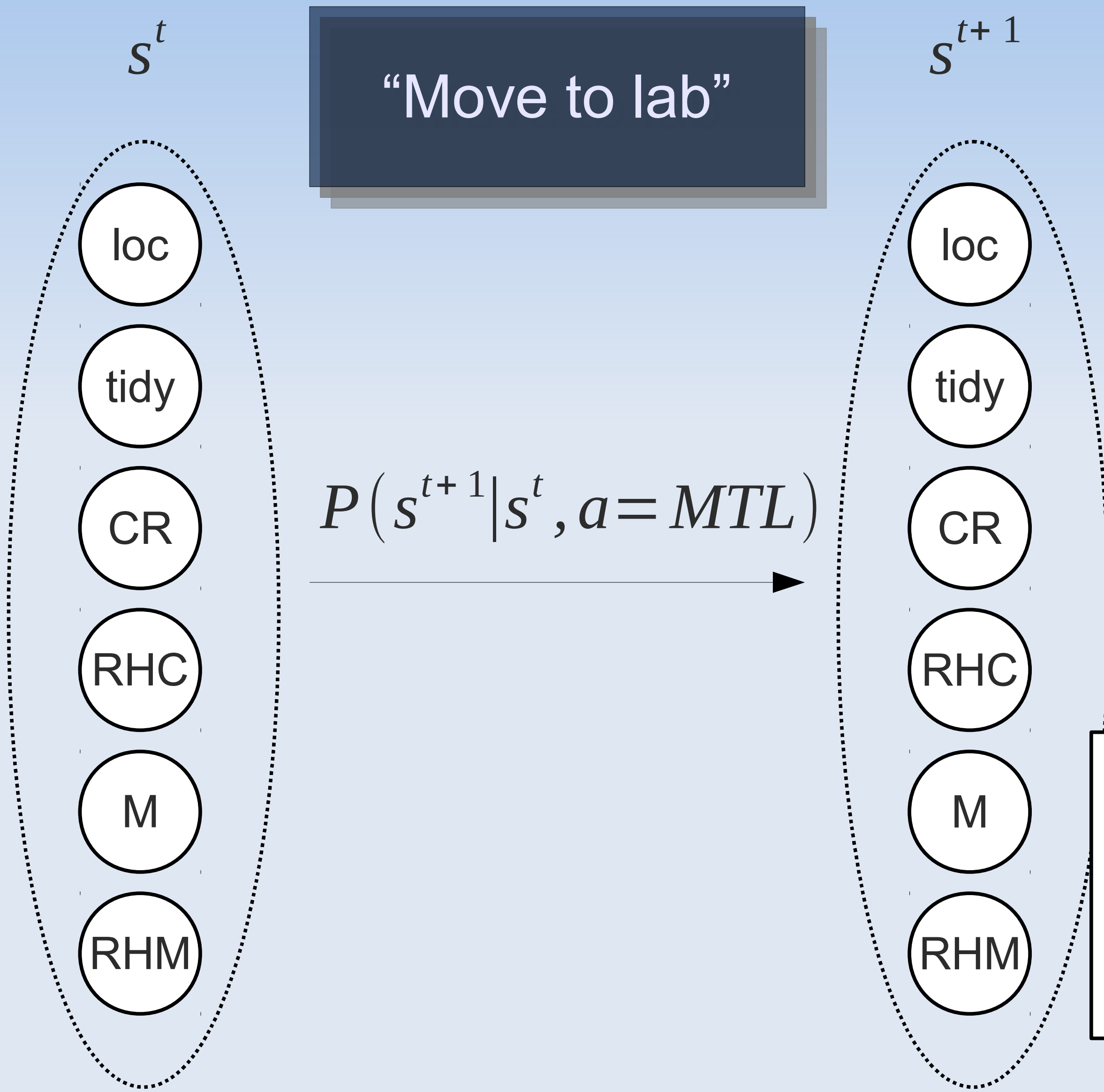
Factored States & Transitions



Factored States & Transitions

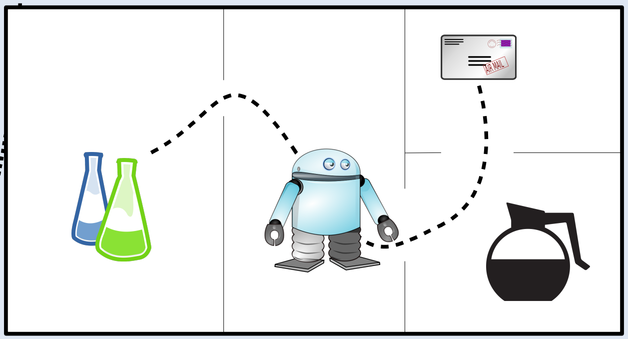


Factored States & Transitions

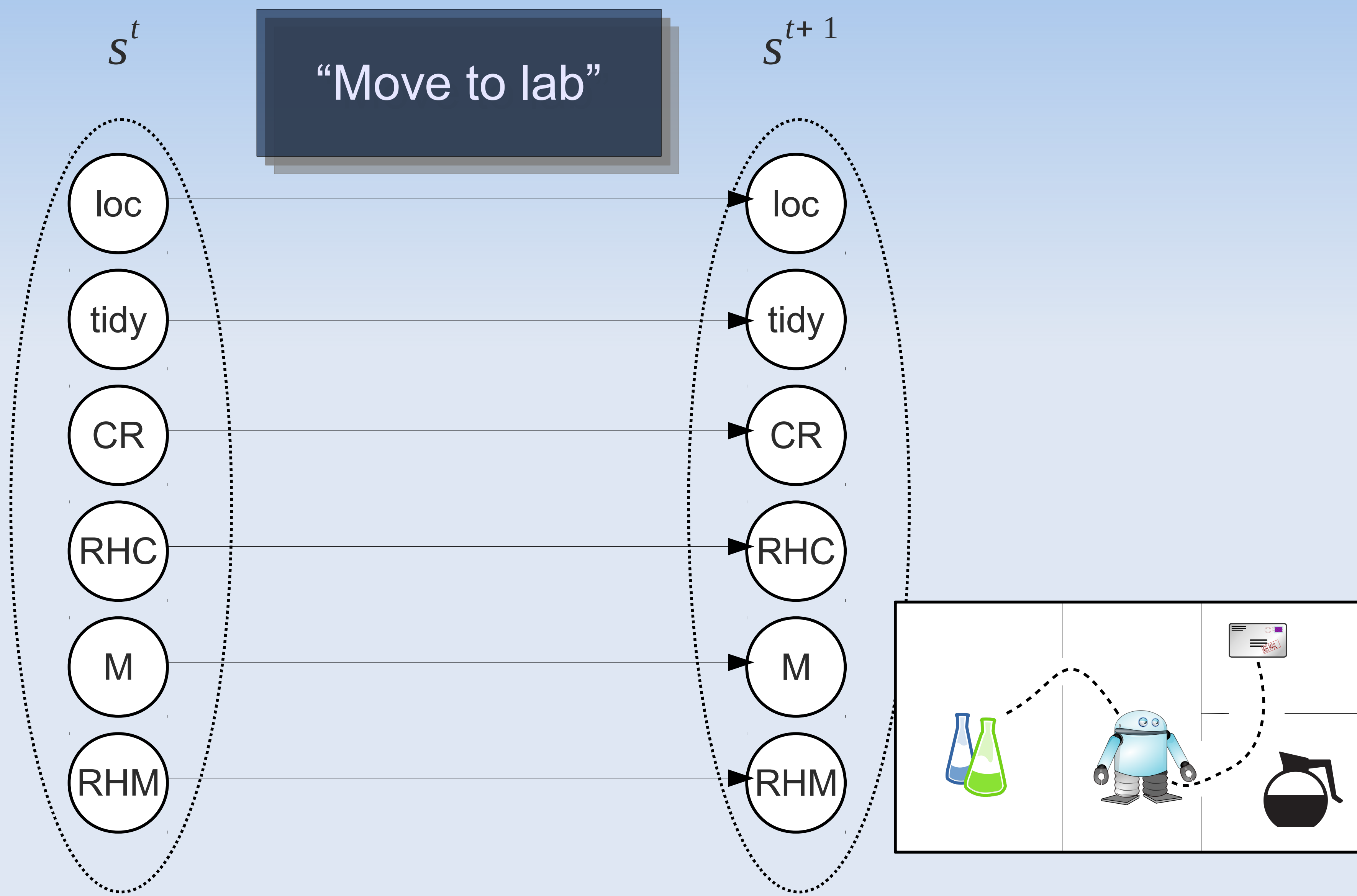


But there is conditional independence!

