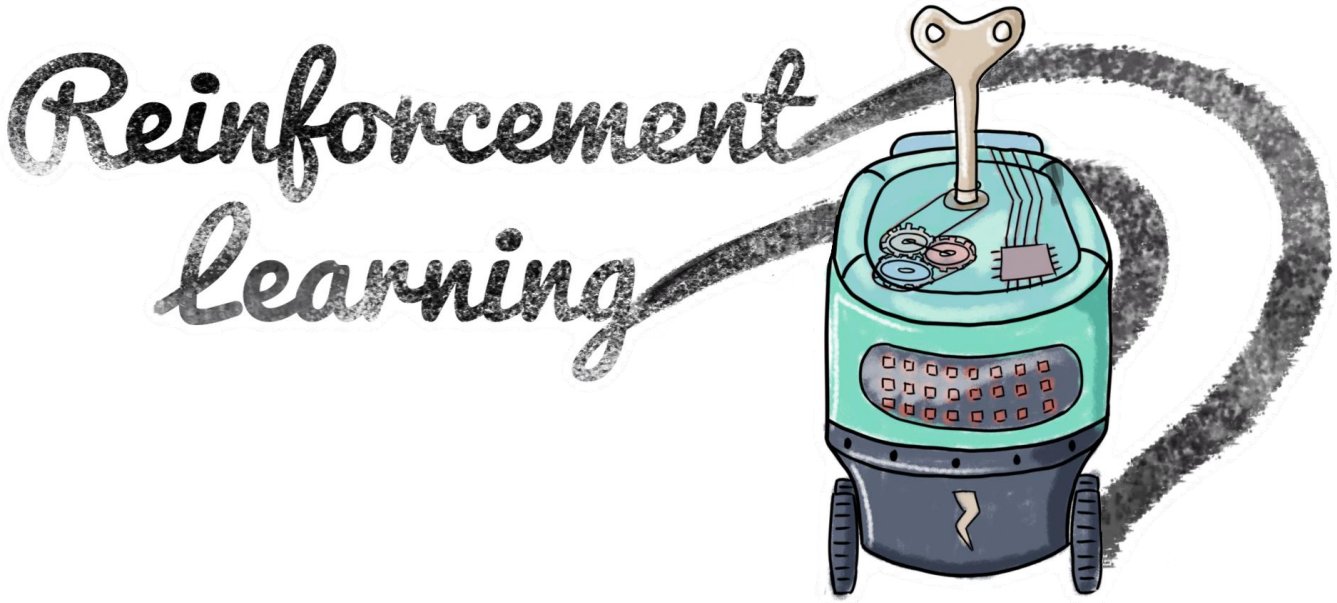


Planning and Learning with Tabular Methods

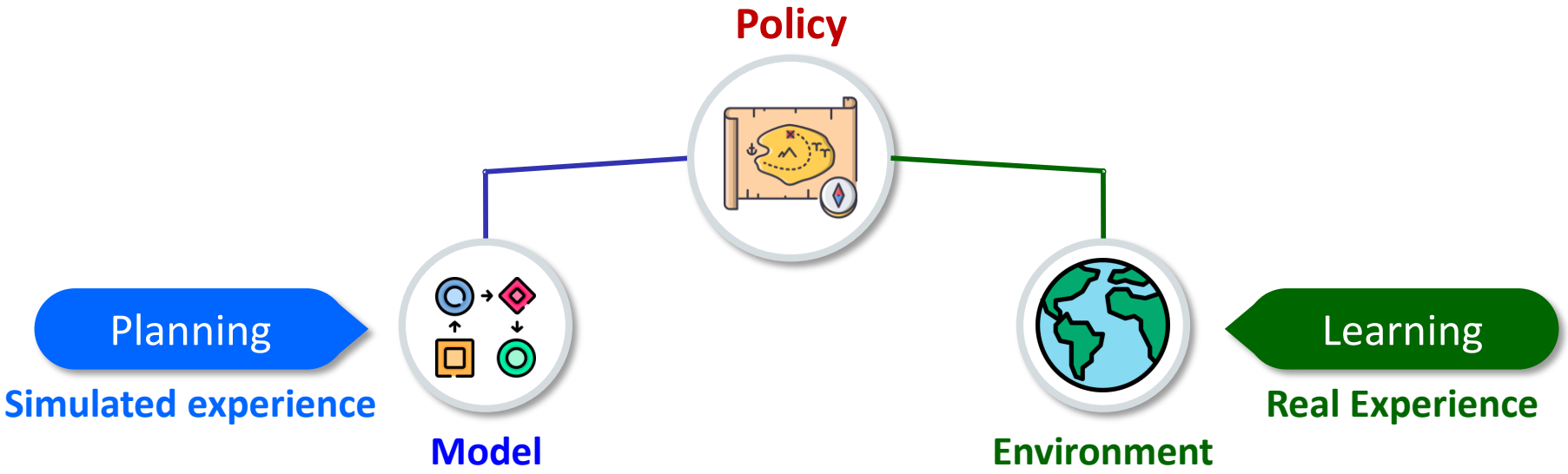
Mohammad **Dehghani**

Mechanical and Industrial Engineering Department

Northeastern University

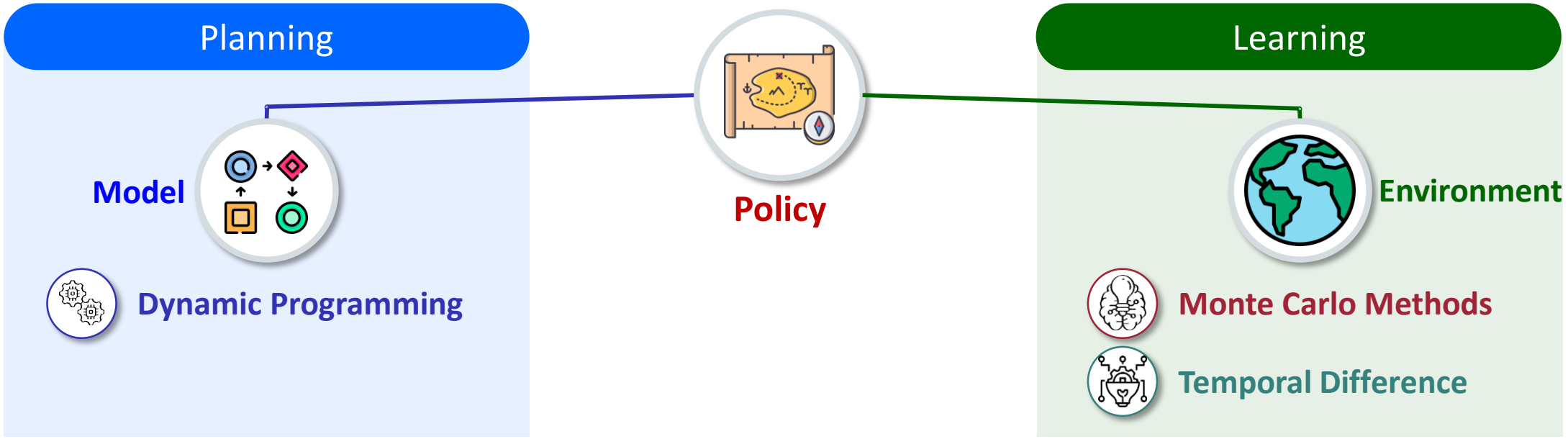
Mar-20

- **Recall:** In reinforcement learning
  - **Goal:** is to find the policy to maximize the accumulated long term reward
  - **Policy:** is a set of rules to select an action in each possible state



Introduction

Unified view of Reinforcement Learning

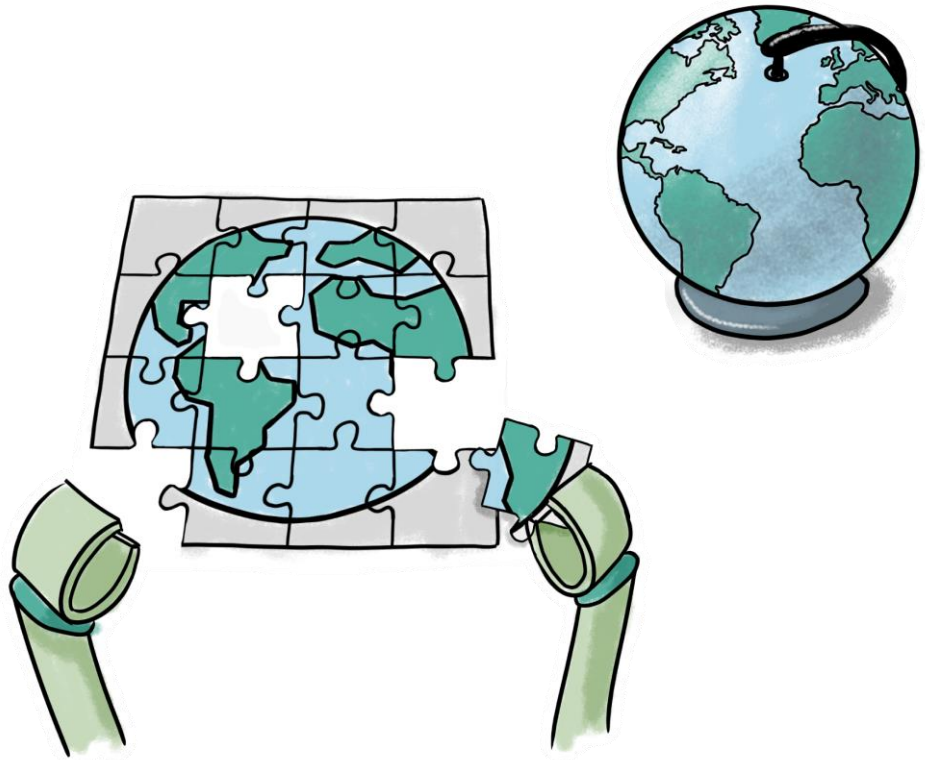


Introduction

Outline

- **Models and Planning**
  - How to learn the model
  - how can we leverage the model of environment to help the learning process
- **Integrated Planning, Acting, and Learning**
  - A unified RL approach: by taking the advantage of both learning and planning
  - Q-Planning
  - Dyna-Q
- **Environment changes**
  - Dyna-Q+
- **Other approaches**
  - Sweeping
  - Heuristics

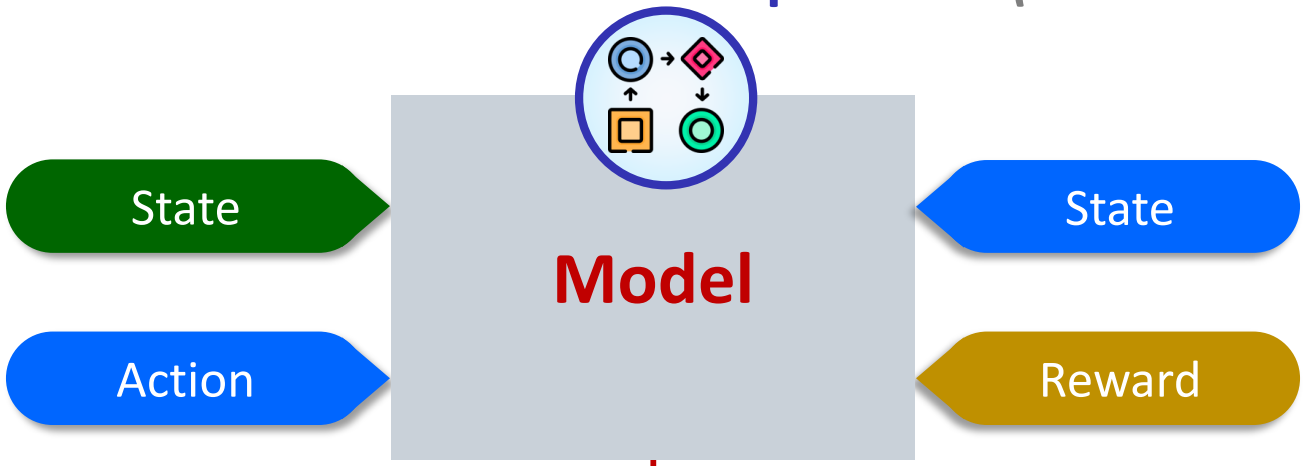
Models and Planning



Models and Planning

What is a Model?

- ***Model*** **is anything the agent can use to predict how the environment will respond to its actions**
- **Models can be used to *mimic* or *simulate* experience *(simulated experience)***



Distribution model

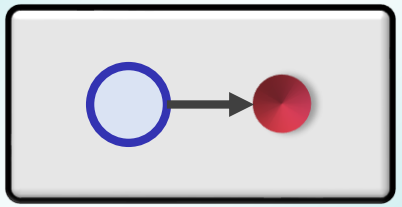
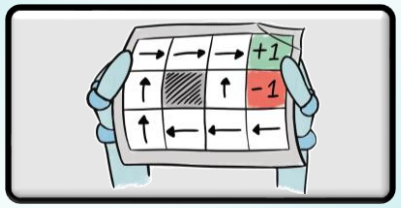
- **Model produces a description of all possibilities and their probabilities**
- **Stronger, but difficult to obtain**

Sample model

- **Model produces just one of the possibilities sampled according to the probabilities**
- **Weaker, but easier to obtain**

Models and Planning

Distribution model and Sample model

Distribution model		Sample model
generates all possible transitions weighted by their probabilities of occurring.	starting state and action 	produces a possible transition
generates all possible episodes and their probabilities	starting state and policy 	produces an entire episode

Models and Planning

Model-based vs Model-free learning

True MDP

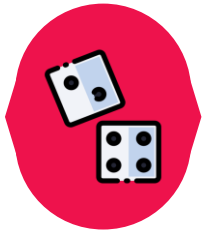
- **Model-based learning**
  - Dynamic programming
  - Heuristic search

Use the *distribution model* to produce all possible sums and their probabilities of occurring

$6^{12} \sim 2.18$ **billion outcomes**

- **Joint Distribution function (exact)**
- **Calculate the *Expected Value* and *Var***
- **Complicated Task**

sum of a dozen dice



Approximate MDP

- **Model-free learning**
  - Monte Carlo Methods
  - Temporal Difference

A *sample model* would produce an individual sum drawn according to this probability distribution

Generate a random number from 1-6, 12 times (n replications)

- **Less computational:** Easy to implement
- **Approximation of true values**

Models and Planning

Pros and Cons: Distribution model and Sample model

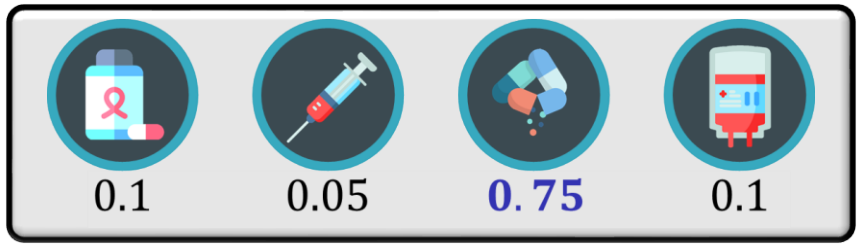
Distribution model

- Distribution models compute the exact expected outcome
  - sumproduct of weights (probs) and outcomes
- Distribution models are computationally expensive
- Assessing the risk with more confidence

Sample model

- Sample models can only approximate this expected outcome
  - averaging many samples together
- Sample models require less memory.

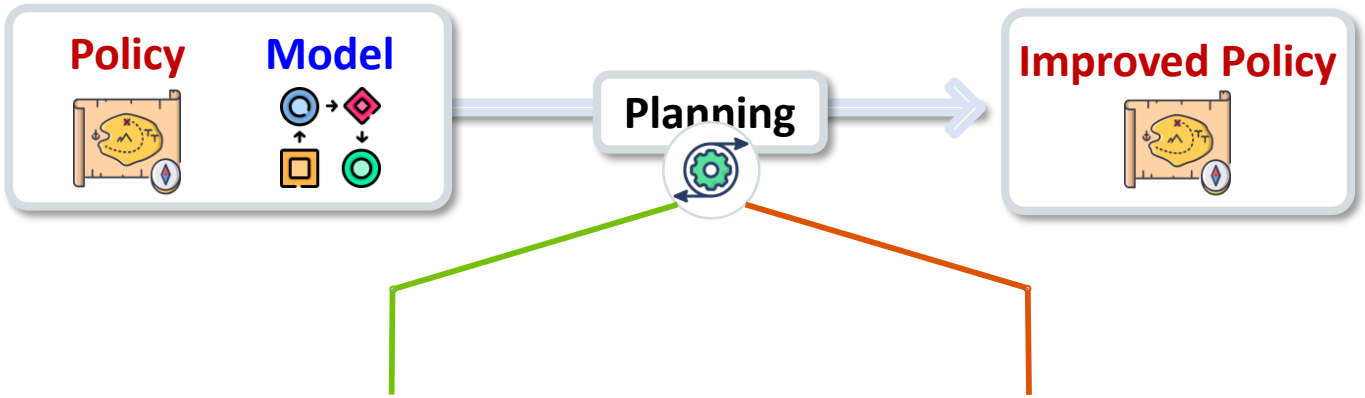
- **Example:** Medical Trials



Models and Planning

What is Planning?

- **Planning:** any computational process that takes a model as input and produces or improves a policy for interacting with the modeled environment



State-space planning

- A search through the ***state space*** for an ***optimal policy*** or an ***optimal path*** to a goal
- Actions cause transitions from state to state, and value functions are computed over states.

