Reinforcement Learning: Model Based

Earl Wong

Model Based

- In model free approaches, the agent learns a policy by interacting with the environment.
- Because the agent had no knowledge about the environment, the agent requires many “trial and error” interactions, in order to learn a good policy.
- Suppose the agent did have a model of the environment.
- Now, the agent can plan future actions by using the information contained in the model.

Model Based

- In many situations, the model is known a-priori.
- This is often the case with games.
- When a model is known, the model can provide a sample efficient method to determine a policy and / or value function.
- Recall: Once a value function V(s) is known, the action “a” that takes us from state s to s’ with the largest V(s’), becomes the policy.

Model Based

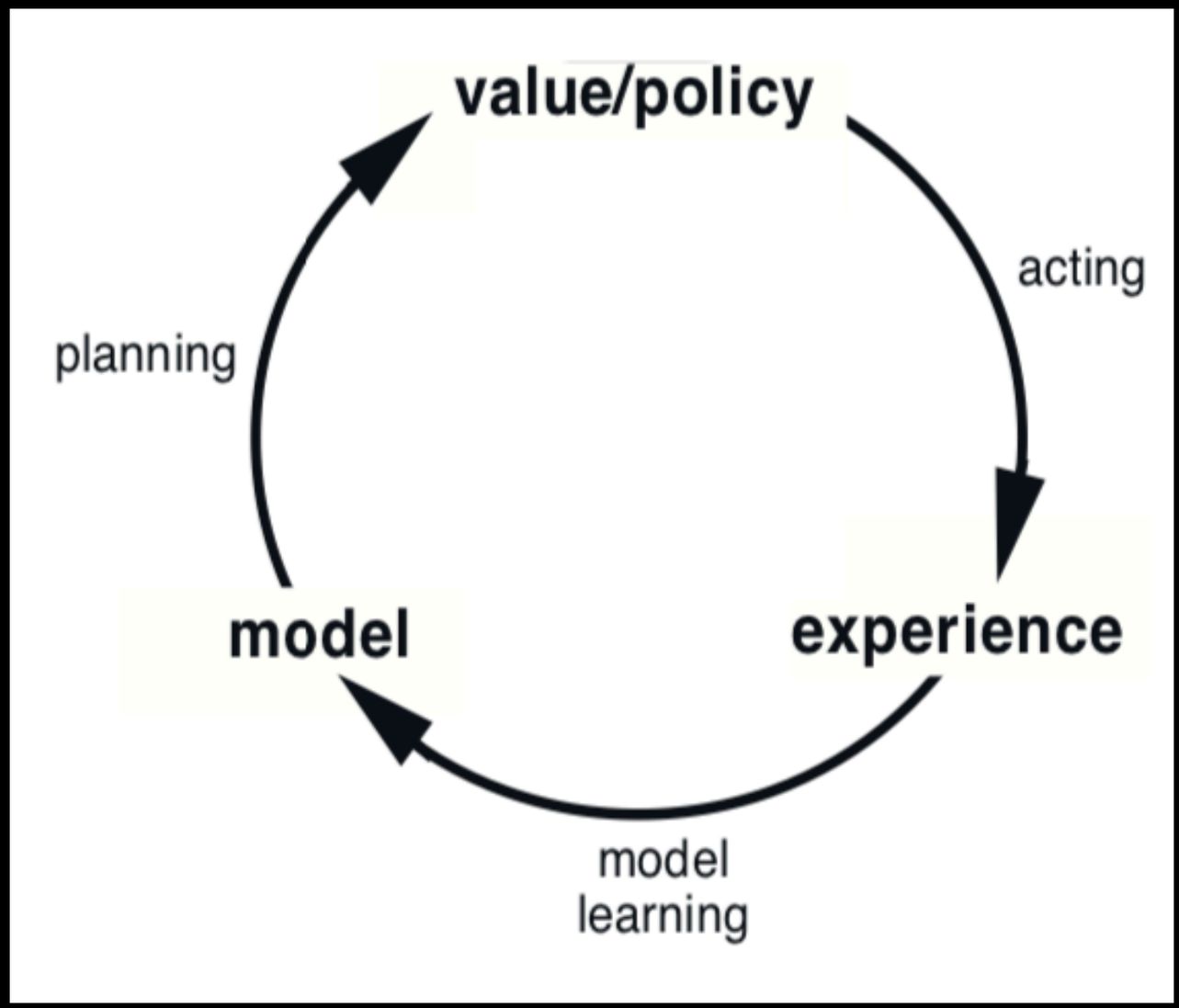
- Even when the model is unknown, a model can (still) be learned by interacting with the environment.

- However, the learned model may or may not be accurate.

- If the learned model is inaccurate, the learned policy will be bad and / or the computed value function will be inaccurate.

- In addition, by introducing a learned model into the process, a second source of (potential) error is introduced into the policy and / or computed value function.

Model Based



The following figure was taken from the Sutton and Barto text, and illustrates the process of model learning in policy planning.

What needs to be learned, if we are not given the model?

Recall the Markov Decision Process

$$(S, A, R, P, \gamma)$$

where R and P were given by:

$$R = E\left[R_t \,\middle|\, S_t = s, A_t = a\right]$$

$$P = \Pr\left[S_{t+1} = s' \,\middle|\, S_t = s, A_t = a\right]$$

Since R and P are now unknown, they need to be learned / approximated.

$$\hat{R} \approx R \qquad \hat{P} \approx P$$

Learning R and P

Model learning is accomplished by performing repeated interactions with the environment, producing episodic rollouts:

$$\{S_0, A_0, R_0, S_1, A_1, R_1, \ldots S_T\}$$

These results are then aggregated into "frequentist' statistics or used to 2) fit a regression function (to the rewards) and a density function (to the transition densities).

For the rewards R, we have

$$S_0, A_0 \rightarrow R_0$$

$$S_1, A_1 \rightarrow R_1$$

⋮

For the transition densities P, we have

$$S_0, A_0 \rightarrow S_1$$

$$S_1, A_1 \rightarrow S_2$$

⋮

These input-output pairs allow us to compute accurate estimates for R and P.

Summary

In general, an agent can learn from both real and simulated experiences.

- Real Experiences - agent learns from direct interaction with the environment
- Simulated Experiences - agent learns from indirect interaction with the environment (using a model with learned R and P parameters)
- These experiences result in model free, model based and hybrid RL algorithms.

Summary

Model Free Approach:

Learn a policy and / or value function from real experiences.

Model Based Approach:

Learn a model.

