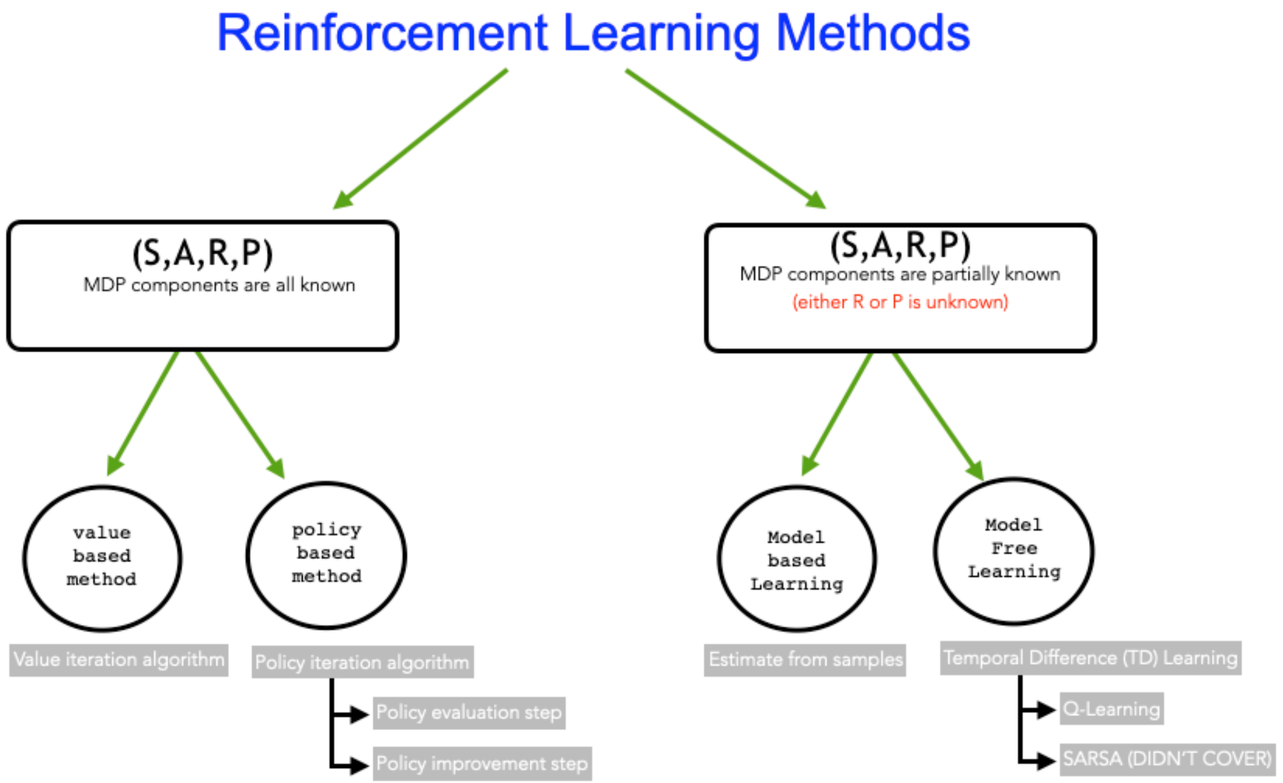
CS143: Artificial Intelligence

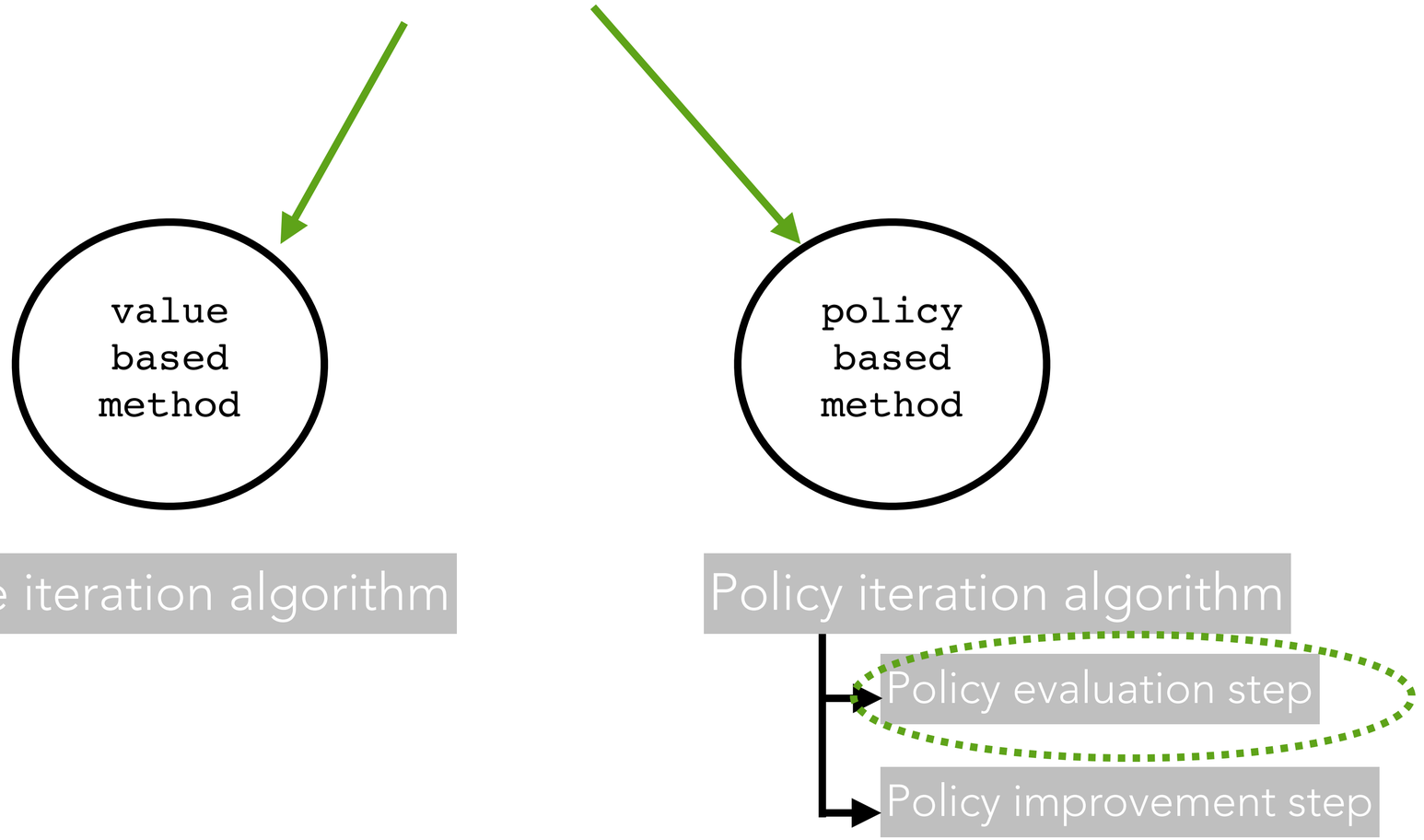


Temporal Difference Learning

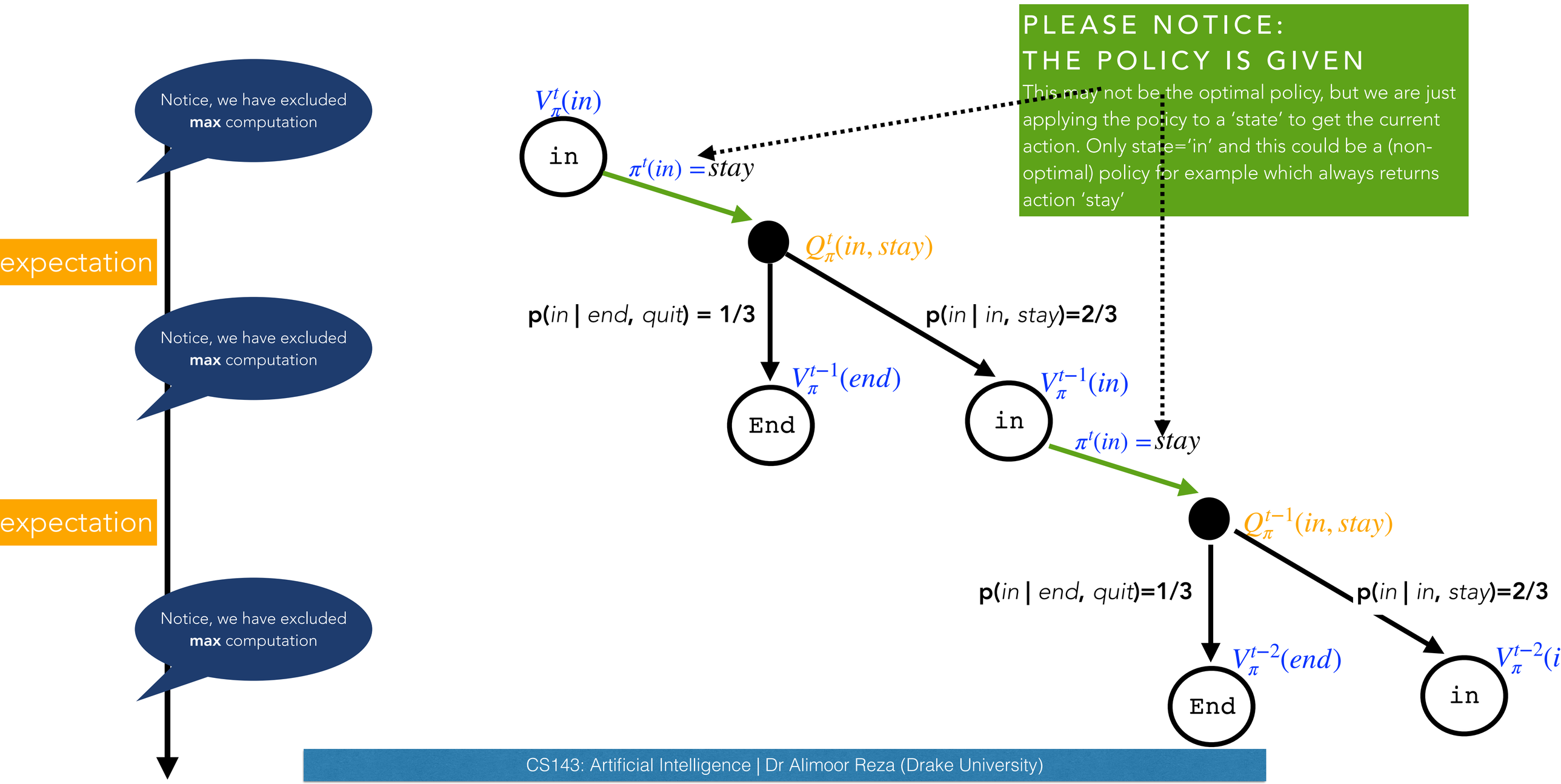
Q-Learning

Drake UNIVERSITY

Review: Reinforcement Learning Solvers (so far)

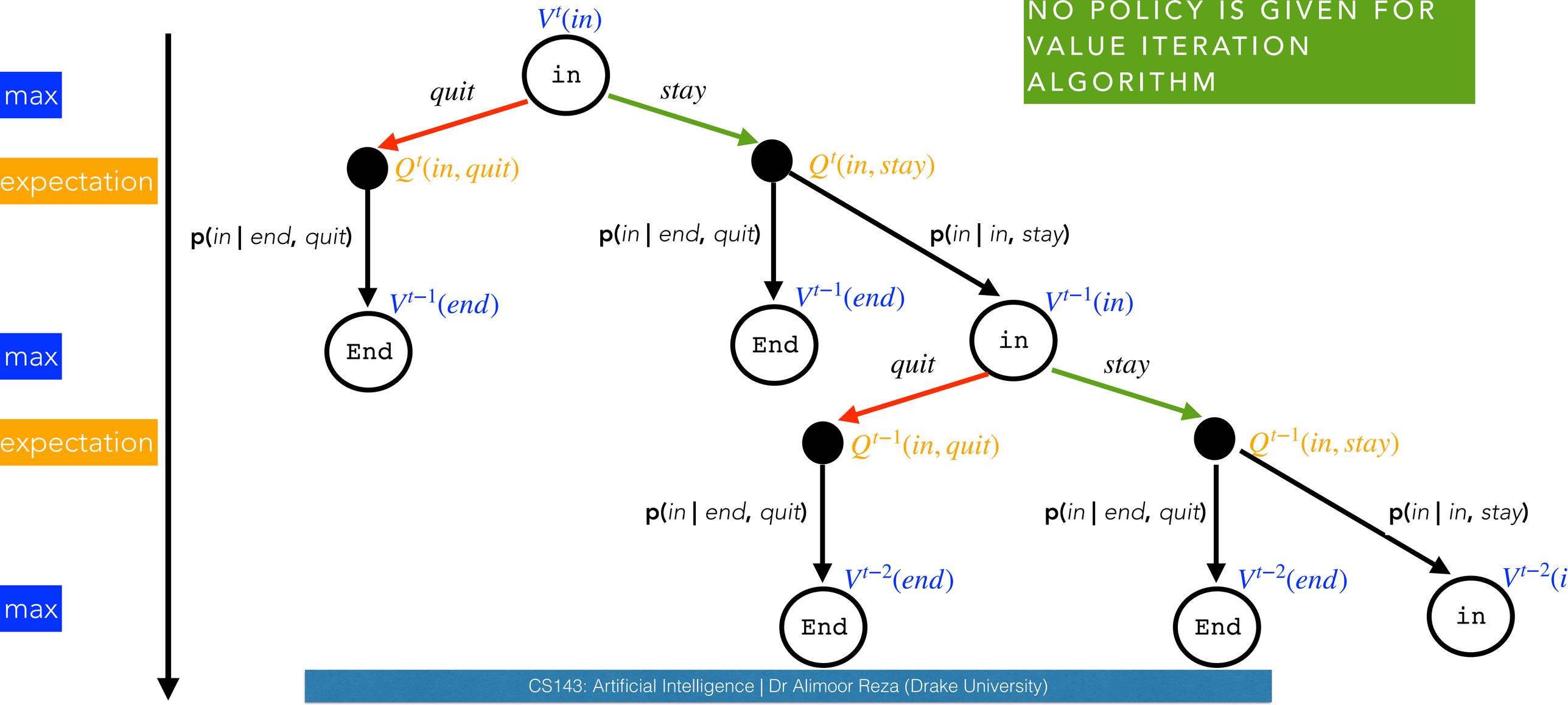


Build an unrolled simpler tree without max computation

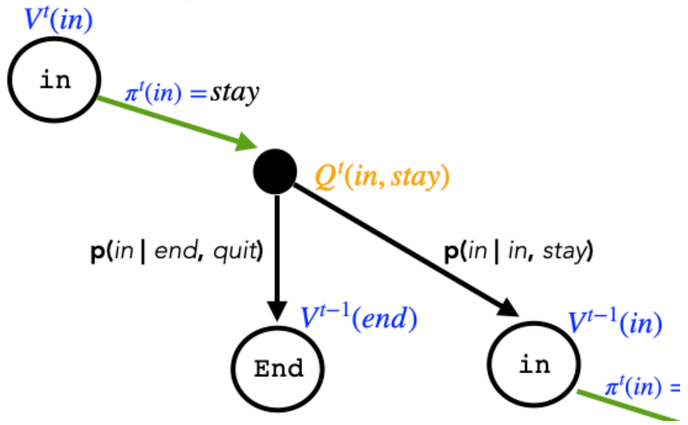


In contrast: value iteration with max computation

PLEASE NOTICE:
NO POLICY IS GIVEN FOR VALUE ITERATION ALGORITHM



Review: Solution: 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	4	6.67	8.44	9.63	10.42	10.95	11.30	11.53	11.69

Review: Contrast: value iteration vs. policy evaluation

Value Iteration Algorithm	*Policy Evaluation Algorithm*
No policy is given	a policy (optimal/non-optimal) 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

