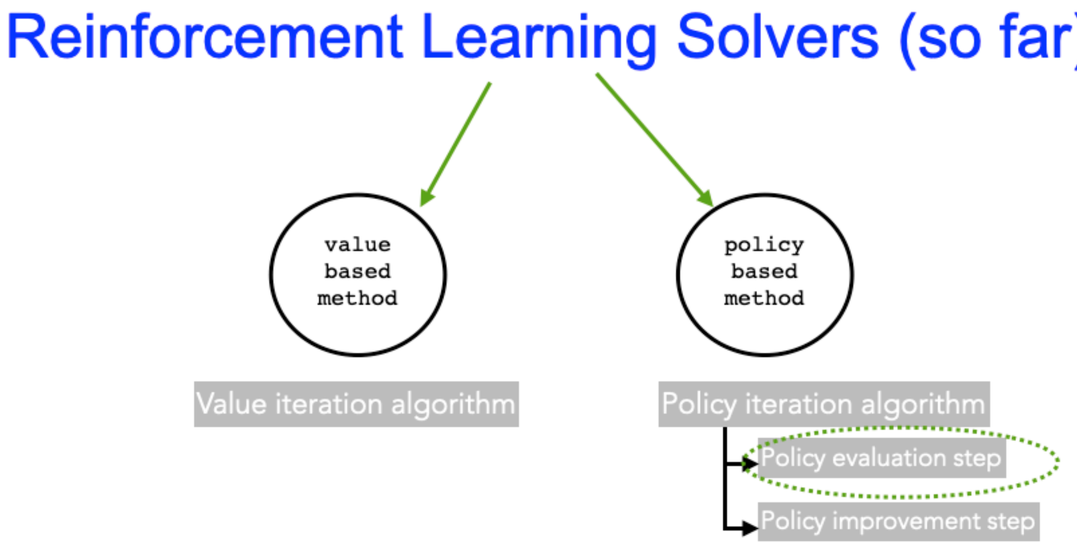
CS143: Artificial Intelligence



Policy Iteration Algorithm

Drake UNIVERSITY

What is the MDP for Frozen Lake?

- The transition function encodes the movements in the Lake
  - In Frozen Lake, model or dynamics is known
  - $P(s'|a,s)$ is read as agent transitions from state s to s' by taking action a

- How many states are there?
  - How can you show the probability transitions $P(s'|a,s)$ from each state to the next?

- Let's simplify and consider a simpler problem where there only a couple of states
  - Let's consider a simple dice game with 2 states only

MDP for a Simple Dice Game with 2 States

MDP for a Simple Dice Game

- For each round t = 1,2,3, …
  - You choose stay or quit
    - If quit, you get $10 and we end the game
    - If stay, you get $4 and then I roll a 6-sided dice
      - If the dice results in 1 or 2, we end the game
      - Otherwise, if the dice results in 3, 4, 5, or 6, we continue to the next round

MDP for a Simple Dice Game

- You choose stay or quit
  - If quit, you get $10 and we end the game
  - If stay, you get $4 and then I roll a 6-sided dice
    - If the dice results in 1 or 2, we end the game
    - Otherwise, if the dice results in 3, 4, 5, or 6, we continue to the next round

WHAT IS THE BEST STRATEGY FOR THIS GAME?

stay

Expected rewards:

12.0

quit

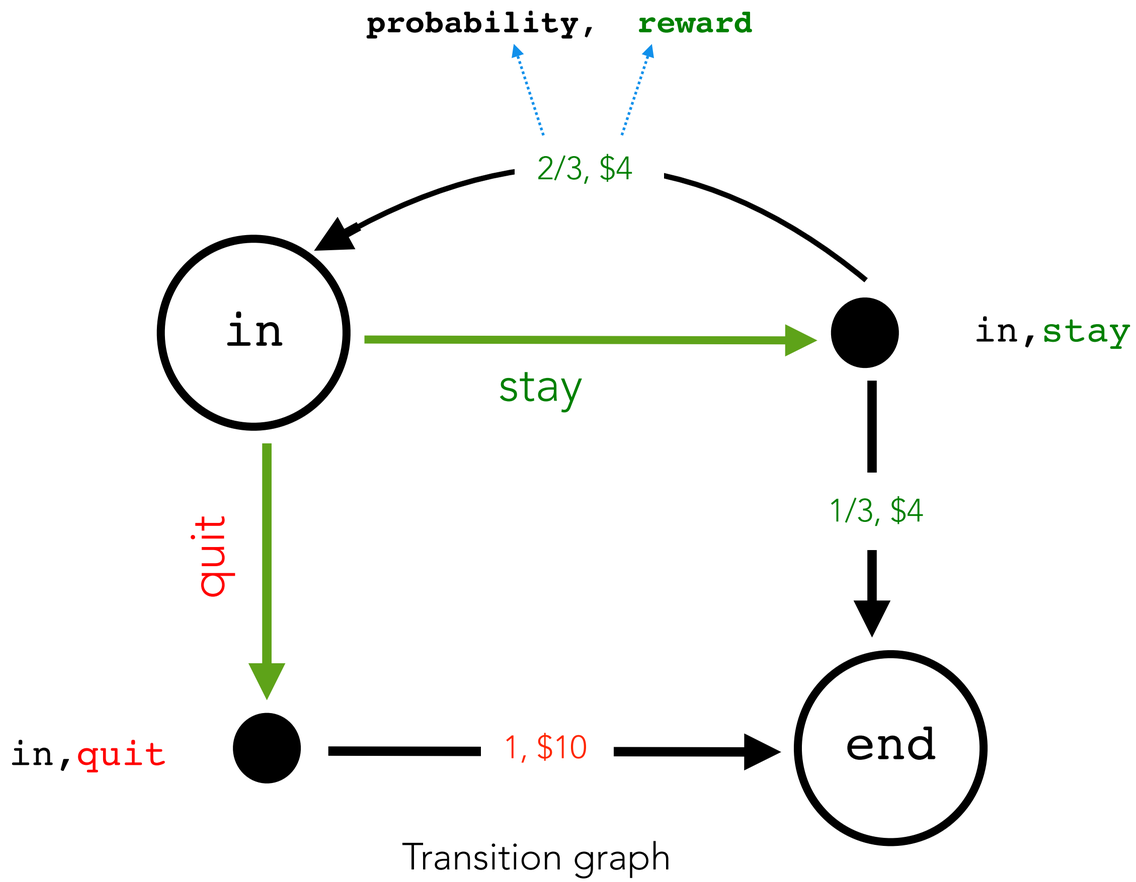
Expected rewards:

10.0

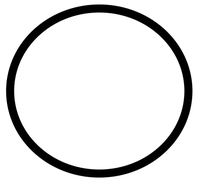
If you quit, then you'll get a reward of $10. Therefore, the stay strategy is preferred, even though sometimes you'll get less than $10.

