9. Integrating Planning and Learning

2019 Fall

Yusung Kim
yskim525@skku.edu

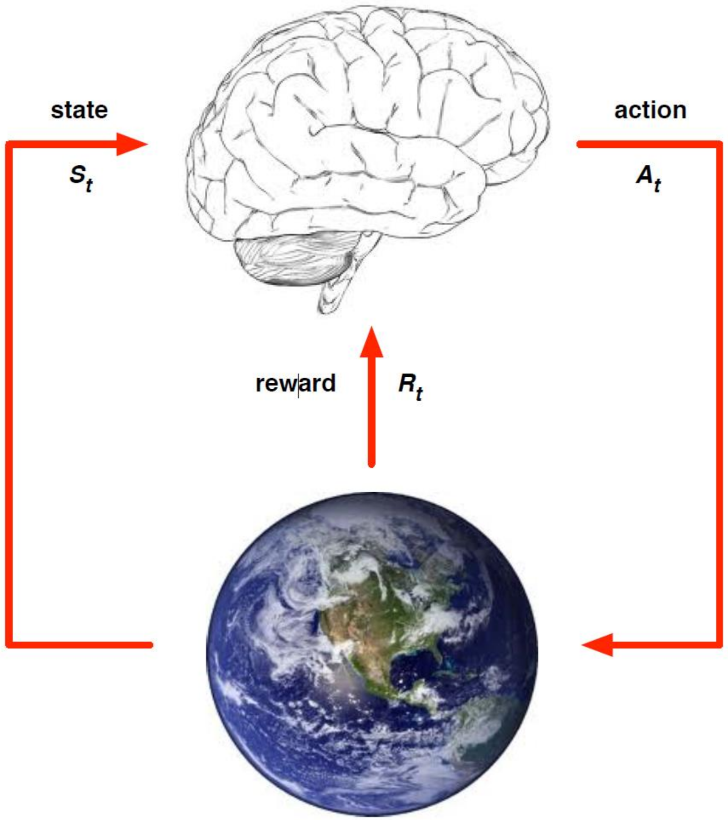
Model-Based Reinforcement Learning

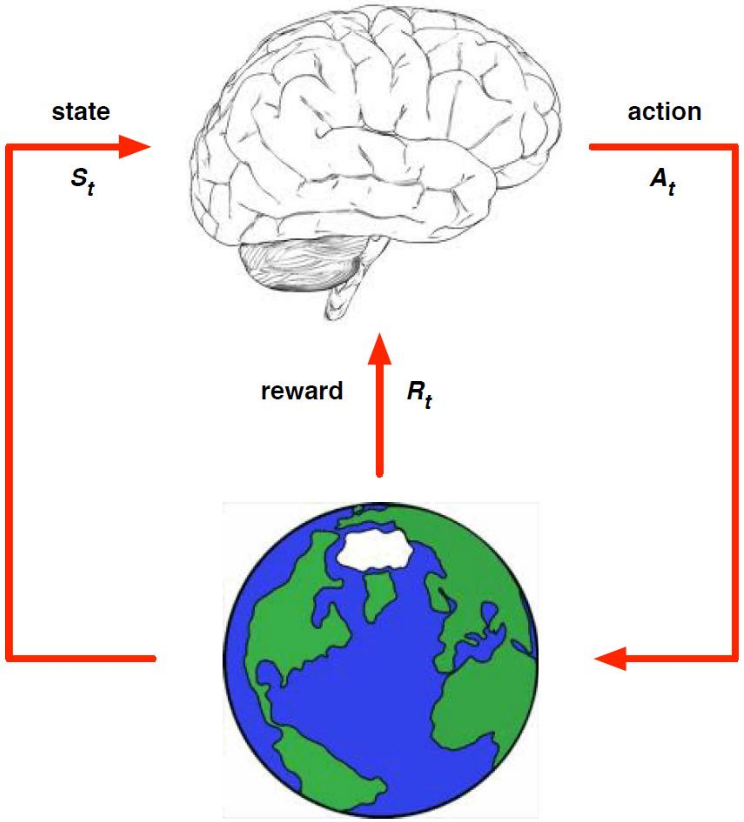
- Learn a model directly from experience
- Use planning to construct a value function or policy
- Integrating planning and learning into a single architecture

