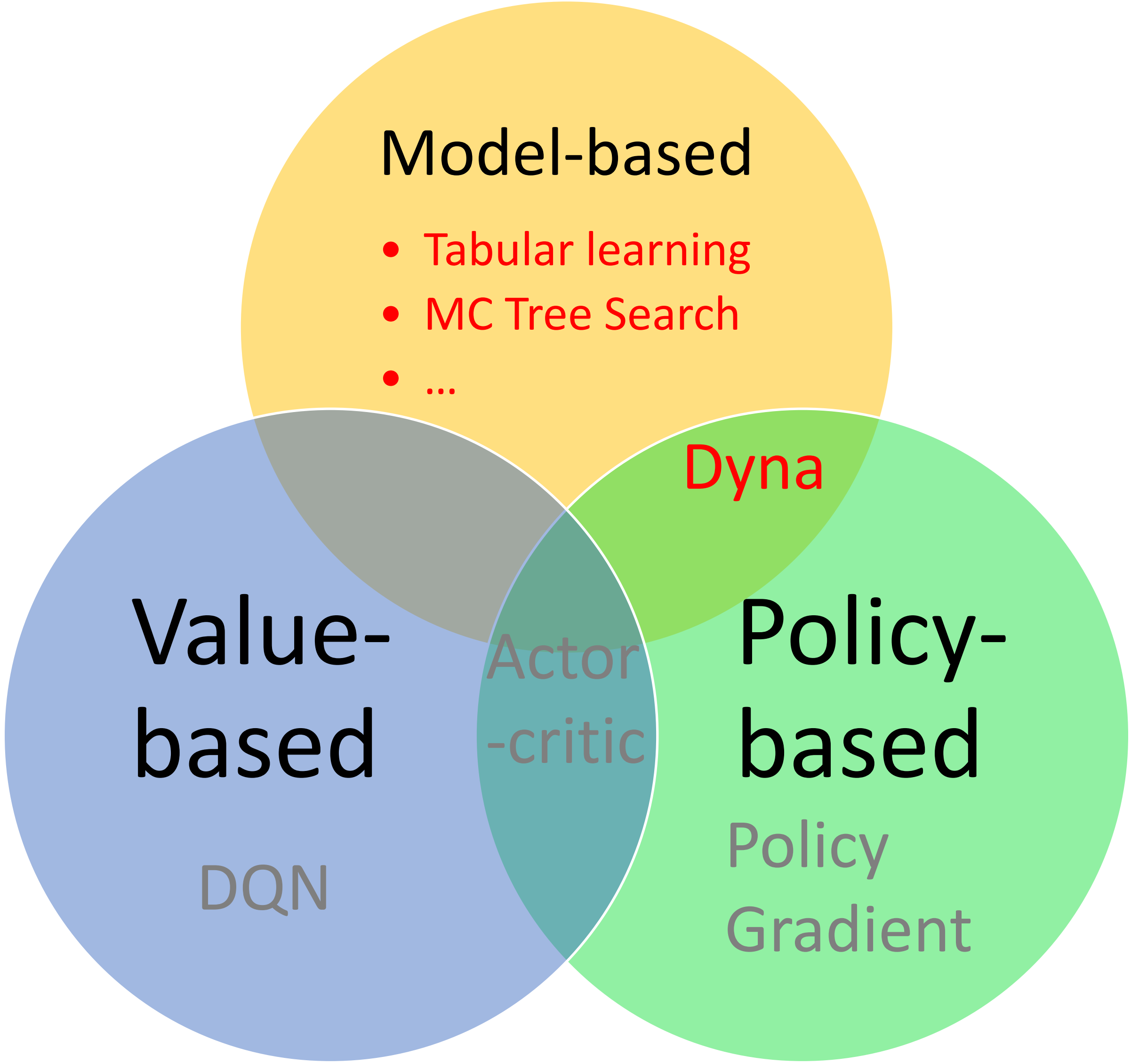
Planning and Learning

Prof. Kuan-Ting Lai

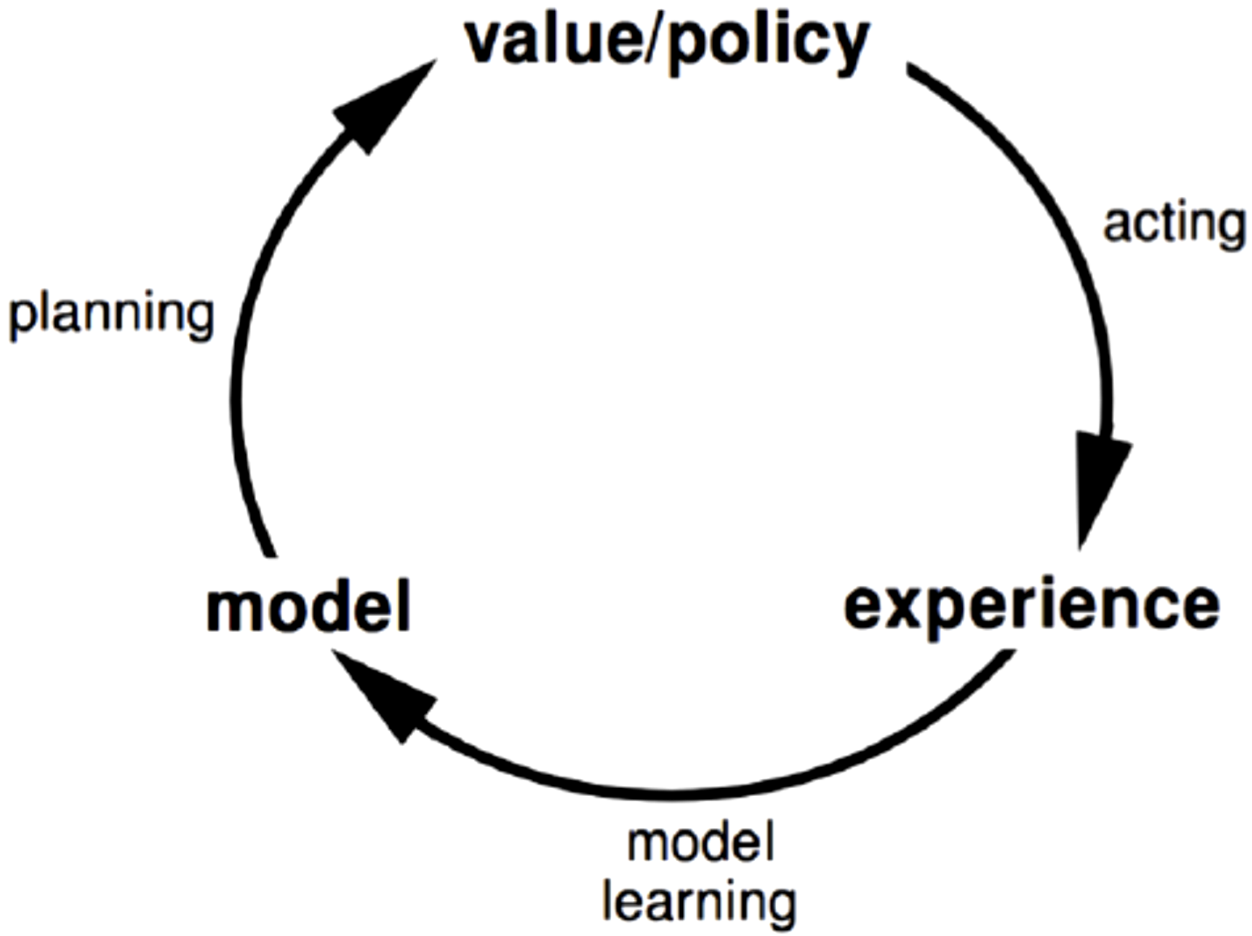
2020/5/22

Model-based Learning

- Learn a model from experience
- Plan value function (and/or policy) from model



Model-based RL



Pros and Cons of Model-based RL

Advantages

- Learn model efficiently by supervised learning methods
- Can reason about model uncertainty

Disadvantages

- First learn a model, then construct a value function
  - Two sources of approximation error

What is a Model?

