

# Chapter 6: Temporal Difference Learning

---

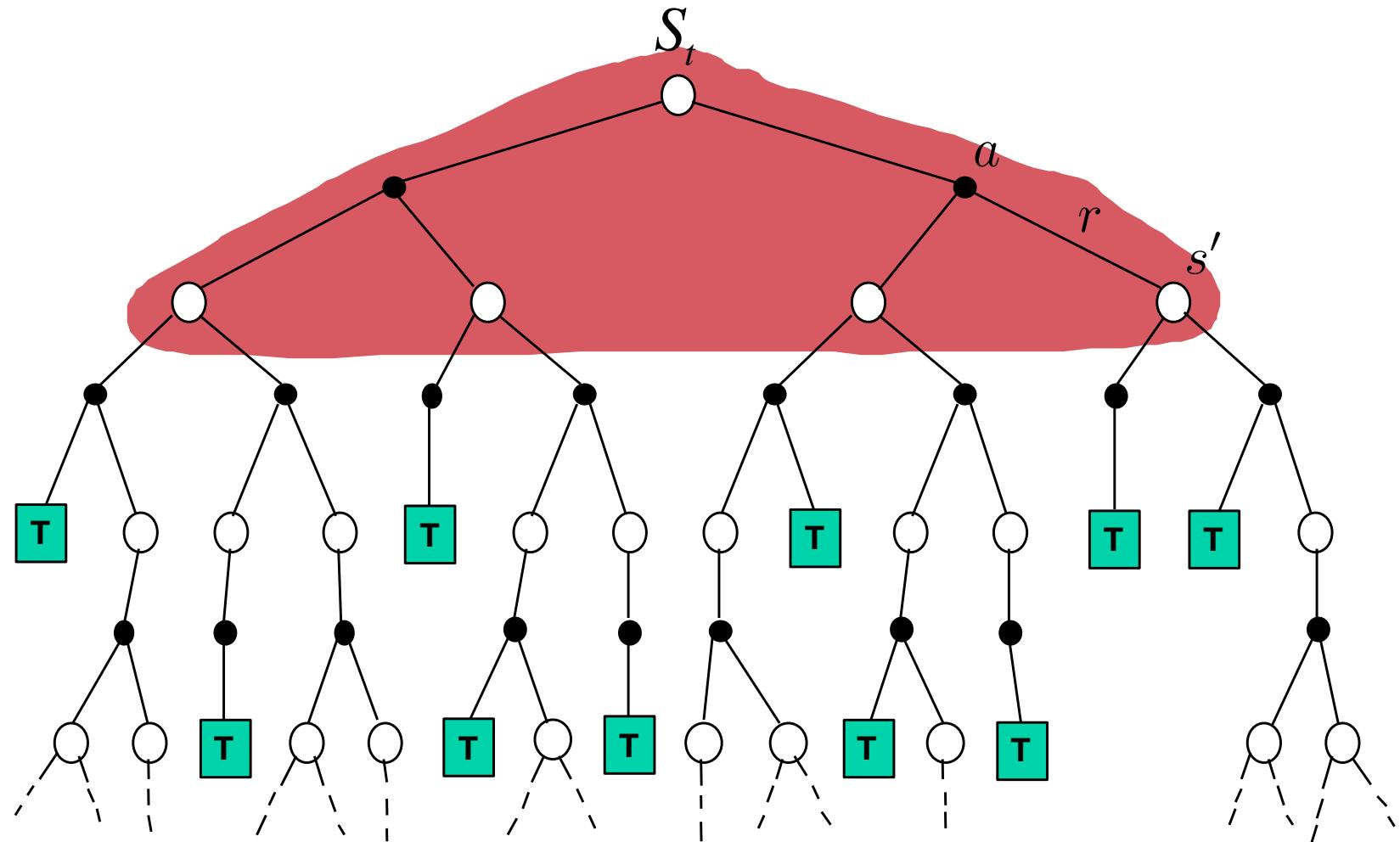
Objectives of this chapter:

- Introduce Temporal Difference (TD) learning
- Focus first on policy evaluation, or prediction, methods
- Compare efficiency of TD learning with MC learning
- Then extend to control methods

# cf. Dynamic Programming

---

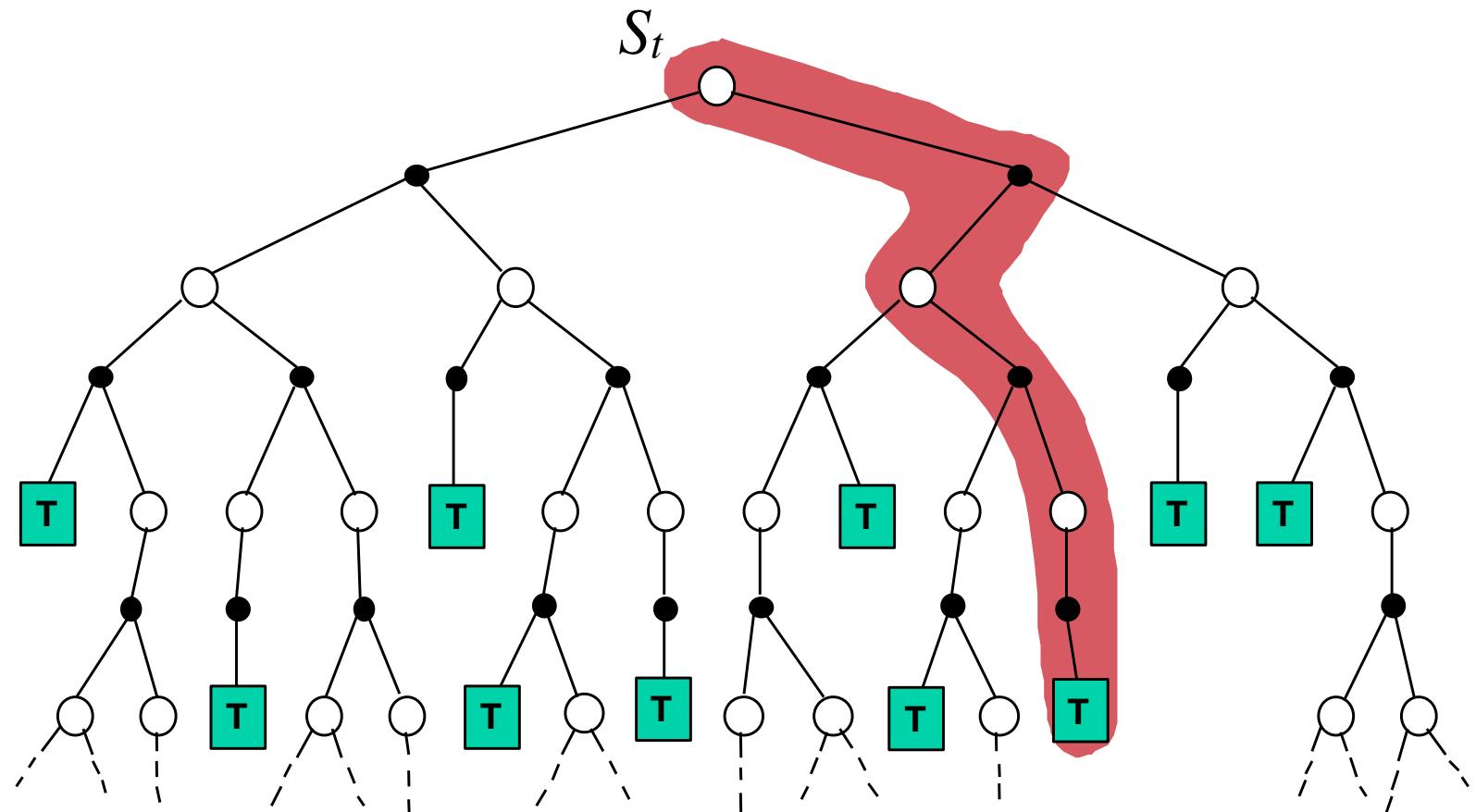
$$V(S_t) \leftarrow E_{\pi} \left[ R_{t+1} + \gamma V(S_{t+1}) \right] = \sum_a \pi(a|S_t) \sum_{s',r} p(s',r|S_t,a) [r + \gamma V(s')]$$



