10703 Deep Reinforcement Learning and Control

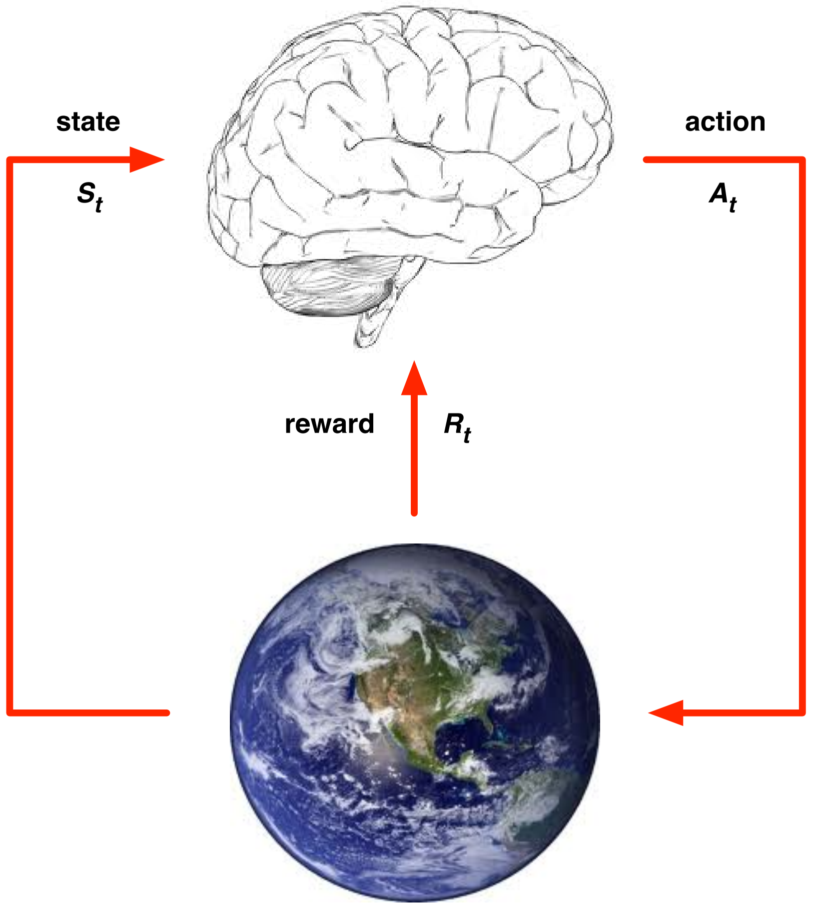
Russ Salakhutdinov

Slides borrowed from
Katerina Fragkiadaki

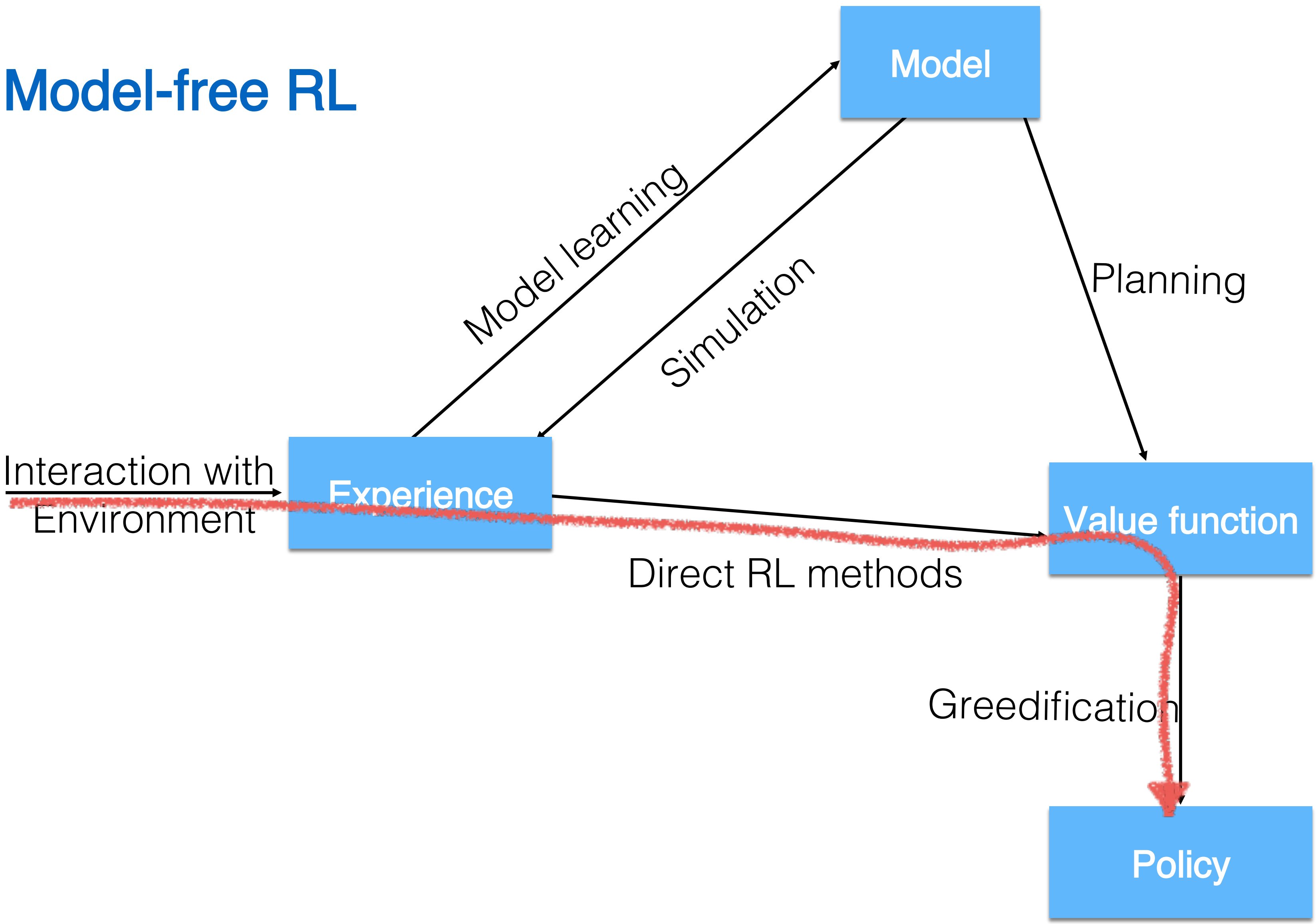
Learning and Planning with Tabular Methods

What can I learn by interacting with the world?

Past classes: the agent learned to estimate value functions and optimal policies from experience.



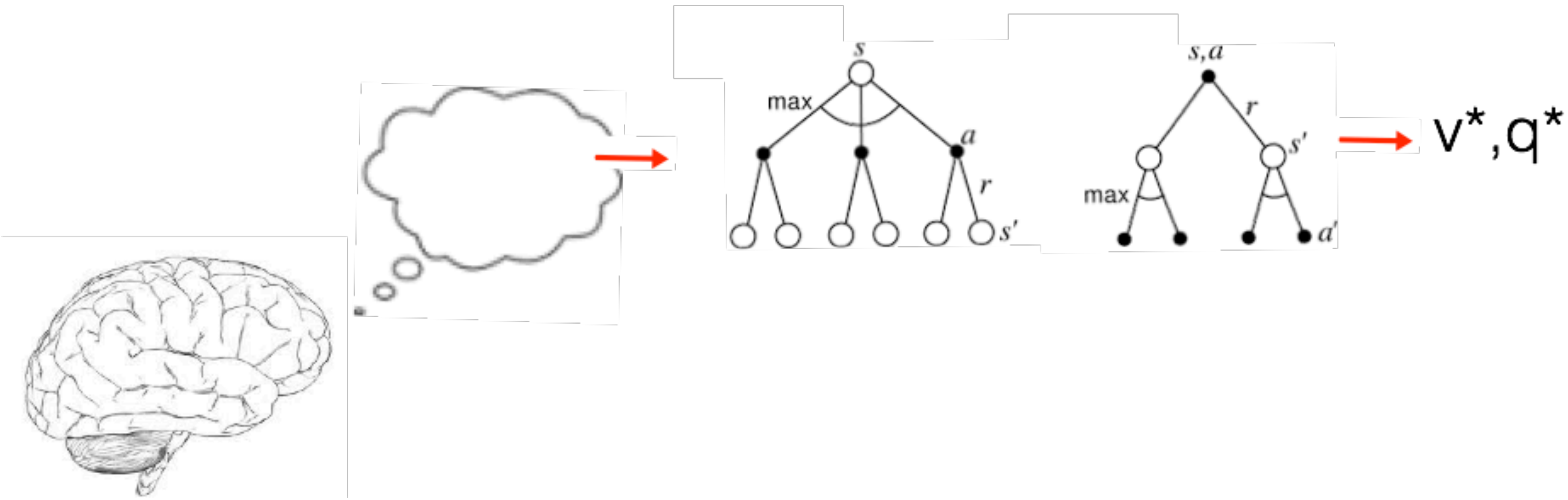
Model-free RL



What can I learn by interacting with the world?

-**So far:** We know the true environment (dynamics and rewards) and just use it to ***plan and estimate value functions*** (value iteration, policy iteration using exhaustive state sweeps of Bellman back-up operations).

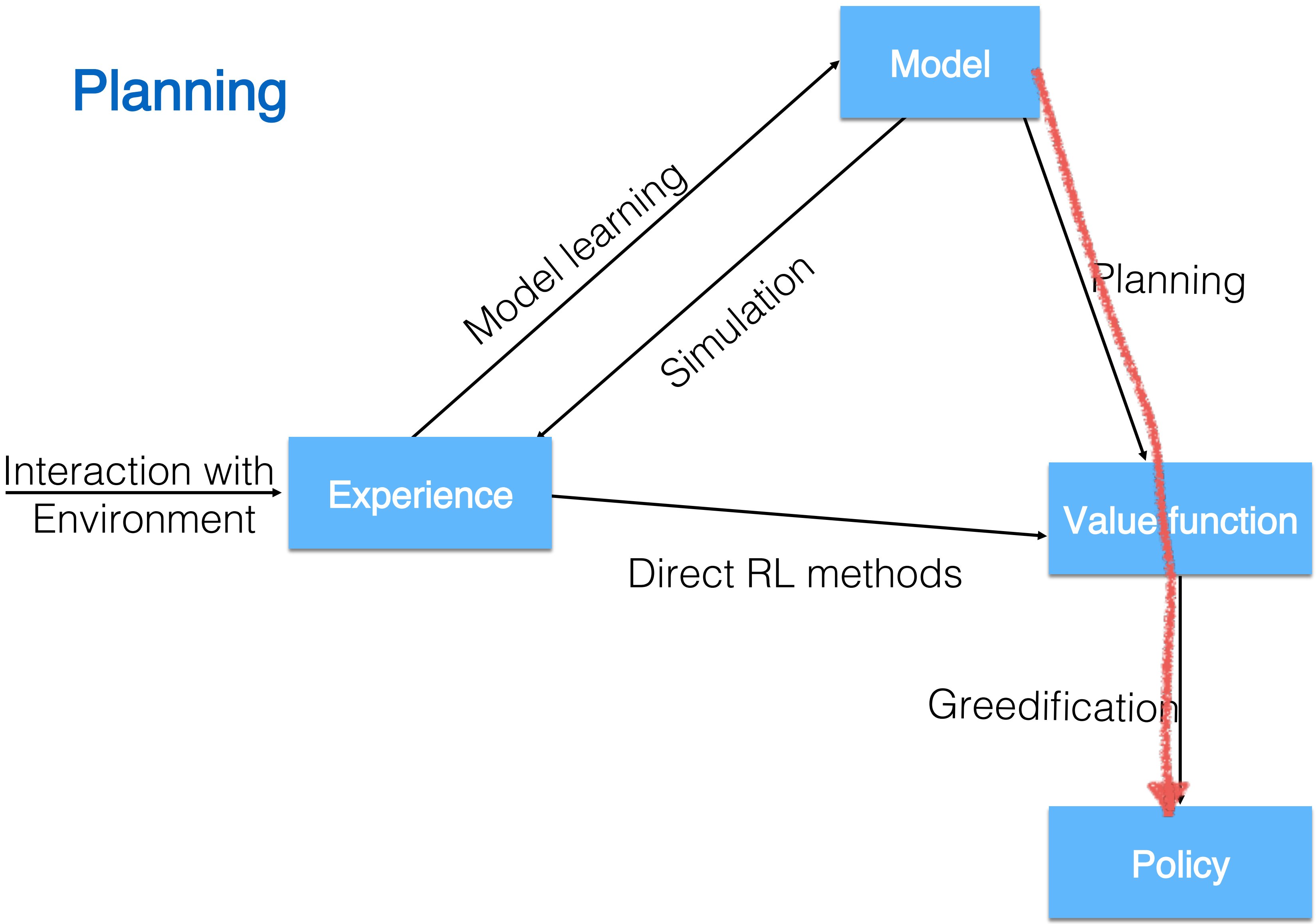
- Very slow when many states.



