PLANNING AND LEARNING IN REINFORCEMENT LEARNING

SAN DIEGO MACHINE LEARNING

JULY 10, 2021

HOW TO PARTICIPATE

- One discussion leader, and everyone welcome to participate
- Majority of material comes from Reinforcement Learning by Sutton and Barto
- Options to approach the content:
    - Treat this as a standalone webinar
    - Read the book first, and come with questions and discussion items
    - Use this meetup as a primer and read the chapters afterward
- Ask questions
- Give feedback. Too fast or too slow? Want to see more of something or less of something else?
- Have fun!

AGENDA

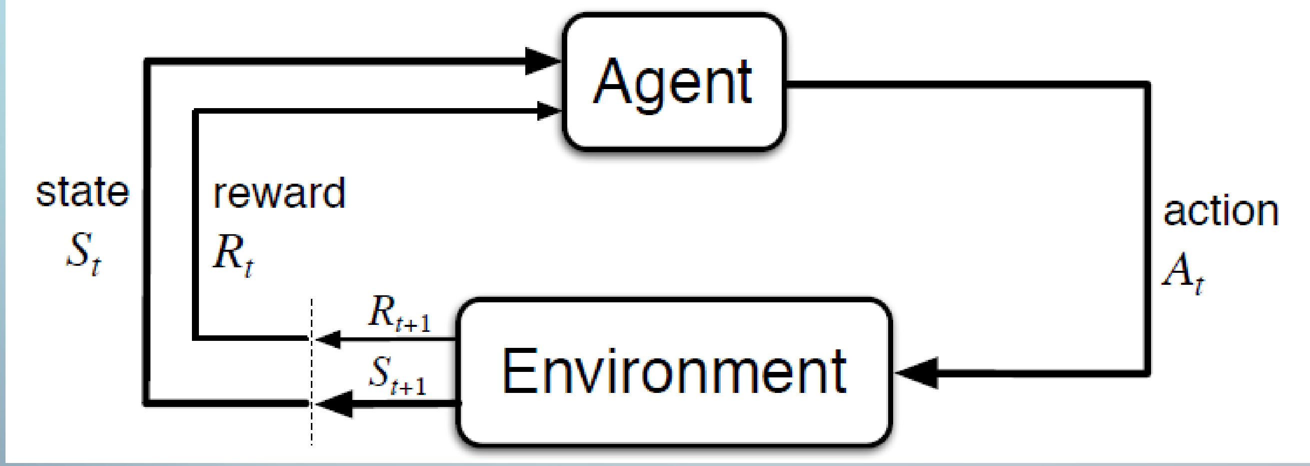
- Recap what reinforcement learning (RL) is
  - Elements and formulation as Markov decision processes (MDP)
  - Terminology and notation used in RL
  - The Bellman equations
  - Generalized policy iteration
- Planning and learning
  - Planning versus learning
  - Dyna algorithm
  - Heuristic and Monte Carlo tree search
  - Wrap up of tabular RL

REINFORCEMENT LEARNING

- Reinforcement learning (RL) is about an *agent* learning from interacting with its uncertain *environment*
  - The agent interacts by choosing from a set of allowed *actions*
  - It gets feedback from a numeric *reward* signal
  - Goal is to maximize the *return*, which is the total rewards received
- Reinforcement learning is about exploring the environment and recording useful information for the future
- RL is sequential decision making; time is intrinsic

MARKOV DECISION PROCESSES

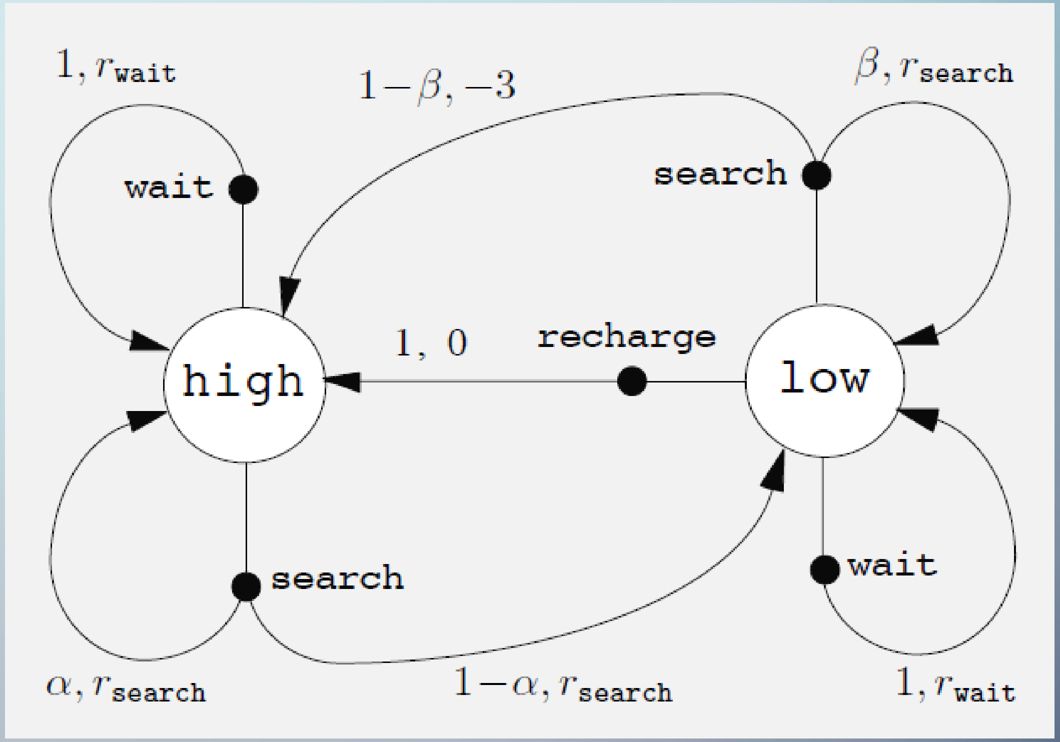
- Elements of the fully observable Markov Decision Process (MDP):
  - State - at each time step t, the environment is in some state S_t
  - Action - at each time step t, the agent chooses an action A_t
  - Reward - after taking the action, the agent is given a reward signal R_{t+1} and subsequently finds itself in a new state S_{t+1}