- Representation of an MDP (S, A, P, R) parameterized by η
- Assume state space S and action space A are known
- A model $M = (P_\eta, R_\eta)$ represents state transitions and rewards:

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} \mid S_t, A_t)$$
$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} \mid S_t, A_t)$$

- Assume conditional independence between state transitions and rewards

$$\mathbb{P}[S_{t+1}, R_{t+1} \mid S_t, A_t] = \mathbb{P}[S_{t+1} \mid S_t, A_t]\,\mathbb{P}[R_{t+1} \mid S_t, A_t]$$

Model Learning

- Goal: estimate model M_η from experience $\{S_1, A_1, R_2, ..., S_T\}$
- A Supervised learning problem!

$$
\begin{gathered}
S_1, A_1 \to R_2, S_2 \\
S_2, A_2 \to R_3, S_3 \\
\vdots \\
S_{T-1}, A_{T-1} \to R_T, S_T
\end{gathered}
$$

- Learning $s, a \to r$ is a regression problem
- Learning $s, a \to s'$ is a density estimation problem

Examples of Models

- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Gaussian Process Model
- Deep Belief Network Model
- ...

Table Lookup Model

- Count visits $\mathrm{N}(\mathrm{s}, a)$ to each state action pair

$$\hat{\mathcal{P}}^a_{s,s'} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t, S_{t+1} = s, a, s')$$

$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t = s, a) R_t$$

- Another method: Sample Memory
  - Record experience tuple: $(S_t, A_t, R_{t+1}, S_{t+1})$
  - To sample model, randomly pick tuple matching $(S_t, A_t, \cdot, \cdot)$

AB Example

- Two states A;B; no discounting; 8 episodes of experience

A, 0, B, 0

B, 1

B, 1

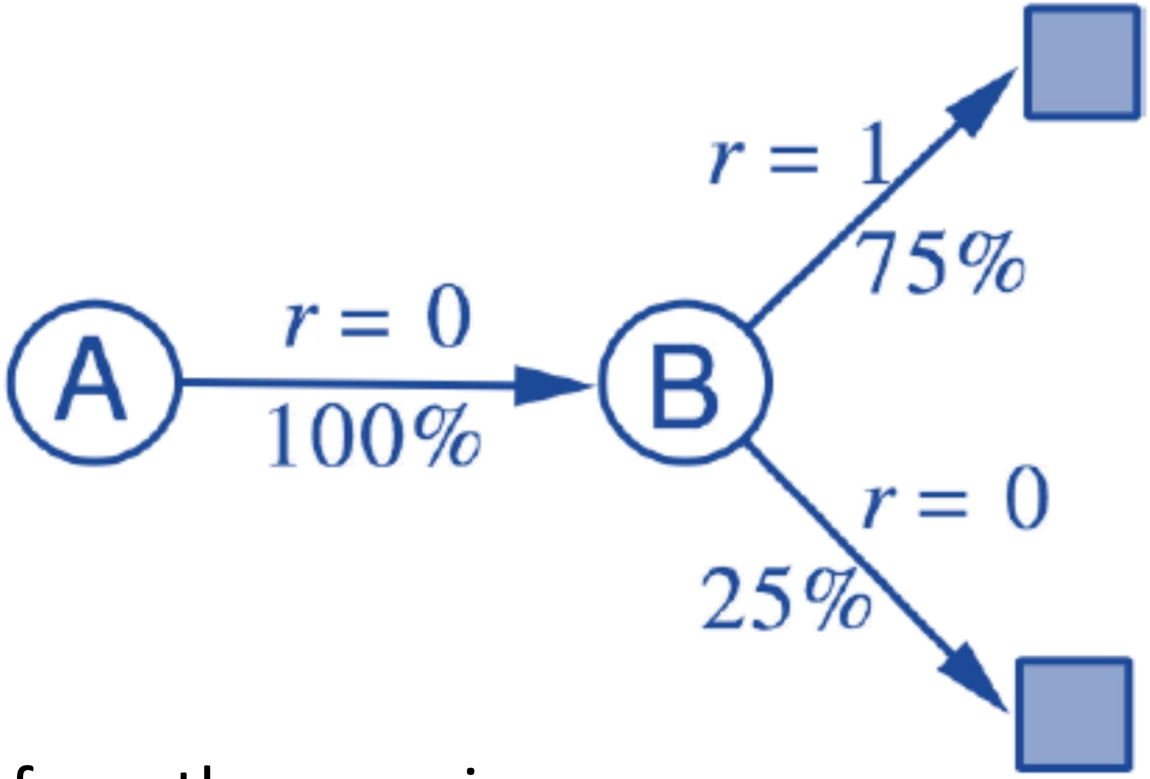
B, 1

B, 1

B, 1

B, 1

B, 0



Construct a table lookup model from the experience

Sample-Based Planning

- Use the model only to generate samples
- Sample experience from model

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} \mid S_t, A_t)$$
$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} \mid S_t, A_t)$$

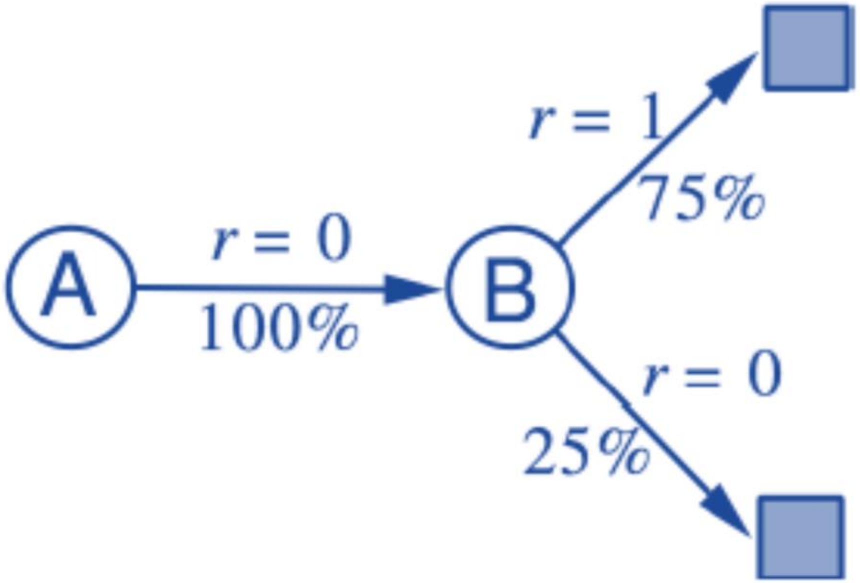
- Apply model-free RL to samples
  - Monte-Carlo control
  - Sarsa
  - Q-learning
- Sample-based planning methods are often more e cient

Back to the AB Example

- Construct a table-lookup model from real experience
- Apply model-free RL to sampled experience