# Simple Monte Carlo

---

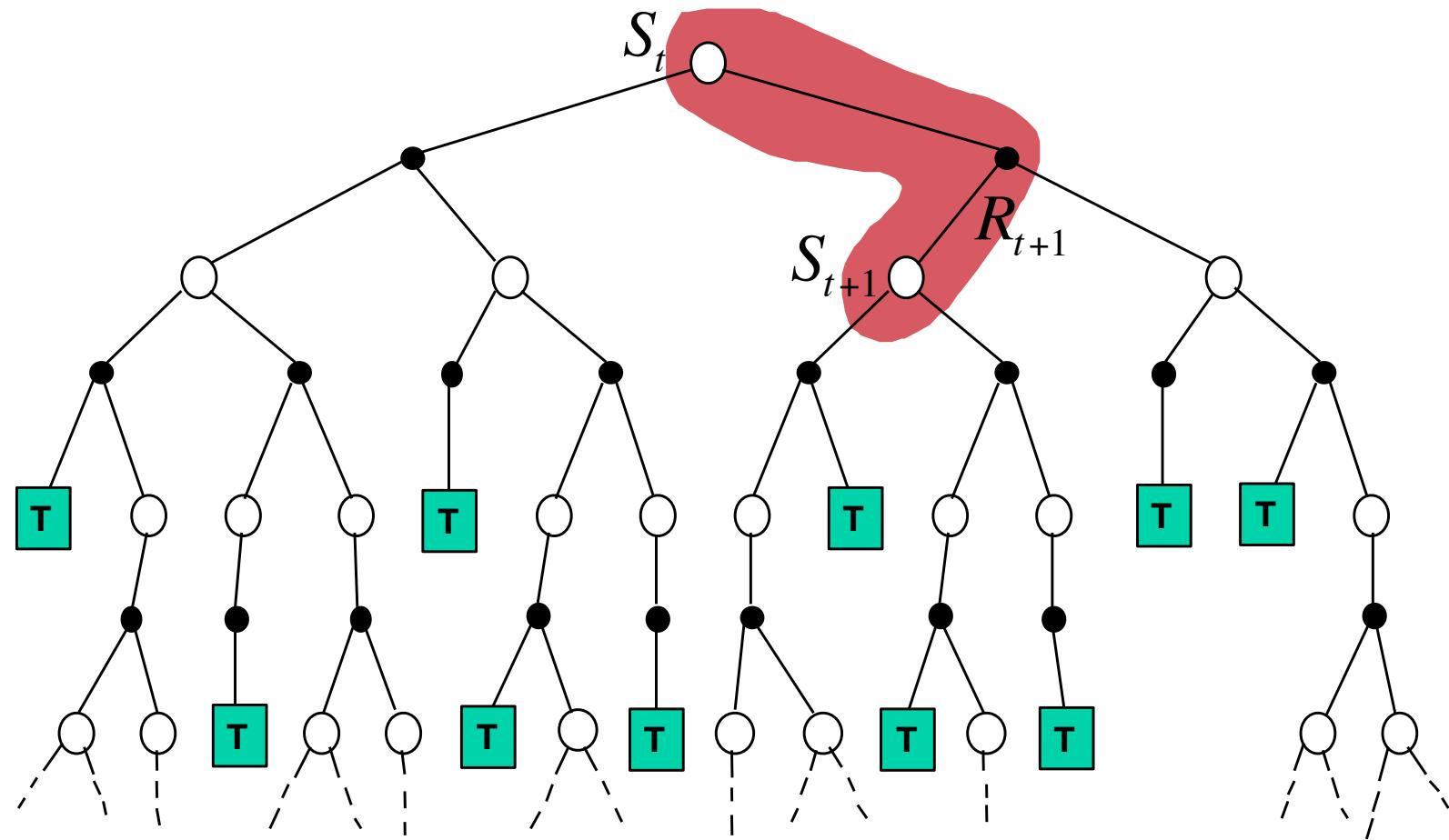
$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)]$$



# Simplest TD Method

---

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



# TD methods bootstrap and sample

---

- **Bootstrapping:** update involves an *estimate*
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- **Sampling:** update does not involve an *expected value*
  - MC samples
  - DP does not sample
  - TD samples

# TD Prediction

---

**Policy Evaluation (the prediction problem):**

for a given policy  $\pi$ , compute the state-value function  $v_\pi$

Recall: Simple every-visit Monte Carlo method:

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

**target**: the actual return after time  $t$

The simplest temporal-difference method TD(0):

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

**target**: an estimate of the return

# Example: Driving Home

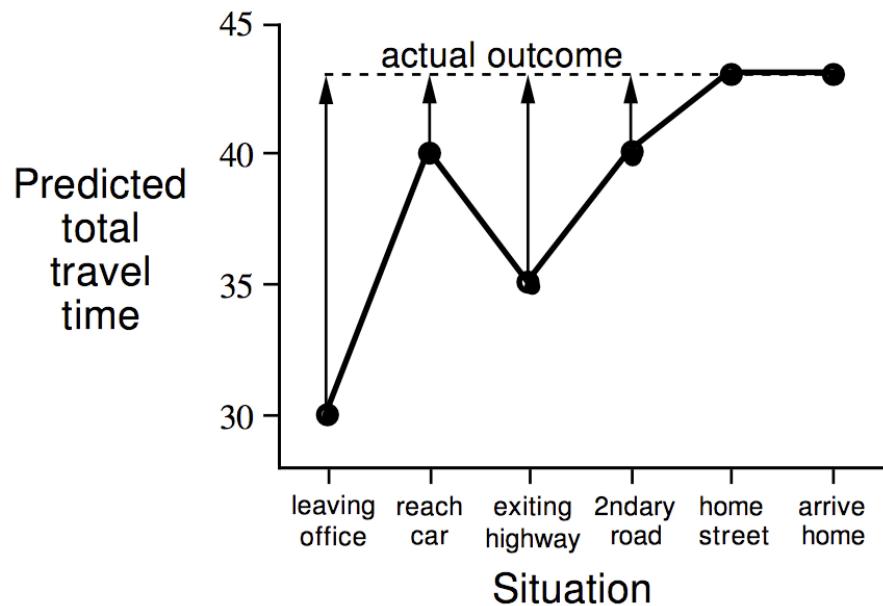
---

<i>State</i>	<i>Elapsed Time</i> (minutes)	<i>Predicted</i> <i>Time to Go</i>	<i>Predicted</i> <i>Total Time</i>
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

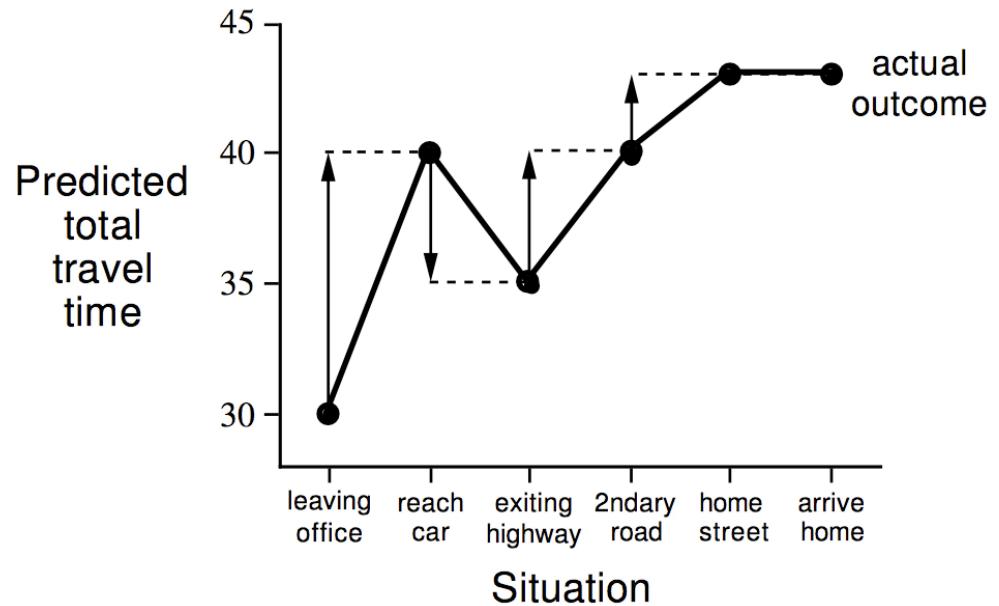
# Driving Home

---

Changes recommended by  
Monte Carlo methods ( $\alpha=1$ )



