Artificial Intelligence Foundations and Applications

Stochastic Planning – MDP

Sudeshna Sarkar

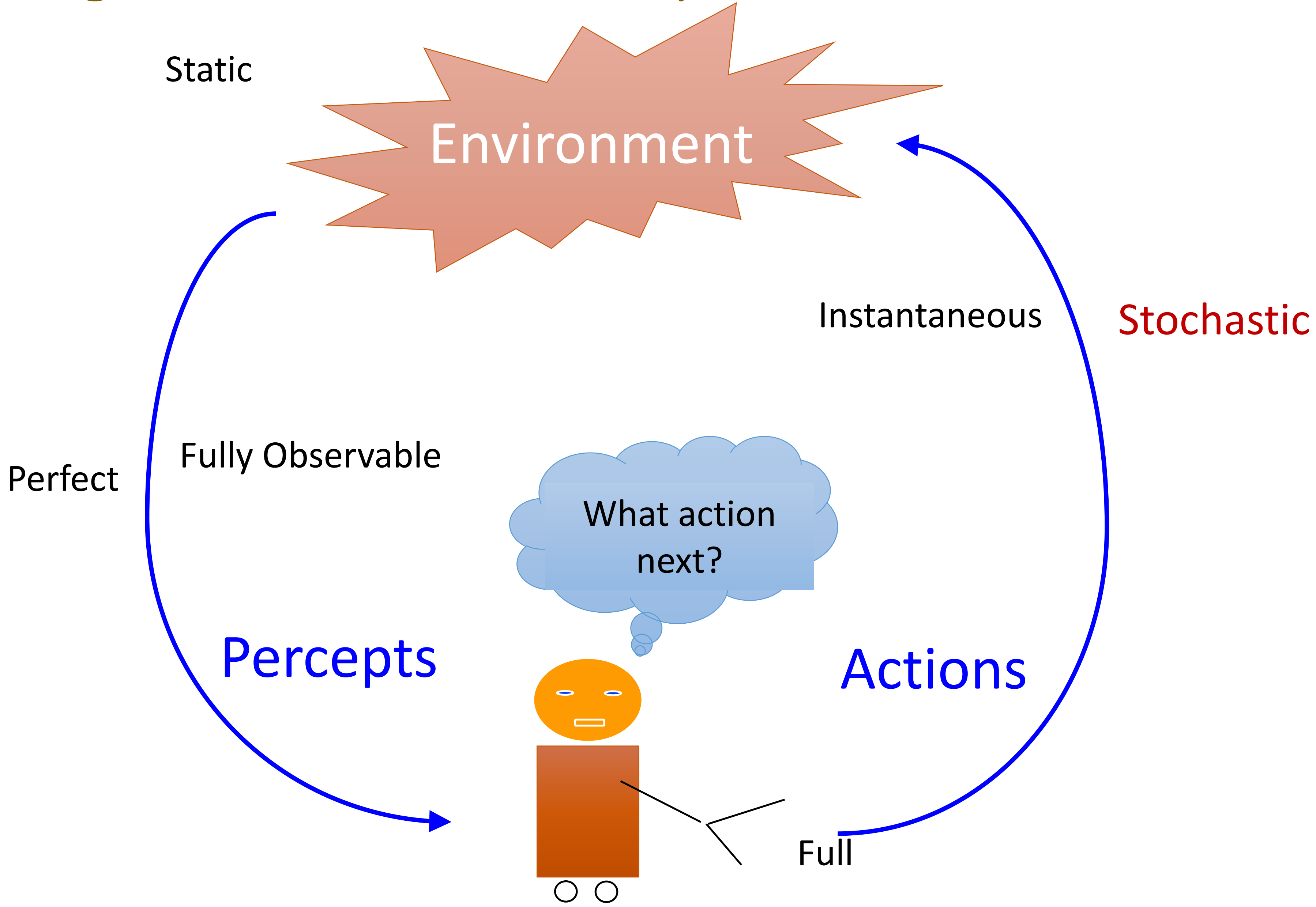
Nov 1 2022

Centre of Excellence in Artificial Intelligence, IIT Kharagpur

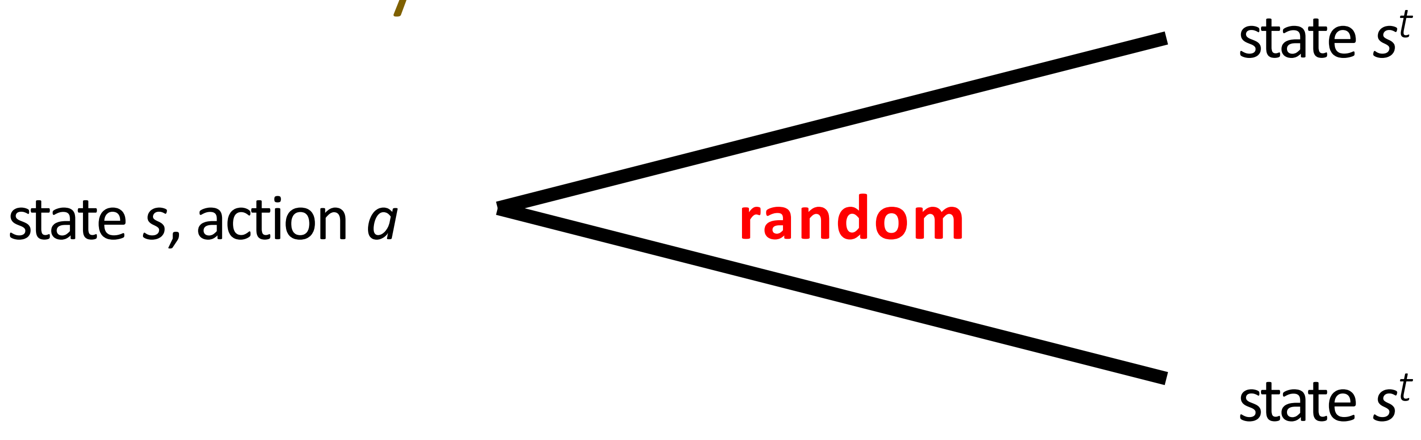
- Markov Decision Processes (MDPs)
- Value Iteration
- Policy Iteration

Many slides adapted fromCS 188:University of California, Berkeley
Pieter Abbeel

Planning under Uncertainty



Uncertainty



Randomness:

- could be caused by limitations of the sensors and actuators of the robot
- could be caused by market forces or nature
- …

Robotics: decide where to move, but actuators can fail, hit unseen obstacles, etc.

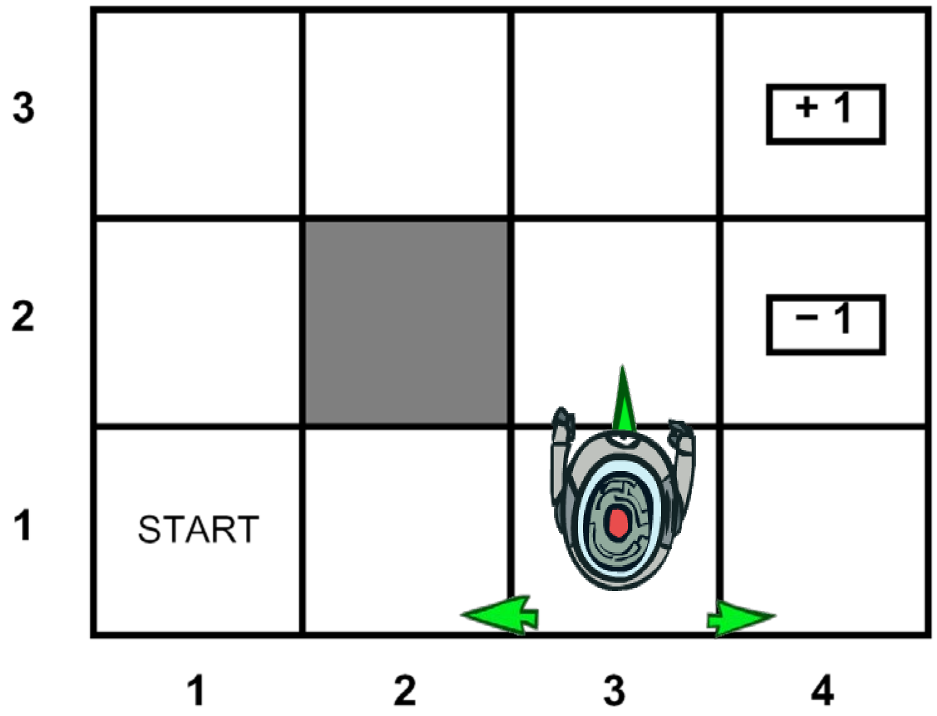
Resource allocation: decide what to produce, don't know the customer demand for various products

