Planning and Learning

Chris Amato
Northeastern University

with some slides from Rob Platt, U Alberta, Mykel Kochenderfer, and Frans Oliehoek

Announcements

- Exam
- Project proposals due 10/28
  - More formal (i.e., less English)
- Ex5 (TD) due 10/24
- Ex6 (Planning and learning) due 11/1
- Exam 2 on 11/25
- Read SB 9.1--9.5, 9.8

RL research opportunities

- Undergrad co-op on security for RL
- MS Research apprenticeships
    - Nominations
    - Projects

Overview: Planning and Learning

- We will talk about
  - various planning methods that use a model or simulator to generate solutions (planning)
  - Model-based RL methods that learn models of the dynamics as part of RL (which use planning with the model)

Planning and Learning: AlphaGo

Planning

What do you think of when you think about "planning"?

– often, the word "planning" often means a specific class of algorithm

– here, we use "planning" to mean any computational process that uses a model to create or improve a policy

Planning

<u>What do you think of when you think about "planning"?</u>

– often, the word "planning" often means a specific class of algorithm

– here, we use "planning" to mean any computational process that uses a model to create or improve a policy

Model: "Anything that an agent can use to predict how the environment will respond to its actions." $p(s', r \mid s, a)$

For example: an unusual way to do planning

model $\longrightarrow$ simulated experience $\xrightarrow{\text{backups}}$ values $\longrightarrow$ policy

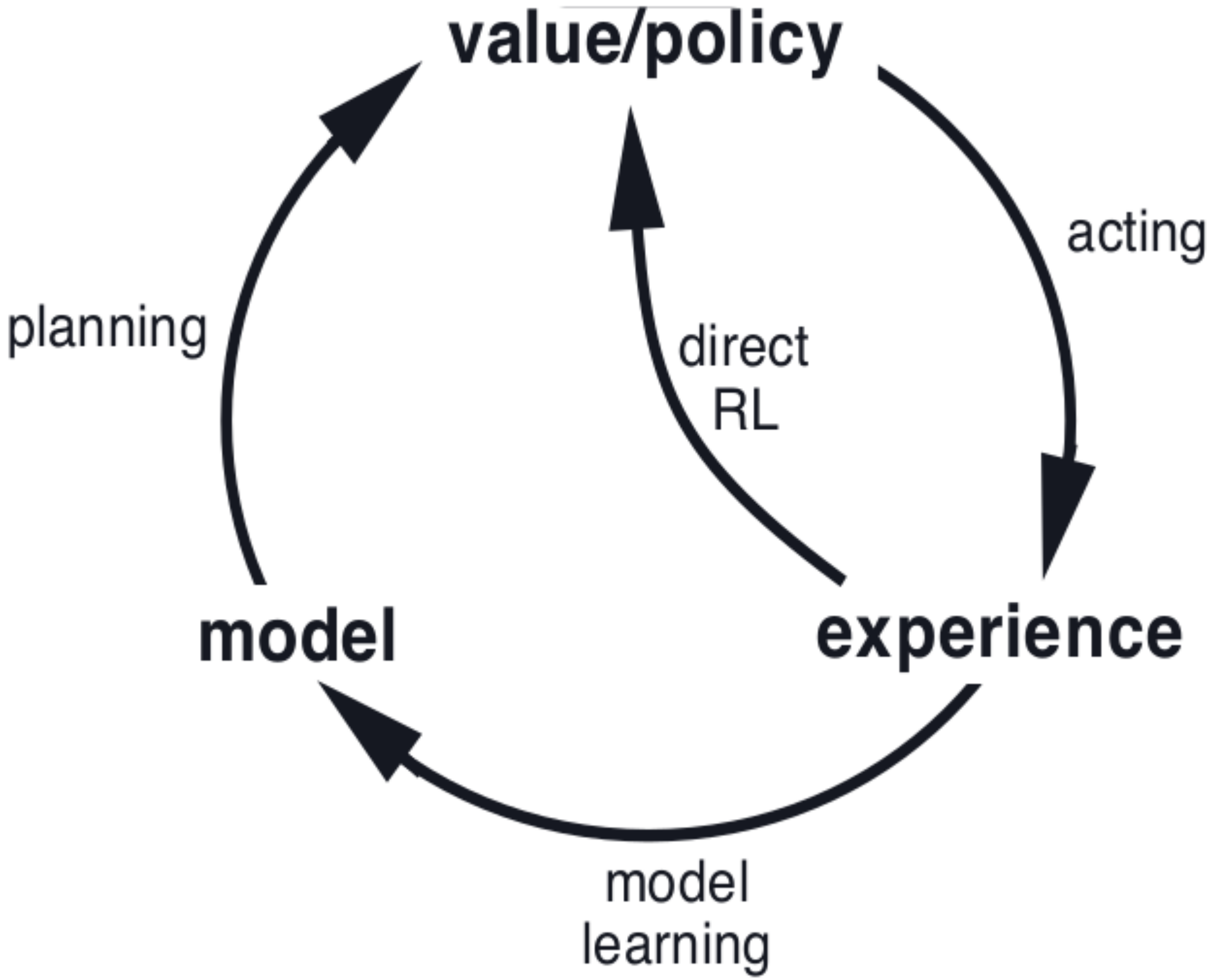
Random-sample one-step tabular Q-planning

Loop forever:

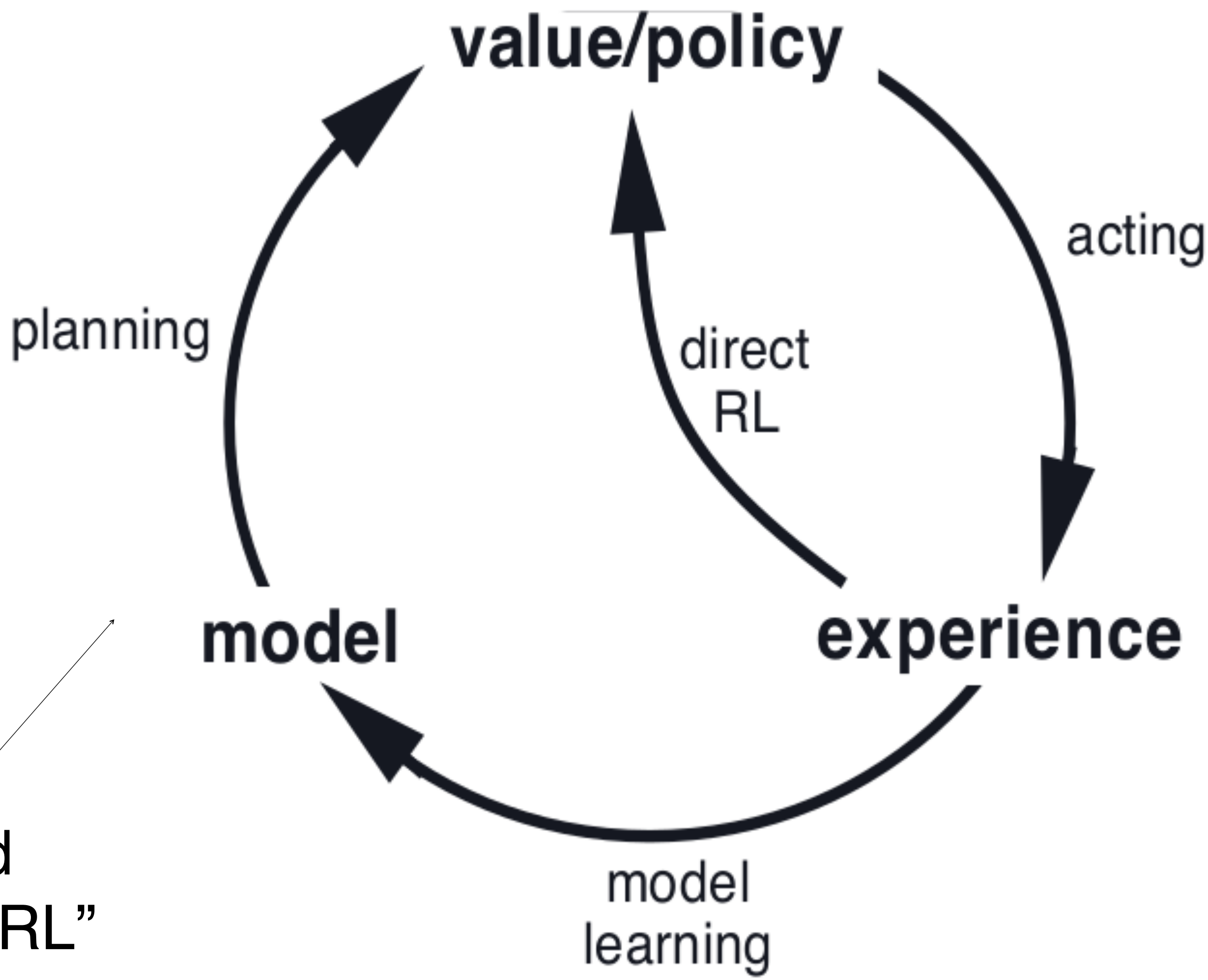
1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
   a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
   $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a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

– why does this satisfy our expanded definition?

Planning vs Learning



Planning vs Learning



Often called
"model-based RL"

Models in RL

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

Two types of models:

1. Distribution model: description of all transition possibilities and their probabilities

2. Sample model: a.k.a. a simulator
   – given a *s,a* pair, the sample model returns next state & reward
   – a sample model is often much easier to get than the distribution model

Models in RL

This is how we defined "model" at the beginning of this course

Model: anything the a ent will respond to its actions

Two types of models:

1. Distribution model: description of all transition possibilities and their probabilities

2. Sample model: a.k.a. a generative/simulation model
   – given a *s,a* pair, the sample model returns next state & reward
   – a sample model is often much easier to get than the distribution model

In this section, we're going to use this type of model a lot

Planning vs learning

Planning typically assumes you use the model to look more than one step into the future (e.g., testing and evaluating possible policies)

Two types of models:

1. Distribution model: this is the typical planning case (use the distribution model to generate a full policy and then can execute that policy) --- this is what we did for dynamic programming (e.g., VI and PI)

2. Sample model: this is used for 'sample-based' or online planning where you only have access to a simulator and are not generating a full policy (e.g., a single action at a times) --- this is what we'll do for Monte-Carlo tree search

Here, we're using a sample model,
but we don't learn the model

Planning vs learning

Planning typically assumes you use the model to look more than one step into the future (e.g., testing and evaluating possible policies)

Two types of models:

1. Distribution model: this is the typical planning case (use the distribution model to generate a full policy and then can execute that policy) --- this is what we did for dynamic programming (e.g., VI and PI)

2. Sample model: this is used for 'sample-based' or online planning where you only have access to a simulator and are not generating a full policy (e.g., a single action at a times) --- this is what we'll do for Monte-Carlo tree search

Planning methods don't learn the model by themselves

Planning

<u>An unusual way to do planning:</u>

$$\text{model} \longrightarrow \text{simulated experience} \xrightarrow{\text{backups}} \text{values} \longrightarrow \text{policy}$$

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

Planning

<u>An unusual way to do planning:</u>

$$\text{model} \longrightarrow \text{simulated experience} \xrightarrow{\text{backups}} \text{values} \longrightarrow \text{policy}$$

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

Here, we're using a sample model,
but we don't learn the model

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

Direct RL: Do update directly using real experience

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

Direct RL: Do update directly using real experience
Indirect RL: Learn model using real experience

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

Direct RL: Do update directly using real experience
Indirect RL: Learn model using real experience,
then simulate experience using model, do 'planning' update using simulated experience

Question

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

This version assumes deterministic transitions and rewards, how would you extend it to the stochastic case?

Question

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

Think of as many ways as you can to implement “model”

Question

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

Think of as many ways as you can to implement “model”
- just buffer all previously experienced transitions (really a model?)
- estimate tabular transition probabilities for s,a pairs
- learn a neural network that samples from next states

Question

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

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

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

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

(f) Loop repeat n times:

$S \leftarrow$ random previously observed state

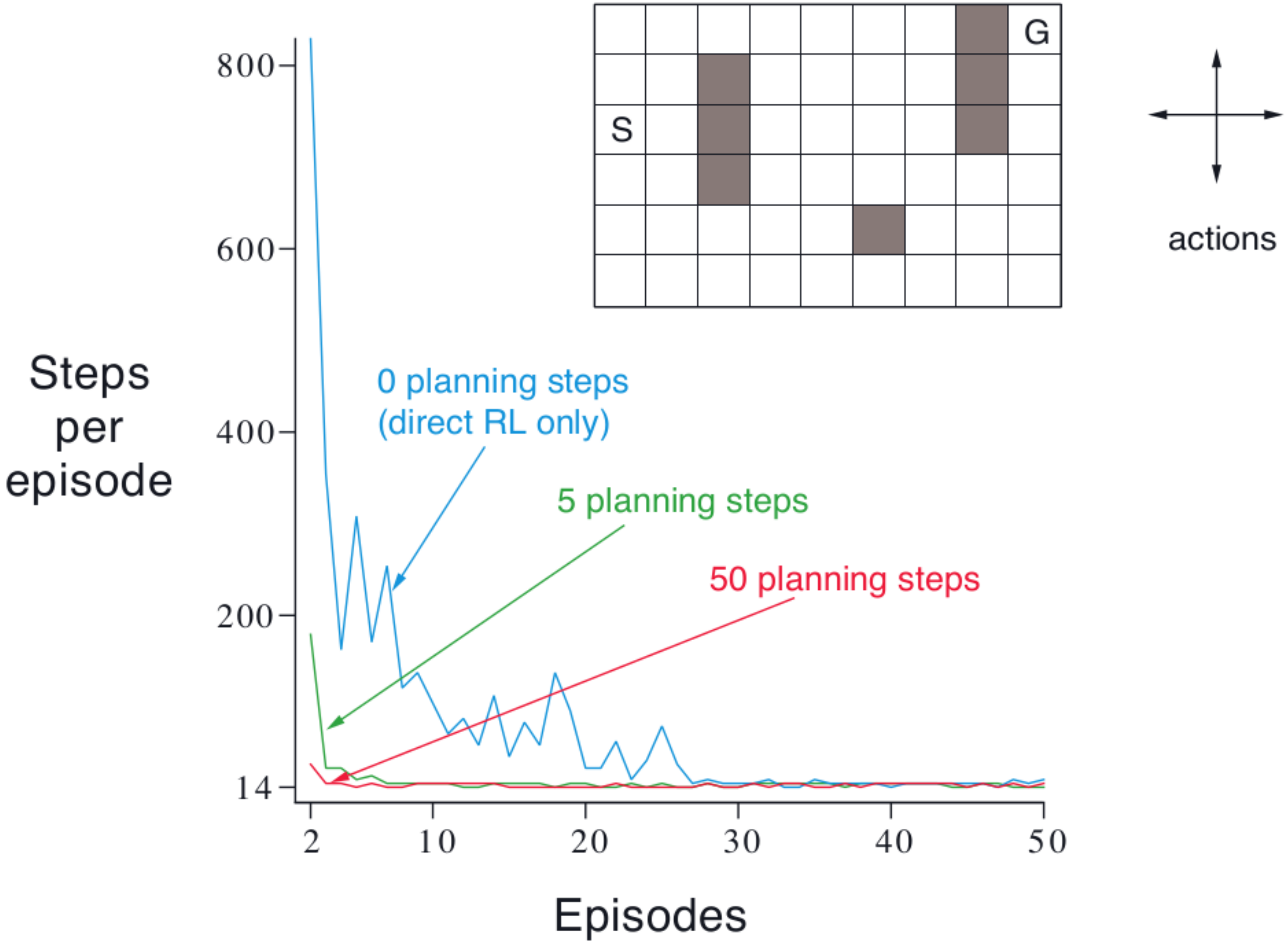
$A \leftarrow$ random action previously taken in S

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

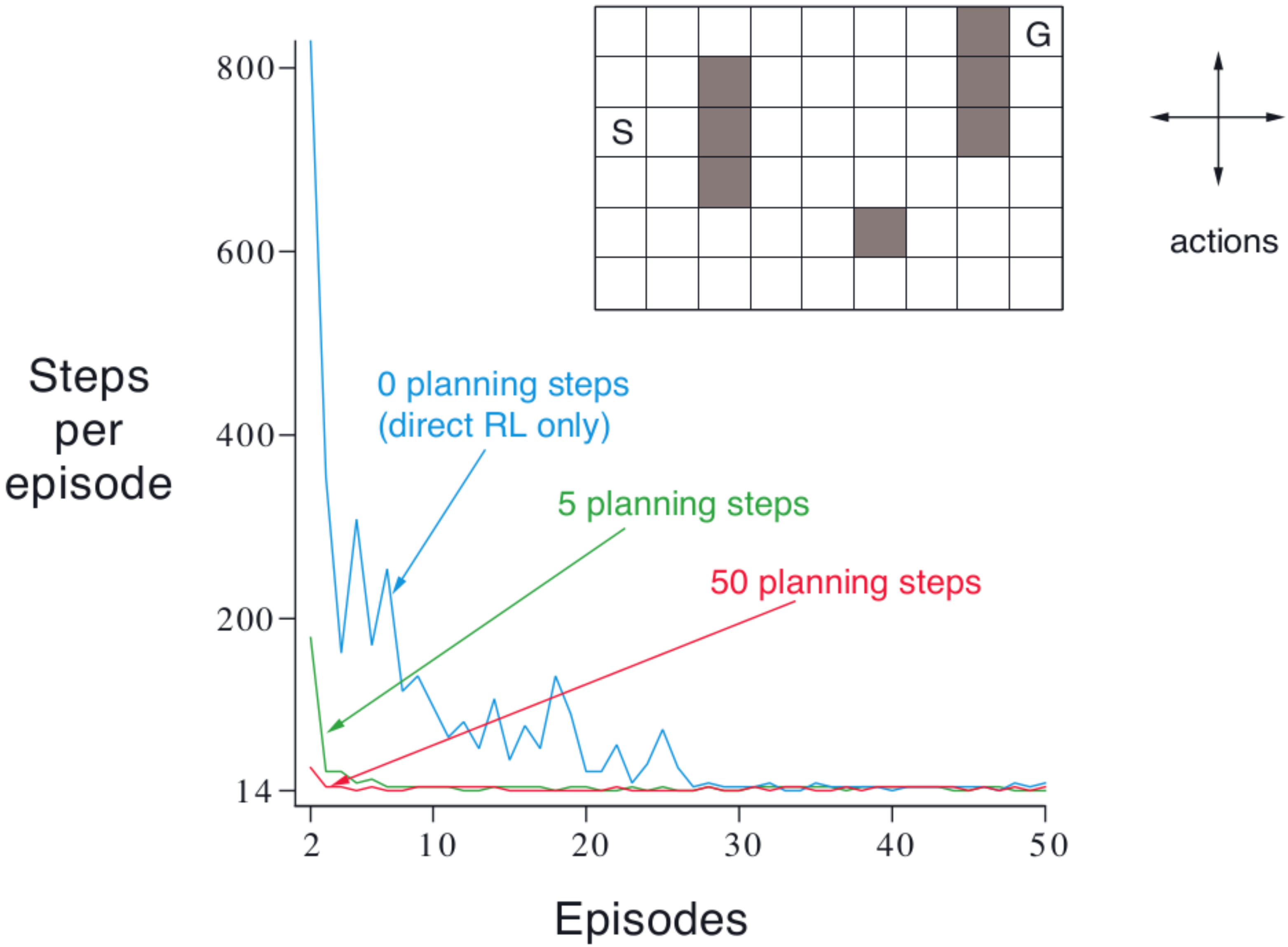
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Modern versions learn a NN for a model

Dyna-Q on a Simple Maze



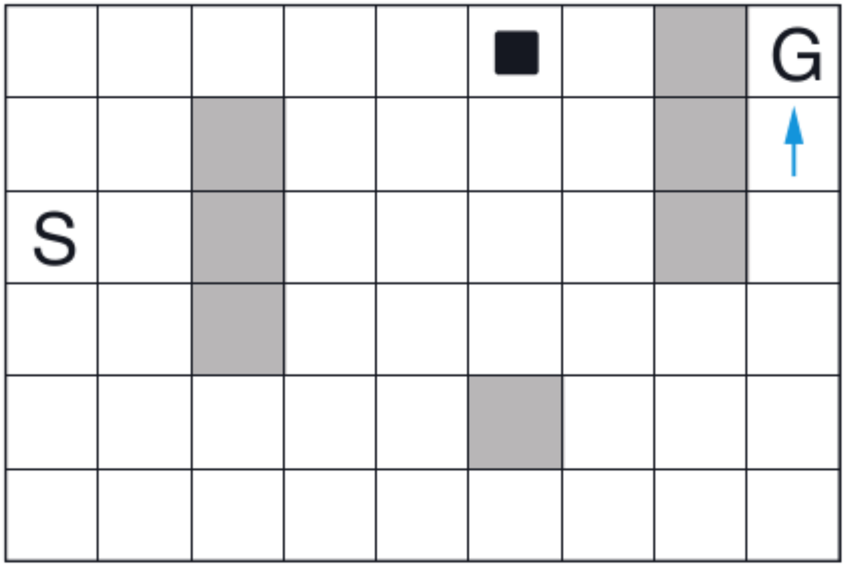
Dyna-Q on a Simple Maze



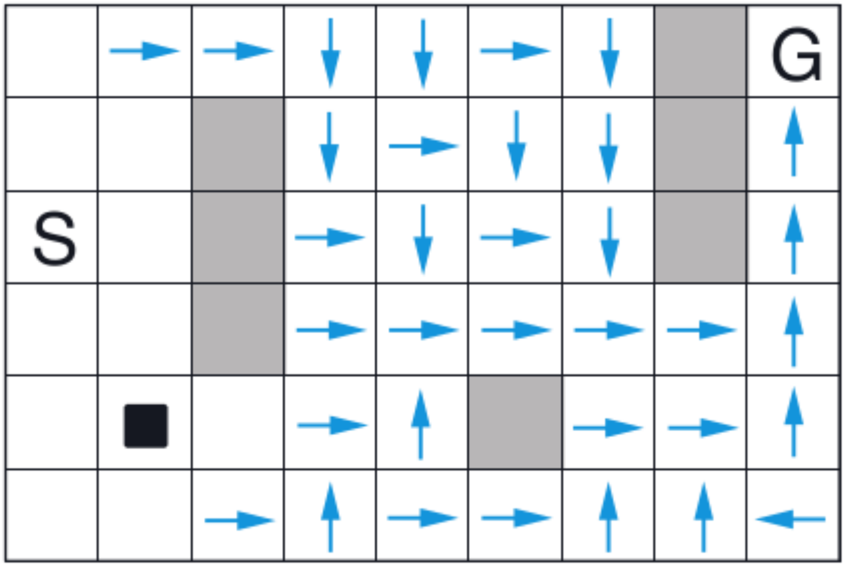
Would the results look as good if the domain was stochastic?

Why does Dyna-Q do so well?

WITHOUT PLANNING (n=0)



WITH PLANNING (n=50)



Policies found using Q-learning vs dyna-Q halfway through second episode

- dyna-Q w/ n=50
- optimal policy after three episodes!

