复旦大学大数据学院
School of Data Science, Fudan University

魏忠钰

# Markov Decision Processes

Data Intelligence and Social Computing Lab (DISC)

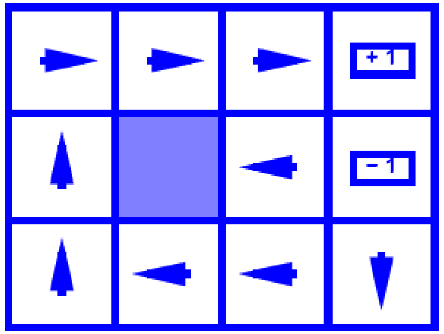
November 9th, 2021

# Outline

- Sequential decision making problems

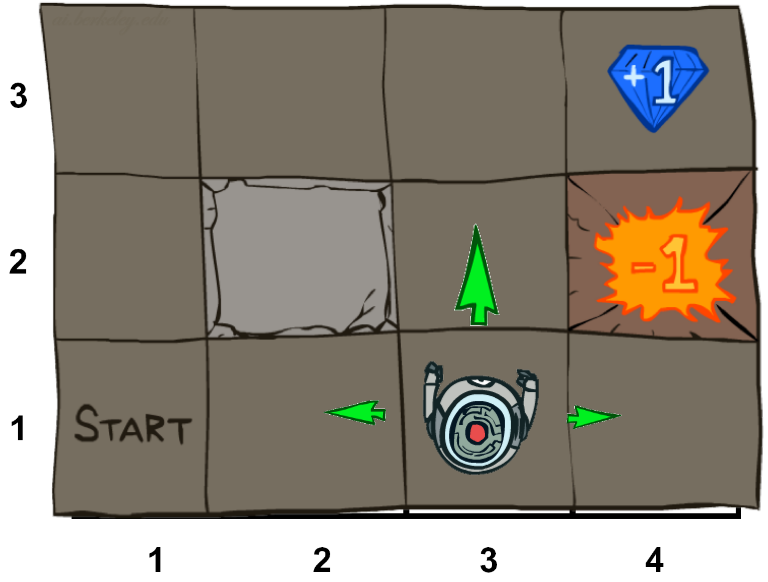
# Grid World with Uncertainties

- The agent takes actions and results are uncertain
  - **Actions**: up, down, right, left
  - **As planned**: 80% to the intended direction.
  - **Uncertain**: 20% to the right angle of the intended direction.
  - **Stay**: the agent stays if it hits a wall



Policy

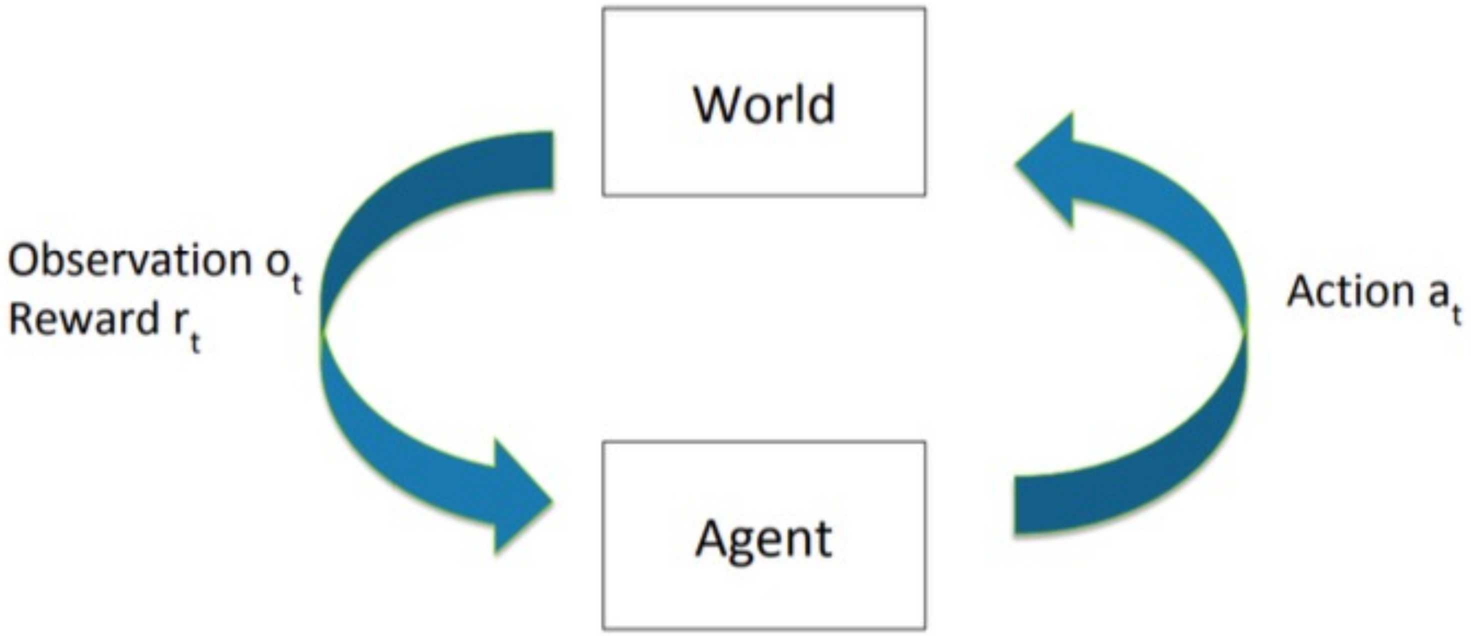
- The agent collects rewards
  - Rewards for each step, it can be two kinds
    - **Immediate reward**: each step (can be negative)
    - **End game reward**: come at the end (good or bad)




- Goal: **maximize sum of rewards**
- Solution: policy $\pi$, i.e., recommended action for each state.

# Sequential Decision Making

- An agent makes a sequence of actions: $\{a_t\}$
- Observes a sequence of observations: $\{o_t\}$ i.e., $\{s_t\}$
- Receives a sequence of rewards: $\{r_t\}$
- Trajectory / episodes: $s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, r_3$ …



# Outline

- Sequential decision making problems
- Markov decision processes

# Markov Process

- S is a finite set of states s $\in$ S
- P is dynamics / transitions model $p(s_{t+1} = s' | s_t = s)$
- Satisfies Markov Property

$$p(s_{t+1} | s_t) = p(s_{t+1} | s_t, \dots, s_1)$$

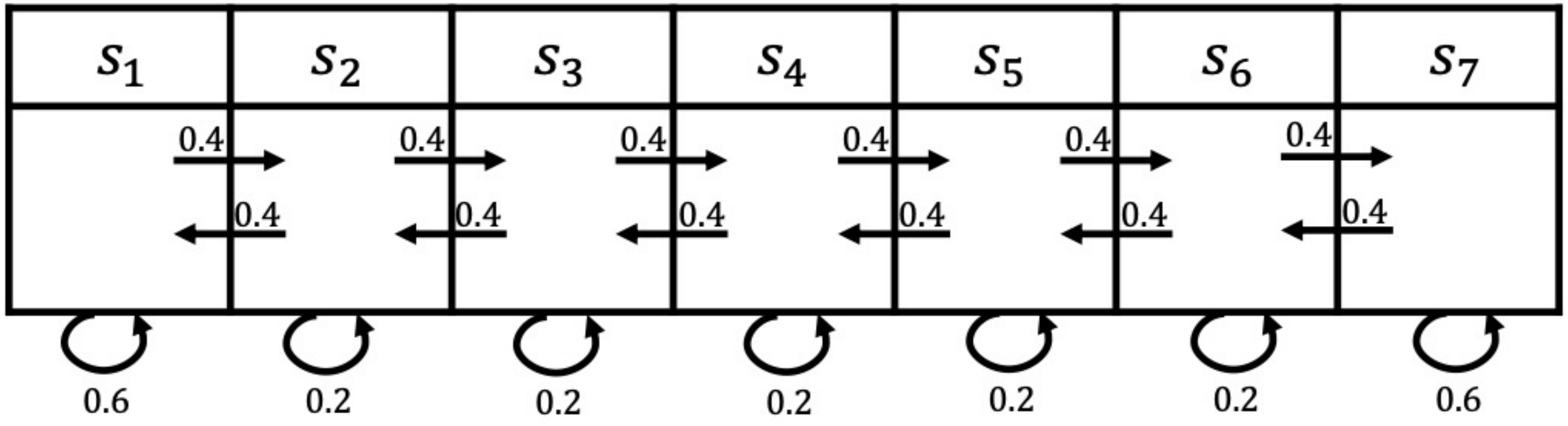
- Future is independent of past states given present state

$$P = \begin{pmatrix} P(s_1|s_1) & P(s_2|s_1) & \cdots & P(s_N|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & \cdots & P(s_N|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_N) & P(s_2|s_N) & \cdots & P(s_N|s_N) \end{pmatrix}$$

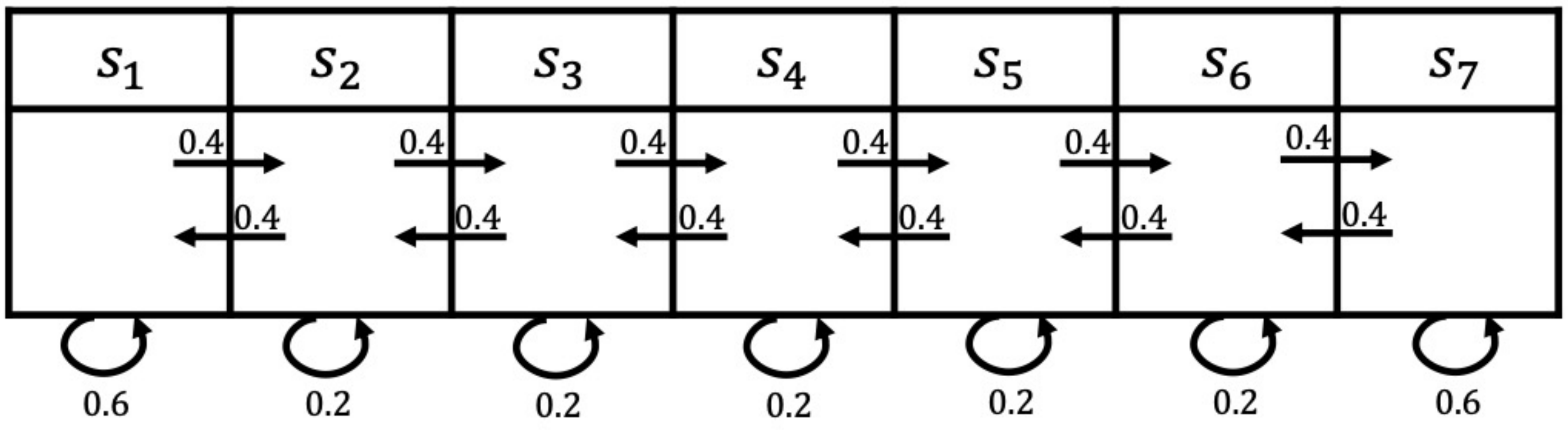
*Suppose there are N states, P can be represented as a matrix of NxN*

# Example: Mars Rover Markov Model

- Mars Rover is moving in a horizontal space of 7 states.
  - $\{S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$
- Move one time step by another with a transition probability.
  - $S_1$: 60% stay, 40% move to $S_2$
  - $S_2$: 20% stay, 40% move to $S_2$, 40% move to $S_3$
  - ….