Model, Planning, Learning

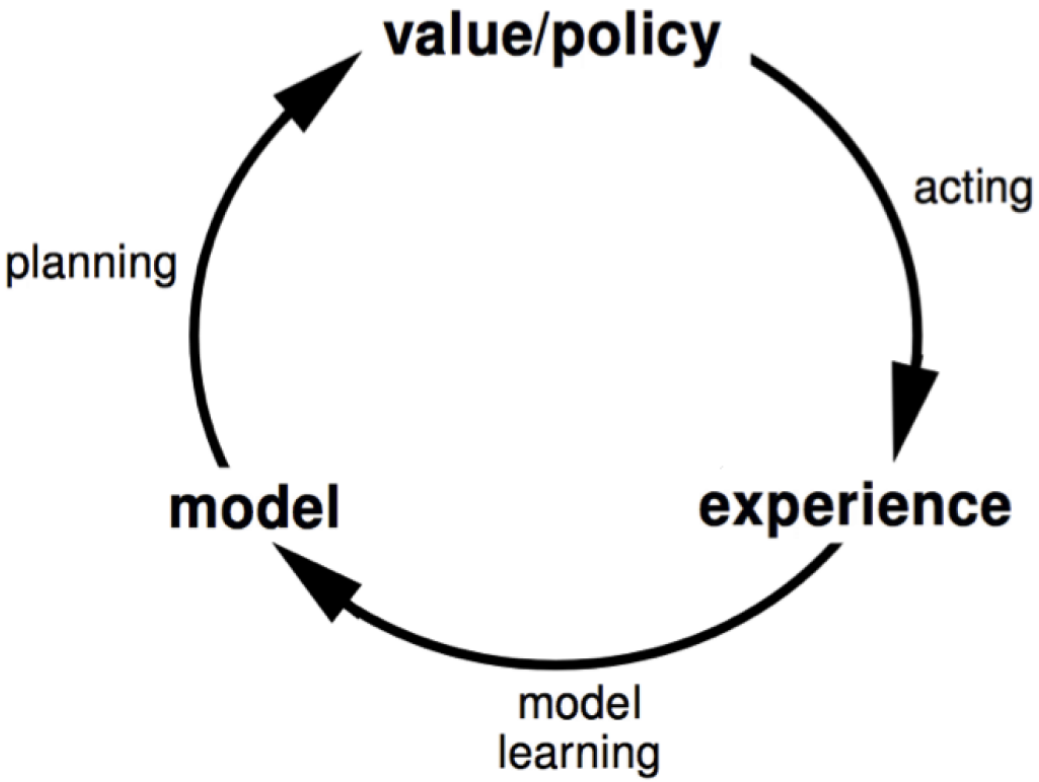
- Planning
  - Model-based & dynamic programming
- Model-free RL
  - No model
  - Learn value function and/or policy from experience
- Model-based RL
  - learn a model from experience
  - Plan value function and/or policy from model

Model-based RL vs. Model-free RL





Model-based RL



Advantages of Model-based RL

- Advantages:
  - Can efficiently learn model by supervised learning methods
  - Can reason about model uncertainty (like in upper confidence bound methods for exploration / exploitation trade offs)

- Disadvantages:
  - First learn a model, then construct a value function

  → two sources of approximation error

What is a Model ?

- A model M is a representation of an MDP $\langle S, A, P, R \rangle$, parameterized by η
- We will assume state space S and action space A are known
- So a model $M = \langle P_\eta , R_\eta \rangle$ represents state transitions $P_\eta \approx P$

  and rewards $R_\eta \approx R$

$$S_{t+1} \sim P_\eta \left(S_{t+1} | \, S_t, A_t \right)$$

$$R_{t+1} = R_\eta \left(R_{t+1} | \, S_t, A_t \right)$$

Model Learning

- Goal: estimate model M_η from experience $\{S_1, A_1, R_2 \ldots, S_T\}$
- This is a supervised learning problem
- Learning $s, a, \rightarrow r$ is a regression problem
- Learning $s, a, \rightarrow s'$ is a density estimation problem
- Pick loss function, e.g. mean-squared error, KL divergence, …
- Find parameters η that minimize empirical loss

Table Lookup Model

- Model is an explicit MDP, $\hat{P}, \hat{R}$
- Count visits $N(s,a)$ to each state action pair

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

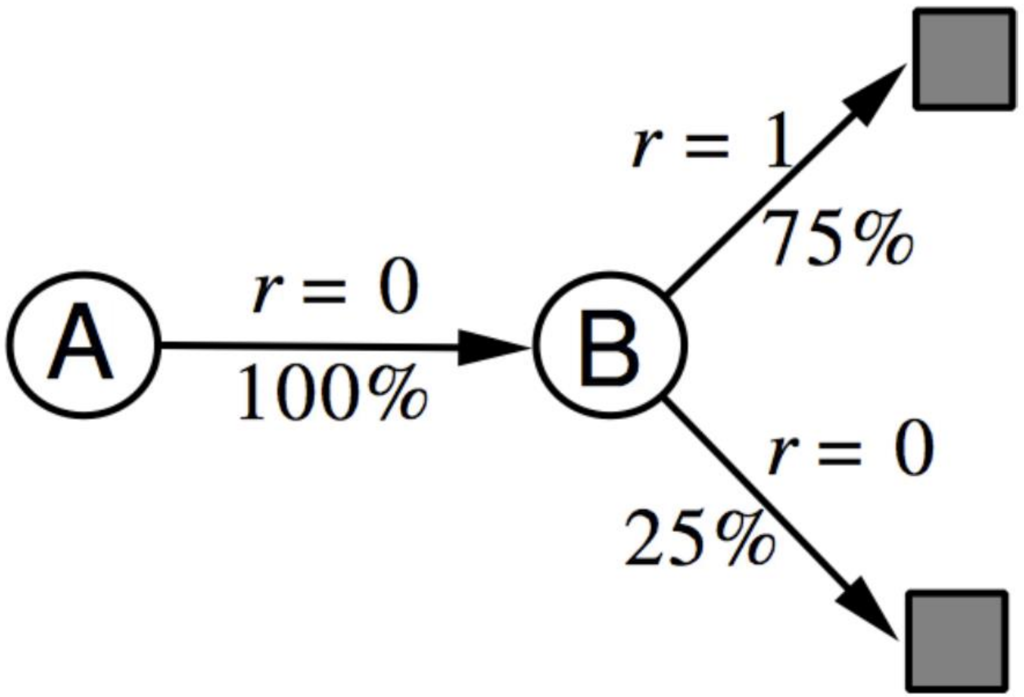
$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t = s, a) R_t$$

