Lecture 8: Integrating Learning and Planning

David Silver

Outline

build model for env.

combine all the ideas.

imagine & learn by imagination.

Outline

Model-Based Reinforcement Learning

- *Last lecture*: learn policy directly from experience
- *Previous lectures*: learn value function directly from experience
- *This lecture*: learn model directly from experience
- and use planning to construct a value function or policy
- Integrate learning and planning into a single architecture

Model-Based and Model-Free RL

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience

no model - not represent transition dynamics or reward function that reward env operates on.

Learn → weights

Model-Based and Model-Free RL

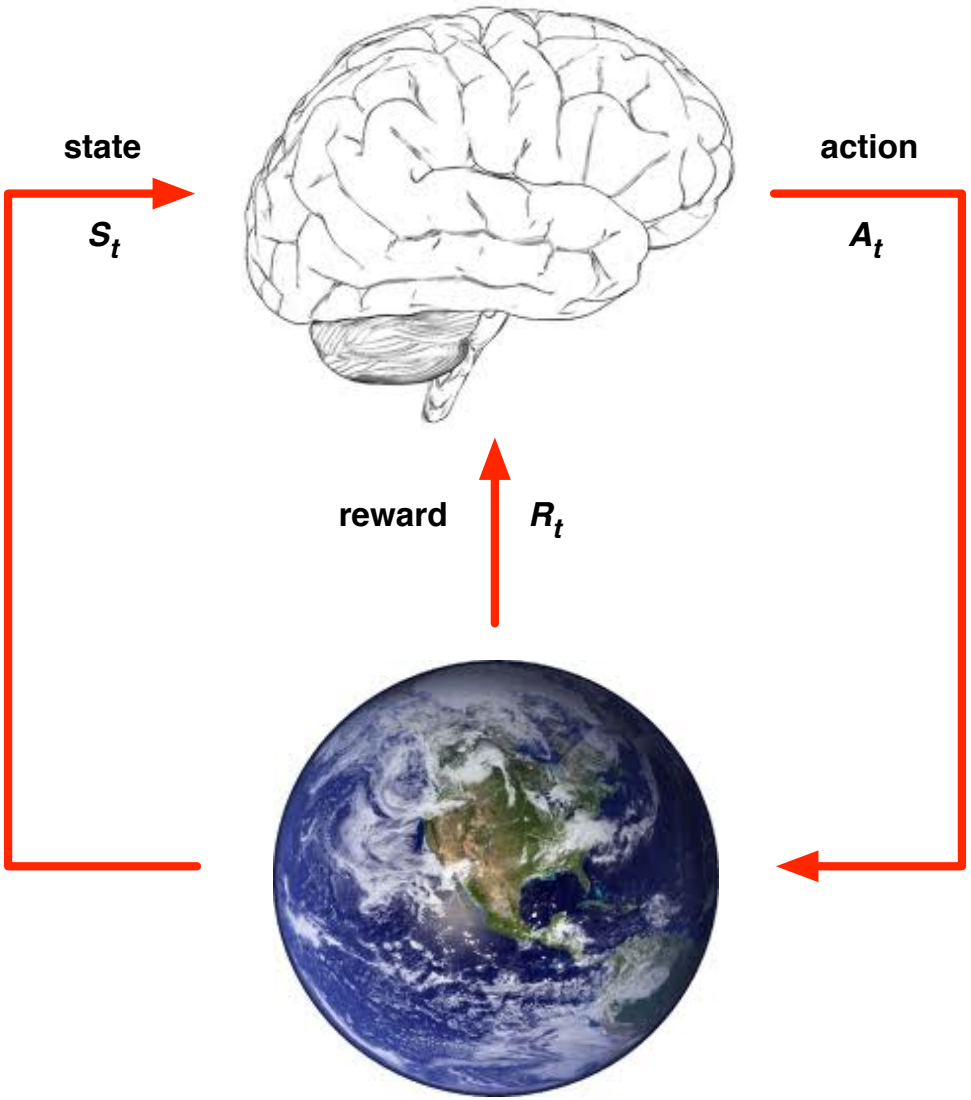
- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience
- Model-Based RL
  - Learn a model from experience
  - Plan value function (and/or policy) from model

construct transition dynamic reward

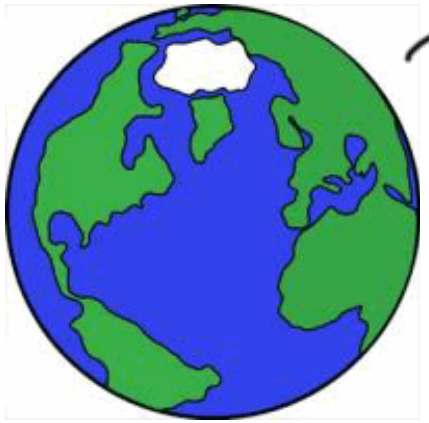
look ahead to select which right

fune or right action are to select.

Model-Free RL



Model-Based RL



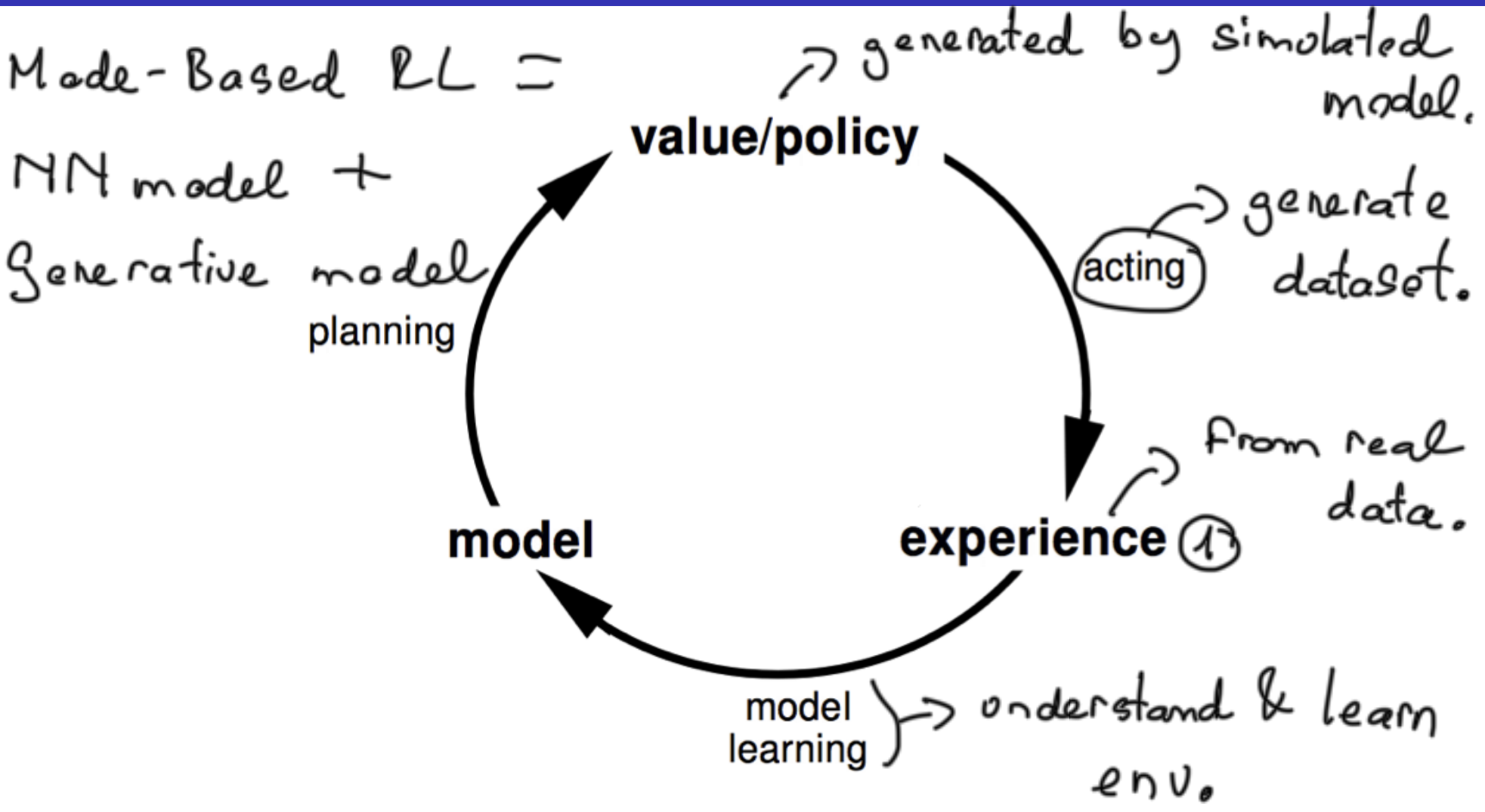
w.r.t. internal model.

replace the real world by the simulated one.

give ability to think to plan without any further interaction with env.

Outline

Model-Based RL



Advantages of Model-Based RL

illustrate some MDP
's really hard to optimize
without planning.

chess → how to give reward for one movement?
→ Am I likely to win from this position?

construct value fonc

Advantages:

- Can efficiently learn model by supervised learning methods
  → data + label
- Can reason about model uncertainty

Disadvantages:

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

What is a Model?

