Model-Based RL

Reinforcement Learning Summer School

Martha White
University of Alberta and AMII

Comments for the lecture

- Please ask questions (this is a summer school)
- I will pause a few times and get you to answer questions/exercises
- Outcomes: you will
  - understand how models can be used to learn optimal values/policies
  - understand in-depth one strategy, called Dyna, for online setting
  - recognize some of the other ways models can be used

What is model-based RL?

What is model-based RL?

Could mean RL when **given** the model

What is model-based RL?

Could mean RL when **given** the model

Could mean RL with a **learned** model

What is model-based RL?

Could mean RL when **given** the model

Could mean RL with a **learned** model

What is model-based RL?

Could mean RL when **given** the model

Could mean RL with a **learned** model

High-level Outline

- Part 1: Learning the optimal policy given the model (offline)
- Part 2: Moving to learned models (online)
  - Particularly looking at a formalism called Dyna
- Part 3: A brief discussion about other ways to use models

High-level Outline

- **Part 1: Learning the optimal policy given the model (offline)**
- Part 2: Moving to learned models (online)
  - Particularly looking at a formalism called Dyna
- Part 3: A brief discussion about other ways to use models

Imagine we have the model

- Joint transition and reward dynamics

$$p(s', r \mid s, a)$$

- **Then**, we can learn **offline** without interacting with the world!

Bellman equations & Dynamic Programming to find the optimal policy

- We can directly solve for the (optimal) action-values, using Bellman equations

Bellman equations & Dynamic Programming to find the optimal policy

- We can directly solve for the (optimal) action-values, using Bellman equations

$$q_*(s,a) = \sum_{s'} \sum_{r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]$$

Bellman equations & Dynamic Programming to find the optimal policy

- We can directly solve for the (optimal) action-values, using Bellman equations

$$q_*(s,a) = \sum_{s'} \sum_{r} p(s',r \mid s,a) \left[r + \gamma \max_{a'} q_*(s',a') \right]$$

$$q_{k+1}(s,a) = \sum_{s'} \sum_{r} p(s',r \mid s,a) \left[r + \gamma \max_{a'} q_k(s',a') \right]$$

called Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize Q(s,a) = 0 for all s,a

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}, a \in \mathcal{A}$
| $\quad v \leftarrow Q(s,a)$
| $\quad Q(s,a) \leftarrow \Sigma_{s',r} p(s', r \mid s,a)[r + \gamma \max_{a'} Q(s',a')]$
| $\quad \Delta \leftarrow \max(\Delta, |v - Q(s,a)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a Q(s,a)$

High-level Outline

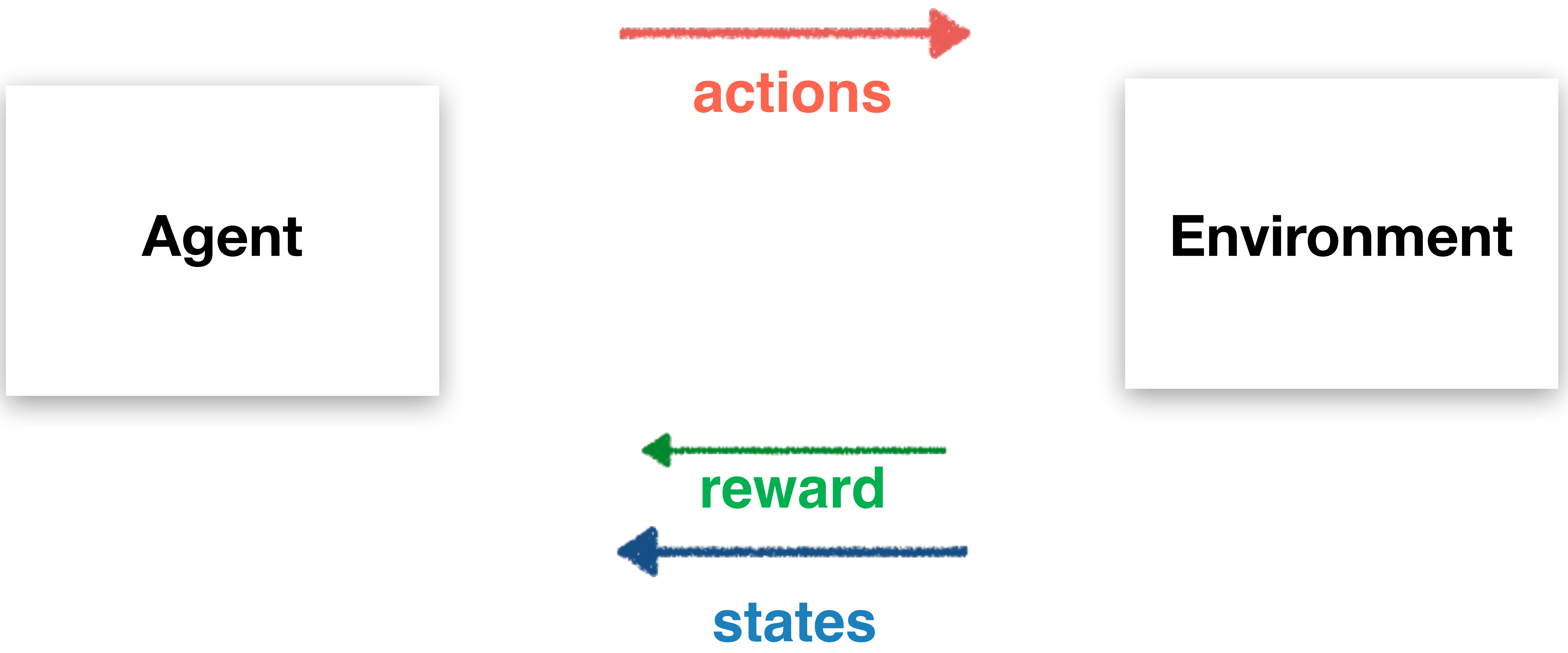
- Part 1: Learning the optimal policy given the model (offline)
- **Part 2: Moving to learned models (online)**
- Part 3: A brief discussion about other ways to use models

High-level Outline

- Part 1: Learning the optimal policy given the model (offline)
- **Part 2: Moving to learned models (online)**
- Part 3: A brief discussion about other ways to use models

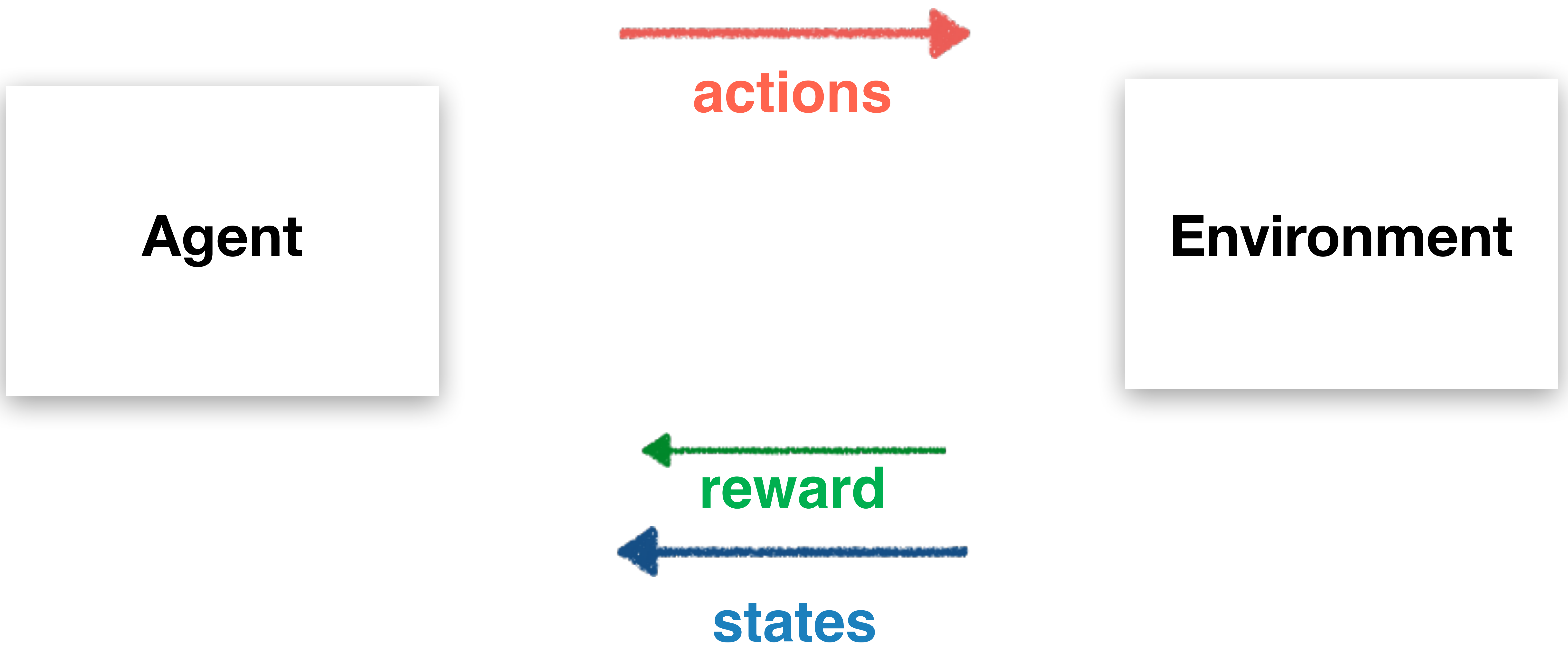
RL with learned models can use a similar approach to Dynamic Programming
but online

Online Reinforcement Learning



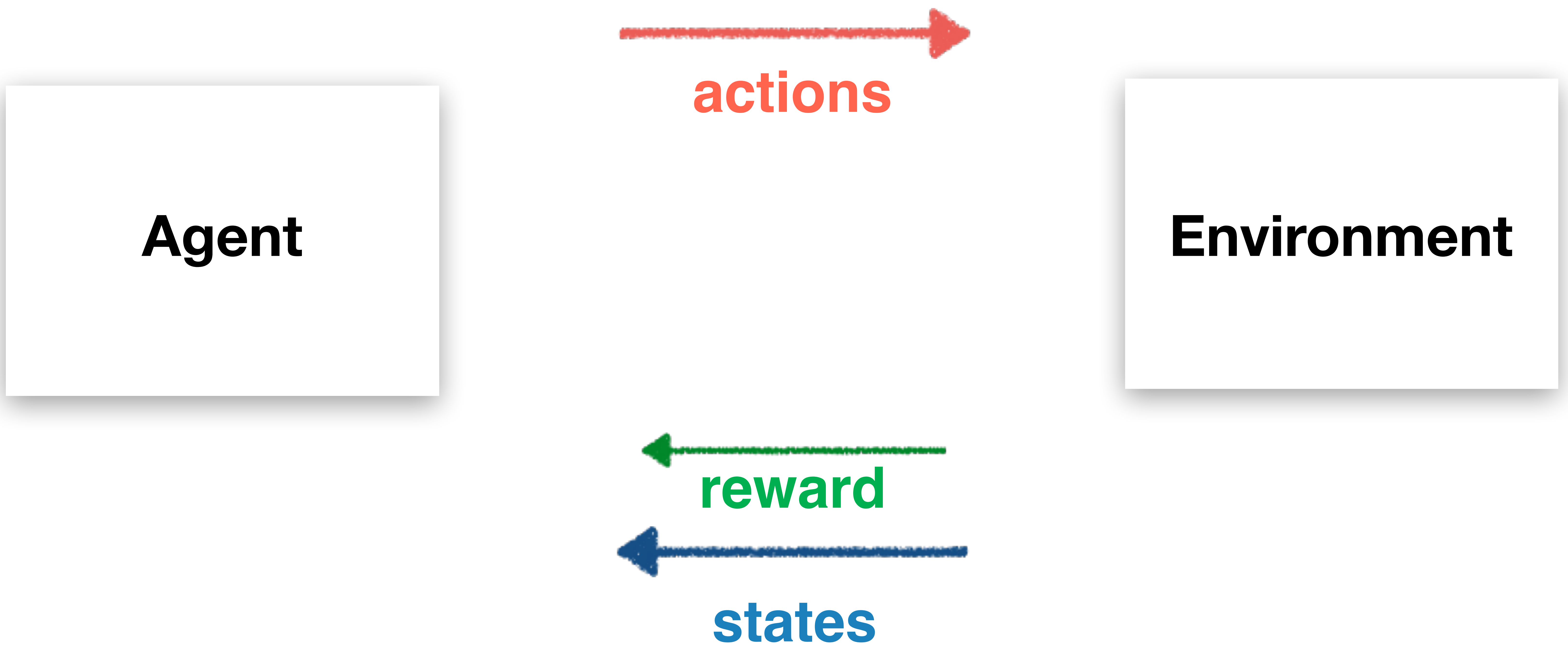
Online Reinforcement Learning

Online Reinforcement Learning



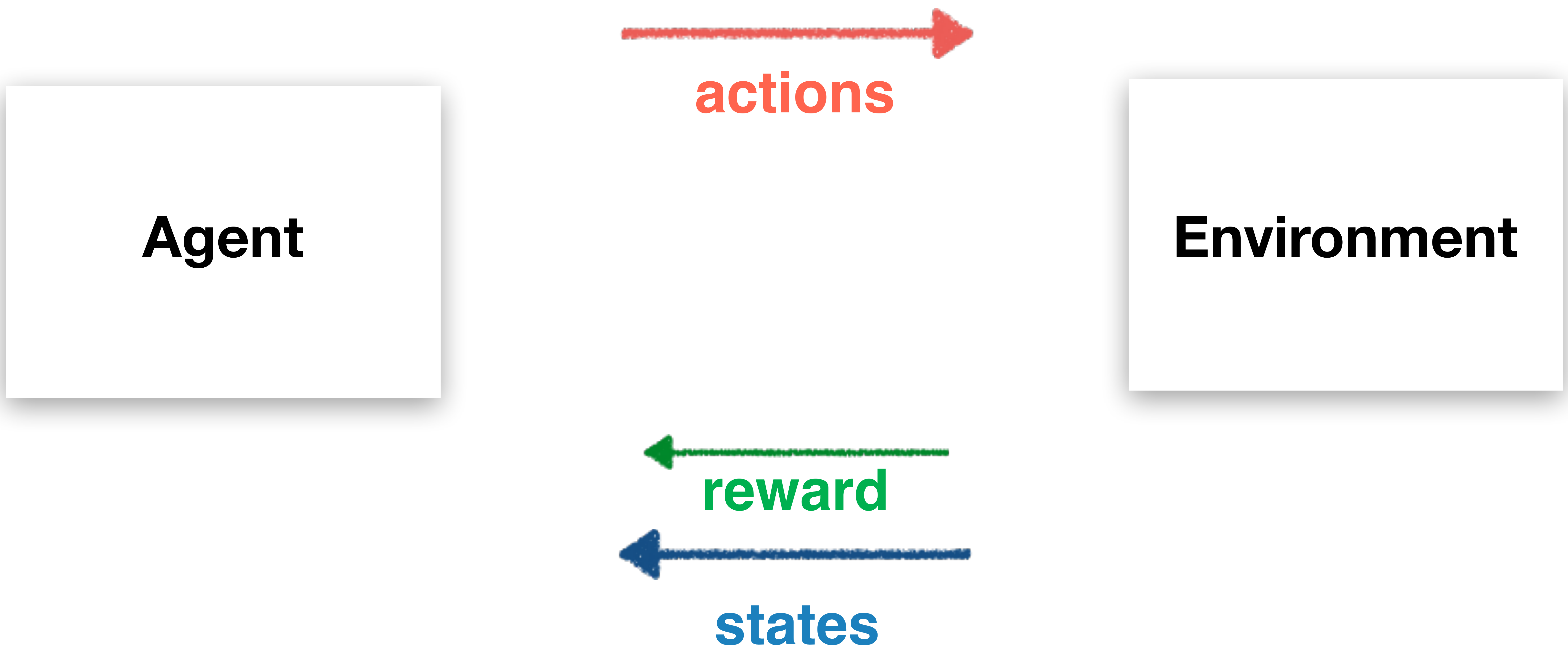
S_0 A_0

Online Reinforcement Learning



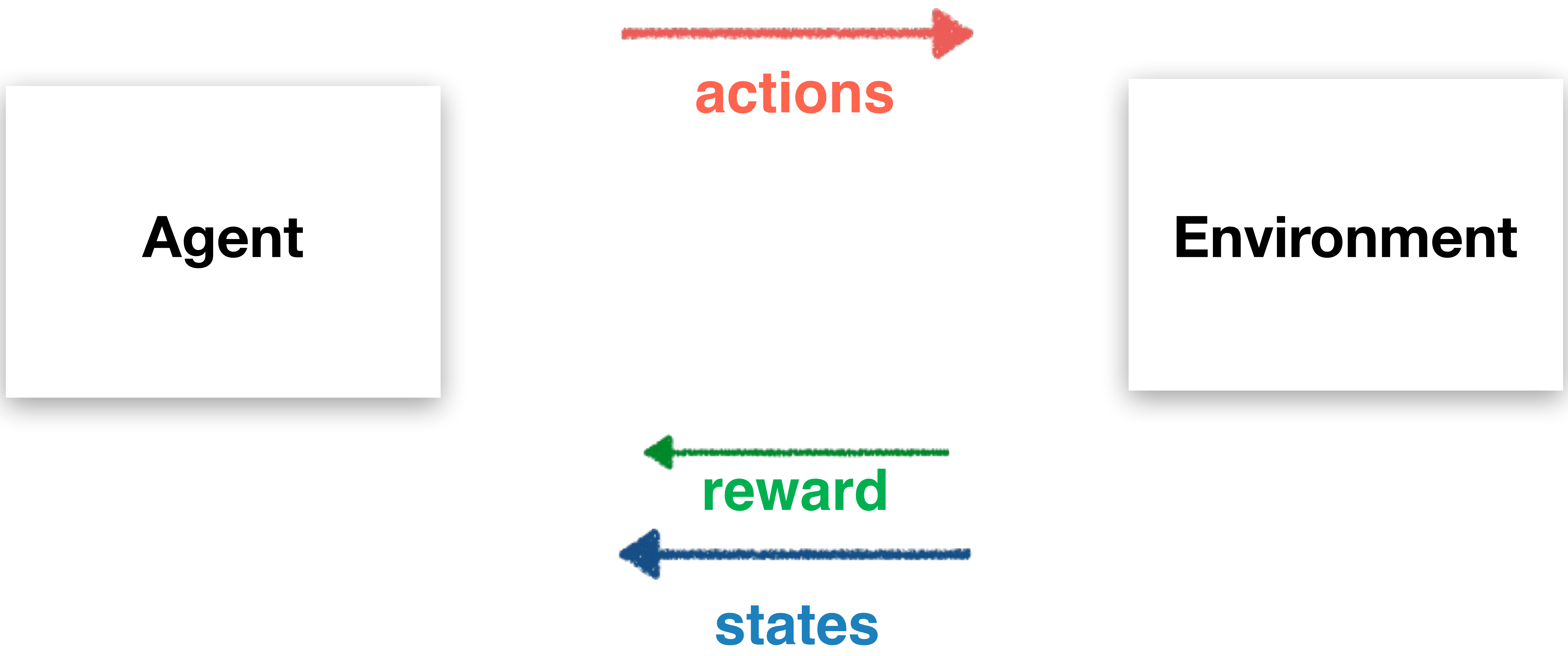
S_0 A_0 R_1 S_1

Online Reinforcement Learning



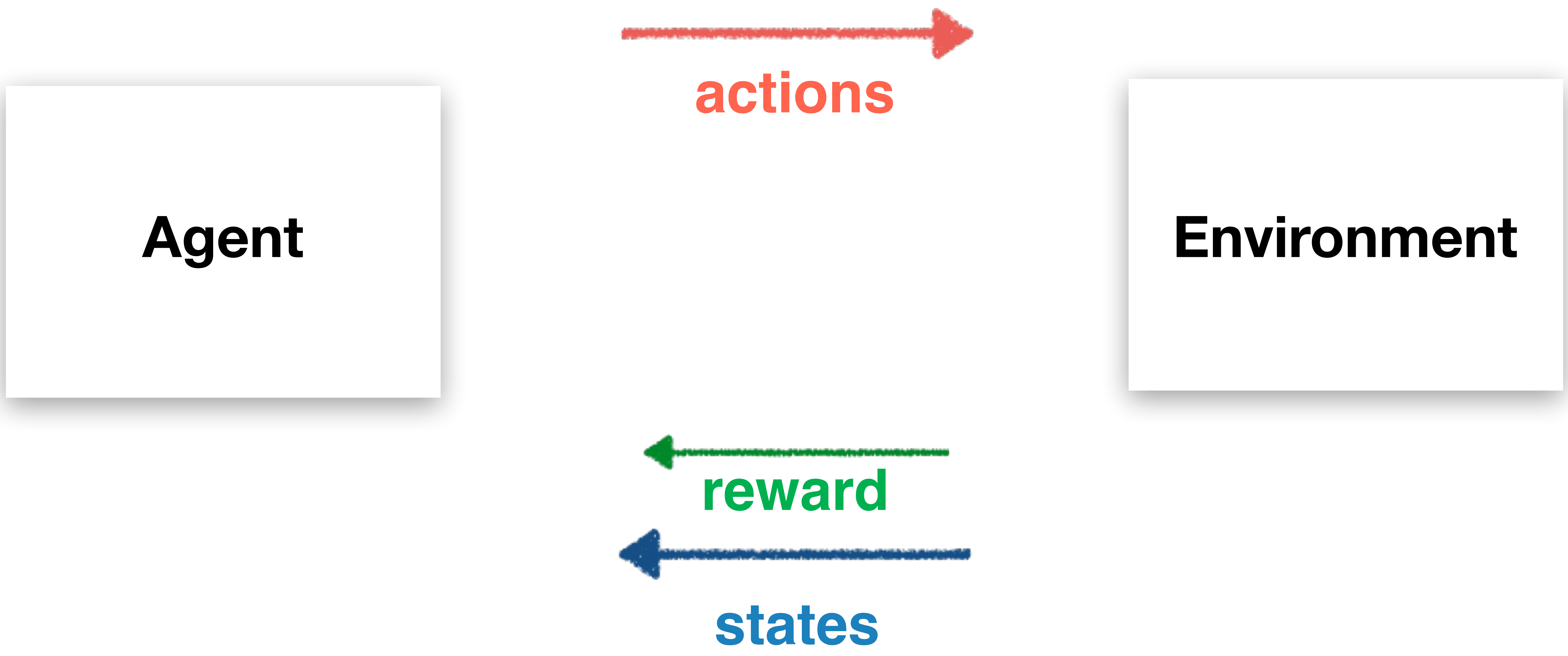
$S_0 \quad A_0 \quad R_1 \quad S_1 \quad A_1$

Online Reinforcement Learning



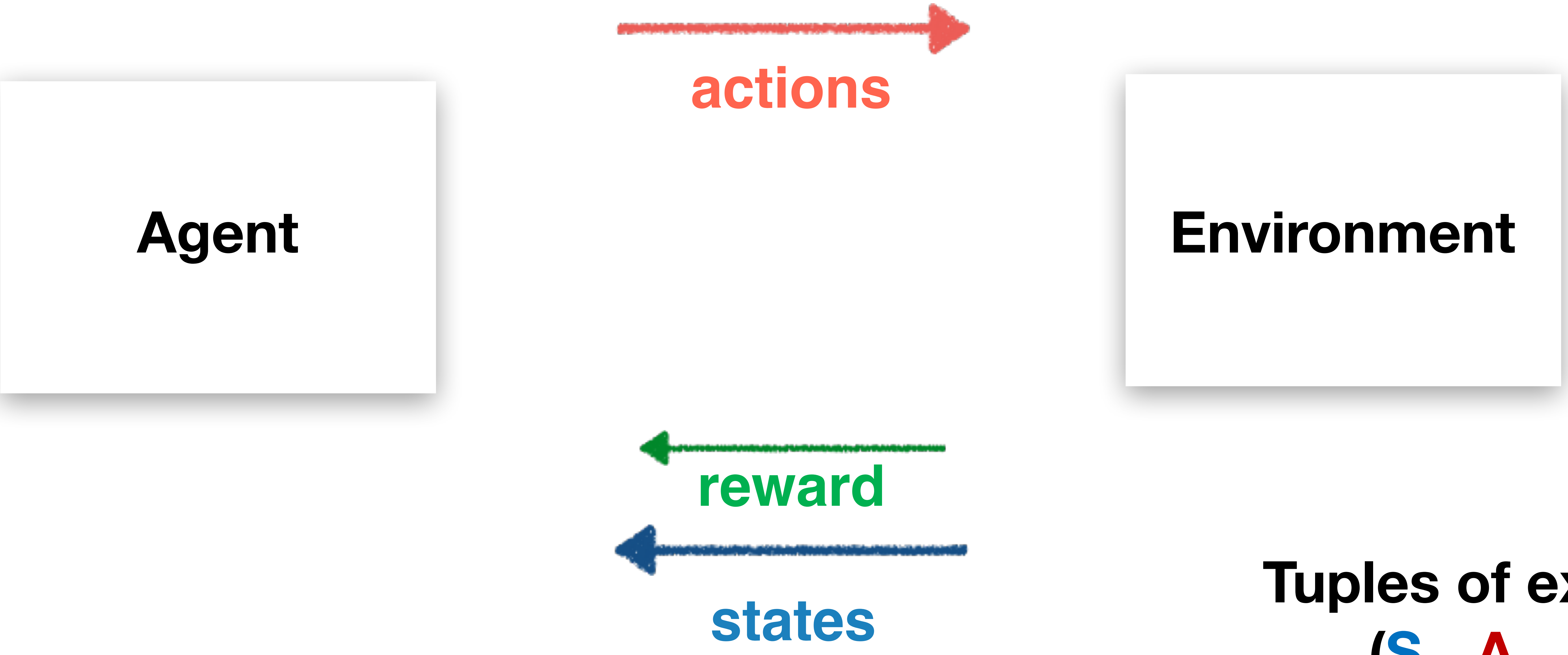
$S_0 \quad A_0 \quad R_1 \quad S_1 \quad A_1 \quad R_2 \quad S_2$

Online Reinforcement Learning



$S_0 \quad A_0 \quad R_1 \quad S_1 \quad A_1 \quad R_2 \quad S_2 \quad A_2 \ldots$

Online Reinforcement Learning



$S_0 \quad A_0 \quad R_1 \quad S_1 \quad A_1 \quad R_2 \quad S_2 \quad A_2 \ldots$

Tuples of experience:

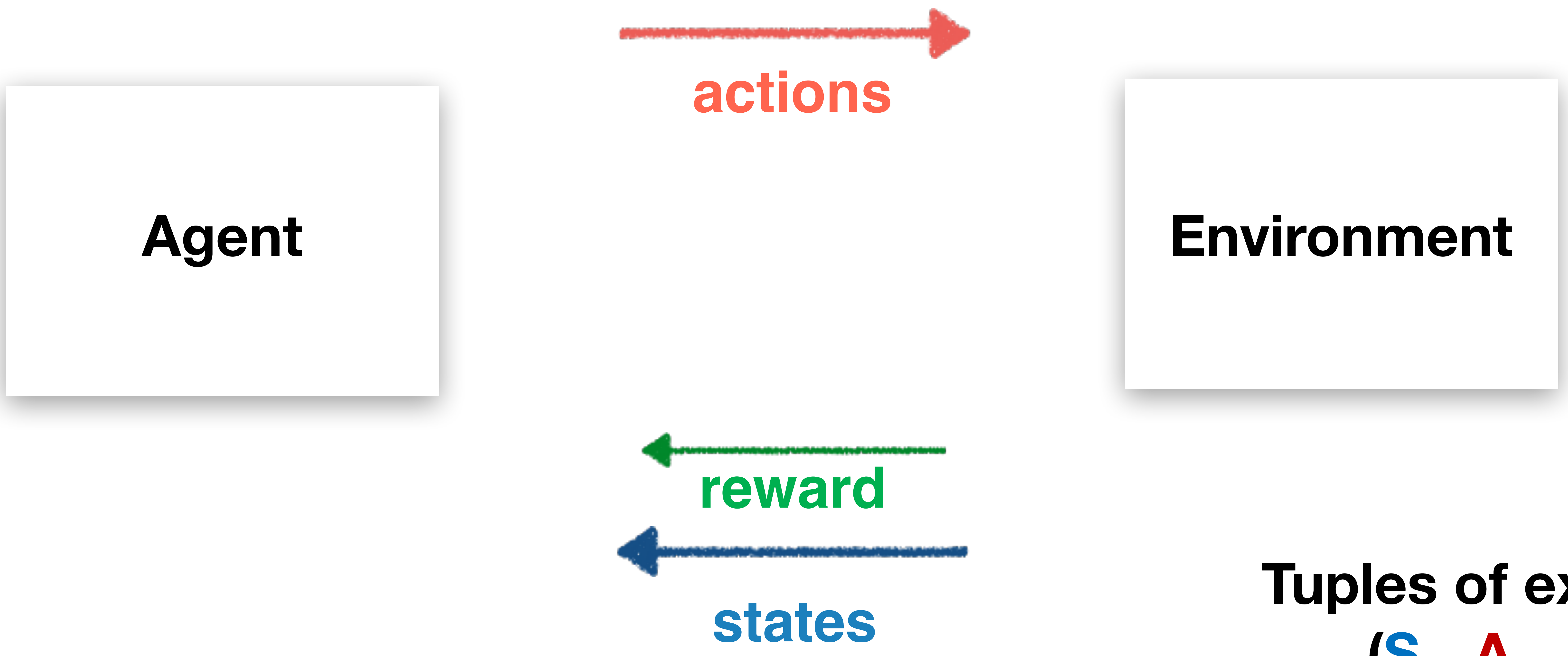
(S_0, A_0, R_1, S_1)

(S_1, A_1, R_2, S_2)

(S_2, A_2, R_3, S_3)

...