Planning: any computational process that uses a model to create or improve a policy

$$\text{Model} \xrightarrow{\text{Planning}} \text{Policy}$$

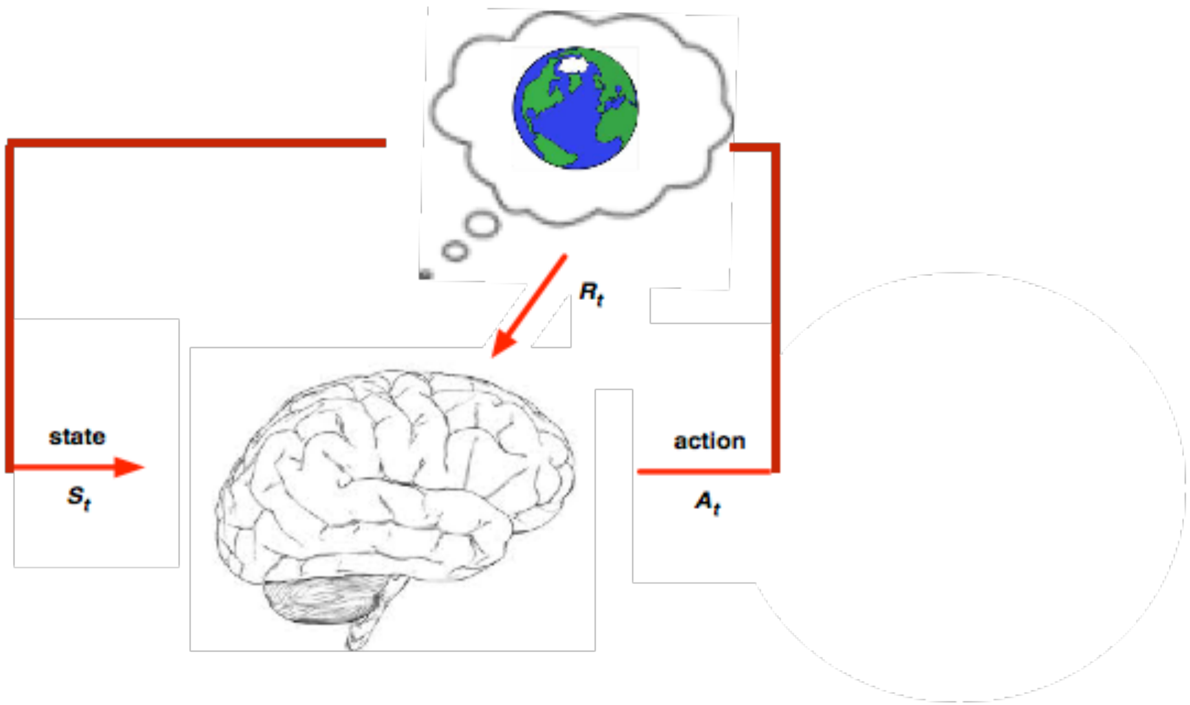
Planning


Model
Model learning
Simulation
Planning
Interaction with
Environment
Experience
Value function
Direct RL methods
Greedification
Policy

What can I learn by interacting with the world?

This lecture: we will combine both, learning from experience and planning:

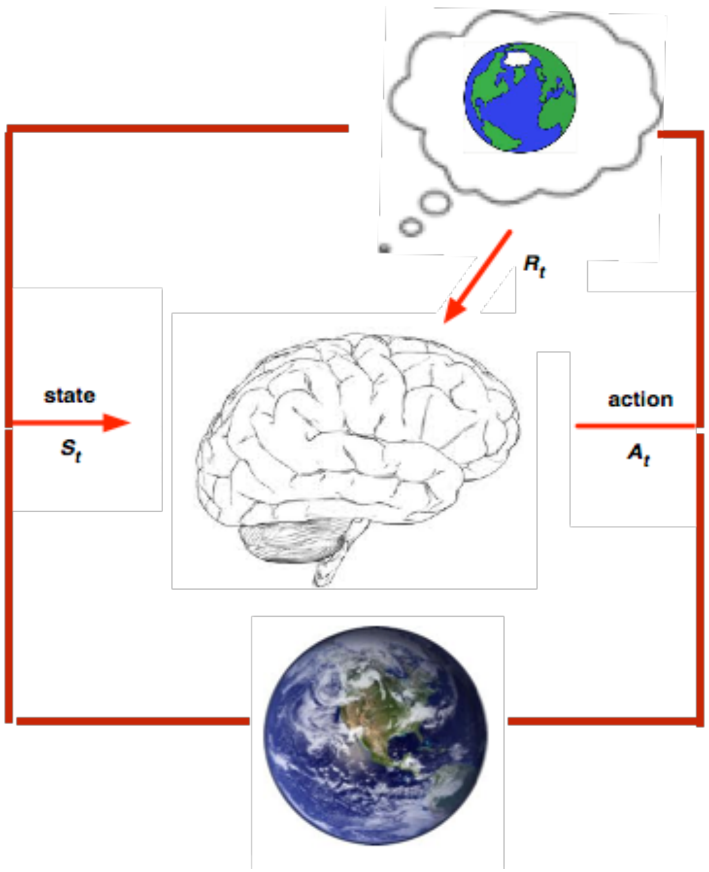
- If the model is unknown, we will learn the model.



What can I learn by interacting with the world?

This lecture: we will combine both, learning from experience and planning:

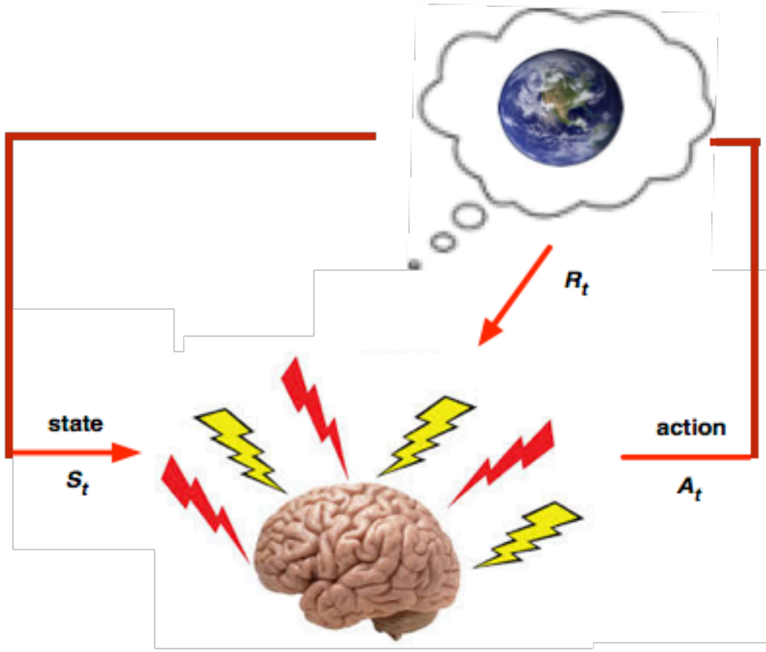
- If the model is unknown, we will learn the model
- Learn value functions using both real and simulated experience



What can I learn by interacting with the world?

This lecture: we will combine both, learning from experience and planning:

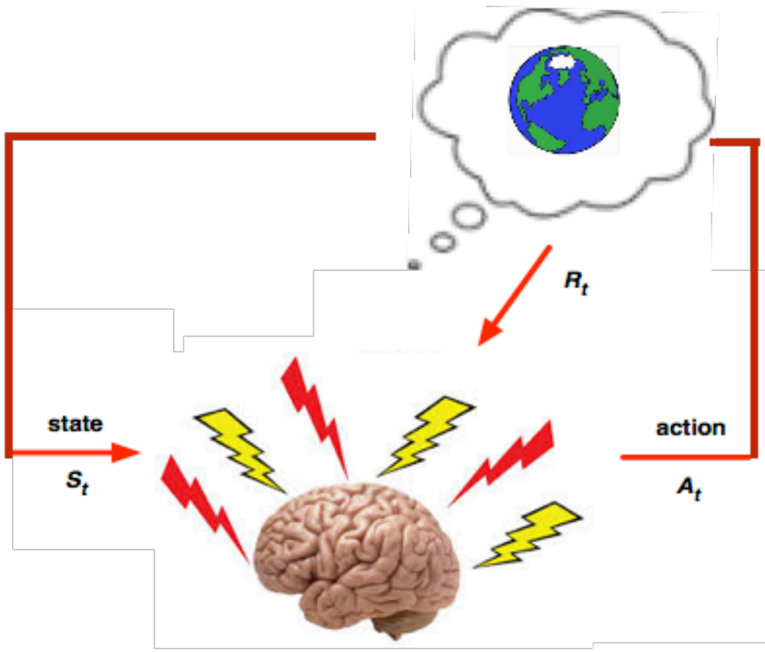
- If the model is unknown, we will learn the model
- Learn value functions using both real and simulated experience
- Learn value functions online using model-based look-ahead search



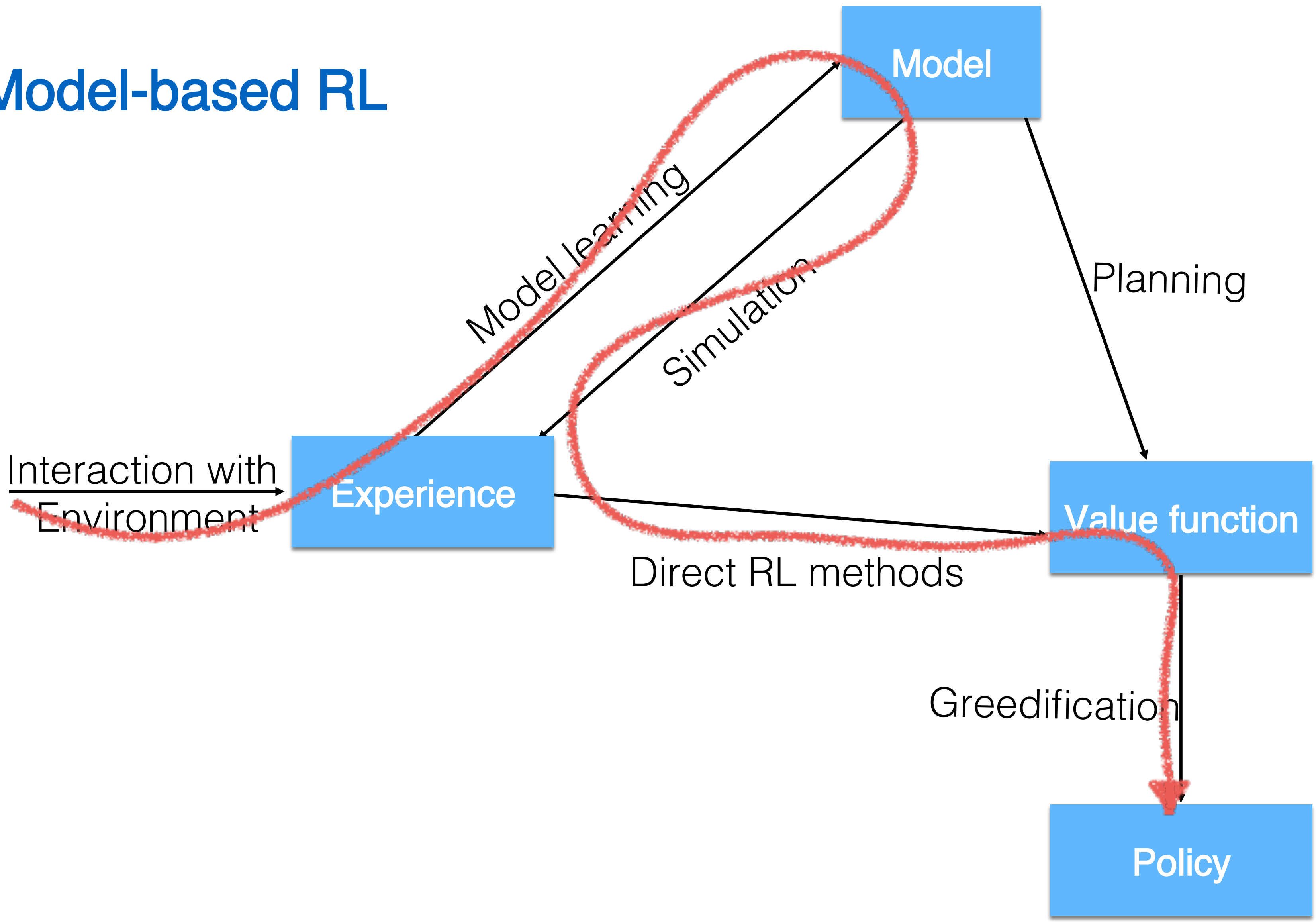
What can I learn by interacting with the world?

This lecture: we will combine both, learning from experience and planning:

- If the model is unknown, we will learn the model
- Learn value functions using both real and simulated experience
- Learn value functions online using model-based look-ahead search



Model-based RL



Advantages of Model-Based RL

Advantages:

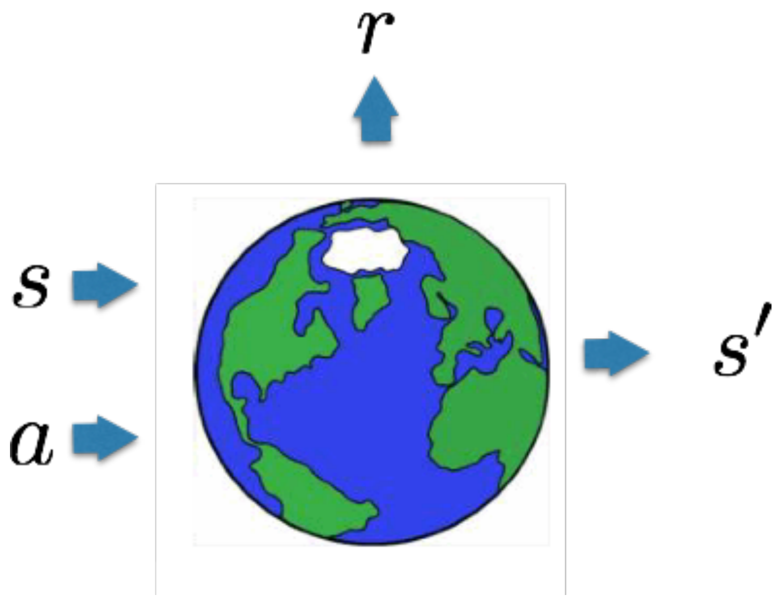
- Model learning transfers across tasks and environment configurations (learning physics)
- Better exploits experience in case of sparse rewards
- It is probably what the brain does (more later)
- Helps exploration: Can reason about model uncertainty

Disadvantages:

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

What is a Model?

