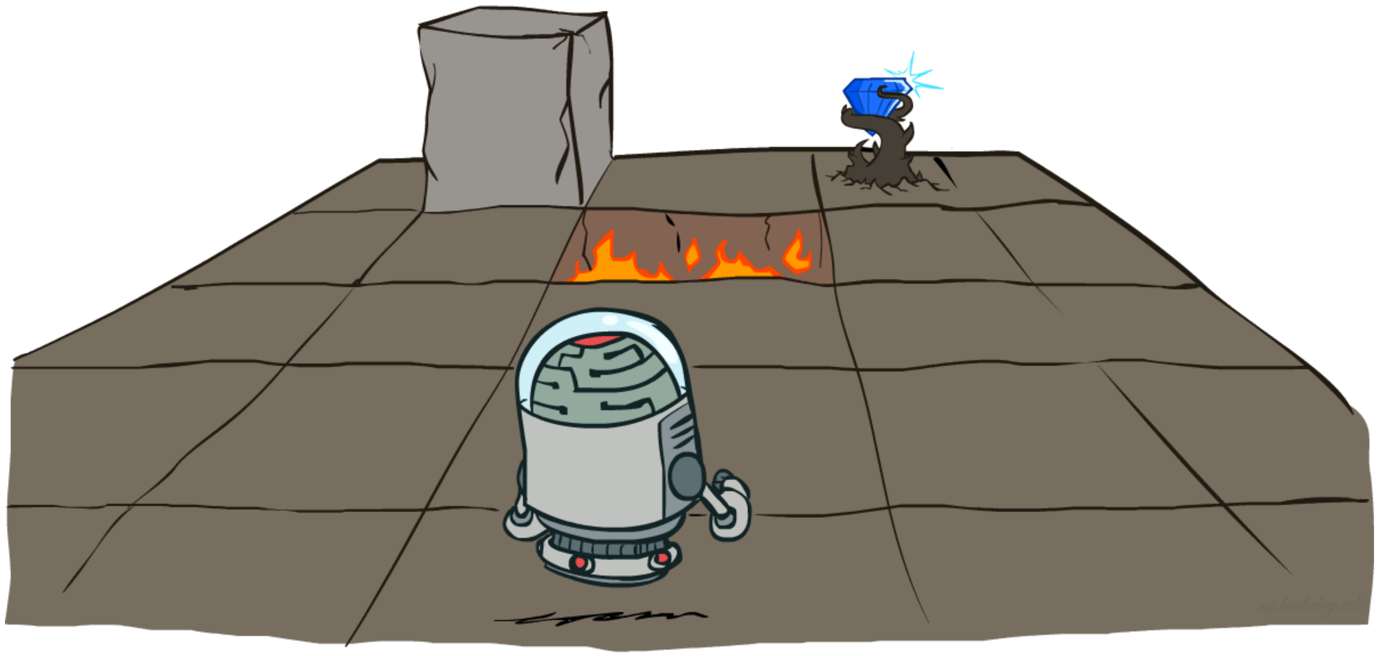
Markov Decision Processes



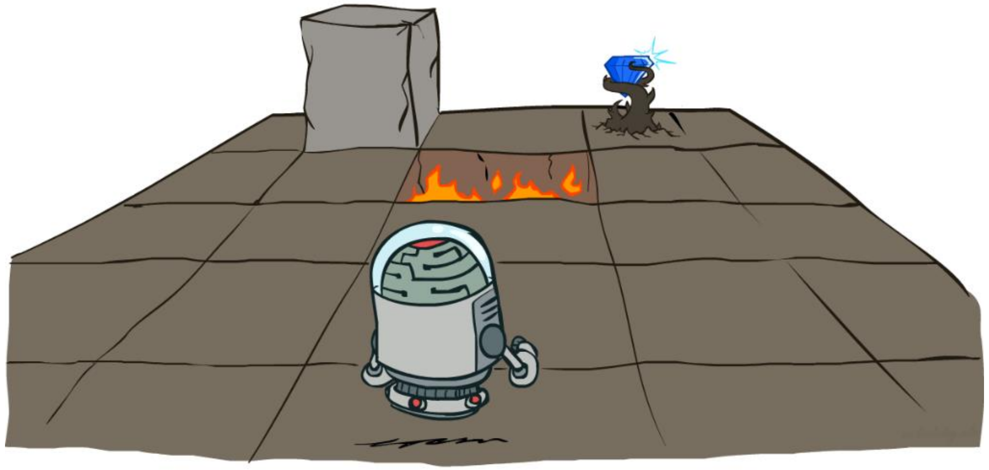
These slides were adapted from the MDP lecture slides of cs188 developed at UC Berkeley, http://ai.berkeley.edu.

Santa Clara University

Stochastic Environment

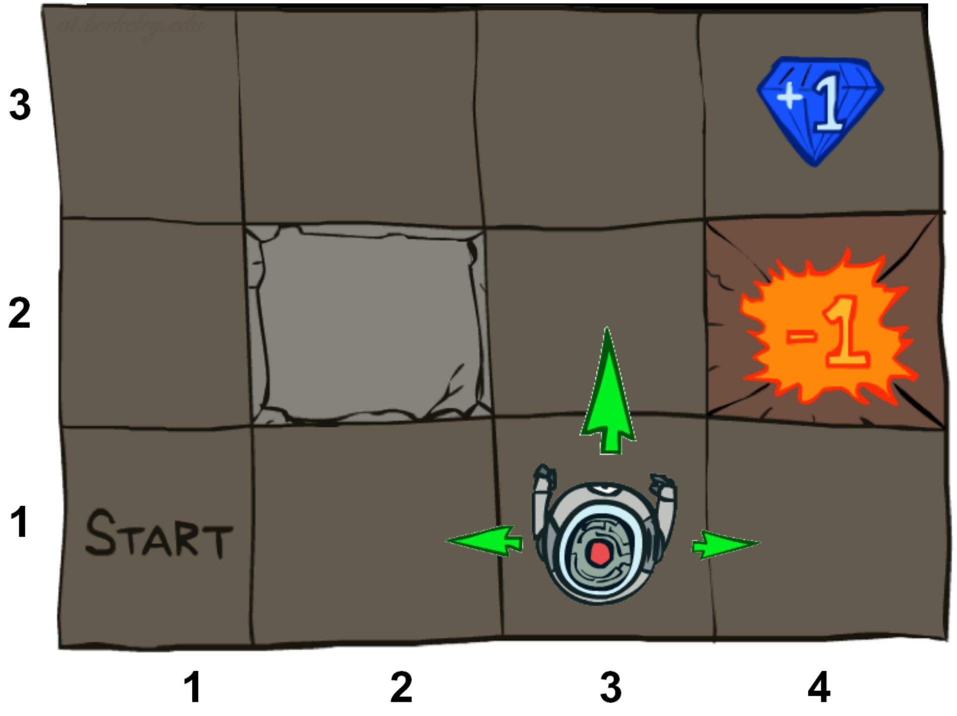
Motivation of Markov Decision Processes

- Another way to solve the decision-making problem for stochastic games
- Sometimes it's difficult to use the Minimax algorithm to determine actions, for example
  - When the game tree is endless
  - When the time spent to finish the game (finish the task) also affects the payoff



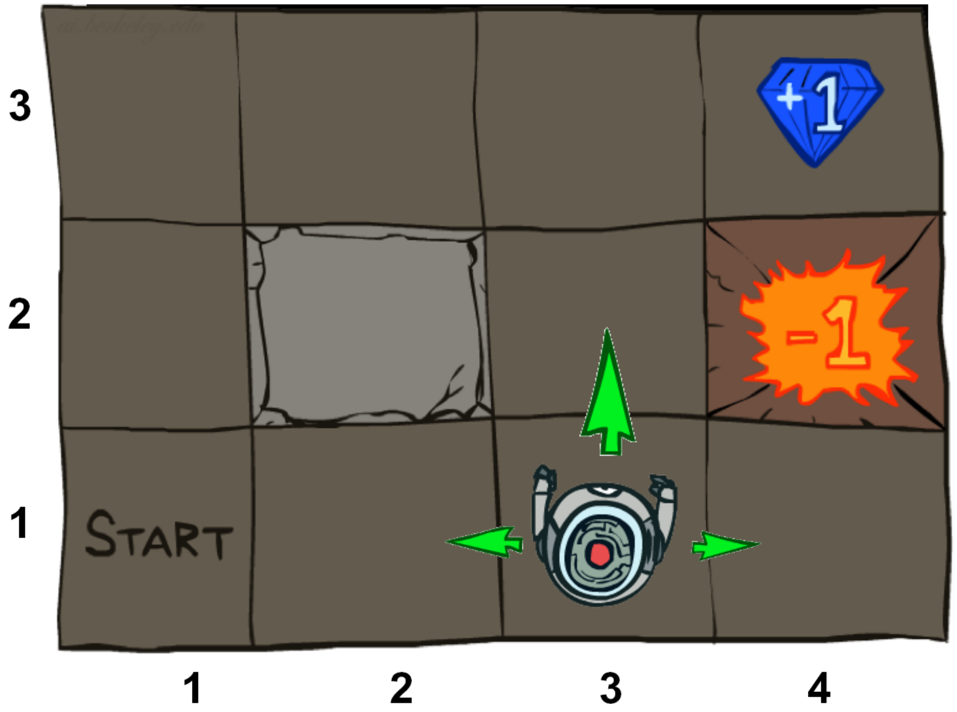
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- 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



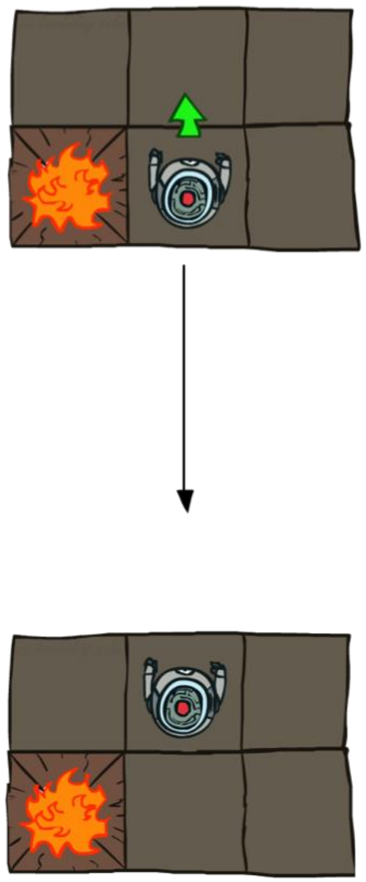
Example: Grid World

- The agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards by the end of the game

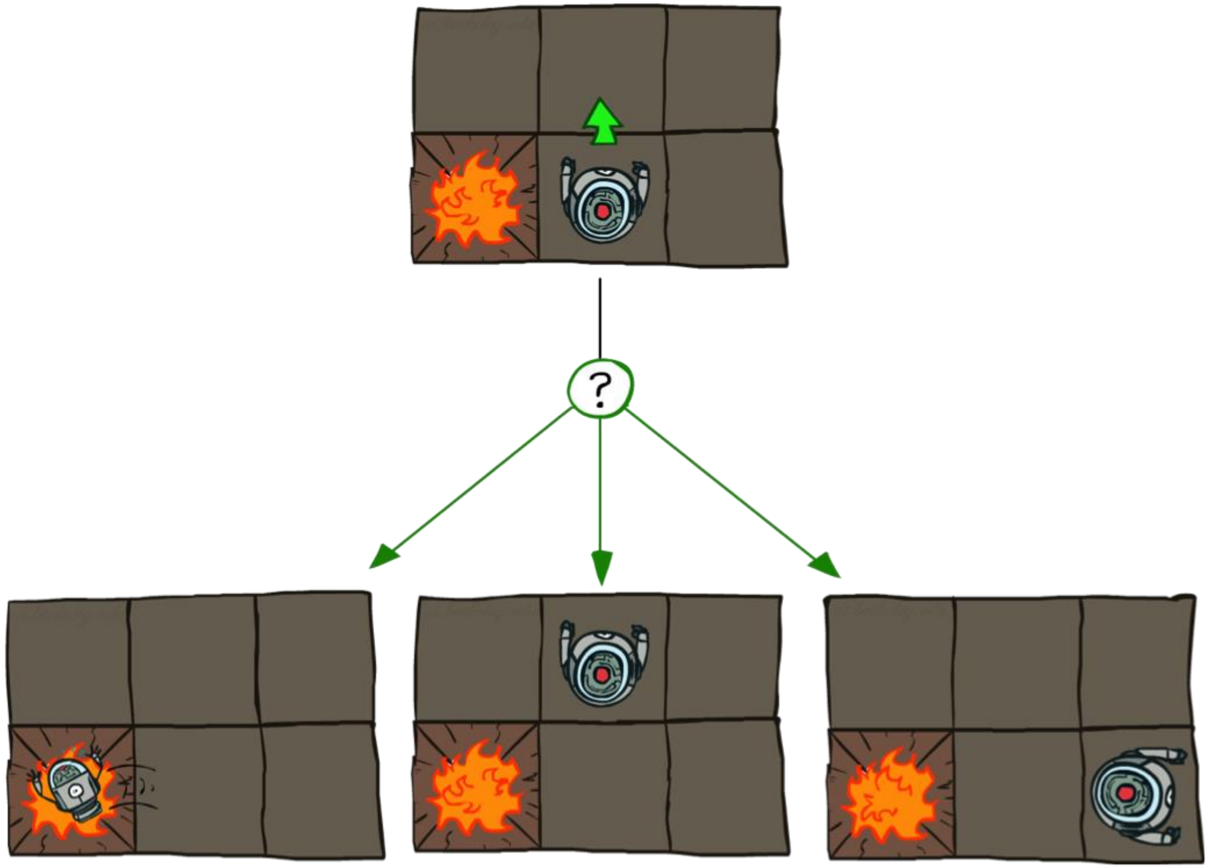


Grid World Actions

Deterministic Grid World



Stochastic Grid World

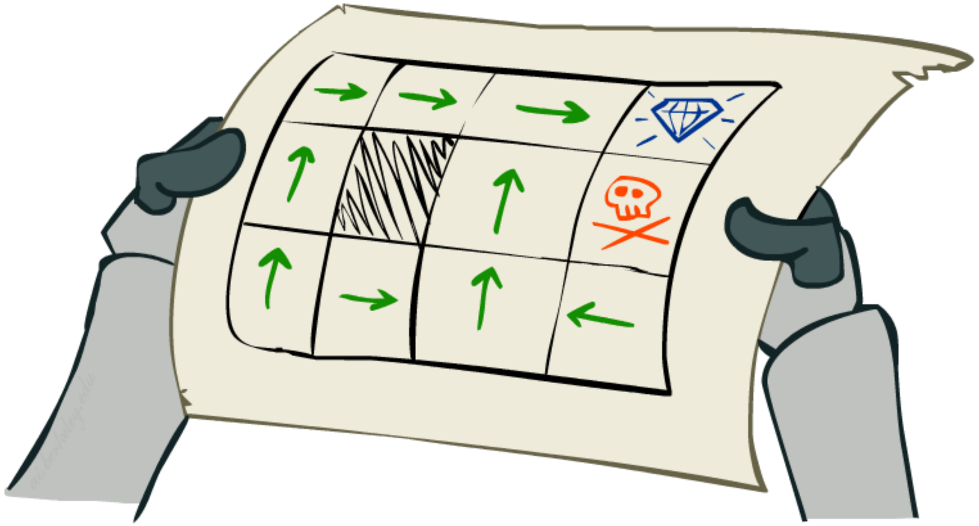


These slides were adapted from the MDP lecture slides of cs188 developed at UC Berkeley, http://ai.berkeley.edu.