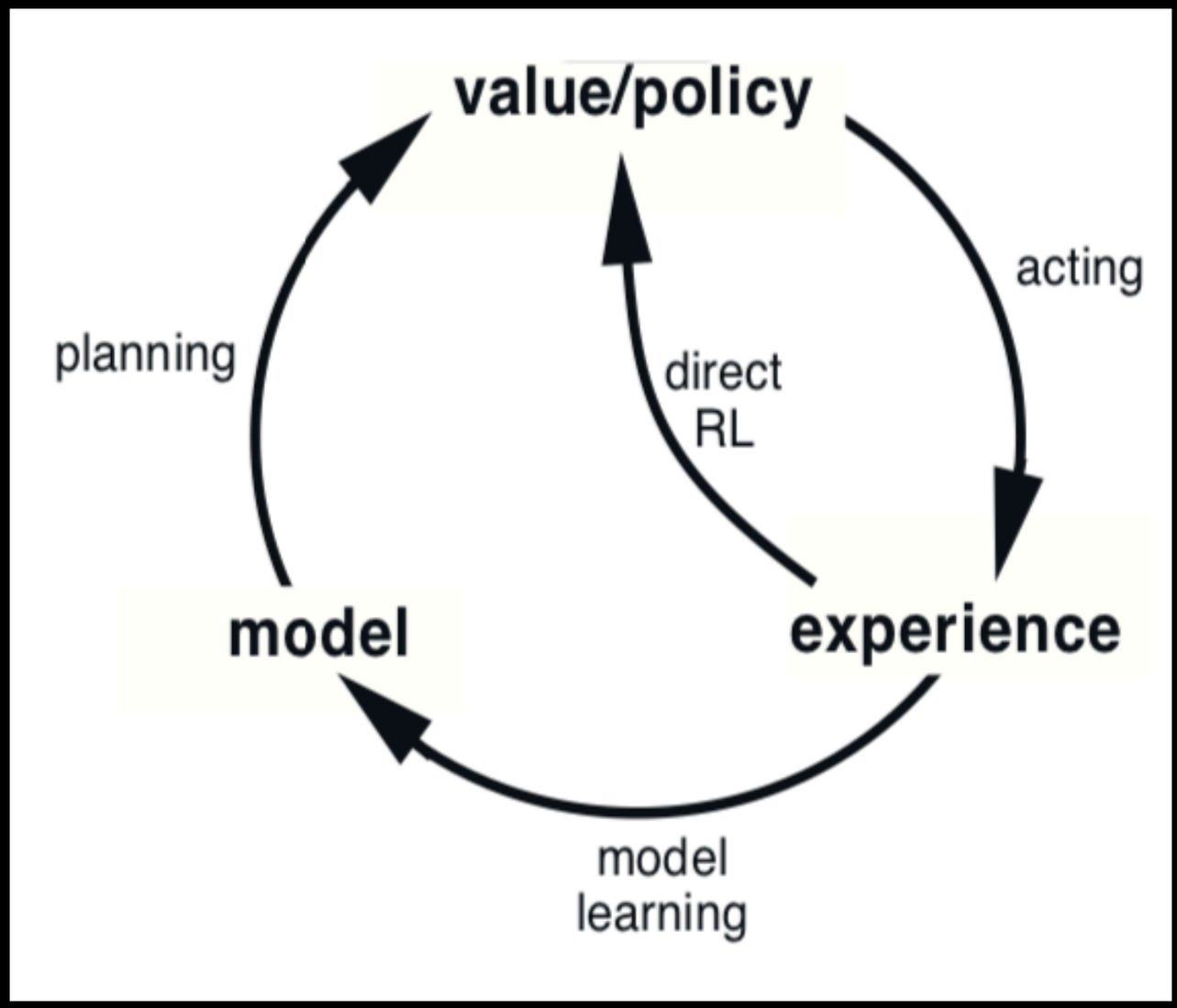
Plan a policy and / or value function from simulated experiences.

Hybrid Approach (Dyna):

Learn a model.

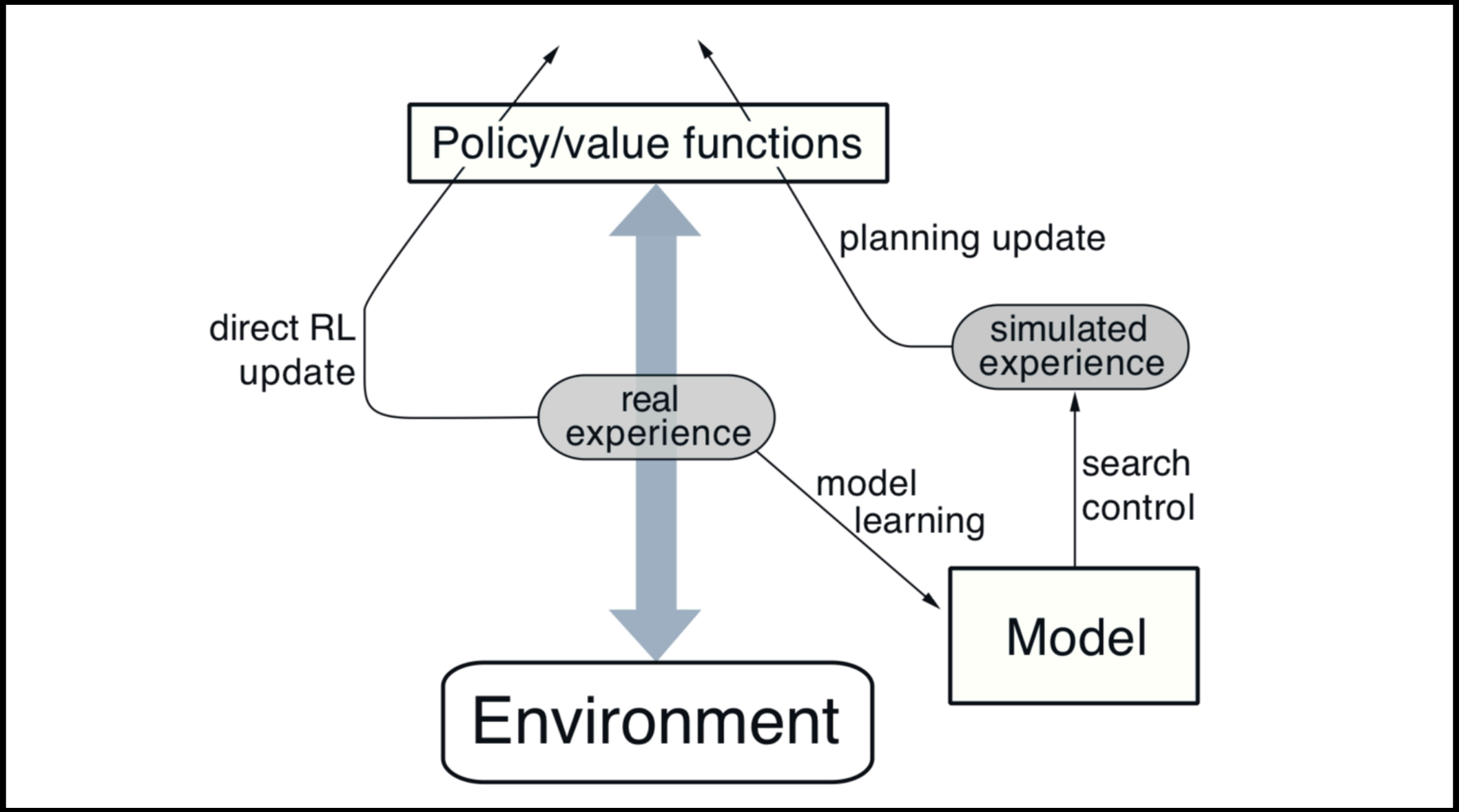
Plan a policy and / or value function from BOTH real and simulated experiences.

Hybrid Based (Dyna)



The following figure was taken from the Sutton and Barto text, and illustrates the policy learning cycle using both real (direct RL) and simulated (planning) experiences.

Hybrid Based (Dyna)



This figure was also taken from the Sutton and Barto text, and illustrates a "real experience centric" view. Real experience is used to create a policy, as well as a model. The model assists in the policy planning.

