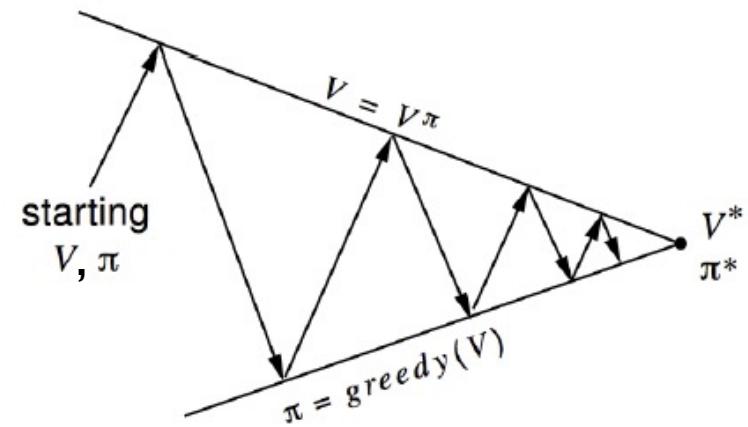
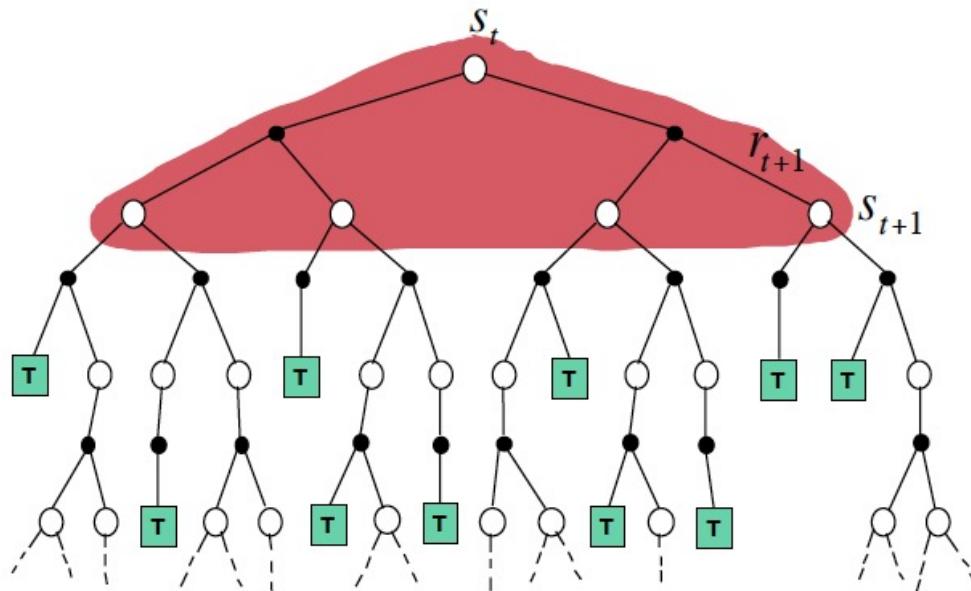


# Model Free Control

Lecture 5  
Subir Varma

# Model Based Policy Evaluation and Optimal Control (Lecture 3)

Model Based  
Full Sweep  
Full Backup



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

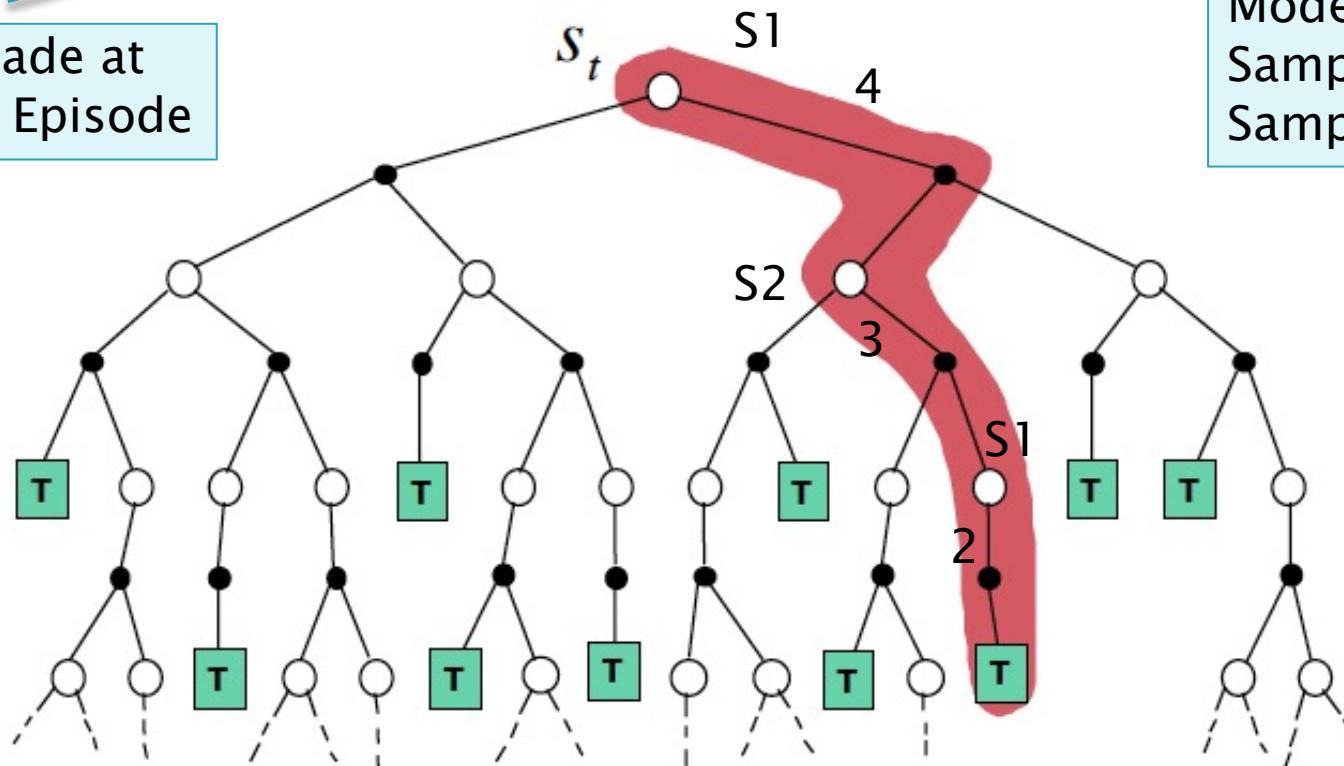
Full Policy Iteration  
Generalized Policy Iteration

# Model Free Policy Evaluation: Monte Carlo Learning (Lecture 4)

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

Update made at end of an Episode

Model Free Sample Sweep Sample Backup



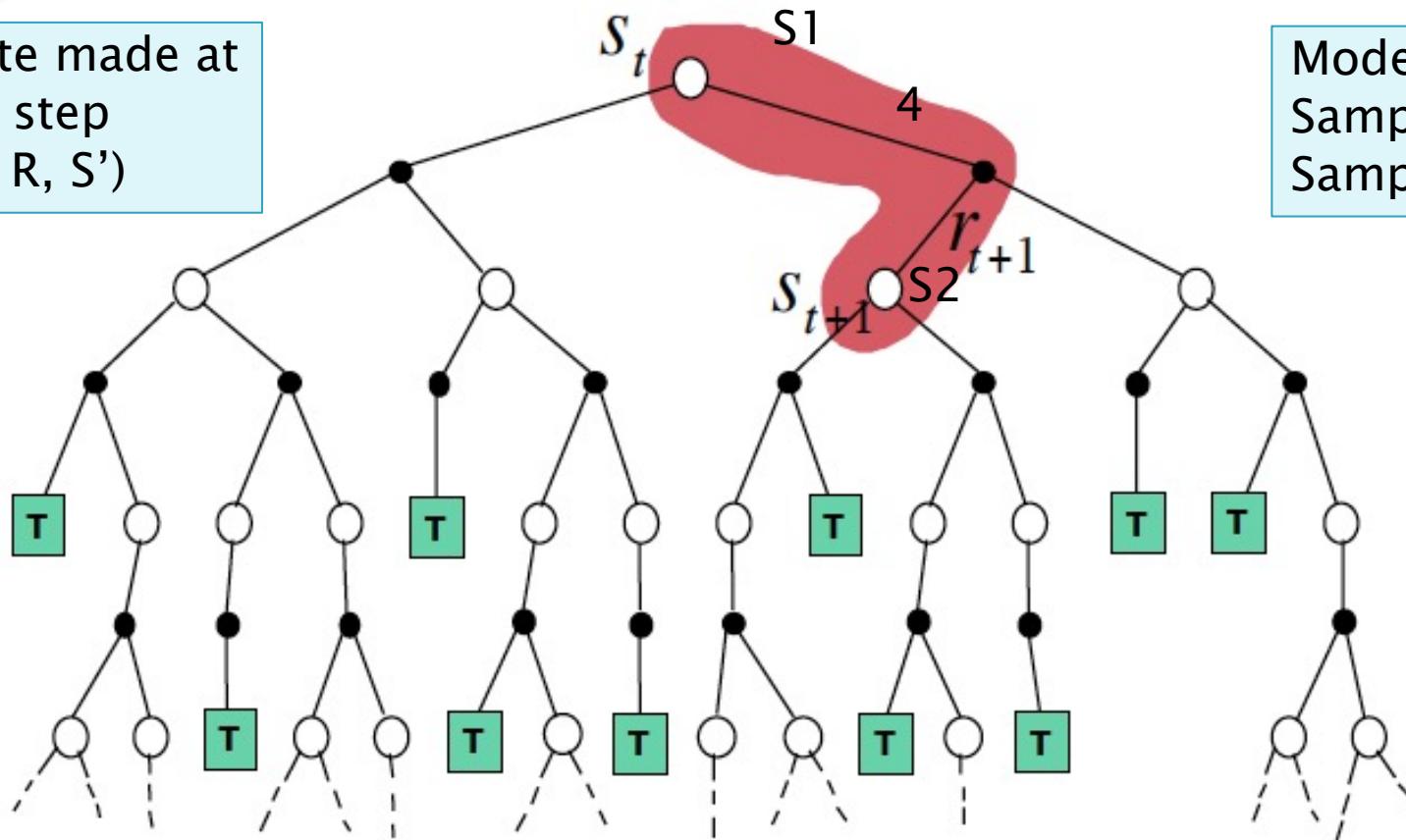
Instead of a Model, we now have sample episodes from the MDP