MDP for a Simple Dice Game

- Let us formalize the dice game as a Markov decision process (MDP)



Transition graph



state node

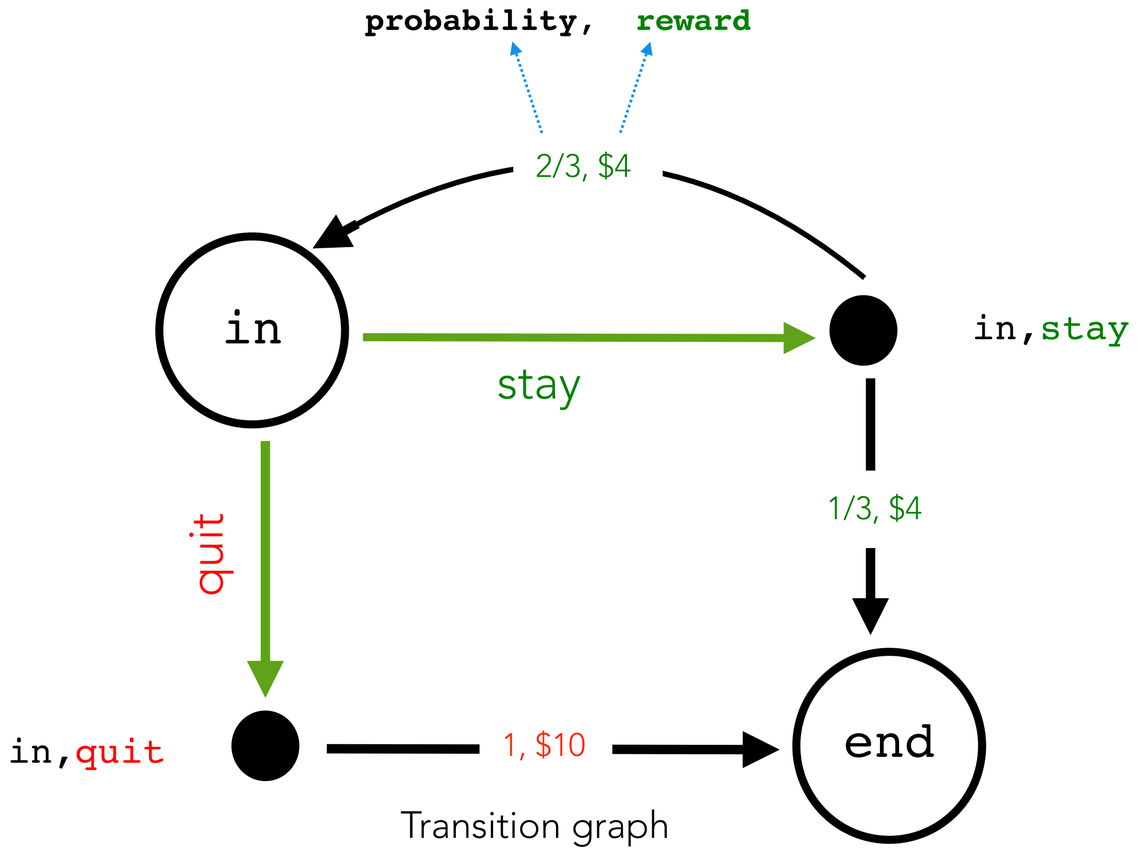
- Edges coming out of a state node are the possible actions from that state node leading to a action node

action node

- Edges coming out of a action node are the possible random outcomes of that action, which end up back in state nodes
- Sum of probabilities coming out of a action node is 1.0

Optimal state-value or value function

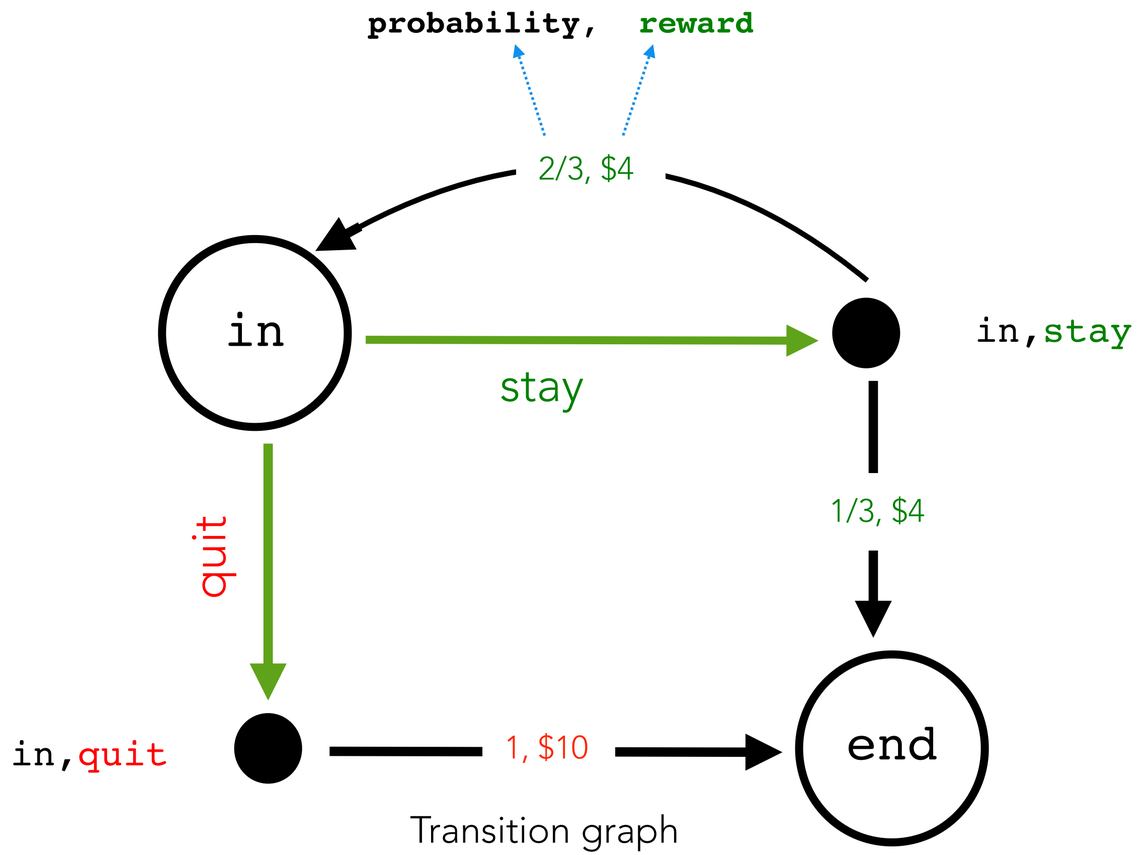
- **value function** $V_{opt}(s)$: estimate of *how good* it is to be in a given state ***s*** and acting optimally
- $V_{opt}(s)$ is the expected reward received when starting in state ***s*** and following optimal policy π_{opt} thereafter