Dyna Algorithm

Initialize Q(S, A) and Model(S, A) for all s and a possibilities

Loop (non model based) forever:

$$S \leftarrow current\ \ s$$

$$A \leftarrow \epsilon - greedy\, Q(S, a)$$

$$R,\ S' \leftarrow Take\ action\ A$$

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

Non Model Based

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

Loop (model based) N times:

$$S \leftarrow random\ previously\ observed\ state$$

$$A \leftarrow random\ action\ previously\ taken\ in\ S$$

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

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

Model Based

Model Based Search

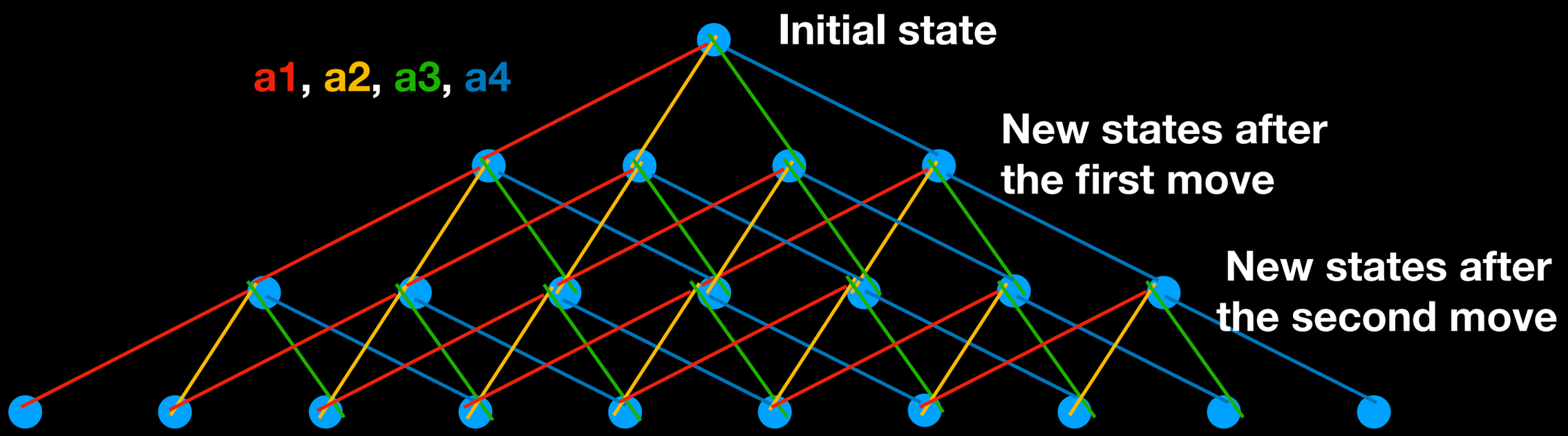
- By having an accurate model, the outcomes for different actions can be evaluated.
- The best action (for a given state), can then be chosen.
- This is the objective of model based search.
- In general, the model can be used to perform Monte Carlo or Temporal Difference based searches.
- In the following slides, the focus will be placed on tree searches.

Model Based MC Tree Search

Given learned $\hat{R} \approx R$ $\hat{P} \approx P$

we obtain the following MDP: $(S,\ A,\ \hat{R},\ \hat{P},\ \gamma)$

Suppose we played a board game using this model. Here, an entire tree for the game can be created, where each node represents a different board state. In the example below, we restrict ourselves to 4 possible actions, for every board state.

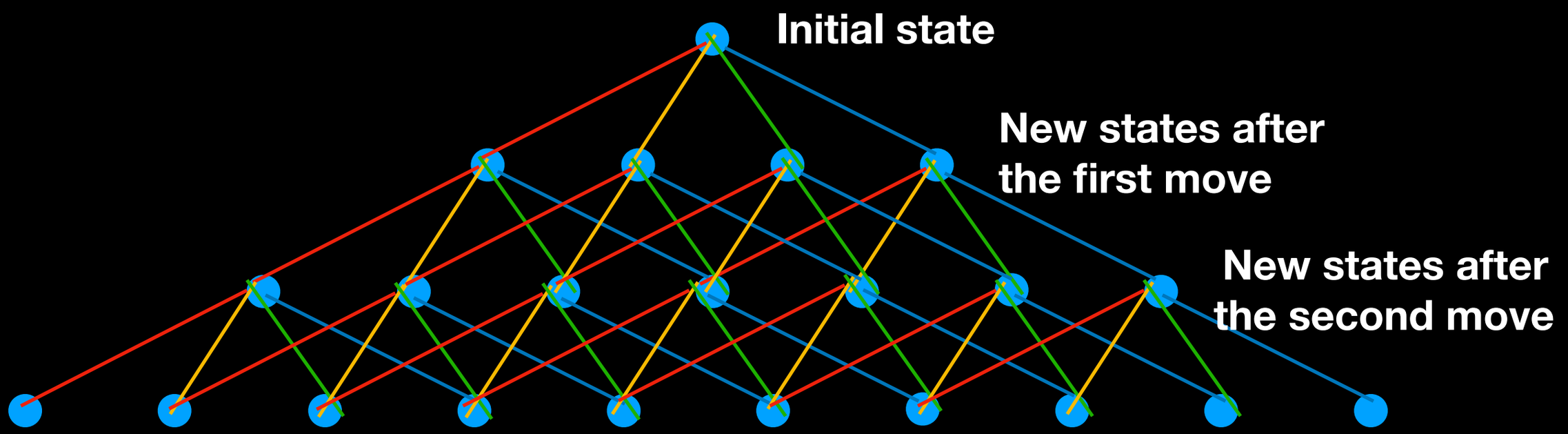


Model Based MC Tree Search

In addition, we can also compute value functions Q(s,a) associated with the different {state, action} pairs: Q(s0, a1), Q(s0, a2), Q(s0, a3), Q(s0, a4), Q(s1, a0), ….

In theory, a complete tree “build out” could occur, using the MDP model.

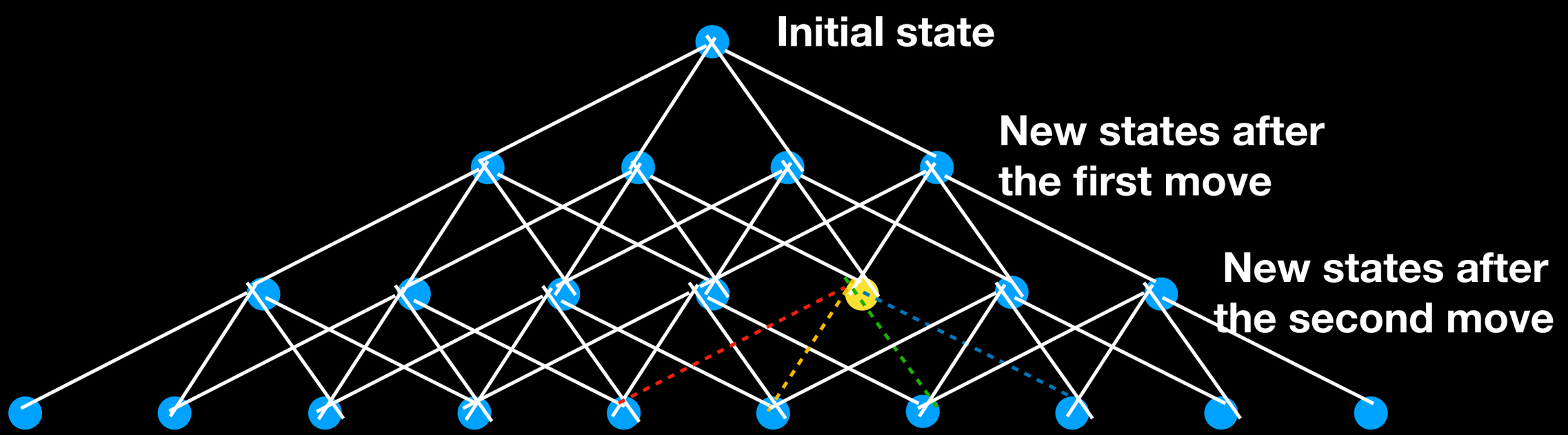
However, this tree “build out” would be very costly and time consuming.