Review: Contrast: value iteration vs. policy evaluation

Value Iteration Algorithm	*Policy Evaluation Algorithm*

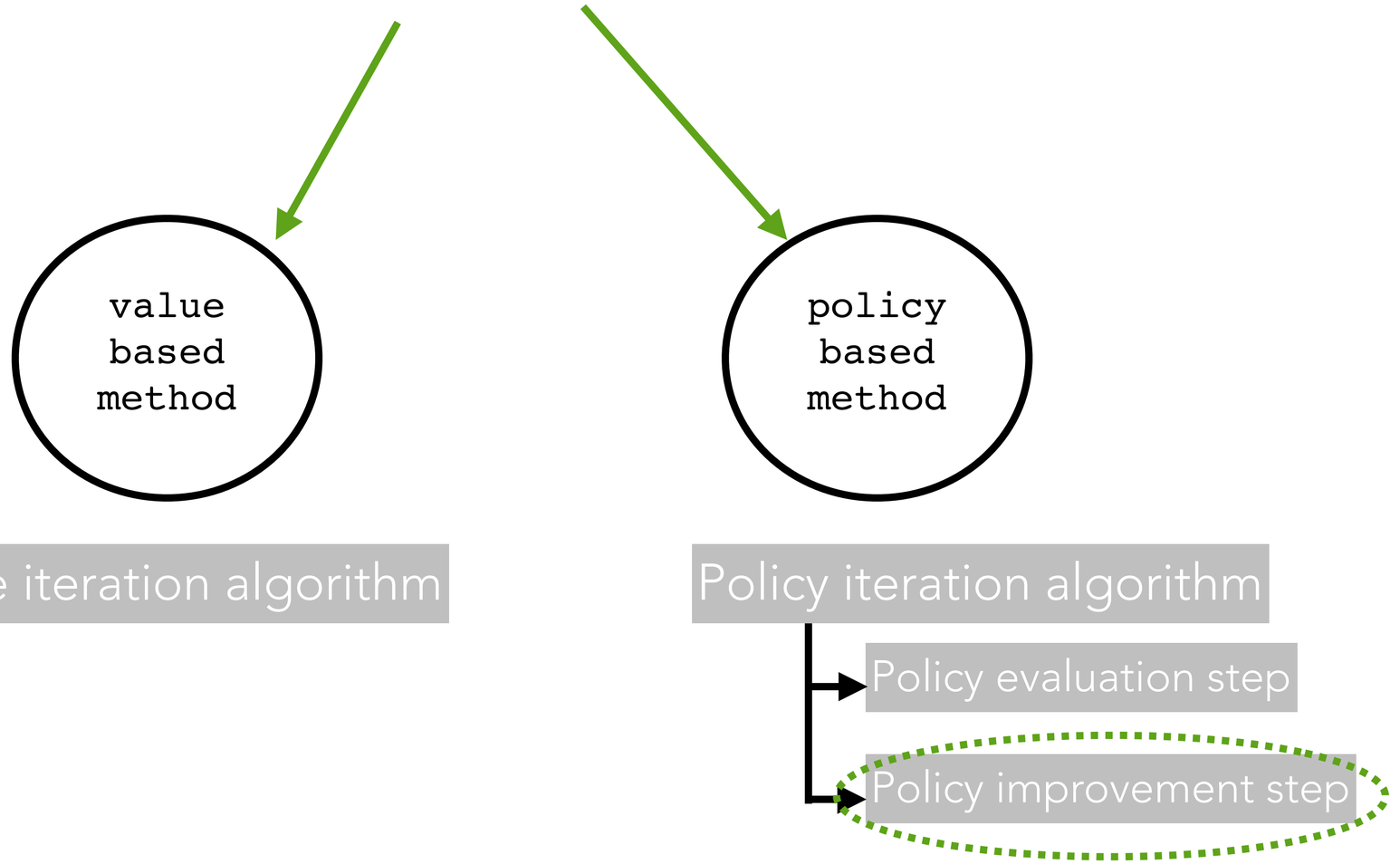
Returns: $(MDP) \rightarrow V_{opt}$ and π_{opt}

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

Review: Reinforcement learning solvers (so far)



Review: Policy Iteration Algorithm

Iterate `j=1, 2, 3, …. T` steps

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

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

- **<u>Step 2:</u> update the policy** using one step look-ahead

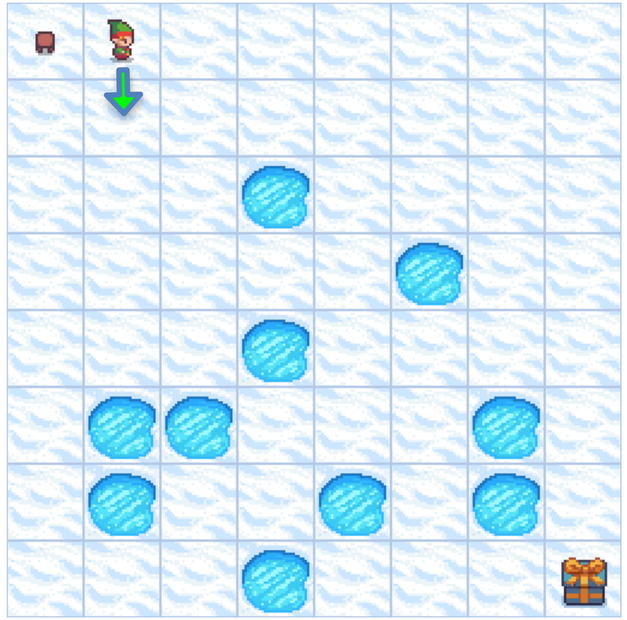
$$\pi_{j+1}(s) = \arg\max_{a} \sum_{s'} p(s' \mid s, a)[R(s, a, s') + \gamma V_{\pi_j}(s')]$$

New Material

Let's Revisit MDP

- Components of an MDP is a tuple (S,A,R,P):
  - States: $s \in S$
  - Actions: $a \in A$
  - Reward function: $R(s, a, s')$
  - Transition function: $P(s' \mid a, s)$

- The transition function encodes the movements in the Lake
  - In Frozen Lake, model or dynamics is known
  - $R(s, a, s')$ is read: agent receives reward **R** as it transitions from state **s** to **s'** by taking action **a**
  - $P(s' \mid a, s)$ is read: agent transitions from state **s** to **s'** by taking action **a**



In frozen lake instance, all the components of (S,A,R,P) were known to us. Hence we could apply Value iteration or Policy iteration Algorithms

What happens when we know MDP components partially?

- Components of an MDP is a tuple (S,A,R,P):
  - Known components are:
    - States: $s \in S$
    - Actions: $a \in A$
  - Either of the two components are unknown:
    - Reward function: unknown $R(s, a, s')$ *or*
    - Transition function: unknown $P(s' \mid a, s)$

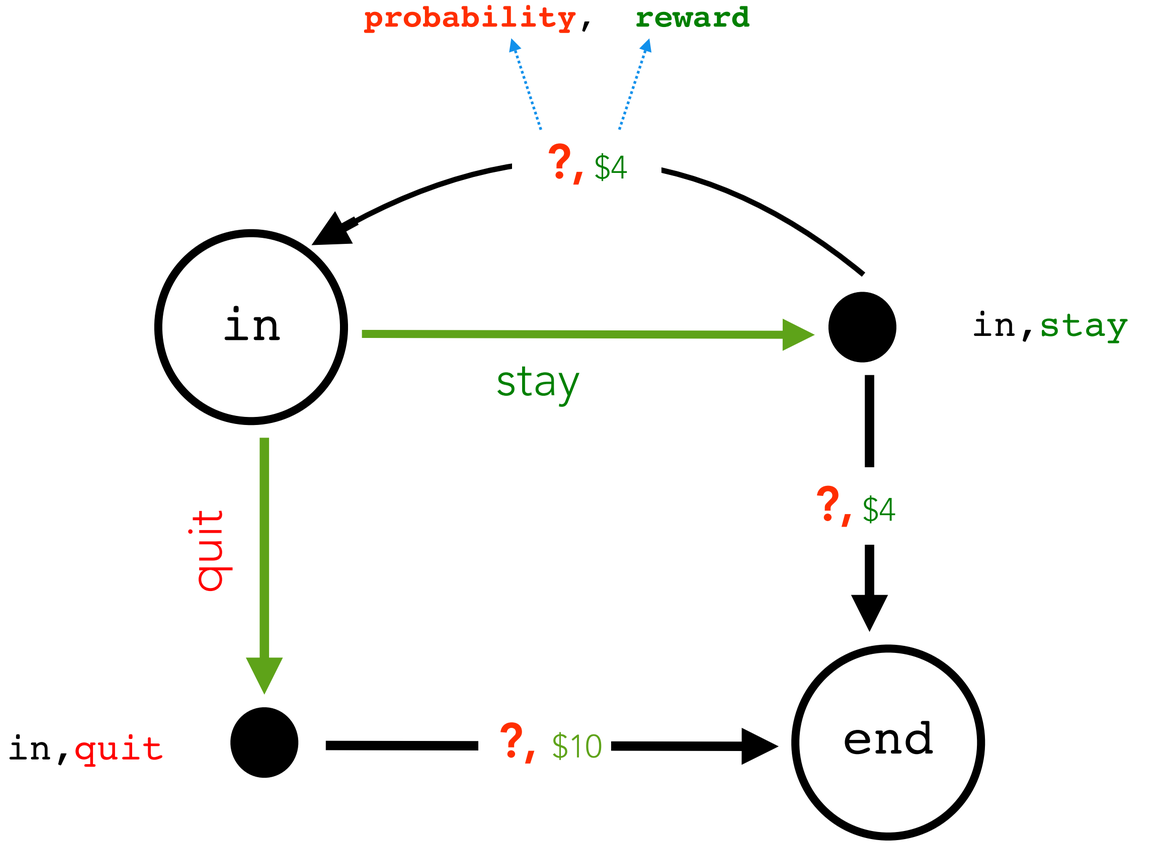


Figure: For our simple dice game, we know the state nodes and action nodes but we don't know the probability transitions.