Transition graph

Optimal action-value or q-value function

- **q-value function** $Q_{opt}(s, a)$: estimate of the expected return obtained by taking action ***a*** in state ***s***, and thereafter following the optimal policy π_{opt} optimally thereafter



Transition graph

Relationship: value and q-value functions

- Looks like from the previous to visualizations, the relationship between the **value function** and the **Q-value** function can be expressed recursively. For example, from the previous tree graph, it can be seen that the value function can be computed by taking the maximum over multiple Q-values corresponding to different actions. Mathematically, this can be written as follows:

$$V_{opt}(s) = \max \begin{Bmatrix} Q_{opt}(s, quit) \\ Q_{opt}(s, stay) \end{Bmatrix}$$

THERE IS ONLY ONE NON-TERMINAL STATE: in

$V_{opt}(in) = ?$

TERMINAL STATE: end

$V_{opt}(end) = 0$

$$V_{opt}(in) = \max \begin{Bmatrix} P(in \mid s, quit)[R(s, quit, in) + \gamma V_{opt}(in)] + P(end \mid s, quit)[R(s, quit, end) + \gamma V_{opt}(end)] \\ P(in \mid s, stay)[R(s, stay, in) + \gamma V_{opt}(in)] + P(end \mid s, stay)[R(s, stay, end) + \gamma V_{opt}(end)] \end{Bmatrix}$$

$$= \max \begin{Bmatrix} P(end \mid s, quit)[R(s, quit, end) + \gamma V_{opt}(end)] \\ P(in \mid s, stay)[R(s, stay, in) + \gamma V_{opt}(in)] + P(end \mid s, stay)[R(s, stay, end) + \gamma V_{opt}(end)] \end{Bmatrix}$$

$Q_{opt}(in, quit)$

$$\frac{2}{3}(4 + \gamma V_{opt}(in)) + \frac{1}{3}(4 + \gamma V_{opt}(end))$$

p(s'|s,a) r(s,a,s') Discount p(s'|s,a) r(s,a,s') Discount

$Q_{opt}(in, stay)$

Relationship: value and q-value functions

- In general, we can write the value function can be computed by taking the maximum over multiple Q-values corresponding to different actions. Mathematically, this can be written as follows:

$$V_{opt}(s) = \max_a Q_{opt}(s,a)$$
$$= \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_{opt}(s')]$$

Question: Then how do you solve an MDP?

Solution to an MDP is called a **‘policy’**

Answer: Value Iteration Algorithm

An iterative algorithm to find the **optimal policy** (best solution) for an MDP

Policy: Solution to an MDP

- What is a solution to an MDP?
  - Solution to an MDP is a **policy** π which is a mapping **from each state** *s* to an action *a*

s	$\pi(s)$
in	stay
end	-

- In search problem, what was the solution?
  - Solution was the path, which was a sequence of actions from the **start state** to **goal state** only
  - Eg, In an arbitrary 8-puzzle problem the solution was *UP, LEFT, LEFT, RIGHT, …*

Mathematically: How to calculate V(s) values?

- How do we calculate $V_{opt}(in)$(optimal V-values for each state) from the recursive Bellman equation:

$$V_{opt}(in) = \max_{a \in actions(s)} \sum_{s'} p(s' \mid in, a)[R(in, a, s') + \gamma V_{opt}(s')]$$

A RECURRENCE RELATION

- Turn recursive Bellman equations into update equations:

$$V^0_{opt}(in) = 0$$

$$V^t_{opt}(in) = \max_{a \in actions(s)} \sum_{s'} p(s' \mid in, a)[R(in, a, s') + \gamma V^{t-1}_{opt}(s')]$$

Mathematically: How to calculate V(s) values?