Agriculture: decide what to plant; don't know weather and thus crop yield

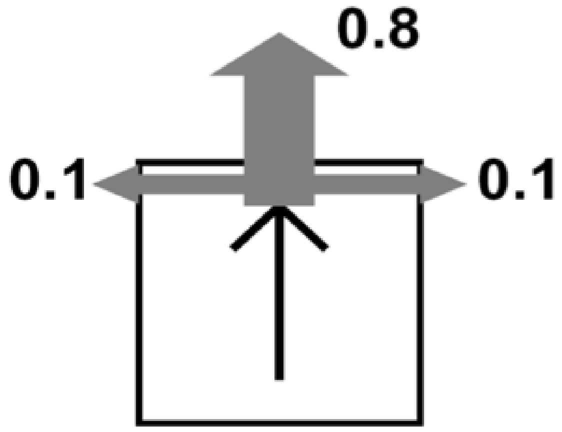
Example: Grid World

Noisy movement: 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

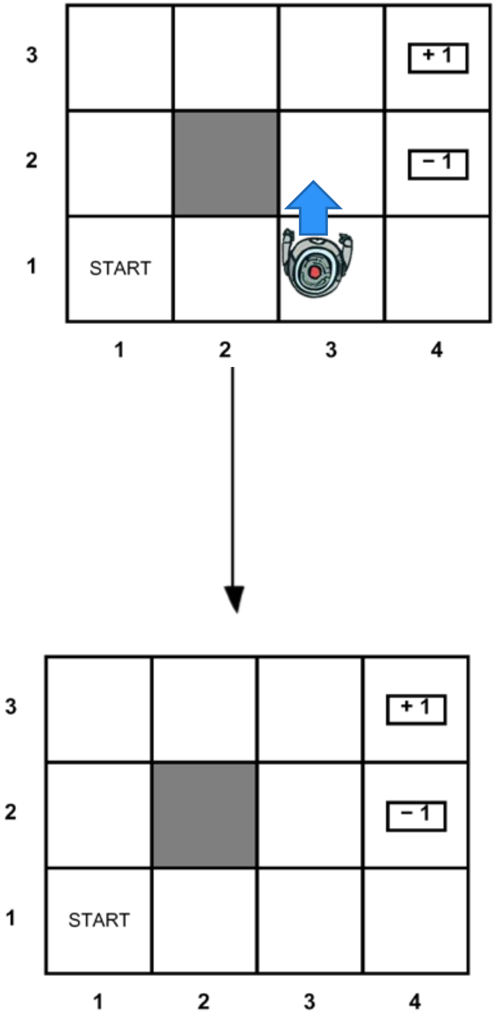


- The agent receives rewards
  - Small "living" reward each step
  - Big rewards come at the end
- Goal: maximize sum of rewards

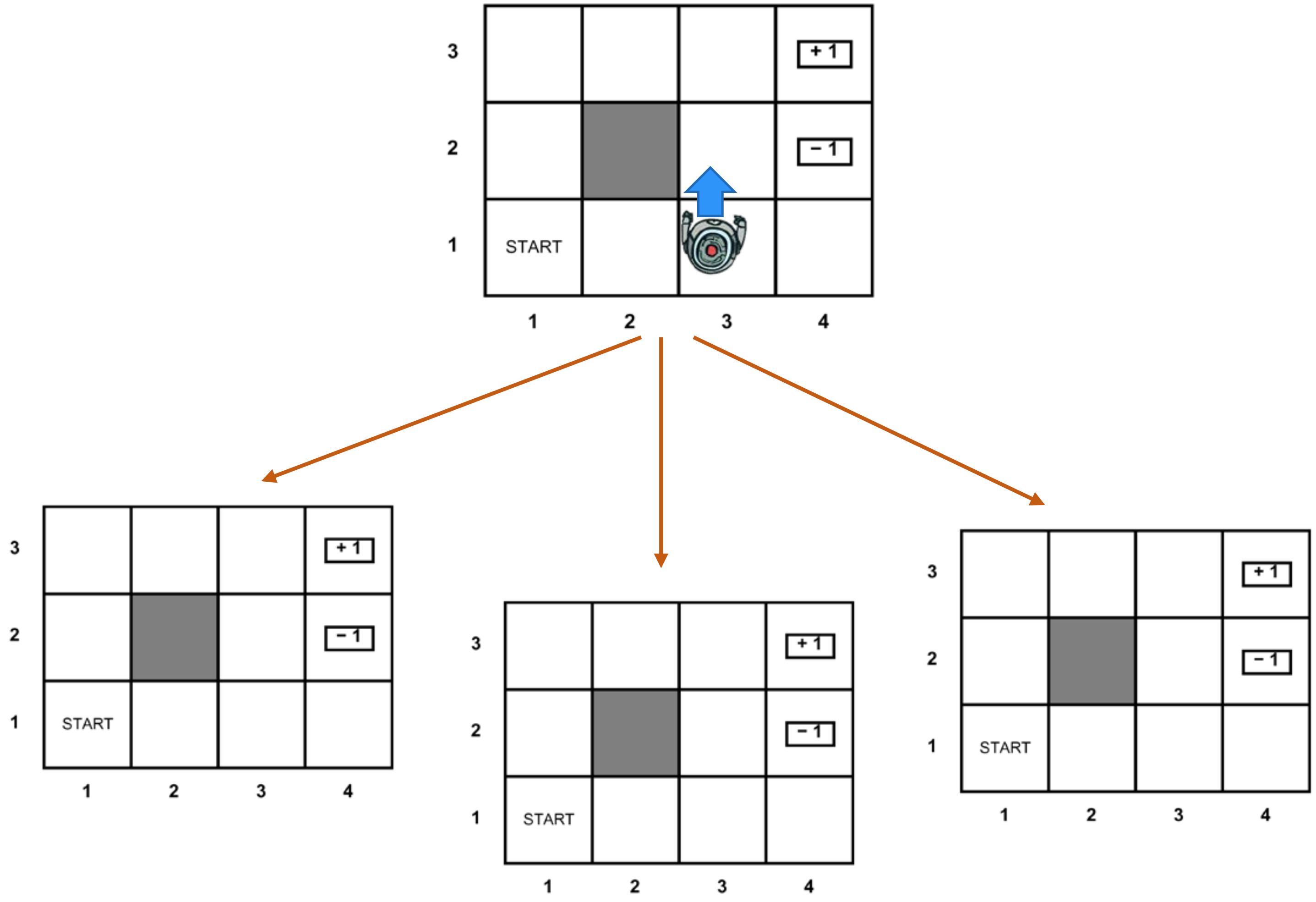


Grid World Actions

Deterministic Grid World



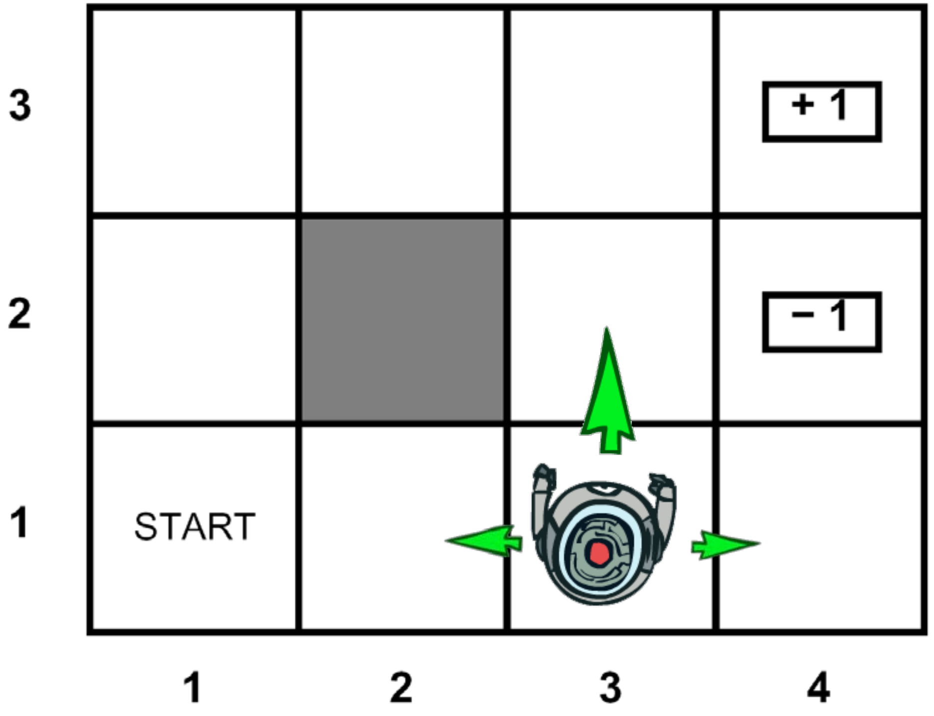
Stochastic Grid World



Markov Decision Processes

An MDP is defined by:

- A set of states s $\in$ S
- A set of actions a $\in$ A
- A transition function T(s, a, s')
  - Probability that a from s leads to s', i.e., P(s' | s, a)
  - Also called the model or the dynamics
- A reward function R(s, a, s')
  - Sometimes just R(s) or R(s')
- A start state
- Maybe a terminal state



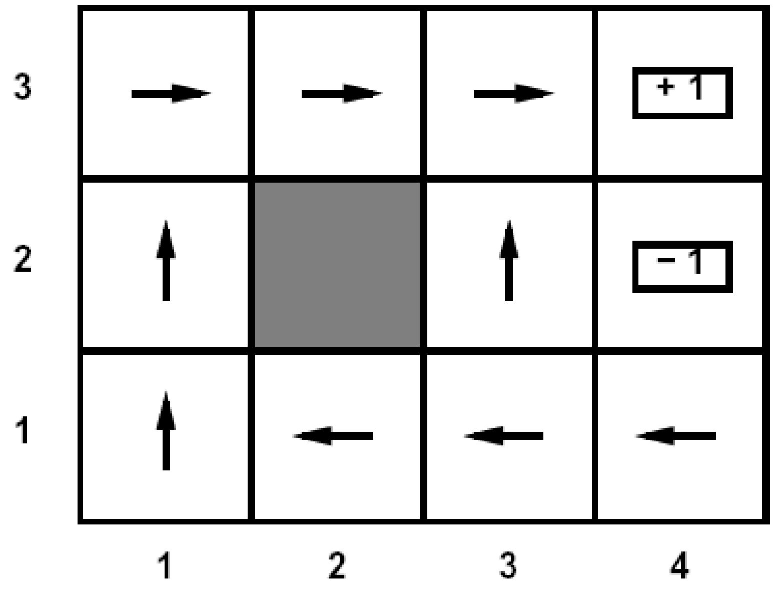
[Demo – gridworld manual intro (L8D1)]

Solution to MDP: Policies

For MDPs, we want an optimal

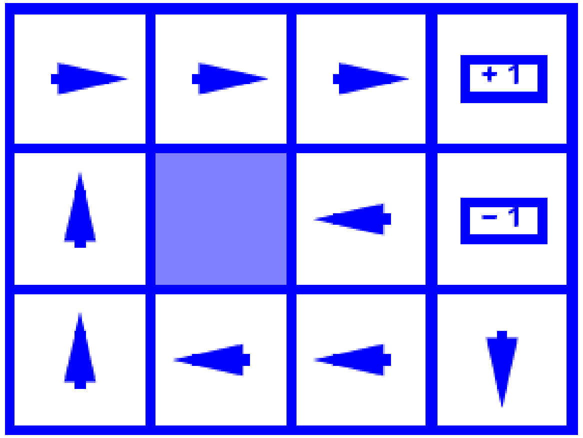
policy π^*: S → A

- A policy π gives an action for each state
- An optimal policy is one that maximizes expected utility if followed

