CS 221: Artificial Intelligence

Lecture 8: MDPs

Sebastian Thrun and Peter Norvig

Slide credit: Dan Klein, Stuart Russell

Rhino Museum Tourguide

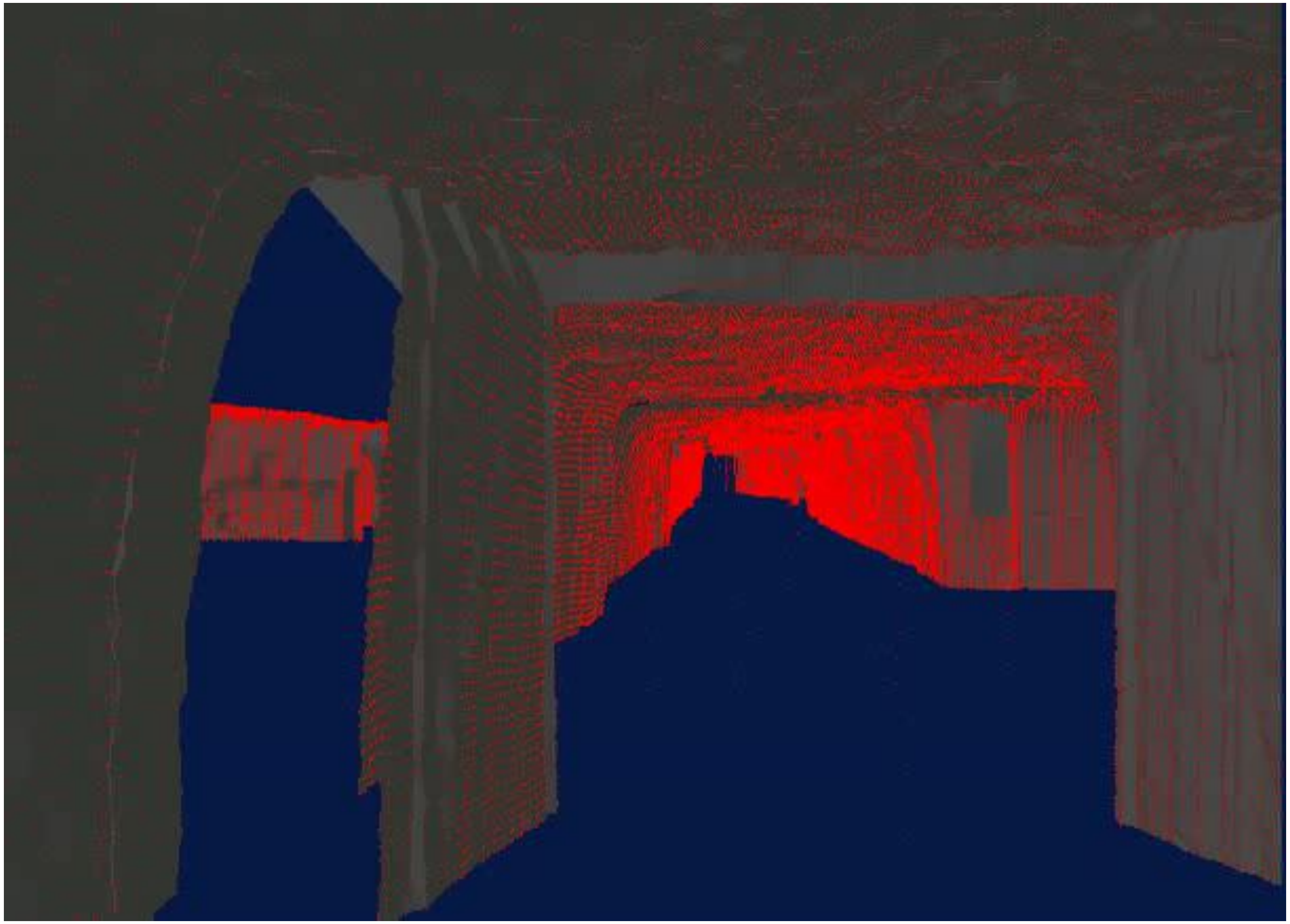
Minerva Robot

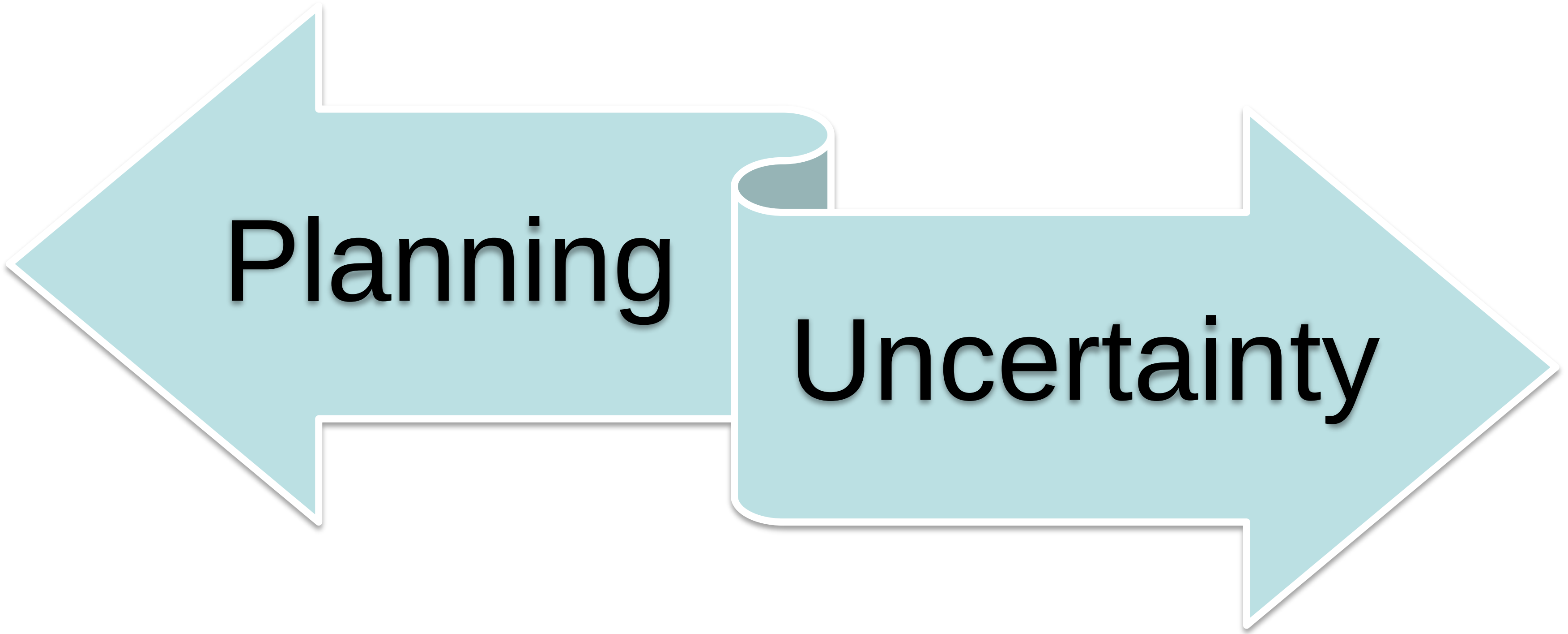
Pearl Robot


Nursebot Pearl
Cocktail Hour at the
Longwood Independent
Living Facility

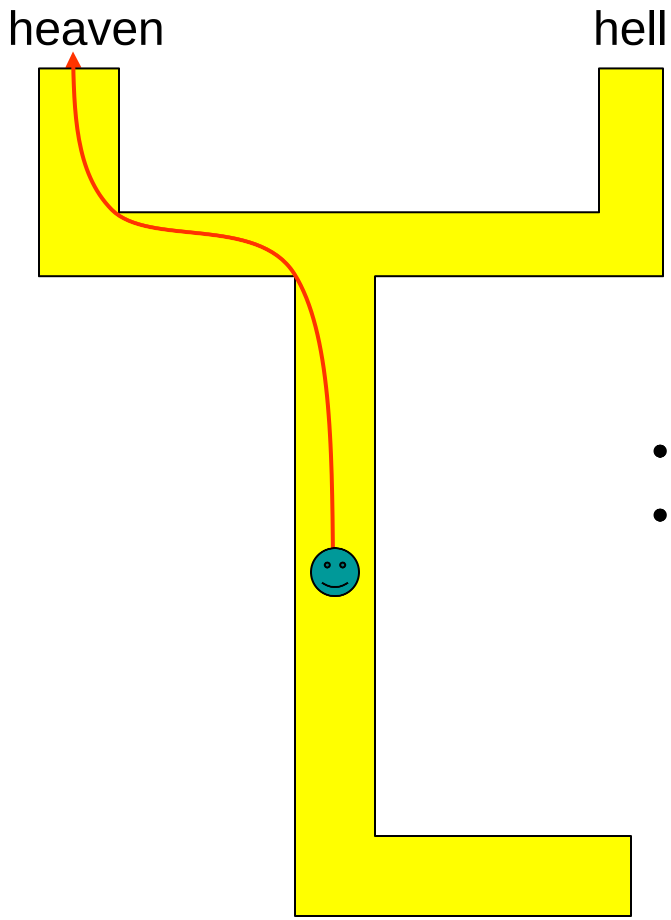
Mine Mapping Robot Groundhog

Mine Mapping Robot Groundhog



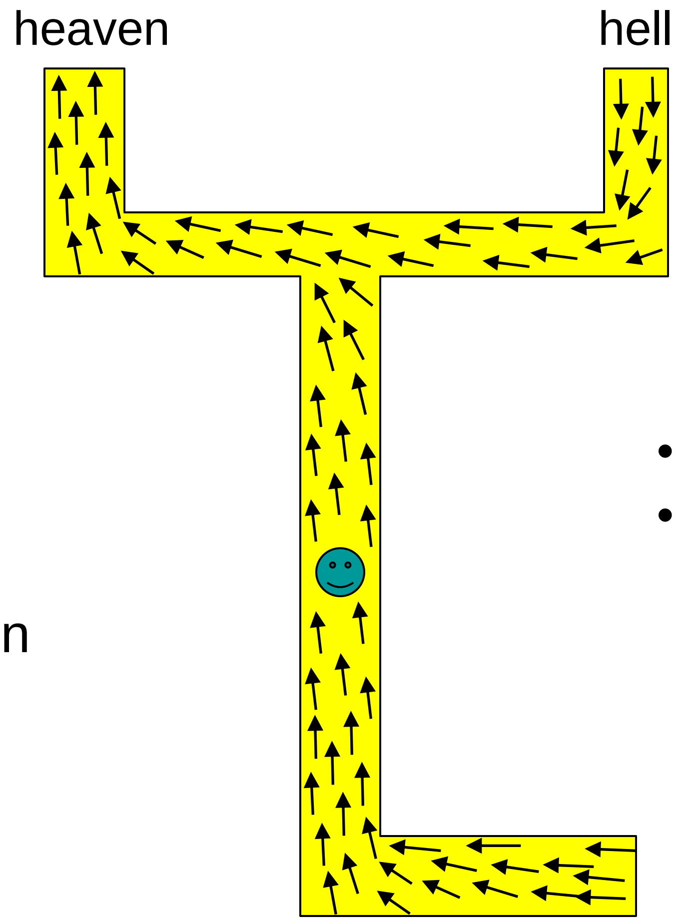