- More precisely the update equations are as follows:

NON-TERMINAL STATE

$$V^t_{opt}(in) = \max \left\{ \begin{array}{c} Q^t_{opt}(in, quit) \\ Q^t_{opt}(in, stay) \end{array} \right\}$$

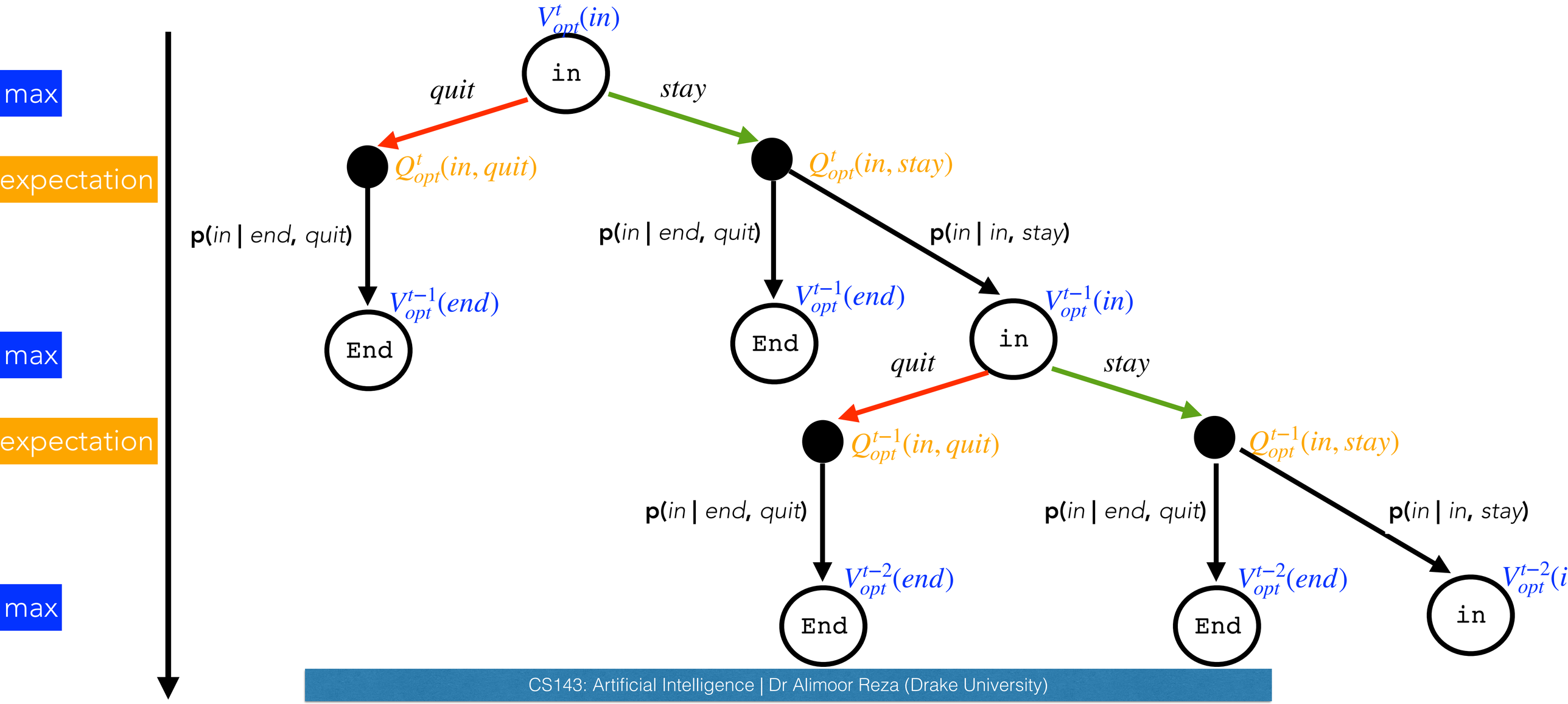
$$= \max \left\{ \begin{array}{c} \frac{1}{1}(10 + \gamma V^{t-1}_{opt}(end)) \\ \frac{2}{3}(4 + \gamma V^{t-1}_{opt}(in)) + \frac{1}{3}(4 + \gamma V^{t-1}_{opt}(end)) \end{array} \right\}$$

TERMINAL STATE

$$V_{opt}(end) = 0$$

THE RECURRENCE RELATION IS NOW EXPRESSED AS ITERATIVE UPDATES THAT CAN BE COMPUTED INCREMENTALLY

Build an Expectiminimax-like search tree



Algorithmically: Value iteration algorithm

INITIALIZE VALUES WITH SOME ROUGH ESTIMATES

REFINE IT OVER AND OVER AGAIN UNTIL THEIR VALUES DO NOT CHANGE

Algorithm 2: Value Iteration Algorithm

Initialize state-value $v^0_{opt}(s)$ for all the states s

//Repeat the iterative procedure for some iterations, let's say $T = 10000$

for $t = 1$ **to** T **do**

  for *each state* s **do**

    $v^t_{opt}(s) \leftarrow \max_{a \in actions(s)} \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v^{t-1}_{opt}(s')]$

This is because the "value iteration algorithm" is now finding the best policy out of all possible candidate policies

Value iteration algorithm on our MDP

TERMINAL STATE

$$V_{opt}(end) = 0$$

NON-TERMINAL STATE

$$V^t_{opt}(in) = \max \left\{ \begin{array}{c} Q^t_{opt}(in, quit) \\ Q^t_{opt}(in, stay) \end{array} \right\}$$

$$= \max \left\{ \begin{array}{c} \frac{1}{1}(10 + \gamma V^{t-1}_{opt}(end)) \\ \frac{2}{3}(4 + \gamma V^{t-1}_{opt}(in)) + \frac{1}{3}(4 + \gamma V^{t-1}_{opt}(end)) \end{array} \right\}$$

Algorithm 2: Value Iteration Algorithm

Initialize state-value $v^0_{opt}(s)$ for all the states s

//Repeat the iterative procedure for some iterations, let's say $T = 10000$

for $t = 1$ **to** T **do**

  for *each state* s **do**

    $v^t_{opt}(s) \leftarrow \max_{a \in actions(s)} \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v^{t-1}_{opt}(s')]$

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
$V_{opt}(end)$	0	0	0	0	0	0	0	0	0	0
$\pi_{opt}(end)$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
$V_{opt}(in)$	0									
$\pi_{opt}(in)$										