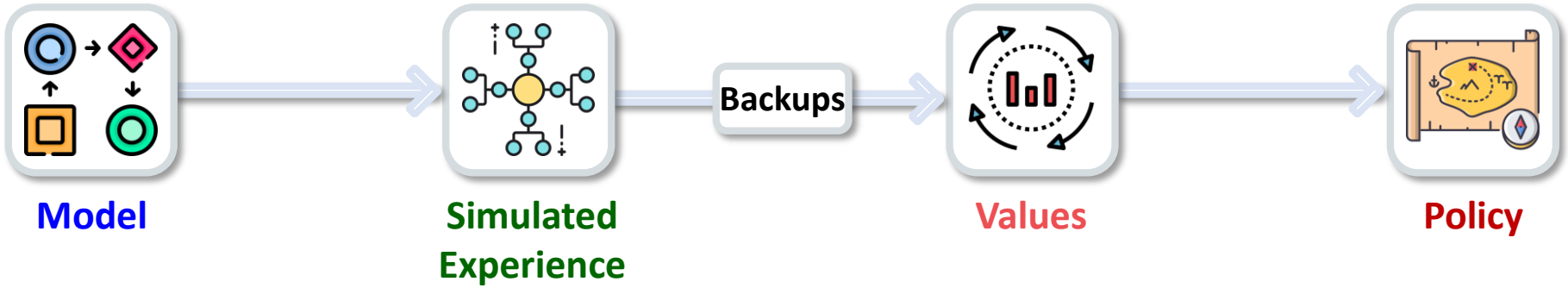
Plan-space planning

- Search through the space of plans
- Operators transform one plan into another, and value functions, if any, are defined over the space of plans.
  - i.e. **Evolutionary methods** and **partial-order planning**

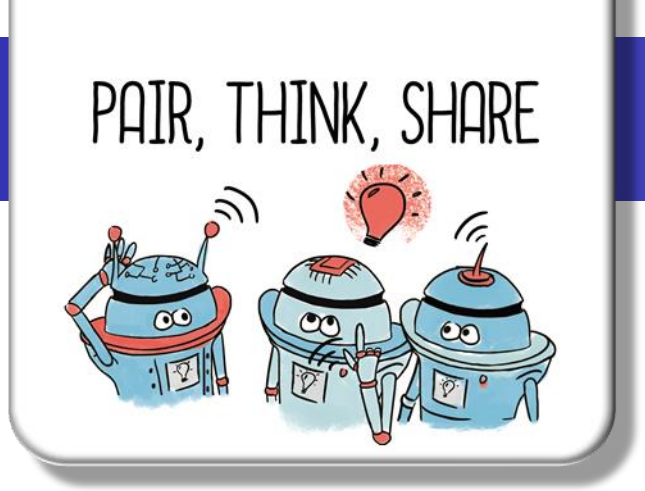
- **All state-space planning methods share a common structure.**

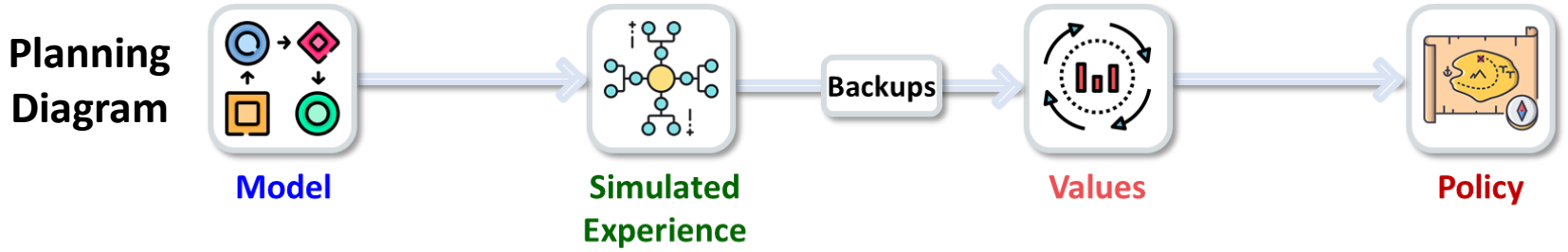
There are two basic ideas:

1. all state-space planning methods involve computing ***value functions*** as a key intermediate step toward improving the policy
2. they compute value functions by ***updates*** or ***backup operations*** applied to simulated experience



Dynamic Programming

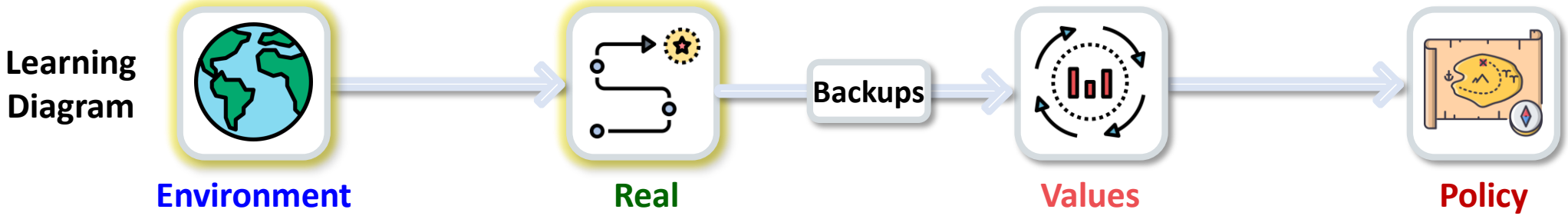




Learning Diagram vs Planning Diagram:

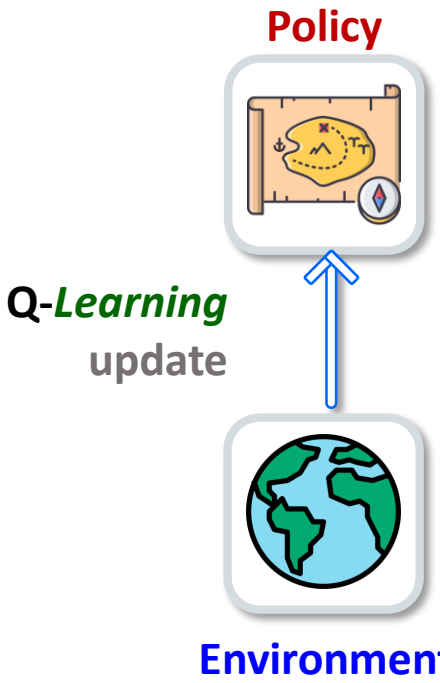
The heart of both ***learning*** and ***planning*** methods is the **estimation of value functions** by backing-up update operations.

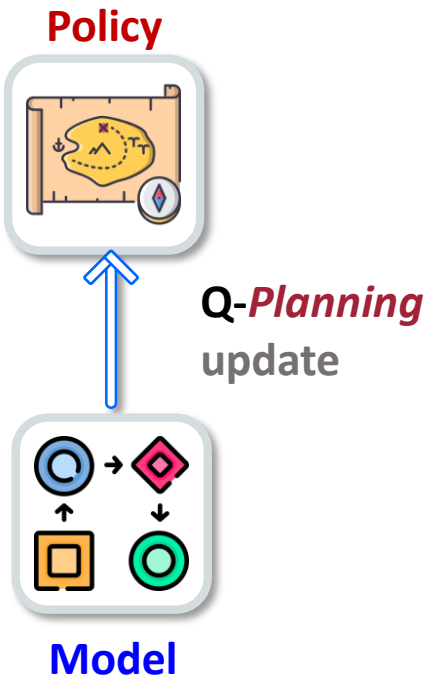
How we can modify the shown "Planning Diagram" to design the "Learning Diagram"



Models and Planning

Q-Planning





- **Q-learning** learns based on samples directly drawn from the **environment**
- This is called **direct RL**

- **Q-planning** takes the advantage of **model** to better inform decision-making without having to interact with the environment
- This is called **indirect RL**

Pseudocode: Q-Planning

Random-sample one-step tabular Q-planning

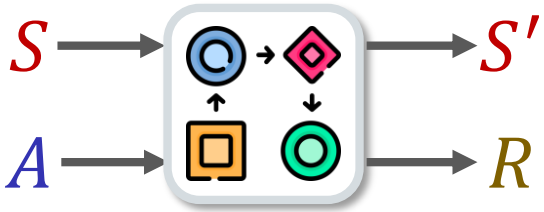
1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a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

1 Initialization

$\mathcal{S} \longrightarrow S$

$\mathcal{A} \longrightarrow A$

2 Sampling



3 Q-Learning Update

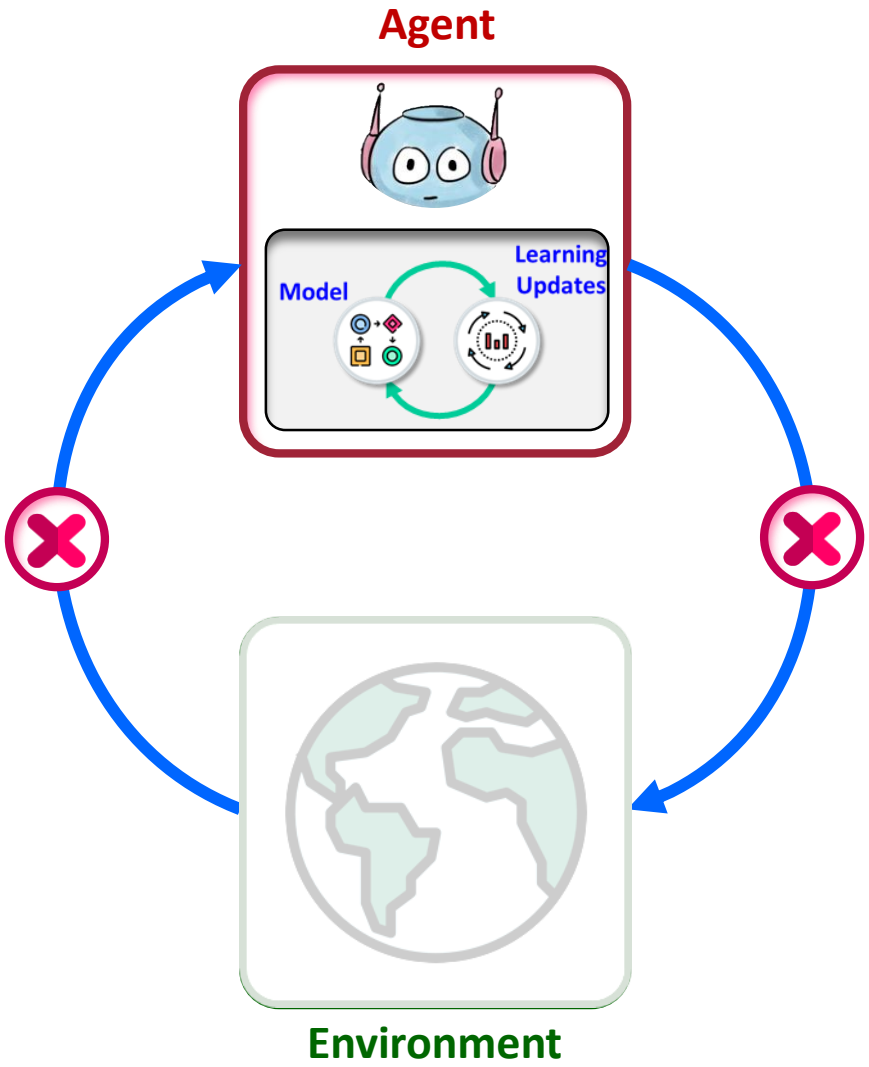
$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$$

4 Greedy Policy Improvement

$$\pi(S) = \operatorname{argmax}_a Q(S,A)$$

Models and Planning

Q-Planning structure



Models and Planning

The unified approach

Model–based RL

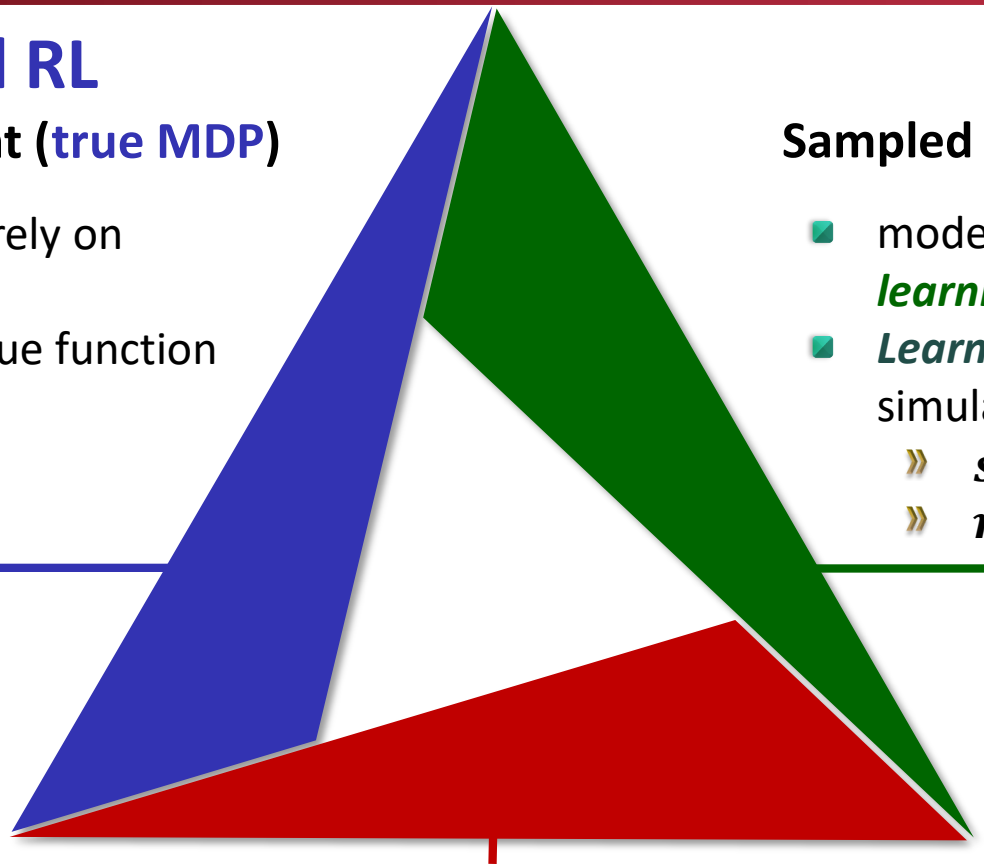
Sampled from environment (true MDP)

- Model-based methods rely on ***planning***
- ***Plan*** a policy and/or value function from real experience
  - $s' \sim P(s'|s,a)$
  - $r = R(s,a)$

Model–free RL

Sampled from model (approximate MDP)

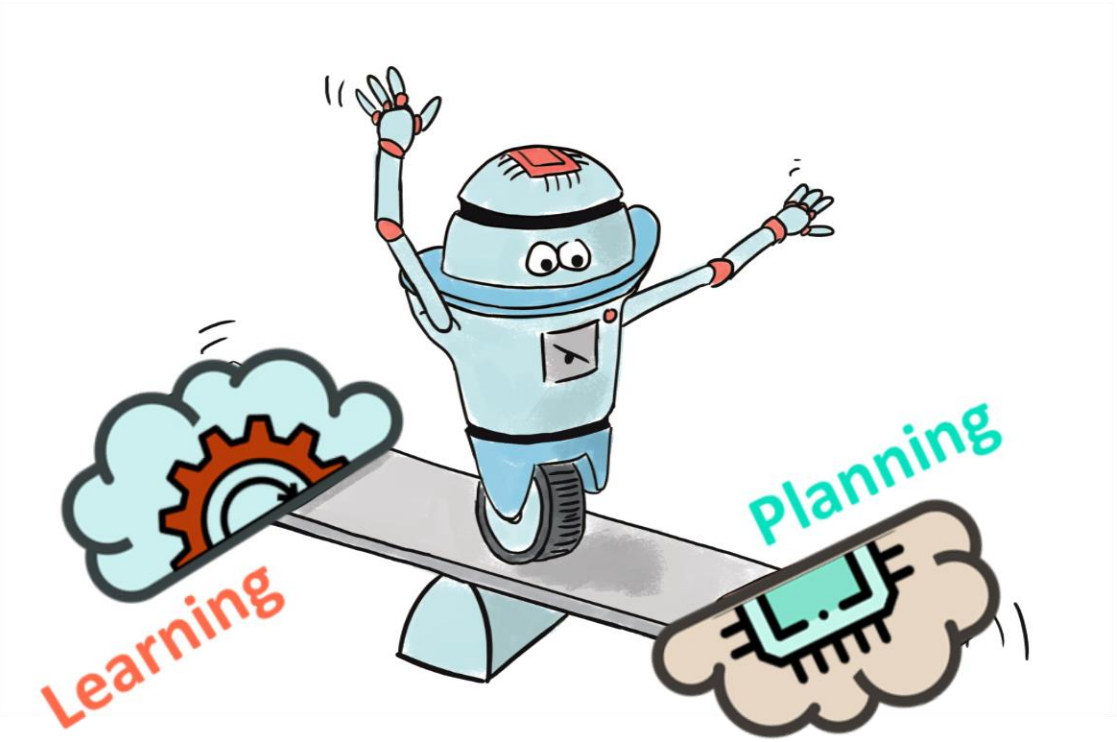
- model-free methods primarily rely on ***learning*** (***No*** model)
- ***Learn*** a policy and/or value function from simulated experience
  - $s' \sim P_\theta(s'|s,a)$
  - $r = R_\theta(s,a)$



Model–based + Model–free

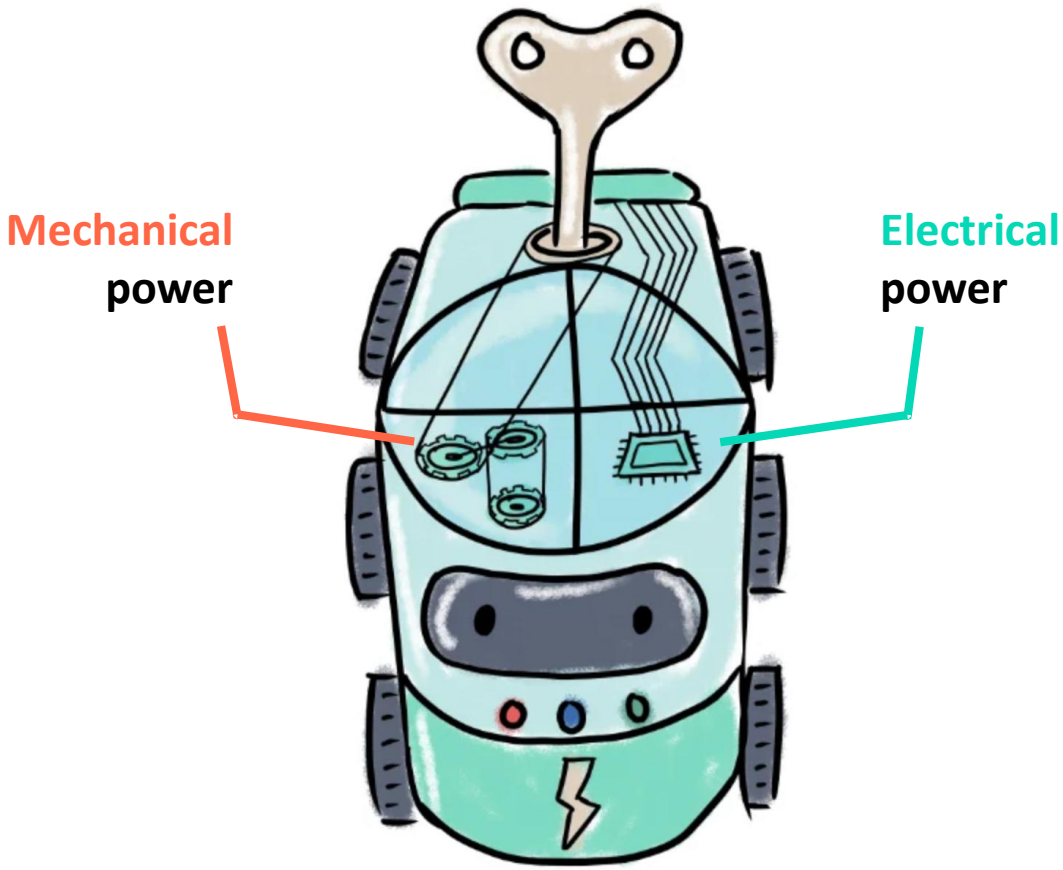
- Learn a model from real experience
- ***Learn*** and ***plan*** from real and simulated experience