Model: anything the agent can use to predict how the environment will respond to its actions: -- specifically, the transition (dynamics) T(s’|s,a) and reward functions R(s,a).



this includes transitions of the state of the environment and the state of the agent.

What is a Model?

Model: anything the agent can use to predict how the environment will respond to its actions, specifically:

- the transition function (dynamics)
- reward function

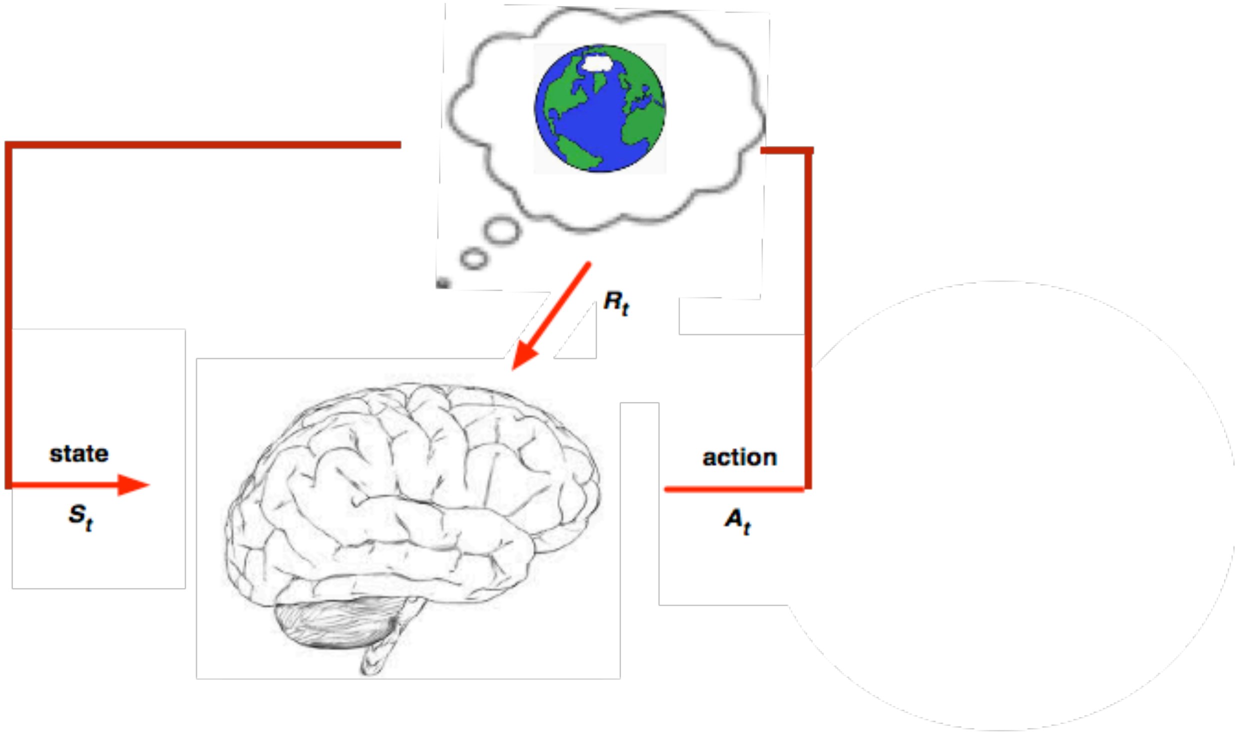
Distribution model: description of all possibilities and their probabilities, $T(s'|s,a)$ for all (s, a, s')

Sample model, a.k.a. a simulation model: produces sample experiences for a given ***s,*** often much easier to come by

Both types of models can be used to produce hypothetical experience (what if…)

Model Learning

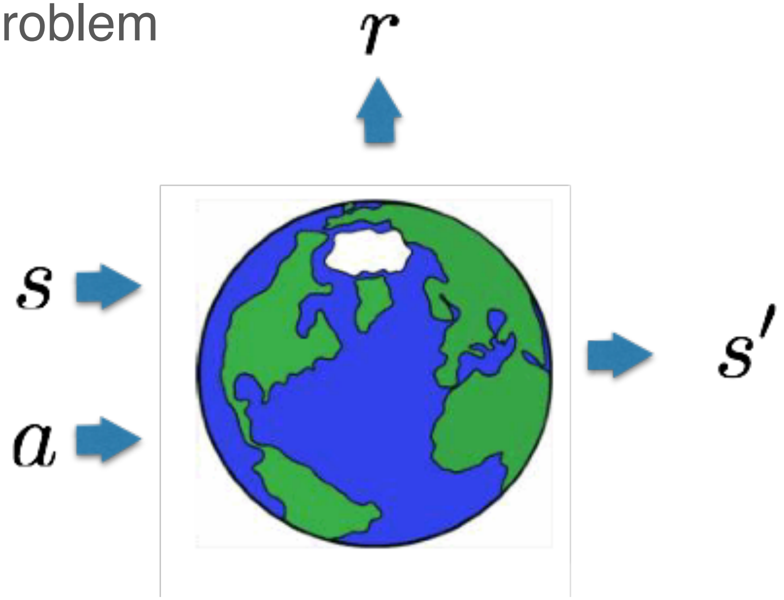
- If the model is unknown, we will learn the model.
- Learn value functions using both real and simulated experience
- Learn value functions online using model-based lookahead search



Model Learning

- Goal: estimate model $\mathcal{M}_\eta$ from experience $\{\mathcal{S}_1, \mathcal{A}_1, R_2, ..., \mathcal{S}_T\}$
- This can be thought as a supervised learning problem

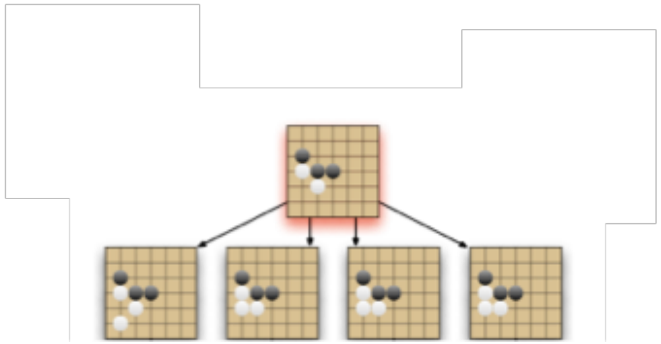
$$\mathcal{S}_1, \mathcal{A}_1 \rightarrow R_2, \mathcal{S}_2$$
$$\mathcal{S}_2, \mathcal{A}_2 \rightarrow R_3, \mathcal{S}_3$$
$$\vdots$$
$$\mathcal{S}_{T-1}, \mathcal{A}_{T-1} \rightarrow R_T, \mathcal{S}_T$$



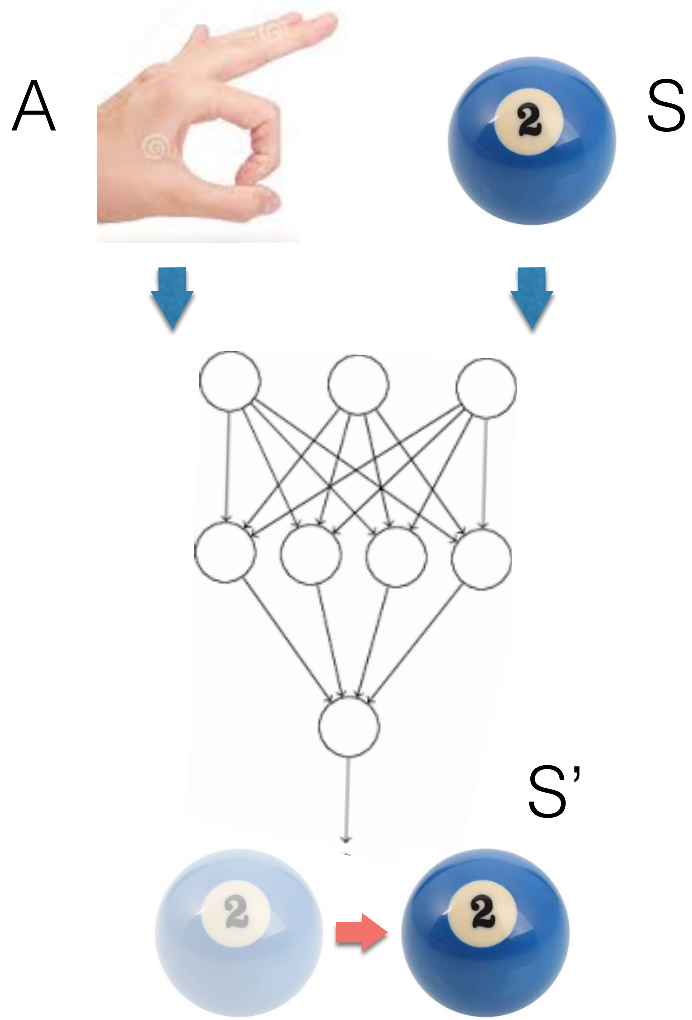
- Learning $s, a \rightarrow r$ is a *regression* problem
- Learning $s, a \rightarrow s'$ is a *density estimation* problem
- Pick loss function, e.g. mean-squared error, KL divergence
- Find parameters η that minimize empirical loss

Examples of Models for T(s'|s,a)

Table lookup model (tabular): book-keeping a probability of occurrence for each transition (s,a,s')

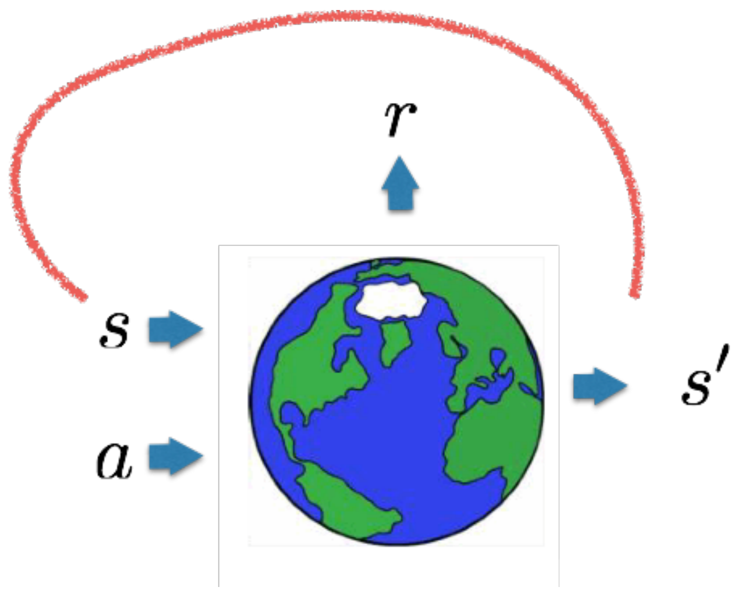


Transition function is approximated through some function approximator



A supervised learning problem?

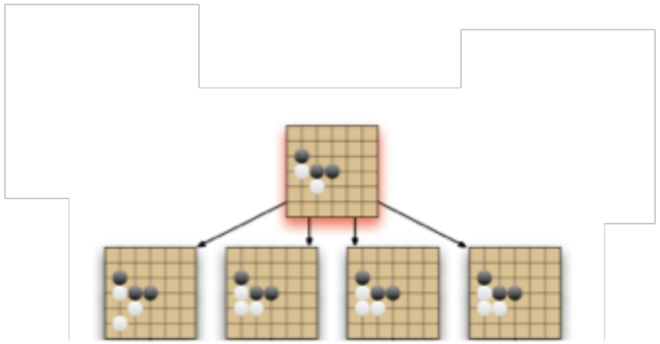
- To look ahead far in the future you need to chain your dynamic predictions
- Data is sequential
- i.i.d. assumptions break
- Errors accumulate in time



- Solutions:
  - Hierarchical dynamics models
  - Linear local approximations

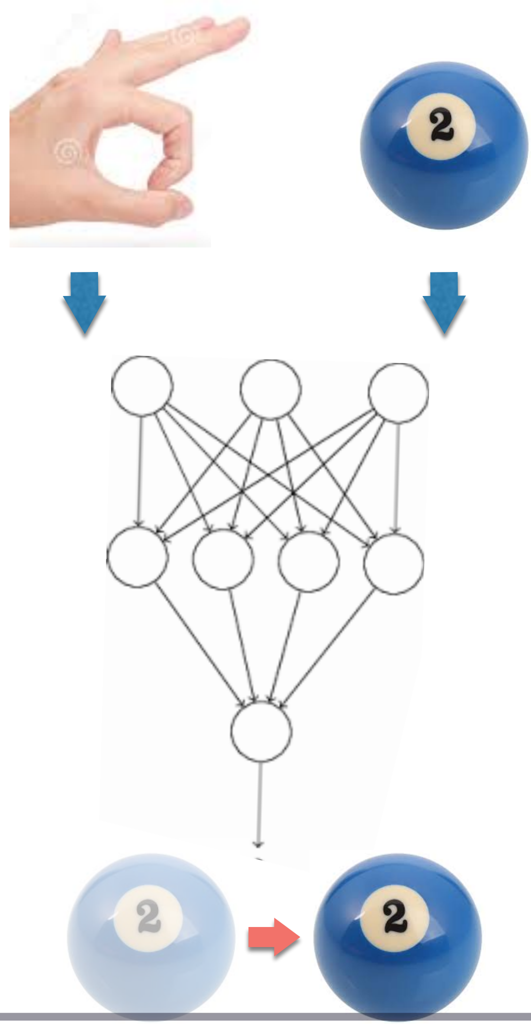
Examples of Models for T(s'|s,a)

Table lookup model (tabular): bookkeeping a probability of occurrence for each transition (s,a,s')



This Lecture

Transition function is approximated through some features



Later..