Online Reinforcement Learning



$S_0 \quad A_0 \quad R_1 \quad S_1 \quad A_1 \quad R_2 \quad S_2 \quad A_2 \ldots$

Tuples of experience:

(S_0, A_0, R_1, S_1)

(S_1, A_1, R_2, S_2)

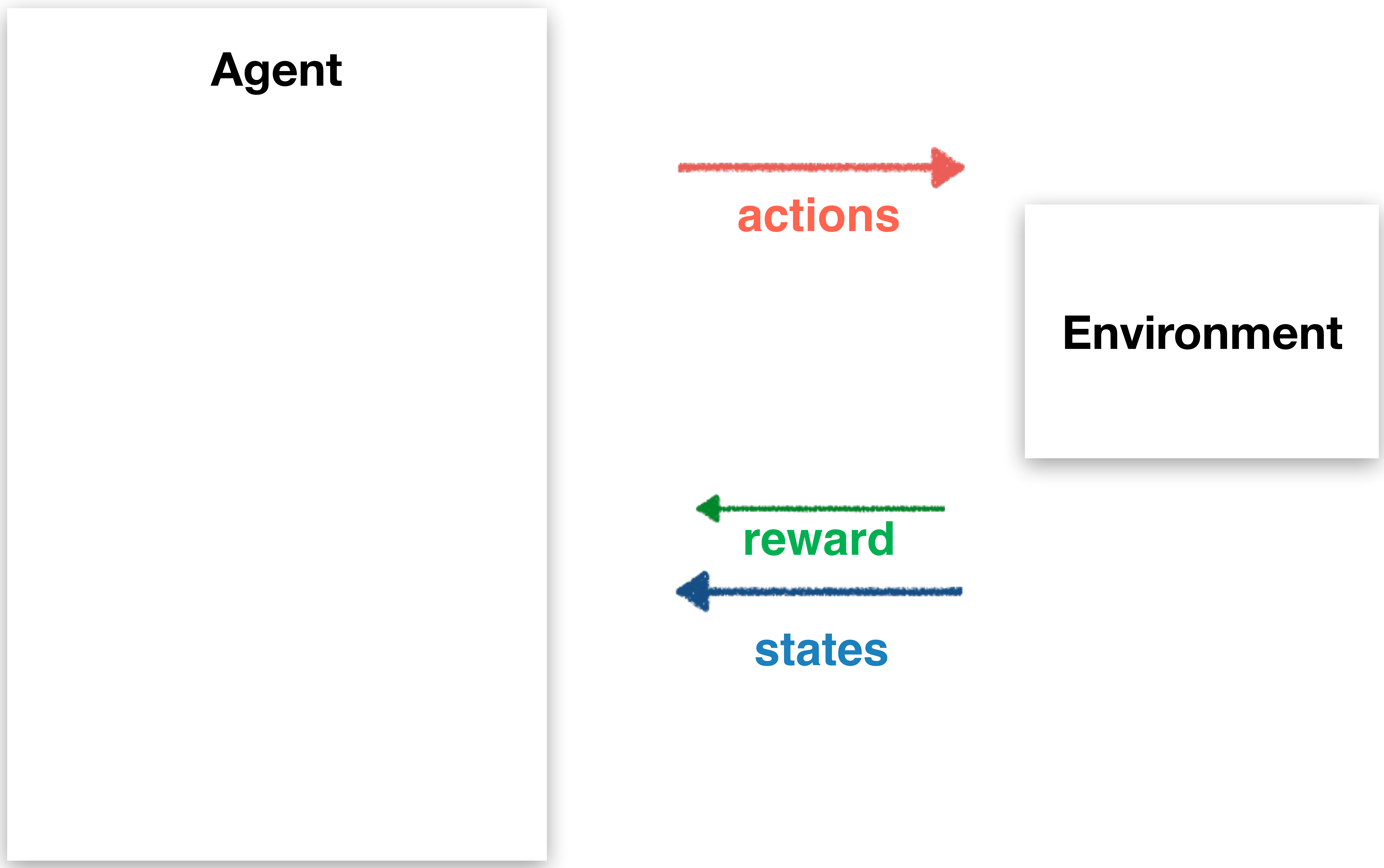
(S_2, A_2, R_3, S_3)

...

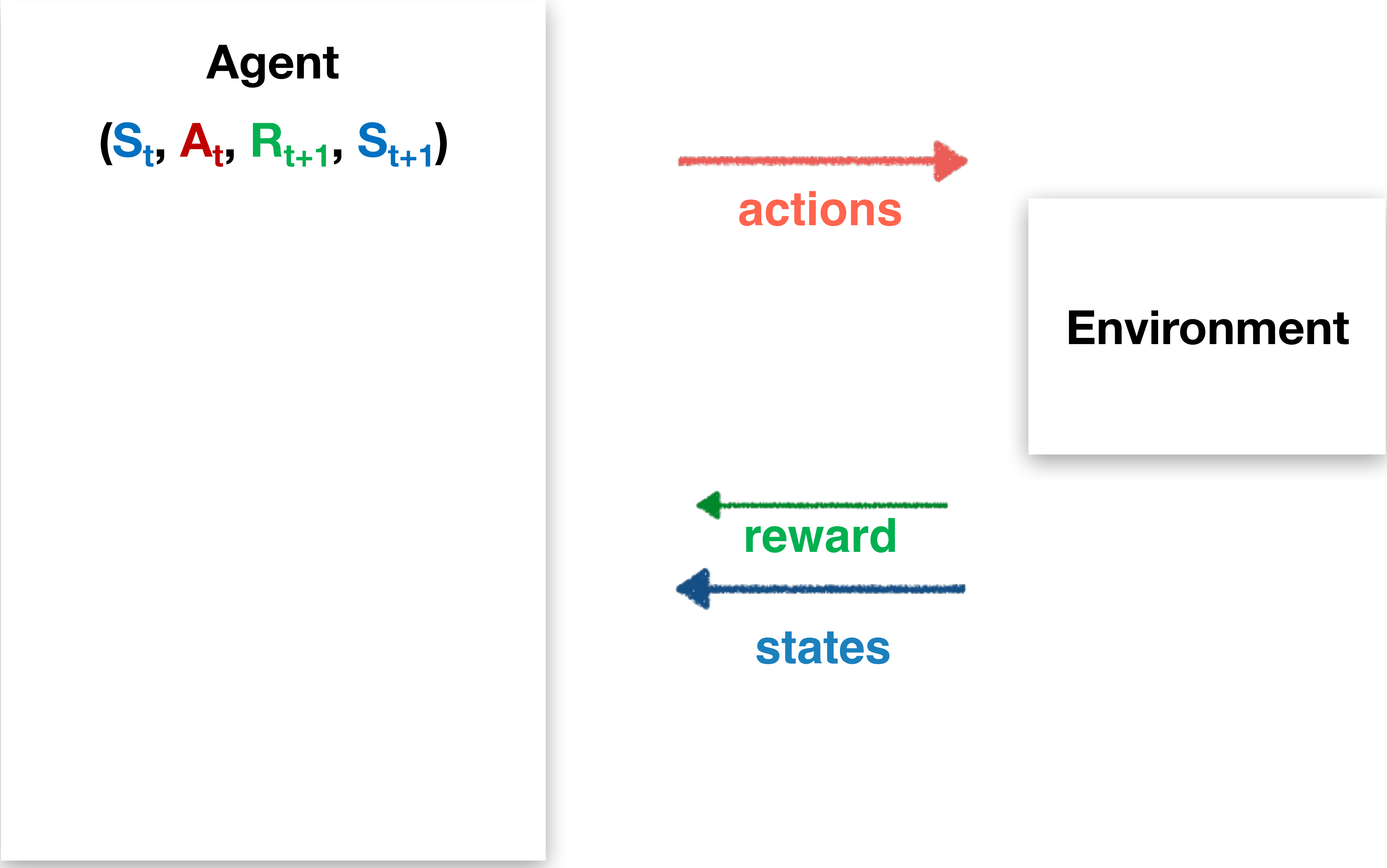
Q-learning update:

$$Q(S, A) = Q(S, A) + \alpha\,[\,R + \gamma \max Q(S', A') - Q(S, A)]$$

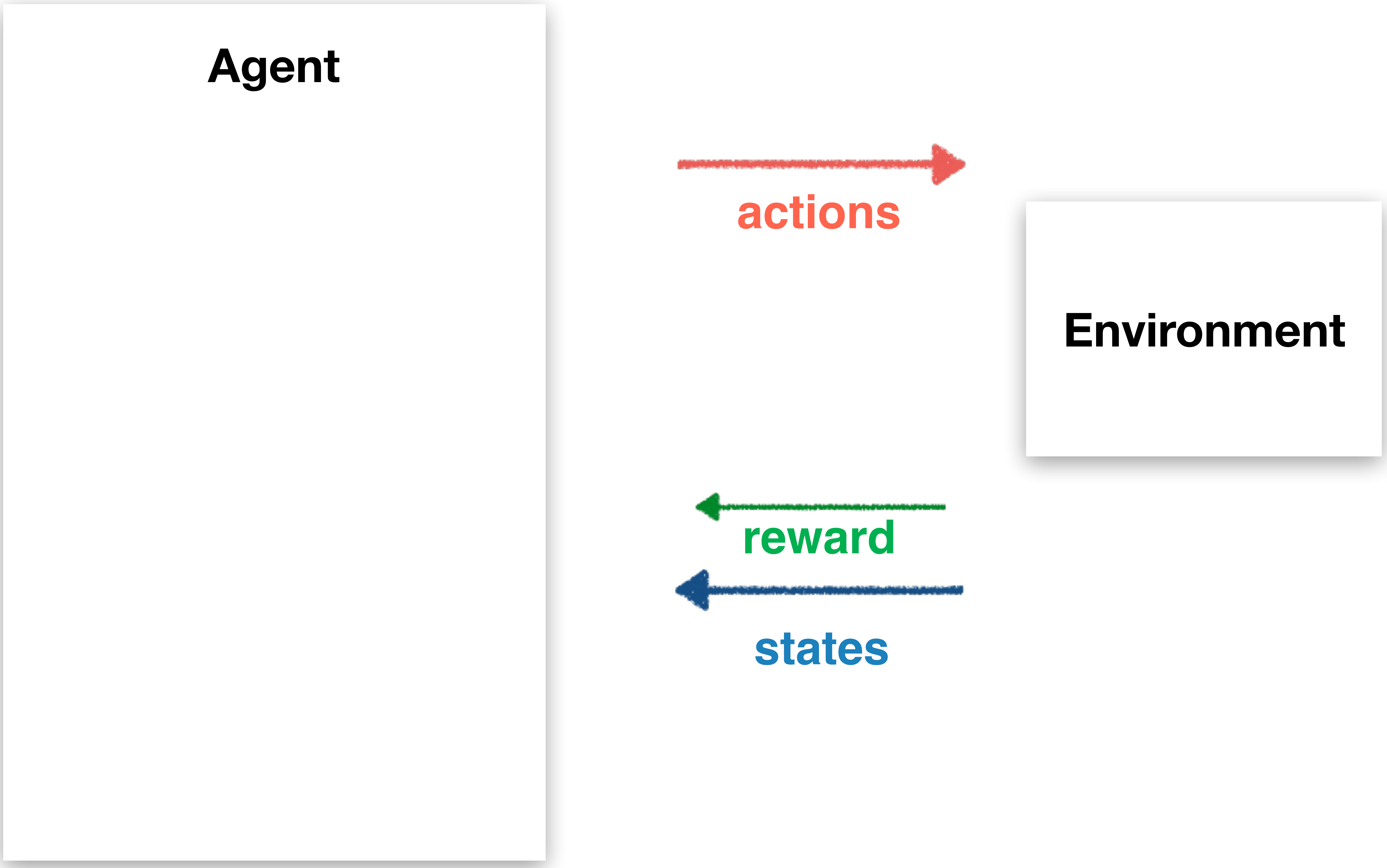
Online RL without a Model



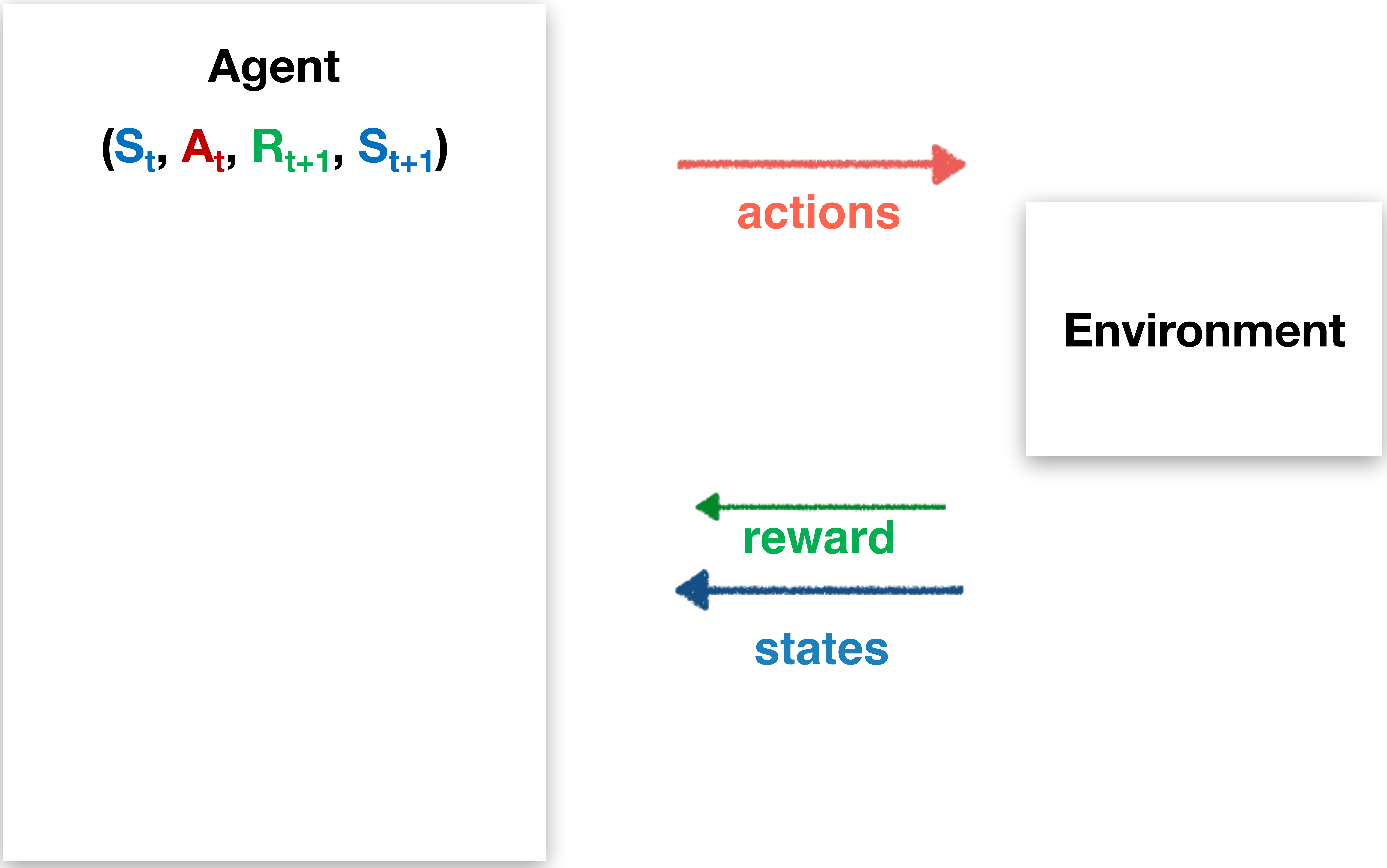
Online RL without a Model



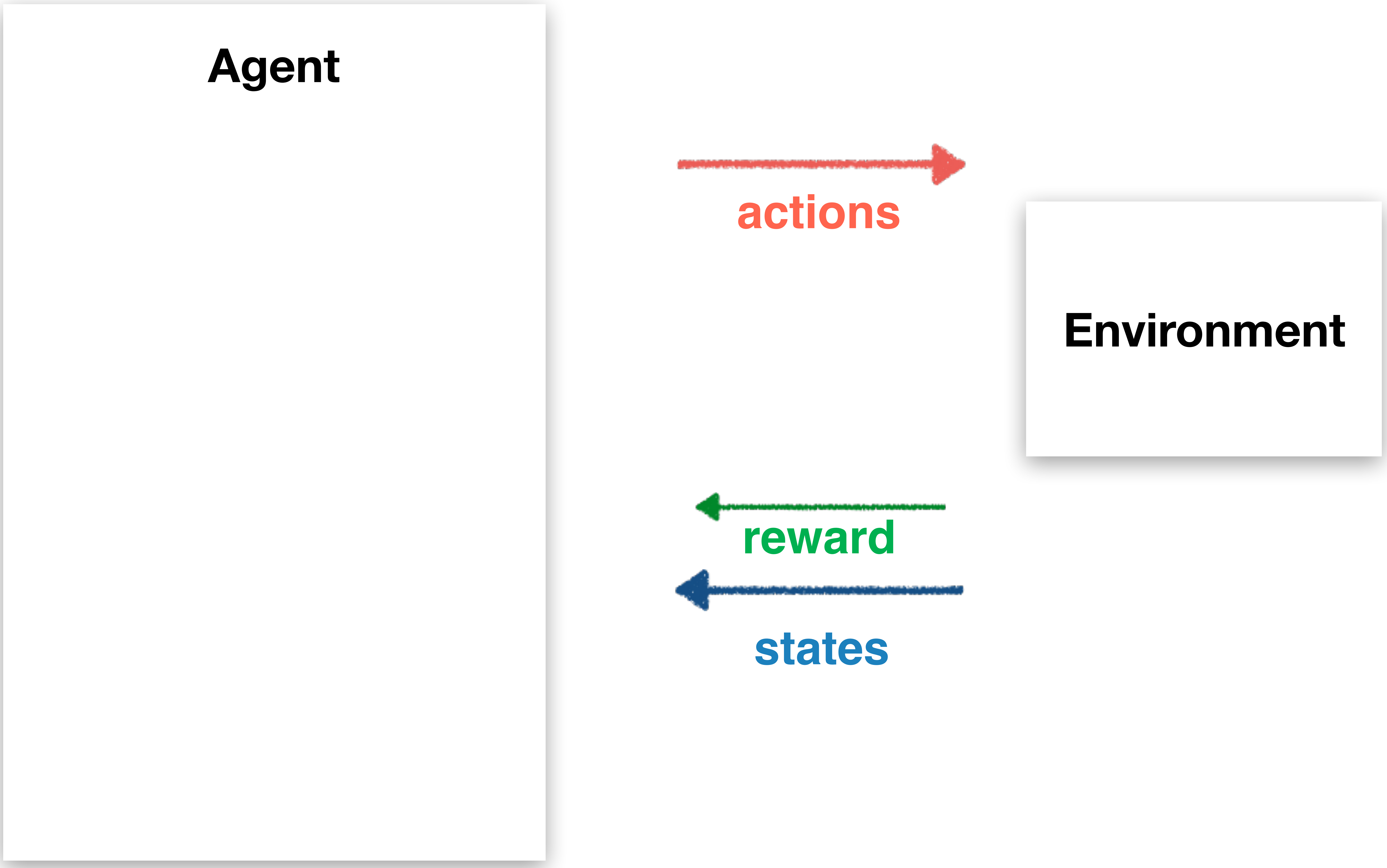
Online RL without a Model



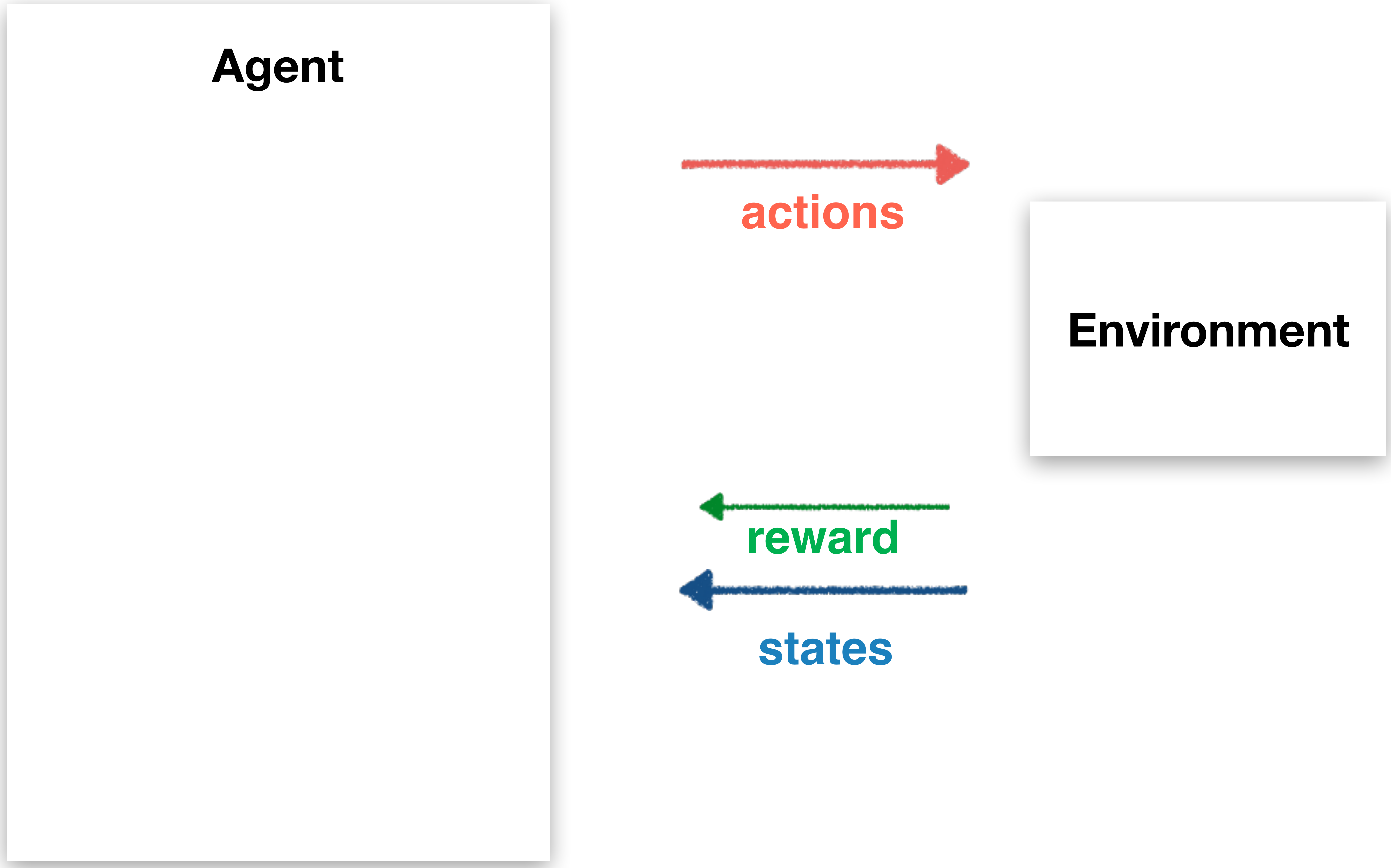
Online RL without a Model



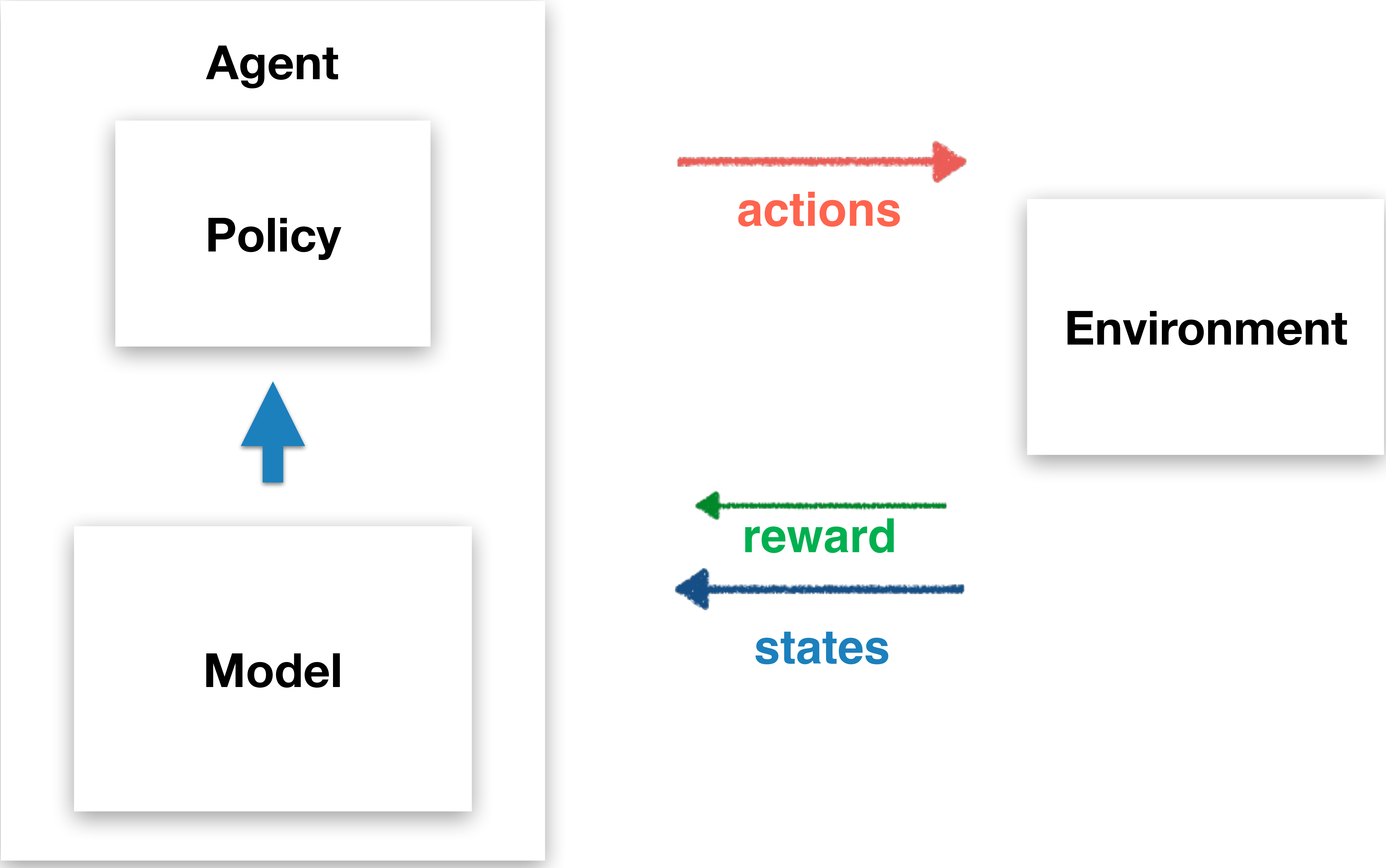
Online RL without a Model



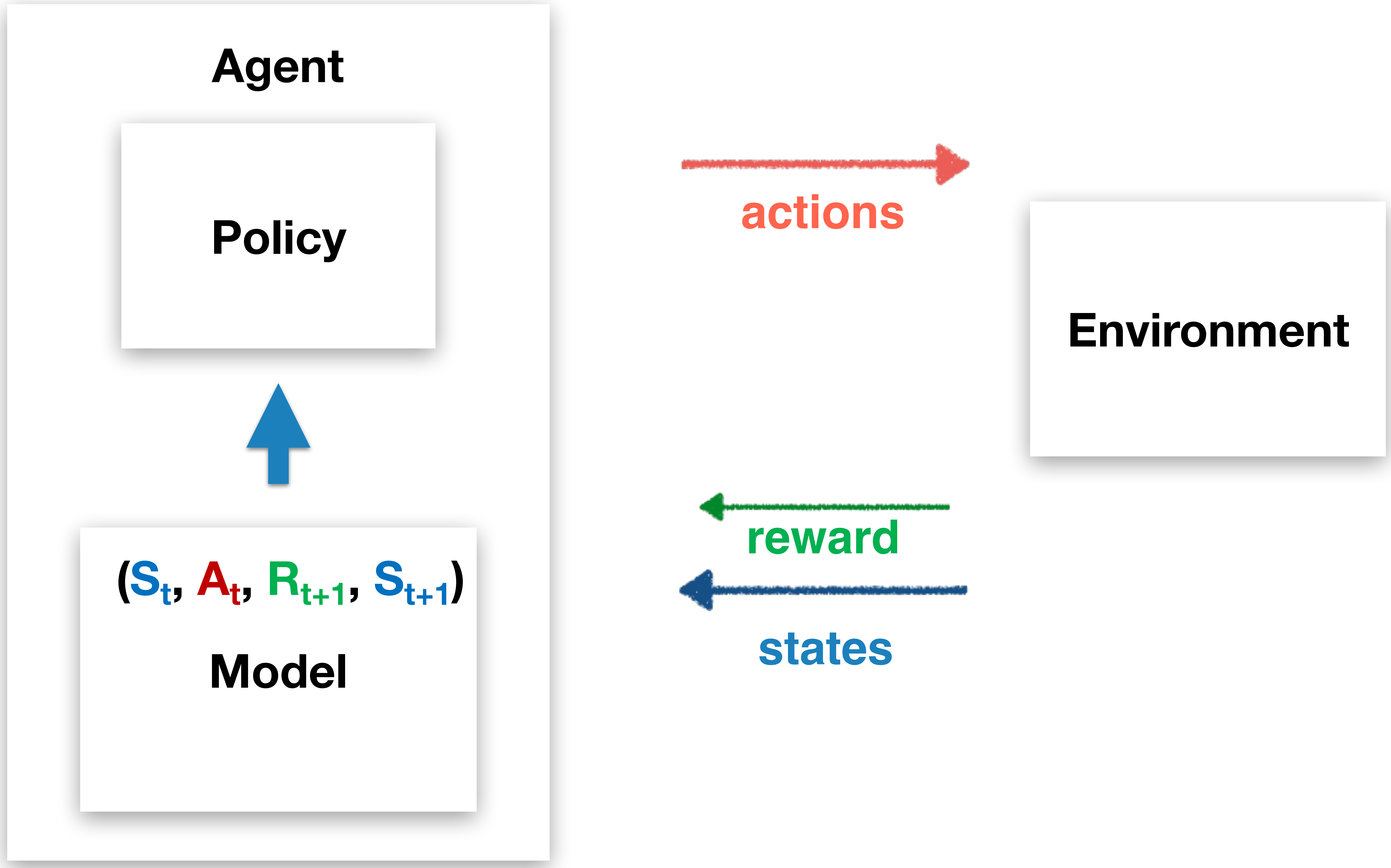
Online RL with a Model



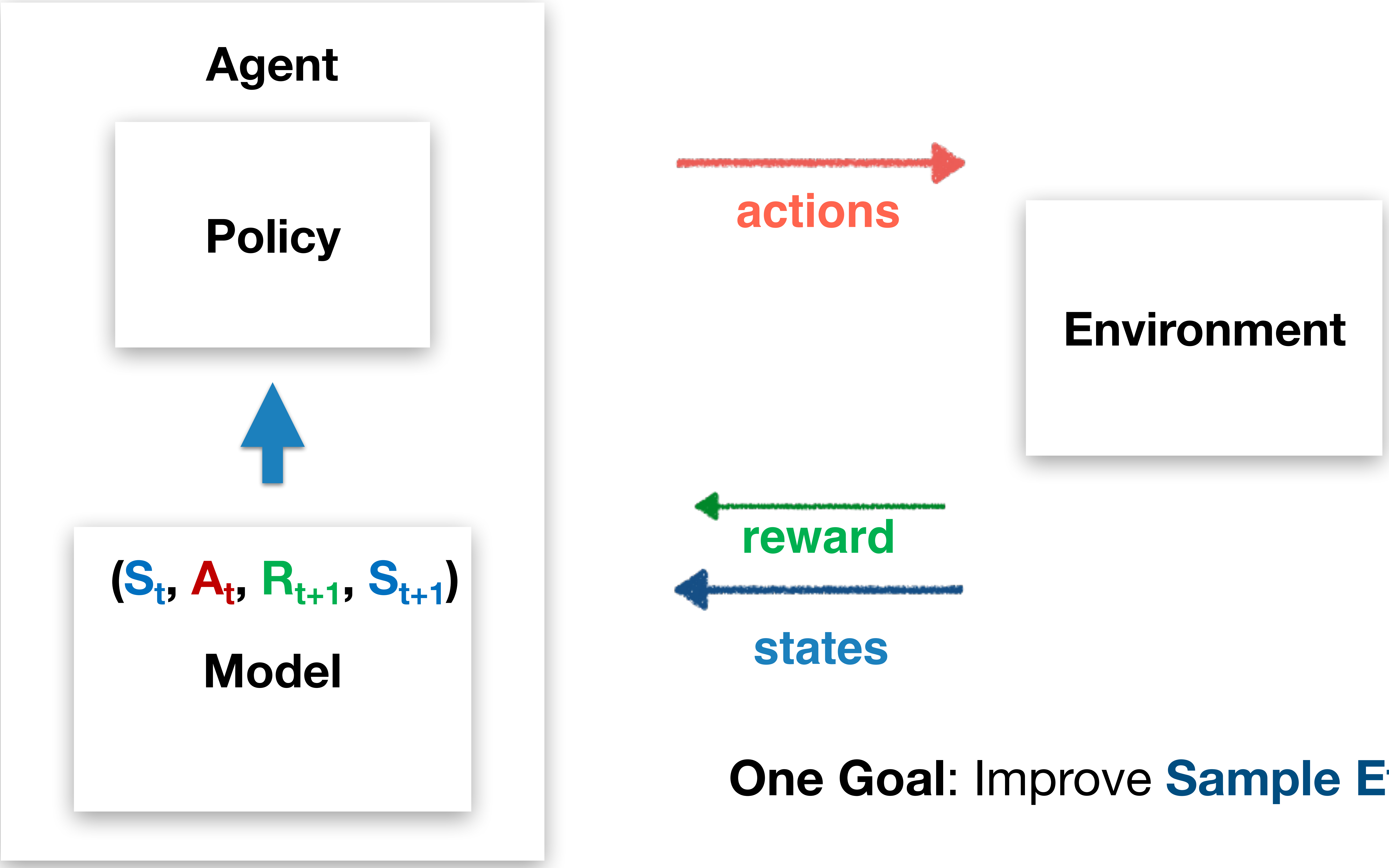
Online RL with a Model



Online RL with a Model



Online RL with a Model



One Goal: Improve **Sample Efficiency**

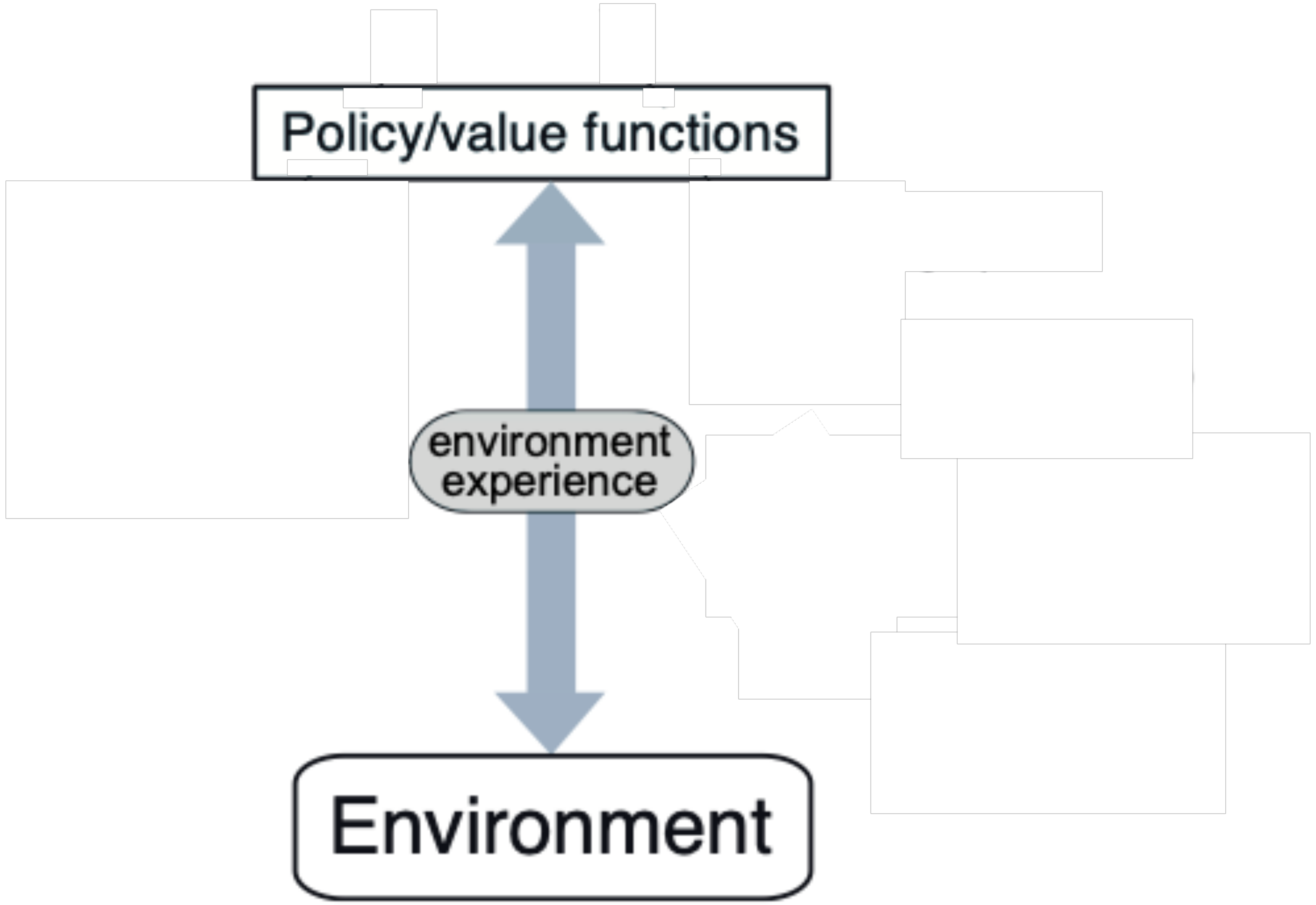
What are possible learned models?

- **Most obvious answer**: $\hat{p}(s', r \mid s, a)$
- **Realistically**: models with state abstraction and temporal abstraction
- **For now**: let's assume we learn approximation $\hat{p}(s', r \mid s, a)$

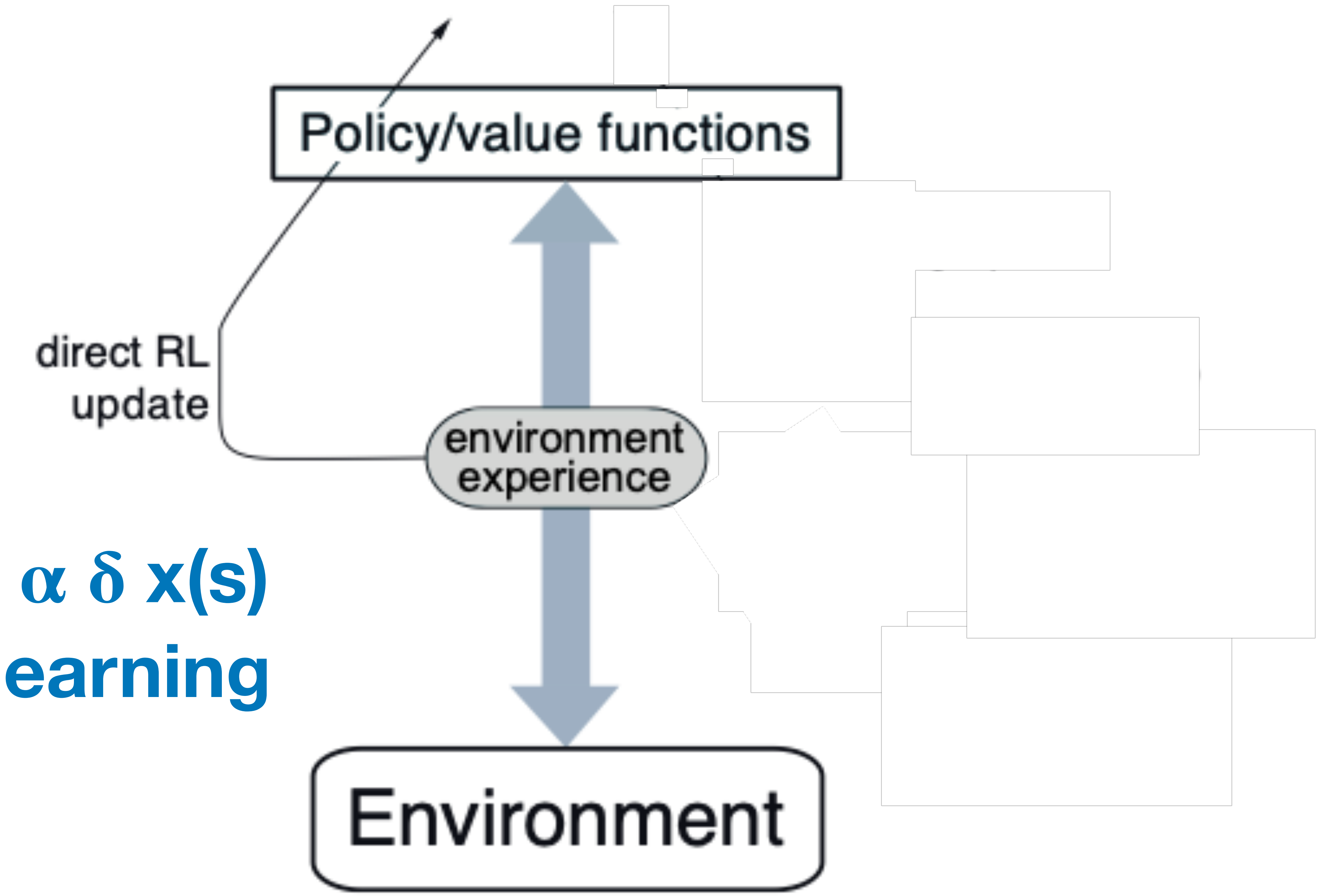
Outline for Part 2:
Moving to Learned Models

- Introduce a **planning** framework called **Dyna**
  - Explain how Experience Replay is a simple instance of Dyna
- Discuss two key choices in Dyna: **Model** and **Search Control**
- Discuss different choices for the **Model**
- Discuss different choices for **Search Control**

What is Dyna?

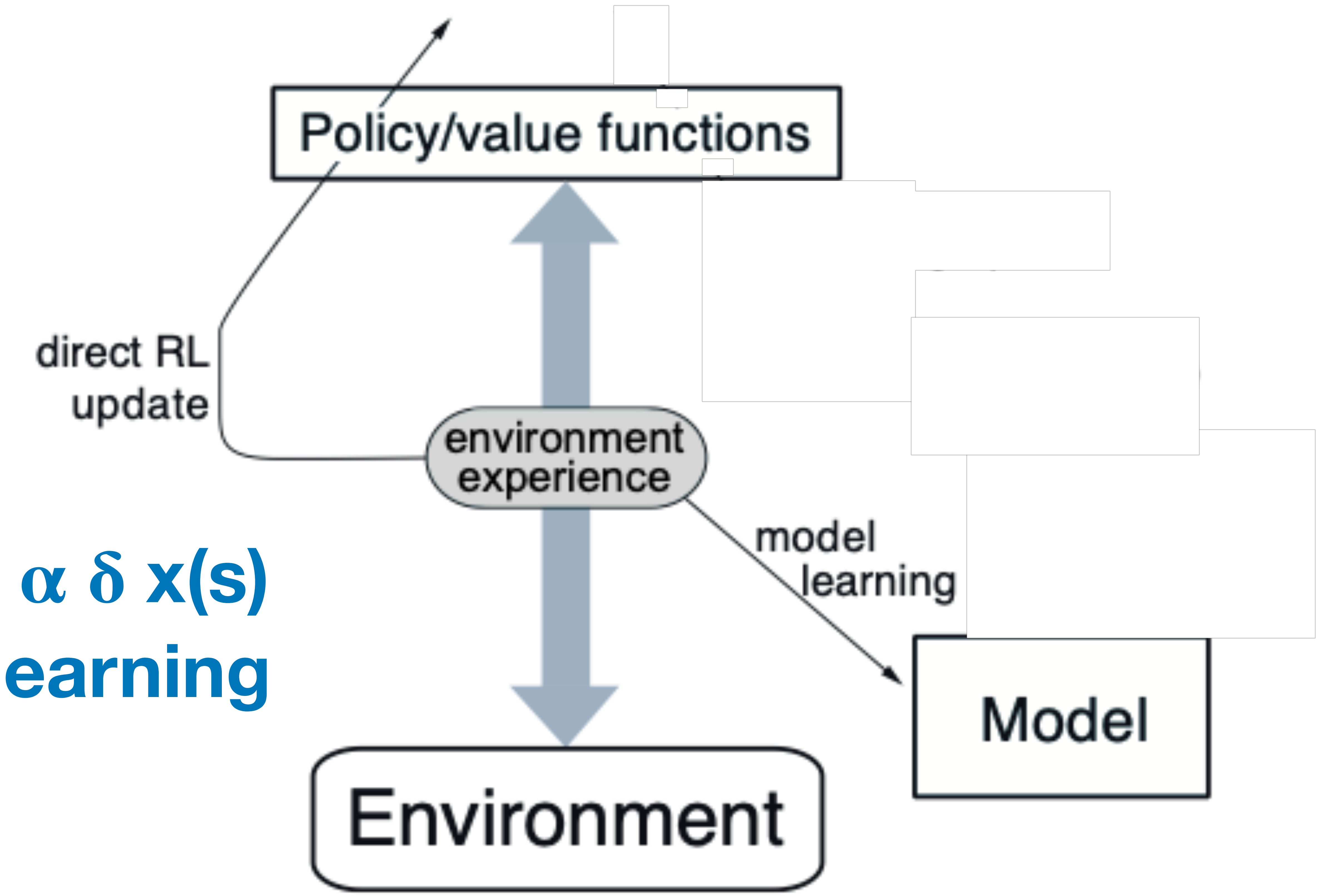

Policy/value functions
environment experience
Environment

What is Dyna?



$w = w + \alpha\ \delta\ x(s)$
e.g., Q-learning

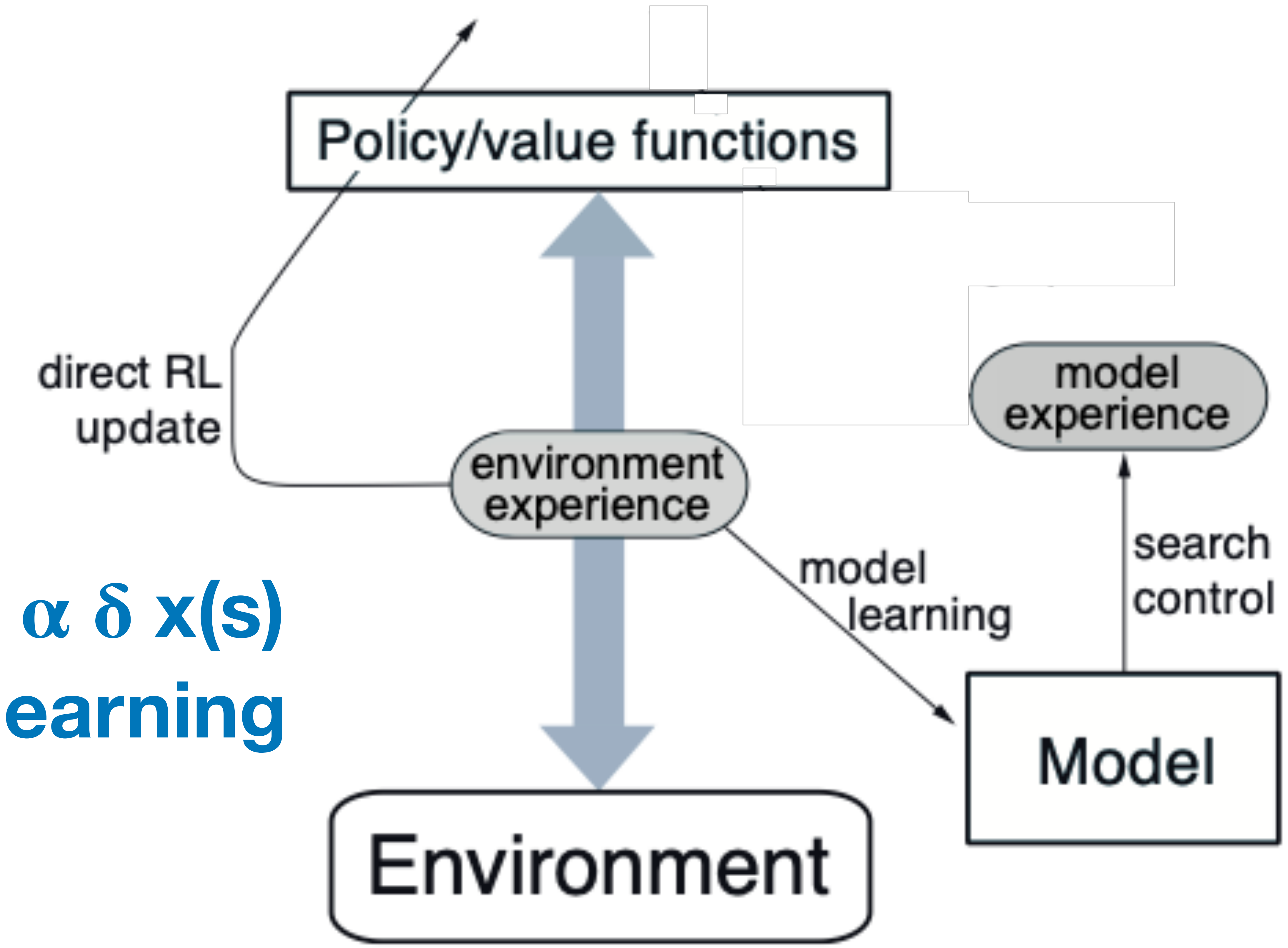
What is Dyna?



w = w + α δ x(s)

e.g., Q-learning

What is Dyna?

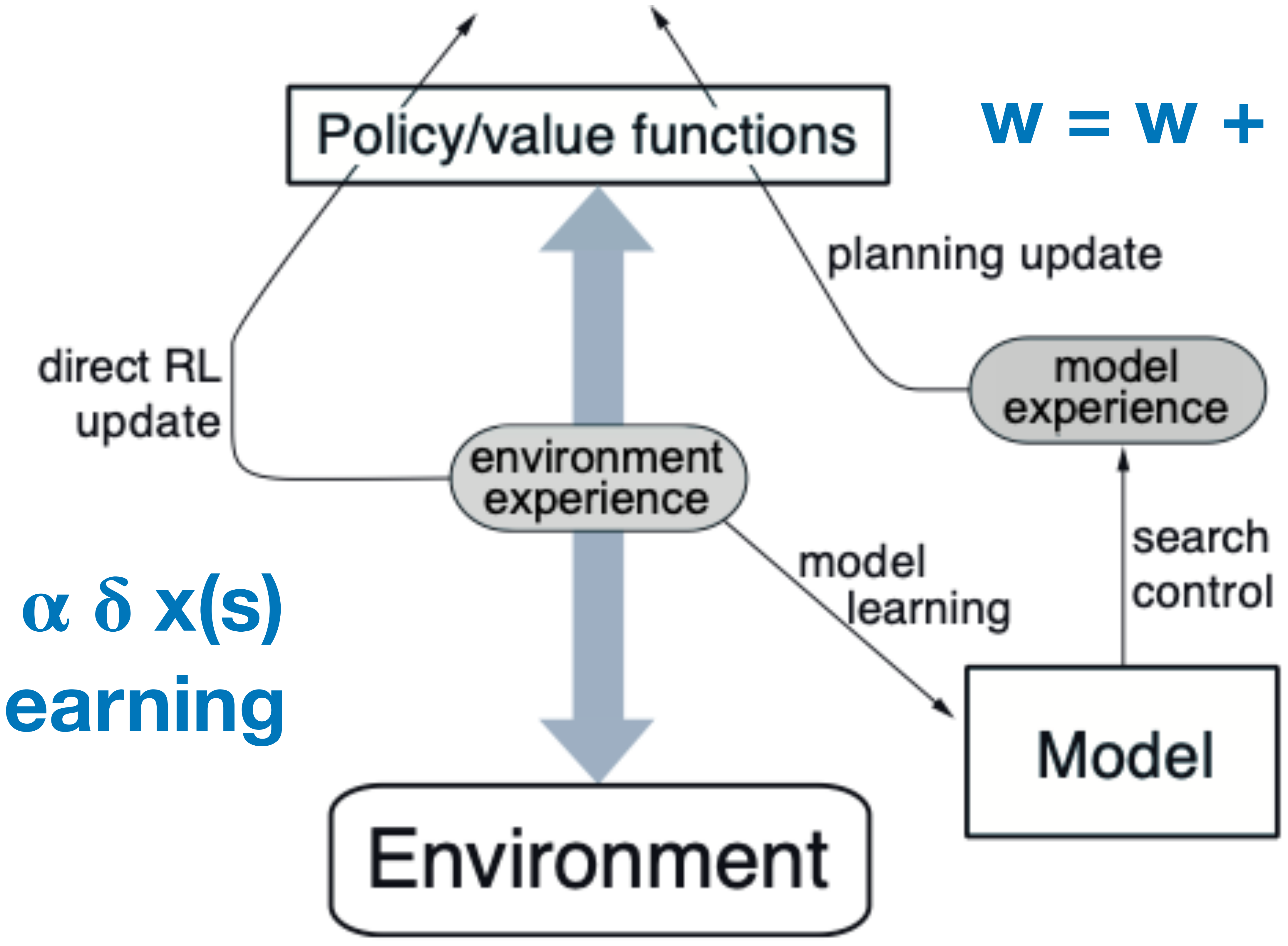


$w = w + \alpha\, \delta\, x(s)$
e.g., Q-learning

s’, r ~ M(s,a)

(s,a)

What is Dyna?