Real experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



Sampled experience

B, 1
B, 0
B, 1
A, 0, B, 1
B, 1
A, 0, B, 1
B, 1
B, 0

e.g. Monte-Carlo learning: $V(A) = 1, V(B) = 0.75$

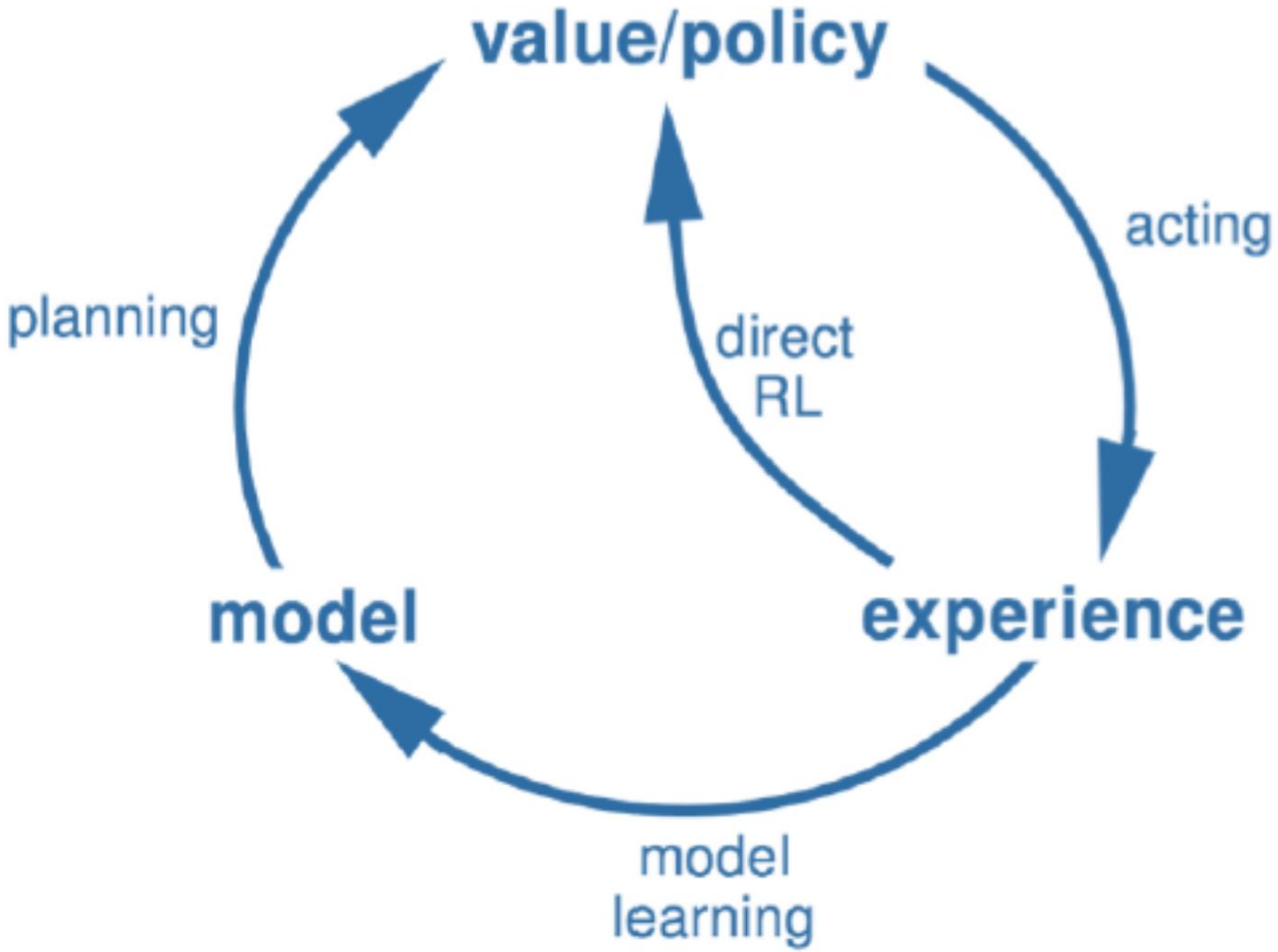
Planning with an Inaccurate Model

- Model-based RL is only as good as the estimated model
- When the model is inaccurate, planning process will compute a suboptimal policy
    1. when model is wrong, use model-free RL
    2. reason explicitly about model uncertainty

Integrating Learning and Planning

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from real experience
- Model-Based RL (using Sample-Based Planning)
  - Learn a model from real experience
  - Plan value function (and/or policy) from simulated experience
- Dyna
  - Learn a model from real experience
  - Learn and plan value function (and/or policy) from real and simulated experience

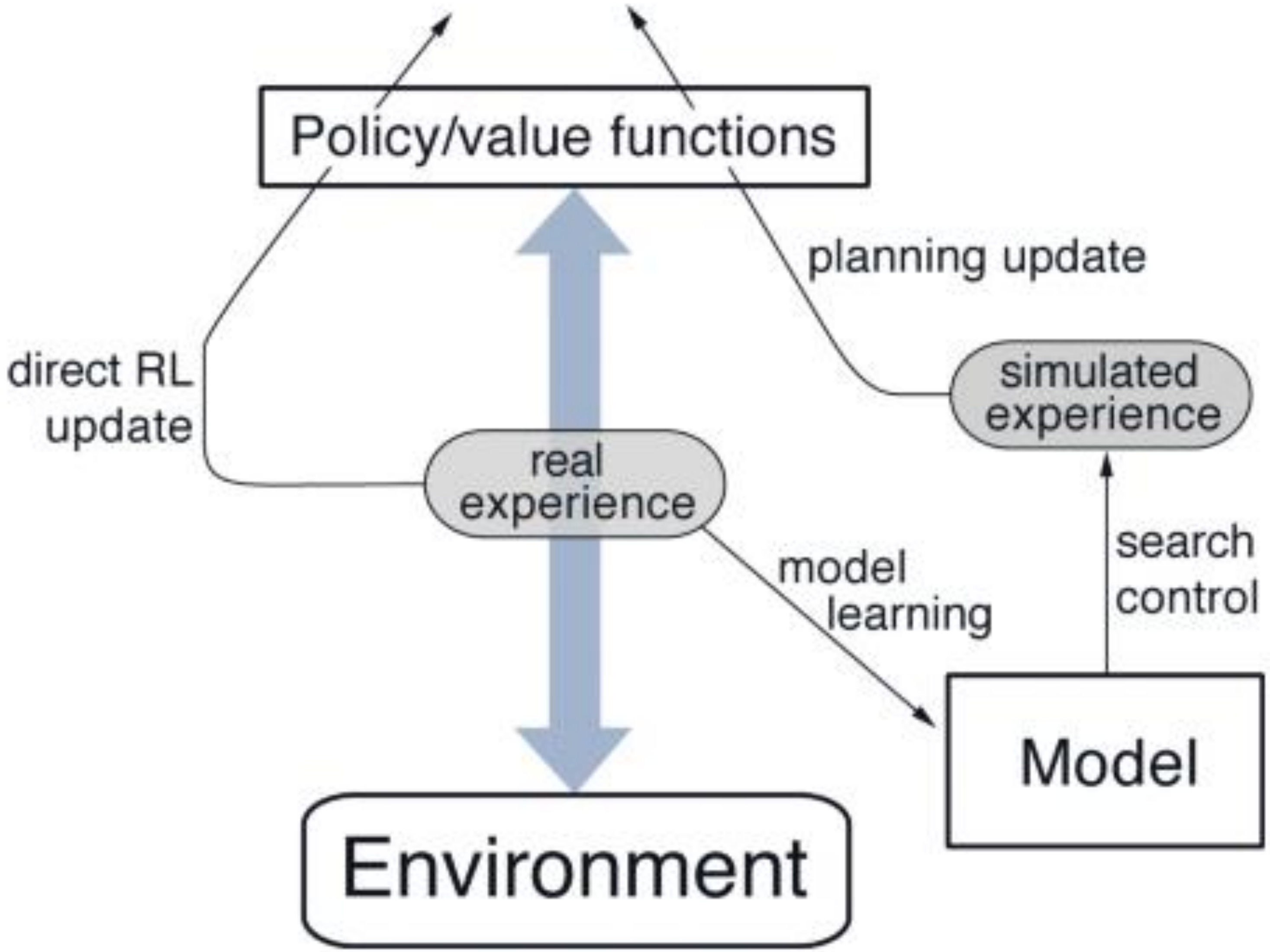
Dyna: Integrated Planning, Acting and Learning