Dyna-Q

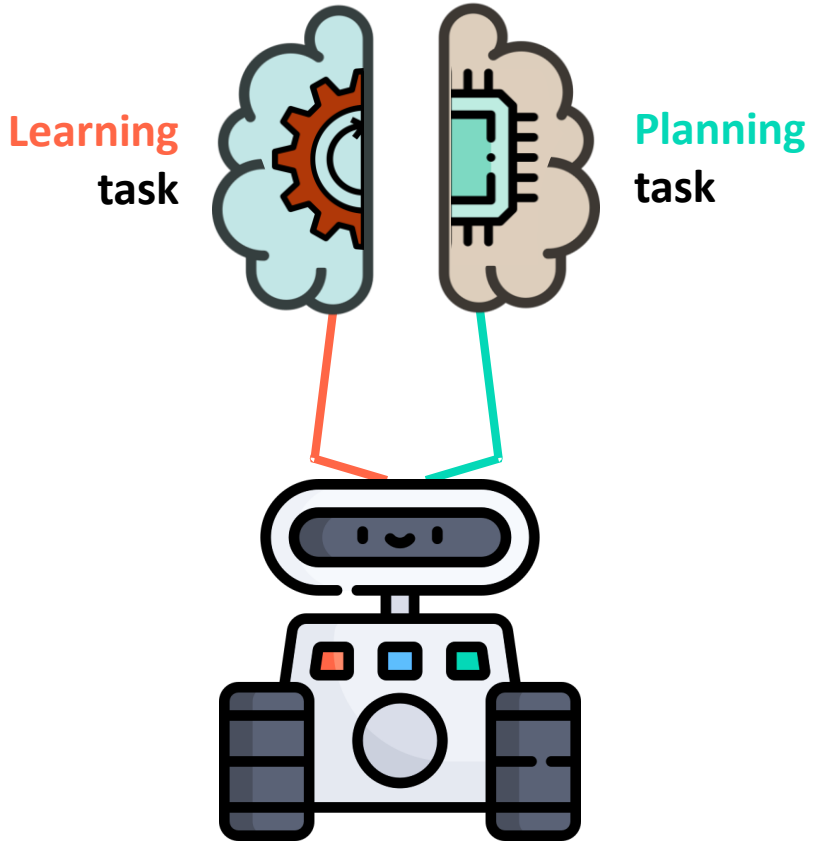


Dyna: Integrated Planning, Acting, and Learning

Dyna-Q agent



Hybrid Car



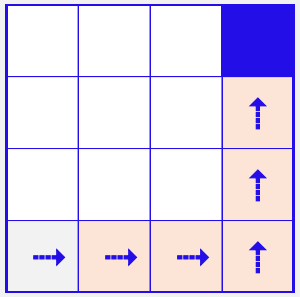
Dyna-Q agent

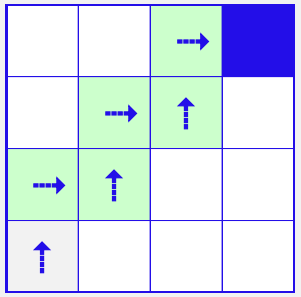
Dyna: Integrated Planning, Acting, and Learning

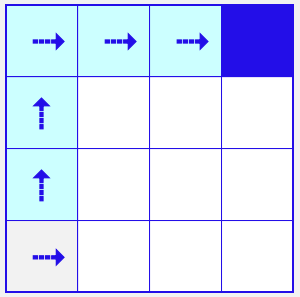
Simultaneous learning and planning (overview)

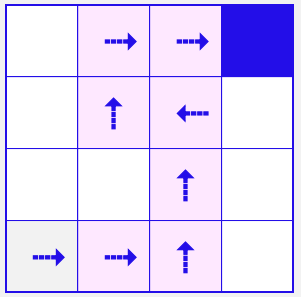
Learning task

Environment









Planning task

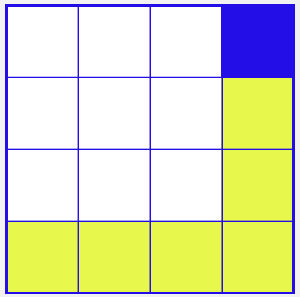
Episode 1

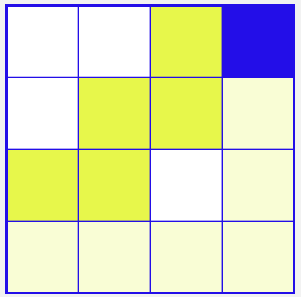
Episode 2

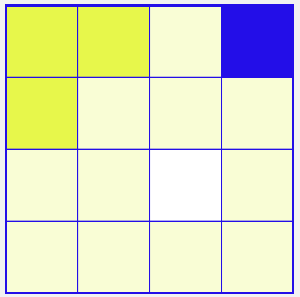
Episode 3

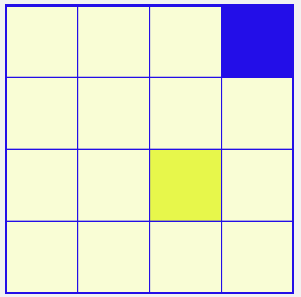
Episode 4

Model







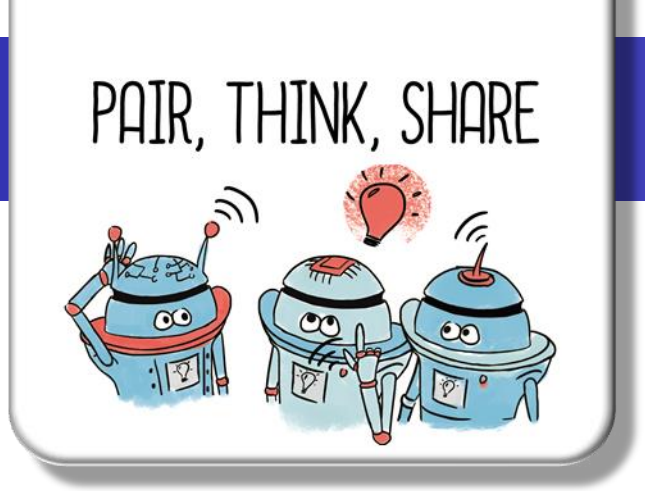


Dyna: Integrated Planning, Acting, and Learning

Simultaneous learning and planning (closer look)

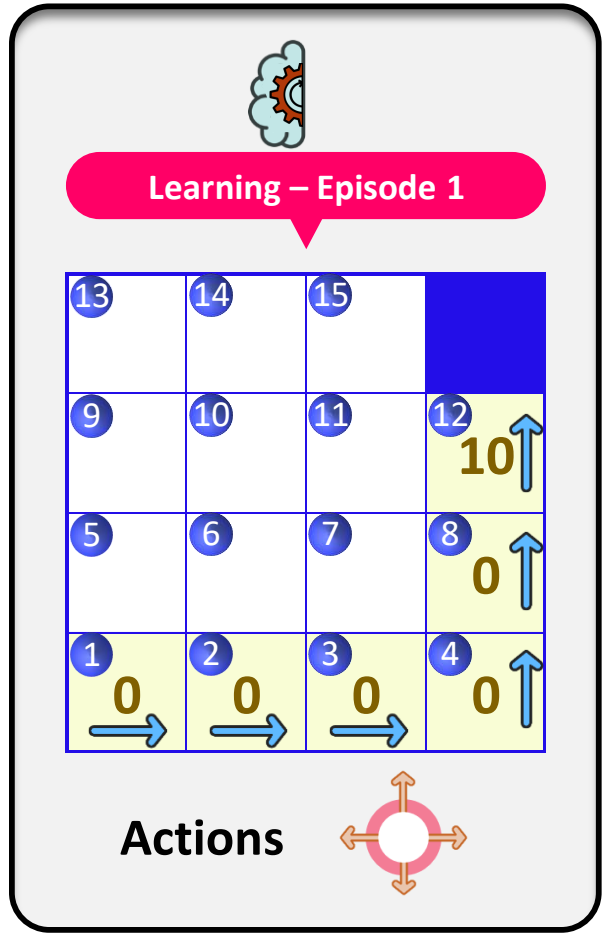
- The agent only change the value of the state action pair ***beside*** the goal.
- A ***large number of steps*** needs to be taken in each episode to obtain value function

- Planning provides many ***more updates*** to the value function
- It makes ***better*** use of its ***limited interaction*** with the environment.



Learning Phase

The following figure shows the action value results of *Learning* after completion of the first (1st) episode.

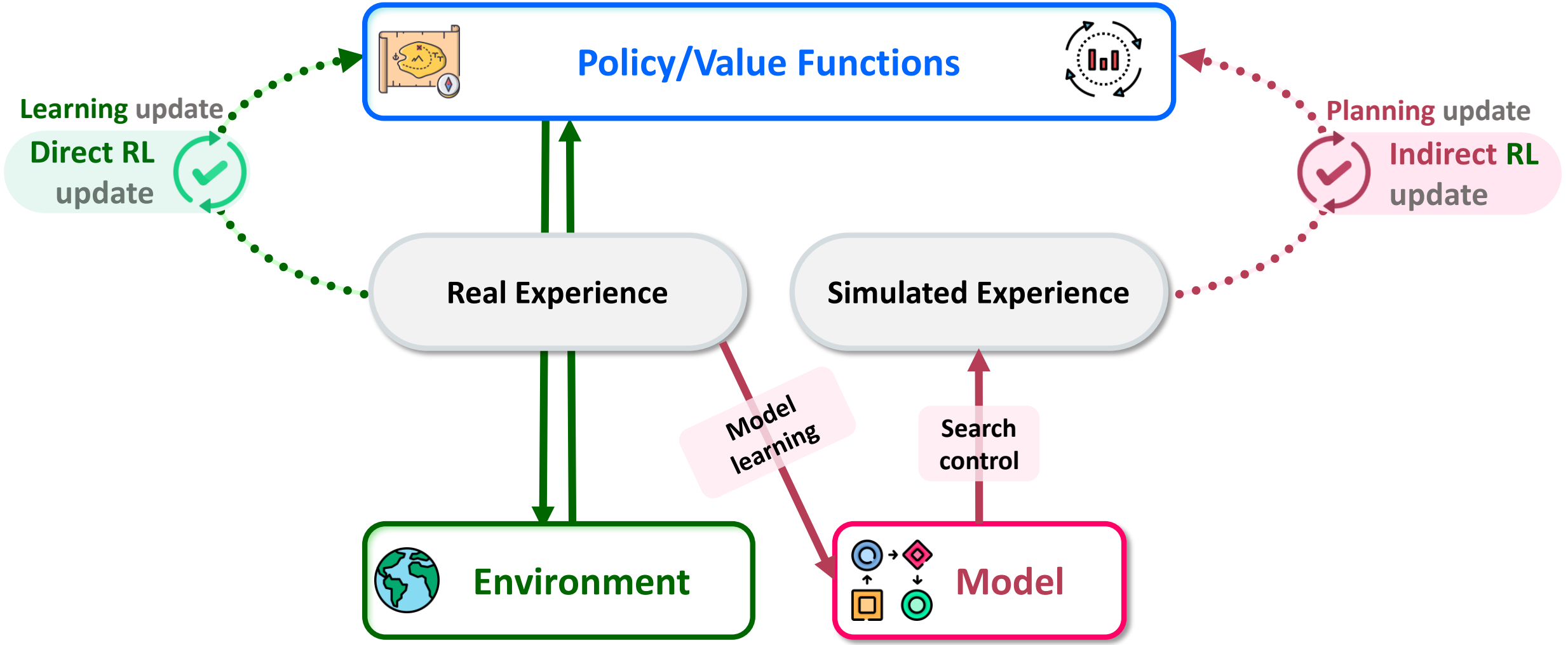


Planning Phase (1-step update)

- **Which state-action pairs can be selected during the planning phase?**
- **Does the order of selection matter?**
- **If yes, how you would prioritize the *selection* operation *(Planning Strategy)***

Dyna: Integrated Planning, Acting, and Learning

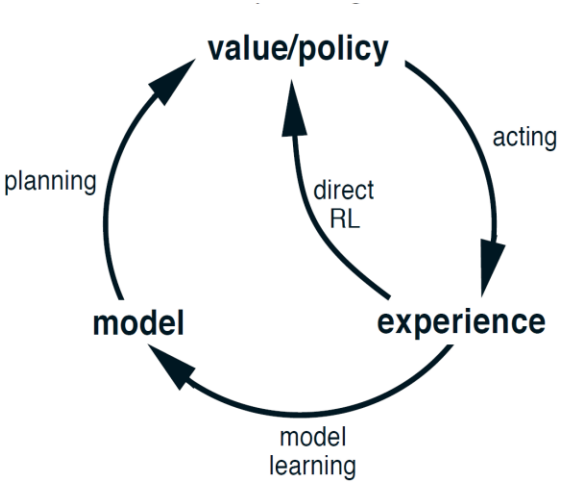
Structure



Dyna: Integrated Planning, Acting, and Learning

Model Learning and Search control

- **Model Learning /updating**
  - **The model-learning method assumes the environment is deterministic.**
  - **Model learning is to implement a lookup table (tabular).**
  - **Once a new transaction occurs**:
    - transition and its results will be tabulated (agent memorizes the next state and reward for the given state action pair)
    - then, if the model is queried with a state–action pair that has been experienced before, it simply returns the last-observed next state and next reward as its prediction



- **Search control**
  - **The process that selects the starting states and actions for the simulated experiences generated by the model**

Dyna-Q

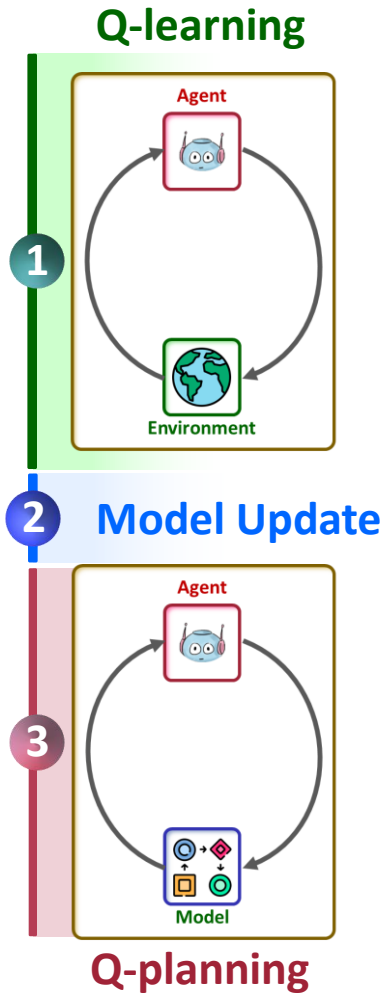
Pseudocode: Dyna-Q

Tabular Dyna-Q

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

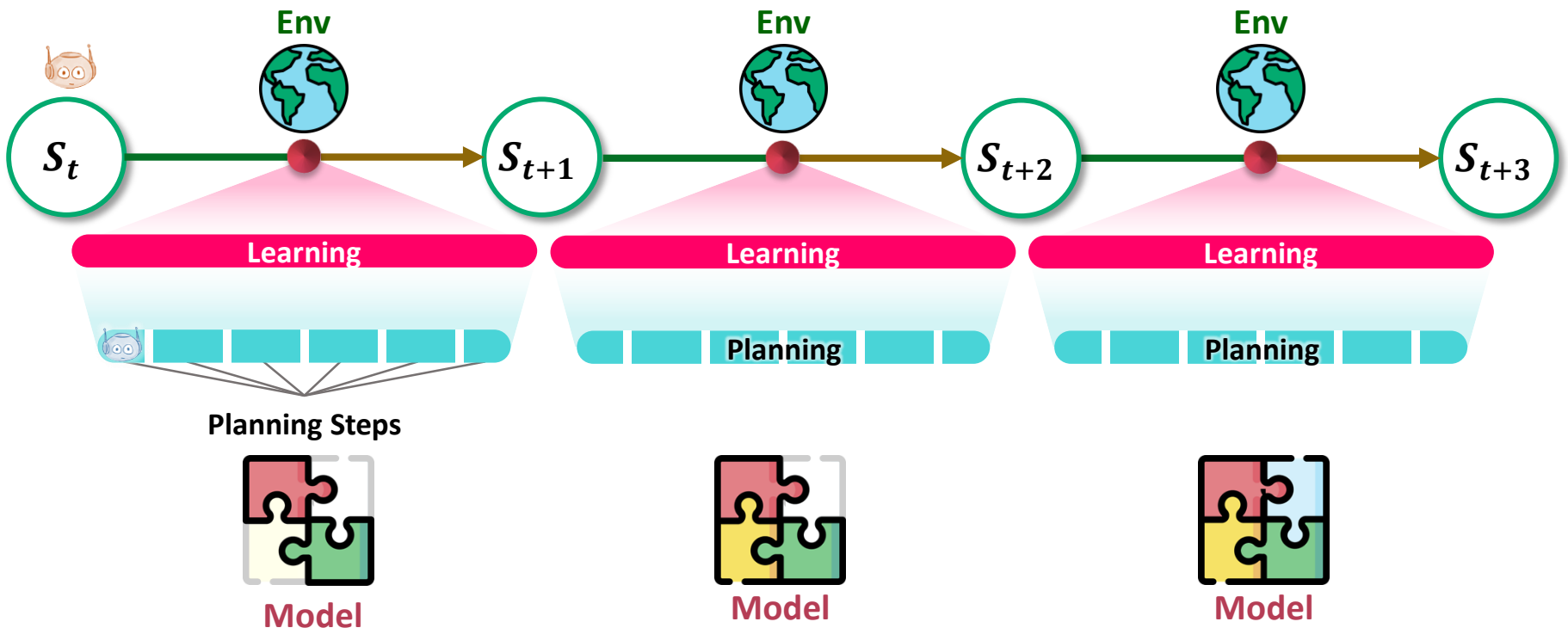
Loop forever:

1. (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) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
2. (e) $Model(S,A) \leftarrow R, S'$ assuming deterministic environment
3. (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: Integrated Planning, Acting, and Learning

Advantage of Planning

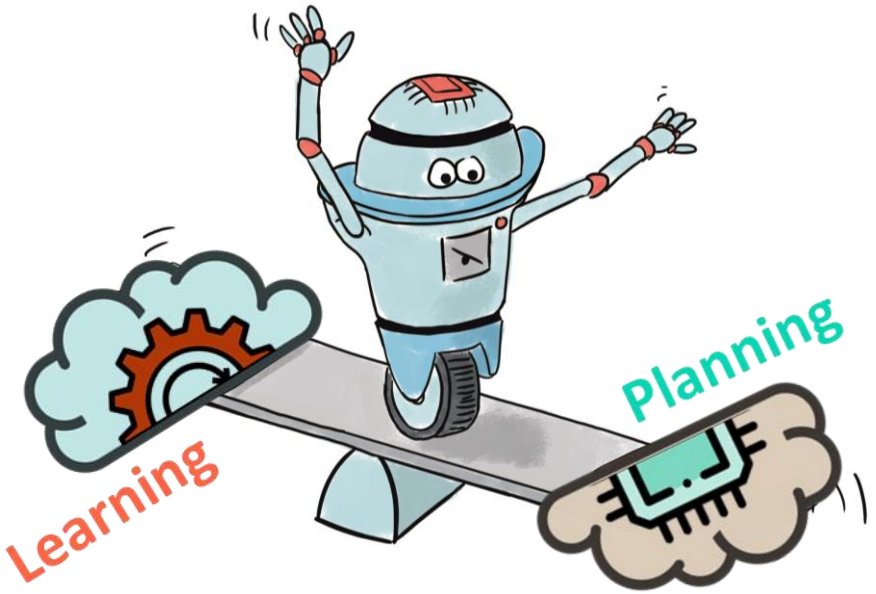


- **Learning and Planning perform in parallel with the interaction loop**
- **Learning updates can be executed relatively fast.**
  - Query from the tabular model (a look-up table)

Dyna: Integrated Planning, Acting, and Learning

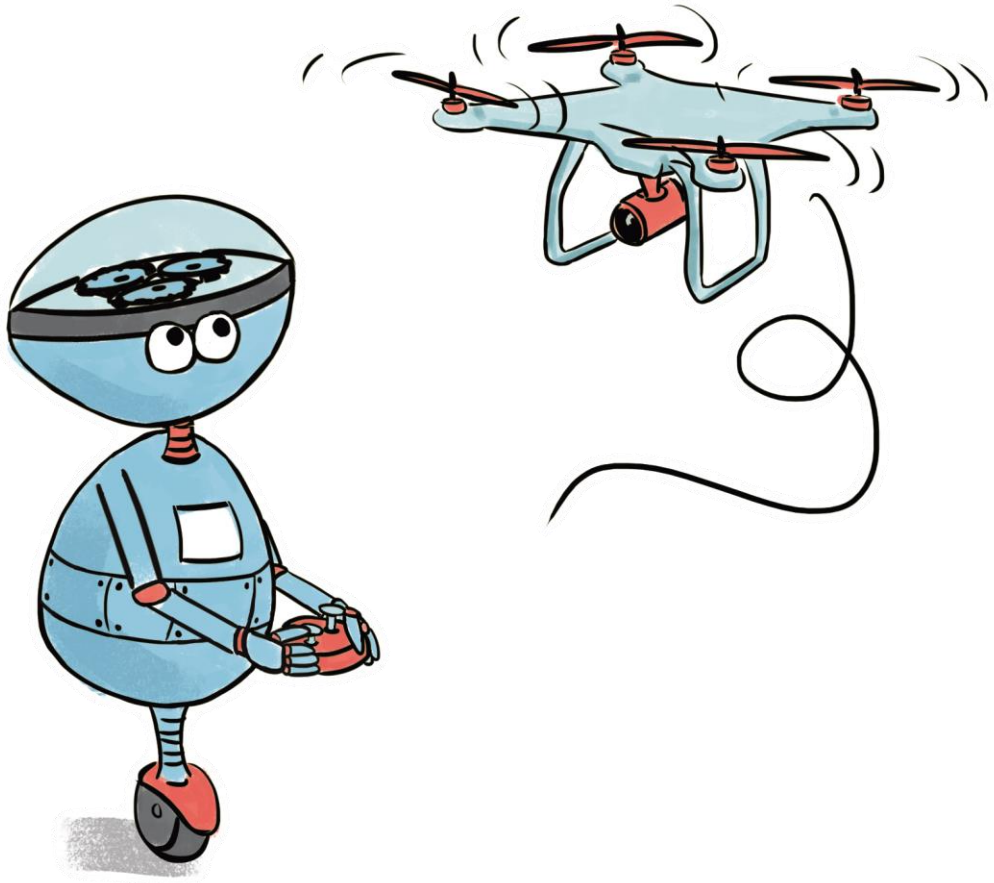
Computation trade-offf

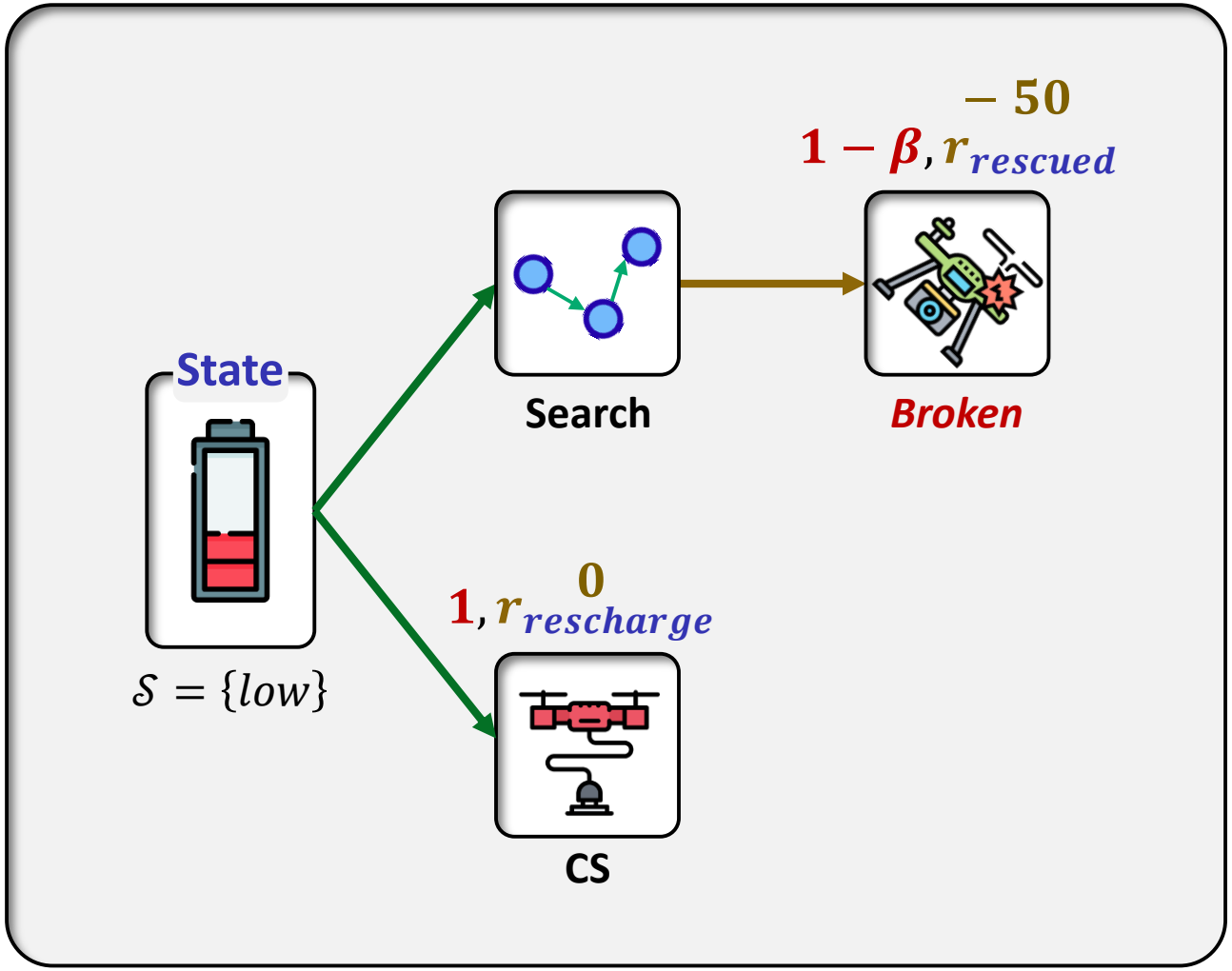
- **Decision making and model learning are both computation-intensive processes,**
- **The available computational resources may need to be divided between them.**



Dyna: Integrated Planning, Acting, and Learning

Advantage of Planning: Drone Example

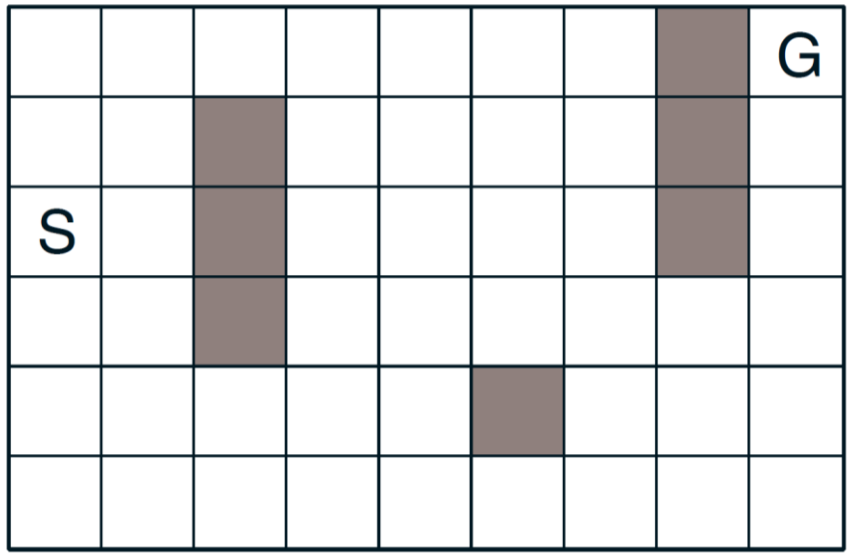




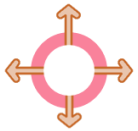
Dyna: Integrated Planning, Acting, and Learning

Example: The Dyna Maze

Problem Settings



Actions



Rewards

$R(s) = 0$

$R(G) = +1$

$\gamma = 0.95$

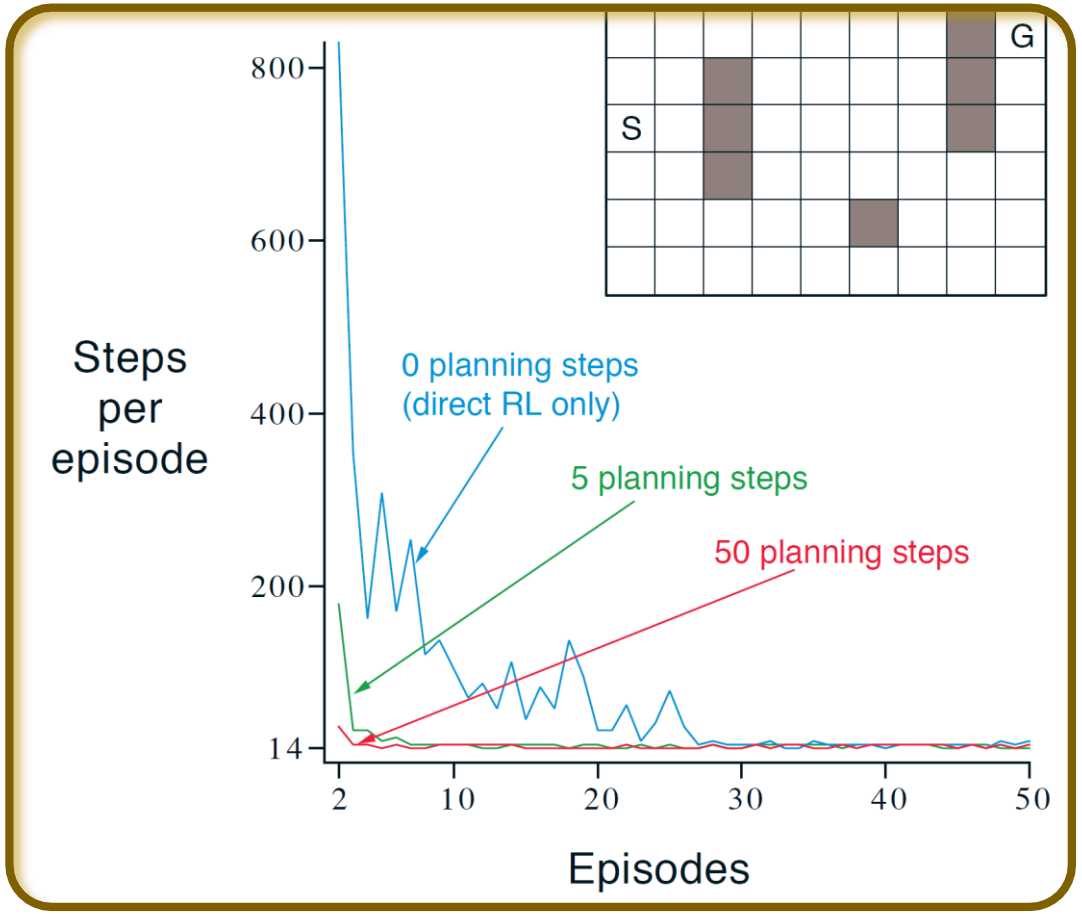
$\alpha = 0.1$

$\varepsilon = 0.1$

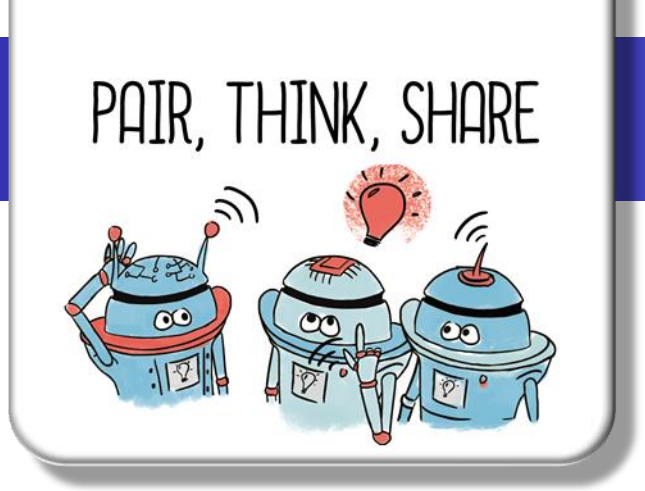
Dyna: Integrated Planning, Acting, and Learning

Example results (1): The Dyna Maze

the number of steps taken by the agent to reach the goal

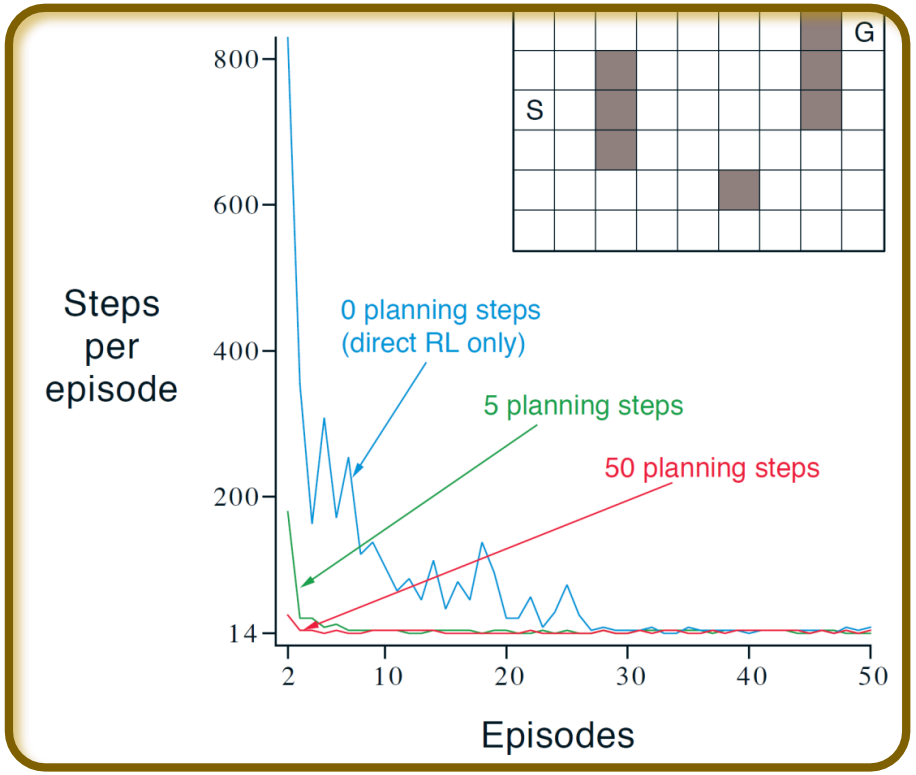


- **Average results of 30 replications**
- **Results:** No. of episodes to reach (ε-)optimal performance
  - **Non-planning agent** ~ **Q-learning agent**
    - $n = 0$: slowest agent on this problem (25 episodes)
  - **Planning agents** ~ **Dyna-Q agent:**
    - $n = 5$: takes 5 episode to complete
    - $n = 50$: takes 3 episodes to complete
- **Performance improved for all values of n, but much more rapidly for larger values.**