E.g.: 'M' does not influence 'loc'

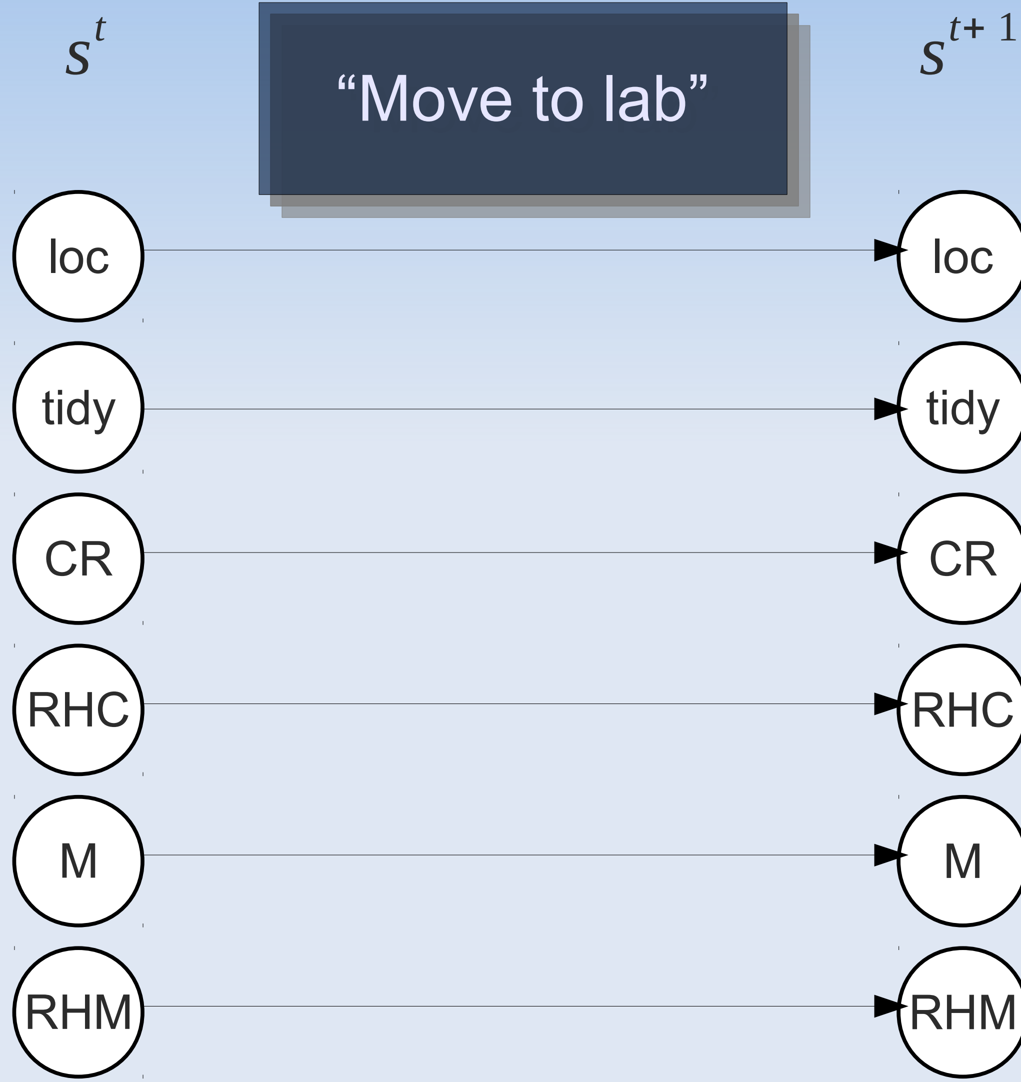


Factored States & Transitions



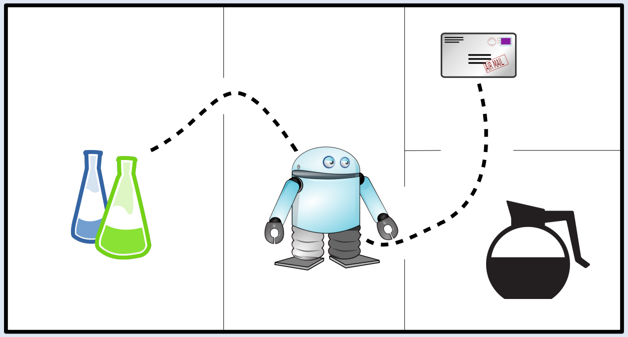
Factored States & Transitions

“Move to lab”



		L	H	K	M
	L	1	.9	0	0
t+1	H	0	.1	.9	0
	K	0	0	.1	.9
	M	0	0	0	.1

conditional probability table (CPT)



Factored States & Transitions

		L	H	K	M
	L	1	.9	0	0
t+1	H	0	.1	.9	0
	K	0	0	.1	.9
	M	0	0	0	.1

conditional probability table (CPT)

- Each next-stage variable has a CPT
- This allows for a much more compact representation!
- "Two-stage dynamic Bayesian network" (2DBN)

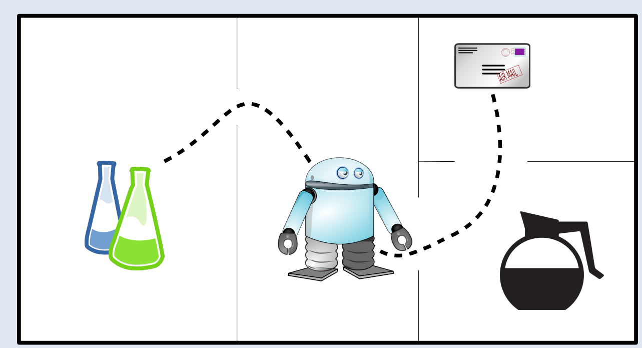
Factored States & Transitions

		L	H	K	M
	L	1	.9	0	0
t+1	H	0	.1	.9	0
	K	0	0	.1	.9
	M	0	0	0	.1

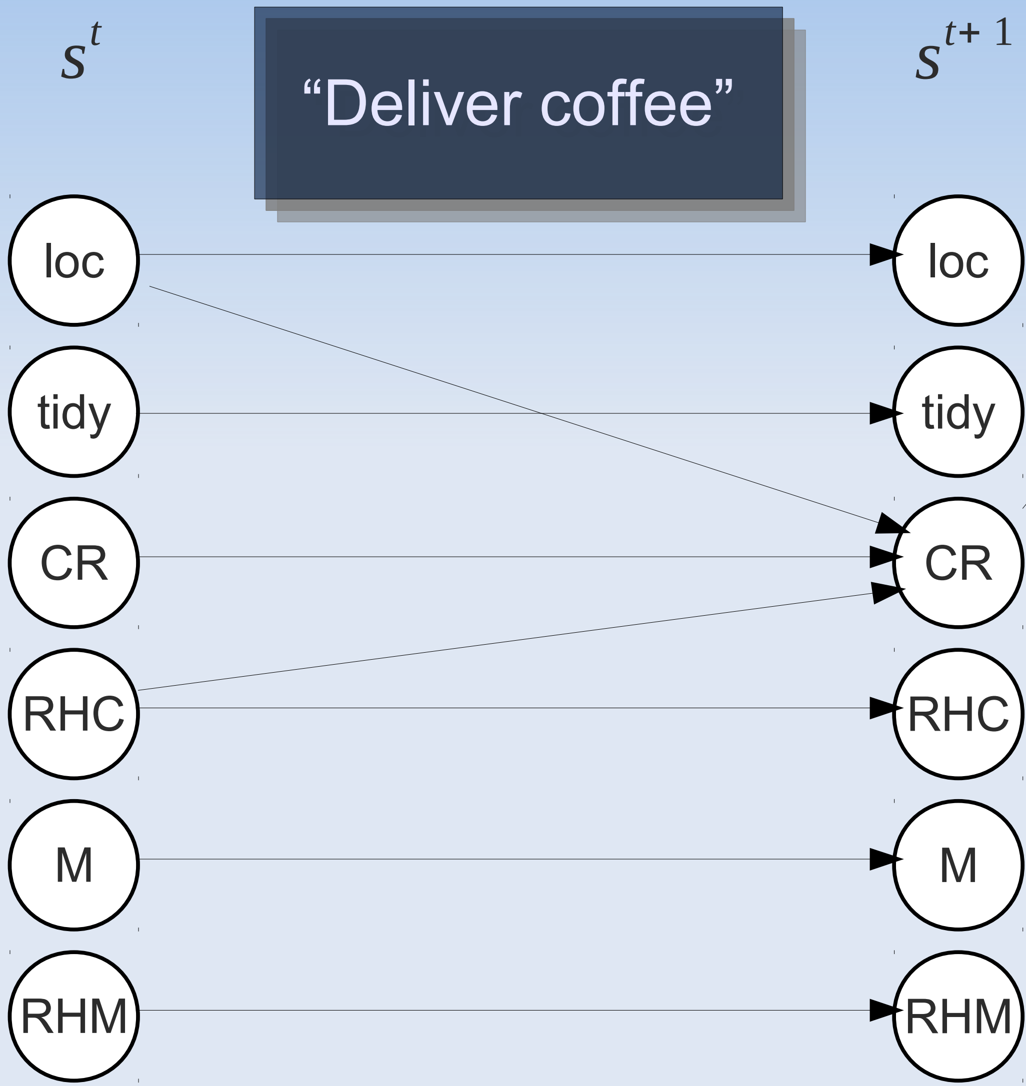
conditional probability table (CPT)

Do we always have so much independence?

(what about other actions?)