- Alternatively
  - At each time-step t, record experience tuple $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$
  - To sample model, randomly pick tuple matching $\langle s, a, \cdot, \cdot \rangle$

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



We have constructed a table lookup model from the experience

Planning with a Model

- Given a model $M_\eta = \langle P_\eta, R_\eta \rangle$
- Solve the MDP $\langle S, A, P_\eta, R_\eta \rangle$
- Using favorite planning algorithm
  - Policy iteration
  - Value iteration
  - …

Sample-based Planning

- A simple but powerful approach to planning
- Use the model only to generate samples
- Sample experience from model

$$S_{t+1} \sim P_\eta \left(S_{t+1} \mid S_t, A_t\right)$$

$$R_{t+1} = R_\eta \left(R_{t+1} \mid S_t, A_t\right)$$

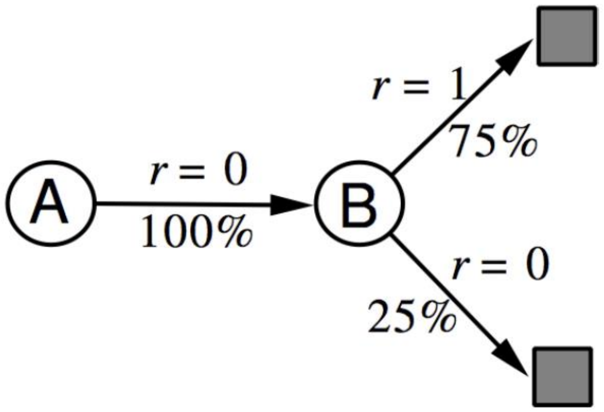
- Apply model-free RL to samples (MC, Sarsa, Q-learning …)
- Sample-based planning methods are often more efficient

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

- Given an imperfect model $\langle P_\eta , R_\eta \rangle \neq \langle P, R \rangle$
- Performance of model-based RL is limited to optimal policy for approximate MDP $\langle S, A, P_\eta , R_\eta \rangle$
  - Model-based RL is only as good as the estimated model
- When the model is inaccurate, planning process will compute a suboptimal policy
  - Solution 1: when model is wrong, use model-free RL
  - Solution 2: reason explicitly about model uncertainty (Exploration & Exploitation)

Real and Simulated Experience

- We consider two sources of experience
- Real experience Sampled from environment (true MDP)

$$S' \sim \mathcal{P}_{s,s'}^{a}$$
$$R = \mathcal{R}_{s}^{a}$$

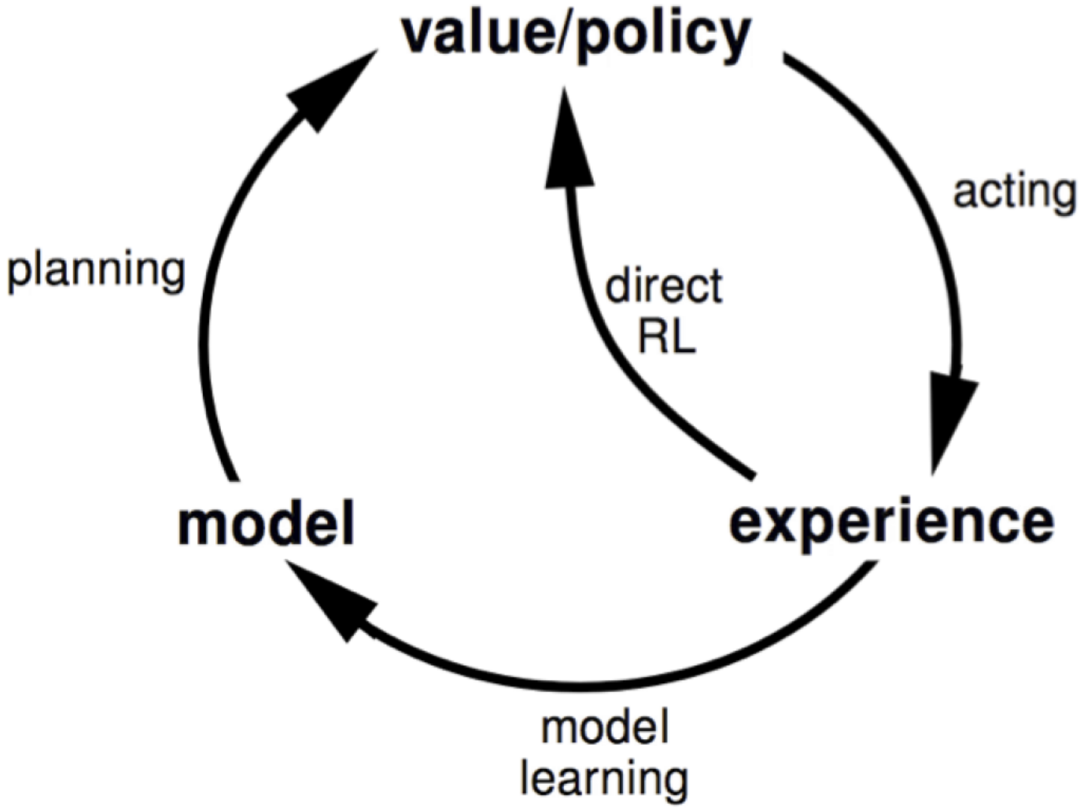
- Simulated experience Sampled from model (approximate MDP)