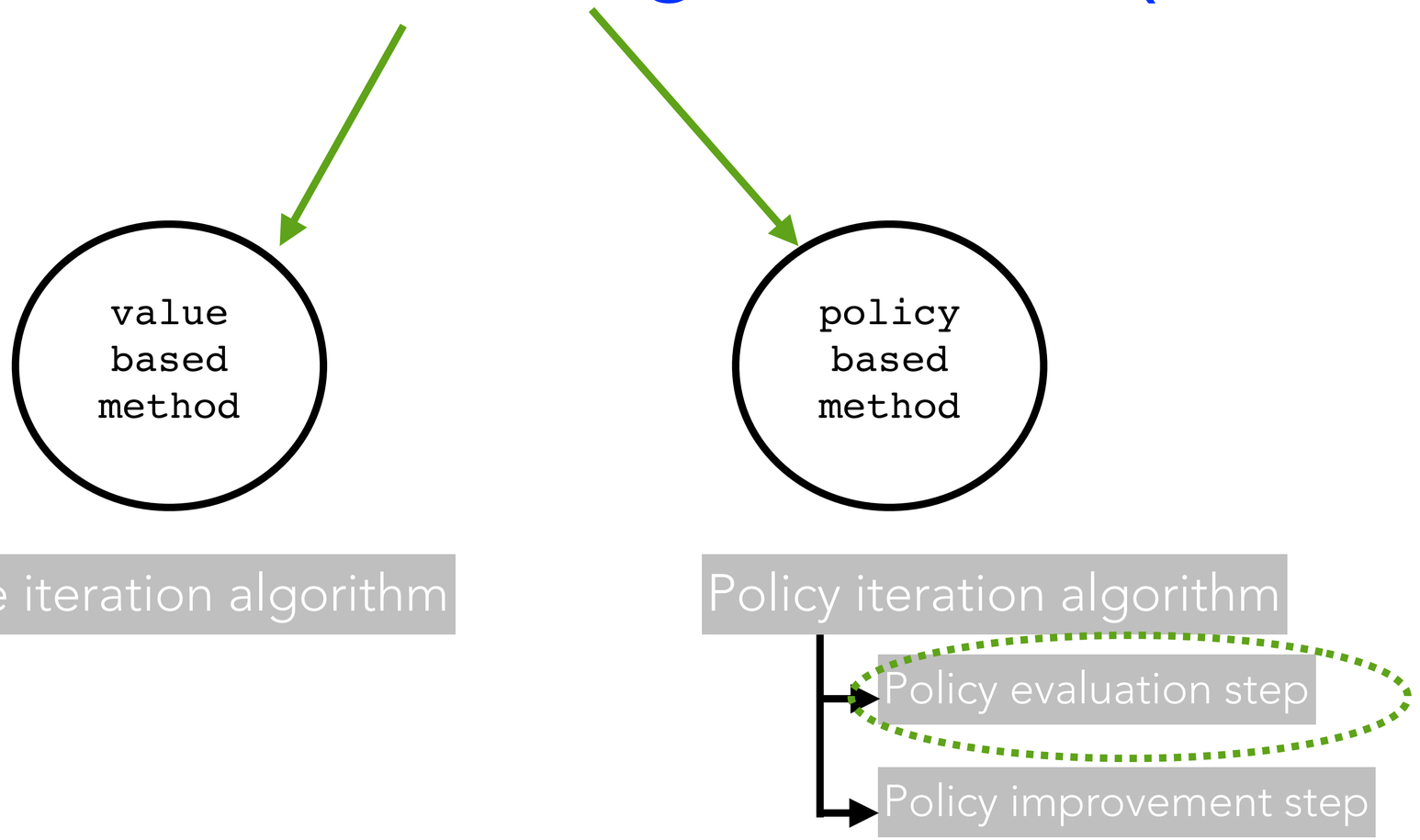
Question: Is there any other way to solve an MDP besides the Value Iteration algorithm?

Solution to an MDP is called a **‘policy’**

Answer (yes): Policy Iteration Algorithm

Another iterative algorithm to find the **optimal policy** (best solution) for an MDP

Reinforcement Learning Solvers (so far)



Let's discuss **Policy Evaluation Algorithm** first as it will be used by Policy Iteration Algorithm

- For an arbitrary MDP and **its given policy**, we can **apply** policy evaluation algorithm repeatedly until the the state-value and action-values converge for each state

- We can apply this on a very large MDP, not just the ones with 2 states (eg, our simple dice game)

- Let's say for our dice game, we are **given a policy** that returns action **'stay'** for each state. Since we have only one non-terminal state **'in'**, it can be written as follows:

$$\pi(in) = stay$$

Policy evaluation algorithm

INITIALIZE STATE-VALUES WITH SOME ROUGH ESTIMATES

REFINE IT OVER AND OVER AGAIN UNTIL THEIR VALUES DO NOT CHANGE

Algorithm 1: Policy Evaluation Algorithm

Initialize state-value $v_\pi^0(s)$ for all the states s

//Repeat the iterative procedure for some iterations, let's say $T = 10000$

for $t = 1$ **to** T **do**

  for *each state* s **do**

    $v_\pi^t(s) \leftarrow \sum_{s'} p(s'|s, \pi(s))[r(s, \pi(s), s') + \gamma v_\pi^{t-1}(s')]$

It is a specific action returned by **the given policy** $\pi(s)$

Policy evaluation algorithm

- We can apply policy evaluation algorithm repeatedly until the state-values **converge** for all the states

- How to find **convergence**? Find the difference of a state-value between two adjacent time steps until it goes down below a small threshold

$$|\, v_\pi^t(s) - v_\pi^{t-1}(s) \,| \leq \epsilon$$
$$\leq 0.001$$

Mathematically: How to calculate $V_\pi(s)$ values?

- How do we calculate $V_\pi(in)$ for **a fixed policy** $\pi(in)$ from the recursive Bellman equation:

$$V_\pi(in) = \sum_{s'} p(s' \mid in, \pi(s))[R(in, \pi(s), s') + \gamma V_\pi(s')]$$

A RECURRENCE RELATION

- Turn recursive Bellman equations into update equations:

$$V_\pi^0(in) = 0$$

$$V_\pi^t(in) = \sum_{s'} p(s' \mid in, \pi(in))[R(in, \pi(in), s') + \gamma V_\pi^{t-1}(s')]$$

Mathematically: How to calculate $V_\pi(s)$ values?

