# FIT 3080: Intelligent Systems

# Markov Decision Processes (MDPs)

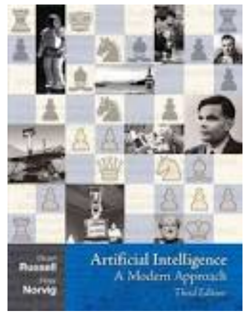
Gholamreza Haffari – Monash University

Many slides over the course adapted from Stuart Russell, Andrew Moore, or Dan Klein

# Announcements

- Online Reading:
  - Reinforcement Learning: An Introduction, by Richard Sutton and Andrew Barto, MIT Press
  - Chapter 3 and Chapter 4
  - Accessible from: http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html
  - Different treatment and notation than the R&N book, beware!
  - Lecture version is the standard for this class


- R&N book:
  - Sections 17.1-3



# Non-Deterministic Search

- How do you plan when your actions might fail?

# Outline

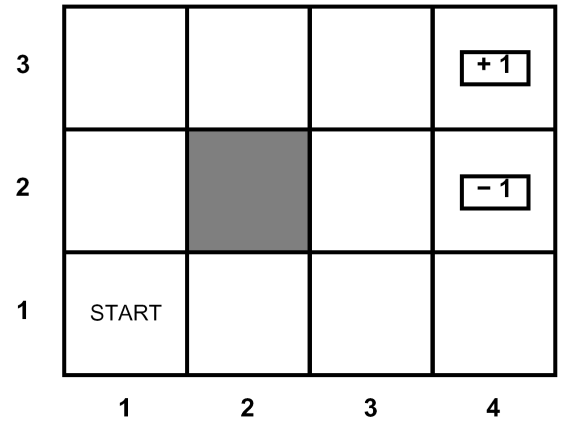
- MDPs
  - Definition
  - MDPs as Search Trees
  - Utility of a Sequence of Actions
- Solving MDPs
  - Bellman Equation
  - Value Iteration
  - Policy Iteration

# Outline

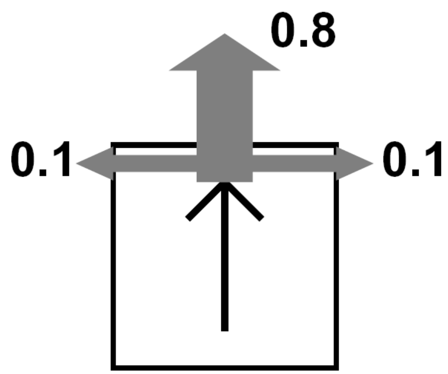
- MDPs
  - Definition
  - MDPs as Search Trees
  - Utility of a Sequence of Actions
- Solving MDPs
  - Bellman Equation
  - Value Iteration
  - Policy Iteration

# Grid World

- The agent lives in a grid
  - Agent has four actions: North, South, East, West
  - Walls block the agent's path
  - If outcome of actions are deterministic, we can use standard planning/search algorithms to go from the initial state to the favorable goal state

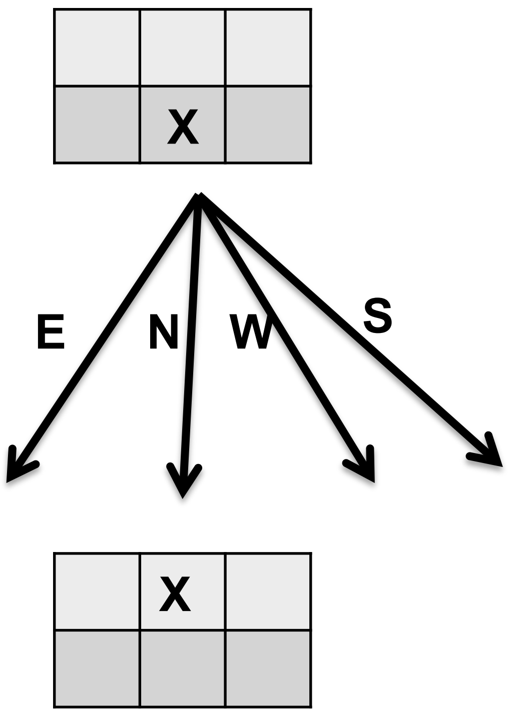



- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North
    (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put