# Example: Mars Rover Markov Model




Sample episodes starting from $s_4$:

- $s_4, s_5, s_6, s_7, s_7, s_7, \ldots$
- $s_4, s_4, s_5, s_4, s_5, s_5, \ldots$
- $s_4, s_3, s_3, s_3, s_2, s_2, \ldots$
- $s_4, s_4, s_5, s_3, s_4, s_5, \ldots$

$$P = \begin{pmatrix} 0.6 & 0.4 & 0 & 0 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 & 0 & 0 & 0 \\ 0 & 0.4 & 0.2 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0.2 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0.2 & 0.4 \\ 0 & 0 & 0 & 0 & 0 & 0.4 & 0.6 \end{pmatrix}$$

# Markov Reward Process

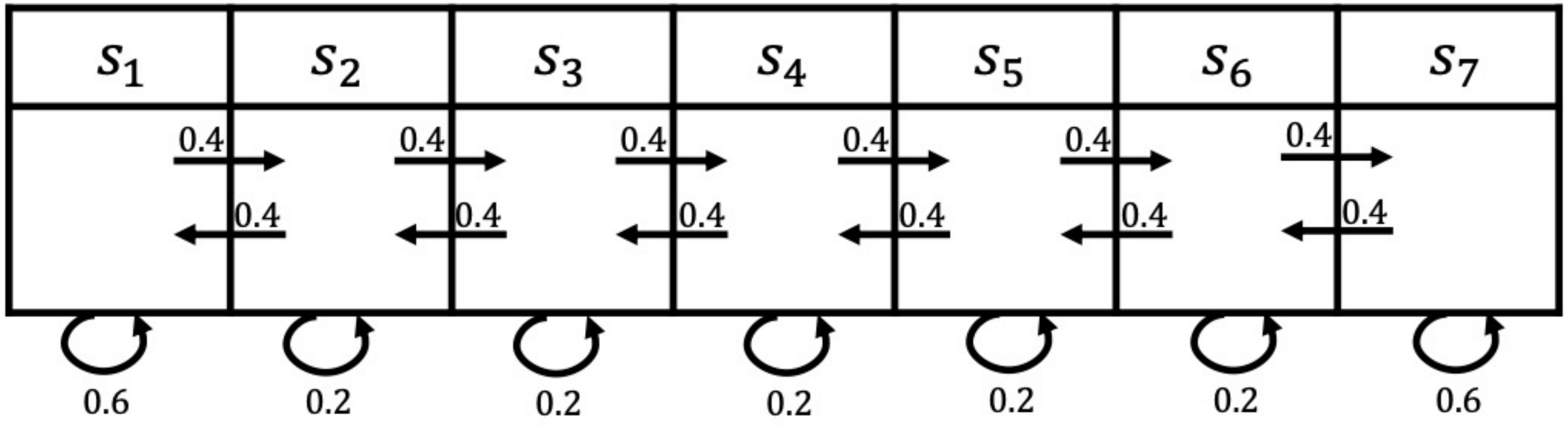
- **Markov Process + Reward**
- S is a finite set of states $s \in$ S
- P is dynamics / transitions model $p(s_{t+1} = s' | s_t = s)$
- R is a reward function $R(s_t = s) = E[r_t | s_t = s]$
- $\gamma$ is discount factor $\gamma \in [0,1]$

$$P = \begin{pmatrix} P(s_1|s_1) & P(s_2|s_1) & \cdots & P(s_N|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & \cdots & P(s_N|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_N) & P(s_2|s_N) & \cdots & P(s_N|s_N) \end{pmatrix}$$

$$R = \begin{Bmatrix} R(s_1) \\ R(s_1) \\ \cdots \\ R(s_N) \end{Bmatrix}$$

# Example: Mars Rover Markov Reward Process

- Mars Rover is moving in a horizontal space of 7 states.
- Move one time step by another with a transition probability.
- Get a reward at each position
  - R = {+1, 0, 0, 0, 0, 0, +10}




- Sample episodes starting from S4:
  - $s_4, 0, s_5, \ \ 0, \ s_6, 0, \ s_7, +10, s_7, +10, s_7, +10, \ldots$
  - $s_4, 0, s_4, \ \ 0, \ s_5, 0, \ s_4, 0, s_5, 0, s_5, \ldots$

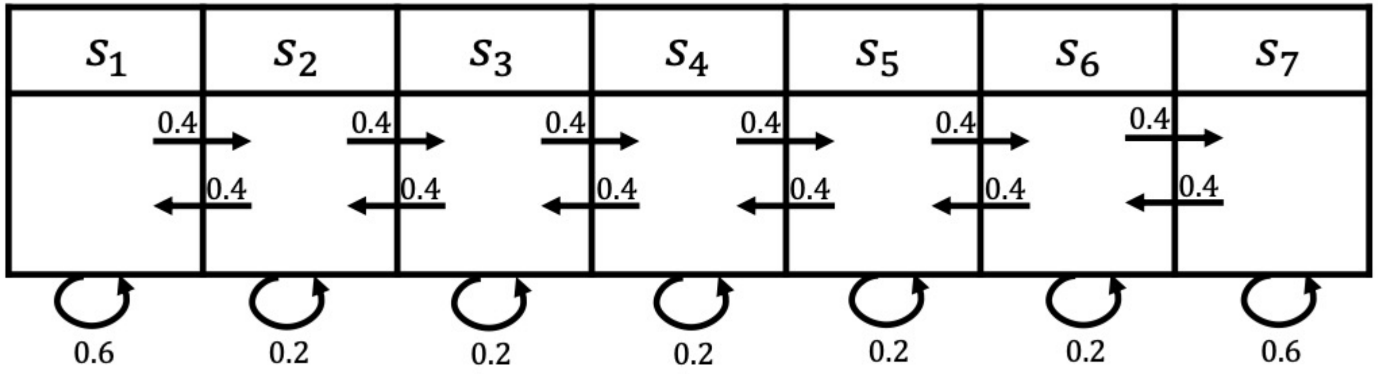
# Return & Value Function for MRP

- Definition of return from time step t, $G_t$ : also known as utility
  - Discounted sum of rewards from time step t to horizon
  - Horizon: number of time steps in the trajectory
  - $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots$

- Definition of Value Function, V(s)
  - Expected return from starting in state s

$$V(s) = E[G_t | s_t = s] = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots | s_t = s]$$

# Example II : Mars Rover Markov Reward Process




- Reward: +1 in $s_1$, +10 in $s_7$, 0 in all other states
- Sample returns for 4-step episodes, $\gamma = 1/2$:
  - $s_4, 0, s_5,\ \ 0,\ \ s_6, 0,\ \ s_7, +10$

$$0 + \frac{1}{2} * 0 + \frac{1}{4} * 0 + \frac{1}{8} * 10 = 1.25$$

- Value function: expected return from starting in state s

$$V(s) = E[G_t | s_t = s] = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots | s_t = s]$$

- V = [1.53, 0.37, 0.13, 0.22, 0.85, 3.59, 15.31]

# Markov Decision Processes (MDPs)

- Markov Process + Reward + Action
- $S$ is a (finite) set of Markov states s ∈ S
- $A$ is a (finite) set of actions a ∈ A
- $P$ is dynamics / transition-model for each action,

$$P(s_{t+1} = s' | s_t = s, a_t = a)$$

- $R$ is a reward function

$$R(s_t = s, a_t = a) = E[r_t | s_t = s, a_t = a]$$

- $\gamma$ is discount factor $\gamma \in [0,1]$
- MDP is a tuple: $(S, A, P, R, \gamma)$

# Example: Mars Rover Markov Decision Process

- Mars Rover is moving in a horizontal space of 7 states.
- Move one time step by another with a transition probability.
- Get a reward at each position
- Two actions: left or right
  - Without uncertainty

| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $s_7$ |
|---|---|---|---|---|---|---|
| | | |  | | | |

- What is the transition model?
  - With N*N*A parameters

左

$$P(s'|s,a_1) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

右

$$P(s'|s,a_2) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Sample episodes starting from S4:

- $s_4, r, 0, s_5, r, 0,\ s_6, r, 0,\ s_7, r, +10,\ s_7, r, +10,\ s_7, r, +10$
- $s_4, l, 0, s_3, l, 0,\ s_2, l, 0$

# MDP Policies

- For MDPs, the solution is called policy $\pi$ that gives an action for each
  - Gi
  - $\pi$(
  - Po

| $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $s_7$ |
|---|---|---|---|---|---|---|
| | | |  | | | |

- In this course, we only consider deterministic policy, which is a mapping from (S -> A)

**Two deterministic actions for each state.**

**How many policies do we have?**

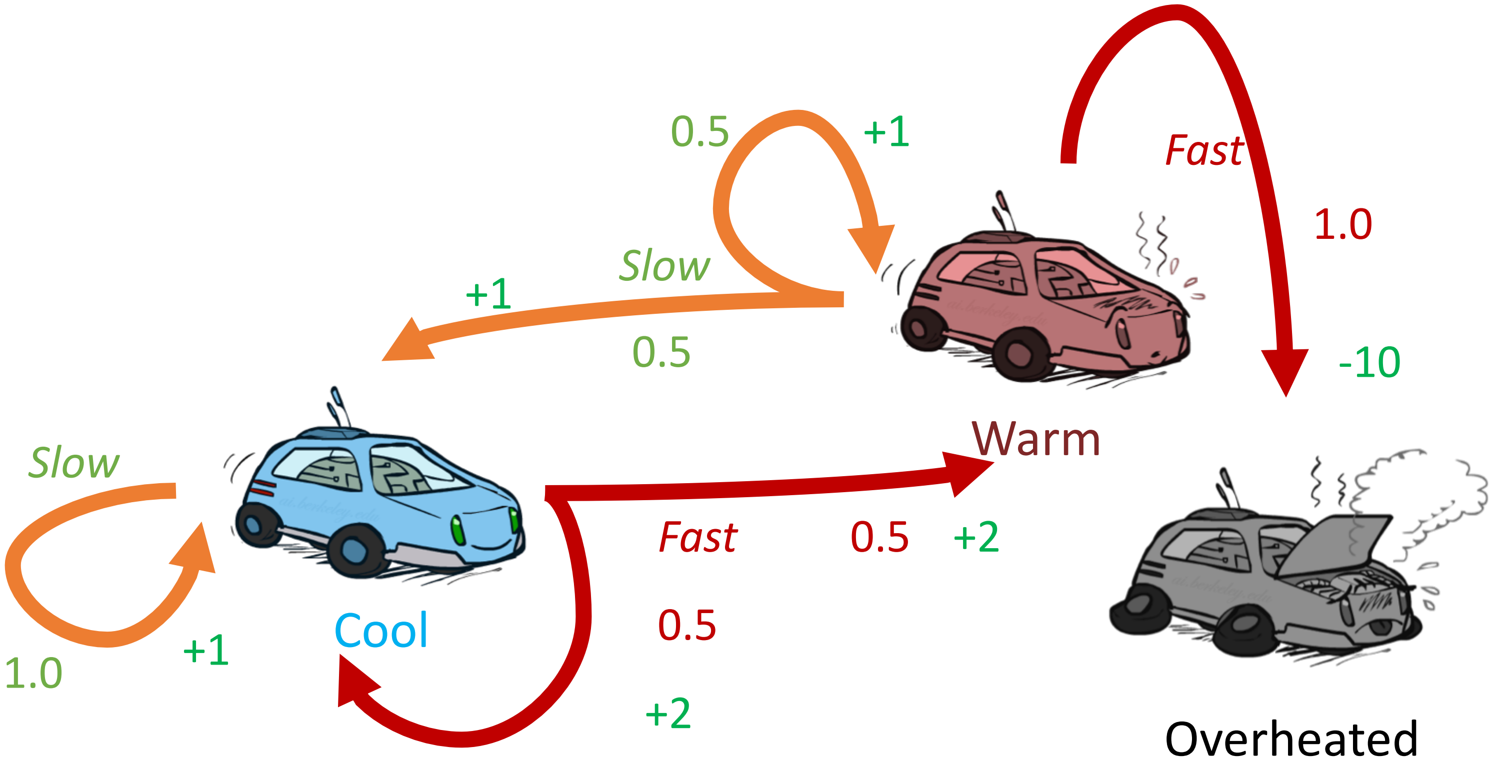
- Optimal policy $\pi^*$: A policy is one that maximizes expected return if followed