- More precisely the update equations are as follows:

NON-TERMINAL STATE

$$V_\pi^t(in) = Q_\pi^t(in, \pi(in))$$

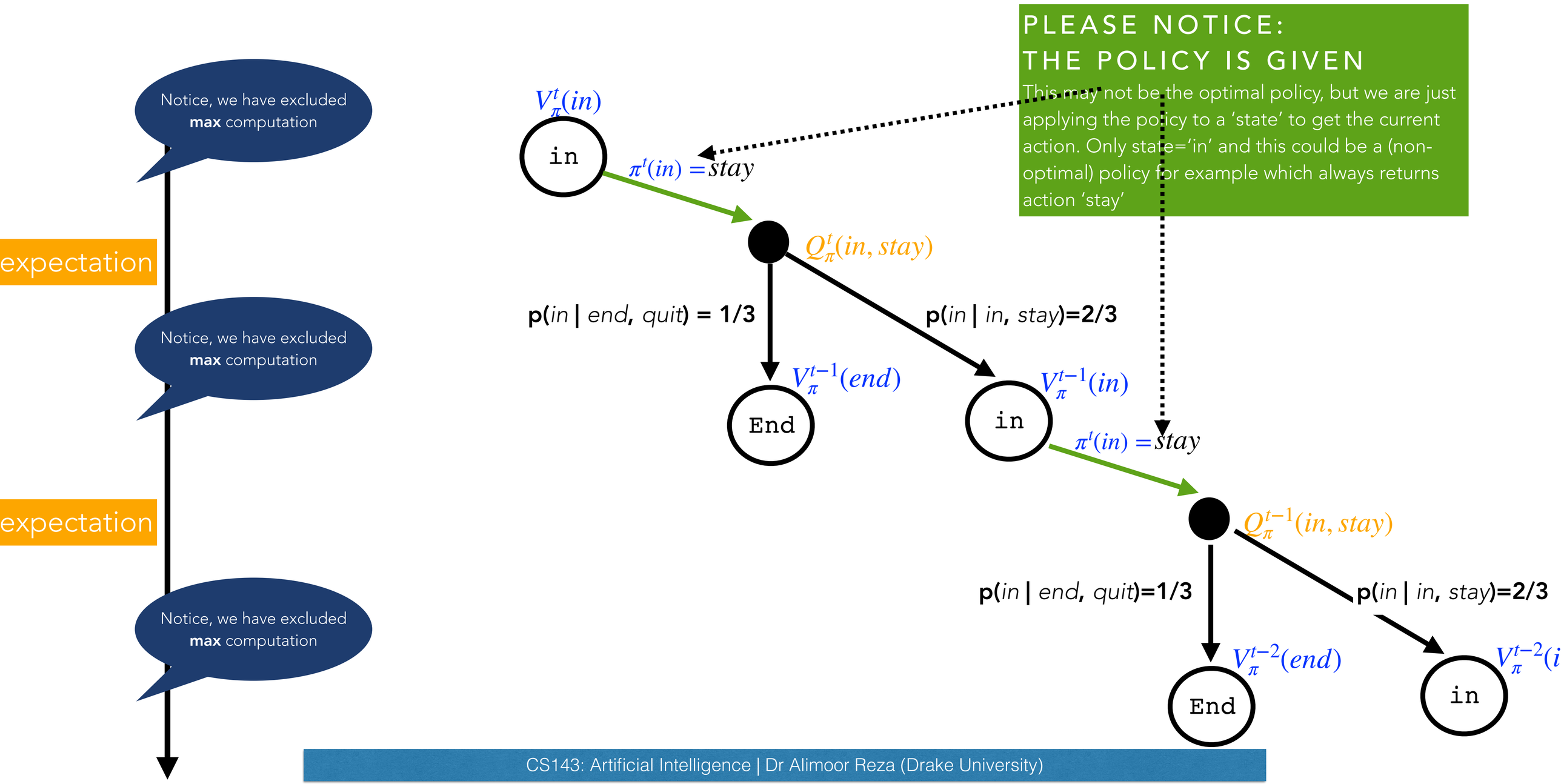
$$= \begin{cases} \frac{2}{3}(4 + \gamma V_\pi^{t-1}(in)) + \frac{1}{3}(4 + \gamma V_\pi^{t-1}(end)) & if \pi(in) = stay \\ \frac{1}{1}(10 + \gamma V_\pi^{t-1}(end)) & if \pi(in) = quit \end{cases}$$