Planning
Uncertainty

Planning: Classical Situation



- World deterministic
- State observable

MDP-Style Planning

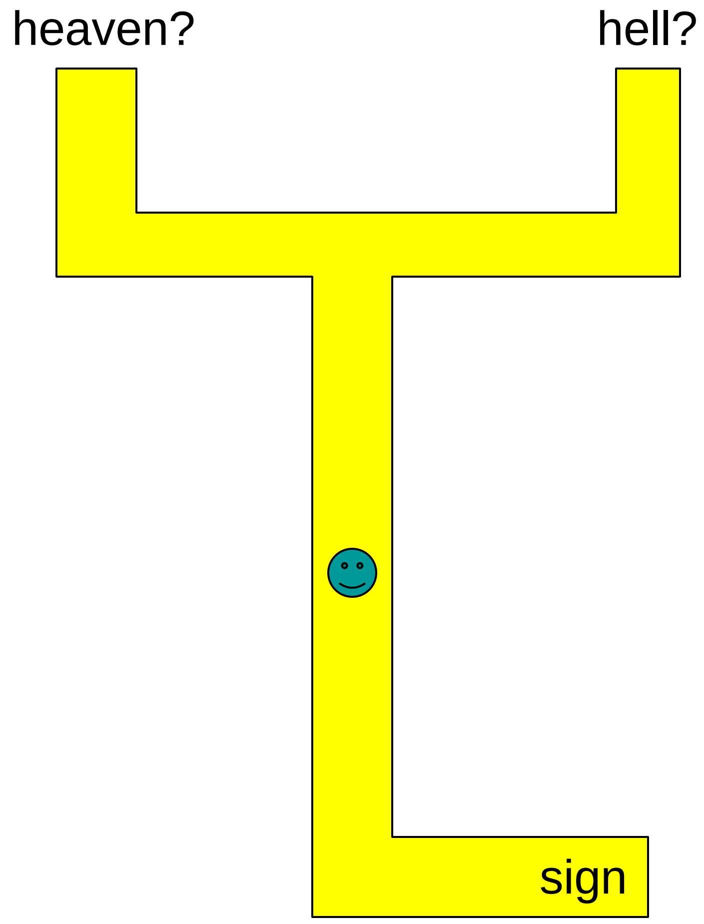


- Policy
- Universal Plan
- Navigation function

- World stochastic
- State observable

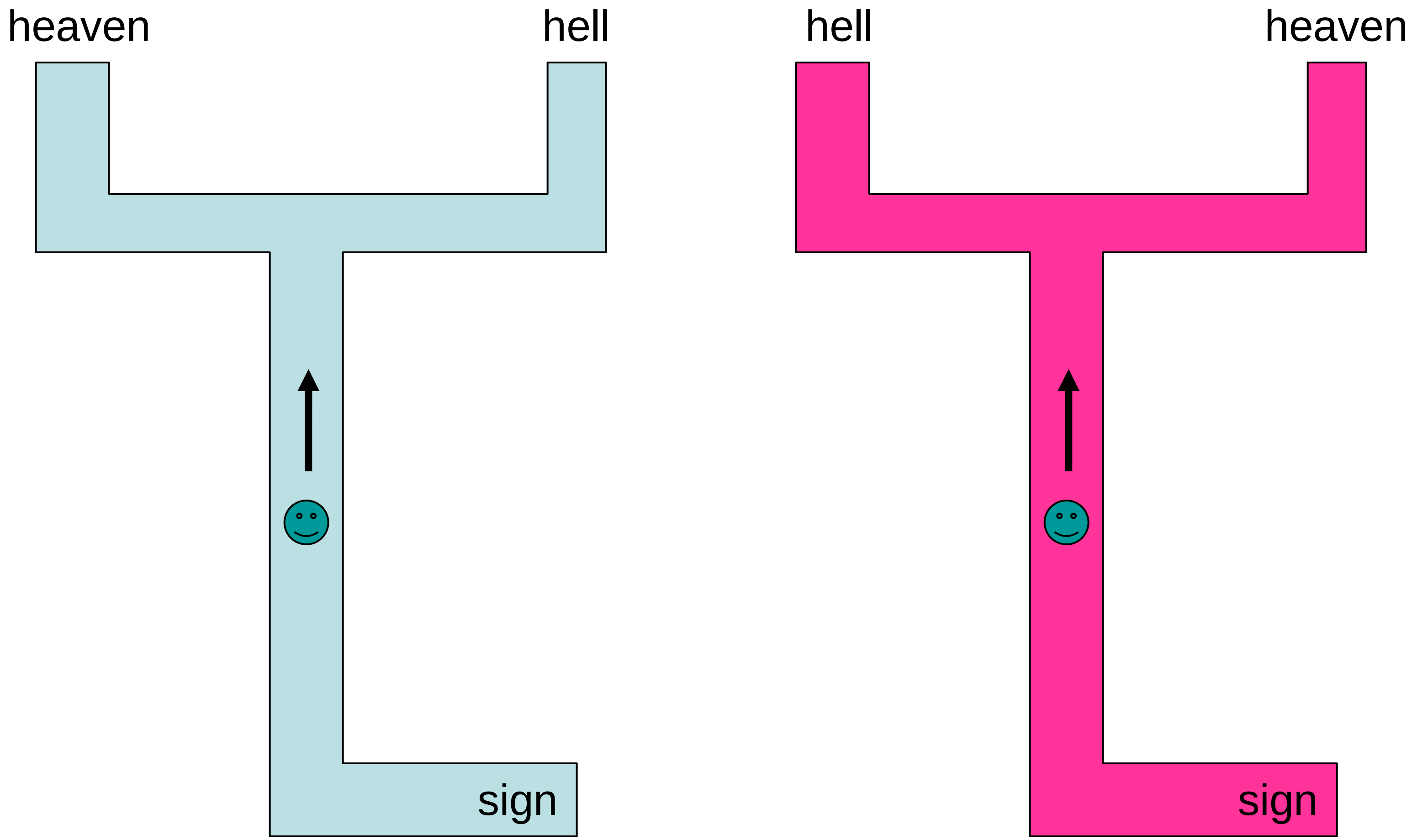
[Koditschek 87, Barto et al. 89]

Stochastic, Partially Observable

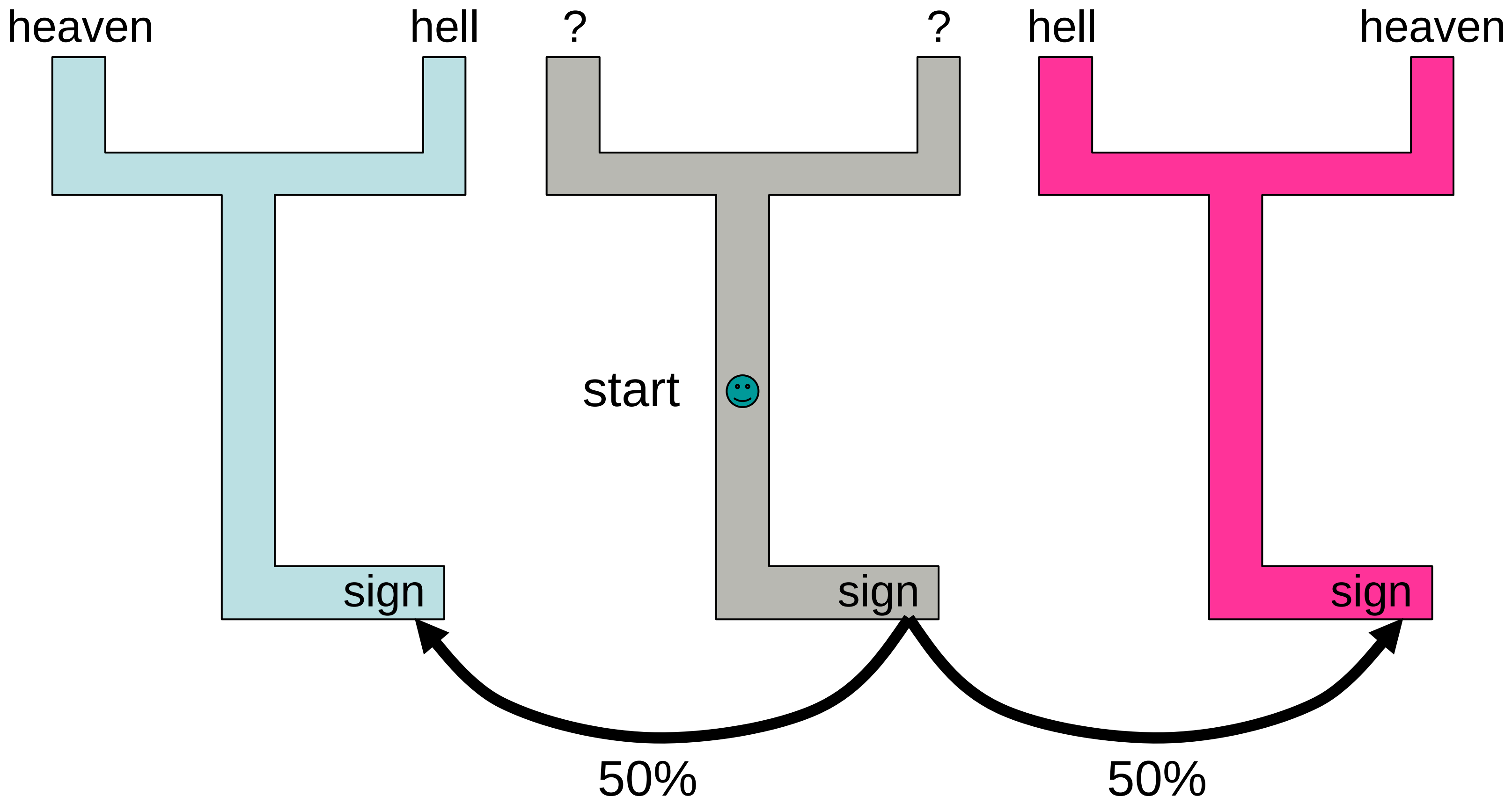


[Sondik 72] [Littman/Cassandra/Kaelbling 97]