# Model Free Policy Evaluation: Temporal-Difference (TD) Learning (Lecture 4)

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

Update made at every step ( $S, A, R, S'$ )

Model Free Sample Sweep Sample Backup



Instead of a Model, we now have sample episodes from the MDP

# The Next Step..

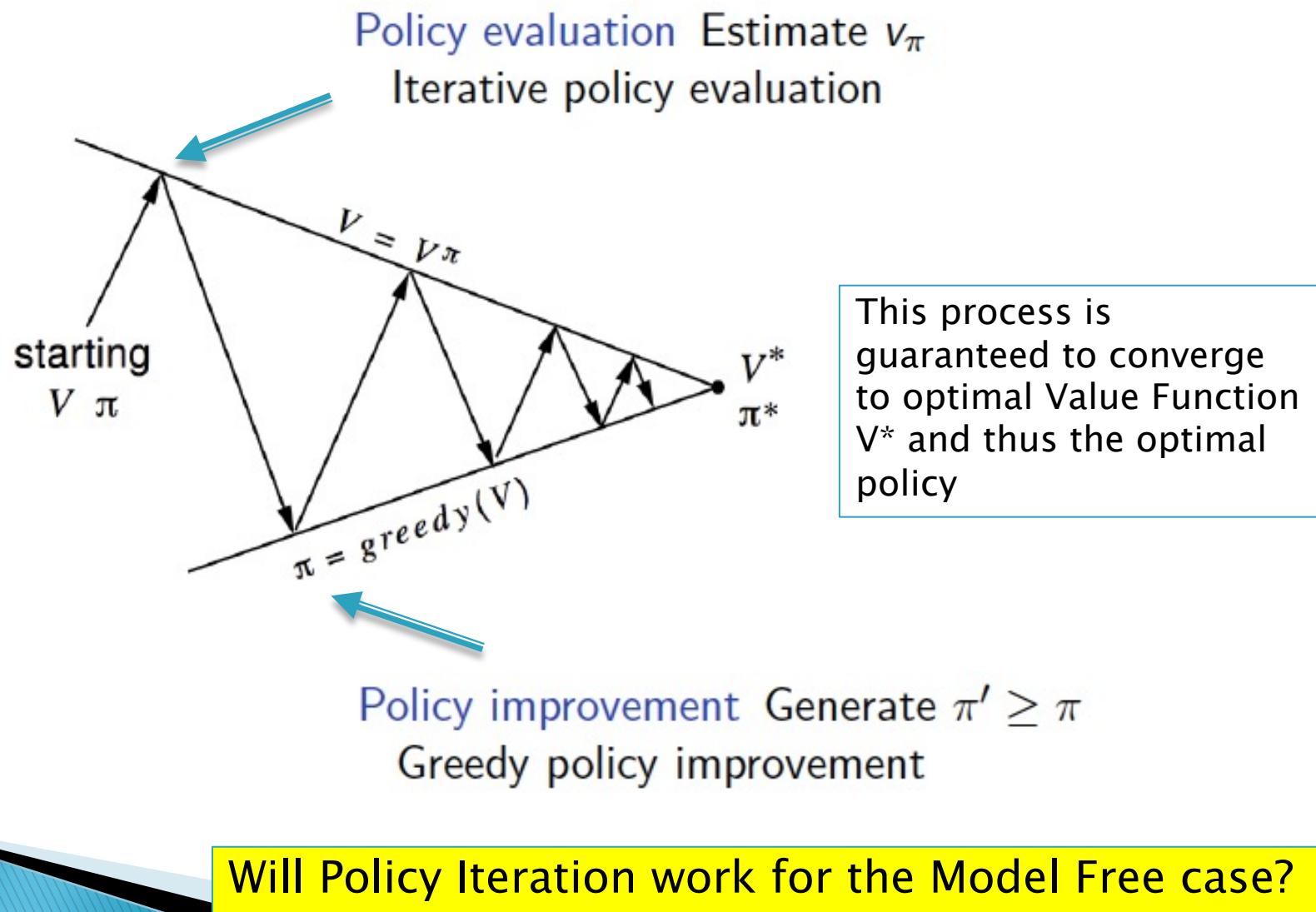
So Far: We have algorithms to find the Value Function  $v_\pi$ , given a policy  $\pi$  (with or without a model)

But: We are really interested in finding the Optimal Policy  $\pi_*$

# Model Free Reinforcement Learning

- Last lecture:
  - Model-free prediction
  - Estimate the value function of an *unknown* MDP
- This lecture:
  - Model-free control
  - Optimise the value function of an *unknown* MDP

# From Lecture 3: Finding the Optimal Policy for the Model Based Case with Policy Iteration



# Uses of Model-Free Control

Some example problems that can be modelled as MDPs

- Elevator
- Parallel Parking
- Ship Steering
- Bioreactor
- Helicopter
- Aeroplane Logistics
- Robocup Soccer
- Quake
- Portfolio management
- Protein Folding
- Robot walking
- Game of Go

For most of these problems, either:

- MDP model is unknown, but experience can be sampled
- MDP model is known, but is too big to use, except by samples

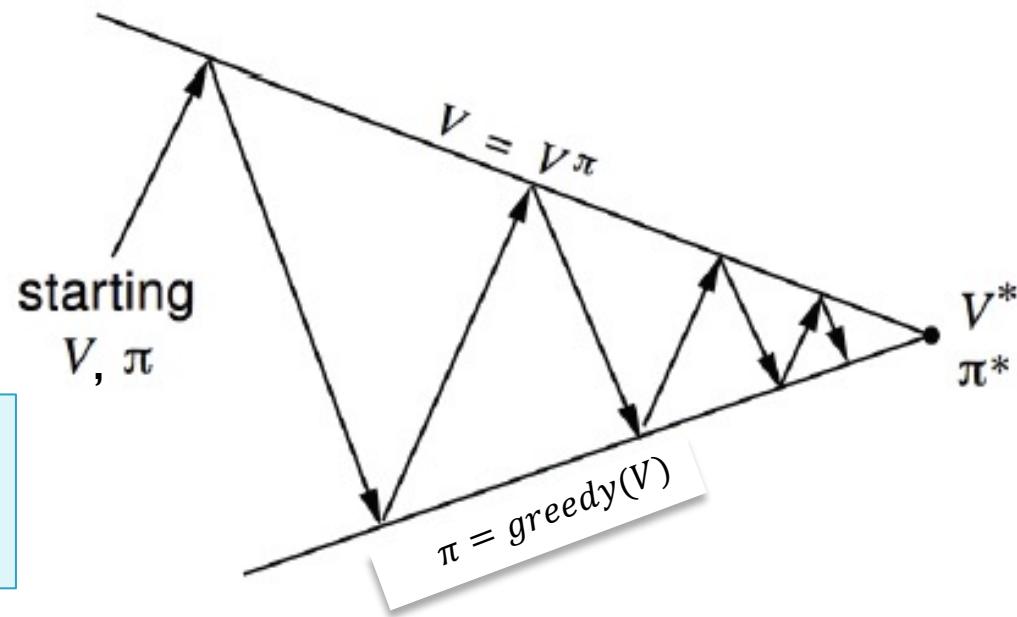
Model-free control can solve these problems

# Monte Carlo Control

# Generalized Policy Iteration

We computed  $v_\pi$  by using  
the Bellman Expectation  
Equation

Can we replace  $v_\pi$  by  $V_\pi$   
obtained with Monte Carlo  
Evaluation?



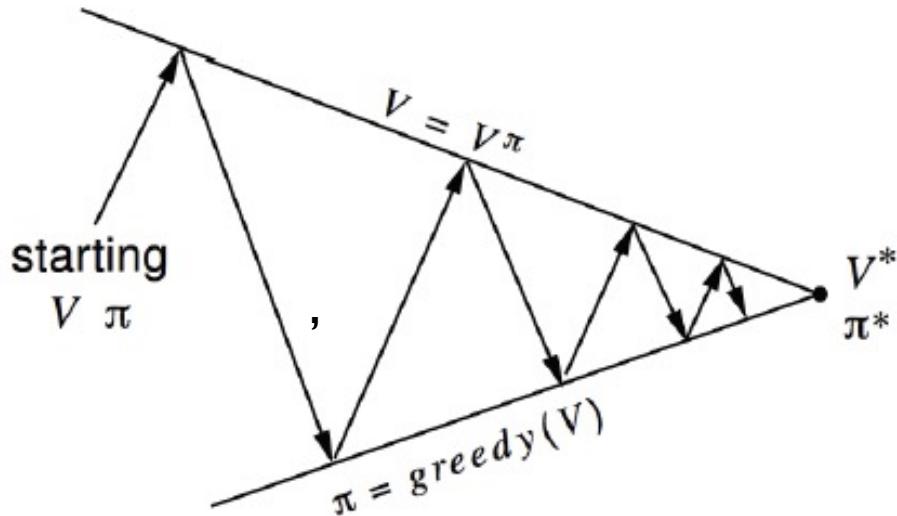
Policy evaluation Estimate  $V_\pi$

e.g. Iterative policy evaluation

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

e.g. Greedy policy improvement

# Generalized Policy Iteration with Monte Carlo Evaluation



Instead of using Bellman Expectation Equation to compute  $v_\pi$ , we are using Monte Carlo Policy evaluation to estimate  $V_\pi$

Policy evaluation Monte-Carlo policy evaluation,  $V = v_\pi$ ?

Policy improvement Greedy policy improvement?