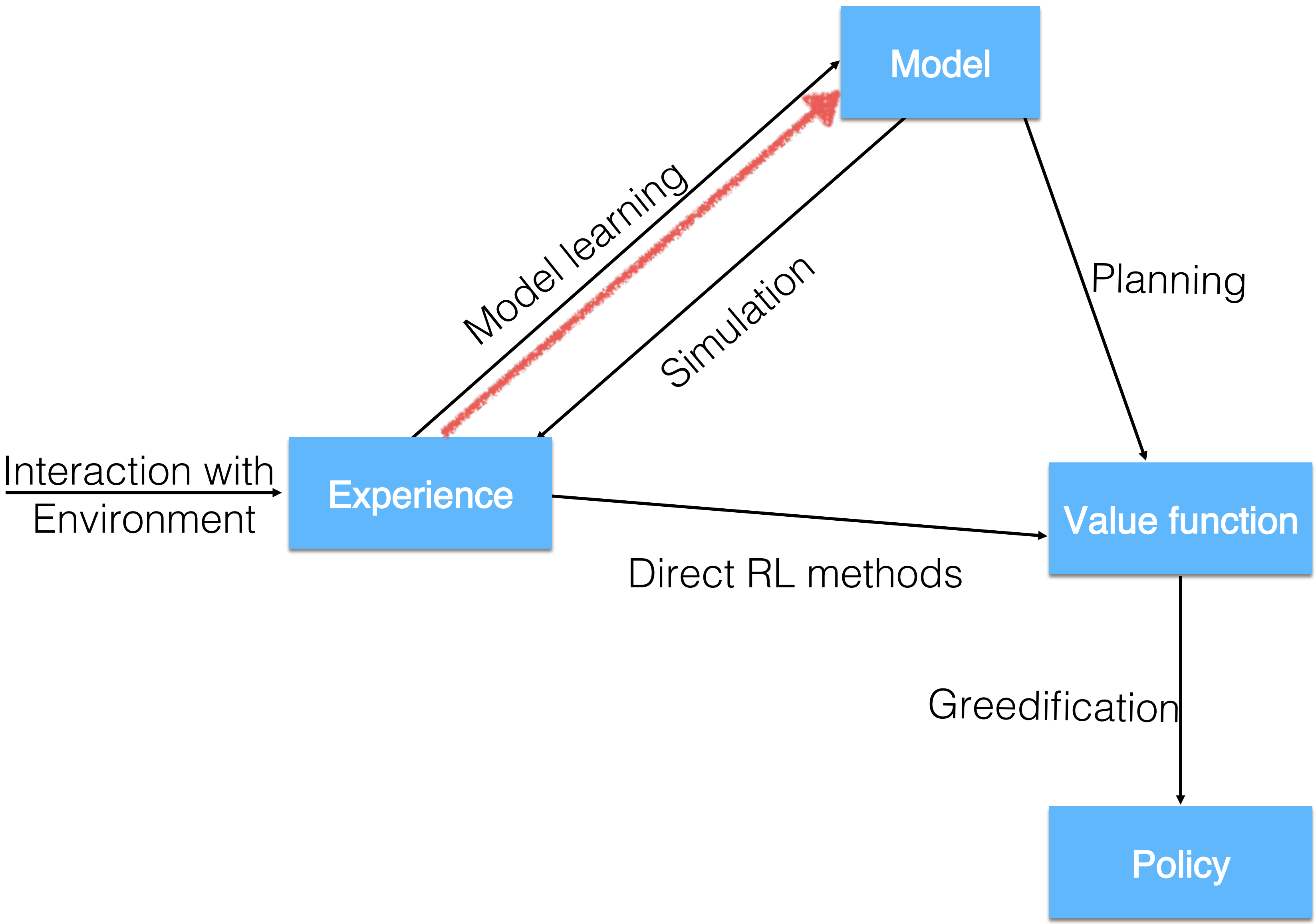

Model
Model learning
Simulation
Planning
Interaction with
Environment
Experience
Value function
Direct RL methods
Greedification
Policy

Table Lookup Model

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

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

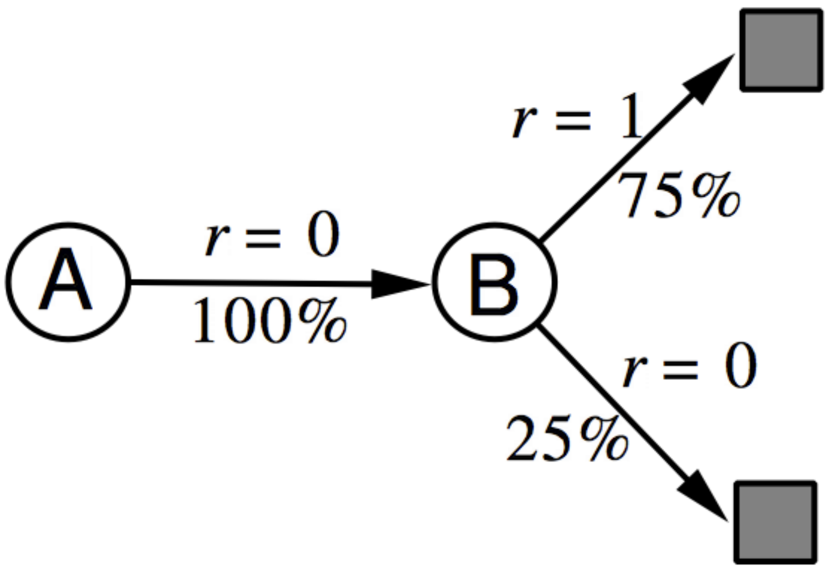
$$\hat{R}(s,a) = \frac{1}{N(s,a)} \sum_{t=1}^{\mathcal{T}} \mathbf{1}(S_t, A_t = s, a) R_t$$

- 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 $\langle s, a, \cdot, \cdot \rangle$

A simple 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 $\mathcal{M}_\eta = \langle T_\eta, \mathcal{R}_\eta \rangle$

Solve the MDP $\langle \mathcal{S}, \mathcal{A}, T_\eta \mathcal{R}_\eta \rangle$

Using favorite planning algorithm

- Value iteration
- Policy iteration
- Tree search

curse of dimensionality!

Planning with a Model

Given a model $\mathcal{M}_\eta = \langle T_\eta, \mathcal{R}_\eta \rangle$

Solve the MDP $\langle \mathcal{S}, \mathcal{A}, T_\eta \mathcal{R}_\eta \rangle$

Using favorite planning algorithm

- Value iteration
- Policy iteration
- Tree search
- Sample-based planning

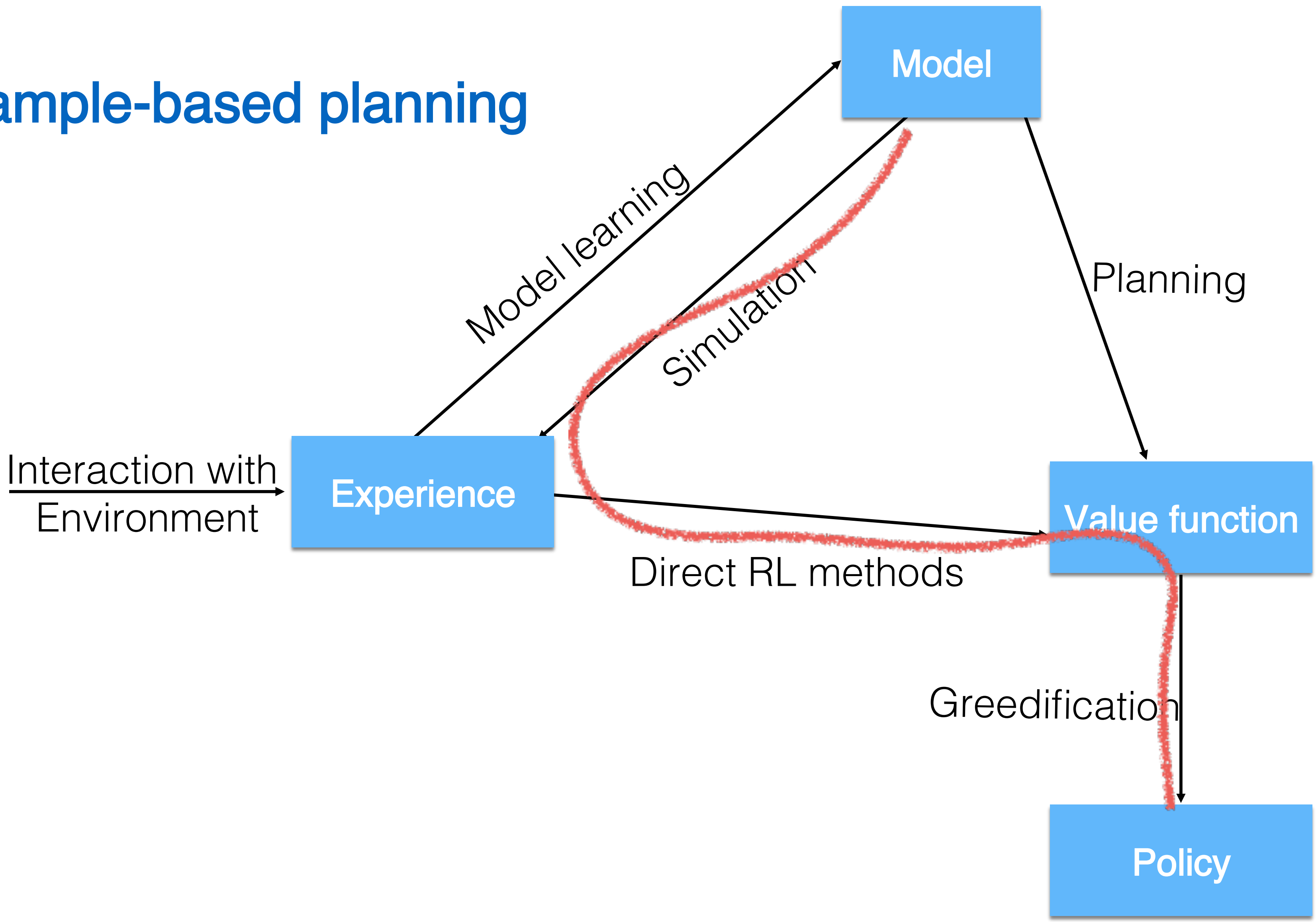
Sample-based Planning

- Use the model only to generate samples, not using its transition probabilities and expected immediate rewards
- Sample experience from model

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

- Apply model-free RL to samples, e.g.:
  - Monte-Carlo control
  - Sarsa
  - Q-learning
- Sample-based planning methods are often more efficient: rather than exhaustive state sweeps: we focus on what is likely to happen

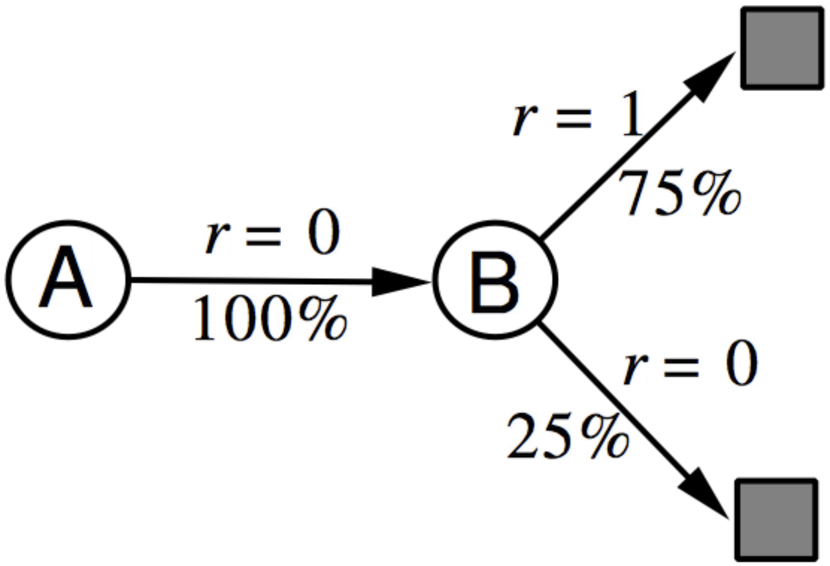
Sample-based planning



A Simple Example

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

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



e.g. Monte-Carlo learning: $\mathrm{v}(A) = 1, \mathrm{v}(B) = 0.75$

Planning with an Inaccurate Model

Given an imperfect model $< \mathcal{T}_\eta, \mathcal{R}_\eta > \neq < \mathcal{T}, \mathcal{R} >$