# Outline

- Sequential decision making problems
- Markov decision processes
- MDP examples

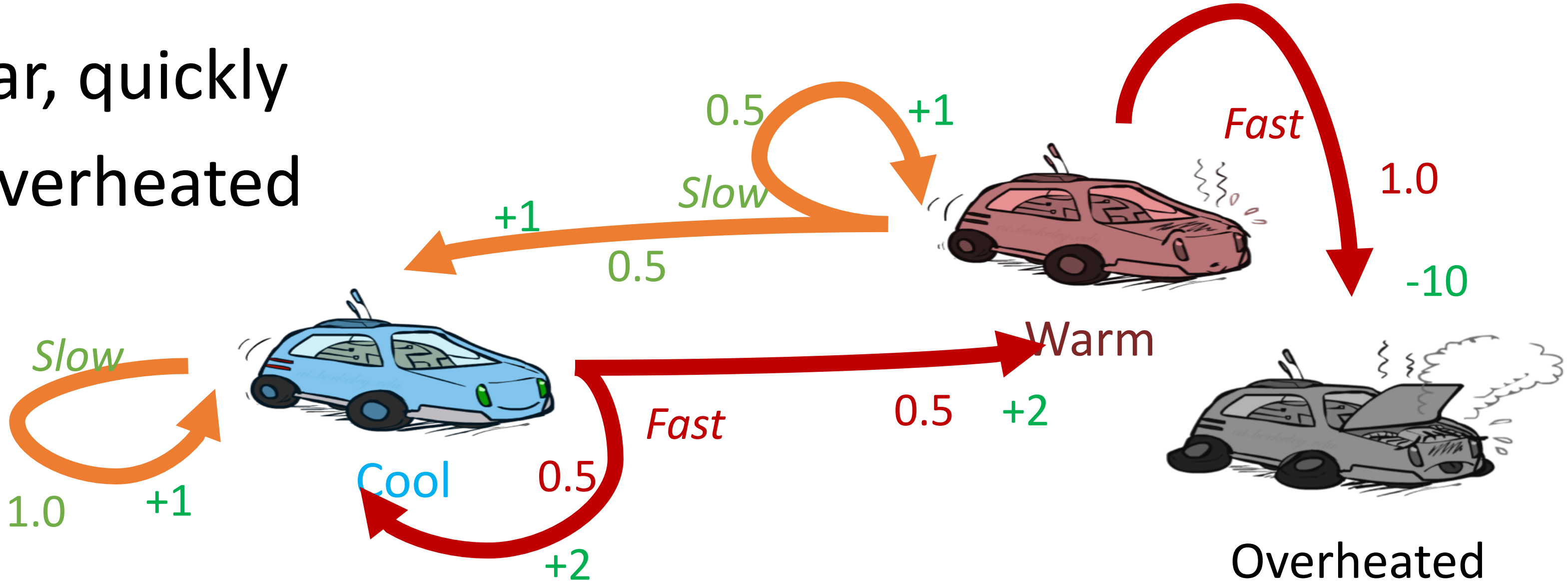
# Example: Racing

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



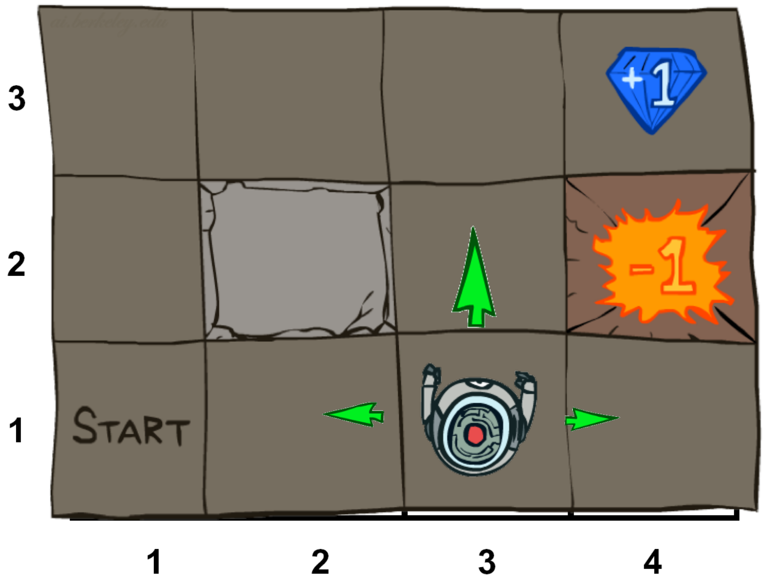
# Exercise: Write down transition and reward model

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

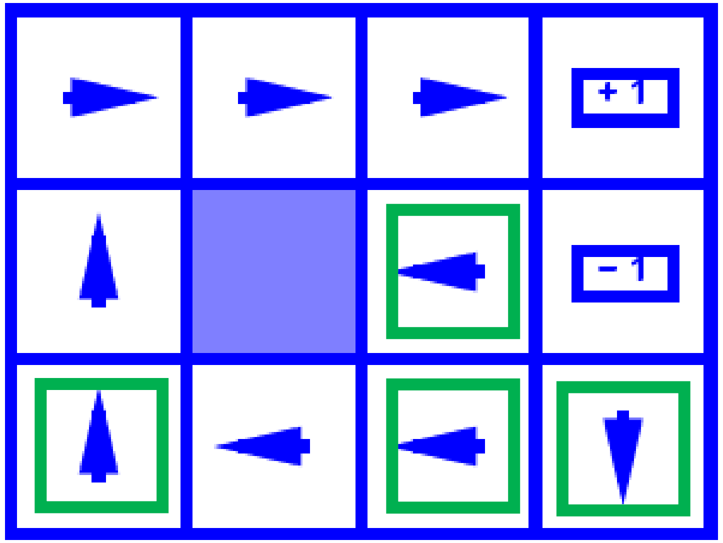


# Grid World with Uncertainties

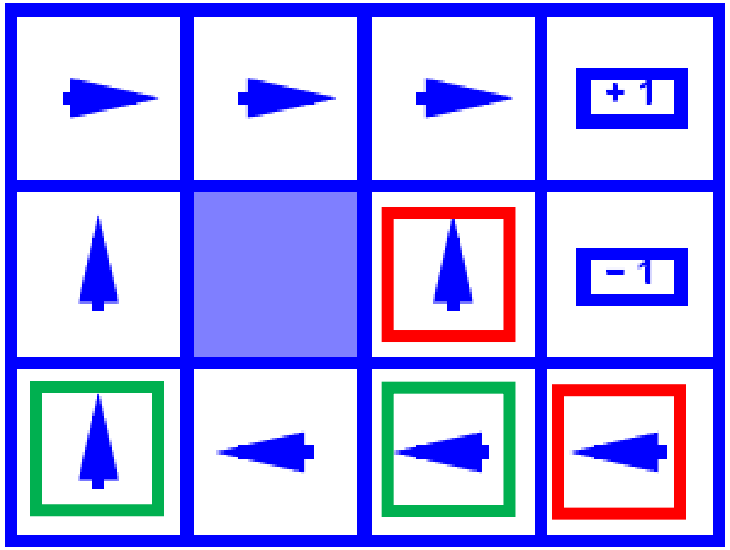
- States: (1, 1)→ (4,3)
- Actions: Up, Down, Right, Left
- Transition Model (4 matrix of 12 X 12): 80%, 10%, 10%.
- Reward Model (vector of 12 X 4):
- Gamma: predefined



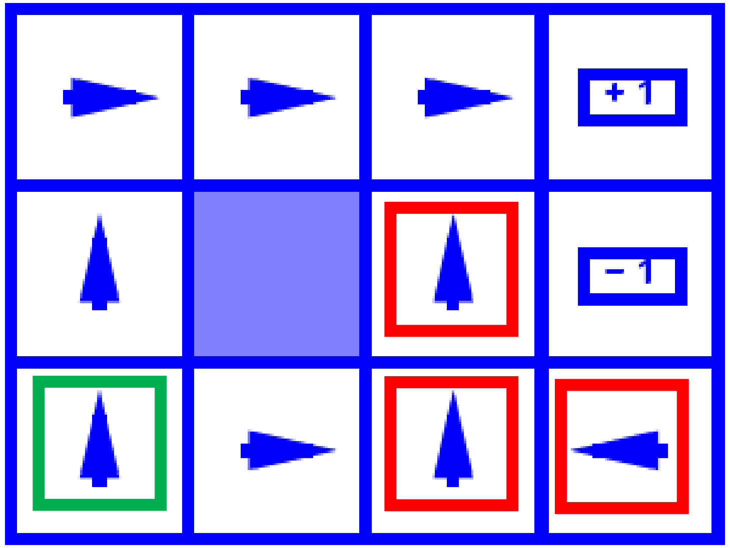
# How reward affects?