Dyna-Q Algorithm

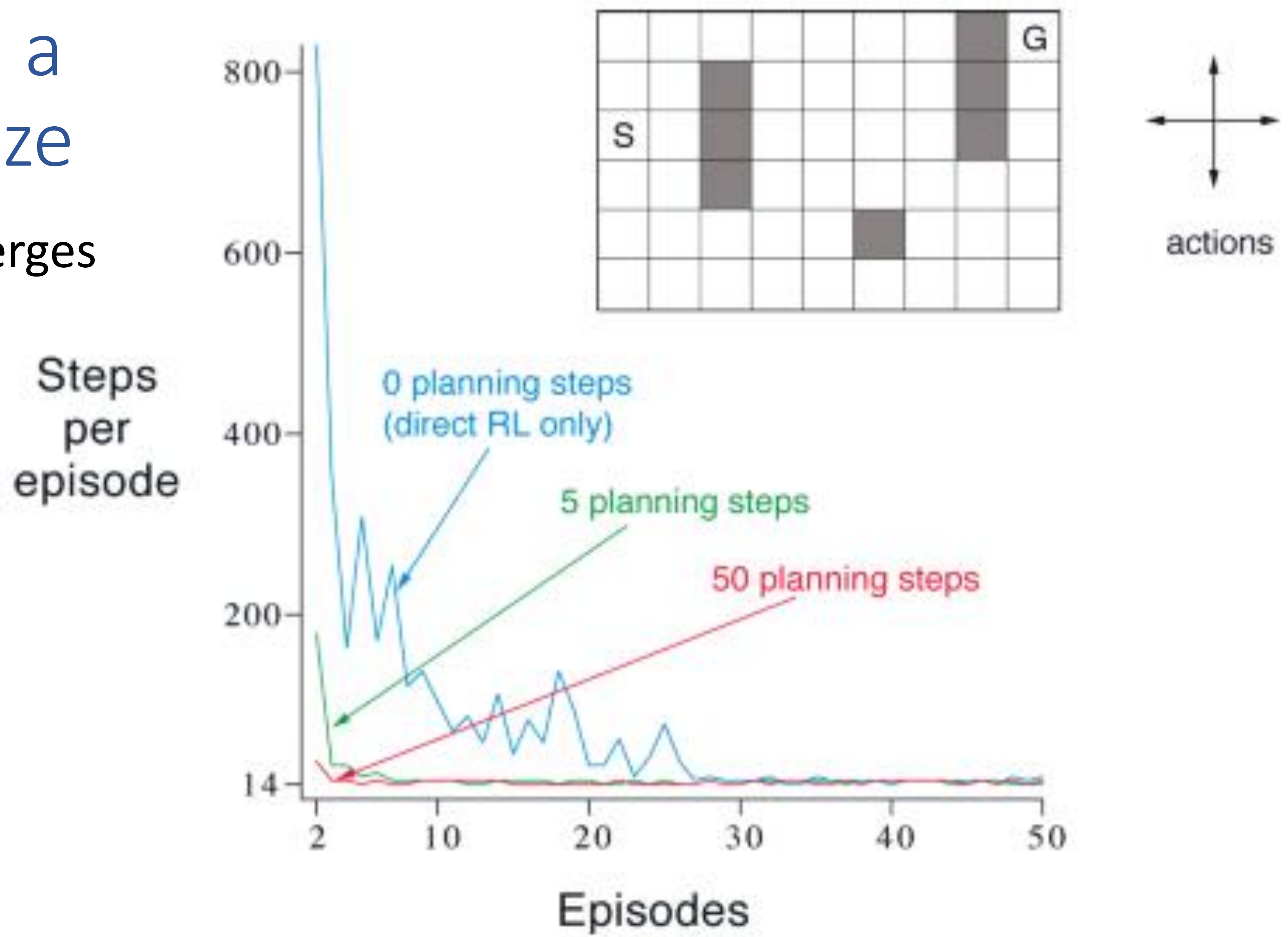
Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
 (a) $S \leftarrow$ current (nonterminal) state
 (b) $A \leftarrow \varepsilon\text{-greedy}(S, Q)$
 (c) Execute action A; observe resultant reward, R, and state, S'
 (d) $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
 (e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
 (f) Repeat n times:
  $S \leftarrow$ random previously observed state
  $A \leftarrow$ random action previously taken in S
  $R, S' \leftarrow Model(S, A)$
  $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$