The Objective

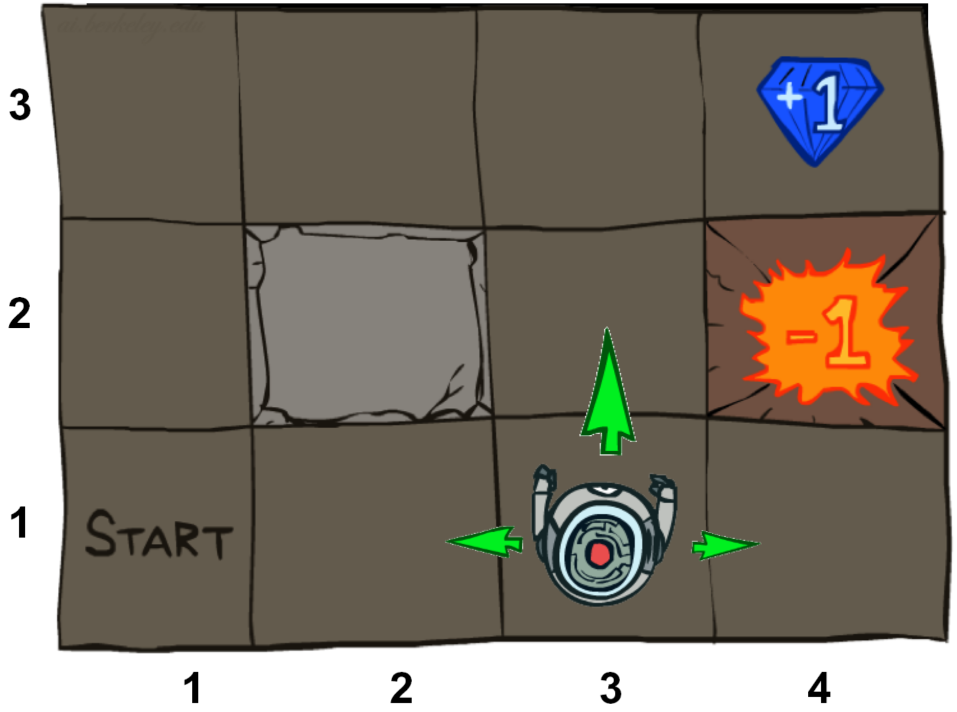
- In deterministic single-agent search problems, we wanted an optimal plan, or a sequence of actions, that takes the agent from the start to a goal state

- For a Markov Decision Process, we want to determine **the best action** for each state in the state space



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')
    - Probability that taking action a from state s leads to state s', i.e., P(s'| s, a)
    - Also called the model or the dynamics



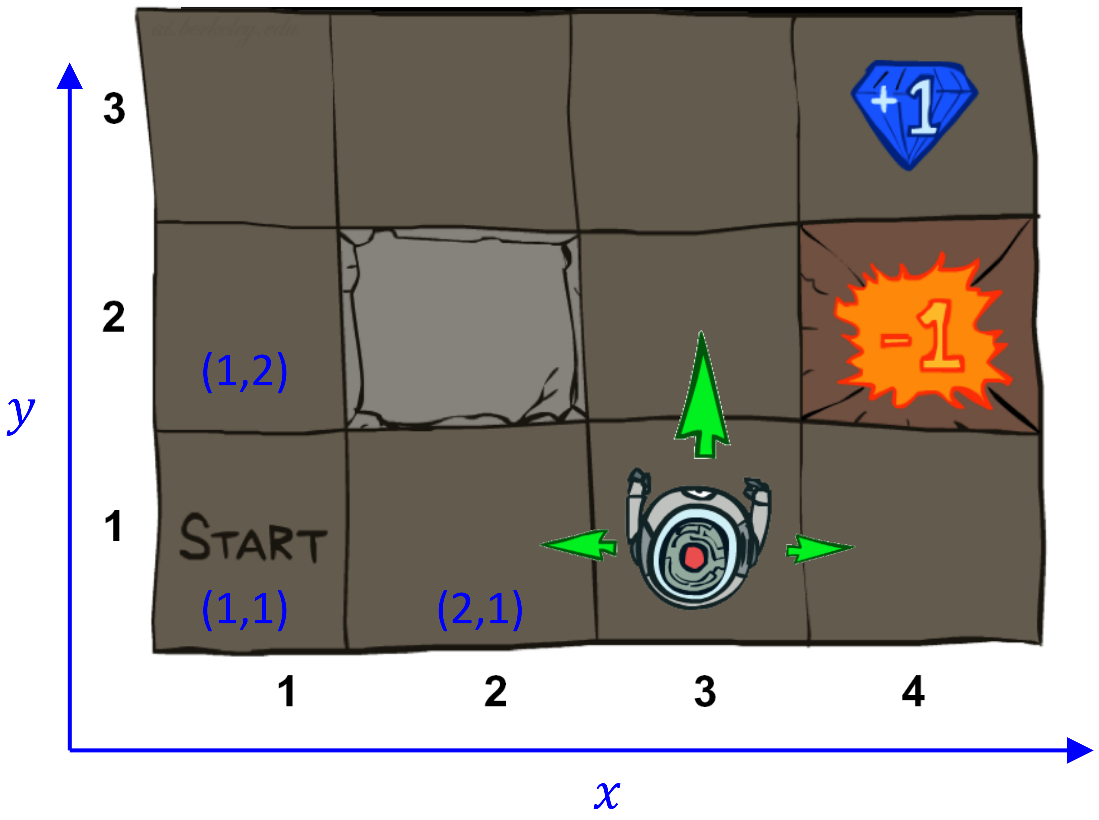
These slides were adapted from the MDP lecture slides of cs188 developed at UC Berkeley, http://ai.berkeley.edu.

Markov Decision Processes

- **Transition function T(s, a, s')**
  - For example
  - T(s=(1,1), a=North,s'=(1,2))=?
  - T(s=(1,1), a=North,s'=(2,1))=?
  - T(s=(1,1), a=North,s'=(1,1))=?

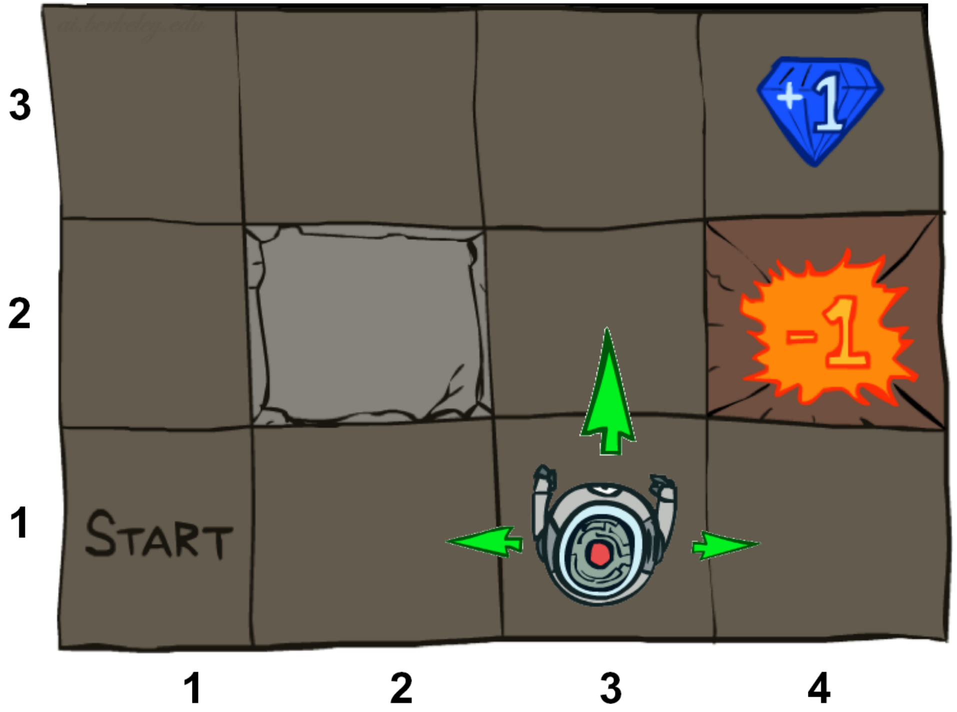
  - T(s=(1,1), a=East,s'=(2,1))=?
  - T(s=(1,1), a=East,s'=(1,2))=?
  - T(s=(1,1), a=East,s'=(1,1))=?

  - That is, for any possible combination (s,a,s'), we need to define T(s,a,s')



Markov Decision Processes

- Example: when the agent takes an action
  - 80% of time he lands in the desired square;
  - 20% of time he lands in the square in an orthogonal direction of the desired direction.



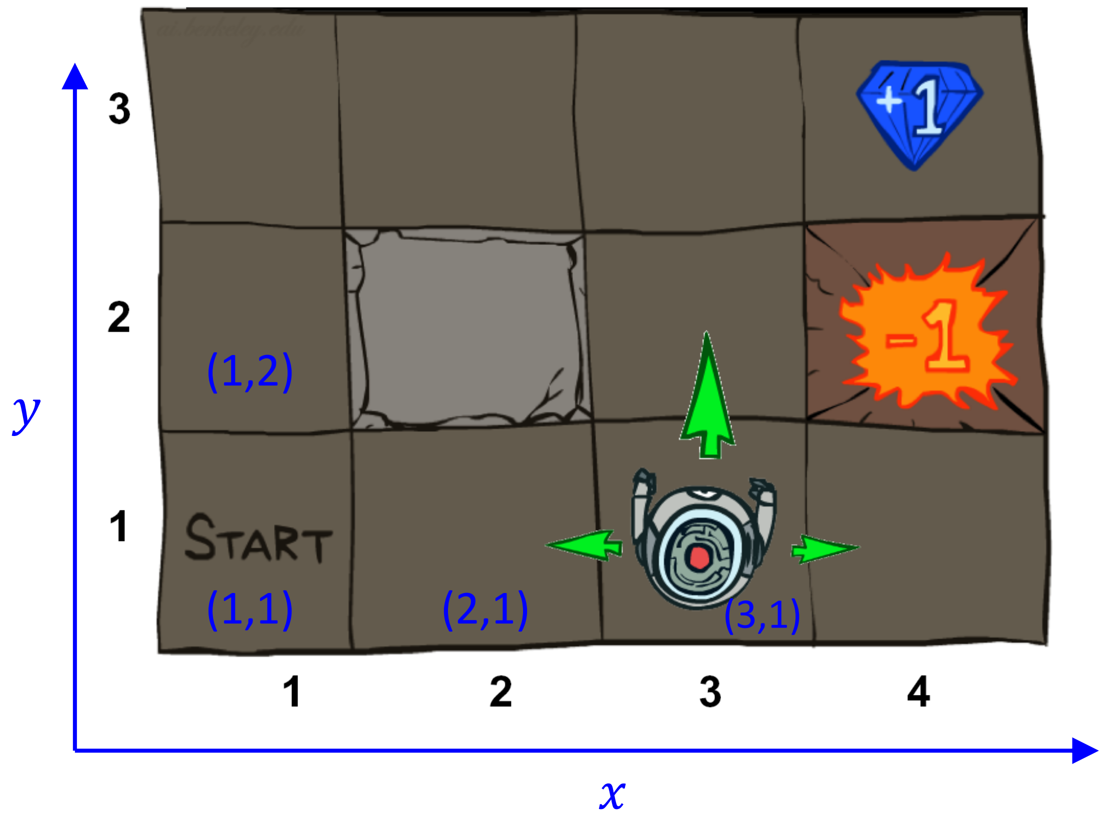
Markov Decision Processes

- What are the following transition probabilities? Let the location be (x,y)

  - T(s,North,s'), when s= (1,1), s'=(2,1)
  - Answer: 0.1

  - T(s,East,s'), when s=(1,1), s'=(2,1)
  - Answer: 0.8

  - T(s,East,s'), when s=(2,1), s'=(3,1)
  - Answer: 0.8
  - Where else can the agent land?
  - Answer: stay at (2,1)

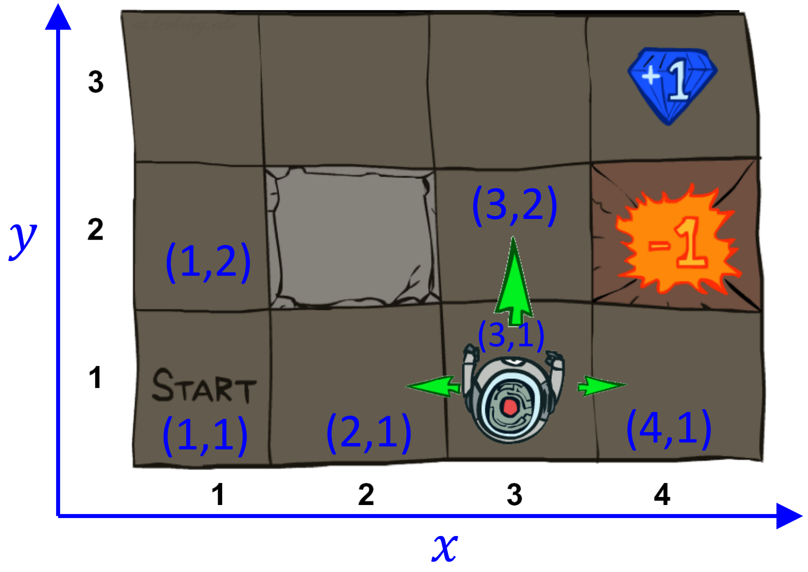


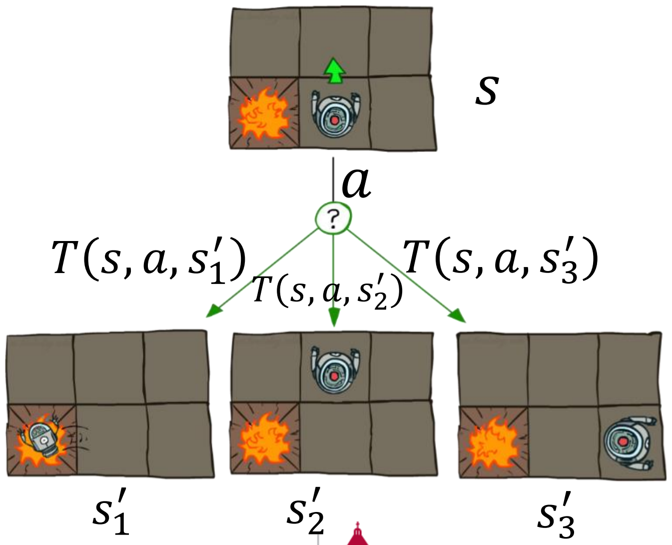
Markov Decision Processes

- $\sum_{s'} T(s,a,s') = 1$

  - For example
  - $\sum_{s'} T((1,1), North, s')$
    $= T((1,1), North, (1,2))$
    $+ T((1,1), North, (2,1))$
    $+ T((1,1), North, (1,1)) = 0.8 + 0.1 + 0.1$
    $= 1$

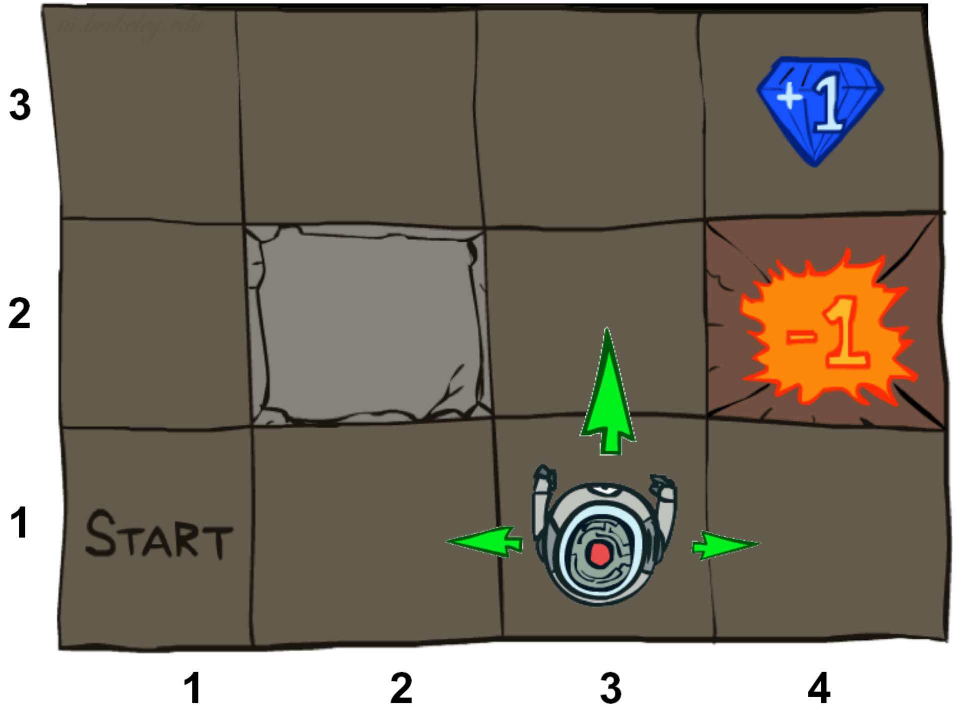
  - $\sum_{s'} T((3,1), North, s')$
    $= T((3,1), North, (3,2))$
    $+ T((3,1), North, (2,1))$
    $+ T((3,1), North, (4,1)) = 0.8 + 0.1 + 0.1$
    $= 1$