- In a *Markov* Decision Process, the transition at any given time t only depends on the state S_t and action chosen A_t

MDPS AS A GRAPH

- Sometimes it is easier to visualize a MDP as a directed graph
  - The states are nodes (big white circles)
  - The actions are edges leading from nodes (here with small black circles)
  - The rewards are values along directed edges that take you to a new state
- Here is the recycling robot from the book:



REINFORCEMENT LEARNING NOTATION

Letter	Used for
s	**<u>S</u>**tate
a	**<u>A</u>**ction
r	**<u>R</u>**eward
γ	Discount rate
G	Return – sum of all future rewards
p	Transition **<u>p</u>**robability
v	**<u>V</u>**alue function for states
q	Value function for state-action pairs
π	Policy (**<u>π</u>**ολιτική)
*	Optimal choices, e.g. π_*

BELLMAN EQUATION

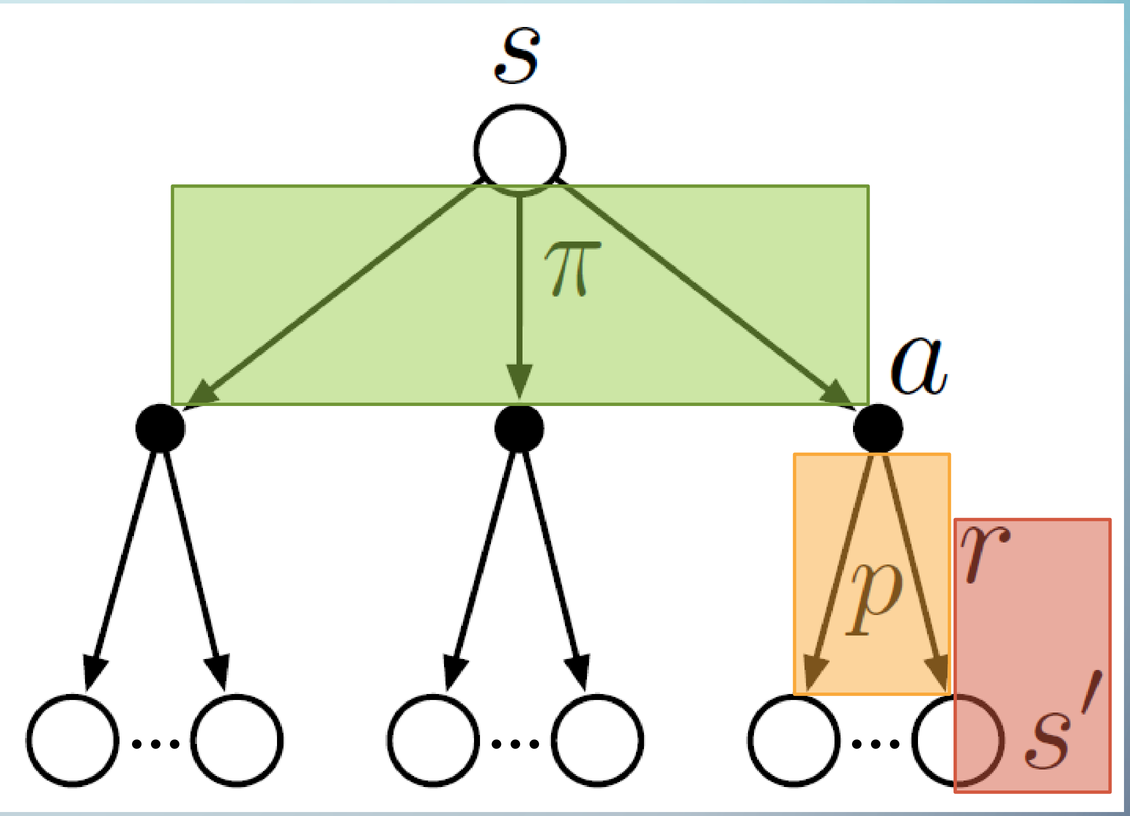
- The value function for state s under policy π is a sum of the rewards received and the value functions for each future state s' times the probability of winding up there
- Formally:

$$v_\pi(s) = \sum_a \underbrace{\pi(a, s)}_{\substack{\textbf{Probability you}\\ \textbf{take action } a}} \sum_{s', r} \underbrace{p(s', r | s, a)}_{\substack{\textbf{Probability you}\\ \textbf{get reward } r\\ \textbf{and end in state } s'}} [\underbrace{r + \gamma v_\pi(s')}_{\substack{\textbf{Reward plus}\\ \textbf{discounted value}\\ \textbf{of new state } s'}}]$$

BELLMAN EQUATION VISUALIZED

This is a *backup diagram* for $v_\pi(s)$. To compute it:

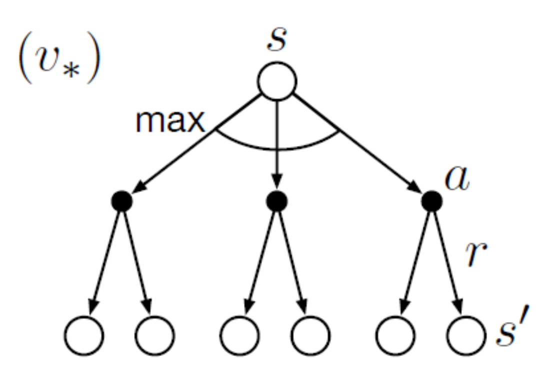
- We need to sum over each branch of π(), based on the probability of each action *a*
- And sum over of each branch of *p*(), based on probability we wind up in state *s'*
- The quantity we sum is the reward and the discounted value of possible state *s'*



$$v_\pi(s) = \sum_a \pi(a,s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]$$

BELLMAN OPTIMALITY EQUATIONS – V()