- A *model* $\mathcal{M}$ is a representation of an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, parametrized by η
- We will assume state space $\mathcal{S}$ and action space $\mathcal{A}$ are known
- So a model $\mathcal{M} = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ represents state transitions $\mathcal{P}_\eta \approx \mathcal{P}$ and rewards $\mathcal{R}_\eta \approx \mathcal{R}$

transition

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

reward

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

- Typically assume conditional independence between state transitions and rewards

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

Model Learning

- Goal: estimate model $\mathcal{M}_\eta$ from experience $\{S_1, A_1, R_2, ..., S_T\}$
- This is a supervised learning problem (transformed into)

state space & action space concerns

Input Labels

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

- Learning $s, a \to r$ is a *regression* problem
- Learning $s, a \to s'$ is a *density estimation* problem (learn the whole dist. of state.)
- Pick loss function, e.g. mean-squared error, KL divergence, ...
- Find parameters η that minimise empirical loss

Examples of Models

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

Table Lookup Model

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

↳ use empirical count to construct prob.

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

→ transition dynamics.

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

→ average reward.

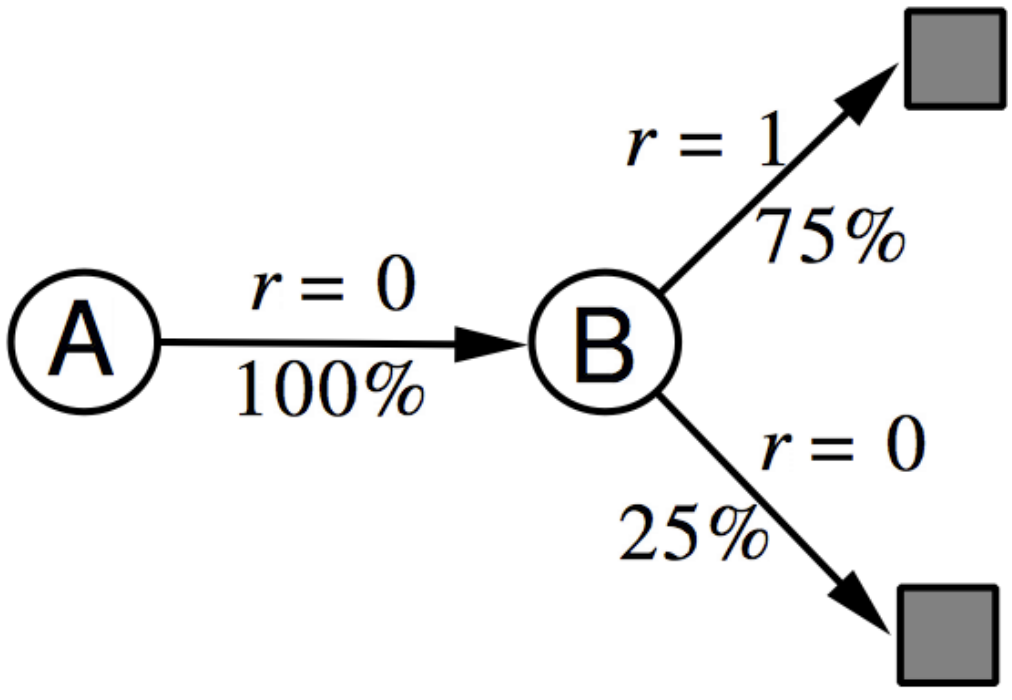
- Alternatively
  - At each time-step t, record experience tuple $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$ → sample uniformly by fixing $\langle S_t, A_t \rangle$.
  - To sample model, randomly pick tuple matching $\langle s, a, \cdot, \cdot \rangle$

parametric VS. non-parametric

AB Example

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

$$
\begin{array}{ll}
ep^1 & A, 0, B, 0 \\
ep^2 & B, 1 \\
ep^3 & B, 1 \\
ep^4 & B, 1 \\
ep^5 & B, 1 \\
ep^6 & B, 1 \\
ep^7 & B, 1 \\
ep^8 & B, 0
\end{array}
$$



We have constructed a table lookup model from the experience

Planning with a Model

- Given a model $\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ → estimated MDP
- Solve the MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ → try to find the best things to do.
- Using favourite planning algorithm → DP with assumption of MDP being given.
  - Value iteration
  - Policy iteration
  - Tree search
  - ...

Sample-Based Planning

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

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

- Apply model-free RL to samples, e.g.:
  - Monte-Carlo control
  - Sarsa
  - Q-learning
- Sample-based planning methods are often more efficient

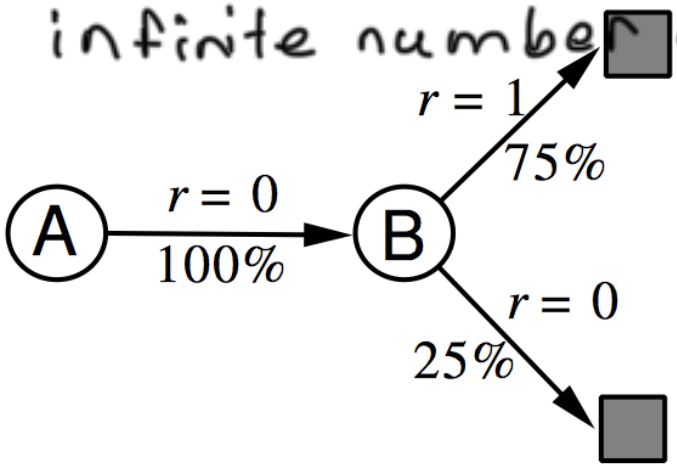
Back to the AB Example

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

when we have the model, it means that we have infinite number of data.

Real experience

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

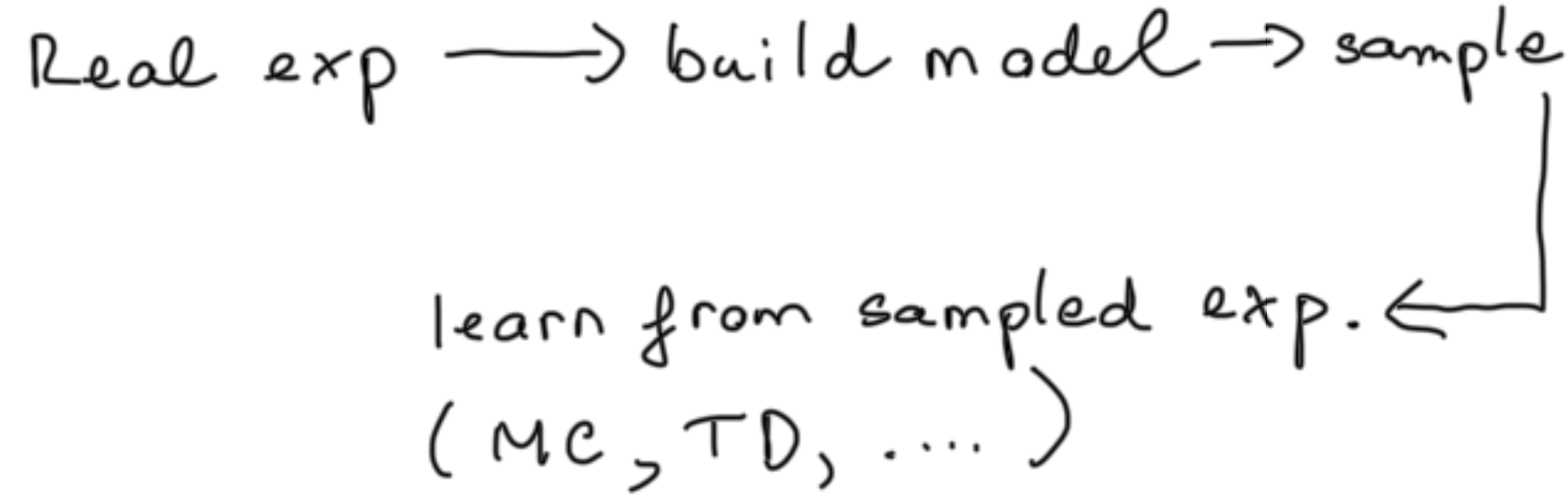


Sampled experience

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: $V(A) = 1, V(B) = 0.75$

$= \frac{1+1}{2} = 1 \quad = \frac{6}{8} = 0.75$



How to evaluate our mdp model is good or bad?

Planning with an Inaccurate Model

- Given an imperfect model $\langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle \neq \langle \mathcal{P}, \mathcal{R} \rangle$
- Performance of model-based RL is limited to optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- i.e. 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

Model-based can be applied to continuos state-action space.

Outline

bring model-free & model-based together

Real and Simulated Experience

We consider two sources of experience

Real experience Sampled from environment (true MDP)

$$S' \sim \mathcal{P}^a_{s,s'}$$
$$R = \mathcal{R}^a_s$$

Simulated experience Sampled from model (approximate MDP)

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

How we learn to explore?

next lecture !!!