Factored States & Transitions



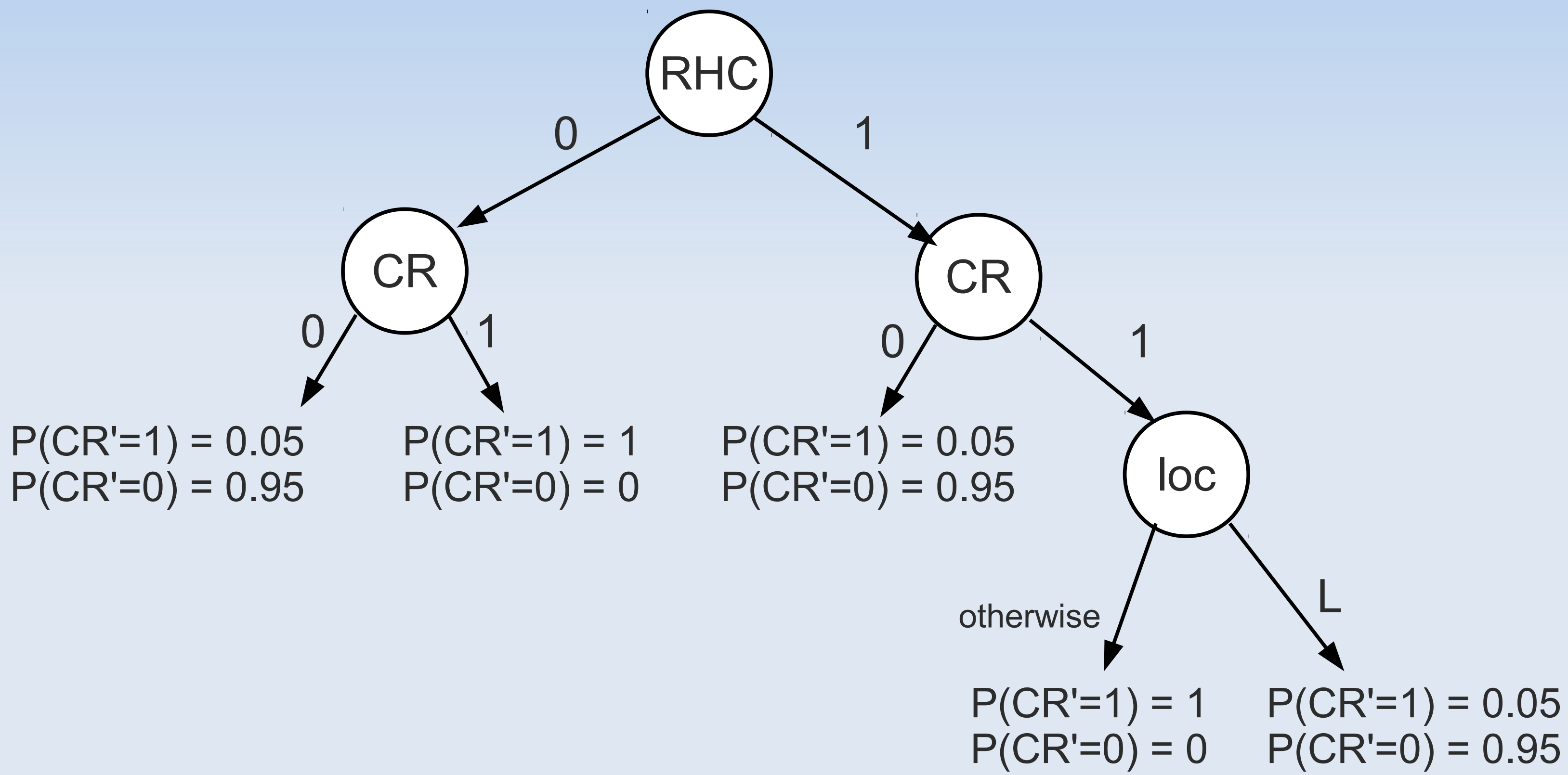
CPT encodes that IF

- loc=lab
- CR=1

→ high probability of CR becoming 0

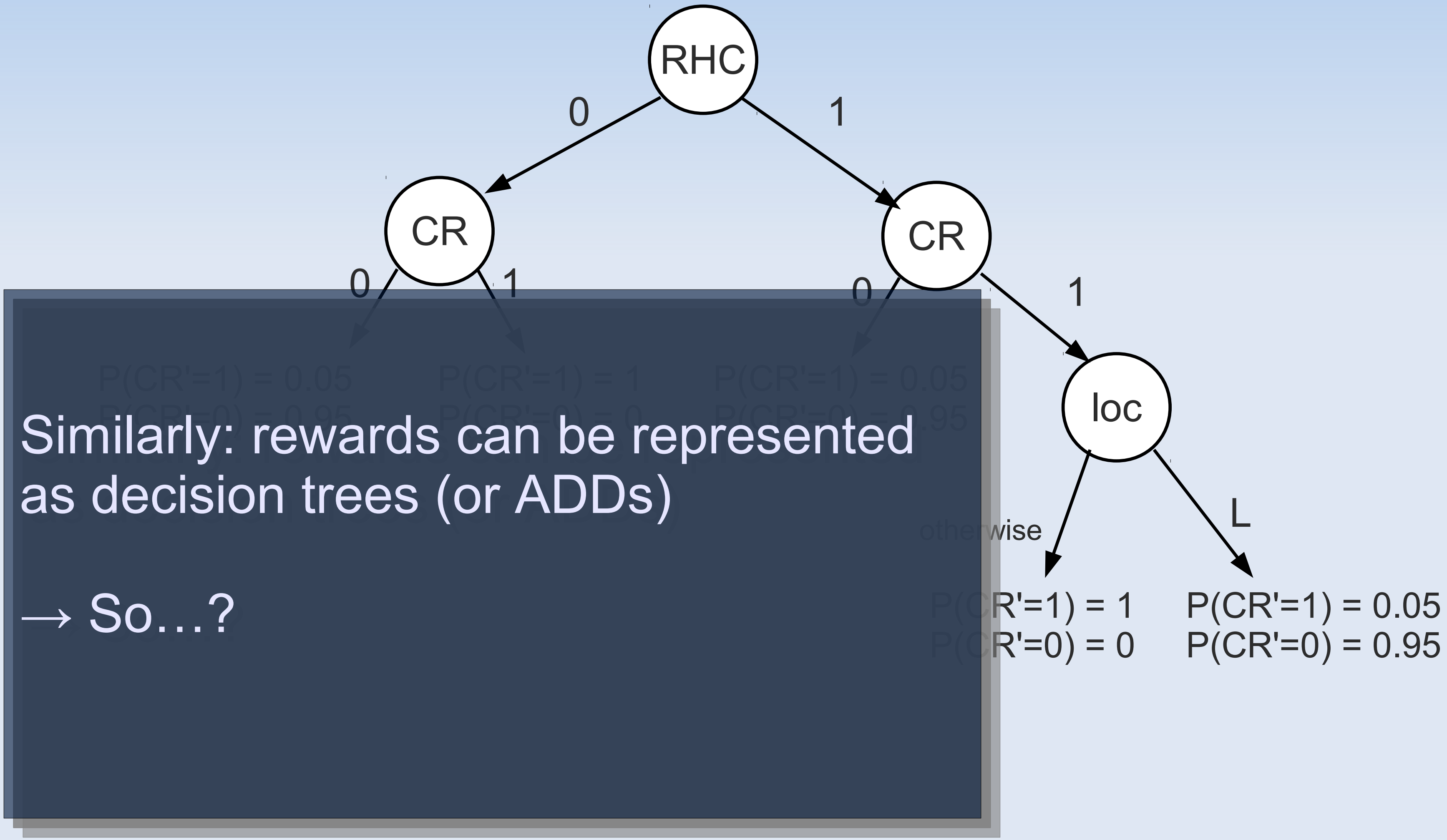
Solving Factored MDPs

- CPT also representable as a decision tree



Solving Factored MDPs

- CPT also representable as a decision tree

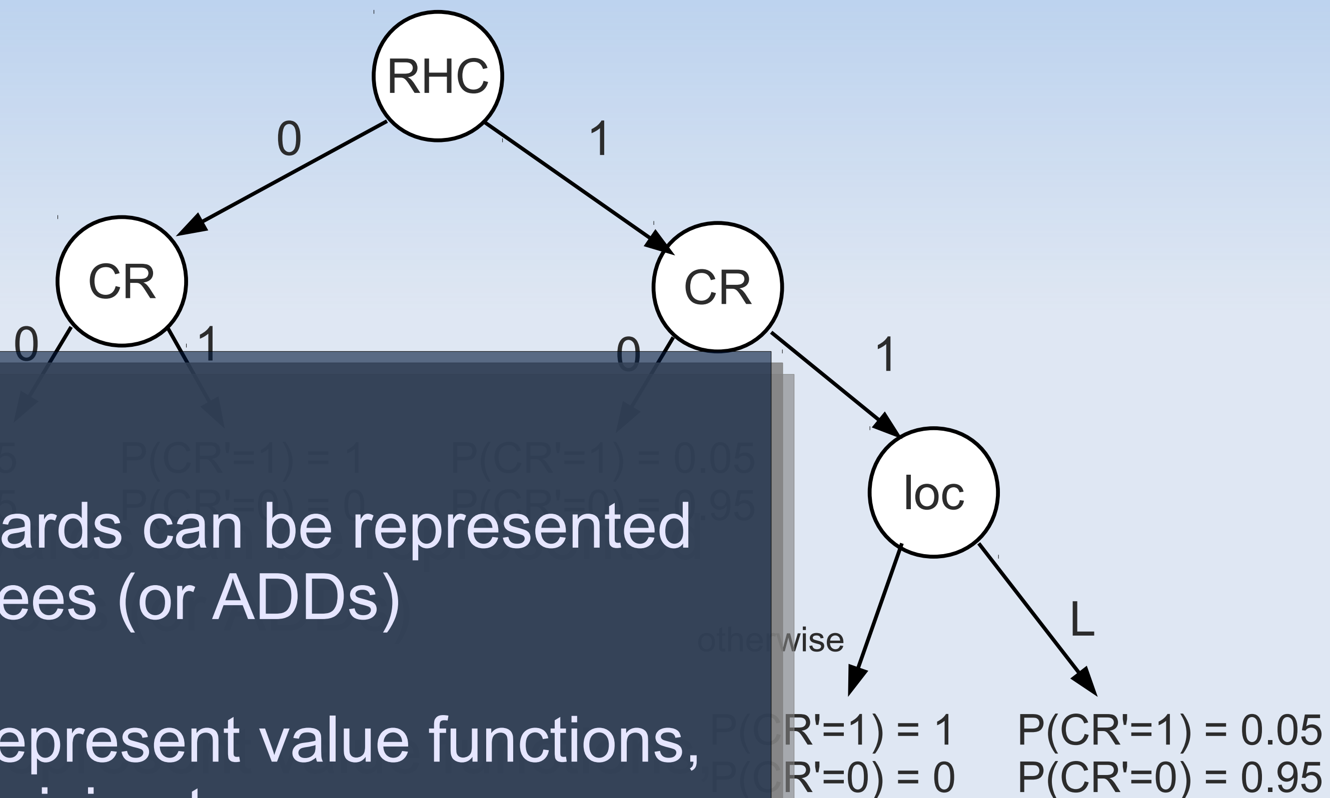


Similarly: rewards can be represented as decision trees (or ADDs)

→ So…?