What is wrong with this approach?

# Model-Free Policy Iteration Using Action-Value Function

- Greedy policy improvement over  $V(s)$  requires model of MDP

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} \left[ \mathcal{R}_s^a + \sum \mathcal{P}_{ss'}^a V(s') \right]$$

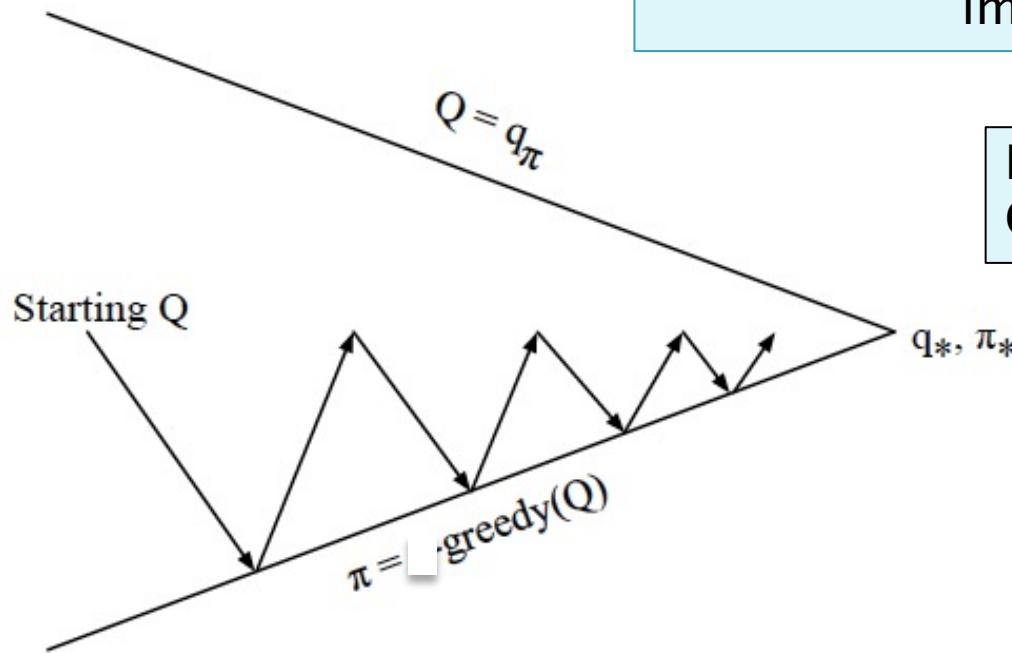
- Greedy policy improvement over  $Q(s, a)$  is model-free

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$$

Estimating  $V_\pi(s)$  is not enough, we need to estimate  $Q_\pi(s, a)$

# Generalized Policy Iteration with Action-Value Function

Run a single episode, and then immediately improve the Policy!



Evaluate new action VF  $Q_\pi(s,a)$   
Compute new policy

Every episode:

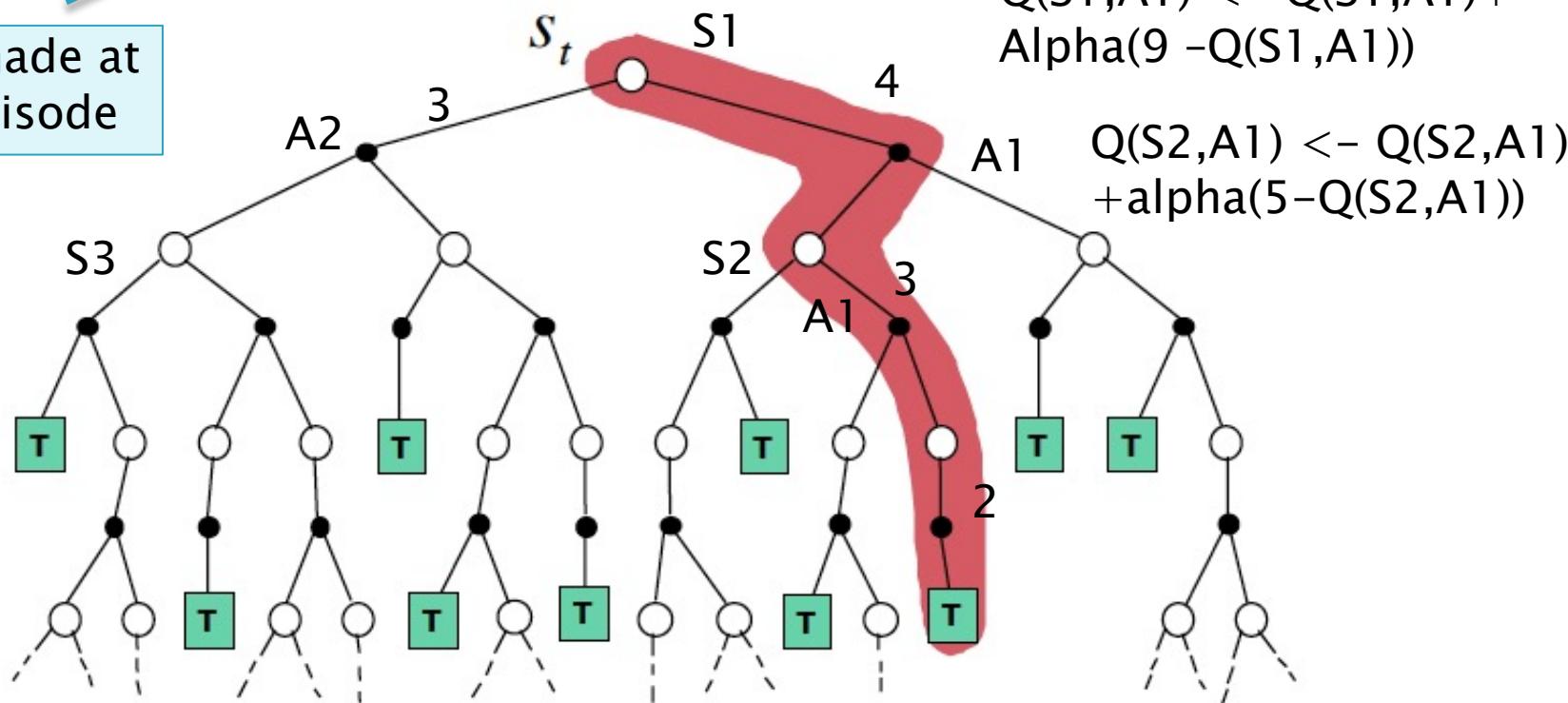
Policy evaluation Monte-Carlo policy evaluation,  $Q \approx q_\pi$

Policy improvement  $\epsilon$ -greedy policy improvement

# Monte Carlo Backup for Q

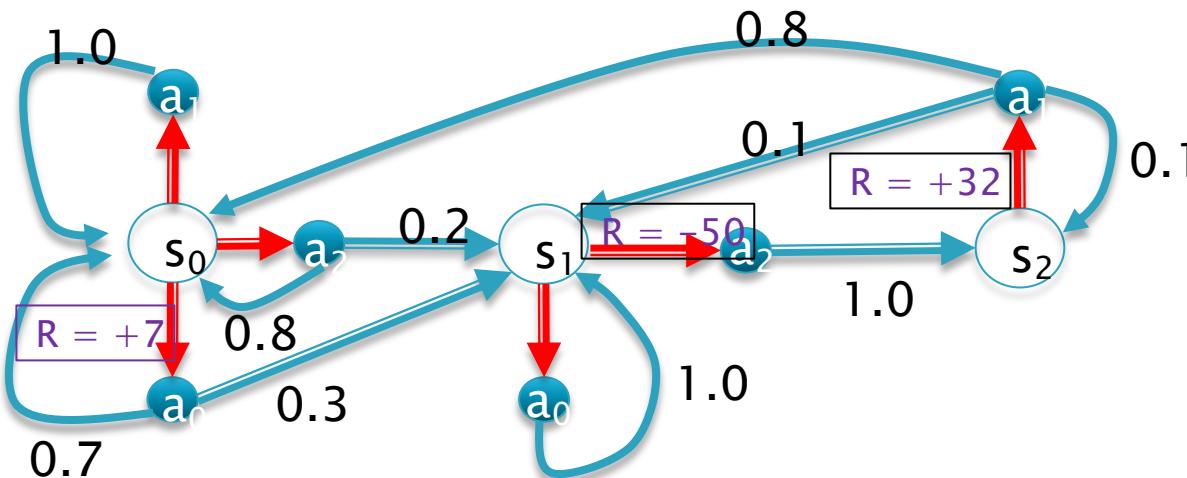
$$Q(S, A) \leftarrow Q(S, A) + \alpha(G - Q(S, A))$$

Update made at  
end of episode



Another Problem:  
How to ensure that  
every (S,A) pair is  
visited?