Integrating Learning and Planning

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from real experience

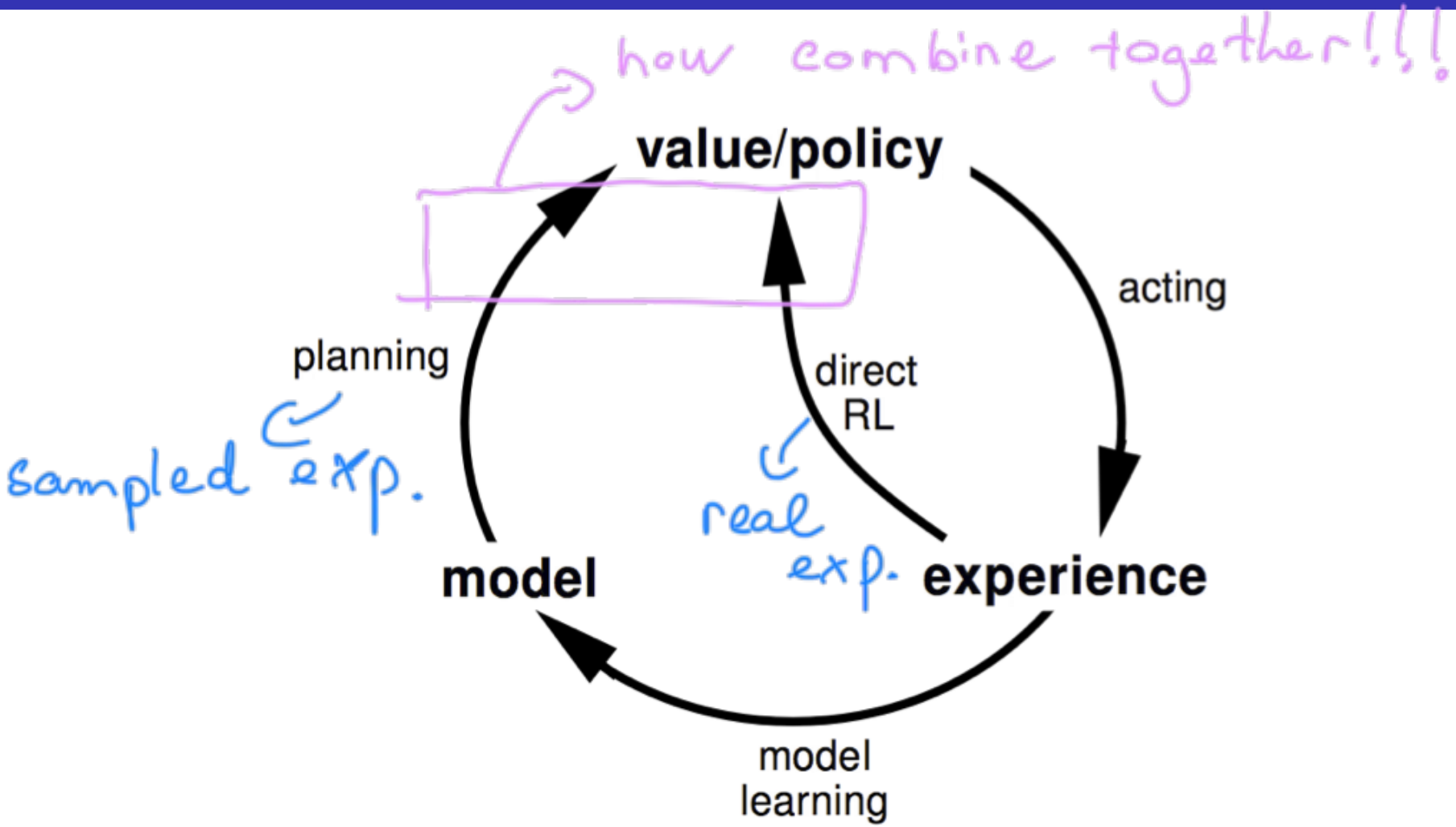
Integrating Learning and Planning

- 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

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 Architecture



Dyna-Q Algorithm

simply table look up model

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

(c) Execute action A; observe resultant reward, R, and state, S'

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

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment) — supervised learning

(f) Repeat n times: → how much thinking do I have to do?

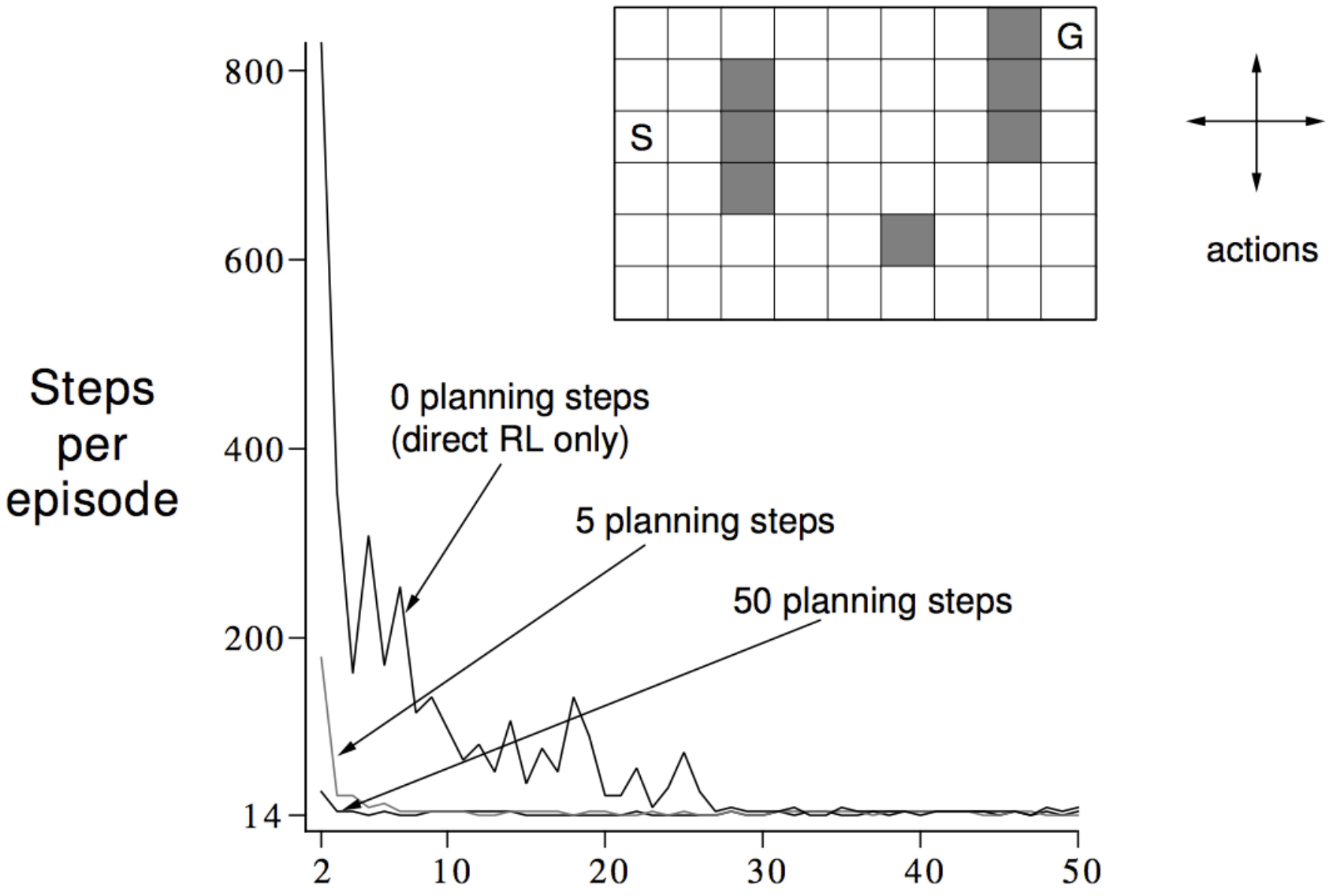
$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$ → imagine transition