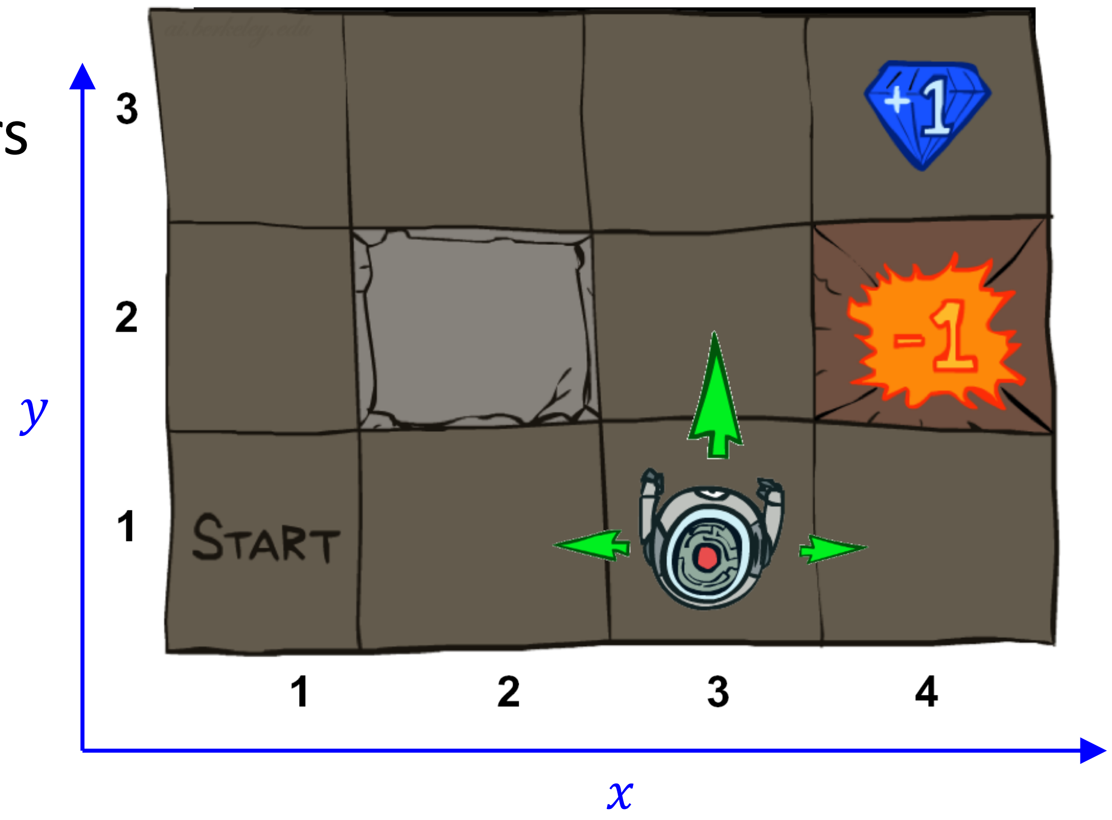
Markov Decision Processes

- **An MDP is defined by:**
  - A reward function
    - R(s,a,s')
    - R(s,a)
    - R(s)
  - A start state
  - Maybe a terminal state
- **MDPs are non-deterministic search problems**



Markov Decision Processes

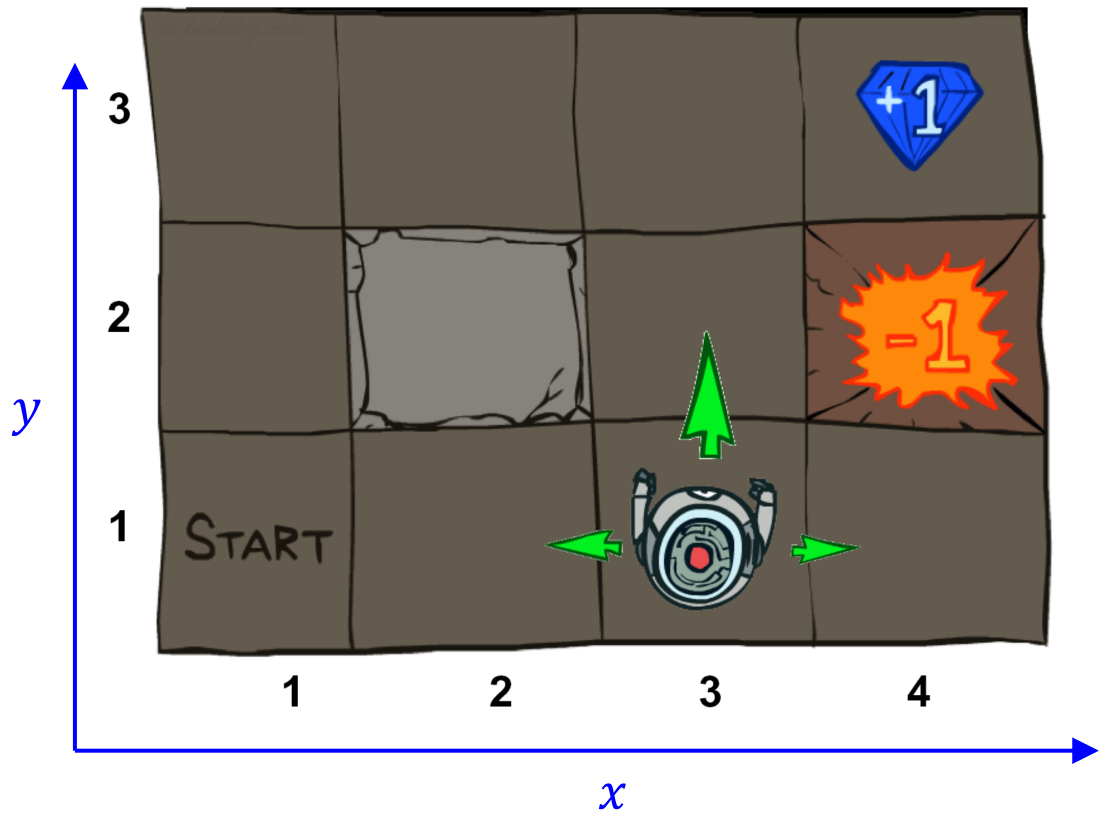
- **Example:**
- Assume the game is over if the agent enters (4,3) or (4,2).
- Assume a reward of +1 is gained when the agent enters (4,3); a reward of -1 is gained when the agent enters (4,2); for any other state transitions the reward is zero.
- What are the values of the following rewards?
  - When s=(3,3), a=East, s'=(4,3), R(s,a,s') =
  - When s=(3,2), a=East, s'=(4,2), R(s,a,s') =
  - When s=(4,1), a=North, s'=(4,2), R(s,a,s') =
  - When s=(4,1), a=North, s'=(3,1), R(s,a,s') =



Markov Decision Processes

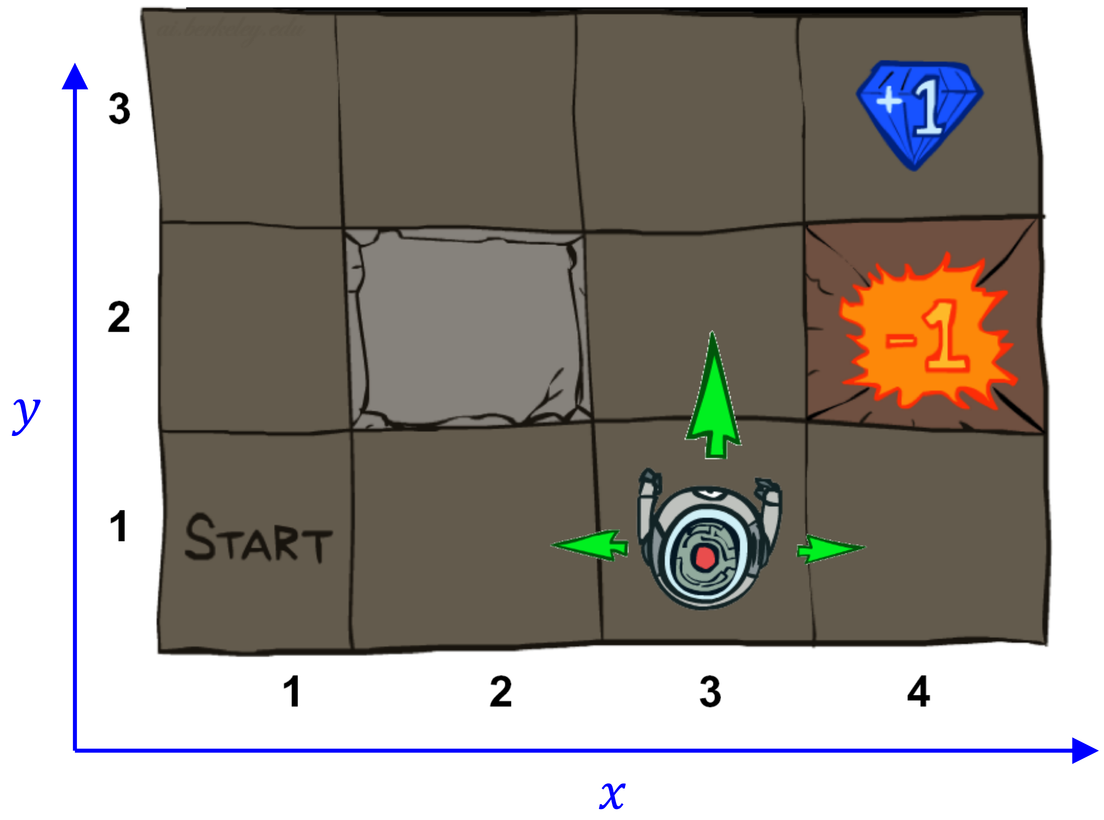
- **Example:**
  - Assume R(s,a,s’) is defined the same way as in the previous slide
  - Assume the agent is in (3,2)
  - He goes North
  - What’s the expected reward?

- **Answer: the reward expression is**
  - R((3,2),North,s’)
  - if s’ = (3,3), then R = 0. What’s the prob. of this situation? 0.8
  - if s’ = (4,2), then R = -1. What’s the prob. of this situation? 0.1
  - E[R] = 0.8*0 + 0.1*(-1)+0.1*0 = -0.1



Markov Decision Processes

- **Example:**
- If a reward of +1 is gained when the agent is in (4,3), and a reward of -1 is gained when the agent is in (4,2), and the reward is zero when the agent is in any other possible grid, then how to represent this kind of rewards?
  - R(s) = +1, when s = (4,3)
  - R(s) = -1, when s = (4,2)
  - R(s) = 0, for any other possible s

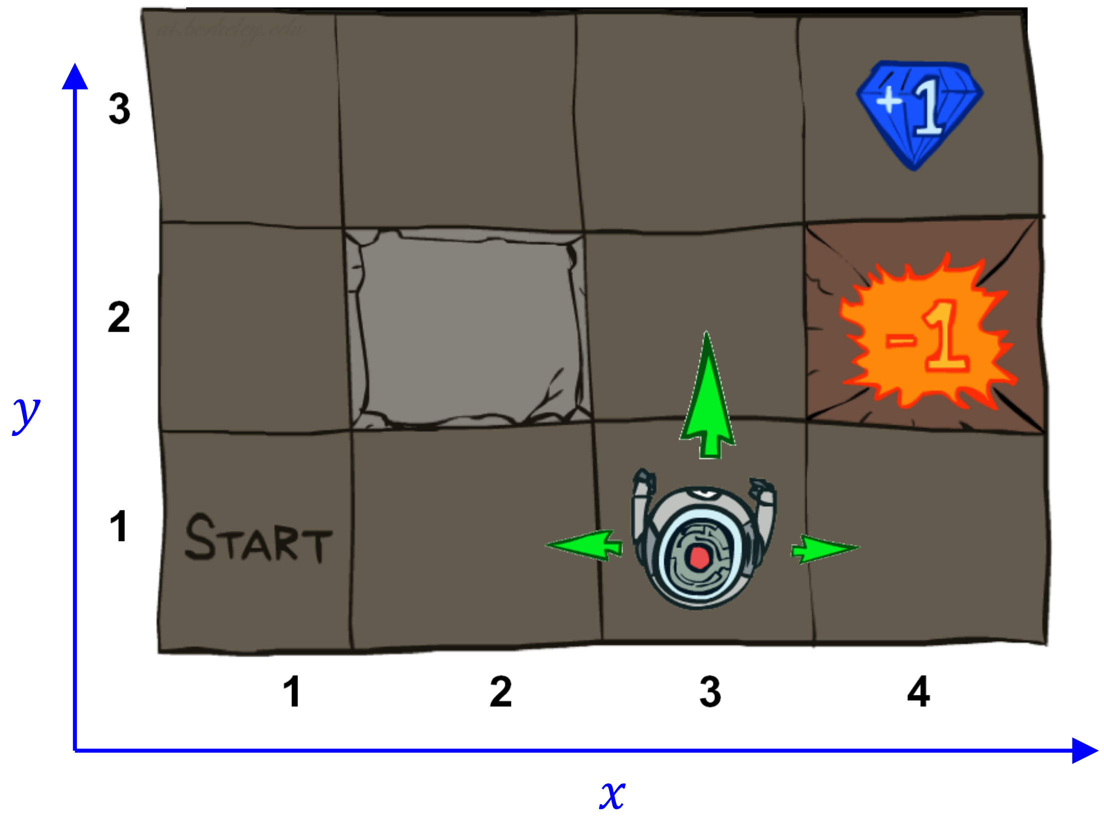


Markov Decision Processes

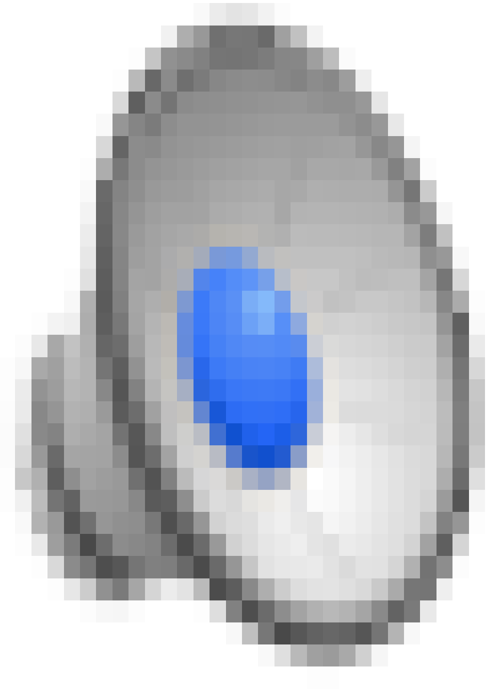
- Watch the video demo in the next slide.
- Think about how the reward function is defined in the video.

- R(s,a)
- R(s=(4,3), a=EXIT) = 1
- R(s=(4,2), a=EXIT) = -1

- R(s,a) = -0.1, for any other valid (s,a)
  - This is the living reward



Video Demo of Gridworld



These slides were adapted from the MDP lecture slides of cs188 developed at UC Berkeley, http://ai.berkeley.edu.

What is Markov about MDPs?