Changes recommended  
by TD methods ( $\alpha=1$ )

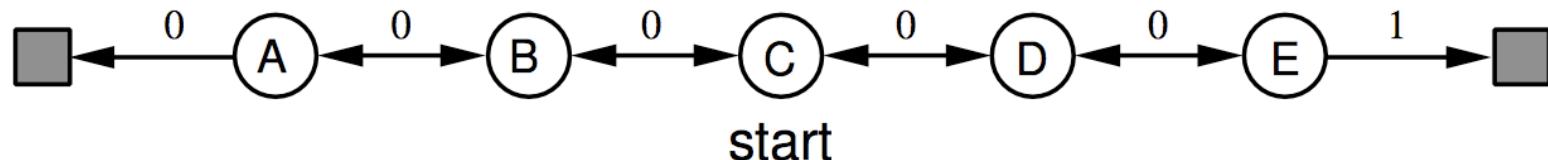


# Advantages of TD Learning

---

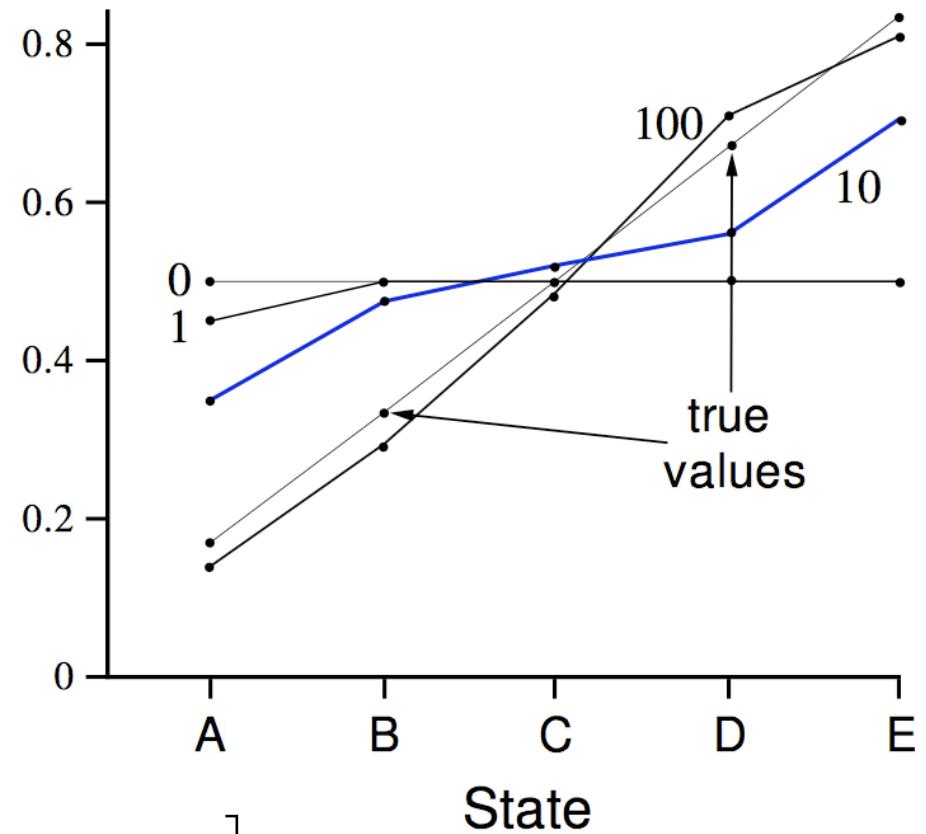
- TD methods do not require a model of the environment, only experience
- TD, but not MC, methods can be fully incremental
  - You can learn **before** knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn **without** the final outcome
    - From incomplete sequences
- Both MC and TD converge, but which is faster?

# Random Walk Example



Estimated  
value

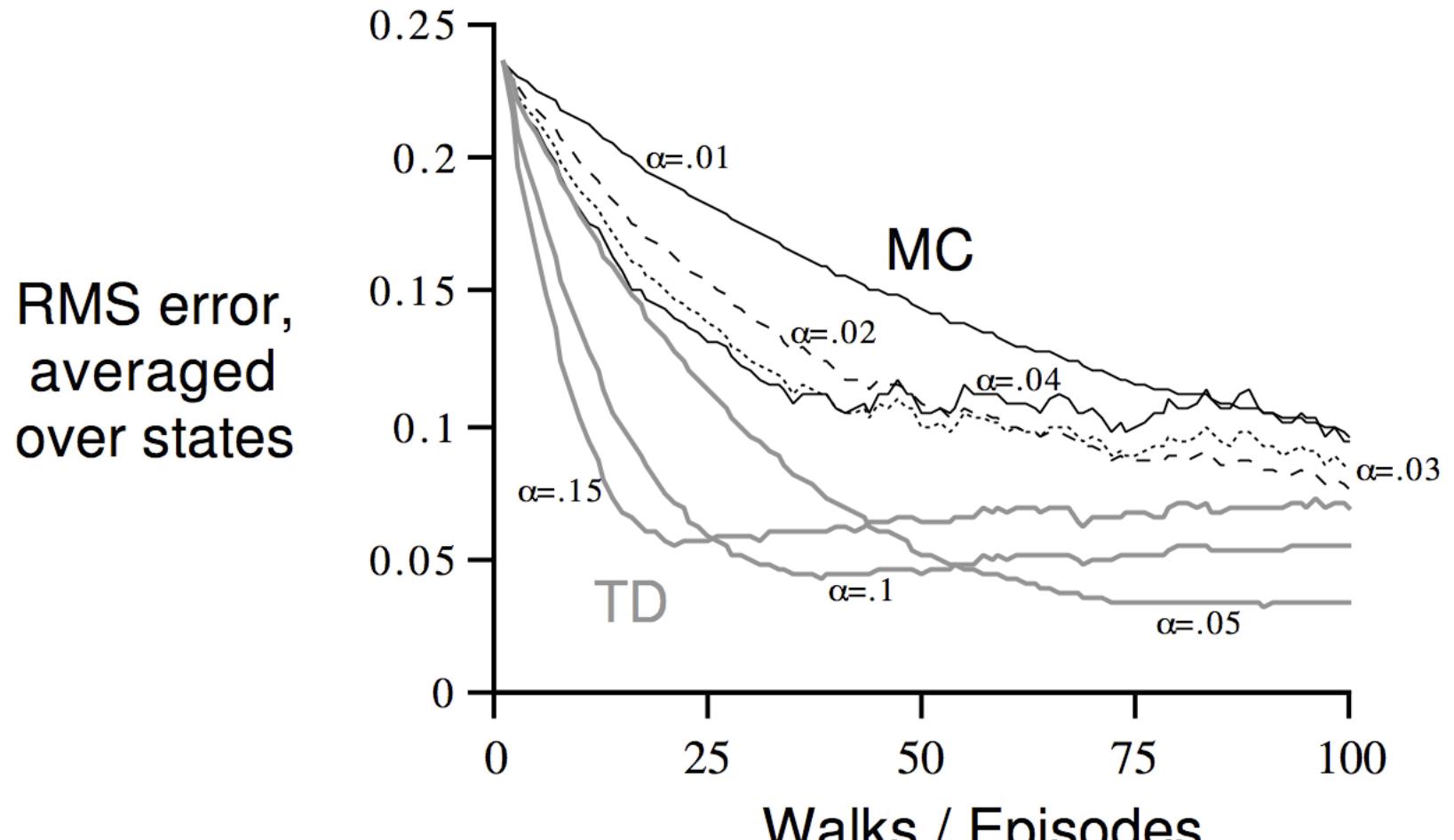
Values learned by TD after  
various numbers of episodes



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

# TD and MC on the Random Walk

---



Data averaged over  
100 sequences of episodes

# Batch Updating in TD and MC methods

---

**Batch Updating:** train completely on a finite amount of data,  
e.g., train repeatedly on 10 episodes until convergence.