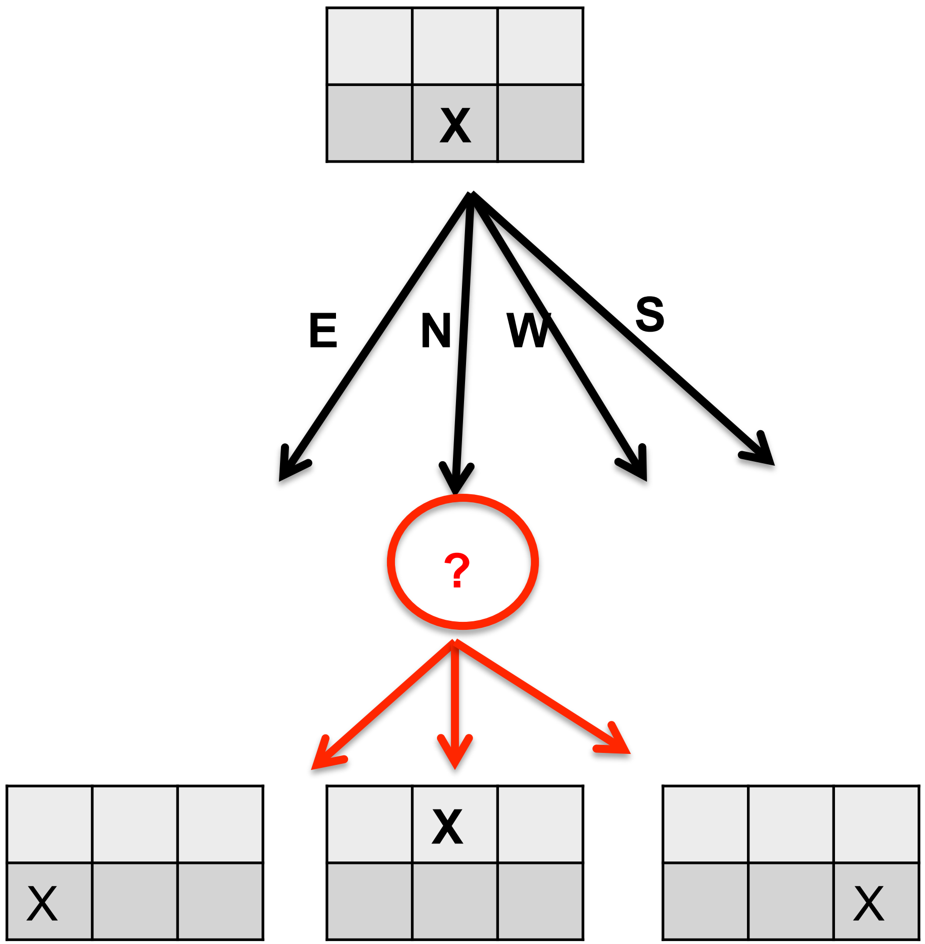


# Action Results

Deterministic Grid World

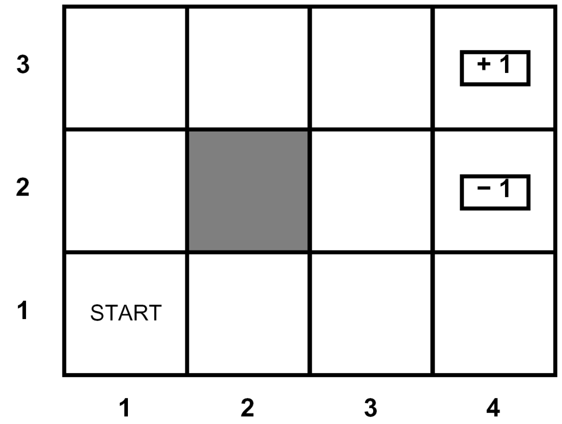



Non-Deterministic Grid World



# Grid World

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



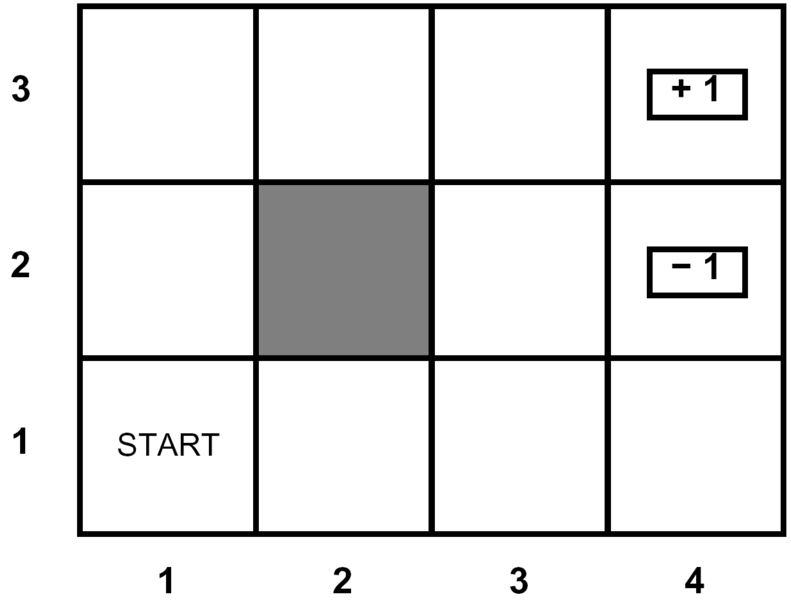

- Question: What is the best action to take from each state to maximize **sum of rewards**?

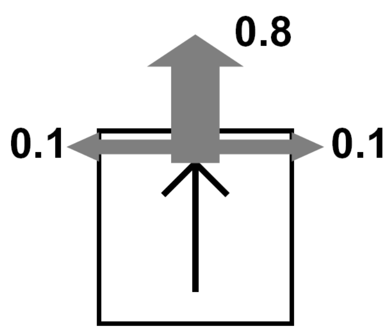
- This week lectures:
  - Firstly, formalize the problem more generally as a Markov Decision Process (MDP)
  - Secondly, present algorithms to solve MDPs (to solve stochastic planning/game problems)

# 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



# What is Markov about MDPs?

- Andrey Markov (1856-1922)
- “Markov” generally means that given the present state, the future and the past are independent
- For Markov decision processes, “Markov” means:

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

$$=$$

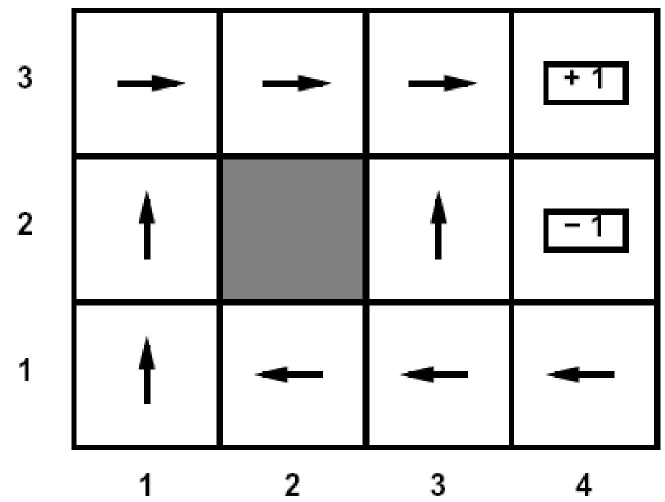
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

# Solving MDPs

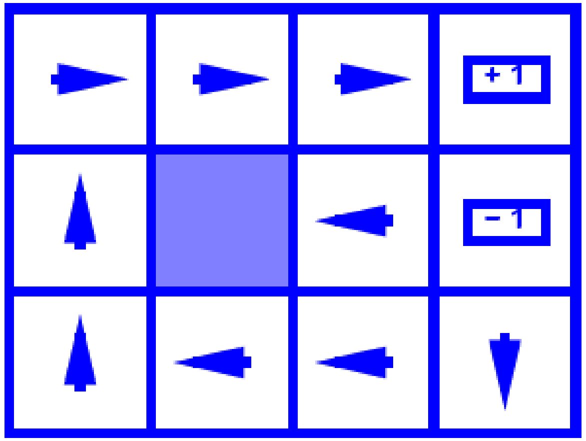
- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal

- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy $\pi$ gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent

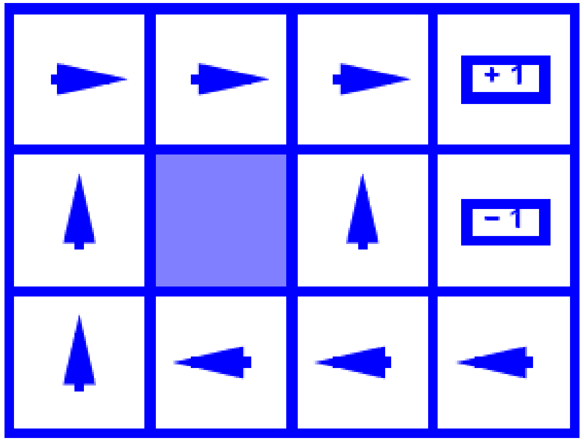
Optimal policy when R(s, a, s' ) = -0.03 for all non-terminals s