Model Based MC Tree Search

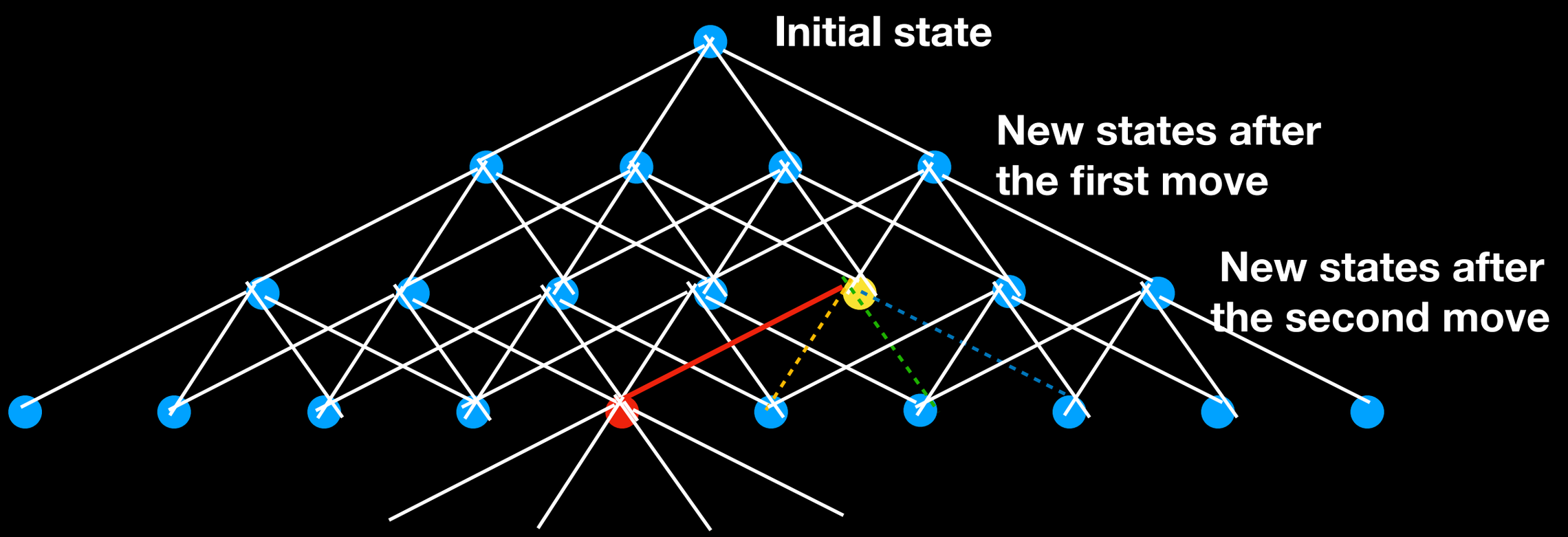
Since our main concern is to determine the optimal “next move / next action”, we only need to build out the tree from our current state.

In the tree below, the yellow disk represents the current state that we wish to “build out” from.



Model Based MC Tree Search

We start with action a1(red path), and determine the average return using our model and a fixed policy.



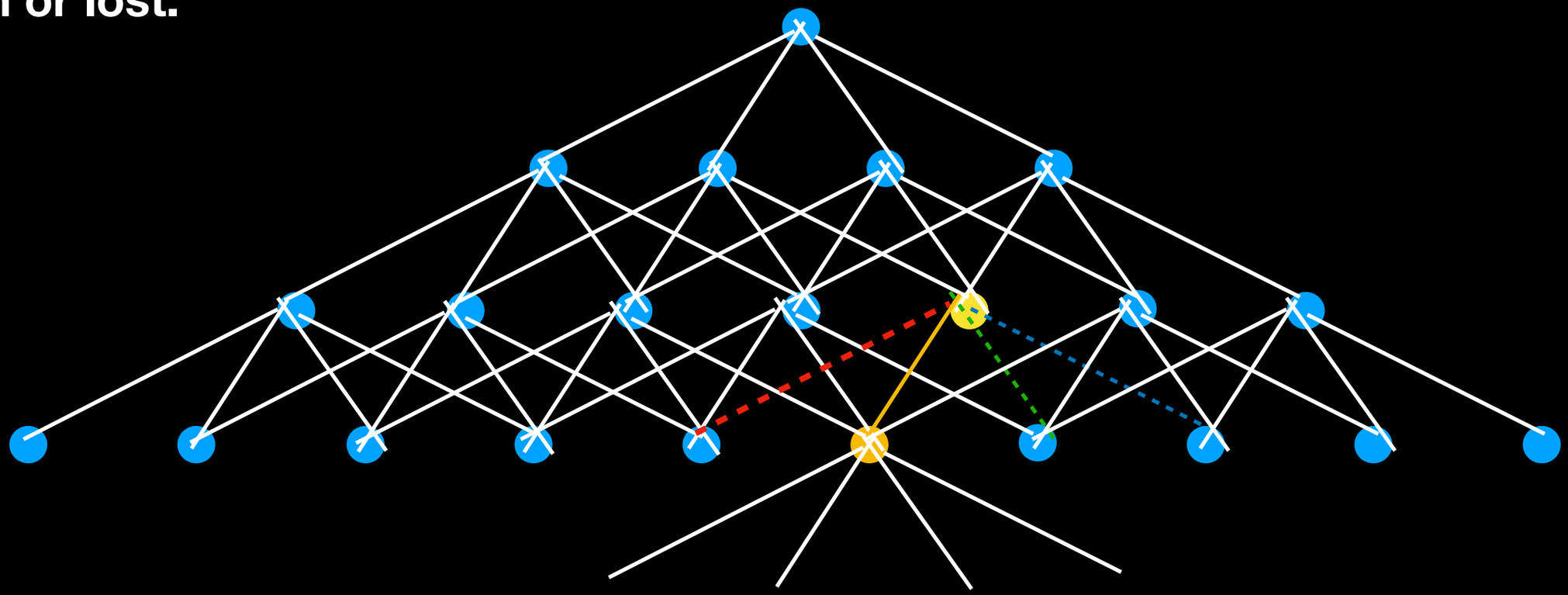
Model Based MC Tree Search

Next, we choose action a2 (peach path), and determine the average return using our model and a fixed policy, etc.