$w = w + \alpha \delta x(s)$

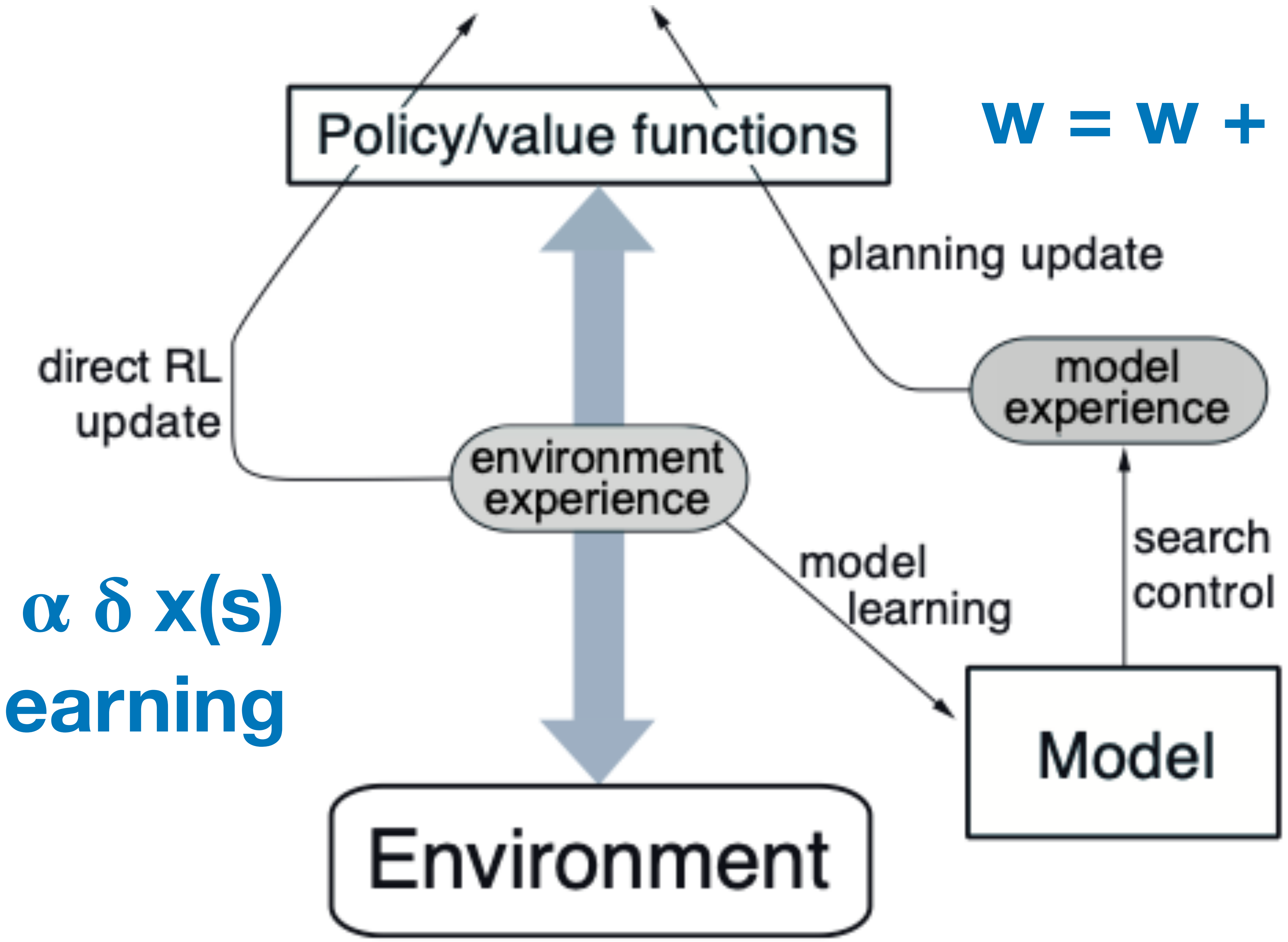
$s', r \sim M(s,a)$

(s,a)

$w = w + \alpha \delta x(s)$

e.g., Q-learning

What is Dyna?



$w = w + \alpha\, \delta\, x(s)$

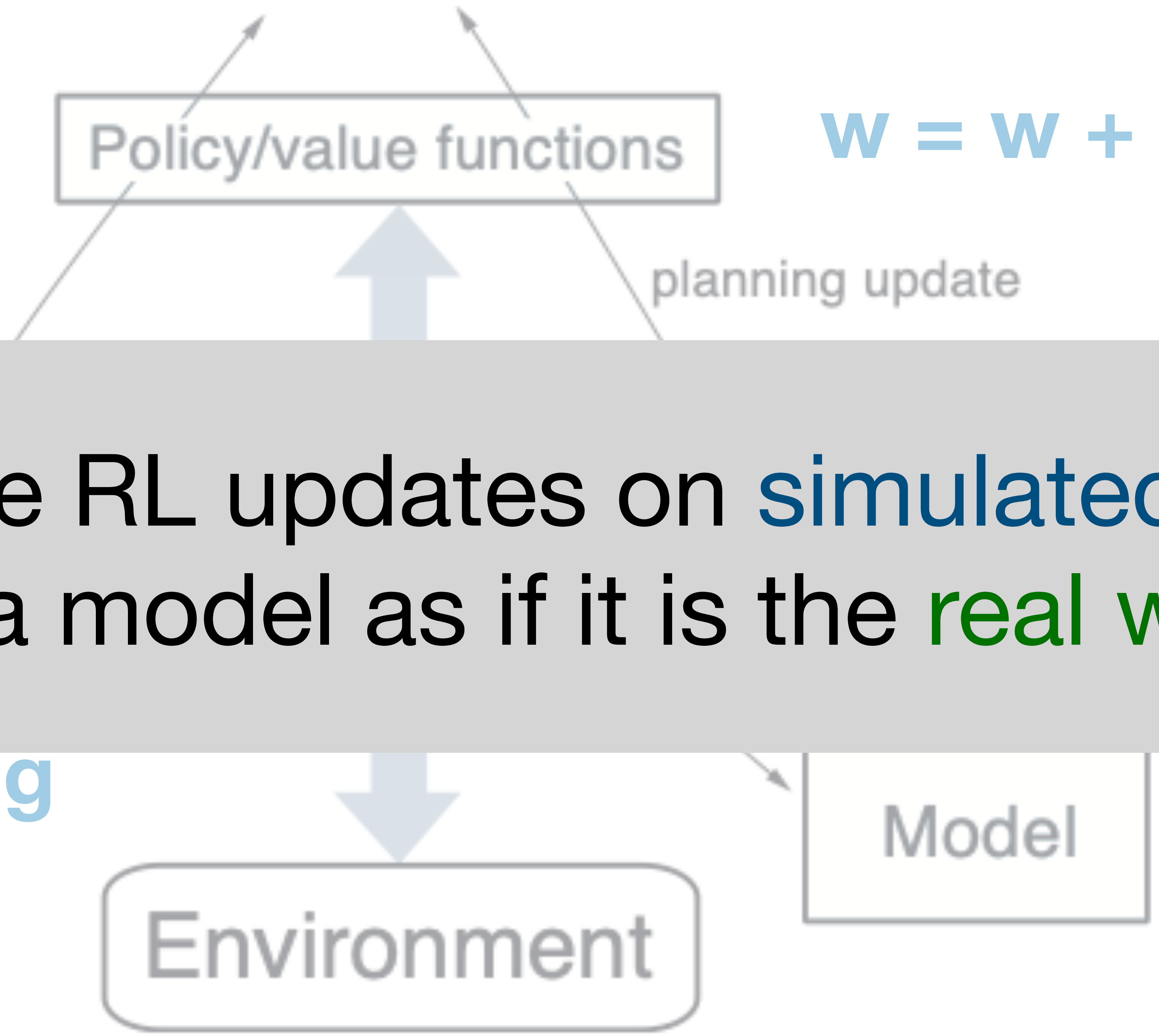
$s', r \sim M(s,a)$

(s,a)

$w = w + \alpha\, \delta\, x(s)$

e.g., Q-learning

What is Dyna?



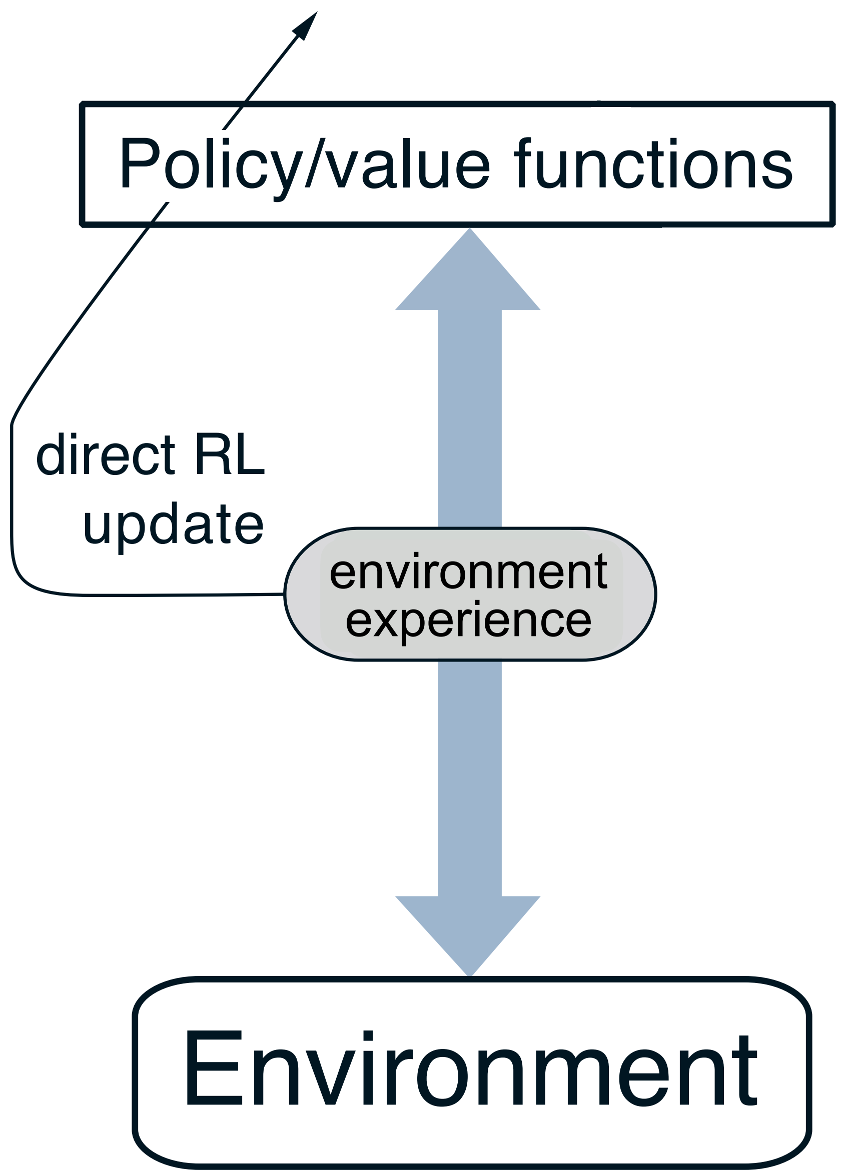
$w = w + \alpha\, \delta\, x(s)$

e.g., Q-learning

Key Idea: Use RL updates on simulated experience from a model as if it is the real world

Pseudocode

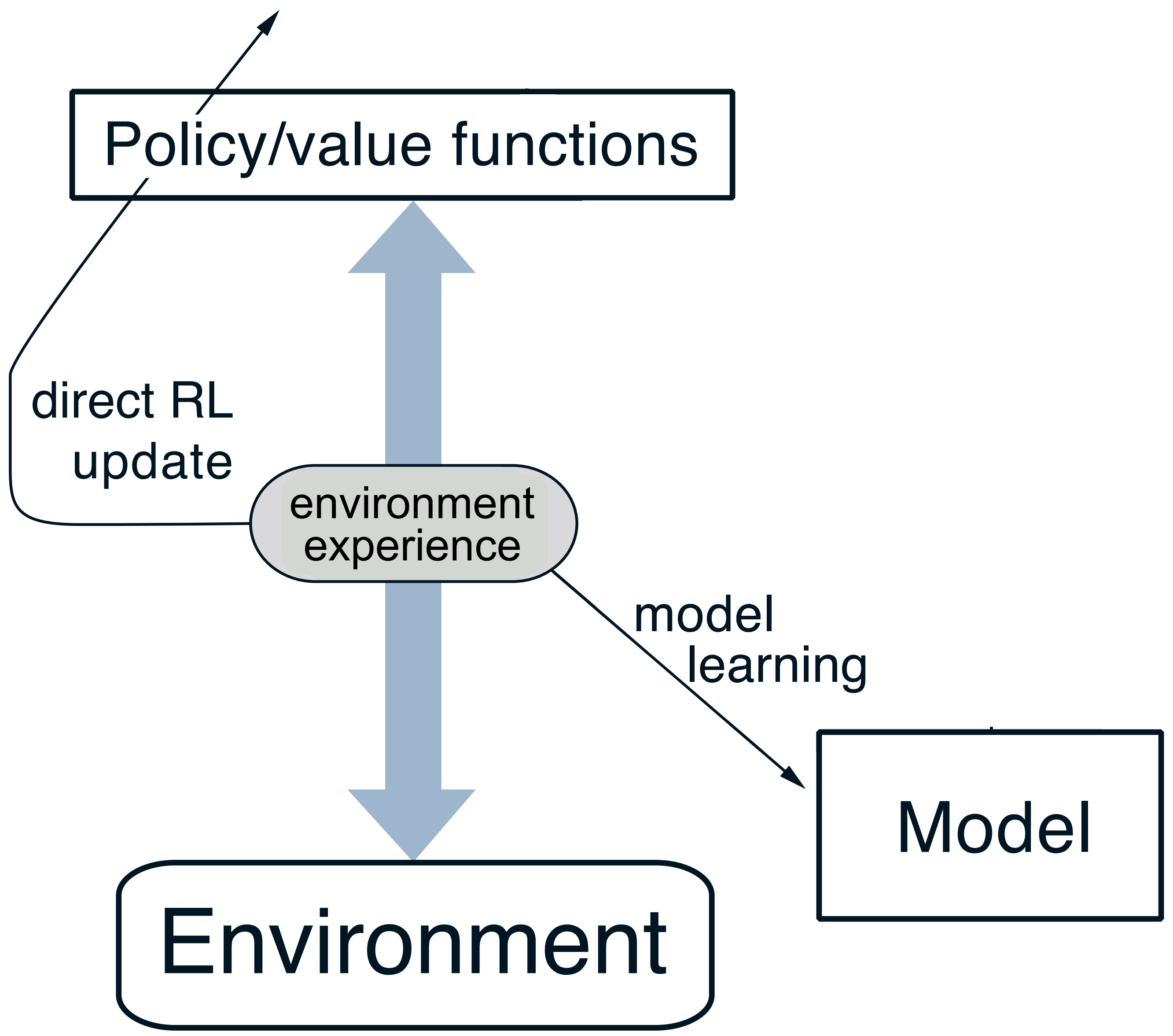
Dyna-Q



Pseudocode

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



Pseudocode

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

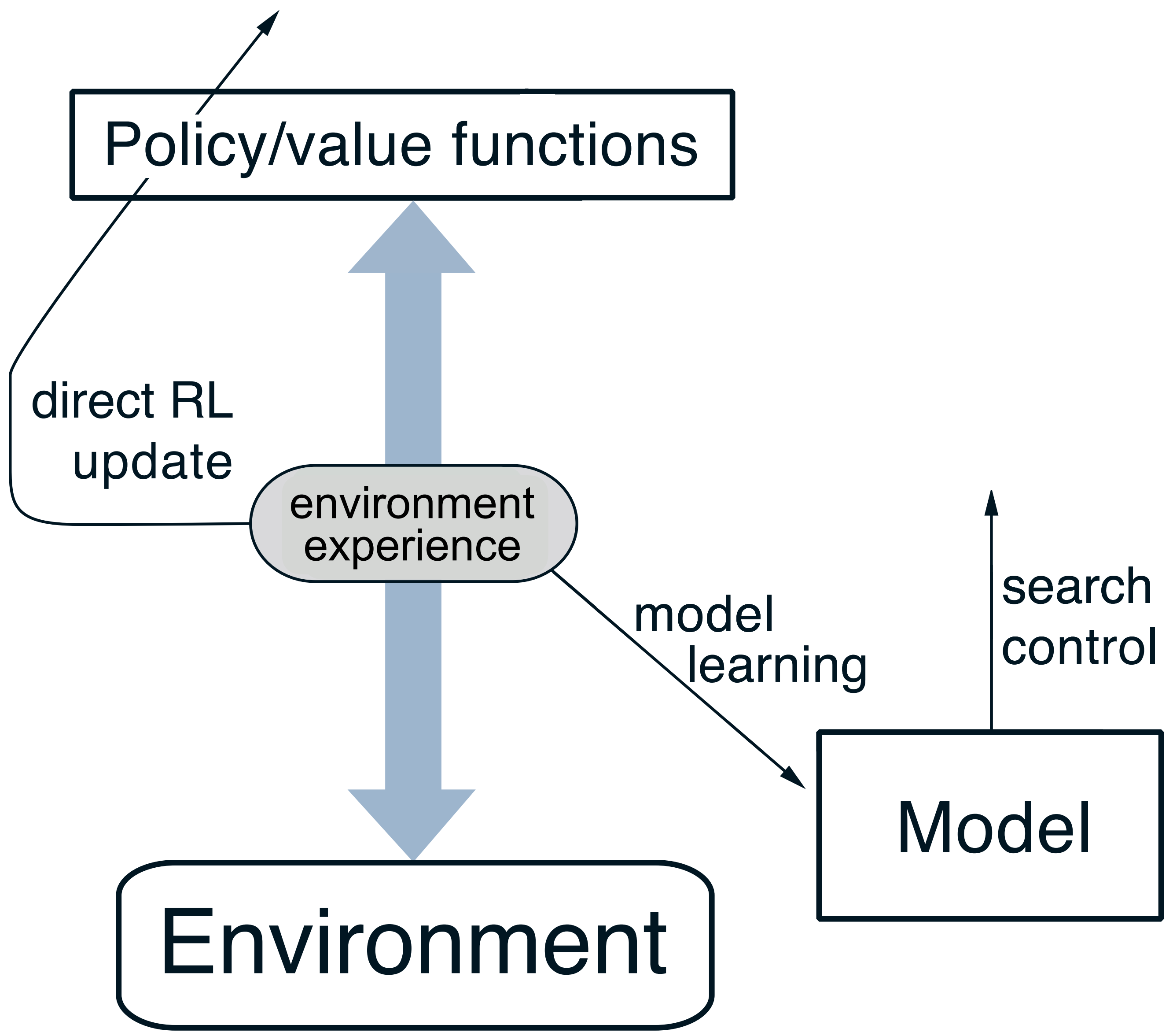
(b) $A \leftarrow \varepsilon$-greedy(S,Q)

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

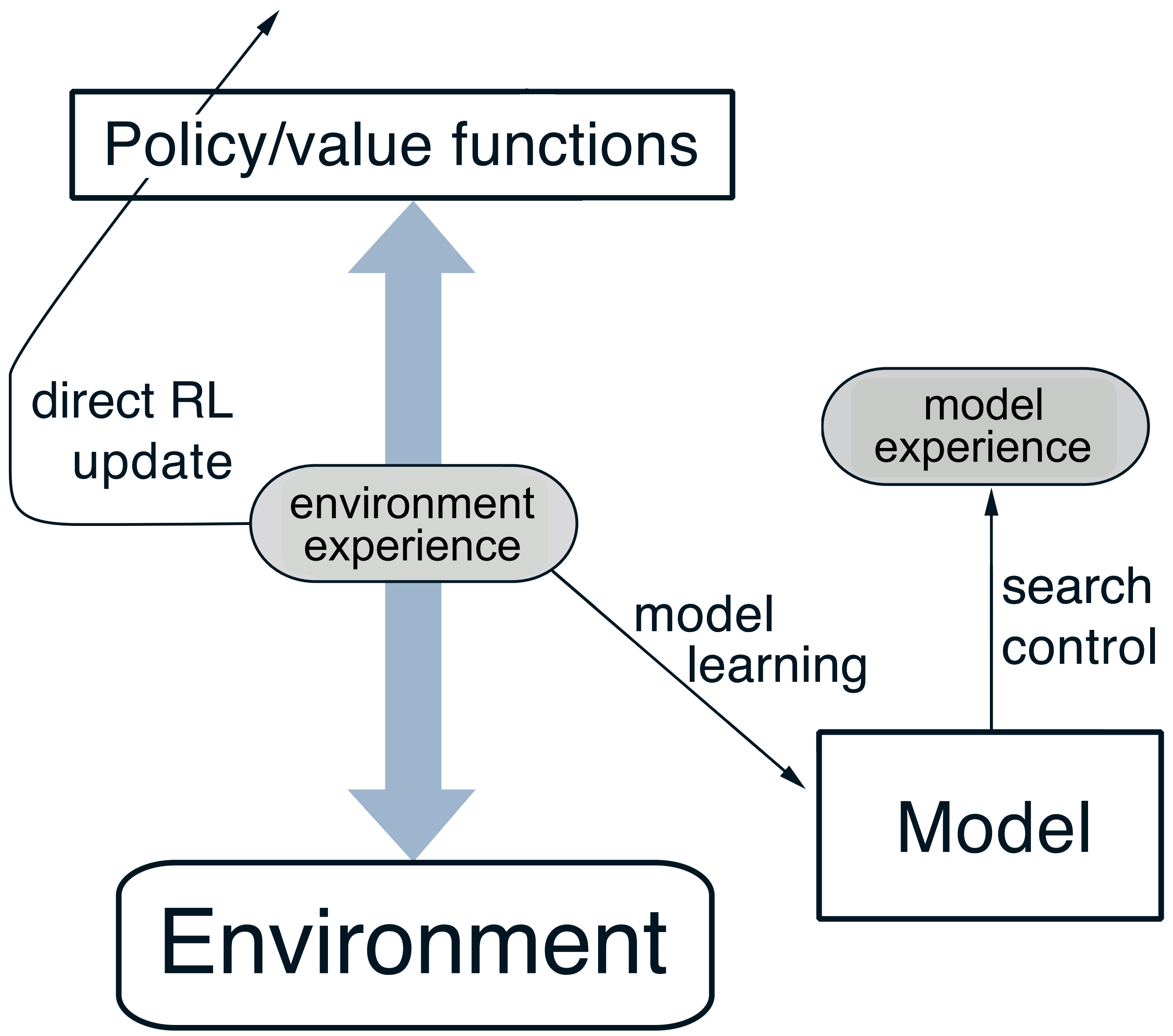
Pseudocode



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

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



Pseudocode

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'

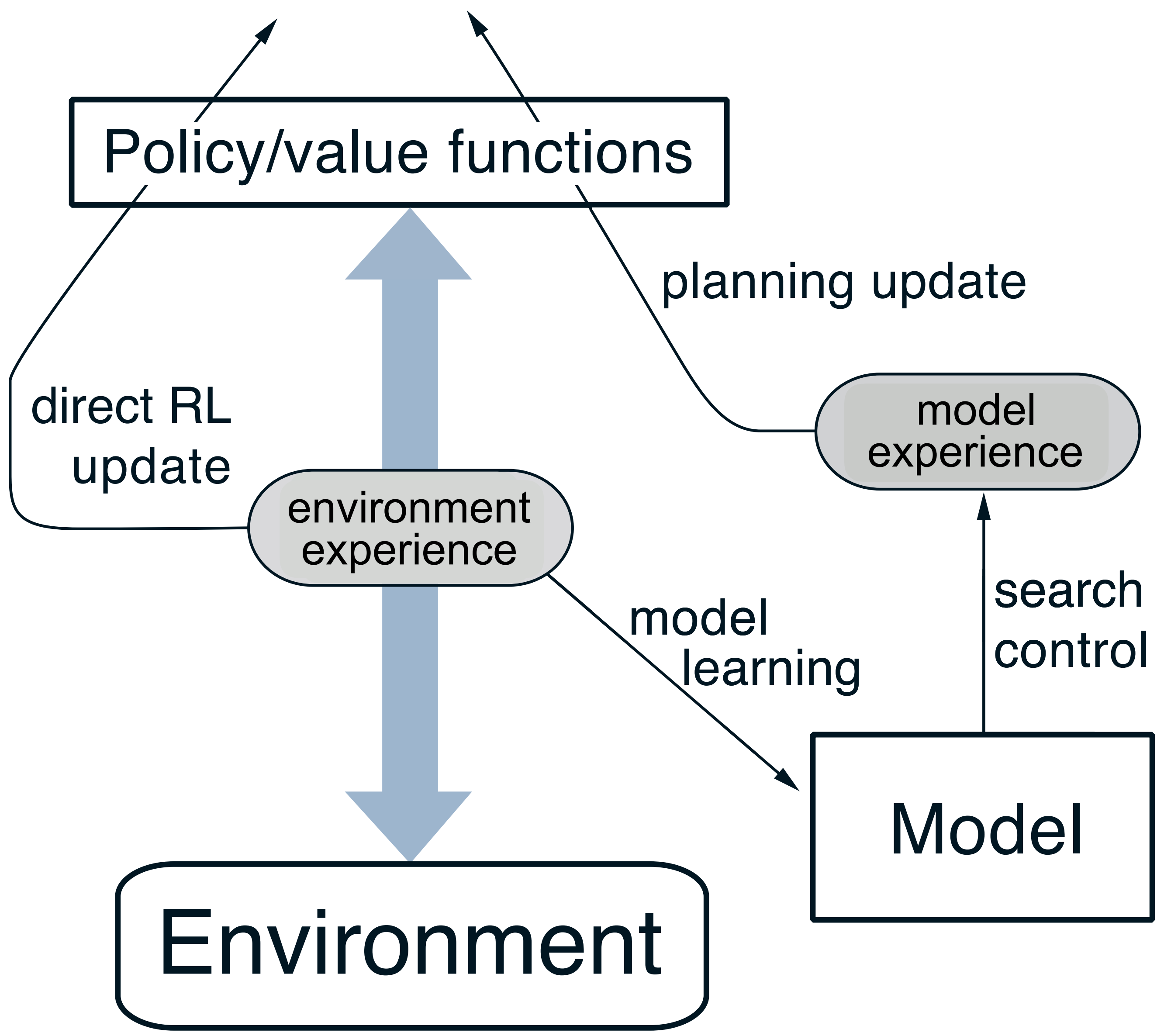
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

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

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

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



Pseudocode

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

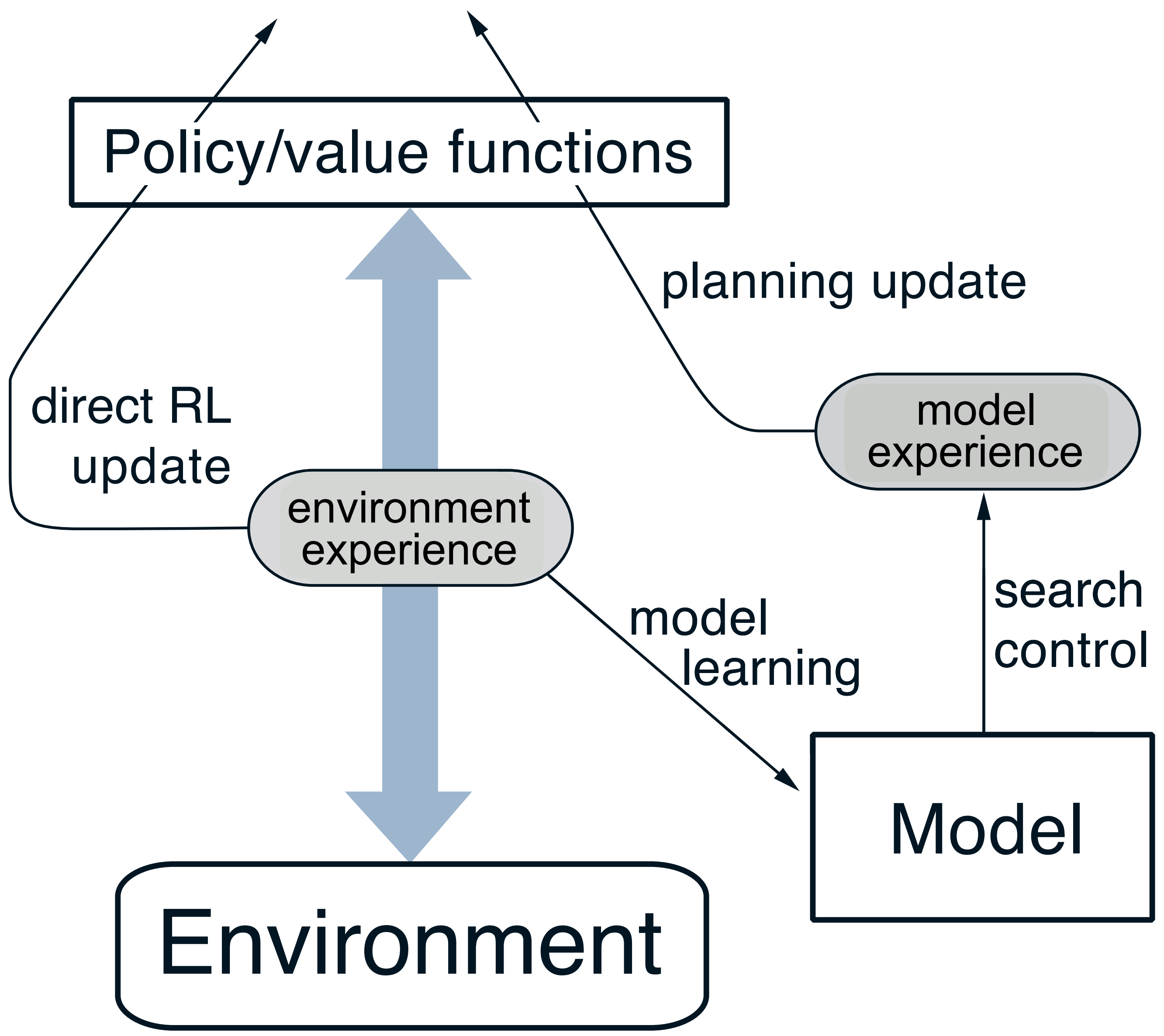
(e) $Model(S, A) \leftarrow R, S'$

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

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

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



Pseudocode

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

(f) Loop repeat n times:

$S \leftarrow$ random previously observed state

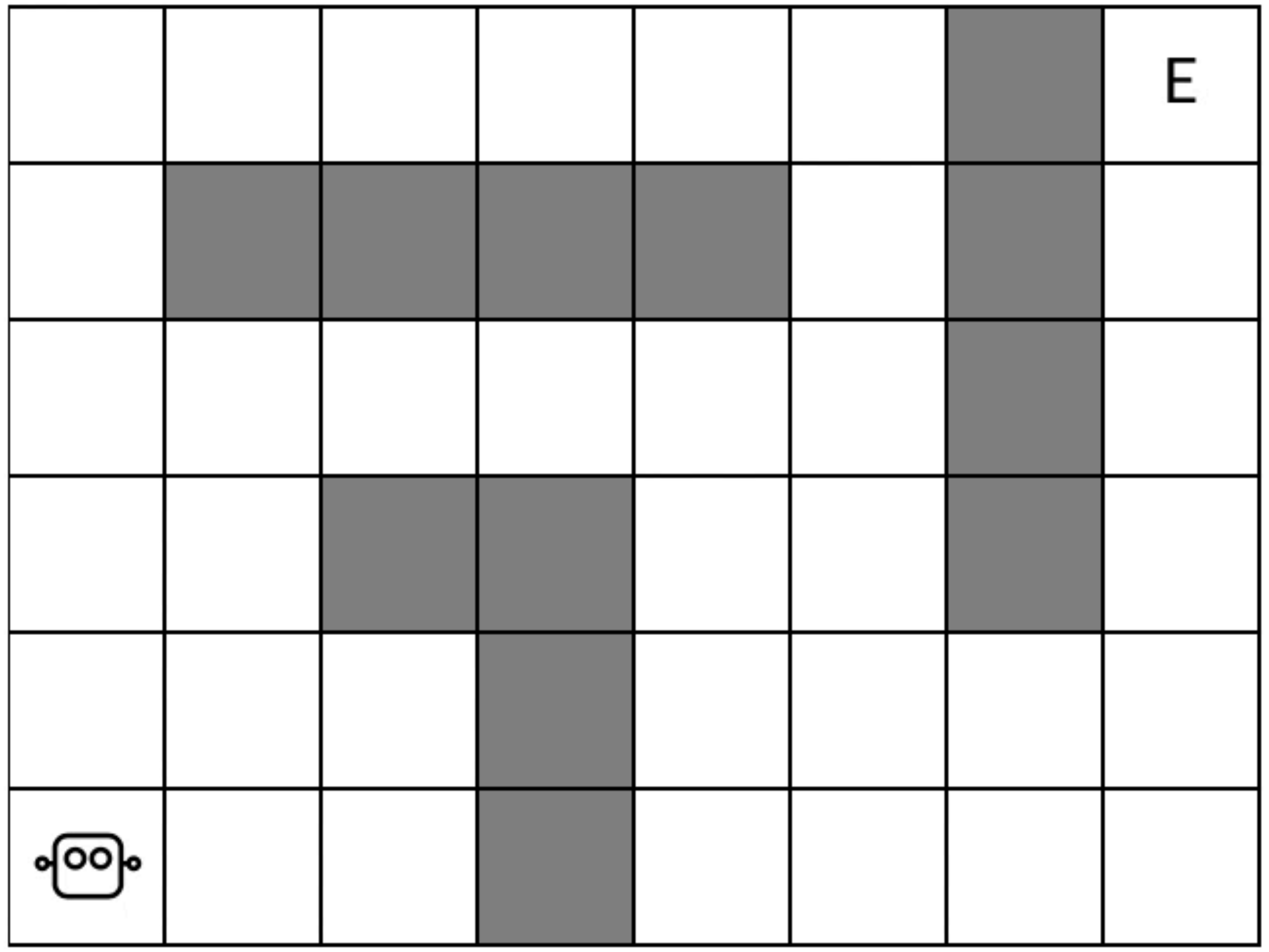
$A \leftarrow$ random action previously taken in S

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

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

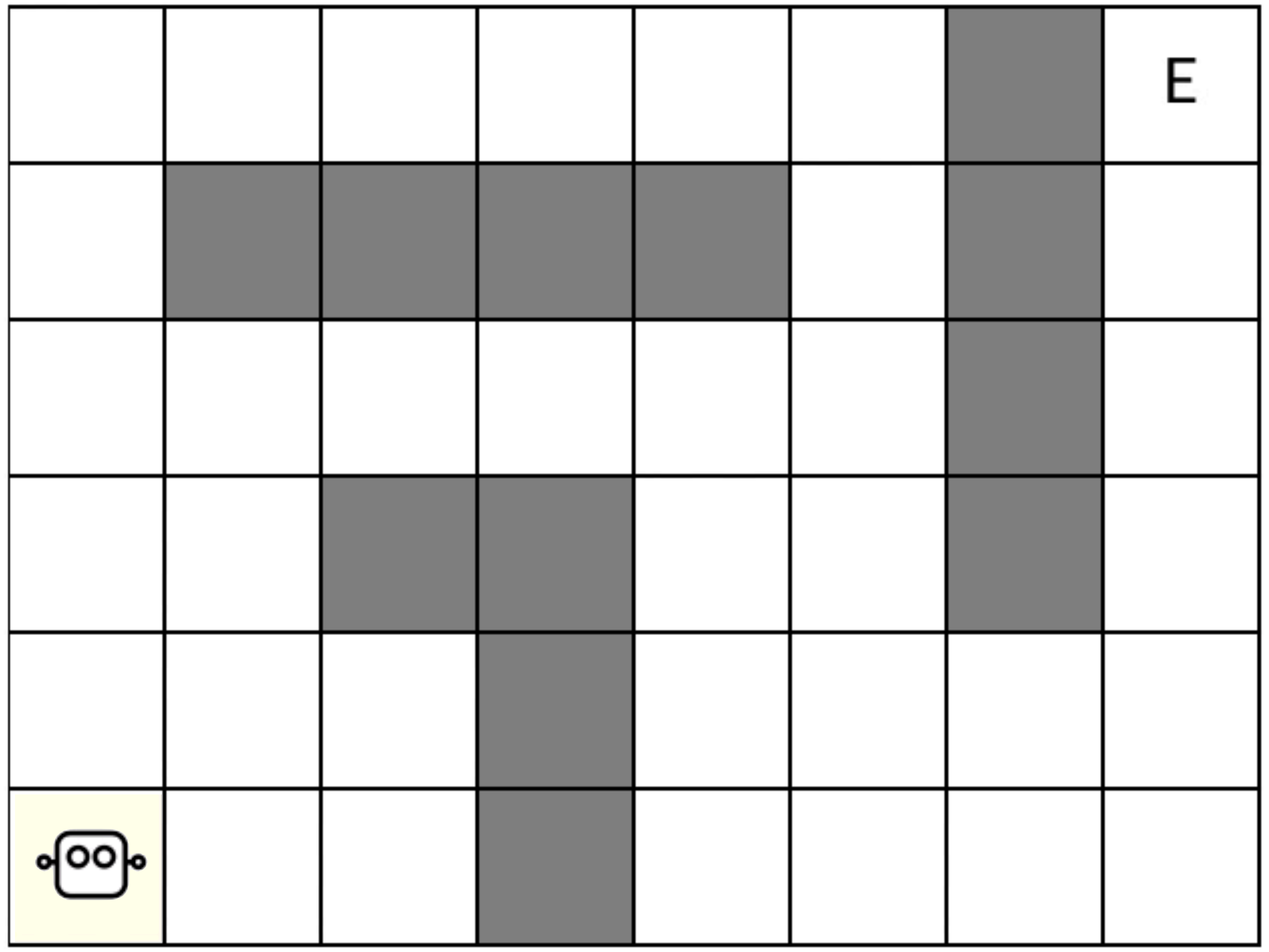
Let's see how much **better** an agent can do with **Dyna**

Agent in the first episode

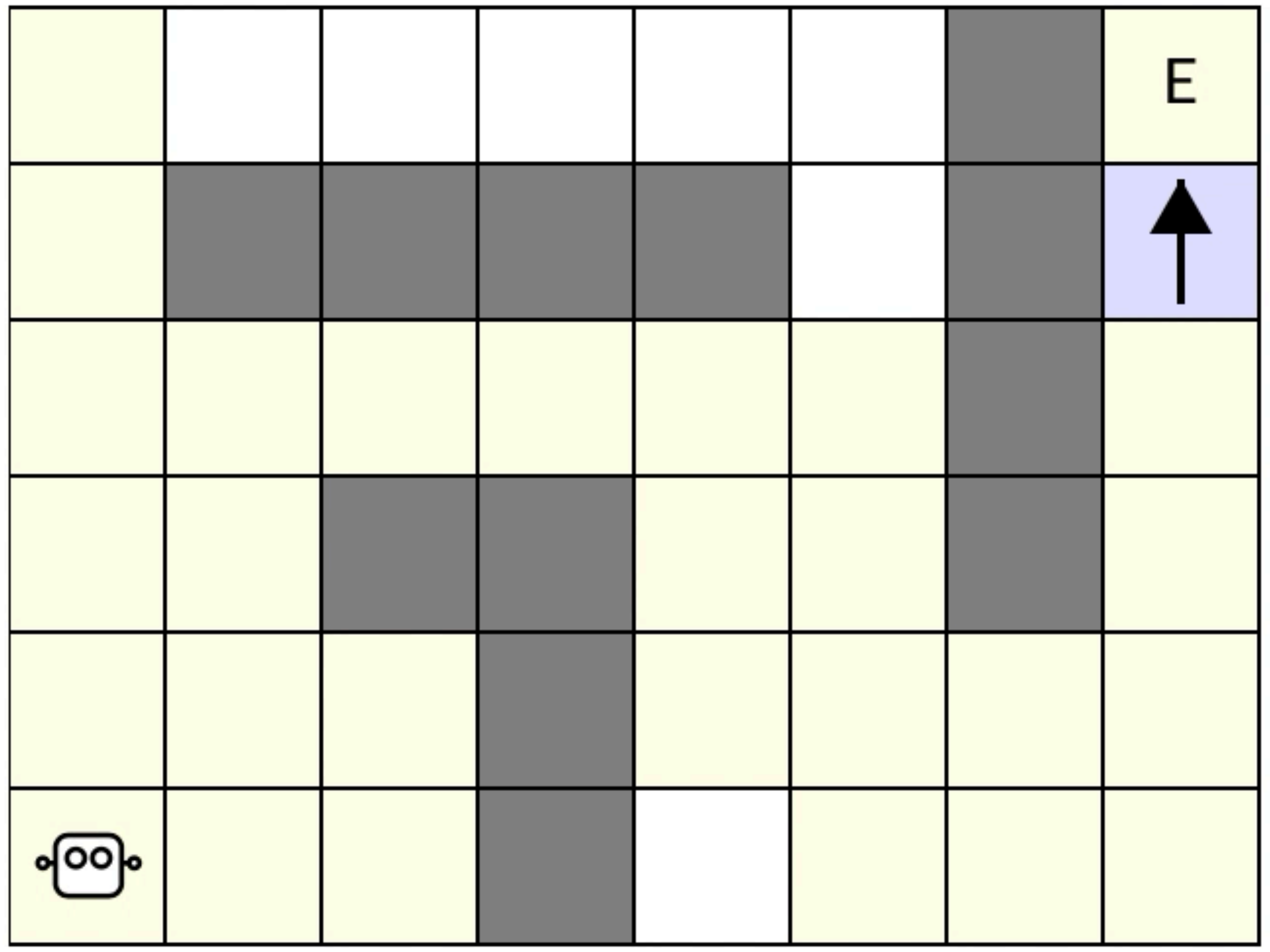


Agent in the first episode

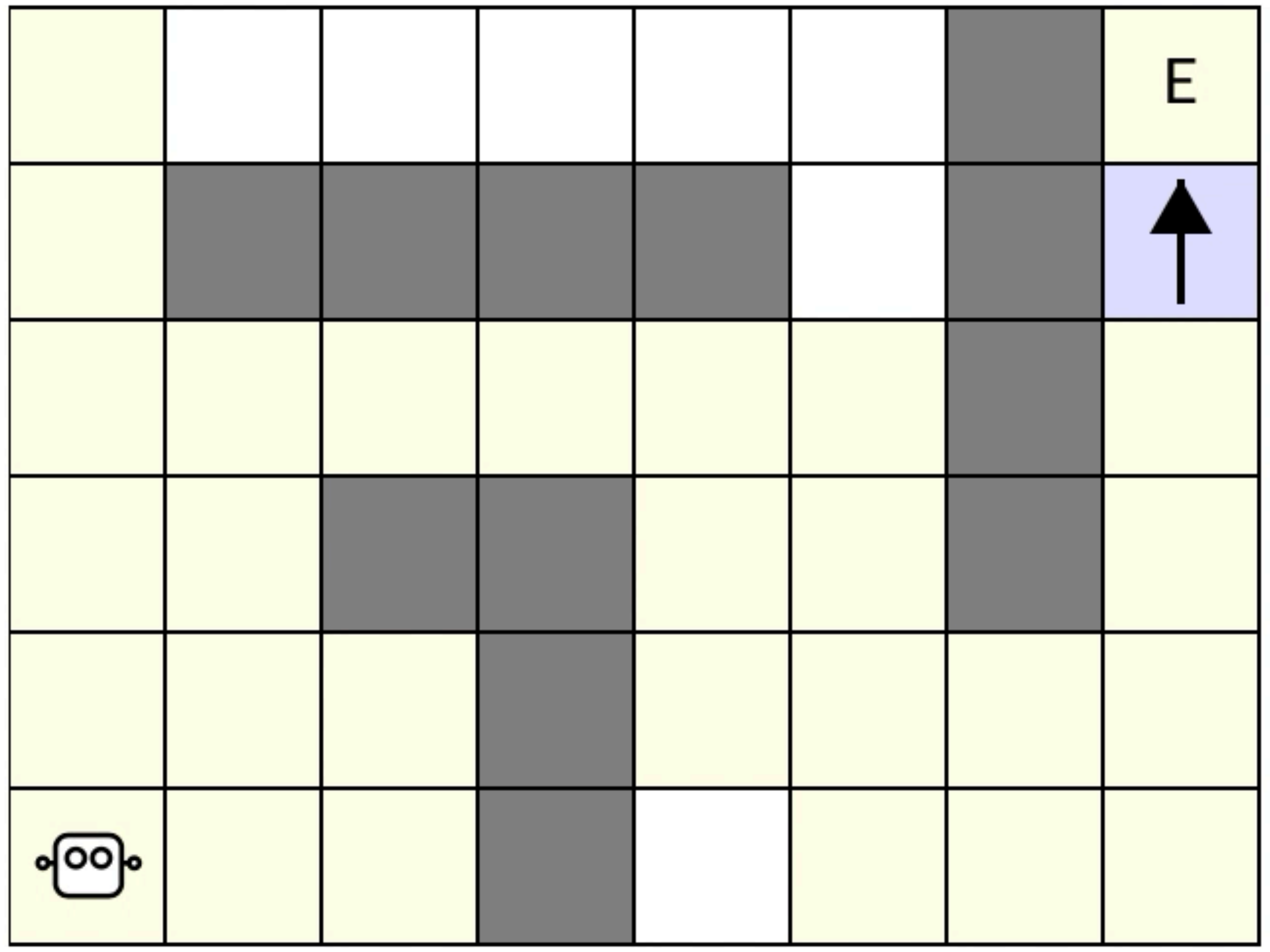
$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \cdot max_{a'} Q(s',a') - Q(s,a) \right)$$



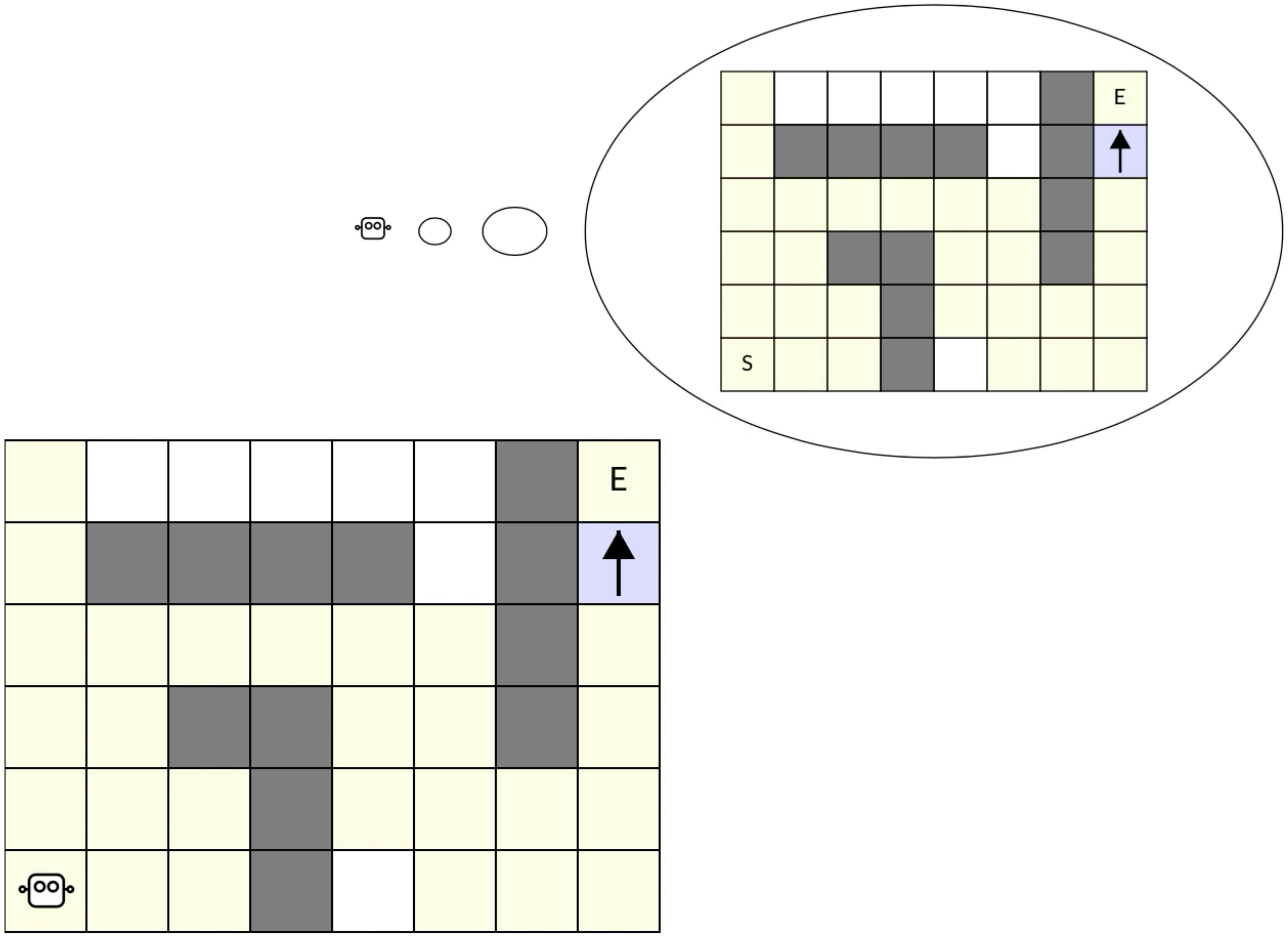
Agent's knowledge after the first episode



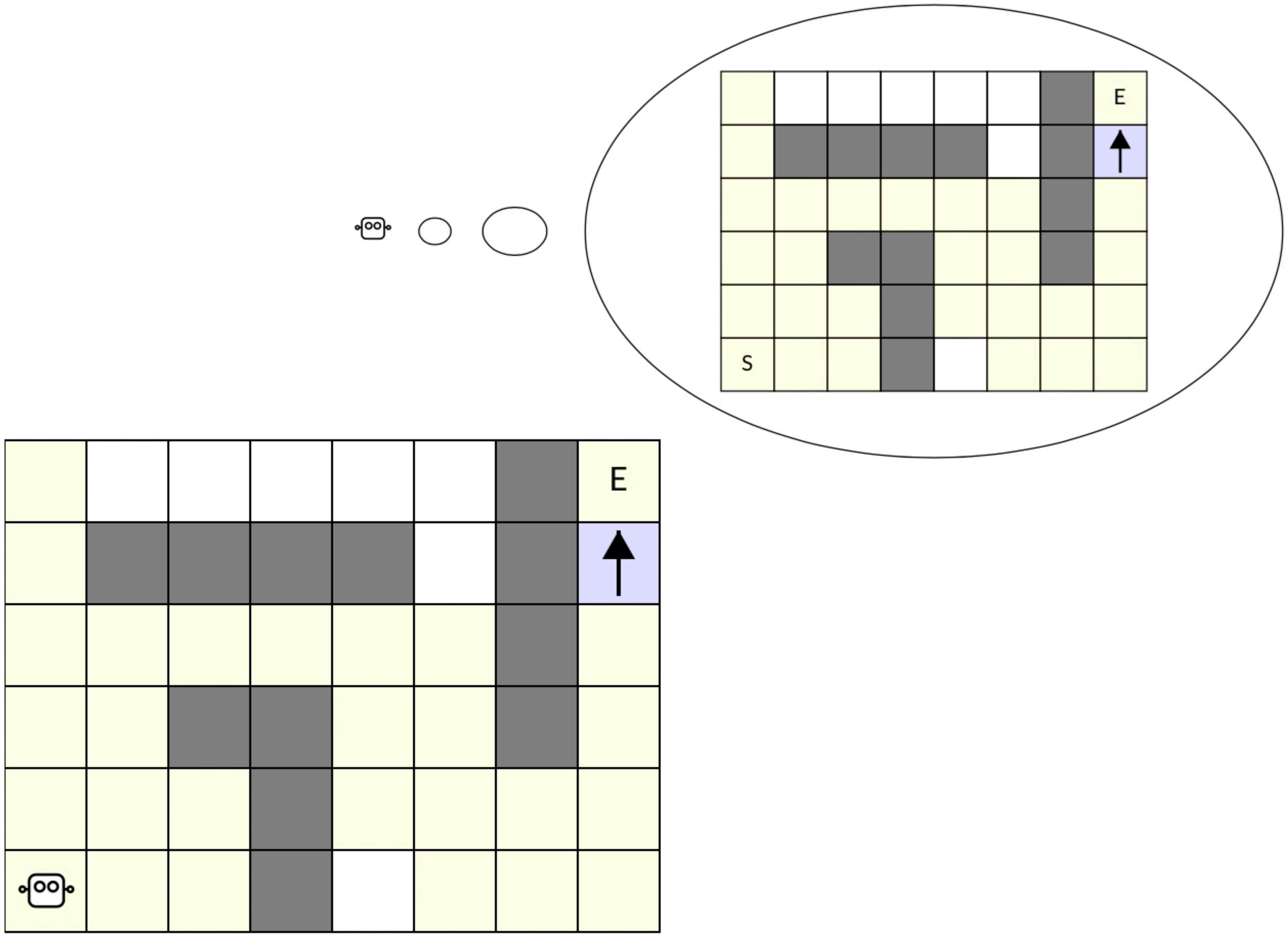
Agent’s knowledge after the first episode