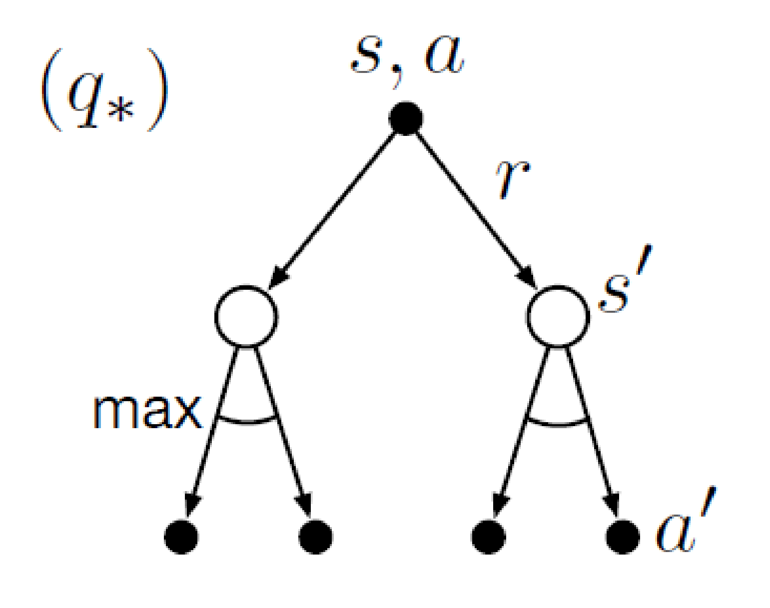
- The *Bellman optimality equation* says the optimal value for a state must be the same as the return from the best action
- We can rewrite it recursively



$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] && \text{(by (3.9))} \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] && (3.18) \\
&= \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big]. && (3.19)
\end{aligned}
$$

BELLMAN OPTIMALITY EQUATIONS – Q()

- The *Bellman optimality equation* for state-action pairs is very similar.
- The optimal value for a state-action pair must be the same as the return from the reward and best next action
- It also can be written recursively



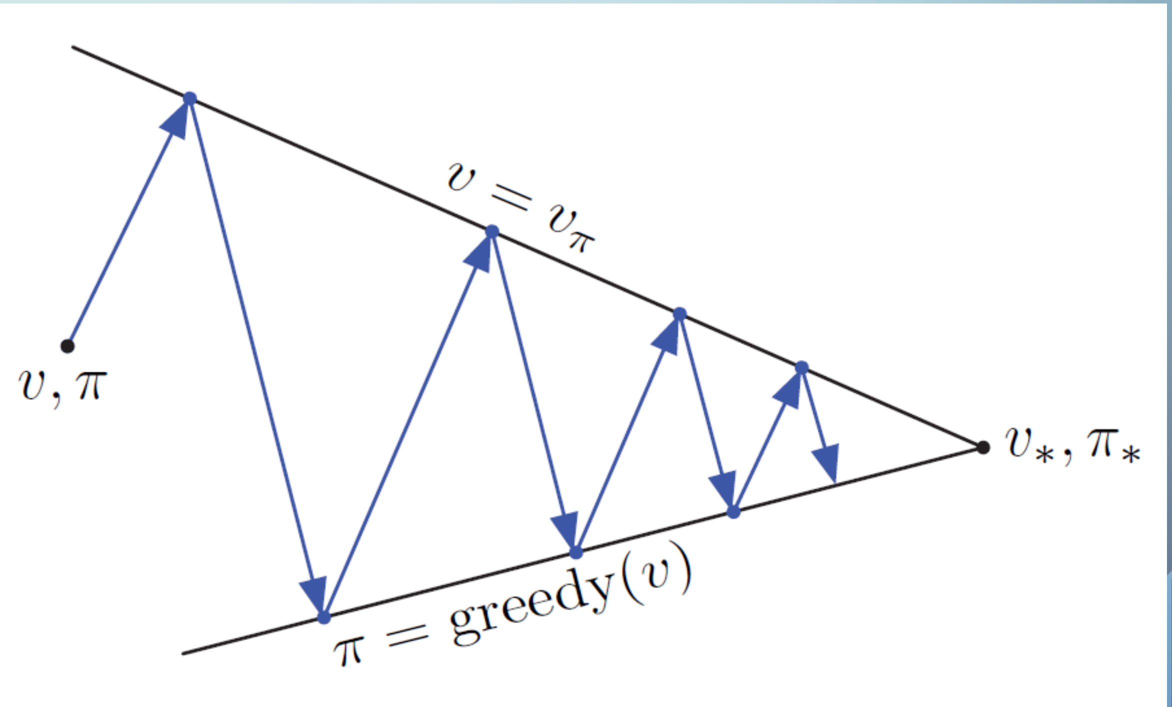
$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\Big|\, S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r \,|\, s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big]. \end{aligned} \tag{3.20}$$

POLICY ITERATION

- The book shows a sequence like this:

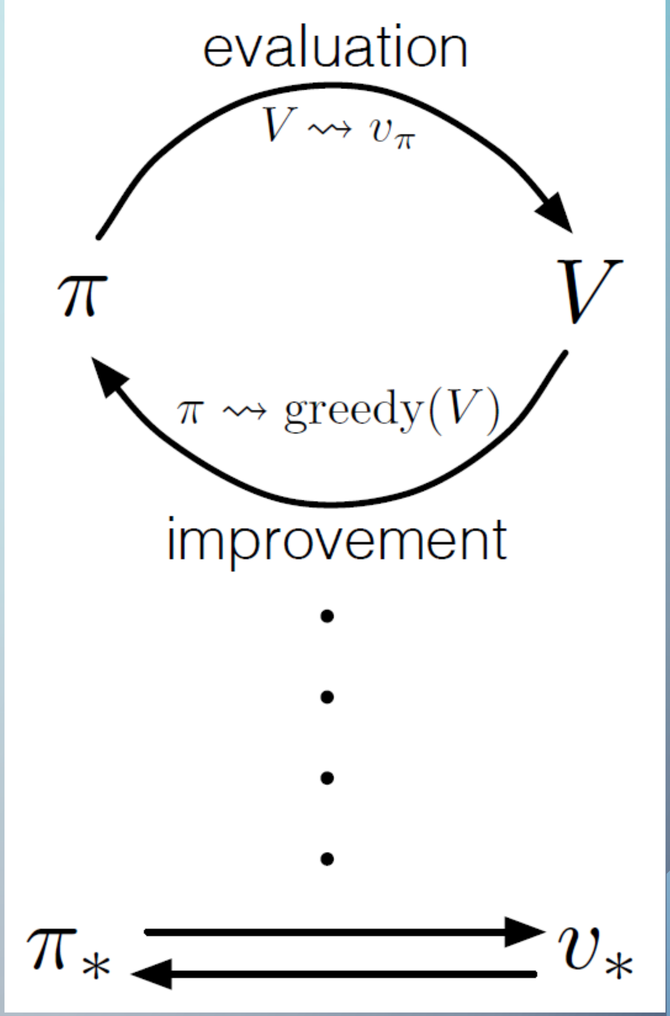
$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_1 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