Dyna Architecture

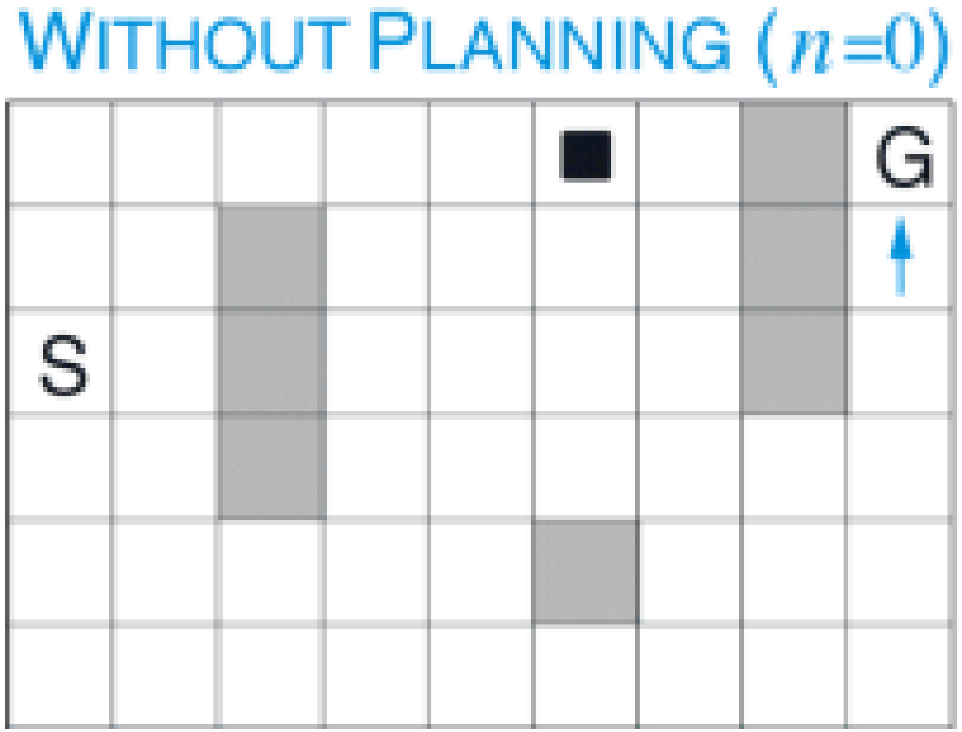


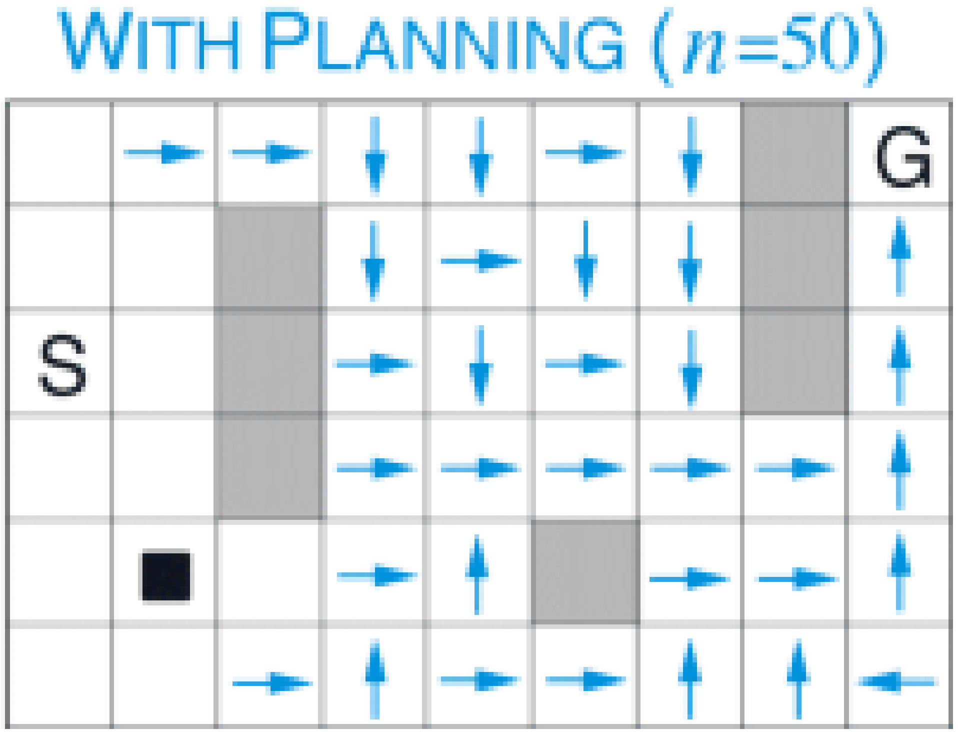
Dyna-Q on a Simple Maze

- Planning converges faster



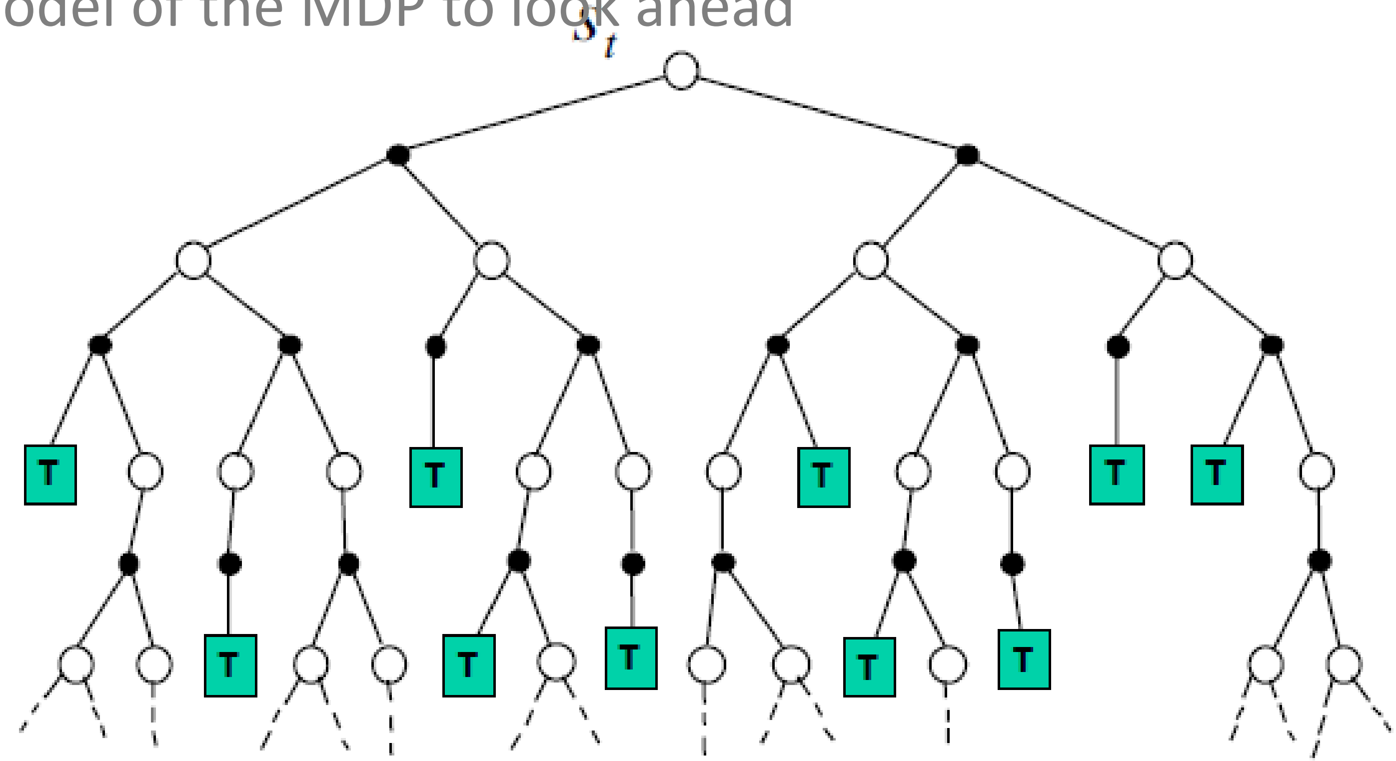
Policies Learned by Non-Planning & Planning





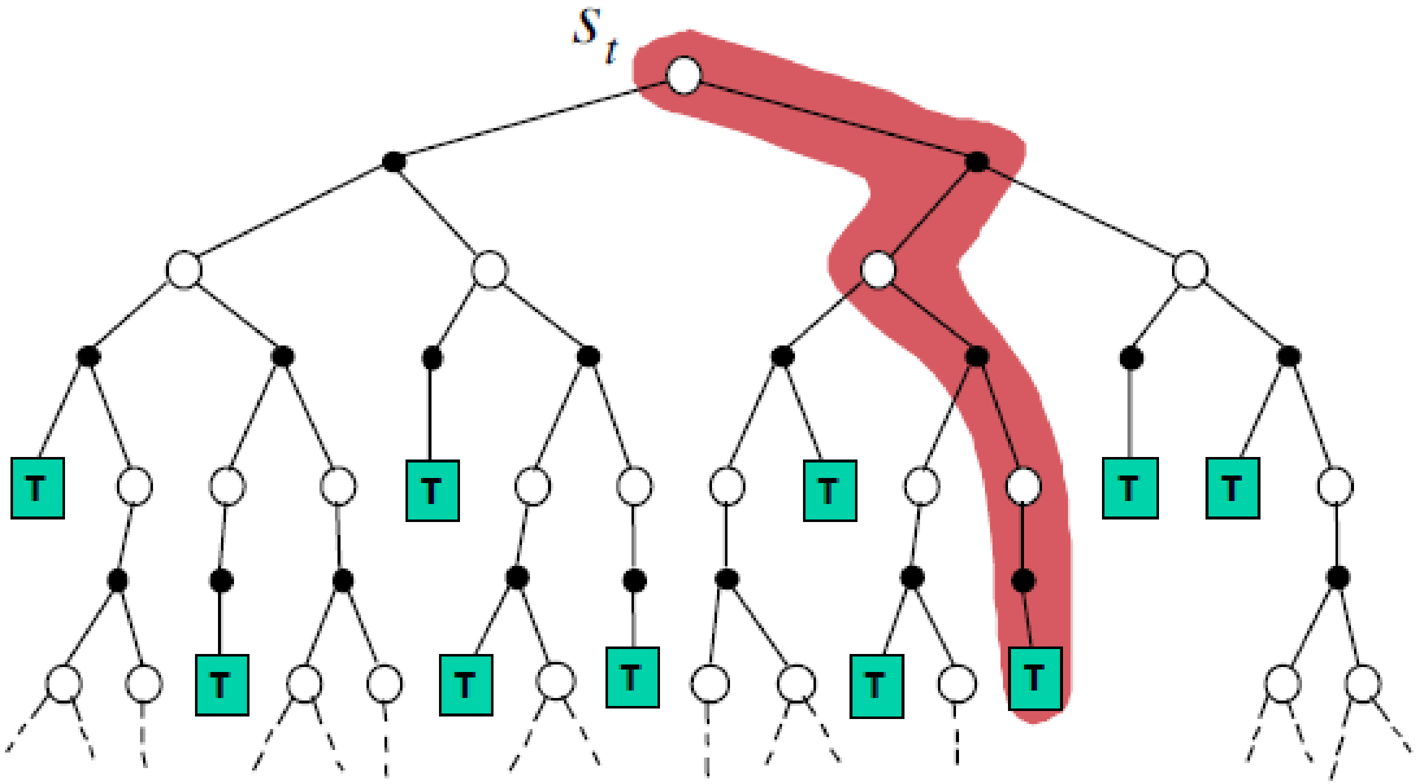
Simulated-based Search

- Forward search
  - Select the best action by lookahead
  - Build a search tree with the current state s_t at the root
  - Using a model of the MDP to look ahead



Simulated-based Search

- Forward search paradigm using sample-based planning
- Simulate episodes of experience from now with the model
- Apply model-free RL to simulated episodes



Model-Carlo Tree Search (Evaluation)

- Given a model $\mathcal{M}_\nu$
- Simulate K episodes from current state s_t using current simulation policy π

$$\{s_t, A_t^k, R_{t+1}^k, S_{t+1}^k, ..., S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$$

- Build a search tree containing visited states and actions
- Evaluate states $Q(s,a)$ by mean return of episodes from s, a

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{u=t}^{T} \mathbf{1}(S_u, A_u = s, a) G_u \xrightarrow{P} q_\pi(s,a)$$

- After search is finished, select current (real) action with maximum value in search tree

$$a_t = \operatorname*{argmax}_{a \in \mathcal{A}} Q(s_t, a)$$

Monte-Carlo Tree Search (Simulation)