How this will affect 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**

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

Probability transition function $P(s' \mid a, s)$ **is missing. We won't be able to compute this step.**

How this will affect policy evaluation algorithm?

INITIALIZE STATE-VALUES WITH SOME ROUGH ESTIMATES

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**

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

Probability transition function $P(s'|a, \pi(s))$ **is missing. We won't be able to compute this step.**

Implication on Reinforcement Learning Solvers (so far)

value based method

policy based method

Value iteration algorithm

Policy iteration algorithm

- Policy evaluation step
- Policy improvement step

Won't be able to use **Value iteration Algorithm**

Won't be able to use **Policy Iteration Algorithm**

What could we do?

- Components of an MDP is a tuple (S,A,R,P):
  - Either of the two components are unknown:
    - Reward function: unknown $R(s, a, s')$ *or*
    - Transition function: unknown $P(s'|a, s)$

- Let us try to estimate R(s,a,s') and (P(s'|s,a) by interacting with the environment. This interactive process will generate sample episodes.

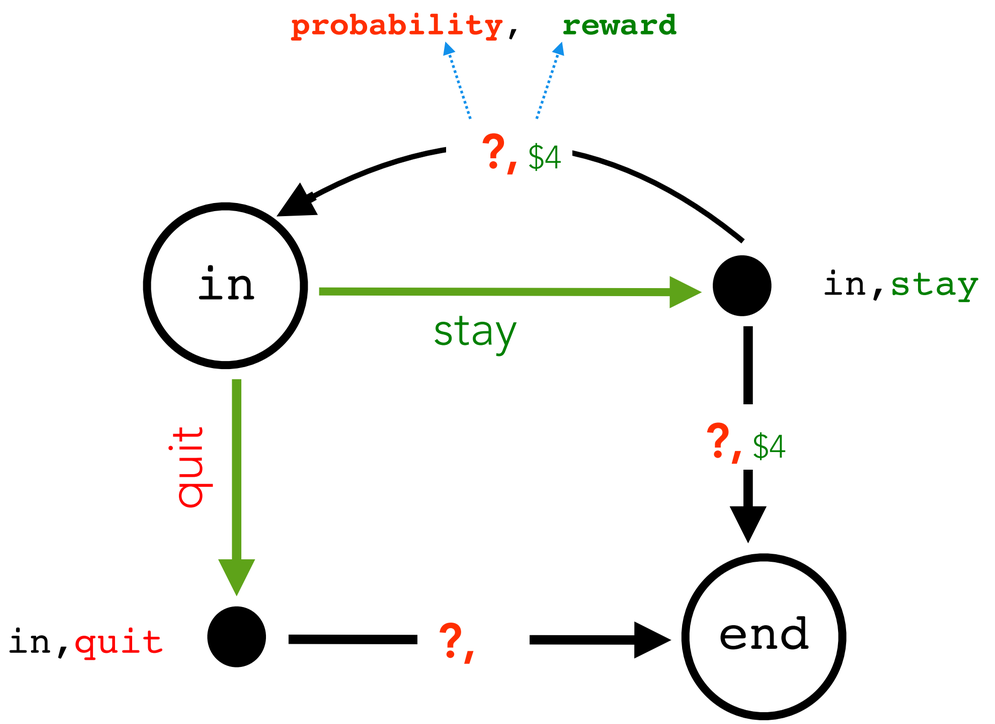


Figure: For our simple dice game, we know the state nodes and action nodes but we don't know the probability transitions.

Learning/estimating missing MDP components

Episode#1

in, stay, in, $4

in, stay, in, $4

in, stay, end, $4

Episode#2

in, stay, end, $4

Episode#3

in, stay, in, $4

in, stay, end, $4

Episode#4

in, stay, in, $4

in, stay, in, $4

in, stay, in, $4

in, stay, end, $4

Episode#5

in, stay, in, $4

in, stay, end, $4

Episode#6

in, quit, end, $10

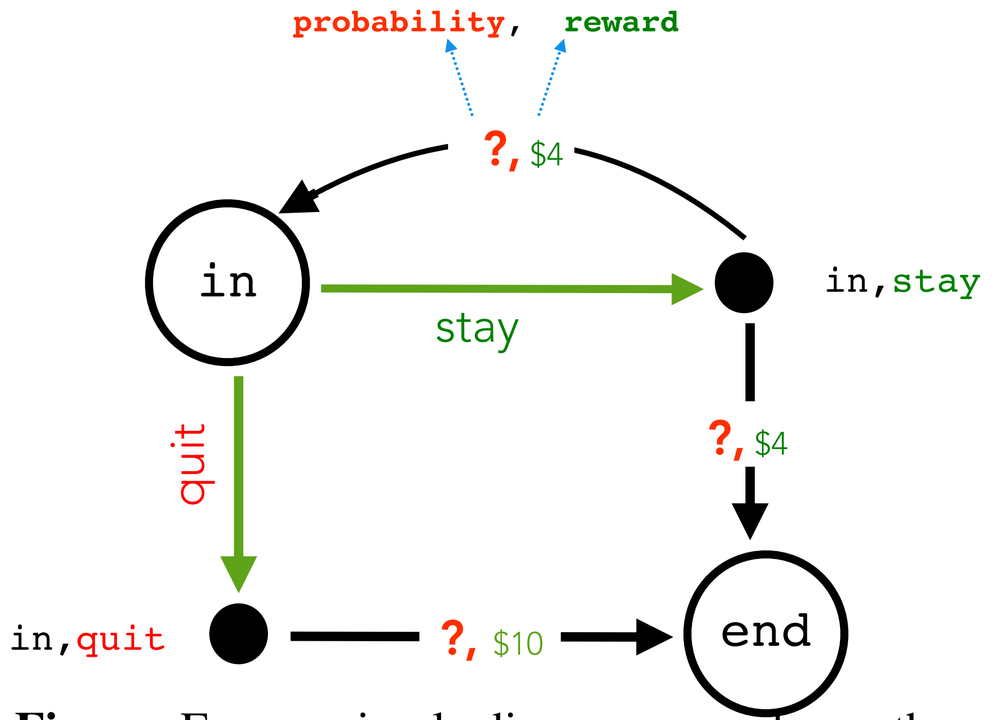


Figure: For our simple dice game, we know the state nodes and action nodes but we don't know the probability transitions. Let's try to estimate the transition probabilities by sampling tuples (s,a,s',r) from the environment interaction.

Estimate Transition Probabilities

Episode#1

in, stay, in, $4

in, stay, in, $4

in, stay, end, $4

Episode#2

in, stay, end, $4

Episode#3

in, stay, in, $4

in, stay, end, $4

Episode#4

in, stay, in, $4

in, stay, in, $4

in, stay, in, $4

in, stay, end, $4

Episode#5

in, stay, in, $4

in, stay, end, $4

Episode#6

in, quit, end, $10

$$P(\texttt{in} \mid \text{stay}, \texttt{in}) = \frac{7}{12} = 0.583$$

$$P(\texttt{end} \mid \text{stay}, \texttt{in}) = \frac{5}{12} = 0.417$$

Estimate Transition Probabilities

Episode#1

in, stay, in, $4

in, stay, in, $4

in, stay, end, $4

Episode#2

in, stay, end, $4

Episode#3

in, stay, in, $4

in, stay, end, $4

Episode#4

in, stay, in, $4

in, stay, in, $4

in, stay, in, $4

in, stay, end, $4

Episode#5

in, stay, in, $4

in, stay, end, $4

Episode#6

in, quit, end, $10

$$P(\text{end} \mid \text{quit, in}) = \frac{1}{1} = 1.0$$

Can We Use these Reinforcement Learning Algorithms Now To Find Optimal Policy?

value based method

policy based method

Value iteration algorithm

Policy iteration algorithm

- Policy evaluation step
- Policy improvement step

We should be able to use **Value iteration Algorithm** using our estimated probabilities

We should be able to use **Policy iteration Algorithm** using our estimated probabilities

Question: How can we find the optimal policy without estimating model dynamics?