The Dyna Maze:

The first episode was exactly the same (about 1700 steps) for all values of n (constant seed) – therefore not shown in the graph



1. **Why the first episode takes too long?**

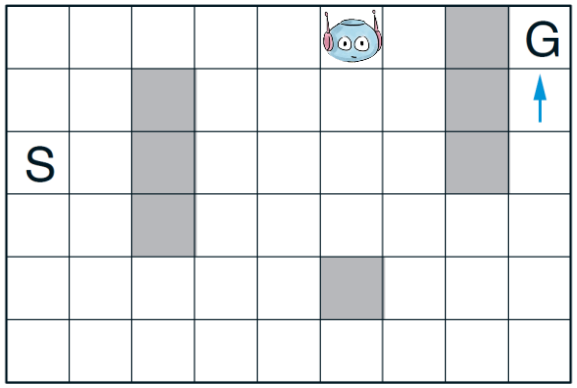
2. **Why there is not a significant difference between the performance of Q-learning (n=0) and Dyna-Q on the first episode?**

Dyna: Integrated Planning, Acting, and Learning

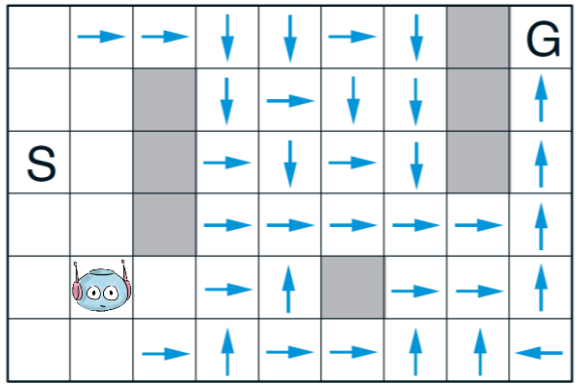
Example results (2): The Dyna Maze

halfway through the second episode

WITHOUT PLANNING (n=0)

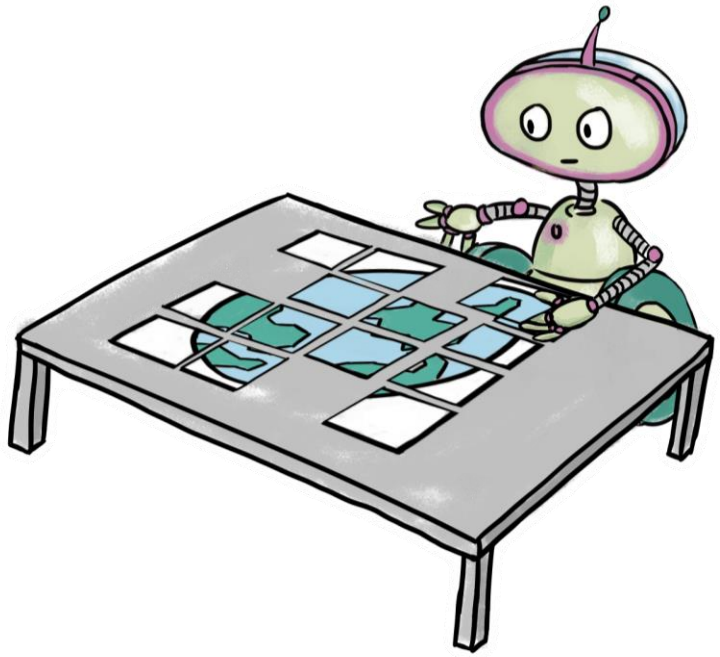


WITH PLANNING (n=50)



- **The *planning* agents found the solution so much faster than the *non-planning* agent**
- **Results:** No. of episodes to reach (ε-)optimal performance
  - **Non-planning agent** ~ **Q-learning agent** $(n = 0)$
    - each episode adds only one additional step to the policy
    - only one step (the last) has been learned so far
  - **Planning agents** ~ **Dyna-Q agent** $(n = 50)$
    - only one step is learned during the first episode
    - During the second episode an extensive policy has been developed that by the end of the episode will reach almost back to the start state.

When the Model is Wrong



When the Model is Wrong

Planning with an Inaccurate Model

- **Models may be incorrect because:**

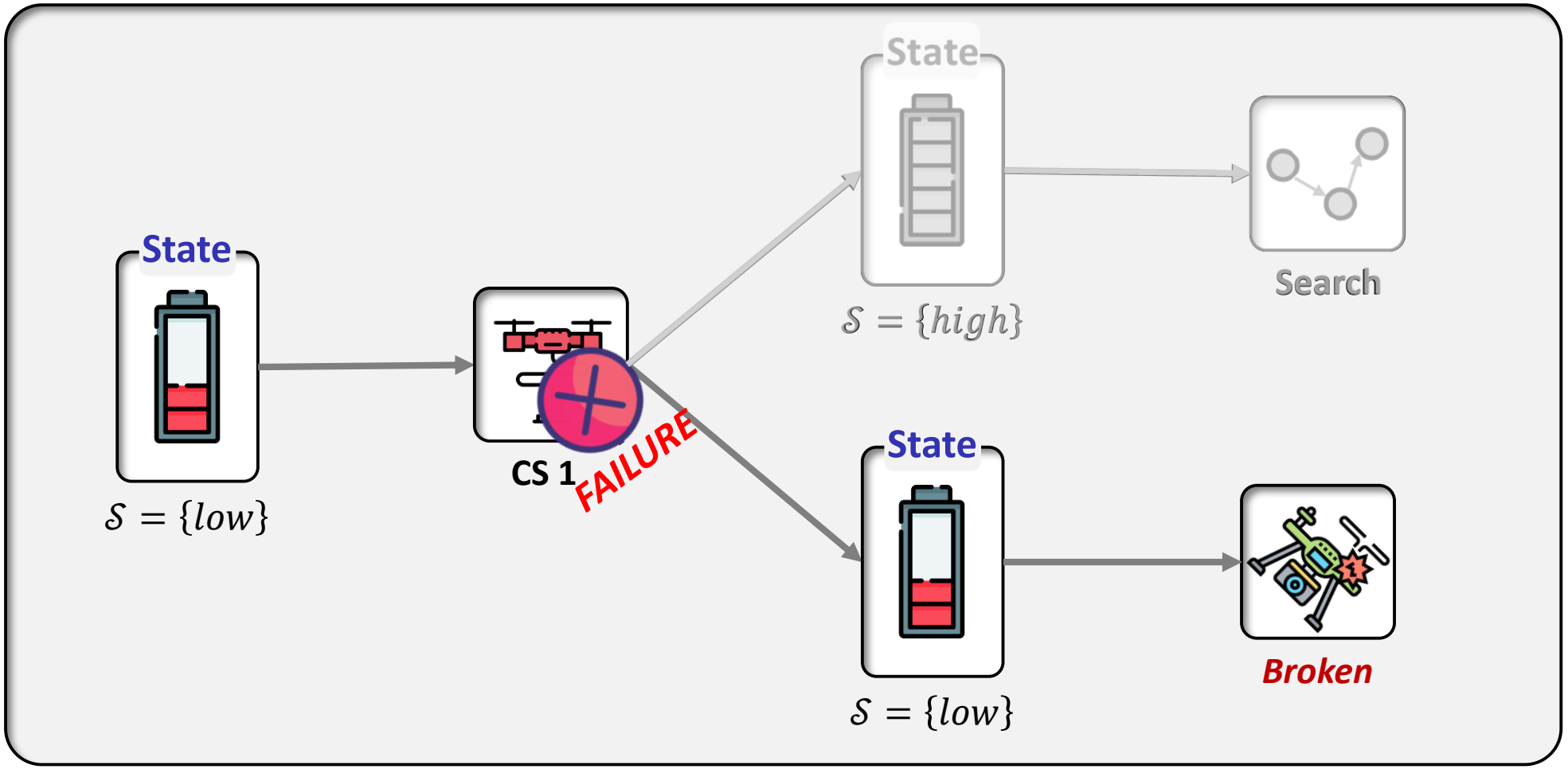
  1. the environment is stochastic and only a limited number of samples have been observed
  2. the model was learned using function approximation that has generalized imperfectly
  3. environment has changed and its new behavior has not yet been observed

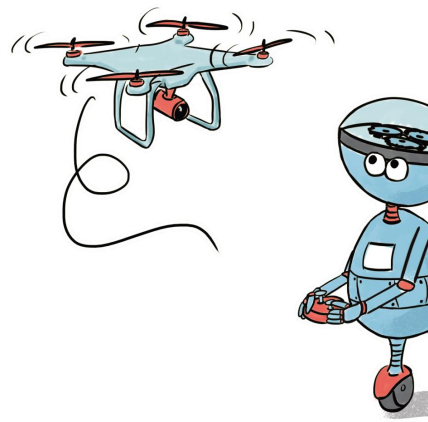
- **What is the consequence?**
  - When the model is incorrect, the planning process is likely to compute a suboptimal policy.

- **Solution**
  - In some cases, the suboptimal policy computed by planning quickly leads to the discovery and correction of the modeling error.

When the Model is Wrong

Inaccurate Model: Drone Example





When the Model is Wrong

Dyna-Q+ with exploration bonus

- **Uses an "exploration bonus":**
  - **Keeps track of time since each state-action pair was tried for real**

Actual reward

Small constant

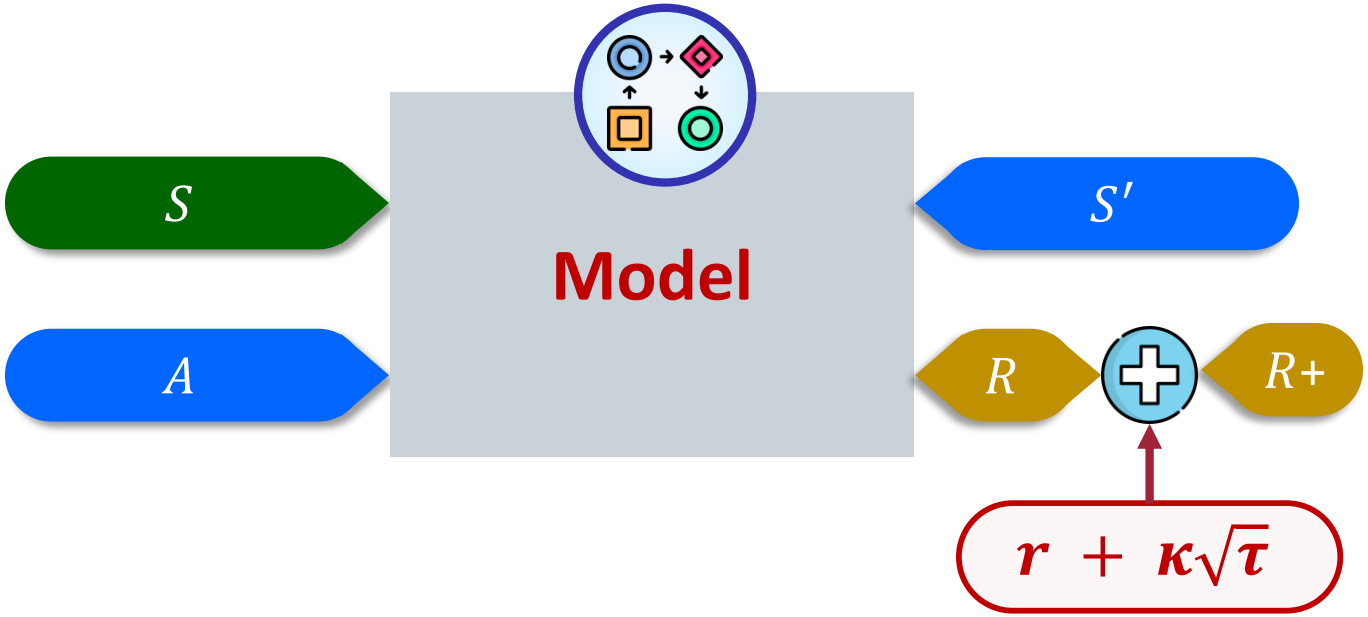
$$\text{New reward} = r + \kappa\sqrt{\tau}$$

Time step since the transition was last tried

- **An extra reward is added for transitions caused by state-action pairs related to how long ago they were tried: the longer unvisited, the more reward for visiting**
- **The agent actually "plans" how to visit long unvisited states**

When the Model is Wrong

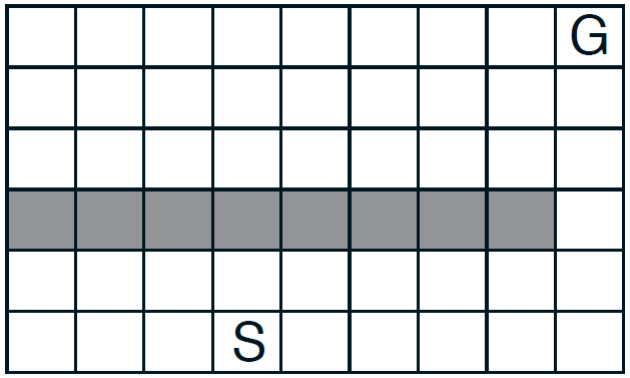
Dyna-Q+ structure



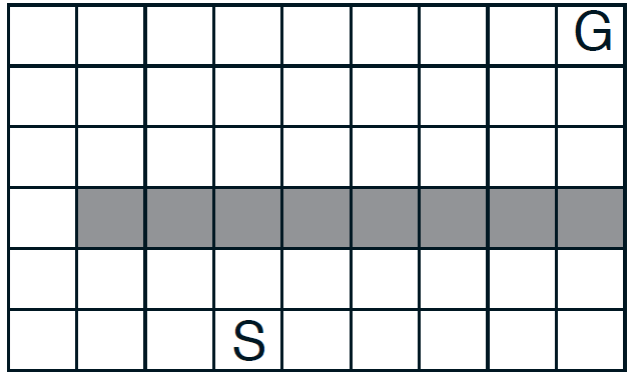
- As τ grows, the bonus becomes bigger and bigger.
  - **Eventually, planning will change the policy to go directly to S due to the large bonus.**
  - **When the agent finally visit state S, it might see a big reward, or it might be disappointed.**