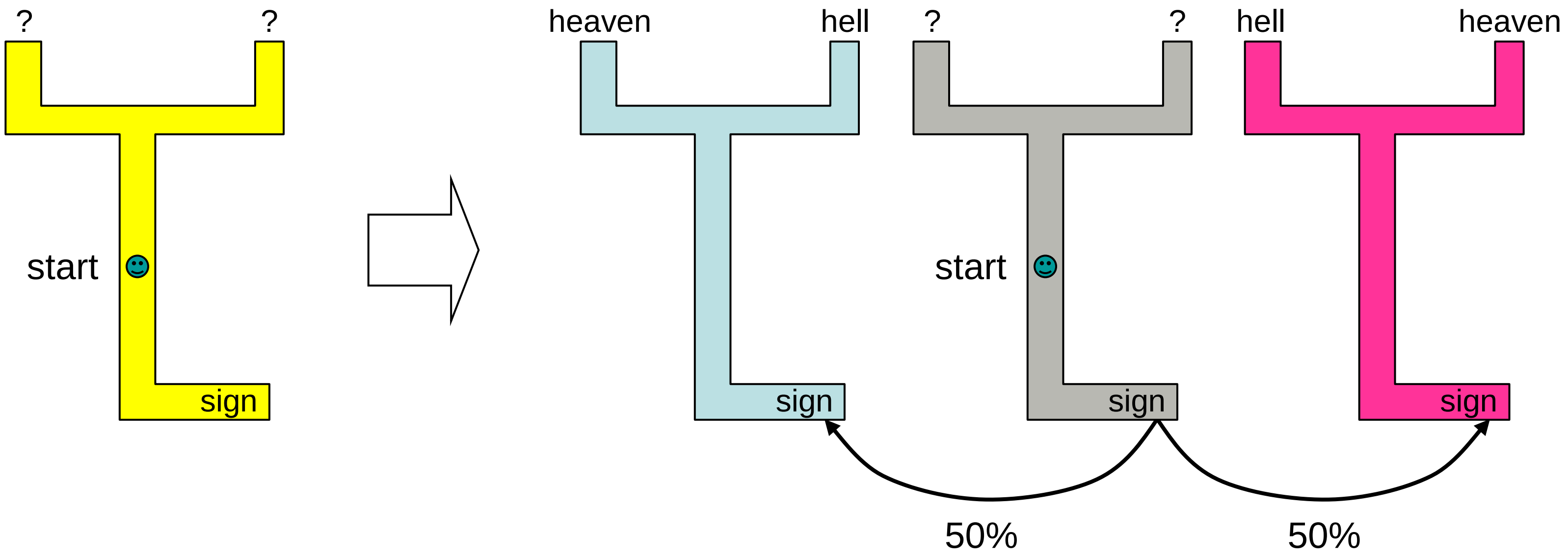
Stochastic, Partially Observable



Stochastic, Partially Observable



Stochastic, Partially Observable



A Quiz

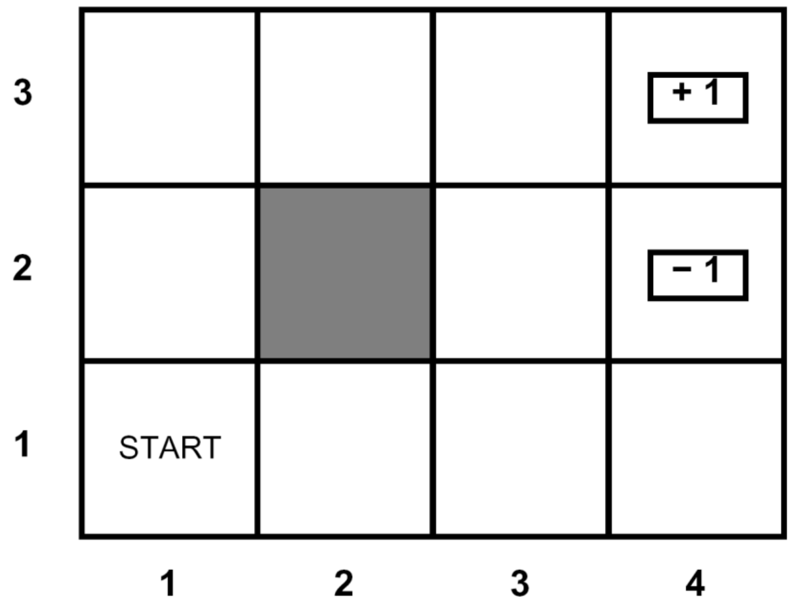
# states	sensors	actions	size belief space?
3	perfect	deterministic	3: s_1, s_2, s_3
3	perfect	stochastic	3: s_1, s_2, s_3
3	abstract states	deterministic	2^3-1: s_1, s_2, s_3, s_{12}, s_{13}, s_{23}, s_{123}
3	stochastic	deterministic	2-dim continuous: $p(S=s_1)$, $p(S=s_2)$
3	none	stochastic	2-dim continuous: $p(S=s_1)$, $p(S=s_2)$
1-dim continuous	stochastic	deterministic	∞-dim continuous
1-dim continuous	stochastic	stochastic	∞-dim continuous
∞-dim continuous	stochastic	stochastic	aargh!

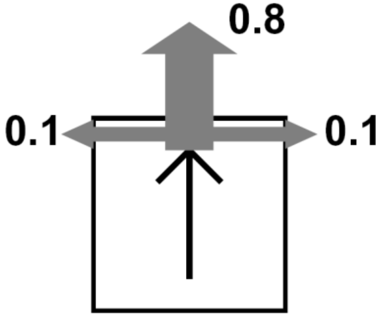
MPD Planning

- Solution for Planning problem
  - Noisy controls
  - Perfect perception
  - Generates “universal plan” (=policy)

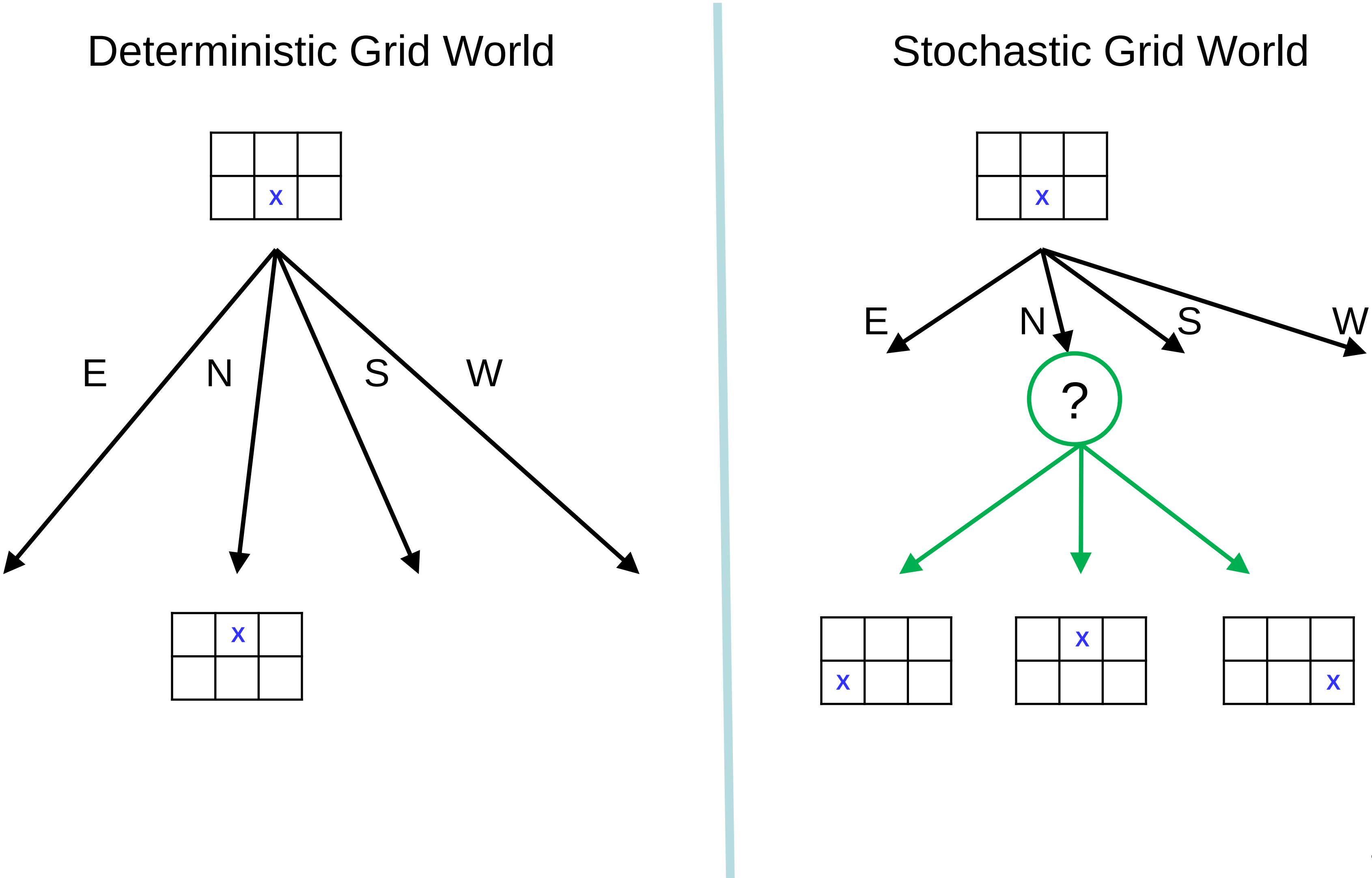
Grid World

- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North
    (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Small "living" reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards*



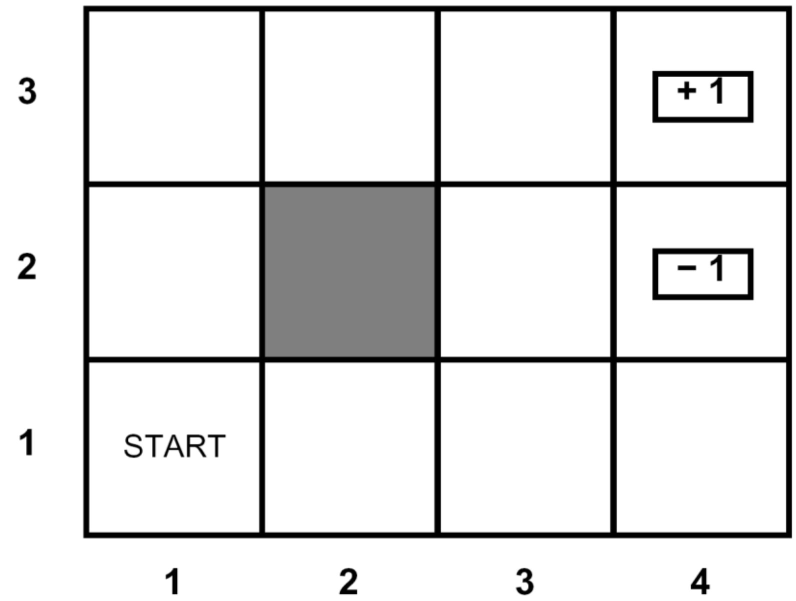


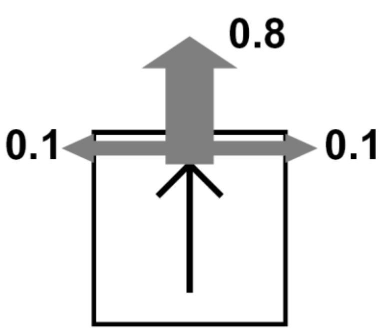
Grid Futures



Markov Decision Processes

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function T(s,a,s')
    - Prob that a from s leads to s'
    - i.e., P(s' | s,a)
    - Also called the model
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state (or distribution)
  - Maybe a terminal state
- MDPs are a family of non-deterministic search problems
  - Reinforcement learning: MDPs where we don't know the transition or reward functions

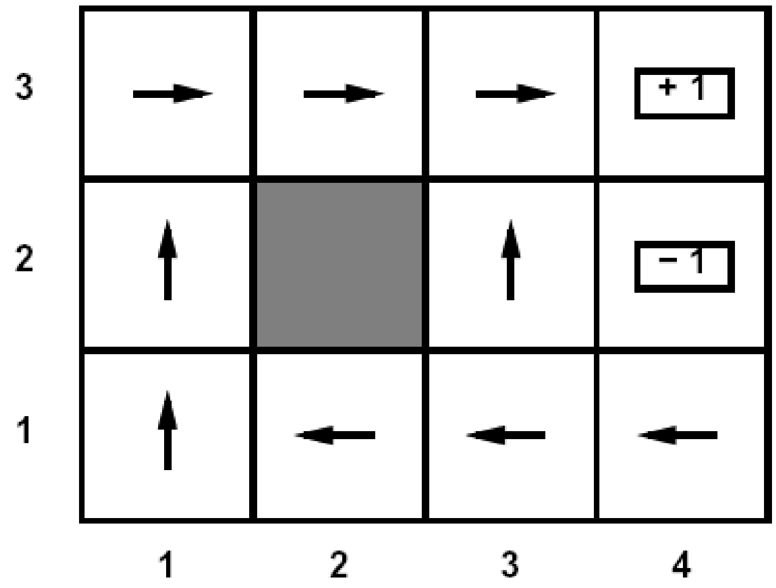




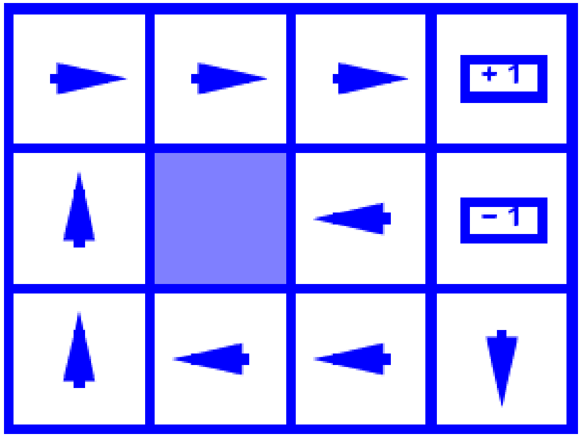
Solving MDPs

- In deterministic single-agent search problems, want an optimal **plan**, or sequence of actions, from start to a goal
- In an MDP, we want an optimal **policy π^*: S → A**
  - A policy π gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent

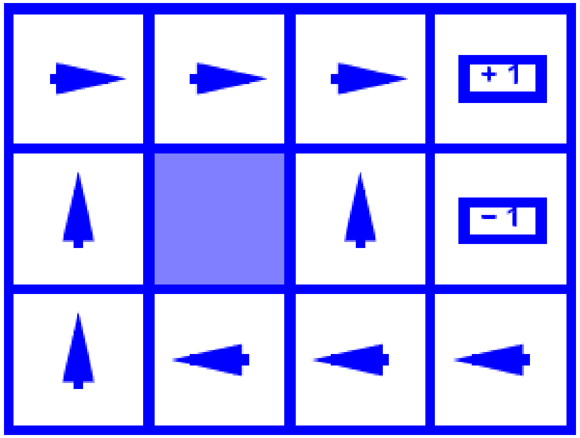
Optimal policy when R(s, a, s') = -0.03 for all non-terminals s



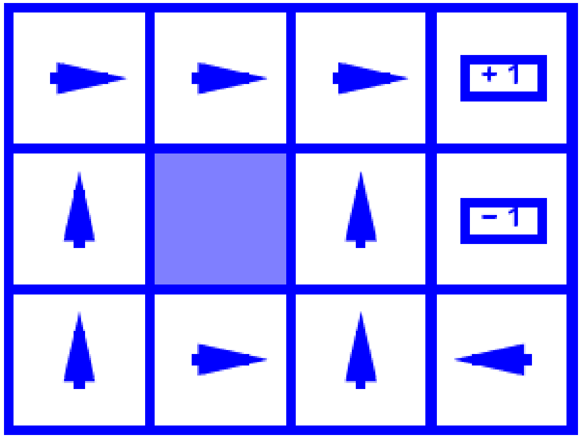
Example Optimal Policies



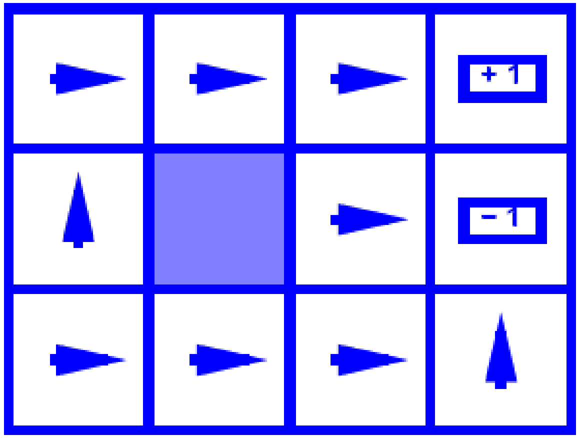
R(s) = -0.01



R(s) = -0.03



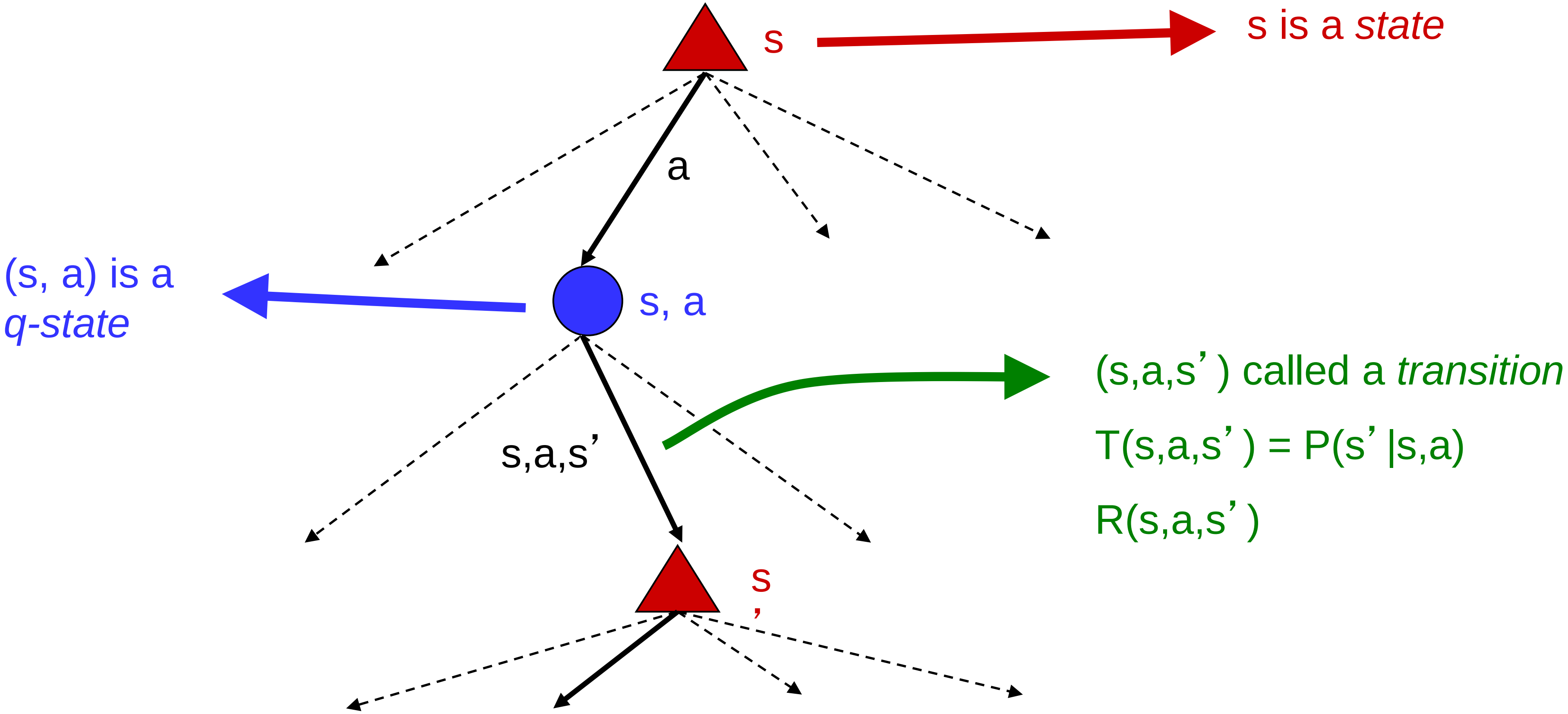
R(s) = -0.4



R(s) = -2.0

MDP Search Trees

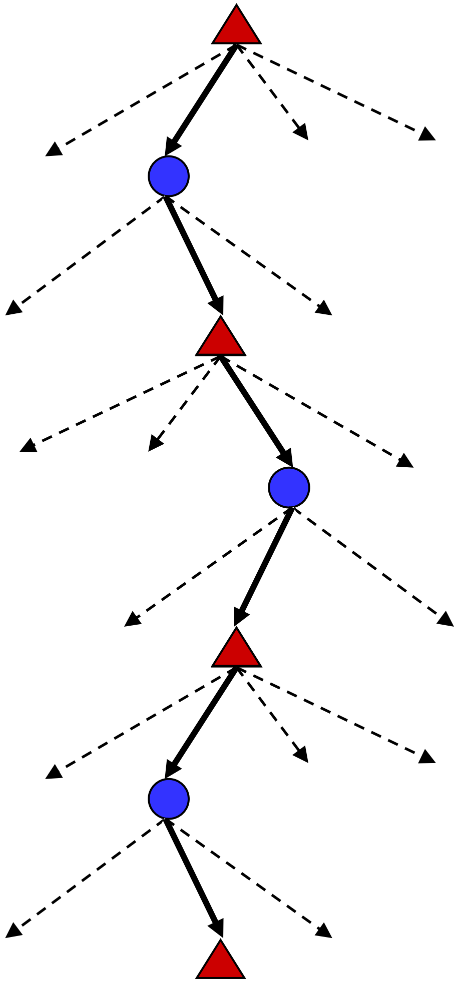
- Each MDP state gives a search tree



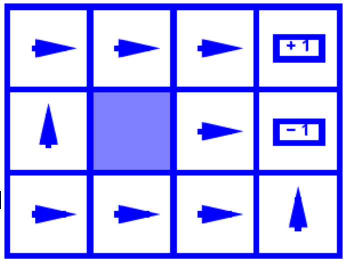
Why Not Search Trees?

- Why not solve with conventional planning?

- Problems:
  - This tree is usually infinite (why?)
  - Same states appear over and over (why?)
  - We would search once per state (why?)



Utilities



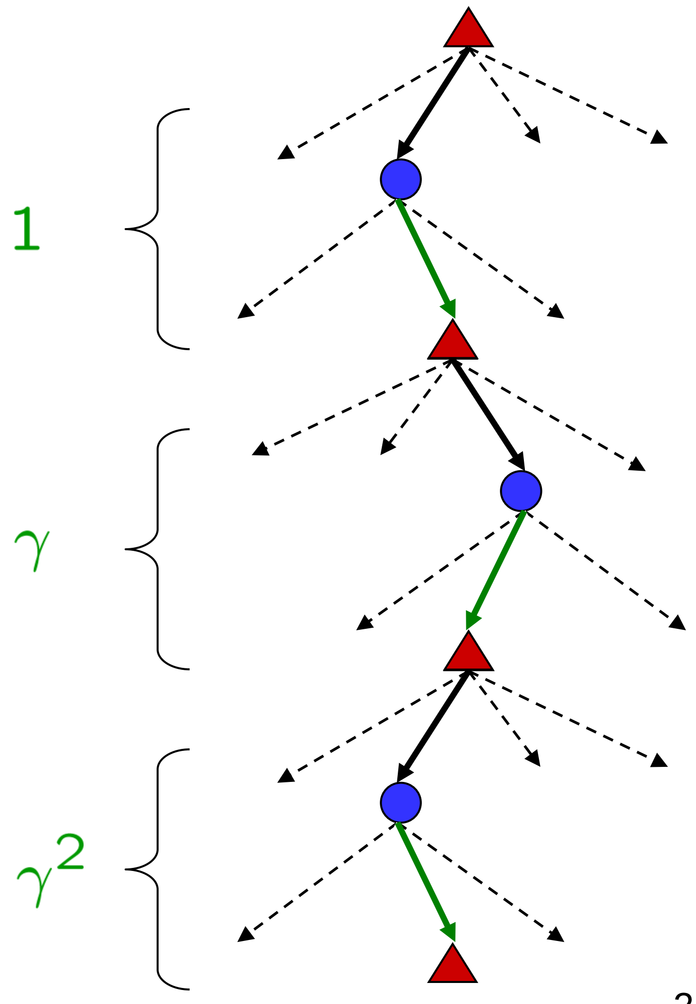
- Utility = sum of future reward
- Problem: infinite state sequences have infinite rewards
- Solutions:
  - Finite horizon:
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "done" for High-Low)
  - Discounting: for $0 < \gamma < 1$