- The arrows with E's are full cycles of iterative policy evaluation
- And the arrows with I's are policy improvement



GENERALIZED POLICY ITERATION

- The term *generalized policy iteration* (GPI) refers to the general idea of letting policy evaluation and policy improvement processes interact
  - Doesn't matter how fully each evaluation or improvement step runs, or if they exactly alternate



REINFORCEMENT LEARNING CONTROL

- With this foundation, there's a lot we can tackle
  - Algorithms for learning
  - Dealing with memory and compute limitations
  - Getting models to converge quickly
- We also still have many challenges
  - Reward design – effectively communicating the real goal
  - Sparse rewards
  - Credit assignment – which actions in trajectory contributed
  - Exploration vs. exploitation

MONTE CARLO EXPLORING STARTS

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
- $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
- $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
- $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
- Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
- Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
- $G \leftarrow 0$
- Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{t+1}$
  - Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
    - Append G to $Returns(S_t, A_t)$
    - $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
    - $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

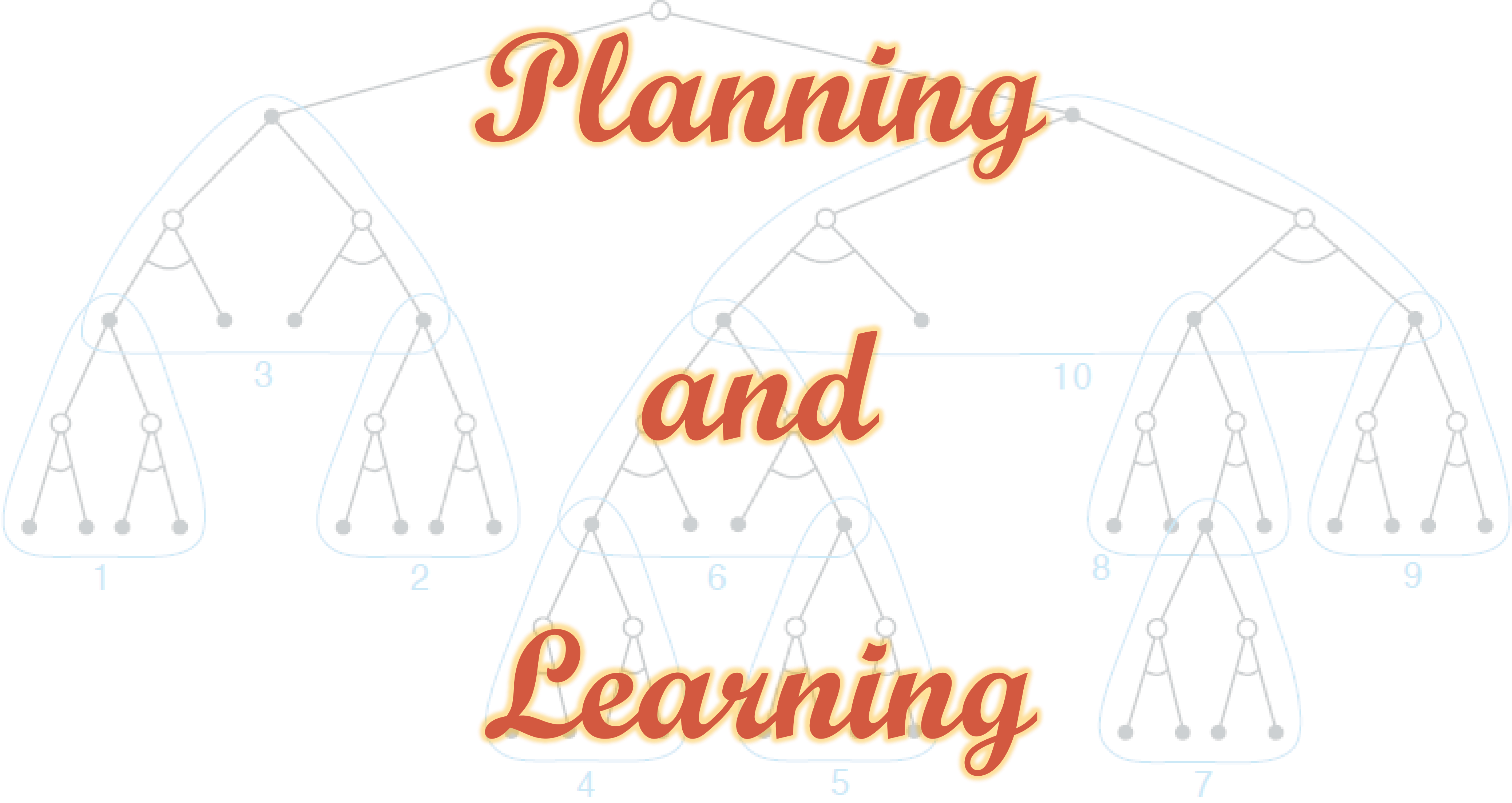
Q-LEARNING

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Planning and Learning



MODELS

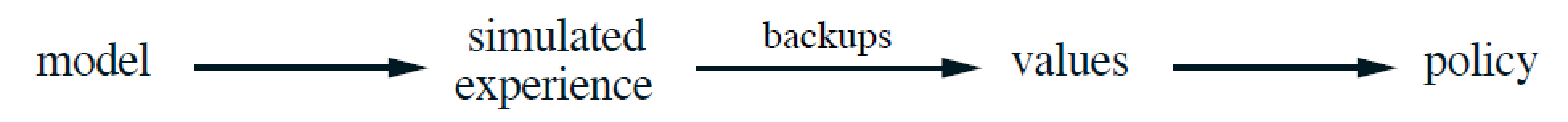
- In RL, the term *model* is used specifically to refer to the component that given a state and action, predicts the reward and next state
- For stochastic environments, the model can output a probability distribution (a *distribution model*), or it can simply output samples (a *sample model*)
- Distribution models are stronger, but grow complex fast, and are harder and more error-prone to build