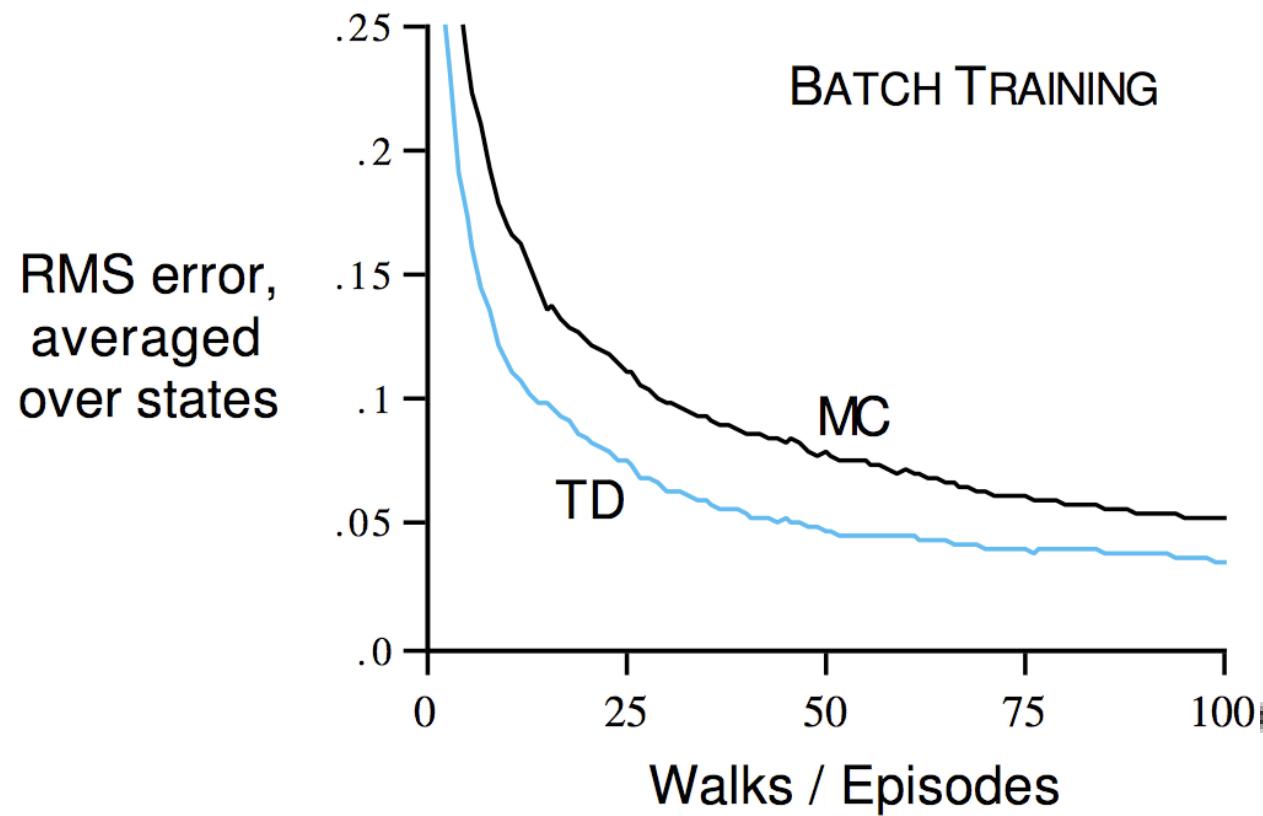
Compute updates according to TD or MC, but only update  
estimates after each complete pass through the data.

For any finite Markov prediction task, under batch updating,  
TD converges for sufficiently small  $\alpha$ .

Constant- $\alpha$  MC also converges under these conditions, **but to**  
**a different answer!**

# Random Walk under Batch Updating

---



After each new episode, all previous episodes were treated as a batch, and algorithm was trained until convergence. All repeated 100 times.

# You are the Predictor

---

Suppose you observe the following 8 episodes:

A, 0, B, 0

B, 1

B, 1

$V(B)$ ? 0.75

B, 1

$V(A)$ ? 0?

B, 1

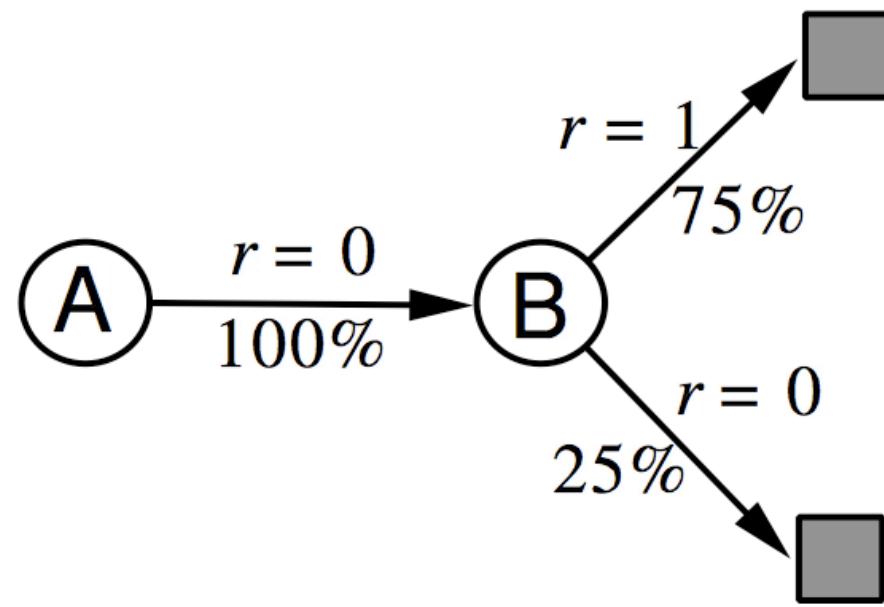
B, 1

B, 0

Assume Markov states, no discounting ( $\gamma = 1$ )

# You are the Predictor

---



$V(A)? \quad 0.75$

# You are the Predictor

---

- The prediction that best matches the training data is  $V(A)=0$ 
  - This **minimizes the mean-square-error** on the training set
  - This is what a batch Monte Carlo method gets
- If we consider the sequentiality of the problem, then we would set  $V(A)=.75$ 
  - This is correct for the **maximum likelihood** estimate of a Markov model generating the data
  - i.e, if we do a best fit Markov model, and assume it is exactly correct, and then compute what it predicts
  - This is called the **certainty-equivalence estimate**
  - This is what TD gets