# Example: Q(S,A) Evaluation



$$\begin{aligned}\pi(s_0) &= a_0 \\ \pi(s_1) &= a_2 \\ \pi(s_2) &= a_1\end{aligned}$$

```
policy_fire
States (+rewards): 0 1 (-50) 2 (40) 0 (10) 0 (10) 0 (10) 0 1 (-50) 2 (40) 0 ... Total rewards = -220
States (+rewards): 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 ... Total rewards = 40
States (+rewards): 0 (10) 0 (10) 0 (10) 0 1 (-50) 2 (40) 0 (10) 0 1 (-50) 2 (40) ... Total rewards = 160
States (+rewards): 0 (10) 0 (10) 0 (10) 0 (10) 0 (10) 0 1 (-50) 2 (40) 0 (10) 0 (10) ... Total rewards = 280
States (+rewards): 0 (10) 0 1 (-50) 2 1 (-50) 2 (40) 0 (10) 0 (10) 0 (10) 0 (10) ... Total rewards = 190
Summary: mean=122.2, std=134.956674, min=-340, max=490
```

$0 \rightarrow 1 (-50) \rightarrow 2 (32) \rightarrow 0 (7) \rightarrow 0 (7) \rightarrow 0 (7) \rightarrow 0 \rightarrow 1 (-50) \rightarrow 2 (32) \rightarrow 0 \dots$

Estimating V

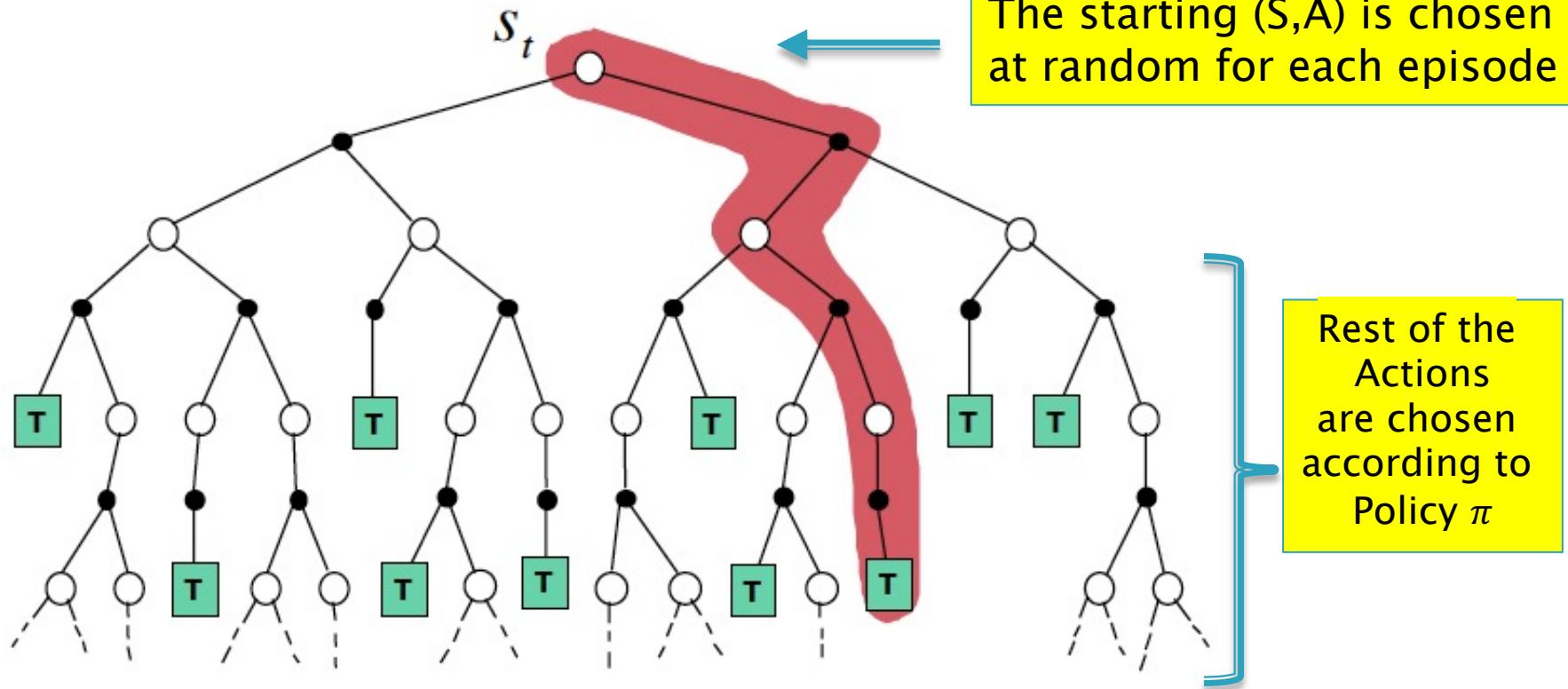
Estimating Q

$0,2 \rightarrow 1,2 (-50) \rightarrow 2,1 (32) \rightarrow 0,0 (7) \rightarrow 0,0 (7) \rightarrow 0,0 (7) \rightarrow 0,0 \dots$

# Monte Carlo Backup for Action Value Functions with Exploring Starts

Update made at  
end of an Episode

$$Q(S, A) \leftarrow Q(S, A) + \alpha(G - Q(S, A))$$



# Monte Carlo Control with Exploring Starts

Monte Carlo ES (Exploring Starts), for estimating  $\pi \approx \pi_*$

Initialize, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$Q(s, a) \leftarrow$  arbitrary

$\pi(s) \leftarrow$  arbitrary

$Returns(s, a) \leftarrow$  empty list

Repeat forever:

Choose  $S_0 \in \mathcal{S}$  and  $A_0 \in \mathcal{A}(S_0)$  s.t. all pairs have probability  $> 0$

Generate an episode starting from  $S_0, A_0$ , following  $\pi$

For each pair  $s, a$  appearing in the episode:

$G \leftarrow$  the return that follows the first occurrence of  $s, a$

Append  $G$  to  $Returns(s, a)$

$Q(s, a) \leftarrow$  average( $Returns(s, a)$ )

For each  $s$  in the episode:

$\pi(s) \leftarrow \arg \max_a Q(s, a)$

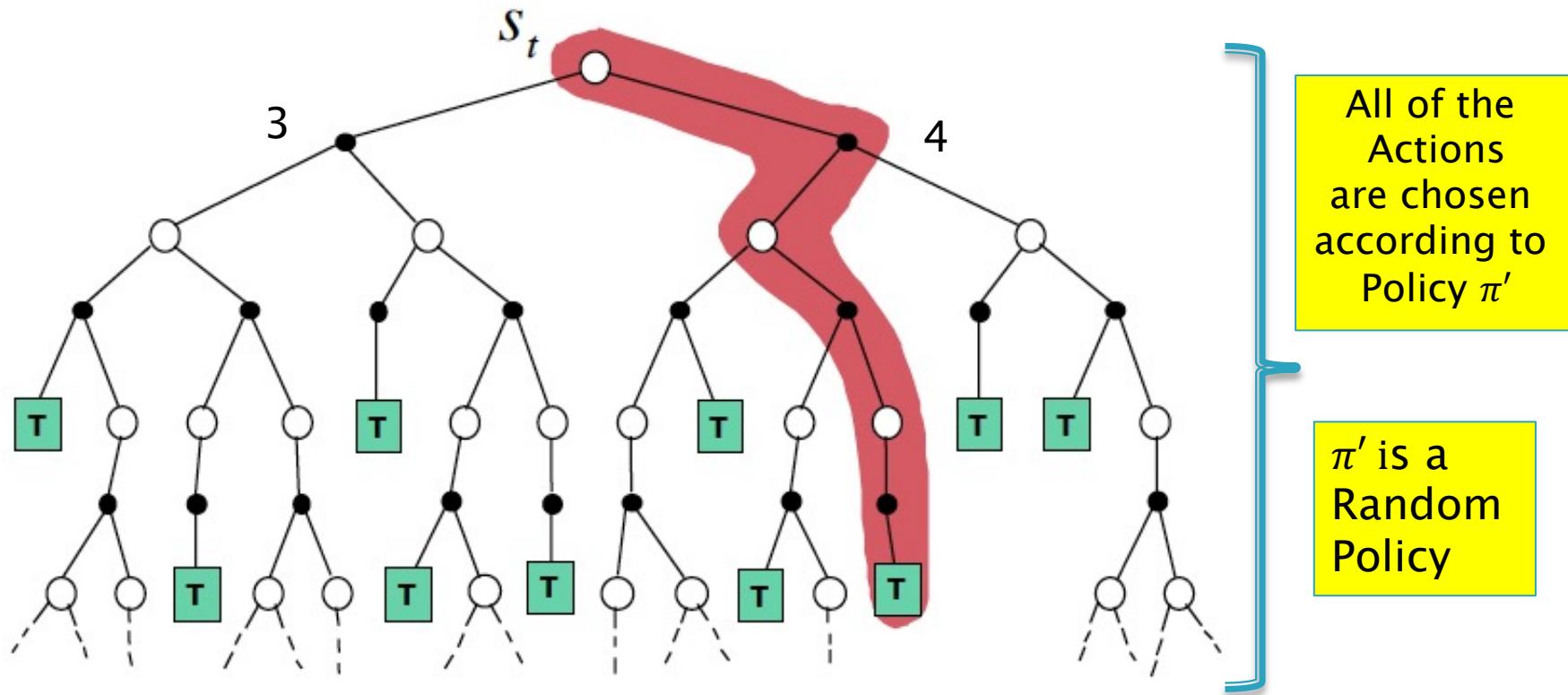
# How to Avoid the Exploring Starts Assumption

General Strategy: Continue to select all possible Actions (even during an episode)

But: The agent is supposed to follow Policy  $\pi$ .