Solving Factored MDPs

- CPT also representable as a decision tree



Similarly: rewards can be represented as decision trees (or ADDs)

→ Can also represent value functions, policies as decision trees [Boutilier et al 99]

Factored POMDPs

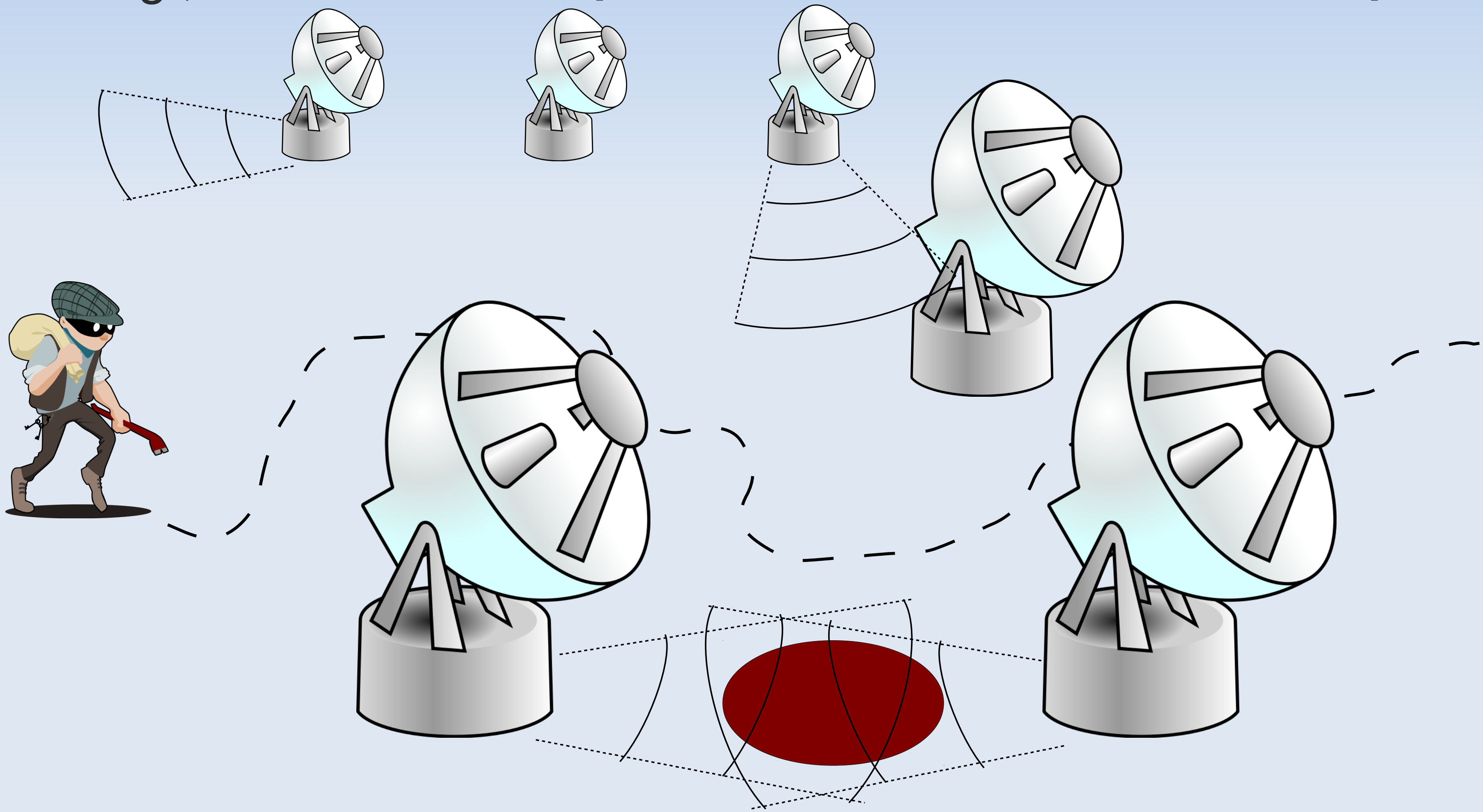
- Of course POMDP models can also be factored
- Similar ideas applied [Hansen & Feng 2000, Poupart 2005, Shani et al. 2008]
  - α-vectors represented by ADDs
  - beliefs too.
- This does not solve all problems:
  - over time state factors get more and more correlated, so representation grows large.

Factored Multiagent Models

- Of course multiagent models can also be factored!
- Work can be categorized in a few directions:
  - Trying to execute the factored (PO)MDP policy
    [Roth et al. 2007, Messias et al. 2011]
  - Trying to execute independently as much as possible
    [Spaan & Melo 2008, Melo & Veloso 2011]
  - Exploiting graphical structure between agents
    (ND-POMDPs, Factored Dec-POMDPs)
  - Influence-based abstraction of policies of other agents
    (TOI-Dec-MDPs, TD-POMDPs, IBA for POSGs)

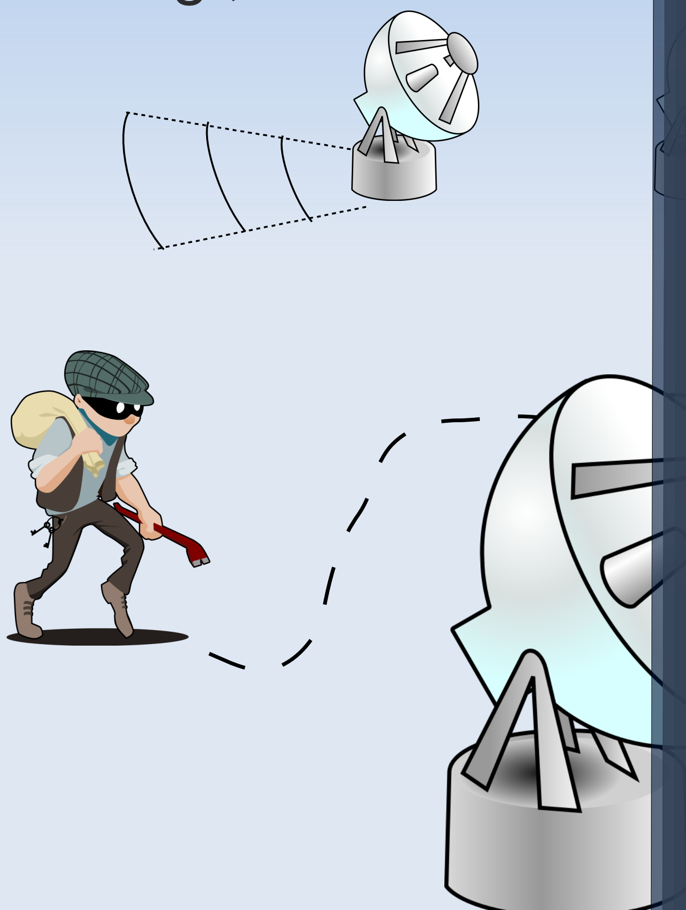
Graphical Structure between Agents

- Exploit (conditional) independence between agents
  - E.g., sensor networks [Nair et al '05 AAAI, Varakantham et al. '07 AAMAS]



Graphical Structure between Agents

- Exploit (conditional) independence between agents
  - E.g., sensor networks [Nair et al. 05 AAAI, Varakantham 07 AAMAS]