Based on the computations for all four actions (Q(si, a0), Q(si, a1), Q(si, a2), Q(si, a3)), we can then select the best action for the board state, si.

We take this action, and repeat the process for the next state, s'.

We continue the process, until a terminal state is reached, where the game is won or lost.



Forward Look Ahead

- Instead of solving the entire MDP, we have solved a sub MDP, starting from a given board state, which we will refer to as NOW.
- We did this by computing the average returns associated with the different actions associated with the board state NOW.
- The average returns were computed using our MDP model.
- The action resulting in the largest average return from board state NOW was then selected.
- This process is called sample based planning.

Forward Look Ahead - Tree and Default Policy

- Under the previous approach, the sampling information from the intermediate board states was never re-used.

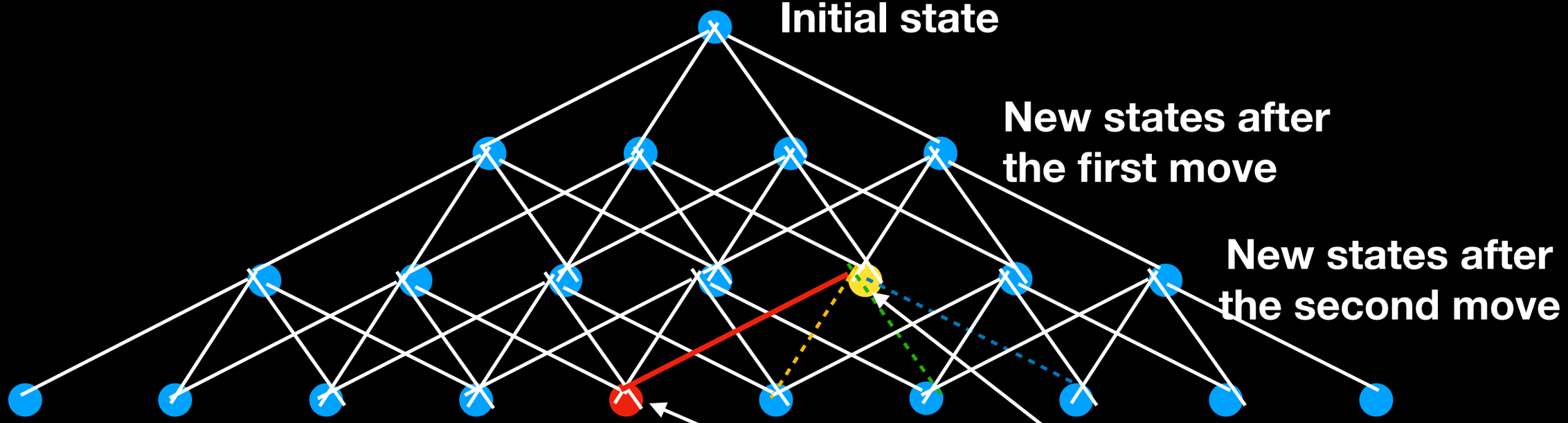
- Ideally, we would like to store / re-use this information for improved action value function results.

- How do we do this?

- If a board state is encountered for the first time, the number of visits and total return is (initially) set to 0. No update is performed during backpropagation.

- When the board state is encountered again, the number of visits and total return is then updated during backpropagation.

Example

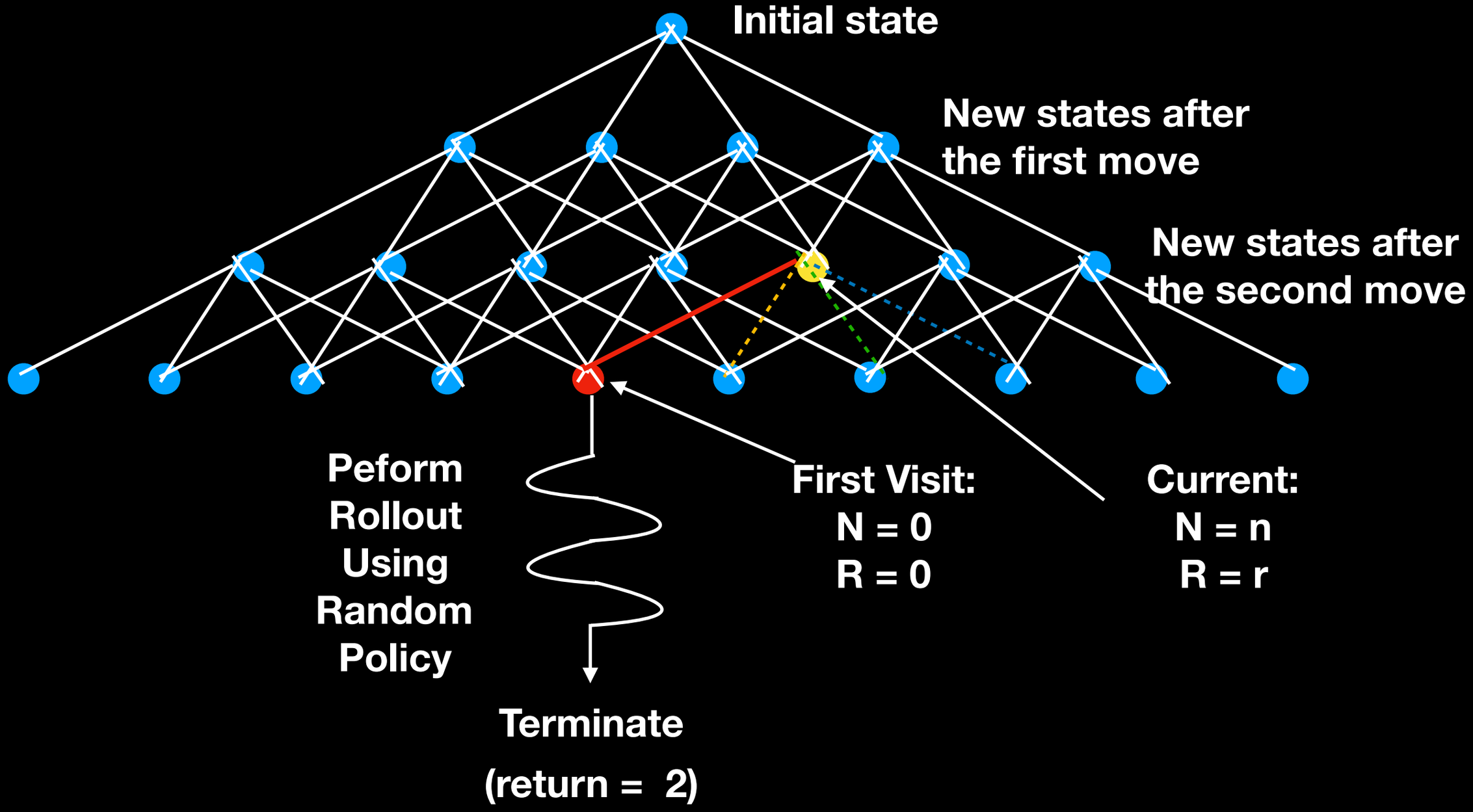


The solid red line indicates the action taken. The current board state is denoted by the yellow disk.