$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

    - Smaller γ means smaller "horizon" – shorter term focus

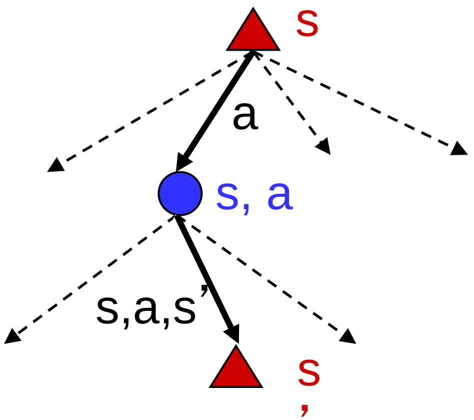
Discounting

- Typically discount rewards by $\gamma < 1$ each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge



Recap: Defining MDPs

- Markov decision processes:
  - States S
  - Start state s_0
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)



- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards

Optimal Utilities

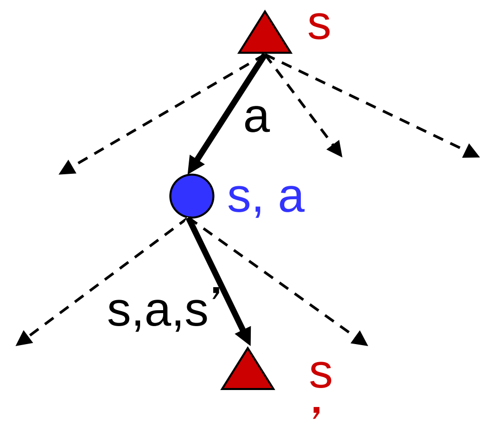
- Fundamental operation: compute the values (optimal expect-max utilities) of states s

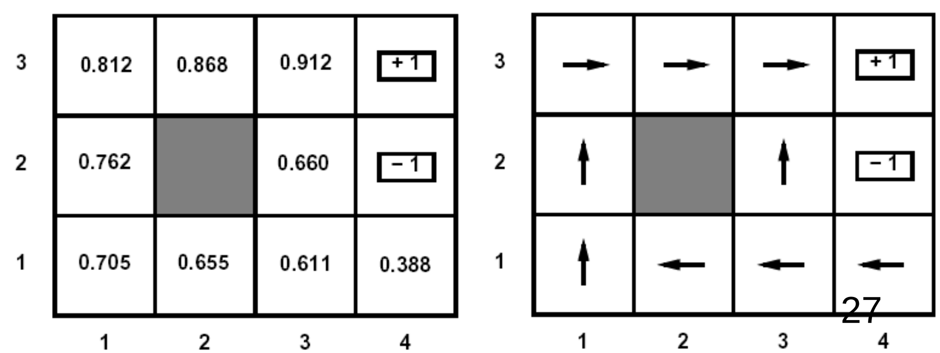
- Why? Optimal values define optimal policies!

- Define the value of a state s:
  $V^*(s)$ = expected utility starting in s and acting optimally

- Define the value of a q-state (s,a):
  $Q^*(s,a)$ = expected utility starting in s, taking action a and thereafter acting optimally

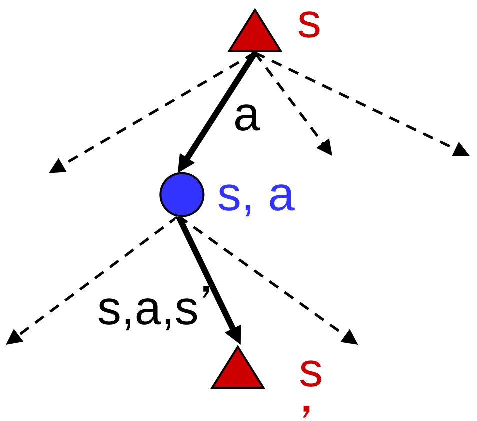
- Define the optimal policy:
  $\pi^*(s)$ = optimal action from state s





The Bellman Equations

- Definition of "optimal utility" leads to a simple one-step lookahead relationship amongst optimal utility values:



Optimal rewards = maximize over first action and then follow optimal policy

- Formally:

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

Solving MDPs

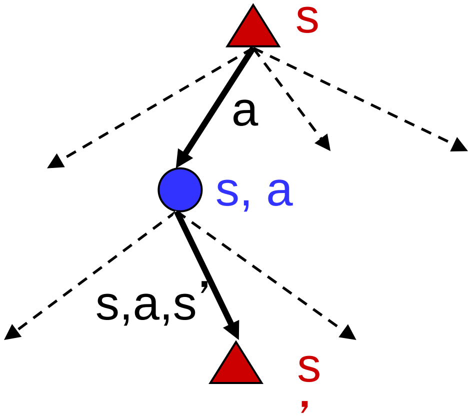
- We want to find the optimal policy π^*

- Proposal 1: modified expect-max search, starting from each state s:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a Q^*(s,a)$$



Value Iteration

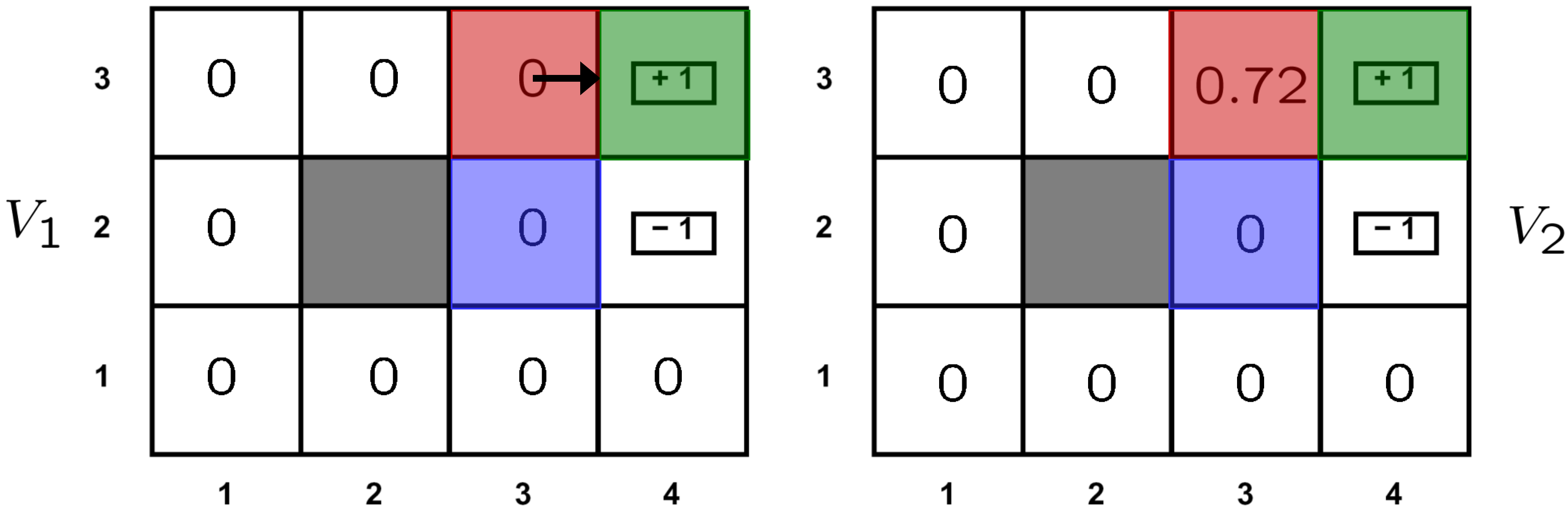
- Idea:
  - Start with $V_0^*(s) = 0$
  - Given V_i^*, calculate the values for all states for depth i+1:

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right]$$

  - This is called a value update or Bellman update
  - Repeat until convergence
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Example: γ=0.9, living reward=0, noise=0.2

Example: Bellman Updates



$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma\, V_i(s') \right]$$

$$V_2(\langle 3,3\rangle) = \sum_{s'} T(\langle 3,3\rangle, \text{right}, s') \left[R(\langle 3,3\rangle) + 0.9\, V_1(s') \right]$$

max happens for a=right, other actions not shown

$$= 0.9\, [0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0]$$

Example: Value Iteration

V_2

3	0	0	0.72	+1
2	0		0	−1
1	0	0	0	0
	1	2	3	4

V_3

3	0	0.52	0.78	+1
2	0		0.43	−1
1	0	0	0	0
	1	2	3	4

- Information propagates outward from terminal states and eventually all states have correct value estimates

Computing Actions

- Which action should we chose from state s:
  - Given optimal values V?

$$\arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

  - Given optimal q-values Q?

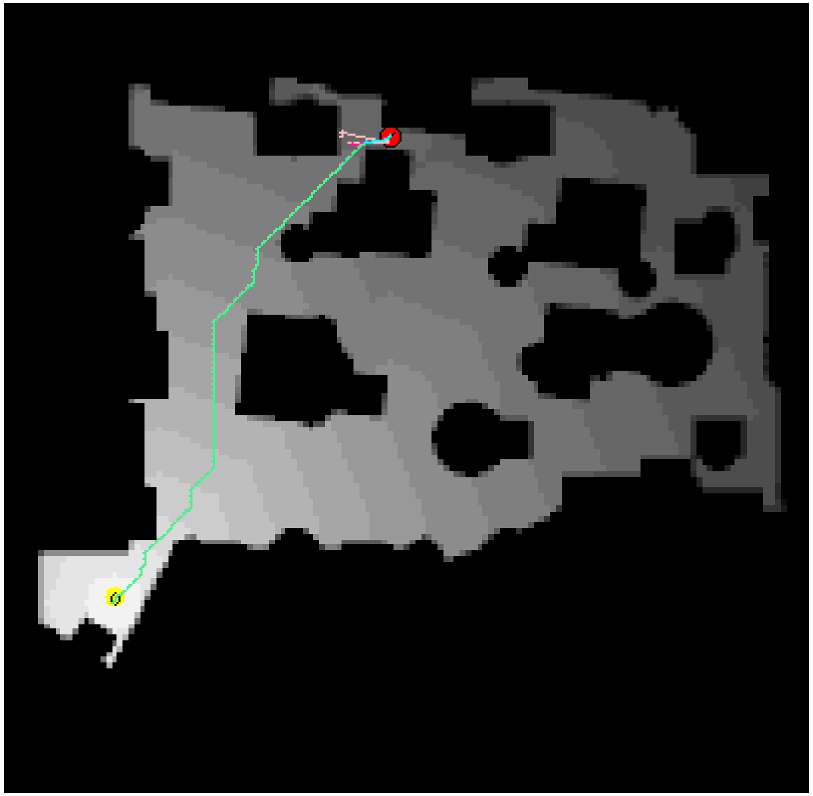
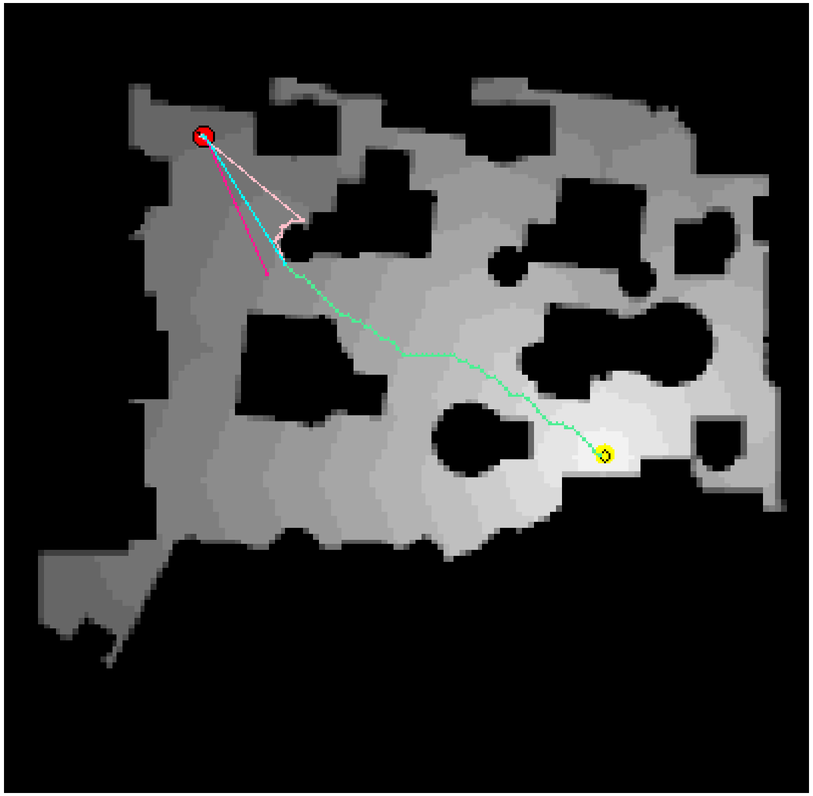
$$\arg\max_a Q^*(s,a)$$

  - Lesson: actions are easier to select from Q's!