- **“Markov” generally means that given the present state, the future and the past are independent**

- **For Markov decision processes, “Markov” means action outcomes depend only on the current state**

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

$$= \quad P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

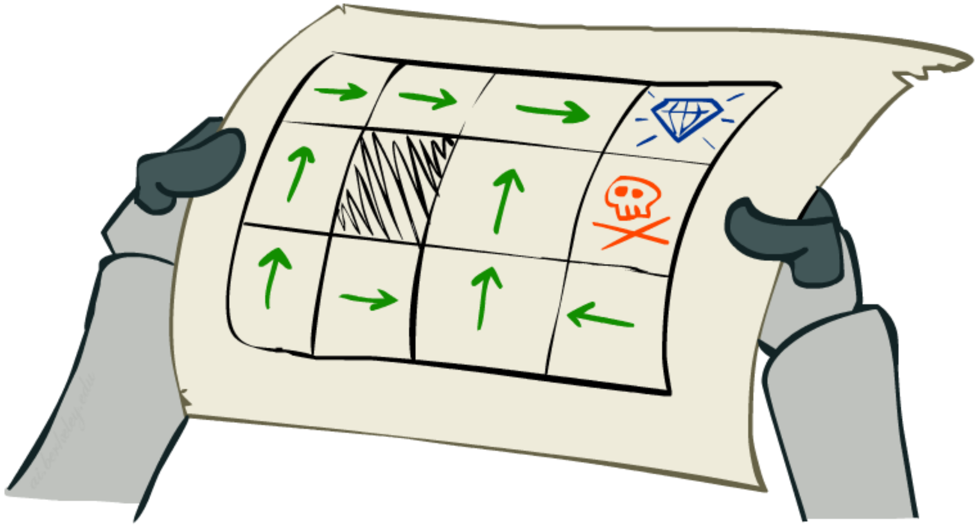
Andrey Markov
(1856-1922)

Policies

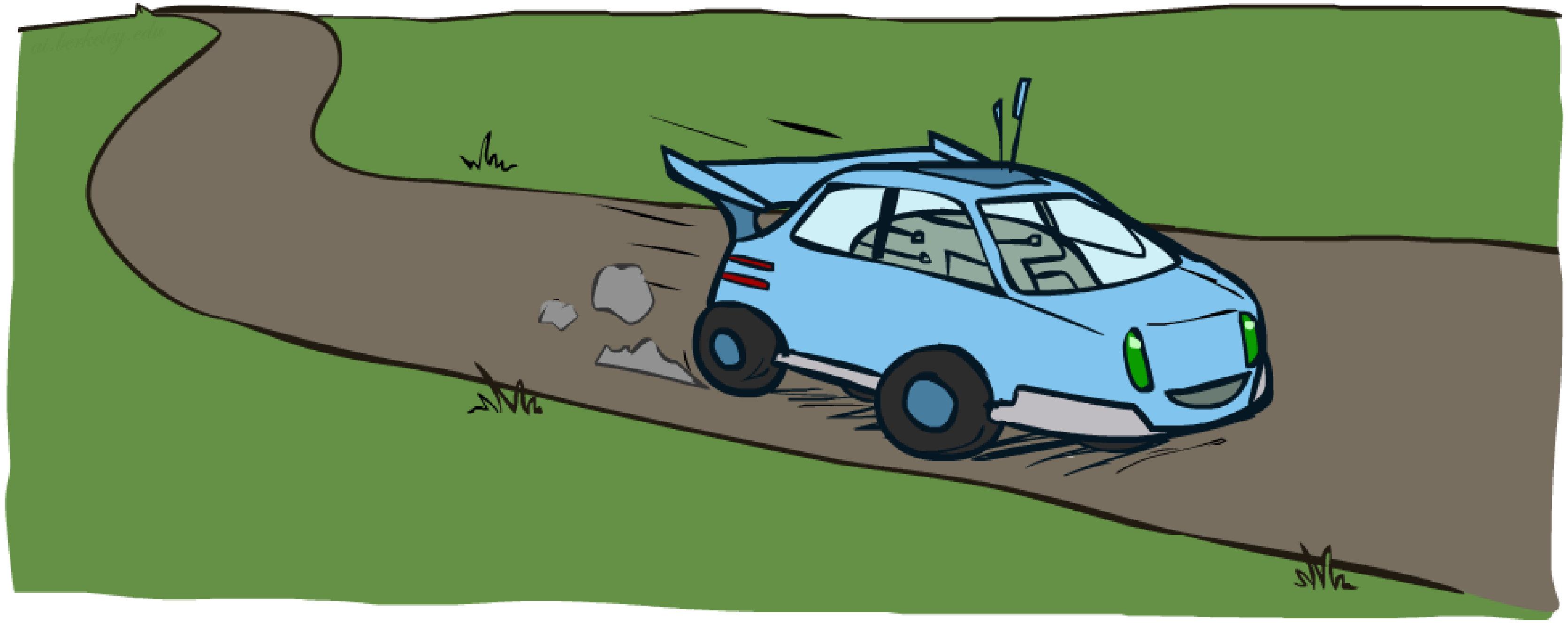
- In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal

- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$

  - A policy π: a mapping from states to actions
  - A policy π gives an action for each state

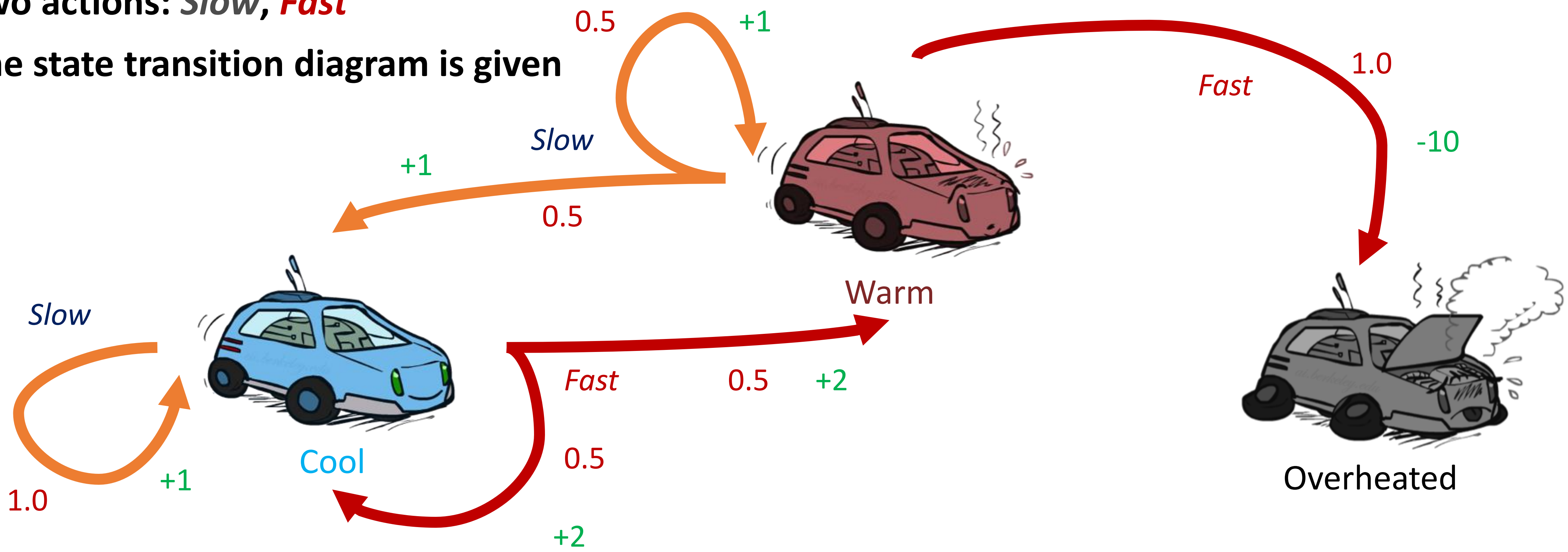


Example: Racing

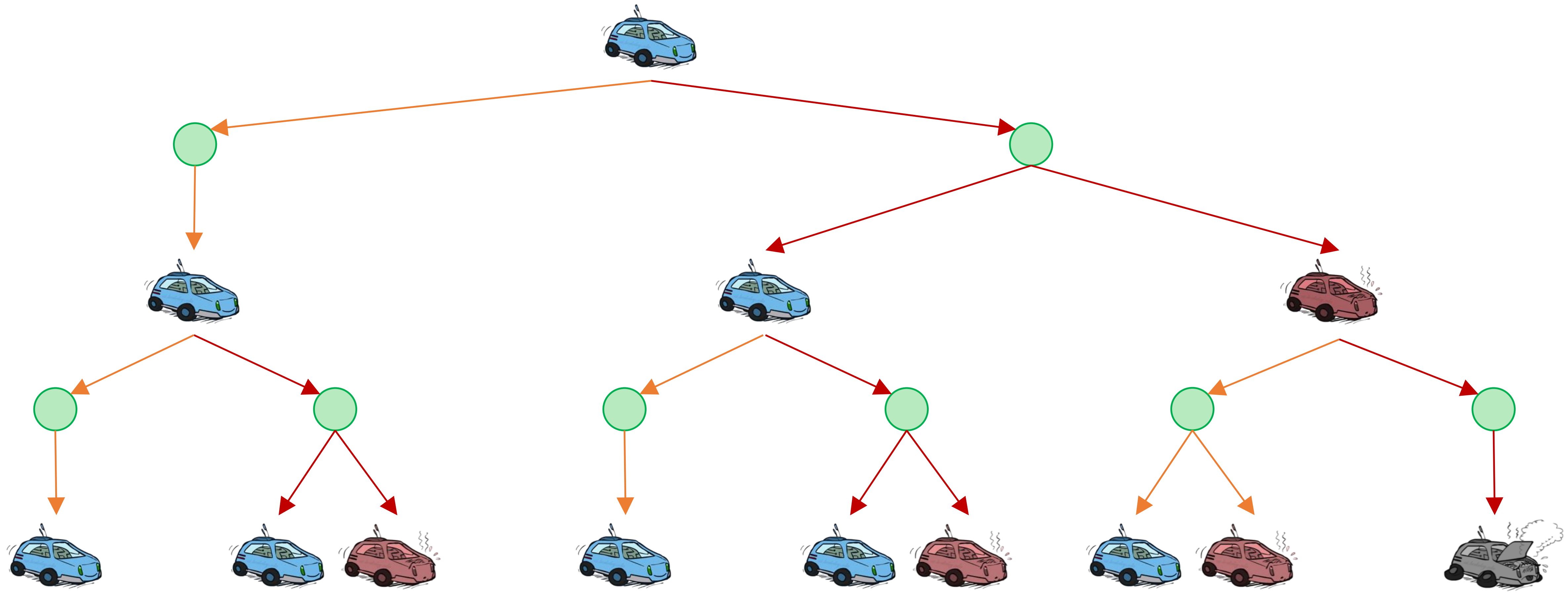


Example: Racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, ***Fast***
- The state transition diagram is given



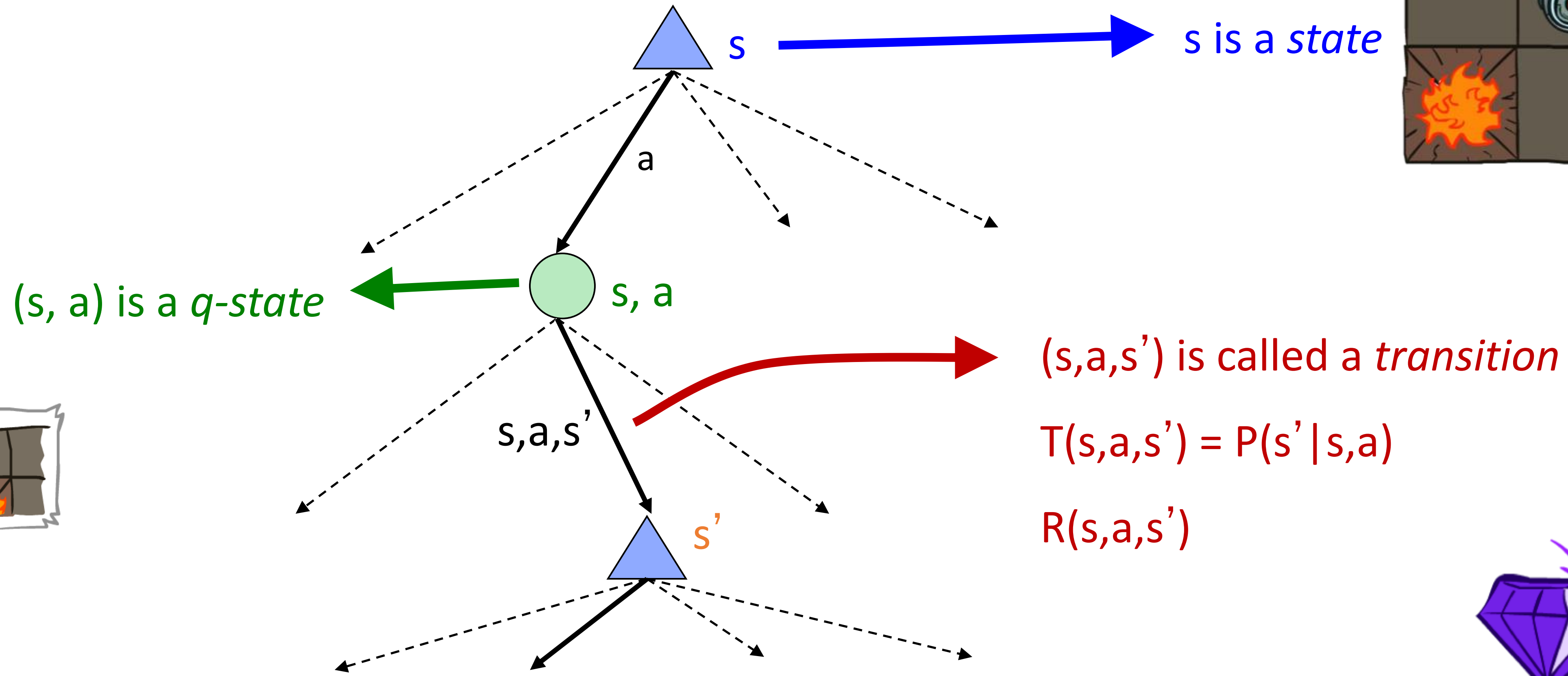
Racing Search Tree

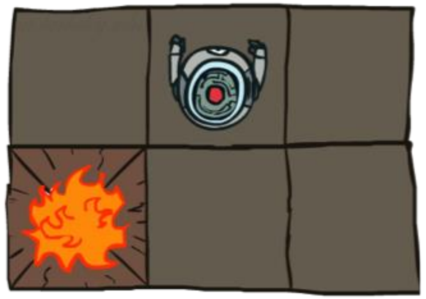


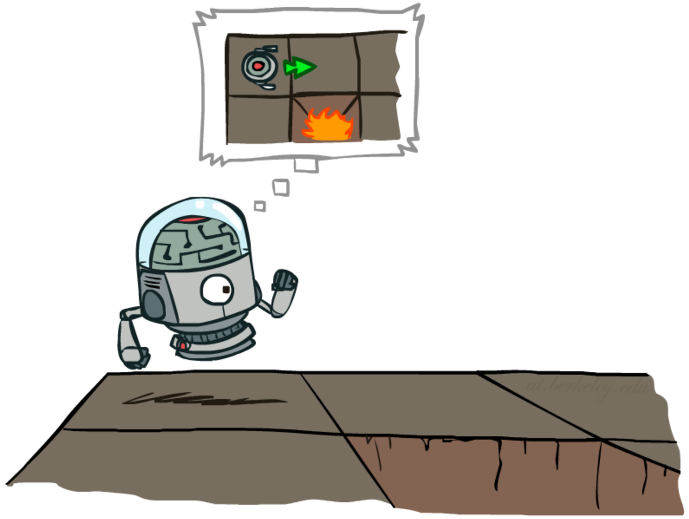
Can you label the branches with the action a, transition probability T(s,a,s’), and reward R(s,a,s’)?