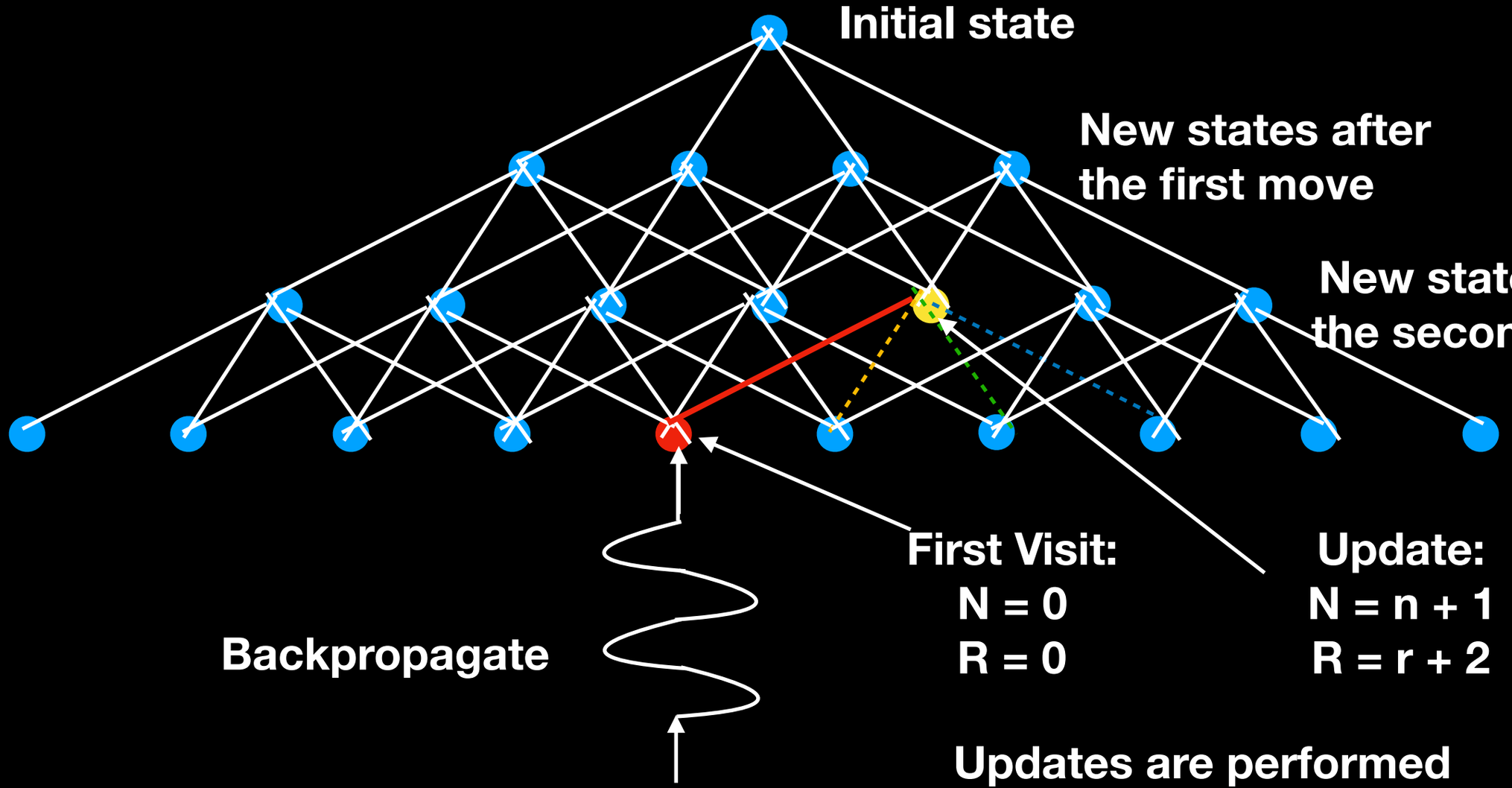
First Visit:
N = 0
R = 0

Current:
N = n
R = r

Example

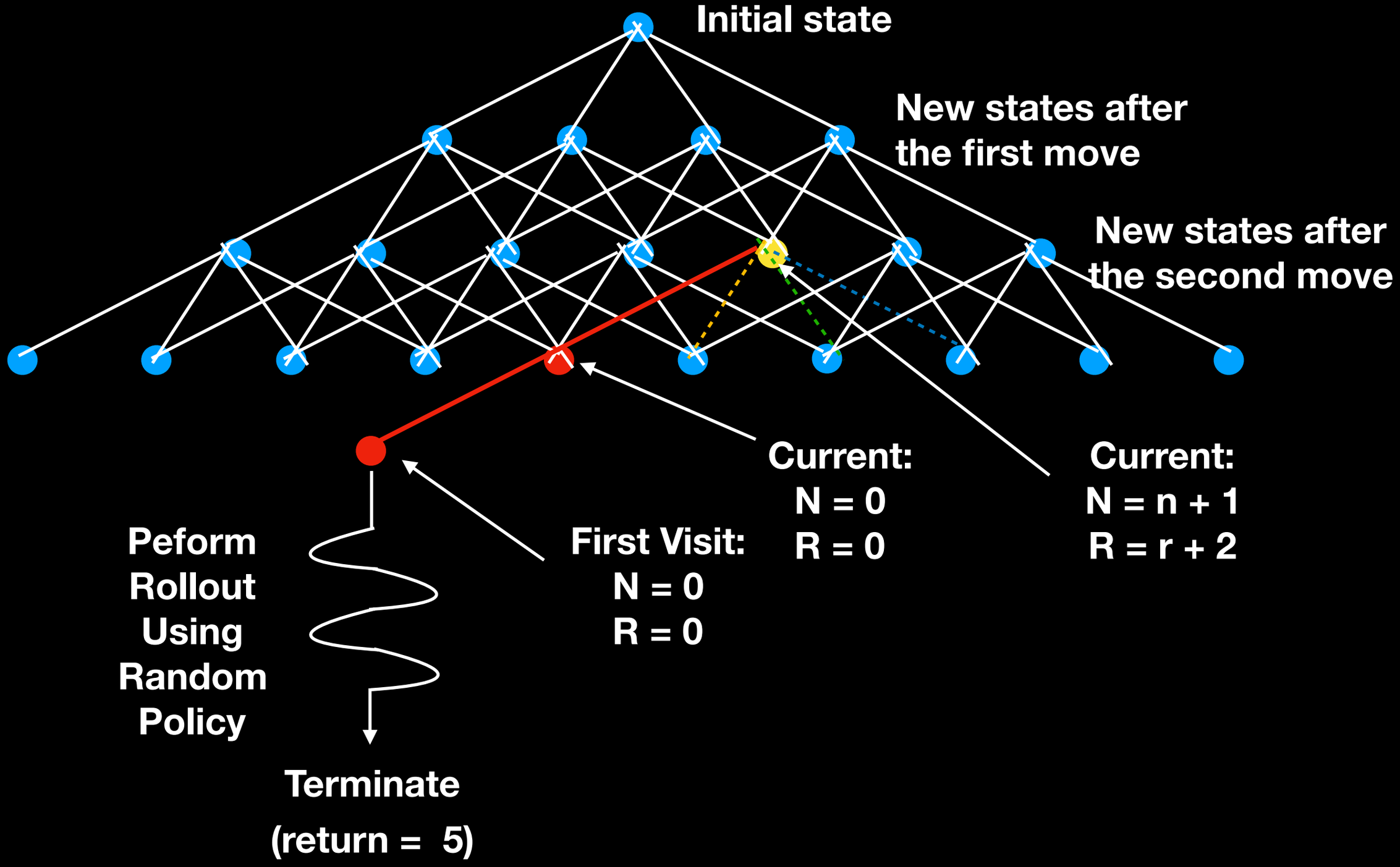

Initial state
New states after the first move
New states after the second move
Peform Rollout Using Random Policy
Terminate
(return = 2)
First Visit:
N = 0
R = 0
Current:
N = n
R = r