TERMINAL STATE

$$V_\pi(end) = 0$$

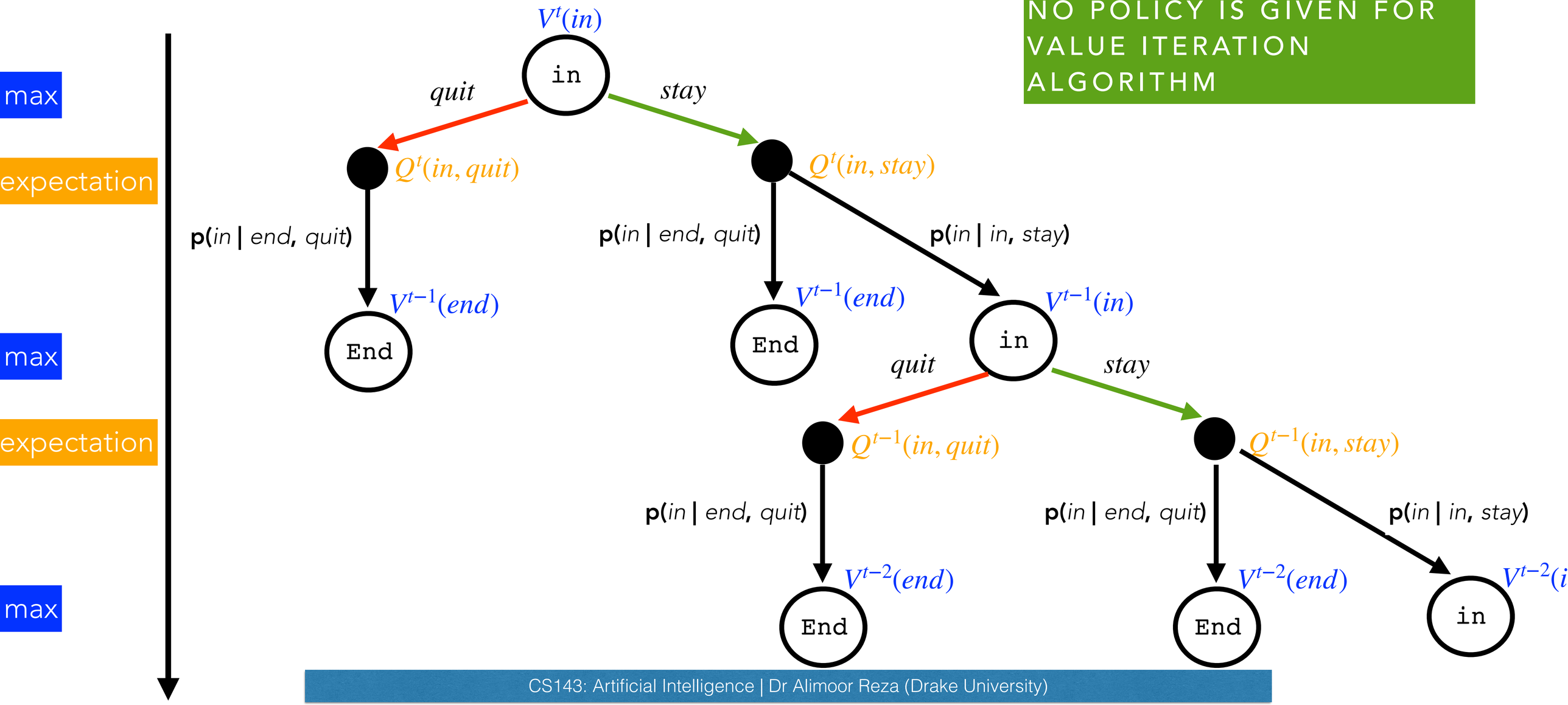
THE RECURRENCE RELATION IS NOW EXPRESSED AS ITERATIVE UPDATES THAT CAN BE COMPUTED INCREMENTALLY

Build an unrolled simpler tree without max computation

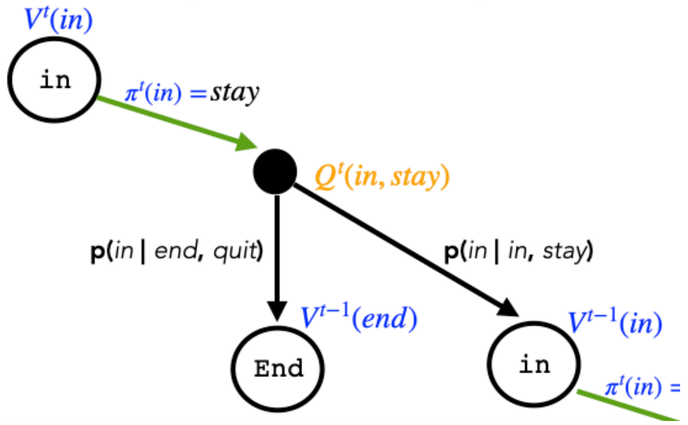


In contrast: value iteration with max computation

PLEASE NOTICE:
NO POLICY IS GIVEN FOR VALUE ITERATION ALGORITHM



Policy evaluation on dice game



Algorithm 1: Policy Evaluation Algorithm

Initialize state-value $v_\pi^0(s)$ for all the states s

//Repeat the iterative procedure for some iterations, let's s

for $t = 1$ **to** T **do**

  for *each state* s **do**

    $v_\pi^t(s) \leftarrow \sum_{s'} p(s'|s, \pi(s))[r(s, \pi(s), s') + \gamma v_\pi^{t-1}(s')]$

TERMINAL STATE

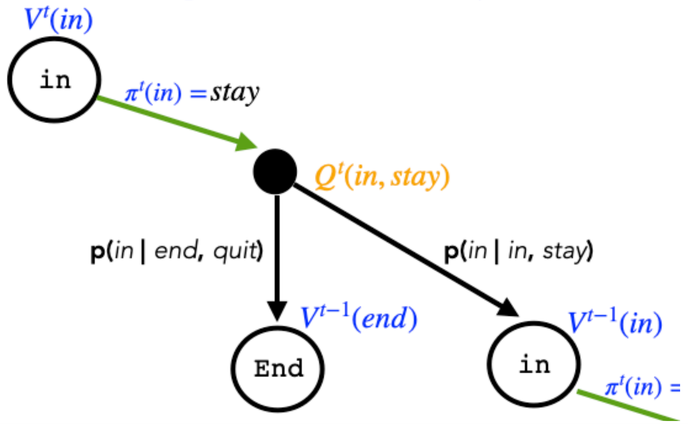
$$v_\pi(end) = 0$$

NON-TERMINAL STATE

$$v_\pi^t(in) = q_\pi^t(in, a)$$
$$= \frac{2}{3}(4 + \gamma v_\pi^{t-1}(in)) + \frac{1}{3}(4 + \gamma v_\pi^{t-1}(end))$$

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
$v_\pi(end)$	0	0	0	0	0	0	0	0	0	0
$v_\pi(in)$	0	-	-	-	-	-	-	-	-	-

Solution



Algorithm 1: Policy Evaluation Algorithm

Initialize state-value $v_\pi^0(s)$ for all the states s

//Repeat the iterative procedure for some iterations, let's s

for $t = 1$ **to** T **do**

  for *each state* s **do**

    $v_\pi^t(s) \leftarrow \sum_{s'} p(s'|s, \pi(s))[r(s, \pi(s), s') + \gamma v_\pi^{t-1}(s')]$

TERMINAL STATE

$$v_\pi(end) = 0$$

NON-TERMINAL STATE

$$v_\pi^t(in) = q_\pi^t(in, a)$$
$$= \frac{2}{3}(4 + \gamma v_\pi^{t-1}(in)) + \frac{1}{3}(4 + \gamma v_\pi^{t-1}(end))$$