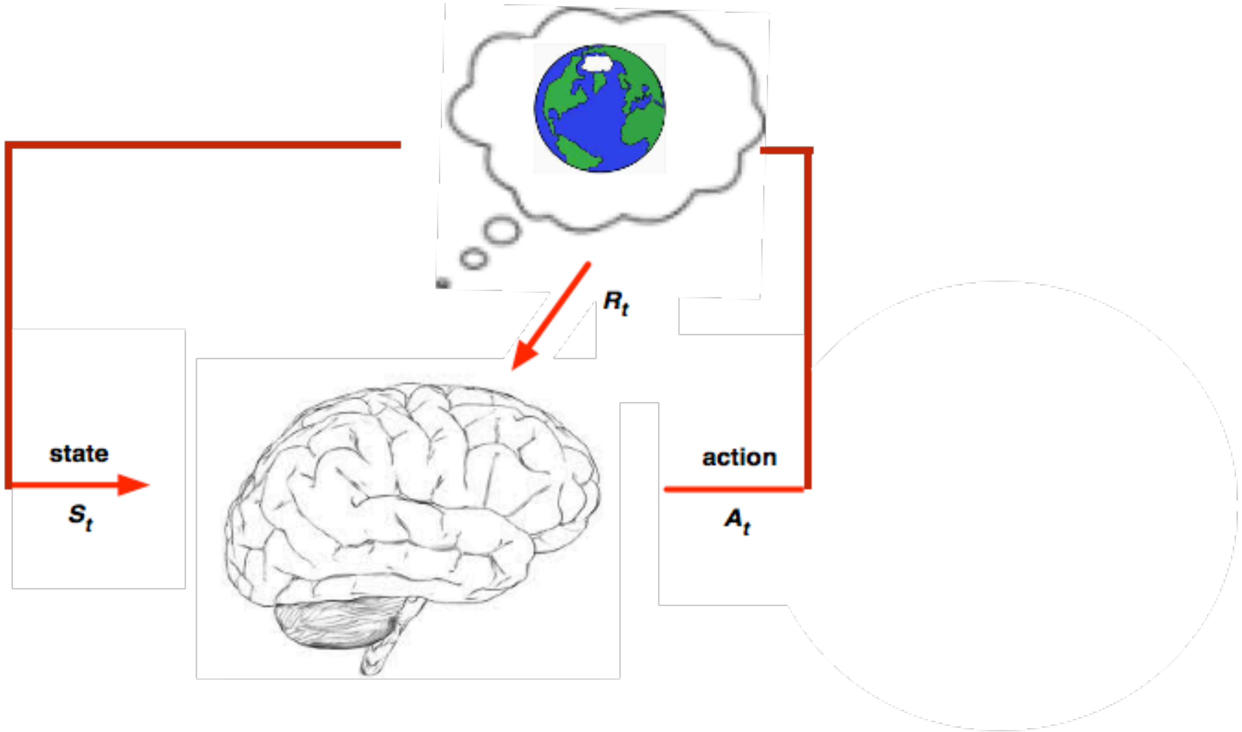
- Performance of model-based RL is limited to optimal policy for approximate MDP $< \mathcal{S}, \mathcal{A}, \mathcal{T}_\eta, \mathcal{R}_\eta >$
- i.e. Model-based RL is only as good as the estimated model

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

Combine real and simulated experience

- If the model is unknown, we will learn the model.
- Learn value functions using both real and simulated experience
- Learn value functions online using model-based lookahead search



Real and Simulated Experience

We consider two sources of experience

- Real experience - Sampled from environment (true MDP)

$$S' \sim T(s'|s, a)$$

$$R = r(s, a)$$

- Simulated experience - Sampled from model (approximate MD)

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

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

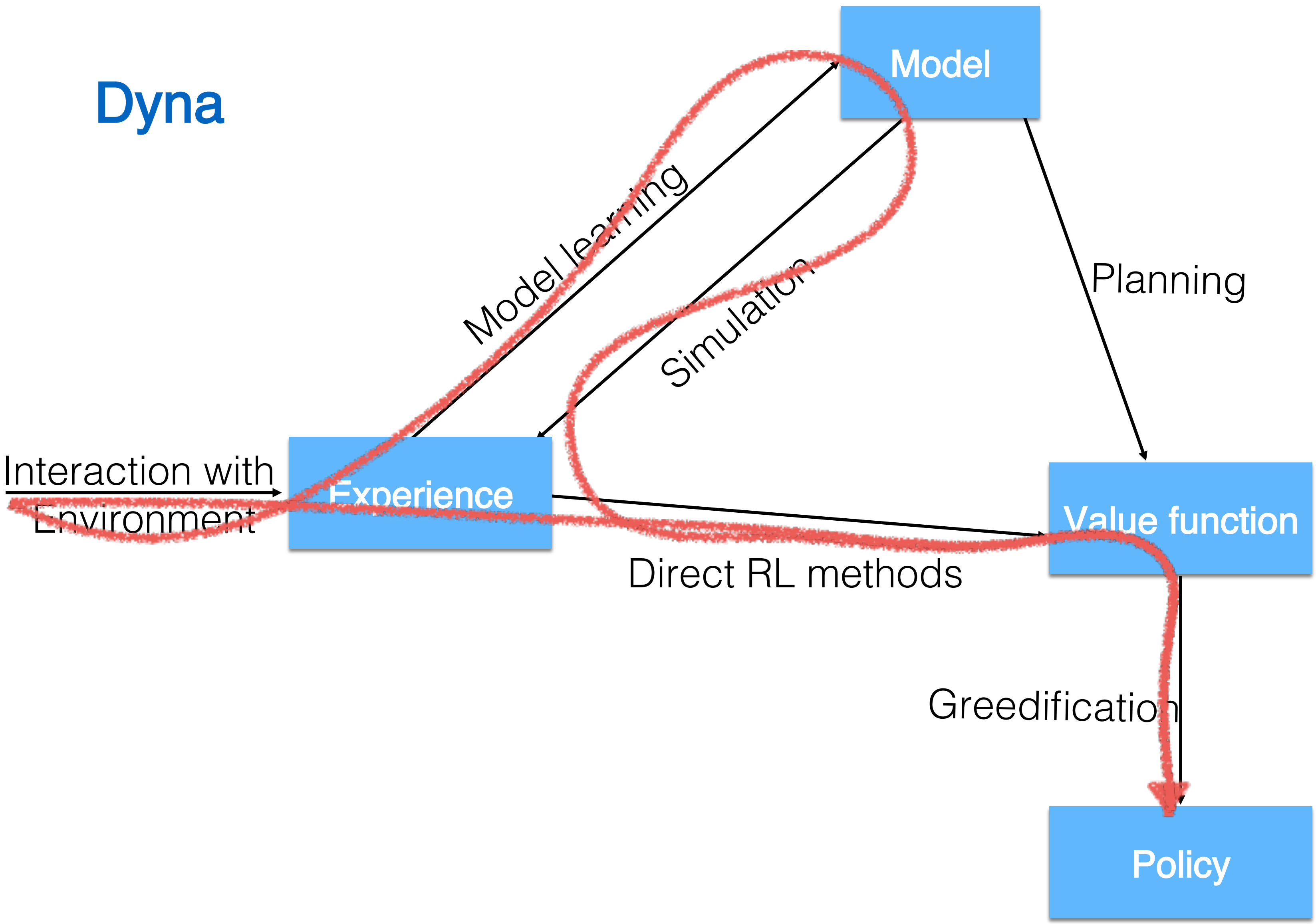
Model-Based RL (using Sample-Based Planning)

- 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

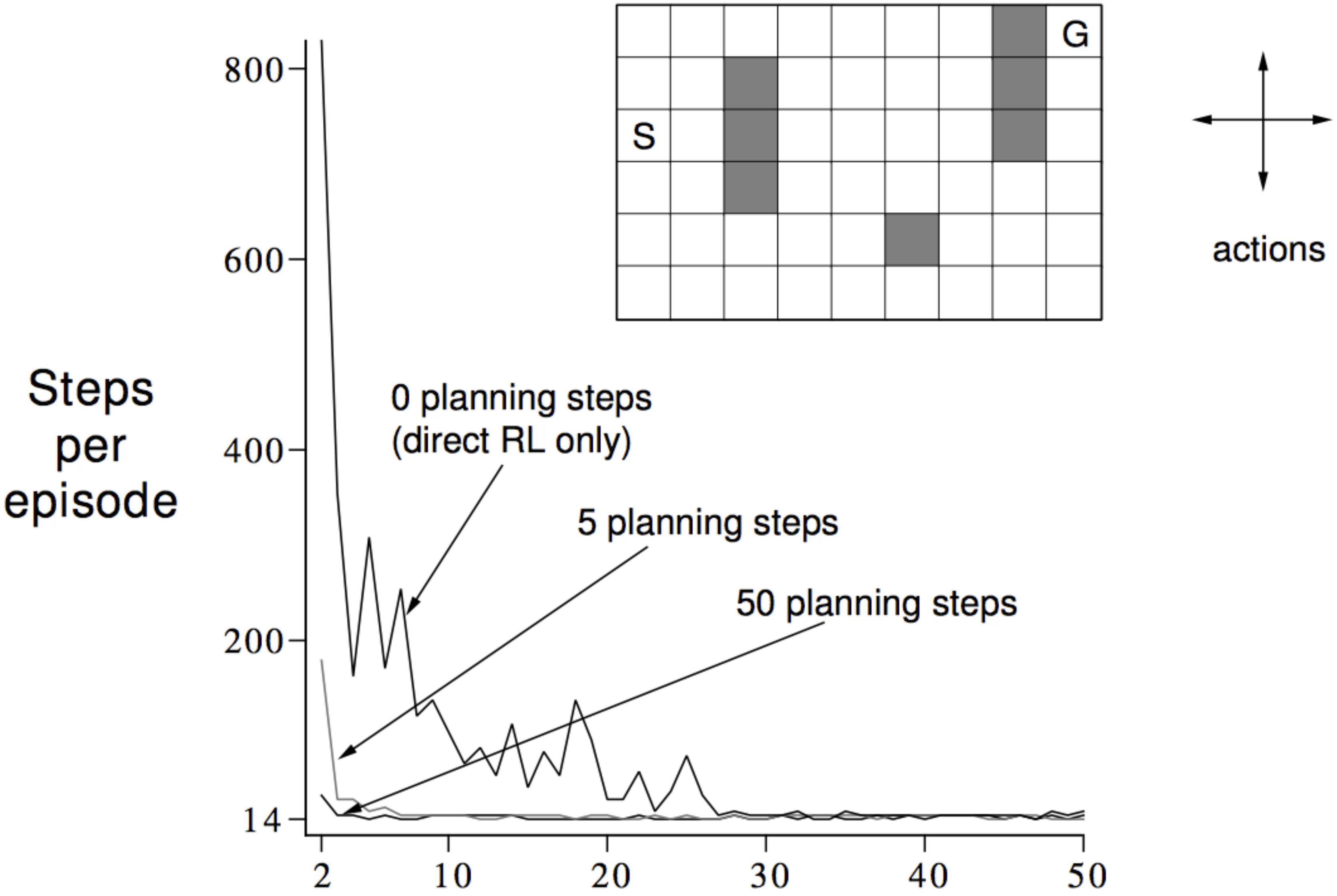


Model
Model learning
Simulation
Planning
Interaction with Environment
Experience
Value function
Direct RL methods
Greedification
Policy

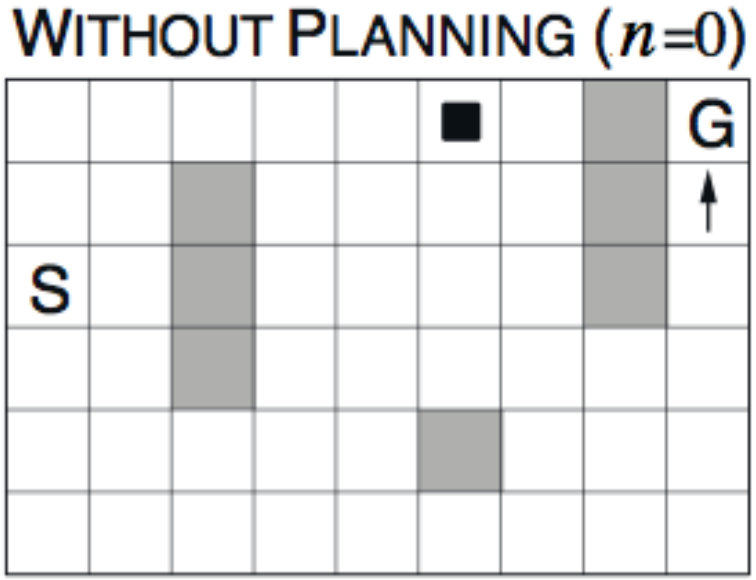
Dyna-Q Algorithm

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) $A \leftarrow \varepsilon$-greedy(S, Q)
  (c) Execute action A; observe resultant reward, R, and state, S'
  (d) $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
  (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
    $R, S' \leftarrow Model(S, A)$
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$