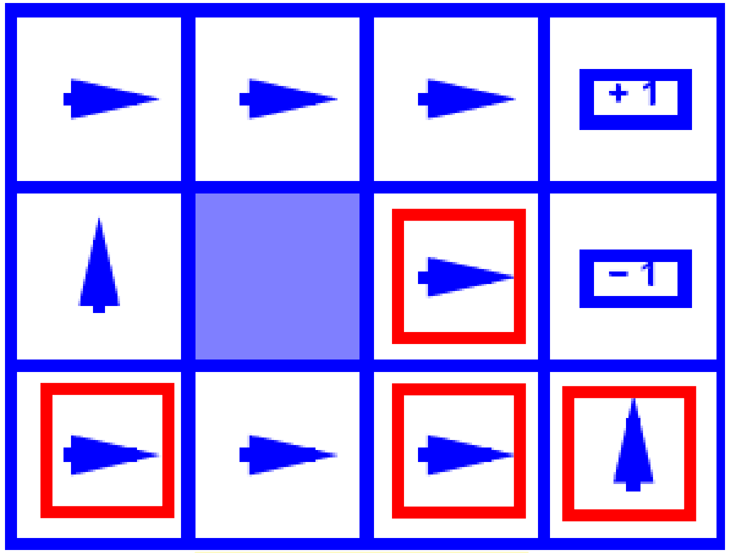

$R(s,a) = -0.01$




$R(s,a) = -0.03$




$R(s,a) = -0.4$




$R(s,a) = -2.0$

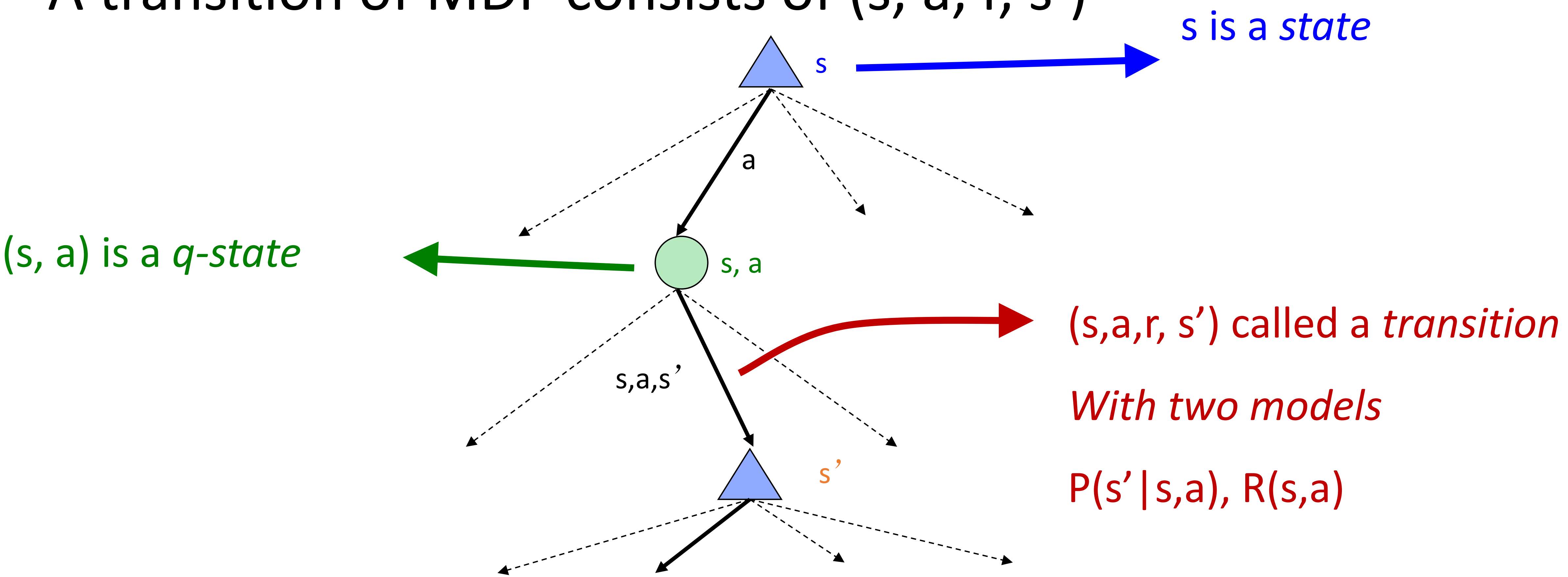
# Outline

- Sequential decision making problems
- Markov decision processes
- MDP examples

- Value Iteration for MDPs

# MDP Search Trees

- Each MDP state projects an expectimax-like search tree
- A transition of MDP consists of (s, a, r, s')

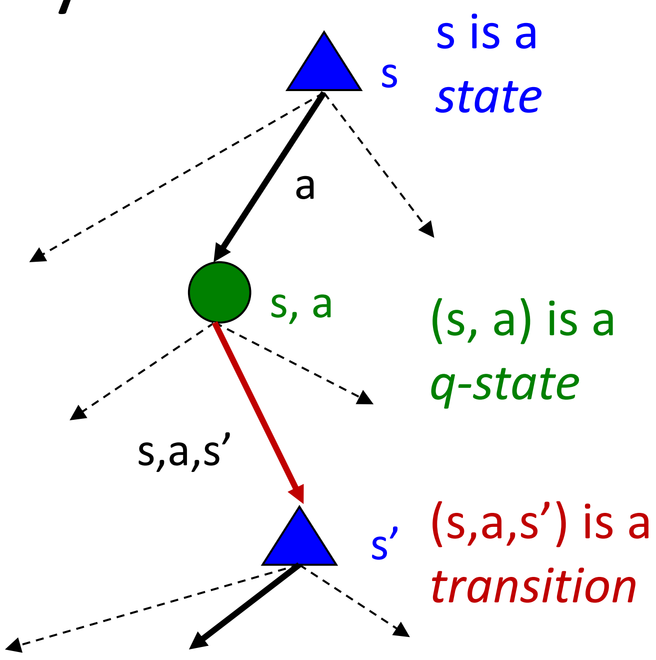


# Optimal Quantities

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

- The value of a q-state (s,a):
  $Q^*(s,a)$ = expected return starting out having taken action $a$ from state s and (thereafter) acting optimally

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



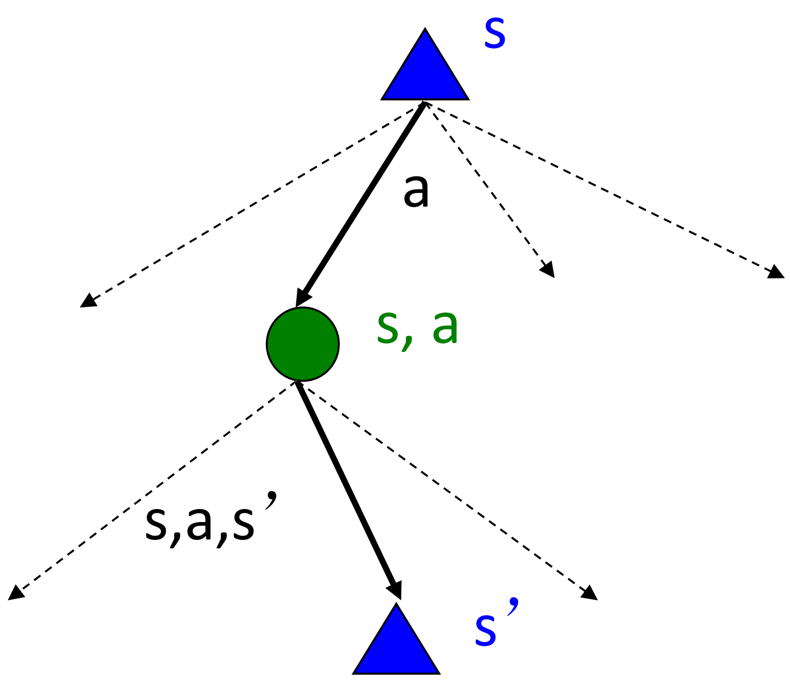
# Bellman equations for solving MDPs

- Bellman equation:

$$V^*(s) = \max_{a\in A(s)} Q^*(s,a)$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s')$$

$$V^*(s) = \max_{a\in A(s)} \{R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s')\}$$




- Fundamental operation: compute the value of a state
  - Expected return under optimal action
  - Average sum of (discounted) rewards

# Value Iteration (Bellman Update Equation)

- Start with $V_0(s) = 0$
- Given vector of $V_k(s)$ values:

$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s')\}$$

- Repeat until convergence

收 斂

Complexity of each iteration: $O(S^2A)$

# Value Iteration Algorithm

**function** VALUE-ITERATION($mdp, \epsilon$) **returns** a utility function
  **inputs**: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount $\gamma$
    $\epsilon$, the maximum error allowed in the utility of any state
  **local variables**: $U$, $U'$, vectors of utilities for states in $S$, initially zero
    $\delta$, the maximum change in the utility of any state in an iteration

  **repeat**
    $U \leftarrow U'$; $\delta \leftarrow 0$
    **for each** state $s$ in $S$ **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
      **if** $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  **until** $\delta < \epsilon(1-\gamma)/\gamma$
  **return** $U$

**Figure 17.4** The value iteration algorithm for calculating utilities of states. The termination condition is from Equation (17.8).