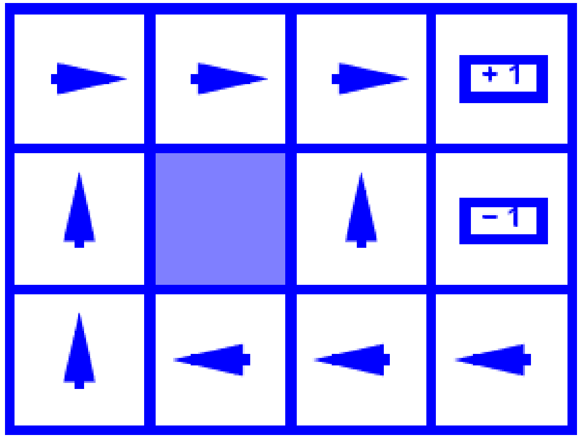


Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

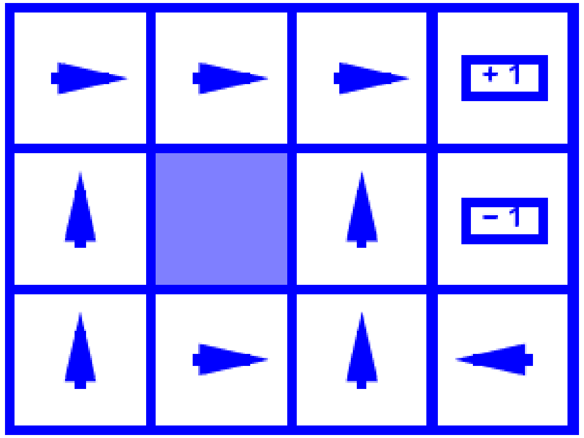
Optimal Policies



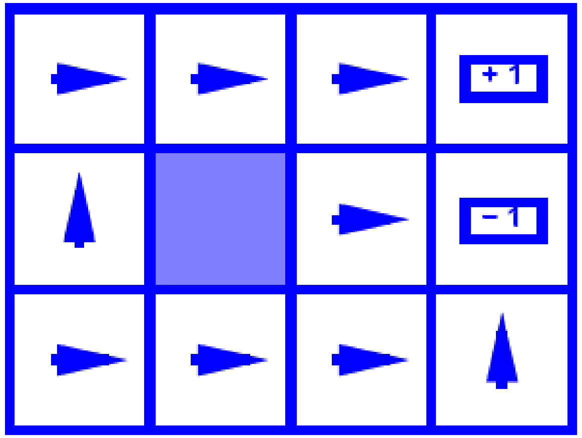
R(s) = -0.01



R(s) = -0.03



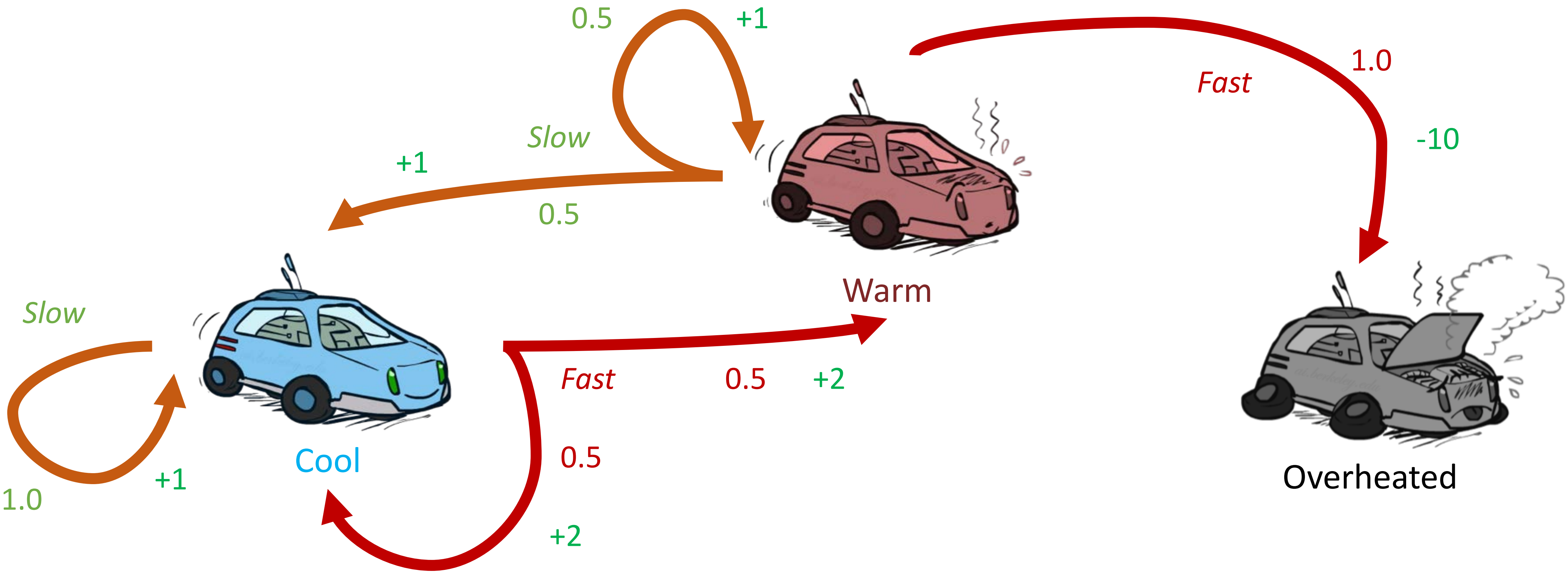
R(s) = -0.4



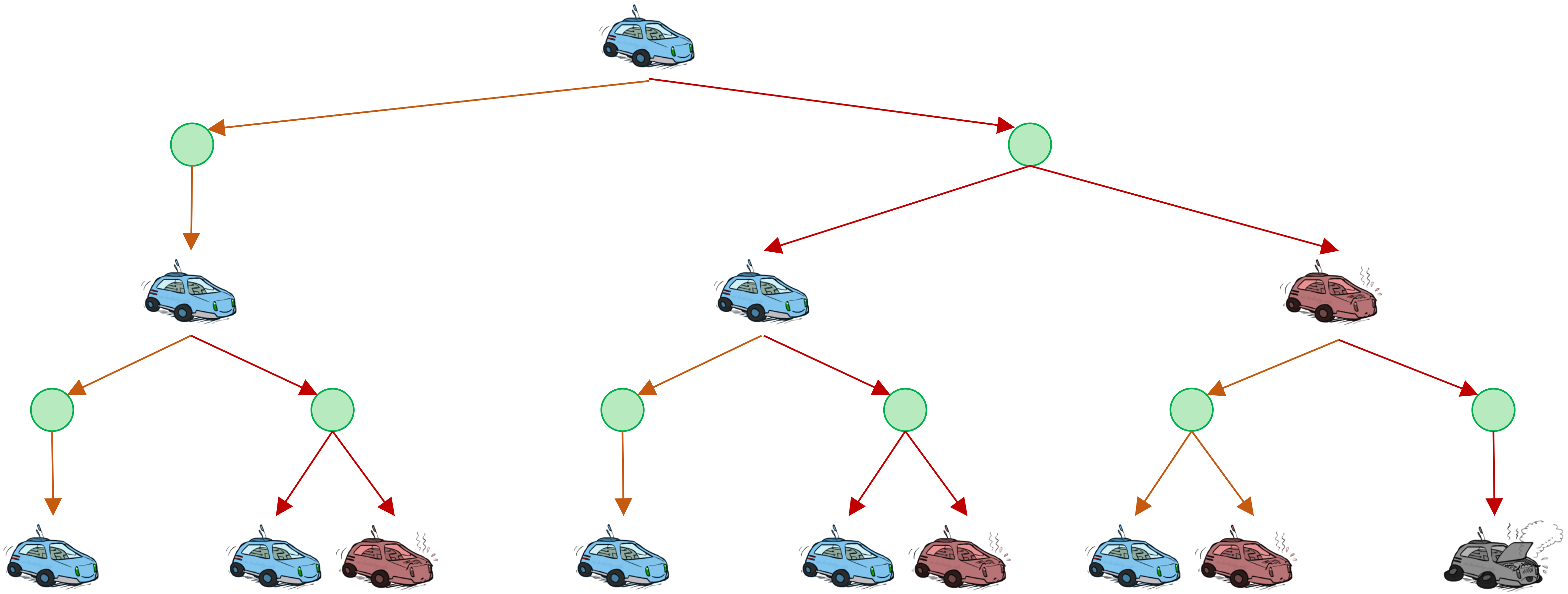
R(s) = -2.0

Example: Racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward

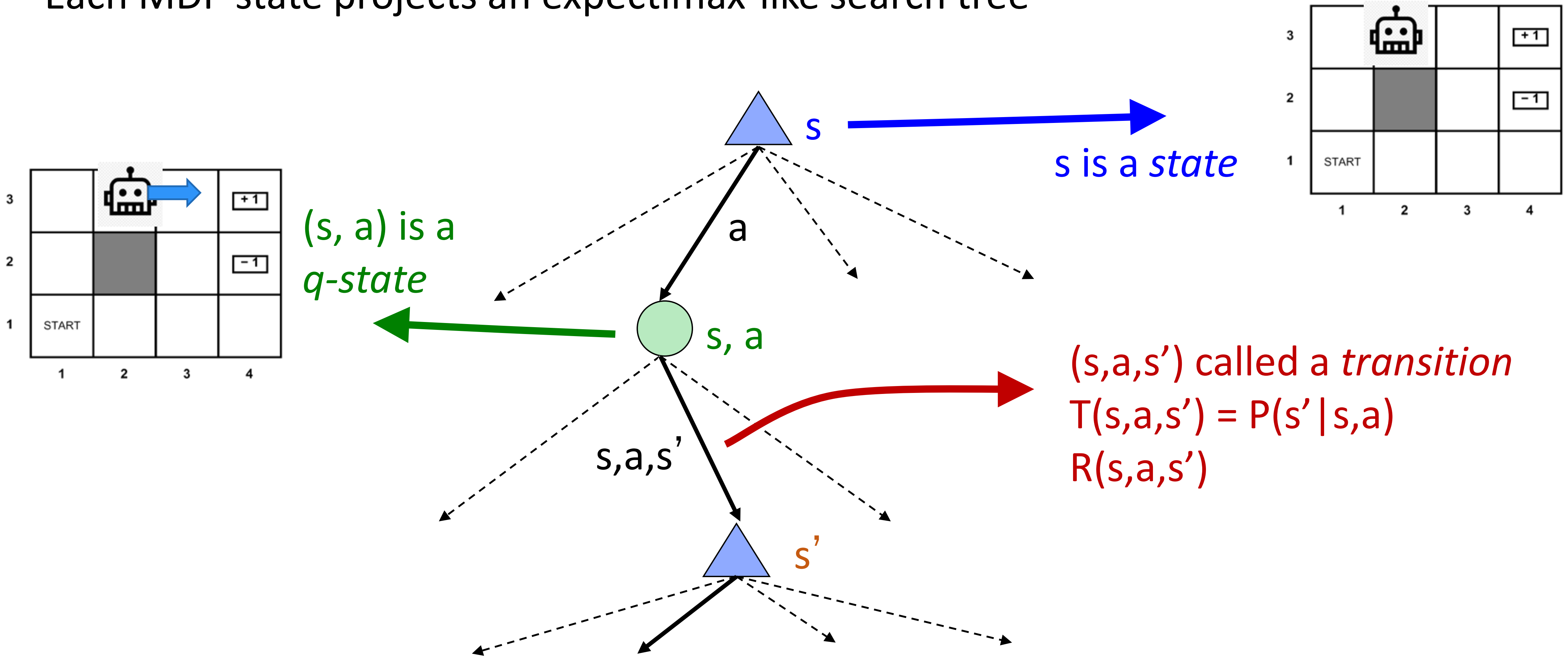


Racing Car Search Tree

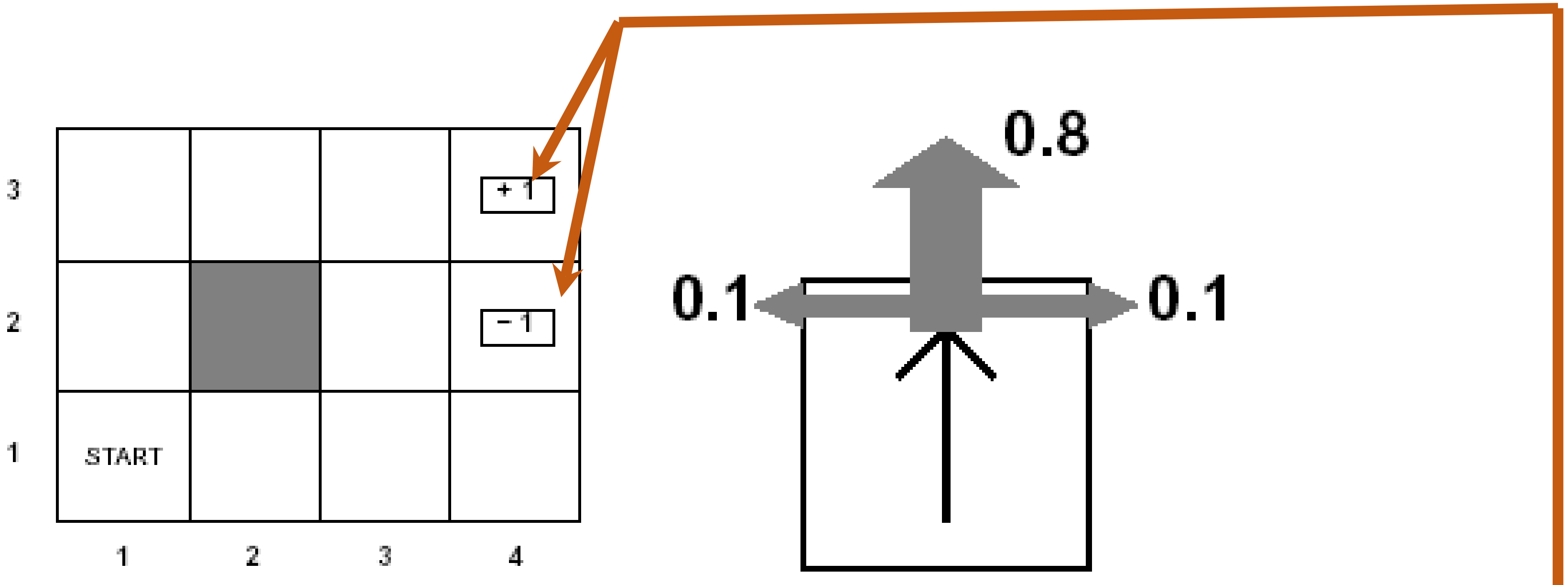


MDP Search Trees

- Each MDP state projects an expectimax-like search tree



Example



$$R(s) = \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

Utilities

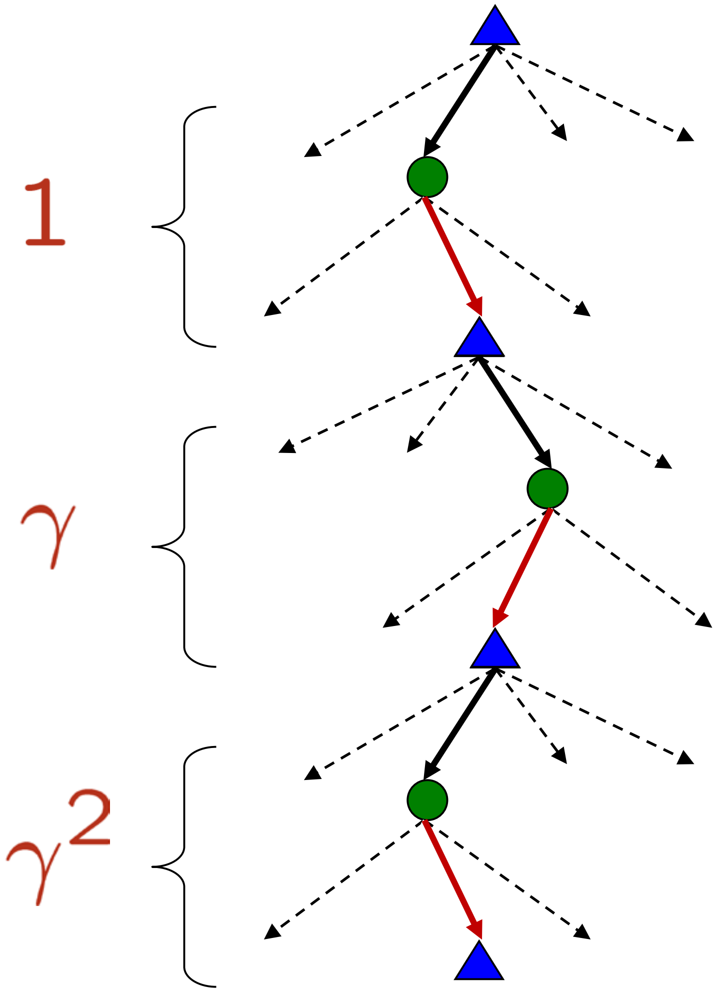