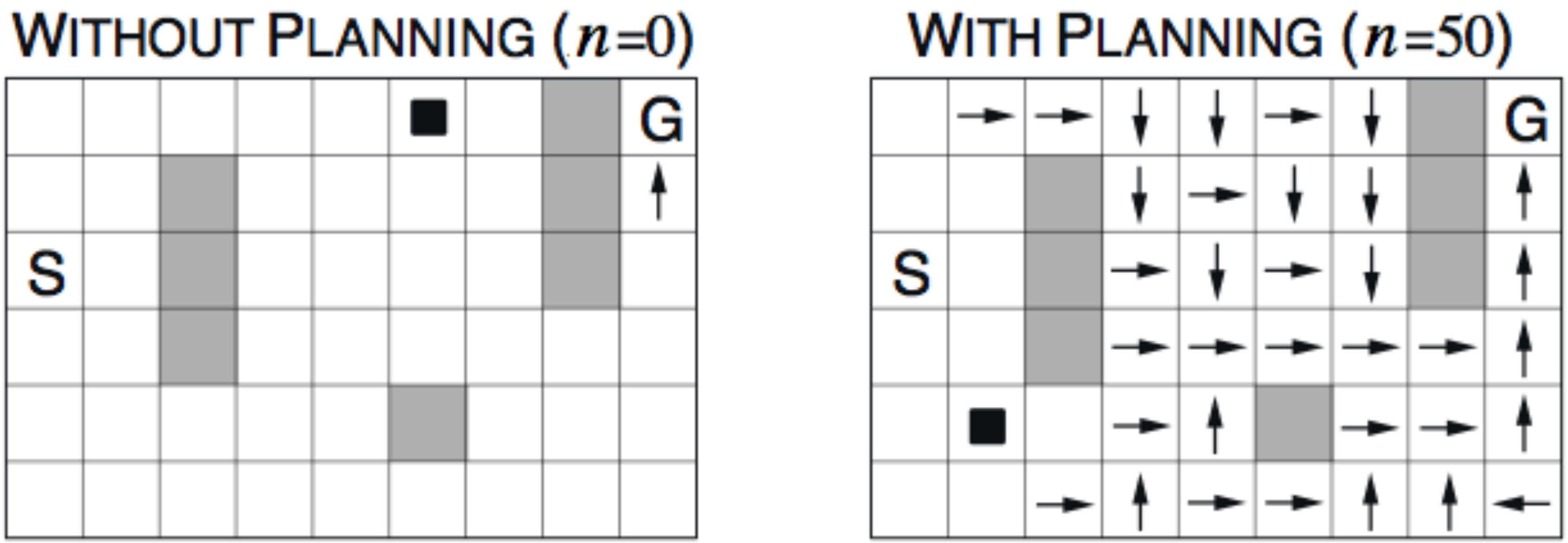
Dyna-Q on a Simple Maze



Midway in 2nd Episode

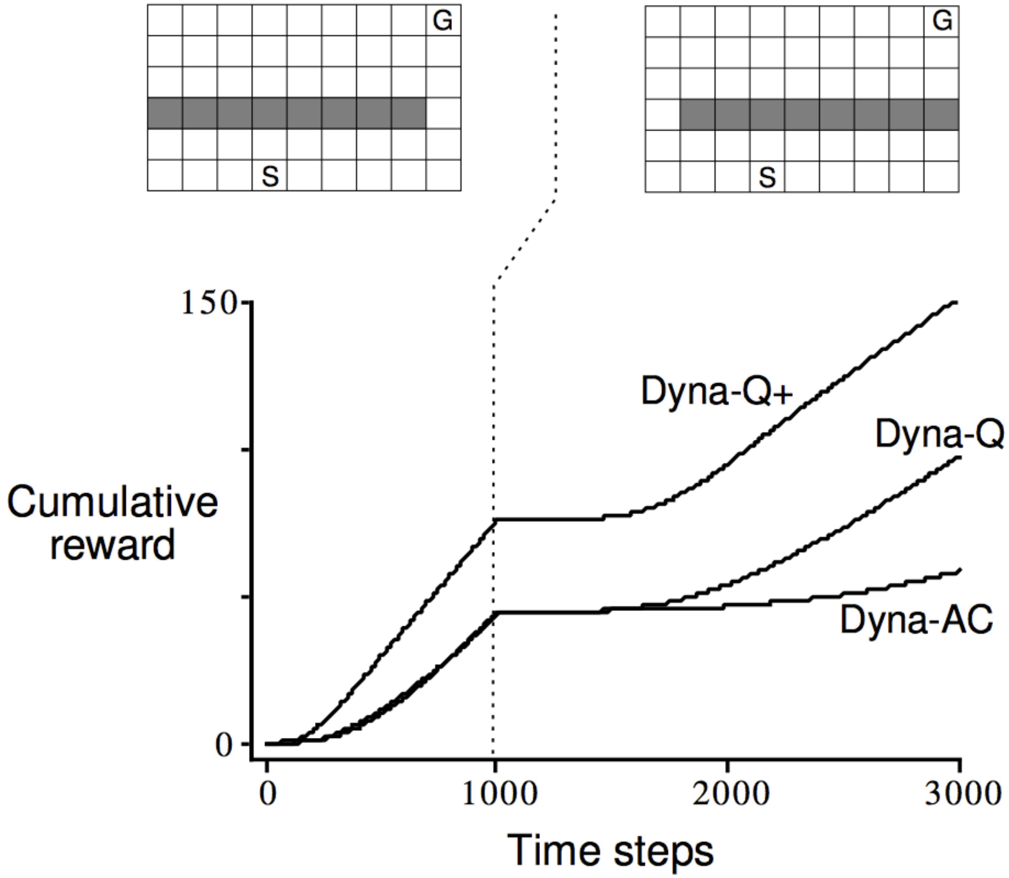


Midway in 2nd Episode



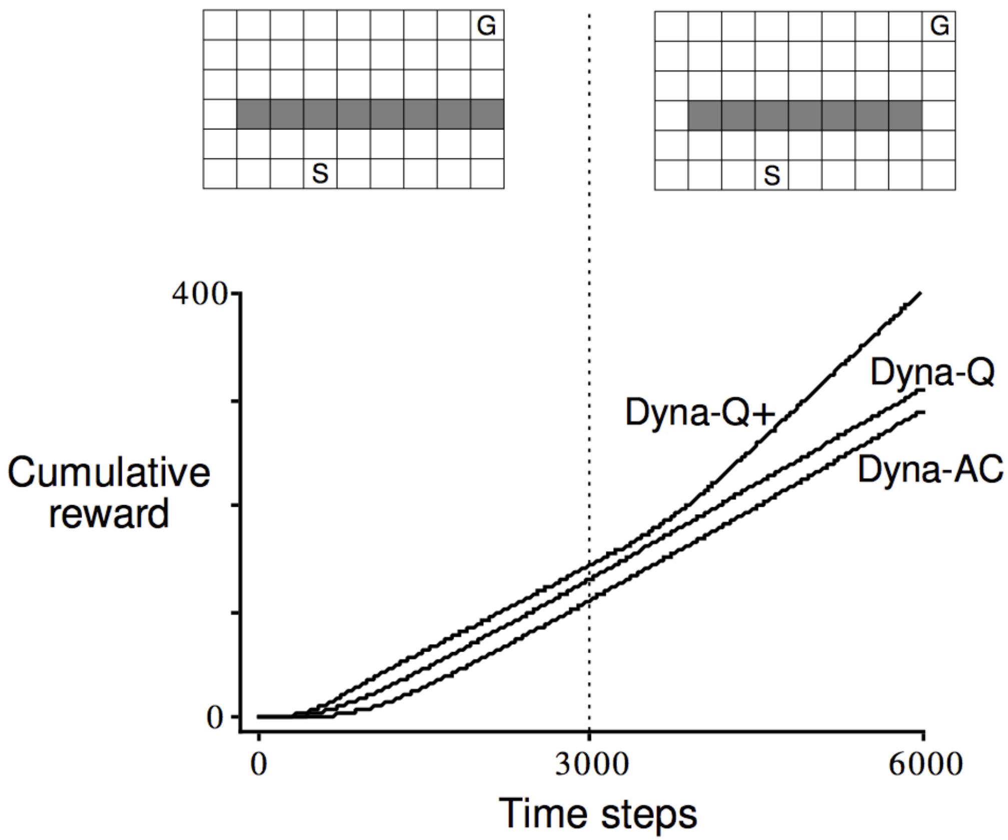
Dyna-Q with an Inaccurate Model

- The changed environment is harder



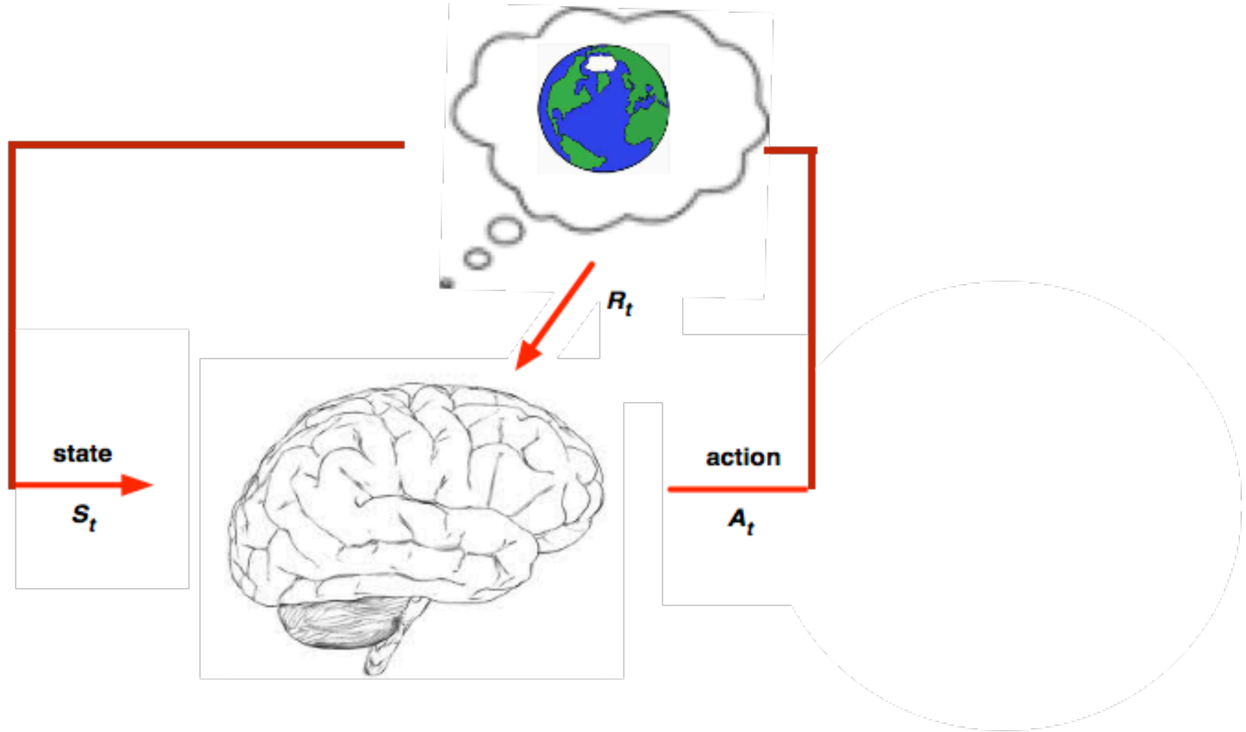
Dyna-Q with an Inaccurate model Cont.

- The changed environment is easier

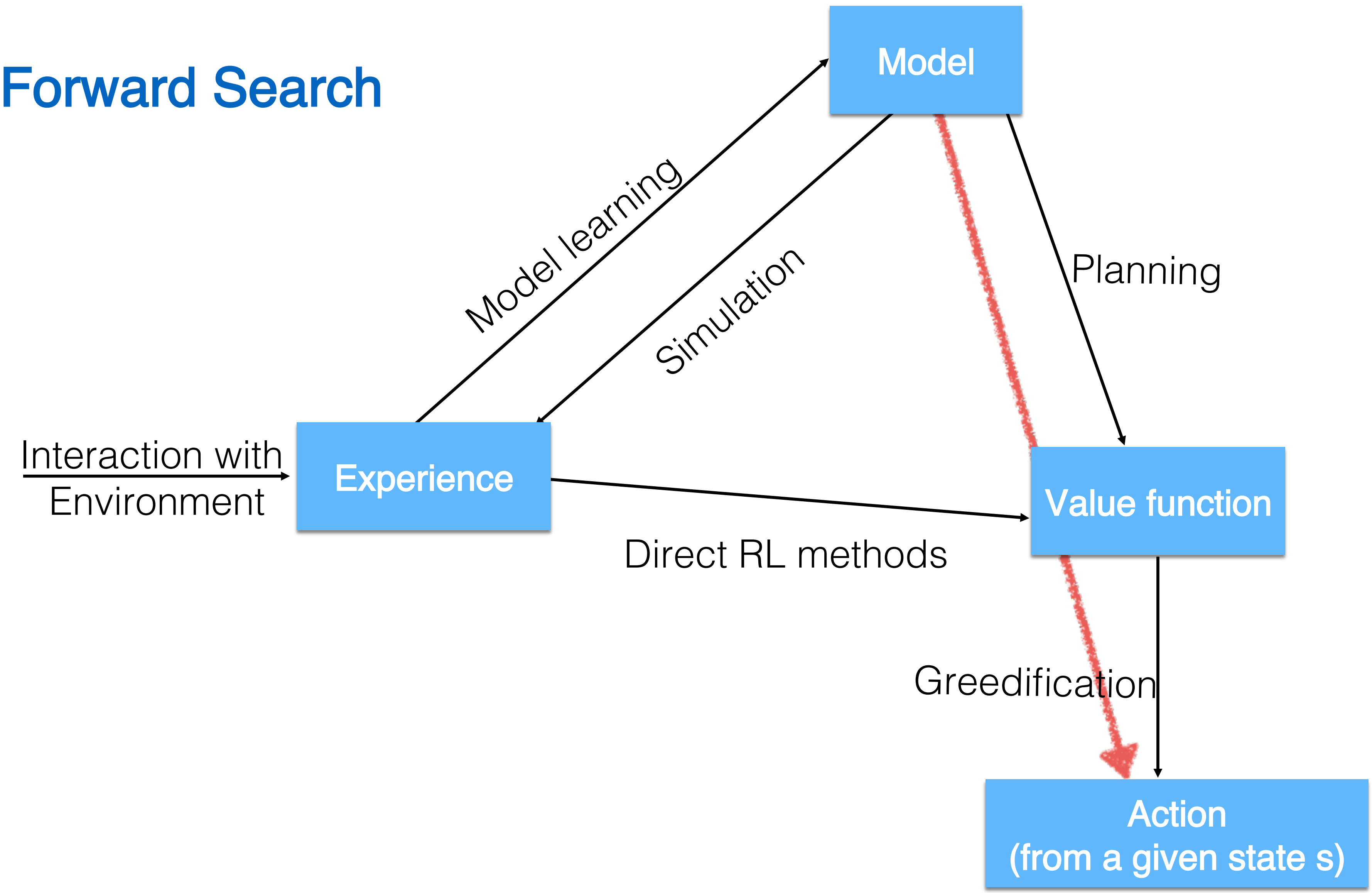


Sampling-based look-ahead search

- If the model is unknown, we will learn the model.
- Learn value functions using both real and simulated experience
- Learn value functions online using model-based lookahead search

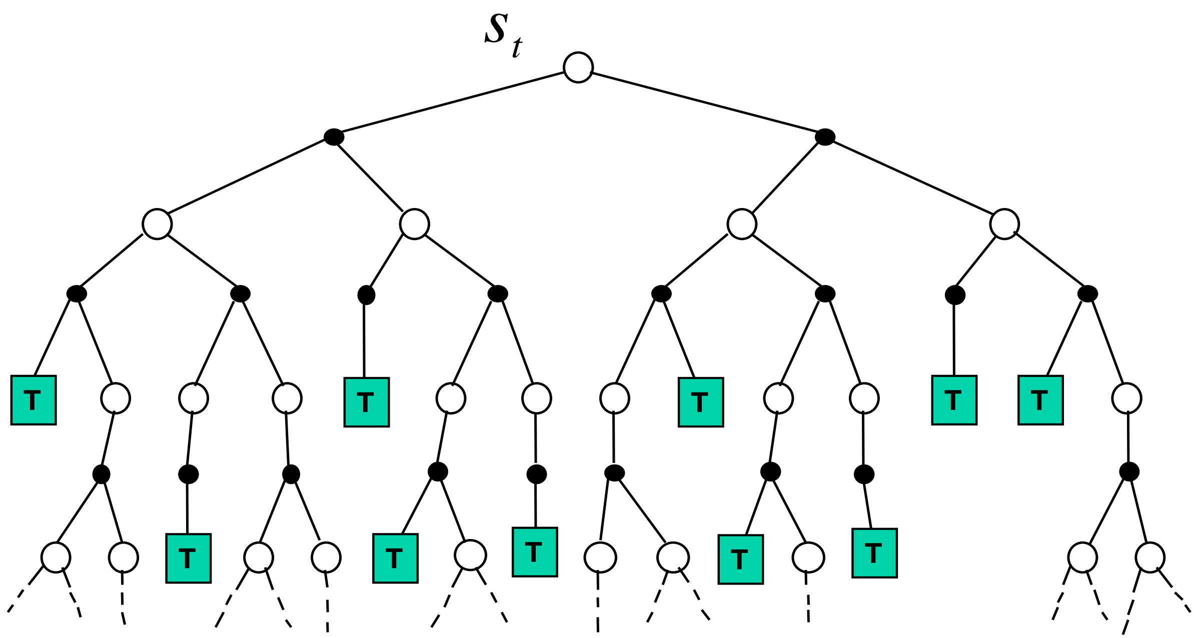


Forward Search



Forward Search

- Prioritizes the state the agent is currently in
- Using a model of the MDP to look ahead (exhaustively)
- Builds a search tree with the current state at the root
- Focus on sub-MDP starting from now, often dramatically easier than solving the whole MDP



Why Forward search?