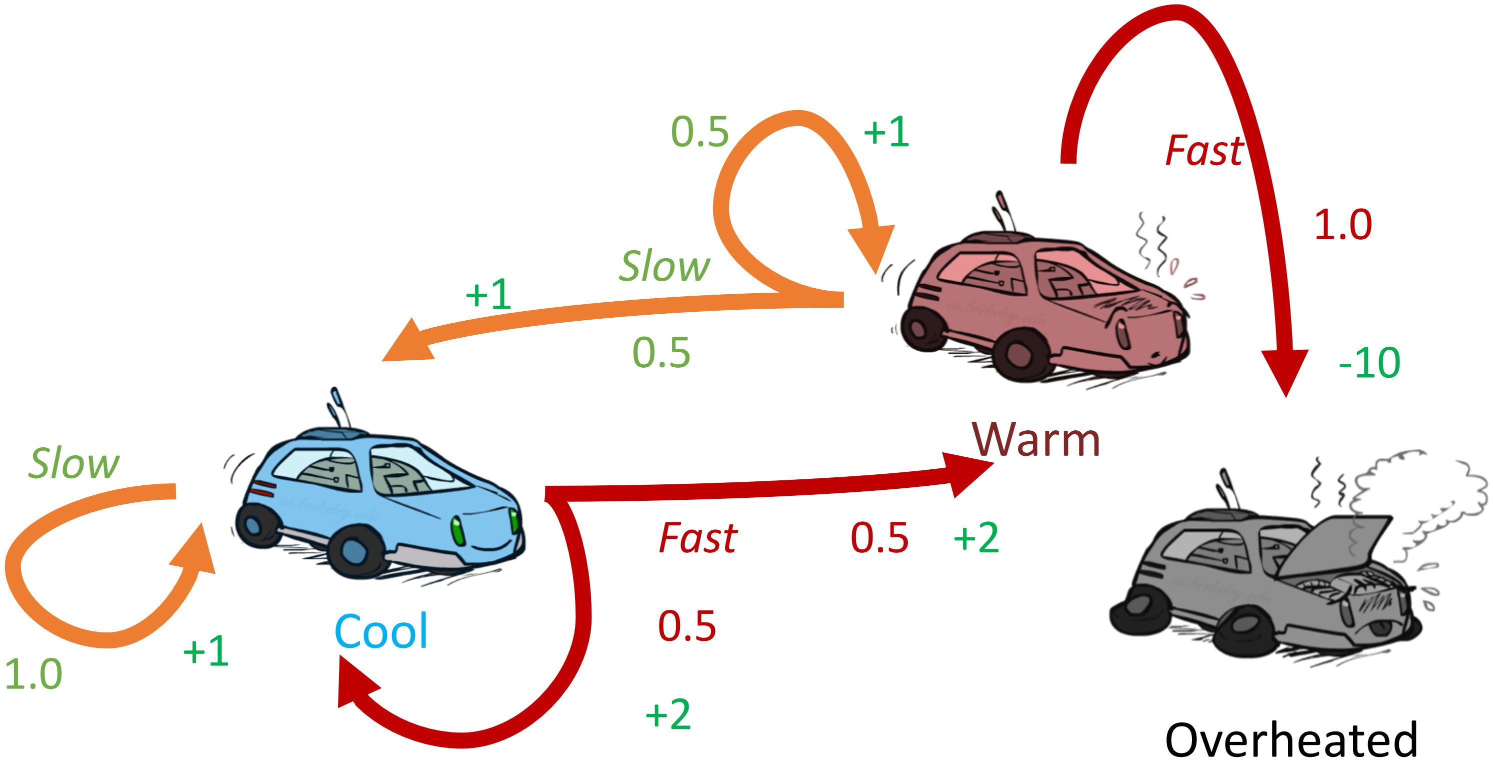
# Outline

- Sequential decision making problems
- Markov decision processes
- MDP examples

- Value Iteration for MDPs
- Value Iteration on expected-max tree

# Example: Racing

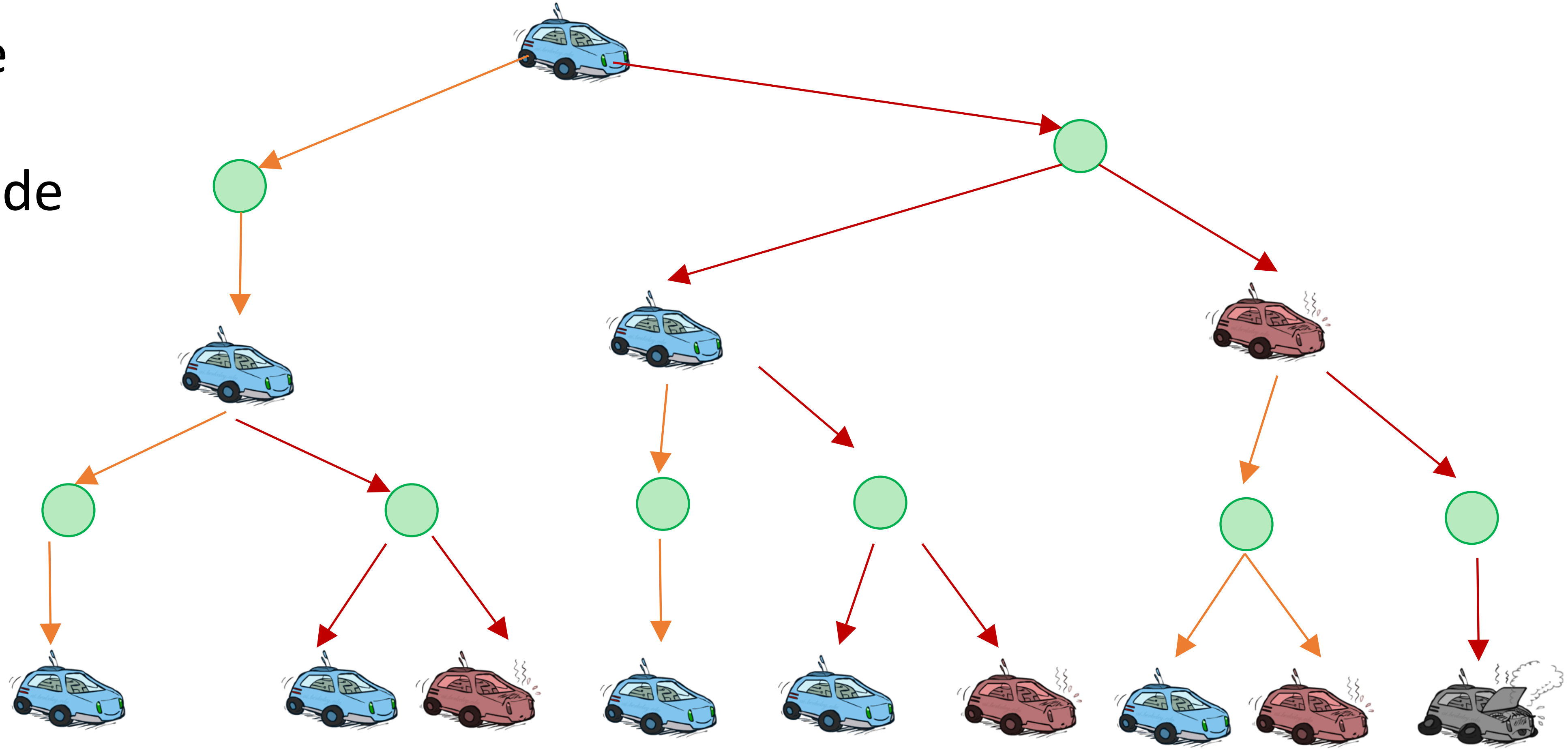
- A robot car wants to travel far, quickly
- Three **states**: Cool, Warm, Overheated
- Two **actions**: *Slow*, *Fast*



# Racing MDP Search Tree

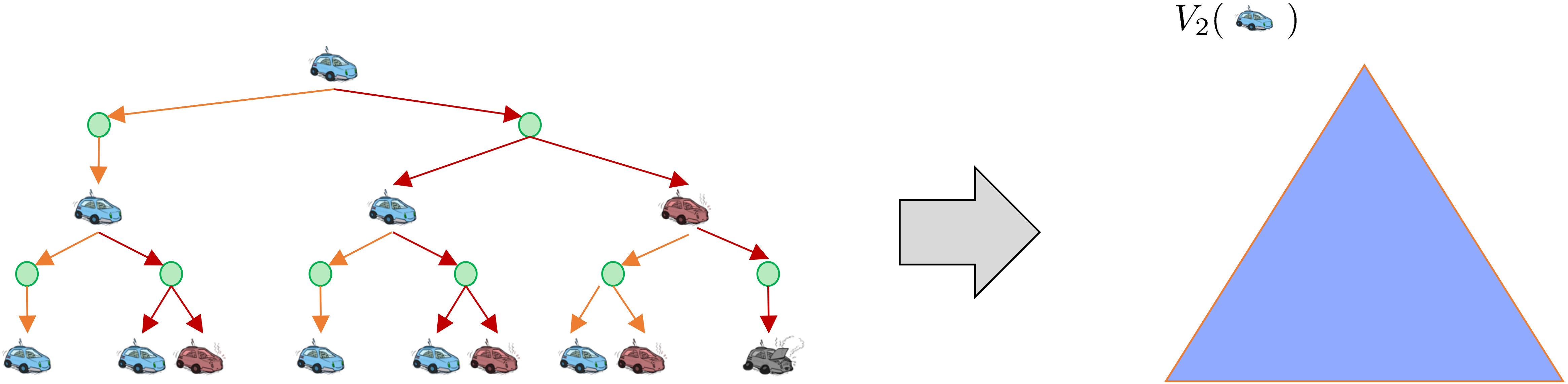
State Node

Chance Node

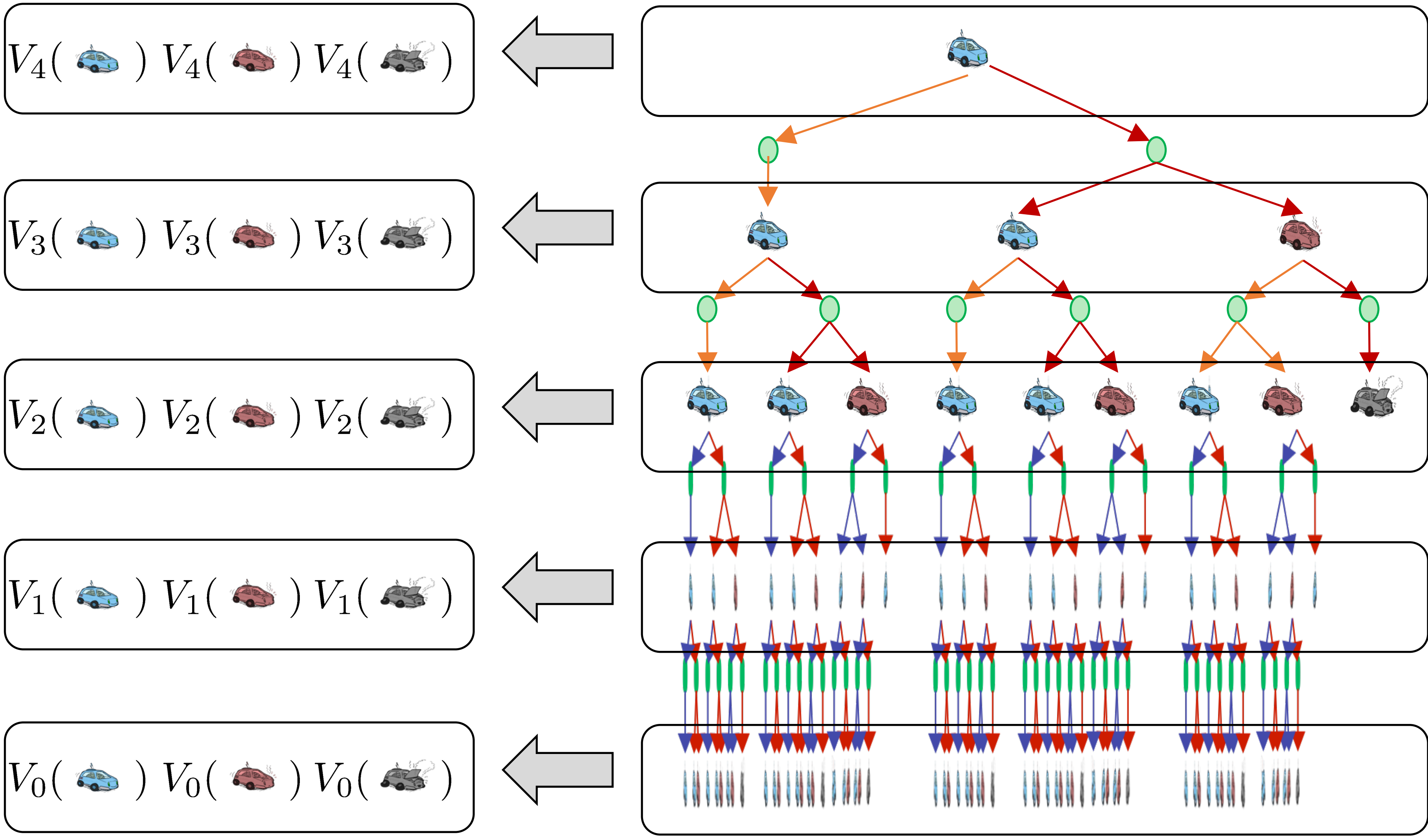


# Time-Limited Values

- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
  - Equivalently, it's what a depth-k expectimax would give from s