MODELS AND PLANNING

- Techniques that don't require a model, such as TD and Monte Carlo, are called *model-free*, and they are said to do *learning*
- Algorithms that require a model, such as dynamic programming, are called *model-based*, and they are said to do *planning*
  - *Planning* – a process that inputs a model and improves a policy
- If we have a model, then given a starting state and action, we can produce simulated experience
  - A sample model might produce an episode, and a distributional might produce all possible transitions along with their probabilities

PLANNING

- The book covers *state-space planning*, where value functions are computed, and we search the state space for an optimal policy
- The planning process has the following steps:



- Going back to dynamic programming, it follows this structure, using a distributional model to evaluate all possible transitions, and backing up values one step ahead

Q-PLANNING

- A simple new example of planning is we can turn Q-learning into Q-planning by replacing the actual experience with the environment with simulated experience with a model

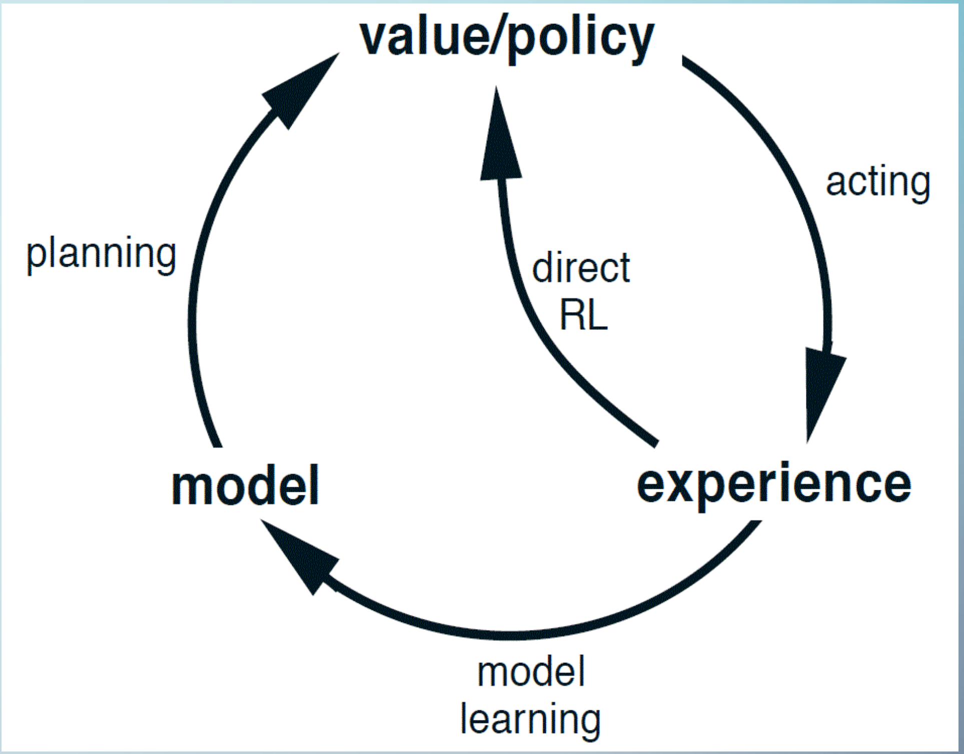
Random-sample one-step tabular Q-planning

Loop forever:

1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

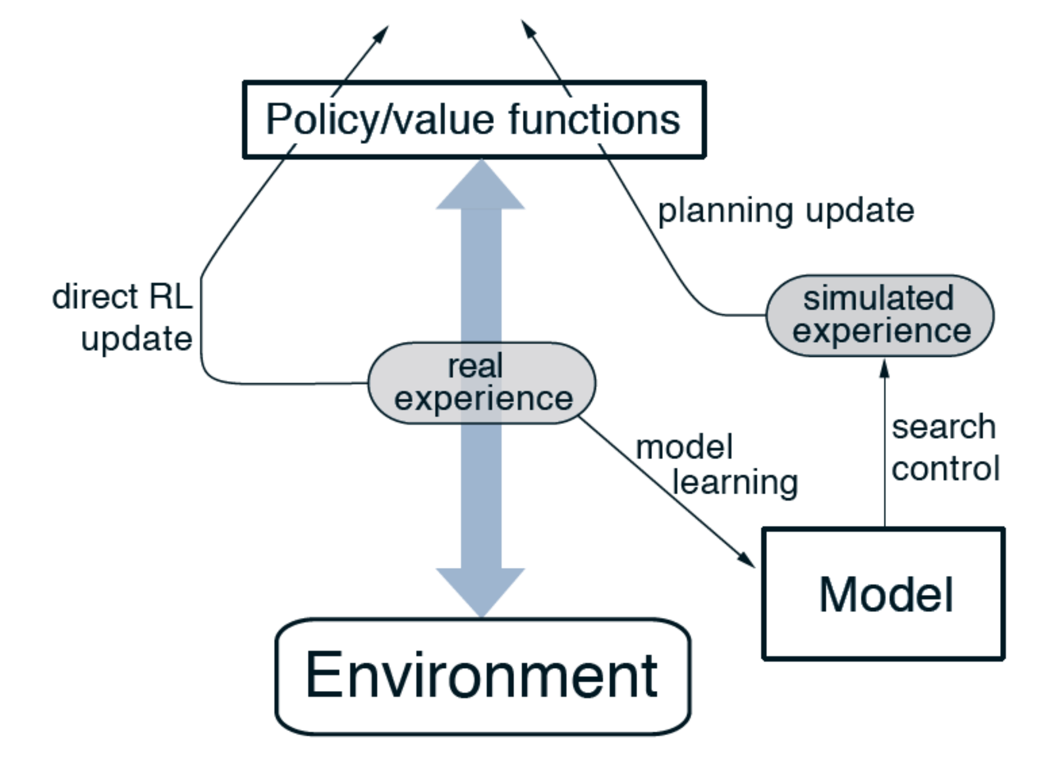
ONLINE PLANNING

- If we do planning online, experience from interactions can:
  - Improve the model – called *model learning*
  - Perform learning (improve the value function and policy) – *direct RL*
- And planning itself can improve the value function and policy
  - Sometimes called *indirect RL*
- Planning tends to be more sample efficient