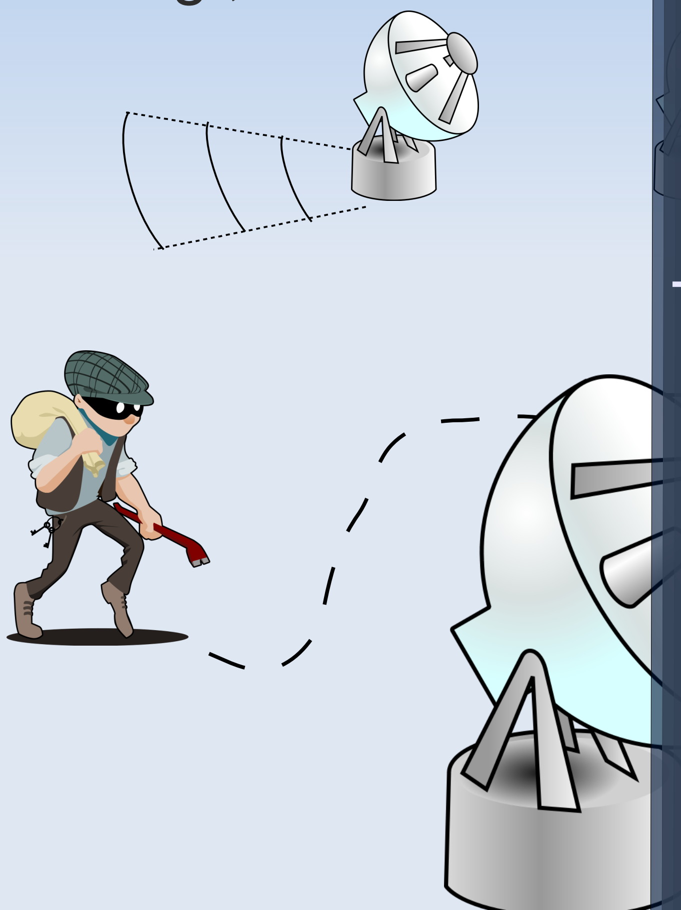
These problems have

- State that cannot be influenced
- Factored reward function

$$R(s,a)=\sum_e R_e(s,a_e)$$

Graphical Structure between Agents

- Exploit (conditional) independence between agents
  - E.g., sensor networks

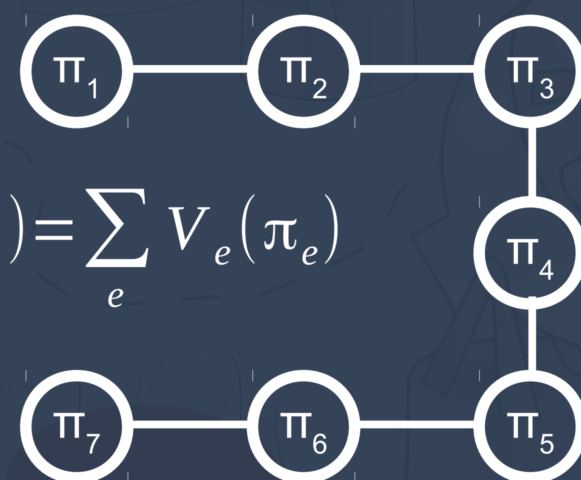


These problems have
- State that cannot be influenced
- Factored reward function

$$R(s,a)=\sum_e R_e(s,a_e)$$

This allows a reformulation as a (D)COP

$$V(\pi)=\sum_e V_e(\pi_e)$$



Graphical Structure between Agents

- Exploit (conditional) independence between agents
  - E.g., sensor networks [Nair et al. 05 AAAI; Varakantham 07 AAMAS]

This can be solved more efficiently than by looping through all π !

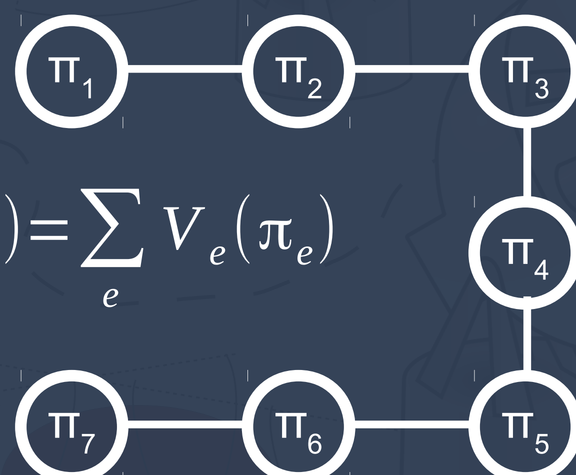
These problems have

- State that cannot be influenced
- Factored reward function

$$R(s,a)=\sum_e R_e(s,a_e)$$

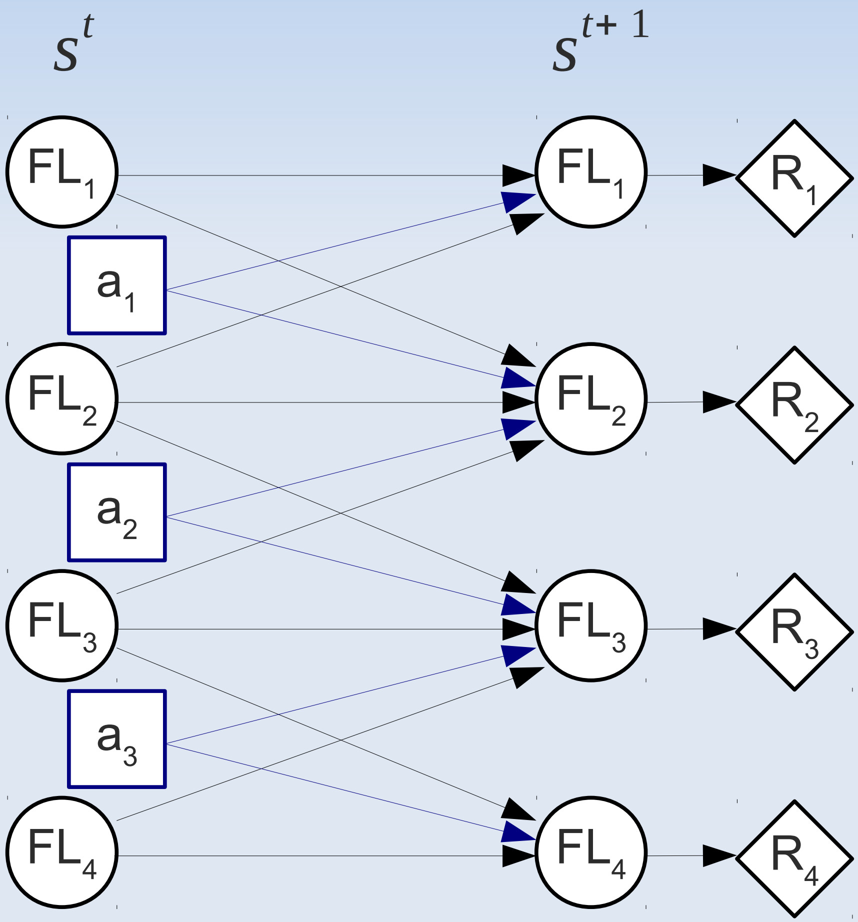
This allows a reformulation as a (D)COP

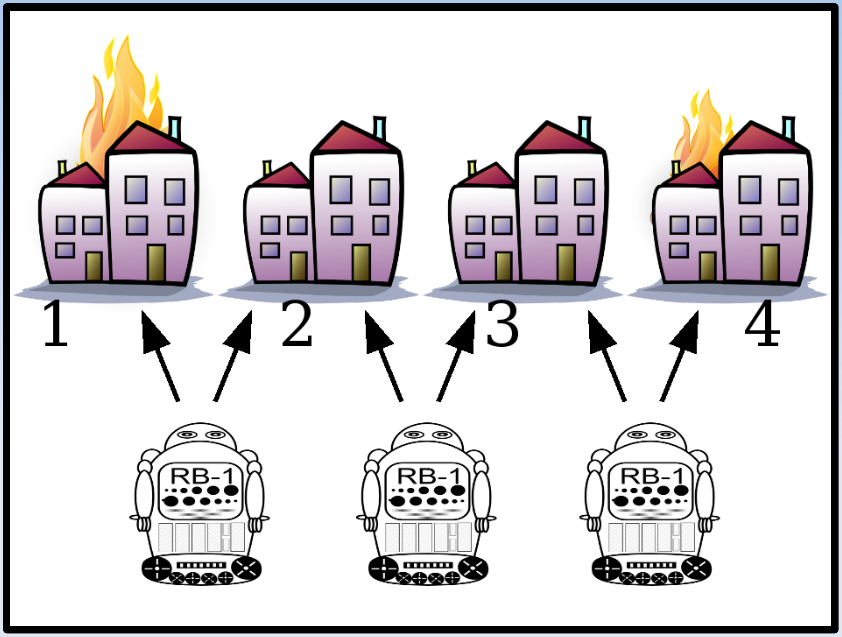
$$V(\pi)=\sum_e V_e(\pi_e)$$



Graphical Structure between Agents

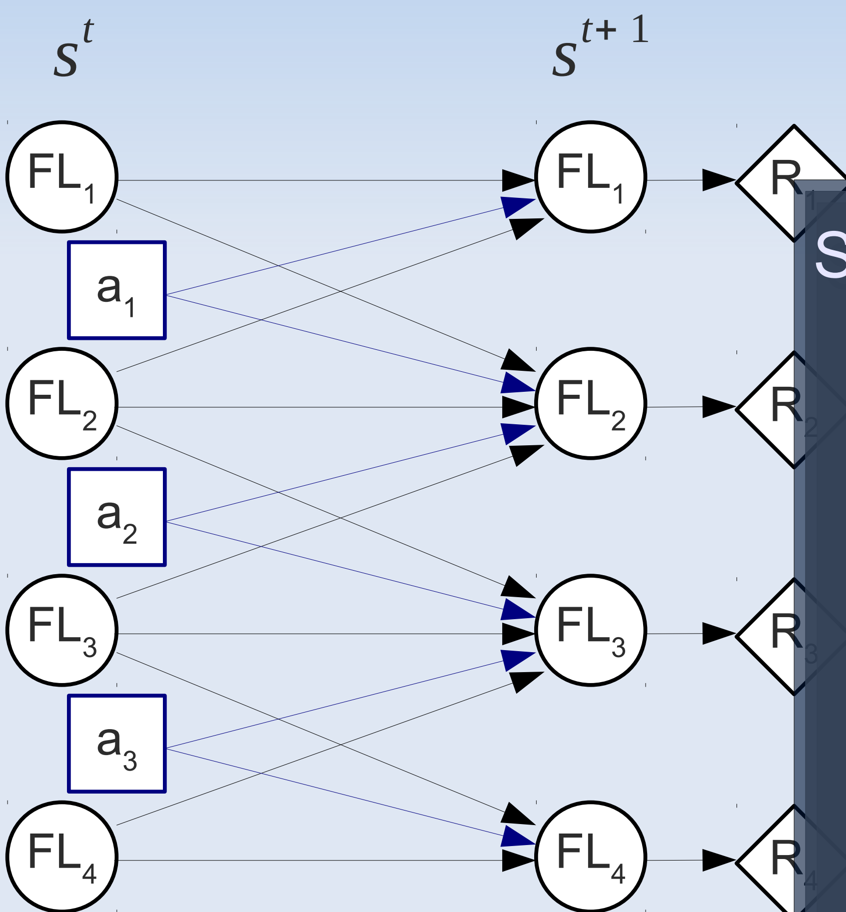
- Factored Dec-POMDPs
  [Oliehoek et al. 2008 AAMAS]