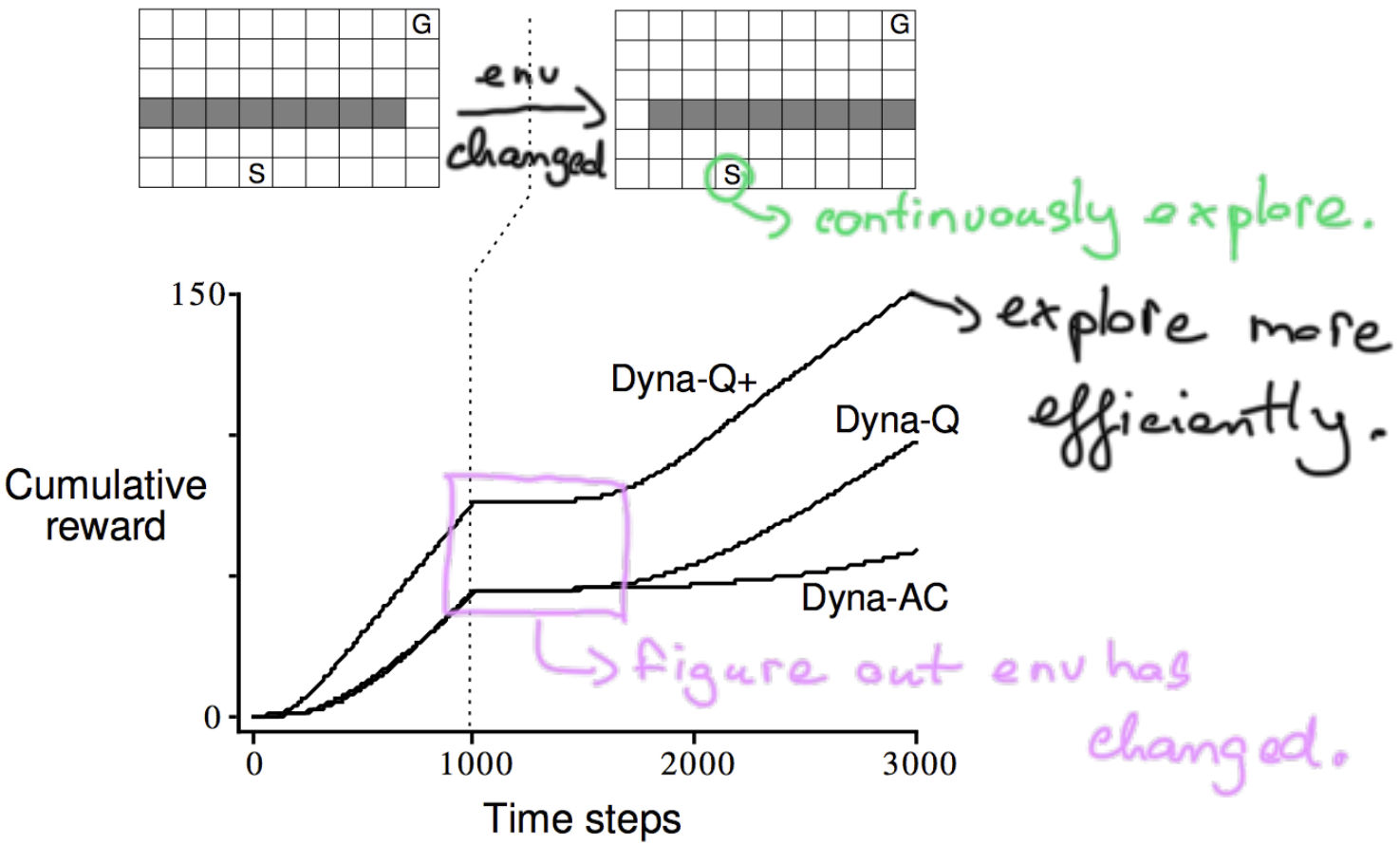
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Dyna-Q on a Simple Maze



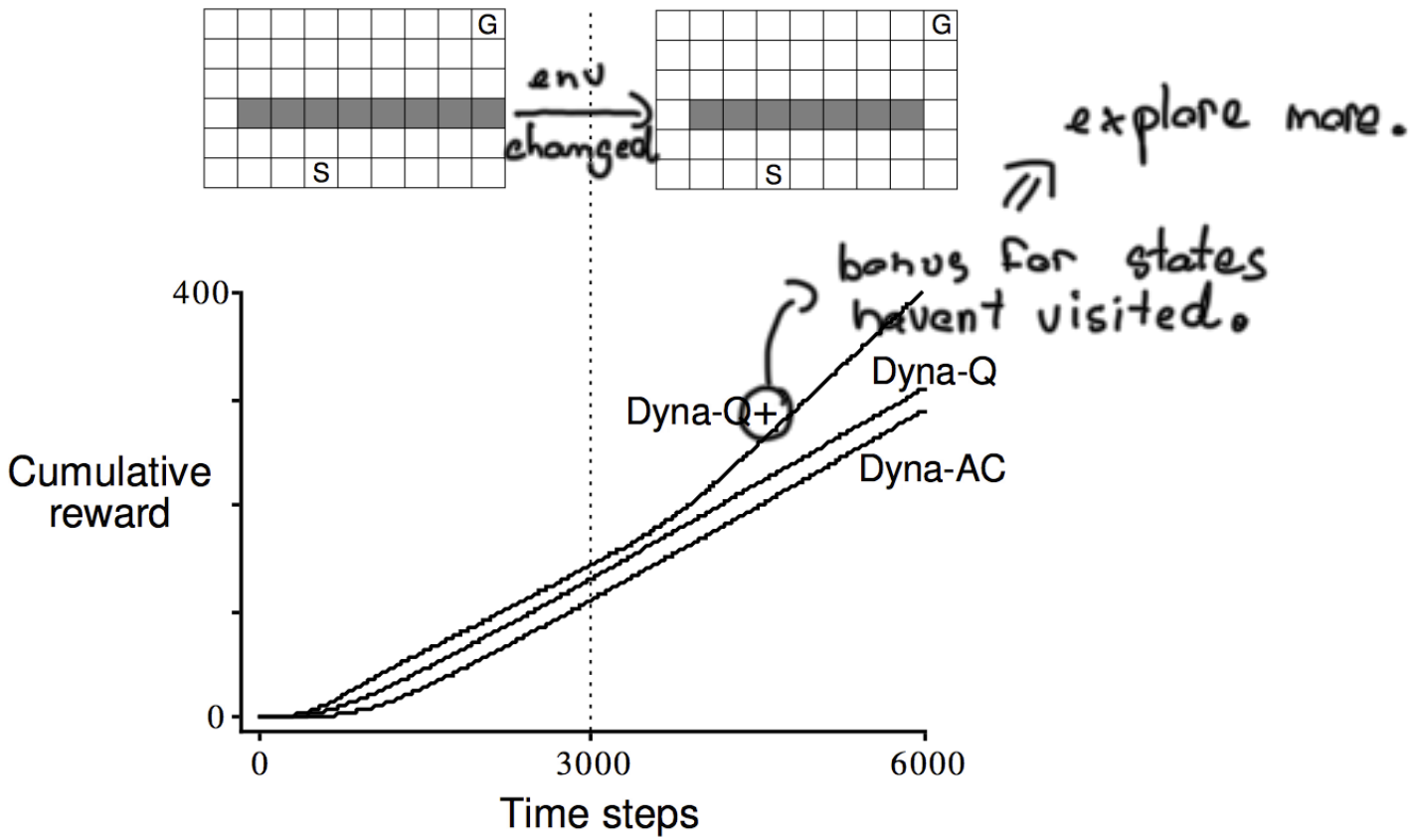
Dyna-Q with an Inaccurate Model

- The changed environment is harder



Dyna-Q with an Inaccurate Model (2)

- The changed environment is easier



Outline

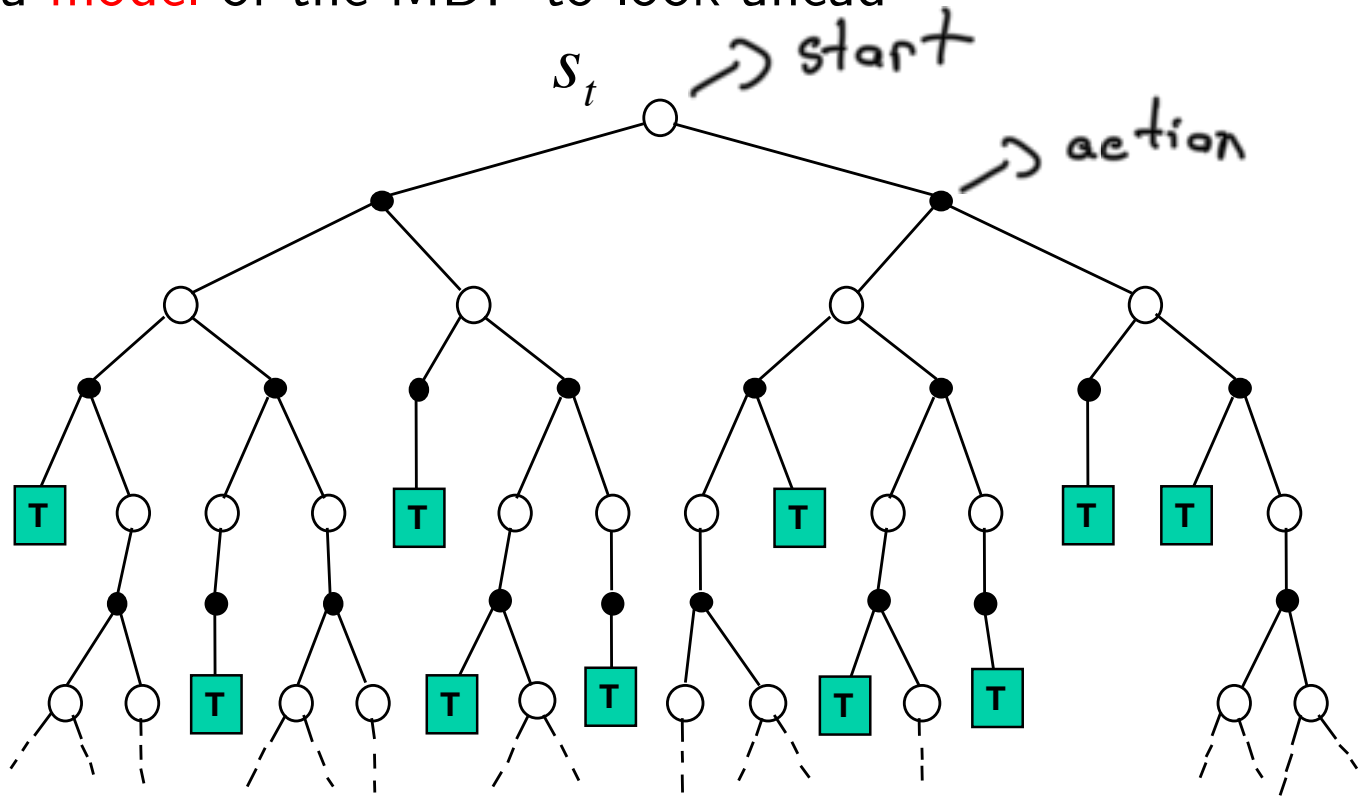
Focus on planning part.

how to plan effectively.