DYNA-Q

- Dyna-Q interacts and then does both direct RL, and model learning & planning
  - Direct RL is Q-learning
  - Planning is Q-planning
  - Model learning is simplistic. Assumes deterministic, and each transition from a state and action is assumed to always be the next state and reward (any previous info recorded is overwritten)
  - Search control is how state action pairs are chosen for planning



DYNA-Q ALGORITHM

- Here is the pseudocode for basic tabular Dyna-Q:

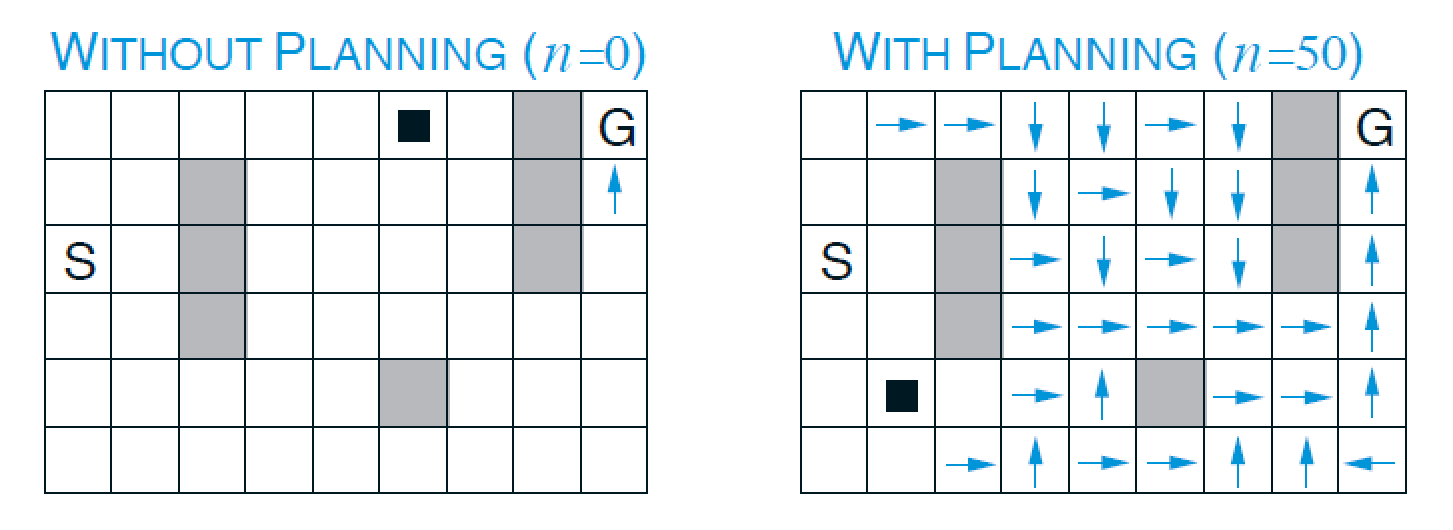
Tabular Dyna-Q

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) $A \leftarrow \varepsilon\text{-greedy}(S,Q)$
  (c) Take 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) Loop repeat n times:
    $S \leftarrow$ random previously observed state
    $A \leftarrow$ random action previously taken in S
    $R, S' \leftarrow Model(S,A)$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

Here, search control is random

DYNA-Q PERFORMANCE

- In the Dyna-Q algorithm shown, we perform *n* planning steps for each interaction with the environment
  - When running online, it's realistic that we have time to repeat planning
- The more planning we do, the fewer interactions we need



IMPROVING PLANNING

- Dyna-Q works well with missing values, but doesn't explore well
- Dyna-Q+ keeps track of how long since each state-action pair was visited, and a bonus reward which increases with time is added to simulation
- Prioritized sweeping tracks which values have changed the most and prioritizes states the lead to those changed states
- For large state spaces, *trajectory sampling* picks states and actions by following the current policy
  - Following the on-policy distribution accelerates learning early by focusing on useful states. Might not explore enough in the long term.

PLANNING AT DECISION TIME

- We've been talking about general planning to improve the overall value function an policy, called *background planning*
- Planning at decision time can look a little different
- At decision time, rather than trying to improve our policy, we are trying select the best action
- We can do planning in a similar fashion to everything previously discussed, but now we root all activity in the current state