When the Model is Wrong

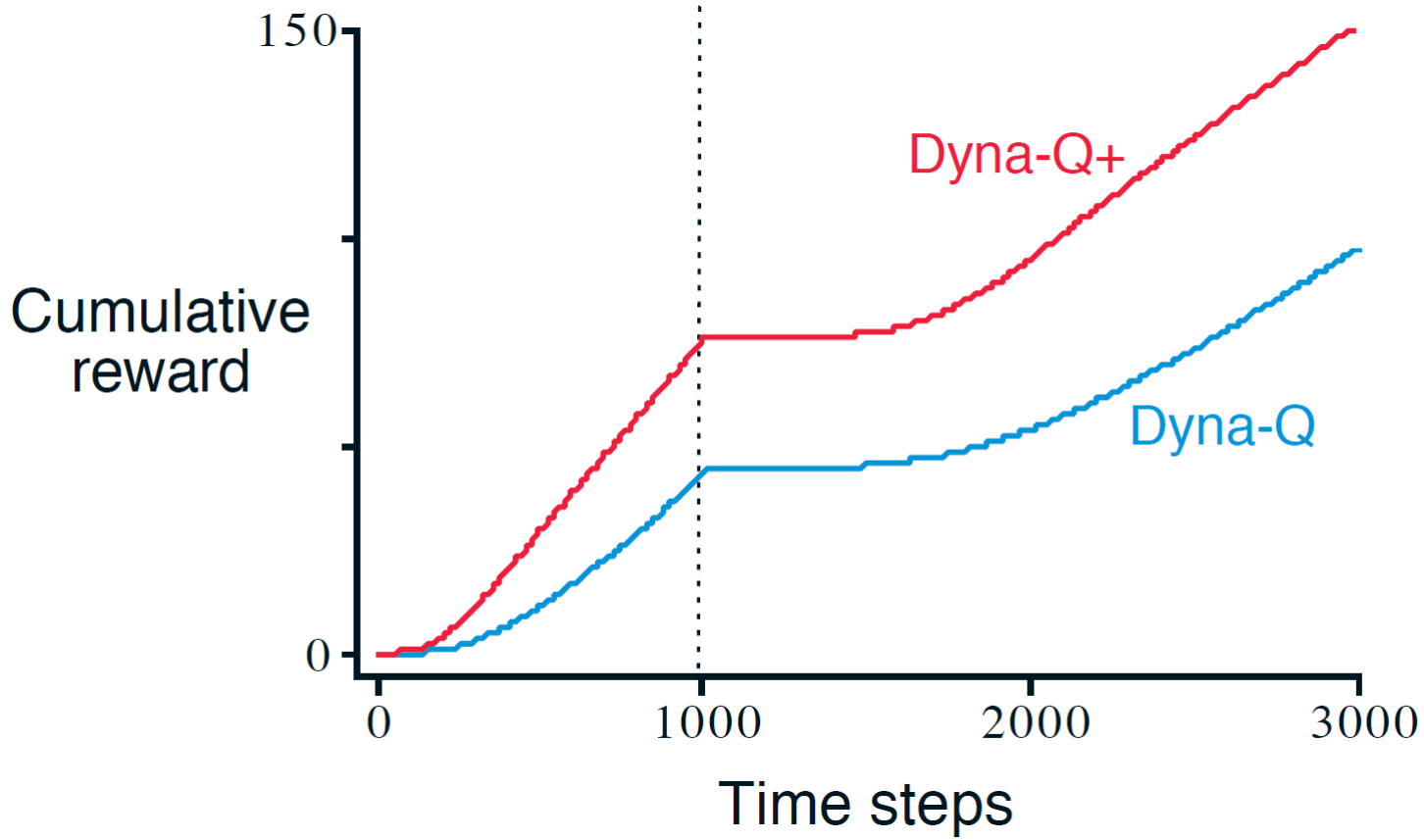
Example: Blocking Maze



Initial Environment

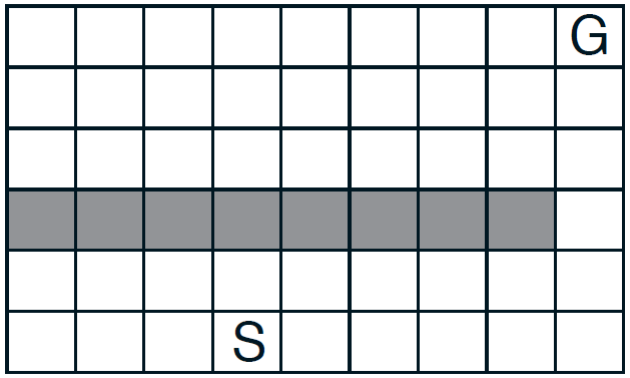


After 1000 time steps

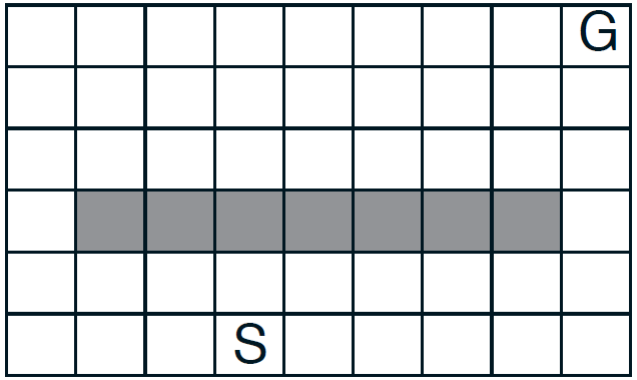


When the Model is Wrong

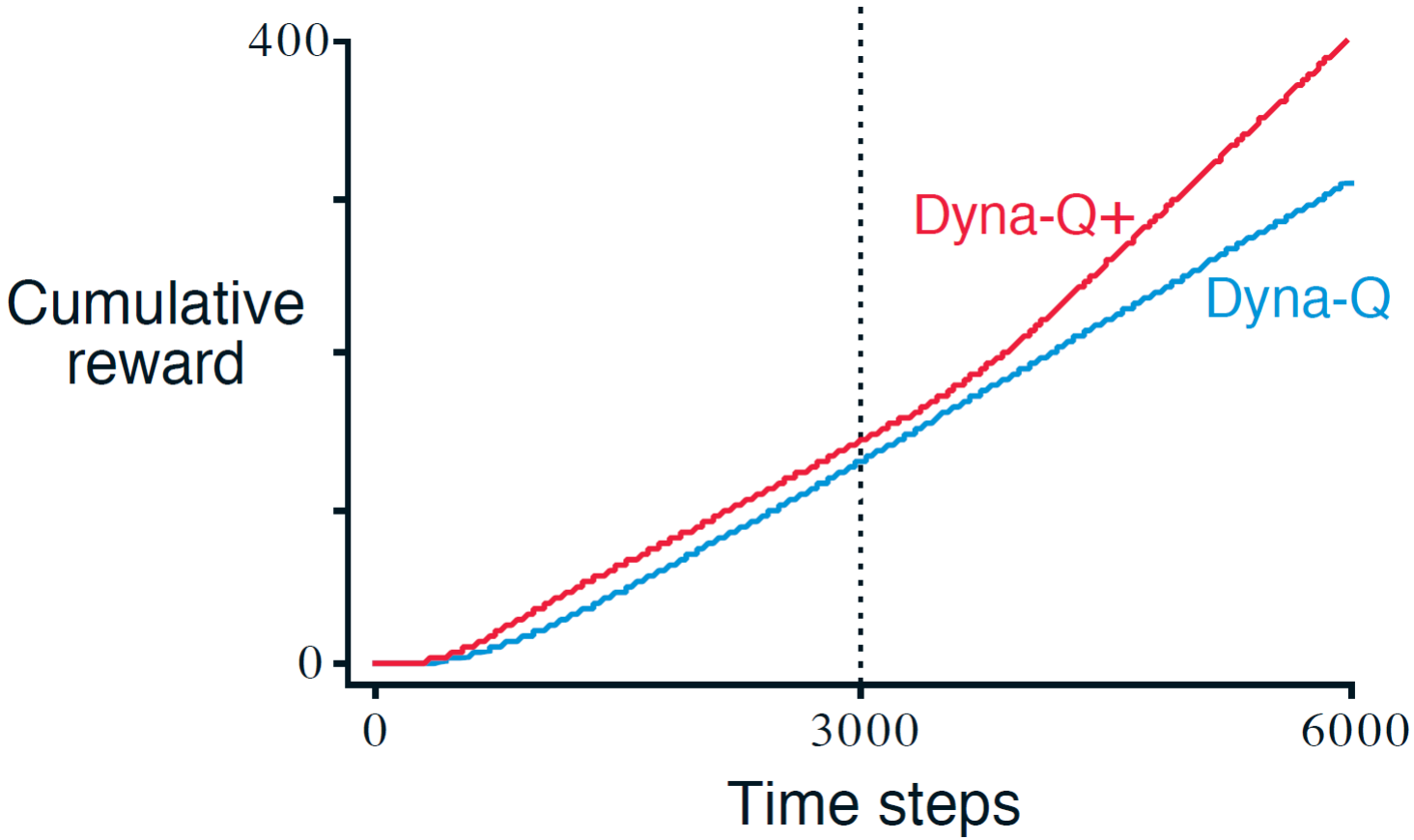
Example: Changing world in a better way

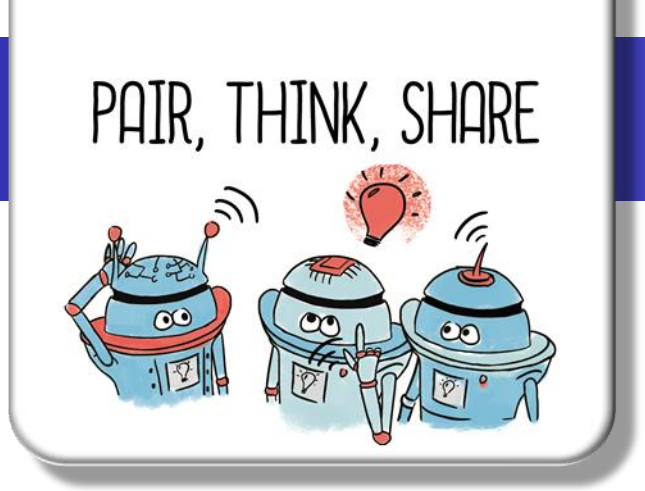


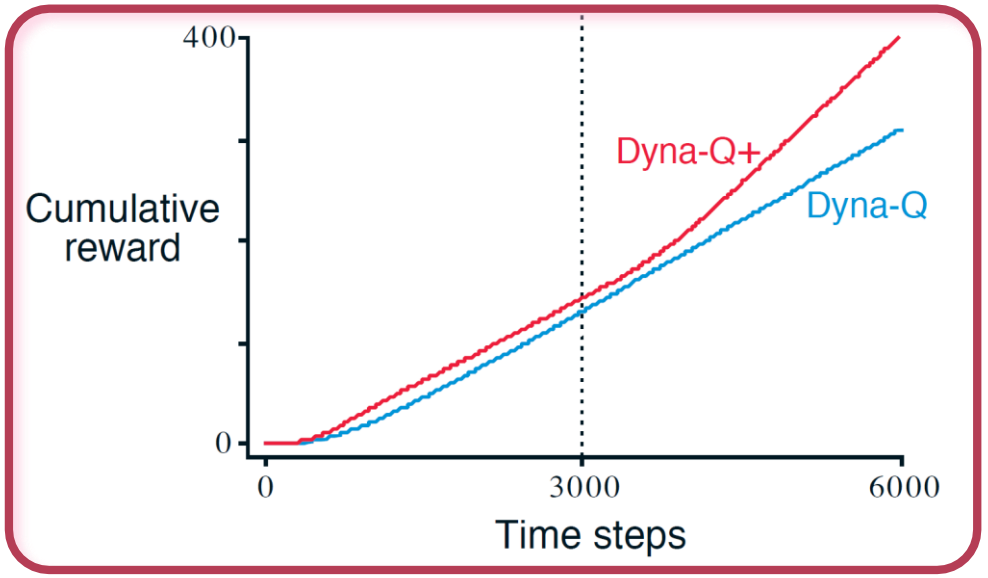
Initial Environment



After 3000 time steps







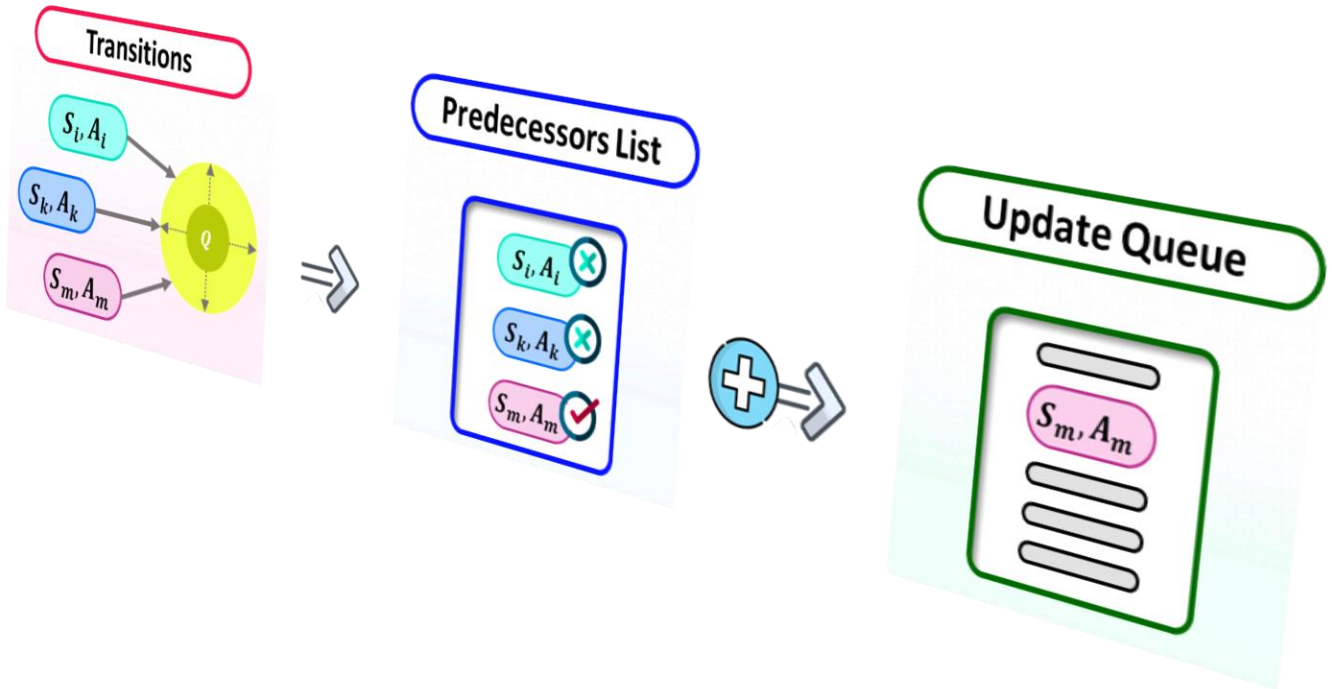
Dyna-Q+ vs. Dyna-Q

Careful inspection of Shortcut Maze results reveals that the difference between Dyna-Q+ and Dyna-Q.

Why this difference narrowed slightly over the first part of the experiment.

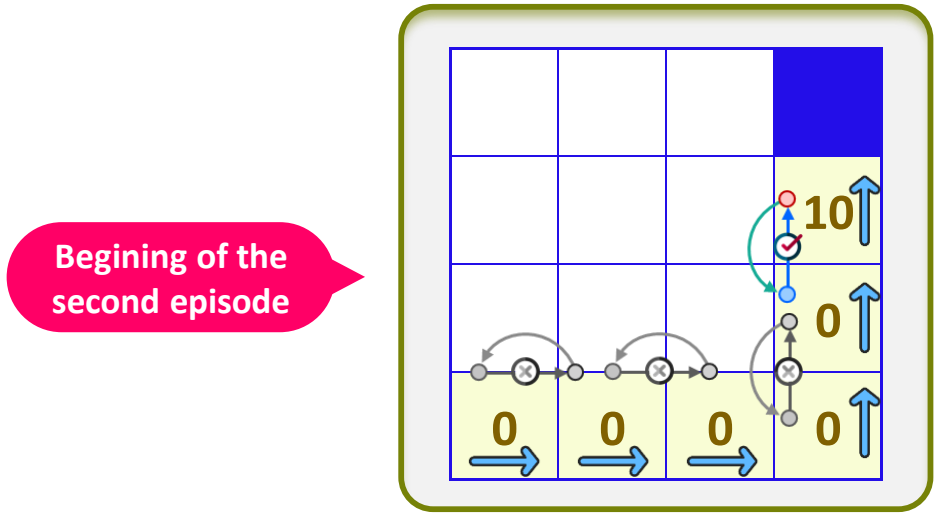
Dyna-Q+ is at a slight disadvantage in that its exploration bonus causes it to constantly explore. This worked to its advantage earlier in that it helped it to discover the change in the environment faster. But now that the environment is stable, the constant exploring actually causes Dyna-Q+ to do somewhat worse. Such is the cost of persistent curiosity.

Prioritized Sweeping



Prioritized Sweeping

- **Unfocused search** (i.e. Dyna-Q)
  - **If simulated transitions are generated uniformly, then many wasteful updates will be made before stumbling onto one of these useful ones**

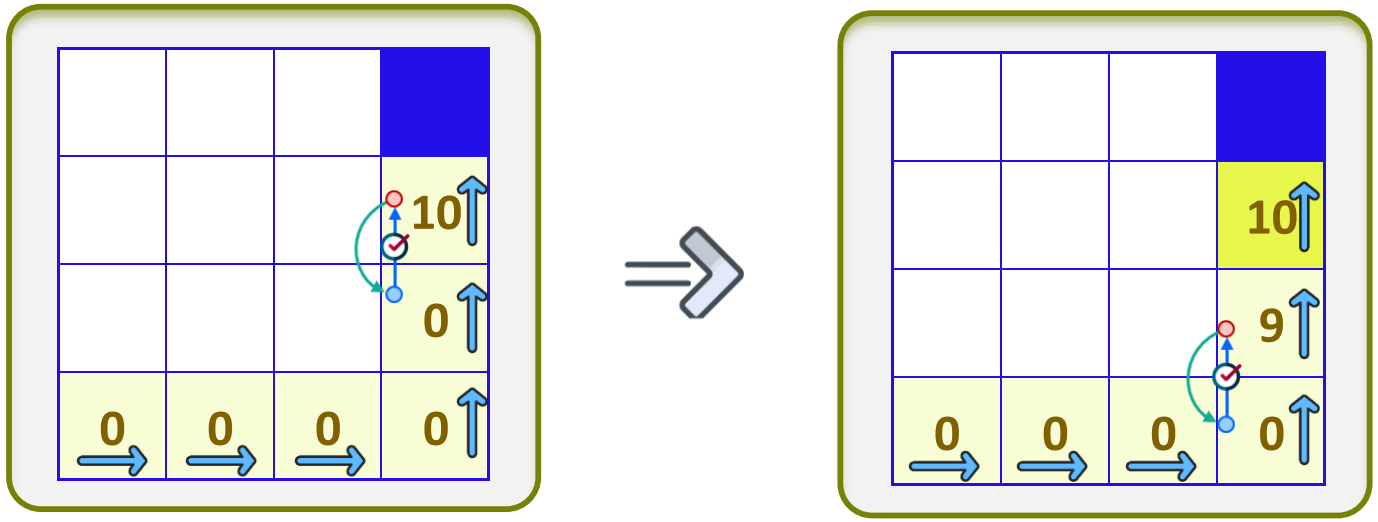


  - **In the much larger problems that are our real objective, the number of states is so large that an unfocused search would be extremely inefficient.**

Prioritized Sweeping

Planning strategies: Backward focusing

- **Backward focusing** (i.e. Prioritized Sweeping)
  - **Work backwards from states whose values have just changed:**
    - Maintain a queue of state-action pairs whose values
    - Would change a lot if backed up, prioritized by the size of the change
    - When a new backup occurs, insert predecessors

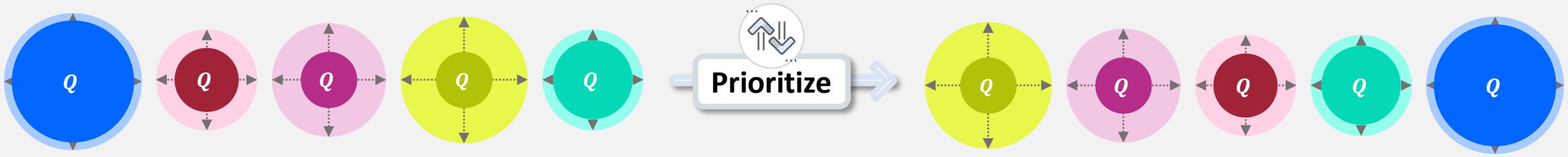


  - **The predecessor pairs of those that have changed a lot are more likely to also change a lot.**

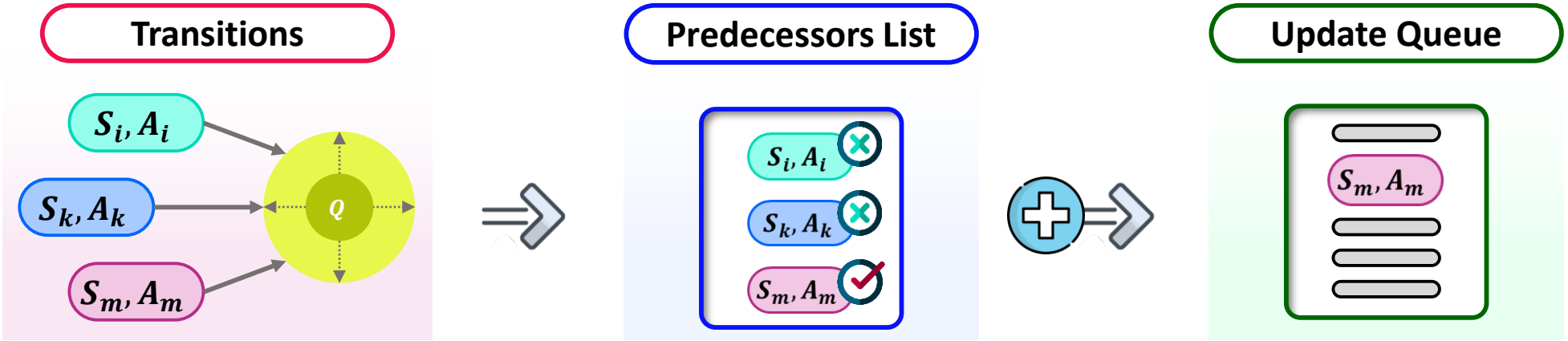
Prioritized Sweeping

Prioritized sweeping concept

- **Prioritized sweeping**
  - A queue is maintained of every state–action pair whose estimated value would change nontrivially if updated , prioritized by the ***size*** of the change.