MDP Search Trees

- **Q-state: (s,a)**

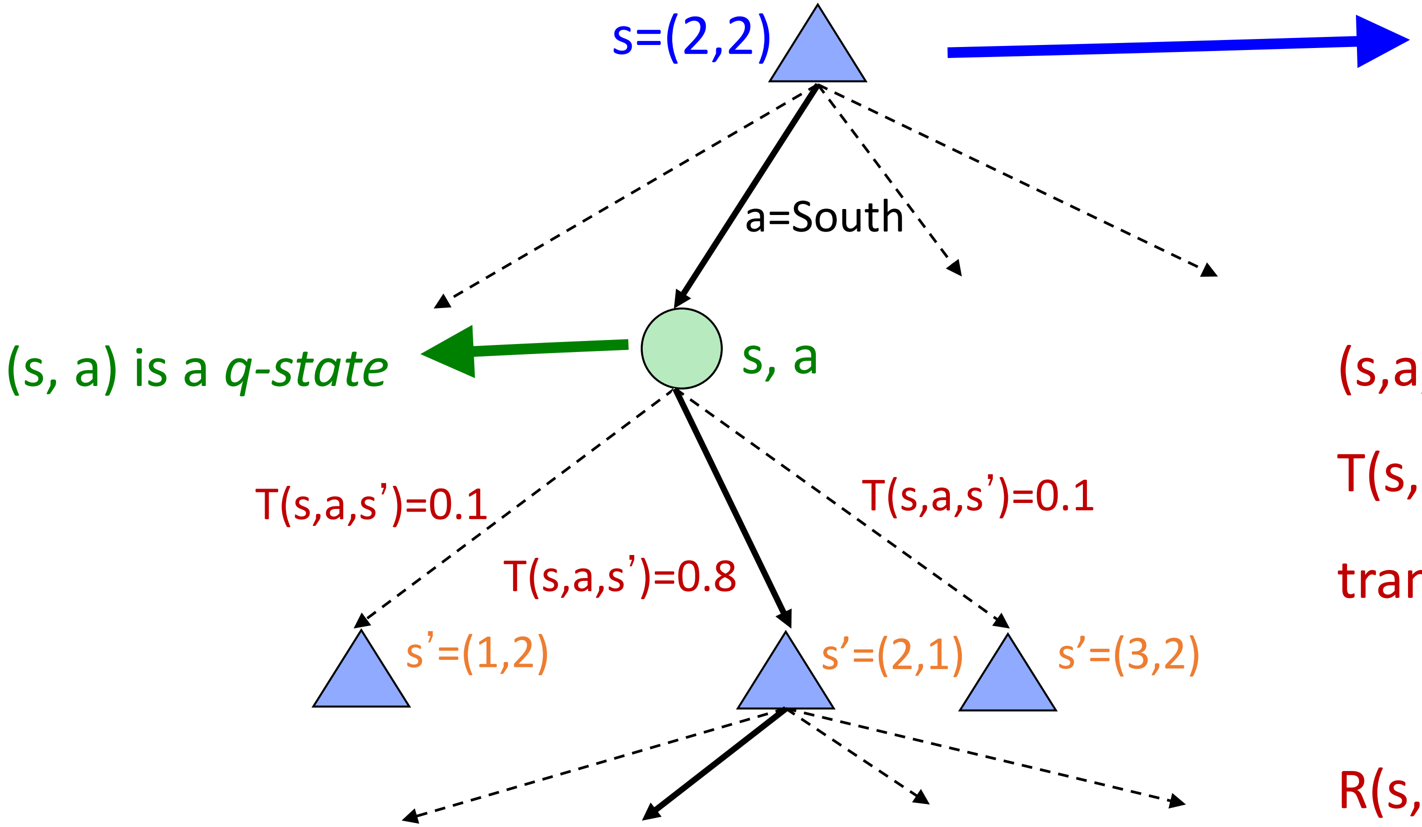




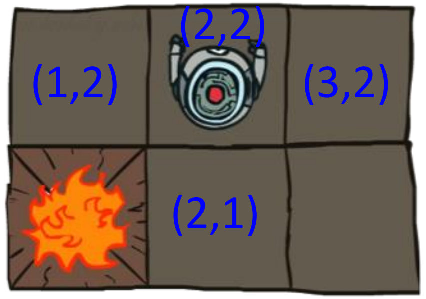


MDP Search Trees

- **Q-state: (s,a)**



s is a *state*



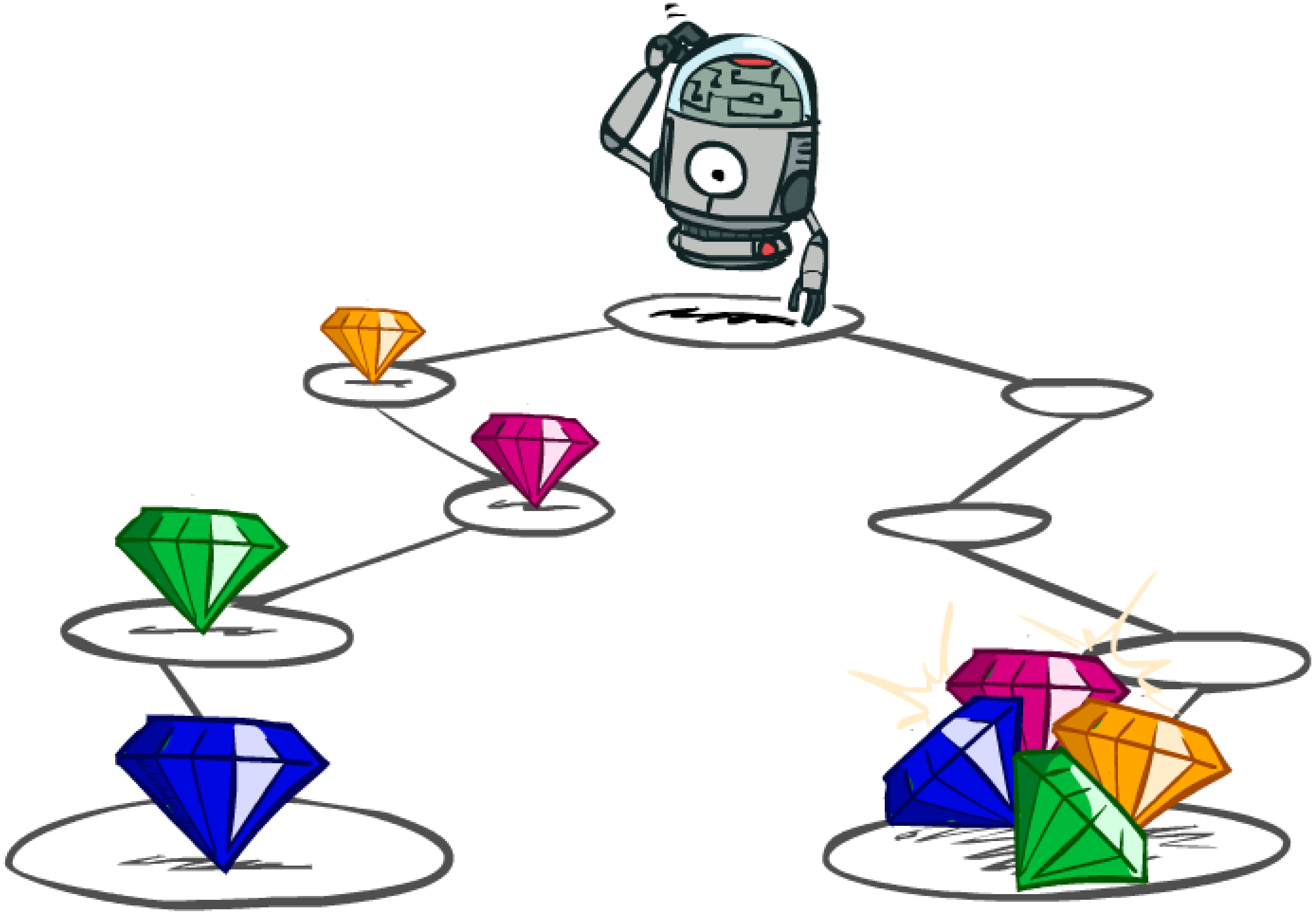
(s,a,s') called a *transition*

T(s,a,s') = P(s'|s,a):

transition probability

R(s,a,s'): (immediate) reward

Reward Sequences



Reward Sequences

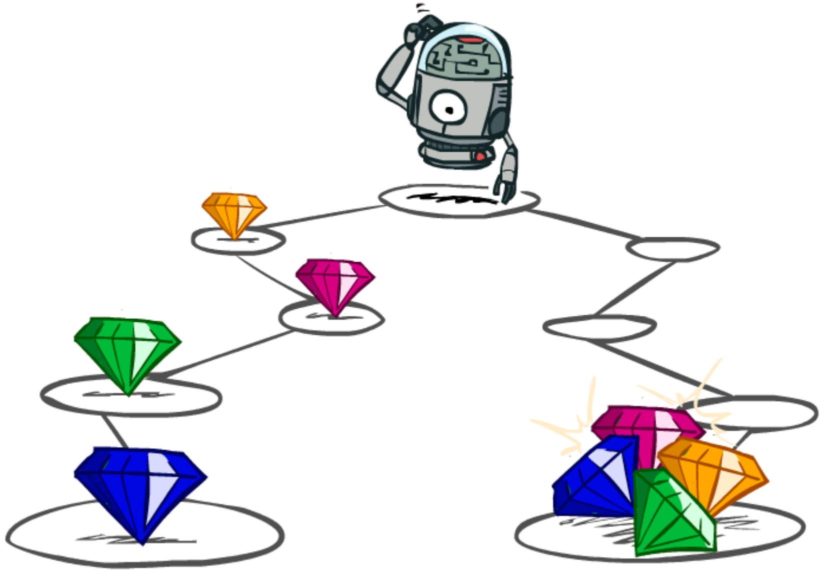
- **What preferences should an agent have over reward sequences?**

- **More or less?**

  [1, 2, 2] or [2, 3, 4]

- **Now or later?**

  [0, 0, 1] or [1, 0, 0]



Discounting

- It’s reasonable to let the agent take actions to maximize the sum of rewards
- It’s also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

1

Worth Now

γ

Worth Next Step

γ^2

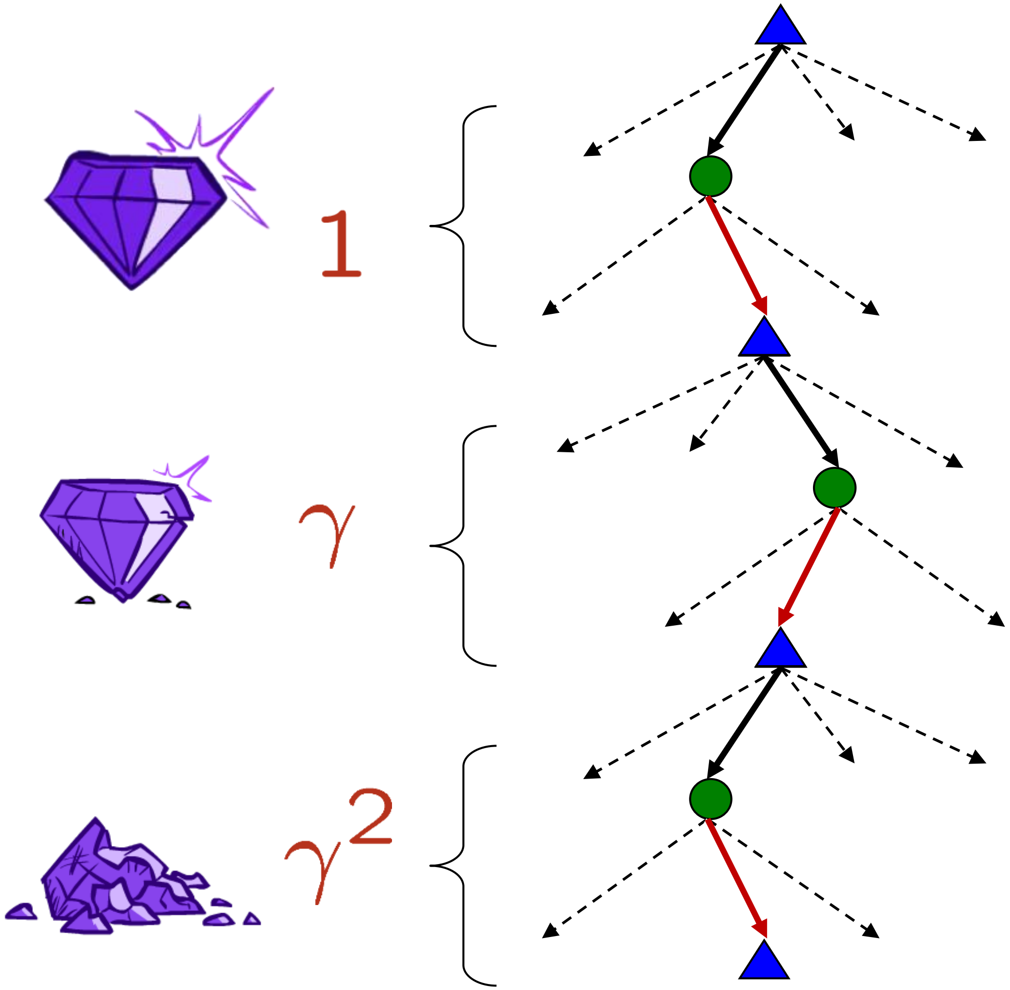
Worth In Two Steps

These slides were adapted from the MDP lecture slides of cs188 developed at UC Berkeley, http://ai.berkeley.edu.

Discounting

- **How to discount?**
  - Each time we descend a level, we multiply in the discount once

- **Why discount?**
  - Sooner rewards probably have higher utility than later rewards