Forward Search

↳ dont explore the entire space.

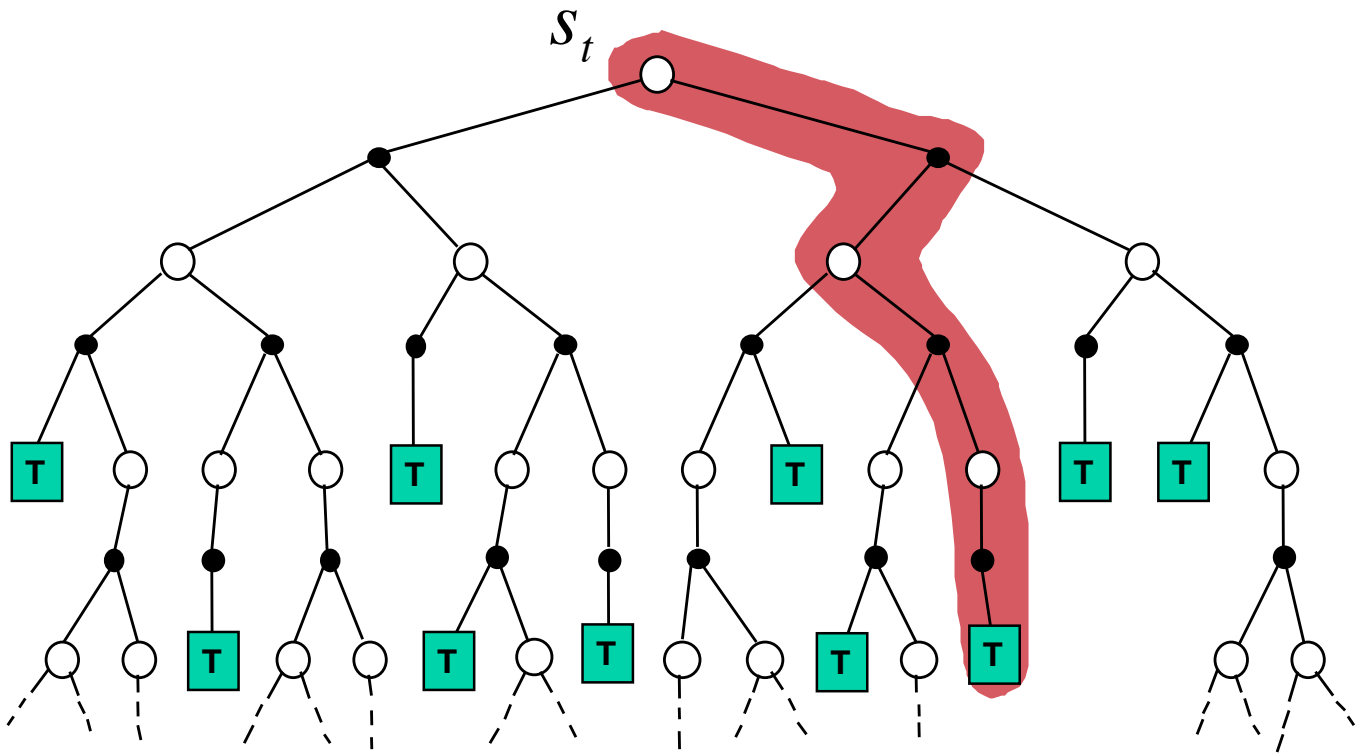
- Forward search algorithms select the best action by lookahead
- They build a search tree with the current state s_t at the root
- Using a model of the MDP to look ahead



- 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 (2)

- Simulate episodes of experience from now with the model

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

simulation-th

- Apply model-free RL to simulated episodes
  - Monte-Carlo control $\rightarrow$ Monte-Carlo search
  - Sarsa $\rightarrow$ TD search

Simple Monte-Carlo Search

- Given a model $\mathcal{M}_\nu$ and a simulation policy π (fixed)
- For each action $a \in \mathcal{A}$
  - Simulate K episodes from current (real) state s_t

$$\{s_t, a, R^k_{t+1}, S^k_{t+1}, A^k_{t+1}, ..., S^k_T\}^K_{k=1} \sim \mathcal{M}_\nu, \pi$$

  - 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

$$a_t = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s_t, a)$$

some w, we gonna pick action in our imagination.

Monte-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

$$a_t = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s_t, a)$$

Monte-Carlo Tree Search (Simulation)

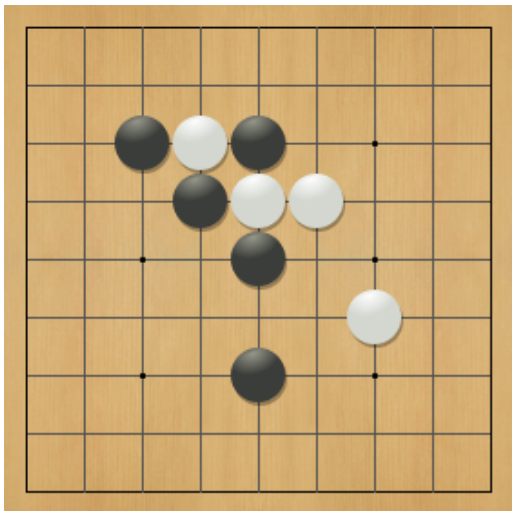
- In MCTS, the simulation policy π improves
- Each simulation consists of two phases (in-tree, out-of-tree)
  - Tree policy (improves): pick actions to maximise $Q(S, A)$ in-tree.
  - Default policy (fixed): pick actions randomly → out-of-tree.
- Repeat (each simulation)
  - Evaluate states $Q(S, A)$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

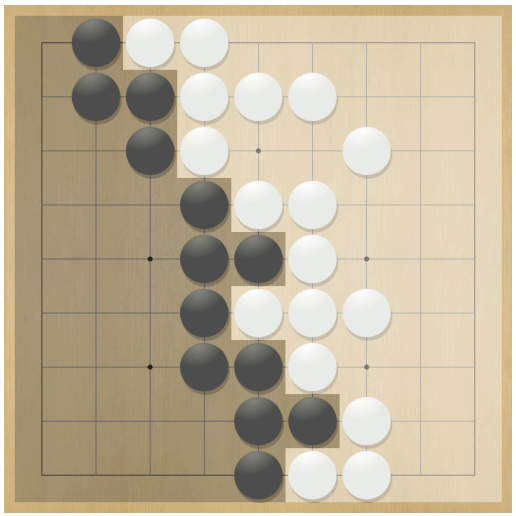
Case Study: the Game of Go

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

Rules of Go

- Usually played on 19x19, also 13x13 or 9x9 board
- Simple rules, complex strategy
- Black and white place down stones alternately
- Surrounded stones are captured and removed
- The player with more territory wins the game





Position Evaluation in Go

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

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

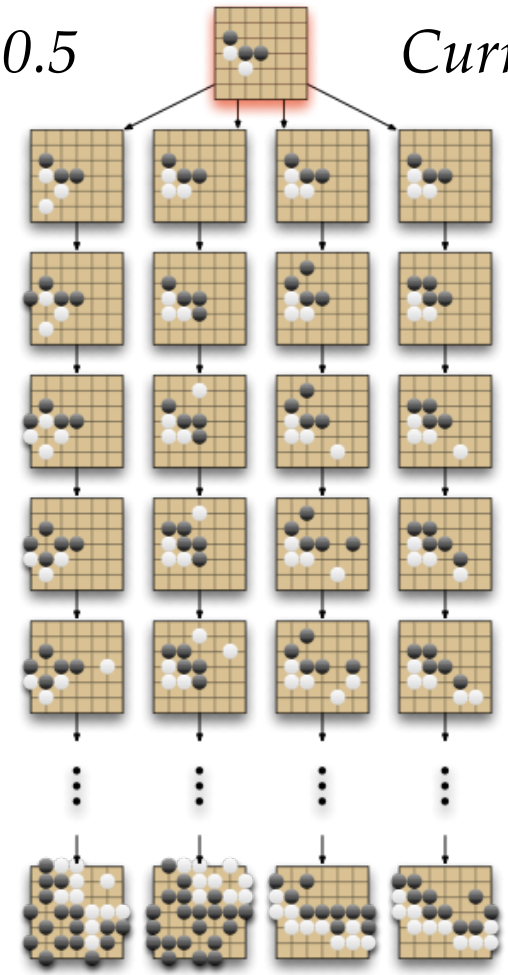
$$R_T = \begin{cases} 1 & \text{if Black wins} \\ 0 & \text{if White wins} \end{cases}$$