- In MCTS, the simulation policy π improves
- Each simulation consists of two phases (in-tree, out-of-tree)
  - Tree policy (improves): pick actions to maximise $Q(S, A)$
  - Default policy (fixed): pick actions randomly
- Repeat (each simulation)
  - Evaluate states $Q(S, A)$ by Monte-Carlo evaluation
  - Improve tree policy, e.g. by $\epsilon - \text{greedy}(Q)$
- Monte-Carlo control applied to simulated experience
- Converges on the optimal search tree, $Q(S, A) \rightarrow q_*(S, A)$

Case Study: The Game of Go

- The ancient oriental game of Go is 2500 years old
- Considered to be the hardest classic board game
- A grand challenge task for AI
- Traditional game-tree search has failed in Go

Position Evaluation in Go

- How good is a position s?
- Reward function (undiscounted):

$$R_t = 0 \text{ for all non-terminal steps } t < T$$

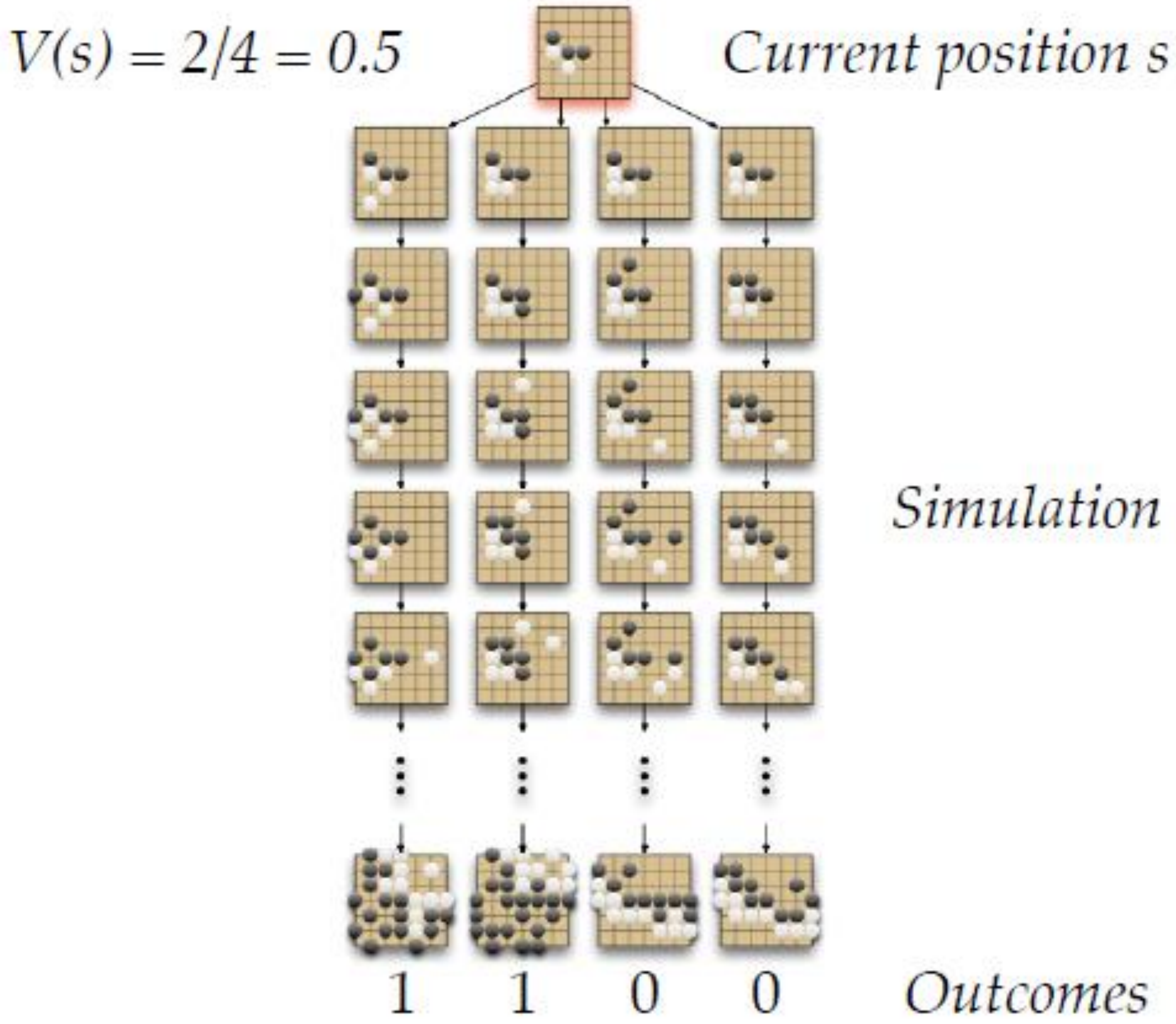
$$R_T = \begin{cases} 1 & \text{if Black wins} \\ 0 & \text{if White wins} \end{cases}$$

- Policy $\pi = \langle \pi_B, \pi_W \rangle$ selects moves for both players
- Value function (how good is position s):

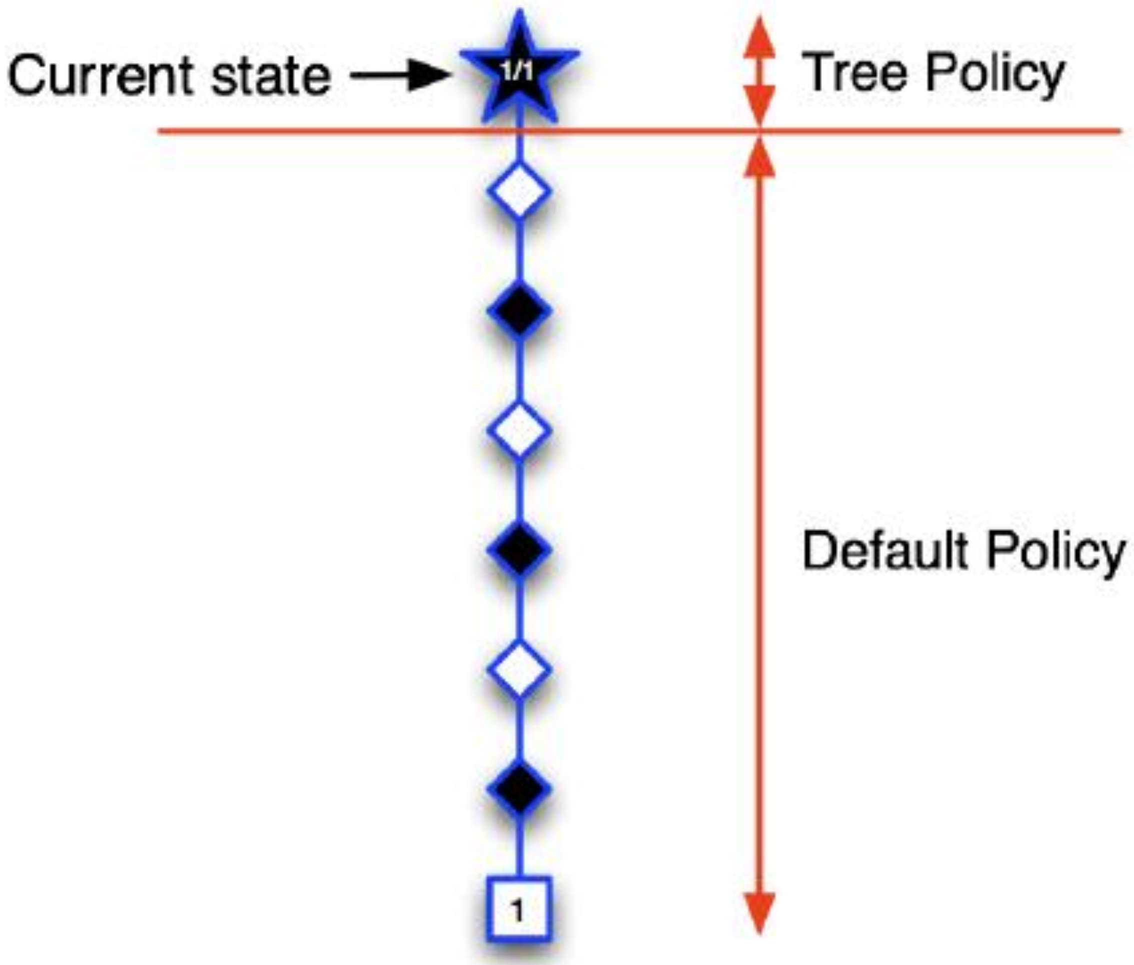
$$v_\pi(s) = \mathbb{E}_\pi \left[R_T \mid S = s \right] = \mathbb{P} \left[\text{Black wins} \mid S = s \right]$$