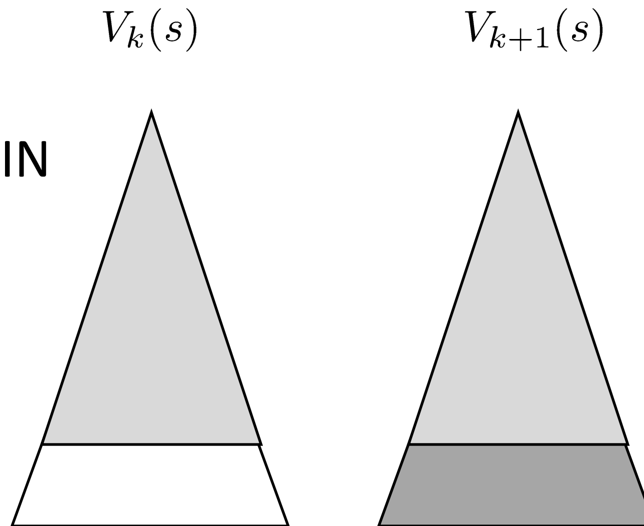
# Computing Time-Limited Values



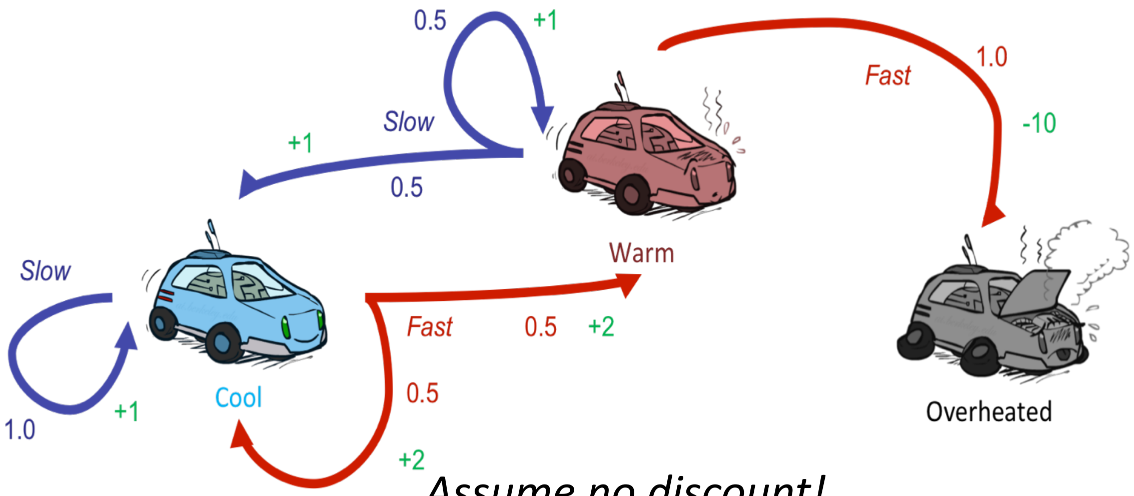

**Equals to:** $V_{k+1}(s) \leftarrow \max_{a \in A(s)} \{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s')\}$

# Convergence*

- How do we know the $V_k$ vectors are going to converge?
- Case 1: If the tree has maximum depth M, then $V_M$ holds the actual un-truncated values
- Case 2: If the discount is less than 1
  - For any state $V_k$ and $V_{k+1}$ can be viewed as depth k+1 expectimax results in nearly identical search trees
  - The difference is that on the bottom layer, $V_{k+1}$ has actual rewards while $V_k$ has zeros
  - That last layer is at best all $R_{MAX}$ and at worst $R_{MIN}$
  - But everything is discounted by $\gamma^k$
  - So $V_k$ and $V_{k+1}$ are at most $\gamma^k \max|R|$ different
  - So as k increases, the values converge



# Example: Value Iteration




*Assume no discount!*

$\gamma = 1$

| Transitions (s, a, s') | Transition Probability | reward |
|---|---|---|
| (cool, slow, cool) | 1 | 1 |
| (warm, slow, cool) | 0.5 | 1 |
| (warm, slow, warm) | 0.5 | 1 |
| (cool, fast, cool) | 0.5 | 2 |
| (cool, fast, warm) | 0.5 | 2 |
| (warm, fast, overheated) | 1 | -10 |

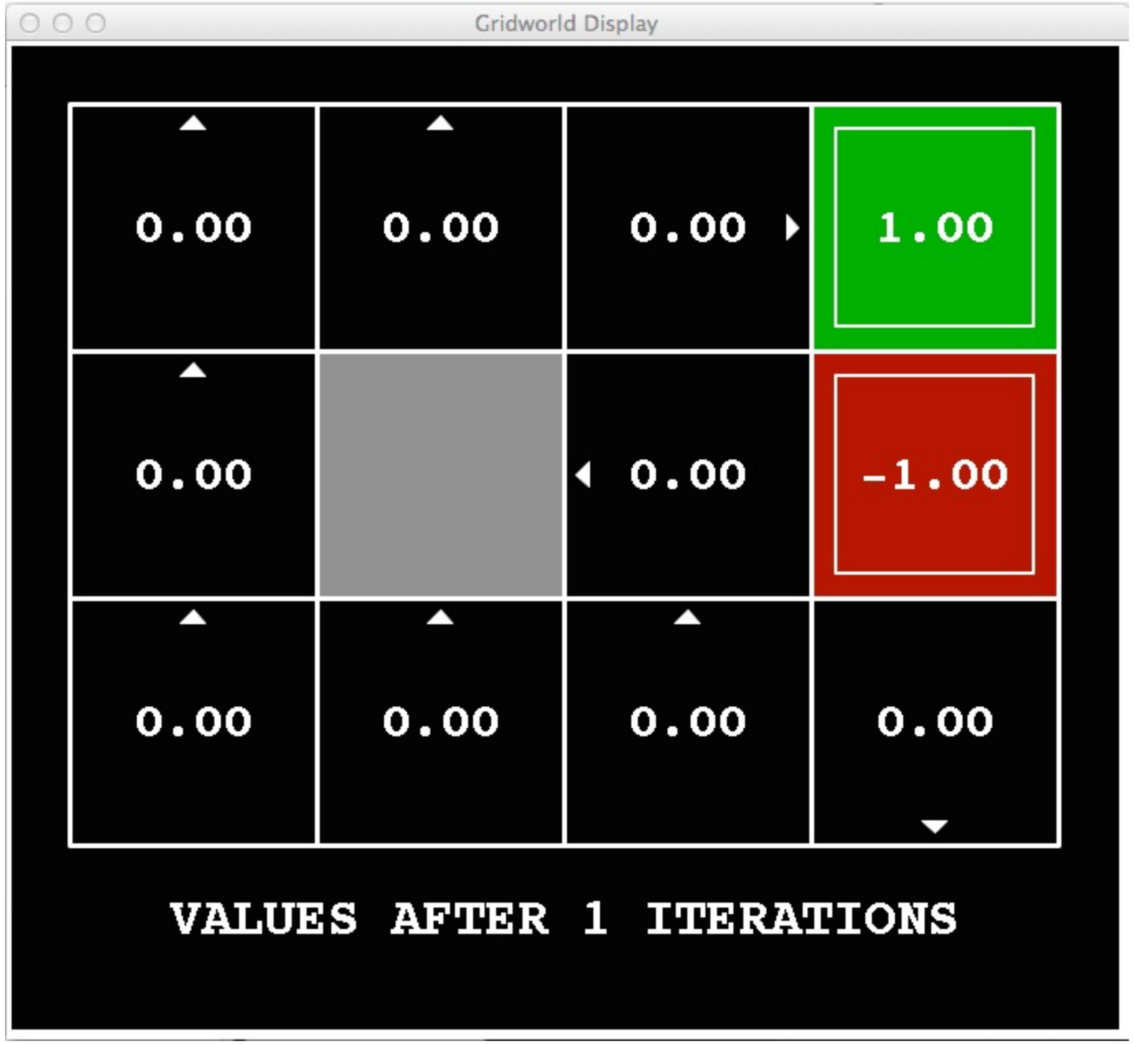
$V_2$

$V_1$

$V_0$ | 0 | 0 | 0 |

$$V_{k+1}(s) \leftarrow \max_{a \in A(s)} \{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s')\}$$

# k=1




Noise = 0.2
Discount = 0.9

- *80% of the time, each action achieves the intended direction.*
- *20% of the time, each action moves the agent at right angles to the intended direction.*
- *move north, 80% go to north, 10% to the west, 10% to the east*
- *Reward for (4,4) is 1, (3,4) is -1; 0 for others*

- ***Write-down the transition model, reward model***
- ***Compute values for all states in 10 steps.***

# Value Iteration Property on Grid World

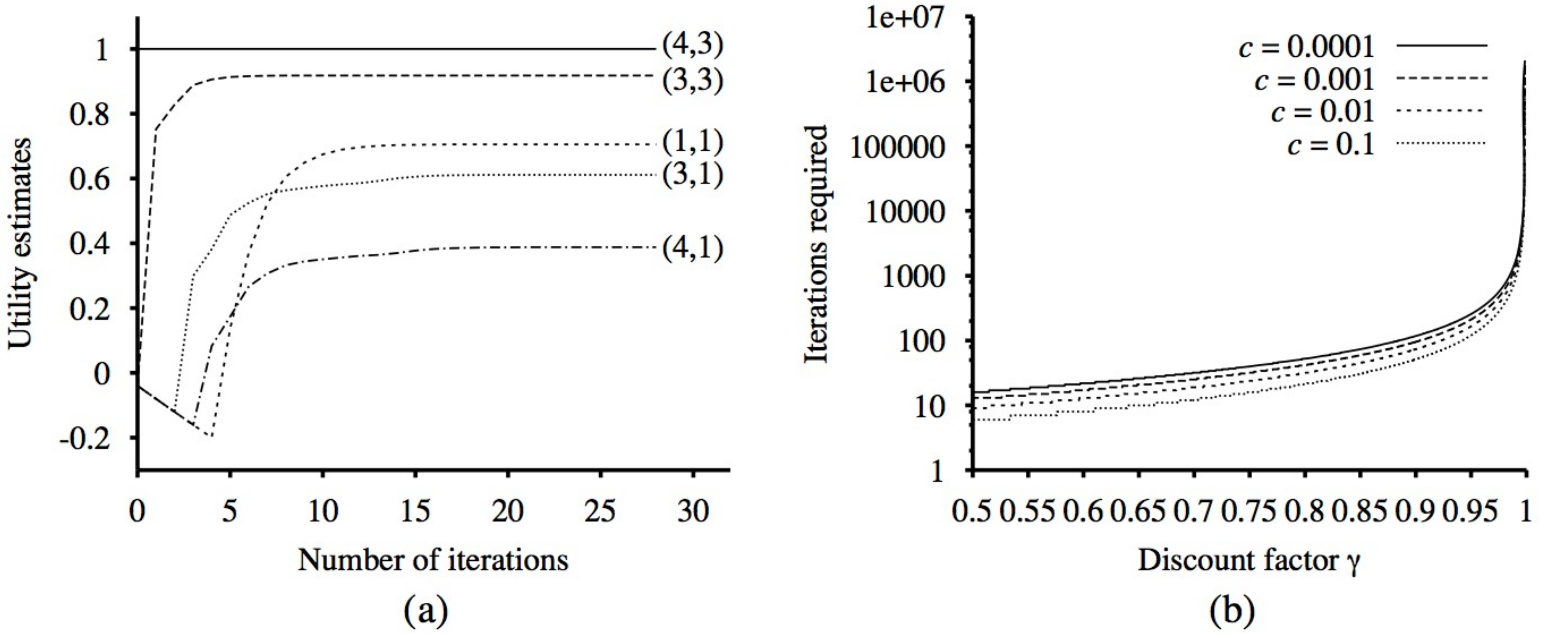
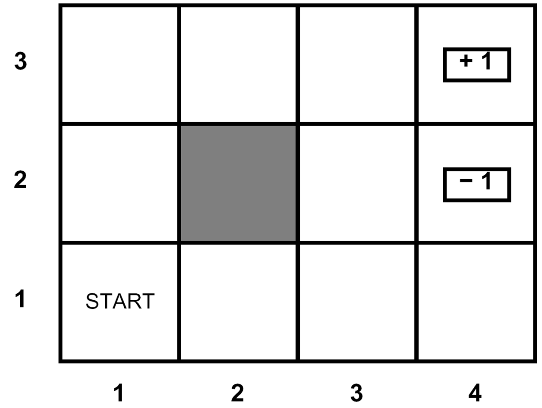



**Figure 17.5** (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations $k$ required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of $c$, as a function of the discount factor $\gamma$.