Example

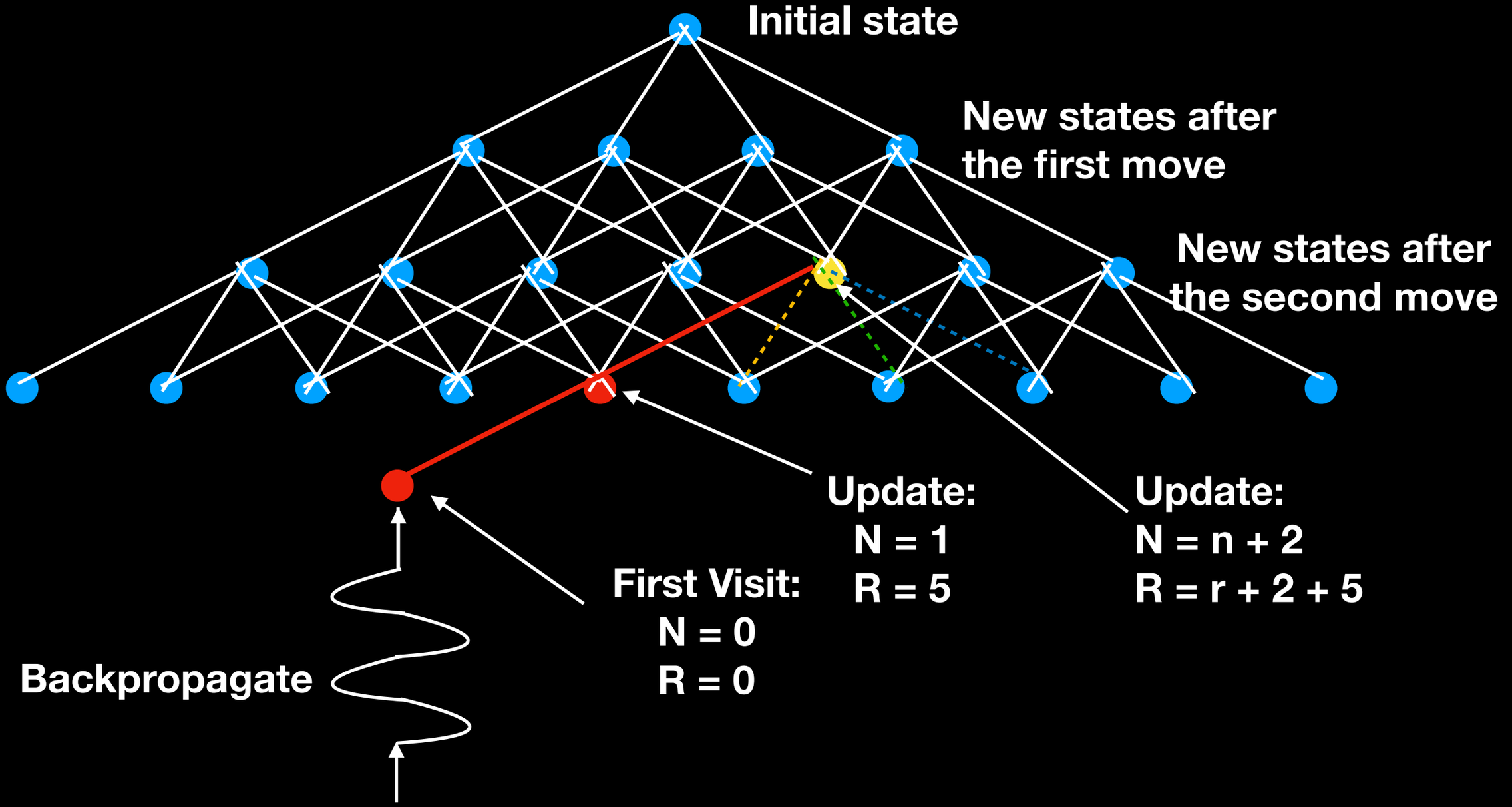


Updates are performed for all “non first visit” board states starting at the yellow disk.