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
$v_\pi(end)$	0	0	0	0	0	0	0	0	0	0
$v_\pi(in)$	0	4	6.67	8.44	9.63	10.42	10.95	11.30	11.53	11.69

Contrast: value iteration vs. policy evaluation

Value Iteration Algorithm	*Policy Evaluation Algorithm*
No policy is given	a non-optimal policy is given
Has max computations for V(s) nodes	No max computations for V(s) nodes
Max operation on several action branches for V(s) nodes	Just one action branch for V(s) nodes
Returns: $(MDP) \rightarrow V_{opt}$ and π_{opt}	Returns: $(MDP, \pi) \rightarrow V_{\pi}$

Explore the MDP and do two things:
i) calculate state-value of each state
ii) find the best policy

A (non-optimal) policy is given to you, you don't need to find it

Contrast: value iteration vs. policy evaluation

Value Iteration Algorithm	*Policy Evaluation Algorithm*

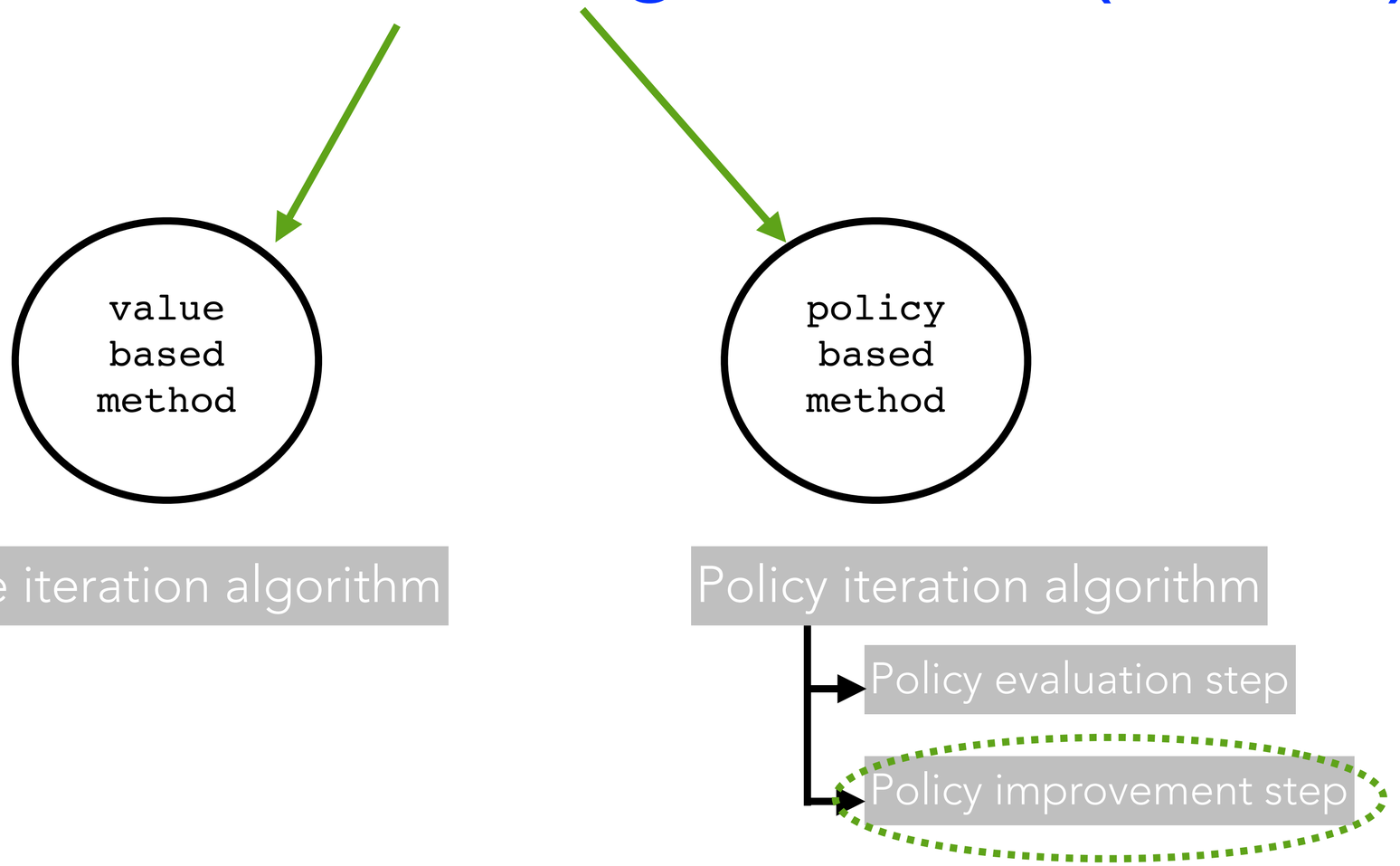
Returns: $(MDP) \rightarrow V_{opt}$ and π_{opt}

Returns: $(MDP, \pi) \rightarrow V_{\pi}$ and ?

Explore the MDP and do two things:
i) calculate state-value of each state
ii) find the best policy

A (non-optimal) policy is given to you, you don't need to find it

Reinforcement learning solvers (so far)



Policy Iteration Algorithm

Iterate `j=1, 2, 3, …. T` steps

- **Step 1:** For an MDP and **its given policy**, **apply** policy evaluation algorithm repeatedly until the the state-value and action-values converge for each state

- **Step 2: update the policy** using one step look-ahead

Policy Iteration Algorithm

Iterate `j=1, 2, 3, …. T` steps

- **Step 1:** For an MDP and **its given policy**, **apply** policy evaluation algorithm repeatedly until the the state-value and action-values converge for each state

$$V_{\pi_j}^{t}(s) \quad = \sum_{s'} p(s' \mid in, \pi_j(s))[R(s, \pi_j(s), s') + \gamma V_{\pi_j}^{t-1}(s')]$$

- **Step 2: update the policy** using one step look-ahead

$$\pi_{j+1}(s) \quad = \arg max_a \sum_{s'} p(s' \mid s, a)[R(s, a, s') + \gamma V_{\pi_j}(s')]$$