# Outline

- Sequential decision making problems
- Markov decision processes
- MDP examples

- Value Iteration for MDPs
- Value Iteration on expected-max tree

- Policy Evaluation of MDPs

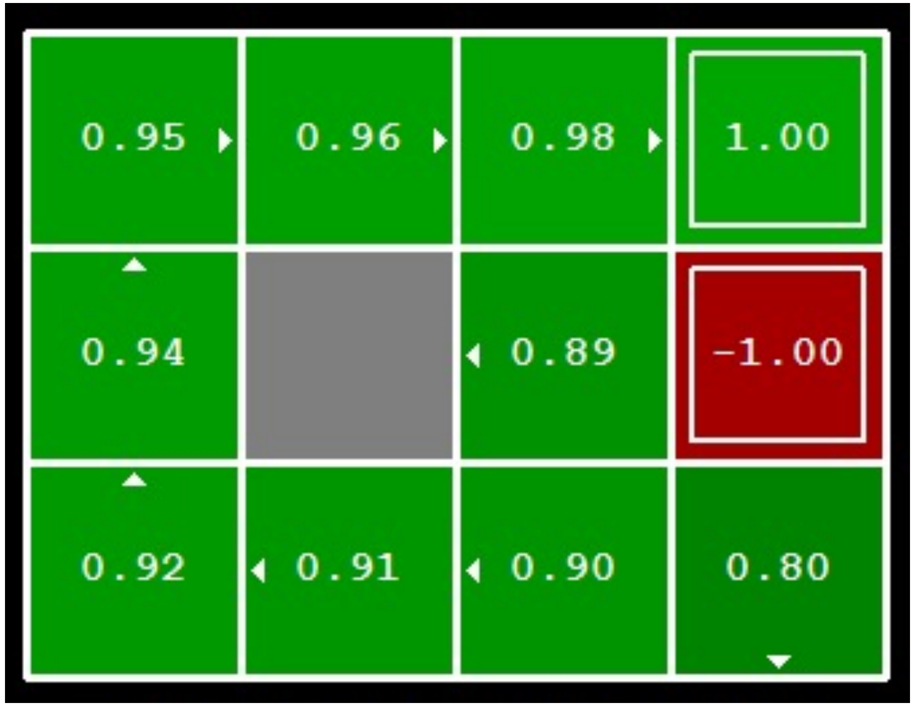
# Computing Actions from Values

- Let's imagine we have the optimal values V*(s)
- How should we act?

- We need to do an expectimax (one step)

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}}\{R(s,a) + \sum_{s'} P(s'|s,a)V^*(s')\}$$

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

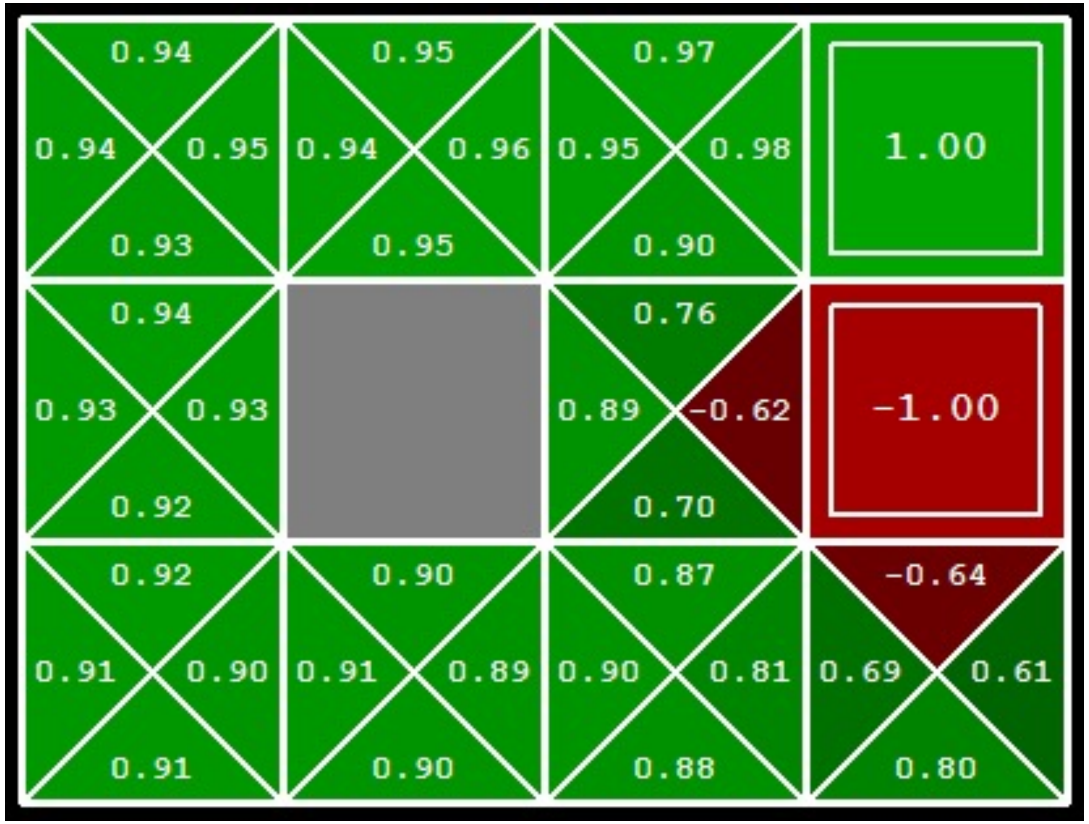


# Computing Actions from Q-Values

- Suppose we have the optimal q-values:

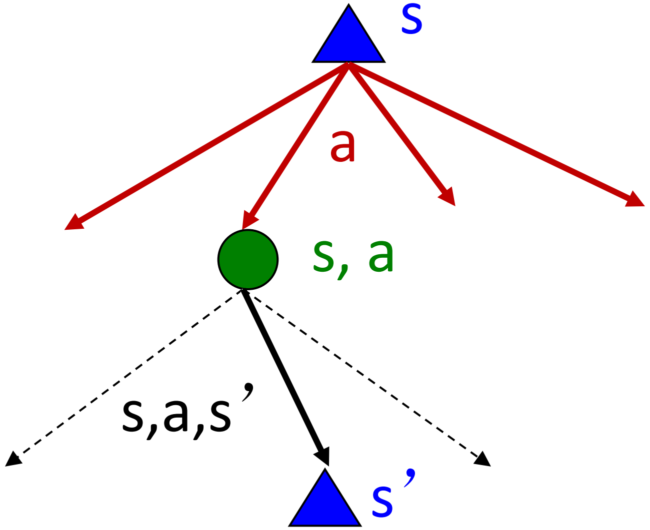
- How should we act?

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}}\, Q^*(s,a)$$

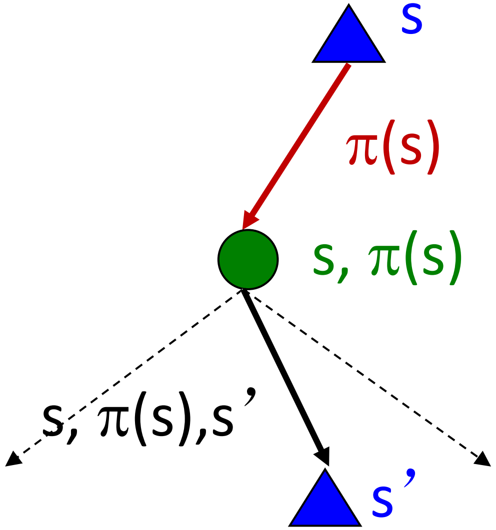


# Fixed Policies

Do the optimal action
(expectimax tree)




Do what $\pi$ says to do
(fixed policy tree)




- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state
  - And the tree's value would depend on which policy we fixed

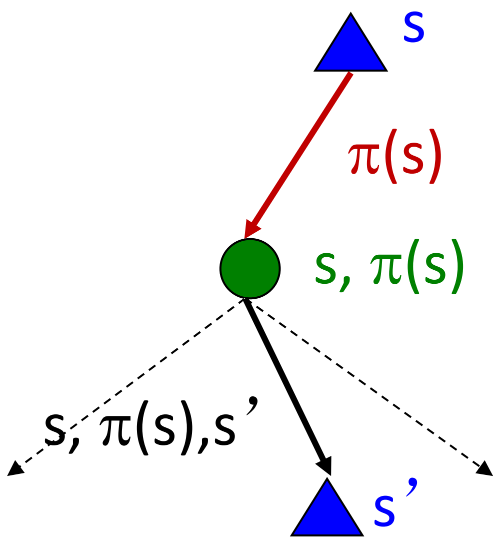
# Policy Evaluation

- Compute the value of a state $s$ under a fixed (generally non-optimal) policy

- Define the utility of a state s, under a fixed policy $\pi$:
  - $V^{\pi}(s)$ = expected total discounted rewards starting in s and following $\pi$

- Policy Evaluation via Bellman equation:

$$V^{\pi}(s) \leftarrow R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi}(s')$$



# Solving the Bellman Equation (policy)

- Iteration: Turn recursive Bellman equations into updates (like value iteration)

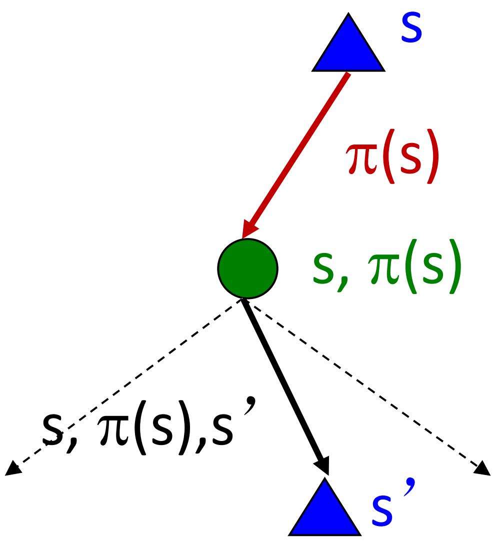
$$V_0^{\pi}(s) \leftarrow 0$$

$$V^{\pi}(s) \leftarrow R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi}(s')$$

Policy Iteration

- Efficiency: $O(S^2)$ per iteration

- Is the value optimal?