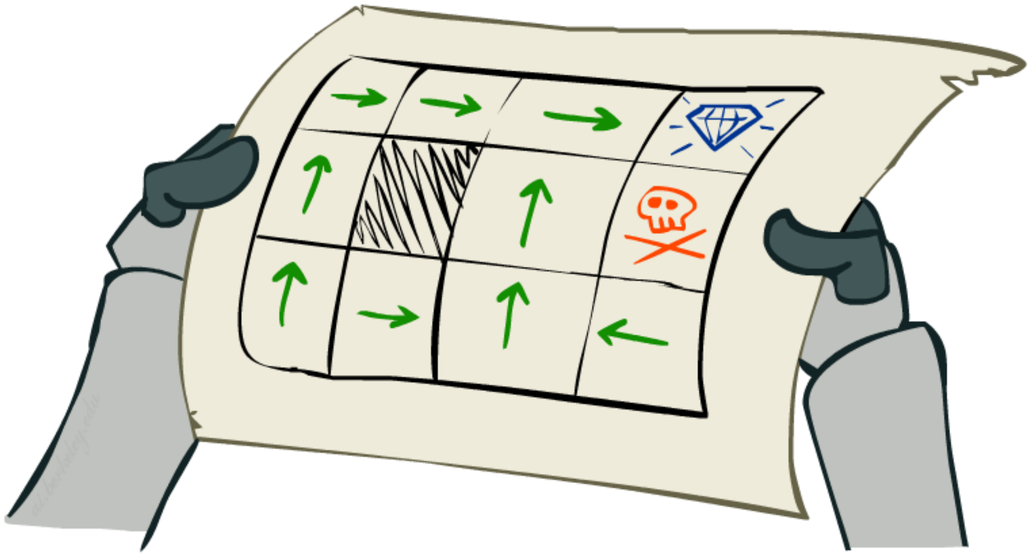


Solving MDPs

- **How to figure out the optimal policy π^* for the agent?**

- **That is, the best action at each state:**

  $\pi^*(s)\ \forall s \in S$

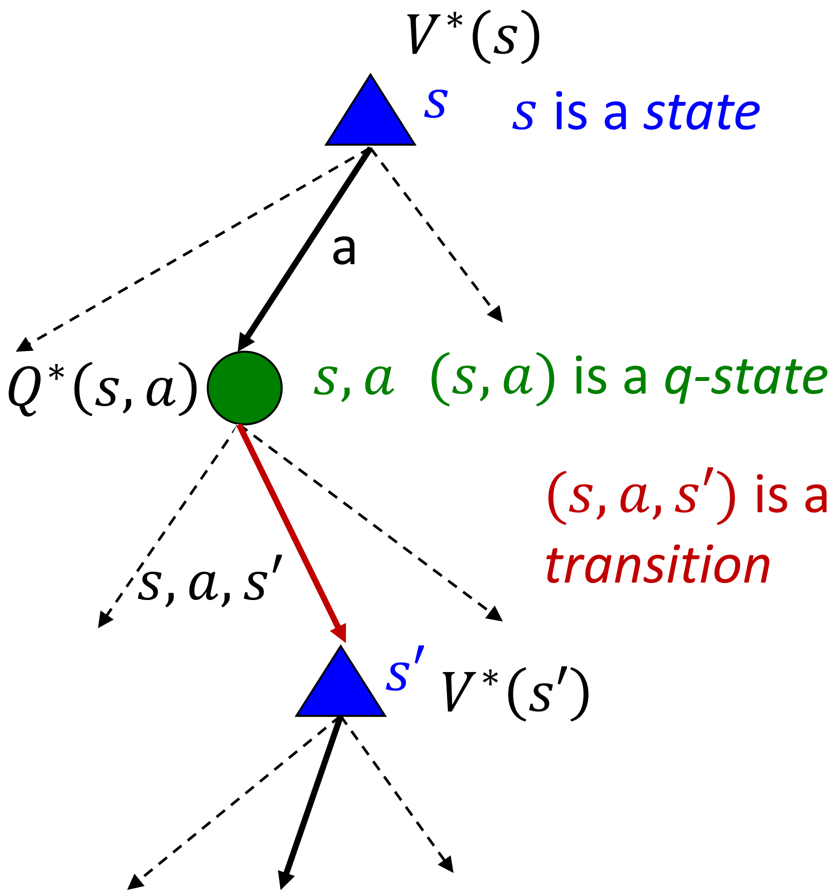


Optimal Quantities

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

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

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



Values of States

- **Values of states**

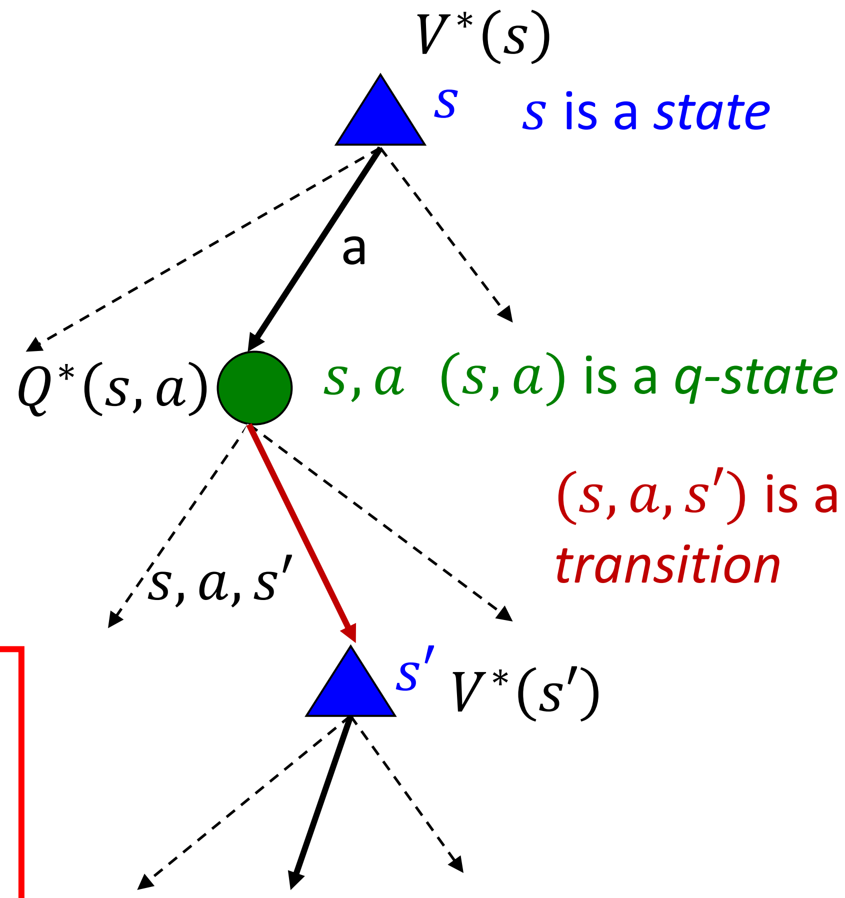
  $$V^*(s) = \max_a Q^*(s,a)$$

- **Q-state value**

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

- **Recursive definition of value:**
- **Bellman Equation**

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



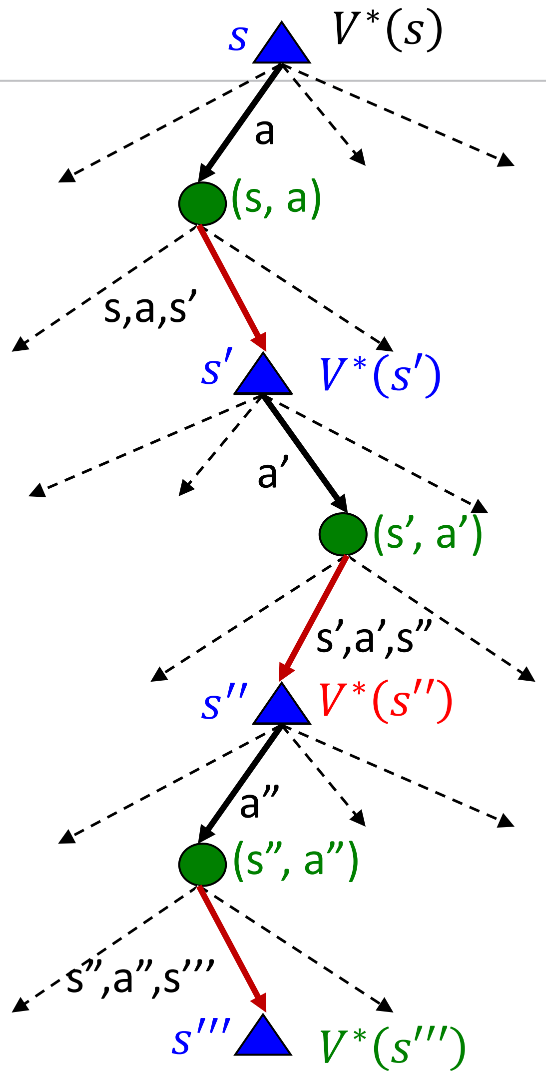
Bellman Equation

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

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

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

…



Bellman Equation

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

- To simplify the problem, let's assume the actions are fixed, and no randomness:

$$V^*(s) = R(s,a,s') + \gamma V^*(s')$$

$$\uparrow$$

$$\boxed{R(s',a',s'') + \gamma V^*(s'')}$$

$$\uparrow$$

$$\boxed{R(s'',a'',s''') + \gamma V^*(s''')}$$

$$V^*(s) = R(s,a,s') + \gamma R(s',a',s'') + \gamma^2 R(s'',a'',s''') + \cdots$$

$V^*(s)$ is the sum of discounted rewards.

Bellman Equation

- **When there is no randomness, and the actions are fixed:**
  $V^*(s)$ is the sum of discounted rewards.

- **In general, there is randomness, and the agent needs to consider multiple action choices, hence**

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

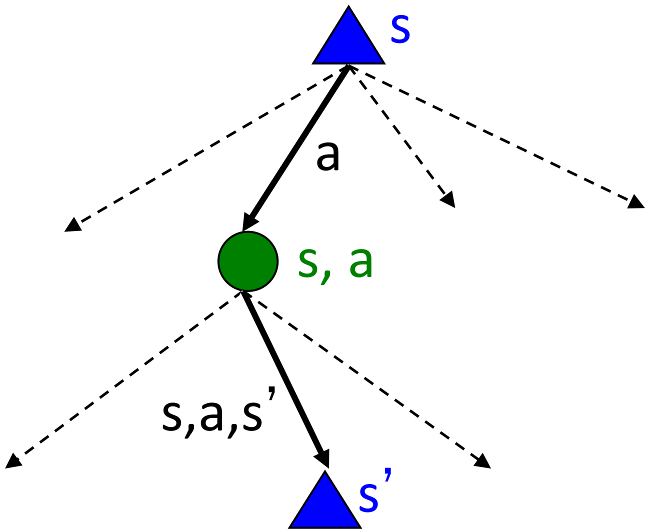
$V^*(s)$ = **Max (Expectation of (the sum of discounted rewards))**

- **Recall: V*(s) = expected value/utility starting in s and acting optimally**

How to calculate $V^*(s)$?

- **Recursive definition of value:**
  - **Bellman Equation**

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



$$V^*(s_1) = f_1\big(V^*(s_1), V^*(s_2), \dots, V^*(s_N)\big)$$
$$V^*(s_2) = f_2\big(V^*(s_1), V^*(s_2), \dots, V^*(s_N)\big)$$
$$\vdots$$
$$V^*(s_N) = f_N\big(V^*(s_1), V^*(s_2), \dots, V^*(s_N)\big)$$

Assume there are N states, then
we have N unknowns and N nonlinear equations

Linear Equation

- Two unknowns x and y
- Two equations

$$c_1 = a_1 x + b_1 y$$
$$c_2 = a_2 x + b_2 y$$

- How to solve for x and y?

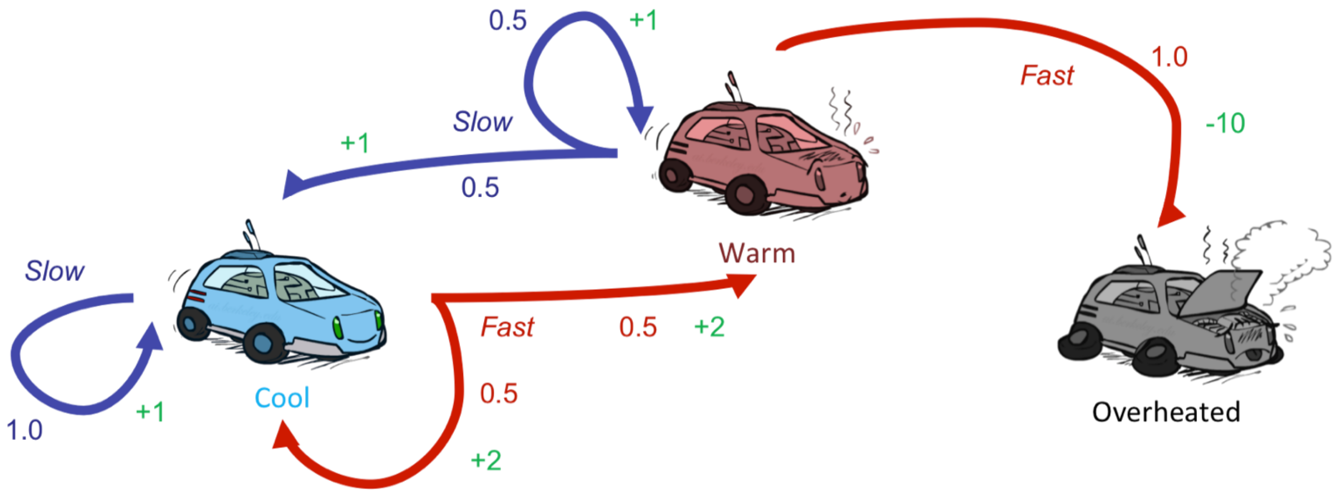
$$x = \frac{c_1 - b_1 y}{a_1} \quad \dots (1)$$

$$c_2 = a_2 \frac{c_1 - b_1 y}{a_1} + b_2 y$$

$\Rightarrow$ Solve for y, then plug y into (1) to solve for x

Value Iteration

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



Assume no discount!

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

Value Iteration

- **Step 1:** Initialize $V_0(s) = 0$, for $s = s_1, s_2, s_3, \ldots$
- **Step 2:** k=1 (1^{st} iteration)

  Update $V_1(s)$, for $s = s_1, s_2, s_3, \ldots,$

  using $V_0(s) = 0$, $s = s_1, s_2, s_3, \ldots$

  $$V_1(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_0(s')]$$

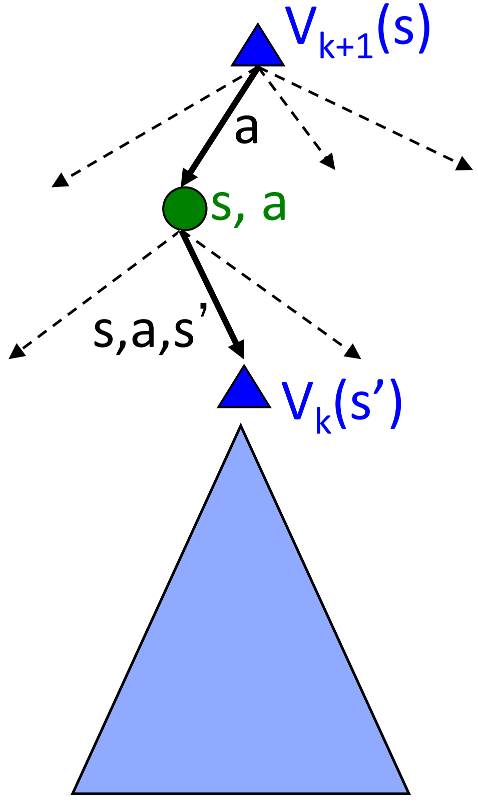
- **Step 3:** k=2 (2^{nd} iteration)

  Update $V_2(s)$, for $s = s_1, s_2, s_3, \ldots,$

  using $V_1(s)$, $s = s_1, s_2, s_3, \ldots$

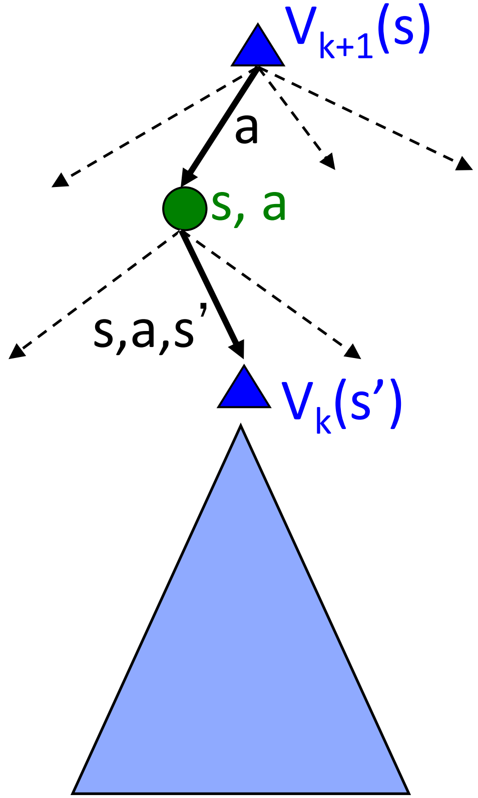
  $$V_2(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_1(s')]$$

- **Keep running the iterations till the values $V_k(s)$ converge.**



Value Iteration

- **Start with $V_0(s) = 0$**
- **Given vector of $V_k(s)$ values, do one ply of expectimax from each state:**



- **Bellman Update**

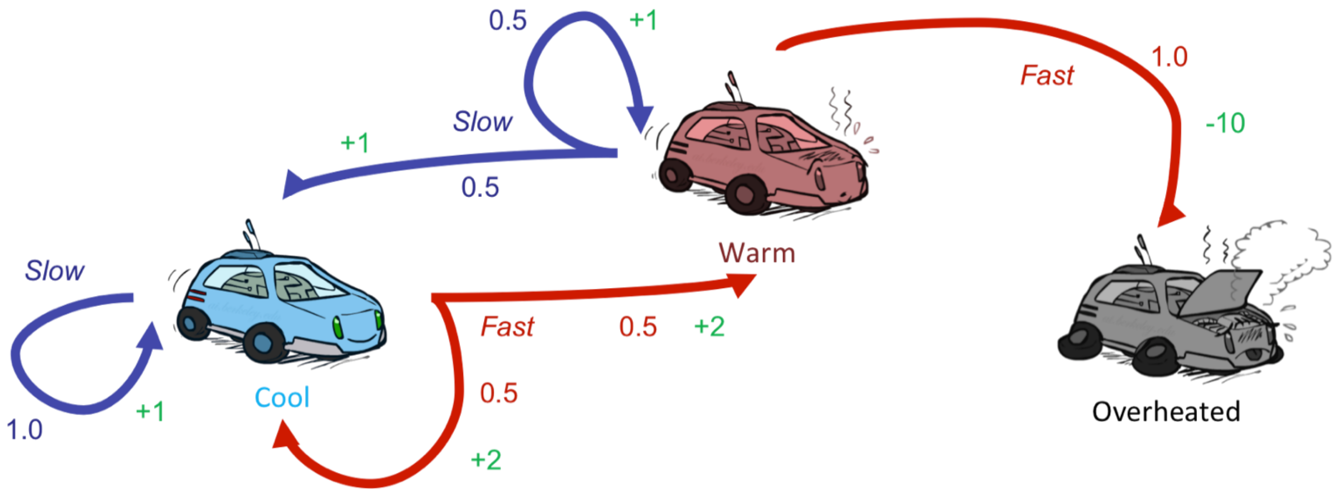
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

- **Repeat until convergence**

- **Complexity of each iteration:** $O(|S|^2 \times |A|)$
  - $|S|$: The cardinality of the set S, i.e. the number of elements in the set S

Example: Value Iteration

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



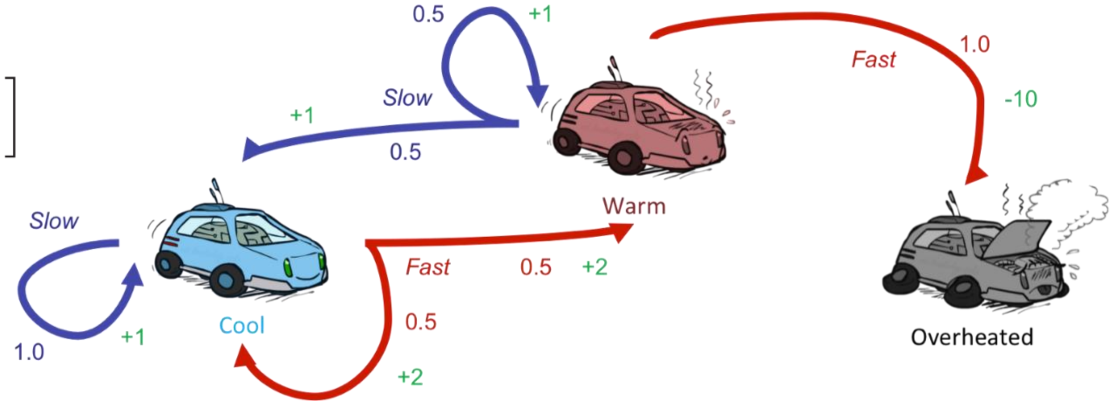
Assume no discount!