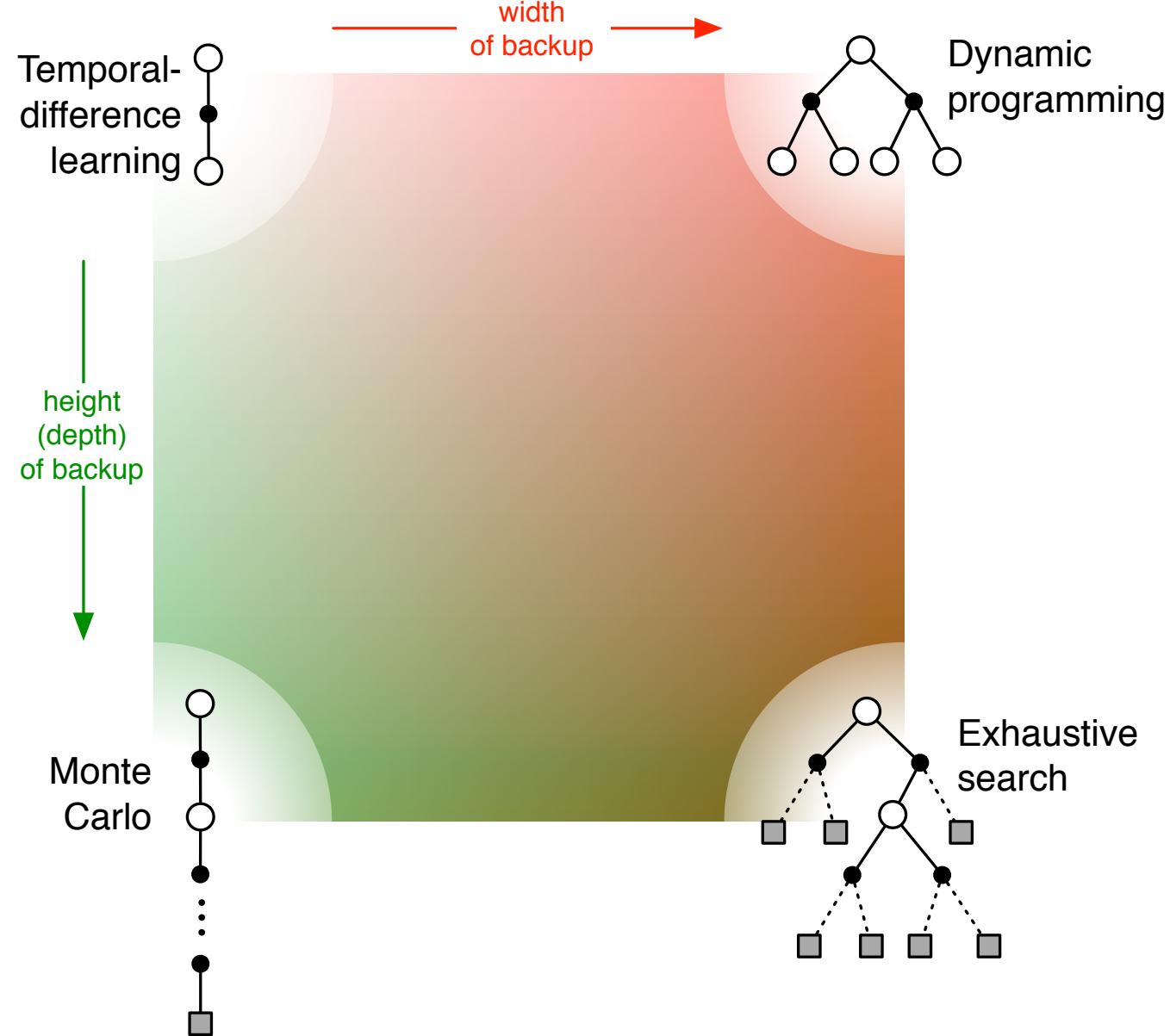
# Summary so far

---

- Introduced *one-step tabular model-free TD methods*
- These methods bootstrap and sample, combining aspects of DP and MC methods
- TD methods are *computationally congenial*
- If the world is truly Markov, then TD methods will learn faster than MC methods
- MC methods have lower error on past (training) data, but higher error on future data

# Unified View

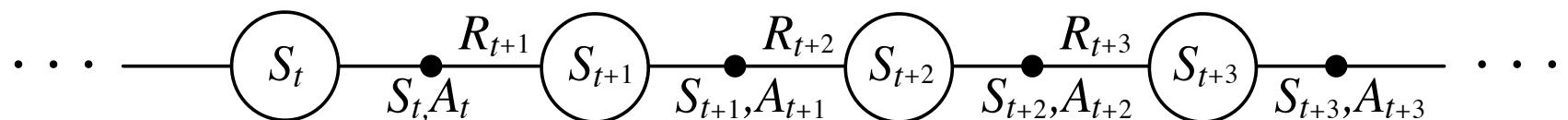
---



# Learning An Action-Value Function To Control

---

Estimate  $q_\pi$  for the current policy  $\pi$



After every transition from a nonterminal state,  $S_t$ , do this:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

If  $S_{t+1}$  is terminal, then define  $Q(S_{t+1}, A_{t+1}) = 0$

# Sarsa: On-Policy TD Control

---

Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , 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.,  $\varepsilon$ -greedy)

    Repeat (for each step of episode):

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

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

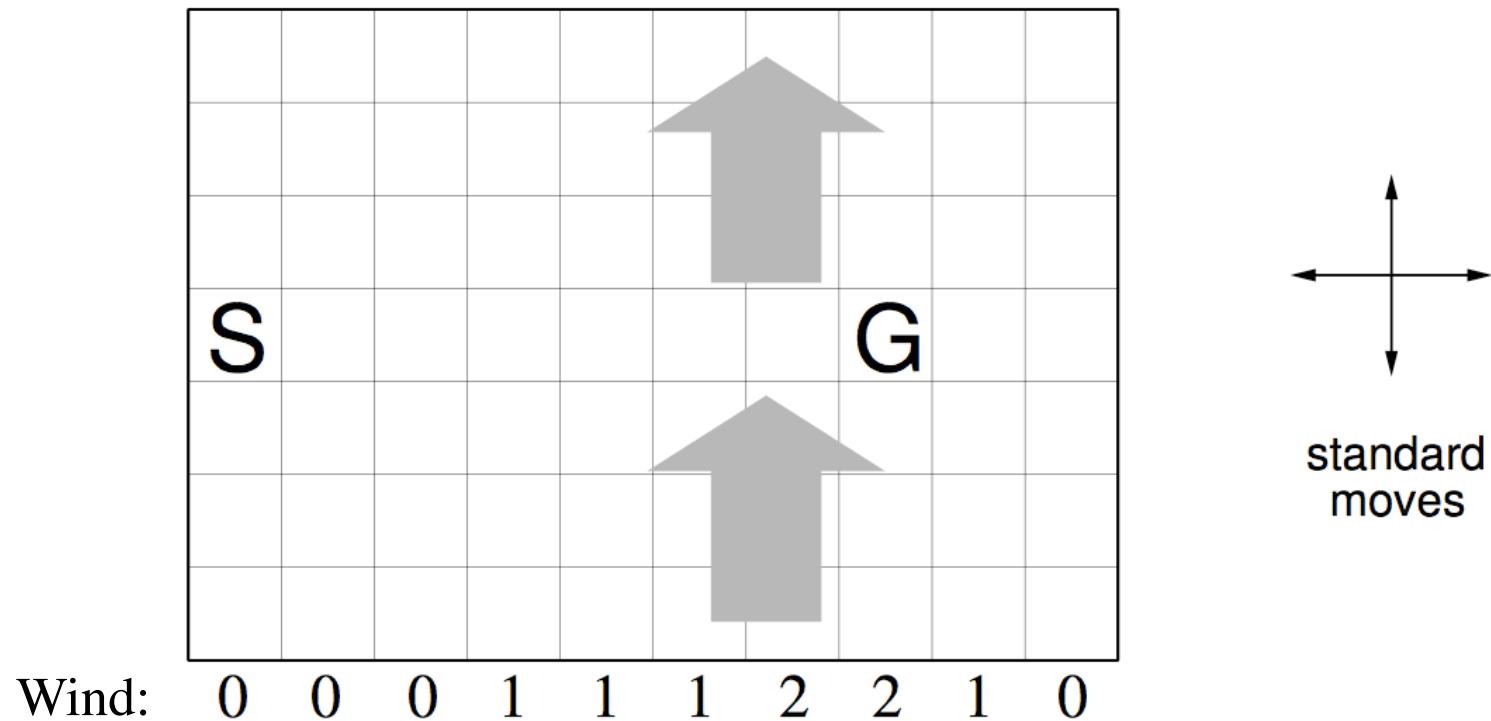
$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$$