$$S' \sim \mathcal{P}_{\eta}(S' \mid S, A)$$
$$R = \mathcal{R}_{\eta}(R \mid S, A)$$

Integrating Learning and Planning

- Model-Free RL
  - 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 Architecture



Dyna-Q Algorithm

Tabular Dyna-Q

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

Loop forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \varepsilon$-greedy(S, Q)

(c) Take 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) Loop repeat n times:

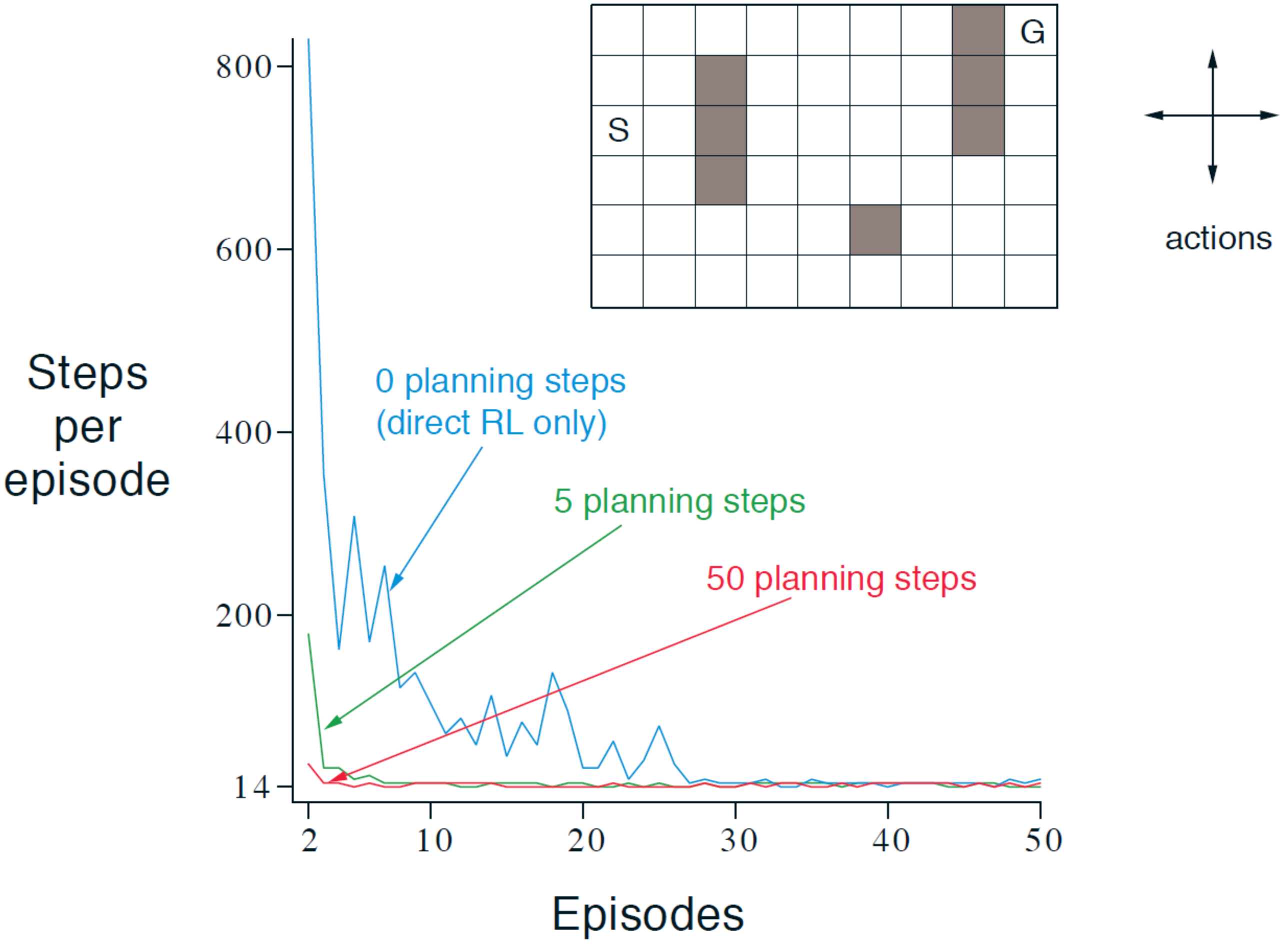
$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$