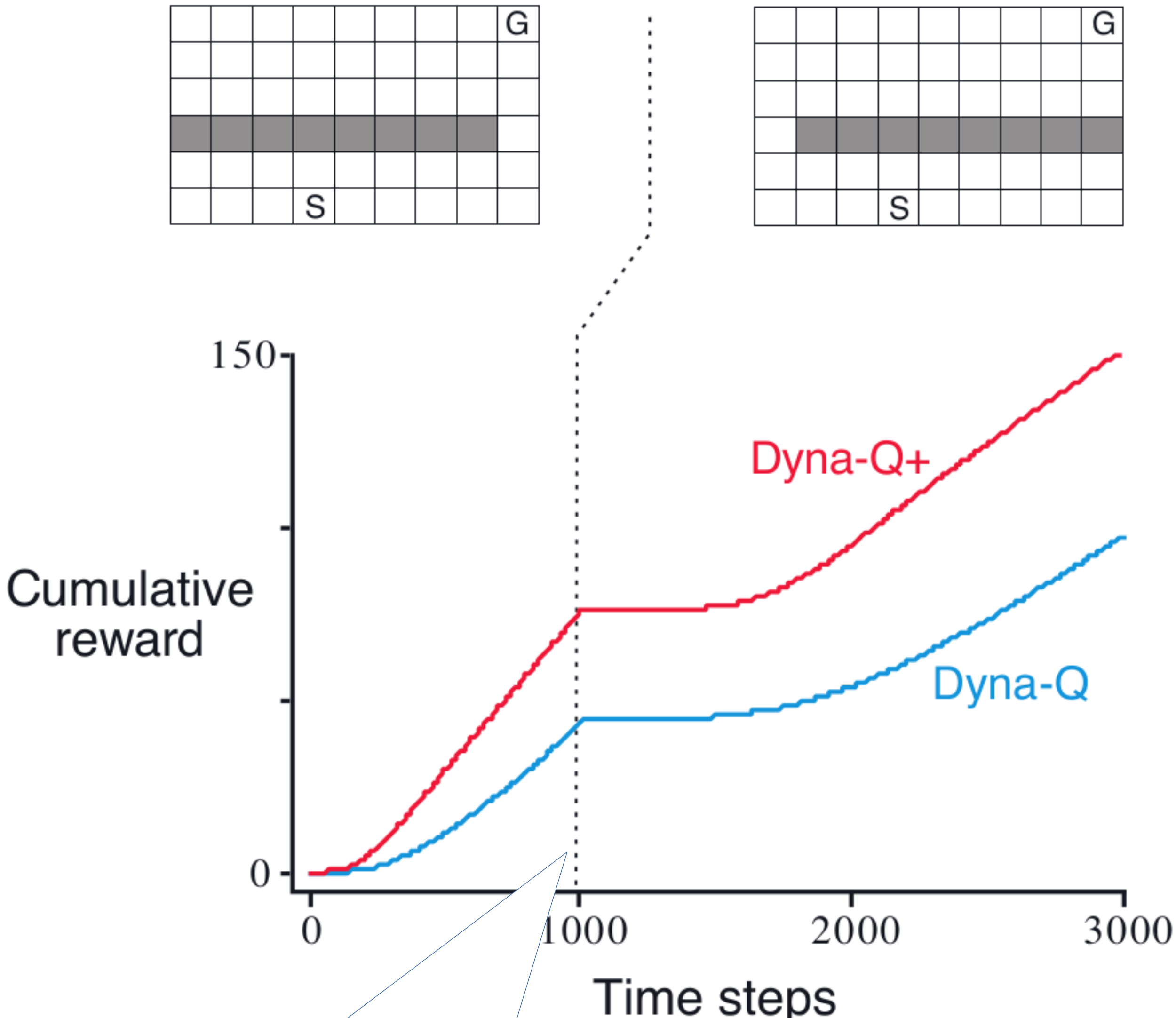
When the model is wrong

- So far, we have considered models that:
  - Start empty and are always updated with correct info
- The model can be wrong! Because:
  - environment might be stochastic and we have only seen a few samples
  - the environment has changed
- Planning is likely to compute a suboptimal policy in this case

- Imagine the world changed, and:
  - The suboptimal policy leads to discovery and correction of the modeling error

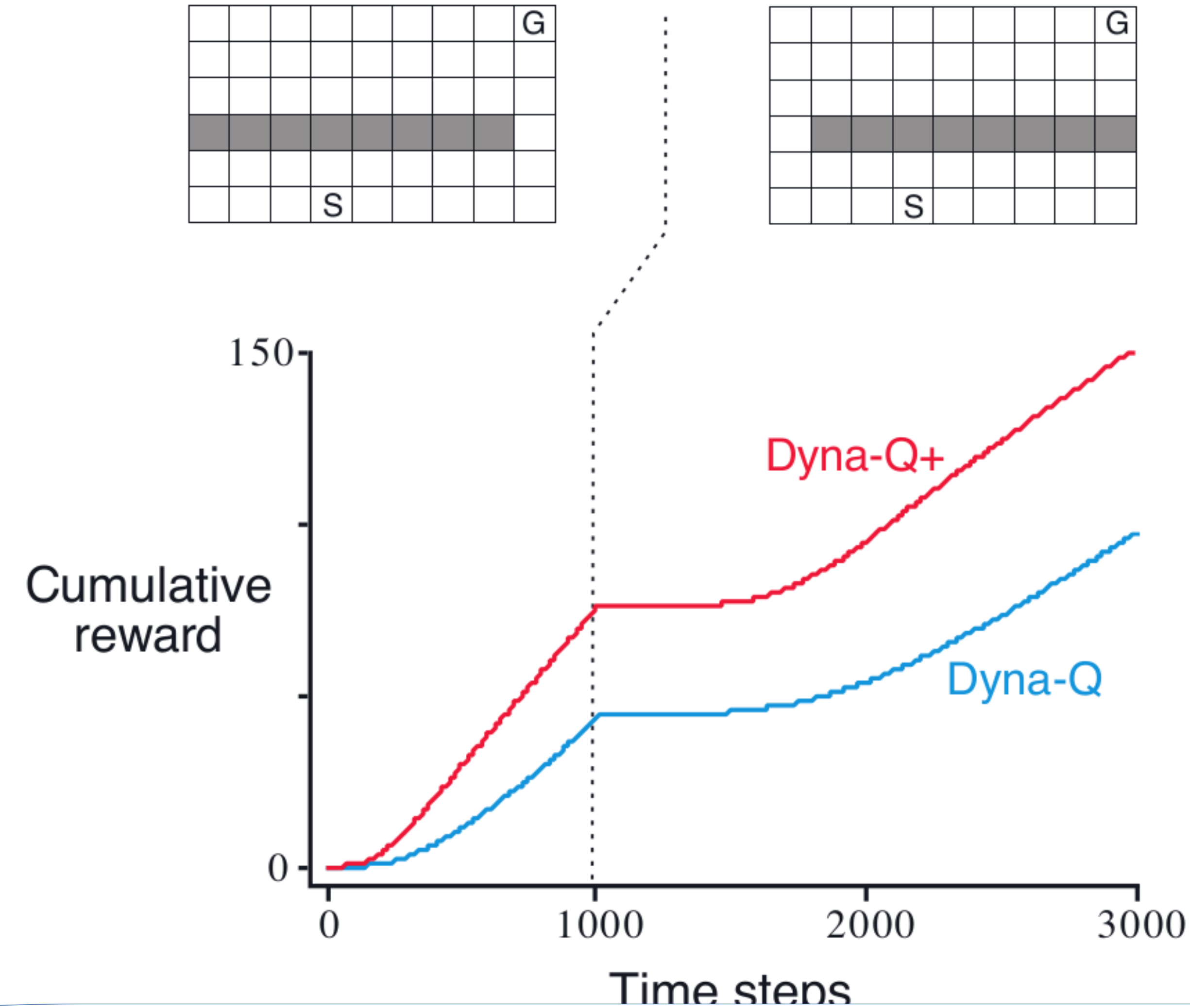
What happens with incorrect model?

(SB, Example 8.2)



Environment changes here

Think-pair-share



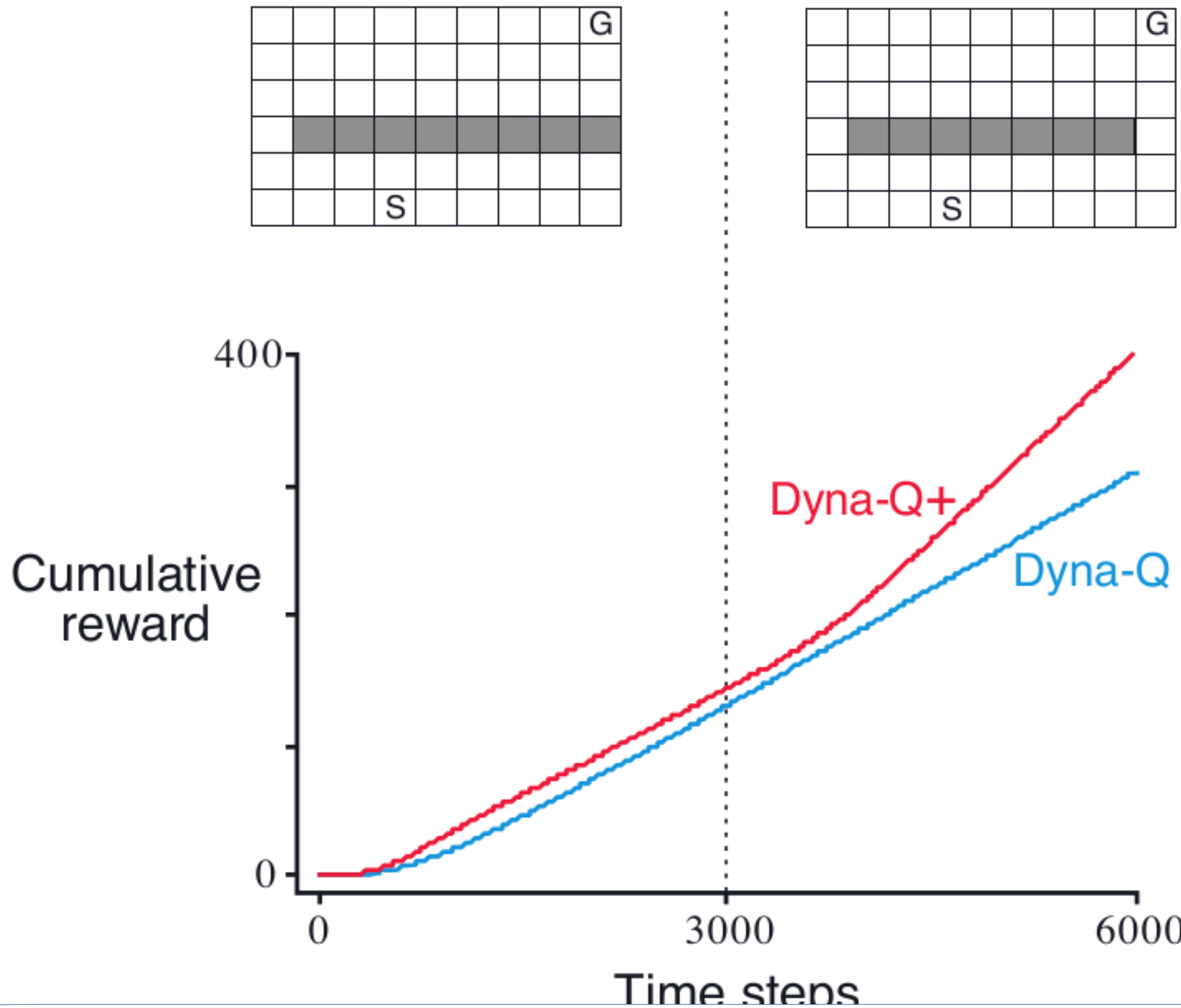
(SB, Example 8.2)

Questions:

- why does dyna-Q stop getting reward?
- why does it start again?
- what would happen if the change was the other way around (in either case)?

Think-pair-share

(SB, Example 8.2)



Questions:

– why does dyna-Q stop getting reward?
– why does it start again?
– what would happen if the change was the other way around (in either case)?

What is Dyna-Q+

- Even with e-greedy policy Dyna-Q won't explore enough to find new path
- Dyna-Q+ uses an "exploration bonus" (like UCB):
  - Keeps track of time since each state-action pair was tried for real
  - 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

$$R + \kappa\sqrt{\tau}$$

some small constant to balance exploration

time since last visiting the state-action pair

  - What impact does adding this exploration bonus have?

What is Dyna-Q+

- Even with e-greedy policy Dyna-Q won’t explore enough to find new path
- Dyna-Q+ uses an “exploration bonus” (like UCB):
  - Keeps track of time since each state-action pair was tried for real
  - 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

$$R + \kappa\sqrt{\tau}$$

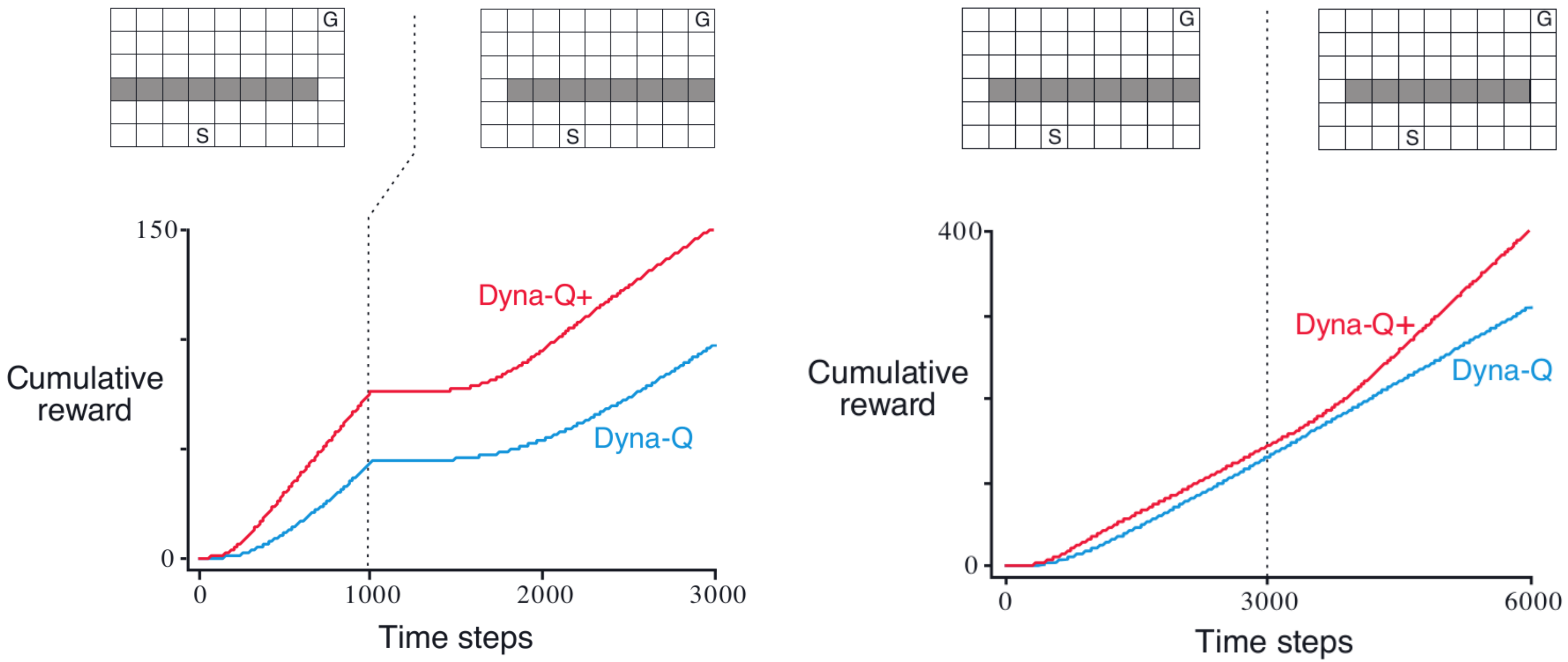
time since last visiting the state-action pair

some small constant to balance exploration

  - The agent actually “plans” how to visit long unvisited states

Note: Better exploration is a key benefit in model-based approaches

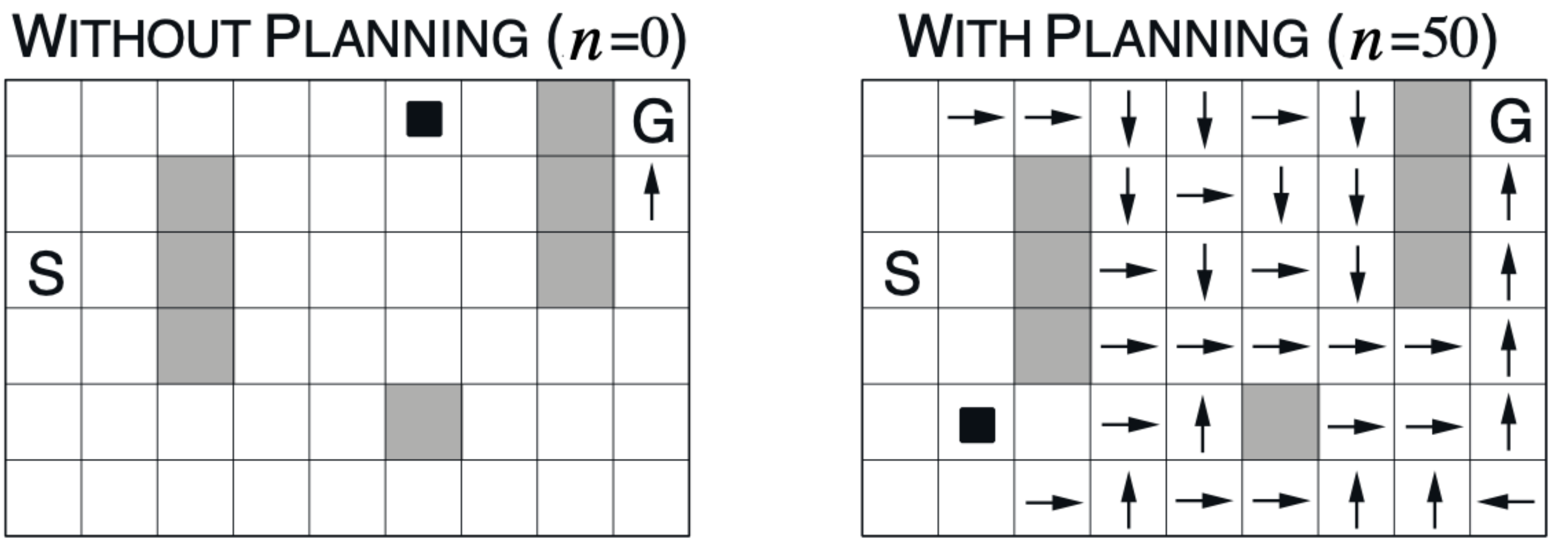
Think-pair-share



Exercise 8.2 Why did the Dyna agent with exploration bonus, Dyna-Q+, perform better in the first phase as well as in the second phase of the blocking and shortcut experiments? □

Prioritizing Search Control

- Consider the second episode in the Dyna maze
    - The agent has successfully reached the goal once…



- In larger problems, the number of states is so large that unfocused planning would be extremely inefficient

Prioritized Sweeping

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

Unfocused replay from model

Prioritized Sweeping

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

Unfocused replay from model – can we do better?

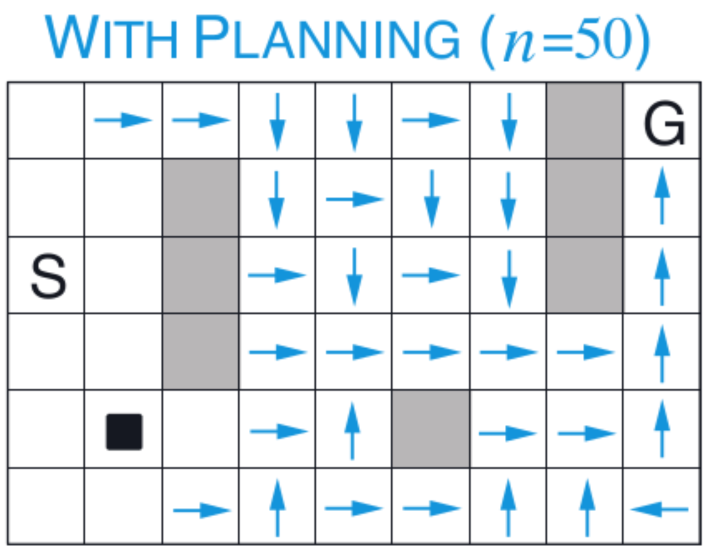
Prioritized Sweeping

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

Unlike real experience, we can control which state-action pairs to replay

Prioritized Sweeping



Instead of replaying **all** of these transitions on each iteration, just replay the important ones…

– Which states or state-action pairs should be generated during planning?

– Work backward from states whose value has just changed

– Maintain a priority 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 according to their priorities

Prioritized Sweeping

Prioritized sweeping for a deterministic environment

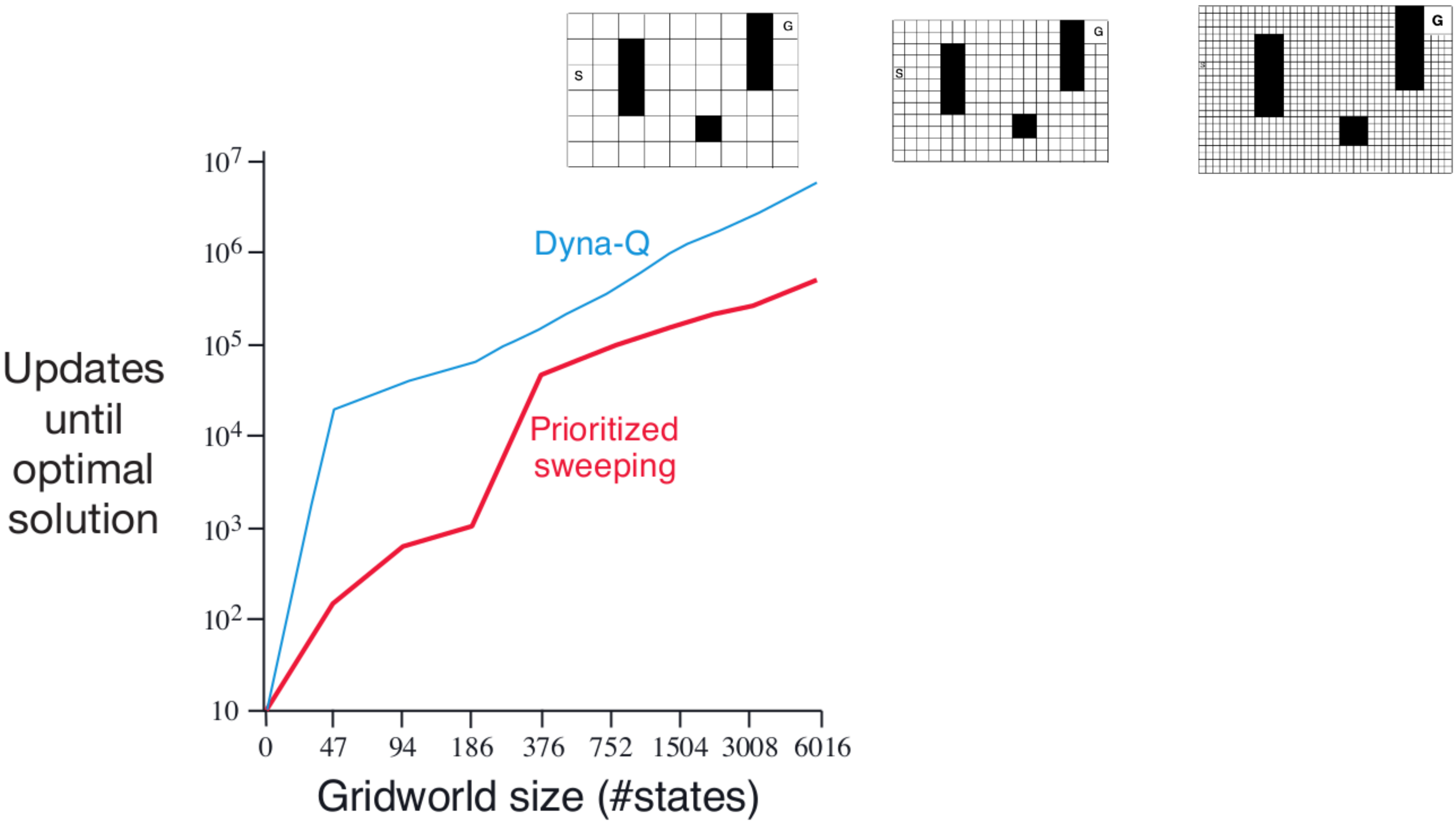
Initialize $Q(s,a)$, $Model(s,a)$, for all s,a, and $PQueue$ to empty
Loop forever:
  (a) $S \leftarrow$ current (nonterminal) state
  (b) $A \leftarrow policy(S,Q)$
  (c) Take action A; observe resultant reward, R, and state, S'
  (d) $Model(S,A) \leftarrow R, S'$
  (e) $P \leftarrow |R + \gamma \max_a Q(S',a) - Q(S,A)|$.
  (f) if $P > \theta$, then insert S,A into $PQueue$ with priority P
  (g) Loop repeat n times, while $PQueue$ is not empty:
    $S, A \leftarrow first(PQueue)$
    $R, S' \leftarrow Model(S,A)$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    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

TD error

what's this part doing?