  - When the top pair in the queue is updated, the effect on each of its predecessor pairs is computed.



Prioritized Sweeping

Pseudocode: Prioritized sweeping

Prioritized sweeping for a deterministic environment

Initialize $Q(s,a)$, $Model(s,a)$, for all s, a, and $PQueue$ to empty
Loop forever:

1
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow policy(S,Q)$
- (c) Take action A; observe resultant reward, R, and state, S'

2
- (d) $Model(S,A) \leftarrow R, S'$

3
- (e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$.
- (f) if $P > \theta$, then insert S, A into $PQueue$ with priority P

4
- (g) Loop repeat n times, while $PQueue$ is not empty:
  - $S, A \leftarrow first(PQueue)$
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

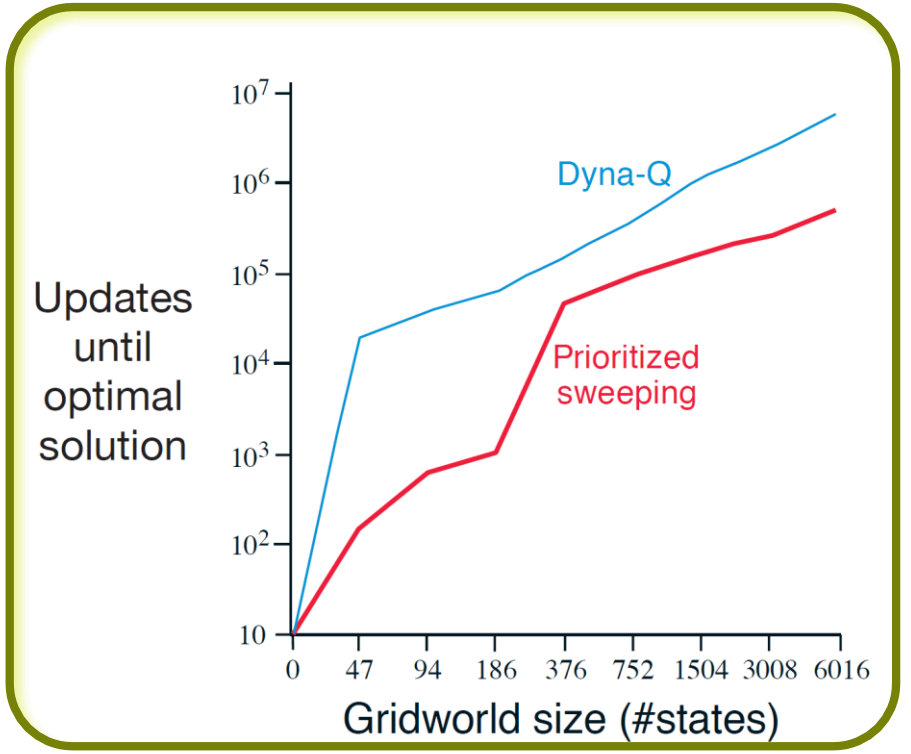
5
  - Loop for all $\bar{S}, \bar{A}$ predicted to lead to S:
    - $\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
    - $P \leftarrow |\bar{R} + \gamma \max_a Q(S,a) - Q(\bar{S},\bar{A})|$.
    - if $P > \theta$ then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority P

1. **State/Action Selection**
2. **Model Learning**
3. **Prioritizing pairs** (priority queue formation)
4. **Planning update** (sweeping)
5. **Update Queue** (check predecessors)

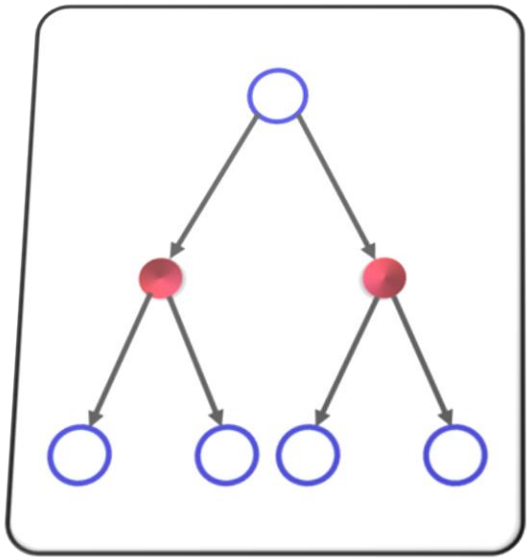
Prioritized Sweeping

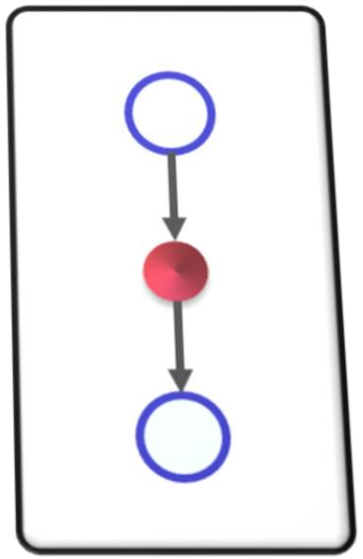
Example: Prioritized Sweeping on Mazes

- **Observations:**
  - Prioritized showed superiority over Dyna-Q
    - Often by a factor of 5 to 10.
    - Prioritized sweeping maintained a decisive advantage over unprioritized Dyna-Q.
  - The global behavior of Prioritized Sweeping
    - when a real-world transition is “surprising” then lots of computation is directed to propagate this new information back to relevant predecessor states
    - When the real-world transition is “boring”, then computation continues in the most deserving part of the space



Expected Updates
vs.
Sample Updates

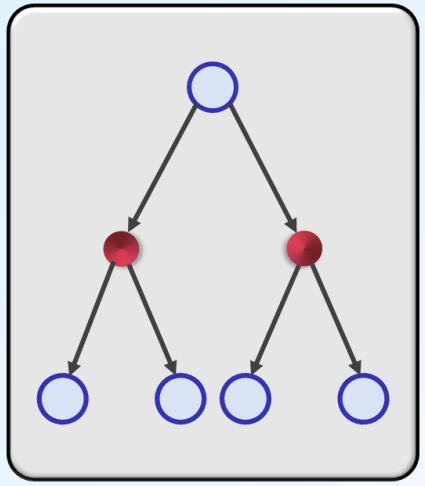




Expected vs. Sample Updates

Expected Updates

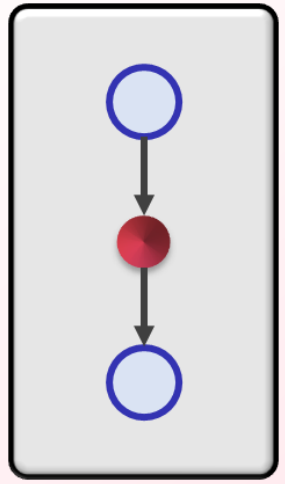
- Estimate the value by considering *all possible* events that might happen



$$V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})]$$

Sample Updates

- Estimate the value by considering a *single* sample of what might happen

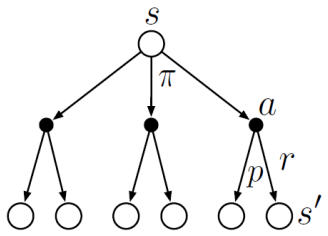
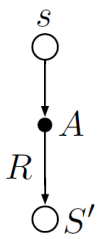
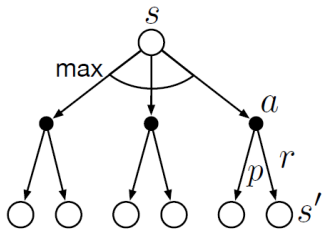


$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

Expected vs. Sample Updates

Comparison

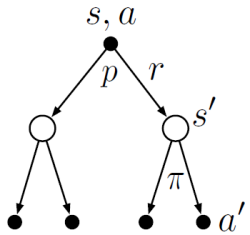
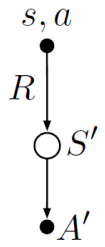
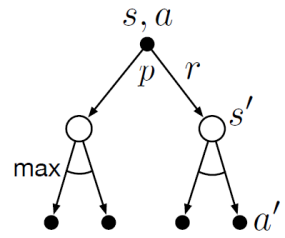
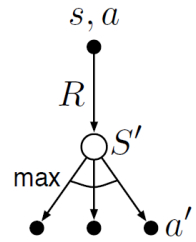
State Values

Value estimated	Expected updates (DP)	Sample updates (one-step TD)
$v_\pi(s)$	 policy evaluation	 TD(0)
$v_*(s)$	 value iteration	

State Values

Action Values

Action Values

Value estimated	Expected updates (DP)	Sample updates (one-step TD)
$q_\pi(s,a)$	 q-policy evaluation	 Sarsa
$q_*(s,a)$	 q-value iteration	 Q-learning

Expected vs. Sample Updates

Expected Updates

- **In the absence of a distribution model**
  - expected updates are not possible
- **Updates yield a better estimate**
  - No sampling error
  - require more computation

Sample Updates

- **In the absence of a distribution model**
  - updates can be done using sample transitions
    - from the environment or a model
- **Updated are corrupted**
  - Because of sampling error

- **If only one next state is possible, then the expected and sample updates given above are identical**
- **If there are many possible next states, then there may be significant differences.**

Expected vs. Sample Updates

Stochastic Environment

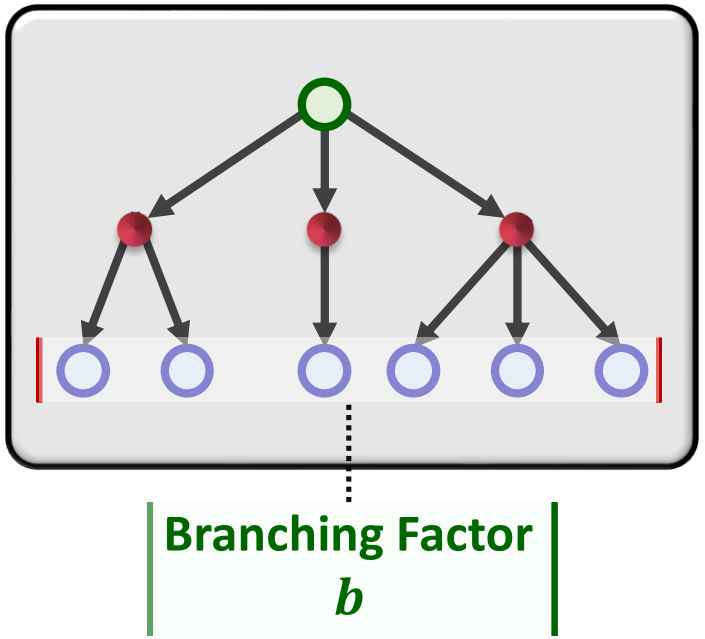
Expected Updates

Exact computation

- Correctness is limited only by the correctness of the at successor

$$Q(s,a) \leftarrow Q(s',a')$$

Stochastic Environment

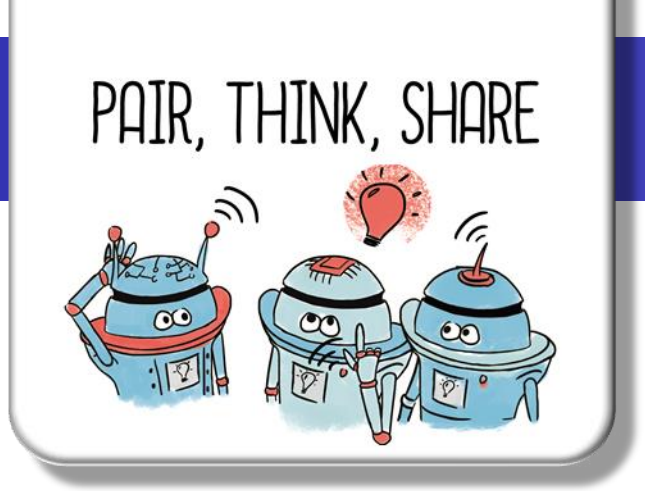


Sample Updates

Computationally cheap

- only one next state, not all possible next states.

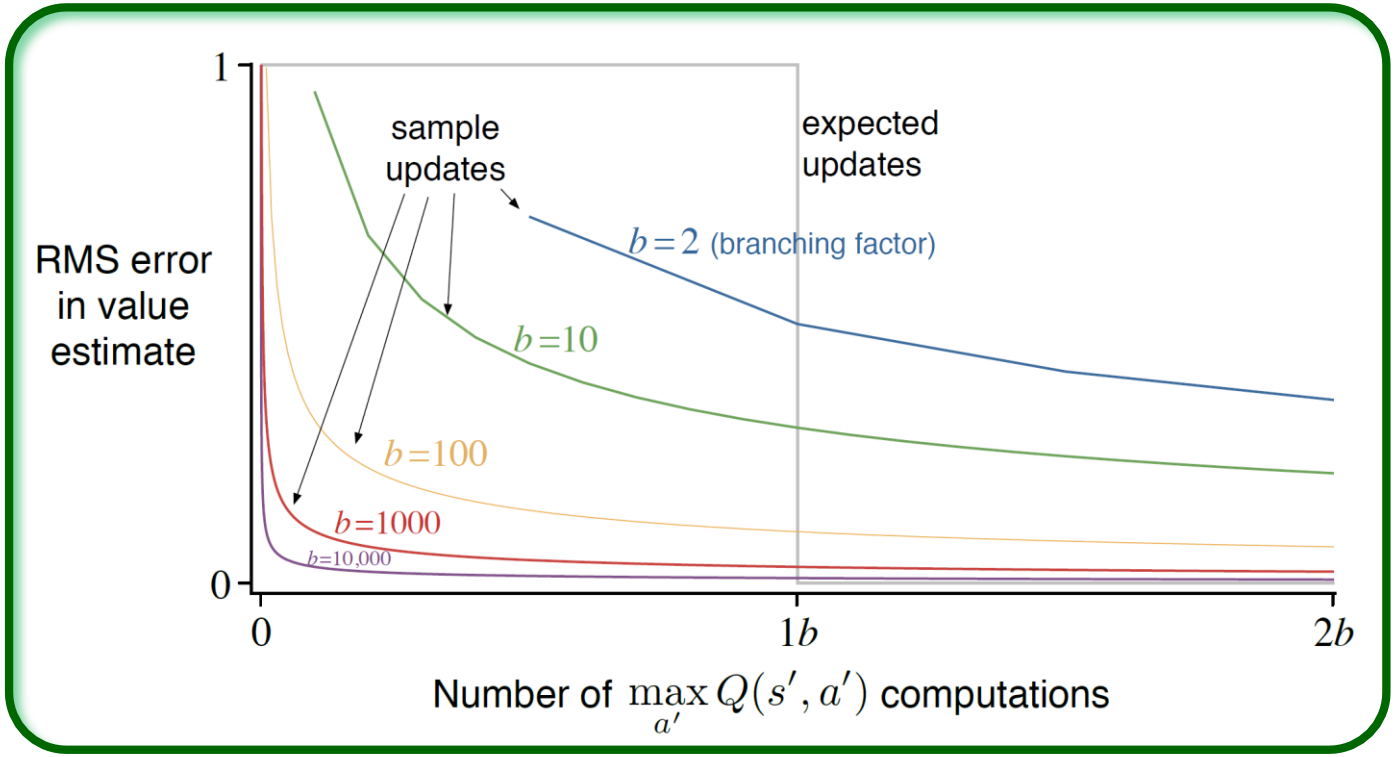
- **An expected update of this pair requires roughly b times as much computation as a sample update.**



Expected vs. Sample updates

Figure shows a comparison of efficiency of expected and sample updates.