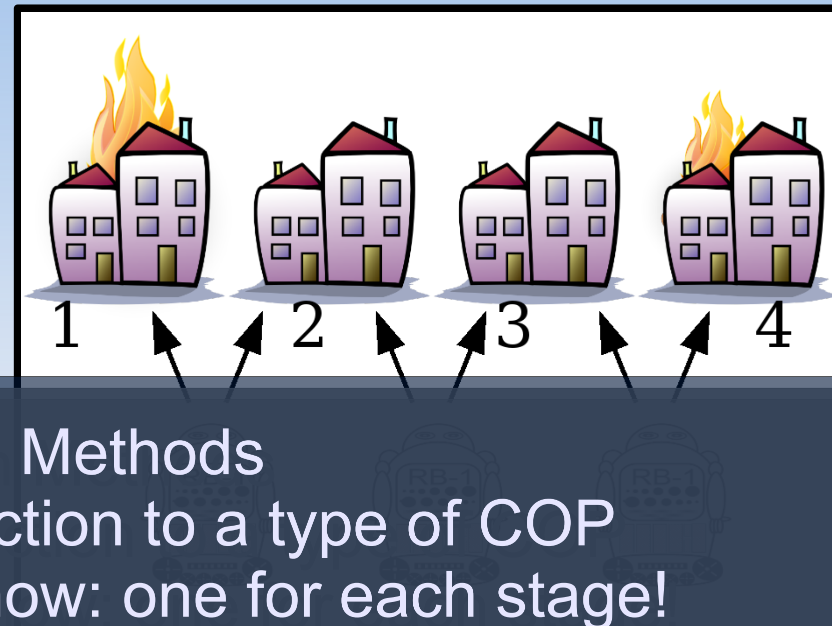




Graphical Structure between Agents

- Factored Dec-POMDPs
  [Oliehoek et al. 2008 AAMAS]





Solution Methods

- reduction to a type of COP
- but now: one for each stage!

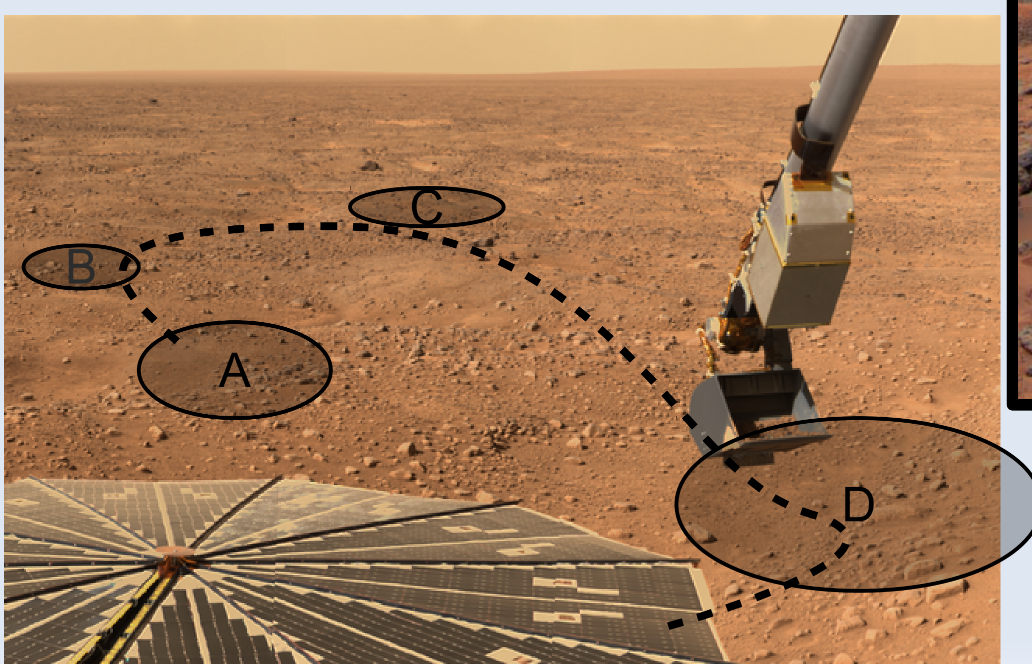
- δ is a decision rule
  (part of policy for 1 stage t)

→ leads to factored form of heuristic search
[Oliehoek 2013 AAMAS]

Influence-Based Abstraction

- Try to define agents' **local state**
- Analyze how policies of other agents affect it
  - find compact description for this **influence**

- Example: Mars Rovers [Becker et al. 2004 JAIR]
  - 2 rovers collect data at 4 sites

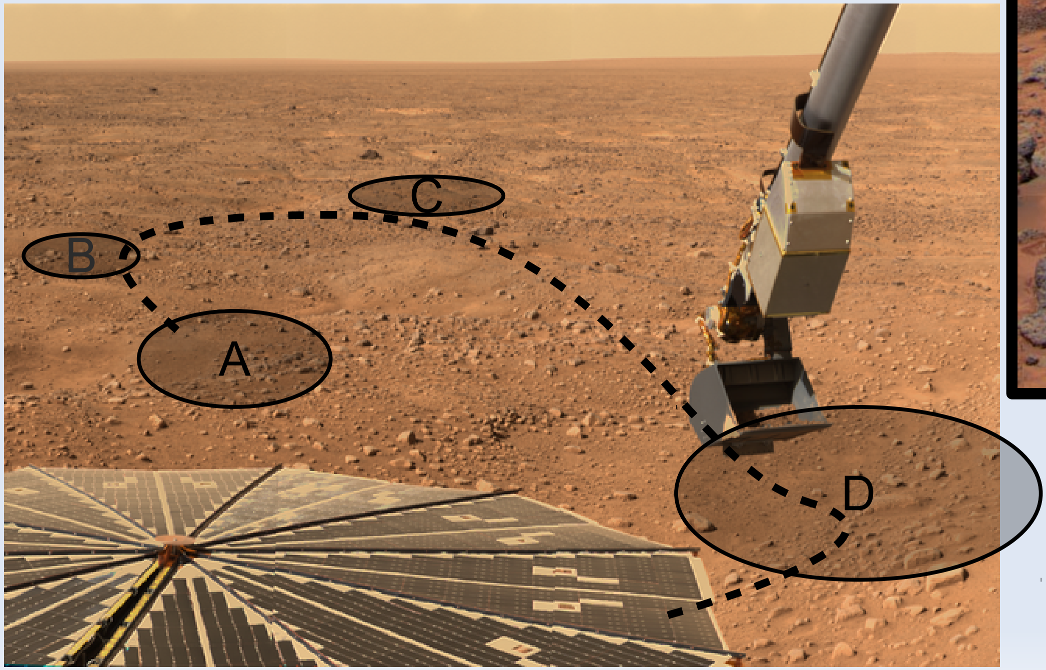


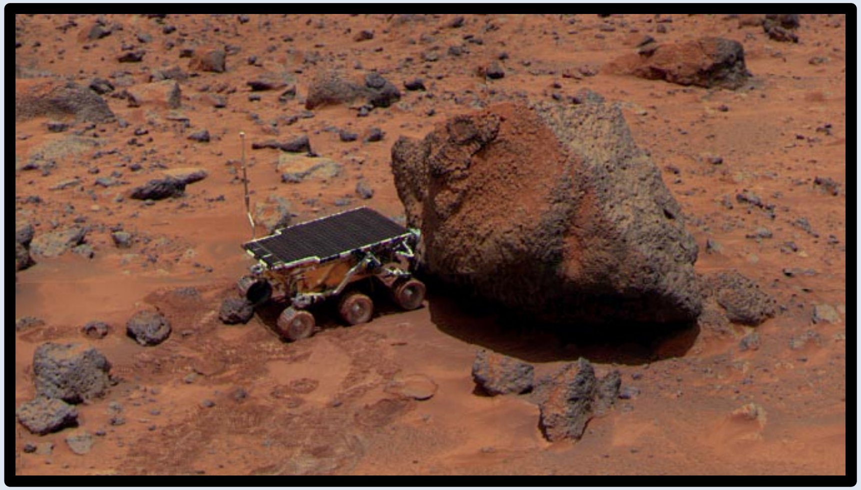
Influence-Based Abstraction

Transitions **independent**: Rovers drive independently
Rewards are **dependent**:

- 2 same soil samples of same site not so useful (sub additive)
- 2 pictures of (different sides) of same rock is useful (super additive)