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

    until  $S$  is terminal

# Windy Gridworld

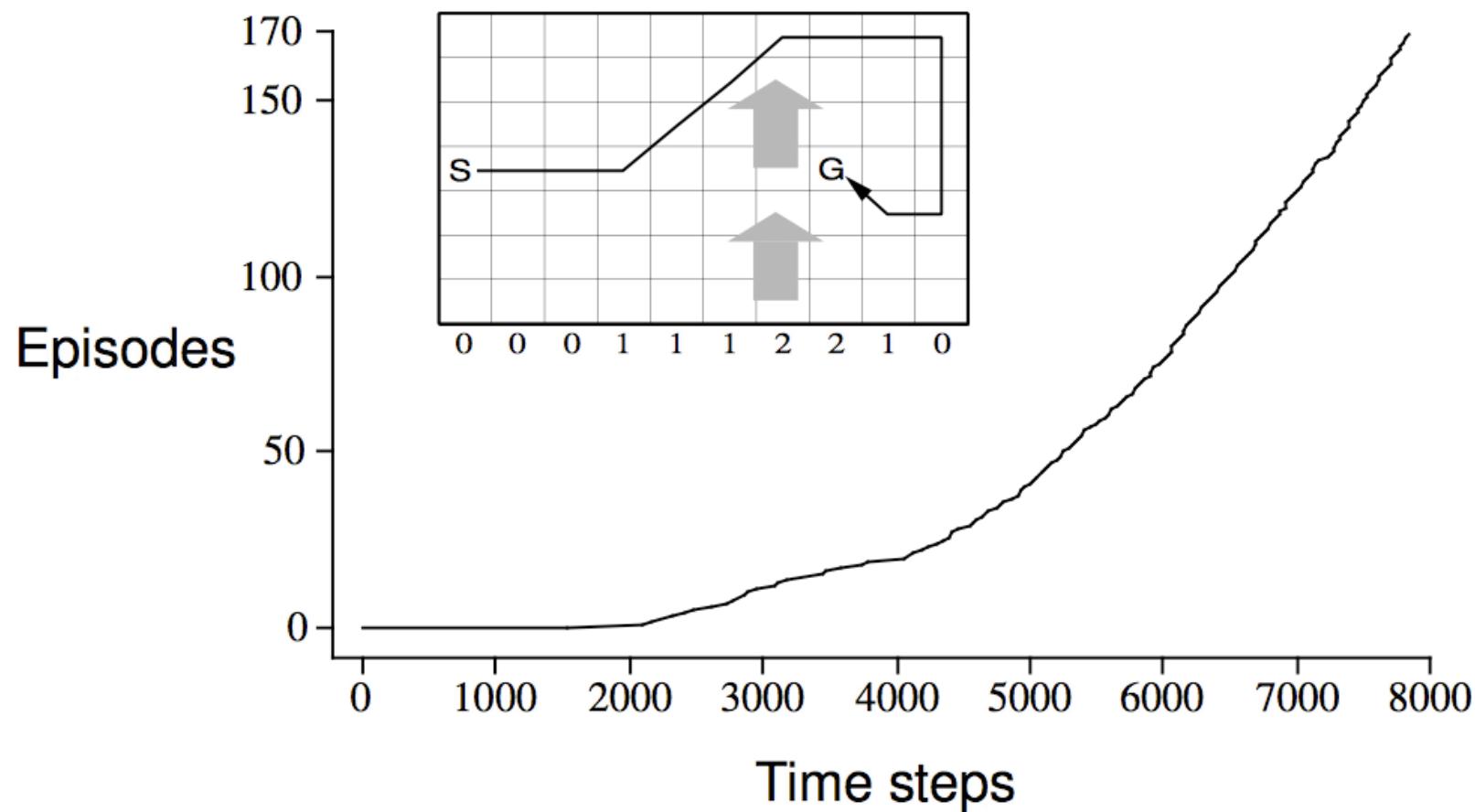
---



undiscounted, episodic, reward =  $-1$  until goal

# Results of Sarsa on the Windy Gridworld

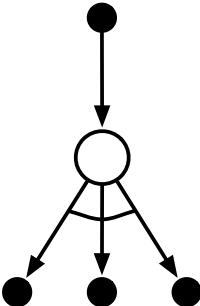
---



# Q-Learning: Off-Policy TD Control

One-step Q-learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$



Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , 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.,  $\varepsilon$ -greedy)

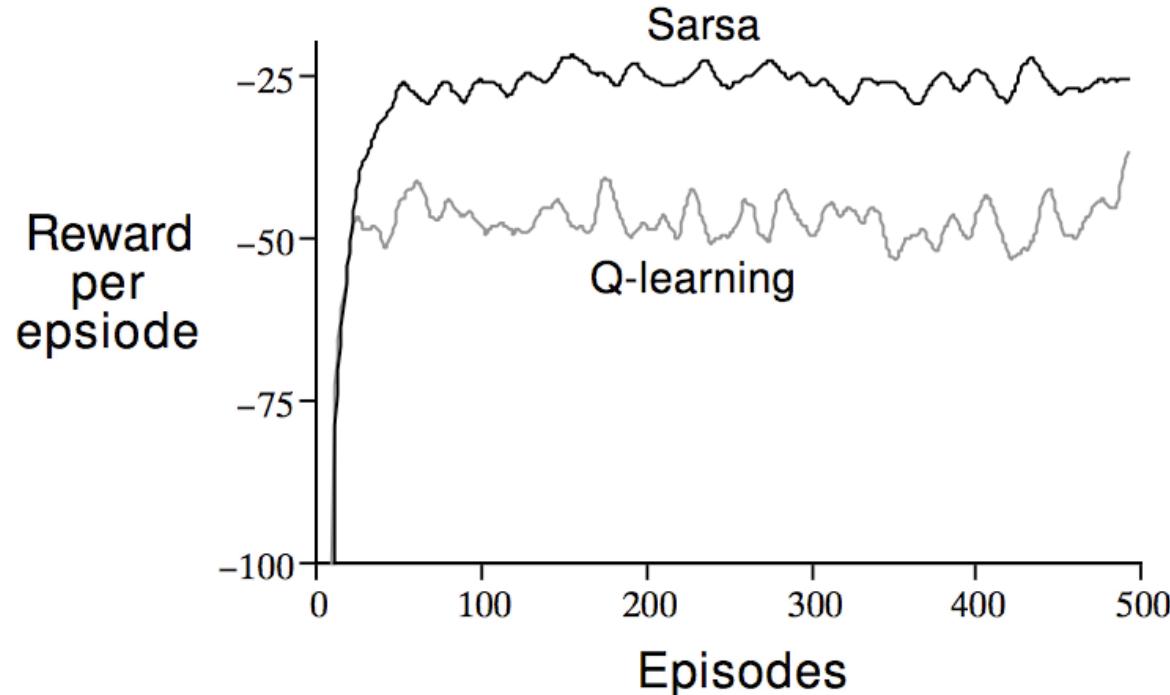
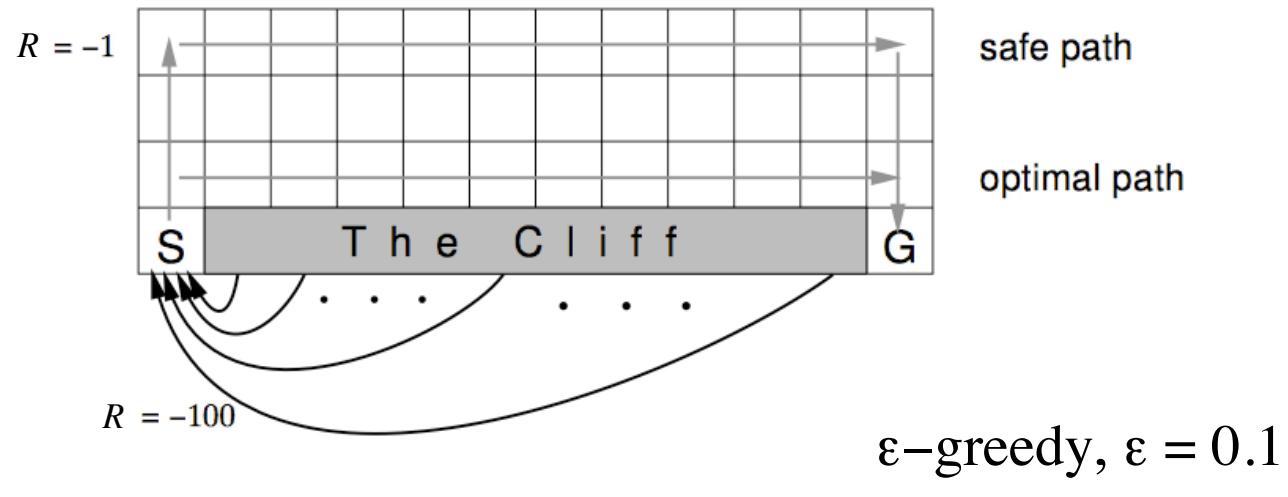
        Take action  $A$ , observe  $R, S'$

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

$S \leftarrow S'$ ;

    until  $S$  is terminal

# Cliffwalking

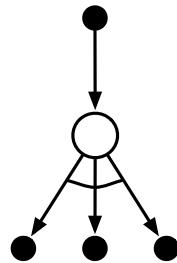


# Expected Sarsa

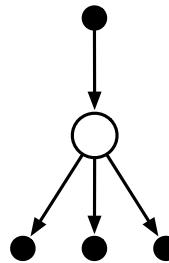
---

- Instead of the *sample* value-of-next-state, use the expectation!