# Example: Policy Evaluation

Policy: Always Go Right


Policy: Always Go Forward

# Outline

- Sequential decision making problems
- Markov decision processes
- MDP examples

- Value Iteration for MDPs
- Value Iteration on expected-max tree

- Policy Evaluation of MDPs
- Policy Iteration for MDPs

# Policy Iteration

- Step 1: Policy evaluation: calculate utilities for some **fixed policy** until convergence.

$$V^{\pi}(s) \leftarrow R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi}(s')$$

- Step 2: Policy improvement (policy extraction): update policy using one-step look-ahead with **current value**

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \{R(s,a) + \sum_{s'} P(s'|s,a) V^*(s')\}$$

- Repeat steps until policy converges

- It’s still optimal. Can converge (much) faster under some conditions

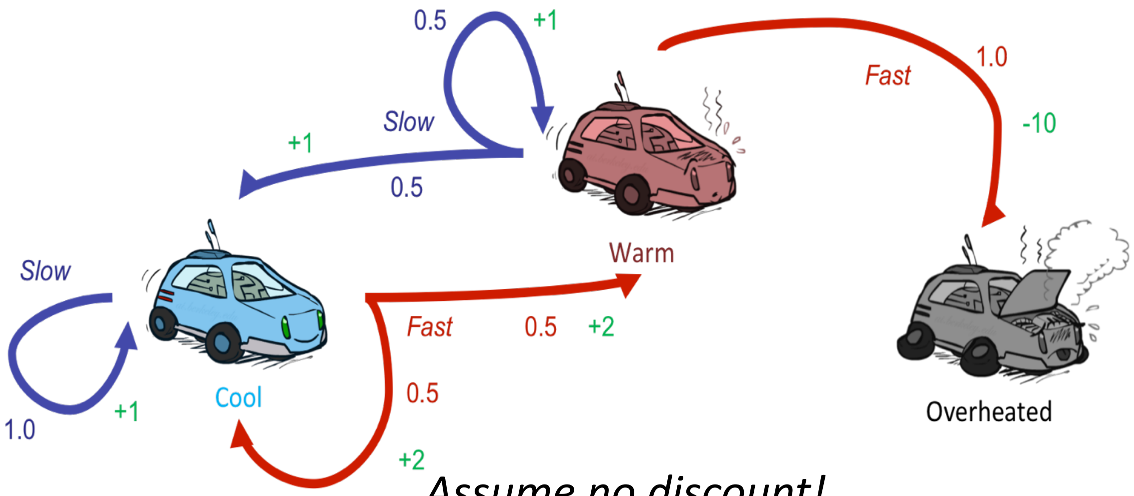
# Policy Iteration

**function** POLICY-ITERATION($mdp$) **returns** a policy
**inputs**: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$
**local variables**: $U$, a vector of utilities for states in $S$, initially zero
$\pi$, a policy vector indexed by state, initially random

**repeat**
$U \leftarrow$ POLICY-EVALUATION($\pi, U, mdp$)
$unchanged? \leftarrow$ true
**for each** state $s$ **in** $S$ **do**
**if** $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
$\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
$unchanged? \leftarrow$ false
**until** $unchanged?$
**return** $\pi$

**Figure 17.7** The policy iteration algorithm for calculating an optimal policy.

# Example: Policy Iteration




*Assume no discount!*

$\gamma = 1$

| Transitions (s, a, s') | Transition Probability | reward |
|---|---|---|
| (cool, slow, cool) | 1 | 1 |
| (warm, slow, cool) | 0.5 | 1 |
| (warm, slow, warm) | 0.5 | 1 |
| (cool, fast, cool) | 0.5 | 2 |
| (cool, fast, warm) | 0.5 | 2 |
| (warm, fast, overheated) | 1 | -10 |

$V_2^{\pi}$ (s)

$V_1^{\pi}$ (s)

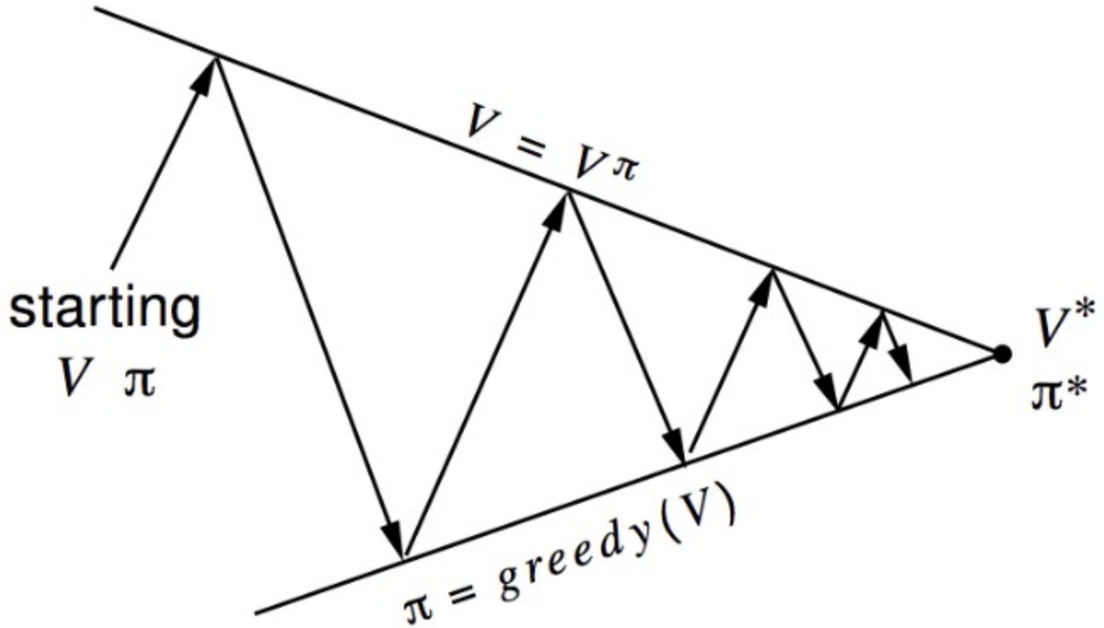
$V_0^{\pi}$ (s)

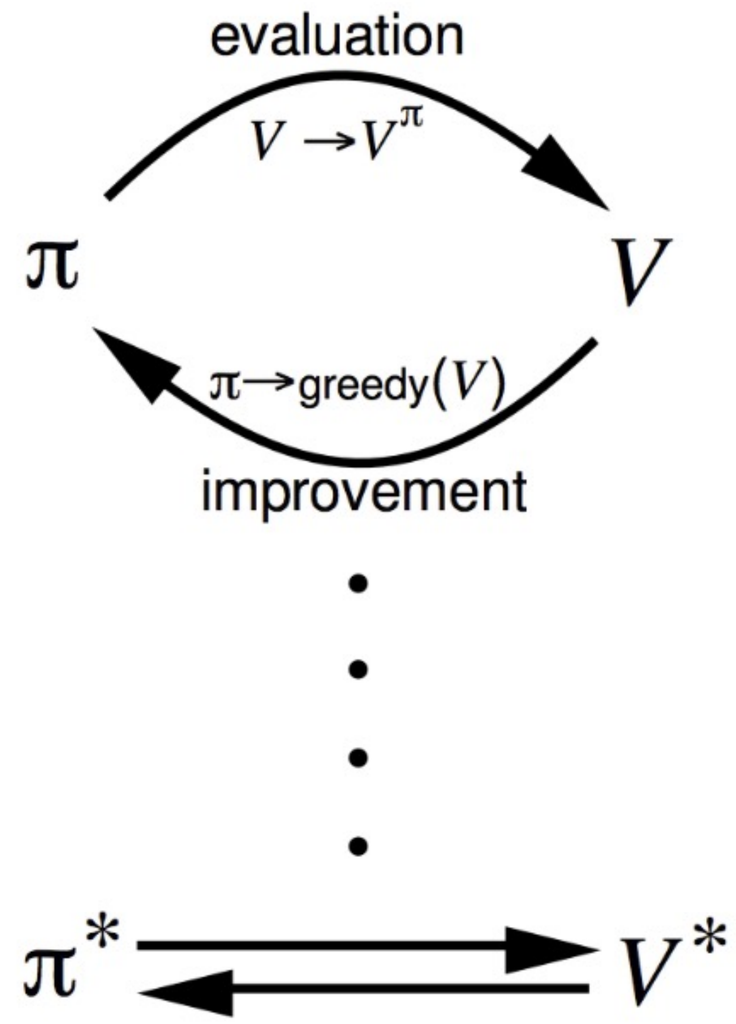
| | | |
|---|---|---|
| 0 | 0 | 0 |

$\pi = \{slow, slow\}$

$$V^{\pi}(s) \leftarrow R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi}(s')$$

# Policy Iteration




Policy evaluation Estimate $v_\pi$

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

# Modified Policy Iteration

- Does policy evaluation need to converge to $V^{\pi}$?
  - Or should we introduce a stopping condition
    - E.g. epsilon-convergence of value function
    - Or simply stop after k iterations of iterative policy evaluation?

- Why not update policy every iteration? i.e. stop after k = 1
  - This is equivalent to value iteration.

# Comparison

- Both value iteration and policy iteration compute optimal values

- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but use greedy action for value updating.

- In policy iteration:
  - We do several passes that update utilities with fixed policy (only one action at each pass)
  - After the policy evaluation, a new policy is chosen (a value iteration pass)
  - The new policy will be better

- **Both are dynamic programing-based approaches**

# Outline

- Sequential decision making problems
- Markov decision processes
- MDP examples

- Value Iteration for MDPs
- Value Iteration on expected-max tree

- Policy Evaluation of MDPs
- Policy Iteration for MDPs

- More about DP based solutions

# Bellman Equation (policy) in Matrix Form

- The Bellman equation can be expressed concisely using matrices

$$v = \mathrm{R} + \gamma P v$$

- Where v is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

# Solving the Bellman Equation (policy)

- The Bellman equation (policy) is a linear equation
- It can be solved directly

$$
\begin{aligned}
v &= \mathcal{R} + \gamma \mathcal{P} v \\
(I - \gamma \mathcal{P})\, v &= \mathcal{R} \\
v &= (I - \gamma \mathcal{P})^{-1} \mathcal{R}
\end{aligned}
$$

- Computational complexity is $O(n^3)$ for n states
- Direct solution only possible for small MDPs

# Synchronous Dynamic Programming Algorithms

- Both value iteration and policy iteration used **synchronous backups**
  - i.e. all states are backup up in parallel

$$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{old}(s') \right)$$

$$v_{old} \leftarrow v_{new}$$

- **Asynchronous DP** backs up states individually, in any order
  - For each selected state, apply the appropriate backup
  - Can significantly reduce computation
  - Guaranteed to converge if all states continue to be selected

# Synchronous Dynamic Programming Algorithms

- Three simple ideas for asynchronous dynamic programming:
  - In-place dynamic programming
  - Prioritized sweeping
  - Real-Time Dynamic Programming

# In-place Dynamic Programming

- Synchronous value iteration stores two copies of value function
  - For all s in S

$$v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{old}(s') \right)$$

$$v_{old} \leftarrow v_{new}$$

- In-place value iteration only stores one copy of value function
  - For all s in S

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right)$$

# Prioritised Sweeping

- Use magnitude of Bellman error to guide state selection, e.g.

$$\left| \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right) - v(s) \right|$$

- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Can be implemented efficiently by maintaining a priority queue

# Real-Time Dynamic Programming

- Idea: only states that are relevant to agent
- Use agent's experience to guide the selection of states
- After each time-step
- Backup the state $S_t$

$$v(S_t) \leftarrow \max_{a \in \mathcal{A}} \left( \mathcal{R}^a_{S_t} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{S_t s'} v(s') \right)$$

- Focus the DP's backups onto parts of states that are most relevant to the agents

复旦大学大数据学院
School of Data Science, Fudan University

魏忠钰

# Markov Decision Processes

Data Intelligence and Social Computing Lab (DISC)

November 9th, 2021