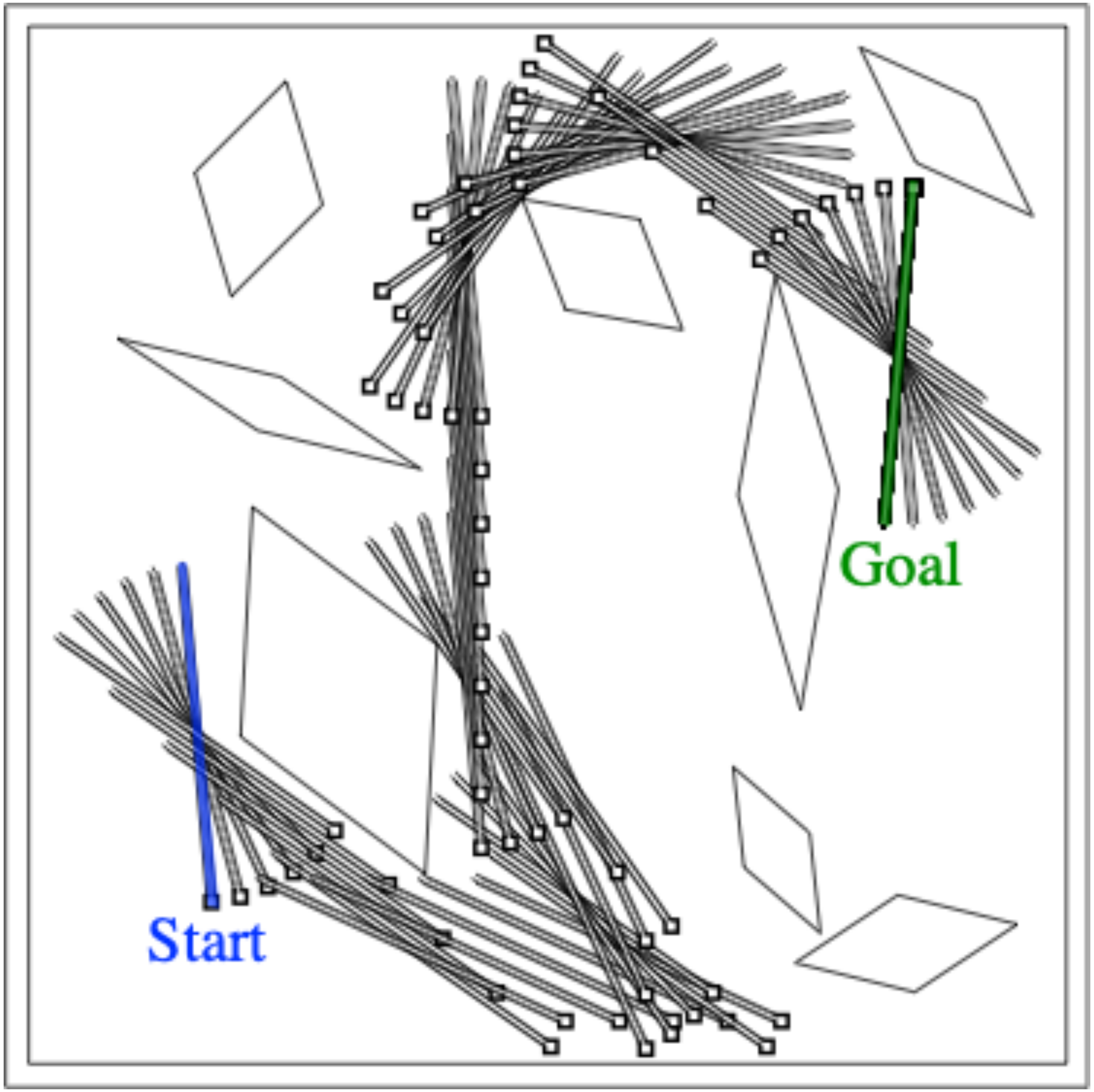
Prioritized sweeping isn't really a planning method so it can be used with lots of RL methods

Prioritized Sweeping: Performance



Both use n=5 backups per environmental interaction

Rod Maneuvering (Moore and Atkeson 1993)

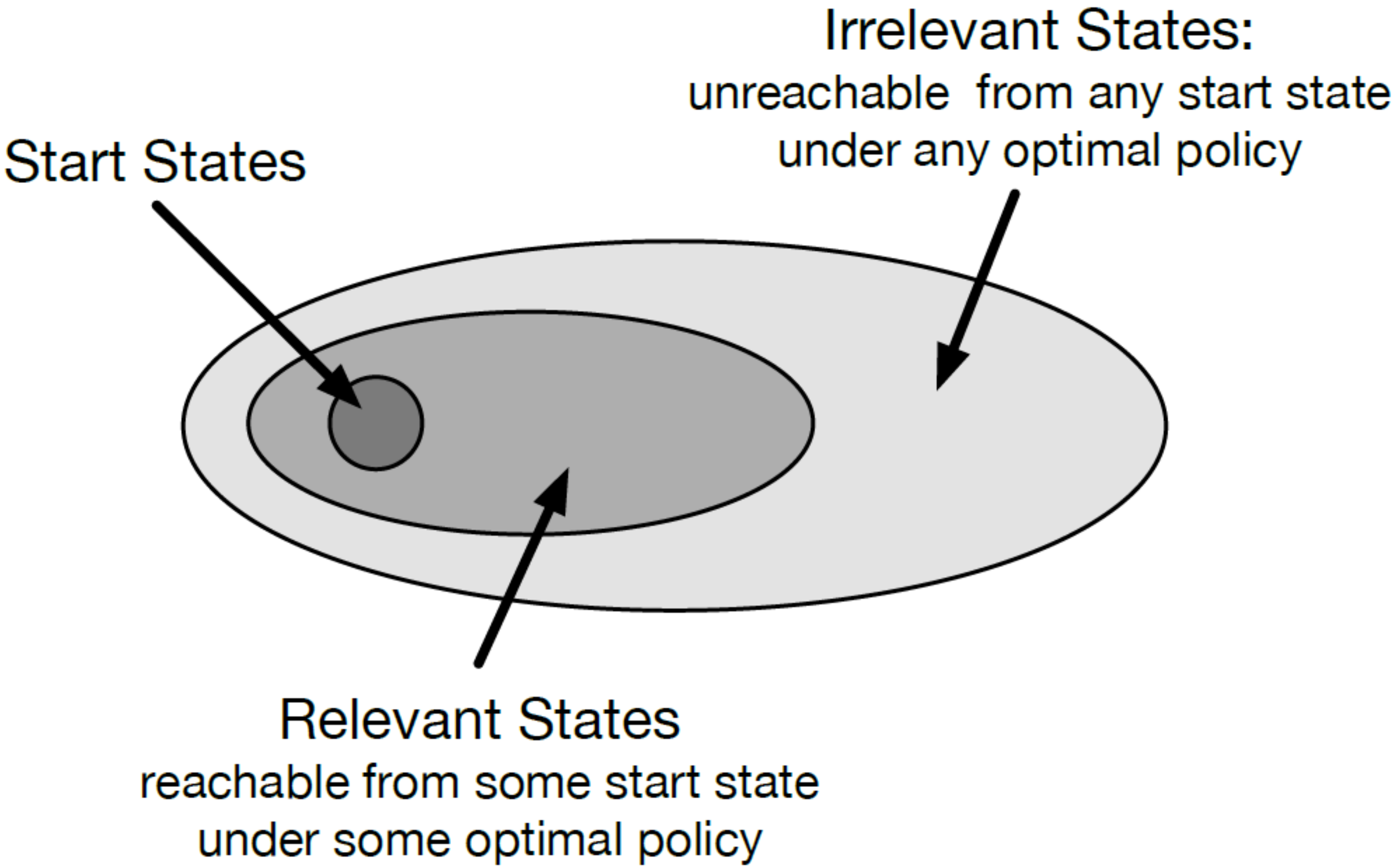


Solved by prioritized sweeping, but probably too large for unprioritized methods

Model-based RL and sample-based planning

- This chapter only touches the surface of both model-based RL and different planning methods
- Model-based RL
  - Typically, reinforcement learning methods that combine model-learning to improve exploration, transfer, etc.
  - Examples include Dyna, MuZero, World Models, Dreamer, etc.
- Sample-based planning
  - Typically, assumes the model is given in the form of a simulator (so can access any s,a and get s’, r)
  - Examples include, RTDP and MCTS (next)

Optimality without visiting all states



Online planning methods

- Dyna used planning to generate more data to improve the value function for all states (book calls this background planning)

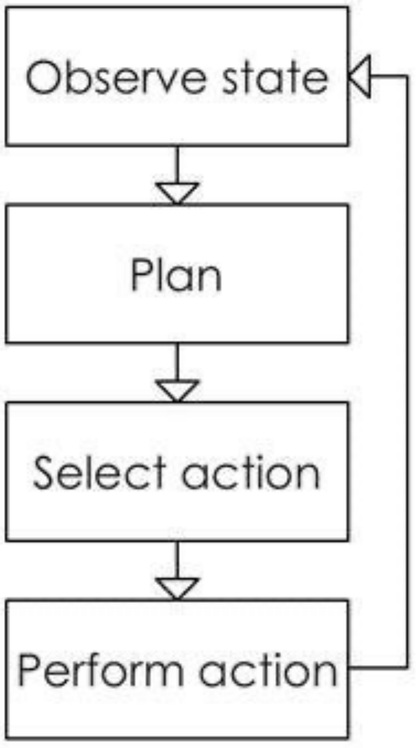
- Instead, online planning methods try to compute an optimal action from current state (book calls this decision-time planning)

  - Plan up to some horizon

  - States reachable from the current state is typically small compared to full state space

  - Heuristics and branch-and-bound techniques allow search space to be pruned

  - Monte Carlo methods provide approximate solutions



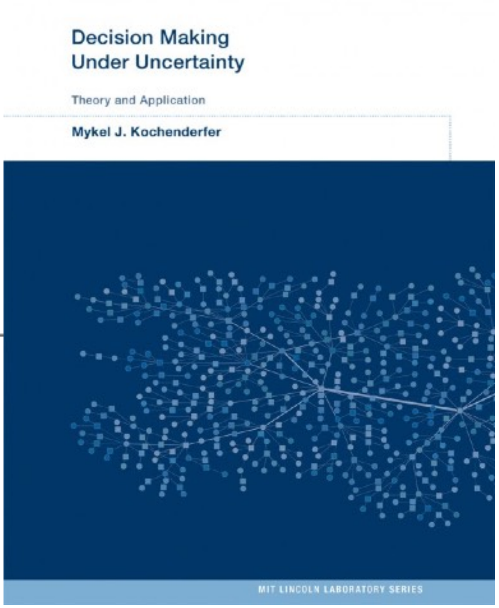
Forward Search

- Provides optimal action from current state s up to depth d
- Recall

Algorithm 4.6 Forward search

function SELECTACTION(s, d)
    if $d = 0$
        return $(\text{NIL}, 0)$
    $(a^*, v^*) \leftarrow (\text{NIL}, -\infty)$
    for $a \in A(s)$
        $v \leftarrow R(s, a)$
        for $s' \in S(s, a)$
            $(a', v') \leftarrow$ SELECTACTION$(s', d - 1)$
            $v \leftarrow v + \gamma T(s' \mid s, a) v'$
        if $v > v^*$
            $(a^*, v^*) \leftarrow (a, v)$
    return (a^*, v^*)

Pseudocode is from here:



- Time complexity is $O((|S| \times |A|)^d)$
- Tree search (expectimax) over and over again!

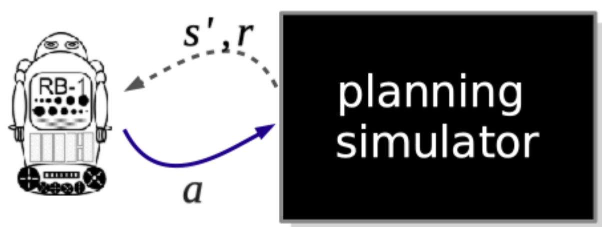
Online Planning

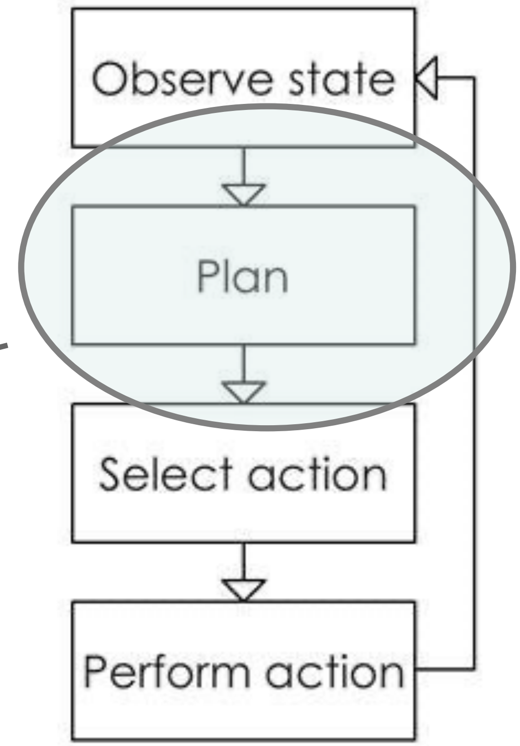
Most planning: use model to compute policy

- ►e.g., **dynamic programming**

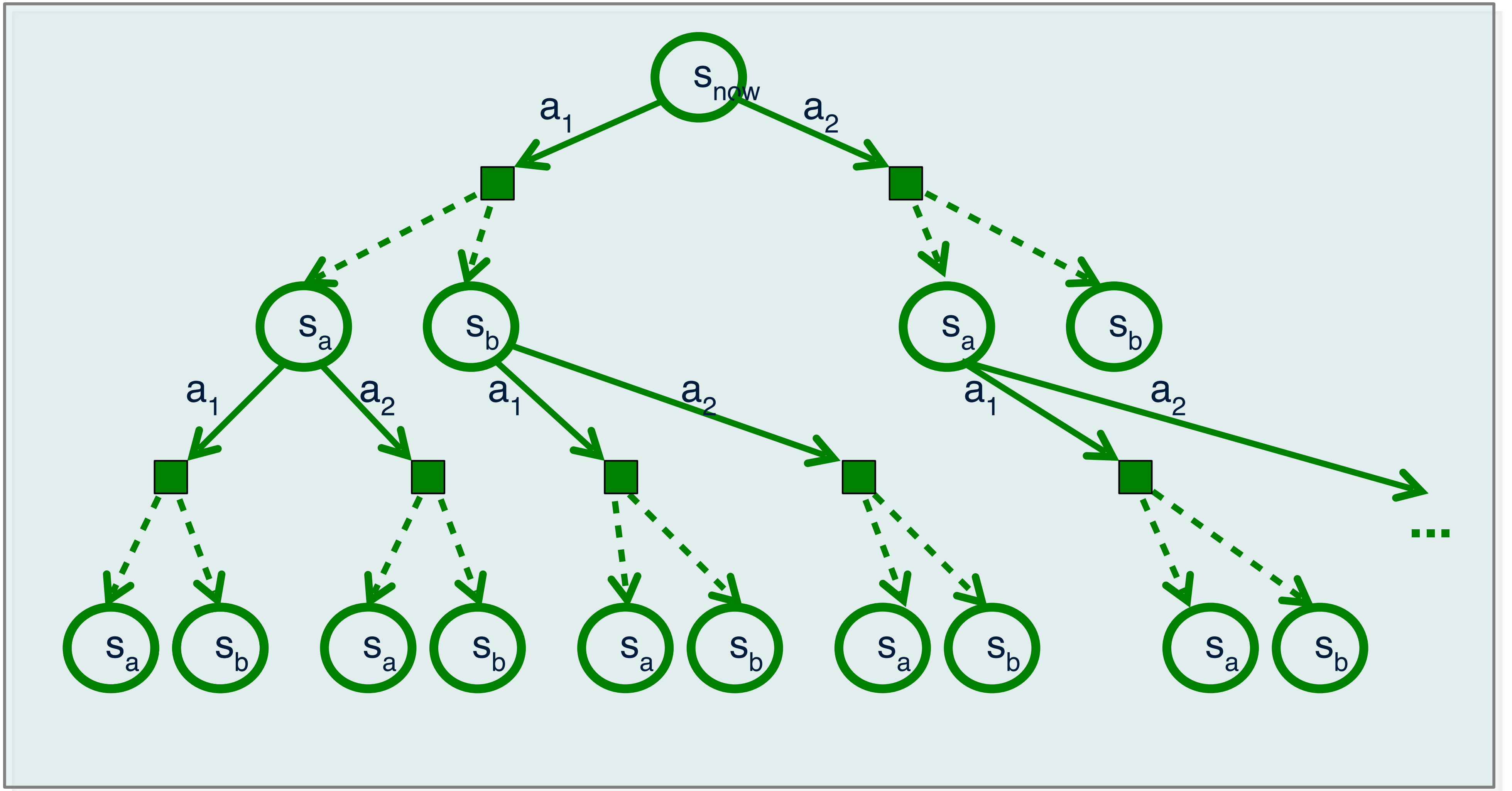
Simulation-based planning, e.g. **MCTS**:

- ►use sampled trajectories of simulated experience



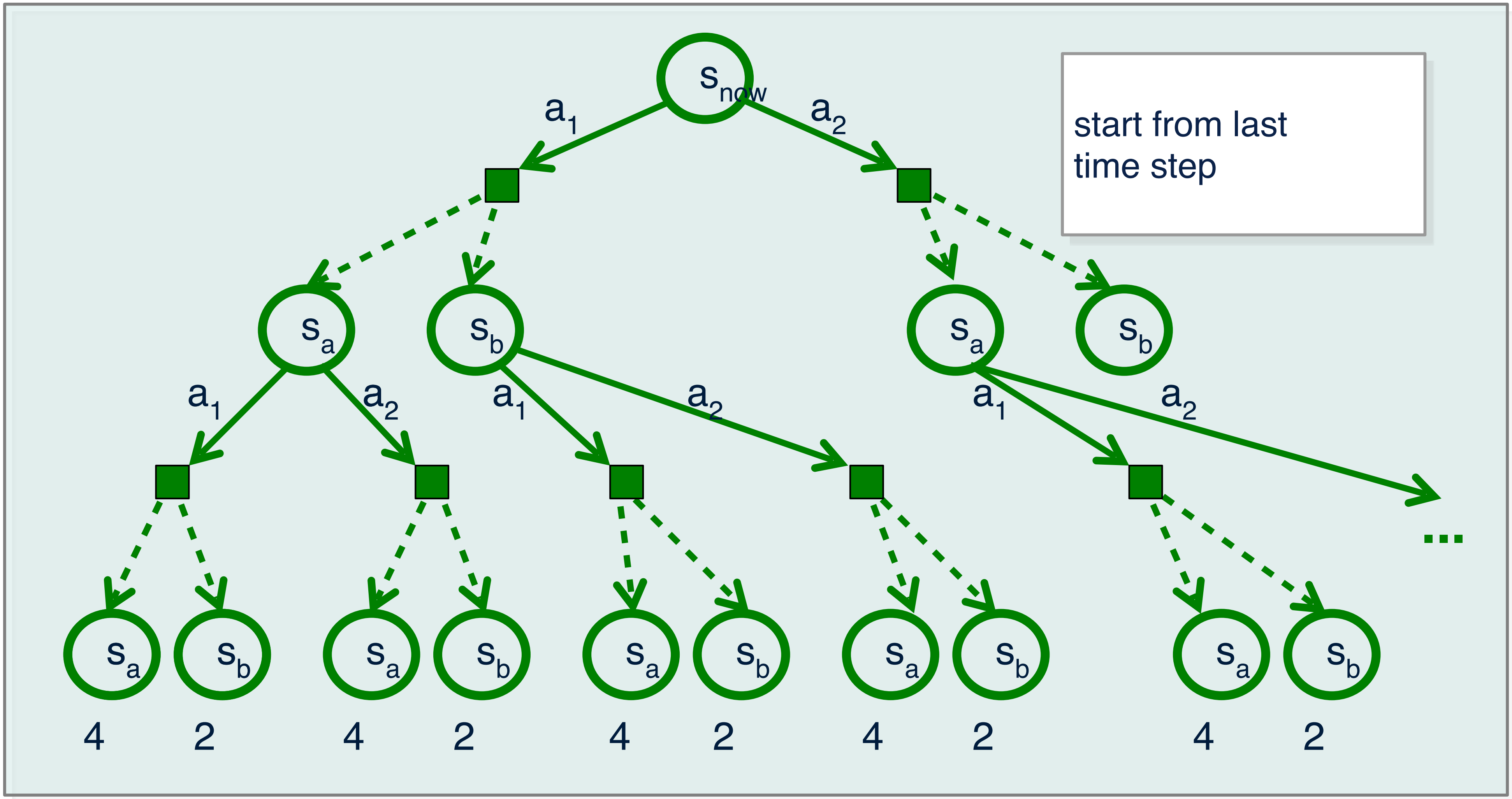


Forward Search

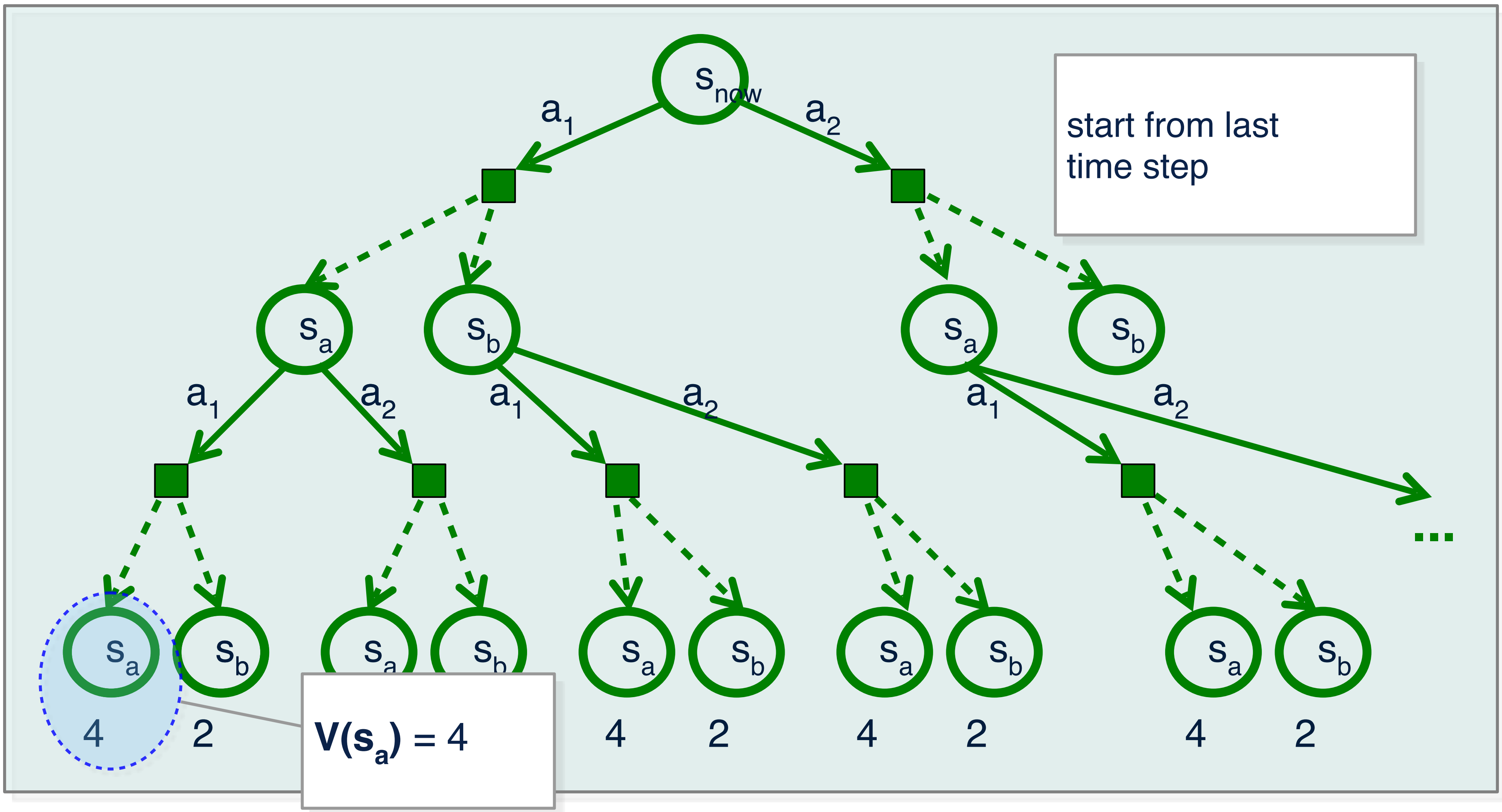


- Construct a plan for *T* time steps into the future

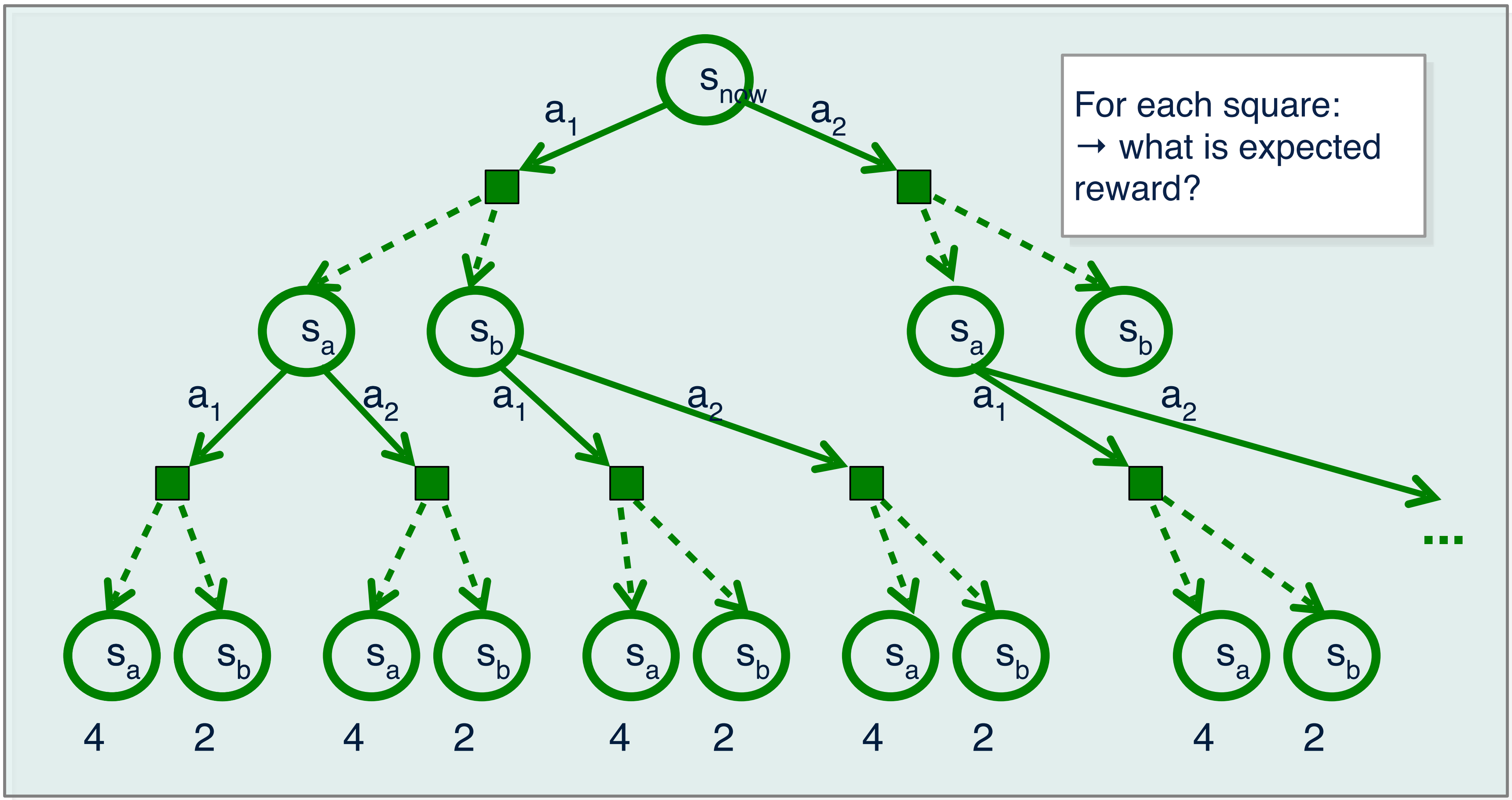
Forward Search



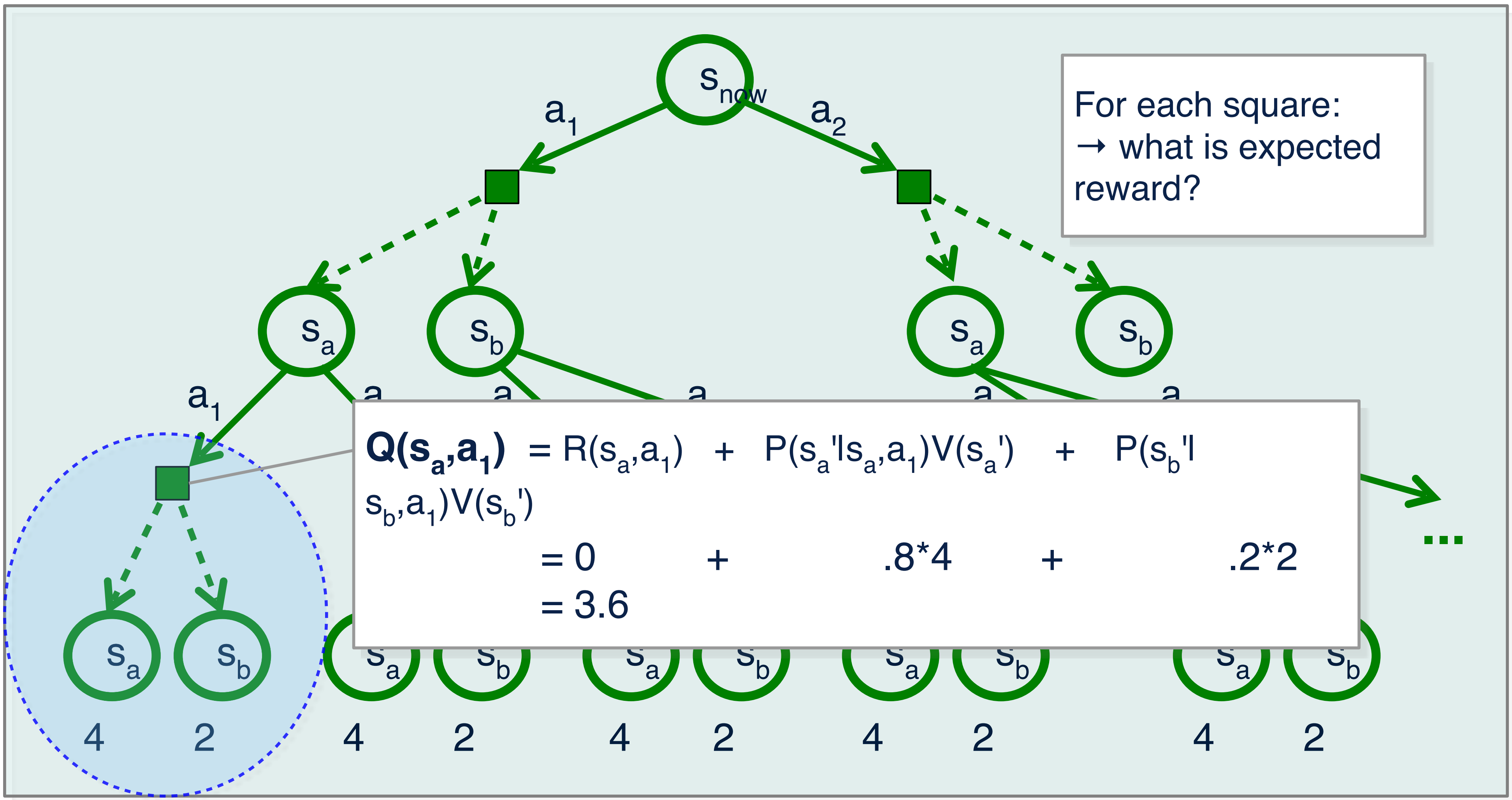
Forward Search



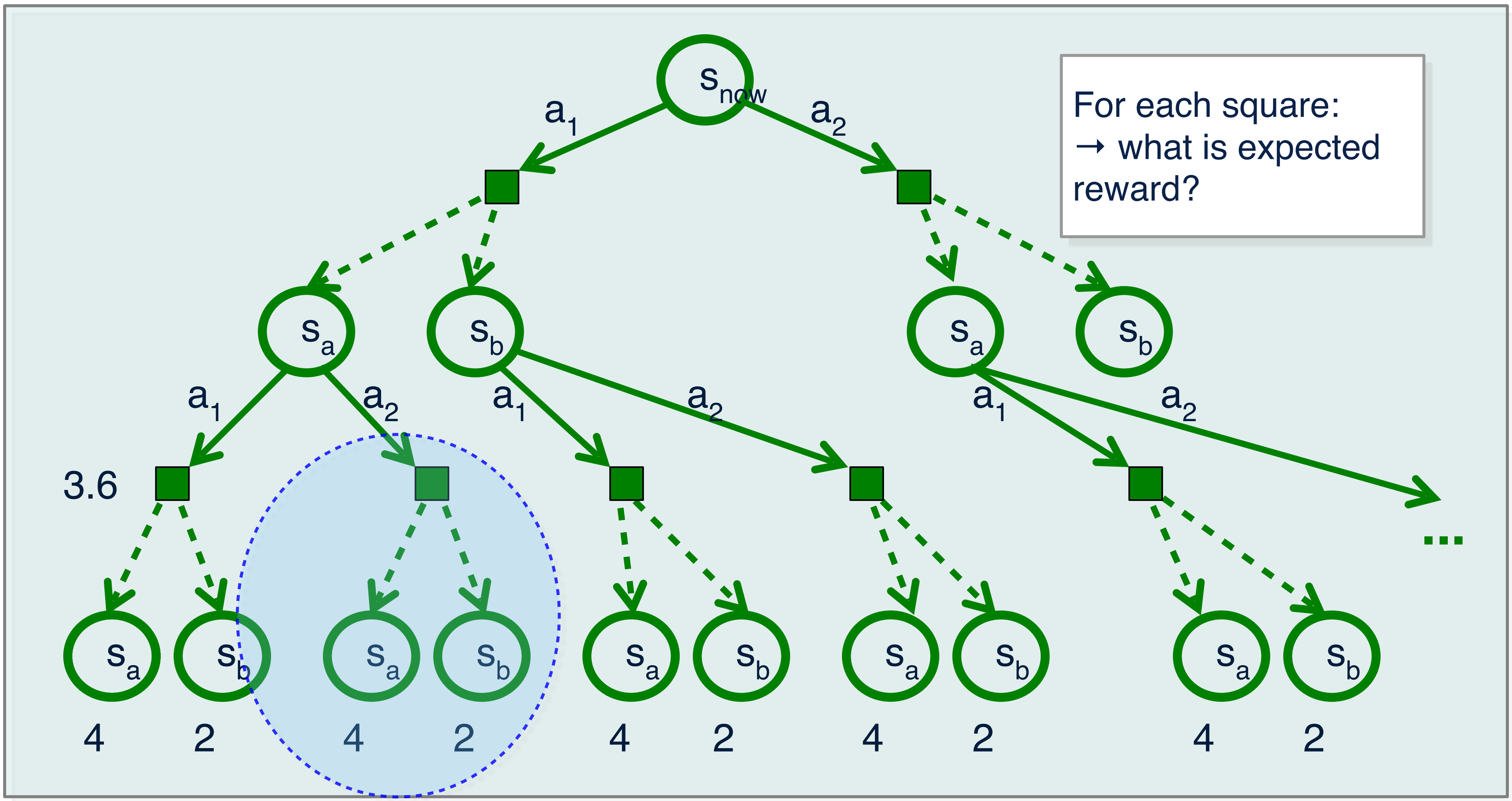
Forward Search



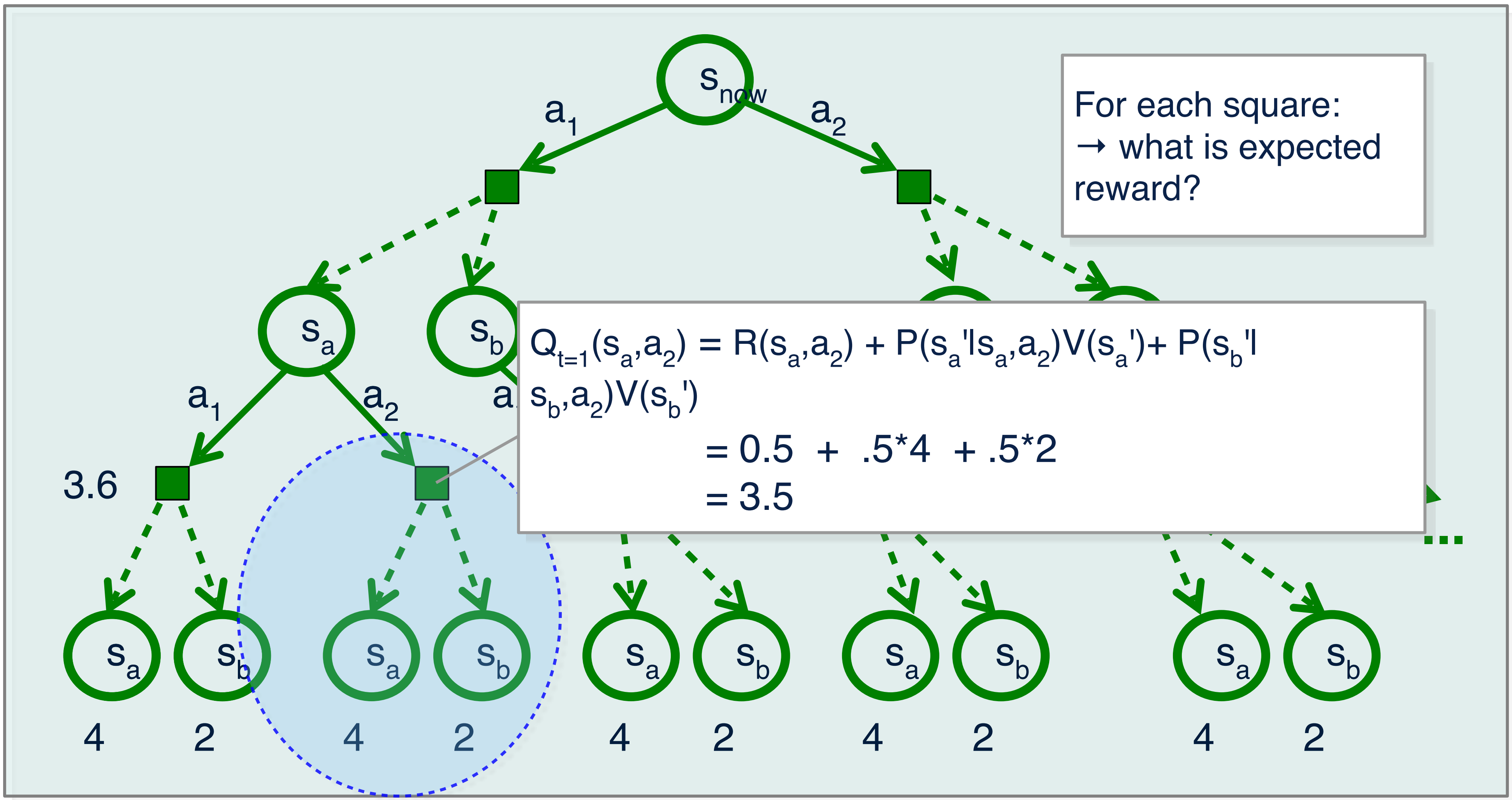
Forward Search



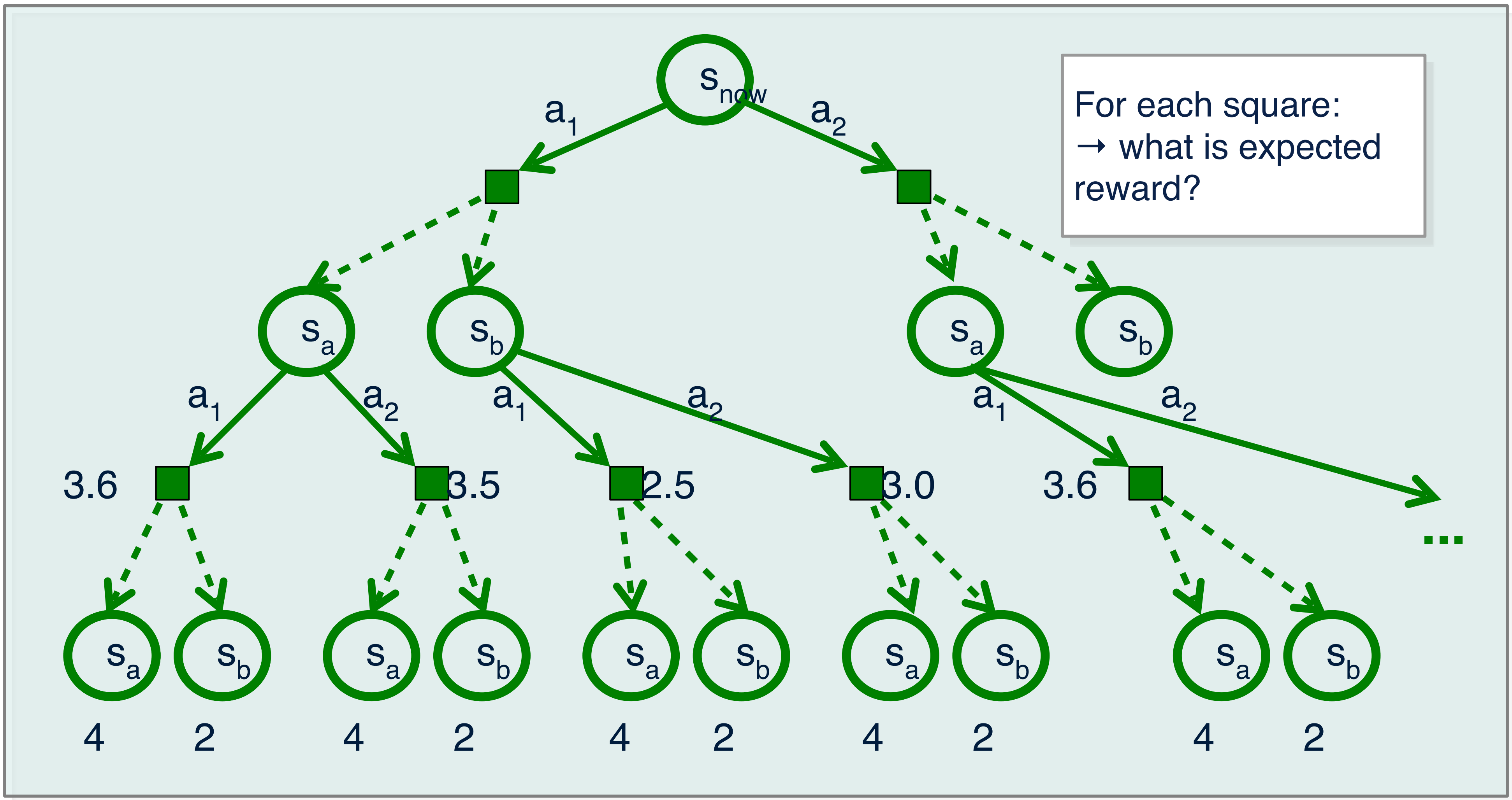
Forward Search



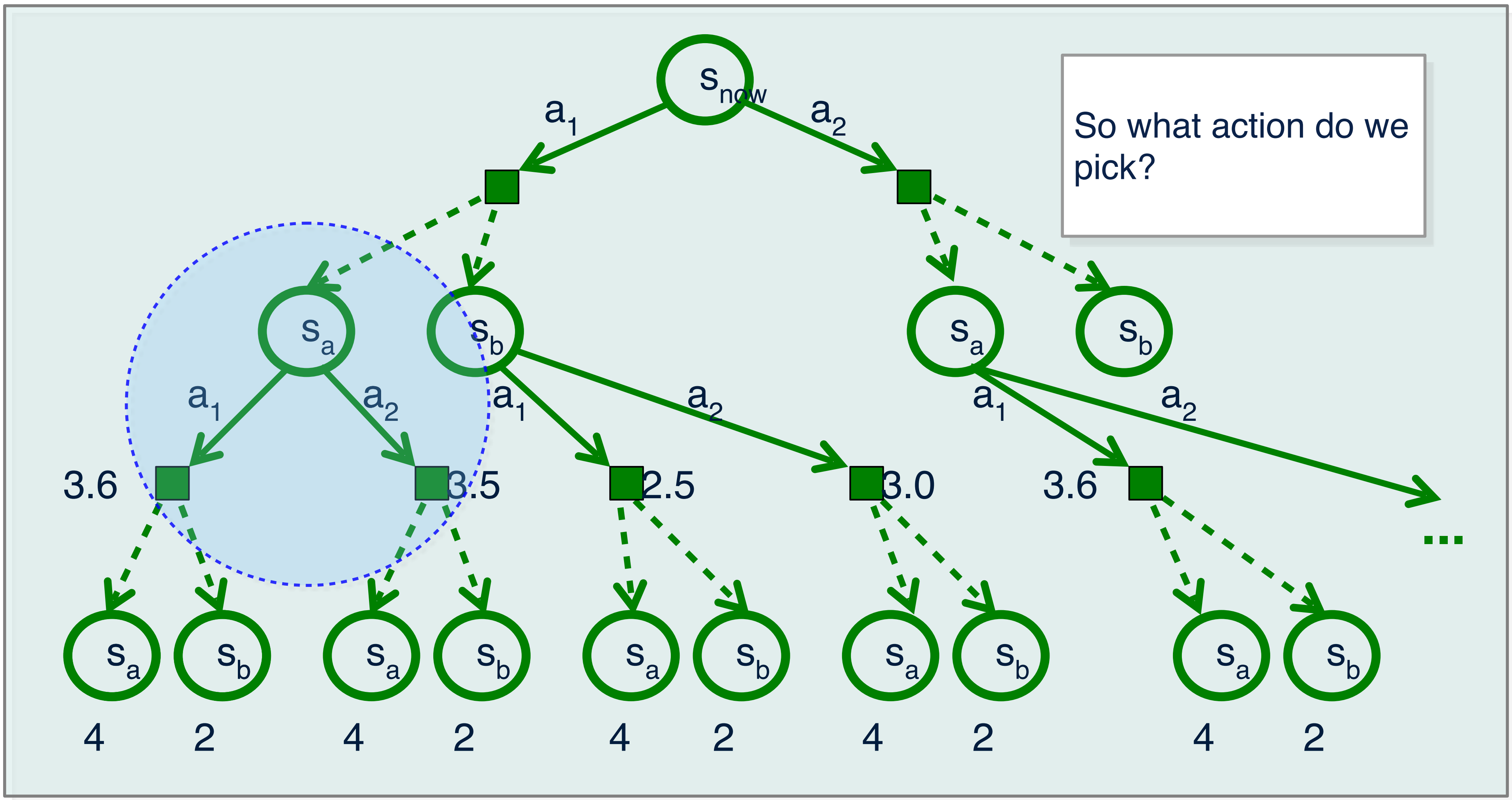
Forward Search



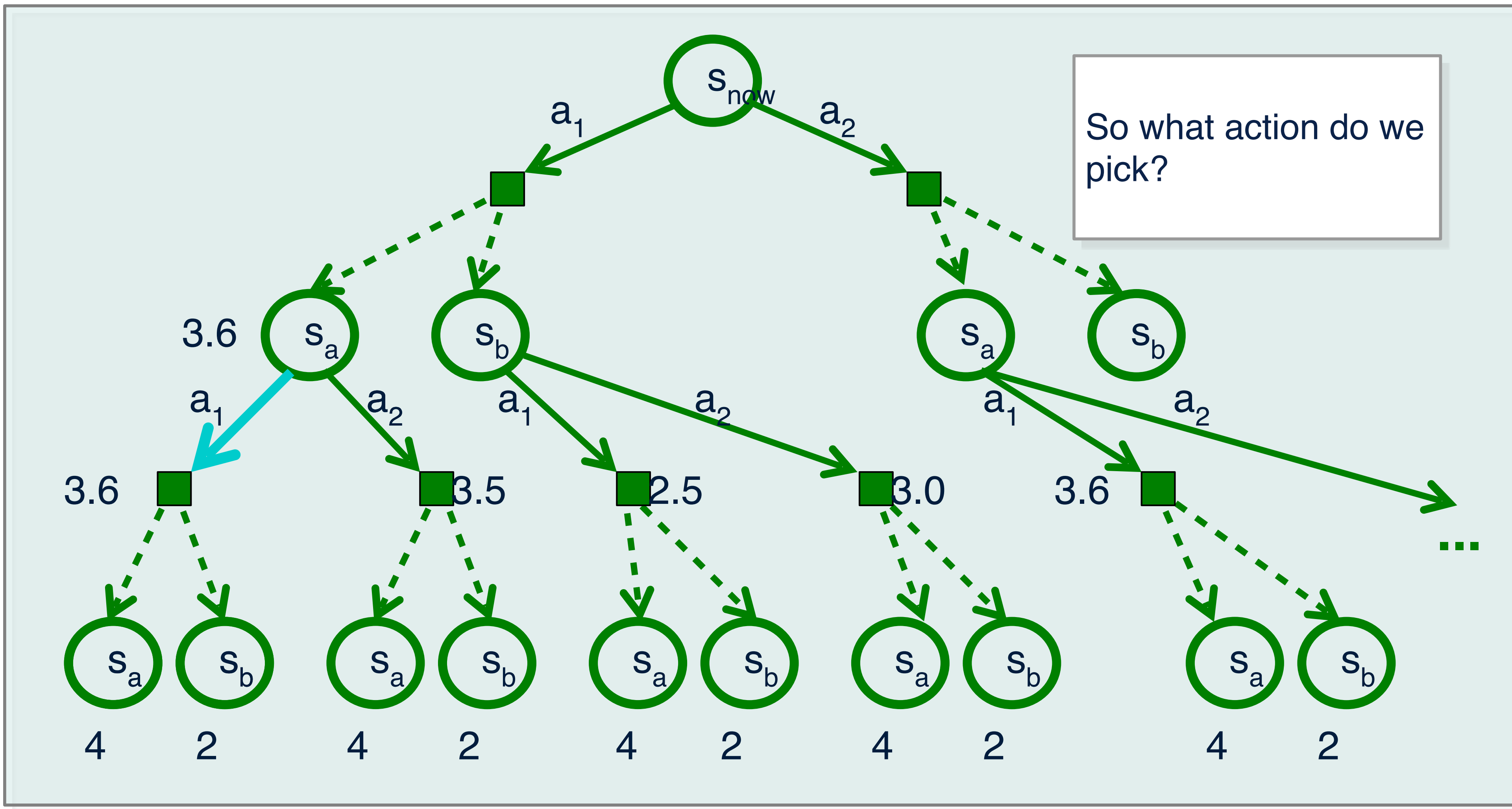