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

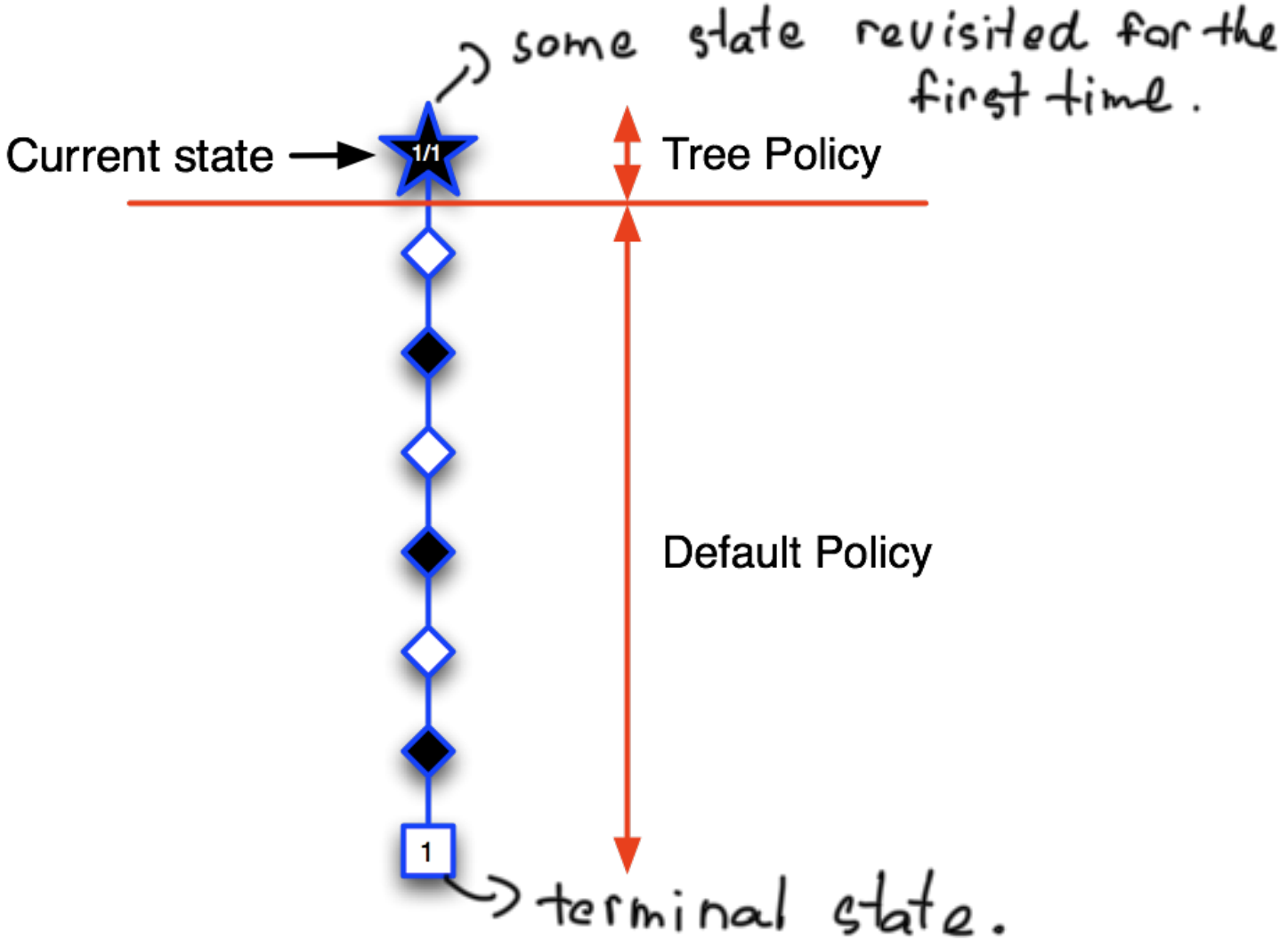
$$v_\pi(s) = \mathbb{E}_\pi \left[R_T \mid S = s \right] = \mathbb{P} \left[\text{Black wins} \mid S = s \right]$$