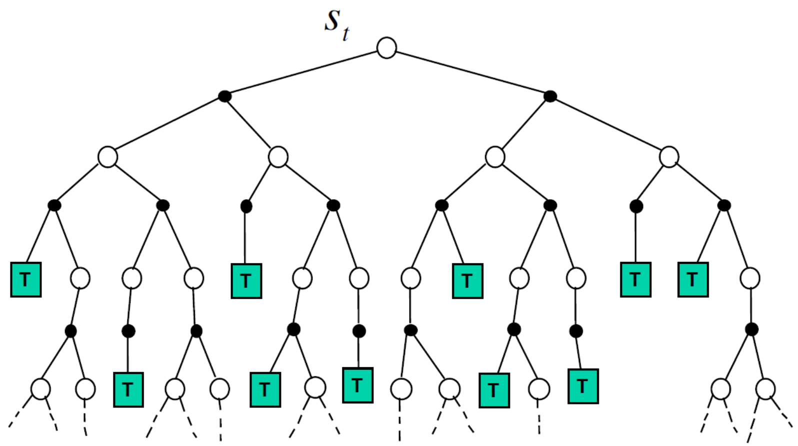
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Dyna-Q on a Simple Maze



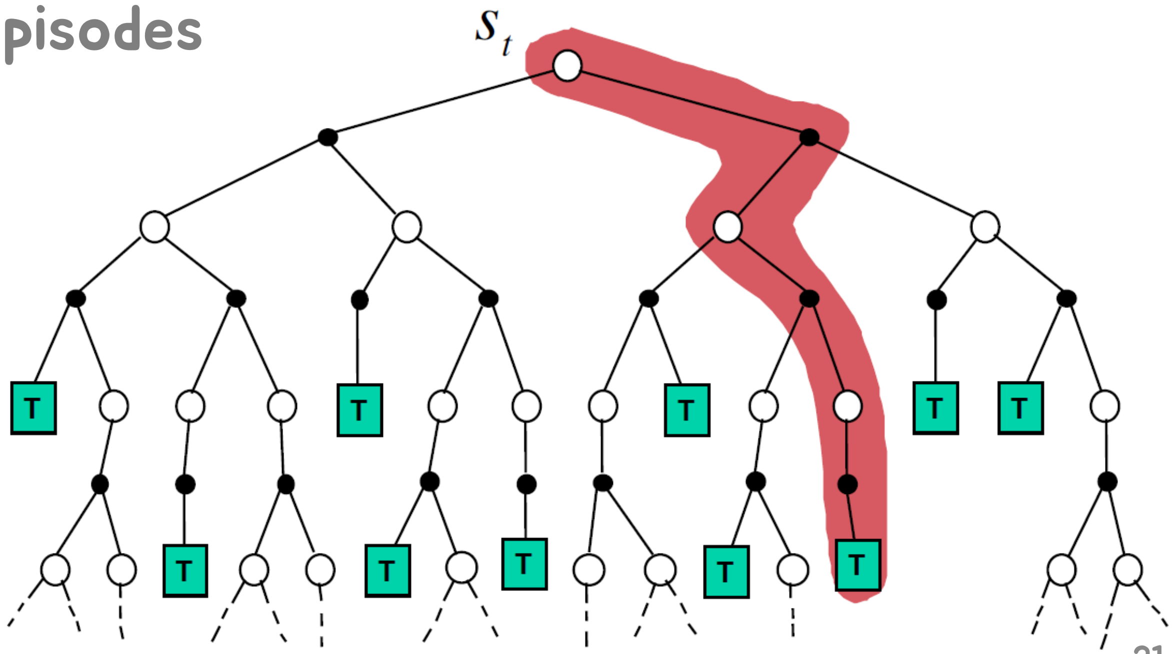
Forward Search

- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead
- No need to solve whole MDP, just sub-MDP starting from now



Simulation-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



Simulation-Based Search

- Simulate episodes of experience from now with the model

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

- Apply model-free RL to simulated episodes
  - e.g. Monte-Carlo control → Monte-Carlo search

Simple Monte-Carlo Search

- Given a model M_v and a simulation policy π
- For each action $a \in A$
  - Simulate K episodes from current (real) state s_t

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

  - Evaluate actions by mean return (Monte-Carlo evaluation)