Why don't we learn a value function directly for every state offline, so that we do not waste time online?

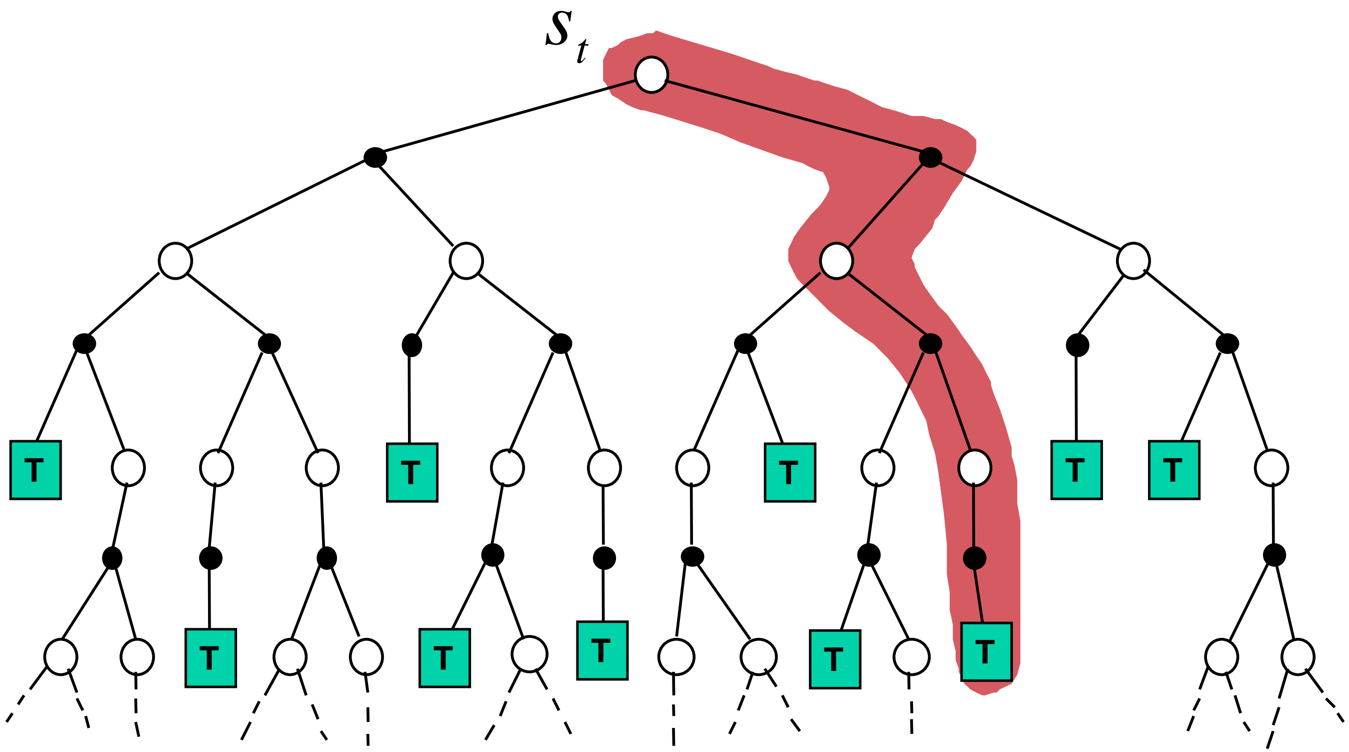
- Because the environment has many many states (consider Go 10^170, Chess 10^48, real world ….)
- Very hard to compute a good value function for each one, most you will never even visit
- Thus, it makes sense, condition on the current state you are in, to try to estimate the value function of the relevant part of the state space online
- Use the the online forward search to pick the best action

Disadvantages:

- Nothing is learnt from episode to episode

Simulation-based Search I

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



Simulation-Based Search II

- Simulate episodes of experience from now with the model

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

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

Simple Monte-Carlo Search

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

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

  - Evaluate action value function of the root 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 (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 for all states and actions in the tree

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

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

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

Monte-Carlo Tree Search (Simulation)

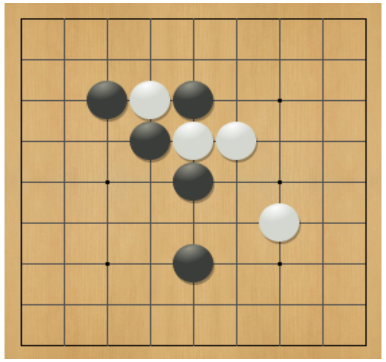
- In MCTS, the simulation policy π improves
- Each simulation consists of two phases (in-tree, out-of-tree)
  - Tree policy (improves): pick actions to maximize $Q(s, a)$
  - Default policy (fixed): pick actions randomly
- Repeat (each simulation)
  - Evaluate states $Q(s, a)$ by Monte-Carlo evaluation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

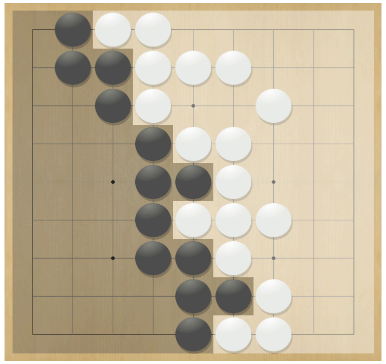
Case Study: the Game of Go

- The ancient oriental game of Go is 2500 years old
- Considered to be the hardest classic board game
- 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):

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

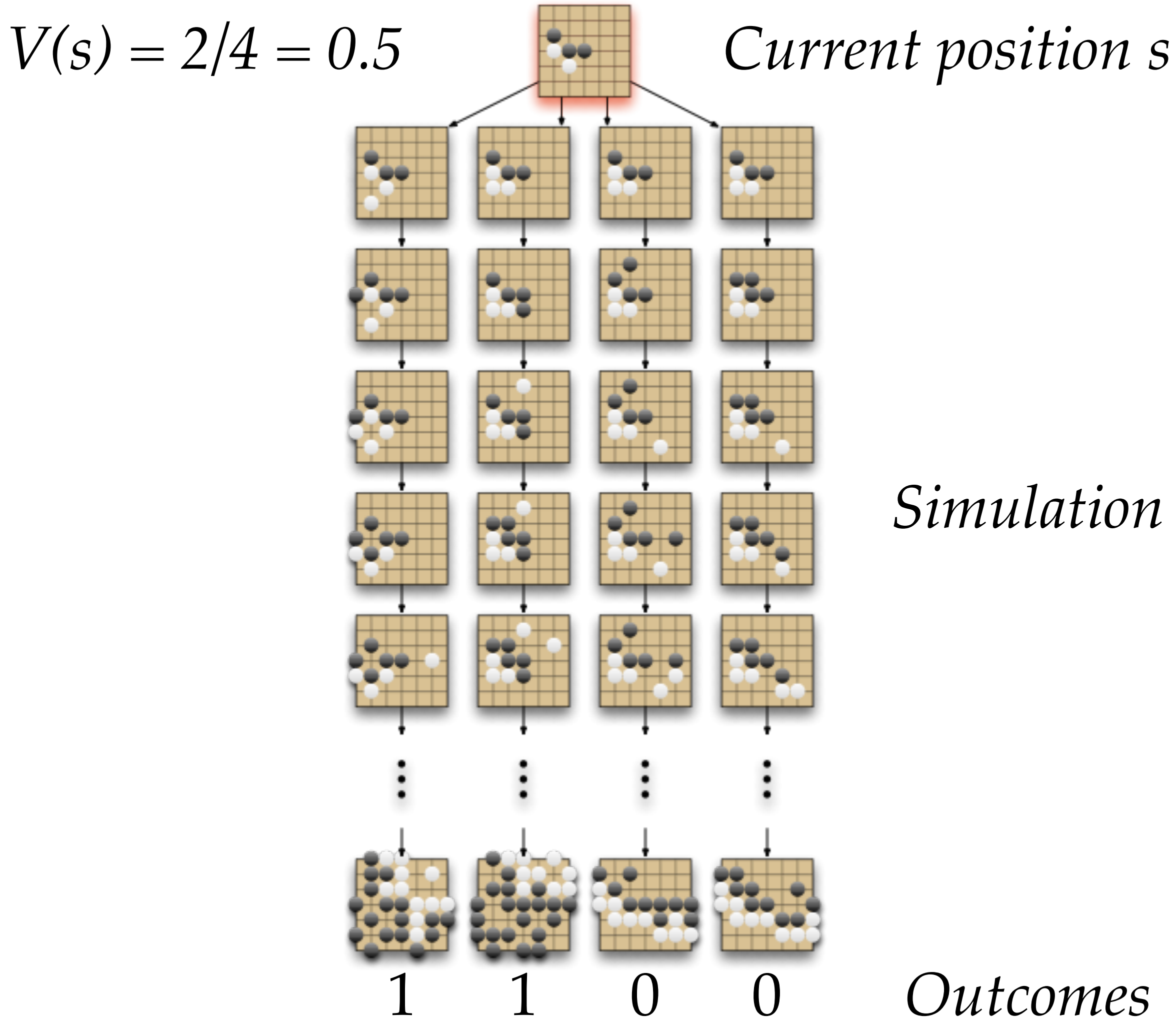
$$R_{\mathcal{T}} = \begin{cases} 1, & \text{if Black wins.} \\ 0, & \text{if White wins.} \end{cases}$$

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

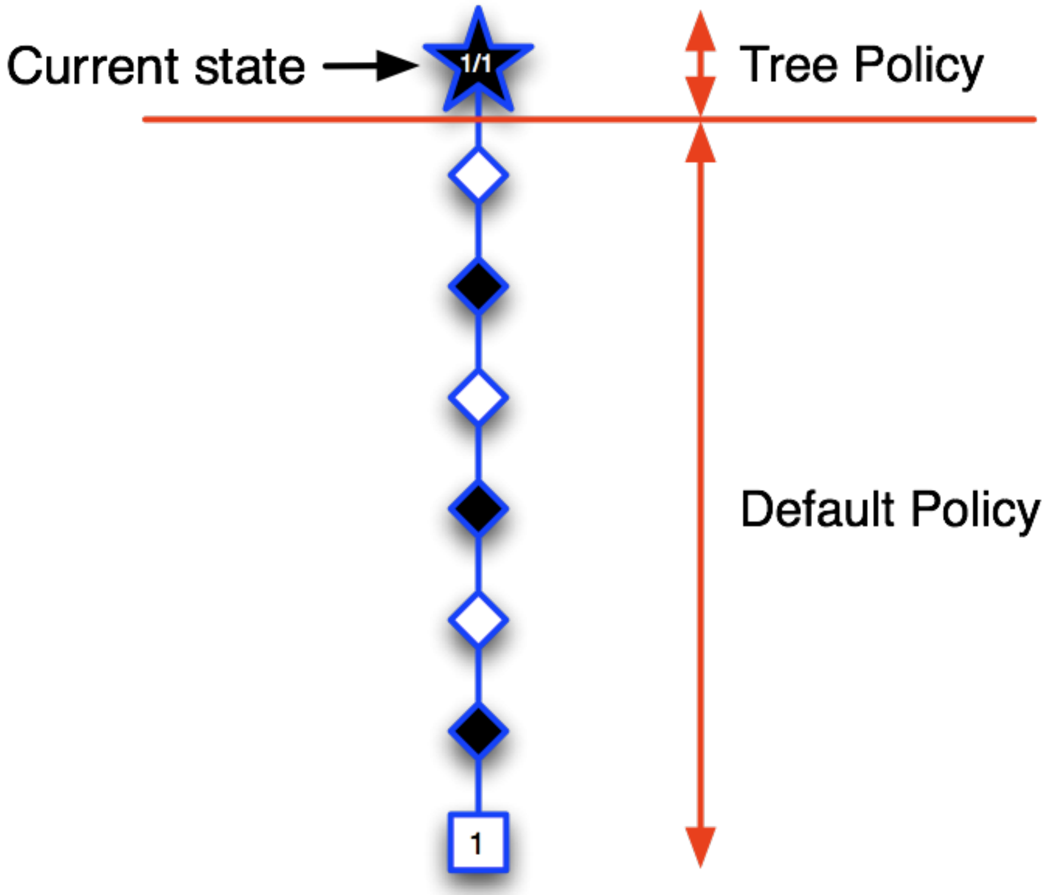
$$v_\pi(s) = \mathbb{E}_\pi[R_{\mathcal{T}} | S = s] = \mathbb{P}[Black \;\; wins | S = s]$$