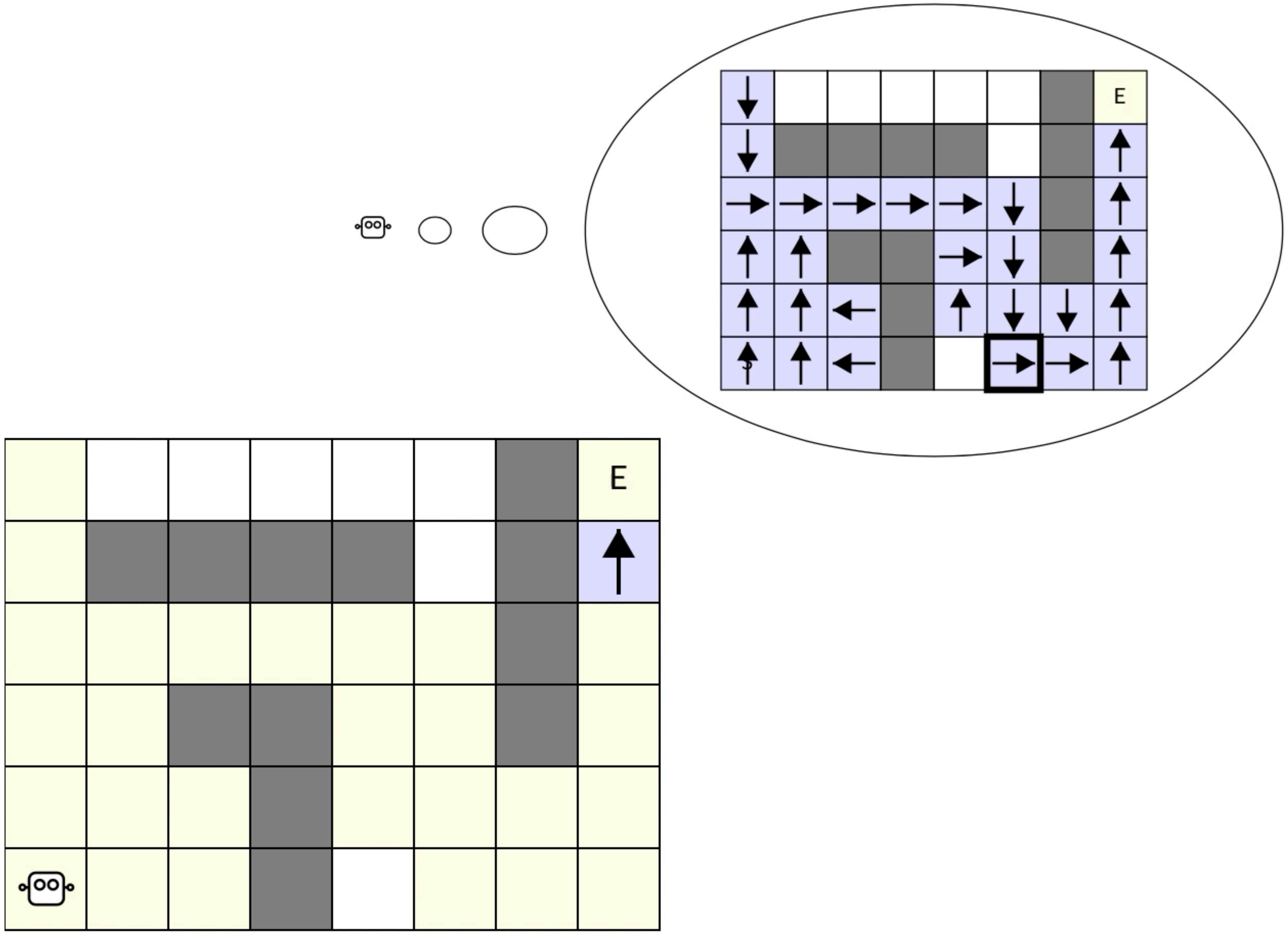
Agent using many planning steps in Dyna



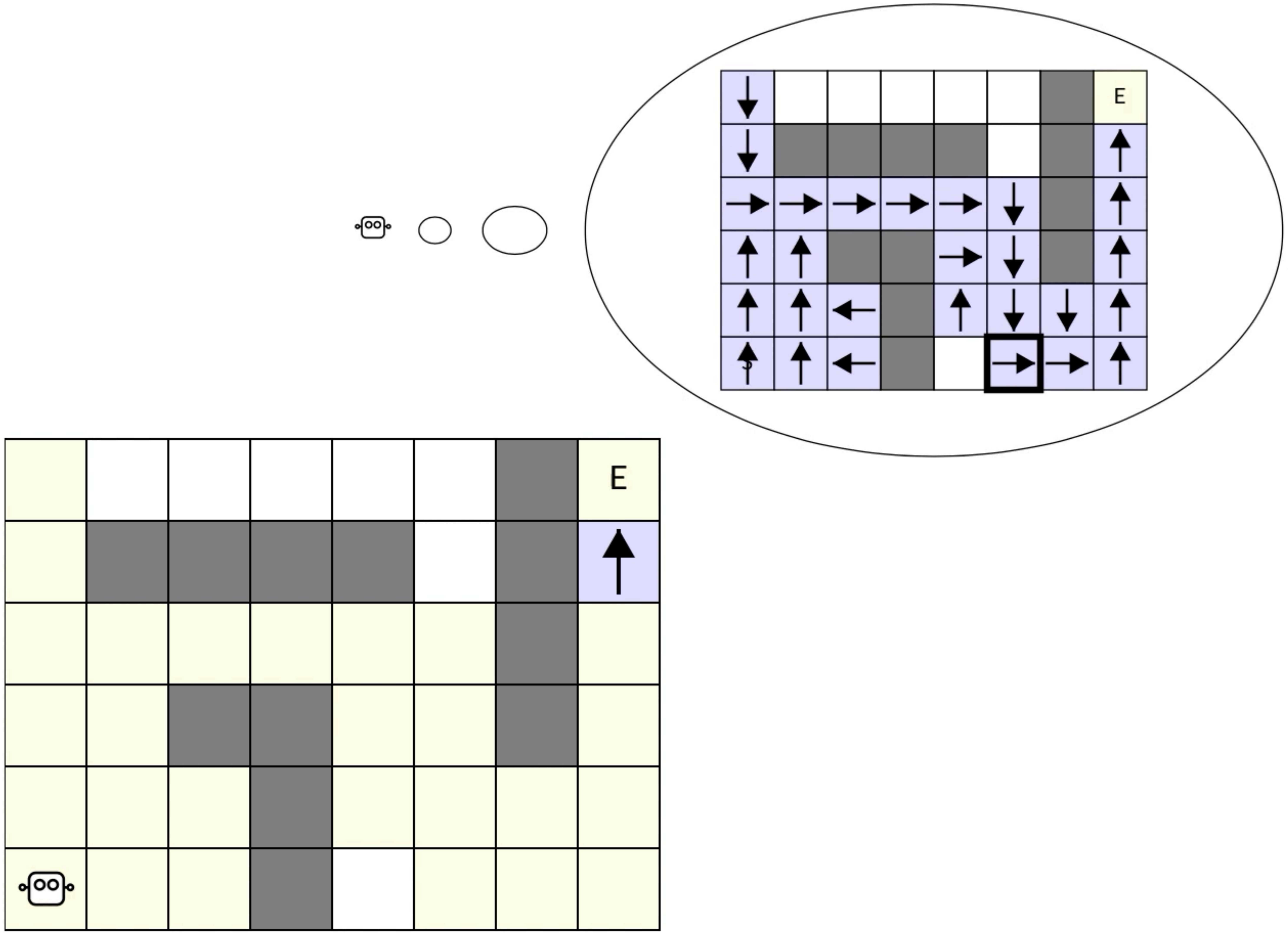
Agent using many planning steps in Dyna



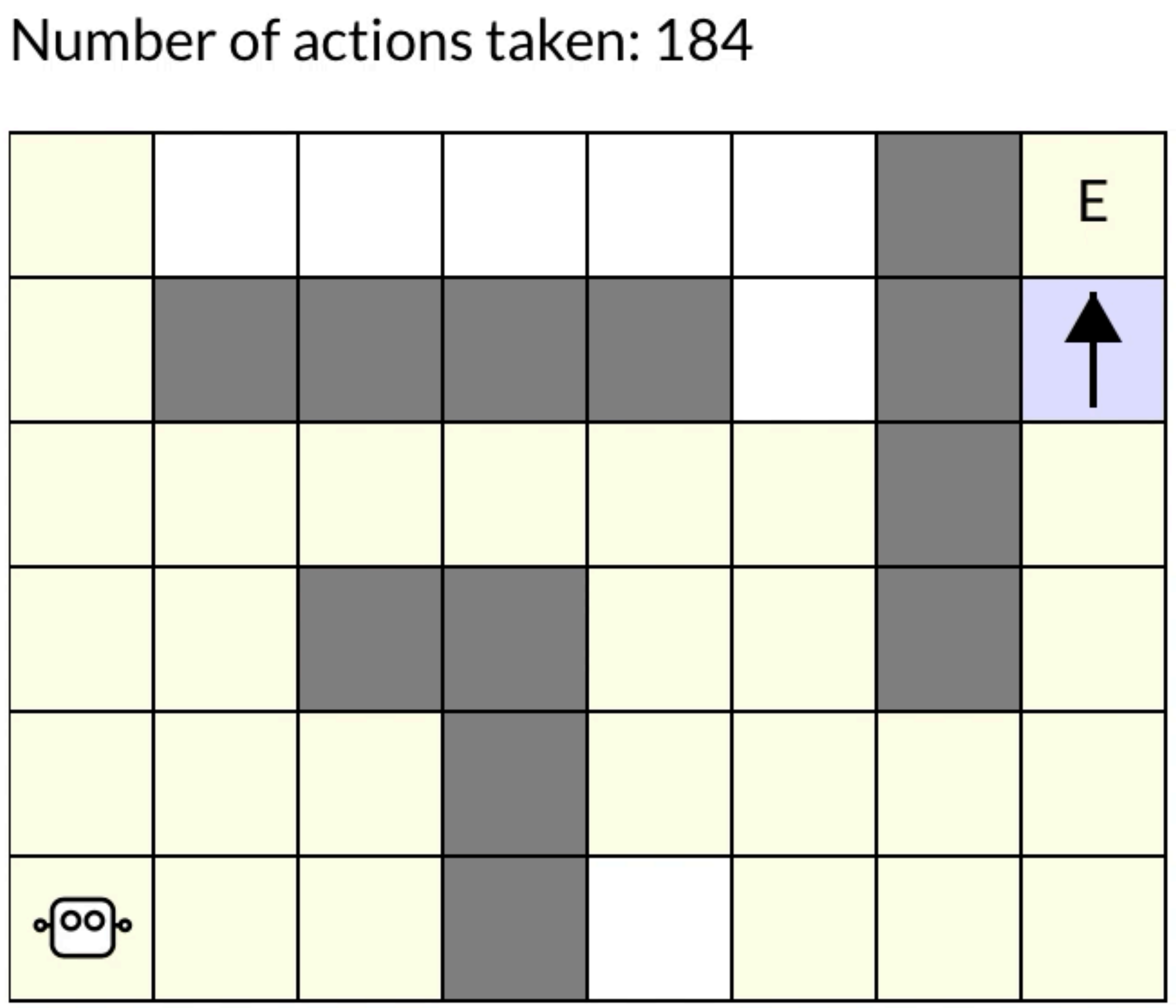
Agent has the optimal policy after just one episode



Agent has the optimal policy after just one episode

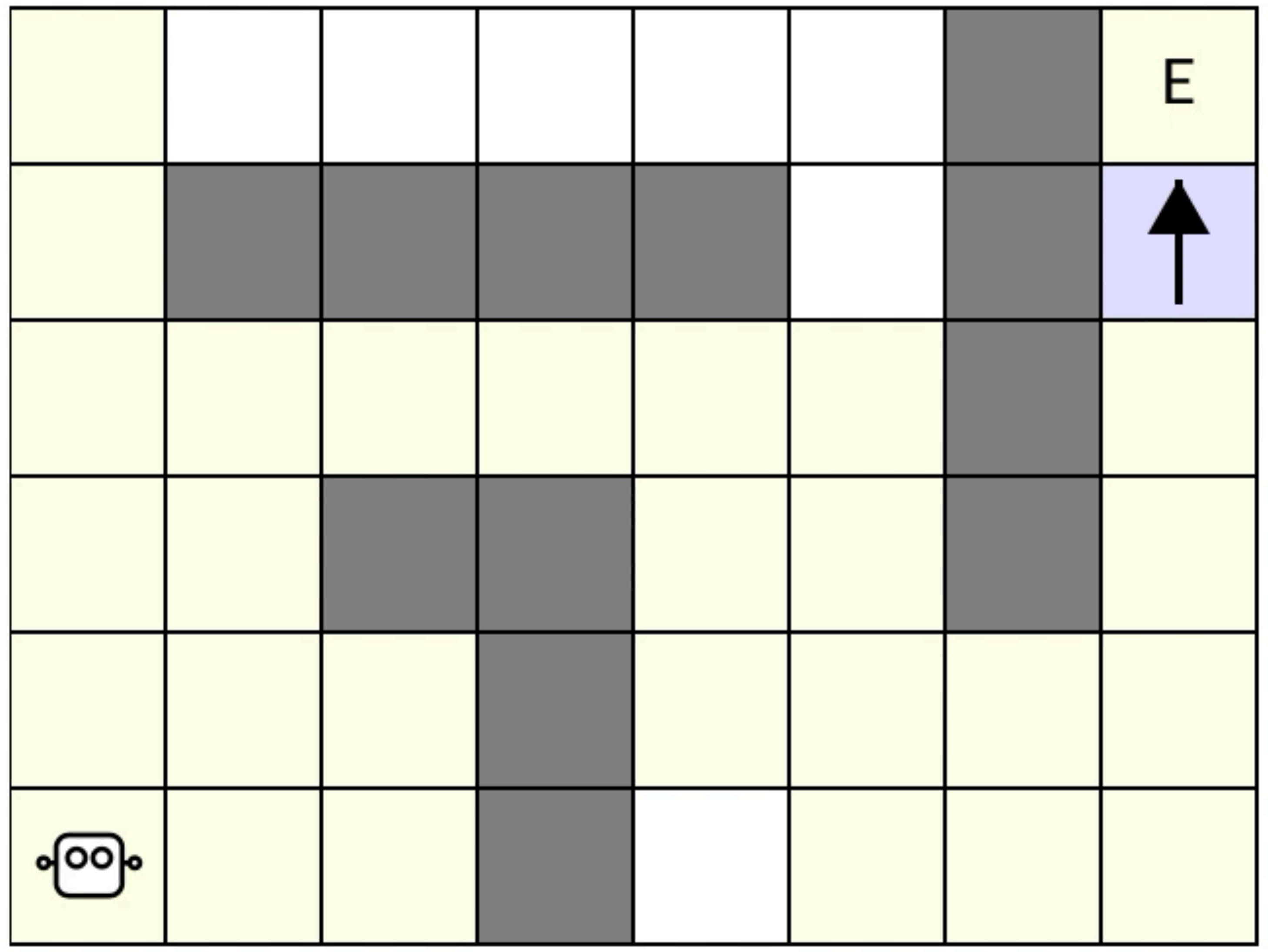


Interleaving planning and acting

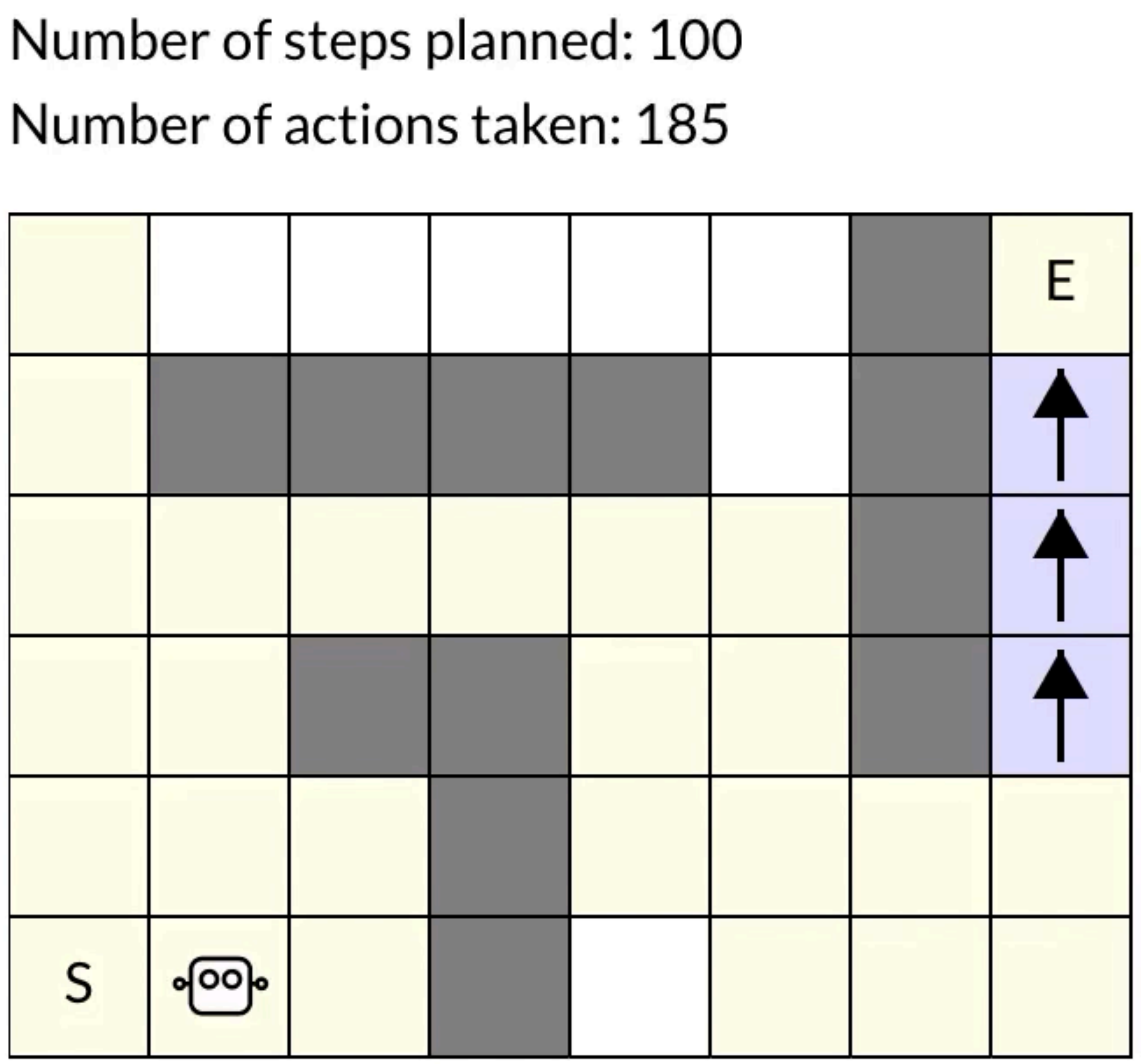


Interleaving planning and acting

Number of actions taken: 184



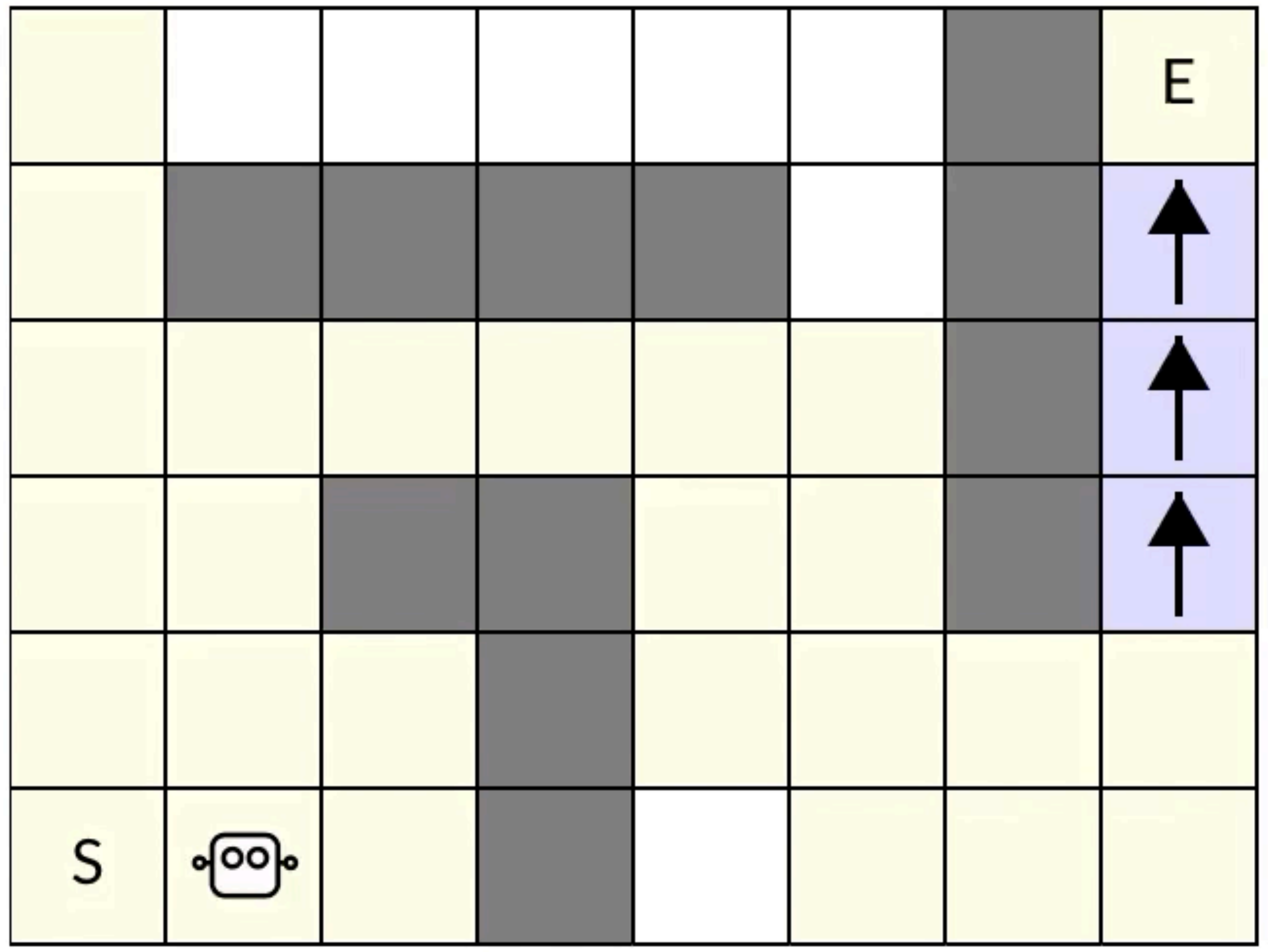
Interleaving planning and acting



Interleaving planning and acting

Number of steps planned: 100

Number of actions taken: 185



Dyna = Background Planning

- Given **unlimited computation**, each planning update in the background could essentially solve the Bellman equation for the current model
  - Loop over all states and actions many times
  - At extreme of computation, behaves like Dynamic Programming
- In practice, have **limited** computation
- Use any **extra computation** for **background planning**: do as many updates to the value function or policy as computation allows

Advantages of Dyna

- **Anytime** planning (asynchronous, occurs in the background)
  - contrasts Decision-time planning
- Can take advantage of **parallelism**
- Naturally enables **partial models**
- Can still do **long-term planning** use temporal abstraction, but avoids multi-step rollouts

Now let’s dive into specific instances of Dyna

Important choices

- The type of model
- Search-control

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) $Model(S, A) \leftarrow R, S'$
(f) Loop repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S, A)$
$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$

Important choices: Model

- The type of model
- Search-control

Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon\text{-greedy}(S,Q)$
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $\boxed{Model(S,A) \leftarrow R, S'}$
(f) Loop repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$\boxed{R, S' \leftarrow Model(S,A)}$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Important choices: Search Control

- The type of model
- Search-control

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

Let’s look at a **simple** example of **Dyna**:

Experience Replay

Experience Replay

- Essentially using a batch method in an online setting
- Store buffer of recent transitions (s, a, s’, r)
  - e.g., sliding window buffer
- Sample mini-batch updates from the buffer, for updates to the value function or policy

Exercise: How can ER be seen as an instance of Dyna?
What is the choice for the **Model** and for **Search Control**?

Experience Replay Pseudocode

Initialize buffer B and Q(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) Add (S, A) to buffer B, drop oldest sample

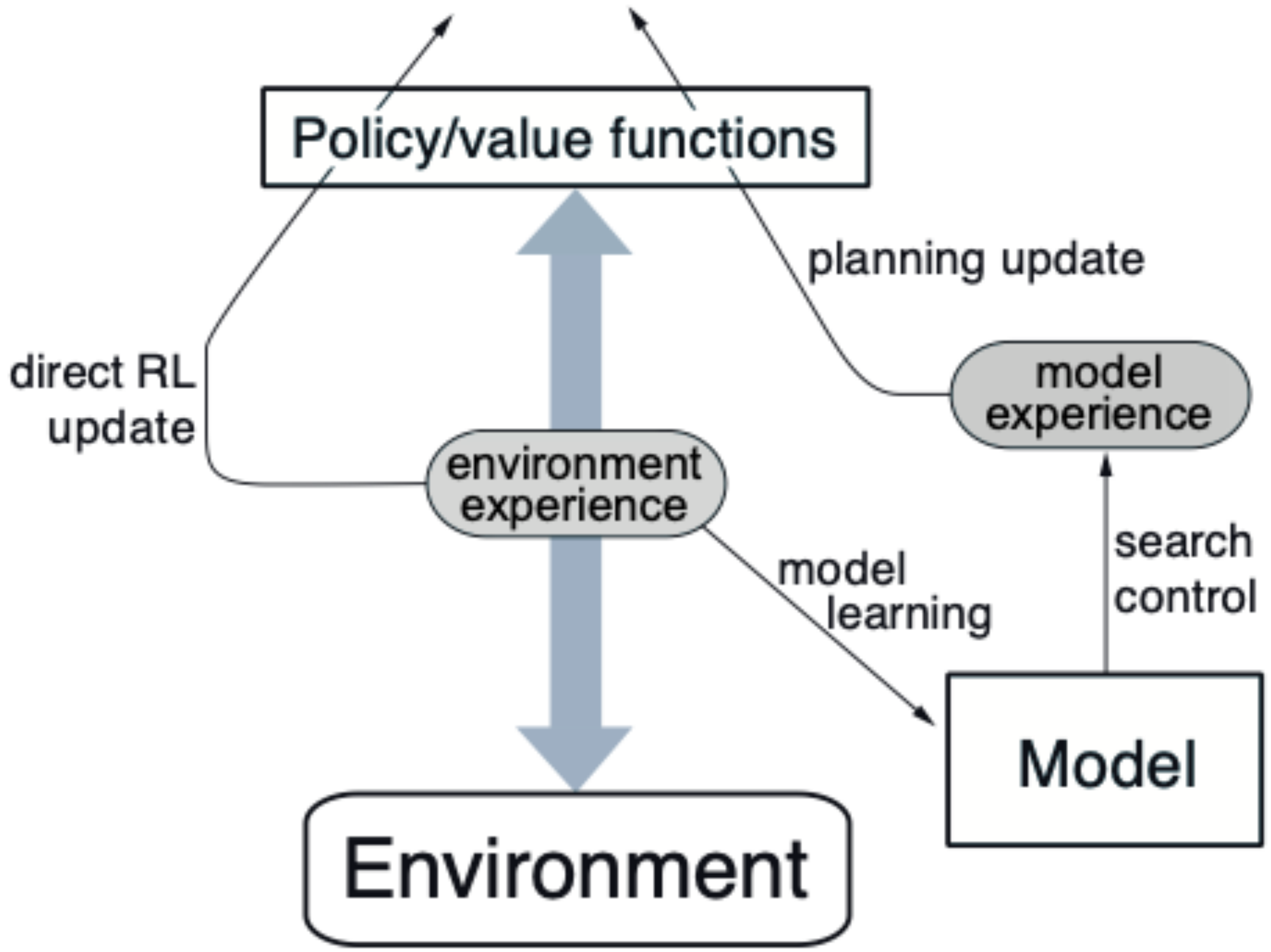
(f) Loop repeat n times:

Grab random (S, A, S', R) from buffer B

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

Exercise: How can ER be seen as an instance of Dyna?
What is the choice for the **Model** and for **Search Control**?