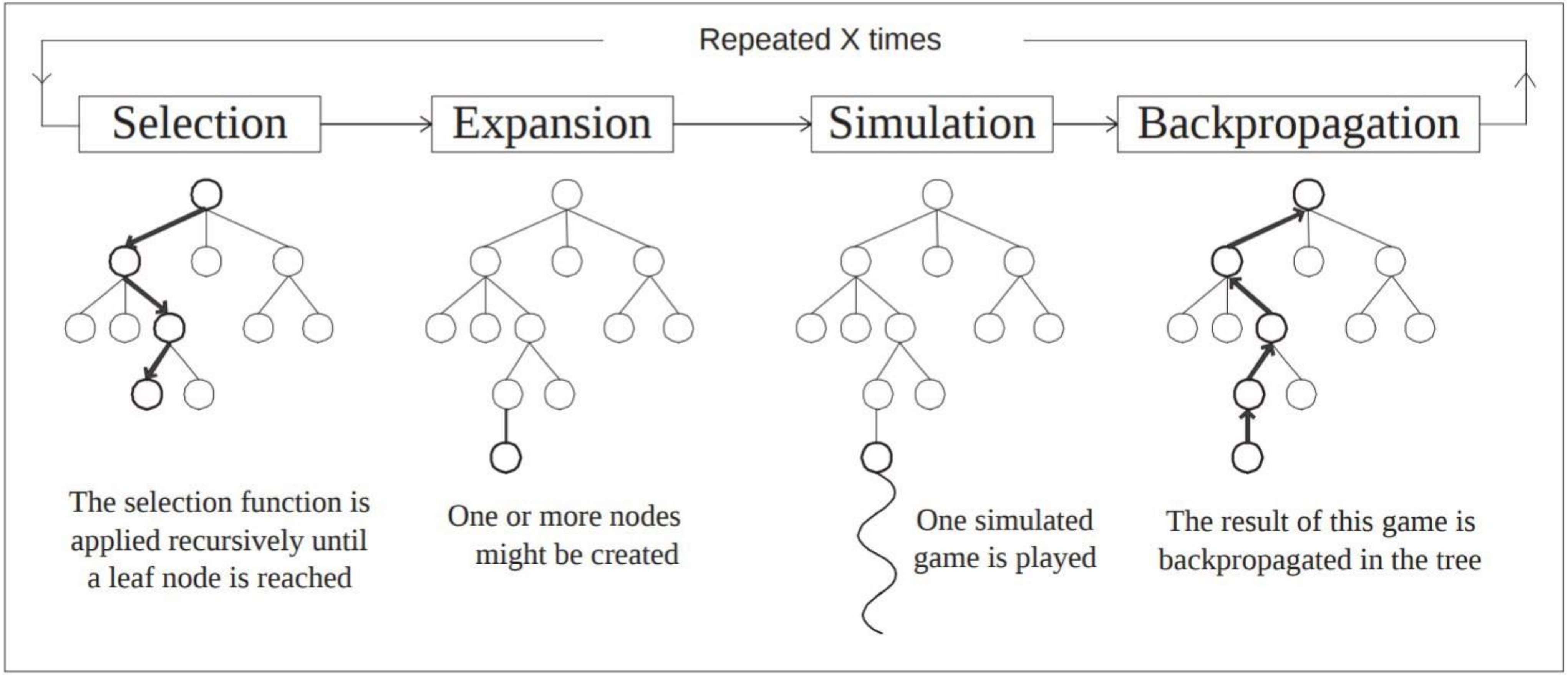
$$Q(s_t, a) = \frac{1}{K} \sum_{k=1}^{K} G_t \xrightarrow{P} q_\pi(s_t, a)$$

- Select current (real) action with maximum value $a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(s_t, a)$