# Idea: Randomize the Policy!

$$Q(S, A) \leftarrow Q(S, A) + \alpha(G - Q(S, A))$$



Policy Improvement Update made at end of an Episode

# $\epsilon$ –Greedy Exploration

- Simplest idea for ensuring continual exploration
- All  $m$  actions are tried with non-zero probability
- With probability  $1 - \epsilon$  choose the greedy action
- With probability  $\epsilon$  choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in A}{\operatorname{argmax}} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Works very well in practice

Guarantees that you continue to explore everything

Guarantees that you improve your policy

Toss Coin

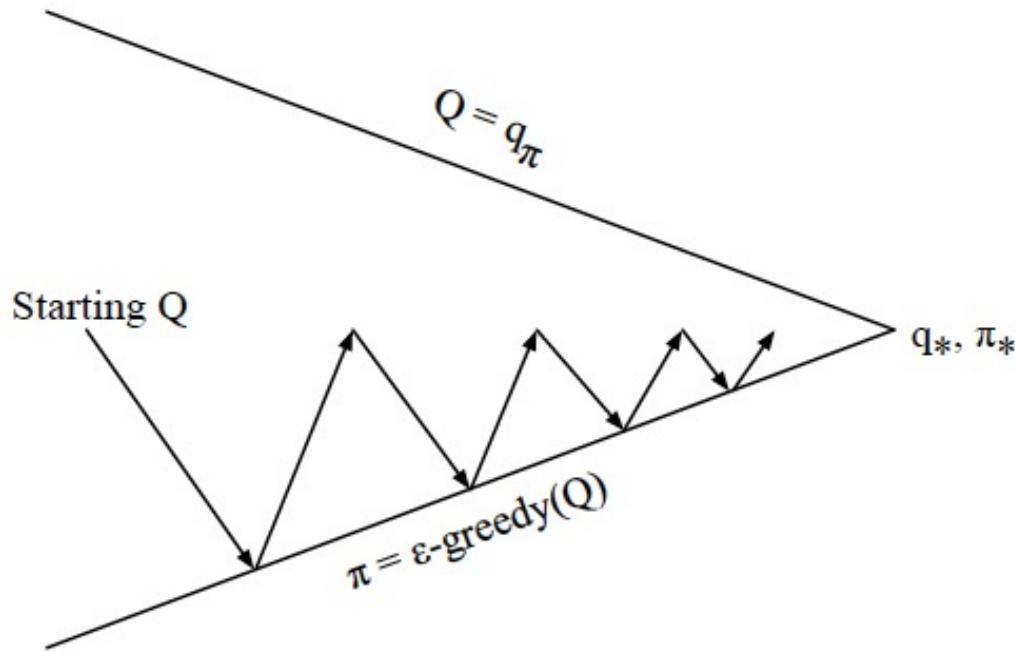
$\epsilon$

$1 - \epsilon$

Choose  
Random  
Action

Choose  
Greedy  
Action

# Generalized Policy Iteration with Action–Value Function and $\epsilon$ Greedy Exploration



Every episode:

Policy evaluation Monte-Carlo policy evaluation,  $Q \approx q_\pi$

Policy improvement  $\epsilon$ -greedy policy improvement

# On-Policy First Visit MC Control with $\epsilon$ Greedy Policies

On-policy first-visit MC control (for  $\epsilon$ -soft policies), estimates  $\pi \approx \pi_*$

Initialize, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$Q(s, a) \leftarrow$  arbitrary

$Returns(s, a) \leftarrow$  empty list

$\pi(a|s) \leftarrow$  an arbitrary  $\epsilon$ -soft policy

Repeat forever:

(a) Generate an episode using  $\pi$

(b) For each pair  $s, a$  appearing in the episode:

$G \leftarrow$  the return that follows the first occurrence of  $s, a$

Append  $G$  to  $Returns(s, a)$

$Q(s, a) \leftarrow$  average( $Returns(s, a)$ )

(c) For each  $s$  in the episode:

$A^* \leftarrow \arg \max_a Q(s, a)$  (with ties broken arbitrarily)

For all  $a \in \mathcal{A}(s)$ :

$$\pi(a|s) \leftarrow \begin{cases} 1 - \epsilon + \epsilon / |\mathcal{A}(s)| & \text{if } a = A^* \\ \epsilon / |\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

# Exploration and Exploitation

- *Exploration* finds more information about the environment
- *Exploitation* exploits known information to maximise reward
- It is usually important to explore as well as exploit

# Examples of Exploration and Exploitation

- Restaurant Selection

Exploitation Go to your favourite restaurant

Exploration Try a new restaurant

- Online Banner Advertisements

Exploitation Show the most successful advert

Exploration Show a different advert

- Oil Drilling

Exploitation Drill at the best known location

Exploration Drill at a new location

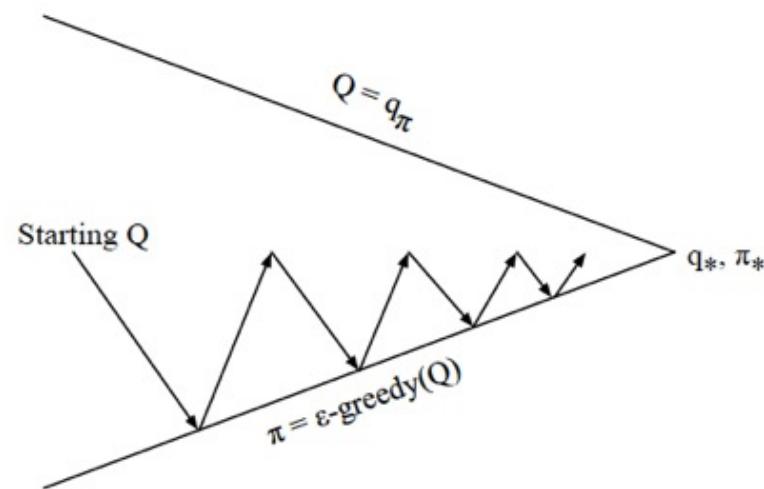
- Game Playing

Exploitation Play the move you believe is best

Exploration Play an experimental move

# One More Problem ...

- ▶ We know that the Optimal Policy is NOT Random
- ▶ We need a way to gradually reduce the randomness in the Policy



# Solution: GLIE

## Definition

*Greedy in the Limit with Infinite Exploration* (GLIE)

- All state-action pairs are explored infinitely many times,

$$\lim_{k \rightarrow \infty} N_k(s, a) = \infty$$

True for  
epsilon greedy

- The policy converges on a greedy policy,

$$\lim_{k \rightarrow \infty} \pi_k(a|s) = \mathbf{1}(a = \operatorname{argmax}_{a' \in \mathcal{A}} Q_k(s, a'))$$

Policy  
eventually  
becomes  
greedy

- For example,  $\epsilon$ -greedy is GLIE if  $\epsilon$  reduces to zero at  $\epsilon_k = \frac{1}{k}$

# GLIE Monte-Carlo Control

- Sample  $k$ th episode using  $\pi$ :  $\{S_1, A_1, R_2, \dots, S_T\} \sim \pi$
- For each state  $S_t$  and action  $A_t$  in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- Improve policy based on new action-value function

$$\epsilon \leftarrow 1/k$$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

## Theorem

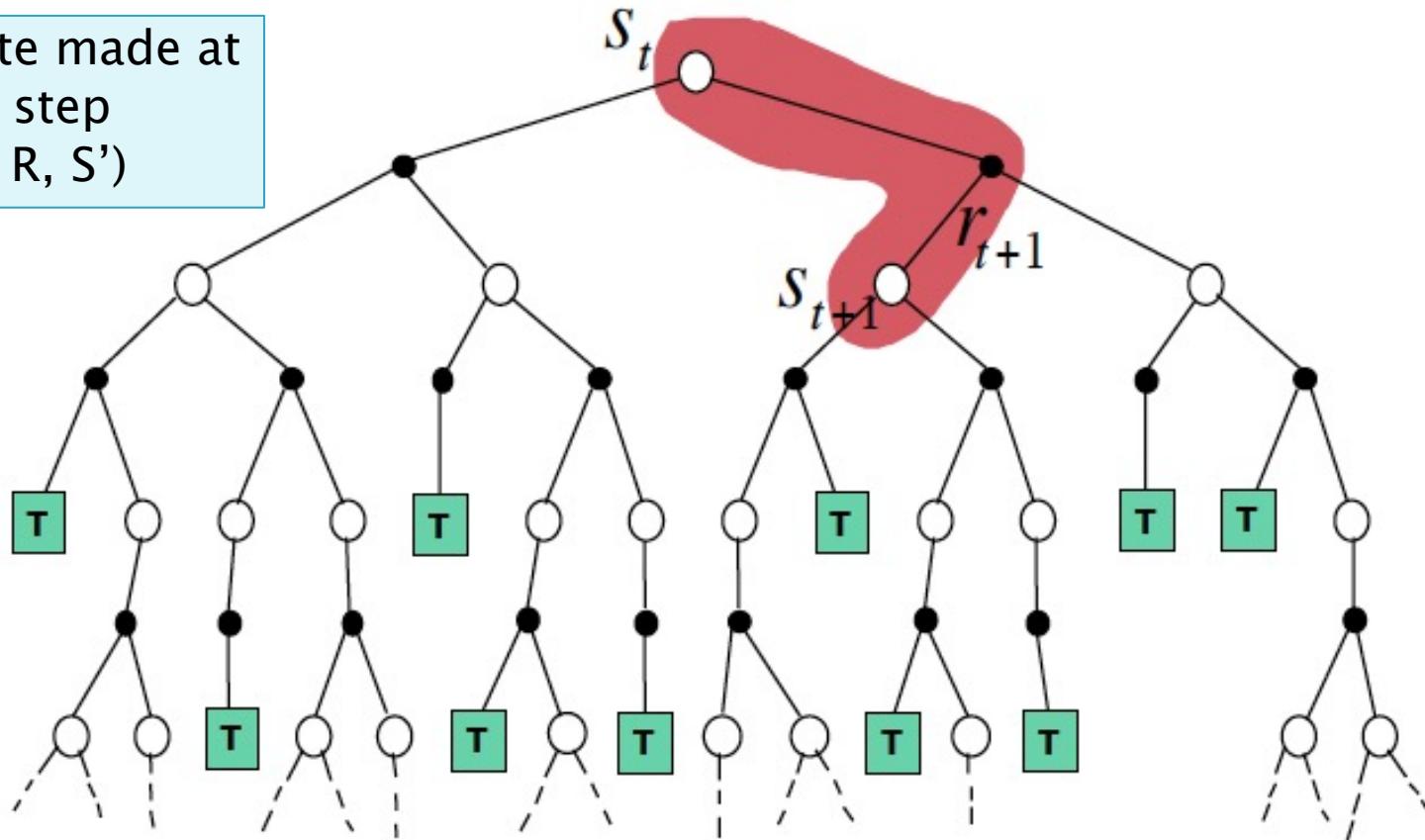
*GLIE Monte-Carlo control converges to the optimal action-value function,  $Q(s, a) \rightarrow q_*(s, a)$*