Forward Search



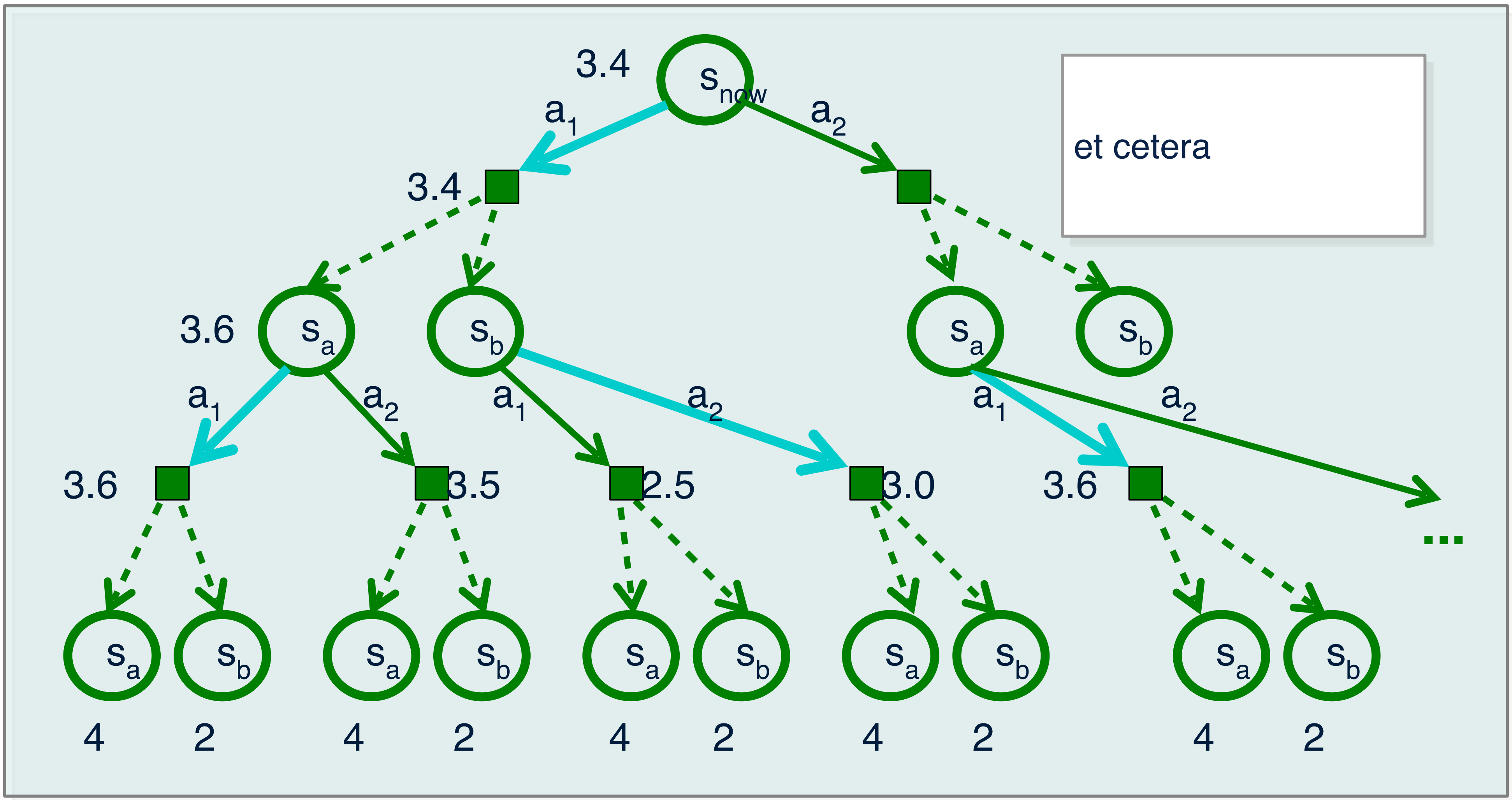
Forward Search



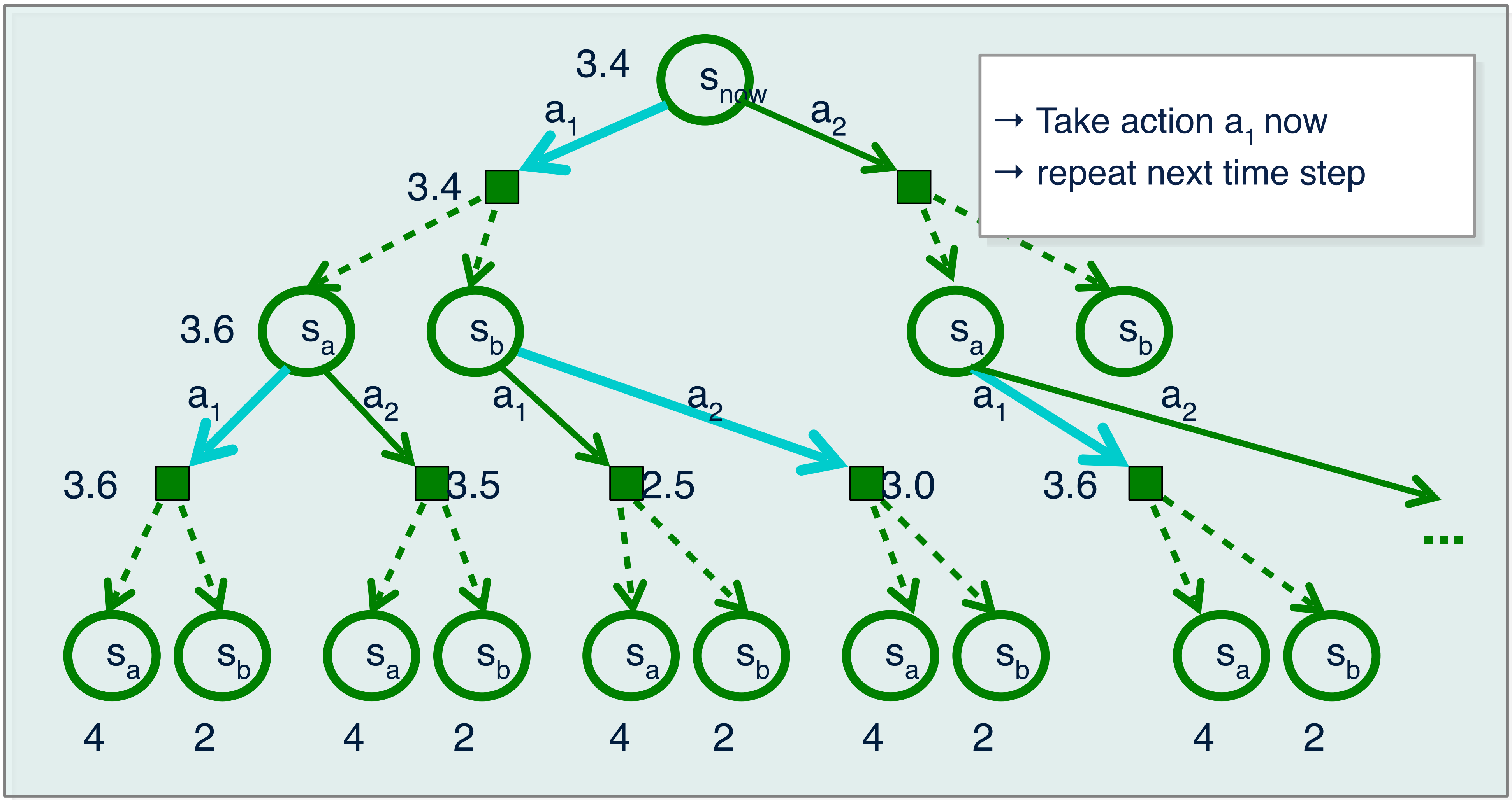
Forward Search



Forward Search



Forward Search



Forward Search: Limitations

- OK, so that seems nice enough...?
- Problem?

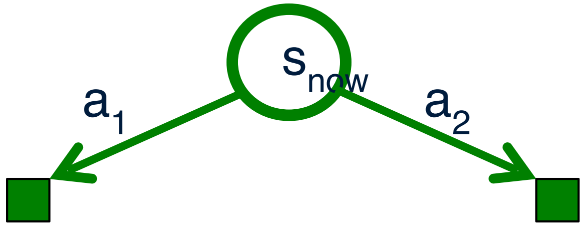
Forward Search: Limitations

- OK, so that seems nice enough...?
- Problem: trees get huge...!
  → not practical

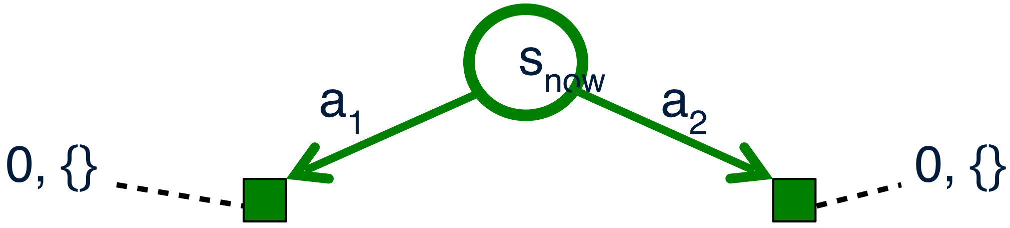
Monte Carlo Tree Search (MCTS)

- MCTS provides leverage by:
  - **incrementally** constructing a **sampled version** of the tree
  - focusing on **promising regions**