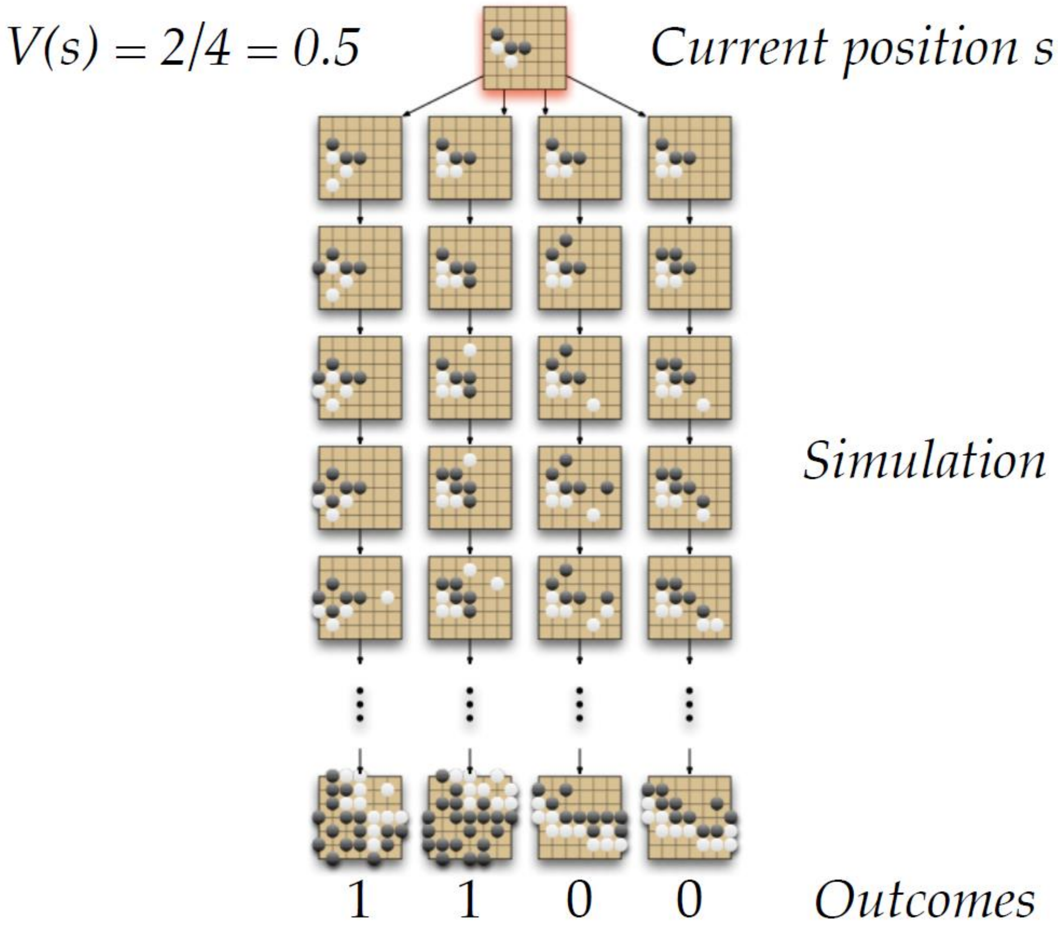
Monte-Carlo Tree Search (MCTS)

- In MCTS, the simulation policy π improves
- Each simulation consists of two phases
  - Tree policy : pick actions to maximize Q(S;A)
  - Default/Rollout policy : pick actions randomly or another policy
- Repeat (each simulation)
  - Evaluate states Q(S,A) by Monte-Carlo evaluation
  - Improve tree policy, e.g. by ϵ $-$greedy(Q)