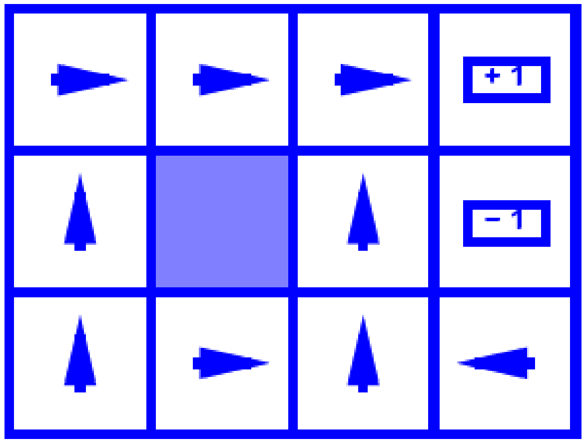
# Example Optimal Policies



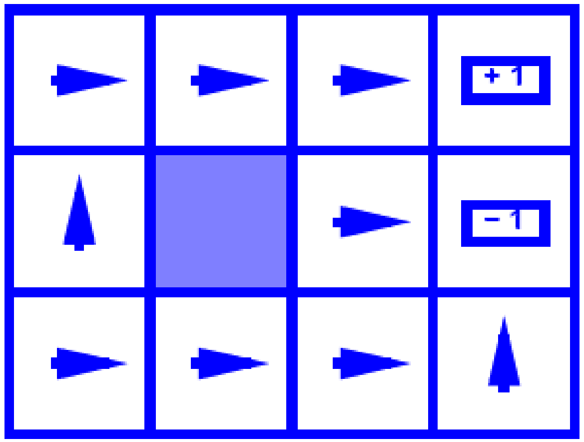

R(s) = -0.01




R(s) = -0.03



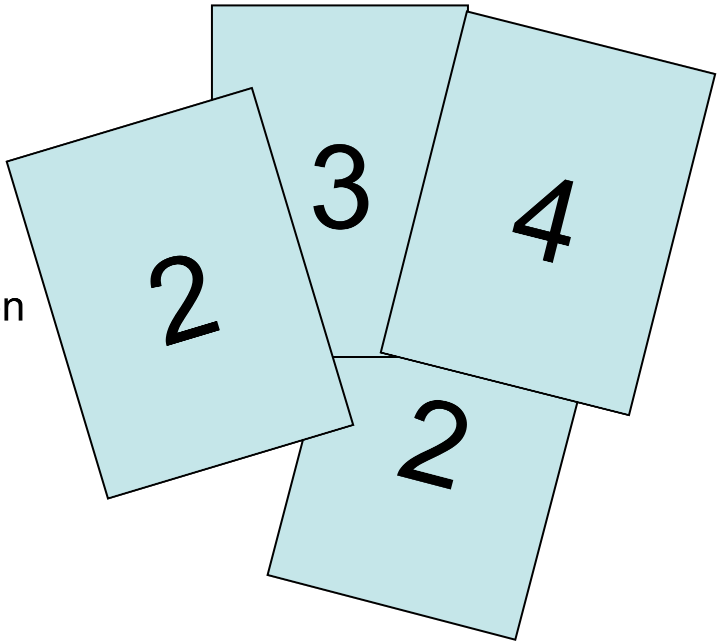

R(s) = -0.4




R(s) = -2.0

# Example: High-Low

- Rules:
  - Three card types: 2, 3, 4
    - Infinite deck, twice as many 2's
  - Start with 3 showing
  - After each card, you say "high" or "low"
    - New card is flipped
    - If you're right, you win the points shown on the new card
    - Ties are no-ops
    - If you're wrong, game ends



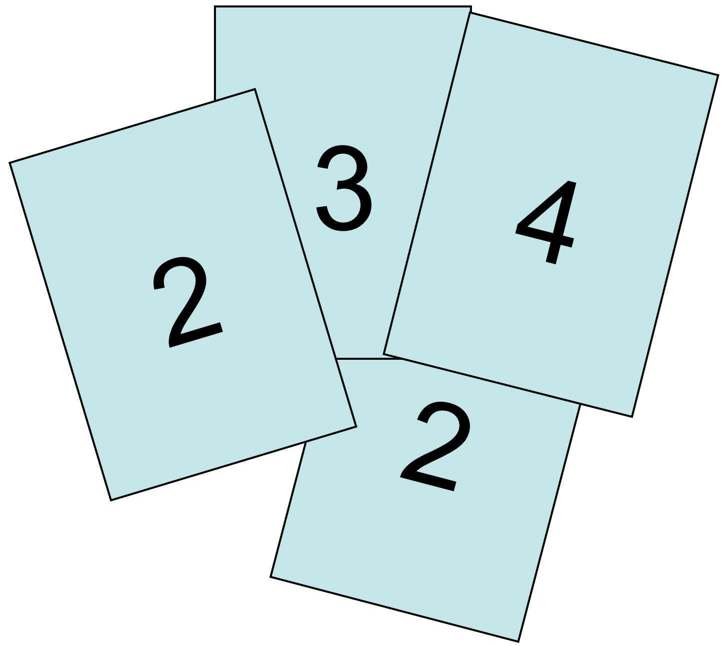

- How is this different from the "chance" games in the last lecture?
  - #1: get rewards as you go
  - #2: you might play forever