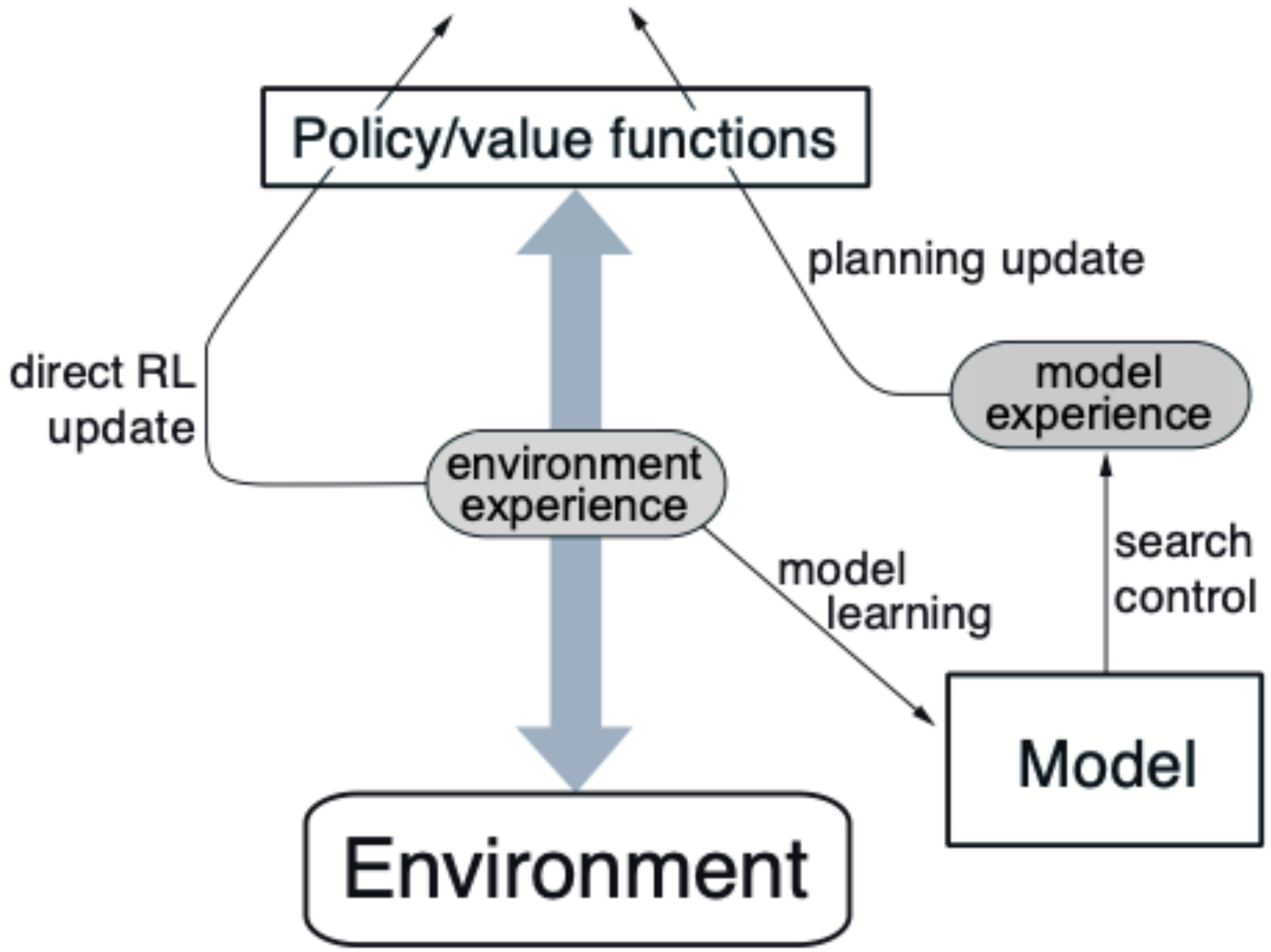
Experience Replay:
A Simple Example of Dyna



s', r ~ M(s,a)

(s,a)

Experience Replay: A Simple Example of Dyna



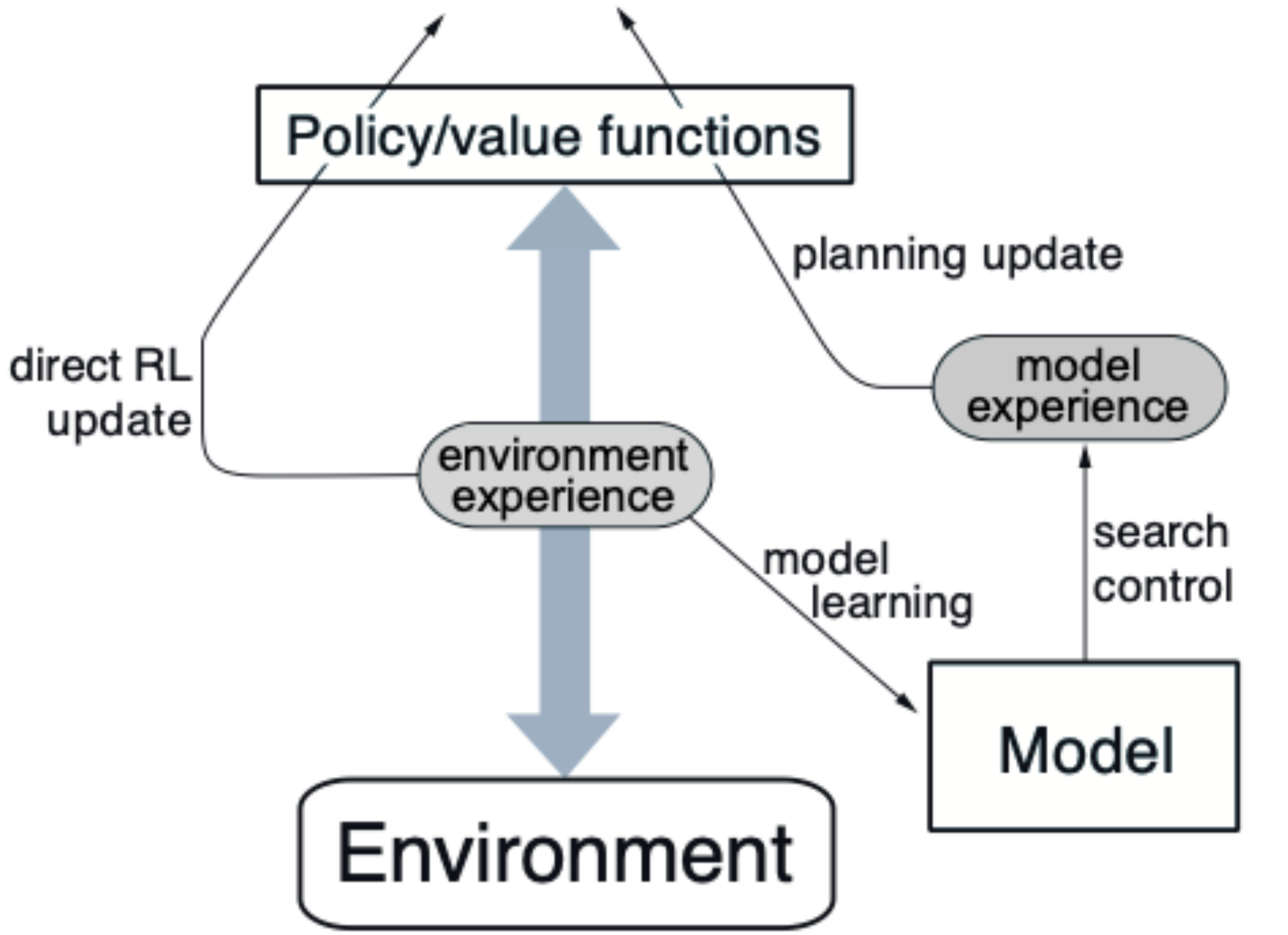
$s', r \sim M(s,a)$

(s,a)

Model is tuples of experience:

(s_0, a_0, r_1, s_1)
(s_1, a_1, r_2, s_2)
(s_2, a_2, r_3, s_3)
…

Experience Replay:
A Simple Example of Dyna



Model is tuples of experience:

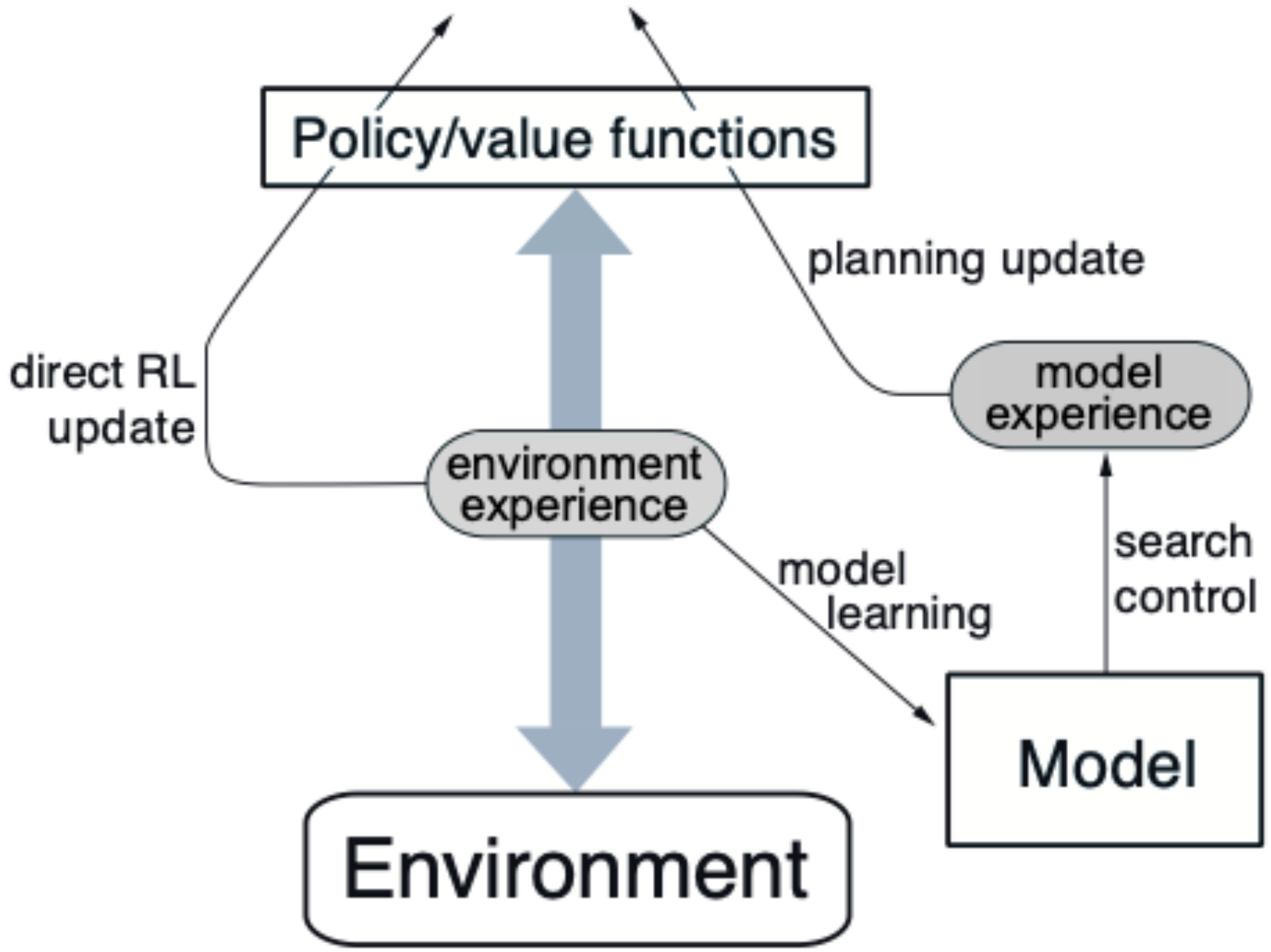
(s_0, a_0, r_1, s_1)

(s_1, a_1, r_2, s_2)

(s_2, a_2, r_3, s_3)

...

Experience Replay:
A Simple Example of Dyna



(r_{i+1}, s_{i+1})

(s_i, a_i)

Model is tuples of experience:

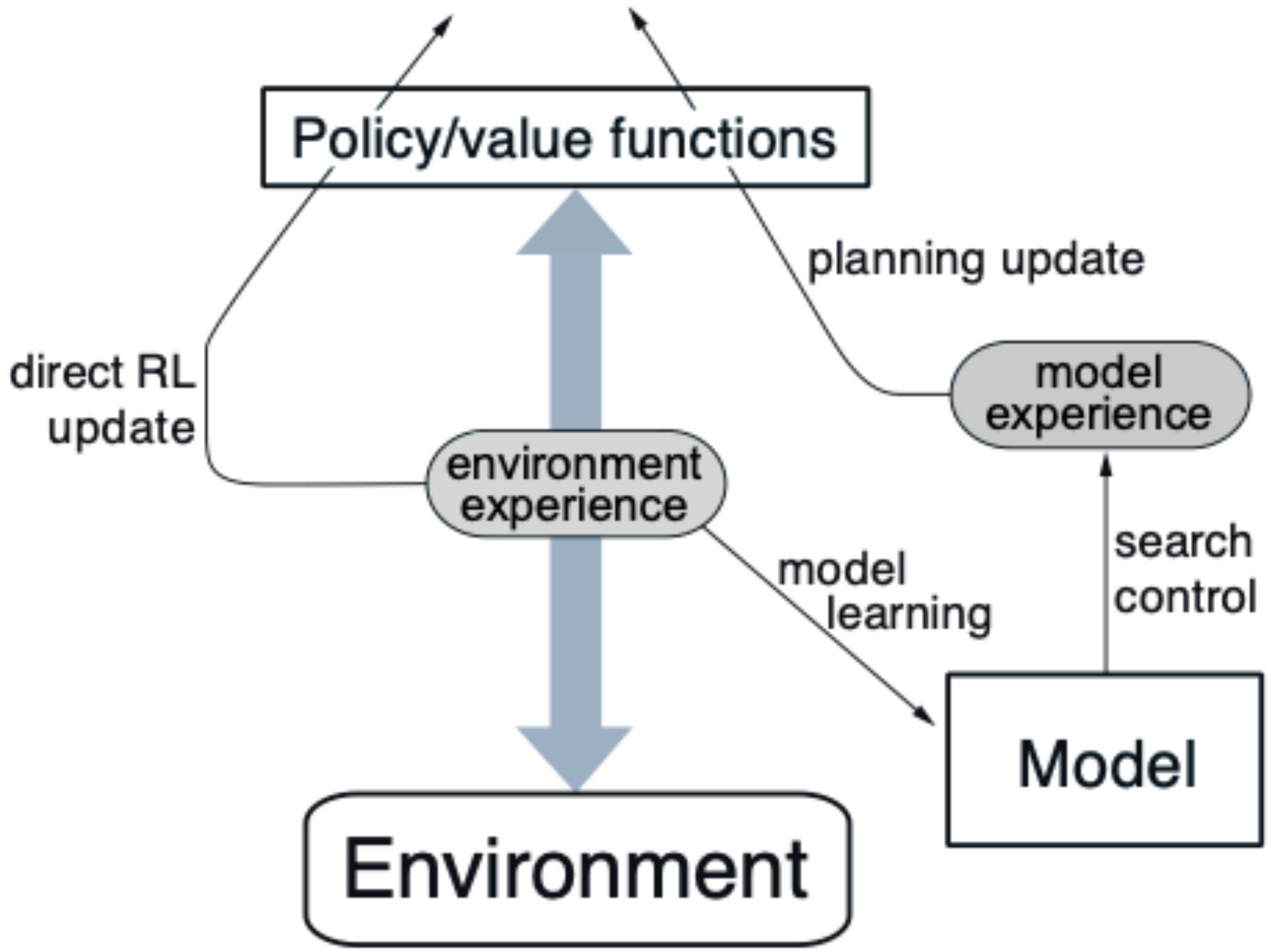
(s_0, a_0, r_1, s_1)

(s_1, a_1, r_2, s_2)

(s_2, a_2, r_3, s_3)

...

Experience Replay:
A Simple Example of Dyna



(r_{i+1}, s_{i+1})

(s_i, a_i)

Model is tuples of experience:

(s_0, a_0, r_1, s_1)

(s_1, a_1, r_2, s_2)

(s_2, a_2, r_3, s_3)

...

We should be able to get a better **Model**
and smarter **Search Control**

Advantages of a Learned Model over a Transition Buffer

- **Compactness**: summarizes experience
- **Coverage**: cannot store all experience, so in ER common to use most recent experience (does not cover space)
- **Querying**: can query a model from a particular (s,a)

Important choices: Model

- The type of model
- Search-control

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

What are possible learned models?

$$\hat{p}(s', r \mid s, a)$$

Increasing
Abstraction
and/or
Simplicity

What are possible learned models?

- Most obvious answer: $\hat{p}(s', r \mid s, a)$

Increasing
Abstraction
and/or
Simplicity

What are possible learned models?

- Most obvious answer: $\hat{p}(s', r \mid s, a)$
- Transition Model with **agent state**
  - agent state = constructed vector summarizing key information

$\downarrow$ **Increasing Abstraction and/or Simplicity**

What are possible learned models?

- Most obvious answer: $\hat{p}(s', r \mid s, a)$
- Transition Model with **agent state**
  - agent state = constructed vector summarizing key information
- Predictions about some **observations/features** in the future

Increasing Abstraction and/or Simplicity

What are possible learned models?

- Most obvious answer: $\hat{p}(s', r \mid s, a)$
- Transition Model with **agent state**
    - agent state = constructed vector summarizing key information
- Predictions about some **observations/features** in the future
- Predictions about **(cumulative) rewards** in the future

Increasing Abstraction and/or Simplicity

What are possible learned models?

- Most obvious answer: $\hat{p}(s', r \mid s, a)$
- Transition Model with **agent state**
  - agent state = constructed vector summarizing key information
- Predictions about some **observations/features** in the future
- Predictions about **(cumulative) rewards** in the future
- …or even **Q(s,a)**?

Increasing Abstraction and/or Simplicity

So is Sarsa a model-based RL algorithm?

- This question only arises due to being imprecise
- Let’s try to be more precise

What does the model do?

- The **agent** uses knowledge/predictions about the world (a **model**) to
  - improve estimates of the **optimal** value function/policy
  - learn about **new** things **faster**
    - e.g., learn new option policies (new skills)
    - e.g., help agent re-visit parts of the space in non-stationary problems

Notice now that Q(s,a) does not really count as a model

What does the model do?

- The **agent** uses knowledge/predictions about the world (a **model**) to
  - improve estimates of the **optimal** value function/policy
  - learn about **new** things **faster**
    - e.g., learn new option policies (new skills)
    - e.g., help agent re-visit parts of the space in non-stationary problems

Notice now that Q(s,a) does not really count as a model

But the model does not have to be the transition dynamics

Important aspects of the model

- **State-to-State vs Observation-to-Observation**

Models on Agent State

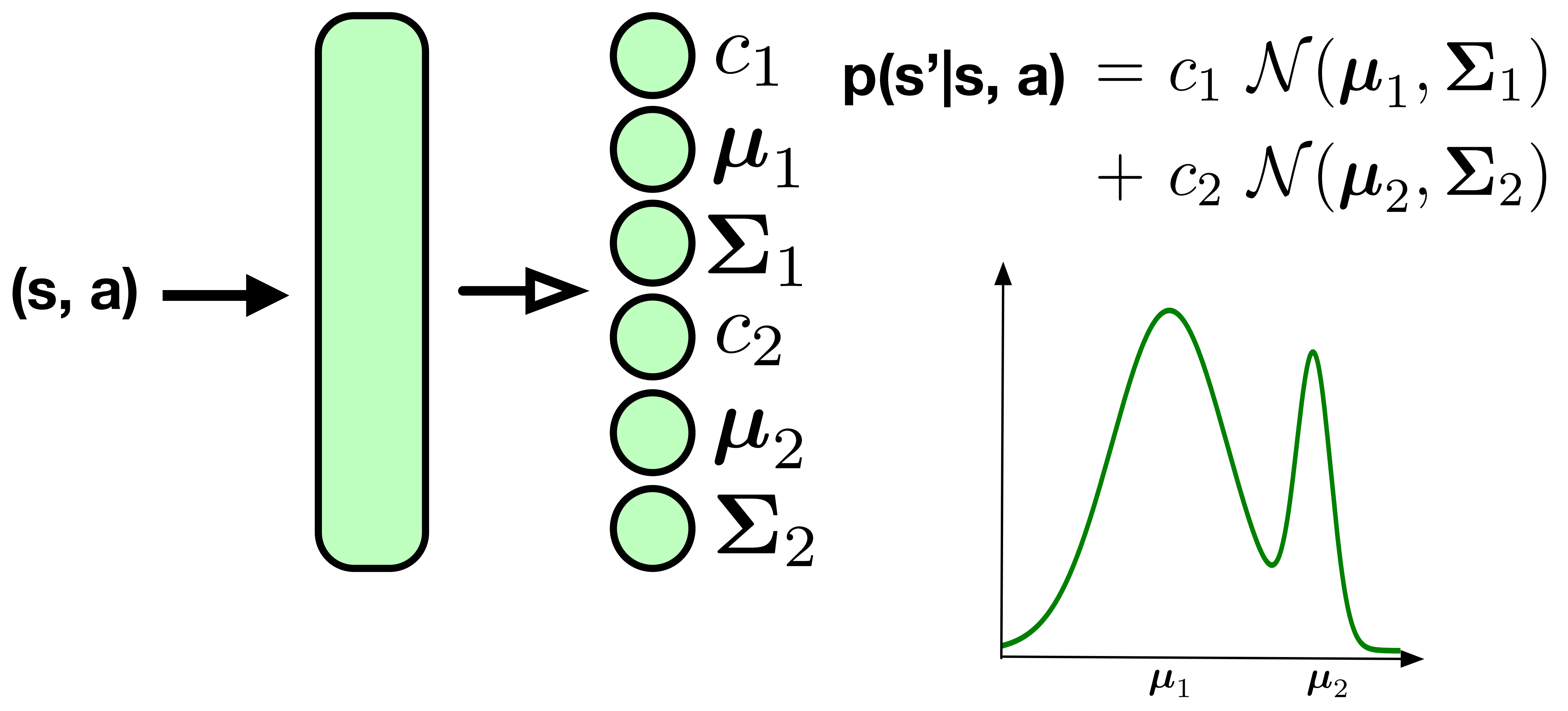
- Construct agent state $\hat{s}$
  - e.g., recurrent neural network to summarize history (POMDPs)
  - e.g., remove unnecessary detail from an image, only keep key info in the agent state needed to make predictions
- Learn one-step model for agent state $\hat{p}(\hat{s}', r \mid \hat{s}, a)$
- Only model what the agent thinks is important, avoid pixel-to-pixel models

Important aspects of the model

- State-to-State vs Observation-to-Observation
- **Expectation vs Sample Models**

Sample Models

- Given (s,a), obtain a **sample** of s' and r
- Examples:
  - Conditional Gaussian distribution
  - Conditional Mixture Model
  - Mixture Density Network



Expectation Model

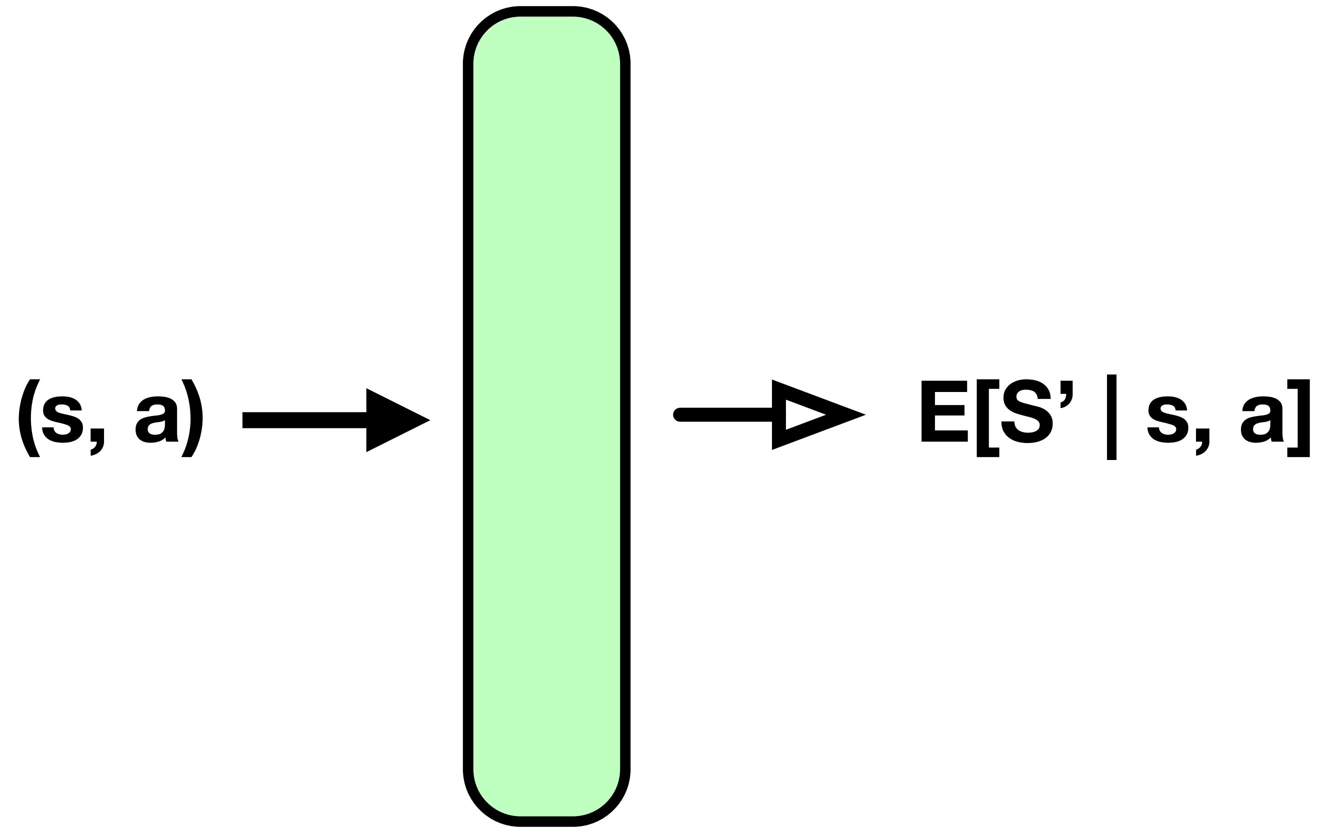
- Given (s,a), output **expected** next state and reward
- **Exercise**: Imagine you train a feedforward NN with input-output pairs ((s,a), (s', r)), with a squared error
- Would this result in a Sample Model or Expectation Model
  - or something else?

Expectation Model

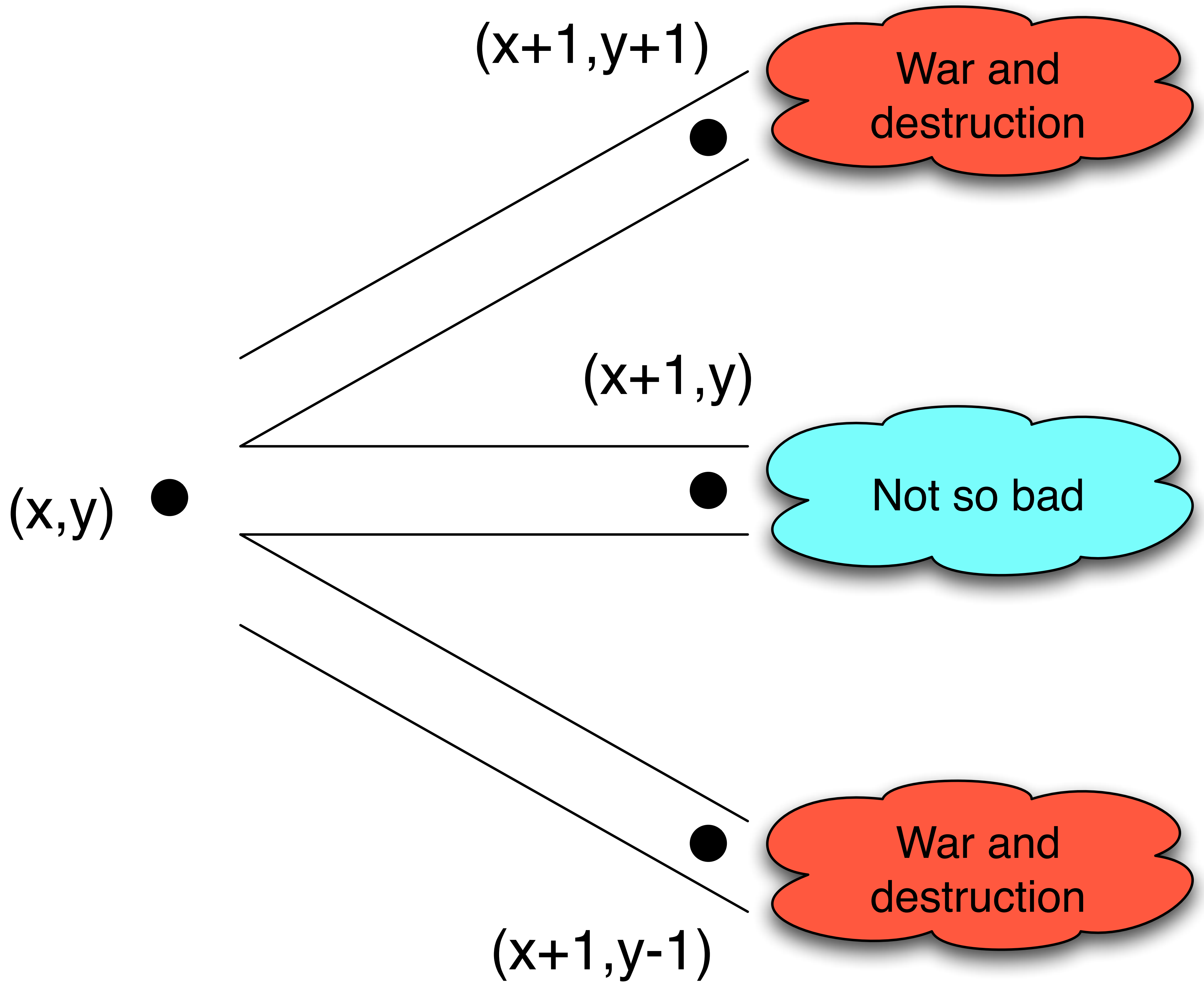
- Given (s,a), output expected next state and reward
- **Exercise**: Imagine you train a feedforward NN with input-output pairs ((s,a), (s', r)), with a squared error
- Would this result in a Sample Model or Expectation Model
  - or something else?
- **Answer**: Expectation Model

Expectation Model

- Given (s,a), output **expected** next state and reward
- Examples:
    - Linear function of (features of) (s,a)
    - Neural Network