# On Policy TD Control: The SARSA Algorithm

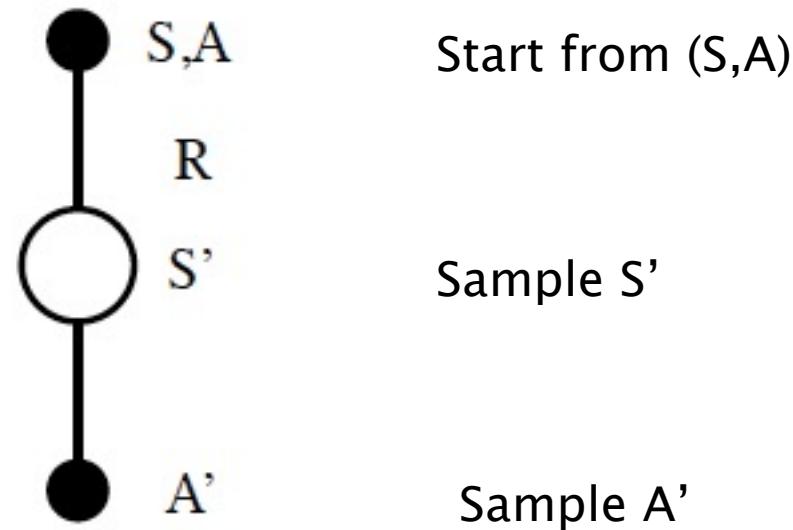
# Recall: Temporal-Difference (TD) Learning

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

Update made at  
every step  
(S, A, R, S')



# Updating Q Functions with SARSA



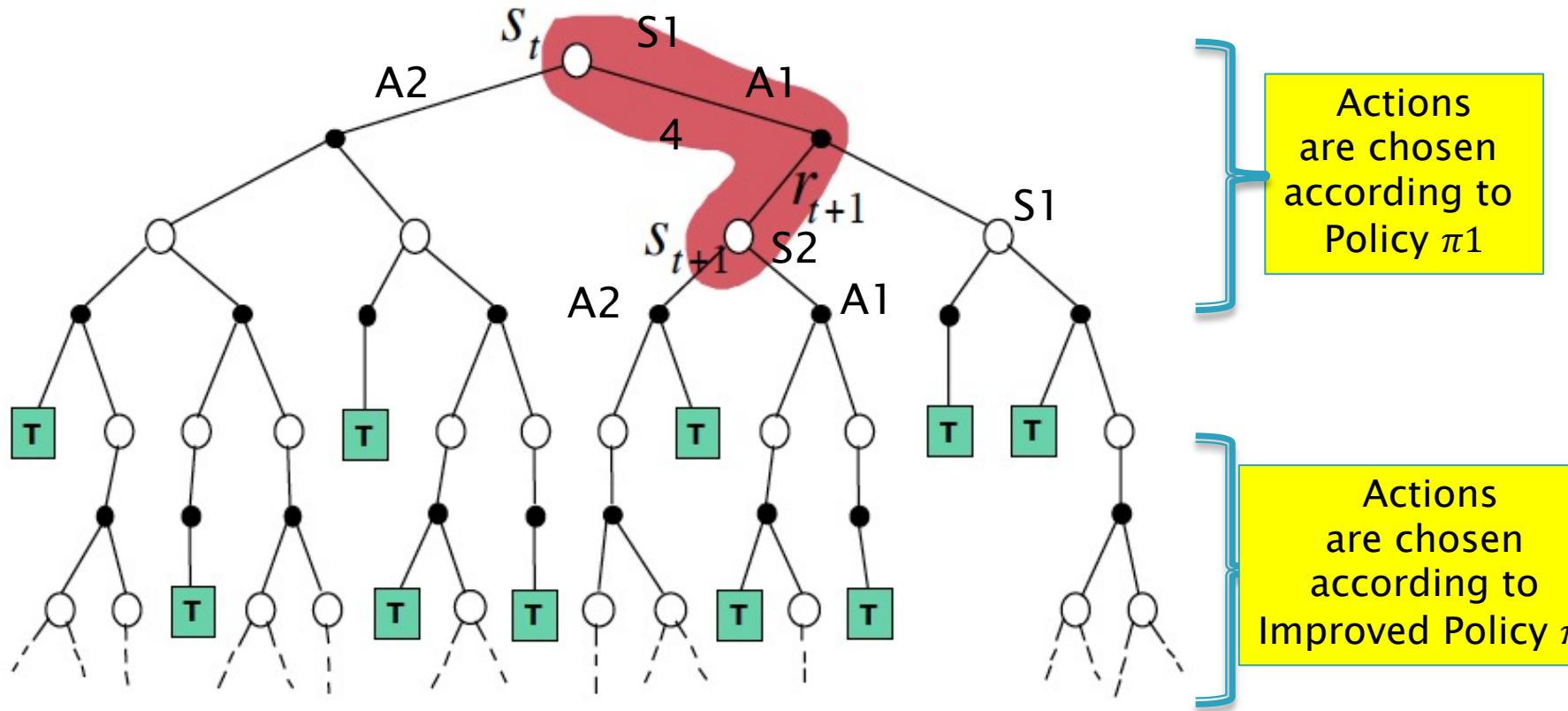
$$Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma Q(S', A') - Q(S, A))$$

For Value Functions:

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

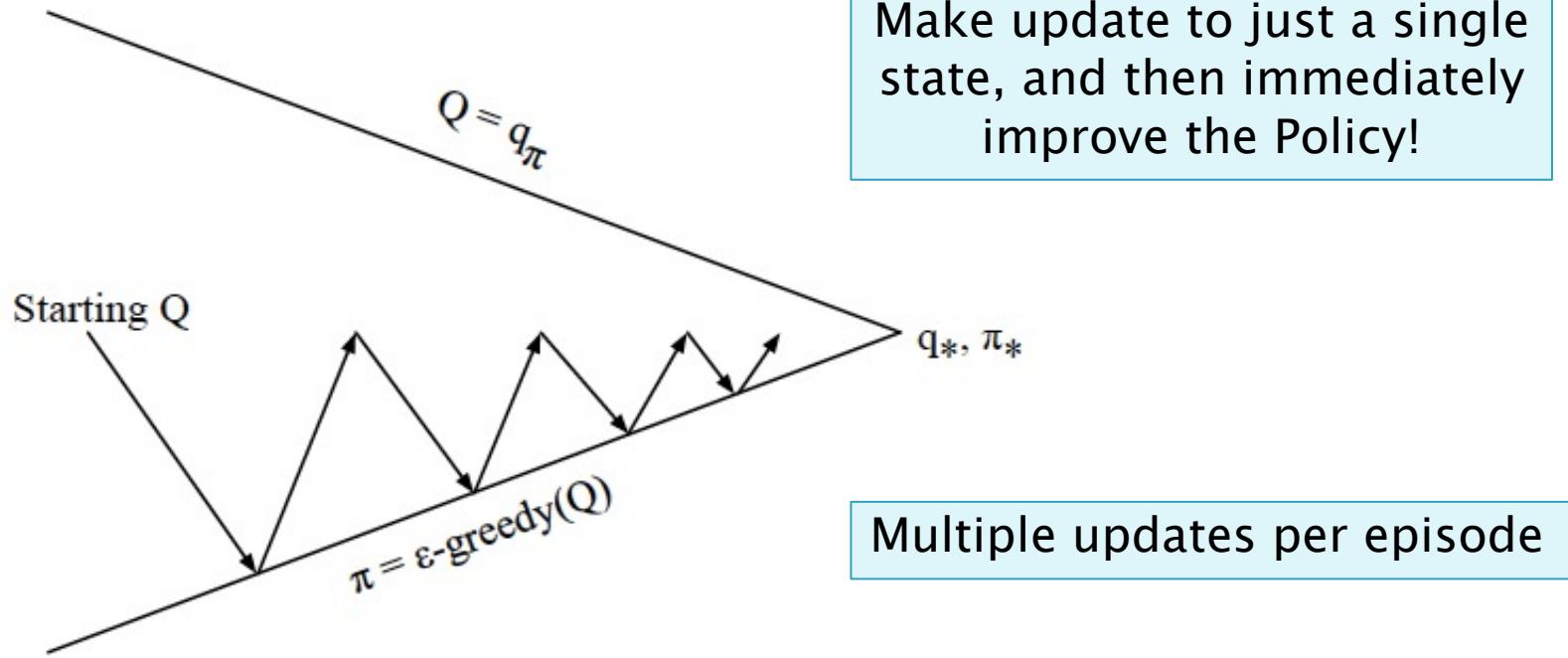
# Model Free On Policy Temporal-Difference Algorithm: SARSA

$$Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma Q(S', A') - Q(S, A))$$



Policy Improvement Update made after each step!

# On-Policy Control with SARSA



Every time-step:

Policy evaluation **Sarsa**,  $Q \approx q_\pi$

Policy improvement  $\epsilon$ -greedy policy improvement

# MC vs TD Control

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences

# SARSA Algorithm for On-Policy Control

**Sarsa (on-policy TD control) for estimating  $Q \approx q_*$**

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

    Initialize  $S$

    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

    Repeat (for each step of episode):

        Take action  $A$ , observe  $R, S'$

        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$S \leftarrow S'; A \leftarrow A'$ ;

    until  $S$  is terminal

# Convergence of SARSA

## Theorem

*Sarsa converges to the optimal action-value function,  
 $Q(s, a) \rightarrow q_*(s, a)$ , under the following conditions:*

- GLIE sequence of policies  $\pi_t(a|s)$
- Robbins-Monro sequence of step-sizes  $\alpha_t$

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

# Off Policy Temporal Difference Control: The Q Learning Algorithm

# On and Off-Policy Learning

- On-policy learning
  - “Learn on the job”
  - Learn about policy  $\pi$  from experience sampled from  $\pi$
- Off-policy learning
  - “Look over someone’s shoulder”
  - Learn about policy  $\pi$  from experience sampled from  $\mu$

# Off Policy Control with Q Learning

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \operatorname{argmax}_{a'} Q(S', a') - Q(S, A))$$

Behavior Policy

Target Policy

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

Next Action chosen according to a randomized Behavior Policy  
This ensures Exploration of the State Space