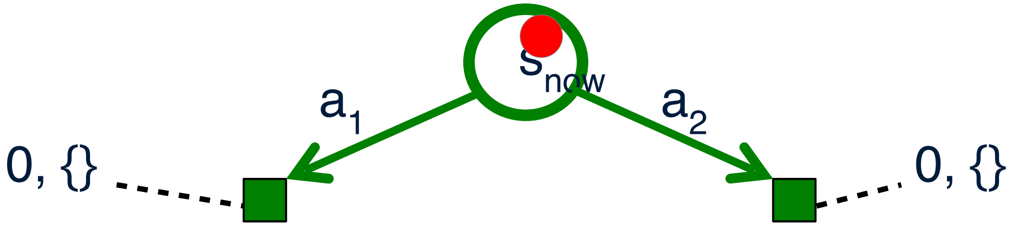
MCTS – Example



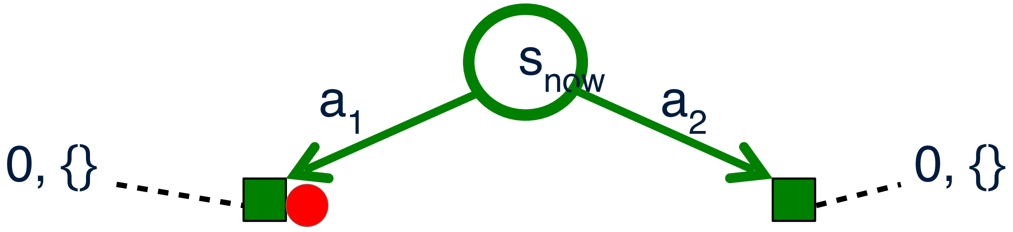
MCTS – Example



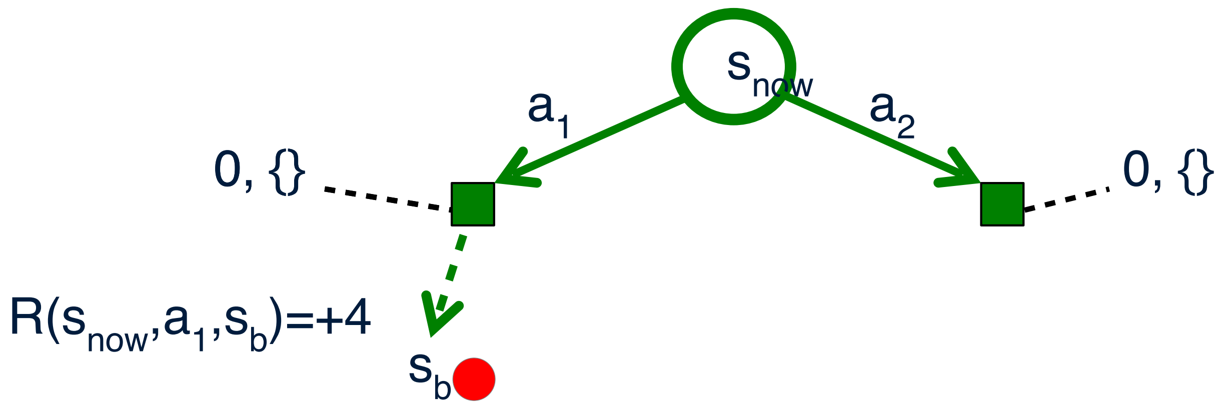
MCTS – Example



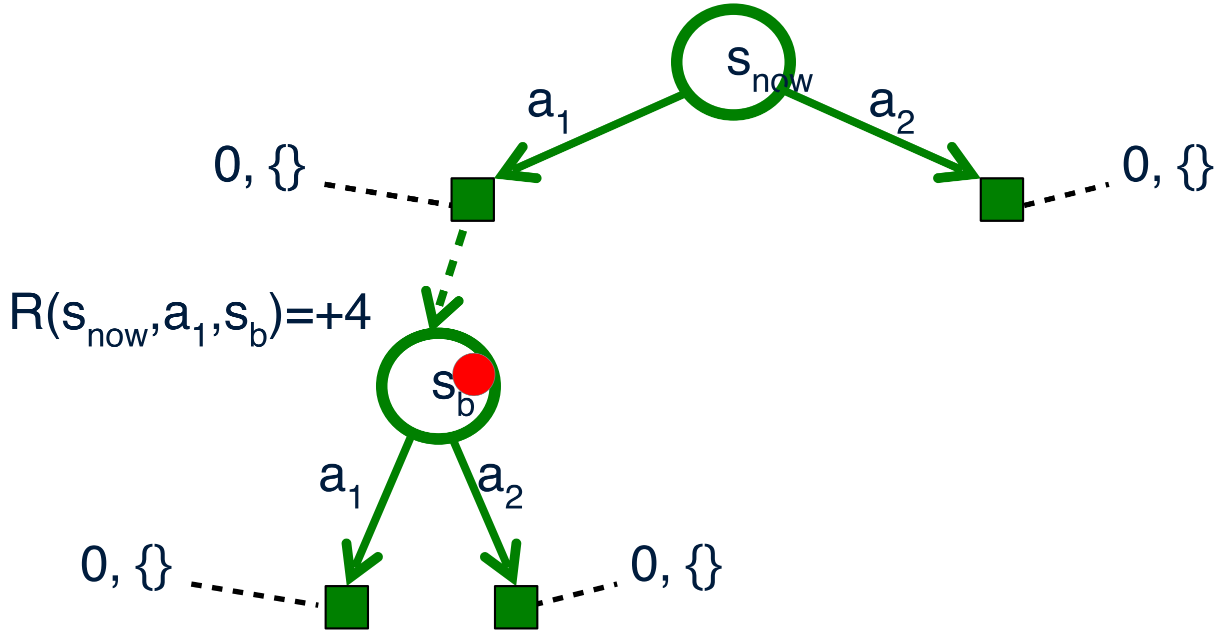
MCTS – Example



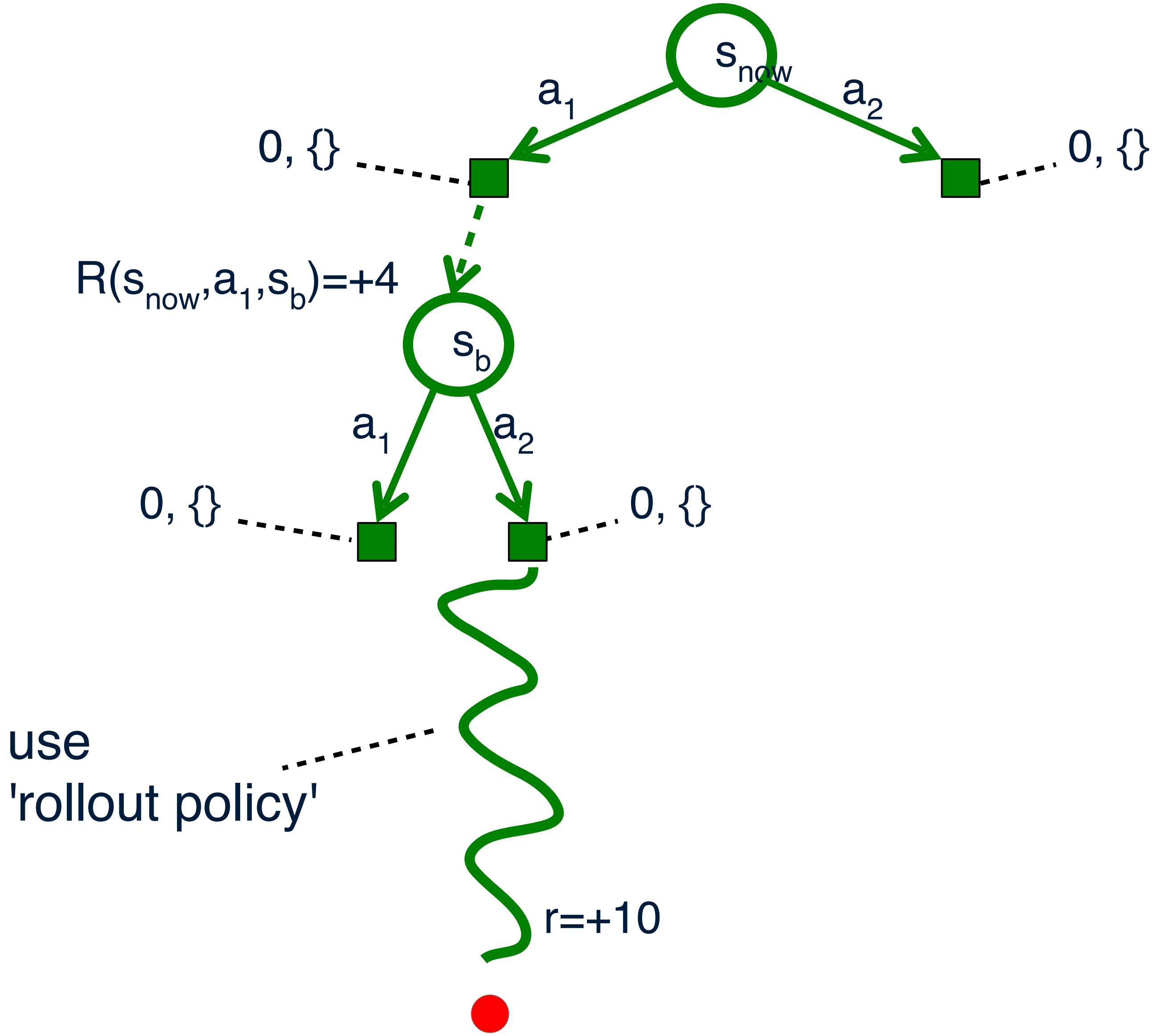
MCTS – Example



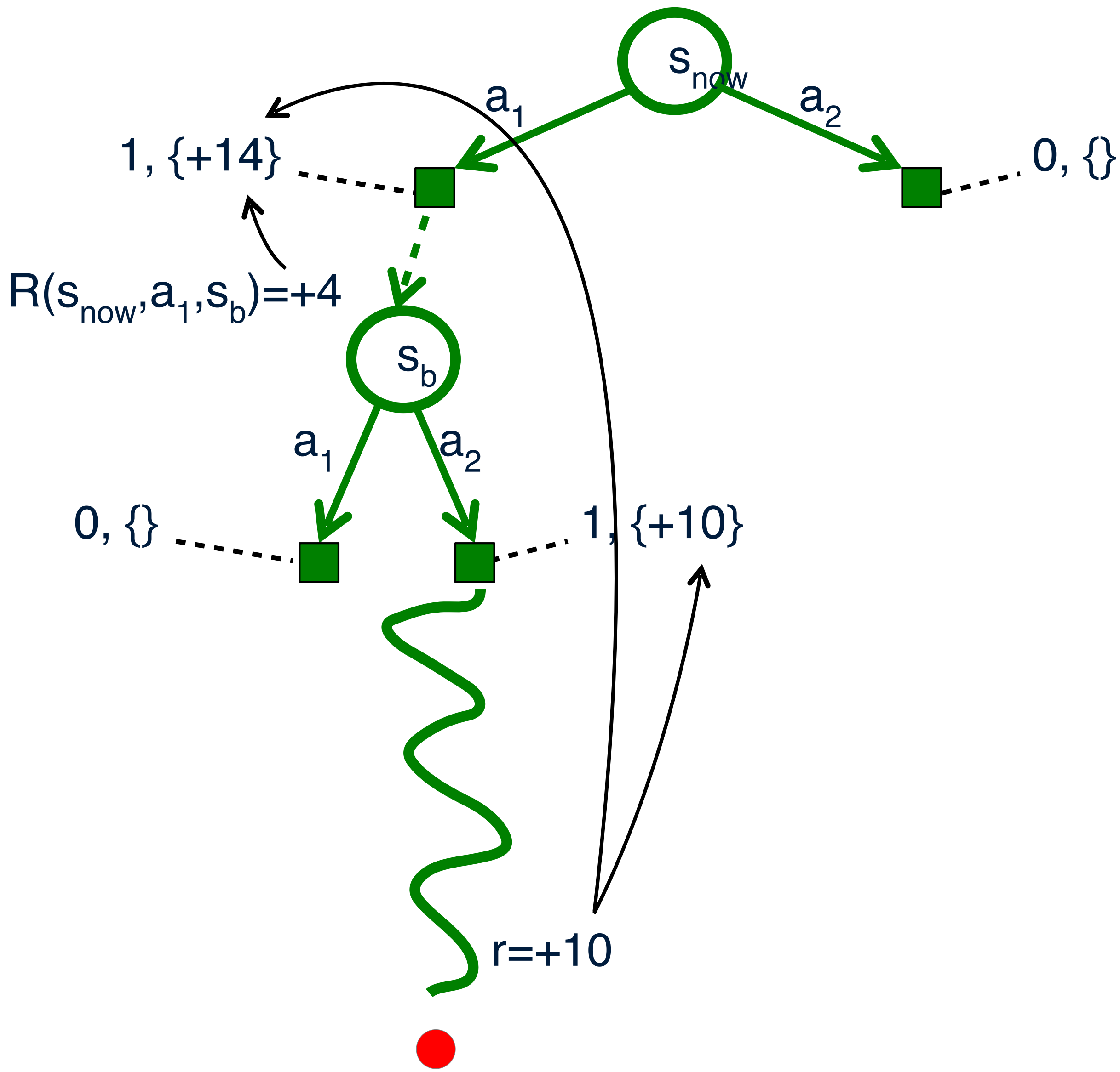
MCTS – Example



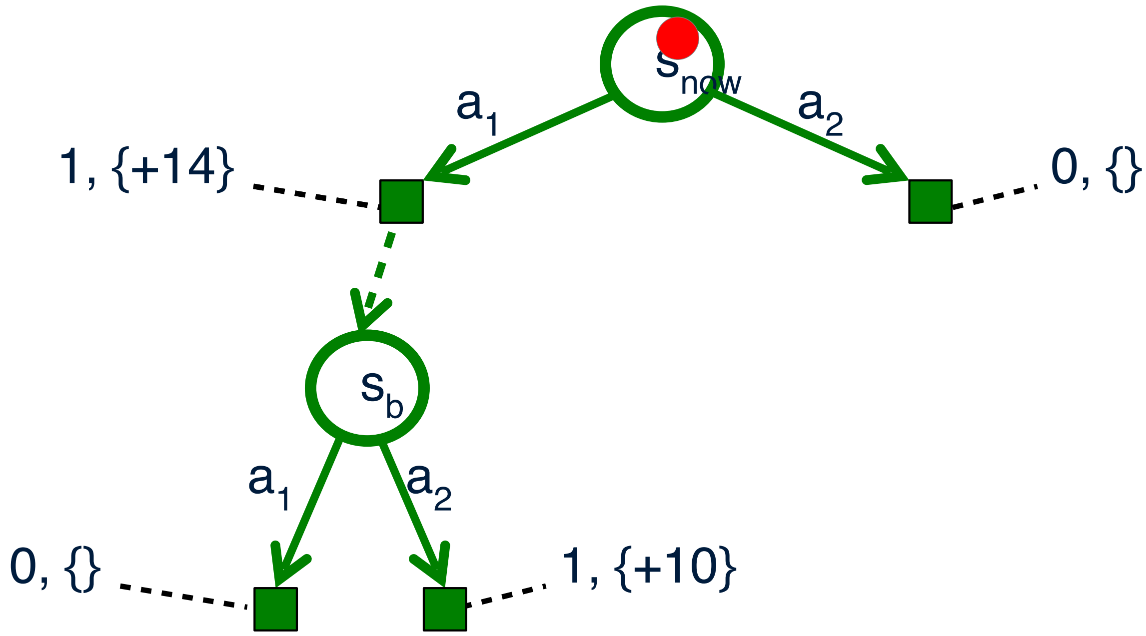
MCTS – Example



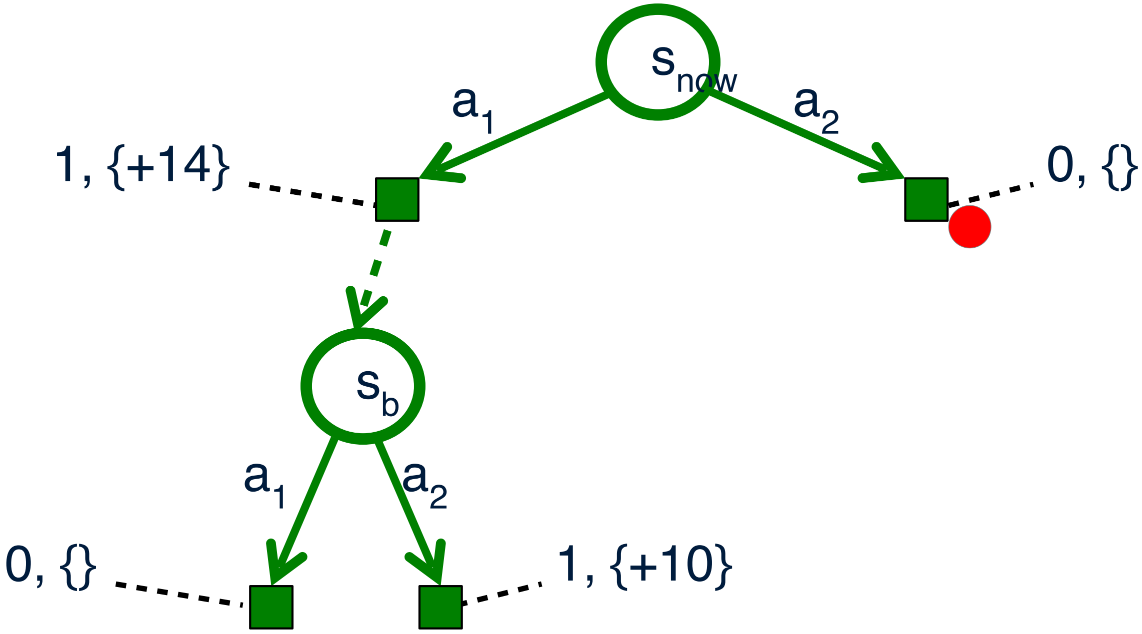
MCTS – Example



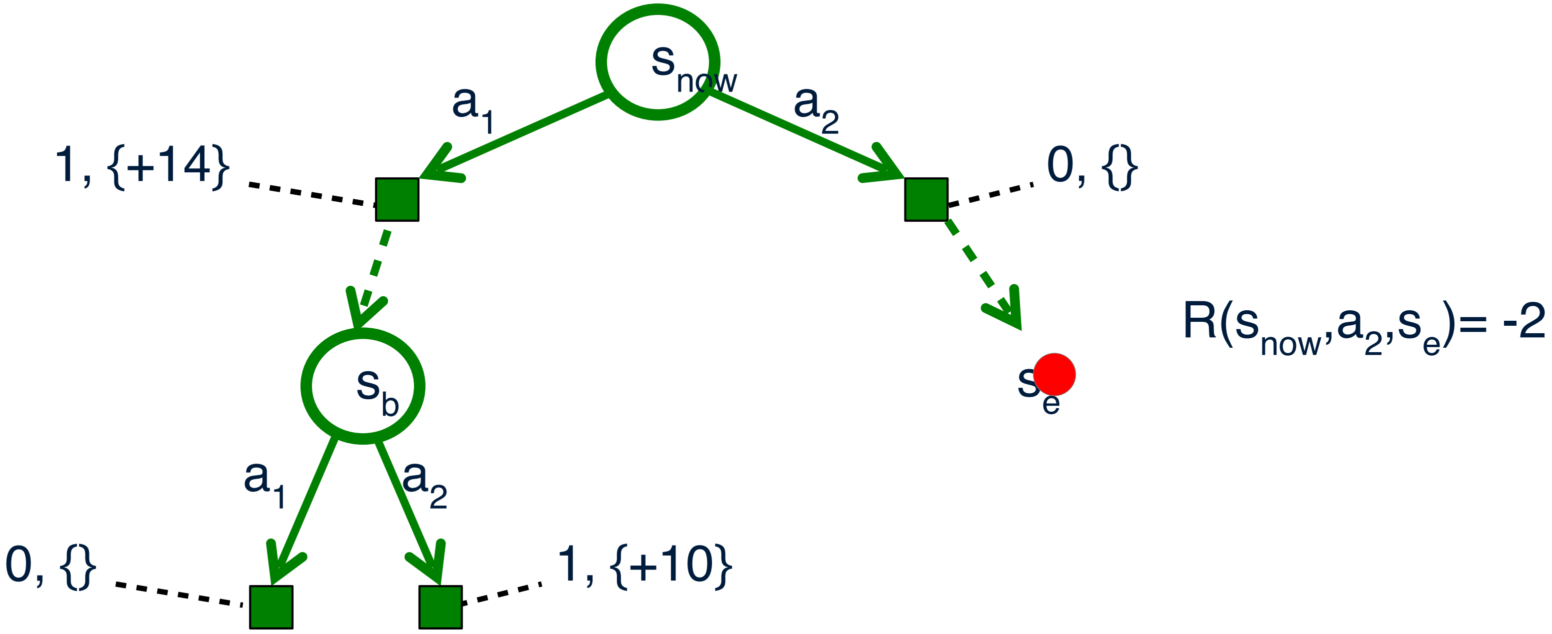
MCTS – Example



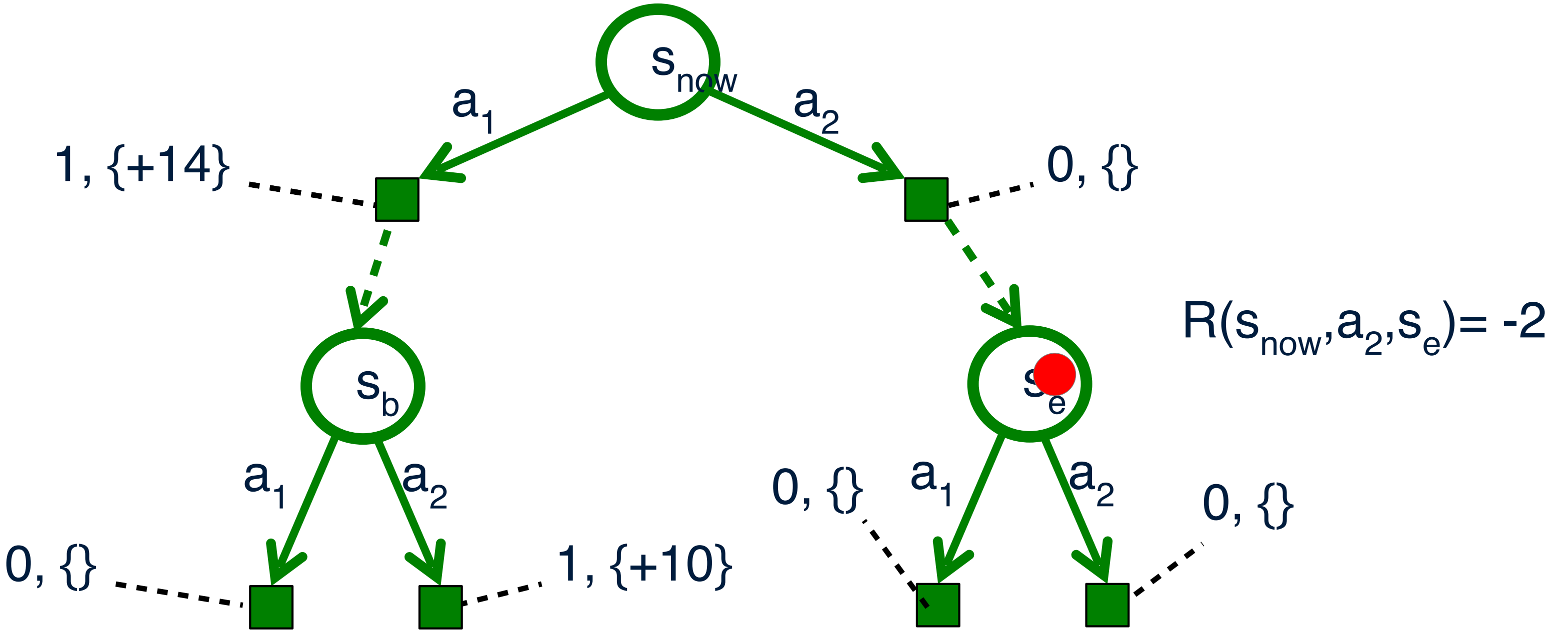
MCTS – Example



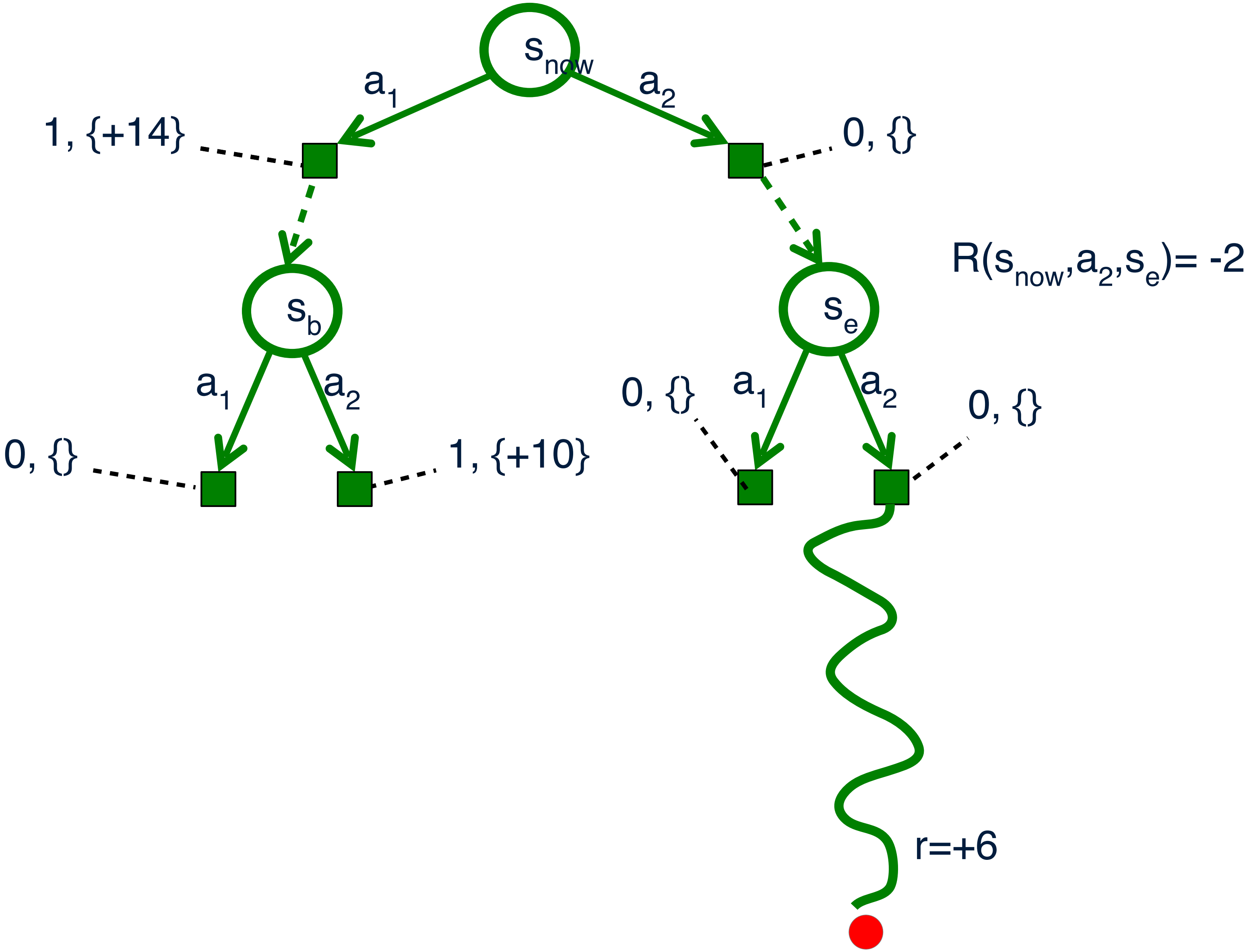
MCTS – Example



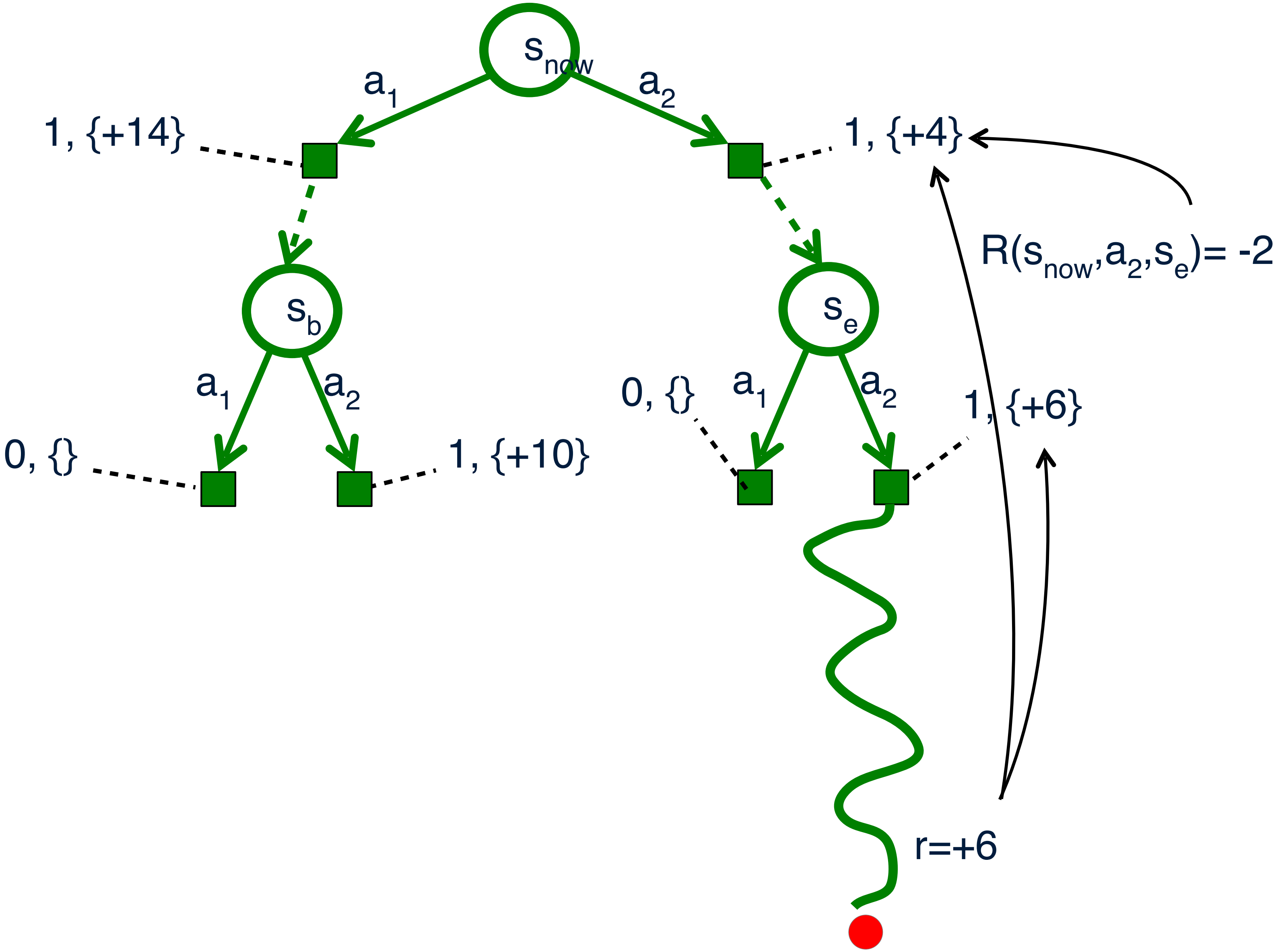
MCTS – Example



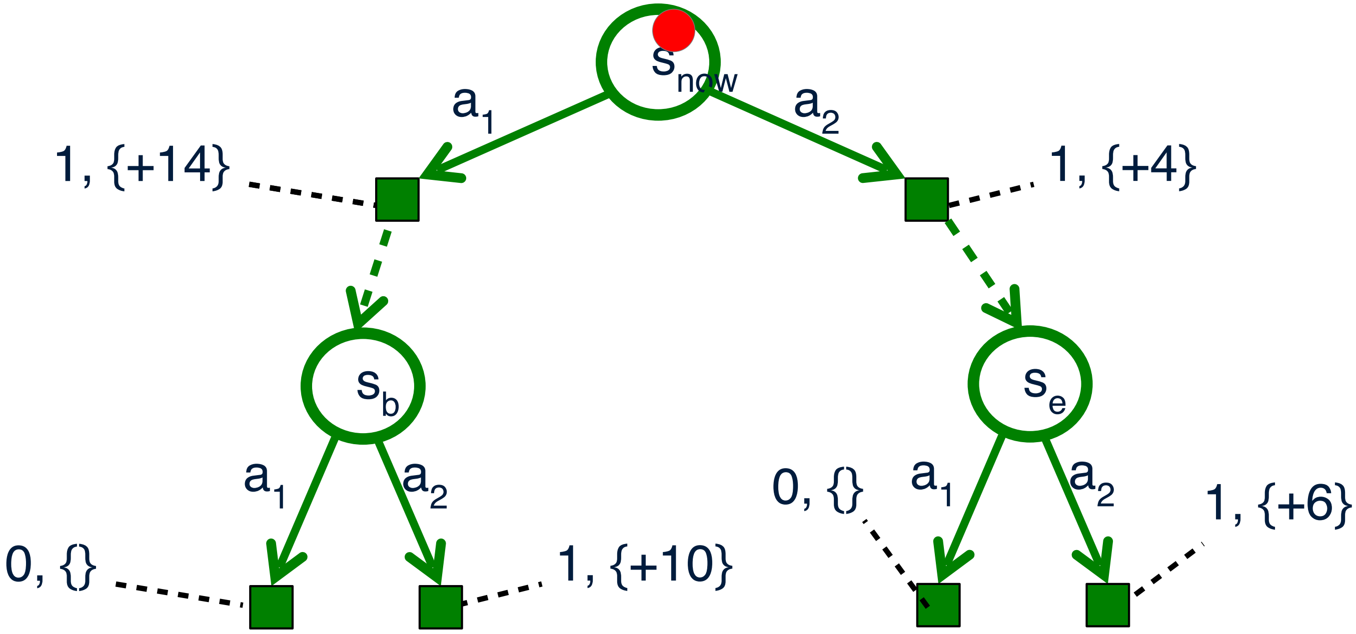
MCTS – Example



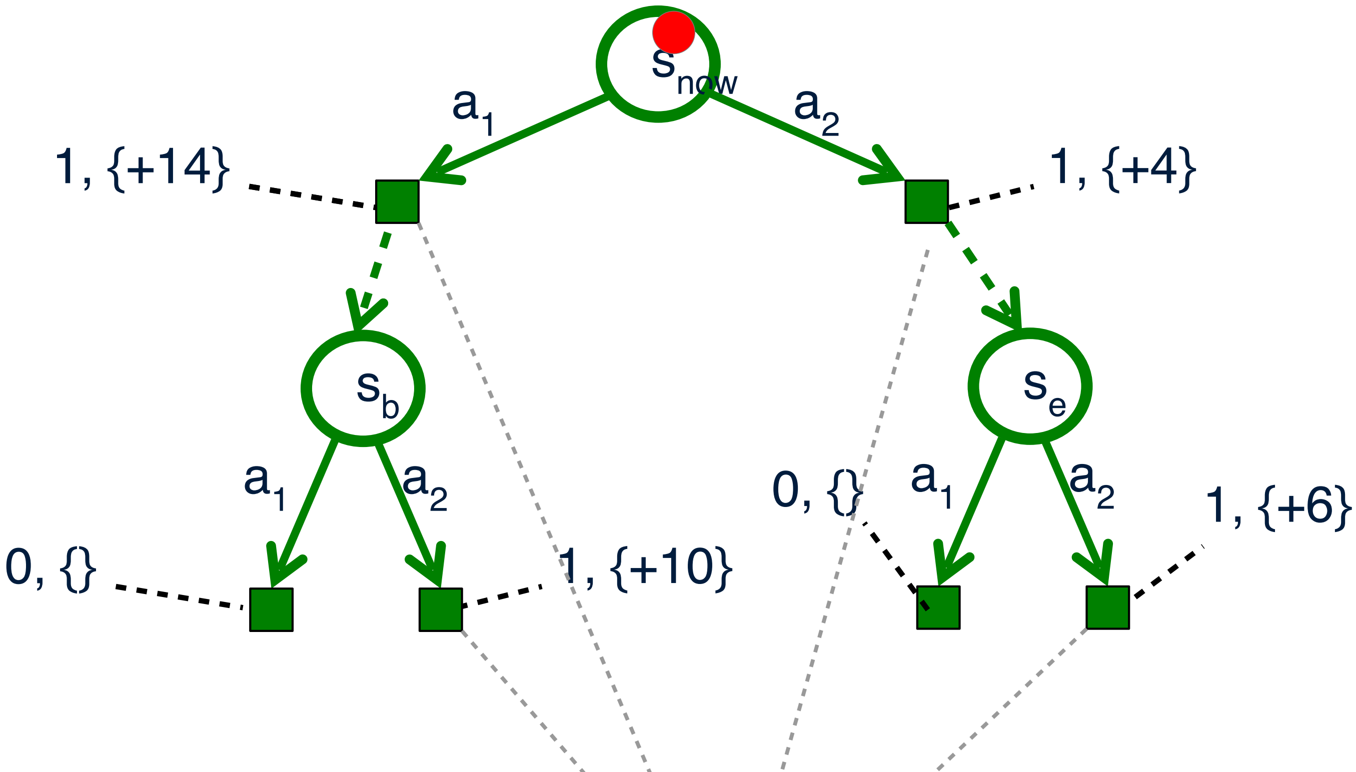
MCTS – Example



MCTS – Example

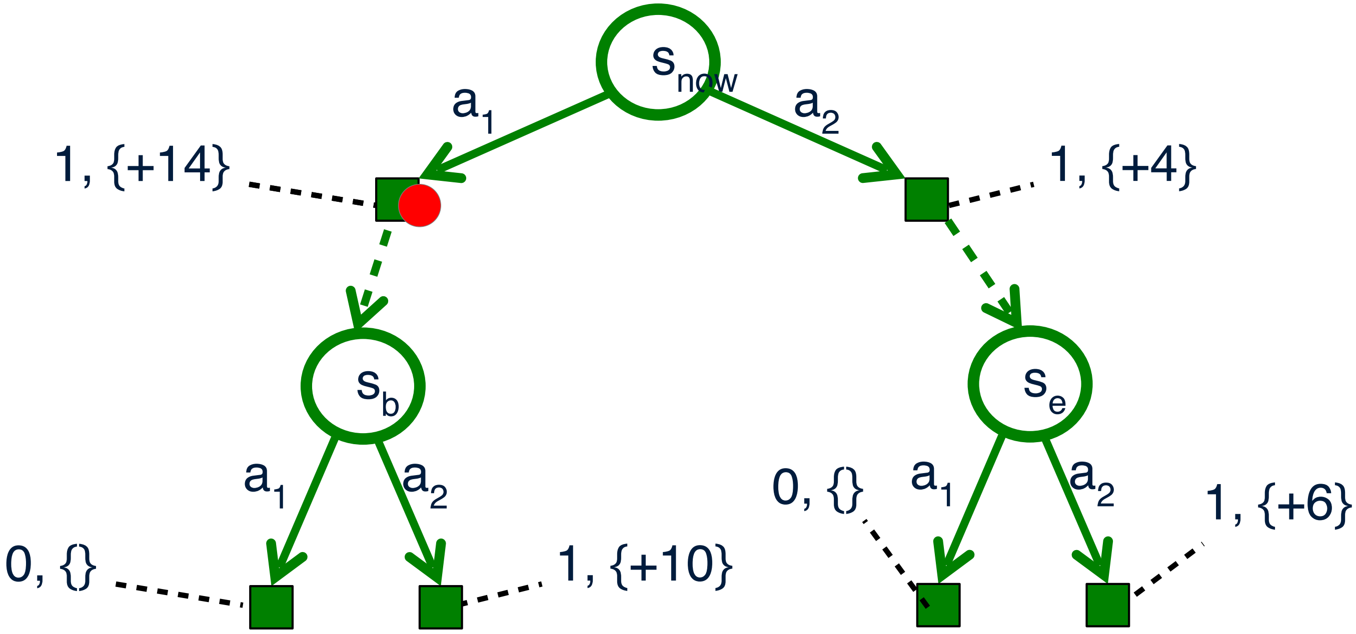


MCTS – Example

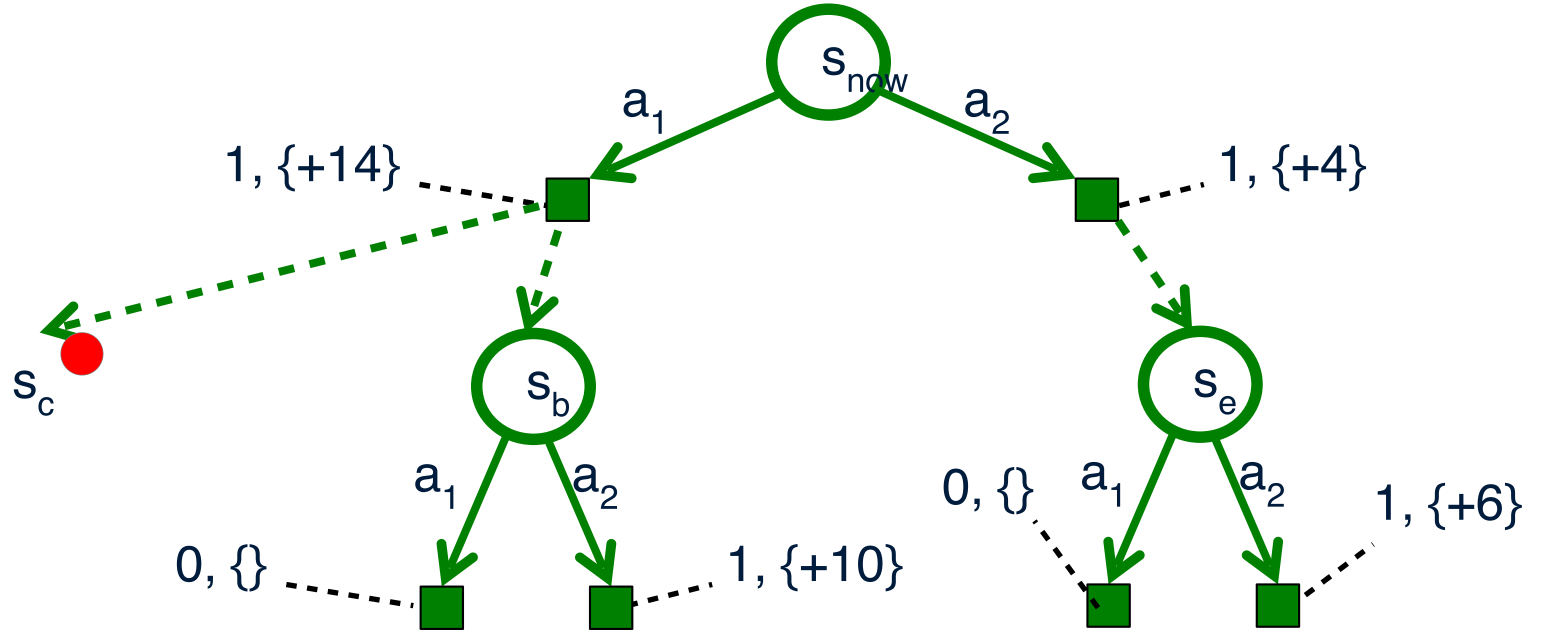


NOTE: the statistics maintained, represent an estimate of **Q(s,a)**

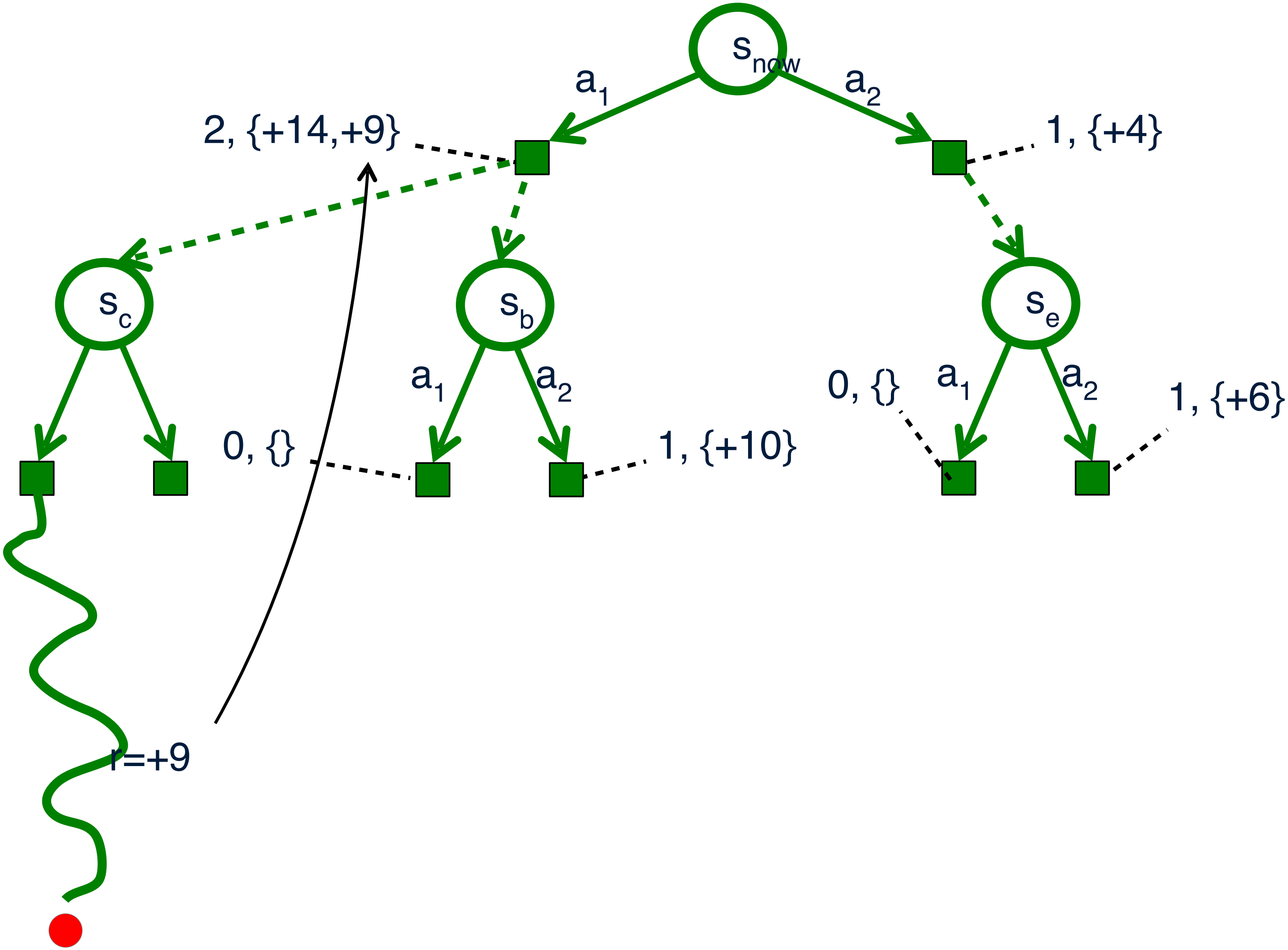
MCTS – Example



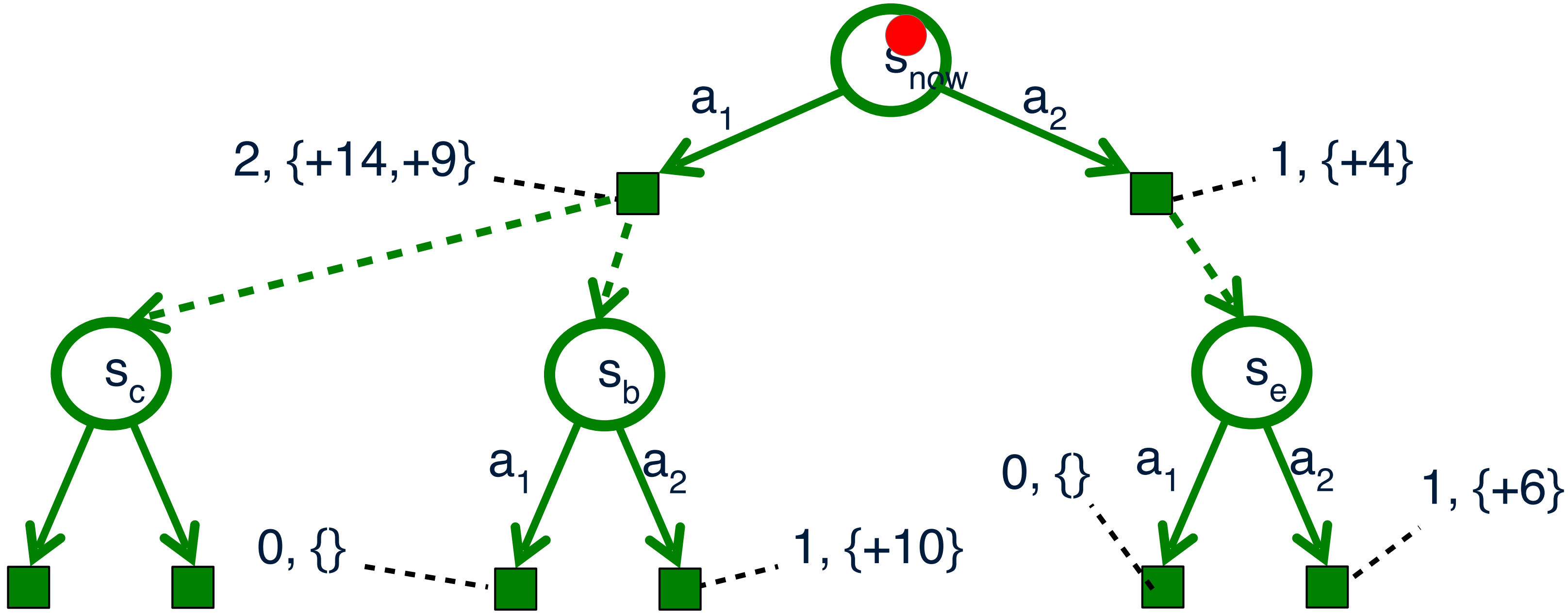
MCTS – Example



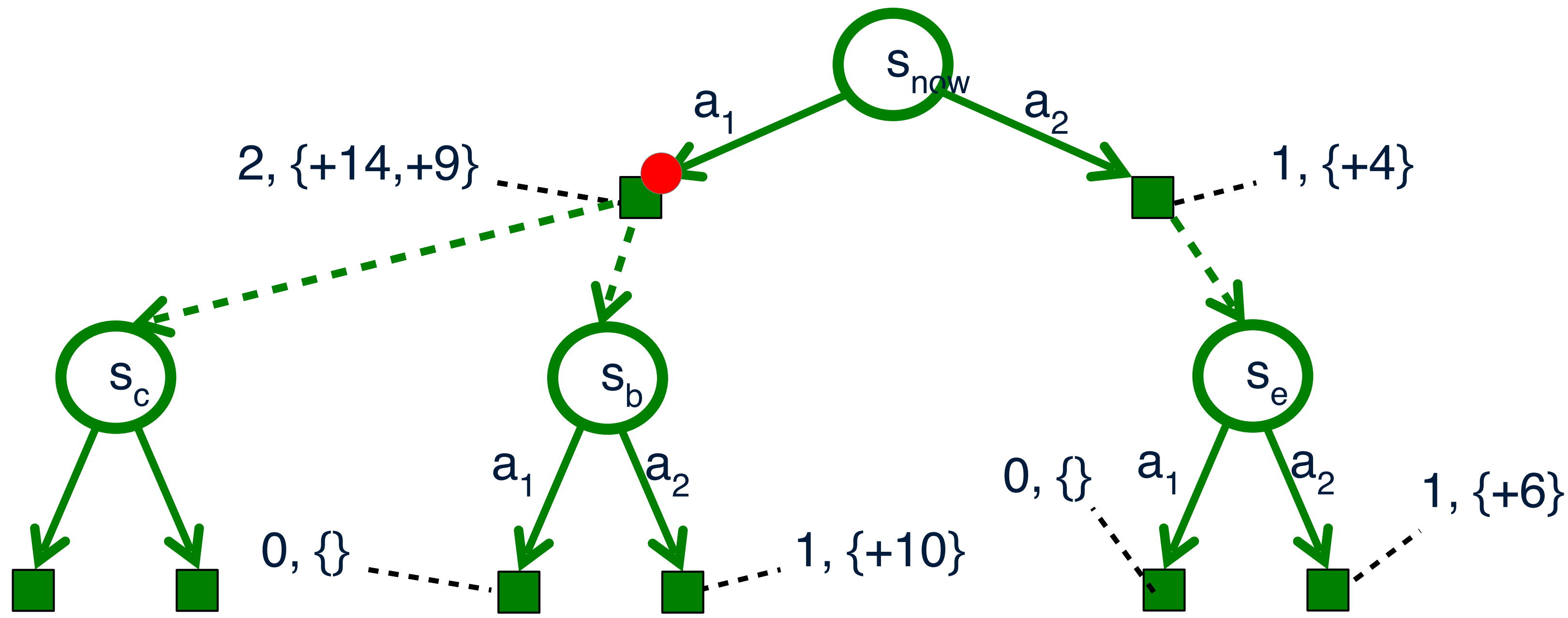
MCTS – Example



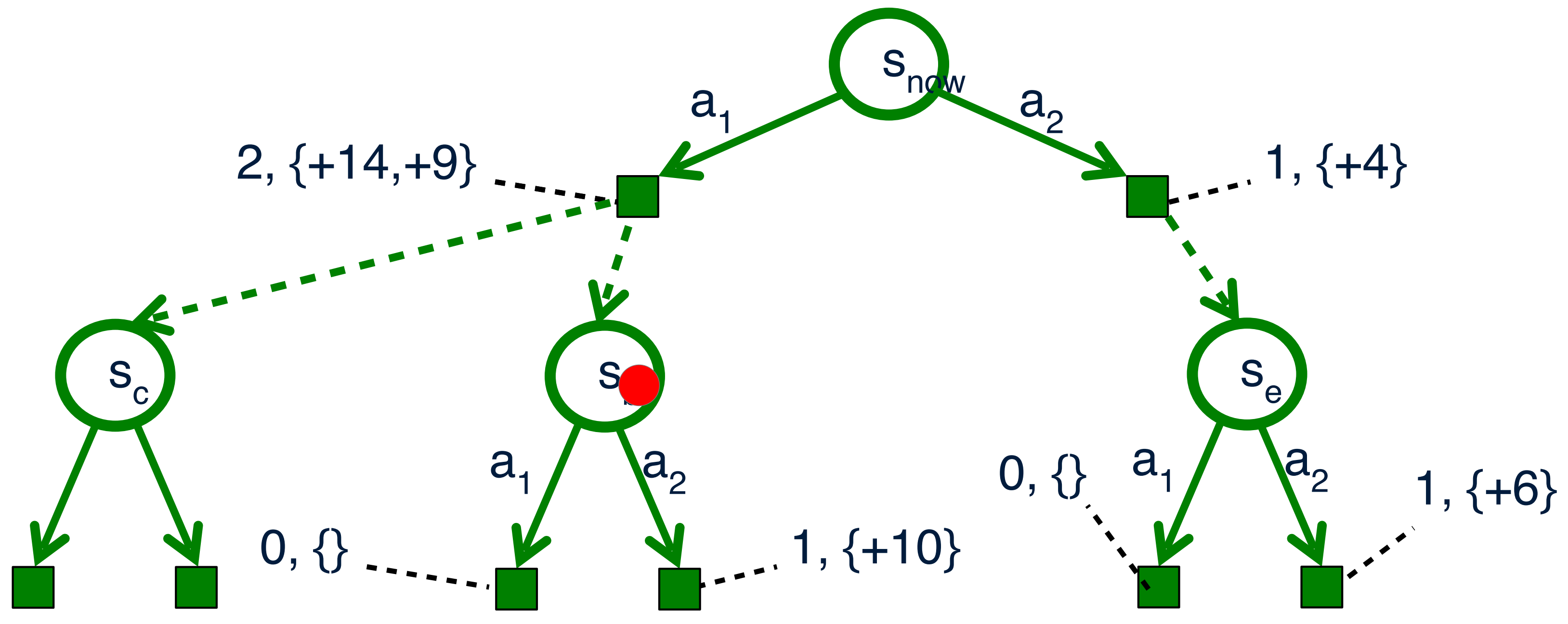
MCTS – Example



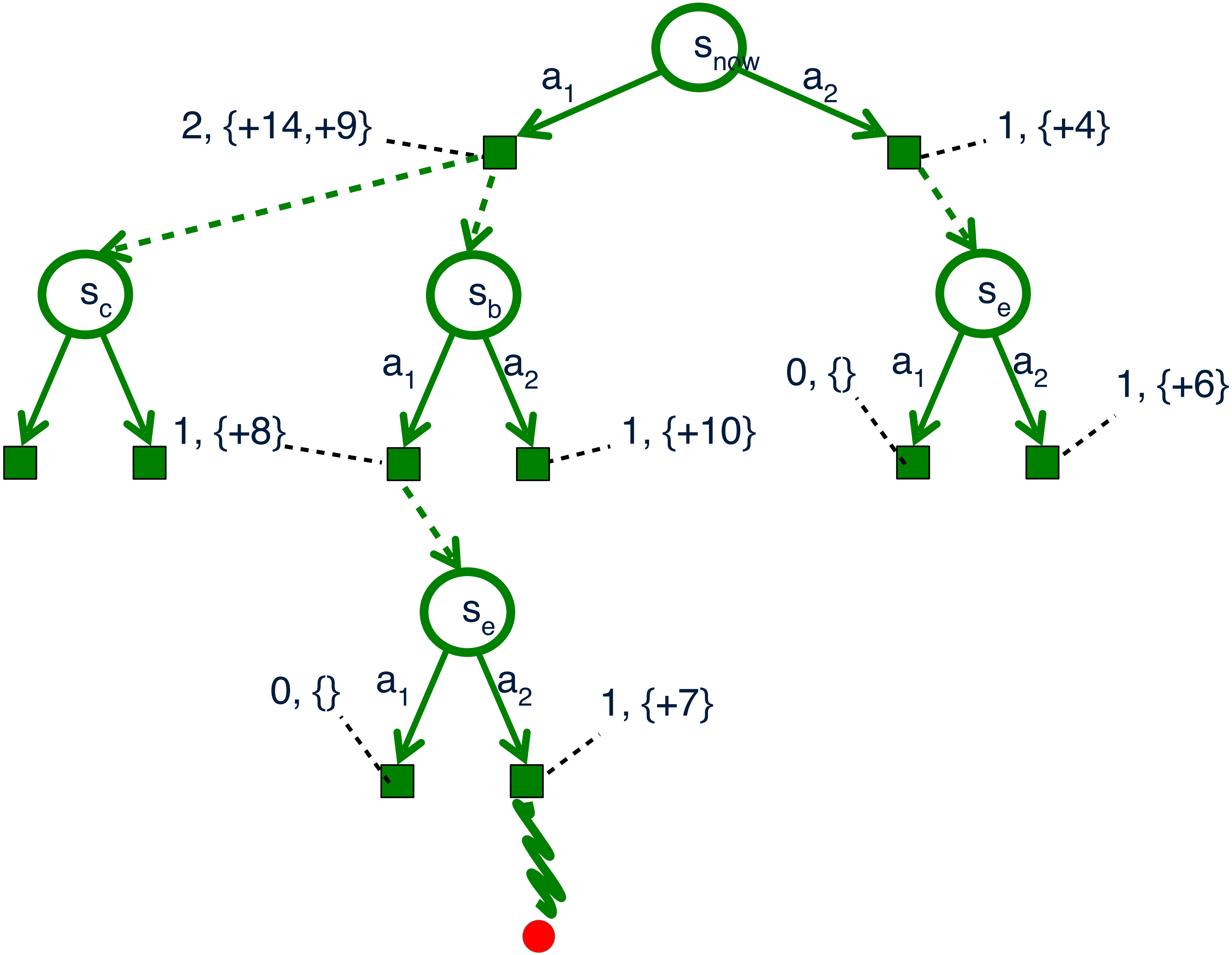
MCTS – Example



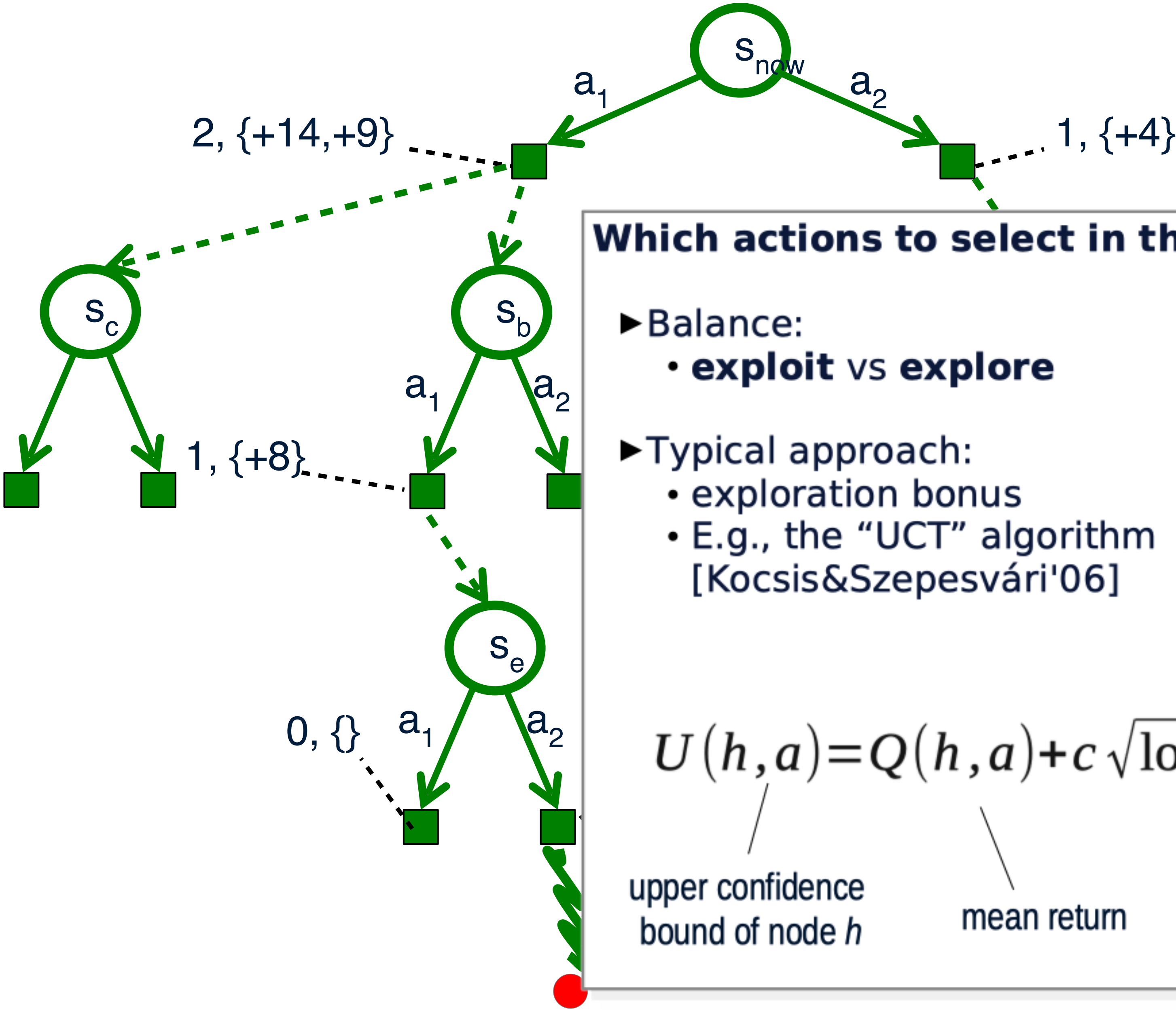
MCTS – Example



MCTS – Example



MCTS – Example

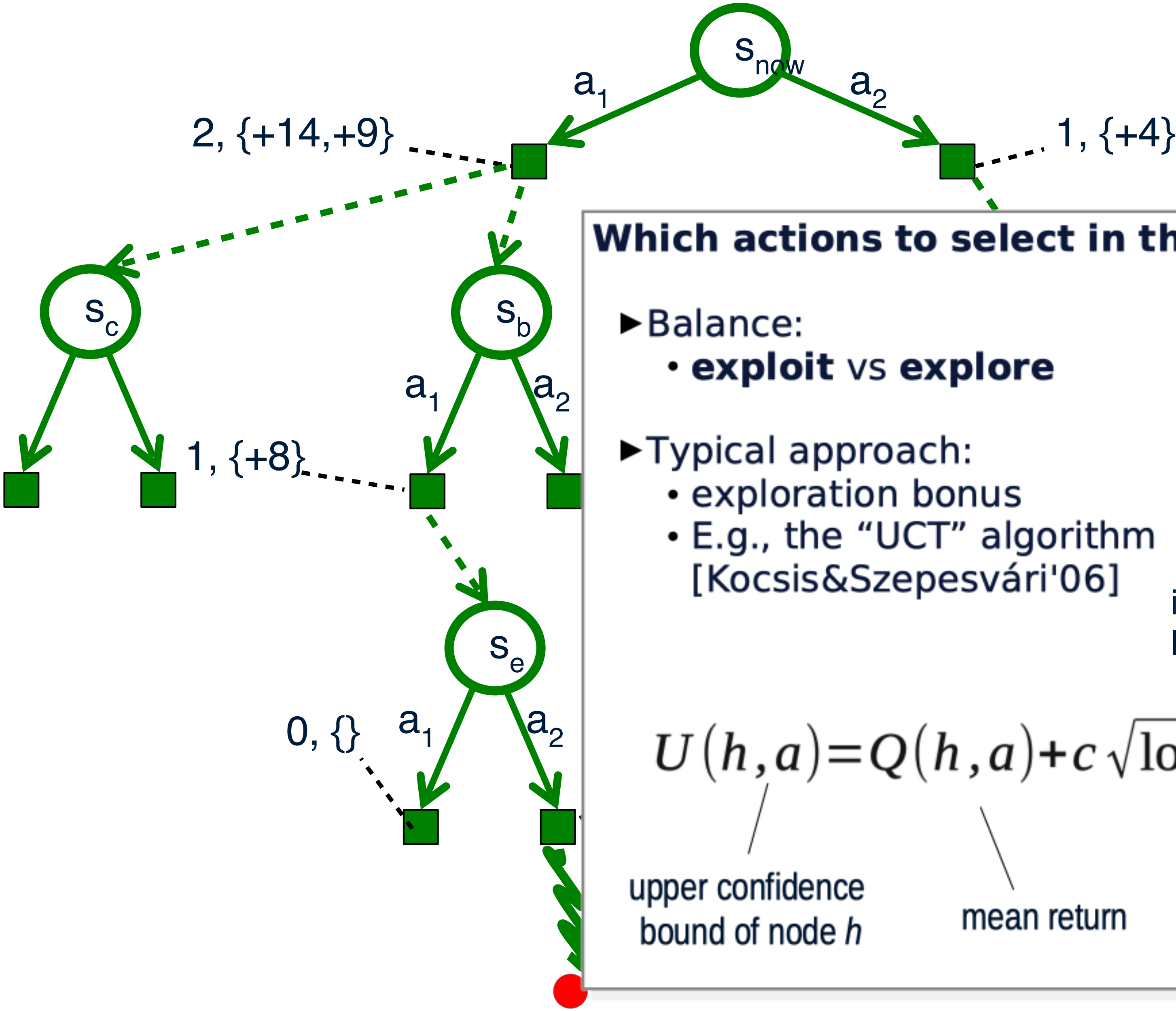


Which actions to select in the tree?

- Balance:
  - **exploit** vs **explore**
- Typical approach:
  - exploration bonus
  - E.g., the "UCT" algorithm [Kocsis&Szepesvári'06]

$$\underbrace{U(h,a)}_{\text{upper confidence bound of node } h}=\underbrace{Q(h,a)}_{\text{mean return}}+\underbrace{c\sqrt{\log(N_h+1)/N_a}}_{\text{exploration bonus}}$$

MCTS – Example



Which actions to select in the tree?

- Balance:
  - **exploit** vs **explore**
- Typical approach:
  - exploration bonus
  - E.g., the “UCT” algorithm [Kocsis&Szepesvári'06]

if *a* tried more often → less bonus

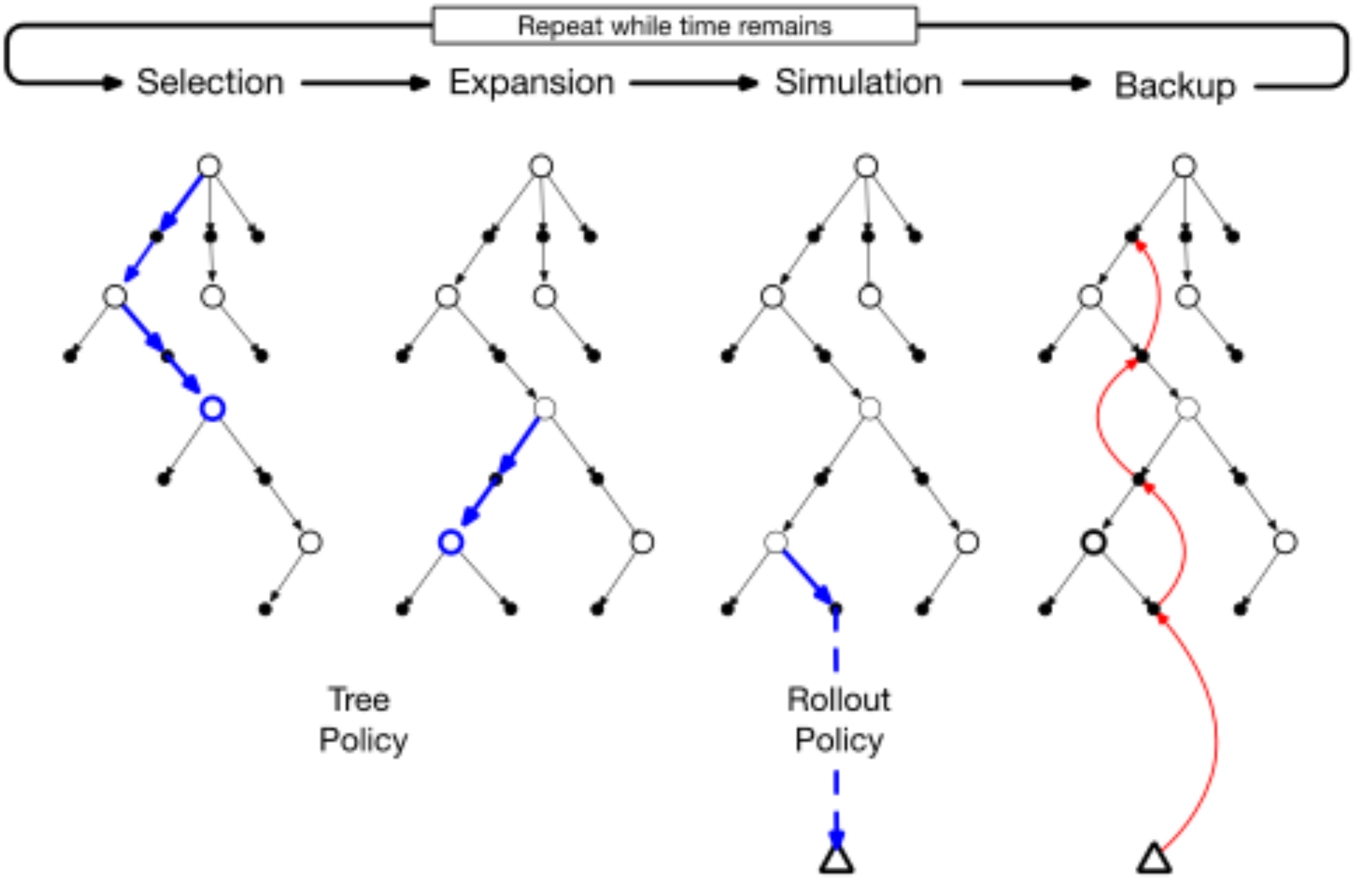
$$U(h,a) = Q(h,a) + c\sqrt{\log(N_h+1)/N_a}$$