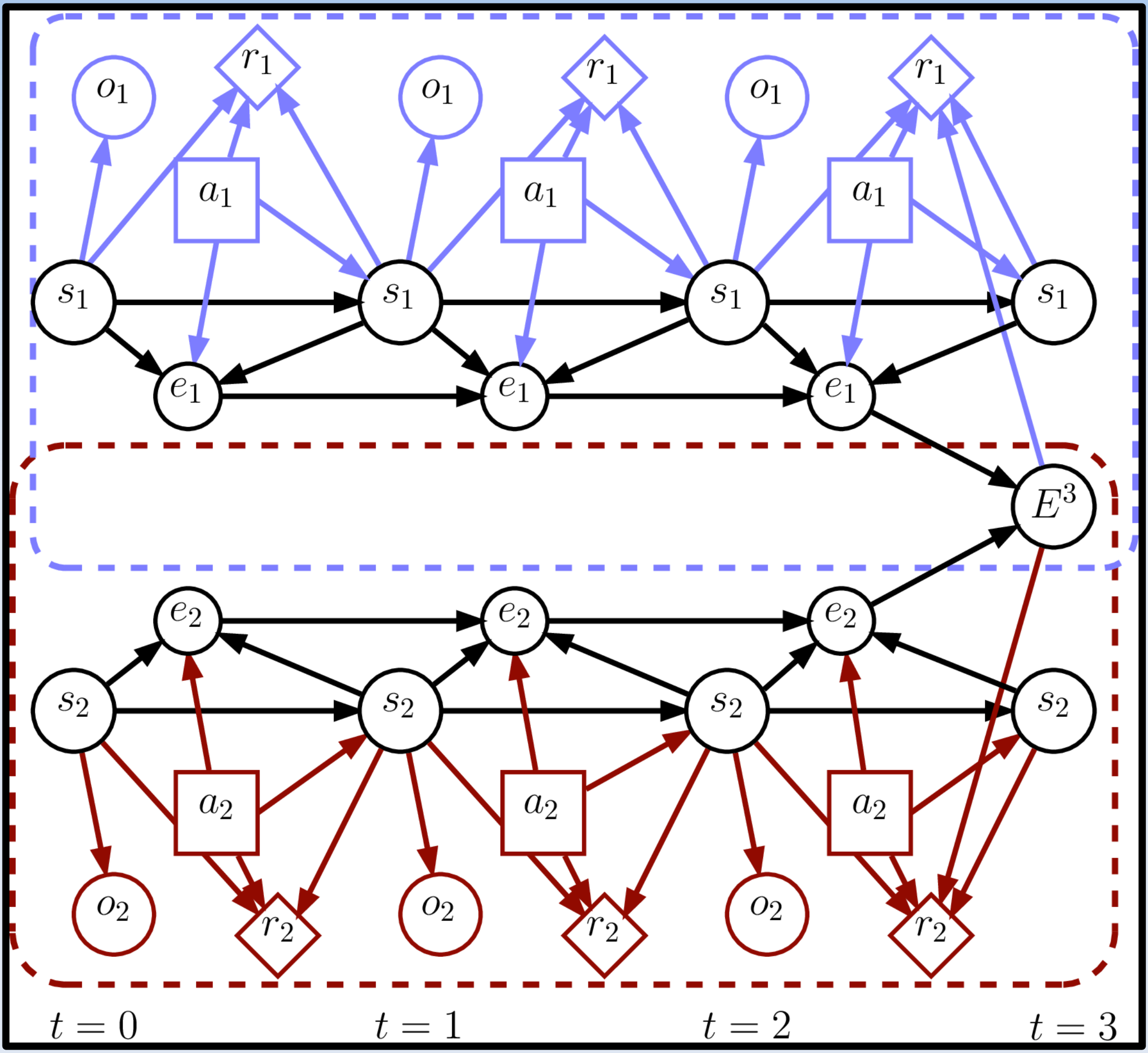
- Example: Mars Rovers [Becker et al. 2004 JAIR]
  - 2 rovers collect data at 4 sites





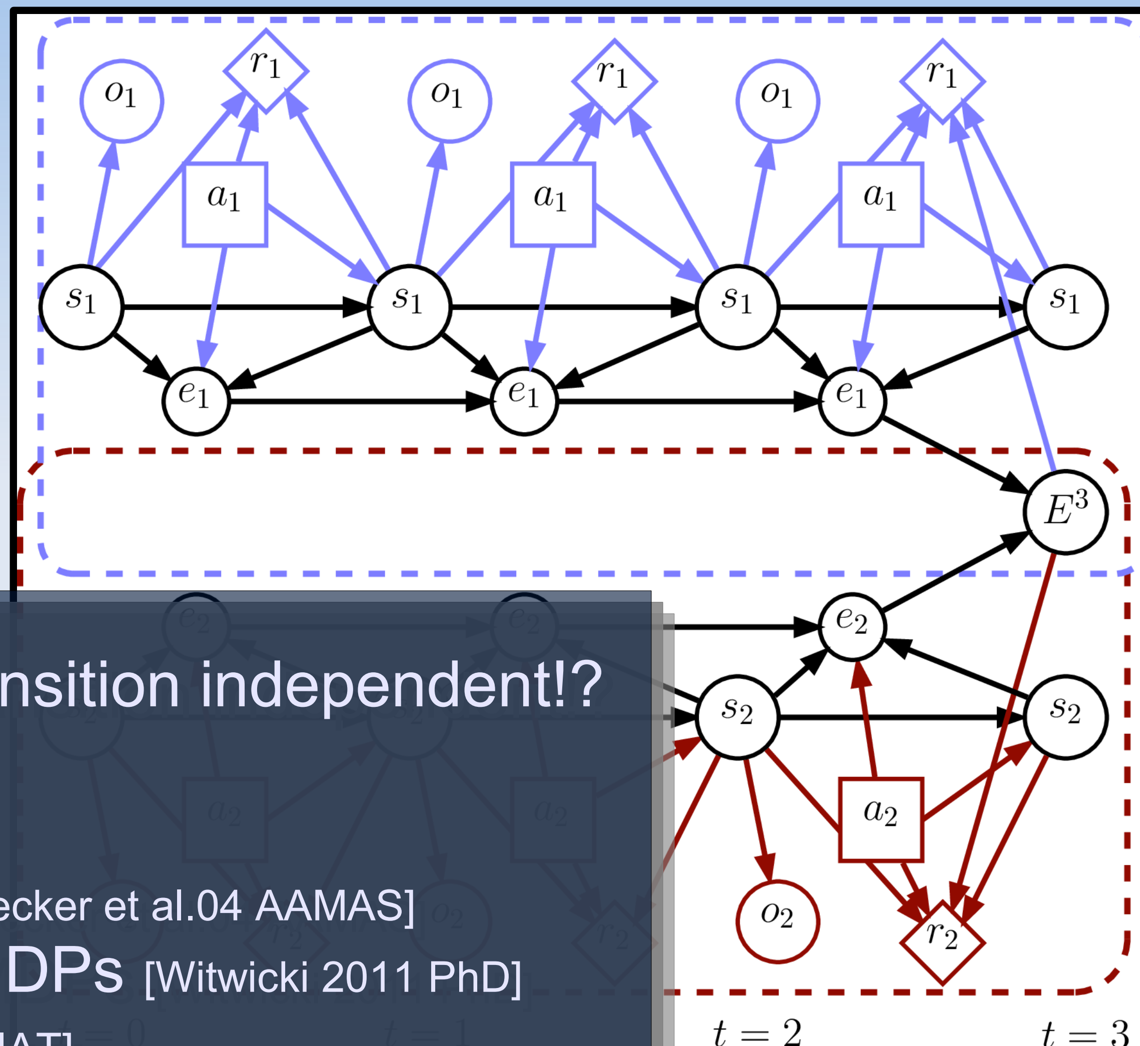
Influence-Based Abstraction

- TI Dec-MDP
- extra reward (or penalty) at the end **if** 'joint event' happens
- joint event $E = \langle e_1, e_2 \rangle$
- From agent i's perspective: if it realizes e_i → extra reward with probability $P(e_j)$



Influence-Based Abstraction

- TI Dec-MDP
- extra reward (or penalty) at the end **if** 'joint event' happens
- joint event *E*=<e_1,e_2>
- From agent i's perspective:



But most problems are not transition independent!?

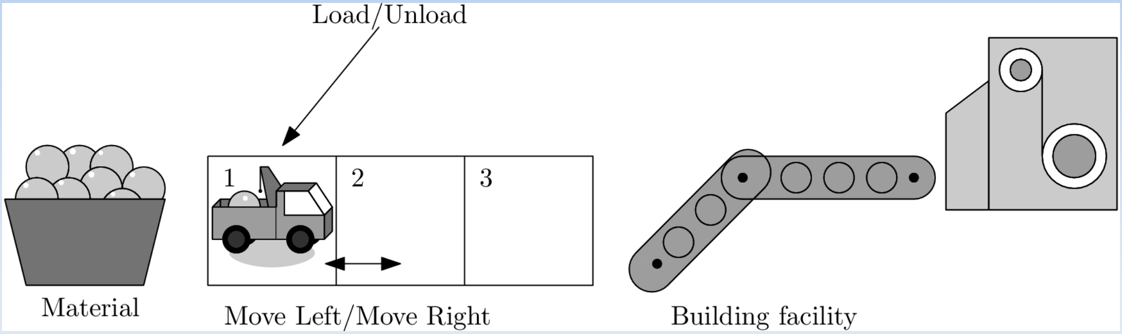
Much further research, e.g.:

- Event-driven Dec-MDPs [Becker et al.04 AAMAS]
- Transition-decoupled POMDPs [Witwicki 2011 PhD]
- EDI-CR [Mostafa & Lesser 2009 WIIAT]
- IBA for Factored POSGs [Oliehoek et al. 2012 AAAI]