You can patch expectimax to deal with #1 but not #2

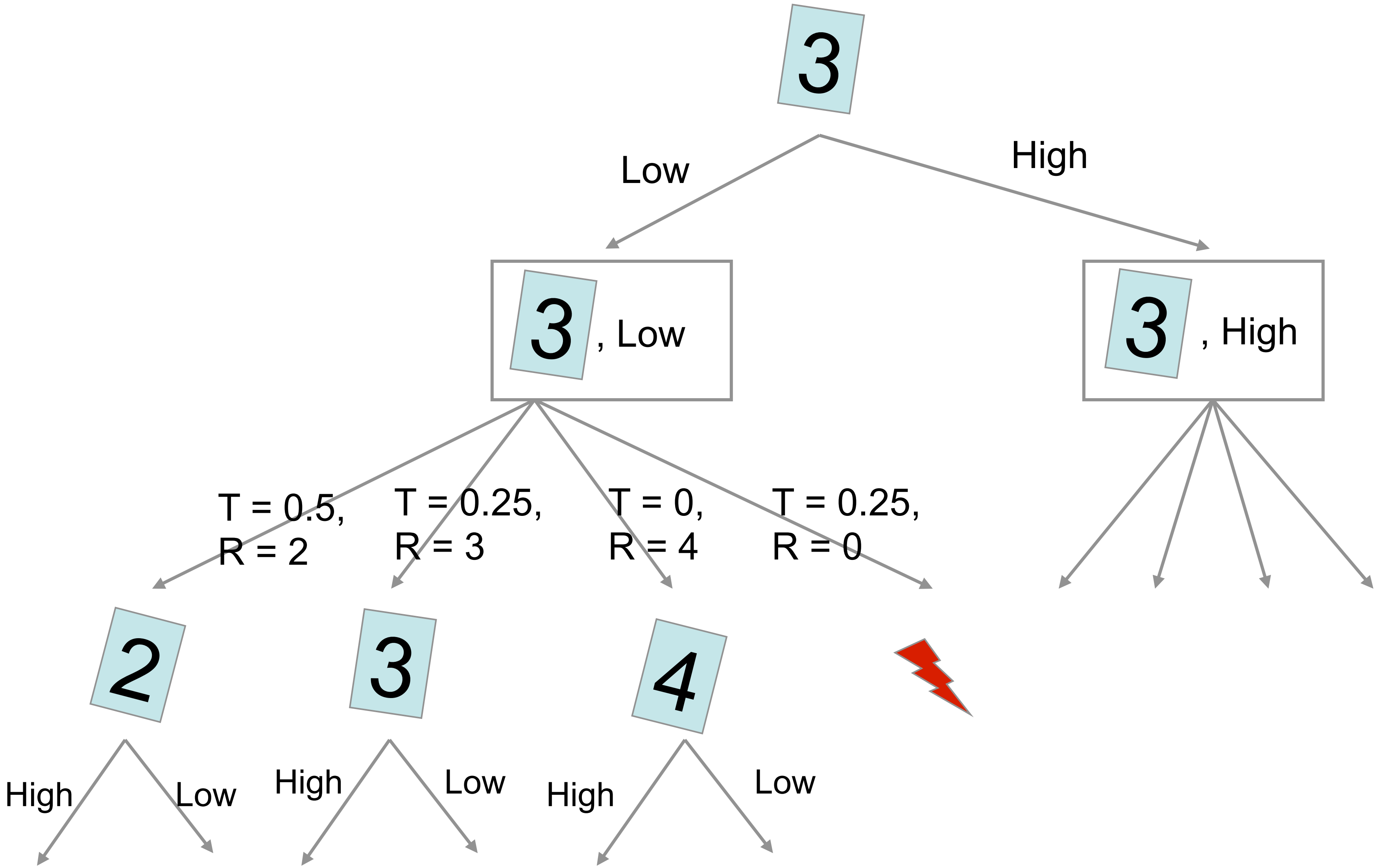
# High-Low as an MDP

- States: 2, 3, 4, done
- Actions: High, Low
- Model: T(s, a, s'):
  - P(s' =4 | 4, Low) = 1/4
  - P(s' =3 | 4, Low) = 1/4
  - P(s' =2 | 4, Low) = 1/2
  - P(s' =done | 4, Low) = 0

  - P(s' =4 | 4, High) = 1/4
  - P(s' =3 | 4, High) = 0
  - P(s' =2 | 4, High) = 0
  - P(s' =done | 4, High) = 3/4
  - …
- Rewards: R(s, a, s'):
  - Number shown on s' if $s \neq s'$
  - 0 otherwise



# Example: High-Low



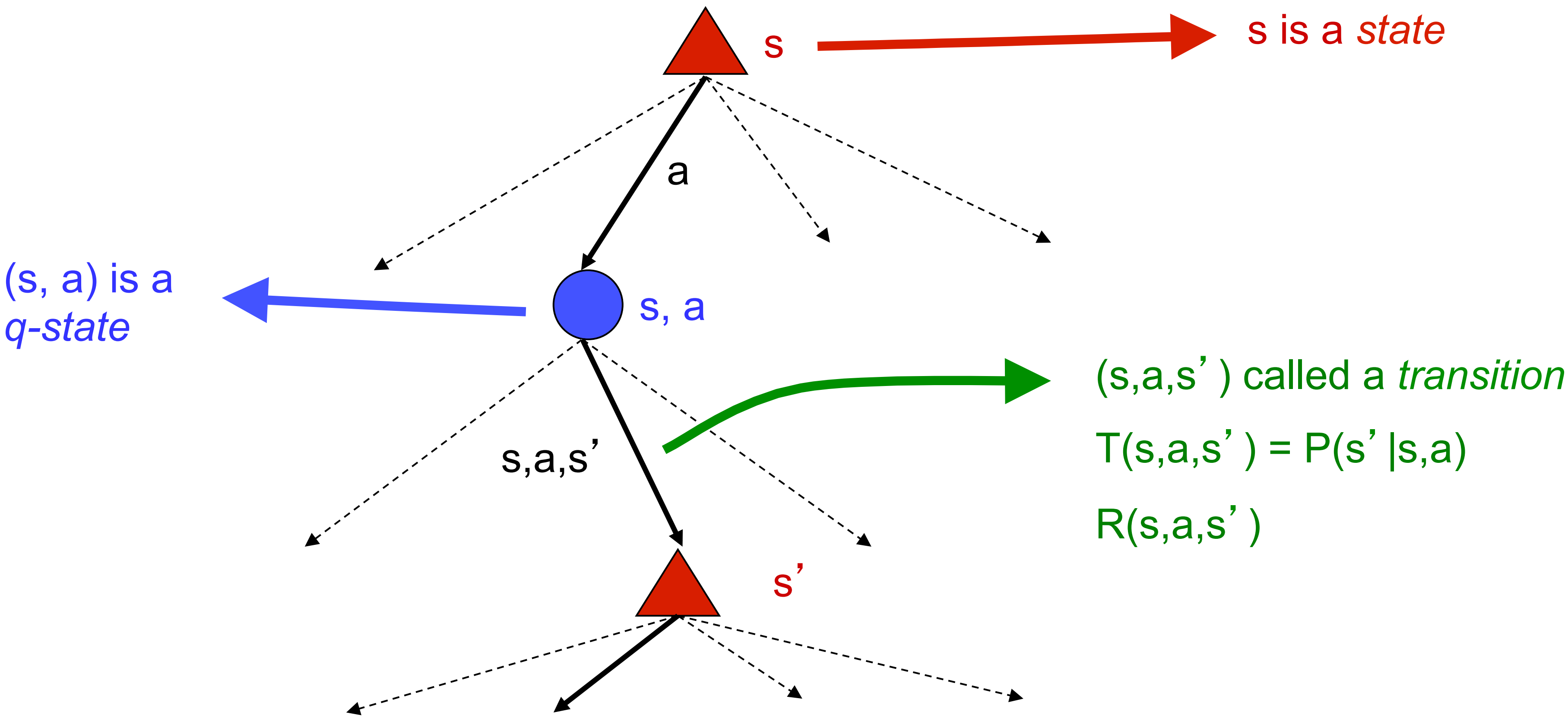
# Outline

- MDPs
  - Definition
  - MDPs as Search Trees
  - Utility of a Sequence of Actions