Two ways to define utilities

- Additive utility: $$U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$$

- Discounted utility: $$U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$$

Discounting

- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Reward now is better than later
  - Also helps our algorithms converge
- Example: discount of 0.5
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])



Optimal Quantities

- The value (utility) of a state s:

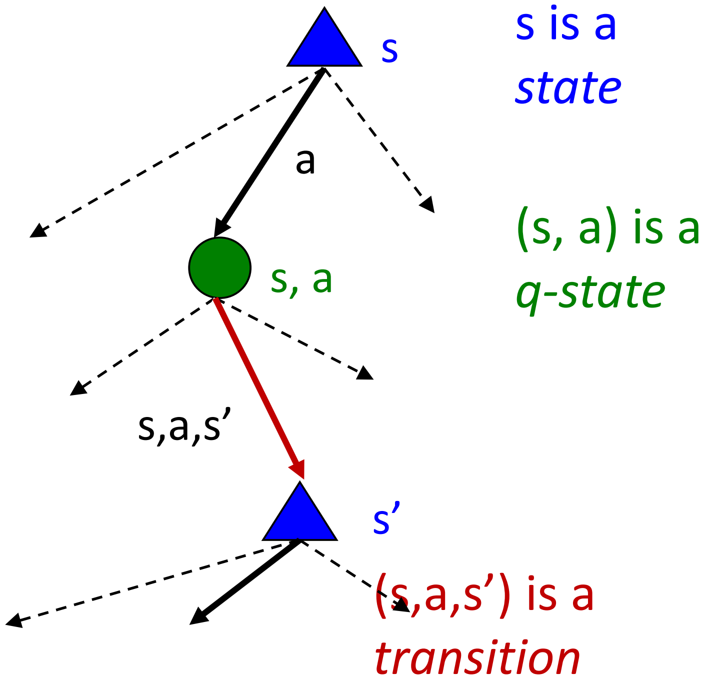
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):

  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

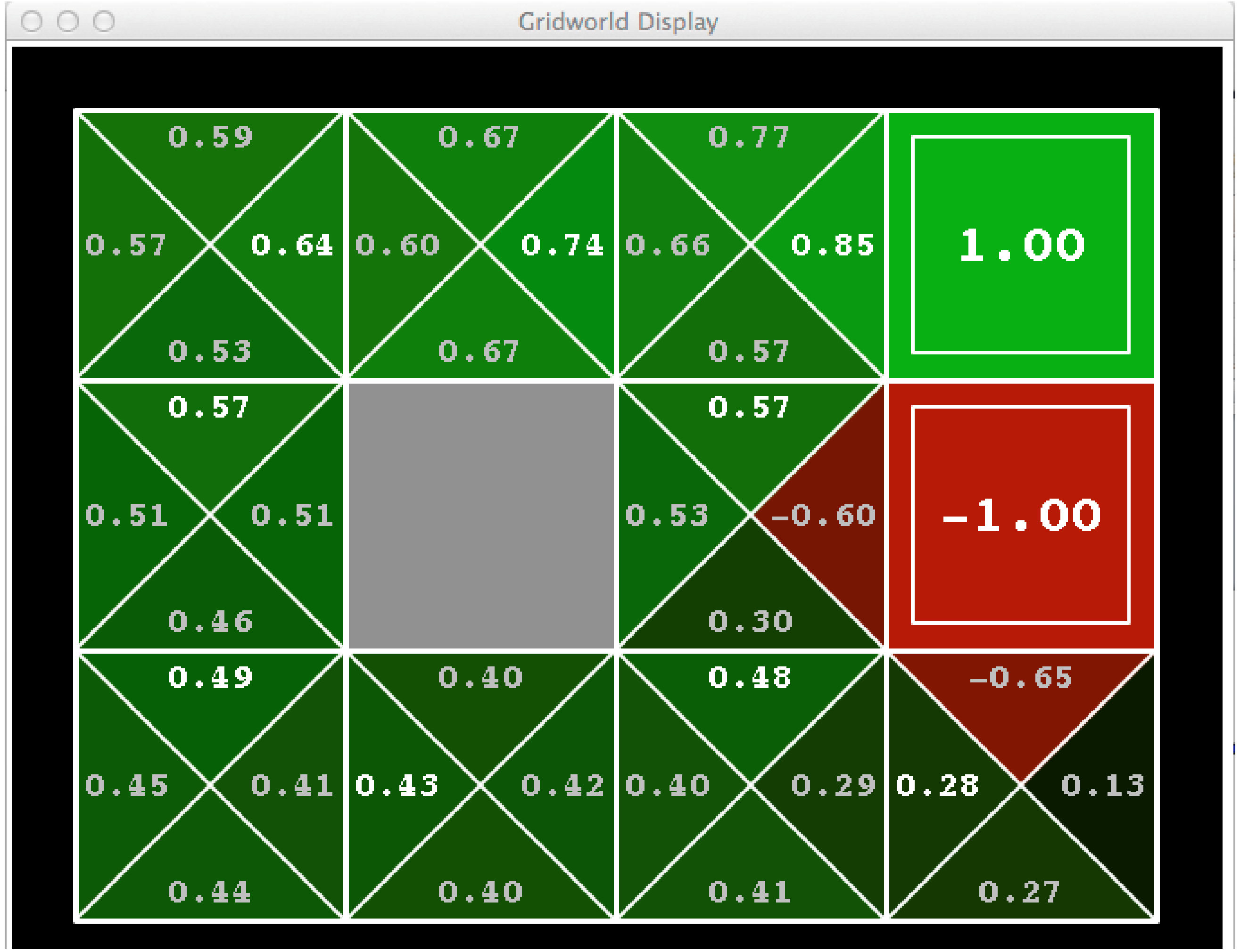
  $\pi^*(s)$ = optimal action from state s