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \right] \\ &\leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right] \end{aligned}$$



Q-learning

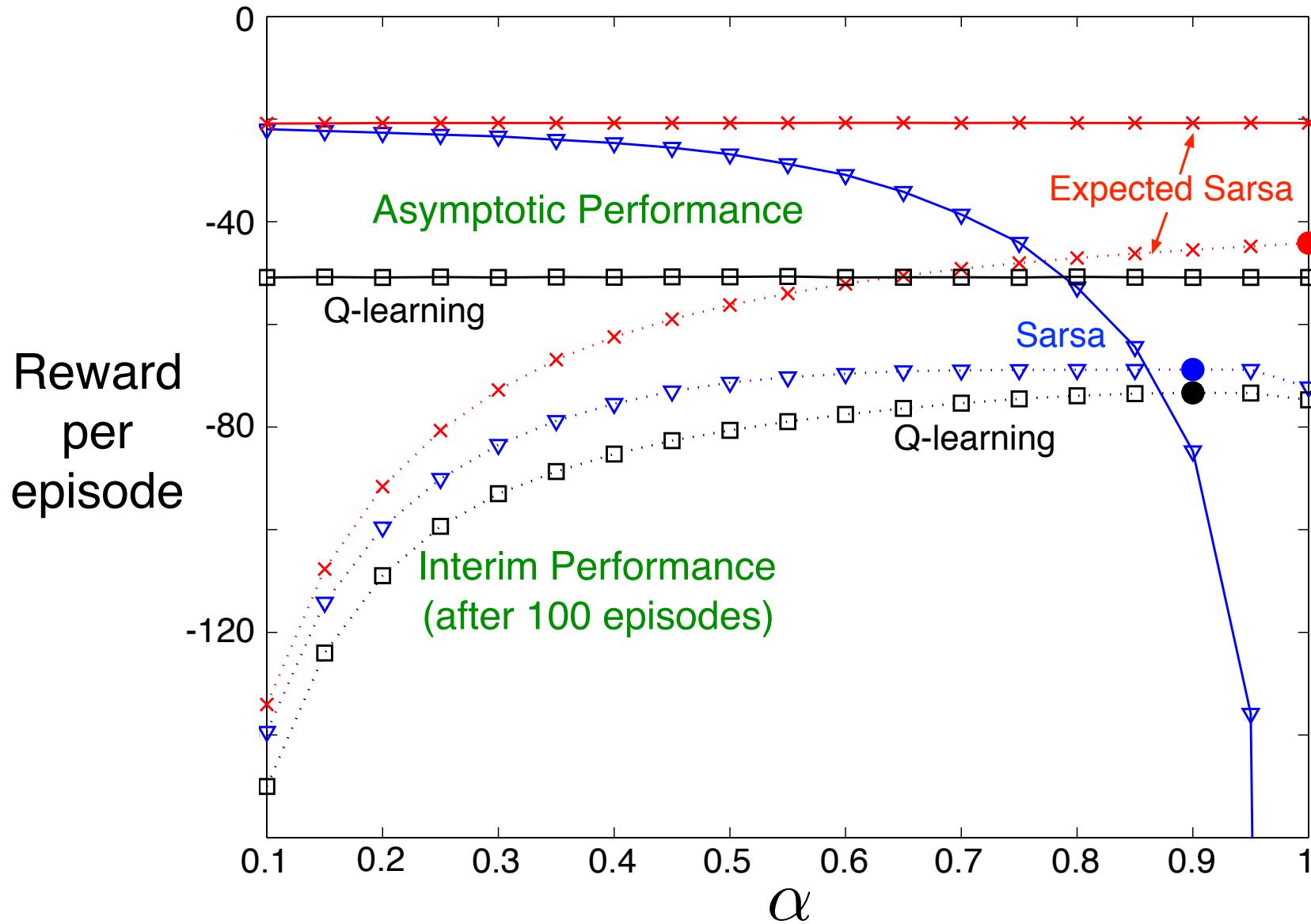


Expected Sarsa

- Expected Sarsa's performs better than Sarsa (but costs more)

# Performance on the Cliff-walking Task

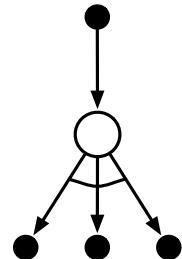
---



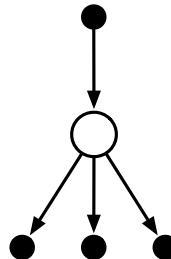
# *Off-policy* Expected Sarsa

- Expected Sarsa generalizes to arbitrary behavior policies  $\mu$ 
  - in which case it includes Q-learning as the special case in which  $\pi$  is the greedy policy

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \right] \\ &\leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t) \right] \end{aligned}$$

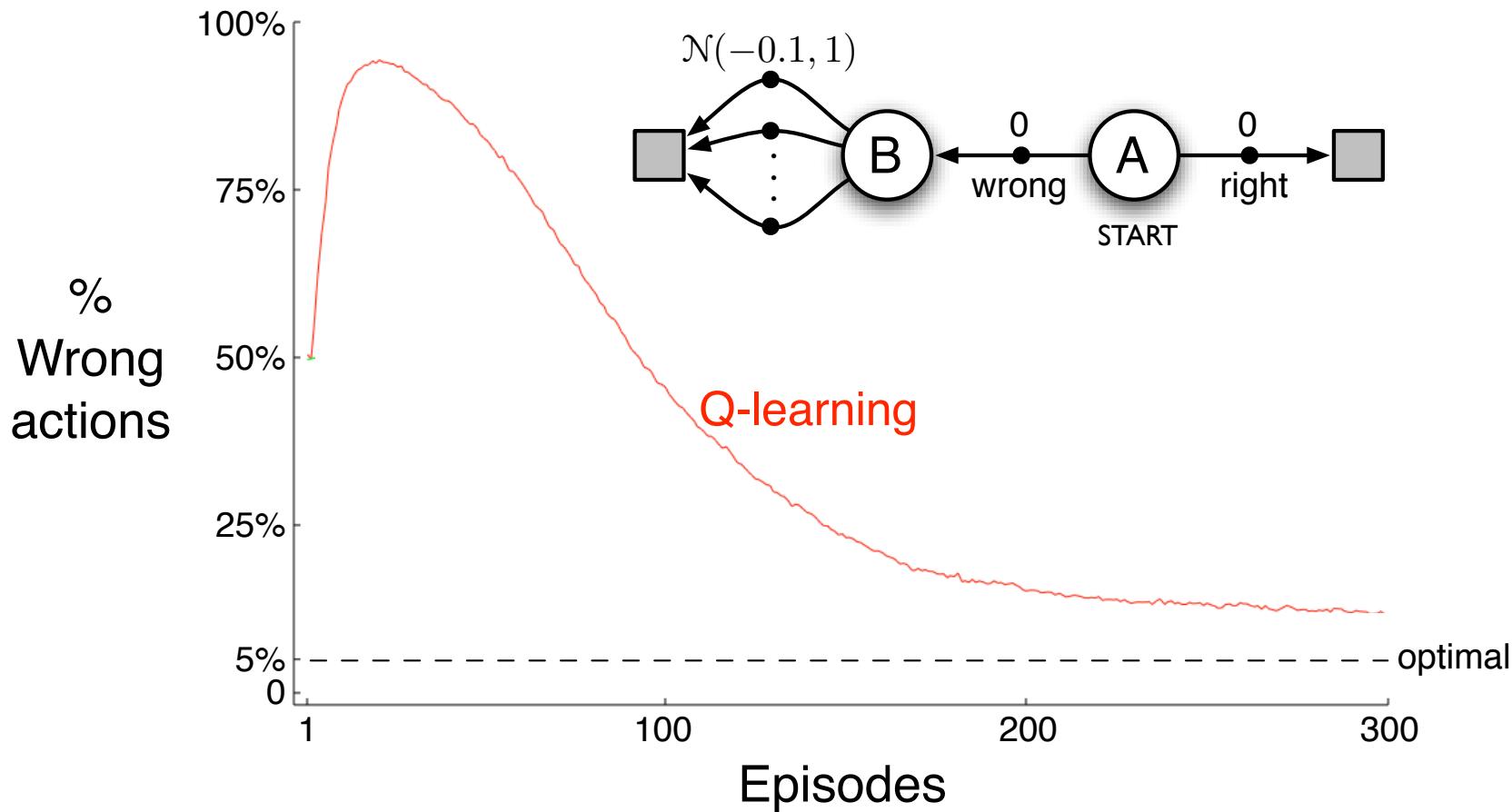


Q-learning



Expected Sarsa

# Maximization Bias



**Tabular Q-learning:** 
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

# Double Q-Learning

---

- Train 2 action-value functions,  $Q_1$  and  $Q_2$
- Do Q-learning on both, but
  - never on the same time steps ( $Q_1$  and  $Q_2$  are indep.)
  - pick  $Q_1$  or  $Q_2$  at random to be updated on each step
- If updating  $Q_1$ , use  $Q_2$  for the value of the next state:
$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left( R_{t+1} + Q_2(S_{t+1}, \arg \max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right)$$
- Action selections are (say)  $\varepsilon$ -greedy with respect to the sum of  $Q_1$  and  $Q_2$