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

$V_0(cool) = 0,\ \ V_0\ (warm) = 0,\ \ V_0(overheated) = 0$

- $V_1(cool)$:

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



- **1. a=slow**
  - $T(s = cool, a = slow, s' = cool) = 1$

    $R(cool, slow, s' = cool) = 1$
  - $V_0(s' = cool) = 0$

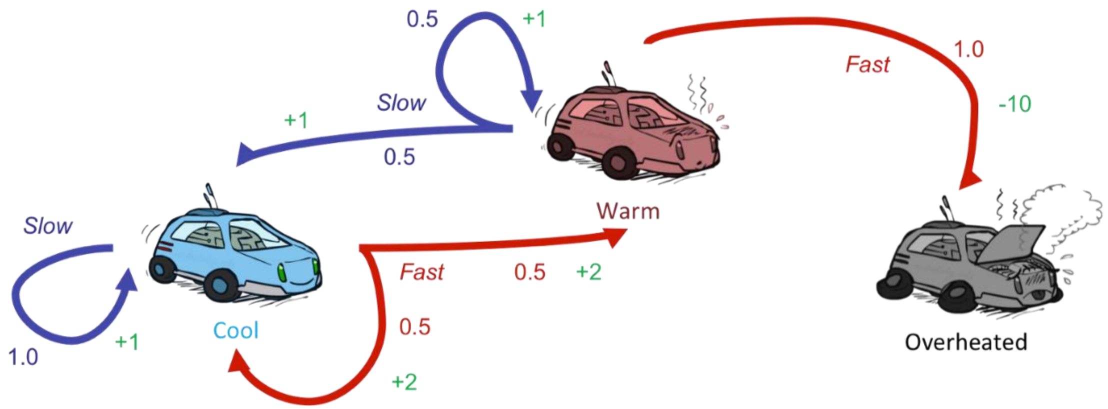
  - $Result = T(cool, slow, s' = cool) \times [R(cool, slow, s' = cool) + \gamma \times V_0(s' = cool)]$
    $= 1 \times [1 + 1 \times 0] = 1$

$$V_0(cool) = 0,\ \ V_0\ (warm) = 0,\ \ V_0(overheated) = 0$$

- $V_1(cool)$:

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

- **2. a=fast**
  - $T(cool, fast, s' = cool) = 0.5$
    $R(cool, fast, s' = cool) = 2$
  - $T(cool, fast, s' = warm) = 0.5$
    $R(cool, fast, s' = warm) = 2$



- $Result\ = T(cool, fast, s' = cool) \times [R(cool, fast, s' = cool) + \gamma \times V_0(s' = cool)]$
$+ T(cool, fast, s' = warm) \times [R(cool, fast, s' = warm) + \gamma \times V_0(s' = warm)] = 2$

Hence, $V_1(cool) = 2$

$V_0(cool) = 0,\ \ V_0\ (warm) = 0,\ \ V_0(overheated) = 0$

- $V_1(warm)$:

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

- **1. a=slow**

$$T(s = warm, a = slow, s' = cool) = 0.5$$

$$R(warm, slow, s' = cool) = 1$$

$$T(warm, slow, s' = warm) = 0.5$$

$$R(warm, slow, s' = warm) = 1$$

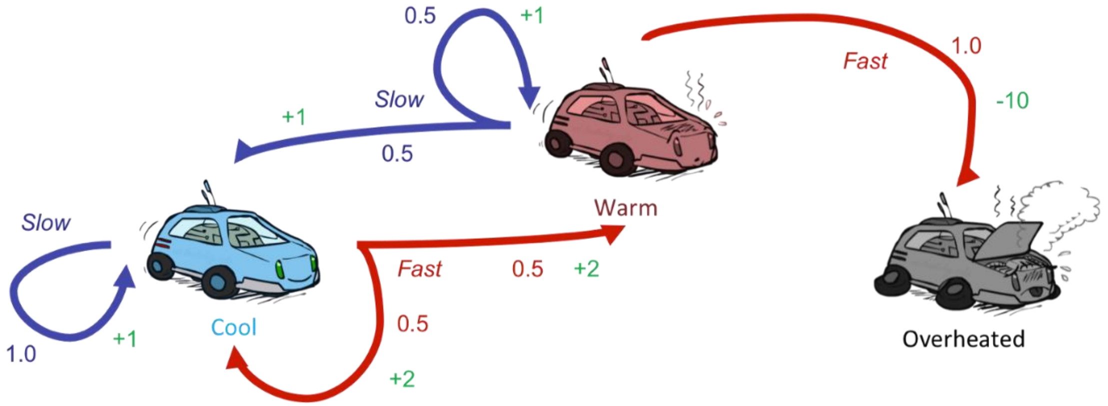
- $Result\ = T(warm, slow, s' = cool) \times [R(warm, slow, s' = cool) + \gamma \times V_0(s' = cool)]$

$+ T(warm, slow, s' = warm) \times [R(warm, slow, s' = warm) + \gamma \times V_0(s' = warm)] = 1$

$V_0(cool) = 0,\ V_0\ (warm) = 0,\ V_0(overheated) = 0$

- $V_1(warm)$:

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



- **2. a=fast**
  - $T(warm, fast, s' = overheated) = 1$

    $R(warm, fast, s' = overheated) = -10$
  - $V_0(s' = overheated) = 0$

  - $Result = T(warm, fast, s' = overheated) \times [R(warm, fast, s' = overheated) + \gamma \times V_0(s' = overheated)]$

    $= 1 \times [-10 + 1 \times 0] = -10$

Hence, $V_1(warm) = 1$

$V_0(cool) = 0,\ V_0\ (warm) = 0,\ V_0(overheated) = 0$

- $V_1(overheated)$:

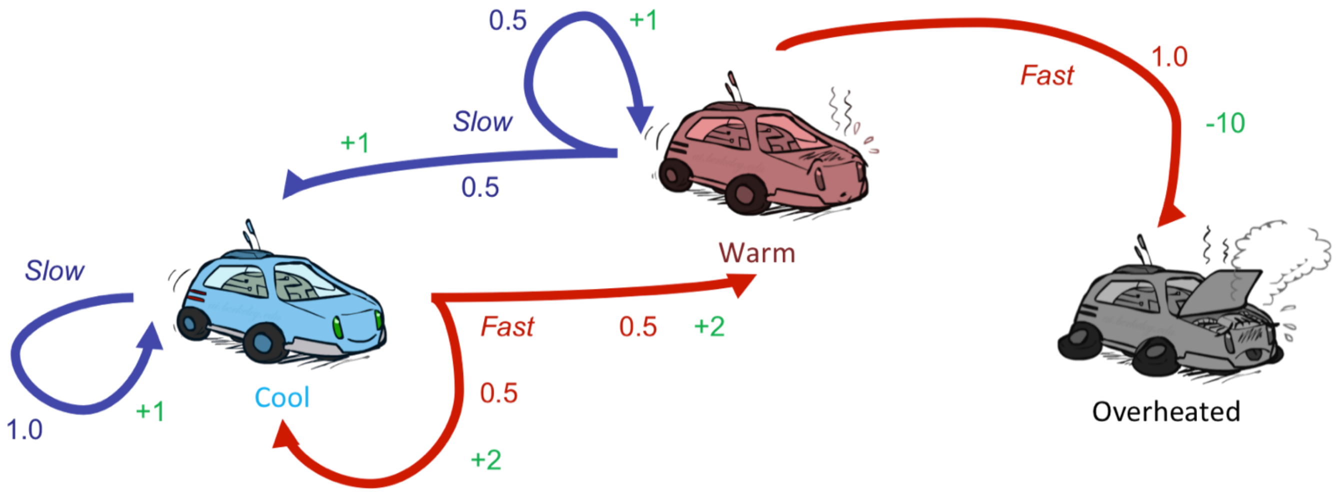
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

- **Overheated is the end state**
  - No more state transition (no s')
  - No $T(overheated, a, s$'), no $R(overheated, a, s$')

Hence, $V_1(overheated) = 0$

Example: Value Iteration

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



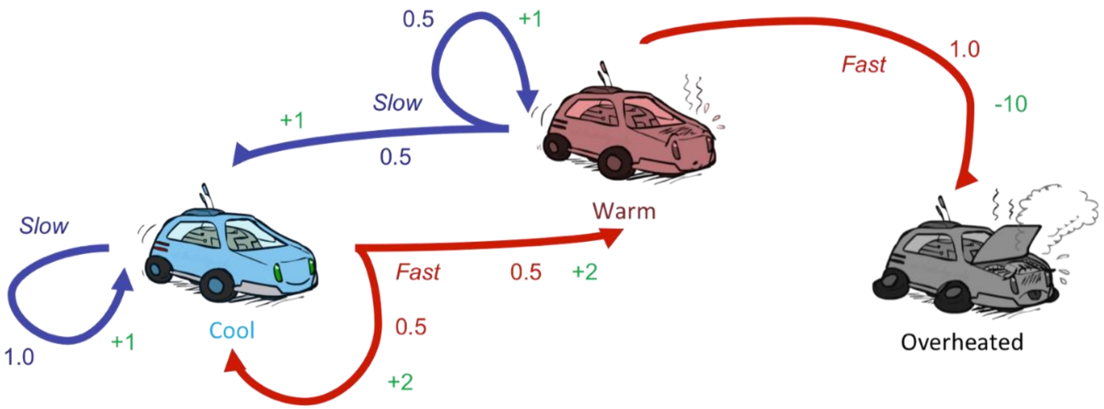
Assume no discount!

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

$V_1(cool) = 2,\ \ V_1\ (warm) = 1,\ \ V_1(overheated) = 0$

- $V_2(cool)$:

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



- **1. a=slow**
  - $T(cool, slow, s' = cool) = 1$
    $R(cool, slow, s' = cool) = 1$
  - $V_1(s' = cool) = 2$

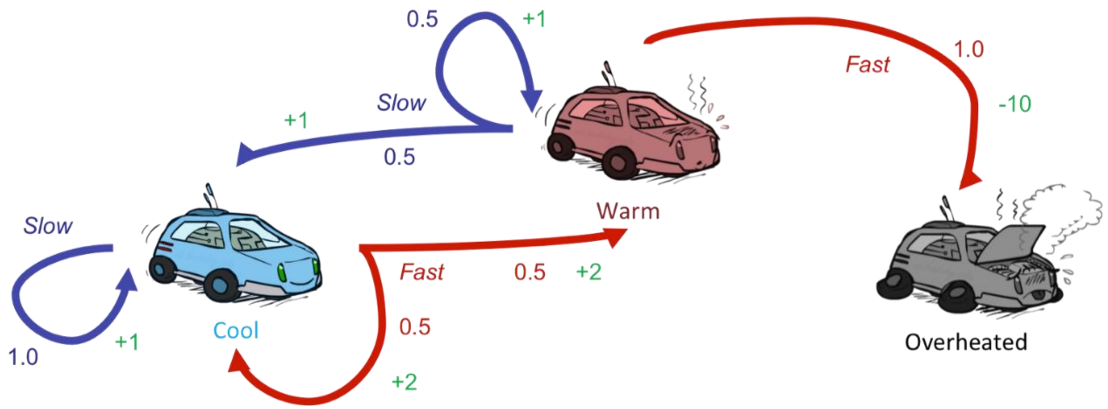
  - $Result = T(cool, slow, s' = cool) \times [R(cool, slow, s' = cool) + \gamma \times V_1(s' = cool)]$
    $= 1 \times [1 + 1 \times 2] = 3$

$V_1(cool) = 2, \; V_1\,(warm) = 1, \; V_1(overheated) = 0$

- $V_2(cool)$:

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

- **2. a=fast**
  - $T(cool, fast, s' = cool) = 0.5$
    $R(cool, fast, s' = cool) = 2$
  - $T(cool, fast, s' = warm) = 0.5$
    $R(cool, fast, s' = warm) = 2$



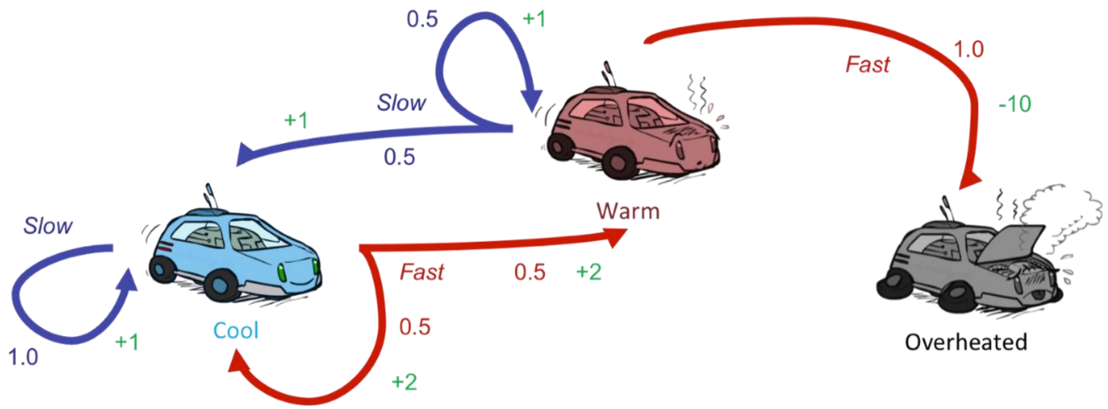
- $Result = T(cool, fast, s' = cool) \times [R(cool, fast, s' = cool) + \gamma \times V_1(s' = cool)]$
  $+ T(cool, fast, s' = warm) \times [R(cool, fast, s' = warm) + \gamma \times V_1(s' = warm)]$
  $= 0.5 \times [2 + 1 \times 2] + 0.5 \times [2 + 1 \times 1] = 2 + 1.5 = 3.5$

Hence, $V_2(cool) = 3.5$

$V_1(cool) = 2,\ V_1\ (warm) = 1,\ V_1(overheated) = 0$

- $V_2(warm)$:

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



- **1. a=slow**
  - $T(warm, slow, s' = cool) = 0.5$
    $R(warm, slow, s' = cool) = 1$
  - $T(warm, slow, s' = warm) = 0.5$
    $R(warm, slow, s' = warm) = 1$

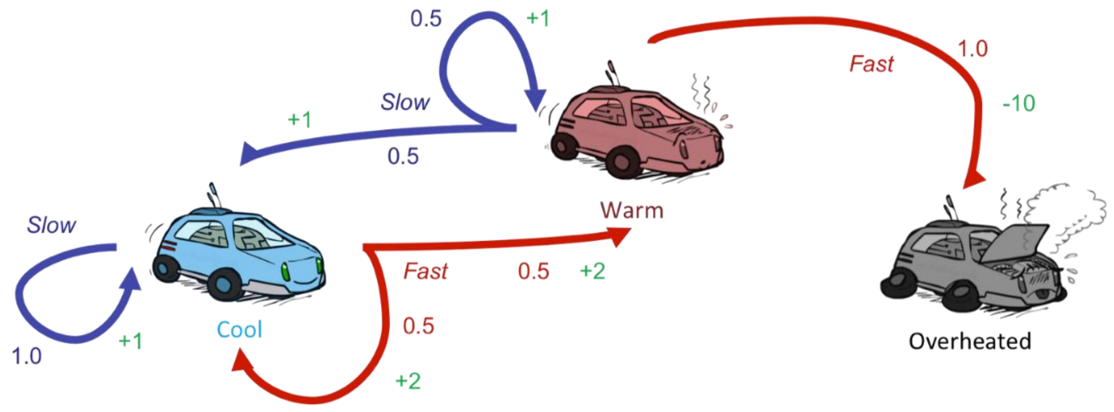
- $Result\ = T(warm, slow, s' = cool) \times [R(warm, slow, s' = cool) + \gamma \times V_1(s' = cool)]$
$+ T(warm, slow, s' = warm) \times [R(warm, slow, s' = warm) + \gamma \times V_1(s' = warm)]$

$$= 0.5 \times [1 + 1 \times 2] + 0.5 \times [1 + 1 \times 1] = 1.5 + 1 = 2.5$$

$V_1(cool) = 2,\ V_1\ (warm) = 1,\ V_1(overheated) = 0$

- $V_2(warm)$:

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



- **2. a=fast**
  - $T(warm, fast, s' = overheated) = 1$

    $R(warm, fast, s' = overheated) = -10$
  - $V_1(s' = overheated) = 0$

  - $Result = T(warm, fast, s' = overheated) \times [R(warm, fast, s' = overheated) + \gamma \times V_1(s' = overheated)]$

  $= 1 \times [-10 + 1 \times 0] = -10$

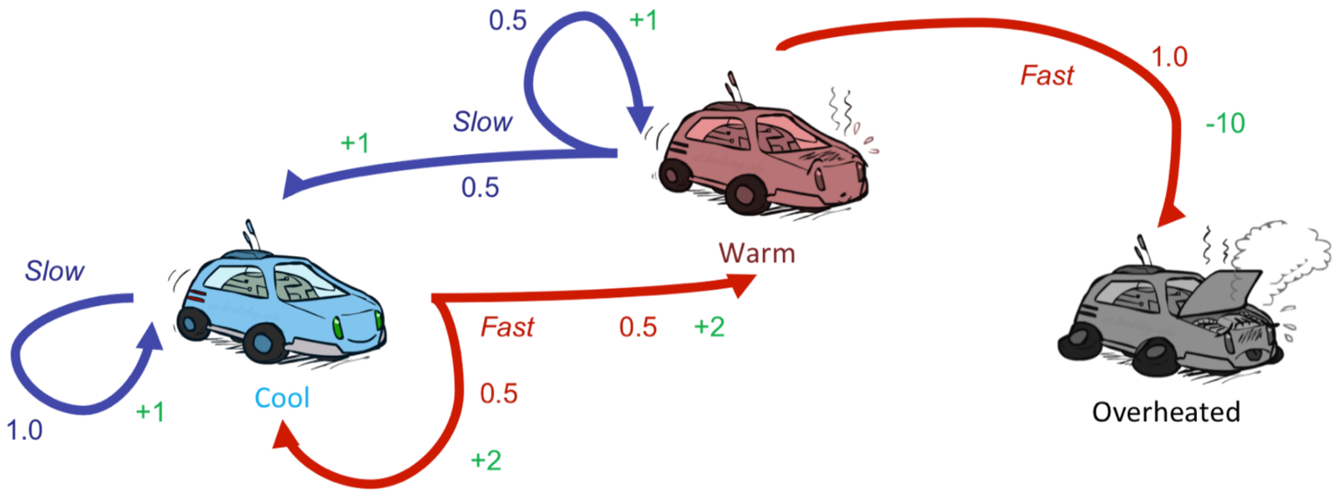
Hence, $V_2(warm) = 2.5$

$V_1(cool) = 2,\ \ V_1\ (warm) = 1,\ \ V_1(overheated) = 0$

- $V_2(overheated) = 0$

Example: Value Iteration

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



Assume no discount!