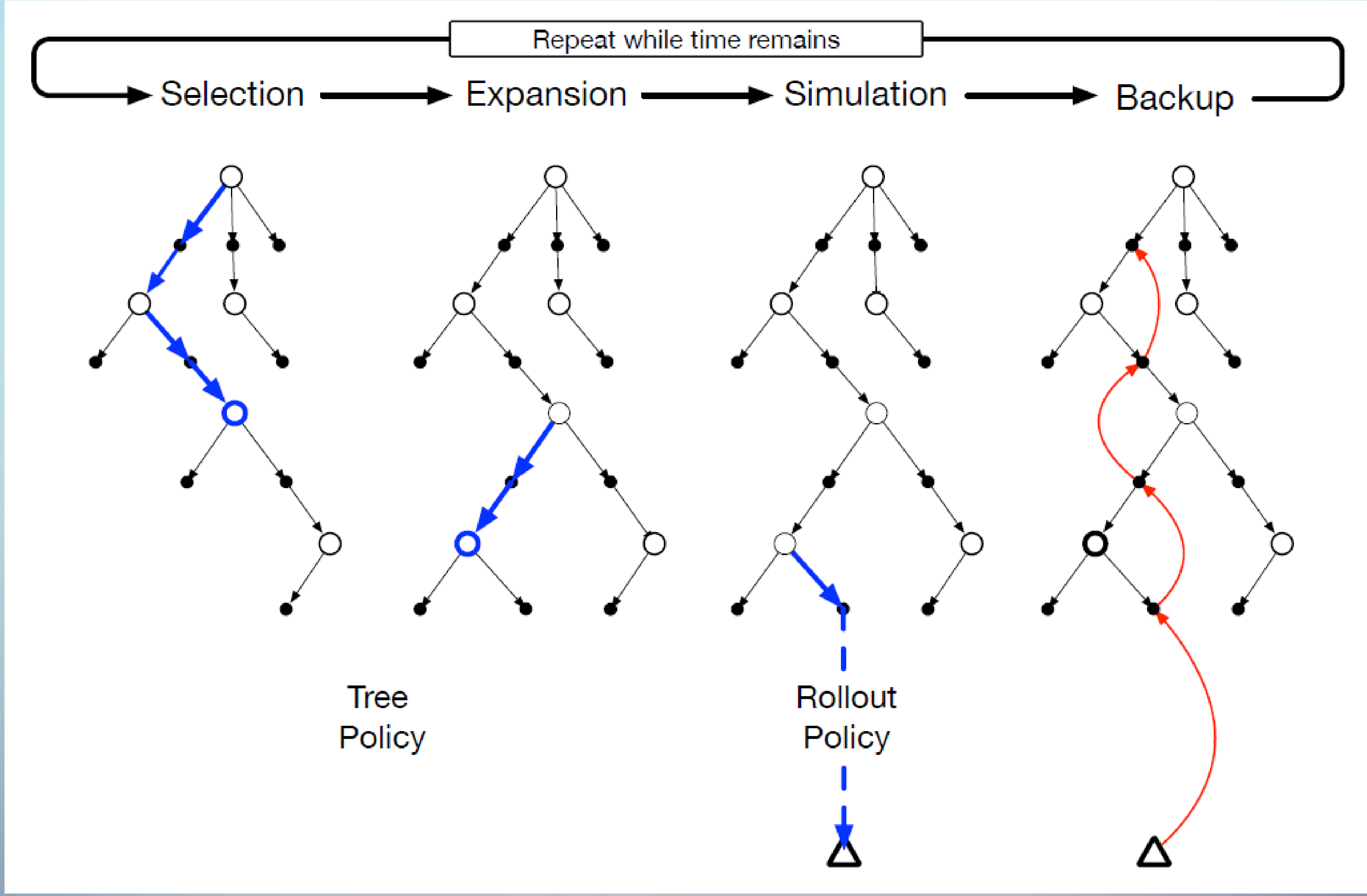
HEURISTIC SEARCH

- In *heuristic search*, for each state encountered, a large tree of possible continuations is considered, and values of leaves are backed up
- Somewhat surprisingly, backed up values historically were discarded instead of updating the value function (which may have been hand-crafted, not learned)
- *Rollout algorithms* simulate trajectories starting from the current state, following a rollout policy
  - Similar to Monte Carlo methods, but starting at current state
  - Not run to find optimal policy, but will improve rollout policy as more trajectories sampled

MONTE CARLO TREE SEARCH

- Monte Carlo Tree Search (MCTS) takes the rollout algorithm and adds a number of optimizations
- As value estimates are forming, MCTS directs simulations toward more promising trajectories, using four steps:
  - **Selection**. Starting at the root node, a tree policy tree traverses the tree to select a leaf node.
  - **Expansion**. On some iterations, the tree is expanded from the selected leaf node by adding unexplored actions.
  - **Simulation**. From the selected node, or from one of its newly-added child nodes (if any), simulation of a complete episode is run with actions selected by the rollout policy
  - **Backup**. The return generated by the simulated episode is backed up to update the action values attached to the edges of the tree