# Double Q-Learning: Code

---

Initialize  $Q_1(s, a)$  and  $Q_2(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily

Initialize  $Q_1(\text{terminal-state}, \cdot) = Q_2(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

    Initialize  $S$

    Repeat (for each step of episode):

        Choose  $A$  from  $S$  using policy derived from  $Q_1$  and  $Q_2$  (e.g.,  $\varepsilon$ -greedy in  $Q_1 + Q_2$ )

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

        With 0.5 probability:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha \left( R + \gamma Q_2(S', \arg \max_a Q_1(S', a)) - Q_1(S, A) \right)$$

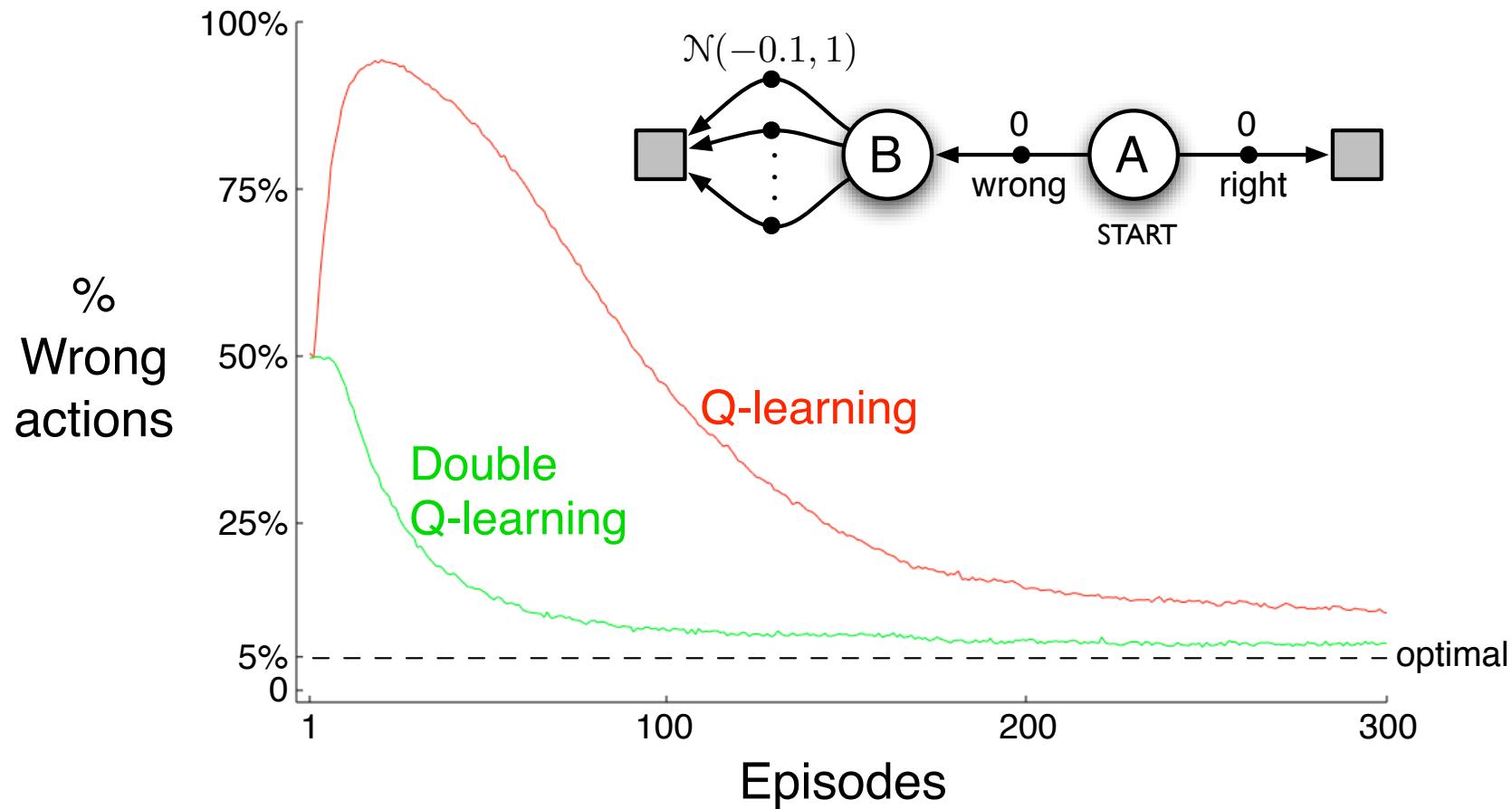
        else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha \left( R + \gamma Q_1(S', \arg \max_a Q_2(S', a)) - Q_2(S, A) \right)$$

$S \leftarrow S'$ ;

    until  $S$  is terminal

# Example of Maximization Bias



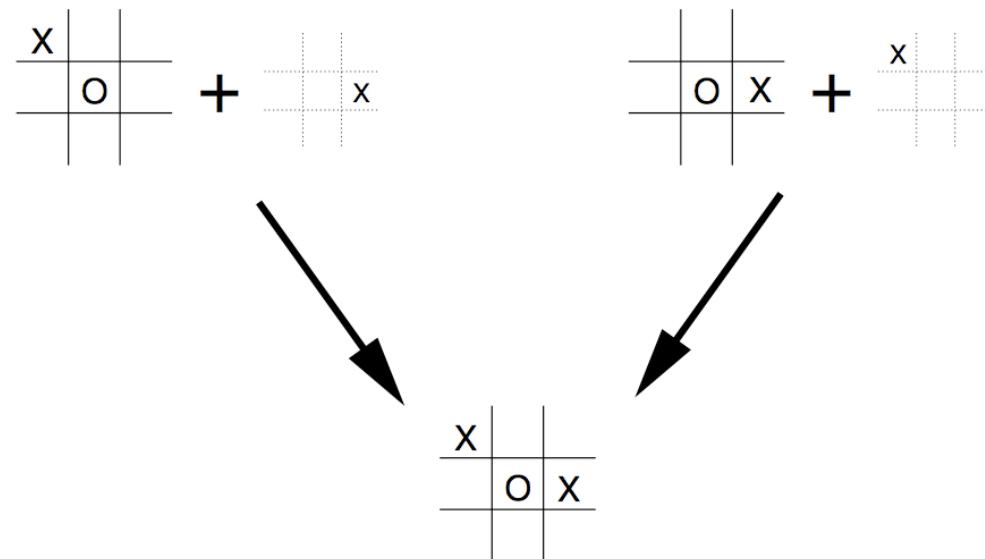
**Double Q-learning:**

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma Q_2(S_{t+1}, \arg \max_a Q_1(S_{t+1}, a)) - Q_1(S_t, A_t) \right]$$

# Afterstates

---

- Usually, a state-value function evaluates states in which the agent can take an action.
- But sometimes it is useful to evaluate states **after** agent has acted, as in tic-tac-toe.
- Why is this useful?



# Summary

---

- Introduced *one-step tabular model-free TD methods*
- These methods bootstrap and sample, combining aspects of DP and MC methods
- TD methods are *computationally congenial*
- If the world is truly Markov, then TD methods will learn faster than MC methods
- MC methods have lower error on past data, but higher error on future data
- Extend prediction to control by employing some form of GPI
  - On-policy control: **Sarsa, Expected Sarsa**
  - Off-policy control: **Q-learning, Expected Sarsa**
- Avoiding maximization bias with Double Q-learning