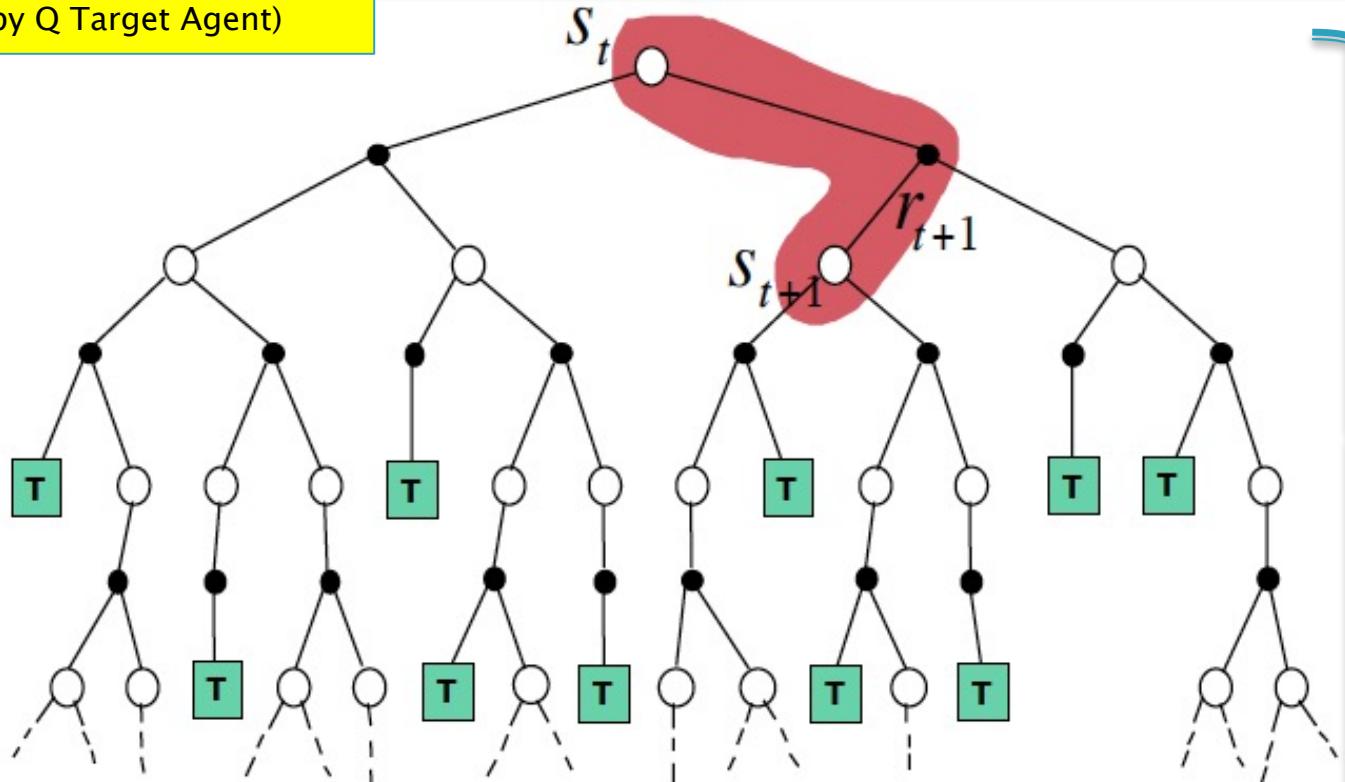
But: Q-Value Update made according to the ‘Optimal’ Target Policy

$$Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma Q(S', A') - Q(S, A))$$

# Model Free Off Policy Temporal-Difference Algorithm - Q Learning

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

The Q value updates made at each step according to Target policy (by Q Target Agent)



All of the Actions chosen according to Action Policy  $\pi_1$  (by Behavior Agent)

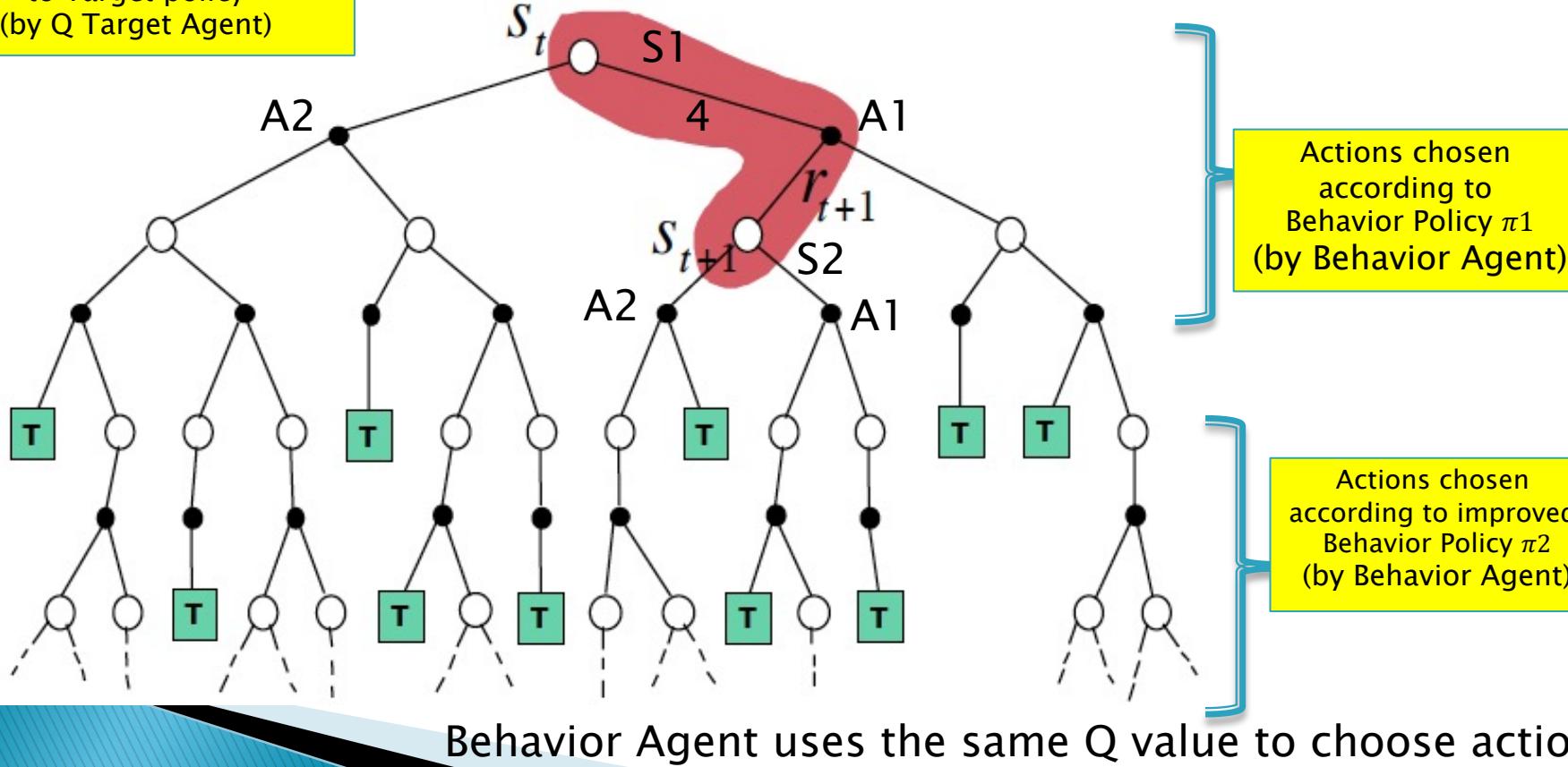
Two Agents  
Behavior Agent  
Target Agent

# Q Learning: Behavior Policy also Evolves

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

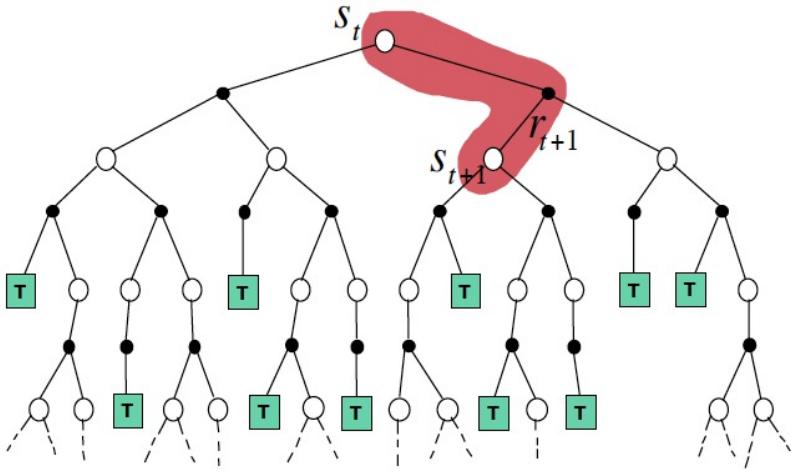
The Q value updates made at each step according to Target policy (by Q Target Agent)

Use  $\epsilon$  Greedy to Choose Action Policy

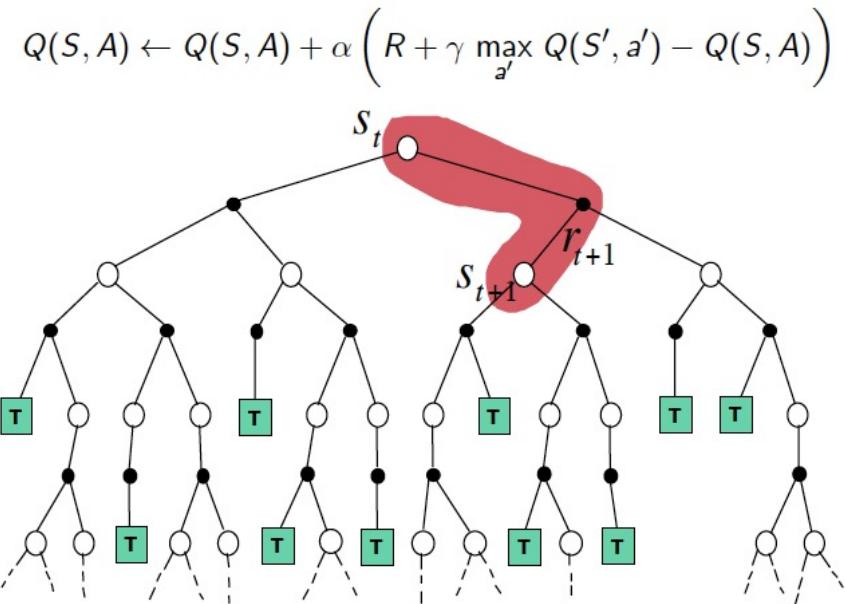


# General Off Policy Learning

Behavior Agent chooses actions  
Based on its own policy. For example  
It can simply choose action with  
Equal probability