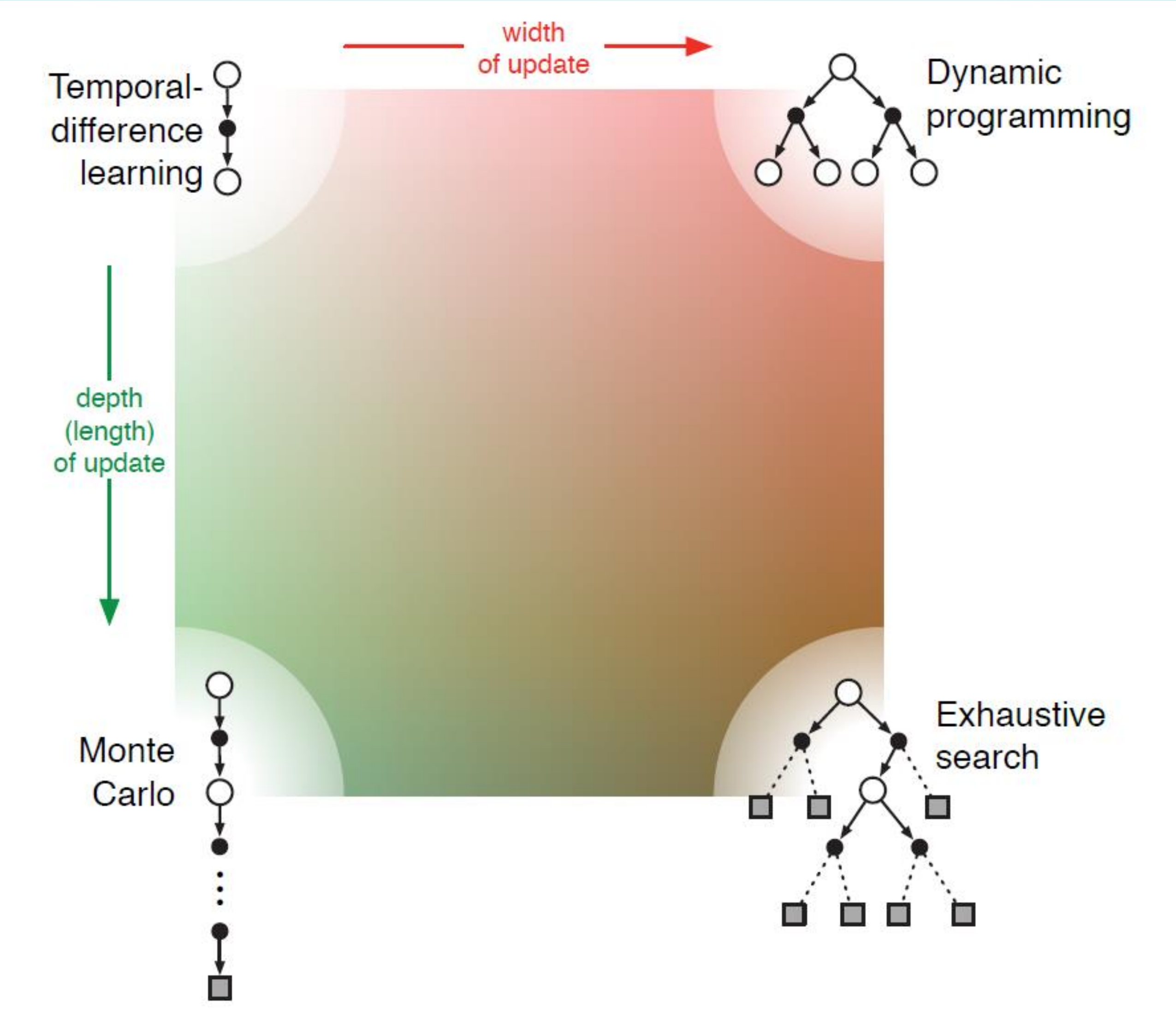
MCTS VISUALIZATION



WRAP UP OF TABULAR RL

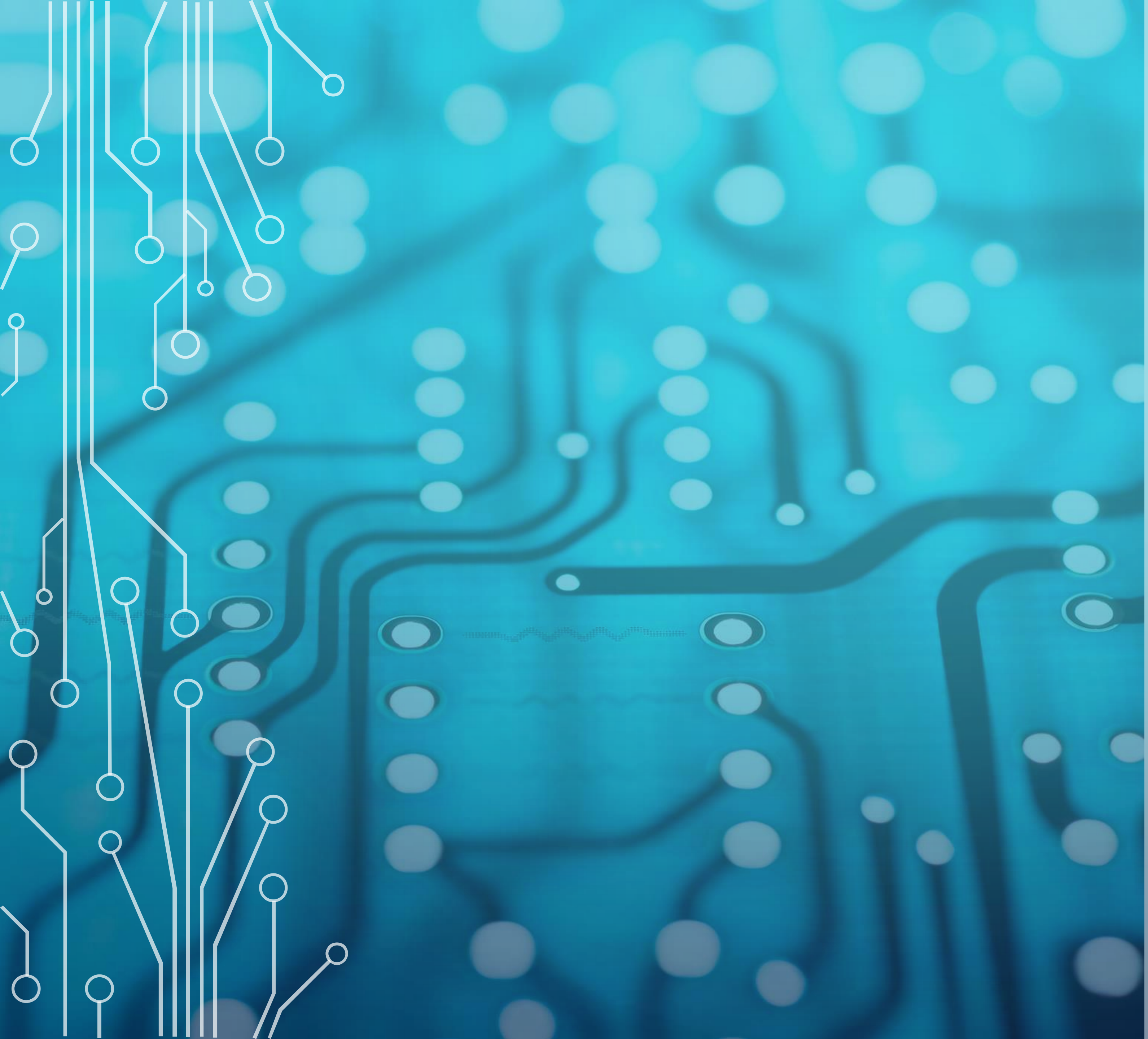
- Rather than separate methods, everything discussed so far can be considered coherent ideas varying on certain dimensions
- All of these:
  - Estimate value functions
  - Back up values along trajectories
  - Follow the general strategy of Generalized Policy Improvement
- Two key dimensions along with the algorithms we've discussed differ are the width and depth of updates.
  - Visualizing the space of these two dimensions unifies many RL methods

DEPTH AND WIDTH OF UPDATES



RECAP

- Review what reinforcement learning (RL) is
  - Elements and formulation as Markov decision processes (MDP)
  - Terminology and notation used in RL
  - The Bellman equations
  - Generalized policy iteration
- Planning and learning
  - Planning versus learning
  - Dyna algorithm
  - Heuristic and Monte Carlo tree search
  - Wrap up of tabular RL



QUESTIONS

&

DISCUSSION

NEXT SESSION

- Next week, Sat. July 17, there will be a guest speaker talking about PySpark, MLlib, and MLFlow
- The following week, on Sat. July 24, the RL topic is still TBD. It will either be about RL using function approximation, or it will be about policy gradient algorithms