Gridworld V* Values

Noise = 0.2
Discount = 0.9
Living reward = 0

Gridworld Q* Values



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Function

Value function for a policy $\pi: S \rightarrow A$

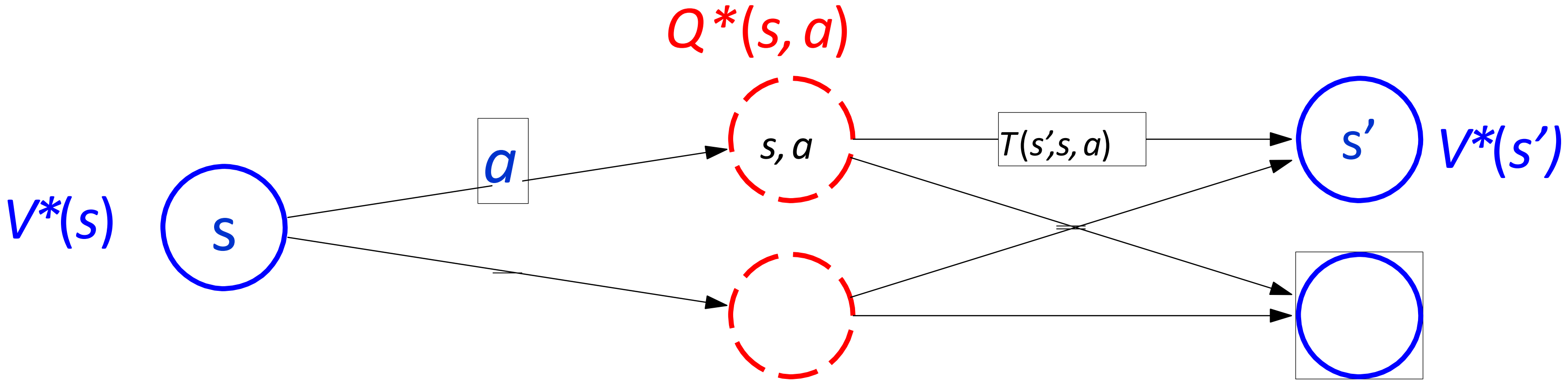
$$V^{\pi}(s) = R(s) + \gamma \sum_{s\prime \in S} T(s, \pi(s), s') \, V^{\pi}(s')$$

Optimum value function

$$V^{*}(s) = \max_{\pi} V^{\pi}(s)$$

$Q^{\pi}(s, a)$: the expected utility of taking action a from state s, and then following policy π.

Optimal values and Q-values

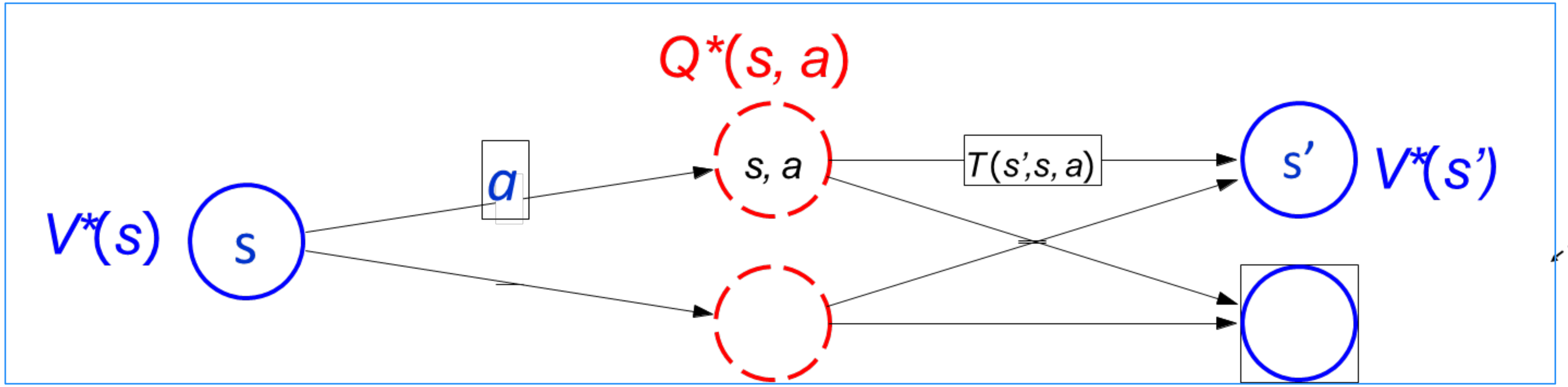


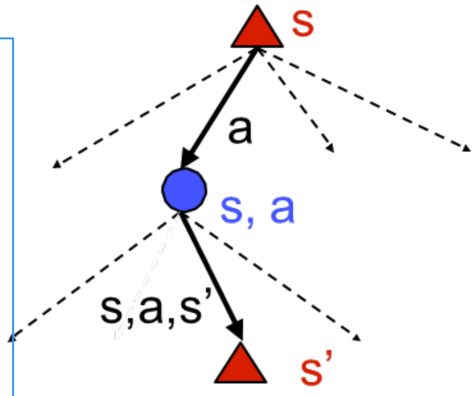
$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

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

The Bellman Equations

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





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

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

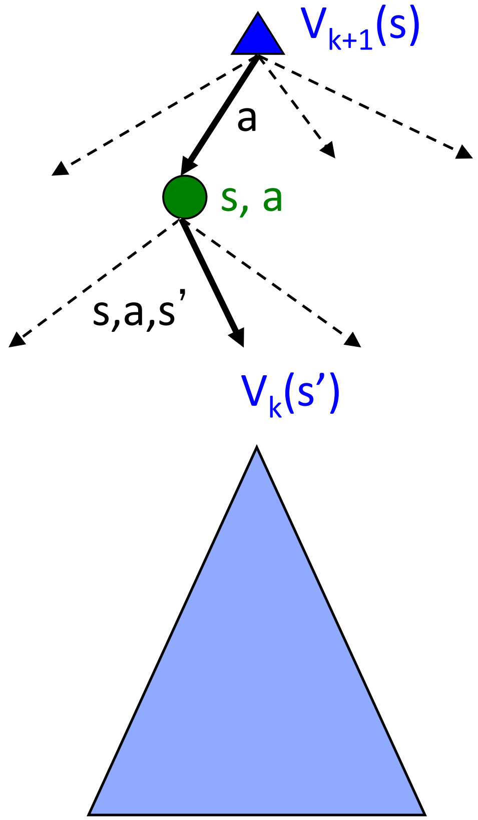
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

Optimal policies: ?