Behavior Agent



Target Agent

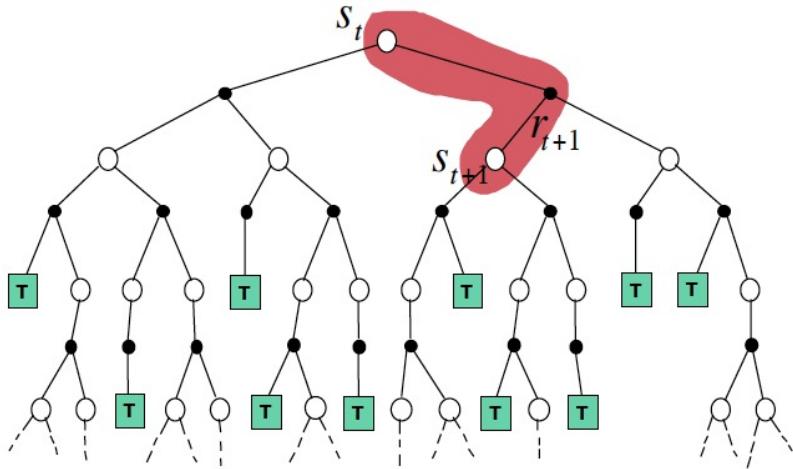
Follows Behavior Agent  
AND In Parallel  
Computes Best Possible Action

# Q Learning

A special case of  
Off Policy Learning

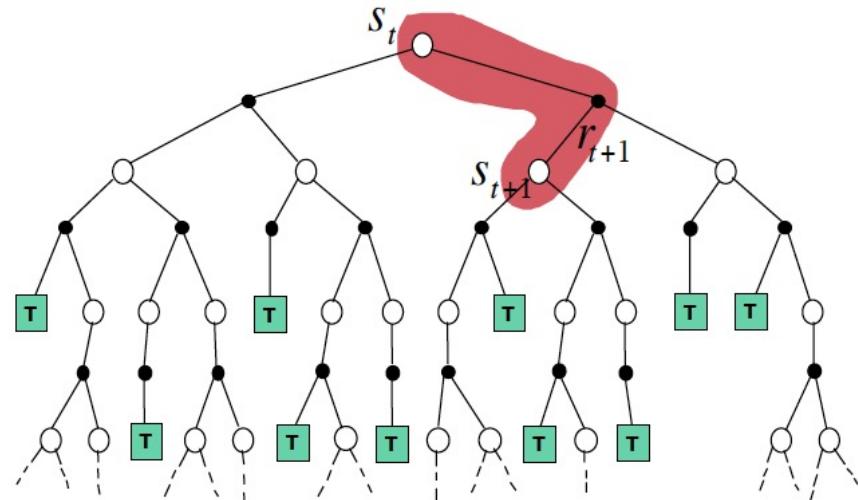
Behavior Agent chooses actions using  
The Q values that the Target Agent  
computes

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$



Behavior Agent

Controls All Actions Actually Taken  
Using epsilon-greedy algo



Target Agent

Follows Behavior Agent  
AND In Parallel  
Computes Best Possible Action

# Q Learning Algorithm for Off Policy Control

Q-learning (off-policy TD control) for estimating  $\pi \approx \pi_*$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

    Initialize  $S$

    Repeat (for each step of episode):

        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

        Take action  $A$ , observe  $R, S'$

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

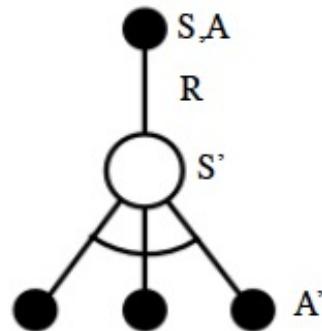
$S \leftarrow S'$

    until  $S$  is terminal

Action Policy

Target Policy

# Q Learning Control Algorithm



$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

## Theorem

*Q-learning control converges to the optimal action-value function,*  
 $Q(s, a) \rightarrow q_*(s, a)$

# Other Uses of Off Policy Learning

- Learn about *optimal* policy while following *exploratory* policy
- Learn from observing humans or other agents
- Learn about *multiple* policies while following *one* policy
- Re-use experience generated from old policies  $\pi_1, \pi_2, \dots, \pi_{t-1}$



Critical Idea used in  
Deep Reinforcement Learning

# Q Learning in Batch Mode

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

(S1,A1,R1,S1')

(S2,A2,R2,S2')

(S3,A3,R3,S3')

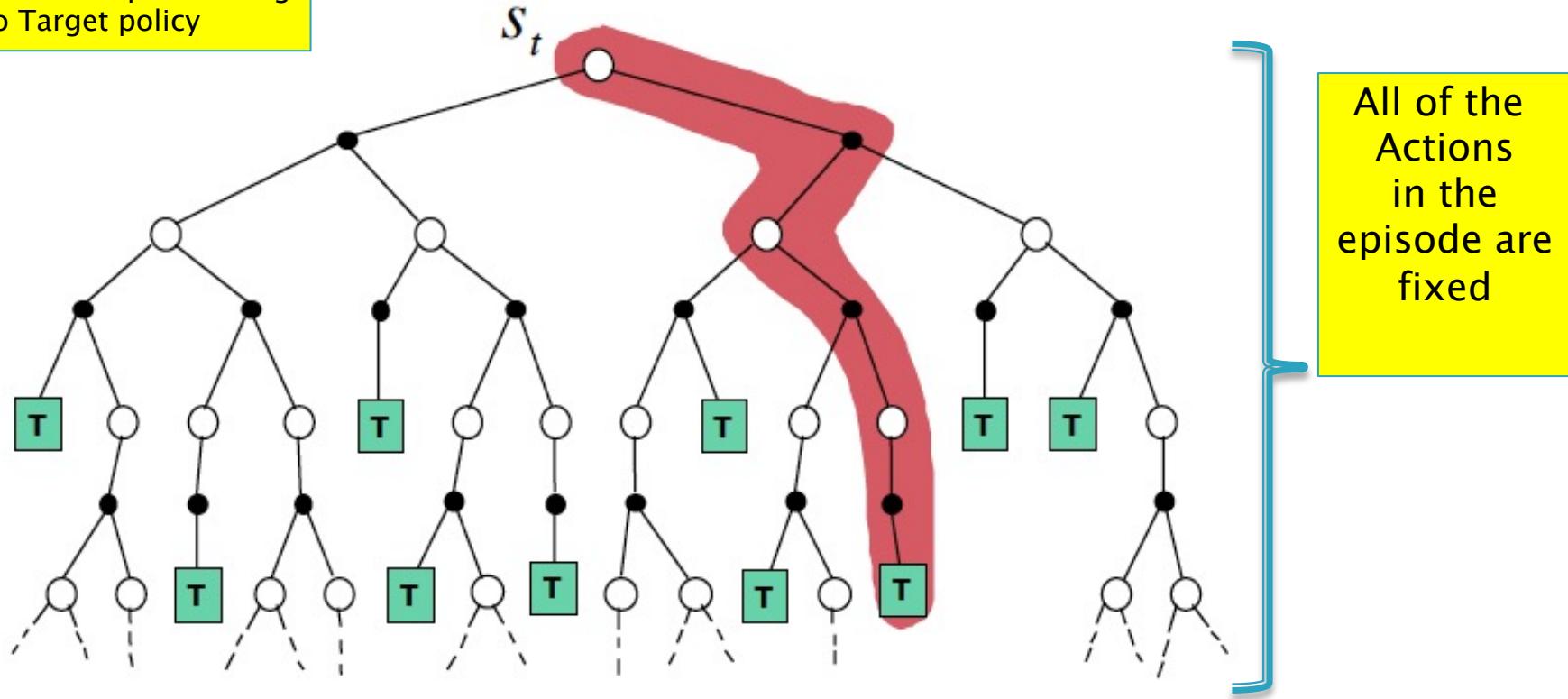
(S4,A4,R4,S4')

A Collection of  
1-Step Transitions

# Q Learning in Batch Mode

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

The Q value updates made at each step according to Target policy



Allows reuse of a single episode multiple times

# Summary

# Summary

- ▶ **On Policy Monte Carlo Control:** This involves a Policy Evaluation step followed by a Policy Improvement Step. Policy Evaluation is done based on the data from a complete episode of the MDP. This is followed by Policy Improvement using the new Q values, and the new policy is used to generate the next episode.
- ▶ **On Policy Temporal Difference Control (SARSA):** This also involves Policy Evaluation followed by Policy Improvement. However the Policy Evaluation is based on the data from a single step of the MDP. This is immediately followed by Policy Improvement, and the improved policy is used to generate the next step of the MDP.
- ▶ **Off Policy Temporal Difference Control (Q Learning):** In this case the Agent taking the Actions (using the Behavior Policy) is different from the Agent computing the optimal Q function (using the Target Policy). Behavior Policy uses some randomness to traverse the MDP, and Target Policy uses the data generated from this traversal to compute the optimal Q function.

# Further Reading

Sutton and Barto:

- ▶ Chapter 5: Sections 5.3 – 5.4
- ▶ Chapter 6: Sections 6.4 – 6.5