Answer: **model-free** algorithms (eg, Temporal Difference Learning)

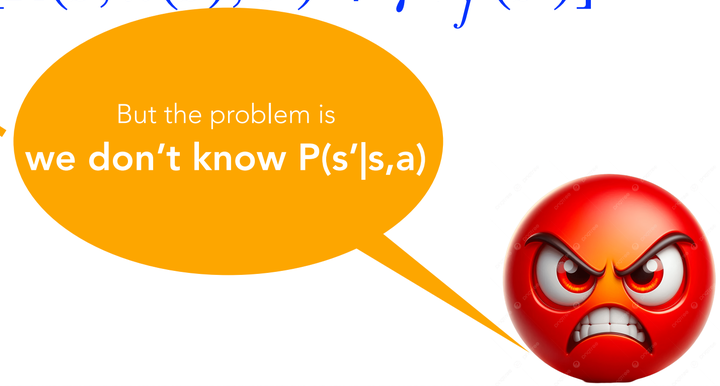
New Idea: Maintain a Running Average for Value Function

- Components of an MDP is a tuple (S,A,R,P):
  - If all components are known:
    - Reward function: known $R(s, a, s')$
    - Transition function: known $P(s' \mid a, s)$

- Then we could compute the average value using our Bellman equation:

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

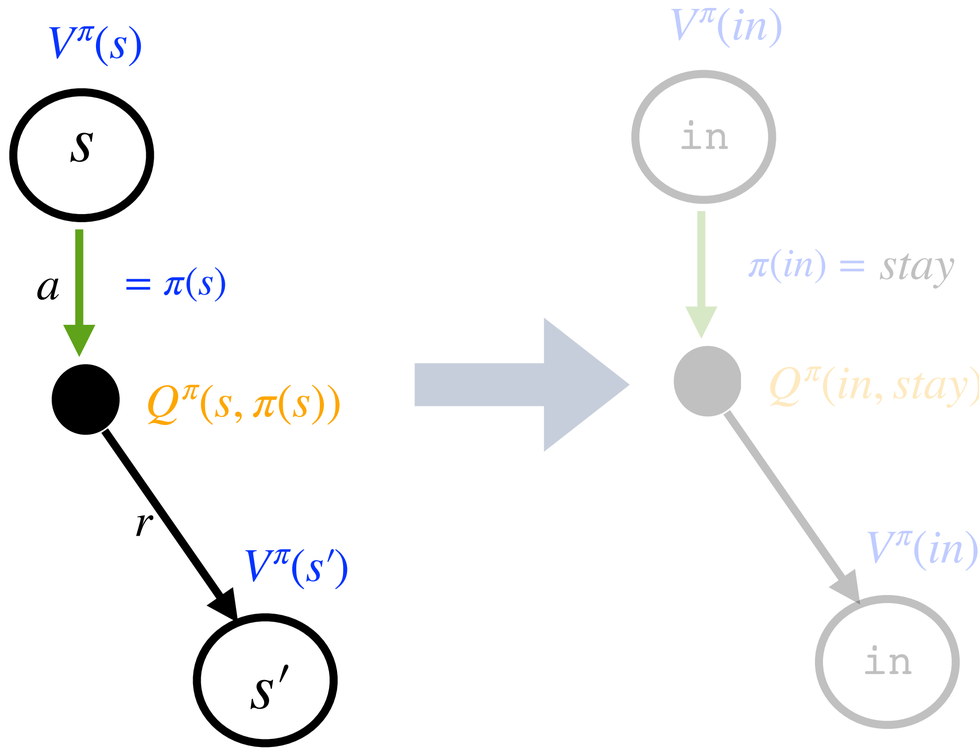
$$V_{j+1}^\pi(s) \leftarrow \sum_{s'} p(s' \mid s, \pi(s))[R(s, \pi(s), s') + \gamma V_j^\pi(s')]$$



Solution: Generate Samples

- Components of an MDP is a tuple (S,A,R,P):
  - Either of the two components are unknown:
    - Reward function: unknown $R(s, a, s')$ *or*
    - Transition function: unknown $P(s' \mid a, s)$

- Let us interact with the environment by acting upon it. It will generate samples of the form
  - (s, a, s', r)
  - This sample is shown in the image on the right. Notice that there is only one transition from the Q(,) node because it represents a single sample. We do not have probabilities for transitions into multiple branches.



(s, a, s', r)

Example: Generate Samples

Episode#1

sample$_1$ `in`, stay, `in`, $4

sample$_2$ `in`, stay, `in`, $4

sample$_3$ `in`, stay, `end`, $4

Episode#2

sample$_4$ `in`, stay, `end`, $4

Episode#3

sample$_5$ `in`, stay, `in`, $4

sample$_6$ `in`, stay, `end`, $4

Episode#4

sample$_7$ `in`, stay, `in`, $4

sample$_8$ `in`, stay, `in`, $4

sample$_9$ `in`, stay, `in`, $4

sample$_{10}$ `in`, stay, `end`, $4

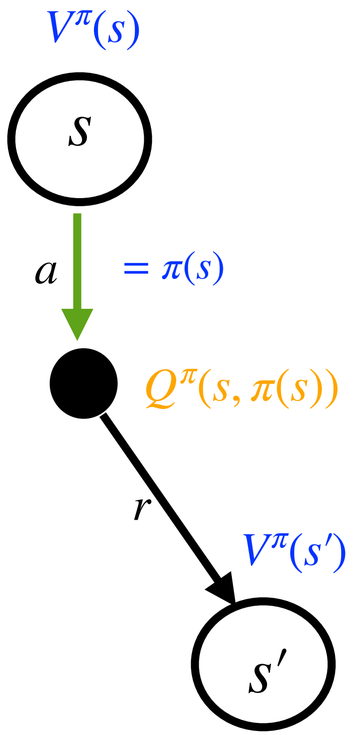
Episode#5

sample$_{11}$ `in`, stay, `in`, $4

sample$_{12}$ `in`, stay, `end`, $4

Episode#6

sample$_{13}$ `in`, quit, `end`, $10



(s, a, s', r)

But How do we utilize the generated samples for calculating value function?

Answer: Apply Monte Carlo Approximation

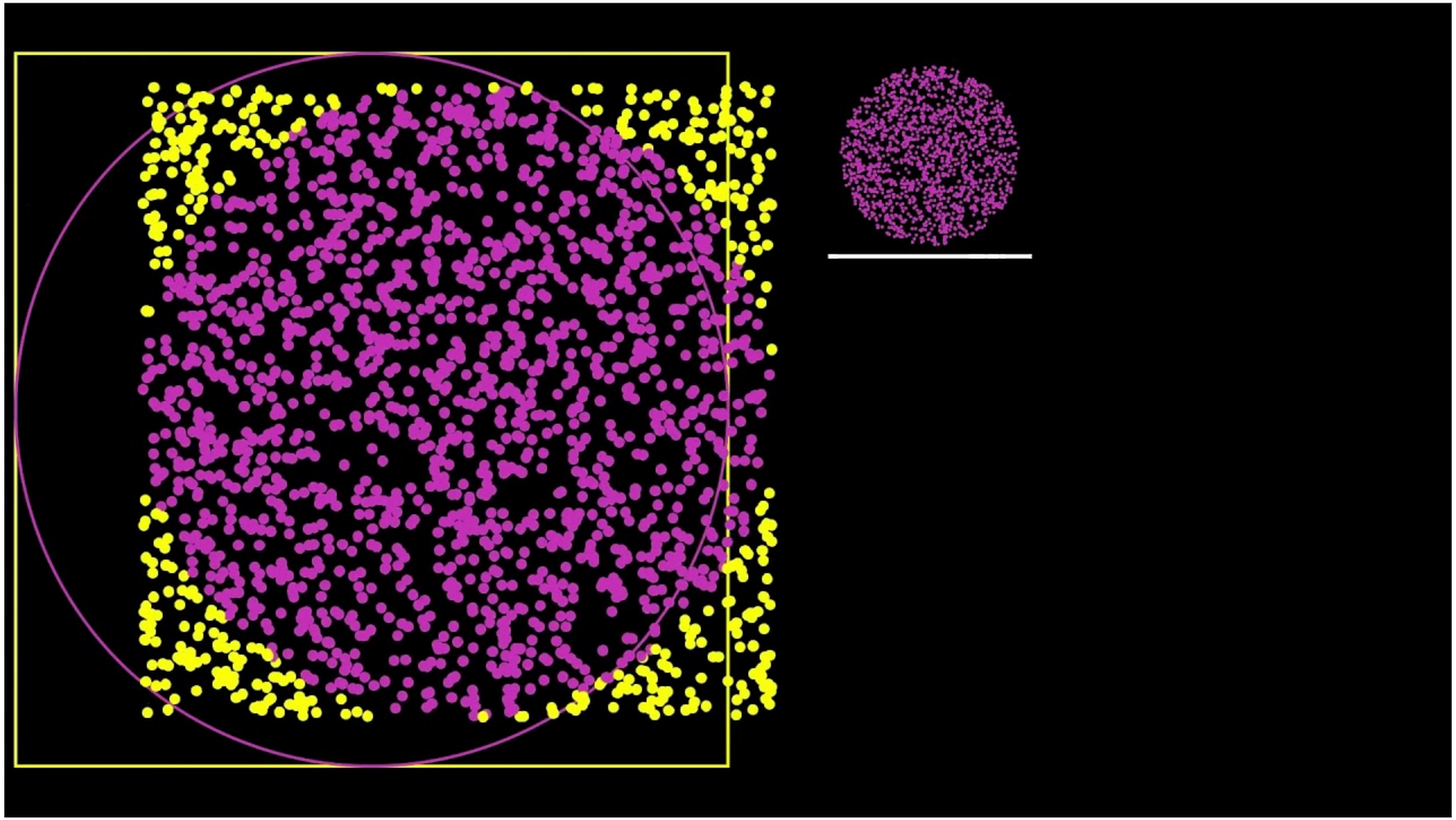
Monte Carlo Approximation

- Sometimes it is difficult to compute a function *g(y)*. Because *x and y* are related via:
  - *y = f(x)*
  - *But there is no analytical formula for computing g(y)*