upper confidence bound of node *h*

mean return

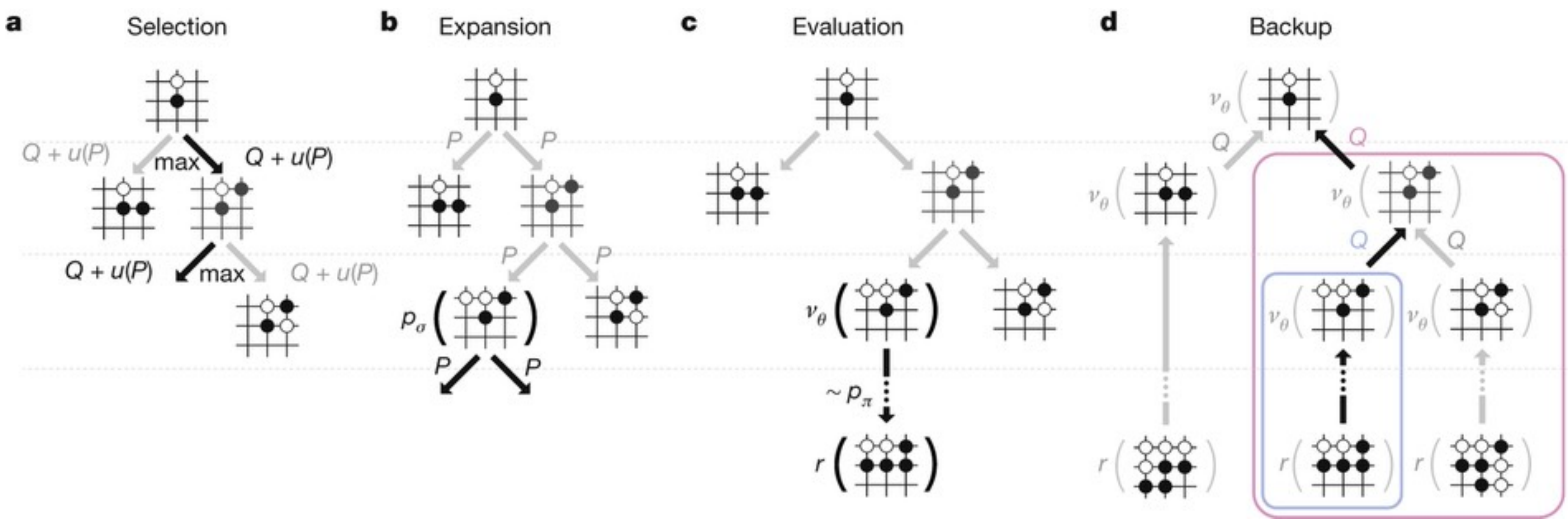
exploration bonus

Phases of MCTS



AlphaGo

- Uses UCT with neural net to approximate opponent choices and state values



Monte Carlo Evaluation

Algorithm 4.11 Monte Carlo policy evaluation

Depth

function MonteCarloPolicyEvaluation(λ, d)

  for $i \leftarrow 1$ **to** n

    $s \sim b$ Initial state distribution

    $u_i \leftarrow$ Rollout(s, d, π_λ) Rollout policy

  return $\frac{1}{n}\sum_{i=1}^{n} u_i$

Algorithm 4.10 Rollout evaluation

function Rollout(s, d, π_0)

  if $d = 0$

    return 0

  $a \sim \pi_0(s)$

  $(s', r) \sim G(s, a)$ Simulate

  return $r + \gamma$ Rollout($s', d - 1, \pi_0$)

- Estimate value of a policy by sampling from a simulator

Monte Carlo Tree Search

Algorithm 4.9 Monte Carlo tree search

function $\textsc{SelectAction}(s, d)$
  loop
    $\textsc{Simulate}(s, d, \pi_0)$
  return $\arg\max_a Q(s, a)$
function $\textsc{Simulate}(s, d, \pi_0)$
  if $d = 0$
    return 0
  if $s \notin T$
    for $a \in A(s)$
      $(N(s, a), Q(s, a)) \leftarrow (N_0(s, a), Q_0(s, a))$
    $T = T \cup \{s\}$
    return $\textsc{Rollout}(s, d, \pi_0)$
  $a \leftarrow \arg\max_a Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$
  $(s', r) \sim G(s, a)$
  $q \leftarrow r + \gamma\,\textsc{Simulate}(s', d-1, \pi_0)$
  $N(s, a) \leftarrow N(s, a) + 1$
  $Q(s, a) \leftarrow Q(s, a) + \frac{q - Q(s,a)}{N(s,a)}$
  return q

UCT (Upper Confidence Bounds for Trees)

Just like our bandit case!

Monte Carlo Tree Search

Algorithm 4.9 Monte Carlo tree search

function $\textsc{SelectAction}(s, d)$
    loop
        $\textsc{Simulate}(s, d, \pi_0)$
    return $\arg\max_a Q(s, a)$

Don't use UCB to choose actions in the real world

function $\textsc{Simulate}(s, d, \pi_0)$
    if $d = 0$
        return 0
    if $s \notin T$

State isn't in tree (so initialize stats)

        for $a \in A(s)$
            $(N(s,a), Q(s,a)) \leftarrow (N_0(s,a), Q_0(s,a))$
        $T = T \cup \{s\}$
        return $\textsc{Rollout}(s, d, \pi_0)$

Add to tree and then do rollout