$$v_*(s) = \max_{\pi_B} \min_{\pi_W} v_\pi(s)$$

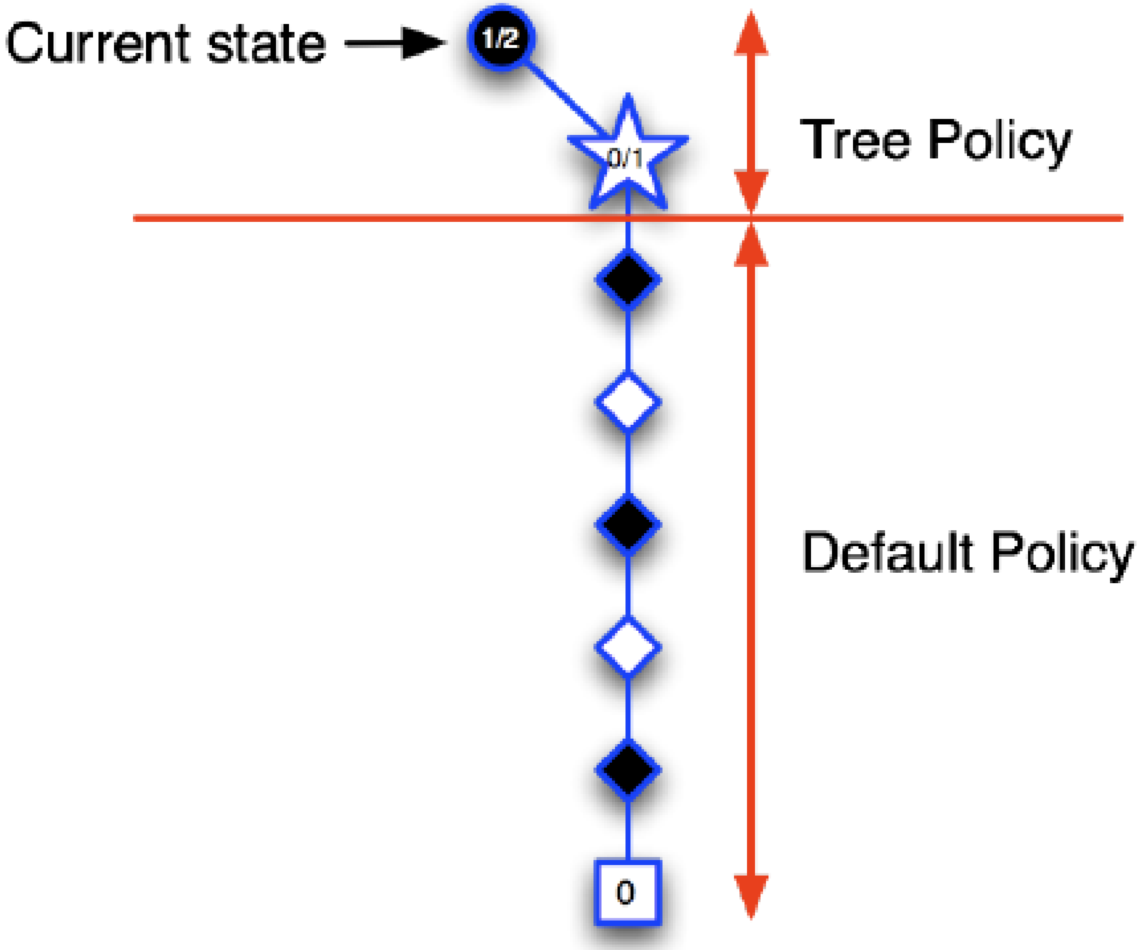
Monte-Carlo Evaluation in Go



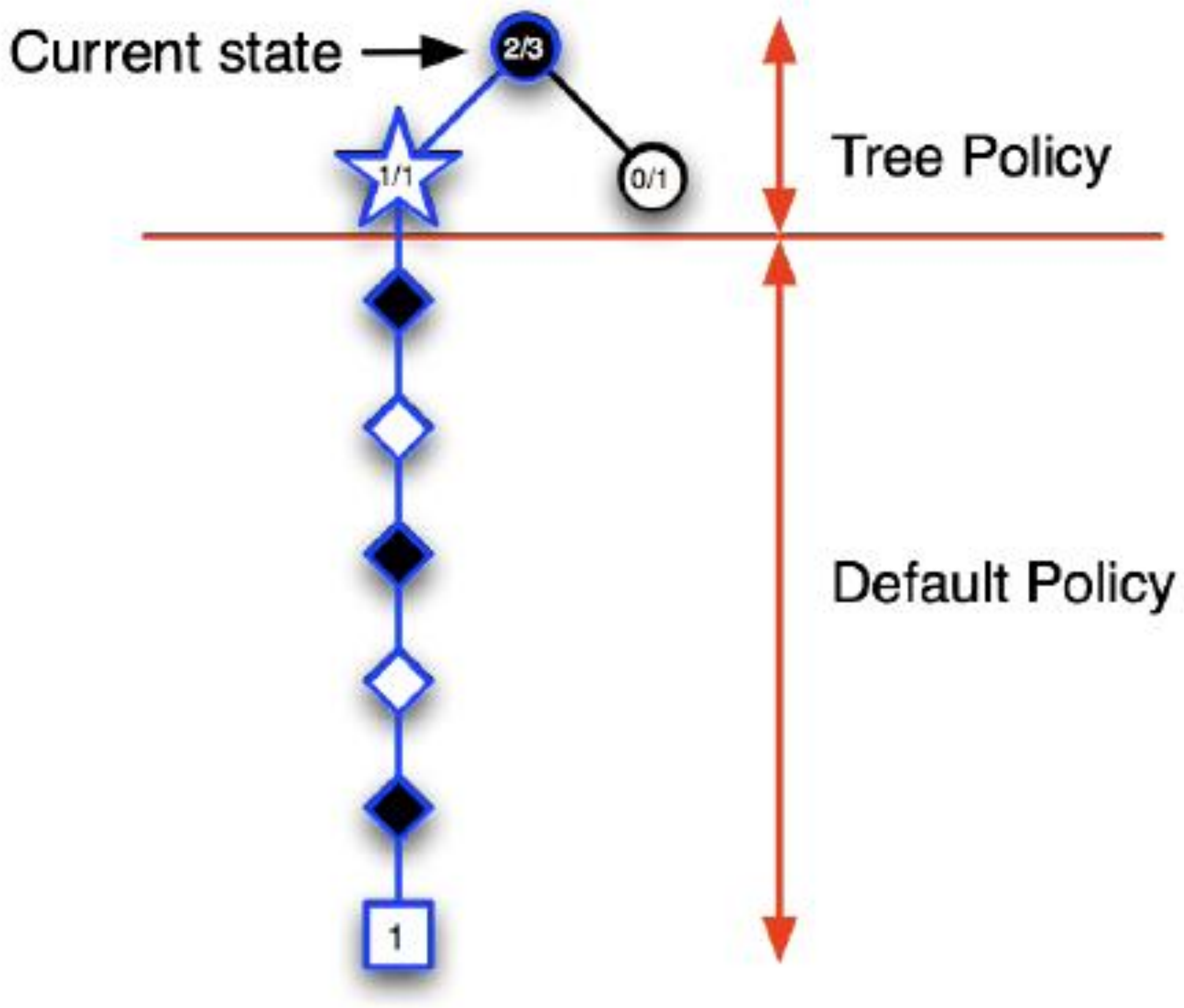
Applying Monte-Carlo Tree Search (1)