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

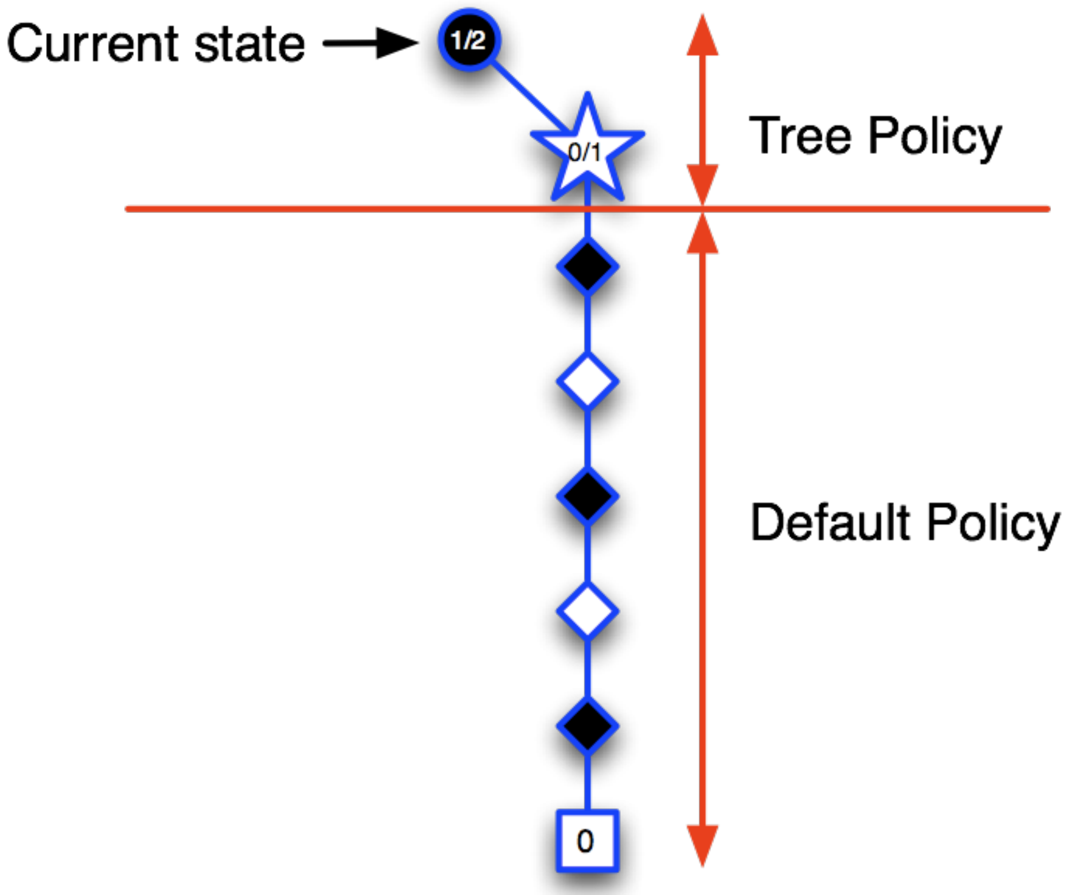
Monte-Carlo Evaluation in Go



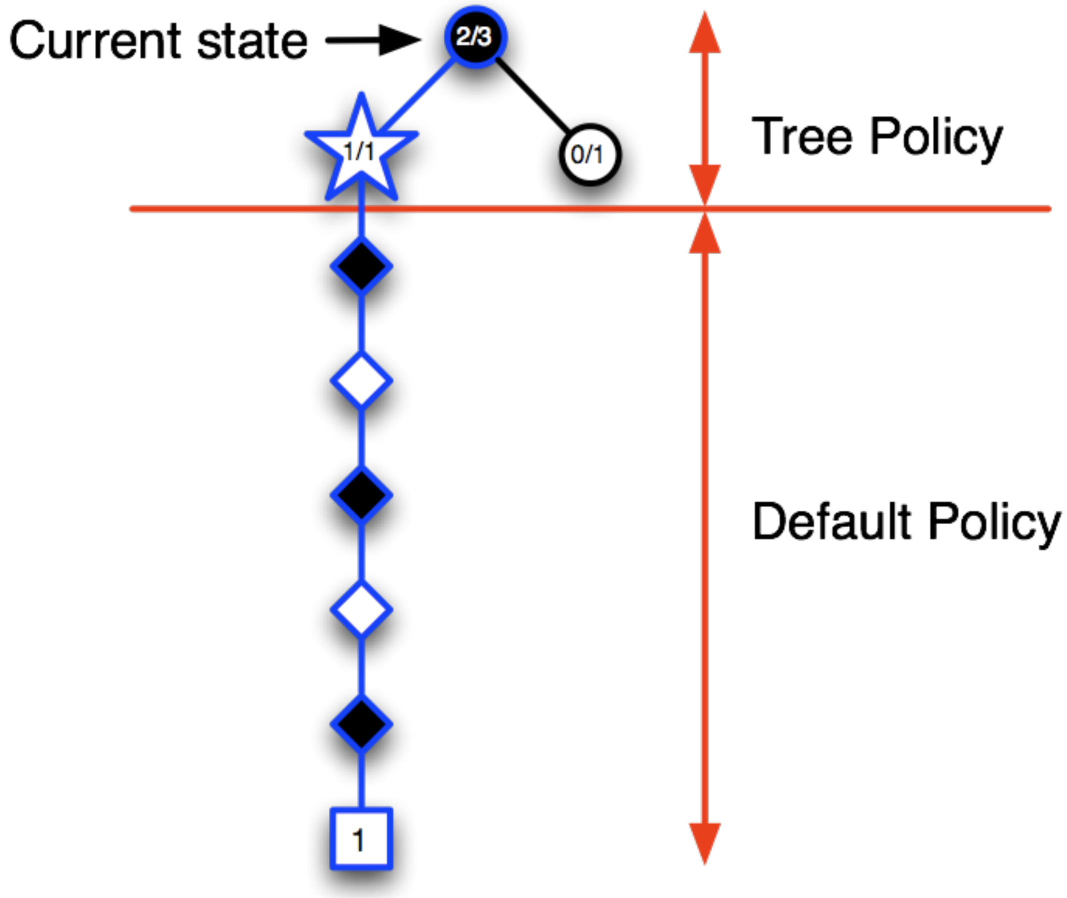
Applying Monte-Carlo Tree Search



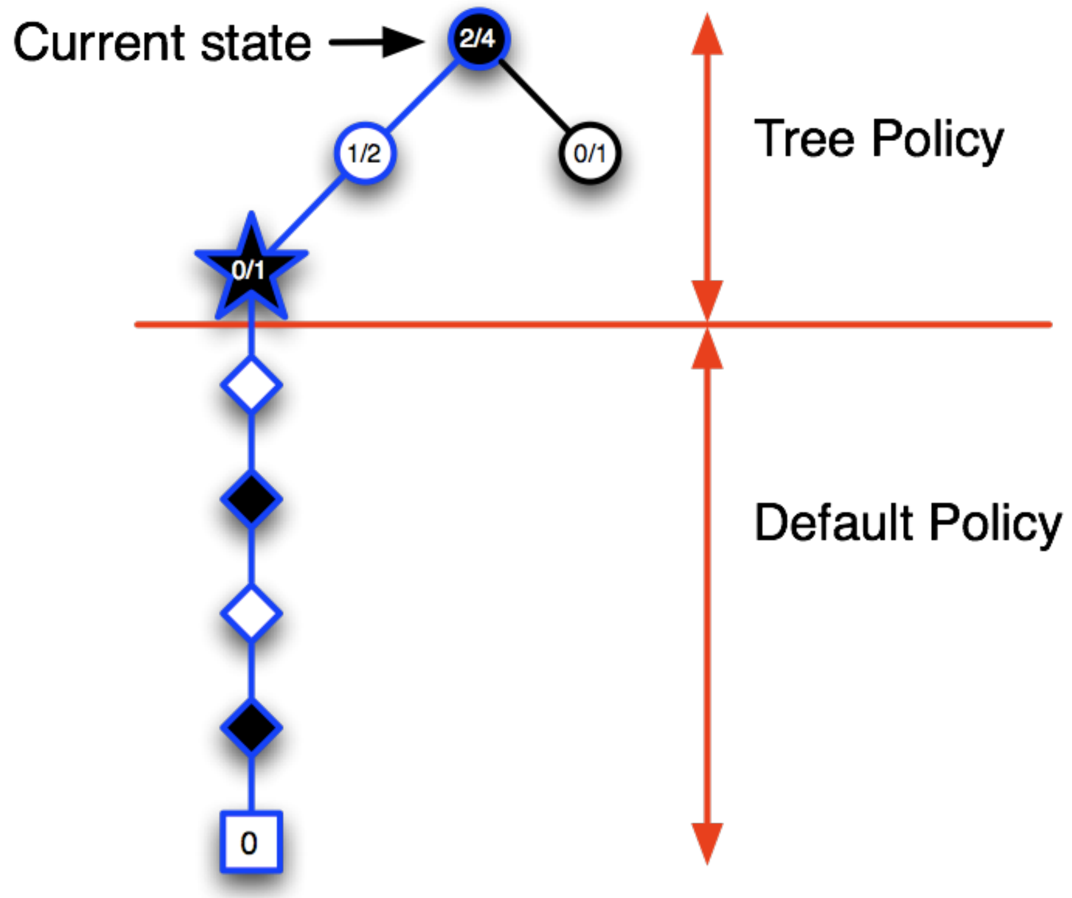
Applying Monte-Carlo Tree Search



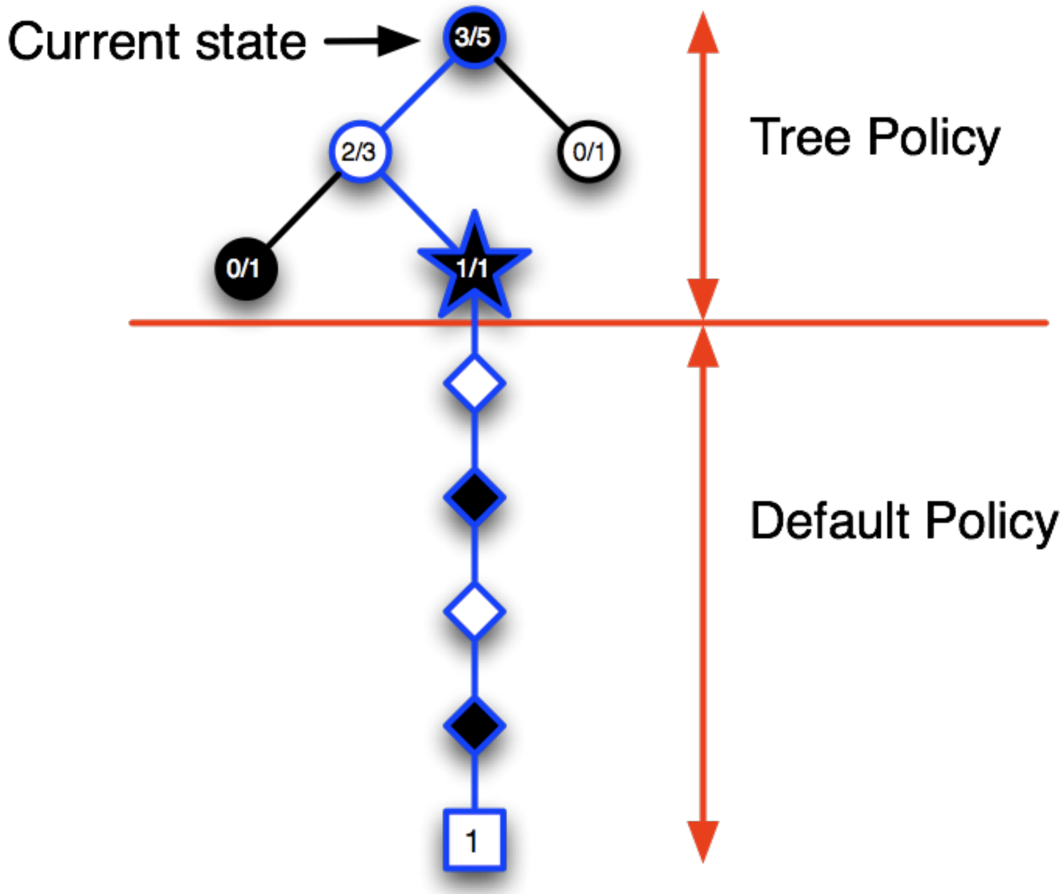
Applying Monte-Carlo Tree Search



Applying Monte-Carlo Tree Search



Applying Monte-Carlo Tree Search



Advantages of MC Tree Search

- Highly selective: best-first search
- Evaluate states dynamically (unlike e.g. DP)
- Uses sampling to break curse of dimensionality
- Computationally efficient, anytime, parallelizable

Combining offline and online value function estimation

- Use policy networks to have priors on Q(s,a):

$$a_t = \operatorname{argmax}_a (Q(s_t, a) + u(s_t, a))$$

$$u(s,a) \propto \frac{P(s,a)}{1 + N(s,a)} \qquad P(s,a) = \pi_\sigma(a|s)$$

- Use fast and light policy networks for rollouts (instead of random policy)