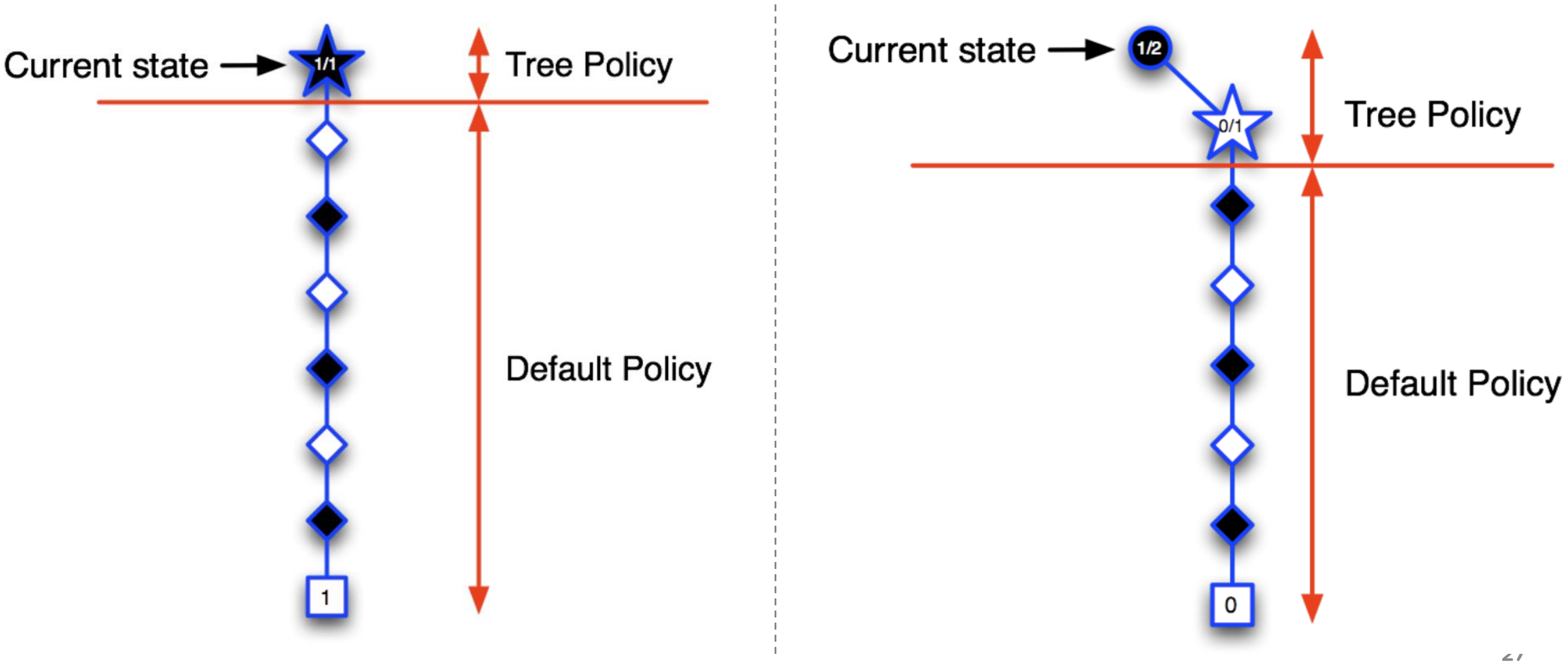
General Process in MCTS



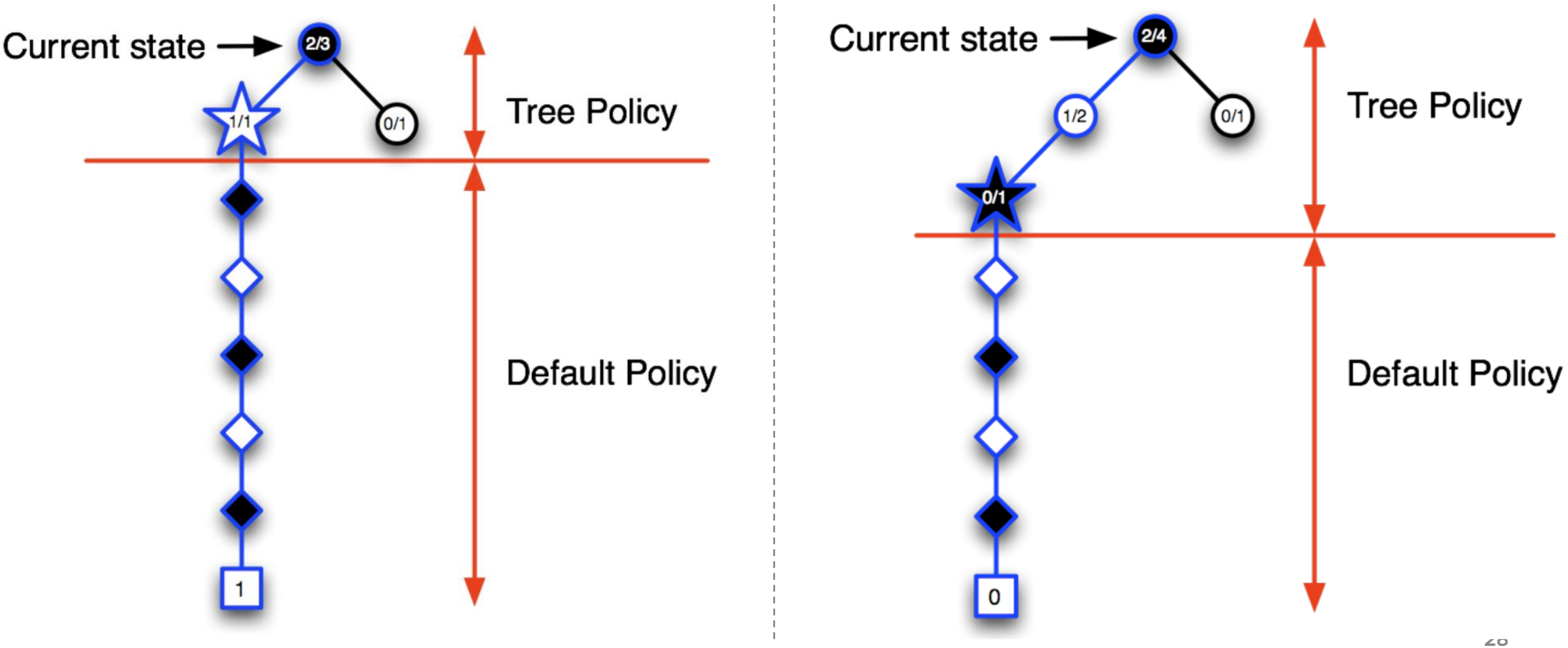
Monte-Carlo Evaluation in Go



Applying Monte-Carlo Tree Search



Applying Monte-Carlo Tree Search



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, anytime, parallelizable

Exploration and Exploitation

- How to select child nodes
  - Balancing exploitation and exploration
- UCT (Upper Confidence bounds applied to Trees) algorithm

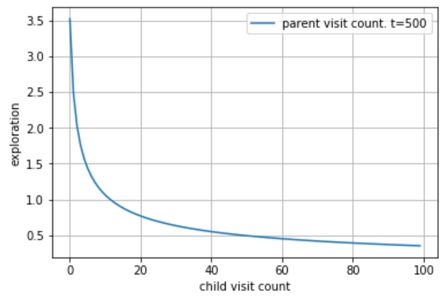
$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

w_i : the number of wins for the node

n_i : the number of simulations for the node

N_i : for the total number of simulations

c : the exploration parameter; in practice usually chosen empirically



Summary: MTCS

- Efficiently navigate very large state spaces
- Not dependent on expert knowledge
- More simulations, better results
- Do its best to find best/good results within a limited time