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

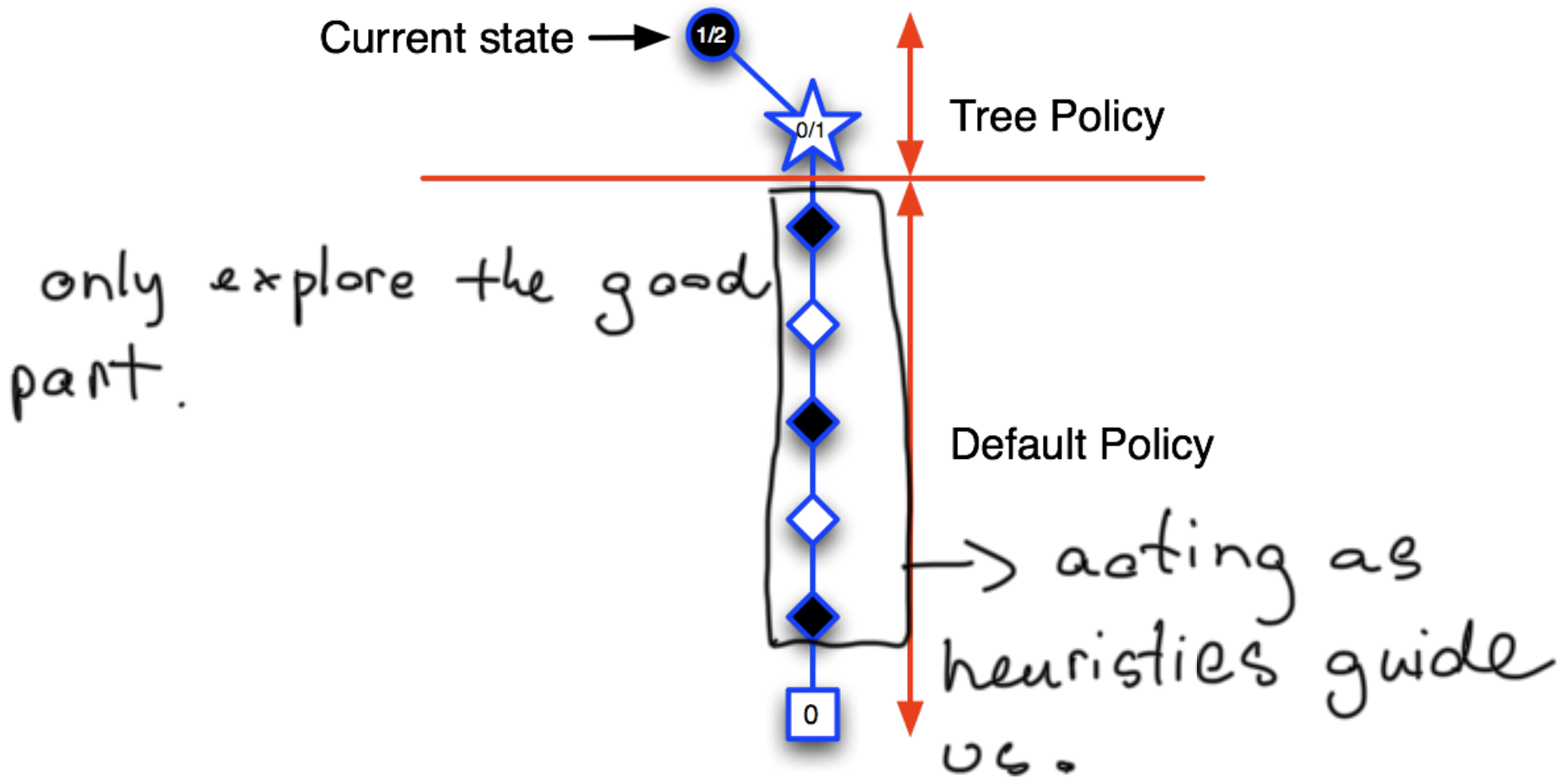
Monte-Carlo Evaluation in Go



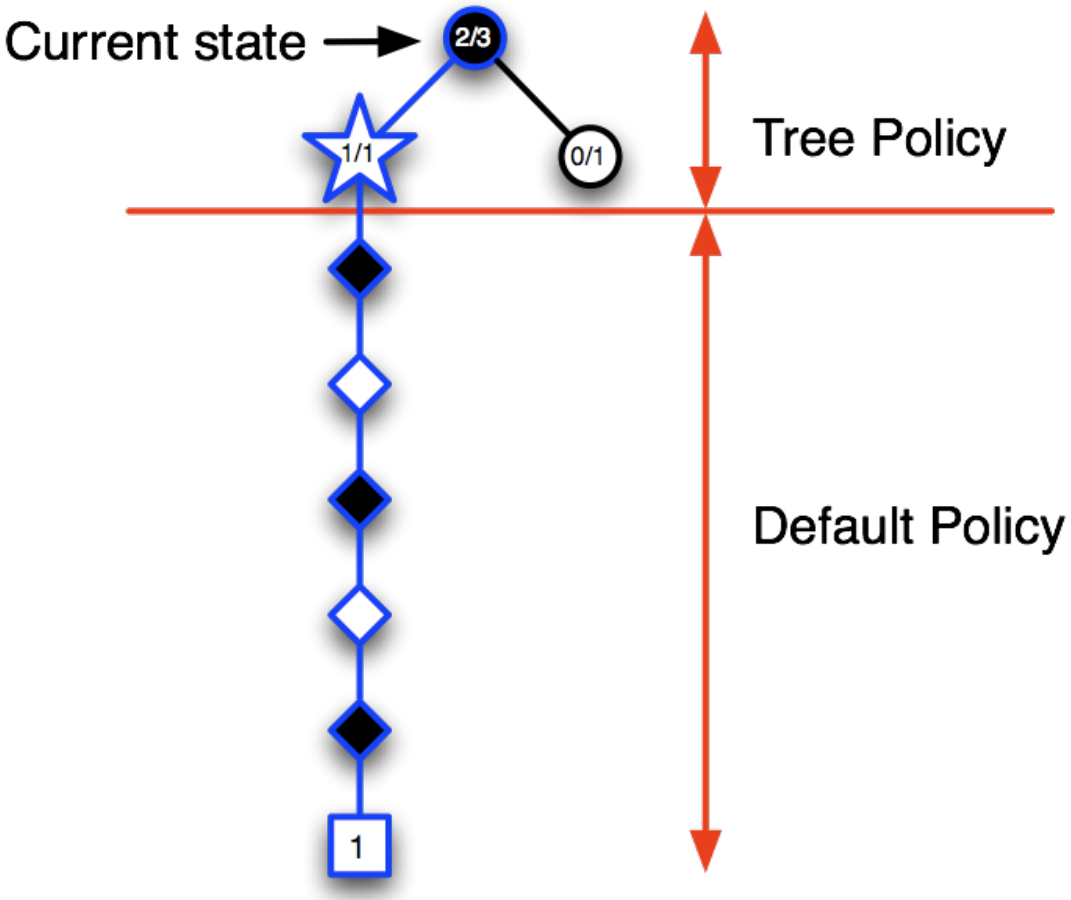
Applying Monte-Carlo Tree Search (1)



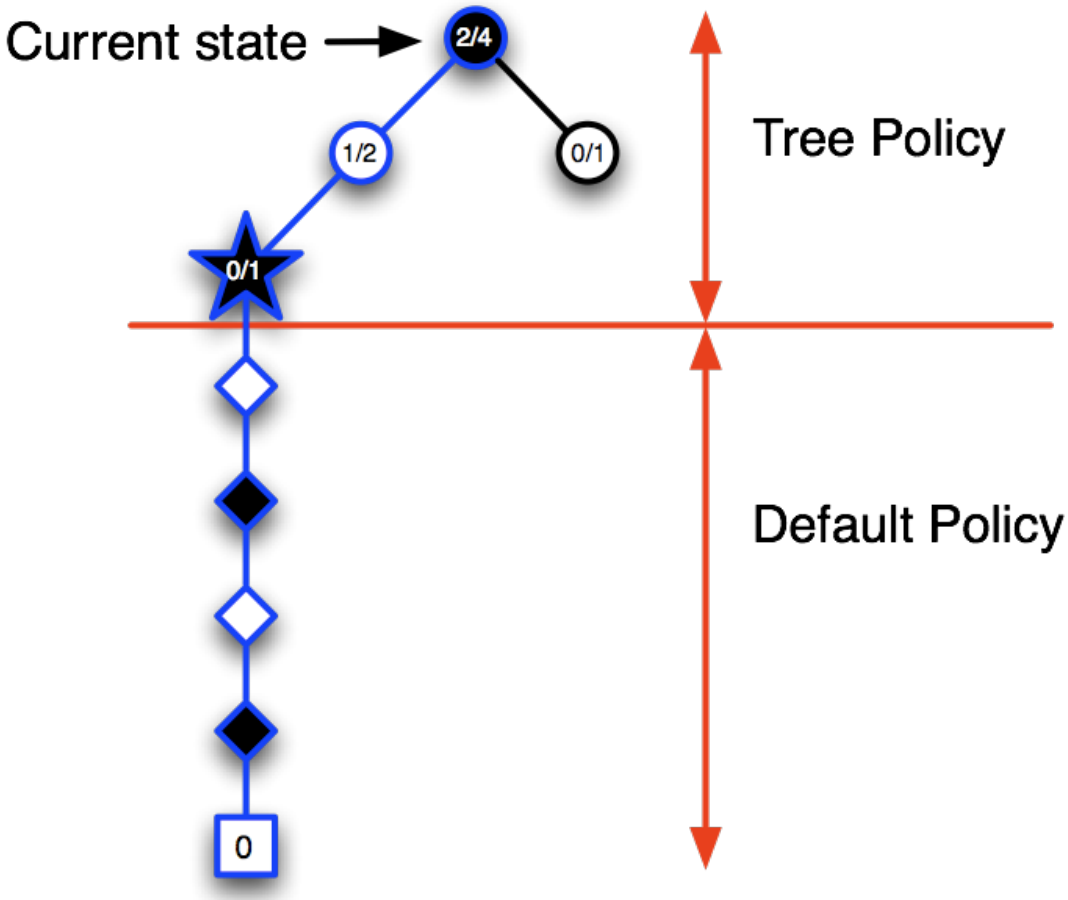
Applying Monte-Carlo Tree Search (2)



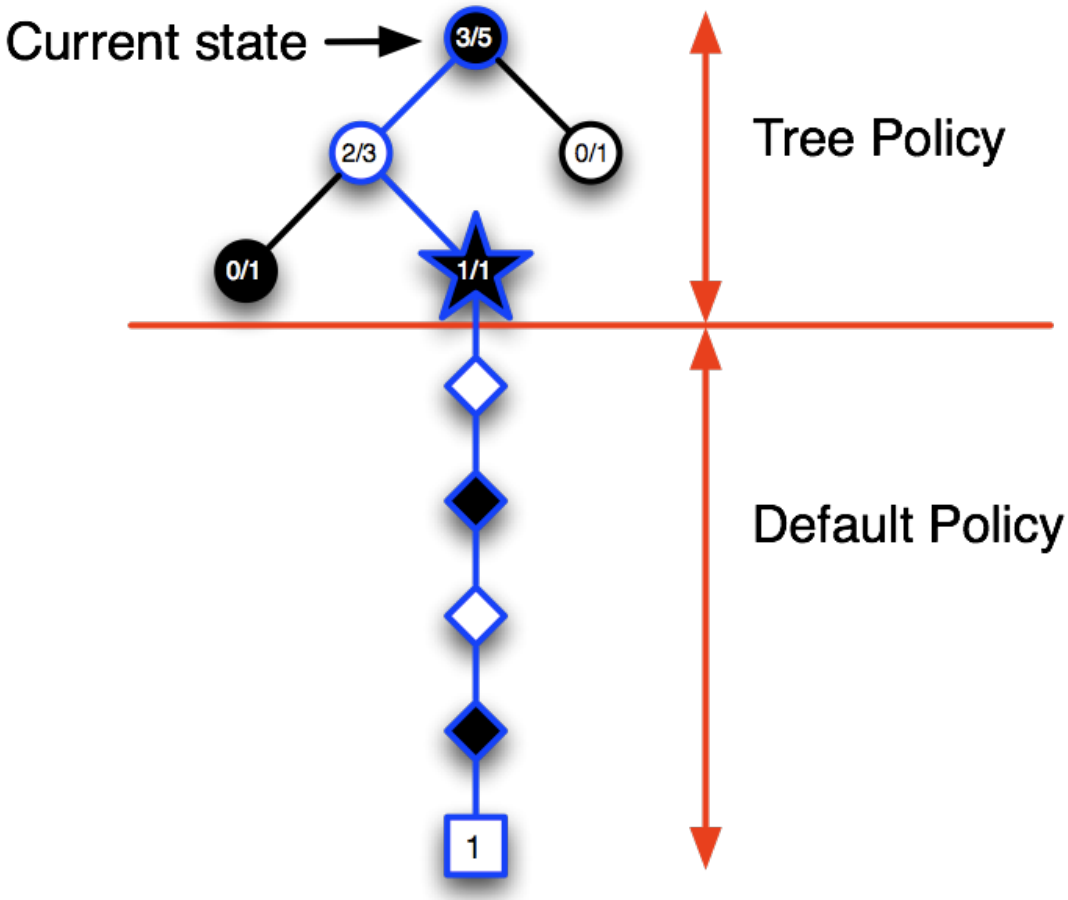
Applying Monte-Carlo Tree Search (3)



Applying Monte-Carlo Tree Search (4)



Applying Monte-Carlo Tree Search (5)