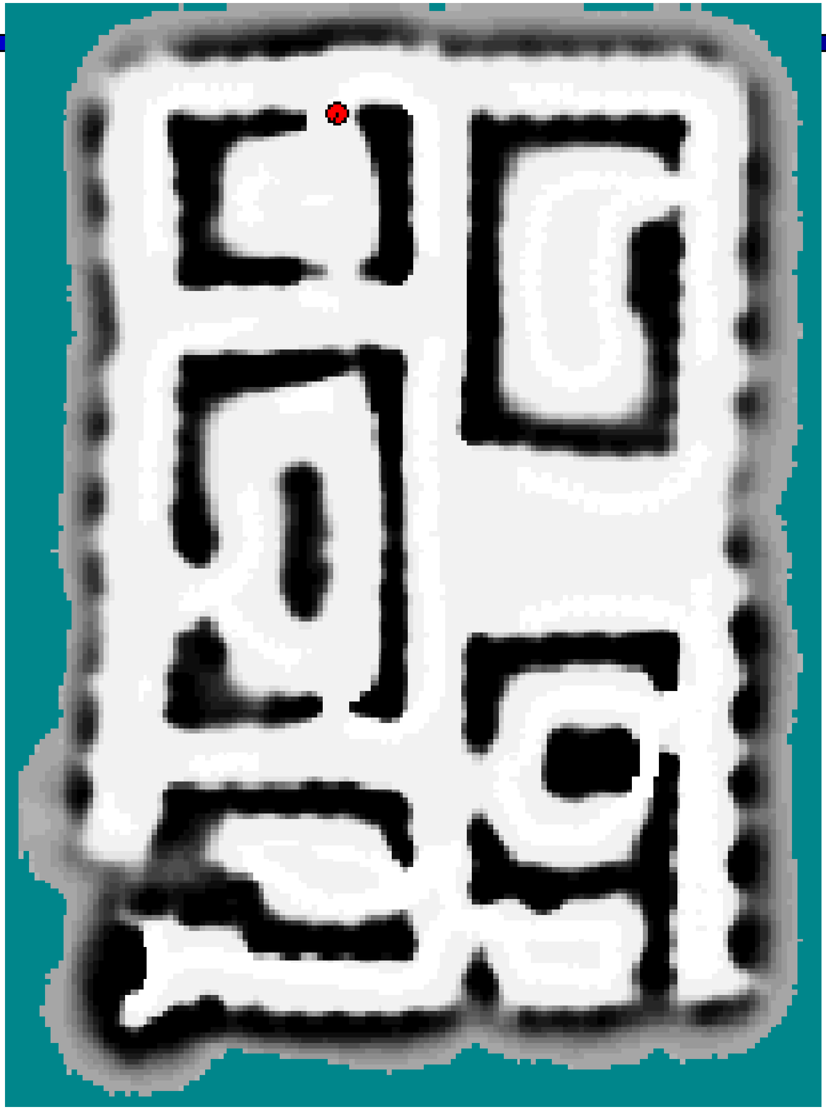
Asynchronous Value Iteration*

- In value iteration, we update every state in each iteration
- Actually, *any* sequences of Bellman updates will converge if every state is visited infinitely often
- In fact, we can update the policy as seldom or often as we like, and we will still converge
- Idea: Update states whose value we expect to change:
  If $|V_{i+1}(s) - V_i(s)|$ is large then update predecessors of s