    $a \leftarrow \arg\max_a Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$

UCT (Upper Confidence Bounds for Trees)

Just like our bandit case!

    $(s', r) \sim G(s, a)$
    $q \leftarrow r + \gamma \textsc{Simulate}(s', d - 1, \pi_0)$
    $N(s, a) \leftarrow N(s, a) + 1$
    $Q(s, a) \leftarrow Q(s, a) + \frac{q - Q(s,a)}{N(s,a)}$
    return q

UCT continued

- Search/Selection (within the tree, T)
  - Execute action that maximizes $$Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$$
  - Update the value $Q(s,a)$ and counts $N(s)$ and $N(s,a)$
  - c is a exploration constant
- Expansion (outside of the tree, T)
  - Create a new node for the state
  - Initialize $Q(s,a)$ and $N(s,a)$ (usually to 0) for each action
- Rollout and Backup (outside of the tree, T)
  - Only expand once and then use a rollout policy to select actions (e.g., random policy)
  - Add the rewards gained during the rollout with those in the tree:

$$r + \gamma \textsc{Rollout}(s', d-1, \pi_0)$$

UCT continued

- Continue UCT until some termination condition
- Guarantees?

UCT continued

- Continue UCT until some termination condition (usually a fixed number of samples)
- MCTS (and UCT) will approach the true Q-values in the limit (for a fixed horizon)
  - You can see this by considering what happens when the full tree is generated
  - This doesn’t depend on rollout policy, but does require exploration of actions
  - In practice no one runs MCTS until optimality, so it does matter how rollout/evaluation and exploration is done

MCTS Pros/Cons

- Pros:
  - rapidly zooms in on promising regions
  - can be used to improve policies
  - basis of many successful application

- Limitations:
  - needle in the hay-stack problems
  - problems with high branching factor
  - both cause poor approximations in limited time

Summary

- 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
    - prioritized sweeping
    - sample backups
    - trajectory sampling: backup along trajectories
    - heuristic search/MCTS
- Size of backups: full/sample; deep/shallow

Next time

- Linear function approximation (scaling beyond tabular methods)!
- Read SB 9.1--9.5, 9.8