Value Iteration

1. For each state *s*, initialize $V(s) := 0$.
2. **for** until convergence **do**
3. For every state, update

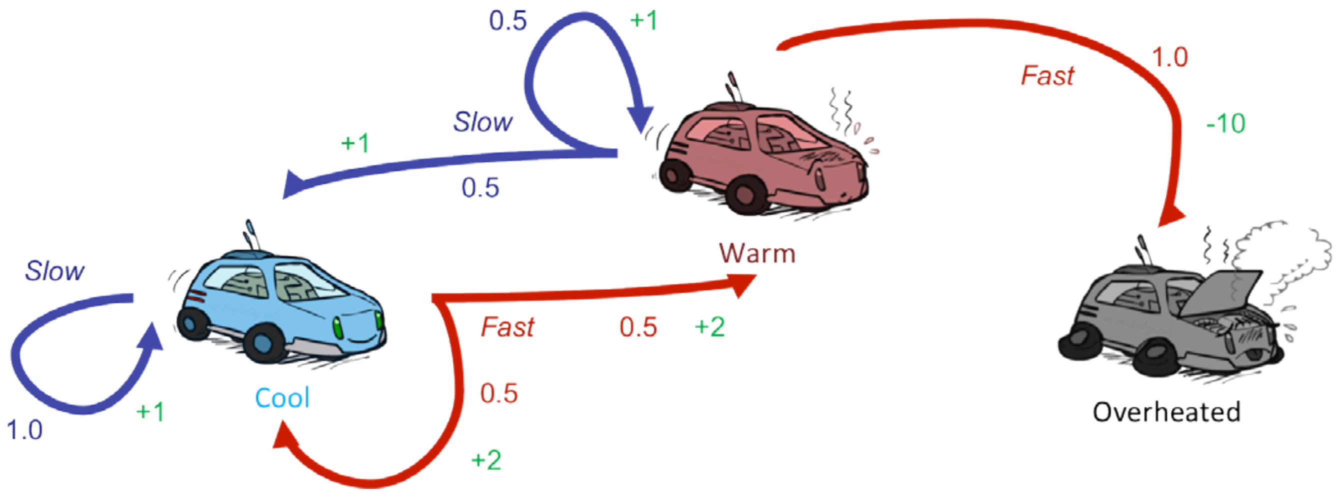
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$



- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Example: Value Iteration

V_2			
V_1	S: 1 F: .5*2+.5*2=2		
V_0	0	0	0

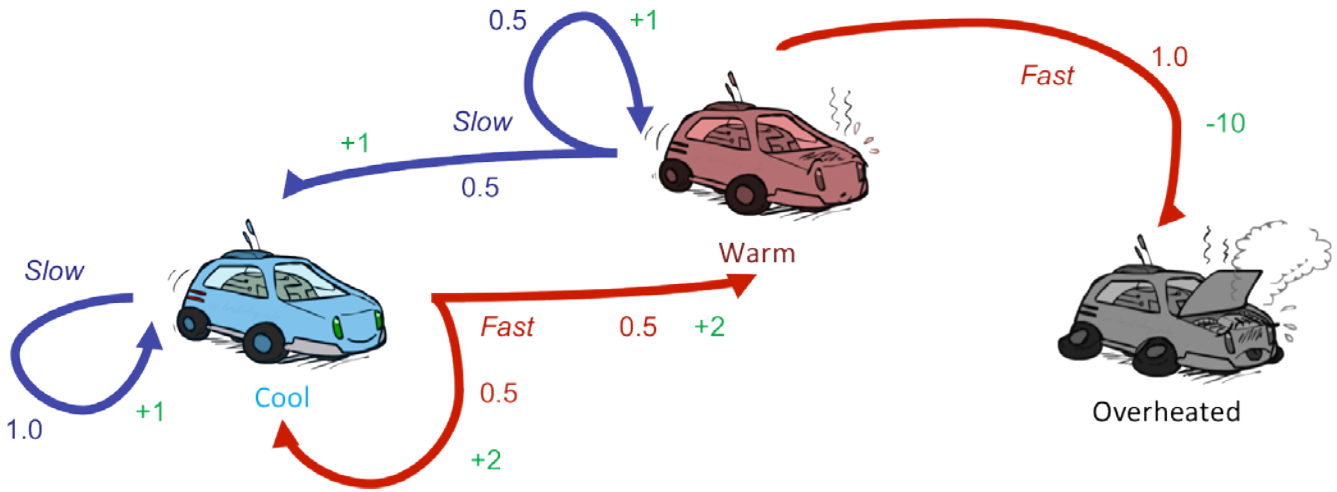


Assume no discount!

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

Example: Value Iteration

V_2			
V_1	2	S: .5*1+.5*1=1 F: -10	
V_0	0	0	0

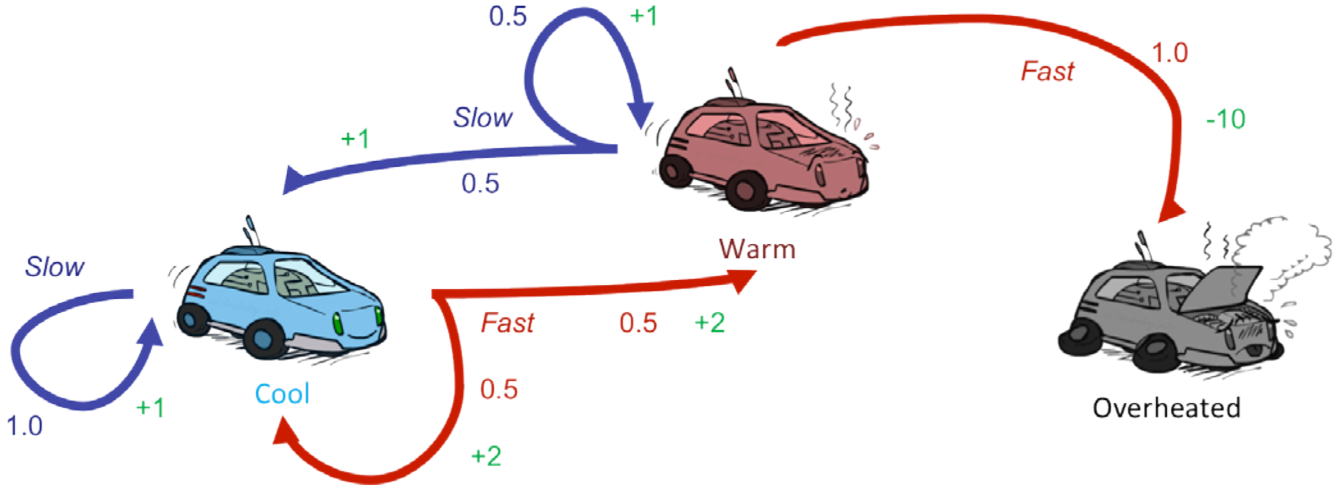


Assume no discount!

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

Example: Value Iteration

V_2			
V_1	2	1	0
V_0	0	0	0

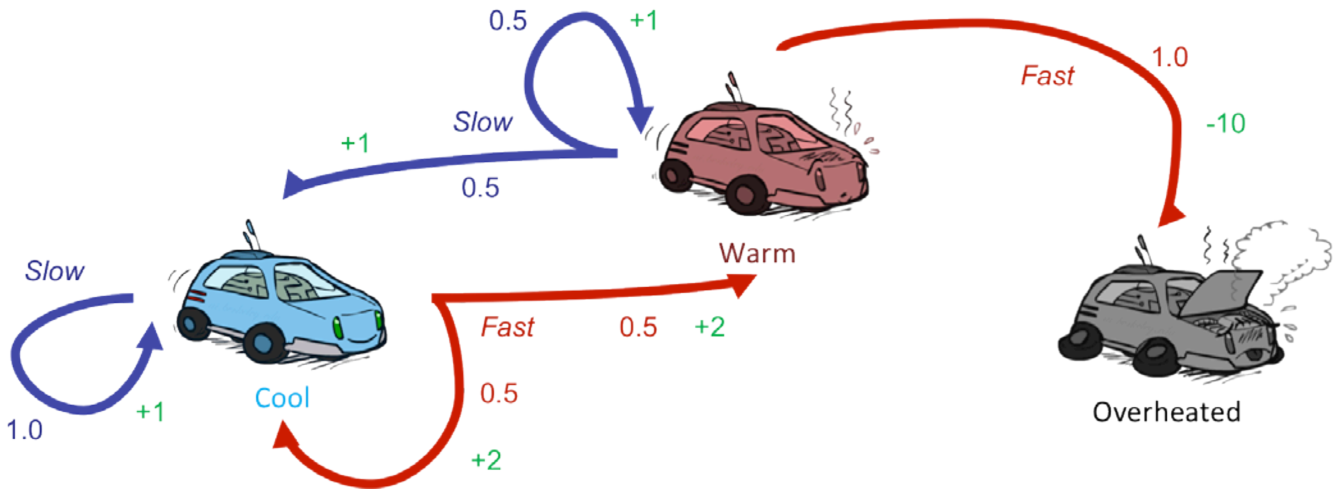


Assume no discount!