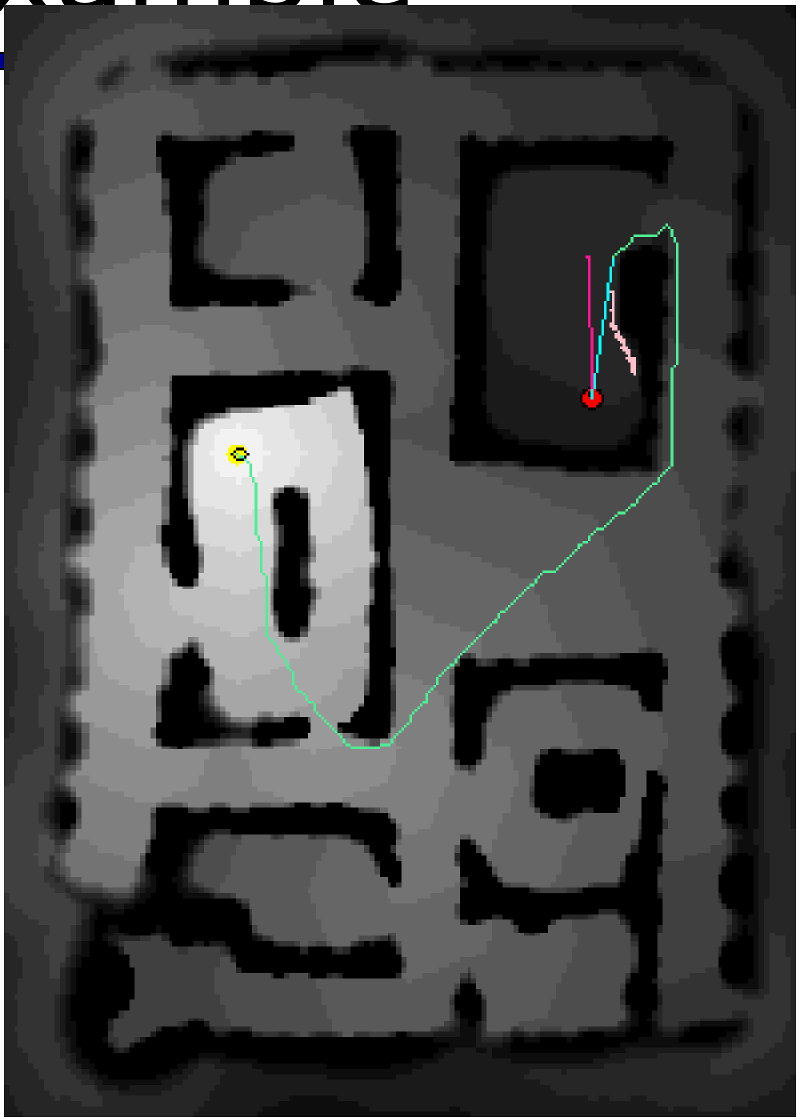
Value Iteration: Example

Another Example

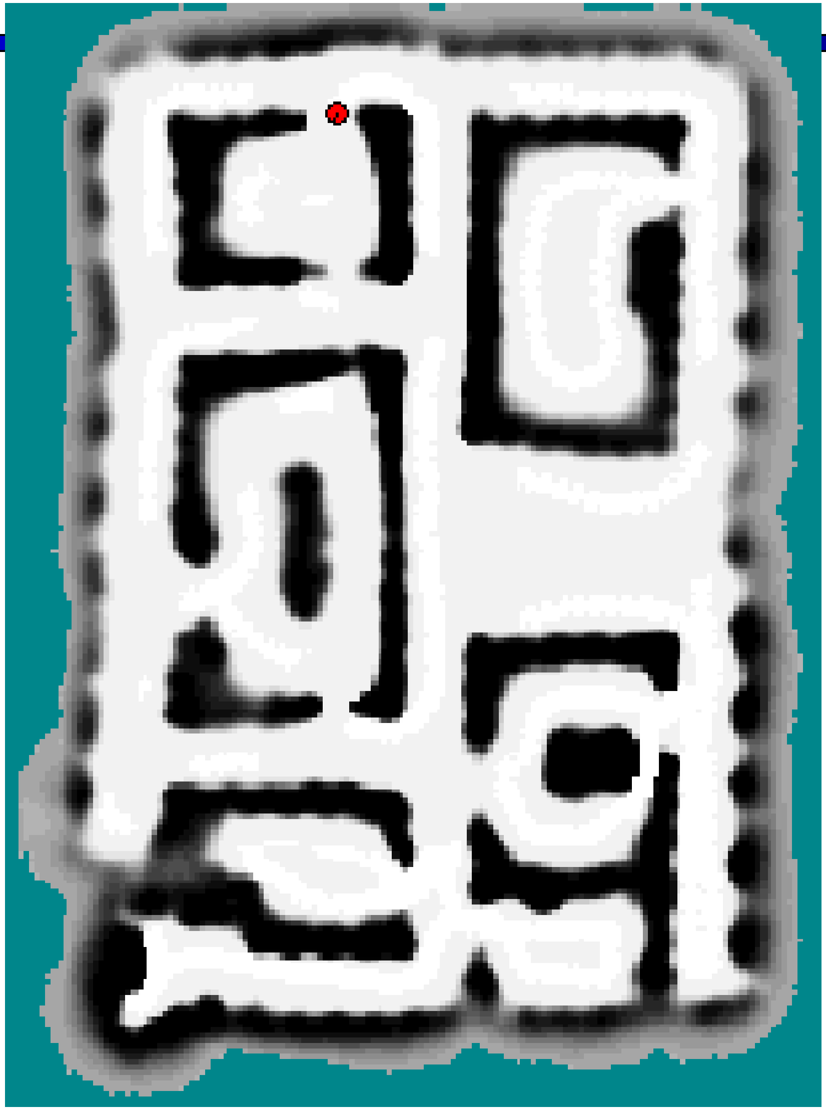


Map

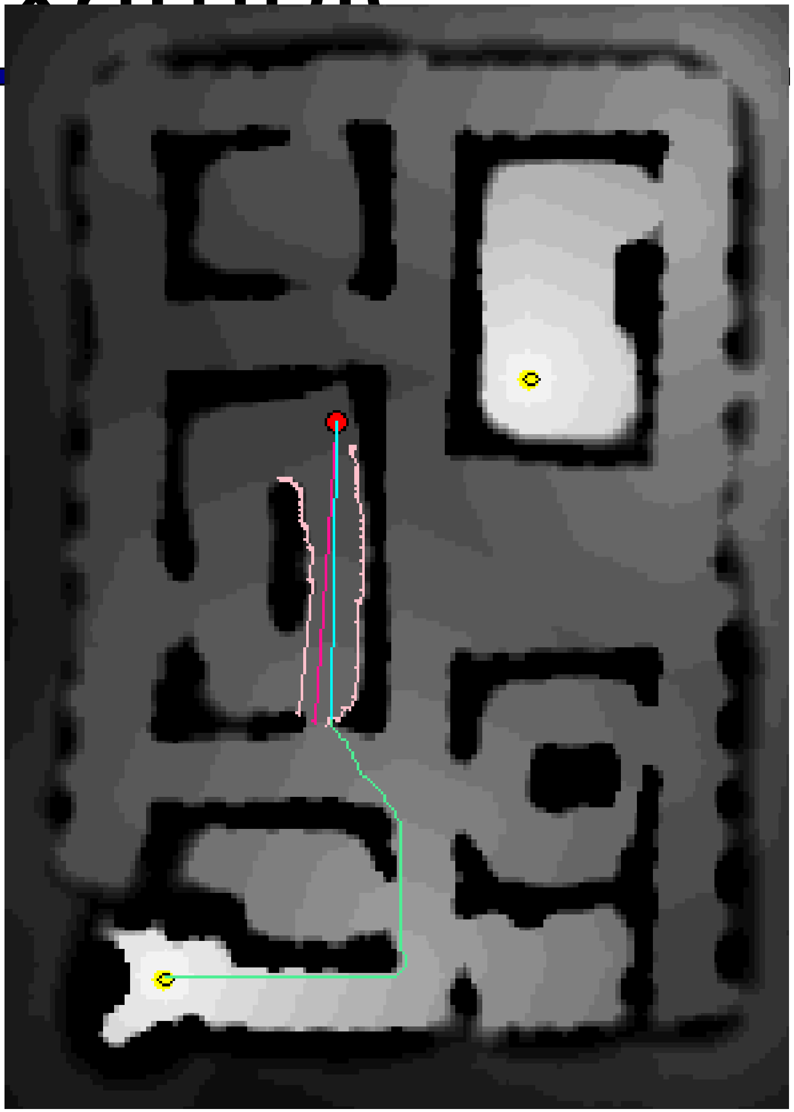


Value Function and Plan

Another Example

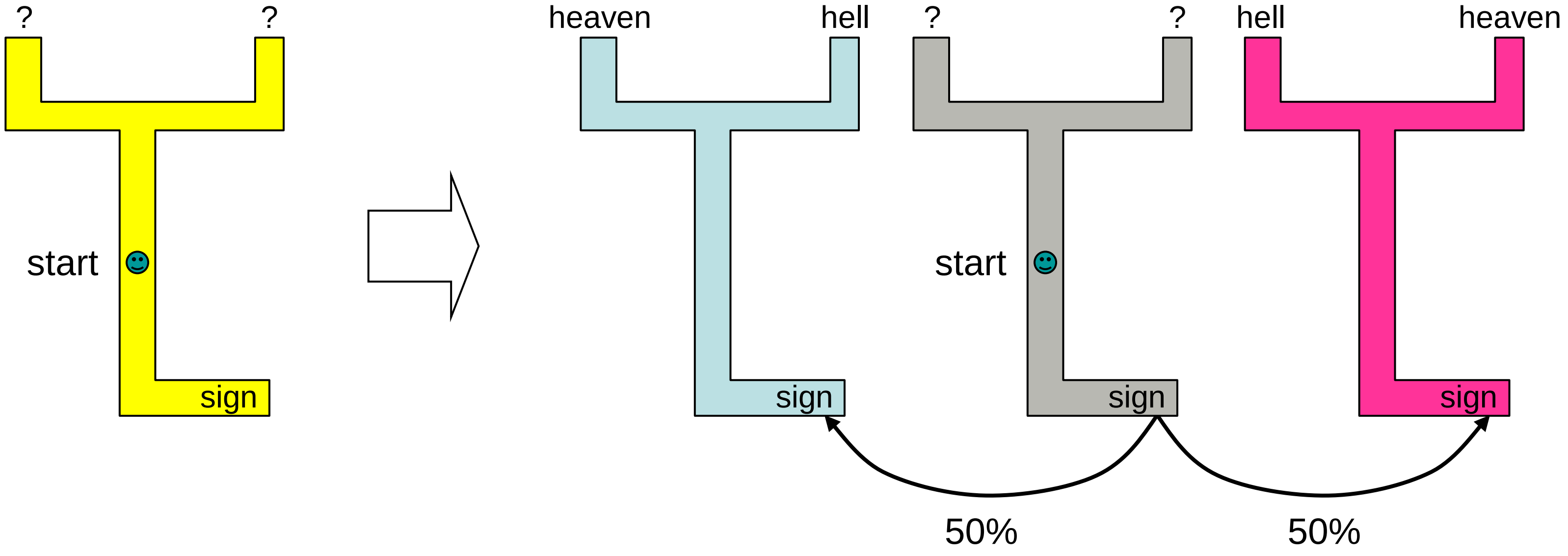


Map



Value Function and Plan

Stochastic, Partially Observable

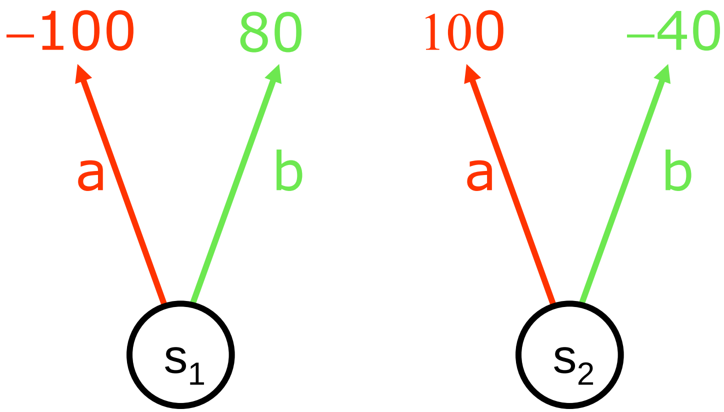


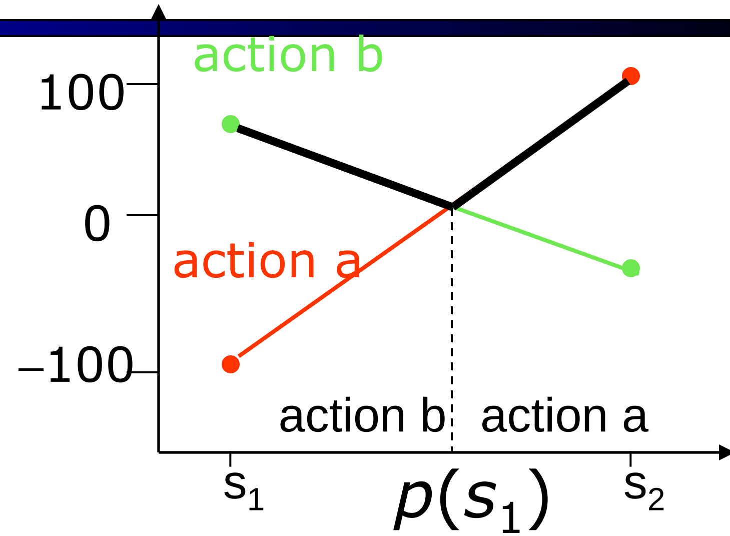
Value Iteration in Belief space: POMDPs

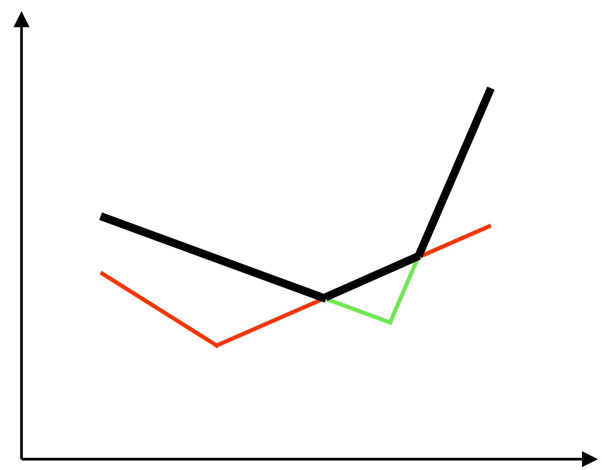
- Partially Observable Markov Decision Process
  - Known model (learning even harder!)
  - Observation uncertainty
  - Usually also: transition uncertainty
  - Planning in belief space = space of all probability distributions

  - Value function: Piecewise linear, convex function over the belief space

Introduction to POMDPs (1 of 3)

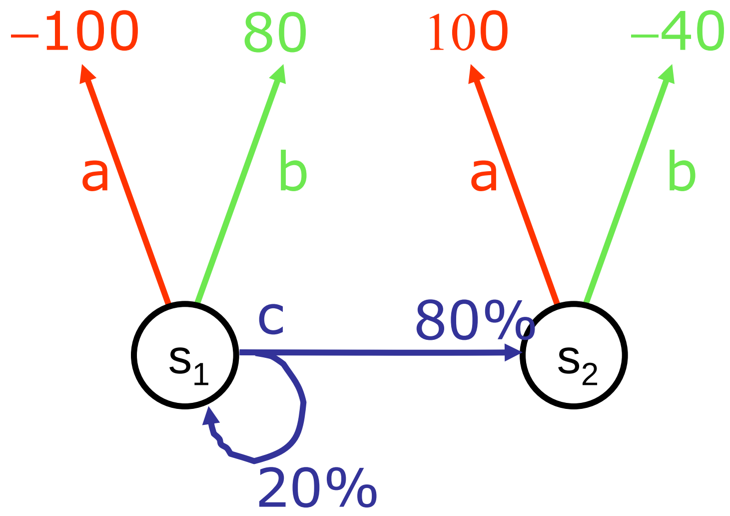


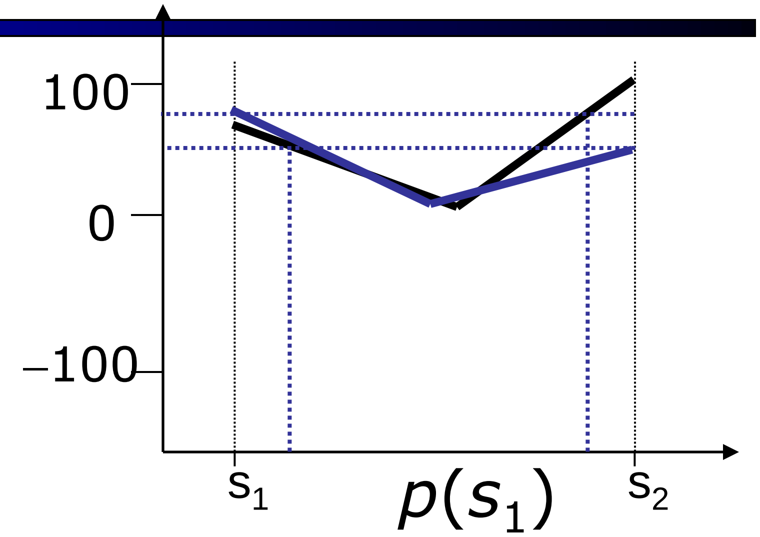


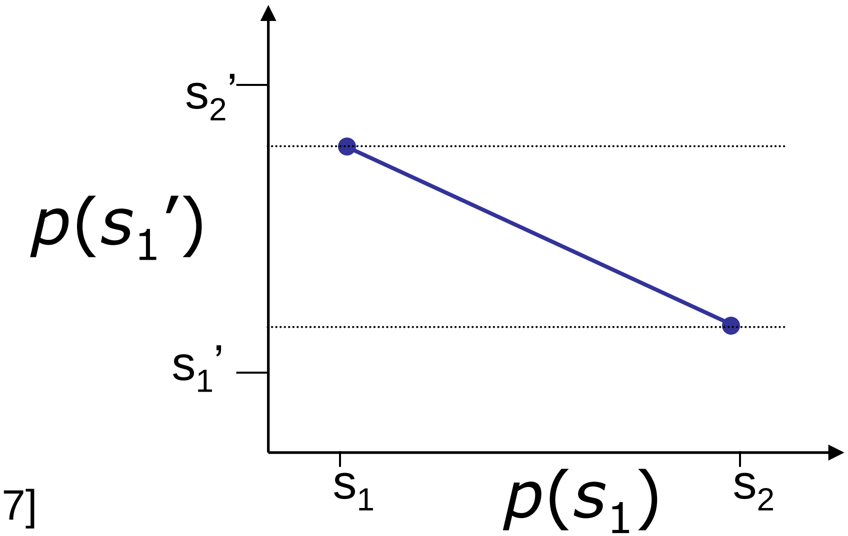


[Sondik 72, Littman, Kaelbling, Cassandra ‘97]