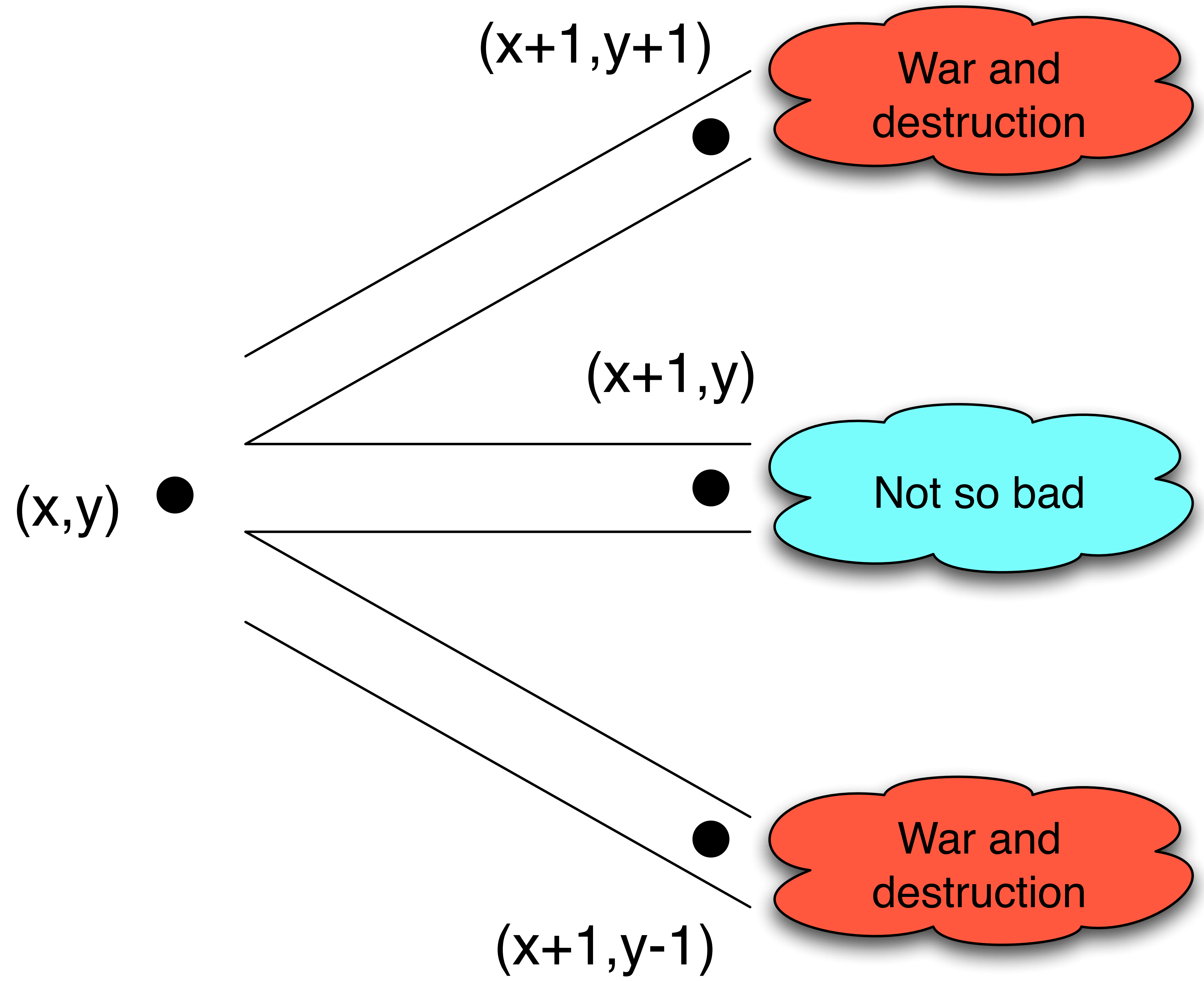


Potential Issues with an Expectation Model



* from Cosmin Paduraru's nice thesis, "Planning with Approximate and Learned Models of Markov Decision Processes"

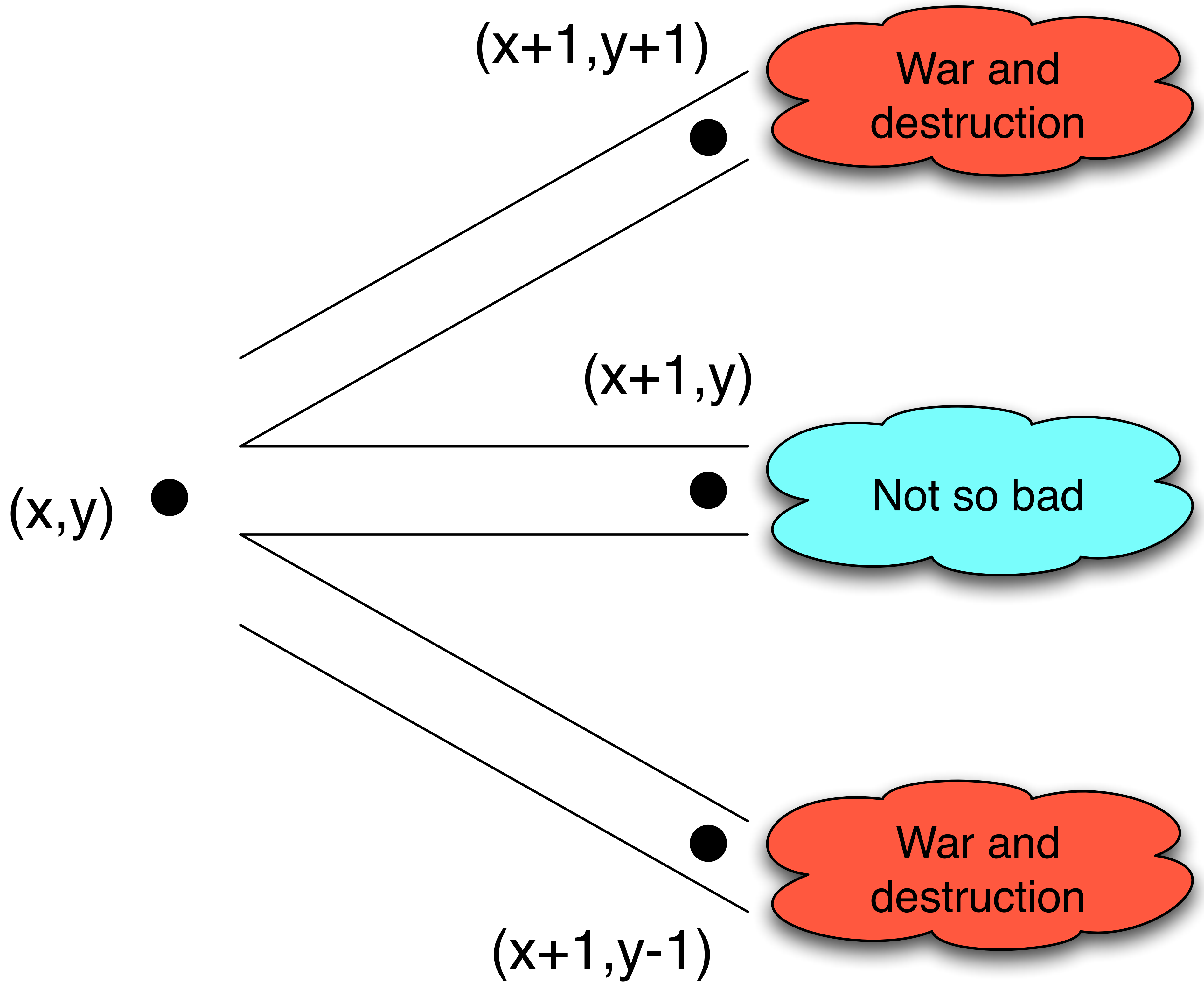
Potential Issues with an Expectation Model



$$
\begin{aligned}
&\mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \\
&= \mathbb{E}[R_{t+1} | S_t = s] + \gamma \mathbb{E}[v(S_{t+1}) | S_t = s] \\
&\neq \mathbb{E}[R_{t+1} | S_t = s] + \gamma v(\mathbb{E}[S_{t+1} | S_t = s])
\end{aligned}
$$

*** from Cosmin Paduraru's nice thesis, "Planning with Approximate and Learned Models of Markov Decision Processes"**

Potential Issues with an Expectation Model



$$\mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$
$$= \mathbb{E}[R_{t+1} | S_t = s] + \gamma \underline{\mathbb{E}[v(S_{t+1}) | S_t = s]}$$
$$\neq \mathbb{E}[R_{t+1} | S_t = s] + \gamma \underline{v(\mathbb{E}[S_{t+1} | S_t = s])}$$

*** from Cosmin Paduraru's nice thesis, "Planning with Approximate and Learned Models of Markov Decision Processes"**

Some Benefits of an Expectation Model

- Likely simpler to learn
  - Modeling entire distributions more difficult than just statistics like the mean
- If the world is deterministic, then expectation model = sample model
  - …maybe this is not so unreasonable
- If the value function is linear in agent state, then there is no disadvantage to using an expectation model
  - See “Planning with Expectation Models”, Wan et al, 2019

Important aspects of the model

- State-to-State vs Observation-to-Observation
- Expectation vs Sample Models
- **Rollouts vs Temporal Abstraction**

Issues with Rollouts

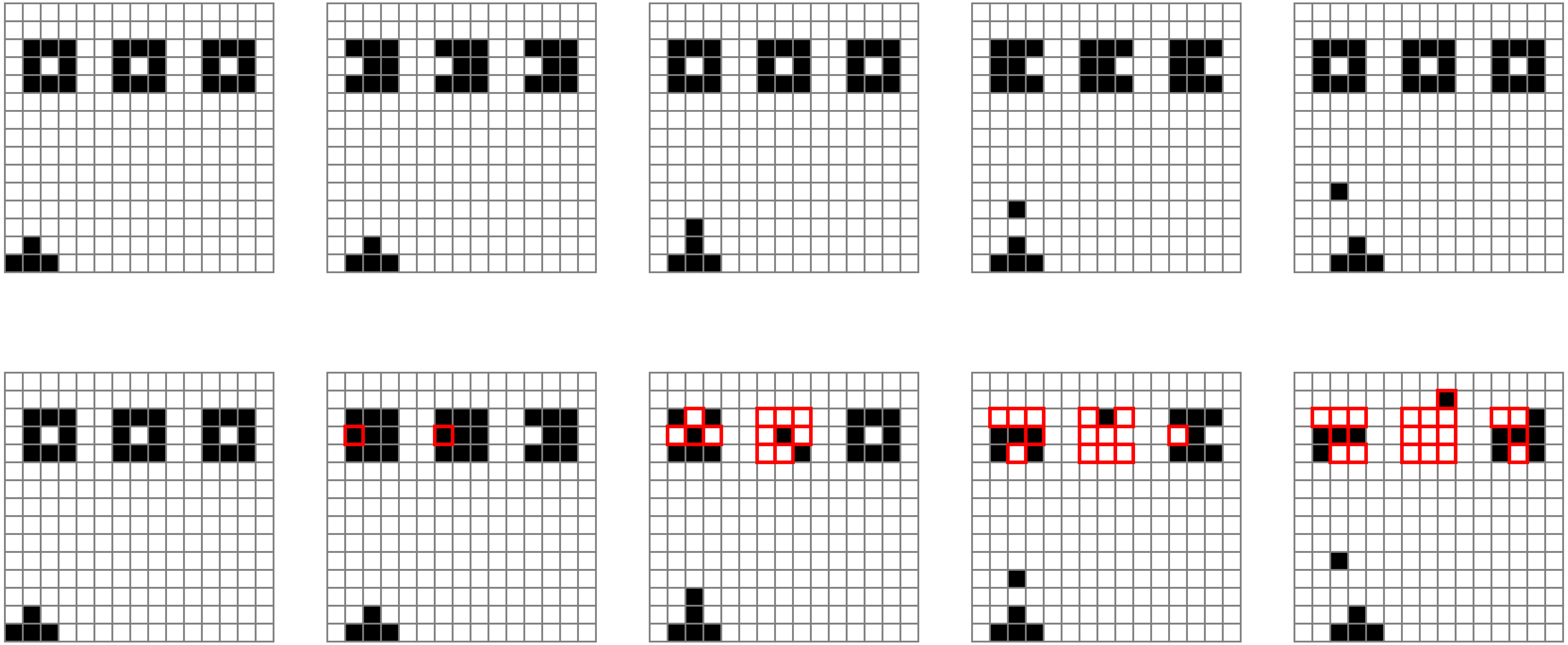


*image from Erin Talvitie, “Self-Correcting Models for Model-Based Reinforcement Learning”, 2017

Temporal Abstraction

- Use options to define **macro-actions**
  - e.g., Imagine a navigation robot. It could have a policy that tells it how to get to the door (policy defined by option, Andre will talk about this more)
- Agent can plan over options, $\hat{p}(s', r \mid s, \pi)$
  - e.g., can ask: "What is the resulting agent-state and (accumulated) reward from a given agent-state when following the option policy?"
- **Advantage**: Can reason about longer horizons (multiple steps into future)
  - Without rolling out the model many steps

Important aspects of the model

- State-to-State vs Observation-to-Observation
- Expectation vs Sample models
- Rollouts vs Temporal abstraction
- Full transition dynamics or a subset of predictions about the future
- Whether model outputs certainty estimates
- Sample efficiency in learning the model
- Computational efficiency for querying/sampling from the model

Important aspects of the model

- State-to-State vs Observation-to-Observation
- Expectation vs Sample models
- Rollouts vs Temporal abstraction
- Full transition dynamics or a subset of predictions about the future
- Whether model outputs certainty estimates
- Sample efficiency in learning the model
- Computational efficiency for querying/sampling from the model

Any other suggestions?

Important choices: Search Control

- The type of model
- Search-control

Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon\text{-greedy}(S,Q)$
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
(e) $Model(S,A) \leftarrow R, S'$
(f) Loop repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

Search Control strategies: How to pick (s,a)

Search Control strategies: How to pick (s,a)

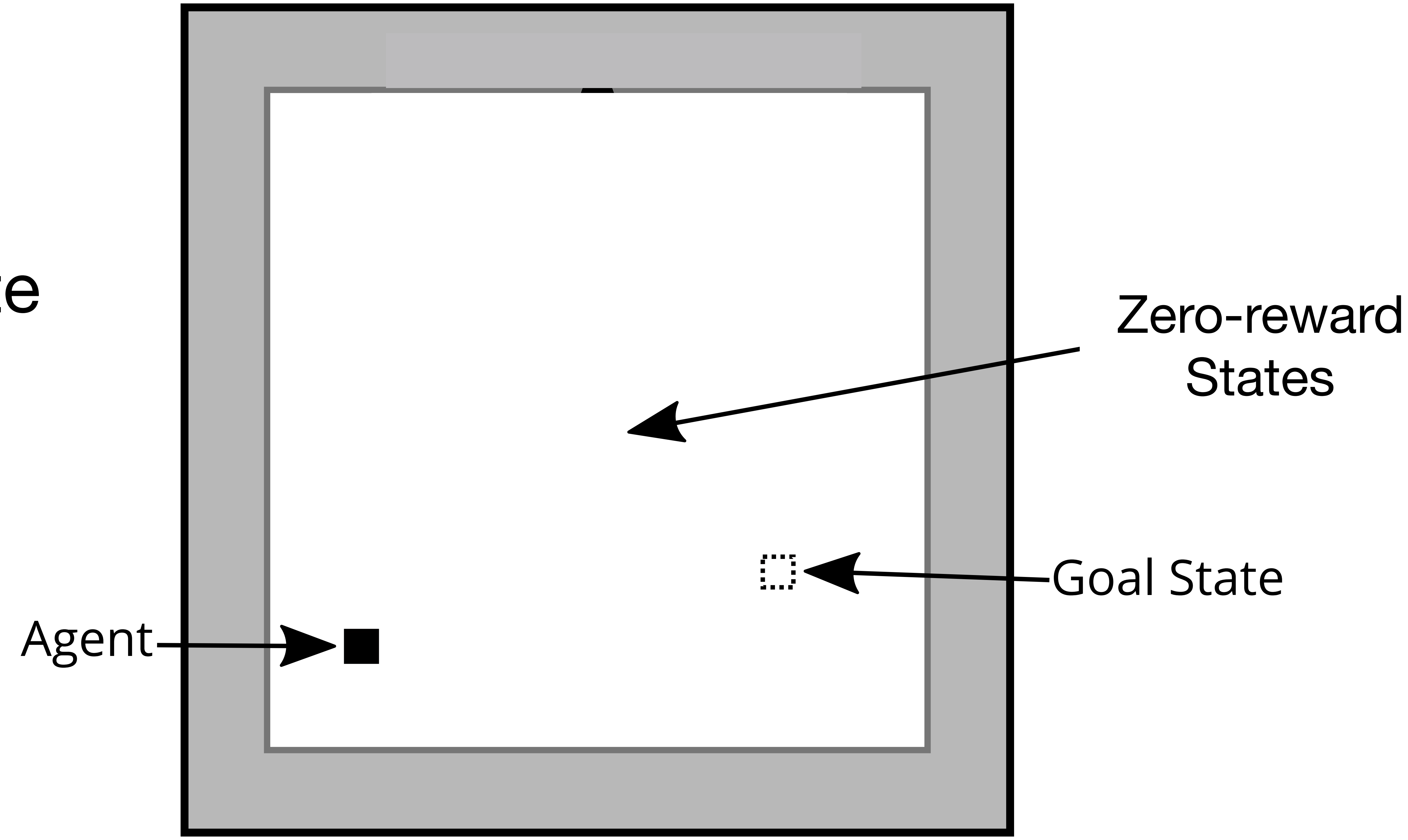
- Prioritize samples with high error
  - e.g., store observed (s,a) and associated TD-error

Search Control strategies: How to pick (s,a)

- Prioritize samples with high error
  - e.g., store observed (s,a) and associated TD-error
- Update backwards from “important” states
  - e.g., generate predecessor states from state with high TD-error

The utility of updating with predecessor states

- Imagine agent initializes values to zero
- Updates in the center are all zero!
- Then imagine it reaches the Goal State and transitions back to the Start
- What happens if the agent updates around the start state now?
- What happens if the agent updates predecessors around the goal?



Search Control strategies:
How to pick (s,a)

Search Control strategies: How to pick (s,a)

- Prioritize samples with high error
    - e.g., store observed (s,a) and associated TD-error
- Update backwards from “important” states
    - e.g., generate predecessor states from state with high TD-error

Search Control strategies: How to pick (s,a)

- Prioritize samples with high error
  - e.g., store observed (s,a) and associated TD-error
- Update backwards from “important” states
  - e.g., generate predecessor states from state with high TD-error
- Update with on-policy transitions
  - e.g., store observed states s and query current policy for action a

Search Control strategies: How to pick (s,a)

- Prioritize samples with high error
  - e.g., store observed (s,a) and associated TD-error
- Update backwards from “important” states
  - e.g., generate predecessor states from state with high TD-error
- Update with on-policy transitions
  - e.g., store observed states s and query current policy for action a
- See “Organizing experience: a deeper look at replay mechanisms for sample-based planning in continuous state domains”, Pan et al, 2018

Search Control strategies: How to pick (s,a)

- Prioritize samples with high error
- Update backwards from “important” states
- Update with on-policy transitions
- **Update current state or nearby region around current state**
  - improve values right before they are used
  - see Dyna-2 (Silver et al., 2016), “Hill Climbing on Value Estimates for Search-control in Dyna”, Pan et al., 2019

Search Control strategies: How to pick (s,a)

- Prioritize samples with high error
- Update backwards from “important” states
- Update with on-policy transitions
- Update current state or nearby region around current state

Search Control strategies: How to pick (s,a)

- Prioritize samples with high error
- Update backwards from “important” states
- Update with on-policy transitions
- Update current state or nearby region around current state

Any other suggestions?

So, are we done?

So, are we done?

- Can we just learn an accurate model with a deep NN and use Dyna?

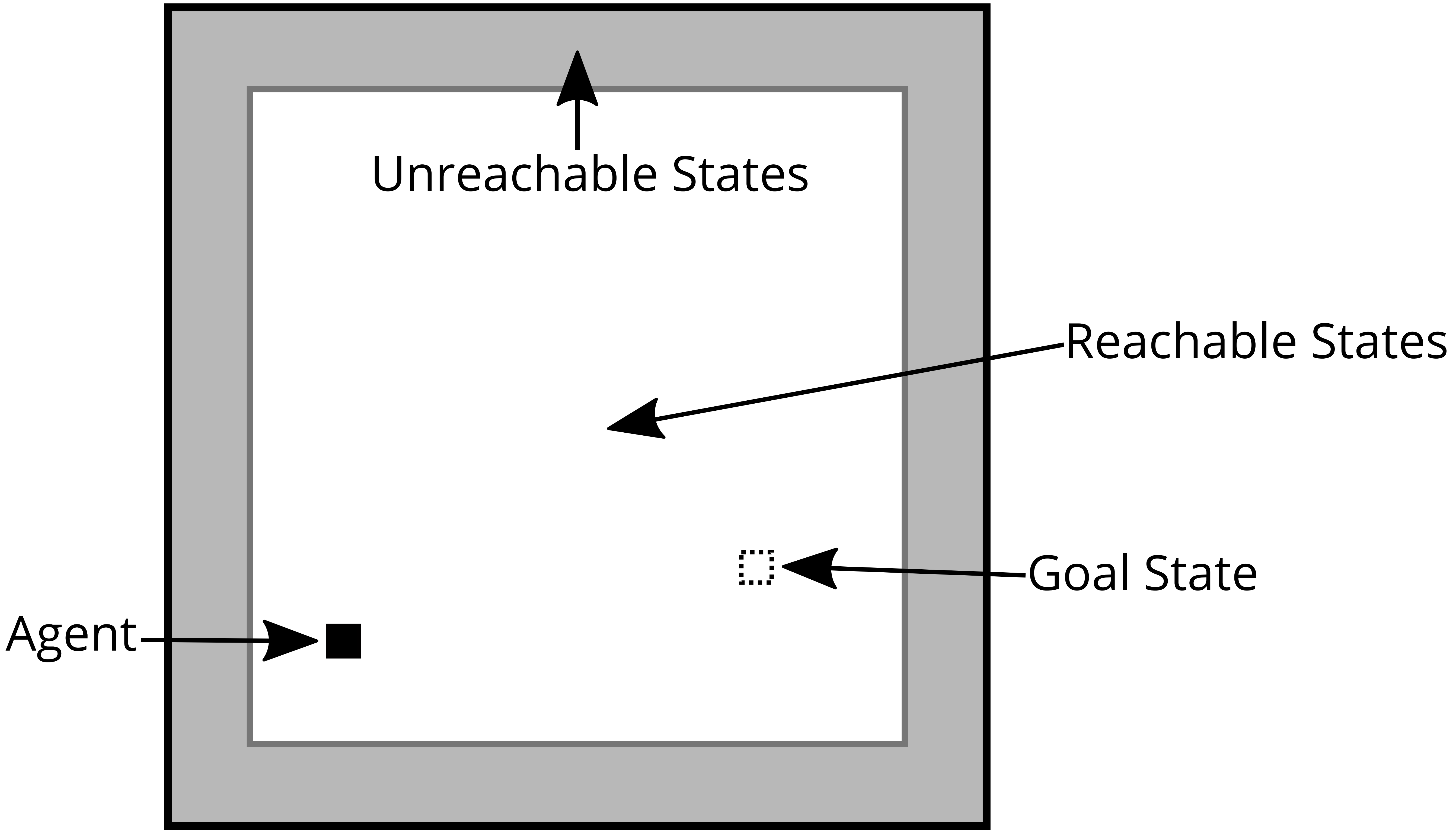
So, are we done?

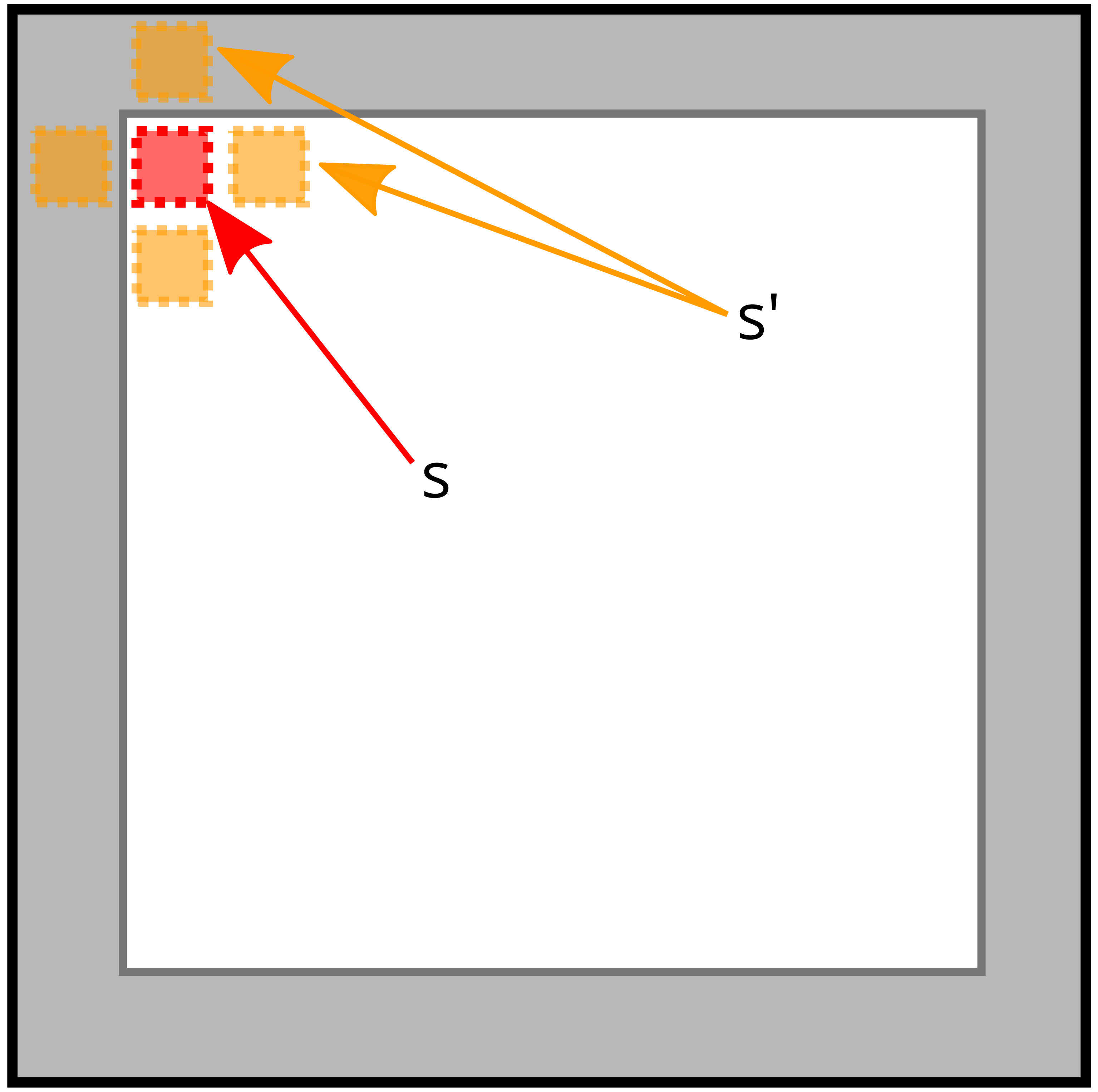
- Can we just learn an accurate model with a deep NN and use Dyna?
- **Two key take-aways:**
- How we use the model (search-control) can have a huge impact on how useful it is (can have very little impact, just a waste of computation)

So, are we done?

- Can we just learn an accurate model with a deep NN and use Dyna?
- **Two key take-aways:**
- How we use the model (search-control) can have a huge impact on how useful it is (can have very little impact, just a waste of computation)
- Small errors in the model can result in big errors in the policy

Examples of Bad Errors





- Imagine we initialize optimistically
- Imagine we do search-control from observed states

*** credit to Taher Jafferjee**

High-level Outline

- Part 1: Learning the optimal policy given the model (offline)
- Part 2: Moving to learned models (online)
- **Part 3: A brief discussion about other ways to use models**

Models are useful. They have been used in a variety of ways in RL.

- **Most related**: Learn a model and then use dynamic programming on this learned model to obtain approximate values
  - e.g., KBRL, KBSF, Compressed CME, Pseudo-MDPs
- Decision-time Planning
  - Model Predictive Control (see work from Byron Boots), MCTS
- Use model to improve exploration
- Use model to obtain better estimates of policy gradients (PILCO)
- Use model as inductive bias on value function (e.g., Predictron)

KBRL, KBSF, and CCME

- Learn values only for a representative set of points
- Define smaller (pseudo)-MDP only on these states
- Use value iteration (dynamic programming) on this smaller MDP, which is reasonably efficient
- Value function for whole state-space a simple weighting of the values for these representative states

Exploration with models

- Huge research area using learned models for sound exploration
  - Often consider an optimistic model in the set/distribution of models
  - Most algorithms though are very computationally expensive
- Reward bonuses: accuracy of learned models to incentivize exploration
  - Only indirectly using model, no planning

Implicit Planning

- Optimize model and planner based on the reward the agent receives, using end-to-end learning
  - Contrasts learning the model using a separate objective and updating using explicit planning steps
- Can be seen as an inductive bias on value function architecture
- Examples:
  - Predictron (DeepMind)
  - TreeQN and ATreeC (Whiteson and others)

High-level Outline

- Part 1: Learning the optimal policy given the model (offline)
- Part 2: Moving to learned models (online)
  - Dyna for background planning
  - Search-control and Model choices
- Part 3: A brief discussion about other ways to use models

High-level Outline

- Part 1: Learning the optimal policy given the model (offline)
- Part 2: Moving to learned models (online)
  - Dyna for background planning
  - Search-control and Model choices
- Part 3: A brief discussion about other ways to use models

Questions?