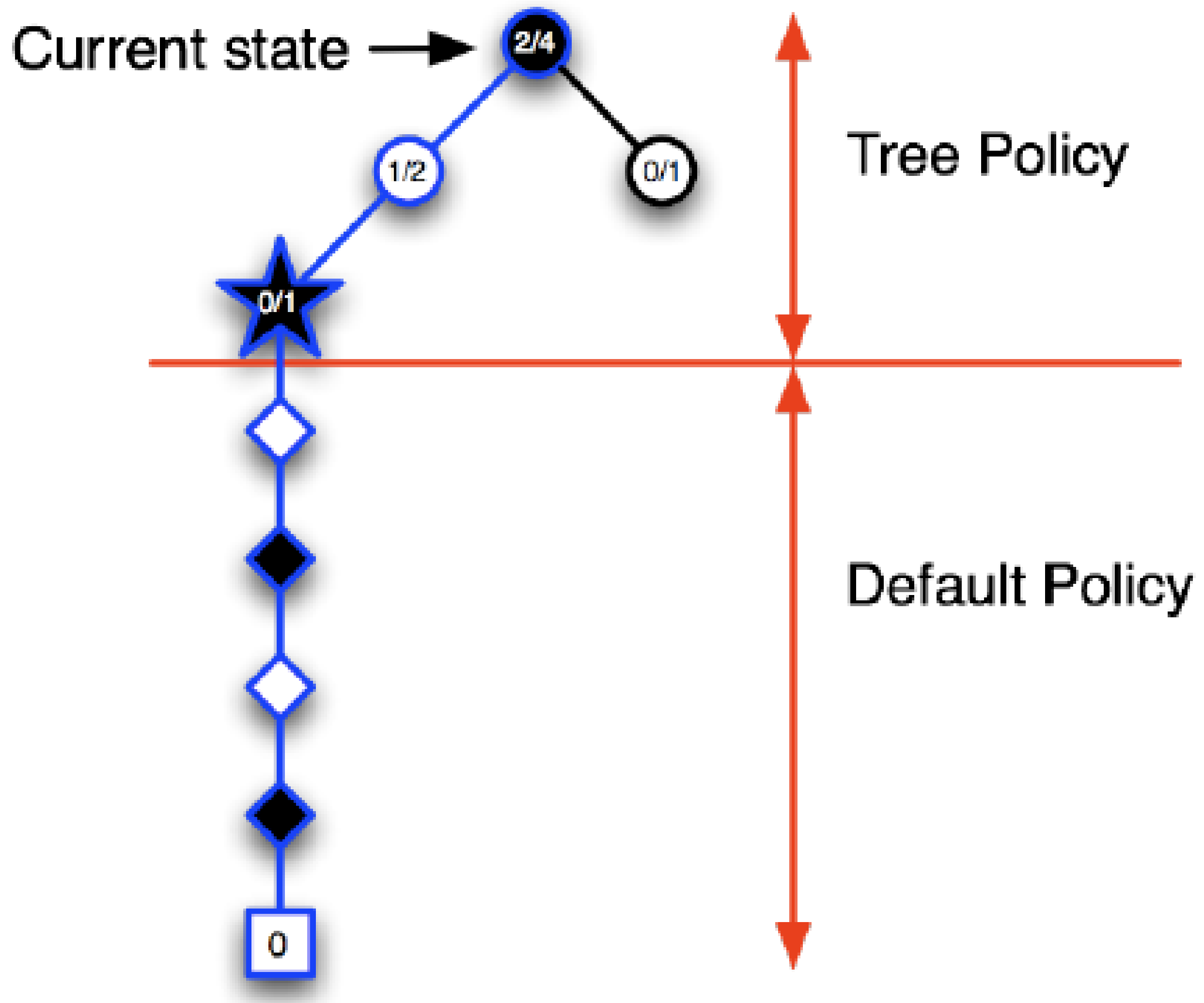
Applying Monte-Carlo Tree Search (2)



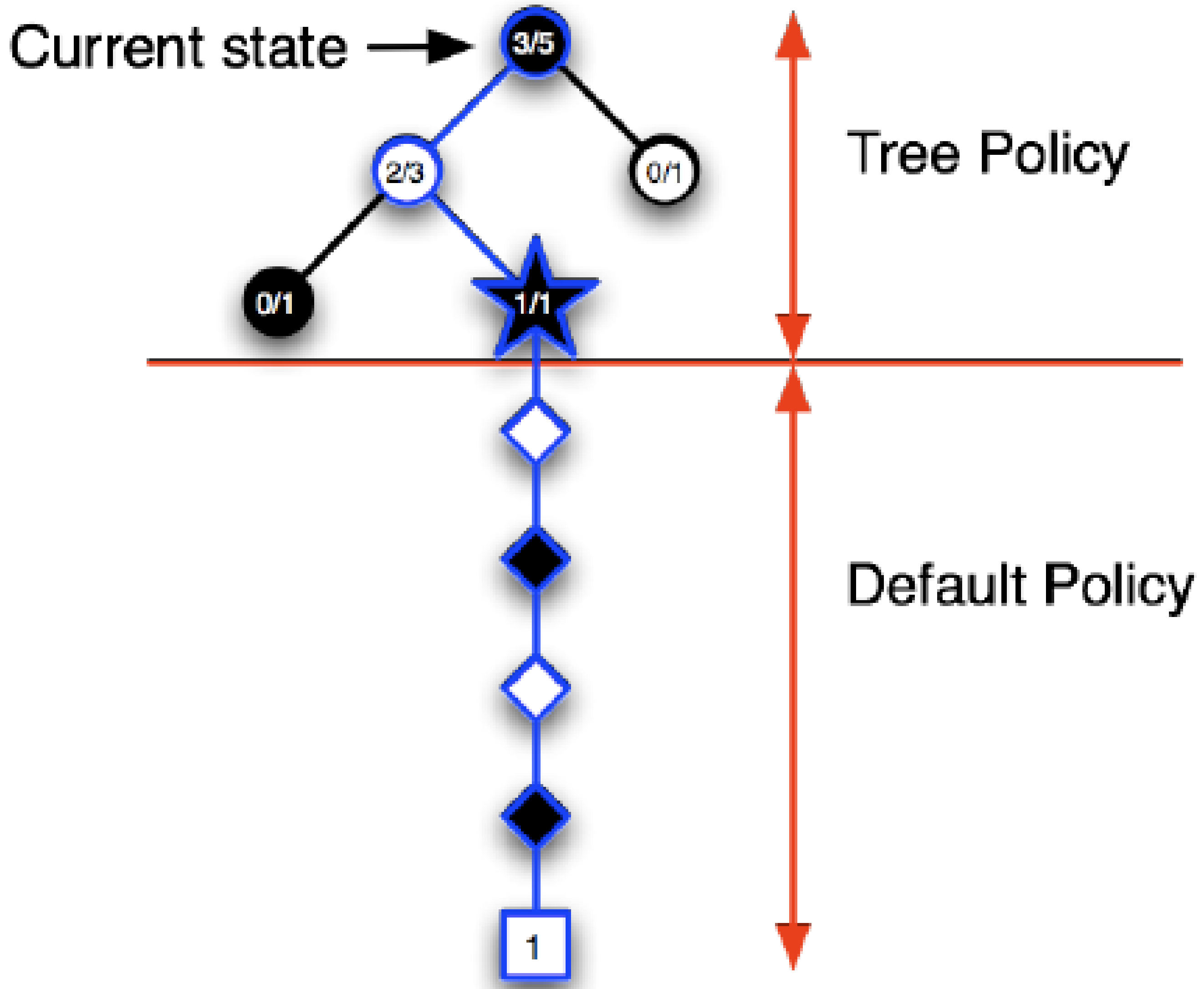
Applying Monte-Carlo Tree Search (3)



Applying Monte-Carlo Tree Search (4)



Applying Monte-Carlo Tree Search (1)



Advantages of MC Tree Search

- Highly selective best-first search
- Evaluates states dynamically (unlike e.g. DP)
- Uses sampling to break curse of dimensionality
- Works for \black-box" models (only requires samples)
- Computationally efficient and parallelisable

Reference

1. David Silver, Lecture 8: Integrating Learning and Planning
2. Chapter 8, Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning: An Introduction," 2nd edition, Nov. 2018