- One simple but powerful alternative is to draw a large number of samples from the function *x* and use these samples to an approximate value of *g(y)*

- Let's see how monte carlo approximation can help us for calculating the area of an difficult shape *g(y)*

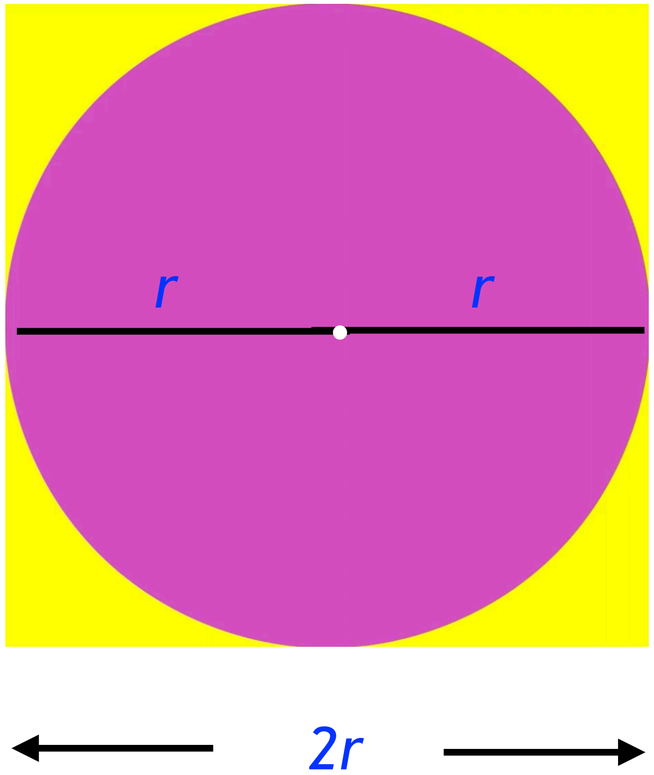
Monte Carlo Approximation



Reference: https://www.youtube.com/watch?v=8276ZswRw7M

Monte Carlo Approximation

- Suppose a circle of radius r is inside a rectangle of side $2r$.
- Randomly throw points into the rectangle.
  - Area(rectangle): $(2r)^2=4r^2$
  - Inside points = count the points landing inside circle
  - Outside points = Count the points landing outside the circle
  - Total points = Inside points + Outside points

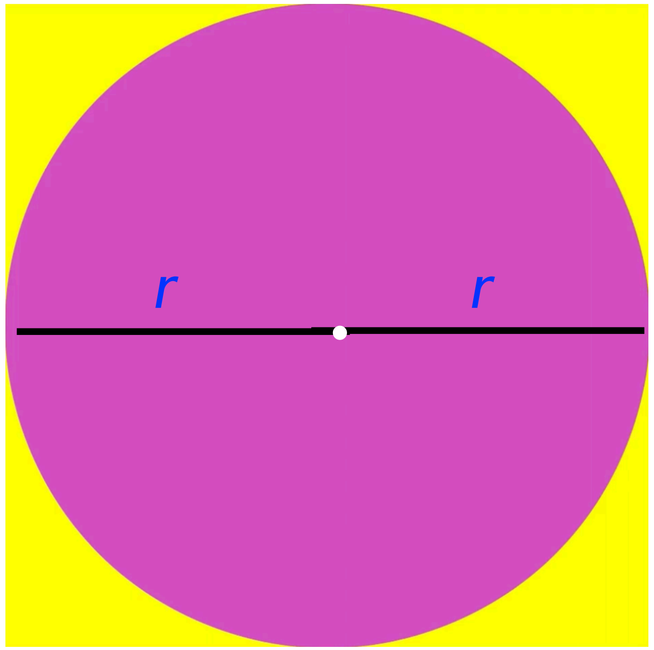


Area(circle): ????

Monte Carlo Approximation

$$\text{Area(circle)} \approx \frac{\text{Inside points}}{\text{Total points}} * \text{Area(rectangle)}$$

$$\approx \frac{\text{Inside points}}{\text{Total points}} * 4r^2$$

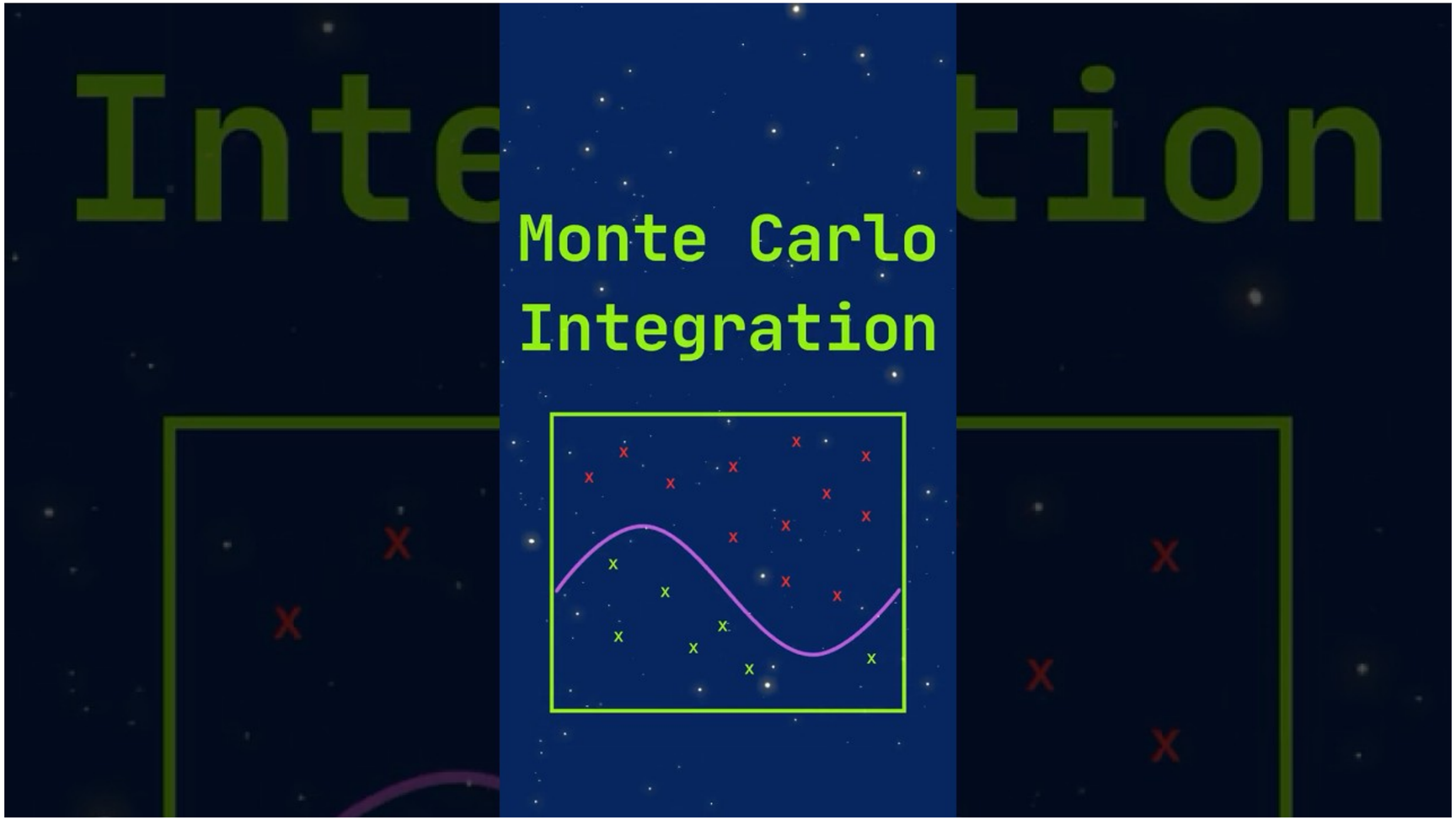


- If we randomly throw 10000 points into the rectangle and 7850 fall inside the circle with radius *r=1*