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

Convergence of Value Iteration

- **Theorem: will converge to unique optimal values**
- **Stopping Criterion**
- **Let the discount factor be γ**
- **If we want to achieve: $\max_s |V_{k+1}(s) - V^*(s)| < \epsilon$,**

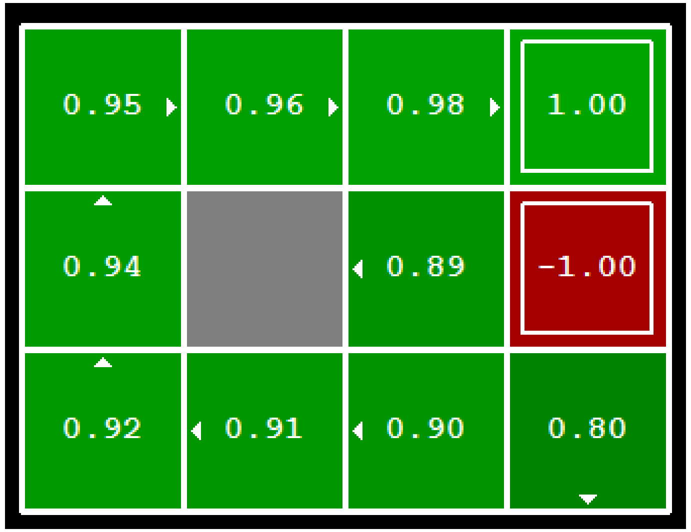
  then we need to run the iterations until

$$\max_s |V_{k+1}(s) - V_k(s)| < \epsilon(1-\gamma)/\gamma$$

Policy Extraction

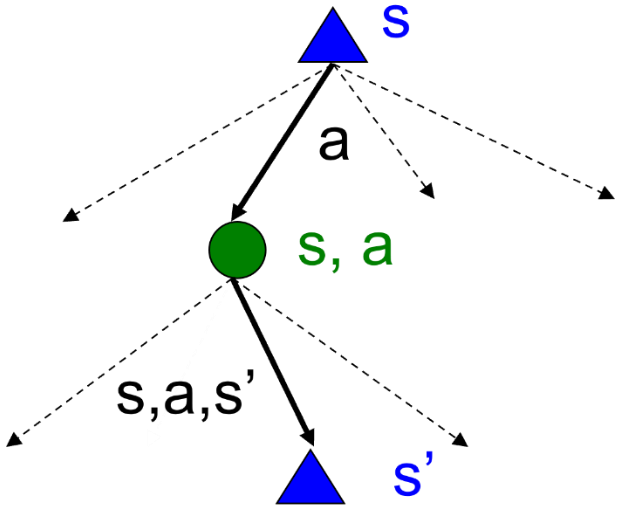
- **Assume we already calculated the optimal values V*(s)**
- **How to figure out the best action at state s?**
  - It's not obvious!

- **We need to do a mini-expectimax (one step look-ahead)**



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

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

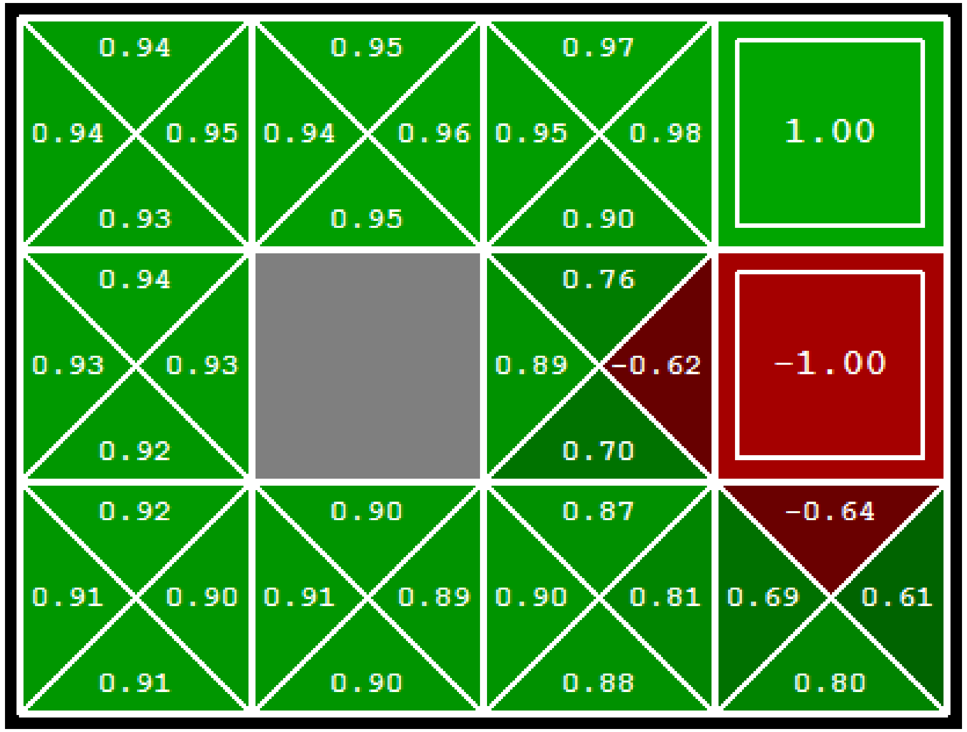


- **This is called policy extraction**
  - It gets the actions implied by the values

Computing Actions from Q-Values

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

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



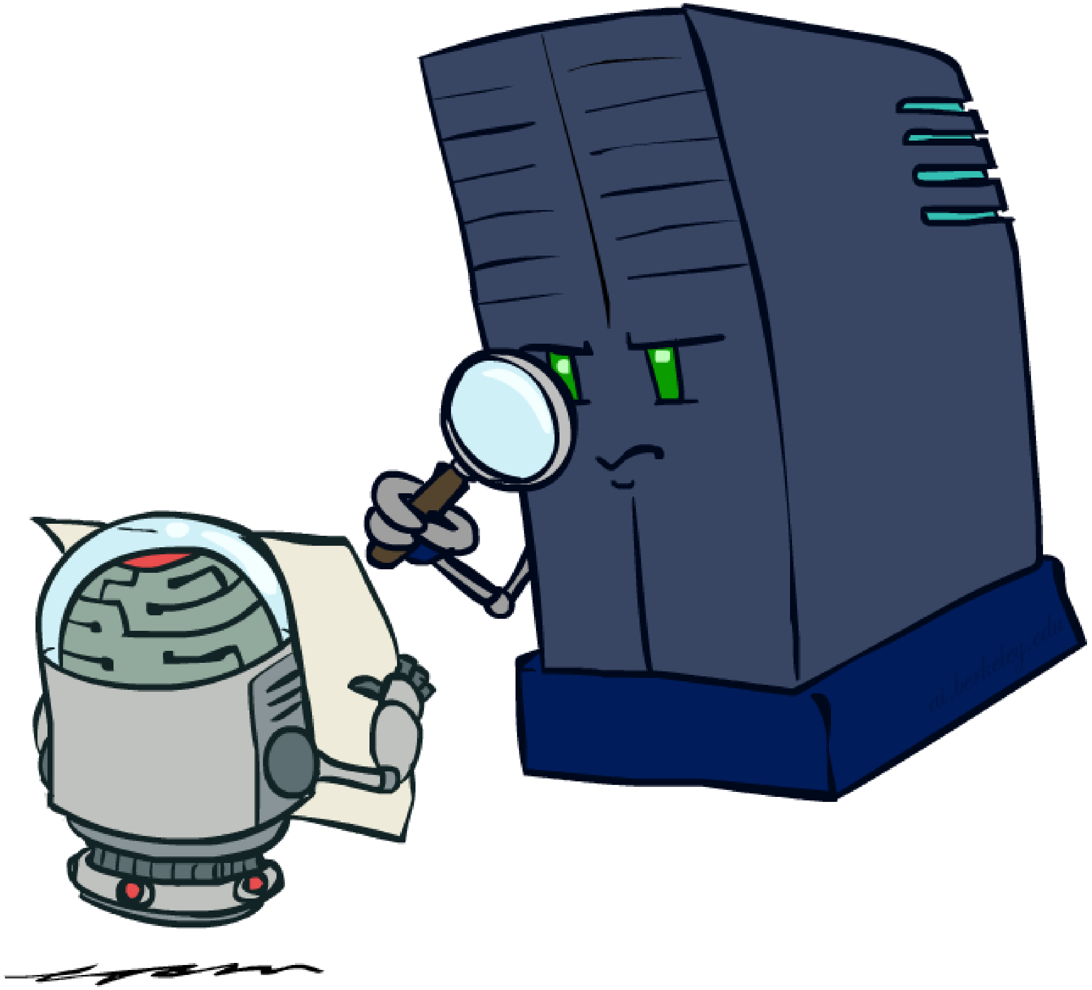
- **How to figure out the best action at state s?**
- **Completely trivial to decide!**

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

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

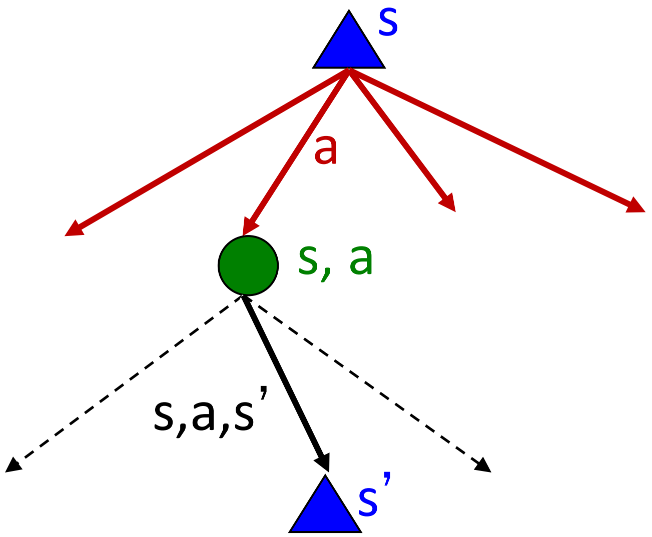
These slides were adapted from the MDP lecture slides of cs188 developed at UC Berkeley, http://ai.berkeley.edu.

Policy Evaluation

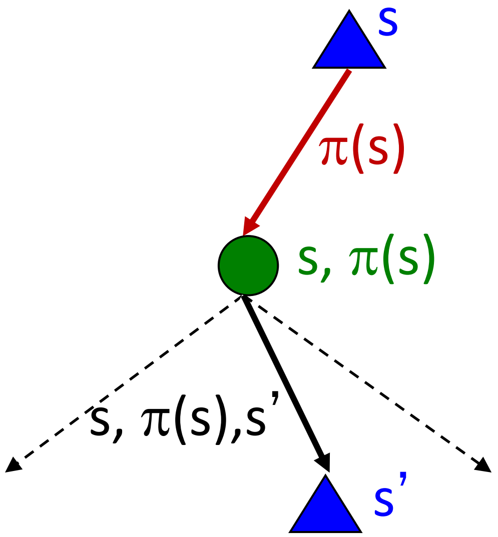


Fixed Policies

When no policy is told:
Do the optimal action



When a policy π is told:
Do what π says to do



- **When no policy is told: Expectimax trees max over all actions to compute the optimal values**
- **When a fixed policy π is told: the tree would be simpler - only one action per state**

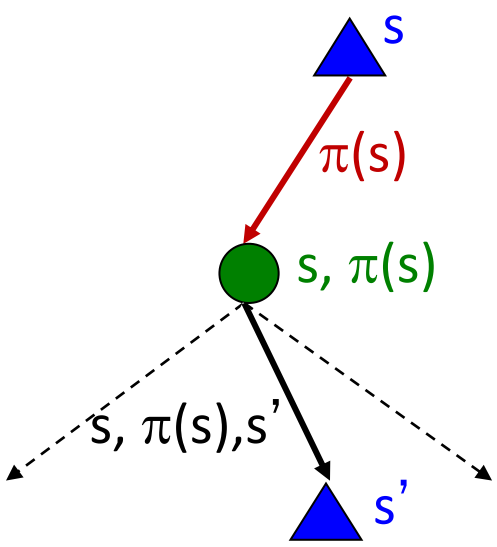
Utilities for a Fixed Policy

- **Define the value/utility of a state s, under a fixed policy π:**

  $V^{\pi}(s)$ = expected total discounted rewards starting in s and following π

- **Recursive relation**

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

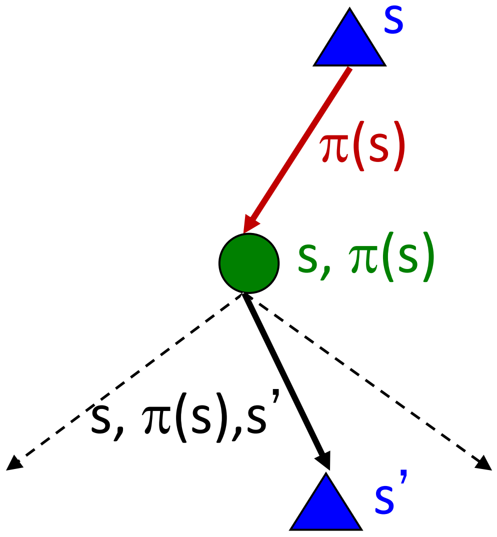


These slides were adapted from the MDP lecture slides of cs188 developed at UC Berkeley, http://ai.berkeley.edu.

Policy Evaluation

- **How do we calculate the values $V^{\pi}(s)$ for a fixed policy π?**

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



- **Idea 1: Turn recursive Bellman equations into updates (like value iteration)**

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

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

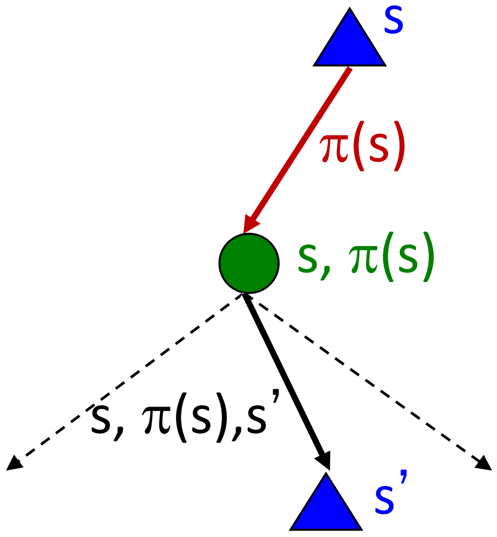
- **Complexity: $O(|S|^2)$ per iteration**

Policy Evaluation

- **How do we calculate the values $V^{\pi}(s)$ for a fixed policy π?**

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

- **Idea 2: Without the maxes, the Bellman equations are just a linear system**
  - Solve with Python (or your favorite linear system solver)



$$V^{\pi}(s_1) = f_1(V^{\pi}(s_1), V^{\pi}(s_2), \dots, V^{\pi}(s_N))$$
$$V^{\pi}(s_2) = f_2(V^{\pi}(s_1), V^{\pi}(s_2), \dots, V^{\pi}(s_N))$$
$$\vdots$$
$$V^{\pi}(s_N) = f_N(V^{\pi}(s_1), V^{\pi}(s_2), \dots, V^{\pi}(s_N))$$

Assume there are N states, then
we have N unknowns and N linear equations