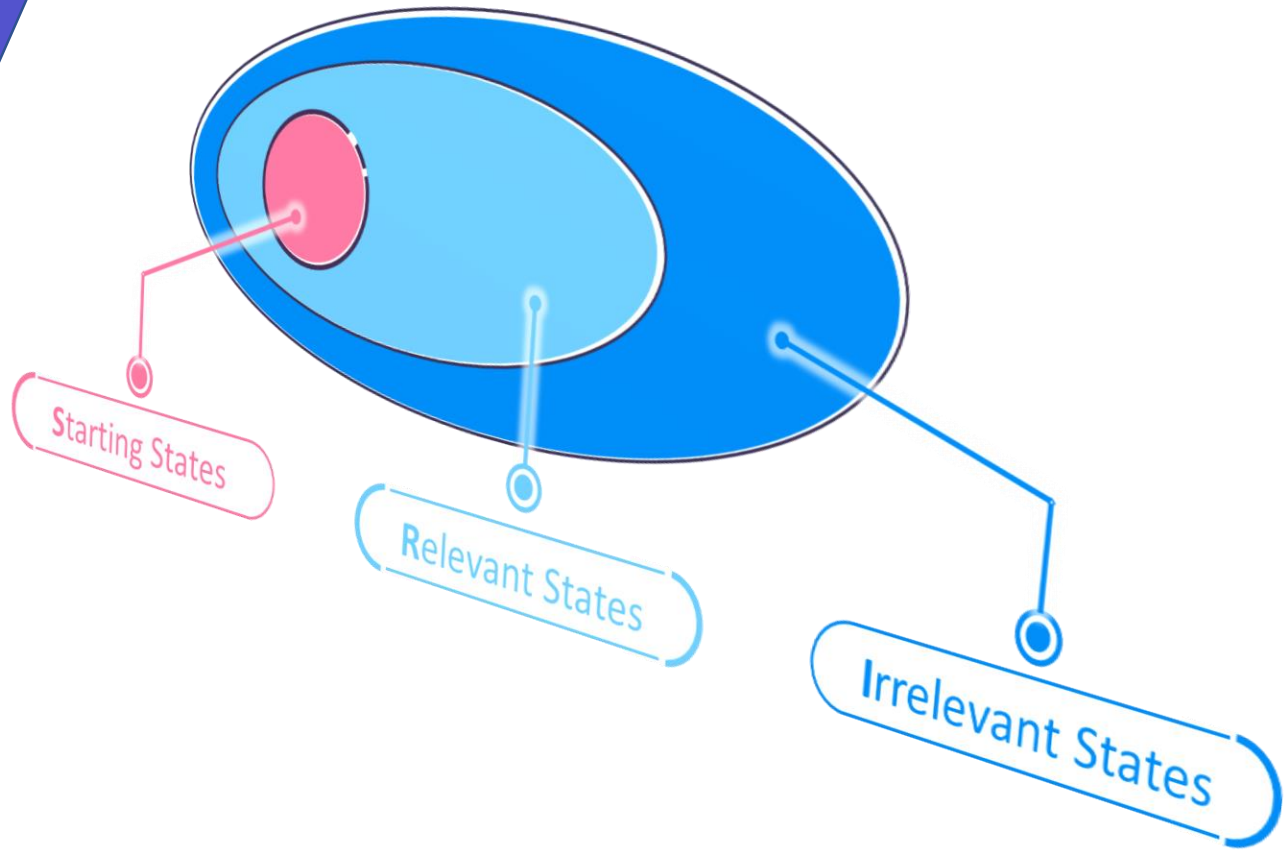
- All b successor **states are equally likely**



1. **Which approach preforms better?**

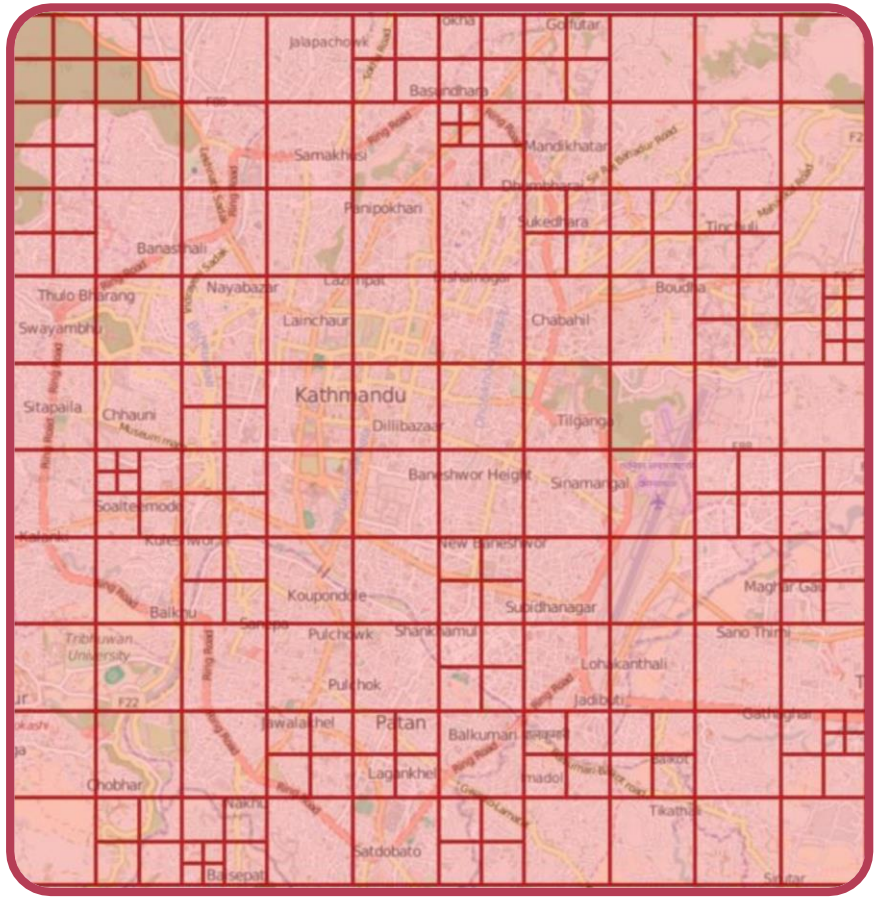
2. **Suppose instead that the distribution was highly skewed, that some of the b states were much more likely to occur than most. Would this strengthen or weaken the case for sample updates over expected updates?**

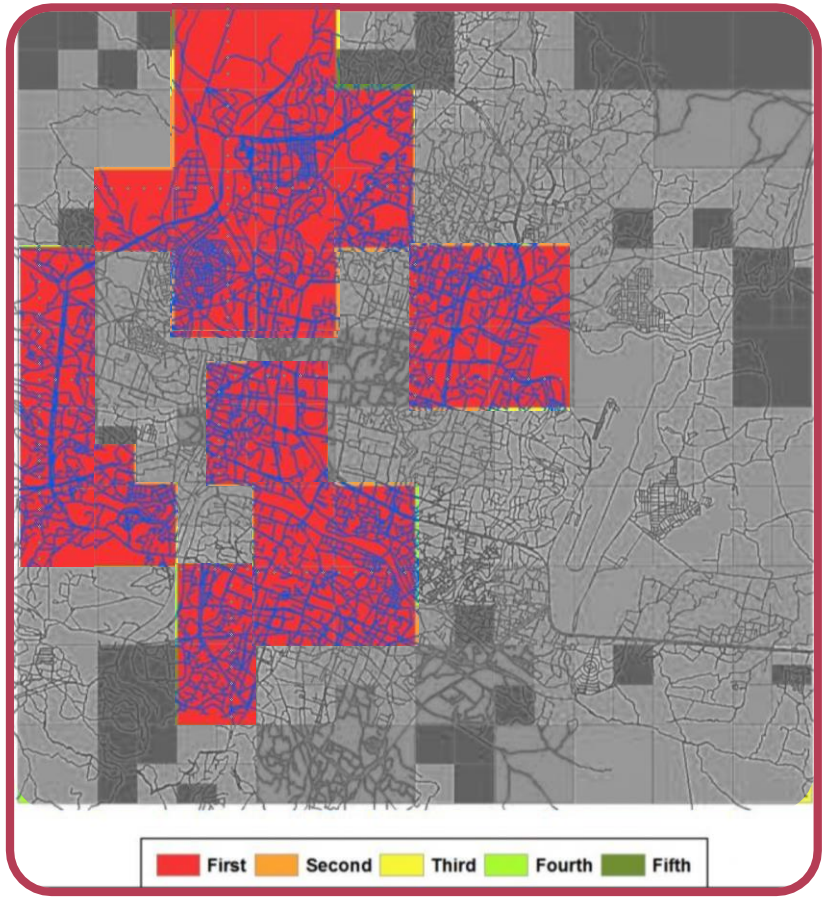
Trajectory Sampling



Trajectory Sampling

Example: Drone: Disaster response





Trajectory Sampling

- **Another interesting way to focus search on relevant regions of the problem space is "trajectory sampling"**

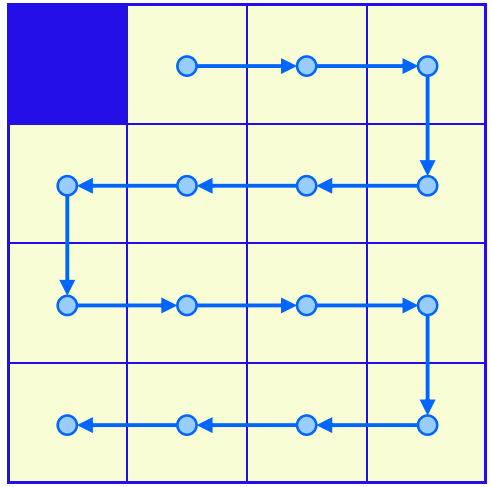
  - **Advantages:**
    - Ignores uninteresting parts of the space.

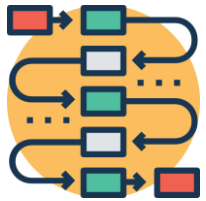
  - **Disadvantage:**
    - When the evaluations and policy become accurate, the same old parts of the space are backed up over and over without improvement of evaluations and policy.

Trajectory Sampling

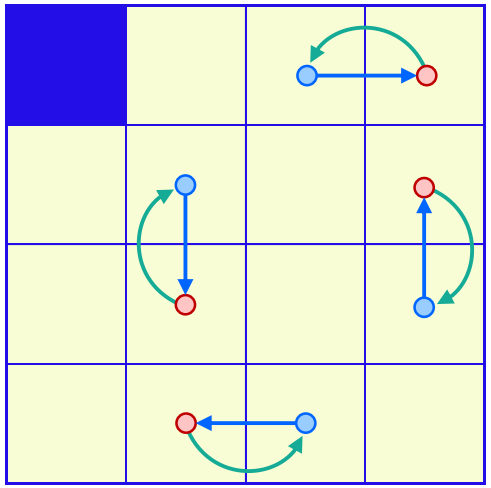
Exhaustive sweeps vs Trajectory sampling

Exhaustive Sweeps





Trajectory sampling



- **sweeps through the entire state (or state-action) space, updating each state (or state-action pair) once per sweep.**
  - Problematic on large task
  - Majority of the states are irrelevant

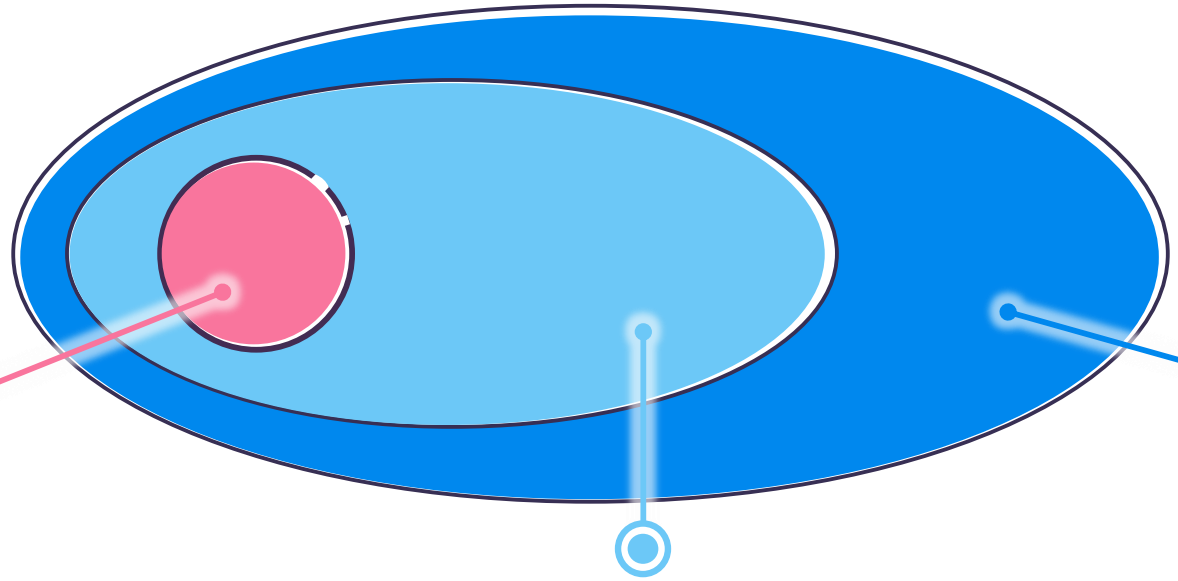
Dynamic Programming ✓

- **sample from the state or state–action space according to some *distribution***

Dyna-Q ✓

Trajectory Sampling

Real-time Dynamic Programming



Starting States

Relevant States

- reachable from some start state under some optimal policy

Irrelevant States

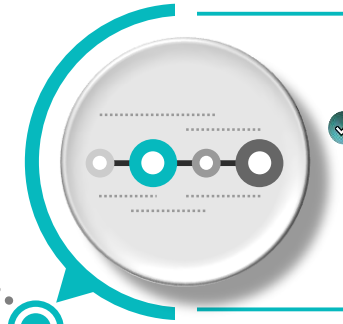
- unreachable from any start state under any optimal policy

- **i.e. Playing Chess:**
  - study positions that might arise in real games

Trajectory Sampling

Uniform vs On-Policy

Trajectory Sampling

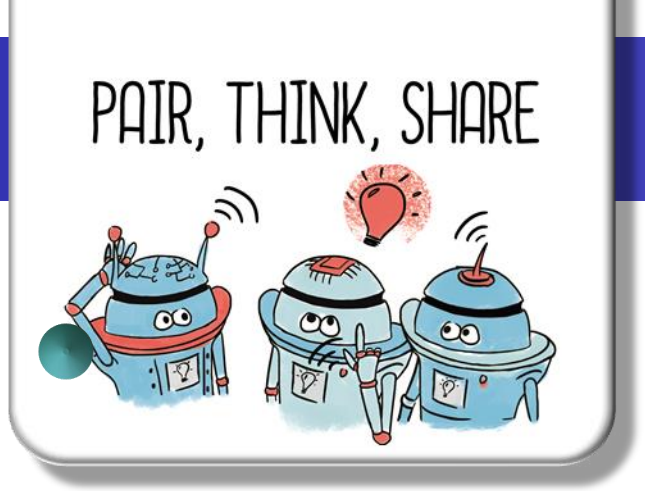


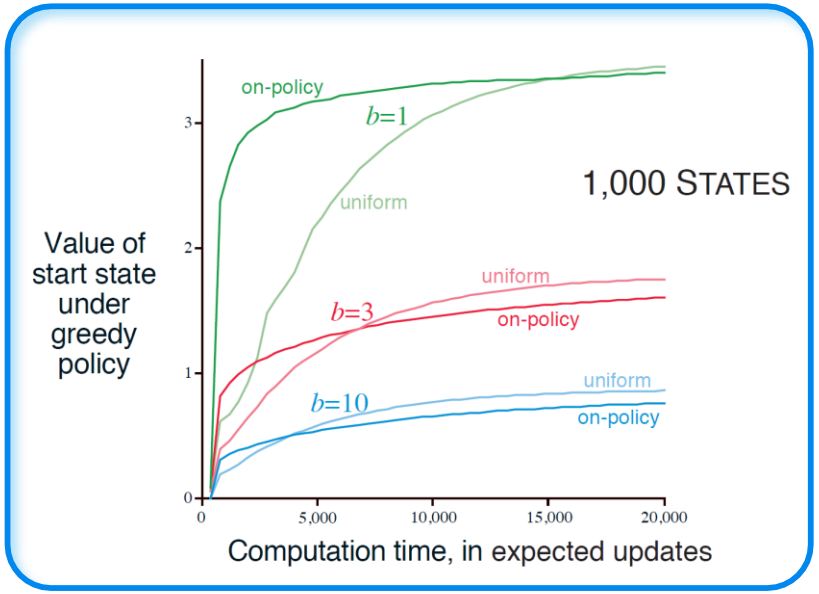
Uniform

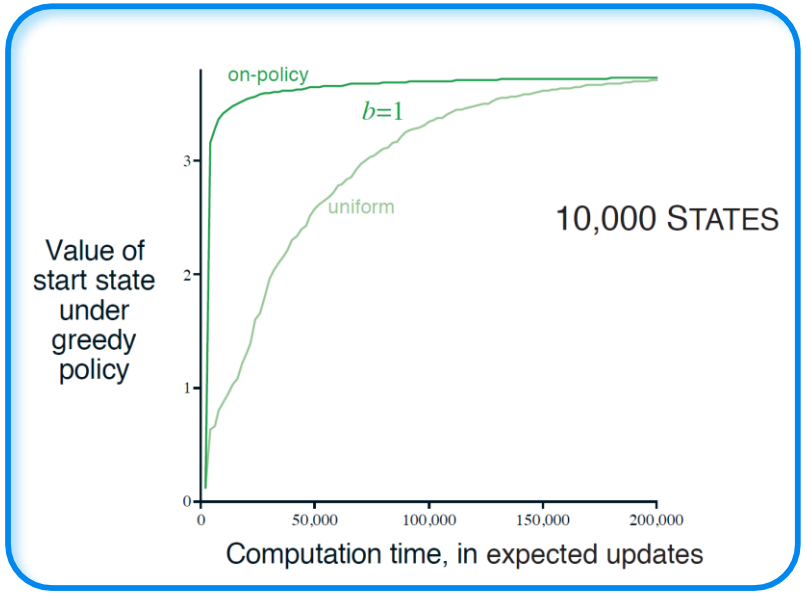
- **Sample states (state-action pairs) uniformly to update**
  - as in the Dyna-Q agent

On-Policy

- **Distribute updates according to the on-policy dist.**
  - sample state transitions and rewards are given by the model
  - sample actions are given by the current policy







Expected vs. Sample updates

Figure shows a comparison of ***uniform*** vs ***on-policy*** trajectory sampling

- All b successor states are equally likely

Which approach preforms better?

Planning at Decision Time

Decision Tree Planning

Example: Alpha-Go

Summary

- **Dyna unifies planning, learning, and acting**
  - Dyna-Q will take the Q-learning transition and perform a model learning step with it.
  - We introduced **model learning** and **search control**

- **Emphasized close relationship between planning and learning**

- **Important distinction between distribution models and sample models Looked at some ways to integrate planning and learning**
  - Synergy among planning, acting, model learning

- **Distribution of backups: focus of the computation**
  - trajectory sampling: backup along trajectories
  - prioritized sweeping
  - heuristic search

- **Size of backups: full vs. sample; deep vs. shallow**

References

- [1] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.
- [2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” Journal of artificial intelligence research, vol. 4, pp. 237–285, 1996.
- [3] Y. Hu, K. Janowicz, and H. Couclelis, “Prioritizing disaster mapping tasks for online volunteers based on information value theory,” Geographical Analysis, vol. 49, no. 2, pp. 175–198, 2017.
- [4] G. Baldassarre, “Planning with neural networks and reinforcement learning,” PhD Thesis, University of Essex, 2001.