$$\text{Area(circle)} \approx \frac{7850}{10000} * 4$$

$$\approx 3.14$$

Monte Carlo Approximation

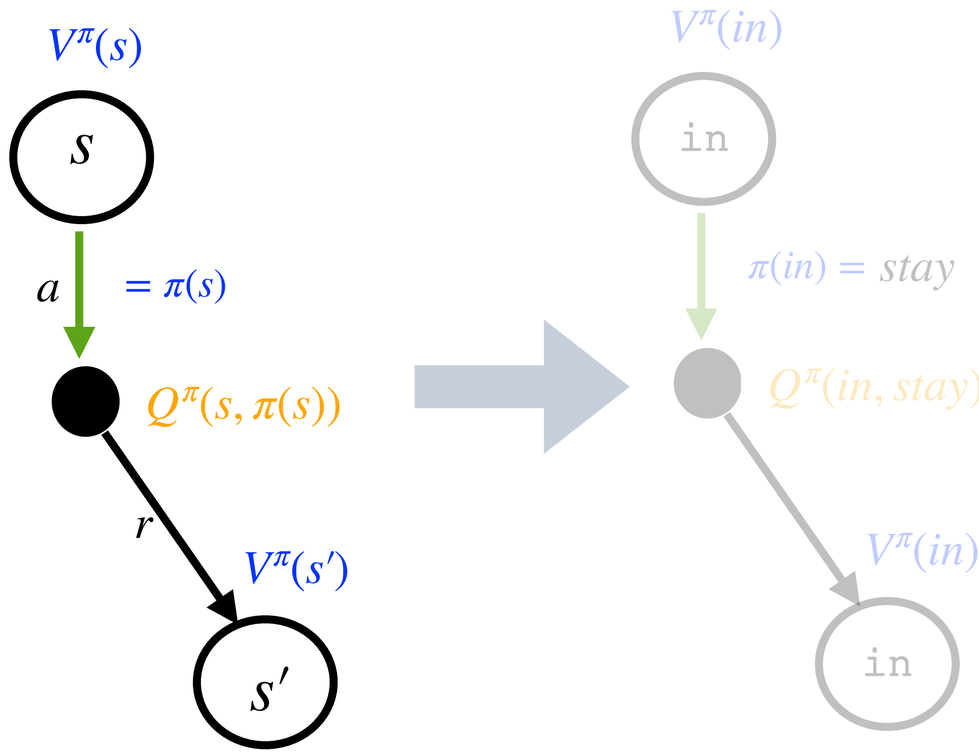


Reference: https://www.youtube.com/watch?v=wJ7i7zK-fU8

Solution: Generate Samples

- Components of an MDP is a tuple (S,A,R,P):
  - Either of the two components are unknown:
    - Reward function: unknown $R(s, a, s')$ *or*
    - Transition function: unknown $P(s' \mid a, s)$

- Let us interact with the environment by acting upon it. It will generate samples of the form
  - (s, a, s', r)
  - This sample is shown in the image on the right. Notice that there is only one transition from the Q(,) node because it represents a single sample. We do not have probabilities for transitions into multiple branches.



(s, a, s', r)

Example: Average for Value Function from Samples

- Calculate the average of the value function from the samples as follows:

Episode#1

$sample_1$ in, stay, in, $4

$sample_2$ in, stay, in, $4

$sample_3$ in, stay, end, $4

Episode#2

$sample_4$ in, stay, end, $4

Episode#3

$sample_5$ in, stay, in, $4

$sample_6$ in, stay, end, $4

Episode#4

$sample_7$ in, stay, in, $4

$sample_8$ in, stay, in, $4

$sample_9$ in, stay, in, $4

$sample_{10}$ in, stay, end, $4

Episode#5

$sample_{11}$ in, stay, in, $4

$sample_{12}$ in, stay, end, $4

Episode#6

$sample_{13}$ in, quit, end, $10

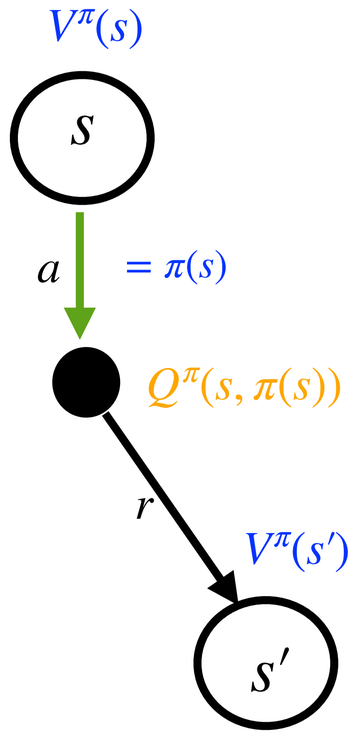
$$sample_1 = R(in, stay, in) + \gamma V_j^\pi(in)$$
$$sample_2 = R(in, stay, in) + \gamma V_j^\pi(in)$$
$$sample_3 = R(in, stay, end) + \gamma V_j^\pi(end)$$
$$sample_4 = R(in, stay, end) + \gamma V_j^\pi(end)$$
$$sample_5 = R(in, stay, in) + \gamma V_j^\pi(in)$$
$$sample_6 = R(in, stay, end) + \gamma V_j^\pi(end)$$
$$sample_7 = R(in, stay, in) + \gamma V_j^\pi(in)$$
$$sample_8 = R(in, stay, in) + \gamma V_j^\pi(in)$$
$$sample_9 = R(in, stay, in) + \gamma V_j^\pi(in)$$
$$sample_{10} = R(in, stay, end) + \gamma V_j^\pi(end)$$
$$sample_{11} = R(in, stay, in) + \gamma V_j^\pi(in)$$
$$sample_{12} = R(in, stay, end) + \gamma V_j^\pi(end)$$
$$sample_{13} = R(in, quit, end) + \gamma V_j^\pi(end)$$

$$V_{j+1}^\pi(in) \leftarrow \frac{1}{13} \sum_i sample_i$$

More Formally: Temporal Difference (TD) Learning

- Components of an MDP is a tuple (S,A,R,P):
  - Either of the two components are unknown:
    - Reward function: unknown $R(s, a, s')$ *or*
    - Transition function: unknown $P(s' \mid a, s)$

- Let us interact with the environment by acting upon it. It will generate samples of the form
  - (s, a, s', r)
  - This sample is shown in the image on the right. Notice that there is only one transition from the Q(,) node because it represents a single sample. We do not have probabilities for transitions into multiple branches.

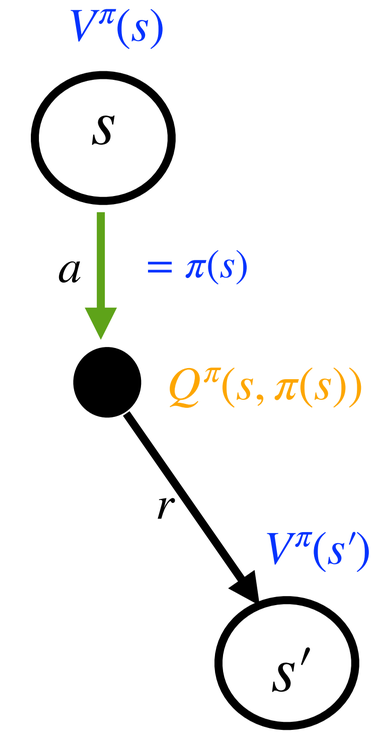


$$sample_i = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

Temporal Difference (TD) Learning

- Let us interact with the environment by acting upon it. It will generate samples of the form
  - (s, a, s', r)
  - We will use this sample to update value function $V^{\pi}(s)$
  - Like Policy Evaluation Algorithm, we will given a fixed policy, we will update the values from the samples until convergence.
  - Rather than directly computing average (as shown in slides before), we will maintain a running average of value function $V^{\pi}(s)$ as follows:

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha \ sample_i$$



$sample_i = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

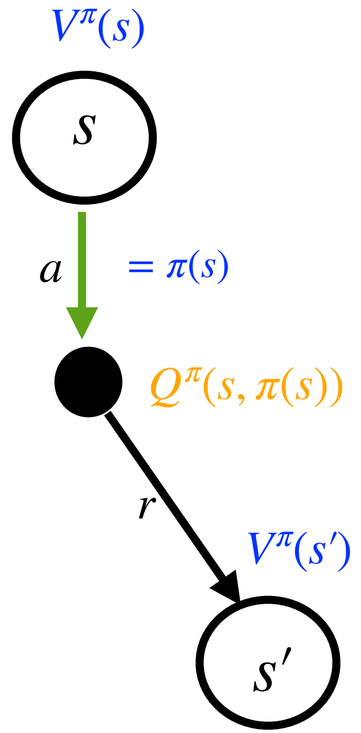
α = *learning rate which can vary between 0 to 1.0*

Temporal Difference (TD) Learning

- Let us interact with the environment by acting upon it. It will generate samples of the form

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha \; sample_i$$

$$\boxed{V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha[R(s,\pi(s),s') + \gamma V^{\pi}(s')]}$$



$sample_i = R(s,\pi(s),s') + \gamma V^{\pi}(s')$

α = *learning rate which can vary between 0 to 1.0*

Question: How can we find the optimal policy without estimating model dynamics?