Introduction to POMDPs (2 of 3)

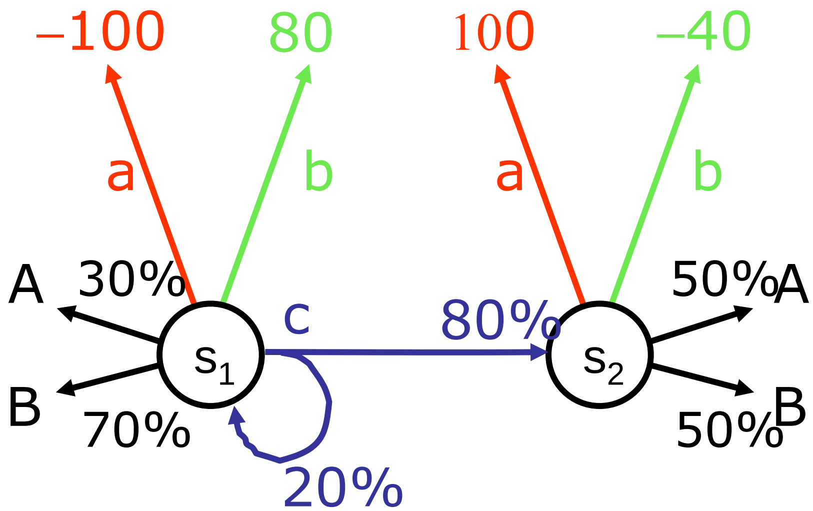


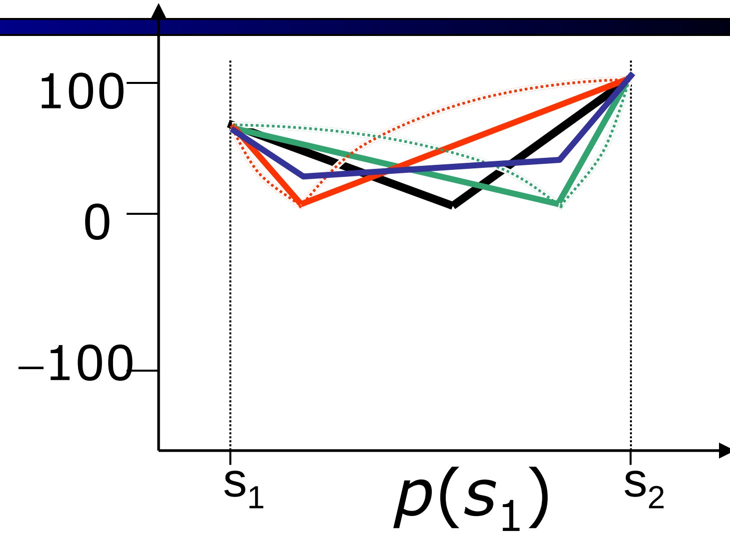


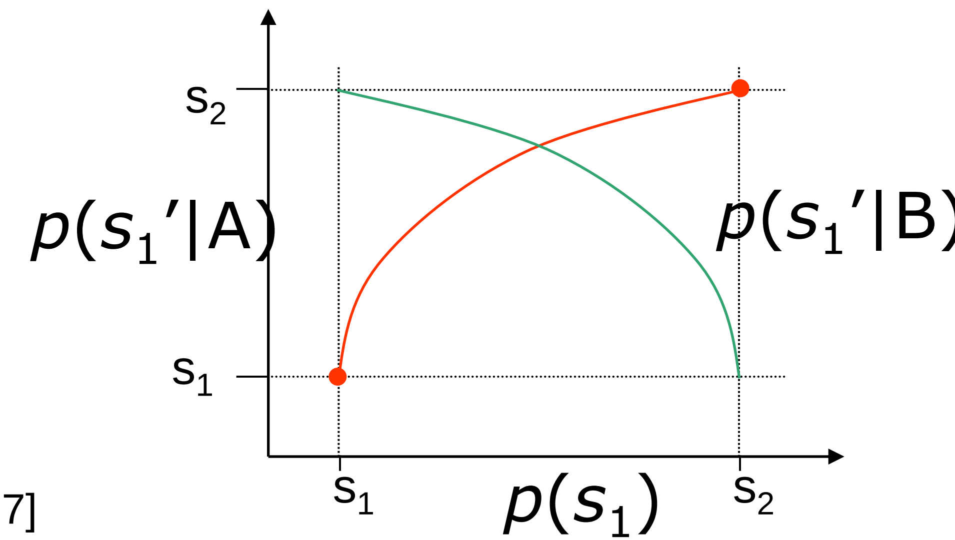


[Sondik 72, Littman, Kaelbling, Cassandra '97]

Introduction to POMDPs (3 of 3)







$$V(p(s_1)) = \sum_{z=\{A,B\}} V(p(s_1 \mid z))\, p(z)$$

[Sondik 72, Littman, Kaelbling, Cassandra ‘97]

POMDP Algorithm

- Belief space = Space of all probability distribution (continuous)
- Value function: Max of set of linear functions in belief space
- Backup: Create new linear functions

- Number of linear functions can grow fast!

Why is This So Complex?

State Space Planning
(no state uncertainty)

Belief Space Planning
(full state uncertainties)

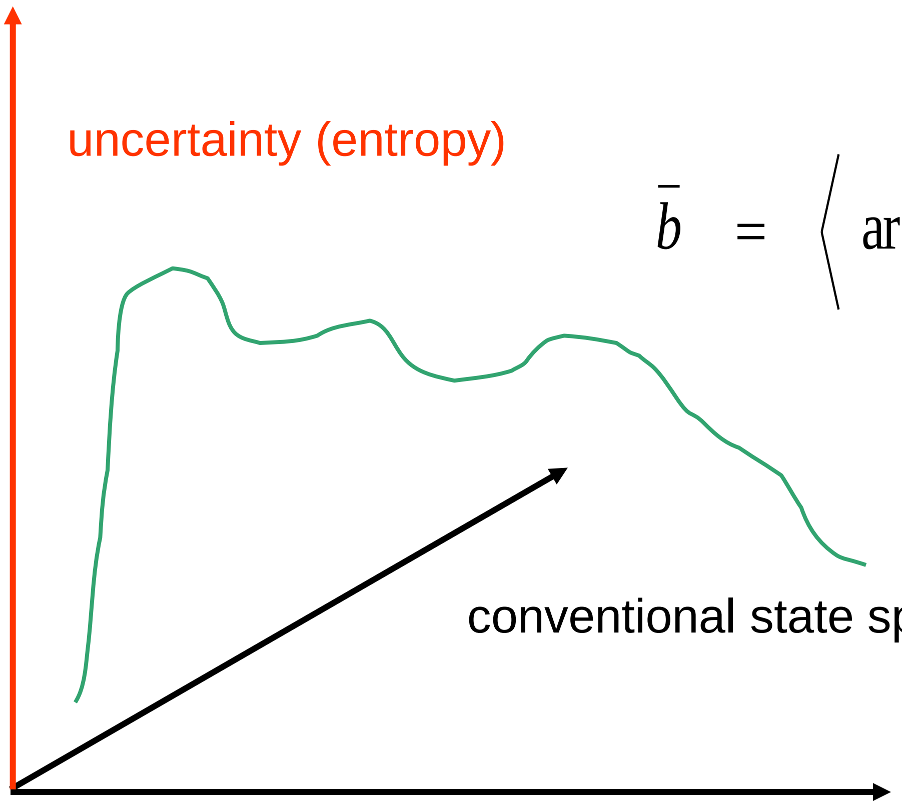
?

Belief Space Structure

The controller may be globally uncertain...

but not usually.

Augmented MDPs:

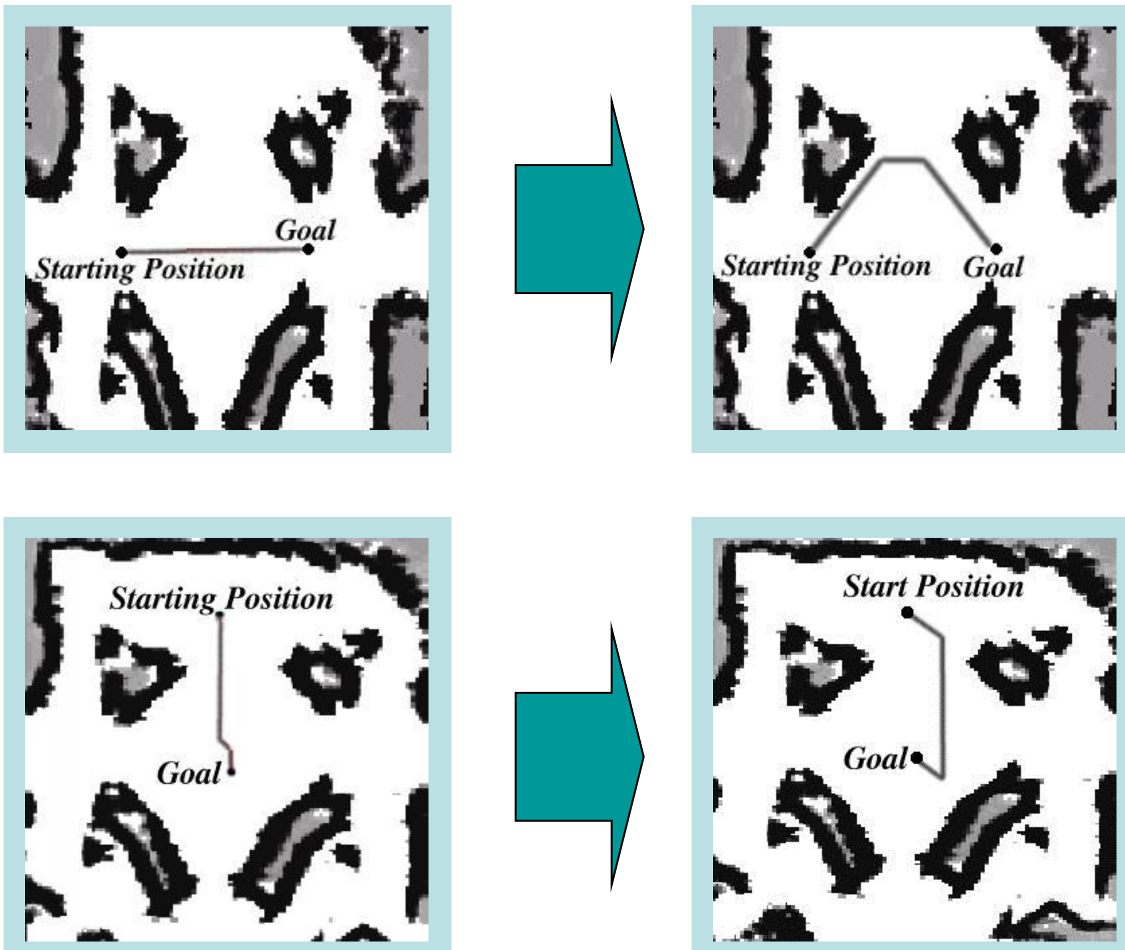


$$\bar{b} = \left\langle \arg\max_{s} b(s);\ H[s] \right\rangle$$

[Roy et al, 98/99]

Path Planning with Augmented MDPs

Conventional planner | Probabilistic Planner



[Roy et al, 98/99]