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

Example: Value Iteration

V_2	S: 1+2=3 F: .5*(2+2)+.5*(2+1)=3.5		
V_1	2	1	0
V_0	0	0	0

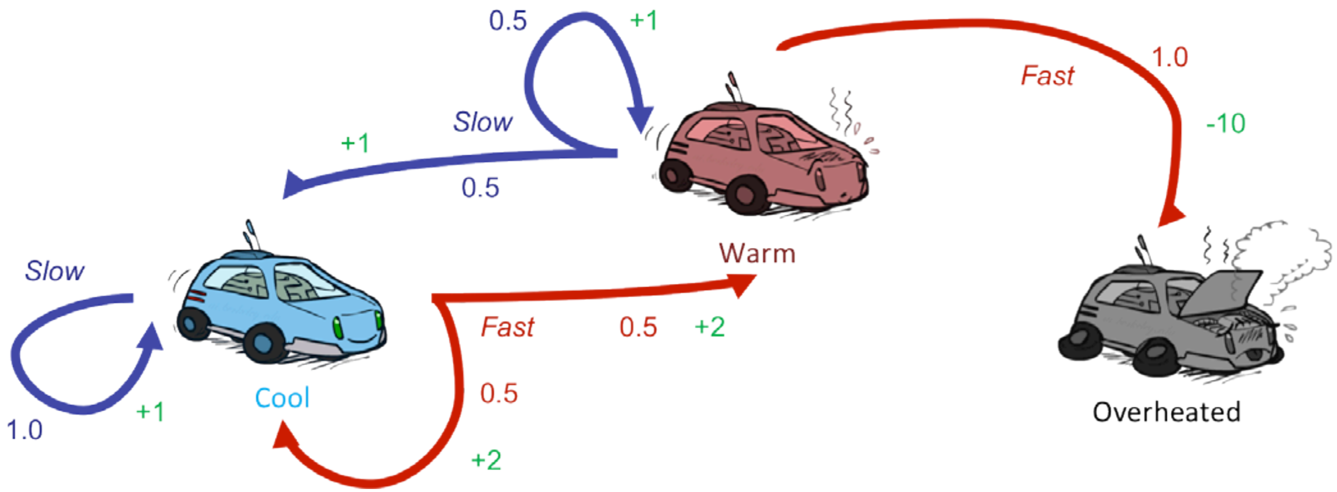


Assume no discount!

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

Example: Value Iteration

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0



Assume no discount!

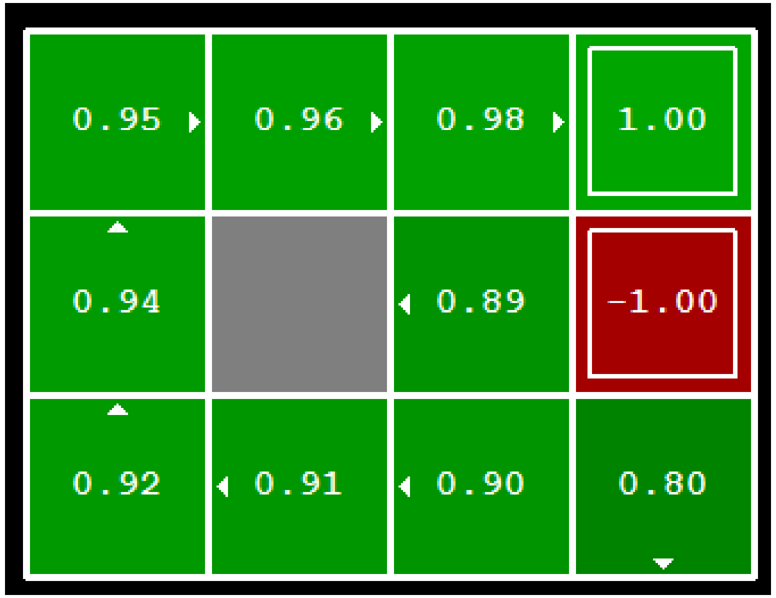
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

Computing Actions from Values

- Let's imagine we have the optimal values V*(s)

- How should we act?

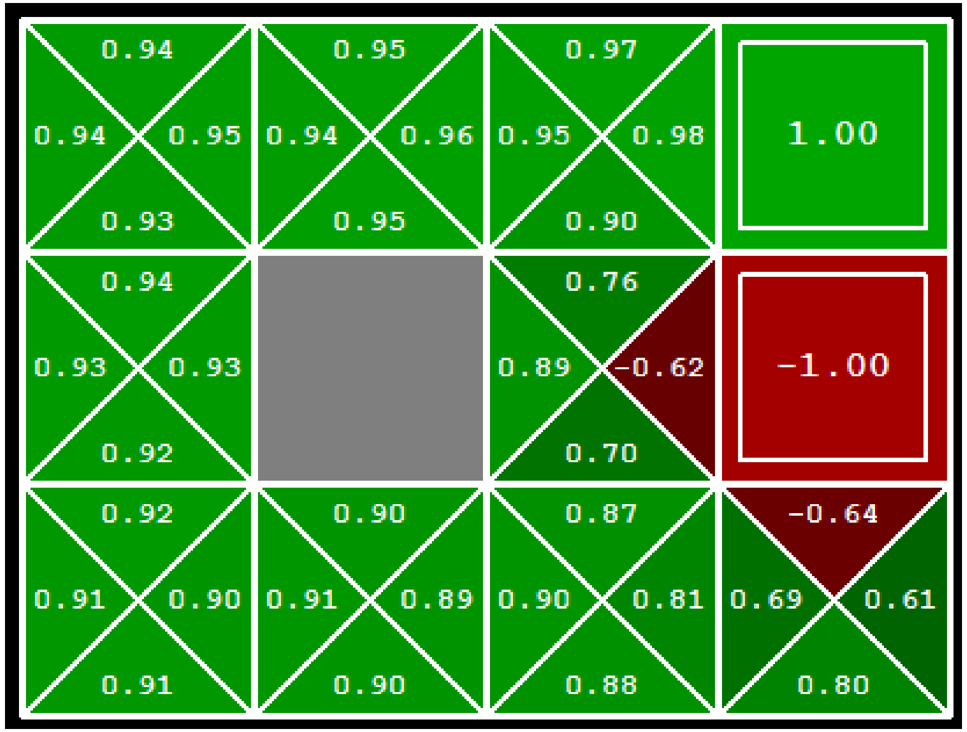
$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$



- This is called policy extraction, since it gets the policy implied by the values

Computing Actions from Q-Values

- Let's imagine we have the optimal q-values:



- How should we act?
  - Completely trivial to decide!

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

- Important lesson: actions are easier to select from q-values than values!

Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy Evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy Improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

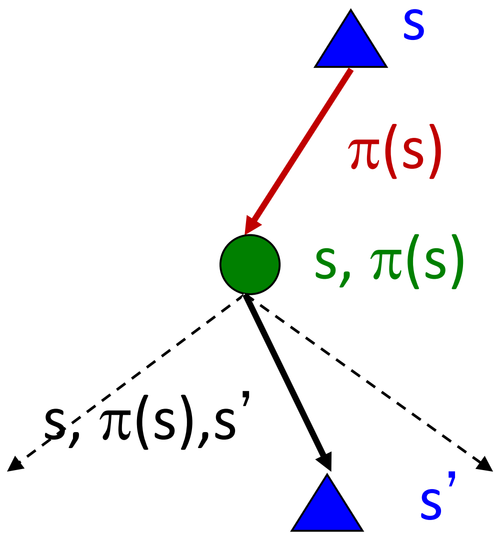
- This is Policy Iteration
  - It’s still optimal!
  - Can converge (much) faster under some conditions

Policy Evaluation

- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates

$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$



Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma \, V_k^{\pi_i}(s')\right]$$

- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we’re done)
- Both are dynamic programs for solving MDPs