Answer: **model-free** algorithms (eg, Q-Learning)

Recall: Value Iteration Algorithm

- Turning recursive Bellman equations into update equations:

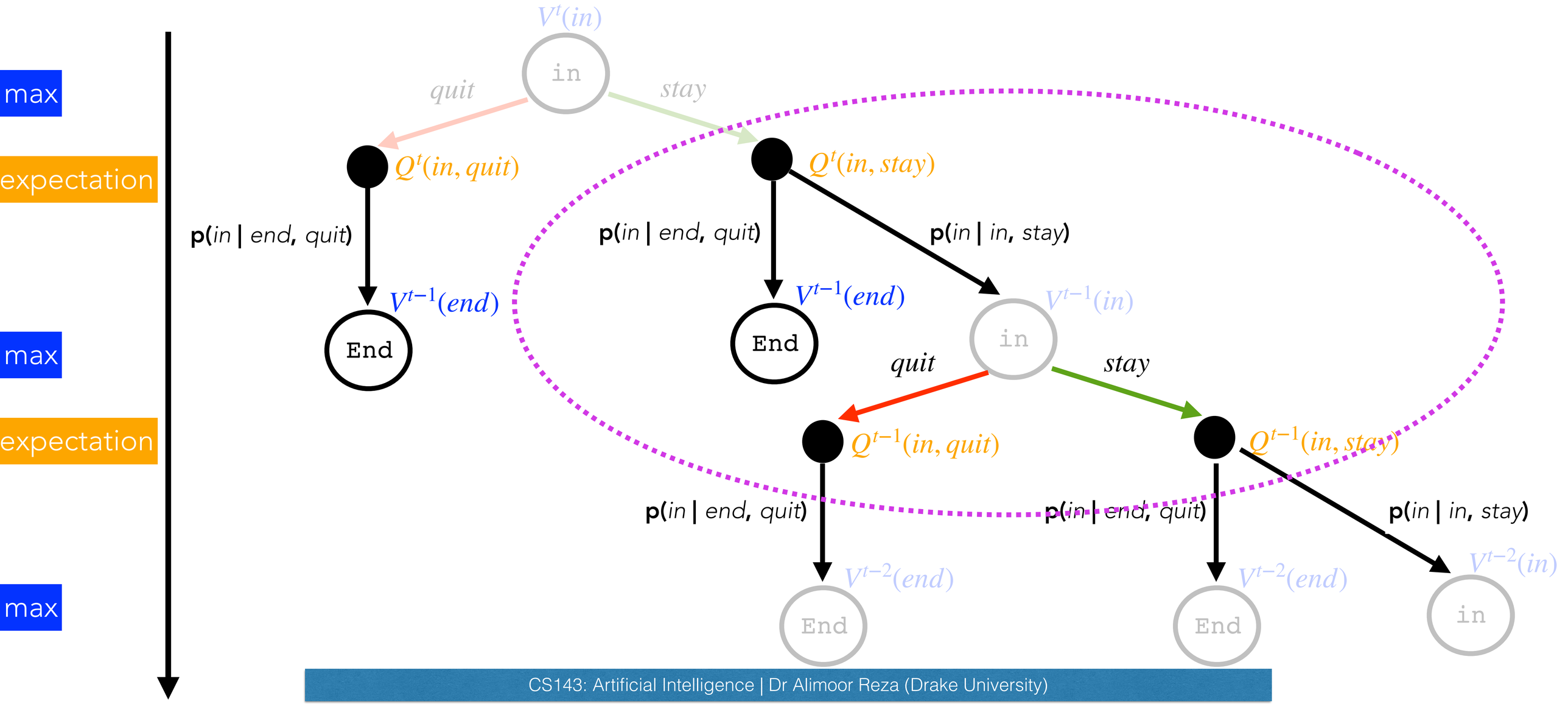
$$V^0(s) = 0$$

$$V^t(s) \leftarrow \max_{a \in actions(s)} \sum_{s'} p(s' \mid s, a)[r(s, a, s') + \gamma V^{t-1}(s')]$$

q-value in time step ***t*** can be computed via retrieving values from its previous iteration ***(t-1)***

- Instead of calculating value functions, let's **directly calculate q-value function**

Infer the relationship between Q-nodes



Recall: Value Iteration Algorithm

- Instead of calculating value functions, let's directly calculate q-value function

$$Q^0(s,a) = 0$$

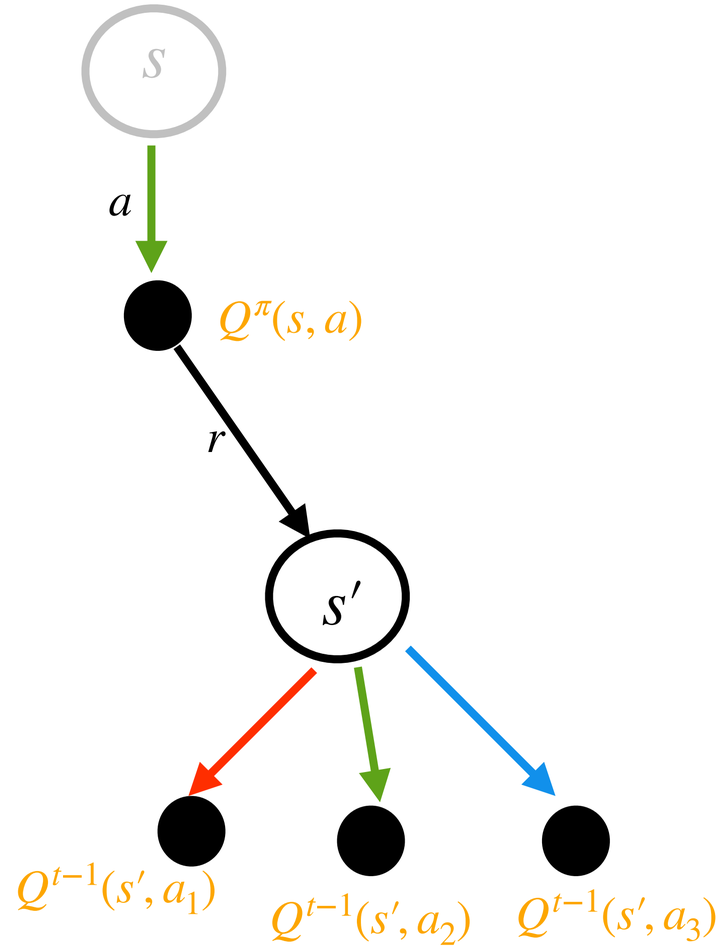
$$Q^t(s,a) \leftarrow \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma \max_{a' \in actions(s)} Q^{t-1}(s',a')]$$

q-value in time step ***t*** can be computed via retrieving values from its previous iteration ***(t-1)***

Q-Learning

- In Q-learning, you estimate the Q(s,a) function using samples
  - (s, a, s', r)
  - We will use this sample to update value function $Q^{\pi}(s, a)$
  - Like TD learning, we will maintain a running average of q-value function $Q^{\pi}(s, a)$ as follows:

$$Q^{\pi}(s, a) \leftarrow (1 - \alpha) Q^{\pi}(s, a) + \alpha \; sample_i$$



$$sample_i = R(s, \pi(s), s') + \gamma \max_{a'} Q^{\pi}(s', a')$$

α = learning rate which can vary between 0 to 1.0

Reinforcement Learning Methods