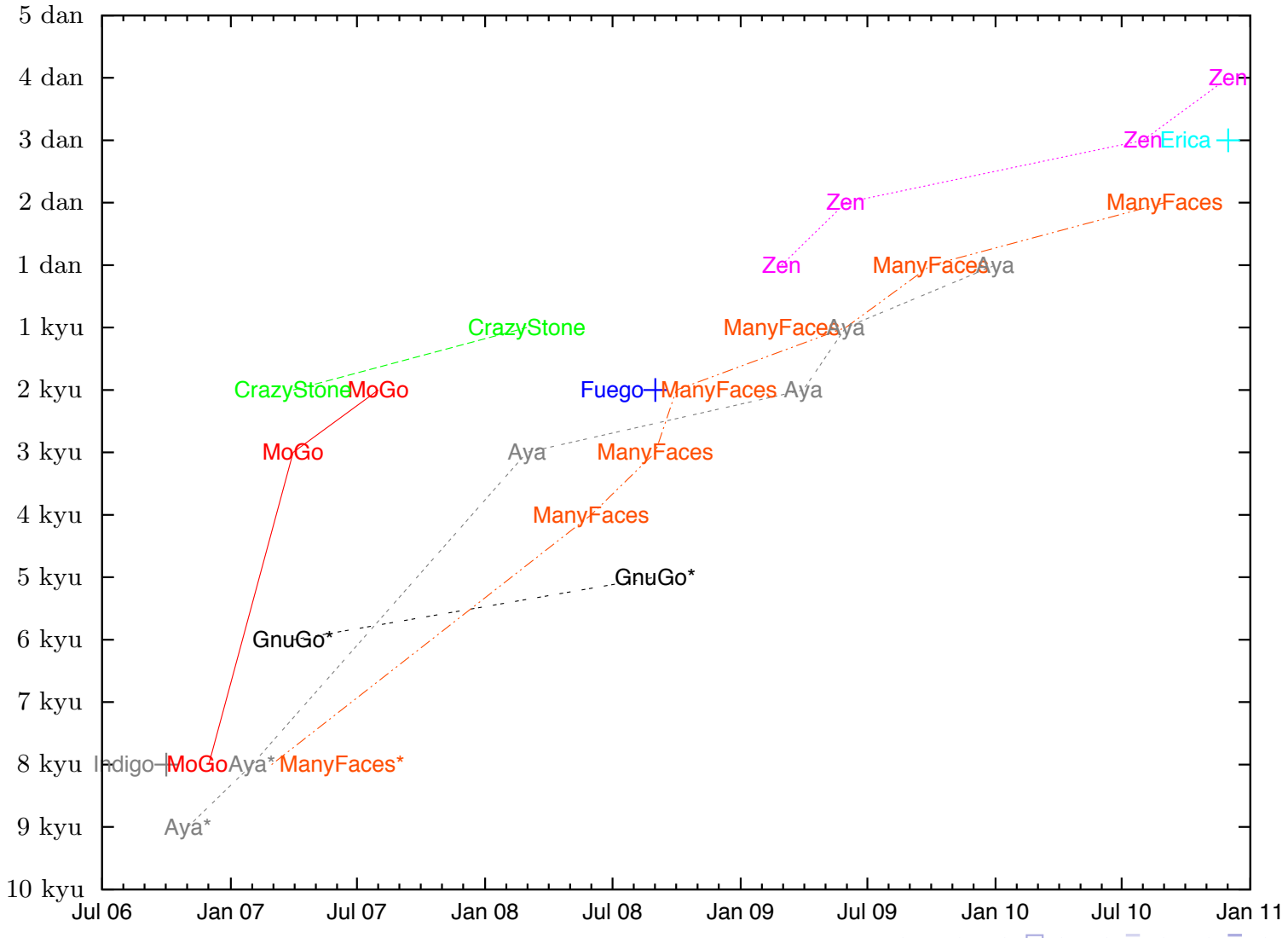
Advantages of MC Tree Search

- 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, parallelisable

dont have to consider all possible things.

Example: MC Tree Search in Computer Go

strength of model



Temporal-Difference Search

- Simulation-based search
- Using TD instead of MC (bootstrapping)
- MC tree search applies MC control to sub-MDP from now
- TD search applies Sarsa to sub-MDP from now

MC vs. TD search

- For model-free reinforcement learning, bootstrapping is helpful
  - TD learning reduces variance but increases bias
  - TD learning is usually more efficient than MC
  - TD(λ) can be much more efficient than MC
- For simulation-based search, bootstrapping is also helpful
  - TD search reduces variance but increases bias
  - TD search is usually more efficient than MC search
  - TD(λ) search can be much more efficient than MC search

TD Search

- Simulate episodes from the current (real) state s_t
- Estimate action-value function $Q(s, a)$
- For each step of simulation, update action-values by Sarsa

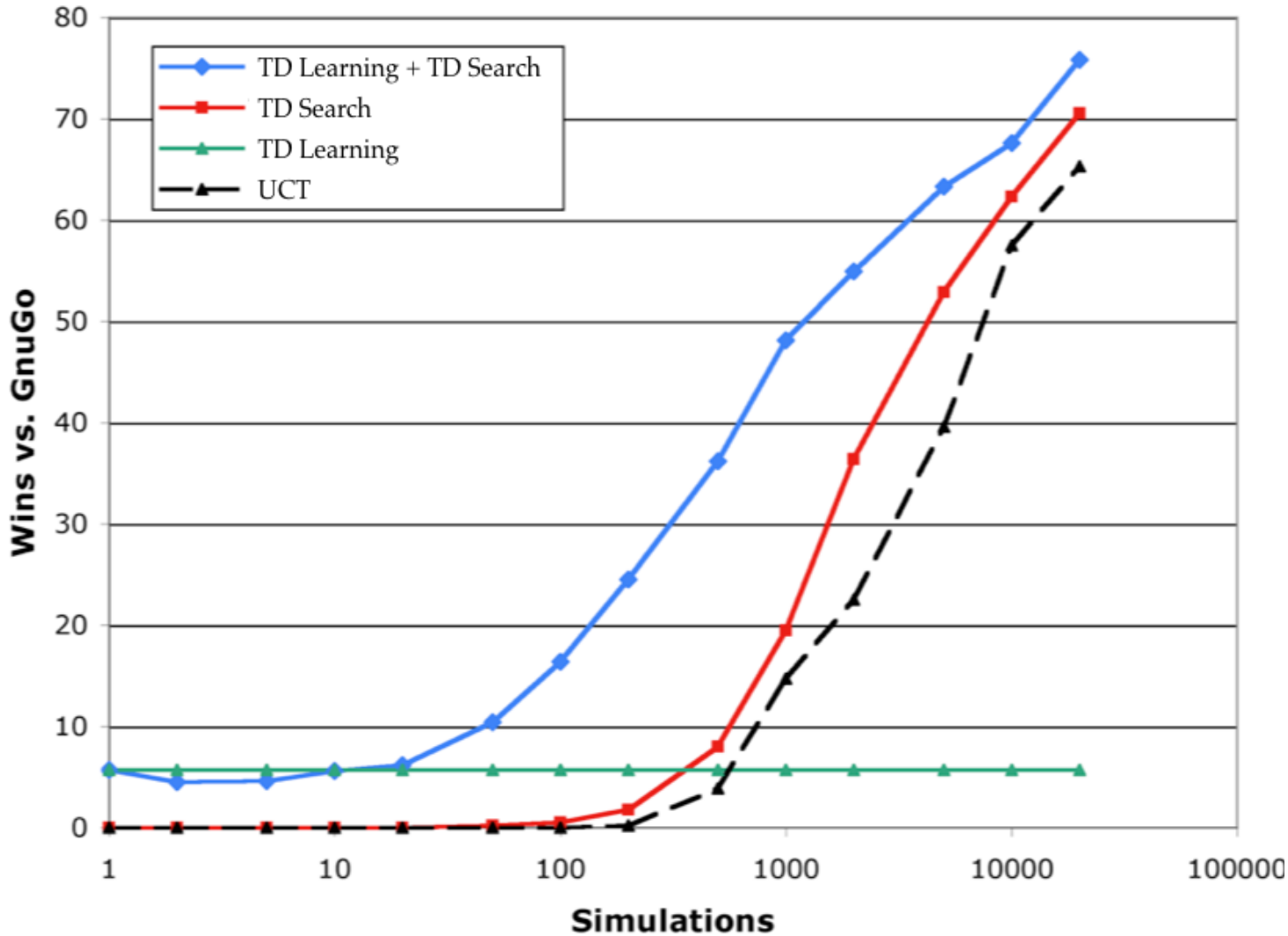
$$\Delta Q(S, A) = \alpha(R + \gamma Q(S', A') - Q(S, A))$$

- Select actions based on action-values $Q(s, a)$
  - e.g. ϵ-greedy
- May also use function approximation for Q

Dyna-2

- In Dyna-2, the agent stores two sets of feature weights
  - Long-term memory
  - Short-term (working) memory
- Long-term memory is updated from real experience using TD learning
  - General domain knowledge that applies to any episode
- Short-term memory is updated from simulated experience using TD search
  - Specific local knowledge about the current situation
- Over value function is sum of long and short-term memories

Results of TD search in Go



Questions?