- Solving MDPs
  - Bellman Equation
  - Value Iteration
  - Policy Iteration

# MDP Search Trees

- Each MDP state gives an “expectimax” search tree



# Outline

- MDPs
  - Definition
  - MDPs as Search Trees
  - Utility of a Sequence of Actions
- Solving MDPs
  - Bellman Equation
  - Value Iteration
  - Policy Iteration

# Utilities of Sequences

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards
- Typically consider stationary preferences:

$$[r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots]$$
$$\Leftrightarrow$$
$$[r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots]$$

- Theorem: only two ways to define stationary utilities
  - Additive utility:

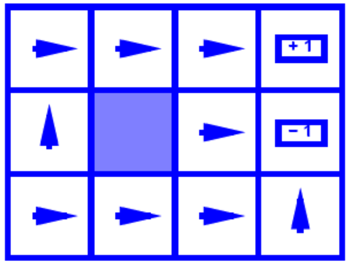
$$U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$$

  - Discounted utility:

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

# Infinite Utilities?!

- Problem: infinite state sequences have infinite rewards



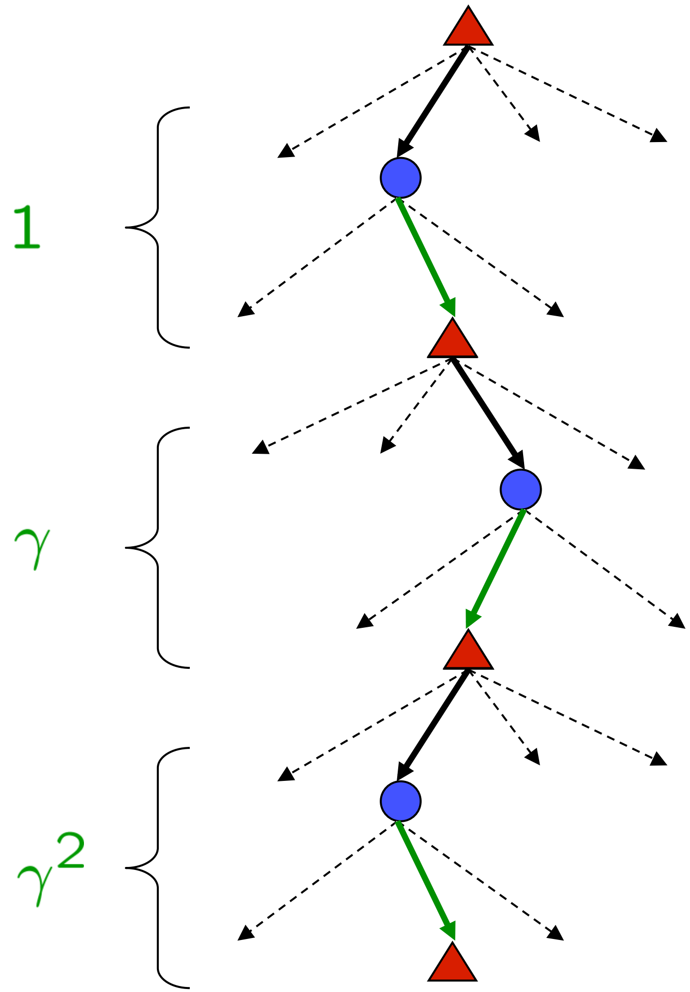

- Solutions:
  - Finite horizon:
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies ($\pi$ depends on time left)
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "done" for High-Low)
  - Discounting: for $0 < \gamma < 1$

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

    - Smaller $\gamma$ 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

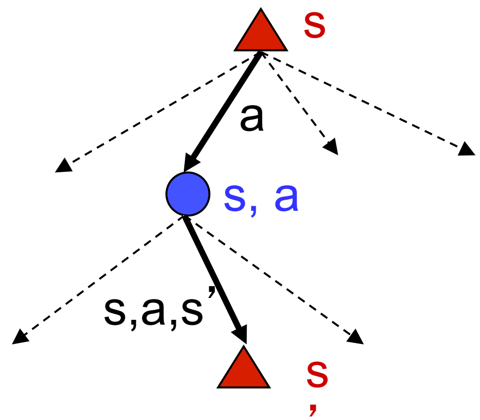


# Outline

- MDPs
  - Definition
  - MDPs as Search Trees
  - Utility of a Sequence of Actions
- Solving MDPs
  - Bellman Equation
  - Value Iteration
  - Policy Iteration

# 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 $\gamma$)




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

# Optimal Utilities

- Define the value of a state s:

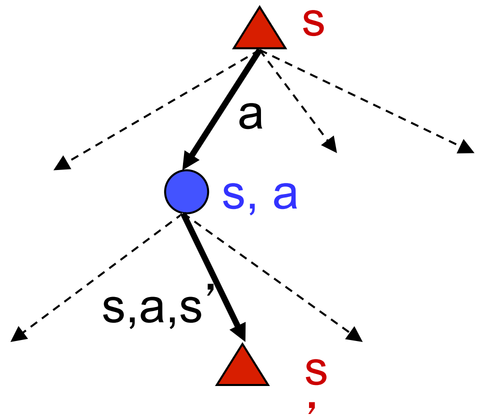
  $V^*(s)$ = expected utility starting in s and acting optimally