Recap: Decision Making under Uncertainty

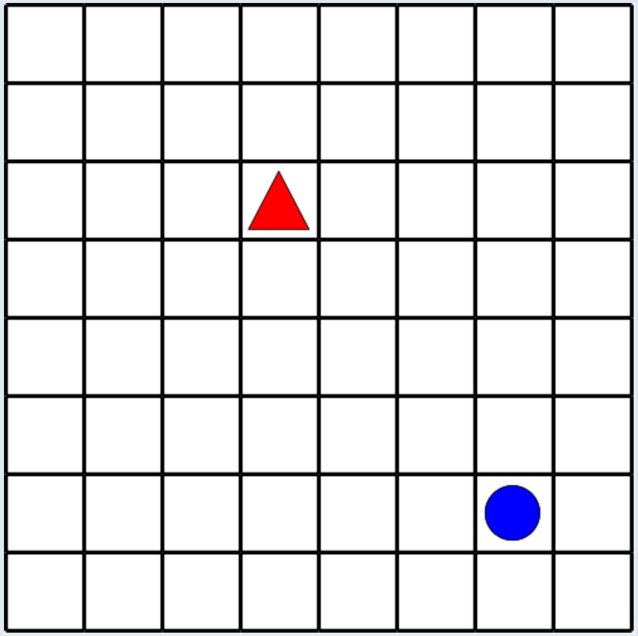
Recap: MDPs

- MDPs:
  - 1 agent
  - perfectly observable
  - outcome uncertainty



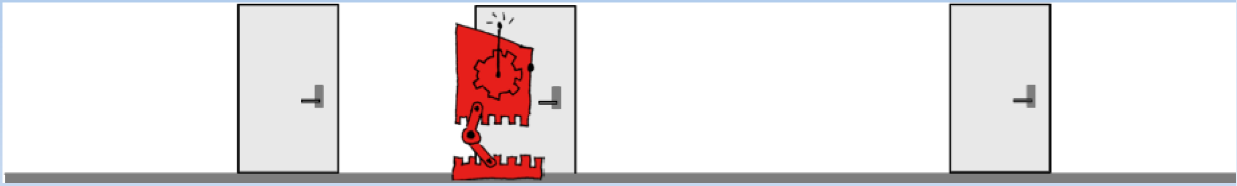
- Bellman equation

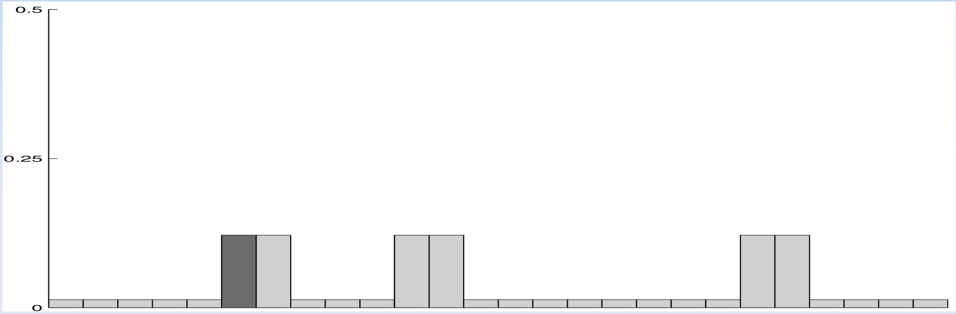
- Value iteration



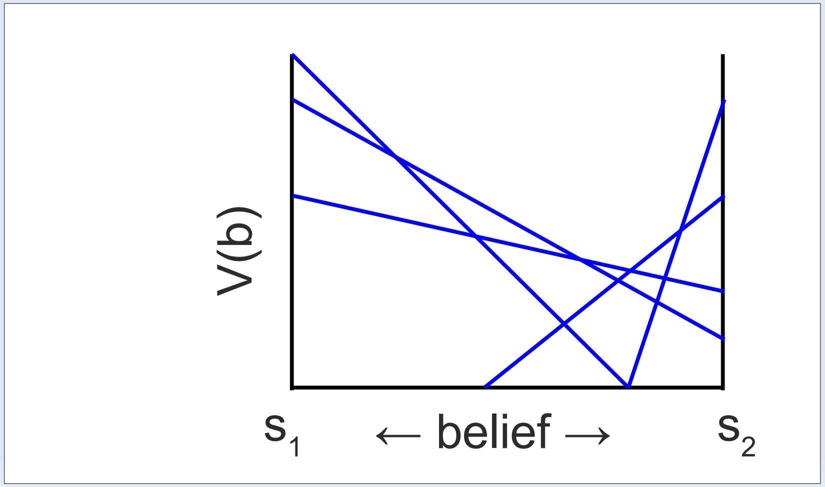
Recap: POMDPs

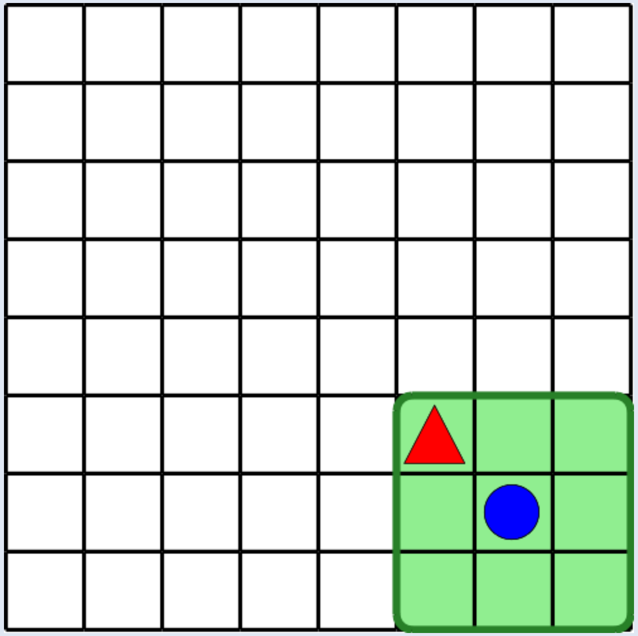
- POMDP
  - 1 agent
  - state uncertainty





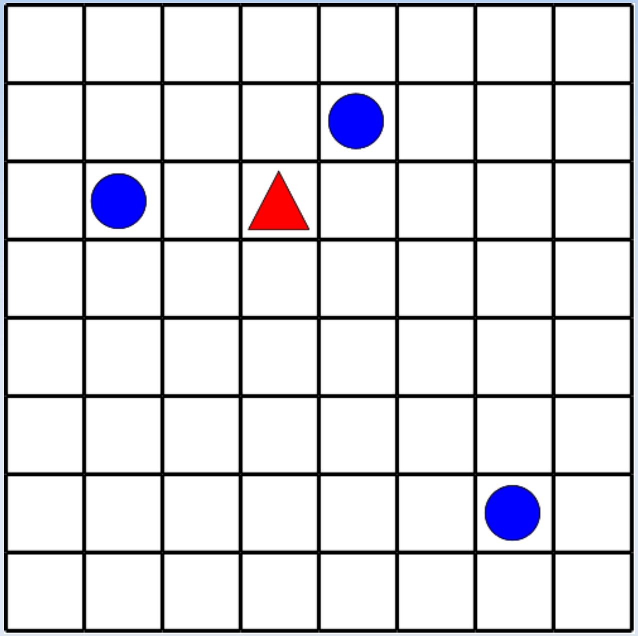
- Reduction: belief-state MDP
  - continuous states
  - vectors for value iteration





Recap: Multiagent MDP

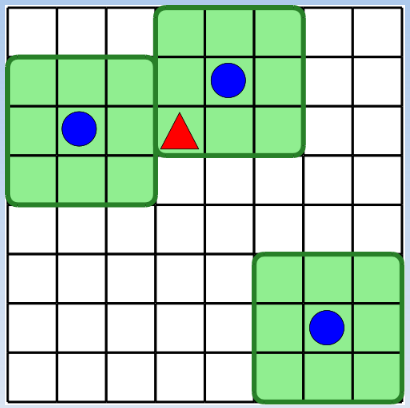
- Multiagent MDP (MMDP)
  - multiple agents
  - outcome uncertainty
  - fully observable



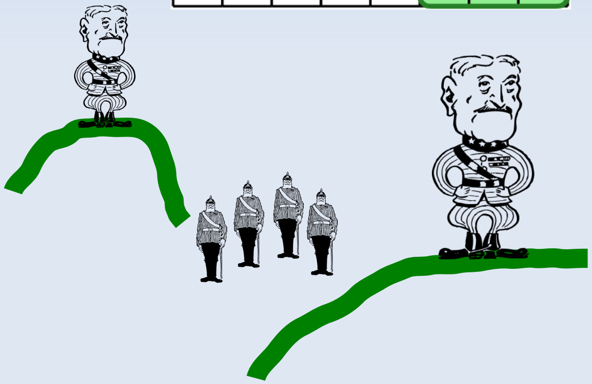
- Reduction to single-agent problem
  - 'puppeteer'
  - value iteration, etc.
  - but exponentially many joint actions – e.g., [Guestrin et al. 2002 NIPS]

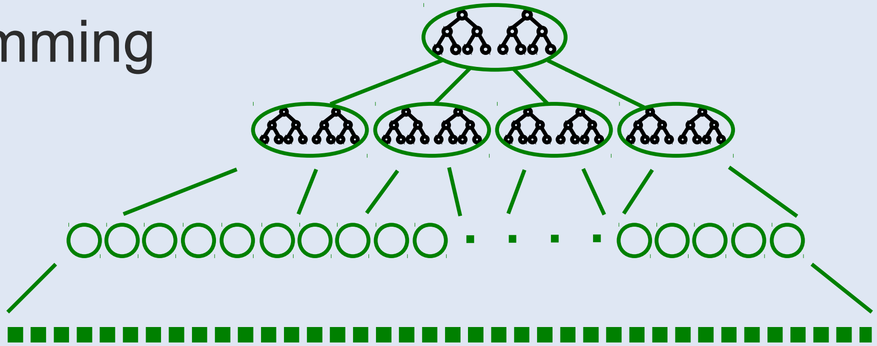
Recap: Partially Observable MAS

- Multiagent POMDP
  - Free communication
  - Reduces to single-agent problem



- Dec-POMDP
  - No (free) communication
  - Harder: NEXP-complete
  - Solution methods:
    - Bottom-up: dynamic programming
    - Top-down: heuristic search





References

- References can be found on the tutorial website:

  www.st.ewi.tudelft.nl/~mtjspaan/tutorialDMuU/

- Further references can be found in

  Frans A. Oliehoek. Decentralized POMDPs. In Wiering, Marco and van Otterlo, Martijn, editors, *Reinforcement Learning: State of the Art*, Adaptation, Learning, and Optimization, pp. 471–503, Springer Berlin Heidelberg, Berlin, Germany, 2012.

  - Available from http://people.csail.mit.edu/fao/