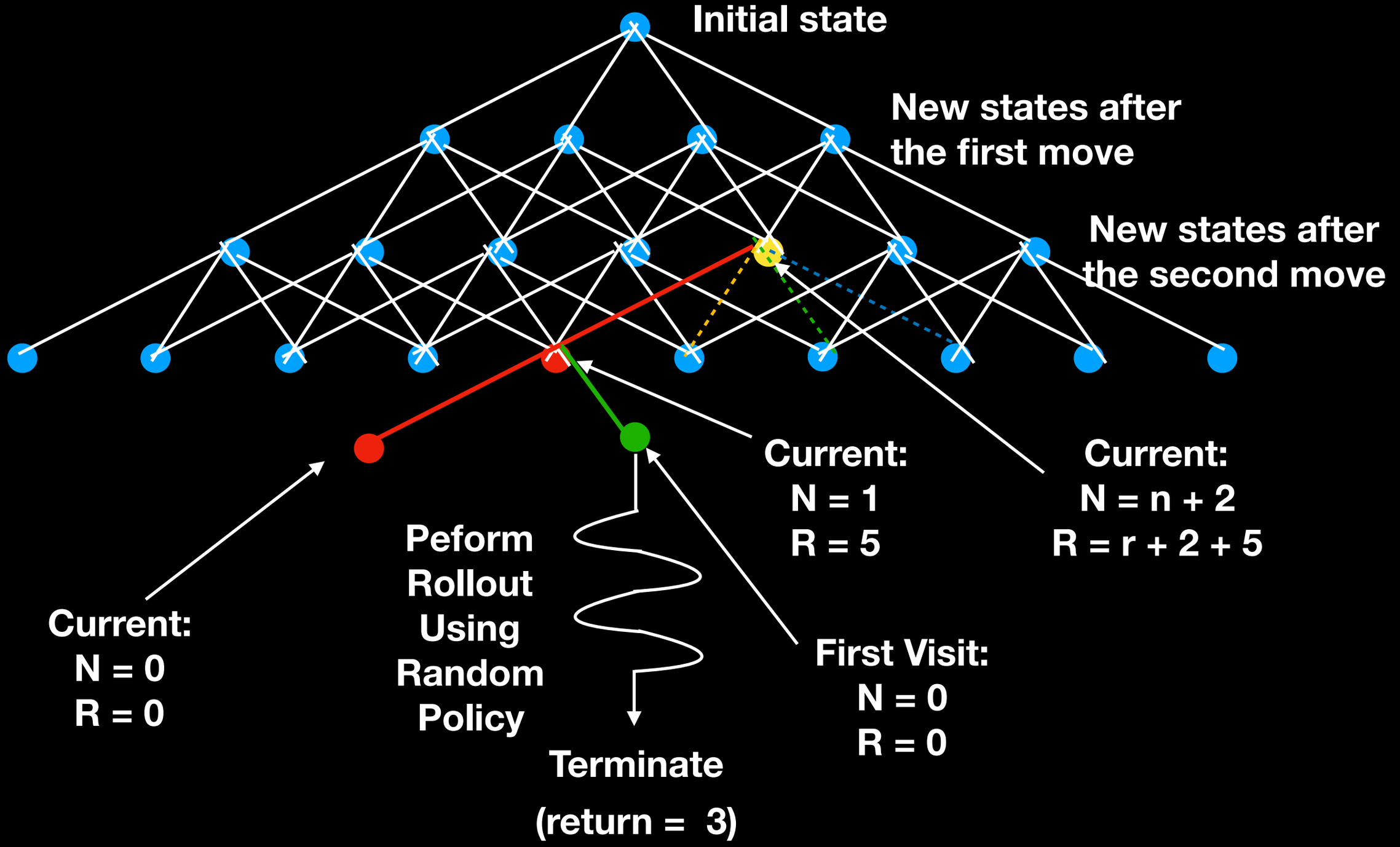
Example


Initial state
New states after the first move
New states after the second move
Current: N = 0 R = 0
Current: N = n + 1 R = r + 2
First Visit: N = 0 R = 0
Peform Rollout Using Random Policy
Terminate (return = 5)

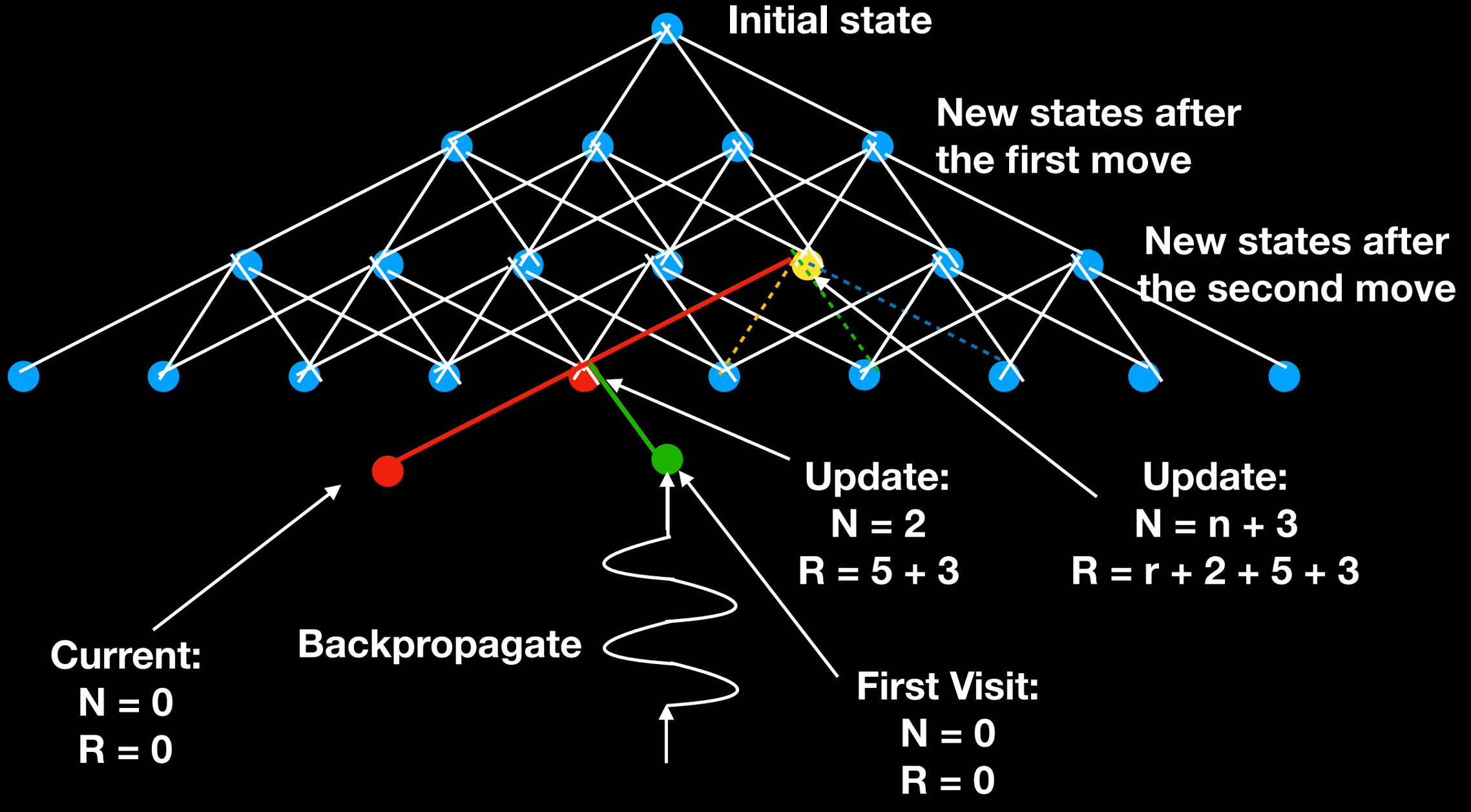
Example


Initial state
New states after
the first move
New states after
the second move
Update:
N = 1
R = 5
Update:
N = n + 2
R = r + 2 + 5
First Visit:
N = 0
R = 0
Backpropagate

Example


Initial state
New states after the first move
New states after the second move
Current:
N = 0
R = 0
Peform Rollout Using Random Policy
Terminate
(return = 3)
Current:
N = 1
R = 5
First Visit:
N = 0
R = 0
Current:
N = n + 2
R = r + 2 + 5

Example


Initial state
New states after the first move
New states after the second move
Update:
N = 2
R = 5 + 3
Update:
N = n + 3
R = r + 2 + 5 + 3
Current:
N = 0
R = 0
Backpropagate
First Visit:
N = 0
R = 0

Example

- By building a tree in this manner, improved Q(s,a) action value functions can be computed, using the information stored in the visited board states.

- In practice, an epsilon greedy strategy is employed to encourage tree exploration, for all previously visited board state nodes. (i.e. Nodes prior to the rollout.)

- In practice, the MC tree search approach builds out the most promising areas of the search tree.

Example

- By inspection, two policies are at work here.
- The tree policy is the policy associated with the action value functions associated with the visited board state nodes.
- The default policy is the policy used in the rollouts associated with first time visited board states. (Typically, this is a random policy.)
- As more and more rollouts occur, the number of board state nodes associated with the tree policy increases, while the number of board state nodes associated with the default policy decreases.

Model Based TD Search

- Using the same forward search and sampling methodology, a TD search can also be performed, in place of the MC search.
- Now, instead of using complete rollouts, a SARSA based temporal update is used.
- Recall: $Q(S,\ A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$
- In practice, TD updates can be very effective in non directed graphs / trees, where “loop backs” (to a previously visited state) can occur.
- Additional benefits of TD search (versus MC search) include lower variance and higer efficiency.

Summary

- Learning from simulations can be an effective way to perform search.
- Learning from simulation is possible, if a known model exists.
- Knowledge extracted from simulations can greatly assist in planning.
- Specifically, sample trajectories can be generated and applied to RL planning algorithms like MC Control, SARSA and Dyna.