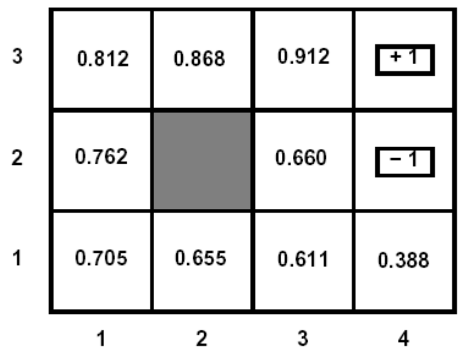
- Define the value of a q-state (s,a):

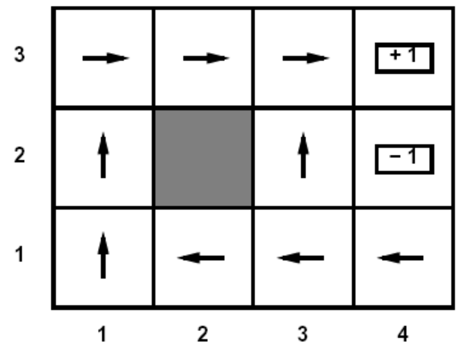
  $Q^*(s,a)$ = expected utility starting in s, taking action a and thereafter acting optimally

- Define the optimal policy:

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



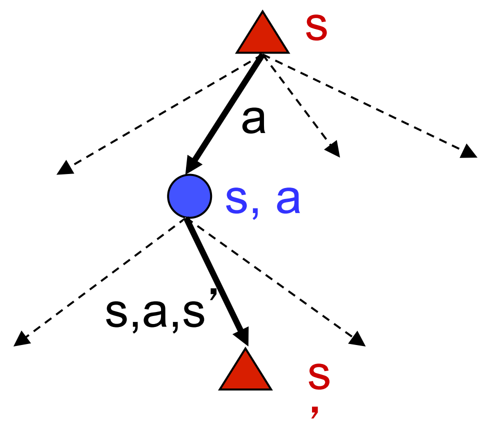
# Solving MDPs

- We want to find the optimal policy $\pi^*$

- How to do that?

# 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:

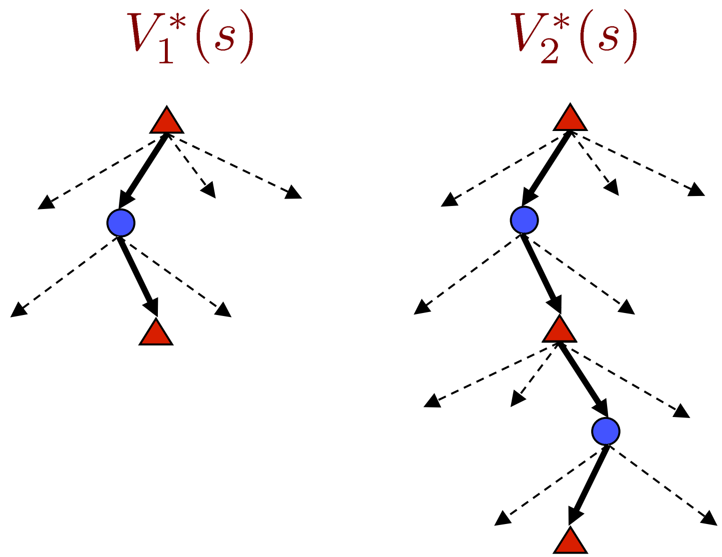
$$V^*(s) = \max_a Q^*(s,a)$$

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

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

# Value Estimates

- Calculate estimates $V_k^*(s)$
  - Not the optimal value of s!
  - The optimal value considering only next k time steps (k rewards)
  - As $k \rightarrow \infty$, it approaches the optimal value
- Almost solution: recursion (i.e. expectimax)
- Correct solution: dynamic programming



# Outline

- MDPs
  - Definition
  - MDPs as Search Trees
  - Utility of a Sequence of Actions
- Solving MDPs
  - Bellman Equation
  - Value Iteration
  - Policy Iteration

# Value Iteration

- Idea:
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given $V_i^*$, calculate the values for all states for depth i+1:

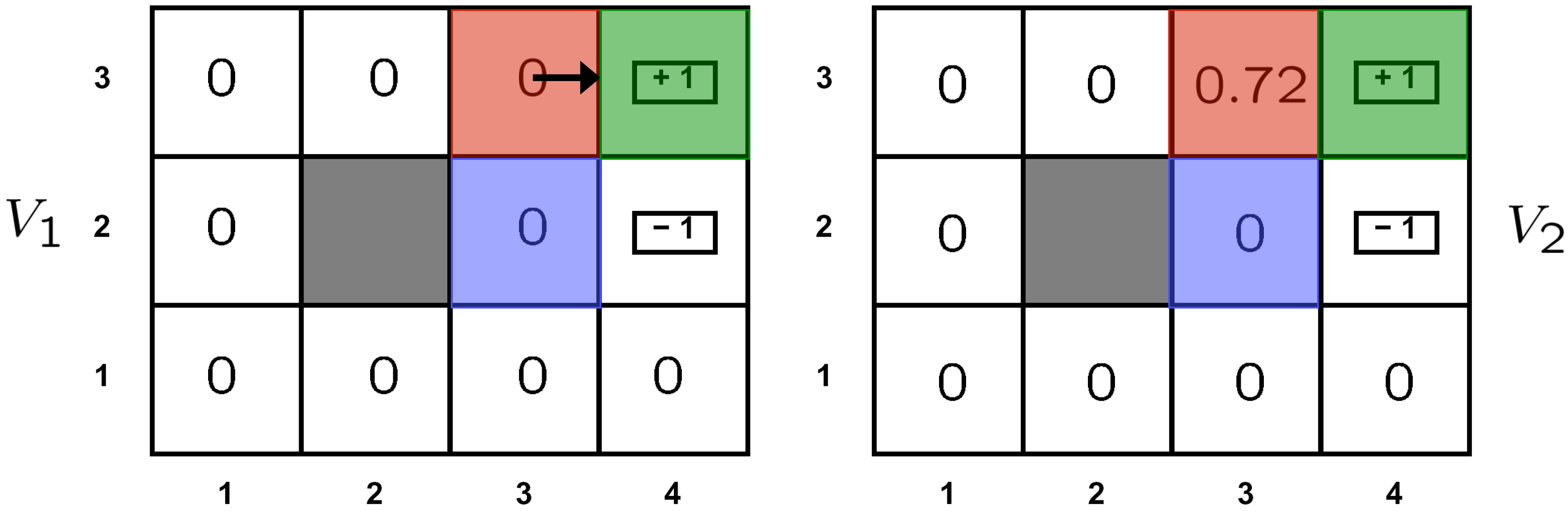
$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V_i(s') \right]$$

  - 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: $\gamma$=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

$V_2$

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

$V_3$

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

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

# Convergence*

- Define the max-norm: $||U|| = \max_s |U(s)|$
- Theorem: For any two approximations U and V

$$||U^{t+1} - V^{t+1}|| \leq \gamma \, ||U^t - V^t||$$

  - I.e. any distinct approximations must get closer to each other, so, in particular, any approximation must get closer to the true U and value iteration converges to a unique, stable, optimal solution
- Theorem:

$$||U^{t+1} - U^t|| < \epsilon, \Rightarrow ||U^{t+1} - U|| < 2\epsilon\gamma/(1-\gamma)$$

  - I.e. once the change in our approximation is small, it must also be close to correct

# Practice: 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!

# Aside: Q-Value Iteration

- Value iteration: find successive approx optimal values
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given $V_i^*$, calculate the values for all states for depth i+1:

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

- But Q-values are more useful!
  - Start with $Q_0^*(s,a) = 0$, which we know is right (why?)
  - Given $Q_i^*$, calculate the q-values for all q-states for depth i+1:

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

# Outline

- MDPs
  - Definition
  - MDPs as Search Trees
  - Utility of a Sequence of Actions
- Solving MDPs
  - Bellman Equation
  - Value Iteration
  - Policy Iteration

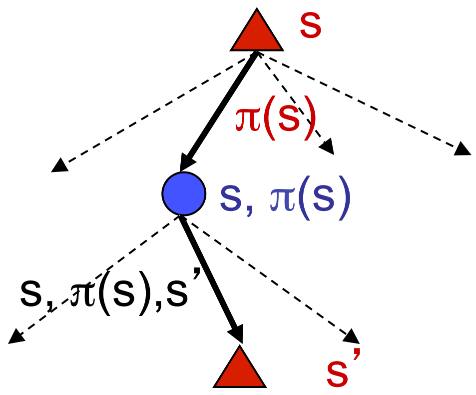
# Utilities for Fixed Policies

- Another basic operation: compute the utility of a state s under a fix (general non-optimal) policy

- Define the utility of a state s, under a fixed policy $\pi$:

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

- Recursive relation (one-step look-ahead / Bellman equation):




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

# Policy Evaluation

- How do we calculate the V’s for a fixed policy?
- Idea one: modify Bellman updates

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

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

- Idea two: it’s just a linear system, solve with Matlab (or whatever)

# Policy Iteration

- Problem with value iteration:
  - Considering all actions each iteration is slow: takes |A| times longer than policy evaluation
  - But policy doesn't change each iteration, time wasted

- Alternative to value iteration:
  - Step 1: Policy evaluation: calculate utilities for a fixed policy (not optimal utilities!) until convergence (fast)
  - Step 2: Policy improvement: update policy using one-step lookahead with resulting converged (but not optimal!) utilities (slow but infrequent)
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge faster under some conditions

# Policy Iteration

- Policy evaluation: with fixed current policy $\pi$, find values with simplified Bellman updates
  - Iterate until values converge

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

- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead

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

# Policy Iteration Complexity

- Problem size:
  - |A| actions and |S| states
- Each Iteration
  - Computation: $O(|S|^3 + |A|.|S|^2)$
  - Space: $O(|S|)$
- Number of iterations
  - Unknown, but can be faster in practice
  - Convergence is guaranteed

# Comparison

- In value iteration:
  - Every pass (or "backup") updates both utilities (explicitly, based on current utilities) and policy (possibly implicitly, based on current policy)
- In policy iteration:
  - Several passes to update utilities with frozen policy
  - Occasional passes to update policies
- Hybrid approaches (asynchronous policy iteration):
  - Any sequences of partial updates to either policy entries or utilities will converge if every state is visited infinitely often